お客様各位

カタログ等資料中の旧社名の扱いについて

2010年4月1日を以ってNECエレクトロニクス株式会社及び株式会社ルネサステクノロジが合併し、両社の全ての事業が当社に承継されております。従いまして、本資料中には旧社名での表記が残っておりますが、当社の資料として有効ですので、ご理解の程宜しくお願い申し上げます。

ルネサスエレクトロニクス ホームページ（http://www.renesas.com）

2010年4月1日
ルネサスエレクトロニクス株式会社

【発行】ルネサスエレクトロニクス株式会社（http://www.renesas.com）

【問い合わせ先】http://japan.renesas.com/inquiry

ご注意書き

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

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

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

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

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

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

アプリケーション・ノート

NEC

V850E/ME2

32ビット・シングルチップ・マイクロコンピュータ

USBファンクション・ドライバ編

μPD703111A

資料番号　U17069JJ1V0AN00（第1版）
発行年月　March 2004 NS　CP(K)

〔メ　モ〕

目次要約

CMOSデバイスの一般的注意事項

① 入力端子の印加波形

入力ノイズや反射波による波形歪みは誤動作の原因になりますので注意してください。

CMOSデバイスの入力がノイズなどに起因して，V_{IL}（MAX.）からV_{IH}（MIN.）までの領域にとどまるような場合は，誤動作を引き起こす恐れがあります。入力レベルが固定な場合はもちろん，V_{IL}（MAX.）からV_{IH}（MIN.）までの領域を通過する遷移期間中にチャタリングノイズ等が入らないようご使用ください。

② 未使用入力の処理

CMOSデバイスの未使用端子の入力レベルは固定してください。

未使用端子入力については，CMOSデバイスの入力に何も接続しない状態で動作させるのではなく，プルアップかプルダウンによって入力レベルを固定してください。また，未使用の入出力端子が出力となる可能性（タイミングは規定しません）を考慮すると，個別に抵抗を介してV_{DD}またはGNDに接続することが有効です。

資料中に「未使用端子の処理」について記載のある製品については，その内容を守ってください。

③ 静電気対策

MOSデバイス取り扱いの際は静電気防止を心がけてください。

MOSデバイスは強い静電気によってゲート絶縁破壊を生じることがあります。運搬や保存の際には，当社が出荷梱包に使用している導電性のトレーやマガジン・ケース，または導電性の緩衝材，金属ケースなどを利用し，組み立て工程にはアースを施してください。プラスチック板上に放置したり，端子を触ったりしないでください。

また，MOSデバイスを実装したボードについても同様の扱いをしてください。

④ 初期化以前の状態

電源投入時，MOSデバイスの初期状態は不定です。

電源投入時の端子の出力状態や入出力設定，レジスタ内容などは保証しておりません。ただし，リセット動作やモード設定で定義している項目については，これらの動作ののちに保証の対象となります。

リセット機能を持つデバイスの電源投入後は，まずリセット動作を実行してください。

- 本資料に記載されている内容は2004年3月現在のもので，今後，予告なく変更することがあります。量産設計の際には最新の個別データ・シート等をご参照ください。
- 文書による当社の事前の承諾なしに本資料の転載複製を禁じます。当社は，本資料の誤りに関し，一切その責を負いません。
- 当社は，本資料に記載された当社製品の使用に関連し発生した第三者の特許権，著作権その他の知的財産権の侵害等に関し，一切その責を負いません。当社は，本資料に基づき当社または第三者の特許権，著作権その他の知的財産権を何ら許諾するものではありません。
- 本資料に記載された回路，ソフトウエアおよびこれらに関する情報は，半導体製品の動作例，応用例を説明するものです。お客様の機器の設計において，回路，ソフトウエアおよびこれらに関する情報を使用する場合には，お客様の責任において行ってください。これらの使用に起因しお客様または第三者に生じた損害に関し，当社は，一切その責を負いません。
- 当社は，当社製品の品質，信頼性の向上に努めておりますが，当社製品の不具合が完全に発生しないことを保証するものではありません。当社製品の不具合により生じた生命，身体および財産に対する損害の危険を最小限度にするために，冗長設計，延焼対策設計，誤動作防止設計等安全設計を行ってください。
- 当社は，当社製品の品質水準を「標準水準」，「特別水準」およびお客様に品質保証プログラムを指定していただく「特定水準」に分類しております。また，各品質水準は，以下に示す用途に製品が使われることを意図しておりますので，当社製品の品質水準をご確認ください。

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

  特別水準：輸送機器（自動車，電車，船舶等），交通用信号機器，防災・防犯装置，各種安全装置，生命維持を目的として設計されていない医療機器

  特定水準：航空機器，航空宇宙機器，海底中継機器，原子力制御システム，生命維持のための医療機器，生命維持のための装置またはシステム等

  当社製品のデータ・シート，データ・ブック等の資料で特に品質水準の表示がない場合は，標準水準製品であることを表します。意図されていない用途で当社製品の使用をお客様が希望する場合には，事前に当社販売窓口までお問い合わせください。

（注）

（1）本事項において使用されている「当社」とは，NECエレクトロニクス株式会社およびNECエレクトロニクス株式会社がその総株主の議決権の過半数を直接または間接に保有する会社をいう。

（2）本事項において使用されている「当社製品」とは，（1）において定義された当社の開発，製造製品をいう。

M8E　02.11

はじめに

対 象 者　このアプリケーション・ノートは，V850E/ME2の機能を理解し，それらを使用した応用システムを設計するユーザを対象とします。

目　　的　このアプリケーション・ノートでは，V850E/ME2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・プログラムとして3種類を取り上げ，その構成をユーザに理解していただくことを目的としています。

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。

- ●V850E/ME2の概要
- ●USBバス・ドライバ
- ●USBストレージ・クラス用ドライバ
- ●USBコミュニケーション・クラス用ドライバ

備考　各種ドライバのサンプル・プログラムは，次のホーム・ページから入手できます。

http://www.necel.com/micro/v850/devicedata/index.html#SAMPLE

読 み 方　このマニュアルの読者には，電気，論理回路，およびマイクロコンピュータに関する一般知識を必要とします。

V850E/ME2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V850E/ME2　**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。

V850E/ME2の命令機能を知りたいとき
→別冊のV850E1　**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。

このマニュアルでは，「xxxレジスタのyyyビット」を「xxx.yyyビット」と表記しています。ただし，プログラムにそのまま「xxx.yyy」と記述しても，コンパイラ／アセンブラでは正しく認識できませんので，注意してください。

凡　　例

データ表記の重み	：左が上位桁，右が下位桁
アクティブ・ロウの表記	：$\overline{\text{xxx}}$（端子，信号名称に上線）または ／xxx（信号名称の前に“／”記号）
メモリ・マップのアドレス	：上部－上位，下部－下位
注	：本文中に付けた注の説明
注意	：気を付けて読んでいただきたい内容
備考	：本文の補足説明

数の表記　　：2進数 … xxxxまたはxxxxB

10進数 … xxxx

16進数 … xxxxH

2のべき数を示す接頭語（アドレス空間，メモリ容量）　：K（キロ） …2^{10} = 1024

M（メガ） …2^{20} = 1024^2

G（ギガ） …2^{30} = 1024^3

関連資料　関連資料は暫定版の場合がありますが，この資料では「暫定」の表示をしておりません。あらかじめご了承ください。

V850E/ME2に関する資料

資　料　名	資料番号
V850E1　ユーザーズ・マニュアル　アーキテクチャ編	U14559J
V850E/ME2　ユーザーズ・マニュアル　ハードウエア編	U16031J
V850E/ME2　アプリケーション・ノート　ハードウエア編	U16794J
V850E/ME2　アプリケーション・ノート　USBファンクション・ドライバ編	このマニュアル

開発ツールに関する資料（ユーザーズ・マニュアル）

資　料　名		資料番号
CA850　Ver.2.50　Cコンパイラ・パッケージ	操作編	U16053J
	C言語編	U16054J
	アセンブリ言語編	U16042J
PM plus　Ver.5.10		U16569J
ID850　Ver.2.50　統合ディバッガ	操作編	U16217J
ID850NW　Ver.2.51　統合ディバッガ	操作編	U16454J
RX850　Ver.3.13以上　リアルタイムOS	基礎編	U13430J
	インストレーション編	U13410J
	テクニカル編	U13431J
RX850 Pro　Ver.3.15　リアルタイムOS	基礎編	U13773J
	インストレーション編	U13774J
	テクニカル編	U13772J
RX-NET　TCP/IPライブラリ		U15083J
RD850　Ver.3.01　タスク・ディバッガ		U13737J
RD850 Pro　Ver.3.01　タスク・ディバッガ		U13916J
AZ850　Ver.3.0　システム・パフォーマンス・アナライザ		U14410J

目　　次

第1章　V850E/ME2の概要

V850E/ME2は，NECエレクトロニクスのシングルチップ・マイクロコンピュータ「V850シリーズ」の1製品です。この章では，V850E/ME2の概要を簡単に説明します。

1.1　概　　説

V850E/ME2は，システム・オン・チップ時代のシステムLSIの核となるASIC用32ビットRISC命令型CPUコア「V850E1 CPU」を搭載した32ビット・シングルチップ・マイクロコンピュータです。キャッシュ，データRAM，命令RAM，および，各種メモリ・コントローラ，DMAコントローラ，リアルタイム・パルス・ユニット，シリアル・インタフェース，USBファンクション・コントローラ（USBF），A/Dコンバータなどの周辺機能を内蔵し，大容量データ処理と高度なリアルタイム制御を実現します。

1.2 特　　徴

○命令数　　83

○最小命令実行時間　　10 ns/7.5 ns/6.7 ns（内部100 MHz/133 MHz/150 MHz動作時）

○汎用レジスタ　　32ビット×32本

○命令セット　　V850E1 CPU
符号付き乗算（16ビット×16ビット→32ビット，または32ビット×32ビット→64ビット）：1-2クロック
飽和演算命令（オーバフロー／アンダフロー検出機能付き）
32ビット・シフト命令：1クロック
ビット操作命令
ロング／ショート形式を持つロード／ストア命令
符号付きロード命令

○メモリ空間　　256 Mバイト・リニア・アドレス空間（プログラム／データ共有）
チップ・セレクト出力機能：8空間
メモリ・ブロック分割機能：2 M, 4 M, 6 M, 8 M, 64 Mバイト／ブロック
プログラマブル・ウエイト機能
アイドル・ステート挿入機能

○外部バス・インタフェース
32ビット・データ・バス（アドレス／データ分離型バス）
32/16/8ビット・バス・サイジング機能
外部バス分周機能：1/1,1/2,1/3,1/4（66 MHz（MAX.））
バス・ホールド機能
外部ウエイト機能
アドレス・セットアップ・ウエイト機能
エンディアン制御機能

○内蔵メモリ　　命令RAM：128 Kバイト
データRAM：16 Kバイト

○命令キャッシュ　　8 Kバイト2ウエイ・セット・アソシアティブ

○割り込み／例外　　外部割り込み　：40本（NMI含む）
内部割り込み　：59要因
例外　：2要因
8レベルの優先順位指定可能

○メモリ・アクセス制御

DRAMコントローラ（SDRAMに対応）
ページROMコントローラ
先読み／ライト・バッファ機能

○DMAコントローラ　4チャネル構成
転送単位　：8ビット／16ビット／32ビット
最大転送回数：65,536（2^{16}）回
転送タイプ　：フライバイ（1サイクル）転送／2サイクル転送
転送モード　：シングル転送／シングルステップ転送／ブロック転送
転送対象　：メモリ←→メモリ，メモリ←→I/O
転送要求　：外部要求／内蔵周辺I/O／ソフトウエア
DMA転送終了（ターミナル・カウント）出力信号
ネクスト・アドレス設定機能

○I/Oライン　入力ポート　：1
入出力ポート：77

○リアルタイム・パルス・ユニット
16ビット・タイマ／イベント・カウンタ：6 ch（2 chはキャプチャ動作なし）
16ビット・タイマ：6本
16ビット・キャプチャ／コンペア・レジスタ：12本
16ビット・インターバル・タイマ：4 ch
2相エンコーダ入力用16ビット・アップ／ダウン・カウンタ／タイマ：2 ch
16ビット・キャプチャ／コンペア・レジスタ：4本
16ビット・コンペア・レジスタ：4本

○シリアル・インタフェース（SIO）
アシンクロナス・シリアル・インタフェースB（UARTB）
クロック同期式シリアル・インタフェース3（CSI3）
CSI3/UARTB　：1 ch
UARTB　：1 ch
CSI3　：1 ch
USBファンクション・コントローラ（USBF）：1 ch
フルスピード（12 Mbps）
エンドポイント　コントロール転送：64バイト×2本
インタラプト転送：8バイト×2本
バルク転送（IN）：64バイト×2バンク×2本
バルク転送（OUT）：64バイト×2バンク×2本

○A/Dコンバータ　10ビット分解能A/Dコンバータ：8 ch

○PWM（Pulse Width Modulation）

16ビット分解能PWM：2 ch

○クロック・ジェネレータ

SSCGによる8逓倍機能

○パワー・セーブ機能

HALT／IDLE／ソフトウエアSTOPモード

○パッケージ　176ピン・プラスチックLQFP（ファインピッチ）（24×24）

240ピン・プラスチックFBGA（16×16）

○CMOS構造　完全スタティック回路

1.3　オーダ情報

品　名	パッケージ	最高動作周波数
μPD703111AGM-10-UEU	176ピン・プラスチックLQFP（ファインピッチ）（24×24）	100 MHz
μPD703111AGM-13-UEU	〃	133 MHz
μPD703111AGM-15-UEU	〃	150 MHz
μPD703111AF1-10-GA3	240ピン・プラスチックFBGA（16×16）	100 MHz
μPD703111AF1-13-GA3	〃	133 MHz
μPD703111AF1-15-GA3	〃	150 MHz

1. 4　端子接続図

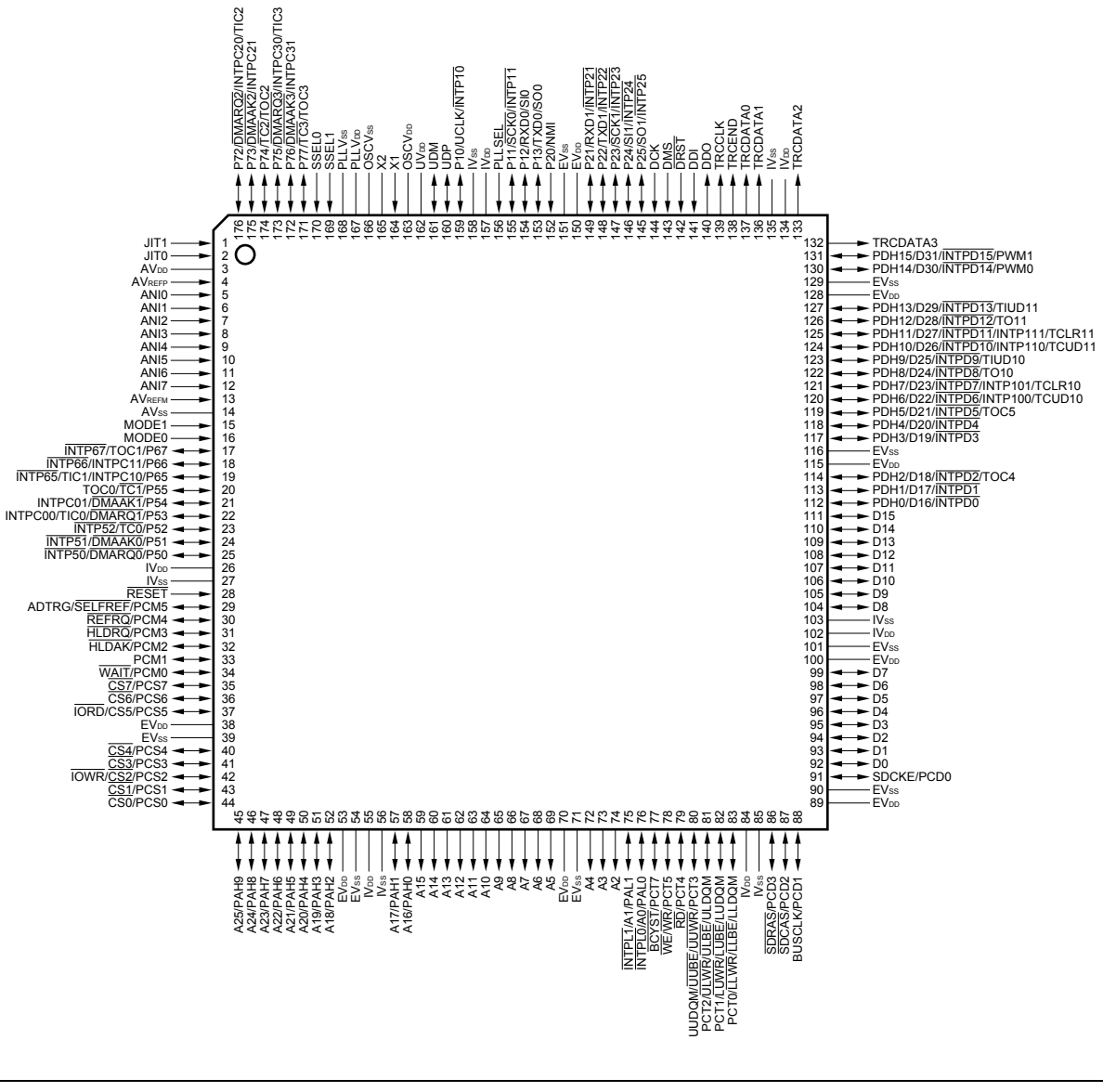
・176ピン・プラスチックLQFP（ファインピッチ）（24×24）

μPD703111AGM-10-UEU

μPD703111AGM-13-UEU

μPD703111AGM-15-UEU

Top View

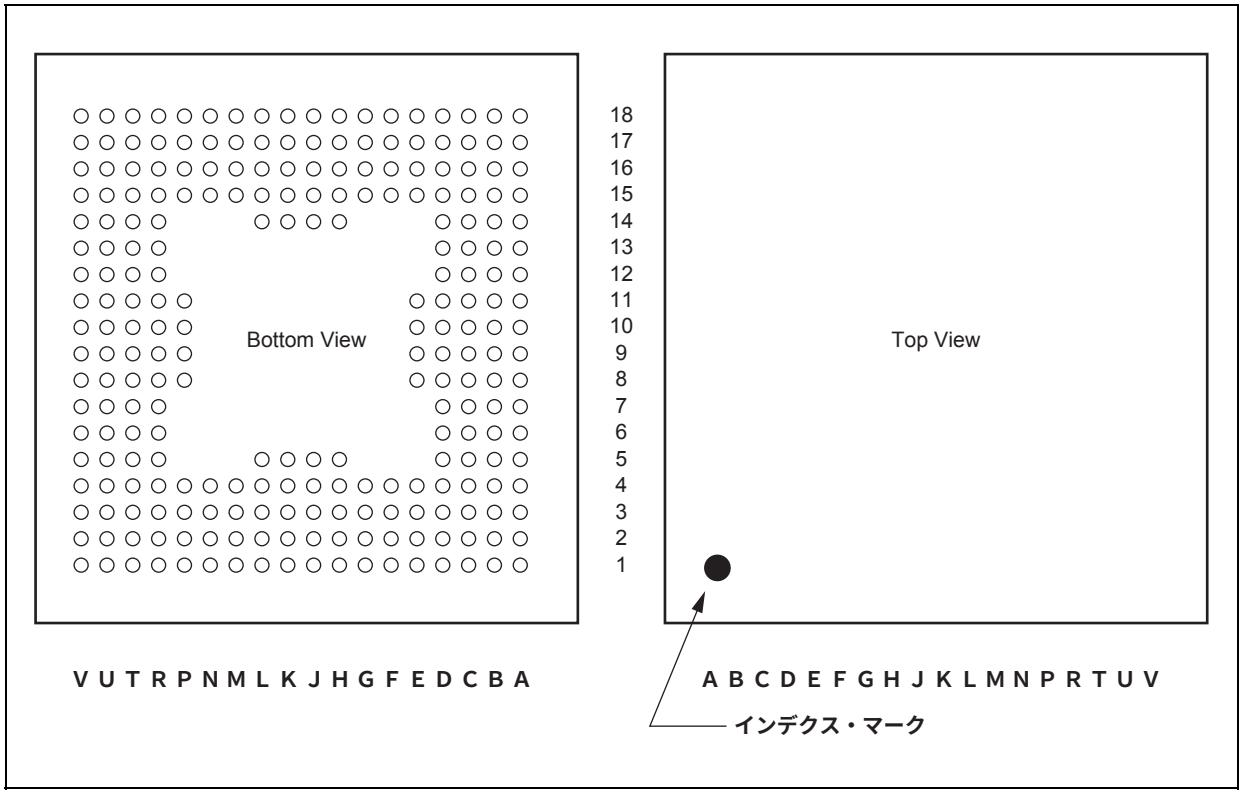


・240ピン・プラスチックFBGA（16×16）

μPD703111AF1-10-GA3

μPD703111AF1-13-GA3

μPD703111AF1-15-GA3



(1/2)

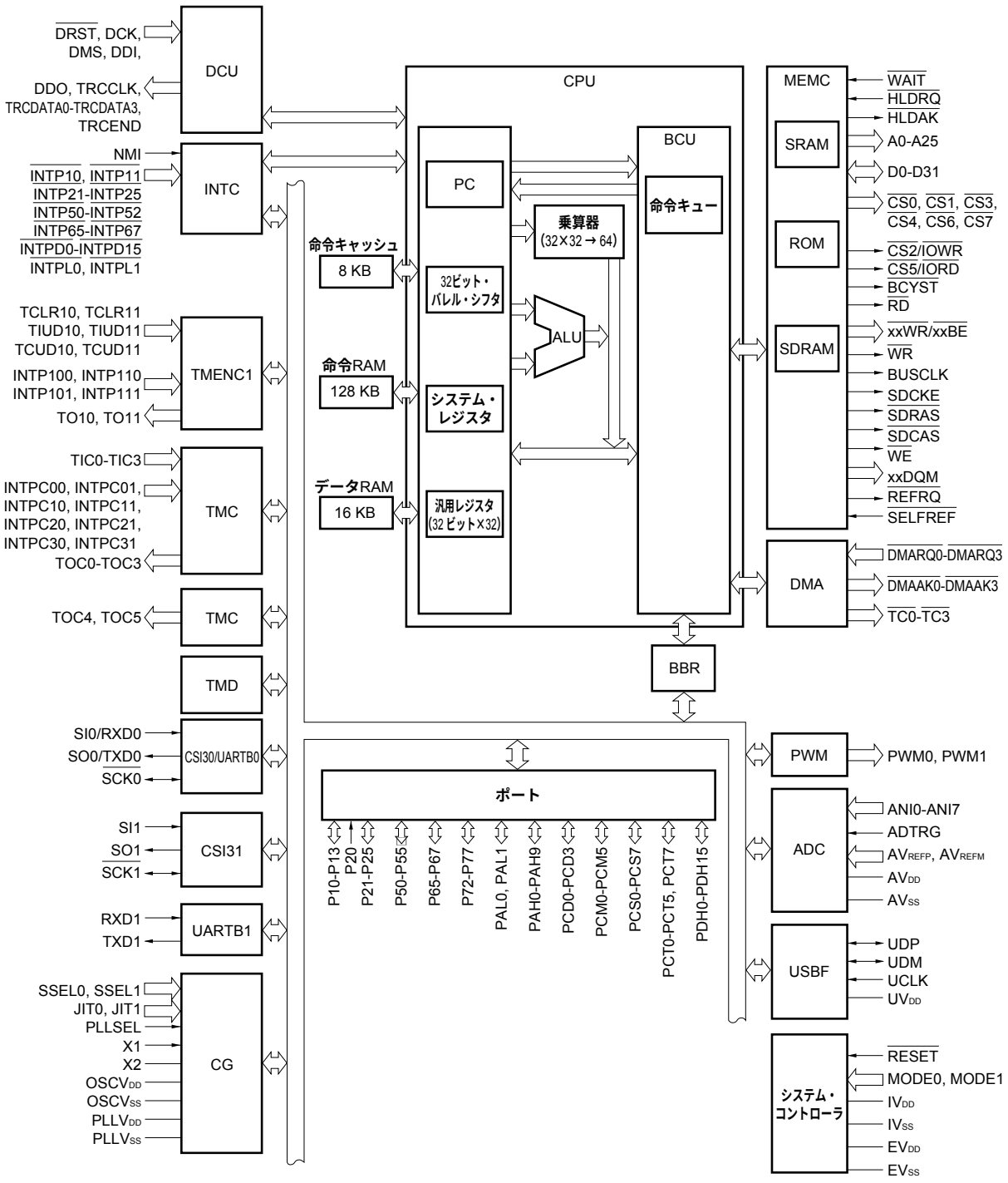
ピン番号	名　称	ピン番号	名　称	ピン番号	名　称
A1	−	C12	IV_{DD}	G3	EV_{SS}
A2	IV_{SS}	C13	PAH2/A18	G4	D7
A3	PCT0/$\overline{\text{LLWR}}$/$\overline{\text{LLBE}}$/LLDQM	C14	PAH4/A20	G15	PCM1
A4	−	C15	PAH6/A22	G16	PCM3/$\overline{\text{HLDRQ}}$
A5	PCT4/$\overline{\text{RD}}$	C16	−	G17	PCM4/$\overline{\text{REFRQ}}$
A6	−	C17	PCS0/$\overline{\text{CS0}}$	G18	PCM5/ADTRG/$\overline{\text{SELFREF}}$
A7	−	C18	−	H1	−
A8	EV_{DD}	D1	D0	H2	D8
A9	A9	D2	EV_{SS}	H3	D9
A10	−	D3	PCD0/SDCKE	H4	D10
A11	A14	D4	EV_{DD}	H5	IV_{SS}
A12	IV_{SS}	D5	PCT1/$\overline{\text{LUWR}}$/$\overline{\text{LUBE}}$/LUDQM	H14	−
A13	EV_{DD}	D6	−	H15	$\overline{\text{RESET}}$
A14	−	D7	PAL0/$\overline{\text{INTPL0}}$/A0	H16	IV_{SS}
A15	PAH5/A21	D8	A4	H17	−
A16	PAH7/A23	D9	A6	H18	IV_{DD}
A17	PAH9/A25	D10	−	J1	−
A18	−	D11	A13	J2	D11
B1	−	D12	EV_{SS}	J3	D12
B2	PCD1/BUSCLK	D13	PAH3/A19	J4	−
B3	PCD2/$\overline{\text{SDCAS}}$	D14	−	J5	D13
B4	−	D15	−	J14	−
B5	PCT3/$\overline{\text{UUWR}}$/$\overline{\text{UUBE}}$/UUDQM	D16	PCS2/$\overline{\text{CS2}}$/$\overline{\text{IOWR}}$	J15	P50/$\overline{\text{INTP50}}$/$\overline{\text{DMARQ0}}$
B6	PCT7/$\overline{\text{BCYST}}$	D17	PCS3/$\overline{\text{CS3}}$	J16	P51/$\overline{\text{INTP51}}$/$\overline{\text{DMAAK0}}$
B7	A2	D18	EV_{DD}	J17	P52/$\overline{\text{INTP52}}$/$\overline{\text{TC0}}$
B8	−	E1	D3	J18	P53/INTPC00/TIC0/$\overline{\text{DMARQ1}}$
B9	A8	E2	D2	K1	D14
B10	A12	E3	D1	K2	D15
B11	PAH0/A16	E4	−	K3	PDH0/D16/$\overline{\text{INTPD0}}$
B12	−	E8	A3	K4	PDH1/D17/$\overline{\text{INTPD1}}$
B13	−	E9	A5	K5	PDH2/D18/$\overline{\text{INTPD2}}$/TOC4
B14	−	E10	A10	K14	P55/TOC0/$\overline{\text{TC1}}$
B15	−	E11	PAH1/A17	K15	P54/INTPC01/$\overline{\text{DMAAK1}}$
B16	PAH8/A24	E15	PCS4/$\overline{\text{CS4}}$	K16	P65/$\overline{\text{INTP65}}$/INTPC10/TIC1
B17	−	E16	EV_{SS}	K17	P66/$\overline{\text{INTP66}}$/INTPC11
B18	PCS1/$\overline{\text{CS1}}$	E17	PCS5/$\overline{\text{CS5}}$/$\overline{\text{IORD}}$	K18	−
C1	−	E18	PCS6/$\overline{\text{CS6}}$	L1	EV_{DD}
C2	−	F1	D6	L2	−
C3	PCD3/$\overline{\text{SDRAS}}$	F2	D5	L3	EV_{SS}
C4	IV_{DD}	F3	D4	L4	PDH3/D19/$\overline{\text{INTPD3}}$
C5	PCT2/$\overline{\text{ULWR}}$/$\overline{\text{ULBE}}$/ULDQM	F4	−	L5	PDH4/D20/$\overline{\text{INTPD4}}$
C6	PCT5/$\overline{\text{WE}}$/$\overline{\text{WR}}$	F15	−	L14	MODE1
C7	PAL1/$\overline{\text{INTPL1}}$/A1	F16	PCS7/$\overline{\text{CS7}}$	L15	−
C8	EV_{SS}	F17	PCM0/$\overline{\text{WAIT}}$	L16	MODE0
C9	A7	F18	PCM2/$\overline{\text{HLDAK}}$	L17	−
C10	A11	G1	IV_{DD}	L18	P67/$\overline{\text{INTP67}}$/TOC1
C11	A15	G2	EV_{DD}	M1	−

(2/2)

ピン番号	名　称	ピン番号	名　称	ピン番号	名　称
M2	PDH5/D21/INTPD5/TOC5	R7	DCK	U4	−
M3	PDH6/D22/INTPD6/INTP100/TCUD10	R8	EV_{DD}	U5	TRCCLK
M4	−	R9	P11/INTP11/SCK0	U6	DRST
M15	ANI6	R10	IV_{SS}	U7	P25/INTP25/SO1
M16	AV_{REFM}	R11	UDM	U8	P22/INTP22/TXD1
M17	ANI7	R12	X2	U9	EV_{SS}
M18	AV_{SS}	R13	$PLLV_{DD}$	U10	IV_{DD}
N1	PDH7/D23/INTPD7/INTP101/TCLR10	R14	SSEL0	U11	−
N2	PDH8/D24/INTPD8/TO10	R15	−	U12	$OSCV_{DD}$
N3	PDH9/D25/INTPD9/TIUD10	R16	AV_{REFP}	U13	−
N4	PDH10/D26/INTPD10/INTP110/TCUD11	R17	AV_{DD}	U14	−
N15	ANI2	R18	−	U15	P76/INTPC31/DMAAK3
N16	ANI3	T1	EV_{DD}	U16	P73/INTPC21/DMAAK2
N17	ANI4	T2	TRCDATA3	U17	P72/INTPC20/TIC2/DMARQ2
N18	ANI5	T3	−	U18	−
P1	−	T4	TRCDATA1	V1	−
P2	PDH11/D27/INTPD11/INTP111/TCLR11	T5	TRCEND	V2	TRCDATA2
P3	PDH13/D29/INTPD13/TIUD11	T6	DDI	V3	IV_{SS}
P4	−	T7	−	V4	TRCDATA0
P8	P23/INTP23/SCK1	T8	P21/INTP21/RXD1	V5	−
P9	P12/SI0/RXD0	T9	P20/NMI	V6	DMS
P10	−	T10	−	V7	P24/INTP24/SI1
P11	UV_{DD}	T11	UDP	V8	−
P15	−	T12	X1	V9	P13/SO0/TXD0
P16	ANI0	T13	$OSCV_{SS}$	V10	PLLSEL
P17	ANI1	T14	SSEL1	V11	P10/INTP10/UCLK
P18	−	T15	P75/INTPC30/TIC3/DMARQ3	V12	−
R1	PDH12/D28/INTPD12/TO11	T16	−	V13	−
R2	EV_{SS}	T17	JIT1	V14	−
R3	PDH14/D30/INTPD14/PWM0	T18	JIT0	V15	$PLLV_{SS}$
R4	IV_{DD}	U1	PDH15/D31/INTPD15/PWM1	V16	P77/TOC3/TC3
R5	−	U2	−	V17	P74/TOC2/TC2
R6	DDO	U3	−	V18	−

備考　A1, A4, A6, A7, A10, A14, A18, B1, B4, B8, B12-B15, B17, C1, C2, C16, C18, D6, D10, D14, D15, E4, F4, F15, H1, H14, H17, J1, J4, J14, K18, L2, L15, L17, M1, M4, P1, P4, P10, P15, P18, R5, R15, R18, T3, T7, T10, T16, U2-U4, U11, U13, U14, U18, V1, V5, V8, V12-V14, V18**の端子は，オープンにしてください。**

1. 5　内部ブロック図



備考　xx：LL, LU, UL, UU

1. 6　内蔵メモリ

V850E/ME2は，128 Kバイト（64 Kバイト×2バンク）の命令RAM，8 Kバイト（2ウエイ・セット・アソシアティブ）の命令キャッシュ，16 KバイトのデータRAMを内蔵しています。

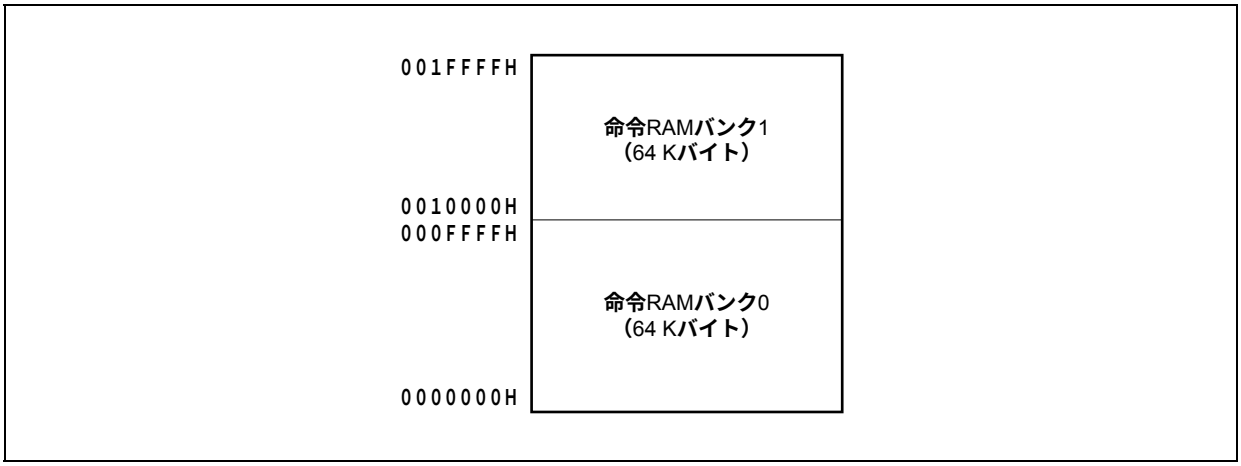
1. 6. 1　内蔵命令RAM

内蔵命令RAMには，リード・モードとライト・モードの2つのモードがあり，内蔵命令RAMモード・レジスタ（IRAMM）で設定します。

リセット後はライト・モードに初期化されます。したがって，プログラムまたはDMAコントローラにより，内蔵命令RAMへ命令データを転送してからリード・モードを設定します。また，リード・モード時には，内部システム・クロック×1クロックで命令をフェッチします。

注意　内蔵命令RAMバンク0には，リセット以外のすべての割り込み／例外ハンドラが配列されています。内蔵命令RAMバンク0へのライト動作が完了するまでは，すべての割り込み／例外を発生させないようにしてください。

図1－1　内蔵命令RAMのメモリ・マップ



1.6.2　命令キャッシュ機能

$\overline{CS0}$-$\overline{CS2}$空間がキャッシャブル領域です。キャッシュ・コンフィギュレーション・レジスタ（BHC）により，各空間をキャッシュ可能領域にするか，キャッシュ不可能領域にするかを設定します。また，命令キャッシュ・コントロール・レジスタ（ICC）により，ウエイ0のキャッシュ・ロック状態，ウエイ0のオートフィル，ウエイ0, 1のタグ・クリアを設定します。

注意1. BHCレジスタへの書き込みはリセット後に行ってください。書き込み後は，値を変更しないでください。

2. 通常のシステムでは，$\overline{CS0}$-$\overline{CS2}$空間に配列したメモリは，キャッシュ領域で使用します。ブート・プログラムによりプログラムをダウンロードするようなシステムでは，ダウンロードが完了してからキャッシュ可能領域に設定します。ただし，BHCレジスタの設定を行う命令自身が存在する領域をキャッシュ不可領域→キャッシュ可能領域，またはキャッシュ可能領域→キャッシュ不可領域にすることはできません。キャッシュ可能領域に設定する場合は，キャッシュ不可領域または内蔵命令RAM領域で行ってください。

