ALGO

マニュアル

産業用高機能パネル PC FP4-*****シリーズ用**

『Windows Embedded Standard 7 32 ビット版』

について

目　次

はじめに

１）　お願いと注意 ………… １
２）　対応機種について ………… １
３）　バンドル製品について ………… １

第１章　概要

１－１ 機能と特長 ………… １－１
１－１－１　FP シリーズ用 Windows Embedded Standard 7 32 ビット版とは ………… １－１
１－１－２　機能と特長 ………… １－１
１－２ システム構成 ………… １－３
１－２－１　ドライブ構成 ………… １－３
１－２－２　フォルダ/ファイル構成 ………… １－３
１－２－３　ユーザーアカウント ………… １－３
１－２－４　コンピューター名 ………… １－４
１－３ アプリケーション開発と実行 ………… １－５

第２章　システムの操作

２－１ OS の起動と終了 ………… ２－１
２－１－１　OS の起動 ………… ２－１
２－１－２　OS の終了 ………… ２－１
２－２ 外部 RTC ………… ２－２
２－２－１　RTC とシステム時刻について ………… ２－２
２－２－２　外部 RTC によるシステム時刻更新機能 ………… ２－２
２－２－３　日付と時刻の設定 ………… ２－２
２－３ EWF 機能 ………… ２－３
２－３－１　EWF とは ………… ２－３
２－３－２　ドライブと EWF 設定 ………… ２－４
２－３－３　EWF の設定方法 ………… ２－４
２－３－４　EWF を使用するにあたっての注意事項 ………… ２－８

２－４‐ログオン設定 ･･･ ２－１１
２－４－１　自動ログオン設定 ･･･ ２－１１
２－５‐言語設定 ･･･ ２－１２
２－５－１　マルチ言語機能 ･･･ ２－１２
２－５－２　言語の変更方法 ･･･ ２－１２
２－６‐モニタ設定 ･･･ ２－１３
２－６－１　マルチモニタ機能 ･･･ ２－１３
２－６－２　モニタ設定の変更方法 ･･･ ２－１３
２－７‐タッチパネル設定 ･･･ ２－１８
２－７－１　モニタ構成の設定 ･･･ ２－１８
２－７－２　タッチパネルの調整 ･･･ ２－１９
２－７－３　タッチ音について ･･･ ２－２１
２－８‐タッチパネルインターロック機能（AT シリーズの接続） ･･･ ２－２２
２－８－１　AT シリーズの接続 ･･･ ２－２２
２－８－２　タッチパネルデバイスの追加 ･･･ ２－２２
２－８－３　タッチパネルインターロック ･･･ ２－２３
２－９‐サービス設定 ･･･ ２－２５
２－９－１　サービス設定の変更 ･･･ ２－２５
２－１０　ASD Config Tool ･･･ ２－２６
２－１０－１　ASD Config Tool ･･･ ２－２６
２－１０－２　Serial Port Setting ･･･ ２－２６
２－１０－３　Touch Panel Setting ･･･ ２－２７
２－１０－４　Board Information ･･･ ２－２８
２－１０－５　初期値 ･･･ ２－２８
２－１１　EWF Config Tool ･･･ ２－２９
２－１１－１　EWF Config Tool ･･･ ２－２９
２－１１－２　Volume Infomation ･･･ ２－２９
２－１１－３　EWF Configuration ･･･ ２－３０
２－１２　Ramdisk Drive Config Tool ･･･ ２－３１
２－１２－１　Ramdisk Drive Config Tool ･･･ ２－３１
２－１２－２　Ramdisk Drive Configuration ･･･ ２－３１
２－１２－３　初期値 ･･･ ２－３１
２－１３　RAS Config Tool ･･･ ２－３２
２－１３－１　RAS Config Tool ･･･ ２－３２
２－１３－２　Temperature ･･･ ２－３３

２－１３－３　Temperature Configuration…… ２－３４
２－１３－４　Watchdog Timer …… ２－３５
２－１３－５　Watchdog Timer Configuration…… ２－３７
２－１３－６　Secondary RTC …… ２－３８
２－１３－７　Secondary RTC Configuration…… ２－３９
２－１３－８　Wake On Rtc Timer 設定例…… ２－４０
２－１３－９　Backup Battery Monitor…… ２－４１
２－１３－１０　初期値 …… ２－４２
２－１４　ユーザーアカウント制御…… ２－４３
２－１５　S.M.A.R.T.機能…… ２－４４

第３章　産業用高機能パネル PC FP4-*******について

３－１ 産業用高機能パネル PC FP4-*******に搭載された機能について…… ３－１
３－２ Windows 標準インターフェース対応機能 …… ３－３
３－２－１　グラフィック …… ３－３
３－２－２　LCD 輝度調整…… ３－３
３－２－３　タッチパネル …… ３－３
３－２－４　シリアルポート …… ３－３
３－２－５　有線 LAN…… ３－４
３－２－６　サウンド …… ３－４
３－２－７　USB ポート…… ３－４
３－２－８　PCI 拡張スロット…… ３－４
３－２－９　PCI-e 拡張スロット…… ３－４
３－３ 組込みシステム機能 …… ３－５
３－３－１　タイマ割込み機能 …… ３－６
３－３－２　汎用入出力 …… ３－６
３－３－３　RAS 機能…… ３－６
３－３－４　シリアルコントロール機能…… ３－６
３－３－５　バックアップ SRAM…… ３－６
３－３－６　ハードウェア・ウォッチドッグタイマ機能 …… ３－６
３－３－７　ソフトウェア・ウォッチドッグタイマ機能 …… ３－６
３－３－８　RAM ディスク…… ３－７
３－３－９　外部 RTC 機能 …… ３－９
３－３－１０　Wake On Rtc Timer 機能…… ３－９

３－３－１１　温度監視機能 ･･･ ３－９
３－３－１２　停電検出機能 ･･･ ３－９
３－３－１３　バックアップバッテリモニタ･･･ ３－９

第４章　EWF API

４－１ EWF APIについて･･･ ４－１
４－２ EWF API関数リファレンス･･･ ４－２
４－３ EWF API関数の使用について･･･ ４－１７
４－４ サンプルコード ･･･ ４－１７
４－４－１　EWFの有効／無効･･ ４－１７
４－４－２　オーバーレイデータのコミット･･･ ４－２０

第５章　組込みシステム機能ドライバ

５－１ ドライバの使用について ･･･ ５－１
５－１－１　開発用ファイル ･･･ ５－１
５－１－２　DeviceIoControlについて･･･ ５－２
５－２ タイマ割込み機能 ･･･ ５－３
５－２－１　タイマ割込み機能について･･･ ５－３
５－２－２　タイマドライバについて･･･ ５－３
５－２－３　タイマデバイス ･･･ ５－４
５－２－４　タイマドライバの動作 ･･･ ５－５
５－２－５　ドライバ使用手順 ･･･ ５－６
５－２－６　DeviceIoControlリファレンス ･･･ ５－７
５－２－７　サンプルコード ･･･ ５－１１
５－３ 汎用入出力･･･ ５－１６
５－３－１　汎用入出力について ･･･ ５－１６
５－３－２　汎用入出力ドライバについて･･･ ５－１７
５－３－３　汎用入出力デバイス ･･･ ５－１８
５－３－４　DeviceIoControlリファレンス ･･･ ５－１９
５－３－５　サンプルコード ･･･ ５－２１
５－４ RAS機能･･･ ５－２６
５－４－１　RAS機能について･･･ ５－２６
５－４－２　RAS-INドライバについて ･･･ ５－２６

５－４－３　RAS-IN デバイス･････ ５－２７
５－４－４　IN1 割込みの使用手順･････ ５－２８
５－４－５　複数アプリケーションで IN1 割込み発生時のイベントを同時に使用する場合･････ ５－２９
５－４－６　DeviceIoControl リファレンス･････ ５－３０
５－４－７　サンプルコード･････ ５－３６
５－５ シリアルコントロール機能･････ ５－４６
５－５－１　シリアルコントロール機能について･････ ５－４６
５－５－２　シリアルコントロールドライバについて･････ ５－４６
５－５－３　SciCtl デバイス･････ ５－４７
５－５－４　DeviceIoControl リファレンス･････ ５－４８
５－５－５　サンプルコード･････ ５－５０
５－６ バックアップ SRAM･････ ５－５３
５－６－１　バックアップ SRAM について･････ ５－５３
５－６－２　SRAM ドライバについて･････ ５－５３
５－６－３　SRAM デバイス･････ ５－５４
５－６－４　DeviceIoControl リファレンス･････ ５－５５
５－６－５　サンプルコード･････ ５－５６
５－７ ハードウェア・ウォッチドッグタイマ機能･････ ５－５９
５－７－１　ハードウェア・ウォッチドッグタイマ機能について･････ ５－５９
５－７－２　ハードウェア・ウォッチドッグタイマドライバについて･････ ５－５９
５－７－３　ハードウェア・ウォッチドッグタイマデバイス･････ ５－６１
５－７－４　DeviceIoControl リファレンス･････ ５－６３
５－７－５　サンプルコード･････ ５－６８
５－８ ソフトウェア・ウォッチドッグタイマ機能･････ ５－７５
５－８－１　ソフトウェア・ウォッチドッグタイマ機能について･････ ５－７５
５－８－２　ソフトウェア・ウォッチドッグタイマドライバについて･････ ５－７５
５－８－３　ソフトウェア・ウォッチドッグタイマデバイス･････ ５－７６
５－８－４　DeviceIoControl リファレンス･････ ５－７８
５－８－５　サンプルコード･････ ５－８３
５－９ RAS 監視機能･････ ５－９０
５－９－１　RAS 監視機能について･････ ５－９０
５－９－２　RAS DLL について･････ ５－９０
５－９－３　RAS DLL I/F 関数リファレンス･････ ５－９１
５－９－４　サンプルコード･････ ５－９２
５－１０　外部 RTC 機能･････ ５－９６

５－１０－１　外部 RTC 機能について･･ ５－９６
５－１０－２　RAS DLL について･･ ５－９６
５－１０－３　RAS DLL 時刻設定関数リファレンス･･ ５－９７
５－１０－４　RAS DLL Wake On Rtc Timer 設定関数リファレンス･････････････････････････････ ５－９８
５－１０－５　サンプルコード ･･ ５－１０１
５－１１　停電検出機能･･･ ５－１０６
５－１１－１　停電検出機能について･･ ５－１０６
５－１１－２　停電検出ドライバについて･･･ ５－１０６
５－１１－３　停電検出デバイス ･･ ５－１０７
５－１１－４　DeviceIoControl リファレンス ･･･ ５－１０８
５－１１－５　サンプルコード ･･ ５－１１０
５－１２　バックアップバッテリモニタ ･･･ ５－１１４
５－１２－１　バックアップバッテリモニタについて･･･ ５－１１４
５－１２－２　バックアップバッテリモニタドライバについて ･････････････････････････････････ ５－１１４
５－１２－３　バックアップバッテリモニタデバイス･･･ ５－１１５
５－１２－４　DeviceIoControl リファレンス ･･･ ５－１１６
５－１２－５　サンプルコード ･･ ５－１１７

第６章　システムリカバリ

６－１ リカバリ DVD について ･･･ ６－１
６－１－１　リカバリ DVD 起動 ･･･ ６－２
６－１－２　リカバリ作業 ･･･ ６－５
６－１－３　リカバリ後処理 ･･･ ６－６
６－２ システムの復旧（工場出荷状態） ･･･ ６－７
６－３ システムのバックアップ ･･･ ６－１１
６－４ システムの復旧（バックアップデータ） ･･･････････････････････････････････････ ６－１５

付録

A－1　マイクロソフト製品の組込み用 OS（Embedded）について･･････････････････････････ １

はじめに

この度は、アルゴシステム製品をお買い上げいただきありがとうございます。

弊社製品を安全かつ正しく使用していただくために、お使いになる前に本書を十分に理解していただくようお願い申し上げます。

1）　お願いと注意

本書では、産業用高機能パネル PC FP4-*******シリーズ（以降「FP シリーズ」と表記します）用 Windows Embedded Standard 7 32 ビット版について説明します。

Windows Embedded Standard 7 は Windows Embedded Standard 2009 の後継製品で、Windows 7 をベースとした組込み用 OS です。Windows Embedded Standard 7 には 32 ビット版と 64 ビット版があります。本書では、FP シリーズ用の Windows Embedded Standard 7 32 ビット版に特有の仕様、操作について説明します。一般的な Windows の仕様、操作については省略させていただきます。

Windows Embedded Standard 7 は Windows 7 をベースとした OS ですが、組込み用 OS のため通常の PC 用 Windows とは動作が異なる可能性があります。詳しくは、「付録　マイクロソフト製品の組込み用 OS について」を参照してください。

本書は、アプリケーション開発、専用ドライバ仕様などの専門的な内容を含んでいます。これらの内容は、Windows アプリケーション開発、デバイス制御プログラミングに関する技術を必要とします。ご注意ください。

2）　対応機種について

本書では、FP シリーズについて説明しています。その他の機種については、それぞれの機種に対応するマニュアルを用意しております。機種に対応したマニュアルを参照してご使用ください。

3）　バンドル製品について

本書では、FP シリーズ用 Windows Embedded Standard 7 32 ビット版の標準品について説明しています。バンドル製品については、本書の説明と異なる箇所がある場合があります。詳しくは、バンドル製品の開発環境に含まれるドキュメントを参照してください。

第１章　概要

本章では、FP シリーズ用 Windows Embedded Standard 7 32 ビット版の概要について説明します。

１－１　　機能と特長

１－１－１　FP シリーズ用 Windows Embedded Standard 7 32 ビット版とは

Windows Embedded Standard 7 は、Windows Embedded Standard 2009 の後継製品です。Windows 7 をベースとしており、Windows オペレーティングシステムのデスクトップ技術を組込みデバイスで使用することが可能となります。Windows Embedded Standard 7 はコンポーネント形式で提供されます。必要な機能を選択し、組込みデバイスに合わせた Windows オペレーティングシステムを構築することができます。Windows Embedded Standard 7 には、Windows 7 と同様に 32 ビット版と 64 ビット版が存在します。

FP シリーズ用 Windows Embedded Standard 7 32 ビット版は、Windows Embedded Standard 7 32 ビット版を FP シリーズ用にカスタマイズしたものです。FP シリーズ用に選定したコンポーネント、オンボード搭載デバイス用のドライバおよび設定ツールで構成されています。

１－１－２　機能と特長

FP シリーズ用 Windows Embedded Standard 7 32 ビット版は、オンボードデバイスのサポートと Windows アプリケーション互換性のためのテンプレートを基本として構築されています。このため Windows 7 32 ビット版で動作しているアプリケーションをほぼそのまま動作させることが可能となっています。また、Enhanced Write Filter のストレージ保護機能、IIS などのネットワークサーバー機能を追加することにより、組込みシステムとしてより堅牢で柔軟なシステムを構築できるようになっています。

● 互換性確保のためのテンプレート
　Application Compatibility

表 1-1-2-1 に FP シリーズ用 Windows Embedded Standard 7 32 ビット版に搭載されている主な機能を示します。

表 1-1-2-1. FP シリーズ用 Windows Embedded Standard 7 32 ビット版の主な機能

機能	内容
Windows 7	Windows 7 がベースとなっています。
.NET Framework 3.5	.NET Framework 3.5 を搭載しています。 .NET Framework 2.0、3.0、3.5 を利用したアプリケーションを動作させることができます。
Internet Explorer 8	Internet Explorer 8 を搭載しています。 Web コンテンツの閲覧、Web アプリケーションへのアクセスが可能となります。
Windows Media Player 12	Windows Media Player 12 を搭載しています。 画像の閲覧、デジタルミュージック、動画の再生が可能です。
Internet Information Services (IIS)	Internet Information Services を搭載しています。 以下のサービスに対応しています。 ・ FTP サービス ・ Web サービス
Telnet Service	Telnet クライアント、Telnet サーバを搭載しています。
Remote Desktop	リモートデスクトッププロトコルに対応しています。 ターミナルサービス、リモートデスクトップを使用することができます。

Kernel Mode Driver Framework (KMDF)	Kernel Mode Driver Framework に対応しています。 KMDF を利用して開発されたカーネルモードデバイスドライバを動作させることができます。
Enhanced Write Filter (EWF)	Enhanced Write Filter を搭載しています。 EWF を用いるとドライブを書込み禁止状態にして OS を動作させることができます。EWF はドライブ毎に設定することができます。 書込み禁止状態で起動した OS は、シャットダウン処理なしで電源断することができます。(強制電源断)
Multi Language Support	マルチ言語対応。11 言語に対応しています。 設定によって切替えることができます。
Windows 標準インターフェース対応機能	Windows 標準インターフェースを使用して使用できる機能、デバイスを搭載しています。 ・ グラフィック ・ タッチパネル ・ モニタ出力(HDMI) ・ シリアルポート ・ 有線 LAN ・ サウンド(HDMI) ・ USB ポート ・ PCI 拡張スロット (※) ・ PCI-e 拡張スロット (※)
組込みシステム機能	組込みシステム向けの独自機能が搭載されています。 ・ タイマ割込み機能 ・ 汎用入出力 ・ RAS 機能 ・ シリアルコンロール機能(RS-232C/422/485 切替え) ・ バックアップ SRAM ・ ハードウェア・ウォッチドッグタイマ ・ ソフトウェア・ウォッチドッグタイマ ・ 外部 RTC ・ Wake On Rtc Timer 機能 ・ RAM ディスク ・ ASD Config(機能設定用コンパネアプリ) ・ 停電検出機能 ・ バックアップバッテリモニタ機能

※　PCI 拡張スロット、PCI-e 拡張スロットは、機種によってスロット数が異なります。詳細は、ハードウェアマニュアルを参照してください。

１－２　システム構成

１－２－１　ドライブ構成

OS を格納するメインストレージは、16GByte の mSATA SSD です。メインストレージには、C ドライブが割り当てられています。C ドライブはシステムドライブとして OS 本体を格納しています。ドライブ構成を表 1-2-1-1 に示します。

表 1-2-1-1. FP シリーズ ドライブ構成

ドライブ	容量	空き容量	内容
C	14.7 GByte	約 8.59 GByte	システムドライブ オペレーティングシステム本体を格納しています。

１－２－２　フォルダ/ファイル構成

Windows Embedded Standard 7 は Windows 7 がベースとなっています。システムドライブのフォルダ、ファイル構成は Windows 7 に準拠したものです。

ドライブトップに存在するフォルダ、ファイルの構成を表 1-2-2-1 に示します。

表 1-2-2-1. FP シリーズ Windows Embedded Standard 7 フォルダ/ファイル構成

ドライブ	フォルダ/ファイル（※）
C	<inetpub> <Intel> <PerfLogs> <Program Files> <Program Files (x86)> <Users> <Windows>

※ フォルダは<>で表記しています。システム属性、隠し属性のフォルダ/ファイルは表記していません。

１－２－３　ユーザーアカウント

Windows Embedded Standard 7 は、ログイン可能なユーザーが１つ必要です。初期状態ではログイン可能なユーザーアカウントは Administrator ユーザーのみとなっています。初期状態での Administrator ユーザーの状態を表 1-2-3-1 に示します。

パスワードの変更、別のユーザーアカウントが必要な場合は、OS 起動後に設定するようにしてください。

表 1-2-3-1. Administrator ユーザー

ユーザー名	パスワード	グループ	説明
Administrator	Administrator	Administrators	完全な管理者権限を持つビルトインのユーザーアカウントです。

１－２－４　コンピューター名

Windows Embedded Standard 7 は、他の Windows システムと同様に「コンピューター名」、「ドメイン」、「ワークグループ」の設定が必要となります。ネットワーク上の Windows システムは、これらの設定を用いて各々のシステム識別を行います。

初期状態での「コンピューター名」、「ドメイン」、「ワークグループ」の設定を表 1-2-4-1 に示します。

※　同一ネットワーク上に FP シリーズまたは、他の弊社 Windows 製品を複数台接続する場合は、「コンピューター名」が重複しないように変更してください。

表 1-2-4-1. コンピューター名、ドメイン、ワークグループの初期設定

コンピューター名	WIN-*********** *の部分は本体ごとに異なった文字となります。
ワークグループ	WORKGROUP

１－３　　アプリケーション開発と実行

FP シリーズ用 Windows Embedded Standard 7 32 ビット版では、アプリケーション開発、ドライバ開発に Microsoft Visual Studio など、普段使い慣れた Windows 用の開発環境を使用することができます。ただし、組込みシステムの制限として FP シリーズ本体での開発ができません。開発は Windows オペレーティングシステムが動作している PC で行います。作成したアプリケーションは、FP シリーズ本体にインストールして動作確認を行います。（クロス開発）

● アプリケーションの開発

Windows オペレーティングシステムが動作している PC を使用してアプリケーションの開発を行います。アプリケーションの開発には Microsoft Visual Studio などの一般的な Windows アプリケーション開発環境を使用します。FP シリーズ用 Windows Embedded Standard 7 32 ビット版は互換性を重視して構築されていますので、開発 PC で動作したものをほぼそのまま動作させることができます。このため開発用 PC を使用してデバッグ、動作確認を行うことが可能です。

※　FP シリーズ特有のデバイスを使用している場合は、開発 PC で動作させることができませんので注意してください。

● アプリケーションの実行

FP シリーズ本体で最終動作の確認を行います。開発 PC で動作したものがほぼそのまま動作するように構築されていますが、組込み OS であるため動作が異なる可能性があります。アプリケーションを実際に利用する前に十分な動作検証を行ってください。

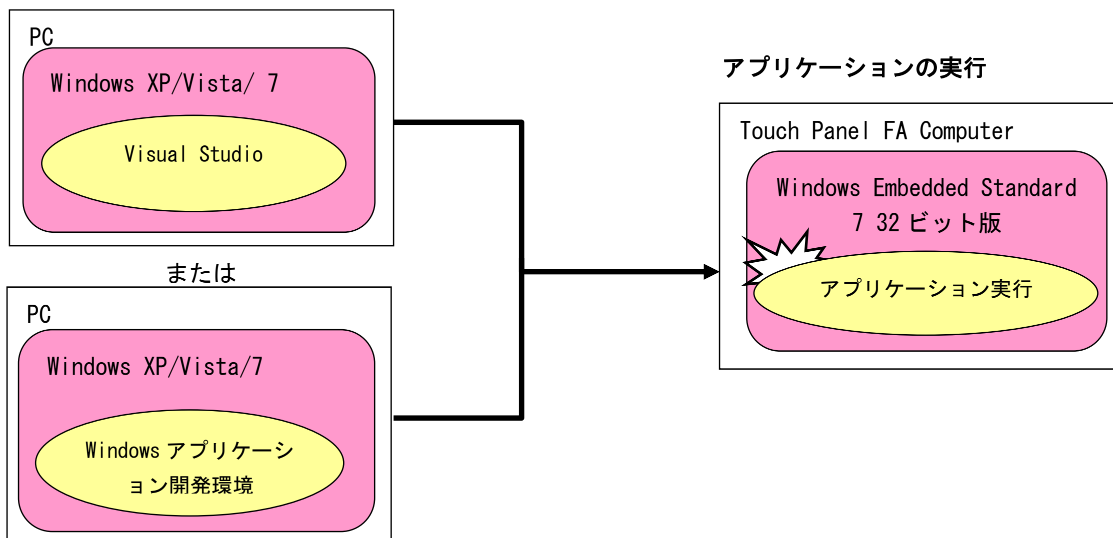

図 1-3-1. アプリケーション開発と実行

第２章　システムの操作

本章では、FP シリーズ用 Windows Embedded Standard 7 32 ビット版の基本的な操作方法について説明します。

２－１　OS の起動と終了

２－１－１　OS の起動

FP シリーズ本体に電源を投入します。Windows ロゴの起動画面が表示され、Windows Embedded Standard 7 が起動します。正常に起動すると、図 2-1-1-1 のようなデスクトップ画面が表示されます。

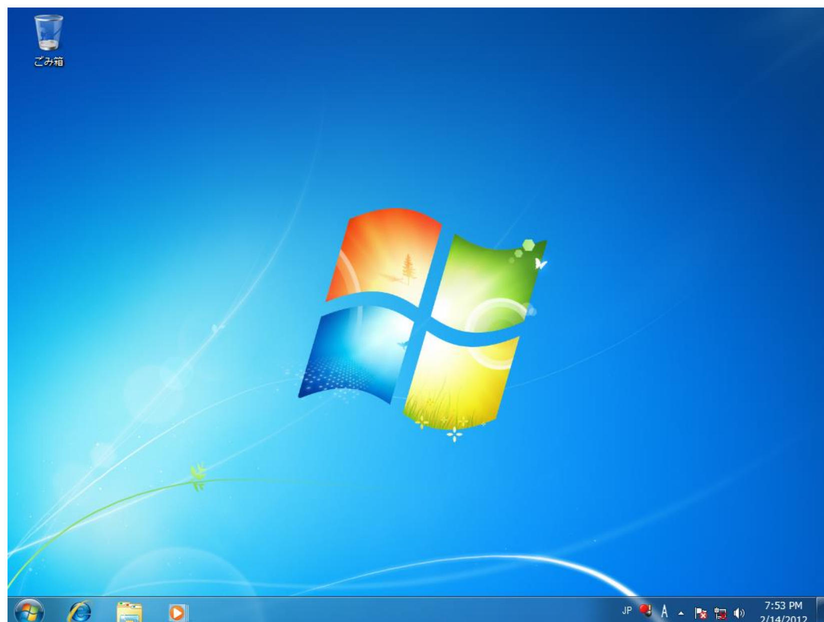

図 2-1-1-1. デスクトップ

２－１－２　OS の終了

スタートメニューから[シャットダウン]を選択します。画面表示、POWER LED が消え、電源が待機状態になることを確認してください。

※　EWF が有効になっている場合は、終了処理を行わずに電源断を行うことができます（強制電源断）。EWF についての詳細は「2-3 EWF 機能」を参照してください。

２－２　　外部 RTC

２－２－１　　RTC とシステム時刻について

FP シリーズでは CPU 内部 RTC とは別に、温度変化による誤差が少ない高精度 RTC を外部に実装しています。外部 RTC を使用してシステム時刻（CPU 内部 RTC）の初期化、更新を行うことができます。

２－２－２　　外部 RTC によるシステム時刻更新機能

「RAS Config Tool」の「Secondary RTC Configuration」を使用して、Auto Update 機能を「Enable System Auto Update」に設定することで、自動的に外部 RTC によるシステム時刻の初期化、更新が行われます。（図 2-2-2-1）

※　「RAS Config Tool」については、「2-13 RAS Config Tool」を参照してください。

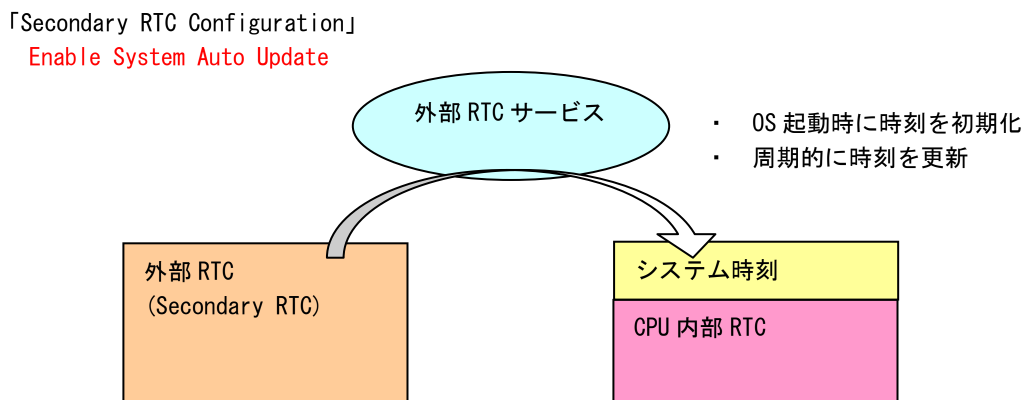

図 2-2-2-1.　System Auto Update 機能有効

２－２－３　　日付と時刻の設定

外部 RTC によるシステム時刻更新機能を使用している場合は、OS に標準で用意されている「日付と時刻」を使用して日時の設定をしても、更新機能によりシステム時刻が変更されてしまいます。システム時刻更新機能を使用している場合は、「RAS Config Tool」の「Secondary RTC Configuration」を使用して日付、時刻を設定してください。

ユーザーアプリケーションで時刻設定を行う場合も同様に、Win32API の SetSystemTime()、SetLocalTime()を使用するとシステム時刻のみが更新されてしまいます。外部 RTC、システム時刻の両方を設定するために G5_SetSystemTime()、G5_SetLocalTime()を用意していますのでこちらを使用するようにしてください。

※　「RAS Config Tool」については、「2-13 RAS Config Tool」を参照してください。
※　G5_SetSystemTime()、G5_SetLocalTime については、「5-10 外部 RTC 機能」を参照してください。

２－３　EWF 機能

２－３－１　EWF とは

EWF(Enhanced Write Filter)とは、Windows Embedded Standard 7 の機能で、書込みアクセスからドライブを保護する機能です。EWF を有効にするとドライブを書込み禁止にした状態で、システムを正常に動作させることが可能となります。

組込みデバイスでは、急なシャットダウン等の電源断に対しても機能が失われないシステム設計を要求されたり、書込み回数に制限のあるフラッシュメディアデバイスへの書込みを抑止する必要があります。EWF は、組み込みデバイスにおけるこのようなニーズに対して提供されている機能です。システム運用中に誤って設定ファイルの変更がされた場合でも、再起動することによって EWF を有効にする直前の状態に戻すこともできます。

EWF では、書込み操作を実際のドライブとは別の記憶領域にリダイレクトすることによりドライブを保護します。ドライブ自体のデータは変更されないため、システム本体、ユーザーデータを保護することが可能となります。リダイレクトされる記憶領域のことをオーバーレイと呼びます。FP シリーズでは、オーバーレイに RAM を使用します。

● EWF の特徴

Enhanced Write Filter の略
ドライブの変更内容を RAM に保存
EWF で保護されたドライブの内容は変更されない
ドライブ保護の有効・無効が変更可能（Enable/Disable）
変更内容を保護されたボリュームに反映することも可能（Commit）

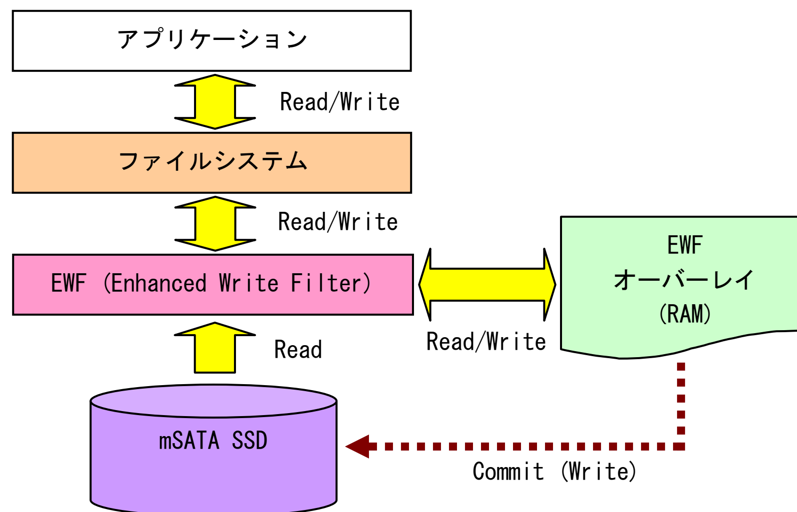

図 2-3-1-1．EWF の仕組み

２－３－２　ドライブと EWF 設定

初期状態の EWF の状態は表 2-3-2-1 のようになっています。EWF の状態を変更する場合は「2-3-3 EWF の設定方法」を参照してください。

※　EWF が有効の場合、設定の変更、データの書き換えができません。変更を行う場合には、EWF を無効にしてください。

表 2-3-2-1. FP シリーズ 初期 EWF 設定

ドライブ	EWF	ドライブ内容
C	無効	システムドライブ オペレーティングシステム本体を格納します。

２－３－３　EWF の設定方法

EWF Manager コマンドを使用して EWF を操作することができます。EWF Manager コマンドはコンソールアプリケーションです。スタートメニューから[すべてのプログラム]→[アクセサリ]→[コマンド プロンプト]を選択しコマンドプロンプトを開き、コマンドを実行します。

● EWF 有効

C ドライブの EWF を有効にする場合は以下のとおりです。次回起動時に ENABLE コマンドが実行され EWF が有効になります。

```
> ewfmgr c: -enable        ← C ドライブを EWF 有効に

*** Enabling overlay

Protected Volume Configuration
Type            RAM
State           DISABLED
Boot Command    ENABLE     ← 次回起動時のコマンドに ENABLE が登録されます
Param1        0
Param2        0
          ¦
```

● EWF 無効

C ドライブの EWF を無効にする場合は以下のとおりです。次回起動時に DISABLE コマンドが実行され EWF が無効になります。

```
> ewfmgr c: -disable                ← C ドライブを EWF 無効に

*** Disabling overlay

Protected Volume Configuration
Type            RAM
State           ENABLED
Boot Command    DISABLE    ← 次回起動時のコマンドに DISABLE が登録されます
Param1        0
Param2        0
         ⋮
```

● コミット

EWF が有効の場合、ドライブへの書込みはオーバーレイにリダイレクトされます。そのまま終了させるとドライブへの書込みデータは消えてしまいます。コミットを行うとオーバーレイのデータをドライブに書込むことができます。C ドライブをコミットする場合は以下のとおりです。終了時にオーバーレイのデータがドライブに書込まれます。

```
> ewfmgr c: -commit                 ← C ドライブの変更分をコミット

*** Committing overlay to the protected volume.

Protected Volume Configuration
Type            RAM
State           ENABLED
Boot Command    COMMIT     ← 終了時に COMMIT が実行されます
Param1        0
Param2        0
         ⋮
```

※　変更分がドライブに書込まれるため、終了処理に時間がかかることがあります。終了処理中に電源を落とさないようにしてください。

● EWF の状態確認

C ドライブの EWF の状態を確認する場合は以下のとおりです。

```
> ewfmgr c:                ←――――― C ドライブの EWF 状態を確認

Protected Volume Configuration
Type            RAM
State           DISABLED
Boot Command    NO_CMD
Param1        0
Param2        0
        ⋮
```

● その他のコマンド

EWFMGR には、説明したコマンドの他にもコマンドが存在します。表 2-3-3-1 に FP シリーズで使用できるコマンド一覧を示します。

表 2-3-3-1. EWFMGR コマンド一覧

コマンド	内容
コマンドなし	ドライブを指定しない場合は、EWF ボリュームの表示と保護ドライブの一覧を表示します。 ドライブを指定する場合は、そのドライブの EWF 保護状態とオーバーレイの状態を確認することができます。 --- 例 1 --- > ewfmgr --- 例 2 --- > ewfmgr C:
-all	EWF の対象になっているすべてのドライブの EWF 保護状態を表示します。 --- 例 --- > ewfmgr -all
-enable	指定したドライブを EWF 有効にします。コマンド実行後、次回起動時に EWF が有効になります。 --- 例 --- > ewfmgr C: -enable
-disable	指定したドライブを EWF 無効にします。コマンド実行後、次回起動時に EWF が無効になります。 --- 例 --- > ewfmgr C: -disable
-commit	指定したドライブをコミットします。コマンド実行後の終了時にコミットされます。 --- 例 --- > ewfmgr C: -commit
-commitanddisable	指定したドライブをコミットし、EWF 無効にします。コマンド実行後の終了時にコミットが行われます。次回起動時に EWF は無効となります。

	-live を指定するとその場でコミットと EWF 無効処理が行われます。この場合、コマンド実行後すぐに EWF が無効となります。 --- 例 --- > ewfmgr C: -commitanddisable --- 例 --- > ewfmgr C: -commitanddisable -live

● 設定ツールでの EWF 操作

スタートメニューから[すべてのプログラム]→[EWF]→[EWF_Config]に EWF 操作を行う設定ツールが登録されています。この設定ツールを使用しても基本的な操作(有効/無効)が可能です。詳細は「2-11 EWF Config Tool」を参照してください。

● アプリケーションからの EWF 操作

EWF API を使用することによってアプリケーションから EWF の操作が可能です。詳細は「第 4 章 EWF API」を参照してください。

２－３－４　EWF を使用するにあたっての注意事項

① EWF によるシステムメモリの消費

EWF はオーバーレイにシステムメモリを使用します。OS と EWF オーバーレイでシステムメモリを共有する構成となるため、EWF オーバーレイで消費された分だけ、OS が利用できるメモリは少なくなります。

OS が必要とするメモリと EWF オーバーレイで消費するメモリの合計が搭載メモリのサイズを超えた場合のシステムの動作は保証されません。

② EWF の消費メモリの解放

EWF で保護されたドライブに新たにファイルを作成、またはコピーした場合、EWF オーバーレイによってシステムメモリが消費されます。このとき消費されたメモリは、作成したファイルを削除しても解放されません。EWF オーバーレイは RAM Disk や Disk Cache と違い、システムを再起動するまで一度消費したメモリを解放しません。

EWF を有効にした状態で長時間システムを使用する場合は、OS で使用するメモリと EWF オーバーレイで使用するメモリの合計が搭載メモリを超える前に再起動させる必要があります。

EWF オーバーレイのメモリ使用量は、「EWFMGR コマンド」で確認することができます。また、アプリケーションからは、「EWF API」を使用して確認することができます。

③ OS によるファイル作成

OS はレジストリ情報や、イベントログ、テンポラリファイルなどユーザが普段意識しないところでファイル作成、ファイル更新を行っています。システムドライブの EWF 保護を有効にする場合、これらのファイル作成、ファイル更新は EWF オーバーレイのメモリ消費を増加させてしまいます。設定を変更することで、ファイルの出力先を EWF 保護が無効なドライブへ変更することができます。表 2-3-4-1 に OS が作成するファイルと出力先の変更方法を示します。

表 2-3-4-1. OS が作成するファイルと出力先の変更方法

項目	内容
イベントログ	イベントログの場所を変更します。 Windows の管理メニューから出力先を変更することができます。 変更方法を後述します。
インターネット一時ファイル	インターネット一時ファイルフォルダの場所を変更します。 インターネット一時ファイルはデフォルトでは「%USERPROFILE%¥Local Settings¥Temporary Internet Files」に設定されています。 レジストリ値を変更することによって、出力先を変更することができます。 表 2-3-4-2 に設定レジストリを示します。
TEMP、TMP フォルダ	TEMP、TMP フォルダの場所を変更します。 レジストリ値を変更することによって、出力先を変更することができます。 表 2-3-4-3 に設定レジストリを示します。

● イベントログ出力先設定手順

① スタートメニューから[コンピュータ]を右クリックし「管理」を選択します。

② [コンピュータの管理]が開きます。[イベントビューアー ― Windows ログ]内の項目を右クリックし、「プロパティ」を選択します。

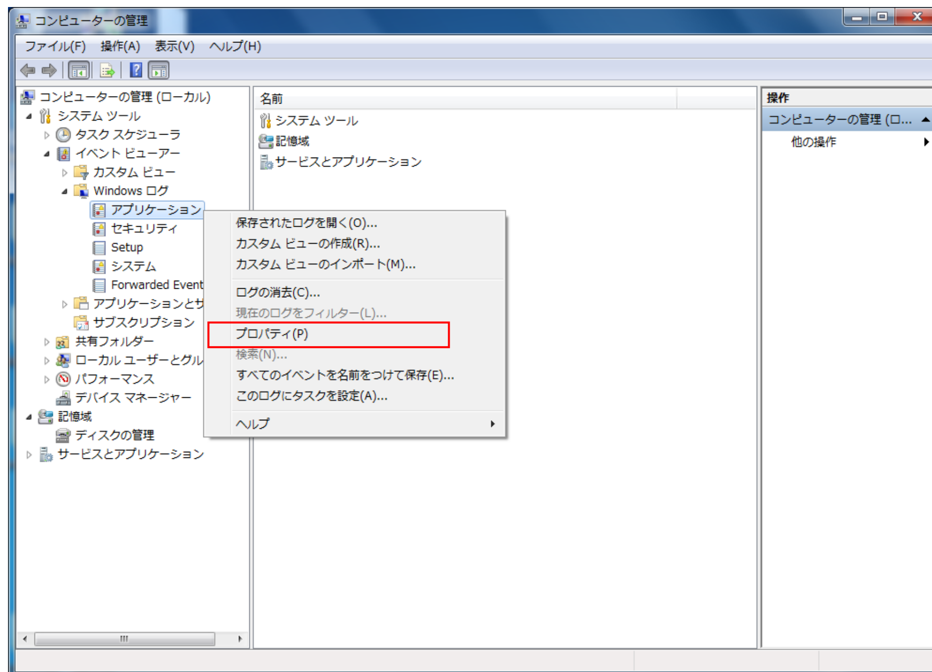

図 2-3-4-1. Windows ログのプロパティ選択

③ [ログのパス]を任意のパスに変更します。

④ [OK]ボタンをクリックします。

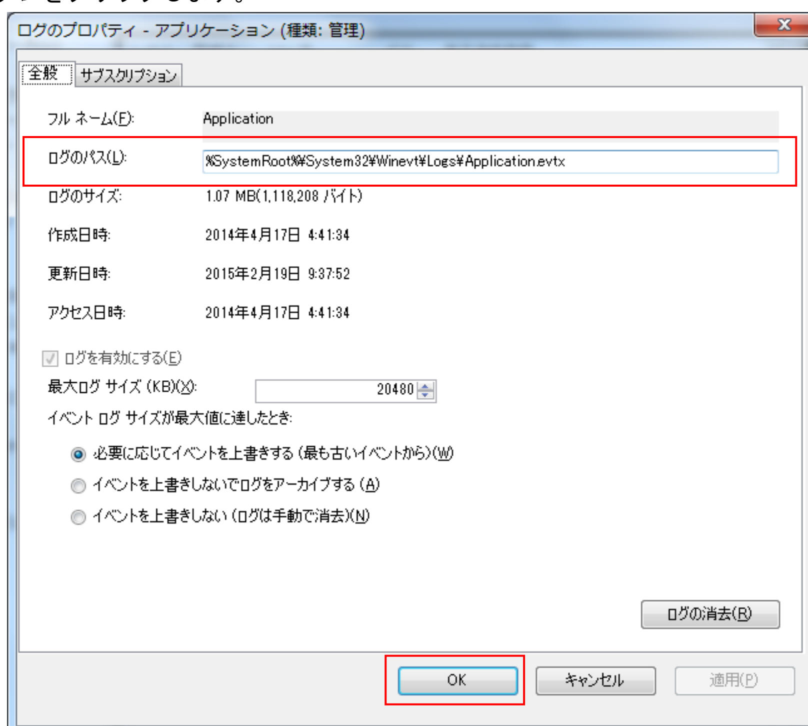

図 2-3-4-2. イベントログ出力先の変更

表 2-3-4-2. インターネット一時ファイルの出力先変更

キー	HKEY_CURRENT_USER¥Software¥Microsoft¥Windows¥CurrentVersion¥Explorer¥User Shell Folders		
	名前	種類	データ
	Cache	REG_EXPAND_SZ	＜ドライブ名とパス＞
キー	HKEY_CURRENT_USER¥Software¥Microsoft¥Windows¥CurrentVersion¥Explorer¥Shell Folders		
	名前	種類	データ
	Cache	REG_EXPAND_SZ	＜ドライブ名とパス＞

表 2-3-4-3. TEMP、TMP フォルダの変更

キー	HKEY_CURRENT_USER¥Environment		
	名前	種類	データ
	TEMP	REG_SZ	＜ドライブ名とパス＞
キー	HKEY_CURRENT_USER¥Environment		
	名前	種類	データ
	TMP	REG_SZ	＜ドライブ名とパス＞

④ アプリケーションでの注意

EWF が有効の場合、利用可能なメモリが減少します。アプリケーションでのメモリ、リソース確保時には注意が必要となります。また、中間ファイルを出力する可能性がある言語を使用する場合は、中間ファイルの出力先なども考慮する必要があります。表 2-3-4-4 にアプリケーションでの注意事項を示します。

表 2-3-4-2. アプリケーションでの注意事項

項目	内容
C++	malloc など、ヒープを確保する場合には、戻り値の確認を必ず行ってください。リソースについても同様にエラーチェックを行ってください。ダイアログの作成･フォームの作成などについてもハンドルのチェックを行うようにしてください。
.NET Framework	CLR アセンブラは、ファイルとして実行時に作成されます。これらも EWF オーバーレイとして RAM を消費することになります。EWF を有効にする前に、使用する.NET Framework アプリケーションを一度実行する方が望ましいです。
ASP.NET	IE での履歴、テンポラリファイル出力で EWF オーバーレイとして RAM を消費します。 テンポラリファイルの出力先を変更する場合は、「③ OS によるファイル作成」を参考に変更してください。

２－４　ログオン設定

２－４－１　自動ログオン設定

ログオンは、初期状態では Administrator アカウントで自動ログオンするように設定されています。ログオンアカウントを選択してログオンしたい場合は、設定を変更する必要があります。

● 設定手順

① スタートメニューから[すべてのプログラム]→[アクセサリ]→[コマンド プロンプト]を選択しコマンドプロンプトを開きます。

② 以下のコマンドを実行します。

```
> control userpasswords2
```

③ [ユーザーアカウント]ダイアログが開きます。[ユーザーがこのコンピューターを使うには、ユーザー名とパスワードの入力が必要]チェックボックスにチェックを入れ、[OK]ボタンを押します。

２－５　　言語設定

２－５－１　　マルチ言語機能

Windows Embedded Standard 7 は、マルチ言語対応 OS となっています。FP シリーズでは、以下の言語に対応しています。初期状態では日本語環境で起動します。

対応言語一覧

イタリア語	スペイン語	ドイツ語	フランス語
ポルトガル語	英語	韓国語	中国語(簡体)
中国語(繁体)	中国語(香港)	日本語	

２－５－２　　言語の変更方法

言語はコントロールパネルから変更可能となっています。英語環境にするには、以下の手順で設定を行ってください。

● 設定手順

① スタートメニューから[コントロール パネル]を選択します。
② [コントロール パネル]が開きます。[地域と言語]を実行します。[地域と言語]ダイアログが開きます。
③ [形式]タブを選択します。[形式]のリストから英語を選択します。
④ [場所]タブを選択します。[現在の場所]のリストから英語地域を選択します。
⑤ [キーボードと言語]タブを選択します。[表示言語を選んでください]のリストから、[English]を選択し、[適用]ボタンを押します。[表示言語の変更]ダイアログが表示されますので、[今すぐログオフ]を選択します。
⑥ 再ログイン後、コントロールパネルスタートメニューからコントロールパネルを選択し[Region and Language]を選択します。[Administrative]タブ（[管理]タブ）の[Copy Settings...]ボタン（[設定のコピー]ボタン）を押して表示されるダイアログで、[Welcome screen and system accounts]（[ようこそ画面とシステムアカウント]）と[New user accounts]（[新しいユーザーアカウント]）にチェックを入れ、[OK]ボタンを押します。再起動確認のダイアログが表示されますので、[Restart now]ボタンを押して再起動します。
⑦ 再起動後、コントロールパネルスタートメニューからコントロールパネルを選択し[Region and Language]を選択します。[Administrative]タブ（[管理]タブ）の[Change system locale]ボタン（[システムロケールの変更]ボタン）をクリックします。ダイアログが表示されますので、リストから英語を選択し、[OK]ボタンを押します。再起動確認のダイアログが表示されますので、[Restart now]ボタンを押して再起動します。

２－６　　モニタ設定

２－６－１　　マルチモニタ機能

FP シリーズでは、「マルチモニタ」機能を利用することで複数のディスプレイに表示ができます。

２－６－２　　モニタ設定の変更方法

モニタ設定はコントロールパネルの[ディスプレイ]からと[Intel® Graphics and Media]からの 2 種類の設定方法があります。

以下に標準的な設定方法を示します。

● 設定手順

[ディスプレイの場合]

⑤　スタートメニューから[コントロール パネル]を選択します。

⑥　[コントロール パネル]が開きます。[ディスプレイ]を実行します。[ディスプレイ]ダイアログが開きます。

⑦　[解像度の調整]を選択します。[画面の解像度]ダイアログが開きます。

※　デスクトップの何もない部分を右クリックし、[画面の解像度]を実行しても同じダイアログが開きます。

⑧　モニタなどを増設すると検出されたモニタが表示されます。メインにしたいモニタをクリックします。(青枠になります)

