ユーザーズマニュアル

産業用組込み PC EC1G-01x/02x

実行環境

目　次

産業用組込みPC EC1G-01x/02x

実行環境

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　デバイスについて

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 EC1G-01x/02x（以下、EC1G-01x/02x）用ディストリビューション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

EC1G-01x/02x 実行環境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。EC1G-01x/02x 実行環境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、EC1G-01x/02x 実行環境の具体的な内容を説明する前に、EC1G-01x/02x 実行環境の概要について説明します。

１－１　EC1G-01x/02x 実行環境について

「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

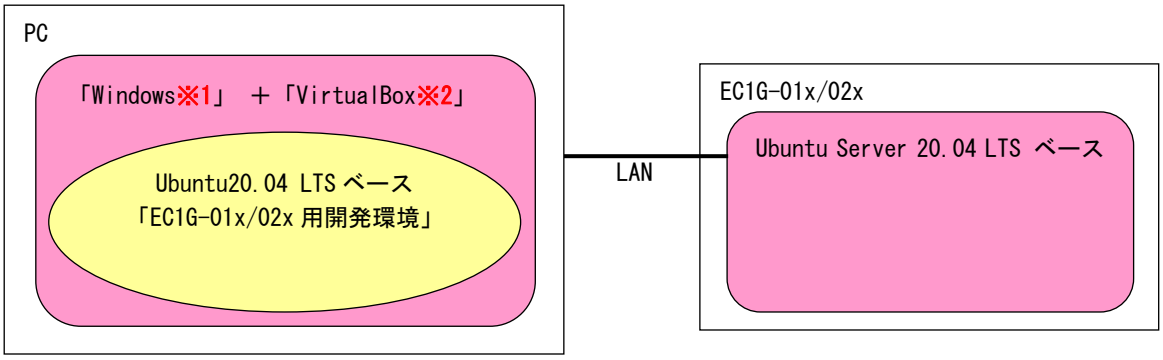
Linux カーネルといくつかの必要なソフトウェアパッケージをまとめて、OS として使えるようにしたものを Linux ディストリビューションといいます。

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。

EC1G-01x/02x 実行環境は、「Ubuntu Server 20.04 LTS」という Linux ディストリビューションをベースに EC1G-01x/02x 用の独自のブートローダー、カーネルを組込んだものです。

パソコン上に EC1G-01x/02x 用の開発環境をインストールすることで、EC1G-01x/02x 用のソフトウェアを開発することができます。

EC1G-01x/02x 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. EC1G-01x/02x 用開発環境

開発環境の詳細については、別紙『EC1G-01x/02x 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。

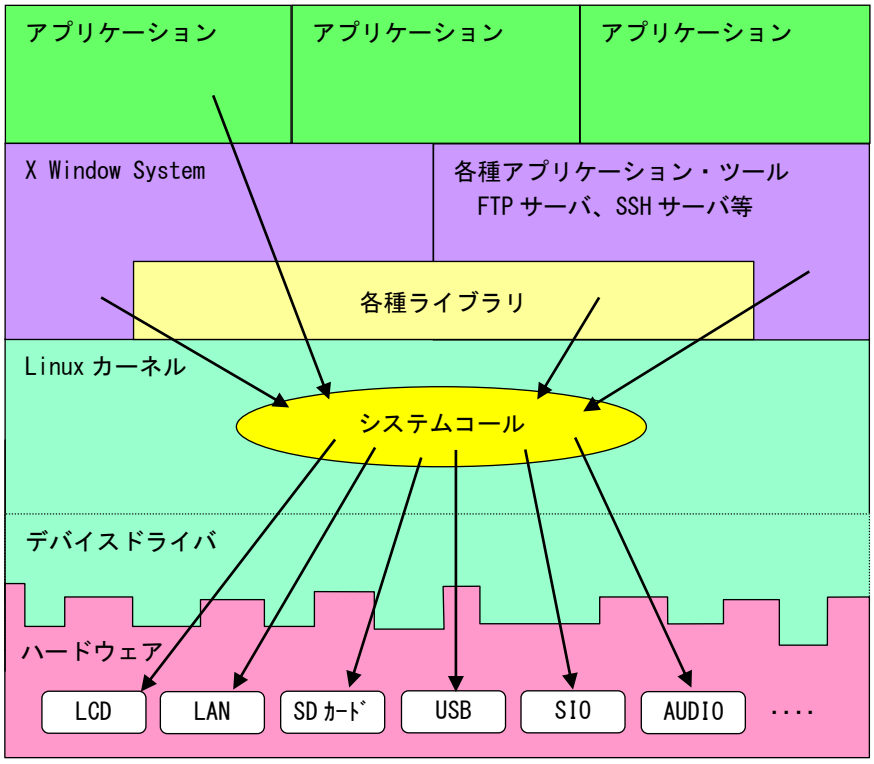


図 1-2-1. Linux ソフトウェア構成図

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　EC1G-01x/02x 実行環境パッケージについて

EC1G-01x/02x 実行環境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 Ubuntu Server 20.04 LTS 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
linux-image-5.4.24	Linux Kernel 本体	5.4.24
ubuntu-desktop-minimal	デスクトップ環境	3.36.3
onboard	スクリーン キーボード	1.4.1-2

EC1G-01x/02x 実行環境では、Ubuntu Server 20.04 LTS 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。

EC1G-01x/02x 実行環境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」を同時入力するか、左下の「アプリケーションを表示する」→「端末」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。

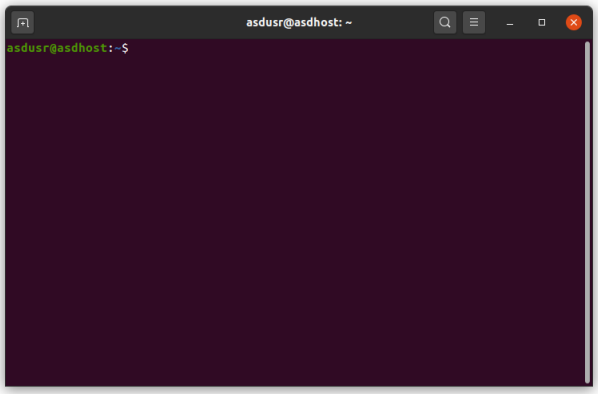


図 2-1-1. ターミナル画面

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

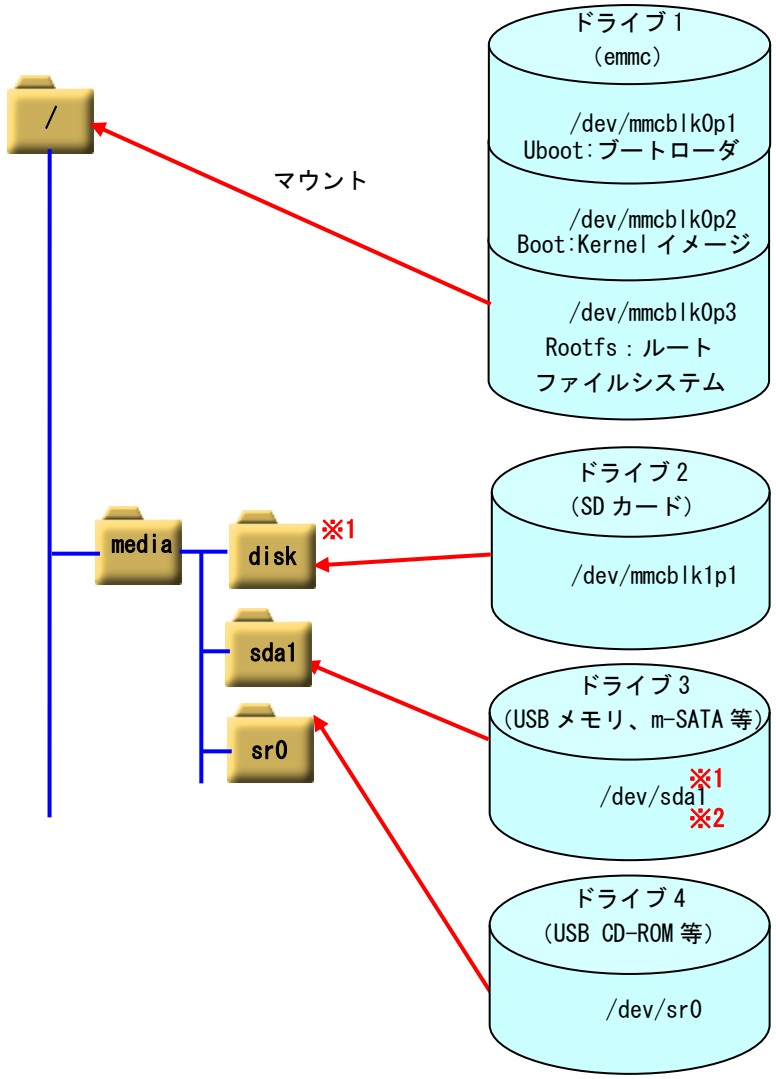
3：パッケージのバージョンおよびリビジョン

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

２－２　EC1G-01x/02x 実行環境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。



※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. Ubuntu のツリー構造

EC1G-01x/02x 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。

EC1G-01x/02x 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. EC1G-01x/02x 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	9Mbyte	-	-	-	-	UBOOT：ブートローダ
/dev/mmcblk0p2	100MByte	-	-	-	-	BOOT：Kernel イメージ
/dev/mmcblk0p3	15Gbyte	4.7Gbyte	9.0Gbyte	35%	/	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
+--media
+--mnt
+--opt
+--proc
+--root
+--run
+--sbin
+--srv
+--sys      --+--fs     --+--cgroup
+--tmp
+--usr      --+--bin
              +--games
              +--include
              +--lib
              +--libexec
              +--local
              +--sbin
              +--share
              +--src
+--var      --+--backups
              +--cache
              +--crash
              +--lib
              +--local
              +--lock
              +--log
              +--mail
              +--metrics
              +--opt
              +--run
              +--snap
              +--spool
              +--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(抜粋)

ディレクトリ名	内容	tempfs
/	ルートディレクトリ	
/bin	/usr/bin へのシンボリックリンクです。	
/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。	
/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。EC1G-01x/02x 実行環境では、「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	/usr/sbin のシンボリックリンクです。	
/srv	HTTP 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　EC1G-01x/02x 設定ツール「ASD Config」について

