

マニュアル

産業用組込み PC 4A IoT シリーズ用
『Windows 10 IoT Enterprise』
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 産業用組込み PC シリーズ用 Windows 10 IoT Enterprise 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-UWF機能 2-3
 - 2-3-1 UWFとは 2-3
 - 2-3-2 ドライブとUWF設定 2-4
 - 2-3-3 UWFの設定方法 2-4
 - 2-3-4 UWF Config Tool 2-13

2-3-5	UWF を使用するにあたっての注意事項	2-16
2-4	UPS 機能・RAM バックアップ機能	2-19
2-4-1	UPS 機能	2-19
2-4-2	RAM バックアップ機能	2-20
2-5	ログオン設定	2-21
2-5-1	自動ログオン設定	2-21
2-6	言語設定	2-22
2-6-1	マルチ言語機能	2-22
2-6-2	言語の変更方法	2-22
2-7	サービス設定	2-23
2-7-1	サービス設定の変更	2-23
2-8	ASD Config Tool	2-24
2-8-1	ASD Config Tool	2-24
2-8-2	Serial Port Setting	2-24
2-8-3	Board Information	2-25
2-8-4	初期値	2-25
2-9	RAS Config Tool	2-26
2-9-1	RAS Config Tool	2-26
2-9-2	Temperature	2-27
2-9-3	Temperature Configuration	2-28
2-9-4	Watchdog Timer	2-29
2-9-5	Watchdog Timer Configuration	2-31
2-9-6	Secondary RTC	2-32
2-9-7	Secondary RTC Configuration	2-33
2-9-8	Wake On RTC Timer 設定例	2-34
2-9-9	Backup Battery Monitor	2-35
2-9-10	初期値	2-36
2-10	UPS Config Tool	2-37
2-10-1	UPS Config Tool	2-37
2-10-2	UPS サービスの設定	2-37
2-10-3	RAM Backup	2-39
2-10-4	UPS を使用するにあたっての注意事項	2-40
2-10-5	初期値	2-41
2-11	ユーザーアカウント制御	2-42
2-12	S. M. A. R. T. 機能	2-43

2-1 2-1	iSMART	2-4 3
2-1 2-2	S.M.A.R.T. Monitor Tool	2-4 4

第3章 産業用組込み PC 4A IoT について

3-1	産業用組込み PC 4A IoT*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	3-5
3-2-1	グラフィック	3-5
3-2-2	シリアルポート	3-5
3-2-3	有線 LAN	3-5
3-2-4	サウンド	3-5
3-2-5	USB2.0 ポート	3-5
3-2-6	USB3.0 ポート	3-5
3-2-7	LTE(オプション)	3-6
3-2-8	無線 LAN (オプション)	3-6
3-2-9	Bluetooth (オプション)	3-6
3-3	組込みシステム機能	3-7
3-3-1	タイマ割込み機能	3-8
3-3-2	汎用入出力	3-8
3-3-3	RAS 機能	3-8
3-3-4	シリアルコントロール機能	3-8
3-3-5	ダミーSRAM	3-8
3-3-6	初期化スイッチ	3-8
3-3-7	汎用 LED	3-9
3-3-8	ハードウェア・ウォッチドッグタイマ機能	3-9
3-3-9	ソフトウェア・ウォッチドッグタイマ機能	3-9
3-3-10	外部 RTC 機能	3-9
3-3-11	温度監視機能	3-9
3-3-12	バックアップバッテリーモニタ	3-9
3-3-13	Wake On RTC Timer 機能	3-9
3-3-14	UPS 機能	3-10

第4章 組込みシステム機能ドライバ

4-1	ドライバの使用について	4-1
-----	-------------	-----

4-1-1	開発用ファイル	4-1
4-1-2	DeviceIoControl について	4-2
4-2	タイマ割込み機能	4-3
4-2-1	タイマ割込み機能について	4-3
4-2-2	タイマドライバについて	4-3
4-2-3	タイマデバイス	4-4
4-2-4	タイマドライバの動作	4-5
4-2-5	ドライバ使用手順	4-6
4-2-6	DeviceIoControl リファレンス	4-7
4-2-7	サンプルコード	4-11
4-3	汎用入出力	4-16
4-3-1	汎用入出力について	4-16
4-3-2	汎用入出力ドライバについて	4-17
4-3-3	汎用入出力デバイス	4-18
4-3-4	DeviceIoControl リファレンス	4-19
4-3-5	サンプルコード	4-21
4-4	RAS 機能	4-26
4-4-1	RAS 機能について	4-26
4-4-2	RAS-IN ドライバについて	4-26
4-4-3	RAS-IN デバイス	4-27
4-4-4	IN1 割込みの使用手順	4-28
4-4-5	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	4-29
4-4-6	DeviceIoControl リファレンス	4-30
4-4-7	サンプルコード	4-34
4-5	シリアルコントロール機能	4-42
4-5-1	シリアルコントロール機能について	4-42
4-5-2	シリアルコントロールドライバについて	4-42
4-5-3	SciCtl デバイス	4-43
4-5-4	DeviceIoControl リファレンス	4-44
4-5-5	サンプルコード	4-46
4-6	ダミーSRAM	4-49
4-6-1	ダミーSRAM について	4-49
4-6-2	ダミーSRAM ドライバについて	4-49
4-6-3	ダミーSRAM デバイス	4-50
4-6-4	DeviceIoControl リファレンス	4-51

4-6-5	サンプルコード	4-52
4-7	初期化スイッチ、汎用 LED	4-55
4-7-1	初期化スイッチ、汎用 LED について	4-55
4-7-2	初期化スイッチ/汎用 LED ドライバについて	4-56
4-7-3	初期化スイッチ/汎用 LED デバイス	4-57
4-7-4	DeviceIoControl リファレンス	4-58
4-7-5	サンプルコード	4-60
4-8	ハードウェア・ウォッチドッグタイマ機能	4-68
4-8-1	ハードウェア・ウォッチドッグタイマ機能について	4-68
4-8-2	ハードウェア・ウォッチドッグタイマドライバについて	4-68
4-8-3	ハードウェア・ウォッチドッグタイマデバイス	4-70
4-8-4	DeviceIoControl リファレンス	4-72
4-8-5	サンプルコード	4-77
4-9	ソフトウェア・ウォッチドッグタイマ機能	4-84
4-9-1	ソフトウェア・ウォッチドッグタイマ機能について	4-84
4-9-2	ソフトウェア・ウォッチドッグタイマドライバについて	4-84
4-9-3	ソフトウェア・ウォッチドッグタイマデバイス	4-85
4-9-4	DeviceIoControl リファレンス	4-87
4-9-5	サンプルコード	4-92
4-10	RAS 監視機能	4-99
4-10-1	RAS 監視機能について	4-99
4-10-2	RAS DLL について	4-99
4-10-3	RAS DLL I/F 関数リファレンス	4-100
4-10-4	サンプルコード	4-101
4-11	外部 RTC 機能	4-105
4-11-1	外部 RTC 機能について	4-105
4-11-2	RAS DLL について	4-105
4-11-3	RAS DLL 時刻設定関数リファレンス	4-106
4-11-4	RAS DLL Wake On RTC Timer 設定関数リファレンス	4-107
4-11-5	サンプルコード	4-110
4-12	ビープ音	4-115
4-12-1	ビープ音について	4-115
4-12-2	ビープドライバについて	4-115
4-12-3	ビープデバイス	4-116
4-12-4	DeviceIoControl リファレンス	4-117

4-12-5	サンプルコード	4-121
4-13	バックアップバッテリーモニタ	4-125
4-13-1	バックアップバッテリーモニタについて	4-125
4-13-2	バックアップバッテリーモニタドライバについて	4-125
4-13-3	バックアップバッテリーモニタデバイス	4-126
4-13-4	DeviceIoControl リファレンス	4-127
4-13-5	サンプルコード	4-128
4-14	UPS サービス	4-130
4-14-1	UPS サービスについて	4-130
4-14-2	通知イベント、警告イベント	4-131
4-14-3	UPS 状態	4-131
4-14-4	UPS サービスの使用について	4-132
4-14-5	サンプルコード	4-132
4-15	S. M. A. R. T. イベント通知	4-142
4-15-1	S. M. A. R. T. イベント通知について	4-142
4-15-1	サンプルコード	4-143

第5章 システムリカバリ

5-1	リカバリ DVD について	5-1
5-1-1	リカバリ準備	5-2
5-1-2	リカバリ USB 起動	5-5
5-1-3	リカバリ作業	5-7
5-1-4	リカバリ後処理	5-8
5-2	システムの復旧 (バックアップデータ)	5-10
5-3	システムのバックアップ	5-15

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、産業用組込み PC 4A IoT シリーズ EC4A-xxxGA/GT（以降「産業用組込み PC シリーズ」と表記します）用 Windows 10 IoT Enterprise について説明します。

Windows 10 IoT Enterprise は Windows の産業機器向けラインナップである Windows Embedded シリーズの後継製品で、Windows 10 Enterprise をベースとした、あらゆるデバイスが相互につながり新たな価値を創造する Internet of Things (IoT) の世界において、デバイス間やクラウドへの接続性を重視し、IoT デバイス上での開発効率のよいアプリ実行環境を提供する OS です。本書では、産業用組込み PC シリーズ用の Windows 10 IoT Enterprise 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 10 IoT Enterprise は Windows 10 Enterprise と完全互換の OS ですが、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、産業用組込み PC 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、産業用組込み PC シリーズ用 Windows 10 IoT Enterprise 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、産業用組込み PC シリーズ用 Windows 10 IoT Enterprise の概要について説明します。

1-1 機能と特長

1-1-1 産業用組込み PC シリーズ用 Windows 10 IoT Enterprise とは

Windows 10 IoT Enterprise は、Windows の産業機器向けラインナップである Windows Embedded シリーズの後継製品です。Windows 10 Enterprise と完全互換の OS であり、あらゆるデバイスが相互につながり新たな価値を創造する Internet of Things (IoT) の世界において、デバイス間やクラウドへの接続性を重視し、IoT デバイス上での開発効率のよいアプリ実行環境を提供する OS です。

産業用組込み PC シリーズ用 Windows 10 IoT Enterprise は、Windows 10 IoT Enterprise を産業用組込み PC シリーズ用にカスタマイズしたものです。産業用組込み PC シリーズ用に用意された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

産業用組込み PC シリーズ用 Windows 10 IoT Enterprise は、Windows 10 Enterprise と完全互換の OS で、全ての Windows 10 デバイスで動作できるユニバーサル アプリケーションを提供、また同じ開発環境 (Visual Studio 2015) で作成できます。また、Windows Embedded 8 Standard から追加された「Unified Write Filter (UWF)」のファイルシステム保護機能、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。

※注：UWF は電源断には対応していません。シャットダウンしてください。

表 1-1-2-1 に産業用組込み PC シリーズ用 Windows 10 IoT Enterprise に搭載されている主な機能を示します。

表 1-1-2-1. 産業用組込み PC シリーズ用 Windows 10 IoT Enterprise の主な機能

機能	内容
Windows 10 IoT Enterprise	Windows 10 Enterprise と完全互換。
UWF (Unified Write Filter)	メディアへの書き込み動作をフィルタし、初期設定を保持します。
BitLocker	ドライブ暗号化を使用して、ドライブ全体のファイルの保護をサポートします。
Granular UX Control	アプリの操作や設計に関するマイクロソフトが SDK で提供している以上のコンポーネントの開発や利用ができます。
Device Guard	新しいセキュリティ技術で、紛失漏洩など各種の問題が起きたときに、ロックダウンや、データを保護する仕組みです。
Credential Guard	不正なアプリや目的外アプリのインストールを避けることができます。
Universal Apps	一つのユニバーサルアプリですべてのエディションの Windows 10 に展開が可能です。
DirectX 12	すべての Windows 10 デバイスで動作します。
Enterprise Data Protection	外から不正なプログラムの持ち込みやデータの持ち出しを禁止できます。
AppLocker	アプリの挙動を管理できます。

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。 <ul style="list-style-type: none">・ グラフィック・ シリアルポート・ 有線 LAN・ 無線 LAN・ サウンド(ヘッドホン出力)・ USB ポート・ SD カードスロット・ 無線 LAN・ Bluetooth・ LTE
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。 <ul style="list-style-type: none">・ タイマ割込み機能・ 汎用入出力・ RAS 機能(機能設定用コンパネアプリ)・ シリアルコントロール機能(RS-232C/422/485 切替)・ 初期化スイッチ・ 汎用 LED・ 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・ ビープ音・ バックアップバッテリーモニタ機能・ Wake On RTC Timer 機能・ ASD Config(機能設定用コンパネアプリ)

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、64GByte の m-SATA 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. 産業用組込み PC シリーズ ドライブ構成

ドライブ	容量	空き容量	内容
C	59.1 GByte	約 45 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

システムドライブのフォルダ、ファイル構成は Windows 10 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. 産業用組込み PC シリーズ Windows 10 IoT Enterprise 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)> <Windows> <ユーザー>

※ 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 10 IoT Enterprise は、ログイン可能なユーザーアカウントが 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 10 IoT Enterprise 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産業用組込み PC 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	DESKTOP-***** *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

産業用組込み PC シリーズ用 Windows 10 IoT Enterprise では、アプリケーション開発、ドライバ開発に Microsoft Visual Studio 2015 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産業用組込み PC シリーズ本体での開発ができません。開発は Windows 10 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、産業用組込み PC 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 10 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 2015 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。産業用組込み PC シリーズ用 Windows 10 IoT Enterprise 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。

※ 産業用組込み PC 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

産業用組込み PC 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。

アプリケーションの開発

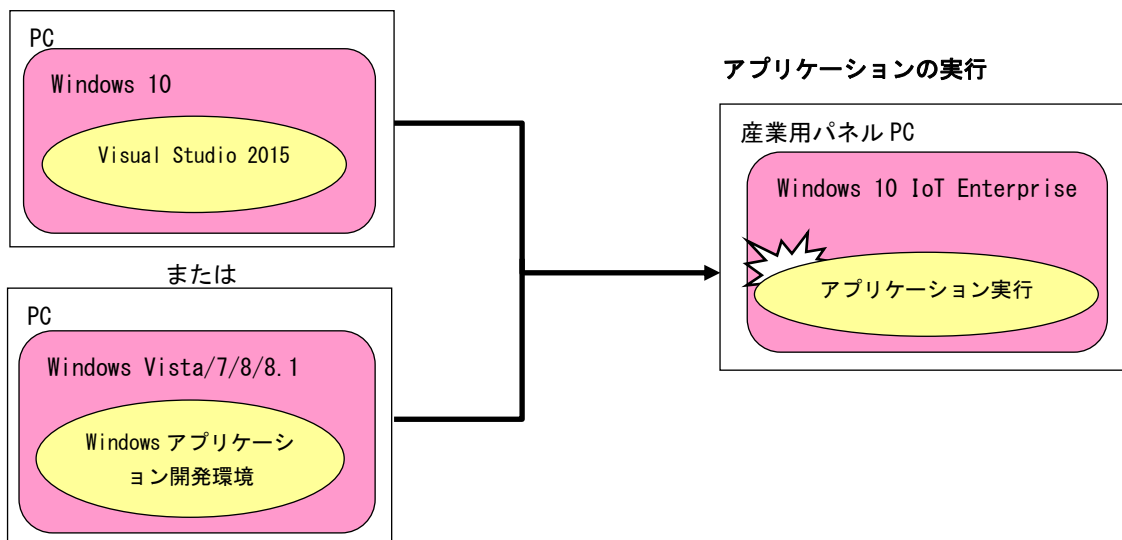


図 1-3-1. アプリケーション開発と実行

第 2 章 システムの操作

本章では、産業用組込み PC シリーズ用 Windows 10 IoT Enterprise 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

産業用組込み PC 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 10 IoT Enterprise 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。

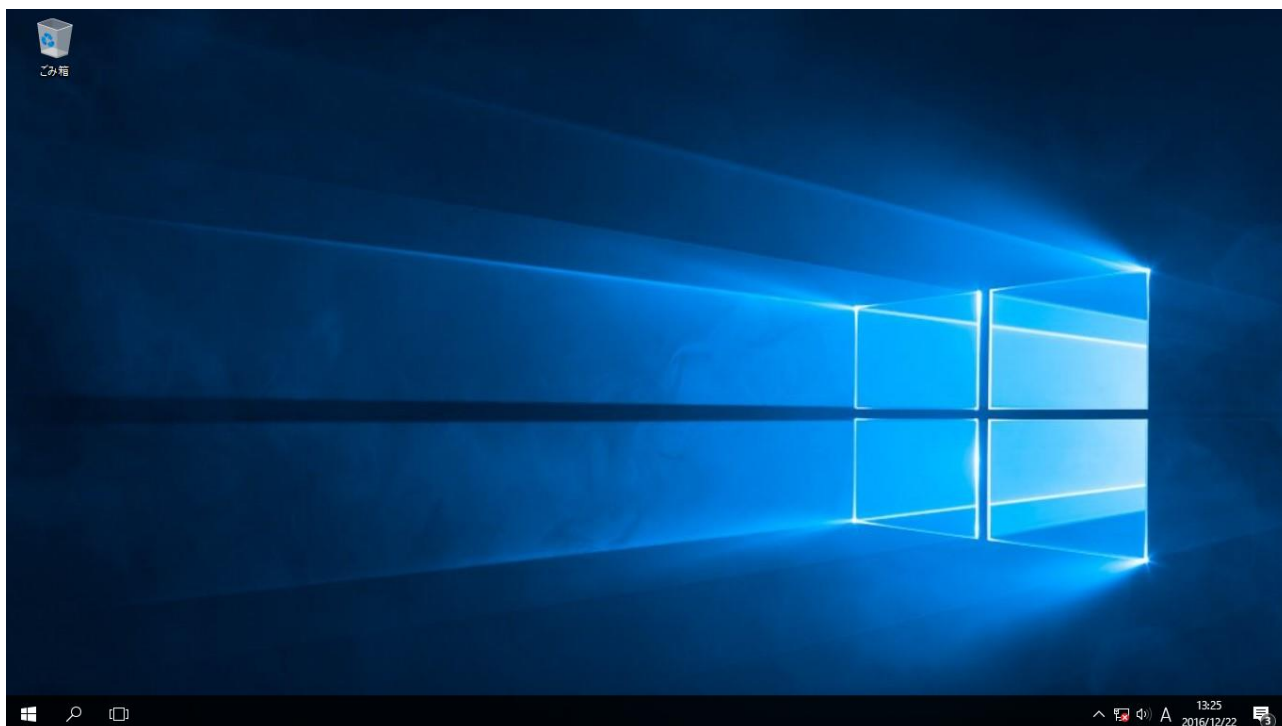


図 2-1-1-1. デスクトップ

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。

2-2 外部 RTC

2-2-1 RTC とシステム時刻について

産業用組込み PC 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、「2-9 RAS Config Tool」を参照してください。

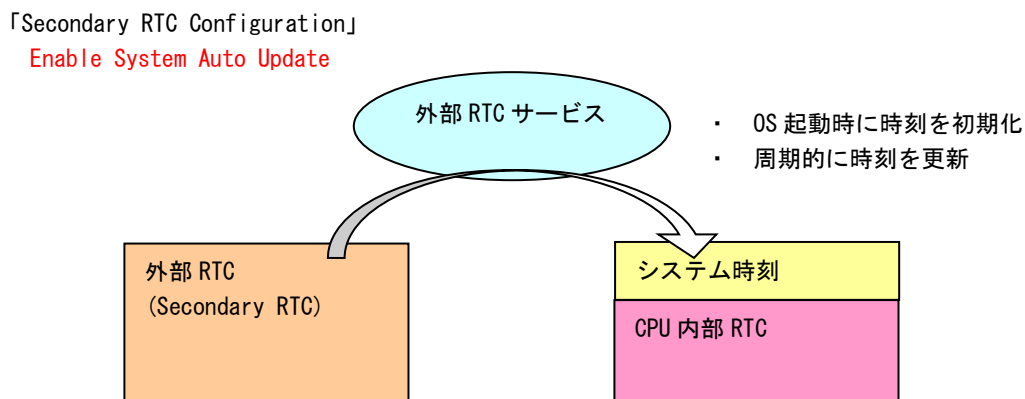


図 2-2-2-1. System Auto Update 機能有効

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、「2-9 RAS Config Tool」を参照してください。

※ `G5_SetSystemTime()`、`G5_SetLocalTime()` については、「4-11 外部 RTC 機能」を参照してください。

2-3 UWF 機能

2-3-1 UWF とは

UWF (Unified Write Filter) とは、Windows Embedded 8 Standard で新たに追加された機能で、Enhanced Write Filter (EWF) と File-Based Write Filter (FBWF) の双方の利点を組み合わせて、ファイルシステム保護を行う機能です。UWF では、セクターベースの保護とファイルベースの保護を行うことが可能です。また、レジストリの保護は、Windows Embedded Standard 7 までは Registry Filter を用いていましたが、UWF にもレジストリ保護機能が統合されました。UWF 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、書き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き込みを抑止する必要があります。UWF 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 UWF を有効にする直前の状態に戻すこともできます。

UWF には、「Unified Write Filter Servicing Mode」という機能が用意されています。Unified Write Filter Servicing Mode に移行させて、Windows Update や Windows Server Update Services (WSUS) から OS のアップデートプログラムを適用することも可能です。

UWF では、書き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。産業用組込み PC シリーズでは、オーバーレイに RAM を使用します。

※注：UWF は、電源断によるシステムディスクの保護は行いません。UWF を有効にしても、シャットダウンさせてから電源を OFF してください。

● UWF の特徴

Unified Write Filter の略

ドライブの変更内容を RAM に保存

UWF 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)

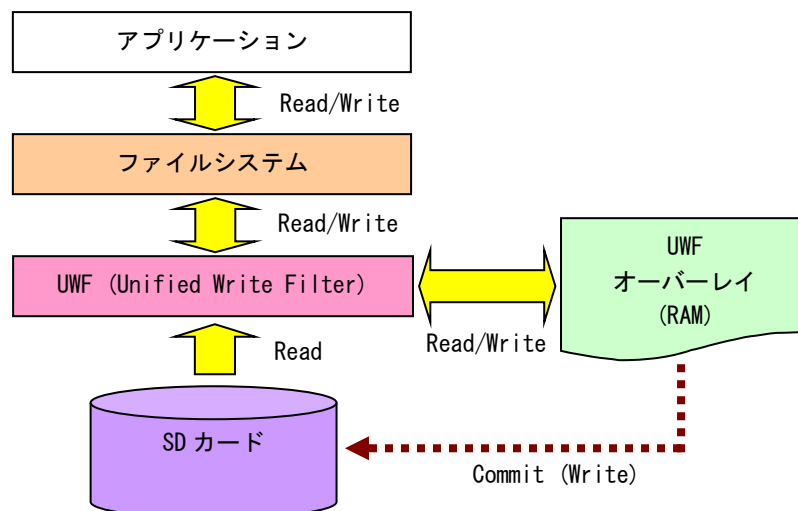


図 2-3-1-1. UWF の仕組み

2-3-2 ドライブと UWF 設定

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

※ UWF 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、UWF を無効にしてください。

表 2-3-2-1. 産業用組込み PC シリーズ 初期 UWF 設定

ドライブ	UWF	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 UWF の設定方法

UWF Manager コマンドを使用して UWF を操作することができます。UWF Manager コマンドはコンソールアプリケーションです。スタートボタンの右クリックから[コマンドプロンプト(管理者)]を開き、コマンドを実行します。

● UWF 有効

C ドライブの UWF 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 UWF が有効になります。

```
> uwfmgr volume protect c: ← UWF にて保護されるボリュームを C: にします
統合書き込みフィルター構成ユーティリティ バージョン 10.0.10240
Copyright © Microsoft Corporation. All right reserved.

ボリューム c: は UWF が有効になった後に統合書き込みフィルターによって保護されます。

> uwfmgr filter enable ← 次回起動時のコマンドに ENABLE が登録されます
統合書き込みフィルター構成ユーティリティ バージョン 10.0.10240
Copyright © Microsoft Corporation. All right reserved.

統合書き込みフィルターはシステム再起動後に有効になります。
```

● UWF 無効

C ドライブの UWF 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 UWF が無効になります。

```
> uwfmgr filter disable ← 次回起動時に UWF が無効になります
統合書き込みフィルター構成ユーティリティ バージョン 10.0.10240
Copyright © Microsoft Corporation. All right reserved.

統合書き込みフィルターはシステム再起動後に無効になります。
```

- コミット

UWF が有効の場合、ドライブへの書き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き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き込むことができます。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。

```
> uwfmgr file commit [ファイル名] ← 終了時に COMMIT が実行されます
統合書き込みフィルター構成ユーティリティ バージョン 10.0.10240
Copyright © Microsoft Corporation. All right reserved.

“ファイル名”は正常にコミットされました
```

※ 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。

- UWF の状態確認

C ドライブの UWF の状態を確認する場合は以下のとおりです。

```
> uwfmgr volume get-config c: ← C ドライブの UWF 状態を確認
統合書き込みフィルター構成ユーティリティ バージョン 10.0.10240
Copyright © Microsoft Corporation. All right reserved.

現在のセッションの設定
ボリューム xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxxx [C:]
  ボリュームの状態:      保護
  ボリューム ID:         xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxxx

  ファイル除外
  ボリューム xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxxx [C:]に対する現在のセッション除外
  *** 除外なし

  次のセッションの設定
  ボリューム xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxxx [C:]
  ボリュームの状態:      保護
  ボリューム ID:         xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxxx

  ファイル除外
  ボリューム xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxxx [C:]に対する現在のセッション除外
  *** 除外なし
```

● コマンドラインオプションとパラメーター一覧

UWF Manager には、説明したコマンド以外にもコマンドが存在します。表 2-3-3-1～表 2-3-3-7 に産業用組込み PC シリーズで使用できる UWF Manager のコマンド一覧を示します。

Uwfmgr [. exe] parameter [コマンド] [引数]

UWF ロックダウン オプションを構成します。

表 2-3-3-1. UWFMgr コマンド一覧

コマンド	内容
help ?	基本的なパラメータのコマンドラインヘルプを表示します。 --- 例 --- > uwfmgr ?
get-config	現在と次回のセッションのすべての構成設定情報を表示します。 --- 例 --- > uwfmgr get-config
filter	フィルター処理の状態などの UWF の基本設定を構成および表示します。(表 2-3-3-2)
volume	UWF で保護されるボリューム フィルター処理の設定を構成および表示します。(表 2-3-3-3)
file	UWF のファイル除外の設定を構成および表示します。(表 2-3-3-4)
registry	UWF のレジストリキー除外の設定を構成および表示します。(表 2-3-3-5)
overlay	UWF のオーバーレイの設定を構成および表示します。(表 2-3-3-6)
servicing	サービスモードの設定を構成および表示します。(表 2-3-3-7)

Uwfmgr [. exe] filter [コマンド]

フィルターの設定またはグローバル設定を構成します。

表 2-3-3-2. UWFMR Filter コマンド一覧

コマンド	内容
help ?	フィルター設定のコマンドラインヘルプを表示します。 --- 例 --- > uwfmgr filter ?
enable	システムの再起動後、次回のセッションで UWF の保護を有効にします。 --- 例 --- > uwfmgr filter enable
disable	システムの再起動後、次回のセッションで UWF の保護を無効にします。 --- 例 --- > uwfmgr filter disable
reset-settings	UWF 設定をリセットします。 --- 例 --- > uwfmgr filter reset-settings
restart	オーバーレイが最大または最大に近くなると、直ぐにデバイスを再起動します。 --- 例 --- > uwfmgr filter restart
shutdown	オーバーレイが最大または最大に近くなると、直ぐにデバイスをシャットダウンします。 --- 例 --- > uwfmgr filter shutdown

Uwfmgr [. exe] volume [コマンド] [引数]
 ボリューム固有のフィルター設定を構成します。

表 2-3-3-3. UWFMR volume コマンド一覧

コマンド	内容
help ?	ボリューム設定のコマンドラインヘルプを表示します。 --- 例 --- > uwfmgr volume ?
get-config {<volume> all}	指定されたボリューム、または全て (all) のボリュームの現在と次のセッションの UWF 構成設定を表示します。 --- 例 --- > uwfmgr volume get-config c:
protect {<volume> all}	UWF で保護されるボリュームリストに指定されたボリュームを追加します。UWF フィルタリングが有効になっている場合は、次のシステム再起動後にボリュームの保護を開始します。 --- 例 --- > uwfmgr volume protect c:
unprotect {<volume> all}	UWF で保護されるボリュームリストから指定されたボリュームを削除します。次のシステム再起動後にボリュームの保護を停止します。 --- 例 --- > uwfmgr volume unprotect c:

Uwfmgr [. exe] file [コマンド] [ボリューム名] [パス] [ファイル名]
 ファイルとディレクトリの除外設定を構成します。

表 2-3-3-4. UWFMgr File コマンド一覧

コマンド	内容
help ?	ファイル設定のコマンドラインヘルプを表示します。 --- 例 --- > uwfmgr file ?
get-exclusions {<volume> all}	指定されたボリュームの全てのファイルとディレクトリの除外リストを表示され、現在と次回のセッション情報を表示します。 --- 例 --- > uwfmgr file get-exclusions c:
add-exclusion <file>	UWF で保護されたボリュームのファイル除外リストに指定されたファイルを追加します。次回のシステム再起動後にファイルの保護を開始します。 --- 例 --- > uwfmgr file add-exclusion c:¥dir1¥dir2.txt
remove-exclusion <file>	UWF で保護されたボリュームのファイル除外リストから指定されたファイルを削除します。次回のシステム再起動後にファイルの保護を開始します。 --- 例 --- > uwfmgr file remove-exclusion c:¥dir1¥dir2.txt
commit <file>	指定されたファイルのオーバーレイの変更内容を保護されたボリュームに反映します。 --- 例 --- > uwfmgr file commit c:¥dir1¥dir2.txt
commit-delete <file>	指定されたファイルをオーバーレイと物理ボリュームの両方から削除します。 --- 例 --- > uwfmgr file commit-delete c:¥dir1¥dir2.txt

Uwfmgr [.exe] registry [コマンド] [キー] [値]

レジストリ除外の設定構成とレジストリの変更を反映します。

表 2-3-3-5. UWFMR Registry コマンド一覧

コマンド	内容
help ?	レジストリ設定のコマンドラインヘルプを表示します。 --- 例 --- > uwfmgr file ?
get-exclusions	現在と次回のセッション情報のレジストリ除外リスト内の全てのレジストリキーを表示します。 --- 例 --- > uwfmgr resistry get-exclusions
add-exclusion <key>	UWF のレジストリ除外リストに指定されたレジストリキーを追加します。次回のシステム再起動後にファイルの保護を開始します。 --- 例 --- > uwfmgr resistry add-exclusion HKLM¥Software¥Test
remove-exclusion <key>	指定されたレジストリキーを UWF のレジストリ除外リストから削除します。次回のシステム再起動後にファイルの保護を開始します。 --- 例 --- > uwfmgr resistry remove-exclusion HKLM¥Software¥Test
commit <key> [<value>]	指定されたレジストリキーと値の変更を値の指定がない場合は全ての値を反映します。 --- 例 --- > uwfmgr resistry commit HKLM¥Software¥Test TestValue
commit-delete <key> [<value>]	指定されたレジストリキーと値を削除し、削除を反映します。値の指定がない場合は、全ての値とサブキーを削除します。 --- 例 --- > uwfmgr resistry commit-delete HKLM¥Software¥Test TestValue