1.6.3　内蔵データRAM

内蔵データRAM領域は，FFFB000H-FFFEFFFH番地の16 Kバイトの領域に実装されています。命令コードは，配列できません。

1.7　先読み機能（リード・バッファ機能）

V850E/ME2は，CPUの処理を高速に行うために4ワード分（128ビット）の先読み機能用リード・バッファを内蔵しています。ライン・バッファ・コントロール・レジスタ0, 1（LBC0, LBC1）により，$\overline{CSn}$空間ごとに先読みタイミングを設定できます（n = 0-7）。

注意　一般的に連続したアドレスに配列されないユニット（I/Oデバイスなど）や，CPUと非同期に内容が変化するメモリ（他のバス・マスタからライトされるデュアル・ポート・メモリなど）は，先読み機能を禁止します。

1. 8　初期設定端子

V850E/ME2は，さまざまな動作モードを決定する初期設定端子があります。

1. 8. 1　MODE0, MODE1端子

データ・バスの動作モードを指定する入力端子です。MODE0, MODE1端子の指定は応用システムにおいて固定とし，動作中に変更した場合の動作は保証しません。

表1－1　データ・バスの設定

MODE1	MODE0	動作モード	
L	L	通常動作モード	32ビット・データ・バス
L	H		16ビット・データ・バス
上記以外		設定禁止	

備考　L：ロウ・レベル入力
　　H：ハイ・レベル入力

1. 8. 2　PLLSEL, SSEL0, SSEL1端子

X1, X2端子への入力周波数（Fx）に応じて設定する入力端子です。Fx×8 = fx（メイン・クロック）の値に応じて，PLLSEL, SSEL0, SSEL1端子を設定してください。

表1－2　周波数一覧

逓倍数	PLLSEL端子	SSEL1端子	SSEL0端子	入力周波数（MHz）（目標値）	メイン・クロック（fx）周波数（MHz）
8	H	L	H	設定禁止	設定禁止
		H	L	10.00～10.19	80.00～81.59
		H	H	10.20～11.99	81.60～95.99
	L	L	L	12.00～14.39	96.00～115.19
		L	H	14.40～17.39	115.20～139.19
		H	L	17.40～18.75	139.20～150.00
		H	H	設定禁止	設定禁止

注意　f_{CLK}のMAX.値は100 MHz品の場合は100 MHz, 133 MHz品の場合は133 MHz, 150 MHz品の場合は150 MHzです。
　　f_{CLK}（MAX.）＜fxとなる構成で使用した場合の動作は保証しません。
　　fxは各製品の動作保証周波数を越えない値にしてください。

備考　L：ロウ・レベル入力
　　H：ハイ・レベル入力

1. 8. 3　JIT0, JIT1端子

SSCG出力の周波数変調率（f_{DIT}）を指定する入力端子です。JIT0, JIT1端子の設定により，SSCGCレジスタのADJON, ADJ2-ADJ0ビットの初期値（リセット後）は次のようになります。

表1－3　SSCGC.ADJON, ADJ2-ADJ0ビットの初期値

JIT1端子	JIT0端子	初期値			
		ADJONビット	ADJ2ビット	ADJ1ビット	ADJ0ビット
L	L	0	0	0	0
L	H	1	0	0	1
H	L	1	0	1	1
H	H	1	1	0	1

備考　L：ロウ・レベル入力

H：ハイ・レベル入力

第2章　USBバス・ドライバ

2.1 概　　説

2.1.1 概　　要

USBバス・ドライバは，V850E/ME2に内蔵しているUSBファンクション・コントローラ用のサンプル・プログラムです。Universal Serial Bus Specification Revision 1.1に準拠しており，組み込み型制御用リアルタイム・オペレーティング・システムRX850 Pro（μITRON3.0仕様準拠）上で動作します。

このサンプル・プログラムでは，コントロール・エンドポイント（エンドポイント番号0）だけを使用します。クラスとしては，ベンダ固有のクラスを定義し，USB接続時のEnumeration処理（標準デバイス・リクエスト処理のみ）を行います。

このサンプル・プログラムは，ハードウエアの実行環境として評価ボードSolutionGear®MINI（SG-703111-1）を使用しています。SolutionGear MINIとサンプル・プログラムをそのまま使用する場合は，2. 6　**ロード・モジュール**の手順に従って実行オブジェクトを作成し，2. 2　**ロード・モジュールの実行**に従って動作を確認してください。

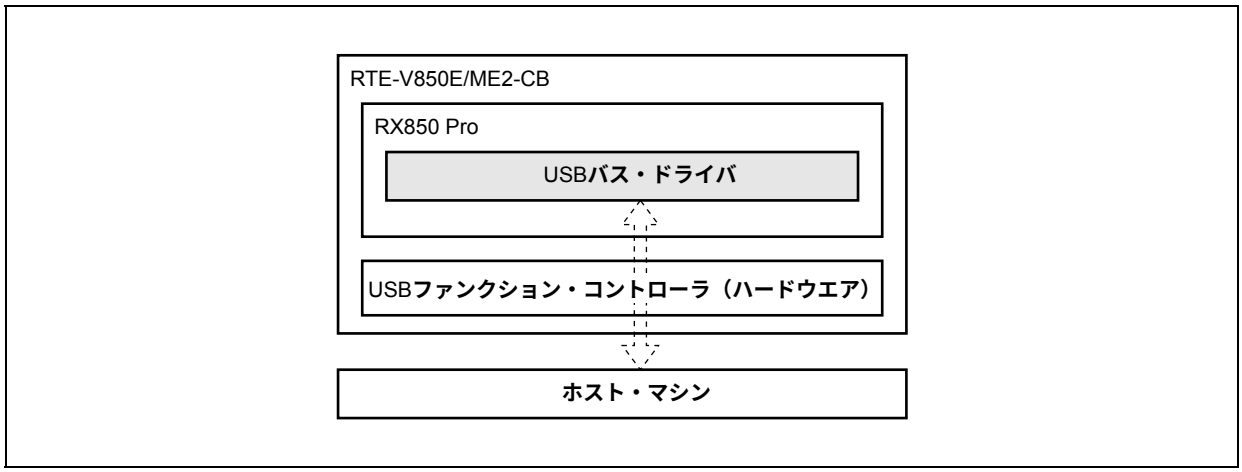
SolutionGear MINIからほかのターゲット・ボードに変更して使用する場合は，2. 3　**システム構築**，2. 4 RX850 Pro**依存処理プログラム**，2. 5　**セクション・マップ・ファイル**を参照し，ボードの仕様に従って変更してください。

SolutionGear MINIおよびサンプル・プログラムの両方を変更して使用する場合は，2. 3　**システム構築**，2. 4 RX850 Pro**依存処理プログラム**，2. 5　**セクション・マップ・ファイル**，2. 6　**ロード・モジュール**，2. 7　USB**ドライバの機能**を参照し，必要な変更を行ってください。

USBバス・ドライバの位置付けを次に示します。

備考　2. 2. 1　**ロード・モジュールの実行手順**は，2. 1. 3　**実行環境**で示した環境を想定して記述しています。

図2－1　USBバス・ドライバの位置付け



2. 1. 2　開発環境

サンプル・プログラムを使用してシステムを開発するうえで必要となるハードウエア環境およびソフトウエア環境として，次に示す環境を想定して記述しています。

・ハードウエア環境

ホスト・マシン　　：PC/AT™互換機（OS：Windows®XP）

・ソフトウエア環境

リアルタイムOS　　：RX850 Pro Version 3.15

USBバス・ドライバ　　：この章で説明するサンプル・プログラム一式

Cコンパイラ・パッケージ：MULTI2000（CCV850 Version 3.5（Green Hills Software™, Inc.製））

注意　サンプルのビルド・ファイルで扱うディレクトリ構成と違う場合は，環境にあわせてビルド・ファイルを編集してください。

備考　ビルド・ファイルの記述内容については，MULTI™（Green Hills Software, Inc.製）のヘルプを参照してください。

2. 1. 3　実行環境

サンプル・プログラムを使用したロード・モジュールを実行するときのハードウエア環境およびソフトウエア環境として，次に示す環境を想定して記述しています。

・ハードウエア環境

ホスト・マシン	：PC/AT互換機（OS：Windows XP）
IE制御用マシン	：PC/AT互換機（OS：Windows XP）
ターゲット・ボード	：SolutionGear MINI（SG-703111-1）
インサーキット・エミュレータ（IE）	：N-wire IE（RTE-2000-TP）：マイダス・ラボ社製
JTAGプローブ	
USBケーブル	

・ソフトウエア環境

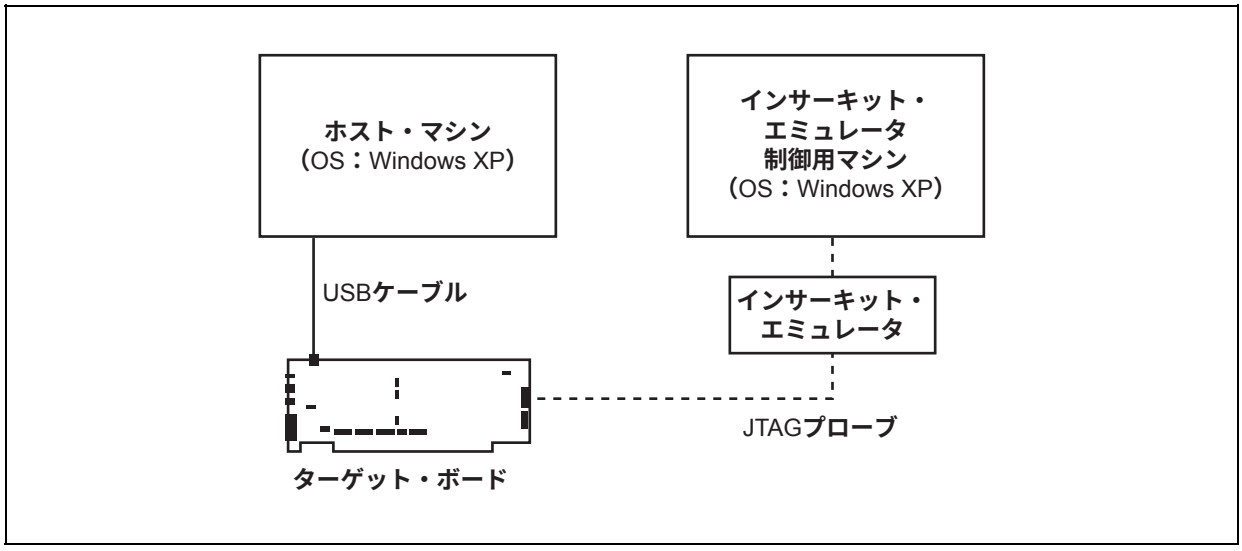
IE用ソフトウエア	：PARTNER Setup Program Version 1.242

備考1. 実行環境のセットアップ方法についての詳細は，付録A　SG-703111-1ボード，およびSG-703111-1　ユーザーズ・マニュアルを参照してください。

2. インサーキット・エミュレータ（RTE-2000-TP）のセットアップ方法についての詳細は，RTE-2000-TP　ハードウエア・ユーザーズ・マニュアルを参照してください。

3. PARTNERについての詳細は，PARTNER　ユーザーズ・マニュアル　V800シリーズ「V800シリーズ共通編」，「NB85E-CB個別編」を参照してください。

図2－2　実行環境



2. 2　ロード・モジュールの実行

2. 2. 1　ロード・モジュールの実行手順

このサンプル・プログラムを使用したロード・モジュールを例にとり，2. 1. 3　**実行環境**で示した環境において実行するときの手順を，次に示します。

① **インサーキット・エミュレータ（IE）制御用マシンの準備**

IE制御用マシンに電源を投入し起動しておきます。また，インサーキット・エミュレータにも電源を投入し準備しておきます。

② **ホスト・マシンの準備**

ホスト・マシンに電源を投入し起動しておきます（ホスト・マシンは，IE制御用マシンで兼用できますが，開発時には別のホスト・マシンを用意することを強く推奨します）。

③ SG-703111-1**ボードのリセット**

SG-703111-1**ボードのリセット・ボタン**RESET**を押下し，**SG-703111-1**ボードのリセットを行います。**

④ **インサーキット・エミュレータの起動**

インサーキット・エミュレータを起動します。

Windows**の「スタート」ボタンから「すべてのプログラム」－「**PARTNER**」－「**RPTSETUP (NB85ET)**」を選択します。**

[PARTNER**の起動例]**

次に示す画面が起動されます。

[PARTNERの起動画面]

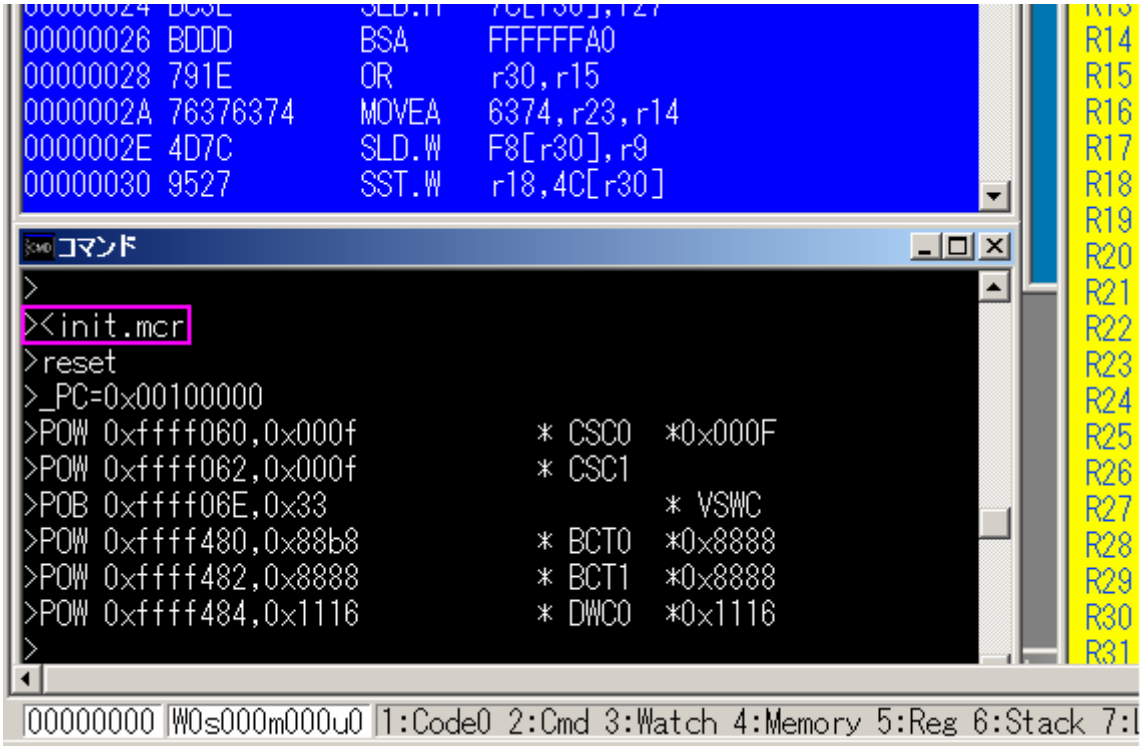
「開く」ボタンをクリックし，プロジェクト・ファイルを指定すると，「起動」ボタンが有効になります。次に，「起動」ボタンをクリックするとPARTNERが起動されます。起動後，ボードの設定を行います。このとき，起動時に読み込まれる設定ファイルをあらかじめ作成しておくと便利です。ここで説明するサンプル用の設定ファイルについては，**付録**A　SG-703111-1**ボード**，および，PARTNER　**ユーザーズ・マニュアル**　V800**シリーズ**「V800**シリーズ共通編**」，「NB85E-CB**個別編**」を参照してください。

注意1.　**インサーキット・エミュレータの起動は，必ずターゲット・ボードの電源投入後に行ってください。**

2.　**インサーキット・エミュレータの起動後，ターゲット・ボードのリセットのために設定ファイルを読み込みたい場合は，コマンド・ウインドウから，次のコマンド入力例のようにしてください。設定ファイル（例では**init.mcr）**を読み込むことができます。**

[コマンド入力例]

```
><init.mcr<Enter>
```



⑤　ロード・モジュールの読み込み

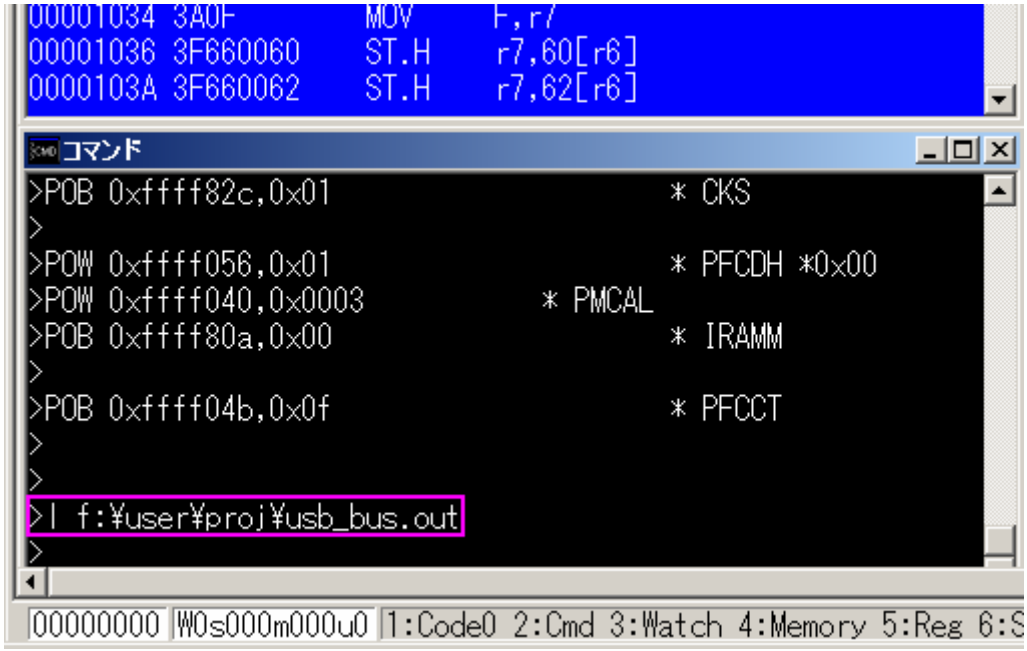
インサーキット・エミュレータの機能を使って，ロード・モジュールをボード上に読み込みます。

ツールバーの「ファイル」－「ロード」からロード・モジュールを読み込むか，コマンド・ウィンドウからL（ファイル読み込み）コマンドを使用してロード・モジュール（例ではusb_bus.out）を読み込みます。

[ファイルのロード例]

[コマンド入力例]

```
>l usb_bus.out<Enter>
```

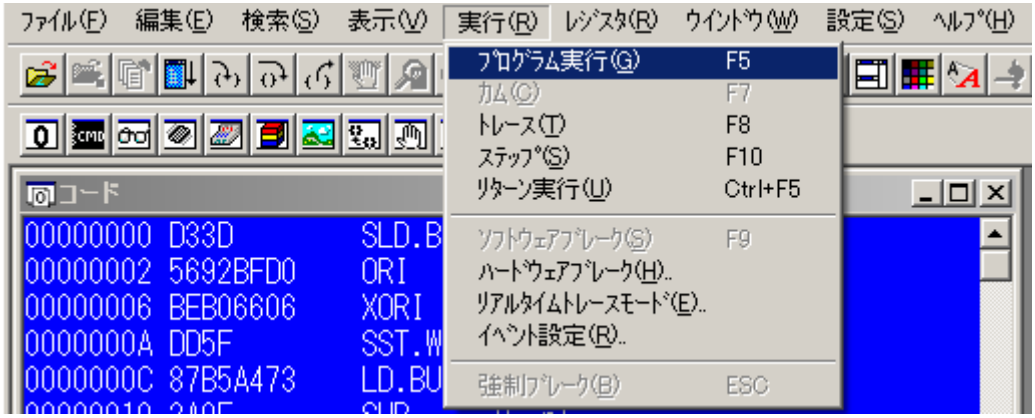


⑥　実行

F5キーまたは実行ボタンで，ボード上に読み込まれたコードを実行します。

注意　ツールバーの「実行」－「プログラムの実行」からも同じように実行できます。

[プログラム実行例]



⑦　USB接続

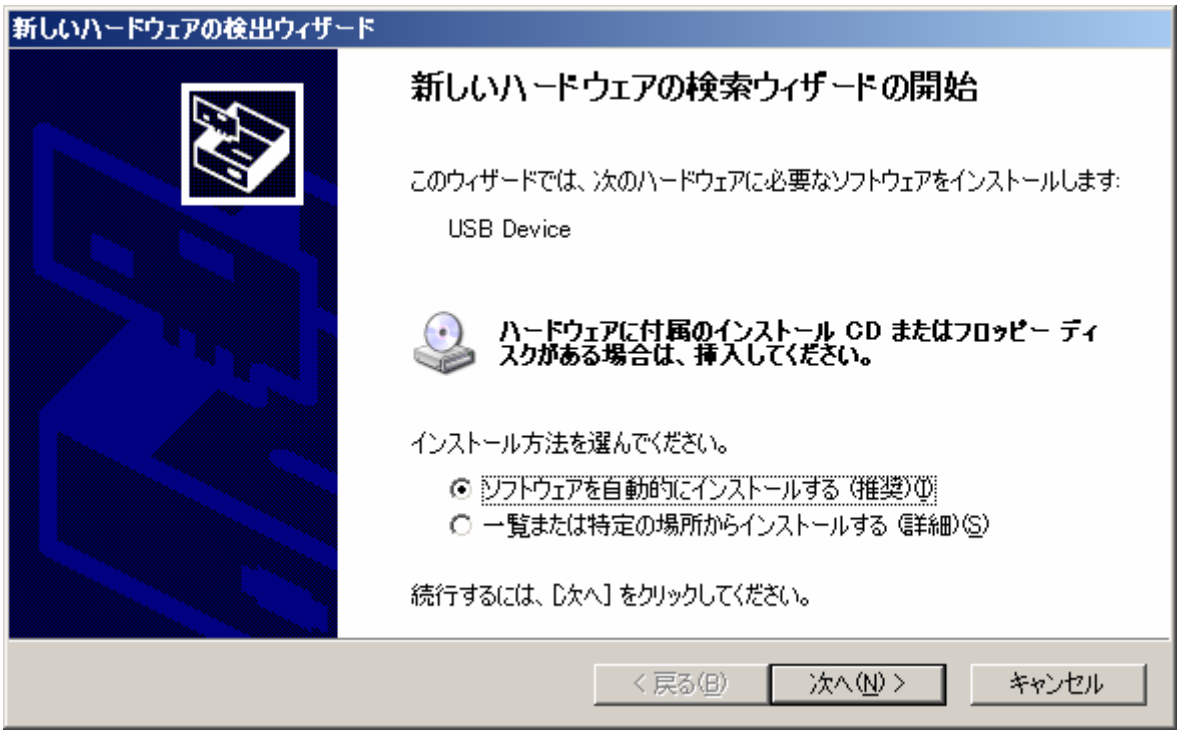
USBケーブルを接続します。

ボード側にBコネクタを接続し，ホスト・マシン側にAコネクタを接続します。

注意1.　**ターゲット・ボードの起動前にUSBを接続しても問題ありません。**

2.　**ホスト・マシン側でデバイスが認識されると，次のようにソフトウエアのインストール画面が現れます。このサンプル・プログラムには専用のホスト・ドライバがないため，ここでは「キャンセル」を選択してください。**

[ソフトウエアのインストール画面]

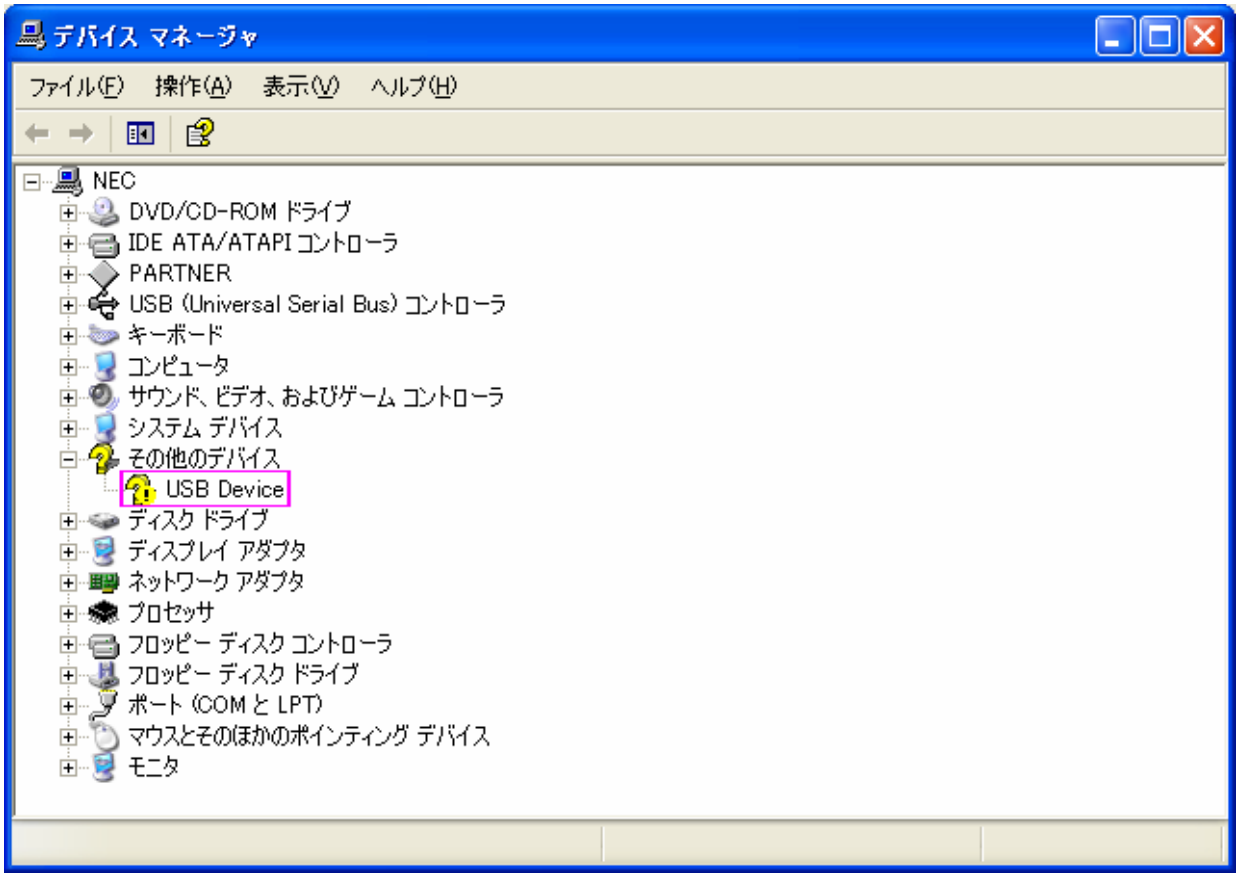


⑧　デバイス マネージャの起動

「マイ コンピュータ」の「プロパティ」から「ハードウェア」のタブを選択します。「デバイス マネージャ」の項目を選択し，「デバイス マネージャ」を起動します。

注意　「マイ コンピュータ」の「管理」や「コントロール パネル」からも起動できます。

[デバイス マネージャ表示例]



⑨　USBデバイスの接続の確認

「デバイス マネージャ」の画面上で，「その他のデバイス」の下に「USB Device」が表示されていることを確認します。

注意　このサンプル・プログラムは，Enumerationの処理までを行うドライバです。したがって，これ以上は動作しません。

⑩　プログラムの終了方法

実行中のプログラムを終了します。

実行中のPARTNERの画面上にある強制終了ボタンをクリックするか，ツールバーの「実行」－「強制ブレーク」を選択し，プログラムの実行を停止します。

[強制終了ボタンによる終了例]

[強制ブレークによる終了例]

⑪　終了方法

インサーキット・エミュレータの終了を行い，ボードを①の手順でリセットします。

ツールバーの「ファイル」－「終了」を選択し，PARTNERを終了してください。

PARTNERを終了してから，ボードを①の手順でリセットします。

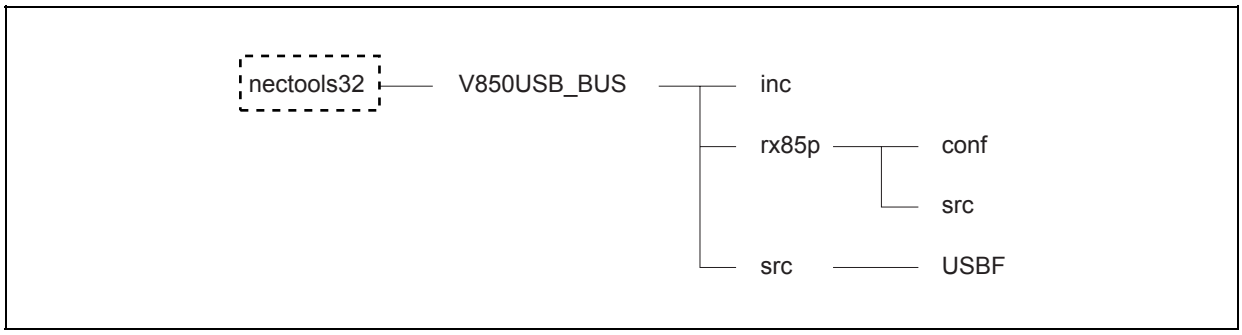
[PARTNERの終了例]

2. 2. 2　ディレクトリ構成

このサンプル・プログラム一式に含まれるファイル群のディレクトリ構成を，次に示します。

注意　USBバス・ドライバ用ファイル郡のディレクトリは，RX850 Proをインストールしたディレクトリ（¥nectools32）の直下に置くことを推奨します。

図2－3　サンプル・プログラムのディレクトリ構成



各ディレクトリの概要を次に示します。

（1）nectools32

RX850 Proをインストールすると作成されるディレクトリです。このディレクトリの直下に，ドライバのディレクトリ（ディレクトリ名：V850USB_BUS）を置きます。

（2）nectools32¥V850USB_BUS

USBバス・ドライバのディレクトリです。

- usb_bus.bld　：USBバス・ドライバのビルド・ファイル
- common.lx　：セクション・マップ・ファイル

（3）nectools32¥V850USB_BUS¥inc

USBバス・ドライバのヘッダ・ファイルが格納されているディレクトリです。

- errno.h　：戻り値用ヘッダ・ファイル
- types.h　：データ・タイプ用ヘッダ・ファイル
- sys.h　：システム情報ヘッダ・ファイル

注意　sys.h（システム情報ヘッダ・ファイル）は，ビルド時に生成されるファイルです。本来，このファイルは，コンフィギュレータを使用してコマンド操作により生成されます。しかし，サンプルのビルド・ファイルを使用すると，ビルド実行時にコマンドを実行する設定になっているため，あらかじめ生成しておく必要はありません。

（4）nectools32¥V850USB_BUS¥rx85p

RX850 Pro用のファイル群が格納されているディレクトリです。

（5）nectools32¥V850USB_BUS¥rx85p¥conf

RX850 Pro用のシステム・ファイルが格納されているディレクトリです。

- sit.850　：システム情報テーブル
- svc.850　：システム・コール・テーブル
- sysi.tbl　：システム情報テーブル
- sysc.tbl　：システム・コール・テーブル

注意1. **このディレクトリにあるファイルは，ビルド時に生成されるファイルです。本来，これらのファイルは，コンフィギュレータを使用してコマンド操作により生成されます。しかし，サンプルのビルド・ファイルを使用すると，ビルド実行時にコマンドを実行する設定になっているため，あらかじめ生成しておく必要はありません。**

2. sit.850とsysi.tbl，svc.850とsysc.tblは，それぞれファイルの拡張子が違うだけで，内容は同じものです。

（6）nectools32¥V850USB_BUS¥rx85p¥src

RX850 Pro用のファイルが格納されているディレクトリです。

- boot.850　：ブート処理用アセンブラ・ファイル
- entry.850　：エントリ処理用アセンブラ・ファイル
- init.c　：ハードウエア初期化部ソース・ファイル
- init.h　：ハードウエア初期化部ヘッダ・ファイル
- sys.cf　：CF定義ファイル
- varfunc.c　：ソフトウエア初期化部ソース・ファイル

（7）nectools32¥V850USB_BUS¥src

USBバス・ドライバのボード依存部のファイルが格納されているディレクトリです。

- port.c　：ポート設定用ソース・ファイル
- port.h　：ポート設定用ヘッダ・ファイル

（8）nectools32¥V850USB_BUS¥src¥USBF

USBバス・ドライバのUSB処理部のファイルが格納されているディレクトリです。

- usbf850.c　：USBバス・ドライバ・ソース・ファイル
- usbf850.h　：USBバス・ドライバ・ヘッダ・ファイル
- usbf850desc.h　：USBバス・ドライバ用ディスクリプタ定義ファイル

2. 3　システム構築

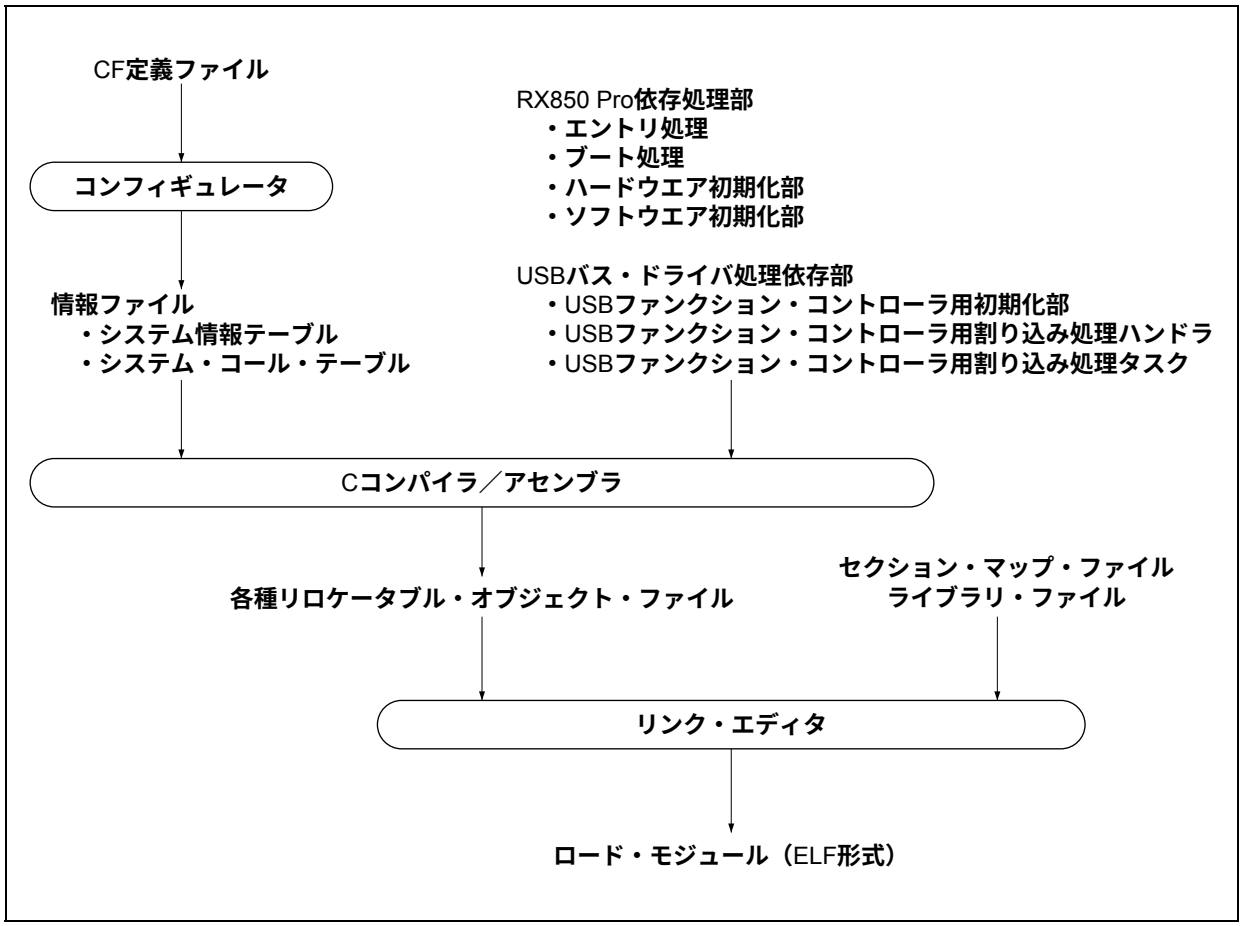
2. 3. 1　概　　要

システム構築とは，USBバス・ドライバ提供媒体からユーザの開発環境（ホスト・マシン）上にインストールしたファイル群を使用して，ロード・モジュールを生成することです。

