V850E2/MN4 マイクロコンピュータ
USB MSC(マス・ストレージ・クラス)ドライバ編

R01AN0011JJ0102
Rev.1.02
2011.03.25

要　旨

このアプリケーション・ノートは，V850E2/MN4 マイクロコントローラに内蔵の USB ファンクション・コントローラ用に作成された MSC（マス・ストレージ・クラス）用サンプル・ドライバについて説明します。

主に次に示す内容で構成されます。

- サンプル・ドライバの仕様
- サンプル・ドライバを利用したアプリケーション・プログラム開発のための環境
- サンプル・ドライバを利用するための参考情報

動作確認デバイス

V850E2/MN4(uPD70F3512)搭載 RTE-V850E2/MN4-EB-S

目　次

1. はじめに

1.1 注　意

このアプリケーション・ノートで使用するサンプル・プログラムはあくまで参考用のものであり，当社がこの動作を保証するものではありません。
サンプル・プログラムを使用する場合，ユーザのセット上で十分な評価をしたうえで使用してください。

1.2 対象者

このアプリケーション・ノートは，V850E2/MN4 マイクロコントローラの機能を理解し，それを用いたアプリケーション・システムを開発しようとするユーザを対象とします。

1.3 目　的

このアプリケーション・ノートは，V850E2/MN4 マイクロコントローラに内蔵の USB ファンクション・コントローラを使用するためのサンプル・ドライバの仕様をユーザに理解していただくことを目的とします。

1.4 構　成

このアプリケーション・ノートは，大きく分けて次の内容で構成しています。

- USB 規格の概要
- サンプル・ドライバの仕様
- 開発環境(CubeSuite/Multi(注 1)/ IAR Embedded Workbench (注 2))
- サンプル・ドライバの応用

(注 1)　Multi は Green Hills SoftwareTM, Inc.の登録商標です。
(注 2)　IAR Embedded Workbench は IAR システムズ株式会社の登録商標です。

1.5 読 み 方

このマニュアルの読者には，電気，論理回路，およびマイクロコントローラに関する一般知識を必要とします。

— V850E2/MN4 マイクロコントローラのハードウエア機能，および電気的特性を知りたいとき
　→　別冊の V850E2/MN4 マイクロコントローラのユーザーズマニュアル　ハードウエア編を参照してください。
— V850E2/MN4 マイクロコントローラの命令機能を知りたいとき
　→　別冊の V850E2M　ユーザーズマニュアル　アーキテクチャ編を参照してください。

2. 概　説

このアプリケーション・ノートは，V850E2/MN4 マイクロコントローラに内蔵の USB ファンクション・コントローラ用に作成された MSC（マス・ストレージ・クラス）用サンプル・ドライバについて説明します。主に次に示す内容で構成されます。

- サンプル・ドライバの仕様
- サンプル・ドライバを利用したアプリケーション・プログラム開発のための環境
- サンプル・ドライバを利用するための参考情報

この章では，サンプル・ドライバの概要と，適用対象となるマイクロコントローラについて説明します。

2.1　概　要

2.1.1　USBファンクション・コントローラの特徴

サンプル・ドライバの制御の対象である V850E2/MN4 マイクロコントローラの USB ファンクション・コントローラには，次のような特徴があります。

- USB（Universal Serial Bus Specification）2.0 に準拠
- フル・スピード（12 Mbps）デバイスとして動作
- エンドポイントを次のように構成

表 2-1　V850E2/MN4 マイクロコントローラのエンドポイント構成

エンドポイント名	FIFO サイズ（バイト）	転送タイプ	備考
Endpoint0 Read	64	コントロール転送（IN）	－
Endpoint0 Write	64	コントロール転送（OUT）	－
Endpoint1	64×2	バルク転送 1（IN）	2 バッファ構成
Endpoint2	64×2	バルク転送 1（OUT）	2 バッファ構成
Endpoint3	64×2	バルク転送 2（IN）	2 バッファ構成
Endpoint4	64×2	バルク転送 2（OUT）	2 バッファ構成
Endpoint7	64	インタラプト転送（IN）	－
Endpoint8	64	インタラプト転送(IN)	－

- USB 標準リクエストには自動応答（一部のリクエストを除く）
- バス・パワードとセルフ・パワードを選択可能
- 内部クロックと外部クロックを選択可能 (注 2)

  内部クロック：外部 9.6MHz×内部 20 逓倍÷4 分周（48 MHz）
  　　または外部 7.2 MHz×内部 20 逓倍÷3 分周（48 MHz）

  外部クロック：USBCLK 端子へ入力（fUSB = 48 MHz）

(注 2) サンプル・ドライバでは内部クロックを選択します。

2.1.2　サンプル・ドライバの特徴

V850E2/MN4 マイクロコントローラ向け MSC（マス・ストレージ・クラス）用サンプル・ドライバには，次のような特徴があります。機能や動作の詳細は第 4 章　サンプル・ドライバの仕様を参照してください。

- セルフ・パワード・デバイスとして動作
- ホスト接続時，マス・ストレージ・クラスのバルク・オンリー・デバイスとして認識
- ホスト側からフォーマットすることで，任意のファイル・システム形式にフォーマット可能
- ファイルやフォルダなどのデータを内蔵 RAM に書き込み可能
- 内蔵 RAM に書き込んだファイルやフォルダを読み出し可能
- 次に示すサイズのメモリを占有（ベクタ・テーブルを除く）

  ROM：約 9.0 K バイト

  RAM：約 25.5 K バイト(注 3)

(注 3) RAM（約 25.5 K バイト）の内 24 K バイトをストレージ用のデータ領域として使用します。
このため，ストレージに保存したデータは，デバイスの電源 OFF，Reset SW の押下で初期化されます。

2.1.3　サンプル・ドライバの構成

サンプル・ドライバは CubeSuite 版・Multi 版・IAR Embedded Workbench 版の 3 種類が用意されています。開発環境に合わせてご使用ください。
各サンプル・ドライバは次のようなファイルで構成されています。

（1）CubeSuite 版

CubeSuite 版サンプル・ドライバのファイル構成は，以下のようになっています。

表 2-2　CubeSuite 版サンプル・ドライバのファイル構成

フォルダ	ファイル	概要
src	main.c	メイン・ルーチン
	scsi_cmd.c	SCSI コマンド処理
	usbf850.c	USB 初期化，エンドポイント制御，バルク転送，コントロール転送
	usbf850_storage.c	MSC 固有処理
	cstart.asm	ブートストラップ
include	main.h	main.c 関数プロトタイプ宣言
	scsi.h	SCSI 関連マクロ定義
	usbf850.h	usbf850.c 関数プロトタイプ宣言
	usbstrg_desc.h	ディスクリプタ定義
	usbf850_errno.h	エラー・コード定義
	usbf850_storage.h	usbf850_storage.c 関数プロトタイプ宣言
	usbf850_types.h	ユーザ型宣言
	reg_v850e2mn4.h	USB ファンクション用レジスタ定義

備考　この他，CubeSuite（ルネサスエレクトロニクス製統合開発ツール）用プロジェクト関連ファイル一式も同梱されています。詳細は「5. 2. 1　ホスト環境整備」を参照してください。

(2) Multi 版

Multi 版サンプル・ドライバのファイル構成は，以下のようになっています。

表 2-3　Multi 版サンプル・ドライバのファイル構成

フォルダ	ファイル	概要
src	main.c	メイン・ルーチン
	scsi_cmd.c	SCSI コマンド処理
	usbf850.c	USB 初期化，エンドポイント制御，バルク転送，コントロール転送
	usbf850_storage.c	MSC 固有処理
	initial.s	ブートストラップ
	vector.s	割り込みベクタテーブル宣言
include	main.h	main.c 関数プロトタイプ宣言
	scsi.h	SCSI 関連マクロ定義
	usbf850.h	usbf850.c 関数プロトタイプ宣言
	usbstrg_desc.h	ディスクリプタ定義
	usbf850_errno.h	エラー・コード定義
	usbf850_storage.h	usbf850_storage.c 関数プロトタイプ宣言
	usbf850_types.h	ユーザ型宣言
	reg_v850e2mn4.h	USB ファンクション用レジスタ定義
	df3512_800.h	V850E2/MN4 用レジスタ定義

備考　この他，Multi（Green Hills SoftwareTM, Inc.製統合開発ツール）用プロジェクト関連ファイル一式も同梱されています。詳細は「5. 4. 1 ホスト環境整備」を参照してください。

(3) IAR Embedded Workbench 版

IAR Embedded Workbench 版サンプル・ドライバのファイル構成は，以下のようになっています。

表 2-4　IAR Embedded Workbench 版サンプル・ドライバのファイル構成

フォルダ	ファイル	概要
src	main.c	メイン・ルーチン
	scsi_cmd.c	SCSI コマンド処理
	usbf850.c	USB 初期化，エンドポイント制御，バルク転送，コントロール転送
	usbf850_storage.c	MSC 固有処理
include	main.h	main.c 関数プロトタイプ宣言
	scsi.h	SCSI 関連マクロ定義
	usbf850.h	usbf850.c 関数プロトタイプ宣言
	usbstrg_desc.h	ディスクリプタ定義
	usbf850_errno.h	エラー・コード定義
	usbf850_storage.h	usbf850_storage.c 関数プロトタイプ宣言
	usbf850_types.h	ユーザ型宣言
	reg_v850e2mn4.h	USB ファンクション用レジスタ定義

備考　この他，IAR Embedded Workbench 用プロジェクト関連ファイル一式も同梱されています。詳細は「5. 6. 1 ホスト環境整備」を参照してください。

2.2　V850E2/MN4 マイクロコントローラ

サンプル・ドライバの制御の対象である V850E2/MN4 マイクロコントローラの詳細については，該当する製品のユーザーズマニュアル　ハードウエア編を参照してください。

2.2.1　適用製品

サンプル・ドライバは，次に示す製品に適用できます。

表 2-5　V850E2/MN4 マイクロコントローラ製品一覧

愛　称	品　名	内蔵メモリ		内蔵 USB 機能	割り込み		UM
		フラッシュ・メモリ	RAM		内部 注 4	外部 注 4	
V850E2/MN4	μPD70F3510	1MB	64 KB ＋64KB	Host / Function	180	29	V850E2/MN4 ユーザーズマニュアル ハードウェア編 (R01UH0011JJ0002)
	μPD70F3512	1MB	64 KB ＋64KB	Host / Function	190	29	
	μPD70F3514	1MB	64 KB×2 ＋64KB	Host / Function	196	29	
	μPD70F3515	2MB	64 KB×2 ＋64KB	Host / Function	196	29	

(注 4)　ノンマスカブル割り込みを含みます。

2.2.2　特　徴

V850E2/MN4 には，主に次のような特徴があります。

- 内蔵メモリ
  RAM：　シングル・コア 64K バイト　デュアル・コア 64K バイト×2
  フラッシュ・メモリ：1 M バイト

- フラッシュ・キャッシュ
  シングル・コア：　16K バイト（4 ウェイ・セット・アソシアティブ）
  デュアル・コア：　16K バイト（4 ウェイ・セット・アソシアティブ）×2

- 外部バス・インタフェース
  2 系統のメモリ・コントローラを搭載
  プライマリ・メモリ・コントローラ（SRAM/SDRAM 接続可能）
  セカンダリ・メモリ・コントローラ（SRAM/SDRAM 接続可能）

- シリアル・インタフェース
  アシンクロナス・シリアル・インタフェース UART : 6ch
  クロック同期式シリアル・インタフェース CSI : 6ch
  アシンクロナス・シリアル・インタフェース UART(FIFO) : 4ch
  クロック同期式シリアル・インタフェース CSI(FIFO) : 4ch
  I2C : 6 チャネル
  CAN : 2 チャネル（μ PD70F3512,70F3514,70F3515）
  USB ファンクション・コントローラ :1 チャネル
  USB ホスト・コントローラ :1 チャネル
  イーサネット・コントローラ :1 チャネル（μ PD70F3512, 70F3514, 70F3515）

- DMA コントローラ
  DMA コントローラ :16 チャネル
  DTS : 最大 128 チャネル

3. USBの概要

この章では，サンプル・ドライバが準拠する USB 規格の概要を説明します。
USB（Universal Serial Bus）は共通のコネクタでさまざまな周辺機器をホスト・コンピュータに接続できるようにするためのインタフェース規格です。ハブと呼ばれる分岐点を追加することで最大 127 個の機器を接続でき，Plug&Play で機器を認識できるホットプラグに対応しているなど，従来のインタフェースより柔軟で使いやすくなっています。現在では PC の USB インタフェース搭載率はほぼ 100%になってきており，PC と周辺機器間の標準インタフェースとして定着したと言えます。
USB 規格の策定と管理は USB Implementers Forum（USB-IF）という団体が行っています。USB 規格の詳細は USB-IF の公式ウェブサイト（www.usb.org）を参照してください。

3.1 転送方式

USB 規格では，4 種類の転送方式（コントロール，バルク，インタラプト，アイソクロナス）が定義されています。各転送方式には表 3－1 に示す特徴があります。

表 3-1　USB の転送方式

項目＼転送方式		コントロール転送	バルク転送	インタラプト転送	アイソクロナス転送
特徴		周辺機器の制御などに必要な情報のやりとりに使用される転送方式	非周期的に大量データを扱う転送方式	周期的でバンド幅が低いデータ転送方式	リアルタイム性が要求される転送方式
設定可能なパケット・サイズ	ハイ・スピード 480 Mbps	64 バイト	512 バイト	1-1024 バイト	1-1024 バイト
	フル・スピード 12 Mbps	8, 16, 32, 64 バイト	8, 16, 32, 64 バイト	1-64 バイト	1-1023 バイト
	ロウ・スピード 1.5 Mbps	8 バイト	－	1-8 バイト	－
転送の優先順位		3	3	2	1

3.2　エンドポイント

エンドポイントはホスト・デバイスが通信相手を特定するための情報の 1 つで，0-15 の番号と方向（IN/OUT）で指定されます。エンドポイントは周辺機器で使用するデータ通信経路ごとに用意しなければならず，複数の通信経路で共用できません(注 5)。たとえば，SD カードへの書き込み／読み出しとプリント出力の機能を持った機器の場合，SD カードへの書き込み用エンドポイント，読み出し用エンドポイント，プリント出力用エンドポイントを個別に持つ必要があります。どのような機器でも必ず使用するコントロール転送には，エンドポイント 0 を使用します。
データ通信を行うとき，ホスト・デバイスは機器を特定する USB デバイス・アドレスとともにエンドポイント（番号と方向）を使用して，機器内部の通信先を特定します。
エンドポイントのための物理的な回路として周辺機器内にバッファ・メモリを装備し，USB と通信先（メモリなど）の速度差を吸収する FIFO の役割も果たします。

(注 5) オルタナティブ設定という仕組みを使い，排他的に切り替える方法はあります。

3.3　クラス

USB を介して接続する周辺機器（ファンクション・デバイス）には，その機能によりさまざまなクラスが定義されています。代表的なクラスとしてマス・ストレージ・クラス（MSC），コミュニケーション・デバイス・クラス（CDC），プリンタ・クラス，ヒューマン・インタフェース・デバイス・クラス（HID）などがあります。各クラスにはプロトコルなどで標準仕様が定められているため，これに準拠していれば共通のホスト・ドライバを使用できます。

3.3.1　マス・ストレージ・クラス（MSC）

マス・ストレージ・クラス（MSC）は，USB で接続した記憶装置を認識し，制御するためのインタフェース・クラスで，フラッシュ・メモリ，ハード・ディスクや光ディスク・ストレージ・デバイスなどが対象となります。
MSC の通信方式にはバルク・オンリー転送プロトコルおよび CBI（コントロール／バルク／インタラプト）転送プロトコルの 2 種類があります。バルク・オンリー転送プロトコルではバルク転送のみを用いてデータ転送を行い，CBI 転送プロトコルではバルク転送に加えてコントロール，インタラプト転送を用います。CBI 転送プロトコルは，フル・スピードのフロッピー・ディスク・ドライブのみで使用される方式です。
サンプル・ドライバは，マス・ストレージ・クラス（MSC）のバルク・オンリー転送プロトコルを使用します。
USB のマス・ストレージ・クラス（MSC）仕様に関しては，MSC 規格書「Universal Serial Bus Mass Storage Class Bulk-Only Transport Revision 1.0」を参照してください。

(1) データ転送

バルク・オンリー転送プロトコルでは，コマンド，ステータスおよびデータなどすべての転送をバルク転送で行います。
ホストはバルク・アウト転送を用いて，デバイスに対してコマンドを送信します。
データ転送を伴うコマンドを送信した場合，バルク・イン／バルク・アウト転送によりデータの入出力動作を実行します。
デバイスはバルク・イン転送を用いて，ホストに対してステータス（コマンド実行結果）を送信します。

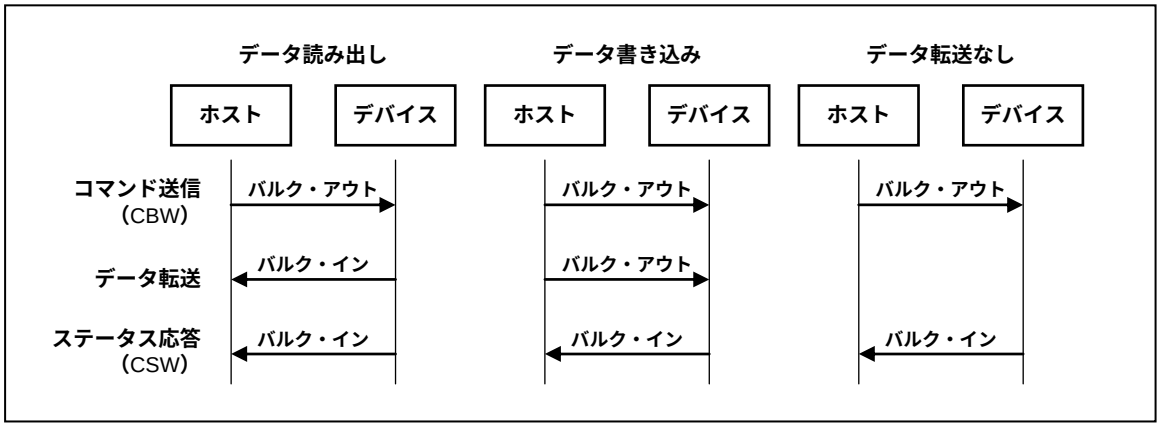


図 3−1　データ転送フロー

(2) CBW フォーマット

コマンド送信時のパケット構造は，Command Block Wrapper（CBW）として定義されています。

表 3-2　CBW フォーマット

ビット / バイト	7	6	5	4	3	2	1	0
0-3	dCBWSignature							
4-7	dCBWTag							
8-11	dCBWDataTransferLength							
12	bmCBWFlags							
13	Reserved				bCBWLUN			
14	Reserved			bCBWCBLength				
15-30	CBWCB							

dCBWSignature：　シグネチャ。0x43425355 固定（リトル・エンディアン）。
dCBWTag：　ホストで任意に番号を定義するタグ。コマンドとステータスを対応させます。
dCBWDataTransferLength：　データ・フェーズで転送するデータの長さ。データがない場合は 0。
bmCBWFlags：　転送方向（ビット 7）。0 = バルク・アウト，1 = バルク・イン。ビット 0-6 は 0 固定。
bCBWLUN：　1 つの USB デバイスに複数のドライブが接続されている場合，そのドライブ番号を指定。
bCBWCBLength：　コマンド・パケットの長さ。
CBWCB：　コマンド・パケット・データ。

(3) CSW のフォーマット

ステータス送信時のパケット構造は，Command Status Wrapper（CSW）として定義されています。

表 3-3　CSW フォーマット

ビット / バイト	7	6	5	4	3	2	1	0
0-3	dCSWSignature							
4-7	dCSWTag							
8-11	dCSWDataResidue							
12	bCSWStatus							

dCSWSignature：　シグネチャ。0x53425355 固定（リトル・エンディアン）。

dCSWTag：　コマンド転送時の dCBWTag に対応させることで，ホストがフェーズの一致を確認。

dCSWDataResidue：　残りのデータ。データ転送中にエラーが発生した場合など，ホストが要求したデータよりもデバイスが応答したデータの方が短い場合，ここに残りのデータ量が設定されます。このため，ステータス（bCSWStatus）が成功を示していても，ここに 0 以外の値が設定されている場合は，デバイスからの応答データが短かったことになります。

dCSWStatus：　CBW 処理結果のステータス。

表 3-4　CBW 処理結果パラメータ

dCSWStatus	設定内容
0x00	成功
0x01	失敗
0x02	フェーズ・エラー
0x03～0xFF	予約

3.3.2　サブクラス

マス・ストレージ・クラス（MSC）では，ホスト・マシンからターゲット・デバイスに送信されるコマンド・フォーマットをサブクラスとして指定します。

(1) サブクラスの種類

USB マス・ストレージ・クラスで規定されているサブクラス・コードを表 3−5 に示します。

表 3-5　USB マス・ストレージ・クラスのサブクラス・コード

サブクラス・コード	規　　格
0x00	SCSI command set not reported（通常は使用しない）
0x01	Reduced Block Commands（RBC），T10 Project 1240-D
0x02	MMC-5（ATAPI）
0x03	SFF-8070i
0x04	USB Floppy Interface（UFI）
0x05	QIC-157（IDE QIC テープ・ドライブ）
0x06	SCSI transparent command set
0x07	Lockable Mass Strage
0x08	IEEE1667
0x09-0xFE	Reserved
0xFF	Specific to device vender

(2) SCSI コマンド

USB メモリや USB カード・リーダーを接続するときは，サブクラスとして SCSI 転送コマンド・セット（0x06）を指定します。SCSI（Small Computer System Interface）はコンピュータと周辺機器をバス型配線で相互接続するためのインタフェース規格です。

CBW の CBWCB（コマンド・パケット・データ）で SCSI コマンドを指定することによりデータ転送や機能設定などを実行します。サンプル・ドライバが対応している SCSI コマンドについては「4. 1. 4　SCSI コマンドへの対応」を参照してください。

3.4　リクエスト

USB 規格では，ホスト・デバイスからすべてのファンクション・デバイスに対してリクエストと呼ばれるコマンドを発行することにより通信が開始されます。リクエストには処理の方向，種類，ファンクション・デバイスのアドレスなどのデータが含まれています。各ファンクション・デバイスはリクエストをデコードして自身に対するリクエストかを判定し，自身に対するリクエストの場合にだけ応答します。

3.4.1　種　類

標準リクエスト，クラス・リクエスト，ベンダ・リクエストの 3 種類があります。
サンプル・ドライバが対応するリクエストについては，「4. 1. 2　リクエストへの対応」を参照してください。

(1) 標準リクエスト

すべての USB 対応機器で共通に使用するリクエストです。bmRequestType フィールドのビット 6，5 の値がともに 0 のとき，そのリクエストは標準リクエストです。各標準リクエストの処理内容については，USB 仕様書（Universal Serial Bus Specification Rev.2.0）を参照してください。

表 3-6　標準リクエスト一覧

リクエスト名	対象ディスクリプタ	概要
GET_STATUS	デバイス	電源（セルフ／バス）とリモート・ウエイクアップの設定の読み取り
	エンドポイント	Halt 状態の読み取り
CLEAR_FEATURE	デバイス	リモート・ウエイクアップのクリア
	エンドポイント	Halt の解除（DATA PID = 0）
SET_FEATURE	デバイス	リモート・ウエイクアップまたはテスト・モードの設定
	エンドポイント	Halt の設定
GET_DESCRIPTOR	デバイス，コンフィギュレーション，ストリング	対象ディスクリプタの読み取り
SET_DESCRIPTOR	デバイス，コンフィギュレーション，ストリング	対象ディスクリプタの変更（オプション）
GET_CONFIGURATION	デバイス	現行設定のコンフィギュレーション値の読み取り
SET_CONFIGURATION	デバイス	コンフィギュレーション値の設定
GET_INTERFACE	インタフェース	対象インタフェースの現行設定のうちオルタナティブ設定値の読み取り
SET_INTERFACE	インタフェース	対象インタフェースのオルタナティブ設定値の設定
SET_ADDRESS	デバイス	USB アドレスの設定
SYNCH_FRAME	エンドポイント	フレーム同期のデータ読み取り

(2) クラス・リクエスト

クラス固有のリクエストです。bmRequestType フィールドのビット 6 の値が 0，ビット 5 の値が 1 のとき，そのリクエストはクラス・リクエストです。
マス・ストレージ・クラス（MSC）のバルク・オンリー転送プロトコルでは次のリクエストに対応する必要があります。

- GET_MAX_LUN（bRequest = 0xFE）
  マス・ストレージ・デバイスの論理装置数（Logical Unit Number）を取得するためのリクエストです。
- MASS_STORAGE_RESET（bRequest = 0xFF）
  マス・ストレージ・デバイスと関連するインタフェースをリセットするためのリクエストです。

(3) ベンダ・リクエスト

ベンダ・リクエストは，ベンダが独自に定義するリクエストです。ベンダ・リクエストを使用する場合，ベンダはそのリクエストに対応するホスト・ドライバを提供する必要があります。
bmRequestType フィールドのビット 6 の値が 1，ビット 5 の値が 0 のとき，そのリクエストはベンダ・リクエストです。

3.4.2 フォーマット

USB リクエストは 8 バイト長で，次のようなフィールドで構成されています。

表 3-7　USB リクエストのフォーマット

オフセット	フィールド		説明
0	bmRequestType		リクエストの属性
		ビット 7	データ転送方向
		ビット 6, 5	リクエスト・タイプ
		ビット 4-0	対象ディスクリプタ
1	bRequest		リクエスト・コード
2	wValue	下位	リクエストで使用する任意の数値
3		上位	
4	wIndex	下位	リクエストで使用するインデックスまたはオフセット
5		上位	
6	wLength	下位	データ・ステージでの転送バイト数（データ長）
7		上位	

3.5　ディスクリプタ

USB 規格では，各ファンクション・デバイス固有の情報を定められた形式でコード化したものをディスクリプタと呼んでいます。ファンクション・デバイスは，ホスト・デバイスからのリクエストに応じてディスクリプタを送信します。

3.5.1　種　類

次に示す 5 種類のディスクリプタが定義されています。

- デバイス・ディスクリプタ
  どのデバイスにも必ず存在するディスクリプタで，対応している USB 仕様のバージョン，デバイス・クラス，プロトコル，Endpoint0 に対する転送で利用可能な最大パケット長，ベンダ ID，プロダクト ID などの基本情報が含まれています。
  GET_DESCRIPTOR_Device リクエストに応答して送信するディスクリプタです。

- コンフィギュレーション・ディスクリプタ
  すべてのデバイスに 1 つ以上存在するディスクリプタで，デバイスの属性（電源供給方法），消費電力などの情報を含みます。
  GET_DESCRIPTOR_Configuration リクエストに応答して送信するディスクリプタです。

- インタフェース・ディスクリプタ
  インタフェースごとに必要なディスクリプタで，インタフェース識別番号，インタフェース・クラス，サポートするエンドポイントの数などが含まれます。
  GET_DESCRIPTOR_Configuration リクエストに応答して送信するディスクリプタです。

- エンドポイント・ディスクリプタ
  インタフェース・ディスクリプタに指定されたエンドポイントごとに必要なディスクリプタで，転送タイプ（転送方向），転送で利用可能な最大パケット長，転送のインターバルを定義します。ただし，Endpoint0 はこのディスクリプタを持ちません。
  GET_DESCRIPTOR_Configuration リクエストに応答して送信するディスクリプタです。

- ストリング・ディスクリプタ
  任意の文字列を含むディスクリプタです。
  GET_DESCRIPTOR_String リクエストに応答して送信するディスクリプタです。

3.5.2　フォーマット

ディスクリプタのサイズとフィールドは，次のように種類ごとに異なります。各フィールドのデータ並びはリトル･エンディアンです。

表 3-8　デバイス・ディスクリプタのフォーマット

フィールド	サイズ（バイト）	説　明
bLength	1	ディスクリプタのサイズ
bDescriptorType	1	ディスクリプタの種類
bcdUSB	2	USB 仕様リリース番号
bDeviceClass	1	クラス・コード
bDeviceSubClass	1	サブクラス・コード
bDeviceProtocol	1	プロトコル・コード
bMaxPacketSize0	1	Endpoint0 の最大パケット・サイズ
idVendor	2	ベンダ ID
idProduct	2	プロダクト ID
bcdDevice	2	デバイスのリリース番号
iManufacturer	1	製造者を表すストリング・ディスクリプタへのインデックス
iProduct	1	製品を表すストリング・ディスクリプタへのインデックス
iSerialNumber	1	デバイスの製造番号を表すストリング・ディスクリプタへのインデックス
bNumConfigurations	1	コンフィギュレーションの数

備考　　ベンダ ID：USB デバイスを開発する各企業が USB-IF から取得する識別番号
　　プロダクト ID：ベンダ ID を取得後，各企業が自社製品に割り振る識別番号

表 3-9　コンフィギュレーション・ディスクリプタのフォーマット

フィールド	サイズ（バイト）	説　明
bLength	1	ディスクリプタのサイズ
bDescriptorType	1	ディスクリプタの種類
wTotalLength	2	コンフィギュレーション，インタフェース，およびエンドポイント・ディスクリプタの総バイト数
bNumInterfaces	1	このコンフィギュレーションが持つインタフェースの数
bConfigurationValue	1	このコンフィギュレーションの識別番号
iConfiguration	1	このコンフィギュレーションを記述するストリング・ディスクリプタへのインデックス
bmAttributes	1	このコンフィギュレーションの特徴
bMaxPower	1	このコンフィギュレーションの最大消費電流（2 □A 単位）

表 3-10　インタフェース・ディスクリプタのフォーマット

フィールド	サイズ（バイト）	説　明
bLength	1	ディスクリプタのサイズ
bDescriptorType	1	ディスクリプタの種類
bInterfaceNumber	1	このインタフェースの識別番号
bAlternateSetting	1	このインタフェースに対するオルタナティブ設定の有無
bNumEndpoints	1	このインタフェースが持つエンドポイントの数
bInterfaceClass	1	クラス・コード
bInterfaceSubClass	1	サブクラス・コード
bInterfaceProtocol	1	プロトコル・コード
iInterface	1	このインタフェースを記述するストリング・ディスクリプタへのインデックス

表 3-11　エンドポイント・ディスクリプタのフォーマット

フィールド	サイズ（バイト）	説　明
bLength	1	ディスクリプタのサイズ
bDescriptorType	1	ディスクリプタの種類
bEndpointAddress	1	このエンドポイントの転送方向 このエンドポイントのアドレス
bmAttributes	1	このエンドポイントの転送タイプ
wMaxPacketSize	2	この転送の最大パケット・サイズ
bInterval	1	このエンドポイントのポーリング間隔

表 3-12　ストリング・ディスクリプタのフォーマット

フィールド	サイズ（バイト）	説　明
bLength	1	ディスクリプタのサイズ
bDescriptorType	1	ディスクリプタの種類
bString	任意	任意のデータ列

4. サンプル・ドライバの仕様

この章では，V850E2/MN4 マイクロコントローラ向け USB マス・ストレージ・クラス（MSC）用サンプル・ドライバの機能と処理内容の詳細，および実装している関数の仕様について説明します。

4.1 概　要

4.1.1 機　能

サンプル・ドライバには次のような処理が実装されています。

(1) メイン・ルーチン

初期化処理を行ったあと，割り込みを待ちます。サスペンド／レジューム割り込みが発生すると，サスペンド／レジューム処理を行います。詳細は「4. 2. 7　サスペンド／レジューム処理」を参照してください。

(2) 初期化

USB ファンクション・コントローラを使用できるようにするため，各種レジスタを操作して設定します。大きく分けて，V850E2/MN4 の CPU レジスタに対する設定と USB ファンクション・コントローラのレジスタに対する設定があります。詳細は「4. 2. 1　CPU 初期化処理」，「4. 2. 2　USB ファンクション・コントローラ初期化処理」を参照してください。

(3) 割り込み処理

INTUSFA0I1 割り込みハンドラでは，コントロール転送用のエンドポイント（Endpoint0）およびバルク・アウト転送（受信）用のエンドポイント（Endpoint2）の状態を監視し，受信したリクエストおよびデータに対応する処理を行います。INTUSFA0I1 割り込みハンドラでは，レジューム割り込み発生時の処理を行います。詳細は「4. 2. 3　USBF 割り込み処理（INTUSFA0I1）」，「4. 2. 4　USBF レジューム割り込み処理（INTUSFA0I2）」を参照してください。

(4) SCSI コマンド処理

受信した CBW データを解析し，SCSI コマンドかどうかを判別します。SCSI コマンドを受信した場合，各コマンドに応じた処理を実行します。詳細は「4. 1. 4　SCSI コマンドへの対応」を参照してください。

4.1.2　リクエストへの対応

表 4−1 にハードウエア（V850E2/MN4）およびファームウエア（サンプル・ドライバ）で定義されている USB リクエストを示します。

表 4-1　USB リクエストの処理

リクエスト名	コード								処　理
	0	1	2	3	4	5	6	7	
標準リクエスト									
GET_INTERFACE	0x81	0x0A	0x00	0x00	0xXX	0xXX	0x01	0x00	HW 自動応答
GET_CONFIGURATION	0x80	0x08	0x00	0x00	0x00	0x00	0x01	0x00	HW 自動応答
GET_DESCRIPTOR Device	0x80	0x06	0x00	0x01	0x00	0x00	0xXX	0xXX	HW 自動応答
GET_DESCRIPTOR Configuration	0x80	0x06	0x00	0x02	0x00	0x00	0xXX	0xXX	HW 自動応答
GET_DESCRIPTOR String	0x80	0x06	0x00	0x03	0x00	0x00	0xXX	0xXX	FW 応答
GET_STATUS Device	0x80	0x00	0x00	0x00	0x00	0x00	0x02	0x00	HW 自動応答
GET_STATUS Interface	0x81	0x00	0x00	0x00	0xXX	0xXX	0x02	0x00	HW 自動 STALL 応答
GET_STATUS Endpoint n	0x82	0x00	0x00	0x00	0xXX	0xXX	0x02	0x00	HW 自動応答
CLEAR_FEATURE Device	0x00	0x01	0x01	0x00	0x00	0x00	0x00	0x00	HW 自動応答
CLEAR_FEATURE Interface	0x01	0x01	0x00	0x00	0xXX	0xXX	0x00	0x00	HW 自動 STALL 応答
CLEAR_FEATURE Endpoint n	0x02	0x01	0x00	0x00	0xXX	0xXX	0x00	0x00	HW 自動応答
SET_DESCRIPTOR	0x00	0x07	0xXX	0xXX	0xXX	0xXX	0xXX	0xXX	FW STALL 応答
SET_FEATURE Device	0x00	0x03	0x01	0x00	0x00	0x00	0x00	0x00	HW 自動応答
SET_FEATURE Interface	0x02	0x03	0xXX	0xXX	0xXX	0xXX	0x00	0x00	HW 自動 STALL 応答
SET_FEATURE Endpoint n	0x02	0x03	0x00	0x00	0xXX	0xXX	0x00	0x00	HW 自動応答
SET_INTERFACE	0x01	0x0B	0xXX	0xXX	0xXX	0xXX	0x00	0x00	HW 自動応答
SET_CONFIGURATION	0x00	0x09	0xXX	0xXX	0x00	0x00	0x00	0x00	HW 自動応答
SET_ADDRESS	0x00	0x05	0xXX	0xXX	0x00	0x00	0x00	0x00	HW 自動応答
クラス・リクエスト									
MASS_STORAGE_RESET	0x21	0xFE	0x00	0x00	0xXX	0xXX	0x00	0x00	FW 応答
GET_MAX_LUN	0xA1	0xFF	0x00	0x00	0xXX	0xXX	0x01	0x00	FW 応答
その他のリクエスト	上記以外								FW STALL 応答

備考　HW：ハードウエア（V850E2/MN4）
FW：ファームウエア（サンプル・ドライバ）
0xXX：不定値

(1) 標準リクエスト

V850E2/MN4 が自動的に応答しないリクエストに対し，サンプル・ドライバは次のような応答処理を行います。

(a) GET_DESCRIPTOR_string

ホストがファンクション・デバイスのストリング・ディスクリプタを取得するためのリクエストです。
このリクエストを受信すると，サンプル・ドライバは要求されたストリング・ディスクリプタの送信処理（コントロール・リード転送）を行います。