Uwfmgr [. exe] overlay [コマンド] [引数]
 オーバーレイの設定を構成します。

表 2-3-3-6. UWFMR Overlay コマンド一覧

コマンド	内容
help ?	オーバーレイ設定のコマンドラインヘルプを表示します。 --- 例 --- > uwfmgr overlay ?
get-config	UWF のオーバーレイの現在と次のセッションの構成設定を表示します。 --- 例 --- > uwfmgr overlay get-config
get-availablespace	現在のセッションで利用可能な残りの領域を表示します。 --- 例 --- > uwfmgr overlay get-availablespace
get-consumption	オーバーレイの使用されている現在のサイズを表示します。 --- 例 --- > uwfmgr overlay get-consumption
set-size <size>	オーバーレイの最大サイズを指定した値 (MByte 単位) に設定します。次のシステム再起動後に有効になります。 --- 例 --- > uwfmgr overlay set-size 1024
set-type {RAM DISK}	オーバーレイの種類を RAM または DISK で指定します。オーバーレイの種類を設定するためには現在のセッションで無効にする必要があります。 --- 例 --- > uwfmgr overlay set-type Disk
set-warningthreshold <size>	現在のセッションでドライバーが警告通知を発行するオーバーレイのサイズ (Mbyte 単位) を設定します。 --- 例 --- > uwfmgr overlay set-warningthreshold 256
set-criticalthreshold <size>	現在のセッションでドライバーが最大通知を発行するオーバーレイのサイズ (Mbyte 単位) を設定します。 --- 例 --- > uwfmgr overlay set-criticalthreshold 1024

Uwfmgr [. exe] servicing [コマンド] [引数]
サービスの設定を構成します。

表 2-3-3-7. UWFMR Servicing コマンド一覧

コマンド	内容
help ?	サービス設定のコマンドラインヘルプを表示します。 --- 例 --- > uwfmgr servicing ?
get-config	現在と次回のセッションのサービスの構成設定を表示します。 --- 例 --- > uwfmgr servicing get-config
enable	次回のセッションの設定で UWF サービスモードを有効にします。再起動後に有効になります。 このコマンドの使用には管理者権限が必要です。 --- 例 --- > uwfmgr servicing enable
disable	次回のセッションの設定で UWF サービスモードを無効にします。再起動後に有効になります。 このコマンドの使用には管理者権限が必要です。 --- 例 --- > uwfmgr servicing disable
update-windows	Windows の更新プログラムを適用します。 このコマンドの使用には管理者権限が必要です。 --- 例 --- > uwfmgr servicing update-windows

● アプリケーションからの UWF 操作

アプリケーションからは UWF を構成するために WMI providers for Unified Write Filter を使用することによって UWF の操作が可能です。

2-3-4 UWF Config Tool

「UWF Config Tool」は、UWF 状態表示、UWF 有効・無効設定、UWF オーバーレイ閾値設定を行うための常駐アプリケーションです。出荷状態ではスタートアップアプリケーションとして登録してあります。

OS が起動すると自動的に起動し、タスクトレイに常駐します。(図 2-3-4-1)

タスクトレイアイコンは UWF 有効・無効状態によって変化します。(図 2-3-4-2)

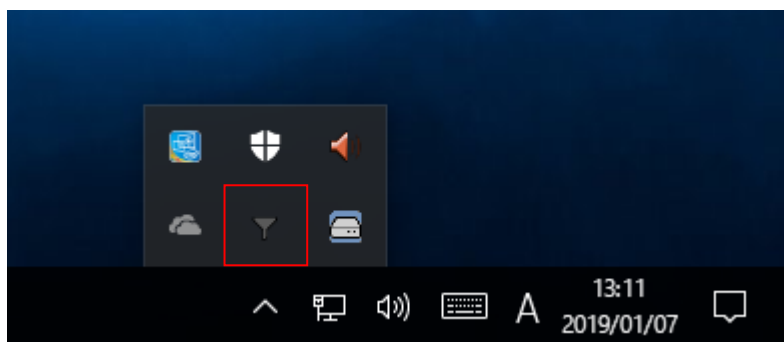


図 2-3-4-1. UWF Config Tool



図 2-3-4-2. UWF Config Tool タスクトレイアイコン

タスクトレイアイコンを右クリックするとメニューが表示され、UWF Config Tool 設定画面の表示、UWF Config Tool の終了を行うことができます。(図 2-3-4-3)

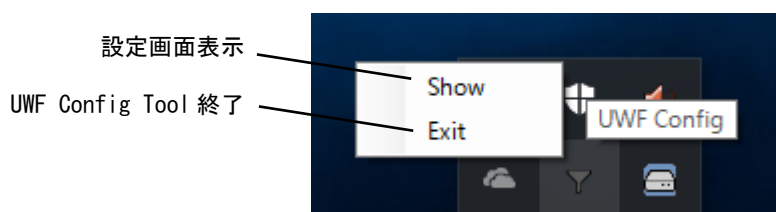


図 2-3-4-3. UWF Config Tool タスクトレイメニュー

「UWF Config Tool 設定画面」では、UWF 状態表示、UWF 有効・無効設定、UWF オーバーレイ閾値設定を行うことができます。（図 2-3-4-4）

※「UWF Config Tool」では、UWF 保護対象、除外対象の設定はできません。
保護対象、除外対象の設定方法は「2-3-3 UWF の設定方法」を参照してください。

The screenshot shows the 'UWF Config [v1.00]' window with the following fields and values:

- Current State:** UWF (Enable), Overlay Size (1024 [MB]), Overlay Consumption (219 [MB]), Available Space (805 [MB]).
- Next State:** UWF (Enable), Overlay Size (1024 [MB]).
- Overlay Threshold:** Warning Size (512 [MB]), Critical Size (1024 [MB]).

Annotations on the right side of the image:

- ① 現在状態 (Current State)
- ② 次回起動時状態 (Next State)
- ③ オーバーレイ閾値 (Overlay Threshold)

図 2-3-4-4. UWF Config Tool 設定画面

- ① 現在状態
現在の UWF 状態を表示します。

項目	内容
UWF	UWF の有効・無効状態 Disable:無効、Enable:有効
Overlay Size	オーバーレイサイズ [MB]
Overlay Consumption	オーバーレイ消費サイズ [MB]
Available Space	オーバーレイ利用可能残りサイズ [MB]

② 次回起動時状態

次回起動時の UWF 状態を表示します。

項目	内容
UWF	UWF の有効・無効状態 Disable:無効、Enable:有効
Overlay Size	オーバーレイサイズ [MB]

[Configure...] ボタンを押すと次回起動時状態設定を変更することができます。

項目	内容
UWF	UWF の有効・無効状態 Disable:無効、Enable:有効
Overlay Size	オーバーレイサイズ [MB] ※ 現在の UWF 状態が Enable の場合は変更できません

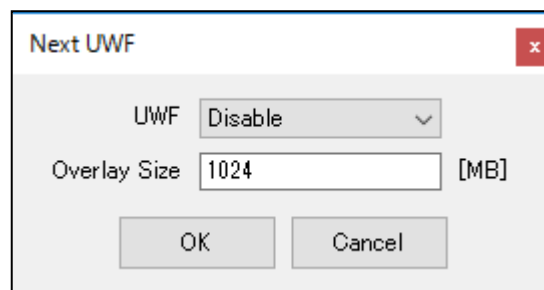


図 2-3-4-5. 次回起動時状態設定画面

③ オーバーレイ閾値

オーバーレイ閾値の状態を表示します。

項目	内容
Warning Size	警告閾値 [MB]
Critical Size	重大閾値 [MB]

[Configure...] ボタンを押すとオーバーレイ閾値設定を変更することができます。

項目	内容
Warning Size	警告閾値 [MB]
Critical Size	重大閾値 [MB]

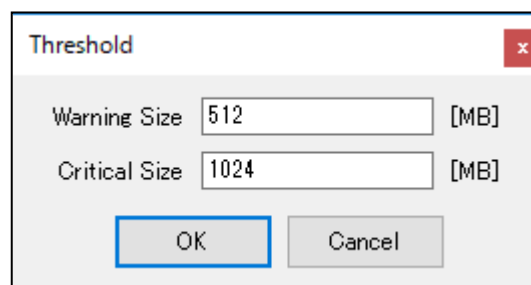


図 2-3-4-6. オーバーレイ閾値設定画面

2-3-5 UWF を使用するにあたっての注意事項

① UWF によるシステムメモリの消費

UWF はオーバーレイにシステムメモリを使用します。OS と UWF 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、UWF 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 UWF 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 UWF の消費メモリの解放

UWF 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、UWF 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。UWF 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。

UWF 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 UWF 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。

UWF オーバーレイのメモリ使用量は、「UWFMgr Overlay コマンド」で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 UWF 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 UWF 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 UWF 保護が無効なドライブへ変更することができます。表 2-3-5-1 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 2-3-5-1. 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 2-3-5-2 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 2-3-5-3 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 スタートボタンの右クリックから[コンピューターの管理]を選択します。
- ② [コンピューターの管理]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。

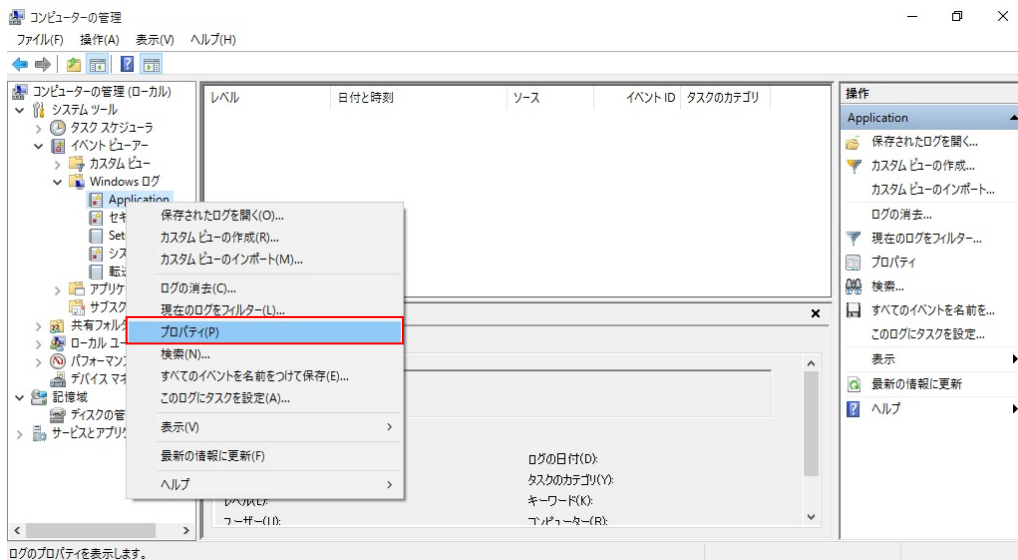


図 2-3-5-1. Windows ログのプロパティ選択

- 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。

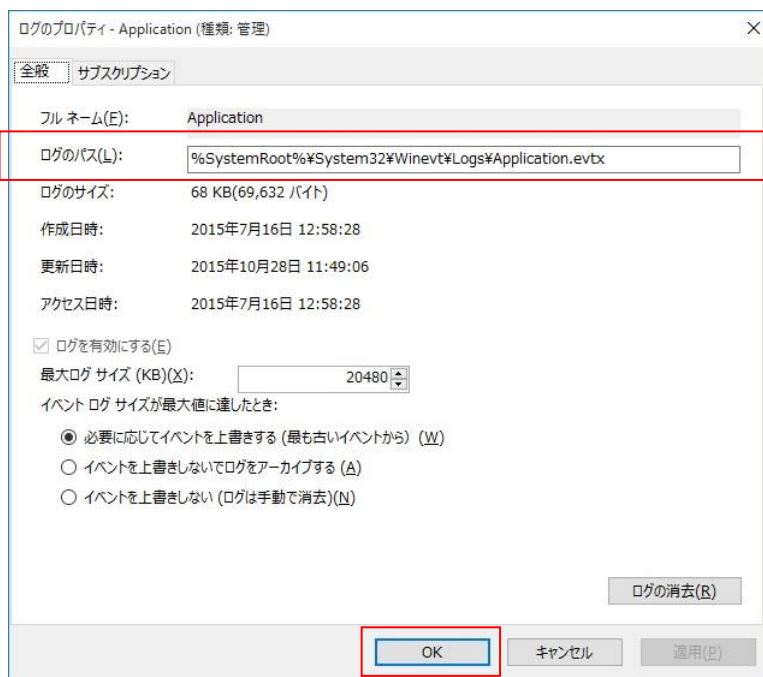


図 2-3-5-2. イベントログ出力先の変更

表 2-3-5-2. 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 2-3-5-3. 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>

⑤ アプリケーションでの注意

UWF 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 2-3-5-4 にアプリケーションでの注意事項を示します。

表 2-3-5-4. 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 UWF オーバーレイとして RAM を消費することになります。UWF 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 UWF 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」を参考に变更してください。

2-4 UPS 機能・RAM バックアップ機能

2-4-1 UPS 機能

産業用組込み PC には停電など突然の電源断対策のためにバッテリーが搭載されています。システムには、このバッテリーを使用した UPS (Uninterruptible Power Supply) 機能が組み込まれています。電源断が発生した場合、端末は最大 5 分間バッテリーで動作し、OS シャットダウン、ユーザーアプリケーションへのイベント通知などを安全に行うことができます。

電源断の検出、OS シャットダウン、イベント通知は UPS サービスが行います。UPS サービスの設定は「2-10 UPS Config Tool」を参照してください。UPS サービスの詳細は「4-14 UPS サービス」を参照してください。

※ 電源断発生から最大 5 分間はバッテリーで動作します。システムを保護するためには、5 分以内にシャットダウンが完了するようにしてください。

● UPS 機能の概要

- 電源断の検出
- バッテリーによる電源供給 (最大 5 分間)
- バッテリー異常の検出
- シャットダウンの実行
- ユーザーアプリケーションへの通知

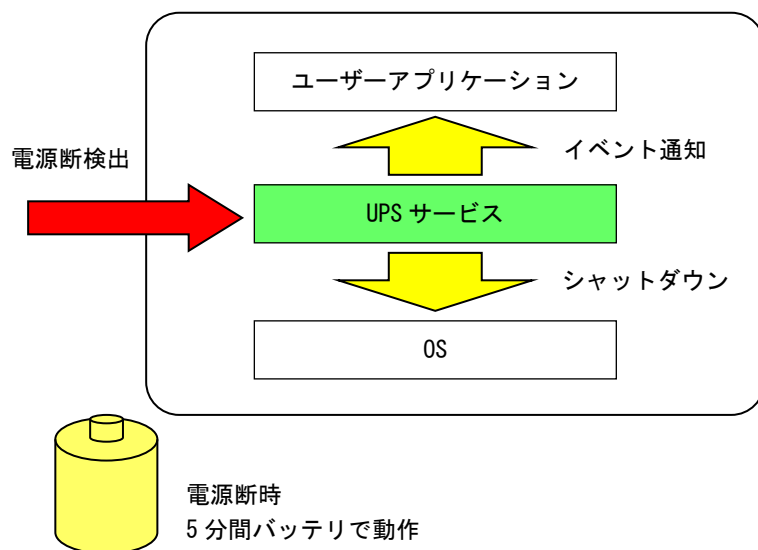


図 2-4-1-1. UPS 機能

2-4-2 RAM バックアップ機能

産業用組込み PC は、旧製品で搭載されていた「バックアップバッテリー付き SRAM」の代わりに、RAM バックアップ機能を搭載しています。

RAM バックアップ機能では、SRAM の代わりに OS システムメモリ上にメモリエリアを確保します。メモリエリアのデータはシャットダウン時にバックアップファイルとして保存されます。バックアップファイルの内容は、次回起動時にメモリエリアに展開されます。

シャットダウン時、起動時の処理は RAM バックアップサービスによって行われます。RAM バックアップサービスの設定は「2-10 UPS Config Tool」を参照してください。

メモリエリアはダミー-SRAM ドライバとして実装されています。ダミー-SRAM ドライバの詳細は「4-6 ダミー-SRAM」を参照してください。

● RAM バックアップ機能の概要

メモリエリア（ダミー-SRAM ドライバ：旧 SRAM ドライバ互換）

メモリエリアのデータをシャットダウン時にバックアップファイル（WRamBackup.dat）として保存
バックアップファイルのデータを起動時にメモリエリアに展開

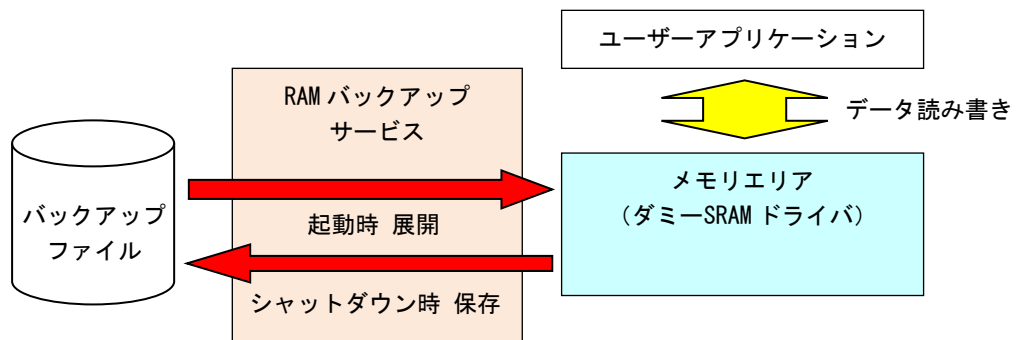


図 2-4-2-1. RAM バックアップ機能

UPS 機能と RAM バックアップ機能を併せて使用することで、電源断発生時もメモリエリアのデータを安全に保存することができます。

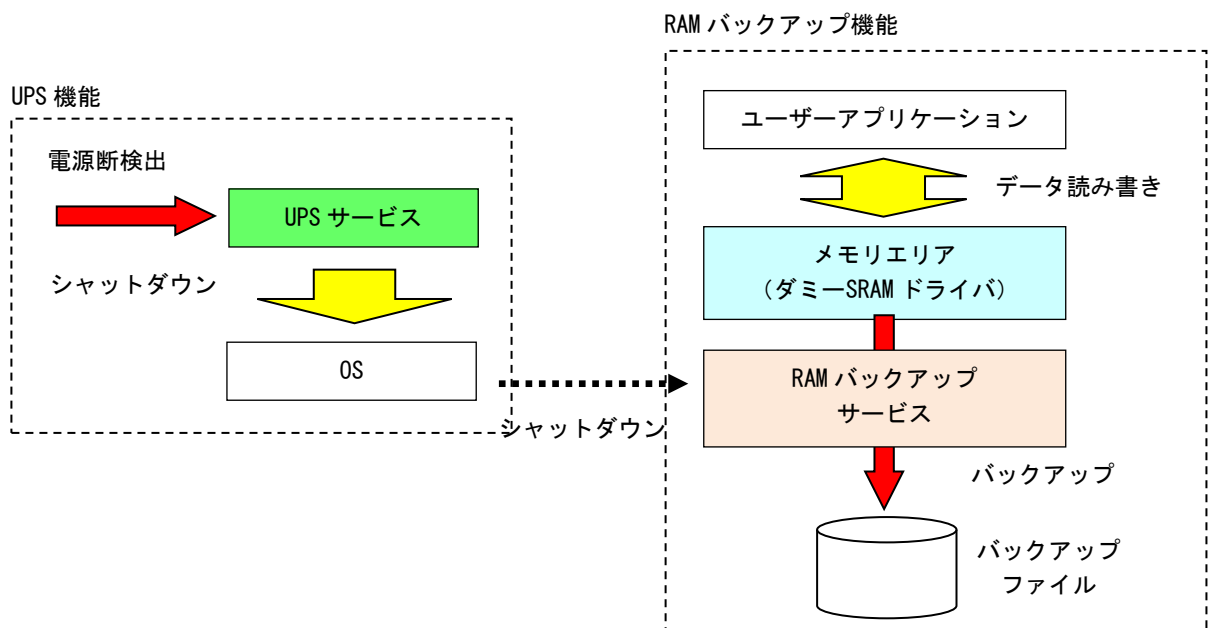


図 2-4-2-2. UPS 機能+RAM バックアップ機能

2-5 ログオン設定

2-5-1 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 スタートボタンの右クリックから [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。

2-6 言語設定

2-6-1 マルチ言語機能

Windows 10 IoT Enterprise は、マルチ言語対応 OS となっています。産業用組込み PC シリーズでは、英語と日本語の言語をインストールしています。英語と日本語以外の言語を使用したい場合は、Windows 10 の追加言語をダウンロードし、インストールすると、言語のコントロールパネルを使用して、メニュー、ダイアログボックス、その他のユーザー インターフェース項目を指定した言語で表示できるようになります。

初期状態では日本語環境で起動します。

2-6-2 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 スタートボタンの右クリックから[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。[言語]を実行します。[言語]ダイアログが開きます。
- ③ [言語の追加]を選択します。リストから[英語]を選択します。
- ④ リストから英語地域を選択し、追加を選択します。言語パックが無い場合はダウンロードします。
- ⑤ 先ほど追加した英語ランゲージパックが表示されますので、右側のオプションをクリックして、[言語のオプション]を表示します。[この言語を第一言語にします]に設定します。[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、表示言語が英語に変更されているので、次に表示言語以外を英語化します。スタートボタンの右クリックから[Control Panel]を選択し[Region]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、スタートボタンの右クリックから[Control Panel]を選択し[Region]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。

2-7 サービス設定

2-7-1 サービス設定の変更

産業用組込み PC シリーズ用 Windows 10 IoT Enterprise 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 2-7-1-1)

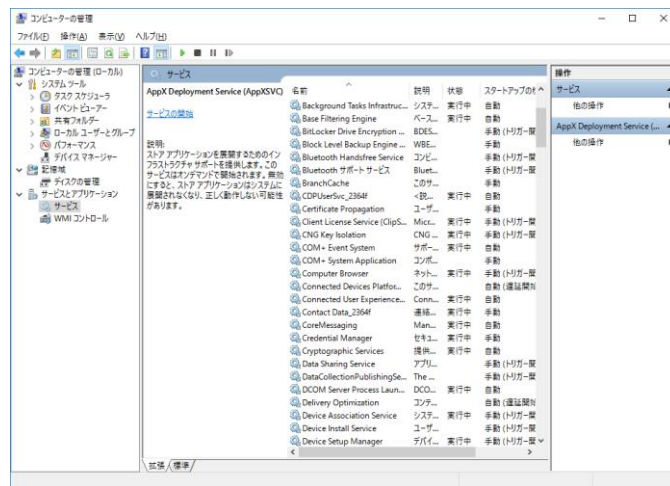


図 2-7-1-1. サービス設定画面

- 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 2-7-1-2)

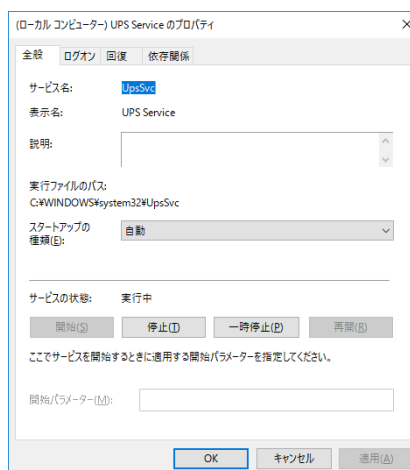


図 2-7-1-2. サービス設定画面

- 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。

2-8 ASD Config Tool

2-8-1 ASD Config Tool

「ASD Config Tool」は、産業用組込み PC 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 2-8-1-1 に示します。

表 2-8-1-1. 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 の切り替え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。

2-8-2 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。

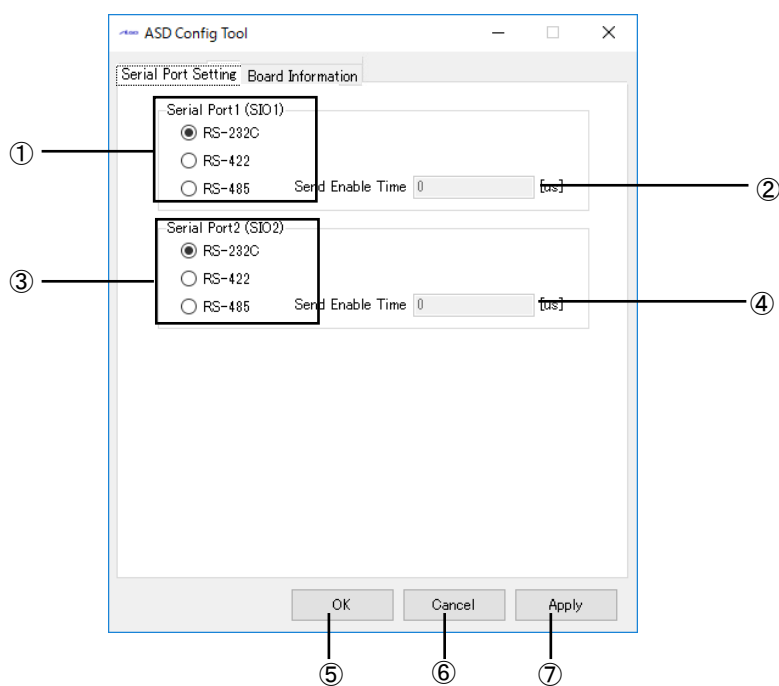


図 2-8-2-1. ASD Config – Serial Port Setting

- 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[μ s]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[μ s]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。

2-8-3 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。

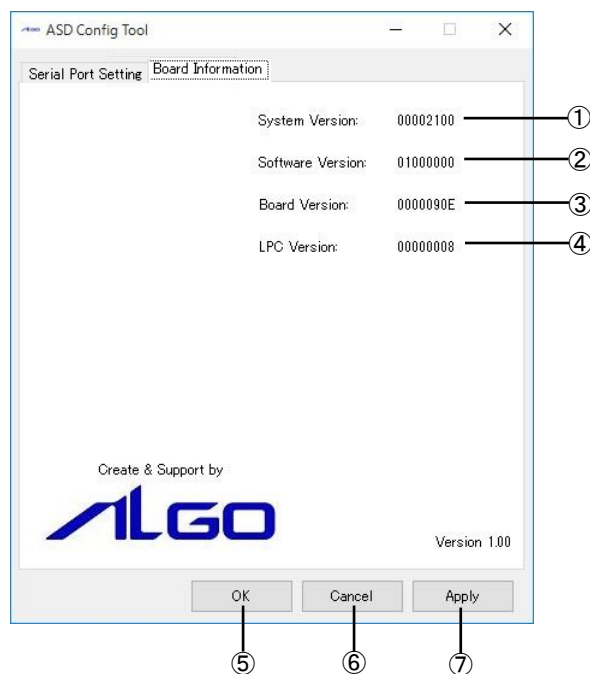


図 2-8-3-1. ASD Config – Board Information

- 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 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。

2-8-4 初期値

「ASD Config Tool」の設定初期値を表 2-8-4-1 に示します。

表 2-8-4-1. ASD Config Tool 設定初期値

タブ	設定項目	初期値
Serial Port Setting	Serial Port1 (SI01) Port Type	RS-232C
	Serial Port1 (SI01) Send Enabled Time	0
	Serial Port2 (SI02) Port Type	RS-232C
	Serial Port2 (SI02) Send Enabled Time	0

2-9 RAS Config Tool

2-9-1 RAS Config Tool

「RAS Config Tool」は、産業用組込み PC 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 2-9-1-1 に示します。

表 2-9-1-1. 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ーの状態を確認できます。

2-9-2 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 2-9-2-1、CPU Core 温度の設定内容を表 2-9-2-2 に示します。

表 2-9-2-1. 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 2-9-2-2. 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。
Temperature 1	基板上の温度を表示します。計測位置は基板バージョンや機種により異なります。
Temperature 2	基板上の温度を表示します。計測位置は基板バージョンや機種により異なります。

※ 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。

2-9-3 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。

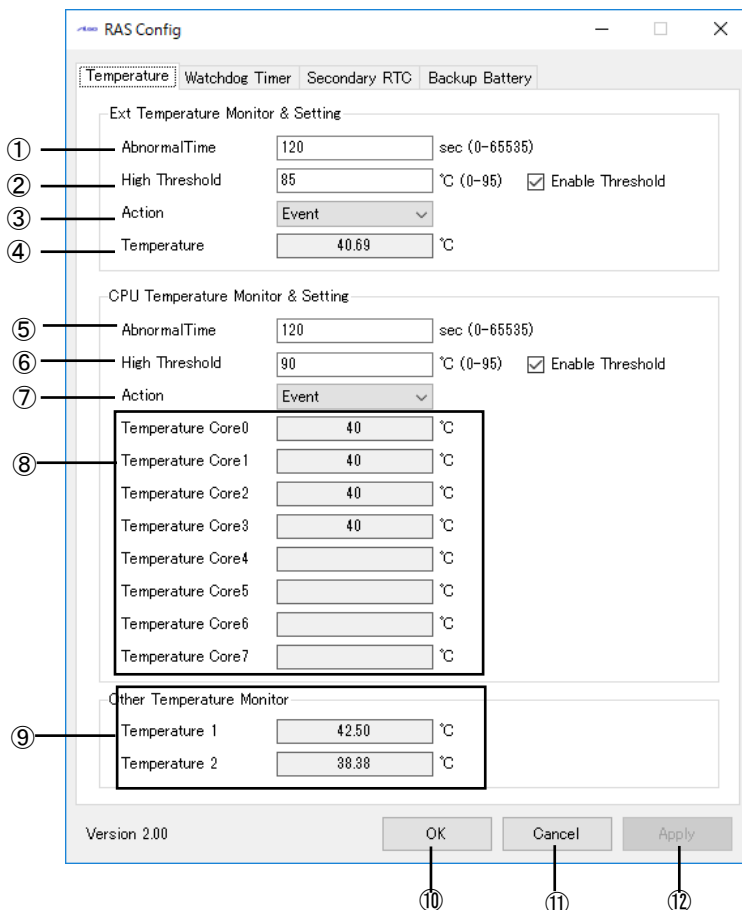


図 2-9-3-1. Temperature Configuration

- 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 基板の温度を表示します。
- ⑩ 設定を保存、適用して終了します。
- ⑪ 設定を破棄して終了します。
- ⑫ 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。

2-9-4 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 2-9-4-1、ソフトウェア・ウォッチドッグタイマの設定内容を表 2-9-4-2 に示します。

表 2-9-4-1. 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。 <ul style="list-style-type: none"> ・ 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 2-9-4-2. 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。 <ul style="list-style-type: none"> ・ 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・ウォッチドッグタイマには無い、「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witch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witch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。

「Power OFF」を選択する場合、電源オプション設定で POWER スwitch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 2-9-4-1)



図 2-9-4-1. 電源オプションの設定

2-9-5 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。

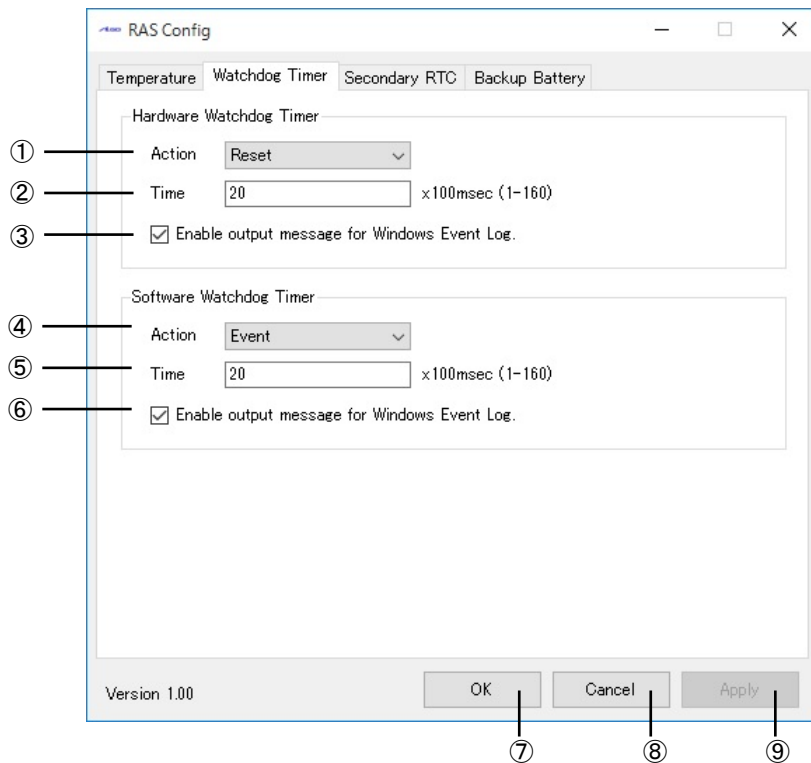


図 2-9-5-1. Watchdog Timer Configuration

- 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。

2-9-6 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 2-9-6-1 に示します。

表 2-9-6-1. 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)