⑨　[複数のディスプレイ]の[表示画面を拡張する]を選択します。

1.　表示画面を複製する

メイン画面と外付け液晶ディスプレイの画面に同じ内容を表示（クローン表示）します。

2.　表示画面を拡張する

メイン画面と外付け液晶ディスプレイの画面を 1 つのデスクトップとして表示(デュアルディスプレイ表示)します。

3.　デスクトップを 1 のみに表示する

メイン画面にのみ表示します。

4.　デスクトップを 2 のみに表示する

外付け液晶ディスプレイの画面にのみ表示します。

⑩　[適用]ボタンをクリックします。

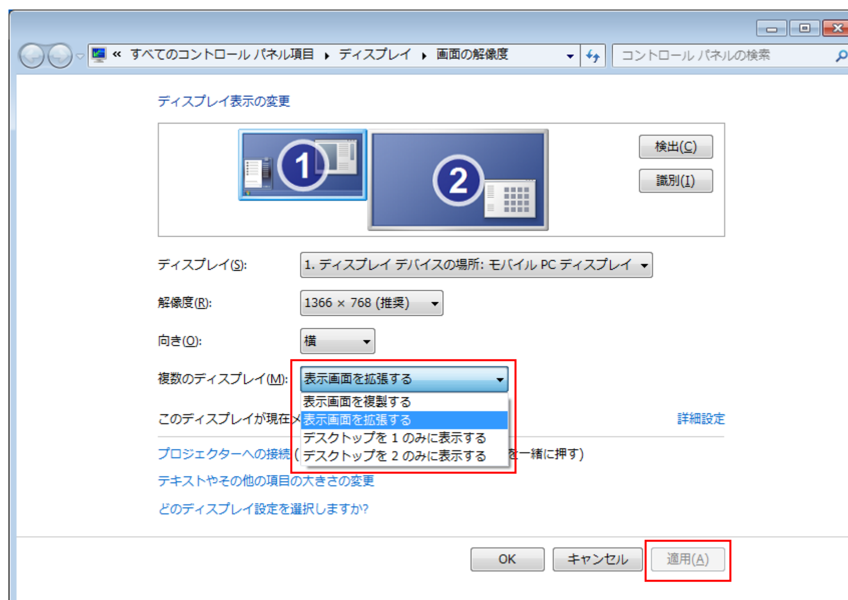

図 2-6-2-1.　画面の解像度設定画面

⑪　図 2-6-2-2 のようなメッセージが表示されるので、15 秒以内に[変更を維持する]ボタンをクリックします。15 秒以内に[変更を維持する]ボタンをクリックしなかった場合は、元の設定状態に戻ります。

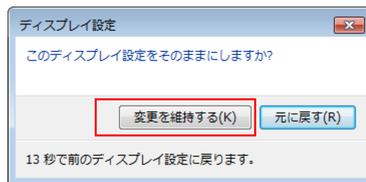

図 2-6-2-2. 画面の解像度設定画面

⑫　増設したモニタをメイン ディスプレイとして利用する場合は、[これをメイン ディスプレイにする]にチェックを入れて[OK]ボタンをクリックします。

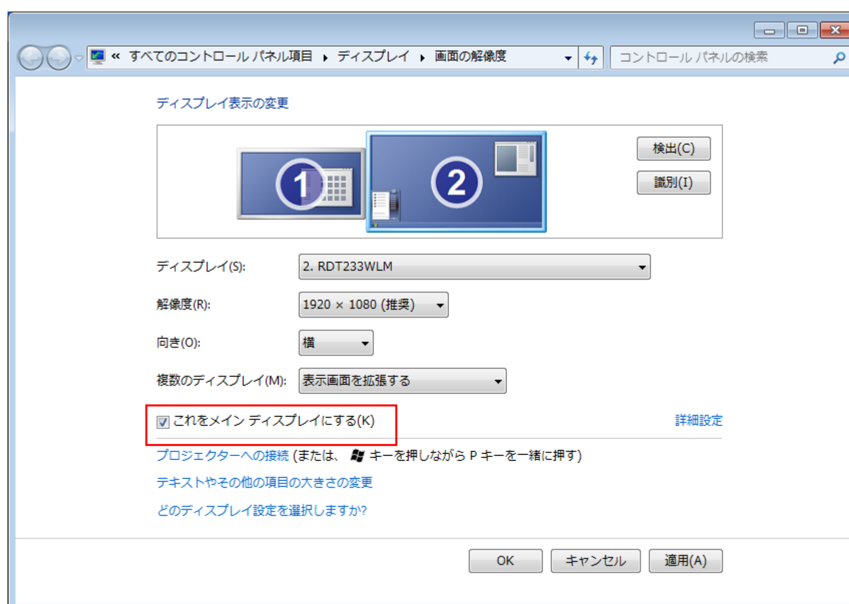

図 2-6-2-3. メインディスプレイ設定

デスクトップ上で「Windows」キーと「P」キーを同時に押すと、プロジェクタや外付けディスプレイの表示方法を決める設定メニューが呼び出せます。ここから、「拡張」を選べばデュアルディスプレイ表示、「複製」を選べばクローン表示と、用途に応じてすばやく切り替えられます。

図 2-6-2-4. 簡易マルチモニタ設定画面

[Intel(R) HD Graphics Control Panel の場合]

① スタートメニューから[コントロール パネル]を選択します。

② [コントロール パネル]が開きます。[Intel(R) HD Graphics]を実行します。[Intel(R) HD Graphics Control Panel]が開きます。

※ デスクトップの何もない部分を右クリックし、[Graphics Properties...]を実行しても同じダイアログが開きます。

③ 「Display」ボタンを押下します。[Displays Settings]画面が開きます。

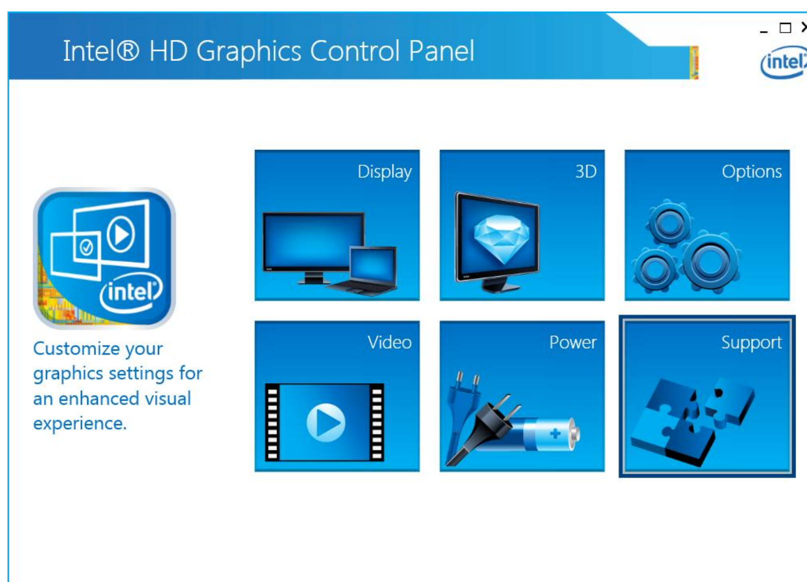

図 2-6-2-5. Intel(R) HD Graphics Control Panel

④ メニューから[Multiple Displays]を選択します。[Multiple Displays]画面が開きます。

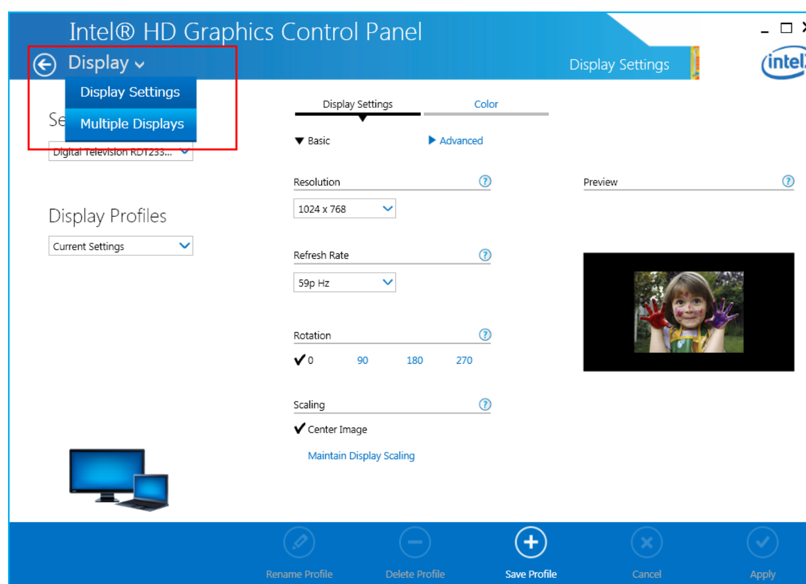

図 2-6-2-6. Display Settings 画面

⑤　[Select Display Mode]で表示モードを指定し、[Select Active Displays]でディスプレイの優先順位を設定します。表示モードが[Extended]の場合は、[Arrenge Displays]でディスプレイ配置を設定します。
※　コントロールパネルの[画面と解像度]で設定するメインディスプレイと番号が異なります。タッチパネルのデバイス設定モニタを設定する場合は注意してください。

1. Single
1 画面にのみ表示します。
2. Clone
メイン画面と外付け液晶ディスプレイの画面に同じ内容を表示（クローン表示）します。
3. Extended
メイン画面と外付け液晶ディスプレイの画面を 1 つのデスクトップとして表示（デュアルディスプレイ表示）します。

⑥　[Apply]ボタンをクリックします。

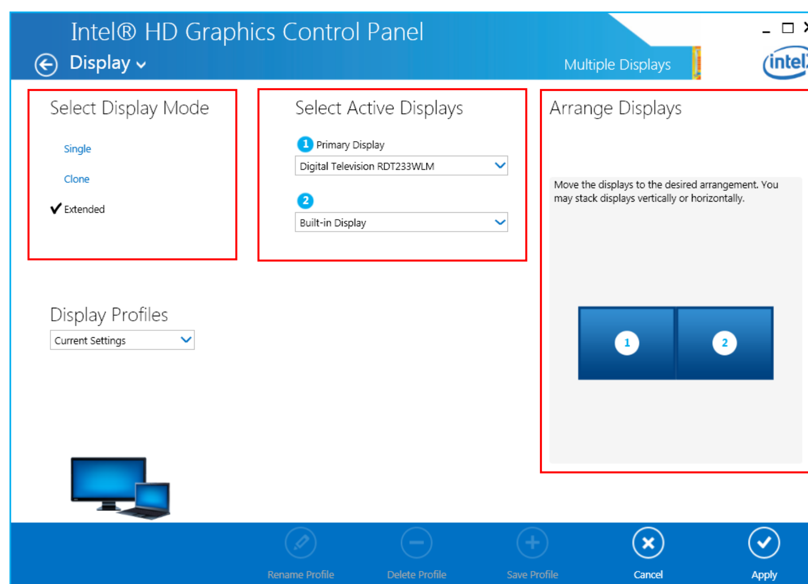

図 2-6-2-7. Multiple Displays 画面

⑦　図 2-6-2-8 のようなメッセージが表示されるので、15 秒以内に[Yes]ボタンをクリックします。15 秒以内に[Yes]ボタンをクリックしなかった場合は、元の設定状態に戻ります。

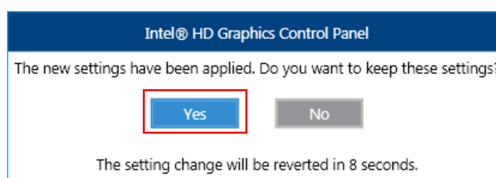

図 2-6-2-8. 設定確認画面

デスクトップの何もない部分を右クリックし、[Graphics Options]を押すと、出力先を決める設定メニューが呼び出せます。ここから、[Extended Desktop]を選べばデュアルディスプレイ表示、｢Clone Displays｣を選べばクローン表示と、用途に応じてすばやく切り替えられます。

※　メインディスプレイを外部モニタに設定した状態で外部モニタを取り外した場合、タスクバーが表示されなくなる場合があります。その場合は[Graphics Options]から[Clone Displays]等を選択し、表示させてください。

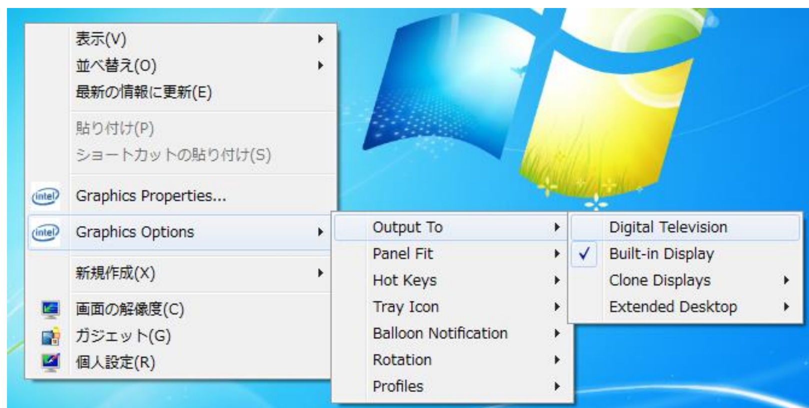

図 2-6-2-9. 簡易出力先設定画面

２－７　　タッチパネル設定

２－７－１　　モニタ構成の設定

タッチパネルで操作するモニタを設定します。

● 設定手順

① スタートメニューからすべてのプログラムを選択します。

② [DMC]->[DMT-DD]を実行し、[タッチパネル設定ツール]を開きます。

③ [デバイスの追加]を選択し、タッチパネルデバイス一覧を表示させます。

④ モニタ設定を行いたいタッチパネルデバイスを選択し、[モニタ構成]ボタンを押下します。

図 2-7-1-1. モニタ構成の設定

⑤ タッチパネルを使用するモニタに[モニタ選択画面]が表示されたら、タッチパネルを 2 回タッチします。目的のモニタでない場合は、Enter]を入力して次のモニタへ進めます。最後のモニタで[Enter]を入力すると設定完了です。

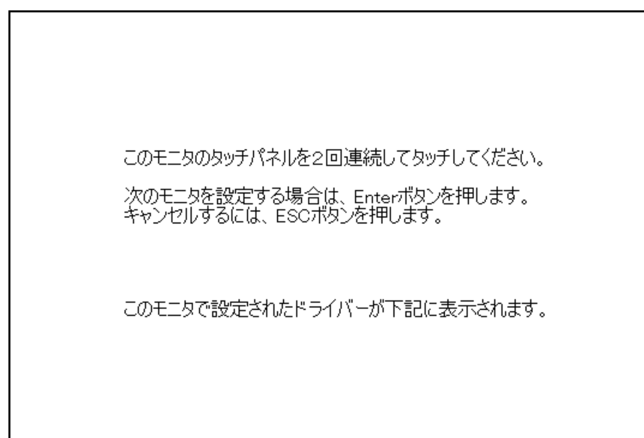

図 2-7-1-2. モニタ設定画面

２－７－２　　タッチパネルの調整

タッチ位置の調整を行います。調整を行うにはタッチ操作を行うモニタを設定する必要があります。調整を行う前に「2-7-1 モニタ構成の設定」を行ってください。

● 設定手順

① スタートメニューからすべてのプログラムを選択します。

② [DMC]->[DMT-DD]を実行し、[タッチパネル設定ツール]を開きます。

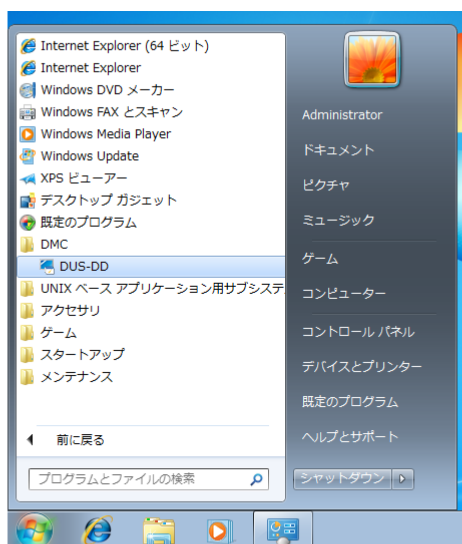

図 2-7-2-1. DMC タッチパネル設定ツールの起動

③ [基本設定]を選択し、基本設定画面を表示させせます。

④ キャリブレーションを行いたいタッチデバイスを選択し、[ソフトウェア設定]タブの[9 点補正]ボタンを押下します。

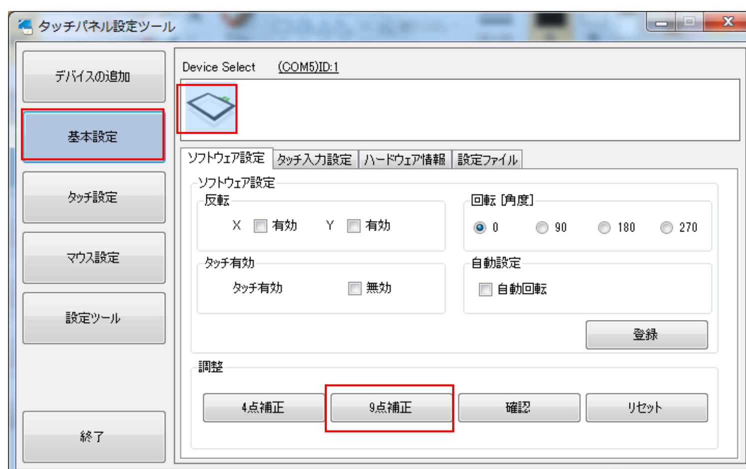

図 2-7-2-2. 基本設定画面

⑤ ［調整画面］が表示されます。表示されている赤い十字を順番にタッチして調整を行います。

図 2-7-2-3. 調整画面

※　キャリブレーションを行う場合は、Windows 標準の調整機能は使用しないでください。
設定を行った場合は事前にリセットが必要です。

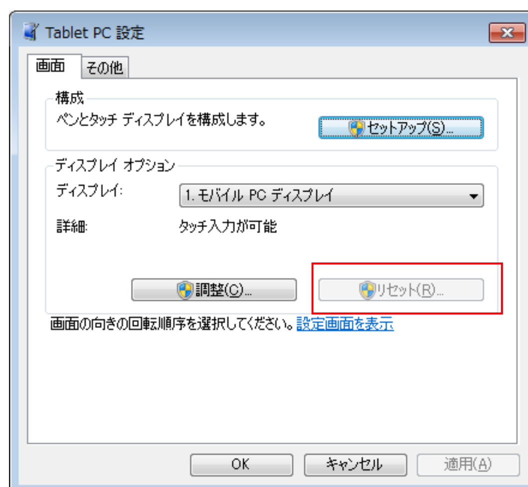

図 2-7-2-4. Windows 標準調整機能のリセット

２－７－３　タッチ音について

タッチパネルをタッチした際にタッチ音が出力されます。
FPシリーズではタッチ音の設定を行うことはできません。

２－８　　タッチパネルインターロック機能（AT シリーズの接続）

シリアル接続のタッチパネルを接続する場合、タッチパネルインターロック機能を使用することができます。タッチパネルインターロック機能の有効・無効は、ASD Config Tool で設定することができます。
弊社「タッチパネルモニタ AT シリーズ シリアル接続タイプ」を使用した例を説明します。

※　タッチパネルインターロック機能を使用する場合、シリアル接続タイプの AT シリーズを使用してください。USB 接続タイプではタッチパネルインターロック機能は使用できません。

２－８－１　AT シリーズの接続

タッチパネルインターロック機能を使用する場合は、AT シリーズのタッチパネル出力を FP シリーズ本体の COM1 へ接続してください。

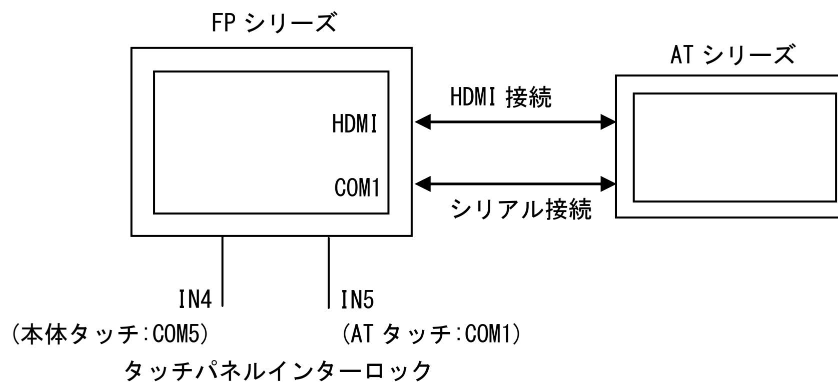

図 2-8-1-1. AT シリーズの接続

２－８－２　タッチパネルデバイスの追加

タッチパネル設定ツールにタッチパネルデバイスを追加し、タッチパネルを使用できるようにします。タッチパネルの調整方法は「2-7 タッチパネル設定」を参照してください。

● 設定手順

① スタートメニューからすべてのプログラムを選択します。

② [DMC]->[DMT-DD]を実行し、[タッチパネル設定ツール]を開きます。

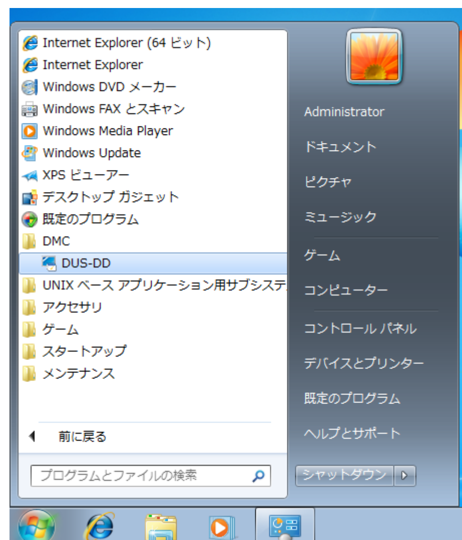

図 2-8-2-1. DMC タッチパネル設定ツールの起動

③　[デバイスの追加]を選択し、タッチパネルデバイス一覧を表示させます。
④　タッチパネルに使用するシリアルポートを選択し、[インストール]ボタンを押下します。
⑤　指示に従いドライバをインストールします。
⑥　インストールが完了したら再起動します。再起動後、設定が有効となります。

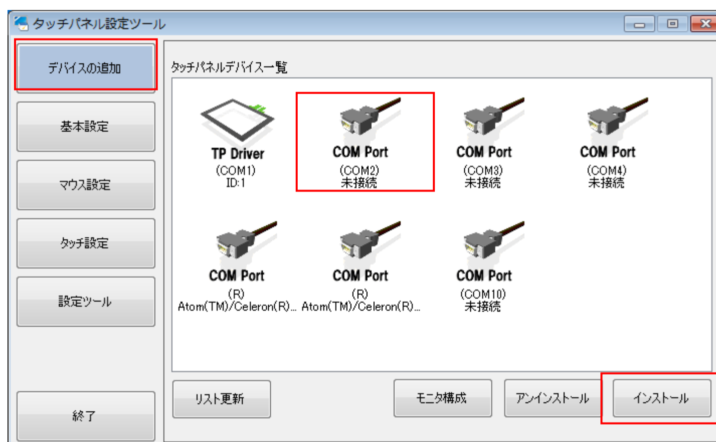

図 2-8-2-2. デバイスの追加

２－８－３　タッチパネルインターロック

タッチパネルインターロック機能を有効にすると、汎用入力 IN4 と IN5 を使用してタッチパネル操作を制限することができます。

タッチパネルインターロック機能の有効・無効の設定は、「ASD Condfig Tool」で行えます。詳細は「2-10-3 Touch Panel Setting」を参照してください。

● [IN4→ON、IN5→ON] の場合

FP シリーズ本体のタッチパネル操作が可能となります。

AT シリーズのタッチパネル操作が可能となります。

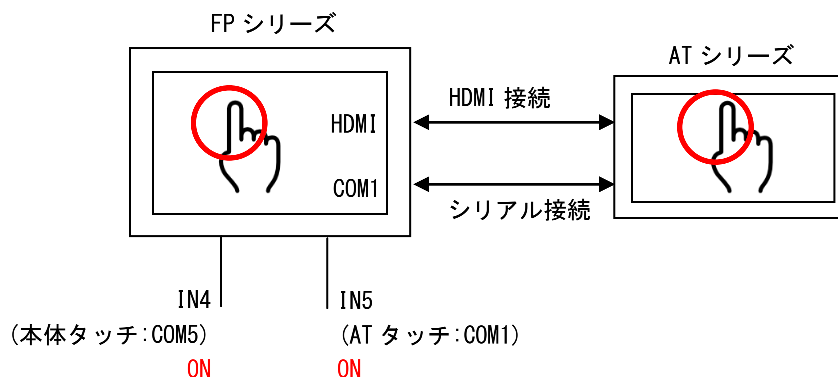

図 2-8-3-1. タッチパネルインターロック IN4→ON、IN5→ON

● [IN4→OFF、IN5→ON] の場合
FP シリーズ本体のタッチパネル操作が不可となります。
AT シリーズのタッチパネル操作が可能となります。

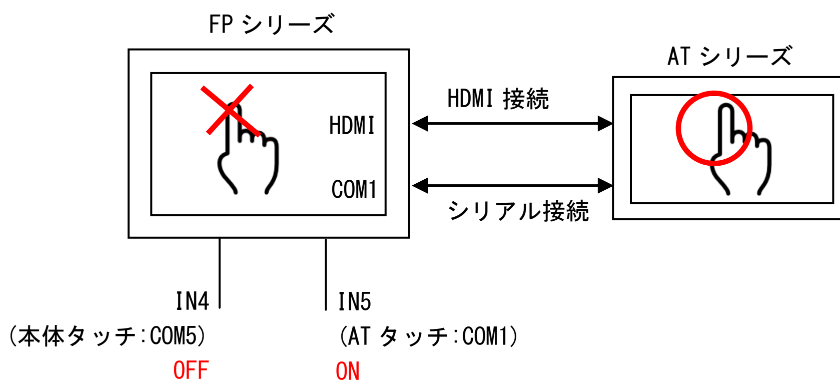

図 2-8-3-2. タッチパネルインターロック IN4→OFF、IN5→ON

● [IN4→ON、IN5→OFF] の場合
FP シリーズ本体のタッチパネル操作が可能となります。
AT シリーズのタッチパネル操作が不可となります。

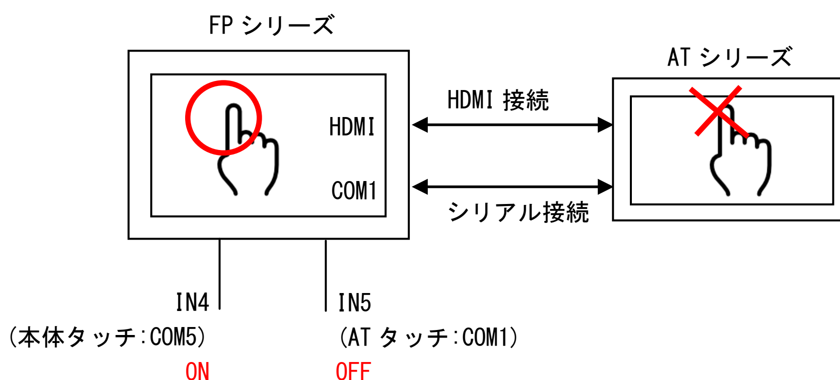

図 2-8-3-3. タッチパネルインターロック IN4→OFF、IN5→ON

● [IN4→ON、IN5→OFF] の場合
FP シリーズ本体のタッチパネル操作が不可となります。
AT シリーズのタッチパネル操作が不可となります。

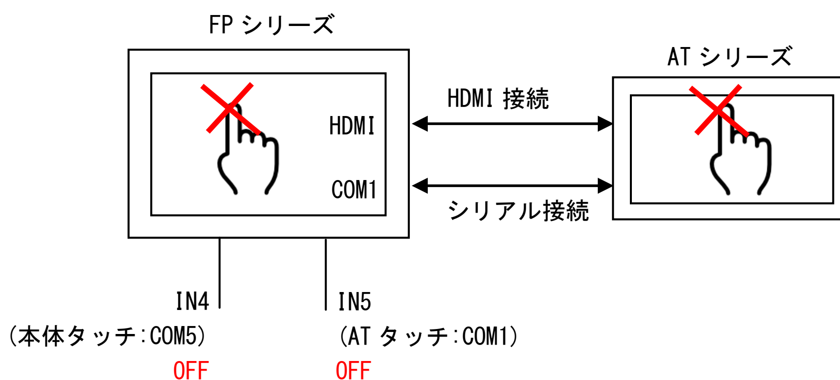

図 2-8-3-4. タッチパネルインターロック IN4→OFF、IN5→ON

２－９　　サービス設定

２－９－１　　サービス設定の変更

　FP シリーズ用 Windows Embedded Standard 7 32 ビット版には、様々なサービスが搭載されています。搭載されているサービスの中には、工場出荷状態では停止しているものもあります。これらのサービスを利用するには、サービスの操作、起動設定の変更などを行う必要があります。
　サービスの操作、起動設定の変更は以下の手順で行います。

● サービス操作、起動設定の変更
① スタートメニューから[コントロール パネル]を選択します。
② [コントロール パネル]から[管理ツール]、[サービス]の順にウィンドウを開きます。
③ [サービス]設定画面が開きます。(図 2-9-1-1)

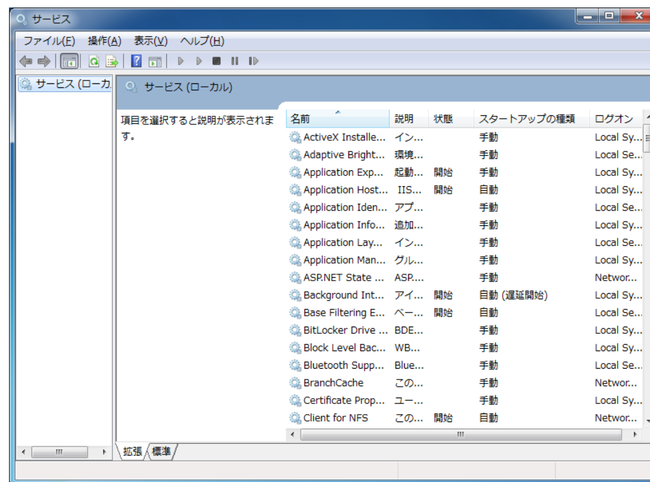

図 2-9-1-1. サービス設定画面

④ 設定を変更するサービスをダブルクリックしてプロパティを開きます。(図 2-9-1-2)

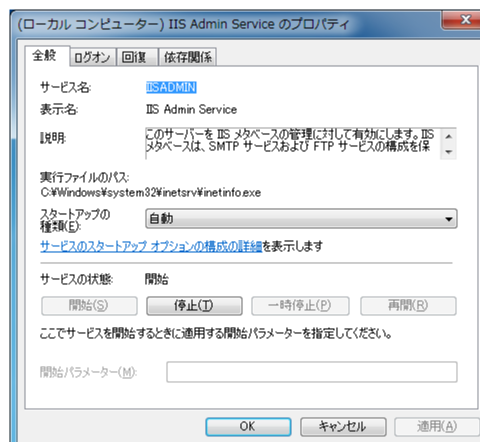

図 2-9-1-2. サービス設定画面

⑤ 起動設定を変更する場合は、[スタートアップの種類]を目的の設定に変更します。
⑥ サービスの操作を行う場合は、[開始]、[停止]、[一時停止]、[再開]ボタンで行います。

２－１０　ASD Config Tool

２－１０－１ ASD Config Tool

「ASD Config Tool」は、FP シリーズ専用のコントロールパネルアプリケーションです。スタートメニューから[コントロール パネル]を開き、[ASD Config]から起動できます。設定内容を表 2-10-1-1 に示します。

表 2-10-1-1. ASD Config Tool 設定/表示内容

タブ	設定/表示内容
Serial Port Setting	シリアルポートの RS-232C/422/485 の切替えを行うことができます。
Touch Panel Setting	タッチパネルの設定を行うことができます。
Board Information	ハードウェア、ソフトウェアのバージョンを確認することができます。

２－１０－２ Serial Port Setting

シリアルポートの RS-232C/422/485 の切替えを行うことができます。

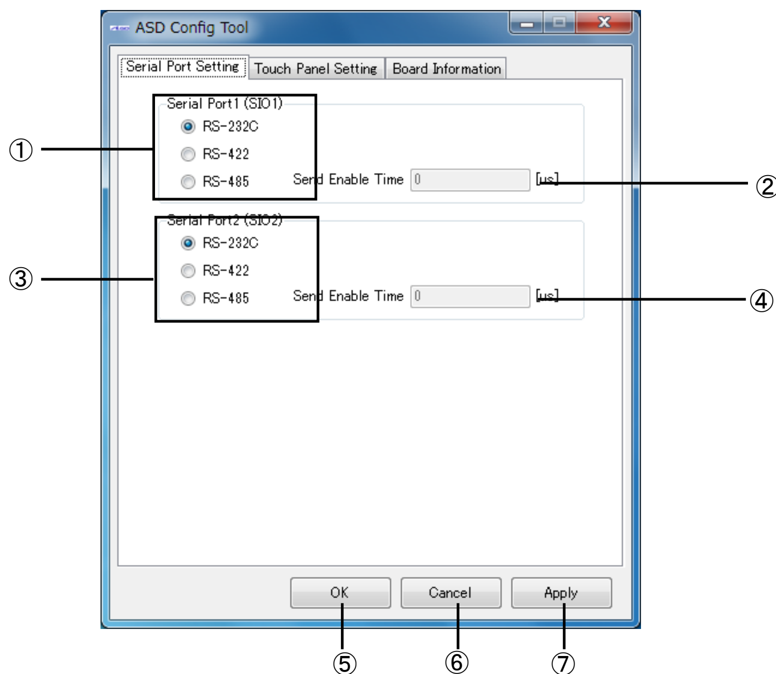

図 2-10-2-1. ASD Config – Serial Port Setting

① SIO1(COM1)の RS-232C/422/485 を切替えます。
② SIO1(COM1)を RS-485 とした場合の送信イネーブルタイムアウト時間[μs]を設定します。(0～65535)
③ SIO2(COM2)の RS-232C/422/485 を切替えます。
④ SIO2(COM2)を RS-485 とした場合の送信イネーブルタイムアウト時間[μs]を設定します。(0～65535)
⑤ 設定を保存、適用して終了します。
⑥ 設定を破棄して終了します。
⑦ 設定を保存、適用します。

２－１０－３ Touch Panel Setting

タッチパネルのインターロック設定、ブザー設定を行うことができます。

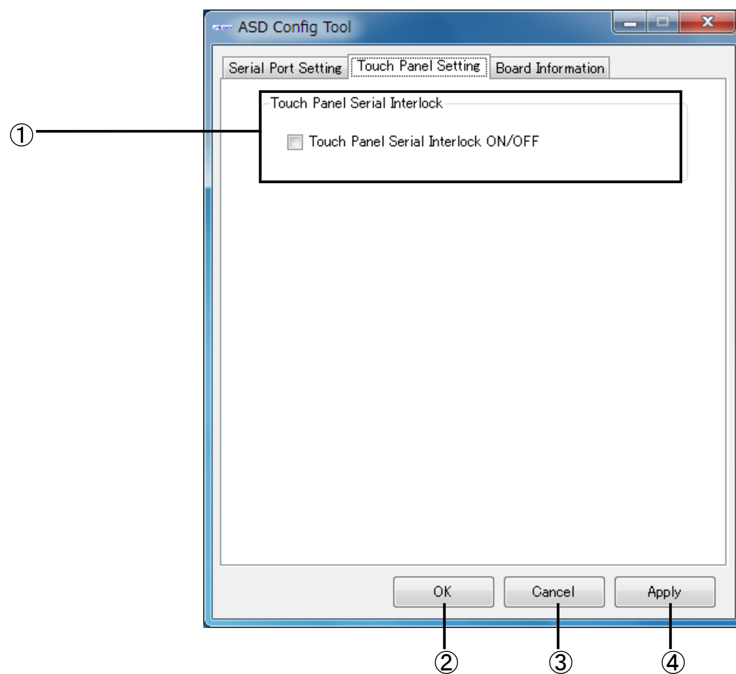

図 2-10-3-1. ASD Config – Touch Panel Setting

① タッチパネルインターロックの ON/OFF を設定できます。(チェックあり: ON、チェックなし: OFF)
② 設定を保存、適用して終了します。
③ 設定を破棄して終了します。
④ 設定を保存、適用します。

２－１０－４ Board Information

ハードウェア、ソフトウェアのバージョンを確認することができます。

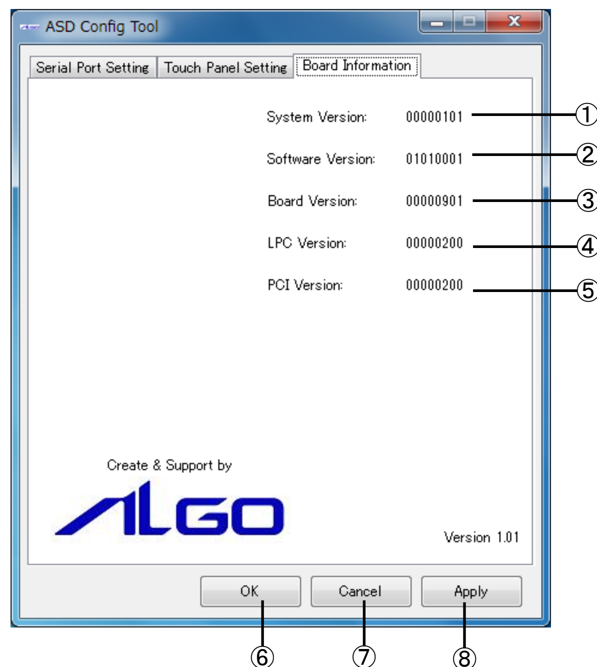

図 2-10-4-1. ASD Config – Board Information

① OS イメージのバージョンを表示します。
② OS イメージのタイプを表示します。
③ メインボードのバージョンを表示します。
④ LPC レジスタのバージョンを表示します。
⑤ PCI レジスタのバージョンを表示します。
⑥ 設定を保存、適用して終了します。
⑦ 設定を破棄して終了します。
⑧ 設定を保存、適用します。

２－１０－５ 初期値

「ASD Config Tool」の設定初期値を表 2-10-5-1 に示します。

表 2-10-5-1. ASD Config Tool 設定初期値

タブ	設定項目	初期値
Serial Port Setting	Serial Port1 (SIO1) Port Type	RS-232C
	Serial Port1 (SIO1) Send Enabled Time	0
	Serial Port2 (SIO2) Port Type	RS-232C
	Serial Port2 (SIO2) Send Enabled Time	0
Touch Panel Setting	Touch Panel Serial Interlock	OFF

２－１１　EWF Config Tool

２－１１－１ EWF Config Tool

「EWF Config Tool」は、FP シリーズ専用の EWF 設定ツールです。スタートメニューから[すべてのプログラム]→[EWF]→[EWF_Config]で起動できます。設定/表示内容を表 2-11-1-1、表 2-11-1-2 に示します。

表 2-11-1-1. EWF Config Tool 表示内容

項目	表示内容
Device Name	保護されたボリュームのデバイス名を表示します。
Type	保護されたボリュームのオーバーレイの種類を表示します。[DISK][RAM][RAM(REG)]のいずれかになります。
State	保護されたボリュームのオーバーレイの状態を表示します。[ENABLED][DISABLED]のいずれかになります。
Boot Command	保護されたボリュームの再起動時に実行するコマンドを表示します。[NO_CMD][ENABLE][DISABLE][COMMIT][SET_LEVEL]のいずれかになります。

表 2-11-1-2. EWF Config Tool 設定内容

コマンド	設定内容
Enable	EWF で保護されたボリュームで現在無効になっているオーバーレイを有効にします。
Disable	EWF で保護されたボリュームで現在有効になったオーバーレイを無効にします。
No Command	現在設定されているコマンドを取り消します。

２－１１－２ Volume Infomation

ボリュームの状態を表示します。

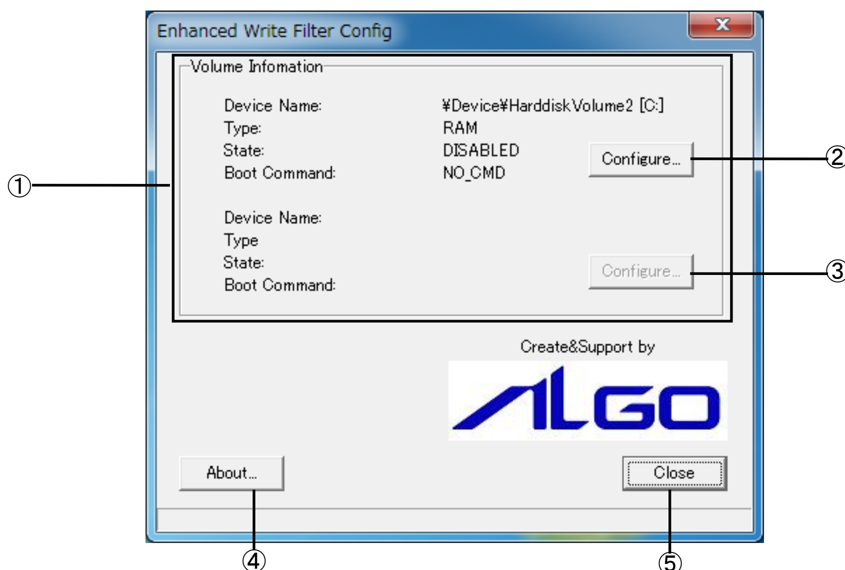

図 2-11-2-1. EWF Config – Volume Infomation

① 保護されたボリュームの現在の状況を表示します。
② 1 つ目の保護されたボリュームの設定画面に移行します。
③ 2 つ目の保護されたボリュームの設定画面に移行します。
④ 設定ツールのバージョン情報を読み出します。
⑤ 終了します。

２－１１－３ EWF Configuration

再起動時に実行するコマンドの変更を行うことができます。

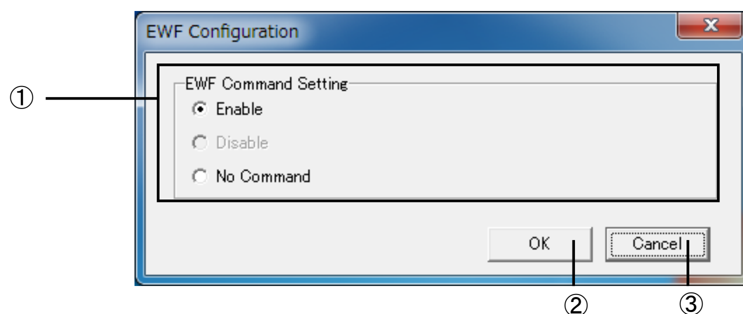

図 2-11-3-1. EWF Config – EWF Configuration

① 再起動時に実行するコマンドを選択します(※)。
② ①で選択したコマンドを有効にして Volume Information 画面に戻ります。
③ 設定を変更しないで Volume Information 画面に戻ります。

※ 保護されたボリュームが現在有効の場合は、[Disable]と[No Command]が、無効の場合は[Enable]と[No Command]がそれぞれ設定可能です。

２－１２　Ramdisk Drive Config Tool

２－１２－１ Ramdisk Drive Config Tool

「Ramdisk Drive Config Tool」は、RAM ディスクの設定を行うためのコントロールパネルアプリケーションです。スタートメニューから[コントロール パネル]を開き、[Ramdisk Config]から起動できます。設定内容を表 2-12-1-1 に示します。

表 2-12-1-1. Ramdisk Drive Config Tool 設定/表示内容

項目	設定/表示内容
Disk Size	RAM ディスクのディスク容量を設定します。 1MByte 単位で容量を設定することができます。
Drive Letter	RAM ディスクのドライブレターを設定します。

２－１２－２ Ramdisk Drive Configuration

RAM ディスクの設定を行うことができます。設定は再起動後に有効となります。

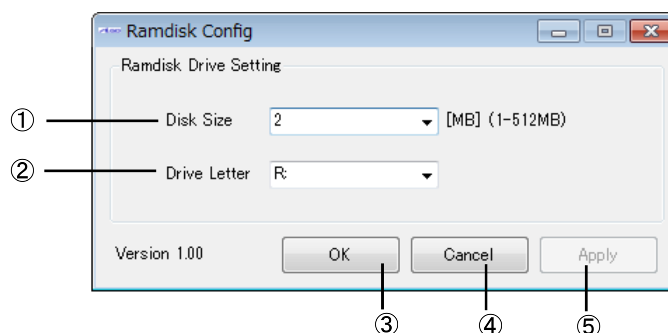

図 2-12-2-1. Ramdisk Drive Configuration

① ディスク容量を設定します。
② ドライブレターを設定します。
③ 設定を保存、適用して終了します。
④ 設定を破棄して終了します。
⑤ 設定を保存、適用します。

２－１２－３ 初期値

RAM ディスクの設定初期値を表 2-12-3-1 に示します。

表 2-12-3-1. Ramdisk Drive Config Tool 設定初期値

項目	初期値
Disk Size	2
Drive Letter	R:

２－１３　RAS Config Tool

２－１３－１ RAS Config Tool

「RAS Config Tool」は、FP シリーズ専用 RAS 機能の設定/表示を行うためのコントロールパネルアプリケーションです。スタートメニューから[コントロール パネル]を開き、[RAS Config]から起動できます。設定内容を表 2-13-1-1 に示します。

表 2-13-1-1. RAS Config Tool 設定/表示内容

タブ	設定/表示内容
Temperature	CPU の Core 温度と内部温度を監視する機能の設定を行うことができます。
Watchdog Timer	ハードウェア・ウォッチドッグタイマ機能、ソフトウェア・ウォッチドッグタイマ機能の設定を行うことができます。
Secondary RTC	外部 RTC の日時、システム日時自動更新機能の設定、Wake On Rtc Timer 機能の設定を行うことができます。
Backup Battery	バックアップバッテリの状態を確認できます。

２－１３－２ Temperature

「Temperature」は、CPU の Core 温度と内部温度を監視する機能の設定を行うことができます。内部温度の設定内容を表 2-13-2-1、CPU Core 温度の設定内容を表 2-13-2-2 に示します。

表 2-13-2-1. Ext Temperature 設定/表示内容

項目	設定/表示内容
AbnormalTime	内部温度の異常判定時間を設定します。 有効設定値: 0～65535 タイマ時間: 設定値 sec
High Threshold	内部温度の高温閾値と有効/無効を設定します。
Action	内部温度の異常時の動作を設定します。 ・Shutdown（シャットダウン） ・Reboot（再起動） ・Popup（ポップアップ通知） ・Event（ユーザーイベント通知） ・None（何もしない）
Temperature	内部温度を表示します。

表 2-13-2-2. CPU Temperature 設定/表示内容

項目	設定/表示内容
AbnormalTime	CPU Core 温度の異常判定時間を設定します。 有効設定値: 0～65535 タイマ時間: 設定値 sec
High Threshold	CPU Core 温度の高温閾値と有効/無効を設定します。
Action	CPU Core 温度の異常時の動作を設定します。 ・Shutdown（シャットダウン） ・Reboot（再起動） ・Popup（ポップアップ通知） ・Event（ユーザーイベント通知） ・None（何もしない）
Temperature Core0	CPU Core #0 の温度を表示します。
Temperature Core1	CPU Core #1 の温度を表示します。
Temperature Core2	CPU Core #2 の温度を表示します。
Temperature Core3	CPU Core #3 の温度を表示します。
Temperature Core4	CPU Core #4 の温度を表示します。
Temperature Core5	CPU Core #5 の温度を表示します。
Temperature Core6	CPU Core #6 の温度を表示します。
Temperature Core7	CPU Core #7 の温度を表示します。

※　表示される CPU の数は、CPU の種類、HyperThreading の設定によって異なります。

２－１３－３ Temperature Configuration

CPU Core 温度の監視設定と内部温度の監視設定を行うことができます。

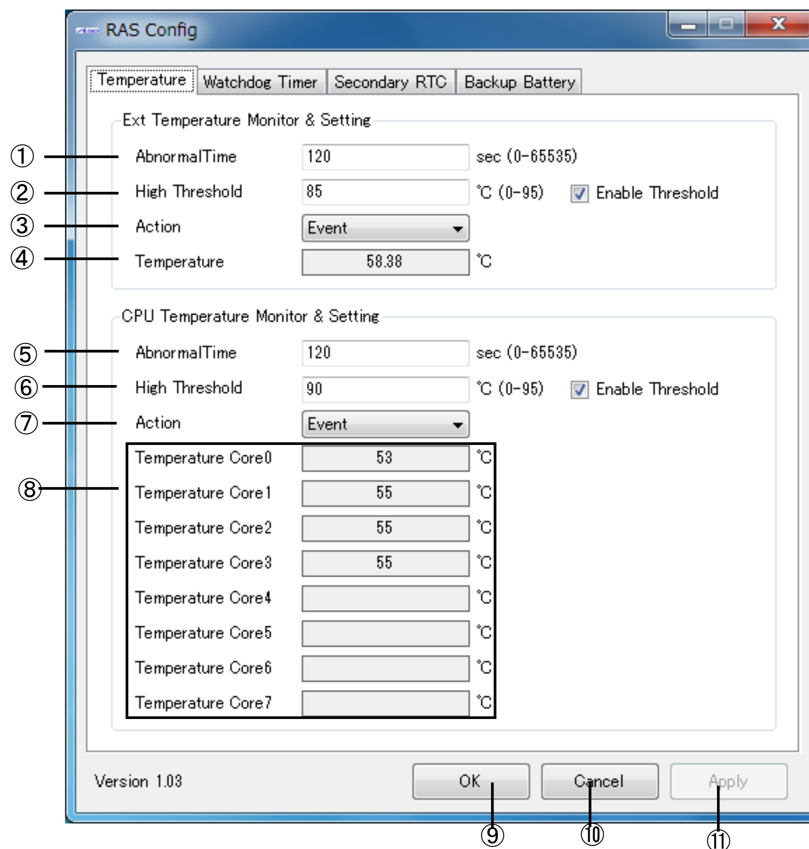

図 2-13-3-1. Temperature Configuration

① 内部温度の異常判定時間を設定します。
② 内部温度の高温閾値と有効/無効を設定します。
③ 内部温度の異常時の動作を設定します。
④ 内部温度を表示します。
⑤ CPU Core 温度の異常判定時間を設定します。
⑥ CPU Core 温度の高温閾値と有効/無効を設定します。
⑦ CPU Core 温度の異常時の動作を設定します。
⑧ CPU Core の温度を表示します。
⑨ 設定を保存、適用して終了します。
⑩ 設定を破棄して終了します。
⑪ 設定を保存、適用します。