(b) SET_DESCRIPTOR

ホストがファンクション・デバイスのディスクリプタを設定するためのリクエストです。
このリクエストを受信すると，サンプル・ドライバは STALL 応答を返します。

(2) クラス・リクエスト

USB マス・ストレージ・クラス（MSC）のバルク・オンリー転送プロトコルのクラス・リクエストに対し，サンプル・ドライバ次のような応答処理を行います。

(a) GET_MAX_LUN

マス・ストレージ・デバイスの論理装置数（Logical Unit Number）を取得するためのリクエストです。
ホストは CBW を送信するときに bCBWLUN フィールドで論理装置の番号を指定します。
サンプル・ドライバは GET_MAX_LUN リクエストを受信すると，0（論理装置数 = 1）を返答します。

表 4-2　GET_MAX_LUN リクエストのフォーマット

bmRequestType	bRequest	wValue	wIndex	wLength	データ
0xA1	0xFE	0x0000	0x0000	0x0001	1 バイト

(b) MASS_STORAGE_RESET

マス・ストレージ・デバイスと関連するインタフェースをリセットするためのリクエストです。
サンプル・ドライバは MASS_STORAGE_RESET リクエストを受信すると，サンプル・ドライバが使用している USB ファンクション・コントローラのインタフェースをリセットします。

表 4-3　GET_MAX_LUN リクエストのフォーマット

bmRequestType	bRequest	wValue	wIndex	wLength	データ
0x21	0xFF	0x0000	0x0000	0x0000	なし

(3) 定義されていないリクエスト

定義されていないリクエストを受信すると，サンプル・ドライバは STALL 応答を返します。

4.1.3　ディスクリプタの設定

サンプル・ドライバでの各ディスクリプタの設定を次に示します。各ディスクリプタの設定は，ヘッダ・ファイル "usbf850_desc.h" に記述されています。

(1) デバイス・ディスクリプタ

GET_DESCRIPTOR_device リクエストに応答して送信されるディスクリプタです。
GET_DESCRIPTOR_device リクエストにはハードウエアが自動的に応答するため，設定内容は USB ファンクション・コントローラの初期化時に USFA0DDn レジスタ（n = 0-17）に格納されます。

表 4-4　デバイス・ディスクリプタの設定

フィールド	サイズ（バイト）	設定値	説　明
bLength	1	0x12	ディスクリプタのサイズ：18 バイト
bDescriptorType	1	0x01	ディスクリプタの種類：デバイス
bcdUSB	2	0x0200	USB 仕様リリース番号：USB 2.0
bDeviceClass	1	0x00	クラス・コード：なし
bDeviceSubClass	1	0x00	サブクラス・コード：なし
bDeviceProtocol	1	0x00	プロトコル・コード：固有プロトコル未使用
bMaxPacketSize0	1	0x40	Endpoint0 の最大パケット・サイズ：64
idVendor	2	0x045B	ベンダ ID：Renesas Electronics
idProduct	2	0x0200	プロダクト ID：V850E2/MN4
bcdDevice	2	0x0001	デバイスのリリース番号：第 1 版
iManufacturer	1	0x01	製造者を表すストリング・ディスクリプタへのインデックス：1
iProduct	1	0x00	製品を表すストリング・ディスクリプタへのインデックス：0
iSerialNumber	1	0x00	デバイスの製造番号を表すストリング・ディスクリプタへのインデックス：0
bNumConfigurations	1	0x01	コンフィギュレーションの数：1

(2) コンフィギュレーション・ディスクリプタ

GET_DESCRIPTOR_configuration リクエストに応答して送信されるディスクリプタです。
GET_DESCRIPTOR_configuration リクエストにはハードウエアが自動的に応答するため，設定内容は USB ファンクション・コントローラの初期化時に USFA0CIEn レジスタ（n = 0-255）に格納されます。

表 4-5　コンフィギュレーション・ディスクリプタの設定

フィールド	サイズ（バイト）	設定値	説　明
bLength	1	0x09	ディスクリプタのサイズ：9 バイト
bDescriptorType	1	0x02	ディスクリプタの種類：コンフィギュレーション
wTotalLength	2	0x0020	コンフィギュレーション，インタフェース，およびエンドポイント・ディスクリプタの総バイト数：32 バイト
bNumInterfaces	1	0x01	このコンフィギュレーションが持つインタフェースの数：1
bConfigurationValue	1	0x01	このコンフィギュレーションの識別番号：1
iConfiguration	1	0x00	このコンフィギュレーションを記述するストリング・ディスクリプタへのインデックス：0
bmAttributes	1	0x80	このコンフィギュレーションの特徴：バス・パワード，リモート・ウエイクアップなし
bMaxPower	1	0x1B	このコンフィギュレーションの最大消費電流：54 mA

(3) インタフェース・ディスクリプタ

GET_DESCRIPTOR_configuration リクエストに応答して送信されるディスクリプタです。
GET_DESCRIPTOR_configuration リクエストにはハードウエアが自動的に応答するため，設定内容は USB ファンクション・コントローラの初期化時に USFA0CIEn レジスタ（n = 0-255）に格納されます。

表 4-6　インタフェース・ディスクリプタの設定

フィールド	サイズ（バイト）	設定値	説　明
bLength	1	0x09	ディスクリプタのサイズ：9 バイト
bDescriptorType	1	0x04	ディスクリプタの種類：インタフェース
bInterfaceNumber	1	0x00	このインタフェースの識別番号：0
bAlternateSetting	1	0x00	このインタフェースに対するオルタナティブ設定の有無：なし
bNumEndpoints	1	0x02	このインタフェースが持つエンドポイントの数：2
bInterfaceClass	1	0x08	クラス・コード：マス・ストレージ・クラス
bInterfaceSubClass	1	0x06	サブクラス・コード：SCSI transparent command set
bInterfaceProtocol	1	0x50	プロトコル・コード：バルク・オンリー転送
iInterface	1	0x00	このインタフェースを記述するストリング・ディスクリプタへのインデックス：0

(4) エンドポイント・ディスクリプタ

GET_DESCRIPTOR_configuration リクエストに応答して送信されるディスクリプタです。
GET_DESCRIPTOR_configuration リクエストにはハードウエアが自動的に応答するため，設定内容は USB ファンクション・コントローラの初期化時に USFA0CIEn レジスタ（n = 0-255）に格納されます。
サンプル・ドライバではエンドポイントを 2 つ使用するため，ディスクリプタも 2 種類設定しています。

表 4-7　Endpoint1（バルク・イン）のエンドポイント・ディスクリプタの設定

フィールド	サイズ（バイト）	設定値	説　　明
bLength	1	0x07	ディスクリプタのサイズ：7 バイト
bDescriptorType	1	0x05	ディスクリプタの種類：エンドポイント
bEndpointAddress	1	0x81	このエンドポイントの転送方向：IN 方向 このエンドポイントのアドレス：1
bmAttributes	1	0x02	このエンドポイントの転送タイプ：バルク
wMaxPacketSize	2	0x0040	この転送の最大パケット・サイズ：64 バイト
bInterval	1	0x00	このエンドポイントのポーリング間隔：0 ms

表 4-8　Endpoint2（バルク・アウト）のエンドポイント・ディスクリプタの設定

フィールド	サイズ（バイト）	設定値	説　　明
bLength	1	0x07	ディスクリプタのサイズ：7 バイト
bDescriptorType	1	0x05	ディスクリプタの種類：エンドポイント
bEndpointAddress	1	0x02	このエンドポイントの転送方向：OUT 方向 このエンドポイントのアドレス：2
bmAttributes	1	0x02	このエンドポイントの転送タイプ：バルク
wMaxPacketSize	2	0x0040	この転送の最大パケット・サイズ：64 バイト
bInterval	1	0x00	このエンドポイントのポーリング間隔：0 ms

(5) ストリング・ディスクリプタ

GET_DESCRIPTOR_string リクエストに応答して送信されるディスクリプタです。
GET_DESCRIPTOR_string リクエストを受信すると，サンプル・ドライバはこのディスクリプタの設定をヘッダ・ファイル "usbf850_desc.h" から取り出して，USB ファンクション・コントローラの USFA0E0W レジスタに格納します。

表 4-9　ストリング・ディスクリプタの設定

（a）String 0

フィールド	サイズ（バイト）	設定値	説　　明
bLength	1	0x04	ディスクリプタのサイズ：4 バイト
bDescriptorType	1	0x03	ディスクリプタの種類：ストリング
bString	2	0x09, 0x04	言語コード：英語（U.S.）

（b）String 1

フィールド	サイズ（バイト）	設定値	説　　明
bLength 注6	1	0x16	ディスクリプタのサイズ：24 バイト
bDescriptorType	1	0x03	ディスクリプタの種類：ストリング
bString 注7	22	–	シリアル番号： V850E2/MN4 ： 020008065010

（注 6）　bString フィールドのサイズにより設定値が異なります。
（注 7）　ベンダにより任意に設定できる領域のため，サイズや設定値は一定ではありません。

4.1.4 SCSIコマンドへの対応

サンプル・ドライバでは，サブクラスとして SCSI 転送コマンド・セット（0x06）を指定しています。サンプル・ドライバの対応する SCSI コマンドを表 4－10 に示します。表 4－10 にないコマンドを受信した場合，STALL 応答を行います。

表 4-10　サンプル・ドライバの対応する SCSI コマンド

コマンド名	コード	バルク転送方向	概　　要
TEST_UNIT_READY	0x00	NO DATA	装置の種類と構成を確認
REQUEST_SENSE	0x03	IN	センス・データ取得
READ6	0x08	IN	データの読み出し
WRITE6	0x0A	OUT	データの書き込み
SEEK	0x0B	NO DATA	データ位置へのシーク動作指示
INQUIRY	0x12	IN	構成情報／属性の取得
MODE_SELECT	0x15	OUT	各種パラメータの設定
MODE_SENSE6	0x1A	IN	各種パラメータの値の読み出し
START_STOP_UNIT	0x1B	NO DATA	媒体のロード／アンロード，モータの起動／停止
PREVENT	0x1E	NO DATA	媒体取り出しの許可／禁止
READ_FORMAT_CAPACITIES	0x23	IN	記憶容量情報の取得
READ_CAPACITY	0x25	IN	容量情報の取得
READ10	0x28	IN	データの読み出し
WRITE10	0x2A	OUT	データの書き込み
WRITE_VERIFY	0x2E	OUT	データの書き込みおよびベリファイの実行
VERIFY	0x2F	NO DATA	ベリファイの実行
SYNCHRONIZE_CACHE	0x35	NO DATA	キャッシュ上に残っているデータを書き込む
WRITE_BUFF	0x3B	OUT	バッファ・メモリへのデータ書き込み
MODE_SELECT10	0x55	OUT	各種パラメータの設定
MODE_SENSE10	0x5A	IN	各種パラメータの値の読み出し

（1）TEST_UNIT_READY コマンド（0x00）

ロジカル・ユニットの状態をイニシエータ（ホスト・デバイス）に通知します。サンプル・ドライバでは，センス・データを初期化して正常終了します。

表 4-11　TEST_UNIT_READY コマンド・フォーマット

ビット / バイト	7	6	5	4	3	2	1	0
0	オペレーション・コード（0x00）							
1	論理ユニット番号（LUN）			Reserved				
2-4	Reserved							
5	Reserved						Flag	Link

(2) REQUEST_SENSE コマンド（0x03）

センス・データをホストに送信します。サンプル・ドライバでは，表 4－14 に示すセンス・データをホストに送信します。

表 4-12　REQUEST_SENSE コマンド・フォーマット

ビット / バイト	7	6	5	4	3	2	1	0
0	オペレーション・コード（0x03）							
1	論理ユニット番号（LUN）			Reserved				
2	Page code							
3	Reserved							
4	追加データ長							
5	Reserved						Flag	Link

表 4-13　REQUEST_SENSE データ・フォーマット

ビット / バイト	7	6	5	4	3	2	1	0
0	VALID	レスポンス・コード						
1	Reserved							
2	Filemark	EOM	ILI	Reserved	センス・キー			
3–6	インフォメーション							
7	追加センス・データ長（n – 7 バイト）							
8–11	コマンド固有情報							
12	ASC（Additional sense code）							
13	ASCQ（Additional sense code qualifier）							
14	FRU（Field Replaceable Unit）コード							
15	SKSV	センス・キー固有情報						
16	センス・キー固有情報							
17	センス・キー固有情報							
18-n	追加センス・データ（データ長可変）							

表 4-14　センス・データ

センス・キー	ASC	ASCQ	概　　要
0x00	0x00	0x00	NO SENSE
0x05	0x00	0x00	ILLEGAL REQUEST
0x05	0x20	0x00	INVALID COMMAND OPERATION CODE
0x05	0x24	0x00	INVALID FIELD IN COMMAND PACKET

(3) READ6 コマンド（0x08）

指定した範囲の論理データ・ブロックのデータをホストに転送します。

表 4-15　READ6 コマンド・フォーマット

ビット / バイト	7	6	5	4	3	2	1	0
0	オペレーション・コード（0x08）							
1	論理ユニット番号（LUN）			論理ブロック・アドレス（LBA）				
2-3	論理ブロック・アドレス（LBA）							
4	転送データ長							
5	Reserved						Flag	Link

(4) WRITE6 コマンド（0x0A）

受信データをストレージ・デバイス上の指定されたブロックに書き込みます。

表 4-16　WRITE6 コマンド・フォーマット

ビット / バイト	7	6	5	4	3	2	1	0
0	オペレーション・コード（0x0A）							
1	論理ユニット番号（LUN）			論理ブロック・アドレス（LBA）				
2-3	論理ブロック・アドレス（LBA）							
4	転送データ長							
5	Reserved						Flag	Link

(5) SEEK コマンド（0x0B）

指定された記録媒体上の位置へのシーク動作を行います。サンプル・ドライバでは，センス・データを初期化して正常終了します。

表 4-17　SEEK コマンド・フォーマット

ビット / バイト	7	6	5	4	3	2	1	0
0	オペレーション・コード（0x0B）							
1	論理ユニット番号（LUN）			論理ブロック・アドレス（LBA）				
2-3	論理ブロック・アドレス（LBA）							
4	Reserved							
5	Reserved						Flag	Link

（6）INQUIRY コマンド（0x12）

デバイスについての構成情報や属性をホストに通知します。サンプル・ドライバでは，INQUIRY_TABLE の値をホストに送信します。

表 4-18　SEEK コマンド・フォーマット

ビット / バイト	7	6	5	4	3	2	1	0
0	オペレーション・コード（0x12）							
1	論理ユニット番号（LUN）			Reserved			CMDDT	EVPD
2	ページ・コード							
3	Reserved							
4	追加データ長							
5	Reserved						Flag	Link

表 4-19　INQUIRY データ・フォーマット

ビット / バイト	7	6	5	4	3	2	1	0
0	識別子				デバイス・タイプ			
1	RMB	デバイス・タイプ修飾子						
2	ISO バージョン		ECMA バージョン			ANSI バージョン		
3	AENC	TmIOP	応答データ形式					
4	追加データ長（n – 4 バイト）							
5-6	Reserved							
7	RelAdr	WBus32	WBus16	Sync	Linked	Reserved	CmdQue	SftRe
8–15	ベンダ ID（ASCII 文字列）							
16-31	プロダクト ID（ASCII 文字列）							
32-35	プロダクト・バージョン（ASCII 文字列）							
36-55	ベンダ固有情報							
56-95	Reserved							
96-n	追加ベンダ固有情報（データ長可変）							

```
UINT8  INQUIRY_TABLE[INQUIRY_LENGTH]={
       0x00,                /*Qualifier, device type code*/
       0x80,                /*RMB, device type modification child*/
       0x02,                /*ISO Version, ECMA Version, ANSI Version*/
       0x02,                /*AENC, TrmIOP, response data form*/
       0x1F,                /*addition data length*/
       0x00,0x00,0x00,      /*reserved*/
       'R','e','n','e','s','a','s',' ',                /*vender ID*/
       'S','t','o','r','a','g','e','F','n','c','D','r','i','v','e','r',
                                                       /*product ID*/
       '0','.','0','1'      /*Product Revision*/
};
```

図 4－1　INQUIRY_TABLE

(7) MODE_SELECT コマンド（0x15）

デバイスのデータ形式などの各種パラメータの設定や変更を行います。サンプル・ドライバでは，MODE_SELECT_TABLE に値を書き込みます。

表 4-20　MODE_SELECT コマンド・フォーマット

ビット / バイト	7	6	5	4	3	2	1	0
0	オペレーション・コード（0x15）							
1	論理ユニット番号（LUN）			PF	Reserved			SP
2-3	Reserved							
4	追加データ長							
5	Reserved						Flag	Link

表 4-21　MODE_SELECT データ・フォーマット

ビット / バイト	7	6	5	4	3	2	1	0
0	モード・パラメータ長							
1	メディア・タイプ							
2	デバイス固有パラメータ							
3	ブロック・ディスクリプタ長							
4	デンシティ・コード							
5-7	ブロック数							
8	Reserved							
9-11	ブロック長							
12	PS	1	ページ・コード					
13	ページ長（n – 13 バイト）							
14-n	モード・パラメータ（データ長可変）							

```
UINT8  MODE_SELECT_TABLE[MODE_SELECT_LENGTH]={
    0x17,               /*length of the mode parameter*/
    0x00,               /*medium type*/
    0x00,               /*device peculiar parameter*/
    0x08,               /*length of the block descriptor*/
    0x00,               /*density code*/
    0x00,0x00,0xC0,     /*number of the blocks*/
    0x00,               /*Reserved*/
    0x00,0x02,0x00,     /*length of the block*/
    0x01,               /*PS, page code*/
    0x0A,               /*length of the page*/
    0x08,0x0B,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00    /*mode parameter*/
};
```

図 4－2　MODE_SELECT_TABLE

(8) MODE_SENSE6 コマンド（0x1A）

デバイスのモード・セレクト・パラメータの値や属性をホストに送信します。サンプル・ドライバでは，MODE_SENSE_TABLE の値をホストに送信します。

表 4-22　MODE_SENSE6 コマンド・フォーマット

ビット / バイト	7	6	5	4	3	2	1	0
0	オペレーション・コード（0x14）							
1	論理ユニット番号（LUN）			Reserved	DBD	Reserved		
2	PC		ページ・コード					
3	Reserved							
4	追加データ長							
5	Reserved						Flag	Link

表 4-23　MODE_SENSE6 データ・フォーマット

ビット / バイト	7	6	5	4	3	2	1	0
0	モード・パラメータ長							
1	メディア・タイプ							
2	デバイス固有パラメータ							
3	ブロック・ディスクリプタ長							
4	デンシティ・コード							
5-7	ブロック数							
8	Reserved							
9-11	ブロック長							
12	PS	Reserved	ページ・コード					
13	ページ長（n – 13 バイト）							
14-n	モード・パラメータ（データ長可変）							

```
UINT8   MODE_SENSE_TABLE[MODE_SENSE_LENGTH]={
   0x17,               /*length of the mode parameter*/
   0x00,               /*medium type*/
   0x00,               /*device peculiar parameter*/
   0x08,               /*length of the block descriptor*/
   0x00,               /*density code*/
   0x00,0x00,0xC0,     /*number of the blocks*/
   0x00,               /*Reserved*/
   0x00,0x02,0x00,     /*length of the block*/
   0x81,               /*PS, page code*/
   0x0A,               /*length of the page*/
   0x08,0x0B,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00   /*mode parameter*/
};
```

図 4－3　MODE_SENSE_TABLE

(9) START_STOP_UNIT コマンド（0x1B）

デバイスへのアクセスを可能にしたり不可能にしたりします。サンプル・ドライバでは，センス・データを初期化して正常終了します。

表 4-24　START_STOP_UNIT コマンド・フォーマット

ビット / バイト	7	6	5	4	3	2	1	0
0	オペレーション・コード（0x1B）							
1	論理ユニット番号（LUN）			Reserved				IMMED
2	Reserved							
3	Reserved							
4	Reserved						Load/Eject	Start
5	Reserved						Flag	Link

(10) PREVENT コマンド（0x1E）

媒体取り出しの許可／禁止を設定します。サンプル・ドライバでは，何も処理せずに正常終了します。

表 4-25　PREVENT コマンド・フォーマット

ビット / バイト	7	6	5	4	3	2	1	0
0	オペレーション・コード（0x1E）							
1	Reserved							
2	Reserved							
3	Reserved							
4	Reserved						Persistent	Prevent
5	Reserved						Flag	Link

（11）READ_FORMAT_CAPACITIES コマンド（0x23）

デバイスの容量（ブロック数，ブロック長）をホストに通知します。サンプル・ドライバでは，READ_FORMAT_CAPACITY_TABLE の値をホストに送信します。

表 4-26　READ_FORMAT_CAPACITIES コマンド・フォーマット

ビット / バイト	7	6	5	4	3	2	1	0
0	オペレーション・コード（0x23）							
1	論理ユニット番号（LUN）			Reserved				
2-6	Reserved							
7-8	転送データ長							
9	Reserved						Flag	Link

表 4-27　READ_FORMAT_CAPACITIES データ・フォーマット

ビット / バイト	7	6	5	4	3	2	1	0
0-2	Reserved							
3	キャパシティ・リスト長（バイト）							
5-7	ブロック数							
8	Reserved							ディスクリプタ・コード
9-11	ブロック長							
12-15	ブロック数							
16	Reserved							
17-19	ブロック長							

```
UINT8  READ_FORMAT_CAPACITY_TABLE[READ_FORM_CAPA_LENGTH]={
       0x00,0x00,0x00,              /* Reserved        */
       0x08,                        /* Capacity List長 */
       0x00,0x00,0x00,0x30,         /* Block数         */
       0x01,                        /* Descriptor Code */
       0x00,0x02,0x00,              /* Block長         */
       0x00,0x00,0x00,0x30,         /* Block数         */
       0x00,                        /* Reserved        */
       0x00,0x02,0x00               /* Block長         */
};
```

図 4−4　READ_FORMAT_CAPACITY_TABLE

（12）READ_CAPACITY コマンド（0x25）

デバイス上のデータ容量をホストに通知します。サンプル・ドライバでは,
READ_CAPACITY_TABLE の値をホストに送信します。

表 4-28　READ_CAPACITY コマンド・フォーマット

ビット / バイト	7	6	5	4	3	2	1	0
0	オペレーション・コード（0x25）							
1	論理ユニット番号（LUN）			Reserved				RA
2-8	Reserved							
9	Reserved						Flag	Link

表 4-29　READ_CAPACITY データ・フォーマット

ビット / バイト	7	6	5	4	3	2	1	0
0-3	論理ブロック・アドレス（LBA）							
4-7	ブロック長（バイト）							

```
UINT8  READ_CAPACITY_TABLE[8]={     /*big endian*/
   0x00,0x00,0x00,0x2F,             /*number of the outline reason blocks - 1*/
   0x00,0x00,0x02,0x00              /*size of the data block(Byte数)*/
};
```

図 4－5　READ_CAPACITY_TABLE

（13）READ10 コマンド（0x28）

指定した範囲の論理データ・ブロックのデータをホストに転送します。

表 4-30　READ10 コマンド・フォーマット

ビット / バイト	7	6	5	4	3	2	1	0
0	オペレーション・コード（0x28）							
1	論理ユニット番号（LUN）			OPD	FUA	Reserved		RA
2-5	論理ブロック・アドレス（LBA）							
6	Reserved							
7-8	転送データ長							
9	Reserved						Flag	Link

（14）WRITE10 コマンド（0x2A）

受信データをデバイス上の指定されたブロックに書き込みます。

表 4-31　WRITE10 コマンド・フォーマット

ビット / バイト	7	6	5	4	3	2	1	0
0	オペレーション・コード（0x2A）							
1	論理ユニット番号（LUN）			OPD	FUA	EBP	TSR	RA
2-5	論理ブロック・アドレス（LBA）							
6	Reserved							
7-8	転送データ長							
9	Reserved						Flag	Link

（15）WRITE_VERIFY コマンド（0x2E）

受信データをデバイス上の指定されたブロックに書き込みます。書き込み後，データの正常性を確認します。サンプル・ドライバでは，書き込みのみを行います。

表 4-32　WRITE_VERIFY コマンド・フォーマット

ビット / バイト	7	6	5	4	3	2	1	0
0	オペレーション・コード（0x2E）							
1	論理ユニット番号（LUN）			OPD	FUA	EBP	BYTCHK	RA
2-5	論理ブロック・アドレス（LBA）							
6	Reserved							
7-8	転送データ長							
9	Reserved						Flag	Link

（16）VERIFY コマンド（0x2F）

デバイス上のデータの正常性を確認します。サンプル・ドライバでは，何も処理せずに終了します。

表 4-33　VERIFY コマンド・フォーマット

ビット / バイト	7	6	5	4	3	2	1	0
0	オペレーション・コード（0x2F）							
1	論理ユニット番号（LUN）			OPD	Reserved		BYTCHK	RA
2-5	論理ブロック・アドレス（LBA）							
6	Reserved							
7-8	転送データ長							
9	Reserved						Flag	Link

（17）SYNCHRONIZE_CACHE コマンド（0x35）

指定範囲のブロックについて，キャッシュ・メモリと媒体の値を一致させます。サンプル・ドライバでは，センス・データを初期化して正常終了します。

表 4-34　SYNCHRONIZE_CACHE コマンド・フォーマット

ビット / バイト	7	6	5	4	3	2	1	0
0	オペレーション・コード（0x35）							
1	論理ユニット番号（LUN）			Reserved			IMMED	RA
2-5	論理ブロック・アドレス（LBA）							
6	Reserved							
7-8	転送データ長							
9	Reserved						Flag	Link

（18）WRITE_BUFF コマンド（0x3B）

メモリ（データ・バッファ）にデータを書き込みます。サンプル・ドライバでは，データを読み捨てて，正常終了します。

表 4-35　WRITE_BUFF コマンド・フォーマット

ビット / バイト	7	6	5	4	3	2	1	0
0	オペレーション・コード（0x3B）							
1	論理ユニット番号（LUN）			OPD	FUA	EBP	Reserved	RA
2-5	論理ブロック・アドレス（LBA）							
6	Reserved							
7-8	転送データ長							
9	Reserved						Flag	Link

（19）MODE_SENSE10 コマンド（0x5A）

デバイスのモード・セレクト・パラメータの値や属性をホストに通知します。サンプル・ドライバでは，MODE_SENSE10_TABLE の値をホストに送信します。

表 4-36　MODE_SENSE10 コマンド・フォーマット

ビット / バイト	7	6	5	4	3	2	1	0
0	オペレーション・コード（0x5A）							
1	Reserved			LLBAA	DBD	Reserved		
2	PC		ページ・コード					
3-6	Reserved							
7-8	追加データ長							
9	Reserved						Flag	Link

表 4-37　MODE_SENSE10 データ・フォーマット

ビット / バイト	7	6	5	4	3	2	1	0
0	モード・パラメータ長							
1	メディア・タイプ							
2	デバイス固有パラメータ							
3	ブロック・ディスクリプタ長							
4	デンシティ・コード							
5-7	ブロック数（0x0000C0）							
8	Reserved							
9-11	ブロック長（0x000200）							
12	PS	Reserved	ページ・コード					
13	ページ長（n – 13 バイト）							
14-n	モード・パラメータ（データ長可変）							

```
UINT8   MODE_SENSE10_TABLE[MODE_SENSE10_LENGTH]={
   0x00,0x1A,          /*length of the mode parameter*/
   0x00,               /*medium type*/
   0x00,               /*device peculiar parameter*/
   0x00,0x00,          /*Reserved*/
   0x00,0x08,          /*length of the block descriptor*/
   0x00,               /*density code*/
   0x00,0x00,0xC0,     /*number of the blocks*/
   0x00,               /*Reserved*/
   0x00,0x02,0x00,     /*length of the block*/
   0x81,               /*PS, page code*/
   0x0A,               /*length of the page*/
   0x08,0x0B,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00   /*mode parameter*/
};
```

図 4−6　READ_CAPACITY_TABLE

（20）MODE_SELECT10 コマンド（0x55）

デバイスのデータ形式などの各種パラメータの設定や変更を行います。サンプル・ドライバでは，MODE_SELECT10_TABLE に値を書き込みます。

表 4-38　MODE_SELECT10 コマンド・フォーマット

ビット / バイト	7	6	5	4	3	2	1	0
0	オペレーション・コード（0x55）							
1	論理ユニット番号（LUN）			PF	Reserved			SP
2-6	Reserved							
7-8	追加データ長							
9	Reserved						Flag	Link

表 4-39　MODE_SELECT10 データ・フォーマット

ビット / バイト	7	6	5	4	3	2	1	0
0	モード・パラメータ長							
1	メディア・タイプ							
2	デバイス固有パラメータ							
3	ブロック・ディスクリプタ長							
4	デンシティ・コード							
5-7	ブロック数							
8	Reserved							
9-11	ブロック長							
12	PS	1	ページ・コード					
13	ページ長（n – 13 バイト）							
14-n	モード・パラメータ（データ長可変）							

```
UINT8  MODE_SELECT10_TABLE[MODE_SELECT10_LENGTH]={
    0x00,0x1A,          /*length of the mode parameter*/
    0x00,               /*medium type*/
    0x00,               /*device peculiar parameter*/
    0x00,0x00,          /*Reserved*/
    0x00,0x08,          /*length of the block descriptor*/
    0x00,               /*density code*/
    0x00,0x00,0xC0,     /*number of the blocks*/
    0x00,               /*Reserved*/
    0x00,0x02,0x00,     /*length of the block*/
    0x01,               /*PS, page code*/
    0x0A,               /*length of the page*/
    0x08,0x0B,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00    /*mode parameter*/
};
```

図 4－7　MODE_SELECT10_TABLE

4.2　各部の動作

サンプル・ドライバを実行すると，次のような一連の処理を行います。ここでは，それぞれの処理について説明します。

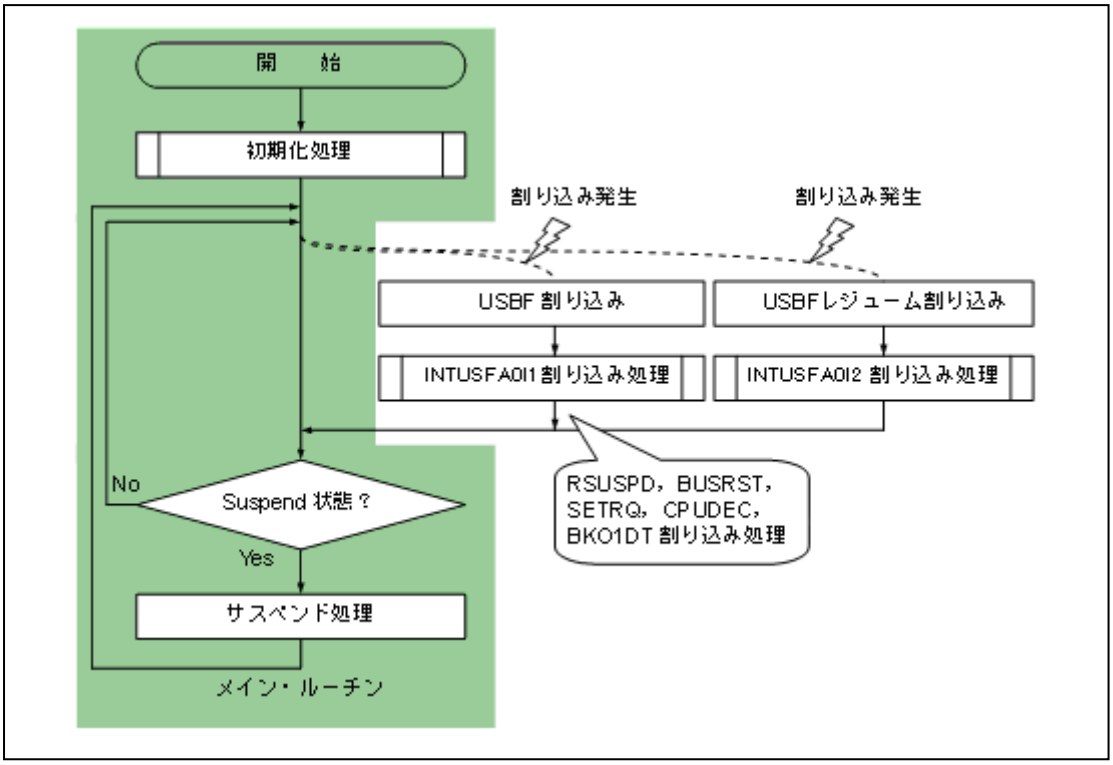


図 4−8　サンプル・ドライバの処理フロー

4.2.1　CPU初期化処理

USB ファンクション・コントローラを使用するために必要な項目を設定します。

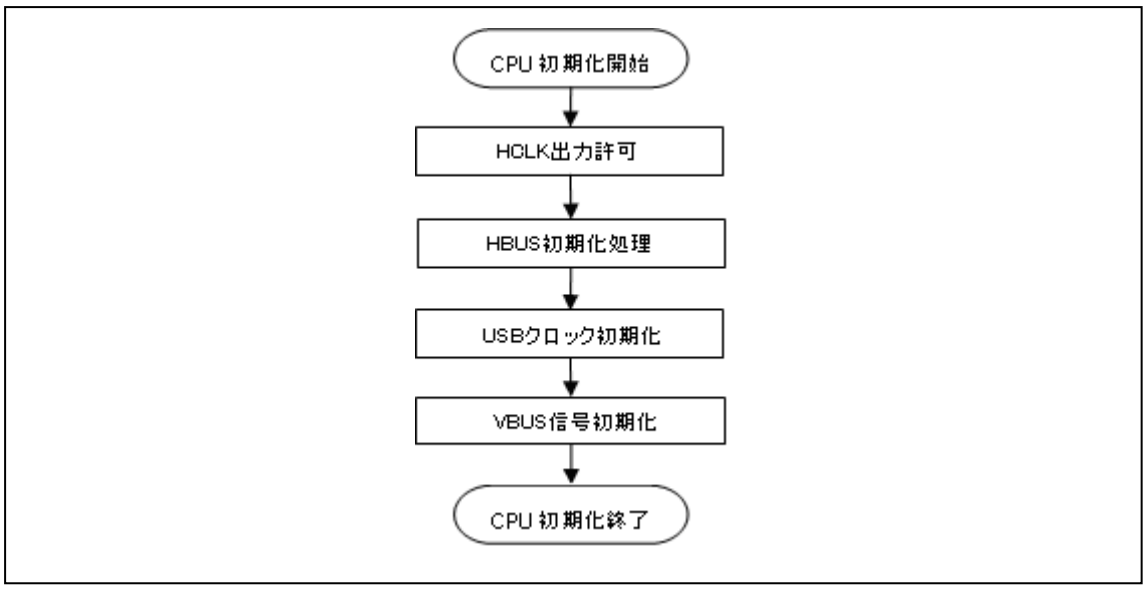


図 4－9　CPU 初期化の処理フロー

(1) HCLK 出力許可設定

HCLK の出力を許可し, H バスに接続される USBF が動作できるように設定します。設定を行う SFRCTL2 レジスタは特定書き込みレジスタとなっているため, 特定書き込みシーケンスを行って設定を行います。

(2) H バス初期化処理

H バスの初期化処理を行います。指定の指示に従い, H バスの初期化を行います。詳細は V850E2/MN4マイクロコントローラのユーザーズマニュアル　ハードウエア編を参照してください。

(3) USB クロックの初期化

UCLK が接続されている兼用端子 P13 の設定を行います。本サンプルでは USB クロックとして UCLK を使用しており，これにより USB クロック入力が行われます。