本項では、EC1G-01x/02x 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
```

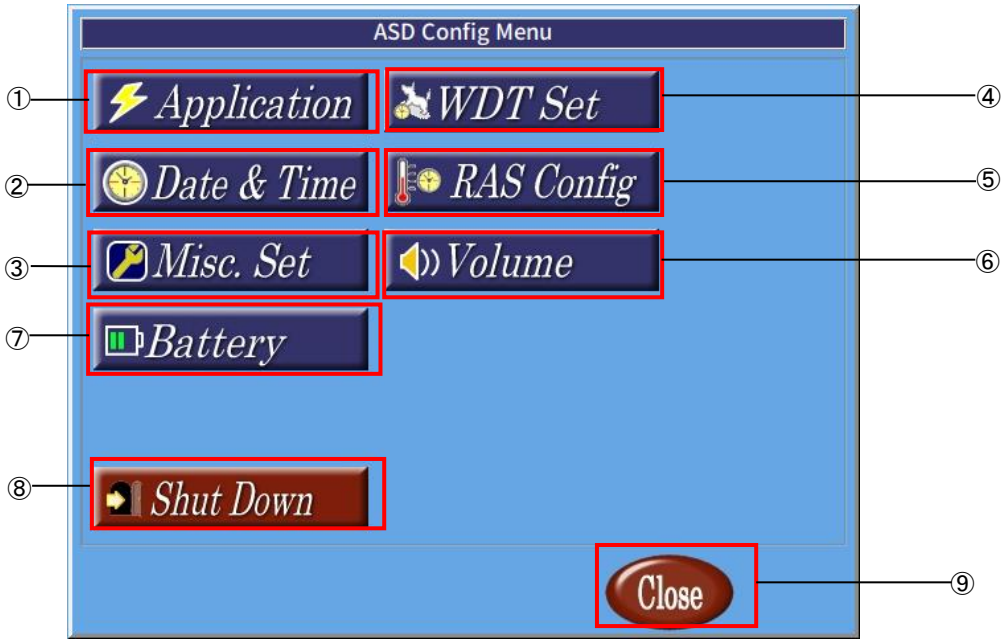


図 2-3-1. ASD Config Menu

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、時計を設定するためのツールです。
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、本製品の製品情報を確認するためのツールです。
詳細は『2-3-3 ASD Misc Setting について』で説明します。

④ 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、『2-3-4 ASD WatchdogTimer Config について』で説明します。

⑤ ASD Ras Config
ASD Ras Config は、CPU 温度の確認や Wake On RTC の設定を行うためのツールです。
詳細は『2-3-5 ASD Ras Config について』で説明します。

⑥　AsdVolumeConfig
AsdVolumeConfig は音量の設定を行うためのツールです。
詳細は『2-3-6 ASD Volume Config について』で説明します。

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

⑧　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、『2-3-8 ASD Shutdown Menu について』で説明します。

⑨　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。

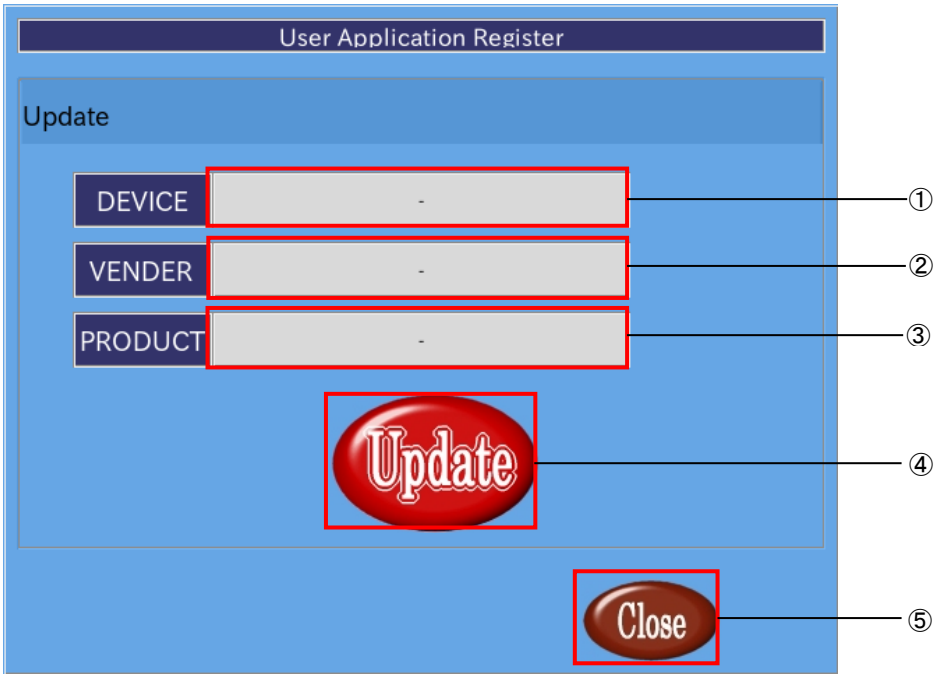


図 2-3-1-1.　アップデート画面

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 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。

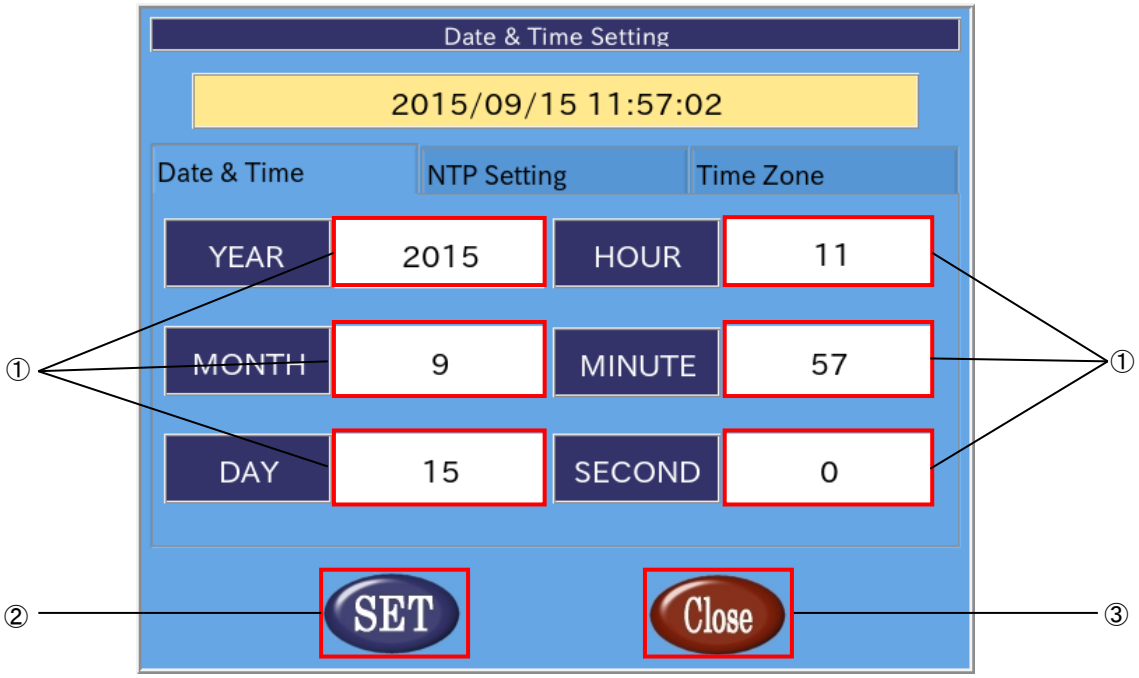


図 2-3-2-1. 時計の設定

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 EC1G-01x/02x 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。

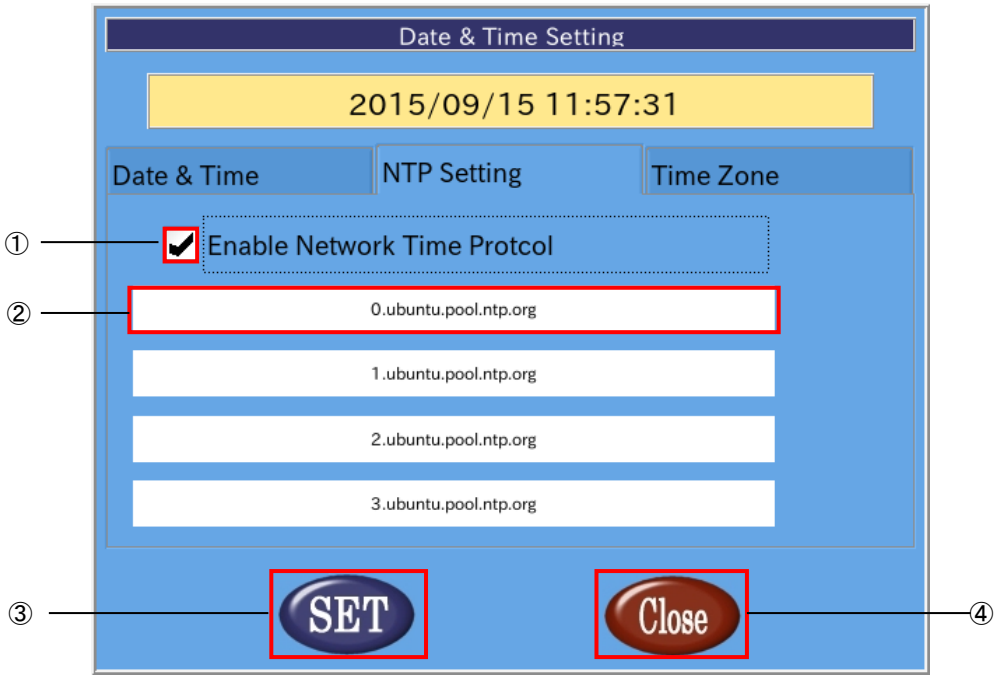


図 2-3-2-2. NTP サーバの設定

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：EC1G-01x/02x 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。

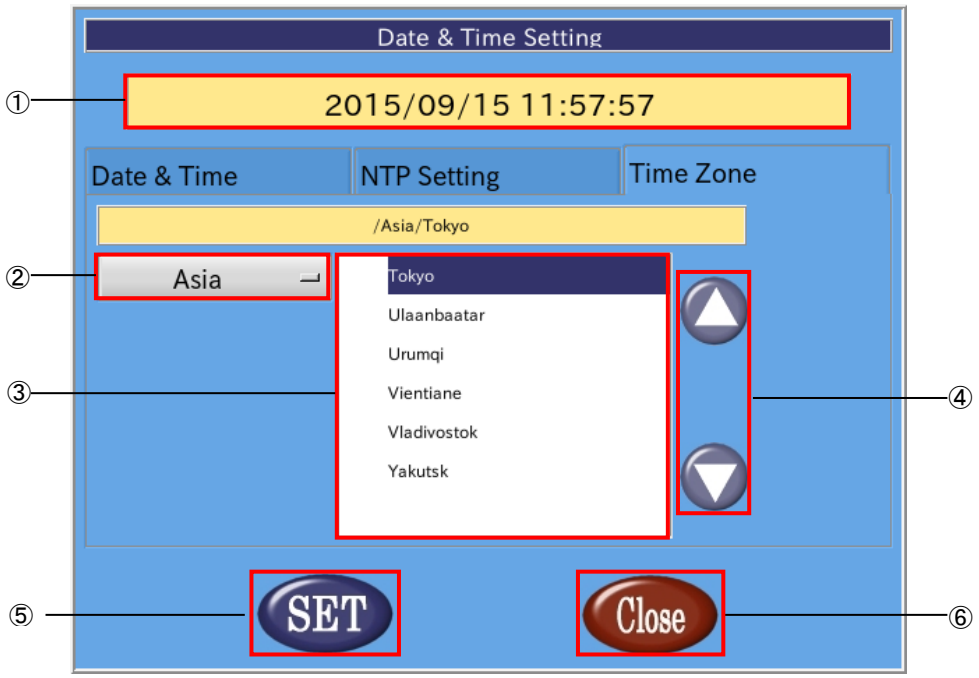


図 2-3-2-3. タイムゾーンの設定

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。

２－３－３ ASD Misc Setting について

ASD Misc Setting は、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。

●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。

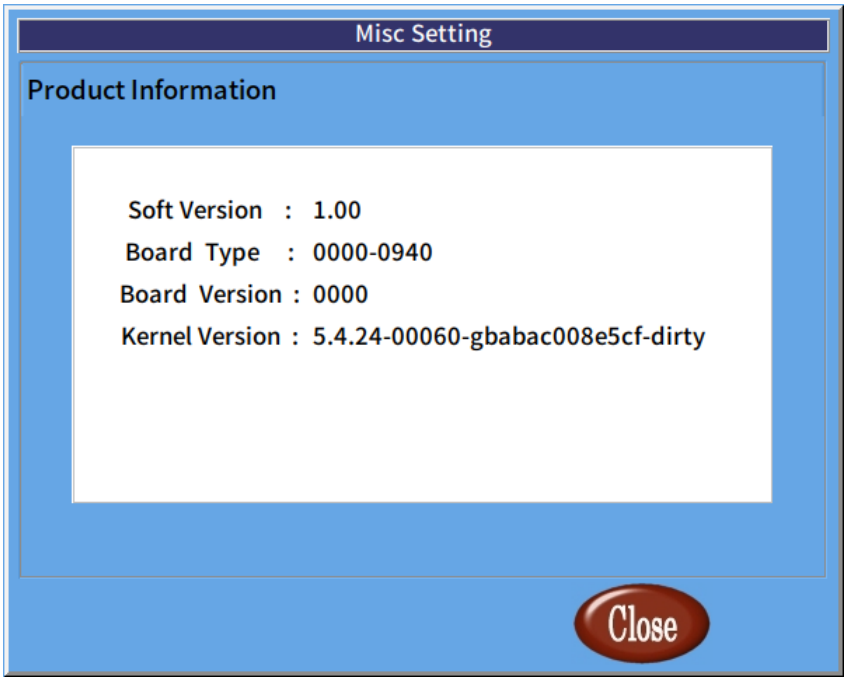


図 2-3-3-1. 製品情報

表 2-3-3-1. 製品情報

項目名	内容
Soft Version	実行環境の OS バージョン
Board Type	基板識別番号
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

２－３－４ 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。

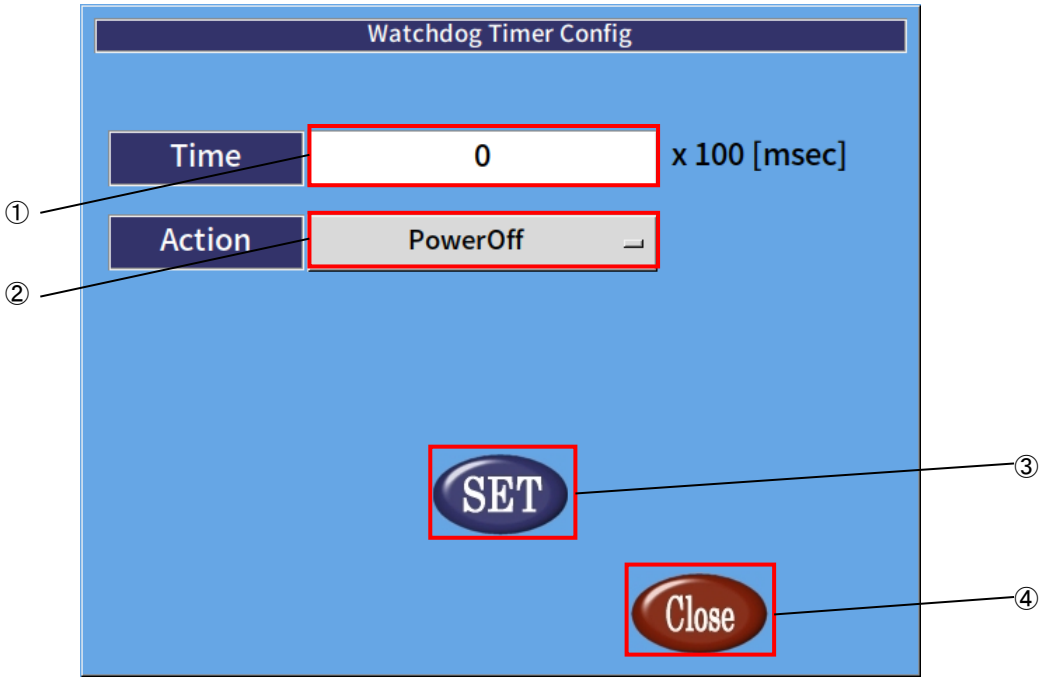


図 2-3-4-1. ハードウェア・ウォッチドッグタイマの設定

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 2-3-4-1 に示します。デフォルトでは PowerOff が選択されます。

