マニュアル

産業用組込み PC EC1A IoT シリーズ用

Linux ディストリビューション

『Algonomix6』について

目　次

はじめに

第１章 概要

第２章 システム構成

第３章 開発環境

第４章 産業用組込みPC EC1A IoT シリーズについて

第５章 システムリカバリ

付録

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、産業用組込み PC EC1A-010AT IoT シリーズ（以降 **1A シリーズ**）用 Linux ディストリビューション（以降 **Algonomix6**）に特化した部分について説明します。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。

２）　保証について

Algonomix6 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。Algonomix6 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。

第１章　概要

本章では、Algonomix6 の具体的な内容を説明する前に、Algonomix6 の概要について説明します。

１－１　Algonomix6 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、・・・・）
- procps（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

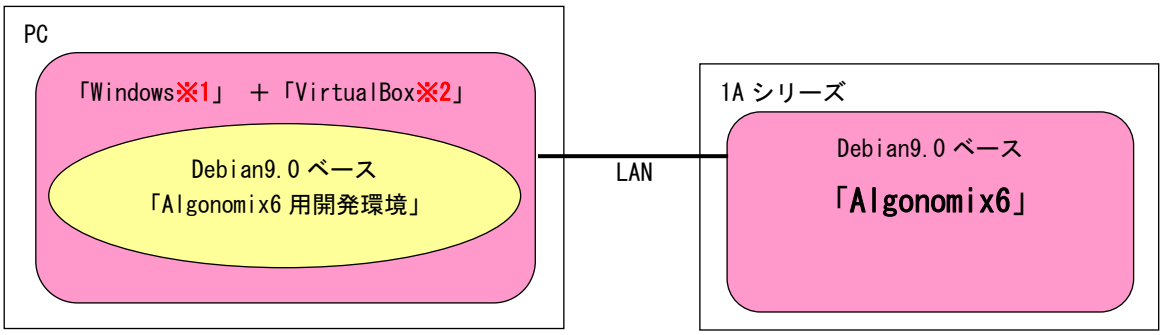
Linux カーネルといくつかの必要なソフトウェアパッケージをまとめて、OS として使えるようにしたものを Linux ディストリビューションといいます。

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。

Algonomix6 は、「Debian 9.0」という Linux ディストリビューションをベースにした、1A シリーズ用の Linux ディストリビューションです。Algonomix6 は、「Debian 9.0」に 1A 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 Algonomix6 用の開発環境をインストールすることで、Algonomix6 用のソフトウェアを開発することができます。

Algonomix6 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. Algonomix6 の開発環境

開発環境の詳細については、別紙『Algonomix6_開発環境ユーザーズマニュアル』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。

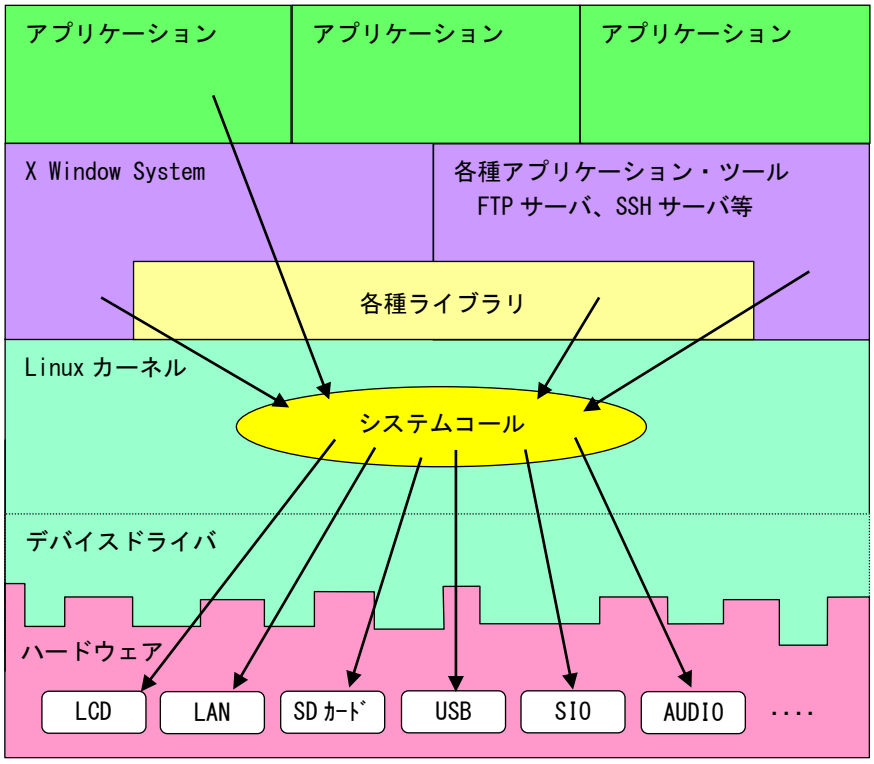


図 1-2-1. Linux ソフトウェア構成図

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　Algonomix6 パッケージについて

Algonomix6 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 linaro 製 DragonBaord410c 用 Debian 9.0 イメージ（Ver17.09）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	バージョン
4.9.56-linaro-lt-qcom	Linux Kernel 本体	4.9.56
LXQt	デスクトップ環境	0.11.1-1
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	1:7.4p1-10+deb9u1
gdbserver	GNU デバッガ	7.12-6
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	0.7.5+git20140201-1+b2
monodevelop	Development Environment for GNOME	5.10.0.871-2
synaptic	Synaptic パッケージマネージャ	0.84.2

Algonomix6 では、linaro 製 DragonBaord410c 用 Debian 9.0 イメージ（Ver17.09）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。
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。

Algonomix6 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 キーボードで「Ctrl＋ALT+T」を押すか、右下の「スタートボタン」→「システムツール」→「Xfce Terminal」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。

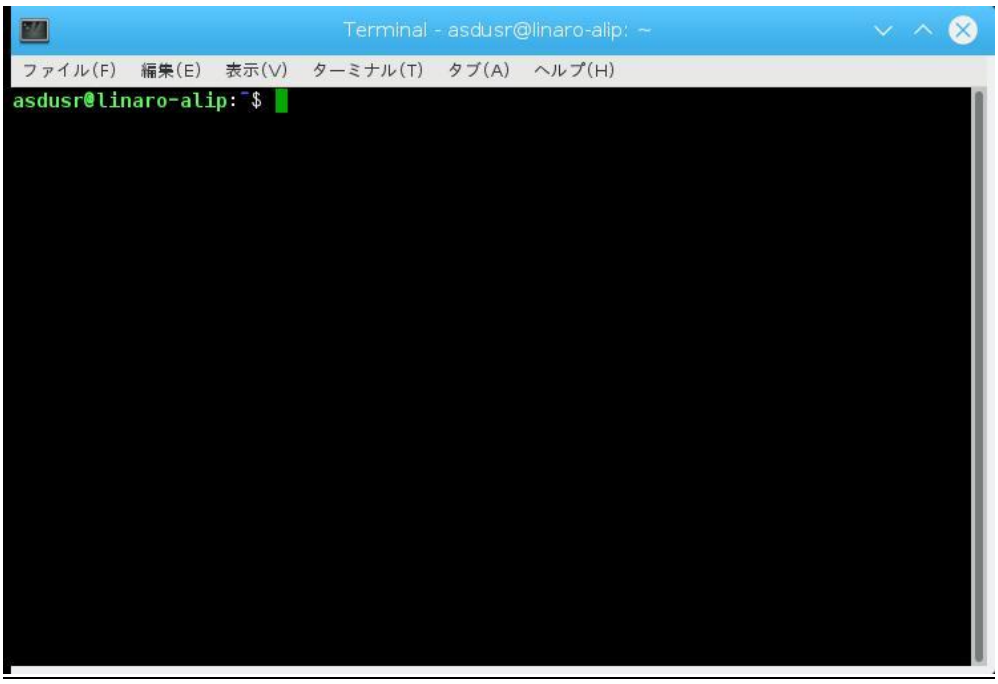


図 2-1-1. ターミナル画面

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

Algonomix6 では、linaro 製 DragonBaord410c 用 Debian 9.0 イメージ（Ver17.09）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。

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、あらかじめご了承頂きますようお願い申し上げます。

Algonomix6 では、インターネットへアクセスするリポジトリを予め設定しています。

リポジトリの制限や追加は、/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。

①　[スタートボタン]→[設定]→[Synaptic パッケージマネージャ]をクリックします。

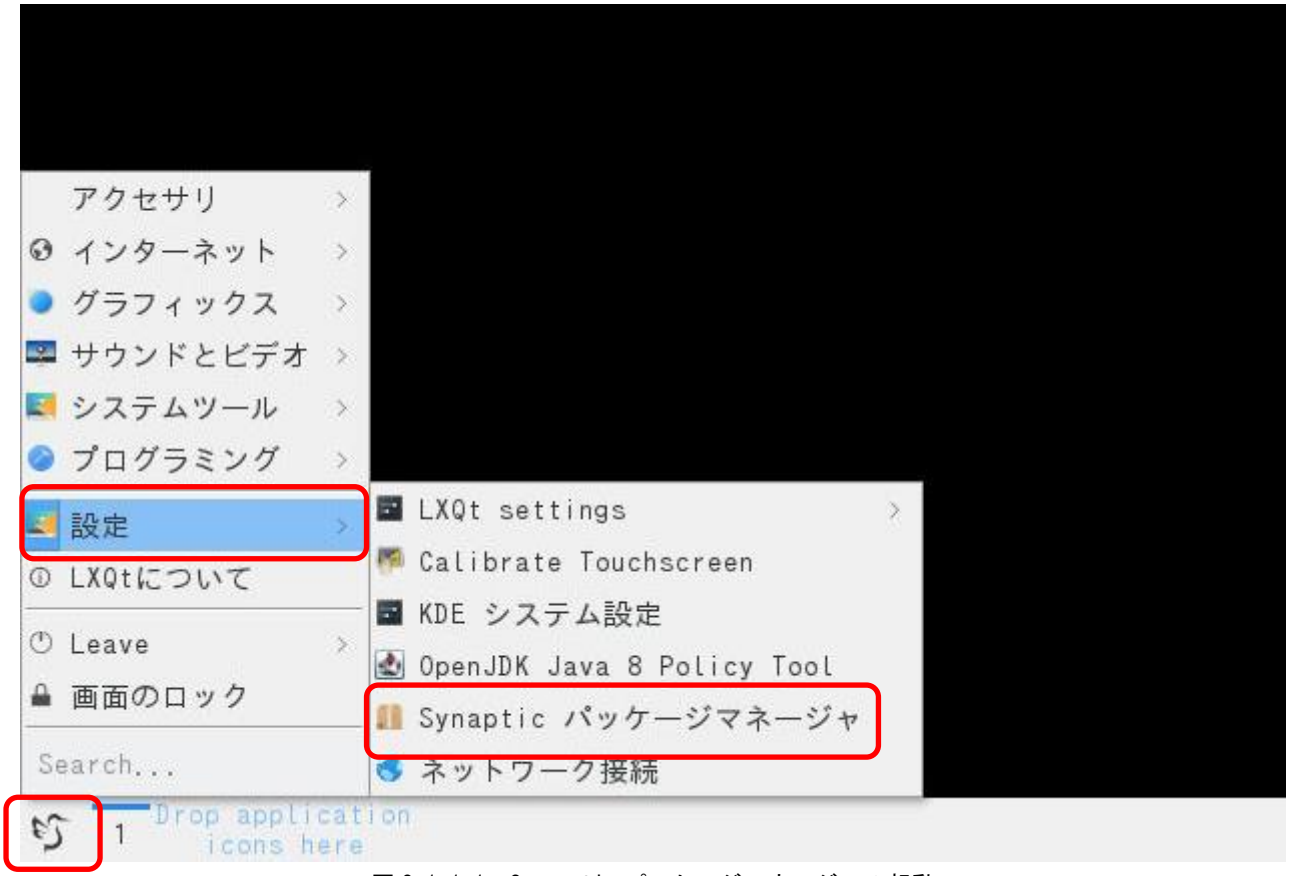


図 2-1-1-1. Sysnaptic パッケージマネージャの起動

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、認証ウインドウが表示されるので、パスワードを入力して「OK」をクリックしてください。