USBバス・ドライバのシステム構築手順を次に示します。

① RX850 Pro依存処理部の記述
② ボード依存部の記述
③ USBバス・ドライバ処理依存部の記述
④ セクション・マップ・ファイルの記述
⑤ ロード・モジュールの生成

図2－4　システム構築手順



2. 3. 2　RX850 Pro依存処理部の記述

USBバス・ドライバが提供する一部の機能は，リアルタイムOS（RX850 Pro）の機能を利用しています。また，ユーザが記述した処理プログラムは，RX850 Proの管理下で実行することになります。

したがって，RX850 Proを正常に動作させるために，RX850 Pro依存処理部の記述が必要となります。

RX850 Pro依存処理部の一覧を次に示します。

○CF定義ファイル
○エントリ処理
○システム初期化処理
・ブート処理
・ハードウエア初期化部
・ソフトウエア初期化部

備考　RX850 Pro依存処理プログラムについての詳細は，2. 4　RX850 Pro**依存処理プログラム**を参照してください。

2. 3. 3　ボード依存部の記述

USBバス・ドライバのソース・プログラムは，ユーザの実行環境／アプリケーション・システムに依存した処理に関する初期化処理を，ボード依存部として切り出しています。

ボード依存部の一覧を次に示します。

・CPUボード依存部
USBバス・ドライバで必要となるI/Oポート入出力操作については，CPUボード依存部として切り出しています。

注意　ポートの設定は，ほかのレジスタ設定と同じように扱うため，専用の関数は用意していません。レジスタの定義は，RX850 Pro**の標準ヘッダ・ファイル**¥nectools32¥inc850¥common¥SFR.h**を参照してください。処理方法は，ブート処理部（**boot.850**）およびソフトウエア初期化部から呼ばれるポート設定用ソース・プログラム（**port.c**）を参照してください。**

2. 3. 4　USBバス・ドライバ処理依存部の記述

このサンプル・プログラムでは，USBバス・ドライバ機能を実現するためのドライバ関数を，USBバス・ドライバ処理依存部として切り出しています。

USBバス・ドライバ処理依存部の一覧を次に示します。

・USBファンクション・コントローラ初期化部
・USBファンクション・コントローラ割り込みハンドラ
・USBファンクション・コントローラの割り込み処理タスク
・USBファンクション・コントローラ用汎用関数

備考　USBバス・ドライバ処理依存部の詳細は，2. 7　USB**バス・ドライバの機能**を参照してください。

2. 3. 5　セクション・マップ・ファイルの記述

セクション・マップ・ファイルは，リンク・エディタが行うアドレス割り付けをユーザが固定化するためのファイルです。

RX850 Proを使用するとき，次の5つのテキスト領域が必須のセクションとなっています。

- 共通部分配置領域　：.systemセクション
- 割り込み処理関連配置領域　：.system_intセクション
- スケジューラ関連配置領域　：.system_cmnセクション
- システム情報領域　：.sitセクション
- インタフェース・ライブラリ／システム・コール配置領域　：.textセクション

備考　セクション・マップ・ファイルについての詳細は，2. 5　**セクション・マップ・ファイル**を参照してください。

2. 3. 6　ロード・モジュールの生成

コーディングを終了したRX850 Pro依存処理プログラム，USBバス・ドライバ処理依存部，セクション・マップ・ファイルに対して，Cコンパイラ，アセンブラ，リンカなどを実行することにより，ELF形式のロード・モジュールを生成します。

備考　ロード・モジュールの生成手順についての詳細は，2. 6　**ロード・モジュール**を参照してください。

2. 4　RX850 Pro依存処理プログラム

2. 4. 1　概　　要

USBバス・ドライバが提供する一部の機能は，リアルタイムOS（RX850 Pro）の機能を利用しています。また，ユーザが記述した処理プログラムは，RX850 Proの管理下で実行することになります。

したがって，RX850 Proを正常に動作させるために，RX850 Pro依存処理部の記述が必要となります。

RX850 Pro依存処理部の一覧を次に示します。

○CF定義ファイル

○エントリ処理

○システム初期化処理

・ブート処理

・ハードウエア初期化部

・ソフトウエア初期化部

2. 4. 2　CF定義ファイル

RX850 Proを使用したシステムを構築する場合，RX850 Proに提供する各種データを保持した情報ファイル（CF定義ファイル）が必要となります。

USBバス・ドライバ機能を実現するうえで必要となる情報を次に示します。

○リアルタイムOS情報
- RXシリーズ情報

○SIT情報
- システム情報
- システム最大値情報
- システム・メモリ情報
- タスク情報
- 割り込みハンドラ情報
- 初期化ハンドラ情報

○SCT情報
- タスク管理／タスク付属同期管理機能システム・コール情報
- 割り込み処理管理機能システム・コール情報
- 時間管理機能システム・コール情報

注意　このサンプル・プログラムでは，3個のタスク，3個の割り込みハンドラ，7種類のシステム・コールを使用して，各種機能を実現しています。このため，CF定義ファイルにおいては，システム最大値情報のタスクの最大生成数に“3個”を，割り込みハンドラの最大生成数に“3個”をUSBバス・ドライバのために確保するほか，タスク管理／タスク付属同期管理機能システム・コール情報に“sta_tsk, ext_tsk, slp_tsk, wup_tskシステム・コール”を，割り込み処理管理機能システム・コール情報に“loc_cpu, unl_cpuシステム・コール”を，時間管理機能システム・コール情報に“dly_tskシステム・コール”を使用するものとして，定義する必要があります。

備考　CF定義ファイルのコーディング方法についての詳細は，RX850 Pro　**ユーザーズ・マニュアル　インストレーション編**，およびサンプルのCF定義ファイル（sys.cf）を参照してください。

(1) 情報ファイルの生成手順

情報ファイル（システム情報テーブル，システム・コール・テーブル，システム情報ヘッダ・ファイル）の生成手順を，次に示します。

なお，情報ファイルの生成は，Windowsのコマンド・プロンプト上で行います。

注意　サンプルのビルド・ファイルを使用する場合，情報ファイルはビルド時に生成されます。このため，次の手順で生成する必要はありません。

① ディレクトリの移動

Windowsのcdコマンドを使用して，CF定義ファイルが格納されているディレクトリに移動します。なお，CF定義ファイルが格納されているディレクトリがC:¥sampleの場合の入力例を，次に示します。

[コマンド入力例]

```
C:>cd C:¥sample¥rx850<Enter>
```

② 情報ファイルの生成

コンフィギュレータcf850pro.exeを使用して，独自の記述形式で作成されたCF定義ファイルから情報ファイルを生成します。

なお，入力ファイル（CF定義ファイル名：sys.cf）から3つの情報ファイル（システム情報テーブル：sit.850，システム・コール・テーブル：svc.850，システム情報ヘッダ・ファイル：sys.h）を生成する場合の入力例を，次に示します。

[コマンド入力例]

```
C:>cf850pro –i sit.850 –c svc.850 –d sys.h sys.cf<Enter>
```

上記の操作により，CF定義ファイルから情報ファイルを生成できます。

注意　このサンプル・プログラムでは，情報ファイルを生成するためのサンプル・ファイル（CF定義ファイル）を提供しています。

備考　コンフィギュレータcf850pro.exeの起動オプションおよび実行方法についての詳細は，RX850 Pro　ユーザーズ・マニュアル　インストレーション編を参照してください。

2.4.3　エントリ処理

マスカブル割り込みが発生したときに，プロセッサが強制的に制御を移すハンドラ・アドレスに対して，割り込みハンドラへの分岐処理を割り付けています。

なお，RX850 Proの管理下で実行される割り込みハンドラ（CF定義ファイルの割り込みハンドラ情報で定義した割り込みハンドラ）に対応したハンドラ・アドレスに対しては，RX850 Proが提供するマクロRTOS_IntEntry_Indirect（RX850 Proが提供する割り込み処理管理機能への分岐処理）を割り付けてください。

備考　エントリ処理のコーディング方法についての詳細は，添付プログラムentry.850を参照してください。

2.4.4　システム初期化処理

システム初期化処理は，RX850 Proが正常に動作するうえで必要となるハードウエアの初期化処理（ブート処理，ハードウエア初期化部），およびソフトウエアの初期化処理（ニュークリアス初期化部，初期化ハンドラ）から構成されています。

したがって，システムが起動したとき，最初に行われる処理がシステム初期化処理となります。

注意　4種類のシステム初期化処理のうち，ニュークリアス初期化部については，RX850 Proが提供する機能の一部であるため，ユーザがその処理を記述する必要はありません。

ニュークリアス初期化部で行われる処理を次に示します。

○CF定義ファイルで定義されたシステム・メモリの確保
- System Pool 0
- User Pool 0

○CF定義ファイルで定義された管理オブジェクトの生成／初期化
- タスクの生成／起動
- 割り込みハンドラの登録

○初期タスクの起動

○アイドル・タスクの生成／起動

○ソフトウエア初期化部の呼び出し

○スケジューラに制御を移す

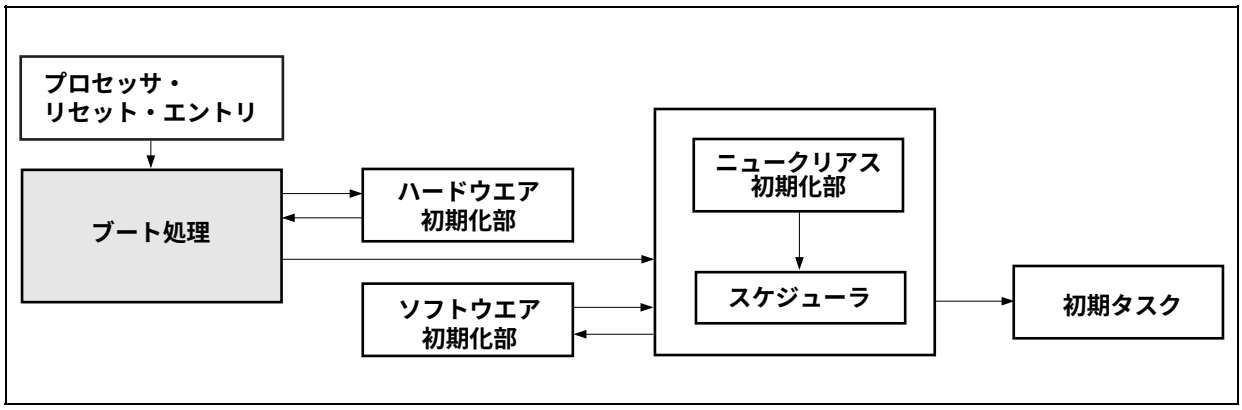
なお，アイドル・タスクとは，RX850 Proの管理下にある処理プログラム（タスク）がrun状態またはready状態でなくなったとき，すなわち，RX850 Proのスケジューリング対象となる処理プログラムがシステム内に1つも存在しなくなったときに，スケジューラから起動される処理ルーチンであり，HALT命令の発行を行っています。

(1) ブート処理

ブート処理は，プロセッサのリセット・エントリに割り付けられた関数であり，システム初期化処理の中でも最初に処理が実行されます。

ブート処理の位置付けを次に示します。

図2－5　ブート処理の位置付け



ブート処理で実行すべき処理内容を次に示します。

備考　ブート処理のコーディング方法についての詳細は，サンプルのboot.850を参照してください。

・tp, gp, epレジスタの設定

システム起動時，各種処理プログラム（ブート処理を含む）を実行するうえで必要となるテキスト・ポインタtp，グローバル・ポインタgp，スタック・ポインタepの値は不定です。そこで，ブート処理の最初の処理として，これらのレジスタの初期設定を行います。

注意　この章では，tpに“0”を，gpに“コンパイラが出力するグローバル・ポインタ・シンボル_gp”を，epに“コンパイラが出力するエレメント・ポインタ・シンボル_ep”を設定することを推奨します。

・ハードウエア初期化部の呼び出し

ターゲット・システム上のハードウエアを初期化するために用意された関数（ハードウエア初期化部）を呼び出します。

なお，ハードウエア初期化部で実行すべき処理内容（内部ユニットの初期化）をほかのところで行う場合は，“ハードウエア初期化部の呼び出し”は不要となります。

注意　この章では，ハードウエア初期化部で実行すべき処理内容（内部ユニットの初期化）をソフトウエア初期化部で行うため，“ハードウエア初期化部の呼び出し”は不要です。詳細は，RX850 Pro　ユーザーズ・マニュアル　インストレーション編を参照してください。

・ニュークリアス初期化部への制御の移行

ニュークリアス初期化部では，システム情報テーブルに記述された情報をもとに，システム・メモリ（System Pool 0, User Pool 0）の確保，および管理オブジェクトの生成／初期化などを行っています。そこで，ニュークリアス初期化部に制御を移す前には，r10レジスタにシステム情報テーブルの先頭アドレス_sitを設定しておく必要があります。

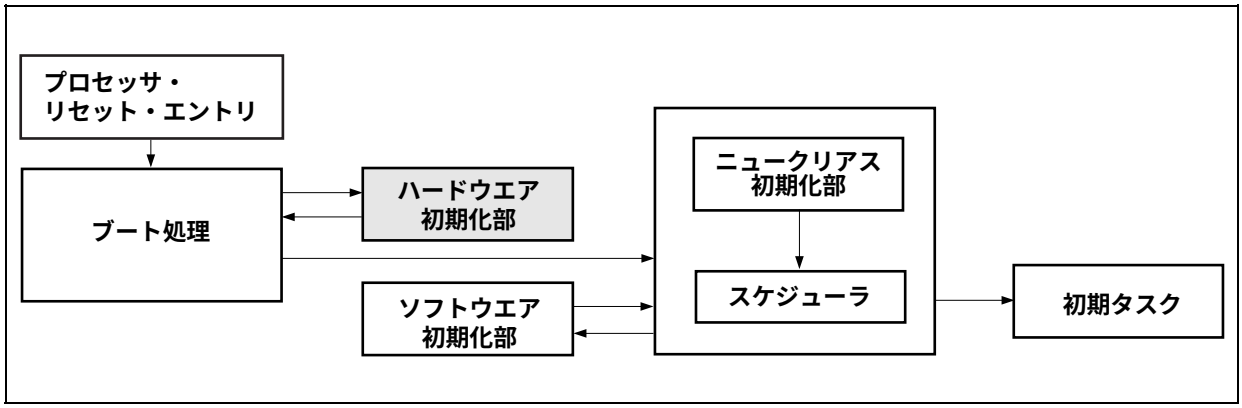
注意　システム情報テーブルとは，独自の記述形式で作成されたCF定義ファイルを，RX850 Proが提供するユーティリティ・ツール（コンフィギュレータcf850pro.exe）を使用して，アセンブリ言語形式に変換したものです。

(2) ハードウエア初期化部

ハードウエア初期化部は，ターゲット・システム上のハードウエアを初期化するために用意された関数であり，ブート処理から呼び出されます。

ハードウエア初期化部の位置付けを次に示します。

図2－6　ハードウエア初期化部の位置付け



ハードウエア初期化部で実行すべき処理内容を次に示します。

注意1. マスカブル割り込みは，初期化時デフォールトでマスクされていますので，特に禁止設定をする必要はありません。

2. サンプル・プログラムでは，ハードウエアの初期化をソフトウエア初期化部で行っています。ハードウエア初期化部の詳細は，RX850 Pro　ユーザーズ・マニュアル　インストレーション編を参照してください。

・ブート処理に制御を戻す

ブート処理の呼び出し関数であるハードウエア初期化部からブート処理に制御を戻す場合，ブート処理におけるハードウエア初期化部の呼び出し時，lpレジスタに対する戻りアドレスの設定が行われているため，“ return( ); ”命令を発行することにより実現されます。

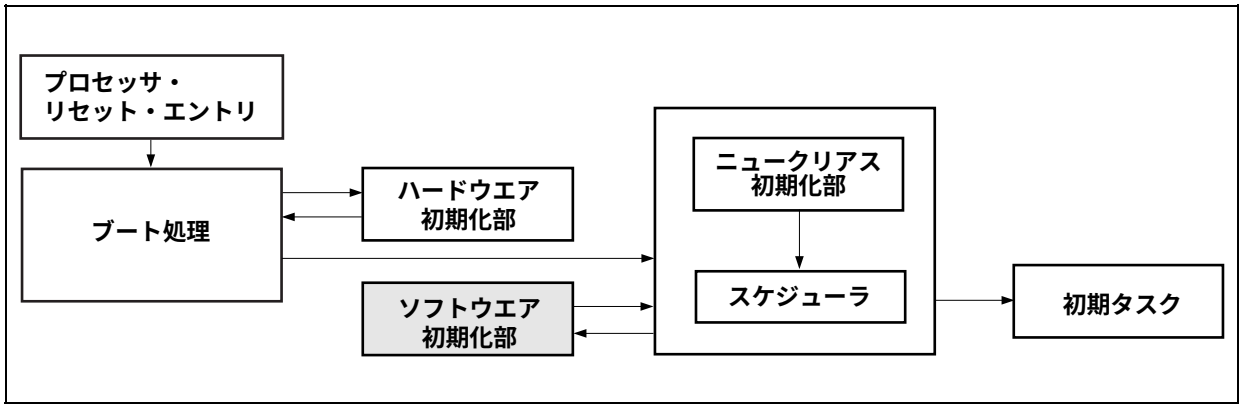
なお，ハードウエア初期化部をアセンブリ言語で記述した場合は，“ jmp [lp] ”命令を発行することにより実現されます。

(3) ソフトウエア初期化部

初期化ハンドラは，ユーザのソフトウエア環境を快適なものとするために用意された関数であり，ニュークリアス初期化部から呼び出されます。

ソフトウエア初期化部の位置付けを次に示します。

図2－7　ソフトウエア初期化部の位置付け



ソフトウエア初期化部で実行すべき処理内容を次に示します。

備考　ソフトウエア初期化部のコーディング方法についての詳細は，サンプルのvarfunc.cを参照してください。

・内部ユニット（リアルタイム・パルス・ユニット（RPU））の初期化

RX850 Proでは，一定周期で発生するタイマ割り込みを利用してタイマ・オペレーション機能（タスクの遅延起床，周期起動ハンドラの起動，タイムアウトなど）を実現しています。

このため，RX850 Proが処理を開始する前にリアルタイム・パルス・ユニットを初期化しておく必要があります。

なお，リアルタイム・パルス・ユニットのコンペア・レジスタCMD0には，CF定義ファイルのシステム情報で定義した基本クロック周期でタイマ割り込みが発生するような値を設定する必要があります。

・タイマ割り込みの受け付け許可

タイマ割り込みの受け付けを許可します。これにより，ニュークリアス初期化部の処理が終了したとき，RX850 Proが提供するタイマ・オペレーション機能（タスクの遅延起床，タイムアウト，周期起動ハンドラの起動など）の利用が可能となります。

・ニュークリアス初期化部に制御を戻す

ニュークリアス初期化部の呼び出し関数である初期化ハンドラからニュークリアス初期化部に制御を戻す場合，ニュークリアス初期化部における初期化ハンドラの呼び出し時に，lpレジスタに対する戻りアドレスの設定が行われているため，“ return( ); ”命令を発行することにより実現されます。

なお，初期化ハンドラをアセンブリ言語で記述した場合は，“ jmp [lp] ”命令を発行することにより実現されます。

2.4.5　時間管理機能

RX850 Proにおける時間管理は，ハードウエア（クロック・コントローラなど）により一定周期で発生するクロック割り込みを利用しています。

クロック割り込みが発生すると，RX850 Proのシステム・クロック処理が呼び出され，システム・クロックの更新やタスクの遅延起床，周期ハンドラの起動などの，時間に関連した処理が行われます。

システム・クロックは，RX850 Proが時間管理のときに使用する時刻（48ビット幅，単位：ms）を保持したソフトウエア・タイマです。

システム・クロックは，システム初期化処理で“0H”に設定されたあと，システム・クロック処理で基本クロック周期（コンフィギュレーション時に指定）を単位として更新されます。

注意　RX850 Proが管理するシステム・クロックは，48ビット幅で構成されています。このため，RX850 Proでは，オーバフローした数値（48ビットでは表せない数値）は無視されます。RX850 Proの時間管理機能の詳細は，RX850 Pro　ユーザーズ・マニュアル　基礎編を参照してください。

2. 5　セクション・マップ・ファイル

2. 5. 1　概　　要

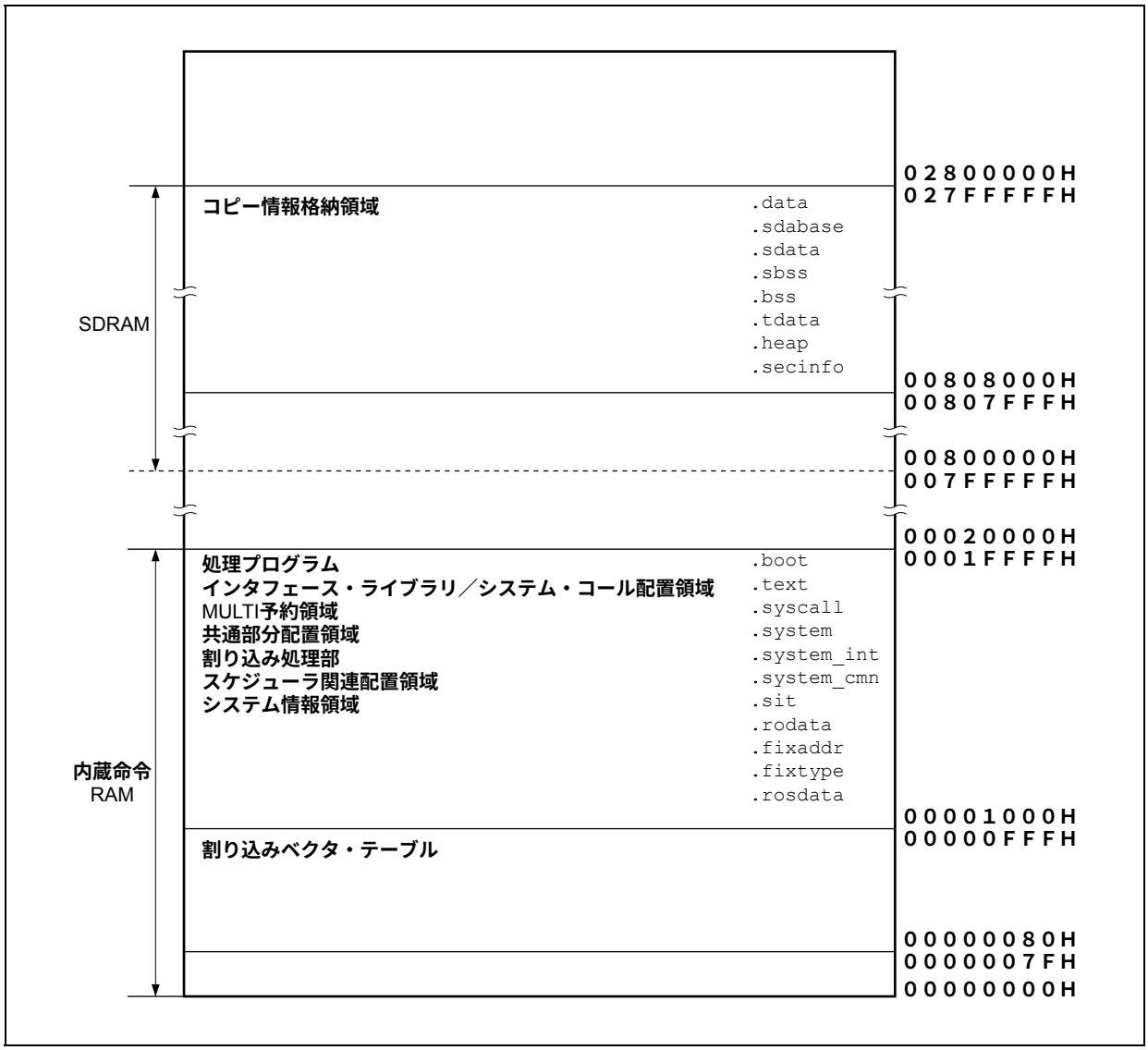
セクション・マップ・ファイルとは，リンク・エディタが行うアドレス割り付けを，ユーザが固定化するためのファイルです。

2. 5. 2　RX850 Proのアドレス割り付け，2. 5. 3　その他のアドレス割り付けに，ユーザの処理プログラム（.data, .bssセクションなど）以外に必要となるアドレス割付けについて示します。

サンプルのcommon.lxで行われているアドレス割り付けを次に示します。

備考　セクション・マップ・ファイルのコーディング方法についての詳細は，サンプルのcommon.lxを参照してください。

図2－8　アドレス割り付け例



2. 5. 2　RX850 Proのアドレス割り付け

RX850 Proは，5つのテキスト領域（共通部分配置領域，割り込み処理関連配置領域，スケジューラ関連配置領域，システム情報領域，インタフェース・ライブラリ／システム・コール配置領域）から構成されています。これにより，大きなサイズが要求されるメモリ領域については外部RAMに，高速なアクセスが要求されるメモリ領域（割り込み処理部，スケジューリング処理部）については内蔵命令RAM（00000000H-0001FFFFH）に割り付けるといったことが可能となります。

注意　サンプル・プログラムでは，5つのテキスト領域のすべてを内蔵命令RAMに配置しています。

・共通部分配置領域（.systemセクション）

RX850 Proの本体処理（タスク管理機能，タスク付属同期機能など）が割り付けられる領域です。

・割り込み処理関連配置領域（.system_intセクション）

RX850 Proが提供する割り込み処理管理機能のうち，割り込みハンドラに制御を移すときに行われる割り込み前処理，およびマスカブル割り込みが発生した処理プログラムに制御を戻すときに行われる割り込み後処理から構成されています。

したがって，割り込み処理部を内蔵命令RAMに割り付けることにより，割り込みハンドラに対する応答性能が向上します。

注意　この章では，割り込み処理部を内蔵命令RAMに割り付けることを推奨します。

・スケジューラ関連配置領域（.system_cmnセクション）

RX850 Proが提供するスケジューリング機能のうち，タスクの起床処理およびタスクのスケジューリング処理から構成されています。

したがって，スケジューリング処理部を内蔵命令RAMに割り付けることにより，タスクの起床処理およびタスクのスケジューリング処理が高速化されるほか，スケジューリング処理を伴ったシステム・コールの処理も高速化されます。

注意　この章では，スケジューリング処理部を内蔵命令RAMに割り付けることを推奨します。

・システム情報領域（.sitセクション）

CF定義ファイルに対してコンフィギュレータcf850.exeを実行することにより生成されたシステム情報テーブルが割り付けられる領域です。

なお，システム情報テーブルは，ニュークリアス初期化部（システム・メモリの確保，管理オブジェクトの生成／初期化）を実行するうえで必要となる各種データから構成されています。

・インタフェース・ライブラリ／システム・コール配置領域（.textセクション）

システム・コールを含む命令が割り付けられる領域です。

・システム・メモリ

RX850 Proが提供する機能を実現するうえで必要となる各種管理ブロック（タスク管理ブロック，セマフォ管理ブロックなど），割り込みハンドラ用スタック，タスク用スタックが割り付けられる領域（System Pool 0），および処理プログラムからダイナミックなメモリ操作（メモリ・ブロックの獲得／開放）を可能とした領域（User Pool 0）から構成されています。

注意1. CF定義ファイル作成時，“システム・メモリの先頭アドレス”を指定する必要があります。したがって，セクション・マップ・ファイルにシステム・メモリの定義を行うときには，必ずアドレスを指定してください。

2. システム・メモリのセクション名については，ユーザが自由に指定できます。

2.5.3　その他のアドレス割り付け

次に，RX850 Pro以外にアドレス割り付けが必要となるセクションについて示します。

・MULTI予約領域（.syscallセクション）

ディバッガMULTI（Green Hills Software, Inc.製）がワーク・エリアとして使用する領域です。

注意1. MULTI使用の有無にかかわらず，.syscallセクションの定義を行う必要があります。

2. .syscallセクションの定義を行うときには，必ず4バイト・アライン指定を行ってください。

・コピー情報格納領域（.secinfoセクション）

セクション・マップ・ファイルで，ROM識別子の指定が行われているセクションのプログラム（データ，テキスト）をROMからRAMに転送するときに必要となる情報（先頭アドレス，サイズ）を，リンク・エディタが出力するための領域です。

なお，ROM識別子の指定は，処理プログラムをROM化するときに必要となるものです。したがって，ROM化を行わない場合には，.secinfoセクションの定義は不要となります。

注意　このサンプル・プログラムでは，ROM識別子の指定を行っていないため，空のセクションとなります。

2. 6　ロード・モジュール

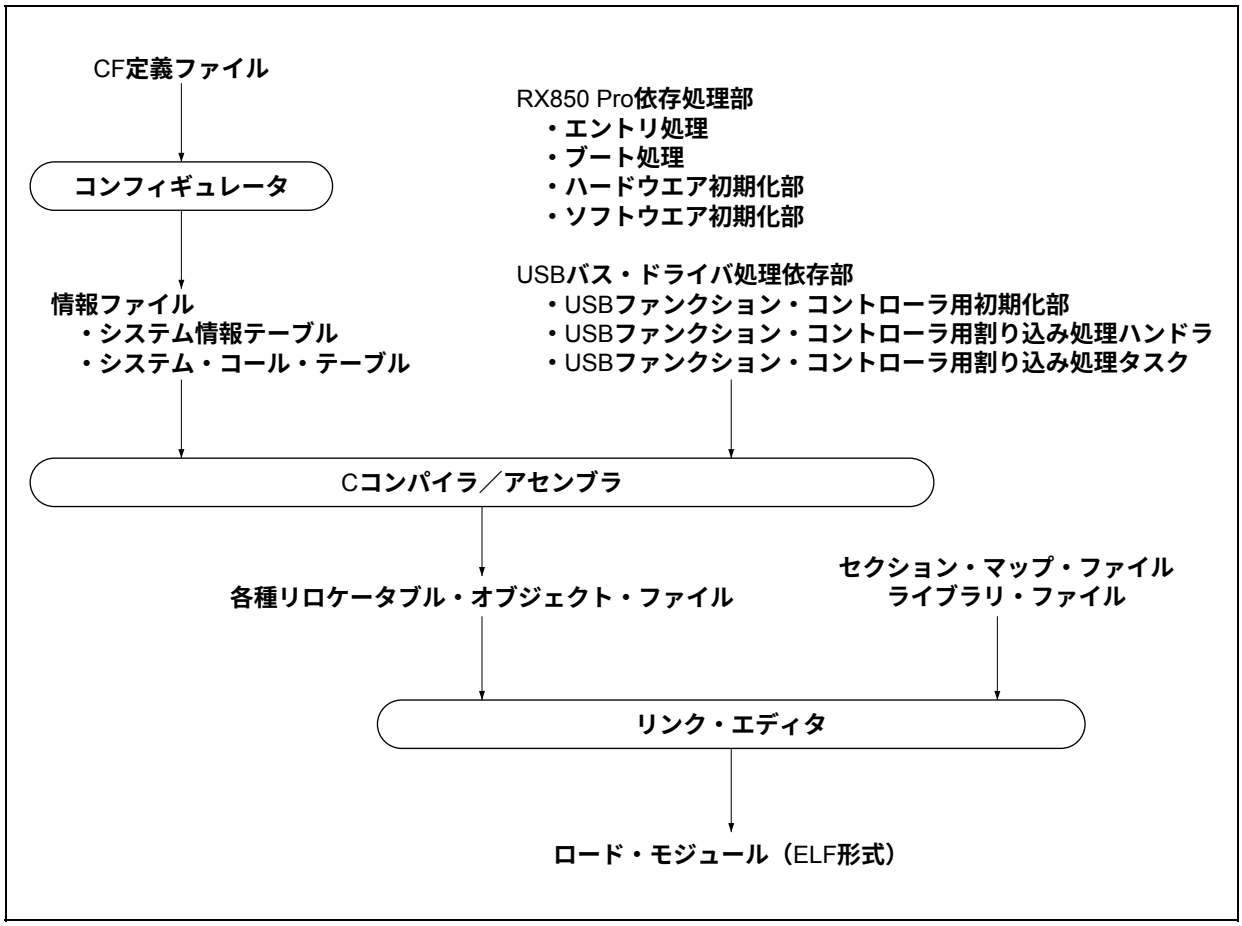
2. 6. 1　概　　要

コーディングを終了したRX850 Pro依存処理プログラム，USBバス・ドライバ処理依存部，セクション・マップ・ファイルに対して，Cコンパイラ，アセンブラ，リンカなどを実行することにより，ELF形式のロード・モジュールを生成します。

ロード・モジュールの生成手順を次に示します。

注意　サンプルの.bldファイルを実行することにより，サンプル・プログラムに対応したロード・モジュールを生成できます。
ただし，.bldファイルのパスの定義については，ユーザの開発環境にあわせた修正が必要です。

図2－9　ロード・モジュールの生成手順



2. 6. 2　ロード・モジュールの生成

コーディングを終了したRX850 Pro依存処理プログラム，USBバス・ドライバ処理依存部，セクション・マップ・ファイルは，次の手順によりELF形式のロード・モジュールとなります。

① システム情報テーブル，システム・コール・テーブルの生成

独自の記述形式で作成されたCF定義ファイルは，ロード・モジュール作成時，リンク・エディタが行うリンク処理の対象外のファイル形式です。

そこで，RX850 Proが提供するユーティリティ・ツール（コンフィギュレータcf850.exe）を使用して，アセンブル可能なファイル（システム情報テーブル，システム・コール・テーブル）を生成します。

備考　システム情報テーブル，システム・コール・テーブルの生成方法についての詳細は，2. 4. 2（1）**情報ファイルの生成手順**を参照してください。

② オブジェクト・ファイルの生成

次に示す各種処理プログラム（C言語／アセンブリ言語形式のファイル）に対して，Cコンパイラ／アセンブラを実行することにより，リロケータブル・オブジェクト・ファイルを生成します。

○RX850 Pro依存処理プログラム
- システム情報テーブル
- システム・コール・テーブル
- エントリ処理
- ブート処理
- ハードウエア初期化部
- 初期化ハンドラ

○USBバス・ドライバ処理依存部

③ ロード・モジュールの生成

②で生成したリロケータブル・オブジェクト・ファイル，および，各種ライブラリ・ファイル，セクション・マップ・ファイルに対して，リンク・エディタを実行することにより，ELF形式のロード・モジュールを生成します。

libansi.a	ANSI Cライブラリ
libind.a	Green Hills Software, Inc.製Cライブラリ（ターゲットCPUに依存しないルーチン群）
libarch.a	Green Hills Software, Inc.製Cライブラリ（ターゲットCPUに依存するルーチン群）
libsys.a	Green Hills Software, Inc.製Cライブラリ（システム・コール，初期化ルーチンなど）

rxcore.o	ニュークリアス共通部分オブジェクト
librxp.a	ニュークリアス・ライブラリ
libchp.a	インタフェース・ライブラリ

なお，rxcore.o, librxp.a, libchp.aは“ RX850 Pro ”から，libansi.a, libind.a, libarch.a, libsys.aは“ CCV850（Green Hills Software, Inc.製） ” から提供されています。

2. 7　USBバス・ドライバの機能

2. 7. 1　概　　要

USBバス・ドライバでは，USBバス・ドライバ処理を実現するためのタスク，割り込みハンドラのほかに，USBファンクション・コントローラの初期化処理の記述が必要となります。

USBバス・ドライバ処理依存部の一覧を次に示します。

・USBファンクション・コントローラの初期化処理

RX850 Proのソフトウエア初期化部から呼び出され，USBファンクション・コントローラの初期化処理を行います。

・USBファンクション・コントローラの割り込みハンドラ

USBファンクション・コントローラの割り込み発生ごとに呼び出される割り込み処理専用ルーチンであり，CF定義ファイルに定義されています。

注意　このサンプル・プログラムでは，必要な割り込み以外はマスクされています。
このサンプル・プログラムで使用する割り込みは，INTUSB0B信号で通知されるCPUDEC（UF0E0STレジスタにFWでデコードを行うリクエストがあることを示す）だけです。

・USBファンクション・コントローラの割り込み処理タスク

USBファンクション・コントローラの割り込みハンドラから呼び出され，割り込み要因ごとの処理（レジスタ設定，データの送信／受信処理など）を行います。

・USBファンクション・コントローラ用汎用関数

USBバス・ドライバで使用される汎用関数として，エンドポイントごとのSTALL応答の設定や，送信，受信の処理を行う関数が用意されています。

備考　USBバス・ドライバ処理依存部のコーディング方法についての詳細は，サンプルのusbf850.cを参照してください。

・USBのサスペンド／レジュームの処理

USBのサスペンド／レジュームの処理はシステムに依存するため，このサンプル・プログラムではサポートしていません。システム上，処理が必要な場合は，次のことに注意して処理を追加してください。

V850E/ME2に内蔵しているUSBファンクション・コントローラでは，サスペンド／レジュームが割り込み（INTUSB0B信号）により通知されます。このため，割り込みハンドラ（INTUSB0B信号用）内でUF0IS0.RSUSPDビットを確認し，セット（1）されていれば，UF0EPS1.RSUMビットを確認してサスペンド状態なのかレジューム状態なのかを判断できます。