表 2-3-4-1. 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
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。

２－３－５ 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。

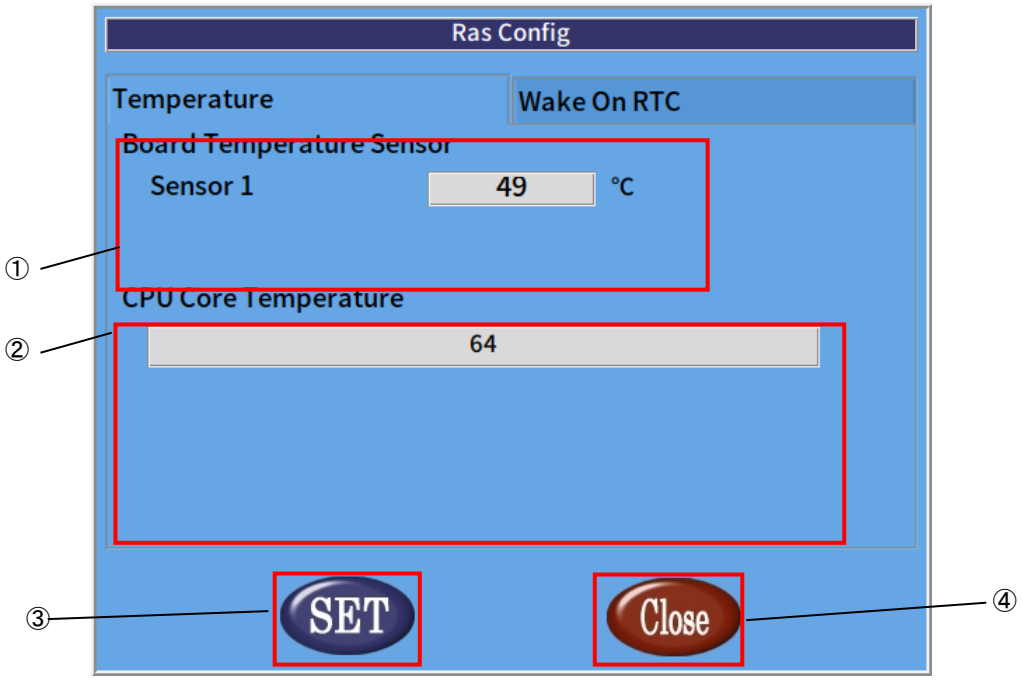


図 2-3-5-1. CPU 温度の表示

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。
※注：EC1G-01x/02x では、常に 0 が表示されます。

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。

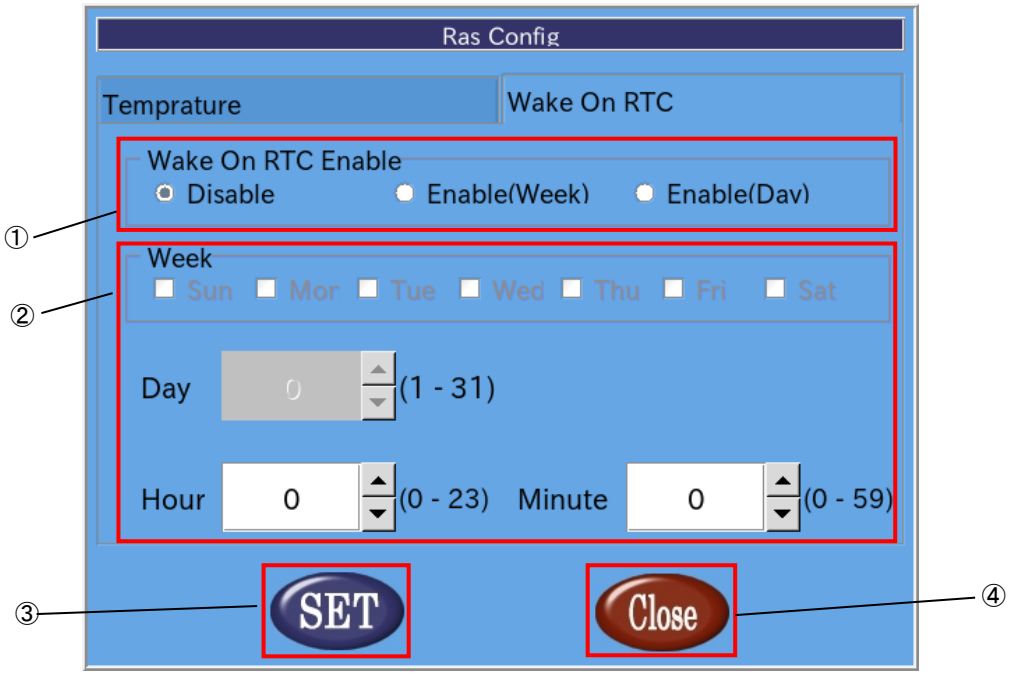


図 2-3-5-2. Wake On RTC の設定

①　Wake On RTC Enable

Wake On Rtc 機能の設定を行います。

表 2-3-5-1. 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。

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 2-3-5-2. 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)
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。

２－３－６ 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 2-3-6-1 に示します。

表 2-3-6-1. 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（左）を設定します。
Master_R	マスターボリューム（右）を設定します。
HeadPhone	ヘッドホン出力を設定します。 ミュートの有無のみ設定可能です。
PCM	PCM のボリュームを調整します。
Front	フロントスピーカボリュームを調整します。
Front Line	フロントラインボリュームを調整します。
Front Mic	フロントマイクボリュームを調整します。
Front Mic Boost	フロントマイクボリュームを調整します。 値を１上げるごとに 10【dB】上昇します。
Line	ラインボリュームを調整します。
Mic	マイクボリュームを調整します。
Mic Boost	マイクボリュームを調整します。 値を１上げるごとに 10【dB】上昇します。
Capture	録音時のボリュームを調整します。
Beep	ビープボリュームを調整します。
Speaker	スピーカ出力を設定します。

※注：EC1G-01x/02x では、Master_L、Master_R のみ設定が有効です。

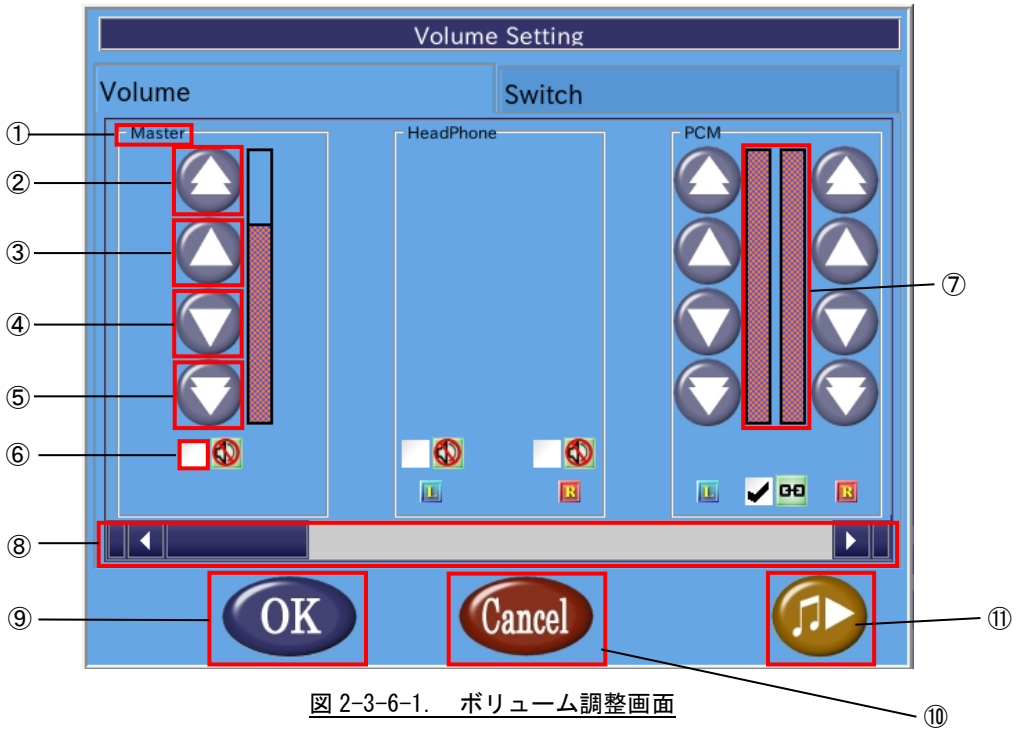


図 2-3-6-1.　ボリューム調整画面

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。

２－３－７　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。

※注：UPS 機能はオプションとなります。非搭載器では、設定は無効です。

●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。

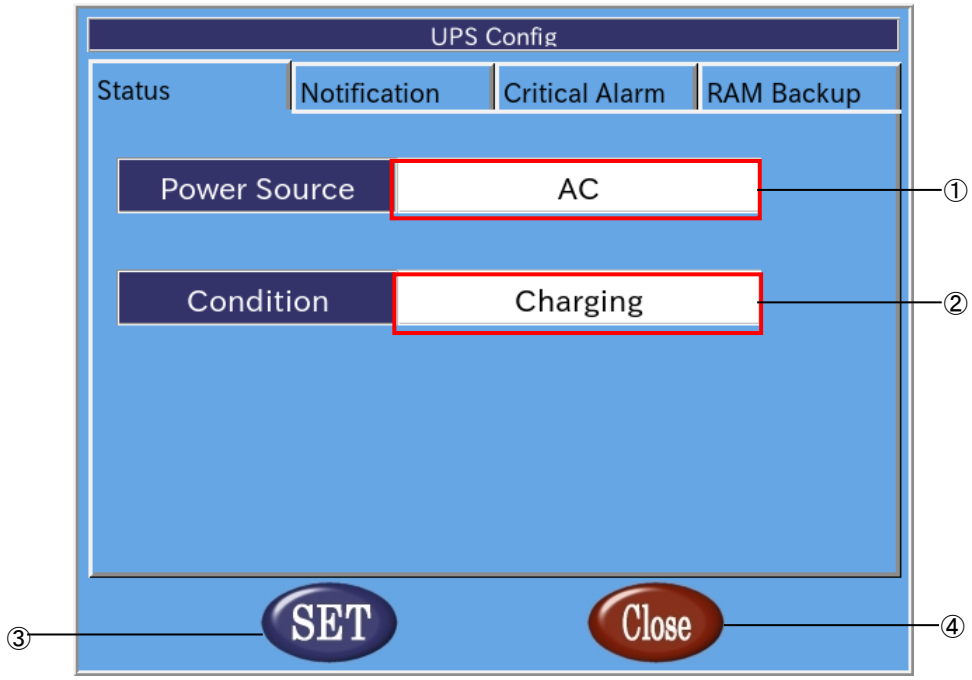


図 2-3-7-1. バッテリ状態の表示

①　電源種別

端末の電源種別を表示します。

表 2-3-7-1. 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 2-3-7-2. 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	バッテリに異常が生じています。

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。

●バッテリ異常通知設定

ASD UPS Config の[Notification]タブを選択することで、バッテリ異常が発生したときの通知について設定できます。

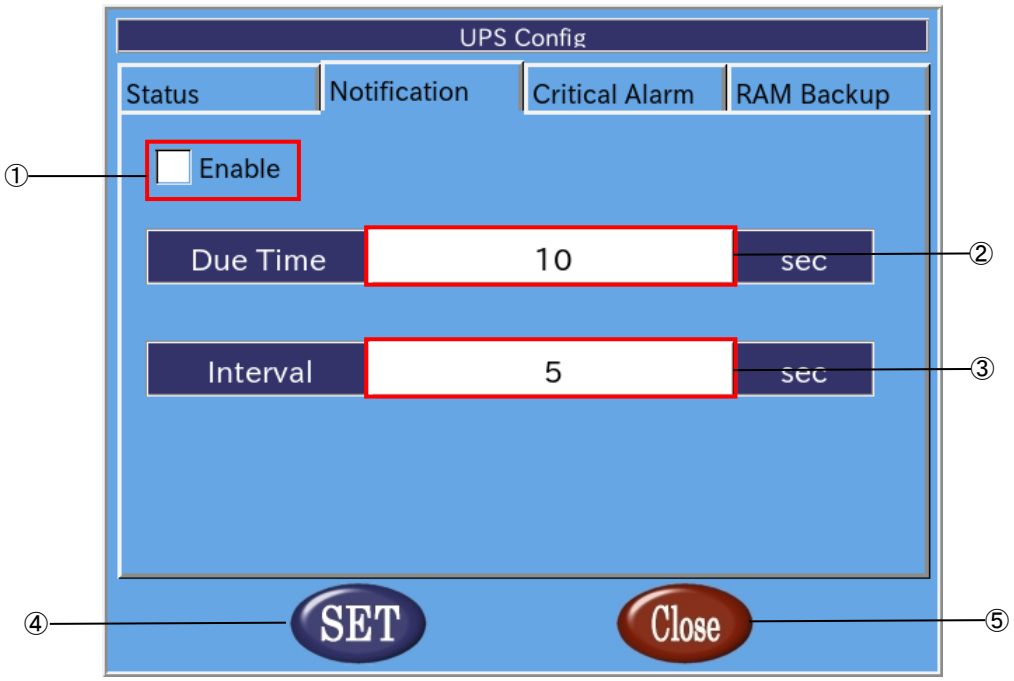


図 2-3-7-2. バッテリ異常通知設定

① バッテリ異常通知有効/無効設定
チェックボックスにチェックを入れることで、バッテリ異常発生時の通知が有効になります。

② 通知開始時間
