V850E2/ML4 マイクロコンピュータ
USB マルチファンクション動作例

R01AN1037JJ0100
Rev.1.00
2012.03.15

要　旨

本アプリケーションノートは，V850E2/ML4 マイクロコントローラに内蔵の USB ファンクション・コントローラの使用方法として,マス・ストレージ・クラス(以下, MSC)とコミュニケーション・デバイス・クラス(以下, CDC)の両方に対応したサンプルプログラムの動作例について説明します。

本アプリケーションノートの内容とソフトウエアは, USB ファンクションモジュールの応用例を説明するものでその内容を保障するものではありません。

動作確認デバイス

V850E2/ML4(uPD70F4022)

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/ML4 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/ML4 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したサンプルプログラム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
- USB マルチファンクションの概要仕様
- サンプルプログラムの仕様
- 開発環境(CubeSuite+)

1.5　読 み 方

このアプリケーションノートの読者には，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/ML4 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/ML4 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/ML4 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。

— MSC ドライバ、CDC ドライバの仕様を知りたいとき
　→　各ドライバのアプリケーションノート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/ML4 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法として, MSC と CDC の両方対応したサンプルプログラムについて説明します。本プログラムではコントロール転送，バルク転送，インタラプト転送，及び USB マルチファンクション(MSC クラスコマンド, CDC クラスコマンド)対応処理を行います。

この章では，サンプルプログラム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

サンプルプログラムの制御の対象である V850E2/ML4 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/ML4 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）
- 内部クロックと外部クロックを選択可能 (注 1)
  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 1) サンプルプログラムでは内部クロックを選択します。

2.1.2　サンプルプログラムの特徴

V850E2/ML4 マイクロコントローラ向けマルチファンクションのサンプルプログラムには，次のような特徴があります。機能や動作の詳細は第 4 章　サンプルプログラムの仕様を参照してください。

- マス・ストレージ・クラスとコミュニケーション・デバイス・クラスとの複合デバイスとして認識
- セルフ・パワード・デバイスとして動作
- 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
- USB コミュニケーション・デバイス・クラス Ver.1.1 の Abstract Control Mode1 に準拠
- 仮想 COM デバイスとして動作
- 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 10.0 K バイト
  RAM：約 26.0 K バイト(注 2)

(注 2) RAM（約 26.0 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　サンプルプログラムの構成

サンプルプログラムは CubeSuite+版が用意されています。サンプルプログラムは次のようなファイルで構成されています。

表 2-2　サンプルプログラムのファイル構成

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_communication.c	CDC 固有処理
	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_communication.h	usbf850_communication.c 関数プロトタイプ宣言
	usbf850_types.h	ユーザ型宣言
	reg_v850e2ML4.h	USB ファンクション用レジスタ定義
inf ファイル	XXX_CDC.inf	CDC の Windows®用 INF ファイル XXX の部分は各マイコン名が入ります：ML4

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。

2.2　V850E2/ML4 マイクロコントローラ

サンプルプログラムの制御の対象である V850E2/ML4 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

サンプルプログラムは，次に示す製品に適用できます。

表 2-3　V850E2/ML4 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	外部 注3	
V850E2/ML4	μPD70F4021	768KB	内蔵 RAM 64 KB ＋共有メモリ 64KB	Host / Function	122	29	V850E2/ML4 ユーザーズマニュアル ハードウェア編 (R01UH0262JJ)
	μPD70F4022	1MB	内蔵 RAM 64 KB ＋共有メモリ 64KB	Host / Function	122	29	

(注 3)　ノンマスカブル割り込みを含みます。

2.2.2 特 徴

V850E2/ML4 には，主に次のような特徴があります。

- 内蔵メモリ
  RAM： 64K バイト
  フラッシュ・メモリ：768K バイト(μPD70F4021)，1 M バイト(μPD70F4022)

- 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）

- 外部バス・インタフェース
  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 : 2ch
  クロック同期式シリアル・インタフェース CSI : 2ch
  アシンクロナス・シリアル・インタフェース UART(FIFO) : 2ch
  クロック同期式シリアル・インタフェース CSI(FIFO) : 2ch
  I2C : 2 チャネル
  CAN : 1 チャネル
  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

- DMA コントローラ
  DMA コントローラ : 8 チャネル
  DTS : 最大 128 チャネル

3. USBの概要

この章では，サンプルプログラムが準拠する 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 4)。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 4) 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。
本プログラムのマルチファンクション処理は，マス・ストレージ・クラス(MSC)とコミュニケーション・デバイス・クラスを使用します。各クラスのデバイスドライバについては，各アプリケーションノートを参照してください。

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。
サンプルプログラムは，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。

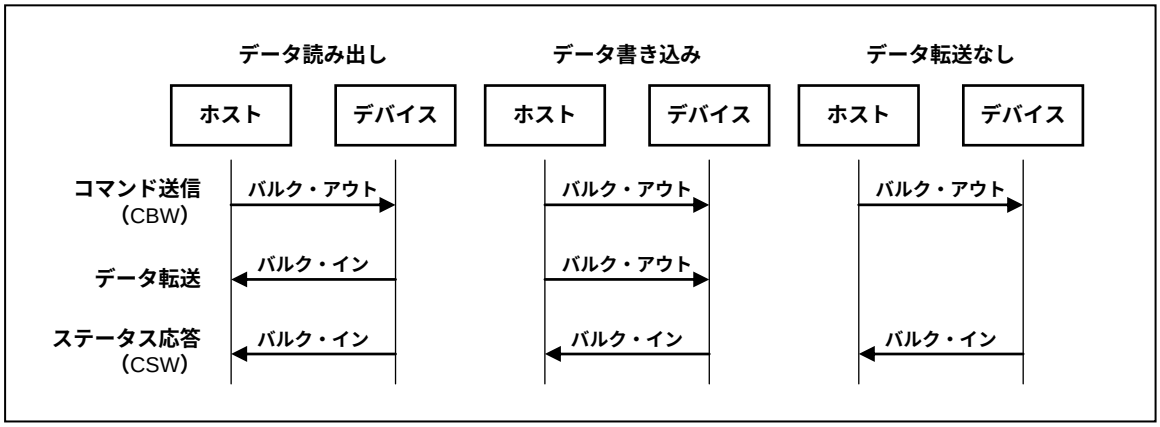


図 3−1　データ転送フロー

(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(MSC)

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。サンプルプログラムが対応している SCSI コマンドについては「4. 1.5　SCSI コマンドへの対応」を参照してください。

3.3.3　コミュニケーション・デバイス・クラス(CDC)

コミュニケーション・デバイス・クラス（CDC）は，ホスト・コンピュータに接続する通信機器のためのクラスで，モデム，FAX，ネットワーク・カードなどが対象となります。最近では PC に RS-232C インタフェースが搭載されなくなっていることから，PC と UART 通信を行う際の USB シリアル変換を実現するデバイスに使われることが多くなっています。なお，CDC には実装する機器によっていくつかのモデルが定義されています。サンプルプログラムはこのなかの Abstract Control Model を使用しています。

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。
サンプルプログラムが対応するリクエストについては，「4. 1.3　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。

CDC の Abstract Control Model に対応したクラス・リクエストへの応答処理を実装しています。応答可能なリクエストは次のとおりです。

- Send Encapsulated Command
  コミュニケーション・クラス・インタフェースの制御プロトコルのフォーマットでコマンドを発行するためのリクエストです。
- Get Encapsulated Response
  コミュニケーション・クラス・インタフェースの制御プロトコルのフォーマットで応答を要求するためのリクエストです。
- Set Line Coding
  シリアル通信の通信フォーマットを指定するためのリクエストです。
- Get Line Coding
  デバイス側の現在の通信フォーマット設定を取得するためのリクエストです。
- Set Control Line State
  RS-232/V.24 形式の制御信号の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 6 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・アソシエーション・ディスクリプタ
  複数のインタフェース・ディスクリプタを用いて一つのファンクションを構成することができます。ファンクションを構成するインタフェースの情報(最初のインタフェース認識番号，インタフェース数，クラスなど)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。

- 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･アソシエーション，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・アソシエーション・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
bFirstInterface	1	このファンクションを構成する最初のインタフェース番号
bInterfaceCount	1	このファンクションが持つインタフェースの数
bFunctionClass	1	このファンクションのクラス・コード
bFunctionSubClass	1	このファンクションのサブクラス・コード
bFunctionProtocol	1	このファンクションのプロトコル・コード
iFunction	1	このファンク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

表 3-11　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 3-12　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 3-13　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. サンプルプログラムの仕様

この章では，MSC と CDC の両方に対応した V850E2/ML4 マイクロコントローラ向けマルチファンクションのサンプルプログラム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。各クラスのサンプルプログラムについては，各アプリケーションノートを参照してください。

4.1 概　要

4.1.1 USBマルチファンクションの概要

USB マルチファンクションとは、複数のファンクション機能を一つのデバイスに実装したものです。本サンプルプログラムの USB マルチファンクション処理は、マス・ストレージ・クラス(MSC)とコミュニケーション・デバイス・クラス(CDC)の 2 つに対応しています。つまり、2 つのファンクションを持つデバイスとして PC に認識されます。更に CDC は、コントロールとデータの 2 つのインタフェースを持ったファンクションです。MSC と合わせて 3 つのインタフェースを持ったデバイスとなります。

表 4-1　USB マルチファンクションの構成

ファンクション	インタフェース	クラス
Function0	interface0	MSC(Bulk-Only)
Function1	interface1	CDC(Control)
	interface2	CDC(Data)

インタフェース・ディスクリプタを使用して、各インタフェースの識別番号をホスト PC へ通知します。また、インタフェース・アソシエーション・ディスクリプタ(IAD)を使用して、interface1 と interface2 が一つのファンクションであることをホスト PC へ通知します。
IAD を持つデバイスは、デバイス・ディスクリプタで下表のクラスを設定する必要があります。

表 4-2　デバイス・ディスクリプタのクラス

Class	SubClass	Protocol	説明
0xEF	0x02	0x01	IAD を持つデバイス

表 4-3　インタフェース・ディスクリプタのクラス

インタフェース	Class	SubClass	Protocol	説明
interface0	0x08	0x06	0x50	MSC Bulk-Only
inteface1	0x02	0x02	0x00	CDC Control
interface2	0x02	0x0A	0x00	CDC Data

4.1.2 機　能

サンプルプログラム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。

CDC を用いたサンプル・アプリケーションとして、バルク・アウト転送（受信）用エンドポイントにあるデータを読み出し，読み出したデータをそのままバルク・イン転送（送信）用エンドポイントに書き込みます。

(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/ML4 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。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。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(MSC)

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. 5　SCSI コマンドへの対応」を参照してください。

4.1.3　リクエストへの対応

表 4−4 にハードウエア（V850E2/ML4）およびファームウエア（サンプルプログラム）で定義されている USB リクエストを示します。

表 4-4　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(MSC)									
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
クラス・リクエスト(CDC)									
SEND_ENCAPSULATED_COMMAND	0x21	0x00	0x00	0x00	0xXX	0xXX	0xXX	0xXX	FW 応答
GET_ENCAPSULATED_RESPONSE	0xA1	0x01	0x00	0x00	0xXX	0xXX	0xXX	0xXX	FW 応答
SET_LINE_CODING	0x21	0x20	0x00	0x00	0xXX	0xXX	0xXX	0xXX	FW 応答
GET_LINE_CODING	0xA1	0x21	0x00	0x00	0xXX	0xXX	0xXX	0xXX	FW 応答
SET_CONTROL_LINE_STATE	0xA1	0x22	0x00	0x00	0xXX	0xXX	0xXX	0xXX	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/ML4）
FW：ファームウエア（サンプルプログラム）
0xXX：不定値

(1) 標準リクエスト

V850E2/ML4 が自動的に応答しないリクエストに対し，サンプルプログラム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，サンプルプログラム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，サンプルプログラムは STALL 応答を返します。

(2) クラス・リクエスト(MSC)

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，サンプルプログラムは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。
サンプルプログラムは 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 4-5　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。
サンプルプログラムは MASS_STORAGE_RESET リクエストを受信すると，サンプルプログラムが使用している 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 4-6　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) クラス・リクエスト(CDC)

USB コミュニケーション・デバイス・クラス（CDC）のクラス・リクエストに対し，サンプルプログラムは次のような応答処理を行います。

(a) SendEncapsulatedCommand

CDC インタフェースの制御プロトコルのフォーマットでコマンドを発行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，サンプルプログラムはリクエストに付随するデータを

取り込み，送信処理（バルク・イン転送）を行います。

(b) GetEncapsulatedResponse

CDC インタフェースの制御プロトコルのフォーマットで応答を要求するためのリクエストです。
サンプルプログラムは現在，このリクエストをサポートしていません。

(c) SetLineCoding

シリアル通信の通信フォーマットを指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，サンプルプログラムはリクエストに付随するデータを取り込んで通信レートなどを設定し，NULL パケットの送信処理（コントロール・リード転送）を行います。

(d) GetLineCoding

デバイス側の現在の通信フォーマット設定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，サンプルプログラムは通信レートなどの設定を読み出し，送信処理（コントロール・リード転送）を行います。

(e) SetControlLineState

RS-232/V.24 形式の制御信号のためのリクエストです。
このリクエストを受信すると，サンプルプログラムは NULL パケットの送信処理（コントロール・リード転送）を行います。

(4) 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，サンプルプログラムは STALL 応答を返します。

4.1.4 ディスクリプタの設定

サンプルプログラム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 4-7 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	0xEF	インタフェース・アソシエーション・ディスクリプタを持つクラス・コード
bDeviceSubClass	1	0x02	
bDeviceProtocol	1	0x01	
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	0x0218	プロダクト ID：V850E2/ML4
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	0x02	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：2
iSerialNumber	1	0x03	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：3
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 4-8　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	0x004F	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：79 バイト
bNumInterfaces	1	0x03	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：3
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	0xC0	このコンフィギュレーションの特徴: セルフ・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・アソシエーション・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。
本プログラムでは、MSC は一つのインタフェース(interface0)を持ち、CDC はコントロール(interface1)とデータ(interface2)の 2 つのインタフェースを持ちます。CDC が 2 つで一つのファンクションであることをインタフェース・アソシエーション・ディスクリプタに記述します。

表 4-9　インタフェース・アソシエーション・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	0x08	ディスクリプタのサイズ：8 バイト
bDescriptorType	1	0x0b	ディスクリプタの種類：インタフェース・アソシエーション
bFirstInterface	1	0x01	このファンクションを構成する最初のインタフェース番号:1
bInterfaceCount	1	0x02	このファンクションが持つインタフェースの数:2
bFunctionClass	1	0x02	このファンクションのクラス・コード: CDC
bFunctionSubClass	1	0x00	このファンクションのサブクラス・コード：なし
bFunctionProtocol	1	0x00	このファンクションのプロトコル・コード：なし
iFunction	1	0x00	このファンク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

(4) 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。
サンプルプログラムは、3 つのインタフェースを持ちます。interface0 が MSC、CDC はコントロールとデータの 2 つのインタフェースを持ち、それぞれ interface1, interface2 です。

表 4-10　インタフェース・ディスクリプタの設定(MSC)

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

表 4-11　インタフェース・ディスクリプタの設定(CDC-コントロール)

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	0x01	このインタフェースの識別番号：1
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	0x01	このインタフェースが持つエンドポイントの数：1
bInterfaceClass	1	0x02	クラス・コード：コミュニケーション・インターフェース・クラス
bInterfaceSubClass	1	0x02	サブクラス・コード：Abstract Control Model
bInterfaceProtocol	1	0x00	プロトコル・コード：固有プロトコル使用せず
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

表 4-12　インタフェース・ディスクリプタの設定(CDC-データ)

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	0x02	このインタフェースの識別番号：2
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	0x0A	クラス・コード：コミュニケーション・インターフェース・クラス
bInterfaceSubClass	1	0x00	サブクラス・コード：Abstract Control Model
bInterfaceProtocol	1	0x00	プロトコル・コード：固有プロトコル使用せず
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

(5) 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。
サンプルプログラムではエンドポイントを 5 つ使用するため，ディスクリプタも 5 種類設定しています。
エンドポイントのアドレスはヘッダ・ファイル(usbf850.h)の下記の define 定義を有効/無効にすることで切り替えが可能です(デフォルトは有効)。

```
#define　　USE_EP_BKI1
#define　　USE_EP_BKO1
```