2-9-7 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。

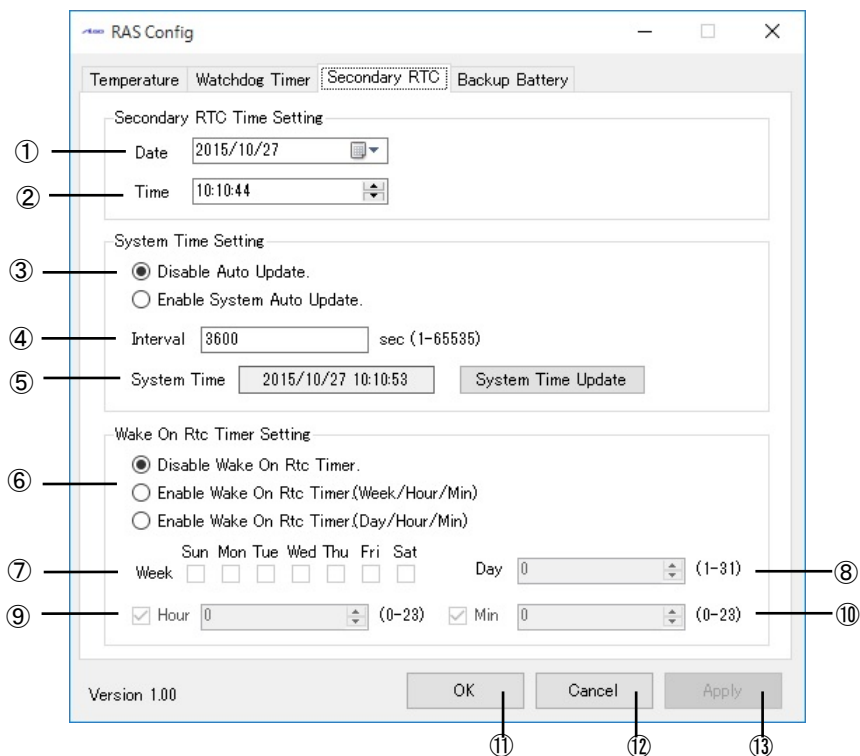


図 2-9-7-1. Secondary RTC Configuration

- 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。
- ⑦ ⑨⑩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。
- ⑧ ⑨⑩で設定した間隔で端末が起動します。
- ⑨ 曜日を設定します。(Enable Wake On Rtc Timer. (Week/Hour/Min) が ON の時のみ有効)
- ⑩ 日を設定します。(Enable Wake On Rtc Timer. (Day/Hour/Min) が ON の時のみ有効)
- ⑪ 時を設定します。
- ⑫ 分を設定します。
- ⑬ 設定を保存、適用して終了します。
- ⑭ 設定を破棄して終了します。
- ⑮ 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。

2-9-8 Wake On RTC Timer 設定例

Wake On RTC Timer の設定例を表 2-9-8-1、表 2-9-8-2 に示します。

表 2-9-8-1. 「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	チェックなし
毎週土・日 毎時 30 分 ※「時」不問	ON	OFF	OFF	OFF	OFF	OFF	ON	チェックなし	30
毎日 午後 6 時 59 分	ON	ON	ON	ON	ON	ON	ON	18	59

表 2-9-8-2. 「日」指定時の Wake On RTC Timer 設定例

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

2-9-9 Backup Battery Monitor

バックアップバッテリーの状態を確認することができます。

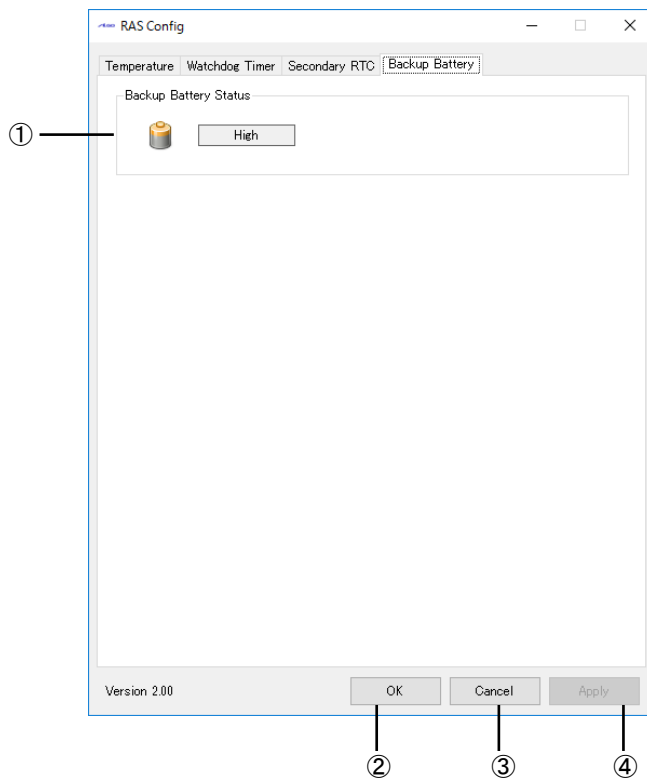


図 2-9-9-1. Backup Battery Monitor

- ① バックアップバッテリーの状態を表示します。(High・Low)
- 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。

2-9-10 初期値

「RAS Config Tool」の設定初期値を表 2-9-10-1 に示します。

表 2-9-10-1. 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

2-10 UPS Config Tool

2-10-1 UPS Config Tool

「UPS Config Tool」は、UPS 機能、RAM バックアップ機能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、[UPS Config]から起動できます。

「UPS Config Tool」では、UPS サービス設定、RAM バックアップサービス設定を行うことができます。UPS サービスの機能詳細は「4-14 UPS サービス」を参照してください。

2-10-2 UPS サービスの設定

UPS サービスの設定を行うことができます。

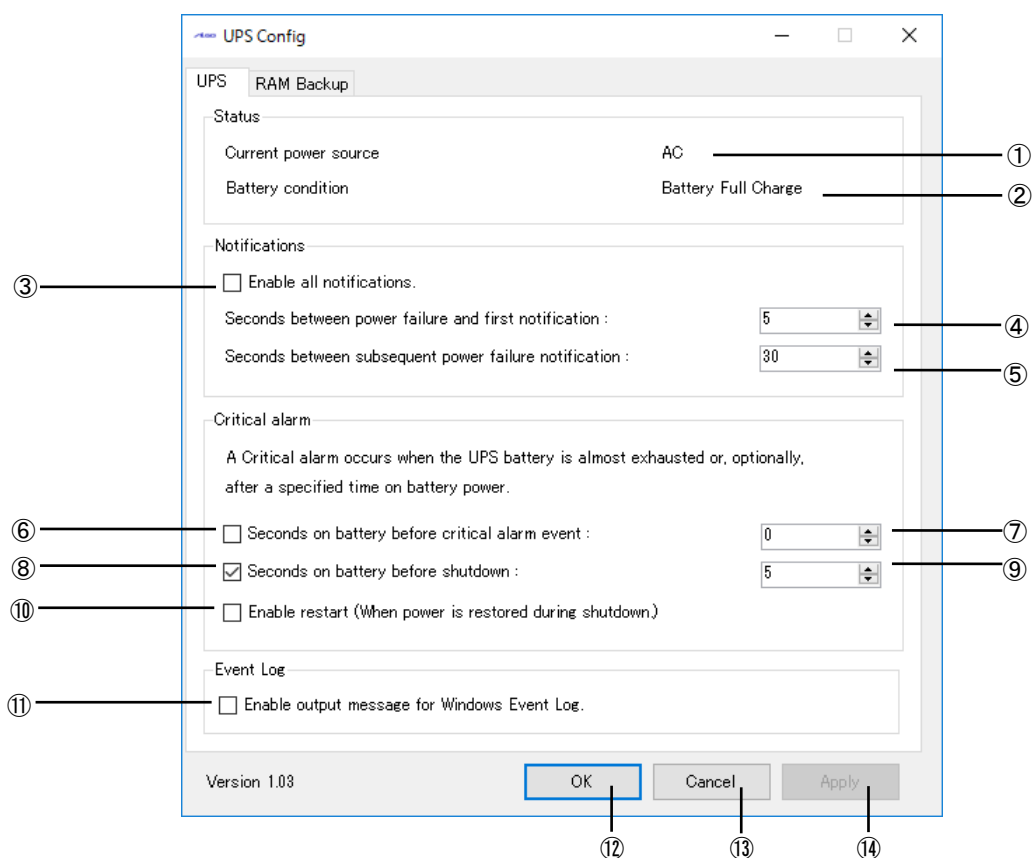


図 2-10-2-1. UPS

- ① 現在の電源を表示します。(AC : AC 電源駆動、BATTERY: バッテリ駆動*)
- ② バッテリの状態を表示します。
 - 0: Battery Charging (バッテリー充電中)
 - 1: Battery Full Charge (バッテリー満充電)
 - 2: Battery Discharging (バッテリー放電中)
 - 3: Battery Error (バッテリー異常)

- ③ バッテリ異常通知イベントの有効/無効を設定できます。
- ④ バッテリ異常が発生してから通知イベントを送信するまでの時間を設定します。(0~120(秒))
- ⑤ ④の通知後の通知間隔を設定します。(5~300(秒))
- ⑥ バッテリ駆動になってから一定時間経過すると警告イベントを発行する/しないを設定できます。
- ⑦ バッテリ駆動開始から警告イベントを送信するまでの時間を設定します。(0~60(秒))
- ⑧ バッテリ駆動になってから一定時間経過するとシステムをシャットダウンする/しないを設定できます。
- ⑨ バッテリ駆動開始からシャットダウンするまでの時間を設定します。(0~60(秒))
- ⑩ 電源断によるシャットダウン中に電源が復帰した場合の設定(チェックすると再起動します)
- ⑪ UPS 機能によるシャットダウン中に電源が復帰した場合、再起動する/しないを設定できます。通知イベント、警告イベント発生時にイベントログにメッセージ出力の設定ができます。
- ⑫ 設定を保存、適用して終了します。
- ⑬ 設定を破棄して終了します。
- ⑭ 設定を保存、適用します。

2-10-3 RAM Backup

RAM バックアップサービスの設定を行うことができます。

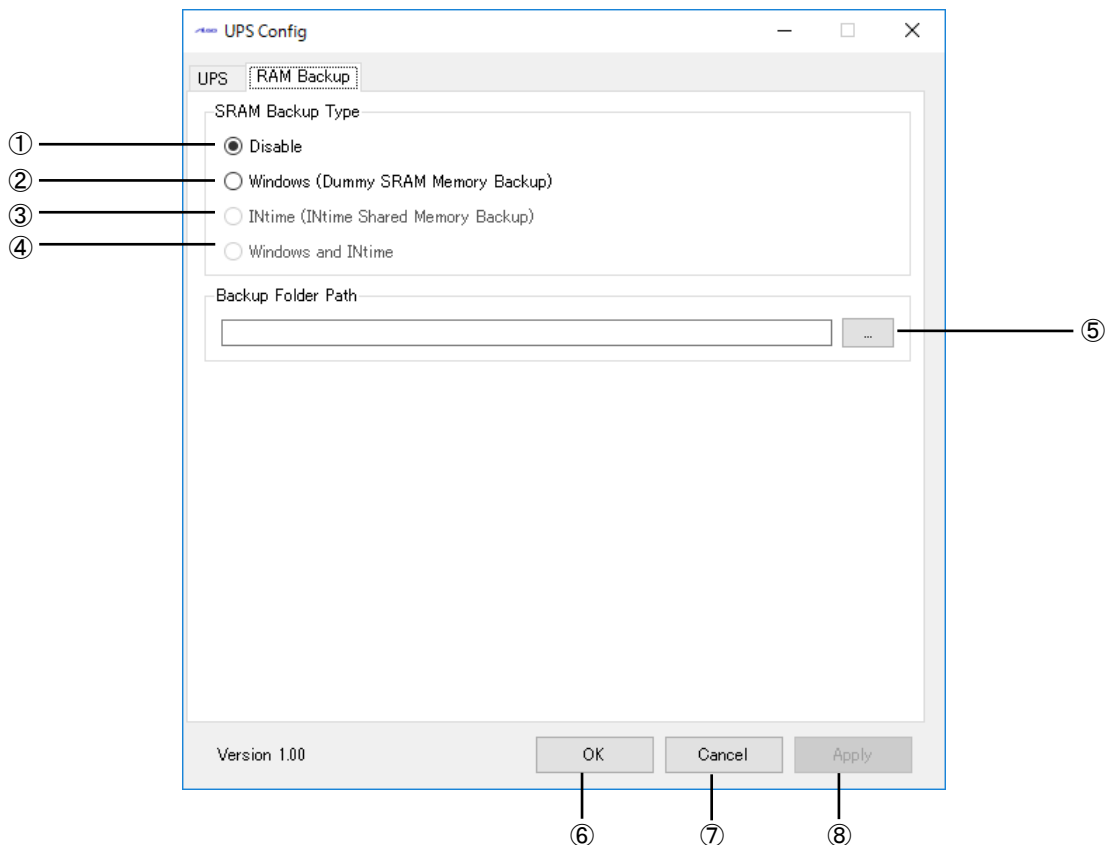


図 2-10-3-1. RAM Backup

- ① RAM バックアップ機能を無効にします。
- ② RAM バックアップ機能を使用します。(ダミー-SRAM ドライバ)
RAM バックアップ機能では 150Mbyte のデータを保存することができます。
- ③ INtime バックアップ機能を使用します。
INtime バックアップ機能では 4Mbyte のデータを保存することができます。
INtime がバンドルされていない環境では選択できません。
動作については、「INtime 省配線導入マニュアル」を参照してください。
- ④ RAM バックアップ機能と INtime バックアップ機能を両方使用します。
INtime がバンドルされていない場合は選択できません。
動作については、「INtime 省配線導入マニュアル」を参照してください。
- ⑤ バックアップファイルを保存するフォルダを指定します。
バックアップファイルは、「設定フォルダ」+「WRamBackup.dat」となります。
 <設定フォルダ>\WRamBackup.dat
無効なフォルダ名を指定した場合や、空白の場合はバックアップファイルを保存しません。
- 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。

2-10-4 UPS を使用するための注意事項

[UPS Config Tool]-[UPS]にて変更した設定を有効にするには端末本体もしくは UPS サービスを再起動する必要があります。

UPS サービスを再起動するには、

- ① コントロールパネルの「管理ツール」をダブルクリックしコンピュータの管理を開きます。
- ② 「サービス」をクリックし、サービスの一覧を開きます。(図 2-10-4-1)

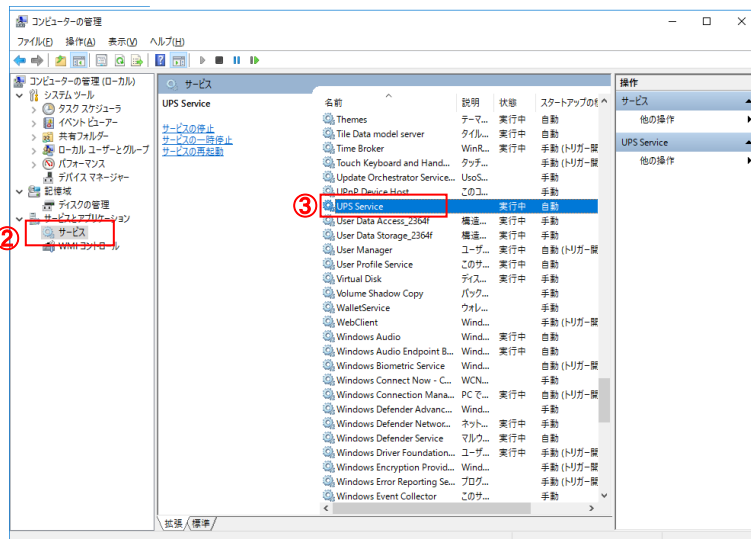


図 2-10-4-1. サービスの一覧

- ③ 「UPS Service」をダブルクリックしプロパティを開きます。
- ④ 「停止」ボタン(*1)をクリックして停止させた後、「開始」ボタン(*2)をクリックして UPS サービスを再起動します。(図 2-10-4-2)

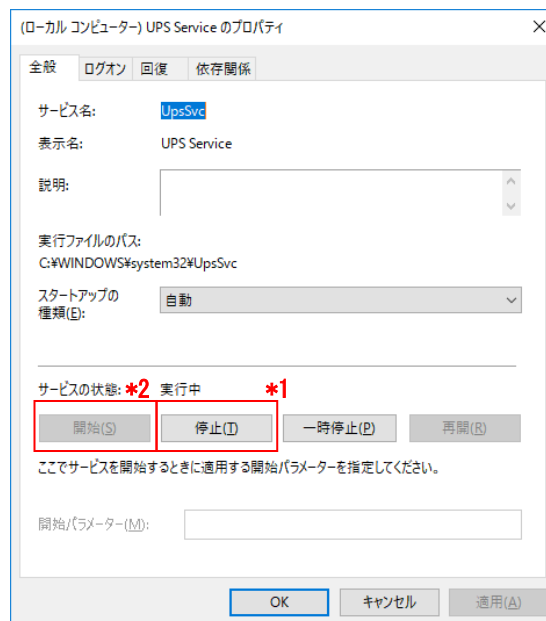


図 2-10-4-2. UPS Configuration Service のプロパティ

- ⑤ 「UPS Service」のスタートアップの種類が「自動」に、且つサービスの状態が「開始」になっていることを確認します。

2-10-5 初期値

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

表 2-10-5-1. UPS Config Tool 設定初期値

タブ	設定項目	初期値
UPS	All notifications	OFF
	Power failure and first notification	5sec
	Subsequent power failure notification	10sec
	Battery before critical alarm event	OFF / 0sec
	Battery before shutdown	ON / 5sec
	Output message for Windows Event Log	OFF
RAM Backup	RAM Backup Type	Disable ※1
	Backup File Path	なし ※2

※1 オールインワン版では初期値は「Windows and Intime」です。

※2 オールインワン版では初期値は「D:¥」です。

2-1-1 ユーザーアカウント制御

Windows 10 IoT Enterprise 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 2-11-1 に示します。

表 2-11-1. 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	<ul style="list-style-type: none"> ・ 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 スタートボタンの右クリックから [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。

2-12 S.M.A.R.T.機能

S. M. A. R. T. は、m-SATA 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。

産業用組込み PC シリーズ用 Windows 10 IoT Enterprise には、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。

2-12-1 iSMART

「iSMART」は、S. M. A. R. T. 情報を参照することができます。mini m-SATA の状態、おおよその寿命を予測することなどができます。

[スタートメニューのタイル]→[iSMART]で起動できます。

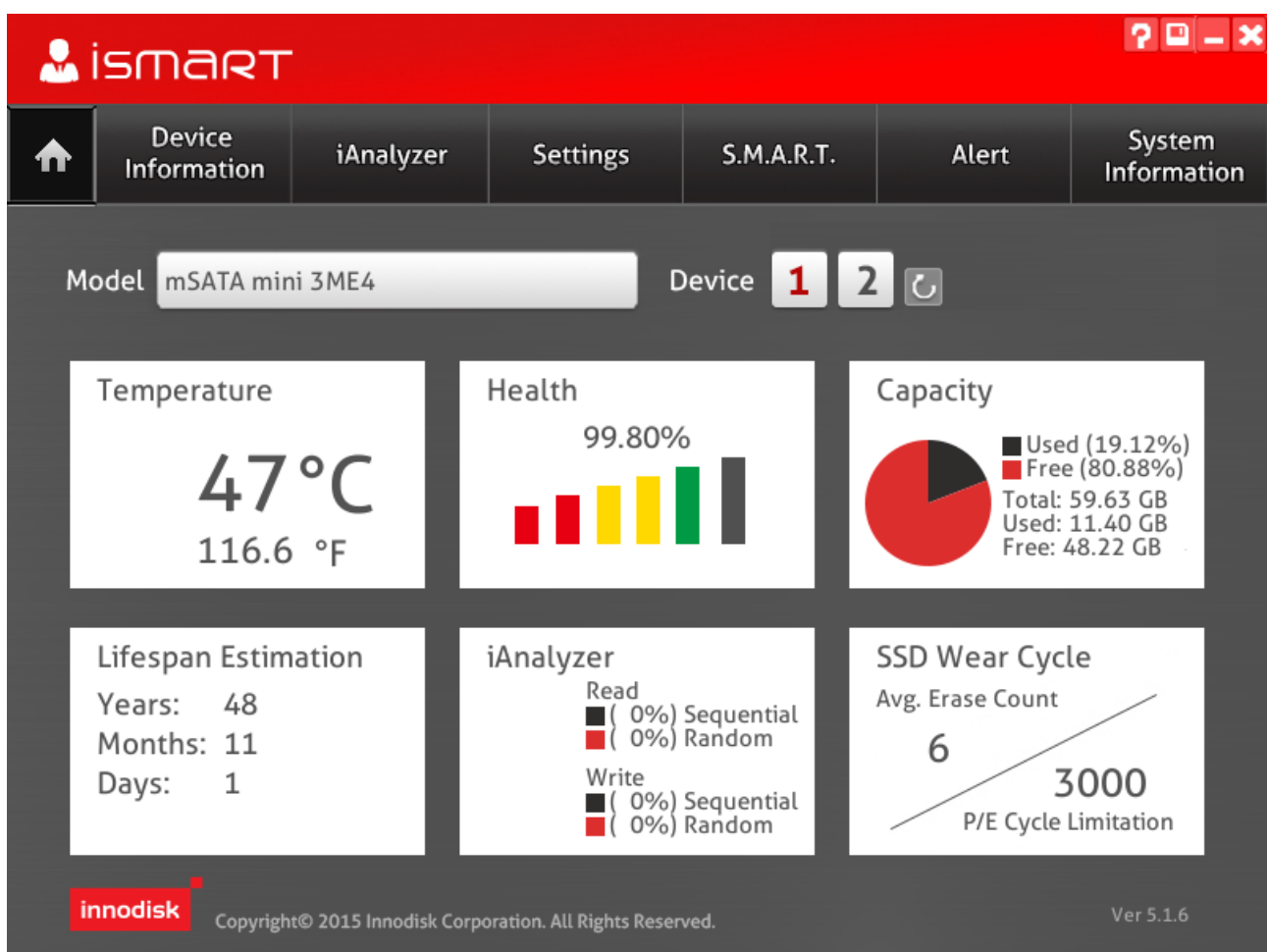


図 2-12-1-1. iSMART 画面

2-12-2 S.M.A.R.T. Monitor Tool

「S.M.A.R.T. Monitor Tool」は、S.M.A.R.T. 情報の「平均消去回数」、「不良ブロック」を監視するための常駐アプリケーションです。出荷状態ではスタートアップアプリケーションとして登録してあります。

OS が起動すると自動的に起動し、タスクトレイに常駐します。(図 2-12-2-1)

タスクトレイアイコンは異常状態によって変化します。(図 2-12-2-2)

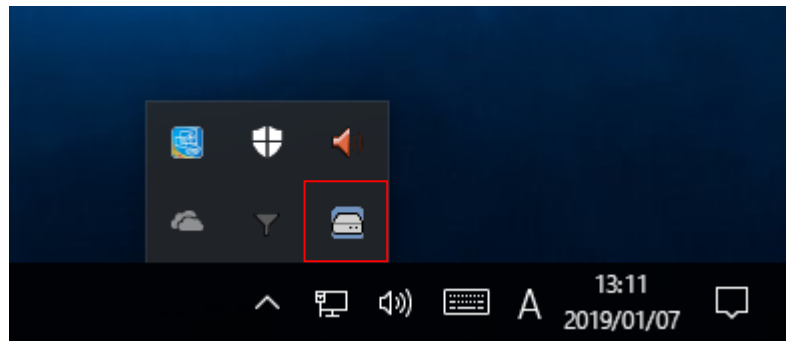


図 2-12-2-1. S.M.A.R.T. Monitor Tool



図 2-12-2-2. S.M.A.R.T. Monitor Tool タスクトレイアイコン

タスクトレイアイコンを右クリックするとメニューが表示され、S.M.A.R.T. Monitor Tool 設定画面の表示、S.M.A.R.T. Monitor Tool の終了を行うことができます。(図 2-12-2-3)

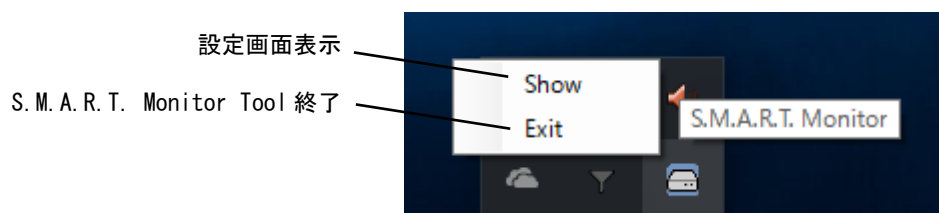


図 2-12-2-3. S.M.A.R.T. Monitor Tool タスクトレイメニュー

S.M.A.R.T. Monitor Tool 設定画面では、監視項目の現在値の確認、警告レベル異常・重大レベル異常検知時の通知方法、監視周期の設定を行うことができます。(図 2-12-2-4)

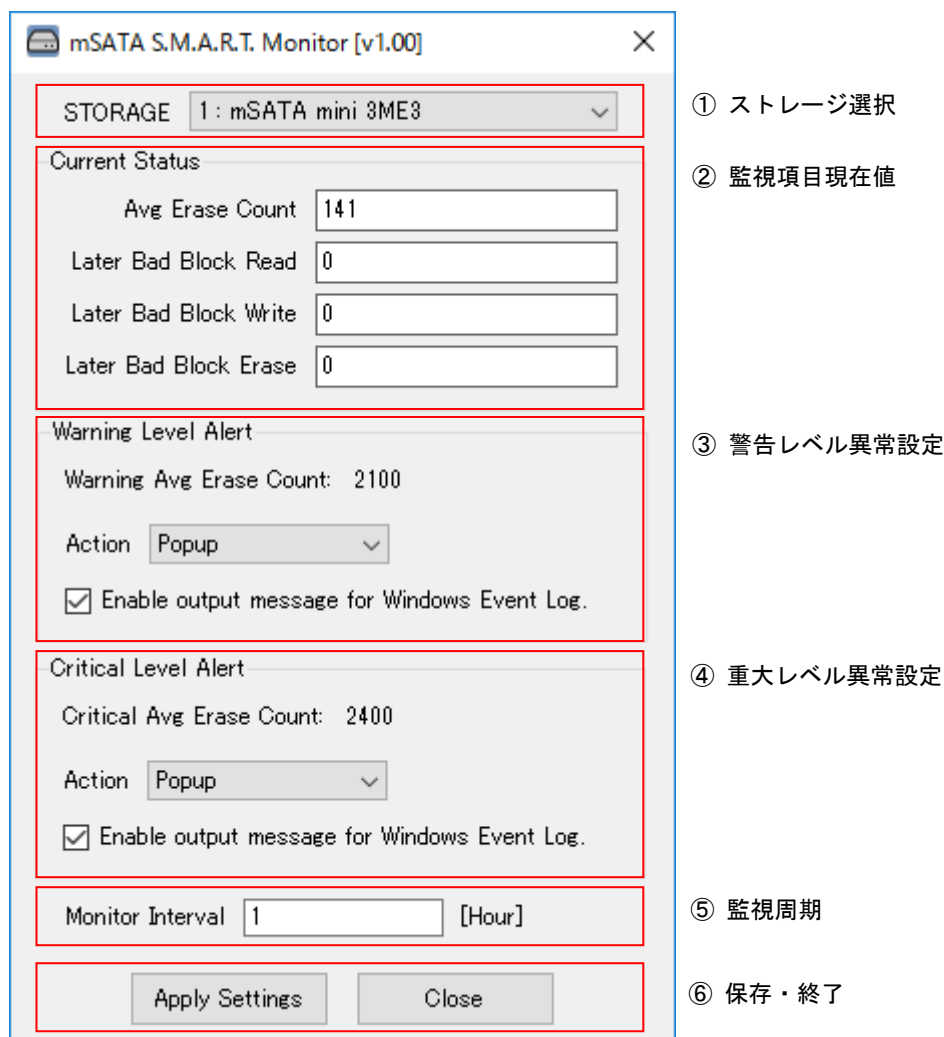


図 2-12-2-4. S.M.A.R.T. Monitor Tool 設定画面

① ストレージ選択

監視対象のストレージを選択します。

選択したストレージの状態・設定を②～⑤に表示します。

設定編集集中にストレージ選択を変更した場合は、編集集中の内容が破棄されます。

② 監視項目現在値

監視項目の現在の値を表示します。

項目	内容
Avg Erase Count	平均消去回数 回数に従って、警告レベル異常、重大レベル異常を判断します
Later Bad Block Read	不良ブロック数（読み込み） 1つでもあれば重大レベル異常となります
Later Bad Block Write	不良ブロック数（書き込み） 1つでもあれば重大レベル異常となります
Later Bad Block Erase	不良ブロック数（消去） 1つでもあれば重大レベル異常となります

③ 警告レベル異常設定

平均消去回数が警告レベル回数以上となった場合に警告レベル異常となります。

警告レベル異常が発生した場合の動作を設定することができます。

項目	内容
Action	通知方法 Disable : 通知なし Popup : ポップアップウインドウを表示 Event : イベント通知
Enable output message for Windows Event Log	Windows イベントログ記録 チェックすると異常発生を Windows イベントログに記録します

※ 警告レベル回数は m-SATA の種類によって異なります。

④ 重大レベル異常設定

平均消去回数が重大レベル回数以上となった場合、不良ブロックを検出した場合に警告レベル異常となります。

重大レベル異常が発生した場合の動作を設定することができます。

項目	内容
Action	通知方法 Disable : 通知なし Popup : ポップアップウインドウを表示 Event : イベント通知
Enable output message for Windows Event Log	Windows イベントログ記録 チェックすると異常発生を Windows イベントログに記録します

※ 重大レベル回数は m-SATA の種類によって異なります。

⑤ 監視周期

ストレージ状態監視の周期を設定します。

項目	内容
Monitor Interval	ストレージ監視周期 1～

⑥ 保存・終了

[Apply Settings] ボタン : 現在の設定内容を確定します。

[Close] ボタン : S.M.A.R.T Monitor Tool 設定画面を閉じます。編集中の設定は破棄されます。

第 3 章 産業用組込み PC 4A IoT について

本章では、産業用組込み PC 4A IoT シリーズ（産業用組込み PC 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 産業用組込み PC 4A IoT* に搭載された機能について

産業用組込み PC 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、産業用組込み PC 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。

各機種別の 1 例として、EC4A-010CT、EC4A-110CA の外形図を以下に示します。他の機種については、ハードウェアのマニュアルを参照してください。

各部名称を表 3-1-1 に示します。

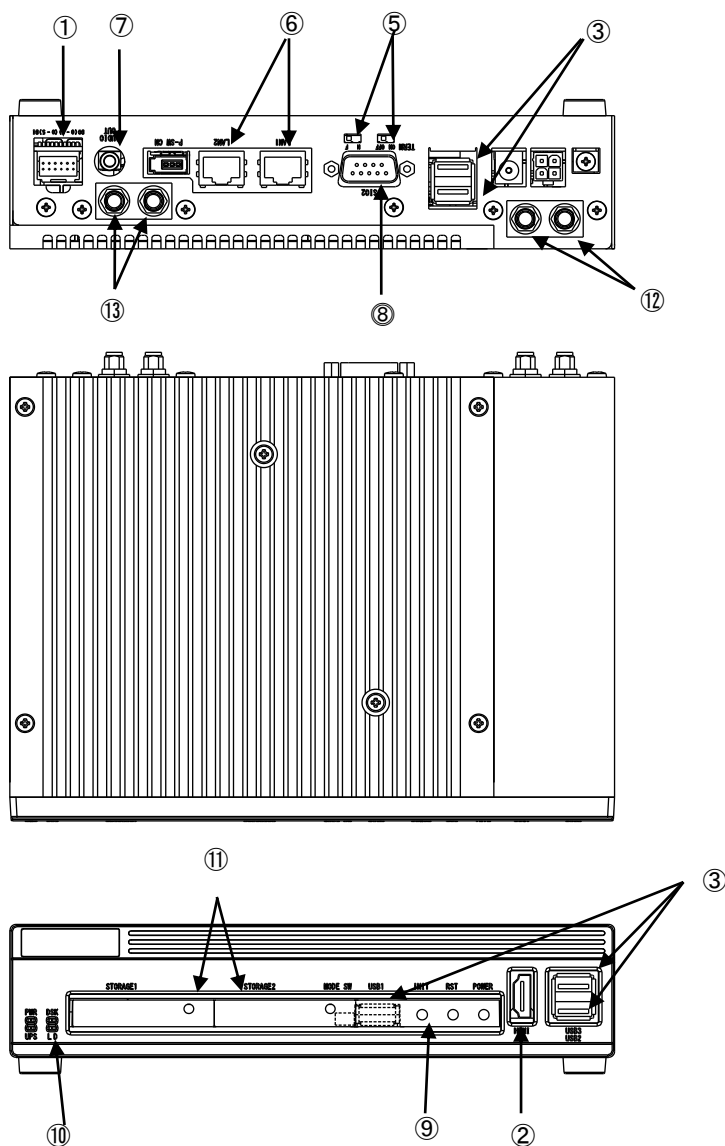


図 3-1-1. 外形図 (EC4A-010CT)