※ この設定とは関係なく、CPU Core 温度が 95℃を超えると、強制的にシャットダウンします。

２－１３－４ Watchdog Timer

「Watchdog Timer」は、ハードウェア・ウォッチドッグタイマ機能、ソフトウェア・ウォッチドッグタイマ機能の設定を行うことができます。

「Watchdog Timer」では、ハードウェア・ウォッチドッグタイマドライバ、ソフトウェア・ウォッチドッグタイマドライバの初期値を設定/表示することができます。ハードウェア・ウォッチドッグタイマ、ソフトウェア・ウォッチドッグタイマドライバを使用する場合、ドライバオープン時にドライバ設定が「Watchdog Timer」で設定された値に初期化されます。ハードウェア・ウォッチドッグタイマの設定内容を表 2-13-4-1、ソフトウェア・ウォッチドッグタイマの設定内容を表 2-13-4-2 に示します。

表 2-13-4-1. Hardware Watchdog Timer 設定/表示内容

項目	設定/表示内容
Action	ウォッチドッグタイマのタイムアウト時の動作を設定します。 ・Power OFF（電源 OFF） ・Reset（リセット） ・Shutdown（シャットダウン） ・Reboot（再起動） ・Popup（ポップアップ通知） ・Event（ユーザーイベント通知）
Time	ウォッチドッグタイマのタイマ時間を設定します。 有効設定値: 1～160 タイマ時間: 設定値 x100msec
Enable output message for Windows Event Log	ウォッチドッグタイマのタイムアウト時に、イベントログにメッセージを記録するかどうかを設定します。

表 2-13-4-2. Software Watchdog Timer 設定/表示内容

項目	設定/表示内容
Action	ウォッチドッグタイマのタイムアウト時の動作を設定します。 ・Shutdown（シャットダウン） ・Reboot（再起動） ・Popup（ポップアップ通知） ・Event（ユーザーイベント通知）
Time	ウォッチドッグタイマのタイマ時間を設定します。 有効設定値: 1～160 タイマ時間: 設定値 x100msec
Enable output message for Windows Event Log	ウォッチドッグタイマのタイムアウト時に、イベントログにメッセージを記録するかどうかを設定します。

● ハードウェア・ウォッチドッグタイマのタイムアウト時の動作について

ハードウェア・ウォッチドッグタイマではタイムアウト時の動作として、ソフトウェア・ウォッチドックタイマには無い、「Power OFF」、「Reset」が選択できます。「Power OFF」を選択した場合、タイムアウト時は POWER スイッチの長押しと同じ状態となります。「Reset」を選択した場合は、RESET スイッチを押した状態と同じ状態となります。これらの場合はシャットダウン処理が行われないため、ファイルの破損などシステムにダメージを与える可能性があります。EWF 機能を使用してシステムの保護を行って使用するようにしてください。

「Power OFF」を選択する場合、電源オプション設定で POWER スイッチが押されたときの動作を設定する必要があります。[コントロール パネル]から[電源オプション]、「電源ボタンの動作を選択する」の順に選択し、[電源ボタンを押したときの動作]を[何もしない]に設定してください。（図 2-13-4-1）

図 2-13-4-1. 電源オプションの設定

２－１３－５ Watchdog Timer Configuration

ハードウェア・ウォッチドッグタイマ機能、ソフトウェア・ウォッチドッグタイマ機能の設定を行うことができます。

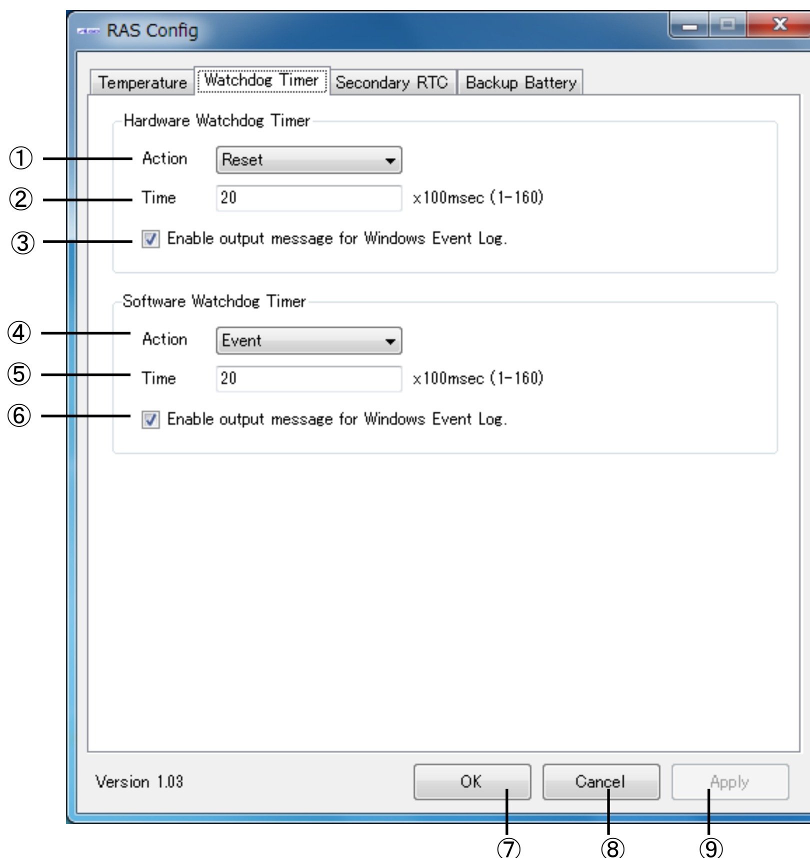

図 2-13-5-1. Watchdog Timer Configuration

① ハードウェア・ウォッチドッグタイマ タイムアウト動作を設定します。
② ハードウェア・ウォッチドッグタイマ タイマ時間を設定します。
③ ハードウェア・ウォッチドッグタイマ イベントログへの記録を指定します。
④ ソフトウェア・ウォッチドッグタイマ タイムアウト動作を設定します。
⑤ ソフトウェア・ウォッチドッグタイマ タイマ時間を設定します。
⑥ ソフトウェア・ウォッチドッグタイマ イベントログへの記録を指定します。
⑦ 設定を保存、適用して終了します。
⑧ 設定を破棄して終了します。
⑨ 設定を保存、適用します。

２－１３－６ Secondary RTC

「Secondary RTC」は、外部 RTC の日時、システム日時自動更新機能、Wake On Rtc Timer 機能の設定を行うことができます。設定内容を表 2-13-6-1 に示します。

表 2-13-6-1. Secondary RTC 設定/表示内容

項目	設定/表示内容
Date	外部 RTC の日付を設定/表示します。 値を編集すると表示の更新は停止します。
Time	外部 RTC の時刻を設定/表示します。 値を編集すると表示の更新は停止します。
Disable Auto Update	自動更新機能を無効に設定します。
Enable System Auto Update	自動的に外部 RTC の日時をシステム日時に更新します。
Interval	システム日時自動更新の更新間隔時間を設定します。 有効設定値: 1～65535 更新間隔時間: 設定値 sec (Enable Auto Update が ON の時のみ有効)
System Time	現在のシステム日時を表示します。
Disable Wake On Rtc Timer	Wake On Rtc Timer 機能を無効に設定します。
Enable Wake On Rtc Timer (Week/Hour/Min)	「曜日」指定の Wake On Rtc Timer 機能を有効に設定します。
Enable Wake On Rtc Timer (Day/Hour/Min)	「日」指定の Wake On Rtc Timer 機能を有効に設定します。
Week	Wake On Rtc Timer 機能を使用したい曜日を設定します。 設定値: 日/月/火/水/木/金/土 (Enable Wake On Rtc Timer(Week/Hour/Min)が ON の時のみ有効)
Day	Wake On Rtc Timer 機能を使用したい日を設定します。 日設定: 1～31 (Enable Wake On Rtc Timer(Day/Hour/Min)が ON の時のみ有効)
Hour	Wake On Rtc Timer 機能を使用したい時を設定します。 チェックを有効にしますと Hour を対象外に設定できます。 時設定: 0～23 (Enable Wake On Rtc Timer が ON の時のみ有効)
Min	Wake On Rtc Timer 機能を使用したい分を設定します。 チェックを有効にしますと Min を対象外に設定できます。 分設定: 0～59 (Enable Wake On Rtc Timer が ON の時のみ有効)

２－１３－７ Secondary RTC Configuration

外部 RTC の日時設定、システム日時自動更新機能、Wake On Rtc Timer 機能の設定を行うことができます。

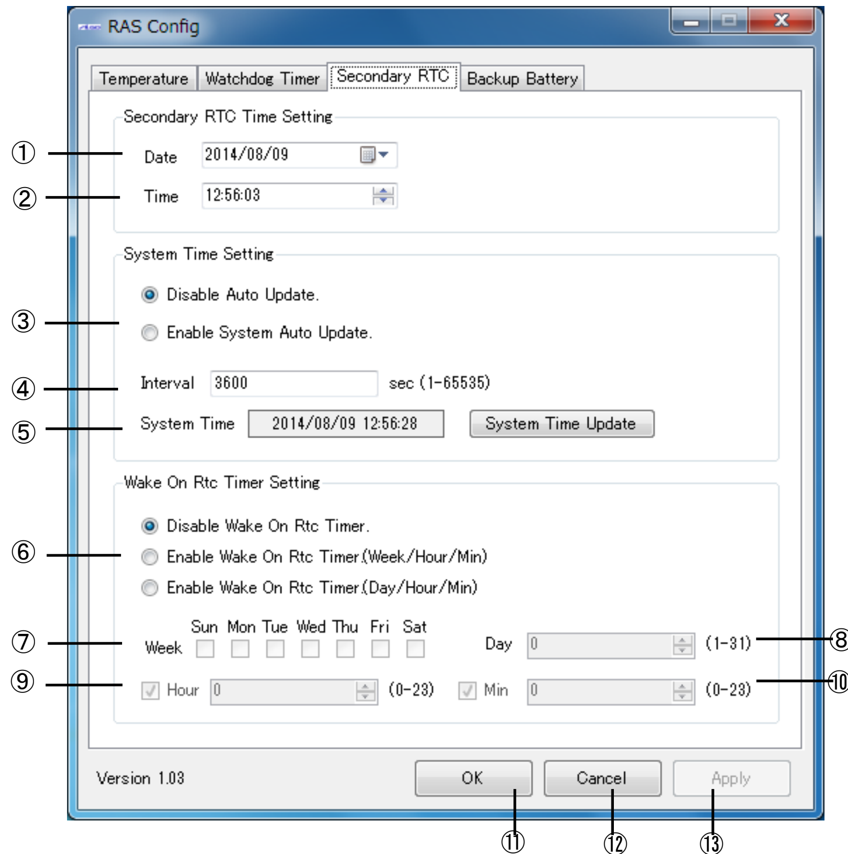

図 2-13-7-1. Secondary RTC Configuration

① 外部 RTC の日付を表示します。また、この日付を変更した状態で『OK』『Apply』ボタンを押下すると、変更した日付を外部 RTC およびシステムの日付に反映させます。

② 外部 RTC の時刻を表示します。また、この時刻を変更した状態で『OK』『Apply』ボタンを押下すると、変更した時刻を外部 RTC およびシステムの時刻に反映させます。

③ Auto Update 機能の設定を行います。

『Disable Auto Update』

Auto Update 機能を無効にします。

『Enable System Auto Update』

System Auto Update 機能を有効にします。

OS 起動時に外部 RTC の日時で、システム日時と内部 RTC の日時を初期化します。

④で設定した間隔ごとに外部 RTC の日時で、システム日時と内部 RTC の日時を更新します。

④ ③の Auto Update 機能の更新間隔時間を設定します。

⑤ システム日時を表示します。

⑥　Wake On Rtc Timer 機能の設定を行います。
　『Disable Wake On Rtc Timer.』
　　Wake On Rtc Timer 機能を無効にします。
　『Enable Wake On Rtc Timer.(Week/Hour/Min)』
　　「曜」指定の Wake On Rtc Timer 機能を有効にします。
　　⑦⑨⑩で設定した間隔で端末が起動します。
　『Enable Wake On Rtc Timer.(Day/Hour/Min)』
　　「日」指定の Wake On Rtc Timer 機能を有効にします。
　　⑧⑨⑩で設定した間隔で端末が起動します。

⑦　曜日を設定します。(Enable Wake On Rtc Timer.(Week/Hour/Min)が ON の時のみ有効)
⑧　日を設定します。(Enable Wake On Rtc Timer.(Day/Hour/Min)が ON の時のみ有効)
⑨　時を設定します。
⑩　分を設定します。
⑪　設定を保存、適用して終了します。
⑫　設定を破棄して終了します。
⑬　設定を保存、適用します。

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

２－１３－８ Wake On Rtc Timer 設定例

Wake On Rtc Timer の設定例を表 2-13-8-1、表 2-13-8-2 に示します。

表 2-13-8-1.「曜」指定時の Wake On Rtc Timer 設定例

「曜」指定時	Sun	Mon	Tue	Wed	Thu	Fri	Sat	Hour	Min
毎週月～金　午前 07 時　※「分」不問	OFF	ON	ON	ON	ON	ON	OFF	7	チェック有効
毎週土・日　毎時 30 分　※「時」不問	ON	OFF	OFF	OFF	OFF	OFF	ON	チェック有効	30
毎時　午後 6 時 59 分	ON	ON	ON	ON	ON	ON	ON	18	59

表 2-13-8-2.「日」指定時の Wake On Rtc Timer 設定例

「曜」指定時	Day	Hour	Min
毎月 01 日　午前 07 時　※「分」不問	1	7	チェック有効
毎月 15 日　毎時 30 分　※「時」不問	15	チェック有効	30
毎月 29 日　午後 6 時 59 分	29	18	59

２－１３－９ Backup Battery Monitor

バックアップバッテリの状態を確認することができます。

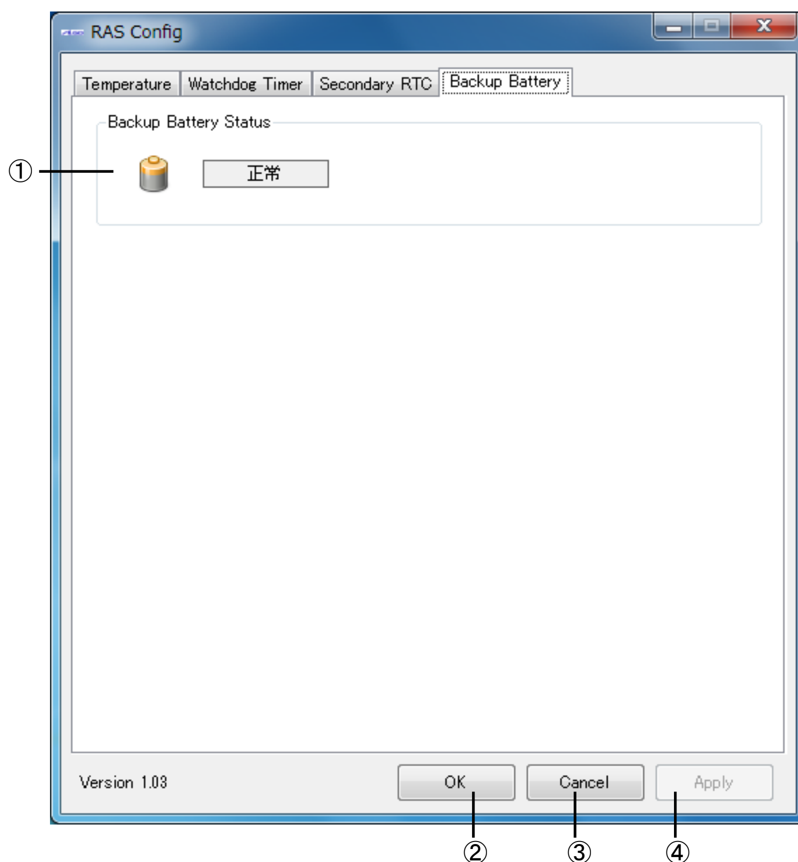

図 2-13-9-1. Backup Battery Monitor

① バックアップバッテリの状態を表示します。（正常・低下）
② 設定を保存、適用して終了します。
③ 設定を破棄して終了します。
④ 設定を保存、適用します。

２－１３－１０初期値

「RAS Config Tool」の設定初期値を表 2-13-10-1 に示します。

表 2-13-10-1. RAS Config Tool 設定初期値

タブ	設定項目	初期値
Temperature	Ext Temperature AbnormalTime	120
	Ext Temperature High Threshold	85
	Ext Temperature High Enable	ON
	Ext Temperature Action	Event
	CPU Temperature AbnormalTime	120
	CPU Temperature High Threshold	90
	CPU Temperature High Enable	ON
	CPU Temperature Action	Event
Watchdog Timer	Hardware Watchdog Action	Reset
	Hardware Watchdog Timer	20
	Hardware Watchdog Enable output message for Windows Event Log	ON
	Software Watchdog Action	Event
	Software Watchdog Timer	20
	Software Watchdog Enable output message for Windows Event Log	ON
Secondary RTC	Date	2010/1/1
	Time	00:00:00
	Disable Auto Update	ON
	Enable System Auto Update	OFF
	Interval	60
	Disable Wake On Rtc Timer	ON
	Enable Wake On Rtc Timer (Week/Hour/Min)	OFF
	Enable Wake On Rtc Timer (Day/Hour/Min)	OFF
	Sun	OFF
	Mon	OFF
	Tue	OFF
	Wed	OFF
	Thu	OFF
	Fri	OFF
	Sat	OFF
	Day	1
	Hour	0
	Min	0

２－１４　ユーザーアカウント制御

Windows Embedded Standard 7 には、問題を起こす可能性のあるプログラムからコンピュータを保護する、ユーザーアカウント制御（UAC）機能が搭載されています。

ユーザーアカウント制御機能は、管理者レベルのアクセス許可を必要とする変更が行われる前に、ユーザーに対して通知を行います。設定レベルを変更することで、ユーザーアカウント制御機能による通知の頻度を変えることができます。

ユーザーアカウント制御の初期設定を表 2-14-1 に示します。

表 2-14-1. ユーザーアカウント制御初期設定値

設定	内容
通知しない	・使用しているコンピュータに対して変更が行われるときにも通知は行われません。管理者としてログオンしている場合、自分の知らないうちに、コンピュータが変更される可能性があります。 ・標準ユーザーとしてログオンしている場合、管理者のアクセス許可を必要とする変更は自動的に拒否されます。 ・この設定を選択した場合、コンピュータを再起動し、UAC 機能をオフにする処理を完了する必要があります。UAC 機能がオフになると、管理者としてログオンしているユーザーは、常に、管理者としてのアクセス許可を持つようになります。

以下の手順で、ユーザーアカウント制御の設定レベルを変更できます。

① スタートメニューから[コントロール パネル]を選択します。
② [コントロールパネル]から[ユーザー アカウント]、[ユーザー アカウント制御設定の変更]の順に選択します。
③ [ユーザー アカウント制御の設定]ダイアログが表示されます。スライドバーを、設定したい通知レベルに変更します。
④ [OK]ボタンを押します。
⑤ 再起動します。

２－１５　S.M.A.R.T.機能

S.M.A.R.T.は、mSATA や SSD の健康状態を自己診断する機能です。S.M.A.R.T.機能を利用することで、ディスク異常の検出や寿命の予測などに役立てることができます。

FP シリーズ用 Windows Embedded Standard 7 32 ビット版は、S.M.A.R.T.機能を利用するためのツールを搭載しています。[すべてのプログラム]→[SMART]→[iSMART]で起動できます。

第３章　産業用高機能パネル PC FP4-*******について

本章では、産業用高機能パネル PC FP4-*******シリーズ（FP シリーズ）に搭載されている機能について説明します。

３－１　産業用高機能パネル PC FP4-*******に搭載された機能について

FP シリーズにはグラフィック表示機能、通信機能、USB 機能などが搭載されています。これらの機能は Windows の標準インターフェースを使用して操作することができます。また、FP シリーズでは組込みシステム向けに独自機能が追加されています。組込みシステム機能は、専用ドライバを使用して操作することが可能です。

FP シリーズの例として、FP4-12022AN の外形図を図 3-1-1 に示します。各部名称を表 3-1-1 に示します。

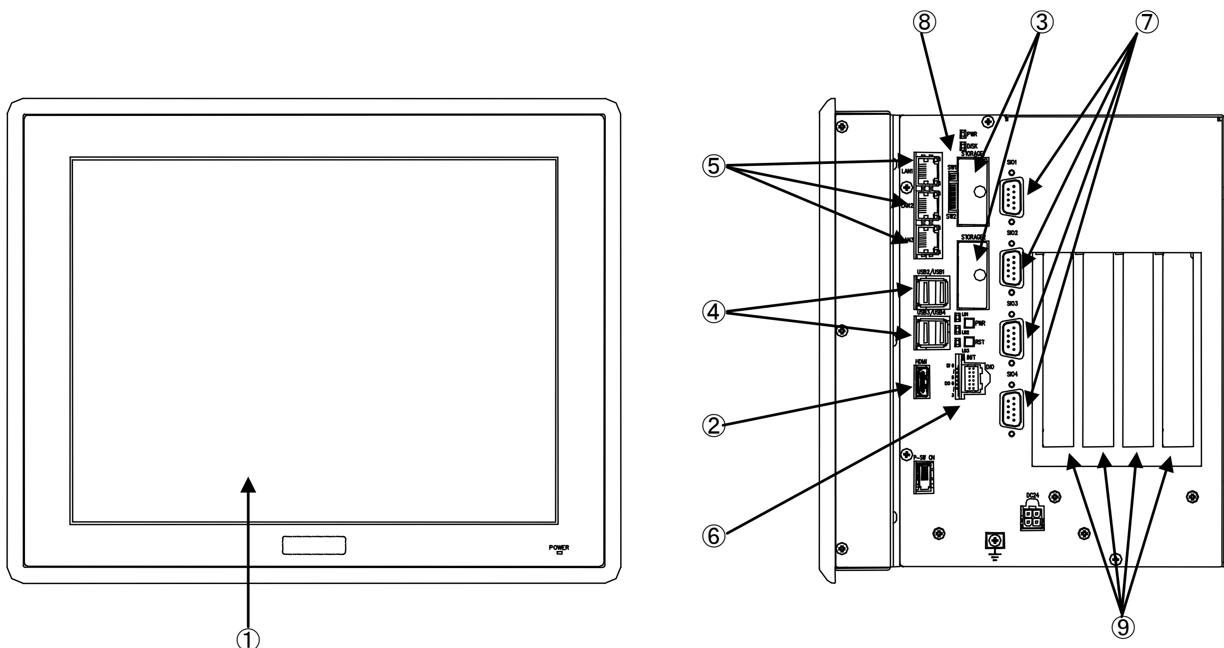

図 3-1-1. 外形図（FP4-12022AN）

表 3-1-1. 各部名称

No.	名称	機能	説明
①	液晶・タッチパネル	グラフィック タッチパネル	画面表示を行います。 タッチパネルはポインティングデバイスとして使用できます。
②	HDMI インターフェース	グラフィック サウンド	外部モニタに画面、音声を出力できます。
③	mSATA スロット	mSATA SSD	記憶領域として mSATA SSD を使用することができます。 mSATA1: メインストレージ システムドライブ(C)として使用します。 mSATA2: サブストレージ オプションでデータ領域を追加することができます。
④	USB インターフェース	USB ポート	USB1.1/2.0 の機器を接続することができます。
⑤	ネットワーク インターフェース	有線 LAN	ネットワークポートとして使用できます。
⑥	DIO インターフェース	汎用入出力	汎用の入出力です。 入力 6 点、出力 4 点を制御できます。
⑦	シリアル インターフェース	シリアルポート	シリアル通信が行えます。 COM1～COM2 は、シリアルポート設定スイッチと組込みシステム機能のシリアルコントロール機能と併用することで、232C/422/485 の通信を行うことができます。 SIO1: COM1 SIO2: COM2 SIO3: COM3 SIO4: COM4
⑧	シリアルポート 設定スイッチ	シリアルポート タイプ設定	SIO1～SIO2 のシリアルポートタイプを設定します。 シリアルポート設定スイッチと組込みシステム機能のシリアルコントロール機能を使用することで 232C/422/485 の通信を行うことができます。 設定方法は、「ユーザーズ（ハードウェア）マニュアル」を参照してください。 **※　シリアルポートタイプを切替える場合は、シリアルコントロール機能の設定に合わせて、シリアルポート設定スイッチを設定してください。**
⑨	PCI/PCI-e バス	PCI/PCI-e 拡張スロット	PCI/PCI-e 拡張ボードを使用できます。 拡張スロットの数は、機種によって異なります。

３－２　Windows 標準インターフェース対応機能

本項では、FP シリーズに搭載されている Windows 標準インターフェース対応機能について説明します。

３－２－１　グラフィック

一般的な Windows と同様に、デスクトップ表示、アプリケーション表示を行います。
スタートメニューから[コントロールパネル]を選択し、[画面]を起動して設定を行います。

３－２－２　LCD 輝度調整

LCD バックライトの輝度調整ができます。
スタートメニューから[コントロールパネル]を選択し、[電源オプション]の設定画面にある[画面の明るさ]調整バーを操作することで変更することができます。

３－２－３　タッチパネル

タッチパネルをタッチすることにより、マウスなどのポインティングデバイス操作を行うことができます。本シリーズに搭載されたタッチパネルには以下のような特徴があります。

- タッチパネルを操作することでマウスと同等な操作環境を実現することができます。
- マウスとの共存が可能なため、特別な設定を行うことなくタッチパネル、マウス双方を切替え使用することができます。
- マウス左右ボタン切替え、クリック操作に関係する詳細な設定、タッチ入力に対するイベントのカスタマイズ、精密なキャリブレーション機能などを提供します。

タッチパネルの設定、キャリブレーション機能などは[スタートメニュー]-[すべてのプログラム]-[DMC]-[DMT-DD]から行ってください。

３－２－４　シリアルポート

一般的な Windows と同様に、COM ポートとしてシリアル通信に使用することができます。アプリケーションからは COM1～COM4 が使用可能です。搭載されている COM ポートの一覧を表 3-2-4-1 に示します。

表 3-2-4-1. シリアルポート

COM ポート	説明
COM1	SIO1（図 3-1-1 ⑦） アプリケーションでシリアル通信に使用できます。 シリアルポート設定スイッチと組込みシステム機能のシリアルコントロール機能を使用することで、RS-232C/422/485 を切替えることができます。
COM2	SIO2（図 3-1-1 ⑦） アプリケーションでシリアル通信に使用できます。 シリアルポート設定スイッチと組込みシステム機能のシリアルコントロール機能を使用することで、RS-232C/422/485 を切替えることができます。
COM3	SIO3（図 3-1-1 ⑦） アプリケーションでシリアル通信に使用できます。
COM4	SIO4（図 3-1-1 ⑦） アプリケーションでシリアル通信に使用できます。
COM5	アプリケーションでシリアル通信に使用できません。

３－２－５　有線 LAN

FP シリーズにはギガビットイーサ対応の有線 LAN ポートが 3 ポート用意されています。一般的な Windows と同様にネットワークポートとして使用することができます。表 3-2-5-1 にネットワーク名称と外部コネクタとの対応を示します。

表 3-2-5-1. 有線 LAN ポート

ネットワーク名称	説明
ローカルエリア接続	LAN1 (図 3-1-1 ⑤): 1000BASE Ethernet
ローカルエリア接続 2	LAN2 (図 3-1-1 ⑤): 1000BASE Ethernet
ローカルエリア接続 3	LAN3 (図 3-1-1 ⑤): 1000BASE Ethernet

３－２－６　サウンド

サウンド機能として HDMI 音声出力を使用することができます。使用するには HDMI に HDMI 音声対応のモニタを接続する必要があります。

サウンド設定は、スタートメニューから[コントロールパネル]を表示して、[サウンド]で行ってください。

３－２－７　USB ポート

USB1.1/2.0 対応の USB ポートを外部コネクタとして 4 ポート用意しています。一般的な Windows と同様に USB 機器を接続して使用することができます。接続する USB 機器のドライバは、別途用意してください。

３－２－８　PCI 拡張スロット

PCI 拡張スロットを搭載しています。一般的な Windows と同様に PCI 拡張ボードを接続して使用することができます。接続する PCI 拡張ボードのドライバは、別途用意してください。

※　PCI 拡張スロット数は、機種によって異なります。詳細は、ハードウェアマニュアルを参照してください。

３－２－９　PCI-e 拡張スロット

PCI-e 拡張スロットを搭載しています。一般的な Windows と同様に PCI-e 拡張ボードを接続して使用することができます。接続する PCI-e 拡張ボードのドライバは、別途用意してください。

※　PCI-e 拡張スロット数は、機種によって異なります。詳細は、ハードウェアマニュアルを参照してください。

３－３　　組込みシステム機能

FP シリーズには、組込みシステム向けに独自の機能が搭載されています。本項では、組込みシステム機能について説明します。組込みシステム機能の一覧を表 3-3-1 に示します。

FP シリーズ用 Windows Embedded Standard 7 32 ビット版では、組込みシステム機能を使用するためにドライバを用意しています。ドライバの使用方法は「第 5 章 組込みシステム機能ドライバ」を参照してください。

表 3-3-1. 組込みシステム機能

機能	説明
タイマ割込み機能	ハードウェアによるタイマ機能です。 完了時にイベントを発生させることができます。
汎用入出力	汎用の入出力です。 入力 6 点、出力 4 点を制御できます。 （図 3-1-1 ⑤）
RAS 機能	汎用入力の IN0、IN1、IN4/5 にリセット機能、割込み機能、タッチパネルインターロック機能があります。IN0 リセット、IN1 割込み、IN4/5 タッチパネルインターロックの制御ができます。
シリアルコントロール機能	シリアルポートの RS-232C/422/485 の切替えができます。 **※　シリアルポートタイプを切替える場合は、シリアルコントロール機能の設定に合わせて、シリアルポート設定スイッチを設定してください。**
バックアップ SRAM	バックアップバッテリ付き SRAM を制御することができます。
ハードウェア ウォッチドッグタイマ機能	ハードウェアによるウォッチドッグタイマを操作することができます。
ソフトウェア ウォッチドッグタイマ機能	ソフトウェアによるウォッチドッグタイマを操作することができます。
RAM ディスク	システムメモリを使用したディスクドライブです。
外部 RTC	外部 RTC の日時をシステム日時に設定することができます。
Wake On Rtc Timer 機能	指定した日時に端末を起動させることができます。
温度監視	CPU Core 温度と内部温度の監視を設定することができます。
停電検出	ハードウェアによる停電検出機能を操作することができます。
バックアップバッテリモニタ	BIOS 設定、RTC、外部 RTC、SRAM に使用されるバックアップバッテリ（図 3-1-1 ⑩）の状態を取得することができます。

３－３－１　タイマ割込み機能

ハードウェアによるタイマ割込み機能が実装されています。この機能を使用すると指定した時間で周期的に割込みを発生させることができます。

アプリケーションでハードウェア割込みによる正確なタイマイベントを受けることができます。

３－３－２　汎用入出力

汎用入出力が搭載されています。アプリケーションから入力 6 点、出力 4 点が制御可能です。

３－３－３　RAS 機能

ハードウェアによる IN0 リセット機能、IN1 割込み機能、IN4/5 タッチパネルインターロック機能が実装されています。

IN0 入力時にハードウェアリセットをかけることができます。アプリケーションでこの機能の有効/無効を制御できます。

IN1 入力時に割込みを発生させることができます。アプリケーションでこの機能の有効/無効を制御できます。また、割込み発生時にイベントを受けることができます。

IN4,5 入力時にタッチパネル操作用にシリアルコントロールのインターロックを掛けることができます。

アプリケーションでこの機能の有効/無効を制御できます。また、起動時の初期状態は、「ASD Config Tool」からも設定可能です。

３－３－４　シリアルコントロール機能

COM1～COM2 は、RS-232C 以外に RS-422、RS-485 通信を行う事ができます。COM ポートごとに通信のタイプを RS-232C/RS-422/RS-485 で切替えることができます。

アプリケーションでシリアル通信を行う前に、通信のタイプを切替えることができます。また、起動時の初期状態は、「ASD Config Tool」からも設定可能です。

通信のタイプを切替える場合は、シリアルポート設定スイッチも設定に合わせて変更してください。

３－３－５　バックアップ SRAM

バックアップバッテリが搭載された SRAM が実装されています。SRAM にデータの読み書きを行うことができます。

アプリケーションで記憶領域としてバックアップ SRAM を使用することができます。

３－３－６　ハードウェア・ウォッチドッグタイマ機能

ハードウェアによるウォッチドッグタイマが実装されています。OS のハングアップ、アプリケーションのハングアップを検出できます。

３－３－７　ソフトウェア・ウォッチドッグタイマ機能

ソフトウェアによるウォッチドッグタイマが実装されています。アプリケーションのハングアップを検出できます。

３－３－８　RAM ディスク

RAM（システムメモリ）をディスクドライブとして利用することができます。通常のディスクドライブと同様にファイルの書込み、読込みを行うことができます。

ハードディスク、USB メモリなどと違い、RAM ディスクは電源が供給されていなければファイルデータを保持できません。このため、RAM ディスクは OS 起動時に初期化され、保存したファイルは OS 起動中のみ使用することができます。

RAM ディスクは、ディスク容量とドライブレターを設定することができます。ディスク容量は 1MByte から MByte 単位で設定することができます。システムメモリ容量を超える容量またはシステムメモリを圧迫するような設定をした場合、システムが不安定になるおそれがあり、最悪の場合 OS が動作しなくなる可能性があります。動作させるアプリケーション、サービスに合わせて適切なディスク容量を設定してください。

RAM ディスクの設定は「Ramdisk Drive Config Tool」を使用します。「2-12 Ramdisk Drive Config Tool」を参照してください。「Ramdisk Drive Config Tool」で変更した設定は、再起動後に有効となります。

出荷時の RAM ディスク設定を表 3-3-8-1 に示します。

表 3-3-8-1. RAM ディスク出荷時設定

設定項目	設定値
ディスク容量	2 MByte
ドライブレター	R:

RAM ディスクを使用しない場合、RAM ディスクを削除することができます。RAM ディスクを削除した場合もインストールを行えば、再度 RAM ディスクを使用できるようになります。RAM ディスクの削除手順、インストール手順を以下に示します。

● RAM ディスクの削除

① スタートメニューから[コントロール パネル]を選択します。
② [コントロール パネル]が開きます。[システム]を実行します。
③ ダイアログが開きます。[デバイス マネージャー]を選択します。
④ [デバイス マネージャー]が開きます。（図 3-3-8-1）

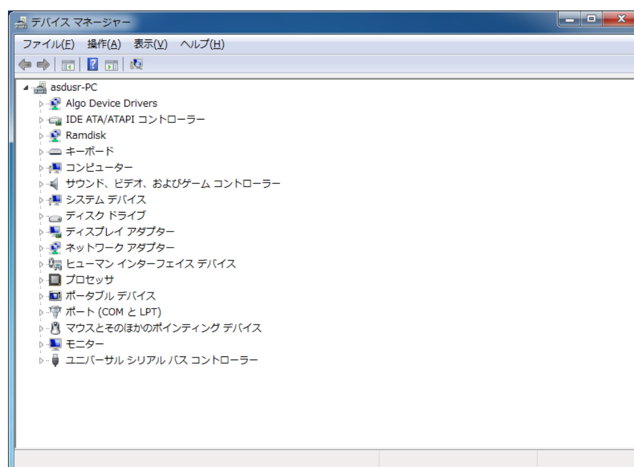

図 3-3-8-1. デバイス マネージャー

⑤ デバイスツリーの[Algo Device Drivers]を選択し、ツリーを展開します。
⑥ [Ramdisk Drive Driver]を右クリックしてメニューを開き、[削除]を選択します。(図 3-3-8-2)

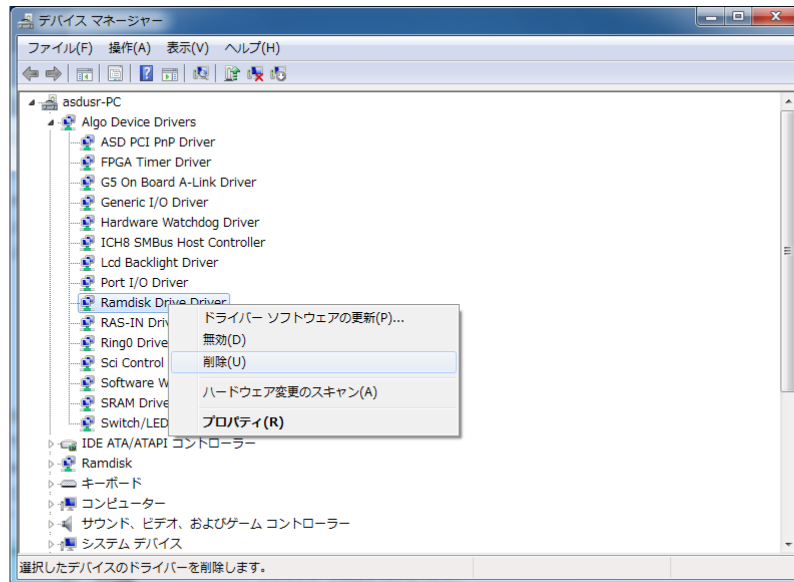

図 3-3-8-2. Ramdisk Drive Driver 削除

⑦ 再起動します。

● RAM ディスクのインストール
① スタートメニューから[すべてのプログラム]→[アクセサリ]→[コマンド プロンプト]を選択しコマンドプロンプトを開きます。
② カレントディレクトリを変更します。

```
> cd “c:¥Program Files¥ALGOSYSTEM¥Ramdisk
```

③ インストールバッチファイルを実行します。

```
> Install.bat
```

④ 再起動します。

３－３－９　外部 RTC 機能

外部の RTC が搭載されています。システム時刻を外部 RTC に同期させることができます。外部 RTC についての詳細は、「2-2 外部 RTC」を参照してください。

外部 RTC を設定するには、「RAS Config Tool」を使用します。詳細は、「2-13 RAS Config Tool」を参照してください。

３－３－１０ Wake On Rtc Timer 機能

外部 RTC を利用して、指定された日時に自動的に端末を起動することができます。

Wake On Rtc Timer 機能を設定するには、「RAS Config Tool」を使用します。詳細は、「2-13 RAS Config Tool」を参照してください。

３－３－１１ 温度監視機能

CPU Core 温度、内部温度の監視機能が実装されています。CPU Core 温度および内部温度が設定された閾値の範囲外になった場合、異常時動作を実行します。アプリケーションで異常発生時にイベントを受けることができます。

３－３－１２ 停電検出機能

ハードウェアによる停電検出機能が実装されています。

停電が発生したかどうかの判別、および停電発生時のタイムスタンプ取得を行うことができます。

３－３－１３ バックアップバッテリモニタ

BIOS、RTC、外部 RTC、SRAM のデータを保持するためのバックアップバッテリの状態を取得することができます。

※　バックアップバッテリ状態は、「正常」・「低下」を確認することができます。「低下」が確認された場合は、ハードウェアのマニュアルに従ってバックアップバッテリの交換を行ってください。

第４章　EWF API

本章では、Enhanced Write Filter（EWF）アプリケーション・プログラミング・インターフェース（API）の使用方法について説明します。

４－１　　EWF API について

EWF API は、アプリケーションから EWF の操作を行うためのインターフェースを提供します。EWF API 関数を使用すると、アプリケーションから保護ドライブの EWF 設定の変更、オーバーレイデータのコミットなどを行うことができます。

FP シリーズで使用できる EWF API 関数の一覧を表 4-1-1 に示します。

表 4-1-1. EWF API 関数一覧

関数	説明
EwfMgrGetDriveLetterFromVolumeName	指定したボリューム名のドライブ文字を取得します。
EwfMgrOpenProtected	EWF で保護されたボリュームを開きます。
EwfMgrClose	EWF で保護されたボリュームを閉じます。
EwfMgrClearCommand	次の再起動時に発生する可能性がある保留中のコマンドをクリアします。
EwfMgrDisable	EWF で保護されたボリュームで現在有効になったオーバーレイを無効にします。
EwfMgrEnable	EWF で保護されたボリュームで現在無効になっているオーバーレイを有効にします。
EwfMgrCommit	オーバーレイのデータを EWF で保護されたボリュームにすべてコミットします。
EwfMgrCommitAndDisableLive	オーバーレイのデータを EWF で保護されたボリュームに直ちにコミットした上で、EWF を無効にします。この関数は、再起動は不要です。
EwfMgrGetProtectedVolumeConfig	EWF で保護されたボリュームの構成情報を取得します。
EwfMgrGetProtectedVolumeList	すべての EWF で保護されたボリュームのリストを取得します。
EwfMgrVolumeNameListIsEmpty	EWF ボリューム名リストが空かどうかを判別します。
EwfMgrVolumeNameEntryPop	EWF ボリューム名リストから現在のエントリを削除し、現在のメモリエントリを解放します。
EwfMgrVolumeNameListDelete	EWF ボリューム名リストのすべてのエントリを削除します。

４－２　　EWF API 関数リファレンス

EwfMgrGetDriveLetterFromVolumeName 関数

機能　指定したボリューム名のドライブ文字を取得します。

書式
```
WCHAR EwfMgrGetDriveLetterFromVolumeName (
    LPCWSTR lpVolumeName
);
```

引数　lpVolumeName
[in] 指定したボリュームへのロング ポインタ。

戻り値　このメソッドが成功すると、ドライブ文字が WCHAR 型で返されます。関数が失敗すると、-1 が返されます。

説明　この関数は、EWF で保護されたボリュームとして指定されたデバイス名のドライブ文字を取得します。
ボリューム名は、任意の有効な手段で表現できます。EwfMgrGetProtectedVolumeList によって返されるボリューム名のいずれか 1 つを使用することができます。

EwfMgrOpenProtected 関数

機能

EWF で保護されたボリュームを開きます。

書式

```
HANDLE EwfMgrOpenProtected (
    LPCWSTR lpVolume
);
```

引数

lpVolume
　[in] 開くボリュームへのロング ポインタ。

戻り値

関数が成功した場合の戻り値は、デバイスへの HANDLE です。関数が失敗した場合の戻り値は INVALID_HANDLE_VALUE です。エラーの詳細情報については、GetLastError を呼び出します。

説明

この関数は、EWF で保護されているボリュームを開きます。EwfMgrClose 関数を使用して、EwfMgrOpenProtected によって返されたオブジェクトハンドルを閉じます。
ボリューム名は、任意の有効な手段で表現できます。EwfMgrGetProtectedVolumeList によって返されるボリューム名のいずれか 1 つを使用することができます。

EwfMgrClose 関数

機能　EWF で保護されたボリュームを閉じます。

書式

```
BOOL EwfMgrClose (
    HANDLE hDevice
);
```

引数

hDevice
　[in] EWF で保護されたデバイスへのハンドル。

戻り値　この関数が正常に実行された場合、戻り値は TRUE です。 この関数が正常に実行されなかった場合、戻り値は FALSE です。エラーの詳細情報については、GetLastError を呼び出します。

説明　この関数は、EWF で保護されたボリュームを閉じます。

EwfMgrClearCommand 関数

機能　次の再起動時に発生する可能性がある保留中のコマンドをクリアします。

書式

```
BOOL EwfMgrClearCommand (
    HANDLE hDevice
);
```

引数　hDevice

[in] EWF で保護されたデバイスへのハンドル。

戻り値　この関数が正常に実行された場合、戻り値は TRUE です。この関数が正常に実行されなかった場合、戻り値は FALSE です。エラーの詳細情報については、GetLastError を呼び出します。

説明　この関数は、保留のコマンドをすべてクリアします。

EwfMgrDisable 関数

機能 EWF で保護されたボリュームで現在有効になったオーバーレイを無効にします。

書式

```
BOOL EwfMgrDisable (
    HANDLE hDevice,
    BOOL fCommit
);
```

引数

hDevice

[in] EWF で保護されたボリュームへのハンドル。

fCommit

[in] TRUE の場合、保護を無効にする前に現在のオーバーレイをコミットします。FALSE の場合、すべてのオーバーレイを破棄し、保護を無効にします。

戻り値 この関数が正常に実行された場合、戻り値は TRUE です。 この関数が正常に実行されなかった場合、戻り値は FALSE です。エラーの詳細情報については、GetLastError を呼び出します。

説明 この関数は、指定したボリュームで現在有効になっているオーバーレイを無効にし、fCommit が TRUE であれば現在のレベルをコミットし、fCommit が FALSE であればすべてのオーバーレイを破棄します。EWF で保護されたボリュームとして指定されたデバイス名のドライブ文字を取得します。
オーバーレイを次の再起動時に無効にします。

EwfMgrEnable 関数

機能　EWF で保護されたボリュームで現在無効になっているオーバーレイを有効にします。

書式

```
BOOL EwfMgrEnable (
    HANDLE hDevice
);
```

引数

hDevice
　[in] EWF で保護されたボリュームへのハンドル。

戻り値　この関数が正常に実行された場合、戻り値は TRUE です。この関数が正常に実行されなかった場合、戻り値は FALSE です。エラーの詳細情報については、GetLastError を呼び出します。

説明　この関数は、特定のボリュームで現在無効になっているオーバーレイを有効にします。
オーバーレイを次の再起動時に無効にします。
ボリュームは EWF 保護用にすでに構成されており、無効状態である必要があります。

EwfMgrCommit 関数

機能　オーバーレイのデータを EWF で保護されたボリュームにすべてコミットします。

書式
```
BOOL EwfMgrCommit (
    HANDLE hDevice
);
```

引数
hDevice
　[in] EWF で保護されたボリュームへのハンドル。

戻り値　この関数が正常に実行された場合、戻り値は TRUE です。この関数が正常に実行されなかった場合、戻り値は FALSE です。エラーの詳細情報については、GetLastError を呼び出します。

説明　この関数は、オーバーレイにある現在のデータをすべて保護されたボリュームにコミットします。
オーバーレイデータを終了時にコミットします。

EwfMgrCommitAndDisableLive 関数

機能

オーバーレイのデータを EWF で保護されたボリュームに直ちにコミットした上で、EWF を無効にします。

書式

```
BOOL EwfMgrCommitAndDisableLive (
    HANDLE hDevice
);
```

引数

hDevice
    [in] EWF で保護されたボリュームへのハンドル。

戻り値

この関数が正常に実行された場合、戻り値は TRUE です。 この関数が正常に実行されなかった場合、戻り値は FALSE です。 エラー情報の詳細については、GetLastError を呼び出します。

説明

この関数は、オーバーレイの現在のすべてのデータを保護されているボリュームにコミットし、EWF を無効化します。
オーバーレイは保護されているボリュームに直ちにコミットされ、EWF は再起動を行わずに無効化されます。

EWF_VOLUME_CONFIG 構造体

機能　EWF で保護されているボリュームの構成情報を格納します。

書式

```
typedef struct _EWF_VOLUME_CONFIG
{
    EWF_TYPE Type;
    EWF_STATE State;
    EWF_COMMAND BootCommand;

    UCHAR PersistentData[EWF_MAX_PERSISTENT_DATA];
    USHORT MaxLevels;
    ULONG ClumpSize;
    USHORT CurrentLevel;

    union
    {
        struct
        {
            LONGLONG DiskMapSize;
            LONGLONG DiskDataSize;
        } DiskOverlay;

        struct
        {
            LONGLONG RamDataSize;
        } RamOverlay;
    };

    ULONG MemMapSize;
    EWF_VOLUME_DESC VolumeDesc;
    EWF_LEVEL_DESC LevelDescArray[1];
} EWF_VOLUME_CONFIG, * PEWF_VOLUME_CONFIG;
```