処理を追加する一例として，割り込みハンドラ（INTUSB0B信号用）に上記の状態判断のコードを追加し，そこから必要な処理を行うタスクを起床するようにすることで実現できます。

2. 7. 2　処理の流れ

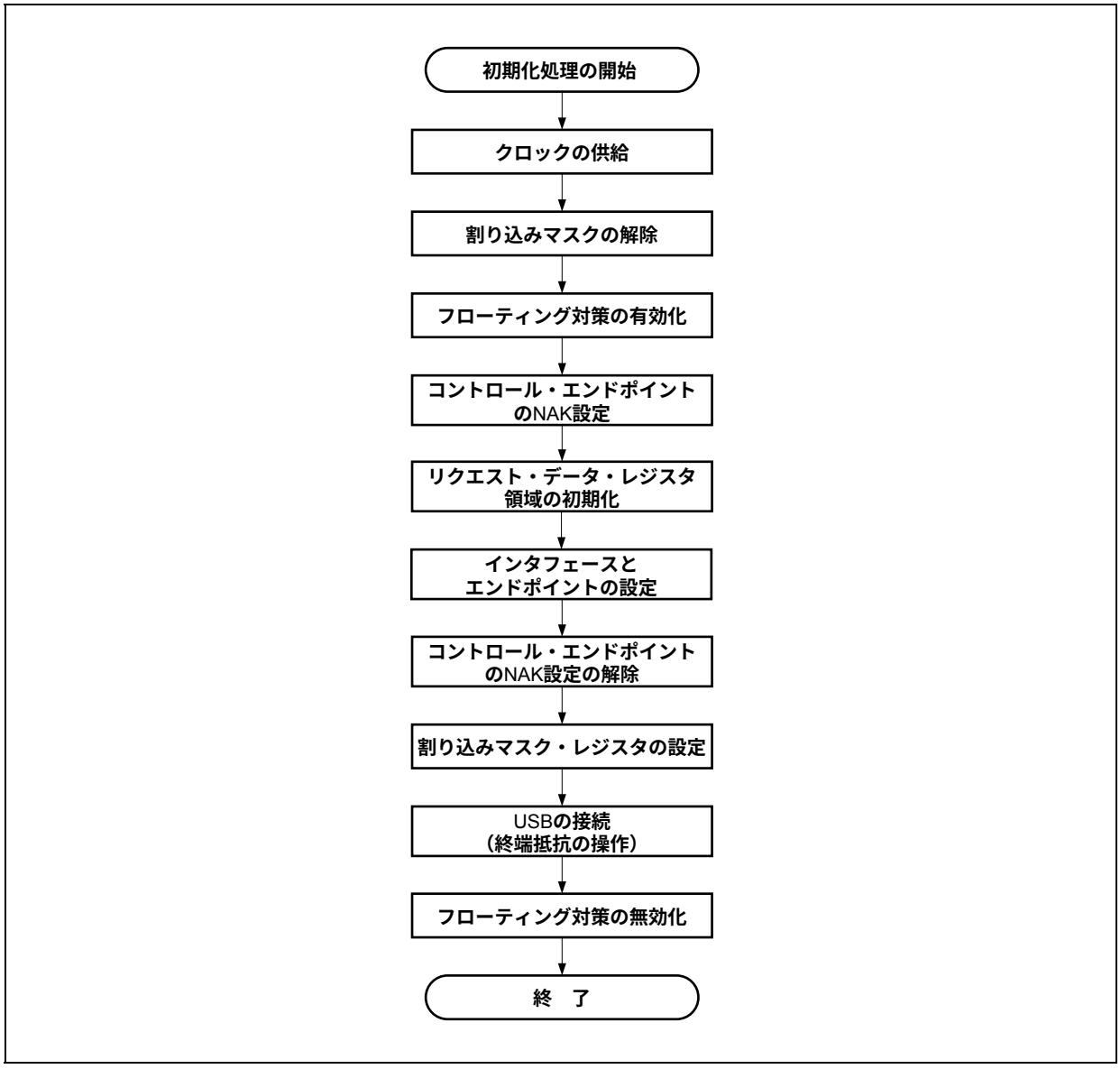
サンプル・プログラムの初期化処理，割り込み処理についての処理の流れを，次に示します。

（1）初期化処理

USBデバイスの初期化は，ソフトウエア初期化部から呼び出され実行されます。

サンプル・プログラムにおけるUSBデバイス初期化処理（電源投入時）の流れを，次に示します。

図2－10　初期化処理のフロー・チャート



初期化処理で実行すべき処理内容を次に示します。

注意　ポートの処理以外では，初期化処理が必要です。ターゲット・ボードが違う場合，端子の割り付けが違うことがありますので，ターゲット・ボードの仕様にあわせて読み替えてください。

・クロックの供給

USBファンクション・コントローラのレジスタを設定する前に，必ずUCKC.UCKCNTビットをセット（1）する必要があります。セット（1）することにより，USBへのクロック供給を許可します。なお，P10端子をクロック入力として使用するので，P10端子を入出力ポート・モードの入力モードに設定し，クロックの入力を許可します。

・割り込みマスクの解除

割り込み制御レジスタでUSB関連の割り込み信号のマスクを解除します。

・フローティング対策の有効化

UF0BC.UBFIORビットをクリア（0）し，ケーブル未接続時の不定値によるBus Resetなどの誤認識を防止します。

・コントロール・エンドポイントのNAK設定

自動実行リクエストを含むすべてのリクエストにNAK応答します。
自動実行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実行リクエストに意図しないデータを返さないように設定します。

・リクエスト・データ・レジスタ領域の初期化

Get Descriptorリクエストに応答するためのディスクリプタ・データなどをレジスタに登録します。登録するデータは，デバイス・ステータス，エンドポイント0ステータス，Device Descriptor, Configuration Descriptor, Interface Descriptor, Endpoint Descriptorです。

注意　クラスによっては，そのクラス用のディスクリプタの登録が必要な場合があります。このサンプル・プログラムでは，ベンダ固有のクラスを定義し，USBの標準ディスクリプタのみを使用します。

・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，Alternative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，レジスタに設定します。

・コントロール・エンドポイントのNAK設定の解除

自動実行リクエスト用のデータ登録が終わったところで，コントロール・エンドポイント（エンドポイント番号0）のNAK設定を解除します。

・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ステータス・レジスタに示される割り込み要因ごとのマスクを設定します。

・USBの接続（終端抵抗の操作）

D+信号をプルアップします。

・フローティング対策の無効化

UF0BC.UBFIORビットをセット（1）し，フローティング対策を無効化します。

（2）割り込み処理

サンプル・プログラムでは，初期化完了後，割り込みイベントによって動作します。イベントがない場合は，常にアイドル状態です。

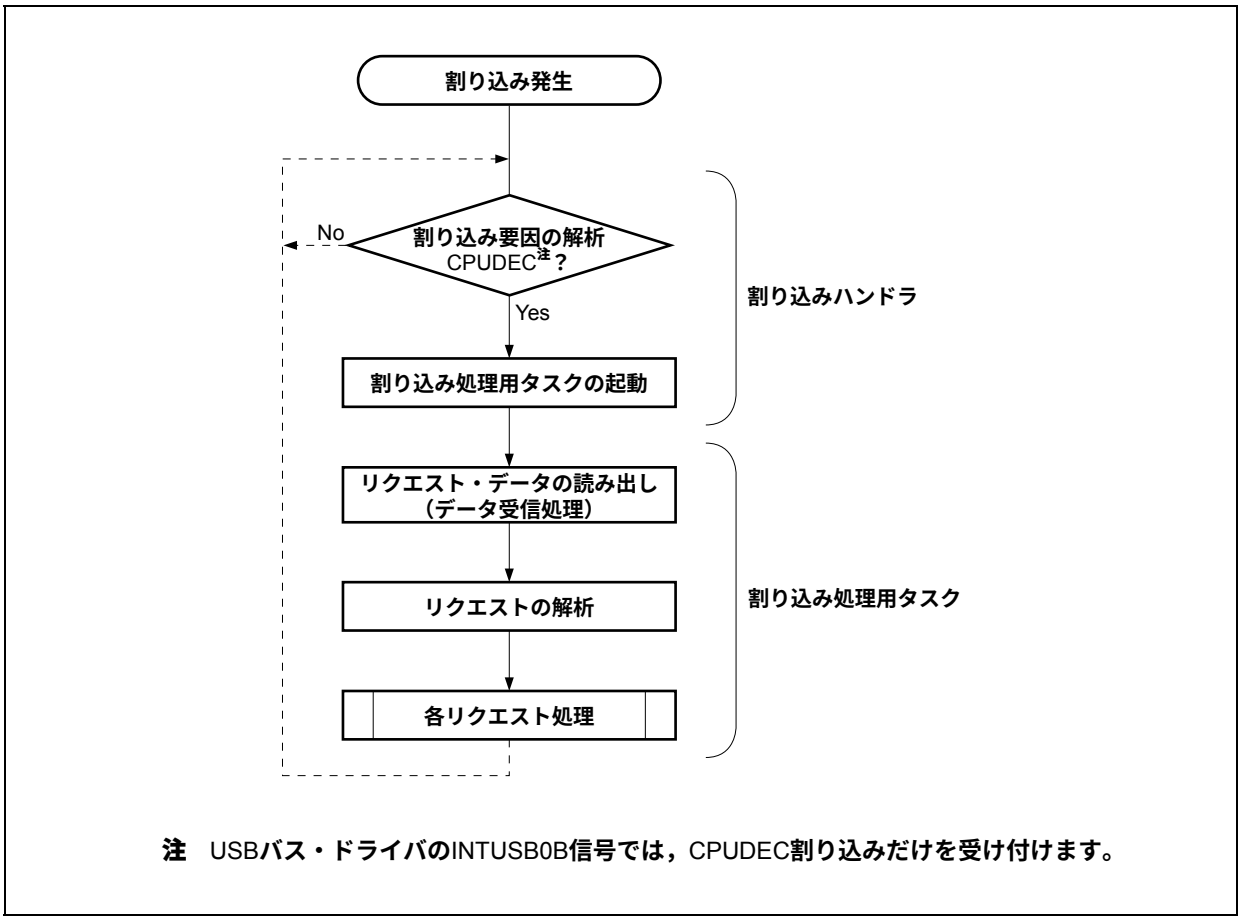
サンプル・プログラムでの割り込み処理の流れを次に示します。

注意1. **このサンプル・プログラムでは対応するホスト・ドライバがないため，**USB**の**Enumeration**処理（デバイスの問い合わせ）までしか動作しません。このため，**V850E/ME2**に内蔵している**USB**ファンクション・コントローラによる自動応答リクエストしかホスト側から発行されないので，常にアイドル状態となっています。**

2. **図**2－11**に示すフロー・チャートは，**USB**ファンクション・コントローラが自動応答しない**Get Descriptor（String Descriptor）**リクエスト，および，デバイス・クラス固有のリクエストを処理する場合の動作となります。**

バルク・エンドポイントでの送受信処理は，第3**章　**USB**ストレージ・クラス用ドライバ，および第**4**章　**USB**コミュニケーション・クラス用ドライバを参照してください。**

図2－11　**割り込み処理のフロー・チャート**



注　USB**バス・ドライバの**INTUSB0B**信号では，**CPUDEC**割り込みだけを受け付けます。**

割り込み処理で実行すべき処理内容を次に示します。

・割り込み要因の解析
実行される割り込みハンドラにより，解析する割り込みステータスが違います。
このサンプル・プログラムでは，CPUDEC割り込みだけに対応しているので，INTUSB0B信号による割り込みハンドラが起動します。この割り込みハンドラの中で，UF0IS1レジスタを読み，割り込み要因がCPUDEC割り込みかどうかを確認します。

注意　このサンプル・プログラムでは，使用する割り込みハンドラをCF定義ファイルであらかじめ登録してあります。

・割り込み処理用のタスクの起動
割り込み要因がCPUDECだった場合，task_usb0bタスクを起動します。

注意　このサンプル・プログラムでは，起動するタスクをCF定義ファイルであらかじめ登録してあります。

・リクエスト・データの読み出し
UF0E0STレジスタからSETUPデータを読み出します。

・リクエストの解析
読み出したSETUPデータを解析し，リクエスト内容を確認します。

・各リクエスト処理
解析したリクエスト内容に対応した処理を行います。
サンプル・プログラムでは，標準デバイス・リクエストのGet Descriptor（String Descriptor）だけを処理します。

2. 7. 3　USBバス・ドライバのディスクリプタ情報

サンプル・プログラムで定義されているUSB標準ディスクリプタについて，次に示します。

(a) - (d) までのディスクリプタは，最低限用意するべきディスクリプタです。

備考　詳細は，Universal Serial Bus Specification Revision 1.1を参照してください。

(a) Device Descriptor

デバイスの一般的な情報を持ち，デバイスごとに1つのDevice Descriptorを用意する必要があります。この情報は，デバイスのコンフィギュレーションで唯一のデバイスを認識するために使用されます。

サンプル・プログラムでは，USB接続時のEnumeration処理（標準デバイス・リクエスト処理）だけを行います。クラスとしては，ベンダ固有のクラスが定義されています。

表2－1　Device Descriptor

オフセット	サイズ（バイト）	値	説　明
0	1	12H	このディスクリプタの Length 値（バイト）
1	1	01H	ディスクリプタ・タイプ（デバイス）
2	2	10H/01H	USB のバージョン（USB1.1）
4	1	FFH	クラス・コード（ベンダ固有のクラス）
5	1	00H	サブクラス・コード
6	1	00H	プロトコル・コード
7	1	40H	Endpoint0 の最大パケット・サイズ
8	2	09H/04H	ベンダ ID（NEC エレクトロニクス）
10	2	FBH/FFH	プロダクト ID
12	2	01H/00H	デバイス・リリース番号
14	1	01H	ストリング・ディスクリプタへのインデクス（Manufacturer）
15	1	00H	ストリング・ディスクリプタへのインデクス（Product）
16	1	00H	ストリング・ディスクリプタへのインデクス（Serial Number）
17	1	01H	可能なコンフィギュレーションの数

（b）Configuration Descriptor

具体的なデバイス・コンフィギュレーションについての情報を保持しています。

表2－2　Configuration Descriptor

オフセット	サイズ（バイト）	値	説　明
0	1	09H	このディスクリプタのLength値（バイト）
1	1	02H	ディスクリプタ・タイプ（コンフィギュレーション）
2	2	12H/00H	Get Descriptor要求でコンフィギュレーション・ディスクリプタと一緒に返されるディスクリプタのトータルLength値
4	1	01H	この構成でサポートされているインタフェース数
5	1	01H	コンフィギュレーション値
6	1	00H	ストリング・ディスクリプタへのインデクス（Configuration）
7	1	C0H	デバイスの構成（Self-powered/Remote Wakeup機能）
8	1	00H	デバイスの最大消費電力

（c）Interface Descriptor

コンフィギュレーション内の具体的なインタフェース情報を保持しています。

サンプル・プログラムでは，コンフィギュレーションは1つのインタフェースを提供します。

Interface Descriptorは，常にConfiguration Descriptorの一部として戻され，Interface Descriptorにおいて，Get DescriptorおよびSet Descriptor要求による直接アクセスはされません。

表2－3　Interface Descriptor（1）

オフセット	サイズ（バイト）	値	説　明
0	1	09H	このディスクリプタのLength値（バイト）
1	1	04H	ディスクリプタ・タイプ（インタフェース）
2	1	00H	インタフェース値
3	1	00H	Alternate設定値
4	1	00H	Endpoint数（Endpoint0を除く）
5	1	FFH	インタフェース・クラス（ベンダ固有のクラス）
6	1	00H	インタフェース・サブクラス
7	1	00H	インタフェース・プロトコル
8	1	00H	ストリング・ディスクリプタへのインデクス（インタフェース）

（d）Endpoint Descriptor

ホストが個々のEndpointのバンド幅要件を決定するために必要な情報を保持しています。

Endpoint Descriptorは，常にGet Descriptor（コンフィギュレーション）要求によってコンフィギュレーション情報の一部として戻されます。

Endpoint Descriptorにおいて，Get DescriptorおよびSet Descriptor要求による直接アクセスはされません。

注意　このサンプル・プログラムは，Enumerationの処理までを行うドライバのため，コントロール・エンドポイント以外は使用しません。このため，Endpoint Descriptorは定義していません。

（e）String Descriptor

ホストが個々のEndpointのバンド幅要件を決定するために必要な情報を保持しています。

表2－4　String Descriptor（1）

オフセット	サイズ（バイト）	値	説　明
0	1	04H	このディスクリプタのLength値（バイト）
1	1	03H	ディスクリプタ・タイプ（ストリング）
2	2	09H/04H	ストリング・ディスクリプタで使用する言語タイプ（English/U.S.）

表2－5　String Descriptor（2）

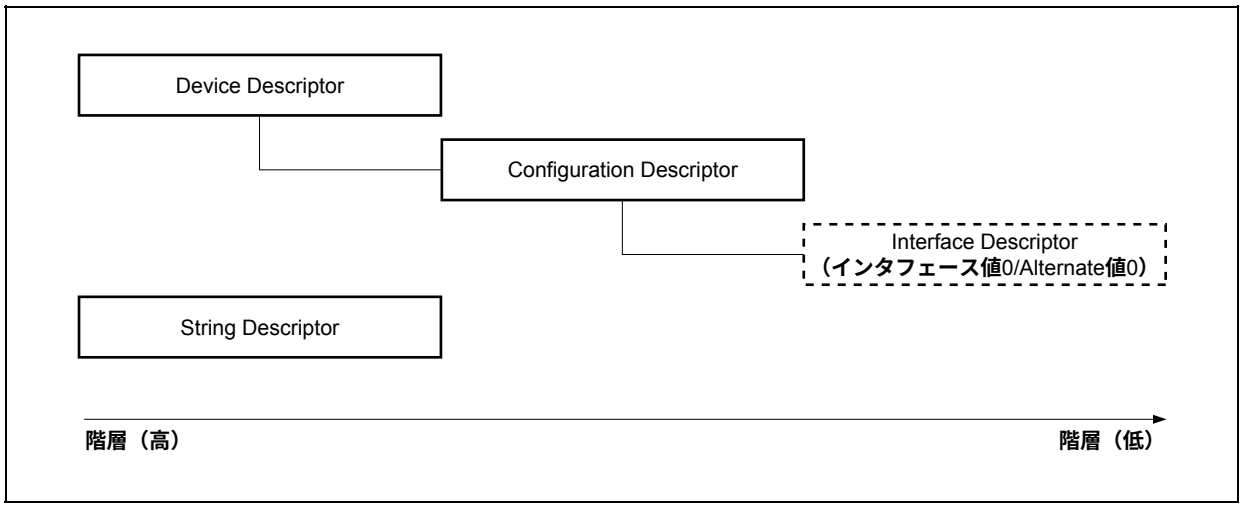
オフセット	サイズ（バイト）	値	説　明
0	1	2AH	このディスクリプタのLength値（バイト）
1	1	03H	ディスクリプタ・タイプ（ストリング）
2	40	'N','E','C','','E','l','e','c','t','r','o','n','i','c','s',' ','C','o','.'	製造社名（Manifacturer） NEC Electronics Co.

(1) ディスクリプタの構成

サンプル・プログラムにおけるディスクリプタの構成を次に示します。前述したディスクリプタ類を次の構成で準備します。

注意　Device Descriptor/Configuration Descriptor/String Descriptorは，それぞれ別々のGet Descriptorリクエストによってアクセスされます。Interface Descriptorは，Configuration Descriptorの一部としてアクセスされます。

図2－12　ディスクリプタ構成図



2. 7. 4　データ・マクロ

USBバス・ドライバ内で使用する各種データ・マクロ（データ・タイプ，戻り値など）について，次に示します。

(1) データ・タイプ

データ・タイプのマクロ定義は，ヘッダ・ファイルnectools32¥USB_BUS¥inc¥types.hで行われています。

注意　このサンプル・プログラムでは，特に使用しているデータ・タイプはありません。

(2) 戻 り 値

戻り値のマクロ定義は，ヘッダ・ファイルnectools32¥USB_BUS¥inc¥errno.hで行われています。

USBバス・ドライバ関数からの戻り値一覧を次に示します。

注意　このサンプル・プログラムでは，特に使用しているデータ・タイプはありません。

表2－6　戻り値一覧

マクロ	数値	意　味
DEV_OK	0	正常終了
DEV_ERROR	－1	異常終了

2. 7. 5　データ構造体

USBバス・ドライバが使用するデータ構造体について，次に示します。

(1) USBデバイス・リクエスト構造体

USBデバイス・リクエスト構造体の定義は，USB用ヘッダ・ファイルnectools32¥V850USB_BUS¥src¥USBF¥usbf850.hで行われています。

USBデバイス・リクエスト構造体USB_SETUPを次に示します。

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

2. 7. 6　関数解説

(1) 概　　要

この章で説明している各処理プログラムの一覧を次に示します。

表2－7　サンプル・プログラムの処理プログラム一覧

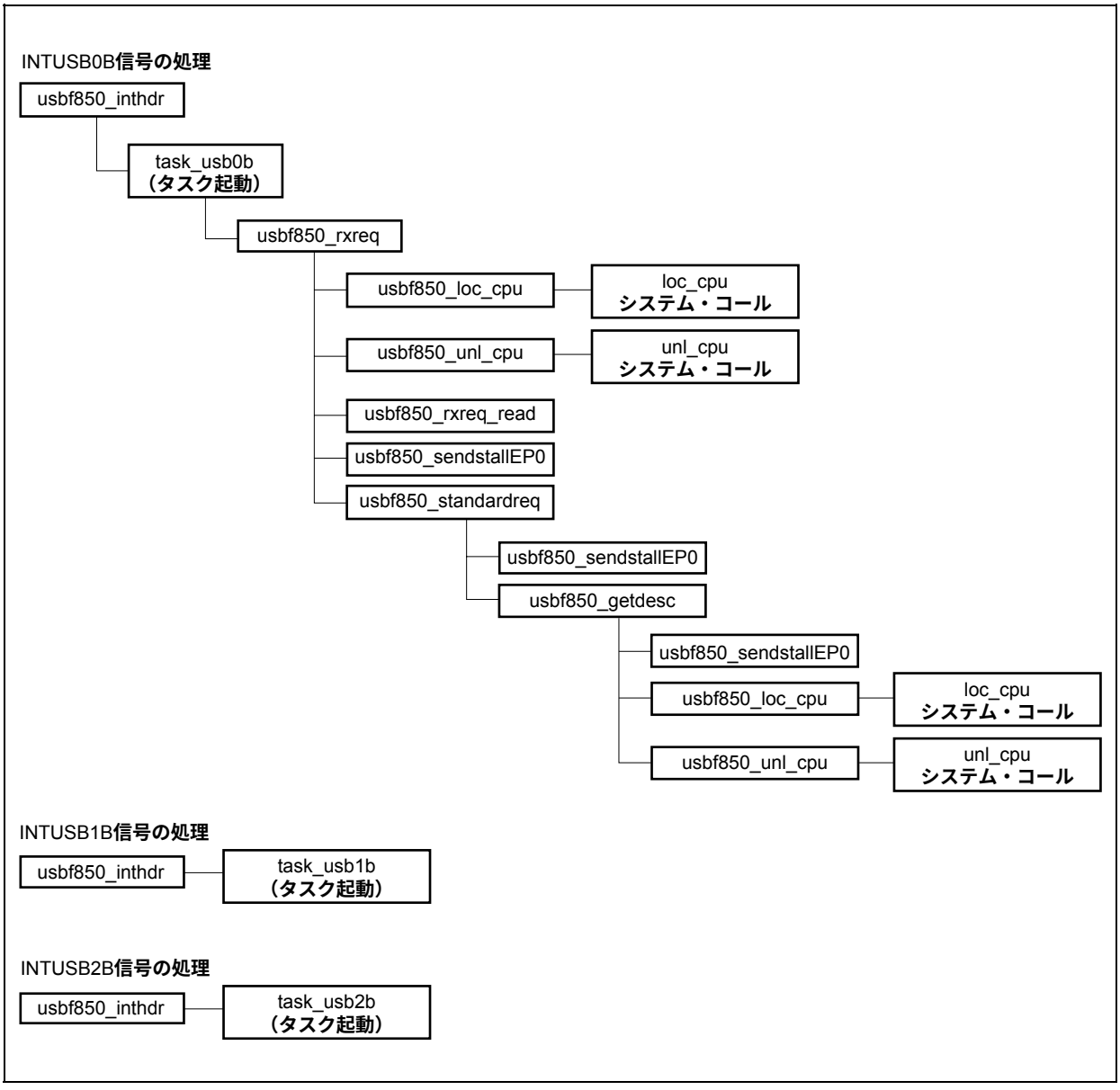
処理プログラム名	関数名	ファイル名	備考
RX850 Pro依存処理プログラム			
CF定義ファイル	－	sys.cf	－
エントリ処理	－	entry.850	アセンブリ言語
ブート処理	boot	boot.850	アセンブリ言語
ハードウエア初期化部	__InitSystemTimer	init.c	C言語
初期化ハンドラ	varfunc	varfunc.c	C言語
ヘッダ・ファイル	－	init.h	－
ボード依存部処理プログラム			
ポートの初期化	port850_reset	port.c	C言語
ヘッダ・ファイル	－	port.h	－
USBバス・ドライバ処理プログラム			
初期化関数	usbf850_init	usbf850.c	C言語
割り込みハンドラ（INTUSB0B信号）	usbf850_inthdr	usbf850.c	C言語
割り込みハンドラ（INTUSB1B信号）	usbf850_inthdr1	usbf850.c	C言語
割り込みハンドラ（INTUSB2B信号）	usbf850_inthdr2	usbf850.c	C言語
割り込み処理用タスク（INTUSB0B信号）	task_usb0b	usbf850.c	C言語
割り込み処理用タスク（INTUSB1B信号）	task_usb1b	usbf850.c	C言語
割り込み処理用タスク（INTUSB2B信号）	task_usb2b	usbf850.c	C言語
データ送信関数	usbf850_data_send	usbf850.c	C言語
データ受信関数	usbf850_data_receive	usbf850.c	C言語
Nullデータ送信関数（エンドポイント0）	usbf850_sendnullEP0	usbf850.c	C言語
Stall応答処理関数（エンドポイント0）	usbf850_sendstallEP0	usbf850.c	C言語
Stall応答処理関数（エンドポイント1）	usbf850_bulkin1_stall	usbf850.c	C言語
Stall応答処理関数（エンドポイント2）	usbf850_bulkout1_stall	usbf850.c	C言語
システム・コール呼び出し関数（loc_cpu）	usbf850_loc_cpu	usbf850.c	C言語
システム・コール呼び出し関数（unl_cpu）	usbf850_unl_cpu	usbf850.c	C言語
リクエスト処理関数	usbf850_rxreq	usbf850.c	C言語
リクエスト・データ読み出し関数	usbf850_rxreq_read	usbf850.c	C言語
標準リクエスト処理関数	usbf850_standardreq	usbf850.c	C言語
Get Descriptorリクエスト処理関数	usbf850_getdesc	usbf850.c	C言語
リクエスト処理関数設定用Stall応答処理関数（エンドポイント0）	usbf850_sstall_ctrl	usbf850.c	C言語
USB用ヘッダ・ファイル	－	usbf850.h	－
USB用ディスクリプタ宣言	－	usbf850desc.h	－
ヘッダ・ファイル			
データ・タイプ宣言	－	types.h	－
戻り値宣言	－	errno.h	－
ビルド・ファイル	－	usb_bus.bld	－
セクション・マップ・ファイル	－	common.lx	－

(2) 関数ツリー

サンプル・プログラムの呼び出し関係（関数ツリー）を次に示します。

注意　usbf850_initは，初期化ハンドラから呼び出されます。

図2－13　サンプル・プログラムの関数ツリー



(3) 関数解説

このサンプル・プログラムの関数群について，次の記述フォーマットにしたがって解説します。

XXXX … ①	**発行有効範囲：－－－－** … ②

[概　　要] … ③

－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－

－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－

[C言語形式] … ④

－－－－－－－－－

[パラメータ] … ⑤

I/O	パラメータ	説　　明

[機　　能] … ⑥

－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－

－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－

[戻 り 値] … ⑦

－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－

－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－

① 名称

関数の名称を示しています。

② 発行有効範囲

関数の発行が可能な処理プログラムの種別を示しています。

タスク	：タスクからだけ発行可能
非タスク	：非タスクからだけ発行可能
タスク｜非タスク	：タスク，非タスクのどちらからも発行可能
－	：割り込みハンドラまたはタスクのため，関数コールできない

③ 概要

関数の機能概要を示しています。

④ C言語形式

関数をC言語で記述された処理プログラムから発行するときの記述形式を示しています。

⑤ パラメータ

関数のパラメータを次の形式で示しています。

I/O	パラメータ	説　明
A	B	C

A ：パラメータの種類

I …USBファンクション・コントローラへの入力パラメータ

O …USBファンクション・コントローラからの出力パラメータ

B ：パラメータのデータ・タイプ

C ：パラメータの説明

⑥ 機能

関数の機能詳細を示しています。

⑦ 戻り値

関数からの戻り値を，データ・マクロおよび数値で示しています。

usbf850_init　　発行有効範囲：非タスク｜タスク

[概　　要]

V850E/ME2に内蔵しているUSBファンクション・コントローラの初期化処理を行う。

[C言語形式]

```
void usbf850_init (void)
```

[パラメータ]

I/O	パラメータ	説　　明
–	–	–

[機　　能]

ソフトウエア初期化部から呼び出され，V850E/ME2に内蔵しているUSBファンクション・コントローラの初期化処理を行います。

備考　初期化処理についての詳細は，2. 7. 2（1）**初期化処理**を参照してください。

[戻 り 値]

なし

usbf850_inthdr

発行有効範囲：－

[概　　要]

V850E/ME2に内蔵しているUSBファンクション・コントローラ用割り込みハンドラ（INTUSB0B信号用）。

[C言語形式]

```
ID usbf850_inthdr (void)
```

[パラメータ]

I/O	パラメータ	説　　明
－	－	－

[機　　能]

INTUSB0B信号（USBファンクション・ステータス0）により起動される割り込みハンドラです。

サンプル・プログラムでは，割り込み要因を調べCPUDEC割り込みだった場合だけ，割り込み処理用のタスク（task_usb0b）を起動します。このハンドラは，CF定義ファイルで定義されています。

[戻 り 値]

オブジェクトID番号（タスクのID番号）

usbf850_inthdr1

発行有効範囲：－

[概　　要]

V850E/ME2に内蔵しているUSBファンクション・コントローラ用割り込みハンドラ（INTUSB1B信号用）。

[C言語形式]

```
ID usbf850_inthdr1 (void)
```

[パラメータ]

I/O	パラメータ	説　　明
－	－	－

[機　　能]

INTUSB1B信号（USBファンクション・ステータス1）により起動される割り込みハンドラです。

サンプル・プログラムでは，割り込み処理用のタスク(task_usb1b)を起動します。このハンドラは，CF定義ファイルで定義されています。

注意　このサンプル・プログラムでは，この関数を使用していません。

[戻 り 値]

オブジェクトID番号（タスクのID番号）

usbf850_inthdr2

発行有効範囲：－

[概　　要]

V850E/ME2に内蔵しているUSBファンクション・コントローラ用割り込みハンドラ（INTUSB2B信号用）。

[C言語形式]

```
ID usbf850_inthdr2 (void)
```

[パラメータ]

I/O	パラメータ	説　　明
－	－	－

[機　　能]

INTUSB2B信号（USBファンクション・ステータス2）により起動される割り込みハンドラです。

サンプル・プログラムでは，割り込み処理用のタスク（task_usb2b）を起動します。このハンドラは，CF定義ファイルで定義されています。

注意　このサンプル・プログラムでは，この関数を使用していません。

[戻 り 値]

オブジェクトID番号（タスクのID番号）

task_usb0b

発行有効範囲：－

[概　　要]

INTUSB0B信号による割り込み処理を行う。

[C言語形式]

```
void task_usb0b (VP exinf)
```

[パラメータ]

I/O	パラメータ	説　　明
I	VP exinf	拡張情報

対象タスクに関する“ユーザ独自の情報”を格納するための領域で，ユーザが自由に利用できます。
exinfに設定された情報は，処理プログラム（タスク，非タスク）からref_tskシステム・コールを発行することにより，ダイナミックに獲得できます。

備考　システム・コールについての詳細は，RX850 Pro　**ユーザーズ・マニュアル　基礎編**を参照してください。

[機　　能]

INTUSB0B割り込み信号（USBファンクション・ステータス0割り込み）用の割り込みハンドラから起動されるタスクです。サンプル・プログラムでは，usbf850_rxreq関数を呼び出し，USB標準デバイス・リクエストの処理を行います。

注意　**このサンプル・プログラムでは，**V850E/ME2**に内蔵している**USB**ファンクション・コントローラで自動応答しない標準デバイス・リクエスト「**Get Descriptor（String Descriptor）**」だけを処理します。**

[戻 り 値]

なし

task_usb1b

発行有効範囲：－

[概　　要]

INTUSB1B信号による割り込み処理を行う。

[C言語形式]

```
void task_usb1b (VP exinf)
```

[パラメータ]

I/O	パラメータ	説　　明
I	VP exinf	拡張情報

対象タスクに関する“ユーザ独自の情報”を格納するための領域で，ユーザが自由に利用できます。
exinfに設定された情報は，処理プログラム（タスク，非タスク）からref_tskシステム・コールを発行することにより，ダイナミックに獲得できます。

備考　システム・コールについての詳細は，RX850 Pro　**ユーザーズ・マニュアル　基礎編**を参照してください。

[機　　能]

INTUSB1B割り込み信号（USBファンクション・ステータス1割り込み）用の割り込みハンドラから起動されるタスクです。サンプル・プログラムでは該当する処理がないため，未処理で戻ります。

注意　このサンプル・プログラムでは，この関数を使用していません。

[戻 り 値]

なし

task_usb2b

発行有効範囲：－

[概　　要]

INTUSB2B信号による割り込み処理を行う。

[C言語形式]

```
void task_usb2b (VP exinf)
```

[パラメータ]

I/O	パラメータ	説　　明
I	VP exinf	拡張情報

対象タスクに関する“ユーザ独自の情報”を格納するための領域で，ユーザが自由に利用できます。
exinfに設定された情報は，処理プログラム（タスク，非タスク）からref_tskシステム・コールを発行することにより，ダイナミックに獲得できます。

備考　システム・コールについての詳細は，RX850 Pro　**ユーザーズ・マニュアル　基礎編**を参照してください。

[機　　能]

INTUSB2B割り込み信号（USBファンクション・ステータス2割り込み）用の割り込みハンドラから起動されるタスクです。サンプル・プログラムでは該当する処理がないため，未処理で戻ります。

注意　このサンプル・プログラムでは，この関数を使用していません。

[戻 り 値]

なし

usbf850_data_send　　発行有効範囲：非タスク｜タスク

[概　　要]

USBファンクション・コントローラ用データ送信関数。

[C言語形式]

```
long usbf850_data_send (unsigned char* data, long len, char ep)
```

[パラメータ]

I/O	パラメータ	説　　明
I	unsigned char* data	送信データの先頭アドレス
I	long len	データ・サイズ
I	char ep	エンドポイント番号

[機　　能]

dataで指定されたアドレスから，lenで指定されたサイズ分のデータを，epで指定されたエンドポイントで送信処理を行います。

注意　このサンプル・プログラムでは，この関数を使用していません。

[戻 り 値]

送信時のステータス

DEV_ERROR ：エンドポイント番号が不正

DEV_OK　　：正常終了

usbf850_data_receive　　発行有効範囲：非タスク｜タスク

[概　　要]

USBファンクション・コントローラ用データ受信関数。

[C言語形式]

```
long usbf850_data_receive (unsigned char* data, long len, char ep)
```

[パラメータ]

I/O	パラメータ	説　　明
I	unsigned char* data	受信データ用バッファの先頭アドレス
I	long len	データ・サイズ
I	char ep	エンドポイント番号

[機　　能]

epで指定されたエンドポイントのバッファから，lenで指定されたサイズ分データを読み出し，dataで指定されたアドレスに格納します。

注意　このサンプル・プログラムでは，この関数を使用していません。

[戻 り 値]

受信時のステータス

DEV_ERROR ：受信データ・サイズが不正，またはエンドポイント番号が不正

DEV_OK　　：正常終了

usbf850_sendnullEP0

発行有効範囲：非タスク｜タスク

[概　　要]

コントロール・エンドポイント（エンドポイント0）用Nullデータ送信関数。

[C言語形式]

```
void usbf850_sendnullEP0 (void)
```

[パラメータ]

I/O	パラメータ	説　　明
–	–	–

[機　　能]

コントロール・エンドポイント（エンドポイント0）で，Nullデータ（データ・サイズ0のデータ）の送信処理を行います。

[戻 り 値]