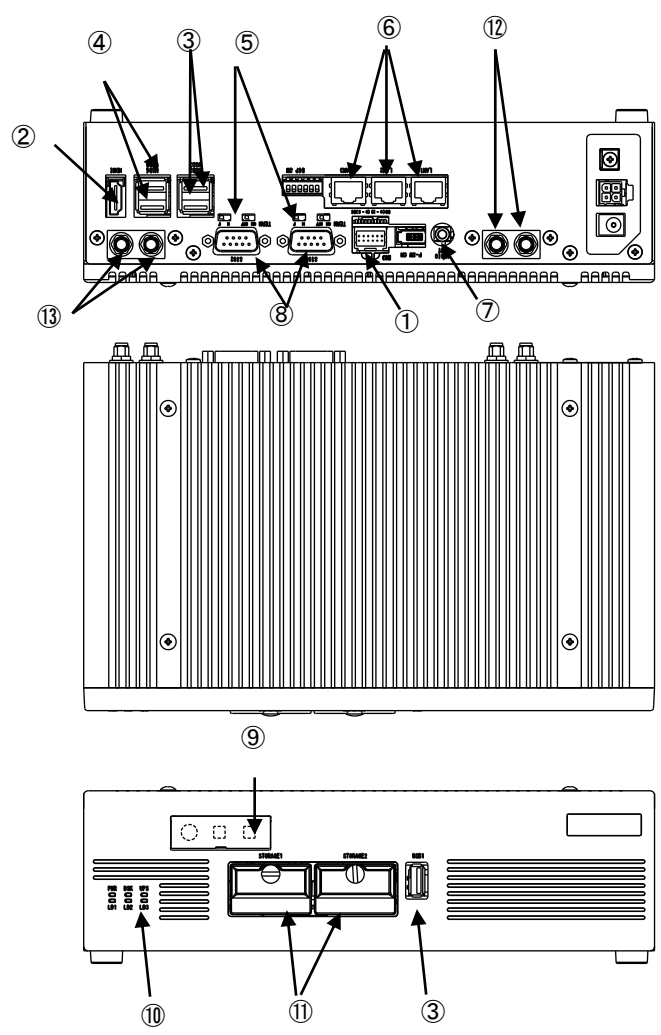


図 3-1-2. 外形図 (EC4A-110CA)

表 3-1-1. 各部名称

No.	名称	機能	説明	実装数		
				010CT	100 CA/CT	110 CA/CT
①	DIO インターフェース	汎用入出力	汎用の入出力です。 入力 6 点、出力 4 点を制御できます。	1	1	1
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。	1	1	1
③	USB2.0 インターフェース	USB2.0 ポート	USB1.1/2.0 の機器を接続することができます。	5	3	3
④	USB3.0 インターフェース	USB3.0 ポート	USB1.1/2.0/3.0 の機器を接続することができます。	-	2	2
⑤	シリアル 設定スイッチ	シリアルポート タイプ設定	SI01、SI02 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、COM1、COM2 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。	2	2	2
⑥	ネットワーク インターフェース	有線 LAN	ネットワークポートとして使用できません。	2	3	3
⑦	サウンド	サウンド	音声出力が使用できます	1	1	1
⑧	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。 SI01: COM1 SI02: COM2	1	2	2
⑨	初期化スイッチ	スイッチ	電源投入時、初期化スイッチを 3 秒間押すことで、LD1 が点滅します。また、アプリケーションにより電源投入時にスイッチが押されたかどうかを確認できます。	1	1	1
⑩	汎用 LED	LED	アプリケーションにより 3 点の LED の点灯/消灯を制御できます。 LD1: 電源投入後、SW1 を 3 秒間長押しすると点滅します。 LD2: アプリケーションで制御できます。 LD3: アプリケーションで制御できません。	1	1	1

⑪	mini m-SATA スロット	mini m-SATA SSD	記憶領域として mini m-SATA SSD を使用することができます。 mini m-SATA1: 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。 mini m-SATA2: 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。	1 (1)	1 (1)	1
	2.5 インチ HDD	2.5 inch HDD	記憶領域として 2.5 inch HDD を使用することができます。	-	-	1
⑫	LTE(外部アンテナ) (オプション)	LTE 通信	LTE 通信を使用することができます。	(2)	(2)	(1)
⑬	無線 LAN(外部アンテナ) (オプション)	無線 LAN	無線ネットワークデバイスとして使用できます。	(2)	(2)	(1)
	Bluetooth (オプション)	Bluetooth 通信	Bluetooth 機器を無線接続することができます。	(1)	(1)	(1)

実装数の () はオプションを表します。

3-2 Windows 標準インターフェース対応機能

本項では、産業用組込み PC 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 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、COM2 が使用可能です。搭載されている COM ポートの一覧を表 3-2-2-1 に示します。

表 3-2-2-1. シリアルポート

COM ポート	説明
COM1	SI01 (図 3-1-2 ⑧) 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	SI02 (図 3-1-2 ⑧) 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	アプリケーションでシリアル通信に使用できません。

3-2-3 有線 LAN

産業用組込み PC 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 3-2-3-1 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 3-2-3-1. 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 3-1-2 ⑥): 1000BASE Ethernet
ローカルエリア接続 2	LAN2 (図 3-1-2 ⑥): 1000BASE Ethernet
ローカルエリア接続 3	LAN3 (図 3-1-2 ⑥): 1000BASE Ethernet

3-2-4 サウンド

サウンド機能として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かオーディオにスピーカ機器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。

3-2-5 USB2.0 ポート

USB1.1/2.0 対応の USB ポートを外部コネクタとして 2 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。

3-2-6 USB3.0 ポート

USB1.1/2.0/3.0 対応の USB ポートを外部コネクタとして 1 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。

3-2-7 LTE(オプション)

EC4A シリーズはオプションで LTE を搭載することができます。一般的な Windows と同様にモバイルデータ通信を利用することができます。

LTE の利用には、別途、回線契約と micro-SIM カードが必要になります。

3-2-8 無線 LAN (オプション)

EC4A シリーズはオプションで無線 LAN を搭載することができます。一般的な Windows と同様に無線ネットワークポートとして使用することができます。

3-2-9 Bluetooth (オプション)

EC4A シリーズはオプションで Bluetooth を搭載することができます。一般的な Windows と同様に Bluetooth 機器を接続して利用することができます。対応するプロファイル一覧を表 3-2-9-1 に示します。

表 3-2-9-1. 対応プロファイル一覧

プロファイル名	説明
GATT	一般属性プロファイル
A2DP 1.2	高度オーディオ配信プロファイル
AVRCP 1.3	オーディオ/ビデオ リモート制御プロファイル
DI 1.3	デバイス識別プロファイル
DUN 1.1	ダイヤルアップ ネットワーク プロファイル
HCRP 1.0	ハードコピー ケーブル置換プロファイル
HFP 1.5	ハンズフリー プロファイル
HID 1.1	ヒューマン インターフェイス デバイス プロファイル
HOGP 1.0	マウスやキーボードなどを接続するためのプロファイル
OPP 1.1	オブジェクト プッシュ プロファイル
PANU 1.0	パーソナルエリア ネットワーク プロファイル
SPP 1.2	シリアルポート プロファイル

3-3 組込みシステム機能

産業用組込み PC 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。

産業用組込み PC シリーズ用 Windows 10 IoT Enterprise 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 4 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 ①)
RAS 機能	汎用入力の IN0、IN1 にリセット機能、割込み機能があります。IN0 リセット、IN1 割込みの制御ができます。
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。
ダミー-SRAM	ダミー-SRAM を制御することができます。
初期化スイッチ	スイッチの状態を読み出すことができます。
汎用 LED	LED の ON/OFF の制御を行うことができます。
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。
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。
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。
ビープ音	ビープ音を制御できます。 ビープ音の ON/OFF、周波数の変更ができます。
バックアップバッテリーモニタ	BIOS 設定、RTC、外部 RTC に使用されるバックアップバッテリーの状態を取得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。
UPS サービス	電源断が発生した場合に OS シャットダウンやユーザアプリへのイベント通知を安全に行うことができます。

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。

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 ダミー-SRAM

RAM バックアップ機能で保護されるメモリエリアを提供します。

RAM バックアップ機能の詳細は「2-4-2 RAM バックアップ機能」を参照してください。

3-3-6 初期化スイッチ

1 点の初期化スイッチが実装されています。

アプリケーションからスイッチの状態を確認することができます。表 3-3-6-1 に初期化スイッチの機能を示します。

表 3-3-6-1. 初期化スイッチ機能

初期化スイッチ	機能
初期化スイッチ	初期化スイッチは、電源 ON 直後に 3 秒間長押しすると、ON 状態でラッチされます。初期化スイッチがラッチ状態の場合、LD1 は点滅状態になります。汎用 LED を操作し LD1 の点滅状態が解除されると、初期化スイッチは OFF 状態になります。

3-3-7 汎用 LED

3 点の汎用 LED が実装されています。

アプリケーションから LED の点灯/消灯を制御可能です。表 3-3-7-1 に各 LED の機能を示します。

表 3-3-7-1. 汎用 LED 機能

汎用 LED	機能
LD1	初期化スイッチが ON ラッチしている間、LD1 は点滅します。 アプリケーションで点灯/消灯を操作することができます。 LD1 が点滅状態のとき LD1 を操作すると、点滅状態は解除され LD1 は指定した状態へと変化します。 LD1 の点滅状態が解除されると初期化スイッチは OFF 状態になります。
LD2	アプリケーションで点灯/消灯を操作することができます。 LD1 が点滅状態のとき LD2 を操作すると、LD1 の点滅状態は解除されます。
LD3	アプリケーションで点灯/消灯を操作することができます。 LD1 が点滅状態のとき LD3 を操作すると、LD1 の点滅状態は解除されます。

3-3-8 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。

3-3-9 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。

3-3-10 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、「2-9 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 バックアップバッテリーモニタ

BIOS、RTC、外部 RTC のデータを保持するためのバックアップバッテリー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ー状態は、「High」・「Low」を確認することができます。
UPS 搭載モデルでは BIOS、RTC、外部 RTC のデータ保持は UPS バッテリーが行います。
UPS 搭載モデルでは OS 起動中は「High」となります。**

3-3-13 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、「2-9 RAS Config Tool」を参照してください。

3-3-14 UPS 機能

電源断時にシャットダウンやユーザアプリへのイベント通知を行うことができる UPS 機能が実装されています。

シャットダウンやイベント通知までの待機時間などを設定できます。UPS 機能の設定の詳細は「2-10 UPS Config Tool」を参照してください。

第 4 章 組込みシステム機能ドライバ

産業用組込み PC 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。産業用組込み PC シリーズ用 Windows 10 IoT Enterprise 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。

4-1 ドライバの使用について

4-1-1 開発用ファイル

「産業用組込み PC シリーズ用 Windows 10 IoT Enterprise 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うことが可能です。DVD に含まれる開発用ファイルの内容を表 4-1-1-1 に示します。

表 4-1-1-1. 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	タイマ割り込み機能 IN1 割り込み機能サンプルコードです。
¥SDK¥Algo¥Sample¥Sample_Reset	INO 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_RamBackup	ダミー-SRAM のサンプルコードです。
¥SDK¥Algo¥Sample¥Sample_SwLed	汎用 SW、汎用 LED 制御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_Beep	ビーブ音制御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ーモニタのサンプルコードです。
¥SDK¥Algo¥Sample¥Sample_UPS	UPS 機能のサンプルコードです。
¥SDK¥Algo¥Sample¥Sample_SMART	SMART イベント通知のサンプルコードです。

4-1-2 DeviceIoControl について

産業用組み込み PC 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

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

4-2 タイマ割り込み機能

4-2-1 タイマ割り込み機能について

産業用組込み PC シリーズには、ハードウェアによるタイマ割り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り込みを発生させることができます。

4-2-2 タイマドライバについて

タイマドライバはタイマ割り込み機能を、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。

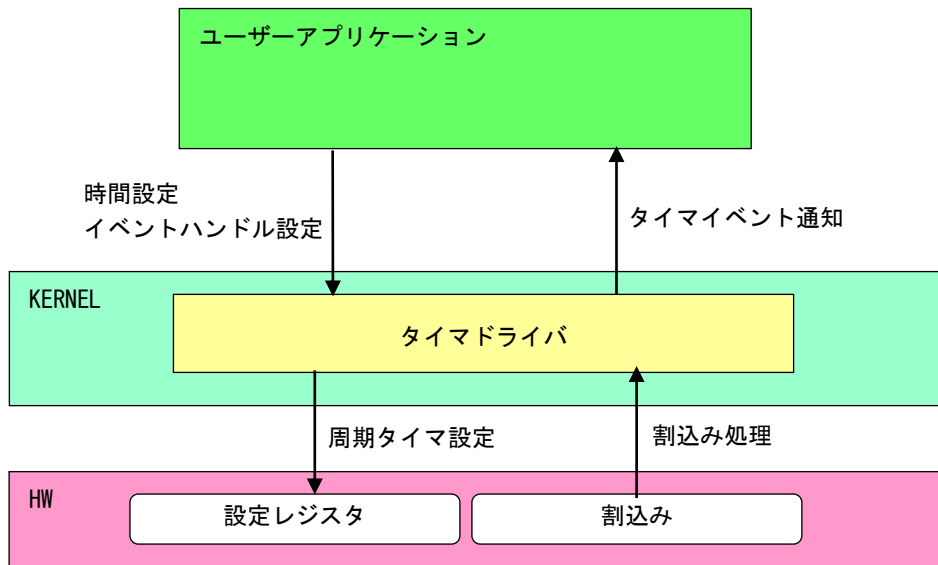


図 4-2-2-1. タイマドライバ

4-2-3 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実行します。

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	<p>[KEY] HKEY_LOCAL_MACHINE¥SYSTEM¥CurrentControlSet¥Services¥FpgaTimer</p> <p>[VALUE: DWORD] TimerResolution</p> <p>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)</p>
CreateFile	<p>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。</p> <pre>hTimer = CreateFile("¥¥¥¥. ¥¥FpgaTimer", GENERIC_READ GENERIC_WRITE, FILE_SHARE_READ FILE_SHARE_WRITE, NULL, OPEN_EXISTING, 0, NULL);</pre>
CloseHandle	<p>デバイスハンドルをクローズします。</p> <pre>CloseHandle(hTimer);</pre>
ReadFile	使用しません。
WriteFile	使用しません。
DeviceIoControl	<ul style="list-style-type: none"> ● 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。

4-2-4 タイムドライバの動作

- 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タ値を加算します。
- ⑤ カウンタ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。カウンタ加算、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。

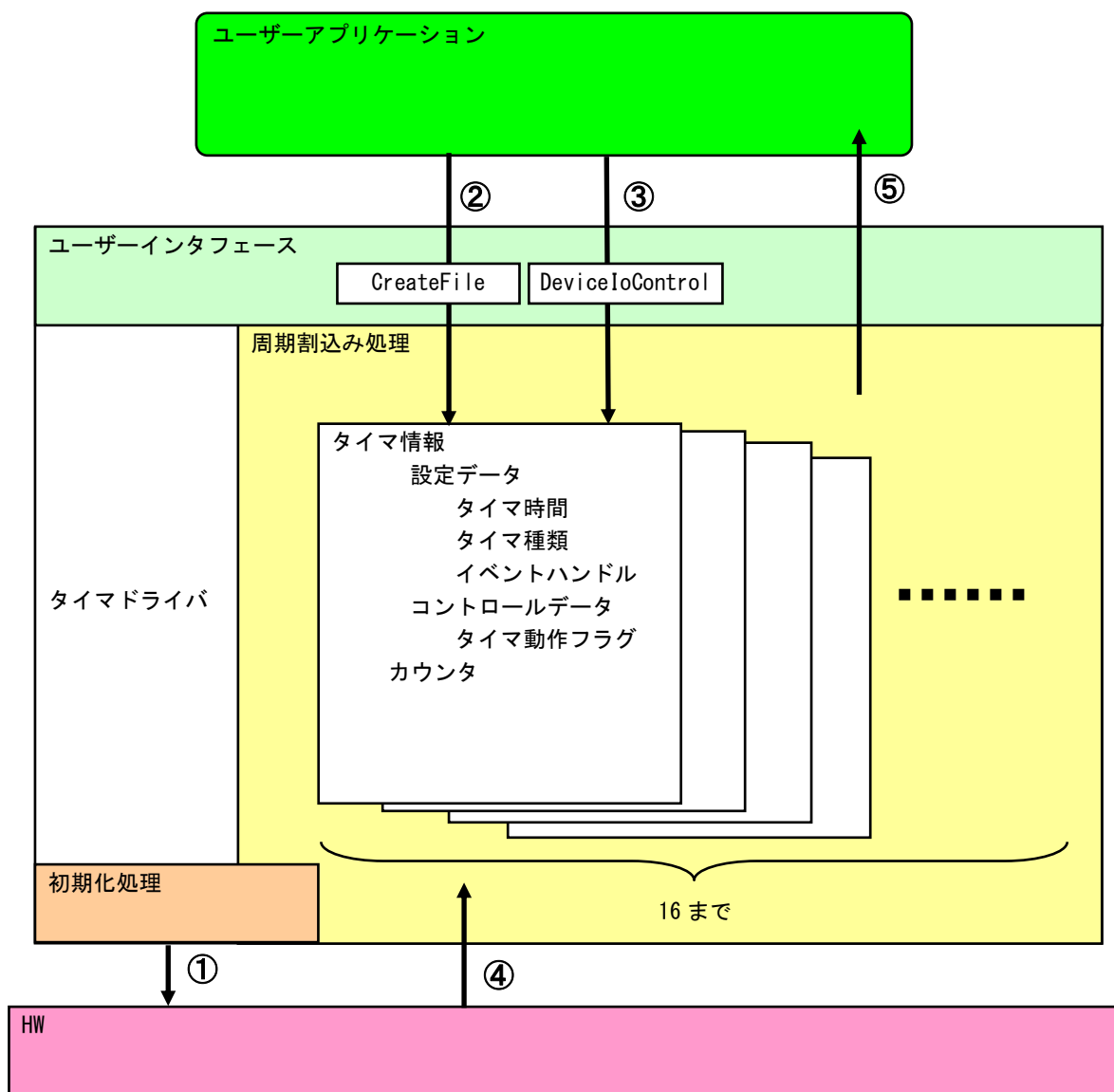


図 4-2-4-1. タイムドライバの動作

4-2-5 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。

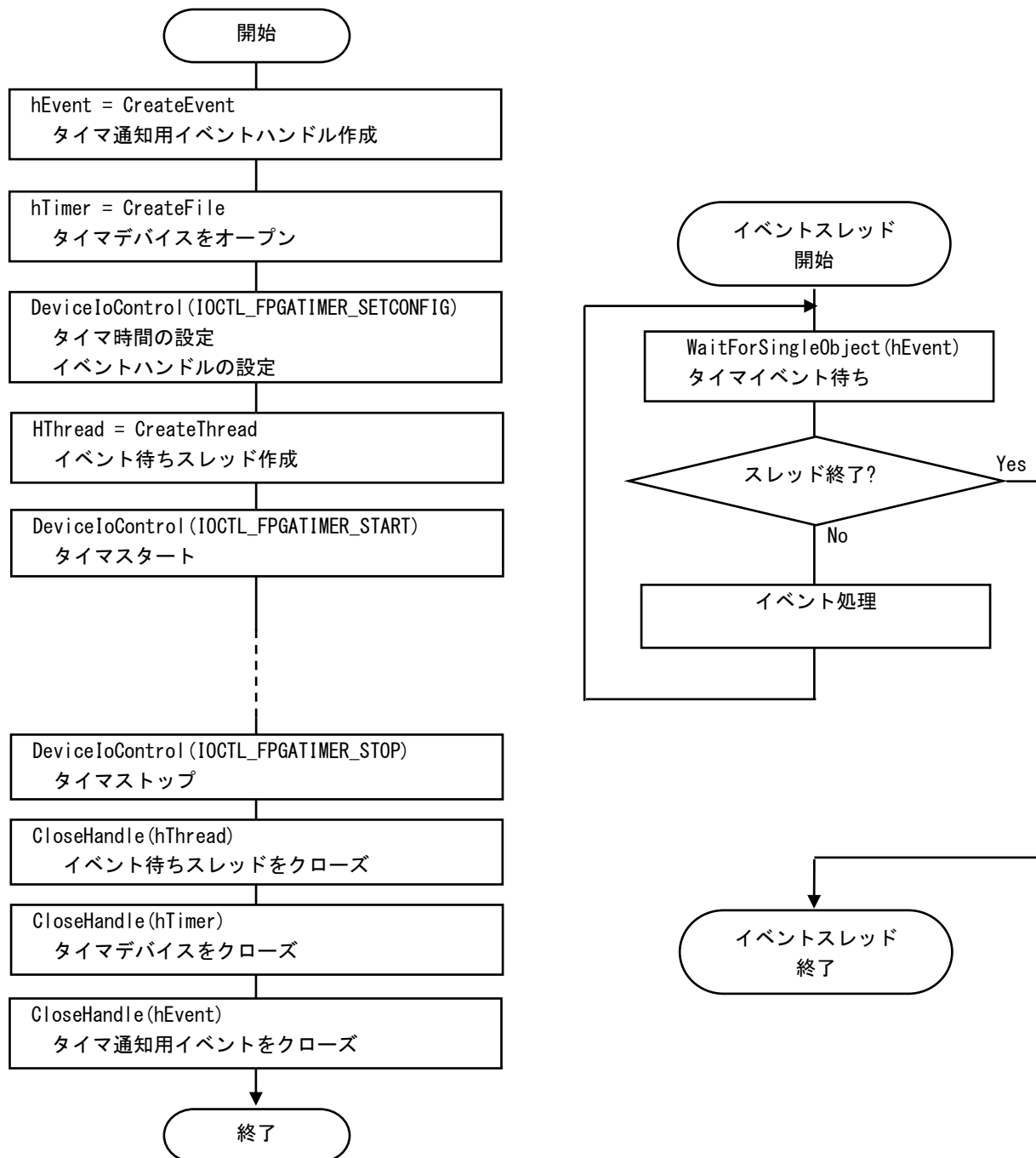


図 4-2-5-1. ドライバ使用手順

4-2-6 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 : 実際の実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 : 実際の実出力バイト数を受け取る変数へのポインタ。
lpOver lapped : 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 : 実際の実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 : 実際の実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。

4-2-7 サンプルコード

「¥SDK¥Algo¥Sample¥Sample_Interrupt¥FpgaTimer」にタイマ割り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 4-2-7-1 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 4-2-7-1. タイマ割り込み機能

```

/**
    タイマ割り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;

//-----
/*
 * 割り込みハンドラ
 */
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;  
}  
  
//-----
```

4-3 汎用入出力

4-3-1 汎用入出力について

産業用組込み PC 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 4-3-1-1 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。

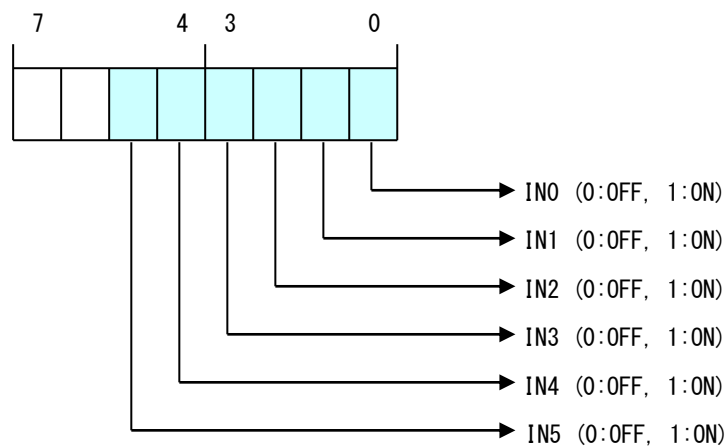


図 4-3-1-1. 入力データ

● 出力ポート

出力ポートのデータ形式を図 4-3-1-2 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。

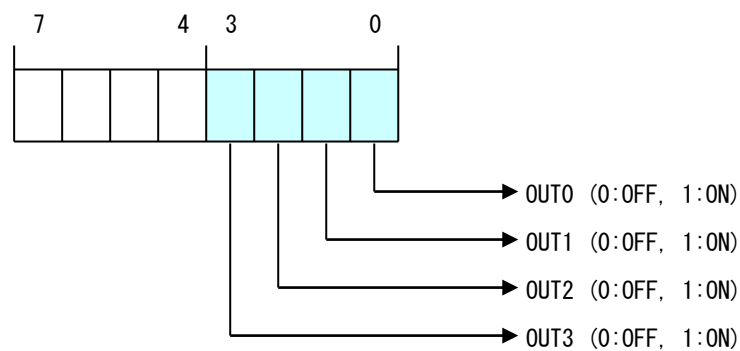


図 4-3-1-2. 出力データ

4-3-2 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。

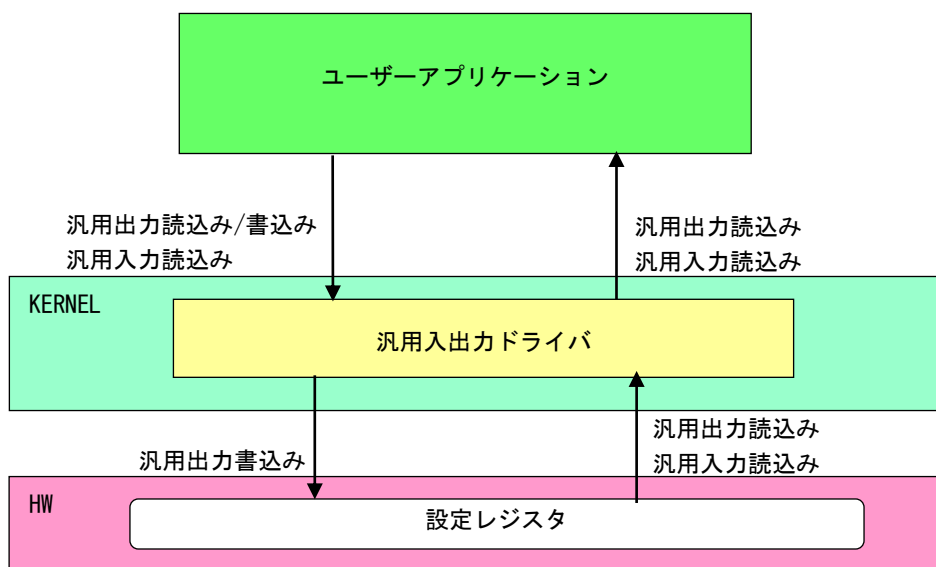


図 4-3-2-1. 汎用入出力ドライバ

4-3-3 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	<p>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。</p> <pre> hGenIo = CreateFile("¥¥¥¥. ¥¥GenIoDrv", GENERIC_READ GENERIC_WRITE, FILE_SHARE_READ FILE_SHARE_WRITE, NULL, OPEN_EXISTING, 0, NULL); </pre>
CloseHandle	<p>デバイスハンドルをクローズします。</p> <pre>CloseHandle(hGenIo);</pre>
ReadFile	使用しません。
WriteFile	使用しません。
DeviceIoControl	<ul style="list-style-type: none"> ● IOCTL_GENIODRV_RW 汎用入出力のリードライトを行います。

4-3-4 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 : 実際の実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 : 読み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。

4-3-5 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 4-3-5-1 にサンプルコードを示します。

リスト 4-3-5-1. 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;

    /* 汎用出力ポートのオープン */
    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");
}