表 4-13　エンドポイントの組み合わせ

	MSC		CDC	
	Bulk In	Bulk Out	Bulk In	Bulk Out
define を有効	EP1	EP2	EP3	EP4
define を無効	EP3	EP4	EP1	EP2

表 4-14　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 4-15　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

表 4-16　Endpoint3（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	0x83	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：3
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：0ms

表 4-17　Endpoint4（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	0x04	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：4
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 4-18　Endpoint7（インタラプト・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	0x87	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：7
bmAttributes	1	0x03	このエンドポイントの転送タイプ：インタラプト
wMaxPacketSize	2	0x0040	この転送の最大パケット・サイズ：64 バイト
bInterval	1	0x0A	このエンドポイントのポーリング間隔：10 ms

(6) 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，サンプルプログラム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 4-19　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5	1	0x40	ディスクリプタのサイズ：64 バイト
bDescriptorType	1	0x03	ディスクリプタの種類：ストリング
bString 注6	62	–	ベンダ：Renesas Electronics Corporation

（注 5）　bString フィールドのサイズにより設定値が異なります。
（注 6）　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。

(c) String 2

フィールド	サイズ（バイト）	設定値	説　明
bLength 注7	1	0x10	ディスクリプタのサイズ：16 バイト
bDescriptorType	1	0x03	ディスクリプタの種類：ストリング
bString 注8	14	–	製品の種類：MultDrv（マルチファンクションドライバ）

（注 7）　bString フィールドのサイズにより設定値が異なります。
（注 8）　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。

(d) String 3

フィールド	サイズ（バイト）	設定値	説　明
bLength 注9	1	0x1A	ディスクリプタのサイズ：26 バイト
bDescriptorType	1	0x03	ディスクリプタの種類：ストリング
bString 注10	24	–	シリアル番号： V850E2/ML4 ： 0216EF020110

（注 9）　bString フィールドのサイズにより設定値が異なります。
（注 10）　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。

4.1.5　SCSIコマンドへの対応

MSC のサンプルプログラムでは，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。
サンプルプログラムの対応する SCSI コマンドを表 4－20 に示します。表 4－20 にないコマンドを受信した場合，STALL 応答を行います。

表 4-20　サンプルプログラムの対応する 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。サンプルプログラムでは，センス・データを初期化して正常終了します。

表 4-21　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。サンプルプログラムでは，表 4－22 に示すセンス・データをホストに送信します。

表 4-22　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 4-23　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 4-24　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 4-25　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 4-26　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。サンプルプログラムでは，センス・データを初期化して正常終了します。

表 4-27　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。サンプルプログラムでは，INQUIRY_TABLE の値をホストに送信します。

表 4-28　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 4-29　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。サンプルプログラムでは，MODE_SELECT_TABLE に値を書き込みます。

表 4-30　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 4-31　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。サンプルプログラムでは，MODE_SENSE_TABLE の値をホストに送信します。

表 4-32　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 4-33　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。サンプルプログラムでは，センス・データを初期化して正常終了します。

表 4-34　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。サンプルプログラムでは，何も処理せずに正常終了します。

表 4-35　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。サンプルプログラムでは，READ_FORMAT_CAPACITY_TABLE の値をホストに送信します。

表 4-36　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 4-37　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。サンプルプログラムでは,
READ_CAPACITY_TABLE の値をホストに送信します。

表 4-38　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 4-39　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 4-40　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 4-41　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。サンプルプログラムでは，書き込みのみを行います。

表 4-42　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。サンプルプログラムでは，何も処理せずに終了します。

表 4-43　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。サンプルプログラムでは，センス・データを初期化して正常終了します。

表 4-44　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。サンプルプログラムでは，データを読み捨てて，正常終了します。

表 4-45　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。サンプルプログラムでは，MODE_SENSE10_TABLE の値をホストに送信します。

表 4-46　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 4-47　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　MODE_SENSE10_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。サンプルプログラムでは，MODE_SELECT10_TABLE に値を書き込みます。

表 4-48　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 4-49　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

サンプルプログラムを実行すると，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。

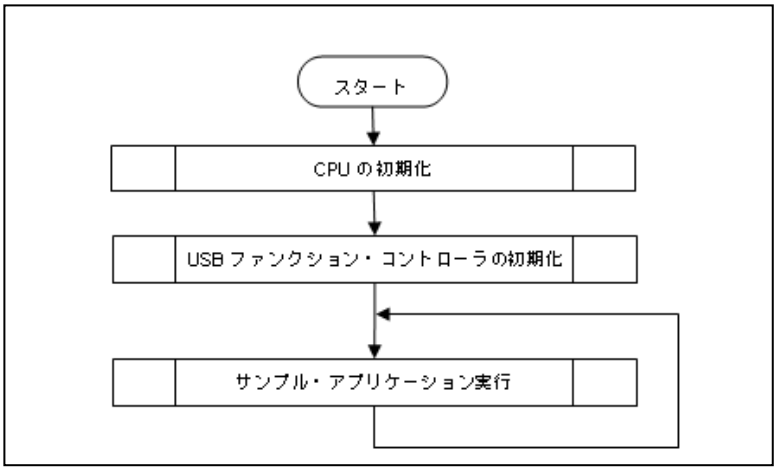


図 4−8 サンプルプログラムの処理フロー

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。

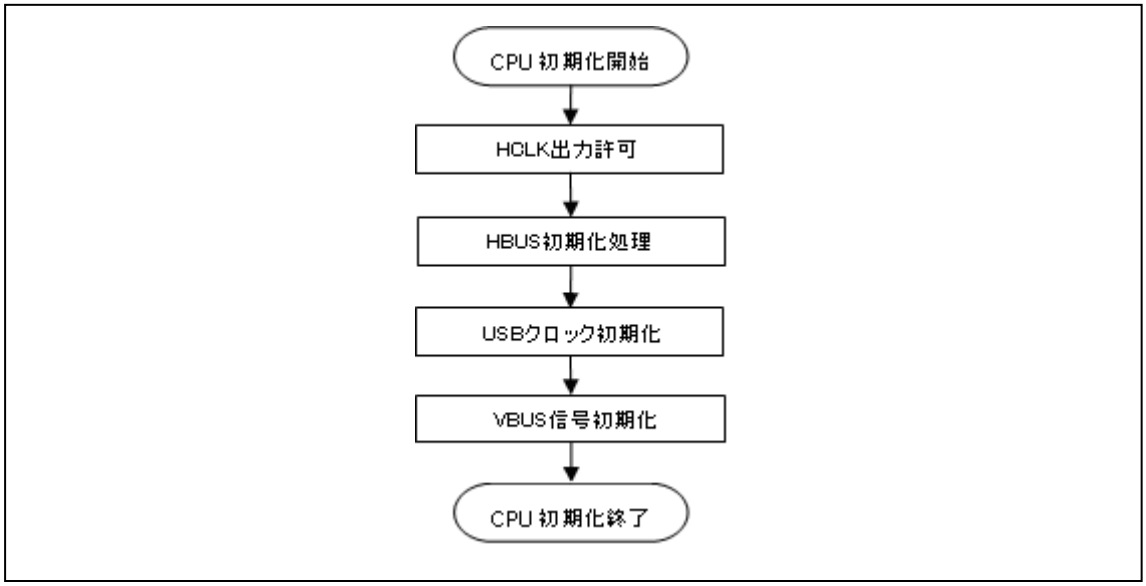


図 4−9 CPU 初期化の処理フロー

(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/ML4 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 P2_11 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。

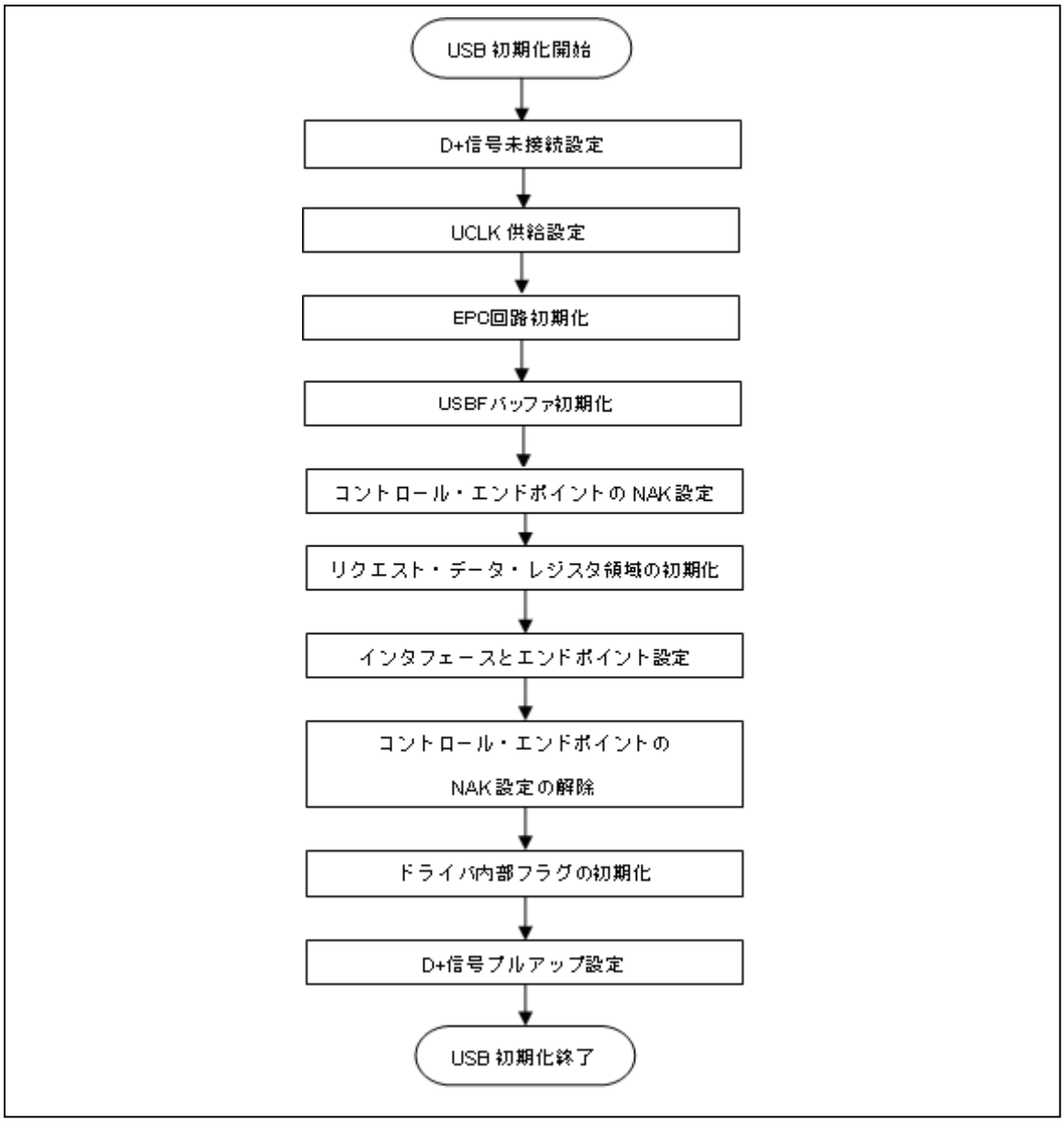


図 4－10 USB ファンクション・コントローラ初期化の処理フロー

(1) D+信号プルダウン設定

CPU の P2_4 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 "0x81" を書き込みます。この設定により，インタフェース 0～2 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。
(e) USFA0E3IM レジスタに "0x60" を書き込みます。この設定により，Endpoint3 が Interface2 にリンクされます。
(f) USFA0E4IM レジスタに "0x60" を書き込みます。この設定により，Endpoint4 が Interface2 にリンクされます。
(g) USFA0E7IM レジスタに "0x40" を書き込みます。この設定により，Endpoint7 が Interface1 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 "0xEE" 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 と BKO2DT 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 P2 レジスタに "0x0010" を書き込みます。この設定により，P2_4 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。サンプルプログラムでは図 4－11 に示すような接続を想定しています。

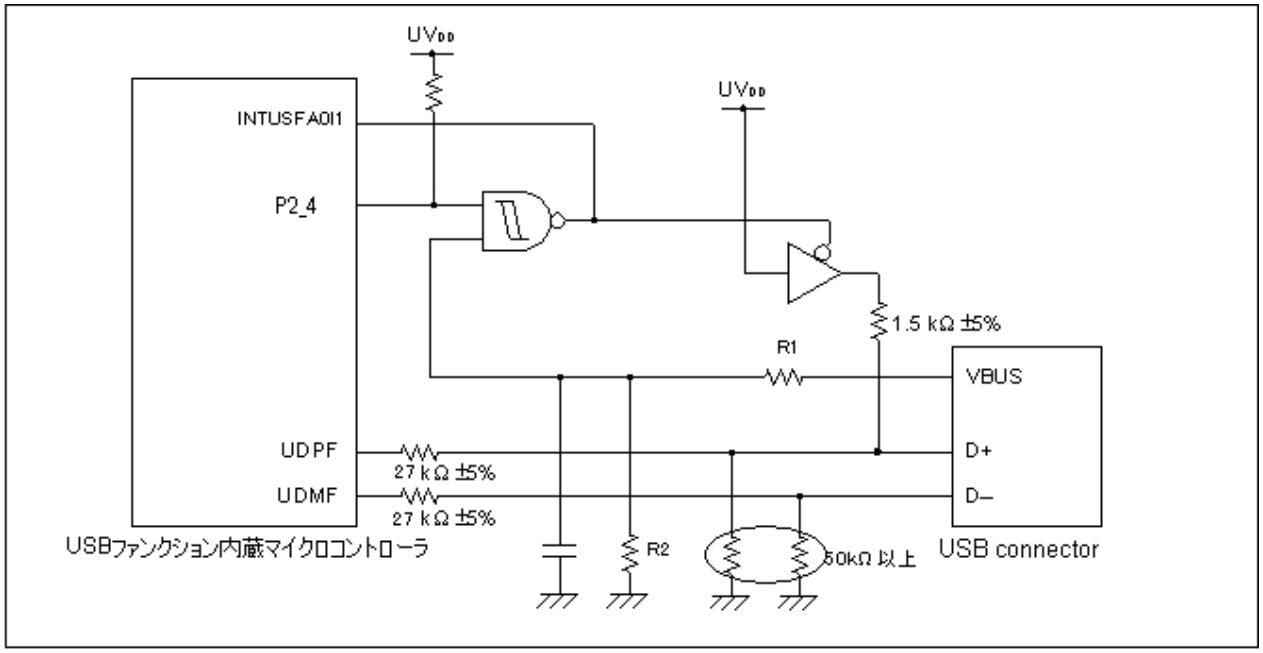


図 4−11　USB ファンクション・コントローラ接続例

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,Endpoint4）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。