asdusr パスワード　　：asdusr

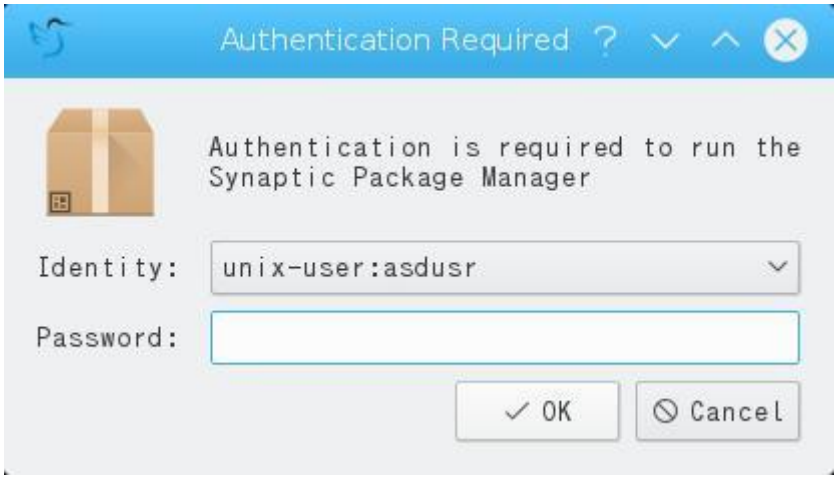


図 2-1-1-2. 管理者権限認証画面

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。

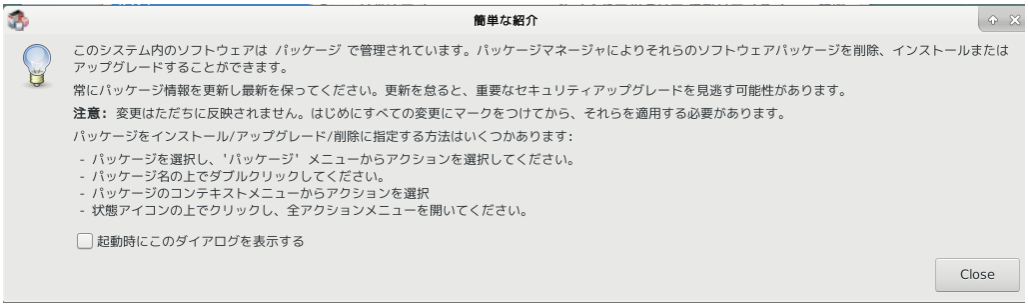


図 2-1-1-3. 紹介画面

④　Synaptic メイン画面を図 2-1-1-4 に示します。

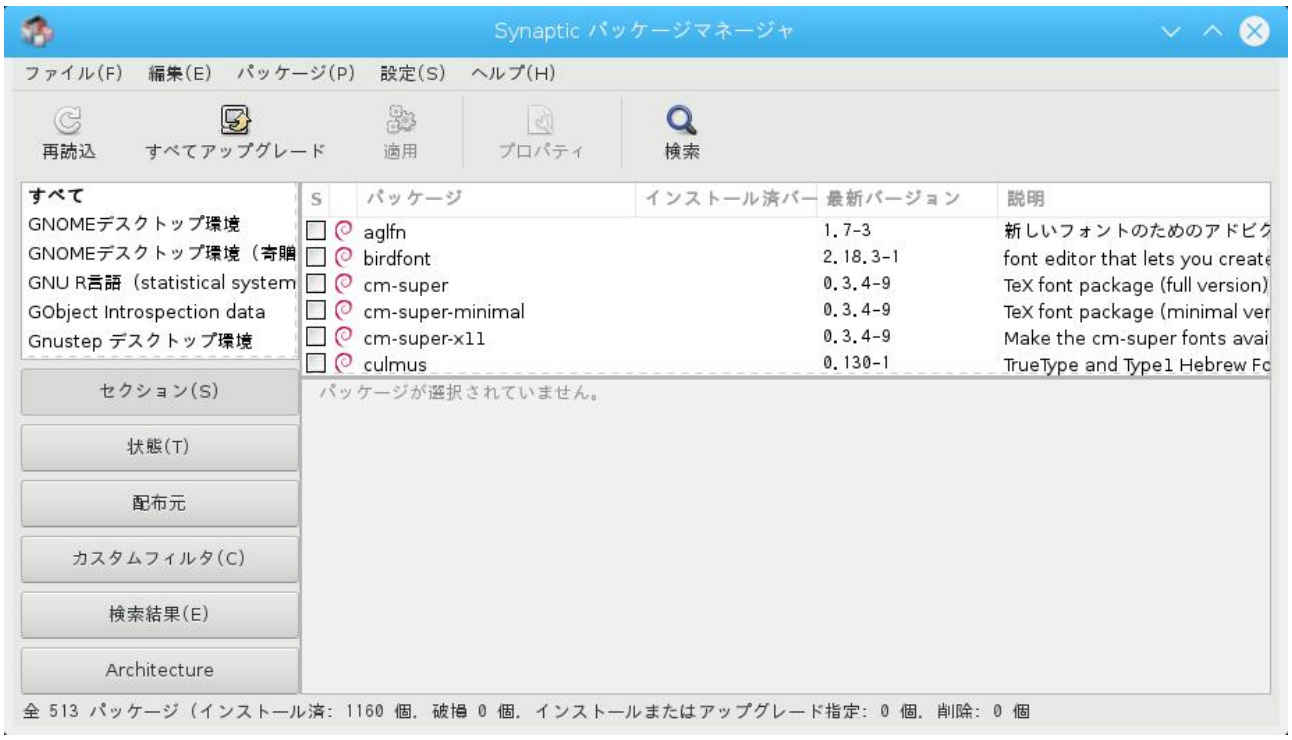


図 2-1-1-4. Synaptic パッケージマネージャメイン画面

⑤　「設定」→「リポジトリ」をクリックしてください。

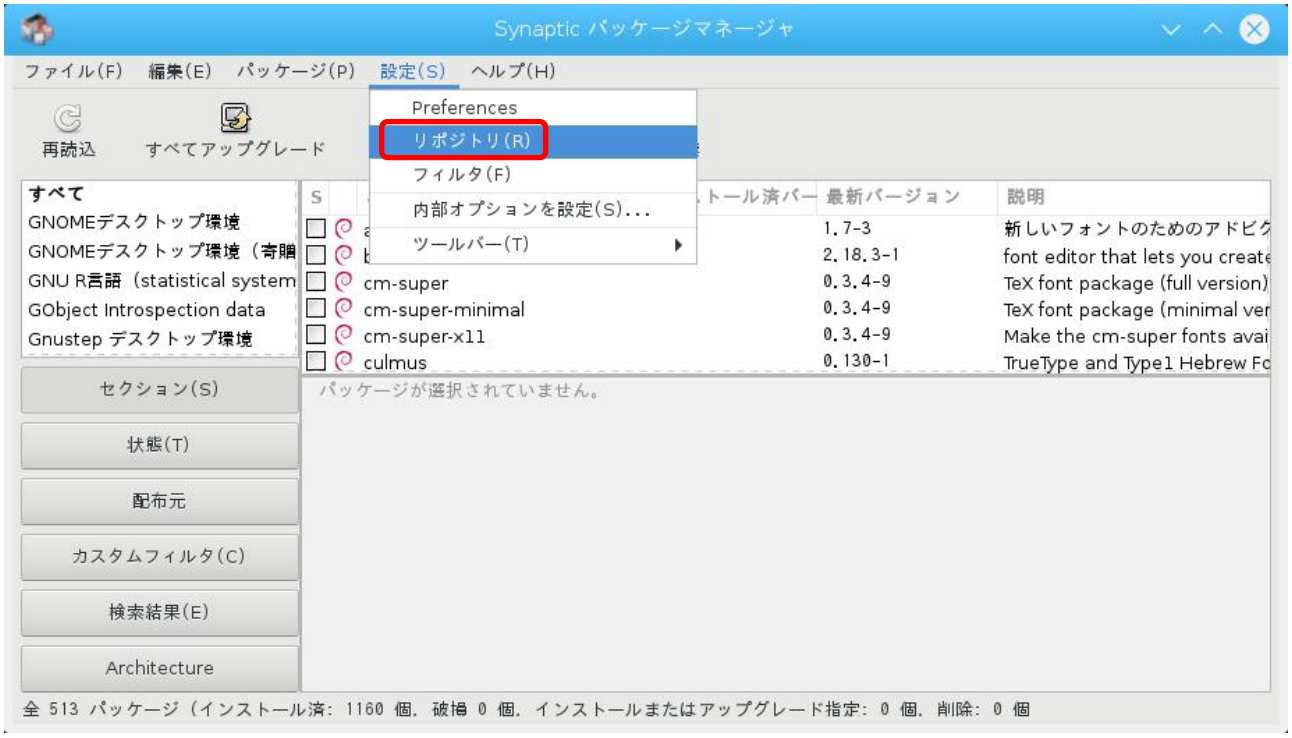


図 2-1-1-5. 設定メニュー

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。

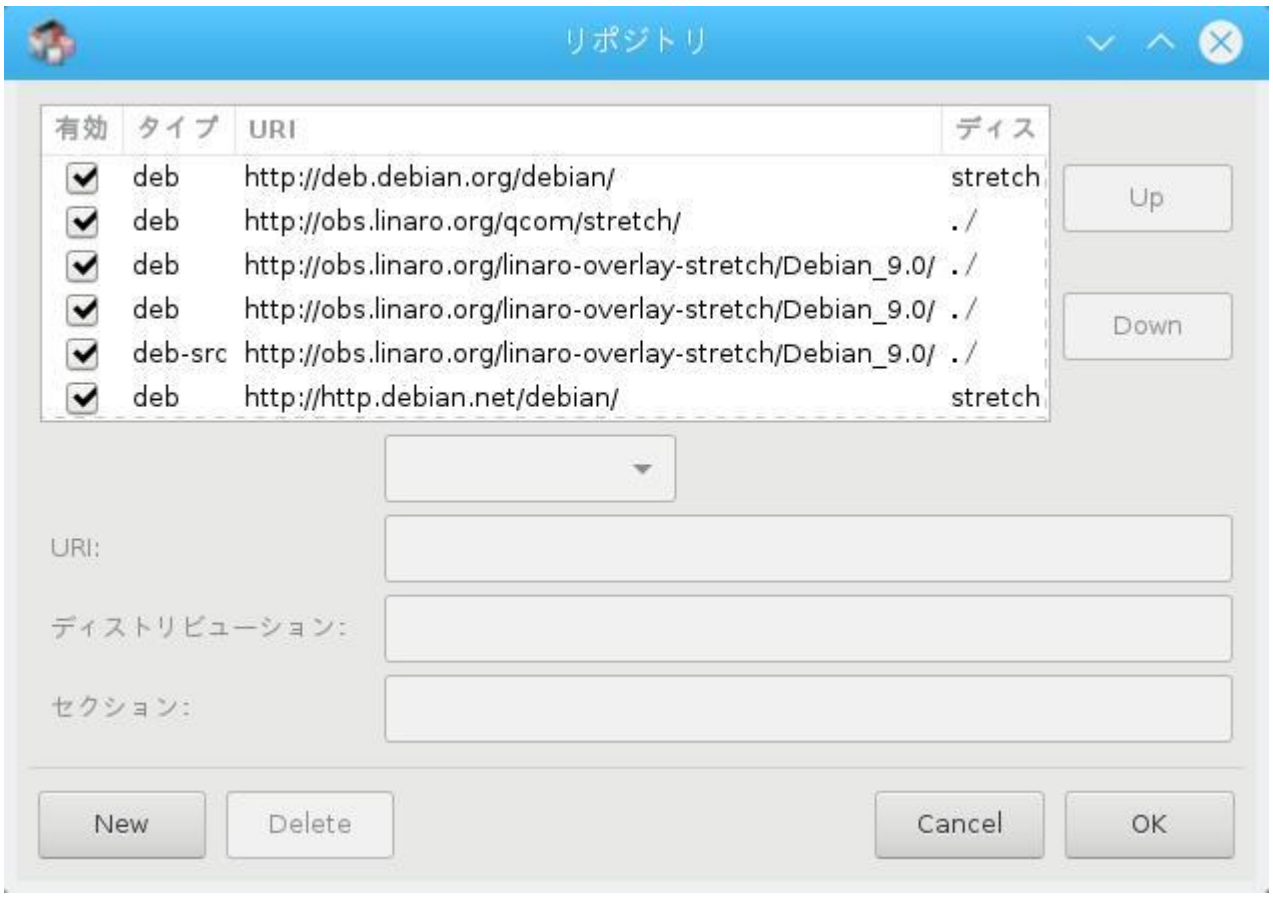


図 2-1-1-6.　リポジトリ画面

デフォルト設定では、リポジトリは 11 種類登録済みです。
「New」ボタンをクリックして、追加することも可能です。

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。

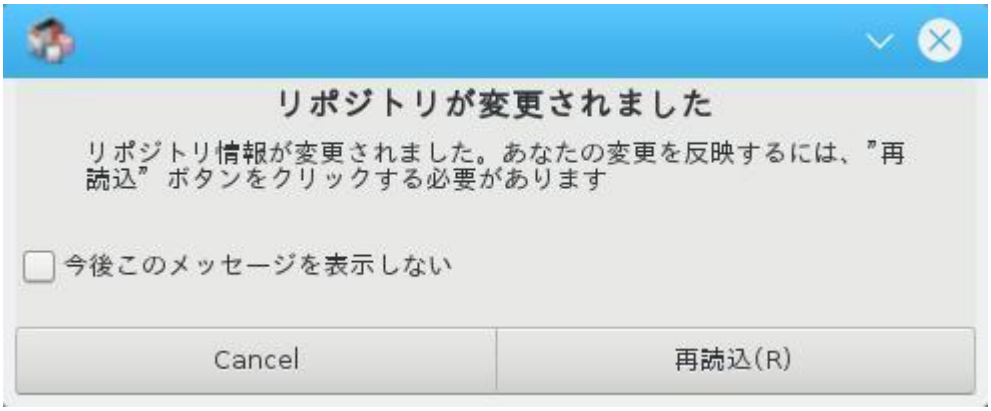


図 2-1-1-7.　リポジトリ変更確認画面

「再読込」をクリックして、変更を反映させてください。

⑧　再読込されると、図 2-1-1-8 のような、警告画面が表示されます。同じリポジトリが定義されているため、警告メッセージが表示されます。特に問題はないので「閉じる」ボタンを押して、閉じてください。

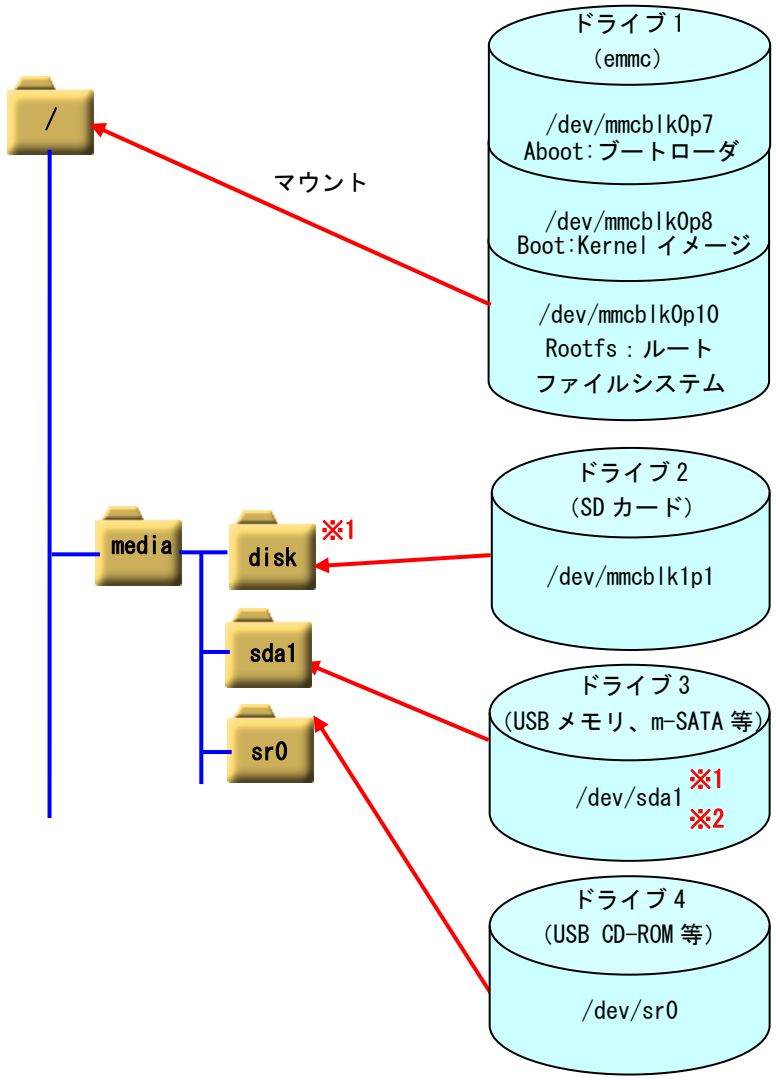
図 2-1-1-8. 警告画面

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、「apt-get」コマンドを使用してください。

２－２　Algonomix6 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、EMMC 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。Algonomix6 はベースイメージが linaro 製 DragonBaord410c 用のイメージをベースにしているため、そのパーティション構成にしたがっています。



※注１：ストレージ機器をマウントするたびに sdb1、sdb1、sdc1 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。

※注２：USB 機器は、認識順により sdb と sdc 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

1A シリーズでは、EMMC SSD 16GByte 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。

1A 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. 1A シリーズの初期パーティション構成(16GByte)

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	名称
/dev/mmcblk0p1	2Kbyte	-	-	-	-	CDT
/dev/mmcblk0p2	512Kbyte	-	-	-	-	SDL1
/dev/mmcblk0p3	512Kbyte	-	-	-	-	RPM
/dev/mmcblk0p4	1Mbyte	-	-	-	-	TZ
/dev/mmcblk0p5	512Kbyte	-	-	-	-	HYP
/dev/mmcblk0p6	16Kbyte	-	-	-	-	SEC
/dev/mmcblk0p7	1Mbyte	-	-	-	-	ABOOT：ブートローダ
/dev/mmcblk0p8	64MByte	-	-	-	-	BOOT：Kernel イメージ
/dev/mmcblk0p9	1Mbyte	-	-	-	-	DEVINFO
/dev/mmcblk0p10	14.4Gbyte	2.9Gbyte	11.2Gbyte	21%	/	ROOTFS：ルートファイルシステム

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
+--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プ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-SATA SSD のデバイスファイルとなりますので、新たに追加した USB メモリ等は 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。Algonomix6 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　Algonomix6 設定ツール「ASD Config」について

本項では、1A 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 3 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．初期化スイッチを押しながら電源を投入したとき、OS 起動後に自動で起動します。
３．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 のような画面が現れます。この画面から、各種設定ツールを起動します。

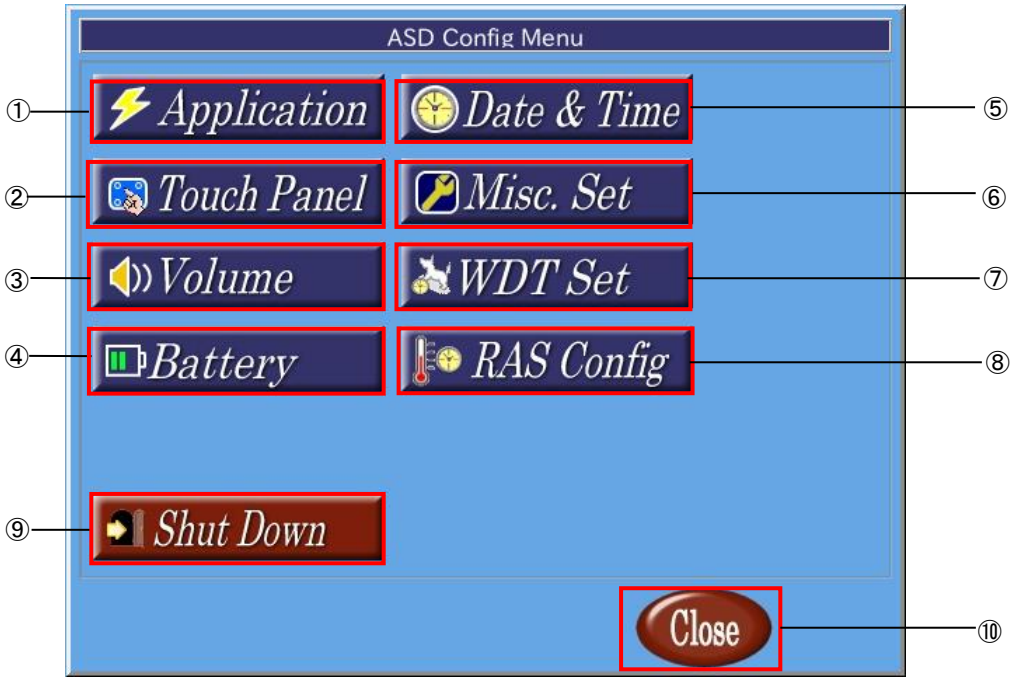


図 2-3-1. ASD Config Menu

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 Touch Panel Config
ASD Touch Config は、タッチパネルのキャリブレーションを行うためのツールです。
詳細は『2-3-2 ASD Touch Panel Config について』で説明します。

③ ASD Volume Config
ASD Volume Config は、音声の設定を行うためのツールです。
詳細は『2-3-3 ASD Volume Config について』で説明します。

④　ASD UPS 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 UPS Config について』で説明します。

⑤　ASD Date Config
ASD Date Config は、時計を設定するためのツールです。
詳細は、『2-3-5 ASD Date Config について』で説明します。

⑥　ASD Misc Setting
ASD Misc Setting は、画面の輝度調節や、本製品の製品情報を確認するためのツールです。
詳細は『2-3-6 ASD Misc Setting について』で説明します。

⑦　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『2-3-7 ASD WatchdogTimer Config について』で説明します。

⑧　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『2-3-8 ASD Ras Config について』で説明します。

⑨　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『2-3-9 ASD Shutdown Menu について』を参照してください。

⑩　ASD Config Menu の終了
ASD Config Menu を終了します。

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、メニュー画面から[Application]を選択します。

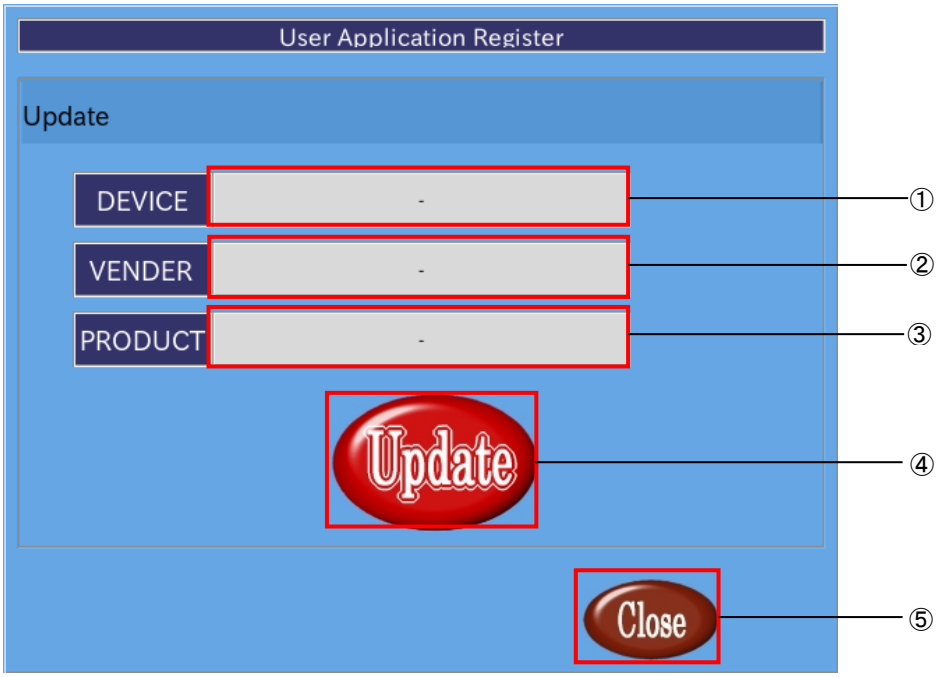


図 2-3-1-1.　アップデート画面

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ければ、「-」が表示されます。

② 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ければ、[-]が表示されます。

③ 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ければ、[-]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 Touch Panel Config について

ASD Touch Panel Config はタッチパネルのキャリブレーションを行うためのツールです。
ASD Touch Panel Config を起動するには、メニュー画面から[Touch Panel]を選択します。