なし

usb850_sendstallEP0

発行有効範囲：非タスク｜タスク

[概　　要]

コントロール・エンドポイント（エンドポイント0）用STALL応答関数。

[C言語形式]

```
void usbf850_sendstallEP0 (void)
```

[パラメータ]

I/O	パラメータ	説　　明
–	–	–

[機　　能]

コントロール・エンドポイント（エンドポイント0）でSTALL応答を設定します。

[戻 り 値]

なし

usbf850_bulkin1_stall　　発行有効範囲：非タスク｜タスク

[概　　要]

バルク・エンドポイント（エンドポイント1）用STALL応答関数。

[C言語形式]

```
void usbf850_bulkin1_stall (void)
```

[パラメータ]

I/O	パラメータ	説　　明
–	–	–

[機　　能]

バルク・エンドポイント（エンドポイント1）でSTALL応答を設定します。

注意　このサンプル・プログラムでは，この関数を使用していません。

[戻 り 値]

なし

usbf850_bulkout1_stall　　発行有効範囲：非タスク｜タスク

[概　　要]

バルク・エンドポイント（エンドポイント2）用STALL応答関数。

[C言語形式]

```
void usbf850_bulkout1_stall (void)
```

[パラメータ]

I/O	パラメータ	説　　明
−	−	−

[機　　能]

バルク・エンドポイント（エンドポイント2）でSTALL応答を設定します。

注意　このサンプル・プログラムでは，この関数を使用していません。

[戻 り 値]

なし

usbf850_loc_cpu　　発行有効範囲：タスク

[概　　要]

マスカブル割り込みの受け付けとディスパッチ処理を禁止する。

[C言語形式]

```
void usbf850_loc_cpu (void)
```

[パラメータ]

I/O	パラメータ	説　　明
–	–	–

[機　　能]

loc_cpuシステム・コールを呼び出します。

備考　システム・コールについての詳細は，RX850 Pro　**ユーザーズ・マニュアル　基礎編**を参照してください。

[戻 り 値]

なし

usbf850_unl_cpu　　発行有効範囲：タスク

[概　　要]

マスカブル割り込みの受け付けとディスパッチ処理を許可する。

[C言語形式]

```
void usbf850_unl_cpu (void)
```

[パラメータ]

I/O	パラメータ	説　　明
–	–	–

[機　　能]

unl_cpuシステム・コールを呼び出します。

備考　システム・コールについての詳細は，RX850 Pro　**ユーザーズ・マニュアル　基礎編**を参照してください。

[戻 り 値]

なし

usbf850_rxreq

発行有効範囲：非タスク｜タスク

[概　　要]

USBリクエスト処理を行う。

[C言語形式]

```
void usbf850_rxreq (void)
```

[パラメータ]

I/O	パラメータ	説　　明
–	–	–

[機　　能]

INTUSB0B割り込み信号により起動されるtask_usb0bタスクによって，呼び出されます。SETUPデータの読み出し処理を呼び出し，読み出したデータを解析します。解析結果に基づき，USBのリクエスト処理を呼び出します。

[戻 り 値]

なし

usbf850_rxreq_read

発行有効範囲：非タスク｜タスク

[概　　要]

USBリクエスト・データを読み出す。

[C言語形式]

```
void usbf850_rxreq_read (void)
```

[パラメータ]

I/O	パラメータ	説　　明
–	–	–

[機　　能]

コントロール転送（エンドポイント0）で，Setupトークンに続いて受信されるSETUPデータを読み出します。

SETUPデータは，通常のデータと区別され専用のレジスタに格納されており，必ず8バイト・リードします。

[戻 り 値]

なし

usbf850_standardreq　　発行有効範囲：非タスク｜タスク

[概　　要]

USB標準リクエストの処理を行う。

[C言語形式]

```
void usbf850_standardreq (void)
```

[パラメータ]

I/O	パラメータ	説　　明
–	–	–

[機　　能]

SETUPデータ読み出し後，リクエストの内容が標準リクエストであった場合に呼び出されます。Get Descriptor リクエストであることを確認し，usbf850_getdesc関数を呼び出します。

[戻 り 値]

なし

usbf850_getdesc　　発行有効範囲：非タスク｜タスク

[概　　要]

USB標準リクエストのGet Descriptor（String Descriptor）の処理を行う。

[C言語形式]

```
void usbf850_getdesc (void)
```

[パラメータ]

I/O	パラメータ	説　　明
−	−	−

[機　　能]

usbf850_standardreq関数から呼び出され，USB標準リクエストのGet Descriptor（String Descriptor）の処理を行います。Get Descriptor（String Descriptor）リクエスト以外の場合は，STALL応答します。

[戻 り 値]

なし

usbf850_sstall_ctrl　　発行有効範囲：非タスク｜タスク

[概　　要]

コントロール・エンドポイント（エンドポイント0）用STALL応答処理関数。

[C言語形式]

```
void usbf850_sstall_ctrl (void)
```

[パラメータ]

I/O	パラメータ	説　　明
–	–	–

[機　　能]

コントロール・エンドポイント（エンドポイント0）でSTALL応答を設定します。

注意　このサンプル・プログラムでは，この関数を使用していません。

[戻 り 値]

なし

第3章　USBストレージ・クラス用ドライバ

3.1 概　　説

3.1.1 概　　要

USBストレージ・クラス用ドライバは，V850E/ME2に内蔵しているUSBファンクション・コントローラ用のサンプル・プログラムです。Universal Serial Bus Specification Revision 1.1，およびUniversal Serial Bus Mass Storage Class Bulk-Only Transport Revision 1.0に準拠しており，組み込み型制御用リアルタイム・オペレーティング・システムRX850 Pro（μITRON3.0仕様準拠）上で動作します。

このサンプル・プログラムでは，コントロール・エンドポイント（エンドポイント番号0），およびバルク・エンドポイントのINとOUT（エンドポイント番号1, 2）の3つを使用します。その上で，Windows XPに標準のストレージ・クラス用ホスト・ドライバと接続し，ストレージ・デバイス（仮想デバイス）の制御を行います。クラスとしては，Mass Storageクラスが定義されています。

このサンプル・プログラムは，ハードウエアの実行環境として評価ボードSolutionGear MINI（SG-703111-1）を使用しています。SolutionGear MINIとサンプル・プログラムをそのまま使用する場合は，3. 6　**ロード・モジュール**の手順に従って実行オブジェクトを作成し，3. 2　**ロード・モジュールの実行**に従って動作を確認してください。

SolutionGear MINIからほかのターゲット・ボードに変更して使用する場合は，3. 3　**システム構築**，3. 4 RX850 Pro**依存処理プログラム**，3. 5　**セクション・マップ・ファイル**を参照し，ボードの仕様に従って変更してください。

SolutionGear MINIおよびサンプル・プログラムの両方を変更して使用する場合は，3. 3　**システム構築**，3. 4 RX850 Pro**依存処理プログラム**，3. 5　**セクション・マップ・ファイル**，3. 6　**ロード・モジュール**，3. 7　USB**ドライバの機能**を参照し，必要な変更を行ってください。

USBストレージ・クラス用ドライバの位置付けを次に示します。

注意　このサンプル・プログラムは，Mass Storage**デバイス（インタフェース・クラス：マス・ストレージ，インタフェース・サブクラス：**SCSI**，インタフェース・プロトコル：**Bulk-Only Transport**プロトコル）として動作します。今回使用するストレージ・デバイスは，結合される論理ユニットはなく，メモリ領域を確保し擬似的にリムーバブル・ディスクが繋がれているように動作するものです（ブロック・サイズ：**512**バイト，論理ブロック数：**192**，容量：**96 K**バイト）**。

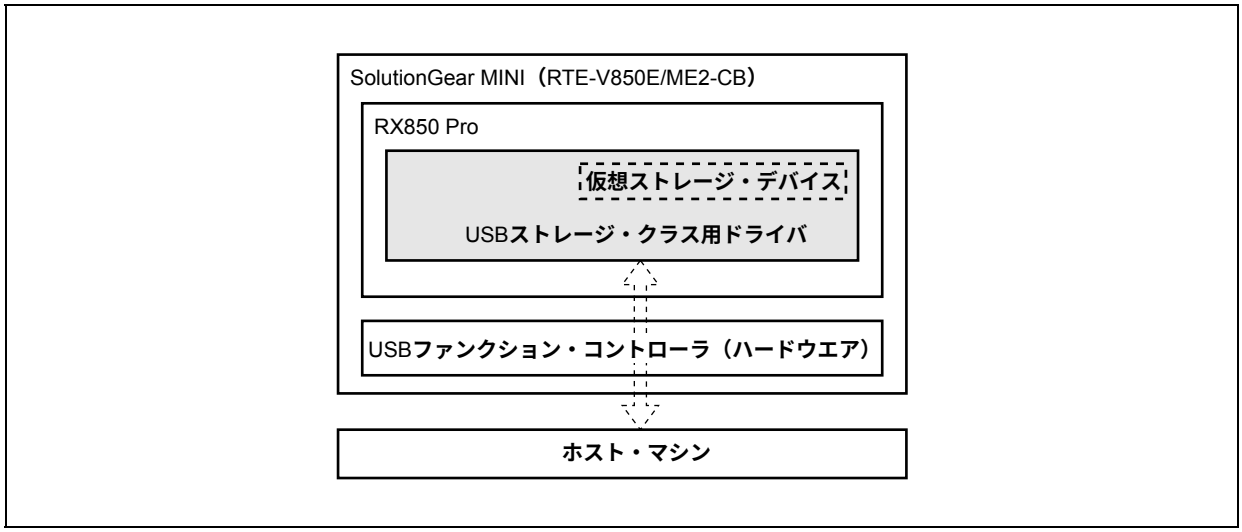
備考1.　USBのMass Storageクラスの詳細は，

- Universal Serial Bus Mass Storage Class Specification Overview Revision 1.1
- Universal Serial Bus Mass Storage Class Bulk-Only Transport Revision 1.0
- Universal Serial Bus Mass Storage Class UFI Command Specification Revision 1.0

を参照してください。

2.　3. 2. 1　**ロード・モジュールの実行手順は，**3. 1. 3　**実行環境**で示した環境を想定して記述しています。

図3－1　USBストレージ・クラス用ドライバの位置付け



3. 1. 2　開発環境

サンプル・プログラムを使用してシステムを開発するうえで必要となるハードウエア環境およびソフトウエア環境として，次に示す環境を想定して記述しています。

・ハードウエア環境

ホスト・マシン	：PC/AT互換機（OS：Windows XP）

・ソフトウエア環境

リアルタイムOS	：RX850 Pro Version 3.15
USBストレージ・クラス用ドライバ	：この章で説明するサンプル・プログラム一式
Cコンパイラ・パッケージ	：MULTI2000 （CCV850 Version 3.5（Green Hills Software, Inc.製））

注意　サンプルのビルド・ファイルで扱うディレクトリ構成と違う場合は，環境にあわせてビルド・ファイルを編集してください。

備考　ビルド・ファイルの記述内容については，MULTI（Green Hills Software, Inc.製）のヘルプを参照してください。

3. 1. 3　実行環境

サンプル・プログラムを使用したロード・モジュールを実行するときのハードウエア環境およびソフトウエア環境として，次に示す環境を想定して記述しています。

・ハードウエア環境

ホスト・マシン	：PC/AT互換機（OS：Windows XP）
IE制御用マシン	：PC/AT互換機（OS：Windows XP）
ターゲット・ボード	：SolutionGear MINI（SG-703111-1）
インサーキット・エミュレータ（IE）	：N-wire IE（RTE-2000-TP）（マイダス・ラボ社製）
JTAGプローブ	
USBケーブル	

・ソフトウエア環境

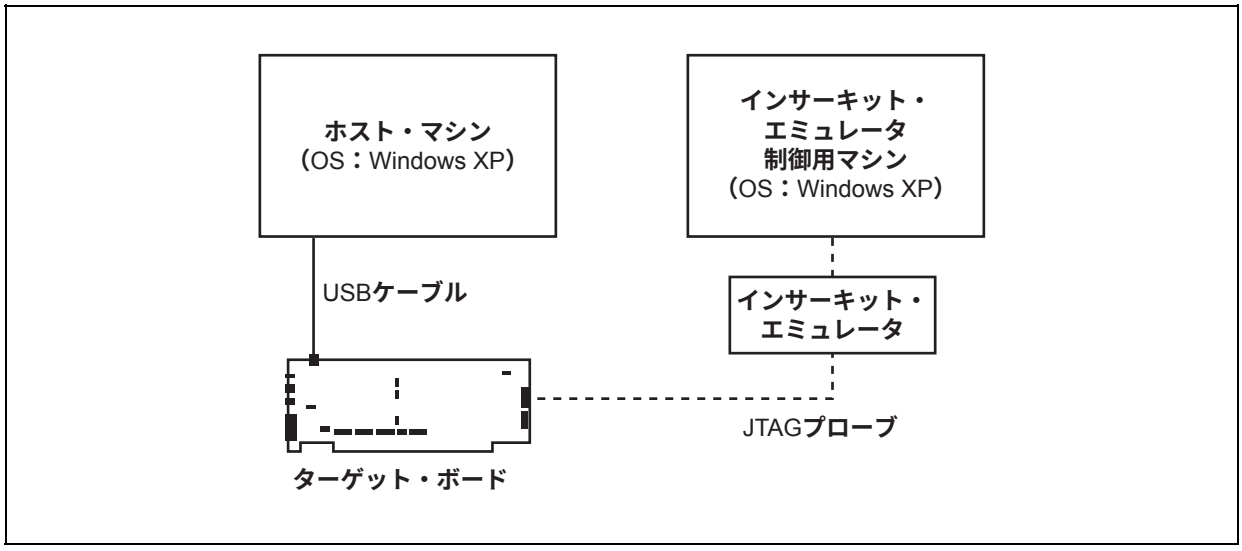
IE用ソフトウエア	：PARTNER Setup Program Version 1.242

備考1. 実行環境のセットアップ方法についての詳細は，**付録**A　SG-703111-1**ボード**，および SG-703111-1　**ユーザーズ・マニュアル**を参照してください。

2. インサーキット・エミュレータ（RTE-2000-TP）のセットアップ方法についての詳細は，RTE-2000-TP　**ハードウエア・ユーザーズ・マニュアル**を参照してください。

3. PARTNERについての詳細は，PARTNER　**ユーザーズ・マニュアル**　V800**シリーズ「**V800**シリーズ共通編」，「**NB85E-CB**個別編」**を参照してください。

図3－2　**実行環境**



3. 2　ロード・モジュールの実行

3. 2. 1　ロード・モジュールの実行手順

このサンプル・プログラムを使用したロード・モジュールを例にとり，3. 1. 3　**実行環境**で示した環境において実行するときの手順を，次に示します。

① **インサーキット・エミュレータ（IE）制御用マシンの準備**

IE制御用マシンに電源を投入し起動しておきます。また，インサーキット・エミュレータにも電源を投入し準備しておきます。

② **ホスト・マシンの準備**

ホスト・マシンに電源を投入し起動しておきます（ホスト・マシンは，IE制御用マシンで兼用できますが，開発時には別のホスト・マシンを用意することを強く推奨します）。

③ SG-703111-1**ボードのリセット**

SG-703111-1**ボードのリセット・ボタン**RESET**を押下し，**SG-703111-1**ボードのリセットを行います。**

④ **インサーキット・エミュレータの起動**

インサーキット・エミュレータを起動します。

Windows**の「スタート」ボタンから「すべてのプログラム」－「**PARTNER**」－「**RPTSETUP (NB85ET)**」を選択します。**

[PARTNER**の起動例]**

次に示す画面が起動されます。

[PARTNERの起動画面]

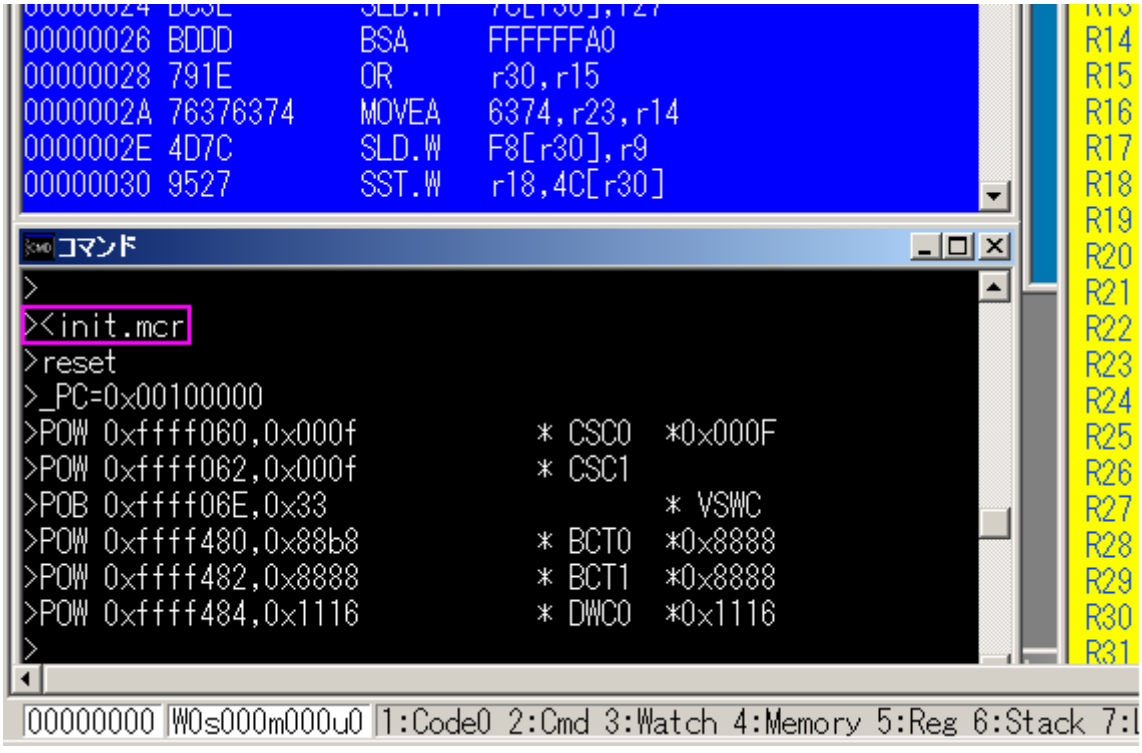
「開く」ボタンをクリックし，プロジェクト・ファイルを指定すると，「起動」ボタンが有効になります。次に，「起動」ボタンをクリックするとPARTNERが起動されます。起動後，ボードの設定を行います。このとき，起動時に読み込まれる設定ファイルをあらかじめ作成しておくと便利です。ここで説明するサンプル用の設定ファイルについては，**付録**A　SG-703111-1**ボード**，および，PARTNER　**ユーザーズ・マニュアル**　V800**シリーズ**「V800**シリーズ共通編**」，「NB85E-CB**個別編**」を参照してください。

注意1.　**インサーキット・エミュレータの起動は，必ずターゲット・ボードの電源投入後に行ってください。**

2.　**インサーキット・エミュレータの起動後，ターゲット・ボードのリセットのために設定ファイルを読み込みたい場合は，コマンド・ウィンドウから，次のコマンド入力例にならって設定ファイル（例では**init.mcr）**を読み込むことが出来ます。**

[コマンド入力例]

```
><init.mcr<Enter>
```



⑤　ロード・モジュールの読み込み

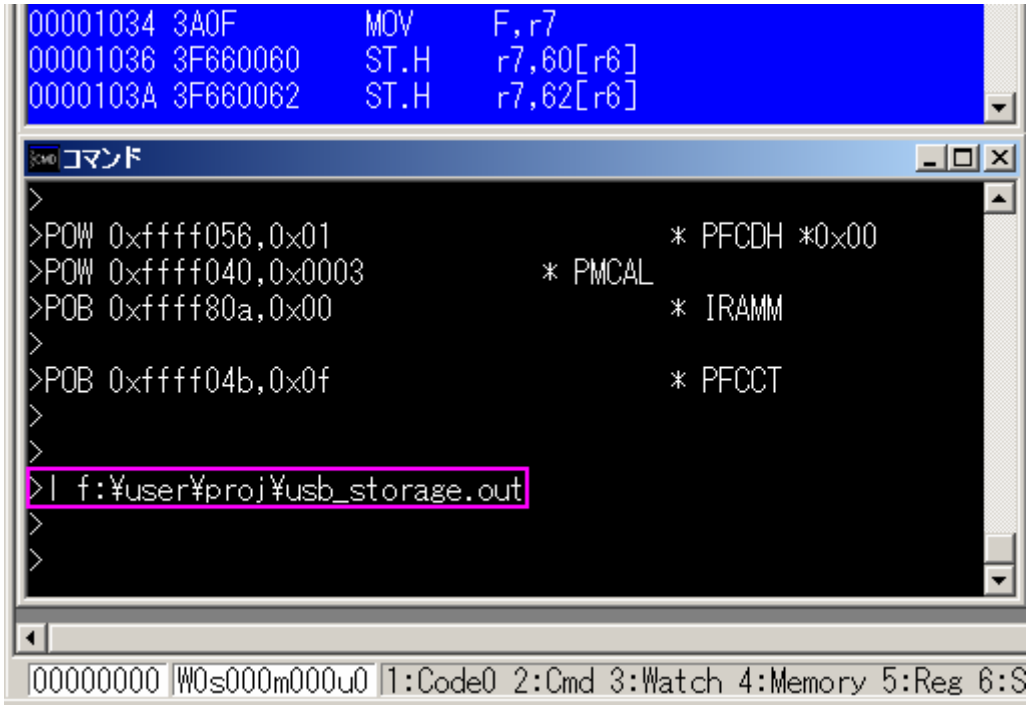
インサーキット・エミュレータの機能を使って，ロード・モジュールをボード上に読み込みます。

ツールバーの「ファイル」－「ロード」からロード・モジュールを読み込むか，コマンド・ウィンドウからL（ファイル読み込み）コマンドを使用してロード・モジュール（例ではusb_storage.out）を読み込みます。

[ファイルのロード例]

[コマンド入力例]

```
>l usb_storage.out<Enter>
```

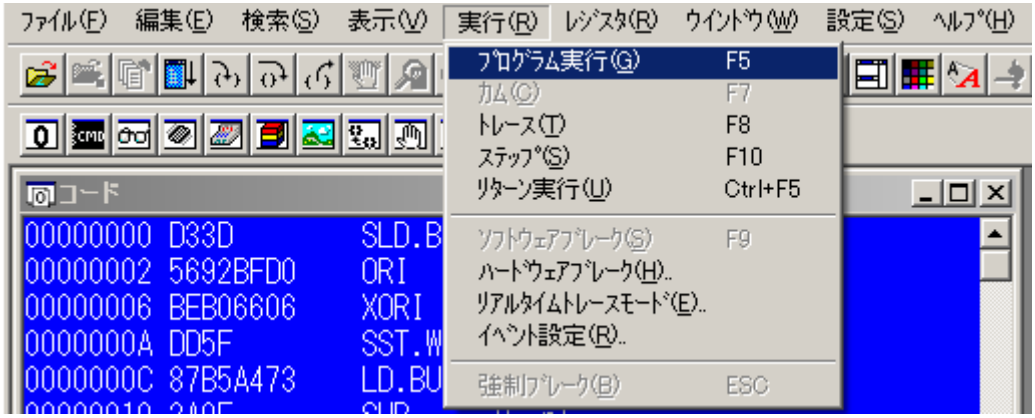


⑥　実行

F5キーまたは実行ボタンで，ボード上に読み込まれたコードを実行します。

注意　ツールバーの「実行」－「プログラムの実行」からも同じように実行できます。

[プログラム実行例]



⑦　USB接続

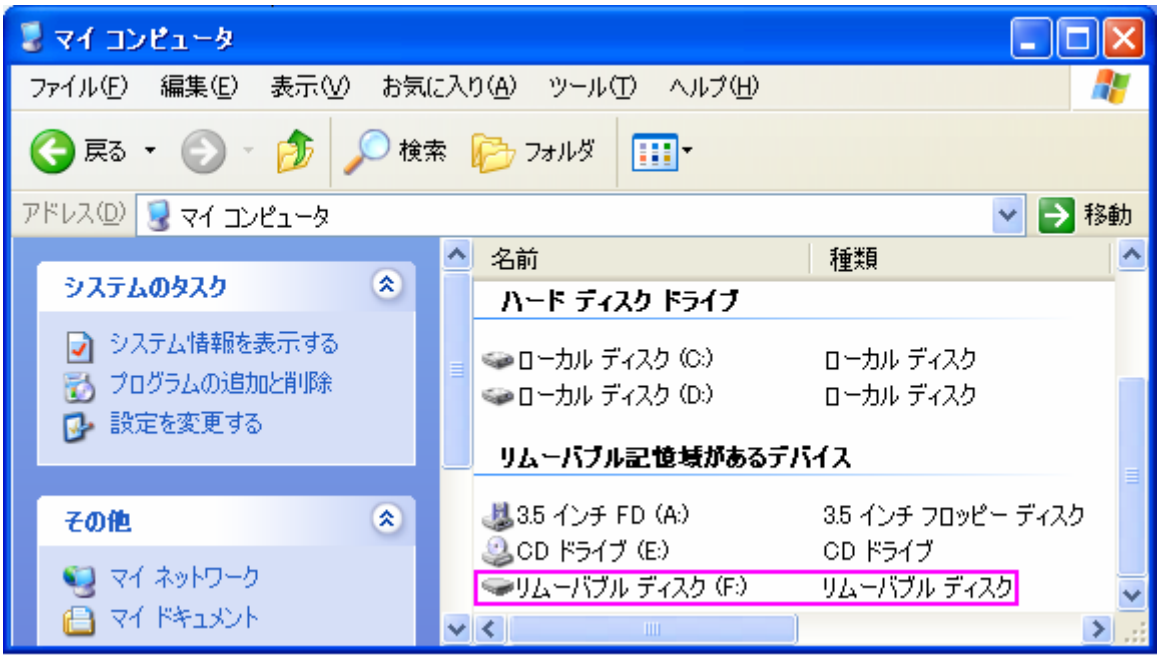
USBケーブルを接続します。

ボード側にBコネクタを接続し，ホスト・マシン側にAコネクタを接続します。

注意1.　**ターゲット・ボードの起動前にUSBを接続しても問題ありません。**

2.　**ホスト・マシン側でデバイスが認識されると，Windows XP標準のUSBストレージ・クラス用ホスト・ドライバが自動的にインストールされます。正常に認識されると，「マイ コンピュータ」上に「リムーバブル ディスク」として表示されます。**

[マイ コンピュータ表示例]

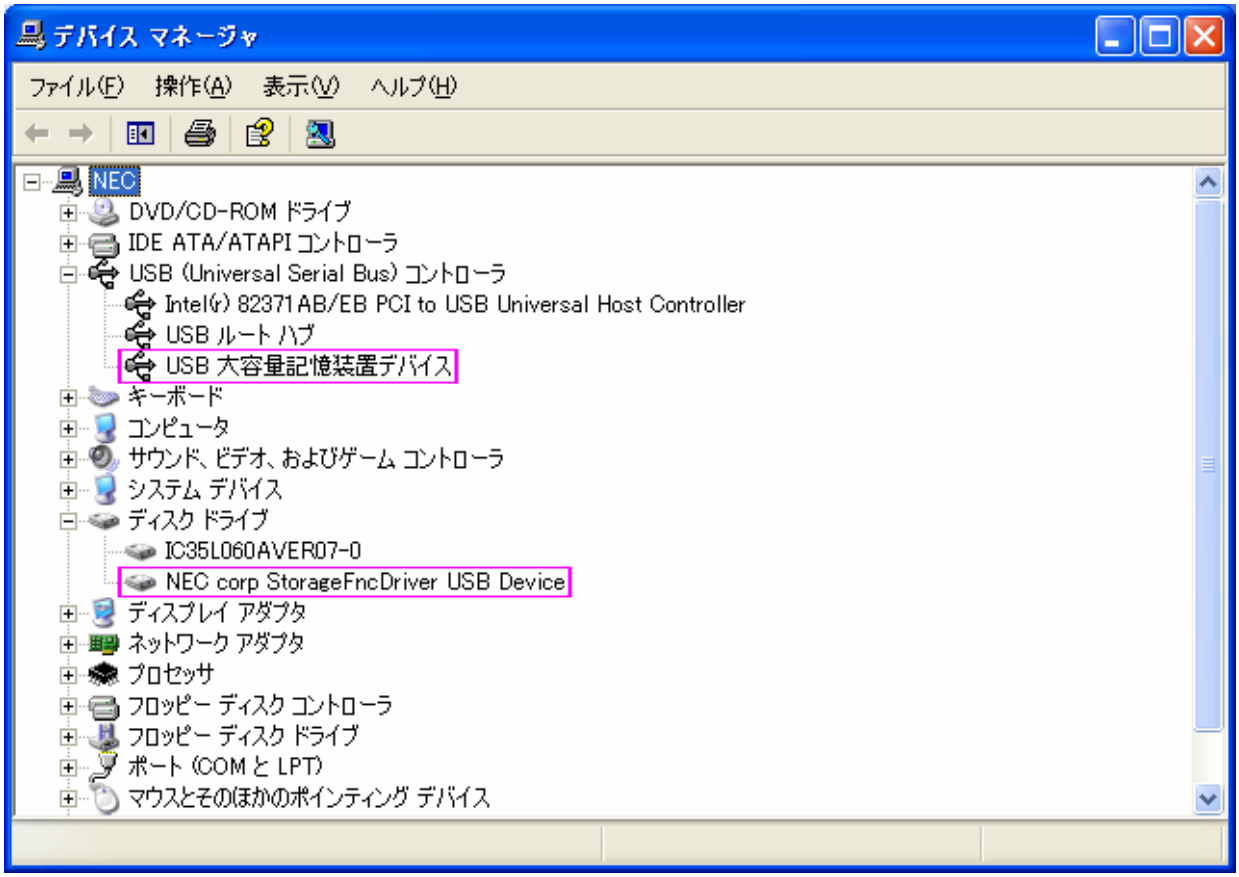


⑧　デバイス マネージャの起動

「マイ コンピュータ」の「プロパティ」から「ハードウェア」のタブを選択します。「デバイス マネージャ」の項目を選択し，「デバイス マネージャ」を起動します。

注意　「マイ コンピュータ」の「管理」や「コントロール パネル」からも起動できます。

[デバイス マネージャ表示例]



⑨　USBデバイスの接続の確認

「デバイス マネージャ」の画面上で，「USBコントローラ」の下に「USB大容量記憶装置デバイス」が，「ディスク ドライブ」の下に「NEC corp StorageFncDriver USB Device」が表示されていることを確認します。

⑩　デバイスの使用方法

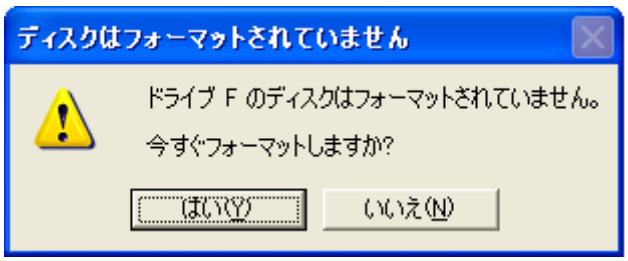
まず，「マイ コンピュータ」上に表示された「リムーバブル ディスク」の「開く」を実行すると，ディスクのフォーマットが要求されます。

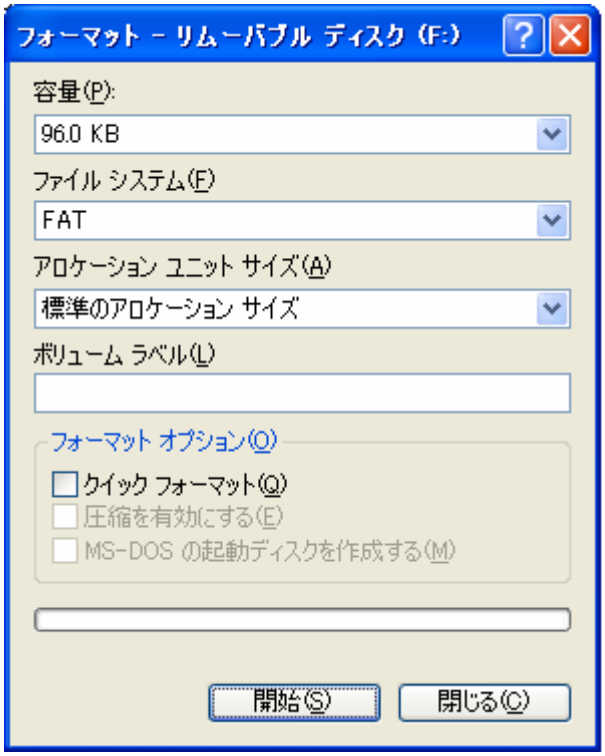
画面の指示にしたがってフォーマットを実行してください。

フォーマットが正常に終了すると，ファイルの読み書き，ファイルの削除など，通常のディスク・デバイスと同じように使用できることが確認できます。

また，ロード・モジュールの実行を止めるまでは，ディスクの内容が保持されます。

[フォーマット実行例]


ディスクはフォーマットされていません
ドライブ F のディスクはフォーマットされていません。
今すぐフォーマットしますか?
はい(Y)
いいえ(N)


フォーマット - リムーバブル ディスク (F:)
容量(P):
96.0 KB
ファイル システム(F)
FAT
アロケーション ユニット サイズ(A)
標準のアロケーション サイズ
ボリューム ラベル(L)
フォーマット オプション(O)
クイック フォーマット(Q)
圧縮を有効にする(E)
MS-DOS の起動ディスクを作成する(M)
開始(S)
閉じる(C)


フォーマット中 - リムーバブル ディスク (F:)
容量(P):
96.0 KB
ファイル システム(F)
FAT
アロケーション ユニット サイズ(A)
標準のアロケーション サイズ
ボリューム ラベル(L)
フォーマット オプション(O)
クイック フォーマット(Q)
圧縮を有効にする(E)
MS-DOS の起動ディスクを作成する(M)
開始(S)
フォーマット中 - リムーバブル ディスク (F:)
フォーマットが完了しました。
OK

⑪　プログラムの終了方法

実行中のプログラムを終了します。

実行中のPARTNERの画面上にある強制終了ボタンをクリックするか，ツールバーの「実行」－「強制ブレーク」を選択し，プログラムの実行を停止します。

[強制終了ボタンによる終了例]

[強制ブレークによる終了例]

⑫　終了方法

インサーキット・エミュレータの終了を行い，ボードを①の手順でリセットします。

ツールバーの「ファイル」－「終了」を選択し，PARTNERを終了してください。

PARTNERを終了してから，ボードを①の手順でリセットします。

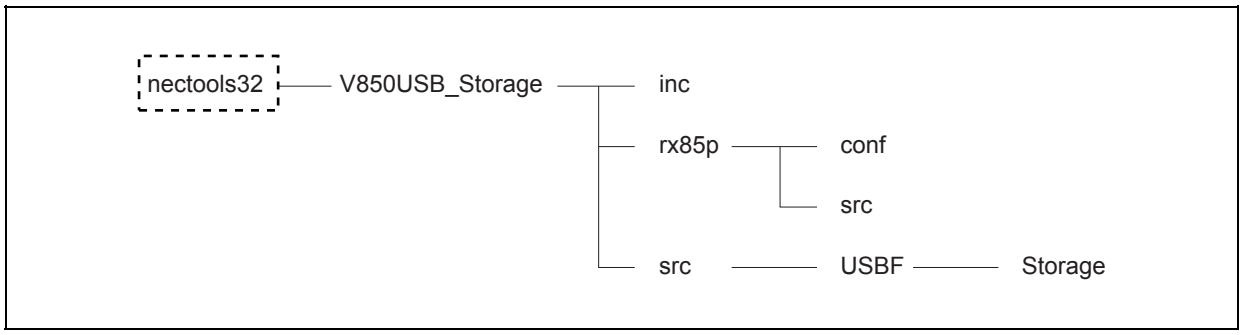
[PARTNERの終了例]

3. 2. 2　ディレクトリ構成

このサンプル・プログラム一式に含まれるファイル群のディレクトリ構成を，次に示します。

注意　USBストレージ・クラス用ドライバ用ファイル郡のディレクトリは，RX850 Proをインストールしたディレクトリ（¥nectools32）の直下に置くことを推奨します。

図3－3　サンプル・プログラムのディレクトリ構成



各ディレクトリの概要を次に示します。

（1）nectools32

RX850 Proをインストールすると作成されるディレクトリです。このディレクトリの直下に，ドライバのディレクトリ（ディレクトリ名：V850USB_Storage）を置きます。