バッテリ異常が発生してから、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。

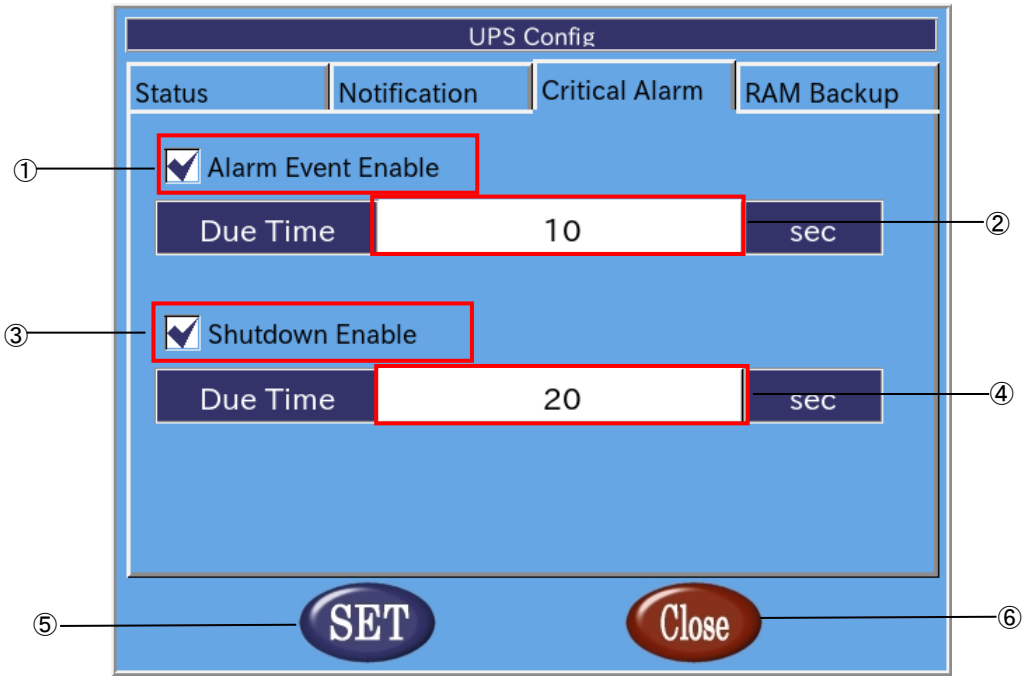


図 2-3-7-3. バッテリ警告設定

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を設定します（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。

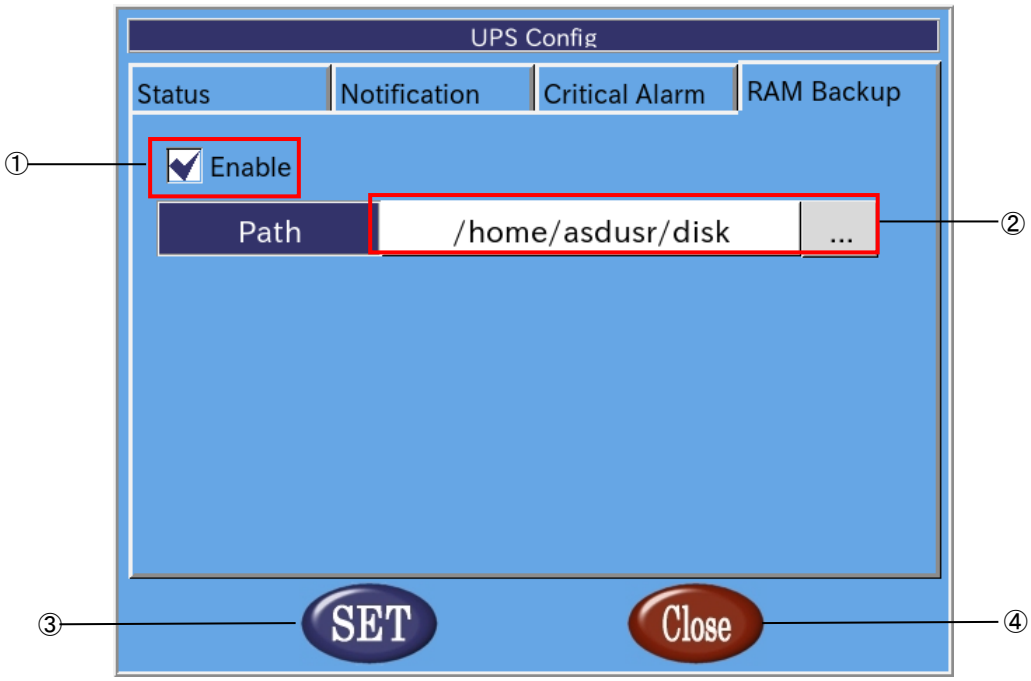


図 2-3-7-4. RAM バックアップ設定

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、『4-4-3 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。

２－３－８　ASD Shutdown Menu について

ASD Shutdown Menu によって、EC1G-01x/02x 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、メニュー画面から[Shutdown]を選択します。

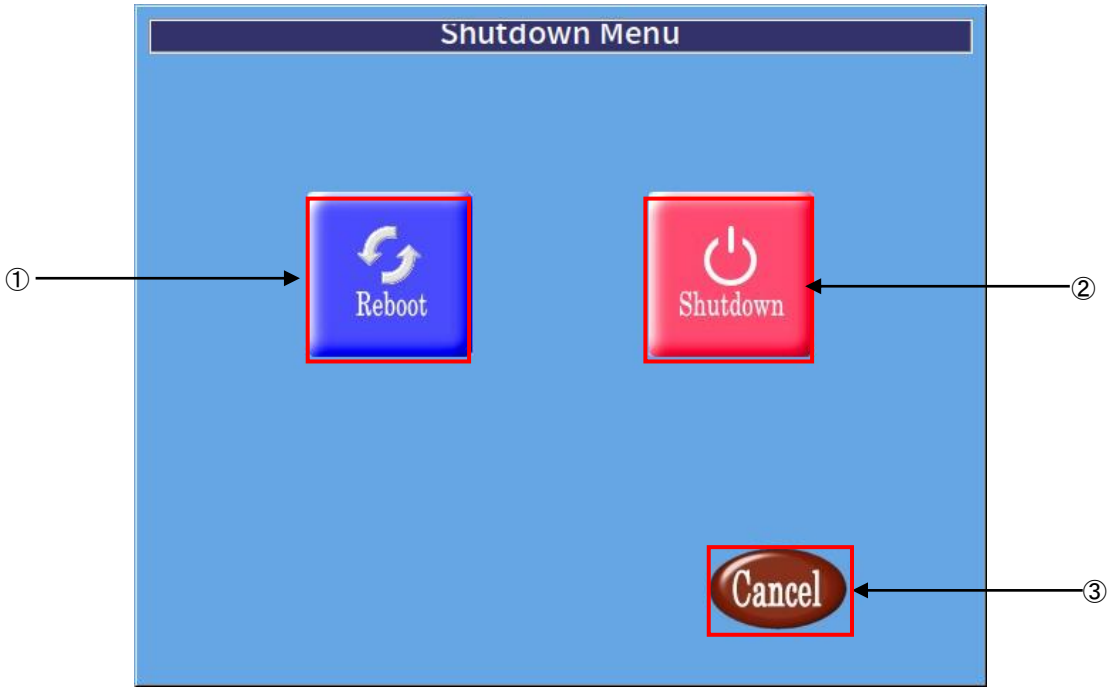


図 2-3-8-1. ASD Shutdown Menu

① 再起動
[Reboot]ボタンを押下することで、EC1G-01x/02x 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、EC1G-01x/02x 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

本節では、EC1G-01x/02x における有線 LAN の設定方法について説明します。

①　画面右上の「▼」をクリックし、「設定」を選択します。

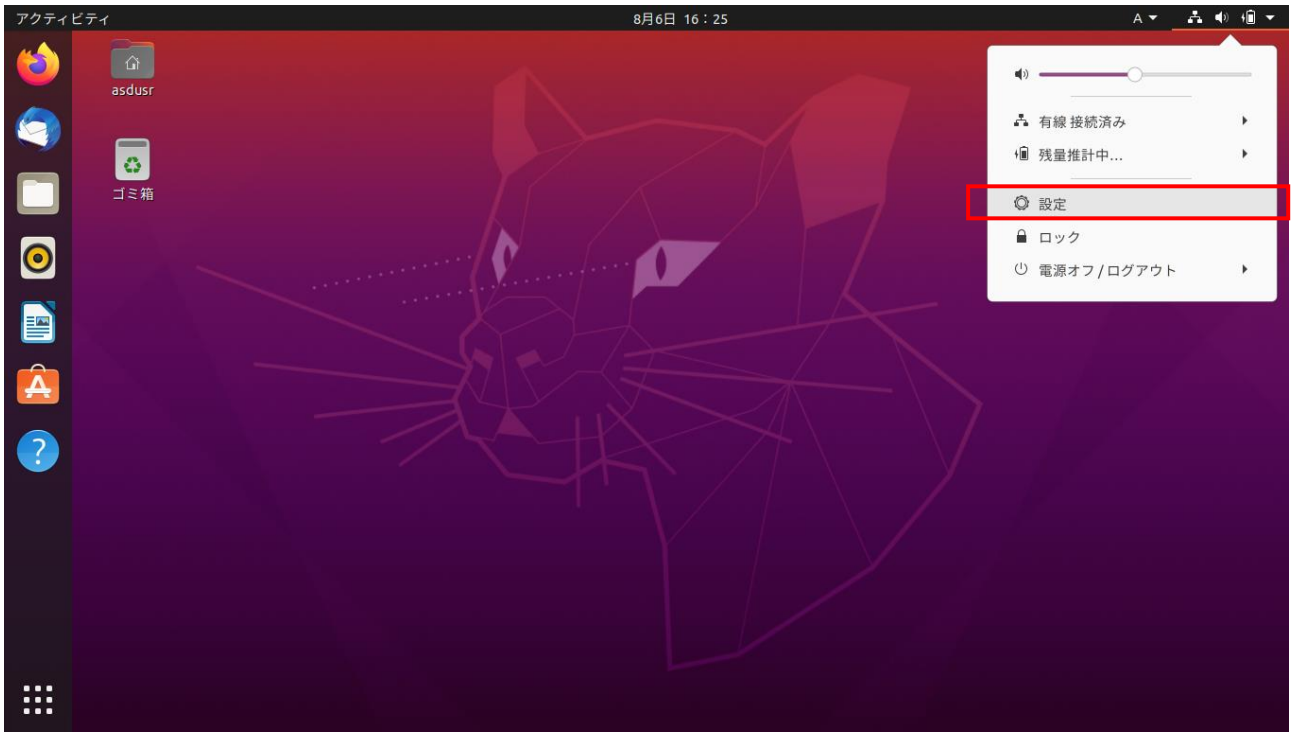


図 2-4-1. ネットワーク設定画面の起動

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 ボタンをクリックします。

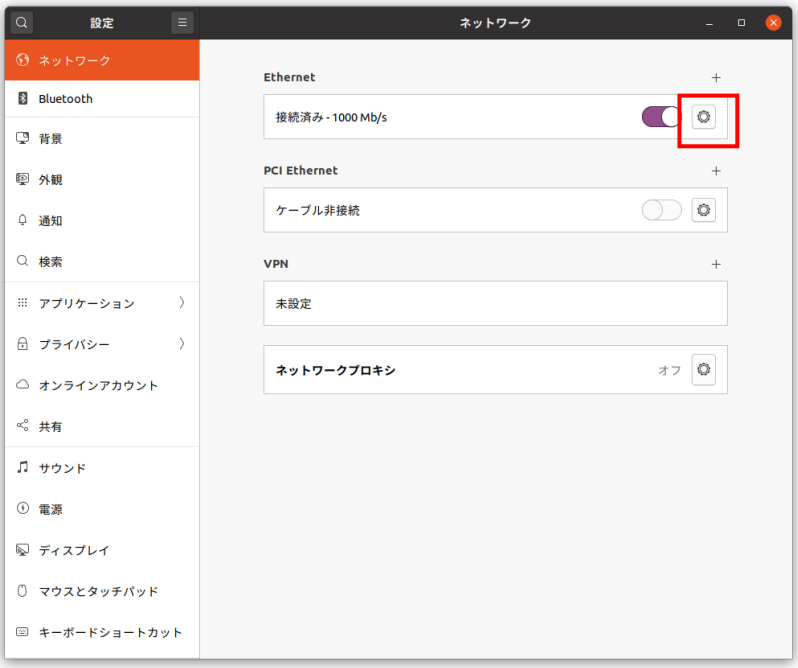


図 2-4-2. ネットワーク設定画面

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、メンテナンスされていないため、数年前に非推奨なパッケージになりました。
今日の Linux ディストリビューションでは、「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。その代わりに、[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

EC1G-01x/02x は、オプションとして無線 LAN アダプタを搭載可能です。
本節では、EC1G-01x/02x における無線 LAN の設定方法について説明します。

①　画面右上の「▼」をクリックし、「設定」を選択します。

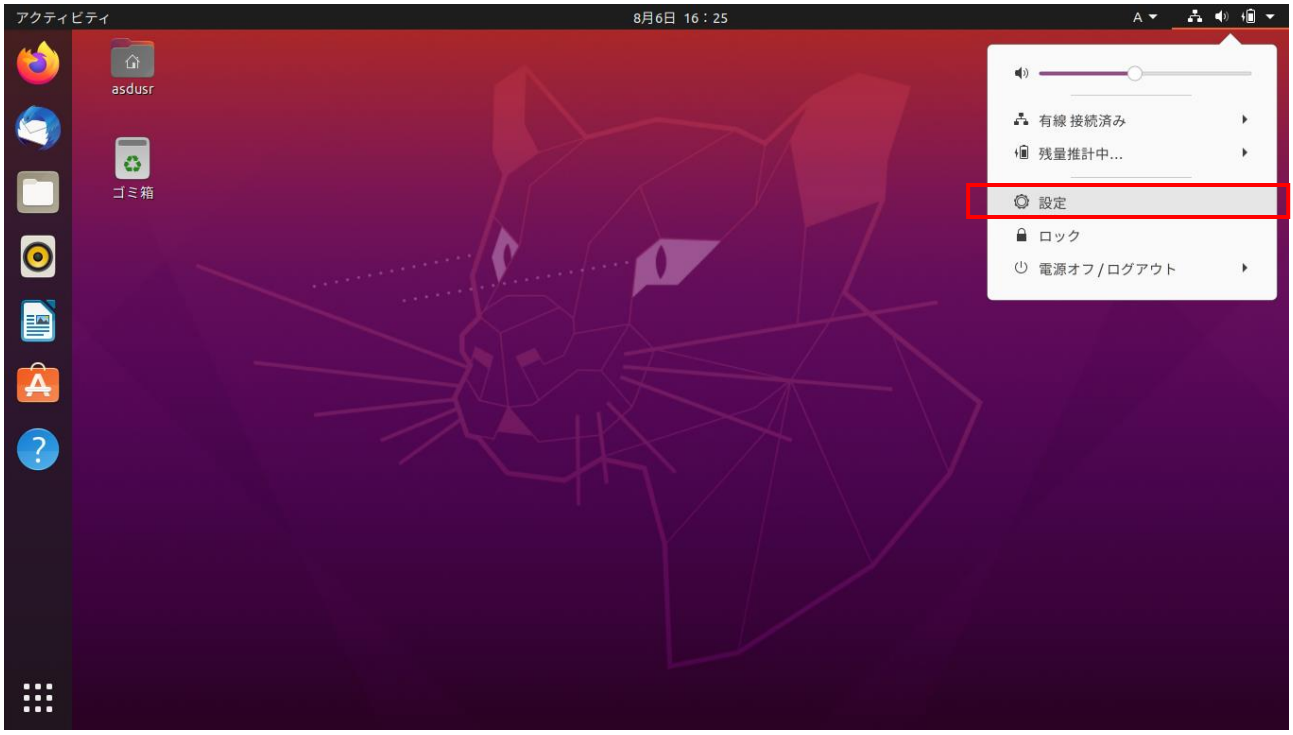


図 2-5-1. ネットワーク設定画面の起動

② 左ペインから「Wi-Fi」を選択することで、図 2-5-2 のようなネットワーク一覧が起動します。接続先をクリックしてください。クリック後、接続先の設定によっては、パスワード入力を求められることがあります。

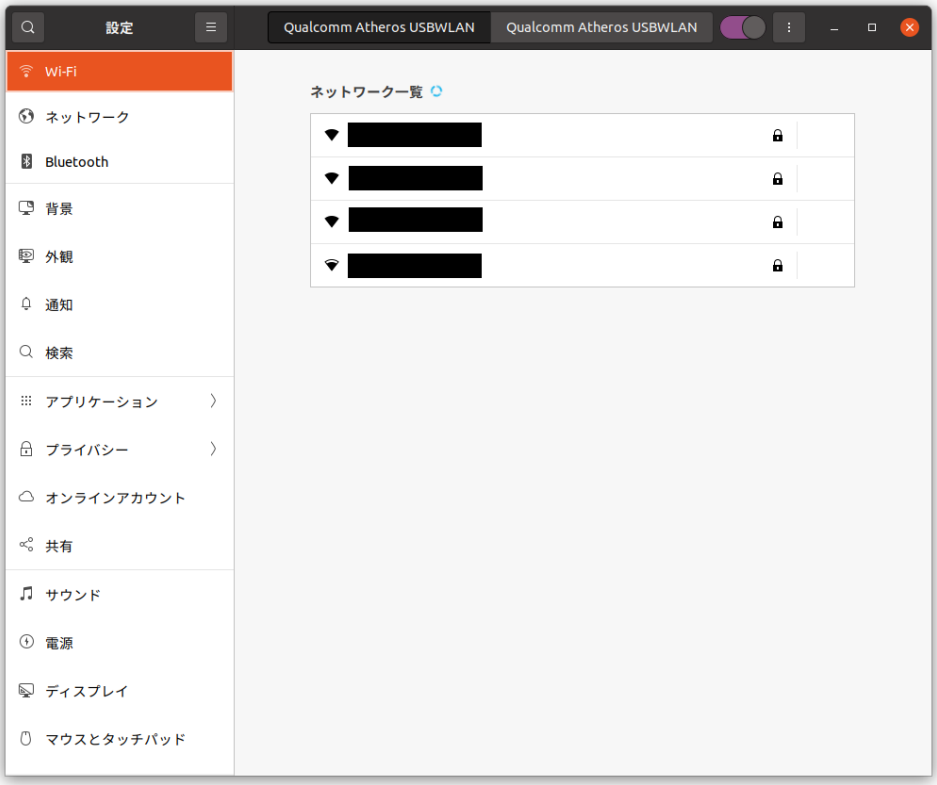


図 2-5-2. ネットワーク設定画面の起動

２－６　Bluetooth の設定について

EC1G-01x/02x は、オプションとして Bluetooth を搭載可能です。
本節では、Bluetooth による機器の接続方法について記述します

①　アプリケーション一覧から「Bluetooth マネージャー」を選択します。

図 2-6-1. Bluetooth マネージャーの起動

② Bluetooth マネージャーが起動します。
「検索」をクリックすることで、Bluetooth デバイス一覧が表示され、設定や接続を行うことができます。

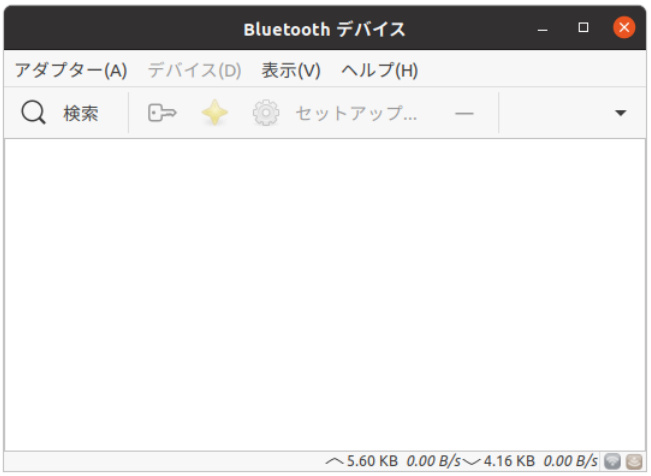


図 2-6-2. Bluetooth マネージャー

２－７　LTE の設定について

EC1G-01x/02x は、オプションとして LTE 機能を搭載可能です。
LTE 機能を使用して LTE 通信を行うことができます。
以下に LTE 通信の設定方法を示します。

① コンソールを起動させ、下記のコマンドを実行します。