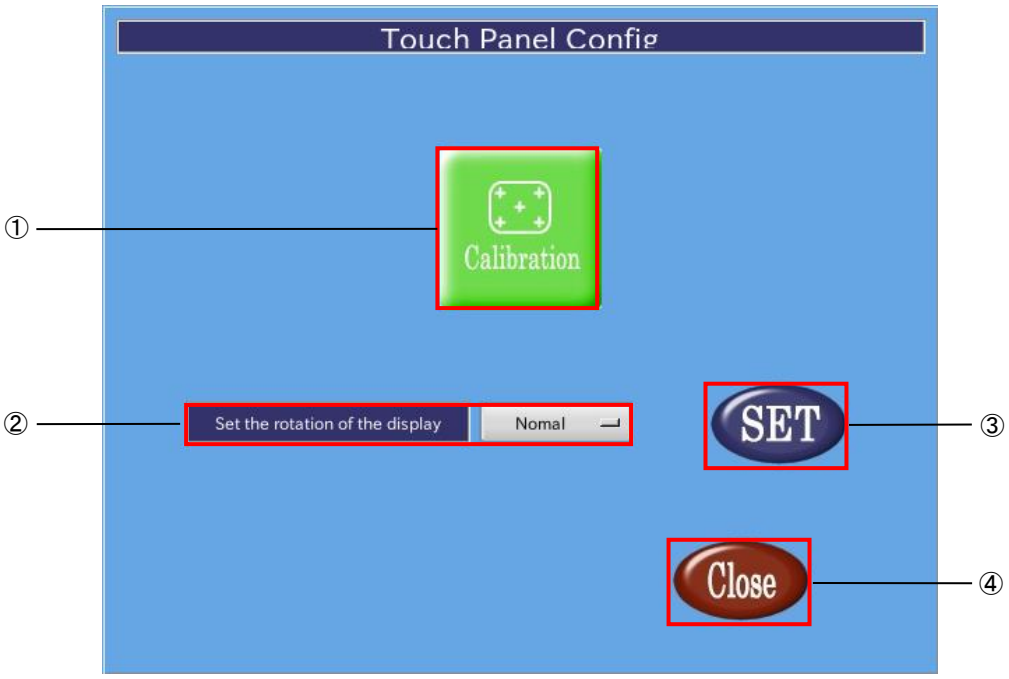


図 2-3-2-1. メニュー画面

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。
③ 設定反映
　②で行った設定を反映し、保存します。
④ 終了
　タッチパネルキャリブレーションを終了します。

●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、図 2-3-2-2 のように、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 2-3-2-1 のように対応しています。
任意の向きを選択して③の[SET]ボタンを押すことで、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。

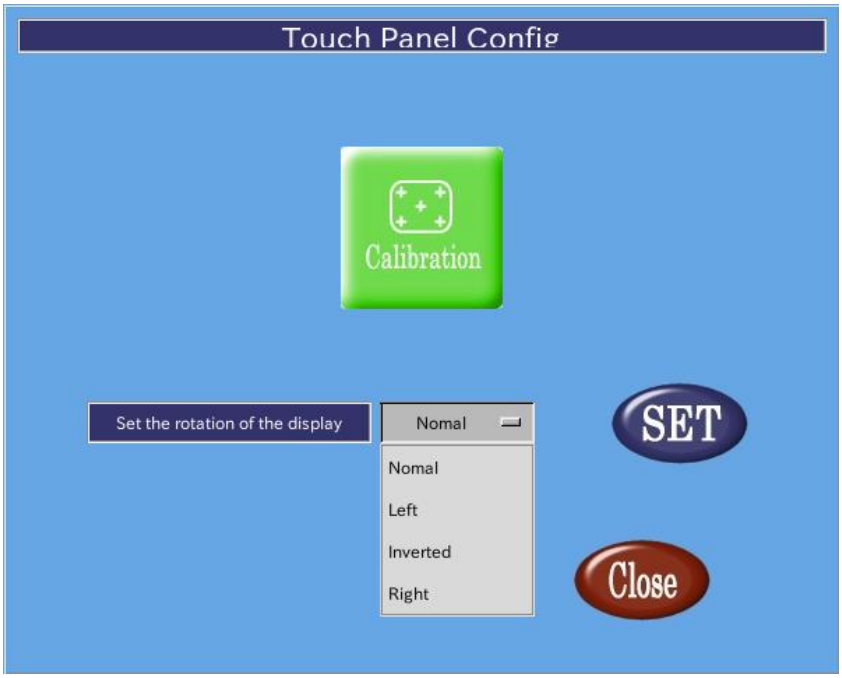


図 2-3-2-2. ディスプレイとタッチパネルの回転

表 2-3-2-1. 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。

２－３－３　ASD Volume Configについて

ASD Volume Configは音声ボリュームを設定する為のツールです。

ASD Volume Configは、メニュー画面から「Volume」を選択することで起動します。

●音声ボリュームの設定

[ASD Volume Config]の[Volume]タブを選択することで、各種音声デバイスのボリュームの調整、ミュートの設定を行うことができます。

ASD Volume Configで設定できる音声デバイスを表 2-3-3-1 に示します。

表 2-3-3-1. ASD Volume Configで設定できる音声デバイス

名称	内容
Master-L	マスターボリューム Left を設定します
Master-R	マスターボリューム Right を設定します

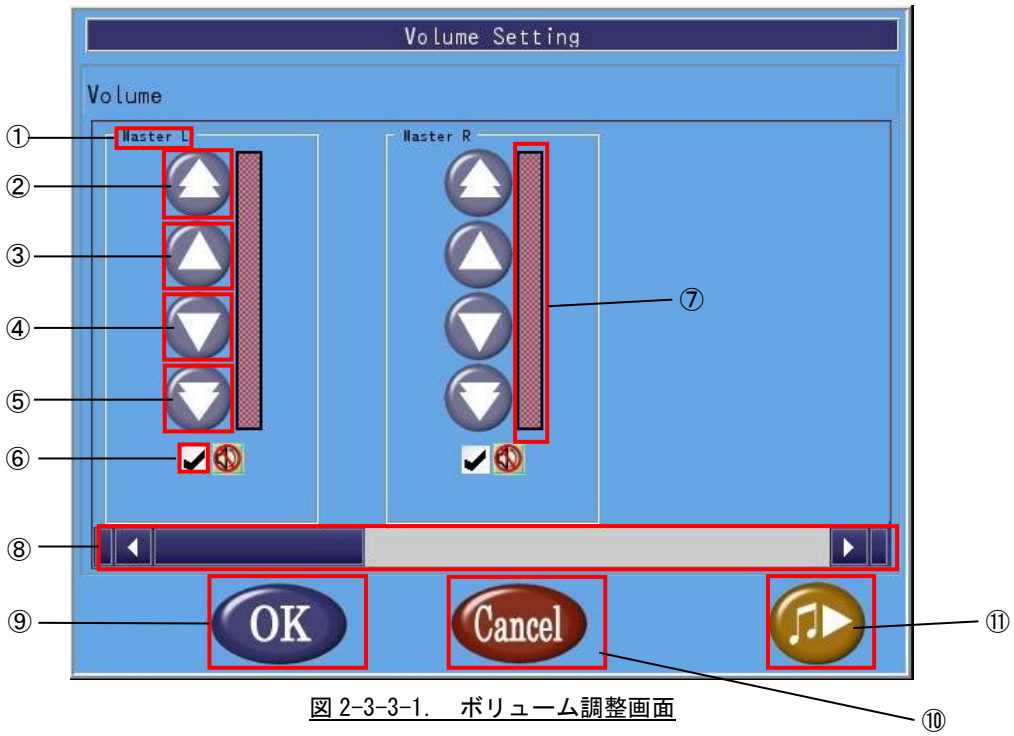


図 2-3-3-1.　ボリューム調整画面

① 音声デバイス名
音声デバイス名を表示します。

② 音量調整（大）
音量を大きく上げます。

③ 音量調整（小）
音量を上げます。

④ 音量調整（小）
音量を下げます。

⑤ 音量調整（大）
音量を大きく下げます。

⑥ ミュート調整
チェックボックスにチェックを入れることでミュート状態になります。チェックボックスを外せばミュートが解除されデバイスが有効になります。

⑦ 音量
現在の音量を表示します。

⑧ スクロールバー
スクロールバーを移動させることで他の音声デバイスを表示します。

⑨ 設定を保存して終了
「OK」ボタンを押下することで、設定を保存して終了します。

⑩ 設定を保存せずに終了
「Cancel」ボタンを押下することで、設定を破棄して終了します。

⑪ サンプル音声
サンプル音声を出力します。

２－３－４　ASD UPS Config について

ASD UPS Config は、UPS 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 UPS Config を起動するには、メニュー画面から[Battery]を選択します。

●バッテリ状態の表示

ASD UPS Config の[Status]タブを選択することで、UPS のバッテリ状態を取得できます。

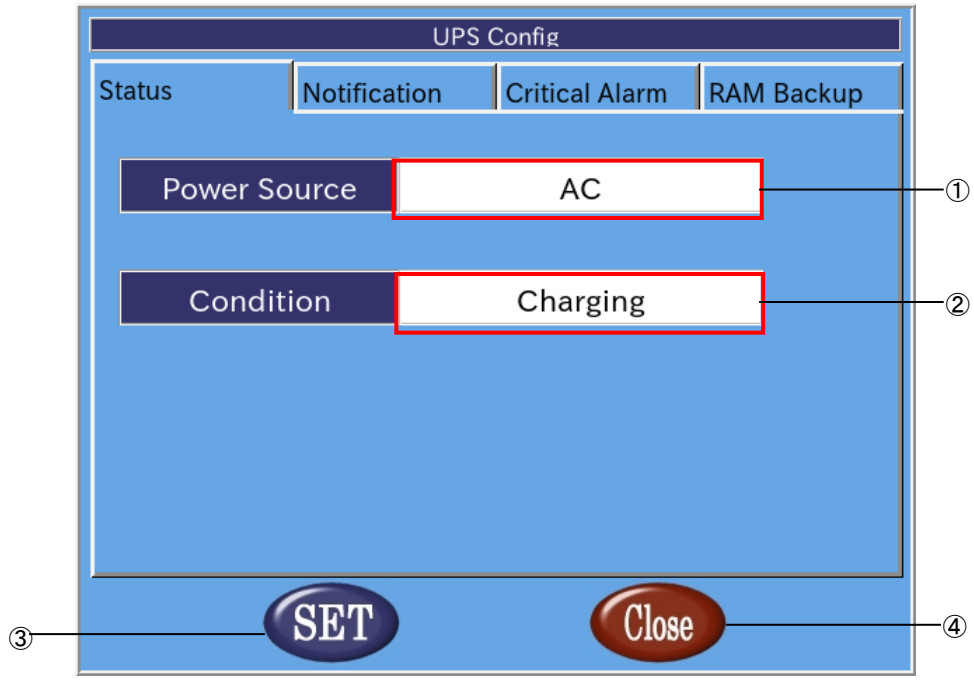


図 2-3-4-1. バッテリ状態の表示

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	バッテリ電源で動作中です。

②　状態

バッテリの状態を表示します。

表 2-3-4-2. バッテリの状態

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

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 UPS 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。

●バッテリ保護機能動作通知設定

ASD UPS Config の[Notification]タブを選択することで、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。

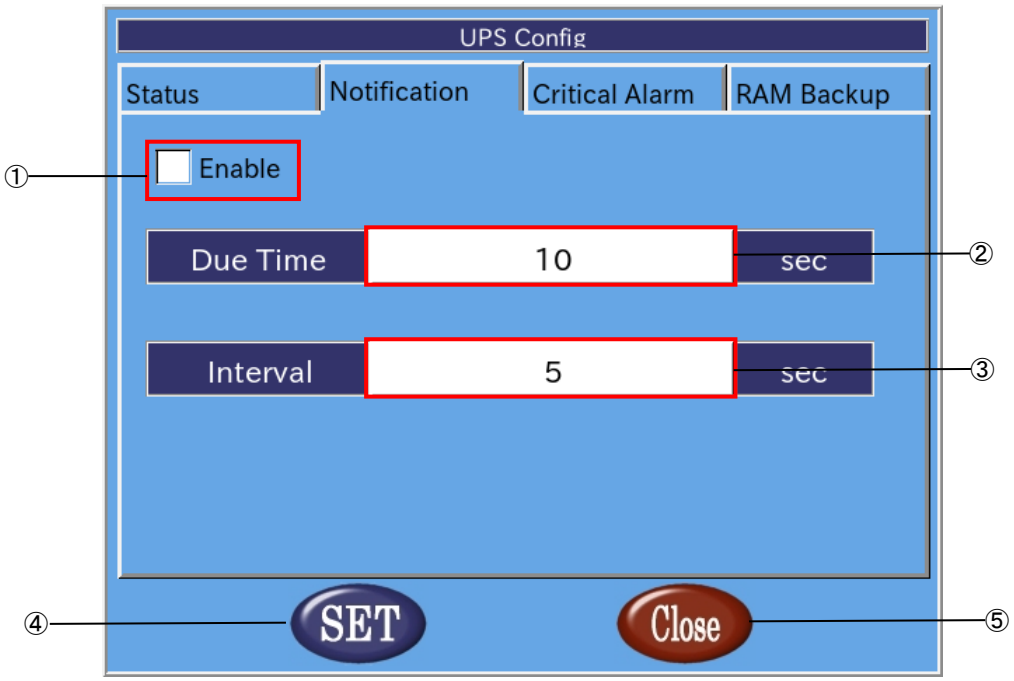


図 2-3-4-2. バッテリ保護機能動作通知設定

① 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、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
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 UPS 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。

●バッテリ警告設定

ASD UPS Config の[Critical Alarm]タブを選択することで、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。

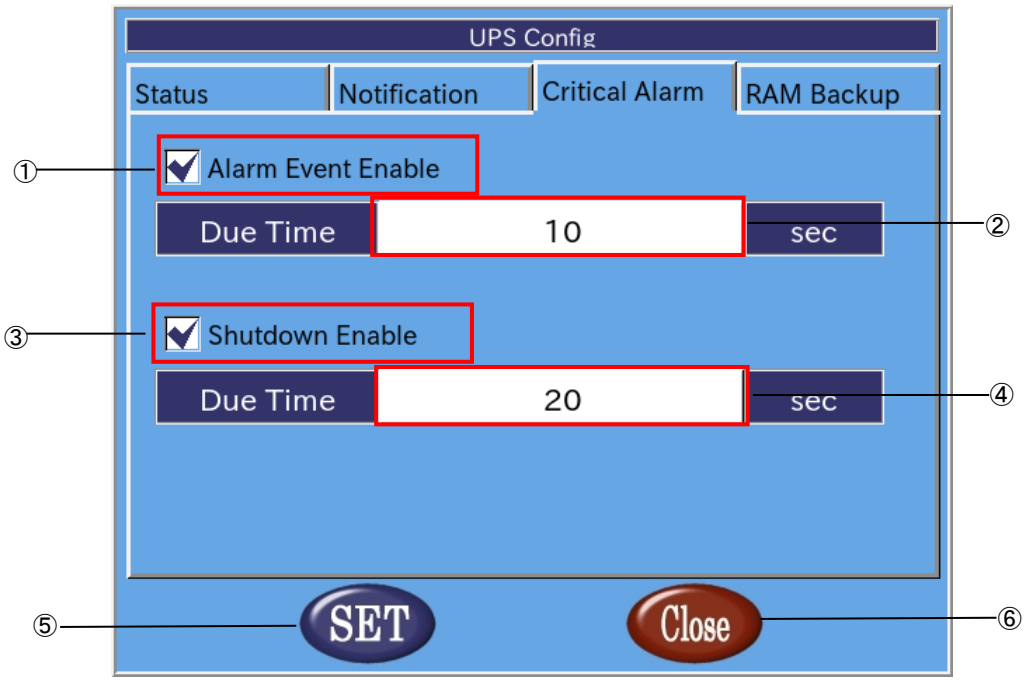


図 2-3-4-3. バッテリ警告設定

① 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、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
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、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
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 UPS 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

ASD UPS Config の[RAM Backup]タブを選択することで、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。

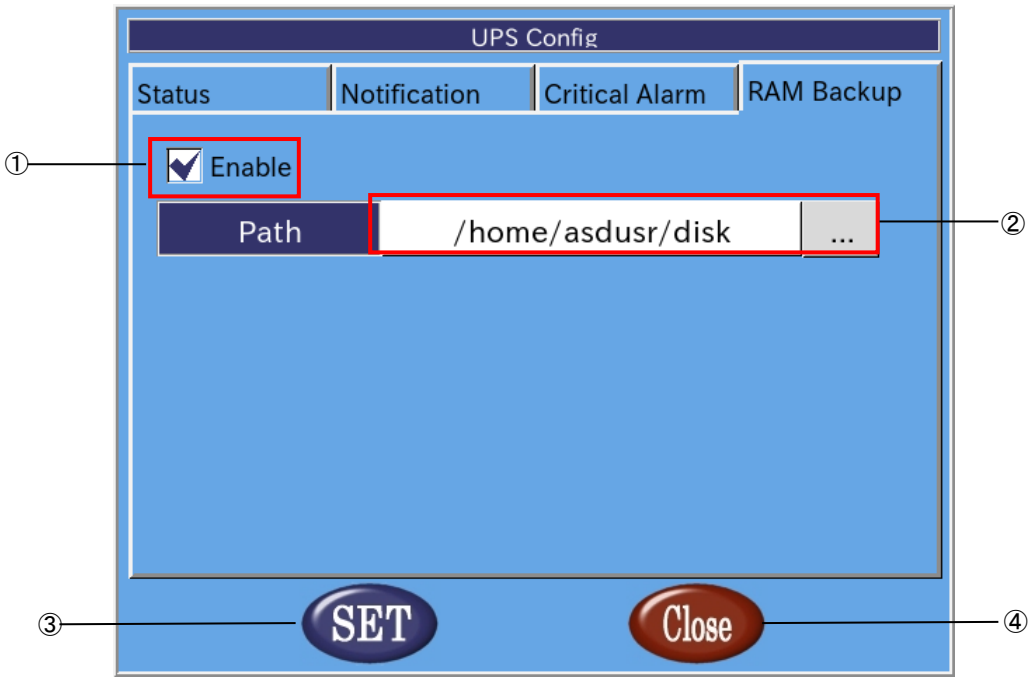


図 2-3-4-4. RAM バックアップ設定

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 UPS 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 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、メニュー画面から[Date & Time]を選択します。