(4) VBUS 信号初期化

VBUS 信号の初期化設定を行います。

4.2.2　USBファンクション・コントローラ初期化処理

USB ファンクション・コントローラの使用を開始するために必要な項目を設定します。

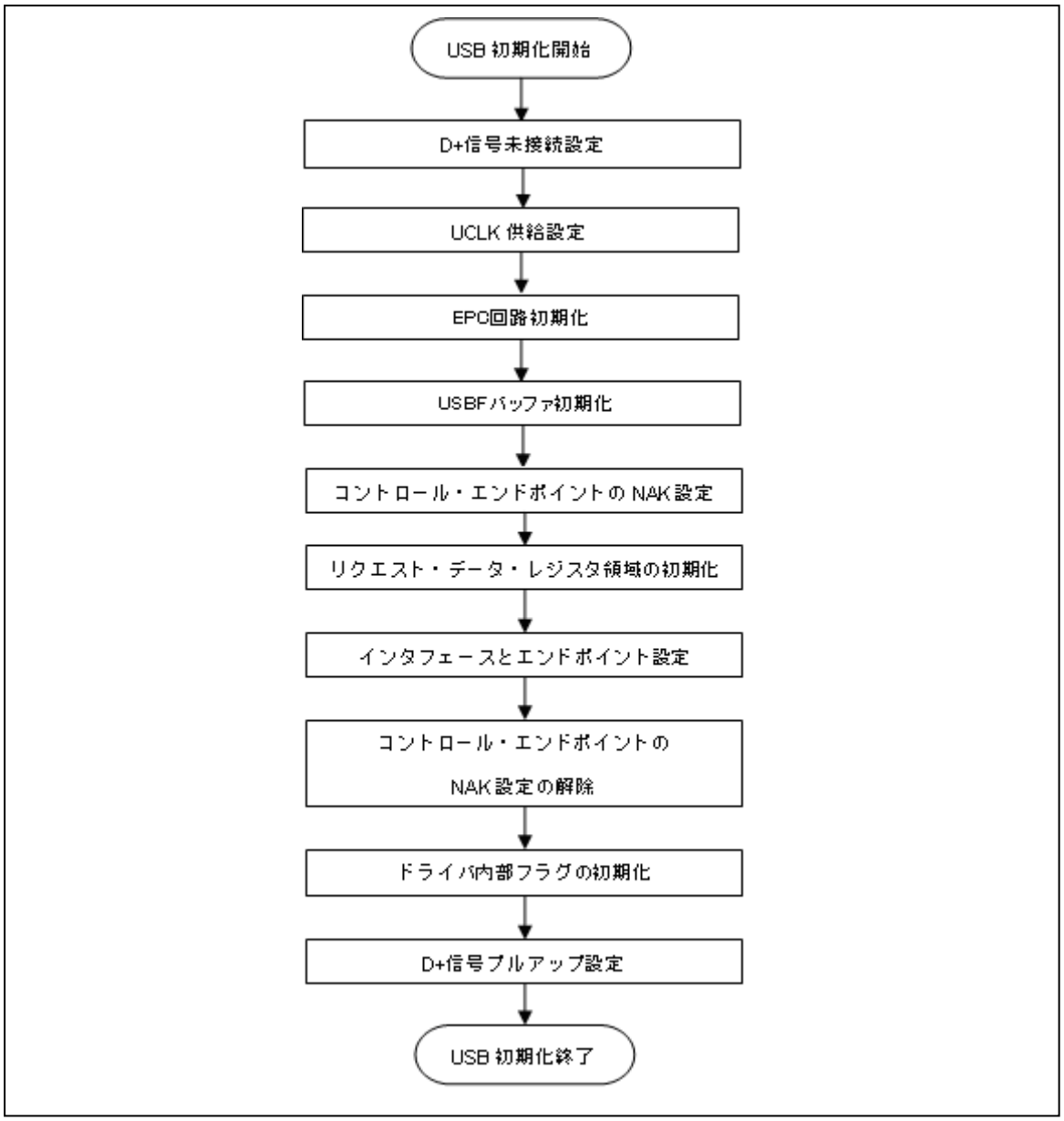


図 4－10　USB ファンクション・コントローラ初期化の処理フロー

(1) D+信号プルダウン設定

CPU の P4.10 に”0”を設定します。これにより，D+信号をロウ･レベルの出力にして，ホスト側にデバイスの接続を検知されない様にします。

(2) UCLK 供給設定

SFRCTL3 レジスタに”0x48”を設定し，USB ファンクションへのクロック供給を許可します。

(3) EPC 回路初期化

USFA0EPCCTL レジスタに”0x00000000”を設定し，EPC リセット信号を解除します。

(4) USB ファンクション・バッファ初期化

USFBC レジスタに”0x00000003”を設定し，USBF バッファ有効及びフローティング対策有効設定を行います。

(5) コントロール・エンドポイントの NAK 設定

ここでは USFA0E0NA レジスタの EP0NKA ビットに "1" を書き込みます。この設定により，自動応答リクエストを含むすべてのリクエストに対してハードウエアが NAK で応答します。
このビットは，自動応答リクエストで使用するデータの登録が完了するまで，ハードウエアが自動応答リクエストに対して意図しないデータを返さないようにするために，ソフトウエアが使用します。

(6) リクエスト・データ・レジスタ領域の初期化

GET_DESCRIPTOR リクエストに自動応答するためのディスクリプタ・データなどを各種レジスタに登録します。
ここでは次に示すレジスタにアクセスします。

(a) USFA0DSTL レジスタに "0x01" を書き込みます。この設定により，リモート・ウエイクアップ機能の使用が禁止され，USB ファンクション・コントローラはセルフ・パワード・デバイスとして動作します。
(b) USFA0EnSL レジスタ（n = 0-2）に "0x00" を書き込みます。この設定により，Endpoint n が正常に動作していることを示します。
(c) USFA0DSCL レジスタに，必要なディスクリプタのデータ長の合計（バイト数）を書き込みます。この設定により，使用される USFA0CIEn レジスタ（n = 0-255）の範囲が決まります。
(d) USFA0DDn レジスタ（n = 0-7）にデバイス・ディスクリプタのデータを書き込みます。
(e) USFA0CIEn レジスタ(n = 0-255)にコンフィギュレーション･ディスクリプタ, インタフェース・ディスクリプタ，およびエンドポイント・ディスクリプタのデータを書き込みます。
(f) USFA0MODC レジスタに "0x00" を書き込みます。この設定により，
GET_DESCRIPTOR_configuration リクエストへの自動応答が許可されます。

(7) インタフェースとエンドポイントの設定

サポートするインタフェースの数，オルタナティブ設定の状態，インタフェースとエンドポイントの関係などの情報を各種レジスタに設定します。
ここでは次に示すレジスタにアクセスします。

(a) USFA0AIFN レジスタに "0x00" を書き込みます。この設定により，インタフェース 0 だけを有効にします。
(b) USFA0AAS レジスタに "0x00" を書き込みます。この設定により，オルタナティブ設定を無効にします。
(c) USFA0E1IM レジスタに "0x20" を書き込みます。この設定により，Endpoint1 が Interface0 にリンクされます。
(d) USFA0E2IM レジスタに "0x20" を書き込みます。この設定により，Endopoint2 が Intreface0 にリンクされます。

(8) コントロール・エンドポイントの NAK 設定の解除

ここでは USFA0E0NA レジスタの EP0NKA ビットに "0" を書き込みます。この設定により，自動応答リクエストを含むすべてのリクエストに対して，それぞれに応じた応答が再開されます。

(9) 割り込みマスク・レジスタの設定

USB ファンクション・コントローラの割り込み要因ごとのマスクを設定します。
ここでは次に示すレジスタにアクセスします。

(a) USFA0ICn レジスタ（n = 0-4）に "0x00" を書き込みます。この設定により，すべての割り込み要因がクリアされます。
(b) USFA0FIC0 レジスタに "0xF7" を，USFA0FIC1 レジスタに”0x0F”を書き込みます。この設定により，すべての転送用 FIFO がクリアされます。
(c) USFA0IM0 レジスタに "0x1B" を書き込みます。この設定により，USFA0IS0 レジスタに示される割り込み要因のうち，BUSRST 割り込み，RSUSPD 割り込み，SETRQ 割り込み以外の要因がすべてマスクされます。
(d) USFA0IM1 レジスタに "0x7E" を書き込みます。この設定により，USFA0IS1 レジスタに示される割り込み要因のうち，CPUDEC 割り込み以外の要因がすべてマスクされます。
(e) USFA0IM2 レジスタに "0xF1" を書き込みます。この設定により，USFA0IS2 レジスタに示される割り込み要因がすべてマスクされます。
(f) USFA0IM3 レジスタに "0xFE" を書き込みます。この設定により，USFA0IS3 レジスタに示される割り込み要因のうち，BKO1DT 割り込み以外の要因がすべてマスクされます。
(g) USFA0IM4 レジスタに "0x20" を書き込みます。この設定により，USFA0IS4 レジスタに示される割り込み要因がすべてマスクされます。
(i) USFA0EPCINTE レジスタに”0x0003”を書き込み，EPC_INT0BEN，EPC_INT1BEN ビットが立った時の割り込みを有効にします。
(j) ICUSFA0I1 に”0”を，ICUSFA0I2 に”0”を書き込み，INTUSFA0I1・INTUSFA0I2 を有効にします。

(10) ドライバ内部フラグの初期化

ドライバ内部で使用するフラグ(usbf850_busrst_flg，usbf850_rsuspd_flg，usbf850_rdata_flg)の初期化を行います。

(11) D+信号プルアップ設定

CPU の P4 レジスタに "0x0400" を書き込みます。この設定により，P4_10 から "1" が出力されます。これにより，D+信号からハイ・レベルを出力して，ホスト側にデバイスが接続されたことを通知します。サンプル・ドライバでは図 4−11 に示すような接続を想定しています。

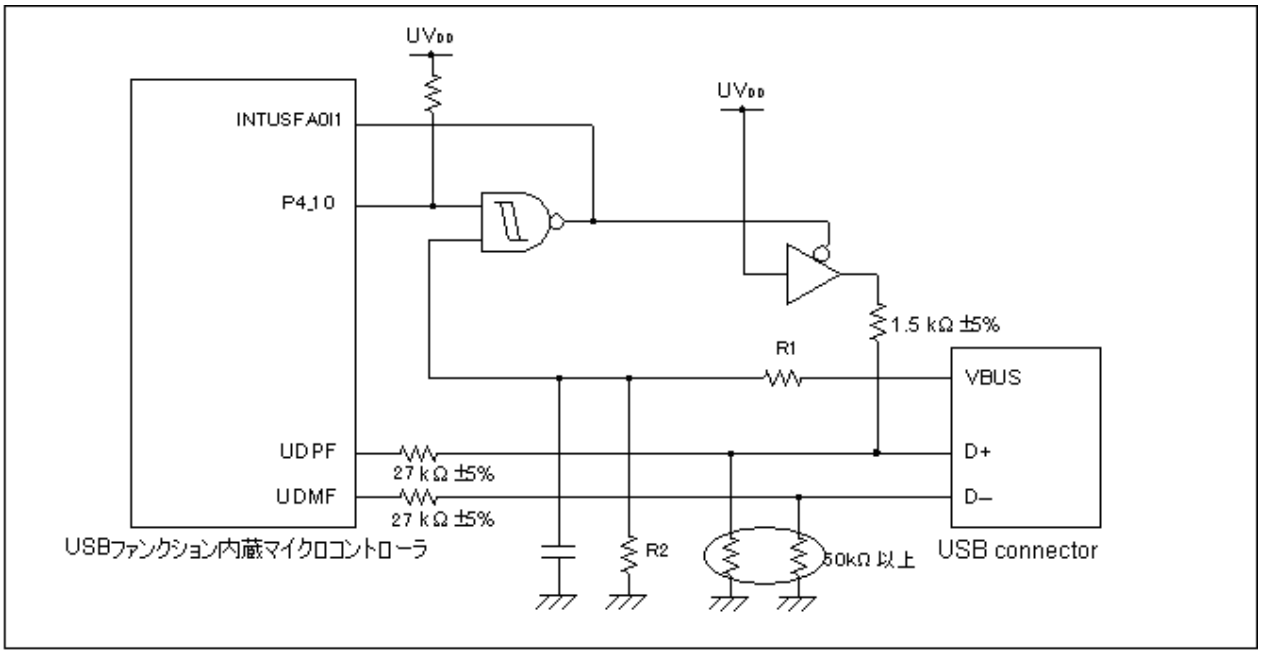


図 4−11　USB ファンクション・コントローラ接続例

4.2.3　USBF割り込み処理（INTUSFA0I1）

INTUSFA0I1 割り込みハンドラでは，コントロール転送用のエンドポイント（Endpoint0）およびバルク・アウト転送（受信）用のエンドポイント（Endpoint2）の状態を監視し，受信したリクエストおよびデータに対応する処理を行います。

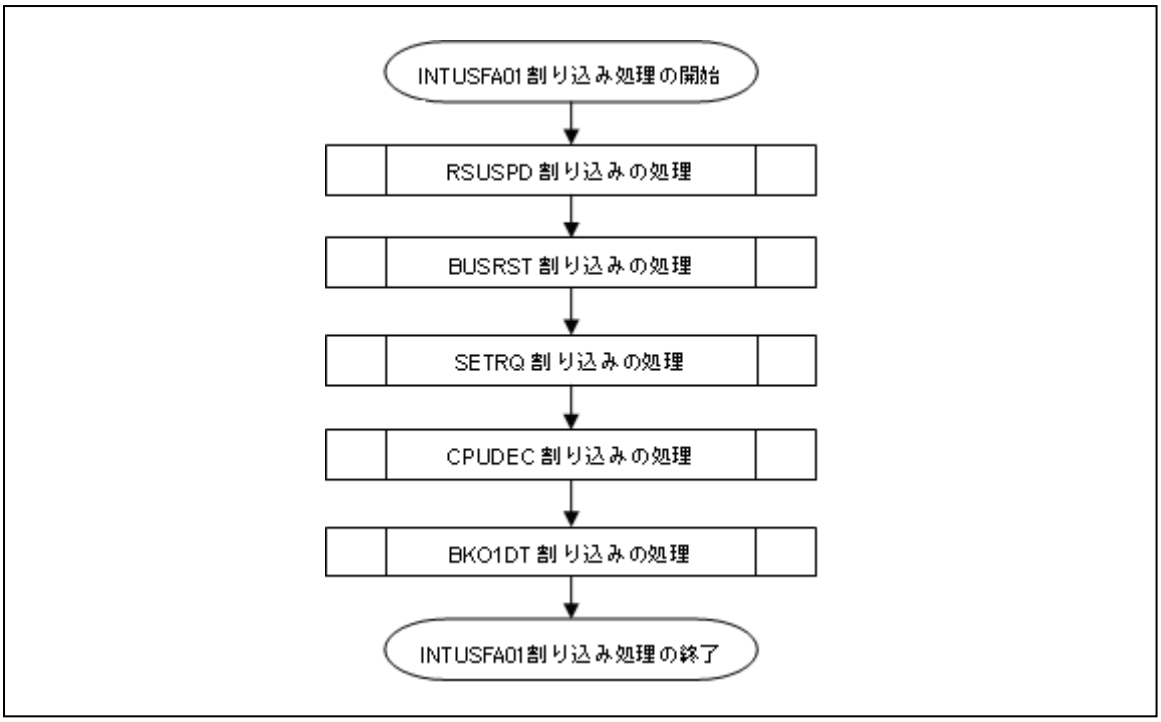


図 4－12　INTUSFA0I1 割り込みハンドラ処理フロー

（1）RSUSPD 割り込みの処理

USFA0IS0 レジスタの RSUSPD ビットが "1" のとき, RSUSPD 割り込みが発生していると判断します。

RSUSPD 割り込みが発生している場合，次の処理を行います。

- 割り込み要因をクリア（USFA0IC0 レジスタの RSUSPDC ビットに "0" を書き込む）
- Suspend/Resume 状態の判定

（2）Suspend 時の処理

USFA0EPS1 レジスタの RSUM ビットが "1" のとき，Suspend 状態にあると判断します。

Suspend 状態ですでにレジューム／サスペンド・フラグ（rs_flag）が "SUSPEND（0x00）" の場合は，以降の処理を行わずに INTUSFA0I1 割り込み処理を終了します。

レジューム／サスペンド・フラグ（rs_flag）が "SUSPEND" でなければ，"SUSPEND" に設定し，USB の割り込み要因をすべてクリアします。これにより，以降の INTUSFA0I1 割り込み処理は省略されます。

(3) BUSRST 割り込みの処理

USFA0IS0 レジスタの BUSRST ビットが "1" のとき, BUSRST 割り込みが発生していると判断します。
BUSRST 割り込みが発生している場合，次の処理を行います。

- 割り込み要因をクリア（USFA0IC0 レジスタの BUSRST ビットに "0" を書き込む）
- BUS Reset 割り込みフラグ（usbf_busrst_flg）に "1" を設定
- バルク・エンドポイント用 FIFO クリア

(4) SETRQ 割り込みの処理

USFA0IS0 レジスタの SETRQ ビットが "1" のとき，割り込みが発生していると判断します。
SETRQ 割り込みが発生している場合，次の処理を行います。

- 割り込み要因をクリア（USFA0IC0 レジスタの SETRQ ビットに "0" を書き込む）
- 自動応答リクエスト（SET_XXXX）の処理

(5) 自動応答リクエスト（SET_XXXX）の処理

UF0SET レジスタの SETCON ビットが "1" のとき，SET_CONFIGURATION リクエストを受信し，自動処理を行った状態にあることを判断します。
自動処理を行った場合，BUS Reset 割り込みフラグ（usbf_busrst_flg）を "0" にします。

（備考）厳密に Configured ステートに入ったことを確認する場合は，UF0CNF レジスタの値を確認してください。

(6) CPUDEC 割り込みの処理

USFA0IS1 レジスタの CPUDEC ビットが "1" のとき，割り込みが発生していると判断します。
CPUDEC 割り込みが発生している場合，次の処理を行います。

- ポート割り込み要因クリア（USFA0IC1 レジスタの PORT ビットに "0" を書き込む）
- 受信データを FIFO から読み込み，リクエスト・データを構成
- リクエスト処理

(7) リクエスト処理

リクエスト・データが，ハードウエアで自動応答しないリクエスト（標準，クラス，ベンダ）かどうかを判断し，リクエスト・タイプに応じて，各リクエストの処理を実行します。
エンドポイント 0 は, コントロール転送用のエンドポイントです。プラグイン時のエニュメレーション処理では，ほとんどの標準デバイス・リクエストがハードウエアによって自動処理されます。ここでは，自動処理対象外の標準リクエスト，クラス・リクエストおよびベンダ・リクエストについて処理します。

(8) BKO1DT 割り込みの処理

USFA0IS3 レジスタの BKODT ビットが "1" のとき，割り込みが発生していると判断します。
BKODT 割り込みが発生している場合，次の処理を行います。

- BKODT 割り込み要因クリア（USFA0IC3 レジスタの BKO1DT ビットに "0" を書き込む）
- CBW データ受信処理関数（usbf850_rx_cbw）を呼び出し，CBW データ受信処理を行います。

4.2.4　USBFレジューム割り込み処理（INTUSFA0I2）

INTUSFA0I2 割り込みハンドラでは，レジューム割り込み発生時の処理を行います。
この処理では，レジューム／サスペンド・フラグ（rs_flag）を "RESUME（0x01）" に設定します。
rs_flag が "RESUME" のときの処理は，メイン・ルーチンで行います。

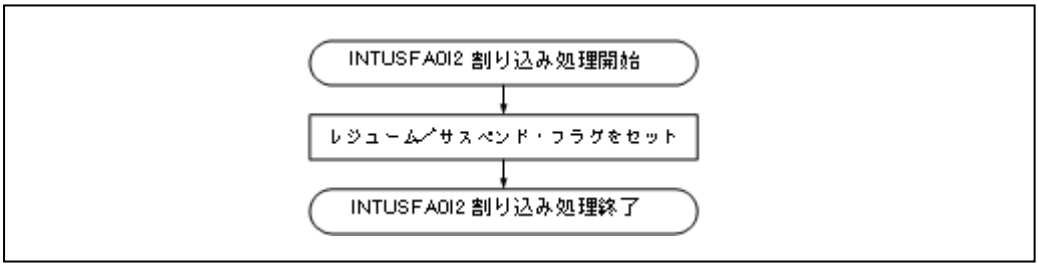


図 4－13　INTUSFA0I2 割り込みハンドラ処理フロー

4.2.5　CBWデータ受信処理

CBW データ受信処理では，バルク・アウト・エンドポイント（Endpoint2）の FIFO からデータを読み出し，CBW データのコマンド解析処理を呼び出します。

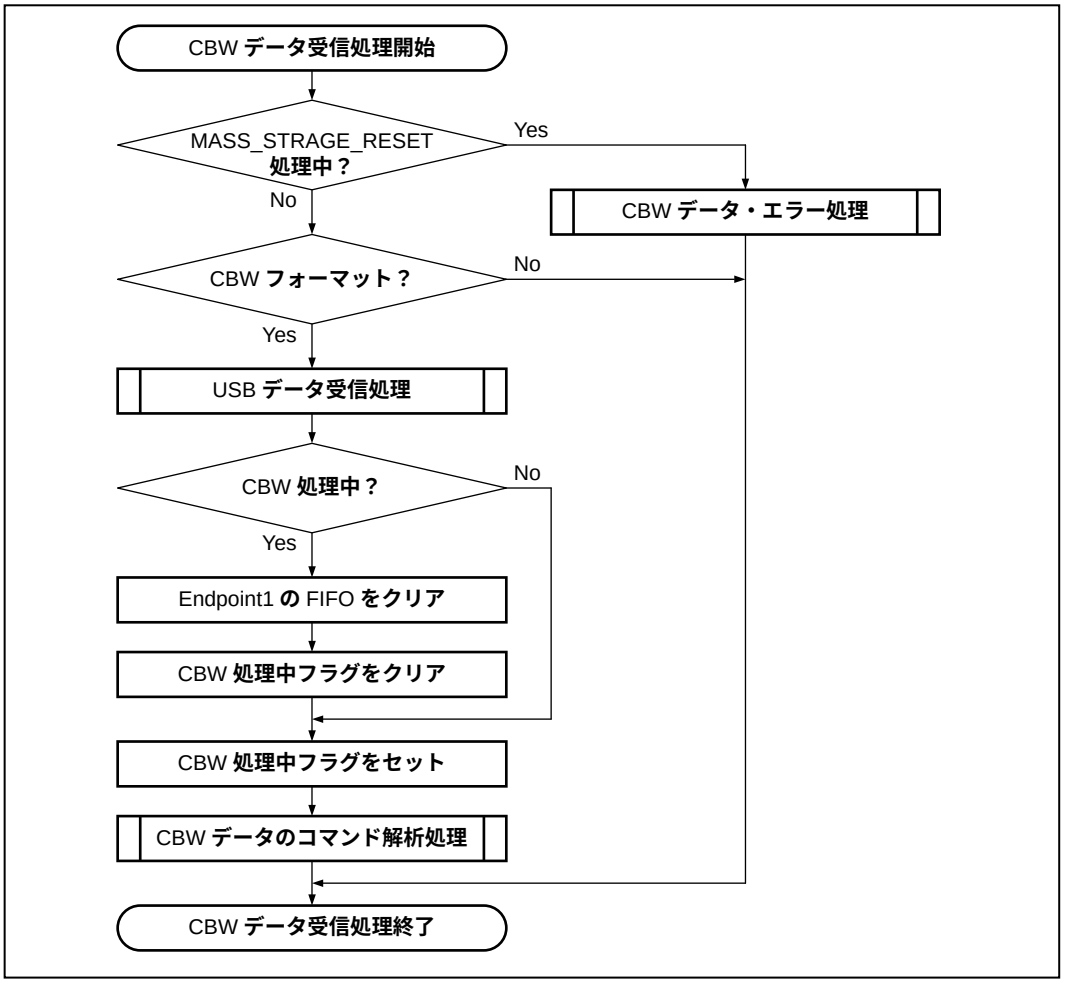


図 4－14　CBW データ受信処理フロー

(1) MASS_STRAGE_RESET 処理中の判定

MASS_STRAGE_RESET 処理フラグ（mass_storage_reset）が "1" の場合，処理中と判断します。
処理中の場合，CBW データのエラー処理関数（usbf850_cbw_error）を呼び出し，CBW データ受信処理を終了します。

(2) CBW フォーマットの判定

UF0 バルク・アウト 1 レングス・レジスタ（USFA0BO1L）からバルク・アウト・エンドポイント（Endpoint2）に格納されているデータの大きさ（データ長）を取得します。データ長が 31 バイトの場合，CBW フォーマットと合致していると判断します。
CBW フォーマットではない場合，CBW データ受信処理を終了します。
CBW フォーマットの場合，USB データ受信処理関数（usbf850_data_receive）を呼び出し，処理を続けます。

(3) CBW 処理中の判定

CBW 処理中フラグ（cbw_in_cbw）が "USB_CBW_PROCESS (0x01)" の場合, 処理中と判断します。
処理中の場合, Endpoint1 の FIFO をクリアし, CBW 処理中フラグ（cbw_in_cbw）を "USB_CBW_END (0x00)" に設定します。

(4) CBW 処理中フラグのセット

CBW 処理中フラグ（cbw_in_cbw）を "USB_CBW_PROCESS (0x01)" に設定します。

(5) CBW コマンド解析処理

CBW コマンド解析処理関数（usbf850_storage_cbwchk）を呼び出し，受信した SCSI コマンドに対する処理を行います。

4.2.6　SCSIコマンド処理

USB で CBW データを受信すると, CBW コマンド解析処理関数（usbf850_storage_cbwchk）を呼び出し, 受信した SCSI コマンドに対する処理を行います。

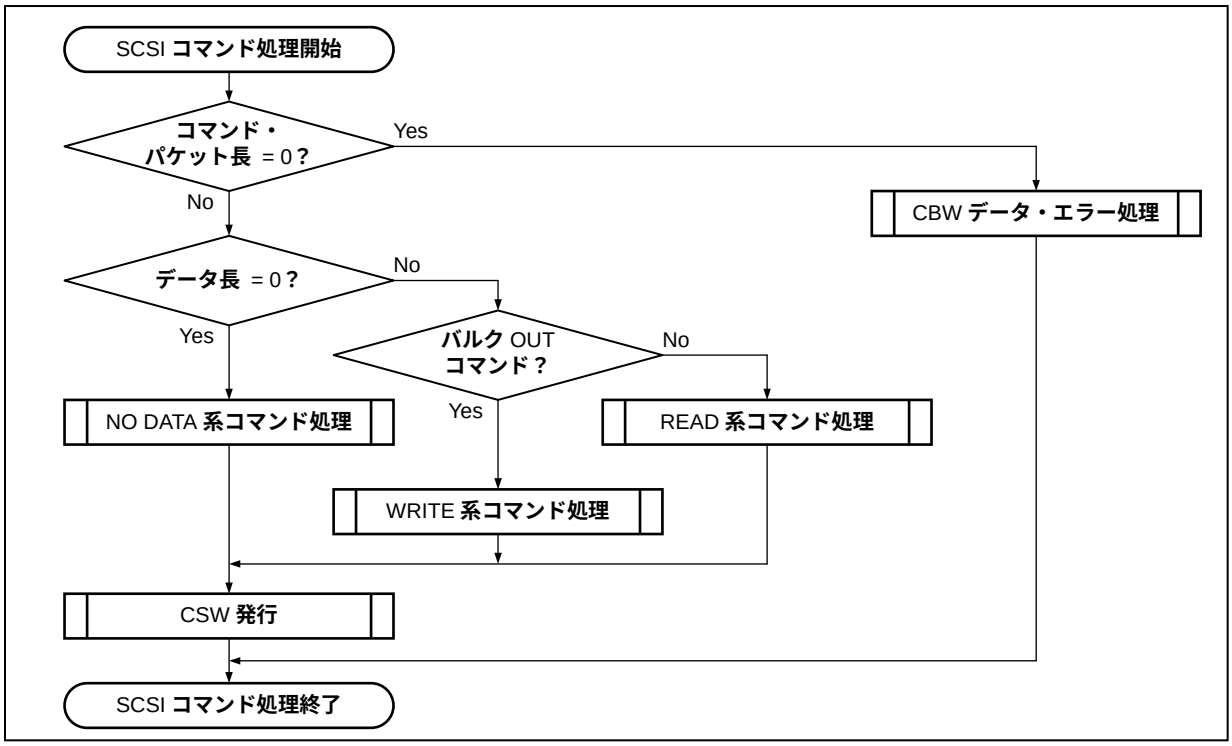


図 4−15　SCSI コマンド処理フロー

（1）SCSI コマンド判断

コマンド・パケット長（bCBWCBLength）が "0x00" の場合, SCSI コマンドではないと判断します。SCSI コマンドでない場合，CBW データのエラー処理関数（usbf850_cbw_error）を呼び出し，SCSI コマンド処理を終了します。

（2）NO DATA 系コマンド判断

データ・フェーズで転送するデータ長（dCBWDataTransferLength）が "0x00000000" の場合，NO DATA 系コマンドと判断します。

NO DATA 系コマンドの場合，NO DATA 系コマンド処理関数（usbf850_no_data）を呼び出し，受信したコマンドと対応する処理を実行します。

コマンド処理が完了すると，CSW 応答処理関数（usbf850_csw_ret）を呼び出し，CSW を送信します。

（3）データ転送方向判断

転送方向（bmCBWFlags）のビット 7 が "0" の場合，WRITE 系コマンドと判断し，DATA OUT 系コマンド処理関数（usbf850_data_out）を呼び出し, 受信したコマンドと対応する処理を実行します。

bmCBWFlags のビット 7 が "1" の場合，READ 系コマンドと判断し，DATA IN 系コマンド処理関数（usbf850_data_in）を呼び出し，受信したコマンドと対応する処理を実行します。

コマンド処理が完了すると，CSW 応答処理関数（usbf850_csw_ret）を呼び出し，CSW を送信します。

4.2.7　サスペンド／レジューム処理

メイン・ルーチン内では，次のフローでサスペンド／レジューム処理を行います。

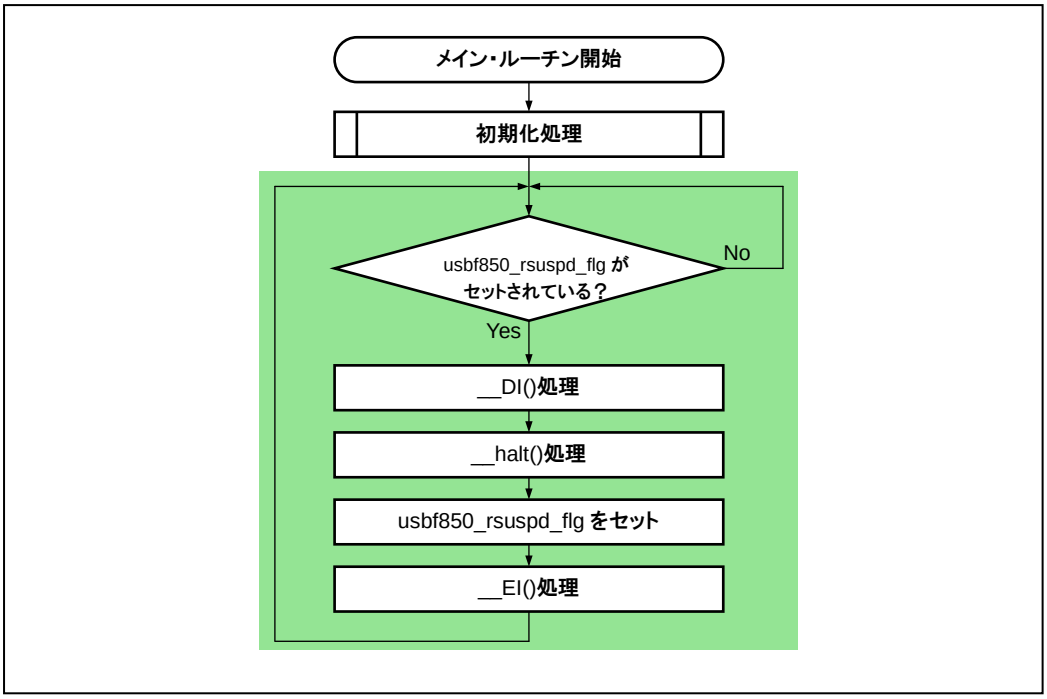


図 4－16　サスペンド／レジューム処理フロー

(1) レジューム／サスペンド・フラグ（usbf850_rsuspd_flg）の監視

サンプル・ドライバにより設定されるレジューム／サスペンド・フラグ（usbf850_rsuspd_flg）を監視します。このフラグが "SUSPEND（0x00）" の場合，USB バスがサスペンド状態になったことを示します。

(2) CPU 割り込み禁止

レジューム／サスペンド・フラグ（usbf850_rsuspd_flg）が"SUSPEND（0x00）"だった場合，CPU 割り込みを禁止します。

(3) CPU HALT 処理

プロセッサを停止し，HALT 状態に遷移します。HALT 状態からの処理再開は，マスカブル割り込み，NMI，リセットにより行われます。本サンプルでは，INTUSFA0I2 のレジューム割り込みにより処理再開します。

(4) レジューム／サスペンド・フラグ（usbf850_rsuspd_flg）の更新

レジューム／サスペンド・フラグ（usbf850_rsuspd_flg）を”RESUME(0x01)”に設定します。

(5) CPU 割り込み許可

CPU 割り込みを許可します。これにより，レジューム処理が完了します。

4.3　関数の仕様

ここでは，サンプル・ドライバに実装されている各種関数について説明します。

4.3.1　関数一覧

サンプル・ドライバでは，ソース・ファイルそれぞれに次のような関数が実装されています。

表 4-40　サンプル・ドライバ内の関数（1/2）

ソース・ファイル	関数名	説　明
main.c	main	メイン・ルーチン
	cpu_init	CPU の初期化
	SetProtectReg	書き込み保護レジスタアクセス処理
usbf850.c	usbf850_init	USBファンクション・コントローラの初期化
	usbf850_intusbf0	Endpoint0 の監視とリクエストへの応答制御
	usbf850_intusbf1	レジューム割り込み処理
	usbf850_data_send	USB データの送信
	usbf850_data_receive	USB データの受信
	usbf850_rdata_length	USB 受信データ長の取得
	usbf850_send_EP0	Endpoint0 の送信
	usbf850_receive_EP0	Endpoint0 の受信
	usbf850_send_null	Bulk/ Interrupt In Endpoint への Null パケット送信処理
	usbf850_sendnullEP0	Endpoint0 用 NULL パケットの送信
	usbf850_sendstallEP0	Endpoint0 用 STALL 応答
	usbf850_ep_status	Bulk/ Interrupt In Endpoint の FIFO 状態通知処理
	usbf850_fifo_clear	Endpoint0 以外の Endpoint の FIFO クリア
	usbf850_standardreq	標準リクエストの処理
	usbf850_getdesc	GET_DESCRIPTOR リクエストの処理
usbf850_storage.c	usbf850_classreq	MSC クラス・リクエストの処理
	usbf850_blkonly_mass_storage_reset	Mass Strage Reset リクエストの処理
	usbf850_max_lun	Get Max Len リクエストの処理
	usbf850_rx_cbw	CBW データの受信
	usbf850_storage_cbwchk	CBW データのコマンドの解析
	usbf850_cbw_error	CBW データのエラーの処理
	usbf850_no_data	SCSI の NO DATA 系コマンドの処理
	usbf850_data_in	SCSI の WRITE 系コマンドの処理
	usbf850_data_out	SCSI の READ 系コマンドの処理
	usbf850_csw_ret	CSW 応答の処理
	usbf850_bulkin_stall	バルク・イン用 STALL 応答の制御
	usbf850_bulkout_stall	バルク・アウト用 STALL 応答の制御

