マニュアル

産業用組込み PC EC4A-***A*シリーズ用

Linux ディストリビューション

『Algonomix4』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 EC4A 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 EC4A-***A*シリーズ（以降 EC4A シリーズ）用 Linux ディストリビューション（以降 **Algonomix4**）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

Algonomix4 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。Algonomix4 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、Algonomix4 の具体的な内容を説明する前に、Algonomix4 の概要について説明します。

１－１　Algonomix4 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

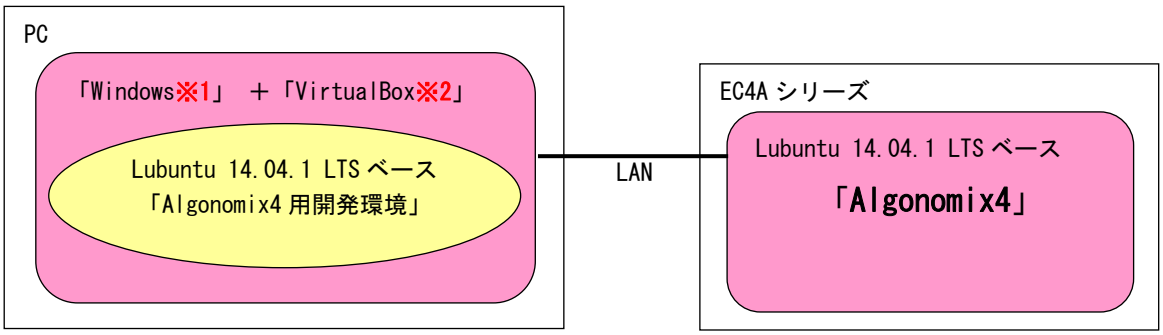
Linux カーネルといくつかの必要なソフトウェアパッケージをまとめて、OS として使えるようにしたものを Linux ディストリビューションといいます。

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。

Algonomix4 は、「Lubuntu 14.04.1 LTS」という Linux ディストリビューションをベースにした、EC4A シリーズ用の Linux ディストリビューションです。Algonomix4 は、「Lubuntu 14.04.1 LTS」に EC4A 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 Algonomix4 用の開発環境をインストールすることで、Algonomix4 用のソフトウェアを開発することができます。

Algonomix4 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. Algonomix4 の開発環境

開発環境の詳細については、『第 3 章 開発環境』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。

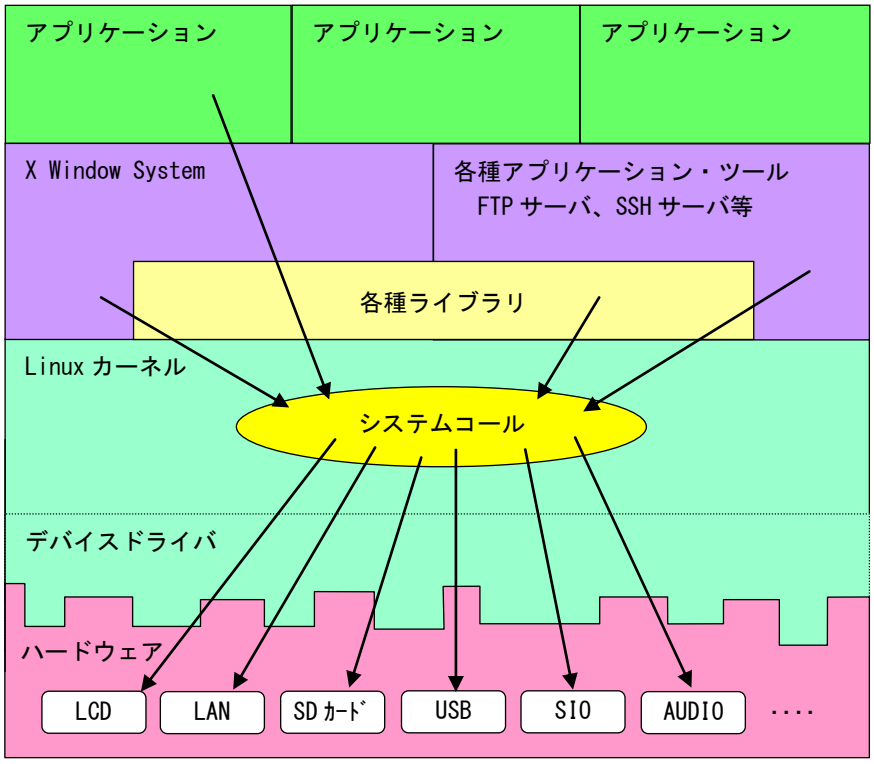


図 1-2-1. Linux ソフトウェア構成図

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　Algonomix4 パッケージについて

Algonomix4 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 Lubuntu14.04.1 の基本パッケージとしてインストールされているパッケージは除きます。

表 2-1-1. プリインストールパッケージ一覧

パッケージ名	内容	バージョン
Linux Kernel	Linux Kernel 本体	3.13.11.4-1-i686
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	1:6.6p1-2ubuntu2
gdbserver	GNU デバッガ	7.7-1-0ubuntu5~14.04.2
vsftpd	セキュリティの高い FTP サーバ	3.0.2-1ubuntu2.14.04.1
lm-sensors	センサ監視ツール、ドライバ	1:3.3.4-2ubuntu1
aptitude	APT 用パッケージ管理ツール	0.6.8.2-1ubuntu4
apache2	Web サーバ	2.4.7-1ubuntu4.1
libjpeg62:i386	JPEG ランタイムライブラリ	6b1-4ubuntu1
chromium-browser	オープンソースのウェブブラウザ	40.0.2214.111-0ubuntu0.14.1.1069
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
ttf-kochi-gotic	True Type ゴシックフォント	20030809-15
ttf-kochi-mincho	True Type 明朝フォント	20030809-15

Algonomix4 および Algonomix4 開発環境では、Lubuntu14.04.1 の基本パッケージと本書に表記したパッケージが入った環境でのみ動作を確認しています。そのため、パッケージの追加に制限をかけています。
詳細に関しては、「3-6 パッケージについて」を参照してください。

Algonomix4 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 デスクトップ左下のアイコン(以後[メニューアイコン]と表記)からメニューを呼び出す→[アクセサリ]→[LXTerminal]を選択し、図 2-1-1 のようなコンソール画面を表示します。

図 2-1-1. コンソール画面

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

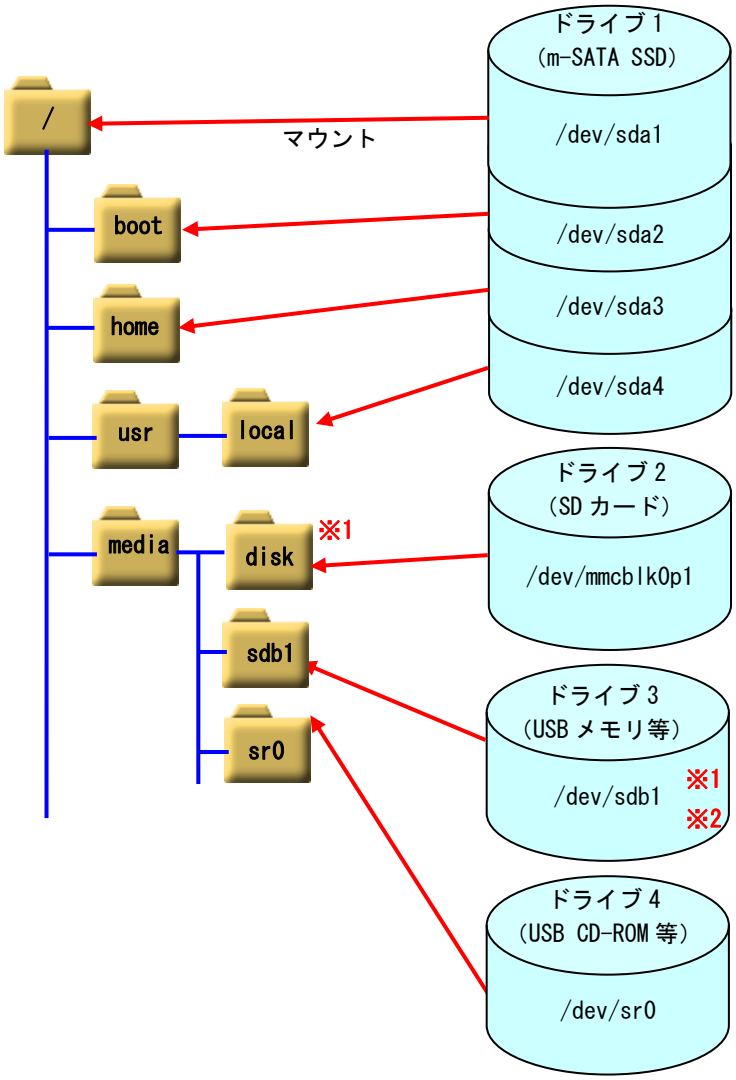
3：パッケージのバージョンおよびリビジョン

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　　Algonomix4 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、HDD 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。Algonomix4 は Lubuntu ベースとなっている為、Lubuntu のディレクトリ構造に従っています。



※注 1：ストレージ機器をマウントするたびに sdb1、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. Lubuntu のツリー構造

EC4A シリーズでは、m-SATA SSD 64GByte 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。

このパーティション構成は、m-SATA SSD を ReadOnly 化する際、保護する領域と保護しない領域を分けるために必要となります。詳細は『2-7 データの保護について』を参照してください。

EC4A 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. EC4A 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
/dev/sda1	88MByte	49MByte	32MByte	61%	/boot
/dev/sda2	15GByte	2.3GByte	12GByte	17%	/
/dev/sda3	29GByte	45MByte	28GByte	1%	/home
/dev/sda4	15GByte	47MByte	14GByte	1%	/usr/local

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
+--cdrom
+--dev
+--etc
+--initrd.img
+--home--+--asdusr
+--lib
+--lost+found
+--media
+--mnt
+--opt
+--proc
+--root
+--run
+--sbin
+--srv
+--sys
+--tmp
+--usr--+--bin
        +--games
        +--include
        +--lib
        +--local
        +--sbin
        +--share
        +--src
+--var
+--vmlinuz
```

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
/	ルートディレクトリ **※注：m-SATA SSD を Read Only として使用する場合でも、この領域は読み書きすることができますが、RAM ディスク上に書かれるため、電源を落とすと変更内容は消去されます。**
/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。 **※注：m-SATA SSD を Read Only として使用する場合でも、この領域は読み書きすることができます。**
/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。 ・フレームバッファ：/dev/fb*　　例：fb0 LCD に表示されるグラフィックイメージを保持する領域です。 ・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。**
/etc	設定ファイルが格納されています。 **※注：m-SATA SSD を Read Only として使用する場合でも、この領域は読み書きすることができますが、RAM ディスク上に書かれるため、電源を落とすと変更内容は消去されます。**
/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。Algonomix4 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 **※注：m-SATA SSD を Read Only として使用する場合でも、この領域は読み書きすることができます。**
/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。
/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、ifconfig、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。
/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。 **※注：m-SATA SSD を Read Only として使用する場合でも、この領域は読み書きすることができます。**
/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。
/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。

２－３　　Algonomix4 設定ツール「ASD Config」について

本項では、EC4A 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．GRUB 起動画面で、AsdConfigMenu モードを選択します。(『2-7-2 ルートファイルシステムの保護について』を参照してください。)

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。

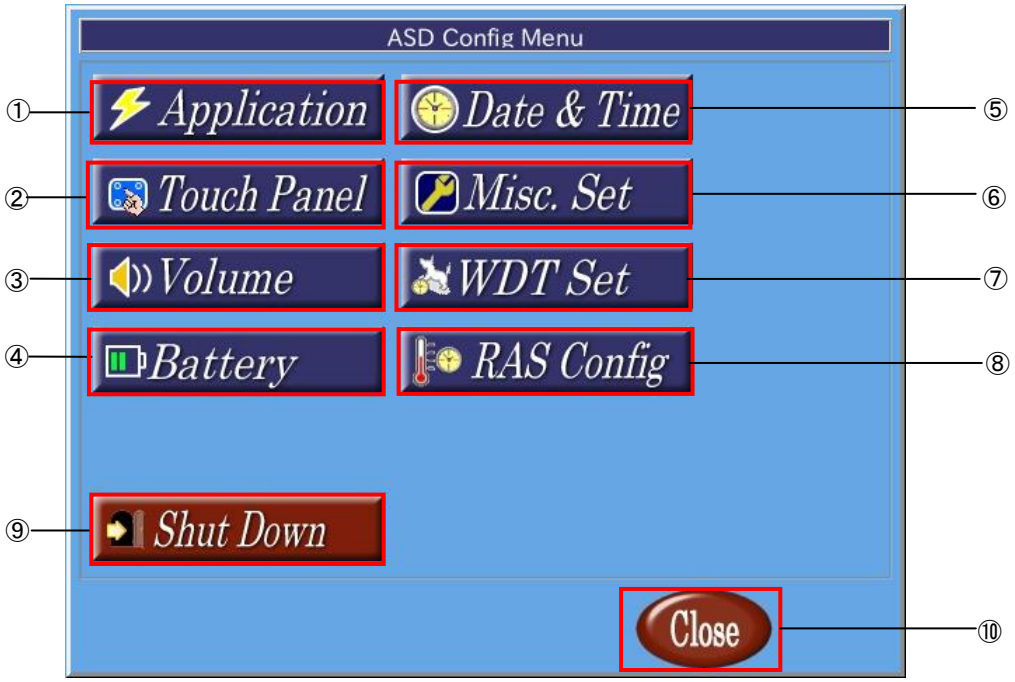


図 2-3-1. ASD Config Menu

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。

※ EC4A シリーズには、タッチパネルは搭載されておりません。

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。

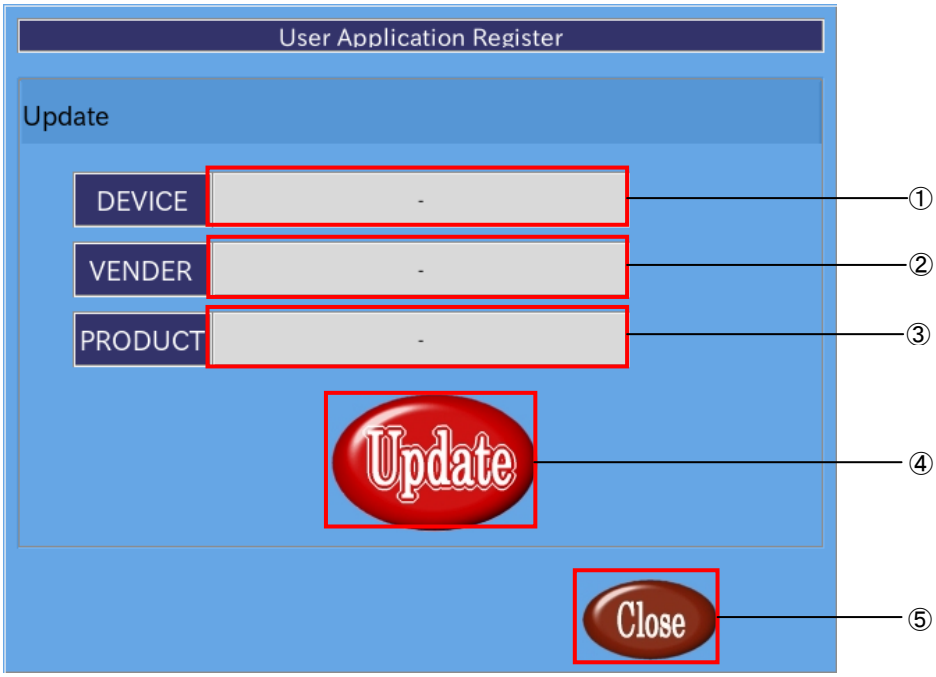


図 2-3-1-1.　アップデート画面

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。

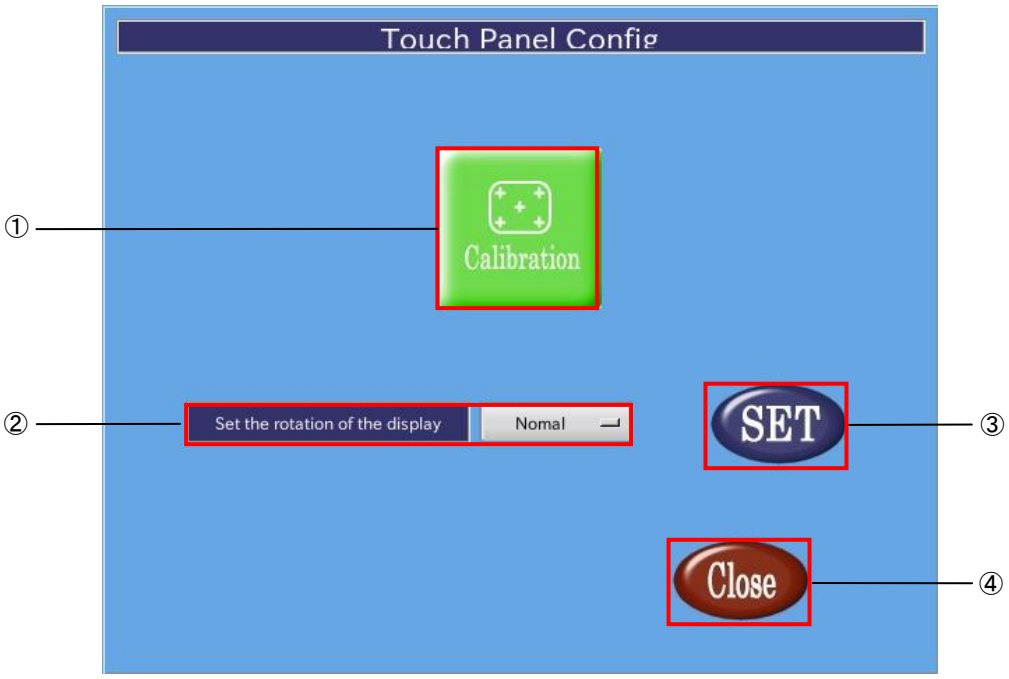


図 2-3-2-1. メニュー画面

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。
※　EC4A シリーズにはタッチパネルが搭載されていないため、このメニューは無効になっています。
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。

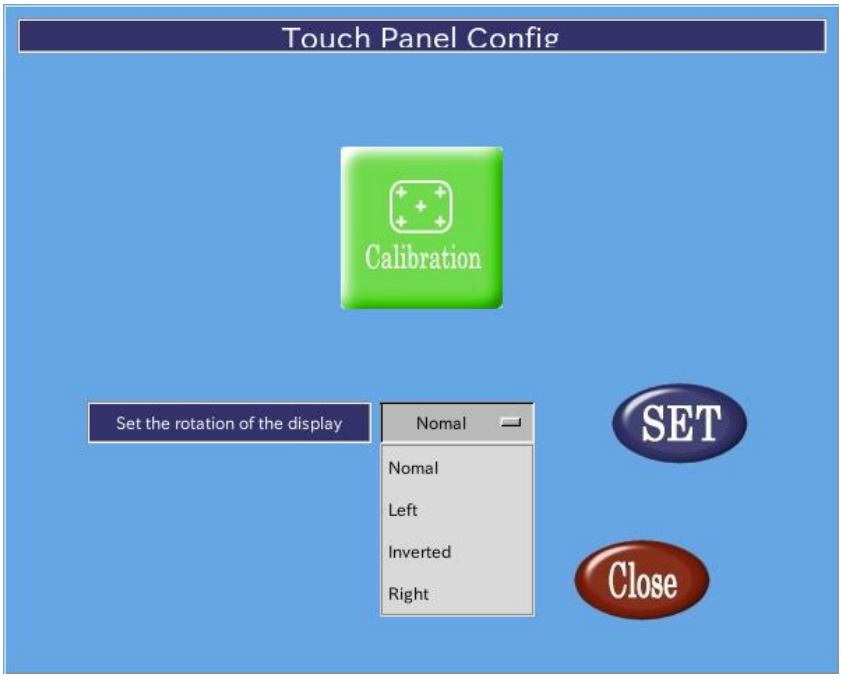


図 2-3-2-2. ディスプレイとタッチパネルの回転

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	マスターボリュームを設定します
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：EC4A シリーズでは、Master、HeadPhone、PCM、Mic のみ設定が有効です。

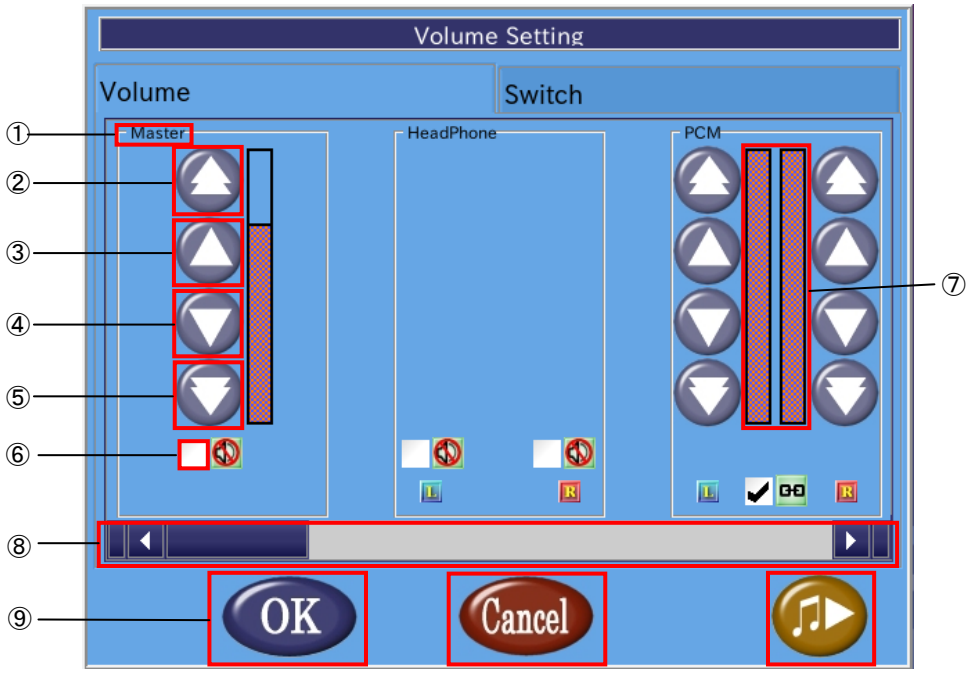


図 2-3-3-1.　ボリューム調整画面

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。

●ボリュームスイッチの変更

[ASD Volume Config]の[Switch]タブを選択することで、IEC958、IEC958 Default の有効・無効の切替え、録音時の音源の切替えを設定できます。

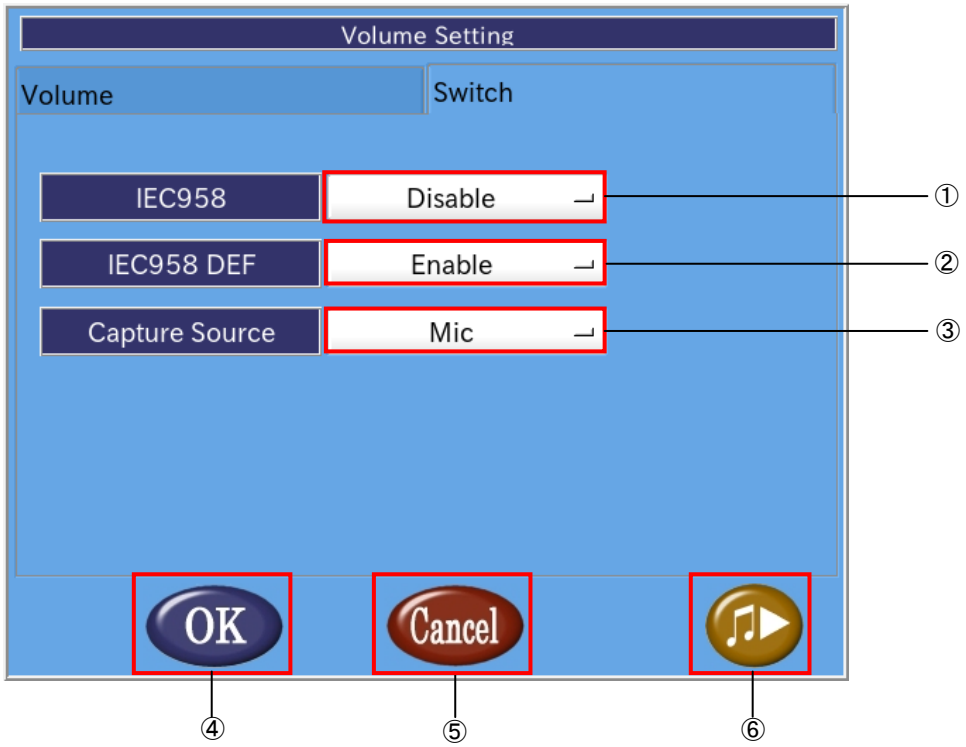


図 2-3-3-2.　ボリュームスイッチ画面

① IEC958
IEC958 プラグインの有効・無効を切替えます。
[Enable]で有効、[Disable]で無効となります。

② IEC958 DEF
デフォルトの IEC958 PCM プラグインの有効・無効を切替えます。
[Enable]で有効、[Disable]で無効となります。
EC4A シリーズでは、デフォルトの PCM プラグインは PCM となります。

③ Capture Source
録音時の音源を切替えます。
マイク、フロントマイク、ライン、フロントラインから選択できます。

※注：EC4A シリーズでは、マイク入力のみ対応しています。

④ 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。

⑤ 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。

⑥ 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。

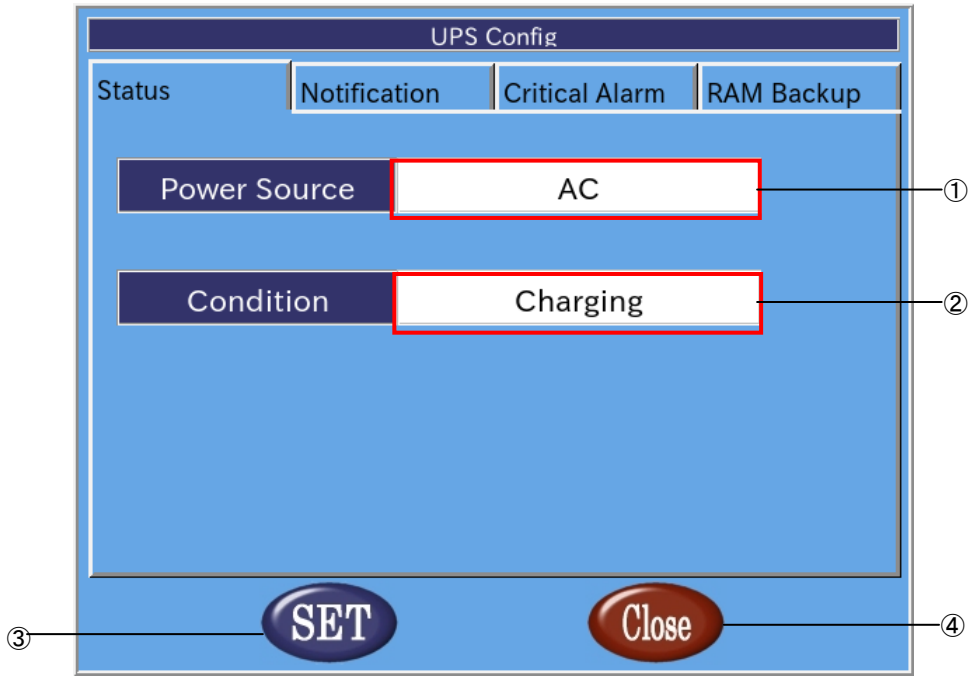


図 2-3-4-1. バッテリ状態の表示

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。

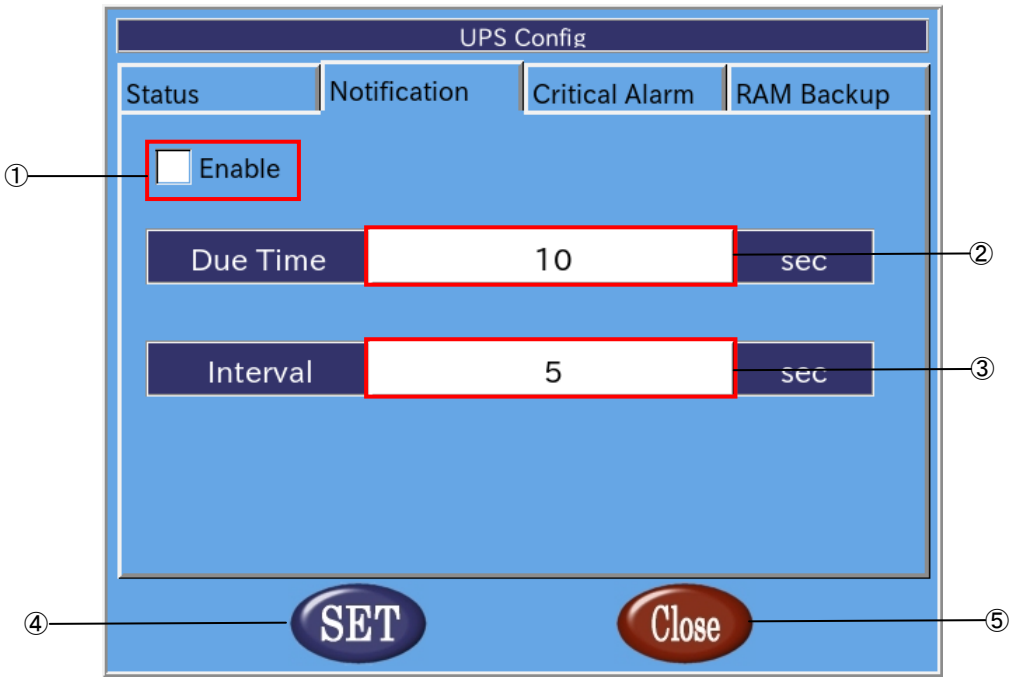


図 2-3-4-2. バッテリ異常通知設定

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。

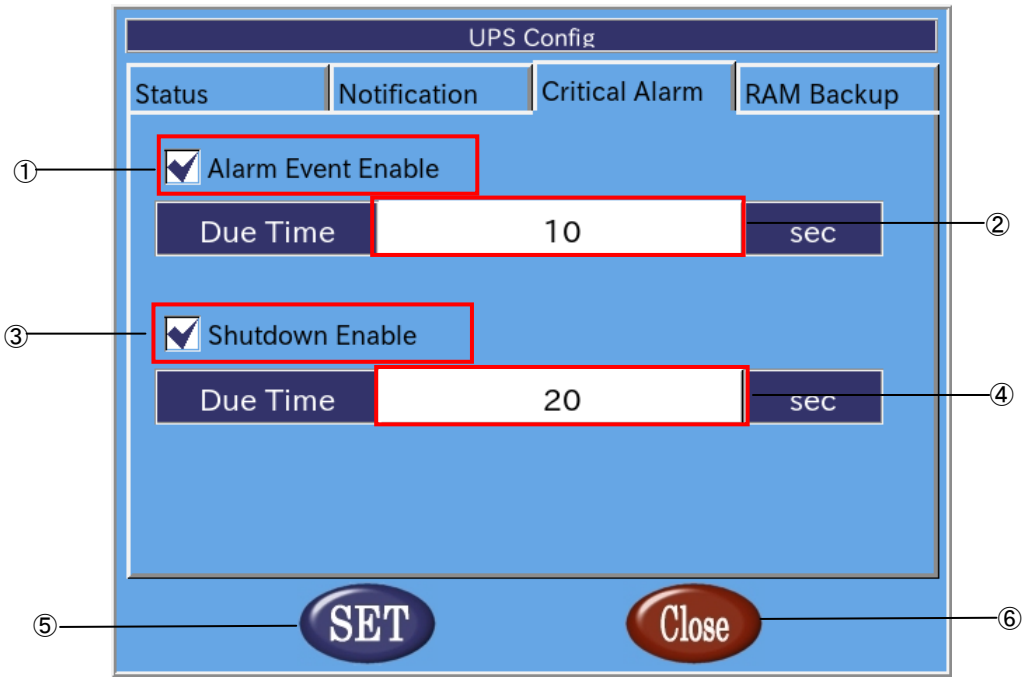


図 2-3-4-3. バッテリ警告設定

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。

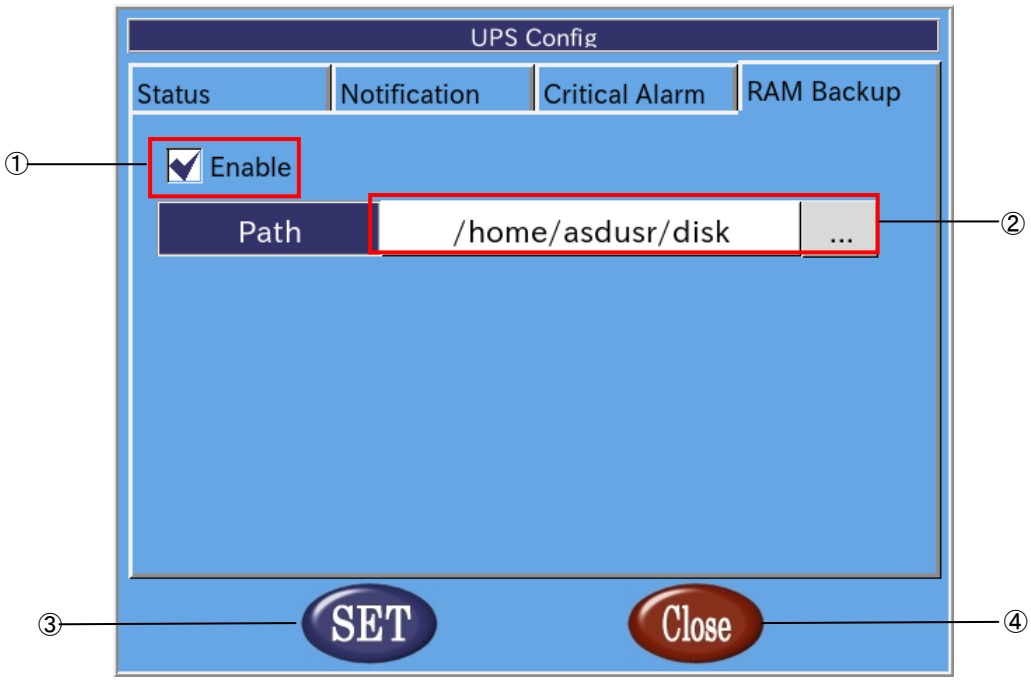


図 2-3-4-4. RAM バックアップ設定

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、『4-8-3 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。

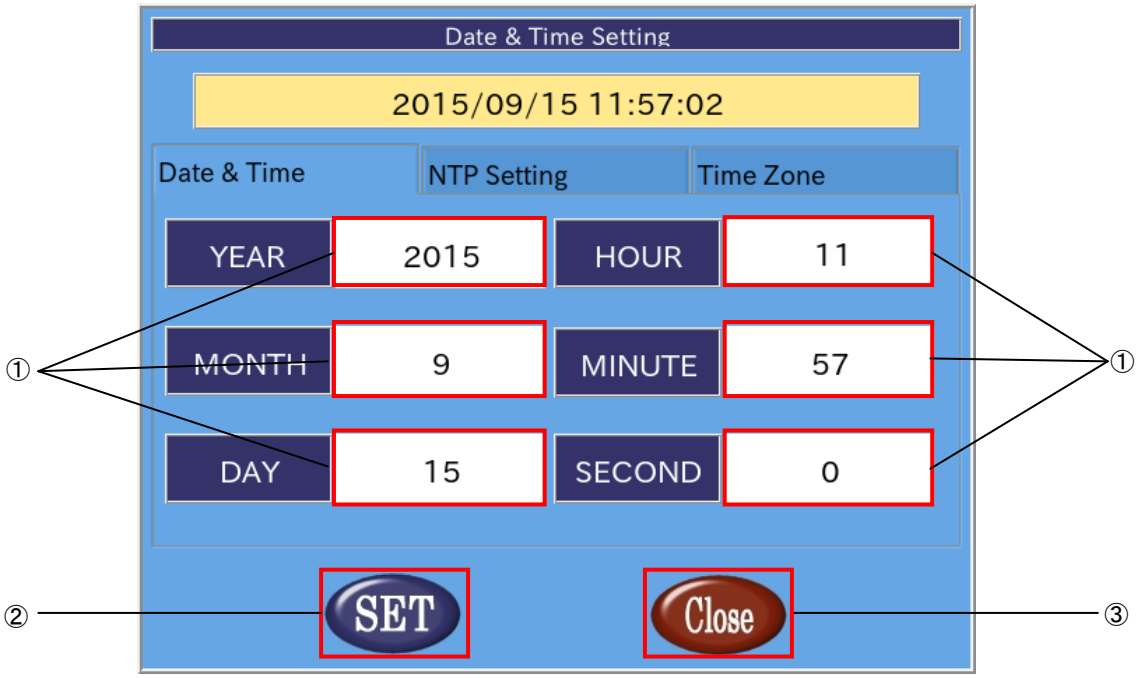


図 2-3-5-1. 時計の設定

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 EC4A 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。

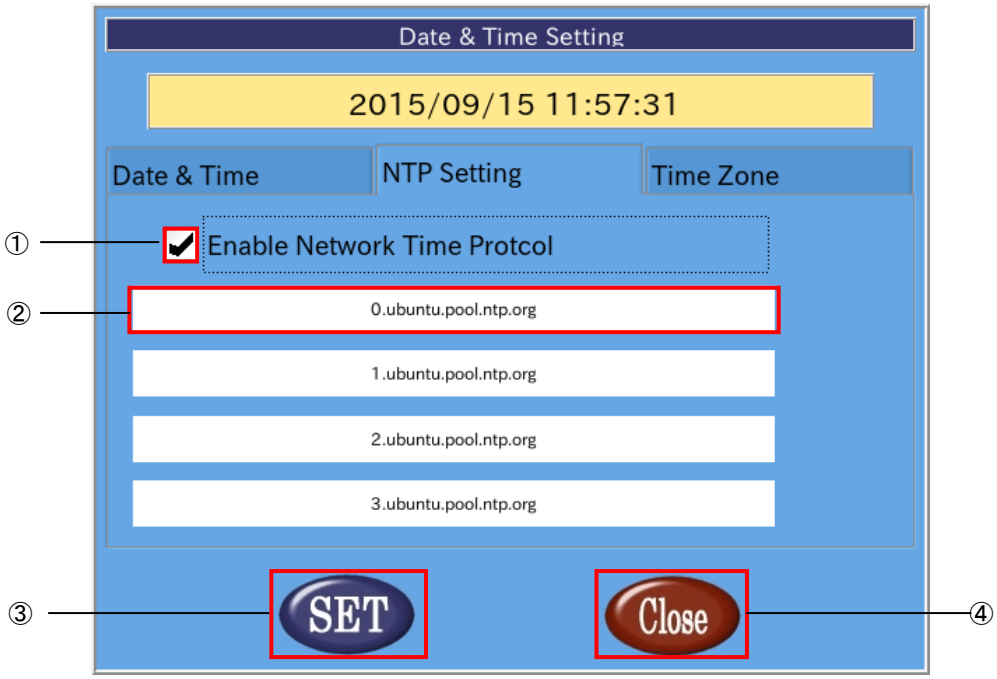


図 2-3-5-2. NTP サーバの設定

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：EC4A 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。

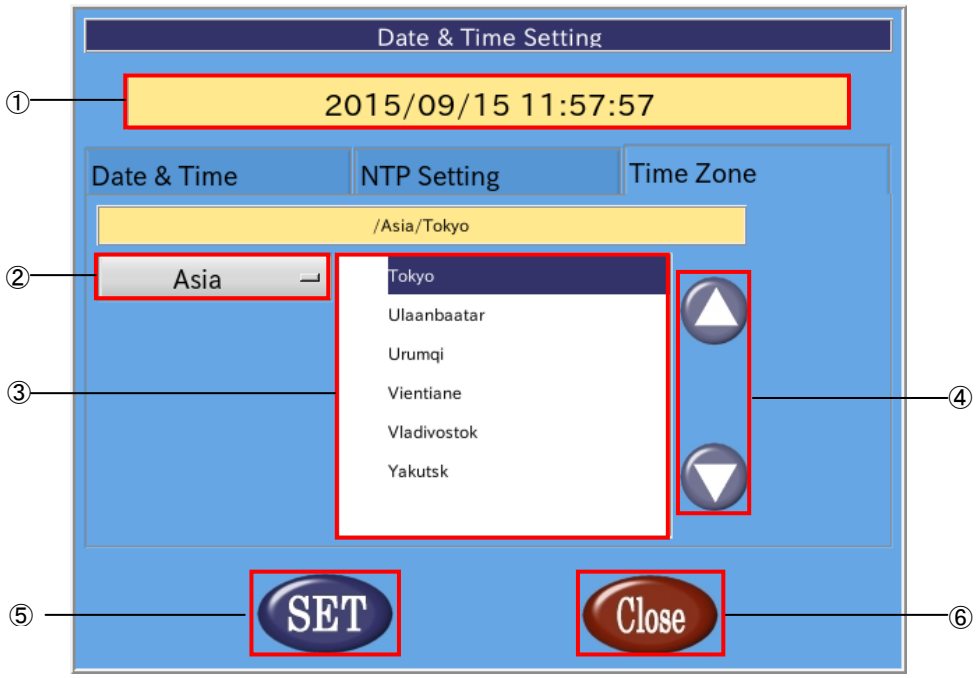


図 2-3-5-3. タイムゾーンの設定

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

ASD Misc Setting の[Backlight & Buzzer Setting]タブを選択することで、図 2-3-5-1 の画面になります。
ここでは、バックライトの調節を行うことができます。

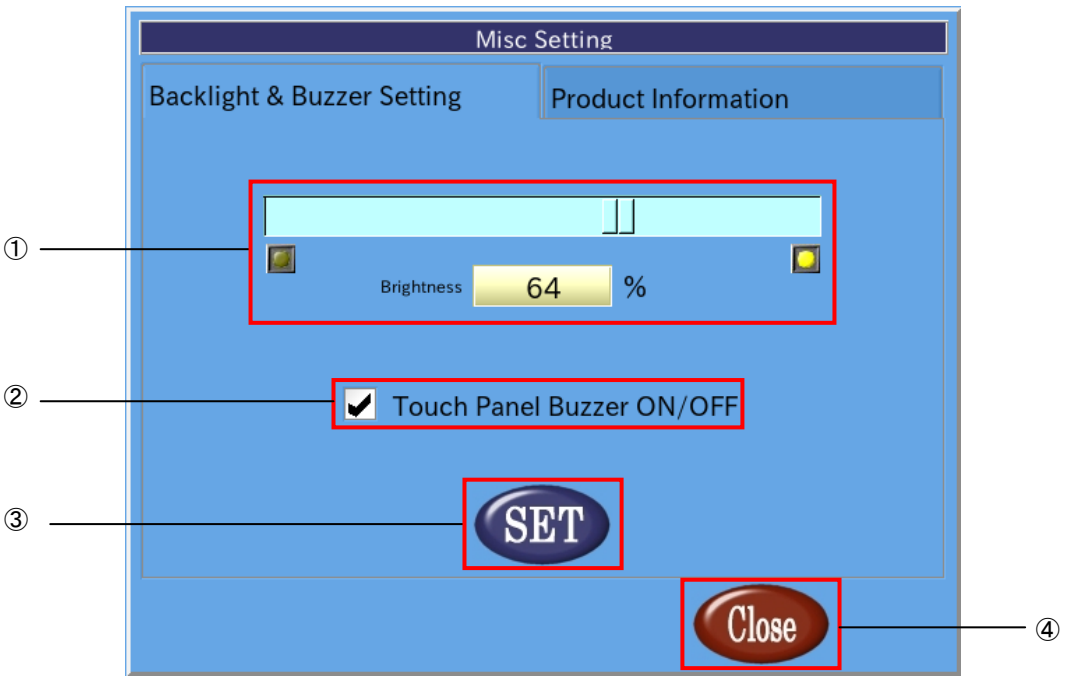


図 2-3-6-1. バックライトとブザーの設定

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 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。

③ 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。

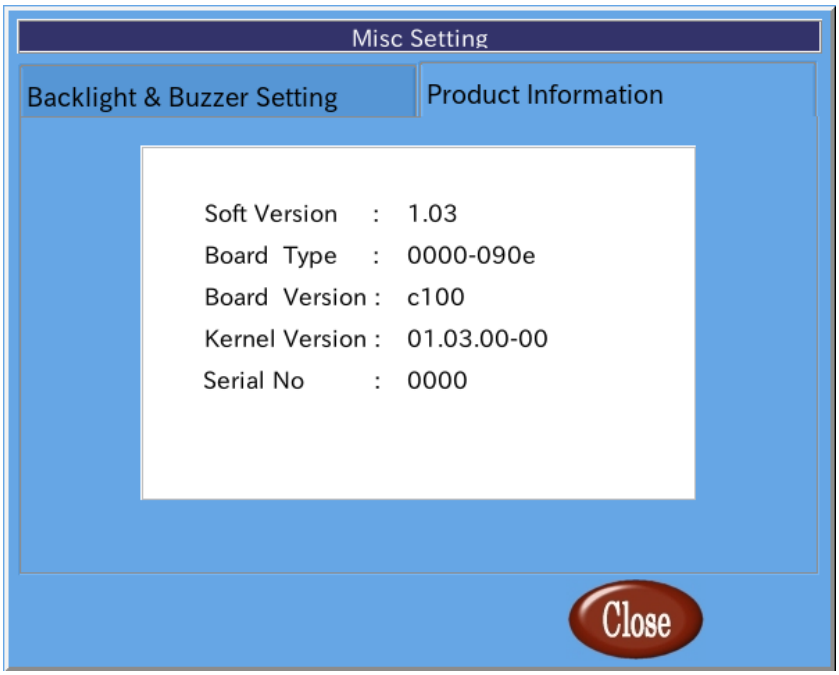


図 2-3-6-2. 製品情報

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
Backup battery is low	バックアップバッテリーの残量が低い時のみ表示

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。

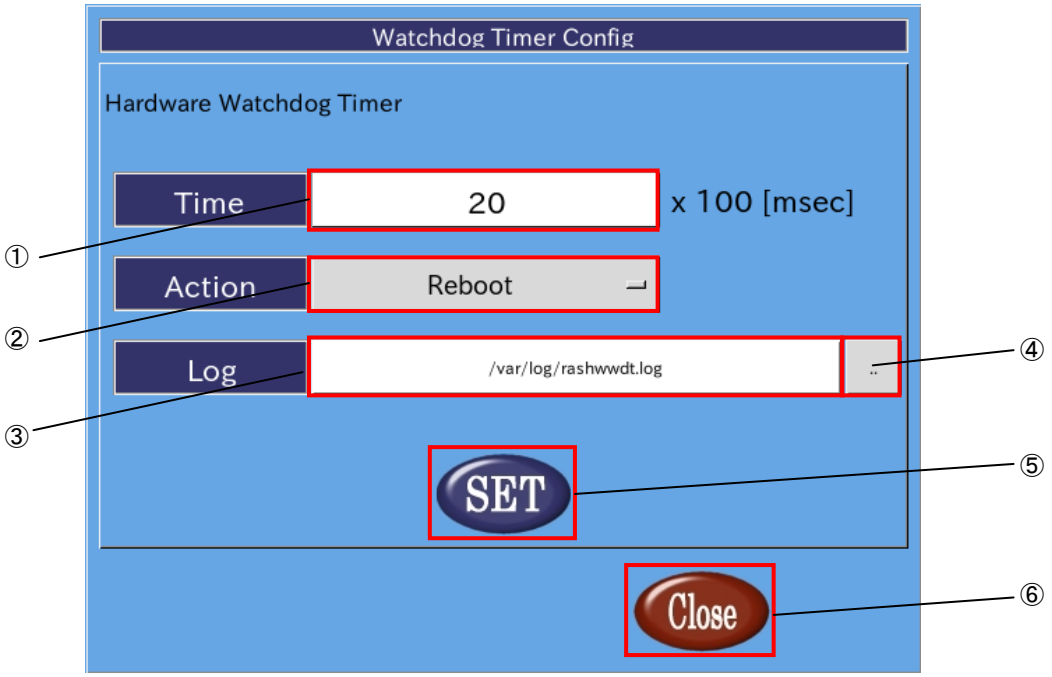


図 2-3-7-1. ハードウェア・ウォッチドッグタイマの設定

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 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。

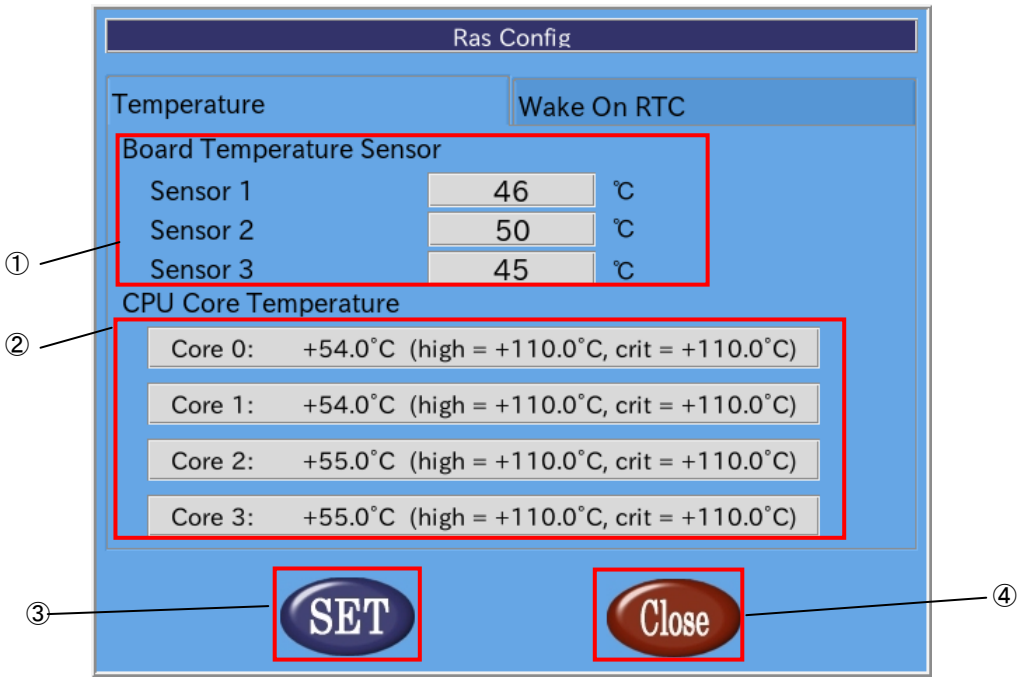


図 2-3-8-1. CPU 温度の表示

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	DRAM 付近の温度センサの測定値を表示します。
Sensor 2	PMIC 付近の温度センサの測定値を表示します。
Sensor 3	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。

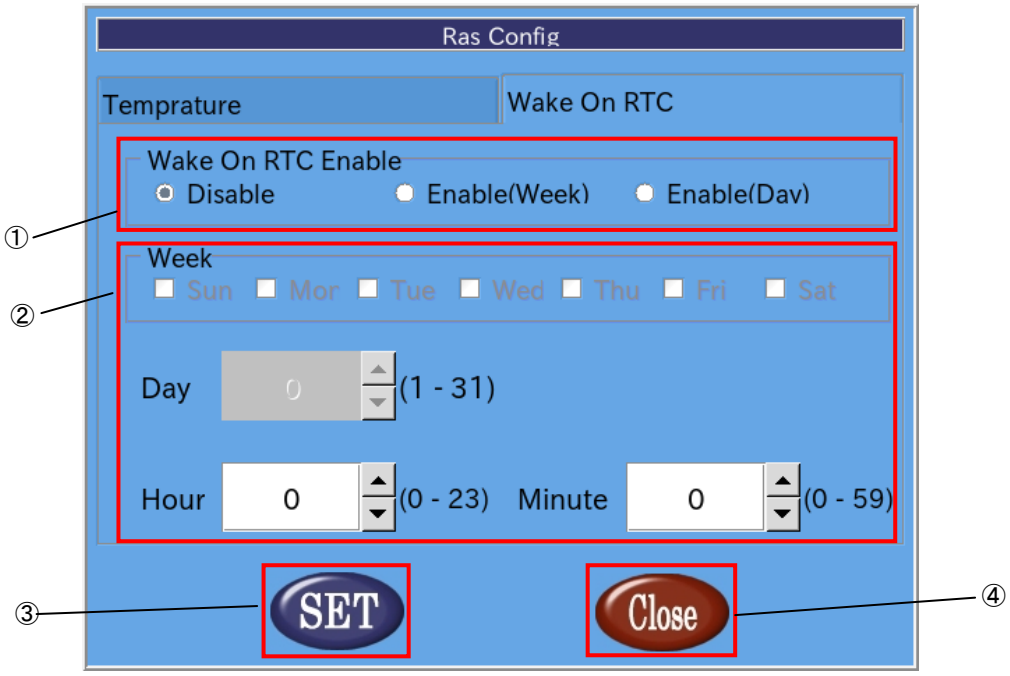


図 2-3-8-2. Wake On RTC の設定

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、EC4A シリーズの再起動、シャットダウンを行うことができます。
また、起動時のメニューを変更することもできます。
ASD Shutdown Menu を起動するには、メニュー画面から[Shutdown]を選択します。

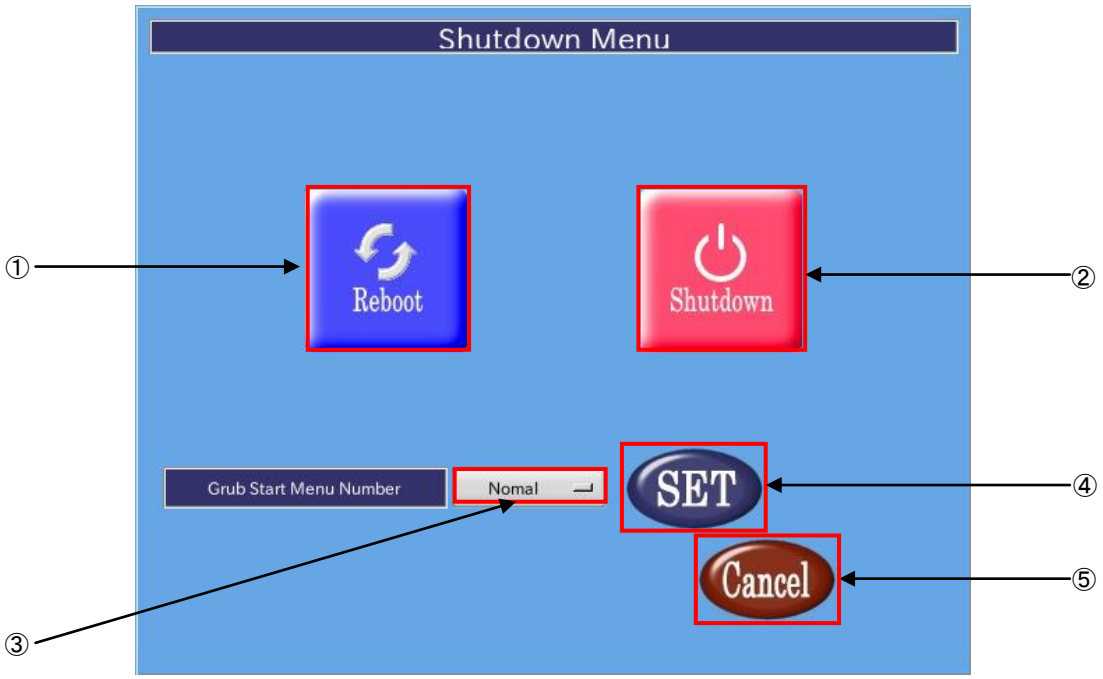


図 2-3-9-1. ASD Shutdown Menu

① 再起動
[Reboot]ボタンを押下することで、EC4A 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、EC4A シリーズがシャットダウンします。

③ Grub Menu の選択
[Grub Start Menu Number]の項目を選択することで、起動モードを変更できます。
選択できるモードを表 2-3-9-1 に示します。各項目の詳細については、『2-7-2 ルートファイルシステムの保護について』を参照してください。

表 2-3-9-1. 起動モード

項目名	内容
Normal	通常モードです。 全てのディレクトリを読書きできます。
AsdConfigMenu	起動時に ASD Config Menu が起動するほかは、Normal と同じです。
ReadOnly	/boot、/home、/usr/local 以外のディレクトリを書き込み不可で起動します。

④ 設定の反映
[SET]ボタンを押下することで、Grub Start Menu Number で選択した起動モードを反映します。
[SET]ボタンを押下しないと設定が反映されないので注意してください。

⑤ 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、EC4A 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　[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 2-4-1.　ネットワーク接続の起動

②　ネットワーク接続が起動します。「追加」ボタンをクリックします。

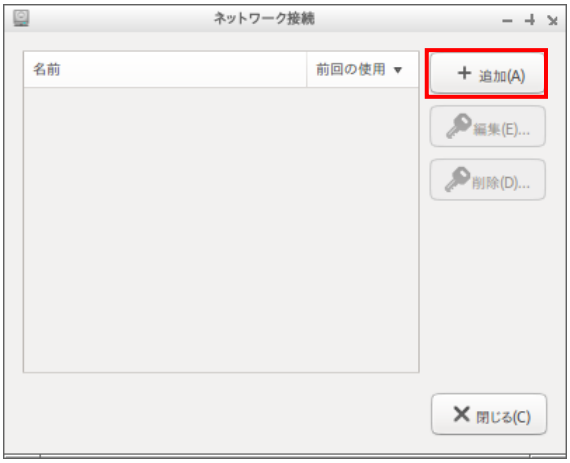


図 2-4-2.　ネットワーク接続の起動

③　ネットワークの種別を選択するダイアログが表示されます。
「Ethernet」を選択し、「作成」ボタンをクリックします。

図 2-4-3.　接続の種類の選択

④　有線 LAN の設定画面が表示されます。
設定するデバイスの MAC アドレスを選択します。

図 2-4-4.　有線 LAN 設定の編集

⑤　「IPv4 設定」タブを開き、IP アドレスを設定します。
IP アドレスを DHCP で設定するには、「方式」に「自動(DHCP)」を選択します。
静的 IP アドレスを割り当てるには、「方式」に「手動」を選択します。その後、「追加」ボタンをクリックし、アドレス、ネットマスク、ゲートウェイを設定します。

図 2-4-5.　IPv4 設定

⑥　「保存」ボタンをクリックし、設定を保存します。

２－５　　無線 LAN の設定について

本項では、EC4A 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。

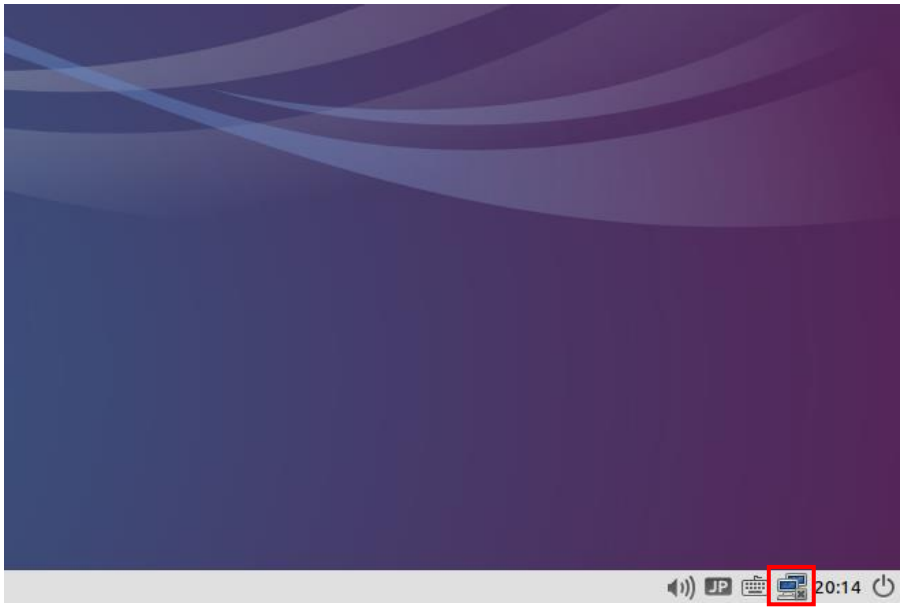


図 2-5-1.　ネットワークアイコン

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。

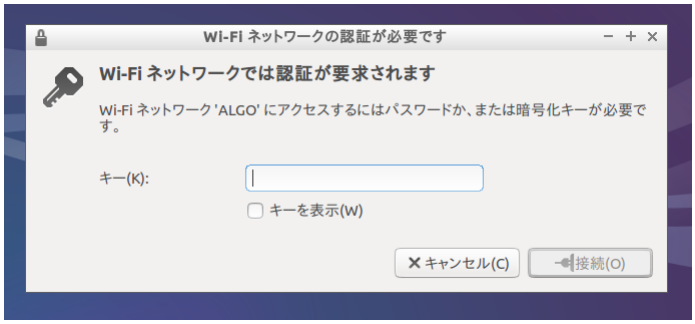


図 2-5-2.　キー入力画面

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。）

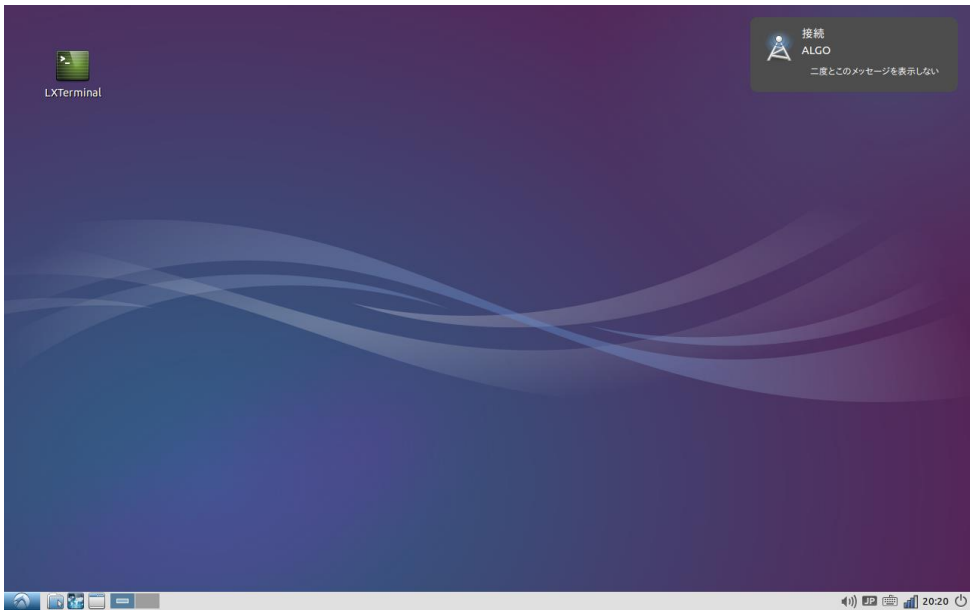


図 2-5-3. 接続成功

●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。

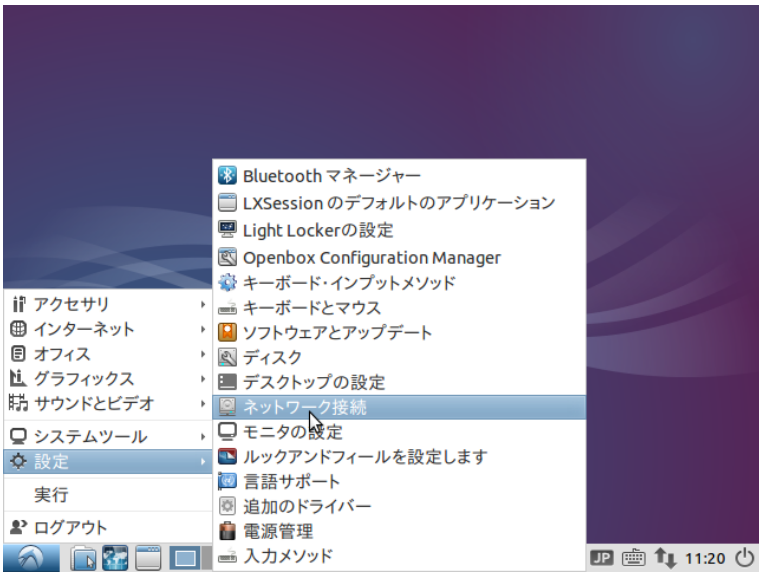


図 2-5-4. ネットワーク設定画面の起動

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。

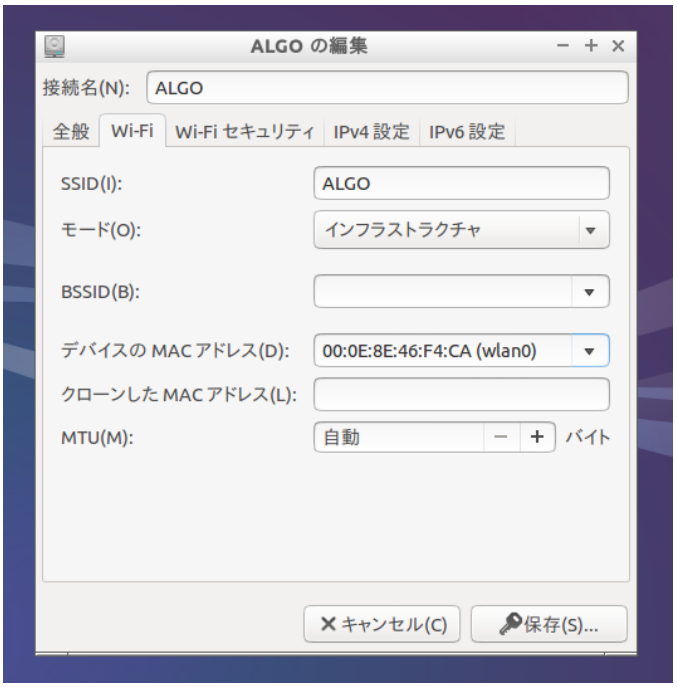


図 2-5-6. 無線 LAN 設定画面

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。

EC4A 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　　汎用 SW と汎用 LED の制御

汎用 SW の状態確認と 3 つの汎用 LED の制御ができます。

表 2-6-1-1. sysfs の汎用 SW の入力状態を確認する項目

sysfs ファイル名	データ	内容
/sys/devices/platform/miscio/miscio_sw	0～1 SW1：1bit 目	Read することで汎用 SW の状態を確認することができます。 汎用 SW1 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。
/sys/devices/platform/miscio/miscio_led	0～7 LED1：1bit 目 LED2：2bit 目 LED3：3bit 目	対応する bit を ON したデータを Write することで 3 つの汎用 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。 **※注：汎用 SW1 が ON ラッチしている間は LED1 が点滅します。**

２－６－２　　基板情報

基板情報を管理します。

表 2-6-2-1. sysfs の基板情報を制御する項目

sysfs ファイル名	データ	内容
/sys/devices/platform/mainboard/boardversion	0XXX	Read することで FPGA バージョンが読み出せます。
/sys/devices/platform/mainboard/buildversion	XXXXXXXX	Read することでカーネルのビルドバージョンが読み出せます。
/sys/devices/platform/mainboard/machcode	0000XXXX	Read することでマシンコードが読み出せます。
/sys/devices/platform/mainboard/dipswitch	X0XXX00000000	Read することで PCI デバイス側のマシンコードが読み出せます。
/sys/devices/platform/asd_sram/size	XXXX	Read することで仮想 RAM のサイズが読み出せます。単位は[kbyte]です。
/sys/devices/platform/asd_ups/condition	X	Read することで UPS 状態が読み出せます。 0：バッテリ充電中 1：バッテリ満充電 2：バッテリ放電中 3：バッテリ異常
/sys/devices/platform/asd_ups/powersource	X	Read することで UPS 電力源が読み出せます。 0：AC 1：バッテリ

２－６－３　　温度センサ

基板上の温度センサの情報を取得します。

表 2-6-3-1. sysfs の基板上温度センサの情報を取得する項目

sysfs ファイル名	データ	内容
/sys/bus/i2c/devices/7-004c/temp1_input	XX000	Read することで基板上の温度センサ(DRAM 付近)の温度を読み取ります。
/sys/bus/i2c/devices/7-004d/temp1_input	XX000	Read することで基板上の温度センサ(PMIC 付近)の温度を読み取ります。
/sys/bus/i2c/devices/7-004e/temp1_input	XX000	Read することで基板上の温度センサ(UPS バッテリ付近)の温度を読み取ります。

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
```

