Algo Smart Panel シリーズ

マニュアル

Algo Smart Panel 用

『PMC T-Kernel』について

AP-2000

AP-3100

AP-3102

目　次

第３章　Algo Smart Panelについて

第４章　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頂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、下記の方法について説明します。
・Algo Smart Panel 用 PMC T-Kernel の使用方法
・Algo Smart Panel 用 PMC T-Kernel 開発環境の構築手順と使用方法

PMC T-Kernel や Linux についての詳細は省略させていただきます。PMC T-Kernel および Linux に関する資料および文献と併せて本書をお読みください。

第１章　PMC T-Kernelについて

ここではT-Engineフォーラムが定めるT-EngineやT-Kernelについての基本的な仕様と、OSとしてのPMC T-Kernelの位置づけについて説明します。

１－１　　T-Engineとは

もともと、T-Engineというのは、T-KernlやT-Kernel Standard Extensionを動作させるためのプラットフォームとして規定されたハードウェアボードのことです。現状では標準T-Engineボードとμ T-Engineボードがあり、それぞれ、ハードウェア仕様を定め、ボードに載せるデバイスの種類と基板の寸法などを含めて標準化しています。

しかし、ハードウェアの仕様を決められた場合、それを一般的に量産しようとすると、性能、大きさ、機能、コスト等で合わない部分が出てきます。そのため、T-Engine 以外の組み込み機器に T-Kernel や T-Kernel Standard Extensionを搭載することができませんでした。

そこで、T-Engine Appliance(T-Engine応用製品)という規定があります。これは、T-KernelやT-Kernel Standard Extensionを搭載した組み込み機器のことをT-Engine Applianceと呼び、OSとしてT-Kernelを使用しているという共通点はありますが、ハードウェア構成の制約は一切ありません。

T-Kernelを搭載したAlgo Smart PanelもT-Engine Applianceということになります。

表 1-1-1. T-Engineプロジェクト標準化範囲

		T-Engine プロジェクト	T-Engine アプライアンス
CPU		規定無し（32bit）	規定無し（32bit）
ボードの物理形状		**標準化**	**自由**
ボードのハードウェア	（仕様）	**標準化**	**自由**
	（実装）	自由	自由
BIOS相当機能（T-Monitor）	（仕様）	標準化	標準化
	（実装）	自由	自由
リアルタイムOS基本機能（T-Kernel/OS）	（仕様）	標準化	標準化
	（実装）	一本化（シングルソース）	一本化（シングルソース）
システム管理機能（T-Kernel/SM）	（仕様）	標準化	標準化
	（実装）	一本化（シングルソース）	一本化（シングルソース）
開発環境	（仕様）	標準化	標準化
	（実装）	自由	自由
基本ミドルウェア（T-Kernel Standard Extension）		標準化	標準化

１－２　　T-Engineのソフトウェア構成

T-Engine のソフトウェア構成を図 1-2-1 に示します。

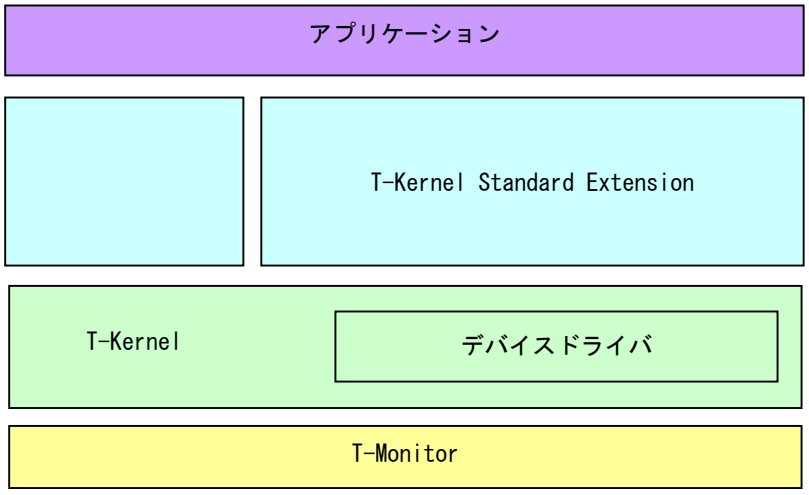


図 1-2-1. T-Engineソフトウェア構成

T-Engine のソフトウェア構成は図 1-2-1 のような階層構造になっており、各階層について説明していきます。

１－２－１　T-Monitor

T-Monitor とは T-Kernel を起動する前のブートローダ的な役割を持っています。T-Monitor の主な機能は以下の通りです。

１．システム機能
- ①　ハードウェアの初期化
- ②　システムの起動
- ③　例外、割り込み、トラップ処理機能

２．デバッグ機能
- ①　モリ操作
- ②　レジスタ操作
- ③　I/O 操作
- ④　逆アセンブル
- ⑤　プログラム／データロード
- ⑥　プログラム実行
- ⑦　ブレークポイント操作
- ⑧　トレース実行
- ⑨　ディスク読み込み／書き込み／ブート

３．プログラムサポート機能
- ①　モニタサービス関数

１－２－２　T-Kernel

T-Engine システムの中心となるリアルタイム OS が T-Kernel です。
T-Kernel は機能的に表 1-2-2-1 に示す機能に分類されます。

表 1-2-2-1. T-Kernel機能

分類	機能
T-Kernel/OS (Operating System)	タスク管理機能 同期・通信機能 メモリ管理機能 例外/割り込み制御機能 時間管理機能 サブシステム管理機能
T-Kernel/SM (System Manager)	システムメモリ管理機能 アドレス空間管理機能 デバイス管理機能 割り込み管理機能 I/O ポートアクセスサポート機能 省電力機能 システム構成情報管理機能
T-Kernel/DS (Debugger Support)	カーネル内部状態参照機能 実行トレース機能

１－２－３　デバイスドライバとサブシステム

T-Kernel ベースのプログラムとして、デバイスドライバとサブシステムがあります。
デバイスドライバはハードウェア／デバイスにアクセスするためのモジュールで、T-Kernel 仕様書のデバイスドライバ仕様に沿って作られたものをいいます。
サブシステムは OS に機能追加して新しいシステムコールを使えるようにするためのモジュールで、T-Kernel 仕様書のサブシステムの仕様に沿って作られたものをいいます。

デバイスドライバとサブシステムの共通点は以下の通りです。

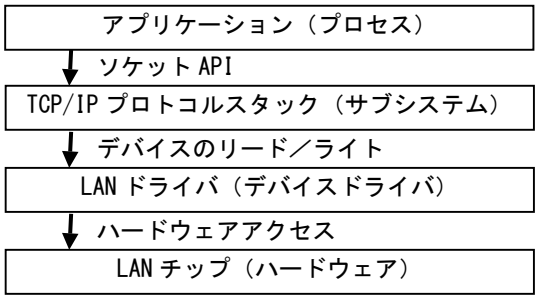
① T-Kernel 上で動作する
② 論理アドレスと物理アドレス両方アクセス可能
③ カーネルと同じシステム共有空間を使用する

この 2 つの仕組みは、物理アドレスを直接制御することができるため、ハードウェアに自由にアクセスできます。

デバイスドライバとサブシステムの相違点は以下の通りです。
デバイスドライバはリード／ライト型のインターフェースで呼び出せます。
サブシステムはサブシステムで任意に用意したシステムコールを呼び出すことでリード／ライトの枠にあてはまらないような、一般的なミドルウェアの構築を実現することできます。

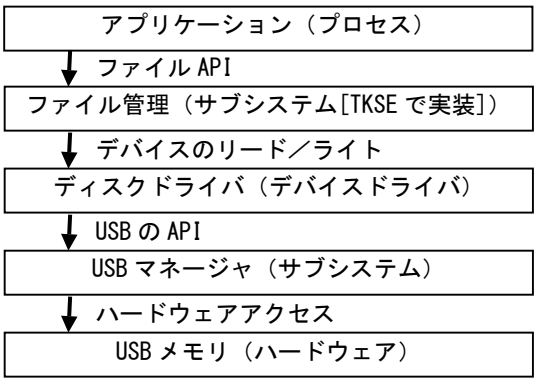
デバイスドライバとサブシステムを使用した実例を以下に示します。

●ネットワーク（TCP/IP）



TCP/IP のプロトコルスタックで標準的なソケット API を実装する場合は、単純なリードライトの枠組みとは異なるので、サブシステムとして組み込むようにします。LAN チップのアクセスをデバイスドライバ化しておけば、LAN チップが変更された場合、デバイスドライバの部分だけを交換すればよくなります。

●USB メモリ



ファイル管理そのものは、ファイル管理のサブシステムとして T-Kernel Standard Extention で実装されています。fopen 等のファイルアクセス関数をアプリケーションから使用した場合、ファイル管理のシステムコールが呼ばれ、ファイル管理からさらにディスクのデバイスドライバを呼び出す形になっています。このとき、ディスクドライバは直接 USB メモリにアクセスしてもいいのですが、USB マネージャというサブシステムを間にいれることで、USB の接続・切断や転送（バルク・インタラプト）等を統一的に扱うことができます。

サブシステムとデバイスドライバの役割分担により、いろいろな階層構造が可能です。

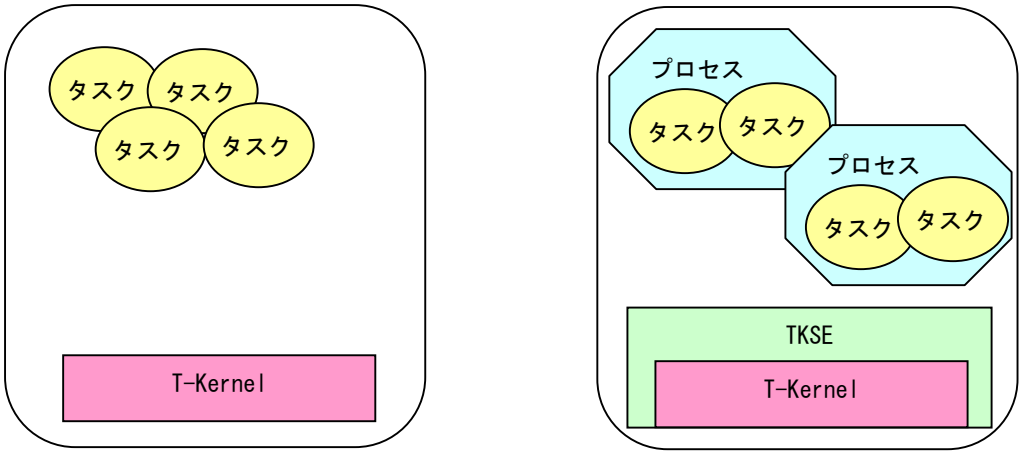
１－２－４　T-Kernel Standard Extension

T-Kernel Standard Extension（以降 TKSE）は T-Kernel 上で動作する拡張プログラムのことで、プロセスベースのプログラミングモデルを実現するものです。

TKSE でできることを以下に示します。

① プロセスプログラミング
② プロセス単位のメモリ保護・管理
③ ファイルシステムの標準サポート
④ 仮想記憶の対応
⑤ T-Kernel のリアルタイム性能を継承

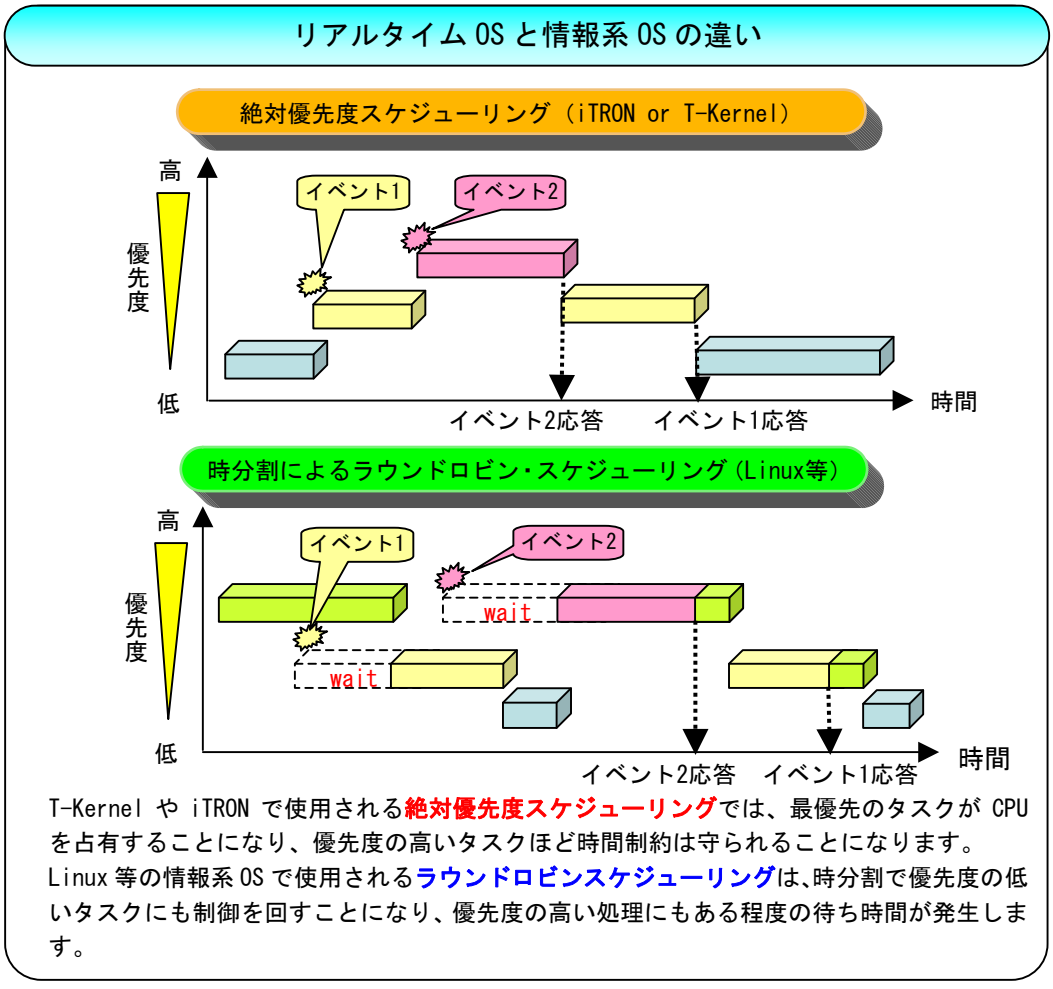
プロセスベースのプログラミングモデルを使用することで、あるプロセスがエラーとなっても、T-Kernel 本体や別のプロセスに影響を与えることはありません。タスクベースでは、1 つのタスクがエラーとなった場合、T-Kernel 全体が停止してしまいます。



(a)タスクベースアプリケーション　　(b)プロセスアプリケーション

図 1-2-4-1 T−Kernelプログラミングモデル

Linux と TKSE での大きな違いは⑤番のリアルタイム性能の継承にあります。TKSE では独自のタスク・スケジューリングアルゴリズムにより、iTRON の優先度順スケジューリングと Linux のラウンドロビンスケジューリングを混在させることができます。



T-Kernel や iTRON で使用される**絶対優先度スケジューリング**では、最優先のタスクが CPU を占有することになり、優先度の高いタスクほど時間制約は守られることになります。
Linux 等の情報系 OS で使用される**ラウンドロビンスケジューリング**は、時分割で優先度の低いタスクにも制御を回すことになり、優先度の高い処理にもある程度の待ち時間が発生します。

図 1-2-4-2. TKSEのスケジューリング方法

優先度を 255 段階で設定でき、3 つのグループに分けられます。
0～127 までを絶対優先度グループ、128～191 をラウンドロビングループ 1、さらに 192～255 をラウンドロビングループとよびます。各タスクはその優先度に従ってスケジューリングが行われます。

A. 絶対優先度グループ（優先度: 0～127）
　厳密に優先度順にスケジューリングされ(0 が最高優先度)、同一優先度の場合は、ラウンドロビン方式で平等にスケジューリングされます。
B. ラウンドロビングループ1（優先度: 128～191）
　このグループ内全体でラウンドロビンスケジューリングが行なわれ、優先度は相対的なスケジューリングの頻度を示します(128 が最高優先度）。従って、優先度が低いタスクでも必ず実行されることが保証されます。
C. ラウンドロビングループ2（優先度: 192～255）
　このグループ内全体でラウンドロビンスケジューリングが行なわれ、優先度は相対的なスケジューリングの頻度を示します(192 が最高優先度)。従って、優先度が低いタスクでも必ず実行されることが保証されます。

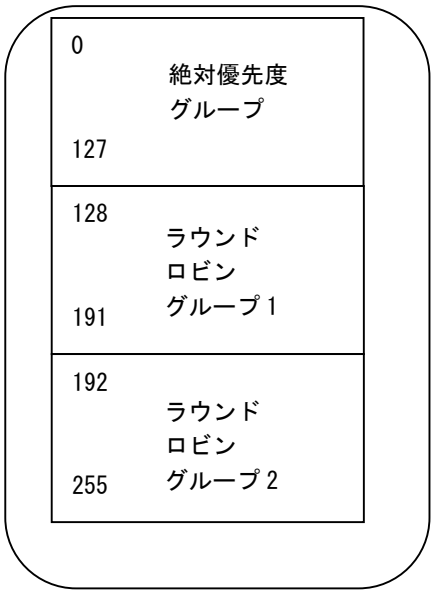


図 1-2-4-3. TKSEタスク優先度グループ

実際のスケジューリングは、全体として以下のように行なわれます。
1. 絶対優先度グループに属する実行可能状態のタスクがあれば､その中の最高優先度のタスクを実行状態とし、実行します。なければ、2. へ進みます。
2. ラウンドロビングループ 1 に属する実行可能状態のタスクがあれば、その中の相対優先度に従って、選択されたタスクを実行状態とし、実行します(最高優先度とは限らない)。 なければ、2.へ進みます。
3. ラウンドロビングループ 2 に属する実行可能状態のタスクがあれば、その中の相対優先度に従って、選択されたタスクを実行状態とし、実行します(最高優先度とは限らない)。なければ、スケジューリングを始めからやり直します。

一度生成されたプロセス / タスクの優先度の変更は、同一グループ内でのみ可能であり、グループが異なる優先度への変更はできません。

１－２－５　アプリケーション

アプリケーションはTKSE上で動作するプロセスプログラムのことで、TKSEやT-Kernelおよびサブシステムやデバイスドライバのシステムコールを使用して作成されます。
ユーザ独自のプログラムを組むこともあれば、ある特定の機能を提供するプログラムもあります。

表1-2-5-1. 主要なアプリケーション一覧

名前	内容
FTP	FTPプロトコルによるファイル転送機能
TELNET	ネットワークコンソール機能

１－３　　PMC T-Kernelについて

今回、Algo Smart Panel 用に T-Kernel を移植するにあたり、パーソナルメディアが提供している PMC T-Kernel を採用しました。Algo Smart Panel 用に提供されるソフトウェア構成図を図 1-3-1 に示します。

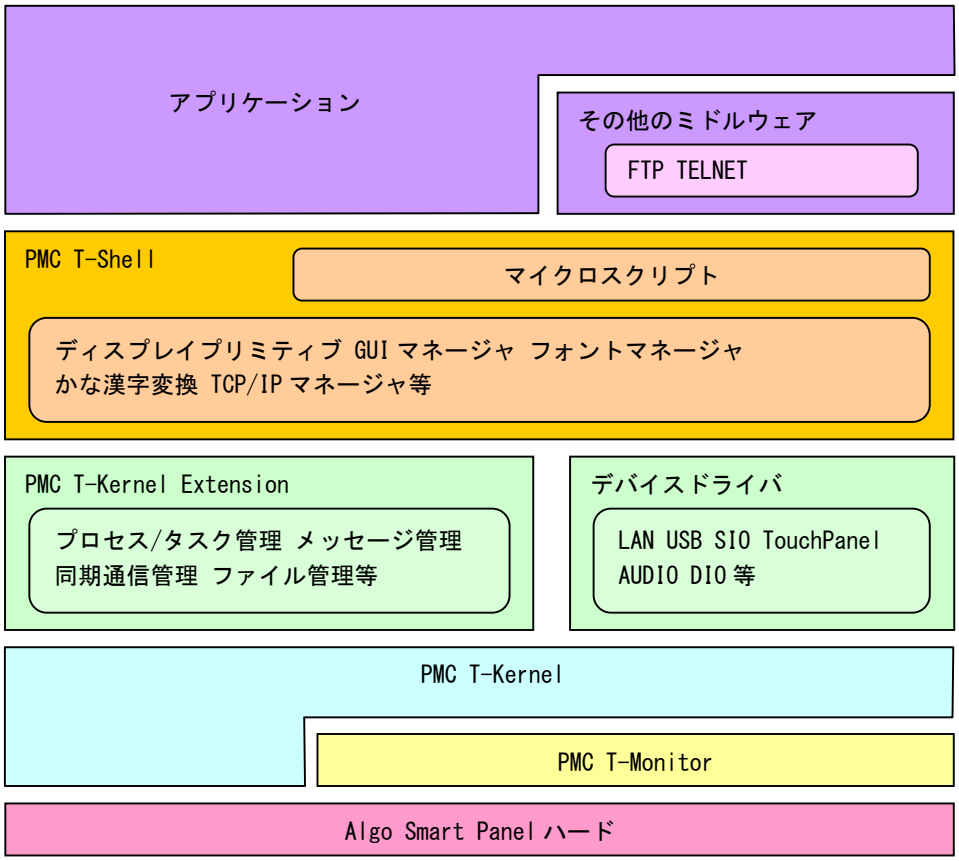


図 1-3-1. Algo Smart PanelでのPMC T-Kernelソフトウェア構成図

構成内容を表 1-3-1 に示します。

表 1-3-1. Algo Smart PanelのPMC T-Kernelソフトウェア構成

名称	内容
PMC T-Monitor	Algo Smart Panel の初期化および PMC T-Kernel のブート
PMC T-Kernel	T-Engine フォーラム版 T-Kernel をパーソナルメディアにて改変したもの
デバイスドライバ	Algo Smart Panel に搭載されているハードウェア用のドライバー式
PMC T-Kernel Extension	開発用基本ミドルウェア（ファイル管理機能やコマンドラインインタラプタ（CLI）等を含む）
PMC T-Shell	T-Kernel のデスクトップ環境 画面上に図形や文字を描画するディスプレイ・プリミティブ、画面上にパーツ類（テキストボックスやスイッチなど）、メニュー、ウィンドウなどを管理する GUI マネージャ、TCP/IP など
アプリケーション	ミドルウェアとして FTP と TELNET を内蔵 開発環境で作成したプログラムが動作

PMC T-Kernel を使用することで、次章より説明する開発環境を使い、リアルタイム性のある GUI アプリケーションを簡単に構築することが可能です。

第２章　　開発環境

本章では、PMC T-Kernel の開発環境について説明します。

２－１　　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、実行ファイルを作成するためのコンパイラ、コンパイルされたプログラムを実行するための実行環境です。
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。

Algo Smart Panel では、ルネサステクノロジ社製の SuperH RISC engine SH4 という CPU を採用しています。つまり、SH4 で動作する形式でのコンパイルが必要になります。
そこで登場するのがクロス開発方式です。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 2-1-1 参照）。
Algo Smart Panel はターゲットマシンとして開発された商品であるため、セルフコンパイル環境は用意していません。

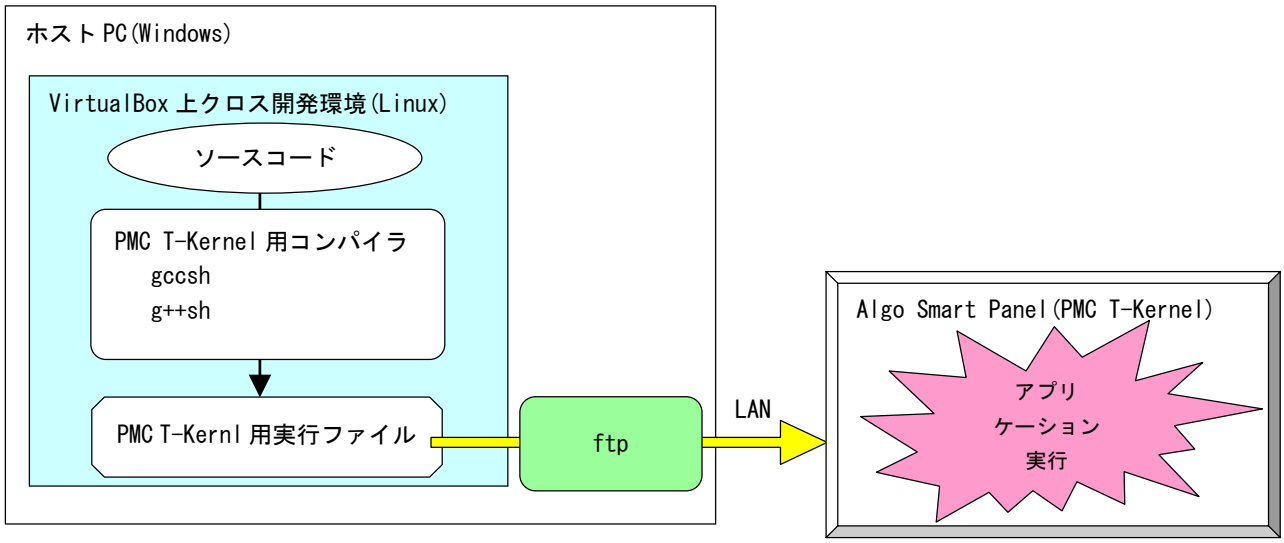


図 2-1-1. クロス開発方式イメージ図

PMC T-Kernel の開発環境は、VirtualBox という仮想マシン上に Ubuntu をインストールし、T-Kernel 用開発環境を組み込んでいます。
Virtual Box をインストールして PMC T-Kernel 開発環境を起動すれば、Algo Smart Panel 用のアプリケーションを開発することが可能です。

２－２　　開発環境インストール

Algo Smart Panel 用のアプリケーション開発環境である **PMC T-Kernel 用開発環境**を Sun Microsystems 社製の VirtualBox 2.XX 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。

２－２－１　　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．　PMC T-Kernel 用開発環境 DVD-ROM VerX.XX

Algo Smart Panel 用アプリケーション開発環境一式です。

※注：通常、開発環境 DVD-ROM は Algo Smart Panel ご購入時には添付しておりません。弊社営業窓口にお問い合わせいただきますと、無償で配布いたします。

２．　VirtualBox Ver2.XX

本書で使用している VirtualBox のバージョンは ver2.1.4 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 Sun Microsystems 社の製品です。VirtualBox の使用にあたっては Sun Microsystems 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

Windows XP または Windows Vista が動作しており、VirtualBox がインストール可能な PC が必要です。VirtualBox のインストールおよび PMC T-Kernel 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 2-2-1-1 の PC スペックが必要になります。

表 2-2-1-1. 必須PCスペック

CPU	X86（1GHz 以上を推奨） Intel, AMD
ﾒﾓﾘ	512MByte 以上（2GByte 以上推奨）**※1**
HDD 空き領域	10GByte 以上（20GByte 以上推奨）**※2**
OS	Windows XP または Windows Vista
ﾌｧｲﾙｼｽﾃﾑ	NTFS **※3**
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、シリアルポート、LAN ポート(ASP と PC の接続に最低でもいずれかひとつが必要になります)

※１　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 PC 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場合があります。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。

※２　PMC T-Kernel 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、開発を進めるにつれ、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空き領域を確保してください。

※３　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。PMC T-Kernel 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 PMC T-Kernel 開発環境は使用できません。

PMC T-Kernel 開発環境をインストールする PC で動作している OS(Windows)と VirtualBox 上で動作する OS(Ubuntu)を区別する為、PC 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。

ここではホスト OS として Windows Vista を用いています。Windows XP を使用する場合、説明のための画像と異なる場合があります。また、説明文中の図は AP-7500/6500/6410/5410/4410 用のものを使用しています。図中のファイル名など AP-2000/3100/3102 のものと異なる場合がございますが、ご了承ください。

２－２－２　VirtualBoxのダウンロード

VirtualBox のインストーラーは 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」を選択してください。

図 2-2-2-1. 公式TOPページ

② 図 2-2-2-2 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」を選択してください。
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。

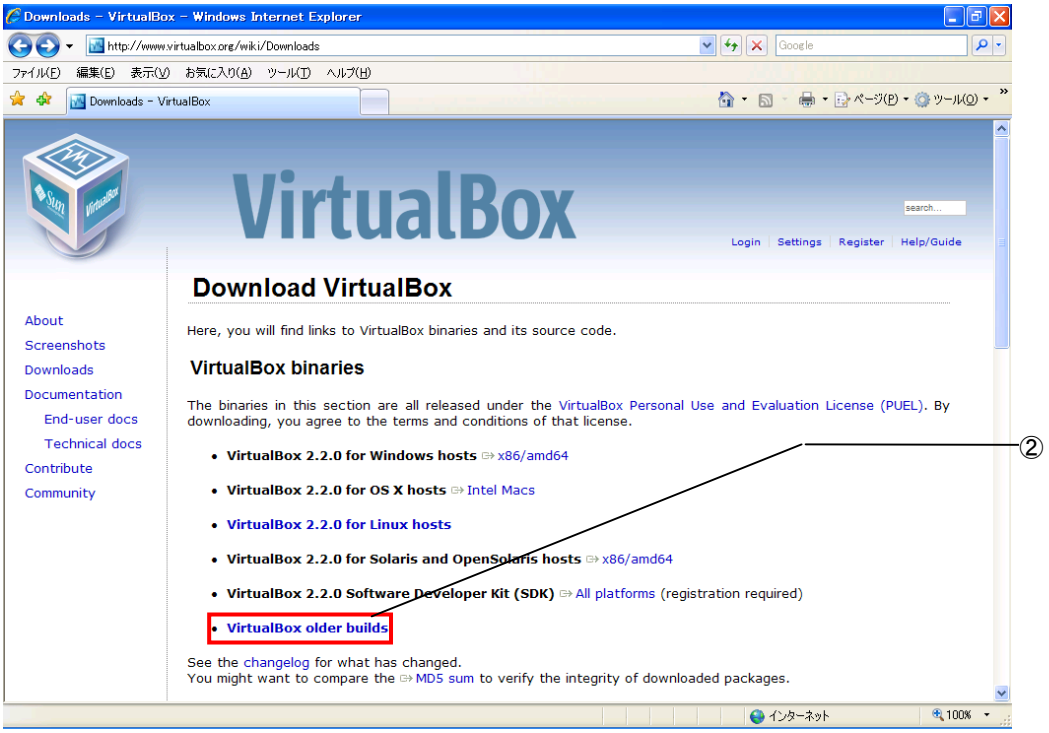


図 2-2-2-2. VirtualBoxダウンロードページ

③　図 2-2-2-3 のような古いバージョンの Virtual Box を選択する画面に変わりますので、「VirtualBox 2.1.4 for Windows hosts」の項目の「x86」を選択してください。

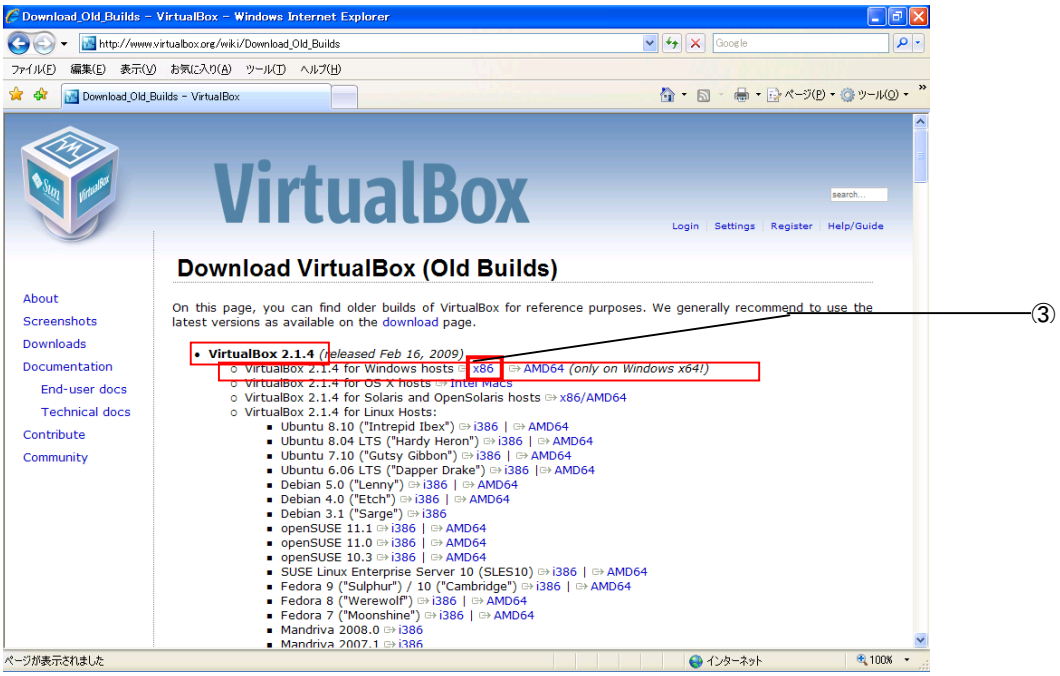


図 2-2-2-3.　古いバージョンのVirtualBoxダウンロードページ

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)」をクリックしてください。ダウンロードが開始されます。

図 2-2-2-4.　ダウンロード確認画面

２－２－３　VirtualBoxのインストール

①　『２－２－２ VirtualBox のダウンロード』でダウンロードした、
VirtualBox-2.1.4-XXXXX-Win_x86.msi のアイコンをダブルクリックしてください。

②　図 2-2-3-1 のセキュリティ警告画面が開きますので、「実行(R)」をクリックしてください。

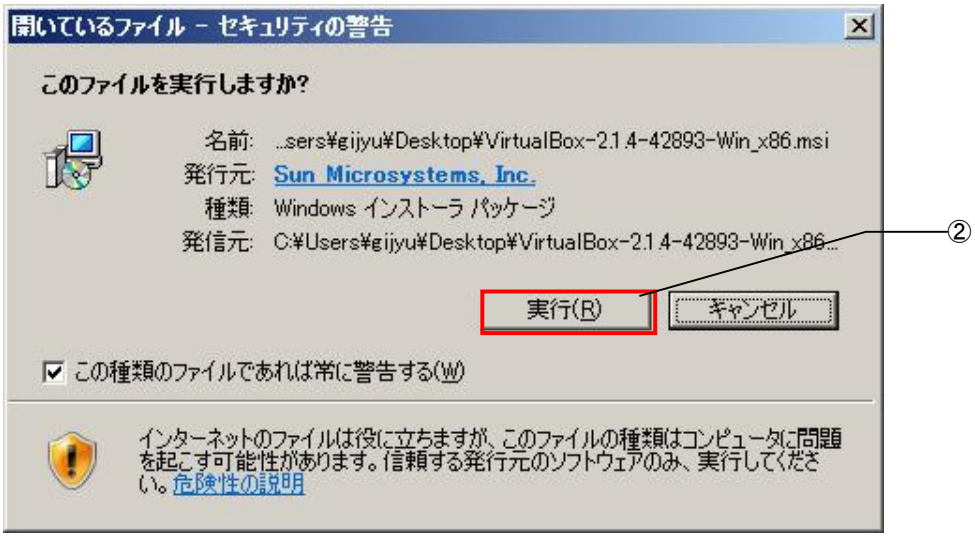


図 2-2-3-1.　セキュリティ警告画面

③　図 2-2-3-2 のような画面が表示されるので、「Next>」をクリックしてください。

図 2-2-3-2.　ダウンローダー起動後の画面

④　図 2-2-3-3 のライセンス確認画面に変わるので、「I accept the terms in the license Agreement」をチェックしてください。

⑤　チェックすると次の画面へ進むことができるようになります。「Next>」をクリックしてください。

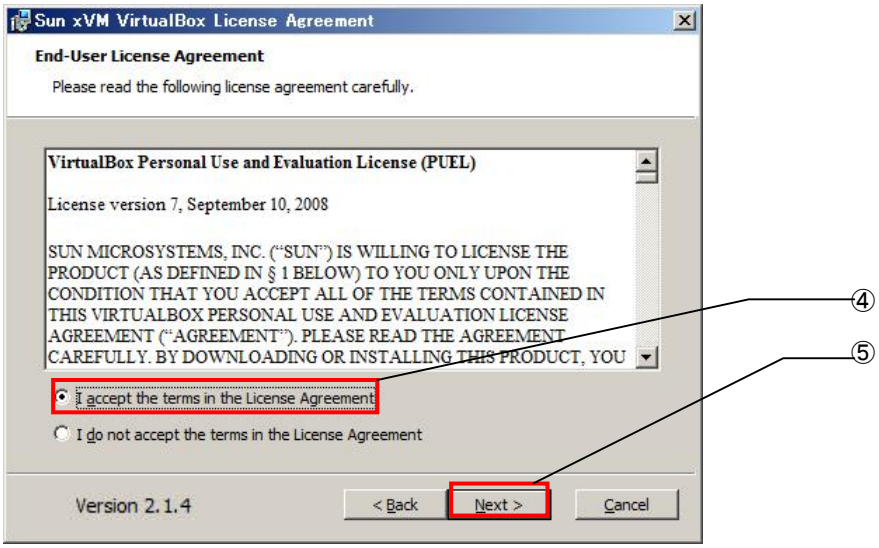


図 2-2-3-3.　ライセンス承諾画面

⑥　図 2-2-3-4 のカスタムインストール画面に変わります。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：お使いの PC の USB 機能を VirtualBox 上で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。