●時計の手動設定

ASD Date Config の[Date & Time]タブを選択することで、手動で時計を設定できます。

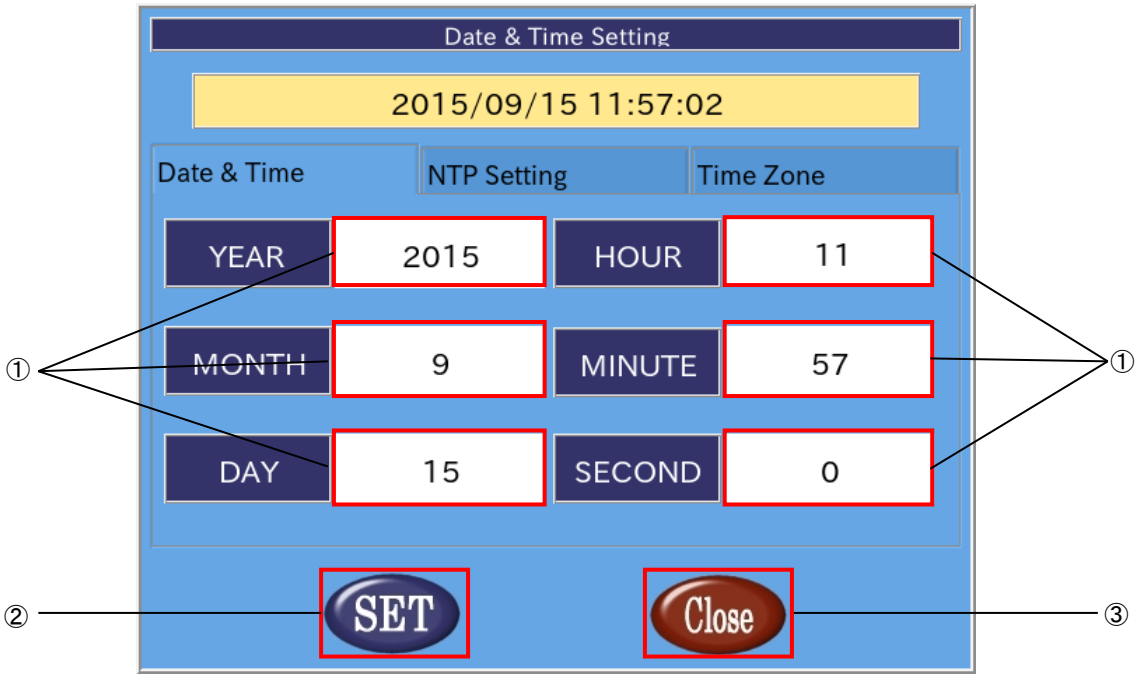


図 2-3-5-1. 時計の設定

① 日付、時刻を設定
白い部分を押下することで、入力画面があらわれ、日付、時間を設定できます。
年、月、日、時、分、秒単位で指定できます。
ntp を有効にした場合、これらの項目は変更できません。

② 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。

③ 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。

●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 1A シリーズの時計が NTP サーバの時計に同期されます。

ASD Date Config の[Network Time Protocol]タブを選択することで、NTP サーバを設定できます。

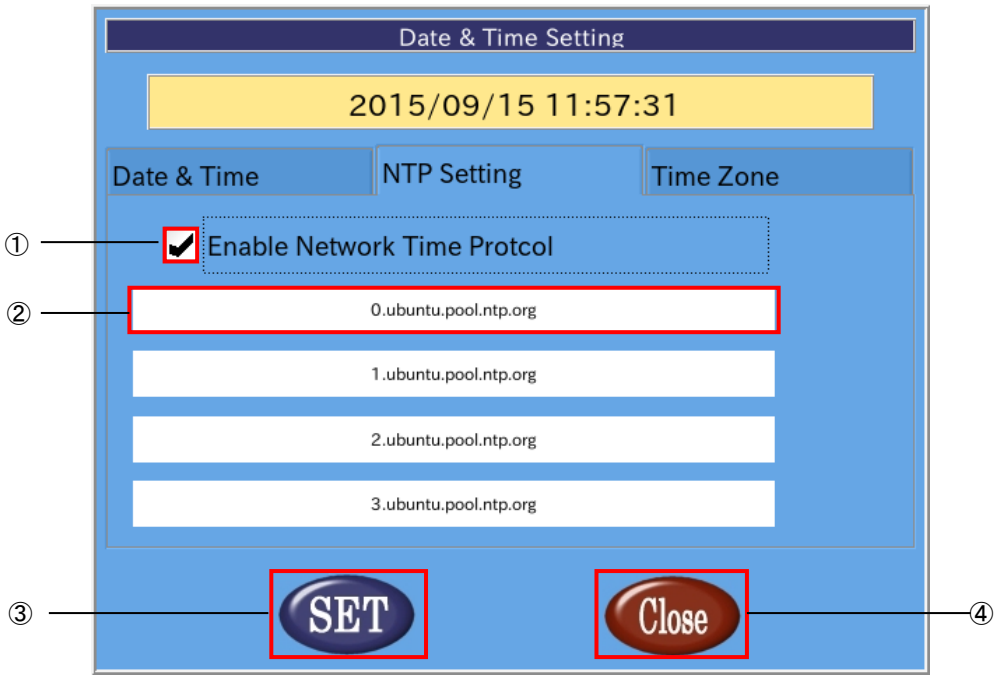


図 2-3-5-2. NTP サーバの設定

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：1A 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。

●タイムゾーンの設定

ASD Date Config の[Time Zone]タブを選択することで、タイムゾーンを設定できます。

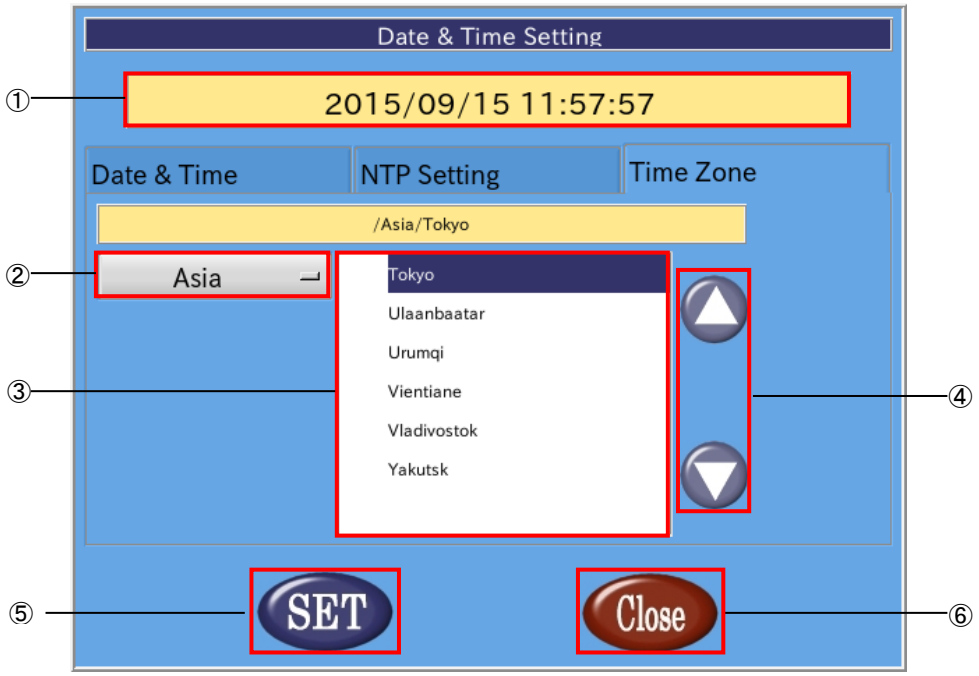


図 2-3-5-3. タイムゾーンの設定

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 地域の選択
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 タイムゾーンの都市リストの変更
都市のリストをスクロールします。

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 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。

２－３－６　ASD Misc Setting について

ASD Misc Setting は、バックライト、ブザーの設定や製品情報の読み出しを行うツールです。
ASD Misc Setting を起動するには、メニュー画面から「Misc. Set」を選択します。

●バックライトの調節と、タッチパネルのブザーの設定

ASD Misc Setting の[Backlight & Buzzer Setting]タブを選択することで、図 2-3-6-1 の画面になります。
ここでは、バックライトの調節とブザーの設定を行うことができます。

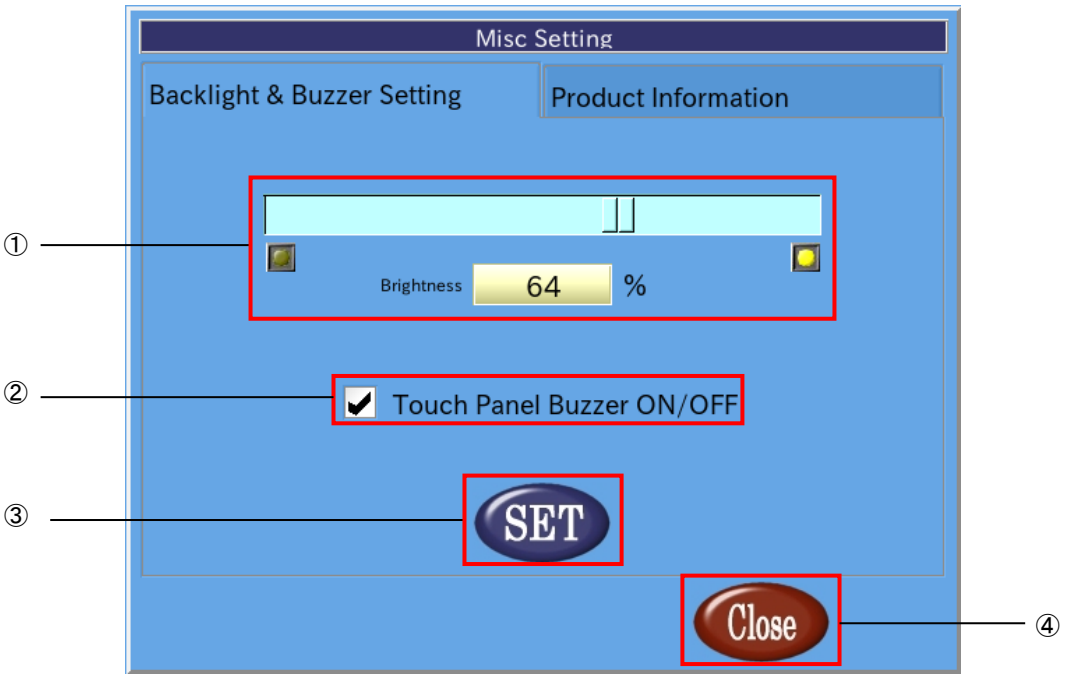


図 2-3-6-1. バックライトとブザーの設定

① バックライトの明るさの調整

スクロールバー上の  をドラッグすることで、バックライトの光量を調節できます。

② タッチパネルブザーの ON/OFF 設定

チェックボックスをチェックすることでタッチパネルをタッチする時にブザーを鳴らすことができます。チェックが外れていればタッチパネルをタッチしてもブザーは鳴りません。

③ 設定の保存

「SET」ボタンを押下することで、現在の設定を保存します。

④ 終了

「Close」ボタンを押下することで ASD Misc Setting 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。

●製品情報読み出し

ASD Misc Setting の[Product Information]タブを選択することで、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。

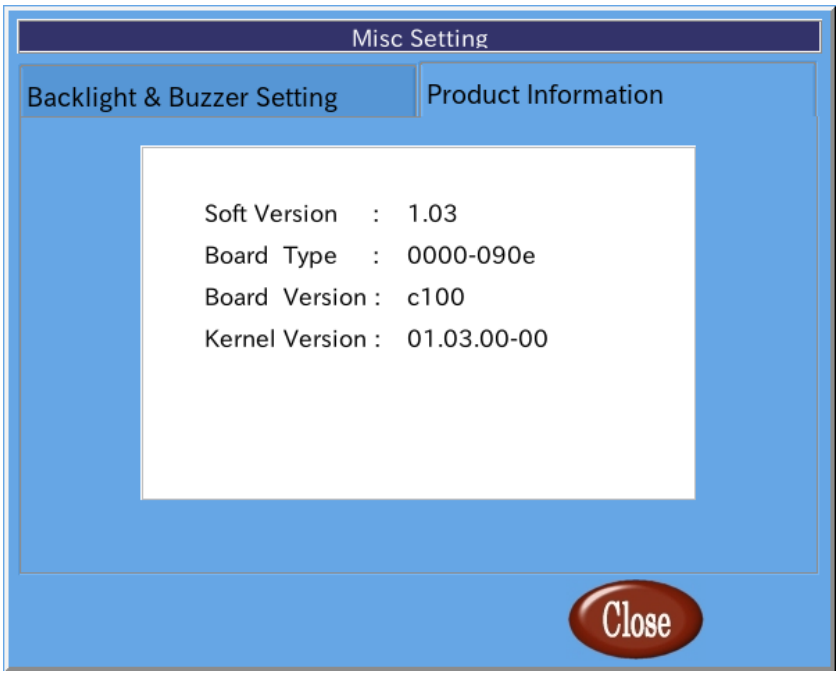


図 2-3-6-2. 製品情報

表 2-3-6-1. 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

２－３－７　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、メニュー画面から[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

ASD WatchdogTimer Config の[Hardware Watchdog Timer]タブを選択することで、ハードウェア・ウォッチドッグタイマを設定できます。

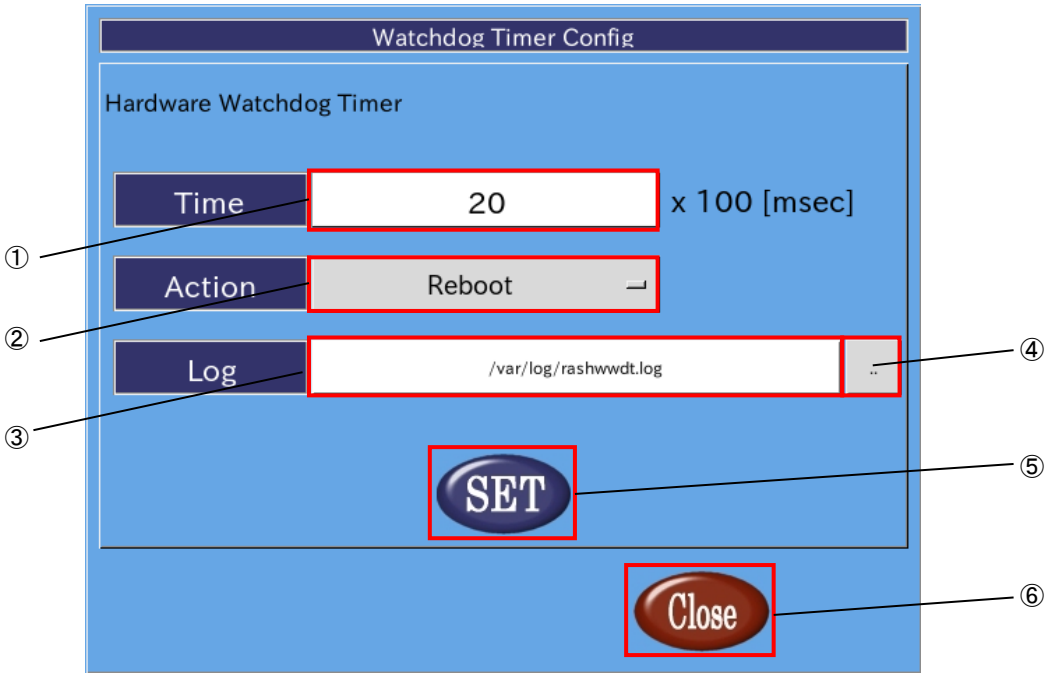


図 2-3-7-1. ハードウェア・ウォッチドッグタイマの設定

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 2-3-7-1 に示します。デフォルトでは Reboot が選択されます。

表 2-3-7-1. 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　ログファイルパスの設定
ハードウェア・ウォッチドッグタイマが起動時、タイムアウト時に出力するログファイルのパスを指定します。デフォルトでは、/var/log/rashwwdt.log にログを出力します。

④　ファイル選択
ファイル選択ダイアログを表示します。

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 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。

２－３－８　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、メニュー画面から[Ras Config]を選択します。

●CPU および周辺回路の温度の表示

ASD Ras Config の[Temperature]タブを選択することで、CPU 温度の表示が確認できます。

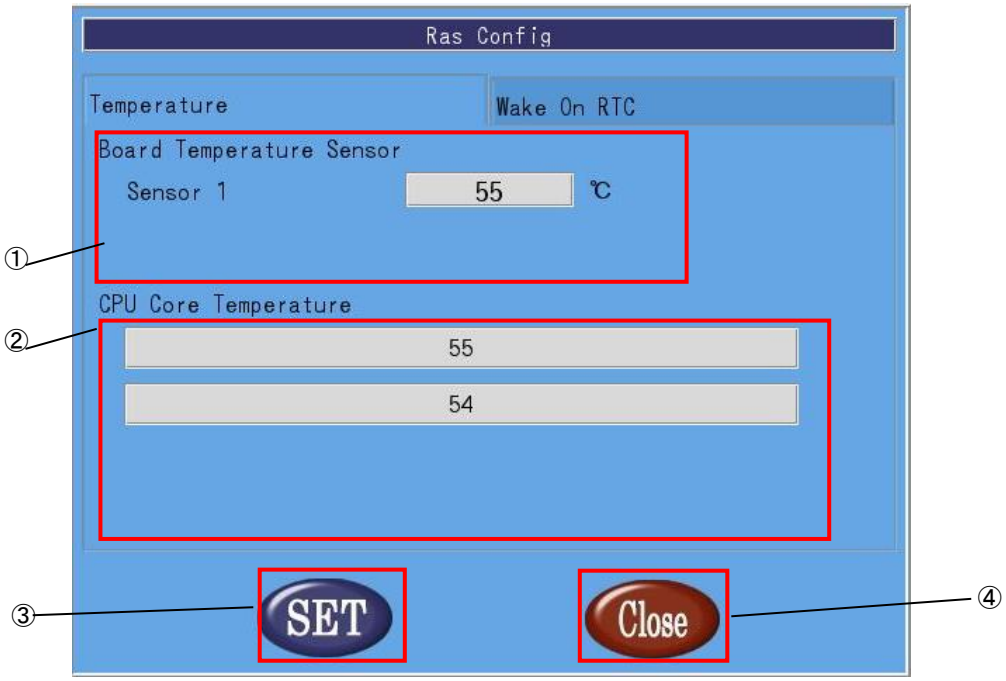


図 2-3-8-1. CPU 温度の表示

①　基板温度

基板内蔵の温度センサの測定値を表示します。

表 2-3-8-1. Board Temperature Sensor

名称	動作
Sensor 1	UPS バッテリ付近の温度センサの測定値を表示します。

②　CPU 温度

各 CPU の Core の温度を表示します。

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 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。