２－７－２　　ルートファイルシステムの保護について

EC4A シリーズでは、ルートファイルシステムは 16GByte の m-SATA SSD 上に構築されています。パーティション構成については、『2-2 Algonomix4 のディレクトリ構造』の表 2-2-1 を参照してください。

2 番目のパーティションにマウントされている「/boot」には、EC4A シリーズを起動するときに最初に実行される GRUB と呼ばれるブートローダが格納されています。

このブートローダで Linux の起動方法を選択することができます。ブートローダの起動メニュー画面を図 2-7-2-1 に示します。キーボード入力が無い場合、3 秒後に選択されているモードで自動起動します。

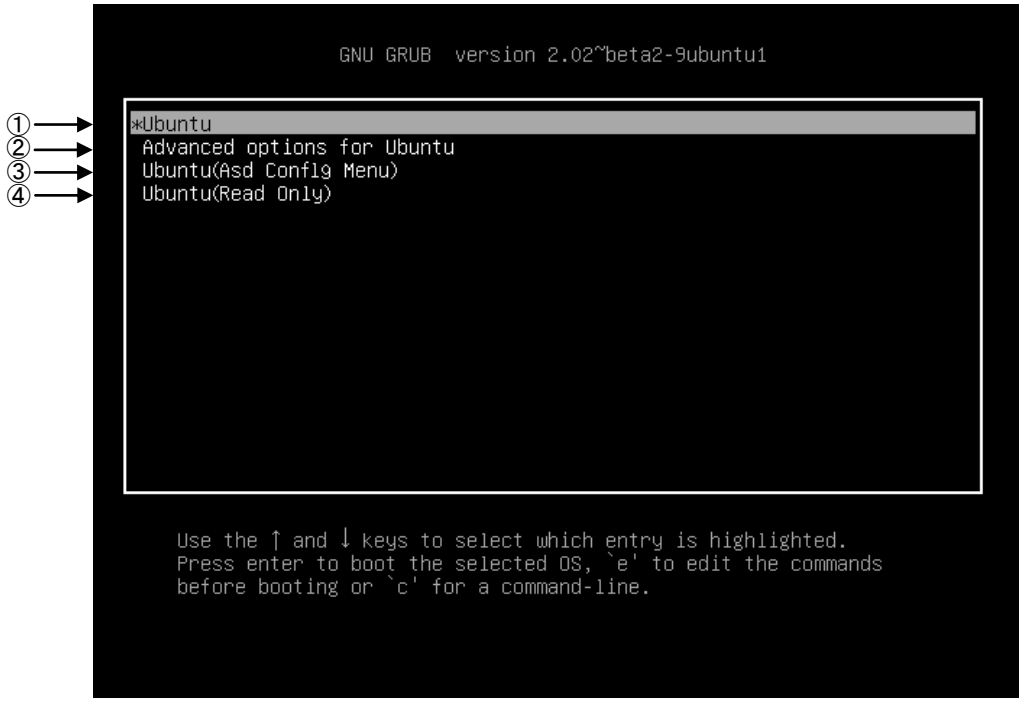


図 2-7-2-1. GRUB 起動メニュー画面

① 通常モード
すべてのディレクトリを読み書きできるモードで起動します。(出荷時設定)
アプリケーションのインストールや、/etc の設定ファイルの変更等を行うときはこのモードで起動してください。

② オプション
リカバリーモードが選択できます。通常モードまたは ReadOnly モードで正常に起動しなくなったときに、復旧用として用意しています。

③ AsdConfigMenu モード
起動時に「/home/asdusr/autostart.sh」に記述されたプログラムのかわりに、ASD Config Menu が立ち上がるほかは、通常モードと同様です。

④ ReadOnly モード
「/boot」と「/home」と「/usr/local」以外のディレクトリをすべて ReadOnly として起動します。
「/home/asdusr/autostart.sh」に記述されたプログラムが自動的に起動します。

④の ReadOnly モードとして起動した後、「/bin」や「/lib」といったディレクトリに書き込みを行うと、書き込みができませんというエラーメッセージが表示されます。通常モードで再起動しない限り、重要なシステム領域は破壊されることはありません。

「/etc」以下の設定ファイルについては、一見、変更できているように見えます。しかし、実際には RAM ディスク上に「/etc」がマウントされており、m-SATA SSD 内の「/etc」には変更が反映されていません。そのため、電源を落とすと ReadOnly モード時に行った設定はすべて元の設定値に戻ります。これは、「/etc」以下が完全に書込みができないようになっていると、アプリケーション起動時にエラーとなる場合があるためです。

アプリケーションが完成し、通常運用とする場合には、ReadOnly モードで起動されることをおすすめします。

第３章 開発環境

本章では、Algonomix4 の開発環境について説明します。

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。

EC4A 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)。EC4A シリーズはターゲットマシンとして開発された商品である為、セルフコンパイル環境は用意していません。

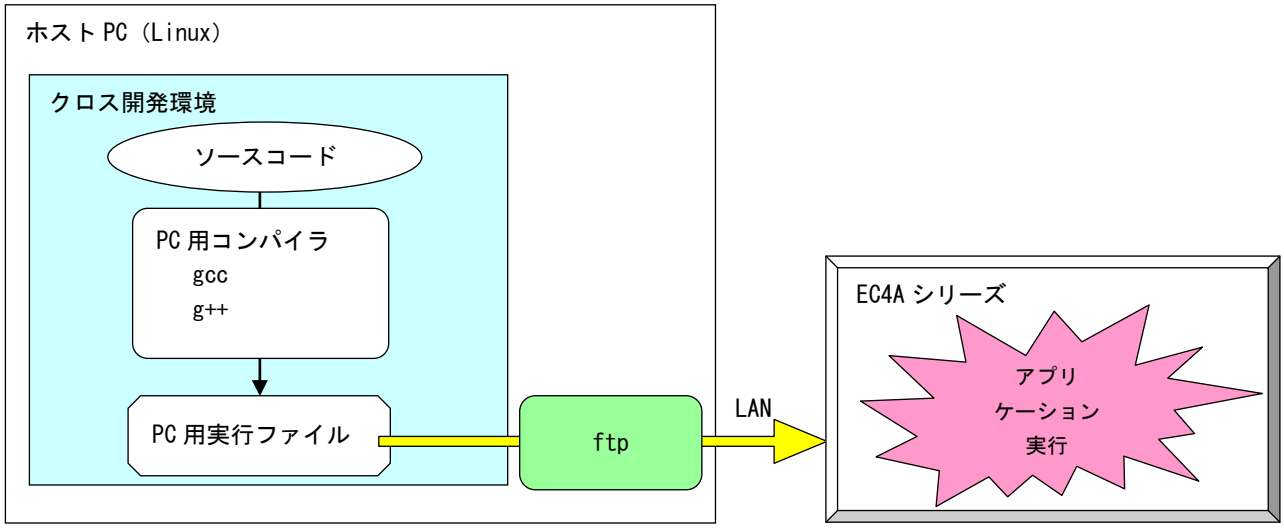


図 3-1-1. クロス開発方式イメージ図

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

表 3-1-1. クロス開発に必要なツール

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

Algonomix4 開発環境は、Ubuntu14.04.1 に表 3-1-1 のような開発ツールをあらかじめ組込んだものです。

VirtualBox という仮想マシン上で Algonomix4 開発環境を起動すれば、EC4A シリーズ用のアプリケーションを開発することが可能です。

３－２　　開発環境インストール

EC4A シリーズ用アプリケーション開発環境である **Algonomix4 用開発環境**を Oracle Corporation 製の VirtualBox 4.3.14 を用いて、Windows パソコン上で構築する手順について説明します。

３－２－１　　開発環境のインストールに必要なもの

開発環境のインストールに必要となるものは下記の 3 点です。

１．　Algonomix4 用開発環境 DVD-ROM

EC4A シリーズ用アプリケーション開発環境一式です。

DVD-ROM の内容を表 3-2-1-1 に示します。

※注：通常、開発環境 DVD-ROM は EC4A シリーズご購入時には添付しておりません。営業までお問い合わせいただきますと、無償で配布いたします。

表 3-2-1-1. Algonomix4 用開発環境 DVD-ROM の構成

DVD-ROM のディレクトリ	内容
doc	Algonomix4 の取扱い説明書が格納されています。 ・AS Series Algonomix4 Software Manual.pdf 本書のことです。
development	VirtualBox を使用せずに直接、パソコン上に開発環境を構築する場合の Algonomix4 用パッケージが格納されています。 ・Algonomix4-tools-0010.tar.gz
VirtualBox	VirtualBox 用の OS イメージファイルが格納されています。 ・Algonomix4_development.exe インストール方法は『3-2-4 仮想マシンの作成』を参照してください。

２．　VirtualBox Ver4.3.14

本書で使用しているものは VirtualBox ver4.3.14 です。VirtualBox のバージョンによっては、本書の画面表示と異なる可能性があります。VirtualBox は Oracle Corporation 社の製品です。VirtualBox の使用にあたっては Oracle Corporation 社の使用許諾条件に従ってご使用ください。

３．　開発環境インストール用 Windows PC

Windows7 が動作しており、VirtualBox がインストール可能なパソコンが必要です。VirtualBox のインストールおよび Algonomix4 開発環境 OS イメージを動作させる為に、最低限必要な環境として表 3-2-1-2 の PC スペックが必要になります。

表 3-2-1-2. 必須 PC スペック

CPU	X86 または X64（1GHz 以上を推奨） Intel, AMD
メモリ	256MByte 以上　（2GByte 以上推奨） **※注：VirtualBox はメモリ領域が 256MByte 未満の場合、正常に動作しない場合があります。その為、必須環境を満たしているパソコンでも、ホスト OS 上で他のソフトウェアを同時に起動している場合、メモリ不足により VirtualBox が正常に動作しない事があります。その場合は他のソフトウェアを一度停止させメモリ領域を開放した上で、もう一度 VirtualBox を起動してください。**
HDD 空き領域	10GByte 以上　（20GByte 以上推奨） **※注：Algonomix4 開発環境では、ゲスト OS のイメージファイルは最大容量 20GByte の可変長の OS イメージとなっています。初期状態では OS イメージファイルは 8GByte ほどの大きさですが、開発をしていくにつれこのサイズは大きくなる為、ゲスト OS の空き領域を超えてしまう場合、正常に動作しなくなる場合があります。その場合はホスト OS の空き領域を増やすか、ゲスト OS 内の不要なファイルを消去し、空き領域を確保してください。**
OS	Windows 7
ファイルシステム	NTFS **※注：ファイルシステムが FAT32 の環境では単独でファイルサイズが 4GByte を超えるファイルは使用できません。Algonomix4 開発環境のゲスト OS のイメージファイルは 8GByte を超えてしまう為、FAT32 上の環境では Algonomix4 開発環境は使用できません。**
その他	DVD-ROM ドライブ（インストールディスク読込み用）、 USB ポート、LAN ポート（EC4A シリーズとパソコンの接続に最低でもいずれかひとつが必要になります）

Algonomix4 開発環境をインストールするパソコンで動作している OS（Windows 7）と VirtualBox 上で動作する OS（Lubuntu 14.04.1）を区別する為、パソコン側の OS を「ホスト OS」、VirtualBox 上の OS を「ゲスト OS」と表記しています。

３－２－２　　VirtualBox のダウンロード

VirtualBox のインストーラは **http://www.virtualbox.org/**よりダウンロードします。

①　VirtualBox 公式ページの左メニューから[Downloads]を選択します。

図 3-2-2-1.　公式 TOP ページ

②　図 3-2-2-2 の使用するホスト OS の環境を選択する画面に変わります。
[VirtualBox older builds]を選択します。
なお、この画面は VirtualBox のバージョンアップに伴い、変化する可能性があります。

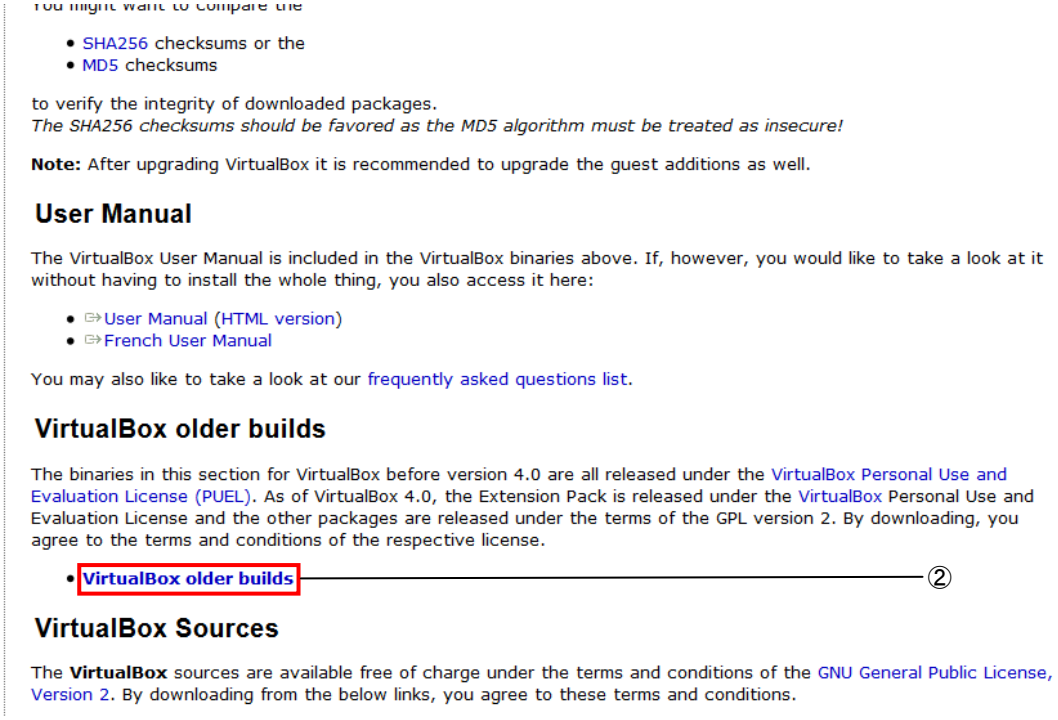


図 3-2-2-2.　VirtualBox ダウンロードページ

③　　最新版が 4.3.14 でない場合、Virtual Box 4.3 をクリックしてください。Virtual Box 4.3 系のリストが並んでいますので、『VirtualBox 4.3.14 for Windows hosts x86/AMD64』という項目をクリックしてください。

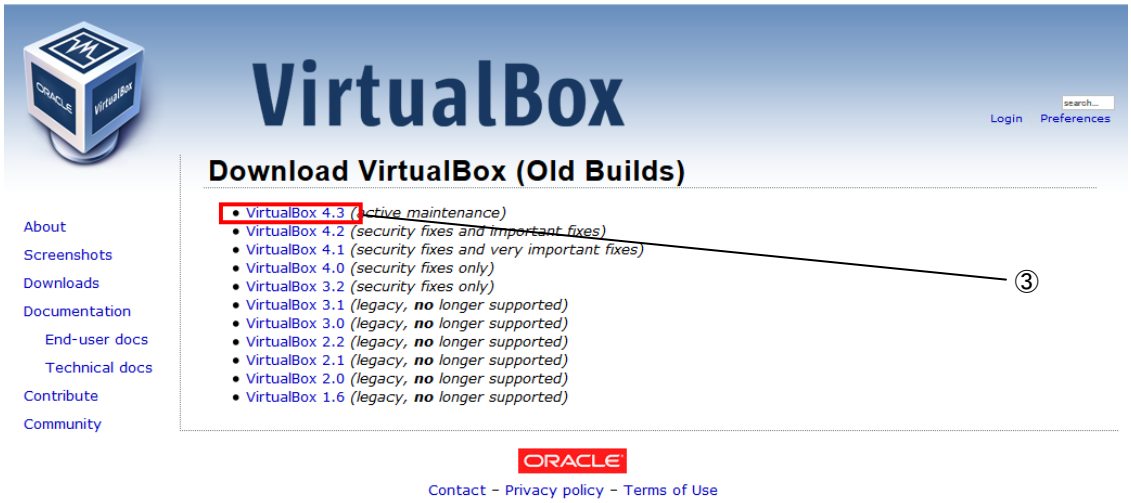


図 3-2-2-3.　Virtual Box Old Builds ダウンロードページ

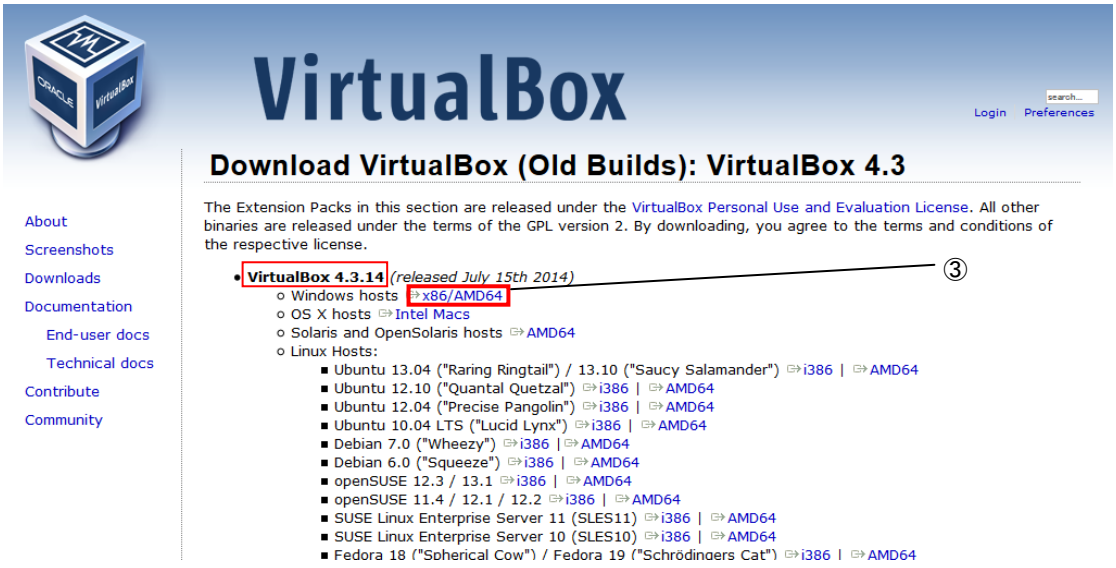


図 3-2-2-4.　古いバージョンの VirtualBox ダウンロードページ

④　　Windows のダウンロード確認画面が表示されるので、任意のフォルダを指定し、[保存(S)]を選択します。ダウンロードが開始されます。

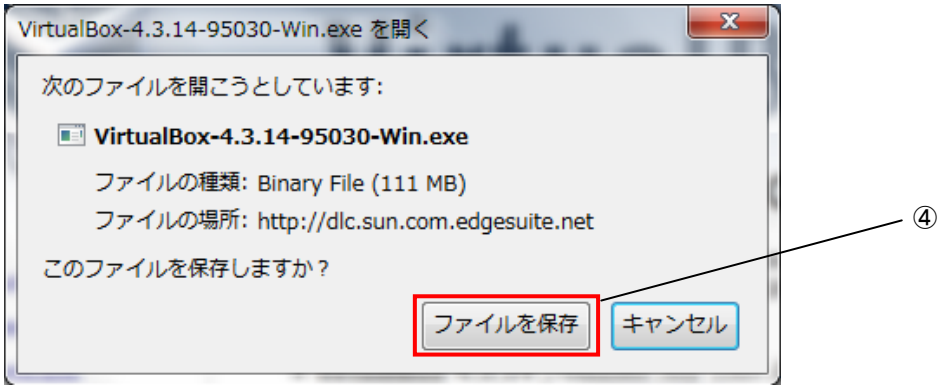


図 3-2-2-5. ダウンロード確認画面

３－２－３　　VirtualBox のインストール

①　　『3-2-2 VirtualBox のダウンロード』でダウンロードした、
VirtualBox-4.3.14-XXXXX-Win_x86.msi のアイコンをダブルクリックします。