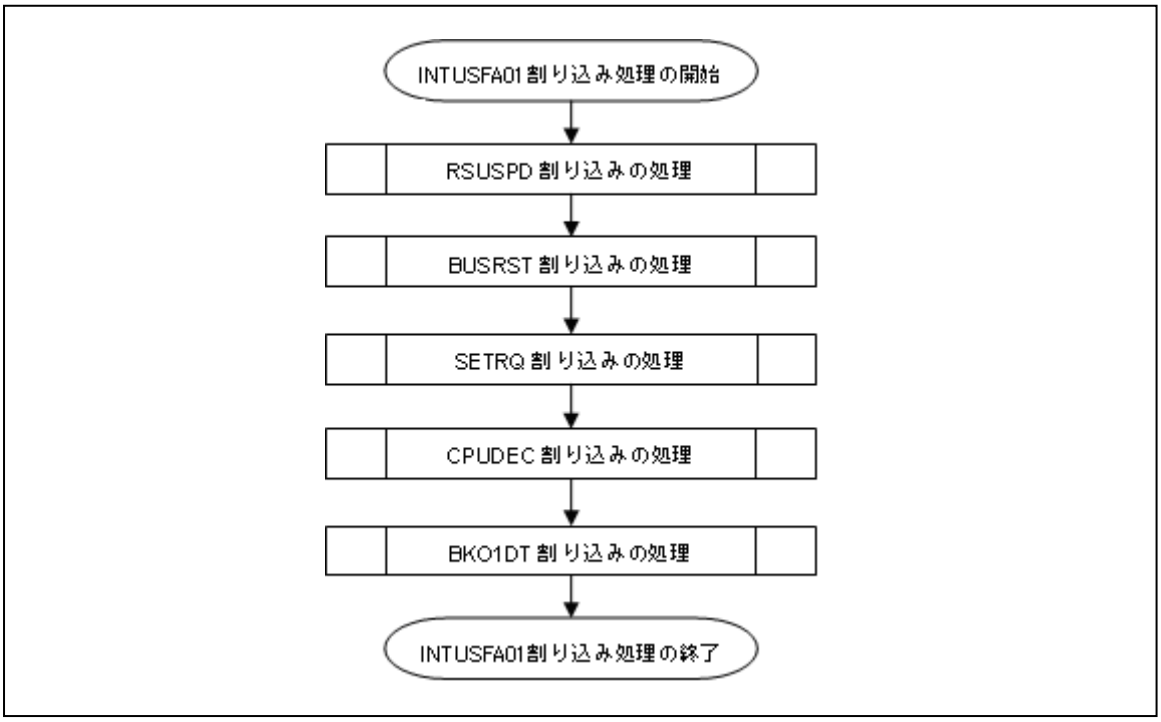


図 4－12　INTUSFA0I1 割り込みハンドラ処理フロー

（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) BKOnDT 割り込みの処理

USFA0IS3 レジスタの BKOnDT 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。
BKO1DT 割り込みが発生している場合，次の処理を行います。

- BKO1DT 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。

BKO2DT 割り込みが発生している場合，次の処理を行います。

- BKO2DT 割り込み要因クリア（USFA0IC3 レジスタの BKO2DT ビットに "0" を書き込む）
- CDC データを受信したことを示すフラグ(usbf850_cdc_rdata_flg)を更新します。

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。

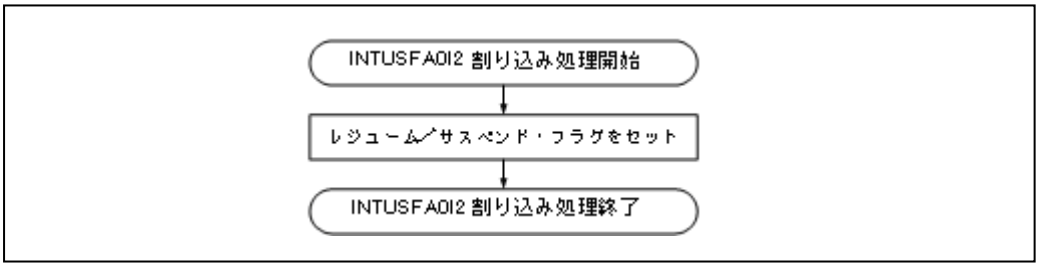


図 4－13　INTUSFA0I2 割り込みハンドラ処理フロー

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。

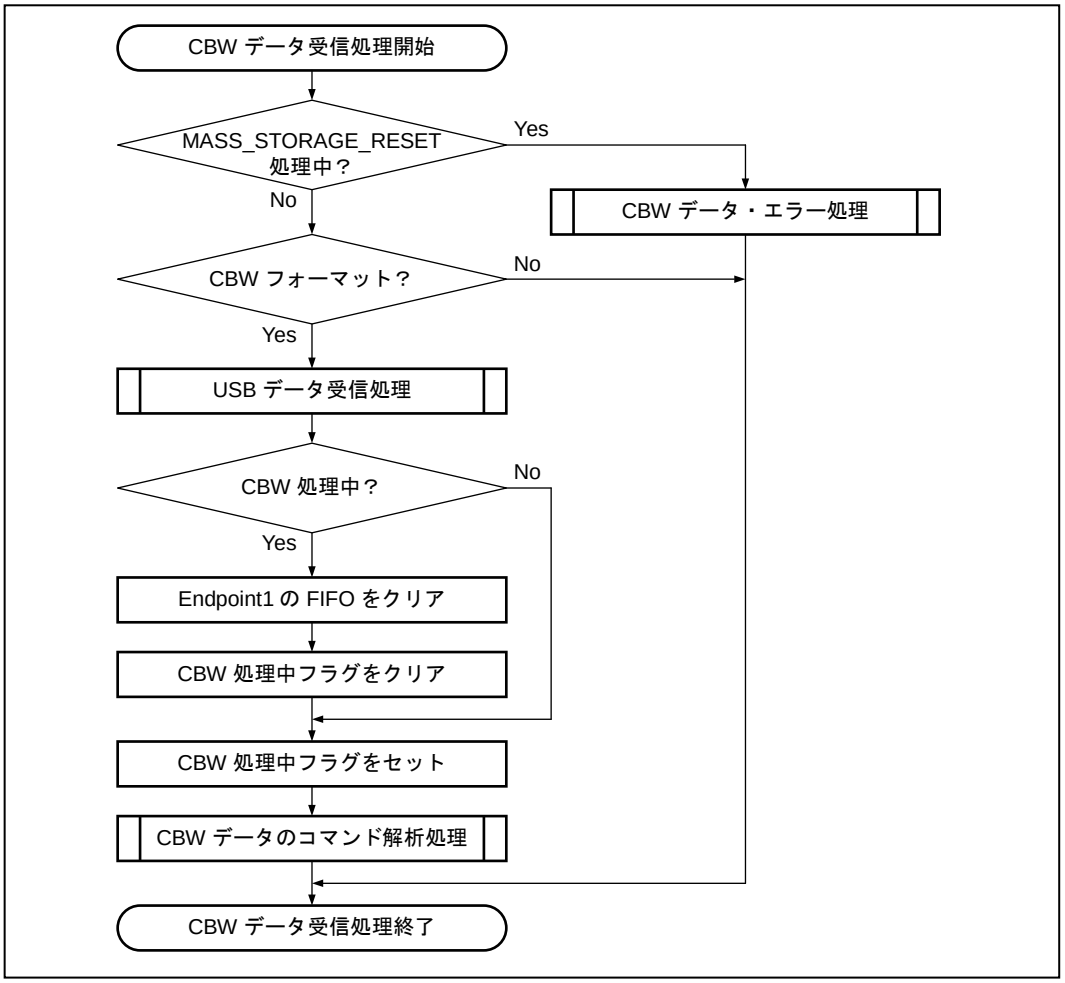


図 4－14　CBW データ受信処理フロー

(1) MASS_STORAGE_RESET 処理中の判定

MASS_STORAGE_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_multiple_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。

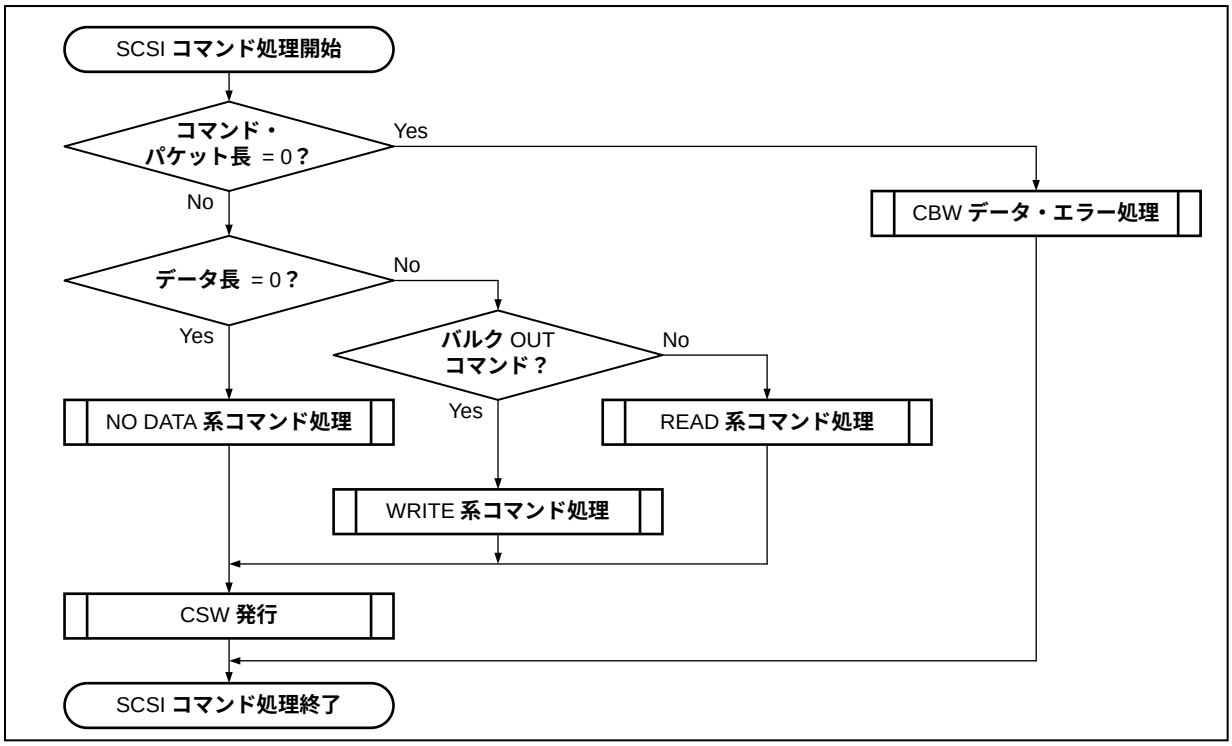


図 4−15　SCSI コマンド処理フロー

（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_multiple_data_send）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_multiple_data_receive）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。

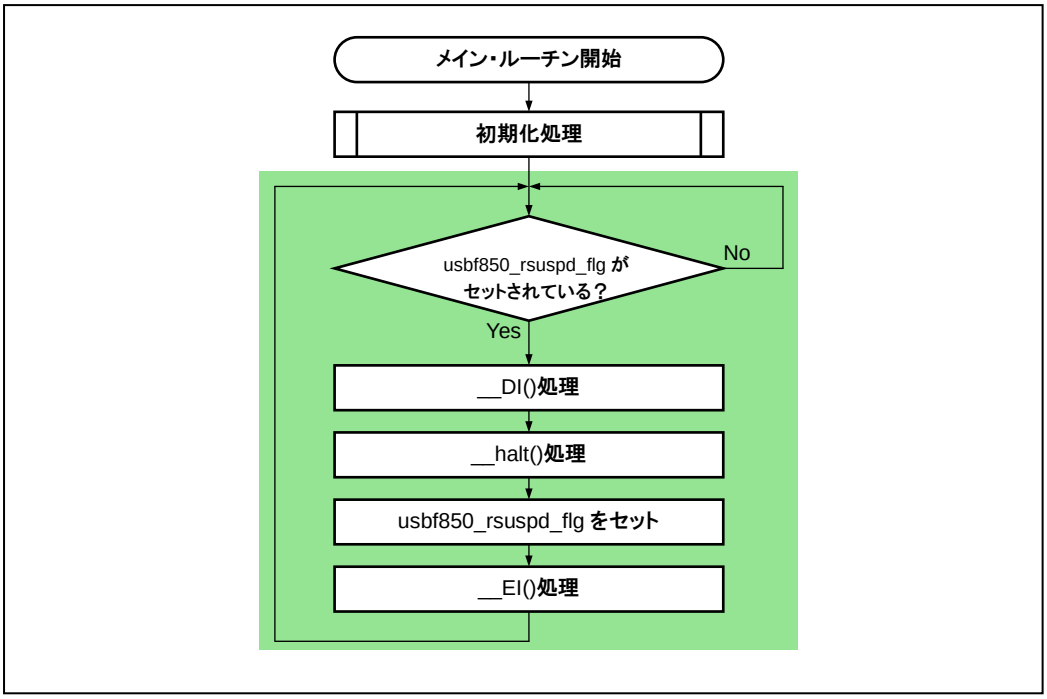


図 4－16　サスペンド／レジューム処理フロー

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

サンプルプログラム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，サンプルプログラム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

サンプルプログラムでは，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 4-50　サンプルプログラム内の関数（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_multiple_data_send	USB データの送信(MSC 用)
	usbf850_data_send	USB データの送信(CDC 用)
	usbf850_multiple_data_receive	USB データの受信(MSC 用)
	usbf850_data_receive	USB データの受信(CDC 用)
	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 Storage 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 4-51　サンプルプログラム内の関数（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）
usbf850_communicaion.c	usbf850_cdc_classreq	CDC クラス・リクエストの処理
	usbf850_send_encapsulated_command	Send Encapsulated Command リクエストの処理
	usbf850_get_encapsulated_response	Get Encapsulated Command リクエストの処理
	usbf850_set_line_coding	Set Line Coding リクエストの処理
	usbf850_get_line_coding	Get Line Coding リクエストの処理
	usbf850_set_control_line_state	Set Control Line State リクエストの処理
	usbf850_buff_init	CDC データ転送用の Endpoint の FIFO クリア処理
	usbf850_get_bufinit_flg	FIFO 初期化処理の実行状態通知処理
	usbf850_send_buf	CDC データの送信処理
	usbf850_recv_buf	CDC のクラス・リクエスト処理関数登録