⑦　「Next>」をクリックしてください。

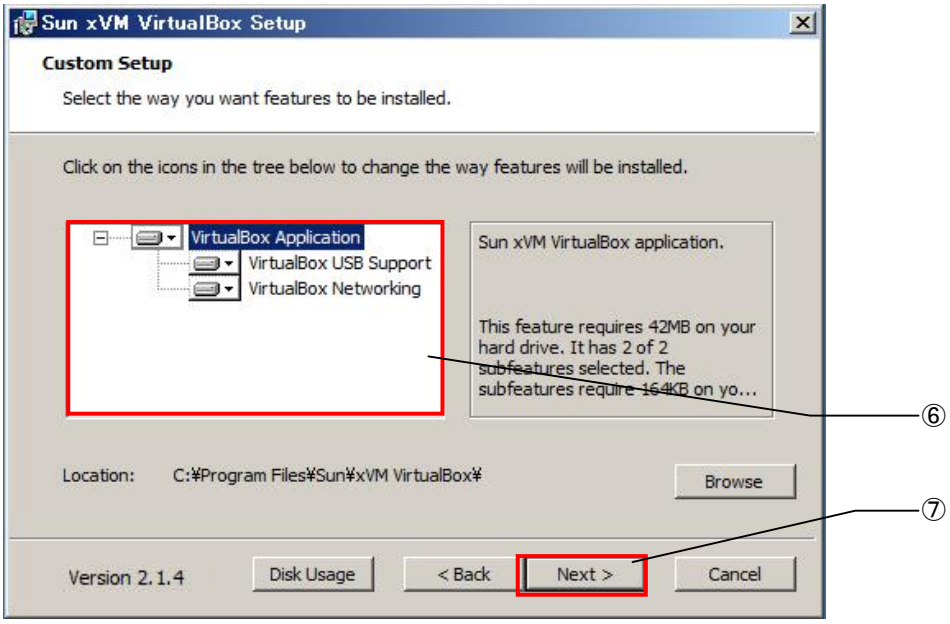


図 2-2-3-4.　カスタムセットアップ画面

⑧　ホスト OS のデスクトップおよびクイックランチャーバーにショートカットを登録するかを選択できます。「Next>」をクリックしてください。

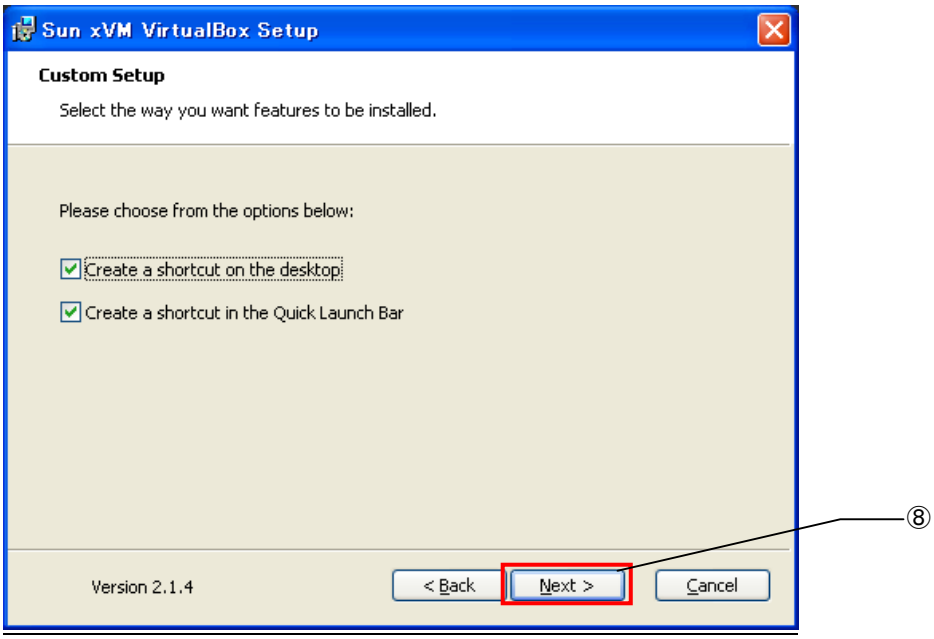


図 2-2-3-5.　ショートカット登録選択画面

⑨　インストール中にネットワーク接続がリセットされることに関しての確認をする画面が表示されます。「Yes」をクリックしてください。

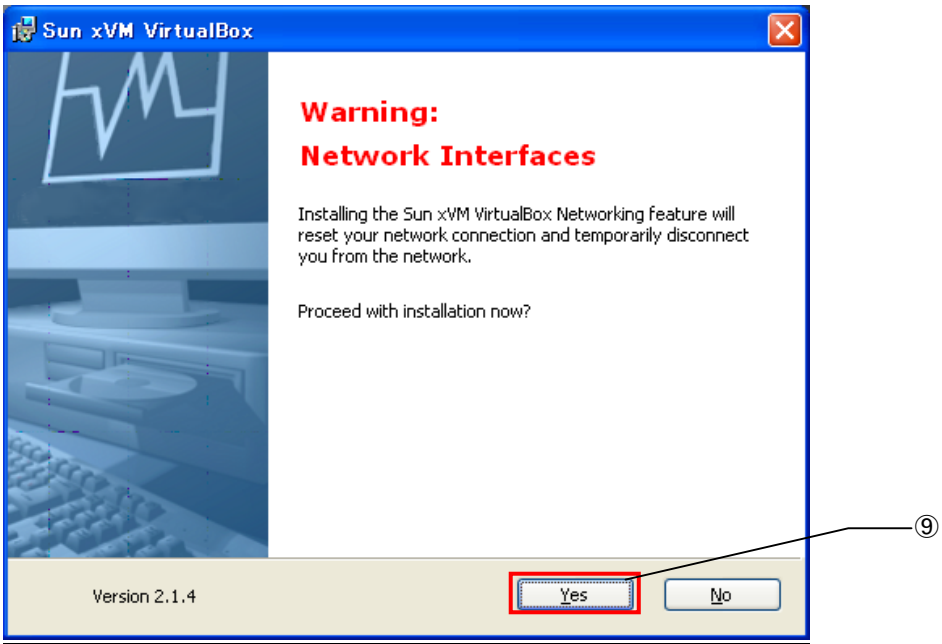


図 2-2-3-6.　ネットワーク警告画面

⑩　図 2-2-3-7 のインストール確認画面に変わるので、「Install」をクリックしてください。

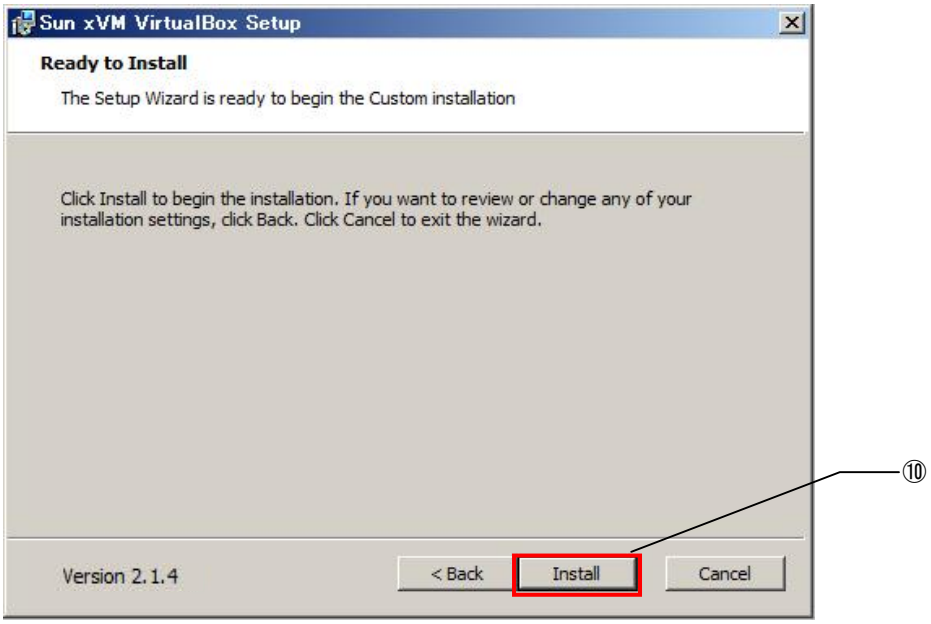


図 2-2-3-7. インストール確認画面

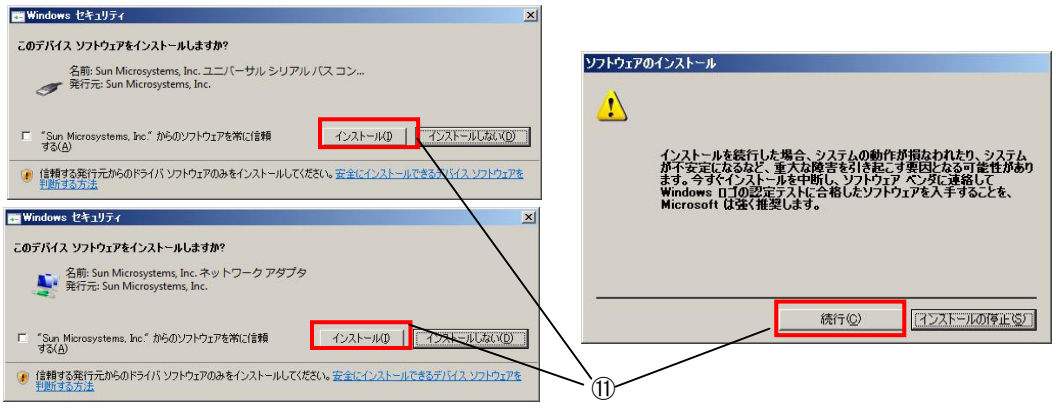
⑪　インストール中、デバイスソフトのインストールを実行するか確認する画面が開きますが、ホスト OS が WindowsXP の場合と WindowsVista の場合とで動作が異なります。

・Windows Vista の場合

インストール中に図 2-2-3-8 のような確認ダイアログが開きますが、いずれも「インストール(I)」をクリックしてください。

・WindowsXP の場合

インストール中に図 2-2-3-8 のような警告ダイアログが２回開きます。これは図 2-2-3-4 の⑥でインストール指定をしたプラグインのドライバのインストールに関する警告です。いずれも「続行(C)」をクリックしてください。



(Windows Vista の場合)　　(Windows Xp の場合)

図 2-2-3-8. ドライバのインストール警告画面

⑫　インストールが終了すると、図 2-2-3-9 が表示されます。「Finish」をクリックしてください。

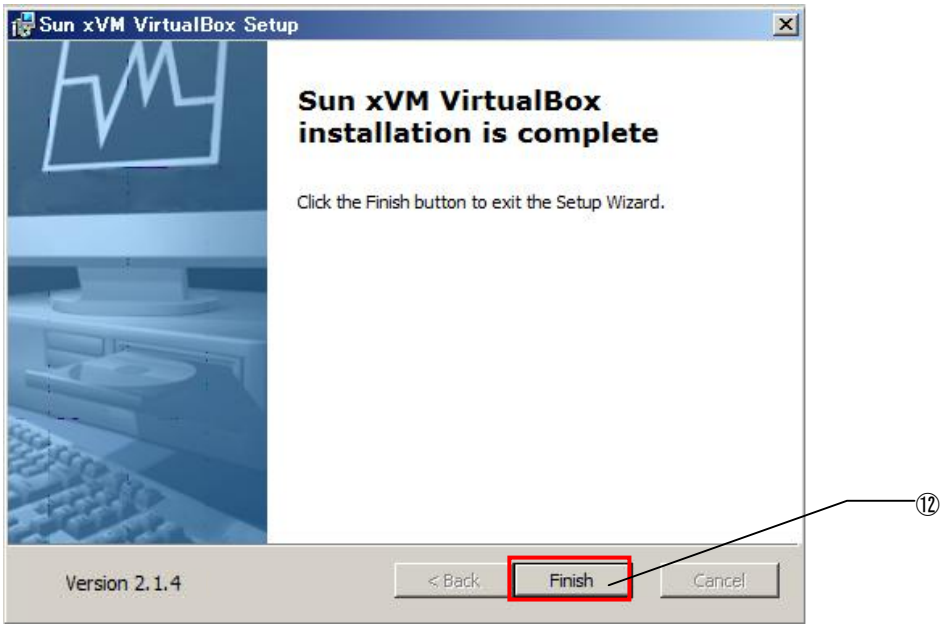


図 2-2-3-9. インストール完了画面

２－２－４　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」→「Sun xVM VirtualBox」→「VirtualBox」を選択します。

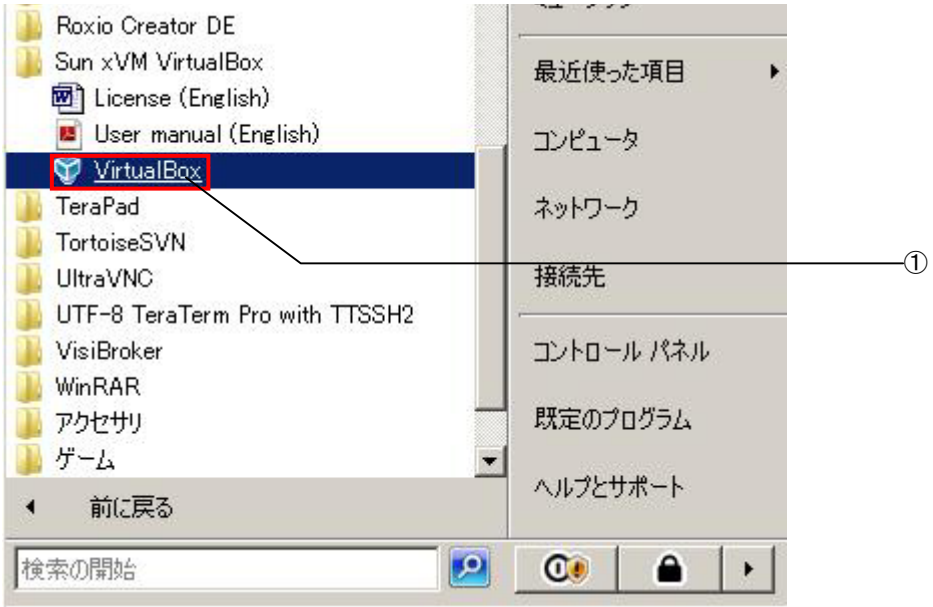


図 2-2-4-1. VirtualBox 起動

② VirtualBox を起動すると、初回起動時に

C:¥User¥<ユーザー名>¥.VirtualBox　　(Windows Vista の場合)
C:¥Document and Setting¥<ユーザー名>¥.VirtualBox　　(Windows XP の場合)

が作成されます。このフォルダは VirtualBox の OS イメージである、vdi 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¥T-Kernel_development_sh7760_v102.exe をこのフォルダに展開してください。
T-Kernel_development_sh7760.exe を展開すると T-Kernel_development_sh7760_v102.vdi が作成されます。

③ VirtualBox が起動し、図 2-2-4-2 のような VirtualBox メイン画面が表示されるので、「新規」をクリックしてください。

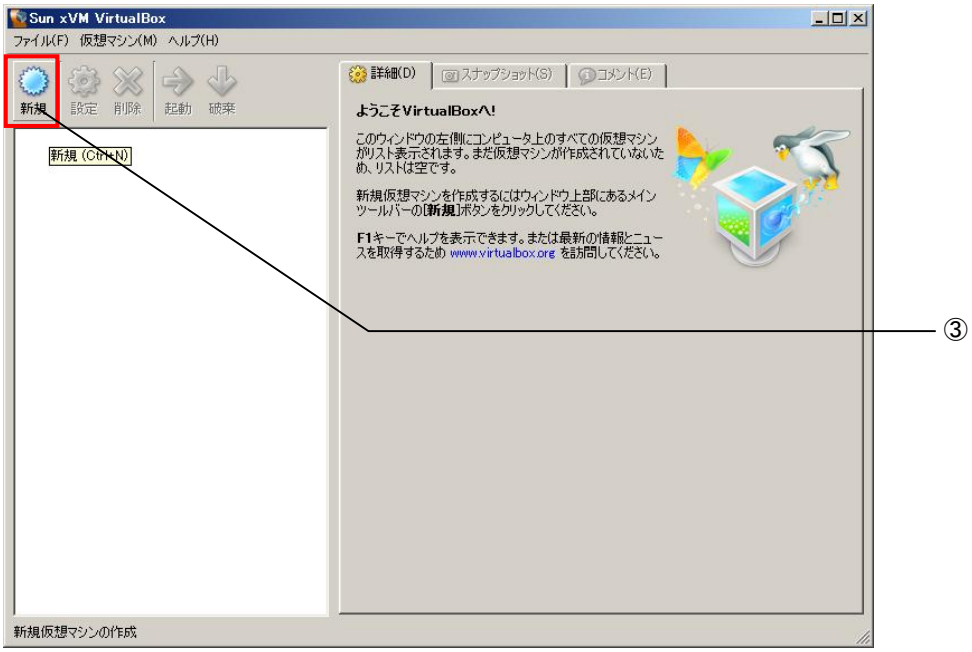


図 2-2-4-2. VirtualBox メイン画面

④　図 2-2-4-3 のような新規仮想マシンの作成ウィザードが起動するので「次へ(N) >」をクリックしてください。

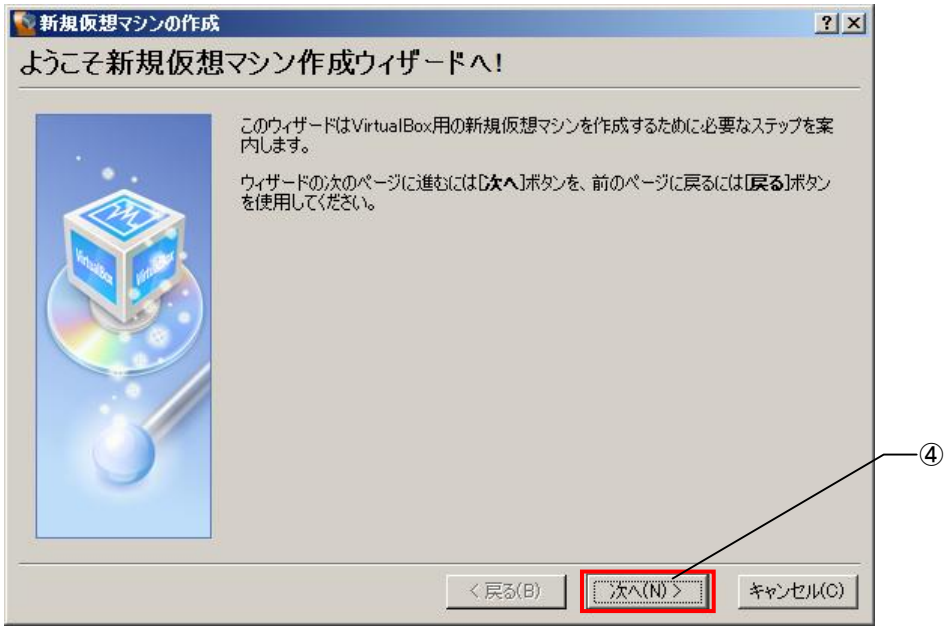


図 2-2-4-3. 新規仮想マシン作成ウィザード

⑤　図 2-2-4-4 のマシン名と OS 種類選択画面に変わるので、「名前(N)」には「T-Kernel_development」と入力てください。

⑥　「OS タイプ(T)」では「Linux」「Ubuntu」を選択してください。

⑦　「次へ(N) >」をクリックしてください。

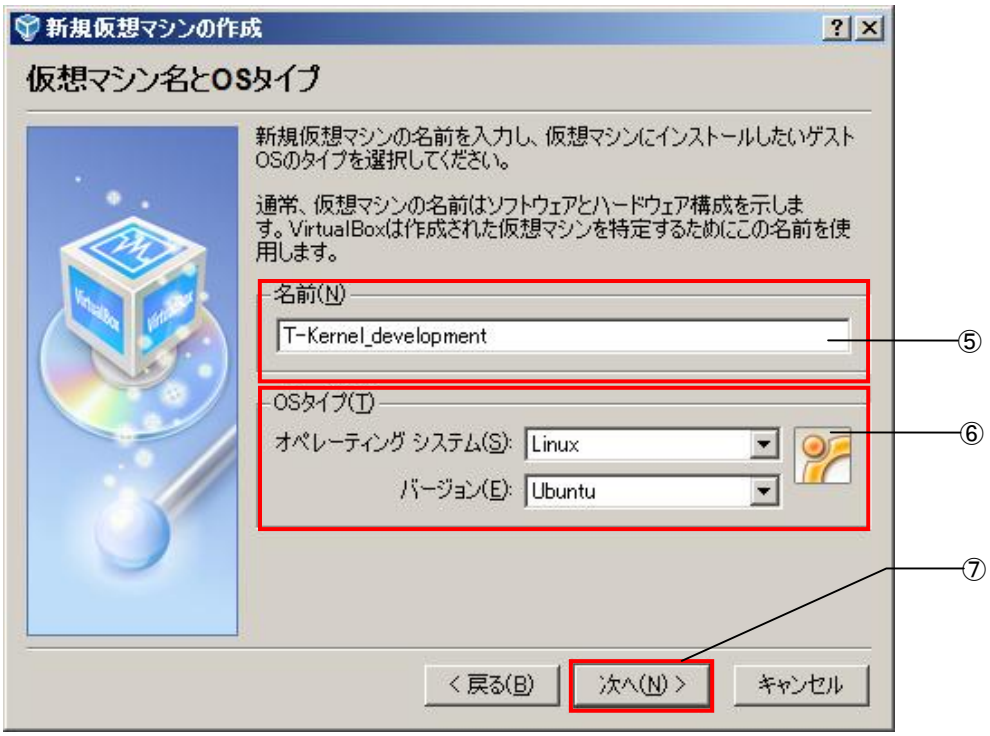


図 2-2-4-4. マシン名、OS タイプの設定

⑧　図 2-2-4-5 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。
※　ご使用の PC に応じてご指定ください。必須環境は 256MByte 以上です。

⑨　「次へ(N) >」をクリックしてください。

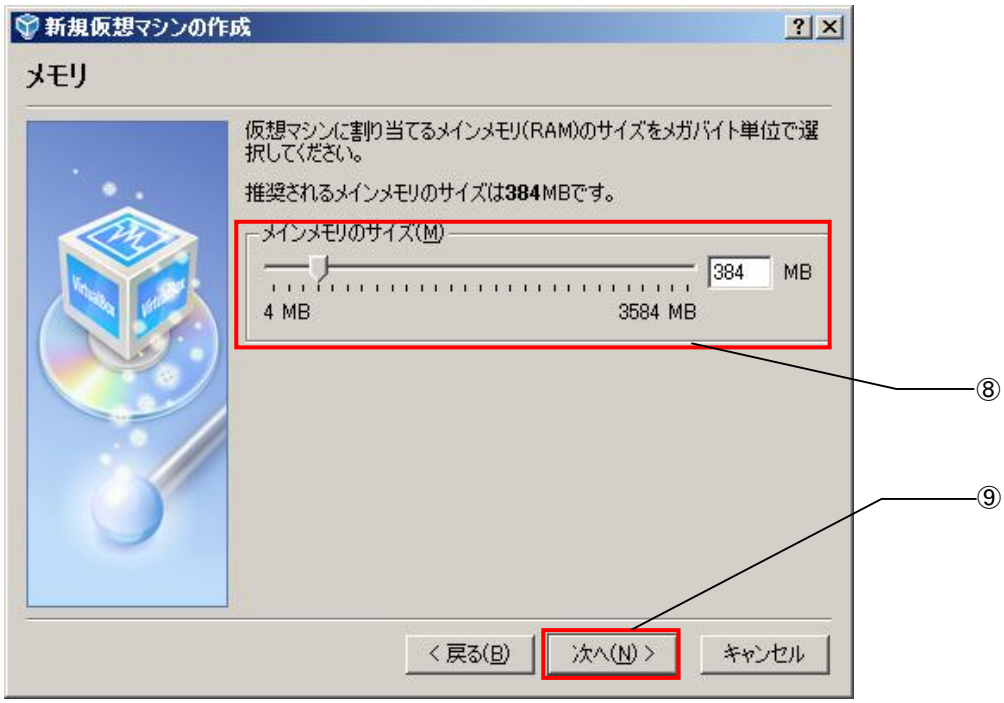


図 2-2-4-5.　メモリの設定

⑩　図 2-2-4-6 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、「選択(X)...」をクリックしてください。

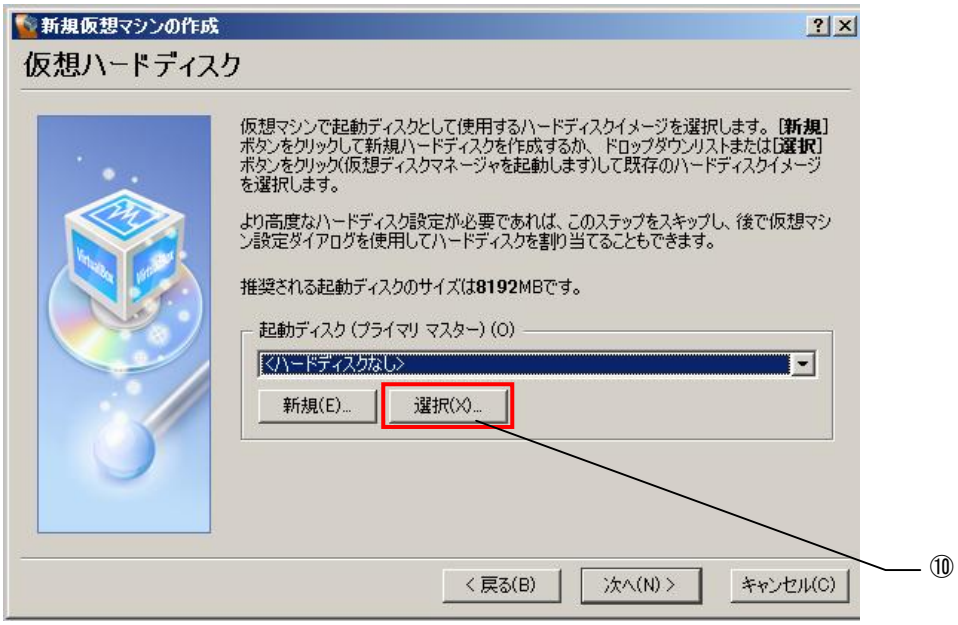


図 2-2-4-6.　仮想ハードディスクのイメージ選択画面

⑪　図 2-2-4-7 の仮想ディスクマネージャ画面が開きます。
「追加」をクリックすると、ファイル選択画面が開くので、あらかじめ展開しておいた、T-Kernel_development_sh7760_v102.vdi を選択してください。中央のリストに T-Kernel_development_sh7760_v102.vdi が追加されます。

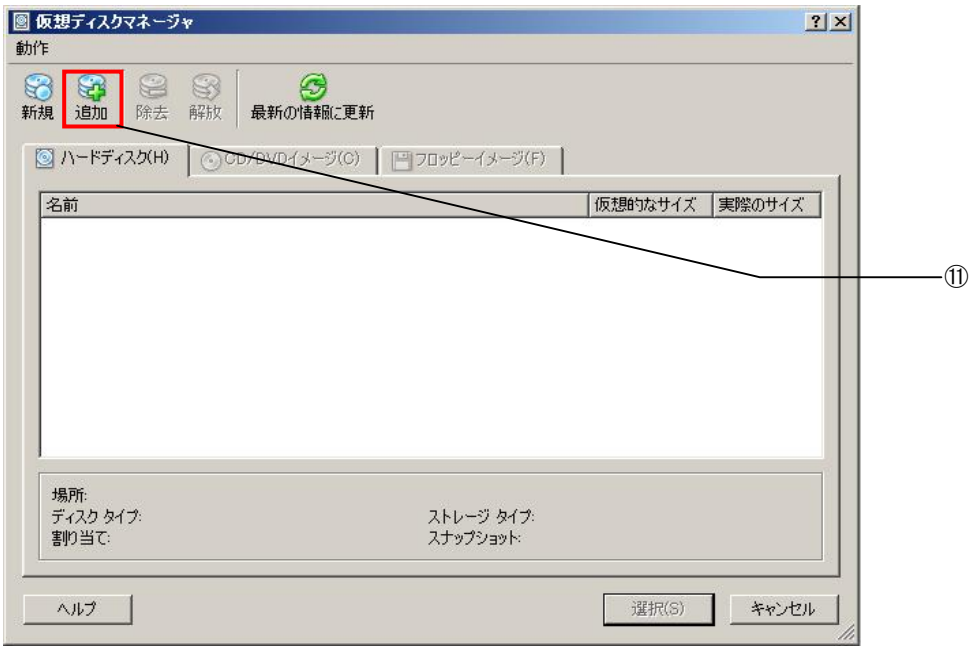


図 2-2-4-7. 仮想ディスクマネージャ

⑫　中央リスト部分に T-Kernel_development_sh7760_v102.vdi が表示されるのでこれを選択します。

⑬　「選択(S)」をクリックしてください。

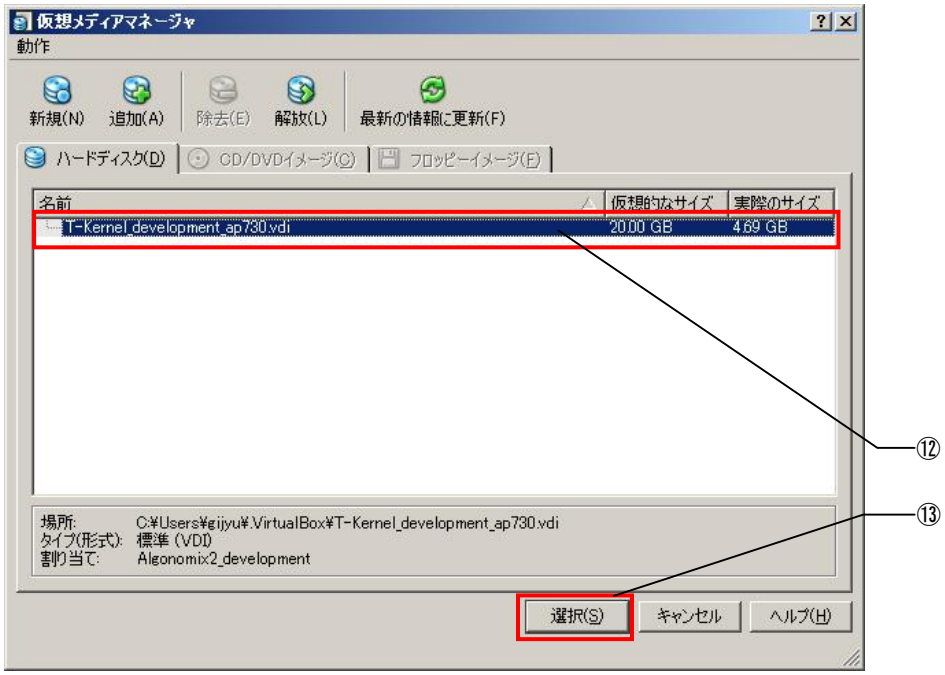


図 2-2-4-8. 仮想ディスクマネージャ

⑭　仮想ハードディスクイメージ選択画面に戻るので、中央メニューに
T-Kernel_development_sh7760_v102.vdi が選択されていることを確認してください。

⑮　「次へ(N)>」をクリックしてください。

図 2-2-4-9. 仮想ハードディスクイメージ選択画面

⑯　図 2-2-4-10 の確認画面へ移ります。これまで設定した情報が表示されます。正しいことを確認し、「完了(F)」をクリックし作業を終了します。

図 2-2-4-10.　確認画面

※　仮想マシンの各種設定を変更する場合は、VirtualBox 付属のドキュメントを参照してください。

２－２－５　仮想マシンの起動

弊社配布の OS イメージ、T-Kernel_development_sh7760_v102.vdi は既に Algo Smart Panel 開発環境がインストールされています。なお、T-Kernel_development_sh7760_v102.vdi のユーザー名とパスワードは、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 ID、Password の入力を求められるので、下記のユーザー名とパスワードを入力してください。

ユーザー名　asdusr
パスワード　asdusr

① VirtualBox メイン画面左側の List の中から PMC T-Kernel_Development を選択してください。

② 「起動」をクリックしてください。

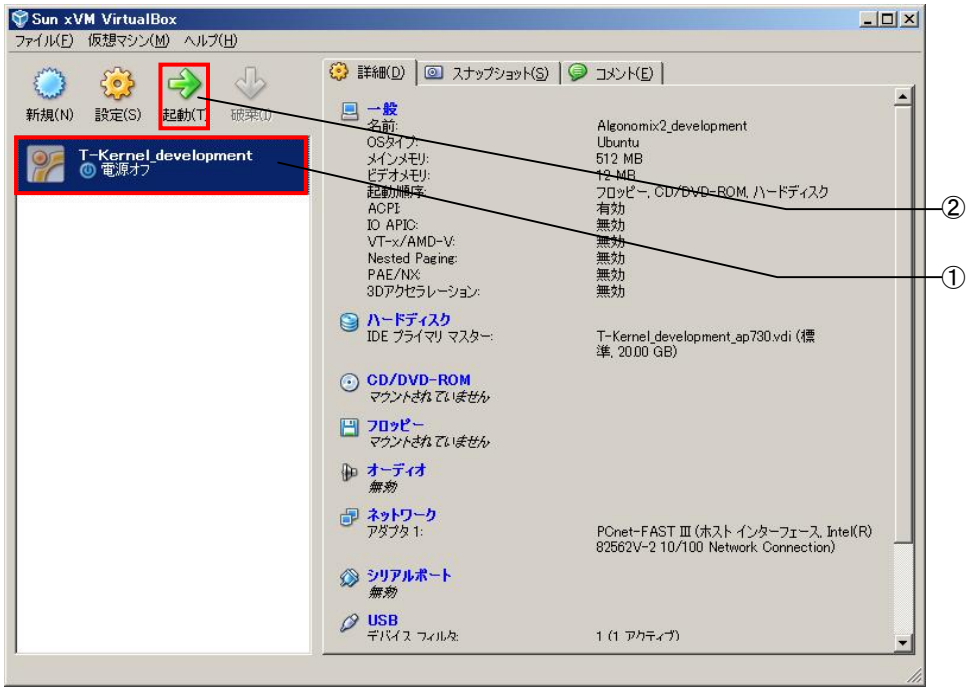


図 2-2-5-1. 仮想マシンの起動

③　Ubuntu が起動し、図 2-2-5-2 のような仮想マシンのユーザー名、パスワード入力画面が表示されます。
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てください。

ユーザ名　　asdusr
パスワード　　asdusr

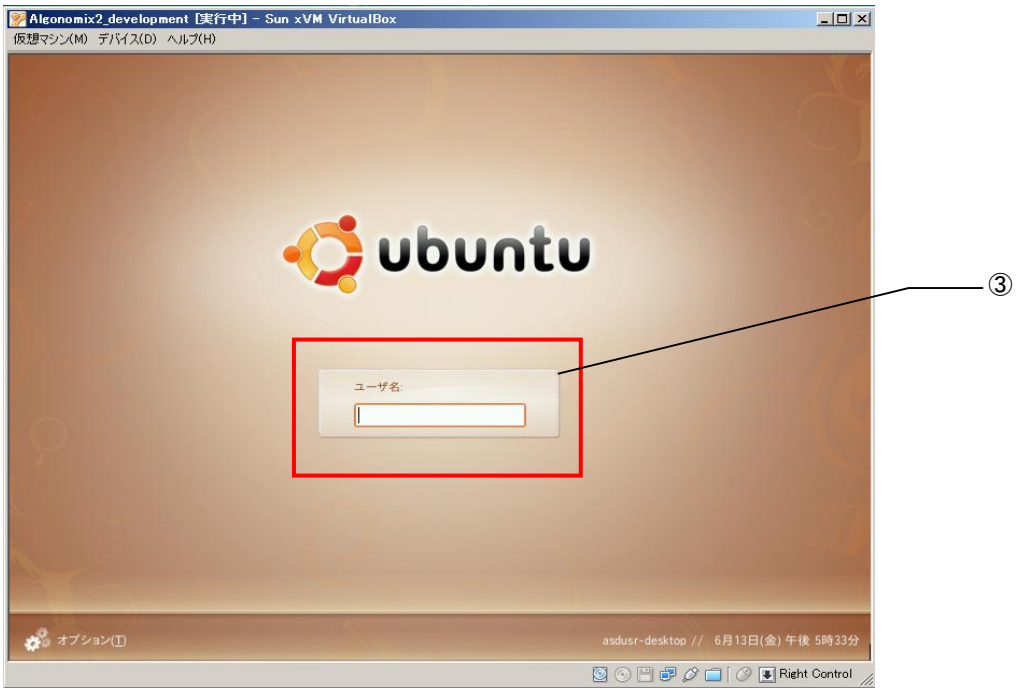


図 2-2-5-2. ユーザー名、パスワード入力画面

図 2-2-5-3 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。

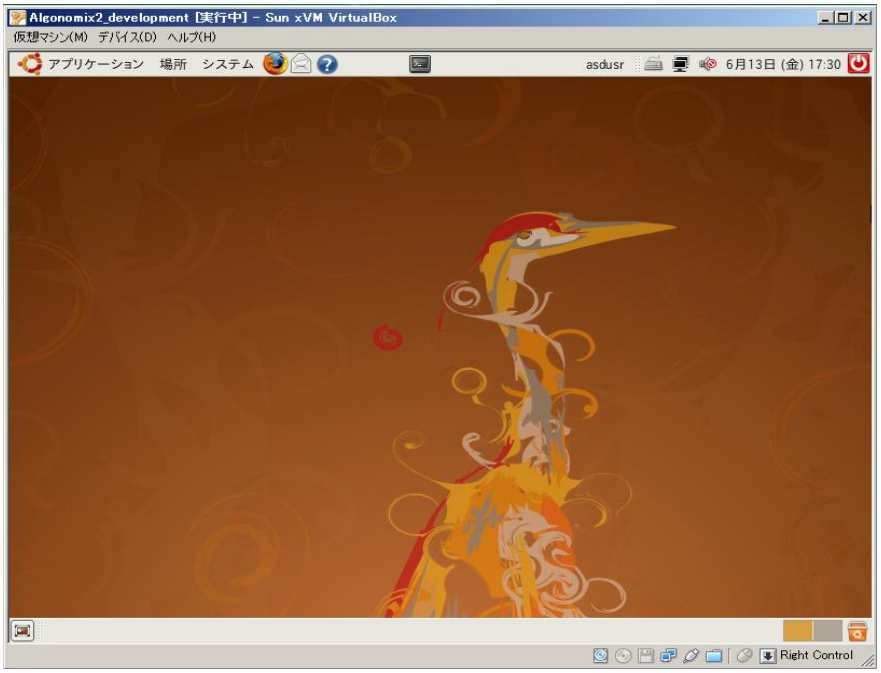


図 2-2-5-3. デスクトップ画面

※　仮想マシン起動中はゲストOSの表示領域にマウスカーソルがマウントされます。ホストOSへマウスカーソルを戻すには「右 Ctrl」キーを押下することで切り替えが可能です。VirtualBox を再びアクティブにすることで再度ゲストOSの表示領域にマウントされます。
また、マウスカーソルはVirtualBoxのGuestAdditionをインストールすることで、ホストOSとゲスト OS の間で共有することができるようになります。GuestAdditionについては「２－５ Guest Additionsのインストール」を参照してください。