/* 汎用入出力デバイスのカース */
CloseHandle(hGenIo);

return 0;
}
```

4-4 RAS 機能

4-4-1 RAS 機能について

産業用組込み PC 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。

4-4-2 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。

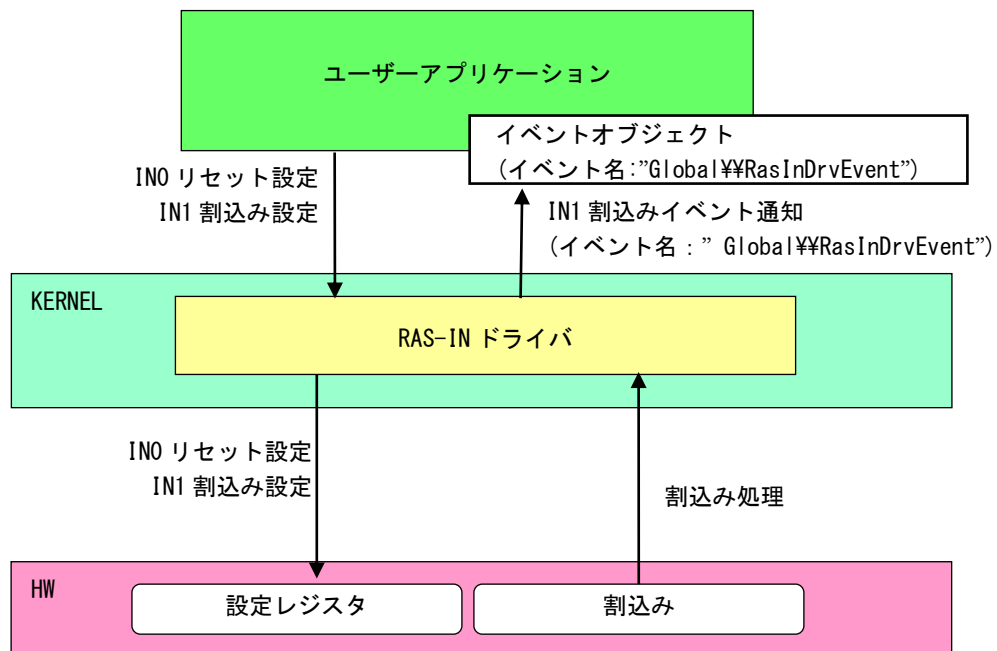


図 4-4-2-1. RAS-IN ドライバ

4-4-3 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	<p>INOリセット、IN1割り込み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。</p>
CreateFile	<p>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。</p> <pre> hRasIn = CreateFile("¥¥¥¥. ¥¥RasInDrv", GENERIC_READ GENERIC_WRITE, FILE_SHARE_READ FILE_SHARE_WRITE, NULL, OPEN_EXISTING, 0, NULL); </pre>
CloseHandle	<p>デバイスハンドルをクローズします。</p> <pre>CloseHandle(hRasIn);</pre>
ReadFile	使用しません。
WriteFile	使用しません。
DeviceIoControl	<ul style="list-style-type: none"> ● IOCTL_RASINDRV_SINORST INOリセットを設定します。 ● IOCTL_RASINDRV_GINORST 現在のINO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。

4-4-4 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。

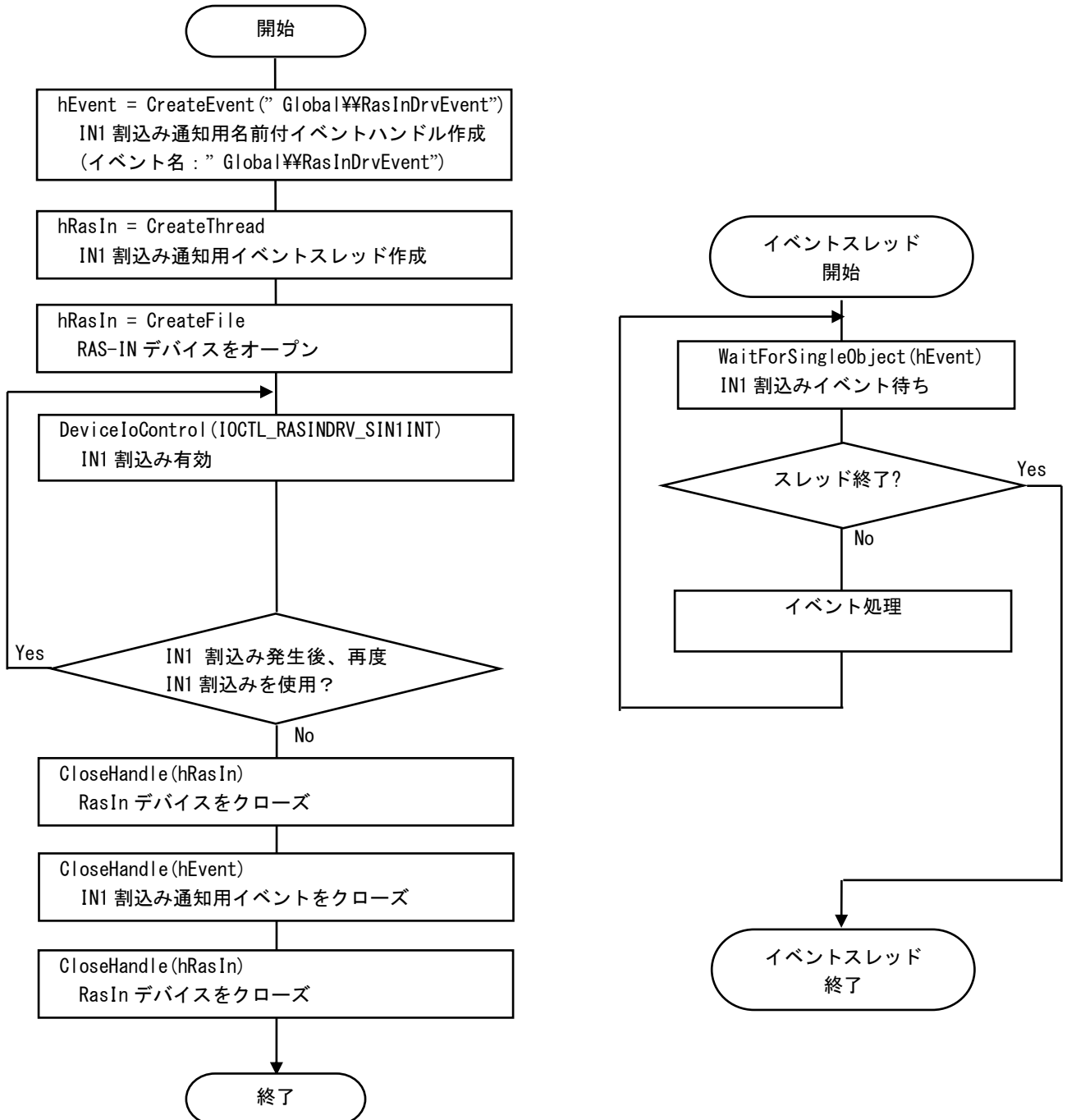


図 4-4-4-1. IN1 割込み仕様手順

4-4-5 複数アプリケーションで IN1 割り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 割り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 割り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 割り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 割り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 割り込み通知用名前付きイベントハンドルを作成後、IN1 割り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 割り込みが発生すれば、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 割り込み有効/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 割り込み有効/無効できませんので注意してください。

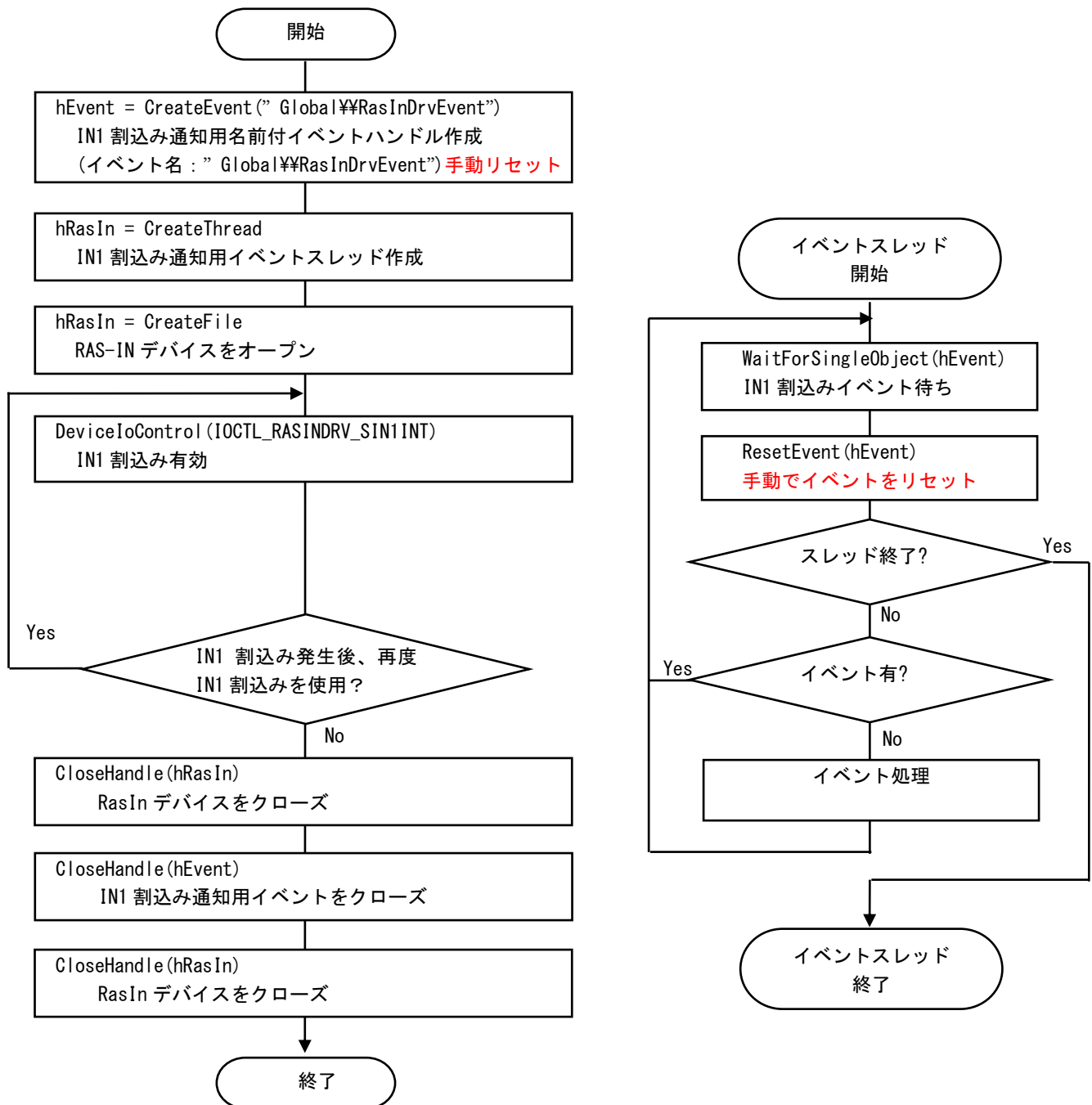


図 4-4-5-1. 複数アプリケーションで IN1 割り込みを使用する手順

4-4-6 DeviceIoControl リファレンス

IOCTL_RASINDRV_SINORST

機能

IN0 リセットを設定します。

パラメータ

lpInBuf : IN0 リセット情報を格納するポインタを指定します。
NInBufSize : IN0 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 : 実際の実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。

IN0 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: IN0 リセット有効, 0: IN0 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

IN0 リセットを設定します。

IN0 リセットを有効にする場合は、IN0 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 : 実際の実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 : 実際の実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 : 実際の実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。

4-4-7 サンプルコード

● INO リセット

「¥SDK¥Algo¥Sample¥Sample_Reset¥In0Reset」に INO 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 4-4-7-1 にサンプルコードを示します。

リスト 4-4-7-1. INO リセット

```
/**
 * 汎用入力 INO 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;
    }

    /*
     * INO 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 4-4-7-2 にサンプルコードを示します。

リスト 4-4-7-2. 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;
}

//-----
```

4-5 シリアルコントロール機能

4-5-1 シリアルコントロール機能について

産業用組み込み PC 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。

4-5-2 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。

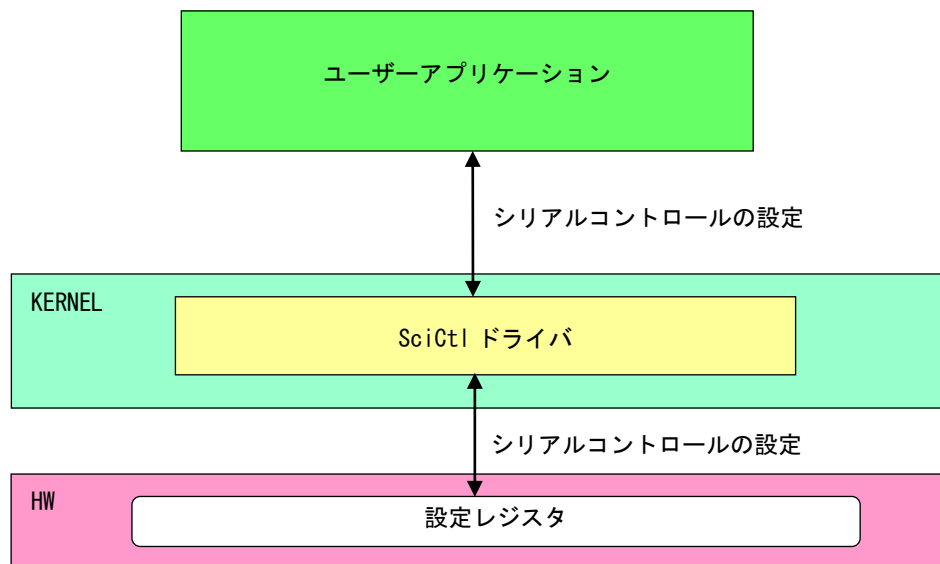


図 4-5-2-1. SciCtl ドライバ

4-5-3 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実行します。

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	<p>[KEY] HKEY_LOCAL_MACHINE¥SYSTEM¥CurrentControlSet¥Services¥SciCtl</p> <p>[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)</p> <p>[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)</p>
CreateFile	<p>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。</p> <pre> hSciCtl = CreateFile("¥¥¥¥. ¥SciCtl", GENERIC_READ GENERIC_WRITE, FILE_SHARE_READ FILE_SHARE_WRITE, NULL, OPEN_EXISTING, 0, NULL); </pre>
CloseHandle	<p>デバイスハンドルをクローズします。</p> <pre>CloseHandle(hSciCtl);</pre>
ReadFile	使用しません。
WriteFile	使用しません。
DeviceIoControl	<ul style="list-style-type: none"> ● 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。

4-5-4 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 : 実際の実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: SI01, 1: SI02]
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 : 実際の実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: SI01, 1: SI02]
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。

4-5-5 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 4-5-5-1 にサンプルコードを示します。

リスト 4-5-5-1. 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¥SciCtlIDD.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;
}
```

4-6 ダミー-SRAM

4-6-1 ダミー-SRAM について

産業用組み込み PC シリーズには、RAM バックアップ機能が搭載されています。ダミー-SRAM は RAM バックアップ機能で使用されるメモリエリアを確保します。ダミー-SRAM ドライバを操作することによって、RAM バックアップ用メモリエリアの読み書きを行うことができます。

4-6-2 ダミー-SRAM ドライバについて

ダミー-SRAM ドライバは RAM バックアップ用メモリエリアの読み書きを、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。

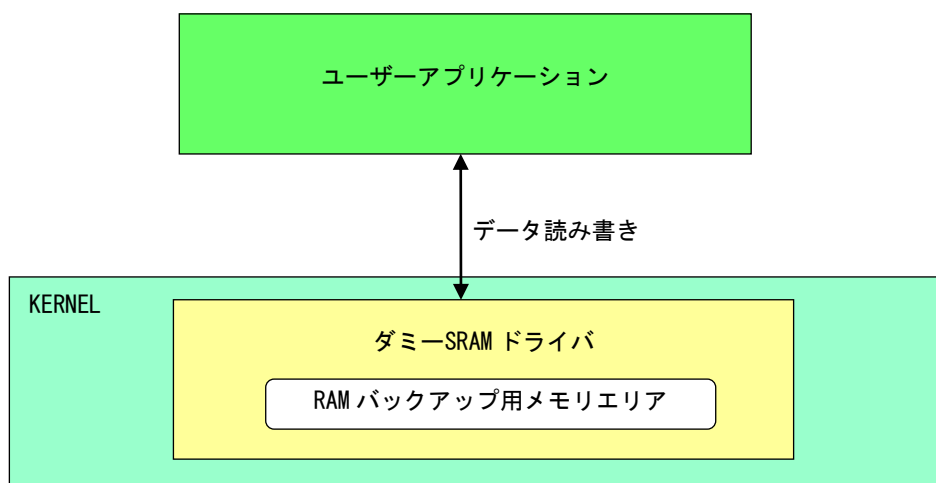


図 4-6-2-1. ダミー-SRAM ドライバ

4-6-3 ダミー-SRAM デバイス

ダミー-SRAM ドライバはダミー-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 RAM バックアップ用メモリエリアの読み書きを行います。

ダミー-SRAM デバイスのアクセス仕様は、旧製品の SRAM デバイスと同じものとなっています。

ダミー-SRAM デバイス	
デバイスファイル	¥¥.¥SramDrv
説明	RAMバックアップ用メモリエリアの読み書きを行うことができます。
CreateFile	<p>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。</p> <pre> hSram = CreateFile("¥¥¥¥. ¥¥SramDrv", GENERIC_READ GENERIC_WRITE, FILE_SHARE_READ FILE_SHARE_WRITE, NULL, OPEN_EXISTING, 0, NULL); </pre>
CloseHandle	<p>デバイスハンドルをクローズします。</p> <pre>CloseHandle(hSram);</pre>
ReadFile	使用しません。
WriteFile	使用しません。
DeviceIoControl	<ul style="list-style-type: none"> ● IOCTL_SRAMDRV_RW RAMバックアップ用メモリエリアの読み書きを行います。

4-6-4 DeviceIoControl リファレンス

IOCTL_SRAMDRV_RW

機能

RAM バックアップ用メモリエリア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 : 実際の実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 Length;
    ULONG64 BufferPtr;
} 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 ~ 0x9600000]
Length : サイズ [0x00000 ~ 0x9600000]
BufferPtr : データのポインタアドレス

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

RAM バックアップ用メモリエリアの読み書きを行います。

RAM バックアップ用メモリエリアの容量は 150M バイトです。

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メモリエリア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、指定した値がメモリエリアに書込まれます。

4-6-5 サンプルコード

「¥SDK¥Algo¥Sample¥Sample_RamBackup¥RamReadWriteCheck」に RAM バックアップ用メモリエリアのデータ Read/Write を行うサンプルコードを用意しています。リスト 4-6-5-1 にサンプルコードを示します。

リスト 4-6-5-1. バックアップ SRAM

```

/**
 * ダミーSRAM RAM バックアップ用メモリエリア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 RAMAREA_SIZE 0x9600000 /* メモリエリアサイズ */

unsigned char ramarea[RAMAREA_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 < RAMAREA_SIZE; i++, d++){
    ramarea[i] = d;
}

/*
 * データの書き込み
 *
 * 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      0x00000000 ~ 0x96000000
 *
 * rw_par.BufferPtr
 *     データポインタ
 *
 * rw_par.Length
 *     サイズ        0x00000000 ~ 0x96000000
 *
 */
rw_par.RW = RW_WRITE;
rw_par.Type = TYPE_BYTE;
rw_par.Addr = 0;
rw_par.BufferPtr = (ULONG64)ramarea;
rw_par.Length = RAMAREA_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(&ramarea[0], 0x00, RAMAREA_SIZE);

rw_par.RW = RW_READ;
rw_par.Type = TYPE_BYTE;
rw_par.Addr = 0;
rw_par.BufferPtr = (ULONG64)ramarea;
rw_par.Length = RAMAREA_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 < RAMAREA_SIZE; i++, d++){
    if(ramarea[i] != d) {
        fprintf(stderr, "ramarea data check: failed\n");
        CloseHandle(hSram);
        return -1;
    }
}
fprintf(stdout, "ramarea compare: success\n");

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

4-7 初期化スイッチ、汎用 LED

4-7-1 初期化スイッチ、汎用 LED について

産業用組込み PC シリーズには、初期化スイッチが 1 点、汎用 LED が 3 点用意されています。

- 初期化スイッチ

表 4-7-1-1 に初期化スイッチの機能を示します。

表 4-7-1-1. 初期化スイッチ機能

初期化スイッチ	機能
初期化スイッチ	初期化スイッチは、電源 ON 直後に 3 秒間長押しされたら、ON 状態でラッチします。初期化スイッチがラッチ状態の場合、LD1 は点滅状態になります。汎用 LED を操作し LD1 の点滅状態が解除されると、初期化スイッチは OFF 状態になります。

初期化スイッチポートのデータ形式を図 4-7-1-1 に示します。初期化スイッチ/汎用 LED 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。

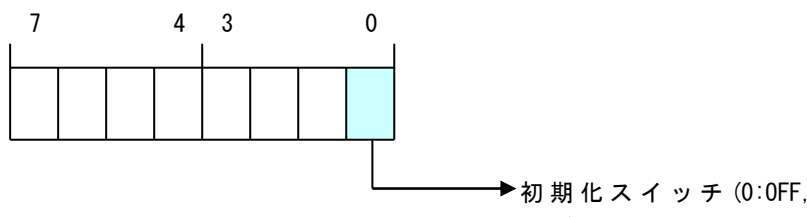


図 4-7-1-1. 初期化スイッチデータ

- 汎用 LED

汎用 LED は LED によって機能が異なります。表 4-7-1-2 に汎用 LED の機能を示します。

表 4-7-1-2. 汎用 LED 機能

汎用 LED	機能
LD1	初期化スイッチが ON ラッチしている間、LD1 は点滅します。LD1 が点滅状態の場合、初期化スイッチは ON 状態になります。アプリケーションで点灯/消灯を操作することができます。LD1 が点滅状態のとき LD1 を操作すると、点滅状態は解除され LD1 は指定した状態へと変化します。LD1 の点滅状態が解除されると初期化スイッチは OFF 状態になります。
LD2	アプリケーションで点灯/消灯を操作することができます。LD1 が点滅状態のとき LD2 を操作すると、LD1 の点滅状態は解除されます。
LD3	アプリケーションで点灯/消灯を操作することができます。LD1 が点滅状態のとき LD3 を操作すると、LD1 の点滅状態は解除されます。

汎用 LED ポートのデータ形式を図 4-7-1-2 に示します。初期化スイッチ/汎用 LED 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。

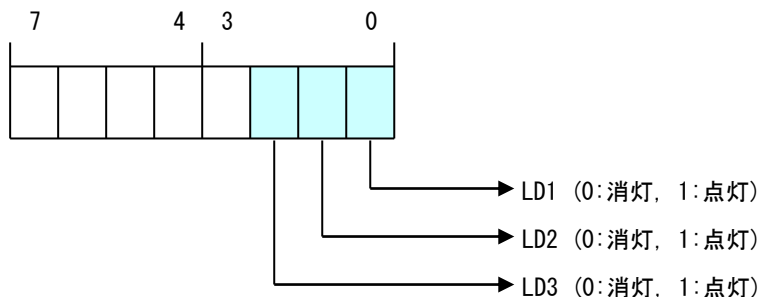


図 4-7-1-2. 汎用 LED データ

4-7-2 初期化スイッチ/汎用 LED ドライバについて

初期化スイッチ/汎用 LED ドライバは初期化スイッチ、汎用 LED 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、初期化スイッチ/汎用 LED ドライバを直接制御することで初期化スイッチ/汎用 LED を制御することが可能です。

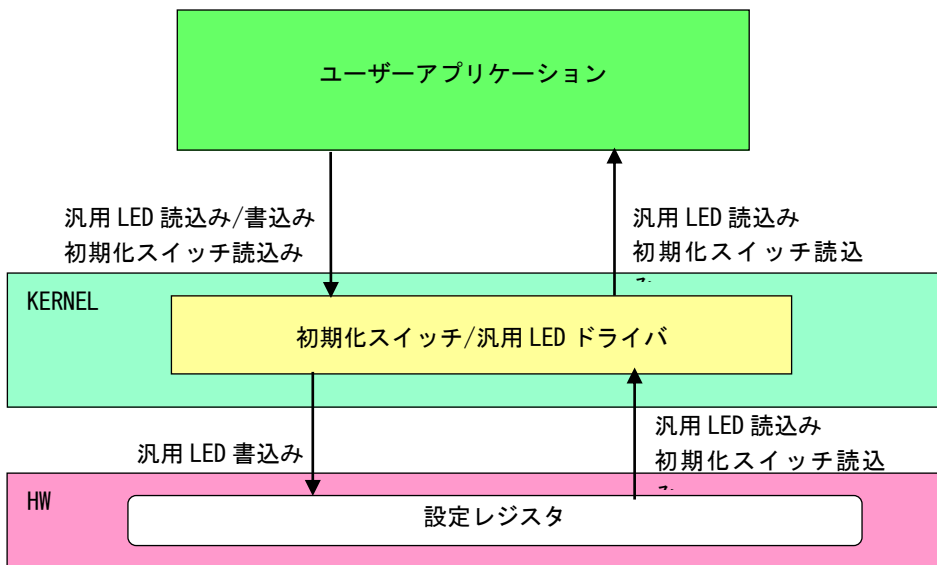


図 4-7-2-1. 初期化スイッチ/汎用 LED ドライバ

4-7-3 初期化スイッチ/汎用 LED デバイス

初期化スイッチ/汎用 LED ドライバは初期化スイッチ/汎用 LED 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初期化スイッチ、汎用 LED を操作します。

初期化SW/汎用LEDデバイス	
デバイスファイル	¥¥.¥SwLedDrv
説明	初期化SW、汎用LEDの制御を行うことができます。
CreateFile	<p>デバイスファイル(¥¥. ¥SwLedDrv) をオープンし、デバイスハンドルを取得します。</p> <pre> hSwLed = CreateFile("¥¥¥¥. ¥¥SwLedDrv", GENERIC_READ GENERIC_WRITE, FILE_SHARE_READ FILE_SHARE_WRITE, NULL, OPEN_EXISTING, 0, NULL); </pre>
CloseHandle	<p>デバイスハンドルをクローズします。</p> <pre>CloseHandle(hSwLed);</pre>
ReadFile	使用しません。
WriteFile	使用しません。
DeviceIoControl	<ul style="list-style-type: none"> ● IOCTL_SWLEDDRV_RW 初期化SW、汎用LEDのリードライトを行います。

4-7-4 DeviceIoControl リファレンス

IOCTL_SWLEDDRV_RW

機能

初期化スイッチ、汎用 LED のリードライトを行います。

パラメータ

lpInBuf	: SWLEDDRV_RW_PAR を格納するためのポインタを指定します。
NInBufSize	: SWLEDDRV_RW_PAR のサイズを指定します。
lpOutBuf	: SWLEDDRV_RW_PAR を格納するためのポインタを指定します。
NOutBufSize	: SWLEDDRV_RW_PAR のサイズを指定します。
LpBytesReturned	: 実際の実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。

SWLEDDRV_RW_PAR

```
typedef struct {
    ULONG RW;
    ULONG Type;
    ULONG Bit;
    ULONG Data;
} SWLEDDRV_RW_PAR, *P_SWLEDDRV_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]
Type	: 入出力ポート [0: 初期化スイッチポート, 1: 汎用 LED ポート]
Bit	: ビット番号 ^(※1) [0Bit~2Bit]
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

初期化スイッチ、汎用 LED 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 SWLEDDRV_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)
 - Type: 初期化スイッチポート・汎用 LED ポートを指定します。
 - Bit: 無視されます。
 - Data: 読み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)
 - Type : 初期化スイッチポート・汎用 LED ポートを指定します。
 - Bit : 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 SWLEDDRV_RW_PAR 構造体を渡します。

- データ型でのライト (RW = 1)
 - Type : 汎用 LED ポートを指定します。
 - Bit : 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)
 - Type : 汎用 LED ポートを指定します。
 - Bit : 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。

4-7-5 サンプルコード

「¥SDK¥Algo¥Sample¥Sample_SwLed¥SwLed」に初期化スイッチ/汎用 LED ドライバを使用したサンプルコードを用意しています。

- 初期化スイッチのデータ型読み込み

「¥SDK¥Algo¥Sample¥Sample_SwLed¥SwLed¥SwReadData.cpp」は、初期化スイッチをデータ型で使用する読み込むプログラム例です。リスト 4-7-5-1 にサンプルコードを示します。

リスト 4-7-5-1. 初期化スイッチデータ型読み込み

```

/**
 フロントパネル SW 入力制御サンプルソース
 DATA アクセス
 **/

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

#include "..¥Common¥SwLedDrvDD.h"

#define DRIVER_FILENAME "¥¥¥¥.¥¥SwLedDrv"

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

    SWLEDDRV_RW_PAR rw_par;

    rw_par.RW = RW_READ;
    rw_par.Type = PORT_SW;
    rw_par.Bit = 0;

    ret = DeviceIoControl(hDevice,
                          IOCTL_SWLEDDRV_RW,
                          &rw_par,
                          sizeof(SWLEDDRV_RW_PAR),
                          &rw_par,
                          sizeof(SWLEDDRV_RW_PAR),
                          &retlen,
                          NULL);

    if(!ret) {
        return FALSE;
    }
    if(retlen != sizeof(SWLEDDRV_RW_PAR)) {
        return FALSE;
    }
    *pBuffer = (USHORT)rw_par.Data;

```

```
    return TRUE;
}

int main(int argc, char **argv)
{
    HANDLE h_swled;
    BOOL ret;
    USHORT data;

    /* デバイスのオープン */
    h_swled = CreateFile(
        DRIVER_FILENAME,
        GENERIC_READ | GENERIC_WRITE,
        FILE_SHARE_READ | FILE_SHARE_WRITE,
        NULL,
        OPEN_EXISTING,
        0,
        NULL
    );

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

    /*
     SW 入力
    */
    ret = ReadIn(h_swled, &data);
    if(!ret) {
        printf("DeviceIoControl: IOCTL_SWLEDDRV_RW NG¥n");
        CloseHandle(h_swled);
        return -1;
    }

    printf("SW: Data=0x%02X¥n", data);

    /* デバイスのクローズ */
    CloseHandle(h_swled);

    return 0;
}
```

● 初期化スイッチのビット指定読み

「¥SDK¥Algo¥Sample¥Sample_SwLed¥SwLed¥SwReadBit. cpp」は、初期化スイッチをビット指定で読むプログラム例です。リスト 4-7-5-2 にサンプルコードを示します。

リスト 4-7-5-2. 初期化スイッチビット指定読み

```
/**
 * フロントパネル SW 入力制御サンプルソース
 * BIT アクセス
 */

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

#include "..¥Common¥SwLedDrvDD. h"

#define DRIVER_FILENAME "¥¥¥¥. ¥¥SwLedDrv"

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

    SWLEDDRV_RW_PAR rw_par;

    rw_par.RW = RW_READBIT;
    rw_par.Type = PORT_SW;
    rw_par.Bit = Bit;

    ret = DeviceIoControl (hDevice,
                           IOCTL_SWLEDDRV_RW,
                           &rw_par,
                           sizeof (SWLEDDRV_RW_PAR),
                           &rw_par,
                           sizeof (SWLEDDRV_RW_PAR),
                           &retlen,
                           NULL);

    if (!ret) {
        return FALSE;
    }
    if (retlen != sizeof (SWLEDDRV_RW_PAR)) {
        return FALSE;
    }
    *pBuffer = (USHORT)rw_par.Data;

    return TRUE;
}
```

```
int main(int argc, char **argv)
{
    HANDLE h_swled;
    BOOL ret;
    int bitno;
    USHORT bitsts;

    /* デバイスのオープン */
    h_swled = CreateFile(
        DRIVER_FILENAME,
        GENERIC_READ | GENERIC_WRITE,
        FILE_SHARE_READ | FILE_SHARE_WRITE,
        NULL,
        OPEN_EXISTING,
        0,
        NULL
    );
    if(h_swled == INVALID_HANDLE_VALUE) {
        printf("CreateFile: NG¥n");
        return -1;
    }

    /*
     * SW 入力
     */
    for(bitno = 0; bitno < 2; bitno++){
        ret = ReadInBit(h_swled, bitno, &bitsts);
        if(!ret) {
            printf("DeviceIoControl: IOCTL_SWLEDDRV_RW NG¥n");
            CloseHandle(h_swled);
            return -1;
        }

        printf("SW%d: Status=%d¥n", bitno, bitsts);
    }

    /* デバイスのクローズ */
    CloseHandle(h_swled);

    return 0;
}
```

● 汎用 LED のデータ型書込み

「¥SDK¥Algo¥Sample¥Sample_SwLed¥SwLed¥LedWriteData.cpp」は、汎用 LED にデータ型で書込むプログラム例です。リスト 4-7-5-3 にサンプルコードを示します。

リスト 4-7-5-3. 汎用 LED データ型書込み

```
/**
 * フロントパネル LED 出力制御サンプルソース
 * DATA アクセス
 */

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

#include "..¥Common¥SwLedDrvDD.h"

#define DRIVER_FILENAME "¥¥¥¥. ¥¥SwLedDrv"

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

    SWLEDDRV_RW_PAR rw_par;

    rw_par.RW = RW_WRITE;
    rw_par.Type = PORT_LED;
    rw_par.Bit = 0;
    rw_par.Data = (ULONG)Data;

    ret = DeviceIoControl(hDevice,
                          IOCTL_SWLEDDRV_RW,
                          &rw_par,
                          sizeof(SWLEDDRV_RW_PAR),
                          &rw_par,
                          sizeof(SWLEDDRV_RW_PAR),
                          &retlen,
                          NULL);

    if(!ret) {
        return FALSE;
    }
    if(retlen != sizeof(SWLEDDRV_RW_PAR)) {
        return FALSE;
    }

    return TRUE;
}
```

```
int main(int argc, char **argv)
{
    HANDLE h_swled;
    BOOL ret;
    int data;

    if(argc != 2) {
        printf("Usage: LedWriteData <Data(0x??)>%n");
        return -1;
    }
    sscanf(*(argv + 1), "0x%x", &data);

    /* デバイスのオープン */
    h_swled = CreateFile(
        DRIVER_FILENAME,
        GENERIC_READ | GENERIC_WRITE,
        FILE_SHARE_READ | FILE_SHARE_WRITE,
        NULL,
        OPEN_EXISTING,
        0,
        NULL
    );
    if(h_swled == INVALID_HANDLE_VALUE) {
        printf("CreateFile: NG%n");
        return -1;
    }

    /*
     * LED 出力
     */
    ret = WriteOut(h_swled, (USHORT)data);
    if(!ret) {
        printf("DeviceIoControl: IOCTL_SWLEDDRV_RW NG%n");
        CloseHandle(h_swled);
        return -1;
    }

    /* デバイスのクローズ */
    CloseHandle(h_swled);

    return 0;
}
```

● 汎用 LED のビット指定書込み

「¥SDK¥Algo¥Sample¥Sample_SwLed¥SwLed¥LedWriteBit.cpp」は、汎用 LED にビット指定で書込むプログラム例です。リスト 4-7-5-4 にサンプルコードを示します。