メンバー

Type

このボリュームのオーバーレイの種類を指定します。RAM、レジストリで説明された RAM、ディスクのいずれかになります。

State

このボリュームのオーバーレイの状態を指定します。状態は有効または無効のいずれかになります。

BootCommand

再起動時に実行するコマンドを指定します。

PersistentData

無効、有効、復元の操作を存続させるデータ。永続データのサイズは、EWF_MAX_PERSISTENT_DATA (既定では 32 バイト)です。

MaxLevels

オーバーレイのチェックポイントレベルの最大数。

ClumpSize

クランプのサイズをバイトで表示したもの。ClumpSize は 512 バイトに設定する必要があります。

CurrentLevel

現在のチェックポイントレベル。

DiskMapSize

保護されたボリュームのディスクに含まれたマッピングデータのサイズをバイトで表したもの(ディスク オーバーレイのみ)。

DiskDataSize

保護されたボリュームのディスクに保存されたデータのサイズをバイトで表したもの(ディスク オーバーレイのみ)。

RamDataSize

保護されたボリュームの RAM に保存されたデータのサイズをバイトで表したもの(RAM オーバーレイのみ)。

MemMapSize

保護されたボリュームのメモリに含まれたマッピングデータのサイズをバイトで表したもの。

VolumeDesc

ボリュームデバイス名とボリューム ID。

LevelDescArray

レベルの説明と終了時間、およびレベルのデータサイズ。エントリ数は、MaxLevels で指定します。

説明

この読み取り専用の構造体には、EWF で保護されたボリュームに関する構成情報が含まれます。

EwfMgrGetProtectedVolumeConfig 関数

機能 EWF で保護されたボリュームの構成情報を取得します。

書式

```
PEWF_VOLUME_CONFIG EwfMgrGetProtectedVolumeConfig (
    HANDLE hDevice
);
```

引数 hDevice

[in] EWF で保護されたボリュームへのハンドル。

戻り値 関数が成功したときの戻り値は、EWF_VOLUME_CONFIG 構造体へのポインタです。関数が失敗した場合の戻り値は NULL です。エラーの詳細情報については、GetLastError を呼び出します。

説明 この関数は、EWF で保護されているボリュームの構成情報を取得します。
呼び出し元は、LocalFree 関数を使用して、不要になった構造に割り当てられたメモリを解放してください。EWF が無効となっている場合、EWF_VOLUME_CONFIG 構造体の CurrentLevel メンバーは 0 に設定されます。

EwfMgrGetProtectedVolumeList 関数

機能 すべての EWF で保護されたボリュームのリストを取得します。

書式

```
PEWF_VOLUME_NAME_ENTRY EwfMgrGetProtectedVolumeList (VOID);
```

引数 なし

戻り値 この関数が正常に実行された場合、戻り値は EWF_VOLUME_NAME_ENTRY 一覧へのポインタになります。関数が失敗した場合の戻り値は NULL です。エラーの詳細情報については、GetLastError を呼び出します。

説明 この関数は、すべての EWF で保護されたボリューム名の一覧を取得します。
呼び出し元は、一覧が不要になったら、EwfMgrVolumeNameListDelete 関数を使用して一覧を解放する必要があります。

EwfMgrVolumeNameListIsEmpty 関数

機能 EWF ボリューム名リストが空かどうかを判別します。

書式

```
BOOL EwfMgrVolumeNameListIsEmpty (
    PEWF_VOLUME_NAME_ENTRY pVolumeNameList
);
```

引数 pVolumeNameList
[in] EWF_VOLUME_NAME_LIST へのポインタ。

戻り値 リストが空だと、戻り値は TRUE になります。リストが空でないと、戻り値は FALSE になります。

説明 この関数は、EWF ボリューム名リスト(EWF_VOLUME_NAME_LIST)が空かどうかを判断します。

EwfMgrVolumeNameEntryPop 関数

機能

EWF ボリューム名リストから現在のエントリを削除し、現在のメモリエントリを解放します。

書式

```
VOID EwfMgrVolumeNameEntryPop (
    PEWF_VOLUME_NAME_ENTRY* ppVolumeNameList
);
```

引数

ppVolumeNameList

[in, out] EWF_VOLUME_NAME_LIST へのポインタのポインタ。

戻り値

この関数は、空の一覧と共に呼び出される場合があります。この関数により、ppVolumeNameList リンクリストのヘッドノードも変更されます。

説明

この関数により、EWF のボリューム名一覧(EWF_VOLUME_NAME_LIST)の現在のエントリを削除し、そのメモリが解放されます。

EwfMgrVolumeNameListDelete 関数

機能　EWF ボリューム名リストのすべてのエントリを削除します。

書式
```
VOID EwfMgrVolumeNameListDelete (
    PEWF_VOLUME_NAME_ENTRY pVolumeNameList
);
```

引数　pVolumeNameList
　[in] EWF_VOLUME_NAME_LIST へのポインタ。

戻り値　なし

説明　この関数により、EWF ボリュームの名前一覧(EWF_VOLUME_NAME_LIST)のすべてのエントリが削除されます。
この関数は、空の一覧と共に呼び出される場合があります。

４－３　　EWF API 関数の使用について

「FP シリーズ用 Windows Embedded Standard 7 32 ビット版 リカバリ/SDK DVD」に EWF API 関数を使用するためのヘッダファイル、ライブラリファイル、EWF API 関数を使用したサンプルコードを用意しています。開発用ファイルは一般的な C/C++言語用です。Microsoft Visual Studio など Windows API を使用できる C/C++言語の開発環境で使用することが可能です。DVD に含まれる開発用ファイルの内容を表 4-3-1 に示します。

表 4-3-1. リカバリ/SDK DVD EWF API 開発用ファイル

DVD-ROM のフォルダ	内容
¥SDK¥EWFAPI¥Develop	EWF API 関数を使用するためのヘッダファイル、ライブラリファイルを格納しています。
¥SDK¥EWFAPI¥Sample	EWF API 関数を使用したサンプルコードを格納しています。

４－４　　サンプルコード

４－４－１　　EWF の有効／無効

● EWF の有効

「¥SDK¥EWFAPI¥Sample¥EwfEnable.cpp」では、EwfMgrEnable 関数を使用して C ドライブの EWF を有効にしています。リスト 4-4-1-1 にサンプルコードを示します。

リスト 4-4-1-1. EWF の有効

```
#include <windows.h>
#include <stdio.h>
#include <stdlib.h>

#define EWFIMP
#include "ewfapi.h"

/*++
ルーチンの説明:
  このルーチンでは、EwfMgrEnable の用法を示す。
  保護されたボリュームへのハンドルを開き、次にコマンドを発行する。

  EwfMgrEnable:
    コマンド: EWF を有効にします。

    再起動: 必要。

引数:
  szProVolName
    EwfMgrGetProtectedVolumeList から取得するボリューム名または
    前に形式 ¥¥ が付いた EwfMgrGetOverlayStoreConfig から取得するデバイス名。
    ¥C: ここで C は、保護されたボリュームのドライブ名を表す。

戻り値:
  成功した場合は TRUE、失敗した場合は FALSE。
--*/
```

```
BOOL DoEwfEnable(LPWSTR szProVolName)
{
    HANDLE hProVol = INVALID_HANDLE_VALUE;
    BOOL bResult = FALSE;

    // ボリューム名を使ってこの保護されたボリュームへのハンドルを開く。
    hProVol = EwfMgrOpenProtected(szProVolName);
    if(hProVol == INVALID_HANDLE_VALUE){
        wprintf(L"EwfMgrOpenProtected failed LE = %u¥n", GetLastError());
        goto __exit;
    }

    // EWF を有効にする。
    bResult = EwfMgrEnable(hProVol);
    if(!bResult){
        wprintf(L"EwfMgrEnable failed LE = %u¥n", GetLastError());
        goto __exit;
    }
    wprintf(L"EwfMgrEnable succeeded¥n");

__exit:
    if(hProVol != INVALID_HANDLE_VALUE){
        EwfMgrClose(hProVol);
    }
    return bResult;
}

int main(void)
{
    BOOL bRet;

    // C ドライブを EWF 有効にします。
    bRet = DoEwfEnable(L"¥¥¥¥.¥¥C:");

    return (bRet ? 0 : -1);
}
```

● EWF の無効

「¥SDK¥EWFAPI¥Sample¥EwfDisable.cpp」では、EwfMgrDisable 関数を使用して C ドライブの EWF を無効にしています。リスト 4-4-1-2 にサンプルコードを示します。

リスト 4-4-1-2. EWF の無効

```
#include <windows.h>
#include <stdio.h>
#include <stdlib.h>

#define EWFIMP
#include "ewfapi.h"

/*++
```

```
ルーチンの説明:
  このルーチンでは、EwfMgrDisable の用法を示す。
  保護されたボリュームへのハンドルを開き、次にコマンドを発行する。

  EwfMgrDisable:
    コマンド: 指定したボリュームで現在有効になっているオーバーレイ
              を無効にする。
    再起動: 必要。

引数:
  szProVolName
    EwfMgrGetProtectedVolumeList から取得するボリューム名または
    前に形式 ¥¥ が付いた EwfMgrGetOverlayStoreConfig から取得するデバイス名。
    ¥C: ここで C は、保護されたボリュームのドライブ名を表す。

戻り値:
  成功した場合は TRUE、失敗した場合は FALSE。
--*/

BOOL DoEwfDisable(LPWSTR szProVolName)
{
    HANDLE hProVol = INVALID_HANDLE_VALUE;
    BOOL bResult = FALSE;

    // ボリューム名を使ってこの保護されたボリュームへのハンドルを開く。
    hProVol = EwfMgrOpenProtected(szProVolName);
    if(hProVol == INVALID_HANDLE_VALUE){
        wprintf(L"EwfMgrOpenProtected failed LE = %u¥n", GetLastError());
        goto __exit;
    }

    // EWF を無効にする。
    // 2 番目のパラメータはコミットするかどうか。
    bResult = EwfMgrDisable(hProVol, FALSE);
    if(!bResult){
        wprintf(L"EwfMgrDisable failed LE = %u¥n", GetLastError());
        goto __exit;
    }
    wprintf(L"EwfMgrDisable succeeded¥n");

__exit:
    if(hProVol != INVALID_HANDLE_VALUE){
        EwfMgrClose(hProVol);
    }
    return bResult;
}

int main(void)
{
    BOOL bRet;

    // C ドライブを EWF 無効にします。
```

```
    bRet = DoEwfDisable(L"¥¥¥¥.¥¥C:");

    return (bRet ? 0 : -1);
}
```

４－４－２　オーバーレイデータのコミット

● C ドライブのオーバーレイデータのコミット

「¥SDK¥EWFAPI¥Sample¥EwfCommit.cpp」では、EwfMgrCommit 関数を使用して C ドライブのオーバーレイデータをコミットしています。リスト 4-4-2-1 にサンプルコードを示します。

リスト 4-4-2-1. オーバーレイデータコミット

```
#include <windows.h>
#include <stdio.h>
#include <stdlib.h>

#define EWFIMP
#include "ewfapi.h"

/*++
ルーチンの説明:
  このルーチンでは、EwfMgrCommit の用法を示す。
  保護されたボリュームへのハンドルを開き、次にコマンドを発行する。

  EwfMgrCommit:
    コマンド: オーバーレイの情報をコミットします。

    再起動: 必要。

引数:
  szProVolName
    EwfMgrGetProtectedVolumeList から取得するボリューム名または
    前に形式 ¥¥ が付いた EwfMgrGetOverlayStoreConfig から取得するデバイス名。
    ¥C: ここで C は、保護されたボリュームのドライブ名を表す。

戻り値:
  成功した場合は TRUE、失敗した場合は FALSE。
--*/

BOOL DoEwfCommit(LPWSTR szProVolName)
{
    HANDLE hProVol = INVALID_HANDLE_VALUE;
    BOOL bResult = FALSE;

    // ボリューム名を使ってこの保護されたボリュームへのハンドルを開く。
    hProVol = EwfMgrOpenProtected(szProVolName);
    if(hProVol == INVALID_HANDLE_VALUE){
        wprintf(L"EwfMgrOpenProtected failed LE = %u¥n", GetLastError());
        goto __exit;
    }
```

```
    // オーバーレイの情報をコミットします。
    bResult = EwfMgrCommit(hProVol);
    if(!bResult){
        wprintf(L"EwfMgrCommit failed LE = %u¥n", GetLastError());
        goto __exit;
    }
    wprintf(L"EwfMgrCommit succeeded¥n");

__exit:
    if(hProVol != INVALID_HANDLE_VALUE){
        EwfMgrClose(hProVol);
    }
    return bResult;
}

int main(void)
{
    BOOL bRet;

    // C ドライブをコミットします。
    bRet = DoEwfCommit(L"¥¥¥¥.¥¥C:");

    return (bRet ? 0 : -1);
}
```

● C ドライブのオーバーレイデータのコミットと EWF 無効（LIVE 処理）

「¥SDK¥EWFAPI¥Sample¥EwfCommitAndDisableLive.cpp」では、EwfMgrCommitAndDisableLive 関数を使用して C ドライブのオーバーレイデータをコミットと EWF 無効を行っています。処理は即座に行われます。リスト 4-4-2-2 にサンプルコードを示します。

リスト 4-4-2-2. コミット+EWF 無効

```
#include <windows.h>
#include <stdio.h>
#include <stdlib.h>

#define EWFIMP
#include "ewfapi.h"

/*++
ルーチンの説明:
  このルーチンでは、EwfMgrCommitAndDisableLive の用法を示す。
  保護されたボリュームへのハンドルを開き、次にコマンドを発行する。

  EwfMgrCommitAndDisableLive:
    コマンド: オーバーレイの情報をコミットし、EWF を無効にします。
    処理はすぐに実行されます。

    再起動: 不要
```

```
引数:
  szProVolName
    EwfMgrGetProtectedVolumeList から取得するボリューム名または
    前に形式 ¥¥ が付いた EwfMgrGetOverlayStoreConfig から取得するデバイス名。
    ¥C: ここで C は、保護されたボリュームのドライブ名を表す。

戻り値:
  成功した場合は TRUE、失敗した場合は FALSE。
--*/

BOOL DoEwfCommitAndDisableLive(LPWSTR szProVolName)
{
    HANDLE hProVol = INVALID_HANDLE_VALUE;
    BOOL bResult = FALSE;

    // ボリューム名を使ってこの保護されたボリュームへのハンドルを開く。
    hProVol = EwfMgrOpenProtected(szProVolName);
    if(hProVol == INVALID_HANDLE_VALUE){
        wprintf(L"EwfMgrOpenProtected failed LE = %u¥n", GetLastError());
        goto __exit;
    }

    // コミットして EWF を無効にします。
    bResult = EwfMgrCommitAndDisableLive(hProVol);
    if(!bResult){
        wprintf(L"EwfMgrCommitAndDisableLive failed LE = %u¥n", GetLastError());
        goto __exit;
    }
    wprintf(L"EwfMgrCommitAndDisableLive succeeded¥n");

__exit:
    if(hProVol != INVALID_HANDLE_VALUE){
        EwfMgrClose(hProVol);
    }
    return bResult;
}

int main(void)
{
    BOOL bRet;

    // C ドライブをコミットして EWF 無効にします。
    bRet = DoEwfCommitAndDisableLive(L"¥¥¥¥.¥¥C:");

    return (bRet ? 0 : -1);
}
```

第５章　組込みシステム機能ドライバ

FP シリーズには、組込みシステム向けに独自の機能が搭載されています。FP シリーズ用 Windows Embedded Standard 7 32 ビット版には、これら機能にアクセスするためのドライバを用意しています。このドライバを使用することでアプリケーションからこれらの機能を使用することができます。本章では、組込みシステム機能ドライバの使用方法について説明します。

５－１　ドライバの使用について

５－１－１　開発用ファイル

「FP シリーズ用 Windows Embedded Standard 7 32 ビット版 リカバリ/SDK DVD」にドライバにアクセスするためのヘッダファイルとドライバを使用したサンプルコードを用意しています。開発用ファイルは一般的な C/C++言語用です。Microsoft Visual Studio など Windows API を使用できる C/C++言語の開発環境で使用することが可能です。DVD に含まれる開発用ファイルの内容を表 5-1-1-1 に示します。

表 5-1-1-1. リカバリ/SDK DVD 開発用ファイル

DVD-ROM のフォルダ	内容
¥SDK¥Algo¥Develop	ドライバアクセスに必要なヘッダファイルを格納しています。
¥SDK¥Algo¥Sample¥Sample_GenIO	汎用入出力制御のサンプルコードです。
¥SDK¥Algo¥Sample¥Sample_Interrupt	タイマ割込み機能 IN1 割込み機能サンプルコードです。
¥SDK¥Algo¥Sample¥Sample_Reset	IN0 リセット機能のサンプルコードです。
¥SDK¥Algo¥Sample¥Sample_SerialControl	シリアルコントロール機能のサンプルコードです。
¥SDK¥Algo¥Sample¥Sample_SRAM	バックアップ SRAM のサンプルコードです。
¥SDK¥Algo¥Sample¥Sample_HwWdt	ハードウェア・ウォッチドッグのサンプルコードです。
¥SDK¥Algo¥Sample¥Sample_SwWdt	ソフトウェア・ウォッチドッグのサンプルコードです。
¥SDK¥Algo¥Sample¥Sample_TempMon	温度監視機能のサンプルコードです。
¥SDK¥Algo¥Sample¥Sample_RASDll	RAS DLL による温度取得のサンプルコードです。
¥SDK¥Algo¥Sample¥Sample_RASDll¥SecondaryRTC	外部 RTC 機能のサンプルコードです。
¥SDK¥Algo¥Sample¥Sample_Pdd	停電検出機能のサンプルコードです。
¥SDK¥Algo¥Sample¥Sample_Interlock	タッチパネルインターロックのサンプルコードです。
¥SDK¥Algo¥Sample¥Sample_BackBat	バックアップバッテリモニタのサンプルコードです。

５－１－２　DeviceIoControl について

FP シリーズ専用機能のドライバは、ほとんどのものがドライバの機能にアクセスするために DeviceIoControl 関数を使用します。以下にその書式を示します。関数仕様の詳細は、Windows API の仕様を参照してください。

コントロールコード、コントロールコードに対応する動作および引数は、ドライバごとにリファレンスを用意していますので、各ドキュメントを参照してください。

関数書式

```
BOOL DeviceIoControl (
        HANDLE          hDevice,
        DWORD           dwIoControlCode,
        LPVOID          lpInBuf,
        DWORD           nInBufSize,
        LPVOID          lpOutBuf,
        DWORD           nOutBufSize,
        LPDWORD         lpBytesReturned,
        LPOVERLAPPED    lpOverlapped
);
```

パラメータ

hDevice	： デバイス、ファイル、ディレクトリいずれかのハンドル
dwIoControlCode	： 実行する動作のコントロールコード
lpInBuf	： 入力データを供給するバッファへのポインタ
nInBufSize	： 入力バッファのバイト単位のサイズ
lpOutBuf	： 出力データを受け取るバッファへのポインタ
nOutBufSize	： 出力バッファのバイト単位のサイズ
lpBytesReturned	： lpOutBuf に格納されるバイト数を受け取る変数へのポインタ
lpOverlapped	： 非同期動作を表す構造体へのポインタ

５－２　　タイマ割込み機能

５－２－１　　タイマ割込み機能について

FP シリーズには、ハードウェアによるタイマ割込み機能が実装されています。タイマドライバを操作することによって、指定した時間で周期的に割込みを発生させることができます。

５－２－２　　タイマドライバについて

タイマドライバはタイマ割込み機能を、ユーザーアプリケーションから利用できるようにします。ユーザーアプリケーションから、タイマの設定とイベントによるタイマ通知の機能を使用することができます。

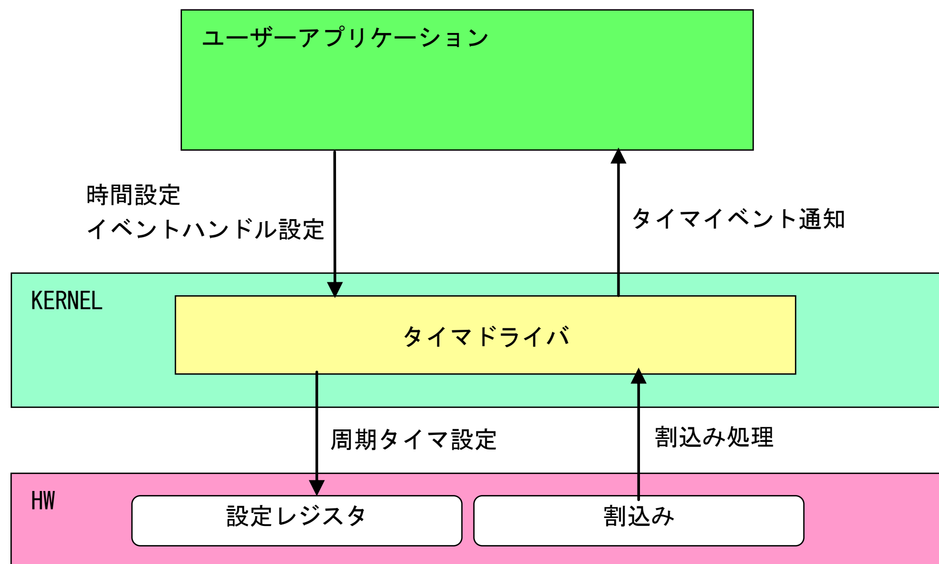

図 5-2-2-1. タイマドライバ

５－２－３　　タイマデバイス

タイマドライバはタイマデバイスを生成します。ユーザーアプリケーションは、デバイスファイルにアクセスすることによってタイマ機能を操作します。

タイマデバイス

デバイスファイル

¥¥.¥FpgaTimer

説明

タイマ時間設定、タイマ開始、停止を行うことができます。

レジストリ設定

[KEY]　　　　　HKEY_LOCAL_MACHINE¥SYSTEM¥CurrentControlSet¥Services¥FpgaTimer
[VALUE:DWORD]　TimerResolution

タイマ解像度をミリ秒単位で設定します。ドライバ起動時(OS起動時)にこの値を参照しタイマ解像度を設定します。(デフォルト値: 10)

CreateFile

デバイスファイル(¥¥.¥FpgaTimer)をオープンし、デバイスハンドルを取得します。

```
hTimer = CreateFile(
    "¥¥¥¥.¥¥FpgaTimer",
    GENERIC_READ | GENERIC_WRITE,
    FILE_SHARE_READ | FILE_SHARE_WRITE,
    NULL,
    OPEN_EXISTING,
    0,
    NULL
);
```

CloseHandle

デバイスハンドルをクローズします。

```
CloseHandle(hTimer);
```

ReadFile

使用しません。

WriteFile

使用しません。

DeviceIoControl

- **IOCTL_FPGATIMER_START**
  タイマを開始します。
- **IOCTL_FPGATIMER_STOP**
  タイマを停止します。
- **IOCTL_FPGATIMER_SETCONFIG**
  タイマを設定します。
- **IOCTL_FPGATIMER_GETCONFIG**
  現在のタイマ設定を取得します。

５－２－４　　タイマドライバの動作

① 起動時に 10msec(レジストリ設定で変更可能)の周期割込み設定を行います。
② オープンされたデバイスハンドル毎に、タイマ情報を作成しタイマ情報テーブルへ追加します。オープンできるハンドルはシステム全体で 16 までとなります。タイマ情報テーブルへの追加はオープンした順番で追加されます。
③ ユーザーアプリケーションからの設定をタイマ情報テーブルへ反映させます。
④ 周期割込みが発生したらタイマ情報テーブルを参照し、各タイマ情報のカウント値を加算します。
⑤ カウント値が設定値に達したものは、イベントハンドルでタイマ通知を行います。カウント加算、イベント通知処理はタイマ情報テーブルの順番で処理されます。

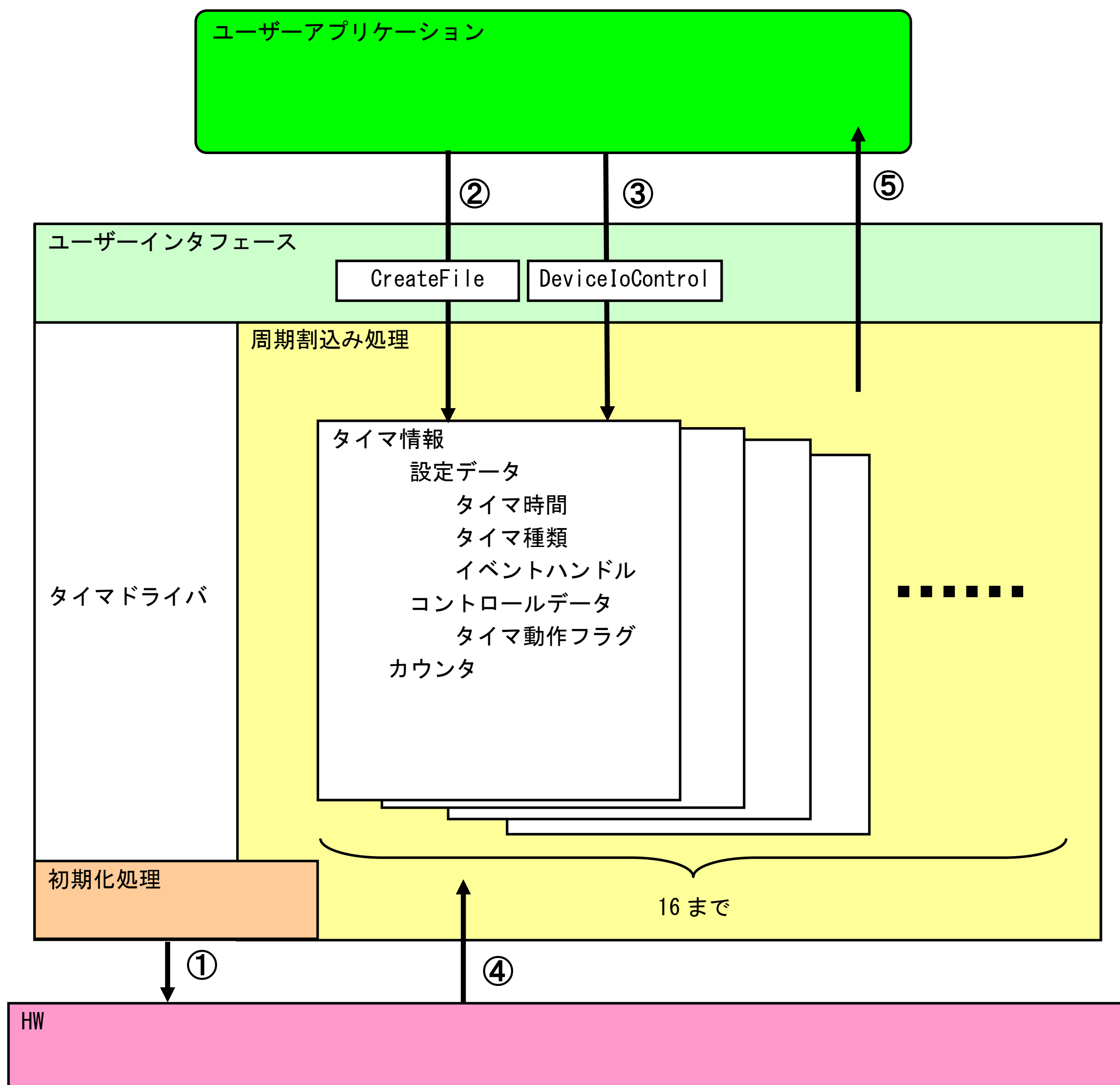

図 5-2-4-1. タイマドライバの動作

５－２－５　ドライバ使用手順

基本的な使用手順を以下に示します。タイマ通知用イベントハンドルを作成後、タイマデバイスにイベントハンドル、タイマ時間を設定します。タイマ通知用イベントハンドルでのイベント待ち準備が整ったところで、タイマをスタートさせます。

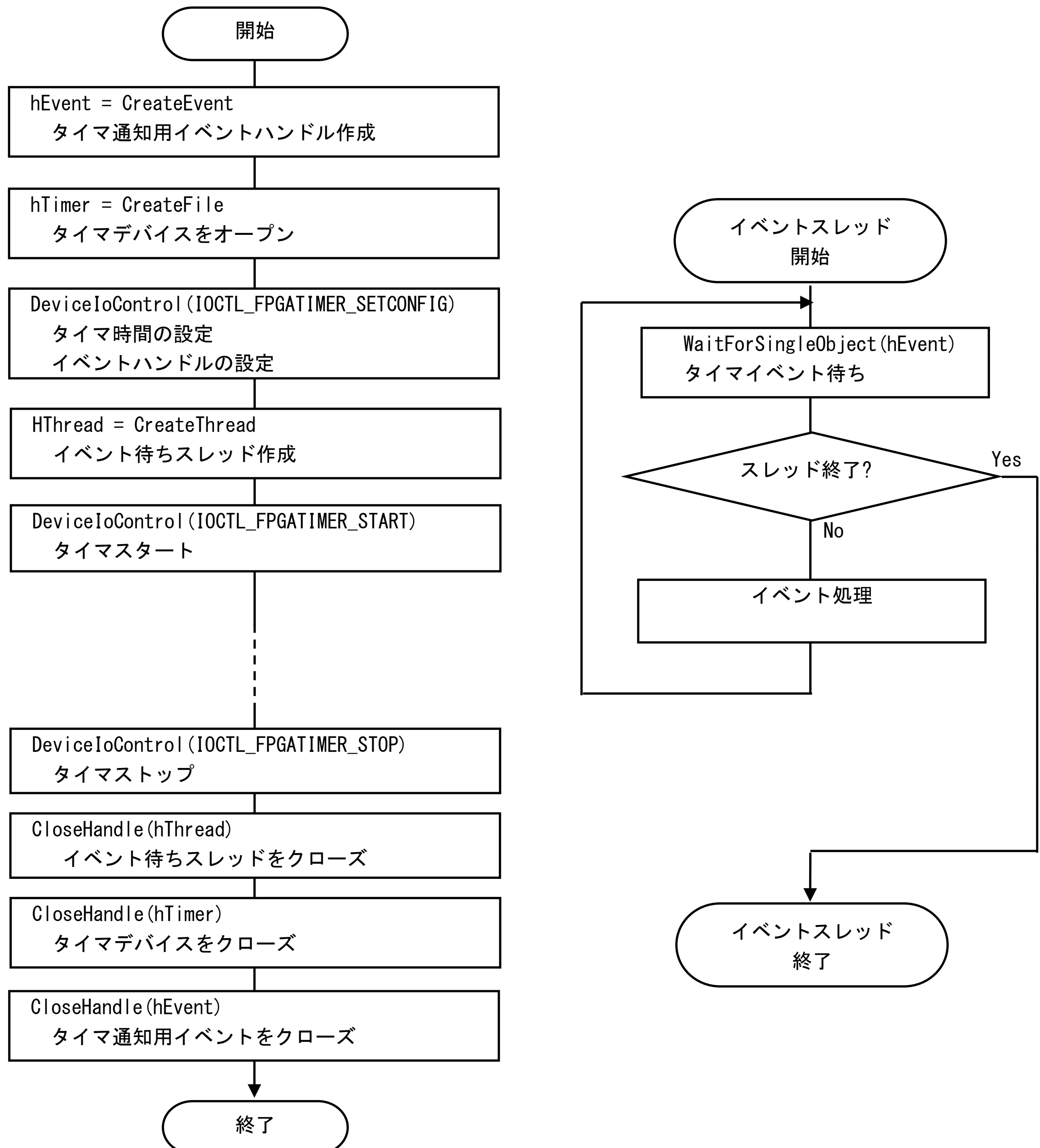

図 5-2-5-1. ドライバ使用手順

５－２－６　DeviceIoControl リファレンス

IOCTL_FPGATIMER_START

機能

タイマ処理を開始します。

パラメータ

lpInBuf	:	NULL を指定します。
NInBufSize	:	0 を指定します。
LpOutBuf	:	NULL を指定します。
NOutBufSize	:	0 を指定します。
LpBytesReturned	:	実際の出力バイト数を受け取る変数へのポインタ。
LpOverlapped	:	NULL を指定します。

戻り値

処理が成功すると TRUE を返します。失敗の場合は FALSE を返します。

説明

IOCTL_FPGATIMER_SETCONFIG に設定した内容でタイマ処理を開始します。このコントロールを実行させる前に、必ず IOCTL_FPGATIMER_SETCONFIG を実行するようにしてください。

IOCTL_FPGATIMER_STOP

機能

タイマ処理を停止します。

パラメータ

lpInBuf　：NULL を指定します。
NInBufSize　：0 を指定します。
LpOutBuf　：NULL を指定します。
NOutBufSize　：0 を指定します。
LpBytesReturned　：実際の出力バイト数を受け取る変数へのポインタ。
LpOverlapped　：NULL を指定します。

戻り値

処理が成功すると TRUE を返します。失敗の場合は FALSE を返します。

説明

タイマ処理を停止します。タイマ通知イベントハンドルを破棄する前には、このコントロールを実行してタイマ通知を停止するようにしてください。

IOCTL_FPGATIMER_SETCONFIG

機能

タイマの設定を行います。

パラメータ

lpInBuf	:	FPGATIMER_CONFIG を格納するためのポインタ。
NInBufSize	:	FPGATIMER_CONFIG のサイズを指定します。
LpOutBuf	:	NULL を指定します。
NOutBufSize	:	0 を指定します。
LpBytesReturned	:	実際の出力バイト数を受け取る変数へのポインタ。
LpOverlapped	:	NULL を指定します。

FPGATIMER_CONFIG

```
typedef struct {
    HANDLE      hEvent;
    ULONG       Type;
    ULONG       DueTime;
} FPGATIMER_CONFIG, *PFPGATIMER_CONFIG;
```

hEvent	:タイマ通知用イベントハンドル
Type	:タイマ動作タイプ [0: 一回で終了, 1: 繰り返し]
DueTime	:タイマ時間

戻り値

処理が成功すると TRUE を返します。失敗の場合は FALSE を返します。

説明

タイマの設定を行います。IOCTL_FPGATIMER_START でタイマを開始する前に、このコントロールを実行してタイマの設定を行うようにしてください。

IOCTL_FPGATIMER_GETCONFIG

機能

タイマ設定を取得します。

パラメータ

lpInBuf	:	NULL を指定します。
NInBufSize	:	0 を指定します。
LpOutBuf	:	FPGATIMER_CONFIG を格納するためのポインタ。
NOutBufSize	:	FPGATIMER_CONFIG のサイズを指定します。
LpBytesReturned	:	実際の出力バイト数を受け取る変数へのポインタ。
LpOverlapped	:	NULL を指定します。

FPGATIMER_CONFIG

```
typedef struct {
    HANDLE      hEvent;
    ULONG       Type;
    ULONG       DueTime;
} FPGATIMER_CONFIG, *PFPGATIMER_CONFIG;
```

hEvent	：タイマ通知用イベントハンドル
Type	：タイマ動作タイプ［0: 一回で終了, 1: 繰り返し］
DueTime	：タイマ時間

戻り値

処理が成功すると TRUE を返します。失敗の場合は FALSE を返します。

説明

現在のタイマ設定値を取得します。

５－２－７　サンプルコード

「¥SDK¥Algo¥Sample¥Sample_Interrupt¥FpgaTimer」にタイマ割込み機能を使用したサンプルコードを用意しています。リスト 5-2-7-1 にサンプルコードを示します。サンプルコードでは、10 個の周期タイマを使用してタイマイベント通知を確認しています。

リスト 5-2-7-1. タイマ割込み機能

```
/**
    タイマ割込み制御サンプルソース
**/
#include <windows.h>
#include <winioctl.h>
#include <stdio.h>
#include <stdlib.h>
#include <mmsystem.h>
#include <conio.h>

#include "..¥Common¥FpgaTimerDD.h"

#define TIMERDRIVER_FILENAME    "¥¥¥¥.¥¥FpgaTimer"
#define MAX_TIMEREVENT          10

//-----------------------------------------------------------------------------
typedef struct {
    int No;
    HANDLE  hEvent;
    HANDLE  hThread;
    volatile BOOL fStart;
    volatile BOOL fFinish;
    HANDLE  hTimer;
    FPGATIMER_CONFIG Config;
} TIMEREVENT_INFO, *PTIMEREVENT_INFO;

//-----------------------------------------------------------------------------
/*
 * 割込みハンドラ
 */
DWORD WINAPI TimerEventProc(void *pData)
{
    PTIMEREVENT_INFO    info = (PTIMEREVENT_INFO)pData;
    DWORD ret;

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

    info->fFinish = FALSE;
    while(1){
        if(WaitForSingleObject(info->hEvent, INFINITE) != WAIT_OBJECT_0){
            break;
        }
        if(!info->fStart){
            break;
        }
```

```
        printf("TimerEventProc: Timer%02d: Tick(%d)¥n", info->No, timeGetTime());
    }
    info->fFinish = TRUE;

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

//--------------------------------------------------------------------------
BOOL CreateTimerEventInfo(int No, PTIMEREVENT_INFO info)
{
    DWORD   thrd_id;
    ULONG   retlen;
    BOOL    ret;

    info->No = No;
    info->hEvent = NULL;
    info->hThread = NULL;
    info->fStart = FALSE;
    info->fFinish = FALSE;
    info->hTimer = INVALID_HANDLE_VALUE;

    /*
     * イベントオブジェクトの作成
     */
    info->hEvent = CreateEvent(NULL, FALSE, FALSE, NULL);
    if(info->hEvent == NULL){
        printf("CreateTimerEventInfo: CreateEvent: NG¥n");
        return FALSE;
    }

    /*
     * イベントスレッドを生成
     */
    info->hThread = CreateThread(
                    (LPSECURITY_ATTRIBUTES)NULL,
                    0,
                    (LPTHREAD_START_ROUTINE)TimerEventProc,
                    (LPVOID)info,
                    CREATE_SUSPENDED,
                    &thrd_id
                );
    if(info->hThread == NULL){
        CloseHandle(info->hEvent);
        printf("CreateTimerEventInfo: CreateThread: NG¥n");
        return FALSE;
    }

    /*
     * ドライバオブジェクトの作成
     */
    info->hTimer = CreateFile(
```

```
                        TIMERDRIVER_FILENAME,
                        GENERIC_READ | GENERIC_WRITE,
                        FILE_SHARE_READ | FILE_SHARE_WRITE,
                        NULL,
                        OPEN_EXISTING,
                        0,
                        NULL
                    );
    if(info->hTimer == INVALID_HANDLE_VALUE){
        CloseHandle(info->hThread);
        CloseHandle(info->hEvent);
        printf("CreateTimerEventInfo: CreateFile: NG¥n");
        return FALSE;
    }

    /*
     * ドライバに初期値を設定
     */
    info->Config.hEvent = info->hEvent;
    info->Config.Type = 1;
    info->Config.DueTime = 200 * (No + 1);
    ret = DeviceIoControl(
                info->hTimer,
                IOCTL_FPGATIMER_SETCONFIG,
                &info->Config,
                sizeof(FPGATIMER_CONFIG),
                NULL,
                0,
                &retlen,
                NULL
          );
    if(!ret){
        CloseHandle(info->hTimer);
        CloseHandle(info->hThread);
        CloseHandle(info->hEvent);
        return FALSE;
    }

    return TRUE;
}

//-----------------------------------------------------------------------------
void StartTimer(PTIMEREVENT_INFO info)
{
    ULONG   retlen;

    /*
     * イベントスレッドのリジューム
     */
    info->fStart = TRUE;
    ResumeThread(info->hThread);
```

```
    /*
     * タイマの開始
     */
    DeviceIoControl(
        info->hTimer,
        IOCTL_FPGATIMER_START,
        NULL,
        0,
        NULL,
        0,
        &retlen,
        NULL
    );
}

//------------------------------------------------------------------------------
void DeleteTimer(PTIMEREVENT_INFO info)
{
    ULONG   retlen;

    /*
     * イベントスレッドの Terminate
     */
    info->fStart = FALSE;
    SetEvent(info->hEvent);

    /*
     * タイマの停止
     */
    DeviceIoControl(
        info->hTimer,
        IOCTL_FPGATIMER_STOP,
        NULL,
        0,
        NULL,
        0,
        &retlen,
        NULL
    );

    while(!info->fFinish){
        Sleep(10);
    }

    /*
     * ハンドルのクローズ
     */
    CloseHandle(info->hThread);
    CloseHandle(info->hEvent);
    CloseHandle(info->hTimer);
}
```

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

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

    while(1){
        if(kbhit()){
            c = getch();
            if(c == 'q' || c == 'Q')
                break;
        }
    }

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

    return 0;
}

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

５－３　　汎用入出力

５－３－１　汎用入出力について

FP シリーズには、入力 6 点、出力 4 点の汎用入出力があります。

● 入力ポート

入力ポートのデータ形式を図 5-3-1-1 に示します。汎用入出力ドライバでは、データ型での操作とビット指定での操作を行うことができます。

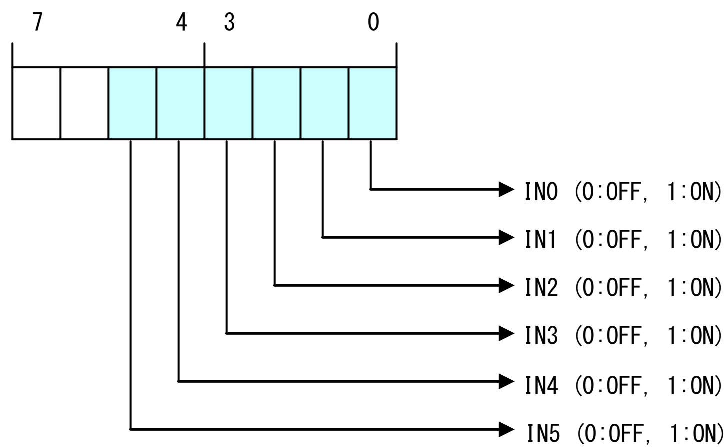

図 5-3-1-1. 入力データ

● 出力ポート

出力ポートのデータ形式を図 5-3-1-2 に示します。汎用入出力ドライバでは、データ型での操作とビット指定での操作を行うことができます。

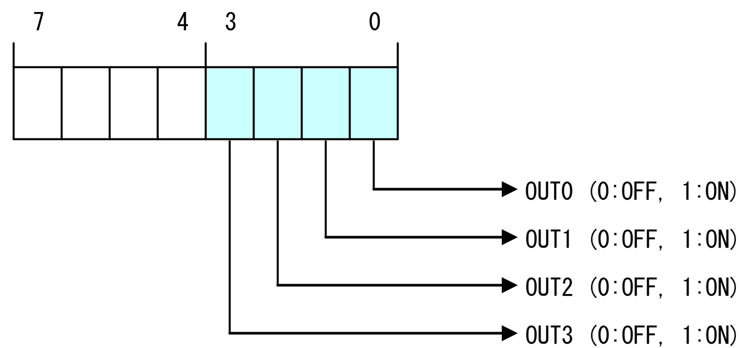

図 5-3-1-2. 出力データ

５－３－２　　汎用入出力ドライバについて

汎用入出力ドライバは汎用入出力を、ユーザーアプリケーションから利用できるようにします。ユーザーアプリケーションからは、汎用入出力ドライバを直接制御することで汎用入出力を制御することが可能です。

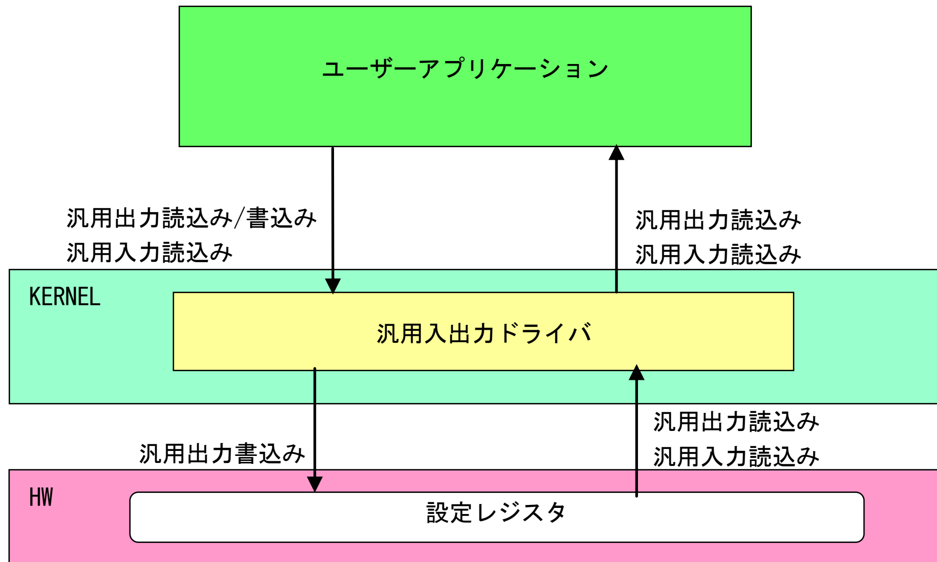

図 5-3-2-1. 汎用入出力ドライバ

５－３－３　汎用入出力デバイス

汎用入出力ドライバは汎用入出力デバイスを生成します。ユーザーアプリケーションは、デバイスファイルにアクセスすることによって汎用入出力を操作します。

汎用入出力デバイス

デバイスファイル

¥¥.¥GenIoDrv

説明

汎用入出力の制御を行うことができます。

CreateFile

デバイスファイル(¥¥.¥GenIoDrv)をオープンし、デバイスハンドルを取得します。

```
hGenIo = CreateFile(
    "¥¥¥¥.¥¥GenIoDrv",
    GENERIC_READ | GENERIC_WRITE,
    FILE_SHARE_READ | FILE_SHARE_WRITE,
    NULL,
    OPEN_EXISTING,
    0,
    NULL
);
```

CloseHandle

デバイスハンドルをクローズします。

```
CloseHandle(hGenIo);
```

ReadFile

使用しません。

WriteFile

使用しません。

DeviceIoControl

● **IOCTL_GENIODRV_RW**

汎用入出力のリードライトを行います。

５－３－４　DeviceIoControl リファレンス

IOCTL_GENIODRV_RW

機能

汎用入出力のリードライトを行います。

パラメータ

lpInBuf	:	GENIODRV_RW_PAR を格納するためのポインタを指定します。
NInBufSize	:	GENIODRV_RW_PAR のサイズを指定します。
LpOutBuf	:	GENIODRV_RW_PAR を格納するためのポインタを指定します。
NOutBufSize	:	GENIODRV_RW_PAR のサイズを指定します。
LpBytesReturned	:	実際の出力バイト数を受け取る変数へのポインタを指定します。
LpOverlapped	:	NULL を指定します。

GENIODRV_RW_PAR

```
typedef struct {
        ULONG RW;
        ULONG IoType;
        ULONG IoBit;
        ULONG Data;
} GENIODRV_RW_PAR, *P_GENIODRV_RW_PAR;
```

RW	: リードライト ［0: リード, 1: ライト, 2: リードビット, 3: ライトビット］
IoType	: 入出力ポート ［0: 入力ポート, 1: 出力ポート］
IoBit	: ビット番号(※1) ［0～5］
Data	: 入出力データ

(※1)　RW が 2:リードビット、3:ライトビットの時のみ有効です。

戻り値

処理が成功すると TRUE を返します。失敗の場合は FALSE を返します。

説明

汎用入出力の制御を行います。
リードする場合は RW に「0:リード」か「2:リードビット」、ポート、ビット番号(リードビットの時のみ)を設定の上、lpInBuf と lpOutBuf に GENIODRV_RW_PAR 構造体を渡します。正常にリードできた場合は、入出力データに読み込んだポートの値が格納されます。

● データ型でのリード (RW = 0)

IoType：　出力ポート・入力ポートを指定します。
IoBit：　無視されます。
Data：　読込んだ値がデータ形式で格納されます。

● ビット指定でのリード (RW = 2)
IoType： 出力ポート・入力ポートを指定します。
IoBit： 読込むビットを指定します。
Data： 読込んだビット状態（0、1）が格納されます。

ライトする場合は RW に「1:ライト」か「3:ライトビット」、ポート、ビット番号(ライトビットの時のみ)、ライトデータを入出力データに設定の上、lpInBuf と lpOutBuf に GENIODRV_RW_PAR 構造体を渡します。

● データ型でのライト (RW = 1)
IoType： 出力ポートを指定します。
IoBit： 無視されます。
Data： 書込む値をデータ形式で格納します。

● ビット指定でのライト (RW = 3)
IoType： 出力ポートを指定します。
IoBit： 書込むビットを指定します。
Data： 書込むビット状態（0、1）を格納します。

５－３－５　サンプルコード

「¥SDK¥Algo¥Sample¥Sample_GenIO¥GenIo」に汎用入出力を使用したサンプルコードを用意しています。リスト 5-3-5-1 にサンプルコードを示します。