２－２－６　PMC T-Kernel用開発環境のディレクトリ構成

PMC T-Kernel 用開発環境のディレクトリ構成の説明を行います。PMC T-Kernel 用開発環境のディレクトリ構成を表2-2-6-1に示します。

表2-2-6-1. PMC T-Kernel用開発環境のディレクトリ構成

```
usr--+--local--+--te-asd_sh7760
               +--te
               +--src
               +--ws
```

また、これらのディレクトリの意味は以下のとおりです。

表2-2-6-2. ディレクトリの内容(抜粋)

ディレクトリ名	内容
/usr/local/te-asd_sh7760	AP-2000/3100/3102用PMC T-Kernel開発環境一式が格納されています。
/usr/local/te	te-asd_sh7760へのリンク PMC T-Kernel標準のディレクトリ
/usr/local/src	WideStudioのソースが格納されています。
/usr/local/ws	WideStudioのインストールディレクトリです。

２－３　　ネットワーク設定

仮想 PC のネットワーク設定を行います。仮想 PC のネットワーク接続の方法には何通りかありますが、本章ではその一例として、ホストインターフェースを用いる方法を紹介します。

２－３－１　　ホストOS側(VirtualBox)からの設定

①T-Kernel_development_sh7760_v102.vdi をマウントしている仮想マシンを選択した状態にしてください。

②「ネットワーク」をクリックしてください。

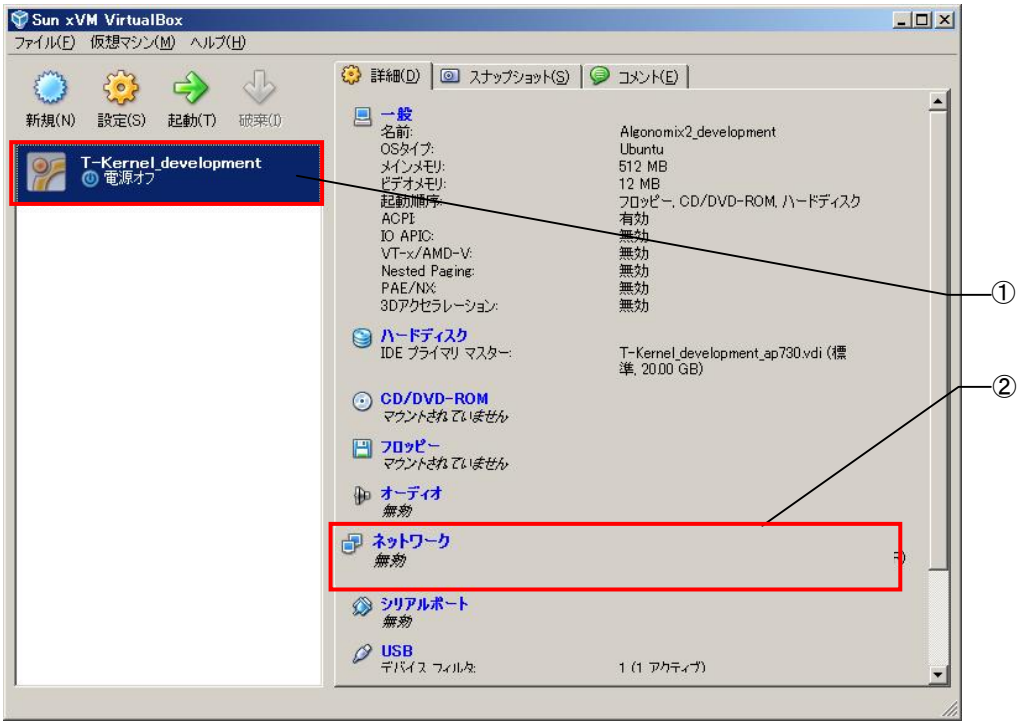


図 2-3-1-1. VirtualBox メイン画面からネットワーク設定画面へ

③「アダプタ 1」のタブが有効になっていることを確認してください。

④「ネットワークアダプタを有効化(E)」にチェックを入れます。

⑤「アダプタタイプ(D)」は Pcnet-FASTⅢ(Am79C973)にします。

⑥「割り当て(A)」をホストインターフェースにします。

⑦「接続(B)」にチェックを入れます。

⑧ホスト OS で設定されているネットワークインターフェースのリストが表示されます。ゲスト OS で使用したいインターフェースを選択してください。

⑨「OK」をクリックすると、VirtualBox メイン画面に戻ります。メイン画面の②の部分に「アダプタ 1」の設定が反映されていることを確認してください。

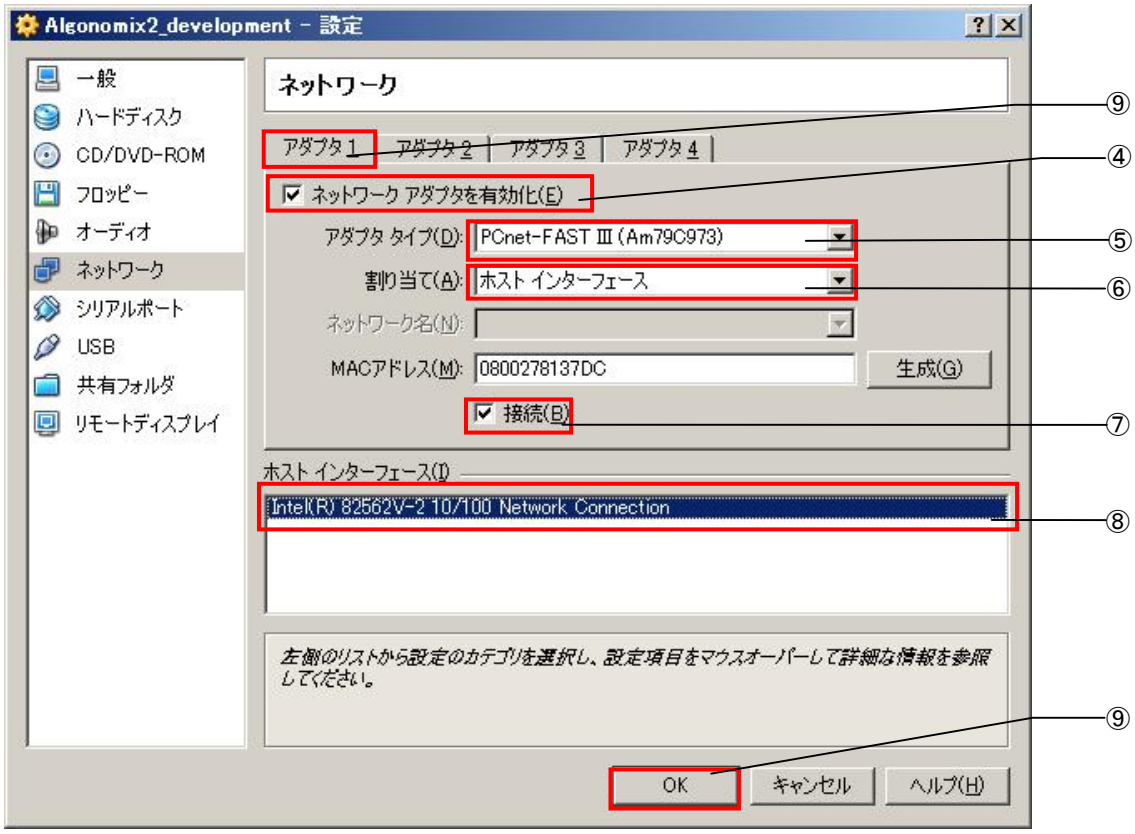


図 2-3-1-2.　ネットワークアダプタの設定

２－３－２　ゲストOS側のネットワーク設定

仮想 PC 側のネットワーク設定を行います。「アプリケーション」 → 「アクセサリ」 → 「端末」を選択し、以下のようにコマンドを入力します。

```
# sudo su
```

ここでパスワードを要求されるのでパスワードを入力します。

```
asdusr

# ifconfig eth0 down
# ifconfig eth0 up
```

この操作には数十秒かかる場合があります。
以上の操作でゲスト OS 側でネットワーク接続ができるようになります。
仮想 PC 再起動時などに、再びネットワーク接続が無効になる場合がありますが、その場合もこのコマンドを入力することで接続を有効にできます。

※　お客様のご利用の環境によってはこの接続方法では正常に接続できない可能性もあります。その他の接続方法に関しては、VirtualBox のマニュアルを参照してください。

２－４　　USB機器の接続

USB 機器を PC に接続した際、仮想マシンでも使用できるように設定します。

２－４－１　　ホストOSからの設定

①VirtualBox メイン画面で仮想マシンを選択し、右のメニューの「USB」を選択してください。

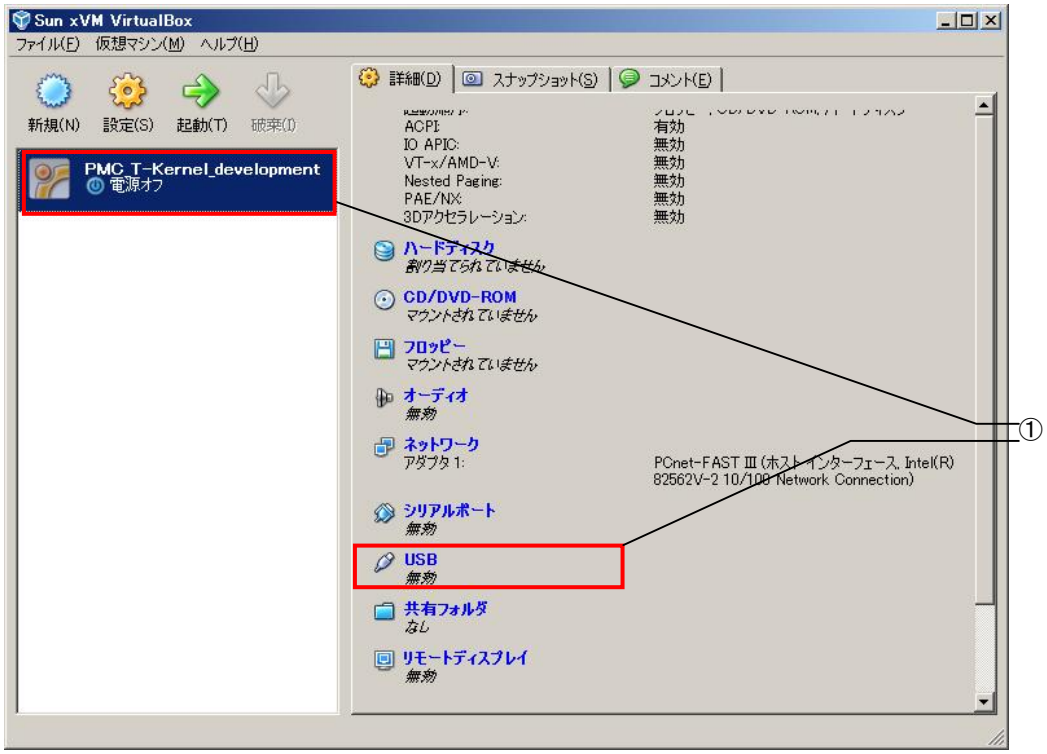


図 2-4-1-1. VirtualBox メイン画面

②USB 設定の画面が開きます。「USB コントローラを有効化(U)」にチェックを入れてください。

③「USB2.0 コントローラを有効化(H)」にチェックを入れてください。
USB デバイスフィルタの設定が有効になります。

④「空のフィルタを追加」をクリックします。
新しいフィルタが 1 項目追加されます。

⑤「OK(O)」をクリックしてください。

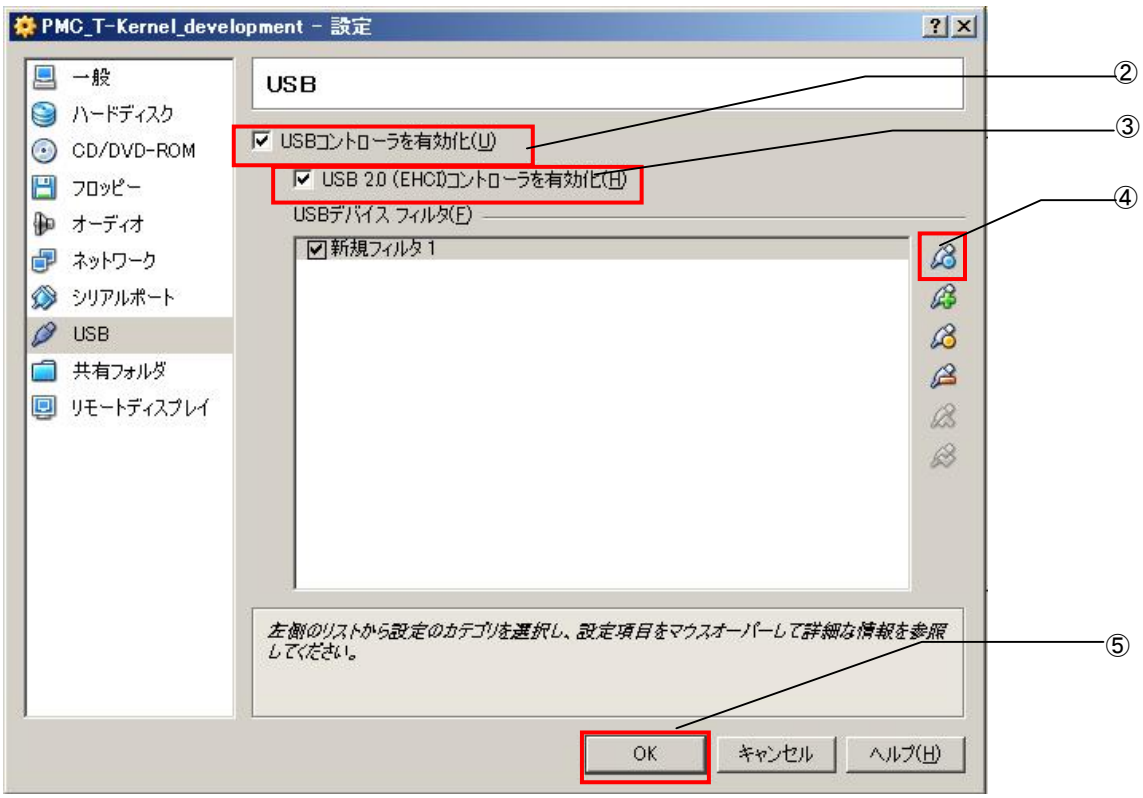


図 2-4-1-2. USB 設定画面

以上でホスト OS 側からの設定は終了しました。
VirtualBox メイン画面の①USB の項目に「デバイスフィルタ　1(1 ｱｸﾃｨﾌﾞ)」と表示されていれば成功です。

２－４－２　仮想マシンでUSB機器を使用する方法

ホスト OS 側で初めて使用する USB 機器の場合、仮想マシンを起動していない状態でホスト OS 側でその USB 機器のドライバをインストールする必要があります。USB メモリのような多くの USB 機器は PC に接続するだけでドライバのインストールが自動的に開始されます。(画像は Windows Vista のものです)

図 2-4-2-1. ホスト OS 上での USB ドライバのインストール

その他、特殊なインストール方法が必要な機器につきましては、それぞれのマニュアル等を参考にドライバのインストールを行ってください。
ドライバのインストールが終了したら、一旦 USB 機器を取り外し、仮想マシンを起動してください。
仮想マシン起動後、USB 機器を PC に接続することで仮想マシンが USB 機器を認識するようになります。
仮想マシン上で USB デバイスを認識すると、図 2-4-3-1 のように、画面上部メニューの「デバイス(D)」-「USB デバイス(U)」の項目に追加されます。

２－４－３　仮想マシンでUSB機器の使用を終了する方法

①USB 機器を取り外す場合は「デバイス(D)」 - 「USB デバイス(U)」を選択し、取り外したい USB 機器を選択してください。

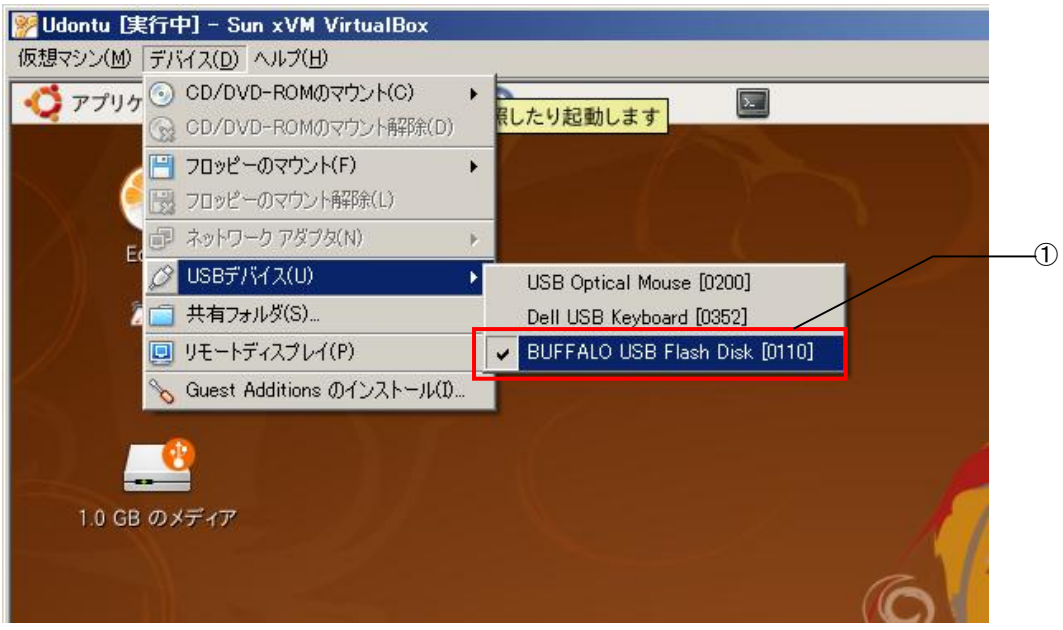


図 2-4-3-1. ゲスト OS 上での USB 機器の取り外し

仮想マシン上で USB デバイスがアンマウントされ、ホスト OS 上で認識されます。
ホスト OS の USB 機器の取り扱い方法に従い、USB 機器を取り外してください。

２－５　　Guest Additionsのインストール

仮想マシンの起動中は仮想マシンの表示領域にマウスカーソルがマウントされ、画面外に出なくなりますが、VirtualBox の GuestAddition をインストールすることで、マウスカーソルをゲスト OS とホスト OS との間で共有できるようになります。
GuestAddition は PMC T-Kernel 開発環境を利用するにあたって必須ではありませんが、GuestAdditdion を導入することで下記の機能が拡張されるため、インストールすることを推奨します。

- ゲスト OS とホスト OS とのマウスカーソルの共有化
- ゲスト OS とホスト OS とのフォルダ共有機能
- 仮想マシンの解像度の変更

①GuestAdditionCD イメージをマウントします。キーボードの右 Ctrl キーを押下しマウスカーソルのマウントを解除し、「デバイス(D)」 → 「Guest Additiona のインストール(I)...」を選択します。
デスクトップに「VBOXADDITIONS_2.X.X_...」がマウントされたことを確認してください。

図 2-5-1. インストールディスクのマウント

GuestAddition をインストールします。「アプリケーション」 → 「アクセサリ」 → 「端末」を選択し、以下のようにコマンドを入力します。

```
# sudo su
```

ここでパスワードを要求されるのでパスワードを入力します。

```
 asdusr

# cd /media/cdrom0/
# ./VBoxLinuxAdditions-x86.run
```

インストールが始まります。この作業には数分かかります。

```
Successfully Installed the VirtualBox Guest Additions.
You must restart your guest system in order to complete the installation.
```

上記のようなメッセージが表示されたらインストールは終了です。
GuestAddition の機能は VirtualBox を再起動させた後から有効になります。

※　Ubuntu のアップデートなどでカーネルが書き換えられた場合、GuestAddition が無効になる場合があります。この場合はもう一度 GuestAddition をインストールしなおす必要があります。

２－６　　シリアルコンソール接続について

VirtualBox 上 Ubuntu から USB-シリアル変換機を通して、シリアルコンソール接続にて Algo Smart Panel と接続する方法の説明をします。あらかじめ『２－４ USB 機器の接続』をよく読み、VirtualBox 上で USB 機器を認識できるよう設定しておいてください。

①USB-シリアル変換機を PC に接続してください。VirtualBox が正常に認識し、図 2-6-1 のように「デバイス(D)」→「USB デバイス(U)」に追加され、チェックマークがついていることを確認してください。チェックマークがついていない場合は、デバイス名をクリックすることでチェックマークをつけることができます。

図 2-6-1. USB デバイスの認識

②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」をクリックするとコンソールが立ち上がります。gterm でボーレート 115200、/dev/ttyUSB0 デバイスドライバを使用して起動します。

```
# /usr/local/te/tool/Linux-i686/etc/gterm –b –l /dev/ttyUSB0
```

デバイス名/dev/ttyUSB0 は Ubuntu が最初に認識した USB シリアル機器に有効です。複数の USB シリアル機器を使用する場合、接続した順に/dev/ttyUSB0、/dev/ttyUSB1、…のようにデバイス名が割り振られます。使用したい USB シリアル機器にあったデバイス名を入力してください。
この操作で PC と接続している Algo Smart Panel と VirtualBox(Ubuntu)がシリアルコンソール接続できるようになります。

２－７　　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。
PMC T-Kernel 用開発環境に組み込まれている WideStudio/MWT は V3.98-1 をベースに Algo Smart Panel 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てみましょう。

※　T-Kernel 版 WideStudio ではコンソールアプリケーションの作成はできません。コンソールアプリを作成する場合は『２－９ WideStudio を使用しないコンソールアプリケーション開発』を参照してください。

２－７－１　WideStudioの起動

デスクトップ上の 

のアイコンをクリックすると WideStudio が起動します。

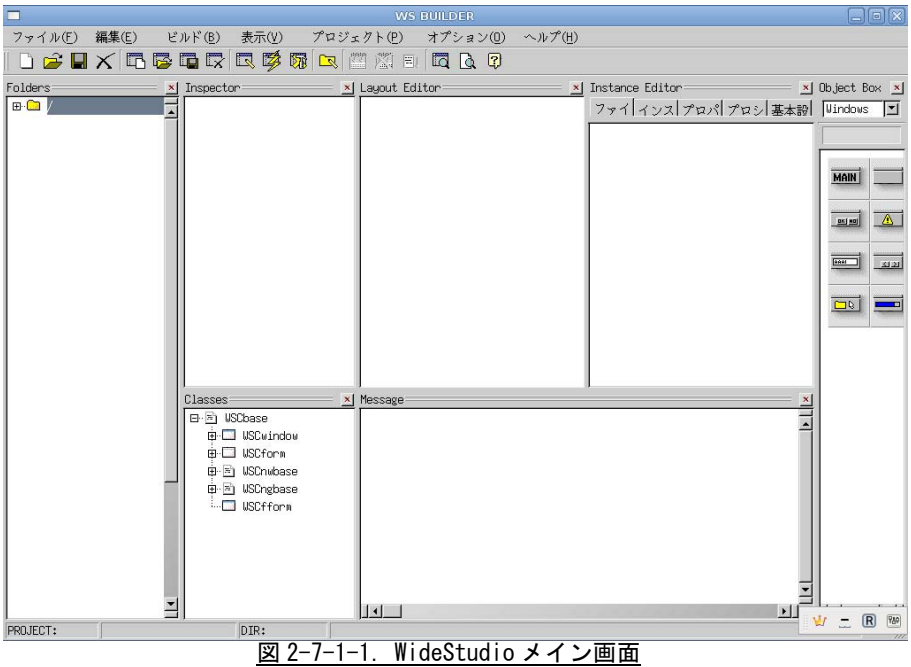


図 2-7-1-1. WideStudio メイン画面

２－７－２　プロジェクトの新規作成

はじめに、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。

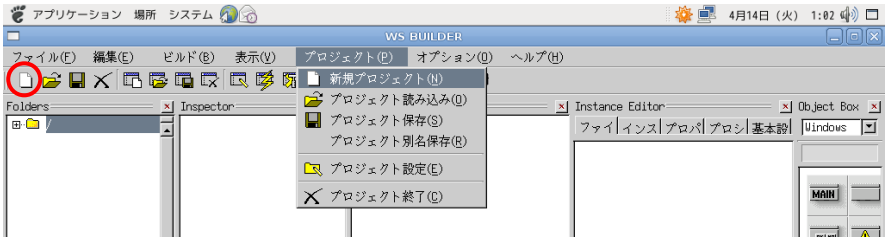


図 2-7-2-1. 新規プロジェクト作成


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)」をクリックすると、図 2-7-2-2 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。今回はデフォルトの newproject とします。

図 2-7-2-2. プロジェクト作成画面

 をクリックすると図 2-7-2-3 のようなファイル選択ダイアログが表示されます。

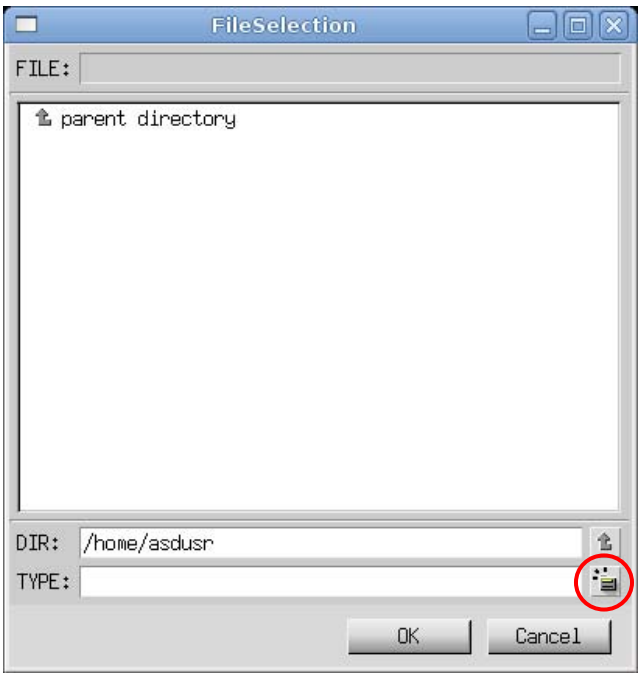


図 2-7-2-3. ファイル選択画面

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。

※　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。

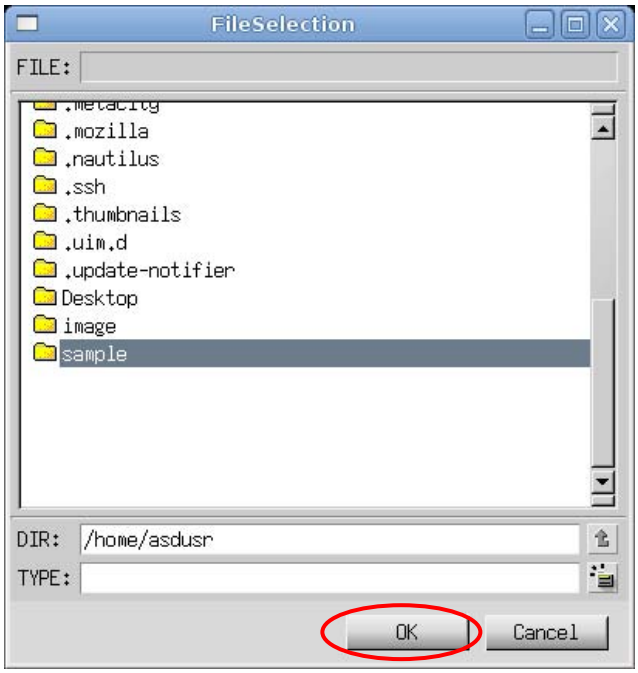


図 2-7-2-4. sample ディレクトリの作成

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。

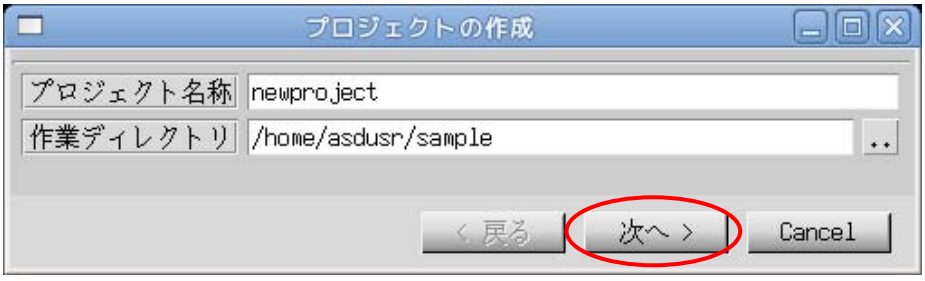


図 2-7-2-5. プロジェクト名と作業ディレクトリの設定完了

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。

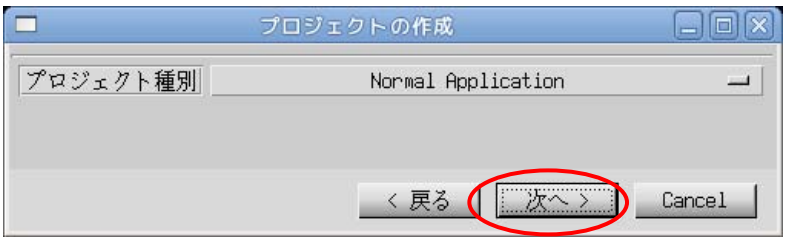


図 2-7-2-6. プロジェクト種別の選択

アプリケーションで使用する文字コード(ロケール種別)、プログラミング言語を選択します。言語種別は C/C++ を選択してください。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、ここのロケール種別は SJIS にするべきです。今回は PMC T-Kernel 標準コードである日本語(EUC)を選択します。「生成」ボタンをクリックします。

※　Algo Smart Panel 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

※　言語種別は C/C++のみサポートしています。

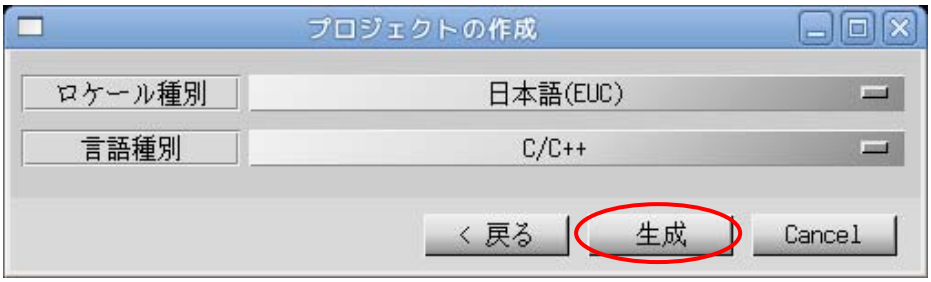


図 2-7-2-7.　ロケール種別と言語種別を選択

以上でプロジェクトの作成は完了です。

２－７－３　　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。

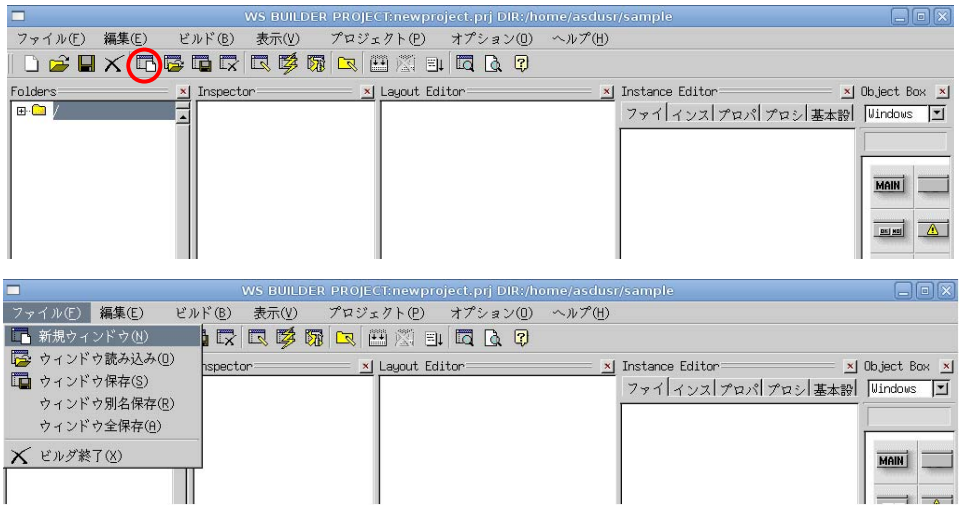


図 2-7-3-1.　新規ウィンドウ作成

 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)」をクリックすると、図 2-7-3-2 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 2-7-3-2.　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。基本的にメインのアプリケーションウィンドウはデフォルト設定のままでいいと思います。設定を変更せずに「次へ」をクリックします。

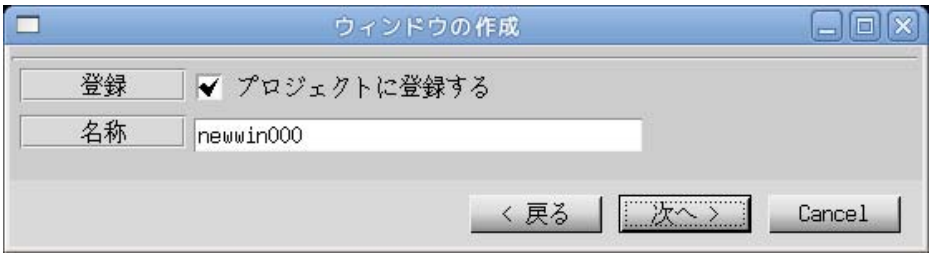


図 2-7-3-3. アプリケーションウィンドウ名称設定

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ぶことで、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自動的に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 2-7-3-4. 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と、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 2-7-3-1 を参考にプロパティを変更してください。

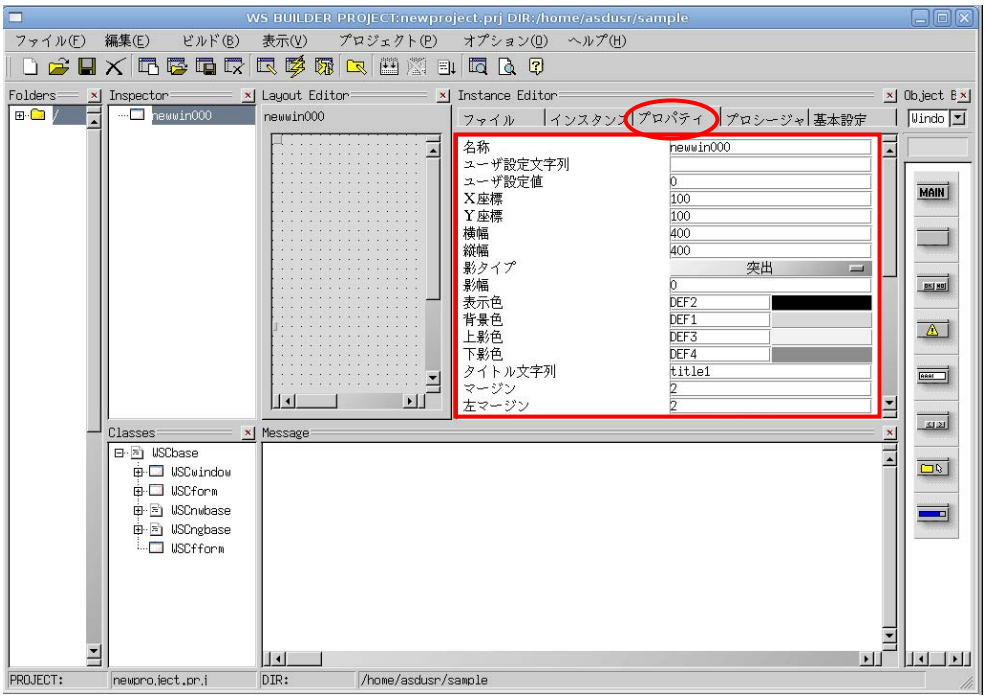


図 2-7-3-5. アプリケーションウィンドウの生成

表 2-7-3-1. 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。

２－７－４　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、「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。

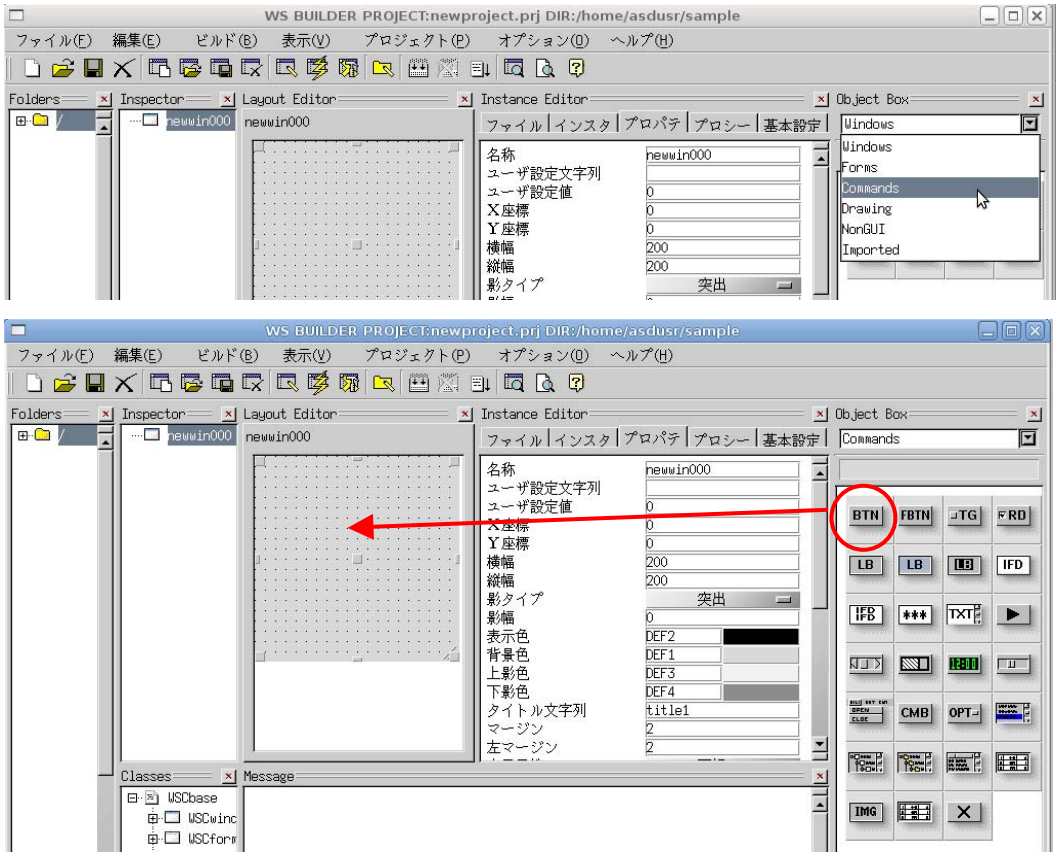


図 2-7-4-1. ボタンの配置

図 2-7-4-2 のようにボタンが配置されましたら、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。