②　　図 3-2-3-1 のセキュリティ警告画面が開きますので、[実行する(R)]をクリックします。

図 3-2-3-1.　セキュリティ警告画面

③　　図 3-2-3-2 のような画面が表示されるので、[Next>]をクリックします。

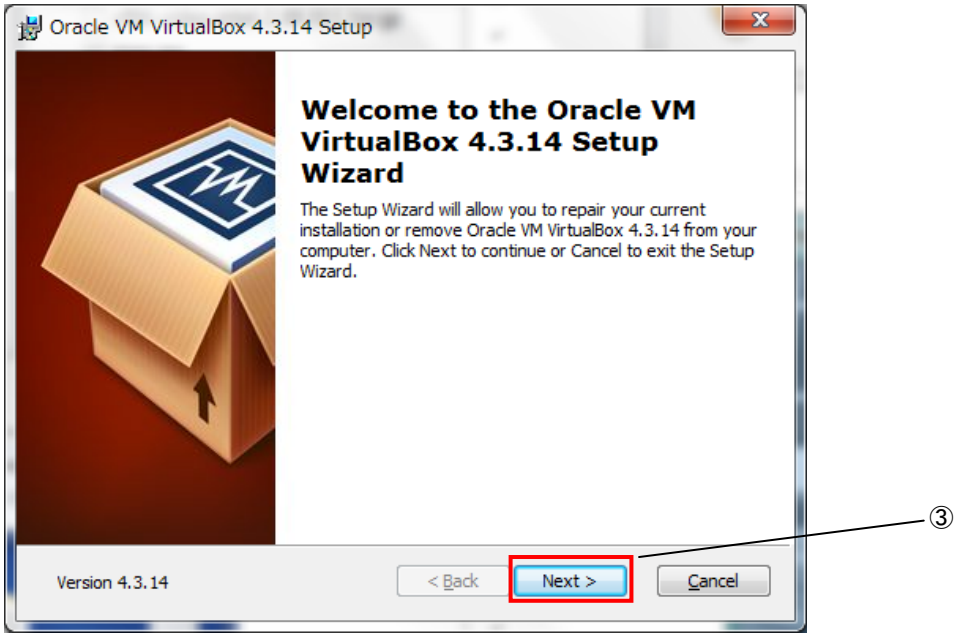


図 3-2-3-2.　ダウンローダー起動後の画面

④　図 3-2-3-3 のカスタムセットアップ画面に変わります。ここでは VirtualBox に追加するプラグイン機能を選択します。

・VirtualBox USB Support：VirtualBox 上で USB 機能を使えるようにするプラグインです。
・VirtualBox Networking　：VirtualBox 上でネットワークへ接続する為のプラグインです。

本製品を使用するに当たって、両方のプラグインをインストールする必要があります。デフォルトではインストールする設定になっています。

⑤　[Next>]をクリックします。

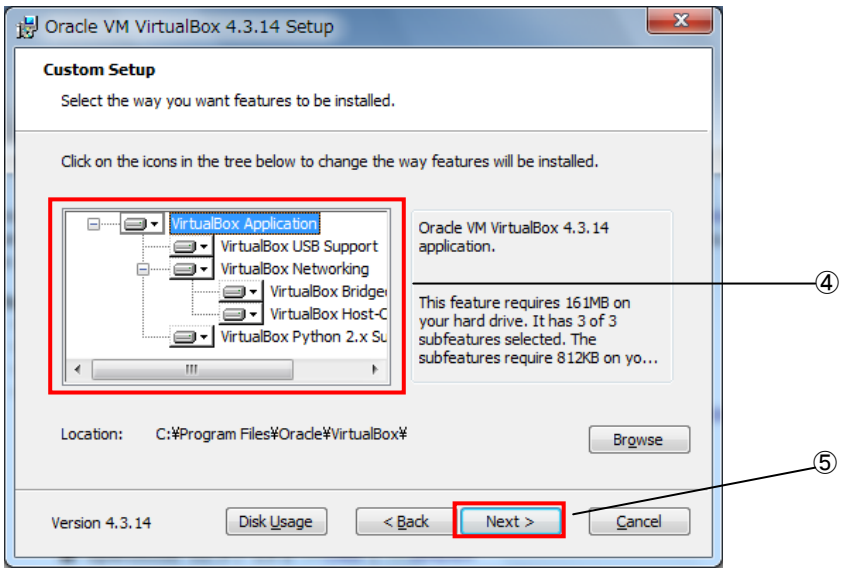


図 3-2-3-3. カスタムセットアップ画面

⑥　図 3-2-3-4 のインストールオプション設定画面に切り替わります。インストールオプションを設定できます。[Next>」をクリックします。

・Create a shortcut on the desktop　：　ショートカットをデスクトップに配置します。
・Create a shortcut in the Quick Launch Bar　：　ショートカットをクイックランチャーに配置します。
・Register file associations　：　ファイルの関連付けを登録します。

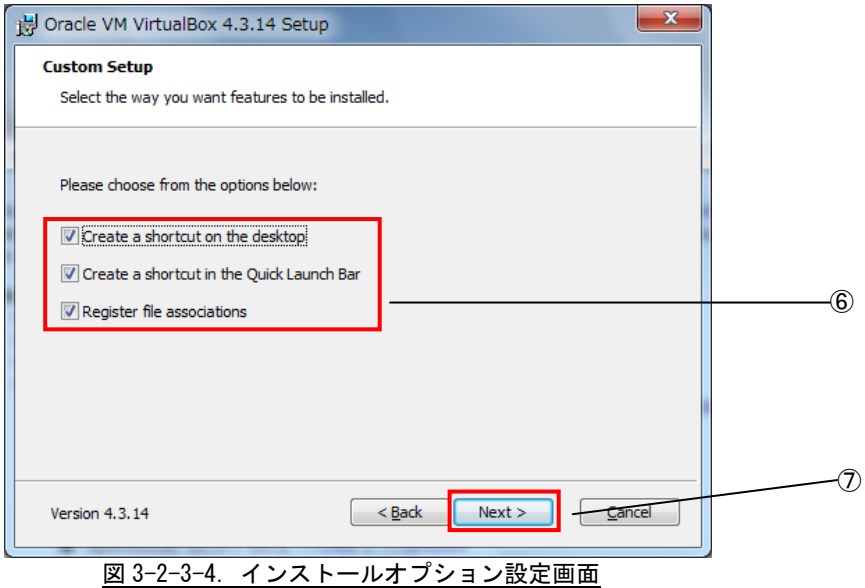


図 3-2-3-4. インストールオプション設定画面

⑦　　図 3-2-3-5 のネットワークインストール確認画面に切り替わりますので、[Yes]をクリックします。

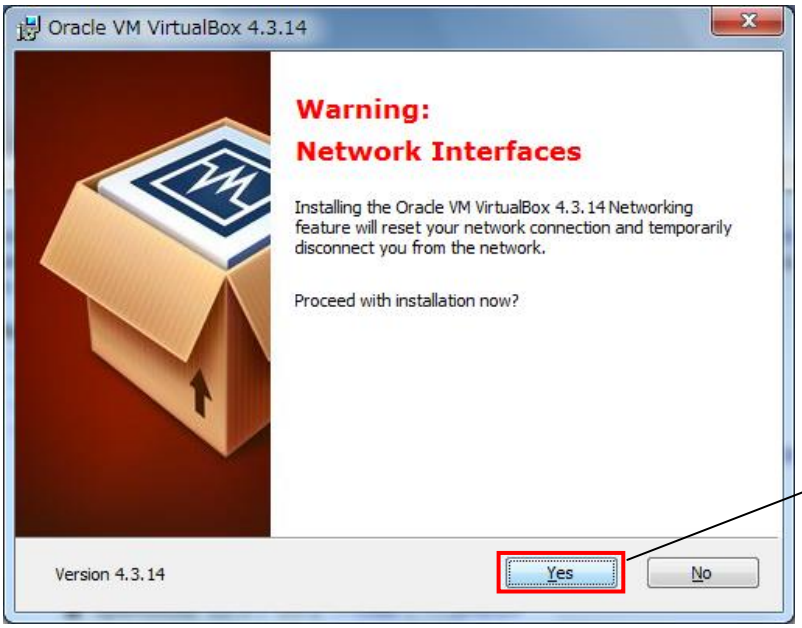


図 3-2-3-5. ネットワークインストール確認画面

⑧　　図 3-2-3-6 のインストール確認画面に切り替わりますので、[Install]をクリックします。

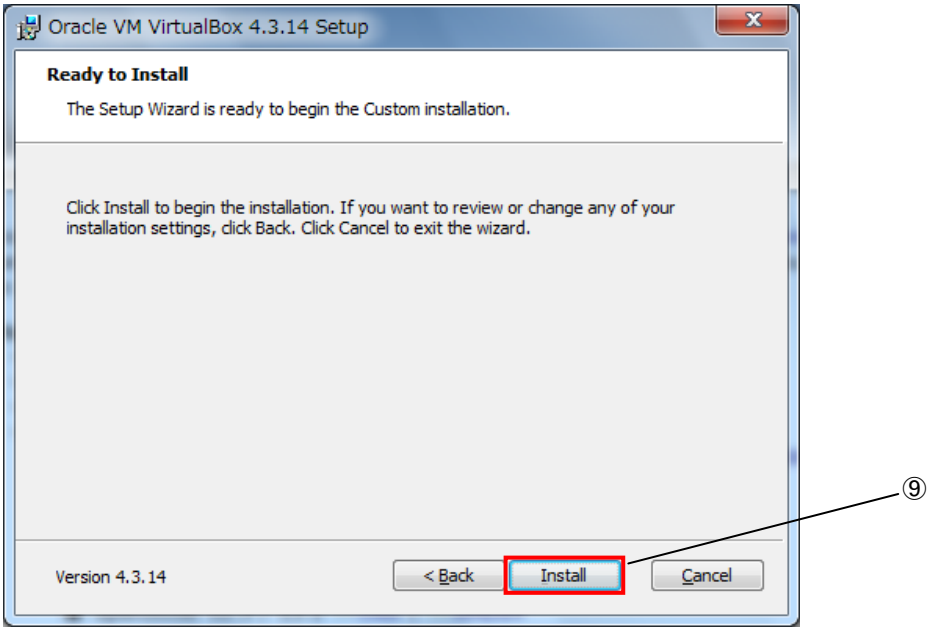


図 3-2-3-6. インストール確認画面

⑨　　インストール中、図 3-2-3-7 のようなデバイスドライバのインストールの実行を確認する画面が開きますが、いずれも[インストール]をクリックします。

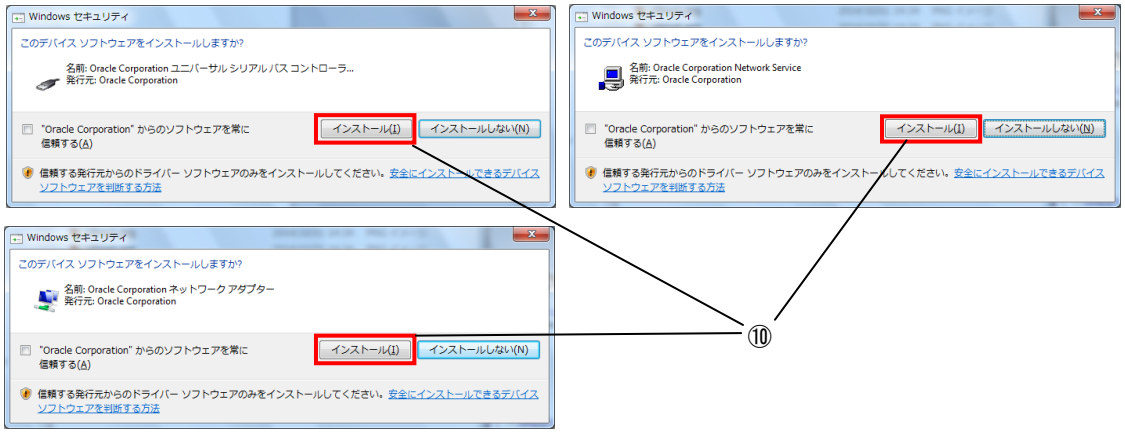


図 3-2-3-7.　ドライバのインストール警告画面

⑩　　インストールが完了すると、図 3-2-3-8 が表示されます。[Finish]をクリックします。

図 3-2-3-8.　インストール完了画面

３－２－４　　仮想マシンの作成

作成する仮想マシンの構成を指定します。

① [スタート]→[全てのプログラム]→[Oracle VM VirtualBox]→[Oracle VM VirtualBox]を選択します。

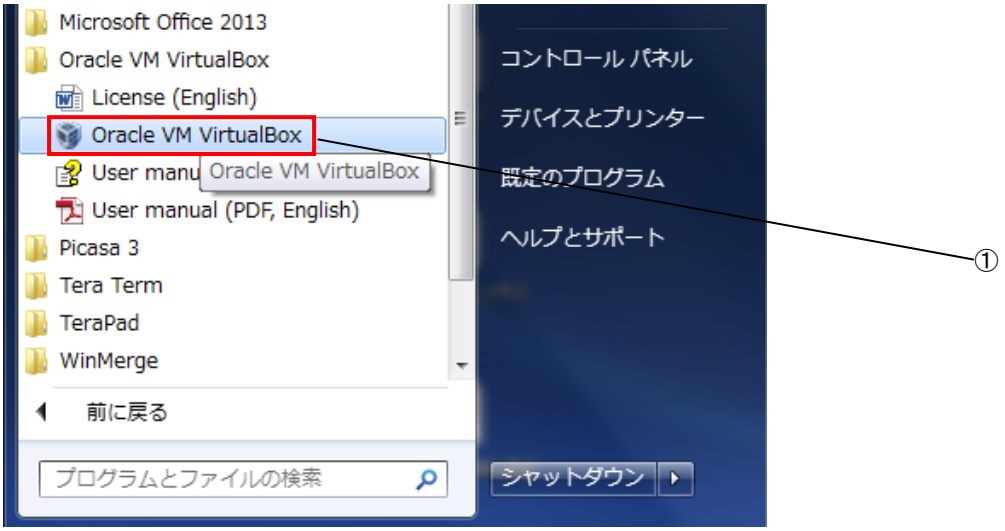


図 3-2-4-1. VirtualBox 起動

② VirtualBox の初回起動時に下記のような設定ファイルが作成されます。

C:¥User¥<ユーザ名>¥.VirtualBox

このフォルダは VirtualBox の OS イメージであるファイルを格納するフォルダです。弊社より配布されている DVD の<DVD>¥VirtualBox¥Algonomix4_development.exe をこのフォルダに展開します（Algonomix4_development.exe を実行することで、展開できます）。

Algonomix4_development.exe を展開することで Algonomix4_development.vdi が作成されます。

③ VirtualBox が起動し、図 3-2-4-2 のような VirtualBox メイン画面が表示されるので、「新規(N)」をクリックします。

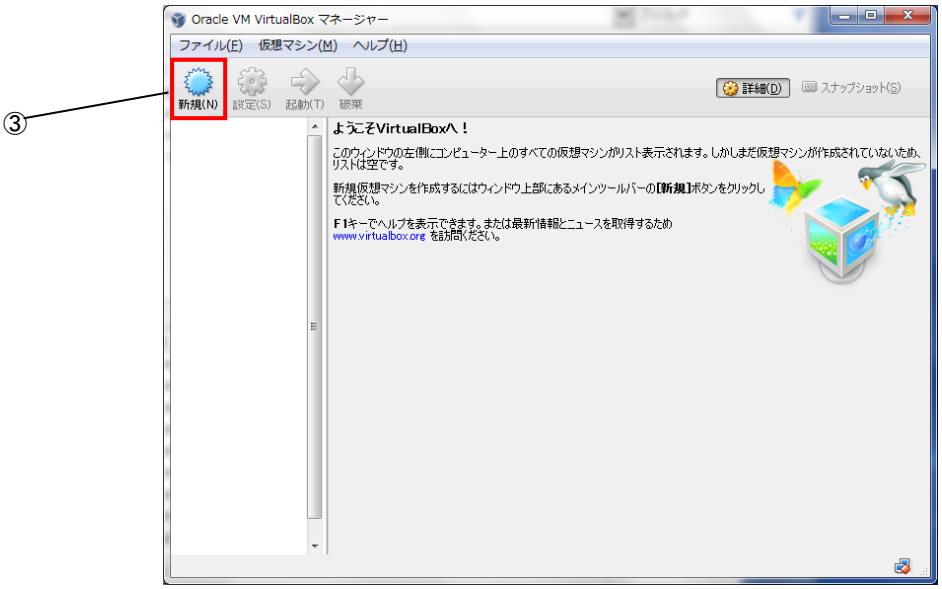


図 3-2-4-2. VirtualBox メイン画面

④　図 3-2-4-3 のマシン名と OS 種類選択画面に変わるので、[名前(N)]には[Algonomix4_development]と入力します。
⑤　[OS タイプ(T)]では[Linux][Ubuntu(32bit)]を選択します。
⑥　[次へ(N)]をクリックします。

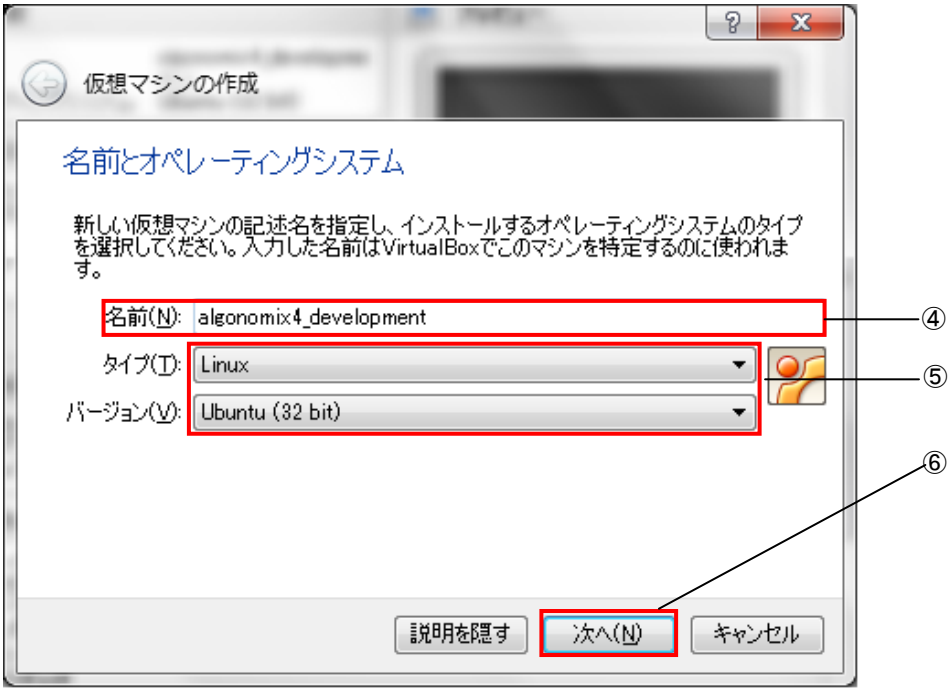


図 3-2-4-3.　マシン名、OS タイプの設定

⑦　図 3-2-4-4 のようなメモリ設定画面に変わります。仮想マシンのメモリサイズを入力します。
※注：ご使用のパソコンに応じてご指定ください。必須環境は 512MByte 以上です。
⑧　[次へ(N)]をクリックします。

図 3-2-4-4.　メモリの設定

⑨　　図 3-2-4-5 の仮想ハードディスクのイメージ選択画面へ変わるので、[すでにある仮想ハードドライブを使用する]をクリックしてください。

⑩　　プルダウンメニューに algonomix4_development.vdi がない場合、  をクリックして⑪に進みます。プルダウンメニューに algonomix4_development.vdi がある場合、[作成]をクリックし、⑬へ進んでください。

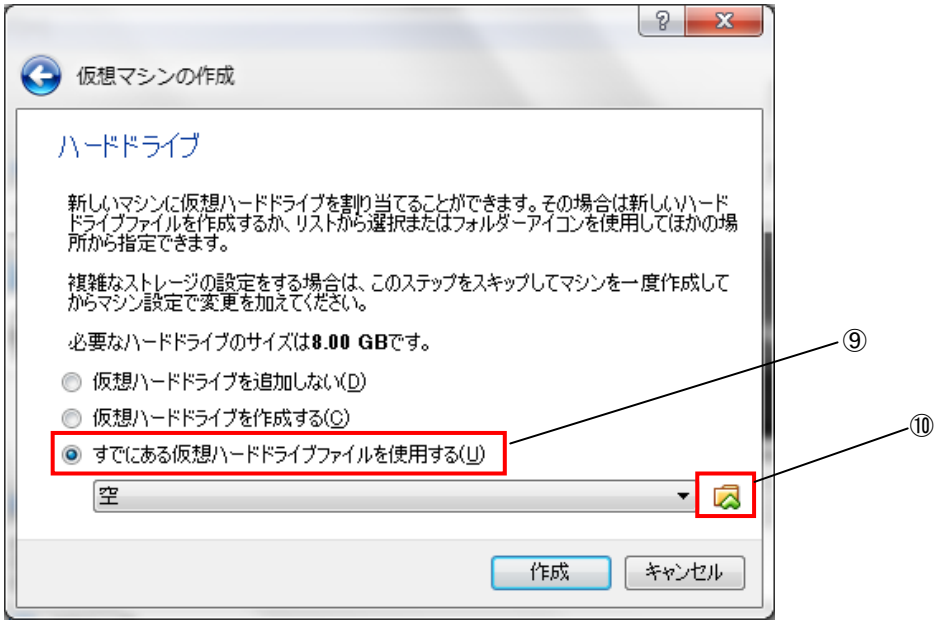


図 3-2-4-5.　仮想ハードディスクのイメージ選択画面

⑪　　図 3-2-4-6 のように仮想ディスク選択画面が開きます。

②であらかじめ展開しておいた Algonomix4_development.vdi を選択し、[開く(O)]を押してください。

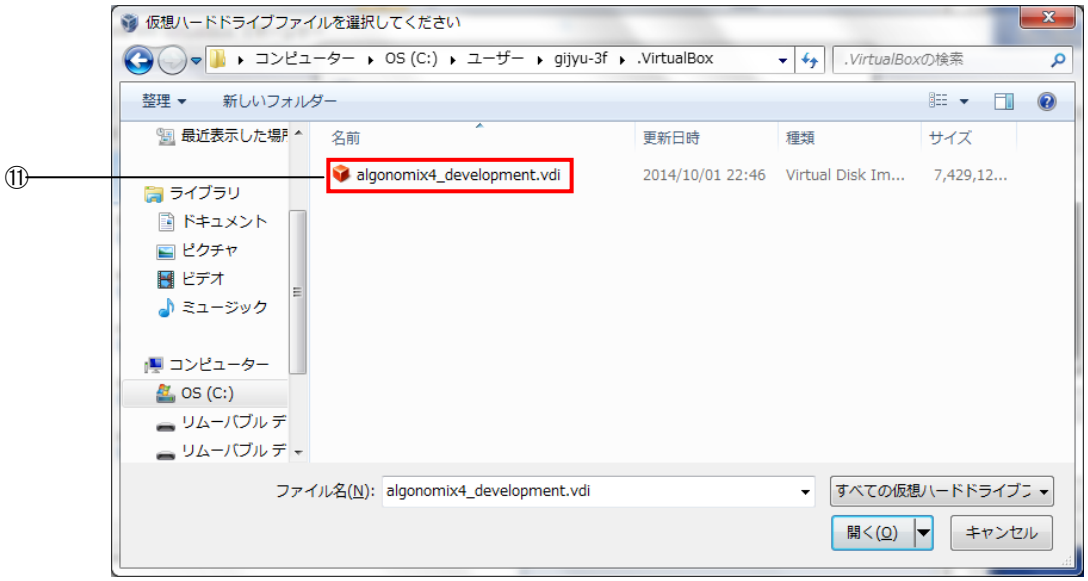


図 3-2-4-6.　仮想ディスク選択画面

⑫　　仮想ハードディスクのイメージ選択画面に戻ります。[作成]を押してください。

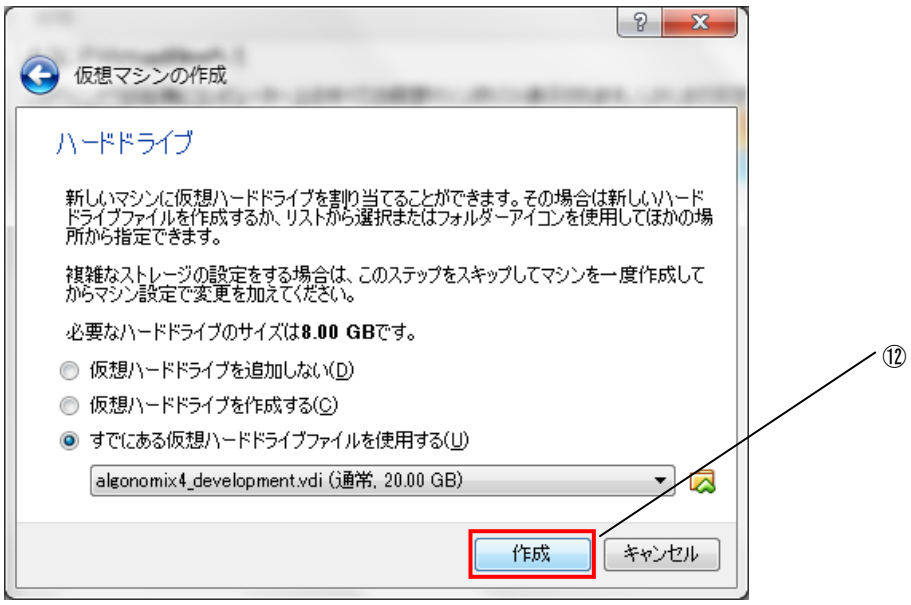


図 3-2-4-7.　仮想ハードディスクのイメージ選択画面

３－２－５　　仮想マシンの起動

弊社配布の OS イメージ、Algonomix4_development.vdi は既に Algonomix4 開発環境がインストールされています。なお、Algonomix4_development.vdi のユーザ名とパスワードは、初期状態では下記のように設定されています。ログイン時やシステム変更時にユーザ名、パスワードの入力を求められるので、下記のユーザ名とパスワードを入力します。

ユーザ名　　：asdusr
パスワード　：asdusr

①　VirtualBox メイン画面左側の List の中から Algonomix4_Development を選択します。
②　[起動(T)]をクリックします。

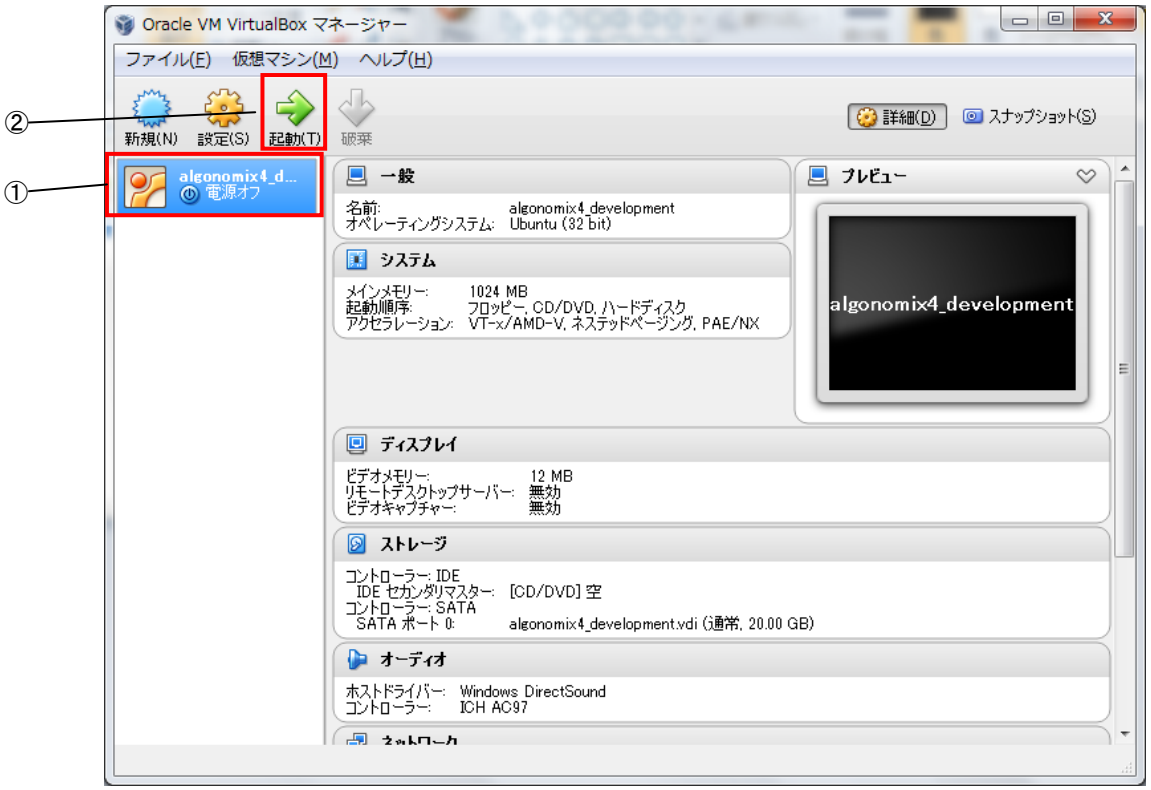


図 3-2-5-1.　仮想マシンの起動

図 3-2-5-2 のような仮想マシンのユーザ名、パスワード入力画面が起動します。

③　ユーザ名とパスワードの入力を求められるので、下記のユーザ名とパスワードを入力します。

ユーザ名　　：　asdusr

パスワード　：　asdusr

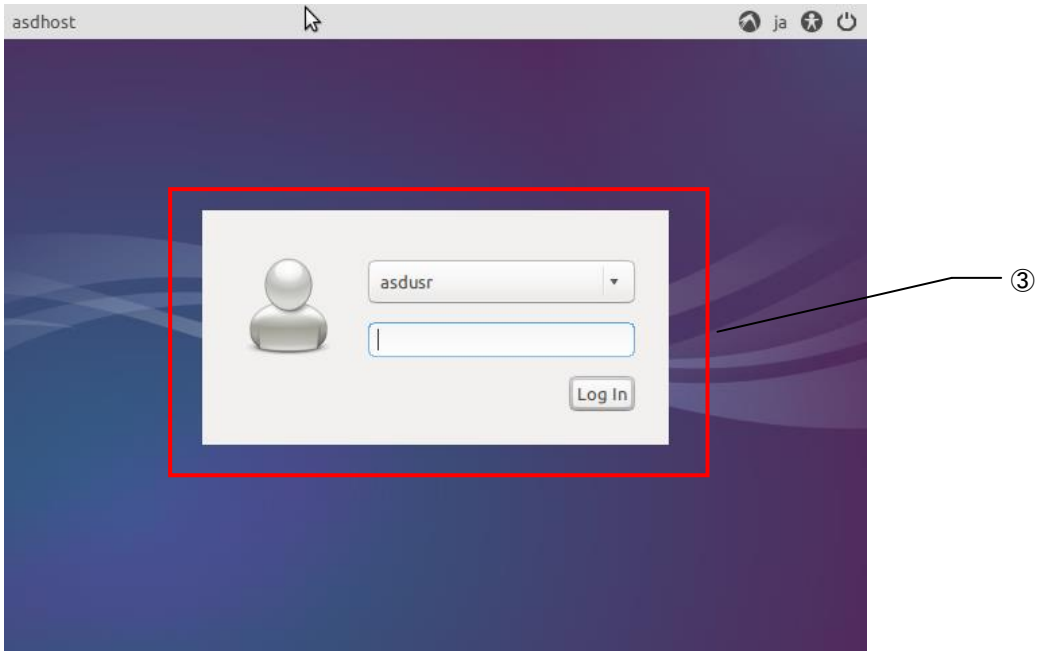


図 3-2-5-2.　ユーザ名、パスワード入力画面

図 3-2-5-3 のような仮想マシンのデスクトップ画面になり、ゲスト OS が使用できるようになります。

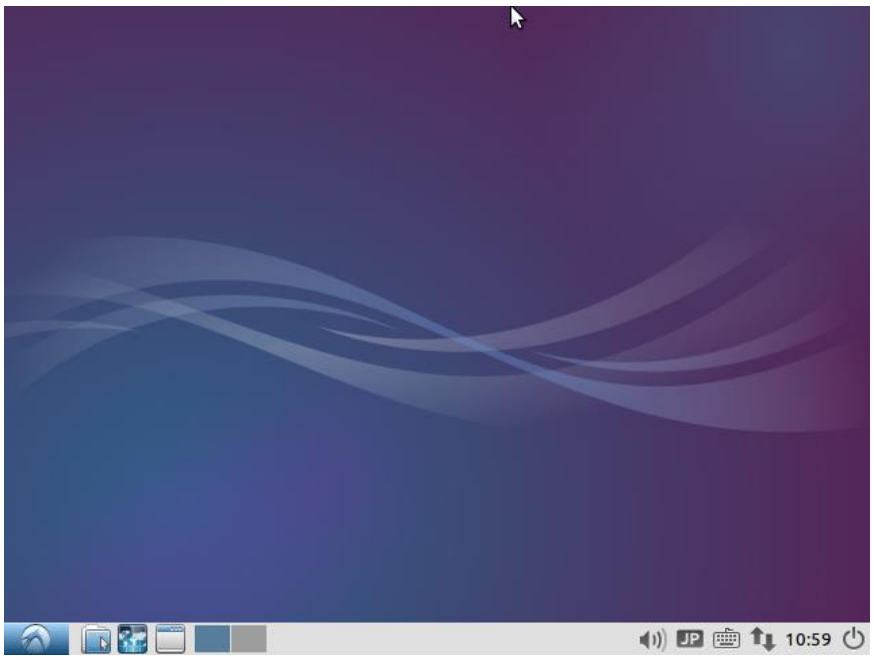


図 3-2-5-3.　デスクトップ画面

３－２－６　　開発環境の各種設定について

ここでは、開発環境を起動後に設定する項目について説明します。

● ネットワーク設定

ゲスト OS でネットワークを使用する際、もし接続できない場合や、IP 固定で使用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。

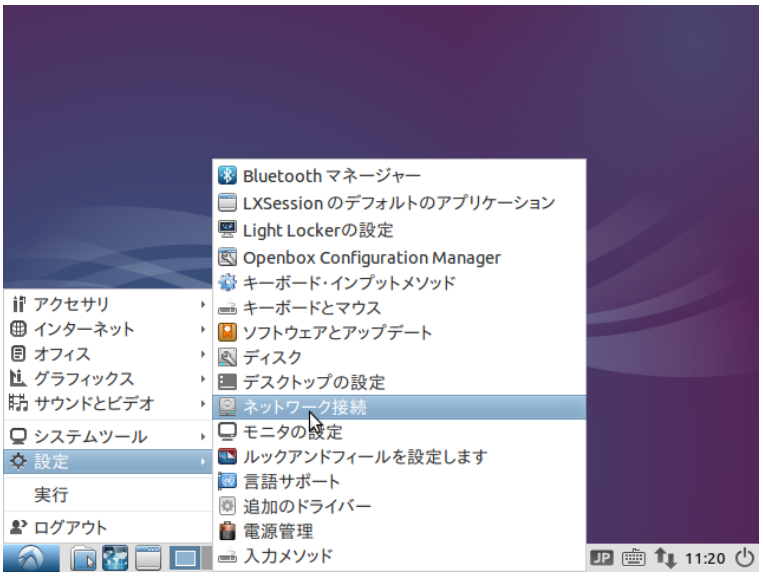


図 3-2-6-1. ネットワーク設定画面の起動

②　図 3-2-6-2 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。

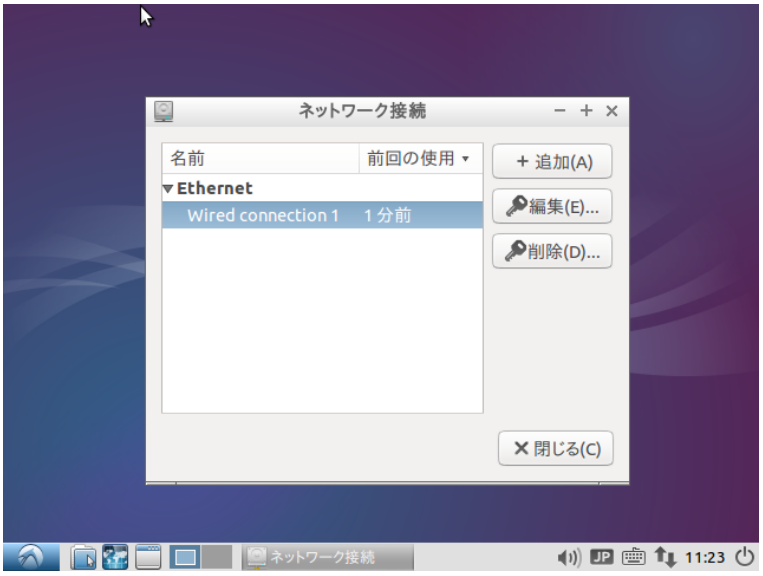


図 3-2-6-2. ネットワーク設定画面

③　図 3-2-6-3 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。

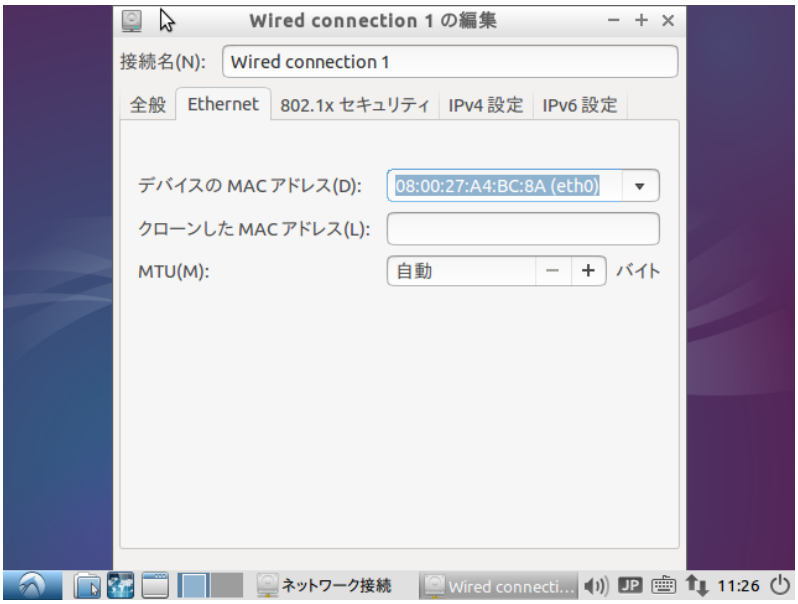


図 3-2-6-3. 有線 LAN 設定画面

④　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

```
$ sudo su
パスワード:asdusr
# ifconfig
```

※注：パスワードは出荷時のものです。また、パスワードは実際には表示されません。

正常に接続されない場合は、下記のコマンドで一端接続を切断し、再接続してみてください。

・指定したデバイスを切断する場合

```
# ifconfig eth0 down
```

・指定したデバイスを接続する場合

```
# ifconfig eth0 up
```

※注：eth0 はデバイス名です。認識しているデバイスの状態は、「ifconfig」コマンドで確認できます。

● USB の認識

仮想マシン上で USB 機器を使用する為には、ゲスト OS を起動する前に、Windows 上で使用する USB 機器のドライバがインストールされており、正常に動作している必要があります。

ゲスト OS が起動している状態で新規の USB 機器を接続されると、その USB 機器は正常に動作しません。一度、ゲスト OS をシャットダウンし、USB 機器を挿入しなおして、Windows 用のドライバをインストールしてください。その後、ゲスト OS を起動することで、新規の USB 機器を使用することができます。

VirtualBox で認識している USB 機器は[デバイス(D)]→[USB デバイス(U)]をクリックすることで一覧表示されます(図 3-2-6-4 参照)。チェックが付いているものはゲスト OS が認識しているものです。このチェックを ON/OFF することで、ゲスト OS から USB 機器を抜き差しすることができます。