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。

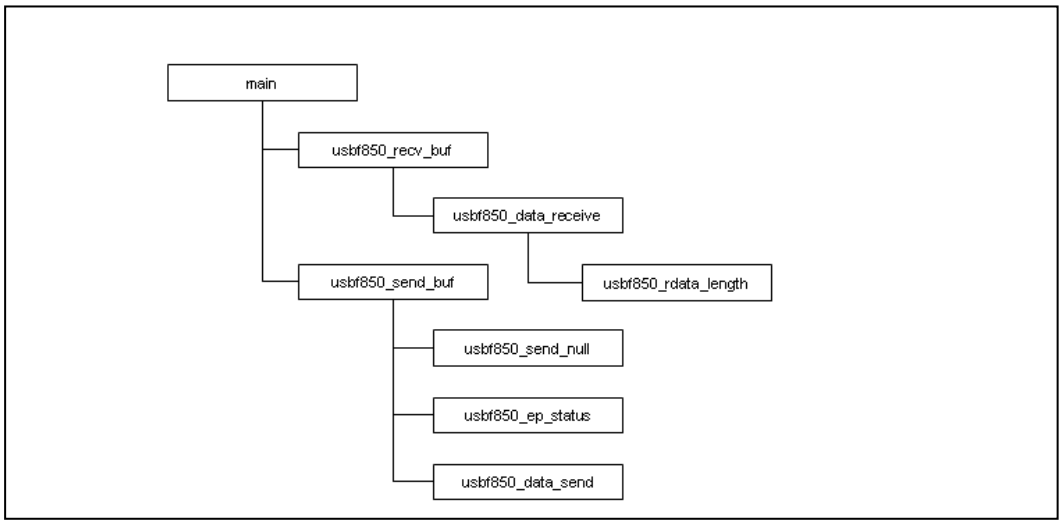


図 4－17　main 処理での関数の呼び出し

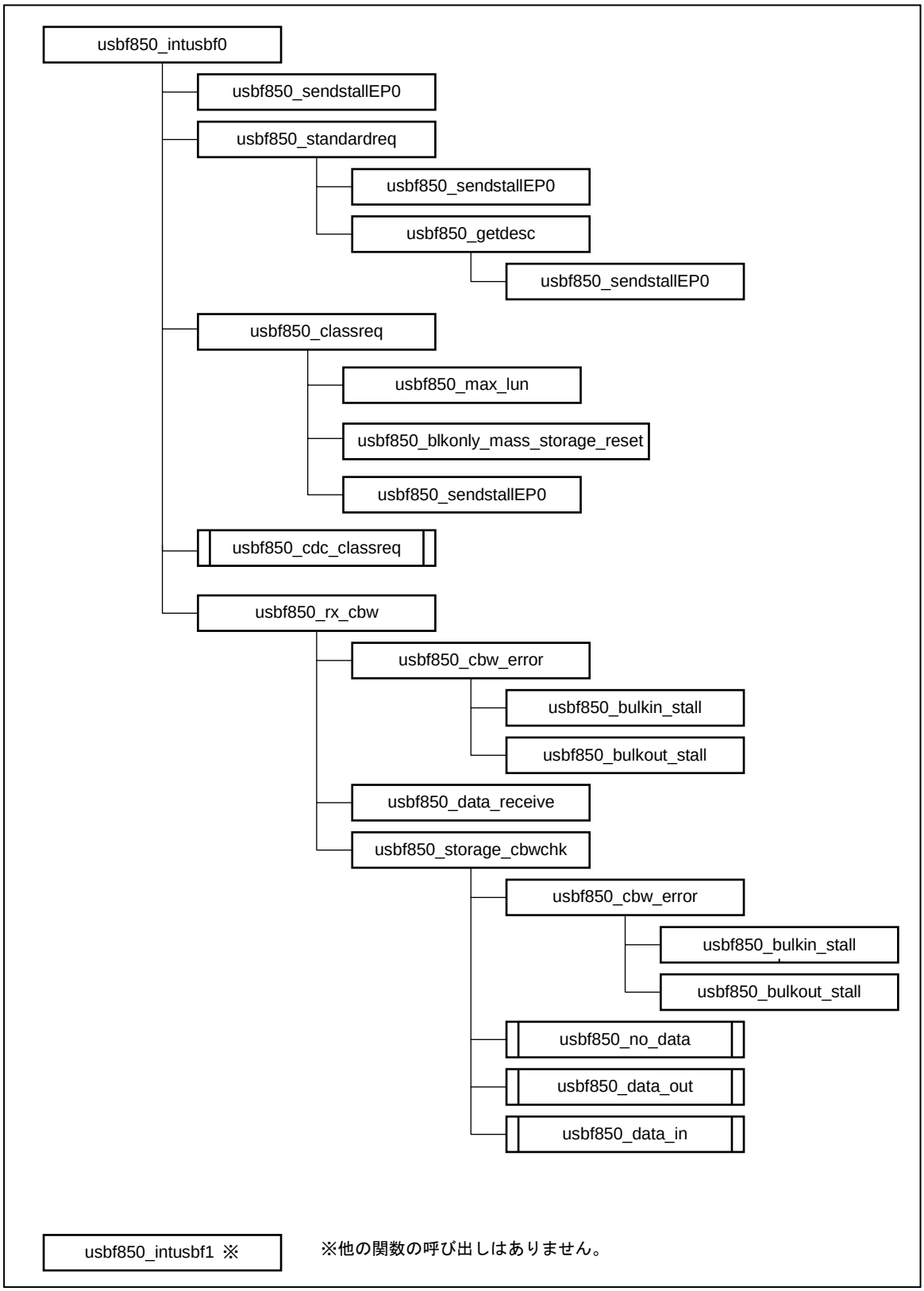


図 4－18　USB 割り込み処理での関数の呼び出し

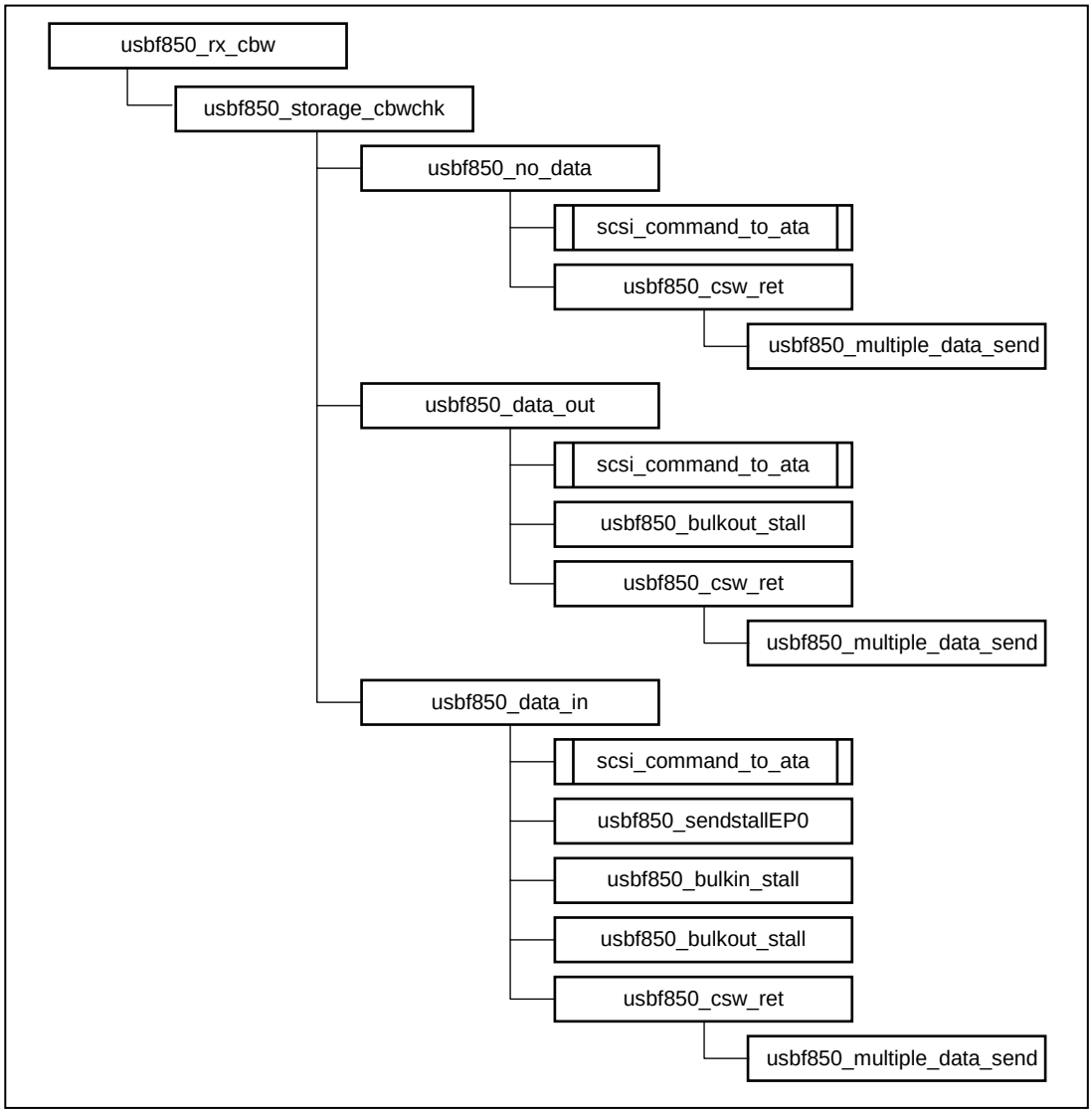


図 4－19　CBW/CSW 処理での関数の呼び出し

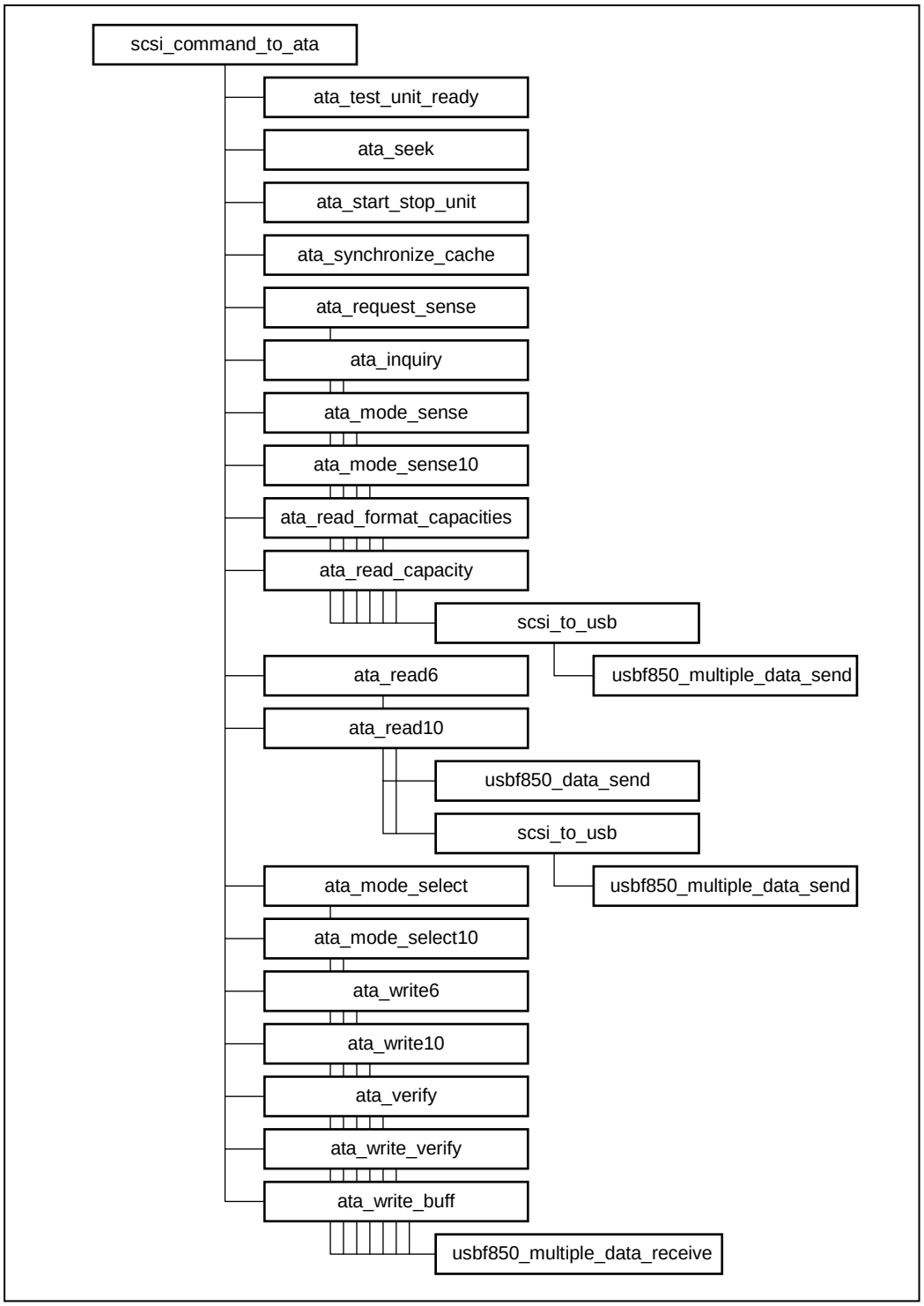


図 4－20　SCSI コマンド処理での関数の呼び出し

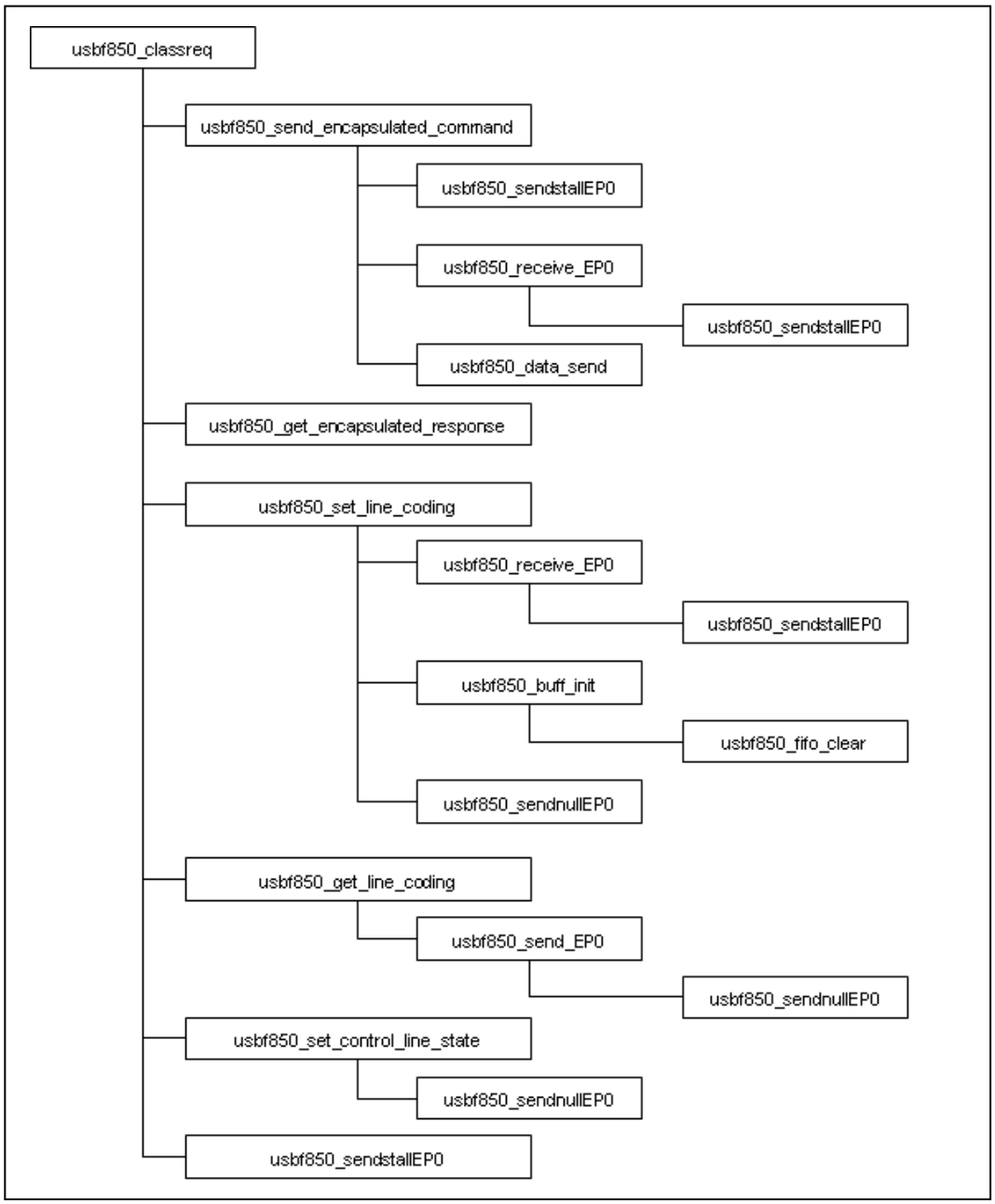


図 4−21　USB コミュニケーション・クラス用処理での関数の呼び出し(1/2)

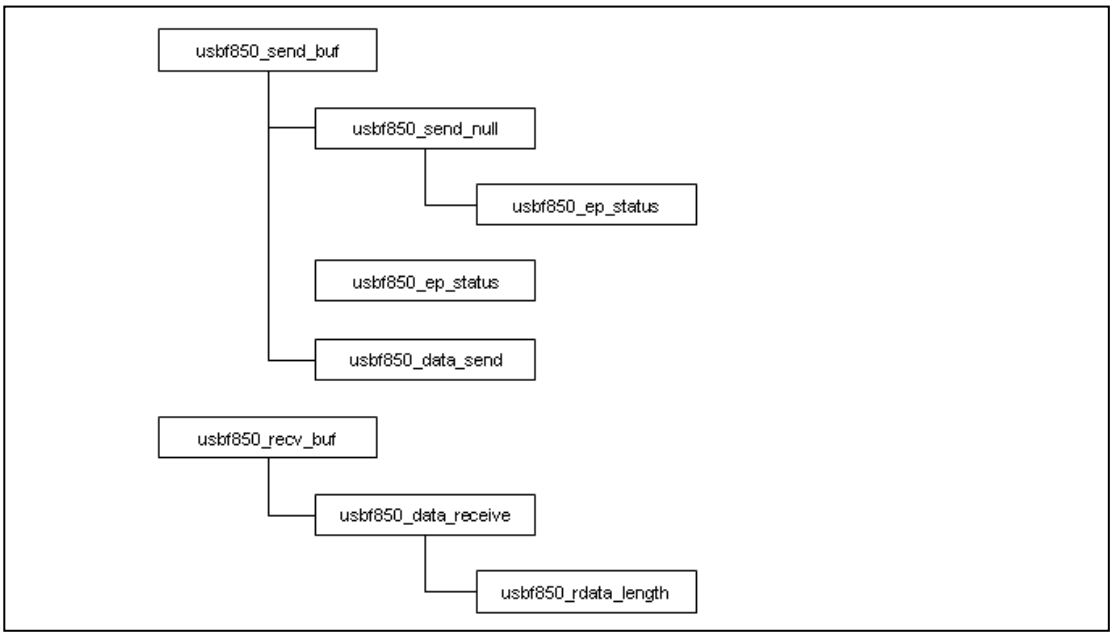


図 4－22　USB コミュニケーション・クラス用処理での関数の呼び出し(2/2)