```
$ sudo wvdialconf /etc/wvdial.conf
```

② OS を再起動してください。

③ 画面右上のメニューから「モバイルブロードバンド設定」を選択します。

図 2-7-1. ネットワーク設定画面の起動

④　図 2-7-2 のようなネットワーク設定画面が起動します。
図 2-7-2 の赤枠で囲った箇所を右にスライドし、モバイルブロードバンドを有効化します。

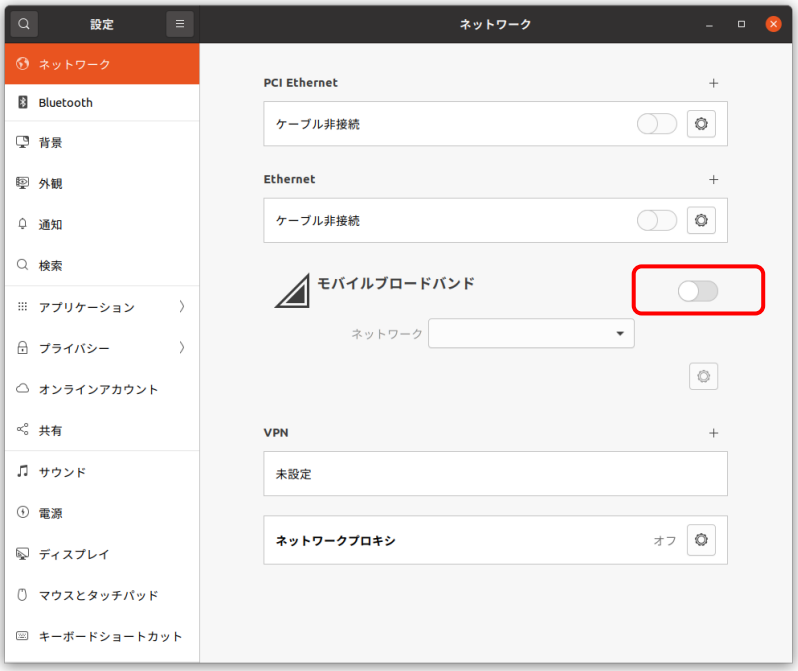


図 2-7-2.　ネットワーク設定画面

⑤　「ネットワーク」のコンボボックスから「新しい接続を追加」を選択します。

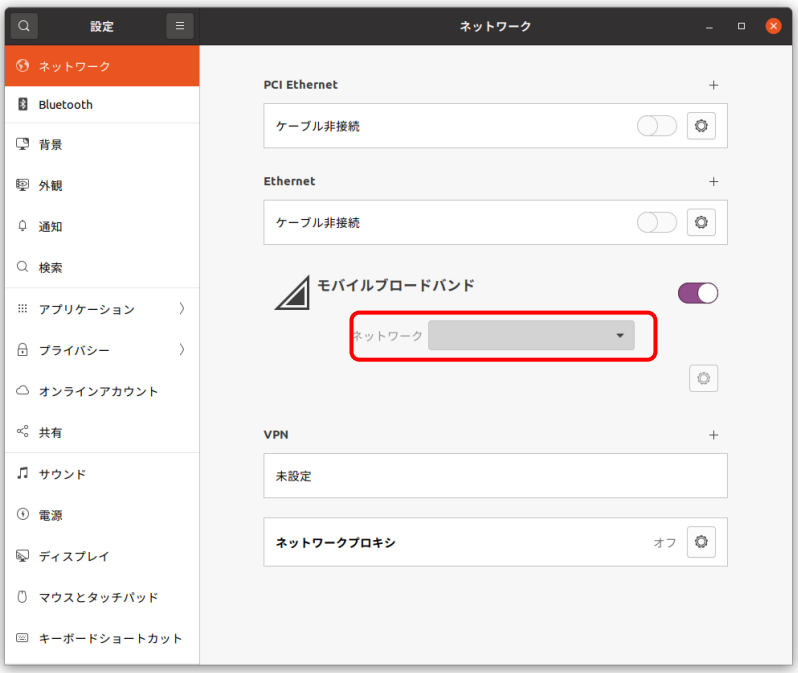


図 2-7-3.　ネットワーク設定画面

⑥　図 2-7-4 のようなダイアログが表示されます。
「進む」ボタンをクリックしてください。

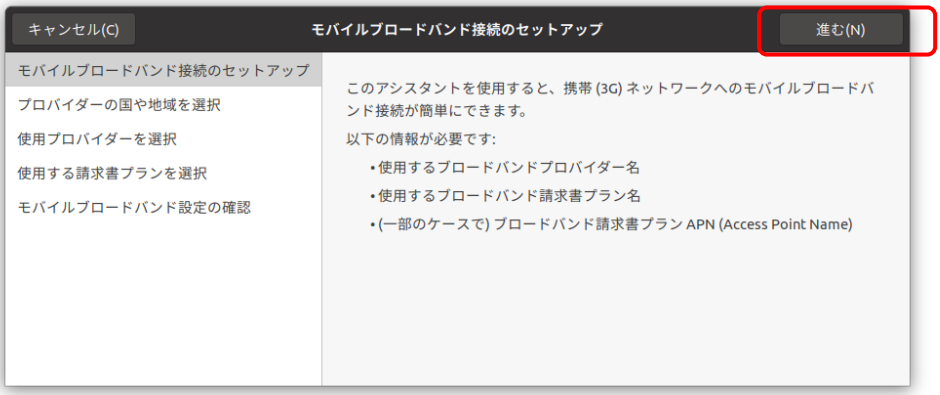


図 2-7-4.　モバイルブロードバンド接続のセットアップ

⑦　モバイルブロードバンド接続の設定画面が開きます。
ご使用の SIM カードの資料を参照し、設定をしてください。

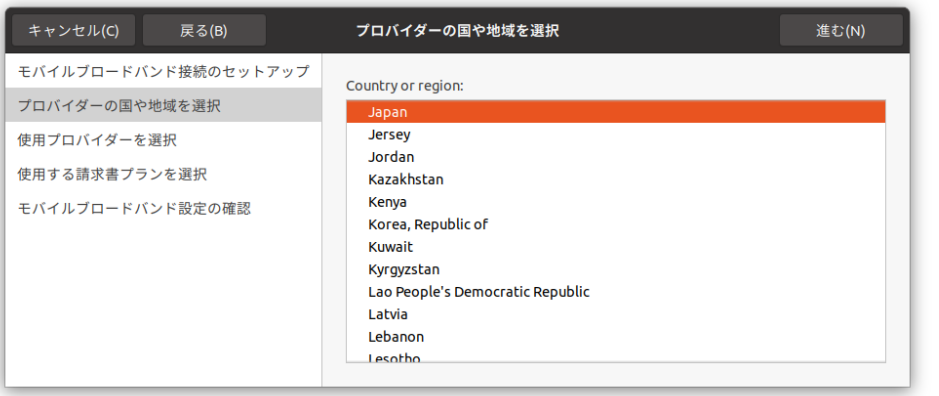


図 2-7-5.　接続のセットアップ

※　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。

⑧　ネットワーク設定画面に戻ります。

 ボタンをクリックすることで、通信設定の編集画面が開きます。

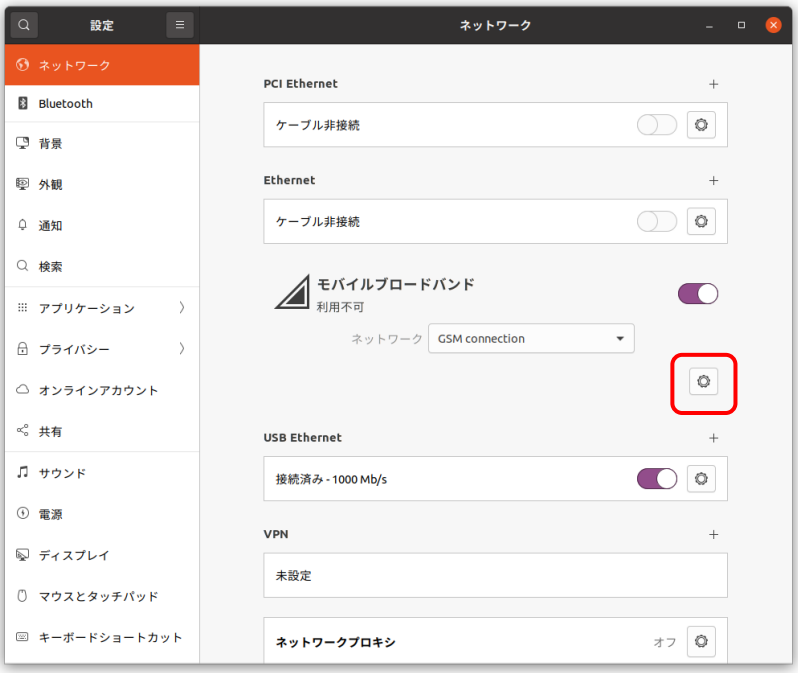


図 2-7-6.　ネットワーク設定

⑨　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 2-7-7.　ネットワーク設定の編集

２－８　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。

EC1G-01x/02x の固有デバイスとして以下のデバイス制御が可能です。

２－８－１ 温度センサ

基板上の温度センサの情報を取得します。

表 2-8-1-1. 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 Core 温度を取得します。

２－９　データの保護について

２－９－１ 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。