（2）nectools32¥V850USB_Storage

USBストレージ・クラス用ドライバのディレクトリです。

- usb_storage.bld ：USBストレージ・クラス用ドライバのビルド・ファイル
- common.lx　　：セクション・マップ・ファイル

（3）nectools32¥V850USB_Storage¥inc

USBストレージ・クラス用ドライバのヘッダ・ファイルが格納されているディレクトリです。

- errno.h　　：戻り値用ヘッダ・ファイル
- types.h　　：データ・タイプ用ヘッダ・ファイル
- sys.h　　：システム情報ヘッダ・ファイル

注意　sys.h（システム情報ヘッダ・ファイル）は，ビルド時に生成されるファイルです。本来，このファイルは，コンフィギュレータを使用してコマンド操作により生成されます。しかし，サンプルのビルド・ファイルを使用すると，ビルド実行時にコマンドを実行する設定になっているため，あらかじめ生成しておく必要はありません。

（4）nectools32¥V850USB_Storage¥rx85p

RX850 Pro用のファイル群が格納されているディレクトリです。

(5) nectools32¥V850USB_Storage¥rx85p¥conf

RX850 Pro用のシステム・ファイルが格納されているディレクトリです。

- sit.850　　：システム情報テーブル
- svc.850　　：システム・コール・テーブル
- sysi.tbl　　：システム情報テーブル
- sysc.tbl　　：システム・コール・テーブル

注意1. **このディレクトリにあるファイルは，ビルド時に生成されるファイルです。本来，これらのファイルは，コンフィギュレータを使用してコマンド操作により生成されます。しかし，サンプルのビルド・ファイルを使用すると，ビルド実行時にコマンドを実行する設定になっているため，あらかじめ生成しておく必要はありません。**

2. sit.850とsysi.tbl，svc.850とsysc.tblは，それぞれファイルの拡張子が違うだけで，内容は同じものです。

(6) nectools32¥V850USB_Storage¥rx85p¥src

RX850 Pro用のファイルが格納されているディレクトリです。

- boot.850　　：ブート処理用アセンブラ・ファイル
- entry.850　　：エントリ処理用アセンブラ・ファイル
- init.c　　：ハードウエア初期化部ソース・ファイル
- init.h　　：ハードウエア初期化部ヘッダ・ファイル
- sys.cf　　：CF定義ファイル
- varfunc.c　　：ソフトウエア初期化部ソース・ファイル

(7) nectools32¥V850USB_Storage¥src

USBストレージ・クラス用ドライバのボード依存部のファイルが格納されているディレクトリです。

- port.c　　：ポート設定用ソース・ファイル
- port.h　　：ポート設定用ヘッダ・ファイル

(8) nectools32¥V850USB_Storage¥src¥USBF

USBストレージ・クラス用ドライバのUSB処理部のファイルが格納されているディレクトリです。

- usbf850.c　　：USBデバイス用ソース・ファイル
- usbf850.h　　：USBデバイス用ヘッダ・ファイル
- usbf850desc.h　　：USB用ディスクリプタ定義ファイル
- usbf850_dma.c　　：DMA制御用ソース・ファイル
- usbf850_dma.h　　：DMA制御用ヘッダ・ファイル
- usbf850_storage.c　　：USB-ストレージ間インタフェース用ソース・ファイル
- usbf850_storage.h　　：USB-ストレージ間インタフェース用ヘッダ・ファイル

（9）nectools32¥V850USB_Storage¥src¥USBF¥Storage

USBストレージ・クラス用ドライバのストレージ・デバイス処理部のファイルが格納されているディレクトリです。

- ata_ctrl.c　：ストレージ・デバイス・コントロール用ソース・ファイル
- ata.h　：ストレージ・デバイス用ヘッダ・ファイル
- scsi_cmd.c　：SCSIコマンド処理用ソース・ファイル
- scsi.h　：SCSIコマンド処理用ヘッダ・ファイル

3. 3　システム構築

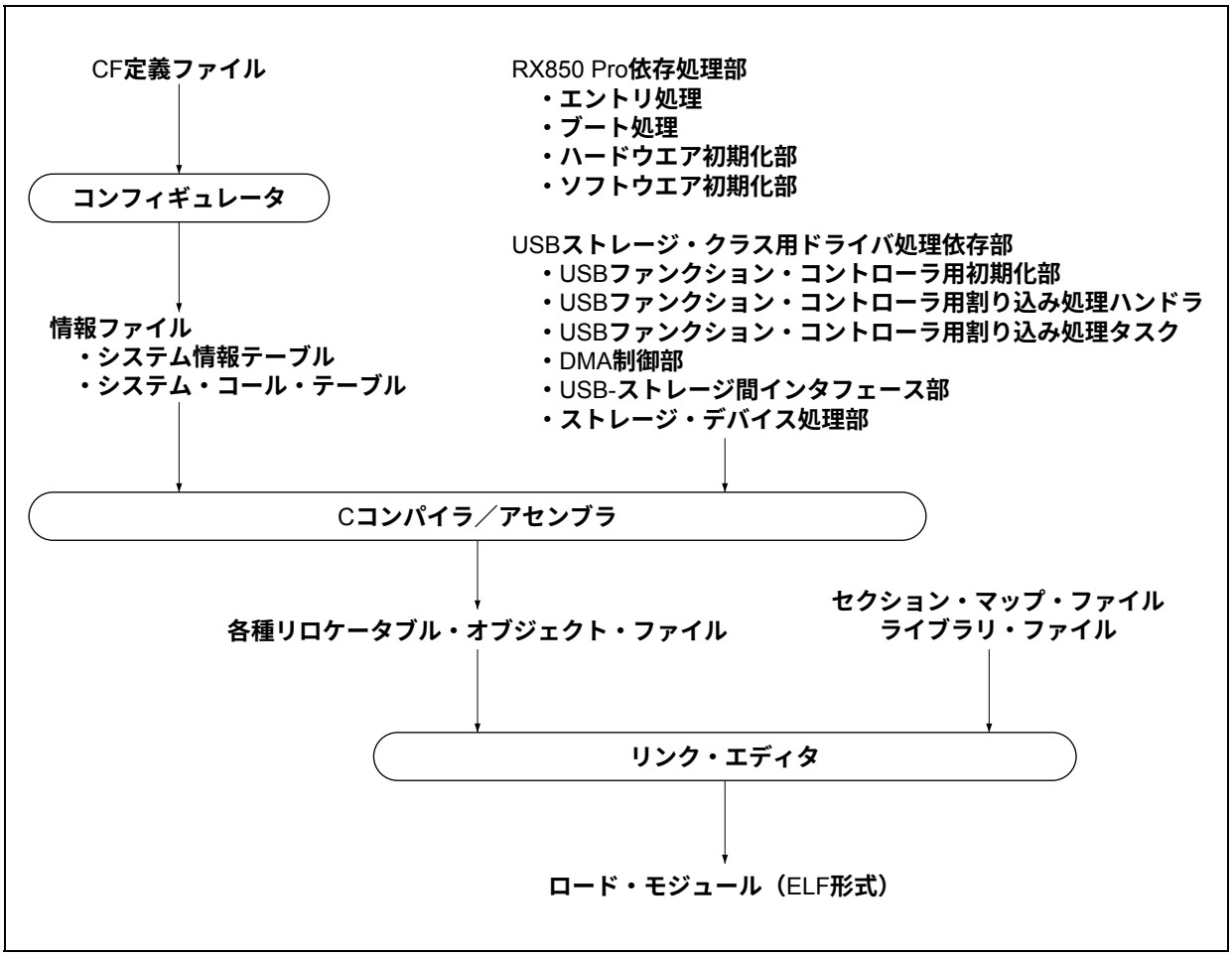
3. 3. 1　概　　要

システム構築とは，USBストレージ・クラス用ドライバ提供媒体からユーザの開発環境（ホスト・マシン）上にインストールしたファイル群を使用して，ロード・モジュールを生成することです。

USBストレージ・クラス用ドライバのシステム構築手順を次に示します。

① RX850 Pro依存処理部の記述
② ボード依存部の記述
③ USBストレージ・クラス用ドライバ処理依存部の記述
④ セクション・マップ・ファイルの記述
⑤ ロード・モジュールの生成

図3－4　システム構築手順



3. 3. 2　RX850 Pro依存処理部の記述

USBストレージ・クラス用ドライバが提供する一部の機能は，リアルタイムOS（RX850 Pro）の機能を利用しています。また，ユーザが記述した処理プログラムは，RX850 Proの管理下で実行することになります。

したがって，RX850 Proを正常に動作させるために，RX850 Pro依存処理部の記述が必要となります。

RX850 Pro依存処理部の一覧を次に示します。

○CF定義ファイル
○エントリ処理
○システム初期化処理
・ブート処理
・ハードウエア初期化部
・ソフトウエア初期化部

備考　RX850 Pro依存処理プログラムについての詳細は，3. 4　RX850 Pro**依存処理プログラム**を参照してください。

3. 3. 3　ボード依存部の記述

USBストレージ・クラス用ドライバのソース・プログラムは，ユーザの実行環境／アプリケーション・システムに依存した処理に関する初期化処理を，ボード依存部として切り出しています。

ボード依存部の一覧を次に示します。

・CPUボード依存部
USBストレージ・クラス用ドライバで必要となるI/Oポート入出力操作については，CPUボード依存部として切り出しています。

注意　ポートの設定は，ほかのレジスタ設定と同じように扱うため，専用の関数は用意していません。レジスタの定義は，RX850 Proの標準ヘッダ・ファイル¥nectools32¥inc850¥common¥SFR.hを参照してください。処理方法は，ブート処理部（boot.850）およびソフトウエア初期化部から呼ばれるポート設定用ソース・プログラム（port.c）を参照してください。

3. 3. 4　USBストレージ・クラス用ドライバ処理依存部の記述

このサンプル・プログラムでは，USBストレージ・クラス用ドライバ機能を実現するためのドライバ関数を，USBストレージ・クラス用ドライバ処理依存部として切り出しています。

USBストレージ・クラス用ドライバ処理依存部の一覧を次に示します。

- USBファンクション・コントローラ初期化部
- USBファンクション・コントローラ割り込みハンドラ
- USBファンクション・コントローラの割り込み処理タスク
- USBファンクション・コントローラ用汎用関数
- DMA制御部
- USB-ストレージ間インタフェース部
- ストレージ・デバイス処理部

備考　USBストレージ・クラス用ドライバ処理依存部の詳細は，3. 7　USB**ストレージ・クラス用ドライバの機能**を参照してください。

3. 3. 5　セクション・マップ・ファイルの記述

セクション・マップ・ファイルは，リンク・エディタが行うアドレス割り付けをユーザが固定化するためのファイルです。

RX850 Proを使用するとき，次の5つのテキスト領域が必須のセクションとなっています。

- 共通部分配置領域　：.systemセクション
- 割り込み処理関連配置領域　：.system_intセクション
- スケジューラ関連配置領域　：.system_cmnセクション
- システム情報領域　：.sitセクション
- インタフェース・ライブラリ／システム・コール配置領域　：.textセクション

備考　セクション・マップ・ファイルについての詳細は，3. 5　**セクション・マップ・ファイル**を参照してください。

3. 3. 6　ロード・モジュールの生成

コーディングを終了したRX850 Pro依存処理プログラム，USBストレージ・クラス用ドライバ処理依存部，セクション・マップ・ファイルに対して，Cコンパイラ，アセンブラ，リンカなどを実行することにより，ELF形式のロード・モジュールを生成します。

備考　ロード・モジュールの生成手順についての詳細は，3. 6　**ロード・モジュール**を参照してください。

3. 4　RX850 Pro依存処理プログラム

3. 4. 1　概　　要

USBストレージ・クラス用ドライバが提供する一部の機能は，リアルタイムOS（RX850 Pro）の機能を利用しています。また，ユーザが記述した処理プログラムは，RX850 Proの管理下で実行することになります。

したがって，RX850 Proを正常に動作させるために，RX850 Pro依存処理部の記述が必要となります。

RX850 Pro依存処理部の一覧を次に示します。

○CF定義ファイル

○エントリ処理

○システム初期化処理

・ブート処理

・ハードウエア初期化部

・ソフトウエア初期化部

3. 4. 2　CF定義ファイル

RX850 Proを使用したシステムを構築する場合，RX850 Proに提供する各種データを保持した情報ファイル（CF定義ファイル）が必要となります。

USBストレージ・クラス用ドライバ機能を実現するうえで必要となる情報を次に示します。

○リアルタイムOS情報
- ・RXシリーズ情報

○SIT情報
- ・システム情報
- ・システム最大値情報
- ・システム・メモリ情報
- ・タスク情報
- ・割り込みハンドラ情報
- ・初期化ハンドラ情報

○SCT情報
- ・タスク管理／タスク付属同期管理機能システム・コール情報
- ・割り込み処理管理機能システム・コール情報
- ・時間管理機能システム・コール情報

注意　このサンプル・プログラムでは，6個のタスク，3個の割り込みハンドラ，7種類のシステム・コールを使用して，各種機能を実現しています。このため，CF定義ファイルにおいては，システム最大値情報のタスクの最大生成数に“6個”を，割り込みハンドラの最大生成数に“3個”をUSBストレージ・クラス用ドライバのために確保するほか，タスク管理／タスク付属同期管理機能システム・コール情報に“sta_tsk, ext_tsk, slp_tsk, wup_tskシステム・コール”を，割り込み処理管理機能システム・コール情報に“loc_cpu, unl_cpuシステム・コール”を，時間管理機能システム・コール情報に“dly_tskシステム・コール”を使用するものとして，定義する必要があります。

備考　CF定義ファイルのコーディング方法についての詳細は，RX850 Pro　**ユーザーズ・マニュアル　インストレーション編**，およびサンプルのCF定義ファイル（sys.cf）を参照してください。

(1) 情報ファイルの生成手順

情報ファイル（システム情報テーブル，システム・コール・テーブル，システム情報ヘッダ・ファイル）の生成手順を，次に示します。

なお，情報ファイルの生成は，Windowsのコマンド・プロンプト上で行います。

注意　サンプルのビルド・ファイルを使用する場合，情報ファイルはビルド時に生成されます。このため，次の手順で生成する必要はありません。

① ディレクトリの移動

Windowsのcdコマンドを使用して，CF定義ファイルが格納されているディレクトリに移動します。なお，CF定義ファイルが格納されているディレクトリがC:¥sampleの場合の入力例を，次に示します。

[コマンド入力例]

```
C:>cd C:¥sample¥rx850<Enter>
```

② 情報ファイルの生成

コンフィギュレータcf850pro.exeを使用して，独自の記述形式で作成されたCF定義ファイルから情報ファイルを生成します。

なお，入力ファイル（CF定義ファイル名：sys.cf）から3つの情報ファイル（システム情報テーブル：sit.850，システム・コール・テーブル：svc.850，システム情報ヘッダ・ファイル：sys.h）を生成する場合の入力例を，次に示します。

[コマンド入力例]

```
C:>cf850pro –i sit.850 –c svc.850 –d sys.h sys.cf<Enter>
```

上記の操作により，CF定義ファイルから情報ファイルを生成できます。

注意　このサンプル・プログラムでは，情報ファイルを生成するためのサンプル・ファイル（CF定義ファイル）を提供しています。

備考　コンフィギュレータcf850pro.exeの起動オプションおよび実行方法についての詳細は，RX850 Pro　ユーザーズ・マニュアル　インストレーション編を参照してください。

3.4.3　エントリ処理

マスカブル割り込みが発生したときに，プロセッサが強制的に制御を移すハンドラ・アドレスに対して，割り込みハンドラへの分岐処理を割り付けています。

なお，RX850 Proの管理下で実行される割り込みハンドラ（CF定義ファイルの割り込みハンドラ情報で定義した割り込みハンドラ）に対応したハンドラ・アドレスに対しては，RX850 Proが提供するマクロRTOS_IntEntry_Indirect（RX850 Proが提供する割り込み処理管理機能への分岐処理）を割り付けてください。

備考　エントリ処理のコーディング方法についての詳細は，添付プログラムentry.850を参照してください。

3.4.4　システム初期化処理

システム初期化処理は，RX850 Proが正常に動作するうえで必要となるハードウエアの初期化処理（ブート処理，ハードウエア初期化部），およびソフトウエアの初期化処理（ニュークリアス初期化部，初期化ハンドラ）から構成されています。

したがって，システムが起動したとき，最初に行われる処理がシステム初期化処理となります。

注意　4種類のシステム初期化処理のうち，ニュークリアス初期化部については，RX850 Proが提供する機能の一部であるため，ユーザがその処理を記述する必要はありません。

ニュークリアス初期化部で行われる処理を次に示します。

○CF定義ファイルで定義されたシステム・メモリの確保
- System Pool 0
- User Pool 0

○CF定義ファイルで定義された管理オブジェクトの生成／初期化
- タスクの生成／起動
- 割り込みハンドラの登録

○初期タスクの起動

○アイドル・タスクの生成／起動

○ソフトウエア初期化部の呼び出し

○スケジューラに制御を移す

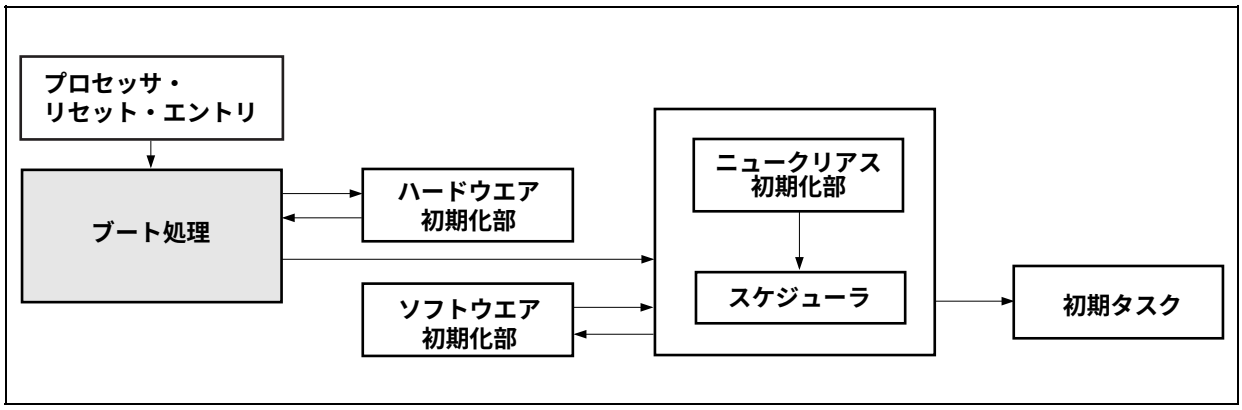
なお，アイドル・タスクとは，RX850 Proの管理下にある処理プログラム（タスク）がrun状態またはready状態でなくなったとき，すなわち，RX850 Proのスケジューリング対象となる処理プログラムがシステム内に1つも存在しなくなったときに，スケジューラから起動される処理ルーチンであり，HALT命令の発行を行っています。

(1) ブート処理

ブート処理は，プロセッサのリセット・エントリに割り付けられた関数であり，システム初期化処理の中でも最初に処理が実行されます。

ブート処理の位置付けを次に示します。

図3－5　ブート処理の位置付け



ブート処理で実行すべき処理内容を次に示します。

備考　ブート処理のコーディング方法についての詳細は，サンプルのboot.850を参照してください。

・tp, gp, epレジスタの設定

システム起動時，各種処理プログラム（ブート処理を含む）を実行するうえで必要となるテキスト・ポインタtp，グローバル・ポインタgp，スタック・ポインタepの値は不定です。そこで，ブート処理の最初の処理として，これらのレジスタの初期設定を行います。

注意　この章では，tpに“0”を，gpに“コンパイラが出力するグローバル・ポインタ・シンボル_gp”を，epに“コンパイラが出力するエレメント・ポインタ・シンボル_ep”を設定することを推奨します。

・ハードウエア初期化部の呼び出し

ターゲット・システム上のハードウエアを初期化するために用意された関数（ハードウエア初期化部）を呼び出します。

なお，ハードウエア初期化部で実行すべき処理内容（内部ユニットの初期化）をほかのところで行う場合は，“ハードウエア初期化部の呼び出し”は不要となります。

注意　この章では，ハードウエア初期化部で実行すべき処理内容（内部ユニットの初期化）をソフトウエア初期化部で行うため，“ハードウエア初期化部の呼び出し”は不要です。詳細は，RX850 Pro　ユーザーズ・マニュアル　インストレーション編を参照してください。

・ニュークリアス初期化部への制御の移行

ニュークリアス初期化部では，システム情報テーブルに記述された情報をもとに，システム・メモリ（System Pool 0, User Pool 0）の確保，および管理オブジェクトの生成／初期化などを行っています。そこで，ニュークリアス初期化部に制御を移す前には，r10レジスタにシステム情報テーブルの先頭アドレス_sitを設定しておく必要があります。

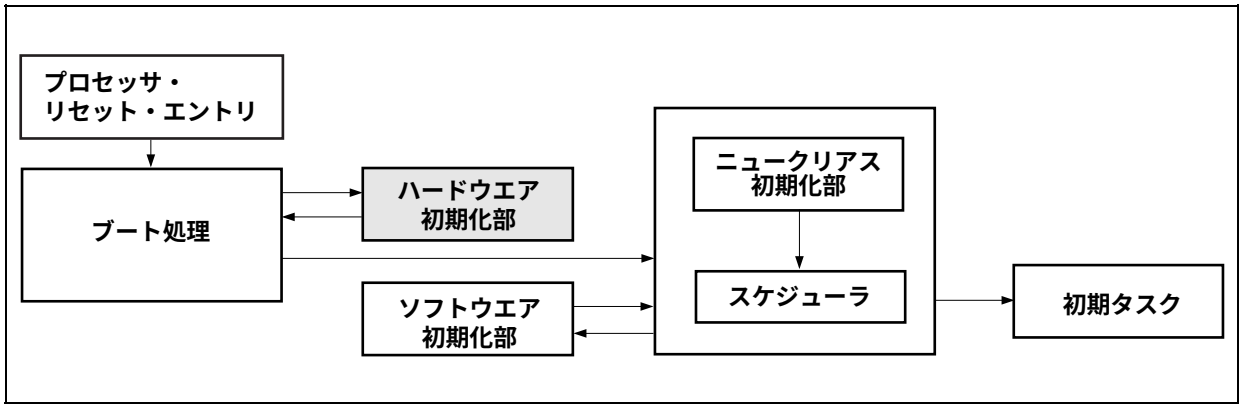
注意　システム情報テーブルとは，独自の記述形式で作成されたCF定義ファイルを，RX850 Proが提供するユーティリティ・ツール（コンフィギュレータcf850pro.exe）を使用して，アセンブリ言語形式に変換したものです。

(2) ハードウエア初期化部

ハードウエア初期化部は，ターゲット・システム上のハードウエアを初期化するために用意された関数であり，ブート処理から呼び出されます。

ハードウエア初期化部の位置付けを次に示します。

図3－6　ハードウエア初期化部の位置付け



ハードウエア初期化部で実行すべき処理内容を次に示します。

注意1. マスカブル割り込みは，初期化時ディフォールトでマスクされていますので，特に禁止設定をする必要はありません。

2. サンプル・プログラムでは，ハードウエアの初期化をソフトウエア初期化部で行っています。ハードウエア初期化部の詳細は，RX850 Pro　ユーザーズ・マニュアル　インストレーション編を参照してください。

・ブート処理に制御を戻す

ブート処理の呼び出し関数であるハードウエア初期化部からブート処理に制御を戻す場合，ブート処理におけるハードウエア初期化部の呼び出し時，lpレジスタに対する戻りアドレスの設定が行われているため，“ return( ); ”命令を発行することにより実現されます。

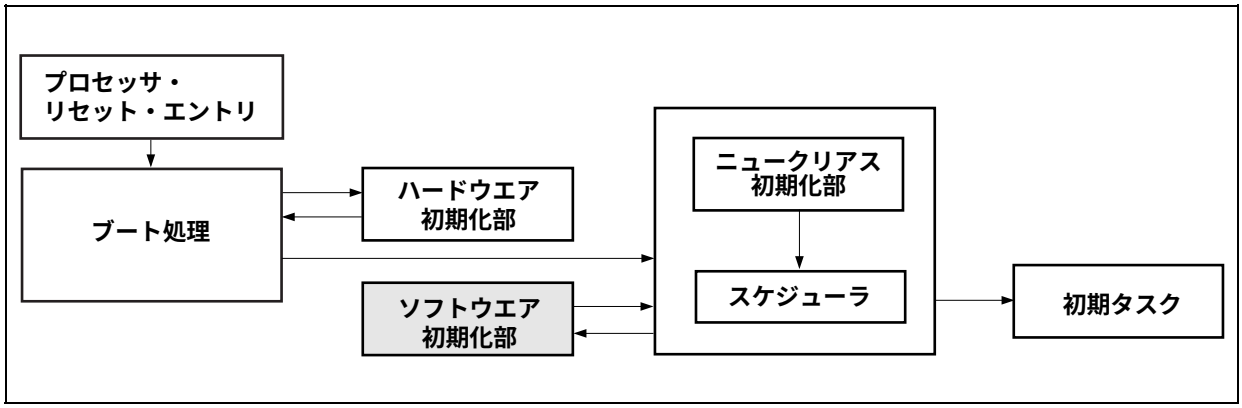
なお，ハードウエア初期化部をアセンブリ言語で記述した場合は，“ jmp [lp] ”命令を発行することにより実現されます。

(3) ソフトウエア初期化部

初期化ハンドラは，ユーザのソフトウエア環境を快適なものとするために用意された関数であり，ニュークリアス初期化部から呼び出されます。

ソフトウエア初期化部の位置付けを次に示します。

図3－7　ソフトウエア初期化部の位置付け



ソフトウエア初期化部で実行すべき処理内容を次に示します。

備考　ソフトウエア初期化部のコーディング方法についての詳細は，サンプルのvarfunc.cを参照してください。

・内部ユニット（リアルタイム・パルス・ユニット（RPU））の初期化

RX850 Proでは，一定周期で発生するタイマ割り込みを利用してタイマ・オペレーション機能（タスクの遅延起床，周期起動ハンドラの起動，タイムアウトなど）を実現しています。

このため，RX850 Proが処理を開始する前にリアルタイム・パルス・ユニットを初期化しておく必要があります。

なお，リアルタイム・パルス・ユニットのコンペア・レジスタCMD0には，CF定義ファイルのシステム情報で定義した基本クロック周期でタイマ割り込みが発生するような値を設定する必要があります。

・タイマ割り込みの受け付け許可

タイマ割り込みの受け付けを許可します。これにより，ニュークリアス初期化部の処理が終了したとき，RX850 Proが提供するタイマ・オペレーション機能（タスクの遅延起床，タイムアウト，周期起動ハンドラの起動など）の利用が可能となります。

・ニュークリアス初期化部に制御を戻す

ニュークリアス初期化部の呼び出し関数である初期化ハンドラからニュークリアス初期化部に制御を戻す場合，ニュークリアス初期化部における初期化ハンドラの呼び出し時に，lpレジスタに対する戻りアドレスの設定が行われているため，“ return( ); ”命令を発行することにより実現されます。

なお，初期化ハンドラをアセンブリ言語で記述した場合は，“ jmp [lp] ”命令を発行することにより実現されます。

3.4.5　時間管理機能

RX850 Proにおける時間管理は，ハードウエア（クロック・コントローラなど）により一定周期で発生するクロック割り込みを利用しています。

クロック割り込みが発生すると，RX850 Proのシステム・クロック処理が呼び出され，システム・クロックの更新やタスクの遅延起床，周期ハンドラの起動などの，時間に関連した処理が行われます。

システム・クロックは，RX850 Proが時間管理のときに使用する時刻（48ビット幅，単位：ms）を保持したソフトウエア・タイマです。

システム・クロックは，システム初期化処理で“0H”に設定されたあと，システム・クロック処理で基本クロック周期（コンフィギュレーション時に指定）を単位として更新されます。

注意　RX850 Proが管理するシステム・クロックは，48ビット幅で構成されています。このため，RX850 Proでは，オーバフローした数値（48ビットでは表せない数値）は無視されます。RX850 Proの時間管理機能の詳細は，RX850 Pro　ユーザーズ・マニュアル　基礎編を参照してください。

3. 5　セクション・マップ・ファイル

3. 5. 1　概　　要

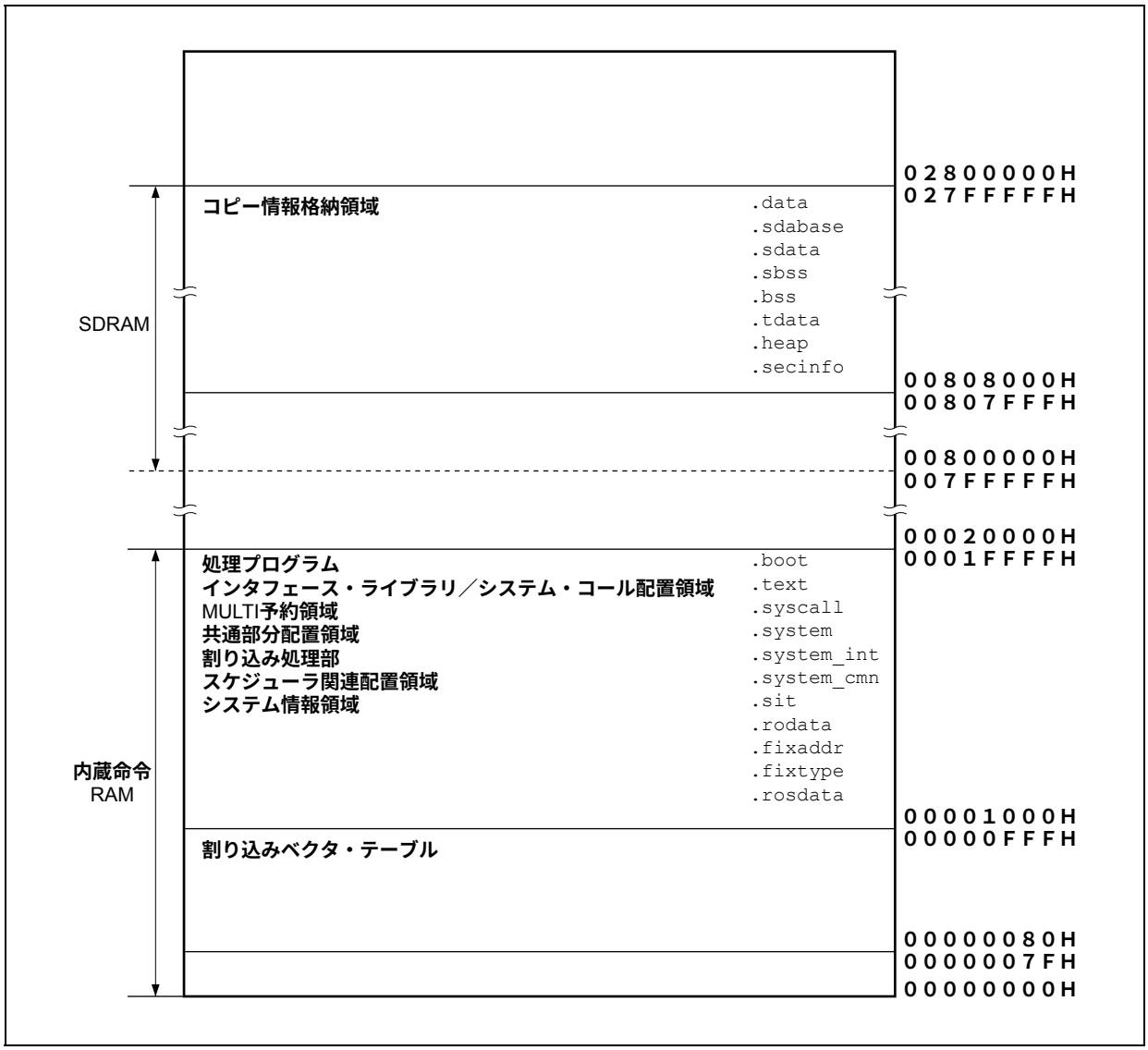
セクション・マップ・ファイルとは，リンク・エディタが行うアドレス割り付けを，ユーザが固定化するためのファイルです。

3. 5. 2　RX850 Proのアドレス割り付け，3. 5. 3　その他のアドレス割り付けに，ユーザの処理プログラム（.data, .bssセクションなど）以外に必要となるアドレス割付けについて示します。

サンプルのcommon.lxで行われているアドレス割り付けを次に示します。

備考　セクション・マップ・ファイルのコーディング方法についての詳細は，サンプルのcommon.lxを参照してください。

図3－8　アドレス割り付け例



3. 5. 2　RX850 Proのアドレス割り付け

RX850 Proは，5つのテキスト領域（共通部分配置領域，割り込み処理関連配置領域，スケジューラ関連配置領域，システム情報領域，インタフェース・ライブラリ／システム・コール配置領域）から構成されています。これにより，大きなサイズが要求されるメモリ領域については外部RAMに，高速なアクセスが要求されるメモリ領域（割り込み処理部，スケジューリング処理部）については内蔵命令RAM（00000000H-0001FFFFH）に割り付けるといったことが可能となります。

注意　サンプル・プログラムでは，5つのテキスト領域のすべてを内蔵命令RAMに配置しています。

・共通部分配置領域（.systemセクション）

RX850 Proの本体処理（タスク管理機能，タスク付属同期機能など）が割り付けられる領域です。

・割り込み処理関連配置領域（.system_intセクション）

RX850 Proが提供する割り込み処理管理機能のうち，割り込みハンドラに制御を移すときに行われる割り込み前処理，およびマスカブル割り込みが発生した処理プログラムに制御を戻すときに行われる割り込み後処理から構成されています。

したがって，割り込み処理部を内蔵命令RAMに割り付けることにより，割り込みハンドラに対する応答性能が向上します。

注意　この章では，割り込み処理部を内蔵命令RAMに割り付けることを推奨します。

・スケジューラ関連配置領域（.system_cmnセクション）

RX850 Proが提供するスケジューリング機能のうち，タスクの起床処理およびタスクのスケジューリング処理から構成されています。

したがって，スケジューリング処理部を内蔵命令RAMに割り付けることにより，タスクの起床処理およびタスクのスケジューリング処理が高速化されるほか，スケジューリング処理を伴ったシステム・コールの処理も高速化されます。

注意　この章では，スケジューリング処理部を内蔵命令RAMに割り付けることを推奨します。

・システム情報領域（.sitセクション）

CF定義ファイルに対してコンフィギュレータcf850.exeを実行することにより生成されたシステム情報テーブルが割り付けられる領域です。

なお，システム情報テーブルは，ニュークリアス初期化部（システム・メモリの確保，管理オブジェクトの生成／初期化）を実行するうえで必要となる各種データから構成されています。

・インタフェース・ライブラリ／システム・コール配置領域（.textセクション）

システム・コールを含む命令が割り付けられる領域です。

・システム・メモリ

RX850 Proが提供する機能を実現するうえで必要となる各種管理ブロック（タスク管理ブロック，セマフォ管理ブロックなど），割り込みハンドラ用スタック，タスク用スタックが割り付けられる領域（System Pool 0），および処理プログラムからダイナミックなメモリ操作（メモリ・ブロックの獲得／開放）を可能とした領域（User Pool 0）から構成されています。

注意1. CF定義ファイル作成時，“システム・メモリの先頭アドレス”を指定する必要があります。したがって，セクション・マップ・ファイルにシステム・メモリの定義を行うときには，必ずアドレスを指定してください。

2. システム・メモリのセクション名については，ユーザが自由に指定できます。

3. 5. 3　その他のアドレス割り付け

次に，RX850 Pro以外にアドレス割り付けが必要となるセクションについて示します。

・MULTI予約領域（.syscallセクション）

ディバッガMULTI（Green Hills Software, Inc.製）がワーク・エリアとして使用する領域です。

注意1. MULTI使用の有無にかかわらず，.syscallセクションの定義を行う必要があります。

2. .syscallセクションの定義を行うときには，必ず4バイト・アライン指定を行ってください。

・コピー情報格納領域（.secinfoセクション）

セクション・マップ・ファイルで，ROM識別子の指定が行われているセクションのプログラム（データ，テキスト）をROMからRAMに転送するときに必要となる情報（先頭アドレス，サイズ）を，リンク・エディタが出力するための領域です。

なお，ROM識別子の指定は，処理プログラムをROM化するときに必要となるものです。したがって，ROM化を行わない場合には，.secinfoセクションの定義は不要となります。