4.3.3 関数の機能

ここでは，サンプルプログラム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

【引数】

引数の説明

引数	説　　明
引数型，名称	*引数概要説明*

【 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)

【引数】
なし

【 戻り値 】
なし

【 機　能 】
サンプルプログラム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)

【引数】
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)

【引数】

引数	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)

【引数】
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)

【引数】

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，Endpoint3）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。
Endpoint3 では, BKO2DT 割り込みを監視します。BKO2DT 割り込み発生時は、CDC データを受信したことを示すフラグ(usbf850_cdc_rdata_flg)を更新します。

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)

【引数】
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_multiple_data_send

【 概　要 】

USB データ送信処理(MSC 用)

【C 言語記述形式】

INT32 usbf850_multiple_data_send(UINT8 *data, INT32 len, INT8 ep)

【引数】

引数	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 】

複数パケットの送信が可能です。

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_send

【 概　要 】

USB データ送信処理(CDC 用)

【C 言語記述形式】

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

【引数】

引数	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(<Max packet size)
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 】

シングルパケットのデータ送信を行います。

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_multiple_data_receive

【 概　要 】

USB データ受信処理(MSC 用)

【C 言語記述形式】

INT32 usbf850_multiple_data_receive(UINT8 *data, INT32 len, INT8 ep)

【引数】

引数	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 】

複数パケットのデータ受信が可能です。
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。複数パケットの受信が可能であり、MSC からのみ使用されます。

usbf850_data_receive

【 概　要 】

USB データ受信処理(CDC 用)

【C 言語記述形式】

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

【引数】

引数	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(< Max Packet Size)
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 】

シングルパケットのデータ受信を行います。
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)

【引数】

引数	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)

【引数】

引数	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)

【引数】

引数	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)

【引数】

引数	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)

【引数】

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)

【引数】
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)

【引数】

引数	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)

【引数】

引数	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)

【引数】

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)

【引数】
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)

【引数】

引数	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)

【引数】
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)

【引数】
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)

【引数】
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)

【引数】

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)

【引数】
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)

【引数】
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)

【引数】
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)

【引数】
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)

【引数】

引数	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)

【引数】
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)

【引数】
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)

【引数】

引数	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)

【引数】

引数	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)

【引数】

引数	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)

【引数】

引数	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)

【引数】

引数	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)

【引数】

引数	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)

【引数】

引数	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 データを送信します。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)

【引数】

引数	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)

【引数】

引数	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)

【引数】

引数	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)

【引数】

引数	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)

【引数】

引数	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)

【引数】

引数	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)

【引数】

引数	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)

【引数】

引数	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)

【引数】

引数	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)

【引数】

引数	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)

【引数】

引数	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)

【引数】

引数	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)

【引数】

引数	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)

【引数】

引数	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_multiple_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。

（6）USB コミュニケーション・クラス用処理の関数

usbf850_cdc_classreq

【 概　要 】

CDC のクラス・リクエスト処理

【C 言語記述形式】

void usbf850_cdc_classreq(USB_SETUP *req_data)

【引数】

引数	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_send_encapsulated_command

【 概　要 】

Send Encapsulated Command リクエスト処理

【C 言語記述形式】

void usbf850_send_encapsulated_command(void)

【引数】

なし

【 戻り値 】

なし

【 機　能 】

データ受信処理関数（usbf850_data_receive）を呼び出して Endpoint0 で受信したデータを取り込み，データ送信処理関数（usbf850_data_send）を呼び出して Endpoint2 からバルク・イン転送（送信）でデータを送信させます。

usbf850_set_line_coding

【 概　要 】
Set Line Coding リクエスト処理

【C 言語記述形式】
void usbf850_set_line_coding(void)

【引数】
なし

【 戻り値 】
なし

【 機　能 】
データ受信処理関数（usbf850_data_receive）を呼び出して Endpoint0 で受信したデータを取り込み，UART_MODE_INFO 構造体に書き込みます。またこの値を基に，転送速度やデータ長などの UART のモードを設定したあと，Endpoint0 用 NULL パケット送信処理関数（usbf850_sendnullEP0）を呼び出します。

usbf850_get_line_coding

【 概　要 】
Get Line Coding リクエスト処理

【C 言語記述形式】
void usbf850_get_line_coding(void)

【引数】
なし

【 戻り値 】
なし

【 機　能 】
データ送信処理関数（usbf850_data_send）を呼び出して，UART_MODE_INFO 構造体の値を Endpoint0 から送信させます。

usbf850_set_control_line_state

【 概　要 】
Set Control Line State リクエスト処理

【C 言語記述形式】
void usbf850_set_control_line_state(void)

【引数】
なし

【 戻り値 】
なし

【 機　能 】
Endpoint0 用 NULL パケット送信処理関数（usbf850_sendnullEP0）を呼び出します。

4.4 データ構造体

サンプルプログラム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 4－23 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 4－24 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 4－25 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 4－26　SCSI SENSE DATA 構造体

5. 開発環境

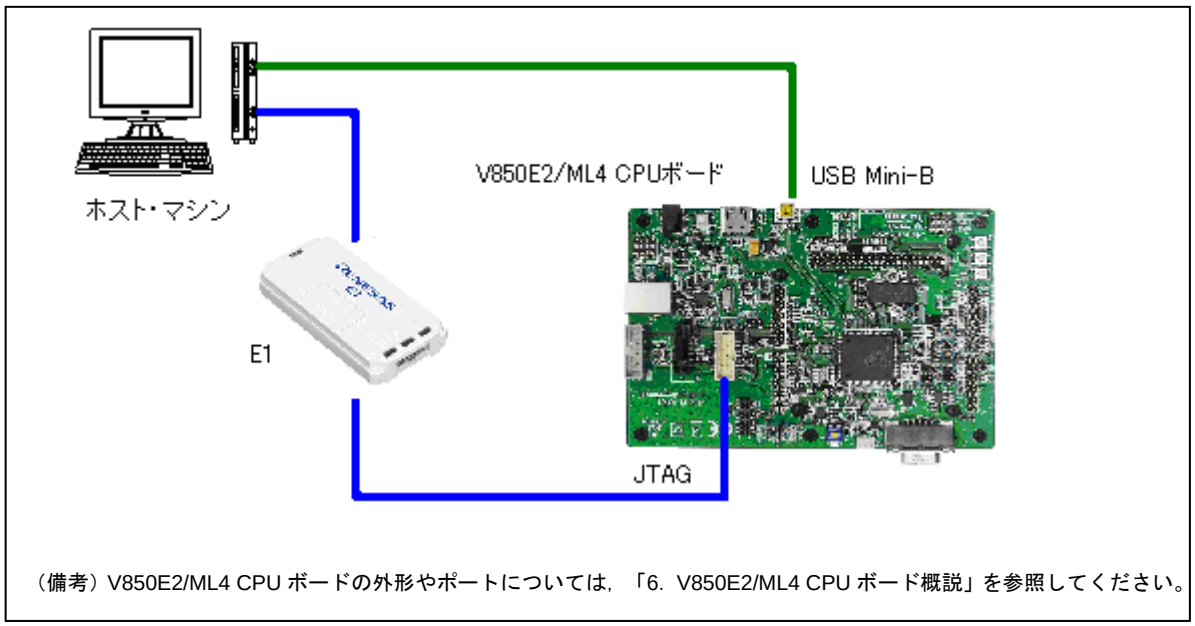
この章では，V850E2/ML4 向け USB マルチファンクションのサンプルプログラムを利用する際の開発環境構築の例と，そこでのデバッグ手順について説明します。

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

サンプルプログラムを利用するシステムの構成を図 5－1 に示します。



（備考）V850E2/ML4 CPU ボードの外形やポートについては，「6. V850E2/ML4 CPU ボード概説」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

サンプルプログラム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® 7, Vista, XP）
ソフトウエア	統合開発ツール	CubeSuite+	V1.00.01
	コンパイラ	CX	V1.20

5.1.3　デバッグ

サンプルプログラム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® 7, Vista, XP）
	ターゲット	V850E2/ML4 CPUボード	ルネサスエレクトロニクス製
	エミュレータ	E1エミュレータ	ルネサスエレクトロニクス製
	USBケーブル	－	mini-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+	V1.00.01
ファイル	デバイス・ファイル	CubeSuite+用V850デバイス依存情報	V.1.00.02
	プロジェクト関連ファイル	－	(注11)

（注 11）CubeSuite+で構築した場合のファイルがサンプルプログラム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。

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

サンプルプログラム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。

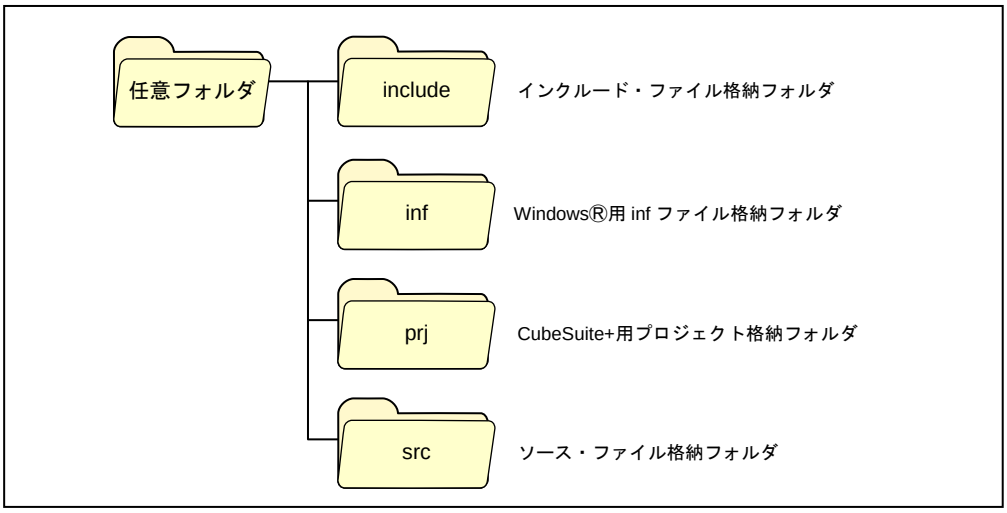


図 5−2　サンプルプログラムのフォルダ構成（CubeSuite+版）

(3) ワークスペースの設定

ここではサンプルプログラム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。

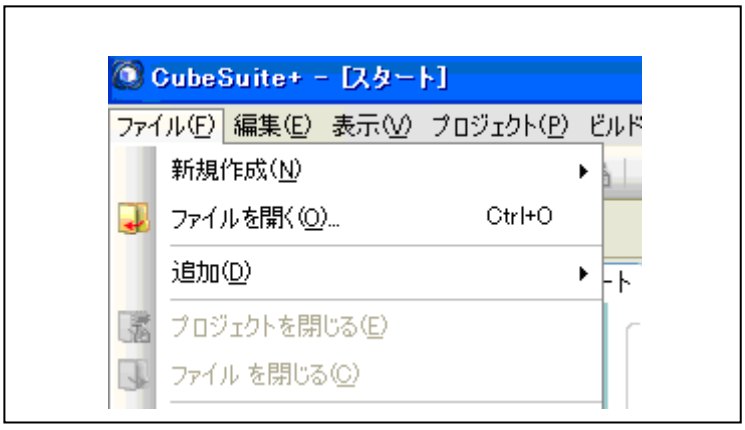


図 5−3　CubeSuite+メニュー選択

<2>　「ファイルを開く」ダイアログが開きます。サンプルプログラム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。

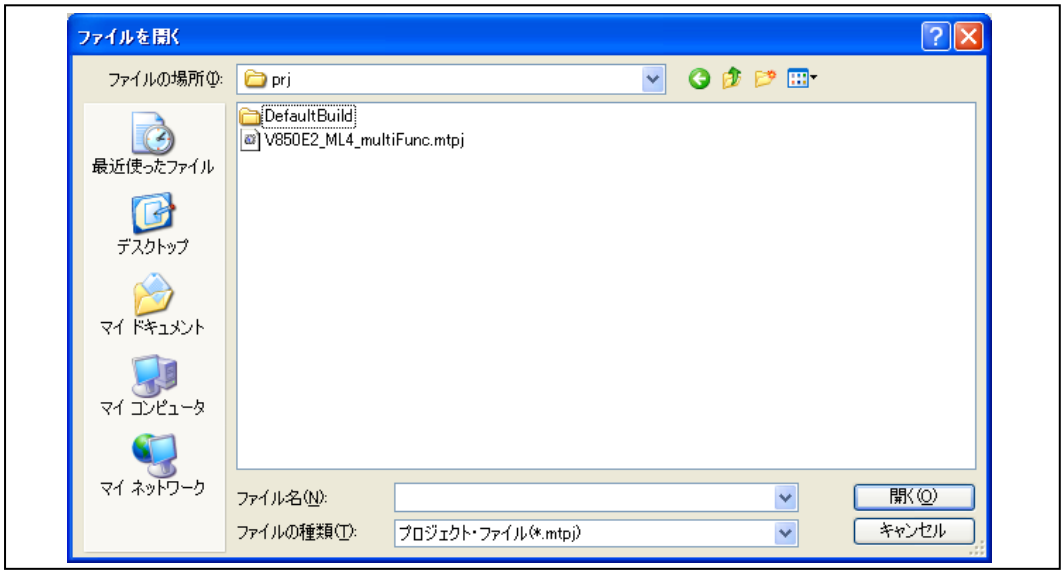


図 5−4　CubeSuite+プロジェクト・ファイル選択

(4) 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 CX のバージョン選択と，デバッグ・ツールとして V850E2M E1 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。

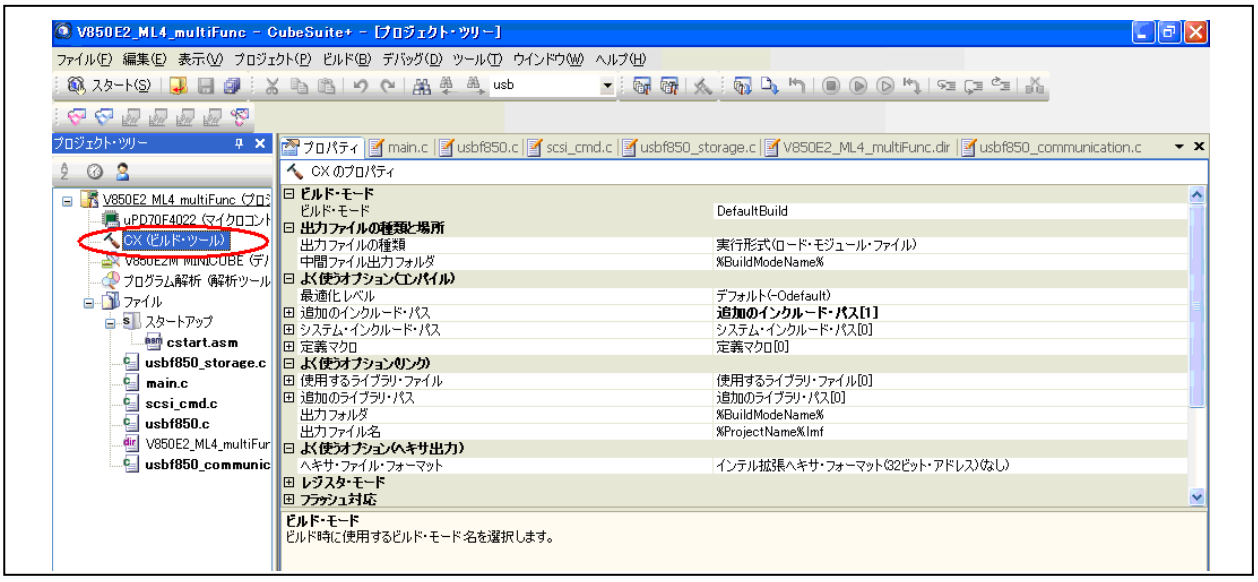


図 5−5　ビルド・ツール選択

<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。

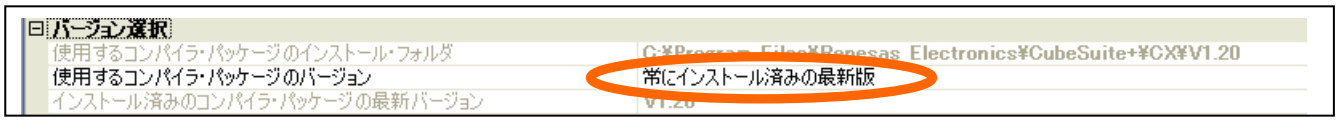


図 5−6　コンパイラ・パッケージ設定

<3>　プロジェクト・ツリーより「V850E2M　E1(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 E1」を選択します。