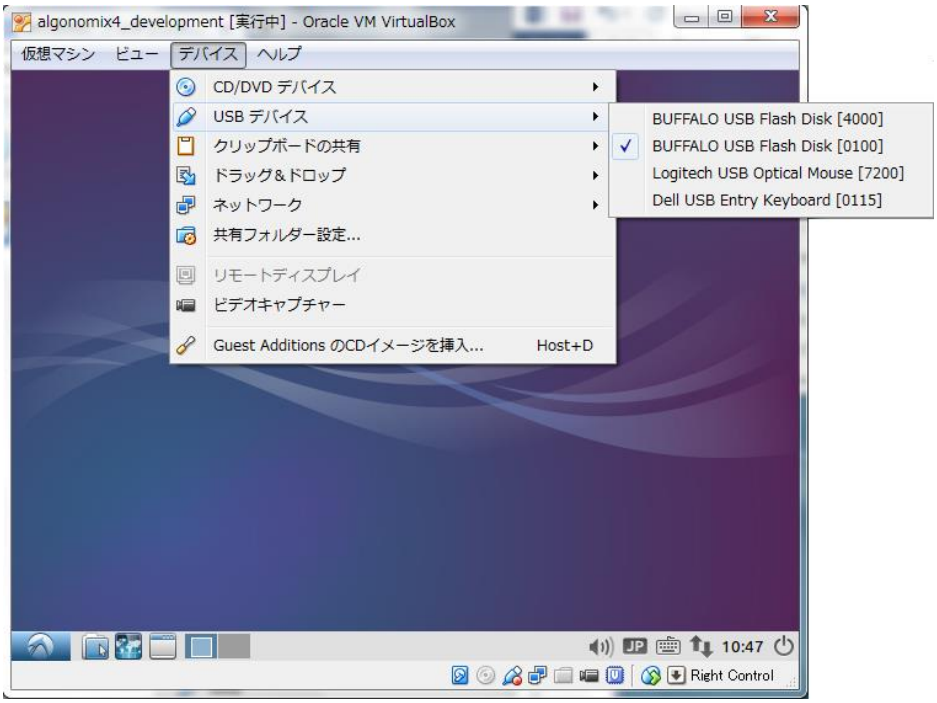


図 3-2-6-4. USB デバイス一覧

３－２－７　　Algonomix4 用開発環境のディレクトリ構成について

Algonomix4 用開発環境のディレクトリ構成について説明します。Algonomix4 用開発環境のディレクトリ構成をリスト 3-2-7-1 に示します。

リスト 3-2-7-1. Algonomix4 用開発環境のディレクトリ構成

```
usr--+--local--+--tools-0010--+--src--+--apl
                                       +--kernel
                                       +--pci
                                            :
                                  +--samples-+--sampleConsole
                                             +--sampleWideStudio
                 +--ws
```

また、これらのディレクトリの内容を表 3-2-7-1 に示します。

表 3-2-7-1. ディレクトリの内容（抜粋）

ディレクトリ名	内　容
/usr/local/tools-0010	Algonomix4 用の開発環境が格納されています
/usr/local/tools-0010/src	EC4A シリーズ用のドライバ等のソースファイルが格納されています。
/usr/local/tools-0010/src/apl	EC4A シリーズ用のアプリケーションです。 ASD コンフィグツールのソースが格納されています。
/usr/local/tools-0010/src/kernel	Algonomix4 用のカーネルソースです。
/usr/local/tools-0010/src/widestudio	本開発環境にインストールしている WideStudio ソースおよび Algonomix4 用のライブラリです。
/usr/local/tools-0010/src/pci	EC4A シリーズ用の PCI ドライバです。
/usr/local/tools-0010/samples	Algonomix4 用のサンプルプログラムです。
/usr/local/ws	WideStudio 本体です。

３－３　　WideStudio/MWT によるアプリケーション開発

WideStudio/MWT はデスクトップアプリケーションを迅速に作成することのできる統合開発環境です。詳細は WideStudio ホームページ（http://www.widestudio.org/index.html）を参照してください。

Algonomix4 用開発環境に組込まれている WideStudio/MWT は V3.98-6 をベースに Algonomix4 用の環境設定を加えてコンパイルしたものです。ここで、WideStudio/MWT で簡単なプログラムをコンパイルして実際に動作させます。

３－３－１　　WideStudio の起動

デスクトップ上の 

のアイコンをダブルクリックすることで WideStudio が起動します。

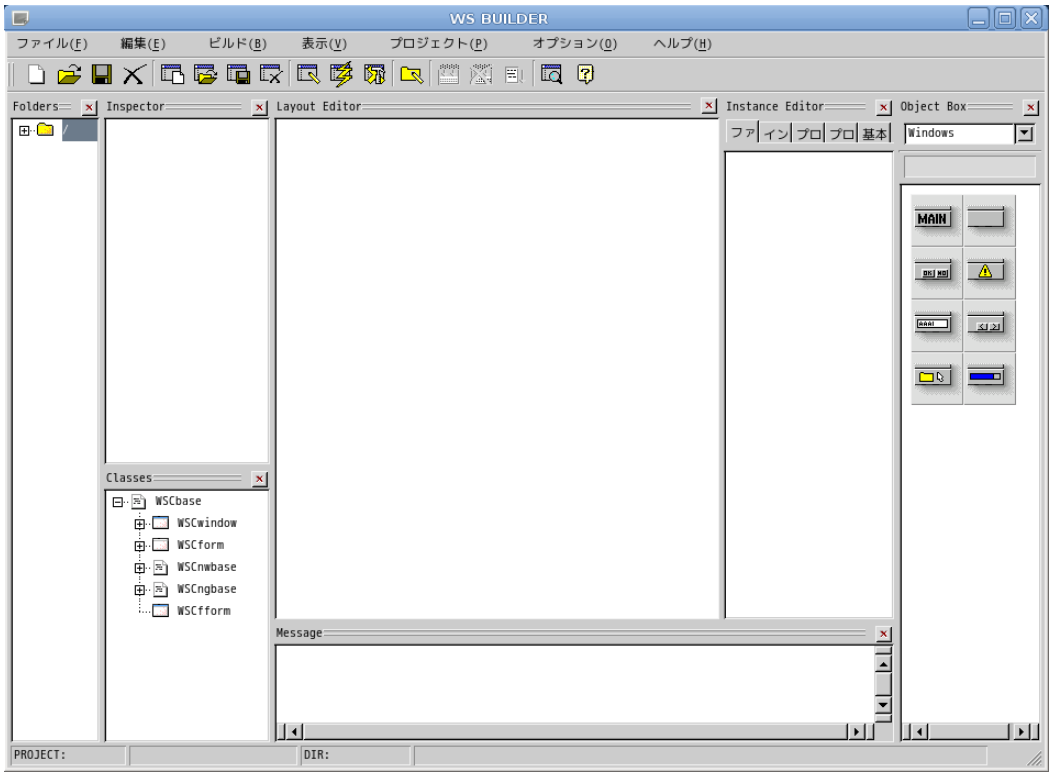


図 3-3-1-1. WideStudio 起動

３－３－２　　プロジェクトの新規作成

アプリケーションを作成するためのプロジェクトを以下の手順で作成します。

図 3-3-2-1. 新規プロジェクト作成

 をクリックするか、メニューの「プロジェクト(P)」→「新規プロジェクト(N)」をクリックすることで、図 3-3-2-2 のような画面が表示されます。ここでプロジェクト名称を記入してください。本項ではデフォルトの newproject とします。

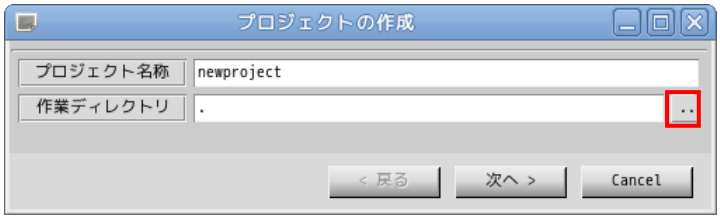


図 3-3-2-2. プロジェクト作成画面

 をクリックすることで、図 3-3-2-3 のようなファイル選択ダイアログが表示されます。

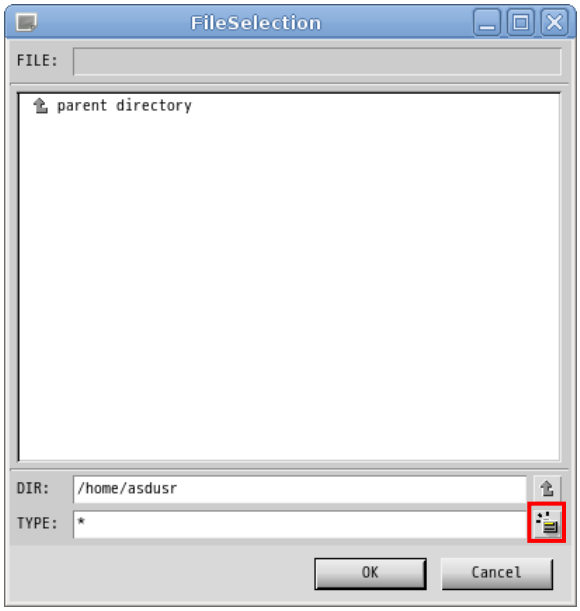


図 3-3-2-3. ファイル選択画面

DIRに「/home/asdusr」を指定します。sample というディレクトリを作成するために、  をクリックし、「sample」と入力し「OK」ボタンをクリックします。

※注：入力はマウスカーソルを入力ダイアログ上に持っていった上で行ってください。

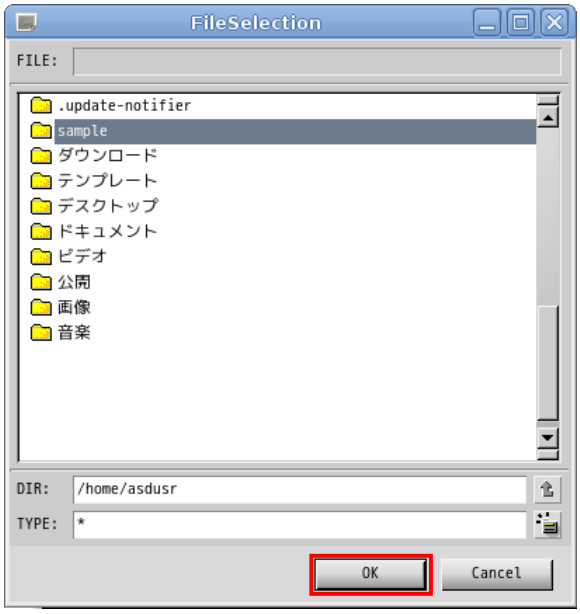


図 3-3-2-4. sample ディレクトリの作成

sample ディレクトリをダブルクリックし、DIR が「/home/asdusr/sample」と指定されたことを確認して「OK」ボタンをクリックします。

プロジェクト名と作業ディレクトリの指定が完了しましたので「次へ」ボタンをクリックします。

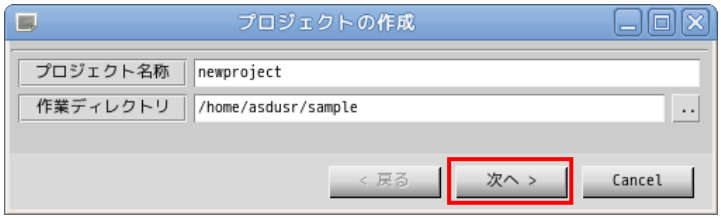


図 3-3-2-5. プロジェクト名と作業ディレクトリの設定完了

プロジェクトの種別を選択します。通常のアプリケーションを作成するので、そのまま「次へ」をクリックします。

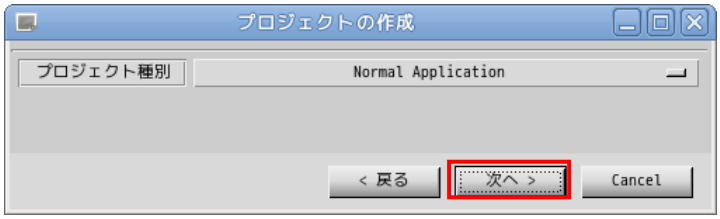


図 3-3-2-6. プロジェクト種別の選択

アプリケーションで使用するロケール種別（文字コード）、言語種別（プログラミング言語）を選択します。本項では、ロケール種別に「UTF8」を、言語種別に「C/C++」を選択します。

Algonomix4 で動作確認したロケール種別は、ユニコード(UTF8)と日本語(EUC)と日本語(SJIS)のみです。また、言語種別は C/C++のみサポートしています。

※注：ロケール種別については注意が必要です。例えば、シリアル通信を通して、SJIS の文字列が送られてきてそれを表示する場合、ここでのロケール種別は SJIS にします。

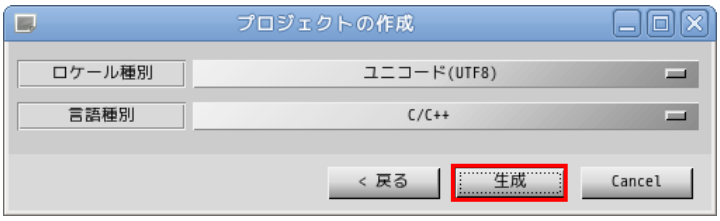


図 3-3-2-7.　ロケール種別と言語種別を選択

以上でプロジェクトの作成は完了です。

３－３－３　　アプリケーションウィンドウの作成

前項でプロジェクトの作成が完了しましたので、次にアプリケーションウィンドウを作成します。

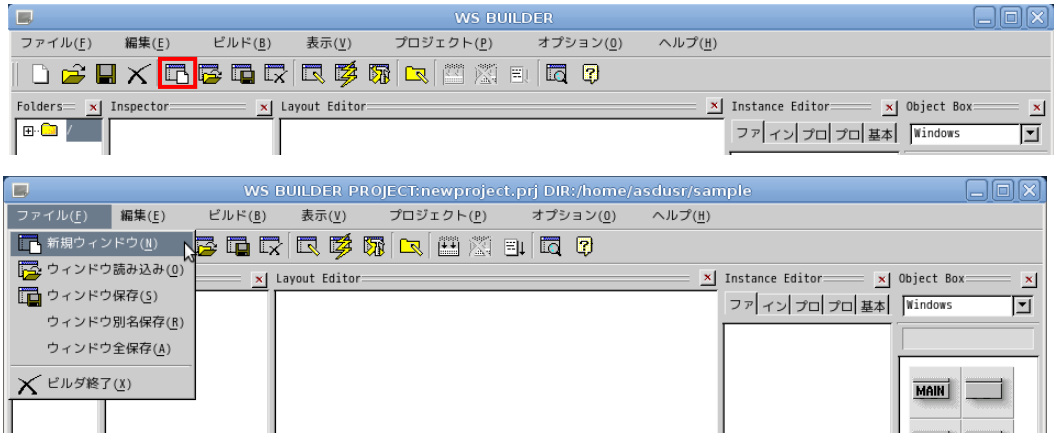


図 3-3-3-1.　新規ウィンドウ作成

 をクリックするか、メニューの「ファイル(F)」→「新規ウィンドウ(N)」をクリックすることで、図 3-3-3-2 の画面が表示されます。「通常のウィンドウ」を選択し、「次へ >」ボタンをクリックします。

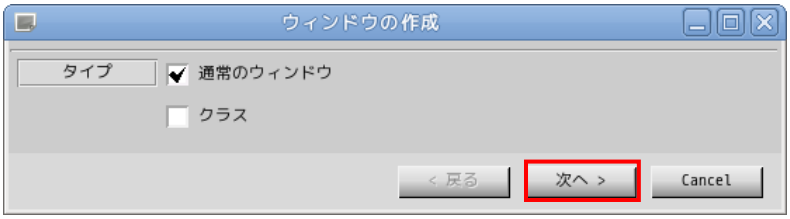


図 3-3-3-2.　アプリケーションウィンドウタイプ選択

アプリケーションウィンドウの名称とプロジェクトに登録するかを選択します。名称は変数として使われるので空白のない英数字のみ有効です。ここではデフォルト設定のままとします。設定を変更せずに「次へ 〉」をクリックします。

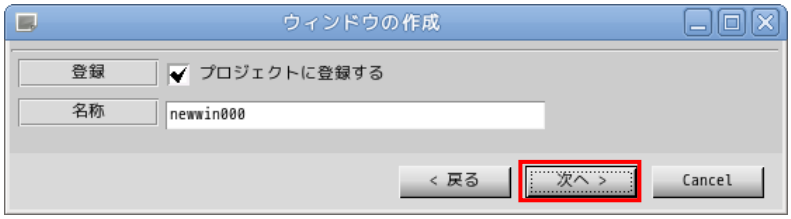


図 3-3-3-3. アプリケーションウィンドウ名称設定

テンプレートの選択を行います。あらかじめ用意されたテンプレートを選択することで、標準的なメニューやツールバーを持つアプリケーションウィンドウを作成することができます。今回はメニューを持たないウィンドウを作成するので、「なし」を選択して「生成」ボタンをクリックします。

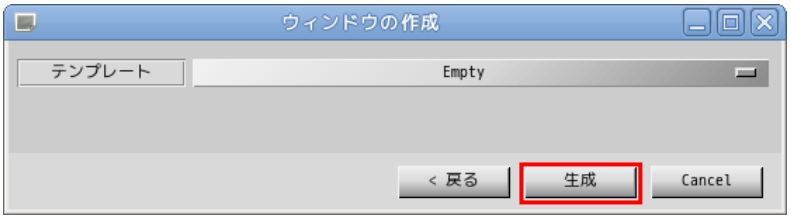


図 3-3-3-4. テンプレートの選択

これで、アプリケーションウィンドウが作成できました。作成したアプリケーションウィンドウをクリックし、「Instance Editor」の「プロパティ」タブをクリックすることで、アプリケーションウィンドウのプロパティが設定できるようになります。表 3-3-3-1 を参考にプロパティを変更してください。

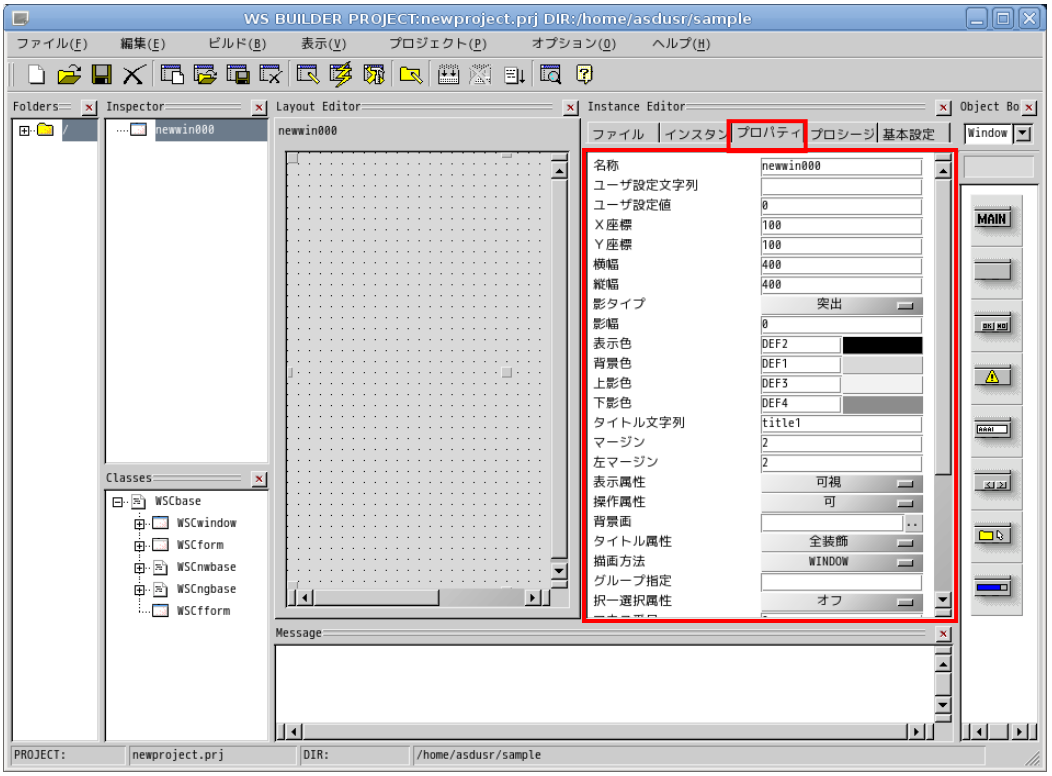


図 3-3-3-5. アプリケーションウィンドウの生成

表 3-3-3-1. アプリケーションウィンドウのプロパティ変更

プロパティ名	説明	設定値
X 座標	ウィンドウの初期起動 X 位置	0
Y 座標	ウィンドウの初期起動 Y 位置	0
横幅	ウィンドウの横幅	200
縦幅	ウィンドウの縦幅	200

以上で、アプリケーションウィンドウの作成は完了です。

３－３－４　　部品の配置

アプリケーションウィンドウの上に部品を配置します。

ボタンを配置しますので、「Object Box」の「Commands」を選択し、ボタンオブジェクトをアプリケーションウィンドウにドラッグ＆ドロップで配置します。

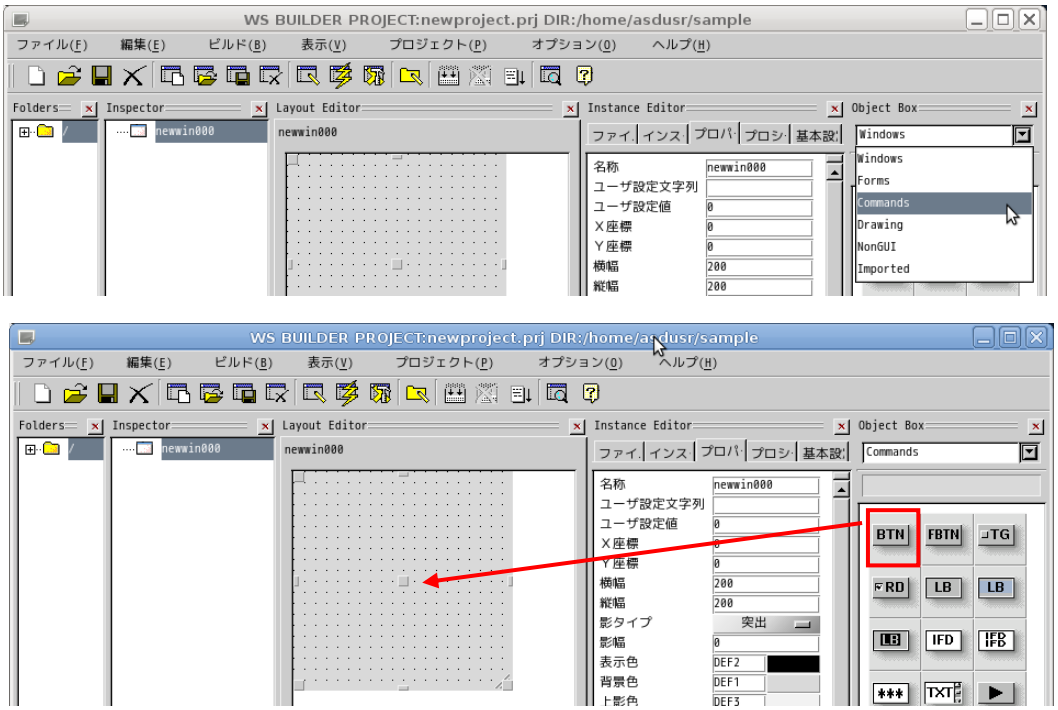


図 3-3-4-1. ボタンの配置

図 3-3-4-2 のようにボタンが配置されます。他の部品も同じようにオブジェクトボックスから目的のアイコンを選び出し、ドラッグ＆ドロップすることでウィンドウ上に配置できます。

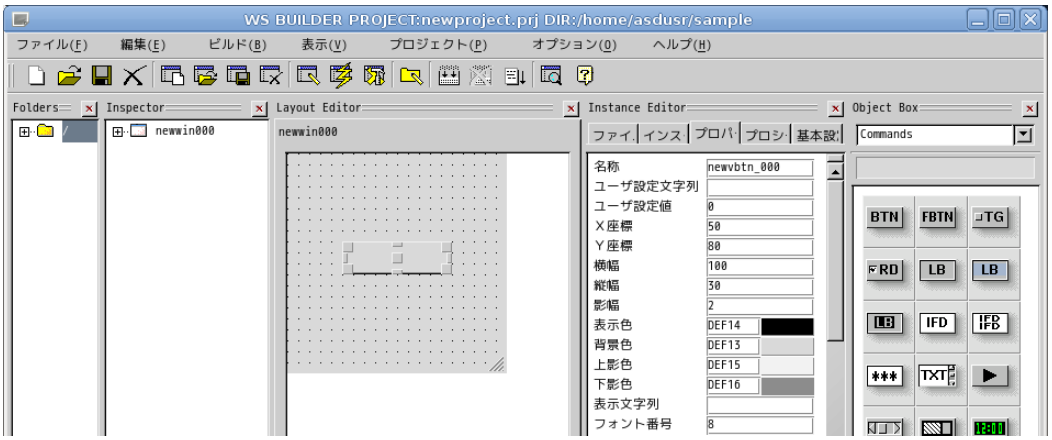


図 3-3-4-2. アプリケーションウィンドウに配置されたボタン

ボタンのプロパティを変更します。アプリケーションウィンドウ上に配置されたボタンをクリックし、「Instance Editor」の「プロパティ」タブをクリックすることで、ボタンのプロパティを設定できるようになります。表 3-3-4-1 を参考に値を変更してください。

表 3-3-4-1. ボタンのプロパティ変更

プロパティ名	説明	設定値
表示文字列	ボタンに表示される文字列の設定	PUSH

※注：ここでは、例としてウィンドウにボタンオブジェクトを貼り付けただけのテストサンプルを作成しています。他の部品の詳細については、ヘルプや WideStudio の書籍を参照してください。

※注：Algonomix4 に組込まれていないライブラリを使用している部品についてはコンパイルできません。組込まれているライブラリと組込まれていないライブラリについては表 3-3-4-2 を参照してください。

表 3-3-4-2. WideStudio コンフィグ時に組込まれるライブラリ

ライブラリ名	状態	内容
OpenGL	No	3D グラフィックスのためのプログラムインターフェイス
JpegLib	HAS	JPEG ファイルを圧縮・伸張するためのライブラリ
PngLib	HAS	PNG ファイルを圧縮・伸張するためのライブラリ
XpmLib	HAS	XPM ファイルを圧縮・伸張するためのライブラリ
ODBC library	No	Open DataBase Connectivity の仕様に基づいたデータベースにアクセスするためのライブラリ
PostgreSQL development library	No	PostgreSQL と呼ばれる、オープンソース系データベースにアクセスするためのライブラリ
MySQL development library	No	MySQL と呼ばれる、オープンソース系データベースにアクセスするためのライブラリ
Python	HAS	Python と呼ばれる、オブジェクト指向スクリプト言語を扱うためのライブラリ
Python header	No	Python を扱うためのヘッダファイル
Ruby	No	Ruby と呼ばれる、オブジェクト指向スクリプト言語を扱うためのライブラリ
Perl	HAS	Perl と呼ばれる、インタプリタ形式のプログラム言語を扱うためのライブラリ

※注：No となっているライブラリは出荷時状態では組込まれていません。

３－３－５　　イベントプロシージャの設定

ボタンのクリックという動作で実行されるプログラムを記述するには、ボタンオブジェクトにプロシージャと呼ばれるプログラムを設定します。

アプリケーションウィンドウ上のボタンをクリックし、「Instance Editor」の「プロシージャ」タブを選択することで、図 3-3-5-1 のような画面が起動します。

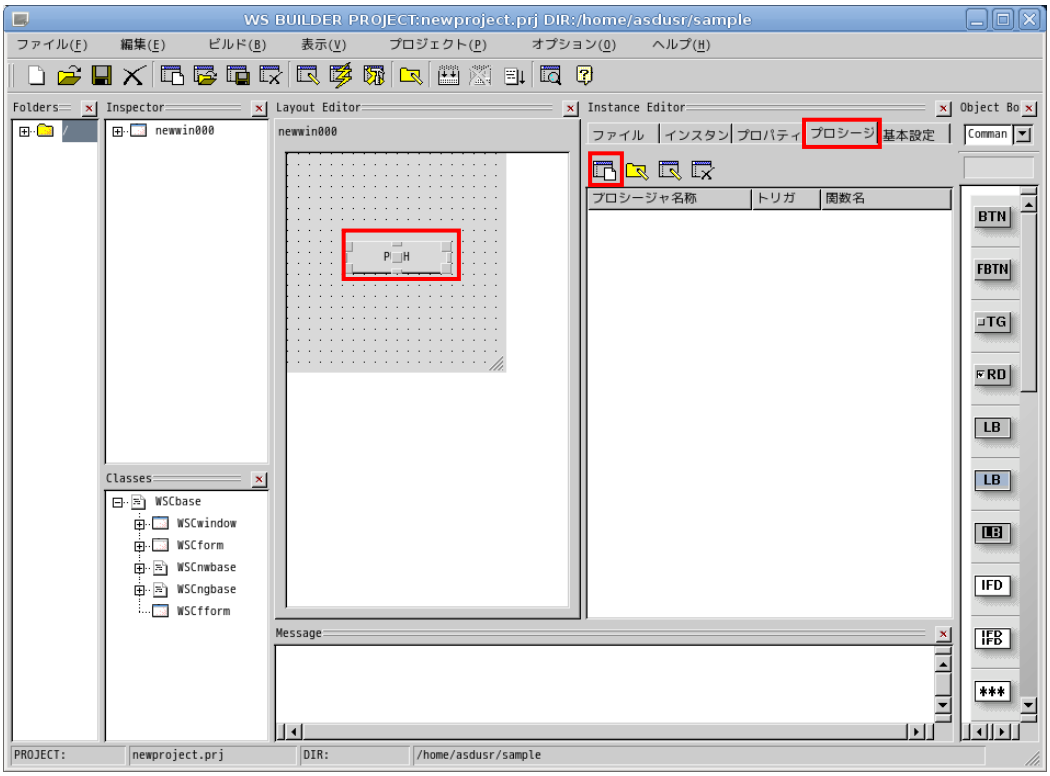


図 3-3-5-1. イベントプロシージャの作成

 をクリックすることで、図 3-3-5-2 のようなイベントプロシージャ作成ダイアログが表示されます。

プロシージャ名は、イベントプロシージャを識別するための名前です。今回は、「Btn_Click」と入力します。起動関数名は、イベント発生時に起動される C/C++の関数名です。この関数に処理を記述します。今回はプロシージャ名と同じ「Btn_Click」と入力します。

起動トリガは、イベントの発生条件を選択します。今回は、ボタンを押して、離されたときに発生するイベントとして「ACTIVATE」を選択します。

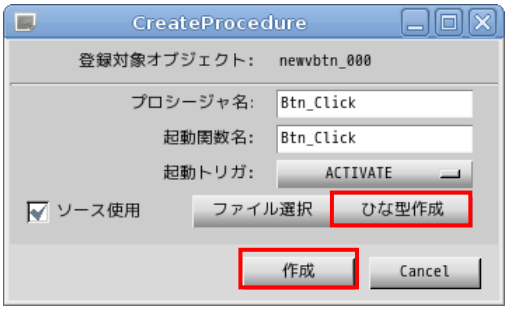


図 3-3-5-2. イベントプロシージャ作成ダイアログ

「ひな型作成」ボタンをクリックすることで、確認ダイアログが表示されるので「OK」ボタンをクリックします。これで、イベントプロシージャのソースコードが自動的に生成されます。「作成」ボタンをクリックします。この操作でイベントプロシージャを作成します。

図 3-3-5-3 のように「Instance Editor」の「プロシージャ」画面に、作成したイベントプロシージャが表示されます。プロシージャ名をダブルクリックすることで、エディタを起動し、処理を記述することができます。

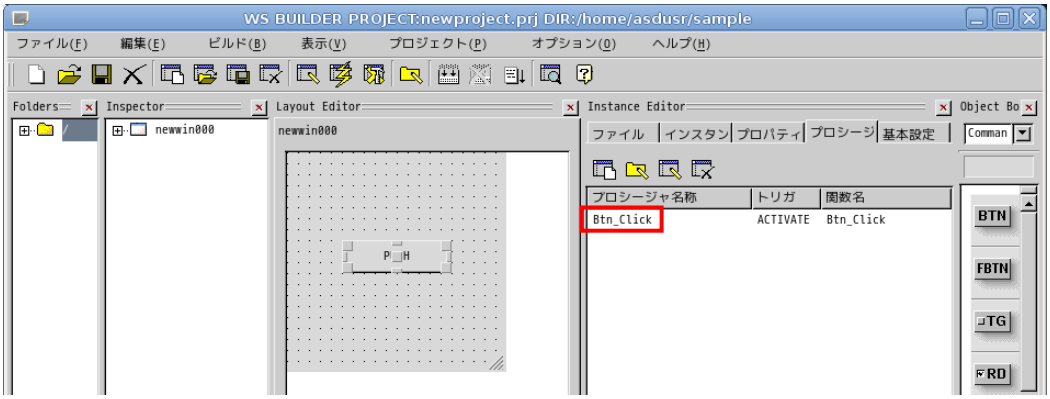


図 3-3-5-3. 作成されたイベントプロシージャ

３－３－６　　イベントプロシージャの編集

ボタンをクリックしたときに、ボタンの表示文字列を変更するコードを記述します。イベントプロシージャの初期状態は図 3-3-6-1 のようになっています。

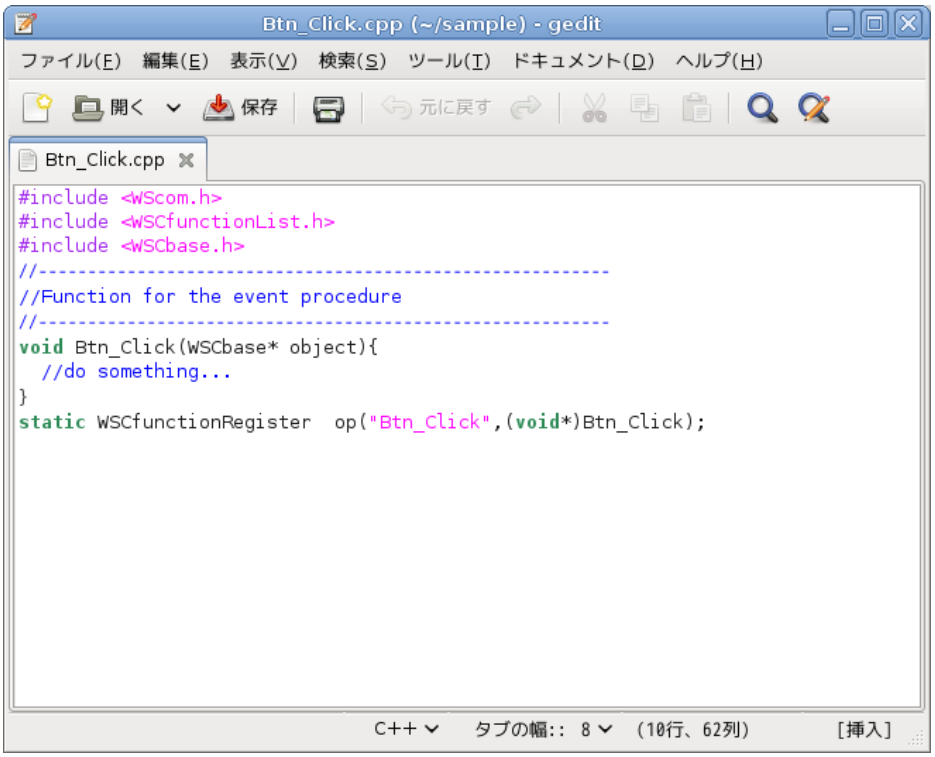


図 3-3-6-1. イベントプロシージャ初期ソースコード

リスト 3-3-6-1 のようにコードを変更します。

リスト 3-3-6-1. 表示文字列を変更するコード

```
#include <WScom.h>
#include <WSCfunctionList.h>
#include <WSCbase.h>
//-------------------------------------------------------------
//Function for the event procedure
//-------------------------------------------------------------
void Btn_Click(WSCbase* object){
    object->setProperty(WSNlabelString,"HELLO!");          //表示文字列変更
}
static WSCfunctionRegister  op("Btn_Click",(void*)Btn_Click);
```

これで、ボタンをクリックしたら、「PUSH」が「HELLO!」となるプログラムができます。保存してエディタを終了します。以上でコーディングは完了です。

３－３－７　　コンパイル

サンプルプログラムのビルドを行います。メニューの「ビルド(B)→リビルド(R)」をクリックしてください。プログラムのコンパイルが始まります。

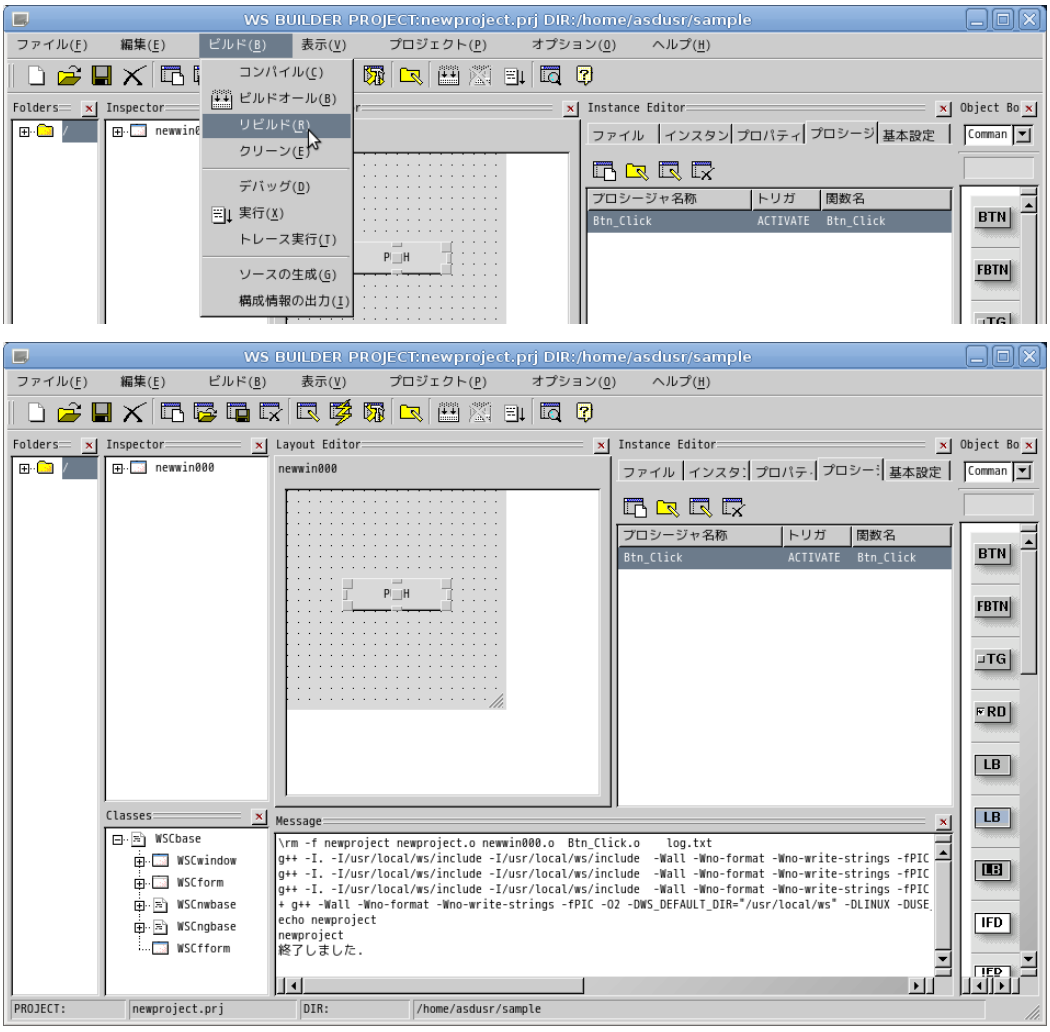


図 3-3-7-1. サンプルプロジェクトのコンパイル

コンパイル完了後、メニューの「ビルド(B)→実行(X)」をクリックしてください。サンプルプログラムが開発環境上で実行されます。「PUSH」ボタンをクリックして、「HELLO!」と表示されることを確認してください。

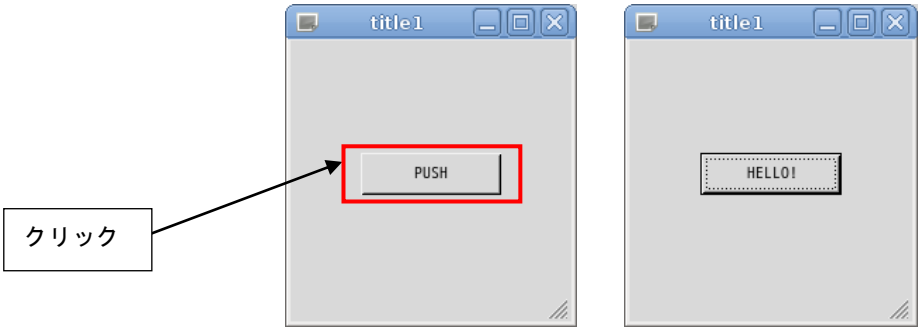


図 3-3-7-2.　プログラム実行画面

動作確認後、プログラムを終了させてください。メニューの「ビルド(B)→実行中止(X)」で終了できます。「終了」または「X」をクリックした場合でも、メニューの「ビルド(B)→実行中止(X)」をクリックしてください。

上記の開発手法は、開発環境 PC 上でコンパイルし、開発環境 PC 上で実行するセルフコンパイル方式です。

EC4A シリーズでは x86 互換の Atom と呼ばれる CPU を採用しているため、開発環境 PC で動作したプログラムはそのまま EC4A シリーズ上でも動作します。この場合、開発環境 PC と実行環境 PC が異なるのでクロスコンパイル方式といいます。

３－３－８　　ファイルの転送

開発環境にて作成した実行プログラムを EC4A シリーズに移して実行します。

実行プログラムだけでなく、設定ファイルや画像データ等 EC4A シリーズにデータを転送するには USB メモリでの転送と、ftp での転送の 2 種類の方法があります。

●USB メモリでの転送

① 開発環境パソコンに USB メモリを挿入します。正常に認識されるとデスクトップ上にハードディスクのアイコンが表示され、内容が表示されます。

② 転送するデータ（サンプルプログラム）を USB メモリにコピーします。

③ ハードディスクアイコンを右クリックし「アンマウント」をクリックします。正常に USB メモリがアンマウントされれば、アイコンが消えますので、USB メモリを抜いてください。