●Wake On RTC の設定

ASD Ras Config の[Wake On RTC]タブを選択することで、Wake On RTC 機能の設定ができます。

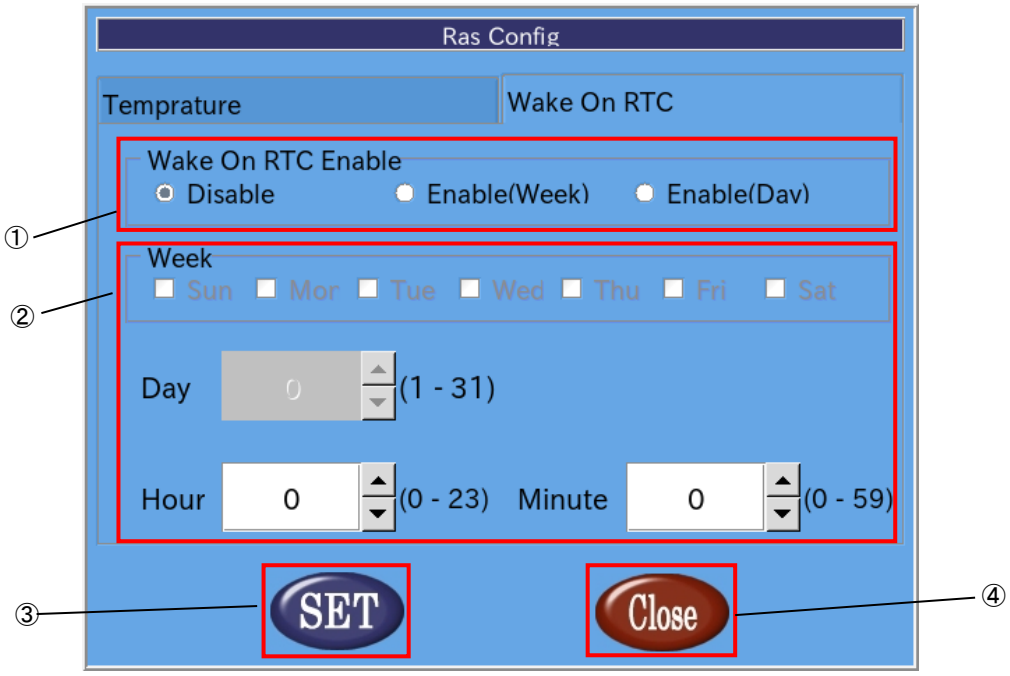


図 2-3-8-2. Wake On RTC の設定

①　Wake On RTC Enable

Wake On Rtc 機能の設定を行います。

表 2-3-8-2. 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。

②　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 2-3-8-3. 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)
Day	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。
Minute	分を設定します。

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 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。

２－３－９　ASD Shutdown Menu について

ASD Shutdown Menu によって、1A 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。
ASD Shutdown Menu を起動するには、メニュー画面から[Shutdown]を選択します。

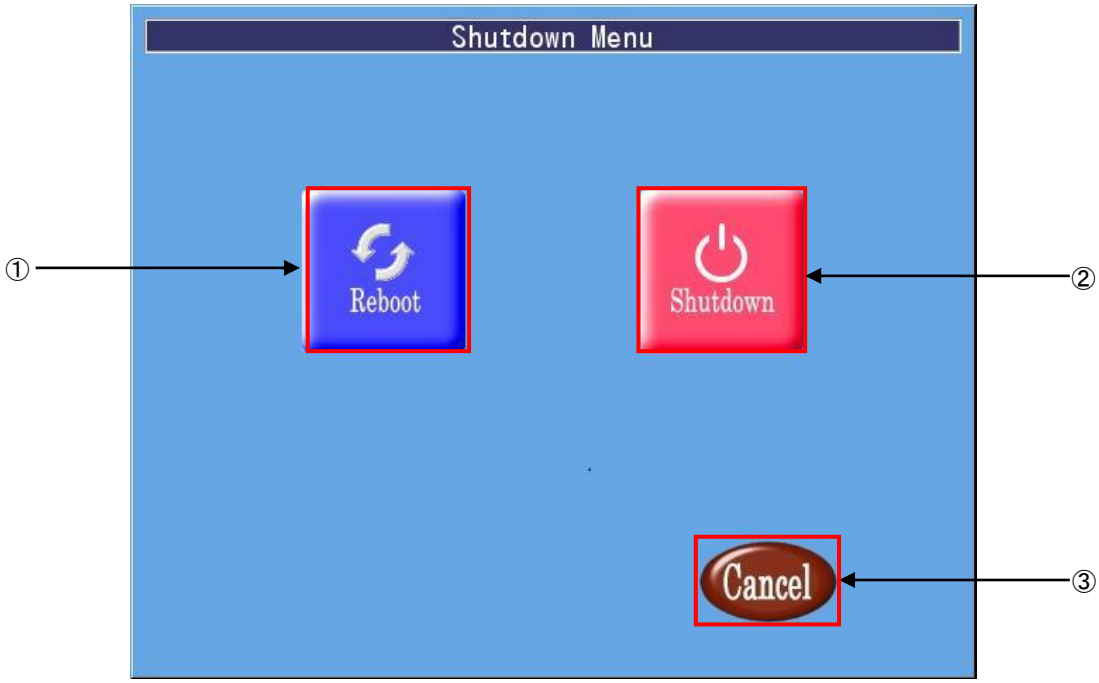


図 2-3-9-1. ASD Shutdown Menu

① 再起動
[Reboot]ボタンを押下することで、1A 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、1A 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 ASD Config Menu に戻ります。

２－４　有線 LAN の設定について

本稿では、1A 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　[スタートボタン]→[設定]→[ネットワーク接続]を選択します。

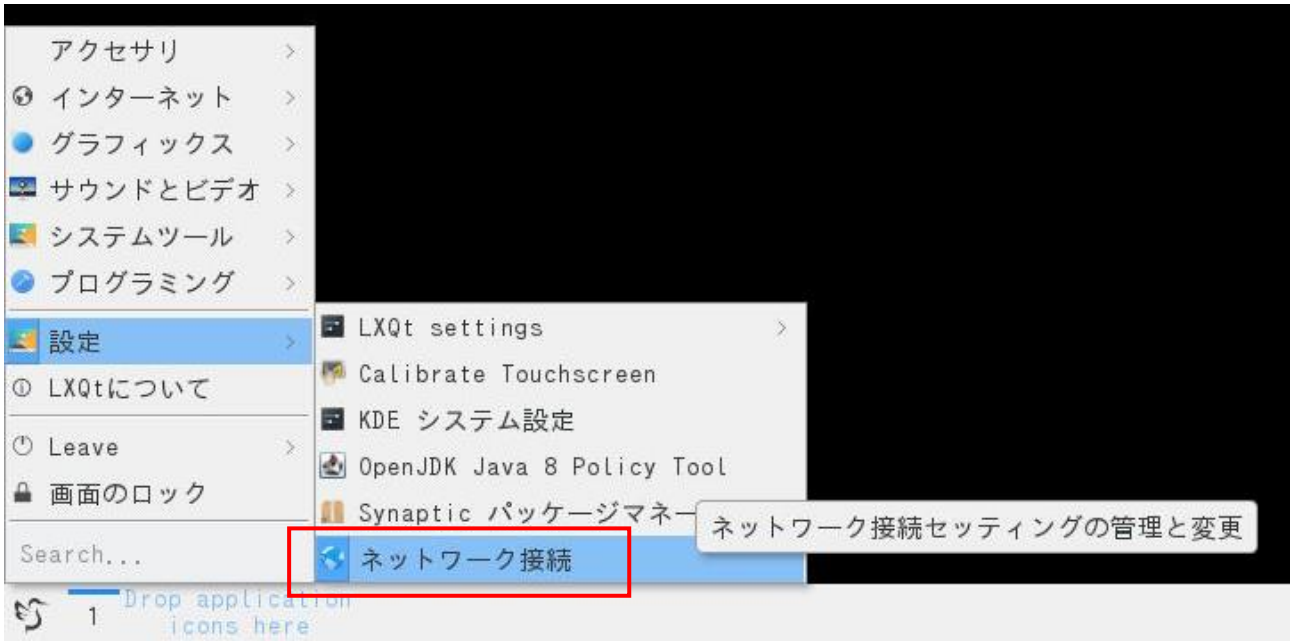


図 2-4-1. ネットワーク設定画面の起動

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。

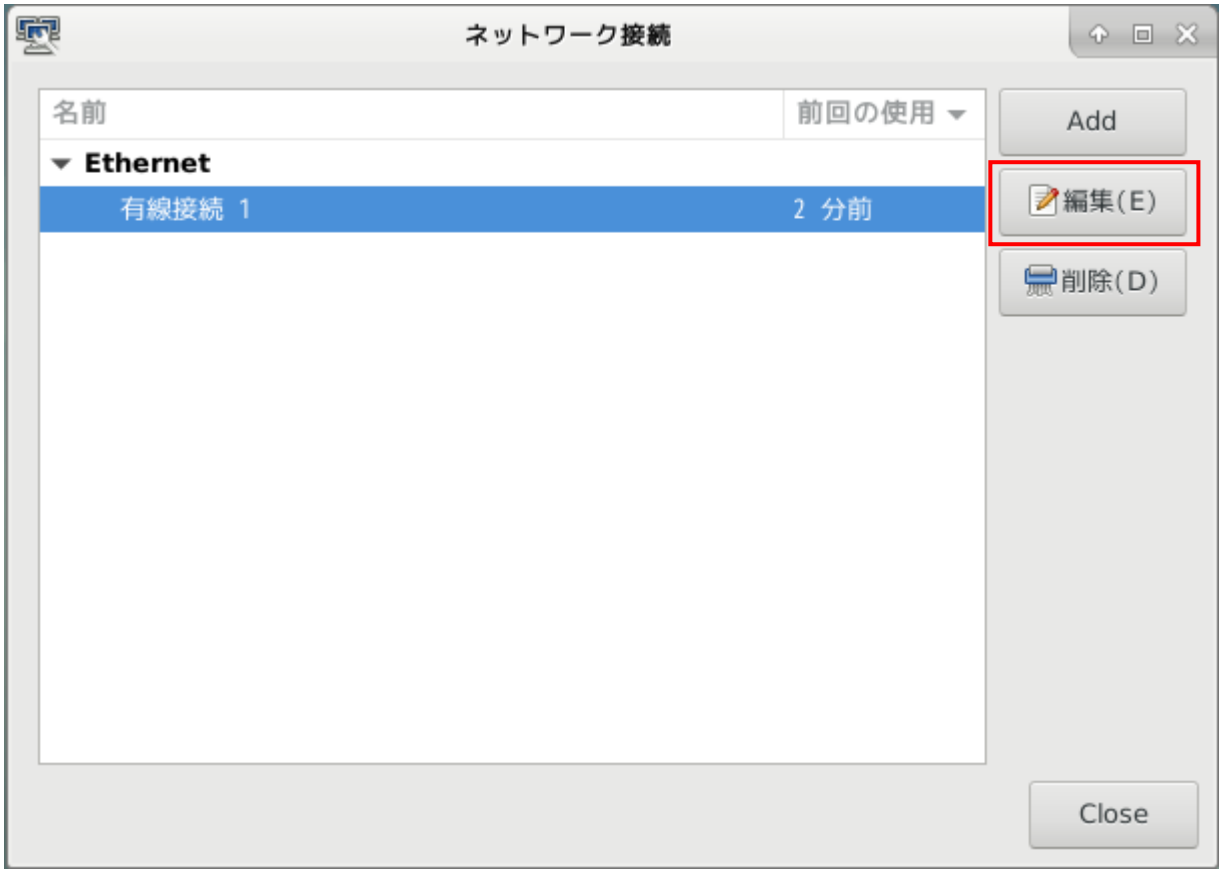


図 2-4-2. ネットワーク設定画面

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。
Debian9.0 からは「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、1A 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。

●接続方法

① デスクトップ画面右下のネットワークアイコン  をクリックすると、通信可能なアクセスポイントの一覧が表示されます。(通信状況によってネットワークアイコンは変化します。)
通信したいアクセスポイントを選択してください。

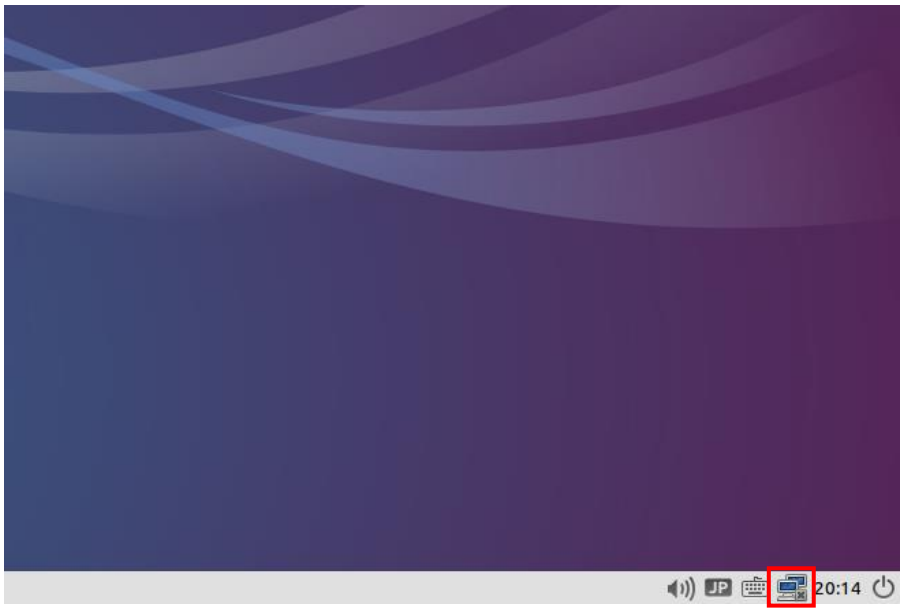


図 2-5-1. ネットワークアイコン

② 図 2-5-2 のようなキー入力画面が表示されます。キーを入力し、[接続]を押してください。

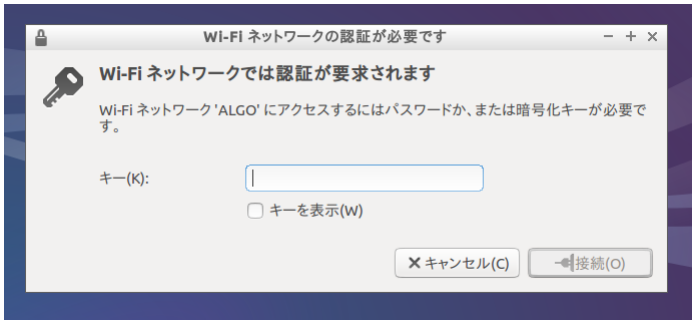


図 2-5-2. キー入力画面

③　接続に成功すると、図 2-5-3 のように、デスクトップ右下のネットワークアイコンが  から  に変化し、画面右上に接続成功の表示が現れます。
（通信状況によってネットワークアイコンは変化します。）

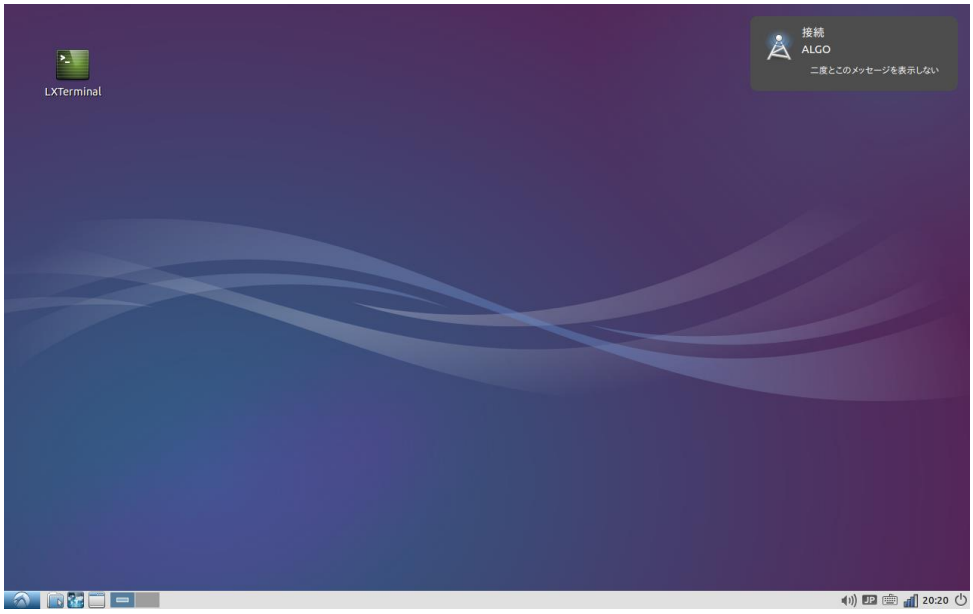


図 2-5-3.　接続成功

●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］→［ネットワーク接続］を選択します。

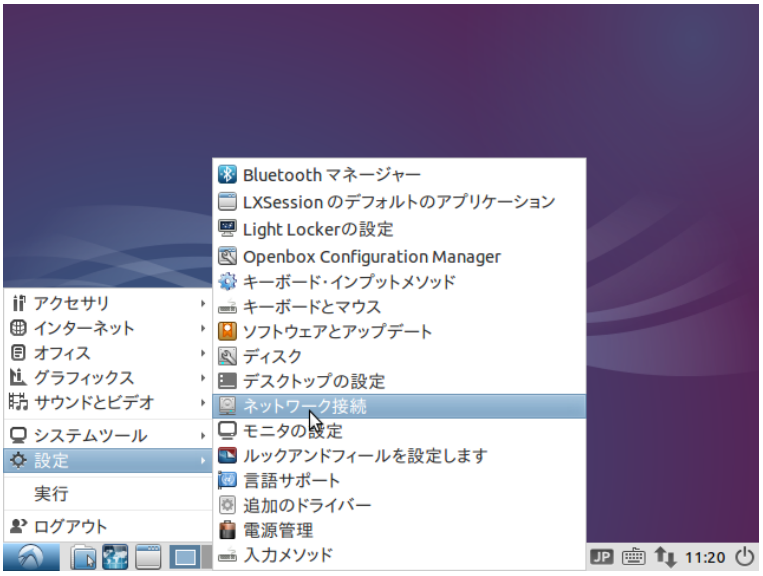


図 2-5-4.　ネットワーク設定画面の起動

②　図 2-5-5 のようなネットワーク設定画面が起動し、アクセスポイントの一覧が表示されます。設定し　たいアクセスポイントを選択して[編集]ボタンをクリックします。

図 2-5-5. ネットワーク設定画面

③　図 2-5-6 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。

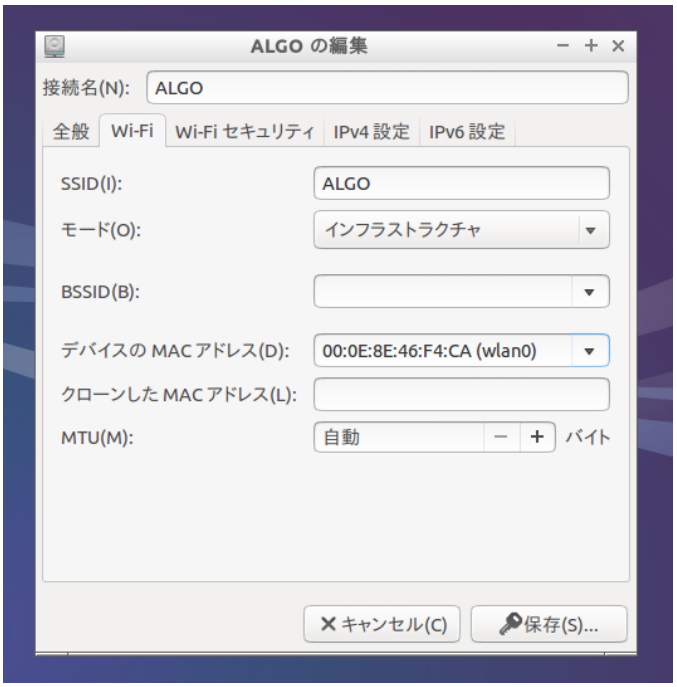


図 2-5-6. 無線 LAN 設定画面

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。

1A 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/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。