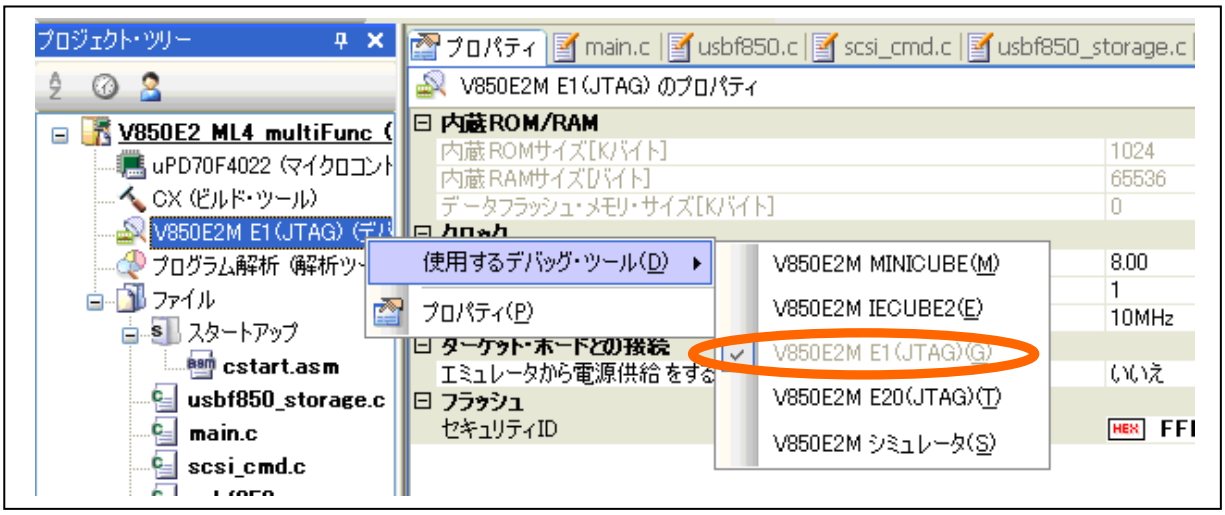


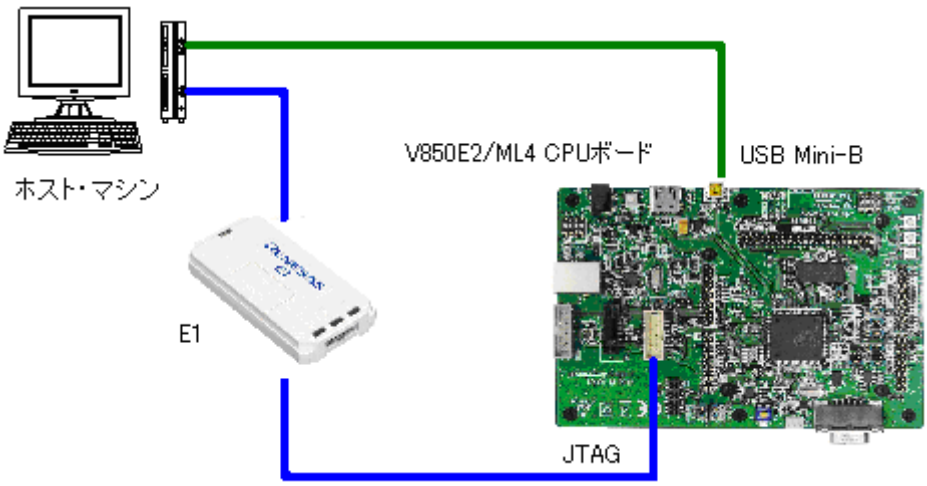
図 5−7　デバッグ・ツール選択

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。

(1) デバッグ・ポートの接続

V850E2/ML4 CPU ボードとホスト・マシンを接続します。V850E2/ML4 CPU ボードとホスト・マシンをデバッグ用に E1 で接続します。また V850E2/ML4 CPU ボードの USB mini-B タイプコネクタとホスト・マシンの USB コネクタを接続します。



（備考）V850E2/ML4 CPU ボードの外形やポートについては，「6. V850E2/ML4 CPU ボード概説」を参照してください。

図 5－8　V850E2/ML4 CPU ボードの接続

(2) ホストドライバのインストール

USB mini-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 mini-B コネクタとの接続に使用するドライバは，Windows 標準のコミュニケーション・デバイス・クラス用ホストドライバとマス・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「5.4　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。

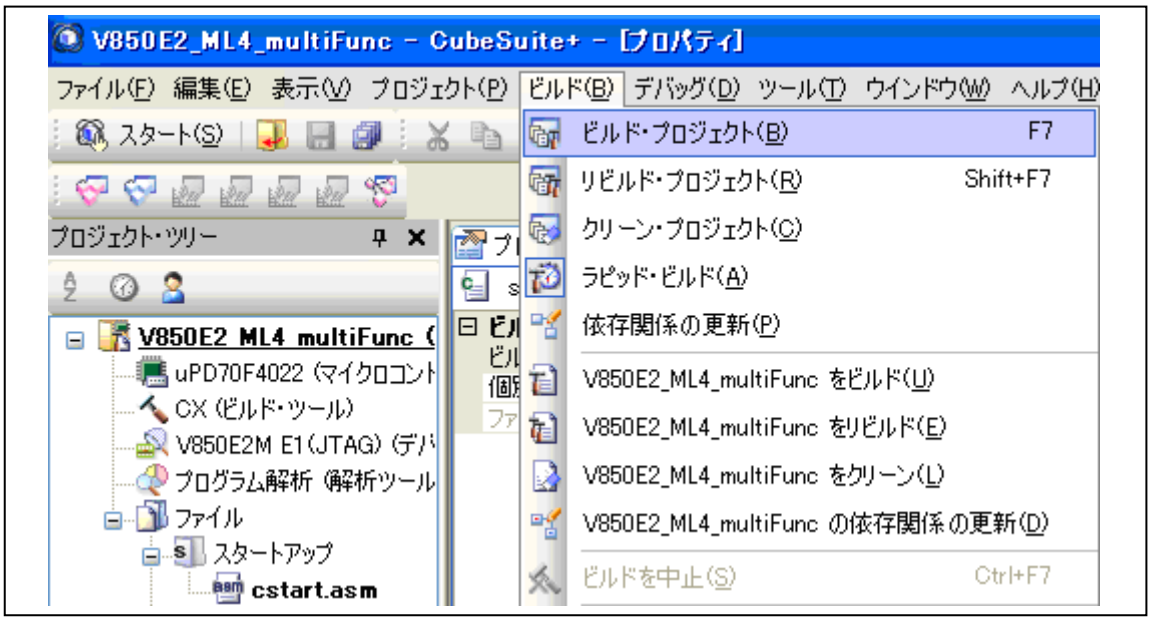


図 5−9 ビルド・プロジェクト選択

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 V850E2/ML4 CPU ボードにロード・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 5－10　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。

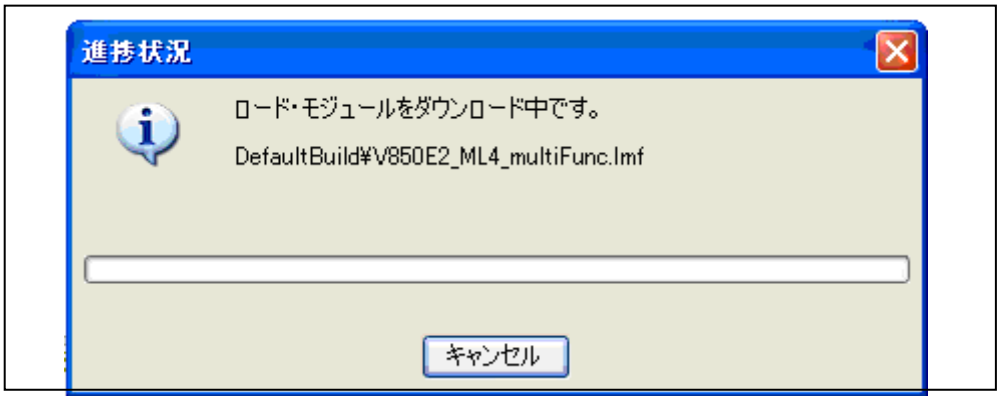


図 5－11　ダウンロード実行

(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。

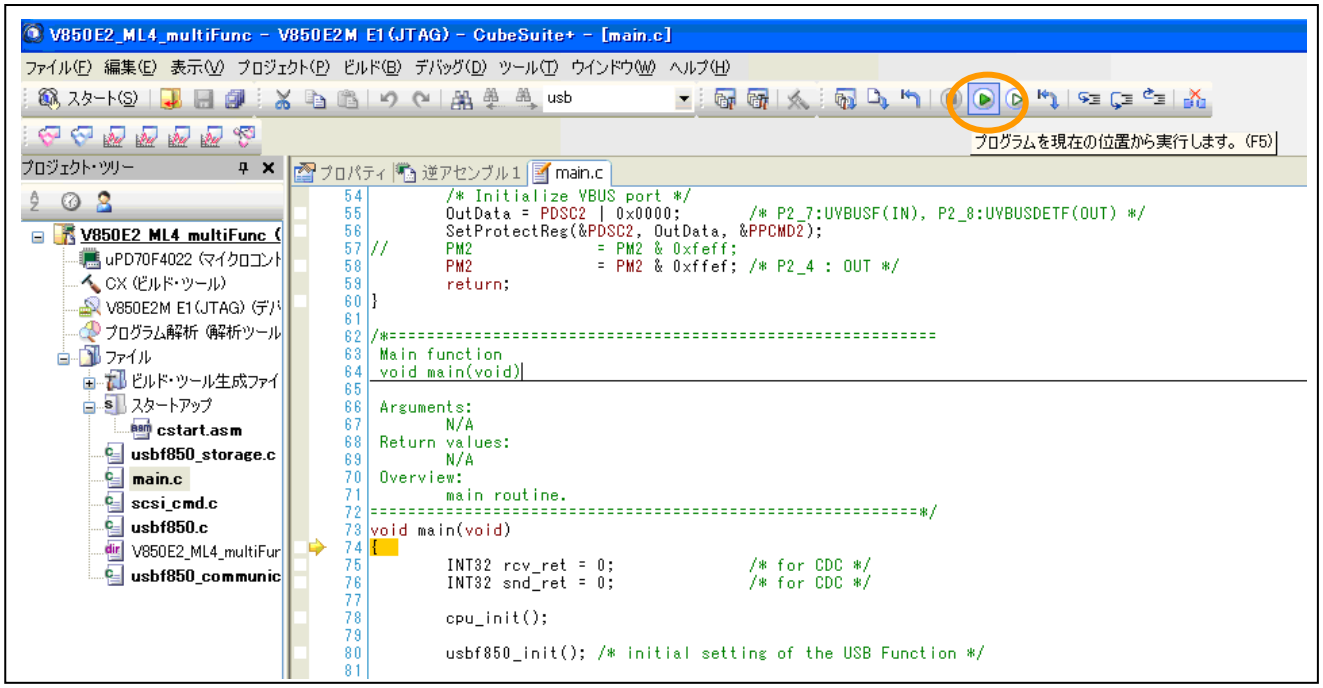


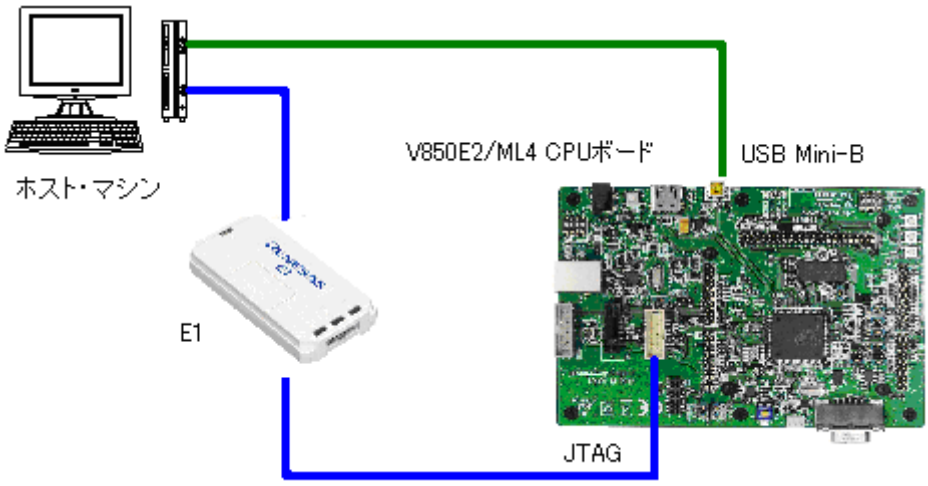
図 5－12　実行開始

5.3.3　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。

(1) デバッグ・ポートの接続

V850E2/ML4 CPU ボードとホスト・マシンを接続します。V850E2/ML4 CPU ボードとホスト・マシンをデバッグ用に E1 で接続します。また V850E2/ML4 CPU ボードの USB mini-B タイプコネクタとホスト・マシンの USB コネクタを接続します。



（備考）V850E2/ML4 CPU ボードの外形やポートについては，「6. V850E2/ML4 CPU ボード概説」を参照してください。

図 5－13　V850E2/ML4 CPU ボードの接続

(2) ホストドライバのインストール

USB mini-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　mini-B コネクタとの接続に使用するドライバは，Windows 標準のコミュニケーション・デバイス・クラス用**ホストドライバ**とマス・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. 4　動作確認」を参照してください。

5.4　動作確認

ここでは，サンプルプログラムのプログラムを CubeSuite+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 mini-B コネクタとの接続

V850E2/ML4 CPU ボードの USB mini-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

<1>　V850E2/ML4 CPU ボードの接続がホスト・マシンに認識されると，「新しいハードウエアが見つかりました」というメッセージが表示されたあと，新しいハードウエアの検出ウィザードが起動します。

<2>　「新しいハードウエアの検出ウィザード」ダイアログが開きます。「いいえ、今回は接続しません」を選択して「次へ」ボタンを押下します。

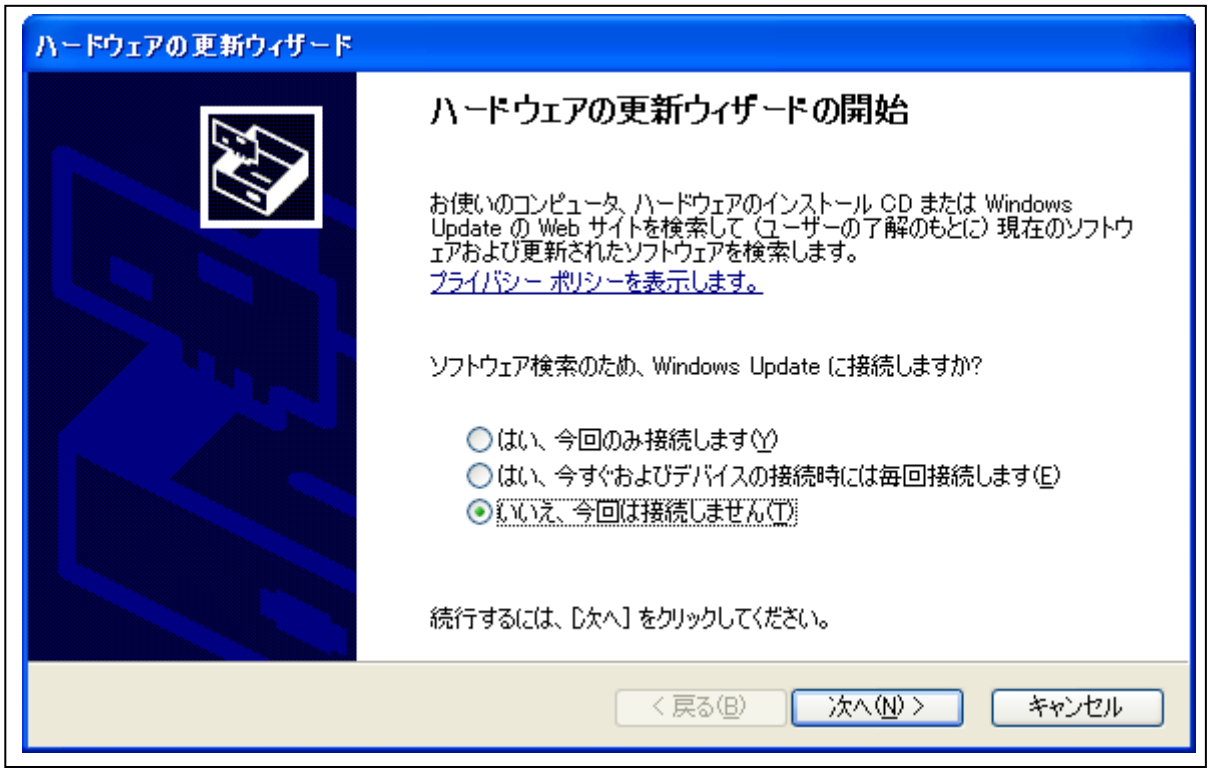


図 5－14　新しいハードウェアの検出ウィザード(1)