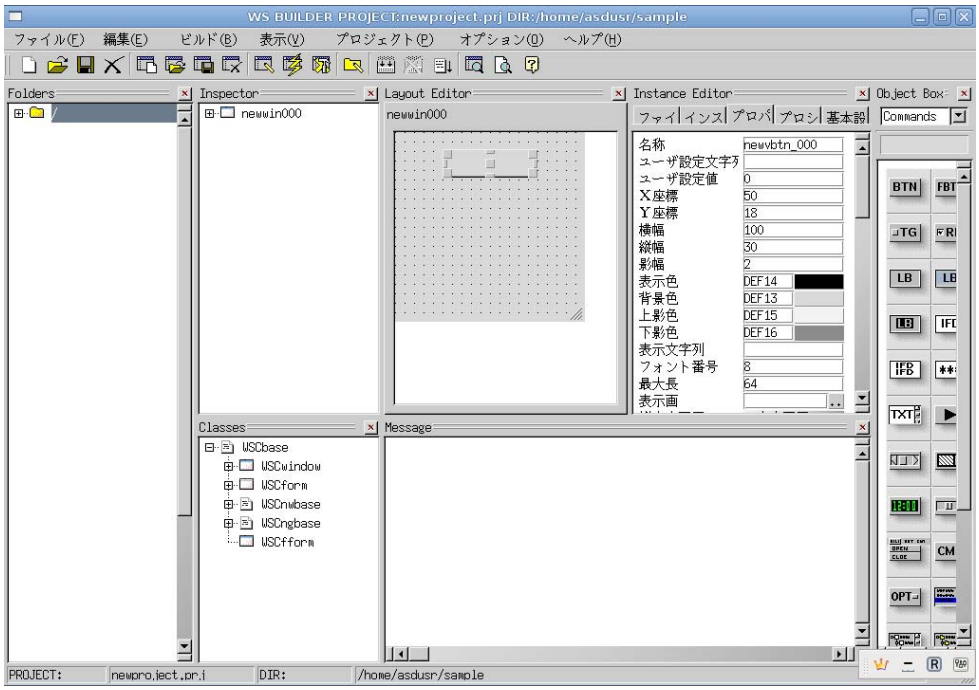


図 2-7-4-2. アプリケーションウィンドウに配置されたボタン

ここでは、例として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ます。他の部品の詳細については、WideStudio ホームページや WideStudio の書籍等を参照してください。

※　Algo Smart Panel は組み込み用の機器のため、CPU やメモリリソース、T-Kernel という OS の違い等の関係上動作しない部品があります。
※　PMC T-Kernel に組み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。組み込まれているドライバについては、DVD に収録されている、<DVD>¥doc¥PMC-TKernel¥library.pdf[T-Engine/ライブラリ説明書]を参照してください。

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と、ボタンのプロパティを設定できるようになります。表 2-7-4-1 を参考に値を変更してください。

表 2-7-4-1. 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

２－７－５　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、プログラムを実行したいとき、プロシージャと呼ばれるプログラムを貼り付けることで実現することができます。

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」タブをクリックすると、図 2-7-5-1 のような画面になります。

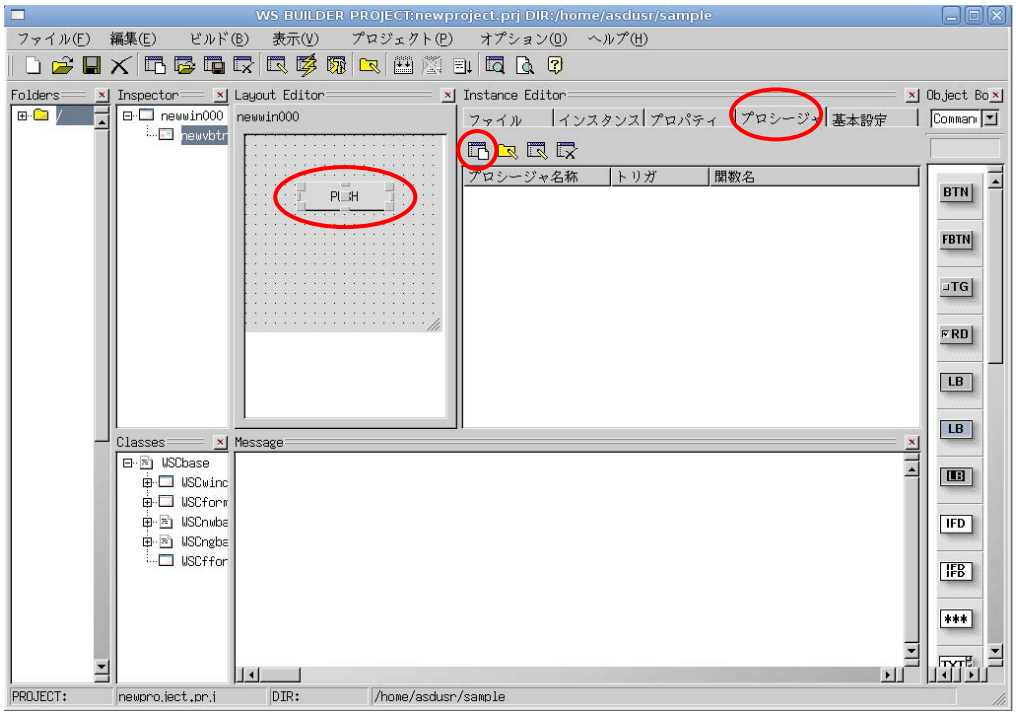


図 2-7-5-1. イベントプロシージャの作成

 をクリックすると、図 2-7-5-2 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。

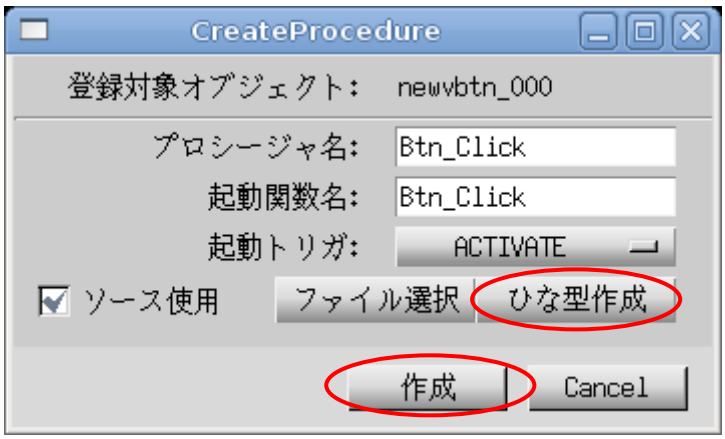


図 2-7-5-2. イベントプロシージャ作成ダイアログ

「ひな型作成」ボタンをクリックすると、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 2-7-5-3 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。

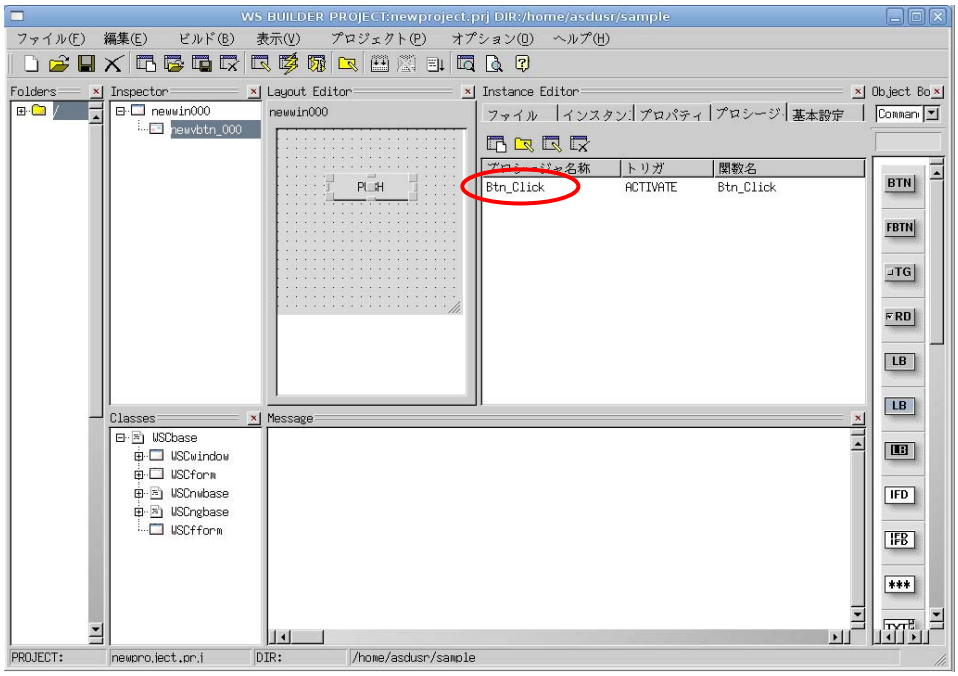


図 2-7-5-3. 作成されたイベントプロシージャ

２－７－６　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 2-7-6-1 のようになっています。

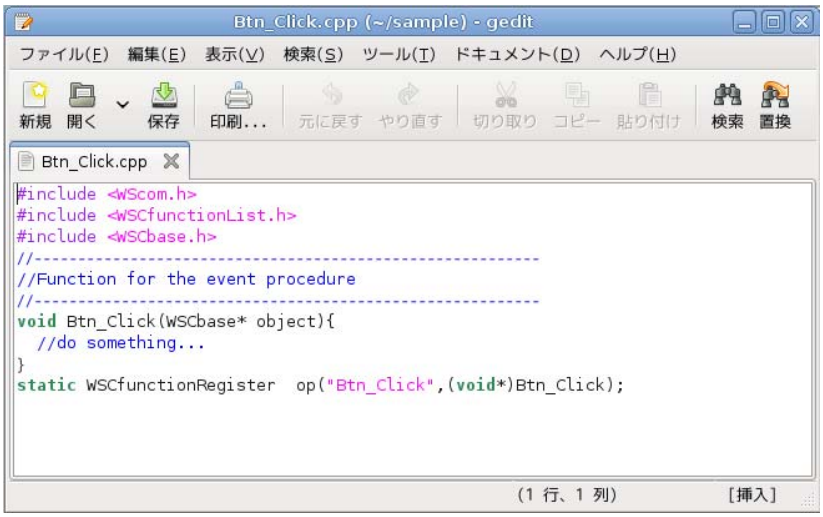


図 2-7-6-1. イベントプロシージャ初期ソースコード

リスト 2-7-6-1 のようにコードを変更します。

リスト 2-7-6-1. 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。

２－７－７　セルフコンパイル

サンプルプログラムのビルドを行います。メニューの「プロジェクト(P)→プロジェクトの設定(E)」をクリックしてください。

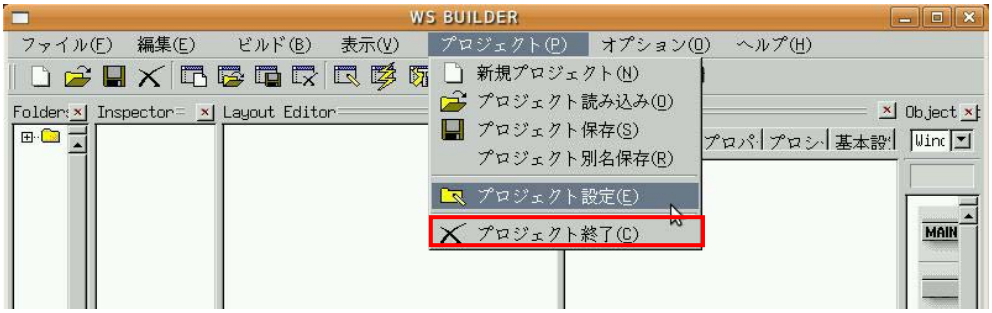


図 2-7-7-1. サンプルプロジェクトの設定

プロジェクトの設定ウィンドウが開くので、「基本設定」タブをクリックしてください。
「TARGET」の項目のプルダウンメニューが「Native」になっていることを確認してください。「Native」以外が表示されている場合はプルダウンメニューをクリックし、Native を選択してください。

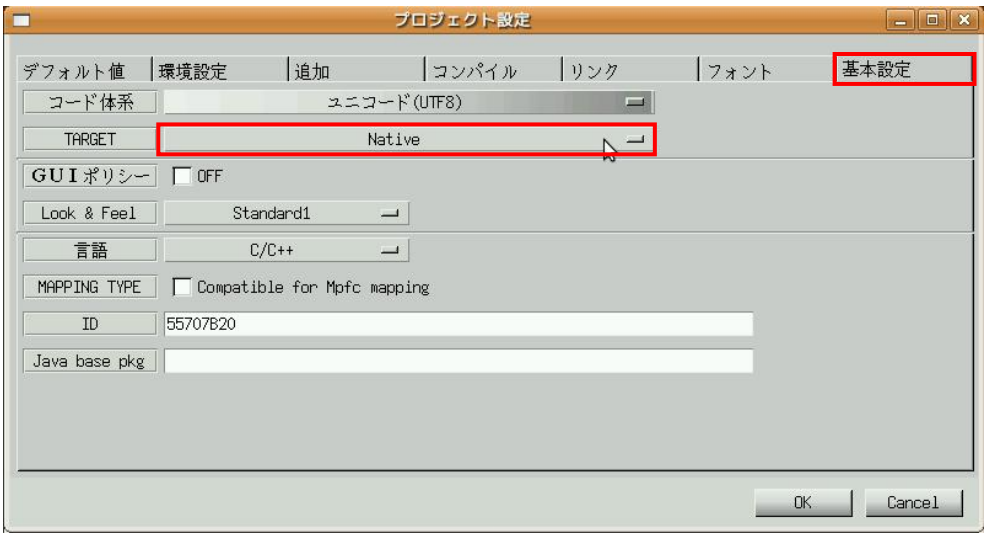


図 2-7-7-2. プロジェクト設定ウィンドウ(基本設定タブ)

TARGET が Native になっていることを確認した後、OK ボタンをクリックしプロジェクト設定ウィンドウを閉じます。

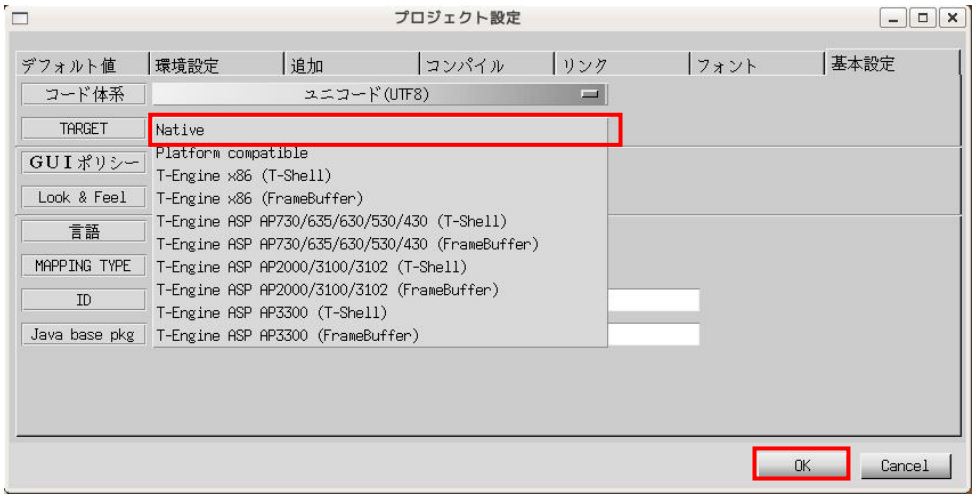


図 2-7-7-3. サンプルプロジェクトの Target の確認

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。

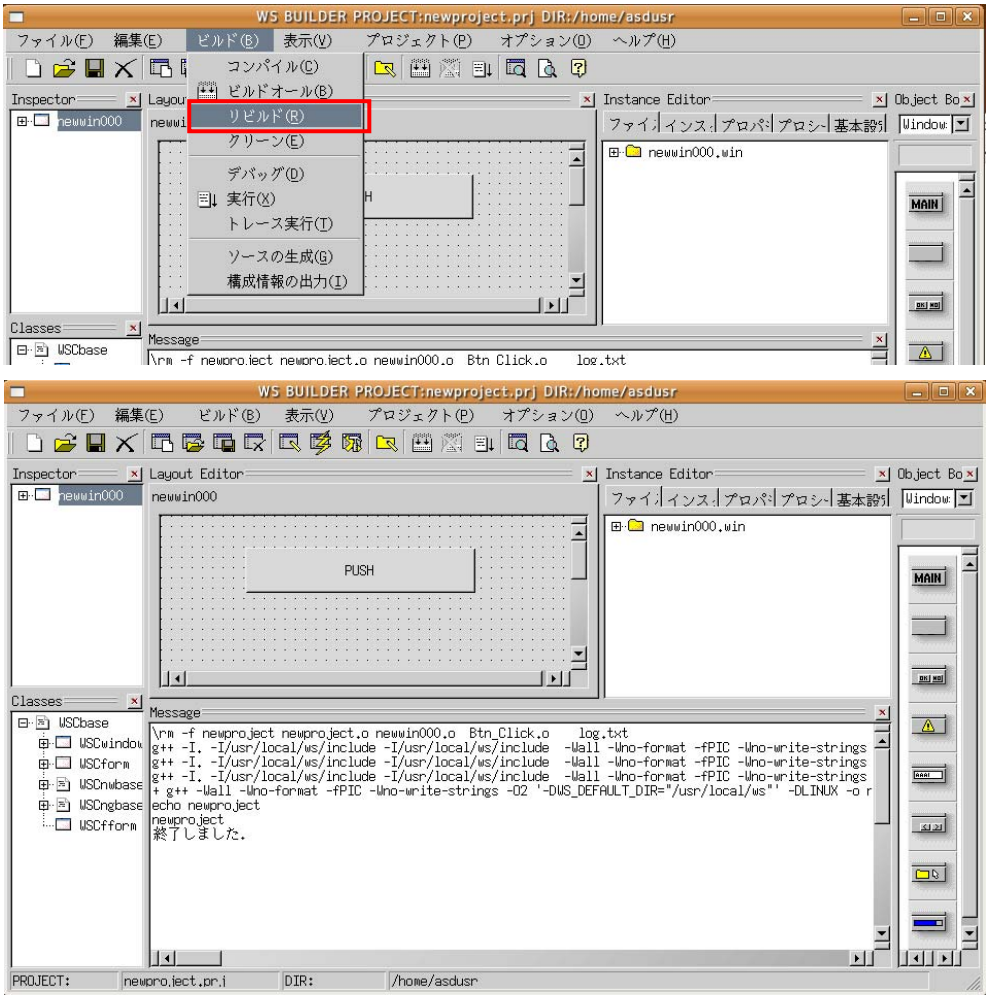


図 2-7-7-4. サンプルプロジェクトのコンパイル

コンパイルが完了したら、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 Ubuntu 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。

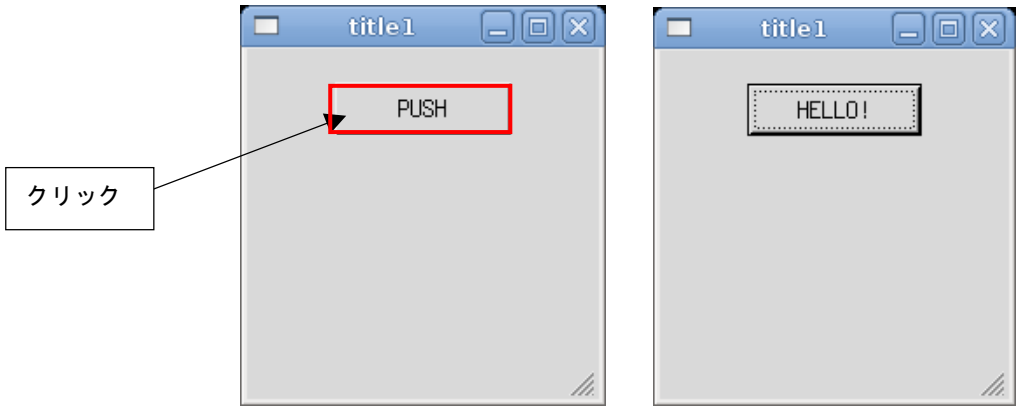


図 2-7-7-5. プログラム実行画面

動作を確認できたら、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。
今行ったのは、パソコン上でコンパイルしてパソコン上で実行したのでセルフコンパイルです。

２－７－８　クロスコンパイル

Algo Smart Panel 上で動作できる実行プログラムを作成するためにはクロスコンパイルをする必要があります。「２－７－７ セルフコンパイル」と同様の手順でプロジェクト設定ウィンドウの基本設定タブを開いてください。TARGET の項目のプルダウンメニューから「T-Engine ASP AP2000/3100/3102(T-Shell)」を選択してください。

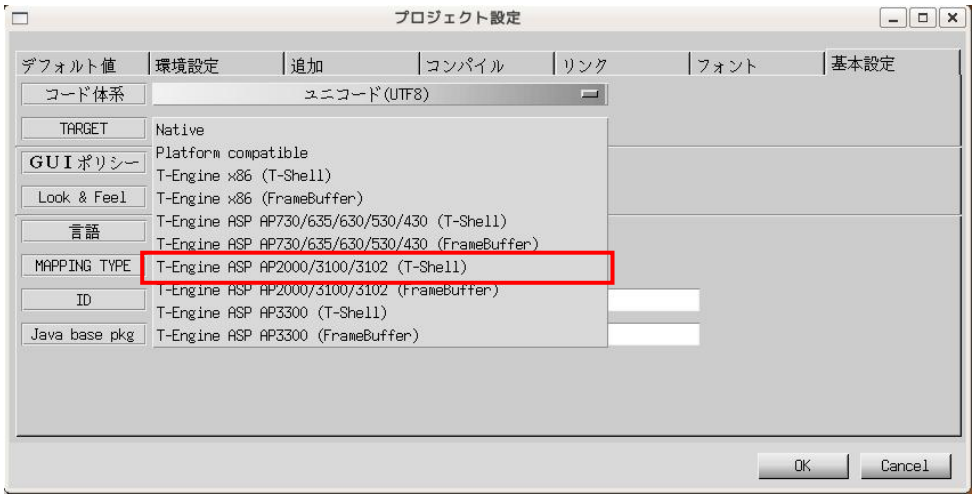


図 2-7-8-1. サンプルプロジェクトの Target の確認

AP-2000/3100/3102 用開発環境において TARGET のプルダウンメニューから選択できる項目は表 2-7-8-1 の 2 種類です。

表 2-7-8-1. AP-2000/3100/3102 開発環境で選択できるターゲット

ターゲット名	説明
Native	Linux PC 上で動作させるための設定
T-Engine ASP AP2000/3100/3102 (T-Shell)	PMC T-Kernel 上の T-Shell 上で動作させる場合の設定

※　これら以外のターゲットではコンパイルできません

また、「リンク」タブをクリックし「リンク方式」の項目の「スタティックリンク(DLL 無し)」にチェックを入れてください。その後、「OK」ボタンをクリックしてプロジェクト設定ウィンドウを閉じてください。

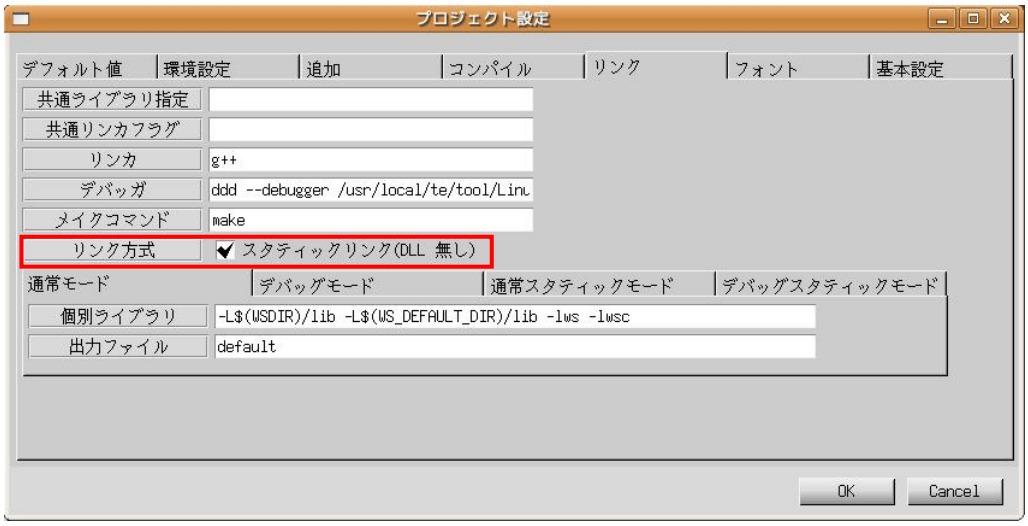


図 2-7-8-2.　プロジェクト設定ウィンドウ(リンクタブ)

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。

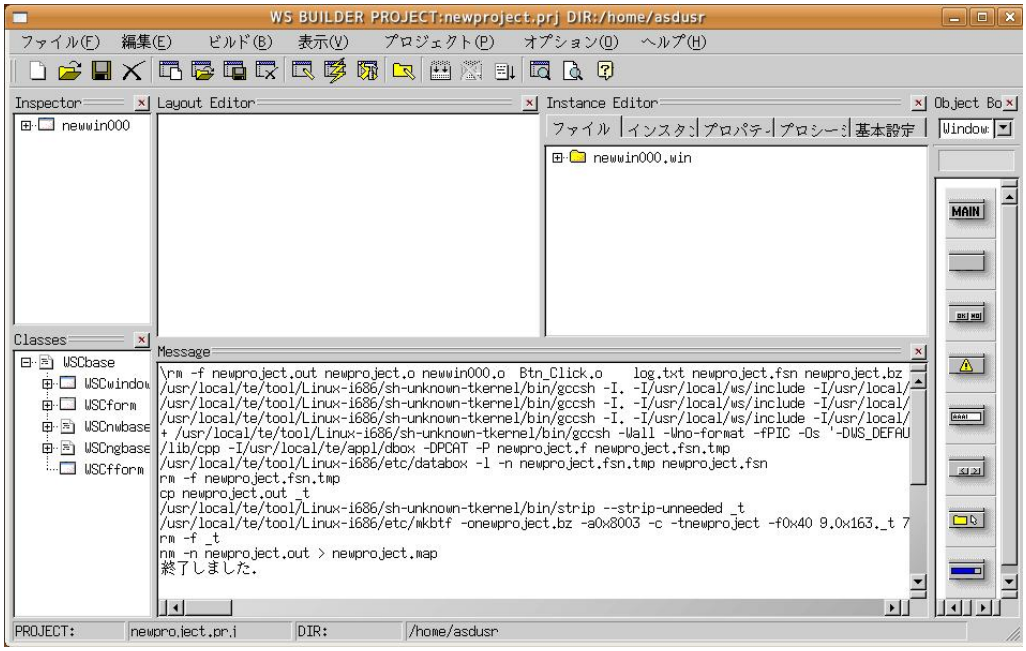


図 2-7-8-3.　サンプルプロジェクトのクロスコンパイル

図 2-7-7-4 のセルフコンパイルのコンパイルログと図 2-7-8-3 のクロスコンパイルのコンパイルログを比べると、セルフコンパイルで g++と表示されていた部分がクロスコンパイルでは /usr/local/te/tool/Linux-i686/sh-unknown-tkernel/bin/gccsh と表示されます。
コンパイルが完了すると newproject.out という実行ファイルと newproject.bz という圧縮ファイルが生成されます。クロスコンパイルによって生成されたファイルは Algo Smart Panel に転送することで使用ができます。

２－７－９　ファイルの転送

クロスコンパイルにて作成した実行プログラムを Algo Smart Panel に移して実行してみましょう。
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、Algo Smart Panel にデータを転送するには USB メモリでの転送とシリアルケーブルでの転送、ftp で転送の 3 種類の方法があります。

●USB メモリでの転送

① パソコンに USB メモリを挿入し、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。
※注：USB メモリは FAT32 フォーマットしておいてください。

② 『２－６ シリアルコンソール接続について』の手順に従い、Algo Smart Panel と VirtualBox (Ubuntu) をシリアルコンソール接続します。

③ 転送するデータが保存された USB メモリを Algo Smart Panel に挿入します。

④ 次に、USB メモリのマウントを行います。

```
[/SYS]% att -m uda0 uda0
```

⑤ 転送するデータを、Algo Smart Panel の「/SYS」にコピーします。
例) 転送するデータ　　：newproject.bz
　　転送先ディレクトリ　　：/SYS
以下のコマンドを入力することで、Algo Smart Panel 上に転送するデータが保存されます。

```
[/SYS]% ux/cp /uda0/newproject.bz .
```

⑥ マウントを解除して USB メモリを抜きます。

```
[/SYS]% det -u uda0
```

以上で USB メモリから Algo Smart Panel へのコピーが完了しました。

●シリアルケーブルでの転送

① 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」をクリックしコンソールを立ち上げます。転送したいファイルのあるディレクトリへ移動してください。
例) 転送するファイルのディレクトリ　:/homu/asdusr/sample
　　転送するファイル名　　:newproject.bz
　　転送先のディレクトリ　　:/SYS

```
$ cd /home/asdusr/sample
```

② 『２－６ シリアルコンソール接続について』の手順に従い、Algo Smart Panel と VirtualBox (Ubuntu) をシリアルコンソール接続します。

③ 以下のコマンドを入力することで Algo Smart Panel 上にデータが転送されます。

```
[/SYS]% recv -d newproject.bz
```

以上でシリアルケーブルを用いての Algo Smart Panel へのコピーが完了しました。

●LAN 経由で ftp 転送

PMC T-Kernel 開発環境には gFTP という ftp 転送用ソフトがインストールされています。

①　クロス LAN ケーブルで PC と Algo Smart Panel を接続します。
例)Algo Smart Panel の IP 設定：「192.168.0.1」
※注）Algo Smart Panel の IP 設定の変更方法については『２－７－１０ Algo Smart Panel 接続設定変更』を参照してください。

②　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」をクリックすると gFTP ソフトウェアが立ち上がります。Algo Smart Panel の IP とユーザー名、パスワードを入力し、ホスト左のパソコンマークを押下することで PC と Algo Smart Panel を ftp 接続することができます。
※注）Algo Smart Panel のユーザー名、パスワードは以下のように設定されています。
ユーザー名：asdusr
パスワード：asdusr

図 2-7-9-1.　プログラム実行画面

③　左側のアドレス入力欄に Ubuntu 上の転送したいファイルのある場所を入力し、右側のアドレス入力欄に Algo Smart Panel 上でファイルを送りたい場所を入力します。
例:転送したいファイル/home/asdusr/newproject.bz
　　転送したい先　　/SYS/newproject.bz

④　左側のファイル一覧から newproject.bz を選択し、→ボタンを押下します。右側のファイル一覧に newproject.bz が追加されることを確認してください。
以上で ftp 転送を用いての Algo Smart Panel へのコピーが完了しました。

・リモート(Algo Smart Panel)側のファイルが正常に表示されないとき
gFTP で T-Kernel をインストールしている Algo Smart Panel へ接続する場合、Algo Smart Panel 側のファイルが全て「不明」と表示され、正常に表示されない場合があります。この場合は以下の設定を行ってください。

① gFTP のメニューから「FTP(F) → オプション」を選択し、gFTP のメニューを開いてください

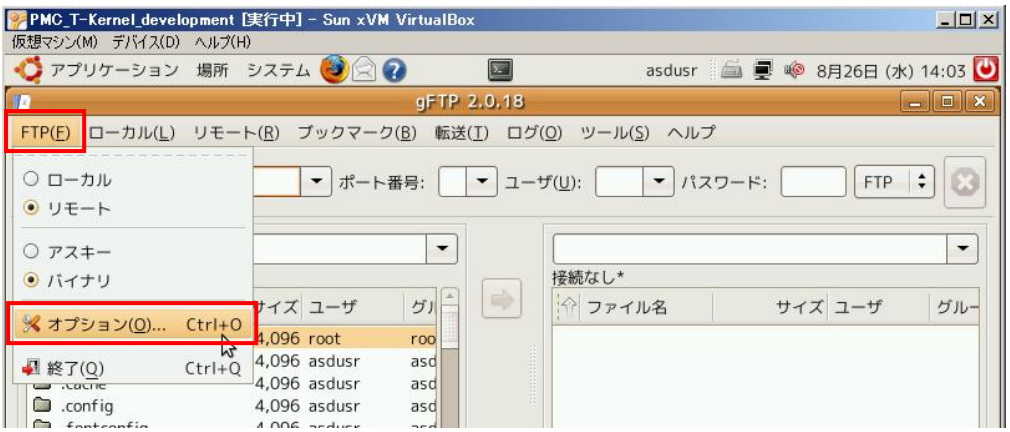


図 2-7-9-2. gFTP メニュー

② 「全般」タブの「隠しファイルを表示する」の項目のチェックが外れていることを確認してください。

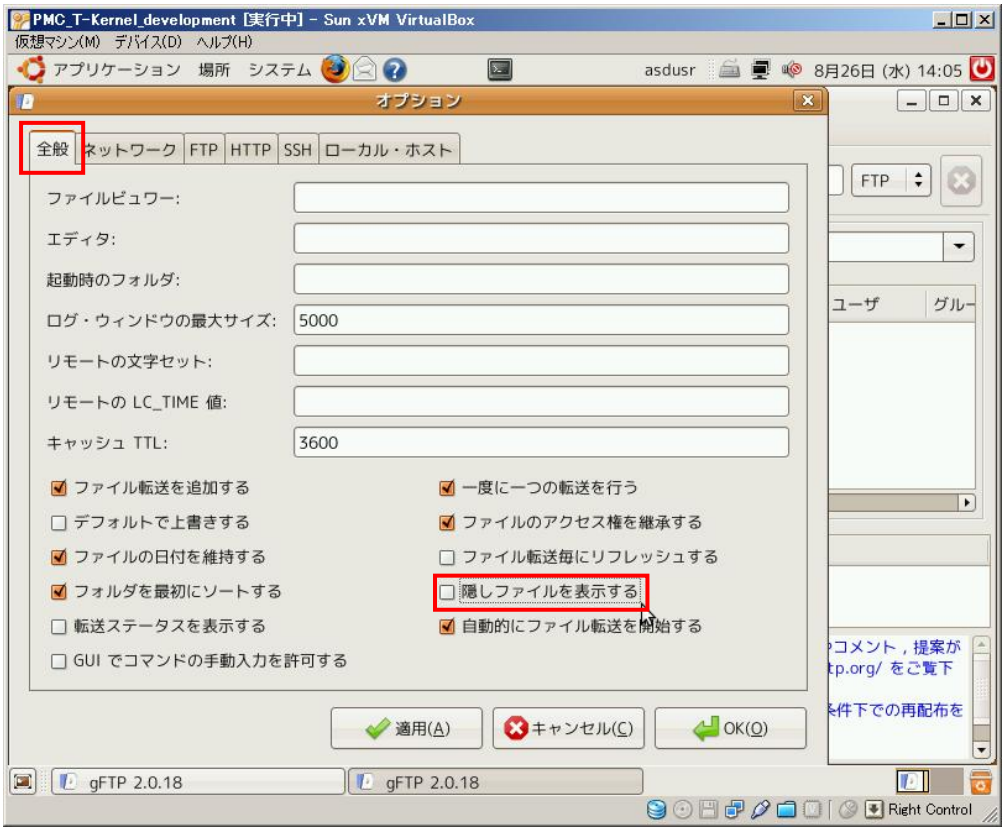


図 2-7-9-3. 全般タブ

③　「FTP」タブの「リモートのシンボリックリンクを有効にする」の項目のチェックが外れていることを確認してください。その後「OK」ボタンを押下してください。

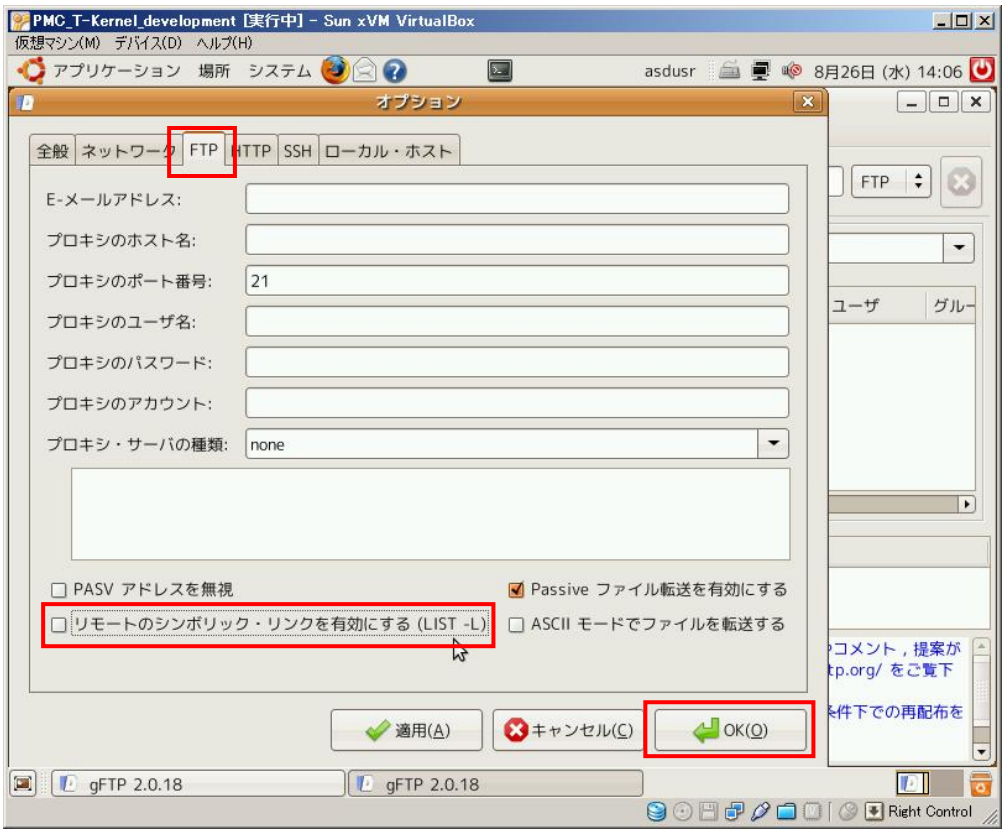


図 2-7-9-4. FTP タブ

転送した newproject.bz について

Algo Smart Panel へ転送した newproject.bz を実際に Algo Smart Panel 上で使用し、コンパイルとファイルの転送が成功していることを確認します。