表 4-41　サンプル・ドライバ内の関数（2/2）

ソース・ファイル	関数名	説　　明
scsi_cmd.c	scsi_command_to_ata	SCSI コマンドの実行
	ata_test_unit_ready	TEST UNIT READY コマンドの処理
	ata_seek	SEEK コマンドの処理
	ata_start_stop_unit	START STOP UNIT コマンドの処理
	ata_synchronize_cache	SYNCHRONIZE CACHE コマンドの処理
	ata_request_sense	REQUEST SENSE コマンドの処理
	ata_inquiry	INQUIRY コマンドの処理
	ata_mode_select	MODE SELECT(6)コマンドの処理
	ata_mode_select10	MODE SELECT(10)コマンドの処理
	ata_mode_sense	MODE SENSE(6)コマンドの処理
	ata_mode_sense10	MODE SENSE(10)コマンドの処理
	ata_read_format_capacities	READ FORMAT CAPACITIES コマンドの処理
	ata_read_capacity	READ CAPACITY コマンドの処理
	ata_read6	READ(6)コマンドの処理
	ata_read10	READ(10)コマンドの処理
	ata_write6	WRITE(6)コマンドの処理
	ata_write10	WRITE(10)コマンドの処理
	ata_verify	VERIFY コマンドの処理
	ata_write_verify	WRITE VERIFY コマンドの処理
	ata_write_buff	WRITE BUFFER コマンドの処理
	scsi_to_usb	USB データ送信処理（SCSI コマンド系）

4.3.2　関数の相関関係

関数によっては，処理の中で別の関数を呼び出しているものもあります。関数の呼び出し関係を次に示します。

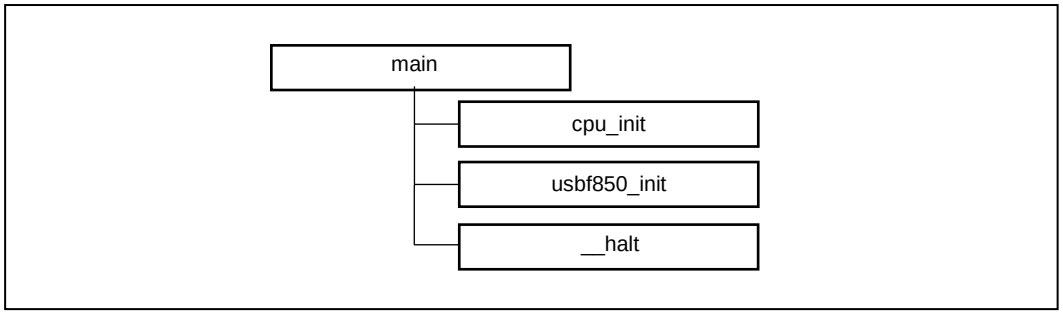


図 4－17　main 処理での関数の呼び出し

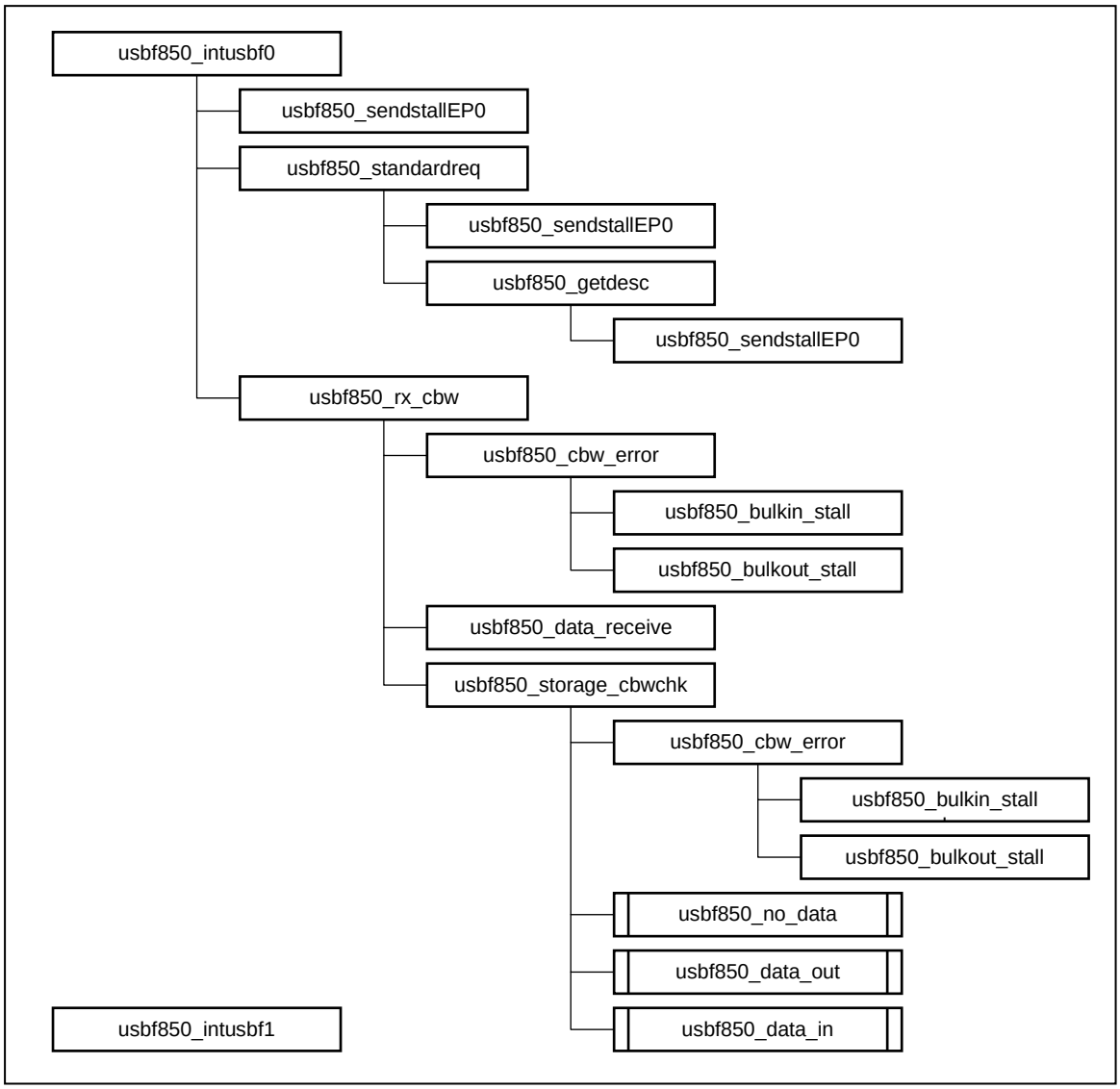


図 4－18　USB 割り込み処理での関数の呼び出し

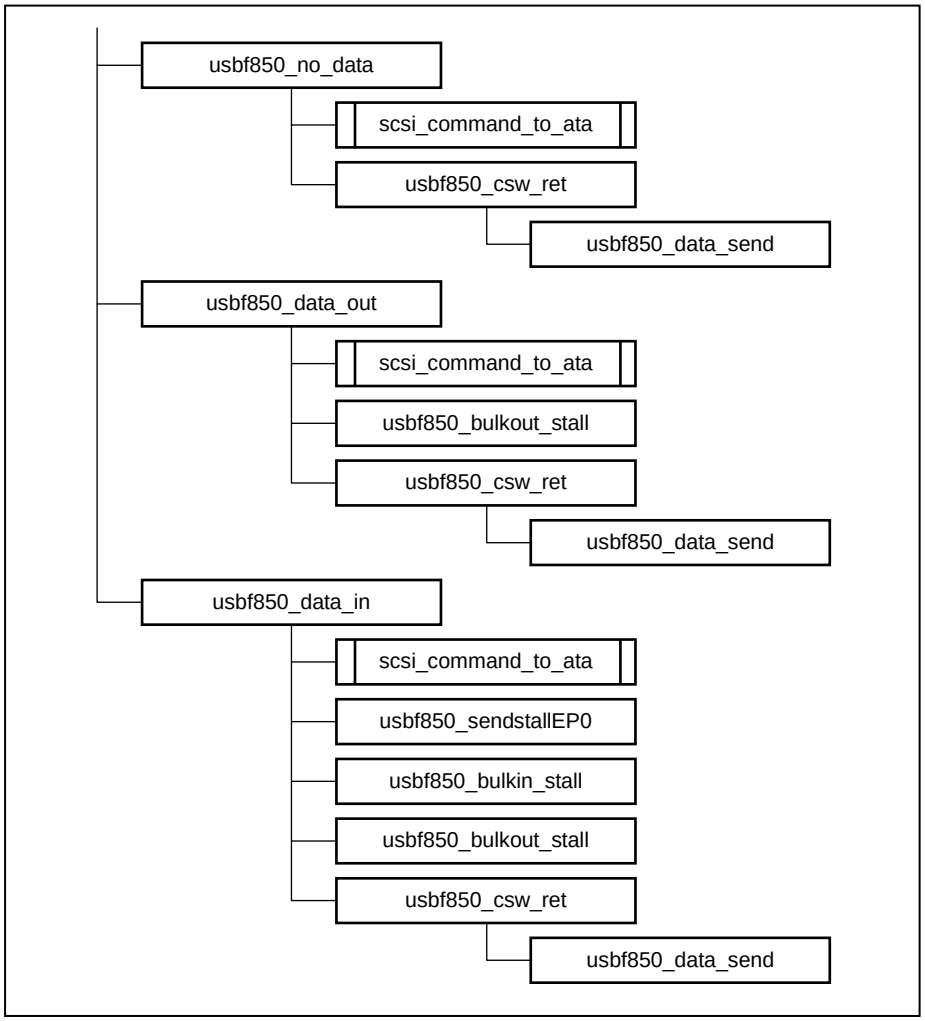


図 4－19　CBW/CSW 処理での関数の呼び出し

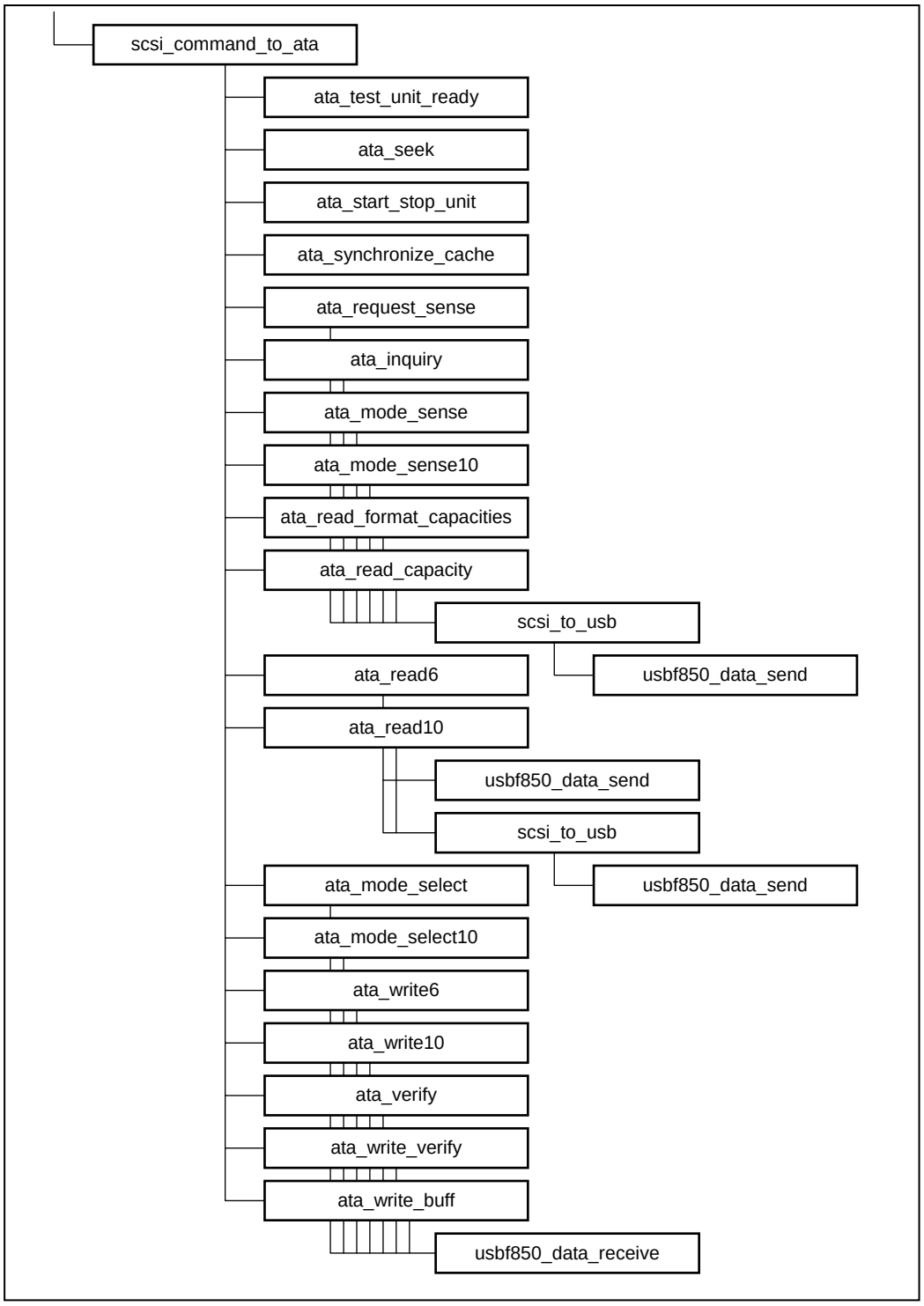


図 4－20　SCSI コマンド処理での関数の呼び出し

4.3.3　関数の機能

ここでは，サンプル・ドライバに実装されている各種関数について解説します。

(1) 関数解説フォーマット

解説は，関数ごとに次の形式で記述されます。

関数名称

【 概　要 】

概要説明

【C 言語記述形式】

C 言語上の記述形式

【パラメータ】

パラメータ（引数）の説明

パラメータ	説　　明
パラメータ型，名称	*パラメータ概要説明*

【 戻り値 】

戻り値の説明

シンボル	説　　明
戻り値型，名称	*戻り値概要説明*

【 機　能 】

機能説明

(2) メイン・ルーチンの関数

main

【 概　要 】

メイン処理

【C 言語記述形式】

void main(void)

【パラメータ】

なし

【 戻り値 】

なし

【 機　能 】

サンプル・ドライバを実行すると最初に呼び出される関数です。

USB 初期化処理関数（usbf850_init）を呼び出したあと，レジューム／サスペンド・フラグ（usbf850_rsuspd_flg）を監視します。usbf850_rsuspd_flg が "SUSPEND（0x00）" になるとサスペンド処理を行います。

cpu_init

【 概　要 】

CPU 初期化処理

【C 言語記述形式】

void cpu_init(void)

【パラメータ】

なし

【 戻り値 】

なし

【 機　能 】

初期化処理で呼び出される関数です。
H バス初期化や USB クロックなど，USB ファンクション・コントローラを使用するために必要な項目等を設定します。

SetProtectReg

【 概　要 】

書き込み保護レジスタへのアクセス

【C 言語記述形式】

void SetProtectReg(volatile UINT32 *dest_reg, UINT32 wr_dt, volatile UINT8 *prot_reg)

【パラメータ】

パラメータ	説　　明
volatile UINT32 *dest_reg	保護されたレジスタアドレス
UINT32 wr_dt	書き込み数値
volatile UINT8 *prot_reg	保護コマンドレジスタアドレス

【 戻り値 】

なし

【 機　能 】

書き込み保護されたレジスタへの書き込み処理を行います。

(3) USB ファンクション・コントローラ用処理の関数

usbf850_init

【 概　要 】

USB ファンクション・コントローラ初期化処理

【C 言語記述形式】

void usbf850_init(void)

【パラメータ】

なし

【 戻り値 】

なし

【 機　能 】

初期化処理で呼び出される関数です。
データ領域の確保と設定，割り込み要求のマスクなど，USB ファンクション・コントローラの使用を開始するために必要な項目を設定します。

usbf850_intusbf0

【 概　要 】
INTUSFA0I1 割り込みハンドラ処理

【C 言語記述形式】
void usbf850_intusbf0(void)

【パラメータ】
なし

【 戻り値 】
なし

【 機　能 】
USB 割り込み（INTUSFA0I1）のハンドラとして呼び出されます。
コントロール転送用のエンドポイント（Endpoint0）およびバルク・アウト転送（受信）用のエンドポイント（Endpoint2）の状態を監視し，受信したリクエストおよびコマンドに対応する処理を行います。
Endpoint0 では，RSUSPD，BUSRST，SETRQ，CPUDEC 割り込みを監視します。CPUDEC 割り込み発生時は，リクエスト・データをデコードし，該当する関数を呼び出して応答処理を行います。
Endpoint2 では，BKO1DT 割り込みを監視します。BKO1DT 割り込み発生時は，CBW データの受信関数（usbf850_rx_cbw）を呼び出し，コマンドに対応する処理を行います。

usbf850_intusbf1

【 概　要 】
INTUSFA0I2 割り込みハンドラ処理

【C 言語記述形式】
void usbf850_intusbf1(void)

【パラメータ】
なし

【 戻り値 】
なし

【 機　能 】
USB レジューム割り込み（INTUSFA0I2）のハンドラとして呼び出されます。
レジューム／サスペンド・フラグ（usbf850_rsuspd_flg）を "RESUME（0x01）" に設定します。

usbf850_data_send

【 概　要 】

USB データ送信処理

【C 言語記述形式】

INT32 usbf850_data_send(UINT8 *data, INT32 len, INT8 ep)

【パラメータ】

パラメータ	説　　明
UINT8 *data	送信データ・バッファ・ポインタ
INT32 len	送信データ長
INT8 ep	データ送信エンドポイント番号

【 戻り値 】

シンボル	説　　明
DEV_OK	正常終了
DEV_ERROR	異常終了

【 機　能 】

送信データ・バッファに格納されているデータを，指定されたエンドポイント用の FIFO に 1 バイトずつ格納します。

usbf850_data_receive

【 概　要 】

USB データ受信処理

【C 言語記述形式】

INT32 usbf850_data_receive(UINT8 *data, INT32 len, INT8 ep)

【パラメータ】

パラメータ	説　　明
UINT8 *data	受信データ・バッファ・ポインタ
INT32 len	受信データ長
INT8 ep	データ受信エンドポイント番号

【 戻り値 】

シンボル	説　　明
DEV_OK	正常終了
DEV_ERROR	異常終了

【 機　能 】

指定されたエンドポイント用の FIFO からデータを 1 バイトずつ読み出し，受信データ・バッファに格納します。

usbf850_rdata_length

【 概　要 】
USB データ受信データ長取得

【C 言語記述形式】
void usbf850_rdata_length(INT32 *len , INT8 ep)

【パラメータ】

パラメータ	説　　明
INT32* len	受信データ長格納アドレス・ポインタ
INT8 ep	データ受信エンドポイント番号

【 戻り値 】
なし

【 機　能 】
指定されたエンドポイントの受信データ長を読み出します。

usbf850_send_EP0

【 概　要 】

Endpoint0 用 USB データ送信処理

【C 言語記述形式】

INT32 usbf850_send_EP0(UINT8* data, INT32 len)

【パラメータ】

パラメータ	説　　明
UINT* data	送信データ・バッファ・ポインタ
INT32 len	送信データ・サイズ

【 戻り値 】

シンボル	説　　明
DEV_OK	正常終了
DEV_ERROR	異常終了

【 機　能 】

送信データ･バッファに格納されているデータを Endpoint0 用送信 FIFO に 1 バイトずつ格納します。

usbf850_receive_EP0

【 概　要 】

Endpoint0 用 USB データ受信処理

【C 言語記述形式】

INT32 usbf850_receive_EP0(UINT8* data, INT32 len)

【パラメータ】

パラメータ	説　　明
UINT* data	受信データ・バッファ・ポインタ
INT32 len	受信データ・サイズ

【 戻り値 】

シンボル	説　　明
DEV_OK	正常終了
DEV_ERROR	異常終了

【 機　能 】

Endpoint0 用受信 FIFO から 1 バイトずつ読み出し，受信データ・バッファに格納します。

usbf850_send_null

【 概　要 】

Bulk/ Interrupt In Endpoint 用 Null パケット送信処理

【C 言語記述形式】

INT32 usbf850_send_null(INT8 ep)

【パラメータ】

パラメータ	説　　明
INT8 ep	データ送信エンドポイント番号

【 戻り値 】

シンボル	説　　明
DEV_OK	正常終了
DEV_ERROR	異常終了

【 機　能 】

指定された Endpoint（送信用）の FIFO をクリアし，データ終了を示すビットをセット(1)する事で，USB ファンクション・コントローラから Null パケットを送信します。

usbf850_sendnullEP0

【 概　要 】
Endpoint0 用 NULL パケット送信処理

【C 言語記述形式】
void usbf850_sendnullEP0(void)

【パラメータ】
なし

【 戻り値 】
なし

【 機　能 】
Endpoint0 用の FIFO をクリアし，データ終了を示すビットをセット（1）することで，USB ファンクション・コントローラから NULL パケットを送信させます。

usbf850_sendstallEP0

【 概　要 】
Endpoint0 用 STALL 応答処理

【C 言語記述形式】
void usbf850_sendstallEP0(void)

【パラメータ】
なし

【 戻り値 】
なし

【 機　能 】
STALL ハンドシェーク使用を示すビットをセット（1）することで，USB ファンクション・コントローラから STALL 応答させます。

usbf850_ep_status

【 概　要 】

Bulk/ Interrupt In Endpoint 用 FIFO 状態通知処理

【C 言語記述形式】

INT32 usbf850_ep_status(INT8 ep)

【パラメータ】

パラメータ	説　　明
INT8 ep	データ送信Endpoint番号

【 戻り値 】

シンボル	説　　明
DEV_OK	正常終了
DEV_RESET	Bus Reset処理中
DEV_ERROR	異常終了

【 機　能 】

指定された Endpoint(送信用)の FIFO 状態を通知します。

usbf850_fifo_clear

【 概　要 】

Bulk/ Interrupt Endpoint 用 FIFO クリア処理

【C 言語記述形式】

void usbf850_fifo_clear(INT8 in_ep, INT8 out_ep)

【パラメータ】

パラメータ	説　　明
INT8 in_ep	データ送信Endpoint
INT8 out_ep	データ受信Endpoint

【 戻り値 】

なし

【 機　能 】

指定された Endpoint（Bulk/Interrupt）の FIFO をクリアし，データ受信フラグ(usbf850_rdata_flg)をクリアします。

usbf850_standardreq

【 概　要 】

USB ファンクション・コントローラが自動応答しない標準リクエストの処理

【C 言語記述形式】

void usbf850_standardreq(void)

【パラメータ】

なし

【 戻り値 】

なし

【 機　能 】

Endpoint0 監視処理から呼び出される関数です。

デコードされたリクエストが GET_DESCRIPTOR の場合，GET_DESCRIPTOR リクエスト処理関数（usbf850_getdesc）を呼び出します。それ以外のリクエストの場合は Endpoint0 用 STALL 応答処理関数（usbf850_sendstallEP0）を呼び出します。

usbf850_getdesc

【 概　要 】
GET_DESCRIPTOR リクエスト処理

【C 言語記述形式】
void usbf850_getdesc(void)

【パラメータ】
なし

【 戻り値 】
なし

【 機　能 】
USB ファンクション・コントローラが自動応答しない標準リクエストの処理で呼び出される関数です。
デコードされたリクエストがストリング・ディスクリプタを要求している場合，USB データ送信処理関数（usbf850_data_send）を呼び出して，Endpoint0 からストリング・ディスクリプタを送信させます。それ以外のディスクリプタを要求している場合は Endpoint0 用 STALL 応答処理関数（usbf850_sendstallEP0）を呼び出します。

(4) USB マス・ストレージ・クラス用処理の関数

usbf850_classreq

【 概　要 】

MSC のクラス・リクエスト処理

【C 言語記述形式】

void usbf850_classreq(USB_SETUP *req_data)

【パラメータ】

パラメータ	説　　明
USB_SETUP *req_data	リクエスト・データ格納ポインタ・アドレス

【 戻り値 】

なし

【 機　能 】

INTUSFA0I1 割り込み処理の CPUDEC 割り込み要因で呼び出される関数です。デコードされたリクエストがコミュニケーション･デバイス・クラス固有のリクエストの場合，各リクエスト処理関数を呼び出します。それ以外の場合は Endpoint0 に Stall を送信します。

usbf850_blkonly_mass_storage_reset

【 概　要 】
Mass Storage Reset 処理

【C 言語記述形式】
void usbf850_blkonly_mass_storage_reset(void)

【パラメータ】
なし

【 戻り値 】
なし

【 機　能 】
エンドポイント 1，エンドポイント 2 の FIFO クリアし，STALL 応答に設定します。
その後，エンドポイント 0 から NULL パケットを送信します。

usbf850_max_lun

【 概　要 】
Get Max Lun 処理

【C 言語記述形式】
void usbf850_max_lun(void)

【パラメータ】
なし

【 戻り値 】
なし

【 機　能 】
マス・ストレージ・デバイスの論理装置数（Logical Unit Number）を送信します。

usbf850_rx_cbw

【 概　要 】
CBW データの受信処理

【C 言語記述形式】
void usbf850_rx_cbw(void)

【パラメータ】
なし

【 戻り値 】
なし

【 機　能 】
バルク・イン用エンドポイント（Endpoint2）の FIFO から CBW データを読み出し，CBW データのコマンド解析処理関数（usbf850_storage_cbwchk）を呼び出します。

usbf850_storage_cbwchk

【 概　要 】

CBW データのコマンド解析処理

【C 言語記述形式】

INT32 usbf850_storage_cbwchk(void)

【パラメータ】

なし

【 戻り値 】

CBW チェック時のステータスを戻します。

シンボル	説　　明
DEV_OK	正常終了
DEV_ERROR	異常終了

【 機　能 】

CBW データを解析し, コマンドの種類（NO DATA 系, DATA IN 系（WRITE 系）, DATA OUT 系（READ 系））を判断し，各コマンドの処理を実行します。

usbf850_cbw_error

【 概　要 】
CBW データのエラー処理

【C 言語記述形式】
void usbf850_cbw_error(void)

【パラメータ】
なし

【 戻り値 】
なし

【 機　能 】
バルク・イン用エンドポイント（Endpoint1），バルク・アウト用エンドポイント（Endpoint2）を STALL 応答にします。

usbf850_no_data

【 概　要 】
SCSI の NO DATA 系コマンド処理

【C 言語記述形式】
void usbf850_no_data(void)

【パラメータ】
なし

【 戻り値 】
なし

【 機　能 】
NO DATA 系コマンド処理を実行し，処理結果を CSW フォーマットで送信します。

usbf850_data_in

【 概　要 】
SCSI の DATA IN 系コマンド処理

【C 言語記述形式】
void usbf850_data_in(void)

【パラメータ】
なし

【 戻り値 】
なし

【 機　能 】
DATA IN 系（WRITE 系）コマンド処理を実行し，処理結果を CSW フォーマットで送信します。

usbf850_data_out

【 概　要 】
SCSI の DATA OUT 系コマンド処理

【C 言語記述形式】
void usbf850_data_out(void)

【パラメータ】
なし

【 戻り値 】
なし

【 機　能 】
DATA OUT 系（READ 系）コマンド処理を実行し，処理結果を CSW フォーマットで送信します。

【 概　要 】
CSW 応答処理

usbf850_csw_ret

【C 言語記述形式】

INT32 usbf850_csw_ret(UINT8 status)

【パラメータ】

パラメータ	説　　明
UINT8 status	コマンド処理結果

【 戻り値 】

CSW 送信処理結果

シンボル	説　　明
DEV_OK	正常終了

【 機　能 】

処理結果から CSW フォーマットのデータを作成し，USB 送信処理を行います。

usbf850_bulkin_stall

【C 言語記述形式】
void usbf850_bulkin_stall(void)

【パラメータ】
なし

【 戻り値 】
なし

【 機　能 】
Endpoint1 の FIFO をクリアし，STALL 応答を行います

usbf850_bulkout_stall

【C 言語記述形式】
void usbf850_bulkout_stall(void)

【パラメータ】
なし

【 戻り値 】
なし

【 機　能 】
Endpoint2 の FIFO をクリアし，STALL 応答を行います

(5) SCSI コマンド用処理の関数

scsi_command_to_ata

【 概　要 】

SCSI コマンド実行処理

【C 言語記述形式】

INT32 scsi_command_to_ata(UINT8 *ScsiCommandBuf, UINT8 *pbData, INT32 lDataSize, INT32 TransFlag)

【パラメータ】

パラメータ	説　　明
UINT8 *ScsiCommandBuf	SCSI コマンド格納バッファ・ポインタ
UINT8 *pbData	コマンド用データ格納バッファ・ポインタ
INT32 lDataSize	データ・サイズ
INT32 TransFlag	データ転送方向

【 戻り値 】

SCSI コマンドの処理結果を戻します。

シンボル	説　　明
DEV_OK	正常終了
DEV_ERR_NODATA	NO DATA 系コマンドで転送方向エラー
DEV_ERR_READ	READ 系コマンドで転送方向エラー
DEV_ERR_WRITE	WRITE 系コマンドで転送方向エラー
DEV_ERROR	各コマンドの実行結果が上記のステータス以外，またはリクエストが不正

【 機　能 】

SCSI コマンドを判断し，各コマンド処理を実行します。
該当コマンドがない場合は, SENSE KEY を ILLEGAL REQUEST として SENSE DATA を更新します。

ata_test_unit_ready

【 概　要 】

TEST UNIT READY コマンド処理

【C 言語記述形式】

INT32 ata_test_unit_ready(INT32 TransFlag)

【パラメータ】

パラメータ	説　　明
INT32 TransFlag	データ転送方向

【 戻り値 】

シンボル	説　　明
DEV_OK	正常終了
DEV_ERR_NODATA	NO DATA 系コマンドで転送方向エラー

【 機　能 】

SENSE DATA をクリア（SENSE KEY = 0x00）します。転送方向が NO DATA でない場合は，SENSE KEY を ILLEGAL REQUEST として SENSE DATA を更新します。

ata_seek

【 概　要 】

SEEK コマンド処理

【C 言語記述形式】

INT32 ata_seek(INT32 TransFlag)

【パラメータ】

パラメータ	説　　明
INT32 TransFlag	データ転送方向

【 戻り値 】

シンボル	説　　明
DEV_OK	正常終了
DEV_ERR_NODATA	NO DATA 系コマンドで転送方向エラー

【 機　能 】

SENSE DATA をクリア（SENSE KEY = 0x00）します。転送方向が NO DATA でない場合は，SENSE KEY を ILLEGAL REQUEST として SENSE DATA を更新します。

ata_start_stop_unit

【 概　要 】

START STOP UNIT コマンド処理

【C 言語記述形式】

INT32 ata_start_stop_unit(INT32 TransFlag)

【パラメータ】

パラメータ	説　　明
INT32 TransFlag	データ転送方向

【 戻り値 】

処理結果

シンボル	説　　明
DEV_OK	正常終了
DEV_ERR_NODATA	NO DATA 系コマンドで転送方向エラー

【 機　能 】

SENSE DATA をクリア（SENSE KEY = 0x00）します。転送方向が NO DATA でない場合は，SENSE KEY を ILLEGAL REQUEST として SENSE DATA を更新します。

ata_synchronize_cache

【 概　要 】

SYNCHRONIZE CACHE コマンド処理

【C 言語記述形式】

INT32 ata_synchronize_cache(INT32 TransFlag)

【パラメータ】

パラメータ	説　　明
INT32 TransFlag	データ転送方向

【 戻り値 】

処理結果

シンボル	説　　明
DEV_OK	正常終了
DEV_ERR_NODATA	NO DATA 系コマンドで転送方向エラー

【 機　能 】

SENSE DATA をクリア（SENSE KEY = 0x00）します。転送方向が NO DATA でない場合は，SENSE KEY を ILLEGAL REQUEST として SENSE DATA を更新します。

ata_request_sense

【 概　要 】

REQUEST SENSE コマンド処理

【C 言語記述形式】

INT32 ata_request_sense(UINT8 *ScsiCommandBuf, UINT8 *pbData, INT32 lDataSize, INT32 TransFlag)

【パラメータ】

パラメータ	説　　明
UINT8 *ScsiCommandBuf	SCSI コマンド格納バッファ・ポインタ
UINT8 *pbData	コマンド用データ格納バッファ・ポインタ
INT32 lDataSize	データ・サイズ
INT32 TransFlag	データ転送方向

【 戻り値 】

シンボル	説　　明
DEV_OK	正常終了
DEV_ERR_NODATA	NO DATA 系コマンドで転送方向エラー
DEV_ERR_READ	READ 系コマンドで転送方向エラー

【 機　能 】

SENSE DATA を送信します。
データ・サイズが 0 で転送方向が NO DATA でない場合は，SENSE KEY を ILLEGAL REQUEST として SENSE DATA を更新します。

ata_inquiry

【 概　要 】

INQUIRY コマンド処理

【C 言語記述形式】

INT32 ata_inquiry(UINT8 *ScsiCommandBuf, UINT8 *pbData, INT32 lDataSize, INT32 TransFlag)

【パラメータ】

パラメータ	説　　明
UINT8 *ScsiCommandBuf	SCSI コマンド格納バッファ・ポインタ
UINT8 *pbData	コマンド用データ格納バッファ・ポインタ
INT32 lDataSize	データ・サイズ
INT32 TransFlag	データ転送方向

【 戻り値 】

シンボル	説　　明
DEV_OK	正常終了
DEV_ERR_READ	READ 系コマンドで転送方向エラー
DEV_ERROR	上記のステータス以外，またはリクエストが不正

【 機　能 】

SENSE DATA をクリア（SENSE KEY = 0x00）し，INQUIRY データを送信ます。コマンド・バイト 1 の CMDDT, EVPD ビットが両方 "1" の場合は, SENSE KEY を ILLEGAL REQUEST として SENSE DATA を更新します。

ata_mode_select

【 概　要 】

MODE SELECT(6)コマンド処理

【C 言語記述形式】

INT32 ata_mode_select(UINT8 *ScsiCommandBuf, UINT8 *pbData, INT32 lDataSize, INT32 TransFlag)

【パラメータ】

パラメータ	説　　明
UINT8 *ScsiCommandBuf	SCSI コマンド格納バッファ・ポインタ
UINT8 *pbData	コマンド用データ格納バッファ・ポインタ
INT32 lDataSize	データ・サイズ
INT32 TransFlag	データ転送方向

【 戻り値 】

シンボル	説　　明
DEV_OK	正常終了
DEV_ERR_WRITE	WRITE 系コマンドで転送方向エラー
DEV_ERROR	上記のステータス以外，またはリクエストが不正

【 機　能 】

SENSE DATA をクリア（SENSE KEY = 0x00）し，受信データで MODE SELECT データ・テーブルを更新ます。

転送方向やデータ･サイズが不正な場合は, SENSE KEY を ILLEGAL REQUEST として SENSE DATA を更新します。

ata_mode_select10

【 概　要 】

MODE SELECT(10)コマンド処理

【C 言語記述形式】

INT32 ata_mode_select10(UINT8 *ScsiCommandBuf, UINT8 *pbData, INT32 lDataSize, INT32 TransFlag)

【パラメータ】

パラメータ	説　　明
UINT8 *ScsiCommandBuf	SCSI コマンド格納バッファ・ポインタ
UINT8 *pbData	コマンド用データ格納バッファ・ポインタ
INT32 lDataSize	データ・サイズ
INT32 TransFlag	データ転送方向

【 戻り値 】

シンボル	説　　明
DEV_OK	正常終了
DEV_ERR_WRITE	WRITE 系コマンドで転送方向エラー
DEV_ERROR	上記のステータス以外，またはリクエストが不正

【 機　能 】

SENSE DATA をクリア（SENSE KEY = 0x00）し，受信データで MODE SELECT(10)データ・テーブルを更新します。

転送方向やデータ･サイズが不正な場合は, SENSE KEY を ILLEGAL REQUEST として SENSE DATA を更新します。

ata_mode_sense

【 概　要 】

MODE SENSE(6)コマンド処理

【C 言語記述形式】

INT32 ata_mode_sense(UINT8 *ScsiCommandBuf, UINT8 *pbData, INT32 lDataSize, INT32 TransFlag)

【パラメータ】

パラメータ	説　　明
UINT8 *ScsiCommandBuf	SCSI コマンド格納バッファ・ポインタ
UINT8 *pbData	コマンド用データ格納バッファ・ポインタ
INT32 lDataSize	データ・サイズ
INT32 TransFlag	データ転送方向