④ 転送するデータが保存された USB メモリを EC4A シリーズに挿入します。

⑤ USB メモリが自動でマウントされます。ホームディレクトリ上で、以下のコマンドを実行します。

***の部分は USB メモリのデバイス名です。

```
# cp /media/asdusr/***/newproject
# ./newproject
```

⑥ ⑤の最後のコマンドで、作成したサンプルプログラムが実行されます。

「PUSH」ボタンを押下したら、「HELLO!」となることを確認してください。

⑦ USB メモリを抜くときは、以下のコマンドを入力します。

```
# sync
# umount  /media/asdusr/***/
```

これで USB メモリがアンマウントされます。

※注：アンマウントせずに USB メモリを抜くとファイルが破損する可能性があります。

●LAN 経由で ftp 転送

① LAN ケーブルを EC4A シリーズに接続します。
EC4A シリーズと開発環境を HUB 無しで接続する場合、クロスケーブルが必要です。

シリーズ側の OS である Algonomix4 のネットワーク設定については、『2-4 有線 LAN の設定について』を参照してください。また、接続する Algonomix4 の IP アドレスを確認しておいてください。

※注：EC4A シリーズと開発環境の IP アドレスは、同一のネットワークアドレスと異なるホストアドレスを指定する必要があります。

② ゲスト OS のデスクトップ上の「アプリケーション」ツールバーから「インターネット」→「gFTP」を選択することで「gFTP」という FTP クライアントが起動されます。

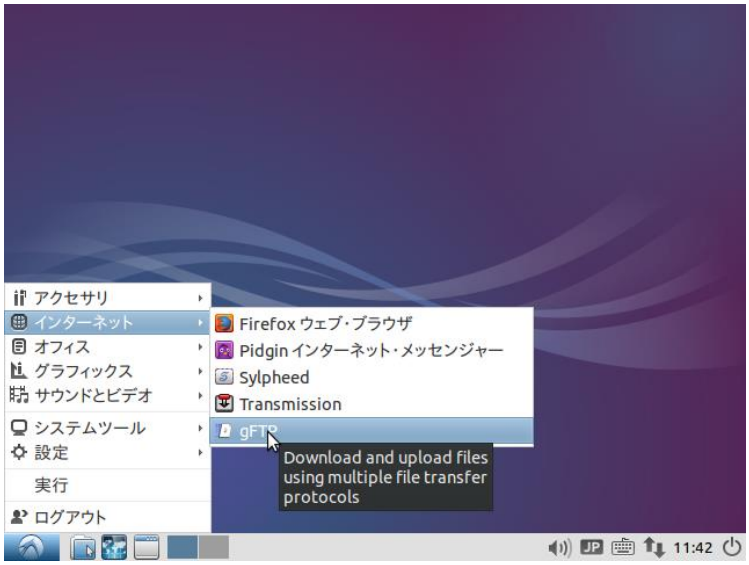


図 3-3-8-1. FTP クライアントの起動

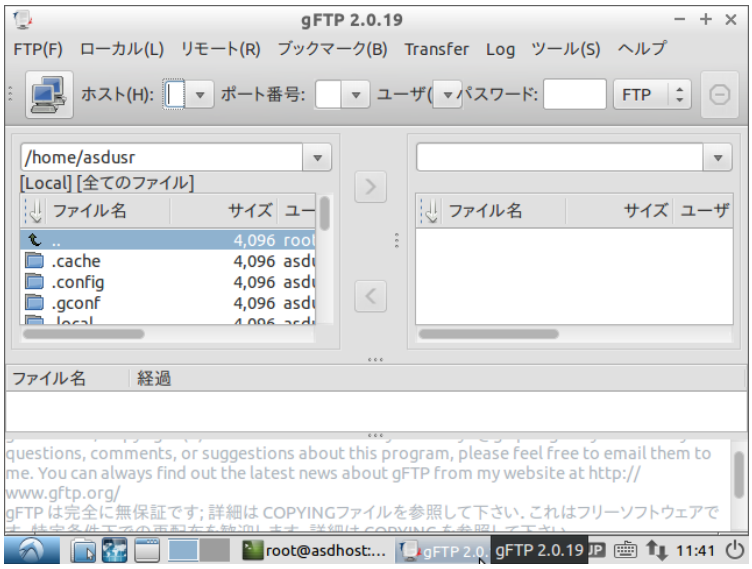


図 3-3-8-2. FTP クライアント画面

③ 接続先である EC4A シリーズの IP アドレスを指定します。

④　　ポート番号は空で大丈夫です。

⑤　　EC4A シリーズへの FTP 接続ユーザ名を指定します。デフォルトでは「asdusr」となっています。

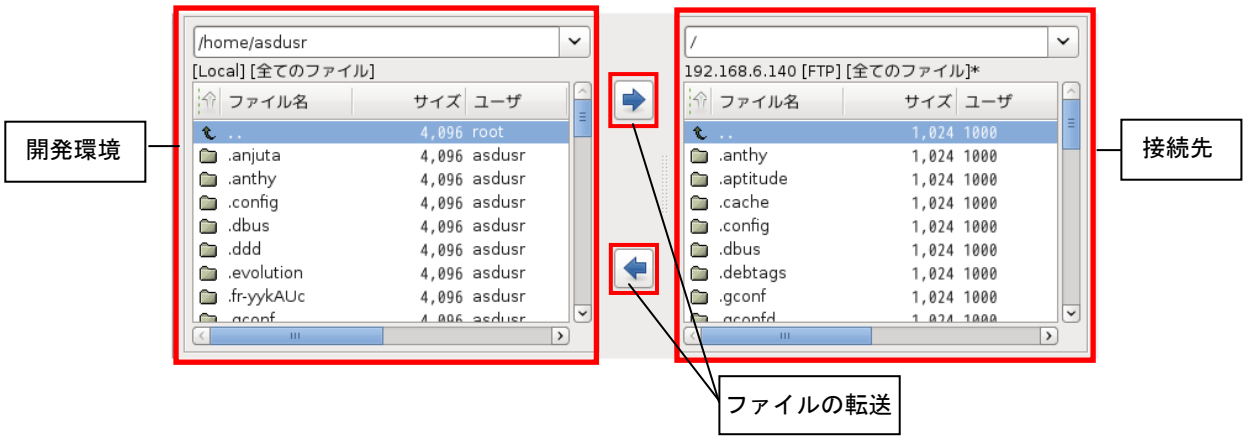
⑥　　EC4A シリーズへの FTP 接続パスワードを指定します。デフォルトでは「asdusr」となっています。

⑦　　通信タイプを指定します。「FTP」としてください。

⑧　　Algonomix4 上で起動している FTP サーバに対して、接続と切断を行います。

⑨　　正常に接続されると、Algonomix4 の「/home/asdusr」ディレクトリがルートディレクトリとしてリスト表示されます。Algonomix4 の FTP サーバ（vsftpd）のデフォルト設定ではセキュリティ上「/home/asdusr」ディレクトリより上のディレクトリには移動できないように設定されています。

移動したいファイルを選択して、矢印キーをクリックすることで、開発環境側と Algonomix4 側でファイルの転送が可能です。



以上でアプリケーション開発の説明は終了です。

３－３－９　　WideStudio/MWT の開発例

本項では、弊社が行った開発事例を列挙します。アプリケーション開発の参考資料としてご使用ください。

●X Window System 上で全画面表示させる方法

X Window System 上でアプリケーションを実行する場合、起動された GUI アプリケーションには必ずタイトルバーがついてきます。これは、X Window System 上で Window Manager が動作しており、Window Manager がタイトルバーをつけ、複数のアプリケーションを管理しているためです。組込み用途で使用する場合、メイン画面はタイトルバーをつけずに全画面表示させる場合があります。表 3-3-9-1 のプロパティを変更することで、タイトルバーのついていないアプリケーションを開発することができます。この場合、起動したプログラムが常に前に表示されるので、アプリケーションウィンドウの表示、非表示を切り替えて使用する必要があります。

表 3-3-9-1. アプリケーションウィンドウのプロパティ変更

プロパティ名	説明	設定値
タイトル属性	ウィンドウのタイトルバー属性の設定	WM 管理外
終了	ウィンドウを非表示にしたとき終了する/しないの設定	オフ

サンプルとして作成した、HELLO プログラムを修正して試してみます。アプリケーションウィンドウのプロパティを表 3-3-9-1 に書かれているように変更してください。Btn_Click のイベントプロシージャをリスト 3-3-9-1 のように記述してください。

このプログラムをコンパイルして実行すると、タイトルバーが付いていない画面が起動されます。ボタンを押下することで、メイン画面が非表示になり、xeyes というプログラムが起動されます。xeyes が終了されると、再度メイン画面が表示されます。

リスト 3-3-9-1. 別アプリケーションの起動サンプル

```
#include "stdlib.h"
#include <WScom.h>
#include <WSCfunctionList.h>
#include <WSCbase.h>
#include "newwin000.h"
//--------------------------------------------------------
//Function for the event procedure
//--------------------------------------------------------
void Btn_Click(WSCbase* object){
    newwin000->setVisible(False);                    //アプリケーションウィンドウの非表示
    system("xeyes");                                 //xeyes というプログラムを起動する
    newwin000->setVisible(True);                     //アプリケーションウィンドウの表示
}
static WSCfunctionRegister  op("Btn_Click",(void*)Btn_Click);
```

●WideStudio のフォント設定について
WideStudio には次の 4 系統のフォント設定があります。
■X11 系の設定
FONT0:サイズ フォント名 Weight(0：無し 1：bold) slant(0：無し 1：イタリック)
[例] FONT0:10 * 0 0
FONT1:12 * 0 0
Algonomix4 の/etc/xunicoderc にて詳細なフォントを定義
■T-Engine 系の設定
FONT0:サイズ フォント ID
[例] FONT0:10 60c6
FONT1:12 60c6
■DirectFB 系 Linux フレームバッファ系、T-Engine フレームバッファ系
FONT0:font0
[例] FONT0:font0
FONT1:font1
/etc/wsfonts にてフォントファイルを定義
■Windows 系の設定
Windows フォントパラメータをカンマ区切りで列挙

WideStudio のフォント設定は、プロジェクト設定のフォント設定タブで設定します。また、「prj ファイル」に記録されているフォント設定の値を直接変更することでフォント設定を行うこともできます。

※注：Algonomix4 上のフォントと開発環境上のフォントは設定が変更されている可能性があるため、開発時に見えているフォントと実際に動作させたときのフォントが違う可能性があります。

●X Window System の場合
Algonomix4 上で WideStudio アプリケーションを動作させる際のフォント設定方法を以下に示します。

リスト 3-3-9-2. X11 用のフォント設定例（prj ファイル）

```
#FONT0
12 東風ゴシック 0 0    /* サイズ 12 フォント名 東風ゴシック Weight 無し   slant 無し */
#FONT1
14 東風ゴシック 0 0    /* サイズ 14 フォント名 東風ゴシック Weight 無し   slant 無し */
#FONT2
16 東風ゴシック 0 0    /* サイズ 16 フォント名 東風ゴシック Weight 無し   slant 無し */
#FONT3
18 東風ゴシック 0 0    /* サイズ 18 フォント名 東風ゴシック Weight 無し   slant 無し */
#FONT4
24 東風ゴシック 0 0    /* サイズ 24 フォント名 東風ゴシック Weight 無し   slant 無し */
#FONT5
30 東風ゴシック 0 0    /* サイズ 30 フォント名 東風ゴシック Weight 無し   slant 無し */
#FONT6
34 東風ゴシック 0 0    /* サイズ 34 フォント名 東風ゴシック Weight 無し   slant 無し */
#FONT7
36 東風ゴシック 0 0    /* サイズ 36 フォント名 東風ゴシック Weight 無し   slant 無し */
```

```
#FONT0
12  *  0  0
サイズ 12 フォント名 * Weight  0:無し 1:太字  slant 0:無し 1:斜体
```

WideStudio Ver3.98-4 から XFT ライブラリを利用し、アンチエイリアスの効いたフォント表示を行うことができます。

Algonomix4 に標準実装されているフォントは「東風ゴシック」のみです。

WideStudio Ver3.98-6 時点ではボールド設定（Weight）とイタリック設定（slant）は使用することが出来ません。

Algonomix4 開発環境では、文字マップというプログラムを使用することで TrueType フォントを確認することができます。起動方法と画面を図 3-3-9-1 に示します。

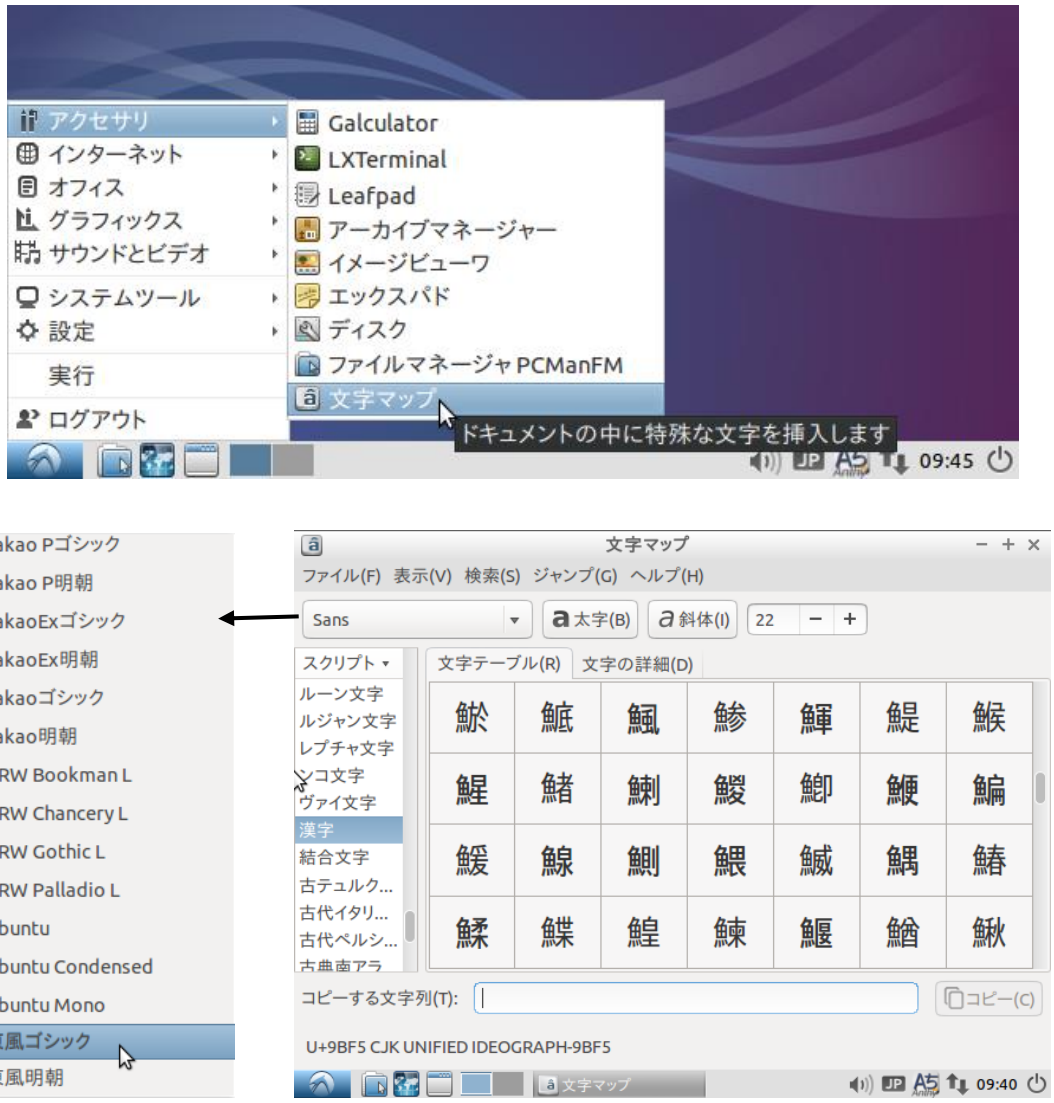


図 3-3-9-1. 文字マップツール

３－４　　GDB によるデバッグ方法の詳細

Algonomix4 上で実行されるプログラムのデバッグには GDB や GDB サーバを使用します。Algonomix4 上では GDB サーバを実行させ、開発環境側では GDB を実行します。開発環境側で GDB 用のコマンドを実行することによりブレークやステップ実行などを行うことができます。

本項では簡単な使い方について説明します。詳細な使用方法については、GDB のマニュアルや解説書などを参照ください。

Algonomix4 では、ネットワークポートを使用したデバッグ方法を標準としています。
以下より『3-3 WideStudio/MWT によるアプリケーション開発』でサンプルプログラムとして作成したプログラムを使用して、GDB にてデバッグする方法を示します。

① デバッグモードでのコンパイル
デバッグ過程がよくわかるように、Hello!と表示するボタンのプロシージャ関数のコードをリスト 3-4-1 のように変更します。

リスト 3-4-1. 表示文字列を変更するコード（デバッグ確認用）

```
#include <WScom.h>
#include <WSCfunctionList.h>
#include <WSCbase.h>
//-------------------------------------------------------------
//Function for the event procedure
//-------------------------------------------------------------
void Btn_Click(WSCbase* object){
    int i,j,k;

    k=0;
    for(i=0;i<10;i++){
        for(j=0;j<10;j++){
            k++;
        }
    }
    object->setProperty(WSNlabelString, k);          //表示文字列変更（k の値を表示）
}
static WSCfunctionRegister  op("Btn_Click",(void*)Btn_Click);
```

GDB を使用する場合、コンパイルオプションとして「-g」や「-ggdb」を付加してコンパイルする必要があります。

WideStudio の場合は、  をクリックする、またはメニューの「プロジェクト(P)」→「プロジェクト設定(E)」をクリックします。「コンパイル」タブをクリックすることで、図 3-4-1 のようなプロジェクト設定画面が表示されます。

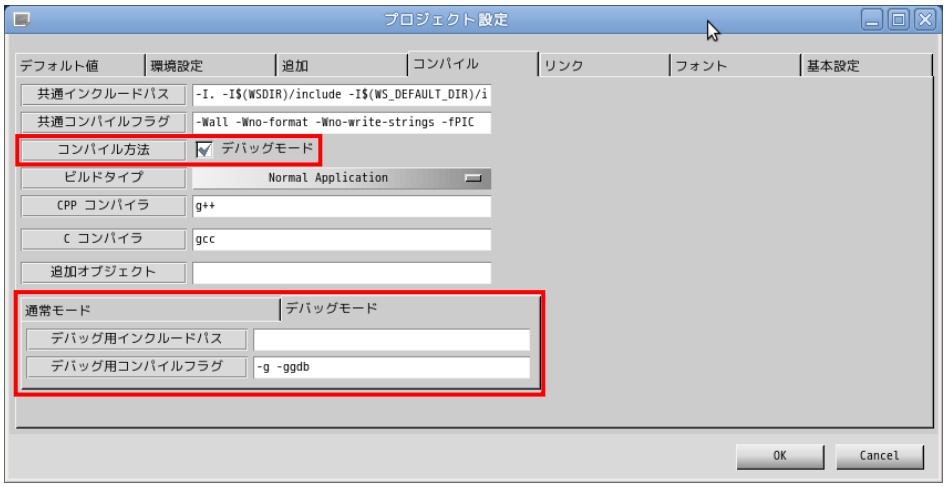


図 3-4-1. プロジェクト設定画面（コンパイルタブ）

「コンパイル方法」項目の、「デバッグモード」のチェックボックスにチェックを入れた場合は、デバッグモードのコンパイルフラグが使用されるようになります。デバッグモードのタブに「-g -ggdb」デバッグ用コンパイルフラグが設定されていますので確認してください。

「OK」ボタンをクリックし、プロジェクト全体をコンパイルします。通常モードで作成される実行プログラム名に「d」が付いた実行プログラムが作成されます。(「newproject」→「newprojectd」)

② GDB サーバの起動

Algonomix4 で「gdbserver」を起動します。『3-3-8 ファイルの転送』の「LAN 経由で ftp 転送」の項目を参考に、①でコンパイルした「newprojectd」を EC4A シリーズに転送します。以下のコマンドを実行して「gdbserver」を起動します。

```
# chmod 755 newprojectd
# gdbserver host1:2345 ./newprojectd
Process ./newprojectd created; pid = 21507
Listening on port 2345
```

③ GDB の起動

開発環境のデスクトップ上の「アプリケーション」ツールバーから「アクセサリ」→「端末」をクリックし、別のコンソールを起動します。デバッグするプログラムのソースディレクトリへ移動します。

```
# cd /home/asdusr/sample
```

GDB を起動し、Algonomix4 の GDB サーバと接続します。

```
# gdb ./newprojectd
GNU gdb (GDB) 7.0.1-debian
Copyright (C) 2009 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the externt permitted by law. Type “show copying”
and “show warrantry” for details.
This GDB was configured as “i484-linux-gnu”.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /home/asdusr/sample/newprojectd...done.
(gdb) target remote 192.168.0.1:2345
Remote debugging using 192.168.0.1:2345
[New Thread 3009]
0x295568c0 in ?? () from /lib/ld-linux.so.2
(gdb)
```

ここまでで、GDB の起動は完了しました。

④　　デバッグ

ブレークポイントを指定します。ここではボタンのプロシージャ関数でブレークするようにします。

```
(gdb) b Btn_Click(WSCbase*)
Breakpoint 1 at 0x401904: file Btn_Click.cpp, line 10.
(gdb)
```

コンティニュー実行します。Algonomix4 上にサンプルプログラムの画面が表示されます。

```
(gdb) c
Continuing.
```

「PUSH」ボタンをクリックすることで、以下のメッセージがでてブレークされます。

```
Breakpoint 1, Btn_Click (object=0x452d08) at Btn_Click.cpp:10
10              k=0;
Current language:  auto
The current source language is “auto; currently c++”.
(gdb)
```

監視する変数を指定します。

```
(gdb) display k
1: k = 152474568
(gdb) display i
2: i = -1076372104
(gdb) display j
3: j = -1207467296
(gdb)
```

ステップ実行します。

```
(gdb) n
11              for(i=0;i<10;i++){
3: j = -1207467296
2: i = -1076372104
1: k = 0
(gdb) n
12                      for(j=0;j<10;j++){
3: j = -1207467296
2: i = 0
```

```
1: k = 0
(gdb) n
13                                  k++;
3: j = 0
2: i = 0
1: k = 0
(gdb) n
12                          for(j=0;j<10;j++){
3: j = 0
2: i = 0
1: k = 1
(gdb)
```

リストを表示します。入力された数値を中心に 10 行表示されます。

```
(gdb) list 13
8               int i,j,k;
9
10              k=0;
11              for(i=0;i<10;i++){
12                      for(j=0;j<10;j++){
13                              k++;
14                      }
15              }
16              object->setProperty(WSNlabelString, k);
17      }
(gdb)
```

16 行目にブレークを張り、コンティニュー実行します。

```
(gdb) b 16
Breakpoint 2 at 0x40193a: file Btn_Click.cpp, line 16.
(gdb) c
Continuing.

Breakpoint 2, Btn_Click (object=0x452d08) at Btn_Click.cpp:16
16              object->setProperty(WSNlabelString, k);
3: j = 10
2: i = 10
1: k = 100
(gdb)
```

この時点で、ボタンの表示が変更される直前まで実行しました。さらにコンティニュー実行を行うとボタンに 100 と表示されます。サンプルプログラムを終了し、GDB を終了します。

```
(gdb) c
Continuing.

Program exited normally.
(gdb) q
#
```

以上で GDB によるデバッグ方法について簡単に説明しました。ここで紹介したコマンドの他にもいろいろなコマンドが用意されています。比較的よく使うと思われるコマンドについて表 3-4-1 に示します。

表 3-4-1. GDB コマンド（抜粋）

コマンド（省略）		書式	説明
実行	continue (c)	c	停止中のプログラムを再開します。
	next (n)	n	実行する行が関数の場合、関数の中へ入らずに次の行まで実行します。
	step (s)	s	実行する行が関数の場合、関数の中に入って実行します。 **※注：実行する関数が WideStudio の関数の場合はライブラリがデバッグ対応でないためステップ実行することができません。**
中断	break (b)	b <関数名> b <行番号> b <ファイル名>:<行番号>	ブレークポイントを設定します。
変数	display (disp)	disp <変数名>	監視する変数を設定します。 プログラムが停止するたびに値が表示されます。
	print (p)	p <変数名>	変数の値をモニタします。
	set	set <変数名>=<値>	変数の値を変更します。
その他	list (l)	l <行番号> l <関数名>	指定した箇所のソースを 10 行表示します。
	delete (d)	d <ブレークポイント> d <監視中の変数>	指定した番号のブレークポイントや監視中の変数を削除できます。
	info (i)	i breakpoints i local i func	デバッグ中の情報を表示します。他のサブコマンドについては「help info」を参照してください。 breakpoints：現在、張っているブレークポイントの表示 local　　：ローカル変数の表示 func　　：関数の表示
	quit (q)	q	デバッガを終了します。

３－５　　VirtualBox を使用しない開発環境構築方法

本項では、パソコンに直接 Lubuntu 14.04.1 をインストールし、Algonomix4 の開発環境を構築する方法について説明します。

３－５－１　　Lubuntu 14.04.1 のインストール

Lubuntu 14.04.1 をインストールするパソコンの推奨スペックを表 3-5-1-1 に示します。

表 3-5-1-1. Lubuntu 14.04.1 インストール必須パソコンスペック

CPU	Pentium4 1GHz 以上
メモリ	256MByte 以上　(512MByte 以上推奨)
HDD 空き領域	8GByte 以上

Algonomix4 では Lubuntu 14.04.1 を使用しています。バージョンが違うと不整合が起こる可能性がありますので、弊社で Algonomix4 を構築するときに使用した、Lubuntu 14.04.1 のインストールディスクを配布します。ネットワークアップデート等は利用しないようにしてください。

下記に示す Lubuntu 14.04.1 のインストール方法はパソコンによって若干変化する可能性があります。基本的には画面の指示通りにインストールしてください。

①　「Lubuntu 14.04.1」の DVD-ROM をパソコンに入れ、DVD-ROM から起動します。
DVD-ROM から起動するには、インストールするパソコンの BIOS 設定を行う必要があります。

②　正常に DVD-ROM から起動されると、図 3-5-1-1 のようなインストーラの言語選択画面が表示されます。使用する言語を選択して、「Enter」キーを押します。ここでは[日本語]を選択しています。

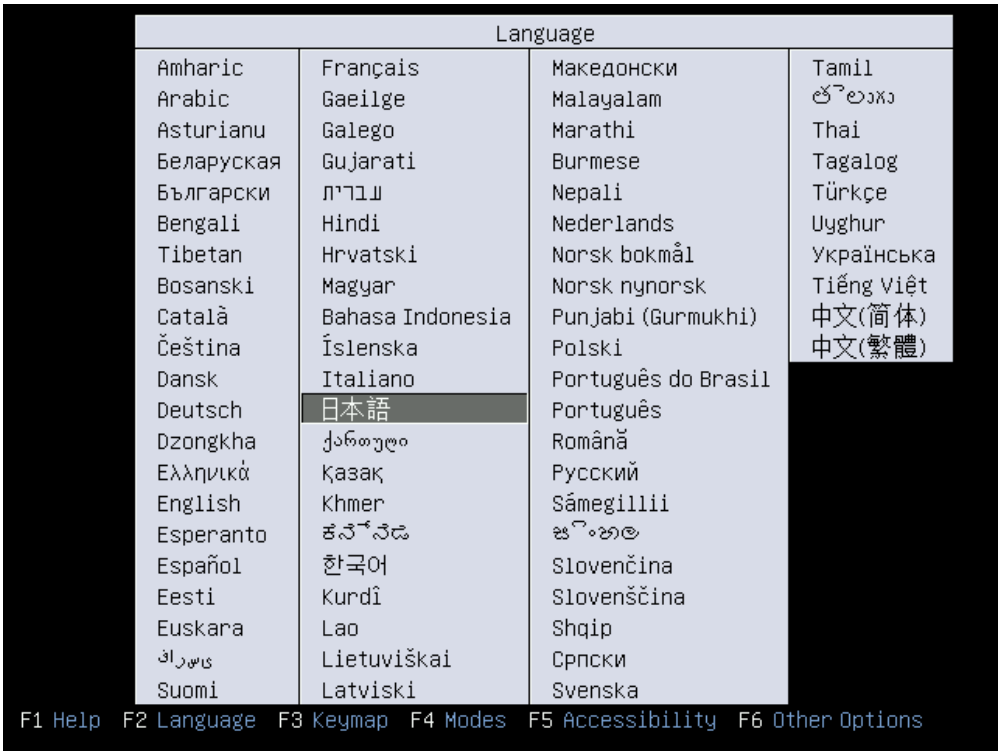


図 3-5-1-1. インストーラの言語選択画面

③　　次に、図 3-5-1-2 のような画面が表示されますので、「Lubuntu をインストール」を選択して「Enter」キーを押します。

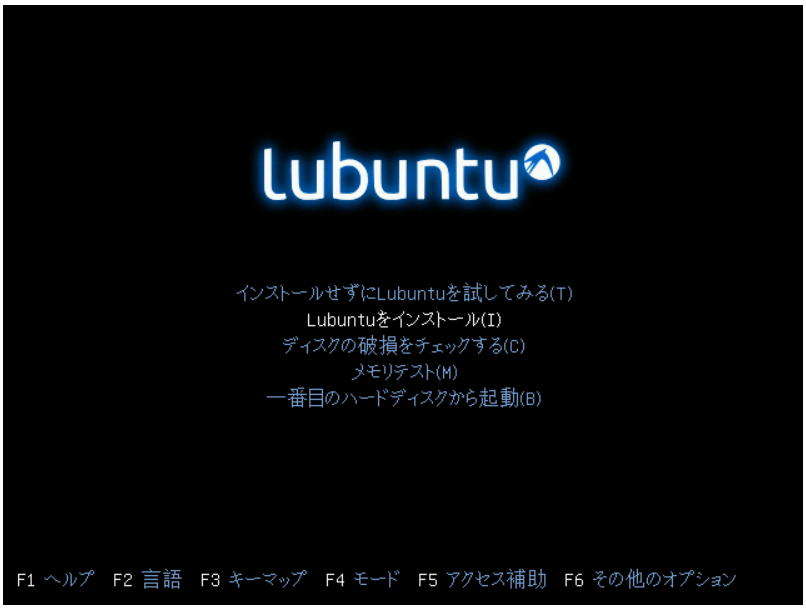


図 3-5-1-2. 起動メニュー画面

④　　図 3-5-1-3 のような言語選択画面が表示されます。ここでは実際にインストールする Lubuntu の言語を選択します。カーソルキーで言語を選んで「Enter」キーを押します。

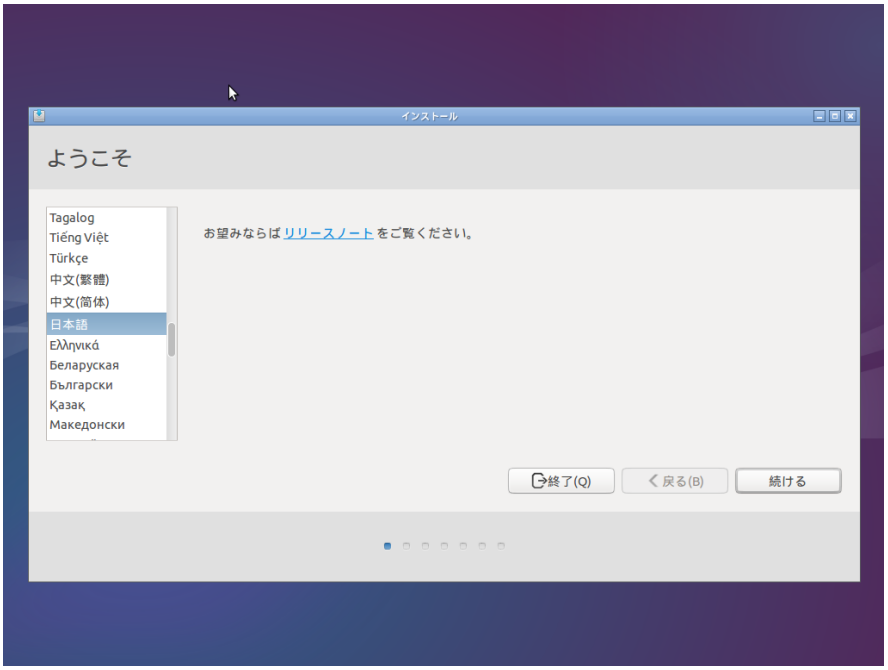


図 3-5-1-3. インストールする Lubuntu の言語選択画面

⑤　図 3-5-1-4 のような画面が表示されます。チェックボックスはどちらもチェックしないで[続ける]をクリックしてください。

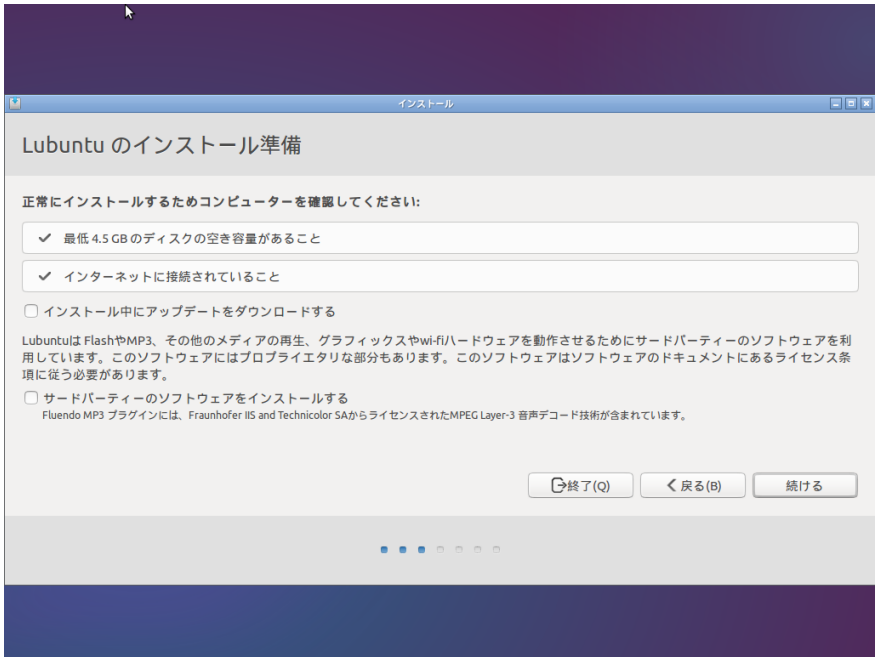


図 3-5-1-4. インストール準備画面

⑥　図 3-5-1-5 のようなディスクのパーティションを設定する画面が表示されます。パーティション分割が必要ない場合、そのまま[インストール]をクリックしてください。

図 3-5-1-5. インストール方法選択画面

⑦　　図 3-5-1-6 のような地域設定画面が表示されます。[続ける]をクリックしてください。

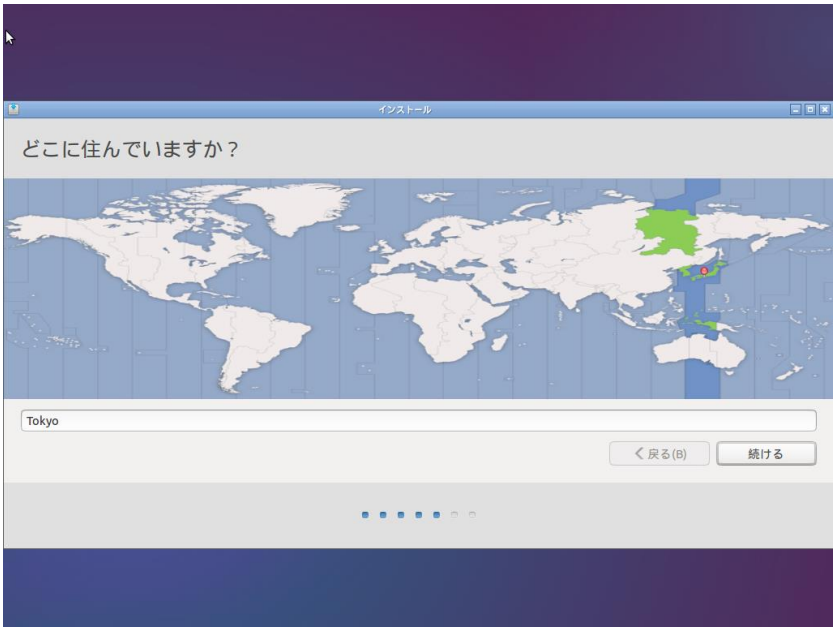


図 3-5-1-6. 地域設定画面

⑧　　図 3-5-1-7 のようなキーボードレイアウト設定画面が表示されます。ご使用のキーボードをにあったものを選択して「続ける」をクリックします。

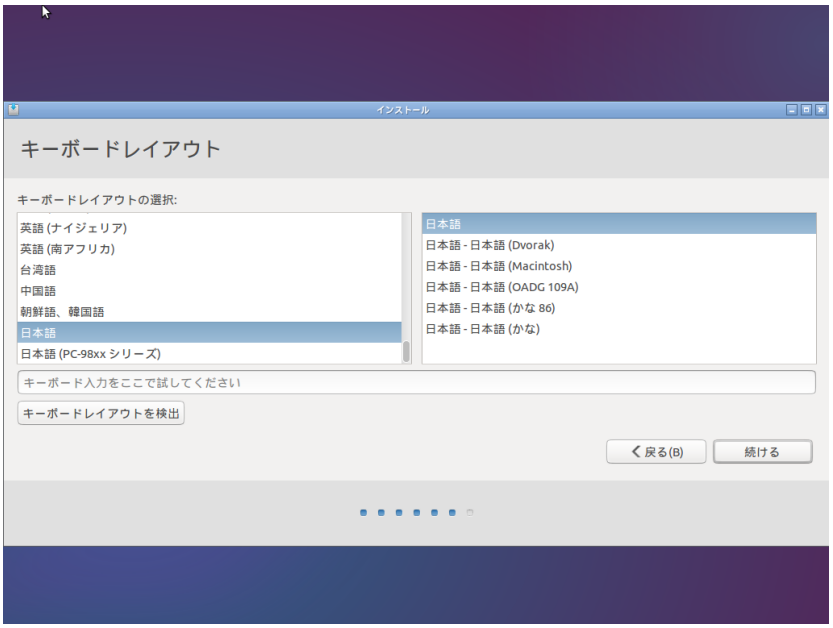


図 3-5-1-7. キーボードレイアウト設定画面

⑨　　図 3-5-1-8 のようなユーザ設定画面が表示されます。
必要な情報を入力後、[続ける]をクリックしてください。

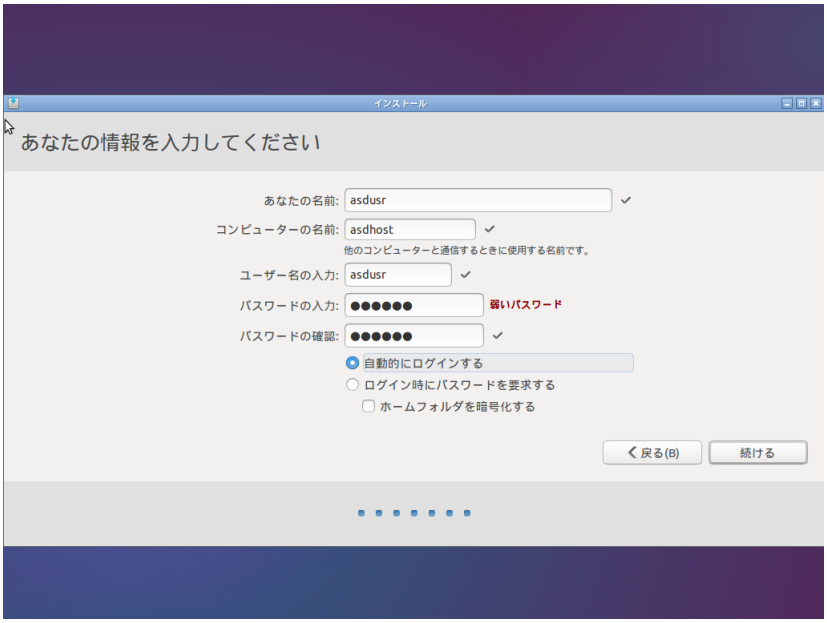


図 3-5-1-8. ユーザ設定画面

⑩　　正常にインストールが完了することで、図 3-5-1-9 のような画面が表示されます。
DVD-ROM を取り出して「続ける」をクリックしてください。パソコンが再起動します。

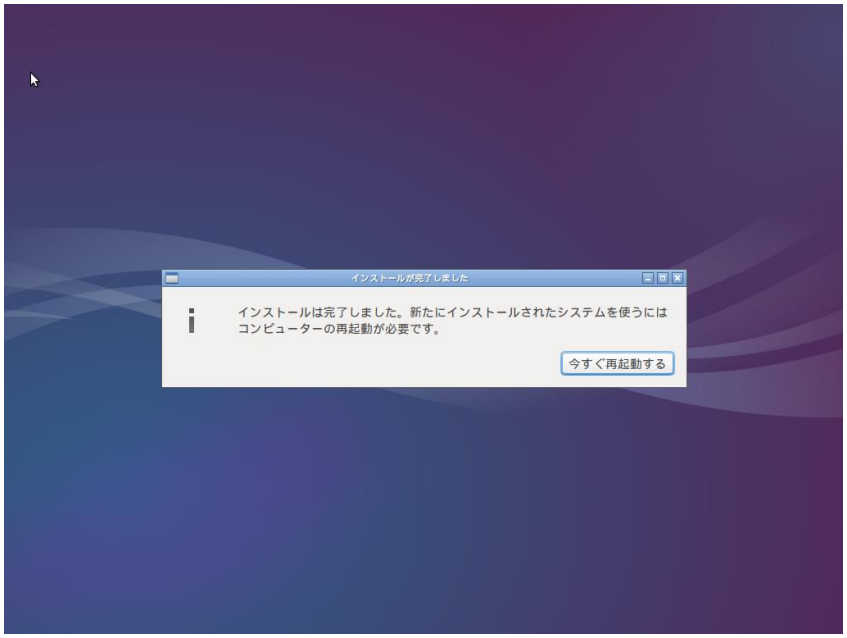


図 3-5-1-9. インストール完了画面

⑪　　再起動されると、ユーザ名入力画面が表示されますので、⑨で設定したユーザ名とパスワードを入力してください。(⑨で自動的にログインするを選択していても、初回起動時はパスワードの入力が必要です)