```
$ sudo 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。

```
$ sudo 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。

```
$ sudo mount –o remount,ro /mnt
```

２－９－２　ルートファイルシステムの保護について

EC1G-01x/02x は UPS を搭載していないため、停電や電源断によりファイルが破損する可能性があります。ファイルの破損を防ぐため、運用時は、ルートファイルシステムのリードオンリー化によるルートファイルシステムの保護をお勧めいたします。

以下に、ルートファイルシステムのリードオンリー化手順を示します。

① 起動時のマウントオプションを変更します。

```
$ sudo ln -sf /etc/fstab.readonly /etc/fstab
```

② 一部のディレクトリを書き込み可能に見せかけるための処理を行います。

```
$ sudo mv /home /home_org
$ sudo mkdir /home /home_rw
$ sudo mv /var /var_org
$ sudo mkdir /var /var_rw
```

③ 再起動します。

リードオンリー状態から非リードオンリー状態に戻すには、以下の手順を実行します。

① ルートファイルシステムを書き込み可能にします。

```
$ sudo mount -o rw,remount /
```

② 起動時のマウントオプションを変更します。

```
$ sudo ln -sf /etc/fstab.normal /etc/fstab
```

③ 書き込み可能に見せかけたディレクトリを復帰します。

```
$ sudo cp -a /home_org/* /home/
$ sudo cp -a /var_org/* /var/
```

２－１０ ソフトウエアキーボードについて

EC1G-01x/02x 実行環境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]→[すべて]→[Onboard]を選択します。

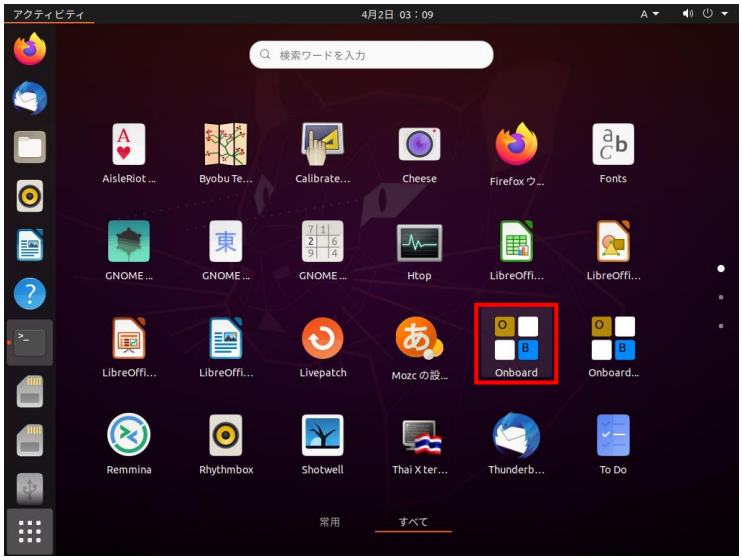


図 2-10-1.　ソフトウエアキーボードの起動

②　ソフトウエアキーボードが表示されます。

図 2-10-2.　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。

２－１１　アプリケーションの自動起動について

EC1G-01x/02x 実行環境で、デスクトップ起動時にアプリケーションを自動起動するには、以下の 2 通りの方法があります。ここでは、これらの方法について説明します。

① autostart.sh を編集する方法
② gnome-session-properties を使用する方法

２－１１－１ autostart.sh による自動起動

EC1G-01x/02x 実行環境は、デスクトップ起動時に、シェルスクリプトファイル「/home/asdusr/autostart.sh」を実行します。このファイルを編集することで、アプリケーションを自動起動させることができます。

例えば、AsdConfigMenu を自動起動させるときは、以下のように記述します。

```
#!/bin/sh

sudo /usr/bin/AsdConfigMenu &
```

※注:autostart.sh は一般ユーザー権限で実行されます。管理者権限が必要なコマンドを実行するには、visudo コマンドを使い、パスワード不要に設定したうえで、sudo コマンドを使って実行してください。

２－１１－２ gnome-session-properties を使用する方法

gnome-session-properties は gnome 標準の自動起動管理ツールです。以下に、このツールを使ってアプリケーションを自動起動させる方法を記します。

① コンソール上で「gnome-screen-properties」と入力するか、アプリケーション一覧で「session」と入力し、「自動起動するアプリケーション」を選択することで、gnome-session-properties が起動します。

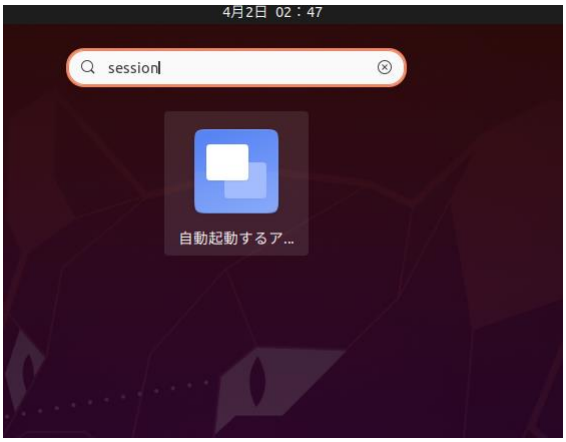


図 2-11-2-1. gnome-session-properties の起動

②　gnome-session-properties が起動します。「追加」をクリックします。

図 2-11-2-2. gnome-session-properties

③　「自動起動するプログラムの追加」ダイアログが表示されるので、名前、コマンド、説明を入力します。

図 2-11-2-3. gnome-session-properties 追加ダイアログ

第３章　開発環境

EC1G-01x/02x 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、「EC1G-01x/02x 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。

EC1G-01x/02x 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）

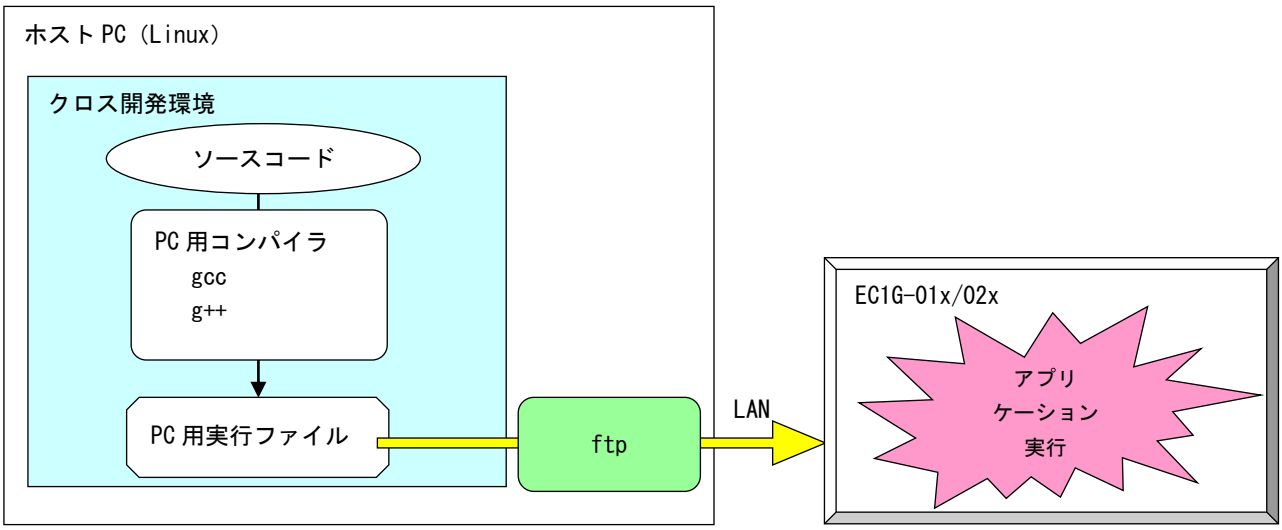


図 3-1-1. クロス開発方式イメージ図

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

EC1G-01x/02x 開発環境は、Ubuntu 20.04 LTS ディストリビューションイメージに、ARM 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 EC1G-01x/02x 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　デバイスについて

本章では、EC1G-01x/02x 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。

EC1G-01x/02x 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『EC1G-01x/02x 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 EC1G-01x/02x 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_Serial	シリアルポート制御方法	『4-1 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	『4-2 ネットワークポート』を参照
sample_HwWdtKeepAlive	ハードウェア・ウォッチドッグタイマ操作	『4-3 RAS 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、EC1G-01x/02x に搭載されたデバイスの仕様について説明します。

EC1G-02x の外観図を示します。また、各部名称を表 4-3 に示します。

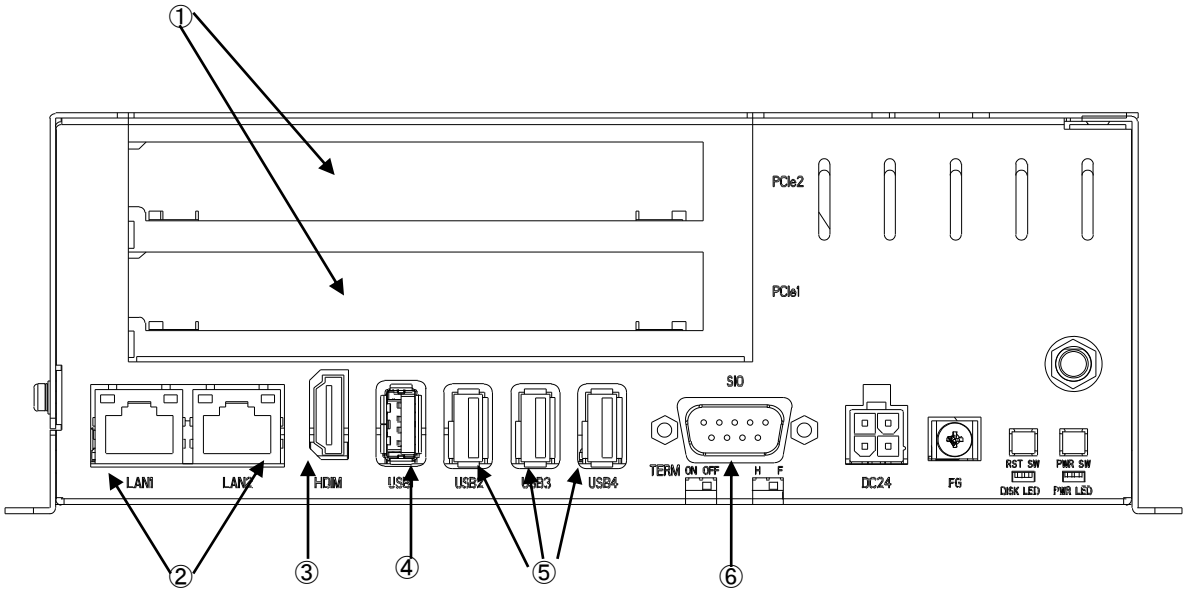


図 4-1. EC1G-02x 外形図

表 4-3. 各部名称

No.	名称	機能	説明	個数
①	PCI Express 拡張スロット	PCI Express Gen2	PCI Express 拡張ボードを接続できます。**※注**	2
②	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
③	HDMI インターフェース	グラフィック サウンド	外部モニタに画面、音声を出力できます。	1
④	USB3.0 インターフェース	USB3.0 ポート	USB1.1/2.0/3.0 の機器を接続することができます。	1
⑤	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。	3
⑥	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、RS-422、RS485 の通信を行うことができます。 SIO1: ttyUSB0	1

※注：　PCI Express 拡張スロットに接続する PCI Express 拡張ボードのドライバ、ライブラリは、各自でご用意ください。

４－１　シリアルポート

４－１－１　シリアルポートについて

EC1G-01x/02x には、RS232C/RS422/RS485 を切り替えることができるシリアルポートが 1 ポーあり、ユーザアプリケーションで使用できます。
表 4-1-1-1 にデバイスファイル名を示します。

表 4-1-1-1. シリアル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

RS232C、RS422、RS485 のデバイスファイルをオープンし、Read／Write することで、アプリケーションからシリアルポートを使用することができます。
シリアルポートの通信タイプを切り替えるには、シリアル通信タイプ切り替え関数を使用します。シリアル通信タイプ切り替え関数は、ライブラリ経由で使用できます。

※注：　RS422、RS485 使用時は、終端抵抗スイッチと全/半二重設定スイッチも切り替える必要があります。詳細は、ハードウェアマニュアルを参照してください。

４－１－２　シリアルポート設定について

●termios について
Linux では「termios」と呼ばれる、非同期通信ポートを制御する為の汎用ターミナルインターフェースがあります。このインターフェースを使用することで、シリアルポートのボーレートや、CTS/RTS の有効無効等、さまざまな設定が可能となります。「termios」についての詳細はインターネットや書籍等を参照してください。

●シリアル通信タイプ切り替え関数について
シリアル通信タイプ切り替え関数を使用することにより、シリアルポートの通信タイプ、RS485 の送信イネーブル時間を設定、取得できます。
以下に、シリアルタイプ通信タイプ切り替え関数の詳細を示します。

asd_scictl_config_set 関数

機能　シリアル通信設定を取得します。

書式　int asd_scictl_config_set(unsigned short ch, unsigned short type, unsigned short timer)

引数
- ch　：対象となるシリアルポート
  EC1G-01x/02x は、1 のみ指定可能です。
- type　：シリアル通信タイプ
  - 0：RS232C
  - 1：RS422
  - 2：RS485
- timer　：送信イネーブル時間[μsec]

戻り値　エラーコード(0:正常, -1:異常)

説明　現在のシリアル通信タイプ、送信イネーブル時間設定を取得します。

asd_scictl_config_get 関数

機能　シリアル通信タイプ、送信イネーブル時間を設定します。

書式　int asd_ asd_scictl_config_get(unsigned short ch, unsigned short* type, unsigned short* timer)

引数
- ch　：対象となるシリアルポート
  EC1G-01x/02x は、1 のみ指定可能です。
- type　：取得したシリアル通信タイプの格納先
  - 0：RS232C
  - 1：RS422
  - 2：RS485
- timer　：取得した送信イネーブル時間の格納先[μsec]

戻り値　エラーコード(0:正常, -1:異常)

説明　現在のシリアル通信タイプ、送信イネーブル時間を取得します。

４－１－３　シリアルポートサンプルプログラム

●Console 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 4-1-3-1 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 4-1-3-1. 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);
}
```