リスト 5-3-5-1. 汎用入出力

```
/**
    汎用入出力制御サンプルソース
**/

#include <windows.h>
#include <winioctl.h>
#include <stdio.h>
#include <stdlib.h>
#include <mmsystem.h>
#include <conio.h>

#include "..¥Common¥GenIoDD.h"

#define DRIVER_FILENAME "¥¥¥¥.¥¥GenIoDrv"

BOOL ReadIn(HANDLE hDevice, USHORT *pBuffer)
{
    BOOL    ret;
    ULONG   retlen;

    GENIODRV_RW_PAR rw_par;

    rw_par.RW = RW_READ;
    rw_par.IoType = PORT_INP;
    rw_par.IoBit = 0;

    ret = DeviceIoControl(hDevice,
                          IOCTL_GENIODRV_RW,
                          &rw_par,
                          sizeof(GENIODRV_RW_PAR),
                          &rw_par,
                          sizeof(GENIODRV_RW_PAR),
                          &retlen,
                          NULL);
    if(!ret){
        return FALSE;
    }
    if(retlen != sizeof(GENIODRV_RW_PAR)){
        return FALSE;
    }
    *pBuffer = (USHORT)rw_par.Data;
    return TRUE;
}
BOOL WriteOut(HANDLE hDevice, USHORT Data)
{
    BOOL    ret;
```

```
    ULONG   retlen;

    GENIODRV_RW_PAR rw_par;

    rw_par.RW = RW_WRITE;
    rw_par.IoType = PORT_OUT;
    rw_par.IoBit = 0;
    rw_par.Data = (ULONG)Data;

    ret = DeviceIoControl(hDevice,
                          IOCTL_GENIODRV_RW,
                          &rw_par,
                          sizeof(GENIODRV_RW_PAR),
                          &rw_par,
                          sizeof(GENIODRV_RW_PAR),
                          &retlen,
                          NULL);
    if(!ret){
        return FALSE;
    }
    if(retlen != sizeof(GENIODRV_RW_PAR)){
        return FALSE;
    }
    return TRUE;
}

BOOL ReadInBit(HANDLE hDevice, ULONG Bit, USHORT *pBuffer)
{
    BOOL    ret;
    ULONG   retlen;

    GENIODRV_RW_PAR rw_par;

    rw_par.RW = RW_READBIT;
    rw_par.IoType = PORT_INP;
    rw_par.IoBit = Bit;

    ret = DeviceIoControl(hDevice,
                          IOCTL_GENIODRV_RW,
                          &rw_par,
                          sizeof(GENIODRV_RW_PAR),
                          &rw_par,
                          sizeof(GENIODRV_RW_PAR),
                          &retlen,
                          NULL);
    if(!ret){
        return FALSE;
    }
    if(retlen != sizeof(GENIODRV_RW_PAR)){
        return FALSE;
    }
    *pBuffer = (USHORT)rw_par.Data;
```

```
    return TRUE;
}

BOOL WriteOutBit(HANDLE hDevice, ULONG Bit, USHORT Data)
{
    BOOL    ret;
    ULONG   retlen;

    GENIODRV_RW_PAR rw_par;

    rw_par.RW = RW_WRITEBIT;
    rw_par.IoType = PORT_OUT;
    rw_par.IoBit = Bit;
    rw_par.Data = (ULONG)Data;

    ret = DeviceIoControl(hDevice,
                          IOCTL_GENIODRV_RW,
                          &rw_par,
                          sizeof(GENIODRV_RW_PAR),
                          &rw_par,
                          sizeof(GENIODRV_RW_PAR),
                          &retlen,
                          NULL);
    if(!ret){
        return FALSE;
    }
    if(retlen != sizeof(GENIODRV_RW_PAR)){
        return FALSE;
    }
    return TRUE;
}

int main(int argc, char **argv)
{
    HANDLE  hGenIo;
    BOOL    ret;
    ULONG   i;
    ULONG   retlen;
    ULONG   temp;
    USHORT  outdata=0x0001;
    USHORT  indata=0x0000;

    /* 汎用出力ﾃﾞﾊﾞｲｽのｵｰﾌﾟﾝ */
    hGenIo = CreateFile(
                    DRIVER_FILENAME,
                    GENERIC_READ | GENERIC_WRITE,
                    FILE_SHARE_READ | FILE_SHARE_WRITE,
                    NULL,
                    OPEN_EXISTING,
                    0,
                    NULL
               );
```

```
    if(hGenIo == INVALID_HANDLE_VALUE){
        printf("CreateFile: NG¥n");
        return -1;
    }

    /* 汎用出力
        汎用出力の 4 点を 1 点ずつ出力を行います。
    */
    for (i=0; i<4; i++){
        ret = WriteOut(hGenIo, outdata);
        if ( !ret ) {
            printf("DeviceIoControl: IOCTL_GENIODRV_RW NG¥n");
            CloseHandle(hGenIo);
            return -1;
        }
        outdata <<= 1;
        Sleep(500);
    }
    outdata = 0x0000;
    ret = WriteOut(hGenIo, outdata);
    if ( !ret ) {
        printf("DeviceIoControl: IOCTL_GENIODRV_RW NG¥n");
        CloseHandle(hGenIo);
        return -1;
    }
    for (i=0; i<4; i++){
        outdata = 0x0001;
        WriteOutBit(hGenIo, i, outdata);
        Sleep(500);
    }
    outdata = 0x0000;
    ret = WriteOut(hGenIo, outdata);
    if ( !ret ) {
        printf("DeviceIoControl: IOCTL_GENIODRV_RW NG¥n");
        CloseHandle(hGenIo);
        return -1;
    }

    /* 汎用入力
        汎用入力の 6 点ずつ出力を行います。
    */
    for(i=0; i<6; i++){
        ret = ReadInBit(hGenIo, i, &indata);
        if ( !ret ) {
            printf("DeviceIoControl: IOCTL_GENIODRV_RW NG¥n");
            CloseHandle(hGenIo);
            return -1;
        }
        else{
            /* IN 状態 */
            if (indata & 1) printf("INBIT%d: ON¥n", i);
            else            printf("INBIT%d: OFF¥n", i);
```

```
        }
        Sleep(500);
    }

    ret = ReadIn(hGenIo, &indata);
    if ( !ret ) {
        printf("DeviceIoControl: IOCTL_GENIODRV_RW NG¥n");
        CloseHandle(hGenIo);
        return -1;
    }
    else{
        indata &= 0x3F;
        /* IN0 状態 */
        if (indata & 0x01) printf("IN0: ON¥n");
        else               printf("IN0: OFF¥n");
        /* IN1 状態 */
        if (indata & 0x02) printf("IN1: ON¥n");
        else               printf("IN1: OFF¥n");
        /* IN2 状態 */
        if (indata & 0x04) printf("IN2: ON¥n");
        else               printf("IN2: OFF¥n");
        /* IN3 状態 */
        if (indata & 0x08) printf("IN3: ON¥n");
        else               printf("IN3: OFF¥n");
        /* IN4 状態 */
        if (indata & 0x10) printf("IN4: ON¥n");
        else               printf("IN4: OFF¥n");
        /* IN5 状態 */
        if (indata & 0x20) printf("IN5: ON¥n");
        else               printf("IN5: OFF¥n");
    }

    /* 汎用入出力ﾃﾞﾊﾞｲｽのｸﾛｰｽﾞ */
    CloseHandle(hGenIo);

    return 0;
}
```

５－４　RAS 機能

５－４－１　RAS 機能について

FP シリーズには、ハードウェアによる IN0 リセット機能、IN1 割込み機能、IN4/5 タッチパネルインターロック機能が実装されています。

RAS-IN ドライバを操作することによって、IN0 入力時にハードウェアリセットをかけることができ、IN1 入力時に割込みを発生させることができます。また、IN4 入力時に液晶ディスプレイのタッチパネルを有効に、IN5 入力時に外部ディスプレイのタッチパネルを有効にすることができます。

５－４－２　RAS-IN ドライバについて

RAS-IN ドライバは IN0 リセット機能、IN1 割込み機能、IN4/5 タッチパネルインターロック機能を、ユーザーアプリケーションから利用できるようにします。ユーザーアプリケーションから、IN0 リセット機能と IN1 割込み機能と IN4/5 タッチパネルインターロック機能の ON/OFF 設定とイベントによる IN1 割込み通知の機能を使用することができます。

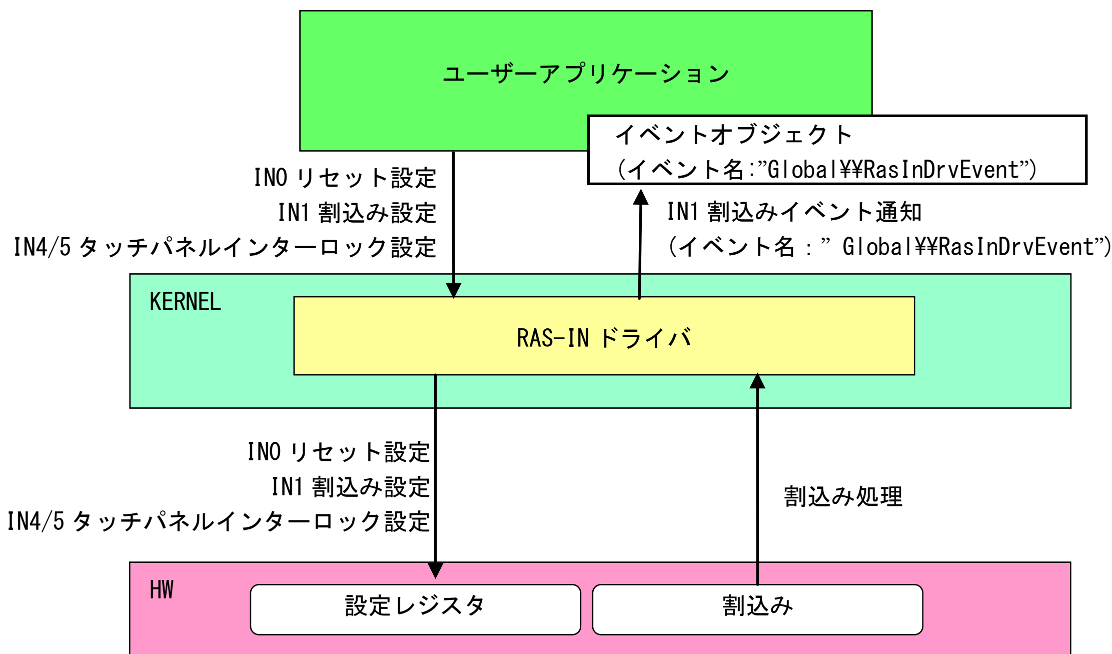

図 5-4-2-1. RAS-IN ドライバ

５－４－３　RAS-IN デバイス

RAS-IN ドライバは RAS-IN デバイスを生成します。ユーザーアプリケーションは、デバイスファイルにアクセスすることによって RAS-IN 機能を操作します。

RAS-INデバイス

デバイスファイル

¥¥.¥RasInDrv

説明

IN0リセット、IN1割り込み、IN4/5タッチパネルインターロックの設定を行うことができます。
デバイスの使用は１アプリケーションのみです。
複数のアプリケーションから使用することはできません。

CreateFile

デバイスファイル(¥¥.¥RasInDrv)をオープンし、デバイスハンドルを取得します。

```
hRasIn = CreateFile(
    "¥¥¥¥.¥¥RasInDrv",
    GENERIC_READ | GENERIC_WRITE,
    FILE_SHARE_READ | FILE_SHARE_WRITE,
    NULL,
    OPEN_EXISTING,
    0,
    NULL
);
```

CloseHandle

デバイスハンドルをクローズします。

```
CloseHandle(hRasIn);
```

ReadFile

使用しません。

WriteFile

使用しません。

DeviceIoControl

- **IOCTL_RASINDRV_SINORST**
  IN0リセットを設定します。
- **IOCTL_RASINDRV_GINORST**
  現在のIN0リセット設定を取得します。
- **IOCTL_RASINDRV_SIN1INT**
  IN1割り込みを設定します。
- **IOCTL_RASINDRV_GIN1INT**
  現在のIN1割り込み設定を取得します。
- **IOCTL_RASINDRV_SIN4_5COMLOCK**
  IN4/5タッチパネルインターロックを設定します。
- **IOCTL_RASINDRV_GIN4_5COMLOCK**
  現在のIN4/5タッチパネルインターロックを取得します。

５－４－４　IN1 割込みの使用手順

基本的な使用手順を以下に示します。

IN1 割込み通知用イベントハンドルを作成後､IN1 割込み通知用イベントハンドルでのイベント待ち準備が整ったところで、RAS-IN デバイスに IN1 割込み有効を設定します。

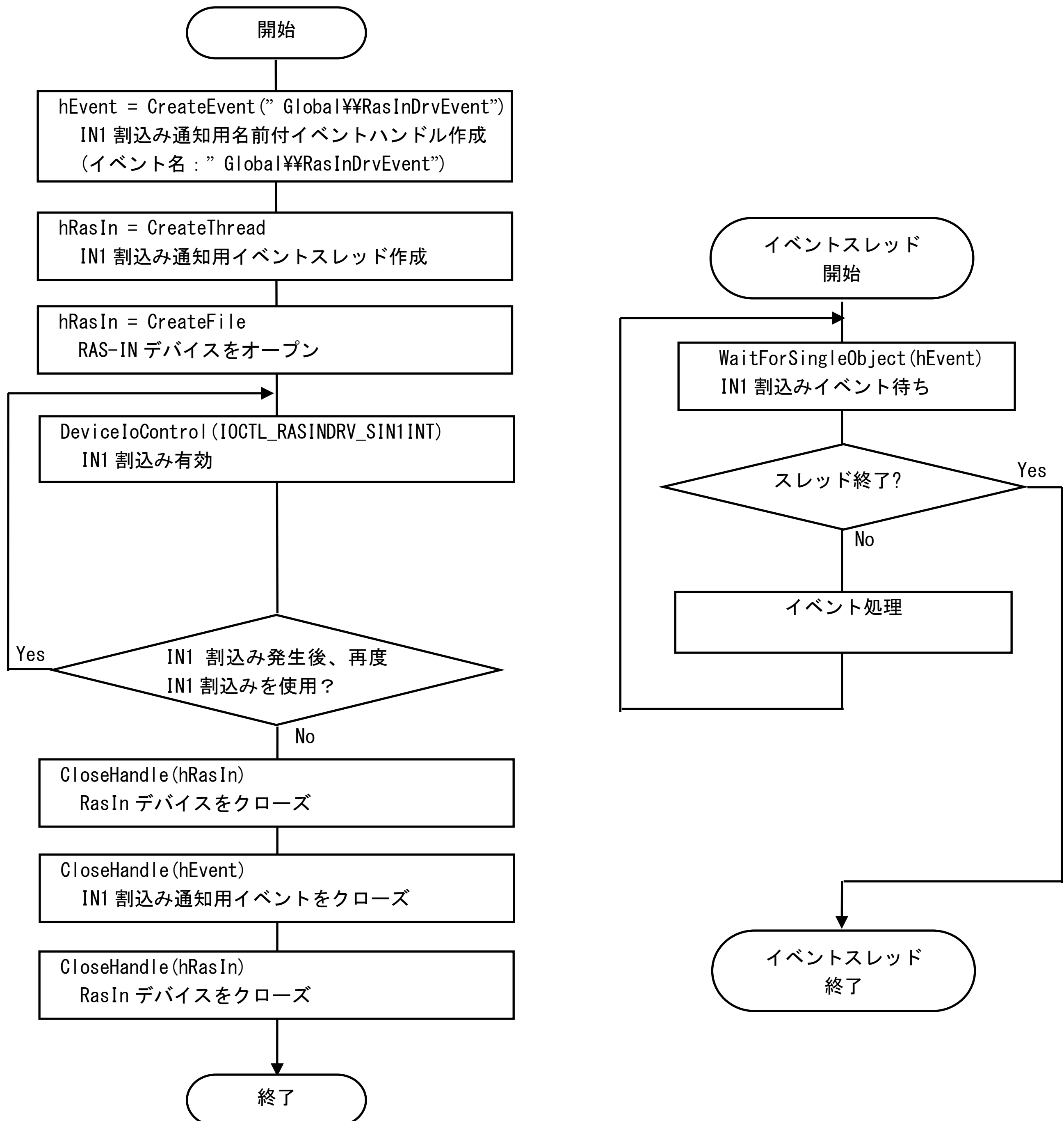

図 5-4-4-1. IN1 割込み仕様手順

５－４－５　複数アプリケーションで IN1 割込み発生時のイベントを同時に使用する場合

複数アプリケーションで IN1 割込み発生時に同時にイベント処理する場合の使用手順を以下に示します。

メインアプリケーションで IN1 割込み通知用名前付き手動リセットのイベントハンドルを作成後、IN1 割込み通知用イベントハンドルでのイベント待ち準備が整ったところで､RAS-IN デバイスに IN1 割込み有効を設定します。サブアプリケーションは、手動リセットの IN1 割込み通知用名前付きイベントハンドルを作成後、IN1 割込み通知用イベントハンドルでのイベント待ち準備を行います。IN1 割込みが発生すれば、メイン、サブの両アプリケーションに同時にイベントが発生します。

※　IN1 割込み有効/無効は RAS-IN デバイスをオープンしたアプリケーションしか行えません。複数のアプリケーションからは IN1 割込み有効/無効できませんので注意してください。

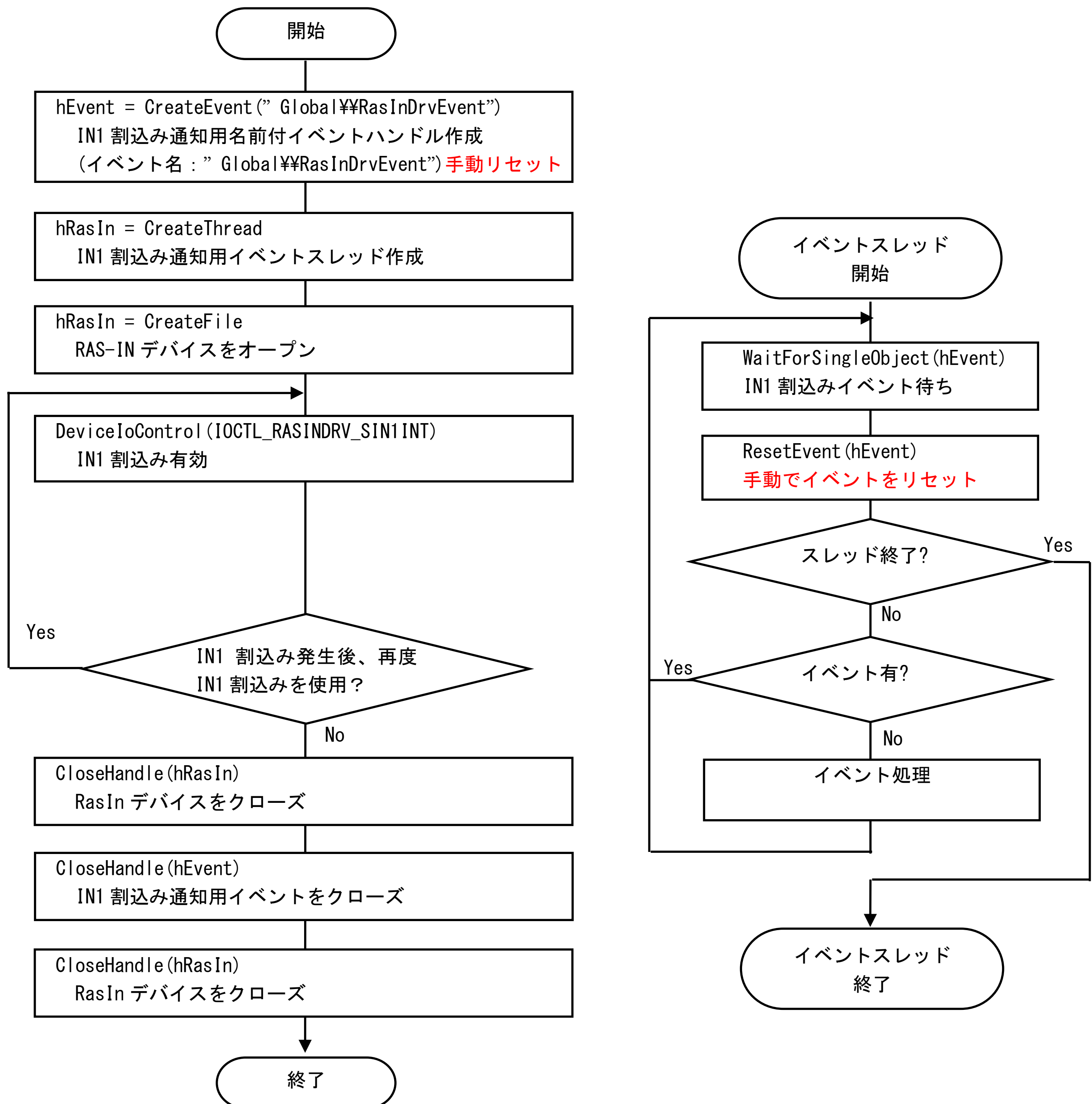

図 5-4-5-1. 複数アプリケーションで IN1 割込みを使用する手順

５－４－６　DeviceIoControl リファレンス

IOCTL_RASINDRV_SINORST

機能

INO リセットを設定します。

パラメータ

lpInBuf	:	INO リセット情報を格納するポインタを指定します。
NInBufSize	:	INO リセット情報を格納するポインタのサイズを指定します。
LpOutBuf	:	NULL を指定します。
NOutBufSize	:	0 を指定します。
LpBytesReturned	:	実際の出力バイト数を受け取る変数へのポインタ。
LpOverlapped	:	NULL を指定します。

INO リセット情報

データタイプ	:	ULONG
データサイズ	:	4 バイト
内容	:	1: INO リセット有効, 0: INO リセット無効

戻り値

処理が成功すると TRUE を返します。失敗の場合は FALSE を返します。

説明

INO リセットを設定します。
INO リセットを有効にする場合は、INO リセットの設定を格納するポインタに 1、無効にする場合は 0 を設定の上、DeviceIoControl を実行してください。

IOCTL_RASINDRV_GINORST

機能

INO リセット設定を取得します。

パラメータ

lpInBuf	:	NULL を指定します。
NInBufSize	:	0 を指定します。
LpOutBuf	:	INO リセット情報を格納するポインタを指定します。
NOutBufSize	:	INO リセット情報を格納するポインタのサイズを指定します。
LpBytesReturned	:	実際の出力バイト数を受け取る変数へのポインタ。
LpOverlapped	:	NULL を指定します。

INO リセット情報

データタイプ	:	ULONG
データサイズ	:	4 バイト
内容	:	1: INO リセット有効, 0: INO リセット無効

戻り値

処理が成功すると TRUE を返します。失敗の場合は FALSE を返します。

説明

現在の INO リセット設定値を取得します。

IOCTL_RASINDRV_SIN1INT

機能

IN1 割込みを設定します。

パラメータ

lpInBuf	:	IN1 割込み情報を格納するポインタを指定します。
NInBufSize	:	IN1 割込み情報を格納するポインタのサイズを指定します。
LpOutBuf	:	NULL を指定します。
NOutBufSize	:	0 を指定します。
LpBytesReturned	:	実際の出力バイト数を受け取る変数へのポインタを指定します。
LpOverlapped	:	NULL を指定します。

IN1 割込み情報

データタイプ	:	ULONG
データサイズ	:	4 バイト
内容	:	1: IN1 割込み有効, 0: IN1 割込み無効

戻り値

処理が成功すると TRUE を返します。失敗の場合は FALSE を返します。

説明

IN1 割込みの設定を行います。
IN1 割込みを有効にする場合は、IN1 割込みの設定を格納するポインタに 1、無効にする場合は 0 を設定の上、DeviceIoControl を実行してください。

IOCTL_RASINDRV_GIN1INT

機能

IN1 割込み設定を取得します。

パラメータ

lpInBuf	:	NULL を指定します。
NInBufSize	:	0 を指定します。
LpOutBuf	:	IN1 割込み情報を格納するためのポインタを指定します。
NOutBufSize	:	IN1 割込み情報のサイズを指定します。
LpBytesReturned	:	実際の出力バイト数を受け取る変数へのポインタを指定します。
LpOverlapped	:	NULL を指定します。

IN1 割込み情報

データタイプ	:	ULONG
データサイズ	:	4 バイト
内容	:	1: IN1 割込み有効, 0: IN1 割込み無効

戻り値

処理が成功すると TRUE を返します。失敗の場合は FALSE を返します。

説明

現在の IN1 割込み設定値を取得します。

IOCTL_RASINDRV_SIN4_5COMLOCK

機能

IN4/5 タッチパネルインターロックを設定します。

パラメータ

lpInBuf	:	IN4/5 タッチパネルインターロック情報を格納するポインタを指定します。
NInBufSize	:	IN4/5 タッチパネルインターロック情報を格納するポインタのサイズを指定します。
LpOutBuf	:	NULL を指定します。
NOutBufSize	:	0 を指定します。
LpBytesReturned	:	実際の出力バイト数を受け取る変数へのポインタ。
LpOverlapped	:	NULL を指定します。

IN4/5 タッチパネルインターロック情報

データタイプ	:	ULONG
データサイズ	:	4 バイト
内容	:	1: IN4/5 タッチパネルインターロック有効 0: IN4/5 タッチパネルインターロック無効

戻り値

処理が成功すると TRUE を返します。失敗の場合は FALSE を返します。

説明

IN4/5 タッチパネルインターロックを設定します。
IN4/5 タッチパネルインターロックを有効にする場合は、IN4/5 タッチパネルインターロックの設定を格納するポインタに 1、無効にする場合は 0 を設定の上、DeviceIoControl を実行してください。

IOCTL_RASINDRV_GIN4_5COMLOCK

機能

IN4/5 タッチパネルインターロック設定を取得します。

パラメータ

lpInBuf　:　NULL を指定します。
NInBufSize　:　0 を指定します。
LpOutBuf　:　IN4/5 タッチパネルインターロック情報を格納するポインタを指定します。
NOutBufSize　:　IN4/5 タッチパネルインターロック情報を格納するポインタのサイズを指定します。
LpBytesReturned　:　実際の出力バイト数を受け取る変数へのポインタ。
LpOverlapped　:　NULL を指定します。

IN4/5 タッチパネルインターロック情報

データタイプ	:	ULONG
データサイズ	:	4 バイト
内容	:	1: IN4/5 タッチパネルインターロック有効 0: IN4/5 タッチパネルインターロック無効

戻り値

処理が成功すると TRUE を返します。失敗の場合は FALSE を返します。

説明

現在の IN4/5 タッチパネルインターロック設定値を取得します。

５－４－７　サンプルコード

●IN0 リセット

「¥SDK¥Algo¥Sample¥Sample_Reset¥In0Reset」に IN0 リセット機能のサンプルコードを用意しています。リスト 5-4-7-1 にサンプルコードを示します。

リスト 5-4-7-1. IN0 リセット

```
/**
    汎用入力 IN0 リセット制御方法サンプルソース
**/
#include <windows.h>
#include <winioctl.h>
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>

#include "..¥Common¥RasinDrvDD.h"

#define RASINDRIVER_FILENAME    "¥¥¥¥.¥¥RasInDrv"

int main(void)
{
    HANDLE hRasin;
    ULONG retlen;
    ULONG reset;
    ULONG errno;
    BOOL ret;

    /*
     * ドライバオブジェクトの作成
     */
    hRasin = CreateFile(
                    RASINDRIVER_FILENAME,
                    GENERIC_READ | GENERIC_WRITE,
                    FILE_SHARE_READ | FILE_SHARE_WRITE,
                    NULL,
                    OPEN_EXISTING,
                    0,
                    NULL
                );
    if(hRasin == INVALID_HANDLE_VALUE){
        printf("CreateFile: NG¥n");
        return -1;
    }

    /*
     * IN0 リセットを有効にする
     *
     * reset: 1    有効
     *      : 0    無効
     */
```

```
    reset = 1;
    ret = DeviceIoControl(
                hRasin,
                IOCTL_RASINDRV_SINORST,
                &reset,
                sizeof(ULONG),
                NULL,
                0,
                &retlen,
                NULL
                );
    if(!ret){
        errno = GetLastError();
        fprintf(stderr, "ioctl set INORST error: %d¥n", errno);
        CloseHandle(hRasin);
        return -1;
    }
    else {
        fprintf(stdout, "ioctl set INORST success: %d¥n", reset);
    }

    /*
     * INO リセットを確認する
     */
    ret = DeviceIoControl(
                hRasin,
                IOCTL_RASINDRV_GINORST,
                NULL,
                0,
                &reset,
                sizeof(ULONG),
                &retlen,
                NULL
                );
    if(!ret){
        errno = GetLastError();
        fprintf(stderr, "ioctl get INORST error: %d¥n", errno);
        CloseHandle(hRasin);
        return -1;
    }
    else {
        fprintf(stdout, "ioctl get INORST success: %d¥n", reset);
    }

    CloseHandle(hRasin);
    return 0;
}
```

●IN1 割込み

「¥SDK¥Algo¥Sample¥Sample_Interrupt¥In1Interrupt」に IN1 割込み機能を使用したサンプルコードを用意しています。リスト 5-4-7-2 にサンプルコードを示します。

リスト 5-4-7-2. IN1 割込み

```
/**
    汎用入力 IN1 割込み制御サンプルソース
**/
#include <windows.h>
#include <winioctl.h>
#include <stdio.h>
#include <stdlib.h>
#include <mmsystem.h>
#include <conio.h>

#include "..¥Common¥RasinDrvDD.h"

#define RASINDRIVER_FILENAME    "¥¥¥¥.¥¥RasInDrv"

//---------------------------------------------------------------------------
typedef struct {
    int No;
    HANDLE  hEvent;
    HANDLE  hThread;
    HANDLE  hRasin;
    volatile BOOL fIn1Int;
    volatile BOOL fStart;
    volatile BOOL fFinish;
} RASINEVENT_INFO, *PRASINEVENT_INFO;

//---------------------------------------------------------------------------
/*
 * 割込みハンドラ
 */
DWORD WINAPI In1IntEventProc(void *pData)
{
    PRASINEVENT_INFO    info = (PRASINEVENT_INFO)pData;
    DWORD ret;

    printf("In1IntEventProc: Start¥n");

    info->fFinish = FALSE;
    while(1){
        if(WaitForSingleObject(info->hEvent, INFINITE) != WAIT_OBJECT_0){
            break;
        }
        ResetEvent(info->hEvent);       // イベントオブジェクト生成時に
                                        // 手動リセットを設定した場合は
                                        // 手動でイベントオブジェクトを
                                        // 非シグナル状態にする必要があります。
        if(!info->fStart){
            break;
```

```
        }
        printf("In1IntEventProc: Interrupt¥n");
    }
    info->fFinish = TRUE;

    printf("In1IntEventProc: Finish¥n");
    return 0;
}

//------------------------------------------------------------------------------
BOOL CreateIn1IntEventInfo(PRASINEVENT_INFO info)
{
    DWORD   thrd_id;
    ULONG   retlen;
    BOOL    ret;

    info->hEvent = NULL;
    info->hThread = NULL;
    info->hRasin = INVALID_HANDLE_VALUE;

    info->fStart = FALSE;
    info->fFinish = FALSE;
    info->fIn1Int = FALSE;

    /* イベントオブジェクトの作成
     *   複数アプリケーションでイベントを共有する場合は、
     *   CreateFile でドライバオブジェクトを作成するより前に
     *   CreateEvent で手動リセットを有効にした名前付き
     *   イベントを作成する必要があります。
     *   単アプリケーションの場合、自動リセットで問題ありません。
     */
    info->hEvent = CreateEvent(
                    NULL,
                    TRUE,                   // 手動リセットを指定します。
                    FALSE,
                    RASINDRV_EVENT_NAME     // イベント名を指定します。
                    );
    if(info->hEvent == NULL){
        printf("CreateIn1IntEventInfo: CreateEvent: NG¥n");
        return FALSE;
    }

    /*
     * イベントスレッドを生成
     */
    info->hThread = CreateThread(
                    (LPSECURITY_ATTRIBUTES)NULL,
                    0,
                    (LPTHREAD_START_ROUTINE)In1IntEventProc,
                    (LPVOID)info,
                    CREATE_SUSPENDED,
                    &thrd_id
```

```
                    );
    if(info->hThread == NULL){
        CloseHandle(info->hEvent);
        printf("CreateIn1IntEventInfo: CreateThread: NG¥n");
        return FALSE;
    }

    /*
     * ドライバオブジェクトの作成
     */
    info->hRasin = CreateFile(
                    RASINDRIVER_FILENAME,
                    GENERIC_READ | GENERIC_WRITE,
                    FILE_SHARE_READ | FILE_SHARE_WRITE,
                    NULL,
                    OPEN_EXISTING,
                    0,
                    NULL
                    );
    if(info->hRasin == INVALID_HANDLE_VALUE){
        CloseHandle(info->hThread);
        CloseHandle(info->hEvent);
        printf("CreateIn1IntEventInfo: CreateFile: NG¥n");
        return FALSE;
    }

    return TRUE;
}

//-----------------------------------------------------------------------------
void DeleteIn1IntEvent(PRASINEVENT_INFO info)
{
    ULONG   retlen;

    // Stop Thread
    info->fStart = FALSE;
    SetEvent(info->hEvent);

    // Wait Thread Stop, Close Thread
    while(!info->fFinish){
        Sleep(10);
    }

    // Close Handle
    CloseHandle(info->hThread);
    CloseHandle(info->hEvent);
    CloseHandle(info->hRasin);
}

//-----------------------------------------------------------------------------
/*
 * IN1 Interrupt の取得
```

```
 */
BOOL Get_IN1Int(HANDLE hdriver, ULONG *value)
{
    BOOL    ret;
    ULONG   retlen;

    // Get IN1 Interrupt
    ret = DeviceIoControl(
                hdriver,
                IOCTL_RASINDRV_GIN1INT,
                NULL,
                0,
                &value,
                sizeof(ULONG),
                &retlen,
                NULL
                );

    return ret;
}

//----------------------------------------------------------------------------
/*
 * IN1 Interrupt の設定
 */
BOOL Set_IN1Int(HANDLE hdriver, ULONG value)
{
    BOOL    ret;
    ULONG   retlen;

    // Set IN1 Interrupt
    ret = DeviceIoControl(
                hdriver,
                IOCTL_RASINDRV_SIN1INT,
                &value,
                sizeof(ULONG),
                NULL,
                0,
                &retlen,
                NULL
                );

    return ret;
}
//----------------------------------------------------------------------------

int main(void)
{
    int     c;
    BOOL    ret;
    DWORD   errno;
    DWORD   onoff;
```

```
RASINEVENT_INFO info;

/*
 * イベントオブジェクト、
 * イベントスレッド、
 * ドライバオブジェクトの作成
 */
if( !CreateIn1IntEventInfo(&info) ){
    printf("CreateIn1IntEvent: NG¥n");
    return -1;
}

/*
 * 現在の設定を取得
 *
 * onoff: 1    有効
 *      : 0    無効
 */
ret = Get_IN1Int(info.hRasin, &onoff);
if( !ret ){
    errno = GetLastError();
    fprintf(stderr, "ioctl get IN1INT error: %d¥n", errno);
    DeleteIn1IntEvent(&info);
    return -1;
}
else {
    fprintf(stdout, "ioctl get IN1INT success: %d¥n", onoff);
}

/*
 * IN1 割込みを有効にする
 *  有効の場合向こうに変更し終了
 * onoff: 1    有効
 *      : 0    無効
 */
if (onoff != 1)
    onoff = 1;
else
    onoff = 0;

ret = Set_IN1Int(info.hRasin, onoff);
if( !ret ){
    errno = GetLastError();
    fprintf(stderr, "ioctl set IN1INT error: %d¥n", errno);
    DeleteIn1IntEvent(&info);
    return -1;
}
else {
    fprintf(stdout, "ioctl set IN1INT success: %d¥n", onoff);
    // Resume Thread
    info.fStart = TRUE;
    ResumeThread(info.hThread);
```

```
        if (onoff == 0){
            fprintf(stdout, "IN1 Interrupt off¥n");
            return 1;
        }
    }

    while(1){
        if( kbhit() ){
            c = getch();
            if(c == 'q' || c == 'Q')
                break;
        }
    }

    /*
     * イベントオブジェクト、
     * イベントスレッド、
     * ドライバオブジェクトの破棄
     */
    DeleteIn1IntEvent(&info);

    return 0;
}

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

●IN4/5 タッチパネルインターロック

「¥SDK¥Algo¥Sample¥Sample_Interlock¥SerialInterlock」に IN4/5 タッチパネルインターロック機能のサンプルコードを用意しています。リスト 5-4-7-3 にサンプルコードを示します。

リスト 5-4-7-3. IN4/5 タッチパネルインターロック

```
/**
    タッチパネル　インターロック制御方法サンプルソース
**/
#include <windows.h>
#include <winioctl.h>
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>

#include "..¥Common¥RasinDrvDD.h"

#define RASINDRIVER_FILENAME    "¥¥¥¥.¥¥RasInDrv"

int main(void)
{
    HANDLE hRasin;
    ULONG retlen;
    ULONG interlock;
    ULONG error;
```

```
BOOL ret;

/*
 * ドライバオブジェクトの作成
 */
hRasin = CreateFile(
                RASINDRIVER_FILENAME,
                GENERIC_READ | GENERIC_WRITE,
                FILE_SHARE_READ | FILE_SHARE_WRITE,
                NULL,
                OPEN_EXISTING,
                0,
                NULL
            );
if(hRasin == INVALID_HANDLE_VALUE){
    printf("CreateFile: NG¥n");
    return -1;
}

/*
 * タッチパネルインターロックを有効にする
 *
 * interlock: 1    有効
 *          : 0    無効
 */
interlock = 1;
ret = DeviceIoControl(
            hRasin,
            IOCTL_RASINDRV_SIN4_5COMLOCK,
            &interlock,
            sizeof(ULONG),
            NULL,
            0,
            &retlen,
            NULL
            );
if(!ret){
    error = GetLastError();
    fprintf(stderr, "ioctl set IN4_5COMLOCK error: %d¥n", error);
    CloseHandle(hRasin);
    return -1;
}
else {
    fprintf(stdout, "ioctl set IN4_5COMLOCK success: %d¥n", interlock);
}

/*
 * タッチパネルインターロックを確認する
 */
ret = DeviceIoControl(
            hRasin,
            IOCTL_RASINDRV_GIN4_5COMLOCK,
```

```
                NULL,
                0,
                &interlock,
                sizeof(ULONG),
                &retlen,
                NULL
                );
    if(!ret){
        error = GetLastError();
        fprintf(stderr, "ioctl get IN4_5COMLOCK error: %d¥n", error);
        CloseHandle(hRasin);
        return -1;
    }
    else {
        fprintf(stdout, "ioctl get IN4_5COMLOCK success: %d¥n", interlock);
    }

    CloseHandle(hRasin);
    return 0;
}
```

５－５　シリアルコントロール機能

５－５－１　シリアルコントロール機能について

FP シリーズには、RS-232C 以外に RS-422、RS-485 通信を行う事ができるシリアルポートが 2 ポート(COM1、COM2)実装されています。COM1 と COM2 は、シリアルコントロールドライバを操作することによって、指定したポートを RS-232C/RS-422/RS-485 に切替えることができます。

※　シリアルポートタイプを切替える場合は、シリアルコントロール機能の設定に合わせて、シリアルポート設定スイッチを設定してください。

５－５－２　シリアルコントロールドライバについて

シリアルコントロール(以下、SciCtl と称します)ドライバはシリアルコントロール機能を、ユーザーアプリケーションから利用できるようにします。ユーザーアプリケーションから、ポートの設定と送信イネーブルタイムアウト時間を設定することができます。

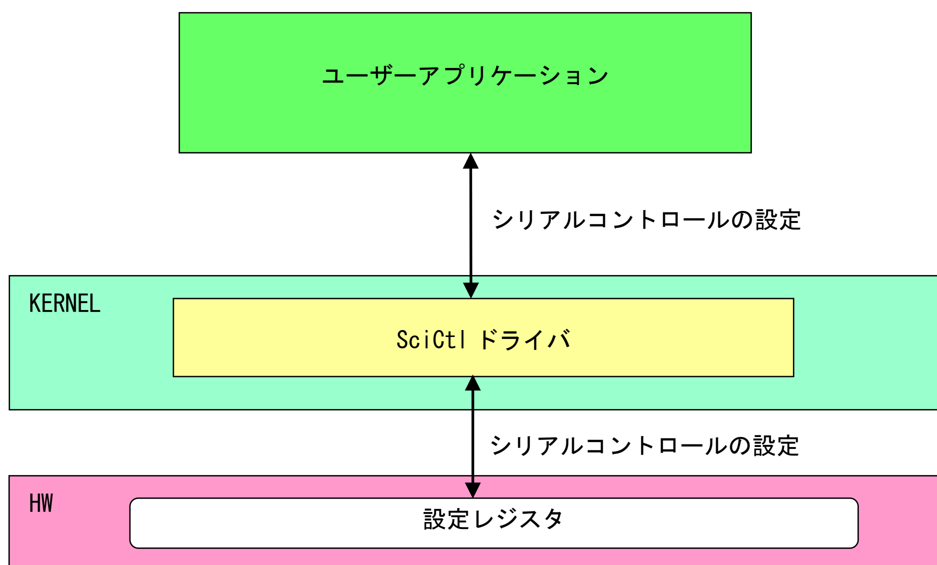

図 5-5-2-1. SciCtl ドライバ

５－５－３　SciCtl デバイス

SciCtl ドライバは SciCtl デバイスを生成します。ユーザーアプリケーションは、デバイスファイルにアクセスすることによってシリアルコントロール機能を操作します。

SciCtlデバイス

デバイスファイル

¥¥.¥SciCtl

説明

シリアルコントロール機能の設定をすることができます。

レジストリ設定

[KEY]　HKEY_LOCAL_MACHINE¥SYSTEM¥CurrentControlSet¥Services¥SciCtl
[VALUE:DWORD]　Com1Type, Com2Type
　シリアルポートタイプを設定します。
　ドライバ起動時(OS起動時)にこの値を参照します。(デフォルト値: 0)
[VALUE:DWORD]　Com1Timer, Com2Timer
　RS-485の送信イネーブル時間を設定します。マイクロ秒単位で設定します。
　ドライバ起動時(OS起動時)にこの値を参照します。(デフォルト値: 0)

CreateFile

デバイスファイル(¥¥.¥SciCtl)をオープンし、デバイスハンドルを取得します。

```
hSciCtl = CreateFile(
    "¥¥¥¥.¥¥SciCtl",
    GENERIC_READ | GENERIC_WRITE,
    FILE_SHARE_READ | FILE_SHARE_WRITE,
    NULL,
    OPEN_EXISTING,
    0,
    NULL
);
```

CloseHandle

デバイスハンドルをクローズします。

```
CloseHandle(hSciCtl);
```

ReadFile

使用しません。

WriteFile

使用しません。

DeviceIoControl

- **IOCTL_SCICTLDRV_SETCONFIG**
  シリアルコントロールを設定します。
- **IOCTL_SCICTLDRV_GETCONFIG**
  シリアルコントロールを取得します。

５－５－４　DeviceIoControl リファレンス

IOCTL_SCICTLDRV_SETCONFIG

機能

シリアルコントロールの設定を行います。

パラメータ

lpInBuf	:	SCICTL_CONF_PAR を格納するためのポインタを指定します。
NInBufSize	:	SCICTL_CONF_PAR のサイズを指定します。
LpOutBuf	:	NULL を指定します。
NOutBufSize	:	0 を指定します。
LpBytesReturned	:	実際の出力バイト数を受け取る変数へのポインタを指定します。
LpOverlapped	:	NULL を指定します。

SCICTL_CONF_PAR

```
typedef struct {
    ULONG ch;
    ULONG type;
    ULONG timer;
} SCICTL_CONF_PAR, *P_SCICTL_CONF_PAR;
```

ch	: チャンネル [0: SIO1, 1: SIO2]
type	: シリアルポートタイプ [0: RS-232C, 1: RS-422, 2: RS-485]
timer	: RS-485 の送信イネーブル時間 [0～65535us]

戻り値

処理が成功すると TRUE を返します。失敗の場合は FALSE を返します。

説明

シリアルコントロールの設定を行います。シリアルの通信を開始する前に、このコントロールを実行してシリアルコントロールの設定を行うようにしてください。

IOCTL_SCICTLDRV_GETCONFIG

機能

シリアルコントロール設定を取得します。

パラメータ

lpInBuf	:	NULL を指定します。
NInBufSize	:	0 を指定します。
LpOutBuf	:	SCICTL_CONF_PAR を格納するためのポインタを指定します。
NOutBufSize	:	SCICTL_CONF_PAR のサイズを指定します。
LpBytesReturned	:	実際の出力バイト数を受け取る変数へのポインタを指定します。
LpOverlapped	:	NULL を指定します。

SCICTL_CONF_PAR

```
typedef struct {
    ULONG ch;
    ULONG type;
    ULONG timer;
} SCICTL_CONF_PAR, *P_SCICTL_CONF_PAR;
```

ch	:チャンネル [0: SIO1, 1: SIO2]
type	:シリアルポートタイプ [0: RS-232C, 1: RS-422, 2: RS-485]
timer	:RS-485 の送信イネーブル時間 [0～65535us]

戻り値

処理が成功すると TRUE を返します。失敗の場合は FALSE を返します。

説明

現在のシリアルコントロール設定値を取得します。

５－５－５　サンプルコード

「¥SDK¥Algo¥Sample¥Sample_SerialControl¥SerialControlConfig」にシリアルポートの RS-232C/422/485 切替えのサンプルコードを用意しています。リスト 5-5-5-1 にサンプルコードを示します。

リスト 5-5-5-1. RS-232C/422/485 切替え

```
/**
    シリアルポート RS422/RS485/RS232C 切替え制御方法サンプルソース
**/
#include <windows.h>
#include <winioctl.h>
#include <stdio.h>
#include <stdlib.h>
#include <mmsystem.h>
#include <conio.h>

#include "..¥Common¥SciCtlDD.h"

#define DRIVER_FILENAME "¥¥¥¥.¥¥SciCtlDrv"

int main(int argc, char **argv)
{
    HANDLE hSciCtl;
    ULONG retlen;
    ULONG errno;
    BOOL ret;
    int ch, type, timer;
    SCICTL_CONF_PAR conf;

    if(argc != 4){
        printf("invalid arg¥n");
        printf("TestSciCtl.exe [ch(0,1)] [type] [timer]¥n");
        return -1;
    }
    sscanf(*(argv + 1), "%u", &ch);
    sscanf(*(argv + 2), "%u", &type);
    sscanf(*(argv + 3), "%u", &timer);

    /*
     * ドライバオブジェクトの作成
     */
    hSciCtl = CreateFile(
                    DRIVER_FILENAME,
                    GENERIC_READ | GENERIC_WRITE,
                    FILE_SHARE_READ | FILE_SHARE_WRITE,
                    NULL,
                    OPEN_EXISTING,
                    0,
                    NULL
                    );
    if(hSciCtl == INVALID_HANDLE_VALUE){
        printf("CreateFile: NG¥n");
```

```
        return -1;
    }

    /*
     *  シリアルコントロール情報を書込む
     *
     *   conf.ch
     *       ポート番号  0: COM1
     *                   1: COM2
     *
     *   conf.type
     *       ポートタイプ  0: RS232C
     *                     1: RS422
     *                     2: RS485
     *
     *   conf.timer
     *       RS485 用 TX ディセーブルタイマ[マイクロ秒]
     *       送信完了から設定時間の間、再送信がなければ
     *       TX がディセーブルされます。
     */
    conf.ch = ch;
    conf.type = type;
    conf.timer = timer;

    ret = DeviceIoControl(
                hSciCtl,
                IOCTL_SCICTLDRV_SETCONFIG,
                &conf,
                sizeof(SCICTL_CONF_PAR),
                NULL,
                0,
                &retlen,
                NULL
          );
    if(!ret){
        errno = GetLastError();
        fprintf(stderr, "ioctl set IOCTL_SCICTLDRV_SETCONFIG error: %d¥n", errno);
        CloseHandle(hSciCtl);
        return -1;
    }

    /*
     * シリアルコントロール情報を読み出す
     */
    ret = DeviceIoControl(
                hSciCtl,
                IOCTL_SCICTLDRV_GETCONFIG,
                &conf,
                sizeof(SCICTL_CONF_PAR),
                &conf,
                sizeof(SCICTL_CONF_PAR),
                &retlen,
```

```
                NULL
          );
    if(!ret){
        errno = GetLastError();
        fprintf(stderr, "ioctl set IOCTL_SCICTLDRV_GETCONFIG error: %d¥n", errno);
        CloseHandle(hSciCtl);
        return -1;
    }
    printf("SciCtlConf.exe ch=%u, type=%u, timer=%u¥n", conf.ch, conf.type, conf.timer);