<3>　次の画面が表示されます。「一覧または特定の場所からインストールする（詳細）」を選択して「次へ」ボタンを押下します。

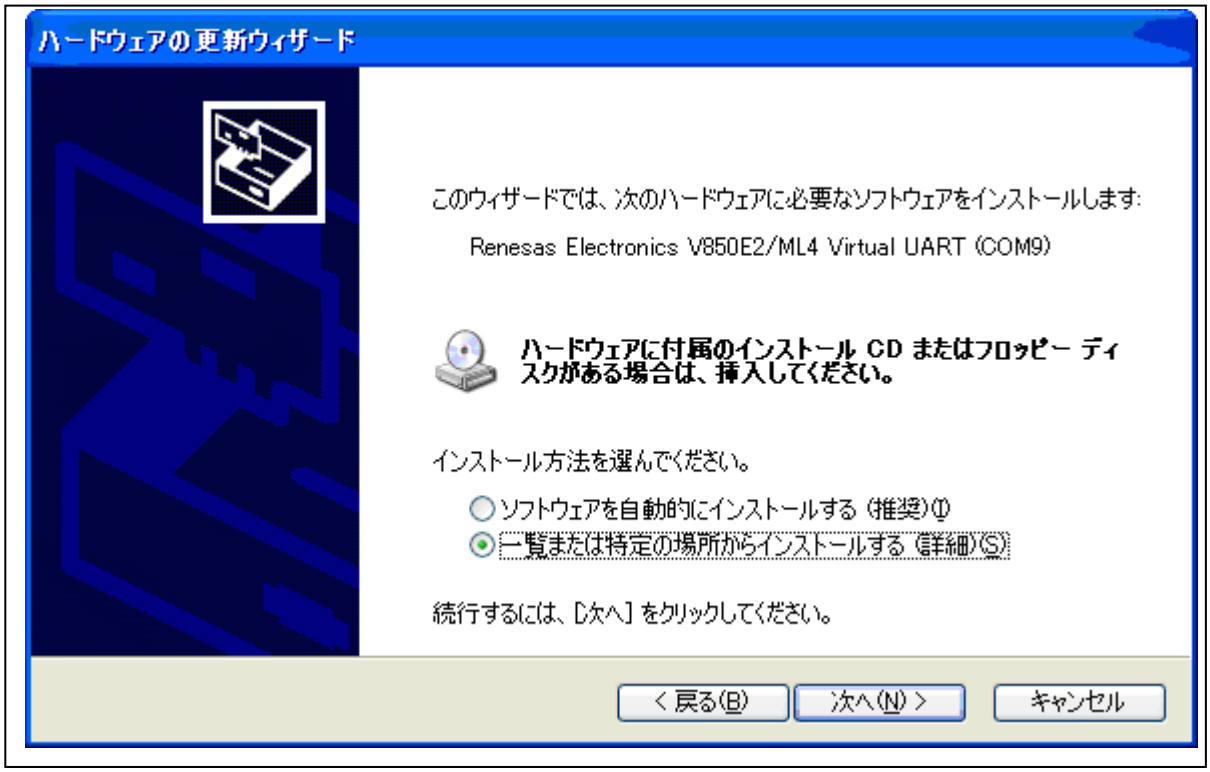


図 5－15　新しいハードウェアの検出ウィザード(2)

<4>　サンプルプログラムに付属している inf ファイルを選択してインストールを行います。

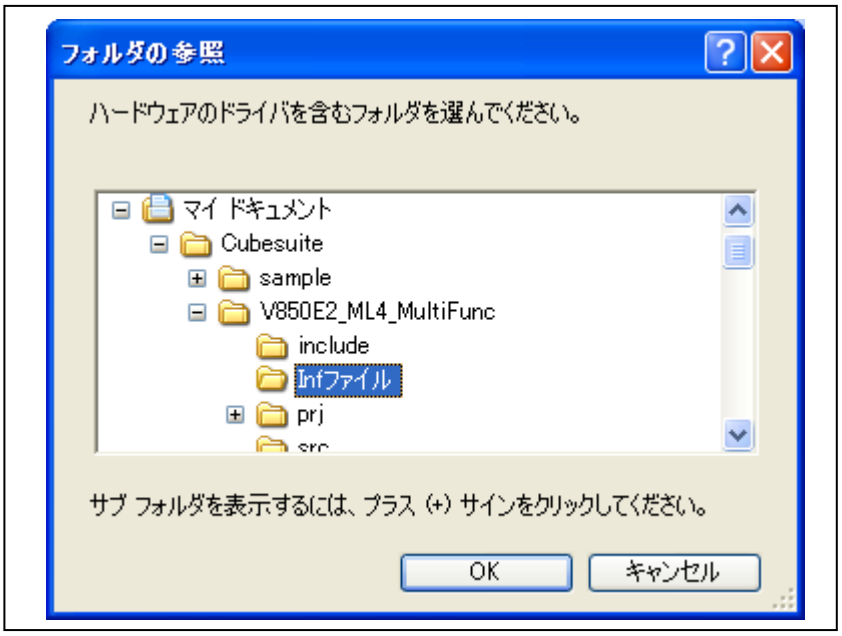


図 5－16　新しいハードウェアの検出ウィザード(3)

(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」と「USB 複合デバイス」が表示されていることを確認します。「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。「ポート(COM と LPT)」のツリーを展開し、「Renesas Electronics V850E2/ML4 Virtual UART(COM)」が表示されていることを確認します。