「/usr/local/tools-arm64/samples/sampleConsol/sample_SerialPortChange」に、シリアル通信タイプを設定、取得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のソースコードをリスト 4-1-3-2 に示します。

asd_scictl_config_set 関数で CH1 の通信タイプを RS485、送信イネーブル時間を 1000[μsec]に設定しています。設定後、asd_scictl_config_get 関数で CH1 の通信タイプを取得し、コンソールに表示します。

リスト 4-1-3-2. シリアル通信タイプ設定、取得を行うソースコード（main.cpp）

```
#include <stdio.h>

#include "asd_misclib.h"

int main(void)
{
    int ret;
    unsigned short ch = SCICTL_CH_1;
    unsigned short set_type = SCICTL_TYPE_RS485;
    unsigned short set_timer = 1000;
    unsigned short get_type;
    unsigned short get_timer;

    ret = asd_scictl_config_set(ch, set_type, set_timer);
    if (ret != ASDMISC_ER_OK) {
        fprintf(stderr, "scictl config set failed¥n");
        return -1;
    }
    ret = asd_scictl_config_get(ch, &get_type, &get_timer);
    if (ret != ASDMISC_ER_OK) {
        fprintf(stderr, "scictl config get failed¥n");
        return -1;
    }
    printf("scictl ch=%d type=%d timer=%d¥n",
        ch, get_type, get_timer);

    return 0;
}
```

４－２　ネットワークポート

４－２－１　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。

４－２－２　ネットワークソケット用システムコールについて

表 4-2-2-1 にサーバ側のソケットシステムコール、表 4-2-2-2 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 4-2-2-3 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 4-2-2-1. 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 4-2-2-2. 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 4-2-2-3. 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。

４－２－３　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 EC1G-01x/02x 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 4-2-3-1 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 4-2-3-1. 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 4-2-2-2 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 4-2-3-2 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 4-2-3-2. 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);
}
```

４－３　　RAS 機能

RAS 機能として以下の様な機能を実装しています。

- 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。

４－３－１　　ハードウェア・ウォッチドッグタイマ機能について

EC1G-01x/02x 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。図 4-3-1-1 に、デバイス、ドライバ、アプリケーション、設定ツールの関係を示します。

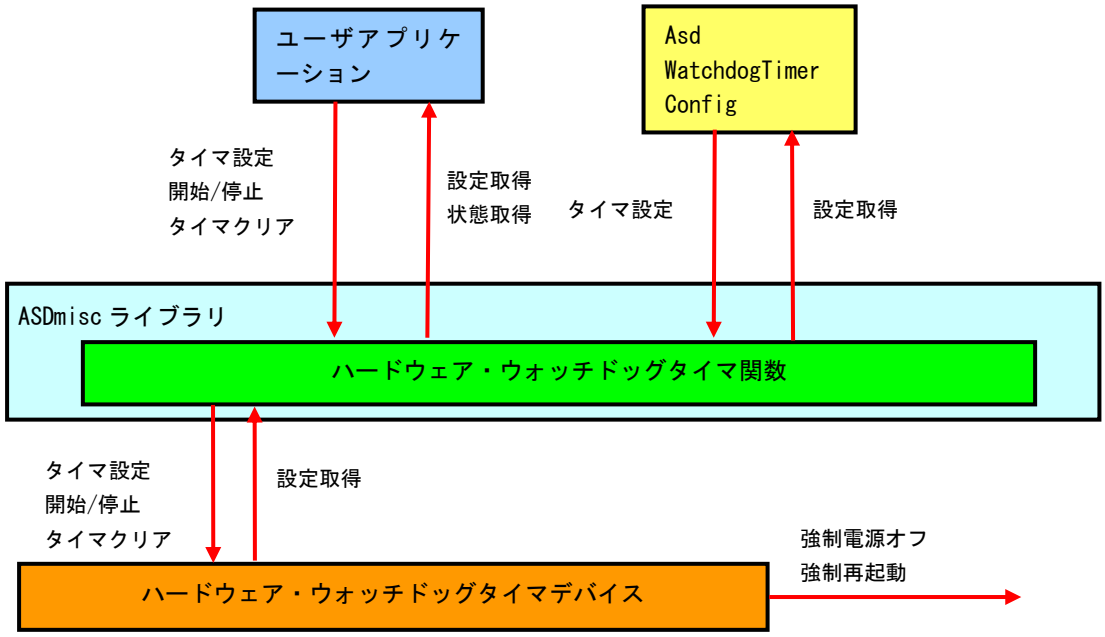


図 4-3-1-1. ハードウェア・ウォッチドッグタイマの構成

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制御関数の詳細については、『システムコントロールユニット ライブラリリファレンスマニュアル』を参照してください。

４－３－２　　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 4-3-2-1 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 4-3-2-1. 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 <signal.h>


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

static RASHWWDT_CONFIG Config;
static volatile int fStart = 0;
static volatile int fStop = 0;
static volatile int fFinish = 0;
static pthread_t Thread_Id;

volatile sig_atomic_t e_flag = 0;

void abrt_handler(int sig);

//---------------------------------------------------------------------------
/*
 * 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(e_flag == 0){
        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;

    wdt = -1;
    action = -1;

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


    while((c = getopt(argc, argv, "t:a:")) != -1){
        switch(c){
            case 't':
                wdt = atoi(optarg);
                break;
            case 'a':
                action = atoi(optarg);
                break;
        }
    }
    if((wdt <= 0) || (action < 0) || (action > RASHWWDT_RESET)){
        printf("  t:test time[ *100msec]¥n");
        printf("  a:action 0:PowerOff 1:Reset¥n");
        return -1;
    }

    Config.action = action;
    Config.time = wdt;

    CreateThread();

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

４－３－３ バックアップ RAM 機能について

EC1G-01x/02x では 4MByte のバックアップ機能付きの 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、『2-3-7 ASD UPS Config について』を参照してください。

４－３－４ 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_sram0) 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。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 4-3-4-1 にデバイスの詳細を示します。

表 4-3-4-1. 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_sram0)を**open**関数でオープンします。 rasramfd = open("/dev/asd_sram0", O_RDWR);
READ
read関数を用いてバックアップRAMの値を読込みます。
WRITE
write関数を用いてバックアップRAMに値を書込みます。
MMAP
mmap関数を用いてバックアップ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); ● **ctlcode** **SRAM_IOCGSIZE** 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

４－３－５ 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-0010/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 4-3-5-1 にソースコードを示します。

リスト 4-3-5-1. バックアップ RAM 制御 Read/Write 方式 (rasram_read.c)

```
#include <stdio.h>
#include <stdlib.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 "asd_rasram.h"

unsigned long rasram_size = 0;

unsigned char *rasram;

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

    desc = open("/dev/asd_ssram0", O_RDWR);
    if(desc < 0){
        printf("open faild: err=%d¥n", errno);
        return -1;
    }

    if(ioctl(desc, SRAM_IOCGSIZE, &rasram_size) != 0){
        close(desc);
        printf("size check faild.¥n");
        return -1;
    }
    printf("rasram size is %lx.¥n", rasram_size);
    rasram = malloc(rasram_size * sizeof(unsigned char));

    /* inc data */
    d = 1;
    dp = rasram;
    for(i = 0; i < rasram_size; i++){
```

```
        *dp++ = d++;
    }
    lseek(desc, 0, SEEK_SET);
    len = write(desc, rasram, rasram_size);
    if(len != rasram_size){
                printf("inc data write faild: err=%d¥n", errno);
                close(desc);
        free(rasram);
        return -1;
    }
    memset(rasram, 0x00, rasram_size);
    lseek(desc, 0, SEEK_SET);
    len = read(desc, rasram, rasram_size);
    if(len != rasram_size){
                printf("dec data read faild: err=%d¥n", errno);
                close(desc);
        free(rasram);
        return -1;
    }
    d = 1;
    dp = rasram;
    for(i = 0; i < rasram_size; i++){
        if(*dp++ != d++){
                    printf("inc data compare faild¥n");
            close(desc);
            free(rasram);
            return -1;
        }
    }

    /* dec data */
    d = 0xff;
    dp = rasram;
    for(i = 0; i < rasram_size; i++){
        *dp++ = d--;
    }
    lseek(desc, 0, SEEK_SET);
    len = write(desc, rasram, rasram_size);
    if(len != rasram_size){
                printf("dec write faild: err=%d¥n", errno);
                close(desc);
        free(rasram);
        return -1;
    }
    memset(rasram, 0x00, rasram_size);
    lseek(desc, 0, SEEK_SET);
    len = read(desc, rasram, rasram_size);
    if(len != rasram_size){
                printf("dec read faild: err=%d¥n", errno);
                close(desc);
        free(rasram);
        return -1;
```

```
    }
    d = 0xff;
    dp = rasram;
    for(i = 0; i < rasram_size; i++){
        if(*dp++ != d--){
                    printf("dec data compare faild¥n");
            close(desc);
            free(rasram);
            return -1;
        }
    }

    close(desc);
    printf("read/write check ok¥n");

    free(rasram);
    return 0;
}
```

「/usr/local/tools-0010/samples/sampleConsole/sample_SRAM」に、バックアップ付き RAM を mmap 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。mmap 関数でマッピングしたアドレスに対して、バイト、ワード、ロングデータでの読み書きを行い、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 4-3-5-2 にソースコードを示します。

リスト 4-3-5-2. バックアップ SRAM 制御 mmap 方式（rasram_mmap.c）

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

#include "asd_rasram.h"

unsigned long rasram_size = 0;

int main(void)
{
    int desc;
    int i;
    void *memmap = NULL;

    unsigned char b_dt;
    volatile unsigned char *b_mem;
    unsigned short w_dt;
    volatile unsigned short *w_mem;
    unsigned long l_dt;
```

```
volatile unsigned long *l_mem;

desc = open("/dev/asd_sram0", O_RDWR);
if(desc < 0){
    printf("open faild: err=%d¥n", errno);
    return -1;
}

if(ioctl(desc, SRAM_IOCGSIZE, &rasram_size) != 0){
    close(desc);
    printf("size check faild.¥n");
    return -1;
}

printf("rasram size is %lx.¥n", rasram_size);

memmap = mmap(0, rasram_size, PROT_READ | PROT_WRITE, MAP_SHARED, desc, 0);
if (!memmap) {
    printf("memmap mmap faild¥n");
            close(desc);
    return -1;
}
fprintf(stderr, "MemMap mmap: %08lx¥n", (unsigned long)memmap);

/* byte check1 */
b_dt = 1;
b_mem = (volatile unsigned char *)memmap;
for(i = 0; i < rasram_size; i++){
    *b_mem++ = b_dt++;
}
b_dt = 1;
b_mem = (volatile unsigned char *)memmap;
for(i = 0; i < rasram_size; i++){
    if(*b_mem++ != b_dt++){
            printf("byte check1 compare faild¥n");
                close(desc);
        return -1;
    }
}
printf("byte check1 ok¥n");

/* byte check2 */
b_dt = 0xff;
b_mem = (volatile unsigned char *)memmap;
for(i = 0; i < rasram_size; i++){
    *b_mem++ = b_dt--;
}
b_dt = 0xff;
b_mem = (volatile unsigned char *)memmap;
for(i = 0; i < rasram_size; i++){
    if(*b_mem++ != b_dt--){
            printf("byte check2 compare faild¥n");
```

```
                close(desc);
            return -1;
        }
    }
    printf("byte check2 ok¥n");

    /* word check1 */
    w_dt = 1;
    w_mem = (volatile unsigned short *)memmap;
    for(i = 0; i < rasram_size/2; i++){
        *w_mem++ = w_dt++;
    }
    w_dt = 1;
    w_mem = (volatile unsigned short *)memmap;
    for(i = 0; i < rasram_size/2; i++){
        if(*w_mem++ != w_dt++){
                printf("word check1 compare faild¥n");
                close(desc);
            return -1;
        }
    }
    printf("word check1 ok¥n");

    /* word check2 */
    w_dt = 0xffff;
    w_mem = (volatile unsigned short *)memmap;
    for(i = 0; i < rasram_size/2; i++){
        *w_mem++ = w_dt--;
    }
    w_dt = 0xffff;
    w_mem = (volatile unsigned short *)memmap;
    for(i = 0; i < rasram_size/2; i++){
        if(*w_mem++ != w_dt--){
                printf("word check2 compare faild¥n");
                close(desc);
            return -1;
        }
    }
    printf("word check2 ok¥n");

    /* long check1 */
    l_dt = 1;
    l_mem = (volatile unsigned long *)memmap;
    for(i = 0; i < rasram_size/4; i++){
        *l_mem++ = l_dt++;
    }
    l_dt = 1;
    l_mem = (volatile unsigned long *)memmap;
    for(i = 0; i < rasram_size/4; i++){
        if(*l_mem++ != l_dt++){
                printf("long check1 compare faild¥n");
                close(desc);
```

```
            return -1;
        }
    }
    printf("long check1 ok¥n");

    /* long check2 */
    l_dt = 0xffffffff;
    l_mem = (volatile unsigned long *)memmap;
    for(i = 0; i < rasram_size/4; i++){
        *l_mem++ = l_dt--;
    }
    l_dt = 0xffffffff;
    l_mem = (volatile unsigned long *)memmap;
    for(i = 0; i < rasram_size/4; i++){
        if(*l_mem++ != l_dt--){
                printf("long check2 compare faild¥n");
                    close(desc);
            return -1;
        }
    }
    printf("long check2 ok¥n");

    close(desc);
    printf("ioctl check finish¥n");
    return 0;
}
```