【 戻り値 】

シンボル	説　　明
DEV_OK	正常終了
DEV_ERR_READ	READ 系コマンドで転送方向エラー
DEV_ERROR	上記のステータス以外，またはリクエストが不正

【 機　能 】

SENSE DATA をクリア（SENSE KEY = 0x00）し，MODE SENSE データを送信します。

ata_mode_sense10

【 概　要 】

MODE SENSE(10)コマンド処理

【C 言語記述形式】

INT32 ata_mode_sense10(UINT8 *ScsiCommandBuf, UINT8 *pbData, INT32 lDataSize, INT32 TransFlag)

【パラメータ】

パラメータ	説　　明
UINT8 *ScsiCommandBuf	SCSI コマンド格納バッファ・ポインタ
UINT8 *pbData	コマンド用データ格納バッファ・ポインタ
INT32 lDataSize	データ・サイズ
INT32 TransFlag	データ転送方向

【 戻り値 】

シンボル	説　　明
DEV_OK	正常終了
DEV_ERR_READ	READ 系コマンドで転送方向エラー
DEV_ERROR	上記のステータス以外，またはリクエストが不正

【 機　能 】

SENSE DATA をクリア（SENSE KEY = 0x00）し，MODE SENSE(10)データを送信します。

ata_read_format_capacities

【 概　要 】

READ FORMAT CAPACITIES コマンド処理

【C 言語記述形式】

INT32 ata_read_format_capacities(UINT8 *ScsiCommandBuf, UINT8 *pbData, INT32 lDataSize, INT32 TransFlag)

【パラメータ】

パラメータ	説　　明
UINT8 *ScsiCommandBuf	SCSI コマンド格納バッファ・ポインタ
UINT8 *pbData	コマンド用データ格納バッファ・ポインタ
INT32 lDataSize	データ・サイズ
INT32 TransFlag	データ転送方向

【 戻り値 】

シンボル	説　　明
DEV_OK	正常終了
DEV_ERR_READ	READ 系コマンドで転送方向エラー
DEV_ERROR	上記のステータス以外，またはリクエストが不正

【 機　能 】

SENSE DATA をクリア（SENSE KEY = 0x00）し，FORMAT CPACITY データを送信します。

ata_read_capacity

【 概　要 】

READ CAPACITY コマンド処理

【C 言語記述形式】

INT32 ata_read_capacity(UINT8 *ScsiCommandBuf, UINT8 *pbData, INT32 lDataSize, INT32 TransFlag)

【パラメータ】

パラメータ	説　　明
UINT8 *ScsiCommandBuf	SCSI コマンド格納バッファ・ポインタ
UINT8 *pbData	コマンド用データ格納バッファ・ポインタ
INT32 lDataSize	データ・サイズ
INT32 TransFlag	データ転送方向

【 戻り値 】

シンボル	説　　明
DEV_OK	正常終了
DEV_ERR_READ	READ 系コマンドで転送方向エラー
DEV_ERROR	上記のステータス以外，またはリクエストが不正

【 機　能 】

SENSE DATA をクリア（SENSE KEY = 0x00）し，CPACITY データを送信します。

ata_read6

【 概　要 】

READ(6)コマンド処理

【C 言語記述形式】

INT32 ata_read6(UINT8 *ScsiCommandBuf, UINT8 *pbData, INT32 lDataSize, INT32 TransFlag)

【パラメータ】

パラメータ	説　　明
UINT8 *ScsiCommandBuf	SCSI コマンド格納バッファ・ポインタ
UINT8 *pbData	コマンド用データ格納バッファ・ポインタ
INT32 lDataSize	データ・サイズ
INT32 TransFlag	データ転送方向

【 戻り値 】

シンボル	説　　明
DEV_OK	正常終了
DEV_ERR_READ	READ 系コマンドで転送方向エラー
DEV_ERROR	上記のステータス以外，またはリクエストが不正

【 機　能 】

SENSE DATA をクリア（SENSE KEY = 0x00）し，データ領域から読み出したデータを送信します。読み出し開始アドレスは，SCSI コマンドの LBA（Local Block Address）とブロック・サイズから算出します。

転送方向や SCSI コマンドの Flag または Link ビットが不正な場合は，SENSE KEY を ILLEGAL REQUEST として SENSE DATA を更新します。

ata_read10

【 概　要 】

READ(10)コマンド処理

【C 言語記述形式】

INT32 ata_read10(UINT8 *ScsiCommandBuf, UINT8 *pbData, INT32 lDataSize, INT32 TransFlag)

【パラメータ】

パラメータ	説　　明
UINT8 *ScsiCommandBuf	SCSI コマンド格納バッファ・ポインタ
UINT8 *pbData	コマンド用データ格納バッファ・ポインタ
INT32 lDataSize	データ・サイズ
INT32 TransFlag	データ転送方向

【 戻り値 】

シンボル	説　　明
DEV_OK	正常終了
DEV_ERR_READ	READ 系コマンドで転送方向エラー
DEV_ERROR	上記のステータス以外，またはリクエストが不正

【 機　能 】

SENSE DATA をクリア（SENSE KEY = 0x00）し，データ領域から読み出したデータを送信します。読み出し開始アドレスは，SCSI コマンドの LBA（Local Block Address）とブロック・サイズから算出します。

転送方向や SCSI コマンドの Flag または Link ビットが不正な場合は，SENSE KEY を ILLEGAL REQUEST として SENSE DATA を更新します。

ata_write6

【 概　要 】

WRITE(6)コマンド処理

【C 言語記述形式】

INT32 ata_write6(UINT8 *ScsiCommandBuf, UINT8 *pbData, INT32 lDataSize, INT32 TransFlag)

【パラメータ】

パラメータ	説　　明
UINT8 *ScsiCommandBuf	SCSI コマンド格納バッファ・ポインタ
UINT8 *pbData	コマンド用データ格納バッファ・ポインタ
INT32 lDataSize	データ・サイズ
INT32 TransFlag	データ転送方向

【 戻り値 】

シンボル	説　　明
DEV_OK	正常終了
DEV_ERR_WRITE	WRITE 系コマンドで転送方向エラー
DEV_ERROR	上記のステータス以外，またはリクエストが不正

【 機　能 】

SENSE DATA をクリア（SENSE KEY = 0x00）し，受信データをデータ領域に書き込みます。
書き込み開始アドレスは，SCSI コマンドの LBA（Local Block Address）とブロック・サイズから算出します。
転送方向や SCSI コマンドの Flag または Link ビットが不正な場合は，SENSE KEY を ILLEGAL REQUEST として SENSE DATA を更新します。

ata_write10

【 概　要 】

WRITE(10)コマンド処理

【C 言語記述形式】

INT32 ata_write10(UINT8 *ScsiCommandBuf, UINT8 *pbData, INT32 lDataSize, INT32 TransFlag)

【パラメータ】

パラメータ	説　　明
UINT8 *ScsiCommandBuf	SCSI コマンド格納バッファ・ポインタ
UINT8 *pbData	コマンド用データ格納バッファ・ポインタ
INT32 lDataSize	データ・サイズ
INT32 TransFlag	データ転送方向

【 戻り値 】

シンボル	説　　明
DEV_OK	正常終了
DEV_ERR_WRITE	WRITE 系コマンドで転送方向エラー
DEV_ERROR	上記のステータス以外，またはリクエストが不正

【 機　能 】

SENSE DATA をクリア（SENSE KEY = 0x00）し，受信データをデータ領域に書き込みます。
書き込み開始アドレスは，SCSI コマンドの LBA（Local Block Address）とブロック・サイズから算出します。
転送方向や SCSI コマンドの Flag または Link ビットが不正な場合は，SENSE KEY を ILLEGAL REQUEST として SENSE DATA を更新します。

ata_verify

【 概　要 】

VERIFY コマンド処理

【C 言語記述形式】

INT32 ata_verify(UINT8 *ScsiCommandBuf, UINT8 *pbData, INT32 lDataSize, INT32 TransFlag)

【パラメータ】

パラメータ	説　　明
UINT8 *ScsiCommandBuf	SCSI コマンド格納バッファ・ポインタ
UINT8 *pbData	コマンド用データ格納バッファ・ポインタ
INT32 lDataSize	データ・サイズ
INT32 TransFlag	データ転送方向

【 戻り値 】

シンボル	説　　明
DEV_OK	正常終了
DEV_ERR_NODATA	NO DATA 系コマンドで転送方向エラー
DEV_ERROR	上記のステータス以外，またはリクエストが不正

【 機　能 】

受信データをデータ領域に書き込みます。
書き込み開始アドレスは，SCSI コマンドの LBA（Local Block Address）とブロック・サイズから算出します。
転送方向や SCSI コマンドの BYTCHK ビットが不正な場合は，SENSE KEY を ILLEGAL REQUEST として SENSE DATA を更新します。

ata_write_verify

【 概　要 】

WRITE VERIFY コマンド処理

【C 言語記述形式】

INT32 ata_write_verify(UINT8 *ScsiCommandBuf, UINT8 *pbData, INT32 lDataSize, INT32 TransFlag)

【パラメータ】

パラメータ	説　　明
UINT8 *ScsiCommandBuf	SCSI コマンド格納バッファ・ポインタ
UINT8 *pbData	コマンド用データ格納バッファ・ポインタ
INT32 lDataSize	データ・サイズ
INT32 TransFlag	データ転送方向

【 戻り値 】

シンボル	説　　明
DEV_OK	正常終了
DEV_ERR_WRITE	WRITE 系コマンドで転送方向エラー
DEV_ERROR	上記のステータス以外，またはリクエストが不正

【 機　能 】

SENSE DATA をクリア（SENSE KEY = 0x00）し，受信データをデータ領域に書き込みます。
書き込み開始アドレスは，SCSI コマンドの LBA（Local Block Address）とブロック・サイズから算出します。
転送方向や SCSI コマンドの Flag または Link ビットが不正な場合は，SENSE KEY を ILLEGAL REQUEST として SENSE DATA を更新します。

ata_write_buff

【 概　要 】

WRITE BUFF コマンド処理

【C 言語記述形式】

INT32 ata_write_buff(UINT8 *ScsiCommandBuf, UINT8 *pbData, INT32 lDataSize, INT32 TransFlag)

【パラメータ】

パラメータ	説　　明
UINT8 *ScsiCommandBuf	SCSI コマンド格納バッファ・ポインタ
UINT8 *pbData	コマンド用データ格納バッファ・ポインタ
INT32 lDataSize	データ・サイズ
INT32 TransFlag	データ転送方向

【 戻り値 】

シンボル	説　　明
DEV_OK	正常終了
DEV_ERR_WRITE	WRITE 系コマンドで転送方向エラー
DEV_ERROR	上記のステータス以外，またはリクエストが不正

【 機　能 】

SENSE DATA をクリア（SENSE KEY = 0x00）し，受信データを読み捨てます。

scsi_to_usb

【 概　要 】

（SCSI コマンド系）USB データ送信処理

【C 言語記述形式】

INT32 scsi_to_usb(UINT8 *pbData, INT32 TransFlag)

【パラメータ】

パラメータ	説　　明
UINT8 *pbData	コマンド用データ格納バッファ・ポインタ
INT32 TransFlag	データ転送方向

【 戻り値 】

シンボル	説　　明
DEV_OK	正常終了
DEV_ERR_READ	READ 系コマンドで転送方向エラー

【 機　能 】

USB データ送信処理関数（usbf850_data_send）を呼び出し，バルク・アウト用エンドポイント（Endpoint1）からデータを送信します。

転送方向が不正な場合は，SENSE KEY を ILLEGAL REQUEST として SENSE DATA を更新します。

4.4　データ構造体

サンプル・ドライバが使用するデータ構造体について次に示します。

(1) USB デバイス・リクエスト構造体

USB デバイス・リクエスト構造体は，"usbf850.h" ファイルで定義されています。

```
typedef struct {
      UINT8  RequstType;    /*bmRequestType */
      UINT8  Request;       /*bRequest      */
      UINT16 Value;         /*wValue        */
      UINT16 Index;         /*wIndex        */
      UINT16 Length;        /*wLength       */
      UINT8* Data;          /*index to Data */
} USB_SETUP;
```

図 4－21　USB デバイス・リクエスト構造体

(2) CBW データ構造体

CBW データ構造体は，"usbf850_storage.h" ファイルで定義されています。

```
typedef struct {    /* CBW(Command Block Wrapper) DATA */
      UINT8   dCBWSignature[4];            /* シグネチャ */
      UINT8   dCBWTag[4];                  /* タグ */
      UINT8   dCBWDataTransferLength[4];   /* 転送データ長 */
      UINT8   bmCBWFlags;                  /* データ方向（OUT/IN/NO DATA）の指定 */
      UINT8   bCBWLUN;                     /* 対象デバイスの番号 */
      UINT8   bCBWCBLength;                /* CBWCB の有効バイト数 */
      UINT8   CBWCB[16];                   /* CBWCB（コマンド） */
} CBW_INFO,*PCBW_INFO;
```

図 4－22　CBW データ構造体

(3) CSW データ構造体

CSW データ構造体は，"usbf850_storage.h" ファイルで定義されています。

```
typedef struct {    /* CSW(Command Status Wrapper) DATA */
      UINT8   dCSWSignature[4];            /* シグネチャ */
      UINT8   dCSWTag[4];                  /* タグ */
      UINT8   dCSWDataResidue[4];          /* 指定転送データ長と処理したデータ長の差 */
      UINT8   bmCSWStatus;                 /* 処理結果のステータス */
} CSW_INFO,*PCSW_INFO;
```

図 4－23　CSW データ構造体

（4）SCSI SENSE DATA 構造体

SCSI SENSE DATA 構造体は，"scsi_cmd.c" ファイルで定義されています。

```
typedef struct _SCSI_SENSE_DATA {
      UINT8  sense_key;
      UINT8  asc;
      UINT8  ascq;
} SCSI_SENSE_DATA, *PSCSI_SENSE_DATA;
```

図 4－24　SCSI SENSE DATA 構造体

5. 開発環境

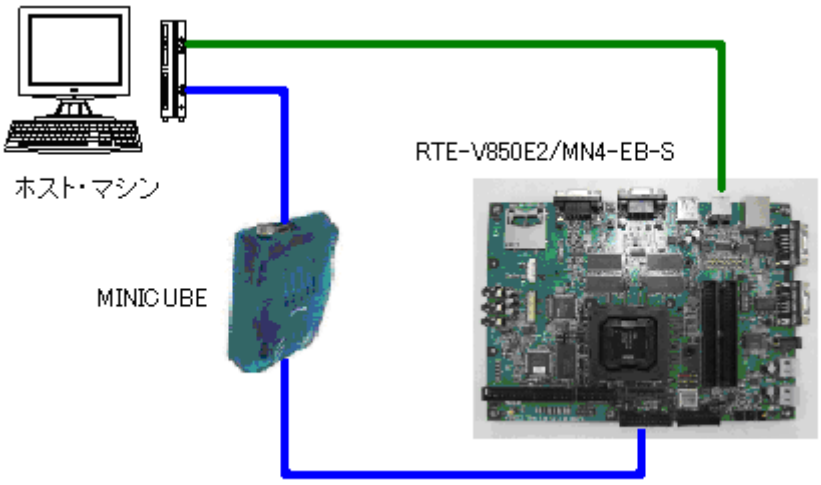
この章では，V850E2/MN4 向け USB マス・ストレージ・クラス用サンプル・ドライバを利用したアプリケーション・プログラムを開発する際の環境構築の例と，そこでのデバッグの手順について説明します。

5.1 開発環境

ここでは，ハードウエア・ツールとソフトウエア・ツールの製品構成例を示します。

5.1.1 システム構成

サンプル・ドライバを利用するシステムの構成を図 5－1 に示します。



（備考）RTE-V850E2/MN4-EB-S の外形やポートについては，「7. スタータ・キット概説」を参照してください。

図 5－1　開発環境のシステム構成

5.1.2　プログラム開発

サンプル・ドライバを利用したシステムを開発する際には，次のようなハードウエアとソフトウエアが必要です。

表 5-1　プログラム開発環境構成例

構成品		製品例	備　考
ハードウエア	ホスト・マシン	－	PC/AT™互換機 （OS：Windows XPまたはWindows Vista®）
ソフトウエア	統合開発ツール	CubeSuite	V1.20
		Multi	V5.1.7D
		IAR Embedded Workbench	V3.71
	コンパイラ	CX850	V1.00
		CCV850	V5.1.7D
		ICCV850	V3.71.2

5.1.3　デバッグ

サンプル・ドライバを利用したシステムをデバッグする際には，次のようなハードウエアとソフトウエアが必要です。

表 5-2　デバッグ環境構成例

構成品		製品例	備　考
ハードウエア	ホスト・マシン	－	PC/AT互換機 （OS：Windows XPまたはWindows Vista®））
	ターゲット	RTE-V850E2/MN4-EB-S	（株）マイダス・ラボ製
	USBケーブル	－	Bコネクタ - Aコネクタ
ソフトウエア	統合開発ツール・デバッガ	CubeSuite	V1.20
		Multi	V5.1.7D
		IAR Embedded Workbench	V3.71
ファイル	デバイス・ファイル	DF703512	V850E2/MN4用（CubeSuite/Multi/IAR Embedded Workbench用別）
	デバッグ・ポート用ホスト・ドライバ	－	(注8)
	プロジェクト関連ファイル	－	(注9)

（注 8）製品や入手方法については弊社までお問い合わせください。

（注 9）CubeSuite / Multi / IAR Embedded Workbench で構築した場合のファイルがサンプル・ドライバに同梱されています。

5.2　CubeSuite環境設定

ここでは，「5. 1　開発環境」に示した製品構成の中で，CubeSuite を用いた開発やデバッグを行うための準備について説明します。Multi を用いた開発やデバッグを行う場合には「5. 4　Multi 環境設定」を、IAR Embedded Workbench を用いた開発やデバッグを行う場合には「5. 6　IAR Embedded Workbench 環境設定」参照してください。

5.2.1　ホスト環境整備

ホスト・マシン上に専用のワークスペースを作成します。

(1) CubeSuite 統合開発ツールのインストール

CubeSuite をインストールします。詳細は CubeSuite のユーザーズマニュアルを参照してください。

(2) ドライバ類の展開

サンプル・ドライバの提供ファイル一式を，フォルダ構成を変えずに任意のディレクトリに格納します。

また，デバッグ・ポート用ホスト・ドライバを任意のディレクトリに格納します。

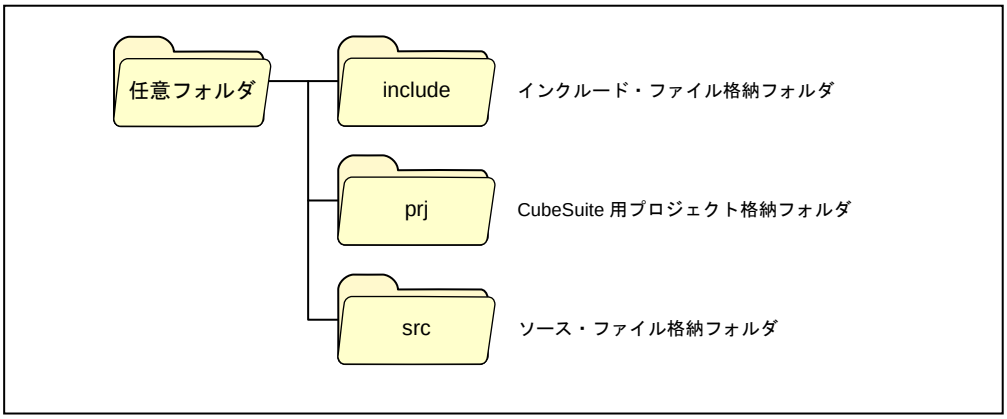


図 5−2　サンプル・ドライバのフォルダ構成（CubeSuite 版）

(3) デバイス・ファイルのインストール

CubeSuite 用 V850E2/MN4 用のデバイス・ファイルを，CubeSuite インストールフォルダにコピーします。

例)　C:¥Program Files¥NEC Electronics CubeSuite¥CubeSuite¥Device_Custom

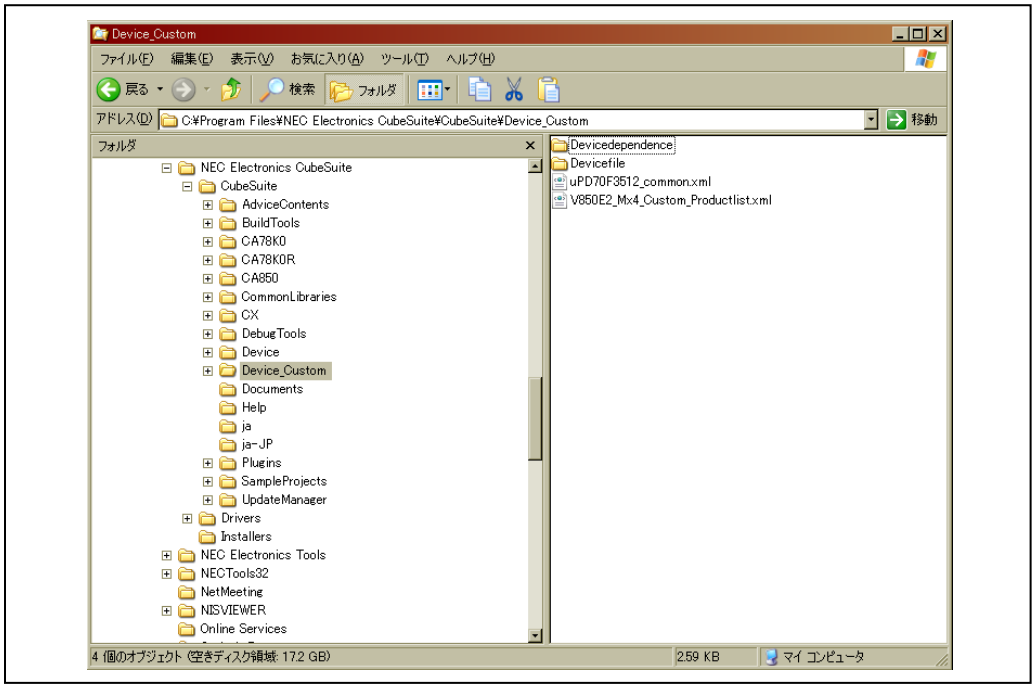


図 5−3　デバイス・ファイルのコピー先の例

(4) ワークスペースの設定

ここではサンプル・ドライバに同梱のプロジェクト関連ファイルを使用する場合の手順を示します。

<1>　CubeSuite を起動し，「ファイル」メニューから「ファイルを開く」を選択します。

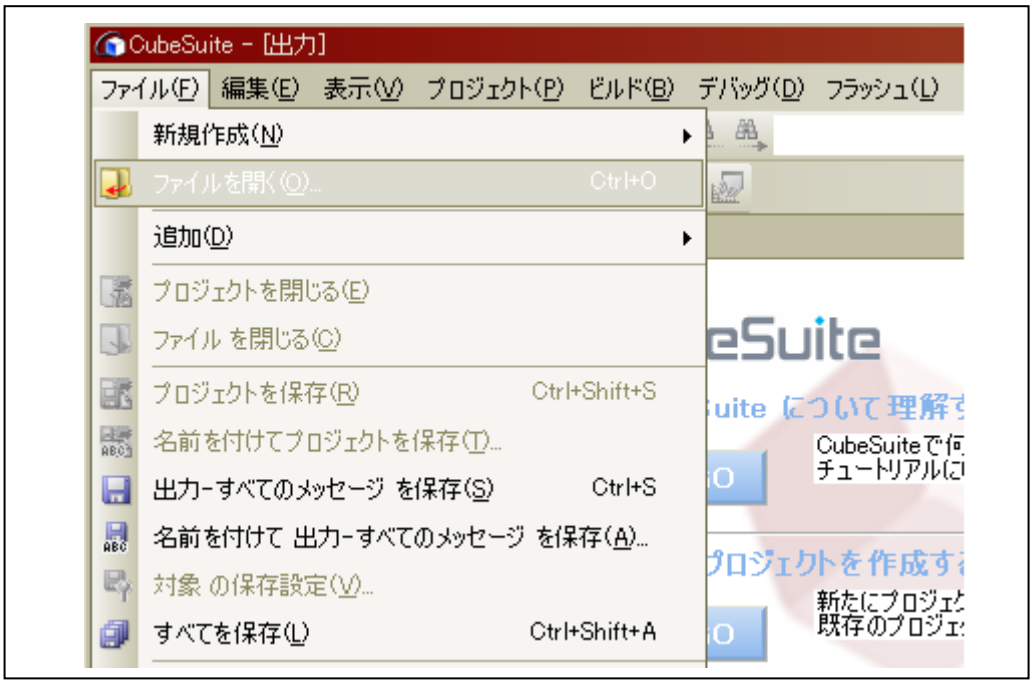


図 5−4　CubeSuite メニュー選択

<2>　「ファイルを開く」ダイアログが開きます。サンプル・ドライバを格納したディレクトリの「prj」フォルダにある CubeSuite 用プロジェクト・ファイルを指定します。

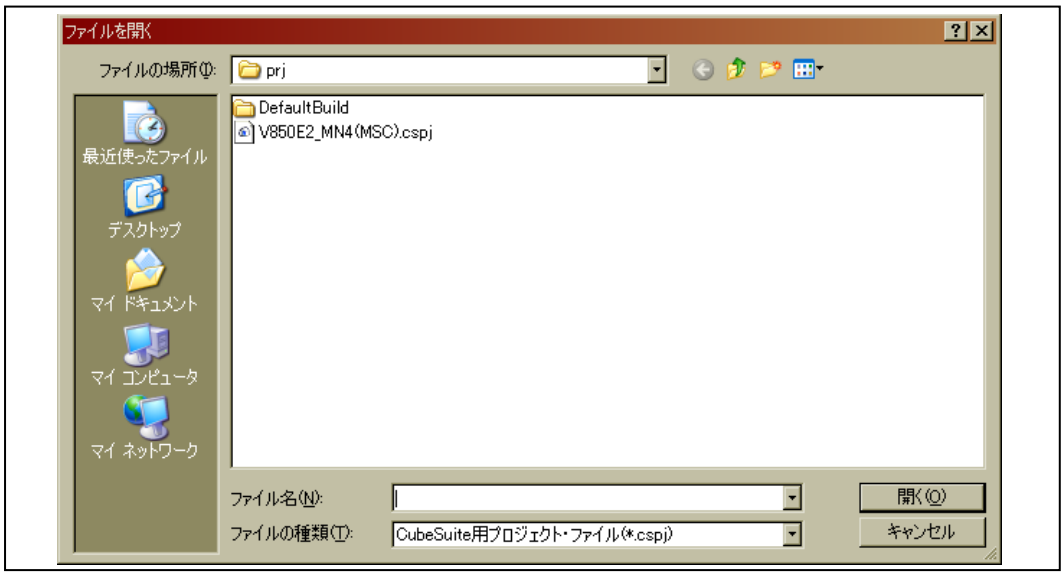


図 5−5　CubeSuite プロジェクト・ファイル選択

(5) ビルド・ツールの設定

ここではビルド・ツールとして使用する CX850 のバージョン選択と，デバッグ・ツールとして V850E2M　MINICUBE を使用する場合の手順を示します。

<1>　CubeSuite の「プロジェクト・ツリー」から「CX(ビルド・ツール)」を選択し，プロパティを表示します。

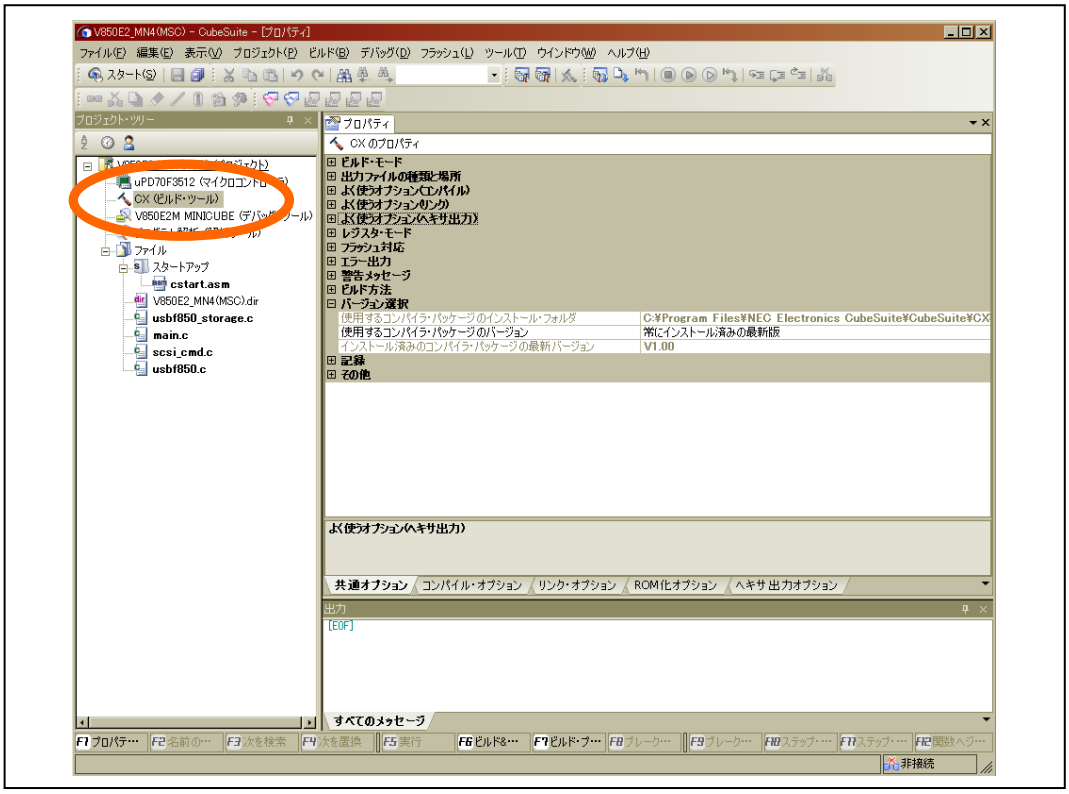


図 5−6　ビルド・ツール選択

<2>　「バージョン選択」の項目を選択し，「使用するコンパイラ・パッケージのバージョン」の項を「常にインストール済みの最新版」に設定します。

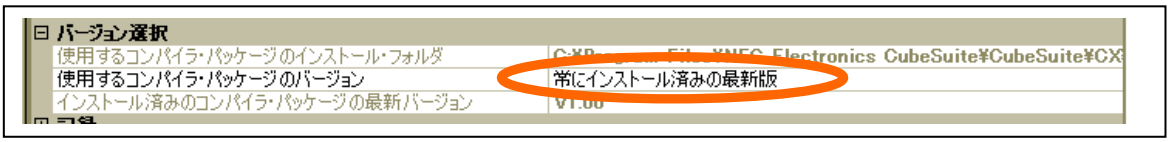


図 5−7　コンパイラ・パッケージ設定

<3>　プロジェクト・ツリーより「V850E2M　MINICUBE(デバッグ・ツール)」を選択し，右クリックメニューから「使用するデバッグ・ツール」→「V850E2M MINICUBE」を選択します。

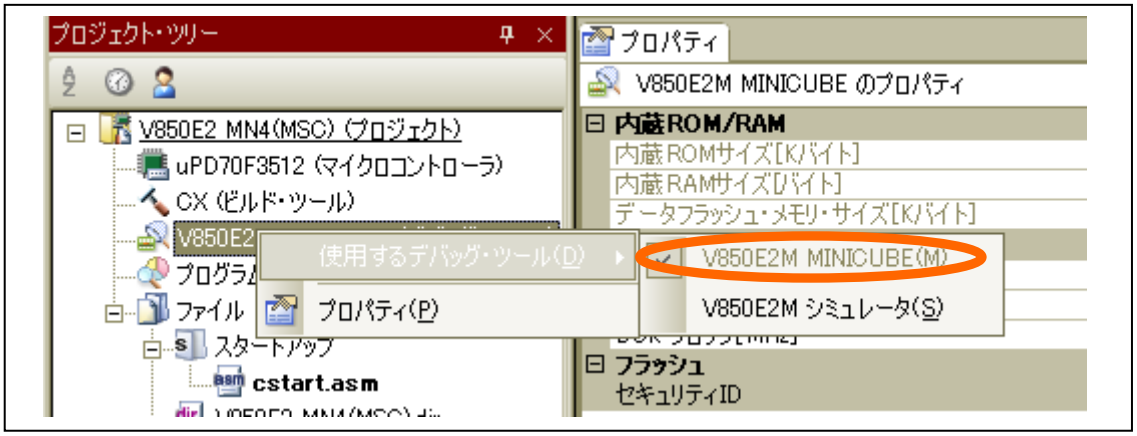


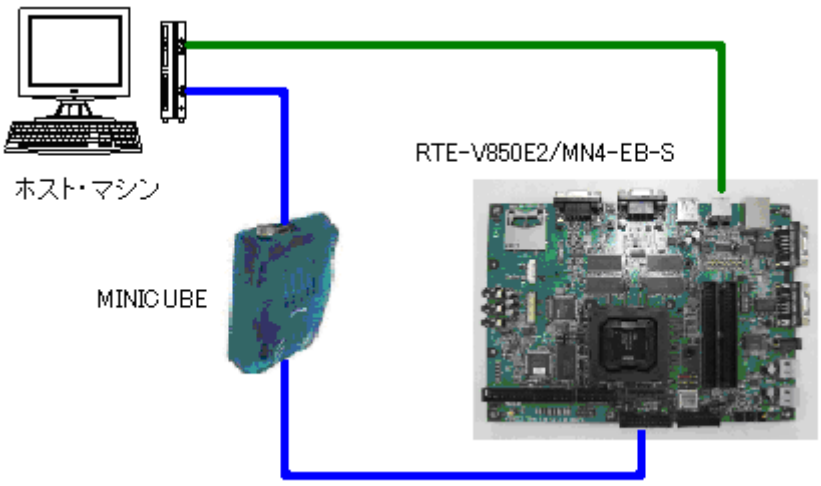
図 5−8　デバッグ・ツール選択

5.2.2　ターゲット環境整備

デバッグに使用するターゲット・デバイスを接続します。接続方法は CubeSuite/Multi/IAR Embedded Workbench 共に同じです。

(1) デバッグ・ポートの接続

RTE-V850E2/MN4-EB-S とホスト・マシンを接続します。RTE-V850E2/MN4-EB-S とホスト・マシンをデバッグ用に MINICUBE で接続します。また RTE-V850E2/MN4-EB-S の USB B タイプコネクタとホスト・マシンの USB コネクタを MSC 用に接続します。



（備考）RTE-V850E2/MN4-EB-S の外形やポートについては，「7. スタータ・キット概説」を参照してください。

図 5－9　RTE-V850E2/MN4-EB-S の接続

(2) ホスト・ドライバのインストール

USB B コネクタを使用してホスト・マシンに接続するには，ドライバをインストールする必要があります。

USB　B コネクタとの接続に使用するドライバは，Windows 標準のマス・ストレージ・クラス用ホスト・ドライバを使用します。詳細は「5. 8　動作確認」を参照してください。

5.3　CubeSuite環境デバッグ

ここでは，「5. 2　CubeSuite 環境設定」に示したワークスペースで開発したアプリケーション・プログラムのデバッグ手順について説明します。

5.3.1　ロード・モジュール生成

ターゲット・デバイスにプログラムを書き込むには，C 言語やアセンブリ言語で記述されたファイルを C コンパイラなどで変換してロード・モジュールを生成します。
CubeSuite では，「ビルド」メニューから「ビルド・プロジェクト」を選択すると，ロード・モジュールが生成されます。