⑫　　起動時、図 3-5-1-10 のようなソフトウェアの更新確認メニューが表示された場合、[後で通知する]をクリックしてください。

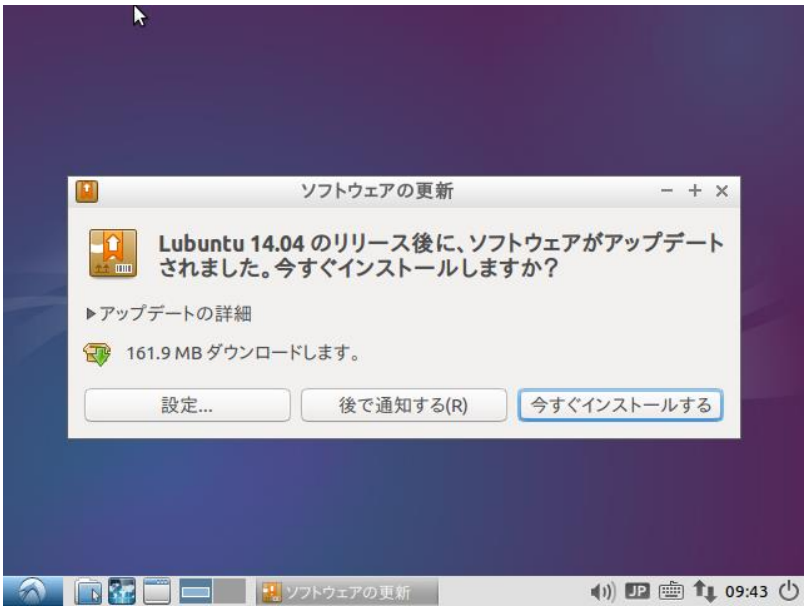


図 3-5-1-10. アップデートタブ

３－５－２　　開発環境用追加パッケージインストール

Algonomix4 用の開発環境を構築する為に必要なパッケージをインストールします。

インストールするパッケージ名称とバージョンの一覧を表 3-5-2-1 に示します。

表 3-5-2-1. 追加インストールパッケージ一覧

パッケージ名称	内容	バージョン
ssh	セキュアなシェルクライアントとサーバ	1:6.6p1-2ubuntu2
samba	Windows 共有用サーバ	2:4.1.6+dfsg-1ubuntu2.1.4.04.3
kernel-package	Linux カーネルを構築する為のユーティリティ	12.036+nmu3
libncurses5-dev	開発用 NCURSES ライブラリ	5.9+20140118-1ubuntu1
gdb	GNU デバッガ	7.7-0ubuntu3.1
gftp	FTP クライアント	2.0.19-4ubuntu2
libX11-dev	開発用 X11 ライブラリ	2:1.6.2-1ubuntu2
libjpeg62-dev	開発用 JPEG ライブラリ	6b1-4ubuntu1
libpng12-dev	開発用 PNG ライブラリ	1.2.50-1ubuntu2
libxpm-dev	開発用 X11 pixmap ライブラリ	1:3.5.10-1
libxt-dev	開発用 X11 toolkit ライブラリ	1:1.1.4-1
libxft-dev	開発用 X FreeType フォント描画ライブラリ	2.3.1-2
libxext-dev	開発用 X11 拡張ライブラリ	2:1.3.2-1
libperl-dev	開発用 Perl ライブラリ	5.18.2-2ubuntu1
ttf-kochi-gotic	True Type ゴシックフォント	20030809-15
ttf-kochi-mincho	True Type 明朝フォント	20030809-15

① 「システム」→「システム管理」→「Synaptic パッケージマネージャ」を選択し、パッケージマネージャを起動します。

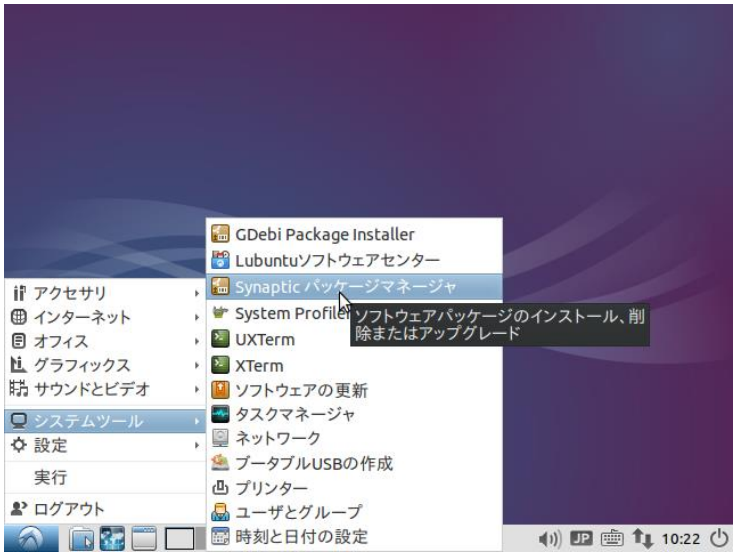


図 3-5-2-1. パッケージマネージャの起動

②　最初の起動では、管理者権限パスワードの問い合わせがありますので、Lubuntu 14.04.1 インストール時に設定したパスワードを入力します。

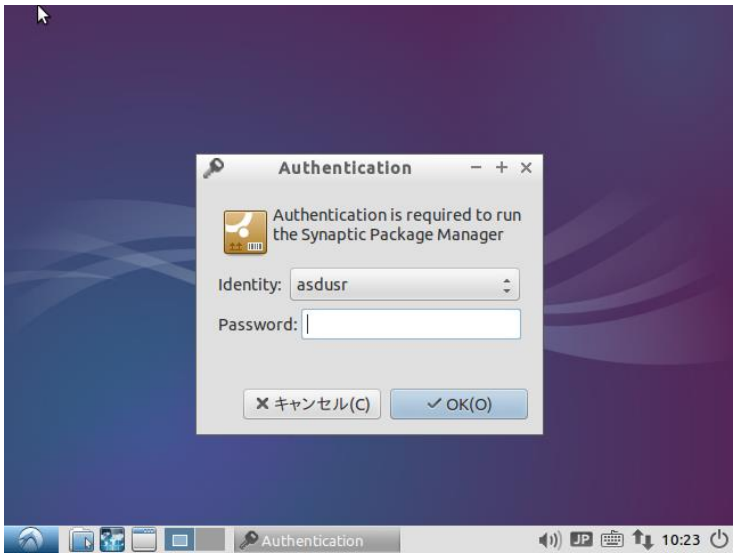


図 3-5-2-2.　管理者権限パスワード入力画面

③　正常に起動されると図 3-5-2-3 のような画面が起動されます。

図 3-5-2-3.　Synaptic パッケージマネージャ起動画面

④　表 3-5-2-1 に示したパッケージを順次インストールしてください。「検索」をクリックし、インストールするパッケージ名を検索します。

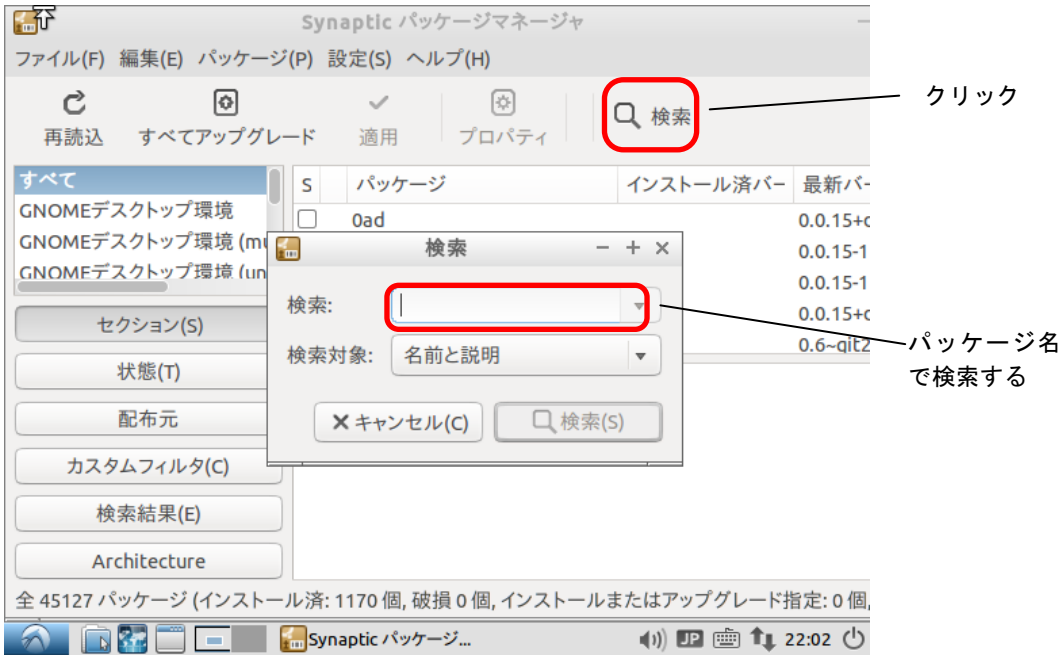


図 3-5-2-4.　パッケージの検索

⑤　インストールするパッケージ名のチェックボックスをクリックすることで、図 3-5-2-5 のようなメニューが表示されるので「インストール指定」をクリックします。このとき、他にインストールする必要のあるパッケージがある場合は、図 3-5-2-6 のような画面が開きますので「マーク(M)」をクリックします。

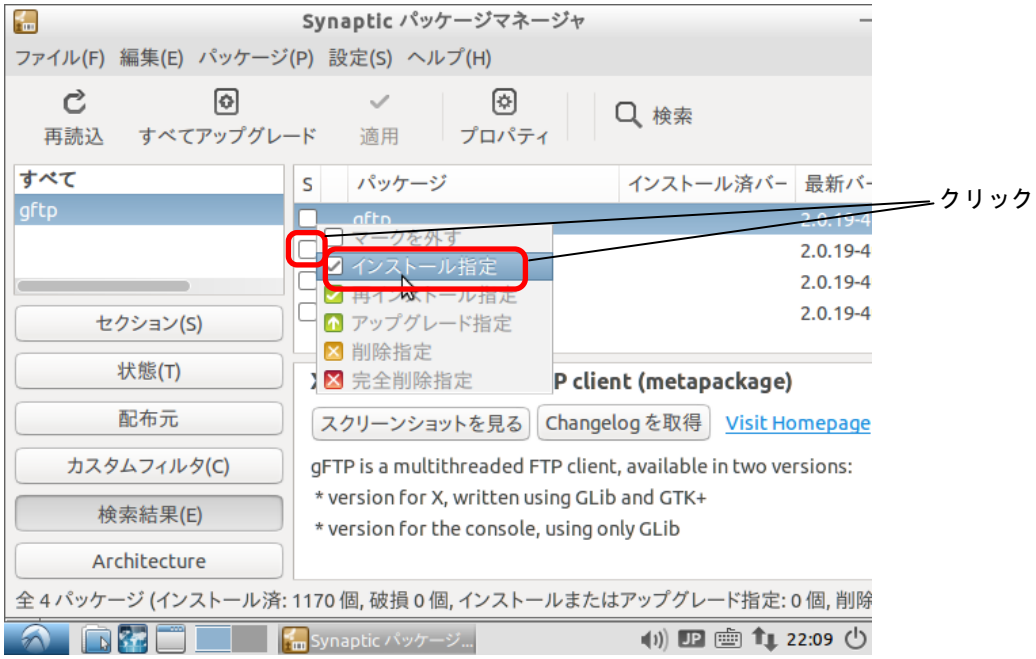


図 3-5-2-5.　パッケージ追加インストール確認画面

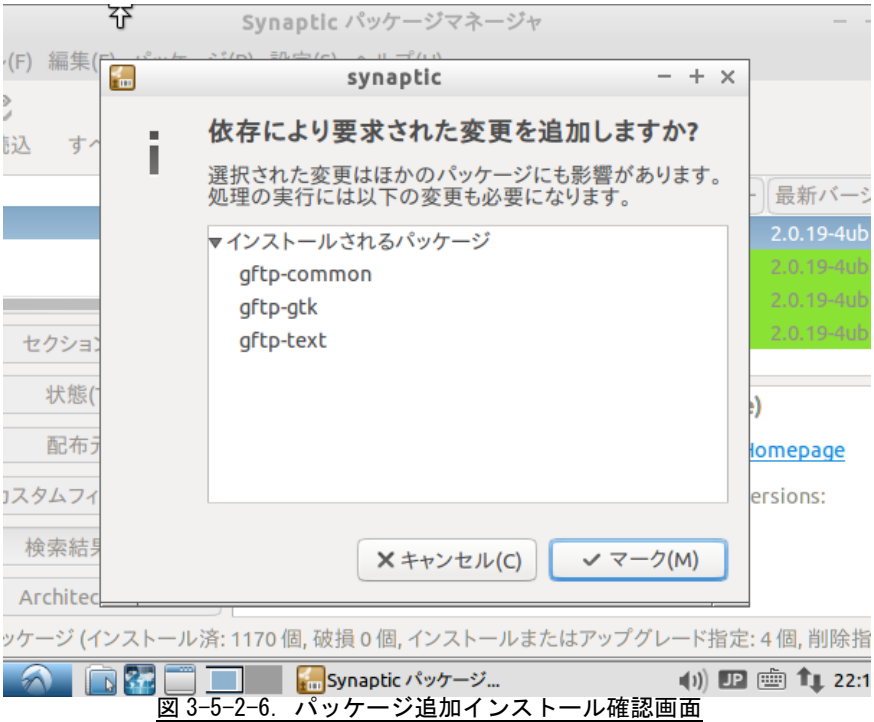


図 3-5-2-6. パッケージ追加インストール確認画面

⑥　「適用」をクリックすることで、インストールが始まります。

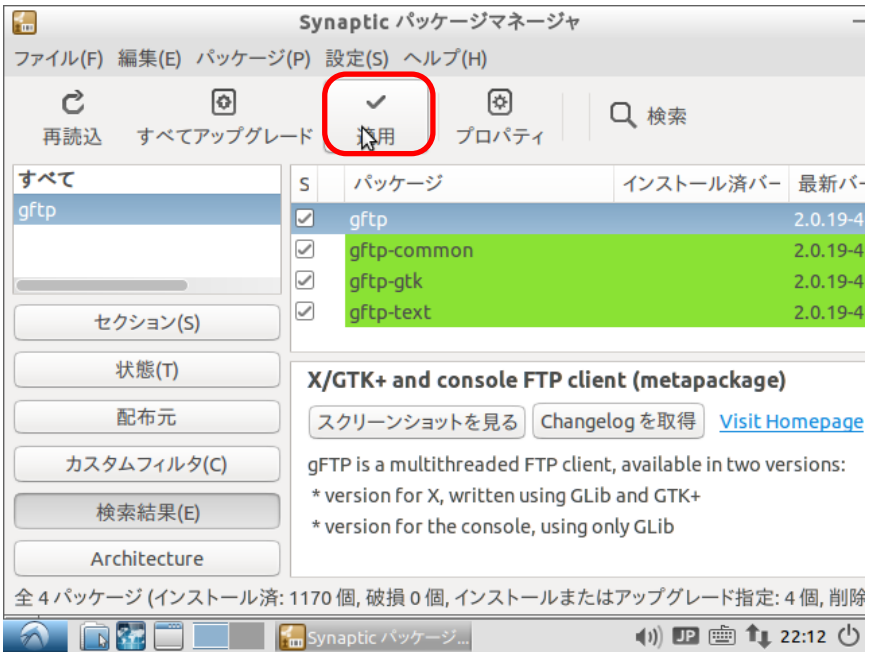


図 3-5-2-7. パッケージ追加画面

⑦　④～⑥を繰り返し、表 3-5-2-1 のすべてのパッケージをインストールしたら再起動してください。

３－５－３　　Algonomix4 用開発環境インストール

Lubuntu 14.04.1 に Algonomix4 用の開発環境をインストールします。Algonomix4 用開発環境 DVD-ROM に格納されている、「Algonomix4-tools-0010.tar.gz」を展開することでインストールされます。

インストールされるパッケージの内容については『3-2-7　Algonomix4 用開発環境のディレクトリ構成について』を参照してください。

① Algonomix4 用開発環境 DVD-ROM をパソコンにセットします。しばらくすると、デスクトップ上に CD-ROM ドライブのアイコンが表示され、内容が表示されます。

② コンソールを起動し、下記のコマンドを実行します。

```
$ sudo su
パスワード：asdusr　　←インストール時に設定したパスワード
# tar zxvf /media/asdusr/デバイス名/development/Algonomix4-tools-0010.tar.gz –C /
```

これで、Algonomix4 用の開発環境一式が展開されます。パソコンスペックにもよりますが、展開終了までに数十分程度かかります。

③ 正常に展開が終了されると「/usr/local/tools-0010」というディレクトリが作成されます。

④ WideStudio をインストールします。下記のようにコマンドを実行してください。

```
# cd /usr/local/tools-0010/src/widestudio/ws-v3.98-6/src
# make install
# ldconfig
```

⑤ WideStudio 実行用スクリプトをコピーします。

```
# cp /usr/local/tools-0010/src/widestudio/wsstart.sh /usr/local/bin
```

⑥ デスクトップ画面に WideStudio 実行用スクリプトのショートカットを登録します。

```
# ln –s /usr/local/bin/wsstart.sh /home/asdusr/デスクトップ/WideStudio
```

図 3-5-3-1.　WideStudio 起動ショートカット

⑦ ダブルクリックして WideStudio が起動されることを確認してください。最後に WideStudio の環境設定を行います。「オプション(O)」→「環境設定(E)」をクリックしてください。

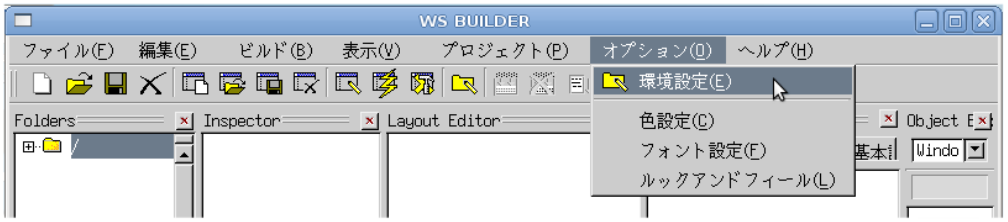


図 3-5-3-2.　WideStudio の環境設定

⑧　　変更するべき箇所は表 3-5-3-1 に書かれた 4 カ所です。

表 3-5-3-1. WideStudio 環境設定項目

タブ	設定項目	設定値
環境設定	ターミナル	LXTerminal
	エディタ	leafpad
	Web ブラウザ	firefox

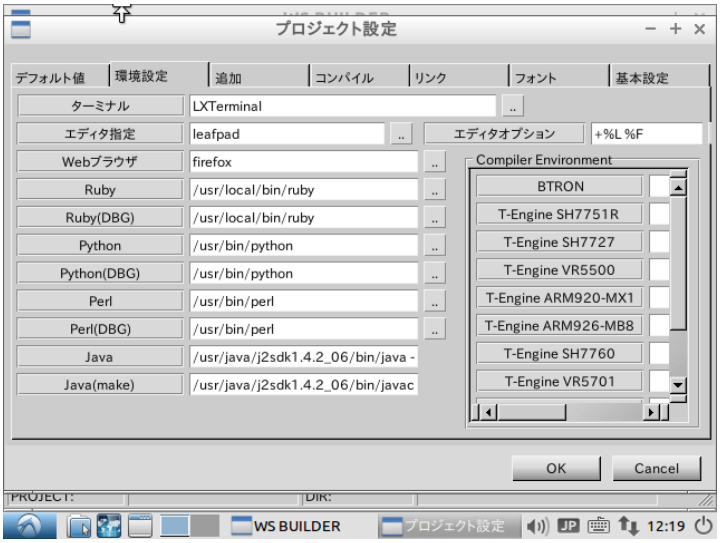


図 3-5-3-3. WideStudio の環境設定画面

３－５－４　　パッケージ更新の停止

Algonomix4 および Algonomix4 開発環境では、Lubuntu14.04.1 の基本パッケージと本書に表記したパッケージが入った環境でのみ動作を確認しています。意図しない動作を防ぐため、インターネットからのパッケージの追加に制限をかけます。

①　　図 3-5-4-1 のように[メニューアイコン]→[設定]→[ソフトウェアとアップデート]を選択して、設定アプリケーションを起動します。

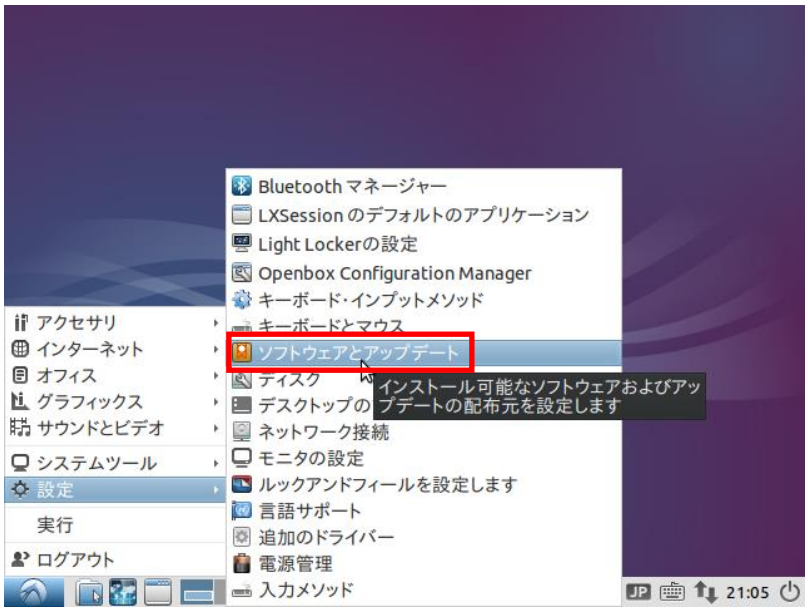


図 3-5-4-1. ［ソフトウェアとアップデート］の起動

②　　[Ubuntu のソフトウェア]タブ、[他のソフトウェア]タブ、[アップデート]タブを表示し、図 3-5-4-2 のようにチェックボックスを全て消し、[アップデートの自動確認]をなしに、Ubuntu の新バージョンの通知をなしにします。

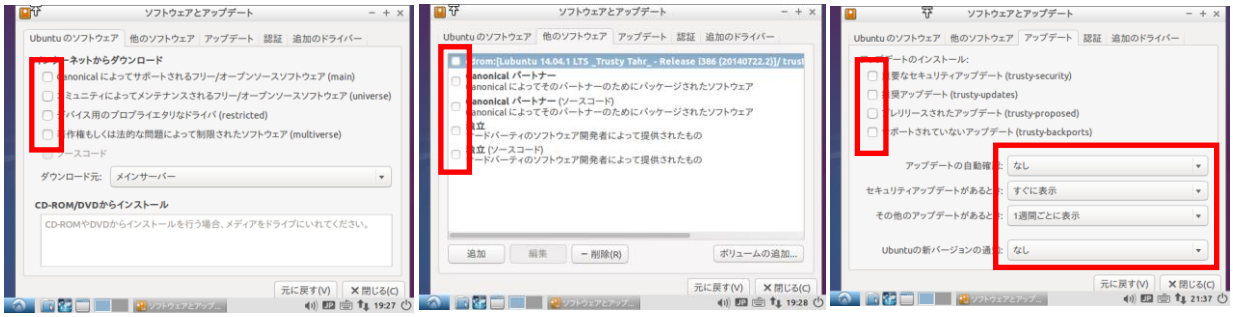


図 3-5-4-2.　ソフトウェアとアップデートの設定

③　　図 3-5-4-3 のような確認画面が表示されます。[再読込]をクリックしてください。

図 3-5-4-3.　確認画面

以上でパソコン上に Algonomix4 開発環境が構築されました。使用方法については、本書の開発環境の章を参照してください。

３－６　　パッケージについて

Algonomix4 および Algonomix4 開発環境では、Lubuntu14.04.1 の基本パッケージと本書に表記したパッケージが入った環境でのみ動作を確認しています。

そのため、インターネットからのパッケージの追加に制限をかけています。

以下にパッケージをさらに追加したい場合の制限の解除方法を示しますが、設定を変更してパッケージを追加された場合、動作の保証は致しかねますので、あらかじめご了承頂きますようお願い申し上げます。

Algonomix4 では、リポジトリを無効にすることでパッケージのインストールに制限をかけています。

制限の解除方法としては、/etc/apt/sources.list を直接編集する方法と、設定用アプリケーションにて間接的に編集する方法があります。ここでは、後者を紹介します。

① [メニューアイコン]→[設定]→[ソフトウェアとアップデート]を選択します。

図 3-6-1. ソフトウェアとアップデート設定画面の起動

②[Ubuntu のソフトウェア]タブや[アップデート]タブで許可したい項目にチェックを入れてください。一回目のチェックの時のみ図 3-6-2 のような認証を要求されます。パスワードを入れて[OK]を押してください。

図 3-6-2. ソフトウェアとアップデート設定画面

第 4 章 産業用組込み PC EC4A シリーズについて

本章では、EC4A 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。

Algonomix4 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『第 3 章 開発環境』を参照してください。

※注：コンソール用サンプルプログラムは「/usr/local/tools-0010/samples/sampleConsole」に、WideStudio 用サンプルプログラムは「/usr/local/tools-0010/samples/sampleWideStudio」に格納されています。

※注：一部のサンプルでは、EC4A シリーズにデバイスが実装されていない為動作しないことがあります。

表 4-1. コンソール用サンプルプログラムのソースコード一覧

ディレクトリ名	内容	機能有無
sample_DIO	汎用入出力制御方法	『4-1 汎用入出力』を参照
sample_SerialPortChange	シリアルポート RS422/R485/RS232C 切り替え	『4-2 シリアルポート』を参照
sample_Serial	シリアルポート制御方法	『4-2 シリアルポート』を参照
sample_TcpIp	ネットワークポート制御方法	『4-3 ネットワークポート』を参照
sample_BeepOnOff	ブザーON/OFF 制御方法	―
sample_Reset	汎用入力 IN0 リセット制御方法	『4-4 RAS 機能』を参照
sample_Interrupt	汎用入力 IN1 割込み制御	『4-4 RAS 機能』を参照
sample_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_Timer	汎用タイマ割込み制御	『4-5 タイマ割込み機能』を参照
sample_UPS	UPS 通知待ち	『4-6 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_GenIO	汎用入出力制御方法	『4-1 汎用入出力』を参照
sample_Sio	シリアルポート制御方法	『4-2 シリアルポート』を参照
sample_Lan	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_MultiLang	多言語表示	『4-7 その他のサンプルプログラム』を参照
sample_10Key	10 キー入力画面	『4-7 その他のサンプルプログラム』を参照
sample_Launcher	起動ランチャー	『4-7 その他のサンプルプログラム』を参照
sample_ColorPalette	カラーパレット画面	『4-7 その他のサンプルプログラム』を参照
sample_JapaneseInsert	日本語入力制御	『4-7 その他のサンプルプログラム』を参照
sample_Gui	hello サンプル	『3-3 WideStudio/MWT によるアプリケーション開発』を参照

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、EC4A 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。

EC4A シリーズ本体の外形図を図 4-1 に示します。各部名称を表 4-3 に示します。

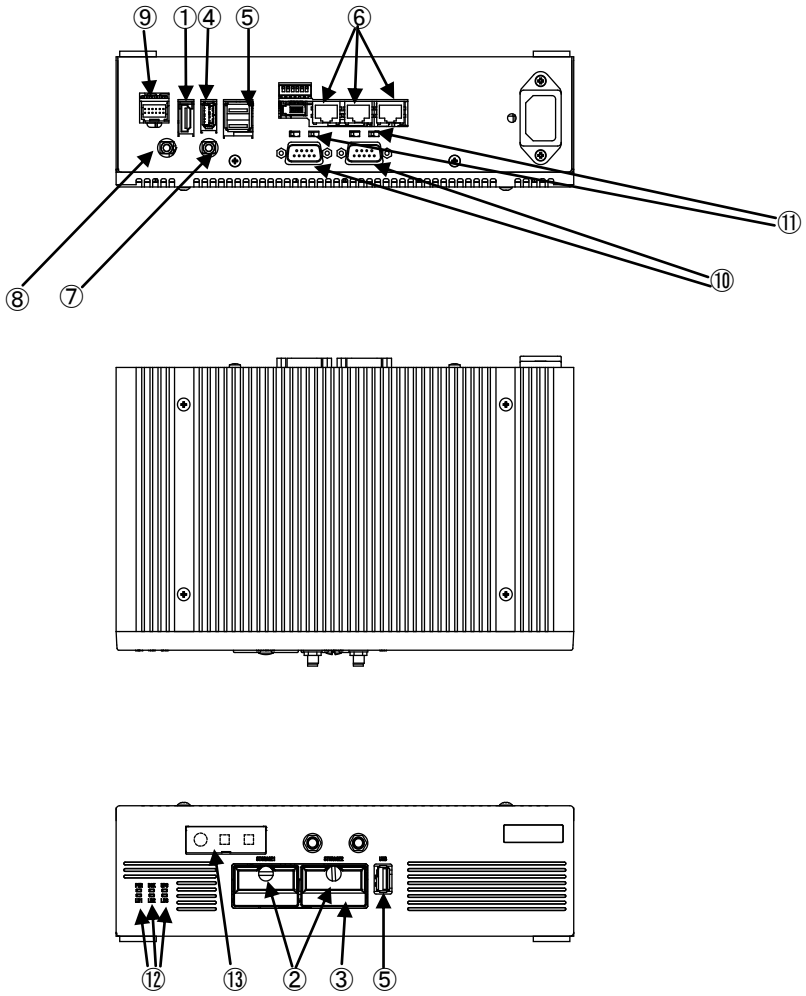


図 4-1. 外形図（EC4A-100AT）

表 4-3. 各部名称

No.	名称	機能	説明
①	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。
②	mSATA スロット	mSATA SSD スロット	記憶領域として mSATA SSD を使用することができます。
③	SD カード インターフェース	SD カード	記憶領域として SD カードを使用できます。
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。
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。
⑥	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。
⑦	音声入力	マイク	音声入力が使用できます。
⑧	音声出力	オーディオ	音声出力が使用できます。
⑨	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。
⑩	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。 SIO1～SIO2 は、シリアルポート設定スイッチと組込みシステム機能のシリアルコントロール機能と併用することで、232C/422/485 の通信を行うことができます。 SIO1: /dev/ttyS4 SIO2: /dev/ttyS5
⑪	シリアルポート 設定スイッチ	シリアルポート タイプ設定	SIO1、SIO2 のシリアルポートタイプを設定します。 シリアルポート設定スイッチと組込みシステム機能のシリアルコントロール機能を使用することで 232C/422/485 の通信を行うことができます。 設定方法は、「ユーザーズ（ハードウェア）マニュアル」を参照してください。 **※　シリアルポートタイプを切替える場合は、シリアルコントロール機能の設定に合わせて、シリアルポート設定スイッチを設定してください。**
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。
⑬	初期化 SW	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。

４－１　　汎用入出力

４－１－１　　汎用入出力について

EC4A 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　　汎用入出力デバイスドライバについて

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

表 4-1-2-1 に汎用入出力のリファレンスを示します。

表 4-1-2-1. 汎用入出力デバイスリファレンス

GENIN,GENOUT

名前

genin　－　汎用入力6点
genout　－　汎用出力4点

説明

汎用入力6点の状態読み出しと、汎用出力4点の制御を行います。

OPEN

汎用入力デバイス(/dev/genin,/dev/genout)をOpen関数でオープンします。

```
infd = open("/dev/genin", O_RDWR);
outfs = open("/dev/genout", O_RDWR);
```

READ

read関数を用いて汎用入出力の状態をモニタすることができます。

```
char in;
char out;
len = read(infs, &in, 1);
len = read(outfs, &out, 1);
```

汎用入出力デバイスをリードすると1Byteのデータが即リターンされます。
リターンされた値は現在の入出力状態です。

bit	7	6	5	4	3	2	1	0
IN			IN6	IN5	IN4	IN3	IN2	IN1

bit	7	6	5	4	3	2	1	0
OUT					OT4	OT3	OT2	OT1

WRITE

write関数を用いて汎用出力を制御することができます。

```
char out;
out = 1;
len = write(outfd, &out, 1);
```

汎用出力データを1Byteライトすることで対応するビットの出力をON/OFFすることができます。

bit	7	6	5	4	3	2	1	0
OUT					OT4	OT3	OT2	OT1

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-0010/samples/sampleWideStudio/sample_GenIO」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。

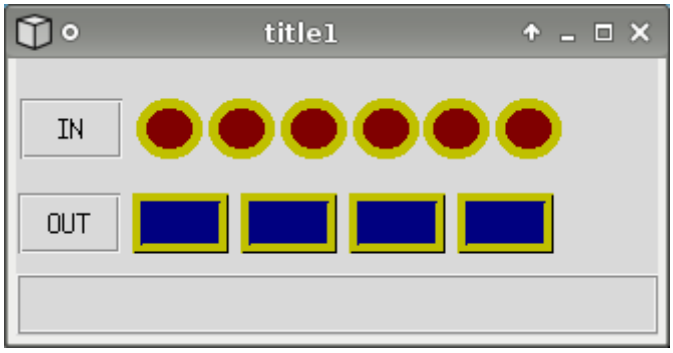


図 4-1-3-1. 汎用入出力デバイス制御サンプルプログラム起動画面

このサンプルプログラムは、スレッド内で汎用入力状態を監視し、入力状態によって IN の色を変化させます。また、OUT のボタンを押下することで、汎用出力が ON/OFF します。

汎用入出力デバイスのオープンとスレッドの生成を記したコードをリスト 4-1-3-1 に示します。

Main_Init 関数は、メインウィンドウの「Initialize」イベントで実行されるように設定されています。この中で、「open」関数を使用し、汎用入力のデバイスファイル「/dev/genin」と汎用出力のデバイスファイル「/dev/genout」をリードライトモードでオープンします。汎用入出力デバイスをリードライトすることで汎用入出力を制御することができます。

また、汎用入力の状態をモニタする為にスレッドを生成しています。

リスト 4-1-3-1. 汎用入出力デバイスのオープンとスレッドの生成（Main_Init.cpp）

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

#include <WScom.h>
#include <WSCfunctionList.h>
#include <WSCbase.h>

#include "newwin000.h"
#include "IOThread.h"

//---------------------------------------------------------
//Function for the event procedure
```

```
//---------------------------------------------------------
void Main_Init(WSCbase* object){
unsigned char data;

        /*汎用出力ﾃﾞﾊﾞｲｽのｵｰﾌﾟﾝ*/
    out_fd = open("/dev/genout", O_RDWR);        //汎用出力ﾃﾞﾊﾞｲｽｵｰﾌﾟﾝ
    if(out_fd < 0){                              //ｴﾗｰ処理
        Status_Bar->setProperty(WSNlabelString,"OUT Device Open Error");
        return;
    }
    data = 0;
    write(out_fd, &data, 1);                     //初期 0 出力

        /*汎用入力ﾃﾞﾊﾞｲｽのｵｰﾌﾟﾝ*/
    in_fd = open("/dev/genin", O_RDWR);          //汎用入力ﾃﾞﾊﾞｲｽｵｰﾌﾟﾝ
    if(in_fd < 0){                               //ｴﾗｰ処理
        Status_Bar->setProperty(WSNlabelString,"IN Device Open Error");
        close(out_fd);
        return;
    }

        /*ｽﾚｯﾄﾞの生成*/
    ioctrl_thr = 0;
    ioctrl_thr = WSDthread::getNewInstance();           //ｽﾚｯﾄﾞｲﾝｽﾀﾝｽ取得
    ioctrl_thr->setFunction(Ioctrl_Thread);             //ｽﾚｯﾄﾞ本体関数を設定
    ioctrl_thr->setCallbackFunction(Io_callback_func);  //ｺｰﾙﾊﾞｯｸ関数を設定
    ioctrl_thr->createThread((void*)0);                 //ｽﾚｯﾄﾞを生成
}
static WSCfunctionRegister  op("Main_Init",(void*)Main_Init);
```

リスト 4-1-3-2 に汎用入力をモニタするスレッド本体とコールバック関数のソースコードを示します。

汎用入力デバイスを「read」関数を使い、1 バイトリードすることで、現在の汎用入力の状態を読み出すことができます。汎用入力デバイスは、「read」関数を実行されたら、遅延せずに即時に ON/OFF 状態を返します。このサンプルプログラムのソースコードでは、汎用入力の状態が変化したときのみコールバック関数を呼び出し画面の状態を変化させています。

リスト 4-1-3-2. 汎用入力スレッドのソースコード（IOThread.cpp）

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

#include <WScom.h>
#include <WSCfunctionList.h>
```

```
#include <WSCbase.h>

#include "newwin000.h"
#include "IOThread.h"

int in_fd;
int out_fd;
WSDthread* ioctrl_thr;
unsigned char o_data;

void *Ioctrl_Thread(WSDthread* obj, void *arg)
{
    int len;
    unsigned char data;
    o_data = data = 0;
    int i;
    i = 0;
    for(;;){
        len = read(in_fd, &data, 1);                    /*汎用入力読みだし*/
        data &= 0x3F;
        if(len==1){
            if(o_data != data){
                o_data = data;
                obj->execCallback((void*)data);
            }
        }
    }
    return(NULL);
}
/*ｽﾚｯﾄﾞから通知され、ﾒｲﾝｽﾚｯﾄﾞで実行されるｺｰﾙﾊﾞｯｸ関数*/
void Io_callback_func(WSDthread *, void *val)
{
unsigned char data;

    data = (unsigned char)(unsigned long)val;
    if(data & 0x01){                    /*入力 1*/
        newvarc_000->setPropertyV(WSNhatchColor,"#FF0000");
    }
    else{
        newvarc_000->setPropertyV(WSNhatchColor,"#800000");
    }
    if(data & 0x02){                    /*入力 2*/
        newvarc_001->setPropertyV(WSNhatchColor,"#FF0000");
    }
    else{
        newvarc_001->setPropertyV(WSNhatchColor,"#800000");
    }
    if(data & 0x04){                    /*入力 3*/
        newvarc_002->setPropertyV(WSNhatchColor,"#FF0000");
    }
    else{
```

```
        newvarc_002->setPropertyV(WSNhatchColor,"#800000");
    }
    if(data & 0x08){                    /*入力 4*/
        newvarc_003->setPropertyV(WSNhatchColor,"#FF0000");
    }
    else{
        newvarc_003->setPropertyV(WSNhatchColor,"#800000");
    }
    if(data & 0x10){                    /*入力 5*/
        newvarc_004->setPropertyV(WSNhatchColor,"#FF0000");
    }
    else{
        newvarc_004->setPropertyV(WSNhatchColor,"#800000");
    }
    if(data & 0x20){                    /*入力 6*/
        newvarc_005->setPropertyV(WSNhatchColor,"#FF0000");
    }
    else{
        newvarc_005->setPropertyV(WSNhatchColor,"#800000");
    }
}
```

汎用出力を制御するソースコードをリスト 4-1-3-2 に示します。

メインウィンドウに配置したボタンに「Activate」イベントのプロシージャを作成します。その中で、汎用出力デバイスを「read」関数で現在の状態を読込み、新しい状態に変更し「write」関数でデータを更新します。これでボタンが押下されるたびに汎用出力の状態が切り替わります。

リスト 4-1-3-3. 汎用出力 ON/OFF のソースコード (Btn_Click.cpp)

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

#include <WScom.h>
#include <WSCfunctionList.h>
#include <WSCbase.h>

#include "newwin000.h"
#include "IOThread.h"

int btn[4]={0,0,0,0};
//-------------------------------------------------------------
//Function for the event procedure
//-------------------------------------------------------------
```

```
void Btn_Click(WSCbase* object){
    int len;
    unsigned char data;
    long code;

    code = object->getProperty(WSNuserValue);              //押されたボタン番号を取得
    len = read(out_fd, &data, 1);                          //汎用出力の出力値取得
    if(len==1){
        if(btn[code]){
            data &= (~(0x0001 << code));                   //出力 OFF
            object->setProperty(WSNbackColor,"#000080");
            btn[code] = 0;
        }
        else{
            data |= (0x0001 << code);                      //出力 ON
            object->setProperty(WSNbackColor,"#0000FF");
            btn[code] = 1;
        }
        len = write(out_fd, &data, 1);                     //汎用出力更新
    }
}
static WSCfunctionRegister  op("Btn_Click",(void*)Btn_Click);
```

●コンソール用サンプルプログラム

「/usr/local/tools-0010/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。コンソールウィンドウを起動して、コンパイルしたコマンドを実行します。ソースコードをリスト 4-1-3-4 に示します。

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

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

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

リスト 4-1-3-4. 汎用入出力 ON/OFF のソースコード (main.c)

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

int main(int argc, char *argv[])
{
    int             i;
```

```
int             err_no;
int             outfd;
int             infd;
unsigned char   outdata=0x01;
unsigned char   indata=0x00;
ssize_t         len;

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

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

/* 汎用出力
    汎用出力の 4 点を 1 点ずつ出力します。
*/
for (i=0; i<4; i++){
    len = write(outfd, &outdata, 1);        /* 汎用出力更新 */
    if (len == -1){
        err_no = errno;
        fprintf(stderr, "out device write: %s¥n", strerror(err_no));
        /* 汎用出力デバイスのｸﾛｰｽﾞ */
        close(outfd);
        return (-1);
    }
    outdata <<= 1;
    usleep(500 * 1000L);
}
outdata = 0x00;
len = write(outfd, &outdata, 1);            /* 汎用出力すべて OFF */
if (len == -1){
    err_no = errno;
    fprintf(stderr, "out device write: %s¥n", strerror(err_no));
    /* 汎用出力デバイスのｸﾛｰｽﾞ */
    close(outfd);
    return (-1);
}

/* 汎用入力
    汎用入力の 6 点ずつ出力を行います。
```

```
    */
    len = read(infd, &indata, 1);              /* 汎用入力の値を取得 */
    if (len == -1){
        err_no = errno;
        fprintf(stderr, "in device read: %s¥n", strerror(err_no));
        /* 汎用入出力デバイスのｸﾛｰｽﾞ */
        close(outfd);
        close(infd);
        return (-1);
    }
    else if (len == 0){
        fprintf(stderr, "in device read: len zero¥n");
        /* 汎用入出力デバイスのｸﾛｰｽﾞ */
        close(outfd);
        close(infd);
        return (-1);
    }
    else{
        indata &= 0x3F;
        /* IN0 状態 */
        if (indata & 0x01) fprintf(stdout, "IN0: ON¥n");
        else               fprintf(stdout, "IN0: OFF¥n");
        /* IN1 状態 */
        if (indata & 0x02) fprintf(stdout, "IN1: ON¥n");
        else               fprintf(stdout, "IN1: OFF¥n");
        /* IN2 状態 */
        if (indata & 0x04) fprintf(stdout, "IN2: ON¥n");
        else               fprintf(stdout, "IN2: OFF¥n");
        /* IN3 状態 */
        if (indata & 0x08) fprintf(stdout, "IN3: ON¥n");
        else               fprintf(stdout, "IN3: OFF¥n");
        /* IN4 状態 */
        if (indata & 0x10) fprintf(stdout, "IN4: ON¥n");
        else               fprintf(stdout, "IN4: OFF¥n");
        /* IN5 状態 */
        if (indata & 0x20) fprintf(stdout, "IN5: ON¥n");
        else               fprintf(stdout, "IN5: OFF¥n");
    }

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

    return(0);
}
```