リスト 4-7-5-4. 汎用 LED ビット指定書込み

```
/**
 * フロントパネル LED 出力制御サンプルソース
 * BIT アクセス
 */

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

#include "..¥Common¥SwLedDrvDD.h"

#define DRIVER_FILENAME "¥¥¥¥. ¥¥SwLedDrv"

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

    SWLEDDRV_RW_PAR rw_par;

    rw_par.RW = RW_WRITEBIT;
    rw_par.Type = PORT_LED;
    rw_par.Bit = Bit;
    rw_par.Data = (ULONG)Data;

    ret = DeviceIoControl(hDevice,
                          IOCTL_SWLEDDRV_RW,
                          &rw_par,
                          sizeof(SWLEDDRV_RW_PAR),
                          &rw_par,
                          sizeof(SWLEDDRV_RW_PAR),
                          &retlen,
                          NULL);

    if(!ret) {
        return FALSE;
    }
    if(retlen != sizeof(SWLEDDRV_RW_PAR)) {
        return FALSE;
    }

    return TRUE;
}
```

```
int main(int argc, char **argv)
{
    HANDLE h_swled;
    BOOL ret;
    int bitno;
    int bitsts;

    if(argc != 3){
        printf("Usage: LedWriteBit <BitNo> <BitStatus>¥n");
        return -1;
    }
    sscanf(*(argv + 1), "%d", &bitno);
    sscanf(*(argv + 2), "%d", &bitsts);

    /* デバイスのオープン */
    h_swled = CreateFile(
        DRIVER_FILENAME,
        GENERIC_READ | GENERIC_WRITE,
        FILE_SHARE_READ | FILE_SHARE_WRITE,
        NULL,
        OPEN_EXISTING,
        0,
        NULL
    );
    if(h_swled == INVALID_HANDLE_VALUE) {
        printf("CreateFile: NG¥n");
        return -1;
    }

    /*
     * LED 出力
     */
    ret = WriteOutBit(h_swled, bitno, bitsts);
    if(!ret) {
        printf("DeviceIoControl: IOCTL_SWLEDDRV_RW NG¥n");
        CloseHandle(h_swled);
        return -1;
    }

    /* デバイスのクローズ */
    CloseHandle(h_swled);

    return 0;
}
```


4-8 ハードウェア・ウォッチドッグタイマ機能

4-8-1 ハードウェア・ウォッチドッグタイマ機能について

産業用組込み PC 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。

※ 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。「2-9-5 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。

4-8-2 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、タイム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。

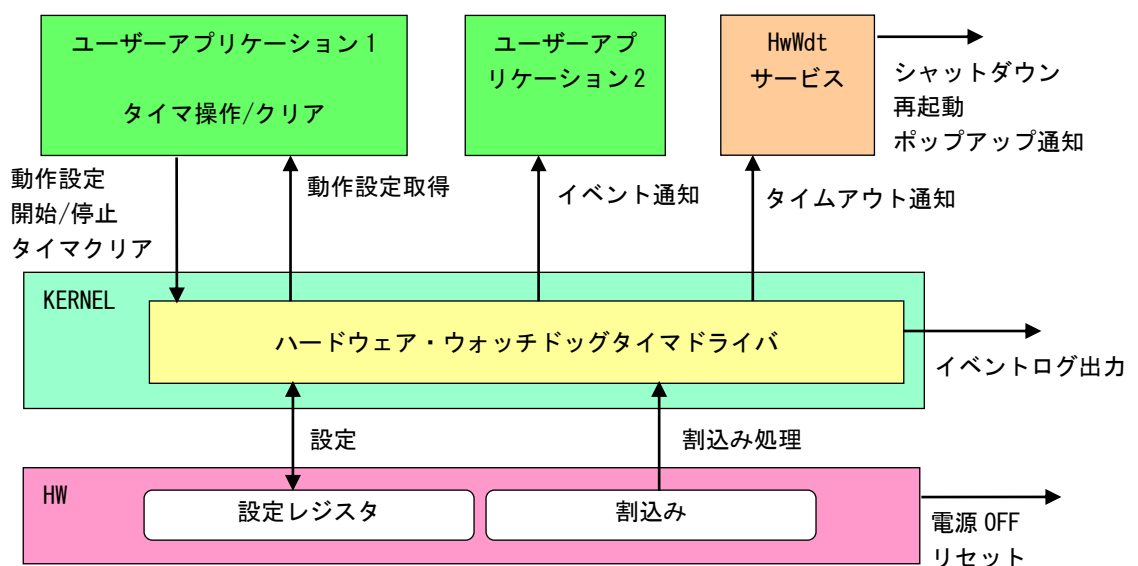


図 4-8-2-1. ハードウェア・ウォッチドッグタイマドライバ

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。**

4-8-3 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、タイムクリアを行うことができます。
レジストリ設定	<p>[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。</p> <ul style="list-style-type: none"> 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 <p>[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</p> <p>[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。</p> <ul style="list-style-type: none"> 0: 無効 1: 有効
CreateFile	<p>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。</p> <pre> hWdog = CreateFile("¥¥¥¥. ¥¥HwWdtDrv", GENERIC_READ GENERIC_WRITE, FILE_SHARE_READ FILE_SHARE_WRITE, NULL, OPEN_EXISTING, 0, NULL); </pre>

ReadFile	
使用しません。	
WriteFile	
使用しません。	
DeviceIoControl	
<ul style="list-style-type: none">● 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。	

4-8-4 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 : 実際の実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せん。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 : 実際の実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。
lpOver lapped : 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 : 実際の実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。
lpOver lapped : 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 : 実際の実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。
lpOver lapped : 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 : 実際の実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。
lpOver lapped : 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。

4-8-5 サンプルコード

●タイマ操作

「¥SDK¥A|go¥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 4-8-5-1 にサンプルコードを示します。

リスト 4-8-5-1. 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_swtdt;
    ULONG retlen;
    BOOL ret;

    int wdt_action;
    int wdt_time;
    int wdt_eventlog;
    HWWD_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;
    }

    /*
     * タイムクリア処理('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 = HWWD_TCONTROL_STOP;
ret = DeviceIoControl(
    h_swwdt,
    IOCTL_HWWD_TCONTROL,
    &startval,
    sizeof(ULONG),
    NULL,
    0,
    &retlen,
    NULL
);
if(!ret) {
    printf("DeviceIoControl: IOCTL_HWWD_TCONTROL 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 4-8-5-2 にサンプルコードを示します。

リスト 4-8-5-2. 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;
}
```


4-9 ソフトウェア・ウォッチドッグタイマ機能

4-9-1 ソフトウェア・ウォッチドッグタイマ機能について

産業用組込み PC 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。

4-9-2 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、タイム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。

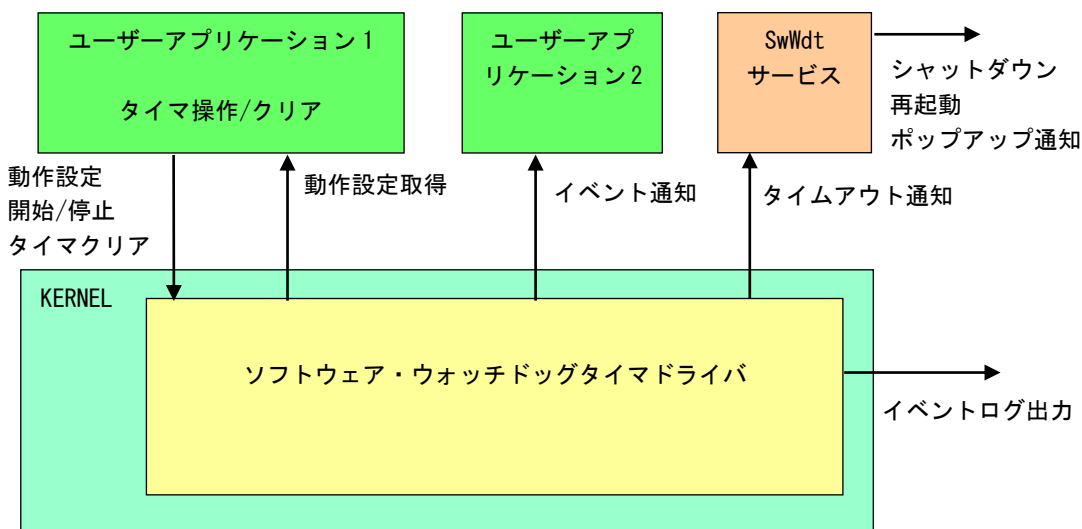


図 4-9-2-1. ソフトウェア・ウォッチドッグタイマドライバ

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。

4-9-3 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	<p>[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。</p> <ul style="list-style-type: none"> 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 <p>[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</p> <p>[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。</p> <ul style="list-style-type: none"> 0: 無効 1: 有効
CreateFile	<p>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。</p> <pre> hWdog = CreateFile("¥¥¥¥. ¥¥SwWdtDrv", GENERIC_READ GENERIC_WRITE, FILE_SHARE_READ FILE_SHARE_WRITE, NULL, OPEN_EXISTING, 0, NULL); </pre>

ReadFile	
使用しません。	
WriteFile	
使用しません。	
DeviceIoControl	
<ul style="list-style-type: none">● 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。	

4-9-4 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 : 実際の実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 : 実際の実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。
lpOver lapped : 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 : 実際の実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。
lpOver lapped : 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 : 実際の実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。
lpOver lapped : 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 : 実際の実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。
lpOver lapped : 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。

4-9-5 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 4-9-5-1 にサンプルコードを示します。

リスト 4-9-5-1. 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;
    }

    /*
     * タイムクリア処理('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 4-9-5-2 にサンプルコードを示します。

リスト 4-9-5-2. 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;
}
```

4-10 RAS 監視機能

4-10-1 RAS 監視機能について

産業用組込み PC 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。

4-10-2 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。

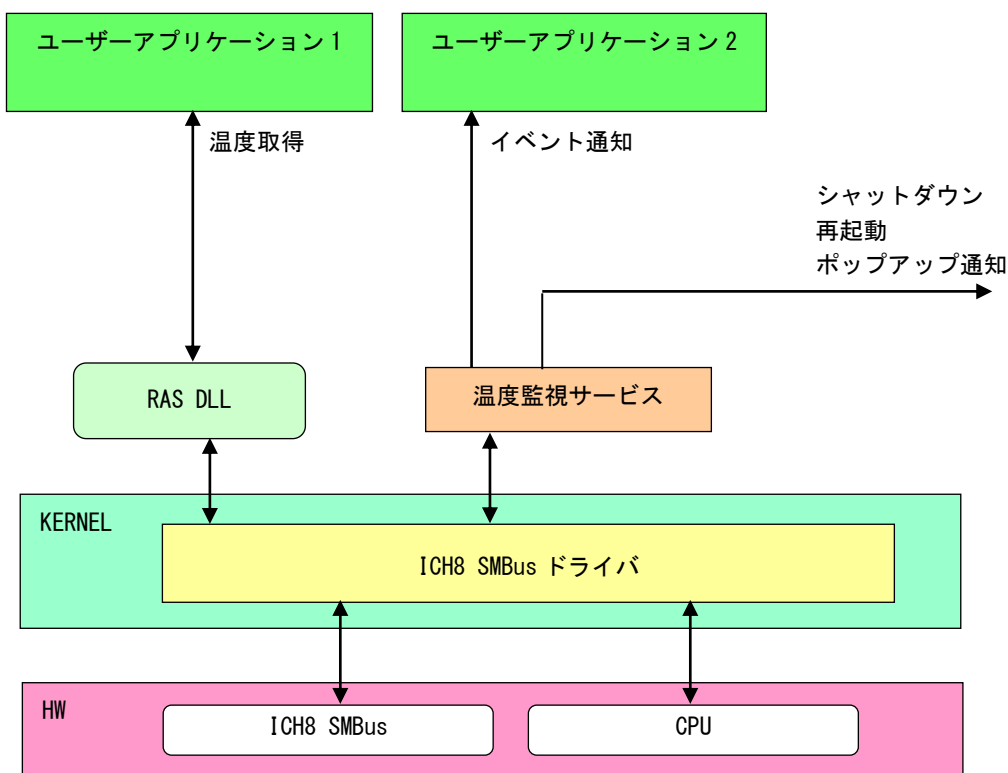


図 4-10-2-1. RAS DLL

4-10-3 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。

4-10-4 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 4-10-4-1 にサンプルコードを示します。

リスト 4-10-4-1. 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°C¥n", exttemp);

    /*
     * CPU 温度表示
     */
    G5_GetCPUtemperature (0, &cputemp);
    printf ("CPU Core#0 Temperature: %d°C¥n", cputemp);
    G5_GetCPUtemperature (1, &cputemp);
    printf ("CPU Core#1 Temperature: %d°C¥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 4-10-4-2 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 4-10-4-2. 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;
}
```

4-1-1 外部 RTC 機能

4-1-1-1 外部 RTC 機能について

産業用組み込み PC 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。(詳細は、「2-2 外部 RTC」を参照してください。)

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、「2-9-7 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。

4-1-1-2 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。

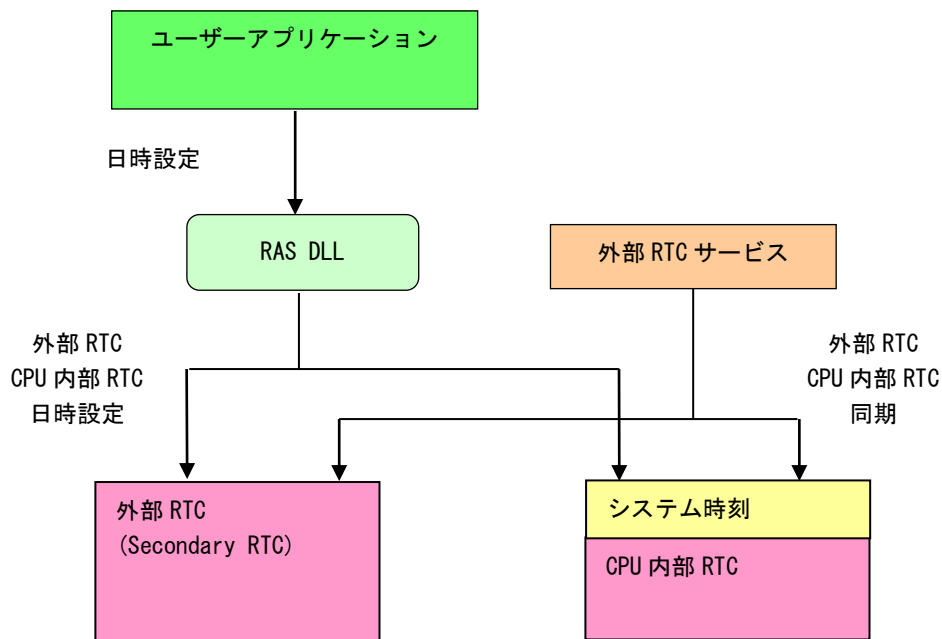


図 4-11-2-1. RAS DLL 日時設定

4-11-3 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 が同時に設定されます。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 が同時に設定されます。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。

4-11-4 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
㊦	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。

4-11-5 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 4-11-5-1 にサンプルコードを示します。

リスト 4-11-5-1. 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 system;

    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
    system.wYear = tm_year;
    system.wMonth = tm_mon;
    system.wDay = tm_mday;
    system.wHour = tm_hour;
    system.wMinute = tm_min;
    system.wSecond = tm_sec;
    system.wMilliseconds = 0;
    if(!G5_SetLocalTime(&system)) {
```

```
    printf("G5_SetLocalTime: NG¥n");
    UnloadG5RasDll();
    return -1;
}

// GetLocalTime
memset(&system, 0x00, sizeof(SYSTEMTIME));
GetLocalTime(&system);
printf("GetLocalTime: %04d:%02d:%02d %02d:%02d:%02d.%03d¥n",
       system.wYear, system.wMonth, system.wDay, system.wHour, system.wMinute, system.wSecond,
       system.wMilliseconds);

    UnloadG5RasDll();

    return 0;
}
```

● システム日時を使用した日時設定

「¥SDK¥Algo¥Sample¥Sample_RASDII¥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 4-11-5-2 にサンプルコードを示します。

リスト 4-11-5-2. 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¥AlG5Ras.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(&system, 0x00, sizeof(SYSTEMTIME));
GetSystemTime(&system);
printf("GetSystemTime: %04d:%02d:%02d %02d:%02d:%02d.%03d¥n",
       system.wYear, system.wMonth, system.wDay, system.wHour, system.wMinute, system.wSecond,
       system.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 4-11-5-3 にサンプルコードを示します。

リスト 4-11-5-3. 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;
}
```

4-12 ビープ音

4-12-1 ビープ音について

Windows 10 IoT Enterprise では、Windows API の Beep 関数を使用するとサウンドデバイスへの出力になり、ビープ音が出されません。産業用組込み PC シリーズには、ハードウェアのビープ音の ON/OFF を行う専用のドライバを用意しています。

4-12-2 ビープドライバについて

ビープドライバは、ビープ音制御レジスタを、ユーザーアプリケーションから操作できるようにします。ユーザーアプリケーションから、ビープ音の ON/OFF、ビープ音の周波数を変更することができます。

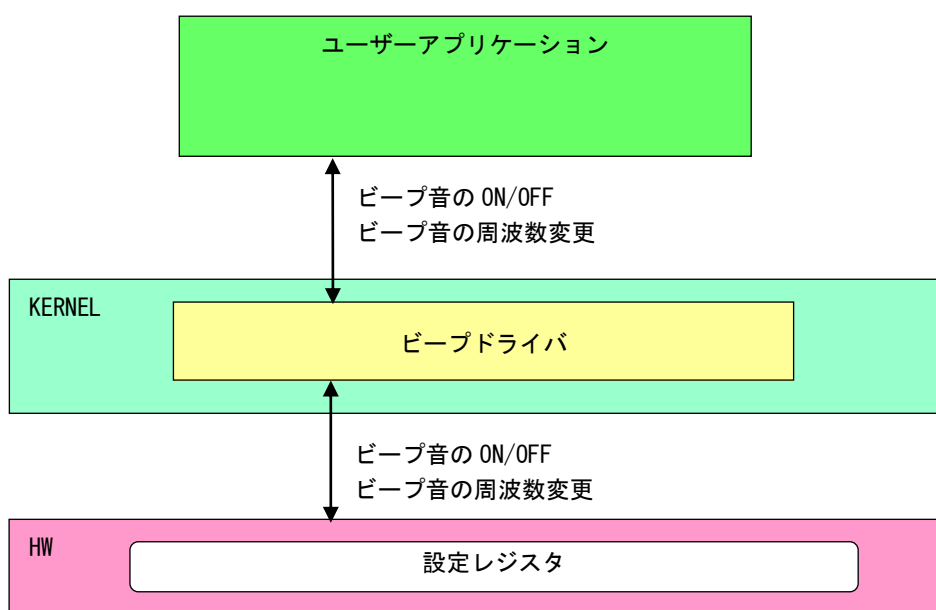


図 4-12-2-1. ビープドライバ

4-12-3 ビープデバイス

ビープドライバはビープ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ビープ音機能进行操作します。

ビープデバイス	
デバイスファイル	¥¥.¥BeepDrv
説明	ビープ音のON/OFF、ビープ音の周波数の設定を行うことができます。
CreateFile	<p>デバイスファイル(¥¥. ¥BeepDrv)をオープンし、デバイスハンドルを取得します。</p> <pre> hBeep = CreateFile("¥¥¥¥. ¥¥BeepDrv", GENERIC_READ GENERIC_WRITE, FILE_SHARE_READ FILE_SHARE_WRITE, NULL, OPEN_EXISTING, 0, NULL); </pre>
CloseHandle	<p>デバイスハンドルをクローズします。</p> <pre>CloseHandle(hBeep);</pre>
ReadFile	使用しません。
WriteFile	使用しません。
DeviceIoControl	<ul style="list-style-type: none"> ● IOCTL_BEEPDRV_GETBEEP ビープ音のON/OFFを取得します。 ● IOCTL_BEEPDRV_SETBEEP ビープ音のON/OFFを設定します。 ● IOCTL_BEEPDRV_GETHZ ビープ音周波数を取得します。 ● IOCTL_BEEPDRV_SETHZ ビープ音周波数を設定します。

4-12-4 DeviceIoControl リファレンス

IOCTL_BEEPDRV_SETBEEP

機能

ビープ音の ON/OFF の設定を行います。

パラメータ

lpInBuf : ビープ音情報を格納するポインタを指定します。
NInBufSize : ビープ音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 : 実際の実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。

ビープ音情報

データタイプ : ULONG
データサイズ : 4 バイト
内容 : 1: ON, 0: OFF

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

ビープ音の ON/OFF の設定を行います。

ビープ音を ON にする場合は、ビープ音情報を格納するポインタに 1、OFF にする場合は 0 を設定してから DeviceIoControl を実行してください。

IOCTL_BEEPDRV_GETBEEP

機能

ビーブ音の ON/OFF の取得を行います。

パラメータ

lpInBuf : NULL を指定します。
NInBufSize : 0 を指定します。
lpOutBuf : ビーブ音情報を格納するポインタを指定します。
NOutBufSize : ビーブ音情報を格納するポインタのサイズを指定します。
lpBytesReturned : 実際の実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。
lpOver lapped : NULL を指定します。

ビーブ音情報

データタイプ : ULONG
データサイズ : 4 バイト
内容 : 1: ON, 0: OFF

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

ビーブ音の ON/OFF の取得を行います。

IOCTL_BEEPDRV_SETHZ

機能

ビーブ音の周波数の設定を行います。

パラメータ

lpInBuf : ビーブ音周波数情報を格納するポインタを指定します。
NInBufSize : ビーブ音周波数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 : 実際の実出力バイト数を受け取る変数へのポインタ。
lpOver lapped : NULL を指定します。

ビーブ音周波数情報

データタイプ : ULONG
データサイズ : 4 バイト
内容 : 37: 低い ~ 32767: 高い [Hz]

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

ビーブ音の周波数の設定を行います。

IOCTL_BEEPDRV_GETHZ

機能

ビーブ音の周波数の取得を行います。

パラメータ

lpInBuf : NULL を指定します。
NInBufSize : 0 を指定します。
lpOutBuf : ビーブ音周波数情報を格納するポインタを指定します。
NOutBufSize : ビーブ音周波数情報を格納するポインタのサイズを指定します。
lpBytesReturned : 実際の実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。
lpOver lapped : NULL を指定します。

ビーブ音周波数情報

データタイプ : ULONG
データサイズ : 4 バイト
内容 : 37: 低い ~ 32767: 高い [Hz]

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

ビーブ音の周波数の取得を行います。

4-12-5 サンプルコード

● ビープ音 周波数

「¥SDK¥Algo¥Sample¥Sample_Beep¥BeepHzCtrl」にビープ音周波数の取得と設定のサンプルコードを用意しています。リスト 4-12-5-1 にサンプルコードを示します。

リスト 4-12-5-1. ビープ音周波数

```
/**
 * ビープ音の周波数変更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¥BeepDrvDD.h"

#define DRIVER_FILENAME "¥¥¥¥. ¥¥BeepDrv"

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

    /*
     * 起動引数からビープ音周波数変更値を取得
     * 37~32767 の範囲で設定します
     * 37 : 低い ~ 32767 : 高い
     */
    if(argc != 2){
        printf("invalid arg¥n");
        return -1;
    }
    sscanf(*(argv + 1), "%d", &set_data);

    /*
     * ビープ音操作ファイルの Open
     */
    hBeep = CreateFile(
        DRIVER_FILENAME,
        GENERIC_READ | GENERIC_WRITE,
        FILE_SHARE_READ | FILE_SHARE_WRITE,
        NULL,
        OPEN_EXISTING,
        0,
```

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

    /*
     * ビープ音周波数変更値を書込み
     */
    ret = DeviceIoControl(
        hBeep,
        IOCTL_BEEPDRV_SETHZ,
        &set_data,
        sizeof(ULONG),
        NULL,
        0,
        &retlen,
        NULL
    );
    if(!ret) {
        printf("DeviceIoControl: IOCTL_BEEPDRV_SETHZ NG¥n");
        CloseHandle(hBeep);
        return -1;
    }

    /*
     * ビープ音周波数変更値を讀出し
     */
    ret = DeviceIoControl(
        hBeep,
        IOCTL_BEEPDRV_GETHZ,
        NULL,
        0,
        &get_data,
        sizeof(ULONG),
        &retlen,
        NULL
    );
    if(!ret) {
        printf("DeviceIoControl: IOCTL_BEEPDRV_GETHZ NG¥n");
        CloseHandle(hBeep);
        return -1;
    }
    printf("Get Beep Sound Hz: %d¥n", get_data);

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

● ビープ音 ON/OFF

「¥SDK¥Algo¥Sample¥Sample_Beep¥BeepOnOff」にビープ音 ON/OFF 制御のサンプルコードを用意しています。
リスト 4-12-5-2 にサンプルコードを示します。

リスト 4-12-5-2. ビープ音 ON/OFF

```
/**
 * ビープ音の ON/OFF 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¥BeepDrvDD.h"

#define DRIVER_FILENAME "¥¥¥¥. ¥¥BeepDrv"

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

    /*
     * 起動引数からビープ音の ON/OFF 変更値を取得します。
     * 1 : ビープ ON
     * 0 : ビープ OFF
     */
    if (argc != 2) {
        printf("invalid arg¥n");
        return -1;
    }
    sscanf(*(argv + 1), "%d", &set_data);

    /*
     * ビープ音操作ファイルの Open
     */
    hBeep = CreateFile(
        DRIVER_FILENAME,
        GENERIC_READ | GENERIC_WRITE,
        FILE_SHARE_READ | FILE_SHARE_WRITE,
        NULL,
        OPEN_EXISTING,
        0,
        NULL
    );

    if (hBeep == INVALID_HANDLE_VALUE) {
```



```
    printf("CreateFile: NG¥n");
    return -1;
}

/*
 * ビープ音 ON/OFF を書き込み
 */
ret = DeviceIoControl(
    hBeep,
    IOCTL_BEEPDRV_SETBEEP,
    &set_data,
    sizeof(ULONG),
    NULL,
    0,
    &retlen,
    NULL
);
if(!ret){
    printf("DeviceIoControl: IOCTL_BEEPDRV_SETBEEP NG¥n");
    CloseHandle(hBeep);
    return -1;
}

/*
 * ビープ音 ON/OFF を読み出し
 */
ret = DeviceIoControl(
    hBeep,
    IOCTL_BEEPDRV_GETBEEP,
    NULL,
    0,
    &get_data,
    sizeof(ULONG),
    &retlen,
    NULL
);
if(!ret){
    printf("DeviceIoControl: IOCTL_BEEPDRV_GETBEEP NG¥n");
    CloseHandle(hBeep);
    return -1;
}
printf("Get Beep Sound OnOff: %d¥n", get_data);

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

4-13 バックアップバッテリーモニタ

4-13-1 バックアップバッテリーモニタについて

産業用組み込み PC シリーズは、BIOS、RTC、外部 RTC のデータを保持するためにバックアップバッテリーとして UPS バッテリーを搭載しています。バックアップバッテリー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ーの状態(正常・低下)を確認することができます。

4-13-2 バックアップバッテリーモニタドライバについて

バックアップバッテリーモニタドライバはバックアップバッテリーの状態を、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。

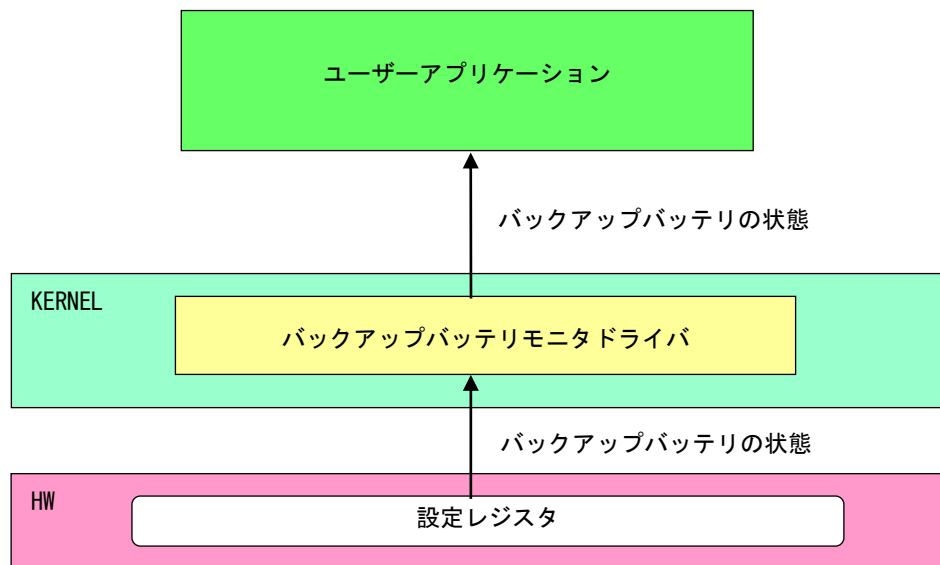


図 4-13-2-1. バックアップバッテリーモニタドライバ

4-13-3 バックアップバッテリーモニタデバイス

バックアップバッテリーモニタドライバはバックアップバッテリー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ーの状態を取得します。

バックアップバッテリーモニタデバイス	
デバイスファイル	¥¥. ¥BackBatDrv
説明	バックアップバッテリーの状態(正常・低下)を取得することができます。
CreateFile	<p>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。</p> <pre> hBackBat = CreateFile("¥¥¥¥. ¥¥BackBatDrv", GENERIC_READ GENERIC_WRITE, FILE_SHARE_READ FILE_SHARE_WRITE, NULL, OPEN_EXISTING, 0, NULL); </pre>
CloseHandle	<p>デバイスハンドルをクローズします。</p> <pre>CloseHandle(hBackBat);</pre>
ReadFile	使用しません。
WriteFile	使用しません。
DeviceIoControl	<p>● IOCTL_BACKBATDRV_GETSTAT バックアップバッテリーの状態を取得します。</p>

4-13-4 DeviceIoControl リファレンス

IOCTL_BACKBATDRV_GETSTAT

機能

バックアップバッテリー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ー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ー状態を格納するポインタのサイズを指定します。
lpBytesReturned : 実際の実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ー状態

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ー状態を取得します。
バックアップバッテリー状態は、「正常」・「低下」を確認できます。

4-13-5 サンプルコード

●バックアップバッテリー状態取得

「¥SDK¥Algo¥Sample¥Sample_BackBat¥BackBatStatus」にバックアップバッテリー状態取得のサンプルコードを用意しています。リスト 4-13-5-1 にサンプルコードを示します。

リスト 4-13-5-1. バックアップバッテリー状態取得

```
/**
 * バックアップバッテリー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ー状態取得
*/
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;
}
```

4-1-4 UPS サービス

4-1-4-1 UPS サービスについて

UPS(Uninterruptible Power Supply) サービスは現在の電源の状態とバッテリー異常が発生した時もしくはバッテリーに切り替わった時の通知、警報をユーザーアプリケーションに通知します。ユーザーアプリケーションは現在のバッテリーの状態を UPS サービスから取得することができます。また通知、警告イベントを取得することができます。

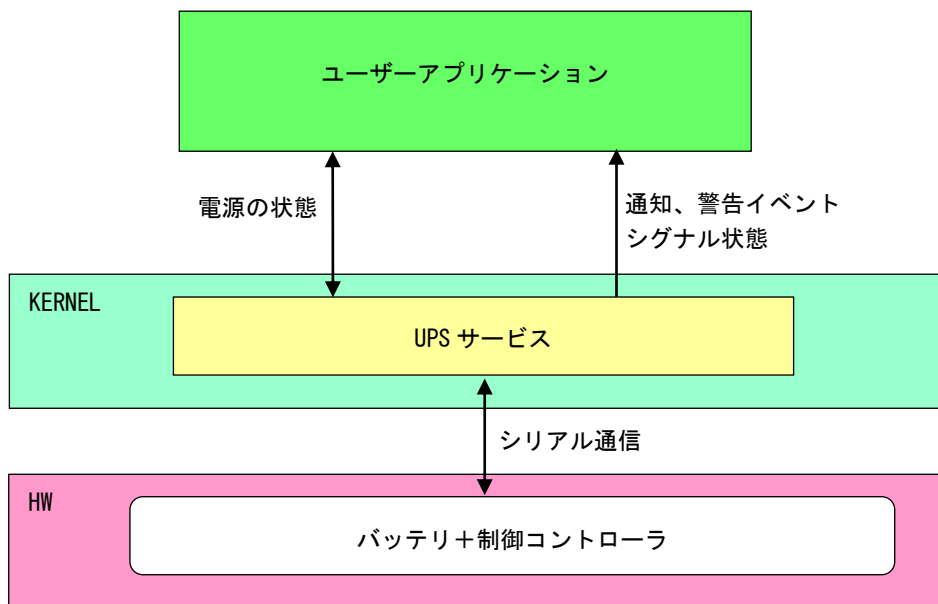


図 4-14-1-1. UPS サービス

UPS サービスの動作は「2-13 UPS Config Tool」で設定することができます。また、設定ツールを使用しないで直接レジストリを設定することも可能です。レジストリ一覧を表 4-14-1-1 に示します。