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/class/thermal/thermal_zone0/temp	XX000	Read することで CPU Core0 温度を取得します。
/sys/class/thermal/thermal_zone1/temp	XX000	Read することで CPU Core1 温度を取得します。

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、書込み可否の切り替えを行います。書込み不可状態では、ファイルの書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
```

２－８　ソフトウエアキーボードについて

Algonomix6 ではソフトウエアキーボードがあらかじめセットアップされています。
ソフトウエアキーボードは以下の手順で起動することができます。

①　[メニューアイコン]→[ユニバーサル・アクセス]→[Onboard]を選択します。

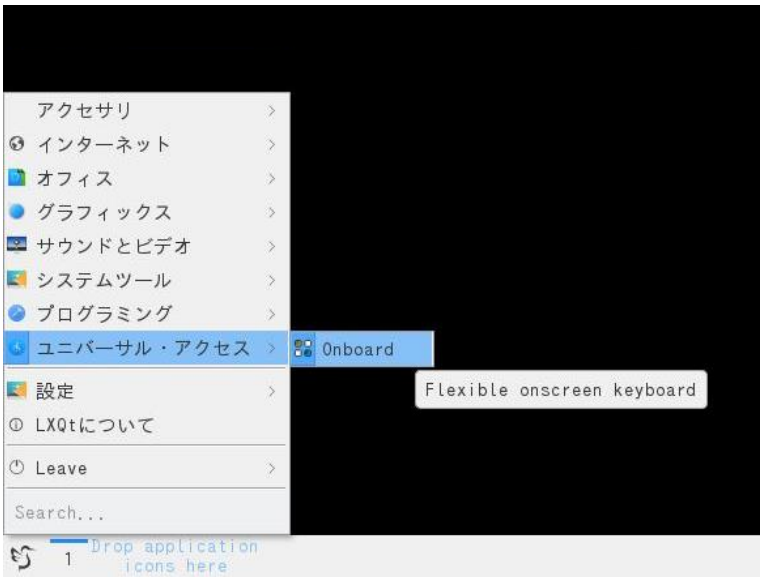


図 2-8-1. ソフトウエアキーボードの起動

②　ソフトウエアキーボードが表示されます。

図 2-8-2. ソフトウエアキーボード

本ソフトウエアキーボードはタッチパネルをマウスとして使用できるようにするモードを持ちます。
この機能を使用する場合はマウスカーソルキー(※)をタッチしてください。

第３章　開発環境

Algonomix6 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、「Algonomix6 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となるのが、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。

1A 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）

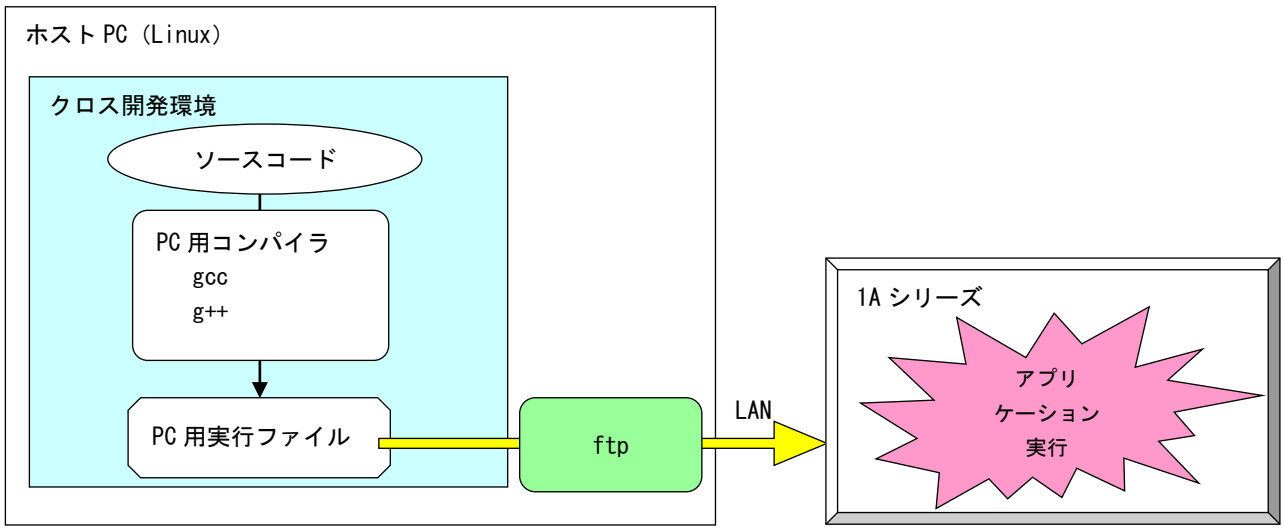


図 3-1-1. クロス開発方式イメージ図

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

Algonomix6 開発環境は、Debian9.0 ディストリビューションイメージに、Intel CPU 用と ARM CPU 用の２種類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 Algonomix6 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　産業用組込み PC EC1A IoT シリーズについて

本章では、1A 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。

Algonomix6 開発環境には、コンソール用と WideStudio 用の 2 種類のサンプルプログラムのソースコードを用意しています。それぞれ表 4-1 にコンソール用サンプル、表 4-2 に WideStudio 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。WideStudio 用のサンプルプログラムは、WideStudio を起動して、プロジェクトファイルをオープンしコンパイルします。コンパイル方法は『Algonomix6 開発環境ユーザーズマニュアル』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-arm64/samples/sampleConsole」に、WideStudio 用サンプルプログラムは「/usr/local/tools-arm64/samples/sampleWideStudio」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 1A 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_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	『4-8 ブザー』を参照
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_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 設定方法	『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	―

表 4-2. WideStudio 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_MultiLang	多言語表示	
sample_10Key	10 キー入力画面	
sample_Launcher	起動ランチャー	
sample_ColorPalette	カラーパレット画面	
sample_ColorPaletteCustom	カラーパレットの変更	
sample_Gui	hello サンプル	
sample_FontSet	WideStudio 上でのフォントの変更	

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、1A 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。

代表例として EC1A-010AT の外観図を示します。また、各部名称を表 4-3 に示します。

※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エアマニュアルをご参照ください。

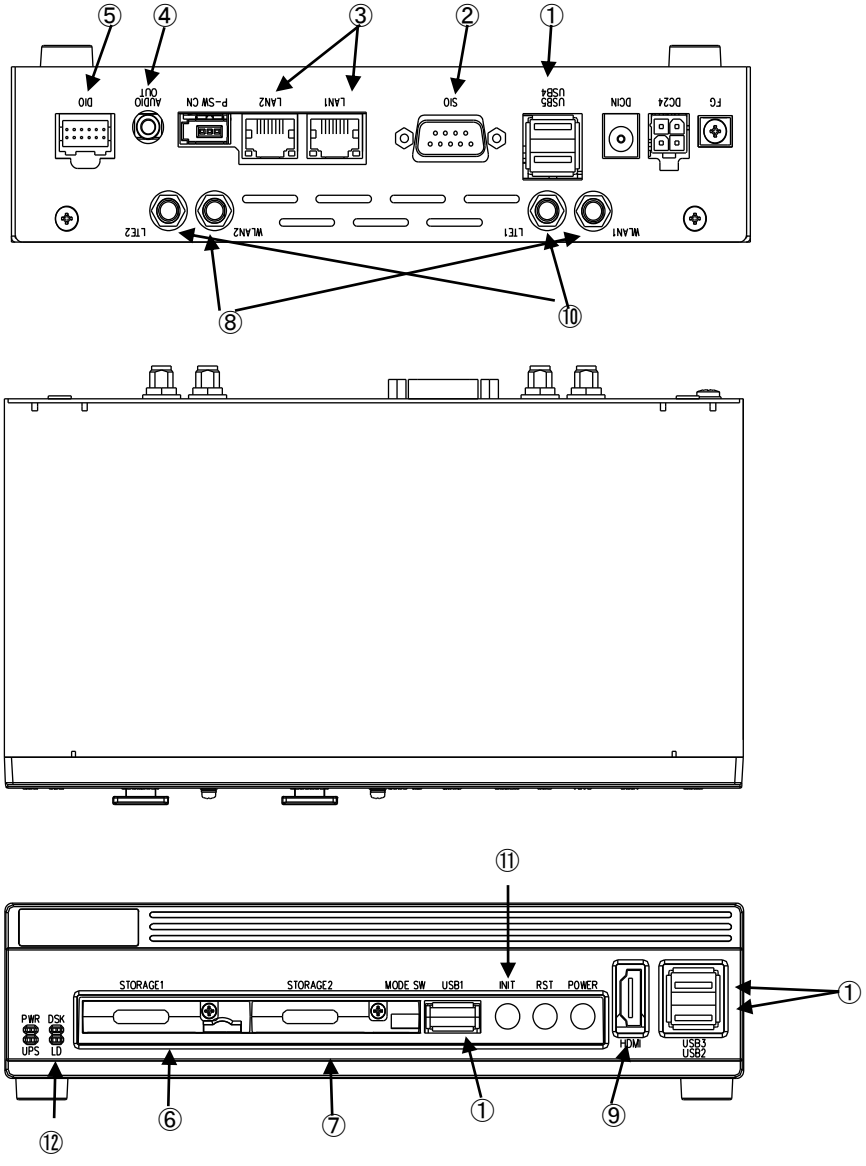


図 4-1.　外形図（EC1A-010AT）

表 4-3. 各部名称

No.	名称	機能	説明	実装数 ※1
①	USB2.0 インターフェース	USB2.0 ポート	USB1.1/2.0 の機器を接続することができます。	5
②	シリアル インターフェース	シリアルポート	シリアル通信が行えます。 RS-232C の通信を行うことができます。 SIO1: ttyUSB0 SIO2: ttyUSB1	1
③	ネットワーク インターフェース	有線 LAN	ネットワークポートとして使用できます。	2
④	オーディオ出力	サウンド	外部スピーカに音声を出力できます。	1
⑤	DIO インター 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。	1
⑥	マイクロ SD カード スロットモジュー ル	マイクロ SD カー ド	CPU モジュールに内蔵されている eMMC へのリカバリ用にマイクロ SD カードを挿入することができます。 詳細は、「**エラー! 参照元が見つかりません。エラー! 参照元が見つかりません。**」を参照してください。	1
⑦	m-SATA ストレージ デバイス （オプション）	mini m-SATA	記憶領域として使用することができます。 Mini mSATA: サブストレージ(オプション) オプションでデータ領域を追加することができます。	(1)
⑧	無線 LAN （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。	(1)
⑨	HDMI インターフェース	グラフィック サウンド	外部モニタに画像と音声を出力することができます。	1
－	Bluetooth※2 (オプション)	Bluetooth	Bluetooth 機器を接続することができます。	(1)
⑩	LTE (オプション)	LTE	LTE 通信を使用することができます。	(1)
⑪	初期化スイッチ	スイッチ	電源投入時、初期化スイッチを 3 秒間押すことで、LD1 が点滅します。また、アプリケーションにより電源投入時にスイッチが押されたかどうかを確認できます。	1
⑫	汎用 LED	LED	アプリケーションにより 3 点の LED の点灯/消灯を制御できます。 LD1: 電源投入後､SW1 を 3 秒間長押しすると点滅します。 LD2: アプリケーションで制御できます。 LD3: アプリケーションで制御できます。	1

※1　実装数の()の数値はオプションを表します。

※2：対応するプロファイル一覧を表 4-4 に示します。

表 4-4. Bluetooth 対応プロファイル一覧

プロファイル名	説明
A2DP 1.3	高度オーディオ配信プロファイル
AVRCP 1.5	オーディオ/ビデオ リモート制御プロファイル
DI 1.3	デバイス識別プロファイル
HDP1.0	ヘルスデバイス プロファイル
HID 1.0	ヒューマン インターフェイス デバイス プロファイル
PAN 1.0	パーソナルエリア ネットワーク プロファイル
SPP 1.1	シリアルポート プロファイル
PXP 1.0	近接プロファイル
HTP 1.0	体温計プロファイル
HoG 1.0	マウスやキーボードなどを接続するためのプロファイル
TIP 1.0	タイム プロファイル
CSCP 1.0	回転速度とケイデンス プロファイル
FTP 1.1	ファイル転送プロファイル
OPP 1.1	オブジェクト プッシュ プロファイル
PBAP 1.1	電話帳情報を伝送するためのプロファイル
MAP 1.0	メッセージ アクセス プロファイル
HFP 1.6 (AG &HF)	ハンズフリー プロファイル

４－１　汎用入出力

４－１－１　汎用入出力について

1A 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　汎用入出力の制御関数について

汎用入出力は値の読み書きを行う制御関数が用意されています。
以下に汎用入出力用制御関数のリファレンスを示します。

asd_genin_get 関数

機能　汎用入力 6 点の読み出しを行います。

書式　int asd_genin_get(unsigned short *dat)

引数　*dat　：汎用入力の読み出しデータ

戻り値　エラーコード(0:正常, -1:異常)

説明　汎用入力 6 点の現在の入力状態の読み出しを行います。
読み出したデータは*dat に格納されます。

bit	7	6	5	4	3	2	1	0
IN	／	／	IN5	IN4	IN3	IN2	IN1	IN0

asd_genout_get 関数

機能　　汎用出力 4 点の読み出しを行います。

書式　　asd_genout_get(unsigned short *dat)

引数　　*dat　　：汎用出力の読み出しデータ

戻り値　　エラーコード(0:正常, -1:異常)

説明　　汎用出力 4 点の現在の出力状態の読み出しを行います。
読み出したデータは*dat に格納されます。

bit	7	6	5	4	3	2	1	0
OUT					OT3	OT2	OT1	OT0

genout_set 関数

機能　　汎用出力 4 点の書き込みを行います。

書式　　genout_set(unsigned short dat)

引数　　dat　　：汎用出力の書き込みデータ

戻り値　　エラーコード(0:正常, -1:異常)

説明　　汎用出力 4 点に dat の値を書き込みます。

bit	7	6	5	4	3	2	1	0
OUT					OT3	OT2	OT1	OT0

４－１－３　汎用入出力サンプルプログラム

●WideStudio 用サンプルプログラム
「/usr/local/tools-arm64/samples/sampleConsole/sample_DIO」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。

このサンプルプログラムは、実行時に汎用出力 4 点を 1 点ずつ順に出力したあと、汎用入力 6 点を 1 点ずつ読み出して標準出力に表示します。

汎用入出力の各関数の使用方法を記したコードをリスト 4-1-3-1 に示します。
main 関数は、asd_genout_set で汎用出力へ書き込みを行い、asd_genin_get 関数で汎用入力の読み出しを行います。

リスト 4-1-3-1. 汎用入出力の各関数（main.cpp）

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

#include "asd_sio.h"
#include "asd_siolib.h"
#include "asd_misclib.h"

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

    /* 汎用出力の 4 点を 1 点ずつ ON */
    for (i=0; i<4; i++){
        asd_genout_set(outdata);
        outdata <<= 1;
        usleep(500 * 1000L);
    }

    /* 汎用出力を OFF */
    outdata = 0x00;
    asd_genout_set(outdata);

    /* 汎用入力 */
    asd_genin_get(&indata);
```

```
    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");

    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

1A シリーズは、RS232C 通信が可能なシリアルポートを持っており、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/ttyUSB0	汎用のシリアルポート 1
2	/dev/ttyUSB1	汎用のシリアルポート 2

※AS1A シリーズは 1 ポートのみです。

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。

４－２－２　シリアルポート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-arm64/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 4-2-2-1 に示します。

シリアルポート 1「/dev/ttyUSB0」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 4-2-2-1. 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/ttyUSB0", O_RDWR | O_NOCTTY);
    if (comm_fd == -1){ /* エラー処理 */
        err_no = errno;
        fprintf(stderr, "ttyUSB0 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, "ttyUSB0 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, "ttyUSB0 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, "ttyUSB0 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, "ttyUSB0 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, "ttyUSB0 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、サーバに接続にい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。

４－３－３　ネットワーク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-arm64/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 1A 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 4-3-3-1 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 4-3-3-1. 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 4-3-3-2 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 4-3-3-2. 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[] = "ABCDEFGHIJKLMNOPQRST";
    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);       /* ﾀｰｹﾞｯﾄｻｰﾊﾞ IP ｱﾄﾞﾚｽ */
        srv_addr.sin_port        = htons(8900);             /* ﾀｰｹﾞｯﾄｻｰﾊﾞﾎﾟｰﾄ番号 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
- 汎用入力 IN1 割り込み
- ハードウェア・ウォッチドッグタイマ
- バックアップ RAM

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。
制御関数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 リセット制御関数について

汎用入力 IN0 リセットは有効・無効を制御する関数が用意されています。
以下に汎用入出力 IN0 リセットのリファレンスを示します。

asd_rasin_in0rst_enable 関数

機能　汎用入力 IN0 リセット機能を有効に設定します。

書式　int asd_rasin_in0rst_enable(void)

引数　なし

戻り値　エラーコード(0:正常, -1:異常)

説明　汎用入力 IN0 リセット機能を有効に設定します。
有効にすることで汎用入力の IN0 が ON した場合に本体をリセットするようになります。

asd_rasin_in0rst_disable 関数

機能　汎用入力 IN0 リセット機能を無効に設定します。

書式　int asd_rasin_in0rst_disable(void)

引数　なし

戻り値　エラーコード(0:正常, -1:異常)

説明　IN0 リセット機能を無効に設定します。
無効にすることで汎用入力の IN0 が ON した場合に本体をリセットしないようになります。

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-arm64/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ソース
**/
#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 "asd_sio.h"
#include "asd_siolib.h"
#include "asd_misclib.h"

int main(void)
{
    asd_rasin_in0rst_enable();

    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。
制御関数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 割込み制御関数について

汎用入力 IN1 割り込みは有効・無効を制御する関数が用意されています。
以下に汎用入力 IN1 割り込みのリファレンスを示します。

asd_rasin_in1int_enable 関数

機能　汎用入力 IN1 割り込み機能を有効に設定します。

書式　int asd_rasin_in1int_enable (void)

引数　なし

戻り値　エラーコード(0:正常, -1:異常)

説明　汎用入力 IN1 割り込み機能を有効に設定します。
有効にすることで汎用入力 IN1 が ON した場合に割り込みを発生させるようになります。

asd_rasin_in1int_disable 関数

機能　汎用入力 IN1 割り込み機能を無効に設定します。

書式　int asd_rasin_in1int_disable (void)

引数　なし

戻り値　エラーコード(0:正常, -1:異常)

説明　汎用入力 IN1 割り込み機能を無効に設定します。
有効にすることで汎用入力 IN1 が ON した場合に割り込みを発生させないようになります。

sio_irqin1 関数

機能　　汎用入力 IN1 割り込みイベントが発生するまで待機します。

書式　　int sio_irqin1(void)

引数　　なし

戻り値　　エラーコード(0:正常, -1:異常)

説明　　汎用入力 IN1 割り込みイベントが発生するまで待機します。
実行すると待機状態になり、汎用入力 IN1 割り込みイベントが発生すると制御を戻します。

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-arm64/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ソース
**/
#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 "asd_sio.h"
#include "asd_siolib.h"
#include "asd_misclib.h"

int main(void)
{
    /*
     * IN1 割り込みを有効にする
     *
     * 有効の場合、無効に変更し終了
     * onoff: 1    有効
     *      : 0    無効
     */
    asd_rasin_in1int_enable();

    /*
     * イベント通知待ち
     */
    while(1){
        asd_intin1_enable();
        sio_irqin1();
        asd_intin1_disable();

        printf("IN1 Int ON¥n");
    }
    asd_rasin_in1int_disable();
    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

1A 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。
ライブラリを使うことにより、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。

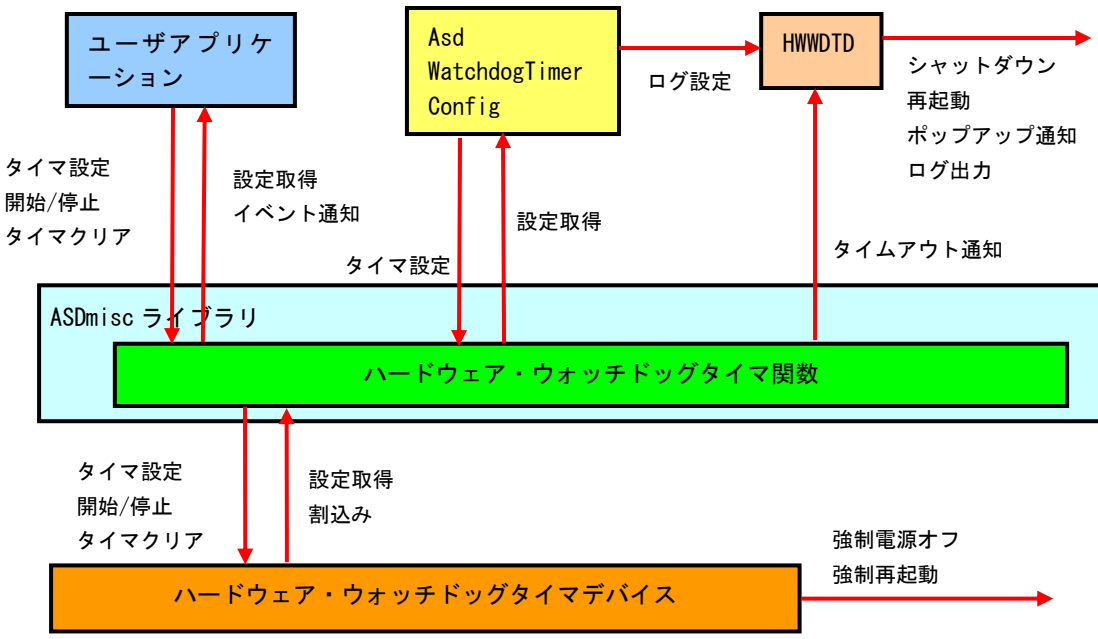


図 4-4-7-1. ハードウェア・ウォッチドッグタイマの構成

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。1A シリーズでは POWER ボタンを 5 秒間長押しすると、強制的に電源 OFF されます。

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、EMMC 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:asdusr
# 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制御関数について

ハードウェア・ウォッチドッグタイマ制御関数を使用することにより、ハードウェア・ウォッチドッグタイマ機能を操作します。

以下に、ハードウェア・ウォッチドッグタイマの詳細を示します。

asd_hwwdt_keepalive 関数

機能　ハードウェア・ウォッチドッグタイマのキープアライブを発行します。

書式　int asd_hwwdt_keepalive(void)

引数　なし

戻り値　エラーコード(0:正常, -1:異常)

説明　ハードウェア・ウォッチドッグタイマのキープアライブを発行します。
キープアライブを発行するとハードウェア・ウォッチドッグタイマのカウントを初期化します。

asd_hwwdt_start 関数

機能　ハードウェア・ウォッチドッグタイマを開始します。

書式　int asd_hwwdt_start (void)

引数　なし

戻り値　エラーコード(0:正常, -1:異常)

説明　ハードウェア・ウォッチドッグタイマを開始します。
ハードウェア・ウォッチドッグタイマを開始するとカウントが開始され、
カウントが完了すると所定のアクションを実行します。

asd_hwwdt_stop 関数

機能　ハードウェア・ウォッチドッグタイマを停止します。

書式　int asd_hwwdt_stop (void)

引数　なし

戻り値　エラーコード(0:正常, -1:異常)

説明　ハードウェア・ウォッチドッグタイマを停止します。
ハードウェア・ウォッチドッグタイマを停止するとカウントを停止します。

asd_hwwdt_getstatus 関数

機能　　ハードウェア・ウォッチドッグタイマのステータスを取得します。

書式　　int asd_hwwdt_getstatus(unsigned int *status)

引数　　*status　　：ハードウェア・ウォッチドッグタイマステータス
0：前回終了時は強制終了
1：前回終了時は通常終了

戻り値　　エラーコード(0:正常, -1:異常)

説明　　ハードウェア・ウォッチドッグタイマのステータスを取得します。
ステータスには前回終了時に強制終了だったか、通常終了だったかが格納されています。

asd_hwwdt_clrstatus 関数

機能　ハードウェア・ウォッチドッグタイマのステータスを初期化します。

書式　int asd_hwwdt_clrstatus(void)

引数　なし

戻り値　エラーコード(0:正常, -1:異常)

説明　ハードウェア・ウォッチドッグタイマのステータスを初期化します。
ステータスには前回終了時に強制終了だったか、通常終了だったかが格納されています。

asd_hwwdt_config_set 関数

機能　ハードウェア・ウォッチドッグタイマの設定を設定します。

書式　int asd_hwwdt_config_set(unsigned short timer, unsigned short action)

引数

timer	：ハードウェア・ウォッチドッグタイマのカウント値 [*100msec]
Action	：ハードウェア・ウォッチドッグタイマのカウント完了時のアクション 0：シャットダウン 1：リブート 2：ポップアップ 3：イベント 4：パワーオフ 5：リセット

戻り値　エラーコード(0:正常, -1:異常)

説明　ハードウェア・ウォッチドッグタイマの設定を設定します。
Timer の値は 100msec 単位での設定となり、このカウントが完了するまでの間に
キープアライブが発行されなかった場合、Action で指定したアクションを
実行するようになります。

asd_hwwdt_config_get 関数

機能　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。

書式　int asd_hwwdt_config_get(unsigned short *timer, unsigned short *action)

引数

*timer　：ハードウェア・ウォッチドッグタイマのカウント値
[*100msec]

*Action　：ハードウェア・ウォッチドッグタイマのカウント完了時のアクション
0：シャットダウン
1：リブート
2：ポップアップ
3：イベント
4：パワーオフ
5：リセット

戻り値　エラーコード(0:正常, -1:異常)

説明　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。
Timer の値は 100msec 単位での設定となり、このカウントが完了するまでの間に
キープアライブが発行されなかった場合、Action で指定したアクションを
実行するようになります。

asd_intwdt_enable 関数

機能　ハードウェア・ウォッチドッグタイマの割り込み待ちを有効にします。

書式　int asd_intwdt_enable (void)

引数　なし

戻り値　エラーコード(0:正常, -1:異常)

説明　ハードウェア・ウォッチドッグタイマの割り込み待ちを有効にします。
有効にすることでハードウェア・ウォッチドッグタイマのカウントが完了した場合に
割り込みを発生させるようになります。

asd_intwdt_disable 関数

機能　ハードウェア・ウォッチドッグタイマの割り込み待ちを無効にします。

書式　int asd_intwdt_disable (void)

引数　なし

戻り値　エラーコード(0:正常, -1:異常)

説明　ハードウェア・ウォッチドッグタイマの割り込み待ちを無効にします。
有効にすることでハードウェア・ウォッチドッグタイマのカウントが完了した場合に
割り込みを発生させないようになります。

sio_irqwdt 関数

機能　ハードウェア・ウォッチドッグタイマの割り込み待ちを無効にします。

書式　int sio_irqwdt (void)

引数　なし

戻り値　エラーコード(0:正常, -1:異常)

説明　ハードウェア・ウォッチドッグタイマの割り込みイベントが発生するまで待機します。
実行すると待機状態になり、ハードウェア・ウォッチドッグタイマの割り込みイベントが発生すると制御を戻します。

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-arm64/samples/sampleConsole/sample_HwWdtKeepalive」に、ハードウェア・ウォッチドッグタイマ制御関数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制御関数ソースコード (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_sio.h"
#include "asd_siolib.h"
#include "asd_misclib.h"

static 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;

//----------0-------------------------------------------------------------
/*
 * キープアライブスレッド
 */
static void *TimerProc(void *arg)
{
    unsigned int status = 0;
    unsigned int ret;

    ret = asd_hwwdt_config_set(Config.time, Config.action);
    fprintf(stdout, "HWWDT Config Setting ret = %d time=%d action=%d¥n", ret, Config.time,
Config.action);

    /*
     * 前回終了状態の取得
     */
    asd_hwwdt_getstatus(&status);
    if(status){
        fprintf(stdout, "Last Shutdown is force shutdown¥n");
    }
    else{
        fprintf(stdout, "Last Shutdown is normal shutdown¥n");
```

```
    }

    /*
     * ハードウェア・ウォッチドッグタイマのスタート
     */
    asd_hwwdt_start();
    printf("TimerProc: Start¥n");

    fFinish = 0;
    while(1){
        if(!fStart){
            break;
        }
        /* KeepAlive */
        if(!fStop){
            asd_hwwdt_keepalive();
        }
        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
     */
    asd_hwwdt_stop();
    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-arm64/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 <time.h>
#include <stdlib.h>
#include <ctype.h>
#include <fcntl.h>
#include <string.h>
#include <unistd.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/select.h>
#include <sys/wait.h>
#include <bits/local_lim.h>
#include <dirent.h>
#include <pthread.h>


#include "asd_sio.h"
#include "asd_siolib.h"
#include "asd_misclib.h"

#define MAX_HWWDTEVENT 10

//-----------------------------------------------------------------------------
typedef struct {
    int No;
    pthread_t ThreadID;
} HWWDTEVENT_INFO;
```