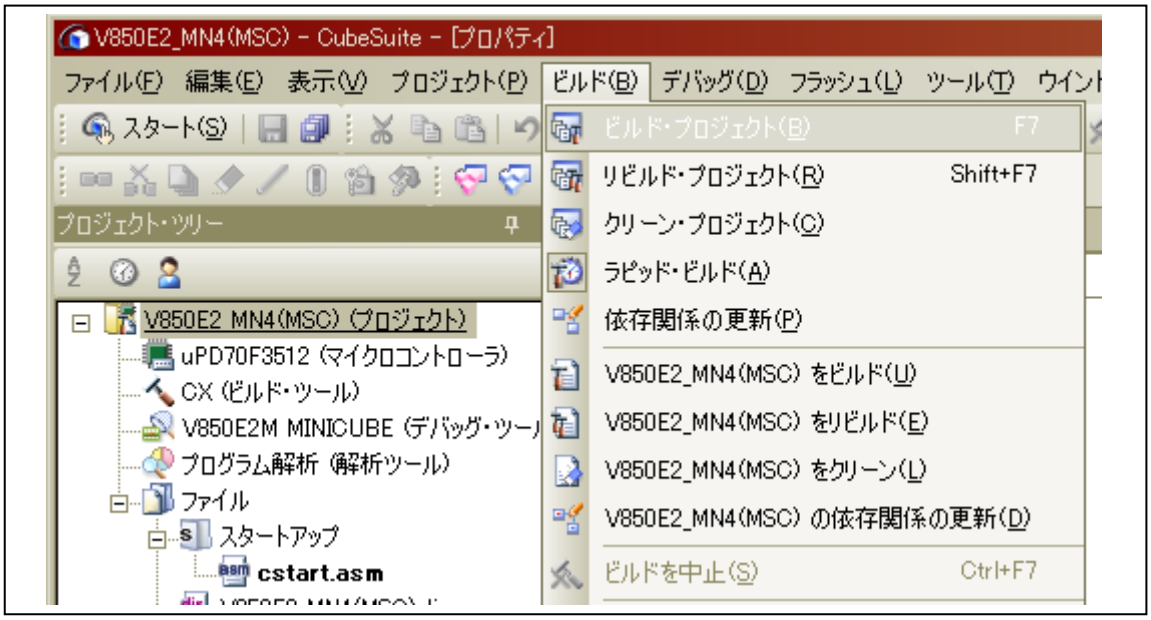


図 5－10　ビルド・プロジェクト選択

5.3.2　ロードと実行

生成したロード・モジュールをターゲットに書き込んで（ロード）実行させます。

(1) ロード・モジュールの書き込み

ここでは CubeSuite を介して RTE-V850E2/MN4-EB-S にロード・モジュールを書き込む手順を示します。

<1>　「デバッグ」メニューから「デバッグ・ツールへダウンロード」を選択してデバッガを起動します。

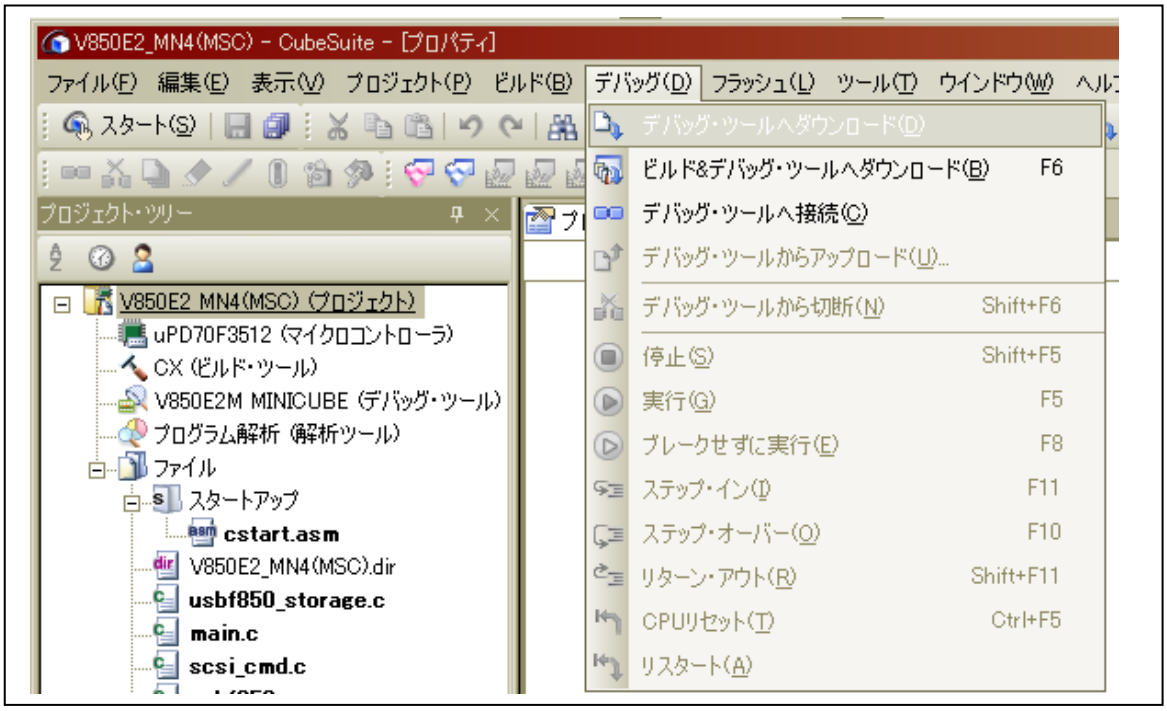


図 5－11　デバッグ・ツールへダウンロード選択

<2>　デバッグ・ツールを介して，ロード・モジュールのダウンロードが開始されます。

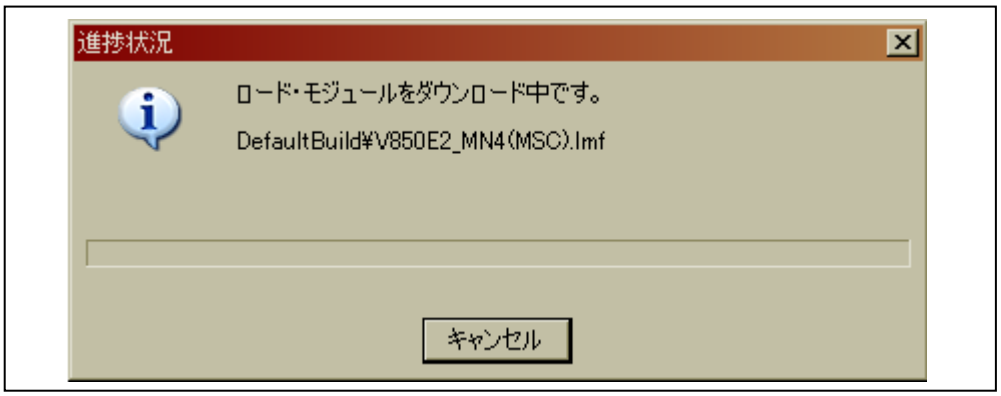


図 5－12　ダウンロード実行

(2) プログラムの実行

CubeSuite の  ボタンを押下します。または「デバッグ」メニューから「実行」を選択します。

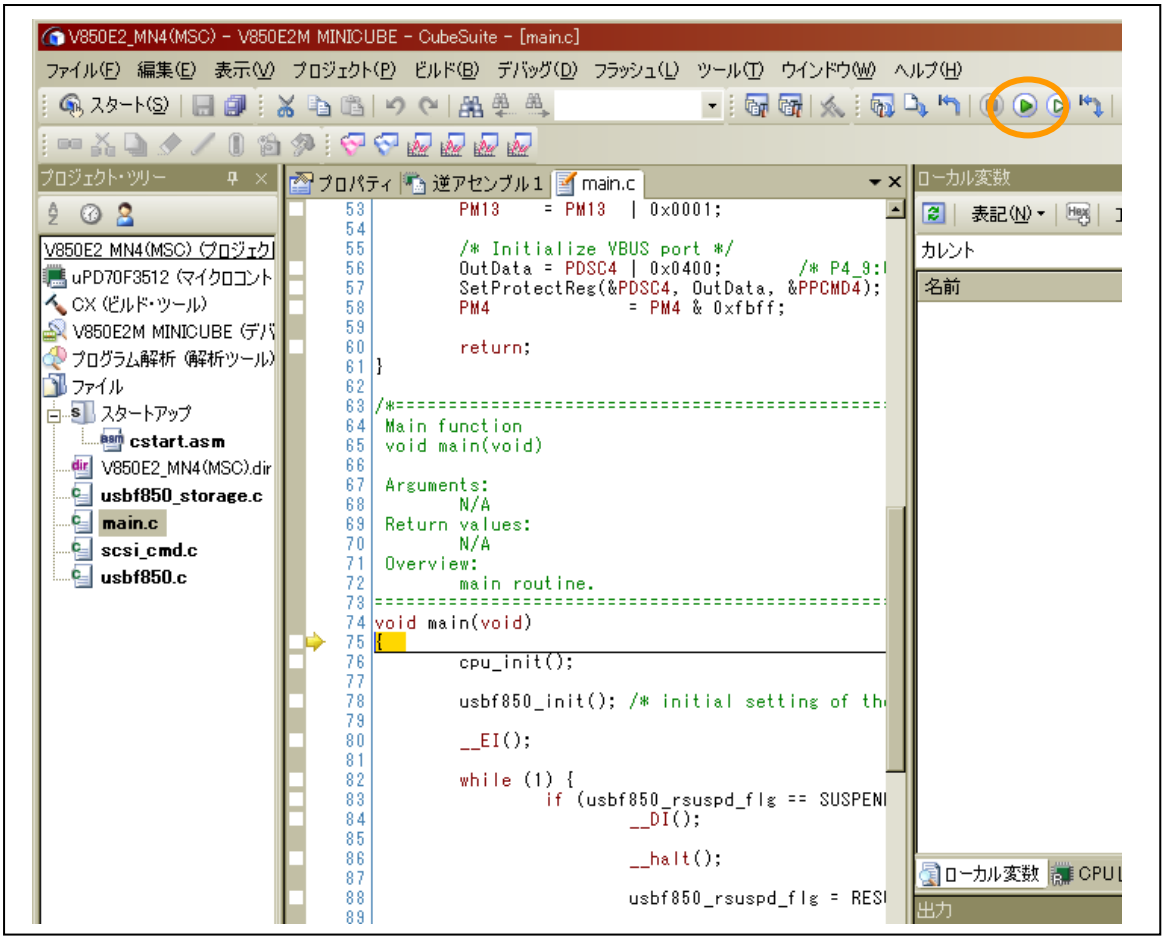


図 5－13　実行開始

5.4　Multi環境設定

ここでは，「5. 1　開発環境」に示した製品構成の中で，Multi を用いた開発やデバッグを行うための準備について説明します。

5.4.1　ホスト環境整備

ホスト・マシン上に専用のワークスペースを作成します。

(1) Multi 統合開発ツールのインストール

Multi をインストールします。詳細は GHS のユーザーズマニュアルを参照してください。

(2) ドライバ類の展開

サンプル・ドライバの提供ファイル一式を，フォルダ構成を変えずに任意のディレクトリに格納します。

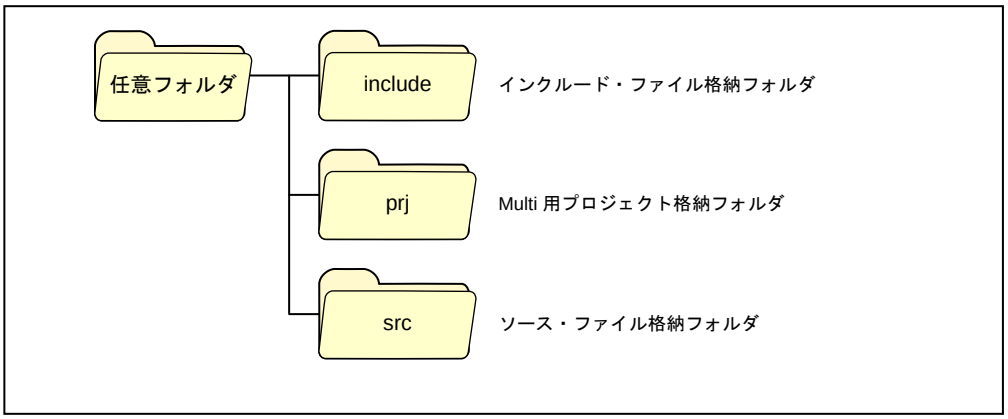


図 5−14　サンプル・ドライバのフォルダ構成（Multi 版）

(3) デバイス・ファイルのインストール

Multi 用 V850E2/MN4 用のデバイス・ファイルを，Multi インストールフォルダにコピーします。

例）　C:¥Green¥V800.V517D¥devicefile

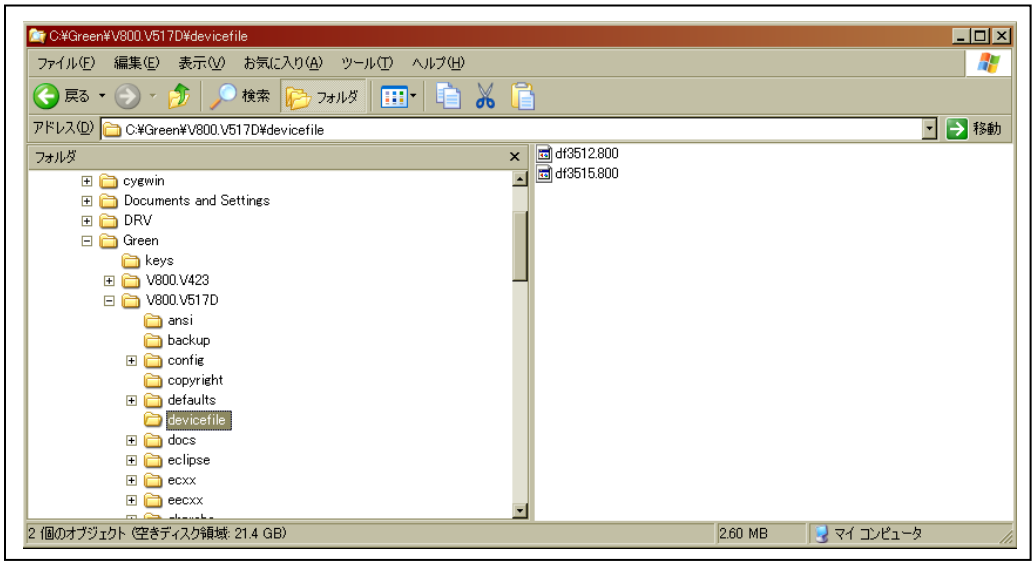


図 5−15　デバイス・ファイルのコピー先の例

(4) Multi の起動

エクスプローラから，サンプル・ドライバに同梱されている「V850E2_MN4(MSC)_GHS.gpj」の Multi Project File を選択し，Multi を起動します。

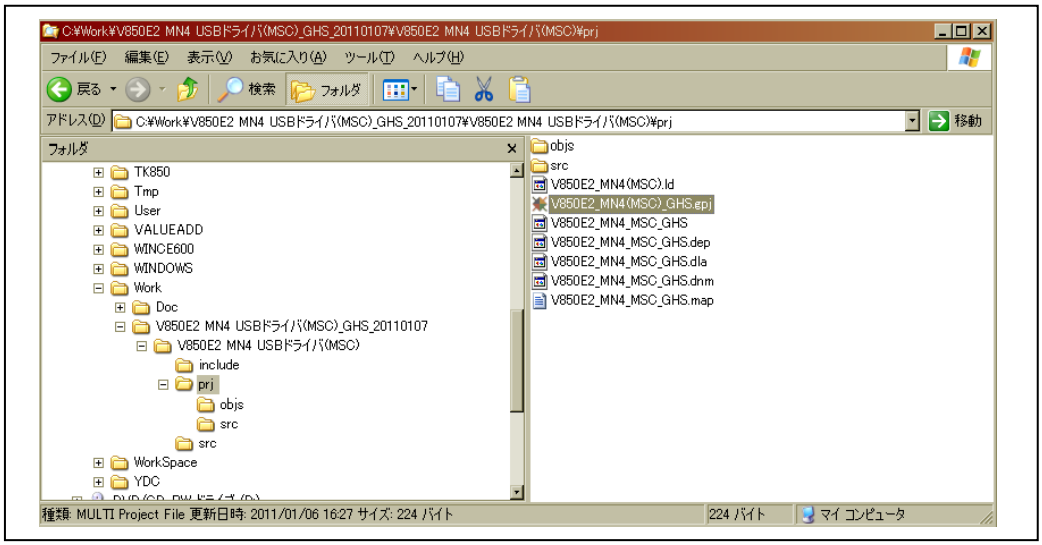


図 5－16　Multi Project File の選択

(5) デバッグ・ツールの設定

ここでは，デバッグ・ツールとして MINICUBE を使用する場合の手順を示します。

<1>　Multi の「Connect」から「Connect」を選択し，Connection Chooser を表示します。

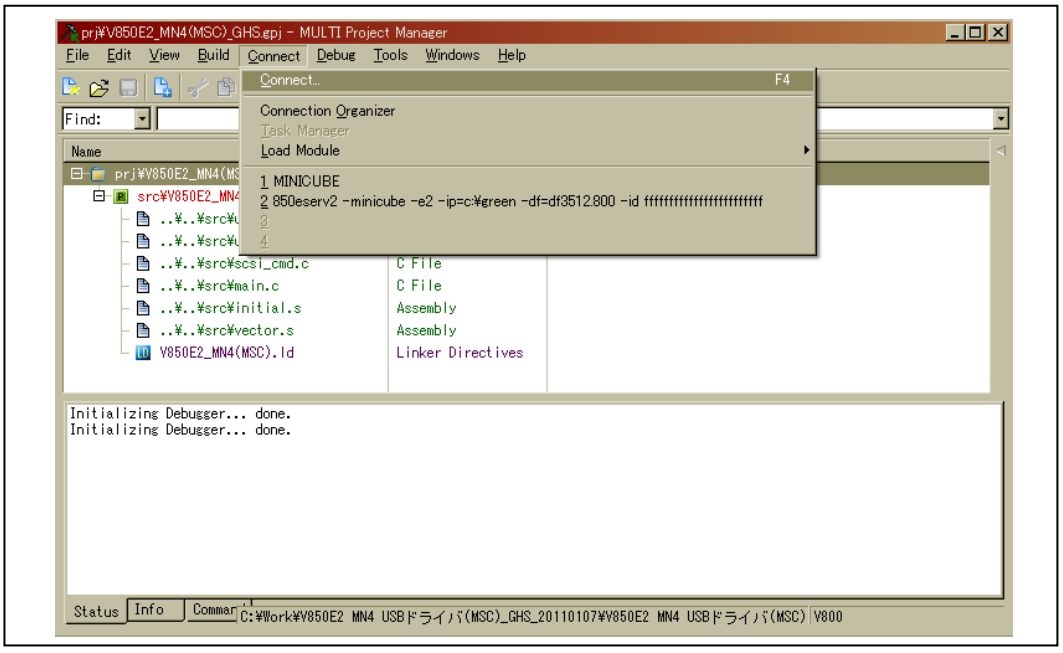


図 5－17　Connection Chooser の起動

<2>　Connection Chooser ダイアログから，「Create a new Connection Method」アイコンを選択します。

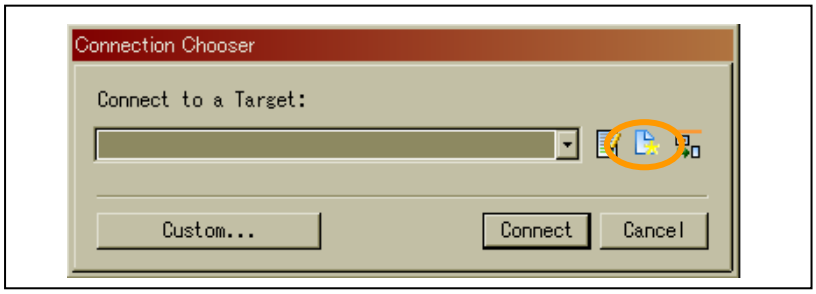


図 5－18　Create a new Connection Method の選択

<3>　Create New Connection Method ダイアログで Name に任意の設定名を，Type に「Custom」を選択して「Create…」ボタンを押下し，MINICUBE 接続設定を作成します。ここでは Name に「MINICUBE」を設定した例を示します。

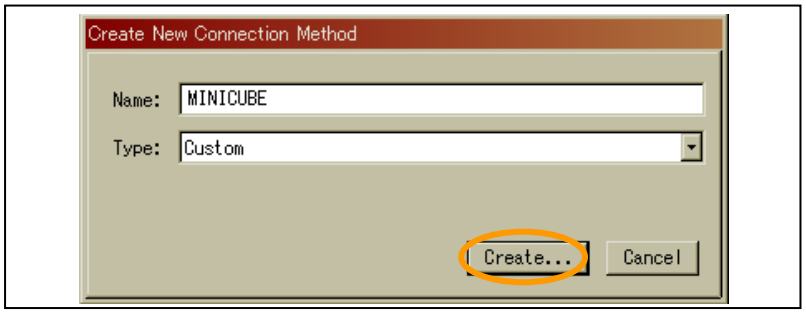


図 5－19　Create New Connection Method の作成

<4>　Connection Editor が起動するので，「Server」及び「Arguments」に以下の設定を記述して OK ボタンを押下します。

Server　　：850eserv2
Arguments：-minicube -e2 -ip=c:¥green¥v800.V517d¥devicefile -df=df3512.800 -id
　　ffffffffffffffffffffffff（注 10）

（注 10）”f”を 24 個

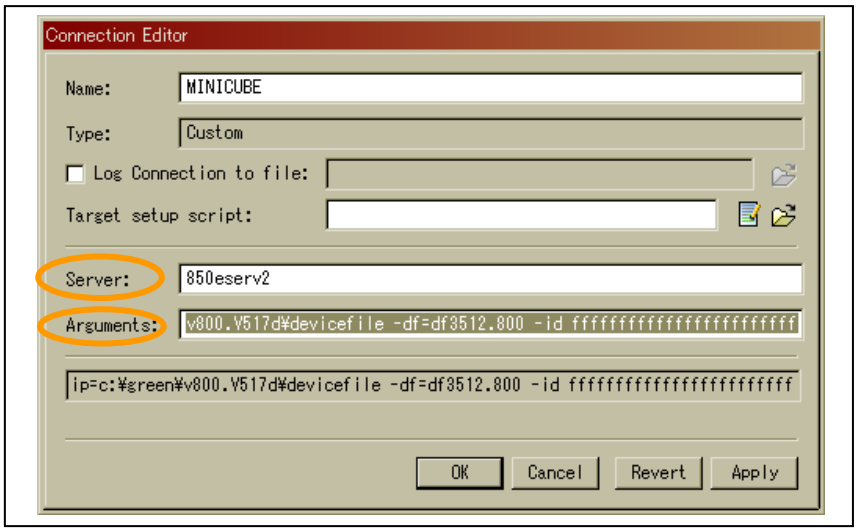


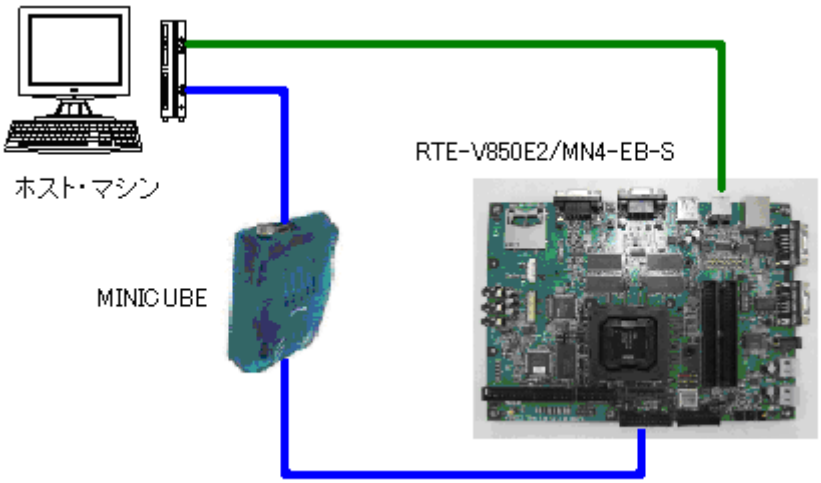
図 5－20　Connection Editor の設定

5.4.2　ターゲット環境整備

デバッグに使用するターゲット・デバイスを接続します。接続方法は CubeSuite/Multi/IAR Embedded Workbench 共に同じです。

(1) デバッグ・ポートの接続

RTE-V850E2/MN4-EB-S とホスト・マシンを接続します。RTE-V850E2/MN4-EB-S とホスト・マシンをデバッグ用に MINICUBE で接続します。また RTE-V850E2/MN4-EB-S の USB B タイプコネクタとホスト・マシンの USB コネクタを MSC 用に接続します。



（備考）RTE-V850E2/MN4-EB-S の外形やポートについては，「7. スタータ・キット概説」を参照してください。

図 5－21　RTE-V850E2/MN4-EB-S の接続

(2) ホスト・ドライバのインストール

USB B コネクタを使用してホスト・マシンに接続するには，ドライバをインストールする必要があります。

USB　B コネクタとの接続に使用するドライバは，Windows 標準のマス・ストレージ・クラス用ホスト・ドライバを使用します。詳細は「5. 8　動作確認」を参照してください。

5.5　Multi環境デバッグ

ここでは，「5. 4　Multi 環境設定」に示したワークスペースで開発したアプリケーション・プログラムのデバッグ手順について説明します。

5.5.1　ロード・モジュール生成

ターゲット・デバイスにプログラムを書き込むには，C 言語やアセンブリ言語で記述されたファイルを C コンパイラなどで変換してロード・モジュールを生成します。

Multi では，「Build」メニューから「Build Top Project V850E2_MN4(MSC)_GHS.gpj」を選択すると，ロード・モジュールが生成されます。

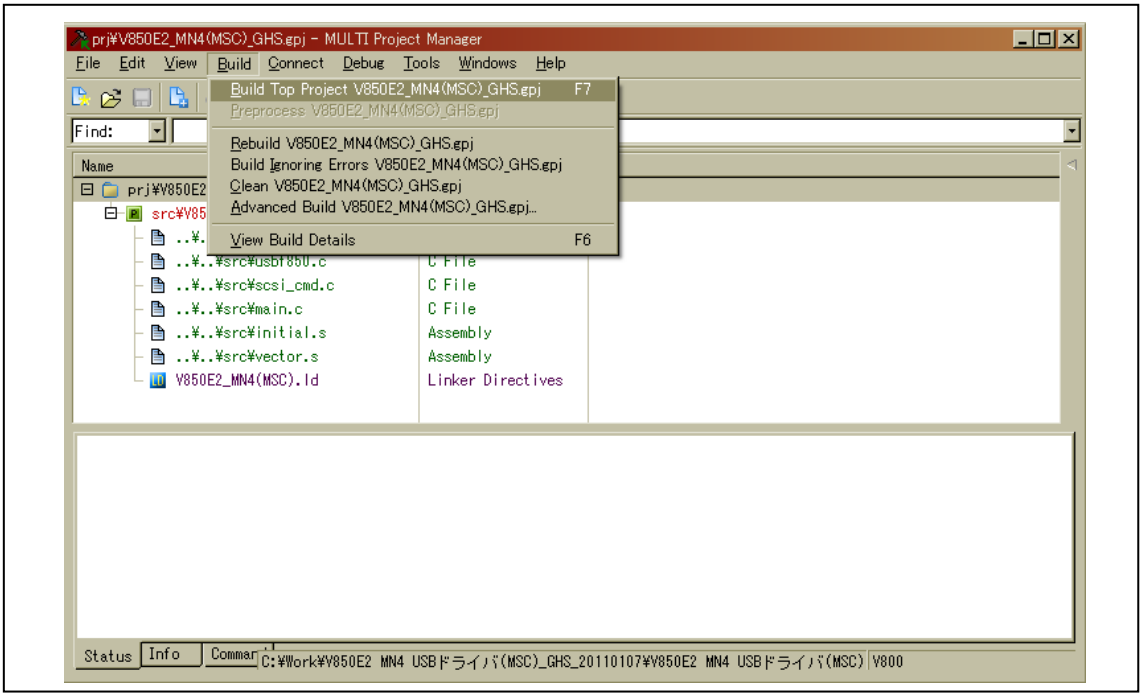


図 5－22　Build の選択

5.5.2　ロードと実行

生成したロード・モジュールをターゲットに書き込んで（ロード）実行させます。

(1) ロード・モジュールの書き込み

ここでは Multi を介して RTE-V850E2/MN4-EB-S にロード･モジュールを書き込む手順を示します。

<1>　Multi の「Connect」から「Connect」を選択し，Connection Chooser を表示します。

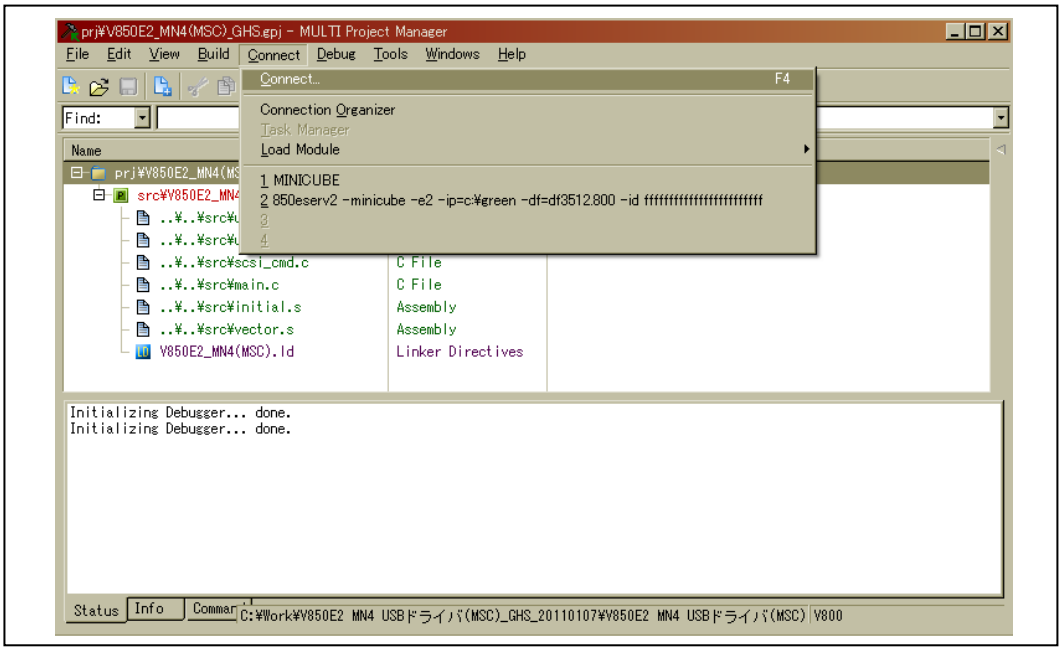


図 5－23　Connection Chooser の起動

<2>　Connection Chooser より，「5. 4. 1 ホスト環境整備」にて作成した MINICUBE 接続設定を選択して「Connect」ボタンを押下します。

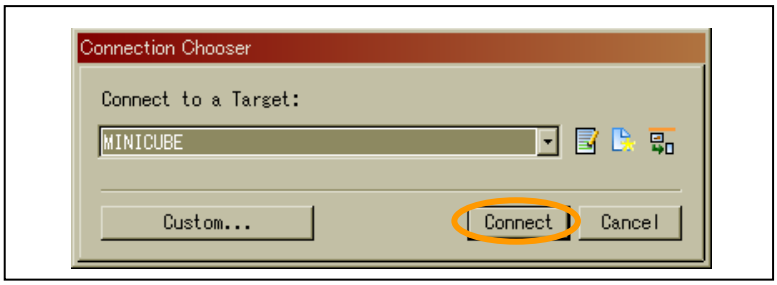


図 5－24　MINICUBE 接続設定選択

<3>　MULTI Debugger が起動するので，「File」メニューより「Debug Program」を選択し，ロード・モジュールをダウンロードします。

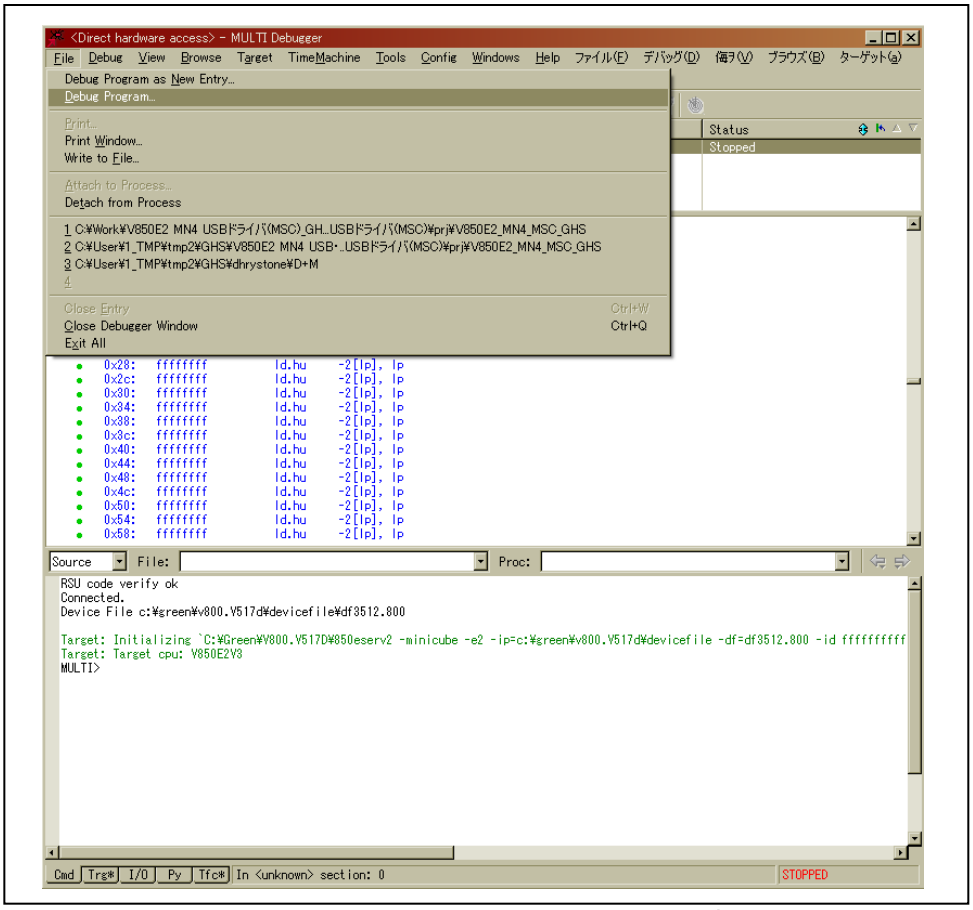


図 5−25　MULTI Debugger メニュー選択

ロード・モジュールは，「prj」フォルダに「V850E2_MN4_MSC_GHS」のファイル名で作成されているので，それを指定して「Debug」ボタンを押下します。