４－４　　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。

４－４－１　　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として通常のディスクと同様に操作することができます。
EC1G-01x/02x 実行環境の初期設定では、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

```
$ sudo 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

```
$ sudo mount /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『4-4-2 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。

４－４－２　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 4-4-2-1 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。

EC1G-01x/02x での認識優先度を表 4-4-2-1 に示します。

表 4-4-2-1. 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 (/dev/mmcblk0p*)
②	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:EMMC ストレージのみ挿入して OS 起動したとき

挿入された USB メモリは挿入された順に sda、sdb･･･として認識されます。

表 4-4-2-2. 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･･･

４－４－３　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、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 4-4-3-1. 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

４－５　　UPS 機能

４－５－１ UPS 機能について

EC1G-01x/02x には、UPS 機能がオプション搭載されています。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、電源異常の通知、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。EC1G-01x/02x 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	電源異常通知
/ups_alarm	電源断警告

電源異常通知、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、電源異常/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、『2-3-7 ASD UPS Config について』を参照してください。

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-0010/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 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;
}
```

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。

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

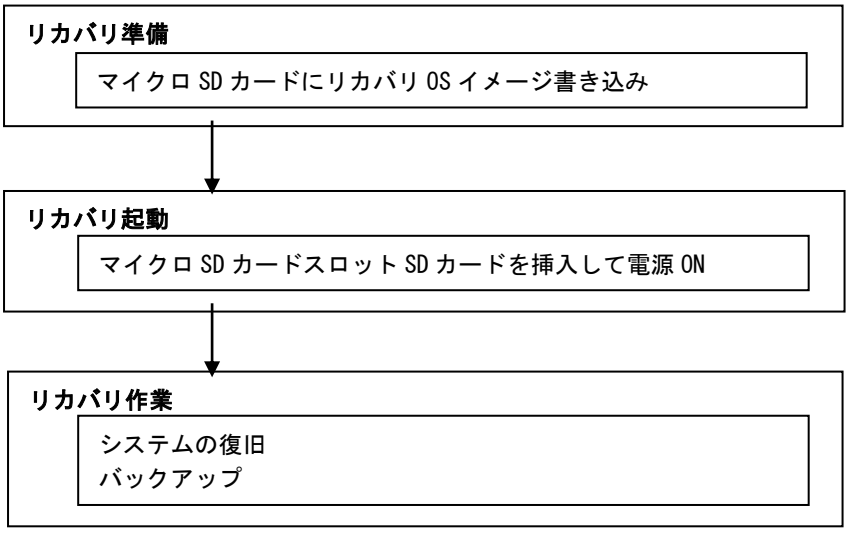


図 5-1-1. リカバリ(マイクロ SD カード利用)の流れ

５－１－１ リカバリ準備

リカバリを実行する前に、リカバリシステムを起動するためのマイクロ SD カードを作成します。
16GByte 以上のサイズのマイクロ SD カードをあらかじめ用意してください。

※注：　ここで用意したマイクロ SD カードの中身は全て消去されます。
あらかじめバックアップをとるなどしておいてください。

● マイクロ SD カード作成手順

① Windows が動作する PC にリカバリ DVD を挿入します。
② 用意したマイクロ SD カードを、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展開します。
［リカバリ DVD]¥Recovery¥RecoveryImage.zip
PC 上の任意の場所に展開してください。
展開が完了すると「RecoveryImage.ddi」というファイルが作成されます。
④ リカバリ DVD の以下のファイルを実行します。
[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。

⑤ DD for Windows というツールが起動します。
⑥ 「対象ディスク」の項目に手順②で接続したマイクロ SD カード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マイクロ SD カードを選択してください。

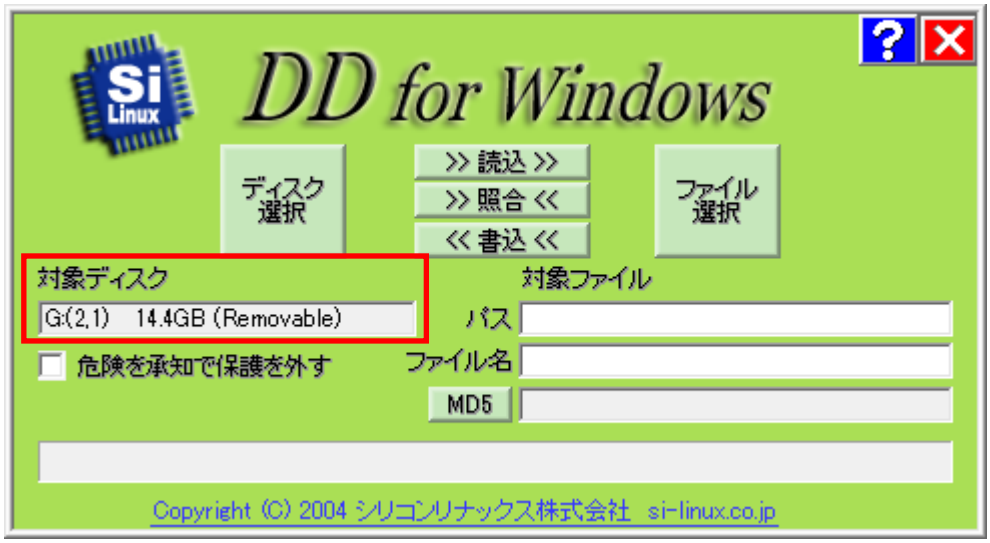


図 5-1-1-1. DD for Windows

⑦　「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で展開した「RecoveryImage.ddi」を選択してください。

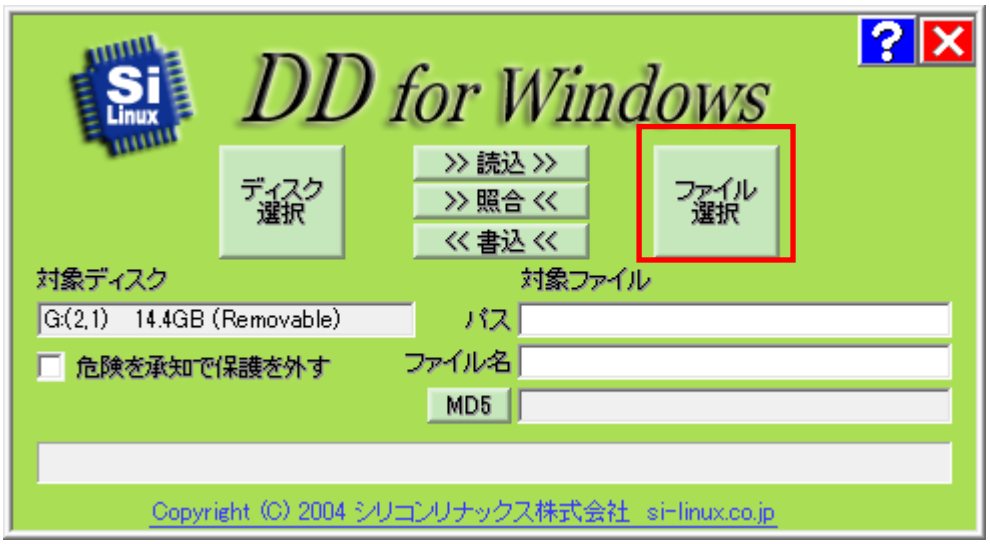


図 5-1-1-2.　ファイル選択

⑧　「対象ファイル」の項目に RecoveryImage.ddi が表示されたことを確認してください。

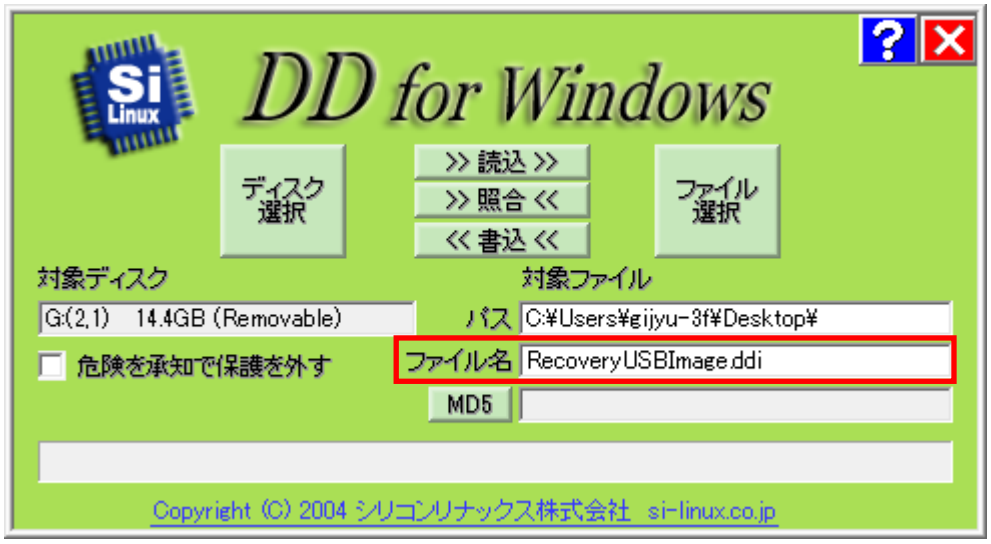


図 5-1-1-3.　対象ファイル

⑨　「<<書込<<」ボタンを押してください。
SD カード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。

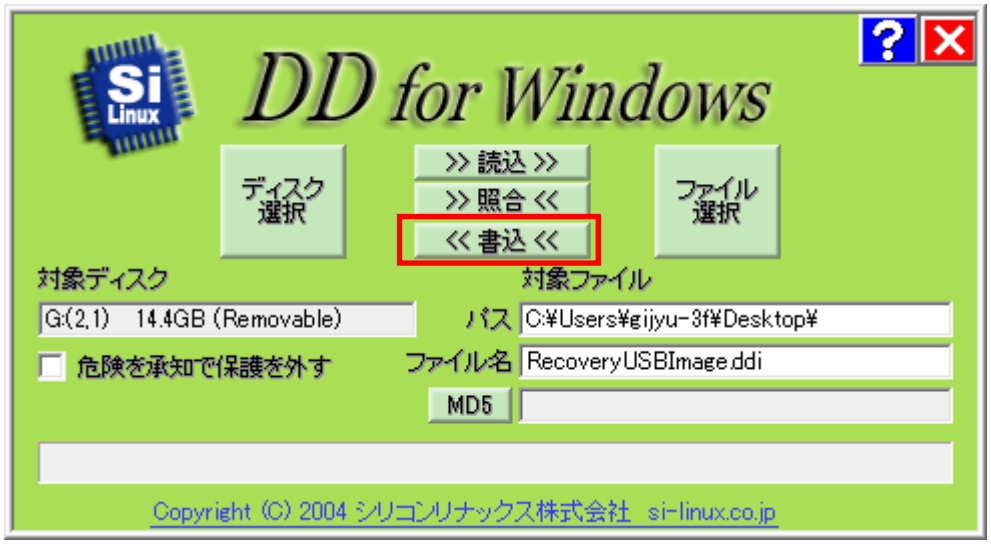


図 5-1-1-4.　書き込み開始

以上でリカバリ用マイクロ SD カードの作成は完了です。
リカバリ用マイクロ SD カードは一度作成すれば次回以降も使用することができます。

５－２　システムの復旧（バックアップデータ）

工場出荷イメージをメインストレージ（eMMC）に書込むことで、システムを工場出荷状態に復旧することができます。

また、「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（eMMC）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。

※　システムを工場出荷状態へ復旧するとメインストレージにあるデータはすべて消えてしまいます。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。

※　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。他の本体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。

●システムの復旧（バックアップデータ）の手順

① 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。
また、工場出荷状態への復旧を行う場合、マイクロ SD のパーティション 4、あるいは 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)にリカバリ DVD 内の工場出荷時イメージファイルをコピーしておいてください。
工場出荷時イメージファイルはリカバリ DVD の以下のフォルダに格納されている xxxxx.img ファイルです。
[リカバリ DVD]¥Image¥

② 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、ストレージメディアを取り外してください。

③ リカバリ用マイクロ SD カードを起動させます。起動方法については、ハードウェアマニュアルを参照してください。

④ 起動すると、リカバリメイン画面(図 5-2-1)が表示されます。[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。

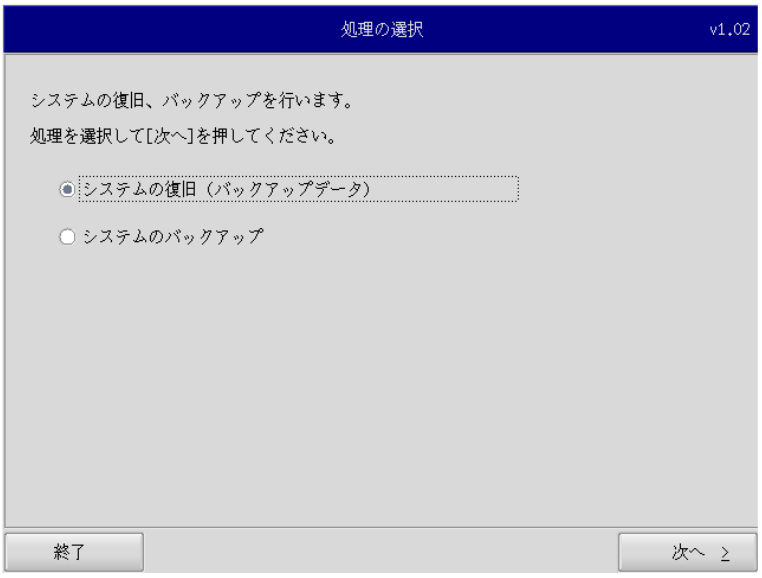


図 5-2-1. リカバリメイン画面

⑤　メディアの選択画面(図 5-2-2)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 5-2-2. メディア選択画面

⑥　メディアとパーティション選択画面（図 5-2-3）が表示されます。あらかじめ用意しておいたストレージメディアを本体に接続し、[メディア情報更新]ボタンを押してください。ストレージメディアのパーティションを選択し、[次へ]ボタンを押します。
ストレージメディアの認識には少し時間がかかります。ストレージメディアを接続してすぐに[メディア情報更新]ボタンを押すと、目的のメディア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。

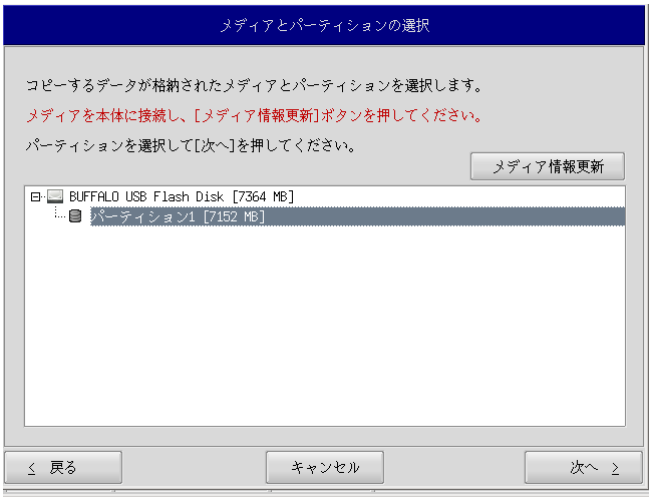


図 5-2-3. メディアとパーティション選択画面

⑦　フォルダ選択画面（図 5-2-4）が表示されます。［参照］ボタンを押します。

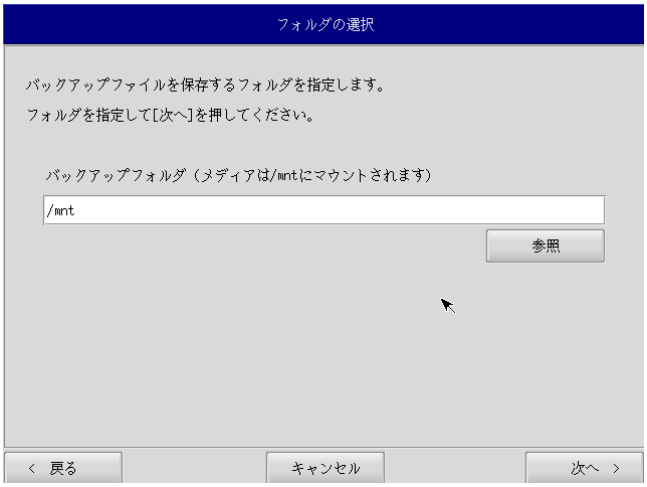


図 5-2-4. フォルダ選択画面

⑧　ファイル参照画面（図 5-2-5）が表示されます。接続したストレージメディアは、/mnt にマウントされていますので、/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。

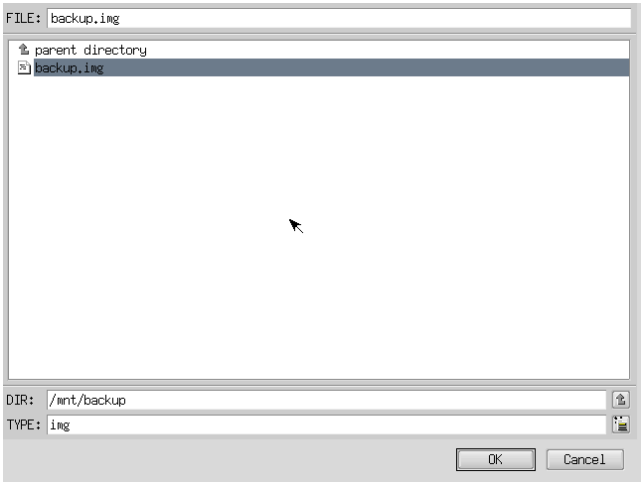


図 5-2-5. ファイル参照画面

⑨　ファイル参照画面（図 5-2-5）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。

⑩　コンペア処理の選択画面(図 5-2-6)が表示されます。
データ買い込み時のコンペ合処理の有無を選択します。

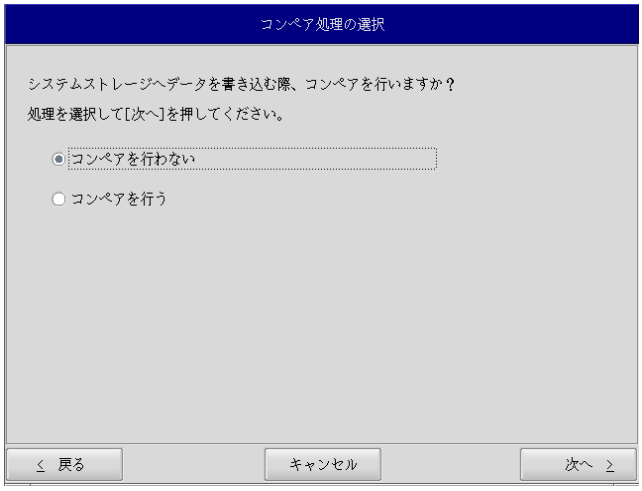


図 5-2-6. コンペア処理選択画面

⑪　確認画面(図 5-2-7) 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。

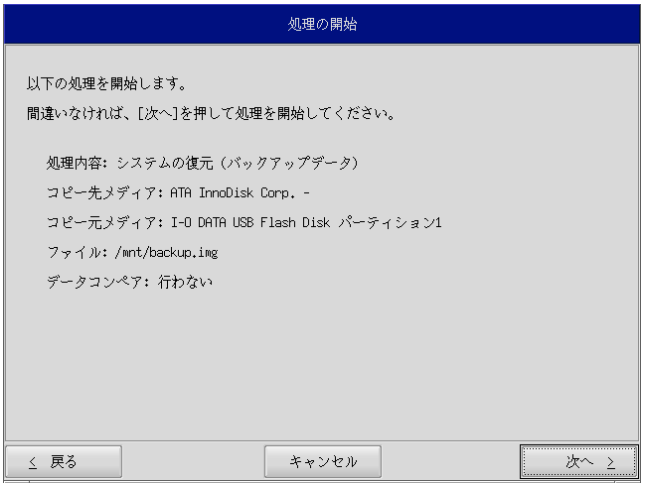


図 5-2-7. 確認画面

⑫　実行中画面（図 5-2-8）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、保存メディア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。

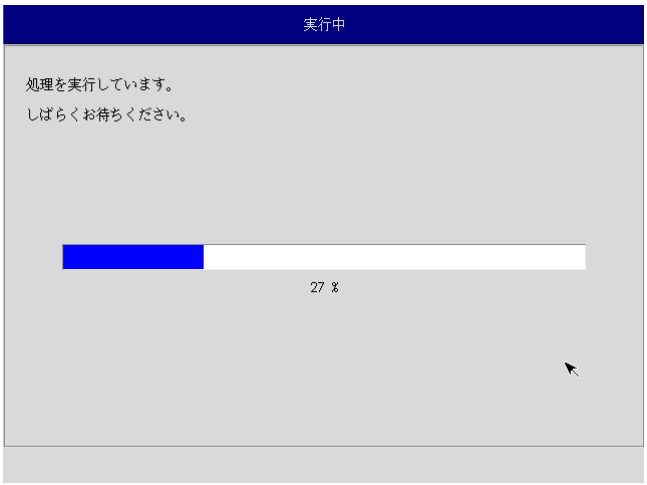


図 5-2-8.　実行中画面

⑬　終了画面（図 5-2-9）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、リカバリ用マイクロ SD カード、保存メディアを外します。シャットダウン後、eMMC ブートに戻します。

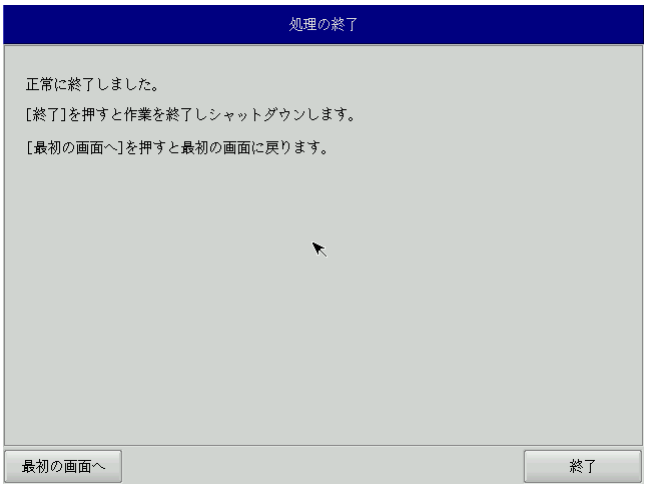


図 5-2-9.　終了画面

⑭　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。

※　システムを工場出荷状態へ復旧する場合、一度目の起動時にシステム再起動を求められる場合があります。この場合は指示に従い再起動してください。

５－３　システムのバックアップ

メインストレージ（eMMC）の状態をファイルに保存します。バックアップファイルはリカバリ用マイクロ SD カードに保存できるほか、外部メモリとして、USB メモリ、SD カードなどに保存することもできます。外部メモリの空き容量は、バックアップファイルの保存に十分な空き容量が必要となります。安全のためメインストレージの容量以上の空き容量がある外部メモリを用意してください。

バックアップソフトにフォーマット機能はありません。外部メモリはあらかじめ Windows、Linux などでフォーマットしてください。対応しているファイルシステムは、NTFS、EXT2、EXT3 となります。

※　バックアップファイルのサイズが 4GByte を超える可能性があるため、FAT、FAT32 ファイルシステムは使用しないでください。

※　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、他の本体では動作しませんので注意してください。

●システムのバックアップの手順

① 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。
② 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、ストレージメディアを取り外してください。
③ ハードウェアマニュアルを参考にリカバリ用マイクロ SD カードを起動させます。
④ リカバリメイン画面で[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。
⑤ 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

⑥　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。本体にメディアを接続し、[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メディアのパーティションを選択し、[次へ]ボタンを押します。

メディアの認識には少し時間がかかります。メディアを接続してすぐに[メディア情報更新]ボタンを押すと、目的のメディア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。

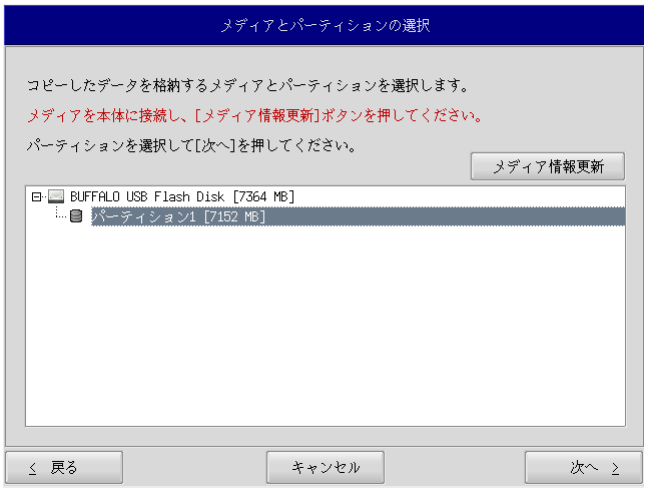


図 5-3-2. メディアとパーティション選択画面

⑦　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。

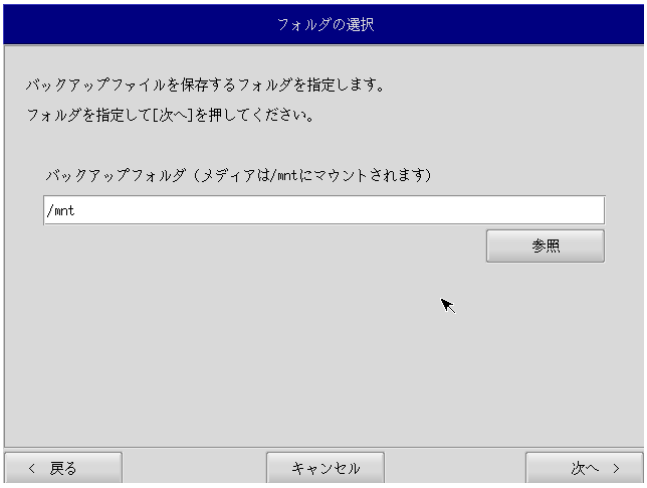


図 5-3-3. フォルダ選択画面

⑧　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。

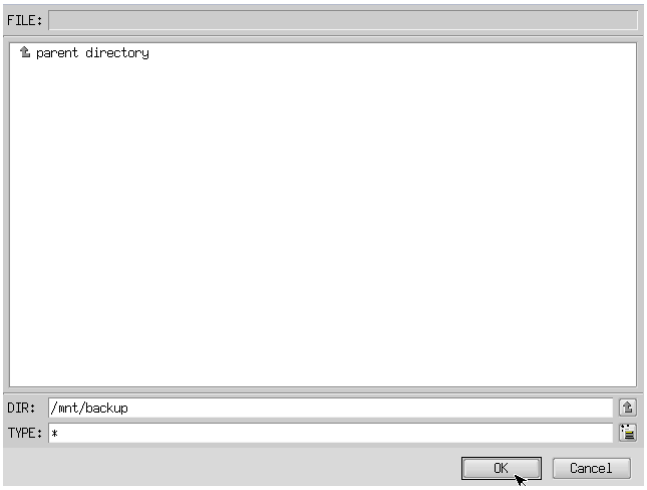


図 5-3-4.　フォルダ参照画面

⑨　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。

⑩　コンペア処理の選択画面(図 5-3-5)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。

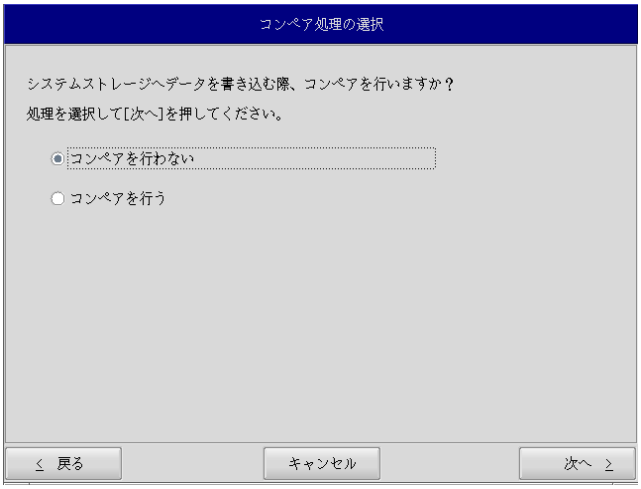


図 5-3-5.　コンペア処理選択画面

⑪　確認画面（図 5-3-6）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。

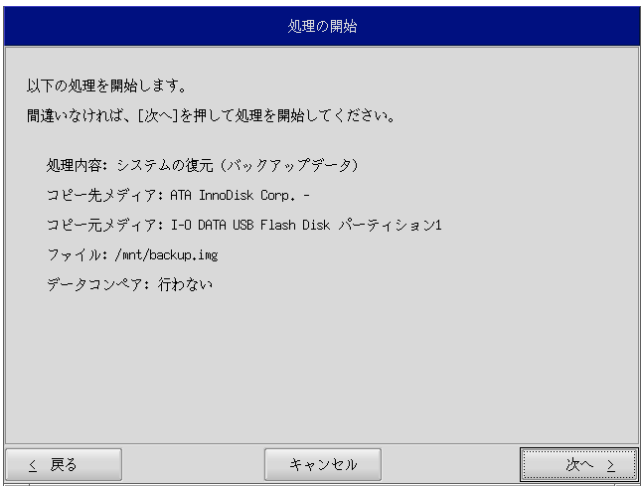


図 5-3-6.　確認画面

⑫　実行中画面（図 5-3-7）が表示され、処理が開始されます。実行中は保存メディア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。

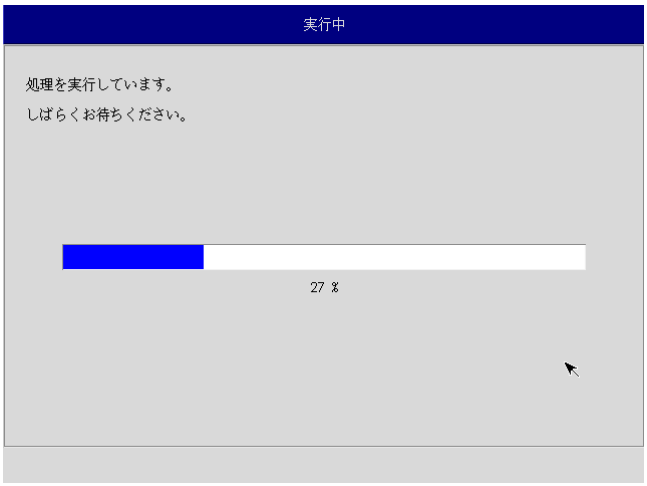


図 5-3-7.　実行中画面

⑬　終了画面（図 5-3-8）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。また、シャットダウン後に eMMC 起動に戻してください。

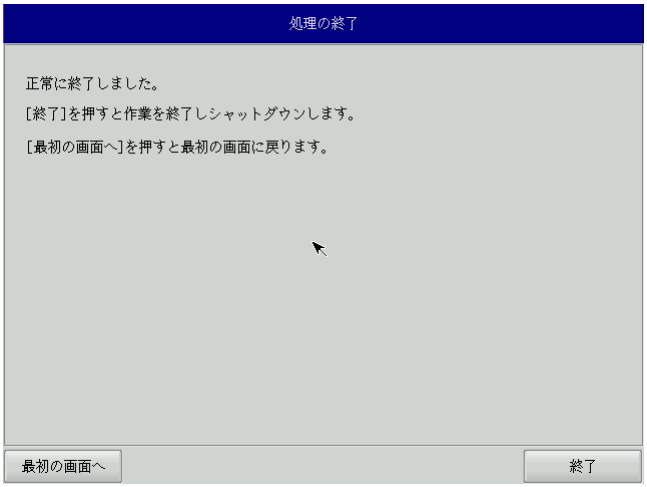


図 5-3-8. 終了画面

⑭　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。

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。

７７ＥＣ６０００１Ａ　　２０２１年　１月　初版

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