    CloseHandle(hSciCtl);
    return 0;
}
```

５－６　　バックアップ SRAM

５－６－１　　バックアップ SRAM について

FP シリーズには、バックアップバッテリが搭載されている SRAM が実装されています。SRAM ドライバを操作することによって、バックアップ SRAM の読み書きを行うことができます。

５－６－２　　SRAM ドライバについて

SRAM ドライバは SRAM の読み書きを、ユーザーアプリケーションから利用できるようにします。

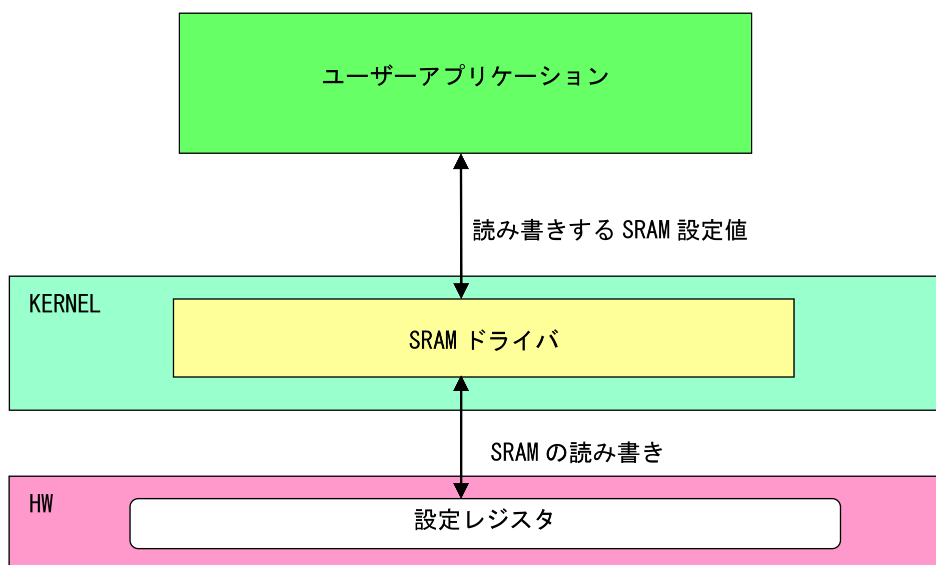

図 5-6-2-1. SRAM ドライバ

５－６－３　SRAM デバイス

SRAM ドライバは SRAM デバイスを生成します。ユーザーアプリケーションは、デバイスファイルにアクセスすることによって SRAM の読み書きを行います。

SRAMデバイス

デバイスファイル

¥¥.¥SramDrv

説明

SRAMの読み書きを行うことができます。

CreateFile

デバイスファイル(¥¥.¥SramDrv)をオープンし、デバイスハンドルを取得します。

```
hSram = CreateFile(
    ”¥¥¥¥.¥¥SramDrv”,
    GENERIC_READ | GENERIC_WRITE,
    FILE_SHARE_READ | FILE_SHARE_WRITE,
    NULL,
    OPEN_EXISTING,
    0,
    NULL
);
```

CloseHandle

デバイスハンドルをクローズします。

```
CloseHandle(hSram);
```

ReadFile

使用しません。

WriteFile

使用しません。

DeviceIoControl

- **IOCTL_SRAMDRV_RW**
  SRAMの読み書きを行います。

５－６－４　DeviceIoControl リファレンス

IOCTL_SRAMDRV_RW

機能

SRAM の読み書きを行います。

パラメータ

lpInBuf	:	SRAMDRV_RW_PAR を格納するためのポインタを指定します。
NInBufSize	:	SRAMDRV_RW_PAR のサイズを指定します。
LpOutBuf	:	SRAMDRV_RW_PAR を格納するためのポインタを指定します。
NOutBufSize	:	SRAMDRV_RW_PAR のサイズを指定します。
LpBytesReturned	:	実際の出力バイト数を受け取る変数へのポインタを指定します。
LpOverlapped	:	NULL を指定します。

SRAMDRV_RW_PAR

```
typedef struct {
    ULONG RW;
    ULONG Type;
    ULONG Addr;
    ULONG BufferPtr;
    ULONG Length;
} SRAMDRV_RW_PAR, *PSRAMDRV_RW_PAR;
```

RW	: リードライト [0: リード, 1: ライト]
Type	: タイプ [1: 8 ビット, 2: 16 ビット]
Addr	: アドレス [0x00000 ～ 0x7F000]
BufferPtr	: データのポインタアドレス
Length	: サイズ [0x00000 ～ 0x7F000]

戻り値

処理が成功すると TRUE を返します。失敗の場合は FALSE を返します。

説明

SRAM の読み書きを行います。
SRAM の容量は 512K バイトですが、4K バイトをシステムで使用しているので、ユーザーが使用できるのは[0x00000～0x7F000]の 508K バイトになります。
リードする場合は開始アドレス、サイズ、データを設定の上、lpInBuf と lpOutBuf に SRAMDRV_RW_PAR 構造体を渡します。正常にリードできた場合は渡したポインタにリードした SRAM の値が格納されています。
ライトする場合は開始アドレス、サイズ、データを設定の上、lpInBuf と lpOutBuf に SRAMDRV_RW_PAR 構造体を渡します。正常にライトできた場合は、指定した値が SRAM に書込まれます。

５－６－５　サンプルコード

「¥SDK¥Algo¥Sample¥Sample_SRAM¥SramReadWriteCheck」にバックアップ SRAM のデータ Read/Write を行うサンプルコードを用意しています。リスト 5-6-5-1 にサンプルコードを示します。

リスト 5-6-5-1．バックアップ SRAM

```
/**
    バックアップ SRAM 制御方法サンプルソース
**/
#include <windows.h>
#include <winioctl.h>
#include <stdio.h>
#include <stdlib.h>
#include <mmsystem.h>
#include <conio.h>

#include "..¥Common¥SramDrvDD.h"

#define DRIVER_FILENAME "¥¥¥¥.¥¥SramDrv"

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

unsigned char rasram[RASRAM_SIZE];

int main(int argc, char **argv)
{
    ULONG i;
    HANDLE hSram;
    ULONG retlen;
    BOOL ret;
    UCHAR d;
    SRAMDRV_RW_PAR  rw_par;

    /*
     * ドライバオブジェクトの作成
     */
    hSram = CreateFile(
                DRIVER_FILENAME,
                GENERIC_READ | GENERIC_WRITE,
                FILE_SHARE_READ | FILE_SHARE_WRITE,
                NULL,
                OPEN_EXISTING,
                0,
                NULL
           );
    if(hSram == INVALID_HANDLE_VALUE){
        printf("CreateFile: NG¥n");
        return -1;
    }

    /*
     * 書込みデータの作成
```

```
 */
for (i = 0, d = 1; i < RASRAM_SIZE; i++, d++){
    rasram[i] = d;
}

/*
 *  データの書込み
 *
 *   rw_par.RW
 *       リードライト  0: リード
 *                     1: ライト
 *
 *   rw_par.Type
 *       タイプ        1: 8 ビット
 *                     2: 16 ビット
 *
 *   rw_par.Addr
 *       アドレス      0x00000 ～ 0x7F000
 *
 *   rw_par.BufferPtr
 *       データポインタ
 *
 *   rw_par.Lenght
 *       サイズ        0x00000 ～ 0x7F000
 *
 */
rw_par.RW = RW_WRITE;
rw_par.Type = TYPE_BYTE;
rw_par.Addr = 0;v
rw_par.BufferPtr = (ULONG)rasram;
rw_par.Length = RASRAM_SIZE;

ret = DeviceIoControl(
            hSram,
            IOCTL_SRAMDRV_RW,
            &rw_par,
            sizeof(SRAMDRV_RW_PAR),
            &rw_par,
            sizeof(SRAMDRV_RW_PAR),
            &retlen,
            NULL
            );
if(!ret){
    errno = GetLastError();
    fprintf(stderr, "ioctl set IOCTL_SRAMDRV_RW error: %d¥n", errno);
    CloseHandle(hSram);
    return -1;
}

/*
 * データの読み込み
 */
```

```
    memset(&rasram[0], 0x00, RASRAM_SIZE);

    rw_par.RW = RW_READ;
    rw_par.Type = TYPE_BYTE;
    rw_par.Addr = 0;
    rw_par.BufferPtr = (ULONG)rasram;
    rw_par.Length = RASRAM_SIZE;

    ret = DeviceIoControl(
                hSram,
                IOCTL_SRAMDRV_RW,
                &rw_par,
                sizeof(SRAMDRV_RW_PAR),
                &rw_par,
                sizeof(SRAMDRV_RW_PAR),
                &retlen,
                NULL
                );
    if(!ret){
        errno = GetLastError();
        fprintf(stderr, "ioctl set IOCTL_SRAMDRV_RW error: %d¥n", errno);
        CloseHandle(hSram);
        return -1;
    }

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

    CloseHandle(hSram);
    return 0;
}
```

５－７　　ハードウェア・ウォッチドッグタイマ機能

５－７－１　　ハードウェア・ウォッチドッグタイマ機能について

FP シリーズには、ハードウェアによるウォッチドッグタイマ機能が実装されています。ハードウェア・ウォッチドッグタイマドライバを操作することで、アプリケーションからウォッチドッグタイマ機能を利用できます。

ソフトウェア・ウォッチドッグタイマと異なり、電源 OFF、リセットなどハードウェアによるタイムアウト処理が利用できます。ハードウェアによるタイムアウト処理は、OS がハングアップしたような状況でも強制的に実行させることができます。

シャットダウン、再起動、ポップアップ、イベント通知のタイムアウト処理は、ソフトウェア・ウォッチドッグタイマと同等の処理となります。タイムアウト処理がソフトウェアとなるため、OS がハングアップした場合などは、タイムアウト処理が実行されない場合があります。

※　タイムアウト処理を電源 OFF、リセットにする場合、シャットダウン処理は行われません。EWF 機能を使用しシステムを保護するようにしてください。

※　タイムアウト処理を電源 OFF にする場合、POWER スイッチを押したときの動作を設定する必要があります。「2-13-5 Watchdog Timer Configuration」を参考に電源オプションの設定を行ってください。

５－７－２　　ハードウェア・ウォッチドッグタイマドライバについて

ハードウェア・ウォッチドッグタイマドライバはウォッチドッグタイマ機能を、ユーザーアプリケーションから利用できるようにします。

ユーザーアプリケーションから動作設定、開始/停止、タイマクリアを行うことができます。

タイムアウト処理を電源 OFF、リセットに設定した場合は、タイムアウトと同時にハードウェアによって強制的に電源 OFF、リセットが行われます。

タイムアウト処理をシャットダウン、再起動、ポップアップ通知に設定した場合は、タイムアウトはハードウェア・ウォッチドッグタイマ監視サービスに通知されます。ハードウェア・ウォッチドッグタイマ監視サービスは設定に従い、シャットダウン、再起動、ポップアップ通知の処理を行います。

タイムアウト処理をイベント通知に設定した場合は、タイムアウト通知をユーザーアプリケーションでイベントとして取得することができます。ユーザーアプリケーションで独自のタイムアウト処理を行うことができます。

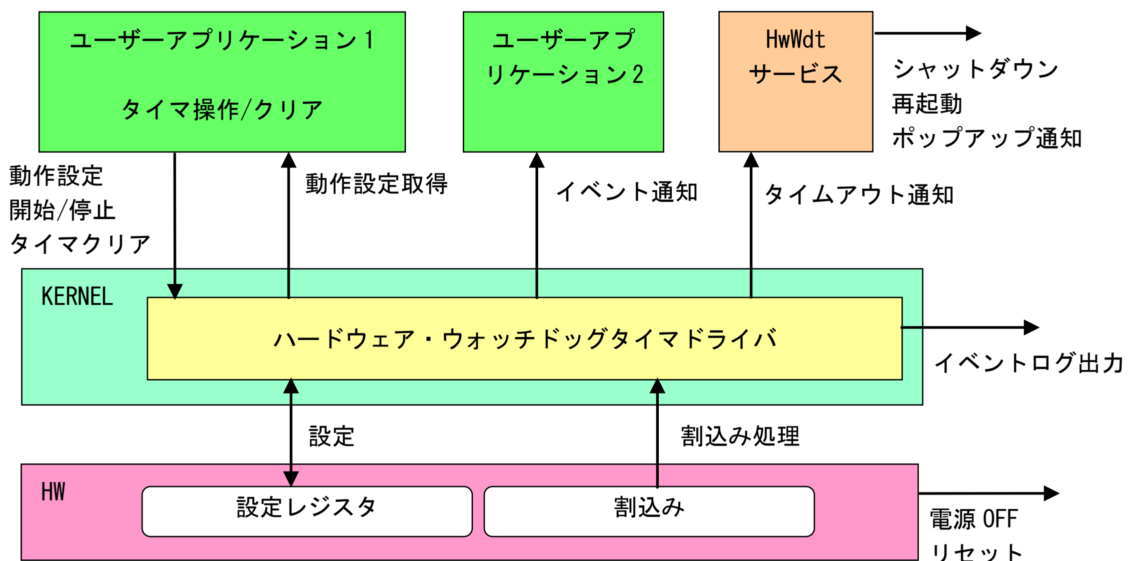

図 5-7-2-1.　ハードウェア・ウォッチドッグタイマドライバ

ハードウェア・ウォッチドッグタイマは、タイムアウト発生を Windows イベントログに記録することができます。
タイムアウト処理が電源 OFF、リセットの場合は、再起動時にイベントログ出力が行われます。
タイムアウト処理がシャットダウン、再起動、ポップアップ通知、イベント通知の場合は、タイムアウト発生直後にイベントログ出力が行われます。

※　タイムアウト処理が電源 OFF の場合、タイムアウト発生後に電源供給を断ってしまうとタイムアウト情報が消えてしまい、イベントログ出力は行われません。イベントログを記録する場合は、電源供給を断つ前に再起動してください。

５－７－３　ハードウェア・ウォッチドッグタイマデバイス

ハードウェア・ウォッチドッグタイマドライバはハードウェア・ウォッチドッグタイマデバイスを生成します。ユーザーアプリケーションは、デバイスファイルにアクセスすることによってウォッチドッグタイマ機能を操作します。

ハードウェア・ウォッチドッグタイマデバイス

デバイスファイル

¥¥.¥HwWdtDrv

説明

ハードウェア・ウォッチドッグタイマの動作設定、開始/停止、タイマクリアを行うことができます。

レジストリ設定

[KEY]　HKEY_LOCAL_MACHINE¥SYSTEM¥CurrentControlSet¥Services¥HwWdtDrv¥Parameters
[VALUE:DWORD]　Action

タイムアウト時の動作を設定します。タイムアウト時の動作は、デバイスオープン時に、このレジスタ値に初期化されます。(デフォルト値: 1)
「Watchdog Timer Config Tool」で設定可能。

0: 電源OFF
1: リセット
2: シャットダウン
3: 再起動
4: ポップアップ通知
5: イベント通知

[KEY]　HKEY_LOCAL_MACHINE¥SYSTEM¥CurrentControlSet¥Services¥HwWdtDrv¥Parameters
[VALUE:DWORD]　Time

タイムアウト時間を設定します。タイムアウト時間は、デバイスオープン時に、このレジスタ値に初期化されます。タイムアウト時間は[Time x 100msec]となります。(デフォルト値: 20)
「Watchdog Timer Config Tool」で設定可能。

1～160

[KEY]　HKEY_LOCAL_MACHINE¥SYSTEM¥CurrentControlSet¥Services¥HwWdtDrv¥Parameters
[VALUE:DWORD]　EventLog

タイムアウト時のイベントログ出力を設定します。イベントログ出力の設定は、デバイスオープン時にこのレジスタ値に初期化されます。(デフォルト値: 1)
「Watchdog Timer Config Tool」で設定可能。

0: 無効
1: 有効

CreateFile

デバイスファイル(¥¥.¥HwWdtDrv)をオープンし、デバイスハンドルを取得します。

```
hWdog = CreateFile(
    "¥¥¥¥.¥¥HwWdtDrv",
    GENERIC_READ | GENERIC_WRITE,
    FILE_SHARE_READ | FILE_SHARE_WRITE,
    NULL,
    OPEN_EXISTING,
    0,
    NULL
);
```

ReadFile

使用しません。

WriteFile

使用しません。

DeviceIoControl

- IOCTL_HWWDT_SETCONFIG
  ハードウェア・ウォッチドッグタイマの動作設定を行います。
- IOCTL_HWWDT_GETCONFIG
  ハードウェア・ウォッチドッグタイマの動作設定を取得します。
- IOCTL_HWWDT_CONTROL
  ハードウェア・ウォッチドッグタイマの開始/停止を行います。
- IOCTL_HWWDT_STATUS
  ハードウェア・ウォッチドッグタイマの動作状態を取得します。
- IOCTL_HWWDT_CLEAR
  ハードウェア・ウォッチドッグタイマのタイマクリアを行います。

５－７－４　DeviceIoControl リファレンス

IOCTL_HWWDT_SETCONFIG

機能

ハードウェア・ウォッチドッグタイマの動作設定を行います。

パラメータ

lpInBuf　　　　: HWWDT_CONFIG を格納するポインタを指定します。
nInBufSize　　 : HWWDT_CONFIG を格納するポインタのサイズを指定します。
lpOutBuf　　　 : NULL を指定します。
nOutBufSize　　: 0 を指定します。
lpBytesReturned : 実際の出力バイト数を受け取る変数へのポインタを指定します。
lpOverlapped　 : NULL を指定します。

HWWDT_CONFIG

```
typedef struct {
        ULONG Action;
        ULONG Time;
        ULONG EventLog;
} HWWDT_CONFIG, *PHWWDT_CONFIG;
```

Action　　: タイムアウト時の動作 (0～5)
[0: 電源 OFF, 1: リセット, 2: シャットダウン, 3: 再起動,
4: ポップアップ, 5: イベント通知]

Time　　　: タイムアウト時間 (1～160)
[Time x 100msec]

EventLog　: イベントログ出力 (0, 1)
[0: 無効, 1: 有効]
Action が 0(電源 OFF)、1(リセット)の場合は無効です。
Action が 0(電源 OFF)、1(リセット)の場合はレジストリ設定に従い、
イベントログ出力を行います。

戻り値

処理が成功すると TRUE を返します。失敗の場合は FALSE を返します。

説明

ハードウェア・ウォッチドッグタイマの動作設定を行います。
動作設定はデバイスオープン時にレジスタ設定値に初期化されます。オープン後に動作を変更したい場合は、この IOCTL コードを実行します。
タイムアウト時の動作が 0(電源 OFF)、1(リセット)の場合は、イベントログ出力の設定は無視されます。この場合、レジストリの設定値に従ってイベントログ出力を行います。

IOCTL_HWWDT_GETCONFIG

機能

ハードウェア・ウォッチドッグタイマの動作設定を取得します。

パラメータ

lpInBuf　: NULL を指定します。
nInBufSize　: 0 を指定します。
lpOutBuf　: HWWDT_CONFIG を格納するポインタを指定します。
nOutBufSize　: HWWDT_CONFIG を格納するポインタのサイズを指定します。
lpBytesReturned　: 実際の出力バイト数を受け取る変数へのポインタを指定します。
lpOverlapped　: NULL を指定します。

戻り値

処理が成功すると TRUE を返します。失敗の場合は FALSE を返します。

説明

ハードウェア・ウォッチドッグタイマの動作設定を取得します。

IOCTL_HWWDT_CONTROL

機能

ハードウェア・ウォッチドッグタイマの開始/停止を行います。

パラメータ

lpInBuf	:	タイマ制御情報を格納するポインタを指定します。
nInBufSize	:	タイマ制御情報を格納するポインタのサイズを指定します。
lpOutBuf	:	NULL を指定します。
nOutBufSize	:	0 を指定します。
lpBytesReturned	:	実際の出力バイト数を受け取る変数へのポインタを指定します。
lpOverlapped	:	NULL を指定します。

タイマ制御情報

データタイプ	:	ULONG
データサイズ	:	4 バイト
内容	:	0: タイマ停止 1: タイマ開始（デバイスクローズ時続行） 2: タイマ開始（デバイスクローズ時停止）

戻り値

処理が成功すると TRUE を返します。失敗の場合は FALSE を返します。

説明

ハードウェア・ウォッチドッグタイマの開始/停止の制御を行います。
タイマ動作を開始する場合、デバイスクローズ時にタイマ動作を停止させるか、続行させるかを指定することができます。

IOCTL_HWWDT_STATUS

機能

ハードウェア・ウォッチドッグタイマの動作状態を取得します。

パラメータ

lpInBuf	:	NULL を指定します。
nInBufSize	:	0 を指定します。
lpOutBuf	:	タイマ制御情報を格納するポインタを指定します。
nOutBufSize	:	タイマ制御情報を格納するポインタのサイズを指定します。
lpBytesReturned	:	実際の出力バイト数を受け取る変数へのポインタを指定します。
lpOverlapped	:	NULL を指定します。

戻り値

処理が成功すると TRUE を返します。失敗の場合は FALSE を返します。

説明

ハードウェア・ウォッチドッグタイマの動作状態を取得します。
IOCTL_HWWDT_CONTROL でのタイマ制御状態を取得することができます。

IOCTL_HWWDT_CLEAR

機能

ハードウェア・ウォッチドッグタイマのタイマクリアを行います。

パラメータ

lpInBuf	:	NULL を指定します。
nInBufSize	:	0 を指定します。
lpOutBuf	:	NULL を指定します。
nOutBufSize	:	0 を指定します。
lpBytesReturned	:	実際の出力バイト数を受け取る変数へのポインタを指定します。
lpOverlapped	:	NULL を指定します。

戻り値

処理が成功すると TRUE を返します。失敗の場合は FALSE を返します。

説明

ハードウェア・ウォッチドッグタイマのタイマクリアを行います。
タイマクリアすると、タイマが初期化されタイマカウントが再開されます。

５－７－５　サンプルコード

●タイマ操作

「¥SDK¥Algo¥Sample¥Sample_HwWdt¥HwWdt」にハードウェア・ウォッチドッグタイマのタイマ操作のサンプルコードを用意しています。リスト 5-7-5-1 にサンプルコードを示します。

リスト 5-7-5-1. ハードウェア・ウォッチドッグタイマ タイマ操作

```
/**
    ハードウェアウォッチドッグタイマ
    タイマ操作サンプルソース
**/
#include <windows.h>
#include <winioctl.h>
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>

#include "..¥Common¥HwWdtDD.h"

#define DRIVER_FILENAME "¥¥¥¥.¥¥HwWdtDrv"

int main(int argc, char **argv)
{
    HANDLE h_swwdt;
    ULONG retlen;
    BOOL ret;

    int wdt_action;
    int wdt_time;
    int wdt_eventlog;
    HWWDT_CONFIG wdt_config;

    ULONG startval;

    int keych;

    /*
     * 引数から動作を取得します。
     * 引数なしの場合は、動作設定を変更しません。
     */
    if(argc == 4){
        sscanf(*(argv + 1), "%d", &wdt_action);
        sscanf(*(argv + 2), "%d", &wdt_time);
        sscanf(*(argv + 3), "%d", &wdt_eventlog);
    }
    else if(argc != 1){
        printf("invalid arg¥n");
        printf("HwWdtClear.exe [<WDT ACTION> <WDT TIME> <WDT EVENTLOG>]¥n");
        return -1;
    }
```

```
/*
 * ハードウェアウォッチドッグの OPEN
 */
h_swwdt = CreateFile(
                DRIVER_FILENAME,
                GENERIC_READ | GENERIC_WRITE,
                FILE_SHARE_READ | FILE_SHARE_WRITE,
                NULL,
                OPEN_EXISTING,
                0,
                NULL
          );
if(h_swwdt == INVALID_HANDLE_VALUE){
    printf("CreateFile: NG¥n");
    return -1;
}

/*
 * OPEN 直後は、動作設定がデフォルト値となります。
 * 引数でタイムアウト動作、タイムアウト時間を
 * 指定した場合は、動作設定を変更します。
 */
if(argc != 1){
    wdt_config.Action = (ULONG)wdt_action;
    wdt_config.Time = (ULONG)wdt_time;
    wdt_config.EventLog = (ULONG)wdt_eventlog;
    ret = DeviceIoControl(
                h_swwdt,
                IOCTL_HWWDT_SETCONFIG,
                &wdt_config,
                sizeof(HWWDT_CONFIG),
                NULL,
                0,
                &retlen,
                NULL
          );
    if(!ret){
        printf("DeviceIoControl: IOCTL_HWWDT_SETCONFIG NG¥n");
        CloseHandle(h_swwdt);
        return -1;
    }
}

/*
 * 動作設定の表示
 */
memset(&wdt_config, 0x00, sizeof(HWWDT_CONFIG));
ret = DeviceIoControl(
            h_swwdt,
            IOCTL_HWWDT_GETCONFIG,
            NULL,
            0,
```

```
                &wdt_config,
                sizeof(HWWDT_CONFIG),
                &retlen,
                NULL
        );
    if(!ret){
        printf("DeviceIoControl: IOCTL_HWWDT_GETCONFIG NG¥n");
        CloseHandle(h_swwdt);
        return -1;
    }
    printf("HwWdt Action = %d¥n", wdt_config.Action);
    printf("HwWdt Time = %d¥n", wdt_config.Time);
    printf("HwWdt EventLog = %d¥n", wdt_config.EventLog);

    /*
     * ウォッチドッグタイマスタート
     */
    startval = HWWDT_CONTROL_START;     // クローズしても停止しません
    ret = DeviceIoControl(
                h_swwdt,
                IOCTL_HWWDT_CONTROL,
                &startval,
                sizeof(ULONG),
                NULL,
                0,
                &retlen,
                NULL
        );
    if(!ret){
        printf("DeviceIoControl: IOCTL_HWWDT_CONTROL NG¥n");
        CloseHandle(h_swwdt);
        return -1;
    }

    /*
     * タイマクリア処理('Q'または'q'キーで終了します)
     */
    while(1){
        if(kbhit()){
            keych = getch();
            if(keych == 'Q' || keych == 'q'){
                break;
            }
        }

        /*
         * クリア
         */
        DeviceIoControl(
            h_swwdt,
            IOCTL_HWWDT_CLEAR,
            NULL,
```

```
            0,
            NULL,
            0,
            &retlen,
            NULL
        );
        Sleep(100);
    }

    /*
     * ウォッチドッグタイマ停止
     */
    startval = HWWDT_CONTROL_STOP;
    ret = DeviceIoControl(
                h_swwdt,
                IOCTL_HWWDT_CONTROL,
                &startval,
                sizeof(ULONG),
                NULL,
                0,
                &retlen,
                NULL
          );
    if(!ret){
        printf("DeviceIoControl: IOCTL_HWWDT_CONTROL NG¥n");
        CloseHandle(h_swwdt);
        return -1;
    }

    CloseHandle(h_swwdt);
    return 0;
}
```

●イベント通知取得

タイムアウト時の動作をイベント通知に設定した場合、ユーザーアプリケーションでタイムアウト通知をイベントとして取得することができます。

「¥SDK¥Algo¥Sample¥Sample_HwWdt¥HwWdt」にハードウェア・ウォッチドッグタイマのイベント取得処理のサンプルコードを用意しています。リスト 5-7-5-2 にサンプルコードを示します。

リスト 5-7-5-2. ハードウェア・ウォッチドッグタイマ イベント通知取得

```
/**
    ハードウェアウォッチドッグタイマ
    イベント取得サンプルソース
**/
#include <windows.h>
#include <winioctl.h>
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>

#include "..¥Common¥HwWdtDD.h"

#define THREADSTATE_STOP        0
#define THREADSTATE_RUN         1
#define THREADSTATE_QUERY_STOP  2

#define MAX_EVENT   2

enum {
    EVENT_FIN = 0,
    EVENT_USER
};

HANDLE hEvent[MAX_EVENT];
HANDLE hThread;
ULONG ThreadState;

DWORD WINAPI EventThread(void *pData)
{
    DWORD ret;

    printf("EventThread: Start¥n");
    ThreadState = THREADSTATE_RUN;

    /*
     * ウォッチドッグ ユーザーイベントを待ちます
     */
    while(1){
        ret = WaitForMultipleObjects(MAX_EVENT, &hEvent[0], FALSE, INFINITE);
        if(ret == WAIT_FAILED){
            break;
        }
        if(ThreadState == THREADSTATE_QUERY_STOP){
            break;
        }
```

```
        if(ret == WAIT_OBJECT_0 + EVENT_USER){
            printf("EventThread: UserEvent¥n");
        }
    }
    ThreadState = THREADSTATE_STOP;

    printf("EventThread: Finish¥n");
    return 0;
}


int main(int argc, char **argv)
{
    DWORD thid;
    int keych;
    int i;

    /*
     * スレッド終了用イベント
     */
    hEvent[EVENT_FIN] = CreateEvent(NULL, FALSE, FALSE, NULL);

    /*
     * ウォッチドッグ ユーザーイベント ハンドル取得
     */
    hEvent[EVENT_USER] = OpenEvent(SYNCHRONIZE, FALSE, HWWDT_USER_EVENT_NAME);
    if(hEvent[EVENT_USER] == NULL){
        printf("CreateEvent: NG¥n");
        return -1;
    }

    /*
     * ウォッチドッグ ユーザーイベント取得スレッド ハンドル取得
     */
    ThreadState = THREADSTATE_STOP;
    hThread = CreateThread(
                    (LPSECURITY_ATTRIBUTES)NULL,
                    0,
                    (LPTHREAD_START_ROUTINE)EventThread,
                    NULL,
                    0,
                    &thid
                );
    if(hThread == NULL){
        CloseHandle(hEvent);
        printf("CreateThread: NG¥n");
        return -1;
    }

    /*
     * 'Q'または'q'キーで終了します。
```

```
     */
    while(1){
        if(kbhit()){
            keych = getch();
            if(keych == 'Q' || keych == 'q'){
                break;
            }
        }
    }

    /*
     * スレッドを終了
     */
    ThreadState = THREADSTATE_QUERY_STOP;
    SetEvent(hEvent[EVENT_FIN]);
    while(ThreadState != THREADSTATE_STOP){
        Sleep(10);
    }
    CloseHandle(hThread);
    for(i = 0; i < MAX_EVENT; i++){
        CloseHandle(hEvent[i]);
    }

    return 0;
}
```

５－８　ソフトウェア・ウォッチドッグタイマ機能

５－８－１　ソフトウェア・ウォッチドッグタイマ機能について

FP シリーズには、ソフトウェアによるウォッチドッグタイマ機能が実装されています。ドライバでタイマ機能を構築し、アプリケーションからウォッチドッグタイマ機能を利用できるようにします。

ソフトウェア・ウォッチドッグタイマは、タイムアウト処理がソフトウェアとなります。OS がハングアップした場合などは、タイムアウト処理が実行されない場合があります。

５－８－２　ソフトウェア・ウォッチドッグタイマドライバについて

ソフトウェア・ウォッチドッグタイマドライバはウォッチドッグタイマ機能を、ユーザーアプリケーションから利用できるようにします。

ユーザーアプリケーションから動作設定、開始/停止、タイマクリアを行うことができます。

タイムアウト処理をシャットダウン、再起動、ポップアップ通知に設定した場合、タイムアウトはソフトウェア・ウォッチドッグタイマ監視サービスに通知されます。ソフトウェア・ウォッチドッグタイマ監視サービスは設定に従い、シャットダウン、再起動、ポップアップ通知の処理を行います。

タイムアウト処理をイベント通知とした場合、タイムアウト通知をユーザーアプリケーションでイベントとして取得することができます。ユーザーアプリケーションで独自のタイムアウト処理を行うことができます。

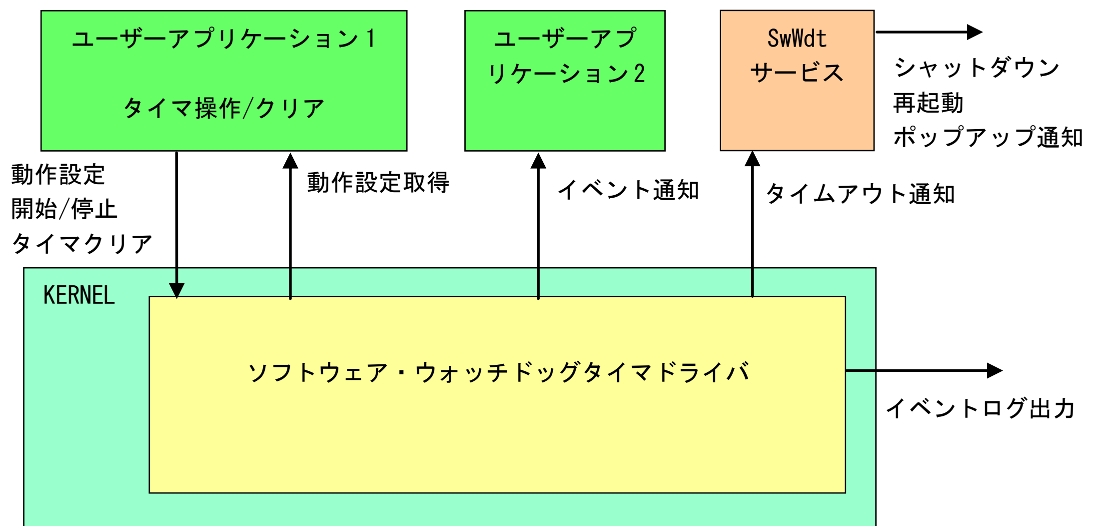

図 5-8-2-1. ソフトウェア・ウォッチドッグタイマドライバ

ソフトウェア・ウォッチドッグタイマは、タイムアウト発生を Windows イベントログに記録することができます。タイムアウト発生直後にイベントログ出力が行われます。

５－８－３　ソフトウェア・ウォッチドッグタイマデバイス

ソフトウェア・ウォッチドッグタイマドライバはソフトウェア・ウォッチドッグタイマデバイスを生成します。ユーザーアプリケーションは、デバイスファイルにアクセスすることによってウォッチドッグタイマ機能を操作します。

ソフトウェア・ウォッチドッグタイマデバイス

デバイスファイル

¥¥.¥SwWdtDrv

説明

ソフトウェア・ウォッチドッグタイマの動作設定、開始/停止、タイマクリアを行うことができます。

レジストリ設定

[KEY]　HKEY_LOCAL_MACHINE¥SYSTEM¥CurrentControlSet¥Services¥SwWdtDrv¥Parameters
[VALUE:DWORD]　Action

タイムアウト時の動作を設定します。タイムアウト時の動作は、デバイスオープン時に、このレジスタ値に初期化されます。(デフォルト値: 2)
「Software Watchdog Timer Config Tool」で設定可能。

0: シャットダウン
1: 再起動
2: ポップアップ通知
3: イベント通知

[KEY]　HKEY_LOCAL_MACHINE¥SYSTEM¥CurrentControlSet¥Services¥SwWdtDrv¥Parameters
[VALUE:DWORD]　Time

タイムアウト時間を設定します。タイムアウト時間は、デバイスオープン時に、このレジスタ値に初期化されます。タイムアウト時間は[Time x 100msec]となります。
(デフォルト値: 20)
「Software Watchdog Timer Config Tool」で設定可能。

1～160

[KEY]　HKEY_LOCAL_MACHINE¥SYSTEM¥CurrentControlSet¥Services¥SwWdtDrv¥Parameters
[VALUE:DWORD]　EventLog

タイムアウト時のイベントログ出力を設定します。イベントログ出力の設定は、デバイスオープン時にこのレジスタ値に初期化されます。(デフォルト値: 1)
「Software Watchdog Timer Config Tool」で設定可能。

0: 無効
1: 有効

CreateFile

デバイスファイル(¥¥.¥SwWdtDrv)をオープンし、デバイスハンドルを取得します。

```
hWdog = CreateFile(
    "¥¥¥¥.¥¥SwWdtDrv",
    GENERIC_READ | GENERIC_WRITE,
    FILE_SHARE_READ | FILE_SHARE_WRITE,
    NULL,
    OPEN_EXISTING,
    0,
    NULL
);
```

ReadFile

使用しません。

WriteFile

使用しません。

DeviceIoControl

- **IOCTL_SWWDT_SETCONFIG**
  ソフトウェア・ウォッチドッグタイマの動作設定を行います。
- **IOCTL_SWWDT_GETCONFIG**
  ソフトウェア・ウォッチドッグタイマの動作設定を取得します。
- **IOCTL_SWWDT_CONTROL**
  ソフトウェア・ウォッチドッグタイマの開始/停止を行います。
- **IOCTL_SWWDT_STATUS**
  ソフトウェア・ウォッチドッグタイマの動作状態を取得します。
- **IOCTL_SWWDT_CLEAR**
  ソフトウェア・ウォッチドッグタイマのタイマクリアを行います。

５－８－４　DeviceIoControl リファレンス

IOCTL_SWWDT_SETCONFIG

機能

ソフトウェア・ウォッチドッグタイマの動作設定を行います。

パラメータ

lpInBuf　：　SWWDT_CONFIG を格納するポインタを指定します。
nInBufSize　：　SWWDT_CONFIG を格納するポインタのサイズを指定します。
lpOutBuf　：　NULL を指定します。
nOutBufSize　：　0 を指定します。
lpBytesReturned　：　実際の出力バイト数を受け取る変数へのポインタを指定します。
lpOverlapped　：　NULL を指定します。

SWWDT_CONFIG

```
typedef struct {
        ULONG Action;
        ULONG Time;
        ULONG EventLog;
} SWWDT_CONFIG, *PSWWDT_CONFIG;
```

Action　：タイムアウト時の動作（0～3）
[0：シャットダウン，1：再起動，2：ポップアップ，3：イベント通知]
Time　：タイムアウト時間（1～160）
[Time x 100msec]
EventLog　：イベントログ出力（0，1）
[0：無効，1：有効]

戻り値

処理が成功すると TRUE を返します。失敗の場合は FALSE を返します。

説明

ソフトウェア・ウォッチドッグタイマの動作設定を行います。
動作設定はデバイスオープン時にレジスタ設定値に初期化されます。オープン後に動作を変更したい場合は、この IOCTL コードを実行します。

IOCTL_SWWDT_GETCONFIG

機能

ソフトウェア・ウォッチドッグタイマの動作設定を取得します。

パラメータ

lpInBuf　：　NULL を指定します。
nInBufSize　：　0 を指定します。
lpOutBuf　：　SWWDT_CONFIG を格納するポインタを指定します。
nOutBufSize　：　SWWDT_CONFIG を格納するポインタのサイズを指定します。
lpBytesReturned　：　実際の出力バイト数を受け取る変数へのポインタを指定します。
lpOverlapped　：　NULL を指定します。

戻り値

処理が成功すると TRUE を返します。失敗の場合は FALSE を返します。

説明

ソフトウェア・ウォッチドッグタイマの動作設定を取得します。

IOCTL_SWWDT_CONTROL

機能

ソフトウェア・ウォッチドッグタイマの開始/停止を行います。

パラメータ

lpInBuf	:	タイマ制御情報を格納するポインタを指定します。
nInBufSize	:	タイマ制御情報を格納するポインタのサイズを指定します。
lpOutBuf	:	NULL を指定します。
nOutBufSize	:	0 を指定します。
lpBytesReturned	:	実際の出力バイト数を受け取る変数へのポインタを指定します。
lpOverlapped	:	NULL を指定します。

タイマ制御情報

データタイプ	:	ULONG
データサイズ	:	4 バイト
内容	:	0: タイマ停止 1: タイマ開始（デバイスクローズ時続行） 2: タイマ開始（デバイスクローズ時停止）

戻り値

処理が成功すると TRUE を返します。失敗の場合は FALSE を返します。

説明

ソフトウェア・ウォッチドッグタイマの開始/停止の制御を行います。
タイマ動作を開始する場合、デバイスクローズ時にタイマ動作を停止させるか、続行させるかを指定することができます。

IOCTL_SWWDT_STATUS

機能

ソフトウェア・ウォッチドッグタイマの動作状態を取得します。

パラメータ

lpInBuf　：　NULL を指定します。
nInBufSize　：　0 を指定します。
lpOutBuf　：　タイマ制御情報を格納するポインタを指定します。
nOutBufSize　：　タイマ制御情報を格納するポインタのサイズを指定します。
lpBytesReturned　：　実際の出力バイト数を受け取る変数へのポインタを指定します。
lpOverlapped　：　NULL を指定します。

戻り値

処理が成功すると TRUE を返します。失敗の場合は FALSE を返します。

説明

ソフトウェア・ウォッチドッグタイマの動作状態を取得します。
IOCTL_SWWDT_CONTROL でのタイマ制御状態を取得することができます。

IOCTL_SWWDT_CLEAR

機能

ソフトウェア・ウォッチドッグタイマのタイマクリアを行います。

パラメータ

lpInBuf	:	NULL を指定します。
nInBufSize	:	0 を指定します。
lpOutBuf	:	NULL を指定します。
nOutBufSize	:	0 を指定します。
lpBytesReturned	:	実際の出力バイト数を受け取る変数へのポインタを指定します。
lpOverlapped	:	NULL を指定します。

戻り値

処理が成功すると TRUE を返します。失敗の場合は FALSE を返します。

説明

ソフトウェア・ウォッチドッグタイマのタイマクリアを行います。
タイマクリアすると、タイマが初期化されタイマカウントが再開されます。

５－８－５　サンプルコード

●タイマ操作

「¥SDK¥Algo¥Sample¥Sample_SwWdt¥SwWdt」にソフトウェア・ウォッチドッグタイマのタイマ操作のサンプルコードを用意しています。リスト 5-8-5-1 にサンプルコードを示します。

リスト 5-8-5-1. ソフトウェア・ウォッチドッグタイマ タイマ操作

```
/**
    ソフトウェアウォッチドッグタイマ
    タイマ操作サンプルソース
**/
#include <windows.h>
#include <winioctl.h>
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>

#include "..¥Common¥SwWdtDD.h"

#define DRIVER_FILENAME "¥¥¥¥.¥¥SwWdtDrv"

int main(int argc, char **argv)
{
    HANDLE h_swwdt;
    ULONG retlen;
    BOOL ret;

    int wdt_action;
    int wdt_time;
    int wdt_eventlog;
    SWWDT_CONFIG wdt_config;

    ULONG startval;

    int keych;

    /*
     * 引数から動作を取得します。
     * 引数なしの場合は、動作設定を変更しません。
     */
    if(argc == 4){
        sscanf(*(argv + 1), "%d", &wdt_action);
        sscanf(*(argv + 2), "%d", &wdt_time);
        sscanf(*(argv + 3), "%d", &wdt_eventlog);
    }
    else if(argc != 1){
        printf("invalid arg¥n");
        printf("SwWdtClear.exe [<WDT ACTION> <WDT TIME> <WDT EVENTLOG>]¥n");
        return -1;
    }
```

```
/*
 * ソフトウェアウォッチドッグの OPEN
 */
h_swwdt = CreateFile(
                DRIVER_FILENAME,
                GENERIC_READ | GENERIC_WRITE,
                FILE_SHARE_READ | FILE_SHARE_WRITE,
                NULL,
                OPEN_EXISTING,
                0,
                NULL
          );
if(h_swwdt == INVALID_HANDLE_VALUE){
    printf("CreateFile: NG¥n");
    return -1;
}

/*
 * OPEN 直後は、動作設定がデフォルト値となります。
 * 引数でタイムアウト動作、タイムアウト時間を
 * 指定した場合は、動作設定を変更します。
 */
if(argc != 1){
    wdt_config.Action = (ULONG)wdt_action;
    wdt_config.Time = (ULONG)wdt_time;
    wdt_config.EventLog = (ULONG)wdt_eventlog;
    ret = DeviceIoControl(
                h_swwdt,
                IOCTL_SWWDT_SETCONFIG,
                &wdt_config,
                sizeof(SWWDT_CONFIG),
                NULL,
                0,
                &retlen,
                NULL
          );
    if(!ret){
        printf("DeviceIoControl: IOCTL_SWWDT_SETCONFIG NG¥n");
        CloseHandle(h_swwdt);
        return -1;
    }
}

/*
 * 動作設定の表示
 */
memset(&wdt_config, 0x00, sizeof(SWWDT_CONFIG));
ret = DeviceIoControl(
            h_swwdt,
            IOCTL_SWWDT_GETCONFIG,
            NULL,
            0,
```

```
                &wdt_config,
                sizeof(SWWDT_CONFIG),
                &retlen,
                NULL
        );
    if(!ret){
        printf("DeviceIoControl: IOCTL_SWWDT_GETCONFIG NG¥n");
        CloseHandle(h_swwdt);
        return -1;
    }
    printf("SwWdt Action = %d¥n", wdt_config.Action);
    printf("SwWdt Time = %d¥n", wdt_config.Time);
    printf("SwWdt EventLog = %d¥n", wdt_config.EventLog);

    /*
     * ウォッチドッグタイマスタート
     */
    startval = SWWDT_CONTROL_START;     // クローズしても停止しません
    ret = DeviceIoControl(
                h_swwdt,
                IOCTL_SWWDT_CONTROL,
                &startval,
                sizeof(ULONG),
                NULL,
                0,
                &retlen,
                NULL
        );
    if(!ret){
        printf("DeviceIoControl: IOCTL_SWWDT_CONTROL NG¥n");
        CloseHandle(h_swwdt);
        return -1;
    }

    /*
     * タイマクリア処理('Q'または'q'キーで終了します)
     */
    while(1){
        if(kbhit()){
            keych = getch();
            if(keych == 'Q' || keych == 'q'){
                break;
            }
        }

        /*
         * クリア
         */
        DeviceIoControl(
            h_swwdt,
            IOCTL_SWWDT_CLEAR,
            NULL,
```

```
            0,
            NULL,
            0,
            &retlen,
            NULL
        );
        Sleep(100);
    }

    /*
     * ウォッチドッグタイマ停止
     */
    startval = SWWDT_CONTROL_STOP;
    ret = DeviceIoControl(
                h_swwdt,
                IOCTL_SWWDT_CONTROL,
                &startval,
                sizeof(ULONG),
                NULL,
                0,
                &retlen,
                NULL
          );
    if(!ret){
        printf("DeviceIoControl: IOCTL_SWWDT_CONTROL NG¥n");
        CloseHandle(h_swwdt);
        return -1;
    }

    CloseHandle(h_swwdt);
    return 0;
}
```

●イベント通知取得

タイムアウト時の動作をイベント通知に設定した場合、ユーザーアプリケーションでタイムアウト通知をイベントとして取得することができます。

「¥SDK¥Algo¥Sample¥Sample_SwWdt¥SwWdt」にソフトウェア・ウォッチドッグタイマのイベント取得処理のサンプルコードを用意しています。リスト 5-8-5-2 にサンプルコードを示します。

リスト 5-8-5-2. ソフトウェア・ウォッチドッグタイマ イベント通知取得

```
/**
    ソフトウェアウォッチドッグタイマ
    イベント取得サンプルソース
**/
#include <windows.h>
#include <winioctl.h>
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>

#include "..¥Common¥SwWdtDD.h"

#define THREADSTATE_STOP        0
#define THREADSTATE_RUN         1
#define THREADSTATE_QUERY_STOP  2

#define MAX_EVENT   2

enum {
    EVENT_FIN = 0,
    EVENT_USER
};

HANDLE hEvent[MAX_EVENT];
HANDLE hThread;
ULONG ThreadState;

DWORD WINAPI EventThread(void *pData)
{
    DWORD ret;

    printf("EventThread: Start¥n");
    ThreadState = THREADSTATE_RUN;

    /*
     * ウォッチドッグ ユーザーイベントを待ちます
     */
    while(1){
        ret = WaitForMultipleObjects(MAX_EVENT, &hEvent[0], FALSE, INFINITE);
        if(ret == WAIT_FAILED){
            break;
        }
        if(ThreadState == THREADSTATE_QUERY_STOP){
            break;
        }
```

```
        if(ret == WAIT_OBJECT_0 + EVENT_USER){
            printf("EventThread: UserEvent¥n");
        }
    }
    ThreadState = THREADSTATE_STOP;

    printf("EventThread: Finish¥n");
    return 0;
}


int main(int argc, char **argv)
{
    DWORD thid;
    int keych;
    int i;

    /*
     * スレッド終了用イベント
     */
    hEvent[EVENT_FIN] = CreateEvent(NULL, FALSE, FALSE, NULL);

    /*
     * ウォッチドッグ ユーザーイベント ハンドル取得
     */
    hEvent[EVENT_USER] = OpenEvent(SYNCHRONIZE, FALSE, SWWDT_USER_EVENT_NAME);
    if(hEvent[EVENT_USER] == NULL){
        printf("CreateEvent: NG¥n");
        return -1;
    }