例）実行ファイル　　：newproject.bz
　　実行ファイルのディレクトリ　：/SYS

以下のコマンドを実行し、newproject.bz を/SYS へ展開します。

```
[/SYS]% expf –v –r newproject.bz
```

次に以下のコマンドを実行し、「サンプルプログラム」という名目で Algo Smart Panel の小物メニューに newproject が登録します。

```
[/SYS]% vup –t newproject /SYS
```

小物メニューの「サンプルプログラム」をダブルクリックすることでアプリケーションを起動するようになったことを確認してください。
コンパイル時に生成される newproject.f を編集して、もう一度ビルドすることで登録される「サンプルプログラム」の題目を変更することができます。
最後に以下のコマンドを実行することで、不要になったファイルを削除します。

```
[/SYS]% rm newproject newproject.bz
```

以上で WideStudio/MWT による、アプリケーション開発の説明は終了です。

２－７－１０ Algo Smart Panelのネットワーク接続設定変更

Algo Smart Panel の IP は初期状態では 192.168.0.1 になっています。IP など接続設定を変更したい場合は以下の手順を実行してください。

① Algo Smart Panel にコンソール接続をします。
② 『２－６ シリアルコンソール接続について』の手順に従い、Algo Smart Panel と VirtualBox（Ubuntu）をシリアルコンソール接続します。
③ 以下のコマンドを入力します。

```
[/SYS]% netconf NETCONF
```

④ 以下の項目が順次表示されるので、変更する場合はその数値を、変更しない場合はそのまま Enter キーを入力してください。

```
hostname   = ?                    (デフォルトは asp)
host ip    = 0.0.0.0 ?             (デフォルトは 192.168.0.1)
dns1name   = ?
dns1 ip    = 0.0.0.0 ?
dns2name   = ?
dns2 ip    = 0.0.0.0 ?
domain     = ?
gateway ip = 0.0.0.0 ?
subnetmask = 0.0.0.0 ?             (デフォルトは 255.255.255.0)
wlan       = none (n/a/i)?
```

⑤ 以上で Algo Smart Panel の接続設定は完了です。以下のコマンドを入力することで、設定した情報の確認ができます。

```
[/SYS]% netconf
```

２－７－１１ WideStudio/MWTの開発例

ここでは、今まで開発したアプリケーションの中で使った方法について列挙していきます。アプリケーション開発の参考資料としてお使いください。

●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

Algo Smart Panel 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で列挙

２－８　　DDDについて

PMC T-Kernel 開発環境では、WideStudio の標準デバッグツールとして DDD を採用しています。DDD とは後述する GDB というコンソール用デバッグツールに GUI の殻をかぶせたものです。

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

WideStudio の場合、  をクリックするか、メニューの「プロジェクト(P)」→「プロジェクト設定(E)」をクリックしてください。「コンパイル」タブをクリックすると、図 2-8-1 のようなプロジェクト設定画面が表示されます。

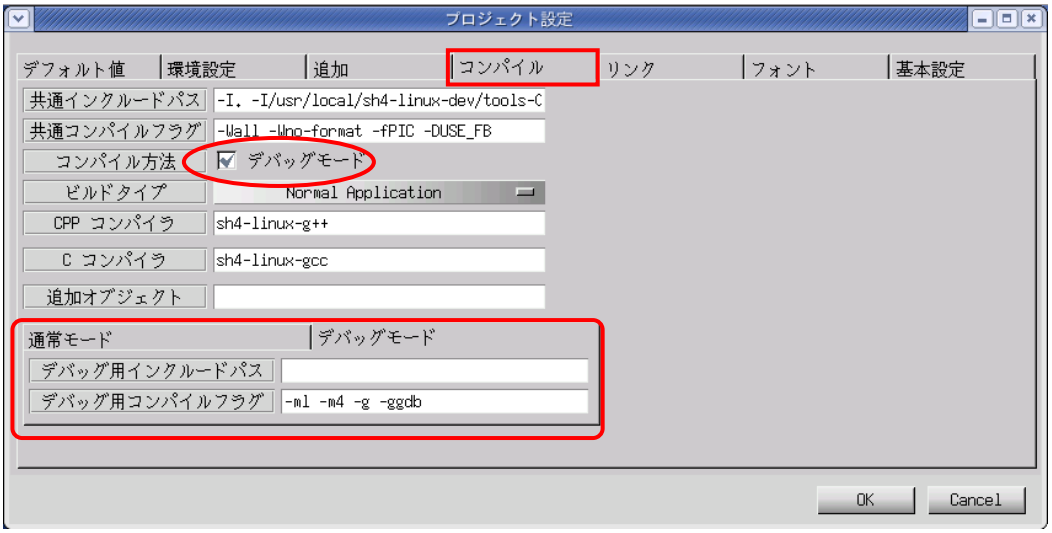


図 2-8-1. プロジェクト設定画面(コンパイルタブ)

「コンパイル方法」という項目の、「デバッグモード」というチェックボックスにチェックを入れます。これで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」という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。

また、デバッガとして gdb を認識させるために、リンクタブのデバッガ設定として「ddd –debugger /usr/local/te/tool/Linux-i686/bin/sh-unknown-tmonitor-gdb」と設定されていることも確認してください。「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直します。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.out」→「newprojectd.out」）。

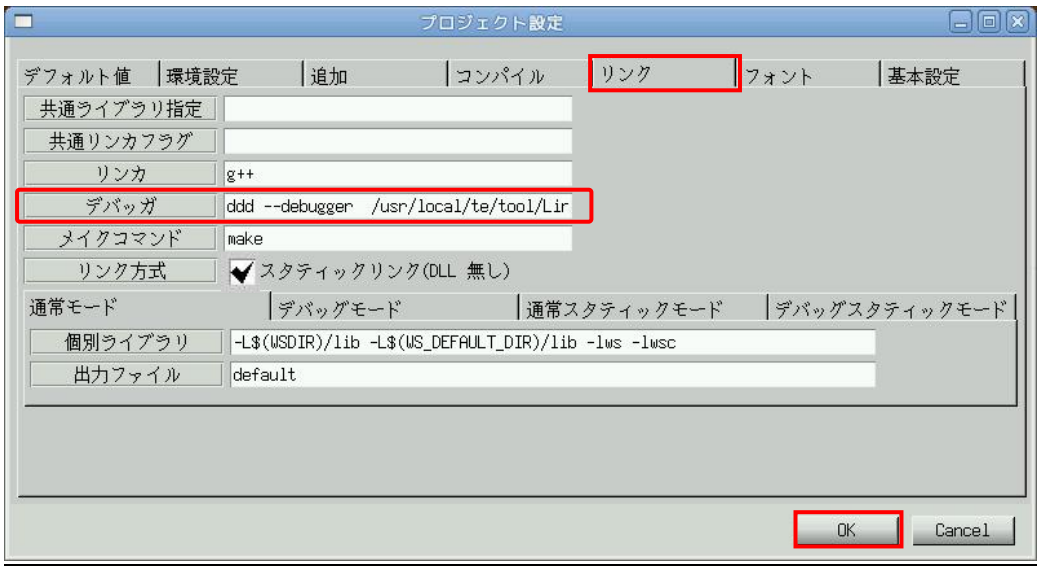


図 2-8-2. プロジェクト設定画面(リンクタブ)

① DDD の起動

リビルド実行後、ビルド→デバッグを実行すると、DDD が起動します。

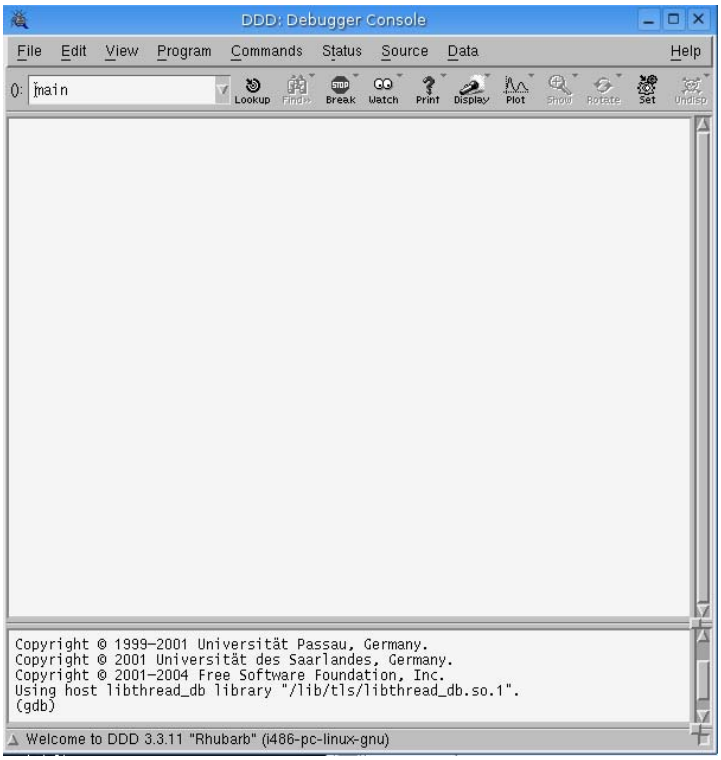


図 2-8-3. DDD 起動画面

DDDは最初、newprojectdというファイルを読み込みにいくためファイルメニューからOpen Programを選択し、実行ファイルnewprojectd.outを選択します

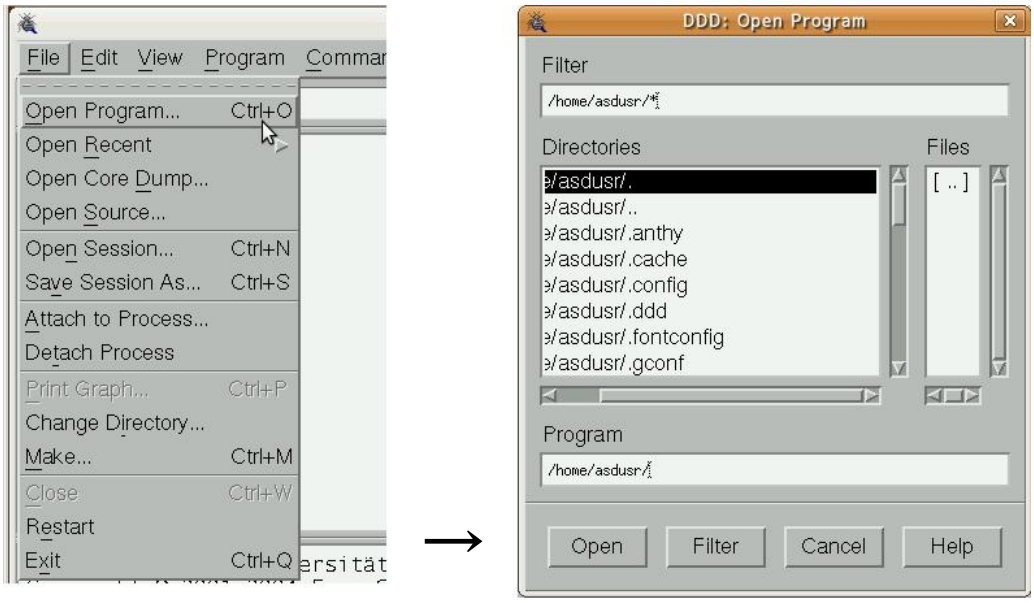


図 2-8-4 プログラムの選択

DDDでファイルメニューからOpen Sourceを選択してソースを選択します。

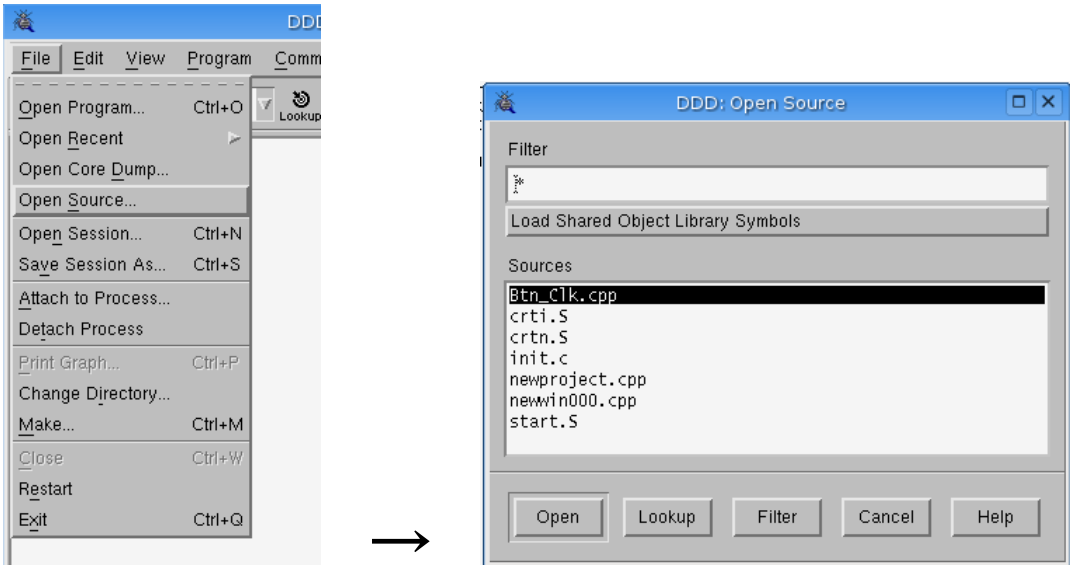


図 2-8-4. ソースの選択

② ASP 側プログラムの起動

Algo Smart Panel へデバッグ用実行プログラムを転送します。『3-2-9 ファイル転送』の「LAN 経由による ftp 転送」の項目を参考に、①でコンパイルした「newprojectd.out」を Algo Smart Panel に転送します。

次に以下のコマンドを実行し、デバッグモードでプログラムを起動し、T-Monitor に入ります。

```
[/SYS]% ref pld 1
[/SYS]% ref dbg 2
[/SYS]% debug newprojectd.out
TM>
```

コンソールを開放するために、一度端末を閉じます。

```
TM> .q
```

③ DDD の接続設定

DDD からコンソールを通して ASP へ接続します。この操作は DDD のウィンドウの下部にある入力欄に入力します。

以下のコマンドを入力します。

```
(gdb) target tmon_sh4 /dev/ttyUSB0
```

④　ブレークポイントの設定

ソースの中から、実行を一時的に止めたい場所にブレークポイントを設定します。
先ほど作成したサンプルプログラムで、ボタンをクリックされた時の処理をデバッグしたいなら、Btn_clk 関数のまえにカーソルを置いてブレークボタンをクリックします。

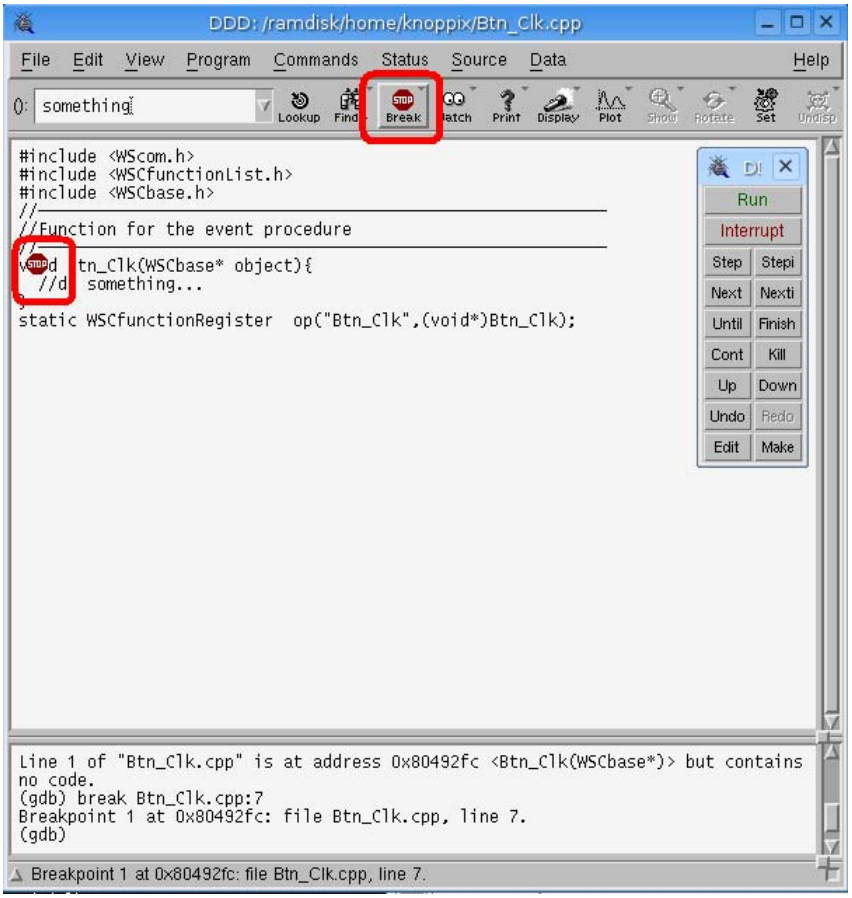


図 2-8-5.　ブレーク設定

ソース上に  アイコンがセットされます。

コマンドボタンウインドウ上の RUN ボタンをクリックするとプログラムが実行され、プログラムのボタンをクリックすると、  アイコンのところで実行が止まり、現在実行中の行が矢印で表示されます。

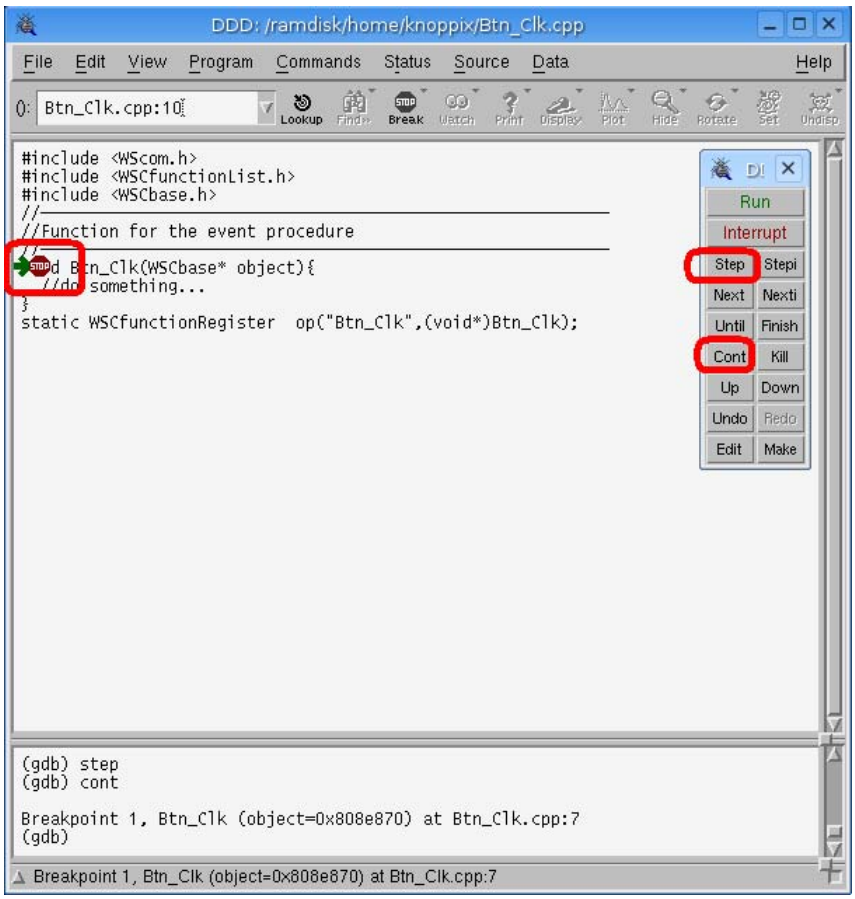


図 2-8-6. ブレークによる実行停止

⑤　ブレークポイントでの変数の内容表示

ソースウィンドウで、表示させたい変数名をドラッグして選択反転させ、そこで右クリックして、"Print 変数名"アイテムを選ぶとコマンドラインに変数の中身が表示さます。
"Display 変数名"アイテムを選択すると、変数表示領域に変数の内容が表示され、break point で停止した時点での変数の内容を常に表示してくれます。
また、ポインタ変数の場合、"Print *変数名"及び"Display *変数名"のアイテムを選択すると、ポインタ変数が示す先(メモリ番地)の内容を表示してくれます。

⑥　一旦停止時からの操作

cont ボタン：　Continue 操作を意味し、次に break point に到達するまで実行続行します。
next ボタン：　一行ずつ実行します。ただし関数呼出し時は、関数内部のコードは表示せずに関数の実行を完了させて、その次の行に制御が移ります。(コマンドラインから next 数値とすると、その数値の行数分をまとめて実行します)。
step ボタン：　一行ずつ実行します。関数呼出し時には関数内のコードも一行ずつ実行します(コマンドラインから step 数値とすると、その数値の行数分をまとめて実行します)。

⑦　ブレークポイントの解除

ブレークポイントを表す  アイコンを右クリックして Delete Breakpoint を選択すると、その breakpoint は解除されます。一時的に解除したい場合は、Disable Breakpoint を選択し、一時解除していた breakpoint を再開するには、Enable Breakpoint を選択します。

２－９　　WideStudioを使用しないコンソールアプリケーション開発

WideStudio の TARGET を T-Kernel にした場合「コンソールアプリケーション」の開発ができません。
T-Kernel 上で動作するコンソールアプリケーションを作成する場合は Makefile と main 関数を作成し、開発環境上コンソールでコンパイルする必要がありますが、開発環境内の「/usr/local/te-asd_sh7760/util/sample/sample19」に、コンソールアプリケーション作成用雛形を収録しています。
本章では T-Kernel 上でコンソールアプリケーションを作成する手順を説明します。

※　本書のコンソールアプリケーション開発についての説明は基本的なコンパイル手順のみにさせていただきます。より詳しい Makefile やソースファイルの作成方法については専門書やインターネットなどを参考にしてください。

２－９－１　　ディレクトリ構成

「/home/asdusr/sample_console」ディレクトリにサンプルプログラムを作成すると仮定します。
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」をクリックしコンソールを立ち上げます。以下のコマンドを実行し、「/usr/local/te-asd_sh7760/util/sample/sample19」のディレクトリをコピーしてください。

```
$ cp –a /usr/local/te-asd_sh7760/util/sample/sample19 /home/asdusr/sample_console
```

正常にコピーが完了すると表 2-9-1-1 のようなディレクトリ構成になります。

表 2-9-1-1. コンソールプログラム作成時のディレクトリ構成

```
home--+--asdusr--+--sample_console--+--src
                                    +--asd_sh7760
```

表 2-9-1-2. ディレクトリ内容

ディレクトリ名	内容
/home/asdusr/sample_console/src	source file と Makefile を格納しています。
/home/asdusr/sample_console/asd_sh7760	プログラムのコンパイル成果物が格納されるディレクトリです。../src/Makefile へのリンクが格納されており、このディレクトリ上でコンパイルを実行します。

２－９－２　ソースファイルの作成

プログラムのソースファイルを作成します。
簡単なサンプルとして、アプリケーションを起動するとコンソールへ「Hello」と出力し、アプリケーションを終了するプログラムを作成します。
「/home/asdusr/sample_console/main.c」を編集します。

リスト 2-9-2-1. コンソールアプリケーション雛形 main.c

```
/*
    Console Aplication Sample
*/
#include    <btron/outer.h>

EXPORT  W   main(W ac, B *av[])
{
//ここにプログラムを記述してください
    return 0;
}
```

main 関数部を以下のように書き換えてください。

リスト 2-9-2-2. コンソールアプリケーション main 関数

```
EXPORT  W   main(W ac, B *av[])
{
    printf("Hello¥n");
    return 0;
}
```

２－９－３　Makefileの作成

プログラムの Makefile を作成します。

「/home/asdusr/sample_console/Makefile」を編集します。

リスト 2-9-3-1. コンソールアプリケーション雛形 Makefile

```
#   @(#)Makefile
#       GNU make 用
#

# ソースの依存関係ファイル（自動生成）
DEPS = Dependencies
DEPENDENCIES_OUTPUT := $(DEPS)

# 標準ルール
ifdef BD
  include $(BD)/util/etc/makerules
else
  include /usr/local/te/util/etc/makerules
endif

#------------------------------------------------------------------------
TARGET = sample_console                                                    ①

# ソースファイルのサーチパス
S = ../src
VPATH = $S

HEADER := $(S) $(HEADER)

# ソースファイル
SRC := main.c                                                              ②

OBJ = $(addsuffix .o, $(basename $(SRC)))
SRC.C   = $(strip $(patsubst %.C, %.c, $(filter %.C, $(SRC))))
OUT = $(addsuffix .out, $(TARGET))

HEADER := $(S) $(HEADER)

CFLAGS += $(CFLAGS_WARNING)

# 追加ライブラリ
LOADLIBES = -lux -lapp

#------------------------------------------------------------------------
.PHONY: all install clean

ALL = $(OUT) $(addsuffix .map, $(TARGET))

all: $(ALL)
```

```
%.map: %

$(OUT): $(OBJ)
    $(LINK.o) $(LDOBJS) $^ $(LOADLIBES) $(LDLIBS) $(OUTPUT_OPTION)

clean:
    $(RM) $(OUT) $(OBJ) $(SRC.C) $(ALL) $(DEPS)

install: $(addprefix $(EXE_INSTALLDIR)/, $(TARGET))

# ソースの依存関係
ifdef DEPENDENCIES_OUTPUT
  $(DEPS): ; touch $(DEPS)
else
  $(DEPS): $(SRC) ; $(MAKEDEPS) $@ $?
endif
include $(DEPS)

$(SRC.C):
$(OBJ):
$(OUT):
```

① 生成される実行ファイル名を指定します。本章では「sample_console」というファイル名で生成します。

② ソースファイル名をここに記述します。ソースファイルを複数使用する場合は、ここにソースファイル名をスペースで区切って追記してください。

２－９－４　コンパイル方法

作成したプログラムのコンパイルを行います。
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」をクリックしコンソールを立ち上げます。以下のコマンドを実行し、asd_sh7760 ディレクトリへ移動します。

```
$ cd /home/asdusr/sample_console/asd_sh7760
```

開発環境へパスを通します。

```
$ . /usr/local/te-asd_sh7760/te.sh
```

Make を行います。

```
$ make clean
$ make
```

asd_sh7760 ディレクトリ内に sample_console.out という実行ファイルが作成されていることを確認してください。
作成された実行ファイルは WideStudio で作成したアプリケーションと同様に T-Kernel 上で扱うことができます。
『２－７－９ ファイルの転送』を参考にして ASP へ実行ファイルを転送し、コンソールから実行してください。
「Hello」とコンソールに表示されればコンパイルは成功です。

２－１０　T-Kernel版WideStudioコンポーネント一覧

WideStudio には様々なコンポーネントが用意されていますが、T-Kernel 版 WideStudio では使用できないものや、T-Kernel 環境上で正常に動作しないものがあります。
WideStudio に用意されているコンポーネントの、T-Kernel 環境上での使用の可否を表 2-10-1 に示します。

表 2-10-1 WideStudio で用意されているコンポーネントの T-Kernel 環境上での使用の可否一覧

項目	コンポーネント名	T-Kernel での動作	備考
windows	WSCmainWindow	○	
	WSCwWindow	○	
	WSCdialog	○	
	WSCmessageDialog	○	
	WSCinputDialog	○	
	WSCwizardDialog	○	
	WSCfileselect	○	
	WSCworkingDialog	○	
forms	WSCform	○	
	WSCindexForm	○	
	WSCsform	○	
	WSCscrForm	○	
	WSCradioGroup	○	
	WSCcheckGroup	○	
	WSCvertForm	×	コンパイル不可
	WSChorzForm	×	コンパイル不可
	WSCmenuArea	○	
	WSCtform	○	
	WSCfform	×	コンパイル不可
	WSCprform	×	コンパイル不可
	WSCj3wform	×	コンパイル不可
	WSCopenglForm	×	貼り付け不可
commands	WSCvbtn	○	
	WSCvfbtn	○	
	WSCvtoggle	○	
	WSCradio	○	
	WSClabel	○	
	WSCvfklabel	○	
	WSCvkslabel	○	
	WSCvifield	○	
	WSCvmifield	○	
	WSCvpifield	○	
	WSCtextfield	○	
	WSCvarrow	○	
	WSCvscrBar	○	
	WSCclock	○	
	WSCvslider	○	

項目	コンポーネント名	T-Kernel での動作	備考
commands	WSCpulldownMenu	○	
	WSCcomboBox	○	
	WSCoption	○	
	WSClist	○	
	WSCtreelist	○	
	WSCdirtree	○	
	WSCverblist	○	
	WSCgrid	○	
	WSCimage	○	
	WSCsheet	×	貼り付け不可
	WSCvendbtn	○	
Drawing	WSCvrect	○	
	WSCvarc	○	
	WSCpoly	○	
	WSCvline	○	
	WSCdrawingArea	○	
	WSCvbarGraph	○	
	WSCvlineGraph	○	
	WSCvgraphScale	○	
	WSCvgraphMatrix	○	
NonGUI	WSCngbase	○	
	WSCvtimer	○	
	WSCvspace	×	T-Kernel 上で正常に動作しない (スペースを確保しない)
	WSCvballoonHelp	○	
	WSCvcsocket	○	
	WSCvssocket	○	
	WSCvmultiServerSocket	×	コンパイル不可
	WSCvudpsocket	○	
	WSCvremoteClient	×	T-Kernel 上で正常に動作しない(通信できない)
	WSCvremoteServer	×	T-Kernel 上で正常に動作しない(通信できない)
	WSCvdb	×	コンパイル不可
	WSCvodbc	×	貼り付け不可
	WSCvdbDataSource	×	コンパイル不可
	WSCvthread	○	

第３章 Algo Smart Panelについて

本章では、Algo Smart Panel 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の作成手法について説明しています。

DVD-ROM に格納されているサンプルソースの一覧を表 3-1 に示します。

※　**サンプルソースは開発環境内の「/usr/local/te-asd_sh7760/util/sample」に格納されています。**
※　**サンプルソースを任意のディレクトリで使用したい場合は、T-Kernel 用パッケージ DVD-ROM にある「g1t1-sample-1.**.tar.gz」を展開してご使用ください(**はバージョン番号)。展開後生成される「g1t1-sample」というディレクトリに各 Sample が含まれています。**

表 3-1. サンプルソース一覧

フォルダ名	内容
Sample01_dio	汎用入出力制御方法
Sample02_serial	シリアルポート制御方法
Sample03_network	ネットワーク通信(LAN)制御方法
Sample04_audio	オーディオデバイス制御方法
Sample05_launcher	起動ランチャーサンプルプログラム
Sample07_wdt	ウォッチドッグタイマ制御方法
Sample08_rasin	汎用入力 IN0 リセット制御方法
Sample09_rasin	汎用出力 IN1 割込み制御方法
Sample10_rasram	バックアップ SRAM 制御方法 (read/write)
Sample17	T-Kernel ベースサンプル
Sample18	ドライバ開発サンプル　- SDI/GDI
Sample19	コンソールアプリケーション雛形
Sample20_backlight	バックライト輝度調整サンプル
Sample21_buzzer	タッチブザー調整サンプル

※　**IN0 リセット、IN1 割込み、バックアップ SRAM の機能は AP-3102 のみのサポートとなります。**

３－１　Algo Smart Panelのデバイスについて

固有デバイスの説明の前に、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。

３－１－１　デバイスドライバについての注意点

デバイスへアクセスするにはデバイスファイルを指定してシステムコール(tk_opn_dev、tk_cls_dev、tk_srea_dev、tk_swri_dev 等)でアクセスすることで行います。一般的な使用方法は、デバイスファイルをオープンし、リード／ライトすることで制御するということになりますが、実際にはデバイスごとに動作方法を設定する必要があります。例えば、「tk_srea_dev」というシステムコールを実行したとき、遅延を行わずにデータがある場合はそのデータをリターンし、無ければエラーをリターンする場合と、なんらかのデータが入ってくるまで遅延し、イベントが発生したらリターンする場合、イベントが発生するまでの遅延をタイムアウトで抜ける場合等があるので、どのモードで動作するかを指定する必要があります。
デバイスごと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、Algo Smart Panel 用に作成したデバイスの仕様について説明します。
※　AP-3102 については現在 USB 未サポートになっています。

３－１－２　デバイスドライバのロード方法

DVD に収録されているサンプルプログラムが用いるデバイスドライバは初期状態でロードされています。新しく追加したドライバや T-Kernel ベースのプログラムを使用するには、それらのデバイスドライバや実行ファイルをロードする必要があります。デバイスをロードする方法として、以下の二つがあります。

・使用する度にデバイスをロードする
デバイスを使用する前に手動でロードを行う方法です。
①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」をクリックしコンソールを立ち上げ、以下のコマンドを実行して T-Kernel にログインしてください。

```
# /usr/local/te/tool/Linux-i686/etc/gterm –b –l /dev/ttyUSB0
```

②デバイスロードコマンドを実行してください。

```
[/SYS] lodspg <デバイス名>  !<優先度>
```

<デバイス名>にはロードしたいデバイスや実行ファイルの名前を入力してください。<優先度>には起動時の優先度を入力してください。優先度の設定は省略することもできます。優先度を設定しなかったデバイスはデフォルトの優先度で起動します。

・起動時に自動でデバイスをロードする
スタートアップ設定ファイルにロードしたいデバイスを登録し、T-Kernel 起動時に毎回デバイスをロードするようにする方法です。スタートアップ設定ファイルについては『３－１２　設定ファイルについて』を参照してください。
デバイスを起動時にロードしたい場合は/SYS/STARTUP.CMD ファイルに追記します。

リスト 3-1-2-1. アプリケーション自動起動(IMS)設定ファイル

```
lodspg   screen     !35
lodspg   kbpd       !30
lodspg   lowkbpd    !28
lodspg   rsdrv      !26
lodspg   netdrv     !23
lodspg   unixemu
lodspg   tcpipmgr
lodspg   font       !120
lodspg   dp
lodspg   tip
lodspg   hmi
lodspg   omgr
lodspg   <デバイス名>   !<優先度>
$$sysdmn            !112 &
$logon
$cli    STARTUP.CLI !224
$logon  E
exit    1
```

<デバイス名>にはロードしたいデバイスドライバの名前を入力してください。
<優先度>には起動時の優先度を入力してください。優先度の設定は省略することもできます。優先度を設定しなかったデバイスはデフォルトの優先度で起動します。
ファイルに書き込んだ後、T-Kernel を再起動することで、追記したデバイスドライバが起動時に自動でロードされるようになります。
優先度に関する詳しい説明は『１－２－４ T-Kernel Standard Extension』を参照してください。

※本開発環境で用意されているデバイスドライバの名前、機能、対応するサンプルプログラムは表 3-1-2-1 のようになります。

表 3-1-2-1 デバイスドライバ対応表

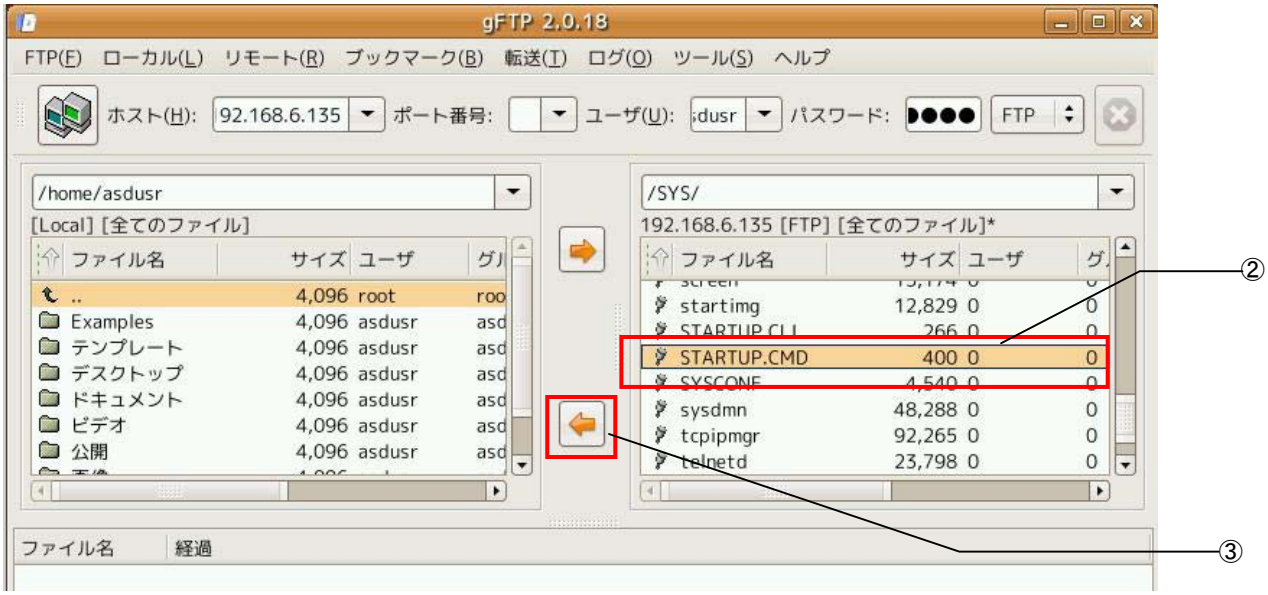
デバイス名	機能	対応する Sample
rsa	デバッグポート	-
rsb	シリアルポート	Sample02
diodrv	汎用入出力	Sample01
audio	音声入出力	Sample04
wdt	ウォッチドッグタイマ	Sample07
rasin	汎用入力 IN0 リセット/IN1 割込み	Sample08/Sample09
rasram	バックアップ SRAM	Sample10
bz	タッチブザー	Sample21

※STARTUP.CMD を編集するにあたって、T-Kernel 環境ではコンソールを用いてファイルの編集をすることができません。したがって、STARTUP.CMD を編集するにはファイルを一度開発環境に吸い上げ、エディタなどで編集してから再び Algo Smart Panel 上の/SYS/STARTUP.CMD に上書きする必要があります。以下にその方法を示します。

①Algo Smart Panel へ LAN 経由で ftp 接続を行います。ftp 接続の方法については『２－７－９ ファイルの転送』の『LAN 経由での ftp 接続について』を参照してください。