注意　このサンプル・プログラムでは，ROM識別子の指定を行っていないため，空のセクションとなります。

3. 6　ロード・モジュール

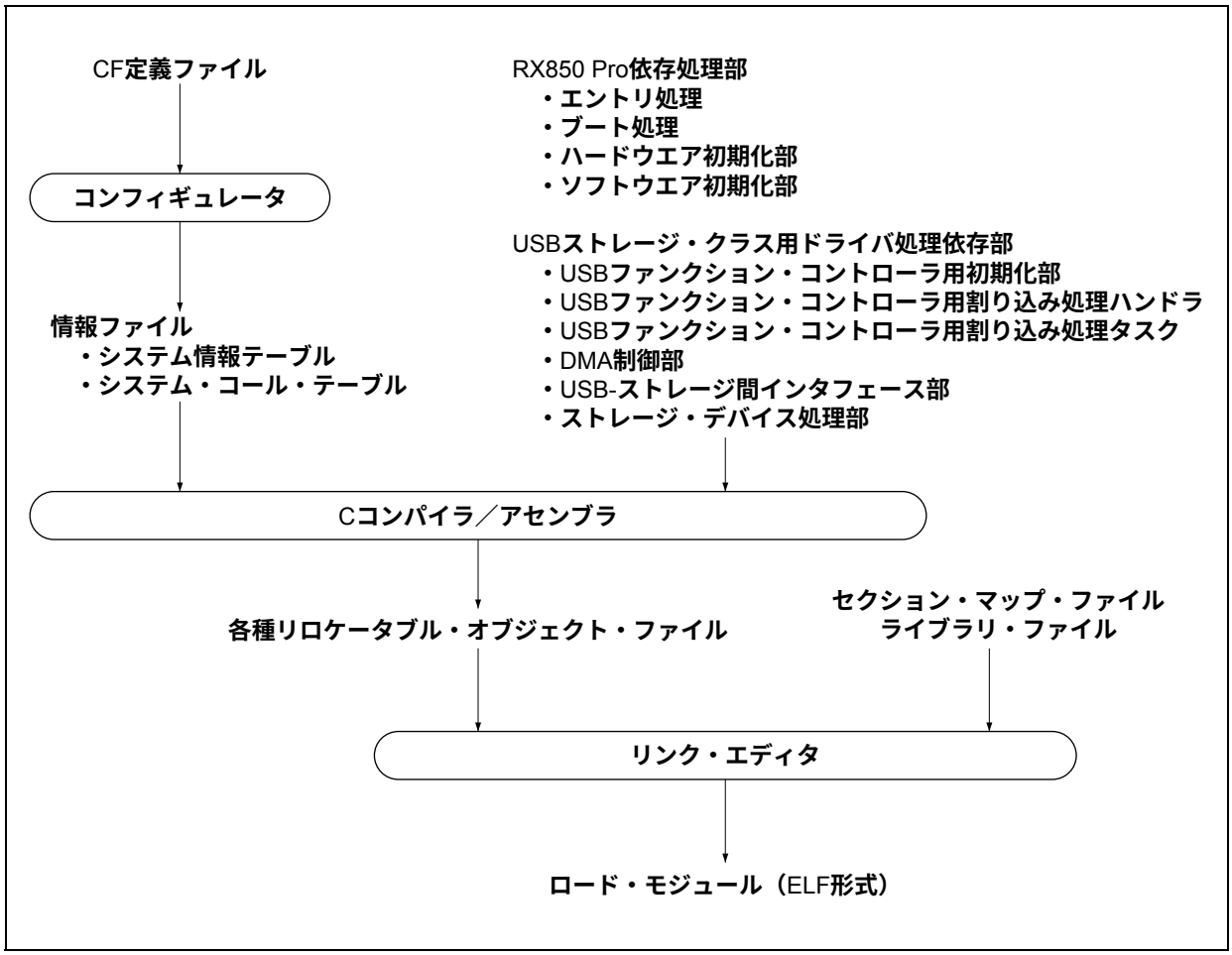
3. 6. 1　概　　要

コーディングを終了したRX850 Pro依存処理プログラム，USBストレージ・クラス用ドライバ処理依存部，セクション・マップ・ファイルに対して，Cコンパイラ，アセンブラ，リンカなどを実行することにより，ELF形式のロード・モジュールを生成します。

ロード・モジュールの生成手順を次に示します。

注意　サンプルの.bldファイルを実行することにより，サンプル・プログラムに対応したロード・モジュールを生成できます。
ただし，.bldファイルのパスの定義については，ユーザの開発環境にあわせた修正が必要です。

図3－9　ロード・モジュールの生成手順



3. 6. 2　ロード・モジュールの生成

コーディングを終了したRX850 Pro依存処理プログラム，USBストレージ・クラス用ドライバ処理依存部，セクション・マップ・ファイルは，次の手順によりELF形式のロード・モジュールとなります。

① システム情報テーブル，システム・コール・テーブルの生成

独自の記述形式で作成されたCF定義ファイルは，ロード・モジュール作成時，リンク・エディタが行うリンク処理の対象外のファイル形式です。

そこで，RX850 Proが提供するユーティリティ・ツール（コンフィギュレータcf850.exe）を使用して，アセンブル可能なファイル（システム情報テーブル，システム・コール・テーブル）を生成します。

備考　システム情報テーブル，システム・コール・テーブルの生成方法についての詳細は，3. 4. 2（1）**情報ファイルの生成手順**を参照してください。

② オブジェクト・ファイルの生成

次に示す各種処理プログラム（C言語／アセンブリ言語形式のファイル）に対して，Cコンパイラ／アセンブラを実行することにより，リロケータブル・オブジェクト・ファイルを生成します。

○RX850 Pro依存処理プログラム
- システム情報テーブル
- システム・コール・テーブル
- エントリ処理
- ブート処理
- ハードウエア初期化部
- 初期化ハンドラ

○USBストレージ・クラス用ドライバ処理依存部

③ ロード・モジュールの生成

②で生成したリロケータブル・オブジェクト・ファイル，および，各種ライブラリ・ファイル，セクション・マップ・ファイルに対して，リンク・エディタを実行することにより，ELF形式のロード・モジュールを生成します。

libansi.a	ANSI Cライブラリ
libind.a	Green Hills software, Inc.製Cライブラリ（ターゲットCPUに依存しないルーチン群）
libarch.a	Green Hills software, Inc.製Cライブラリ（ターゲットCPUに依存するルーチン群）
libsys.a	Green Hills software, Inc.製Cライブラリ（システム・コール，初期化ルーチンなど）

rxcore.o	ニュークリアス共通部分オブジェクト
librxp.a	ニュークリアス・ライブラリ
libchp.a	インタフェース・ライブラリ

なお，rxcore.o, librxp.a, libchp.aは“ RX850 Pro ”から，libansi.a, libind.a, libarch.a, libsys.aは“ CCV850（Green Hills Software, Inc.製） ” から提供されています。

3. 7　USBストレージ・クラス用ドライバの機能

3. 7. 1　概　　要

USBストレージ・クラス用ドライバでは，USBストレージ・クラス用ドライバ処理を実現するためのタスク，割り込みハンドラのほかに，USBファンクション・コントローラの初期化処理の記述が必要となります。

USBストレージ・クラス用ドライバ処理依存部の一覧を次に示します。

・USBファンクション・コントローラの初期化処理

RX850 Proのソフトウエア初期化部から呼び出され，USBファンクション・コントローラの初期化処理を行います。

・USBファンクション・コントローラの割り込みハンドラ

USBファンクション・コントローラの割り込み発生ごとに呼び出される割り込み処理専用ルーチンであり，CF定義ファイルに定義されています。

注意　このサンプル・プログラムでは，必要な割り込み以外はマスクされています。
このサンプル・プログラムで使用する割り込みは，次の4つです。

- INTUSB0B信号で通知されるSHORT割り込み
（DMAモードでFIFOがフルでない場合に，UF0BO1, UF0BO2レジスタのどちらかのFIFOからデータが読み出され，USBSPnB信号（n = 2, 4）をアクティブにしたことを示します）
- INTUSB0B信号で通知されるDMAED割り込み
（Endpoint n用（n = 1-4, 7, 8）DMA終了信号がアクティブになったことを示します）
- INTUSB0B信号で通知されるCPUDEC割り込み
（UF0E0STレジスタにFWでデコードを行うリクエストがあることを示します）
- INTUSB1B信号で通知されるBKO1DT割り込み
（UF0BO1レジスタにデータが正常受信されたことを示します）

・USBファンクション・コントローラの割り込み処理タスク

USBファンクション・コントローラの割り込みハンドラから呼び出され，割り込み要因ごとの処理（レジスタ設定，データの送信／受信処理など）を行います。

・USBファンクション・コントローラ用汎用関数

USBストレージ・クラス用ドライバで使用される汎用関数として，エンドポイントごとのSTALL応答の設定や，送信，受信の処理を行う関数が用意されています。

備考　USBストレージ・クラス用ドライバ処理依存部のコーディング方法についての詳細は，サンプルのusbf850.cを参照してください。

・DMA制御部

DMAの初期化およびDMAの起動処理を行います。

サンプル・プログラムでは，バルク・エンドポイントのデータがMaxPacketサイズ（40バイト）を越える場合，DMA転送が使用されます。

備考　DMA制御部のコーディング方法についての詳細は，サンプルのusbf850_dma.cを参照してください。

・Bulk-Only Transport処理部

USBストレージ・クラス用のデバイス・クラス固有のリクエスト処理，CBW処理，およびCSWの送信処理を行います。

注意　このサンプル・プログラムで受け付けるデバイス・クラス固有のリクエストは，次の2つです。各リクエストの詳細は，Universal Serial Bus Mass Starage Class Bulk-Only Transport Revision 1.0を参照してください。

- Bulk-Only Mass Storage Resetリクエスト
- Get Max LUNリクエスト

備考　Bulk-Only Transport処理部のコーディング方法についての詳細は，サンプルのusbf850_storage.cを参照してください。

・ストレージ・デバイス処理部

ストレージ・デバイスの初期化，SCSIコマンドの処理を行います。

注意1.　このサンプル・プログラムは，Mass Storageデバイス（インタフェース・クラス：マス・ストレージ，インタフェース・サブクラス：SCSI，インタフェース・プロトコル：Bulk-Only Transportプロトコル）として動作します。今回，使用するストレージ・デバイスは，結合される論理ユニットはなく，メモリ領域を確保し疑似的にリムーバブル・ディスクが繋がれているように動作するものです（ブロック・サイズ：512バイト，論理ブロック数：192，容量：96 Kバイト）。仮想デバイス用のメモリ領域は，storage_dataという名前の配列で確保されています。詳細は，サンプルのata.hを参照してください。

2.　仮想デバイスを変更し実際のデバイスを制御する場合は，サンプル・プログラム内で使用するデータやデータ処理について変更が必要です。環境にあわせて変更してください。

備考　ストレージ・デバイス処理部のコーディング方法についての詳細は，サンプルのata_ctrl.cおよびscsi_cmd.cを参照してください。

・USBのサスペンド／レジュームの処理

USBのサスペンド／レジュームの処理はシステムに依存するため，このサンプル・プログラムではサポートしていません。システム上，処理が必要な場合は，次のことに注意して処理を追加してください。

V850E/ME2に内蔵しているUSBファンクション・コントローラでは，サスペンド／レジュームが割り込み（INTUSB0B信号）により通知されます。このため，割り込みハンドラ（INTUSB0B信号用）内でUF0IS0.RSUSPDビットを確認し，セット（1）されていれば，UF0EPS1.RSUMビットを確認してサスペンド状態なのかレジューム状態なのかを判断できます。

処理を追加する一例として，割り込みハンドラ（INTUSB0B信号用）に上記の状態判断のコードを追加し，そこから必要な処理を行うタスクを起床するようにすることで実現できます。

3. 7. 2　処理の流れ

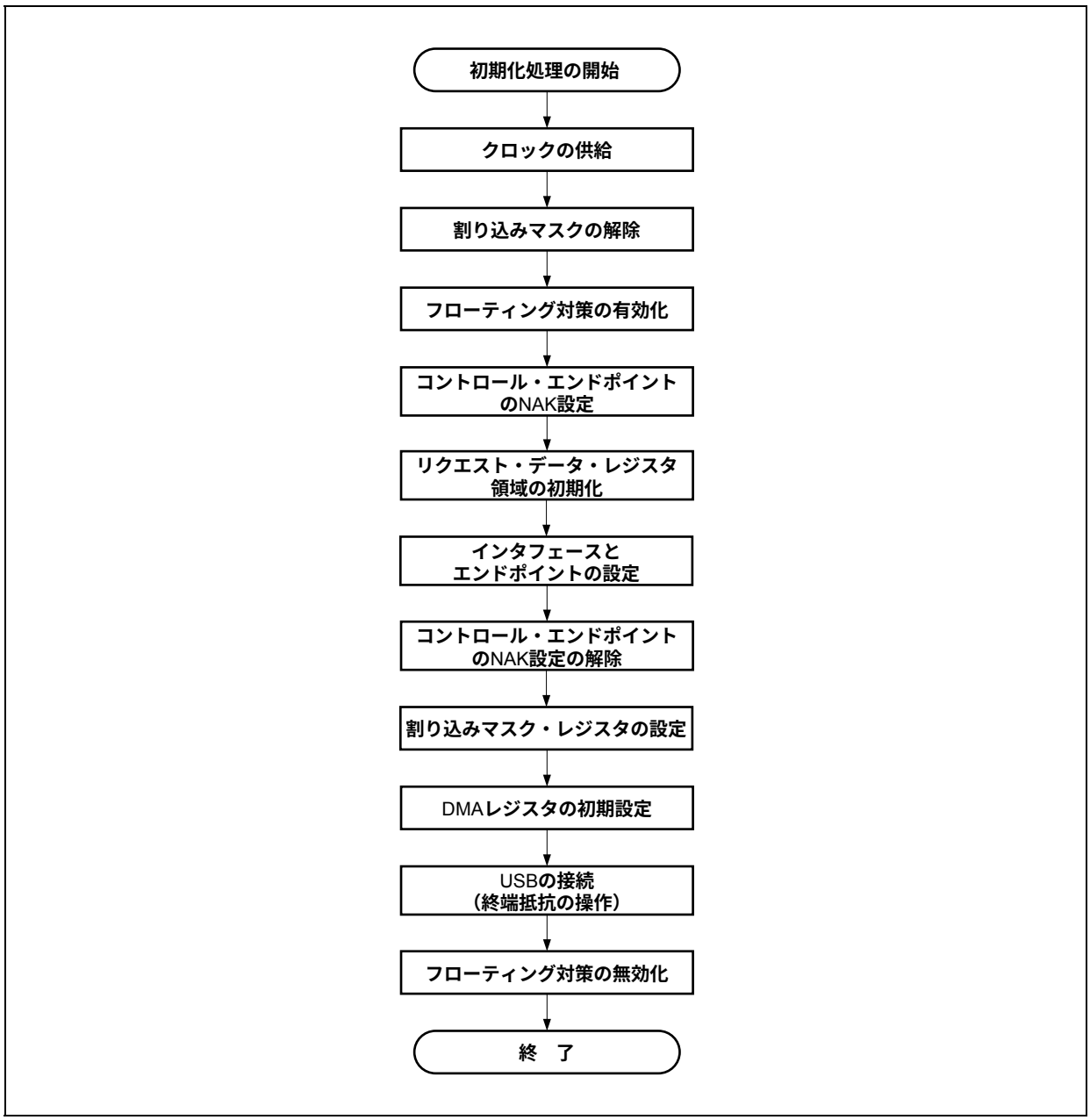
サンプル・プログラムの初期化処理，割り込み処理，CBWデータの処理についての処理の流れを，次に示します。

(1) 初期化処理

USBデバイスの初期化は，ソフトウエア初期化部から呼び出され実行されます。

サンプル・プログラムにおけるUSBデバイス初期化処理（電源投入時）の流れを，次に示します。

図3－10　初期化処理のフロー・チャート



初期化処理で実行すべき処理内容を次に示します。

注意　ポートの処理以外では，初期化処理が必要です。ターゲット・ボードが違う場合，端子の割り付けが違うことがありますので，ターゲット・ボードの仕様にあわせて読み替えてください。

・クロックの供給

USBファンクション・コントローラのレジスタを設定する前に，必ずUCKC.UCKCNTビットをセット（1）する必要があります。セット（1）することにより，USBへのクロック供給を許可します。なお，P10端子をクロック入力として使用するので，P10端子を入出力ポート・モードの入力モードに設定し，クロックの入力を許可します。

・割り込みマスクの解除

割り込み制御レジスタでUSB関連の割り込み信号のマスクを解除します。

・フローティング対策の有効化

UF0BC.UBFIORビットをクリア（0）し，ケーブル未接続時の不定値によるBus Resetなどの誤認識を防止します。

・コントロール・エンドポイントのNAK設定

自動実行リクエストを含むすべてのリクエストにNAK応答します。
自動実行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実行リクエストに意図しないデータを返さないように設定します。

・リクエスト・データ・レジスタ領域の初期化

Get Descriptorリクエストに応答するためのディスクリプタ・データなどをレジスタに登録します。登録するデータは，デバイス・ステータス，エンドポイント0ステータス，Device Descriptor, Configuration Descriptor, Interface Descriptor, Endpoint Descriptorです。

注意　クラスによっては，そのクラス用のディスクリプタの登録が必要な場合があります。USBストレージ・クラスでは，USBの標準ディスクリプタ以外は使用しません。

・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，Alternative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，レジスタに設定します。

・コントロール・エンドポイントのNAK設定の解除

自動実行リクエスト用のデータ登録が終わったところで，コントロール・エンドポイント（エンドポイント番号0）のNAK設定を解除します。

・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ステータス・レジスタに示される割り込み要因ごとのマスクを設定します。

・DMAレジスタの初期設定

DMAを使用するエンドポイントごとに，DMAの初期化処理を呼び出し，初期化を行います。

・USBの接続（終端抵抗の操作）

D+信号をプルアップします。

・フローティング対策の無効化

UF0BC.UBFIORビットをセット（1）し，フローティング対策を無効化します。

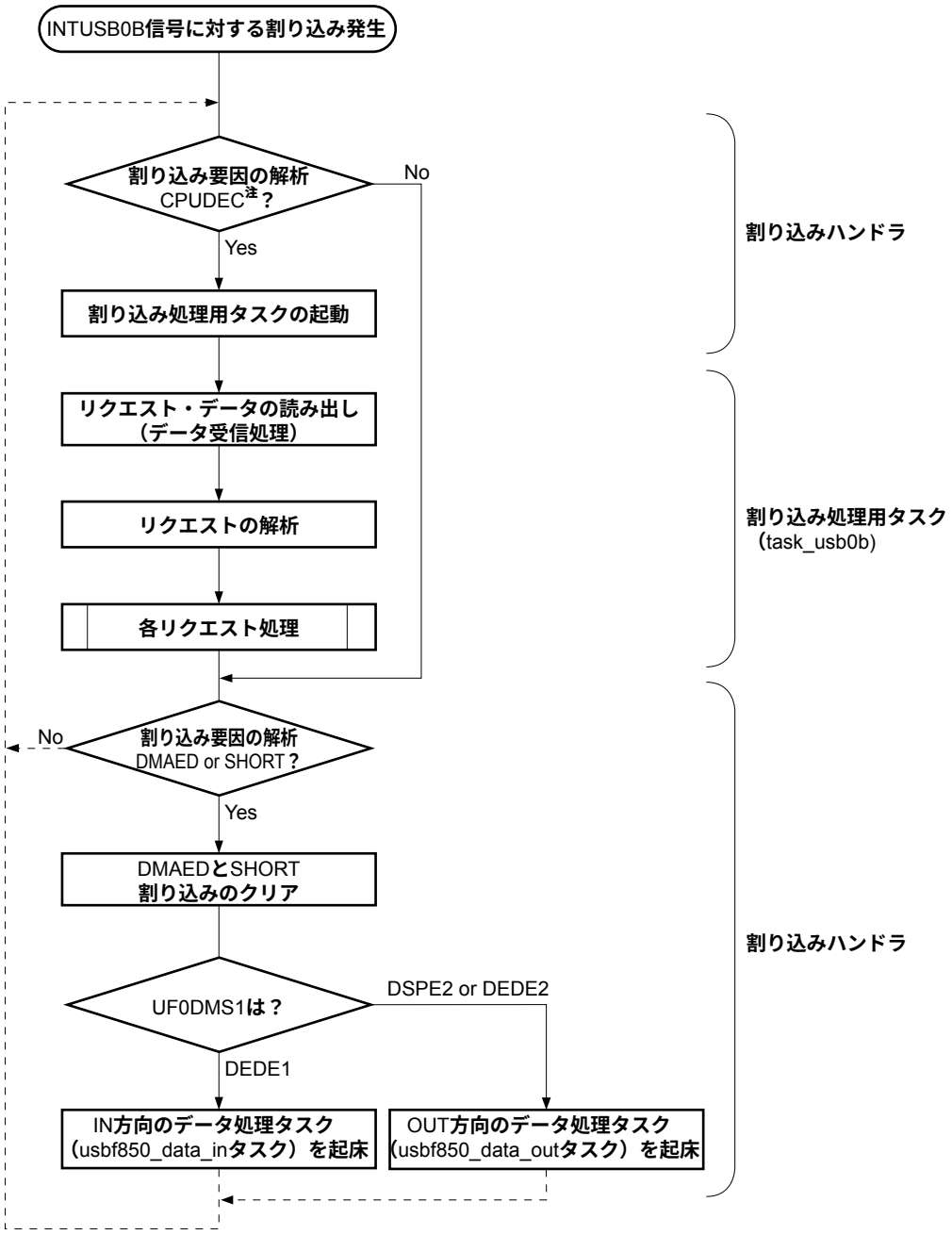
(2) 割り込み処理

サンプル・プログラムでは，初期化完了後，割り込みイベントによって動作します。イベントがない場合は，常にアイドル状態です。また，ストレージ・デバイスからイベントを起こすことはなく，すべてホスト・ドライバからのイベントにより動作します。

サンプル・プログラムでの割り込み処理の流れを，図3－11，図3－12に示します。

注意　図3－11に示すフロー・チャートは，USBファンクション・コントローラのINTUSB0B信号により通知される割り込み処理の流れを示しています。
図3－12に示すフロー・チャートは，USBファンクション・コントローラのINTUSB1B信号により通知される割り込み処理の流れを示しています。

図3－11　割り込み処理のフロー・チャート（1）



注　USBストレージ・クラス用ドライバのINTUSB0B信号では，CPUDEC, DMAED, SHORT割り込みを受け付けます。

INTUSB0B信号による割り込みでのサンプル・プログラムの処理内容を，次に示します。

[割り込みハンドラ内での処理]

・割り込み要因の解析

サンプル・プログラムでは，実行される割り込みハンドラにより，解析する割り込みステータスが違います。

INTUSB0B信号で通知される割り込みについては，CPUDEC, DMAED, SHORT割り込みに対応しています。これらの割り込みが発生すると，INTUSB0B信号による割り込みハンドラが起動します。 この割り込みハンドラの中で，UF0IS1レジスタを読み，割り込み要因がCPUDEC割り込みかどうかを確認します。さらに，UF0IS0レジスタを読み，DMAED, SHORT割り込みかどうかを確認します。

注意　このサンプル・プログラムでは，使用する割り込みハンドラをCF定義ファイルであらかじめ登録してあります。

・割り込み処理用のタスクの起動

割り込み要因がCPUDECだった場合，task_usb0bタスクを起動します。

注意　このサンプル・プログラムでは，起動するタスクをCF定義ファイルであらかじめ登録してあります。

・usbf850_data_inタスク，usbf850_data_outタスクの起床

usbf850_no_dataタスク，usbf850_data_inタスク，およびusbf850_data_outタスクは，SCSIコマンド処理を行うタスクです。これらのタスクは，BKO1DT割り込みで正常なCBWを受信することで起動されます。このうち，usbf850_data_inタスクおよびusbf850_data_outタスクは，DMA起動後スリープします。INTUSB0B信号用の割り込みハンドラでは，割り込み要因がDMAEDまたはSHORTだった場合，UF0DMS1レジスタを読みだし，DEDE1, DSPE2, DEDE2であるかどうかを確認します。

DEDE1だった場合はusbf850_data_inタスクを，DSPE2またはDEDE2だった場合はusbf850_data_outタスクを起床させます。

[task_usb0bタスクでの処理]

・リクエスト・データの読み出し

UF0E0STレジスタからSETUPデータを読み出します。

・リクエストの解析

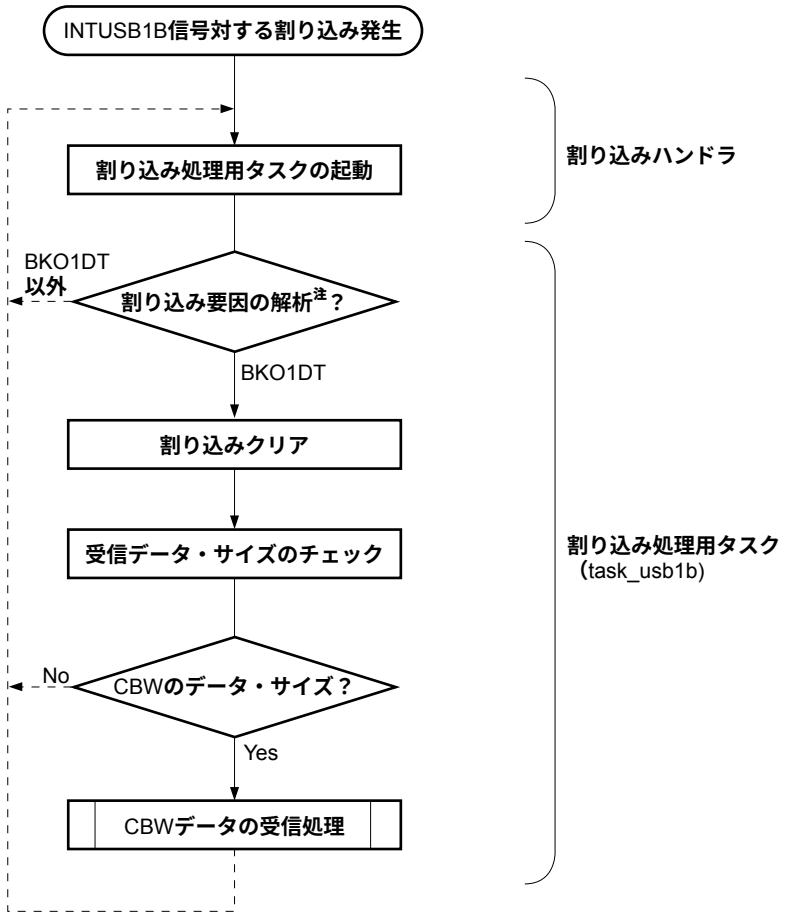
読み出したSETUPデータを解析し，リクエスト内容を確認します。

・各リクエスト処理

解析したリクエスト内容に対応した処理を行います。

サンプル・プログラムでは，標準デバイス・リクエストのGet Descriptor（String Descriptor）リクエスト，およびデバイス・クラス固有のリクエストの処理を行います。

図3－12　割り込み処理のフロー・チャート（2）



注　USBストレージ・クラス用ドライバのINTUSB1B信号では，BKO1DT割り込みだけを受け付けます。

INTUSB1B信号による割り込みでのサンプル・プログラムの処理内容を，次に示します。

・割り込み処理用のタスクの起動

割り込み要因を確認せず，task_usb1bタスクを起動します。

注意　このサンプル・プログラムでは，起動するタスクをCF定義ファイルであらかじめ登録してあります。

・割り込み要因の解析

INTUSB1B信号で通知される割り込みについては，BKO1DT割り込みだけに対応しています。割り込みが発生すると，INTUSB1B信号による割り込みハンドラが起動します。

割り込みハンドラ内では割り込み要因を確認せず，起動されたタスクで割り込み要因がBKO1DTであるかを確認します。

注意　このサンプル・プログラムでは，使用する割り込みハンドラをCF定義ファイルであらかじめ登録してあります。

・受信データ・サイズのチェック

割り込み要因がBKO1DTであれば，UF0BO1Lレジスタを読み出し，受信データ長がCBWのデータ長と等しいかを確認します。CBWのデータ長と等しければ，usbf850_rx_cbw関数を呼び出し，CBWの処理を開始します。

備考　CBWの処理については，（3）CBW**データの処理**を参照してください。

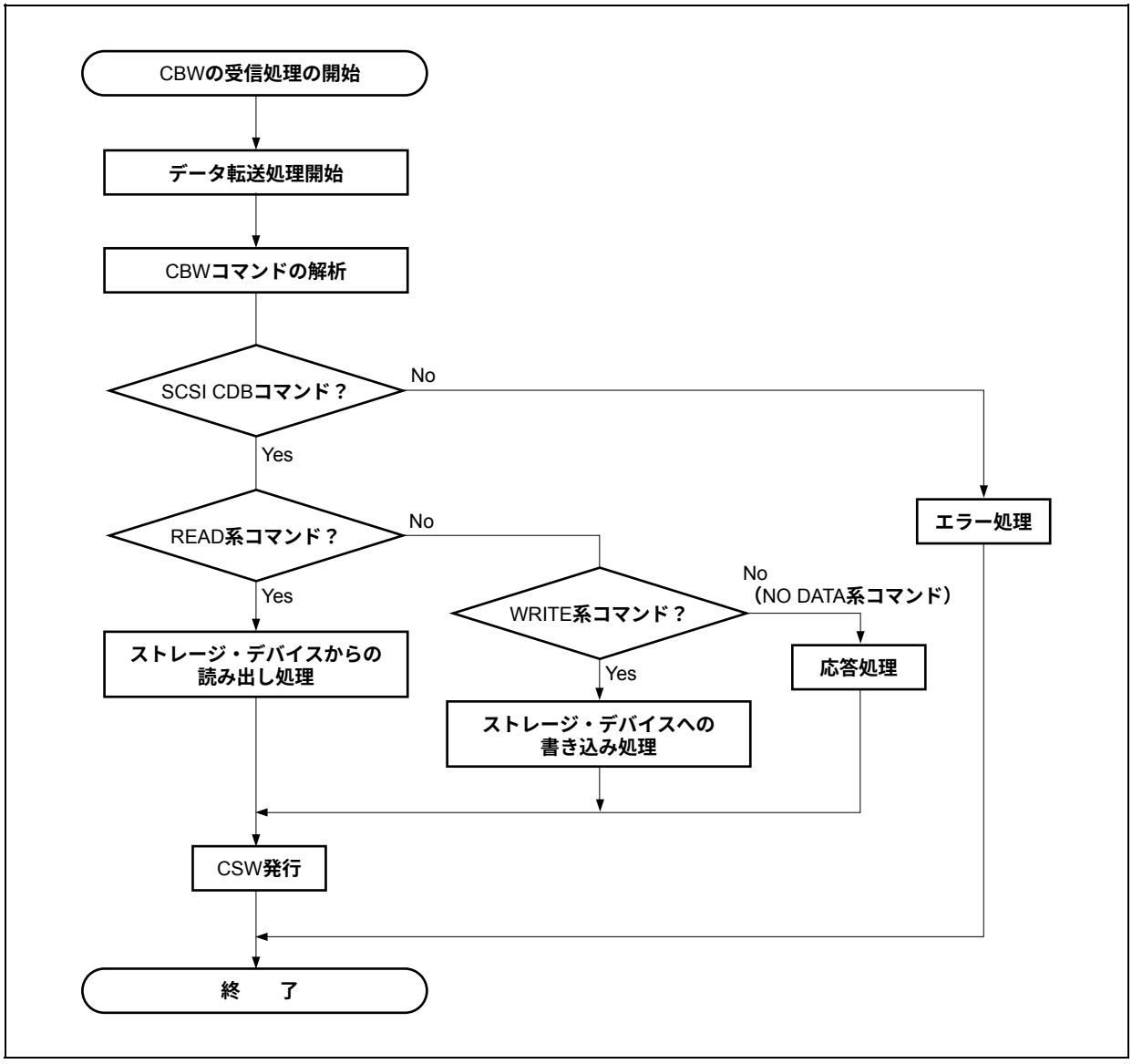
(3) CBWデータの処理

CBWデータの処理は，USBでCBWデータを受信すると開始されます。

CBWデータの処理の流れを次に示します。

また，表3－1にCBWのデータ・フォーマットを，表3－2にCSWのデータ・フォーマットを示します。

図3－13　CBWデータの処理のフロー・チャート



サンプル・プログラムでのCBWデータの処理内容を，次に示します。

・CBWコマンドの解析

CBWデータの受信後，CBWの内容について解析を行います。

ここでは，CBWのタグを保存し，CBWCBの有効データ数，コマンドの方向について確認を行い，READ系，WRITE系，NO DATA系のそれぞれの処理タスクを起動します。

・READ系コマンドの処理

SCSIコマンドのREAD系コマンド処理を行うタスク（usbf850_data_in）を起動します。

このタスクでは，SCSIコマンド処理部を呼び出し，実行結果からCSWの送信ステータスを判定し，CSW発行処理を呼び出します。

・WRITE系コマンドの処理

SCSIコマンドのWRITE系コマンド処理を行うタスク（usbf850_data_out）を起動します。

このタスクでは，SCSIコマンド処理部を呼び出し，実行結果からCSWの送信ステータスを判定し，CSW発行処理を呼び出します。

・NO DATA系コマンドの処理

SCSIコマンドのNO DATA系コマンド処理を行うタスク（usbf850_no_data）を起動します。

このタスクでは，SCSIコマンド処理部を呼び出し，実行結果からCSWの送信ステータスを判定し，CSW発行処理を呼び出します。

・CSW発行

各コマンド処理タスクからコマンド実行結果を引数として呼び出されます。

引数からCSWデータを生成し，送信処理を行います。

[CBWの形式]

CBWは次に示す31バイトのデータで構成されています。

CBWCBの値によりホストからの処理内容を判断できます。

表3－1　CBWのデータ・フォーマット

ビット / バイト	7	6	5	4	3	2	1	0
0	dCBWSignature（55H）							
1	dCBWSignature（53H）							
2	dCBWSignature（42H）							
3	dCBWSignature（43H）							
4-7	dCBWTag（処理対象となった CBW のタグ）							
8-11	dCBWDataTransferLength（転送データ長）							
12	bmCBWFlag（Data-OUT/IN の指定）							
13	Reserved				bCBWLUN（対象デバイスの番号）			
14	Reserved			bCBWCBLength（CBWCB の有効バイト数）				
15-30	CBWCB（コマンド）							

[CSWの形式]

CSWは次に示す13バイトのデータで構成されています。

表3－2　CSWのデータ・フォーマット

ビット / バイト	7	6	5	4	3	2	1	0
0	dCSWSignature（53H）							
1	dCSWSignature（42H）							
2	dCSWSignature（53H）							
3	dCSWSignature（55H）							
4-7	dCSWTag（処理対象となった CBW のタグ）							
8-11	dCSWDataResidue（CBW 指定の転送データ長と処理したデータ長の差）							
12	bmCSWStatus（CBW 処理結果のステータス）							

(4) SCSIコマンドの処理

SCSIコマンドの処理は，USBでCBWデータの受信処理後開始されます。

サンプル・プログラムでは，表3－3に示す19種類のSCSI CDBコマンドに対応しています。

また，図3－14に，SCSIコマンド（READ系コマンド）処理の流れを示します。

注意　サンプル・プログラムで用意しているコマンドは，サンプル・プログラムの動作に必要な最低限のコマンドだけです。ユーザの環境によっては，サンプル・プログラムにないコマンド，および応答データの生成や処理方法について，必要な処理を追加してください。

表3－3　SCSIコマンド一覧

Command	Code	動　作
READ系		
REQUEST SENSE	03H	SENSEデータをホストに転送します。
READ（6）	08H	指定され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。
INQUIRY	12H	ターゲットとロジカル・ユニットについての構成情報や属性をホストに通知します。
MODE SENSE（6）	1AH	ロジカル・ユニットのモード・セレクト・パラメータの値や属性を読み出します。
READ FORMAT CAPACITIES	23H	ロジカル・ユニットの容量（ブロック数，ブロック長）をホストに通知します。
READ CAPACITY	25H	ロジカル・ユニット上のデータ容量をホストに通知します。
READ（10）	28H	READ（6）と同じです。
MODE SENSE（10）	5AH	MODE SENSE（6）と同じです。
WRITE系		
WRITE（6）	0AH	ホストからのデータを媒体上の指定されたブロックに書き込みます。
MODE SELECT（6）	15H	ロジカル・ユニット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。
WRITE（10）	2AH	WRITE（6）と同じです。
WRITE VERIFY	2EH	データを媒体に書き込んだあと，読み出して正常性の確認を行います。
VERIFY	2FH	ドライブ・ユニットの媒体上のデータの正常性確認を行います。
WRITE_ BUFF	3BH	ターゲットのメモリに任意のデータを書き込みます。
MODE_SELECT（10）	55H	MODE SELECT（6）と同じです。
NO DATA系		
TEST UNIT READY	00H	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。
SEEK	0BH	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。
START STOP UNIT	1BH	ロジカル・ユニットの媒体へのアクセスを可能にしたり不可能にしたりします。
SYNCHRONIZE CACHE	35H	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。