    /*
     * ウォッチドッグ ユーザーイベント取得スレッド ハンドル取得
     */
    ThreadState = THREADSTATE_STOP;
    hThread = CreateThread(
                    (LPSECURITY_ATTRIBUTES)NULL,
                    0,
                    (LPTHREAD_START_ROUTINE)EventThread,
                    NULL,
                    0,
                    &thid
                );
    if(hThread == NULL){
        CloseHandle(hEvent);
        printf("CreateThread: NG¥n");
        return -1;
    }

    /*
     * 'Q'または'q'キーで終了します。
```

```
     */
    while(1){
        if(kbhit()){
            keych = getch();
            if(keych == 'Q' || keych == 'q'){
                break;
            }
        }
    }

    /*
     * スレッドを終了
     */
    ThreadState = THREADSTATE_QUERY_STOP;
    SetEvent(hEvent[EVENT_FIN]);
    while(ThreadState != THREADSTATE_STOP){
        Sleep(10);
    }
    CloseHandle(hThread);
    for(i = 0; i < MAX_EVENT; i++){
        CloseHandle(hEvent[i]);
    }

    return 0;
}
```

５－９　　RAS 監視機能

５－９－１　　RAS 監視機能について

FP シリーズには、CPUCore 温度、内部温度を監視する機能が実装されています。

温度監視サービスは、異常を検知した場合、設定に従いシャットダウン、再起動、ポップアップ通知、イベント通知の処理を行います。

異常時処理をイベント通知とした場合、イベント通知をユーザーアプリケーションでイベントとして取得することができます。

また、DLL を介してアプリケーションから CPUCore 温度、内部温度を取得することができます。DLL は 32 ビットアプリケーション用と 64 ビットアプリケーション用を用意しています。DLL ファイル名、使用できる関数は同じです。作成するアプリケーションに合わせて DLL を選択してください。

５－９－２　　RAS DLL について

RAS DLL（G5RAS.dll）は CPUCore 温度、内部温度の取得をユーザーアプリケーションから利用できるようにします。

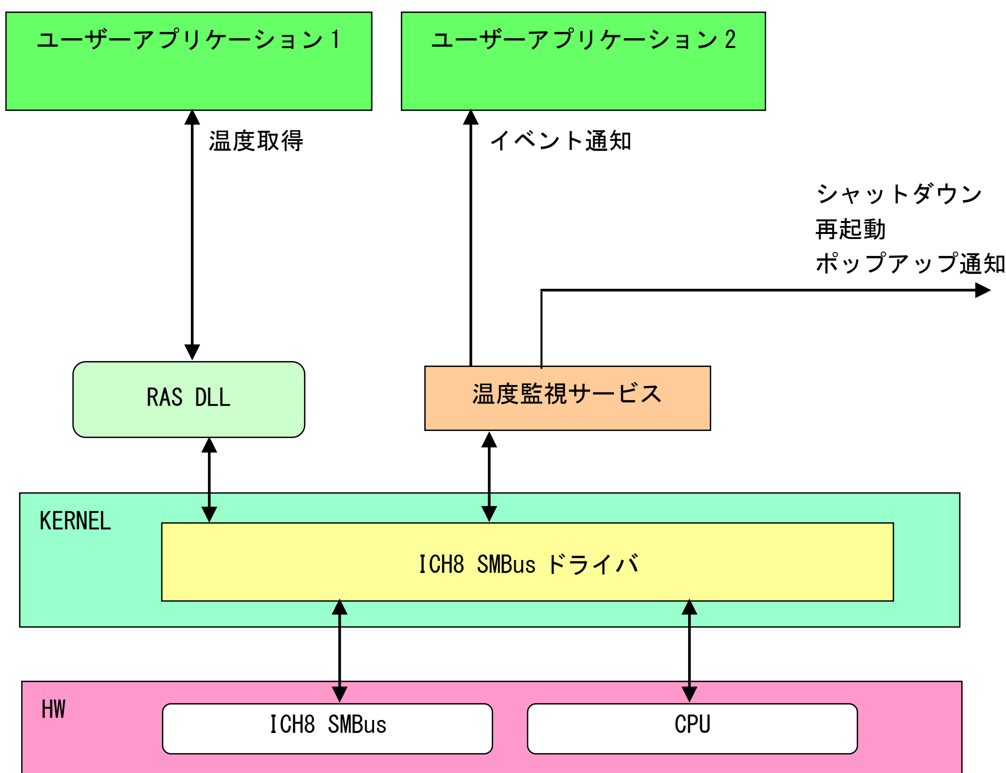

図 5-9-2-1. RAS DLL

５－９－３　RAS DLL I/F 関数リファレンス

G5_GetRemoteTemperature 関数

機能　内部温度を取得します。

書式

```
BOOL  G5_GetRemoteTemperature(double *Temperature)
```

引数　Temperature　　：内部温度を受け取る変数へのポインタ。

戻り値
TRUE　　：正常
FALSE　　：エラー

説明　内部温度を取得します。

G5_GetCPUTemperature 関数

機能　CPUCore 温度を取得します。

書式

```
BOOL  G5_GetCPUTemperature (int CoreNum, WORD *Temperature)
```

引数
CoreNum　　：CPUCore を指定します。(0～1)
Temperature　　：CPUCore 温度を受け取る変数へのポインタ。

戻り値
TRUE　　：正常
FALSE　　：エラー

説明　CPUCore 温度を取得します。

５－９－４　サンプルコード

「¥SDK¥Algo¥Sample¥Sample_RASDll¥RasDll」に RAS DLL を使用して温度の取得を行うサンプルコードを用意しています。リスト 5-9-4-1 にサンプルコードを示します。

リスト 5-9-4-1. RAS DLL

```
/**
    RAS DLL
    サンプルソース
**/
#include <windows.h>
#include <winioctl.h>
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>

#include "..¥Common¥G5RasDef.h"
#include "..¥Common¥AlgG5Ras.h"


int main(int argc, char **argv)
{
    char     fname[100];

    double   exttemp;
    WORD     cputemp;

    //  DLL ロード
    strcpy(&fname[0], "G5RAS.dll");
    LoadG5RasDll(&fname[0]);

    /*
     * 内部温度表示
     */
    G5_GetRemoteTemperature(&exttemp);
    printf("Ext Temperature: %.2f℃¥n", exttemp);

    /*
     * CPU 温度表示
     */
    G5_GetCPUTemperature(0, &cputemp);
    printf("CPU Core#0 Temperature: %d℃¥n", cputemp);
    G5_GetCPUTemperature(1, &cputemp);
    printf("CPU Core#1 Temperature: %d℃¥n", cputemp);

    UnloadG5RasDll();

    return 0;
}
```

●イベント通知取得

異常発生時の動作をイベント通知に設定した場合、ユーザーアプリケーションで異常発生通知をイベントとして取得することができます。

「¥SDK¥Algo¥Sample¥Sample_TempMon¥TempMon」に温度監視のイベント取得処理のサンプルコードを用意しています。リスト 5-9-4-2 に温度監視のイベント取得サンプルコードを示します。

リスト 5-9-4-2. 温度監視 イベント通知取得

```
/**
    温度監視
    イベント取得サンプルソース
**/
#include <windows.h>
#include <winioctl.h>
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>

#include "..¥Common¥G5RasDef.h"

#define THREADSTATE_STOP        0
#define THREADSTATE_RUN         1
#define THREADSTATE_QUERY_STOP  2

#define MAX_EVENT   3

enum {
    EVENT_FIN = 0,
    EVENT_USER_EXT,
    EVENT_USER_CPU
};

HANDLE hEvent[MAX_EVENT];
HANDLE hThread;
ULONG ThreadState;

DWORD WINAPI EventThread(void *pData)
{
    DWORD ret;

    printf("EventThread: Start¥n");
    ThreadState = THREADSTATE_RUN;

    /*
     * 温度監視 ユーザーイベントを待ちます
     */
    while(1){
        ret = WaitForMultipleObjects(MAX_EVENT, &hEvent[0], FALSE, INFINITE);
        if(ret == WAIT_FAILED){
            break;
        }
        if(ThreadState == THREADSTATE_QUERY_STOP){
```

```
            break;
        }

        if(ret == WAIT_OBJECT_0 + EVENT_USER_EXT){
            printf("EventThread: Ext Temperature UserEvent¥n");
        }
        if(ret == WAIT_OBJECT_0 + EVENT_USER_CPU){
            printf("EventThread: CPU Temperature UserEvent¥n");
        }
    }
    ThreadState = THREADSTATE_STOP;

    printf("EventThread: Finish¥n");
    return 0;
}


int main(int argc, char **argv)
{
    DWORD thid;
    int keych;
    int i;

    /*
     * スレッド終了用イベント
     */
    hEvent[EVENT_FIN] = CreateEvent(NULL, FALSE, FALSE, NULL);

    /*
     * 温度監視 ユーザーイベント ハンドル取得
     */
    hEvent[EVENT_USER_EXT]=OpenEvent(SYNCHRONIZE, FALSE, EXT_TEMPERATURE_USER_EVENT_NAME);
    if(hEvent[EVENT_USER_EXT] == NULL){
        printf("CreateEvent: NG¥n");
        return -1;
    }
    hEvent[EVENT_USER_CPU]=OpenEvent(SYNCHRONIZE, FALSE, CPU_TEMPERATURE_USER_EVENT_NAME);
    if(hEvent[EVENT_USER_CPU] == NULL){
        printf("CreateEvent: NG¥n");
        return -1;
    }

    /*
     * 温度監視 ユーザーイベント取得スレッド ハンドル取得
     */
    ThreadState = THREADSTATE_STOP;
    hThread = CreateThread(
                    (LPSECURITY_ATTRIBUTES)NULL,
                    0,
                    (LPTHREAD_START_ROUTINE)EventThread,
                    NULL,
                    0,
```

```
                        &thid
                    );
    if(hThread == NULL){
        CloseHandle(hEvent);
        printf("CreateThread: NG¥n");
        return -1;
    }

    /*
     * 'Q'または'q'キーで終了します。
     */
    while(1){
        if(kbhit()){
            keych = getch();
            if(keych == 'Q' || keych == 'q'){
                break;
            }
        }
    }

    /*
     * スレッドを終了
     */
    ThreadState = THREADSTATE_QUERY_STOP;
    SetEvent(hEvent[EVENT_FIN]);
    while(ThreadState != THREADSTATE_STOP){
        Sleep(10);
    }
    CloseHandle(hThread);
    for(i = 0; i < MAX_EVENT; i++){
        CloseHandle(hEvent[i]);
    }

    return 0;
}
```

５－１０　外部 RTC 機能

５－１０－１ 外部 RTC 機能について

FP シリーズには、外部 RTC（Real Time Clock）が実装されています。外部 RTC サービスにより、外部 RTC と CPU 内部 RTC（システム時刻）を同期させることができます。

また、Wake On Rtc Timer 機能を使用して、目的の時間にコンピュータの電源の起動を行うことができます。（詳細は、「2-13-7 Secondary RTC Configuration」を参照してください。）

また、DLL を介してユーザーアプリケーションから外部 RTC、CPU 内部 RTC の日時設定、及び Wake On Rtc Timer の設定を行えるようにします。

５－１０－２ RAS DLL について

外部 RTC サービスによって外部 RTC と CPU 内部 RTC の同期を行っている場合、日時設定は外部 RTC と CPU 内部 RTC を同時に設定する必要があります。RAS DLL（G5RAS.dll）は、ユーザーアプリケーションから日時設定（外部 RTC、CPU 内部 RTC 同時設定）を行えるようにします。また、Wake On Rtc Timer の設定も行えるようにします。

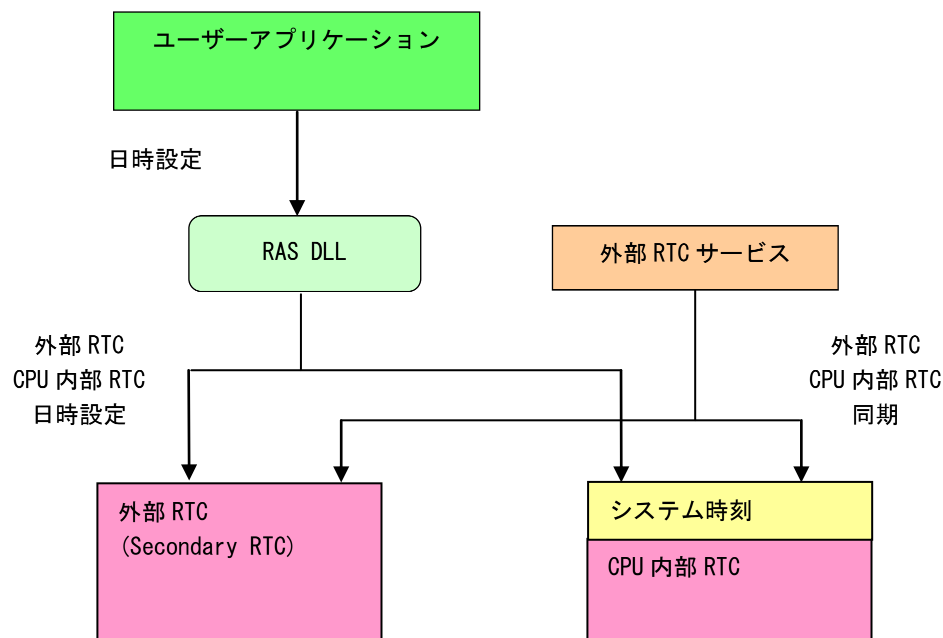

図 5-10-2-1. RAS DLL 日時設定

５－１０－３ RAS DLL 時刻設定関数リファレンス

G5_SetLocalTime 関数

機能　ローカル日時を使用して日時設定を行います。

書式　BOOL　G5_SetLocalTime(SYSTEMTIME *lpSystemTime)

引数　lpSystemTime　：設定するローカル日時を格納するポインタ。

戻り値　TRUE　：正常
FALSE　：エラー

説明　ローカル日時を使用して日時設定を行います。
外部 RTC と CPU 内部 RTC を同時に設定を行います。外部 RTC サービスで外部 RTC と CPU 内部 RTC を同期させているときは、この関数を使用して日時設定を行ってください。

G5_SetSystemTime 関数

機能　システム日時を使用して日時設定を行います。

書式　BOOL　G5_SetSystemTime(SYSTEMTIME *lpSystemTime)

引数　lpSystemTime　：設定するシステム日時を格納するポインタ。

戻り値　TRUE　：正常
FALSE　：エラー

説明　システム日時を使用して日時設定を行います。システム日時は、世界協定時刻（UTC）で表されます。
外部 RTC と CPU 内部 RTC を同時に設定を行います。外部 RTC サービスで外部 RTC と CPU 内部 RTC を同期させているときは、この関数を使用して日時設定を行ってください。

５－１０－４ RAS DLL Wake On Rtc Timer 設定関数リファレンス

G5_GetWakeOnRtcTime 関数

機能　　Wake On Rtc Timer の設定を読み出します。

書式　　BOOL　G5_GetWakeOnRtcTime(PG8WakeOnRtc pWakeOnRtc)

引数　　pWakeOnRtc　：設定するシステム日時を格納するポインタ。

Wake On Rtc Timer 設定情報

```
typedef struct {
    int at_min;
    int at_hour;
    int at_day;
    int at_week;
    int at_flag;
} G8WakeOnRtc, *PG8WakeOnRtc;
```

at_min　：Wake On Rtc タイマ「分」

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
未使用	40	20	10	8	4	2	1

at_hour　：Wake On Rtc タイマ「時」

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
未使用	－	20	10	8	4	2	1

at_day　：Wake On Rtc タイマ「日」

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
－	－	20	10	8	4	2	1

at_week　：Wake On Rtc タイマ「曜日」

Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
土	金	木	水	火	月	日

at_flag　：設定フラグ

Bit4	Bit3	Bit2	Bit1	Bit0
－	曜日	日	－	－

戻り値　　TRUE　：正常
FALSE　：エラー

説明　　現在の Wake On Rtc Timer の設定情報を読み出します。
「時」・「分」の Bit7 が有効な場合は、「時」もしくは「分」が未使用となります。

G5_SetWakeOnRtcTime 関数

機能　Wake On Rtc Timer を設定します。

書式　BOOL　G5_SetWakeOnRtcTime(PG8WakeOnRtc pWakeOnRtc)

引数　pWakeOnRtc　：設定するシステム日時を格納するポインタ。

Wake On Rtc Timer 設定情報

```
typedef struct {
    int at_min;
    int at_hour;
    int at_day;
    int at_week;
    int at_flag;
} G8WakeOnRtc, *PG8WakeOnRtc;
```

at_min　：Wake On Rtc タイマ「分」

Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
40	20	10	8	4	2	1

at_hour　：Wake On Rtc タイマ「時」

Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
20	10	8	4	2	1

at_day　：Wake On Rtc タイマ「日」

Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
20	10	8	4	2	1

at_week　：Wake On Rtc タイマ「曜日」

Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
土	金	木	水	火	月	日

at_flag　：設定フラグ

Bit4	Bit3	Bit2	Bit1	Bit0
ｸﾘｱ	曜日	日	時	分

戻り値　TRUE　：正常
FALSE　：エラー

説明　Wake On Rtc Timer を設定します。
設定フラグは使用したいパラメータを有効にします。ただし、「曜日」もしくは「日」はどちらか片方しか設定することができません。
かならずどちらか片方を設定してください。
また、「時」もしくは「日」は両方設定することは可能です。

G5_ClrWakeOnRtcTime 関数

機能　Wake On Rtc Timer の設定を解除します。

書式　BOOL　G5_ClrWakeOnRtcTime(void)

引数　なし

戻り値　TRUE　：正常
FALSE　：エラー

説明　現在の Wake On Rtc Timer の設定を解除します。

５－１０－５ サンプルコード

「¥SDK¥Algo¥Sample¥Sample_RASDll¥SecondaryRTC」に RAS DLL を使用して日時設定を行うサンプルコードを用意しています。

●ローカル日時を使用した日時設定

「¥SDK¥Algo¥Sample¥Sample_RASDll¥SecondaryRTC¥SetLocalTime.cpp」は、ローカル日時を使用して日時設定を行うサンプルコードです。リスト 5-10-5-1 にサンプルコードを示します。

リスト 5-10-5-1. ローカル日時を使用した日時設定

```
#include <windows.h>
#include <winioctl.h>
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>

#include "..¥Common¥G5RasDef.h"
#include "..¥Common¥AlgG5Ras.h"


int main(int argc, char **argv)
{
    int tm_year;
    int tm_mon;
    int tm_mday;
    int tm_hour;
    int tm_min;
    int tm_sec;
    SYSTEMTIME systm;

    if(argc != 2){
        printf("Usage: SetLocalTime <yyyy:mm:dd:HH:MM:SS>¥n");
        return -1;
    }
    sscanf(*(argv + 1), "%d:%d:%d:%d:%d:%d",
        &tm_year, &tm_mon, &tm_mday, &tm_hour, &tm_min, &tm_sec);

    printf("G5_SetLocalTime: %04d:%02d:%02d %02d:%02d:%02d¥n",
        tm_year, tm_mon, tm_mday, tm_hour, tm_min, tm_sec);

    // DLL ロード
    LoadG5RasDll("G5RAS.dll");

    // SystemTime 設定
    systm.wYear = tm_year;
    systm.wMonth = tm_mon;
    systm.wDay = tm_mday;
    systm.wHour = tm_hour;
    systm.wMinute = tm_min;
    systm.wSecond = tm_sec;
    systm.wMilliseconds = 0;
    if(!G5_SetLocalTime(&systm)){
```

```
        printf("G5_SetLocalTime: NG¥n");
        UnloadG5RasDll();
        return -1;
    }

    // GetLocalTime
    memset(&systm, 0x00, sizeof(SYSTEMTIME));
    GetLocalTime(&systm);
    printf("GetLocalTime: %04d:%02d:%02d %02d:%02d:%02d.%03d¥n",
        systm.wYear, systm.wMonth, systm.wDay, systm.wHour, systm.wMinute, systm.wSecond,
systm.wMilliseconds);

    UnloadG5RasDll();

    return 0;
}
```

●システム日時を使用した日時設定

「¥SDK¥Algo¥Sample¥Sample_RASDll¥SecondaryRTC¥SetLocalTime.cpp」は、システム日時を使用して日時設定を行うサンプルコードです。リスト 5-10-5-2 にサンプルコードを示します。

リスト 5-10-5-2. システム日時を使用した日時設定

```
#include <windows.h>
#include <winioctl.h>
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>

#include "..¥Common¥G5RasDef.h"
#include "..¥Common¥AlgG5Ras.h"


int main(int argc, char **argv)
{
    int tm_year;
    int tm_mon;
    int tm_mday;
    int tm_hour;
    int tm_min;
    int tm_sec;
    SYSTEMTIME systm;

    if(argc != 2){
        printf("Usage: SetSystemTime <yyyy:mm:dd:HH:MM:SS>¥n");
        return -1;
    }
    sscanf(*(argv + 1), "%d:%d:%d:%d:%d:%d",
        &tm_year, &tm_mon, &tm_mday, &tm_hour, &tm_min, &tm_sec);

    printf("G5_SetSystemTime: %04d:%02d:%02d %02d:%02d:%02d¥n",
        tm_year, tm_mon, tm_mday, tm_hour, tm_min, tm_sec);

    //  DLL ロード
    LoadG5RasDll("G5RAS.dll");

    // SystemTime 設定
    systm.wYear = tm_year;
    systm.wMonth = tm_mon;
    systm.wDay = tm_mday;
    systm.wHour = tm_hour;
    systm.wMinute = tm_min;
    systm.wSecond = tm_sec;
    systm.wMilliseconds = 0;
    if(!G5_SetSystemTime(&systm)){
        printf("G5_SetSystemTime: NG¥n");
        UnloadG5RasDll();
        return -1;
    }
```

```
    // GetSystemTime
    memset(&systm, 0x00, sizeof(SYSTEMTIME));
    GetSystemTime(&systm);
    printf("GetSystemTime: %04d:%02d:%02d %02d:%02d:%02d.%03d¥n",
        systm.wYear, systm.wMonth, systm.wDay, systm.wHour, systm.wMinute, systm.wSecond,
systm.wMilliseconds);

    UnloadG5RasDll();

    return 0;
}
```

●Wake On Rtc Timer 設定

「¥SDK¥Algo¥Sample¥Sample_RASDll¥SecondaryRTC¥SetWakeOnRtcTime.cpp」は、Wake On Rtc Timer 設定を行うサンプルコードです。リスト 5-10-5-3 にサンプルコードを示します。

リスト 5-10-5-3. Wake On Rtc Timer 設定

```
#include <windows.h>
#include <winioctl.h>
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>

#include "..¥Common¥G5RasDef.h"
#include "..¥Common¥AlgG5Ras.h"


int main(int argc, char **argv)
{
    int at_week;
    int at_day;
    int at_hour;
    int at_min;
    int at_flag;
    G8WakeOnRtc wakeontm;

    if(argc != 2){
        printf("Usage: SetWakeOnRtcTime <week:day:hour:min:flag>¥n");
        printf("       week: 1:Sunday    2:Monday   4:Tuesday   8:Wednesday ¥n");
        printf("             16:Thursday 32:Friday 64:Saturday ¥n");
        printf("       day : 1 .. 31 ¥n");
        printf("       hour: 0 .. 23 ¥n");
        printf("       min : 0 .. 59 ¥n");
        printf("       flag: 1:Enable Min 2:Enable Hour 4:Enable Day 8:Enable Week ¥n");
        printf("             16:Disable Wake On Rtc ¥n");
        return -1;
    }
    sscanf(*(argv + 1), "%d:%d:%d:%d:%d",
        &at_week, &at_day, &at_hour, &at_min, &at_flag);

    printf("G5_SetWakeOnRtcTime: %d:%d:%d:%d:%d¥n",
```

```
        at_week, at_day, at_hour, at_min, at_flag);

    //  DLL ロード
    LoadG5RasDll("G5RAS.dll");

    // ClrWakeOnRtcTime
    if(!G5_ClrWakeOnRtcTime()){
        printf("G5_ClrWakeOnRtcTime: NG¥n");
        UnloadG5RasDll();
        return -1;
    }
    printf("G5_ClrWakeOnRtcTime: OK¥n");

    // SetWakeOnRtcTime
    wakeontm.at_week = at_week;
    wakeontm.at_day  = at_day;
    wakeontm.at_hour = at_hour;
    wakeontm.at_min  = at_min;
    wakeontm.at_flag = at_flag;
    if(!G5_SetWakeOnRtcTime(&wakeontm)){
        printf("G5_SetWakeOnRtcTime: NG¥n");
        UnloadG5RasDll();
        return -1;
    }
    printf("G5_SetWakeOnRtcTime: OK¥n");

    // GetWakeOnRtcTime
    memset(&wakeontm, 0x00, sizeof(G8WakeOnRtc));
    if(!G5_GetWakeOnRtcTime(&wakeontm)){
        printf("G5_GetWakeOnRtcTime: NG¥n");
        UnloadG5RasDll();
        return -1;
    }
    printf("G5_GetWakeOnRtcTime: %02x:%02d:%02d:%02d:%02x¥n",
        wakeontm.at_week, wakeontm.at_day, wakeontm.at_hour & 0x7F, wakeontm.at_min & 0x7F,
wakeontm.at_flag);

    UnloadG5RasDll();

    return 0;
}
```

５－１１　停電検出機能

５－１１－１ 停電検出機能について

FP シリーズには､ハードウェアによる停電検出機能が実装されています｡停電が発生したかどうかの判別、および停電発生時のタイムスタンプ取得を行うことができます。

５－１１－２ 停電検出ドライバについて

停電検出ドライバは停電検出機能を、ユーザーアプリケーションから利用できるようにします。
ユーザーアプリケーションからタイムスタンプ取得、タイムスタンプクリアを行うことができます。

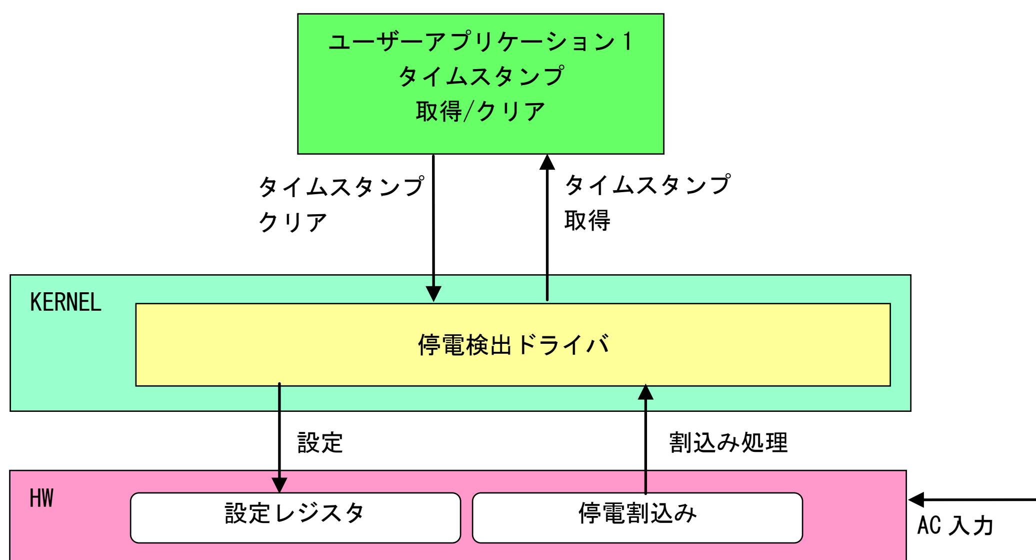

図 5-11-2-1. 停電検出ドライバ

５－１１－３ 停電検出デバイス

停電検出ドライバは停電検出デバイスを生成します。ユーザーアプリケーションは、デバイスファイルにアクセスすることによって停電検出機能を操作します。

停電検出デバイス

デバイスファイル

¥¥.¥PowerDownDetect

説明

停電発生時のタイムスタンプの取得、クリアを行うことができます。

CreateFile

デバイスファイル(¥¥.¥PowerDownDetect)をオープンし、デバイスハンドルを取得します。

```
hPdd = CreateFile(
        ”¥¥¥¥.¥¥PowerDownDetect”,
        GENERIC_READ | GENERIC_WRITE,
        FILE_SHARE_READ | FILE_SHARE_WRITE,
        NULL,
        OPEN_EXISTING,
        0,
        NULL
    );
```

CloseHandle

デバイスハンドルをクローズします。

```
CloseHandle(hPdd);
```

ReadFile

使用しません。

WriteFile

使用しません。

DeviceIoControl

- **IOCTL_POWERDOWNDETECT_CLRTIME**
  タイムスタンプのクリアを行います。
- **IOCTL_POWERDOWNDETECT_GETTIME**
  タイムスタンプの取得を行います。

５－１１－４ DeviceIoControl リファレンス

IOCTL_POWERDOWNDETECT_CLRTIME

機能

停電検出時に保存されるタイムスタンプのクリアを行います。

パラメータ

lpInBuf	:	NULL を指定します。
nInBufSize	:	0 を指定します。
lpOutBuf	:	NULL を指定します。
nOutBufSize	:	0 を指定します。
lpBytesReturned	:	実際の出力バイト数を受け取る変数へのポインタを指定します。
lpOverlapped	:	NULL を指定します。

戻り値

処理が成功すると TRUE を返します。失敗の場合は FALSE を返します。

説明

停電検出時に保存されるタイムスタンプのクリアを行います。
クリア後は「1601 年　1 月　1 日　9 時　0 分　0 秒」にクリアされます。

IOCTL_POWERDOWNDETECT_GETTIME

機能

停電検出時に保存されるタイムスタンプを取得します。

パラメータ

lpInBuf	:	タイムスタンプ情報を格納するポインタを指定します。
nInBufSize	:	タイムスタンプ情報を格納するポインタのサイズを指定します。
lpOutBuf	:	NULL を指定します。
nOutBufSize	:	0 を指定します。
lpBytesReturned	:	実際の出力バイト数を受け取る変数へのポインタを指定します。
lpOverlapped	:	NULL を指定します。

タイムスタンプ情報

```
typedef struct {
    LARGE_INTEGER   time[32];
} PDD_MEMTIME, *PPDD_MEMTIME;
```

time　　: タイムスタンプ
停電が発生した時間を 32 回分、LARGE_INTEGER に値を代入します。

戻り値

処理が成功すると TRUE を返します。失敗の場合は FALSE を返します。

説明

停電検出時に保存されるタイムスタンプを取得します。
LARGE_INTEGER で保存されいますので、システム時間に変更する場合は FileTimeToSystemTime()等の API 関数を使用してください。

５－１１－５ サンプルコード

●停電検出サンプル

「¥SDK¥Algo¥Sample¥Sample_Pdd¥ PddInterrupt」に停電検出のタイムスタンプ取得/クリアのサンプルコードを用意しています。リスト 5-11-5-1 にサンプルコードを示します。

リスト 5-11-5-1. 停電検出のタイムスタンプ取得/クリア

```
/**
    停電検出制御サンプルソース
**/
#include <windows.h>
#include <winioctl.h>
#include <stdio.h>
#include <conio.h>

#include "..¥Common¥PowerDownDetectDD.h"

#define PDDDRIVER_FILENAME  "¥¥¥¥.¥¥PowerDownDetect"

//---------------------------------------------------------------------------
typedef struct {
    HANDLE  hPdd;
} PDDEVENT_INFO, *PPDDEVENT_INFO;

//---------------------------------------------------------------------------
BOOL CreatePddInfo(PPDDEVENT_INFO info)
{
    info->hPdd = INVALID_HANDLE_VALUE;

    /*
     * ドライバオブジェクトの作成
     */
    info->hPdd = CreateFile(
                        PDDDRIVER_FILENAME,
                        GENERIC_READ | GENERIC_WRITE,
                        FILE_SHARE_READ | FILE_SHARE_WRITE,
                        NULL,
                        OPEN_EXISTING,
                        0,
                        NULL
                        );
    if(info->hPdd == INVALID_HANDLE_VALUE){
        printf("CreatePddInfo: CreateFile: NG¥n");
        return FALSE;
    }

    return TRUE;
}

//---------------------------------------------------------------------------
void DeletePddInfo(PPDDEVENT_INFO info)
```

```
{
    // Close Handle
    CloseHandle(info->hPdd);
}

//----------------------------------------------------------------------------
/*
 * Time Stamp の取得
 */
BOOL Get_Time(HANDLE hdriver, PDD_MEMTIME *pddtime)
{
    BOOL    ret;
    ULONG   retlen;

    // Get Time Stamp
    ret = DeviceIoControl(
                hdriver,
                IOCTL_POWERDOWNDETECT_GETTIME,
                NULL,
                0,
                pddtime,
                sizeof(PDD_MEMTIME),
                &retlen,
                NULL
                );

    return ret;
}
//----------------------------------------------------------------------------

/*
 * Time Stamp のクリア
 */
BOOL Clear_Time(HANDLE hdriver)
{
    BOOL    ret;
    ULONG   retlen;

    // Clear Time Stamp
    ret = DeviceIoControl(
                hdriver,
                IOCTL_POWERDOWNDETECT_CLRTIME,
                NULL,
                0,
                NULL,
                0,
                &retlen,
                NULL
                );

    return ret;
}
```

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

int main(void)
{
    int     i;
    int     c;
    BOOL    ret;
    DWORD   err;
    PDDEVENT_INFO   info;
    PDD_MEMTIME pddtime;
    SYSTEMTIME  stTime;

    /*
     * ドライバオブジェクトの作成
     */
    if( !CreatePddInfo(&info) ){
        printf("CreatePddInfo: NG¥n");
        return -1;
    }

    /*
     * 停電発生時のタイムスタンプを取得
     */
    ret = Get_Time(info.hPdd, &pddtime);
    if( !ret ){
        err = GetLastError();
        fprintf(stderr, "ioctl get time error: %d¥n", err);
        DeletePddInfo(&info);
        return -1;
    }
    else {
        for(i=0; i<32; i++){
            ret = FileTimeToSystemTime((FILETIME*)&pddtime.time[i], &stTime);
            if(!ret){
                err = GetLastError();
                fprintf(stderr, "ioctl get time error: %d¥n", err);
                DeletePddInfo(&info);
                return -1;
            }
            fprintf(stdout, "%d 年 %d 月 %d 日 %d 時 %d 分 %d 秒 ¥n",
                stTime.wYear, stTime.wMonth, stTime.wDay, stTime.wHour, stTime.wMinute,
stTime.wSecond);
        }
    }

    /*
     * タイムスタンプを初期化
     */
    ret = Clear_Time(info.hPdd);
    if( !ret ){
        err = GetLastError();
        fprintf(stderr, "ioctl clr time error: %d¥n", err);
```

```
        DeletePddInfo(&info);
        return -1;
    }
    else {
        fprintf(stdout, "ioctl clear time success ¥n");
    }

    while(1){
        if( kbhit() ){
            c = getch();
            if(c == 'q' || c == 'Q')
                break;
        }
    }

    /*
     * ドライバオブジェクトの破棄
     */
    DeletePddInfo(&info);

    return 0;
}

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

５－１２　バックアップバッテリモニタ

５－１２－１ バックアップバッテリモニタについて

FP シリーズは、BIOS、RTC、外部 RTC、SRAM のデータを保持するためにバックアップバッテリを搭載しています。バックアップバッテリモニタレジスタを参照することによって、バックアップバッテリの状態(正常・低下)を確認することができます。

５－１２－２ バックアップバッテリモニタドライバについて

バックアップバッテリモニタドライバはバックアップバッテリの状態を、ユーザーアプリケーションから取得できるようにします。

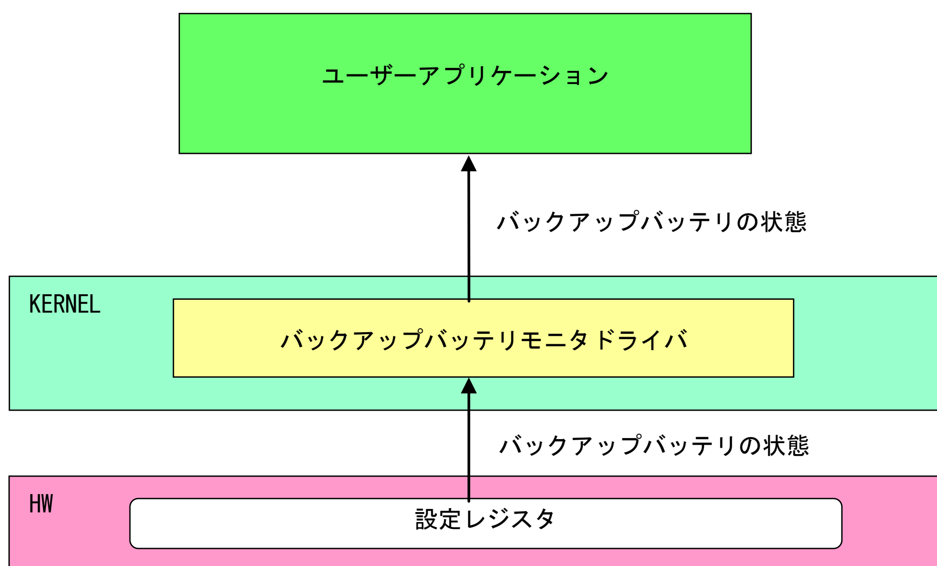

図 5-12-2-1. バックアップバッテリモニタドライバ

５－１２－３　バックアップバッテリモニタデバイス

　バックアップバッテリモニタドライバはバックアップバッテリモニタデバイスを生成します。ユーザーアプリケーションは、デバイスファイルにアクセスすることによってバックアップバッテリの状態を取得します。

バックアップバッテリモニタデバイス

デバイスファイル

¥¥.¥BackBatDrv

説明

バックアップバッテリの状態(正常・低下)を取得することができます。

CreateFile

デバイスファイル(¥¥.¥BackBatDrv)をオープンし、デバイスハンドルを取得します。

```
hBackBat = CreateFile(
    "¥¥¥¥.¥¥BackBatDrv",
    GENERIC_READ | GENERIC_WRITE,
    FILE_SHARE_READ | FILE_SHARE_WRITE,
    NULL,
    OPEN_EXISTING,
    0,
    NULL
);
```

CloseHandle

デバイスハンドルをクローズします。

```
CloseHandle(hBackBat);
```

ReadFile

使用しません。

WriteFile

使用しません。

DeviceIoControl

● **IOCTL_BACKBATDRV_GETSTAT**

　バックアップバッテリの状態を取得します。

５－１２－４ DeviceIoControl リファレンス

IOCTL_BACKBATDRV_GETSTAT

機能

バックアップバッテリの状態を取得します。

パラメータ

lpInBuf　：　NULL を指定します。
NInBufSize　：　0 を指定します。
LpOutBuf　：　バックアップバッテリ状態を格納するポインタを指定します。
NOutBufSize　：　バックアップバッテリ状態を格納するポインタのサイズを指定します。
LpBytesReturned　：　実際の出力バイト数を受け取る変数へのポインタ。
LpOverlapped　：　NULL を指定します。

バックアップバッテリ状態

データタイプ	:	ULONG
データサイズ	:	4 バイト
内容	:	0: 正常、1: 低下

戻り値

処理が成功すると TRUE を返します。失敗の場合は FALSE を返します。

説明

バックアップバッテリ状態を取得します。
バックアップバッテリ状態は、「正常」・「低下」を確認できます。

５－１２－５ サンプルコード

●バックアップバッテリ状態取得

「¥SDK¥Algo¥Sample¥Sample_BackBat¥BackBatStatus」にバックアップバッテリ状態取得のサンプルコードを用意しています。リスト 5-12-5-1 にサンプルコードを示します。

リスト 5-12-5-1. バックアップバッテリ状態取得

```
/**
    バックアップバッテリモニタサンプルソース
**/

#include <windows.h>
#include <winioctl.h>
#include <stdio.h>
#include <stdlib.h>
#include <mmsystem.h>
#include <conio.h>

#include "..¥Common¥BackBatDD.h"

#define DRIVER_FILENAME "¥¥¥¥.¥¥BackBatDrv"

BOOL GetBackBatStatus(HANDLE hDevice, ULONG *pStatus)
{
    BOOL    ret;
    ULONG   status;
    ULONG   retlen;

    ret = DeviceIoControl(hDevice,
                          IOCTL_BACKBATDRV_GETSTAT,
                          NULL,
                          0,
                          &status,
                          sizeof(ULONG),
                          &retlen,
                          NULL);
    if(!ret){
        return FALSE;
    }
    if(retlen != sizeof(ULONG)){
        return FALSE;
    }

    *pStatus = status;
    return TRUE;
}

int main(int argc, char **argv)
{
    HANDLE  h_backbat;
```

```
    BOOL    ret;
    ULONG   status;

    /* デバイスのオープン */
    h_backbat = CreateFile(
                    DRIVER_FILENAME,
                    GENERIC_READ | GENERIC_WRITE,
                    FILE_SHARE_READ | FILE_SHARE_WRITE,
                    NULL,
                    OPEN_EXISTING,
                    0,
                    NULL
                );
    if(h_backbat == INVALID_HANDLE_VALUE){
        printf("CreateFile: NG¥n");
        return -1;
    }

    /*
        バックアップバッテリ状態取得
    */
    ret = GetBackBatStatus(h_backbat, &status);
    if(!ret){
        printf("DeviceIoControl: IOCTL_BACKBATDRV_GETSTAT NG¥n");
        CloseHandle(h_backbat);
        return -1;
    }

    printf("Backup Battery Status: %d¥n", status);

    /* デバイスのクローズ */
    CloseHandle(h_backbat);