②右側のファイル一覧から STARTUP.CMD を選択します。

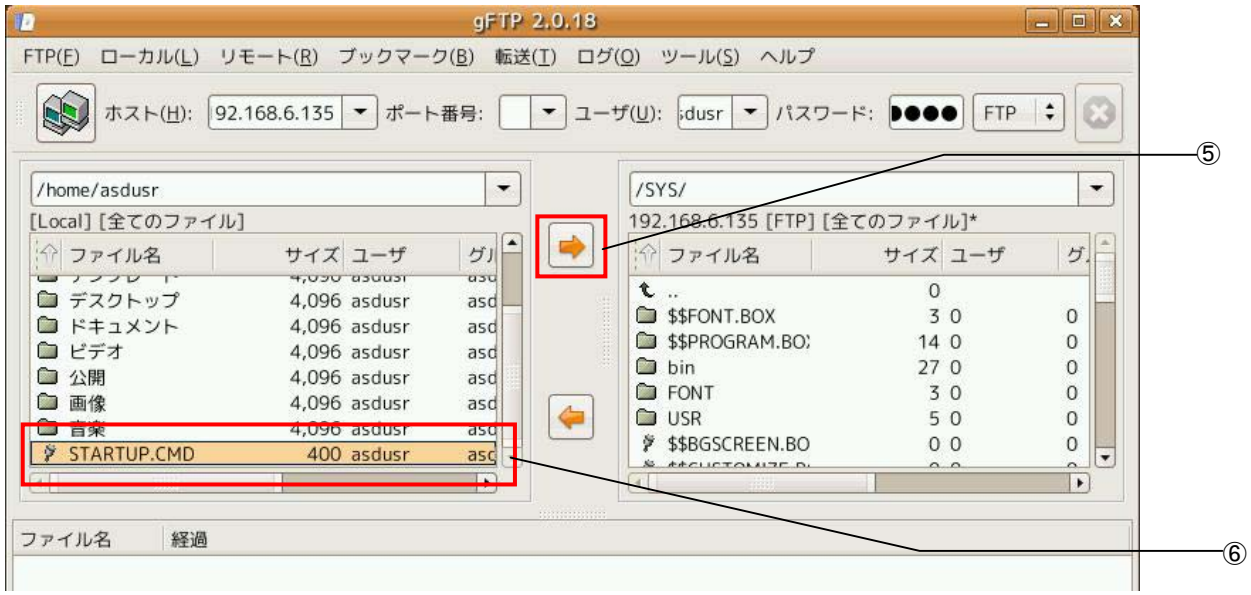
③中央の←ボタンを押下します。



④開発環境側に STARTUP.CMD が吸い上げられるので、gedit などのエディターでファイルを開き、３－２ページの説明に従って、STARTUP.CMD を編集します。

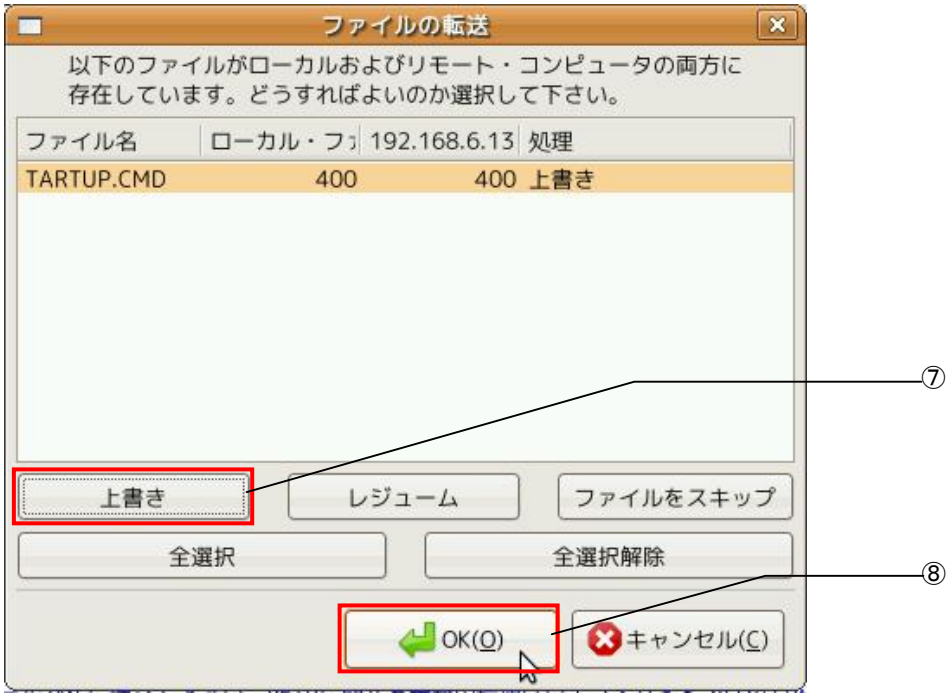
⑤編集した STARTUP.CMD を Algo Smart Panel に転送します。再度 gFTP を起動し、Algo Smart Panel と接続してください。接続後、左のファイル一覧から STARTUP.CMD を選択ます。

⑥中央の→ボタンを押下します。



⑦ファイルを上書きするかどうかの確認ダイアログが開くので、『上書き』ボタンを押下します。

⑧『OK』ボタンを押下してください。



このとき、元の STARTUP.CMD は上書きされてしまうので、必要に応じてバックアップをとるようにしてください。

⑨Algo Smart Panel を再起動することで、設定したデバイスドライバを起動時にロードするようになります。

３－２　　汎用入出力ドライバ

Algo Smart Panel では出力４点、入力６点を制御することができます。これらの入出力は、MAIN 基板の汎用入出力を対象としたデバイスドライバがあります。

３－２－１　デバイス詳細

・デバイス名

デバイス名は、”diodrv”を使用します。

・固有機能

汎用入出力からのデータの入出力機能です。

・属性データ

汎用入出力ドライバは以下の属性データをサポートしています。

```
typedef enum {
    DN_DIO_INPUTL     = -100,     /* LOW バイト 入力   data: UB*          R */
    DN_DIO_INPUTH     = -101,     /* HIGH バイト 入力  data: UB*          R */
    DN_DIO_OUTPUTL    = -102,     /* 未使用                               W */
    DN_DIO_OUTPUTH    = -103,     /* HIGH バイト 出力  data: UB           W */
    DN_DIO_CHGSEL     = -104,     /* 未使用                               W */
} SensorDataNo;
```

① DN_DIO_INPUTL：入力信号を読み出すことができます。
Data：UB*

② DN_DIO_INPUTH：出力信号を読み出すことができます。
Data：UB*

③ DN_DIO_OUTPUTL：未使用です。

④ DN_DIO_OUTPUTH：出力信号を書き込むことができます。
Data：UB

⑤ DN_DIO_CHGSEL：未使用です。

・固有データ

なし

・エラーコード

T-Kernel 仕様書のデバイス管理機能

３－２－２　サンプルプログラム

アクセス方法はキャラクタデバイス方式で入出力の状態を読書きします。
「sample01_dio」に、汎用入出力を使ったサンプルソースが入っています。

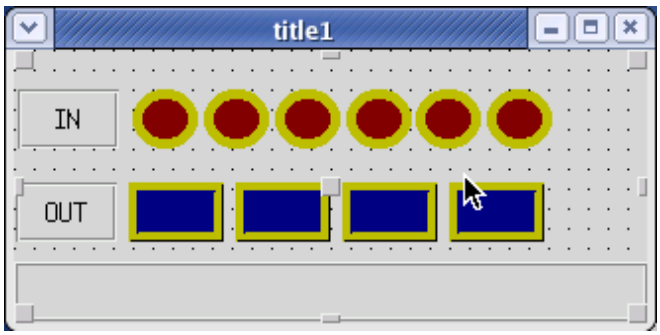


図 3-2-2-1. 汎用入出力デバイス制御サンプル画面

このサンプルは、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 の色を変化させています。また、OUT のボタンをクリックすると汎用出力が 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 3-2-2-1 に示します。

リスト 3-2-2-1. 汎用入出力のオープンとスレッドの生成

```
#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 < device/asd_sh7760/dio.h>

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

#include <basic.h>
#include <btron/tkcall.h>
#include <btron/outer.h>

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

#define DEVNAME (UB *)("diodrv")

//-------------------------------------------------------------
//Function for the event procedure
//-------------------------------------------------------------
void Main_Init(WSCbase* object){
  UB data;
  W  len=0;
  ER err;
```

```
        /*汎用出力デバイスのオープン*/
    dio_fd = tk_opn_dev(DEVNAME, TD_UPDATE);        //汎用出力デバイスオープン
    if(dio_fd < 0){                                 //エラー処理
        printf("diodrv open error¥n");
        Status_Bar->setProperty(WSNlabelString,"diodrv open error");
        tk_cls_dev(dio_fd, 0);
        return;
    }
    data = 0;
    err = tk_swri_dev(dio_fd, DN_DIO_OUTPUTH, &data, sizeof(UB), &len);//初期０出力
    if( (err!=E_OK) || (len!=sizeof(UB)) ) {
        printf("tk_swri_dev error : DN_DIO_OUTPUTH¥n");
        Status_Bar->setProperty(WSNlabelString, "tk_swri_dev error : DN_DIO_OUTPUTH");
        tk_cls_dev(dio_fd, 0);
        return;
    }
        /*スレッドの生成*/
    ioctrl_thr = 0;
    ioctrl_thr = WSDthread::getNewInstance();           //スレッドインスタンス取得
    ioctrl_thr->setFunction(Ioctrl_Thread);             //スレッド本体関数を設定
    ioctrl_thr->setCallbackFunction(Io_callback_func);  //コールバック関数を設定
    ioctrl_thr->createThread((void*)0);                 //スレッドを生成
}
static WSCfunctionRegister  op("Main_Init",(void*)Main_Init);
```

まず、「tk_opn_dev」関数で「diodrv」を開きます。汎用入出力は TD_UPDATE 設定で、リード／ライトすることができます。
汎用入力の状態を見るために、スレッドを生成します。リスト 3-2-2-2 に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。

リスト 3-2-2-2. 汎用入力リードスレッドのソースコード

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <errno.h>
#include < device/asd_sh7760/dio.h>

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

#include <basic.h>
#include <btron/tkcall.h>
#include <btron/outer.h>

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

```
ID dio_fd;
WSDthread* ioctrl_thr;
UB o_data;
UB f_thread_end=0;

void *Ioctrl_Thread(WSDthread* obj, void *arg)
{
  W  len;
  UB data;
  o_data = data = 0;
  ER err;

  for(;;){
      err = tk_srea_dev(dio_fd, DN_DIO_INPUTL, &data, sizeof(UB), &len);/*汎用入力読出し
*/
      data &= 0x3F;
      if((err >= E_OK) || (sizeof(UB) == len)){
          if(o_data != data){
              o_data = data;
              obj->execCallback((void*)(&data));
          }
      }
      else{
          printf("tk_srea_dev error : DN_DIO_INPUTL¥n");
          Status_Bar->setProperty(WSNlabelString,"tk_srea_dev error : DN_DIO_INPUTL");
          return(NULL);
      }
      tk_dly_tsk(100);
      //end ボタンが押された場合ループを抜け、終了キーを送る
      if ( f_thread_end==1 ) break;
  }

  f_thread_end = 2;
  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 Io_callback_func(WSDthread *, void *val)
{
  UB data;

  data = *(UB *)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");
    }
  }
```

汎用入力デバイスを読出すとき、1Byte 読出すことができます。汎用入力の「tk_srea_dev」関数は遅延せずに、汎用入力の ON/OFF 状態を即リターンします。サンプルソースでは、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。

ボタンの「ACTIVATE」プロシージャで、汎用出力の ON/OFF を行っています。リスト 3-2-2-3 に汎用出力の制御ソースコードを示します。

リスト 3-2-2-3. 汎用出力 ON/OFF 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 <fcntl.h>
  #include <unistd.h>
  #include <errno.h>
  #include <device/asd_sh7760/dio.h>

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

  #include <basic.h>
```

```
#include <btron/tkcall.h>
#include <btron/outer.h>

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

W btn[4]={0,0,0,0};
//-------------------------------------------------------------
//Function for the event procedure
//-------------------------------------------------------------
void Btn_Click(WSCbase* object){
  UB data;
  W code;
  W len;
  ER err;

  code = object->getProperty(WSNuserValue);                    //押されたボタン番号を取得
  err = tk_srea_dev(dio_fd, DN_DIO_INPUTH, &data, sizeof(UB), &len);    //汎用出力の出力値取得
  if((err >= E_OK) || (sizeof(UB) == len)){
      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;
      }
      err = tk_swri_dev(dio_fd, DN_DIO_OUTPUTH, &data, sizeof(UB), &len);//汎用出力更新
      if((err < E_OK) || (sizeof(UB) != len)){
          printf("tk_swri_dev error : DN_DIO_OUTPUTH¥n");
          Status_Bar->setProperty(WSNlabelString,"tk_swri_dev error : DN_DIO_OUTPUTH");
          return;
      }
  }
  else{
      printf("tk_srea_dev error : DN_DIO_INPUTH¥n");
      Status_Bar->setProperty(WSNlabelString,"tk_srea_dev error : DN_DIO_INPUTH");
      return;
  }
}
static WSCfunctionRegister  op("Btn_Click",(void*)Btn_Click);
```

ボタンが押されたら、現在の状態から ON/OFF を切替えます。「tk_srea_dev」関数で現在の状態を読込み、新しい状態に変更し、「tk_swri_dev」関数でデータを更新します。

３－３　　シリアルポートドライバ

３－３－１　　シリアルポートについて

Algo Smart Panel に実装されているシリアルポートの使用例について記述します。
本体のケース外にでているシリアルポートは、RS232C が使用することができます。表 3-3-1-1 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。

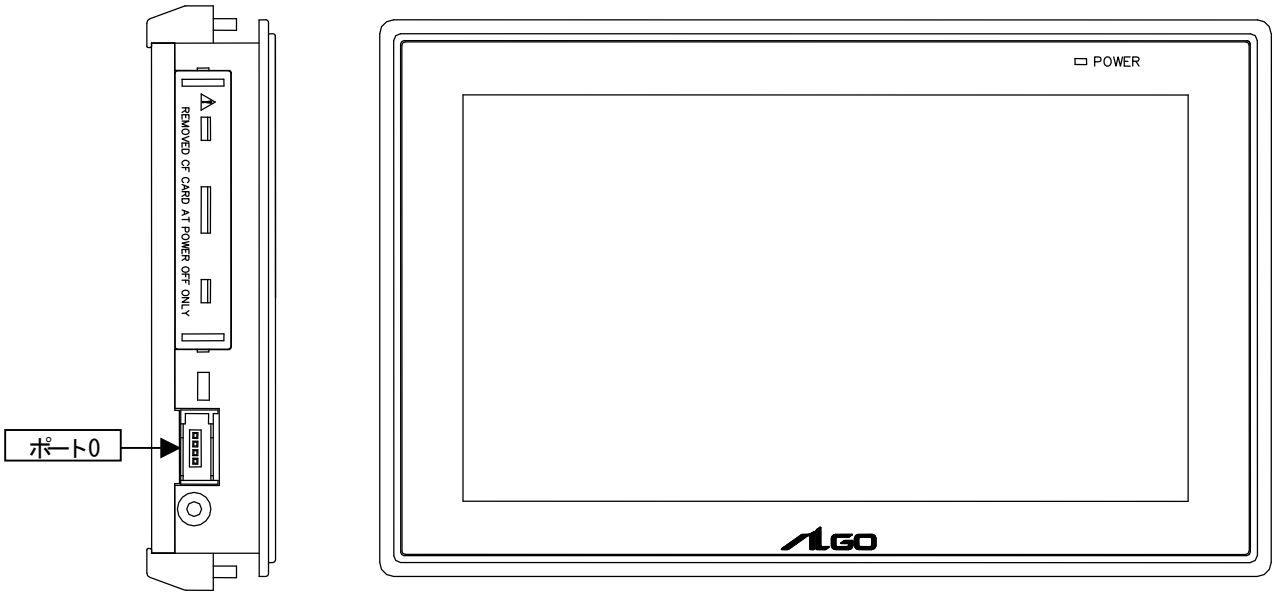


図 3-3-1-1. シリアルポート

表 3-3-1-1. シリアルタイプ別のデバイスファイル名

ﾎﾟｰﾄ番号	RS232C 時のﾃﾞﾊﾞｲｽﾌｧｲﾙ
0	rsb

アプリケーションでシリアルポートを使用するにはデバイスドライバをオープンし、リード／ライトすることで制御します。

使用できるボーレートを表 3-3-1-2 に示します。

表 3-3-1-2. シリアルタイプ別使用ボーレート範囲

RS232C 時の使用ﾎﾞｰﾚｰﾄ
2400～115200[bps]

３－３－２　デバイス詳細

・デバイス名

デバイス名は、”rsb”が使用できます。

・固有機能

シリアルポートからのデータの送受信機能

・属性データ

シリアルドライバは以下の属性データをサポートしています。

RS データ番号：tk_red_dev、tk_wri_dev でアクセスできるデータ番号です。

```
typedef enum {
    /* attribute */
    DN_RSMODE     = -100,         /* Communication mode       data: RsMode    RW */
    DN_RSFLOW     = -101,         /* Flow control             data: RsFlow    RW */
    DN_RSSTAT     = -102,         /* Line status              data: RsStat    R  */
    DN_RSBREAK    = -103,         /* BRAKE signal transmit    data: UW    W  */
    DN_RSSNDTMO   = -104,         /* Transmit timeout         data: UW    RW */
    DN_RSRCVTMO   = -105,         /* Receive timeout          data: UW    RW */
    DN_RSADDIN    = -150,         /* Addin                    data: RsAddIn   RW */

    /* IBM キーボード付加機能専用属性 */
    DN_IBMKB_KBID= -200,          /* キーボード ID     data: UW    RW */
    /* タッチパネル付加機能専用属性 */
    DN_TP_CALIBSTS= -200,         /* キャリブ状態      data: TpCalibSts RW */
    DN_TP_CALIBPAR= -201,         /* キャリブパラメータ   data: TpCalibPar RW */

    /* Hardware attribute */
    DN_RS16450    = -300          /* Hardware configuration   data: RsHwConf_16450 RW */
} RSDataNo;
```

① DN_RSMODE：通信設定を設定、読み出しすることができます。

```
typedef struct {
    UW    parity:2;               /* 0:no parity, 1:odd, 2:even, 3:None */
    UW    datalen:2;              /* 0:5bit, 1:6bit, 2:7bit, 3:8bit */
    UW    stopbits:2;             /* 0:1bit, 1:1.5bit, 2:2bit, 3:None */
    UW    rsv:2;                  /* Reserved */
    UW    baud:24;                /* baut rate */
} RsMode;
```

parity:パリティの設定を行います。パリティ無し、偶数パリティ、奇数パリティが設定可能です。
Datalen:データ長の設定を行います。7bit と 8bit のみ設定可能です。
Stopbits：ストップビットの設定を行います。1bit と 2bit のみ設定可能です。
Baud:ボーレートの設定を行います。1200/2400/4800/9600/19200/38400/57600/115200bps が設定可能です。

設定変更後、送受信バッファ、タイムアウト、フロー制御を初期化します。初期化後、タイムアウト時間は TMO_FEVR に設定されます。

② DN_RSFLOW：フロー制御を設定、読み出しすることができます。
（※フロー制御未サポートの為、使用不可）

```
typedef   struct {
      UW  rsv:26;
      UW  rcvxoff:1;  /* Receive XOFF, send diable */
      UW  csflow:1;   /* CTS */
      UW  rsflow:1;   /* RTS */
      UW  xonany:1;   /* Receive any charactor XON */
      UW  sxflow:1;   /* Send XON/XOFF */
      UW  rxflow:1;   /* Receive XON/XOFF */
} RsFlow;
```

csflow:CTS 信号の制御の有効/無効を設定します。(0：無効,1:有効)
rsflow:RTS 信号の制御の有効/無効を設定します。(0：無効,1:有効)

③ DN_RSSTAT：ラインステータスを読み出しすることができます。(Read Only)

```
typedef   struct {
#if BIGENDIAN
      UW  rsv1:20;
      UW  BE:1;   /* Recv Buffer Overflow Error*/
      UW  FE:1;   /* Framing Error    */
      UW  OE:1;   /* Overrun Error    */
      UW  PE:1;   /* Parity Error     */
      UW  rsv2:2;
      UW  XF:1;   /* Recv XOFF        */
      UW  BD:1;   /* Break Detect     */
      UW  DR:1;   /* Dataset Ready (DSR)  */
      UW  CD:1;   /* Carrier Detect (DCD) */
      UW  CS:1;   /* Clear to Send (CTS)  */
      UW  CI:1;   /* Calling Indicator(RI)*/
#else
      UW  CI:1;   /* Calling Indicator(RI)*/
      UW  CS:1;   /* Clear to Send (CTS)  */
      UW  CD:1;   /* Carrier Detect (DCD) */
      UW  DR:1;   /* Dataset Ready (DSR)  */
      UW  BD:1;   /* Break Detect     */
      UW  XF:1;   /* Recv XOFF        */
      UW  rsv2:2;
      UW  PE:1;   /* Parity Error     */
      UW  OE:1;   /* Overrun Error    */
      UW  FE:1;   /* Framing Error    */
      UW  BE:1;   /* Recv Buffer Overflow Error*/
      UW  rsv1:20;
#endif
} RsStat;
```

PE：パリティエラー発生時にこのビットに１が格納されます。読み出しによりクリアされます。
OE: オーバーランエラー発生時にこのビットに１が格納されます。読み出しによりクリアされます。
FE: フレミングエラー発生時にこのビットに１が格納されます。読み出しによりクリアされます。
BE: 受信バッファオーバーフローエラー発生時にこのビットに１が格納されます。読み出しによりクリアされます。
CI,CS,CD,DR,BD,XF：未サポート

④　DN_RSBREAK：BREAK 送信することができます。
（※未サポートの為、使用不可）

⑤　DN_RSSNDTMO：送信タイムアウト時間を設定、読み出すことができます。
data: UW

送信タイムアウト時間をミリ秒単位で設定できます。data=0 で設定しますとタイムアウト時間を TMO_FEVR で設定しますので注意してください。
また、タイムアウトは１回の送信に対するタイムアウトではなく、直前の１バイト送信から次の１バイト送信までのタイムアウトとなります。

⑥　DN_RSRCVTMO：受信タイムアウト時間を設定、読み出すことができます。
data: UW

受信タイムアウト時間をミリ秒単位で設定できます。data=0 で設定しますとタイムアウト時間を TMO_FEVR で設定しますので注意してください。
また、タイムアウトは１回の受信に対するタイムアウトではなく、直前の１バイト受信してから次の受信までのタイムアウトとなります。

⑦　DN_IBMKB_KBID ： キーボード ID
（※使用不可）

⑧　DN_TP_CALIBSTS: キャリブ状態
（※使用不可）

⑨　DN_TP_CALIBPAR: キャリブパラメータ
（※使用不可）

⑩　DN_RS16450: ハードウェア設定。
（※使用不可）

・固有データ
データ番号：0 に固定
データ数　：送受信を行うデータ数(送受信バッファサイズ以上のデータ数は読み出せません。)
シリアルポートから実際にデータの送受信を行う。

・エラーコード

シリアルの通信異常はE_IOエラー、タイムアウトはE_TMOUTとしてドライバより返ります。
また、エラー詳細情報は、RsErrorに設定されています。

```
typedef  struct {
#if BIGENDIAN
     UW  ErrorClass:16;  /* error class */
     UW  rsv1:2;
     UW  Aborted:1;  /* is abort */
     UW  Timout:1;   /* is timeout */

     UW  BE:1;   /* Recv Buffer Overflow Error*/
     UW  FE:1;   /* Framing Error    */
     UW  OE:1;   /* Overrun Error    */
     UW  PE:1;   /* Parity Error     */
     UW  rsv2:2;
     UW  XF:1;   /* Recv XOFF        */
     UW  BD:1;   /* Break Detect     */
     UW  DR:1;   /* Dataset Ready (DSR)  */
     UW  CD:1;   /* Carrier Detect (DCD) */
     UW  CS:1;   /* Clear to Send (CTS)  */
     UW  CI:1;   /* Calling Indicator(RI)*/
#else
     UW  CI:1;   /* Calling Indicator(RI)*/
     UW  CS:1;   /* Clear to Send (CTS)  */
     UW  CD:1;   /* Carrier Detect (DCD) */
     UW  DR:1;   /* Dataset Ready (DSR)  */
     UW  BD:1;   /* Break Detect     */
     UW  XF:1;   /* Recv XOFF        */
     UW  rsv2:2;
     UW  PE:1;   /* Parity Error     */
     UW  OE:1;   /* Overrun Error    */
     UW  FE:1;   /* Framing Error    */
     UW  BE:1;   /* Recv Buffer Overflow Error*/

     UW  Timout:1;   /* is timeout */
     UW  Aborted:1;  /* is abort */
     UW  rsv1:2;
     UW  ErrorClass:16;  /* error class = E_IO */
#endif
} RsError;
```

ErrorClass: エラークラスです。エラークラスの詳細は、T-Kernel仕様書のデバイス管理機能を参照。
Timout: タイムアウト発生時にこのビットに１が格納されます。読み出しによりクリアされます。
Abort: 通信に失敗時にこのビットに１が格納されます。読み出しによりクリアされます。
PE: パリティエラー発生時にこのビットに１が格納されます。読み出しによりクリアされます。
OE: オーバーランエラー発生時にこのビットに１が格納されます。読み出しによりクリアされます。
FE: フレミングエラー発生時にこのビットに１が格納されます。読み出しによりクリアされます。
BE: 受信バッファオーバーフローエラー発生時にこのビットに１が格納されます。読み出しによりクリアされます。
CI,CS,CD,DR,BD,XF：未サポート

・注意事項

本 RS ドライバでは、タイムアウトに TMO_FEVR を設定時における受信待ち状態では、同ポートのデータの送信はできません。（データ送信中時におけるデータ受信も同様）

これは、T-Kernel から Read 発行中には Write は発行できないという制約に引っかかる為に起こってしまう現象です。もし、データ受信待ち時にデータ送信を行う場合は、データ待ちをタイムアウトを用いるのではなく、ポーリング等で Read 処理が終了できるように工夫してください。

３－３－３　サンプルプログラム

「sample02_serial」に、シリアルポートを使用したサンプルソースが入っています。リスト 3-3-3-1 にシリアルポートのオープンと通信設定を行うソースコードを示します。

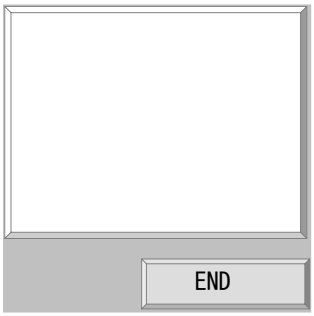


図 3-3-3-1. シリアルポート通信確認画面

リスト 3-3-3-1. シリアルポートのオープンと通信設定

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

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

#define TP_DEVNM "rsb"

//-------------------------------------------------------------
//Function for the event procedure
//-------------------------------------------------------------
void Main_Init(WSCbase* object){
  W len;
  ER err;
  /*シリアルポートオープン*/
  comm_fd = tk_opn_dev((UB*)TP_DEVNM, TD_UPDATE | TD_EXCL);
  if(comm_fd < 0){
      printf("open error¥n");
      Status_Bar->setProperty(WSNlabelString,"ttySC0 Open Error");
      return;
  }
  printf("Open OK¥n", TP_DEVNM);
  /*
   * RsMode : 通信設定
   *  UW Parity:2:        0:noparity, 1:odd,      2:even,     3:None
   *  UW Datelen:2:           0:5bit,     1:6bit,     2:7bit,     3:8bit
   *  UW stopbits:2:          0:1bit,     1:1.5bit,   2:2bit,     3:None
```

```
     *  UW rsv:2:                Reserved
     *  UW baud:24:          Baud Rateの値
     *
     * RsFlow : フロー制御設定(未サポート)
     */
  static RsMode tpmode = { 0, 3, 0, 0, 115200 };
  static RsFlow tpflow = { 0 };
  #define TMOUT 500

    /* 初期化 */
    err = tk_swri_dev(comm_fd, DN_RSMODE, &tpmode, sizeof(RsMode), &len);
    if((err < E_OK) || (sizeof(RsMode) != len)){
        printf("tk_swri_dev error : DN_RSMODE¥n");
        Status_Bar->setProperty(WSNlabelString,("tk_swri_dev error : DN_RSMODE"));
        tk_cls_dev(comm_fd, 0);
        return;
    }
    err = tk_swri_dev(comm_fd, DN_RSFLOW, &tpflow, sizeof(RsFlow), &len);
    if((err < E_OK) || (sizeof(RsFlow) != len)){
        printf("tk_swri_dev error : DN_RSFLOW¥n");
        Status_Bar->setProperty(WSNlabelString,("tk_swri_dev error : DN_RSFLOW"));
        tk_cls_dev(comm_fd, 0);
        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);
```

「tk_opn_dev」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「rsb」をオープンしています。)
「TD_EXCL」は一切の同時オープンを禁止するモードでオープンします。「tk_swri_dev」関数・モード「DN_RSMODE」で通信設定を変更します。「tk_swri_dev」関数・モード「DN_RSFLOW」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。

リスト 3-3-3-2 にシリアル通信スレッドのソースコードを示します。

リスト 3-3-3-2. シリアル通信スレッド

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

#include "newwin000.h"
#include "ComThread.h"
#define TP_DEVNM "rsb"

ID comm_fd;
WSDthread* comctrl_thr;

void *Comctrl_Thread(WSDthread* obj, void *arg)
{
  ER err;
  W len;
  UB data;
  UW i;
  i = 0;
  err = tk_swri_dev(comm_fd, 0, (void *)"Serial send test¥n", 17, &len);      /*Serial
send test とコンソールに表示*/
  if((err < E_OK) || (17 != len)){
      printf("Serial send error¥n");
      Status_Bar->setProperty(WSNlabelString,("Serial send error"));
      return(NULL);
  }
  for(;;){
      err = tk_srea_dev(comm_fd, 0, &data, sizeof(UB), &len);                 /*1 バイト
受信*/
      if(err >= E_OK){
          tk_swri_dev(comm_fd, 0, &data, sizeof(UB), &len);                   /*1 バイト
送信*/
          obj->execCallback((void*)(&data));
      }
      else{
          printf("Serial receive error¥n");
          Status_Bar->setProperty(WSNlabelString,("Serial receive error" ));
          return(NULL);
      }
      tk_dly_tsk(10);
  }
  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)
{
  B data[2];
```

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

「tk_srea_dev」関数で 1Byte 受信するまで待ちます。1Byte 受信したら、受信した文字を「tk_swri_dev」関数で送信します。また、同時にアプリケーションのテキストフィールドに受信した文字を表示します。

本サンプルプログラムではパソコンのシリアルコンソールから送った文字をシリアルポートで受信し、サンプルプログラムのウィンドウ上に表示します。
改行コードを送信するとき、改行コードが「CR+LF」以外の場合は文字化けが起こることがあります。

・注意事項
本 RS ドライバでは、タイムアウトに TMO_FEVR を設定時における受信待ち状態では、同ポートのデータの送信はできません。（データ送信中時におけるデータ受信も同様）
これは、T-Kernel から Read 発行中には Write は発行できないという制約に引っかかる為に起こってしまう現象です。もし、データ受信待ち時にデータ送信を行う場合は、データ待ちをタイムアウトを用いるのではなく、ポーリング等で Read 処理が終了できるように工夫してください。

データ受信待ち時にデータ送信を行いたい場合のサンプルが「sample02_serial_2」に入っています。このサンプルはスレッドを二つ生成し、双方を TMO_FEVR させずに常時回転させることで、Read と Write が衝突しないように作成してあります。
本プログラムを実行すると Read 側のスレッドで受信待ちを行いつつ、Write 側スレッドでシリアルに対して送信を送り続けます。Read 側スレッドは 12 文字を受け取るごとにアプリケーション上のウィンドウに受信した文字列を表示します。
リスト 3-3-3-3 にシリアル通信スレッドの Read 側の、リスト 3-3-3-4 に Write 側のソースコードを示します。

リスト 3-3-3-3. シリアル通信スレッド(Read)

```
#include "ComThread.h"

void *Comctrl_Thread_Read(WSDthread* obj, void *arg)
{
  ER err;
  W len;
  W i = 0;
  UB data;
  B recv_data[13];

  printf("read threat start¥n");

  for(;;){
      err = tk_srea_dev(comm_fd, 0, &data, sizeof(UB), &len);                /*1 ﾊﾞｲﾄ受
信*/

      if(err >= E_OK){
          recv_data[i] = data;
          i++;
      }
      if(i == 12){
          recv_data[12] = NULL;
          tex_recv->addString(recv_data);
```

```
            i = 0;
        }
        tk_dly_tsk(10);
    }
    return(NULL);
}
```

リスト 3-3-3-4. シリアル通信スレッド(Write)

```
#define MSG_START "start¥n"

#include "ComThread.h"

void *Comctrl_Thread_Write(WSDthread* obj, void *arg)
{
    ER err;
    W len;
    W i;
    UB data[] = "Serial_Send¥n";

    printf("write thread start¥n");

    err = tk_swri_dev(comm_fd, 0, (void *)"Start¥n", sizeof(MSG_START), &len);
    /*start とコンソールに表示*/
    if((err < E_OK) || (sizeof(MSG_START) != len)){
        Status_Bar->setProperty(WSNlabelString,("Serial send start error"));
        return(NULL);
    }

    for(i = 0; i < 12; i++){
        err = tk_swri_dev(comm_fd, 0, &data[i], 1, &len);                    /*1ﾊﾞｲﾄ送信*/
        if((err < E_OK) || (1 != len)){
            Status_Bar->setProperty(WSNlabelString,("Serial send error" ));
        }
        if(i == 11){
            i = -1;
        }
        tk_dly_tsk(10);
    }
    return(NULL);
}
```

３－４　　オーディオドライバ

音声出力を行うデバイスドライバです。

３－４－１　デバイス詳細

・デバイス名

デバイス名は、”audioa0”を使用します。

・固有機能

PCMデータの再生

・属性データ

オーディオドライバは以下の属性データをサポートしています。

```
typedef enum {
     /* no common attribute */
     /* specific attribute */
     DN_AUDIO_PLAYAUDIO          = 0,        /* play audio buffer */

     DN_AUDIO_GETAVAILABLEFMTS   = -100,     /* audio format information    R  */
     DN_AUDIO_SETTINGFMTS        = -101,     /* behavior audio format       RW */
     DN_AUDIO_PLAYSTATE          = -102,     /* play control     RW */
     DN_AUDIO_GETPLAYINGPOS      = -103,     /* play data position   R  */
     DN_AUDIO_PLYBUFSZ           = -104,     /* play buffer size RW */
     DN_AUDIO_MIXEROUTPUTVOL     = -105,
} AudioDataNo;
```

① DN_AUDIO_GETAVAILABLEFMTS: 使用可能なオーディオフォーマット情報を読み出すことができます。
data: AudioFMTS

```
typedef struct {
    UW  sampling:16;                    /* sampling frequency */
    UW  datafmt:8;                      /* data type    */
    UW  monoply:1;                      /* play monoral */
    UW  stereoply:1;                    /* play stereo  */
    UW  monorec:1;                      /* record monoral */
    UW  stereorec:1;                    /* record stereo */
    UW  recply:1;                       /* record/play  */
    UW  reserved:3;                     /* reserved     */
} AudioFMTS;
```

sampling: サンプリングレートを設定します。5.512/8.000/11.025/16.000/22.050/44.100/48.000KHz
が設定可能です。
datafmt: データサイズを設定します。8bit、16bitが設定可能です。
monopoly: モノラル再生の設定です。設定時は１になります。
stereorec: ステレオ再生の設定です。設定時は１になります。
monorec: モノラル録音の設定です。設定時は１になります。
stereorec: ステレオ録音の設定です。設定時は１になります。
recply: 再生/録音の設定です。再生時は0、録音時は１になります

② DN_AUDIO_SETTINGFMTS： オーディオプレイモードを設定、読み出すことができます。
Data: AudioFMTS

③ DN_AUDIO_PLAYSTATE： オーディオコントロールを設定、読み出すことができます。
Data: W

AUDIO_RUN : 再生/録音開始
AUDIO_STOP　: 再生/録音停止
AUDIO_PAUSE　: 再生/録音一時停止

④ DN_AUDIO_GETPLAYINGPOS： オーディオバッファの状態を読み出すことができます。(Read Only)
Data: AudioDataPos

```
typedef struct {
      UW  curpos;          /* current position */
      UW  accpos;          /* access position  */
} AudioDataPos;
```

⑤ DN_AUDIO_PLYBUFSZ： オーディオバッファサイズを設定、読み出すことができます。
Data: W

⑥ DN_AUDIO_MIXEROUTPUTVOL： ボリュームを設定、読み出すことができます。
Data: MixerLineVolume

```
typedef struct {
      UB lineId;           /* line ID */
      UB time;             /* the time spent changing the volume [msec] */
      H vol[2];            /* the volume value */
} MixerLineVolume;
```

lineId: ボリューム ID を設定します。
　　MIXER_LINEID_MASTEROUT　:　マスターボリューム
　　MIXER_LINEID_PCMOUT　: PCM ボリューム
　　MIXER_LINEID_MICIN　: マイクボリューム
time: ボリュームを変更するのにかかる時間 （※未使用）
vol: ボリューム値(0～100)

・固有データ
　データ番号：0 に固定

　PCM データをオーディオバッファに書き込みます。

・エラーコード
　T-Kernel 仕様書のデバイス管理機能を参照。

３－４－２　サンプルプログラム

「sample04_audio」に、音声を再生するプログラムのサンプルソースが入っています。単純な処理のフローを図 3-4-2-1 に示します。また、音声再生についてのサンプルプログラムをリスト 3-4-2-1 に示します。

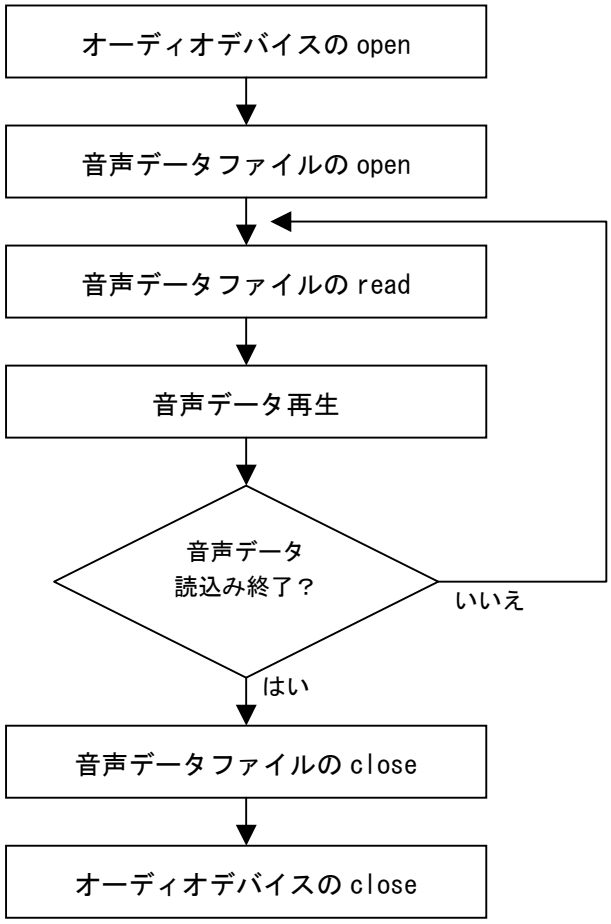


図 3-4-2-1. 音声データ再生フロー

リスト 3-4-2-1. オーディオ再生