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

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

    /*
     * イベント通知待ち
     */
    while(1){
        asd_intwdt_enable();
        sio_irqwdt();
        asd_intwdt_disable();

        if(asd_hwwdt_getstatus(&status) == ASDMISC_ER_OK){
            if(status){
                asd_hwwdt_clrstatus();
                printf("WDT_TimeUp¥n");
            }
        }
    }

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

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

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

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

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

    return 0;
```

```
}

//---------------------------------------------------------------------------
static void DeleteTimer(HWWDTEVENT_INFO *info)
{
    asd_hwwdt_stop();

    pthread_cancel(info->ThreadID);
    info->ThreadID = 0;
}

//---------------------------------------------------------------------------
int main(int argc, char** argv)
{
    int i;
    HWWDTEVENT_INFO info[MAX_HWWDTEVENT];
    char c;

    asd_hwwdt_clrstatus();

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

    return 0;
}
```

４－４－１０　ダミーバックアップ RAM 機能について

1A シリーズでは 4MByte のバックアップ機能付きのダミーバックアップ RAM 領域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ダミーバックアップ 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ダミーバックアップ 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 UPS Config について』を参照してください。

４－４－１１　ダミーバックアップ RAM 領域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 する事によりダミーバックアップ RAM を読み書きできます。また、mmap をつかって、RAM をユーザプログラム内でマッピングし直すことで、直接アクセスすることも可能です。表 4-4-11-1 にデバイスの詳細を示します。

表 4-4-11-1. ダミーバックアップ RAM デバイスリファレンス

RASRAM
名前
ダミーバックアップRAMの制御を行います。
説明
ダミーバックアップ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
ダミーバックアップSRAMデバイス(/dev/asd_sram)を**open**関数でオープンします。 rasramfd = open("/dev/asd_sram", O_RDWR);
READ
read関数を用いてダミーバックアップRAMの値を読込みます。
WRITE
write関数を用いてダミーバックアップRAMに値を書込みます。
MMAP
mmap関数を用いてダミーバックアップRAMを連続したメモリ領域としてマッピングし直すことができます。マッピングし直したメモリアドレスへ直接読み書きすることが可能です。
IOCTL
ioctl関数を用いてダミーバックアップRAMの情報を読み出すことができます。 error = ioctl(fd, ctlcode, param);

４－４－１２　ダミーバックアップ 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-arm64/samples/sampleConsole/sample_BackUpSRAM」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 4-4-12-1 にソースコードを示します。

リスト 4-4-12-1. ダミーバックアップ RAM 制御 Read/Write 方式（rasram_read.c）

```
/**
    バックアップ SRAM 制御方法サンプルソース
**/
#include <fcntl.h>
#include <errno.h>
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>

#define RASRAM_SIZE 0x7F000 /* バックアップ SRAM サイズ */

unsigned char rasram[RASRAM_SIZE];

int main(void)
{
    int             err_no;
    int             desc;
    int             i;
    int             len;
    unsigned char   d;

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

    /*
     * 書込みデータの作成
     */
    for (i = 0, d = 1; i < RASRAM_SIZE; i++, d++){
        rasram[i] = d;
    }

    /*
```

```
     * データの書込み
     */
    lseek(desc, 0, SEEK_SET);
    len = write(desc, &rasram[0], RASRAM_SIZE);
    if (len == -1){
        err_no = errno;
        fprintf(stderr, "rasram write: %s¥n", strerror(err_no));
        close(desc);
        return (-1);
    }
    else if ((len >= 0) &&(len < RASRAM_SIZE)){
        fprintf(stderr, "rasram write: len[%d]¥n", len);
        close(desc);
        return (-1);
    }
    fprintf(stderr, "rasram write: success¥n");
    /*
     * データの読込み
     */
    memset(&rasram[0], 0x00, RASRAM_SIZE);
    lseek(desc, 0, SEEK_SET);
    len = read(desc, &rasram[0], RASRAM_SIZE);
    if (len == -1){
        err_no = errno;
        fprintf(stderr, "rasram read: %s¥n", strerror(err_no));
        close(desc);
        return (-1);
    }
    else if ((len >= 0) &&(len < RASRAM_SIZE)){
        fprintf(stderr, "rasram read: len[%d]¥n", len);
        close(desc);
        return (-1);
    }
    fprintf(stderr, "rasram read: success¥n");

    /*
     * データのチェック
     */
    for (i = 0, d = 1; i < RASRAM_SIZE; i++, d++){
        if(rasram[i] != d){
            fprintf(stderr, "rasram data check: failed¥n");
            close(desc);
            return (-1);
        }
    }
        fprintf(stderr, "rasram compare: success¥n");

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

４－４－１３　汎用スイッチ／汎用 LED について

汎用スイッチは、Read することで汎用スイッチの状態を確認することができます。
汎用スイッチは、電源 ON 直後以外の場合は押下されたときに ON にします。
電源 ON 直後に 3 秒間長押しすると、ON 状態でラッチします。0 を Write することでラッチを解除します。
汎用 LED は、ON したデータを Write することで汎用 LED を点灯することができます。
Read することで、現在の LED の状態を読み出すことができます。

４－４－１４　汎用スイッチ／汎用 LED 制御関数について

汎用スイッチ／汎用 LED の状態を制御する関数が用意されています。
以下に汎用スイッチ／汎用 LED のリファレンスを示します。

asd_miscio_sw_get 関数

機能　汎用スイッチの状態を読み出します。

書式　int asd_miscio_sw_get (unsigned short *dat)

引数　*dat　：汎用スイッチ入力の状態

戻り値　エラーコード(0:正常, -1:異常)

説明　汎用スイッチの状態を確認することができます。
汎用スイッチは、電源 ON 直後に 3 秒間長押しすると、ON 状態でラッチします。0 を Write することでラッチを解除します。
それ以外の場合は押下されたときに ON にします。

asd_miscio_led_get 関数

機能　　汎用 LED 出力の状態を読み出します。

書式　　int asd_miscio_led_get (unsigned short *dat)

引数　　*dat　　：汎用 LED 出力の状態

戻り値　　エラーコード(0:正常, -1:異常)

説明　　汎用 LED 出力の状態を読み出します。

asd_miscio_led_set 関数

機能　　汎用 LED の ON/OFF を制御します。

書式　　int asd_miscio_led_set (unsigned short dat)

引数　　dat　　：汎用 LED の出力

戻り値　　エラーコード(0:正常, -1:異常)

説明　　汎用 LED の ON/OFF の状態を変化させます。

４－４－１５　Wake On RTC 機能について

1A シリーズには RTC デバイスとして、EPOSON 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「２－３－８　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 4-4-15-1 に記述された関数を使って設定します。

表 4-4-15-1. 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。 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/無効をせっていします。
ClearInt	―	WOR 発生フラグをクリアします。

リスト 4-4-15-1 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 4-4-15-1. 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");