表 4-14-1-1. UPS Config Tool レジストリ一覧

キー	HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\UpsSvc		
	名前	種類	データ
	FirstMessageDelay	REG_DWORD	電源障害(バッテリー異常)が発生してから通知イベントを送信するまでの時間(0~120(秒))
	MessageInterval	REG_DWORD	通知イベント通知後の通知間隔(5~300(秒))
キー	HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\UpsSvc\Config		
	名前	種類	データ
	NotifyEnable	REG_DWORD	1: 通知イベントの有効, 0: 無効
	EventOnBatteryEnable	REG_DWORD	1: 警告イベントの有効, 0: 無効
	EventOnBatteryWait	REG_DWORD	バッテリー駆動開始から警告イベントを送信するまでの時間(0~60(秒))
	ShutdownOnBatteryEnable	REG_DWORD	1: シャットダウンの有効, 0: 無効
	ShutdownOnBatteryWait	REG_DWORD	バッテリー駆動開始からシャットダウンするまでの時間(0~60(秒))
	EventLogMessageEnable	REG_DWORD	1: イベントログの有効, 0: 無効

※ 上記以外の設定値を変更すると、システムが正常に動かなくなる場合があります。

4-14-2 通知イベント、警告イベント

UPS サービスは電源状態を監視し、ユーザーアプリケーションにイベントを送信します。UPS サービスから送信されるイベントの一覧を表 4-14-2-1 に示します。イベント送信の有効・無効は「UPS Config Tool」で設定できます。「UPS Config Tool」の詳細は、「2-10 UPS Config Tool」を参照してください。

表 4-14-2-1. イベント一覧

	イベント名	内容
通知イベント	UpsNotificationEvent	電源障害(バッテリー異常)が発生したときに設定された時間に応じてイベントを送信します。
警告イベント	UpsBatteryAlarmEvent	バッテリー駆動に切り替わったときに設定された時間に応じてイベントを送信します。

4-14-3 UPS 状態

UPS 状態を UPS サービスから取得することができます。

ユーザーアプリケーションは、UPS サービスと名前付きパイプを使用して通信することができます。

定格電圧 DC24V 接続の有無、充電ステータスを 1 秒周期でホストにアップします。

充電ステータスは表 4-14-3-1 の通りです。

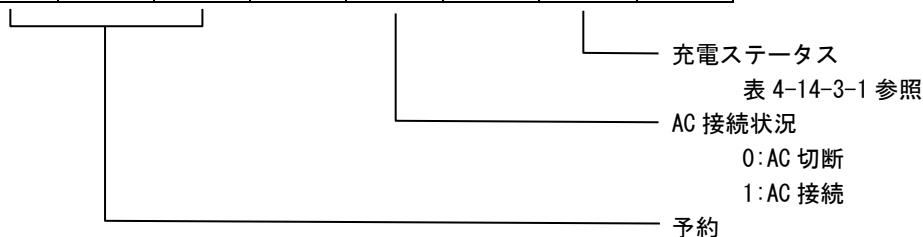
表 4-14-3-1. 充電ステータス

充電ステータス	値	内容
充電中	0	充電条件を満たしたとき
満充電	1	満充電条件を満たしたとき
放電中	2	AC 電源が接続されていない
バッテリー異常	3	バッテリーの温度が高すぎるなど、異常が発生したとき

UPS サービスに以下の 10 バイトを送信すると、UPS サービスから以下の状態を取得することができます。

通信フォーマットは下記の通りです。

B	T	2	5	5	,	0	,	0	LF
---	---	---	---	---	---	---	---	---	----



4-14-4 UPS サービスの使用について

UPS サービスを使用するためのヘッダファイル、UPS サービス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 4-14-4-1 に示します。

表 4-14-4-1. リカバリ/SDK DVD UPS サービス開発用ファイル

DVD-ROM のディレクトリ	内容
¥SDK¥Algo¥Develop	UPS サービスを使用するためのヘッダファイルを格納しています。
¥SDK¥Algo¥Sample¥Sample_UPS	UPS サービスを使用したサンプルコードを格納しています。

4-14-5 サンプルコード

- 電源障害(バッテリー異常)が発生した場合のシグナル状態取得

「¥SDK¥UPS¥Sample¥UpsNotificationEvent.cpp」は、電源障害発生通知 (UpsNotificationEvent) のイベントオブジェクトのシグナル状態を待つプログラム例です。リスト 4-14-5-1 にサンプルコードを示します。

リスト 4-14-5-1. 通知、警告イベントのシグナル状態取得

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

#include "..¥Common¥UpsServiceDD.h"

typedef struct {
    HANDLE hEvent;
    HANDLE hThread;
    volatile BOOL fStart;
    volatile BOOL fFinish;
} UPSEVENT_INFO, *PUPSEVENT_INFO;

/*
 * イベントスレッド関数
 */
DWORD WINAPI UpsNotificationEventProc(void *pData)
{
    PUPSEVENT_INFO info = (PUPSEVENT_INFO)pData;

    printf("UpsNotificationEventProc: Start¥n");

    info->fFinish = FALSE;
    while(1) {

        if(WaitForSingleObject(info->hEvent, INFINITE) != WAIT_OBJECT_0) {
            break;
        }

        if(!info->fStart) {
            break;
        }

        printf("UpsNotificationEventProc: Get Event¥n");
    }
}
```

```
info->fFinish = TRUE;

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

BOOL CreateUpsNotificationEventInfo(PUPSEVENT_INFO info)
{
    DWORD thrd_id;

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

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

    /*
     * イベントオブジェクトの作成
     */
    info->hEvent =
        OpenEvent(SYNCHRONIZE,
                 FALSE,
                 UPS_NOTIFICATION_EVENT_NAME // イベント名を指定します。
                 );

    if(info->hEvent == NULL) {
        printf("CreateUpsNotificationEventInfo: 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) UpsNotificationEventProc,
        (LPVOID) info,
        CREATE_SUSPENDED,
        &thrd_id
    );

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

    return TRUE;
}

void DeleteUpsNotificationEvent(PUPSEVENT_INFO info)
{

```

```
// 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);
}

int main(void)
{
    int    c;
    UPSEVENT_INFO  info;

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

    // リジュームスレッド
    info.fStart = TRUE;
    ResumeThread(info.hThread);

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

    /*
     * イベントオブジェクト、イベントスレッドの破棄
     */
    DeleteUpsNotificationEvent(&info);

    return 0;
}
```

● バッテリ駆動切り替わり時の警告シグナル取得

「¥SDK¥UPS¥Sample¥UpsBatteryAlarmEvent.cpp」は、バッテリー駆動切り替わり時の警告 (UpsBatteryAlarmEvent) のイベントオブジェクトのシグナル状態を待つプログラム例です。リスト 4-14-5-2 にサンプルコードを示します。

リスト 4-14-5-2. UpsBatteryAlarmEvent のシグナル状態取得

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

#include "..¥Common¥UpsServiceDD.h"

typedef struct {
    HANDLE hEvent;
    HANDLE hThread;
    volatile BOOL fStart;
    volatile BOOL fFinish;
} UPSEVENT_INFO, *PUPSEVENT_INFO;

/*
 * イベントスレッド関数
 */
DWORD WINAPI UpsBatteryAlarmEventProc(void *pData)
{
    PUPSEVENT_INFO info = (PUPSEVENT_INFO)pData;

    printf("UpsBatteryAlarmEventProc: Start¥n");

    info->fFinish = FALSE;
    while(1) {

        if(WaitForSingleObject(info->hEvent, INFINITE) != WAIT_OBJECT_0) {
            break;
        }

        if(!info->fStart) {
            break;
        }
        printf("UpsBatteryAlarmEventProc: Get Event¥n");
    }
    info->fFinish = TRUE;

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

BOOL CreateUpsBatteryAlarmEventInfo(PUPSEVENT_INFO info)
{
    DWORD thrd_id;

    info->hEvent = NULL;
```

```
info->hThread = NULL;

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

/*
 * イベントオブジェクトの作成
 */
info->hEvent =
    OpenEvent(SYNCHRONIZE,
             FALSE,
             UPS_BATTERYALARM_EVENT_NAME // イベント名を指定します。
             );
if (info->hEvent == NULL) {
    printf("CreateUpsBatteryAlarmEventInfo: 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) UpsBatteryAlarmEventProc,
    (LPVOID) info,
    CREATE_SUSPENDED,
    &thrd_id
);
if (info->hThread == NULL) {
    CloseHandle(info->hEvent);
    printf("CreateUpsBatteryAlarmEventInfo: CreateThread: NG¥n");
    return FALSE;
}

return TRUE;
}

void DeleteUpsBatteryAlarmEvent (PUPSEVENT_INFO info)
{
    // 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);
}
```

```
}

int main(void)
{
    int    c;
    UPSEVENT_INFO  info;

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

    // Resume Thread
    info.fStart = TRUE;
    ResumeThread(info.hThread);

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

    /*
     * イベントオブジェクト、イベントスレッドの破棄
     */
    DeleteUpsBatteryAlarmEvent(&info);

    return 0;
}
```

●UPS 状態の取得

「¥SDK¥UPS¥Sample¥UpsPipe.cpp」は、UPS の状態を UPS サービスから取得するプログラム例です。リスト 4-14-5-3 にサンプルコードを示します。

※ UPS Config Tool が起動中の時は、名前付きパイプの作成時にエラーになりますので、注意してください。

リスト 4-14-5-3. UPS 状態の取得

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

#include "..¥Common¥UpsServiceDD.h"

char WriteBuf[128];
char ReadBuf[128];

typedef struct {
    HANDLE hThread;
    HANDLE hPipe;

    volatile BOOL fStart;
    volatile BOOL fFinish;
} UPSPPIPE_INFO, *PUPSPPIPE_INFO;

int PipeRead(HANDLE m_hPipe, char *pBuffer, int nLength);
int PipeWrite(HANDLE m_hPipe, char *pBuffer, int nLength);

/*
 * 名前付きパイプスレッド関数
 */
DWORD WINAPI UpsPipeProc(void *pData)
{
    PUPSPPIPE_INFO info = (PUPSPPIPE_INFO)pData;

    printf("UpsPipeProc: Start¥n");

    info->fFinish = FALSE;
    while(1) {

        if(!info->fStart) {
            break;
        }

        // UPS サービスに送信するメッセージ文字列の組み立て
        sprintf(WriteBuf, "BT000,0,0¥n");

        PipeWrite(info->hPipe, WriteBuf, 10);

        Sleep(100);

        while(1) {
```

```
        if(!info->fStart) {
            break;
        }

        if(PipeRead(info->hPipe, ReadBuf, 10) <= 0) {
            Sleep(10);
            continue;
        }
        printf("%s", ReadBuf);

        break;
    }
}
info->fFinish = TRUE;

printf("UpsPipeProc: Finish¥n");

return 0;
}

int PipeRead(HANDLE m_hPipe, char *pBuffer, int nLength)
{
    DWORD    num;

    if(!ReadFile(m_hPipe, pBuffer, (DWORD)nLength, &num, NULL)) {
        return -1;
    }

    return num;
}

int PipeWrite(HANDLE m_hPipe, char *pBuffer, int nLength)
{
    DWORD    num;

    if(!WriteFile(m_hPipe, pBuffer, (DWORD)nLength, &num, NULL)) {
        return -1;
    }

    return (int)num;
}

BOOL CreateUpsPipeInfo(PUPSPIPE_INFO info)
{
    DWORD    thrd_id;
    char     m_szPipeName[MAX_PATH];

    info->hPipe = INVALID_HANDLE_VALUE;
    info->hThread = NULL;

    info->fStart = FALSE;
    info->fFinish = FALSE;
}
```



```
/*
 * 名前付きパイプ(クライアント)の作成
 */
sprintf(m_szPipeName, "¥¥¥¥.¥¥pipe¥¥%s", UPS_GLOBAL_PIPE_NAME); // パイプ名をコピー

info->hPipe =
    CreateFile(
        m_szPipeName,                // パイプ名
        GENERIC_WRITE | GENERIC_READ, // パイプに読み書き
        FILE_SHARE_WRITE | FILE_SHARE_READ, // 複数スレッドからのアクセス
        NULL,                        // セキュリティ
        OPEN_EXISTING,               // 既存のパイプを使用
        FILE_ATTRIBUTE_NORMAL | FILE_FLAG_OVERLAPPED, // ファイル属性なし
        NULL);                       // テンプレートファイルなし

if(info->hPipe == INVALID_HANDLE_VALUE) {
    printf("CreateUpsPipeInfo: CreateFile: NG¥n");
    return FALSE;
}

/*
 * パイプスレッドを生成
 */
info->hThread = CreateThread(
    (LPSECURITY_ATTRIBUTES) NULL,
    0,
    (LPTHREAD_START_ROUTINE) UpsPipeProc,
    (LPVOID) info,
    CREATE_SUSPENDED,
    &thrd_id);

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

return TRUE;
}

void DeleteUpsPipeInfo(PUPSPIPE_INFO info)
{
    // Stop Thread
    info->fStart = FALSE;

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

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

```
    CloseHandle(info->hPipe);
}

int main(void)
{
    int c;
    UPPIPE_INFO    info;

    memset(WriteBuf, 0, 128);
    memset(ReadBuf, 0, 128);

    /*
     * 名前付きパイプ、パイプスレッドの作成
     */
    if( !CreateUpsPipeInfo(&info) ){
        printf("CreateUpsPipeInfo: NG: %n");
        return -1;
    }

    // リジュームスレッド
    info.fStart = TRUE;
    ResumeThread(info.hThread);

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

    /*
     * 名前付きパイプ、パイプスレッドの破棄
     */
    DeleteUpsPipeInfo(&info);

    return 0;
}
```

4-15 S.M.A.R.T. イベント通知

4-15-1 S.M.A.R.T. イベント通知について

S.M.A.R.T. 監視アプリケーション「S.M.A.R.T. Monitor Tool」は異常発生時の通知方法を「Event」に設定した場合、ユーザーアプリケーションにイベント通知を行います。

ユーザーアプリケーションは、S.M.A.R.T. イベント通知として「警告レベル異常イベント」、「重大レベル異常イベント」を取得することができます。

「S.M.A.R.T. Monitor Tool」の詳細は、「2-12-2 S.M.A.R.T. Monitor Tool」を参照してください。

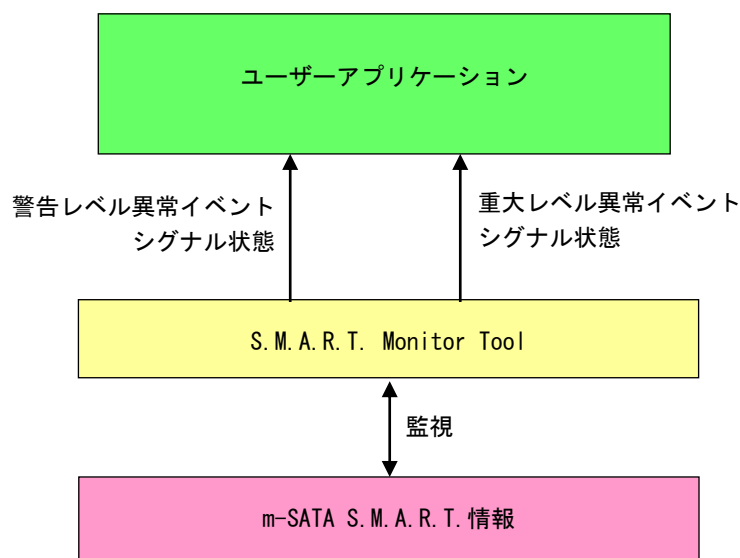


図 4-15-1-1. S.M.A.R.T. イベント通知

表 4-15-1-1. イベント一覧

	イベント名	内容
警告レベル異常イベント	Global¥SmartMonitorWarning N	警告レベル異常発生を通知するイベント N : ストレージ番号 (1~)
重大レベル異常イベント	Global¥SmartMonitorCritical N	重大レベル異常発生を通知するイベント N : ストレージ番号 (1~)

4-15-1 サンプルコード

● S. M. A. R. T. イベント通知のシグナル状態取得

「¥SDK¥Algo¥Sample¥Sample_SMART¥SmartMonitorEvent」に、S. M. A. R. T. イベント通知のシグナル状態取得のサンプルプログラムを用意しています。リスト 4-15-1-1 にサンプルコードを示します。

リスト 4-15-1-1. S. M. A. R. T. イベント通知のシグナル状態取得

```
/**
 * S. M. A. R. T. 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>

#define EVENT_NAME_WARNING_STOR1 "Global¥SmartMonitorWarning1"
#define EVENT_NAME_WARNING_STOR2 "Global¥SmartMonitorWarning2"
#define EVENT_NAME_CRITICAL_STOR1 "Global¥SmartMonitorCritical1"
#define EVENT_NAME_CRITICAL_STOR2 "Global¥SmartMonitorCritical2"

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

enum {
    EVENT_FIN = 0,
    EVENT_WARNING1,
    EVENT_WARNING2,
    EVENT_CRITICAL1,
    EVENT_CRITICAL2,
    MAX_EVENT
};

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

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

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

    /*
     * S. M. A. R. T. 監視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_WARNING1)) {
        printf("EventThread: SMART Monitor Event Warning Level Storage1¥n");
    }

    if(ret == (WAIT_OBJECT_0 + EVENT_WARNING2)) {
        printf("EventThread: SMART Monitor Event Warning Level Storage2¥n");
    }

    if(ret == (WAIT_OBJECT_0 + EVENT_CRITICAL1)) {
        printf("EventThread: SMART Monitor Event Critical Level Storage1¥n");
    }

    if(ret == (WAIT_OBJECT_0 + EVENT_CRITICAL2)) {
        printf("EventThread: SMART Monitor Event Critical Level Storage2¥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);

    /*
     * S. M. A. R. T. 監視イベント 警告レベル ハンドル取得
     */
    hEvent[EVENT_WARNING1] = CreateEvent(NULL, FALSE, FALSE, EVENT_NAME_WARNING_STOR1);
    if(hEvent[EVENT_WARNING1] == NULL) {
        printf("CreateEvent: Warning Storage1 NG¥n");
    }
    hEvent[EVENT_WARNING2] = CreateEvent(NULL, FALSE, FALSE, EVENT_NAME_WARNING_STOR2);
    if(hEvent[EVENT_WARNING2] == NULL) {
        printf("CreateEvent: Warning Storage2 NG¥n");
    }
}
```

```
/*
 * S.M.A.R.T. 監視イベント 重大レベル ハンドル取得
 */
hEvent[EVENT_CRITICAL1] = CreateEvent(NULL, FALSE, FALSE, EVENT_NAME_CRITICAL_STOR1);
if (hEvent[EVENT_CRITICAL1] == NULL) {
    printf("CreateEvent: Critical Storage1 NG\n");
}
hEvent[EVENT_CRITICAL2] = CreateEvent(NULL, FALSE, FALSE, EVENT_NAME_CRITICAL_STOR2);
if (hEvent[EVENT_CRITICAL2] == NULL) {
    printf("CreateEvent: Critical Storage2 NG\n");
}

/*
 * 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;  
}
```

第 5 章 システムリカバリ

本章では、「EC4A シリーズ用 Windows 10 IoT Enterprise リカバリ/SDK DVD」(以下リカバリ DVD と呼びます)を使用したシステムのリカバリとバックアップについて説明します。

5-1 リカバリ DVD について

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

- システムの復旧 (バックアップデータ)
- システムのバックアップ

リカバリを実行するにはリカバリ DVD 以外に以下のものを用意する必要があります。

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

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

- リカバリ実行の流れ

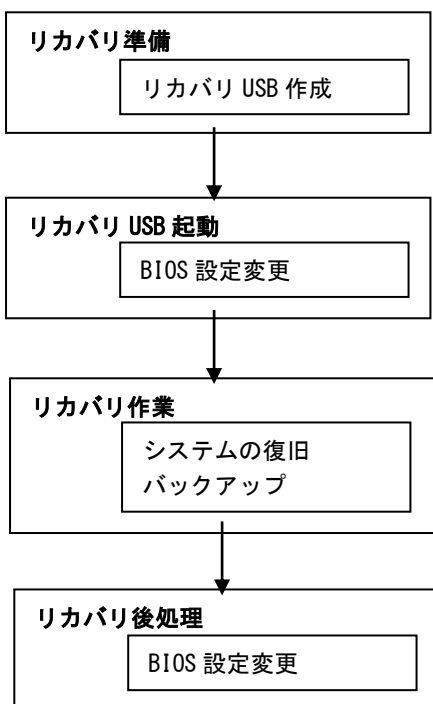


図 5-1-1. リカバリ DVD 使用の流れ

5-1-1 リカバリ準備

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

※注： ここで用意した USB メモリの中身は全て消去されます。
あらかじめバックアップをとるなどしておいてください。

● リカバリ USB 作成手順

- ① Windows が動作する PC にリカバリ DVD を挿入します。
- ② 用意した USB メモリを、手順①の PC に接続します。
- ③ リカバリ DVD の以下のファイルを実行します。
[リカバリ DVD]¥Recovery¥RecoveryUSBImage.exe
- ④ 圧縮ファイルの解凍が始まります。
PC 上の任意の場所に解凍してください。
解凍が完了するまでお待ちください。
解凍が完了すると「RecoveryUSBImage.ddi」というファイルが展開されます。
- ⑤ リカバリ DVD の以下のファイルを実行します。
[リカバリ DVD]¥Recovery¥DDwin¥DDwin.exe
※注： Windows Vista 以降の OS をご使用の場合は管理者権限で起動する必要があります。
- ⑥ DD for Windows というツールが起動します。
- ⑦ 「対象ディスク」の項目に手順②で接続した USB メモリが表示されていることを確認してください。
「ディスク選択」ボタンを押して接続した USB メモリを選択してください。



図 5-1-1-1. DD for Windows

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

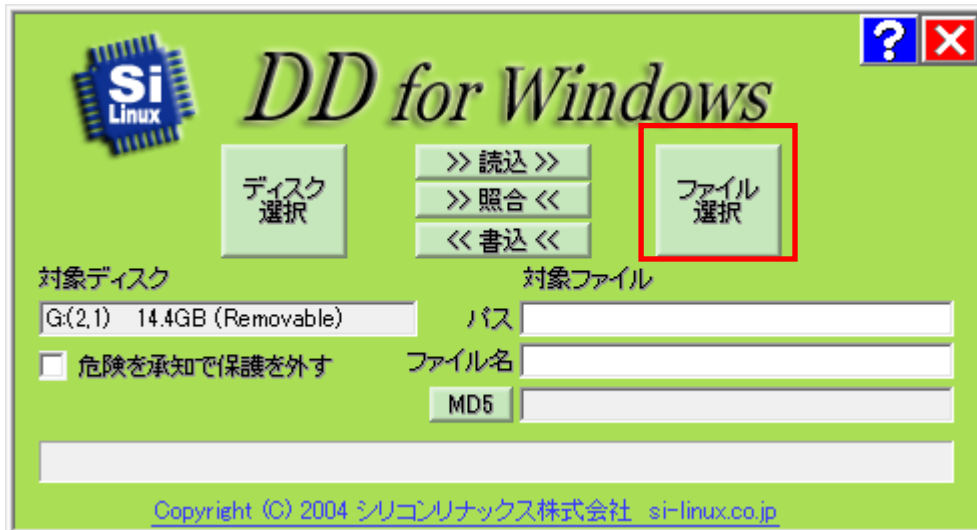


図 5-1-1-2. ファイル選択

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

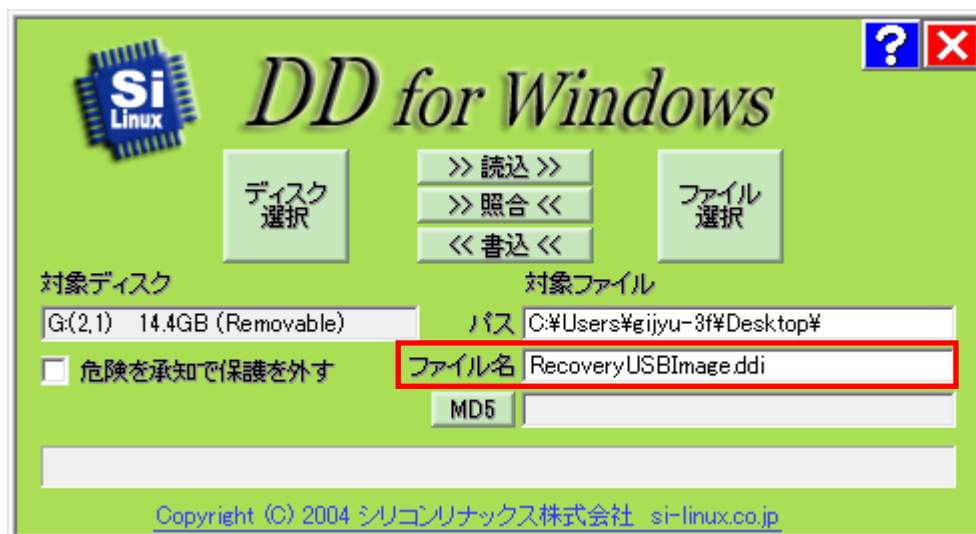


図 5-1-1-3. 対象ファイル

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



図 5-1-1-4. 書き込み開始

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

5-1-2 リカバリ USB 起動

リカバリ USB を起動させる前に、本体に接続されている LAN ケーブル、ストレージ（USB メモリ、SD カードなど）を取り外してください。サブストレージ（m-SATA2）を接続している場合は、サブストレージを取り外してください。

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

- USB キーボード
- USB マウス
- 8GByte 程度の空き容量がある USB 接続可能なストレージメディア (USB メモリなど)

● リカバリ USB 起動手順

- ① LAN ケーブルが接続されている場合は、LAN ケーブルを取り外してください。
- ② リカバリ USB を産業用組込み PC シリーズ本体に接続します。
- ③ USB キーボード、USB マウスを接続します。
- ④ 電源を入れます。BIOS 起動画面が表示されたところで [F2] キーを押し、BIOS 設定画面を表示させます。
- ⑤ BIOS 設定画面が表示されたら、[Advanced] メニューを選択します。（図 5-1-2-1）
- ⑥ [OS Selection] を [Linux] に設定します。

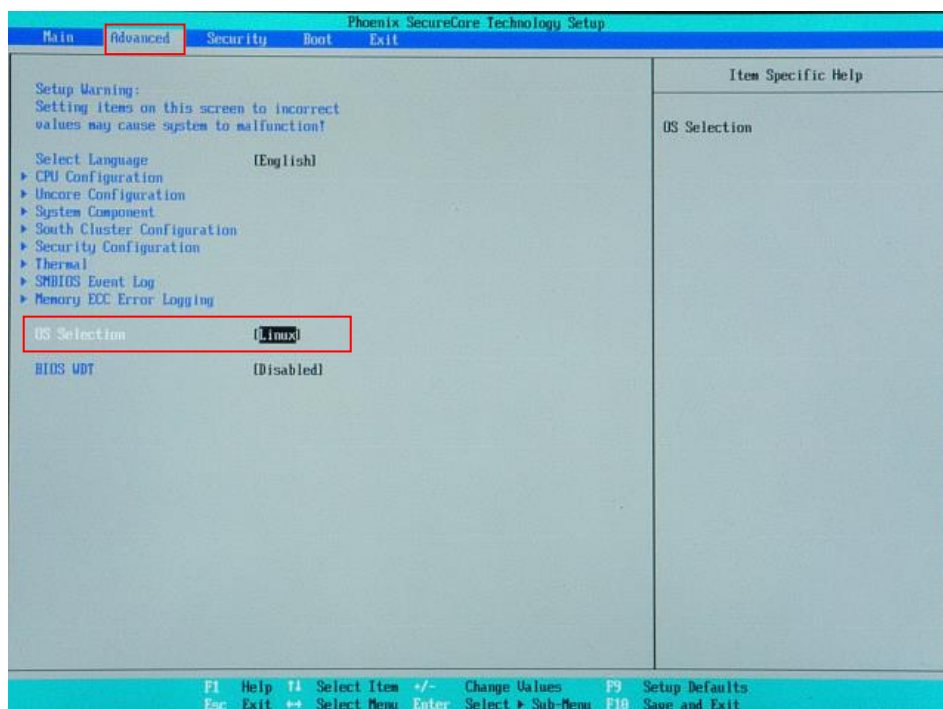


図 5-1-2-1. BIOS 設定 Advanced メニュー

- ⑦ [Boot]メニューを選択します。(図 5-1-2-2)
- ⑧ [USB HDD] (接続した USB メモリ)を[ATA HDD0] (m-SATA メインストレージ)よりも上に設定します。