```
/*
  playsound.c WAVE 再生

  (C) Copyright 2005 by Personal Media Corporation
*/
#include <basic.h>
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <btron/tkcall.h>
#include <btron/outer.h>
#include <device/audio.h>

#define   TEST

#ifdef    TEST
#define   DP(exp)     printf exp
#else
#define   DP(exp)     /**/
#endif

#define   MAX_QUE     AUDIO_MAXREQQ   // 最大キュー数
#define   NQUE        AUDIO_MAXREQQ   // 使用するキュー数
#define   NBLK        (4)             // 書込み単位ブロック数

LOCAL W   audio_dd = -1;      // Audio デバイスディスクリプタ
LOCAL UB  *audio_dbuf = NULL; // Audio データバッファ
LOCAL W   dbuf_offs = 0;      // アラインメント用オフセット
volatile LOCAL UB  audio_dmybuf[AUDIO_DEVBLKSIZE * NBLK * NQUE + 0x100];
//
// AUDIO デバイスのオープン/クローズ
//
LOCAL W   open_audiodev(void)
{
  if (audio_dd < 0) {
      audio_dd = tk_opn_dev("audioa0", TD_UPDATE);
      if (audio_dd < 0) {
          DP(("audio open error [%#x]¥n", audio_dd));
      }
  }
  return audio_dd;
}
LOCAL void    close_audiodev(void)
{
  if (audio_dd >= 0) {
      tk_cls_dev(audio_dd, 0);
      audio_dd = -1;
  }
```

```
}
//
// WAVE 形式のチェック(ファイルヘッダの解析)
// ヘッダファイルから
//   チャンネル数(モノラル,ステレオ)
//   サンプリングレート
// の取得、及び
//   音声ファイルのファイルポインタを読込み位置まで seek
//----------------------------------------------------------------------
//   WAV file format
//----------------------------------------------------------------------
//   0x00   "RIFF"      チャンクタイプ
//   0x04   len:L       チャンク長さ = 合計ファイルサイズ - 8
//   0x08   "WAVE"      形式名 (Form ID)
//   0x0C   "fmt "      サブチャンクタイプ
//   0x10   len:L       サブチャンク長さ >= 16
//   14  0x00:   format:H    サンプリングデータ形式
//               1:PCM, 6:A-Law, 7:u-Law, 17:IMA ADPCM
//       0x02:   ch:H        チャンネル数 (1, 2)
//       0x04:   sample:L    1 秒あたりのサンプリング数
//       0x08:   byterate:L  1 秒あたりの平均バイト数
//       0x0C:   block:H     1 サンプル当たりのバイト数
//       0x0E:   bits:H      1 サンプル当たりの有効ビット数
//       0x10:   ～          その他
//   0xXX   "data"      サブチャンクタイプ
//       len:L       サブチャンク長さ
//       data        PCM データ (len バイト)
//----------------------------------------------------------------------
//
LOCAL W   check_wavfmt(FILE *fp, UW *channel, UW *sample)
{
 UB  buf[256];
 UW  len;
 struct {
     UH  format;
     UH  ch;
     UW  sample;
     UW  byterate;
     UH  block;
     UH  bits;
 } fmt;

 if (fread(buf, 12, 1, fp) != 1)       goto EEXIT;
 if (strncmp(&buf[0], "RIFF", 4) != 0) goto EEXIT;
 if (strncmp(&buf[8], "WAVE", 4) != 0) goto EEXIT;

 *sample = *channel = 0;
 for (;;) {
     if (fread(buf, 8, 1, fp) != 1) break;
     len = *((UW*)&buf[4]);
     if (strncmp(&buf[0], "fmt ", 4) == 0) {
```

```
            if (len < sizeof(fmt)) break;
            if (fread(&fmt, sizeof(fmt), 1, fp) != 1) break;
            len -= sizeof(fmt);

        DP(("fmt=%d ch=%d sample=%d rate=%d blk=%d bits=%d (%d)¥n",
            fmt.format, fmt.ch, fmt.sample, fmt.byterate,
            fmt.block, fmt.bits, len));

            if (fmt.format != 1)            return E_NOSPT;
            if (fmt.ch != 1 && fmt.ch != 2) return E_NOSPT;
            if (fmt.block != 2 * fmt.ch)    return E_NOSPT;
            if (fmt.bits != 16)             return E_NOSPT;
            *sample = fmt.sample;
            *channel = fmt.ch;

        } else if (strncmp(&buf[0], "data", 4) == 0) {
            return len; // OK
        }
        fseek(fp, len, SEEK_CUR);
    }
EEXIT:
    return -1;
}
//
// WAVE データの取出し
//
LOCAL W   get_wavdat(FILE *fp, VP buf, W len)
{
    W   n;

    n = fread(buf, len, 1, fp);
    return n * len;
}
//
// AUDIO ミュート設定
//
LOCAL W   set_mute(W line, W on)
{
    W   er, sz;
    UW  mute;

    mute = line;
    if (on) mute |= 0x80000000;
    er = tk_swri_dev(audio_dd, DN_AUDIO_MIXERMUTELINE,
                        &mute, sizeof(mute), &sz);
    if (er < E_OK) {
        DP(("MIXERMUTELINE error [%#x]¥n", er));
    }
    return er;
}
//
```

```
// AUDIO ボリューム設定
//
LOCAL W   set_volume(W line, W val, W balance)
{
  W          i, er, ch, sz;
  MixerLineVolume     vl;
static    struct {
  MixerAllLinesDesc   d;
  UB          buf[sizeof(MixerLineDesc) * 20];
} mdesc;

  // レンジチェック(0 - 100)
  if      (val <   0)     val =   0;
  else if (val > 100)     val = 100;
  if      (balance <   0) balance =   0;
  else if (balance > 100) balance = 100;

  // ミキサーライン状態
  if (mdesc.d.nLines == 0) {
      er = tk_srea_dev(audio_dd, DN_AUDIO_MIXERENUMLINES,
                  (VP)&mdesc, sizeof(mdesc), &sz);
      if (er < E_OK) {
          DP(("MIXERENUMLINES error [%#x]¥n", er));
          return er;
      }
  }
  DP(("val %d¥n", val));
  // ボリュームのレンジ変換
  ch = 2;
  for (i = 0; i < mdesc.d.nLines; i++) {
      if (mdesc.d.LineDesc[i].lineId == line) {
          val = (val * (mdesc.d.LineDesc[i].volMax -
                  mdesc.d.LineDesc[i].volMin) / 100)
                  + mdesc.d.LineDesc[i].volMin;
          ch = mdesc.d.LineDesc[i].nChannels;
          DP(("volset %d %d %d¥n", mdesc.d.LineDesc[i].lineId, mdesc.d.LineDesc[i].volMax,
mdesc.d.LineDesc[i].volMin));
          break;
      }
  }

  // ボリューム設定
  vl.lineId = line;
  vl.time   = 0;

  vl.vol[0] = vl.vol[1] = val;
  if      (balance > 50)  vl.vol[0] = val * (100 - balance) / 50;
  else if (balance < 50)  vl.vol[1] = val * balance / 50;

  DP(("vol [%d] %d %d ¥n", line, vl.vol[0], vl.vol[1]));
```

```
  er = tk_swri_dev(audio_dd, (line == AUDIO_MIXER_LINEID_RECGAIN) ?
          DN_AUDIO_MIXERSETINPUTVOL : DN_AUDIO_MIXERSETOUTPUTVOL,
          &vl, sizeMixerLineVolume(ch),  &sz);
  if (er < E_OK) {
      DP(("MIXERSETOUT/INPUTVOL (%d) error [%#x]¥n", line, er));
  }
  return er;
}
//
//  再生の初期処理/終了処理
//
EXPORT    ERR initPlaySound(BOOL start)
{
  // 再生バッファを解放
  if (audio_dbuf != NULL) {
      free(audio_dbuf);
      audio_dbuf = NULL;
  }

  if (start) {    // 開始
      // AUDIO デバイスをオープン
      if (open_audiodev() < E_OK) {
          audio_dbuf = NULL;
          return audio_dd;
      }

      // 再生バッファの獲得: 少なくとも 32 バイトアラインの必要
      //             がある  -> 256 バイトアラインとする
      audio_dbuf = malloc(AUDIO_DEVBLKSIZE * NBLK * NQUE + 0x100);
      if (!audio_dbuf) {
          DP(("no memory¥n"));
          return E_NOMEM;
      }
      // 256 バイトアラインとするためのオフセット
      dbuf_offs = (0x100 - (((UW)audio_dbuf) & 0xFF)) & 0xFF;

  } else {    // 終了

      // ミュート設定
      set_mute(AUDIO_MIXER_LINEID_PCMOUT, 1);
      //set_mute(AUDIO_MIXER_LINEID_MASTEROUT, 1);

      // AUDIO デバイスをクローズ
      close_audiodev();
  }
  return E_OK;
}
//
//    WAV ファイルの再生
//
EXPORT    ERR playSound(B *path, W vol, W balance)
```

```
{
  W   er, sz, ioer, init;
  W   tlen, ix, n, k, preq, creq[NQUE];
  W   sample, chn;
  FILE    *fp;
  UB  *dp;
  W   adr;
  AudioDriverDataFormat   fmt;
  if (path == NULL) return initPlaySound(FALSE);  // 終了処理

  // 初期化処理
  init = 0;
  if (audio_dd < 0 || audio_dbuf == NULL) {
      er = initPlaySound(TRUE);
      if (er < E_OK) return er;
      init++;
  }

  // ファイルのオープン
  fp = fopen(path, "r");
  if (!fp) {
      DP(("open error: %s¥n", path));
      er = E_NOEXS;
      goto EEXIT;
  }

  // WAVE 形式のチェック
  tlen = check_wavfmt(fp, &chn, &sample);
  if (tlen <= 0) {
      DP(("format error¥n"));
      er = E_NOSPT;
      goto EEXIT;
  }

  // AUDIO ドライバのモード設定
  fmt.nSize      = sizeof(fmt);              // 定義構造体のサイズ
  fmt.nFormatTag = AUDIO_FORMAT_PCM_S16_LE;  // データフォーマット(固定)
  fmt.nFS        = sample;                   // サンプリングレート(Hz)
  fmt.nChannels  = chn;                      // チャンネル数( >= 1)
  fmt.nInterleaveSample = 1;                 // チャンネル切替サンプル数
  er = tk_swri_dev(audio_dd, DN_AUDIO_SETOUTPUTFMT,
                      &fmt, sizeof(fmt), &sz);
  if (er < E_OK) {
      DP(("SETOUTPUTFMT error [%#x]¥n", er));
      goto EEXIT;
  }

  if (init) { // ボリューム設定は初回のみ

      // ボリューム設定-1
      set_volume(AUDIO_MIXER_LINEID_PCMOUT, vol, balance);
```

```
        // ボリューム設定-2
        set_volume(AUDIO_MIXER_LINEID_MASTEROUT, vol, balance);

        // ミュート解除
        set_mute(AUDIO_MIXER_LINEID_PCMOUT, 0);
        set_mute(AUDIO_MIXER_LINEID_MASTEROUT, 0);
    }
    // リクエスト ID 初期化
    memset(creq, -1, sizeof(creq));
    // ファイル再生
    for (ix = 0;;) {
        // 再生データ読込み
//          dp = &audio_dbuf[dbuf_offs + (ix * AUDIO_DEVBLKSIZE * NBLK)]; // 動的配列を使用
        dp = &audio_dmybuf[dbuf_offs + (ix * AUDIO_DEVBLKSIZE * NBLK)]; // 静的配列を使用
        if (tlen <= 0) {
            sz = 0;     // データ終了
        } else {
            // 最大 NBLK ブロック読込む
            n = AUDIO_DEVBLKSIZE * NBLK;
            if (n > tlen) n = tlen;
            n = get_wavdat(fp, dp, n);

            // 不足する場合(最後)は、0 を詰める
            sz = n / AUDIO_DEVBLKSIZE;
            if ((k = n - sz * AUDIO_DEVBLKSIZE) > 0) {
                memset(dp + n, 0, AUDIO_DEVBLKSIZE - k);
                sz++;
            }
        }
        if (sz <= 0) {  // データ終了
            tlen = er = -1;
        } else {        // 再生データ書込み
            // 再生データ書込み
            er = tk_wri_dev(audio_dd, DN_AUDIO_PLAYAUDIO, dp,
                                sz, TMO_FEVR);
            sz *= AUDIO_DEVBLKSIZE;
            tlen -= sz;
            if (er < E_OK) break;
        }
        creq[ix] = er;          // リクエスト ID 保存
        ix = (ix + 1) % NQUE;   // 次のバッファ
        preq = creq[ix];        // 以前のリクエスト ID
        creq[ix] = -1;

        if (preq < 0) {         // 最初は完了を待たない
            if (tlen <= 0) {
                for (n = NQUE; --n >= 0 && creq[n] < 0;);
                if (n < 0) {
                    er = E_OK;
                    break;      // 再生終了
```

```
                }
            }
        } else {
            // 再生データ書込み完了待ち
            er = tk_wai_dev(audio_dd, preq, &sz, &ioer, TMO_FEVR);
            if (er < E_OK) break;
            if (ioer < E_OK) {er = ioer; break;}
        }
    }

    if (er < E_OK) {
        // 要求の完了を待つ
        for (n = NQUE; --n >= 0; ) {
            if (creq[n] >= 0)
                tk_wai_dev(audio_dd, creq[n], &sz,
                            &ioer, TMO_FEVR);
        }
    }
EEXIT:
    if (fp) fclose(fp);
    return er;
}

#ifdef    TEST

//
//  テスト
//
EXPORT    W   main(W ac, B *av[])
{
    W   er, volume, balance;
    B   *p;
    W   loop;

    volume = 60;
    balance = 50;
    loop = 0;

    // オプションの取出し
    for (av++; --ac > 0; av++) {
        p = *av;
        if (*p++ != '-') break;
        switch(*p++) {
        case 'v':   volume = atoi(p);   break;
        case 'r':
        case 'b':   balance = atoi(p);  break;
        case 'l':   loop = 1;   break;
        default:    goto USAGE;
        }
    }
    if (ac < 1) {
```

```
  USAGE:
       DP(("usage: playsound [-v#][-b#] file ....¥n"));
       DP((" -v#      volume (0 - 100 : default 60)¥n"));
       DP((" -b#(-r#) balance(0 - 100 : default 50)¥n"));
       DP((" file  *.wav¥n"));
       return 0;
    }

    for (;--ac >= 0; av++) {
       do{
       // 再生する
           DP(("Start Play : %s¥n", *av));
           er = playSound(*av, volume, balance);

       // 終了
           playSound(NULL, 0, 0);
           DP(("End Play [%#x]¥n", er));
       }while(loop);
    }

    return 0;
  }
  #endif    // TEST
```

このサンプルプログラムでは、ボタンをクリックすると、「open」関数で「audioa0」をオープンし、音声データのフォーマットを DN_AUDIO_SETOUTPUTFMT を使用して指定します。再生する音声ファイルをオープンして、読出したデータをデバイスファイルに「write」することでオーディオ出力端子から音声が出力されます。音声ファイルの EOF を検出したらデバイスファイルと音声ファイルをクローズして終了です。

sample04 の作業フォルダ内にサンプルプログラムの動作確認用に sample.wav という wav ファイルを用意してあります。

３－５　　ウォッチドッグタイマードライバ

MAIN 基板のウォッチドッグタイマーを対象としたデバイスドライバです。

３－５－１　　デバイス詳細

・デバイス名

デバイス名は、”wdt”を使用します。

・固有機能

①ウォッチドッグタイマーの開始/停止機能

②ウォッチドッグタイマーのカウンタのクリア機能

・属性データ

汎用入出力ドライバは以下の属性データをサポートしています。

```
typedef enum {
    DN_WDT_START     = -100,     /* ウォッチドッグタイマー開始           data: UH    W */
    DN_WDT_STOP      = -101,     /* ウォッチドッグタイマー停止           data: なし  W */
    DN_WDT_CLEAR     = -102,     /* ウォッチドッグタイマーカウンタクリア data: なし  W */
} WdtDataNo;
```

① DN_WDT_START：ウォッチドッグタイマーを開始します。開始後、設定したタイマ値以内にカウントクリアをしないとウォッチドッグタイマーが働きリセットします。

Data：UH

Timer：ウォッチドッグタイマのカウント値を設定します。設定値は 0～65535(msec)です。

② DN_WDT_STOP：ウォッチドッグタイマーを停止します。

Data：なし

③ DN_WDT_CLEAR：ウォッチドッグタイマーのカウンタをクリアします。

Data：なし

・固有データ

なし

・エラーコード

T-Kernel 仕様書のデバイス管理機能を参照。

３－５－２　サンプルプログラム

「sample07_wdt」にウォッチドッグタイマデバイスを使用したサンプルソースが入っています。リスト 3-5-2-1 にウォッチドッグタイマデバイスのオープン、タイマクリア処理を記述したソースコードを示します。

リスト 3-5-2-1. ウォッチドッグタイマ START

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

#include "wdt02.h"
#include "Start.h"

W timeout;
ID fd;
//-------------------------------------------------------------
//Function for the event procedure
//-------------------------------------------------------------
void Start(WSCbase* object){
  W  len;
  ER ret;
  Status_Bar->setProperty(WSNlabelString,(""));
  /*
   * タイムアウトの設定
   */
  timeout = edt_timer->getProperty(WSNlabelString);
  if(timeout < 0){
      printf("timeout value error¥n");
      Status_Bar->setProperty(WSNlabelString,("timeout value error"));
      return;
  }
  /*
   * watchdog デバイスのオープン
   */
  fd = tk_opn_dev((UB*)"wdt", TD_WRITE);
  if (fd < 0) {
      printf("wdt open faild¥n");
      Status_Bar->setProperty(WSNlabelString,("wdt open faild"));
      return;
  }
  /*
   * watchdog スタート
   */
  ret = tk_swri_dev(fd, DN_WDT_START, &timeout, sizeof(UH), &len);
  if((ret < 0) || (sizeof(UH) != len)){
      printf("tk_swri_dev error DN_WDT_START¥n");
      Status_Bar->setProperty(WSNlabelString,("tk_swri_dev error DN_WDT_START"));
      tk_cls_dev(fd, 0);
      return;
  }
}
static WSCfunctionRegister  op("Start",(void*)Start);
```

このサンプルプログラムはノーマルアプリケーションとして作成されています。
画面上のボタン「START」でウォッチドッグの設定を行い、有効とします。このときテキストボックスの時間を読込んでタイムアウトを設定します。
次に「CLEAR」ボタンを押すことにより、ウォッチドッグタイマをクリアします。
「END」で終了した場合は,ウォッチドッグを無効にした状態で終了します。
もし仮に、ウォッチドッグを有効にした状態で終了した場合は、ウォッチドッグタイマのタイムアウトと共にハードウェアリセットが入ります。

リスト 3-5-2-2. ウォッチドッグタイマ CLEAR

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

#include "wdt02.h"
#include "Start.h"
//-------------------------------------------------------------
//Function for the event procedure
//-------------------------------------------------------------
void Clear(WSCbase* object){
  W  len;
  ER ret;
  /*
   * ウォッチドッグクリア処理
   */
  if(fd){
      ret = tk_swri_dev(fd, DN_WDT_CLEAR, 0, 0, &len);        /* クリア */
      if(ret < 0){
          printf("tk_swri_dev error DN_WDT_CLEAR¥n");
          Status_Bar->setProperty(WSNlabelString,("tk_swri_dev error DN_WDT_CLEAR"));
          tk_cls_dev(fd,0);
          return;
      }
      printf("watchdog timer cleared¥n");
  }
  else{
      printf("watchdog not open¥n");
      Status_Bar->setProperty(WSNlabelString,("watchdog not open"));
  }
}
static WSCfunctionRegister  op("Clear",(void*)Clear);
```

リスト 3-5-2-3. ウォッチドッグタイマ END

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

#include "wdt02.h"
#include "Start.h"
//-------------------------------------------------------
//Function for the event procedure
//-------------------------------------------------------
void End(WSCbase* object){
  W  len;
  ER ret;
  /* WDT 無効で終了 */
  if(fd){
      ret = tk_swri_dev(fd, DN_WDT_STOP, 0, 0, &len);     /* WDT STOP */
      if(ret < 0){
          printf("tk_swri_dev error DN_WDT_STOP¥n");
      }
      tk_cls_dev(fd,0);
  }
  /* 終了 */
  exit(0);
}
static WSCfunctionRegister  op("End",(void*)End);
```

３－６　　汎用入力IN0 リセット/IN1 割込みドライバ

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。また、汎用入力 IN1 割込みは、汎用入力の IN1 が ON した場合に割り込みを発生させる機能です。デバイスドライバからこれらの機能の有効・無効を切替えることができます。

※　汎用入力 IN0 リセット/IN1 割込みドライバは AP-3102 のみのサポートとなります。

３－６－１　デバイス詳細

・デバイス名

デバイス名は、”rasin”を使用します。

・固有機能

汎用入力の IN0 が ON した場合に本体をリセットし、IN1 が ON した場合に割り込みを発生させます。

・属性データ

rasin ドライバは以下の属性データをサポートしています。

```
typedef enum {
      DN_RASIN_INORST = -100,      /* RASIN IN0 リセット設定    R/W data: UW */
      DN_RASIN_IN1INT = -101,      /* RASIN IN1 割込み設定      R/W data: RASIN_DATA / UW  */
      DN_RASIN_IN1SET = -102,      /* RASIN IN1 割込み設定      W       data: UW */
} rasinDataNo;
```

① DN_RASIN_INORST　：RASIN の IN0 リセット機能の設定を読書きします。
Data：UW　　0:無効　　0 以外:有効

② DN_RASIN_IN1INT　：RASIN の IN1 割込み機能の有効無効と割込み発生時のコールバック関数の設定を読書きします。

```
Data：RASIN_DATA
typedef struct{
    UW  value;          /* 0:無効     0 以外:有効              */
    UW  callback;       /* 割込み発生時のコールバック関数   */
} RASIN_DATA;
```

③ DN_RASIN_IN1SET　：RASIN の IN1 割込み機能の有効無効のみの設定を読書きします。
Data：UW　　0:無効　　0 以外:有効

・固有データ

なし

・エラーコード

T-Kernel 仕様書のデバイス管理機能を参照。

３－６－２　サンプルプログラム(IN0 リセット)

「sample08_rasin」に汎用入力 IN0 リセット機能を操作したサンプルソースが入っています。このサンプルプログラムはノーマルアプリケーションとして作成されています。シリアルターミナル、telnet などで動作させることができます。リスト 3-6-2-1 にソースコードを示します。
デバイスドライバは、「rasin」です。

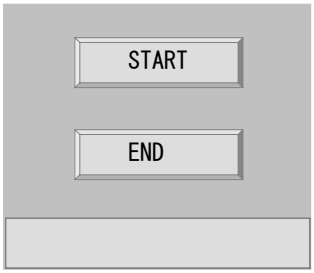


図 3-6-2-1. 汎用入力 IN0 画面

リスト 3-6-2-1. 汎用入力 IN0 リセット

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

#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <string.h>
#include <basic.h>
#include <stdlib.h>
#include <errno.h>

#include <btron/tkcall.h>
#include <btron/outer.h>
#include <device/ap730/rasin.h>

#include <sys/types.h>
#include <sys/stat.h>
#include "start.h"
#include "rasin01.h"

ID desc;
W len;
W err;
UW onoff;
//-------------------------------------------------------------
//Function for the event procedure
//-------------------------------------------------------------
void start(WSCbase* object){
  Status_Bar->setProperty(WSNlabelString,(""));
  /*
   * RAS/Input デバイスのオープン
   */
```

```
    desc = tk_opn_dev((UB*)"rasin", TD_UPDATE);
    if(desc < E_OK){
        printf("rasin open faild¥n");
        Status_Bar->setProperty(WSNlabelString,("rasin open faild"));
        return;
    }
  A
    /*
     * 現在の設定を取得
     *
     * onoff: 1    有効
     *      : 0    無効
     */
    onoff = 0;
    err = tk_srea_dev(desc, DN_RASIN_INORST, &onoff, sizeof(UW), &len);          /*
  len は必ず 0 が返る */
    if(err < E_OK){
        printf("tk_srea_dev error DN_RASIN_INORST¥n");
        Status_Bar->setProperty(WSNlabelString,("tk_srea_dev error DN_RASIN_INORST"));
        tk_cls_dev(desc, 0);
        return;
    }
    printf("tk_srea_dev INORST: %d¥n", onoff);
    /*
     * INO リセットを有効にする
     *
     * onoff: 1    有効
     *      : 0    無効
     */
    onoff = 1;
    err = tk_swri_dev(desc, DN_RASIN_INORST, &onoff, sizeof(UW), &len);          /*
  len は必ず 0 が返る */
    if(err < E_OK){
        printf("tk_swri_dev error DN_RASIN_INORST¥n");
        Status_Bar->setProperty(WSNlabelString,("tk_swri_dev error DN_RASIN_INORST"));
        tk_cls_dev(desc, 0);
        return;
    }
    tk_cls_dev(desc, 0);
  }
  static WSCfunctionRegister  op("start",(void*)start);
```

リセットを無効とする設定を以下に示します。リセットを有効にして INO が ON する前にリセットを無効と変更すると、INO が ON してもリセットは掛からない状態となります。

リスト 3-6-2-2. 汎用入力 INO リセット無効

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

#include "start.h"
```

```
#include "rasin01.h"
//-------------------------------------------------------------
//Function for the event procedure
//-------------------------------------------------------------
void end(WSCbase* object){
  /* rasin デバイスドライバオープン */
  desc = tk_opn_dev((UB*)"rasin", TD_UPDATE);
  if(desc < E_OK){
      printf("rasin open faild¥n");
      Status_Bar->setProperty(WSNlabelString,("rasin open faild"));
      exit(0);
  }
  /*
   * IN0 リセットを無効にする
   *
   * onoff: 1    有効
   *      : 0    無効
   */
  onoff = 0;
  err = tk_swri_dev(desc, DN_RASIN_INORST, &onoff, sizeof(UW), &len);     /* len は必ず 0
が返る */
  if(err < E_OK){
      printf("tk_swri_dev error DN_RASIN_INORST¥n");
  }
  tk_cls_dev(desc, 0);
  /* 終了 */
  exit(0);
}
static WSCfunctionRegister  op("end",(void*)end);
```

３－６－３　サンプルプログラム(IN1 割込み)

「sample09_rasin」に汎用入力 IN1 割込み機能を操作したサンプルソースが入っています。**このサンプルプログラムは T-Kernel ベースアプリケーションとして作成されています。**使用する際は、シリアルターミナル、telnet などから以下のように入力してください。

```
[/SYS]% lodspg sample09_rasin
```

リスト 3-7-3-1 にソースコードを示します。デバイスドライバは、「rasin」です。

リスト 3-6-3-1. 汎用入力 IN1 割込み

```
#include <tk/tkernel.h>
#include <btron/tkcall.h>
#include <btron/outer.h>
#include <device/asd_sh7760/rasin.h>

#define FPGA_ENREG3          0xB0900000
#define FPGA_ENREG3_IN1_MASK 0x02
//
//    Test program Main
//
W         step=0;
UW            value;
RASIN_DATA    rasd;
ID            rasinID;
ID            tid;
T_CTSK        ctsk;
T_DINT        dint;

void  dmy_in1int(UINT dintno)
{
  //割り込み関数
  out_h(FPGA_INTREG1, in_h(FPGA_INTREG1) & ~FPGA_INTREG1_IN1);
  //ここに割り込み発生時に実行する処理を追加してください


  step++;
  return;
}
void task(INT stacd, VP exinf)
{
  W len = 0;
  for(;;){
      if(step > 0){
          step = 0;
          //割り込み有効
          value = RASIN_IN1INT_SET;
          tk_swri_dev(rasinID, DN_RASIN_IN1SET, &value, sizeof(UW), &len);
//            printf("interrupt¥n");
      }
      tk_dly_tsk(100);
  }
```

```
}
EXPORT ER main( INT ac, UB *av[] )
{
  W len = 0;
  if(ac >= 0)                    //LODSPG時実行
  {
      //TASK生成
      ctsk.exinf      = NULL;                    //拡張情報
      ctsk.tskatr     = TA_HLNG | TA_RNG0;       //タスク属性
      ctsk.task       = task;                    //タスク起動アドレス
      ctsk.itskpri    = 100;                     //タスク起動時優先度
      ctsk.stksz      = 16384;                   //スタックサイズ
      tid             = tk_cre_tsk( &ctsk );     //タスク生成
      if(tid < E_OK)                             //タスク生成失敗処理
      {
          printf("task error¥n");
          return 0;
      }
      //デバイスオープン
      rasinID = tk_opn_dev((UB*)"rasin", TD_UPDATE);
      if (rasinID < E_OK) {
          printf("tk_opn_dev() error¥n");
          return 0;
      }
      printf("Ras-In Driver OPEN OK.¥n");
      //割り込み有効設定

      rasd.value      = RASIN_IN1INT_SET;
      rasd.callback   = (UW)dmy_in1int;
      tk_swri_dev(rasinID, DN_RASIN_IN1INT, &rasd, sizeof(RASIN_DATA), &len);
      //TASK実行
      tk_sta_tsk( tid, NULL );
  }else{                         //UNLSPG時実行
      if(tid > 0)
      {
          //タスク終了と削除
          tk_ter_tsk(tid);
          tk_del_tsk(tid);
          tid = NULL;
      }
      if(rasinID > 0)
      {
          //割り込み無効、デバイスクローズ
          value = RASIN_IN1INT_CLR;
          tk_swri_dev(rasinID, DN_RASIN_IN1SET, &value, sizeof(UW), &len);
          tk_cls_dev(rasinID, 0);
          rasinID = NULL;
      }
  }
  return 0;
}
```

最初に「tk_opn_dev」関数で"rasin"を呼び出して、DN_RASIN_IN1INT を「tk_swri_dev」関数で呼び出し、IN1 割込みを有効にします。
TASK を実行し、割り込みがかかるのを待ちます。
割り込みが発生すると dmy_in1int 関数が呼び出され、関数内の処理が実行されます。

３－７　　バックアップSRAMドライバ

バックアップ SRAM は、バックアップバッテリ付きの SRAM です。デバイスドライバから SRAM のデータを読書きできます。

※　バックアップ SRAM ドライバは AP-3102 のみのサポートとなります。

３－７－１　　デバイス詳細

・デバイス名

　　デバイス名は、”rasram”を使用します。

・固有機能

　バックアップバッテリ付きの SRAM への読書きを行います。

・属性データ

　rasram ドライバは属性データを持ちません。

・固有データ

　開始位置を設定します。

・エラーコード

　T-Kernel 仕様書のデバイス管理機能を参照。

３－７－２　　サンプルプログラム

「sample10_rasram」にバックアップ付き SRAM を read/write システムコールで操作したサンプルソースが入っています。このサンプルプログラムはノーマルアプリケーションとして作成されています。シリアルターミナル、telnet などで動作させることができます。リスト 3-8-2-1 にソースコードを示します。

バックアップ SRAM として 512KByte の SRAM を実装しています。4KByte はシステム領域として使用しているため、ユーザー領域は 508KByte を使用することができます。

デバイスドライバは、「rasram」です。

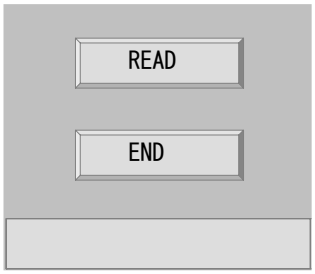


図 3-7-2-1. バックアップ付き SRAM 画面

リスト 3-7-2-1. バックアップ付き SRAM (read/write)

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

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

```
#include <basic.h>
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <btron/tkcall.h>
#include <btron/outer.h>
#include "rasram01.h"

#define RASRAM_SIZE   0x7F000

UB w_rasram[RASRAM_SIZE];
UB r_rasram[RASRAM_SIZE];
//-------------------------------------------------------------
//Function for the event procedure
//-------------------------------------------------------------
void Check(WSCbase* object){
  ID desc;
  W i;
  W len;
  W d;
  ER err;

  Status_Bar->setProperty(WSNlabelString,(""));
  /*
   * RAS/SRAM デバイスのオープン
   */
  desc = tk_opn_dev((UB*)"rasram", TD_UPDATE);
  if(desc < E_OK){
      printf("rasram open faild¥n");
      Status_Bar->setProperty(WSNlabelString,("rasram open faild"));
      return;
  }

  /*
   * 書込みデータの作成
   */
  memset(&(w_rasram[0]), 0x00, RASRAM_SIZE);
  for(i = 0, d = 0; i < RASRAM_SIZE; i++, d++){
      w_rasram[i] = (UB)(d & 0xff);
  }

  /*
   * データの書込み
   */
  err = tk_swri_dev(desc, 0, &(w_rasram[0]), RASRAM_SIZE, &len);
  if( (err != E_OK) || (len != RASRAM_SIZE) ){
      printf("tk_swri_dev error¥n");
      Status_Bar->setProperty(WSNlabelString,("tk_swri_dev error"));
      tk_cls_dev(desc, 0);
      return;
```

```
    }

    /*
     * データの読込み
     */
    len = 0;
    memset(&r_rasram[0], 0x00, RASRAM_SIZE);
    err = tk_srea_dev(desc, 0, &(r_rasram[0]), RASRAM_SIZE, &len);
    if( (err != E_OK) || (len != RASRAM_SIZE) ){
        printf("tk_srea_dev error¥n");
        Status_Bar->setProperty(WSNlabelString,("tk_srea_dev error"));
        tk_cls_dev(desc, 0);
        return;
    }

    /*
     * データのチェック
     */
    if ( memcmp(&(w_rasram[0]), &(r_rasram[0]), RASRAM_SIZE)!=0 ) {
        printf("data compare faild¥n");
        Status_Bar->setProperty(WSNlabelString,("data compare faild"));
        tk_cls_dev(desc, 0);
        return;
    }

    printf("read/write check ok¥n");
    tk_cls_dev(desc, 0);
}
static WSCfunctionRegister  op("Check",(void*)read);
```

３－８　　タッチブザードライバ

画面の特定の領域をタッチした際にブザーを鳴らすドライバです。

３－８－１　デバイス詳細

・デバイス名

デバイス名は、”bz”を使用します。

・属性データ

タッチブザードライバは以下の属性データをサポートしています。

```
typedef enum {
      DN_BZ_START       = -100,    ブザー開始                  data: なし        (Write)
      DN_BZ_STOP        = -101,    ブザー停止                  data: なし        (Write)
      DN_BZ_SETSTATUS   = -102,    ブザー鳴動時間設定書き込み  data: W         (Write)
      DN_BZ_GETSTATUS   = -103,    ブザー鳴動時間設定読み取り  data: W         (Read)
} BzDataNo;
```

① DN_BZ_START　：ブザーを ON にします。
　Data：なし

② DN_BZ_STOP　：ブザーを OFF にします。
　Data：なし

③ DN_BZ_SETSTATUS：ブザーの鳴動時間設定を書き込みます。
　Data:W

※鳴動時間を 0 以下に設定すると、DN_BZ_STOP を送るまで無限にブザーが鳴り続けます。

④ DN_BZ_GETSTATUS：ブザーの鳴動時間設定を読み込みます。
　Data：W

・固有データ

なし

３－８－２　サンプルプログラム

「sample21_buzzer」に、bz を使ったサンプルソースが入っています。

このサンプルは、ボタンを押下することで bz ドライバを呼び出し、予め指定した周波数と鳴動時間のブザーを鳴らしますリスト 3-8-2-1 にソースコードを示します。

リスト 3-8-2-1. タッチブザードライバのオープンと設定

```
#include <WScom.h>
#include <WSCfunctionList.h>
#include <WSCbase.h>
#include <WSCvlabel.h>
#include <WSCvslider.h>
#include <btron/tkcall.h>
#include <btron/outer.h>
#include <device/asd_sh7760/bz.h>
extern WSCvlabel* lab_bzstatus;
extern WSCvslider* sld_time;
//-------------------------------------------------------------
//Function for the event procedure
//-------------------------------------------------------------
void btn_click(WSCbase* object){
  W btn  = 0;
  W len  = 0;
  W time = 0;
  ID fd;
  ER ret;
  char labstatus[256];
  W bz_status;
  W read_status;

  btn  = object->getProperty(WSNuserValue);       /* ｸﾘｯｸされたﾎﾞﾀﾝを特定する */
  time = sld_time->getProperty(WSNvalue);         /* 鳴動時間の取得 */

  bz_status   = time;
  read_status = 0;

  fd = tk_opn_dev((UB*)"bz", TD_UPDATE | TD_EXCL);
  if (fd < 0) {
      printf("bz open faild¥n");
      if(btn < 0){
          exit(0);
      }
      else return;
  }
  if(btn == 2){   /* SET ボタン */
      ret = tk_swri_dev(fd, DN_BZ_SETSTATUS, &bz_status, sizeof(W), &len);
      if(ret < 0){
          printf("tk_swri_dev error DN_BZ_SETSTATUS¥n");
      }
      ret = tk_srea_dev(fd, DN_BZ_GETSTATUS, &read_status, sizeof(W), &len);
      if(ret < 0){
```

```
            printf("tk_swri_dev error DN_BZ_SETSTATUS¥n");
        }
        sprintf(labstatus, "bz_status:time = %d¥n", read_status);
        lab_bzstatus->setProperty(WSNlabelString, labstatus);
    }
    if(btn == 1){   /* ON ボタン */
        ret = tk_swri_dev(fd, DN_BZ_START, 0, 0, &len);
        if(ret < 0){
            printf("tk_swri_dev error DN_BZ_START¥n");
            return;
        }
        ret = tk_srea_dev(fd, DN_BZ_GETSTATUS, &read_status, sizeof(W), &len);
        if(ret < 0){
            printf("tk_swri_dev error DN_BZ_SETSTATUS¥n");
        }
        sprintf(labstatus, "bz_status:time = %d¥n", read_status);
        lab_bzstatus->setProperty(WSNlabelString, labstatus);
    }
    if(btn == 0){   /* OFF ボタン */
        ret = tk_swri_dev(fd, DN_BZ_STOP, 0, 0, &len);
        if(ret < 0){
            printf("tk_swri_dev error DN_BZ_STOP¥n");
            return;
        }
        ret = tk_srea_dev(fd, DN_BZ_GETSTATUS, &read_status, sizeof(W), &len);
        if(ret < 0){
            printf("tk_swri_dev error DN_BZ_SETSTATUS¥n");
        }
        sprintf(labstatus, "bz_status:time = %d¥n", read_status);
        lab_bzstatus->setProperty(WSNlabelString, labstatus);
    }
    if(btn < 0){    /* CLOSE ボタン */
        ret = tk_swri_dev(fd, DN_BZ_STOP, 0, 0, &len);
        if(ret < 0){
            printf("tk_swri_dev error DN_BZ_STOP¥n");
        }
        printf("end¥n");
        tk_cls_dev(fd,0);
        exit(0);
    }
    tk_cls_dev(fd,0);
}
static WSCfunctionRegister  op("btn_click",(void*)btn_click);
```