}
```

４－５　UPS 機能

４－５－１　UPS 機能について

1A 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、AC 電源駆動からバッテリ駆動に切り替わり、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。1A シリーズで使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	バッテリ保護機能動作
/ups_alarm	電源断警告

バッテリ保護機能動作通知、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 UPS Config について』を参照してください。

バッテリ保護機能はバッテリ温度が 4～67℃の範囲外になると動作します。

４－５－２　UPS 機能制御関数について

UPS 機能はバッテリ状態を取得する関数が用意されています。
以下に UPS 機能のリファレンスを示します。

asd_ups_timclr 関数

機能　UPS バッテリのタイマをクリアします。

書式　int asd_ups_timclr(void)

引数　なし

戻り値　エラーコード(0:正常, -1:異常)

説明　UPS バッテリのタイマをクリアします。

asd_ups_monitor 関数

機能　UPS バッテリ電池の状態を取得します。

書式　int asd_ups_monitor(unsigned short *powersource, unsigned short *condition)

引数　*powersource　：駆動状態
　　0：AC 駆動
　　1：バッテリ駆動

　　*condition　：バッテリコンディション
　　0：チャージ中
　　1：満充電
　　2：放電中
　　3：バッテリ保護機能動作中

戻り値　エラーコード(0:正常, -1:異常)

説明　UPS バッテリ電池の状態を取得します。

asd_ups_batterylow 関数

機能　UPS バッテリの電圧状態を取得します。

書式　int asd_ups_batterylow(unsigned short *low)

引数　*low　：バッテリ電圧
0：正常
1：バッテリ電圧低下

戻り値　エラーコード(0:正常, -1:異常)

説明　UPS バッテリの電圧状態を取得します。

asd_ups_shutdownact 関数

機能　UPS バッテリのシャットダウンを実行します。

書式　int asd_ups_shutdownact (void)

引数　なし

戻り値　エラーコード(0:正常, -1:異常)

説明　UPS バッテリのシャットダウンを実行します。

４－５－３　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-arm64/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 4-5-3-1 に示します。

リスト 4-5-3-1. 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>

#include "asd_sio.h"
#include "asd_siolib.h"
#include "asd_misclib.h"

#include "asd_ups.h"

struct ups_t {
    int occured;
    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 occured.¥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。
Algonomix6 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割り振られます。

/dev/sdx　　：　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/sdx*　　：　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。

1A 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	EMMC ストレージ
②	m-SATA 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:mSATA ストレージのみ挿入して OS 起動したとき

mSATA 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-SATA ストレージ	/dev/sda
USB スロット 0～3	/dev/sdb、/dev/sdc･･･

例 2:mSATA ストレージを挿入せずに OS 起動したとき

その後挿入された USB メモリは sda、sdb･･･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
USB スロット 0～3	/dev/sda、/dev/sdb･･･

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(SD カードなど)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、m-SATA を/home/asdusr/media 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. m-SATA 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/sda1   /home/asdusr/media  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。
例として Docomo の SP モードを使用して LTE 通信を行う場合、以下のように設定を行います。

① コンソールを起動させ、下記のコマンドを実行します。

```
# wvdialconf /etc/wvdial.conf
```

② OS を再起動してください。

③ [メニューアイコン]→[設定]→[ネットワーク接続]を選択します。

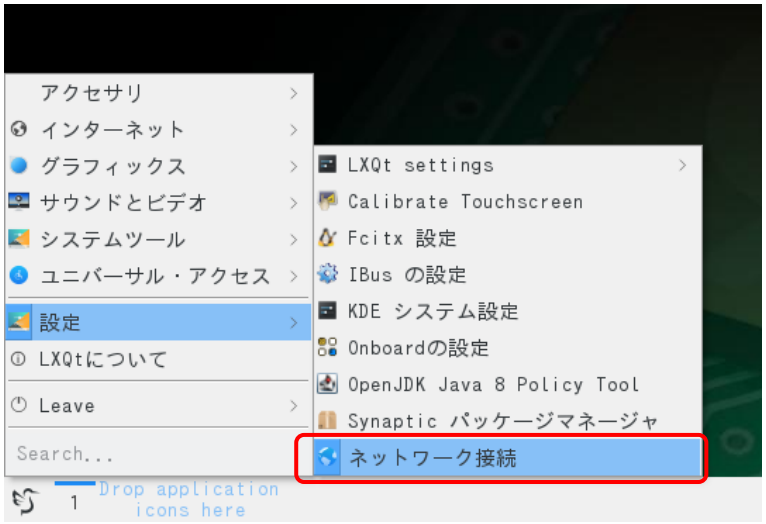


図 4-7-1. ネットワーク設定画面の起動

④ 図 4-7-2 のようなネットワーク設定画面が起動します。
「Add」ボタンをクリックしてください。

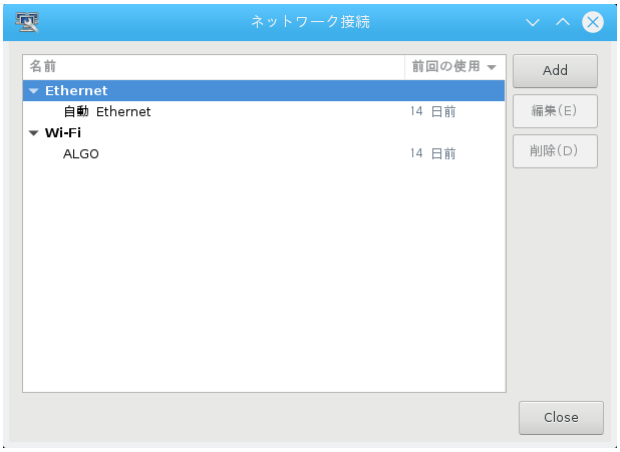


図 4-7-2. ネットワーク設定画面

⑤　図 4-7-3 のような接続種類を選択するダイアログが表示されます。
「モバイルブロードバンド」を選択して「作成」ボタンをクリックしてください。

図 4-7-3.　ネットワーク種類の選択

⑥　モバイルブロードバンド接続のセットアップ画面が開きます。
ご使用の SIM カードの資料を参照し、設定をしてください。

図 4-7-4.　接続のセットアップ

※　設定中、数画面が表示されますが、使用する SIM カードにより画面は異なります。
最終的に図 4-7-6 のような画面が表示されれば完了です。

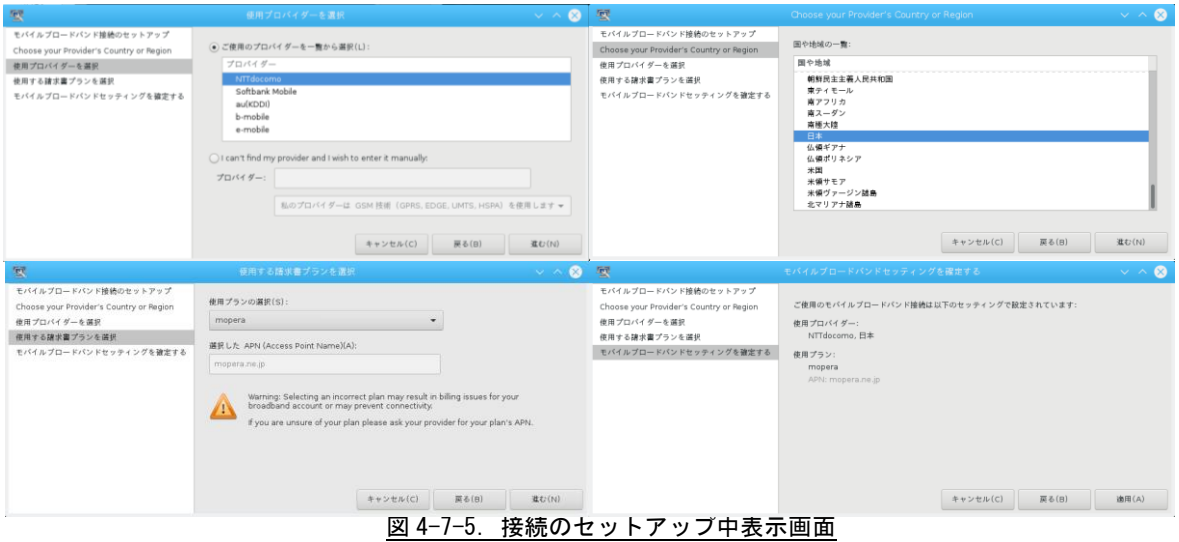


図 4-7-5.　接続のセットアップ中表示画面

⑦　通信設定の編集画面が開きます。
この画面もご使用の SIM カードの内容に従い設定をしてください。

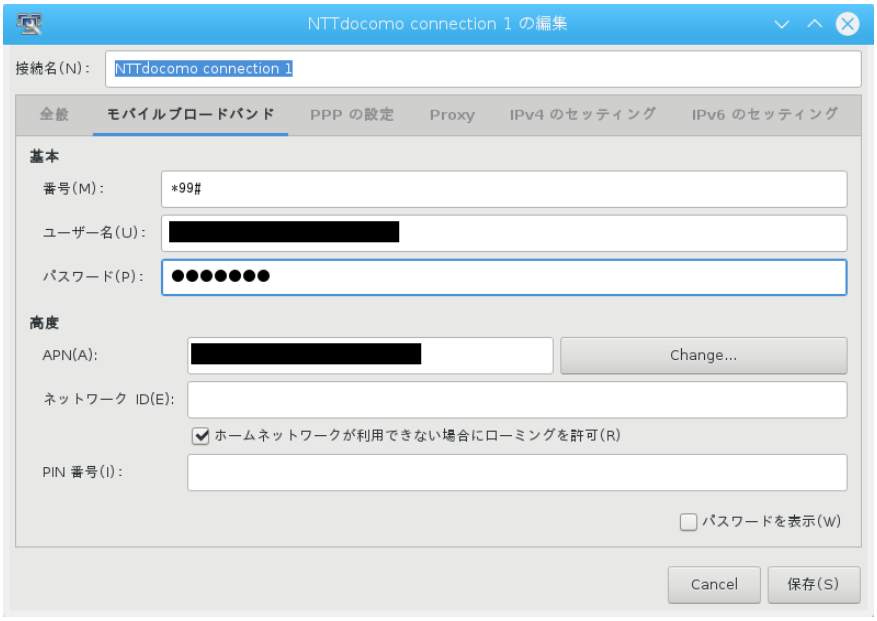


図 4-7-6.　ネットワーク設定の編集

⑧　設定が完了するとタスクバーの右下に設定した項目が追加されます。
この項目をクリックすると通信が開始されます。

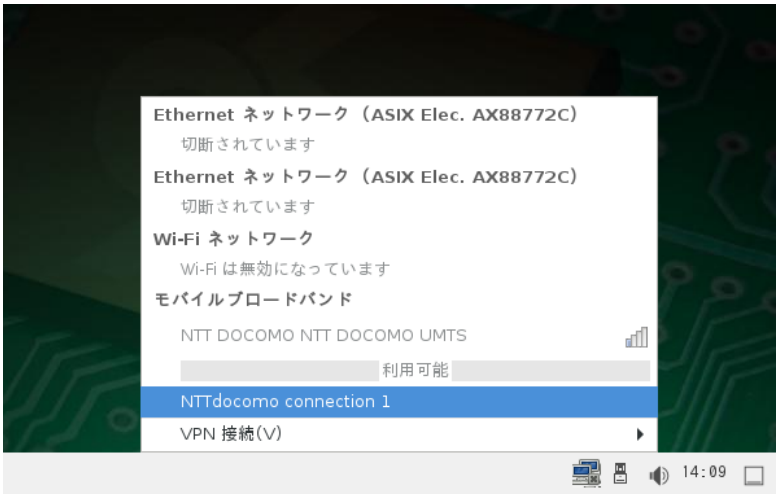


図 4-7-7.　通信の開始

注）　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。

４－８　ブザー

４－８－１　ブザーについて

AP シリーズでは、アプリケーションでブザーの ON/OFF、ブザーの周波数の変更（16 段階）、タッチパネルのタッチ音の ON/OFF を行うことができます。タッチパネルのタッチ音 ON/OFF 設定は、「ASD Misc Setting」からも設定可能です。「２－３－６　ASD Misc Setting について」を参照してください。

ブザーの制御は下記の３種類があります。それぞれ、制御コマンドと、アクセスライブラリ関数を用意しています。

１）タッチパネルブザーの ON/OFF

タッチパネルを押したときに鳴るブザーを ON/OFF します。ON したときは、タッチパネルをタッチするたびにピッという音がなります。

タッチパネルブザーの周波数は固定です。

■コマンド

・タッチパネルブザーを ON するとき

```
# /usr/bin/asd_buzz_autobuzz 1
```

・タッチパネルブザーを OFF するとき

```
# /usr/bin/asd_buzz_autobuzz 0
```

■ライブラリ

libasdmisc.so をリンクさせて、下記の関数を使うことで、アプリケーションから操作することができます。

asd_buzz_autobuzz_set 関数

機能　　タッチパネルブザー機能を ON/OFF させます。

書式　　int asd_buzz_autobuzz_set (int data)

引数　　0：OFF 1：ON

戻り値　　エラーコード(0:正常, -1:異常)

説明　　タッチパネルブザー機能を ON/OFF することが可能です。

２）ブザー制御

ブザーを ON/OFF させることが可能です。このとき、13bit 分独立制御が可能です。13bit のうちいずれかが ON していれば ON したままになります。

■コマンド

・ブザー0 を ON/OFF する場合

```
# /usr/bin/asd_buzz_ctrl 0 1
# /usr/bin/asd_buzz_ctrl 0 0
```

・ブザー12 を ON/OFF する場合

```
# /usr/bin/asd_buzz_ctrl 12 1
# /usr/bin/asd_buzz_ctrl 12 0
```

■ライブラリ

libasdmisc.so をリンクさせて、下記の関数を使うことで、アプリケーションから操作することができます。

asd_buzz_ctrl 関数

機能　指定した番号のブザーを ON/OFF します。

書式　int asd_buzz_ctrl(int bit, int onoff)

引数　bit (0～13：制御する bit を指定する)
onoff (0：OFF 1：ON)

戻り値　エラーコード(0:正常, -1:異常 -2：パラメータ範囲外)

説明　13bit 分あるブザー制御が OR 制御となります。どれかの bit が ON していれば、ブザーは鳴ったままになります。

３）ブザー音色調整

ブザーの音色を 16 段階で設定可能です。300Hz から 4.8KHz まで 300Hz 単位で 16 段階の設定ができます。

※注：タッチパネルタッチ時に鳴るブザーは、1KHz 固定です。

■コマンド

・ブザー音色調節 3KHz(10)に設定する場合

```
# /usr/bin/asd_buzz_tone 10
```

■ライブラリ

libasdmisc.so をリンクさせて、下記の関数を使うことで、アプリケーションから操作することができます。

asd_buzz_tone_set 関数

機能　ブザーの音色を調整します。

書式　int asd_buzz_tone_set (unsigned short tone)

引数　音色（0：300Hz～15：4.8KHz　300Hz 単位 16 段階）

戻り値　エラーコード(0:正常, -1:異常 -2：パラメータ範囲外)

説明　ブザーの音色を 300Hz～4.8KHz まで 300Hz 単位で 16 段階で調整可能です。

asd_buzz_tone_get 関数

機能　ブザーの音色を取得します。

書式　int asd_buzz_tone_get (unsigned short *tone)

引数　現在の状態を格納するポインタ

戻り値　エラーコード(0:正常, -1:異常)

説明　現在設定中のブザーの音色を確認することが可能です。

４－８－２　ブザー制御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-arm64/samples/sampleConsole/sample_BeepOnOff」に、ブザー音を ON/OFF する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は下記のコマンドで実行します。1 を指定することでブザーが鳴り、0 を指定することでブザーが止まります。

```
# sampleBeepOnOff -b 1
# sampleBeepOnOff -b 0
```

サンプルプログラムのソースコードをリスト 4-8-2-1 に示します。

リスト 4-8-2-1. ブザーON/OFF 制御 (main.c)

```
/**
    ブザーの ON/OFF 制御サンプルソース
**/
#include <fcntl.h>
#include <errno.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
#include <signal.h>

#include "asd_sio.h"
#include "asd_siolib.h"
#include "asd_misclib.h"
volatile sig_atomic_t e_flag = 0;

void abrt_handler(int sig);


int main(int argc, char *argv[])
{
    FILE    *fp;
    int     status=-1;
    int     err_no;
    int     c;
    int     res;

    if ( signal(SIGINT, abrt_handler) == SIG_ERR ) {
        exit(-1);
    }

    /*
     * 起動引数からブザーの ON/OFF 変更値を取得します。
     *   0 : ブザーON
     *   1 : ブザーOFF
     */
```

```
    while ((c = getopt(argc, argv, "b:")) != -1){
        switch(c){
            case 'b':
                status = atoi(optarg);
                break;
            default:
                fprintf(stdout, "argument error¥n");
                fprintf(stdout, "  -b : Beep ON/OFF Range[0 or 1]¥n");
                return (-1);
        }
    }
    if ((status != 0) && (status != 1)){
        fprintf(stdout, "Range error Beep ON/OFF Range[0 or 1] ex) -b 0¥n");
        return (-1);
    }

    /*
     * ブザー音の ONOFF の操作を行います
     * ブザーは 13bit のうちいずれかが ON していたら ON されます。
     * ビット番号は 0～13 までを指定できます。
     * 0 を書込むとブザーが OFF します
     * 1 を書込むとブザーが ON します
     *
     */
    asd_buzz_ctrl(0, status);

    return (0);
}
void abrt_handler(int sig) {
  e_flag = 1;
}
```

第５章　システムリカバリ

本章では、マイクロ SD カード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概要を説明します。
詳細は別紙「1A シリーズ リカバリ SD カード取扱説明書」マニュアルを参照してください。

５－１　リカバリについて

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。

- システムの復旧（出荷イメージ）
- 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

リカバリを実行するには以下のものを用意する必要があります。

- マイクロ SD カード

- リカバリ実行の流れ

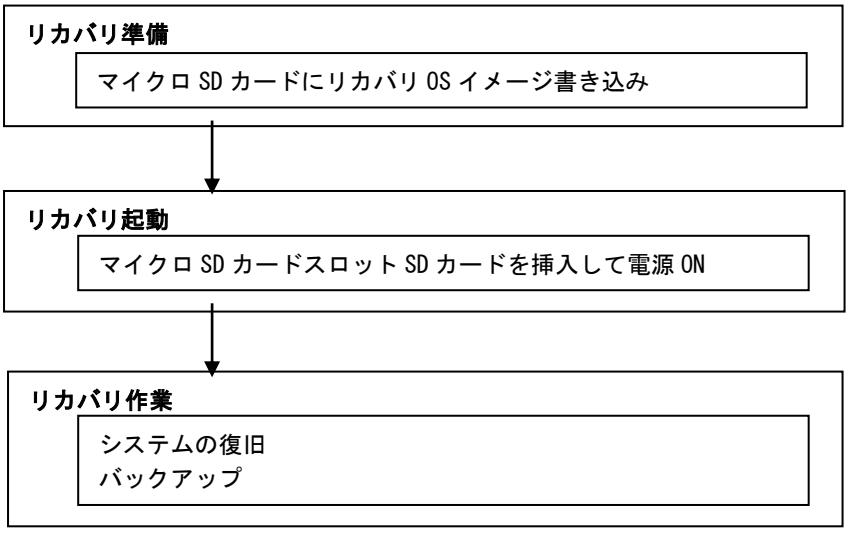


図 5-1-1. リカバリ(マイクロ SD カード利用)の流れ

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付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。

７７Ｇ０５０００６Ｈ　２０２３年　２月　第８版
７７Ｇ０５０００６Ａ　２０１８年　７月　第１版

株式会社アルゴシステム

本社
〒５８７－００２１　大阪府堺市美原区小平尾６５６番地
ＴＥＬ（０７２）３６２－５０６７
ＦＡＸ（０７２）３６２－４８５６

ホームページ　http://www.algosystem.co.jp