    return 0;
}
```

第６章　システムリカバリ

本章では、「FP シリーズ用 Windows Embedded Standard 7 32 ビット版 リカバリ/SDK DVD」を使用したシステムのリカバリとバックアップについて説明します。

６－１　リカバリ DVD について

FP シリーズ本体を「FP シリーズ用 Windows Embedded Standard 7 32 ビット版 リカバリ/SDK DVD」（以降、「リカバリ DVD」と表記します。）で起動させると、システムのリカバリを行うことができます。リカバリで行える処理は以下のとおりです。

- ● システムの復旧（工場出荷状態）
- ● システムのバックアップ
- ● システムの復旧（バックアップデータ）

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

● リカバリ DVD 使用の流れ

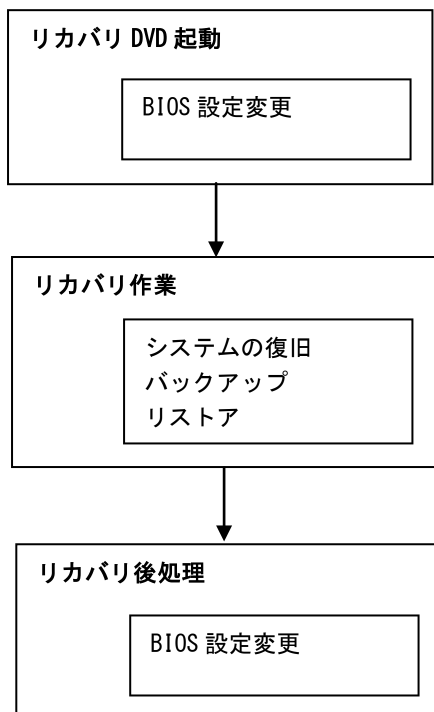

図 5-12-5-1. リカバリ DVD 使用の流れ

６－１－１　リカバリ DVD 起動

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

リカバリ DVD の起動には、USB 接続の DVD ドライブ、USB キーボード、USB マウスが必要です。

● リカバリ DVD 起動手順

① LAN ケーブルが接続されている場合は、LAN ケーブルを取り外してください。
② USB メモリ、SD カードなどのストレージメディアが接続されている場合は、ストレージメディアを取り外してください。また、サブストレージ（mSATA2）が接続されている場合は、サブストレージを取り外してください。
③ USB-DVD ドライブにリカバリ DVD を入れ FP シリーズ本体に接続します。
④ USB キーボード、USB マウスを接続します。
⑤ 電源を入れます。BIOS 起動画面が表示されたところで[F2]キーを押し、BIOS 設定画面を表示させます。
⑥ BIOS 設定画面が表示されたら、[Advanced]メニューを選択します。(図 6-1-2)
⑦ [OS Selection]を[Linux]、[BIOS WDT]を[Disabled]に設定します。

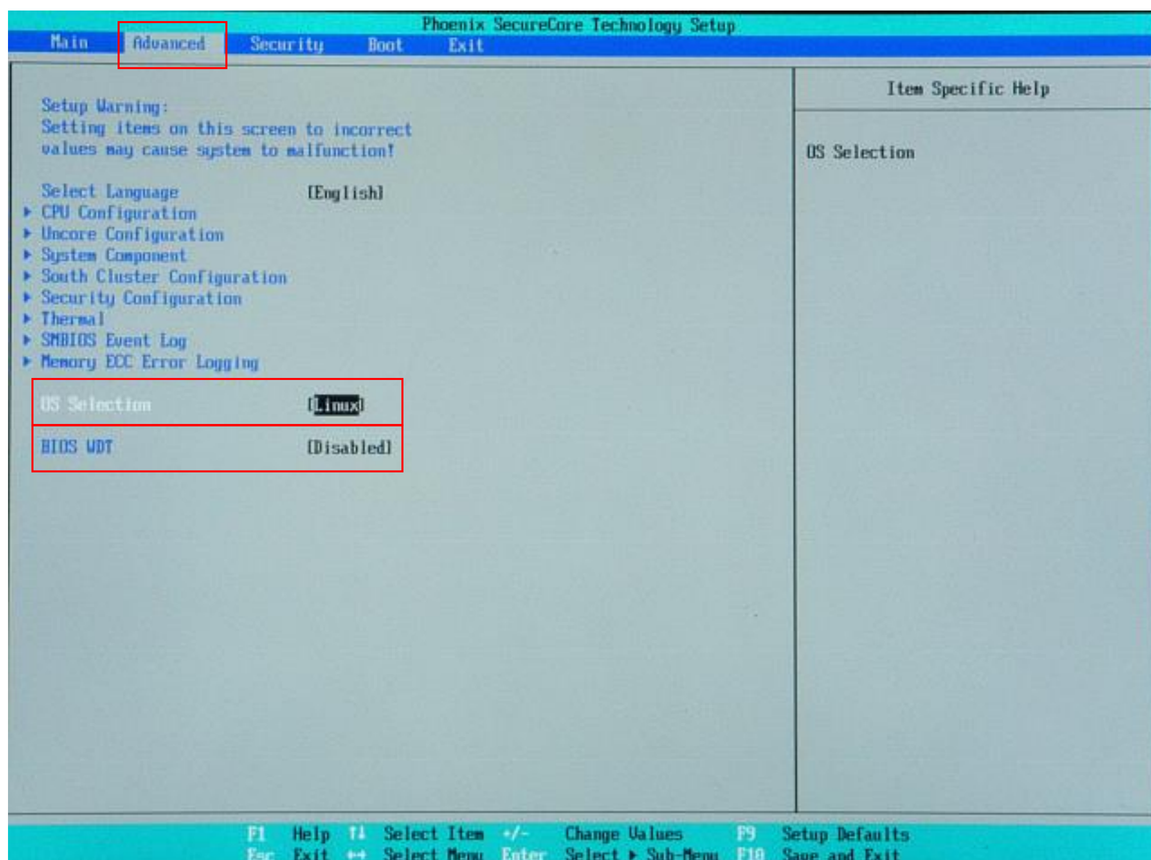

図 6-1-2. BIOS 設定 Advanced メニュー

⑧ [Boot]メニューを選択します。(図 6-1-3)
⑨ [USB CD](接続した USB-DVD ドライブ)を[ATA HDD0](m-SATA メインストレージ)よりも上に設定します。

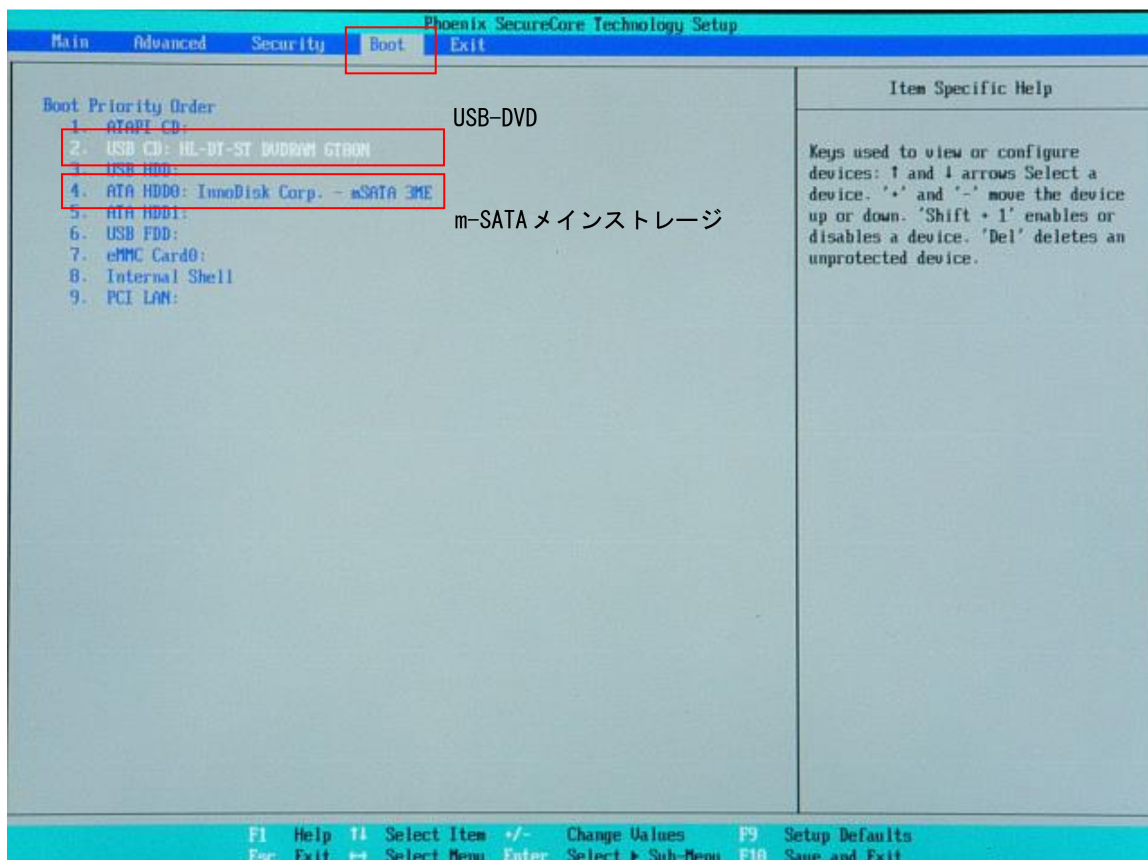

図 6-1-3. Boot デバイスの選択

⑩　[Exit]メニューを選択します。(図 6-1-4)
⑪　[Exit Saving Changes]を実行し、設定を保存して終了します。

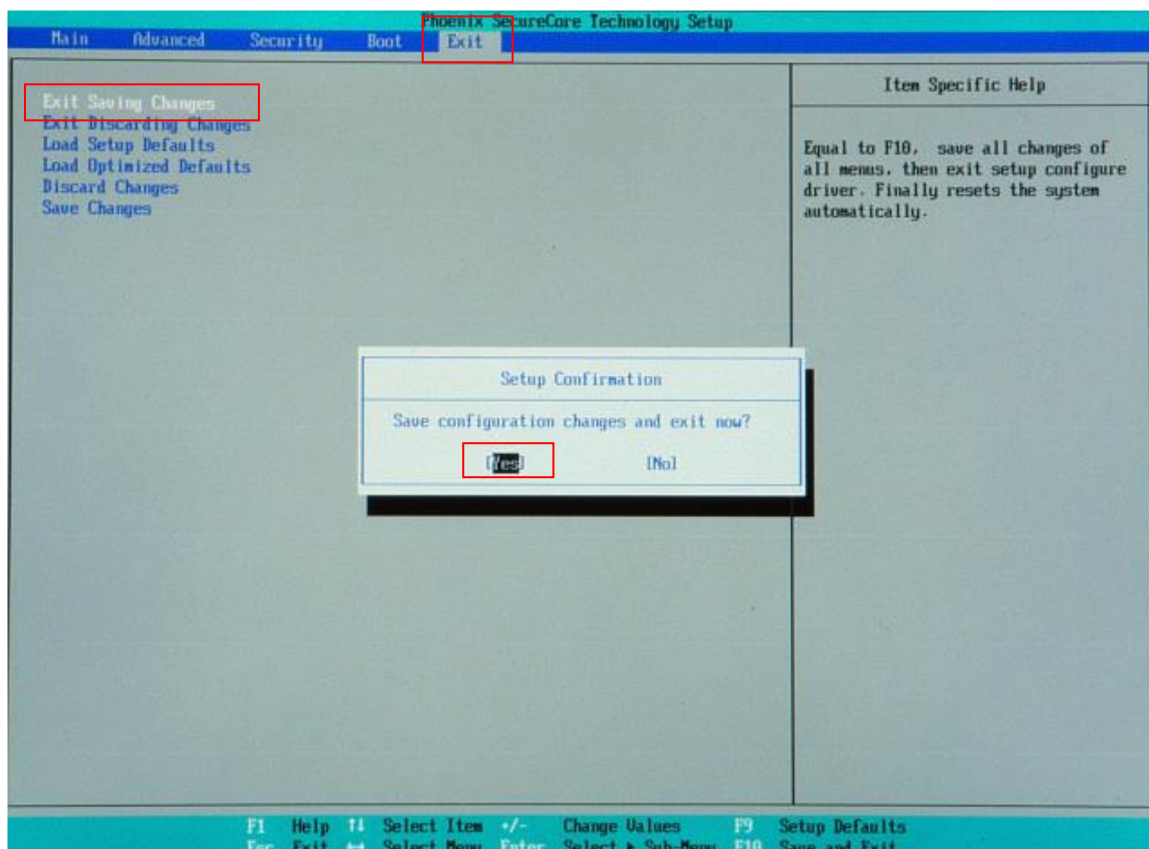

図 6-1-4. BIOS 設定 保存と終了

⑫　再起動し、正常にリカバリ DVD から起動すると図 6-1-5 のリカバリメイン画面が表示されます。

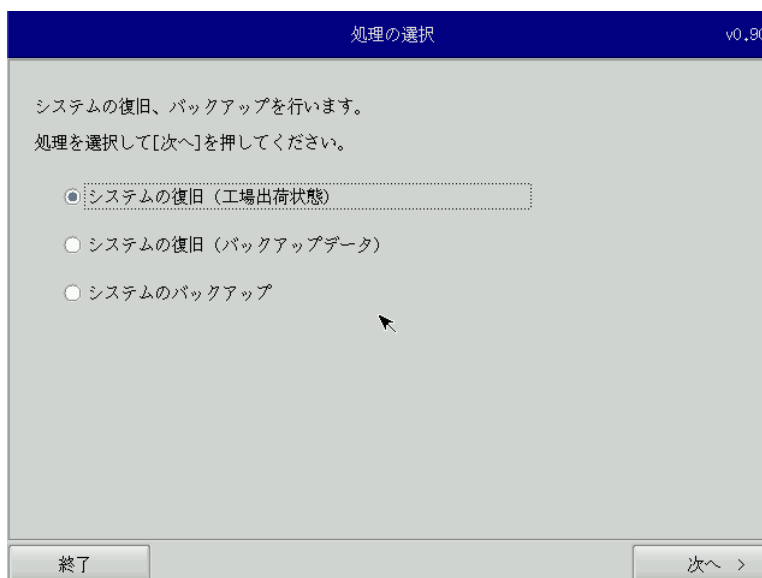

図 6-1-5. リカバリメイン画面

６－１－２　　リカバリ作業

リカバリメイン画面から処理を選んでリカバリ作業を行います。

- システムの復旧（工場出荷状態）
- システムのバックアップ
- システムの復旧（バックアップデータ）

リカバリ作業の詳細は、「6-2 システムの復旧（工場出荷状態）」、「6-3 システムのバックアップ」、「6-4 システムの復旧（バックアップデータ）」を参照してください。

６－１－３　リカバリ後処理

リカバリ作業が終わったら、通常使用のために BIOS 設定を戻します。

● リカバリ後処理手順

① 電源を入れ、BIOS 起動画面が表示されたところで[F2]キーを押し、BIOS 設定画面を表示させせます。
② BIOS 設定画面が表示されたら、[Advanced]メニューを選択します。(図 6-1-2)
③ [OS Selection]を[Windows]、[BIOS WDT]を[Enabled]に設定します。

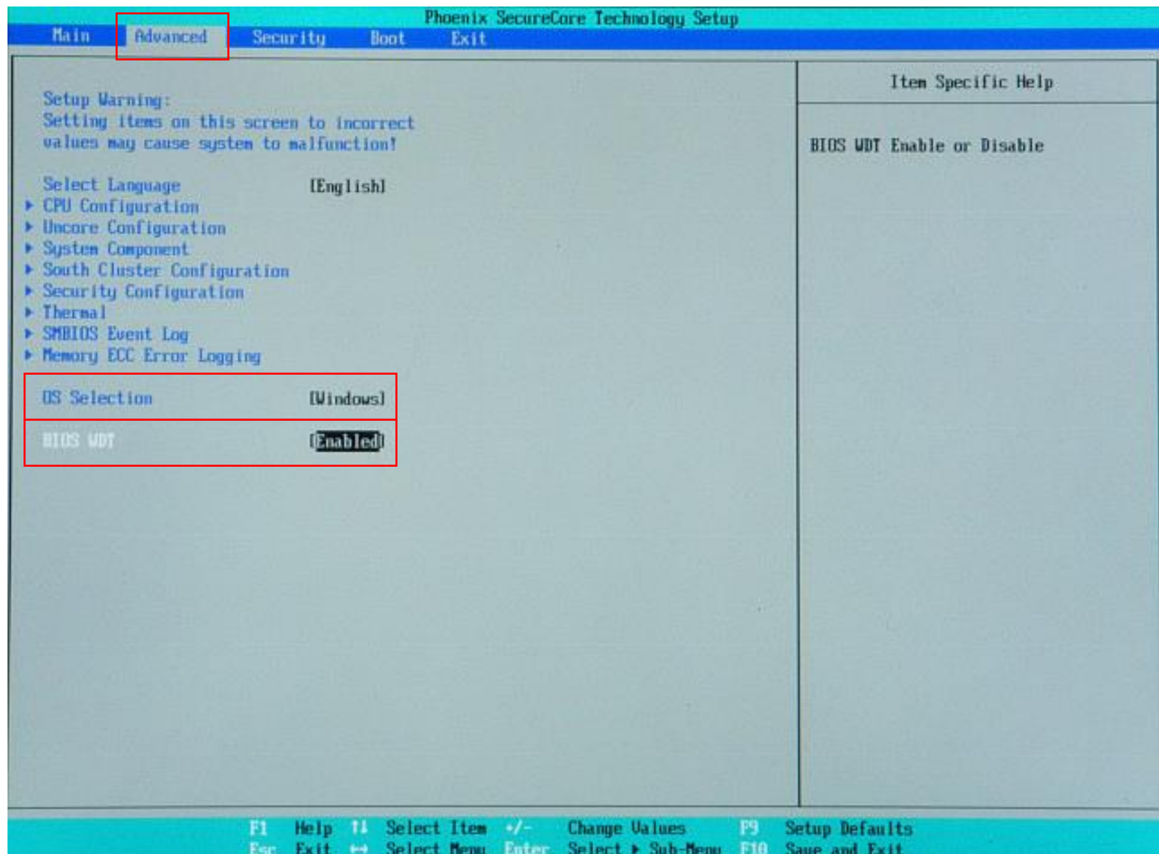

図 6-1-6. BIOS 設定 通常使用設定

④ [Exit]メニューを選択します。
⑤ [Exit Saving Changes]を実行し、設定を保存して終了します。

６－２　　システムの復旧（工場出荷状態）

工場出荷イメージをメインストレージ（mSATA1）に書込むことで、システムを工場出荷状態に復旧することができます。

※　システムを工場出荷状態へ復旧するとメインストレージにあるデータはすべて消えてしまいます。必要なデータがある場合は、復旧作業を行う前に保存してください。

※　工場出荷状態へのシステム復旧には、ソフトウェア使用許諾契約に同意していただく必要があります。ソフトウェア使用許諾契約は、製品に同梱されている「Microsoft Software License Term for: Windows XP Embedded and Windows Embedded Standard Runtime」に記載されています。システム復旧を行う場合は、内容を確認するようにしてください。

● システム復旧（工場出荷状態）の手順

① LAN ケーブルが接続されている場合は、LAN ケーブルを取り外してください。
② USB メモリ、SD カードなどのストレージメディアが接続されている場合は、ストレージメディアを取り外してください。
③ 「6-1-1 リカバリ DVD 起動」を参考にリカバリ DVD を起動させます。
④ リカバリメイン画面（図 6-1-5）で[システムの復旧（工場出荷状態）]を選択し、[次へ]ボタンを押します。
⑤ リカバリデータ選択画面（図 6-2-1）が表示されます。製品に合わせてリカバリデータを選択し、[次へ]ボタンを押します。表 6-2-1 に FP シリーズで選択できるリカバリデータを示します。

表 6-2-1. FP シリーズ リカバリデータ

#	リカバリデータ名	内容
1	WindowsEmbeddedStandard 7 FP シリーズ	FP シリーズ用 Windows Embedded Standard 7 32 ビット版 リカバリデータ

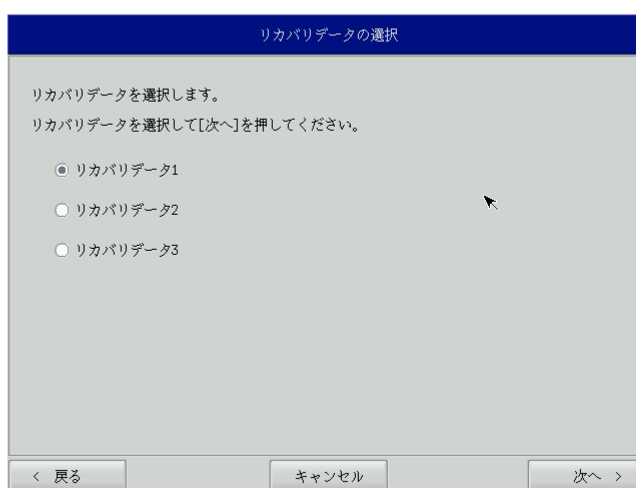

図 6-2-1. リカバリデータ選択画面

⑥　ソフトウェア使用許諾契約確認画面（図 6-2-2）が表示されます。使用許諾契約を確認し、使用許諾契約の諸条件に同意できる場合は[次へ]ボタンを押します。

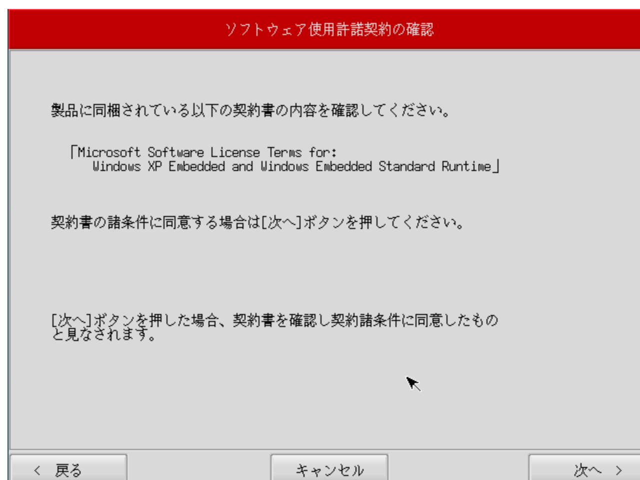

図 6-2-2. ソフトウェア使用許諾契約確認画面

⑦　確認画面（図 6-2-3）が表示されます。リカバリデータを確認し、[次へ]ボタンを押して処理を開始します。

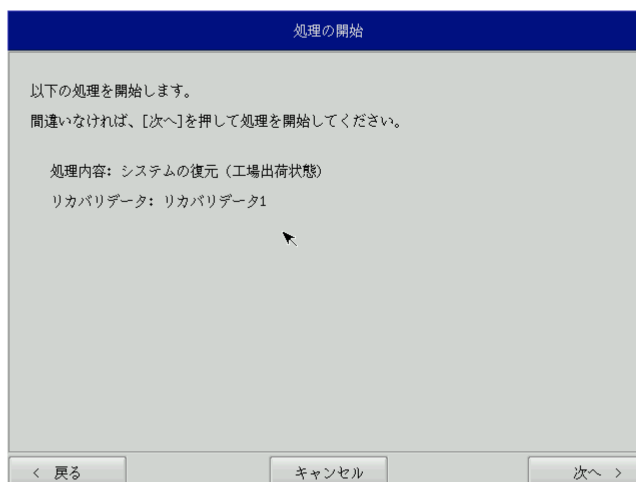

図 6-2-3. 確認画面

⑧　処理が始まります（図6-2-4）。実行中はDVDドライブを外したり、電源を落としたりしないでください。

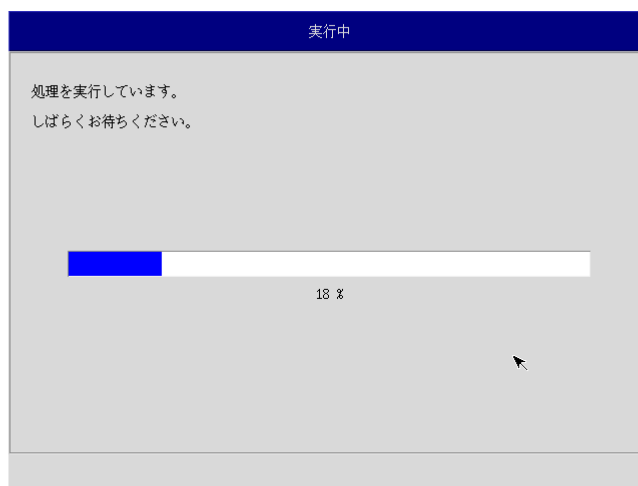

図6-2-4. 実行中画面

⑨　終了画面（図6-2-5）が表示されると工場出荷イメージの書込みは完了です。[終了]ボタンを押して電源を落とし、DVDドライブを外します。

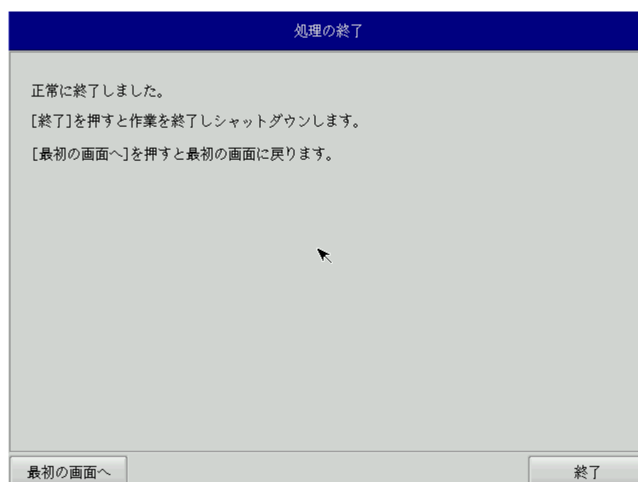

図6-2-5. 終了画面

⑩　電源を入れ、BIOS 起動画面が表示されたところで[F2]キーを押し、BIOS 設定画面を表示させます。
⑪　BIOS 設定画面が表示されたら、[Advanced]メニューを選択します。(図 6-1-2)
⑫　[OS Selection]を[Windows]、[BIOS WDT]を[Disabled]に設定します。

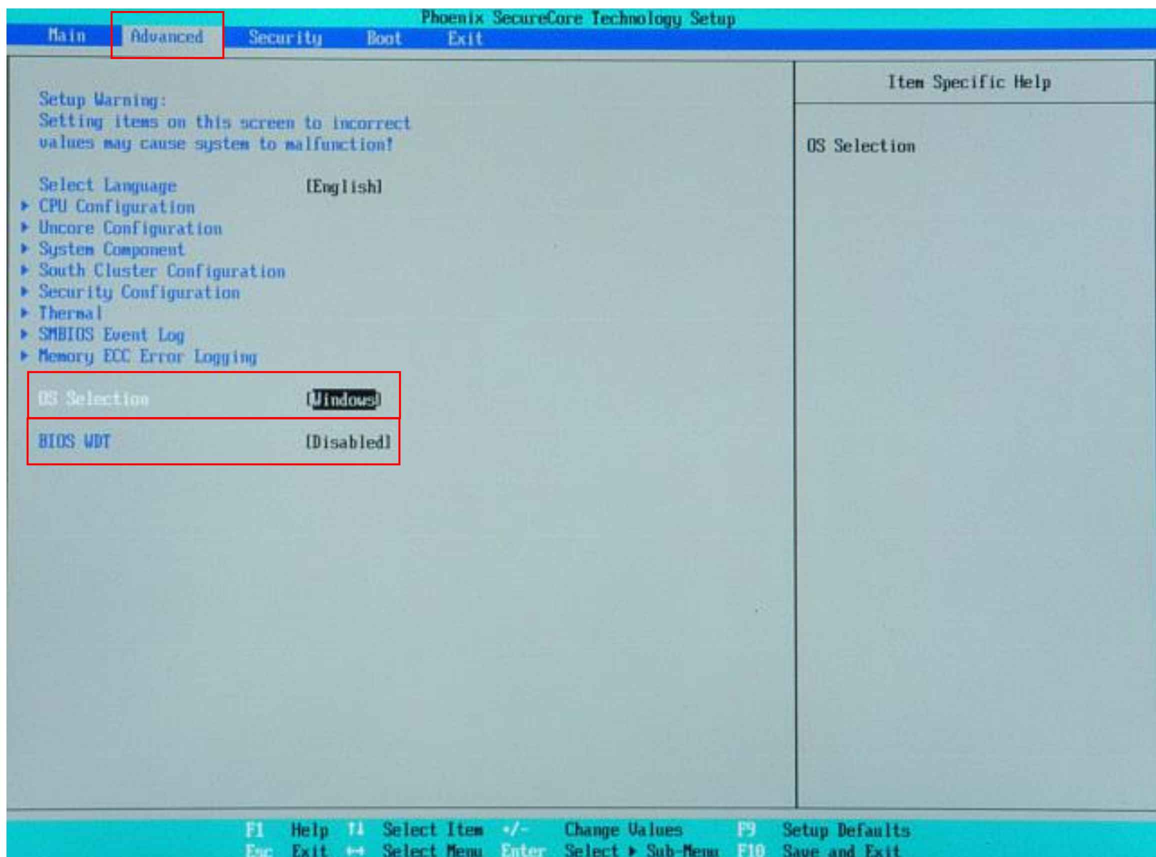

図 6-2-6. BIOS 設定 Windows 起動

⑬　[Exit]メニューを選択します。
⑭　[Exit Saving Changes]を実行し、設定を保存して終了します。
⑮　起動すると初期化処理が開始されます。
⑯　デスクトップが表示されて正常に起動すれば、システム復旧は完了です。
⑰　「6-1-3 リカバリ後処理」を参考に BIOS 設定を通常使用用に戻します。

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

６－３　　システムのバックアップ

メインストレージ（mSATA1）の状態をファイルに保存します。バックアップファイル保存のために外部メモリとして、USB メモリ、SD カードなどが必要となります。外部メモリの空き容量は、バックアップファイルの保存に十分な空き容量が必要となります。安全のためメインストレージの容量以上の空き容量がある外部メモリを用意してください。

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

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

※　バックアップファイルのサイズは、システムの状態によって変化しますので注意してください。

※　作成されたバックアップファイルは、バックアップ作業を行った本体でのみ動作します。同じ型の本体であっても、他の本体では動作しませんので注意してください。

●システムのバックアップの手順

① LAN ケーブルが接続されている場合は、LAN ケーブルを取り外してください。
② USB メモリ、SD カードなどのストレージメディアが接続されている場合は、ストレージメディアを取り外してください。
③ 「6-1-1 リカバリ DVD 起動」を参考にリカバリ DVD を起動させます。
④ リカバリメイン画面（図 6-1-5）で[システムのバックアップ]を選択し、[次へ]ボタンを押します。
⑤ メディアとパーティション選択画面（図 6-3-1）が表示されます。本体にメディアを接続し、[メディア情報更新]ボタンを押してください。バックアップファイルを保存するメディアのパーティションを選択し、[次へ]ボタンを押します。
メディアの認識には少し時間がかかります。メディアを接続してすぐに[メディア情報更新]ボタンを押すと、目的のメディア情報が現れないことがあります。この場合は、1 分程度待って再度、[メディア情報更新]ボタンを押してみてください。

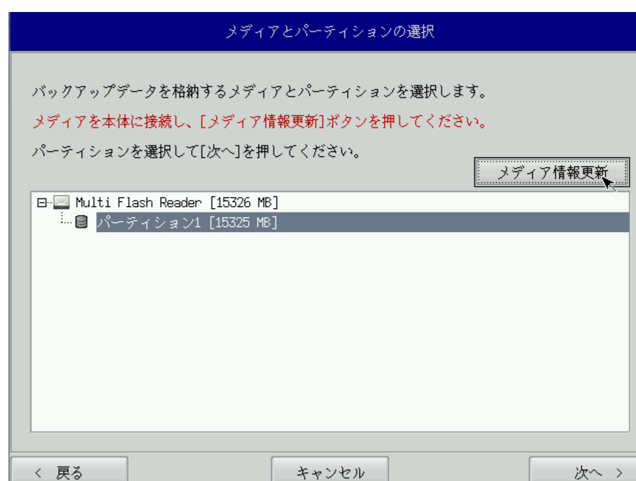

図 6-3-1. メディアとパーティション選択画面

⑥　フォルダ選択画面（図 6-3-2）が表示されます。［参照］ボタンを押します。

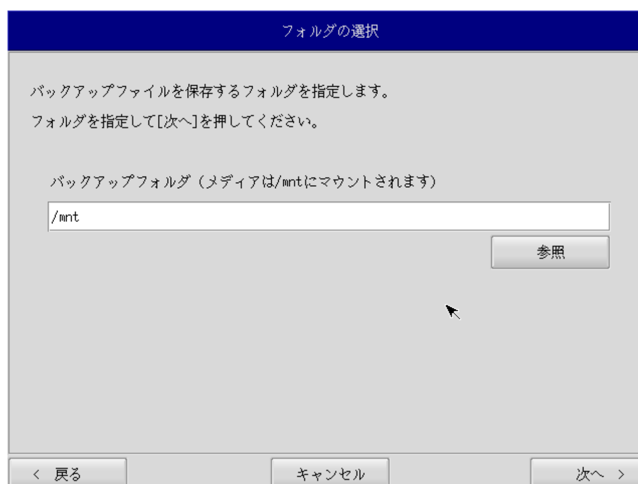

図 6-3-2. フォルダ選択画面

⑦　フォルダ参照画面（図 6-3-3）が表示されます。②で接続したパーティションは、/mnt にマウントされますので、/mnt 以下のフォルダを選択してください。[OK]を押すとフォルダ選択画面にもどります。
※　USB メモリに backup というフォルダがあり、このフォルダに保存する場合
/mnt/backup
を指定します。

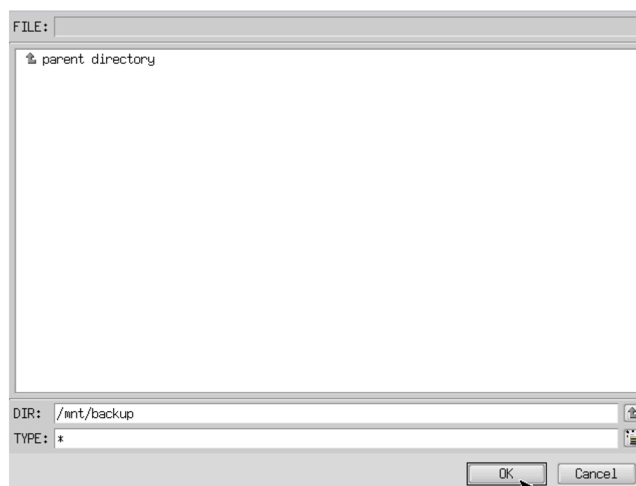

図 6-3-3. フォルダ参照画面

⑧ フォルダ選択画面（図 6-3-2）で指定したバックアップフォルダが入力されていることを確認します。[次へ]ボタンを押します。

⑨ 確認画面（図 6-3-4）が表示されます。メディア、パーティション、保存ファイルを確認します。[次へ]ボタンを押します。
※　保存ファイル名は、現在時刻から自動生成されます。

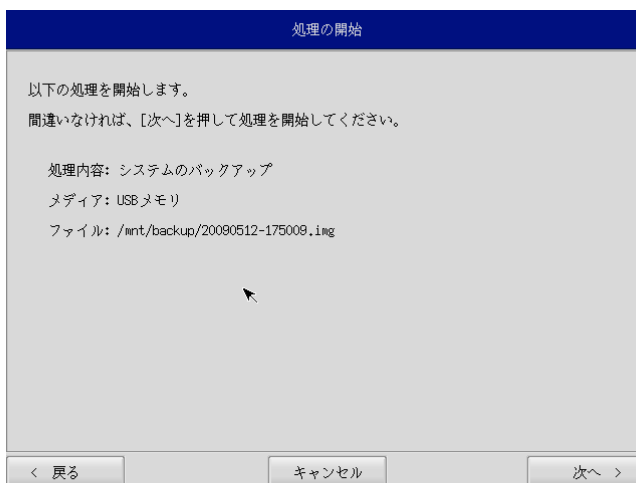

図 6-3-4. 確認画面

⑩ 実行中画面（図 6-3-5）が表示され、処理が開始されます。実行中は DVD ドライブ、保存メディアを外さないでください。また、電源を落とさないようにしてください。

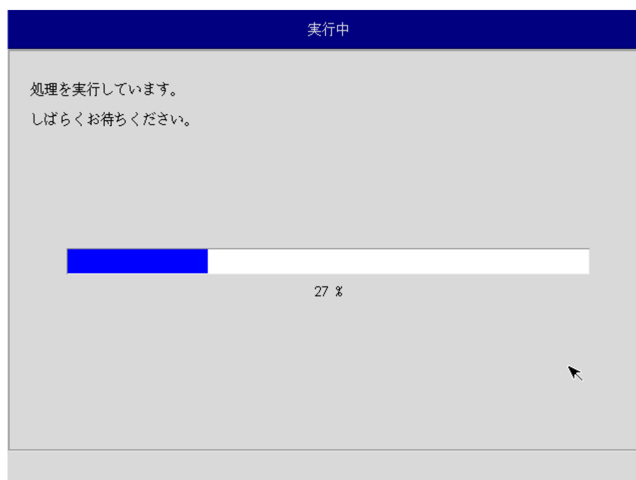

図 6-3-5. 実行中画面

⑪　終了画面（図6-3-6）が表示されるとバックアップ作業は完了です。[終了]ボタンを押して電源を落としてください。

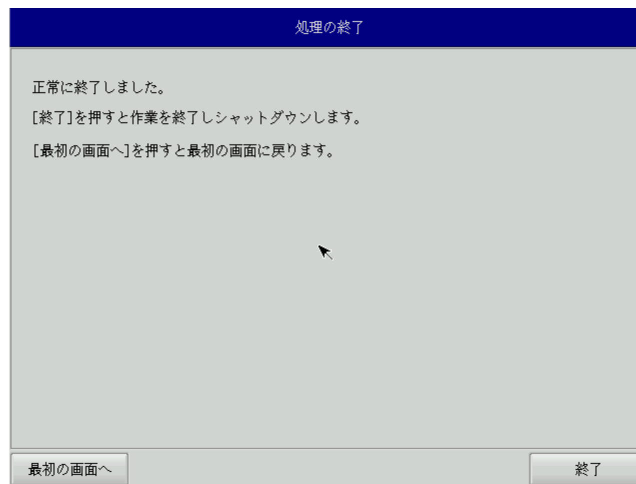

図6-3-6. 終了画面

⑫　「6-1-3 リカバリ後処理」を参考にBIOS設定を通常使用用に戻します

６－４　　システムの復旧（バックアップデータ）

「システムのバックアップ」で作成したバックアップファイルを使用して、メインストレージ（mSATA1）をバックアップファイルの状態に復旧させることができます。

※　バックアップファイルは、必ず対象となる本体で作成されたものを使用してください。他の本体のバックアップファイルでは動作しないので注意してください。

※ バックアップデータで復旧を行うとメインストレージのデータは、バックアップファイルの状態に戻ります。必要なデータがある場合は、復旧作業を行う前に保存してください。

●システムの復旧（バックアップデータ）の手順

① LAN ケーブルが接続されている場合は、LAN ケーブルを取り外してください。
② USB メモリ、SD カードなどのストレージメディアが接続されている場合は、ストレージメディアを取り外してください。
③ 「6-1-1 リカバリ DVD 起動」を参考にリカバリ DVD を起動させます。
④ リカバリメイン画面（図 6-1-5）で[システムの復旧（バックアップデータ）]を選択し、[次へ]ボタンを押します。
⑤ メディアとパーティション選択画面（図 6-4-1）が表示されます。本体にメディアを接続し、[メディア情報更新]ボタンを押してください。バックアップファイルを保存するメディアのパーティションを選択し、[次へ]ボタンを押します。
メディアの認識には少し時間がかかります。メディアを接続してすぐに[メディア情報更新]ボタンを押すと、目的のメディア情報が現れないことがあります。この場合は、1 分程度待って再度、[メディア情報更新]ボタンを押してみてください。

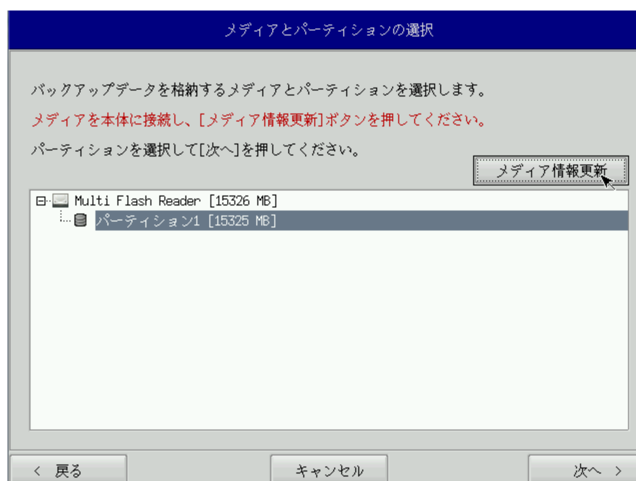

図 6-4-1. メディアとパーティション選択画面

⑥　ファイル選択画面（図 6-4-2）が表示されます。[参照]ボタンを押します。

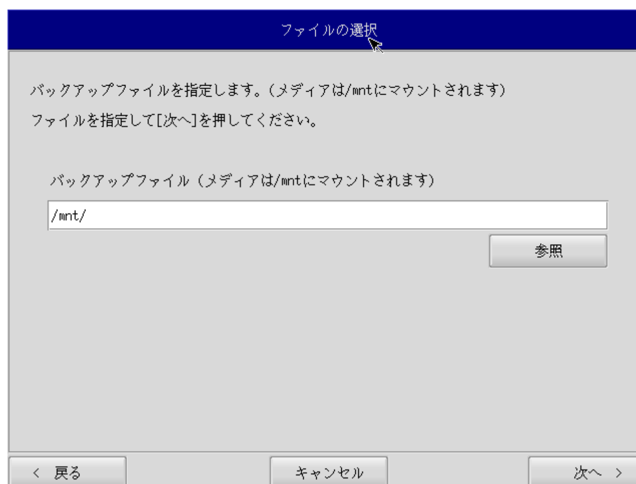

図 6-4-2. ファイル選択画面

⑦　ファイル参照画面（図 6-4-3）が表示されます。②で接続したメディアは、/mnt にマウントされていますので、/mnt 以下から目的のファイルを探してください。[OK]を押すとファイル選択画面にもどります。
※　USB メモリの¥backup¥20090512-175009.img というバックアップファイルを指定する場合
/mnt/backup/20090512-175009.img
を指定します。

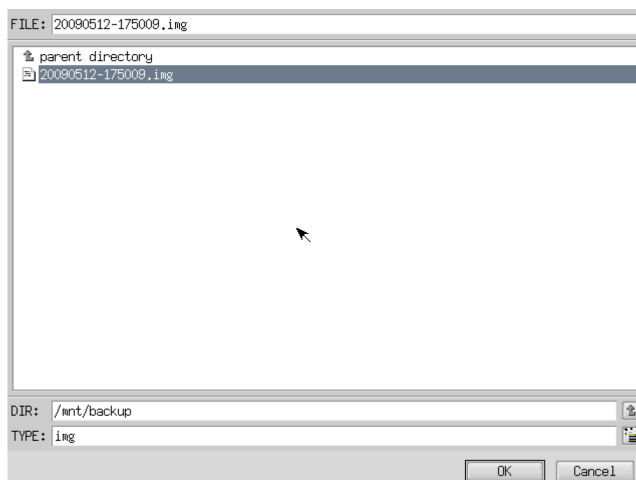

図 6-4-3. ファイル参照画面

⑧ ファイル選択画面（図 6-4-2）で指定したバックアップファイルが入力されていることを確認します。[次へ]ボタンを押します。

⑨ 確認画面（図 6-4-4）が表示されます。メディア、パーティション、バックアップファイルを確認します。[次へ]ボタンを押します。

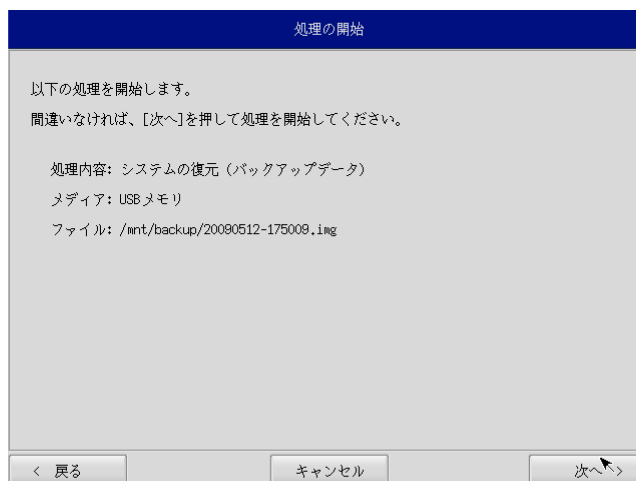

図 6-4-4. 確認画面

⑩ 実行中画面（図 6-4-5）が表示され、処理が開始されます。実行中は DVD ドライブ、保存メディアを外さないでください。また、電源を落とさないようにしてください。

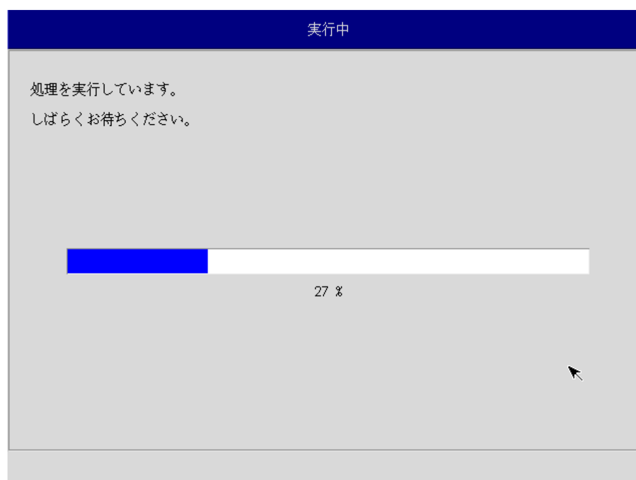

図 6-4-5. 実行中画面

⑪　終了画面（図 6-4-6）が表示されるとシステム復旧作業は完了です。[終了]ボタンを押して電源を落とし、DVD ドライブ、保存メディアを外します。

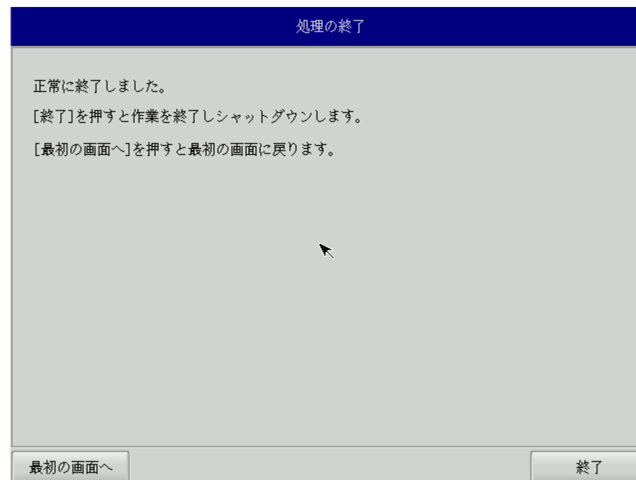

図 6-4-6. 終了画面

⑫　電源を入れ、BIOS 起動画面が表示されたところで[F2]キーを押し、BIOS 設定画面を表示させせます。
⑬　BIOS 設定画面が表示されたら、[Advanced]メニューを選択します。（図 6-1-2）
⑭　[OS Selection]を[Windows]、[BIOS WDT]を[Disabled]に設定します。

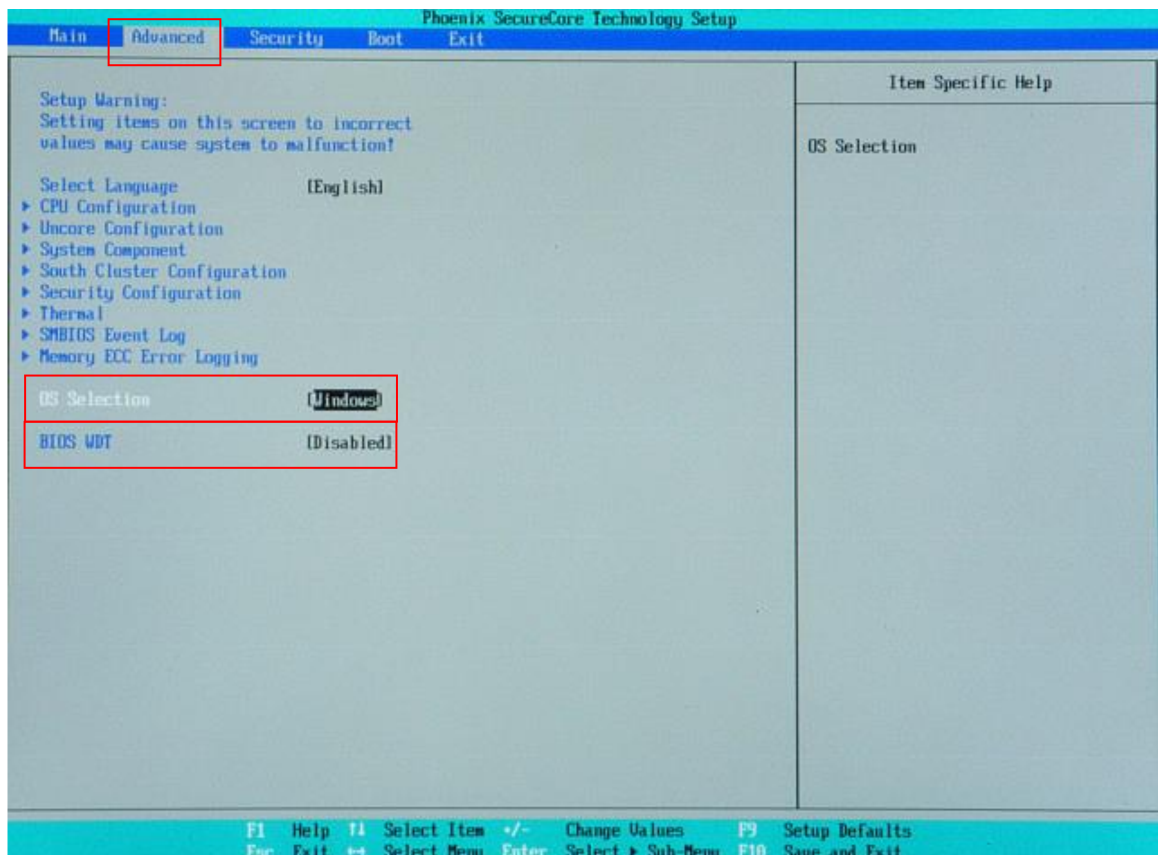

図 6-4-7. BIOS 設定 Windows 起動

⑮　[Exit]メニューを選択します。
⑯　[Exit Saving Changes]を実行し、設定を保存して終了します。
⑰　電源を入れ起動を確認します。
⑱　「6-1-3 リカバリ後処理」を参考に BIOS 設定を通常使用用に戻します。

付録

A－1　マイクロソフト製品の組込み用 OS（Embedded）について

【OS の注意事項】

本製品に搭載している OS は組込み用（Embedded）であり、一般的なパソコンで使用される OS とは異なります。そのためソフトウェアや接続デバイスの動作が異なる（または動作しない、インストールできない）等の可能性がありますので、お客様にて十分な動作確認を実施して頂きますようお願いいたします。

【OS 種別】

本製品には以下のマイクロソフト製品の OS が存在します。各々下記の意味で使用しておりますので、ご認識ください。

○ Windows Embedded Standard 7

：マイクロソフト社が Windows 7 をベースに組込み開発で使用できるようにコンポーネント化した OS

【OS 用のアプリケーション開発】

・Windows Embedded Standard 7 は多言語 OS のため、日本語 OS と動作、表示が異なる場合があります。アプリケーション開発時にはご注意ください。

・クロス環境によるアプリケーション開発をお願いします。実機での開発はできません。

【I/O 機器の接続について】

本製品のインターフェースを介して周辺デバイスを接続する場合、組込み用（Embedded）OS では通常 OS とは機能が異なります。十分な動作確認を実施してください。

各種 I/O 機器の動作において動作不良が発生した場合には、機器提供メーカーへお問い合わせをお願いいたします。

【提案に際して】

・お客様への提案に際して、事前に装置の寿命年数と条件、保守条件（有寿命部品）等の諸条件の説明をお願いいたします。

・本装置は、医療機器･原子力設備や機器、航空宇宙機器・輸送設備など、人命に関わる設備や機器および高度な信頼性を必要とする設備や機器などへの組込み、また、これらの機器の制御などを目的とした使用は意図されておりません。これらの設備や機器、制御システムなどに本装置を使用した結果、人身事故、財産損害などが生じてもいかなる責任も負いかねます。

・本装置（ソフトウェアを含む）が、外国為替および外国貿易法の規定により、輸出規制品に該当する場合は、海外輸出の際に日本国政府の輸出許可申請等必要な手続きをお取り下さい。また、米国再輸出規制等外国政府の規制を受ける場合には、所定の手続きを行ってください。

＜制約事項＞

- Embedded ライセンスは、組込機器や特定業務用機器の OS としてのみご使用いただけます。汎用目的（「市販のアプリケーションをエンドユーザがインストールして使用する」など）ではご使用いただけません。
- OS はお客様で開発した専用アプリケーションとあわせて利用しなければなりません。必ず専用アプリケーションをインストールしてご使用ください。
- 医療機器･原子力設備や機器、航空宇宙機器・輸送設備など、誤動作により被害が想定される装置への使用はできません。
- 製品構成に関する制限
  —Embedded ライセンス契約であることを示す「Certificate of Authenticity」ステッカーを装置本体に貼り付けて出荷します。
  —OS のインストール媒体は添付できません。復旧媒体（アプリケーション込み）の添付は可能です。
- 専用アプリケーションに関する制限
  —専用アプリケーションのユーザインタフェースからのみ、他のアプリケーションやファンクションにアクセスできるようにしなければなりません。
  —Microsoft のユーザインタフェース（ロゴ、ブートアップ、デスクトップ画面、フォルダ、ツールバーなど）を一切表示してはいけません。
- 組込みシステムの再販売･再頒布に際しマイクロソフト社の OS 製品の COA と APM を各システムに必ず貼付・添付しなければなりません。
- 組込み型システムとは別にマイクロソフト社の OS 製品またはその製品の一部を宣伝したり､価格提示したり、あるいは販売したり再頒布したりしてはなりません。
- ここに定めた条件を守っていないことが判明した場合、株式会社アルゴシステムはマイクロソフト株式会社との契約に従って状況報告をマイクロソフト株式会社に行い、出荷停止・状況改善依頼・調査を行うことができます。

＜用語の説明＞

□ 「APM」とは「関連製品資料」のことで、使用許諾製品の再配布可能コンポーネントとしてマイクロソフト社が適宜指定する、使用許諾製品に関連するドキュメント、ソフトウェアや他の有形資料を含んだ外部メデイアを意味します。なお、APM に COA は含まれません。

□ 「COA」すなわち「Certificate of Authenticity」は、マイクロソフト社が使用許諾製品のみに作成した削除不可のステッカーを意味します。

□ 「組込み型アプリケーション」とは、業界または業務固有のソフトウェアプログラムを意味し、以下の属性をすべて備えているものです。

（１） 組込み型システムの主要な機能がある。
（２） 組込み型システムが販売される業界に特有な機能要求に合わせて設計されている。
（３） 使用許諾製品ソフトウェアに加えて重要な機能がある。

□ 「組込み型システム」とは、組込み型アプリケーション用に設計され、組込み型アプリケーションと共に配布され且つ、汎用的なパーソナルコンピューテイングデバイス（パーソナルコンピュータなど）または多機能サーバとして販売もしくは使用されません。また、これらシステムの代替品として商業的に実現不可能な、お客様のイメージ付きコンピューテイングシステムまたはデバイスを意味します。

このマニュアルについて

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

７７Ｗ０１０１９９Ｃ　２０１５年　１０月　第３版
７７Ｗ０１０１９９Ａ　２０１４年　１０月　初版

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

ホームページ　http://www.algosystem.co.jp