４－２　　シリアルポート

４－２－１　　シリアルポートについて

EC4A シリーズは、RS232C、RS422、RS485 を切り替えることができるシリアルポートを 2 つ持っており、それぞれをユーザアプリケーションで使用できます。

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

表 4-2-1-1. シリアルタイプ別のデバイスファイル名

ポート番号	デバイスファイル	用途
1	/dev/ttyS4	汎用のシリアルポート 1
2	/dev/ttyS5	汎用のシリアルポート 2

アプリケーションでシリアルポートを使用するには、RS232C、RS422、RS485 のシリアルタイプを切り替えて、それぞれのデバイスファイルをオープンし、Read／Write することで制御します。

シリアルポートのタイプを切り替えるにはシリアルポート切り替えドライバを使用します。シリアルポートの切り替えは、下記に示す 2 種類の方法で行うことができます。詳細は『4-2-2 シリアルポートデバイスドライバについて』を参照してください。

1. デバイスドライバを使用する方法
2. シリアル切り替えコマンドを使用する方法

※注：同一ポートで RS232C、RS422、RS485 同時の使用はできません。
※注：SIO ポート設定スイッチも選択したタイプに設定する必要があります。

４－２－２　　シリアルポートデバイスドライバについて

●シリアルポートデバイスドライバについて

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。

●RS232C/RS422/RS485 切り替え用デバイスドライバ

各ポートは通信方法制御デバイス（/dev/scictl）を用いて RS232C/RS422/RS485 の切替えを行うことができます。表 4-2-2-1 にデバイスの詳細を示します。

表 4-2-2-1. シリアルポート通信方法制御デバイスリファレンス

SCICTL

名前

シリアルポートの通信方法を制御します。

ヘッダ

```
#include "scictl_ioctl.h"
```

説明

シリアルポートのRS232C/RS422/RS485の通信方法の設定と取得を行うことができます。

OPEN

デバイスファイル（/dev/scictl）を**open**関数でオープンします。

```
fd = open("/dev/scictl", O_RDWR);
```

IOCTL

通信方法を制御するには**ioctl**関数を用います。

```
error = ioctl(fd, ioctl_type, &conf);
```

- ioctl_type
  - ASD_SCICTL_IOCSCONF
    通信方法を設定します。
  - ASD_SCICTL_IOCGCONF
    現在の通信方法を取得します。
- conf
  通信方法設定データです。以下の構造体になります。

```
struct scictl_conf {
    unsigned short ch;
    unsigned short type;
    unsigned short timer;
};
```

ch	ポート番号 1 : ttyS4 2 : ttyS5
type	ポート通信方法 0 : RS232C 1 : RS422 2 : RS485
timer	RS485用のTXディセーブルタイマ【ミリ秒】 送信完了から、この設定時間の間、再送信がなければTXがディセーブルされます。

●RS232C/RS422/RS485 切り替えコマンド

RS232C/RS422/RS485 切り替えは、Algo Smart Panel 付属のコマンドでも行うことができます。表 4-2-2-2 に詳細を示します。

表 4-2-2-2. シリアルポート通信切り替えコマンドリファレンス

SCICTL_CONF

名前

シリアルポート通信切換えコマンド

書式

scictl_conf　ch type timer

説明

シリアルポートのRS232C/RS422/RS485の通信方法の設定を行います。

引数：

ch	ポート番号 1 : ttyS4 2 : ttyS5
type	ポート通信方法 0 : RS232C 1 : RS422 2 : RS485
timer	RS485用のTXディセーブルタイマ【ミリ秒】 送信完了から、この設定時間の間、再送信がなければ TXがディセーブルされます。

ポート 1 を RS422 に設定する場合は、以下の様にコマンドを実行します。

```
# scictl_conf 1 1 0
ch=1, type=1, timer=0
```

４－２－３　　シリアルポートサンプルプログラム

●WideStudio 用サンプルプログラム

「/usr/local/tools-0010/samples/sampleWideStudio/sample_Sio」に、シリアルポートを使ったサンプルプログラムが入っています。

図 4-2-3-1. シリアルポートデバイス制御サンプル画面

このサンプルプログラムは、スレッド内でシリアルポート 0 からの受信待ちを行い、受信した文字をそのまま送信し、同時に中央のテキストフィールドに受信文字列を表示します。

シリアルポートデバイスの切り替えと、オープン、スレッドの生成を記したソースコードをリスト 4-2-3-1 に示します。

シリアルポートデバイスをオープンする前に、シリアルポート切り替えデバイスを「open」関数でオープンし、RS232C として使用するように設定しています。

「open」関数で通信を行いたいポートのデバイスをオープンします。(サンプルではシリアルポート 1 デバイスファイル名「/dev/ttyS4」をオープンしています。)

「O_NOCTTY」は制御端末として使用しないモードでオープンします。「tcgetattr」関数で現状の通信設定(「termios」構造体)を取得します。「termios」構造体のメンバの値を変更することで通信設定を変更します。「tcsetattr」関数で変更した通信設定を反映します。これで通信できる状態になります。

「termios」構造体、ならびにシリアルデバイスの使用方法等の詳細については、インターネットや書籍を参照してください。

このサンプルプログラムの通信設定は 8bit 長、パリティ無し、ストップビット 1bit、ボーレート 38400bps となっています。

リスト 4-2-3-1. シリアルポートデバイスの切り替えとオープンとスレッドの生成 (Main_Init.cpp)

```
#include <termios.h>
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/signal.h>
#include <sys/types.h>

#include <WScom.h>
#include <WSCfunctionList.h>
#include <WSCbase.h>

#include "scictl_ioctl.h"

#include "newwin000.h"
#include "ComThread.h"
//-------------------------------------------------------------
//Function for the event procedure
//-------------------------------------------------------------
void Main_Init(WSCbase* object){
    struct termios tio;
    int stat;
    int desc;
    int ret;
    struct scictl_conf  conf;

    /*
     * ポート切り替えデバイスをオープン
     *
     */
    desc = open("/dev/scictl", O_RDWR);
    if (desc == -1){
        Status_Bar->setProperty(WSNlabelString,"scictl Open Error");
        return;
    }
```

```
/*
 * ttyS1 を RS232C に変更
 *
 *   conf.ch
 *       ポート番号  1: ttyS4
 *                   2: ttyS5
 *
 *   conf.type
 *       ポートタイプ  0: RS232C
 *                     1: RS422
 *                     2: RS485
 *
 *   conf.timer
 *       RS485 用 TX ディセーブルタイマ[マイクロ秒]
 *       送信完了から設定時間の間、再送信がなければ
 *       TX がディセーブルされます。
 */
conf.ch = 1;
conf.type = 0;
conf.timer = 1000;
ret = ioctl(desc, ASP_SCICTL_IOCSCONF, &conf);
if (ret == -1){
    Status_Bar->setProperty(WSNlabelString,"scictl ioctl Error");
    close(desc);
    return;
}
close(desc);

    /*シリアルポートオープン*/
comm_fd = open("/dev/ttyS4", O_RDWR | O_NOCTTY);        //シリアルポートオープン
if(comm_fd < 0){
    Status_Bar->setProperty(WSNlabelString,"ttyS1 Open Error");
    return;
}

stat = tcgetattr(comm_fd, &tio);                          //現在の通信設定を待避
if(stat < 0){
    Status_Bar->setProperty(WSNlabelString,"Terminal Attribute Get Error");
    close(comm_fd);
    return;
}

//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] = 50;                          //キャラクタ間タイムアウト時間 50ms
    tio.c_cc[VMIN] = 1;                            //1 文字取得するまでﾌﾞﾛｯｸ
    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);

    stat=tcsetattr(comm_fd, TCSAFLUSH, &tio);      //変更した通信設定の反映
    if(stat < 0){
        Status_Bar->setProperty(WSNlabelString,"Terminal Attribute Set Error");
        close(comm_fd);
        return;
    }

        /*スレッドの生成*/
    comctrl_thr = 0;
    comctrl_thr = WSDthread::getNewInstance();             //スレッドインスタンス取得
    comctrl_thr->setFunction(Comctrl_Thread);              //スレッド本体関数を設定
    comctrl_thr->setCallbackFunction(Com_callback_func);   //コールバック関数を設定
    comctrl_thr->createThread((void*)0);                   //スレッドを生成
}
static WSCfunctionRegister  op("Main_Init",(void*)Main_Init);
```

リスト 4-2-3-2 にシリアルポートから 1 バイト受信して、そのまま送信するスレッドのソースコードを示します。

「read」関数で 1 バイト受信するまで待ちます。1 バイト受信されたら、「write」関数を使用して 1 バイト送信し、同時にテキストフィールドに受信した文字を表示します。

リスト 4-2-3-2. シリアルポートデバイスからの受送信のソースコード（ComThread.cpp）

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <sys/ioctl.h>
```

```
#include <termios.h>
#include <pthread.h>
#include <fcntl.h>
#include <unistd.h>
#include <errno.h>
#include <math.h>

#include <WScom.h>
#include <WSCfunctionList.h>
#include <WSCbase.h>

#include "newwin000.h"
#include "ComThread.h"

int comm_fd;
WSDthread* comctrl_thr;

void *Comctrl_Thread(WSDthread* obj, void *arg)
{
    int len;
    char data;
    int i;
    i = 0;
    for(;;){
       len = read(comm_fd, &data, 1);              /*1 バイト受信*/
       if(len==1){
           write(comm_fd, &data, 1);               /*1 バイト送信*/
           obj->execCallback((void*)data);
       }
    }
    return(NULL);
}
/*スレッドから通知され、メインスレッドで実行されるコールバック関数*/
void Com_callback_func(WSDthread *, void *val)
{
char data[2];

    data[0] = (char)(unsigned long)val;
    data[1] = 0;
    newtext_001->addString(data);
}
```

●コンソール用サンプルプログラム

「/usr/local/tools-0010/samples/sampleConsole/sample_SerialPortChange」に、シリアルポート切替えを行うサンプルプログラムが入っています。コンソールウィンドウを起動して、コンパイルしたコマンドを実行します。ソースコードをリスト 4-2-3-3 に示します。

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

リスト 4-2-3-3. シリアルポート切替えのソースコード（main.c）

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

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

    /*
     * ttyS4 を RS485 に変更
     *
     *   conf.ch
     *       ポート番号  1: ttyS4
     *                   2: ttyS5
     *   conf.type
     *       ポートタイプ  0: RS232C
     *                     1: RS422
     *                     2: RS485
     *
     *   conf.timer
     *       RS485 用 TX ディセーブルタイマ[ミリ秒]
```

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

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

「/usr/local/tools-0010/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 4-2-3-4 に示します。

シリアルポート 1「/dev/ttyS4」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 4-2-3-4. 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/ttyS4", O_RDWR | O_NOCTTY);
if (comm_fd == -1){ /* エラー処理 */
    err_no = errno;
    fprintf(stderr, "ttyS4 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, "ttyS4 tcgetattr_1: %s¥n", strerror(err_no));
    close(comm_fd);
    return (-1);
}

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

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

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

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

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

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

    /* シリアルデータ受信処理 */
    while (1){
        memset(rbuf, '¥0', 100);
        /* 100 バイト単位で受信 */
        size = read(comm_fd, &rbuf[0], 100);
        if (size == -1){
            err_no = errno;
            fprintf(stderr, "ttyS4 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, "ttyS4 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, "ttyS4 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。

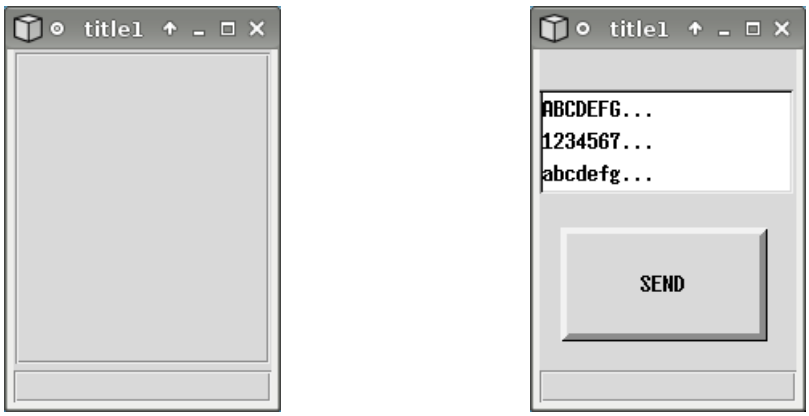
それぞれのシステムコール関数の詳細については、書籍やインターネットを参照してください。

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

●WideStudio 用サンプルプログラム

「/usr/local/tools-0010/samples/sampleWideStudio/sample_Lan」に、ネットワーク通信を使ったサンプルプログラムが入っています。



（サーバー画面）　　（クライアント画面）

図 4-3-3-1. ネットワークサンプルプログラム画面

このサンプルプログラムは、クライアント画面で選択した文字列を「SEND」ボタンをクリックすることでサーバに送り、サーバ側で、受信した文字列をテキストフィールドに表示します。クライアント側のプログラムを起動するときには、サーバプログラムが起動しているパソコンの IP アドレスを引数として入力します。ここでは、同一のパソコン上で動作させるので、ローカルループバックアドレスを指定しています。

EC4A シリーズ本体にサーバプログラムとクライアントプログラムを送り、コンソールウィンドウを起動してプログラムを実行します。図 4-3-3-1 のような画面が起動します。

```
# ./server_spl &
# ./client_spl 127.0.0.1 &
```

リスト 4-3-3-1 にサーバソケット作成を行うソースコードを示します。

接続待ちを行う、サーバソケットを作成し、実際に待ちを行う為のスレッドを作成します。各関数の引数の意味については、インターネットや書籍を参照してください。

リスト 4-3-3-1. サーバソケット生成（./server/Main_Init.cpp）

```
#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/socket.h>
#include <arpa/inet.h>
#include <bits/local_lim.h>

#include <WScom.h>
#include <WSCfunctionList.h>
```

```
#include <WSCbase.h>

#include "newwin000.h"
#include "SrvThread.h"

//-----------------------------------------------------------
//Function for the event procedure
//-----------------------------------------------------------
void Main_Init(WSCbase* object){
        /* ｿｹｯﾄ生成 */
    if ((srv_sock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)) < 0) {
        Status_Bar->setProperty(WSNlabelString,"socket() failed");
        return;
    }

        /* 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
    if (bind(srv_sock, (struct sockaddr *) &srv_addr, sizeof(srv_addr)) < 0) {
        Status_Bar->setProperty(WSNlabelString,"bind() failed");
        close(srv_sock);
        return;
    }

        /* カーネル通知 */
    if (listen(srv_sock, 1) < 0) {
        Status_Bar->setProperty(WSNlabelString,"listen() failed");
        close(srv_sock);
        return;
    }
        /*スレッドの生成*/
    srvctrl_thr = 0;
    srvctrl_thr = WSDthread::getNewInstance();              //スレッドインスタンス取得
    srvctrl_thr->setFunction(Srvctrl_Thread);               //スレッド本体関数を設定
    srvctrl_thr->setCallbackFunction(Srv_callback_func);    //コールバック関数を設定
    srvctrl_thr->createThread((void*)0);                    //スレッドを生成
}
static WSCfunctionRegister  op("Main_Init",(void*)Main_Init);
```

リスト 4-3-3-2 に接続待ちとクライアントからのデータ受信待ちを行うスレッドのソースコードを示します。

「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。通信確立済みのソケット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、受信した文字列をテキストフィールドに表示します。

リスト 4-3-3-2. クライアント接続待ちおよびデータ受信待ちスレッド（./server/SrvThread.cpp）

```
#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/socket.h>
#include <arpa/inet.h>
#include <bits/local_lim.h>

#include <WScom.h>
#include <WSCfunctionList.h>
#include <WSCbase.h>

#include "newwin000.h"
#include "SrvThread.h"

char  tmp[BUFFER];
int flg;
int srv_sock;
int cli_sock;
struct sockaddr_in srv_addr;
struct sockaddr_in cli_addr;
WSDthread* srvctrl_thr;

void *Srvctrl_Thread(WSDthread* obj, void *arg)
{
    int len;

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

        for(;;){
            /* ﾃﾞｰﾀ受信待ち */
            if(flg == 0){ /*exeCallback 関数が実行されるまで処理しない*/
                len = recv(cli_sock, tmp, BUFFER, 0);
                if (len > 0){
                    flg = 1;
                    obj->execCallback((void *)len); /*正常受信完了*/
                }else{
                    close(cli_sock);                /*通信断*/
                    break;
                }
            }
            usleep(10*1000);    /*10mswait*/
        }
```

```
    }
    return(NULL);
}
/*ｽﾚｯﾄﾞから通知され,ﾒｲﾝｽﾚｯﾄﾞで実行されるｺｰﾙﾊﾞｯｸ関数*/
void Srv_callback_func(WSDthread *, void *val)
{
int len;

    len = (int)val;
    tmp[len] = 0;
    newvlab_000->setProperty(WSNlabelString, tmp);
    flg = 0;
}
```

リスト 4-3-3-3 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。通信確立したら 0 が、エラーなら-1 が返されます。これで、通信できる状態です。

リスト 4-3-3-3. クライアントソケット生成 (./client/Main_Init.cpp)

```
#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/socket.h>
#include <arpa/inet.h>
#include <bits/local_lim.h>

#include <WScom.h>
#include <WSCfunctionList.h>
#include <WSCbase.h>
#include <WSDappDev.h>

#include "newwin000.h"

int sock;
struct sockaddr_in srv_addr;
//------------------------------------------------------------
//Function for the event procedure
//------------------------------------------------------------
void Main_Init(WSCbase* object){
    char **argv;
    char *srv_ip;

    /* プログラム起動時の引数でサーバの IP アドレスを取得 */
    if(WSGIappDev()->getArgc() < 2){
```

```
        Status_Bar->setProperty(WSNlabelString,"No IP Address");
        return;
    }
    argv = WSGIappDev()->getArgv();
    srv_ip = *(argv + 1);


    /* ｸﾗｲｱﾝﾄｿｹｯﾄ作成*/
    if ((sock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)) < 0) {
        Status_Bar->setProperty(WSNlabelString,"socket() failed");
        return;
    }

    /* ｻｰﾊﾞに接続 */
    memset(&srv_addr, 0, sizeof(srv_addr));
    srv_addr.sin_family      = AF_INET;
    srv_addr.sin_addr.s_addr = inet_addr(srv_ip);       //ターゲットサーバ IP アドレス
    srv_addr.sin_port        = htons(8900);             //ターゲットサーバポート番号
    if (connect(sock, (struct sockaddr *) &srv_addr, sizeof(srv_addr)) < 0) {
        Status_Bar->setProperty(WSNlabelString,"connect failed");
        close(sock);
        return;
    }

    /* list 初期化 */
    newlist_000->delAll();
    newlist_000->addItem("ABCDEFG...");
    newlist_000->addItem("1234567...");
    newlist_000->addItem("abcdefg...");
    newlist_000->updateList();

}
static WSCfunctionRegister  op("Main_Init",(void*)Main_Init);
```

ボタンの「ACTIVATE」プロシージャで、リストから選んだ文字列を送信します。リスト 4-3-3-4 に文字列送信のソースコードを示します。

リストボックスから送信する文字列を選択し、ボタンを押します。「send」関数で該当する文字列を送信します。

リスト 4-3-3-4. クライアントプログラムボタン「ACTIVATE」プロシージャ (./client/Btn_Click.cpp)

```
#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/socket.h>
#include <arpa/inet.h>
#include <bits/local_lim.h>
```

```
#include <WScom.h>
#include <WSCfunctionList.h>
#include <WSCbase.h>

#include "newwin000.h"

#define  BUFFER  2048

extern int sock;
extern struct sockaddr_in srv_addr;
const char dat[][21]={
    {"ABCDEFGHIJKLMNOPQRST"},
    {"12345678901234567890"},
    {"abcdefghijklmnopqrst"}
};

//-------------------------------------------------------------
//Function for the event procedure
//-------------------------------------------------------------
void Btn_Click(WSCbase* object){
long d;

    d = newlist_000->getSelectedPos();
    if((d < 0)||(d > 2)) return;
    /* データ送信 */
    if (send(sock, &dat[d][0], 20, 0) != 20) {
        Status_Bar->setProperty(WSNlabelString,"send() failed");
    }
}
static WSCfunctionRegister  op("Btn_Click",(void*)Btn_Click);
```

●コンソール用サンプルプログラム

「/usr/local/tools-0010/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 EC4A 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 4-3-3-5 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 4-3-3-5. 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 4-3-3-6 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 4-3-3-6. 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。
デバイスドライバからこの機能の有効・無効を切り替えることができます。

４－４－２　汎用入力 IN0 リセットデバイスドライバについて

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

表 4-4-2-1. 汎用入力 IN0 デバイスリファレンス

RASIN

名前

汎用入力IN0リセット機能の制御を行います。

インクルード

```
#include "rasin.h"
```

説明

汎用入力IN0リセット機能の有効・無効設定及び設定値の取得を行います。
本設定は、本体再起動後に有効となります。

OPEN

汎用入力IN0リセットデバイス(/dev/rasin)を**open**関数でオープンします。

```
rasinfd  = open("/dev/rasin", O_RDWR);
```

IOCTL

IN0リセットを制御するには**ioctl**関数を使用します。

```
error = ioctl(fd, ioctl_type, &conf);
```

- ioctl_type
  RASIN_IOCSINORST
  　有効・無効を設定します。
  RASIN_IOCGINORST
  　現在の設定値を取得します。

- conf
  0　無効
  1　有効

４－４－３　汎用入力 IN0 リセットサンプル

●Console 用サンプルプログラム

「/usr/local/tools-0010/samples/sampleConsole/sample_Reset」に、汎用入力 IN0 リセットを使用するサンプルプログラムが入っています。このサンプルプログラムはコンソールアプリケーションとして作成されています。コンソール上で動作させることができます。リスト 4-4-3-1 にソースコードを示します。

リスト 4-4-3-1. 汎用入力 IN0 リセット (main.c)

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

#include "rasin.h"

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

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

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

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

    /*
     * IN0 リセットを有効にする
     *
     * onoff: 1    有効
     *      : 0    無効
     */
    onoff = 1;
    len = ioctl(desc, RASIN_IOCSINORST, &onoff);
    if (len == -1) {
        err_no = errno;
        fprintf(stderr, "ioctl set INORST faild: %s¥n", strerror(err_no));
        close(desc);
        return (-1);
    }
    else {
        fprintf(stdout, "ioctl set INORST success: %d¥n", onoff);
    }
    close(desc);
    return 0;
}
```

４－４－４　汎用入力 IN1 割込みについて

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

４－４－５　汎用入力 IN1 割込みデバイスドライバについて

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

表 4-4-5-1. 汎用入力 IN1 割込みデバイスリファレンス

RASIN

名前

汎用入力IN1割込み機能の制御を行います。

インクルード

```
#include "rasin.h"
```

説明

汎用入力IN1割込み機能の有効・無効設定及び設定値の取得を行います。
本設定は、本体再起動後に有効となります。

OPEN

汎用入力IN1割込みデバイス(/dev/rasin)を**open**関数でオープンします。

```
rasinfd  = open("/dev/rasin", O_RDWR);
```

IOCTL

IN1割込みを制御するには**ioctl**関数を使用します。

```
error = ioctl(fd, ioctl_type, &conf);
```

- ioctl_type
  - RASIN_IOCSIN1INT
    有効・無効を設定します。
  - RASIN_IOCGIN1INT
    現在の設定値を取得します。

- conf
  - 0　無効
  - 1　有効

４－４－６　汎用入力 IN1 割込みサンプル

●Console 用サンプルプログラム

「/usr/local/tools-0010/samples/sampleConsole/sample_Interrupt」に、汎用入力 IN1 割込みを使用するサンプルプログラムが入っています。このサンプルプログラムはコンソールアプリケーションとして作成されています。コンソール上で動作させることができます。リスト 4-4-6-1 にソースコードを示します。

リスト 4-4-6-1. 汎用入力 IN1 割込み（main.c）

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

#include "rasin.h"

static int  SigOnOff=0;

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

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

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

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

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

    /*
     * 現在の設定値取得
     *
     * onoff: 1    有効
     *      : 0    無効
     */
    len = ioctl(desc, RASIN_IOCGIN1INT, &onoff);
    if (len == -1) {
        err_no = errno;
        fprintf(stderr, "ioctl get IN1INT faild: %s¥n", strerror(err_no));
        close(desc);
        return (-1);
    }
    else {
        fprintf(stdout, "ioctl get IN1INT success: %d¥n", onoff);
    }

    /*
     * IN1 割込みを有効にする
     *
     * 有効の場合、無効に変更し終了
     * onoff: 1    有効
     *      : 0    無効
     */
    if (onoff != 1) onoff = 1;
    else            onoff = 0;
    len = ioctl(desc, RASIN_IOCSIN1INT, &onoff);
    if (len == -1) {
        err_no = errno;
        fprintf(stderr, "ioctl set IN1INT faild: %s¥n", strerror(err_no));
        close(desc);
        return (-1);
    }
    else {
```

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

４－４－７　　ハードウェア・ウォッチドッグタイマ機能について

EC4A シリーズには、ハードウェアによるウォッチドッグタイマ機能が搭載されています。
デバイスドライバ（/dev/rashwwdt）にアクセスすることにより、この機能を使用することができます。
シャットダウン、再起動、ポップアップ通知、ログへの出力は、デーモン（HWWDTD）によって行われています。
図 4-4-7-1 に、デバイス、ドライバ、アプリケーション、設定ツール、デーモンの関係を示します。

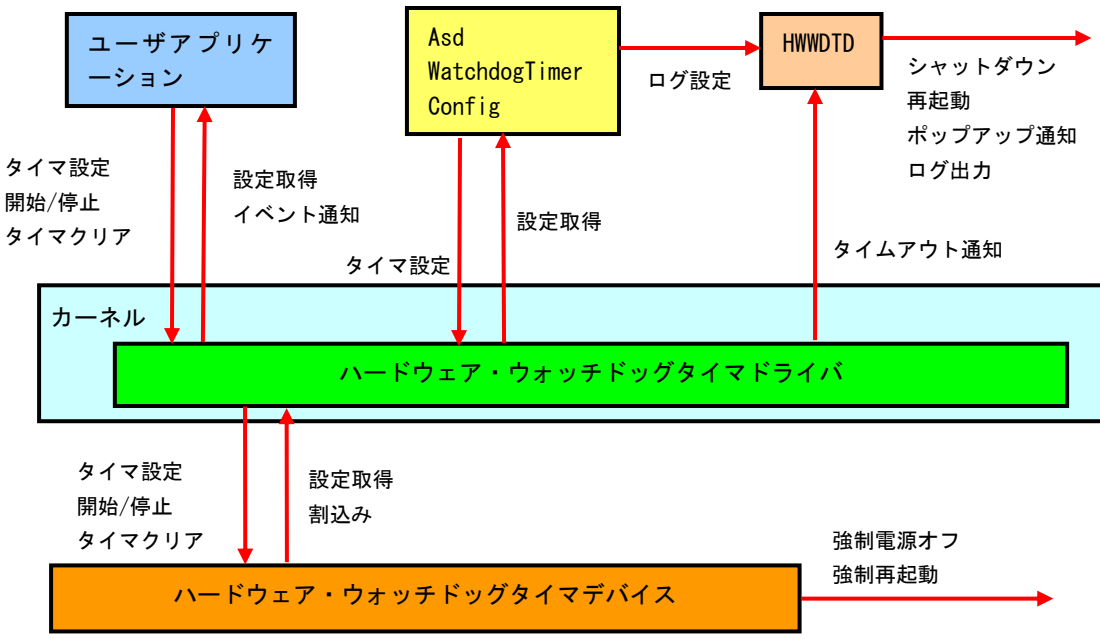


図 4-4-7-1. ハードウェア・ウォッチドッグタイマの構成

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

ハードウェア・ウォッチドッグタイマで**強制電源オフ**を設定した場合、「POWER ボタン」を長押ししたときと同じ処理になります。EC4A シリーズでは POWER ボタンを 4 秒間長押しすると、強制的に電源 OFF されます。また、Algonomix4 では、POWER ボタンを押されるとシャットダウン処理を実行します。

出荷時設定のままでは、シャットダウン処理中に電源が落ちることになり、m-SATA 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デバイスについて

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

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

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

IOCTL

ハードウェア・ウォッチドッグタイマデバイスを操作するには**ioctl**関数を使用します。

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

● **ctlcode**

RASHWWDT_IOCTSTART

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

RASHWWDT_IOCTSTOP

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

RASHWWDT_IOCSCONFIG

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

RASHWWDT_IOCGCONFIG

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

RASHWWDT_IOCQKEEPALIVE

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

RASHWWDT_IOCQWAITEVENT

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

RASHWWDT_IOCQWAKEUP

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

● **param**

RASHWWDT_CONFIG構造体

ハードウェア・ウォッチドッグタイマの設定を格納します。

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

４－４－９　　ハードウェア・ウォッチドッグタイマサンプル

●コンソール用サンプルプログラム

「/usr/local/tools-0010/samples/sampleConsole/sample_HwWdtKeepalive」に、ハードウェア・ウォッチドッグタイマドライバを使用するサンプルプログラムが入っています。リスト 4-4-9-1 にサンプルプログラムのソースコードを示します。

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

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

#define HWWDTDEV_FILENAME   "/dev/rashwwdt"

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

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

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

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

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

    fFinish = 0;
    while(1){
        if(!fStart){
            break;
        }
        /* KeepAlive */
        if(!fStop){
            ioctl(wdtfd, RASHWWDT_IOCQKEEPALIVE, NULL);
        }
        usleep(100 * 1000);
    }
    /*
     * ハードウェア・ウォッチドッグタイマ停止
     * ハードウェア・ウォッチドッグタイマ破棄
     */
    ioctl(wdtfd, RASHWWDT_IOCTSTOP, NULL);
    close(wdtfd);
    printf("TimerProc: Stop¥n");

    fFinish = 1;

    return 0;
}

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

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

    fStart = 1;
```

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

    return 0;
}

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

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

    while((c = getopt(argc, argv, "t:a:s:")) != -1){
        switch(c){
            case 't':
                wdt = atoi(optarg);
                break;
            case 'a':
                action = atoi(optarg);
                break;
            case 's':
                sta_type = atoi(optarg);
                break;
        }
    }
    if((wdt <= 0) || (action < 0) || (action > RASHWWDT_RESET) || (sta_type < 0) || (sta_type
> RASHWWDT_ENDCONTINUE)){
        printf("  t:test time[ *100msec]¥n");
        printf("  a:action 0:Shutdown 1:Reboot 2:Popup 3:Event 4:PowerOff 5:Reset ¥n");
        printf("  s:sta_type 0:endstop 1:continue¥n");
        return -1;
    }

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

    CreateThread();

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

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

    return 0;
}
```

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

「/usr/local/tools-0010/samples/sampleConsole/sample_HwWdtEventWait」に、タイムアウトのイベント通知を取得するサンプルプログラムが入っています。リスト 4-4-9-2 に、サンプルプログラムのソースコードを示します。

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

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

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

//-------------------------------------------------------------------------
```

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

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

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

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

//-----------------------------------------------------------------------------
static int CreateTimerEventInfo(int No, 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;
```

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

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

    /*
     * 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){
        close(info->wdtfd);
        printf("pthread_create: error¥n");
        return -1;
    }

    return 0;
}

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

    pthread_cancel(info->ThreadID);
}

//---------------------------------------------------------------------------
```

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

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

    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 バックアップ RAM 機能について

EC4A シリーズでは 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、『2-3-4 ASD UPS Config について』を参照してください。

４－４－１１ 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/g5sram0）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 4-4-11-1 にデバイスの詳細を示します。

表 4-4-11-1. 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/g5sram0)を**open**関数でオープンします。 rasramfd = open("/dev/g5sram0", 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

４－４－１２ 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 4-4-12-1 にソースコードを示します。

リスト 4-4-12-1. 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/g5sram0", 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 4-4-12-2 にソースコードを示します。

リスト 4-4-12-2. 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/g5sram0", 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;
}
```

４－５　　タイマ割込み機能

４－５－１　　タイマ割込み機能について

EC4A シリーズには、ハードウェアによるタイマ割込み機能が実装されています。タイマ割込み機能を使用することで、指定した時間で周期的に割込みを発生させることができます。タイマドライバによって生成されるタイマデバイスを操作することによって、ユーザアプリケーションからタイマ割込み機能を利用できるようになります。タイマデバイスでは、タイマ設定、タイマイベント通知などの機能を使用することができます。

４－５－２　　タイマ割込み機能デバイスについて

タイマドライバはタイマデバイスを生成します。ユーザアプリケーションは、デバイスファイルにアクセスすることによってタイマ機能を操作します。表 4-5-2-1 にタイマデバイスの詳細を示します。

表 4-5-2-1. タイマデバイスリファレンス

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

IOCTL

タイマデバイスを操作するには**ioctl**関数を使用します。

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

● **ctlcode**

RASTIME_IOCTSTART
タイマを開始します。
[param] なし (NULL)
[error] 0:正常、0以外:異常

RASTIME_IOCTSTOP
タイマを停止します。
[param] なし (NULL)
[error] 0:正常、0以外:異常

RASTIME_IOCSCONFIG
タイマを設定します。
[param] RASTIME_CONFIG型変数のポインタ
[error] 0:正常、0以外:異常

RASTIME_IOCGCONFIG
タイマ設定を取得します。
[param] RASTIME_CONFIG型変数のポインタ
[error] 0:正常、0以外:異常

RASTIME_IOCSTIMERRES
タイマ解像度を設定します。(設定値: 1～65535[msec]、初期値: 10[msec])
システムで使用されている全てのrastimeがこのタイマ解像度で動作します。
変更する場合は、他のrastimeが動作していないかどうか注意してください。
[param] unsigned long型変数のポインタ

RASTIME_IOCGTIMERRES
タイマ解像度を取得します。
[param] unsigned long型変数のポインタ
[error] 0:正常、0以外:異常

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

RASTIME_IOCQWAKEUP
タイマイベント待ちを強制解除します。
[param] なし (NULL)
[error] 0:正常、0以外:異常

● **param**

RASTIME_CONFIG構造体
タイマの設定を格納します。

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

４－５－３　　タイマ割込み機能サンプルプログラム

●コンソール用サンプルプログラム

「/usr/local/tools-0010/samples/sampleConsole/sample_Timer」に、タイマデバイスを制御するサンプルプログラムが入っています。リスト 4-5-3-1 にサンプルプログラムのソースコードを示します。タイマ解像度の変更、タイマ設定、タイマ開始/停止、タイマイベント待ちを行うサンプルです。このサンプルプログラムでは 10 個のタイマを同時に使用しています。

リスト 4-5-3-1. タイマデバイス操作ソースコード（main.c）

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

#define TIMERDEV_FILENAME   "/dev/rastime"
#define MAX_TIMEREVENT      10

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

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

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

    gettimeofday(&tv,NULL);
```

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

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

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

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

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

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

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

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

```
    info->TimerDesc = -1;

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

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

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

    return 0;
}

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

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

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

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

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

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

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

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

```
    if(timer_resolution == 10){
        close(timerfd);
        return 0;
    }
    timer_resolution = 10;
    if(ioctl(timerfd, RASTIME_IOCSTIMERRES, &timer_resolution) != 0){
        close(timerfd);
        printf("ioctl: RASTIME_IOCSTIMERRES: error¥n");
        return -1;
    }
    printf("SetTimerResolution: RASTIME_IOCSTIMERRES: timer_resolution=%ld¥n",
timer_resolution);

    close(timerfd);
    return 0;
}

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

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

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

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

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

    return 0;
}
```

４－６　　UPS 機能

４－６－１　UPS 機能について

EC4A 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、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。EC4A シリーズで使用できるセマフォを表 4-6-1-1 に示します。

表 4-6-1-1. 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、『2-3-4 ASD UPS Config について』を参照してください。

４－６－２　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 4-6-2-1 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 4-6-2-1. 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;
}
```

４－７　　その他のサンプルプログラム

４－７－１　　マルチランゲージ表示サンプルプログラム

●WideStudio 用サンプルプログラム

「/usr/local/tools-0010/samples/sampleWideStudio/sample_MultiLang」に、日英中韓の文字列を同時に表示するプログラムのサンプルプログラムのソースコードが入っています。多言語を同時に表示する為には、文字コードを UTF-8 にする必要があります。その為、サンプルプログラムのソースコードのファイルも UTF-8 形式でないと正常に読み出すことができないのでご注意ください。UTF-8 の文字コードで書かれた文字列を表示するようにして、対応するフォントを用意できれば、多言語プログラムが実現します。フォント設定の仕方については『3-3-9 WideStudio/MWT の開発例』を参照してください。

多言語表示のサンプルプログラムを実行した画面を図 4-7-1-1 に示します。

図 4-7-1-1. 多言語表示実行画面

Algonomix4 標準では、中国語と韓国語のフォントが実装されていないため、正しく表示できません。TrueType の中国語、韓国語のフォントを実装すれば改善されます。

４－７－２　　テンキーボードサンプル

●WideStudio 用サンプルプログラム

「/usr/local/tools-0010/samples/sampleWideStudio/sample_10Key」に、タッチパネルアプリケーションにおいて数字入力でよく使用されるテンキーキーボードを表示するサンプルプログラムのソースコードが入っています。数値入力をしたいボタンのプロシージャで図のようなテンキー画像を出力する関数を呼び出します。数値を入力して「OK」ボタンを押すことで、その数値が最初に起動したボタン上に表示されます。

テンキーボードのサンプルプログラムを実行した画面を図 4-7-2-1 に示します。

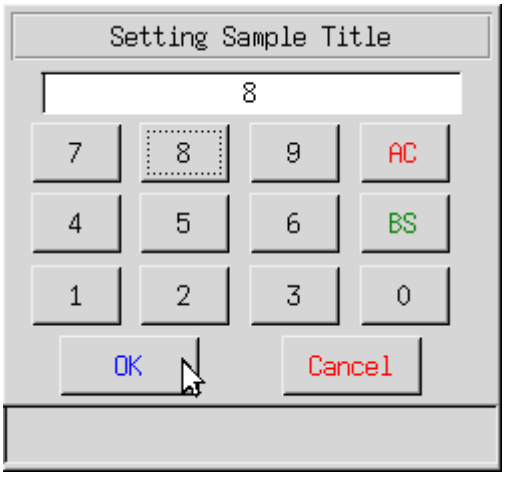


図 4-7-2-1. テンキー入力画面

４－７－３　　起動ランチャーサンプルプログラム

●WideStudio 用サンプルプログラム

「/usr/local/tools-0010/samples/sampleWideStudio/sample_Launcher」に、別のアプリケーションを起動するランチャープログラムのサンプルプログラムのソースコードが入っています。それぞれのボタンをクリックすることで、それぞれ対応したアプリケーションが起動します。

起動ランチャーのサンプルプログラムを実行した画面を図 4-7-3-1 に示します。

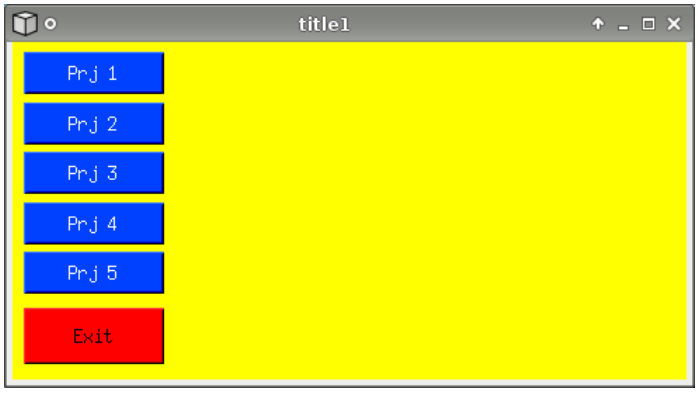


図 4-7-3-1. 起動ランチャー画面

ボタンクリックしたときのプロシージャイベント関数のソースコードをリスト 4-7-3-1 に示します。

リスト 4-7-3-1. 起動ランチャーボタンクリックイベント (Btn_Click.cpp)