「tk_opn_dev」関数で「bz」を開きます。タッチブザードライバは TD_UPDATE | TD_EXCL 設定で、リード／ライトすることができます。
「tk_swri_dev」関数で DN_BZ_SETSTATUS を呼び出すことでブザー設定を指定します。また、「tk_wrea_dev」関数で DN_BZ_STATUS を呼び出すことでブザー設定の状態を読み込むことができます。
「tk_swri_dev」関数で DN_BZ_START を呼び出すことでブザーを ON にします。また、DN_BZ_STOP を呼び出すことでブザー設定を OFF にできます。

３－９　　その他のサンプルプログラム

これまでは、デバイスドライバを使用したサンプルプログラムの説明をしてきました。ここでは、その他のサンプルプログラムについて説明します。

３－９－１　起動ランチャー

「sample05_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ソースが入っています。

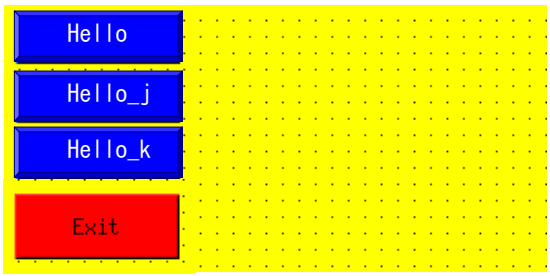


図 3-5-1-1. ランチャーサンプルプログラムメイン画面

このサンプルソースでは、ウインドウマネージャを使用しない設定にしたため、メインウインドウのプロパティを表 3-9-1-1 の様に変更します。

表 3-9-1-1. メイ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	ｳｨﾝﾄﾞｳのﾀｽｸﾊﾞｰ属性の設定	WM 管理外
終了	ｳｨﾝﾄﾞｳを非表示にしたとき終了するかしないかの設定	ｵﾌ

各ボタンのプロパティの「ユーザ設定値」の項目を各ボタンでユニークな番号を設定します。これで、ボタンクリックしたときのプロシージャイベント関数を同じにすることができます。リスト 3-9-1-1 にボタンクリックプロシージャ関数のソースコードを示します。

リスト 3-9-1-1. ランチャーボタンソースコード

```
#include <WScom.h>
#include <WSCfunctionList.h>
#include <WSCbase.h>
#include "newwin000.h"

#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <cnvend.h>
#include <string.h>
#include <basic.h>
#include <tcode.h>
#include <btron/proctask.h>
#include <btron/message.h>
#include <btron/file.h>
#include <stdlib.h>
#include <errno.h>

#include <btron/tkcall.h>
```

```
#include <btron/outer.h>
#include <sys/stat.h>
#include <sys/types.h>
#include <sys/util.h>
#include <btron/typedef.h>

//-------------------------------------------------------
//Function for the event procedure
//-------------------------------------------------------
void Btn_Click(WSCbase* object){
long btn;
    LINK lnk;
    MESSAGE msg;
    W pid;
//program path1 = /SYS/hello.out
    TC path1[] = {TC_FDLM, TK_S, TK_Y, TK_S, TC_FDLM, TK_h, TK_e, TK_l, TK_l, TK_o, TK_PROD,
TK_o, TK_u, TK_t, TNULL};
//program path2 = /SYS/hello_j.out
    TC path2[] = {TC_FDLM, TK_S, TK_Y, TK_S, TC_FDLM, TK_h, TK_e, TK_l, TK_l, TK_o, TK_USCR,
TK_j, TK_PROD, TK_o, TK_u, TK_t, TNULL};
//program path3 = /SYS/hello_k.out
    TC path3[] = {TC_FDLM, TK_S, TK_Y, TK_S, TC_FDLM, TK_h, TK_e, TK_l, TK_l, TK_o, TK_USCR,
TK_k, TK_PROD, TK_o, TK_u, TK_t, TNULL};
    //message setting
    memset(&msg, 0, sizeof(MESSAGE));
    msg.msg_type = MS_TYPE0;
    msg.msg_size = 3;
    msg.msg_body.ANYMSG.msg_str[0] = 00;
    msg.msg_body.ANYMSG.msg_str[1] = 00;
    msg.msg_body.ANYMSG.msg_str[2] = 00;

    btn = object->getProperty(WSNuserValue);    /* クリックされたボタンを特定する */
    switch(btn){
        case 1:
            if(get_lnk(path1, &lnk, F_NORM) < 0){
                printf("get_lnk Error¥n");
                return;
            }
            pid = cre_prc(&lnk, -1, &msg);      /* hallo プログラムを起動 */
            if(pid < 0){
                printf("cre_prc Error = %x¥n", pid);
                return;
            }
            break;
        case 2:
            if(get_lnk(path2, &lnk, F_NORM) < 0){
                printf("get_lnk Error¥n");
                return;
            }
            pid = cre_prc(&lnk, -1, &msg);      /* hallo_j プログラムを起動 */
            if(pid < 0){
```

```
                printf("cre_prc Error = %x¥n", pid);
                return;
            }
            break;
        case 3:
            if(get_lnk(path3, &lnk, F_NORM) < 0){
                printf("get_lnk Error¥n");
                return;
            }
            pid = cre_prc(&lnk, -1, &msg);      /* hallo_k プログラムを起動 */
            if(pid < 0){
                printf("cre_prc Error = %x¥n", pid);
                return;
            }
            break;
        case 0:
            exit(0);                            /* 終了 */
            break;
    }
}
static WSCfunctionRegister  op("Btn_Click",(void*)Btn_Click);
```

「cre_prc」関数の引数で指定した文字列のコマンドを実行します。WideStudio で作成したメイン画面のプロパティの「タイトル属性」を「WM 管理外」としたとき、そのメイン画面が常に最前面で表示されるため、メイン画面を非表示にしないと起動するプログラムが隠れてしまいます。また、メインプログラムを非表示にするとき、メイン画面のプロパティの「終了」が「オン」になっていると、プログラムが終了してしまうので、「オフ」にしています。
「hello」をクリックすると、無地のボタンと Close ボタンのアプリが起動し、無地のボタンを押すと「hello」と表示されます。
「hello_j」をクリックすると、無地のボタンと Close ボタンのアプリが起動し、無地のボタンを押すと「こんにちは」と表示されます。
「hello_k」をクリックすると、無地のボタンと Close ボタンのアプリが起動し、無地のボタンを押すと「コンニチハ」と表示されます。
「Exit」 をクリックすると、ランチャーを終了します。

３－９－２　ソケット通信サンプル(LAN通信)

「sample03_network」に、LAN を使ってソケット通信を行うサンプルプログラムが入っています。

このサンプルは Server 側のアプリケーションと Client 側のアプリケーションの二つに分かれています。
このサンプルを使用するときは以下のように入力してください。

```
[/SYS]% server_spl.out &
[/SYS]% client_spl.out 127.0.0.1 &
```

Client 側アプリケーションは起動したときの引数(IP アドレス)に対してソケット通信を行います。ひとつの ASP 上で動作を確認する場合は自分自身の IP アドレスか、127.0.0.1 を指定してください。

リスト 3-9-2-1. Client 側アプリケーションのソースコード

```
#include <unistd.h>

#include <tk/tkernel.h>
#include <btron/bsocket.h>
#include <btron/proctask.h>
#include <btron/tkcall.h>
#include <btron/message.h>
#include <tstring.h>

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

#include "newwin000.h"

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

    /* ﾌﾟﾛｸﾞﾗﾑ起動時の引数でｻｰﾊﾞの IP ｱﾄﾞﾚｽを取得 */
    if(WSGIappDev()->getArgc() < 2){
        Status_Bar->setProperty(WSNlabelString,"No IP Address");
        return;
    }
    argv = WSGIappDev()->getArgv();
    /* TRON 文字列→EUC 文字列への変換 */
    tcstoeucs(eucstr, (TC *)argv[1]);
    srv_ip = (char *)eucstr;
    printf("ip_address = %s¥n", srv_ip);
```

```
    /* ｸﾗｲｱﾝﾄｿｹｯﾄ作成*/
    sock = so_socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    if (sock < 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);          //ﾀｰｹﾞｯﾄｻｰﾊﾞ IP ｱﾄﾞﾚｽ
    srv_addr.sin_port        = htons(50000);               //ﾀｰｹﾞｯﾄｻｰﾊﾞﾎﾟｰﾄ番号
    connect = so_connect(sock, (struct sockaddr *) &srv_addr, sizeof(srv_addr));
    if (connect < 0){
        Status_Bar->setProperty(WSNlabelString,"connect failed");
        close(sock);
        return;
    }

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

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

リスト 3-9-2-2. Server 側アプリケーションのソースコード

```
#include "SrvThread.h"

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

        /* ﾎﾟｰﾄ番号指定 */
    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(50000);               //ﾎﾟｰﾄ番号指定
```

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

        /* ｶｰﾈﾙ通知 */
    listen = so_listen(srv_sock, 1);
    if (listen < 0){
        Status_Bar->setProperty(WSNlabelString,"listen() failed");
        close(srv_sock);
        return;
    }

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

リスト 3-9-2-3. Server 側アプリケーションのスレッド内のソースコード

```
#include <unistd.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);
        cli_sock = so_accept(srv_sock, (struct sockaddr *) &cli_addr, &len);
        if(cli_sock < 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 = so_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);
                    break;                          /*通信断*/
                }
            }
            tk_dly_tsk(10); /*10mswait*/
        }

    }
    return(NULL);
}
/*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;
}
```

３－９－３　バックライト輝度調整サンプル

「sample20_backlight」にLCDのバックライト輝度を調整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。
本サンプルプログラムは/SYS/DEVCONFファイルの内容を書き換えることにより、バックライト輝度の変更を行っています。
本サンプルプログラムは/SYS/DEVCONF_NEWという一時ファイルを作成し、/SYS/DEVCONFファイルと差し替えます。/SYS/DEVCONF_NEWという名前のファイルを別件で保存している場合は上書きしてしまいますので、別名で保存しなおすなどをするようにしてください。

リスト3-9-3-1. バックライト輝度調整サンプルのソースコード

```
#include <WScom.h>
#include <WSCfunctionList.h>
#include <WSCbase.h>
#include <stdio.h>
#include <WSCvlabel.h>
extern WSCvlabel* lab_value;

//-------------------------------------------------------
//Function for the event procedure
//-------------------------------------------------------
void BackLightChange(WSCbase* object){
    FILE *fp;
    FILE *fp2;
    char fname_org[256] = "/SYS/DEVCONF";
    char fname_new[256] = "/SYS/DEVCONF_NEW";
    char confstr[256];
    char com_name[256] = "LCDBRIGHT";
    int btn;
    int bright = 0;
    int bl_flag = 0;

    btn = object->getProperty(WSNuserValue);    /* ｸﾘｯｸされたﾎﾞﾀﾝを特定する */
    if(btn == -1){
        printf("closed");
        exit(0);
    }
    else if(btn == 1){
        bright = lab_value->getProperty(WSNlabelString);
    }
    bright = (bright * 2.55) + 1;
    if(bright < 0){
        bright = 1;
    }else if(bright > 255){
        bright = 255;
    }

    fp = fopen(fname_org, "r");
    fp2 = fopen(fname_new, "w");
    if(fp == NULL){
        printf("NO FILE¥n");
```

```
        return;
    }

    while(fgets(confstr, 256, fp) != NULL){
        /*書き換えるパラメータの行かどうかをチェックする*/
        char *str = strstr(confstr, com_name);
        /*書き換えるパラメータの行でなければそのまま出力する*/
        if(str == NULL){
            fprintf(fp2, "%s", confstr);
        }/*書き換えるパラメータがあった場合、書き換える*/
        else{
            fprintf(fp2, "%s %d¥n", com_name, bright);
            bl_flag = 1;
        }
    }
    if(bl_flag == 0){
        /*書き換えるパラメータが存在しなかった場合、尾末に加える*/
        fprintf(fp2, "%s %d¥n", com_name, bright);
    }

    fclose(fp);
    fclose(fp2);
    /*リネームする*/
    if(remove(fname_org) == 0){
        printf("remove %s¥n", fname_org);
    }
    else{
        printf("remove error¥n");
    }
    if(rename(fname_new, fname_org) == 0){
        printf("rename %s to %s¥n", fname_new, fname_org);
    }
    else{
        printf("rename error¥n");
    }
    printf("closed");
    exit(0);
}
static WSCfunctionRegister  op("BackLightChange",(void*)BackLightChange);
```

３－９－４　T-Kernelベースサンプル

「sample17」に、T-Kernel ベースのアプリケーションを起動するサンプルソースが入っています。リスト 3-9-4-1 にソースコードを示します。

リスト 3-9-4-1. T-Kernel ベースアプリのソースコード

```
#include "common.h"

/*
 * ドライバ入り口
 */
EXPORT ER main( INT ac, UB *av[] )
{
    ER er;

    if ( ac >= 0 ) {
            printf("Hello world! ¥n");
    }else if ( ac < 0 ) {
    /* 終了処理 */
            printf("finish ¥n");
    }

    return E_OK;
}
```

T-kernel ベースアプリの起動・終了手順は以下の通りです。

```
[/SYS]% lodspg sample17
Hello world!
SYSPRG sample17 [15] 405c2000 - 405c7000
[/SYS]% unlspg <システムプログラム ID>
finish
```

現在ロードされている T-Kernel ベースのプログラムと ID の一覧は「ref spg」で参照することができます。
T-Kernel ベースプログラムをロードしたときに表示される、

```
SYSPRG sample17 [15] 405c2000 - 405c7000
```

というメッセージは、sample17 が ID15 として 405c2000 から 405c7000 にロードされたことを表します。

３－９－５　デバイスドライバ作成サンプル

「sample18」に SDI,GDI それぞれのデバイスドライバのサンプルソースが入っています。
これらのサンプルをリスト 3-9-5-1、リスト 3-9-5-2 に示します。
これらのサンプルは、標準的なデバイスドライバの機能（Open、Close、read、write）についてのベースとしてもご利用いただけます。

リスト 3-9-5-1. 単純デバイスドライバ I/F 層を使用したサンプルドライバ

```
/*
 *    デバイスドライバ サンプルプログラム
 *    単純デバイスドライバ I/F 層を使ったサンプル
 *
 *    (C) Copyright 2002 by Personal Media Corporation
 */

#include "common.h"
#include <device/sdrvif.h>

static TData1 data1;
static TData2 data2;
static SDI    sdi;

/*
    オープン要求の処理
*/
LOCAL ER open_fn( ID devid, UINT omode, SDI sdi)
{
    /* デバイスの使用開始の処理を行う */
    /*（このサンプルでは何もしない）*/

    return E_OK;
}

/*
    クローズ要求の処理
*/
LOCAL ER close_fn( ID devid, UINT option, SDI sdi)
{
    /* デバイスの使用終了の処理を行う */
    /*（このサンプルでは何もしない）*/

    return E_OK;
}

/*
    入力処理
*/
#define MAX_R 8
#define INPUT( v ) (v) = i * i
```

```
LOCAL INT read_fn( ID devid, INT start, INT size, VP buf, SDI sdi )
{
    int i;

    if (start >= 0) { /* 固有データの読込み */

        if (size == 0) {
            return MAX_R; /* 読込みは行わず、現時点で読込み可能なサイズを返す */
        }

        /* 読込みを行う */
        for(i = 0; i < size && i < MAX_R; i++) {
            INPUT( ((char *)buf)[i] );
        }
        return i; /* 読込んだサイズを返す */

    } else {
        switch ( start ) { /* 属性データの読込み */

        case DN_DATA1 :

            if (size == 0) {
                return sizeof(TData1); /* 読込みは行わず、サイズを返す */
            }

            memcpy( buf, &data1, sizeof(TData1) );
            return sizeof(TData1); /* 読込んだサイズを返す */

        case DN_DATA2 :

            if (size == 0) {
                return sizeof(TData2); /* 読込みは行わず、サイズを返す */
            }

            memcpy( buf, &data2, sizeof(TData2) );
            return sizeof(TData2); /* 読込んだサイズを返す */

        default :
            return E_PAR; /* 属性データ番号エラー */
        }
    }
}
#undef MAX_R
#undef INPUT

/*
    出力処理
*/
#define MAX_W 4
#define OUTPUT( v )
LOCAL INT write_fn( ID devid, INT start, INT size, VP buf, SDI sdi )
```

```
{
    int i;

    if (start >= 0) { /* 固有データの書込み */

        if (size == 0) {
            return MAX_W; /* 書込みは行わず、現時点で書込み可能なサイズを返す */
        }

        /* 書込みを行う */
        for(i = 0; i < size && i < MAX_W; i++) {
            OUTPUT( ((char *)buf)[i] );
        }
        return i; /* 書込んだサイズを返す */

    } else switch ( start ) { /* 属性データの書込み */

    case DN_DATA1 :

        memcpy( &data1, buf, sizeof(TData1) );
        return sizeof(TData1); /* 書込んだサイズを返す */

    case DN_DATA2 :

        memcpy( &data2, buf, sizeof(TData2) );
        return sizeof(TData2); /* 書込んだサイズを返す */

    default :
        return E_PAR; /* 属性データ番号エラー */
    }
}
#undef MAX_W
#undef OUTPUT

/*
    イベント処理
*/
LOCAL INT event_fn( INT evttyp, VP evtinf, SDI sdi )
{
    switch ( evttyp ) {

    case TDV_SUSPEND :
        /* サスペンド状態へ移行する処理 */
        /* (このサンプルでは何もしない) */
        return E_OK;

    case TDV_RESUME:
        /* リジューム処理 */
        /* (このサンプルでは何もしない) */
        return E_OK;
```

```
    default : /* コマンドエラー */
        return E_PAR;
    }
}

/*
 * デバイス登録
 */
EXPORT ER def_sdr( void )
{
    SDefDev ddev;
    T_IDEV    idev;
    ER er;
    char *p;

    /* デバイス登録 */
    ddev.exinf = NULL;      /* exinf 拡張情報 */
    p = ddev.devnm; *p++ = 'd'; *p++ = 's';
      *p = '¥0';            /* devnm 物理デバイス名 */
    ddev.drvatr = 0;        /* drvatr ドライバ属性  */
    ddev.devatr = 0;        /* devatr デバイス属性  */
    ddev.nsub   = 0;        /* nsub サブユニット数  */
    ddev.blksz  = 1;        /* blksz 固有データのブロックサイズ */
    ddev.open   = open_fn;  /* open 関数 */
    ddev.close  = close_fn; /* close 関数 */
    ddev.read   = read_fn;  /* read 関数 */
    ddev.write  = write_fn; /* write 関数 */
    ddev.event  = event_fn; /* event 関数 */

    // SDI 登録実行
    er = SDefDevice( &ddev, &idev, &sdi );
    if (er < E_OK) {
        return er;
    }

    return E_OK;
}
```

リスト 3-9-5-2. 汎用デバイスドライバ I/F 層を使用したサンプルドライバ

```
/*
 *    デバイスドライバ サンプルプログラム
 *    汎用デバイスドライバ I/F 層を使ったサンプル
 *
 *    (C) Copyright 2002 by Personal Media Corporation
 */

#include "common.h"
#include <device/gdrvif.h>

typedef struct {
    GDI      Gdi;     /* 汎用ドライバ I/F ハンドル */
    T_DEVREQ *devReq; /* 実行中の要求（アボート対象となる要求）*/
    ID       tskid;   /* タスク ID */
} TInfo;
TInfo info[4];

static TData1 data1;
static TData2 data2;

/*
    オープン要求の処理
*/
LOCAL ER open_fn( ID devid, UINT omode, GDI gdi )
{
    /* デバイスの使用開始の処理を行う */
    /*（このサンプルでは何もしない）*/

    return E_OK;
}

/*
    クローズ要求の処理
*/
LOCAL ER close_fn( ID devid, UINT option, GDI gdi)
{
    /* デバイスの使用終了の処理を行う */
    /*（このサンプルでは何もしない）*/

    return E_OK;
}

/*
    アボート処理
*/
LOCAL ER abort_fn( T_DEVREQ *devReq, GDI gdi )
{
    TInfo *pInfo; ER er;
```

```
    pInfo = GDI_exinf( gdi );
    if ( pInfo -> devReq != NULL && pInfo -> devReq == devReq ) {
        /* アボート実行 */
        /*（このサンプルでは要求受付タスクの待ち解除を行う）*/

        er = tk_rel_wai( pInfo -> tskid );
        if (er < E_OK) {
            return er;
        }
    }

    return E_OK;
}

/*
    イベント処理
*/
LOCAL INT event_fn( INT evttyp, VP evtinf, GDI gdi )
{
    switch ( evttyp ) {

    case TDV_SUSPEND :
        /* サスペンド状態へ移行する処理 */
        /*（このサンプルでは何もしない）*/
        return E_OK;

    case TDV_RESUME:
        /* リジューム処理 */
        /*（このサンプルでは何もしない）*/
        return E_OK;

    default : /* コマンドエラー */
        return E_PAR;
    }
}

/*
 * データ読込み
 */
#define MAX_R 8
#define INPUT( v ) (v) = i * i
LOCAL ER readData( T_DEVREQ *devReq )
{
    int i; ER er;

    if (devReq -> start >= 0) { /* 固有データの読込み */

        if (devReq -> size == 0) {
            return MAX_R; /* 読込みは行わず、現時点で読込み可能なサイズを返す */
        }
```

```
        /* アボート処理のサンプルのため、処理に時間がかかるように待つ */
        er = tk_dly_tsk( 100 );
        if (er < E_OK) {
            return E_ABORT;
        }

        /* 読込みを行う */
        for(i = 0; i < devReq -> size && i < MAX_R; i++) {
            INPUT( ((char *)(devReq -> buf))[i] );
        }
        devReq -> asize = i;
        return E_OK;

    } else switch ( devReq -> start ) { /* 属性データの読込み */

    case DN_DATA1 :

        devReq -> asize = sizeof(TData1);
        if (devReq -> size == 0) {
            break; /* 読込みは行わず、サイズを返す */
        }
        memcpy( devReq -> buf, &data1, sizeof(TData1) );
        break;

    case DN_DATA2 :

        devReq -> asize = sizeof(TData2);
        if (devReq -> size == 0) {
            break; /* 読込みは行わず、サイズを返す */
        }
        memcpy( devReq -> buf, &data2, sizeof(TData2) );
        break;

    default :
        return E_PAR; /* 属性データ番号エラー */
    }

    return E_OK;
}
#undef MAX_R
#undef INPUT

/*
 * データ書込み
 */
#define MAX_W 4
#define OUTPUT( v )
LOCAL ER writeData( T_DEVREQ *devReq )
{
    int i; ER er;
```

```
    if (devReq -> start >= 0) { /* 固有データの書込み */

        if (devReq -> size == 0) {
            return MAX_W; /* 書込みは行わず、現時点で書込み可能なサイズを返す */
        }

        /* アボート処理のサンプルのため、処理に時間がかかるように待つ */
        er = tk_dly_tsk( 100 );
        if (er < E_OK) {
            return E_ABORT;
        }

        /* 書込みを行う */
        for(i = 0; i < devReq -> size && i < MAX_W; i++) {
            OUTPUT( ((char *)(devReq -> buf))[i] );
        }
        devReq -> asize = i;
        return E_OK;

    } else switch ( devReq -> start ) { /* 属性データの書込み */

    case DN_DATA1 :

        devReq -> asize = sizeof(TData1);
        if (devReq -> size == 0) {
            break; /* 書込みは行わず、サイズを返す */
        }
        memcpy( &data1, devReq -> buf, sizeof(TData1) );
        break;

    case DN_DATA2 :

        devReq -> asize = sizeof(TData2);
        if (devReq -> size == 0) {
            break; /* 書込みは行わず、サイズを返す */
        }
        memcpy( &data2, devReq -> buf, sizeof(TData2) );
        break;

    default :
        return E_PAR; /* 属性データ番号エラー */
    }

    return E_OK;
}
#undef MAX_W
#undef OUTPUT

/*
 * ドライバ・メインタスク (read/write 要求受付タスク)
```

```
 */
LOCAL void MainTask( TInfo *pInfo, VP exinf )
{
    T_DEVREQ *devReq;
    ER er;

    for ( ;; ) { /* 要求受付の無限ループ */
        /* 要求受付 */
        /* DRP_NORMREQ の設定により、read,write 要求を受け付ける */
        er = GDI_Accept( &devReq, DRP_NORMREQ, TMO_FEVR, pInfo -> Gdi );
        if ( er < E_OK ) {
            continue;
        }

        /* 読込み/書込み要求の処理 */
        pInfo -> devReq = devReq; /* アボート対象設定 */
        switch ( devReq -> cmd ) {
        case TDC_READ :
            er = readData( devReq );
            break;
        case TDC_WRITE :
            er = writeData( devReq );
            break;
        }
        pInfo -> devReq = NULL; /* アボート対象解除 */

        /* 返答 */
        devReq -> error = er;
        GDI_Reply( devReq, pInfo -> Gdi );
    }
}

/*
 * デバイス登録
 */
EXPORT ER def_gdr( int n )
{
    ER er, tid;
    char *p;
    TInfo *pInfo;
    GDefDev ddev;
    T_IDEV idev;
    T_CTSK ctsk;

    /* デバイス登録 */
    pInfo = &(info[n]);
    ddev.exinf   = pInfo;        /* exinf 拡張情報 */
    p = ddev.devnm;              /* devnm 物理デバイス名 */
    *p++ = 'd'; *p++ = 'g'; *p++ = 'a' + n; *p = '¥0';
    ddev.maxreqq = 1;            /* maxreqq 要求をキューイングする最大数 */
    ddev.drvatr  = 0;            /* drvatr ドライバ属性 */
```

```
    ddev.devatr  = 0;           /* devatr デバイス属性 */
    ddev.nsub    = 0;           /* nsub サブユニット数 */
    ddev.blksz   = 1;           /* blksz 固有データのブロックサイズ */
    ddev.open    = open_fn;     /* open 関数 */
    ddev.close   = close_fn;    /* close 関数 */
    ddev.abort   = abort_fn;    /* abort 関数 */
    ddev.event   = event_fn;    /* event 関数 */

    er = GDefDevice( &ddev, &idev, &(pInfo -> Gdi) ); /* GDI 登録実行 */
    if (er < E_OK) {
        return er;
    }

    /* ドライバ本体の生成・起動 */
    memcpy( &(ctsk.exinf), ddev.devnm, sizeof(VP) );
    ctsk.task    = MainTask;
    ctsk.itskpri = TaskPri; /* タスク優先度 */
    ctsk.stksz   = 4 * 1024;
    ctsk.tskatr  = TA_HLNG | TA_RNG0;

    er = tid = tk_cre_tsk( &ctsk );         /* 生成 */
    if (er < E_OK) {
        return er;
    }

    er = tk_sta_tsk( tid, (int)(pInfo) );   /* 起動 */
    if ( er < E_OK ) {
        return er;
    }

    pInfo -> tskid = tid;
    return E_OK;
}
```

３－１０　コンソールアプリケーション

T-Kernel 版 WideStudio ではコンソールアプリケーションの作成ができません。T-Kernel 上でコンソールアプリケーションを作成する場合は MakeFile を作成し、開発環境上コンソールでコンパイルする必要があります。「sample19」にコンソールアプリケーション作成用のプログラムと MakeFile が入っています。コンソールアプリケーションを作成するときの雛形として使用してください。
コンソールアプリケーションの詳しい作成方法については『２－９ WideStudio を使用しないコンソールアプリケーション開発』を参照してください。

リスト 3-10-1. コンソールアプリケーション雛形

```
/*
    Console Aplication Sample
*/
#include    <btron/outer.h>

EXPORT  W   main(W ac, B *av[])
{
//ここにプログラムを記述してください
    return 0;
}
```

３－１１　システムの復旧

システムに異常が発生した場合など、システムの復旧が必要な場合、以下の手順でシステムの復旧を行うことができます。以下の説明は例として NAND Flash ROM の復旧の手順を示しています。

(1) フォーマット済みの CF カードを挿入して Algo Smart Panel の電源を入れ、T-Monitor を起動します。このときの CF カードのフォーマット形式に制限はありません。

(2) 起動ディスク/SYS が ROM ディスク(rda)になっていることを確認してください。

```
[/SYS]% df
PATH  DEV      TOTAL      FREE USED  UNIT MAXFILE NAME
/SYS  rda       7250         1  99%  1024    256  SYSTEM
```

(3) 区画設定を行います。
すでに区画設定してあれば、再度設定し直す必要はありません。
操作例なので、ここではいったん区画をすべて削除してから再設定しています。

```
[/SYS]% hdpart fla
fla   [C:990  H:32  S:63  B:1998848 (976 MB)]
No System    Boot   StartCHS      EndCHS      SecNo    SecCnt     Size
1  13 BTRON  80     0:  1: 1   124:107:47       63   1998785   975 MB
2  00 ------ 00     0:  0: 0     0:  0: 0        0         0     0 KB
3  00 ------ 00     0:  0: 0     0:  0: 0        0         0     0 KB
4  00 ------ 00     0:  0: 0     0:  0: 0        0         0     0 KB
** Create/Delete/Boot/Edit/Quit ? d
Delete PartNo (1-4,All) ? a
No System    Boot   StartCHS      EndCHS      SecNo    SecCnt     Size
1  00 ------ 00     0:  0: 0     0:  0: 0        0         0     0 KB
2  00 ------ 00     0:  0: 0     0:  0: 0        0         0     0 KB
3  00 ------ 00     0:  0: 0     0:  0: 0        0         0     0 KB
4  00 ------ 00     0:  0: 0     0:  0: 0        0         0     0 KB
** Create/Delete/Boot/Edit/Update/Quit ? c
Create PartNo (1-4) ? 1
Size [GB/MB/KB,All] (<976MB) ? a
No System    Boot   StartCHS      EndCHS      SecNo    SecCnt     Size
1  13 BTRON  00     0:  1: 1   991: 15:47       63   1998785   975 MB
2  00 ------ 00     0:  0: 0     0:  0: 0        0         0     0 KB
3  00 ------ 00     0:  0: 0     0:  0: 0        0         0     0 KB
4  00 ------ 00     0:  0: 0     0:  0: 0        0         0     0 KB
** Create/Delete/Boot/Edit/Update/Quit ? b
Boot PartNo (1-4,Clear) ? 1
No System    Boot   StartCHS      EndCHS      SecNo    SecCnt     Size
1  13 BTRON  80     0:  1: 1   991: 15:47       63   1998785   975 MB
2  00 ------ 00     0:  0: 0     0:  0: 0        0         0     0 KB
3  00 ------ 00     0:  0: 0     0:  0: 0        0         0     0 KB
4  00 ------ 00     0:  0: 0     0:  0: 0        0         0     0 KB
** Create/Delete/Boot/Edit/Update/Quit ? u
** fla: Updated Master Boot Block
```

ブート(Boot)の設定を忘れずに行ってください。

つぎに、設定した区画をフォーマットします。

```
[/SYS]% format -b fla0 SYSTEM
Format fla0 [STD] SYSTEM
Logical Formatting...
Writing BootCode...
Disk Format Success.
```

(4) ROM ディスクの内容を NAND Flash へコピーします。

```
[/SYS]% att fla0 A
fla0 -> A
[/SYS]% rcp -r /SYS /A/=
[/SYS]% det fla0
```

(5) システムを再起動する。
再起動したら、起動ディスク /SYS が NAND Flash (fla0) になっていることをご確認ください。

```
[/SYS]% df
PATH  DEV      TOTAL      FREE USED  UNIT MAXFILE NAME
/SYS  fla0    999392    991132   0%  4096   65535  SYSTEM
```

(6) 開発環境からドライバをコピーします。
開発環境の/usr/local/te-asd_sh7760/kernel/config/asd_sh7760/SYS 内のファイルを全て Algo Smart Panel 内/SYS へ転送します。
ファイル転送の方法については『２-７-９ファイルの転送』を参照してください。

(7) システムを再起動します。
再起動したら、devlist と ls コマンドでドライバがすべて組み込まれていることをご確認ください。

以上で、インストールは完了です。

３－１２　設定ファイルについて

各種設定ファイルについて代表的なものの設定例について説明します。

３－１２－１ /SYS/STARTUP.CMD

リスト 3-12-1-1. アプリケーション自動起動(IMS)設定ファイル

```
lodspg   screen     !35
lodspg   kbpd       !30
lodspg   lowkbpd    !28
lodspg   rsdrv      !26
lodspg   netdrv     !23
lodspg   unixemu
lodspg   tcpipmgr
lodspg   font       !120
lodspg   dp
lodspg   tip
lodspg   hmi
lodspg   omgr
$$sysdmn            !112 &
$logon
$cli    STARTUP.CLI !224
$logon  E
exit    1
```

PMC T-Kernel 起動時に自動起動するプログラム名を設定します。「!**」(**は数値)で優先度を設定します。STARTUP.CMD に記述することで自動起動できるプログラムは T-Kernel ベースアプリケーション、デバイスドライバ、サブシステムに限られます。

３－１２－２ /SYS/STARTUP.CLI

リスト 3-12-2-1. アプリケーション自動起動(CLI)設定ファイル

```
/SYS/.xcli
/SYS/ftpd &
/SYS/telnetd -S -s -e &
*
alias do /SYS/$$PROGRAM.BOX/DLED /SYS/USR 0
*
if &DBG == 0
    do
    exit
else
    do &
endif
```

PMC T-Kernel 起動時に自動起動するプログラム名を設定します。STARTUP.CLI に記述することで、T-Kernel ベースアプリケーション、デバイスドライバ、サブシステム以外のアプリケーション(プロセスベースなど)を自動起動することができます。

３－１３　リムーバブルディスク機器の取り扱いについて

ここでは、USB メモリや SD カードのようなリムーバブルディスクの使用方法について説明します。

３－１３－１ 使用方法

リムーバブルディスク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出来ます。

- リムーバブルディスクのマウント
  リムーバブルディスクのブロックデバイスをマウントします。

  例: USB メモリのマウント(デバイスドライバ uda0)

```
[/SYS]% att -m uda0 uda0
```

- リムーバブルディスク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リムーバブルディスク内のファイルにアクセスができます。リムーバブルディスク内のファイルシステムが tron 以外の場合は通常のファイル操作方法の先頭に『ux/』を追加する必要があります。

  例: USB メモリ内ファイル一覧表示

```
[/SYS]% ux/ls /uda0/
file1     file2     file3
```

  例: USB メモリからのファイルコピー

```
[/SYS]% ux/cp /uda0/file1 .
```

- リムーバブルディスクのアンマウント
  リムーバブルディスク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。リムーバブルディスクを抜く前に必ずアンマウントを行ってください。

  例: USB メモリのアンマウント

```
[/SYS]%  det –u uda0
```

第４章　付録

４－１　マウスカーソルを非表示にする方法について

プログラム内で以下のようなコマンドを入力することでマウスポインタを非表示にすることができます。

```
#include<btron/dp.h>

gdsp_ptr(0);
```

再度表示したい場合は以下のようにしてください。

```
gdsp_ptr(1);
```

４－２　日付表示を非表示にする方法について

画面右下の日付表示の表示/非表示を切り替えるためには、SYSCONF 内の WinfVal で設定します。

日付表示有効：0x02　（デフォルト）
日付表示無効：0x100

SYSCONF を書き換えるには、sysconf コマンドを使用します。

```
[/SYS]% sysconf WinfVal 0x100
```

設定後 OS を終了し、再起動することで有効になります

４－３　　参考文献

●「WideStudio 徹底ガイドブック」
　　監修　　坂村 健
　　編著　　平林 俊一
　　共著　　後藤 渉
　　末竹 弘之
　　川上 正平
　　平林 洋介
　　発行所　　パーソナルメディア
　　発行年　　2004 年

●「T-Kernel 標準ハンドブック」
　　監修　　板村 健
　　編著者　　T-Engine フォーラム
　　発行所　　パーソナルメディア株式会社
　　発行年　　2005 年

●「T-Kernel 組み込みプログラミング 強化書」
　　監修　　板村 健
　　編著者　　パーソナルメディア株式会社
　　発行所　　パーソナルメディア株式会社
　　発行年　　2007 年

本 DVD にはパーソナルメディア社提供の T-Kernel に関するマニュアルも収録しております。
PMC T-Kernel の使用方法や組み込まれているドライバなどについての詳細はそちらを参照してください。
各マニュアルは<DVD>¥doc¥PMC-TKernel¥に収録されています。

このユーザーズ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ください。
(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ください。その際、巻末記載の書籍番号も併せてお知らせください。

７７Ｔ０１００００５Ｄ　２００９年　１１月　第４版
７７Ｔ０１００００５Ａ　２００９年　８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

東京支社
〒１０４－００６１　東京都中央区銀座７－１５－８　銀座堀ビル２Ｆ　ＴＥＬ(０３)３５４１－７１７０
ＦＡＸ(０３)３５４１－７１７５

大阪支社
〒５４２－００８１　大阪府大阪市中央区南船場１－１２－３　船場グランドビル３Ｆ　ＴＥＬ(０６)６２６３－９５７５
ＦＡＸ(０６)６２６３－９５７６

名古屋営業所
〒４６１－０００４　愛知県名古屋市東区葵２－３－１５　ふぁみ一ゆ葵ビル５０３　ＴＥＬ(０５２)９３９－５３３３
ＦＡＸ(０５２)９３９－５３３０

ホームページ　http://www.algosystem.co.jp