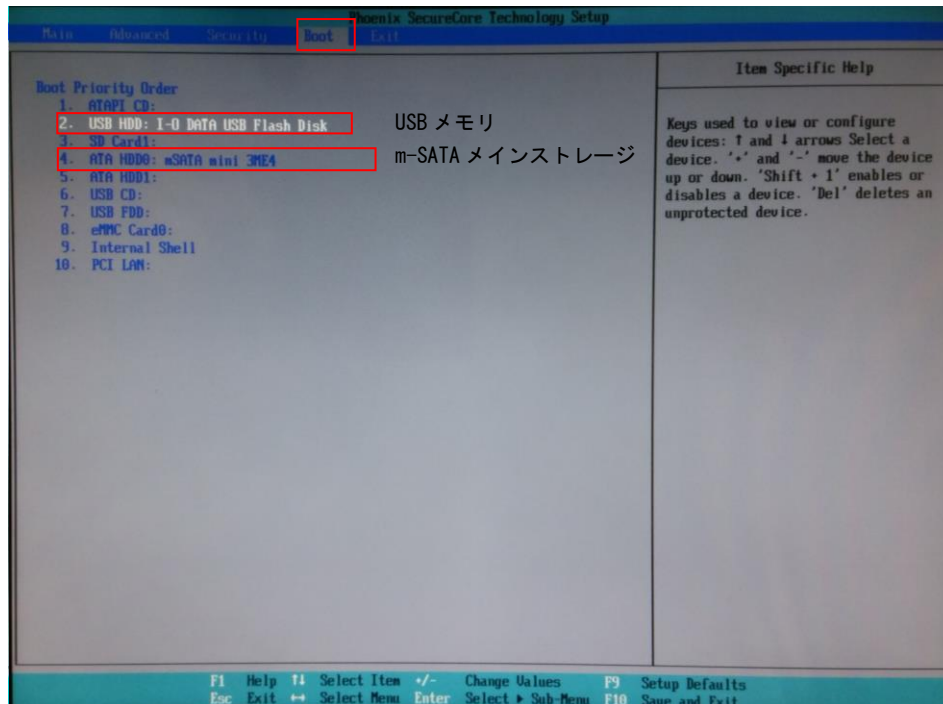


図 5-1-2-2. Boot デバイスの選択

- ⑨ [Exit]メニューを選択します。
- ⑩ [Exit Saving Changes]を実行し、設定を保存して終了します。

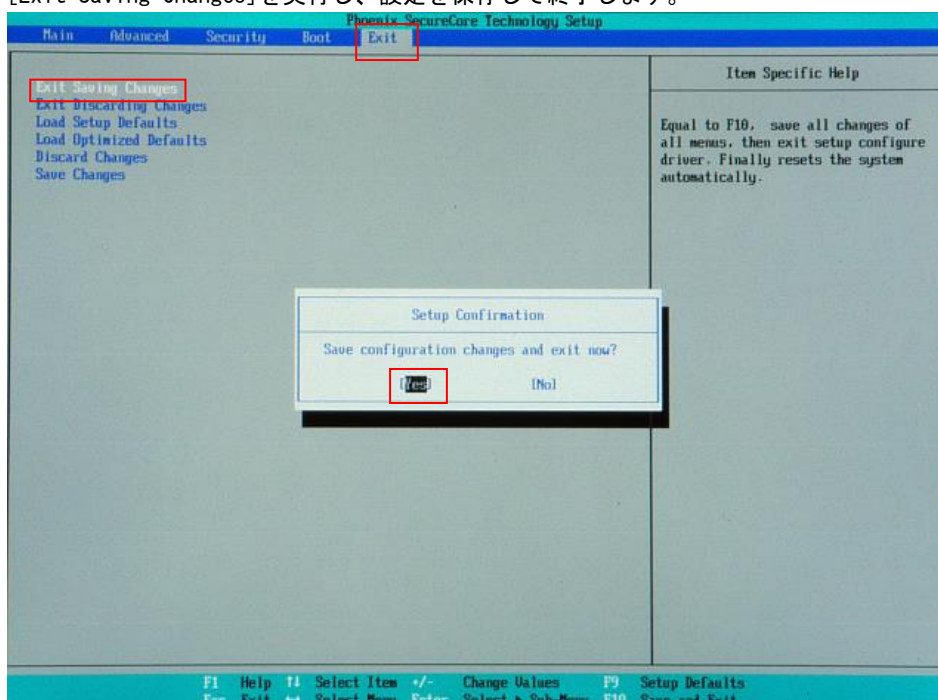


図 5-1-2-3. BIOS 設定 保存と終了

- ⑪ 再起動し、正常にリカバリ USB から起動すると図 5-1-2-4 のリカバリメイン画面が表示されます。

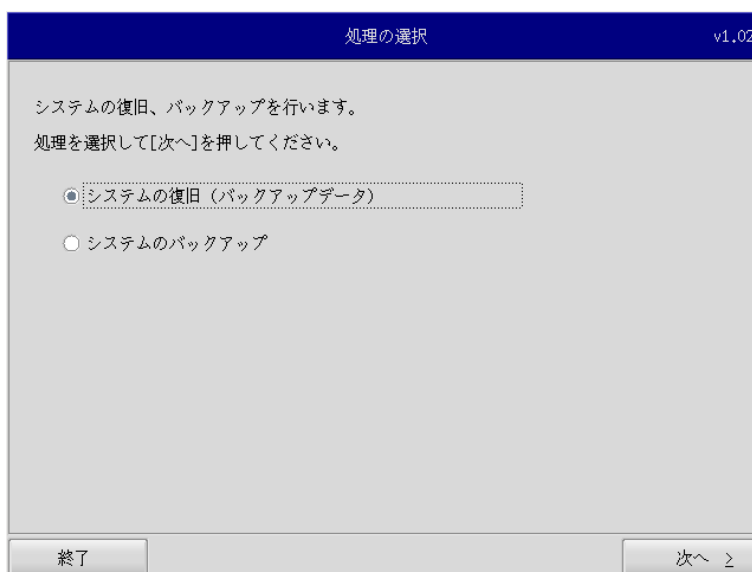


図 5-1-2-4. リカバリメイン画面

5-1-3 リカバリ作業

リカバリメイン画面から処理を選んでリカバリ作業を行います。

- システムの復旧 (バックアップデータ)
- システムのバックアップ

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

5-1-4 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 5-1-4-1)
- ③ [OS Selection] を [Windows] に設定します。

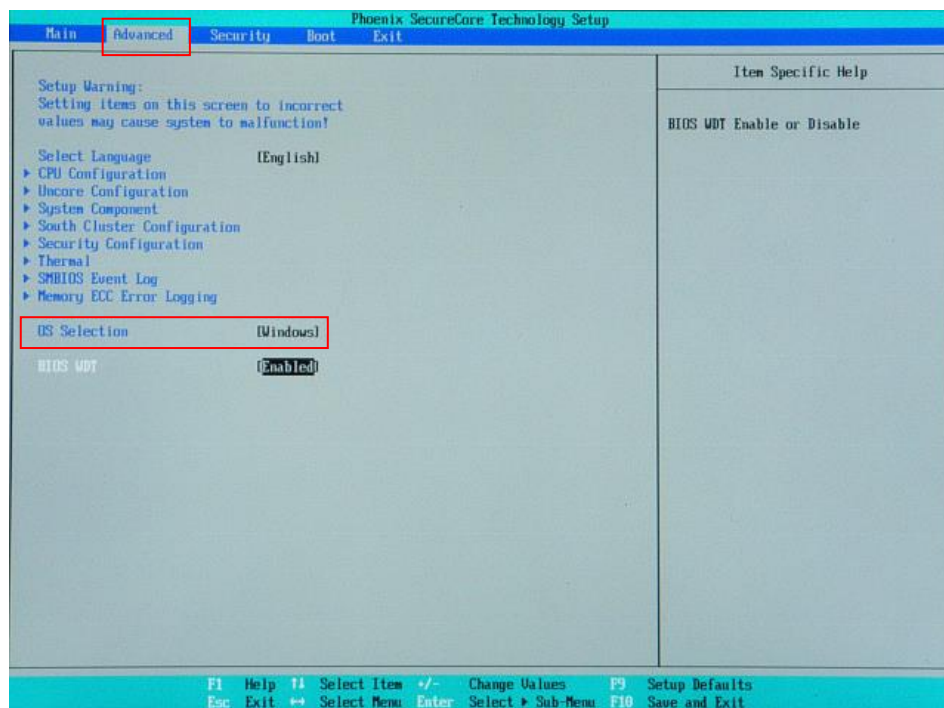


図 5-1-4-1. BIOS 設定 Advanced メニュー

- ④ [Boot]メニューを選択します。(図 5-1-4-2)
- ⑤ [ATA HDD0] (m-SATA メインストレージ) を [USB HDD] (接続した USB メモリ) よりも上に設定します。

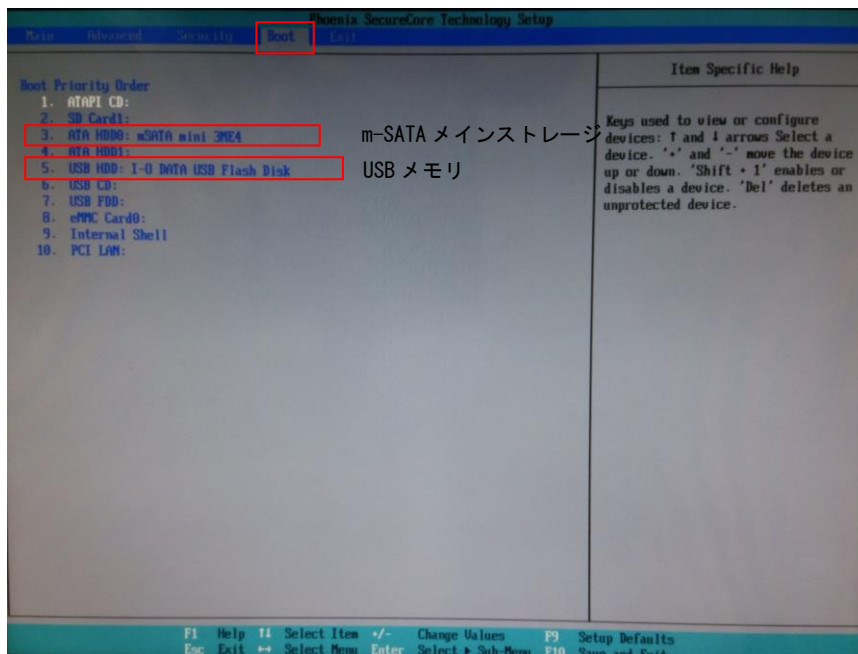


図 5-1-4-2. Boot デバイスの再設定

- ⑥ [Exit]メニューを選択します。
- ⑦ [Exit Saving Changes]を実行し、設定を保存して終了します。

5-2 システムの復旧（バックアップデータ）

工場出荷イメージをメインストレージ（m-SATA1）に書込むことで、システムを工場出荷状態に復旧することができます。

また、「システムのバックアップ」で作成したバックアップファイルを使用して、メインストレージ（m-SATA1）をバックアップファイルの状態に復旧させることができます。

- ※ システムを工場出荷状態へ復旧するとメインストレージにあるデータはすべて消えてしまいます。必要なデータがある場合は、復旧作業を行う前に保存してください。
- ※ 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。
- ※ バックアップファイルは、必ず対象となる本体で作成されたものを使用してください。他の本体のバックアップファイルでは動作しないので注意してください。
- ※ バックアップデータで復旧を行うとメインストレージのデータは、バックアップファイルの状態に戻ります。必要なデータがある場合は、復旧作業を行う前に保存してください。

●システムの復旧（バックアップデータ）の手順

- ① LAN ケーブルが接続されている場合は、LAN ケーブルを取り外してください。
また、工場出荷状態への復旧を行う場合、8GByte 程度の空き容量がある USB 接続可能なストレージメディア（USB メモリなど）にリカバリ DVD 内の工場出荷時イメージファイルをコピーしておいてください。
工場出荷時イメージファイルはリカバリ DVD の以下のフォルダに格納されている xxxxx.img ファイルです。
[リカバリ DVD]¥Recovery¥Image¥
- ② USB メモリ、SD カードなどのストレージメディアが接続されている場合は、ストレージメディアを取り外してください。
- ③ 「5-1-2 リカバリ USB 起動」を参考にリカバリ USB を起動させます。
- ④ リカバリメイン画面（図 5-1-2-4）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 ソフトウェア使用許諾契約確認画面（図 5-2-1）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。

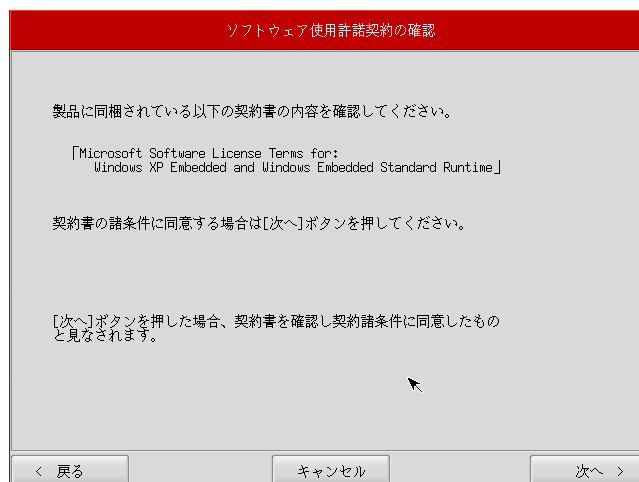


図 5-2-1. ソフトウェア使用許諾契約確認画面

- ⑥ メディアの選択画面 (図 5-2-2) が表示されます。コピー先となるメディアを選択し、「次へ」ボタンを押します。

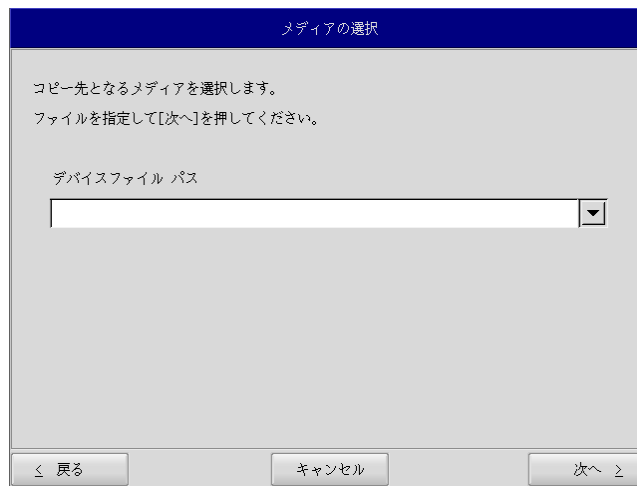


図 5-2-2. メディア選択画面

- ⑦ メディアとパーティション選択画面 (図 5-2-3) が表示されます。あらかじめ用意しておいたストレージメディアを本体に接続し、[メディア情報更新]ボタンを押してください。ストレージメディアのパーティションを選択し、[次へ]ボタンを押します。

ストレージメディアの認識には少し時間がかかります。ストレージメディアを接続してすぐに[メディア情報更新]ボタンを押すと、目的のメディア情報が現れないことがあります。この場合は、1分程度待つて再度、[メディア情報更新]ボタンを押してみてください。

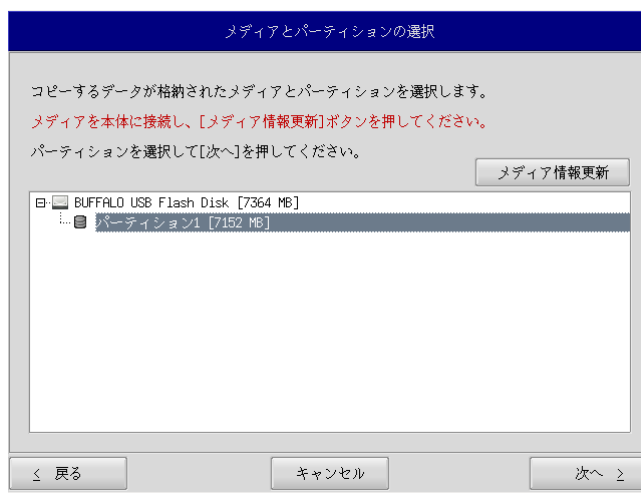


図 5-2-3. メディアとパーティション選択画面

- ⑧ フォルダ選択画面（図 5-2-4）が表示されます。[参照]ボタンを押します。

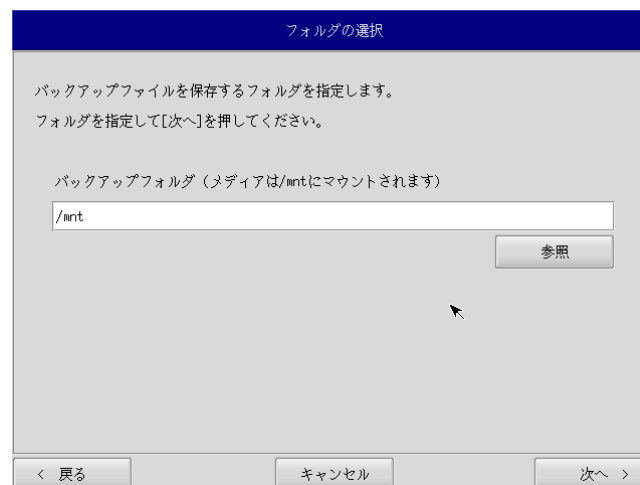


図 5-2-4. フォルダ選択画面

- ⑨ ファイル参照画面（図 5-2-5）が表示されます。接続したストレージメディアは、/mnt にマウントされていますので、/mnt 以下から目的のファイルを探してください。[OK]を押すとファイル選択画面にもどります。

- ※ USB メモリの¥backup¥xxxxxx. img というバックアップファイルを指定する場合 /mnt/backup/xxxxxx. img を指定します。

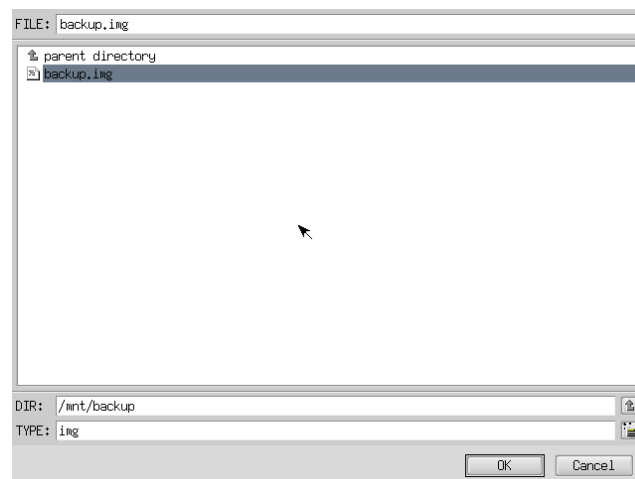


図 5-2-5. ファイル参照画面

- ⑩ ファイル参照画面（図 5-2-5）で指定したバックアップファイルが入力されていることを確認します。[次へ]ボタンを押します。

- ⑪ コンペア処理の選択画面 (図 5-2-6) が表示されます。
データ書き込み時のコンペア処理の有無を選択します。

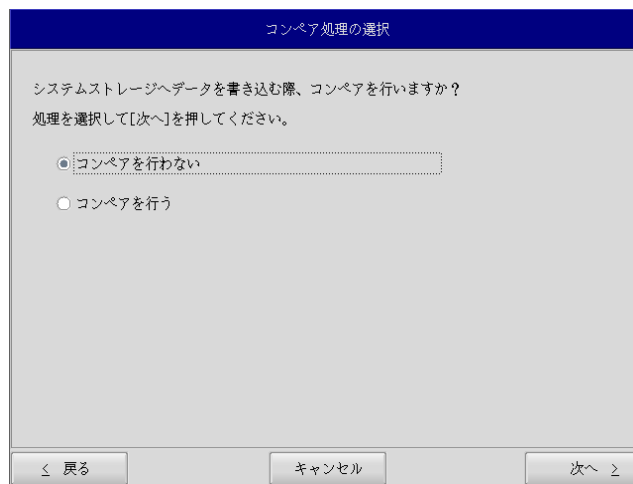


図 5-2-6. コンペア処理選択画面

- ⑫ 確認画面 (図 5-2-7) が表示されます。メディア、パーティション、バックアップファイルを確認します。[次へ] ボタンを押します。

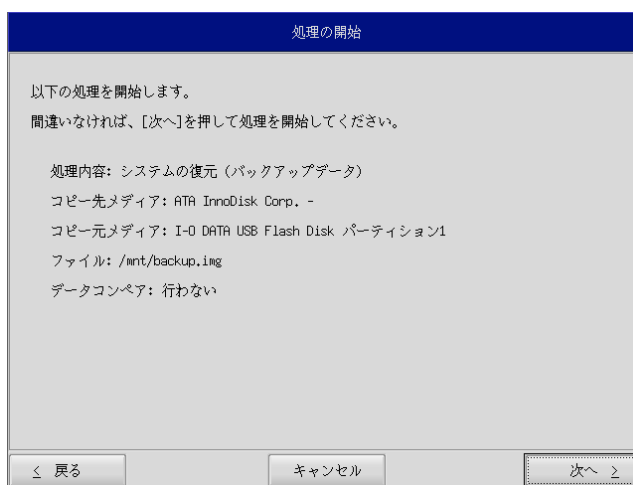


図 5-2-7. 確認画面

- ⑬ 実行中画面（図 5-2-8）が表示され、処理が開始されます。実行中はリカバリ USB メモリ、保存メディアを外さないでください。また、電源を落とさないようにしてください。

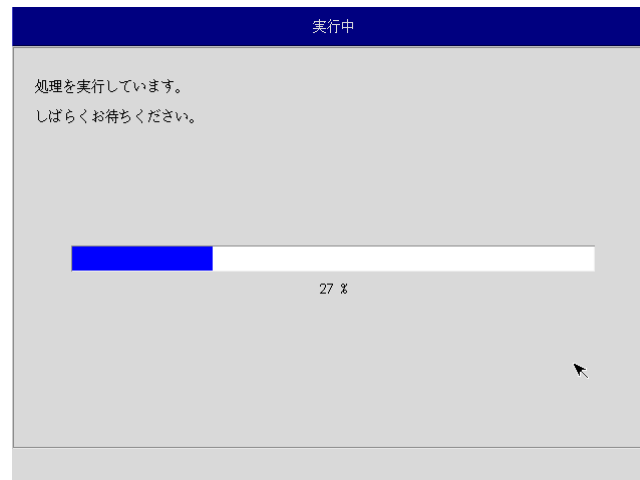


図 5-2-8. 実行中画面

- ⑭ 終了画面（図 5-2-9）が表示されるとバックアップファイルの書き込みは完了です。[終了]ボタンを押して電源を落とし、リカバリ USB メモリ、保存メディアを外します。

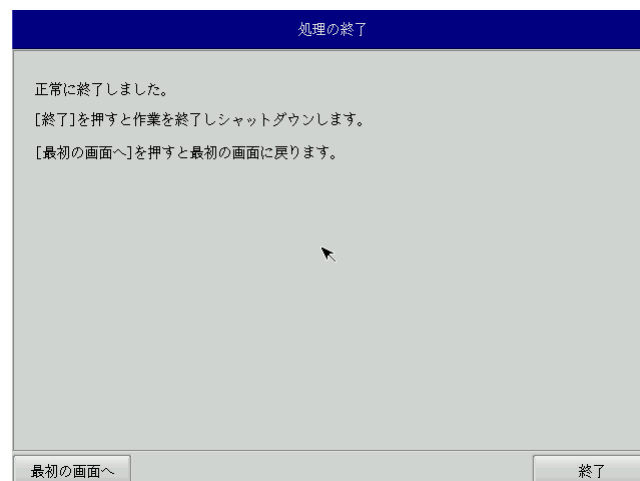


図 5-2-9. 終了画面

- ⑮ 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。
⑯ 「5-1-4 リカバリ後処理」を参考に BIOS 設定を通常使用に戻します。
⑰ デスクトップが表示されて正常に起動すれば、システム復旧は完了です。

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

5-3 システムのバックアップ

メインストレージ (m-SATA1) 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 などでも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 「5-1-2 リカバリ USB 起動」を参考にリカバリ USB を起動させます。
- ④ リカバリメイン画面 (図 5-1-2-4) で [システムのバックアップ] を選択し、[次へ] ボタンを押します。
- ⑤ メディアの選択画面 (図 5-3-1) が表示されます。コピー先となるメディアを選択し、[次へ] ボタンを押します。

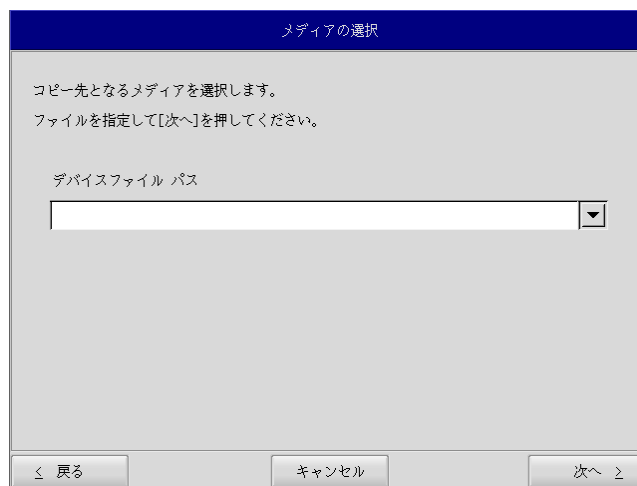


図 5-3-1. メディア選択画面

- ⑥ メディアとパーティション選択画面（図 5-3-2）が表示されます。本体にメディアを接続し、[メディア情報更新] ボタンを押してください。バックアップファイルを保存するメディアのパーティションを選択し、[次へ] ボタンを押します。

メディアの認識には少し時間がかかります。メディアを接続してすぐに[メディア情報更新] ボタンを押すと、目的のメディア情報が現れないことがあります。この場合は、1 分程度待つて再度、[メディア情報更新] ボタンを押してみてください。

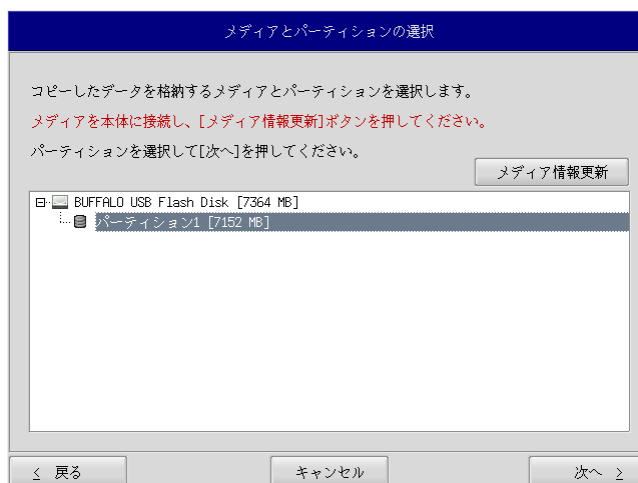


図 5-3-2. メディアとパーティション選択画面

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

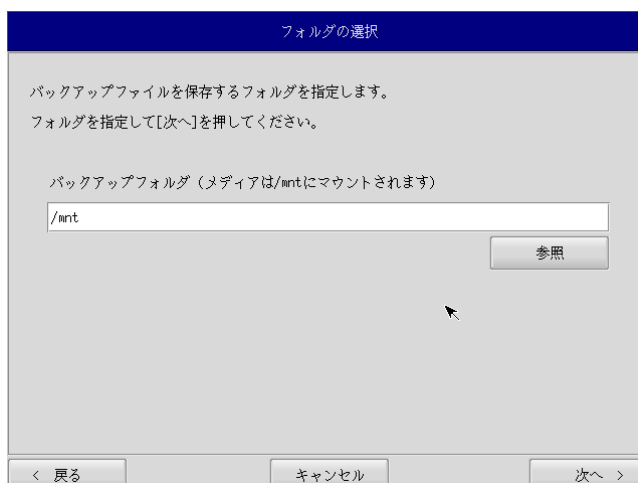


図 5-3-3. フォルダ選択画面

- ⑧ フォルダ参照画面（図 5-3-4）が表示されます。②で接続したパーティションは、/mnt にマウントされますので、/mnt 以下のフォルダを選択してください。[OK]を押すとフォルダ選択画面にもどります。

※ USB メモリに backup というフォルダがあり、このフォルダに保存する場合 /mnt/backup を指定します。

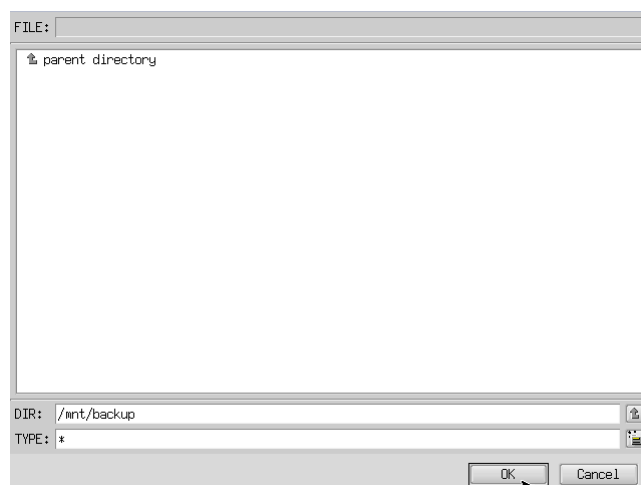


図 5-3-4. フォルダ参照画面

- ⑨ フォルダ選択画面（図 5-3-3）で指定したバックアップフォルダが入力されていることを確認します。[次へ]ボタンを押します。
- ⑩ コンペア処理の選択画面（図 5-3-5）が表示されます。データ書き込み時のコンペア処理の有無を選択します。

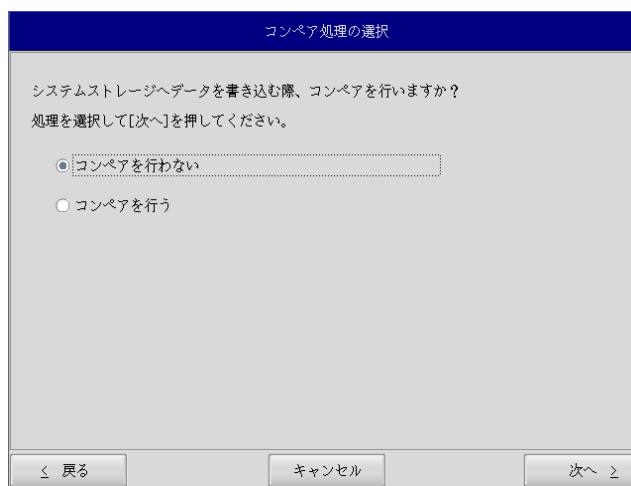


図 5-3-5. コンペア処理選択画面

- ⑪ 確認画面（図 5-3-6）が表示されます。メディア、パーティション、保存ファイルを確認します。
[次へ] ボタンを押します。

※ 保存ファイル名は、現在時刻から自動生成されます。

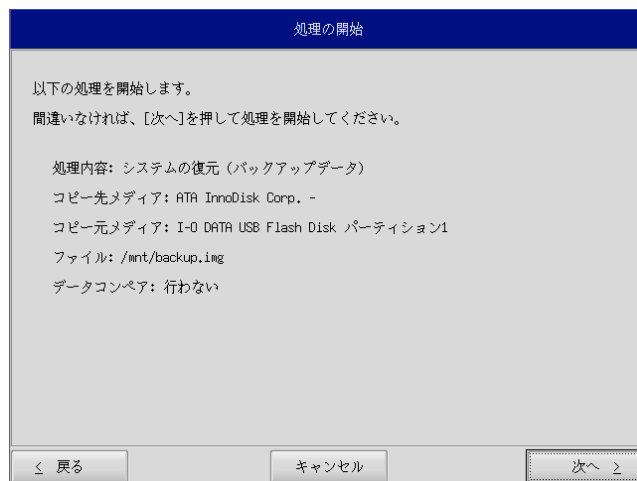


図 5-3-6. 確認画面

- ⑫ 実行中画面（図 5-3-7）が表示され、処理が開始されます。実行中はリカバリ USB メモリ、保存メディアを外さないでください。また、電源を落とさないようにしてください。

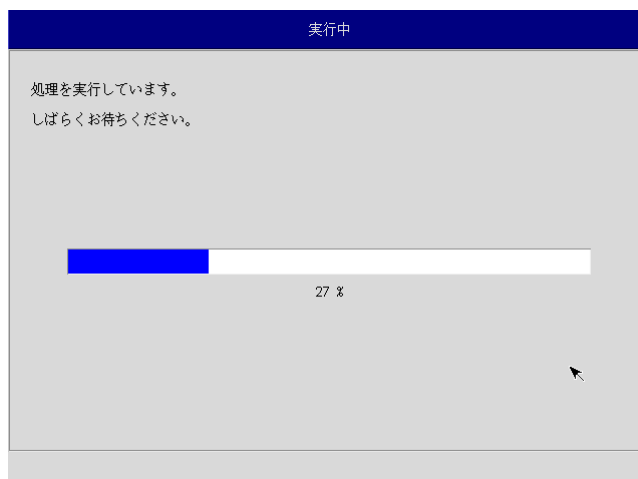


図 5-3-7. 実行中画面

- ⑬ 終了画面（図 5-3-8）が表示されるとバックアップ作業は完了です。[終了]ボタンを押して電源を落としてください。

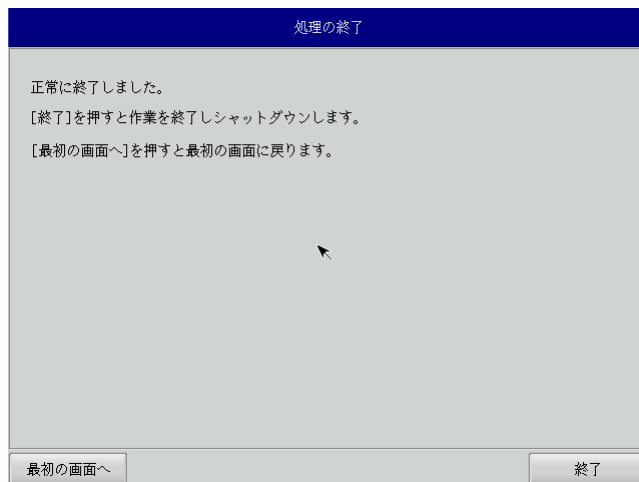


図 5-3-8. 終了画面

- ⑭ 電源を入れ、BIOS 起動画面が表示されたところで [F2] キーを押し、BIOS 設定画面を表示させます。
⑮ 「5-1-4 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。
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。

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

付録

A-1 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 10 IoT Enterprise

【OS 用のアプリケーション開発】

- ・ Windows 10 IoT Enterprise 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 はお客様が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ィアを意味します。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で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ィングデバイス（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ィ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づ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。

 株式会社アルゴシステム

本社
〒587-0021 大阪府堺市美原区小平尾656番地

TEL (072) 362-5067
FAX (072) 362-4856

ホームページ <http://www.algosystem.co.jp>