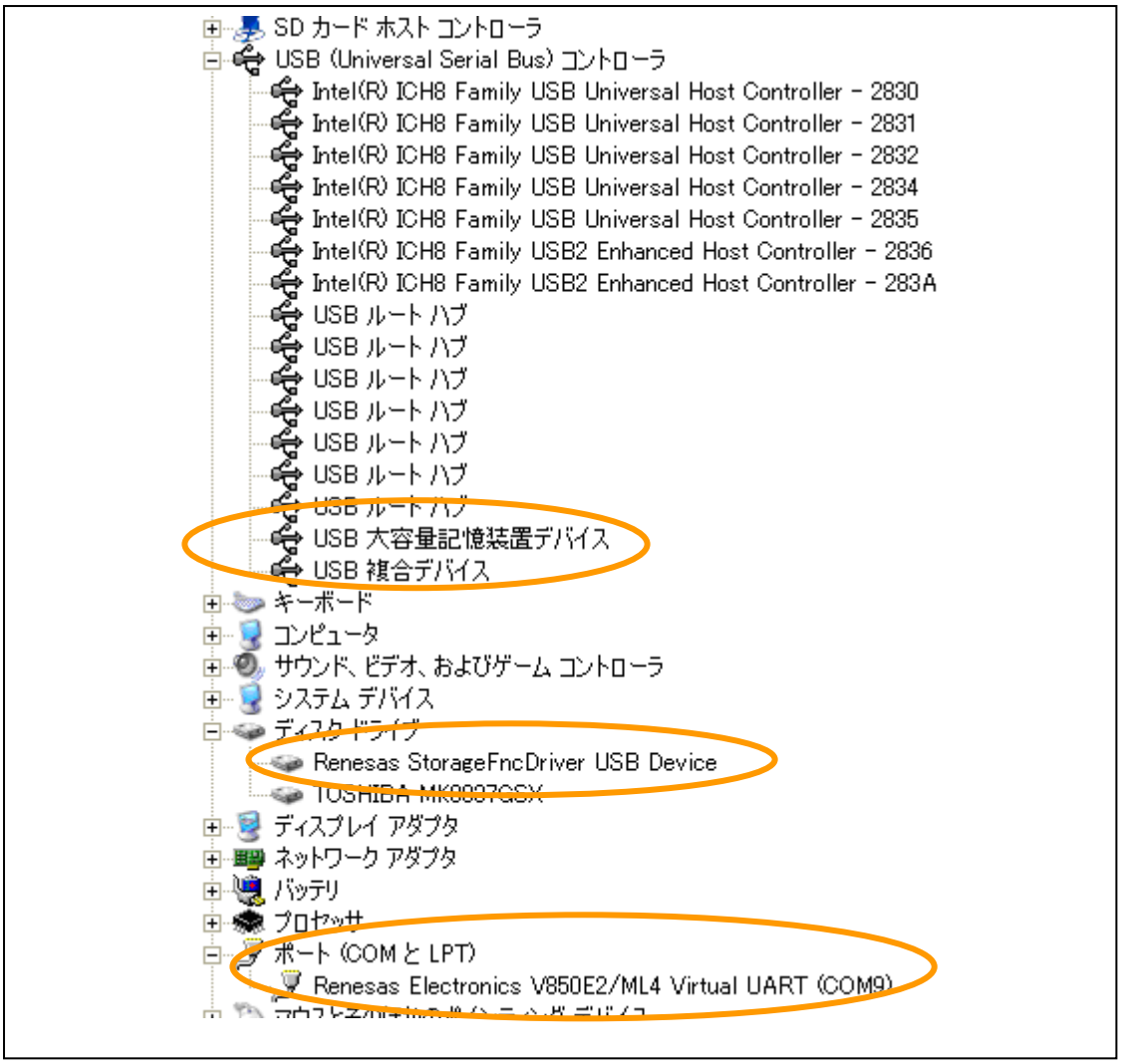


図 5−17　デバイス・マネージャ確認

(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。

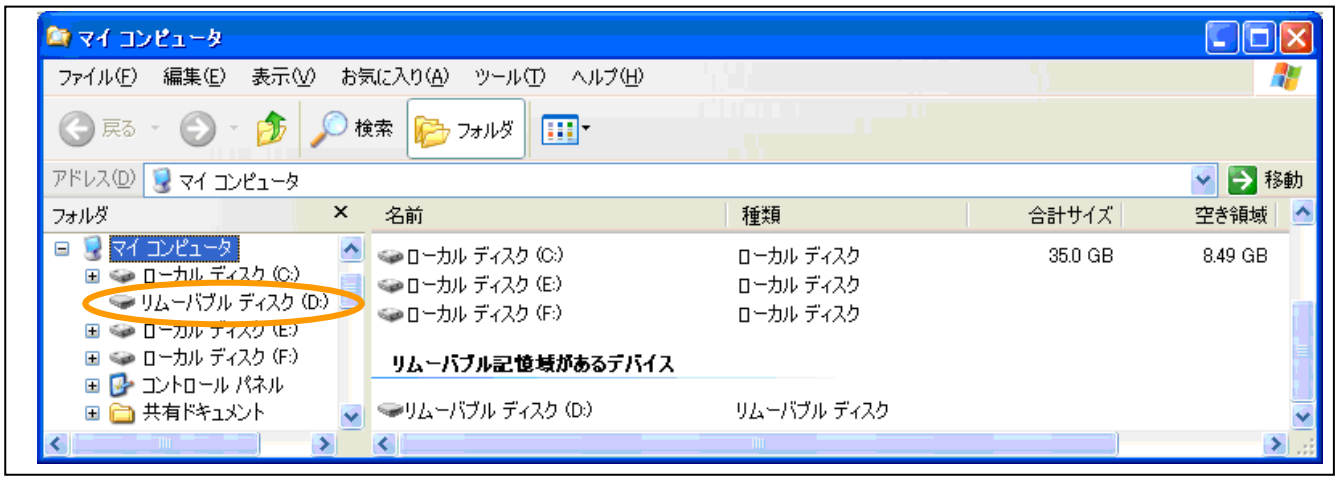


図 5−18　リムーバブル ディスクの確認

（備考）この画面例の「（D:）」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。

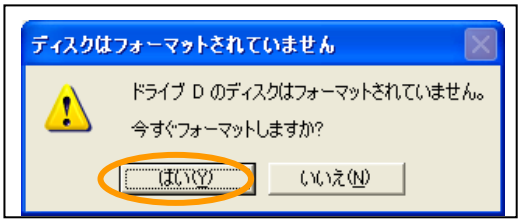


図 5−19　フォーマットの確認ダイアログ

<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。

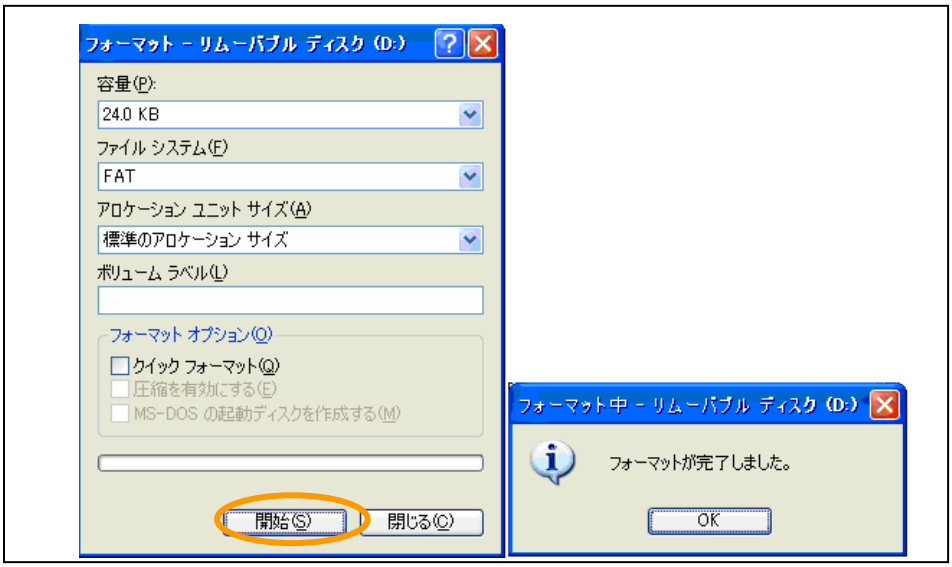


図 5−20　フォーマットメニューと完了ダイアログ

<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 MSC Test フォルダを用意します。

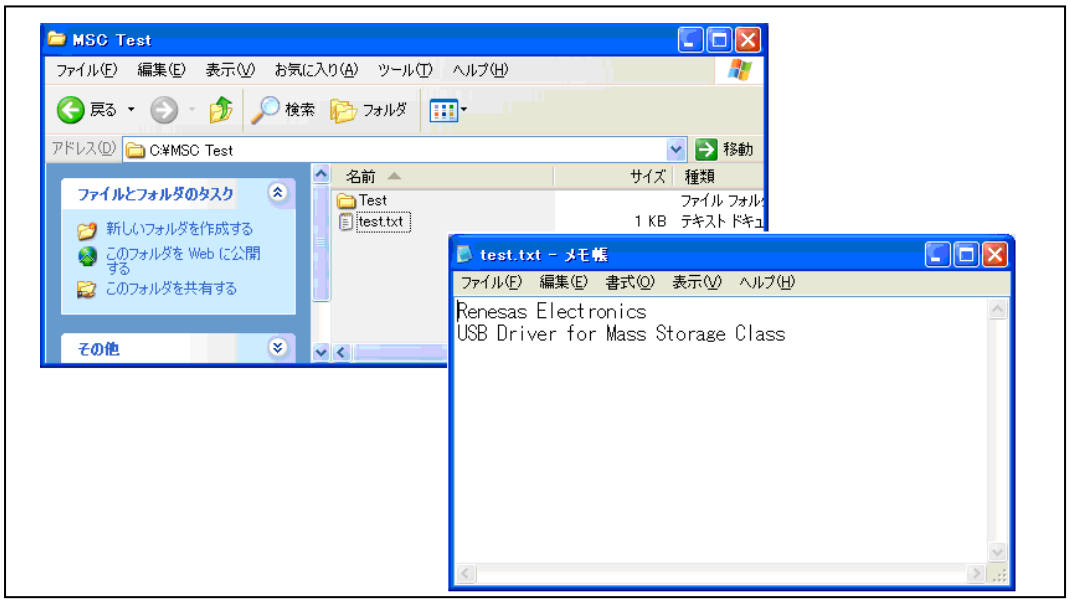


図 5−21　テスト用フォルダとテストデータファイル

<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。

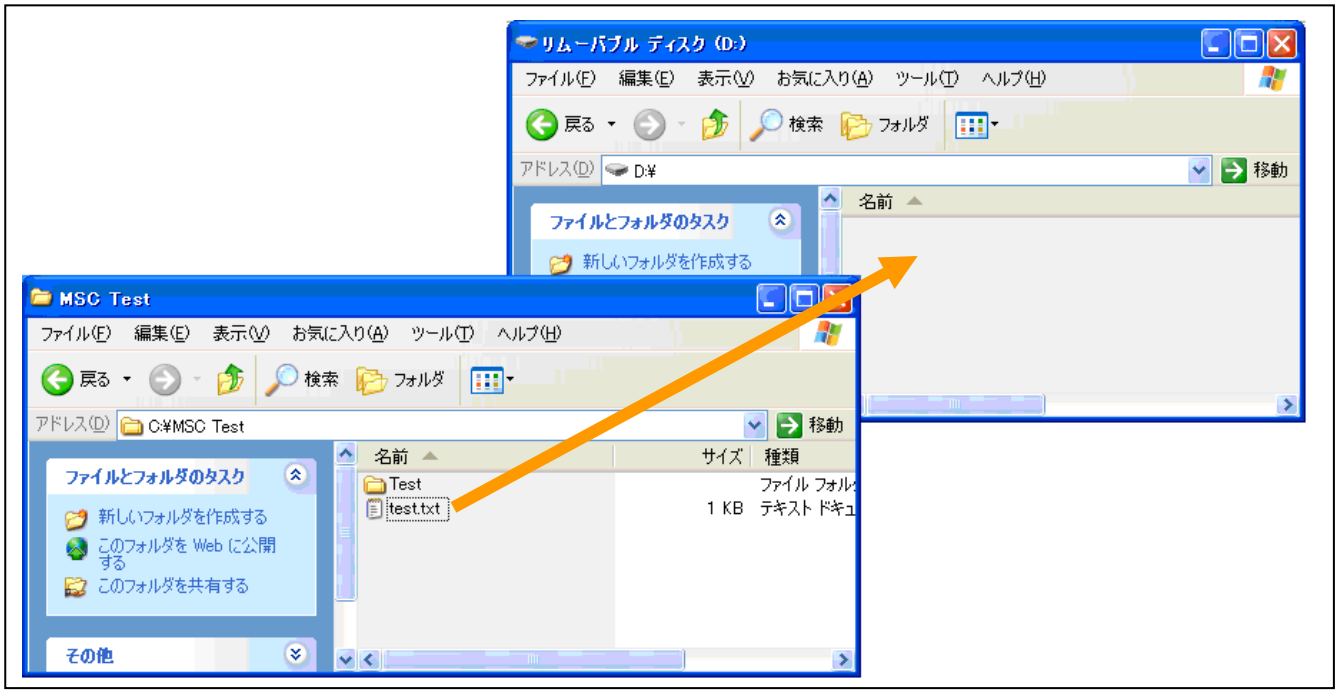


図 5−22　テストデータファイルのコピー

<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。

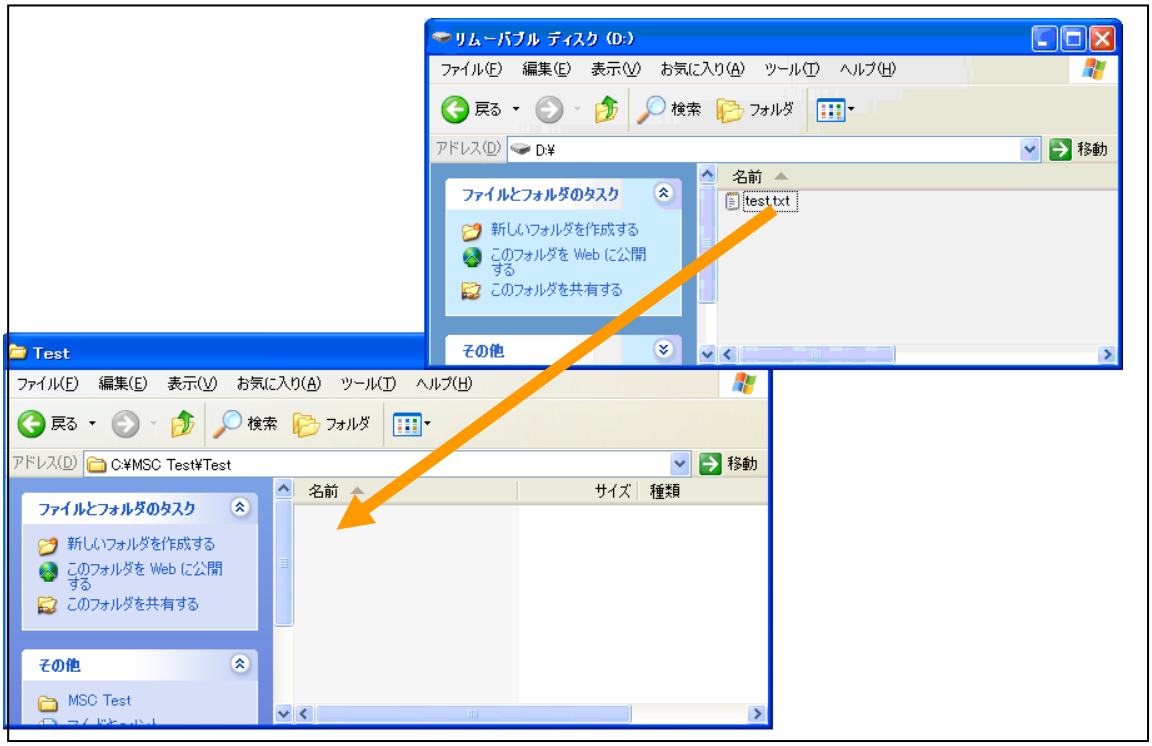


図 5−23　テストデータファイルの再コピー

<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。

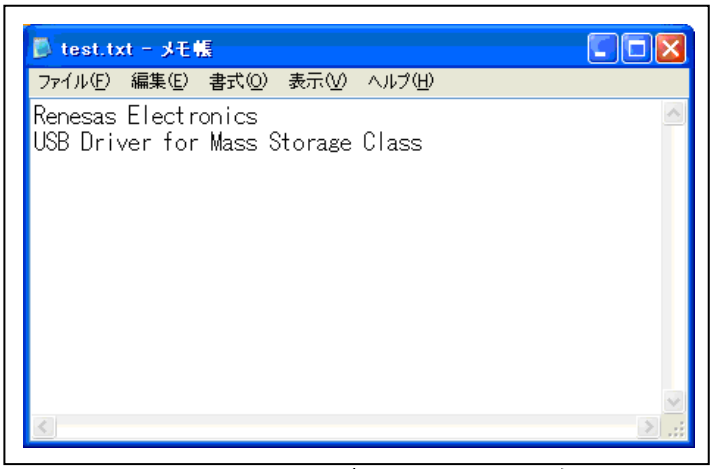


図 5−24　テストデータファイルの確認

（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) COM ポートの動作確認

ホスト上でターミナル・ソフトウエア (Tera Term など) を起動し，「Renesas Electronics V850E2/ML4 Virtual UART(COM)」として認識した COM ポートをオープンしてください。サンプル・アプリケーションは受信したデータをコールバックするプログラムです。ターミナルに入力したキーがコールバックされて表示される事を確認してください。

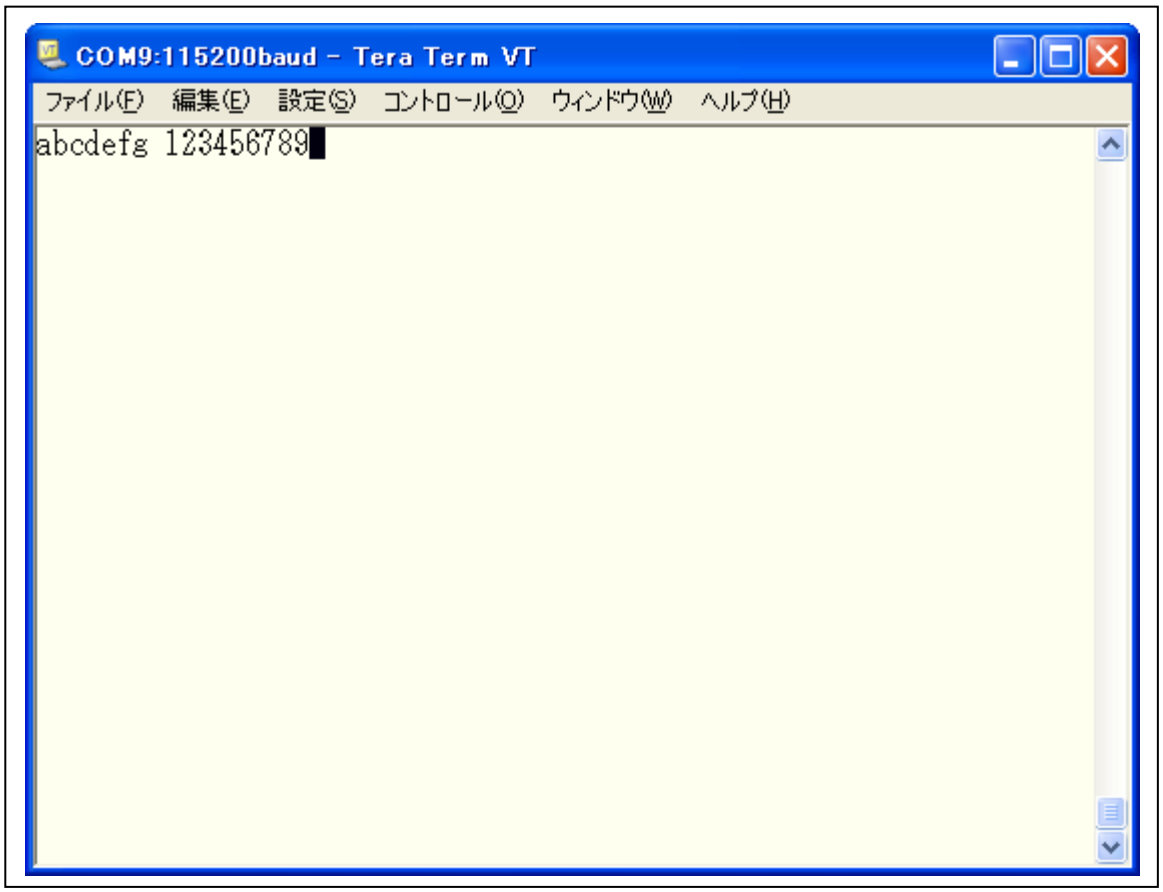


図 5－25　COM ポートの動作確認

6. V850E2/ML4 CPUボード概説

6.1 V850E2/ML4 CPUボード概要

V850E2/ML4 CPU ボードおよび各オプションボードは、ルネサスエレクトロニクスオリジナルマイクロコンピュータ V850E2/ML4 の機能・性能評価及び、アプリケーションソフトウエアの開発・評価を行なうための評価ボードです。

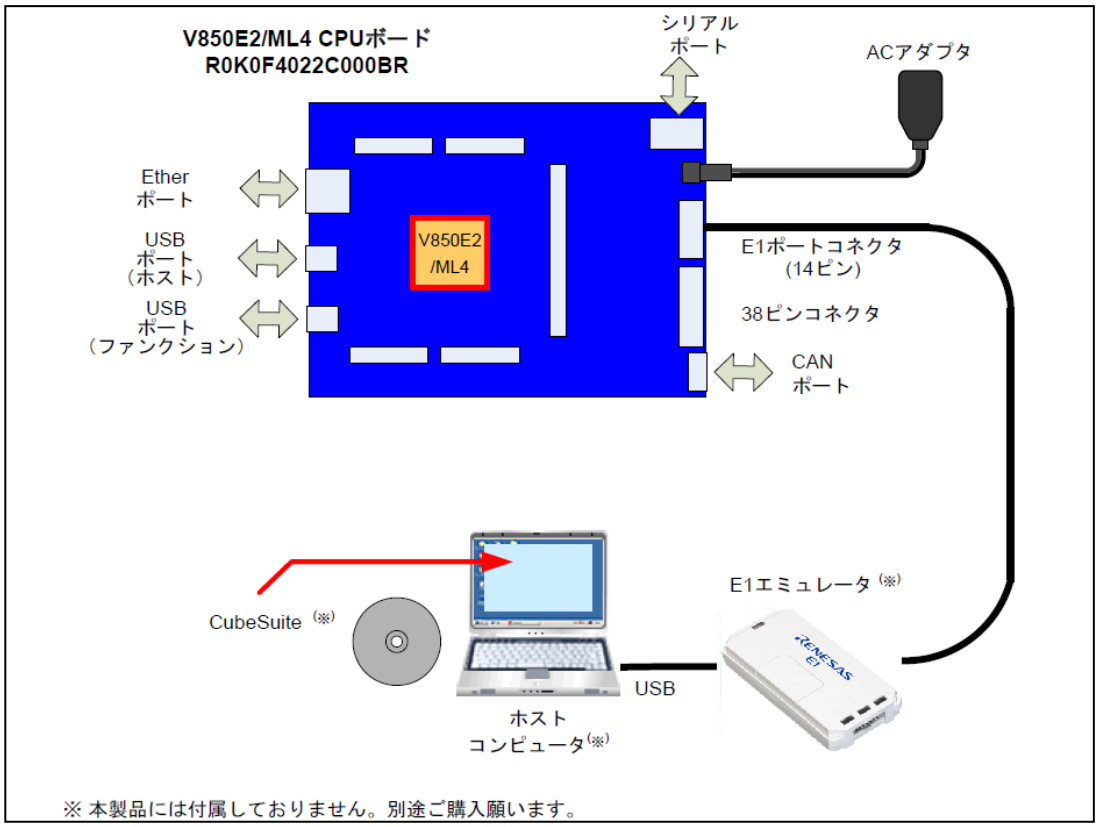


図 6－1　V850E2/ML4 CPU ボードの接続イメージ

6.2 V850E2/ML4 CPUボード特徴

V850E2/ML4 CPU ボード(ボード型名:R0K0F4022C000BR)には次のような特徴があります。

- 外部メモリとして、16M バイトの SDRAM 1 個（16 ビットバス接続）、16K バイトの EEPROM 1 個を標準搭載しています。
- V850E2/ML4 周辺機能インタフェースとして、RS-232C、USB コネクタ、ETHERNET コネクタ、CAN コネクタ を標準搭載しています。
- USB コネクタは、シリーズ A レセプタクルを標準搭載しています。また、USB ホストモジュールの評価用に Mini-B レセプタクル実装が可能な基板パターンになっています。
- V850E2/ML4 のデータバス、アドレスバス、内蔵周辺機能の端子は拡張コネクタへ接続しており、計測機器を用いた周辺デバイスとのタイミング評価や、開発用途に合わせた拡張ボードの開発が可能です。
- ルネサスエレクトロニクス製オンチップエミュレータ E1（14 ピンコネクタ）の使用が可能です。

6.3　主な仕様

V850E2/ML4 CPU ボードの主な仕様は次のとおりです。

○CPU	μPD70F4022（V850E2/ML4）
○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 mini-B タイプ×1 基）
	UART コネクタ搭載
	CAN コネクタ搭載
	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® 7, Vista, XP
○動作電圧	5.0 V
○本体寸法	125×170（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/contact/

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2012.02.24	—	初版発行

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。