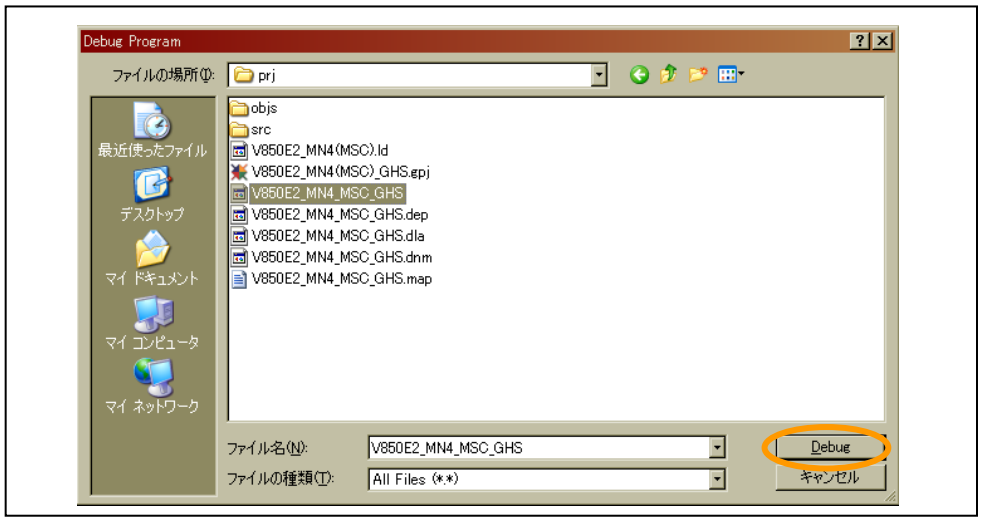


図 5−26　ロード・モジュール選択

(2) プログラムの実行

MULTI Debugger の  ボタンを押下します。または「Debug」メニューから「Go on Selected Items」を選択します。

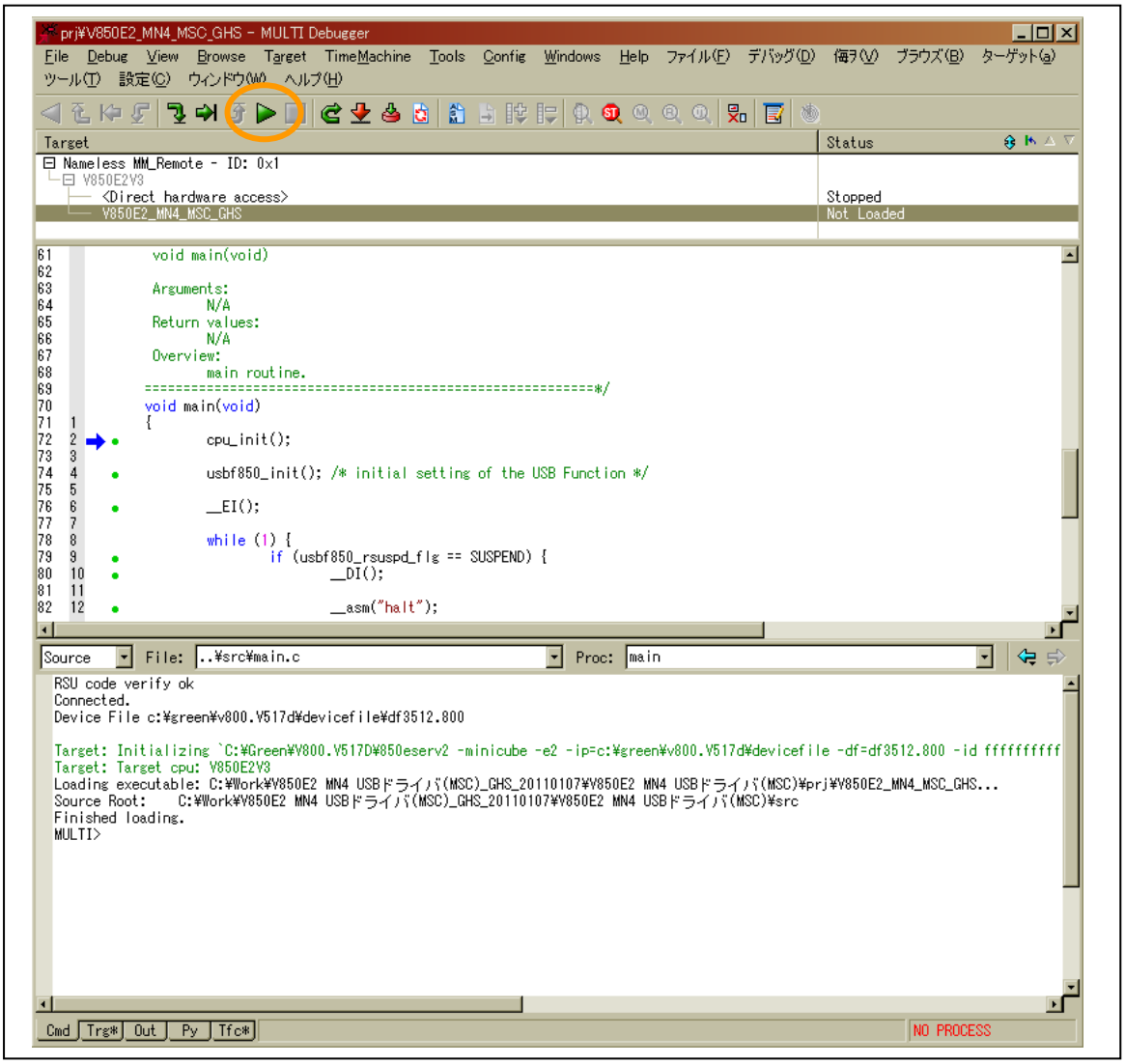


図 5－27　実行開始

5.6　IAR Embedded Workbench環境設定

ここでは，「5. 1　開発環境」に示した製品構成の中で，IAR Embedded Workbench を用いた開発やデバッグを行うための準備について説明します。

5.6.1　ホスト環境整備

ホスト・マシン上に専用のワークスペースを作成します。

(1) IAR Embedded Workbench 統合開発ツールのインストール

IAR Embedded Workbench をインストールします。詳細は IAR Embedded Workbench のユーザーズマニュアルを参照してください。

(2) ドライバ類の展開

サンプル・ドライバの提供ファイル一式を，フォルダ構成を変えずに任意のディレクトリに格納します。

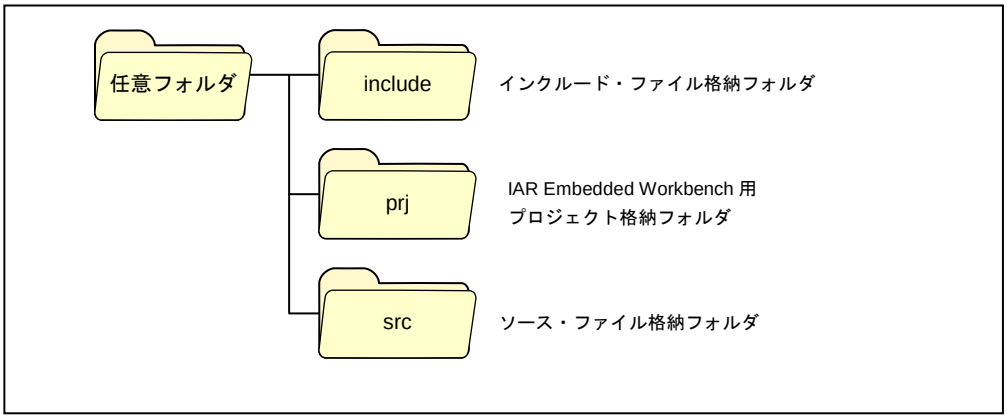


図 5－28　サンプル・ドライバのフォルダ構成（IAR Embedded Workbench 版）

(3) デバイス・ファイルのインストール

IAR Embedded Workbench 用 V850E2/MN4 用のデバイス・ファイルを，IAR Embedded Workbench インストールフォルダにコピーします。

例)　C:¥Program Files¥IAR Systems¥Embedded Workbench 5.4_0¥v850¥inc

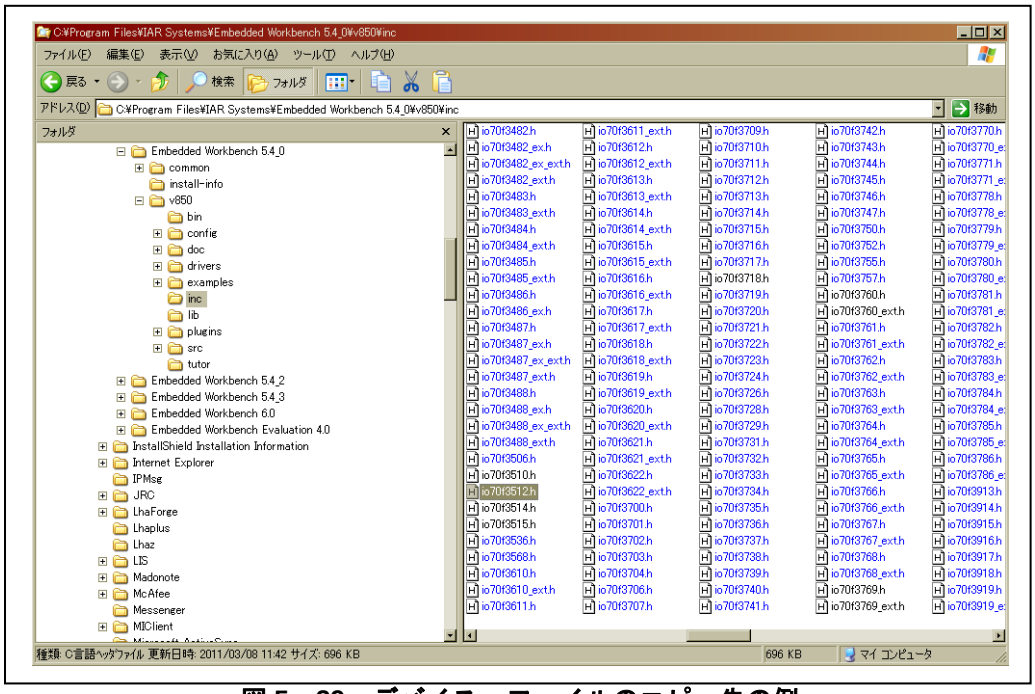


図 5－29　デバイス・ファイルのコピー先の例

(4) IAR Embedded Workbench の起動

エクスプローラから，サンプル・ドライバに同梱されている「V850E2_MN4(MSC)_IAR.eww」の IAR IDE Workspace を選択し，IAR Embedded Workbench を起動します。

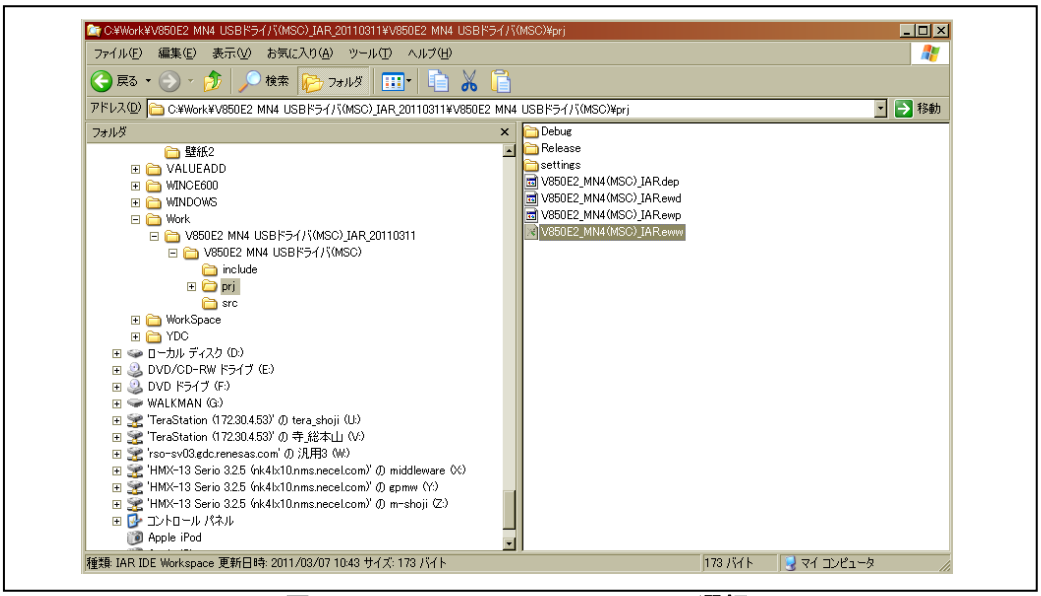


図 5－30　IAR IDE Workspace の選択

(5) デバッグ・ツールの設定

ここでは，デバッグ・ツールとして MINICUBE を使用する場合の手順を示します。

<1> IAR Embedded Workbench のプロジェクトツリーで「V850E2/MN4(MSC)_IAR-Release（または Debug）」にフォーカスを当てて右クリックを押下し「Option」を選択し，Connection Chooser を表示します。

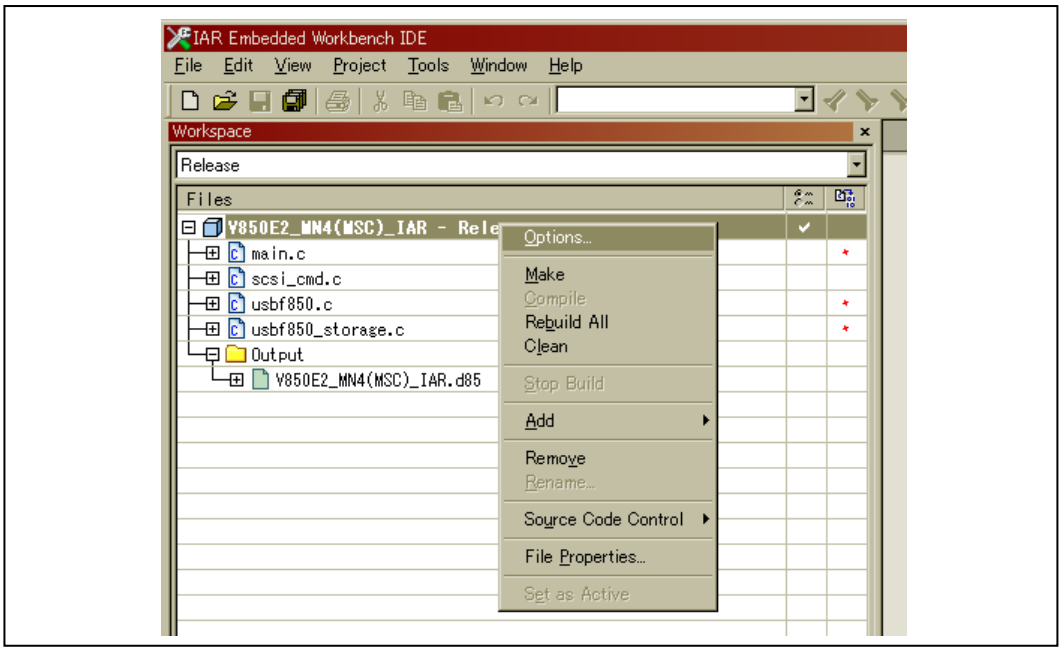


図 5－31　Option の選択

<2> Option for node “V850E2_MN4(MSC)_IAR”ダイアログから，「Category」で「Debugger」を選択します。

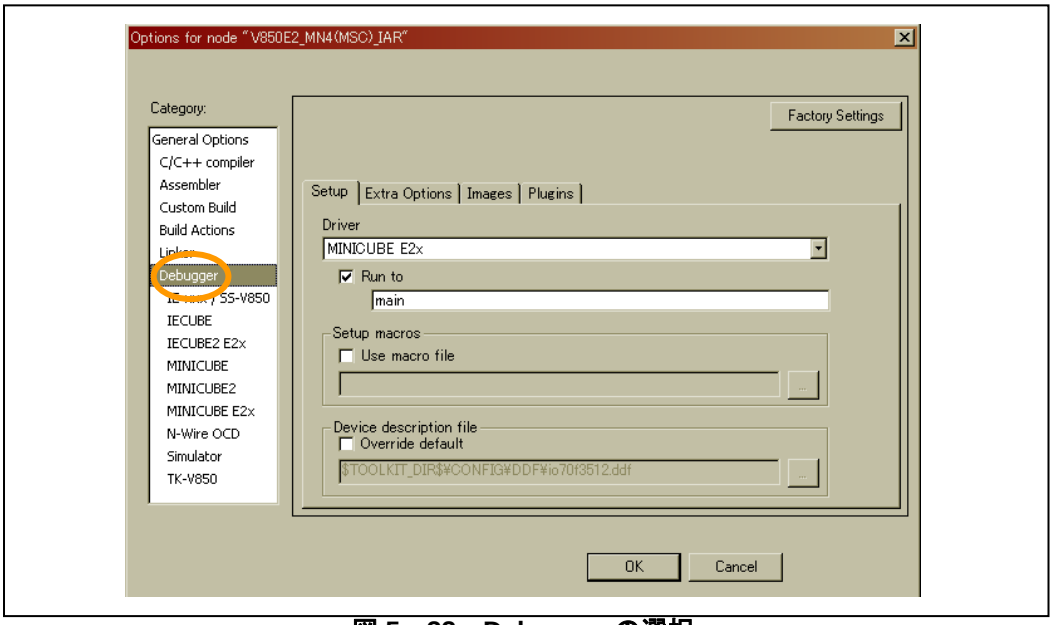


図 5－32　Debugger の選択

<3>　「Setup」タブの Driver 欄で「MINICUBE E2x」を選択して「OK」を押下します 。

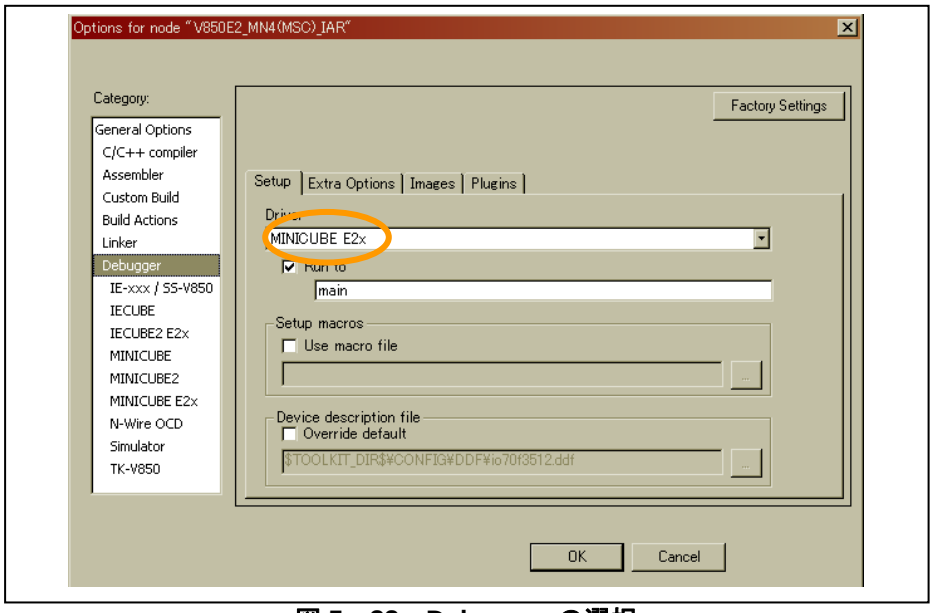


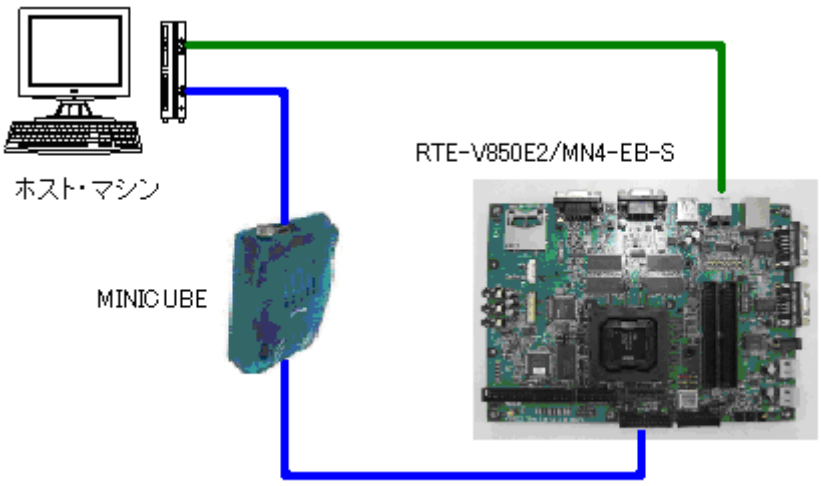
図 5－33　Debugger の選択

5.6.2　ターゲット環境整備

デバッグに使用するターゲット・デバイスを接続します。接続方法は CubeSuite/Multi/IAR Embedded Workbench 共に同じです。

(1) デバッグ・ポートの接続

RTE-V850E2/MN4-EB-S とホスト・マシンを接続します。RTE-V850E2/MN4-EB-S とホスト・マシンをデバッグ用に MINICUBE で接続します。また RTE-V850E2/MN4-EB-S の USB B タイプコネクタとホスト・マシンの USB コネクタを MSC 用に接続します。



（備考）RTE-V850E2/MN4-EB-S の外形やポートについては，「7. スタータ・キット概説」を参照してください。

図 5－34　RTE-V850E2/MN4-EB-S の接続

(2) ホスト・ドライバのインストール

USB B コネクタを使用してホスト・マシンに接続するには，ドライバをインストールする必要があります。

USB　B コネクタとの接続に使用するドライバは，Windows 標準のマス・ストレージ・クラス用ホスト・ドライバを使用します。詳細は「5. 8　動作確認」を参照してください。

5.7　IAR Embedded Workbench環境デバッグ

ここでは,「5. 6　IAR Embedded Workbench 環境設定」に示したワークスペースで開発したアプリケーション・プログラムのデバッグ手順について説明します。

5.7.1　ロード・モジュール生成

ターゲット・デバイスにプログラムを書き込むには，C 言語やアセンブリ言語で記述されたファイルを C コンパイラなどで変換してロード・モジュールを生成します。
IAR Embedded Workbench では，「Project」メニューから「Rebuild all」を選択すると，ロード・モジュールが生成されます。

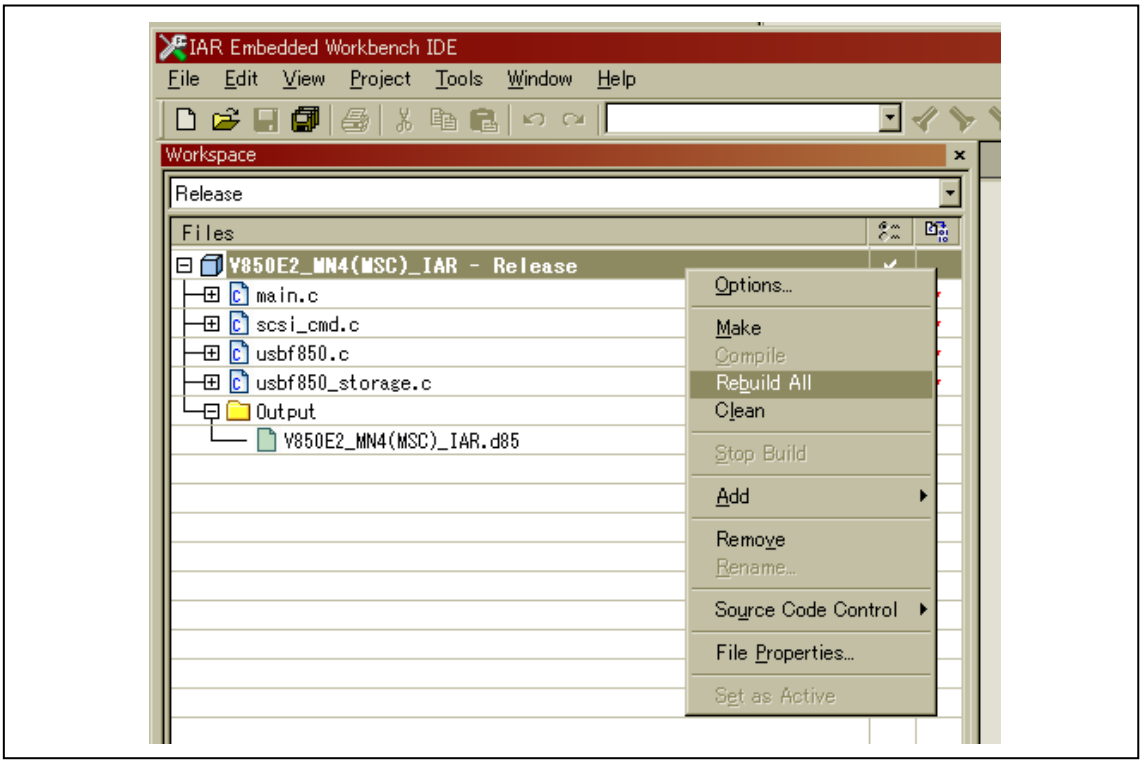


図 5−35　Rebuild all の選択

5.7.2　ロードと実行

生成したロード・モジュールをターゲットに書き込んで（ロード）実行させます。

(1) ロード・モジュールの書き込み

ここでは IAR Embedded Workbench を介して RTE-V850E2/MN4-EB-S にロード・モジュールを書き込む手順を示します。

<1>　IAR Embedded Workbench の「Project」から「Download and Debug」を選択し，ターゲットにロードします。

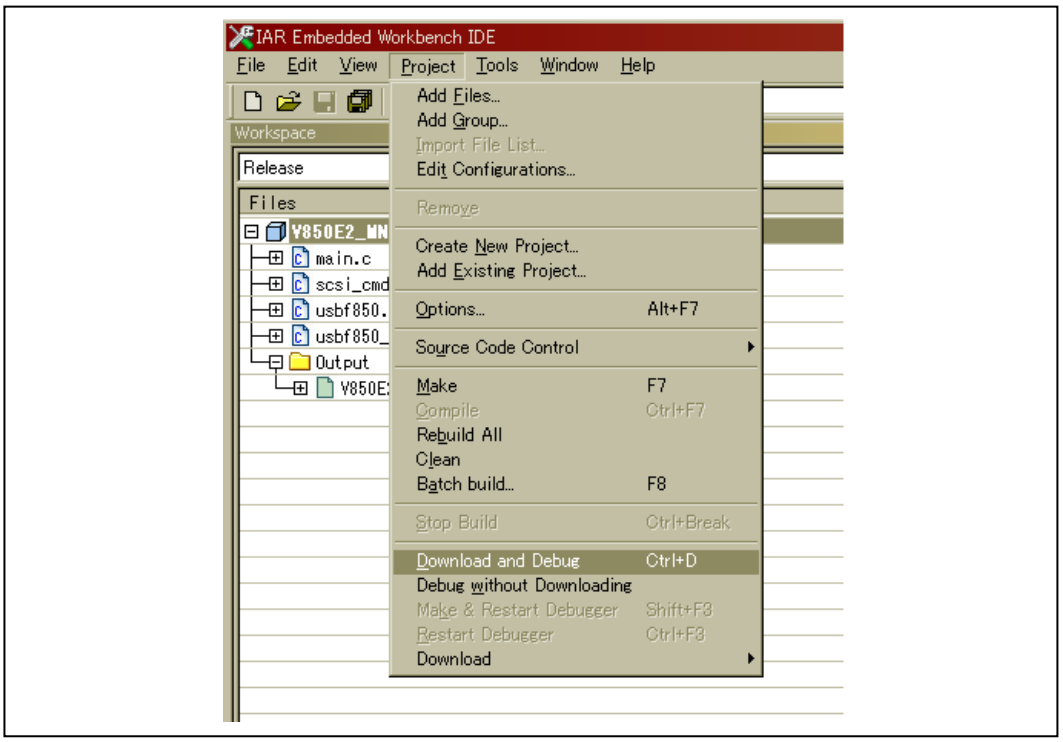


図 5－36　Debugger の起動

<2>　デバッグ・ツールを介して，ロード・モジュールのダウンロードが開始されます。

(2) プログラムの実行

IAR Embedded Workbench の  ボタンを押下します。または「Debug」メニューから「Go」を選択します。

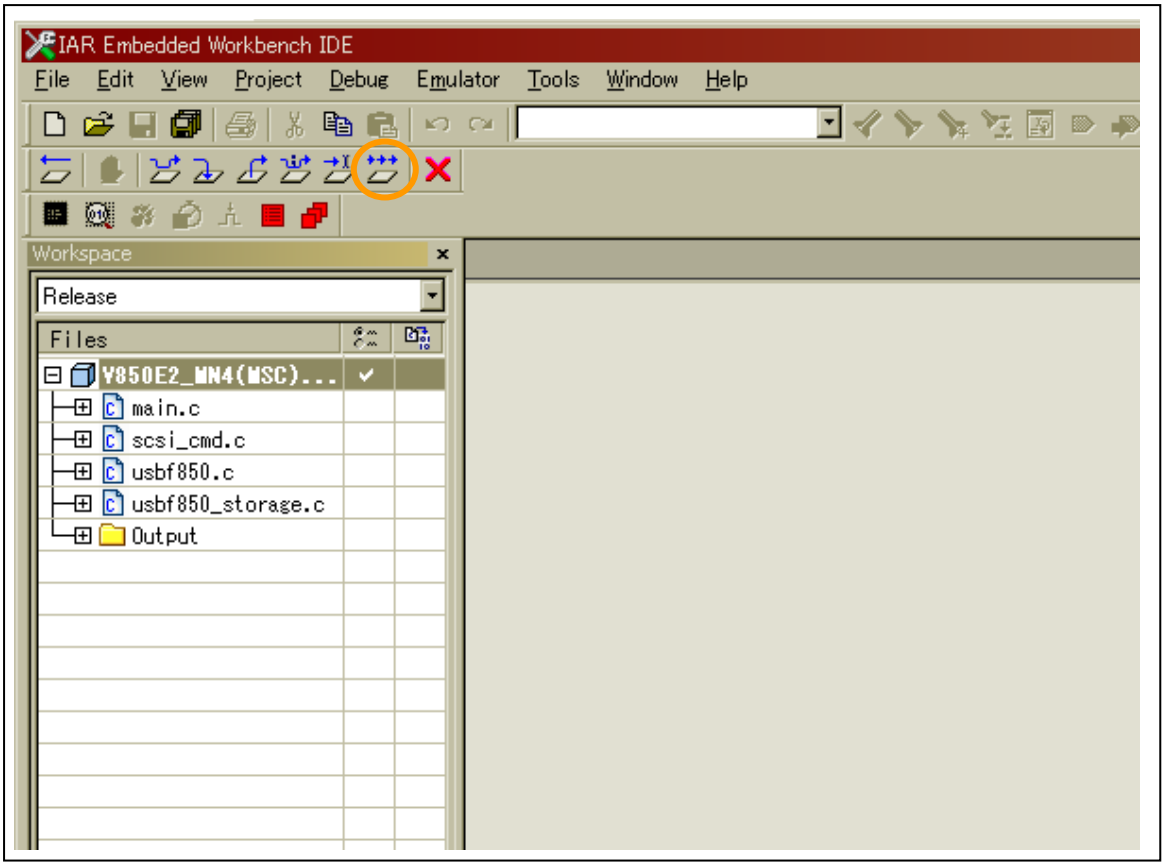


図 5－37　実行開始

5.8　動作確認

ここでは，サンプル・ドライバのプログラムを CubeSuite/Multi/IAR Embedded Workbench 環境で実行したあと，実行結果を確認する手順を示します。

(1) USB B コネクタとの接続

RTE-V850E2/MN4-BE-S の USB B コネクタとホスト・マシンの USB ポートを USB ケーブルで接続します。

(2) ホスト・ドライバのインストール

USB　B コネクタとの接続に使用するドライバは，Windows 標準のマス・ストレージ・クラス用ホスト・ドライバを使用します。サンプル・ドライバを実行している状態でホスト・マシンに接続すると，自動的にドライバがインストールされます。

(3) USB デバイスの接続確認

Windows のデバイス マネージャを開きます。「USB（Universal Serial Bus）コントローラ」のツリーを展開し，「USB 大容量記憶装置デバイス」が表示されていることを確認します。また，「ディスク ドライブ」のツリーを展開し，「Renesas StorageFncDriver USB Device」が表示されていることを確認します。

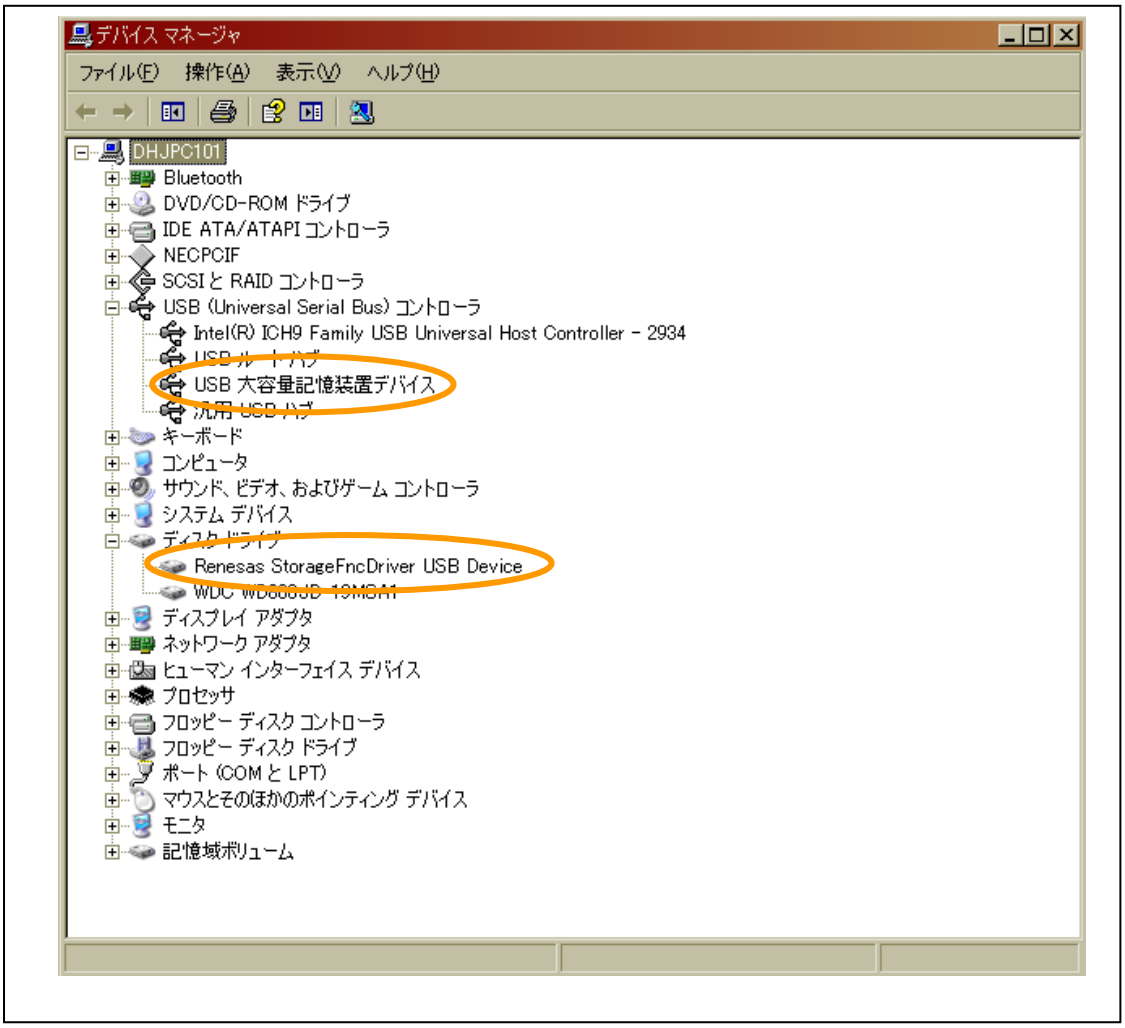


図 5－38　デバイス・マネージャ確認

(4) リムーバブル ディスクのフォーマット

Windows のマイ コンピュータを開くと「リムーバブル ディスク」が表示されます。

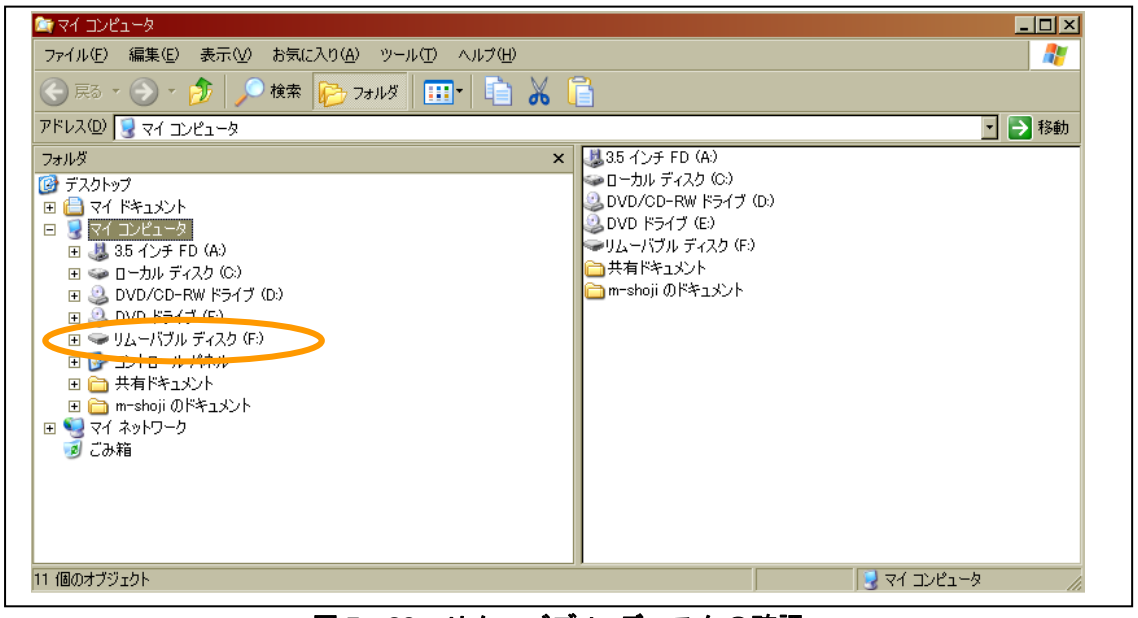


図 5−39　リムーバブル ディスクの確認

（備考）この画面例の「（F:）」は OS が自動的に割り当てるドライブ・レターです。ホスト・マシンの構成によって割り当てられるドライブ・レターが変わります。

<1>　マイ コンピュータの「リムーバブル ディスク」をクリックするとメッセージ「ディスクはフォーマットされていません」が表示されます。「はい」ボタンを押下します。