```
#include <WScom.h>
#include <WSCfunctionList.h>
#include <WSCbase.h>
#include "newwin000.h"
//-------------------------------------------------------
//Function for the event procedure
//-------------------------------------------------------
void Btn_Click(WSCbase* object){
    if(object == btn_prg1){
        system("xeyes &");          /* xeyes ﾌﾟﾛｸﾞﾗﾑを起動 */
    }
    if(object == btn_prg2){
        system("xclock &");         /* xclock ﾌﾟﾛｸﾞﾗﾑを起動 */
    }
    if(object == btn_prg3){
        system("xcalc &");          /* xcalc ﾌﾟﾛｸﾞﾗﾑを起動 */
    }
    if(object == btn_prg4){
        system("xlogo &");          /* xlogo ﾌﾟﾛｸﾞﾗﾑを起動 */
    }
    if(object == btn_prg5){
        system("chromium browser &");       /* chromium browser ﾌﾟﾛｸﾞﾗﾑを起動 */
    }
    if(object == btn_exit){
        exit(0);                /* 終了 */
    }
}
static WSCfunctionRegister  op("Btn_Click",(void*)Btn_Click);
```

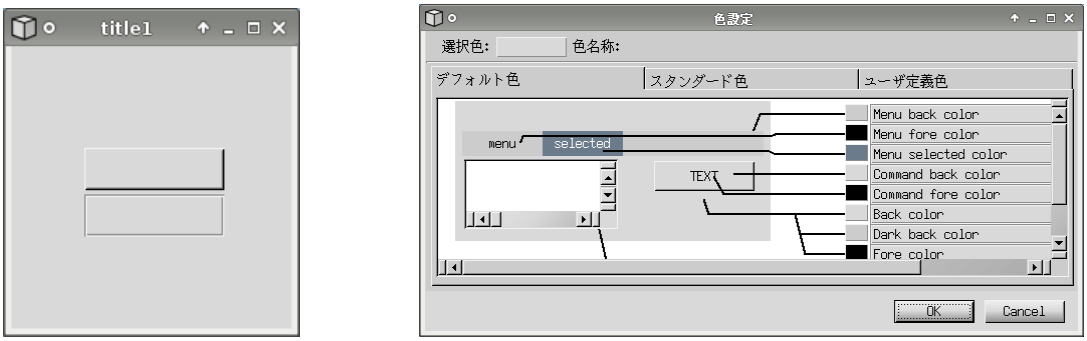
ボタンがクリックされると、object にボタンのポインタが渡されます。ここでは、押されたボタン毎に「system」という関数の引数で渡された文字列のコマンドを実行します。コマンドの後ろに「&」が記述されるとコマンドの終了まで待ちません。「&」を記述しないでコマンドを実行することで、そのコマンドが終了されるまで「system」関数から返りません。

４－７－４　　色選択サンプルプログラム

●WideStudio 用サンプルプログラム

「/usr/local/tools-0010/samples/sampleWideStudio/sample_ColorPalette」に、カラーパレットを表示するサンプルプログラムのソースコードが入っています。ボタンを押すと、ボタンのプロシージャでカラーパレットが呼び出されます。カラーパレットで色を選択し、「OK」ボタンを押すと、ボタンの背景色が選択した色になり、色名称がラベルに表示されます。

色選択サンプルプログラムを実行した画面を図 4-7-4-1 に示します。



（起動画面）　　（カラーパレット）

図 4-7-4-1. 色選択サンプルプログラム

４－７－５　　日本語入力キーボードサンプルプログラム

●WideStudio 用サンプルプログラム

「/usr/local/tools-0010/samples/sampleWideStudio/sample_JapaneseInsert」に、日本語入力可能なソフトウェアキーボードのサンプルプログラムのソースコードが入っています。このプログラムを実行すると、図 4-7-5-1 のようにボタンとインプットフィールドが表示されます。これらをクリックすると、図 4-7-5-2 のようなソフトウェアキーボードを呼び出され、入力した文字が表示されます。

図 4-7-5-2 のソフトウェアキーボードにおいて、「E/J」ボタンを押すと英語入力と日本語入力が切り替わります。また、入力完了後「OK」ボタンを押すと、元のウィンドウに戻り、入力した文字が表示されます。

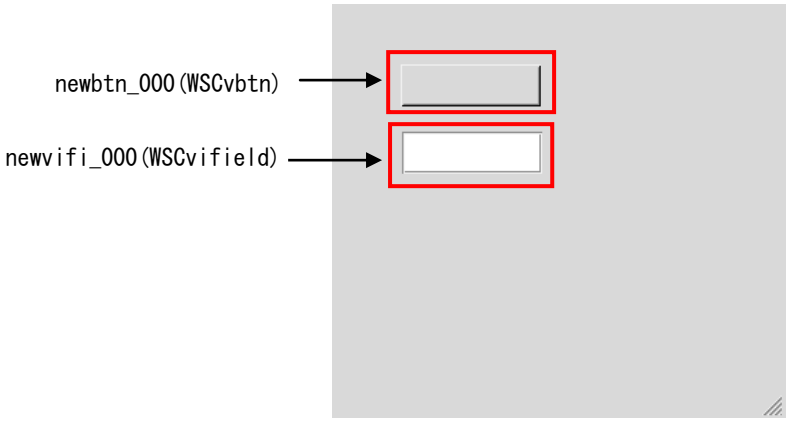


図 4-7-5-1. 日本語入力サンプル画面

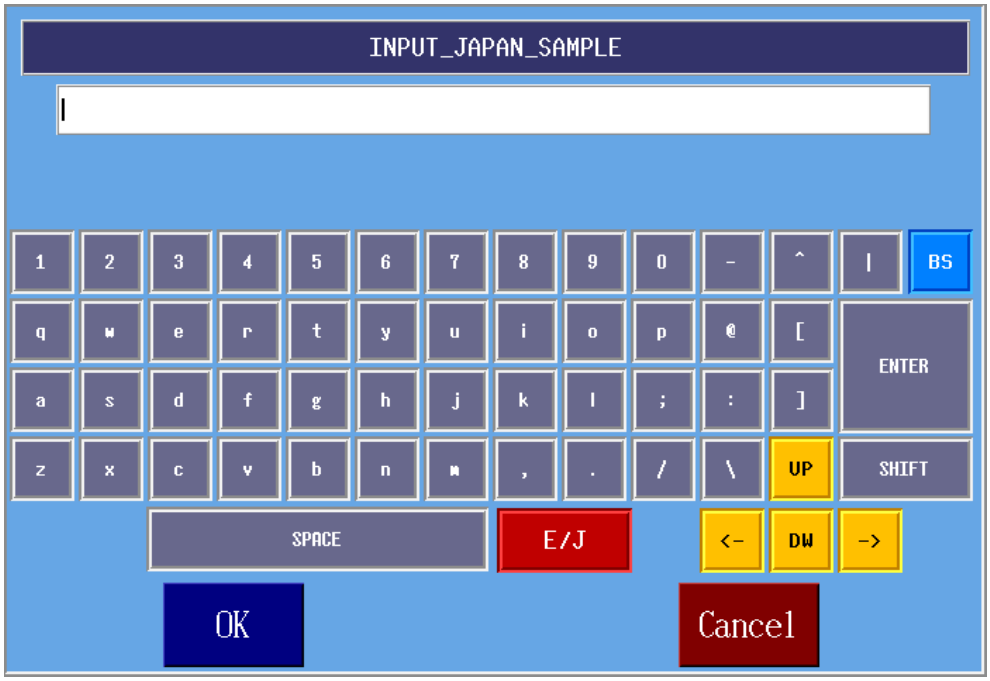


図 4-7-5-2.　ソフトウェアキーボード画面

ソフトウェアキーボードをプロジェクトに組込む方法について説明します。

① /usr/local/tools-0010/samples/sampleWideStudio/sample_JapaneseInsert」フォルダの中にある、以下のファイルを組込みたいプロジェクトのフォルダにコピーします。
　　str_set.cpp
　　str_set.h
　　strsetbtn_Click.cpp
　　strsetbtn_Press.cpp
　　win_strset.cpp
　　win_strset.h
　　win_strset.win
② 組込みたいプロジェクトの wns ファイルに、win_strset.win を追加します。
③ WideStudio を起動し、プロジェクトを開きます。
④ 「プロジェクト」→「プロジェクト設定」で追加タブを選択し、追加ソースコードの「追加」ボタンを押して「str_set.cpp」を選択します。
⑤ 日本語入力したいオブジェクトのプロシージャで「str_set.h」をインクルードし、「strset_Disp」関数を使ってソフトウェアキーボードを呼び出します。

ソフトウェアキーボードを呼び出すサンプルプログラムのソースコードをリスト 4-7-5-1 に示します。
リスト 4-7-5-1 において、「newvifi_OOO」は MOUSE-PRESS イベント、「newvifi_OOO」は ACTIVE イベントでソフトウェアキーボードを呼び出しています。
strset_Disp 関数の第 1 引数に呼び出し元のオブジェクト、第 2 引数にソフトウェアキーボードのタイトルを指定します。また、第 3 引数にはソフトウェアキーボードを呼び出したときに表示される初期入力文字列を指定します。リスト 4-7-5-1 では、呼び出し元の表示文字列を初期文字列として扱っています。初期文字列を空にしたい場合は NULL を指定します。

リスト 4-7-5-1. ソフトウェアキーボード呼び出し (Btn_Click.cpp)

```
#include <WScom.h>
#include <WSCfunctionList.h>
#include <WSCbase.h>

#include "newwin000.h"
#include "str_set.h"
//-----------------------------------------------------------
//Function for the event procedure
//-----------------------------------------------------------
void Btn_Click(WSCbase* object){
    if(object == newvifi_000){
        strset_Disp(newvifi_000,"INPUT_JAPAN_SAMPLE",(char*)newvifi_000->getProperty(WSNlabelStrin
g));
    }
    if(object == newvbtn_000){

    strset_Disp(newvbtn_000,"INPUT_JAPAN_SAMPLE",(char*)newvbtn_000->getProperty(WSNlabelString));
    }
}
static WSCfunctionRegister  op("Btn_Click",(void*)Btn_Click);
```

⑥　　コンパイルして完了です。

このソフトウェアキーボードは、サンプルプログラムとして作成されていますので、ボタンやウィンドウの色や大きさ、配置については自由に変更してください。

※注：このソフトウェアキーボードプログラムは、専用の仮想キーボードドライバを使用しています。一般の Linux システム上では動作しませんのでご注意ください。

４－８　　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。

４－８－１　　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。
Algonomix4 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/sdc1 /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『4-8-2 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。

４－８－２　　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 4-8-2-1 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。

EC4A シリーズでの認識優先度を表 4-8-2-1 に示します。

表 4-8-2-1. ストレージデバイス認識優先度

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

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

例 1:mSATA メインストレージのみ挿入して OS 起動したとき

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

表 4-8-2-2. デバイス名割り振り

スロット名	デバイス名
m-SATA メインストレージ	/dev/sda
USB スロット 0～3	/dev/sdb、/dev/sdc･･･

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

mSATA メインストレージが sda として、m-SATA サブストレージが sdb として認識されます。その後挿入された USB メモリは sdc、sdd･･･として認識されます。

表 4-8-2-3. デバイス名割り振り

スロット名	デバイス名
m-SATA メインストレージ	/dev/sda
m-SATA サブストレージ	/dev/sdb
USB スロット 0～3	/dev/sdc、/dev/sdd･･･

４－８－３　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、SD カードを/home/asdusr/sd 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 4-8-3-1. SD カードの起動時マウント設定例 (/etc/fstab)

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

tmpfs   /tmp        tmpfs   defaults,size=192m  0   0
tmpfs   /var/log    tmpfs   defaults,size=64m   0   0
/dev/mmcblk0p1  /home/asdusr/sd vfat    defaults,nofail 0   0      この行を追加
```

第５章 システムリカバリ

本章では、「EC4A 用 Algonomix4 32 ビット版 リカバリ DVD」を使用したシステムのリカバリとバックアップについて説明します。

５－１　　リカバリ DVD について

EC4A シリーズ本体は、システムのリカバリを行うことができます。リカバリで行える処理は以下のとおりです。

- 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リカバリ DVD 以外に以下のものを用意する必要があります。

- 4GByte 以上の USB メモリ（リカバリ USB 用）
- 8GByte 程度の空き容量がある USB 接続可能なストレージメディア（USB メモリなど）
  （バックアップイメージ保存用）

リカバリ DVD を実行する場合、BIOS 設定が必要となります。また、リカバリ DVD での作業終了後は BIOS の設定を元に戻す必要があります。

- リカバリ実行の流れ

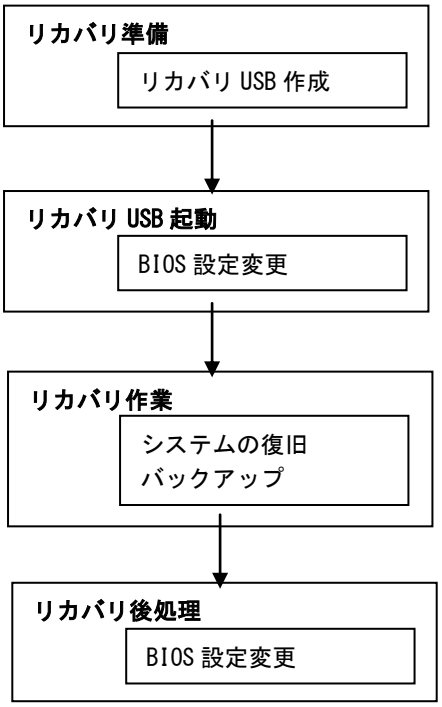


図 5-1-1. リカバリ DVD 使用の流れ

５－１－１　リカバリ準備

リカバリを実行する前に、リカバリシステムを起動するための USB メモリを作成します。
4GByte 以上のサイズの USB メモリをあらかじめ用意してください。

※注：　ここで用意した USB メモリ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。

● リカバリ USB 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 USB メモリを、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実行します。
[リカバリ DVD]¥Recivert¥RecoveryUSBImage.exe

④ 圧縮ファイルの解凍が始まります。
PC 上の任意の場所に解凍してください。
解凍が完了するまでお待ちください。
解凍が完了すると「RecoveryUSBImage.ddi」というファイルが展開されます。
⑤ リカバリ DVD の以下のファイルを実行します。
[リカバリ DVD]¥Recovery¥DDwin¥DDwin.exe
※注：　WindowsVista 以降の OS をご使用の場合は管理者権限で起動する必要があります。

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

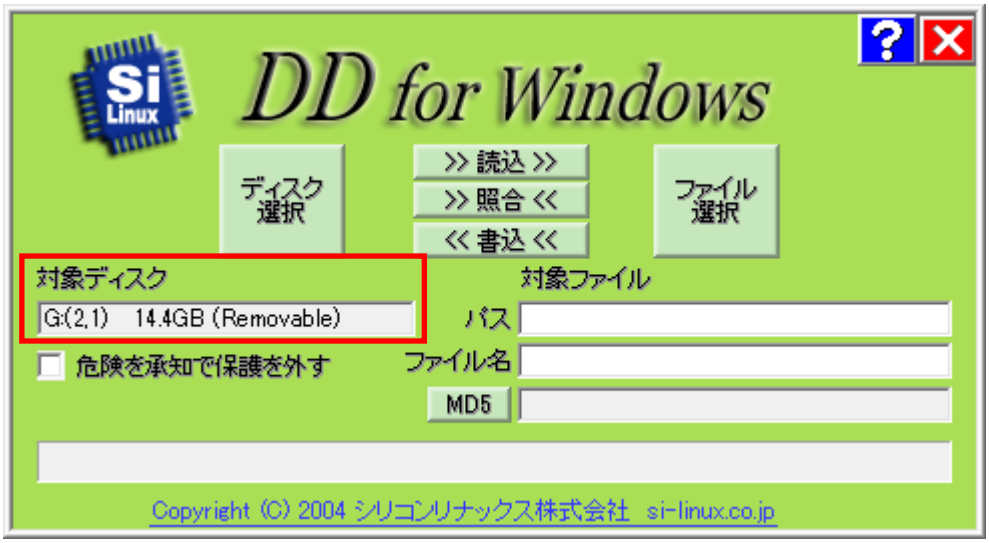


図 5-1-1-1. DD for Windows

⑧　「ファイル選択」ボタンを押してください。
ファイル選択画面が開くので、手順④で解凍した「RecoveryUSBImage.ddi」を選択してください。

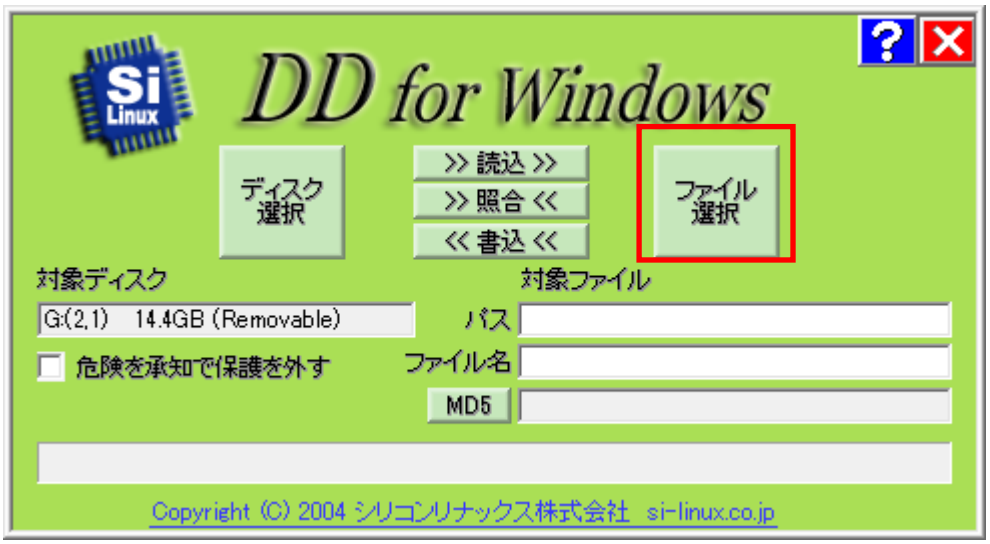


図 5-1-1-2.　ファイル選択

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

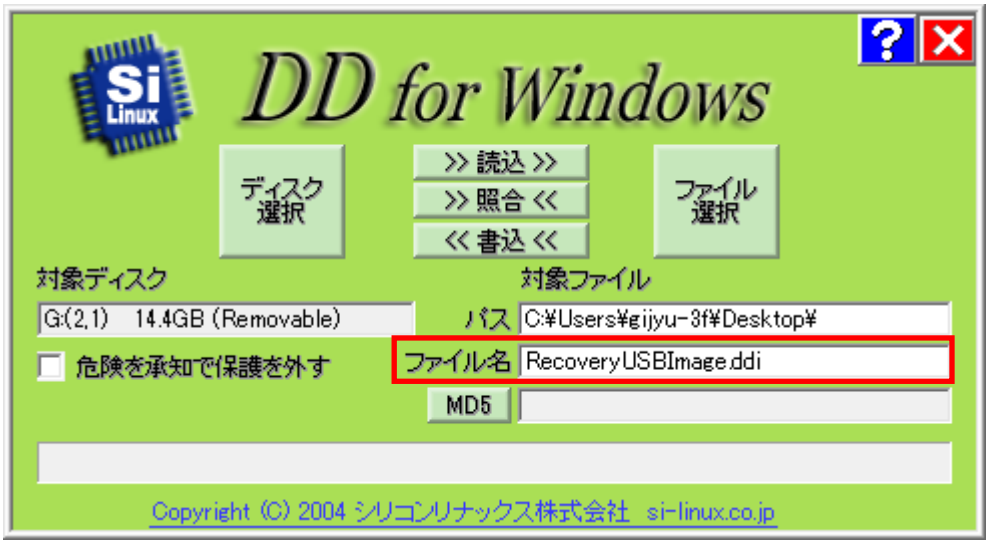


図 5-1-1-3.　対象ファイル

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

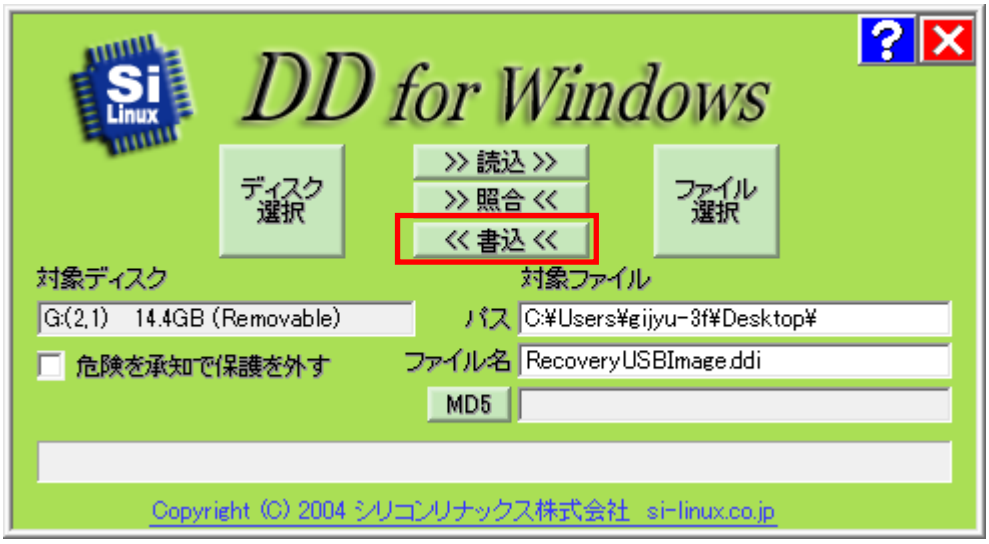


図 5-1-1-4.　書き込み開始

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

５－１－２　リカバリ USB 起動

リカバリ USB を起動させる前に、本体に接続されている LAN ケーブル、ストレージ（USB メモリ、SD カードなど）を取り外してください。サブストレージ（mSATA2）を接続している場合は、サブストレージを取り外してください。

リカバリ USB の起動にはリカバリ USB の他に以下のものを用意する必要があります。

- USB キーボード
- USB マウス
- 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)

● リカバリ USB 起動手順

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 リカバリ DVD を EC4A シリーズ本体に接続します。
③ USB キーボード、USB マウスを接続します。
④ 電源を入れます。BIOS 起動画面が表示されたところで[F2]キーを押し、BIOS 設定画面を表示させます。
⑤ BIOS 設定画面が表示されたら、[Boot]メニューを選択します。(図 5-1-2-1)
⑥ [USB CD](接続した USB-DVD ドライブ)を[ATA HDD0](m-SATA メインストレージ)よりも上に設定します。

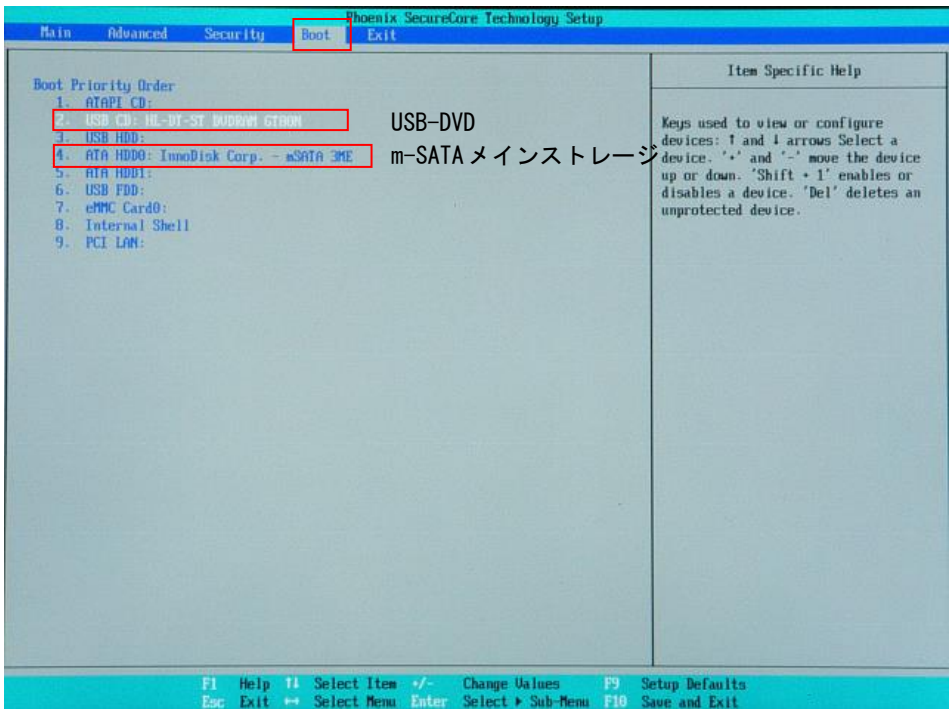


図 5-1-2-1. Boot デバイスの選択

⑦　[Exit]メニューを選択します。
⑧　[Exit Saving Changes]を実行し、設定を保存して終了します。

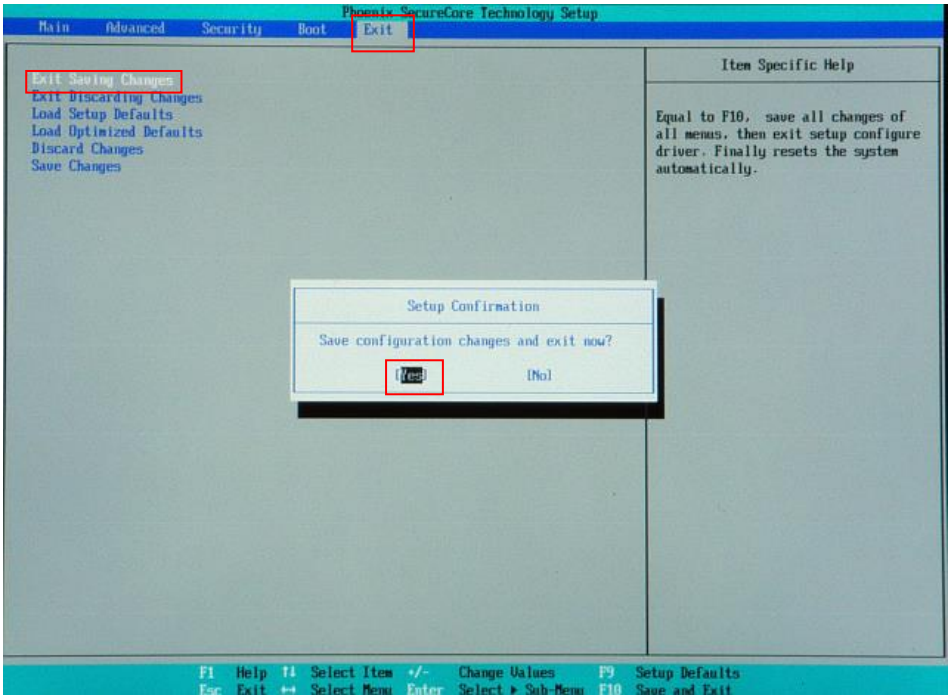


図 5-1-2-2. BIOS 設定 保存と終了

⑨　　再起動し、正常にリカバリ DVD から起動すると図 5-1-2-3 のリカバリメイン画面が表示されます。

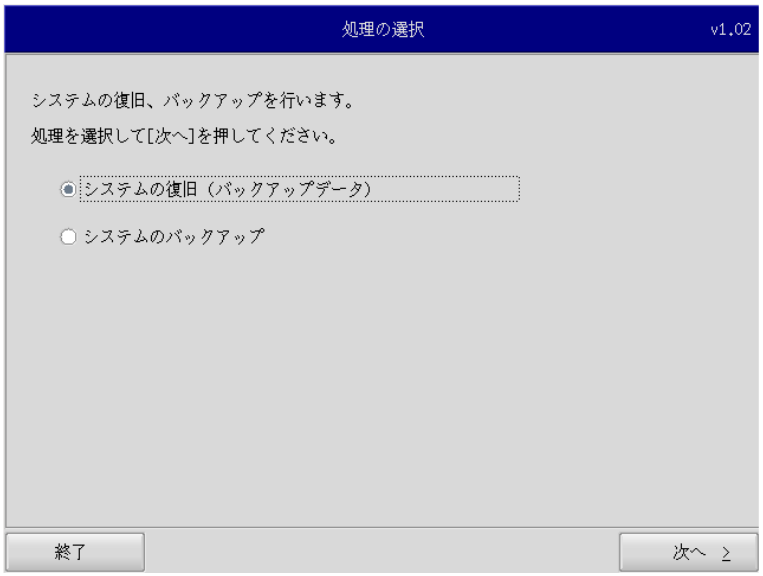


図 5-1-2-3. リカバリメイン画面

５－１－３　　リカバリ作業

リカバリメイン画面から処理を選んでリカバリ作業を行います。

- システムの復旧（バックアップデータ）
- システムのバックアップ

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

５－２　　システムの復旧（バックアップデータ）

工場出荷イメージをメインストレージ（mSATA1）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（mSATA1）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、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]¥Recovery¥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 「**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。**」を参考にリカバリ USB を起動させます。

④ リカバリメイン画面（図 5-1-1-7）で[システムの復旧（バックアップデータ）]を選択し、[次へ]ボタンを押します。

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

図 5-2-1. メディア選択画面

⑥　メディアとパーティション選択画面（図 5-2-2）が表示されます。あらかじめ用意しておいた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。

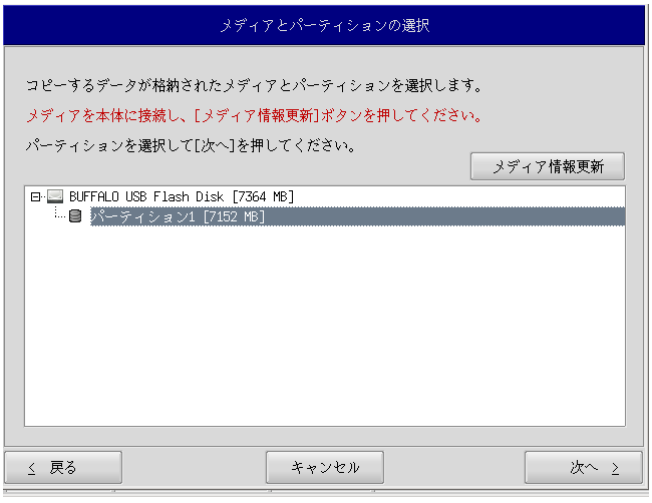


図 5-2-2. メディアとパーティション選択画面

⑦　フォルダ選択画面（図 5-2-3）が表示されます。［参照］ボタンを押します。

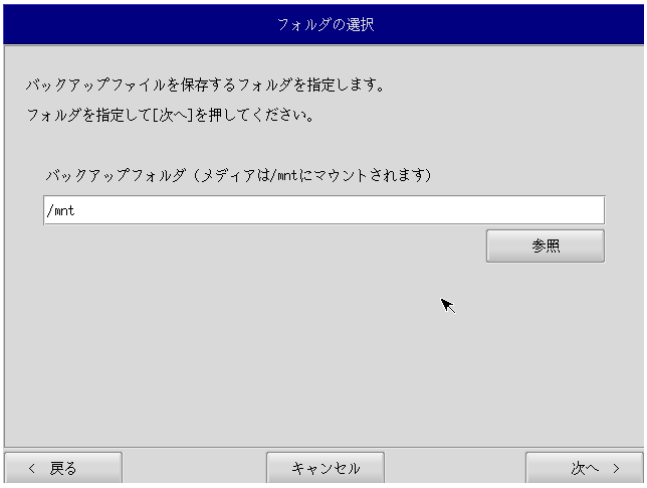


図 5-2-3.　フォルダ選択画面

⑧　ファイル参照画面（図 5-2-4）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。

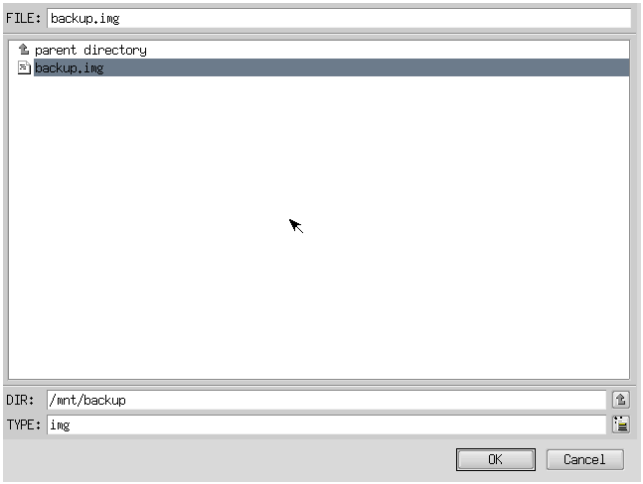


図 5-2-4.　ファイル参照画面

⑨　ファイル参照画面（図 5-2-4）で指定したバックアップファイルが入力されていることを確認します。[次へ]ボタンを押します。

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

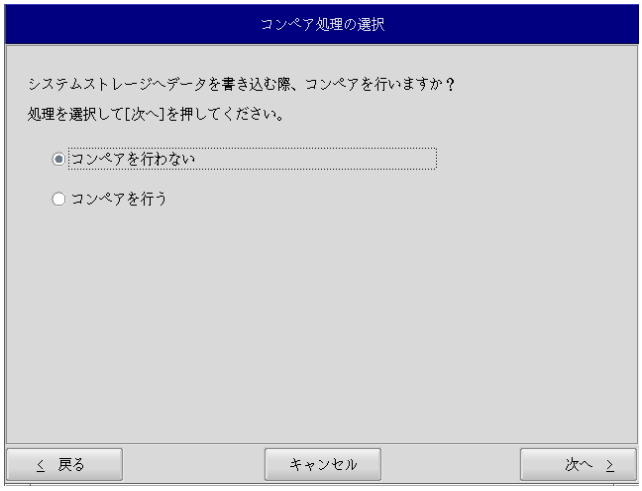


図 5-2-5. コンペア処理選択画面

⑪　確認画面(図 5-2-6) が表示されます。メディア、パーティション、バックアップファイルを確認します。[次へ]ボタンを押します。

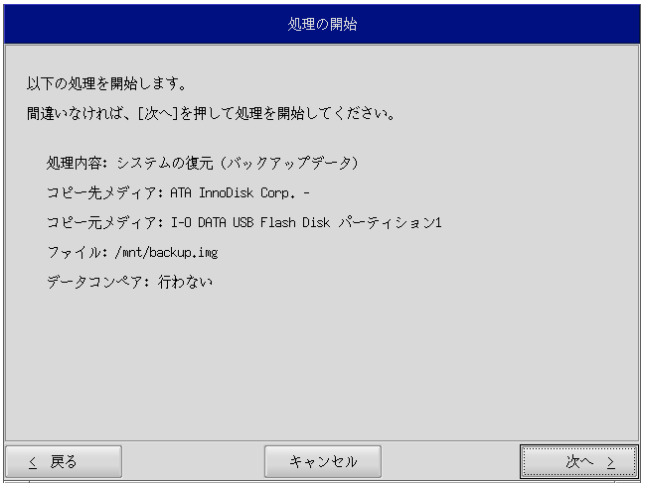


図 5-2-6. 確認画面

⑫　実行中画面（図 5-2-7）が表示され、処理が開始されます。実行中はリカバリ USB メモリ、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。

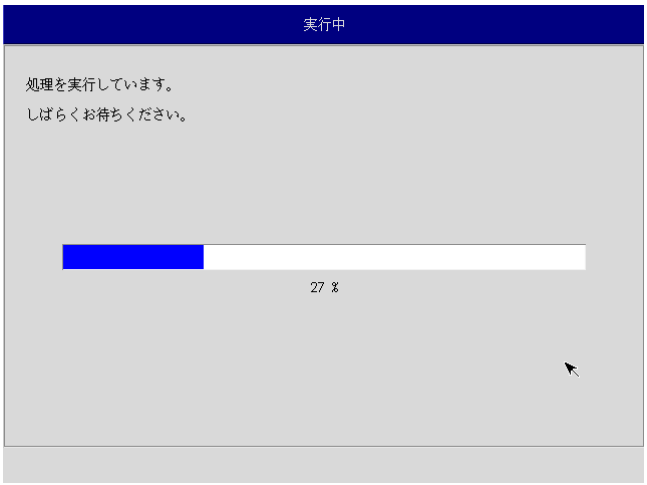


図 5-2-7. 実行中画面

⑬　終了画面（図 5-2-8）が表示されるとバックアップファイルの書き込みは完了です。[終了]ボタンを押して電源を落とし、リカバリ USB メモリ、保存メディアを外します。

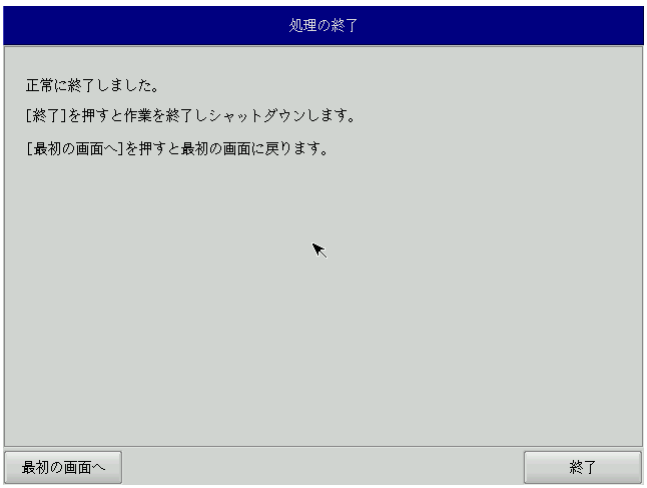


図 5-2-8. 終了画面

⑭　電源を入れ、BIOS 起動画面が表示されたところで[F2]キーを押し、BIOS 設定画面を表示させます。
⑮　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（mSATA1）の状態をファイルに保存します。バックアップファイル保存のために外部メモリとして、USB メモリ、SD カードなどが必要となります。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 「**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。**」を参考にリカバリ USB を起動させます。
④ リカバリメイン画面（図 5-1-1-7）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。コピー先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。

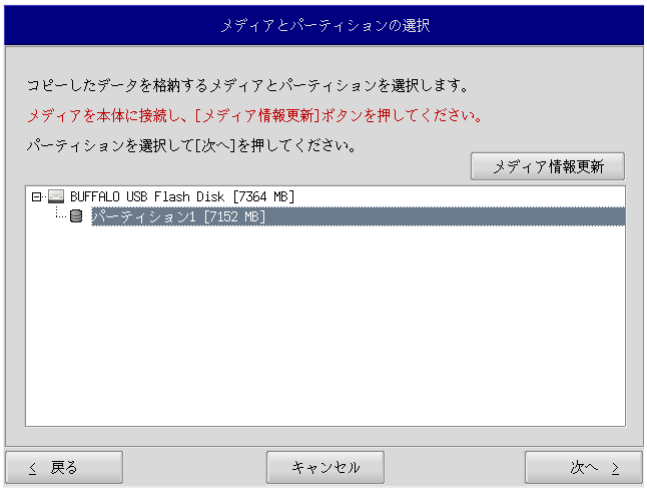


図 5-3-2. メディアとパーティション選択画面

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。

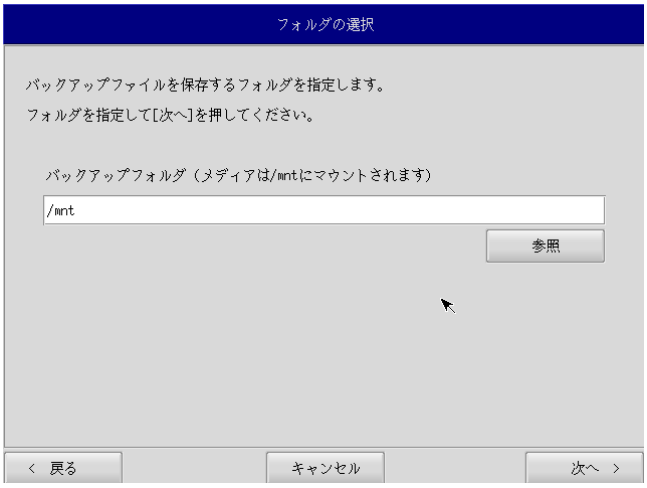


図 5-3-3. フォルダ選択画面

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。

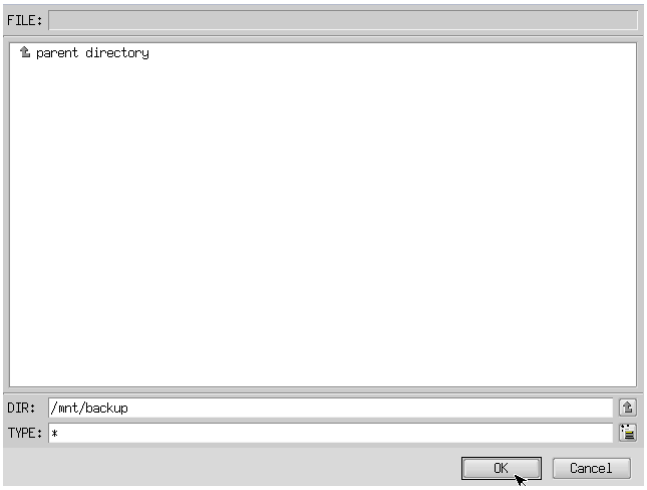


図 5-3-4.　フォルダ参照画面

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。

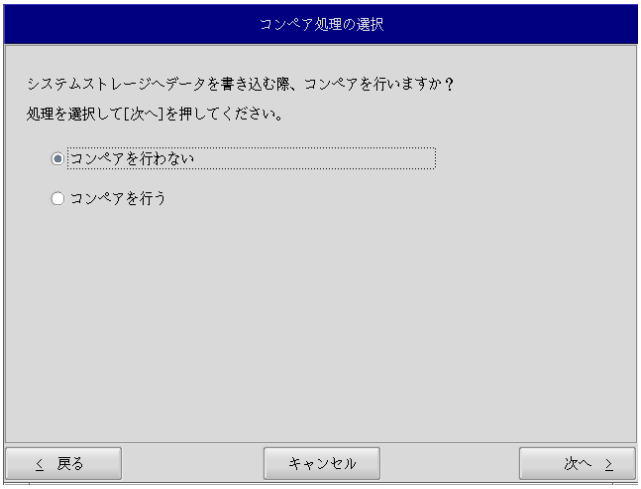


図 5-3-5.　コンペア処理選択画面

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。

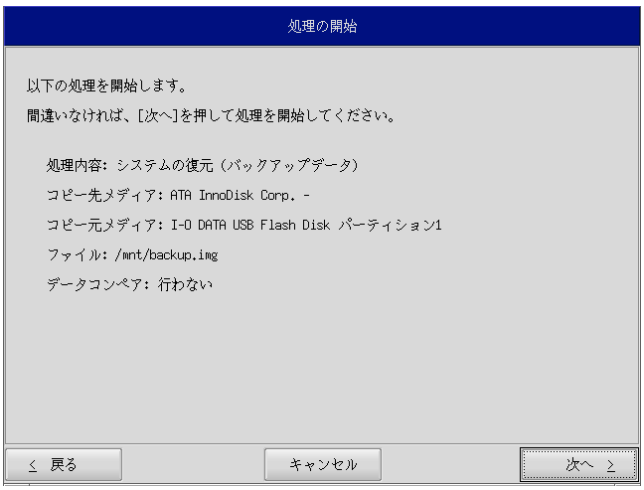


図 5-3-6.　確認画面

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。

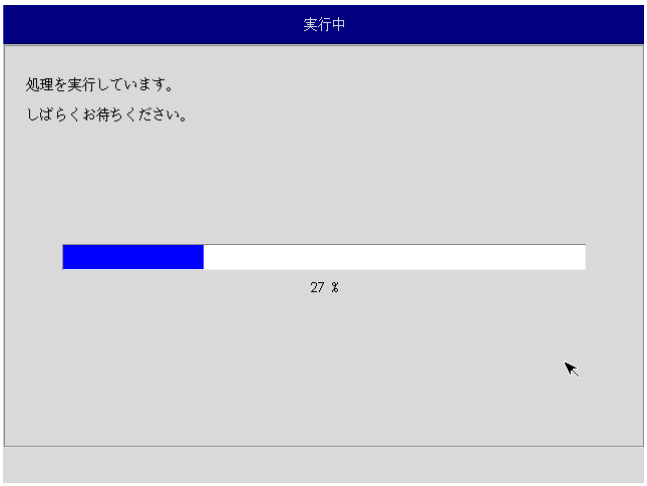


図 5-3-7.　実行中画面

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。

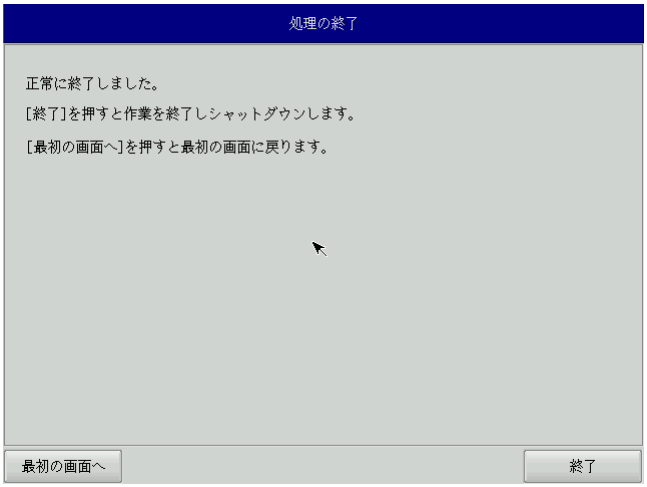


図 5-3-8.　終了画面

⑭　電源を入れ、BIOS 起動画面が表示されたところで[F2]キーを押し、BIOS 設定画面を表示させます。
⑮　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。

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。

７７Ｇ０３００９０Ｂ　２０１８年　３月　第２版
７７Ｇ０３００９０Ａ　２０１７年　６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