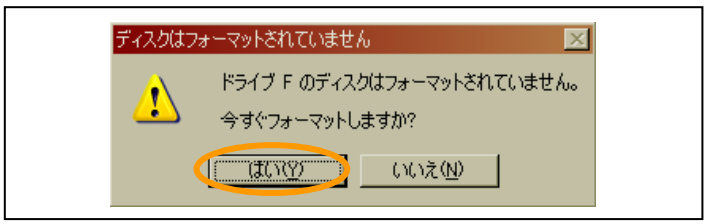


図 5−40　フォーマットの確認ダイアログ

<2>　「フォーマット」ダイアログが開きます。各項目を選択して「開始」ボタンを押下します。

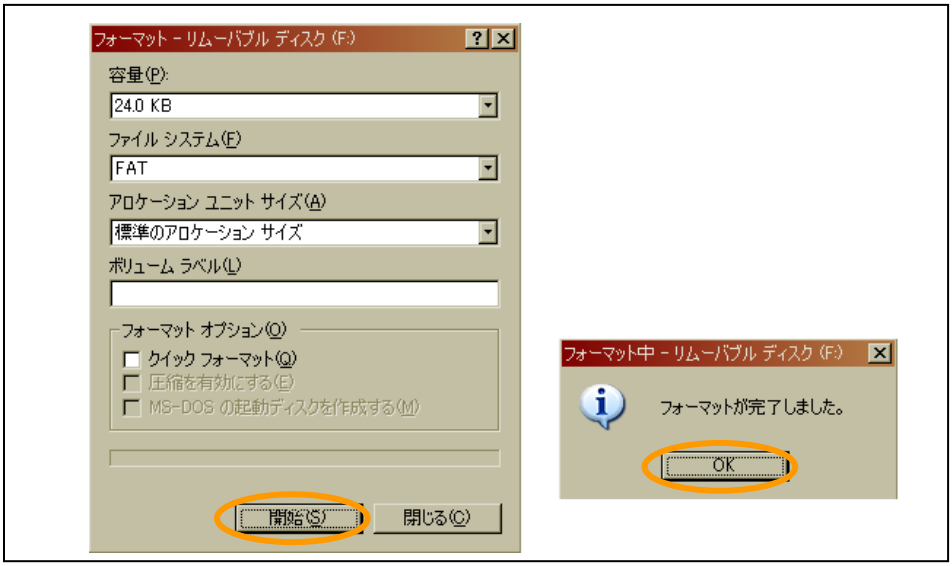


図 5−41　フォーマットメニューと完了ダイアログ

<3>　フォーマットが完了するとメッセージが表示されます。「OK」ボタンを押下します。

(5) ファイルの格納と取り出し

リムーバブル ディスクに対するファイルの書き込みと読み出しを確認します。

<1>　ローカル・ディスク上に TEST.txt ファイルと Test フォルダを用意します。

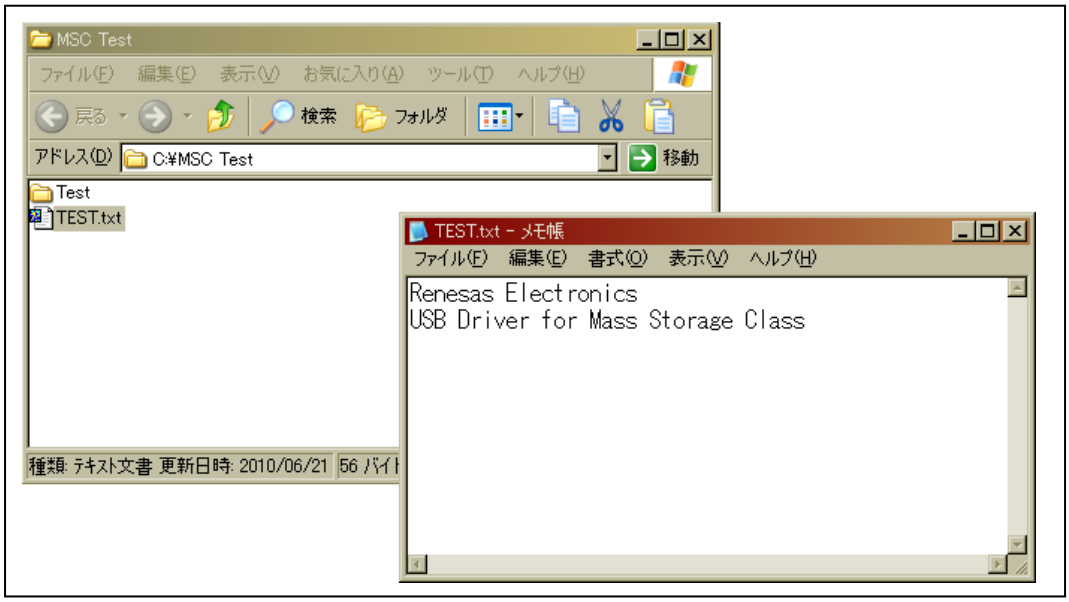


図 5−42　テスト用フォルダとテストデータファイル

<2>　マイ コンピュータのリムーバブル ディスクを開き，ローカル・ディスクからリムーバブル ディスクへ TEST.txt ファイルをコピーします。

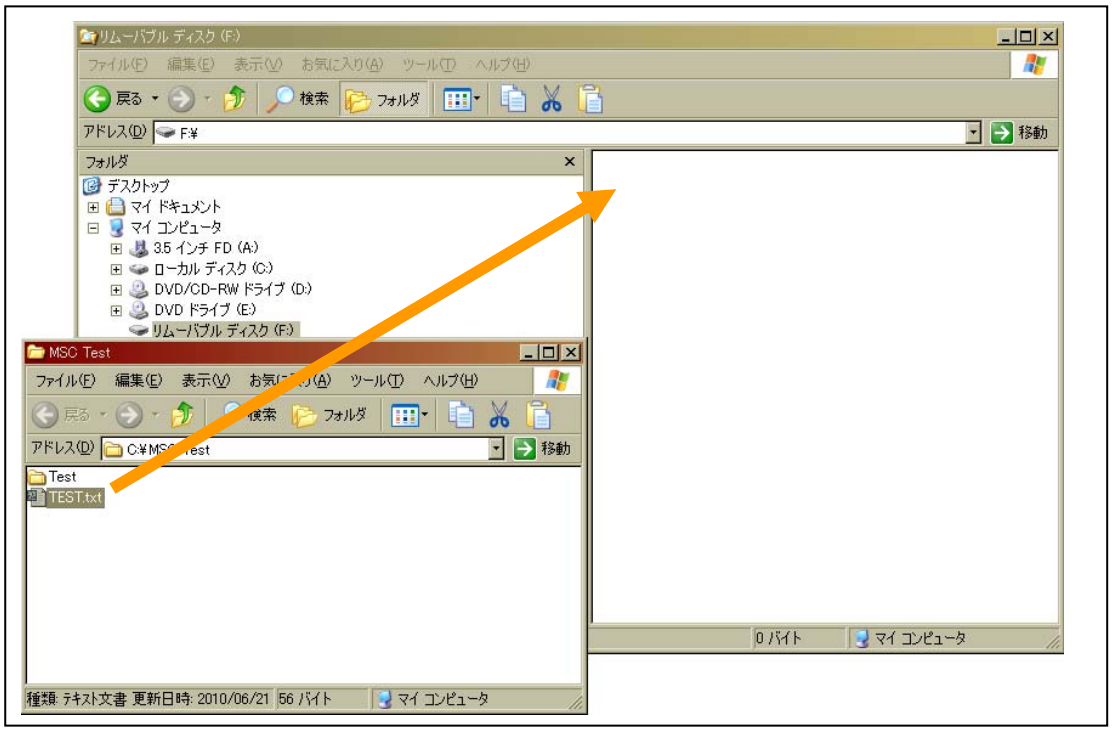


図 5−43　テストデータファイルのコピー

<3>　ローカル・ディスクの Test フォルダを開き，リムーバブル ディスクから Test フォルダへ TEST.txt ファイルをコピーします。

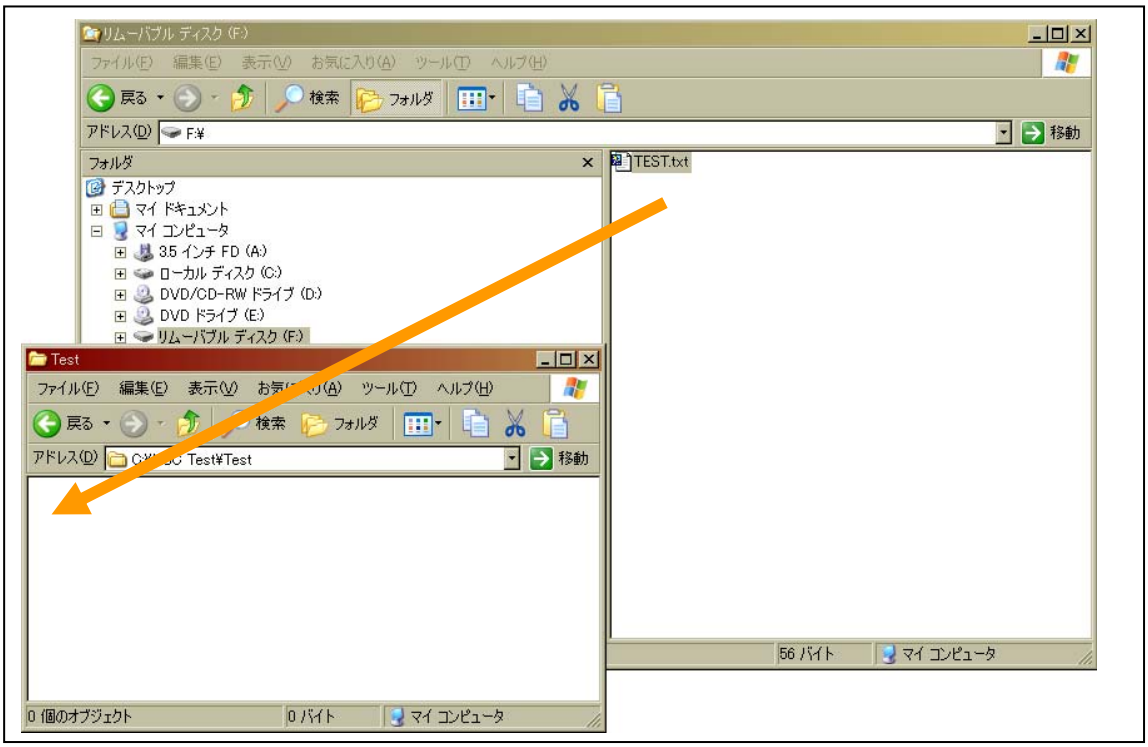


図 5−44　テストデータファイルの再コピー

<4>　Test フォルダの TEST.txt ファイルを開き，ローカル・ディスク上に用意した TEST.txt ファイルと内容が同じであることを確認します。

図 5－45　テストデータファイルの確認

（備考）　内蔵 RAM の内 24 K バイトをデータ領域として使用します。このため，保存したデータは，デバイスの電源 OFF，Reset SW の押下で初期化されます。
また，24 K バイト以上のサイズのファイルを書き込んだときの動作は保証しません。

6.　サンプル・ドライバの応用

この章では，V850E2/MN4 向け USB マス・ストレージ・クラス（MSC）用サンプル・ドライバを利用する際に，知っておいていただきたい情報について説明します。

6.1　概　要

サンプル・ドライバをカスタマイズすることで，ユーザ・システムに合ったドライバを作成します。
主に次に示す部分を必要に応じて書き換えます。

- "main.c" ファイル内のサンプル・アプリケーション部
- "usbf850.h" ファイル内の各種レジスタの設定値
- "usbf850_desc.h" ファイル内の各種ディスクリプタの内容
- "scsi_cmd.c" および "scsi.h"ファイル内の SCSI コマンド処理
- "scsi.h"ファイル内の RAM ディスク容量
- "scsi_cmd.c" ファイル内のベンダ名やプロダクト名

（備考）サンプル・ドライバのファイル構成については「2. 1. 3　サンプル・ドライバの構成」を参照してください。

6.2　カスタマイズ

ここでは，サンプル・ドライバの利用にあたり，必要に応じて書き換える部分について説明します。

6.2.1　アプリケーション部

"main.c" ファイルのメイン・ルーチン処理関数（main）には，サンプル・ドライバの利用例として簡単な処理を記述しています。実際にアプリケーションで使用する処理をこの部分に記述することで，既存の初期化処理や割り込み処理をそのまま利用できます。

```
1  /*============================================================
2   Main function
3   void main(void)
4
5   Arguments:
6       N/A
7   Return values:
8       N/A
9   Overview:
10      main routine.
11 ============================================================*/
12 void main(void)
13 {
14      cpu_init();
15
16      usbf850_init();/* initial setting of the USB Function */
17
18      EI();
19
20      while (1) {
21           if (usbf850_rsuspd_flg == SUSPEND) {
22             __DI();
23
24             __halt();
25
26               usbf850_rsuspd_flg = RESUME;
27
28             __EI();
29           }
30      }
31 }
```

図 6−1　メイン・ルーチンの記述

6.2.2　レジスタの設定

サンプル・ドライバが使用する（書き込みを行う）レジスタとその設定値は，"usbf850.h" ファイルに定義されています。これらのファイル内の値を実際のアプリケーションでの使用法に合わせて書き換えることで，サンプル・ドライバを通してターゲット・デバイスの動作を設定できます。
GHS 版には別途”df3512_800.h”ファイルが用意されており，V850E2/MN4 内蔵 I/O レジスタが定義されています。
また IAR Embedded Workbench 版には別途” io70f3512.h”ファイルが用意されており, V850E2/MN4 内臓 I/O レジスタが定義されています。

（1）"usbf850.h" ファイル
USB ファンクション・コントローラのレジスタ設定値の定義が記述されています。

（2）"df3512_800.h" ファイル（GHS 版のみ）
V850E2/MN4 内蔵 I/O レジスタの定義が記述されています。

（3）"io70f3512.h" ファイル（IAR Embedded Workbench 版のみ）
V850E2/MN4 内蔵 I/O レジスタの定義が記述されています。

6.2.3　ディスクリプタの内容

初期化処理時にサンプル・ドライバが USB ファンクション・コントローラに登録するデータ（「4. 1. 3 ディスクリプタの設定」参照）が "usbf850_desc.h" ファイルに定義されています。このファイル内の値を実際のアプリケーションでの使用法に合わせて書き換えることで，サンプル・ドライバを通してターゲット・デバイスの属性などの情報を設定できます。
また，ストリング・ディスクリプタには任意の情報を登録できます。サンプル・ドライバではシリアル番号を定義していますので，適宜書き換えてください。

```
  :
/* 0 : Language Code*/
DSTR(LangString, 2, (0x09,0x04));
/* 1 : Serial Number*/
USTR(SerialString, 12, ('0','2','0','0','0','8','0','6','5','0','1','0'));
  :
```

図 6－2　"usbf850_desc.h"のストリング・ディスクリプタ設定部

6.2.4 SCSIコマンド処理の変更

SCSI コマンドの処理は，"scsi_cmd.c"，"scsi.h" ファイルに記述されています。サポートする SCSI コマンドを追加する場合，次のように変更します。

- 処理関数を"scsi_cmd.c" ファイル内に追加
- "scsi_cmd.c" ファイルの SCSI コマンド実行処理関数（scsi_command_to_ata）に追加した関数を呼び出す case 文を追加
- "scsi.h" ファイルの Function Declaration 部に追加した関数の宣言を追加

```
INT32
scsi_command_to_ata(UINT8* ScsiCommandBuf, UINT8* pbData, INT32 lDataSize, INT32
TransFlag)
{
   long status;

   /*:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
   ** It summons processing according to the contents of the command.
   **:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::*/

   switch (ScsiCommandBuf[0]) {
/*No data Access*/
   case TEST_UNIT_READY:       /*processing of TEST UNIT READY command*/
               status = ata_test_unit_ready(TransFlag);
               return status;

   case SEEK:                /*processing of SEEK command*/
               status = ata_seek(TransFlag);
               return status;
      :
     中略
      :
   case PREVENT:              /* PREVENT/ALLOW MEDIUM REMOVAL command */
               u.clear_sense_data    = 0;
               return DEV_OK;

   default:                 /*processing of an un- supported command*/
               u.sense_data.sense_key = ILLEGAL_REQUEST;
               u.sense_data.asc      = 0x20;      // Invalid Command Operation Code
               u.sense_data.ascq     = 0x00;
               return DEV_ERROR;
   }
}
```

図 6－3 SCSI コマンド実行処理関数（scsi_command_to_ata）

6.2.5　RAMディスク容量の変更

RAM ディスクの容量は，"scsi.h" ファイルに記述されています。ALL_LOGICBLOCK（総ブロック数）と LOGICBLOCK_SIZE（ブロック・サイズ）の積が RAM ディスクの容量となります（サンプル・ドライバの場合，0x6000（=24 K バイト）に設定されています。ただし，FAT 情報などでディスク領域を消費するため，PC 上で使用できるディスク容量は設定値よりも少なくなります。

```
/*-----------------------------------------
 * data length of the table
 *-----------------------------------------*/
#define    INQUIRY_LENGTH          36       /*36Byte*/
#define    MODE_SENSE_LENGTH       24       /*24Byte*/
#define    MODE_SENSE10_LENGTH     28       /*28Byte*/
#define    MODE_SELECT_LENGTH      24       /*24Byte*/
#define    MODE_SELECT10_LENGTH    28       /*28Byte*/
#define    REQUEST_SENSE_LENGTH    18       /*18Byte*/
#define    READ_FORM_CAPA_LENGTH   20       /*20Byte*/

#define    MODE_SELECT_MIN_LEN     4        /*4Byte */

#define    ALL_LOGICBLOCK    0x30    /*number of the outline reason blocks(48)*/
#define    LOGICBLOCK_SIZE   0x200   /*1 logic block size(512Byte)*/
```

図 6－4　"scsi.h" ファイルのデータ長設定部

6.2.6　ベンダ名やプロダクト名の設定

"scsi_cmd.c" ファイルに定義されている INQUIRY コマンドの応答値を編集することで，ディスク・ドライブのベンダ名やプロダクト名として表示される名前を変更できます。

(1) INQUIRY_TABLE の記述

"scsi_cmd.c" ファイルの "INQUIRY_TABLE" は図 6－5 のように記述されています。

```
1  UINT8  INQUIRY_TABLE[INQUIRY_LENGTH]={
2      0x00,                              /*Qualifier, device type code*/
3      0x80,                              /*RMB, device type modification child*/
4      0x02,                              /*ISO Version, ECMA Version, ANSI Version*/
5      0x02,                              /*AENC, TrmIOP, response data form*/
6      0x1F,                              /*addition data length*/
7      0x00,0x00,0x00,                    /*reserved*/
8      'R','e','n','e','s','a','s',' ',   /*vender ID*/                <1>
9      'S','t','o','r','a','g','e','F','n','c','D','r','i','v','e','r',
                                          /*product ID*/               <2>
10     '0','.','0','1'                    /*Product Revision*/
11 };
```

図 6－5　"scsi_cmd.c" ファイルの "INQUIRY_TABLE" の記述

8 行目<1>でベンダ名，9 行目<2>でプロダクト名を定義しています。ベンダ名には 8 バイト（半角 8 文字），プロダクト名には 16 バイト（半角 16 文字）の文字列を指定できます。
データ送信時は各文字を ASCII コードの数値にして送信します。このため，ASCII コードでデコードできない文字を使用した場合，正常に表示されません。

(2) デバイス名の表示（デバイス一覧）

"INQUIRY_TABLE"で指定したベンダ名とプロダクト名はデバイス マネージャのディスク ドライブ名として表示されます。

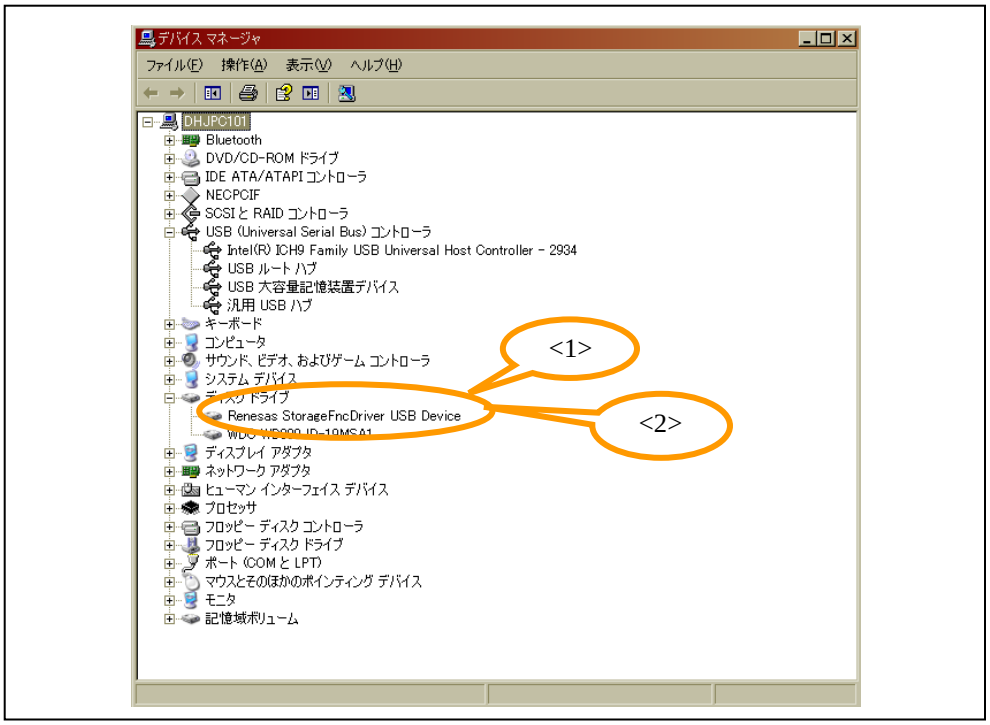


図 6－6　デバイス マネージャ確認

6.3　関数の利用

使用頻度と汎用性の高い処理が定義済みの関数として用意されているため，アプリケーションの記述を単純化でき，コード・サイズの節減にもつながります。各関数の詳細は「4. 3　関数の仕様」を参照してください。
たとえば，"usbf850_storage.c" ファイルの CBW データ受信処理部は図 6－7 のように記述されています。

```
1  void
2  usbf850_rx_cbw(void)
3  {
4              UINT8* data = (UINT8 *)&CBW_TABLE;
5              INT8 len;
6
7              if (mass_storage_reset) {
8                  /*wait "Bulk-Only Mass Storage Reset" request*/
9                  usbf850_cbw_error();
10                 return ;
11             }
12             len = UF0BO1L;
13
14             if (len != 0x1F) {
15                 return ; /*don't CBW*/
16             }
17
18             usbf850_data_receive(data, len, C_BKO1);
19
20             if (cbw_in_cbw) {
21                 /*CBW in CBW*/
22                 UF0FIC0 = (C_BKI1SC | C_BKI1CC); /*Clears EP1 buffers*/
23                 cbw_in_cbw = USB_CBW_END;
24             }
25             cbw_in_cbw = USB_CBW_PROCESS;
26             usbf850_storage_cbwchk();
27             return ;
28 }
```

図 6－7　CBW データ受信処理部

(1) マス・ストレージ・リセット・フラグ（mass_storage_reset）の監視

7 行目では，サンプル・ドライバにより設定されるフラグ（mass_storage_reset）を監視しています。このフラグが "USB_MASS_RESET_WAIT（0x01）" の場合，コマンド処理が失敗するなどして，マス・ストレージ・リセット・リクエストを待っている状態になったことを示します。

(2) データ受信処理

18 行目では，エンドポイントのデータをバッファに転送する処理を定義した関数（usbf850_data_receive）を呼び出しています。エンドポイント番号を示す "C_BKO1" はヘッダ・ファイル "usbf850.h" に定義されています。

7. スタータ・キット概説

この章では，(株)マイダス・ラボ社製の V850E2/MN4 向けスタータ・キット RTE-V850E2/MN4-EB-S について説明します。

7.1 スタータ・キット概要

RTE-V850E2/MN4-EB-S は, V850E2/MN4 を使用したアプリケーション･システムの開発を体験できるキットです。ホスト・マシンに開発ツールや USB ドライバをインストールしてこのキットを MINICUBE 接続するだけで, プログラム作成からビルド, デバッグ, 動作確認といった一連の開発フローに対応できます。

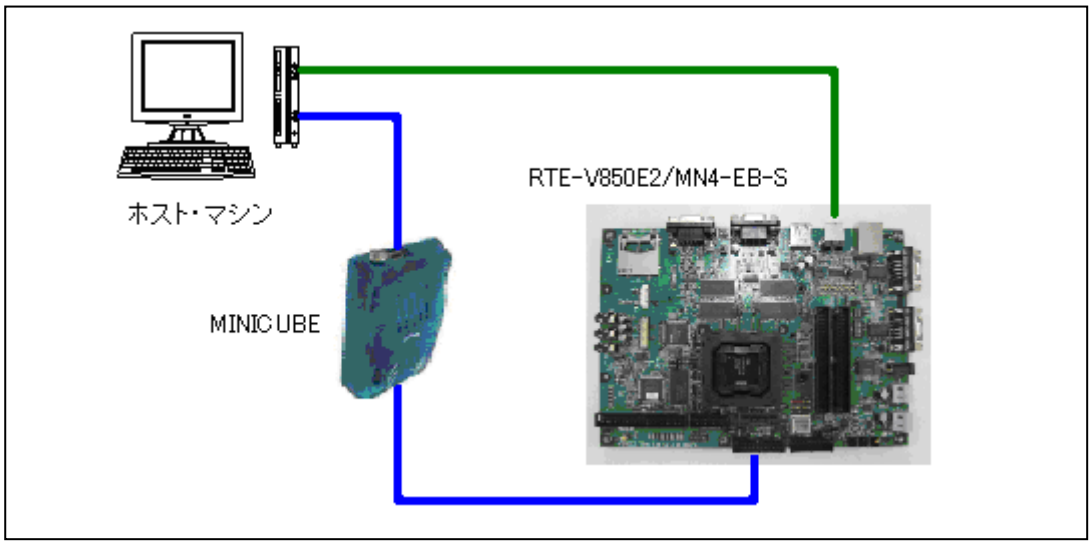


図 7－1　RTE-V850E2/MN4-EB-S の接続イメージ

7.2 スタータ・キット特徴

RTE-V850E2/MN4-EB-S には次のような特徴があります。

- 2 系統のメモリ・コントローラ, DMA，タイマ・アレイ,UART, CSI , CAN, A/D コンバータ, USB ファンクション・コントローラ,USB ホスト・コントローラ，イーサネット・コントローラなどの周辺機能を内蔵
- 入力 7 本，入出力 181 本の I/O ポートを装備
- 統合開発環境（CubeSuite/Multi/IAR Embedded Workbench）と組み合わせて効率的な開発を実現

7.3　主な仕様

RTE-V850E2/MN4-EB-S の主な仕様は次のとおりです。

○CPU	μPD70F3512（V850E2/MN4）
○動作周波数	200 MHz(PLL による 20 逓倍機能)
○インタフェース	USB コネクタ 2 基搭載（USB ホスト A タイプ×1 基，USB ファンクション B タイプ×1 基） N-Wire 用コネクタ UART2 基搭載 CAN2 基搭載 Ethernet コネクタ搭載
○対応機種	ホスト・マシン：USB インタフェース付き PC/AT 互換機 OS：Windows 2000，Windows XP
○動作電圧	5.0 V
○本体寸法	W200×D150（mm）

ホームページとサポート窓口

ルネサス エレクトロニクスホームページ
http://japan.renesas.com/

お問合せ先
http://japan.renesas.com/inquiry

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2010.06.30	—	初版発行
1.01	2011.01.14	全頁	ルネサスエレクトロニクスへの統合に伴うフォーマット改訂 5 章に GHS 版の内容を追加
1.02	2011.03.25		5 章に IAR Embedded Workbench 版の内容を追加

製品ご使用上の注意事項

ここでは、マイコン製品全体に適用する「使用上の注意事項」について説明します。個別の使用上の注意事項については、本文を参照してください。なお、本マニュアルの本文と異なる記載がある場合は、本文の記載が優先するものとします。

1. 未使用端子の処理

   【注意】未使用端子は、本文の「未使用端子の処理」に従って処理してください。

   CMOS製品の入力端子のインピーダンスは、一般に、ハイインピーダンスとなっています。未使用端子を開放状態で動作させると、誘導現象により、LSI周辺のノイズが印加され、LSI内部で貫通電流が流れたり、入力信号と認識されて誤動作を起こす恐れがあります。未使用端子は、本文「未使用端子の処理」で説明する指示に従い処理してください。

2. 電源投入時の処置

   【注意】電源投入時は，製品の状態は不定です。

   電源投入時には、LSIの内部回路の状態は不確定であり、レジスタの設定や各端子の状態は不定です。

   外部リセット端子でリセットする製品の場合、電源投入からリセットが有効になるまでの期間、端子の状態は保証できません。

   同様に、内蔵パワーオンリセット機能を使用してリセットする製品の場合、電源投入からリセットのかかる一定電圧に達するまでの期間、端子の状態は保証できません。

3. リザーブアドレスのアクセス禁止

   【注意】リザーブアドレスのアクセスを禁止します。

   アドレス領域には、将来の機能拡張用に割り付けられているリザーブアドレスがあります。これらのアドレスをアクセスしたときの動作については、保証できませんので、アクセスしないようにしてください。

4. クロックについて

   【注意】リセット時は、クロックが安定した後、リセットを解除してください。

   プログラム実行中のクロック切り替え時は、切り替え先クロックが安定した後に切り替えてください。

   リセット時、外部発振子（または外部発振回路）を用いたクロックで動作を開始するシステムでは、クロックが十分安定した後、リセットを解除してください。また、プログラムの途中で外部発振子（または外部発振回路）を用いたクロックに切り替える場合は、切り替え先のクロックが十分安定してから切り替えてください。

5. 製品間の相違について

   【注意】型名の異なる製品に変更する場合は、事前に問題ないことをご確認下さい。

   同じグループのマイコンでも型名が違うと、内部メモリ、レイアウトパターンの相違などにより、特性が異なる場合があります。型名の異なる製品に変更する場合は、製品型名ごとにシステム評価試験を実施してください。

ご注意書き

1. 本資料に記載されている内容は本資料発行時点のものであり、予告なく変更することがあります。当社製品のご購入およびご使用にあたりましては、事前に当社営業窓口で最新の情報をご確認いただきますとともに、当社ホームページなどを通じて公開される情報に常にご注意ください。
2. 本資料に記載された当社製品および技術情報の使用に関連し発生した第三者の特許権、著作権その他の知的財産権の侵害等に関し、当社は、一切その責任を負いません。当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
3. 当社製品を改造、改変、複製等しないでください。
4. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。お客様の機器の設計において、回路、ソフトウェアおよびこれらに関連する情報を使用する場合には、お客様の責任において行ってください。これらの使用に起因しお客様または第三者に生じた損害に関し、当社は、一切その責任を負いません。
5. 輸出に際しては、「外国為替及び外国貿易法」その他輸出関連法令を遵守し、かかる法令の定めるところにより必要な手続を行ってください。本資料に記載されている当社製品および技術を大量破壊兵器の開発等の目的、軍事利用の目的その他軍事用途の目的で使用しないでください。また、当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器に使用することができません。
6. 本資料に記載されている情報は、正確を期すため慎重に作成したものですが、誤りがないことを保証するものではありません。万一、本資料に記載されている情報の誤りに起因する損害がお客様に生じた場合においても、当社は、一切その責任を負いません。
7. 当社は、当社製品の品質水準を「標準水準」、「高品質水準」および「特定水準」に分類しております。また、各品質水準は、以下に示す用途に製品が使われることを意図しておりますので、当社製品の品質水準をご確認ください。お客様は、当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途に当社製品を使用することができません。また、お客様は、当社の文書による事前の承諾を得ることなく、意図されていない用途に当社製品を使用することができません。当社の文書による事前の承諾を得ることなく、「特定水準」に分類された用途または意図されていない用途に当社製品を使用したことによりお客様または第三者に生じた損害等に関し、当社は、一切その責任を負いません。なお、当社製品のデータ・シート、データ・ブック等の資料で特に品質水準の表示がない場合は、標準水準製品であることを表します。

   標準水準： コンピュータ、OA機器、通信機器、計測機器、AV機器、家電、工作機械、パーソナル機器、産業用ロボット

   高品質水準： 輸送機器（自動車、電車、船舶等）、交通用信号機器、防災・防犯装置、各種安全装置、生命維持を目的として設計されていない医療機器（厚生労働省定義の管理医療機器に相当）

   特定水準： 航空機器、航空宇宙機器、海底中継機器、原子力制御システム、生命維持のための医療機器（生命維持装置、人体に埋め込み使用するもの、治療行為（患部切り出し等）を行うもの、その他直接人命に影響を与えるもの）（厚生労働省定義の高度管理医療機器に相当）またはシステム等
8. 本資料に記載された当社製品のご使用につき、特に、最大定格、動作電源電圧範囲、放熱特性、実装条件その他諸条件につきましては、当社保証範囲内でご使用ください。当社保証範囲を超えて当社製品をご使用された場合の故障および事故につきましては、当社は、一切その責任を負いません。
9. 当社は、当社製品の品質および信頼性の向上に努めておりますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は耐放射線設計については行っておりません。当社製品の故障または誤動作が生じた場合も、人身事故、火災事故、社会的損害などを生じさせないようお客様の責任において冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、機器またはシステムとしての出荷保証をお願いいたします。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様が製造された最終の機器・システムとしての安全検証をお願いいたします。
10. 当社製品の環境適合性等、詳細につきましては製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制するRoHS指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。お客様がかかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。
11. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを固くお断りいたします。
12. 本資料に関する詳細についてのお問い合わせその他お気付きの点等がございましたら当社営業窓口までご照会ください。

注1. 本資料において使用されている「当社」とは、ルネサス エレクトロニクス株式会社およびルネサス エレクトロニクス株式会社がその総株主の議決権の過半数を直接または間接に保有する会社をいいます。

注2. 本資料において使用されている「当社製品」とは、注1において定義された当社の開発、製造製品をいいます。

ルネサス エレクトロニクス株式会社

■営業お問合せ窓口 http://www.renesas.com

※営業お問合せ窓口の住所・電話番号は変更になることがあります。最新情報につきましては、弊社ホームページをご覧ください。

ルネサス エレクトロニクス販売株式会社　〒100-0004　千代田区大手町2-6-2（日本ビル） (03)5201-5307

■技術的なお問合せおよび資料のご請求は下記へどうぞ。
総合お問合せ窓口：http://japan.renesas.com/inquiry

Colophon 1.0