図3－14　READ系コマンドの処理のフロー・チャート（1/2）

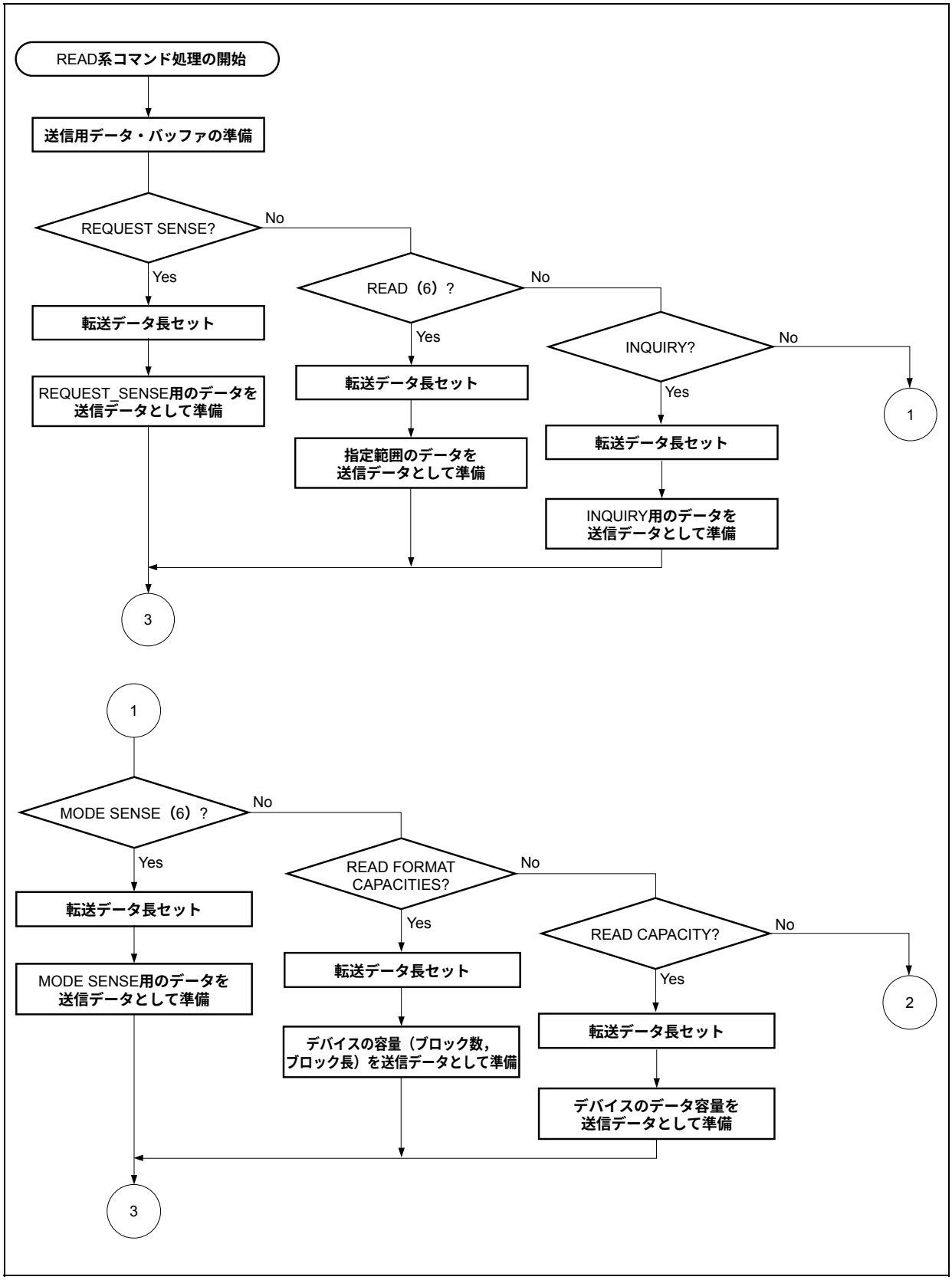
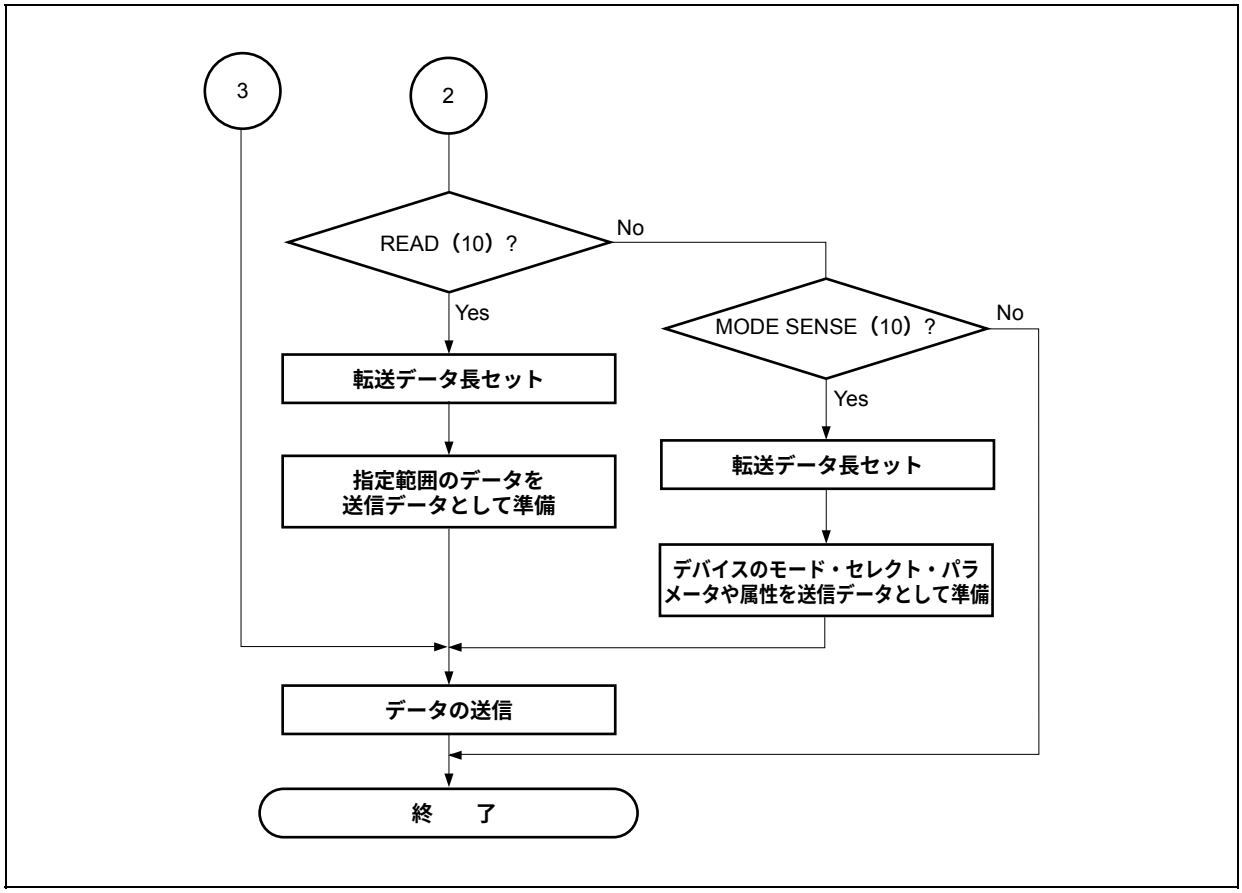


図3－14　READ系コマンドの処理のフロー・チャート（2/2）



[REQUEST SENSEコマンド処理]

センス・データをホストに報告します。

センス・データのフォーマットを次に示します。また，データ値として，サンプル・プログラムで使用するセンス・データについて示します。サンプル・プログラムでは仮想デバイスを扱うため，次に示すデータ値として用意されたデータを返します。

表3－4　センス・データ・フォーマット

ビット / バイト	7	6	5	4	3	2	1	0	データ値
0	VALID	エラー・コード							70H
1	Reserved								00H
2	Reserved		ILI	Reserved	センス・キー				00H
3	インフォメーション								00H
4	インフォメーション								00H
5	インフォメーション								00H
6	インフォメーション								00H
7	追加センス・データ長（n-7）								0AH
8	コマンド固有インフォメーション								00H
9	コマンド固有インフォメーション								00H
10	コマンド固有インフォメーション								00H
11	コマンド固有インフォメーション								00H
12	アディショナル・センス・コード（ASC）								00H
13	アディショナル・センス・コード・クォリファイア（ASCQ）								00H
14	FRU（Field Replaceable Unit）コード								00H
15	SKSV	センス・キー固有インフォメーション							00H
16	センス・キー固有インフォメーション								00H
17	センス・キー固有インフォメーション								00H

サンプル・ドライバ内でホストに送信するセンス・キーの一覧を，次に示します。

表3－5　センス・キー一覧

センス・キー	ASC	ASCQ	Description of Error
00	00	00	NO SENSE
05	00	00	ILLEGAL REQUEST.
05	20	00	INVALID COMMAND OPERATION CODE
05	24	00	INVALID FIELD IN COMMAND PACKET

[READ（6）コマンド処理]

ストレージ・デバイスから指定範囲のデータを読み出し，読み出したデータをホストへ送信します。

サンプル・プログラムでは，仮想デバイスから読み出したデータをホストへ送信します。

［INQUIRYコマンド処理］

デバイスについての情報をホストに報告します。

INQUIRYデータのフォーマットを次に示します。また，サンプル・プログラムでは仮想デバイスを扱うため，次に示すデータ値として用意されたデータを返します。

表3－6　INQUIRYデータ・フォーマット

ビット / バイト	7	6	5	4	3	2	1	0	データ値
0	クォリファイア			デバイス・タイプ・コード					00H
1	RMB	デバイス・タイプ修飾子							80H
2	ISO バージョン		ECMA バージョン			00H			00H
3	AENC	TrmIOP	01H		01H				02H
4	追加データ長（n－4 バイト）								1FH
5	Reserved								00H
6	Reserved								00H
7	Reserved								00H
8	ベンダ ID（ASCII）								注1
：	：								注1
15	ベンダ ID（ASCII）								注1
16	プロダクト ID（ASCII）								注2
：	：								注2
31	プロダクト ID（ASCII）								注2
32	プロダクト版数（ASCII）								注3
：	：								注3
35	プロダクト版数（ASCII）								注3
36	ベンダ固有								none
：	：								none
55	ベンダ固有								none
56	Reserved								none
：	：								none
95	Reserved								none
96	ベンダ固有								none
：	：								none
n	ベンダ固有								none

注1.　“ NEC Corp ”のASCII文字コード

2.　“ StorageFncDriver ”のASCII文字コード

3.　“ 0.12 ”のASCII文字コード

[MODE SENSE（6）コマンド処理]

デバイスのモード・セレクト・パラメータや属性をホストに報告します。

MODE SENSEデータのフォーマットを次に示します。また，サンプル・プログラムでは仮想デバイスを扱うため，次に示すデータ値として用意されたデータを返します。対応しているページ・コードは，01Hだけです。コマンドのページ・コードに関係なく，このデータを返します。

表3－7　MODE SENSEデータ・フォーマット

ビット / バイト	7	6	5	4	3	2	1	0	データ値
0	モード・パラメータ長								注1
1	メディア・タイプ								00H
2	デバイス固有パラメータ								00H
3	ブロック・ディスクリプタ長								08H
4	デンシティ・コード								00H
5	00H								00H
6	00H								00H
7	C0H								C0H
8	Reserved								00H
9	00H								00H
10	00H								02H
11	00H								00H
12	PS	Reserved	ページ・コード						注2
13	ページ長（n－13）								0AH
14	モード・パラメータ								注3
：	：								注3
n	モード・パラメータ								注3

注1. CDBのDBDとページ・コードで指定されるパラメータ・リストか，またはアロケーション長で指定される分のパラメータ・リストかの，どちらか少ない方のバイト数

2. CDBのページ・コード

3. 08H, 0BH, 00H, 00H, 00H, 00H, 00H, 00H, 00H, 00H

[READ FORMAT CAPACITYコマンド処理]

デバイスの容量（ブロック数，ブロック長）をホストに報告します。

READ FORMAT CAPACITYデータのフォーマットを次に示します。また，サンプル・プログラムでは仮想デバイスを扱うため，次に示すデータ値として用意されたデータを返します。

表3－8　READ FORMAT CAPACITY用データ・フォーマット

ビット / バイト	7	6	5	4	3	2	1	0	データ値
0	Reserved								00H
1	Reserved								00H
2	Reserved								00H
3	キャパシティ・リスト長（バイト）								08H
4	ブロック数								00H
5	ブロック数								00H
6	ブロック数								00H
7	ブロック数								C0H
8	Reserved						Descriptor Code		01H
9	ブロック長								00H
10	ブロック長								02H
11	ブロック長								00H
12	ブロック数								00H
13	ブロック数								00H
14	ブロック数								00H
15	ブロック数								C0H
16	Reserved								00H
17	ブロック長								00H
18	ブロック長								02H
19	ブロック長								00H

[READ CAPACITYコマンド処理]

デバイスのデータ容量をホストに報告します。

READ CAPACITYデータのフォーマットを次に示します。また，サンプル・プログラムでは仮想デバイスを扱うため，次に示すデータ値として用意されたデータを返します。

表3－9　READ CAPACITY用データ・フォーマット

ビット / バイト	7	6	5	4	3	2	1	0	データ値
0	論理ブロック・アドレス								00H
1	論理ブロック・アドレス								00H
2	論理ブロック・アドレス								00H
3	論理ブロック・アドレス								BFH
4	ブロック長								00H
5	ブロック長								00H
6	ブロック長								02H
7	ブロック長								00H

[READ（10）コマンド処理]

ストレージ・デバイスから指定範囲のデータを読み出し，読み出したデータをホストへ送信します。

サンプル・プログラムでは，仮想デバイスから読み出したデータをホストへ送信します。

[MODE SENSE（10）コマンド処理]

デバイスのモード・セレクト・パラメータや属性をホストに報告します。

MODE SENSE（10）データのフォーマットを次に示します。また，サンプル・プログラムでは仮想デバイスを扱うため，次に示すデータ値として用意されたデータを返します。対応しているページ・コードは，01Hだけです。コマンドのページ・コードに関係なく，このデータを返します。

表3－10　MODE SENSE（10）データ・フォーマット

ビット / バイト	7	6	5	4	3	2	1	0	データ値
0	モード・パラメータ長								注1
1	モード・パラメータ長								注1
2	メディア・タイプ								00H
3	デバイス固有パラメータ								00H
4	Reserved								00H
5	Reserved								00H
6	ブロック・ディスクリプタ長								00H
7	ブロック・ディスクリプタ長								08H
8	デンシティ・コード								00H
9	ブロック数								00H
10	ブロック数								00H
11	ブロック数								C0H
12	Reserved								00H
13	ブロック長								00H
14	ブロック長								02H
15	ブロック長								00H
16	PS	Reserved	ページ・コード						注2
17	ページ長（n－17）								0AH
18	モード・パラメータ								注3
：	：								注3
n	モード・パラメータ								注3

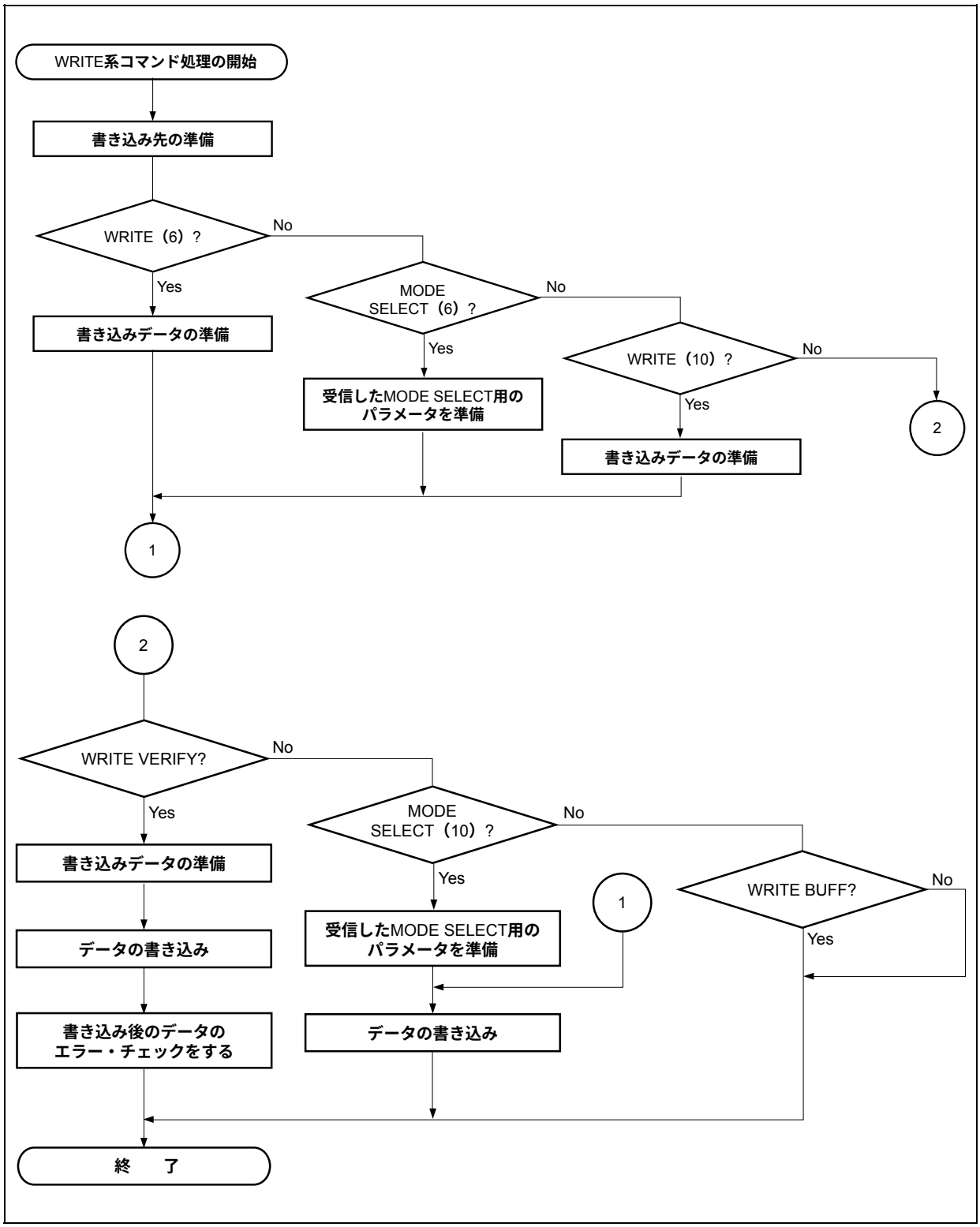
注1. CDBのDBDとページ・コードで指定されるパラメータ・リストか，またはアロケーション長で指定される分のパラメータ・リストかの，どちらか少ない方のバイト数

2. CDBのページ・コード

3. 08H, 0BH, 00H, 00H, 00H, 00H, 00H, 00H, 00H, 00H

SCSIコマンド（WRITE系コマンド）処理の流れを次に示します。

図3－15　WRITE系コマンドの処理のフロー・チャート



[WRITE（6）コマンド処理]

受信データをストレージ・デバイスの指定された領域に書き込みます。

サンプル・プログラムでは，受信データを仮想デバイスの指定された領域に書き込みます。

[MODE SELECT（6）コマンド処理]

ロジカル・ユニットの物理的属性，記憶媒体上のデータ形式，エラー・リカバリの方法や手順など，各種パラメータの設定や変更を行います。

MODE SELECTデータのフォーマットを次に示します。また，サンプル・プログラムでは仮想デバイスを扱うため,コマンドのページ･コードに関係なく,受信したデータをMODE SELECT TABLEに書き込むだけで，すべて正常終了します。なお，データ値をテーブルの初期値として使用するものとします。

表3－11　MODE SELECT（6）データ・フォーマット

ビット / バイト	7	6	5	4	3	2	1	0	データ値
0	モード・パラメータ長								17H
1	メディア・タイプ								00H
2	デバイス固有パラメータ								00H
3	ブロック・ディスクリプタ長								08H
4	デンシティ・コード								00H
5	00H								00H
6	00H								00H
7	C0H								C0H
8	Reserved								00H
9	00H								00H
10	00H								02H
11	00H								00H
12	PS	1	ページ・コード						注1
13	ページ長（n－13）								0AH
14	モード・パラメータ								注2
：	：								注2
n	モード・パラメータ								注2

注1.　CDBのページ・コード

2.　08H, 0BH, 00H, 00H, 00H, 00H, 00H, 00H, 00H, 00H

[WRITE（10）コマンド処理]

受信データを，ストレージ・デバイスの指定された領域に書き込みます。

サンプル・プログラムでは，受信データを仮想デバイスの指定された領域に書き込みます。

[WRITE VERIFYコマンド処理]

受信データをストレージ・デバイスに書き込みます。書き込み後，データのエラー・チェックをします。サンプル・プログラムでは，受信データを仮想デバイスに書き込み，データのエラー・チェックはせずに正常終了します。

[VERIFYコマンド処理]

ストレージ・デバイス上のデータの正常性を確認します。サンプル・プログラムでは仮想デバイスを扱うため，何も処理せず正常終了します。

[WRITE BUFFコマンド処理]

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・プログラムでは仮想デバイスを扱うため，何も処理せず正常終了します。

[MODE SELECT（10）コマンド処理]

ロジカル・ユニットの物理的属性，記憶媒体上のデータ形式，エラー・リカバリの方法や手順など，各種パラメータの設定や変更を行います。

MODE SELECT（10）データのフォーマットを次に示します。また，サンプル・プログラムでは仮想デバイスを扱うため，コマンドのページ・コードに関係なく，受信したデータをMODE SELECT（10）TABLEに書き込むだけで，すべて正常終了します。なお，データ値をテーブルの初期値として使用するものとします。

表3－12　MODE SELECT（10）データ・フォーマット

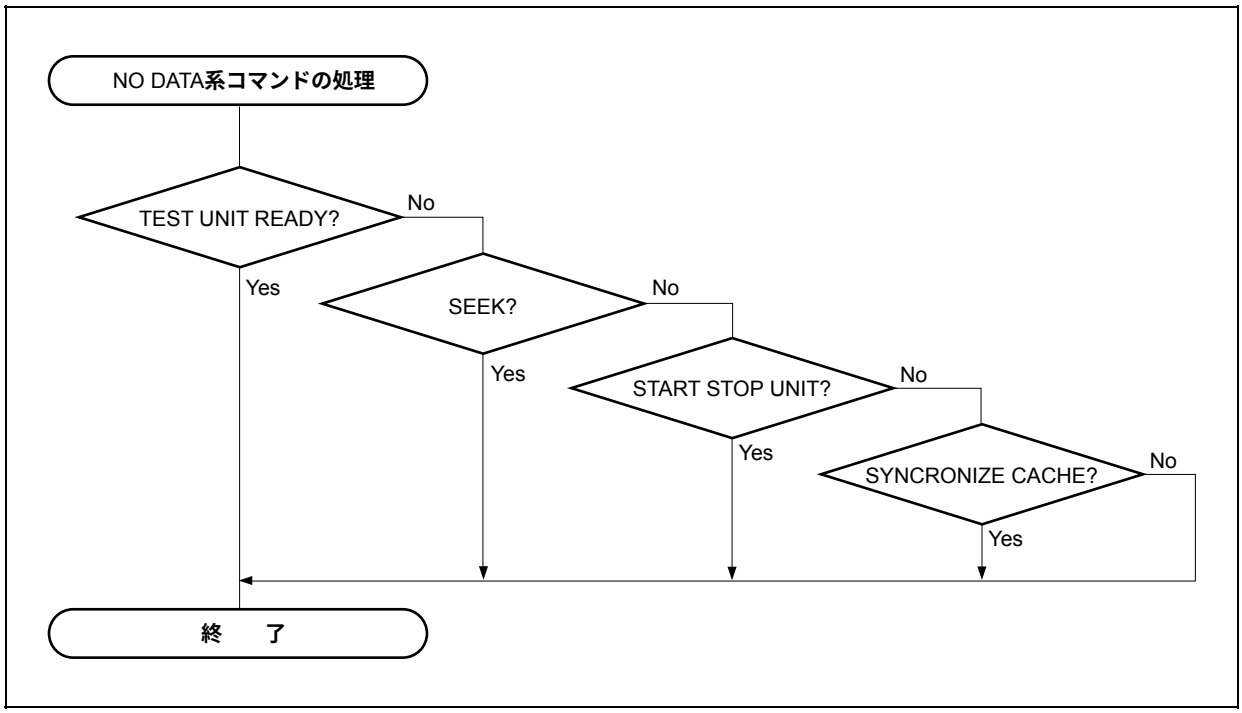
ビット / バイト	7	6	5	4	3	2	1	0	データ値
0	モード・パラメータ長								00H
1	モード・パラメータ長								1AH
2	メディア・タイプ								00H
3	デバイス固有パラメータ								00H
4	Reserved								00H
5	Reserved								00H
6	ブロック・ディスクリプタ長								00H
7	ブロック・ディスクリプタ長								08H
8	デンシティ・コード								00H
9	ブロック数								00H
10	ブロック数								00H
11	ブロック数								C0H
12	Reserved								00H
13	ブロック長								00H
14	ブロック長								02H
15	ブロック長								00H
16	PS	Reserved	ページ・コード						注1
17	ページ長（n－17）								0AH
18	モード・パラメータ								注2
：	：								注2
n	モード・パラメータ								注2

注1.　CDBのページ・コード

2.　08H, 0BH, 00H, 00H, 00H, 00H, 00H, 00H, 00H, 00H

SCSIコマンド（NO DATA系コマンド）処理の流れを次に示します。

図3－16　NO DATA系コマンドの処理のフロー・チャート



[TEST UNIT READYコマンド処理]

ユニットの状態を報告します。サンプル・プログラムでは仮想デバイスを扱うため，何も処理せず正常終了します。

[SEEKコマンド処理]

指定ブロック位置へのシーク動作を行います。サンプル・プログラムでは仮想デバイスを扱うため，何も処理せず正常終了します。

[START STOP UNITコマンド処理]

ユニットへのアクセス制限を設定します。サンプル・プログラムでは仮想デバイスを扱うため，何も処理せず正常終了します。

[SYNCHRONIZE CACHEコマンド処理]

ユニットとCACHEとのデータの整合をとります。サンプル・プログラムでは仮想デバイスを扱うため，何も処理せず正常終了します。

3. 7. 3　USBストレージ・クラス用ドライバのディスクリプタ情報

サンプル・プログラムで定義されているUSB標準ディスクリプタについて，次に示します。

(a) - (d) までのディスクリプタは，必ず用意してください。

備考　詳細は，Universal Serial Bus Specification Revision 1.1を参照してください。

(a) Device Descriptor

デバイスの一般的な情報を持ち，デバイスごとに1つのDevice Descriptorを用意してください。この情報は，デバイスのコンフィギュレーションで唯一のデバイスを認識するために使用されます。USBストレージ・クラスは，デバイス・レベルで現在のクラスにおける具体的な情報は使用しません。

表3ー13　Device Descriptor

オフセット	サイズ（バイト）	値	説　明
0	1	12H	このディスクリプタの Length 値（バイト）
1	1	01H	ディスクリプタ・タイプ（デバイス）
2	2	10H/01H	USB のバージョン（USB1.1）
4	1	00H	クラス・コード
5	1	00H	サブクラス・コード
6	1	00H	プロトコル・コード
7	1	40H	Endpoint0 の最大パケット・サイズ
8	2	09H/04H	ベンダ ID（NEC エレクトロニクス）
10	2	FCH/FFH	プロダクト ID
12	2	01H/00H	デバイス・リリース番号
14	1	01H	ストリング・ディスクリプタへのインデクス（Manufacturer）
15	1	00H	ストリング・ディスクリプタへのインデクス（Product）
16	1	00H	ストリング・ディスクリプタへのインデクス（Serial Number）
17	1	01H	可能なコンフィギュレーションの数

（b）Configuration Descriptor

具体的なデバイス・コンフィギュレーションについての情報を保持しています。

USBストレージ・クラスは，コンフィギュレーション・レベルで現在のクラスにおける具体的な情報は使用しません。

表3－14　Configuration Descriptor

オフセット	サイズ（バイト）	値	説　明
0	1	09H	このディスクリプタのLength値（バイト）
1	1	02H	ディスクリプタ・タイプ（コンフィギュレーション）
2	2	20H/00H	Get Descriptor要求でコンフィギュレーション・ディスクリプタと一緒に返されるディスクリプタのトータルLength値
4	1	01H	この構成でサポートされているインタフェース数
5	1	01H	コンフィギュレーション値
6	1	00H	ストリング・ディスクリプタへのインデクス（Configuration）
7	1	C0H	デバイスの構成（Self-powered/Remote Wakeup機能）
8	1	00H	デバイスの最大消費電力

（c）Interface Descriptor

コンフィギュレーション内の具体的なインタフェース情報を保持しています。

サンプル・プログラムでは，コンフィギュレーションは1つのインタフェースを提供します。また，このインタフェースは2つのEndpointをサポートし，この数だけEndpoint Descriptorを持っています。

Interface Descriptorは，常にConfiguration Descriptorの一部として戻され，Interface Descriptorにおいて，Get DescriptorおよびSet Descriptor要求による直接アクセスはされません。

表3－15　Interface Descriptor（1）

オフセット	サイズ（バイト）	値	説　明
0	1	09H	このディスクリプタのLength値（バイト）
1	1	04H	ディスクリプタ・タイプ（インタフェース）
2	1	00H	インタフェース値
3	1	00H	Alternate設定値
4	1	02H	Endpoint数（Endpoint0を除く）
5	1	08H	インタフェース・クラス（マス・ストレージ・クラス）
6	1	06H	インタフェース・サブクラス（SCSI）
7	1	50H	インタフェース・プロトコル（Bulk-Only）
8	1	00H	ストリング・ディスクリプタへのインデクス（インタフェース）

（d）Endpoint Descriptor

ホストが個々のEndpointのバンド幅要件を決定するために必要な情報を保持しています。

Endpoint Descriptorは，常にコンフィギュレーション・ディスクリプタの一部として戻され，Endpoint Descriptorにおいて，Get DescriptorおよびSet Descriptor要求による直接アクセスはされません。

表3－16　Endpoint Descriptor（Bulk IN）

オフセット	サイズ（バイト）	値	説　明
0	1	07H	このディスクリプタのLength値（バイト）
1	1	05H	ディスクリプタ・タイプ（Endpoint）
2	1	81H	Endpointのアドレス値
3	1	02H	Endpointの転送タイプ
4	2	40H/00H	Endpointの最大パケット・サイズ
6	1	00H	インターバル（ms）：Isochronous, Interrupt Endpointだけ有効

表3－17　Endpoint Descriptor（Bulk OUT）

オフセット	サイズ（バイト）	値	説　明
0	1	07H	このディスクリプタのLength値（バイト）
1	1	05H	ディスクリプタ・タイプ（Endpoint）
2	1	02H	Endpointのアドレス値
3	1	02H	Endpointの転送タイプ
4	2	40H/00H	Endpointの最大パケット・サイズ
6	1	00H	インターバル（ms）：Isochronous, Interrupt Endpointだけ有効

（e）String Descriptor

このサンプル・プログラムでは，デバイスの製造社名情報を保持します。

表3－18　String Descriptor（1）

オフセット	サイズ（バイト）	値	説　明
0	1	04H	このディスクリプタのLength値（バイト）
1	1	03H	ディスクリプタ・タイプ（ストリング）
2	2	09H/04H	ストリング・ディスクリプタで使用する言語タイプ（English/U.S.）

表3－19　String Descriptor（2）

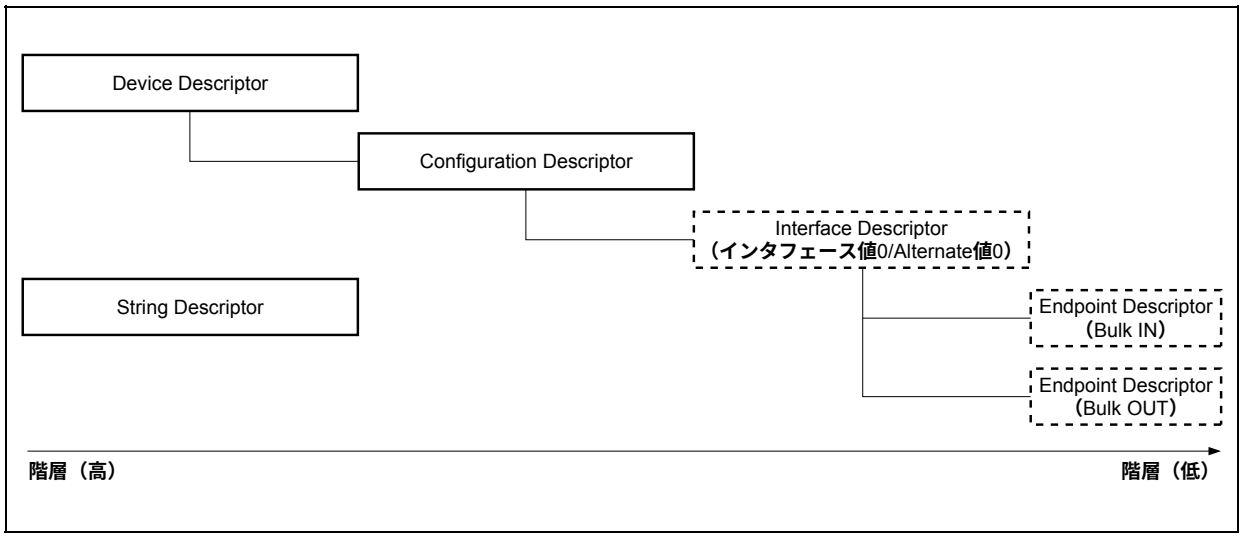
オフセット	サイズ（バイト）	値	説　明
0	1	2AH	このディスクリプタのLength値（バイト）
1	1	03H	ディスクリプタ・タイプ（ストリング）
2	40	'N','E','C','','E','l','e','c','t','r','o','n','i','c','s',' ','C','o','.'	製造社名（Manifacturer） NEC Electronics Co.

(1) ディスクリプタの構成

サンプル・プログラムにおけるディスクリプタの構成を次に示します。前述したディスクリプタ類を次の構成で準備します。

注意　Device Descriptor/Configuration Descriptor/String Descriptorは，それぞれ別々のGet Descriptorリクエストによってアクセスされます。Interface DescriptorおよびEndpoint Descriptorは，Configuration Descriptorの一部としてアクセスされます。

図3－17　ディスクリプタ構成図



3. 7. 4　データ・マクロ

USBストレージ・クラス用ドライバ内で使用する各種データ・マクロ（データ・タイプ，戻り値など）について，次に示します。

(1) データ・タイプ

USBストレージ・クラス用ドライバ関数を発行するときに指定する各種パラメータのデータ・タイプのマクロ定義は，ヘッダ・ファイルnectools32¥USB_Storage¥inc¥types.hで行われています。

データ・タイプ一覧を，次に示します。

表3－20　データ・タイプ一覧

マクロ	型	意　味
ULONG	unsigned long	符号なし32ビット整数
WORD	unsigned long	符号なし32ビット整数
HWORD	unsigned short	符号なし16ビット整数
BYTE	unsigned char	符号なし8ビット整数
(*PFV) ()	void	処理プログラムの起動アドレス

(2) 戻 り 値

USBストレージ・クラス用ドライバ関数からの戻り値のマクロ定義は，ヘッダ・ファイルnectools32¥USB_BUS¥inc¥errno.hで行われています。

戻り値一覧を次に示します。

表3－21　戻り値一覧

マクロ	数　値	意　味
DEV_OK	0	正常終了
DEV_ERROR	－1	異常終了
DEV_ERR_NODATA	－2	NO DATA系コマンドで転送方向エラー
DEV_ERR_READ	－3	READ系コマンドで転送方向エラー
DEV_ERR_WRITE	－4	WRITE系コマンドで転送方向エラー
DEV_ERR_VERIFY	－5	ベリファイ・エラー
DEV_ERR_CBWLENGTH	－6	CBWレングス・エラー
DEV_ERR_CBWCBW	－7	CBW処理中にCBWを受信（エラー）

3. 7. 5　データ構造体

USBストレージ・クラス用ドライバが使用するデータ構造体について，次に示します。

(1) USBデバイス・リクエスト構造体

USBデバイス・リクエスト構造体の定義は，USB用ヘッダ・ファイルnectools32¥V850USB_Storage¥src¥USBF¥usbf850.hで行われています。USBデバイス・リクエスト構造体USB_SETUPを次に示します。

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

(2) CBWデータ構造体

USBストレージ・クラス用ドライバが扱うCBW(Command Block Wrapper)用のデータ構造体の定義は，ヘッダ・ファイルnectools32¥USB_Storage¥inc¥types.hで行われています。CBWデータ構造体を次に示します。

```
typedef struct {
        unsigned char   dCBWSignature[4];           /*CBW シグネチャ*/
        unsigned char   dCBWTag[4];                 /*CBW タグ*/
        unsigned char   dCBWDataTransferLength[4];  /*転送データ長*/
        unsigned char   bmCBWFlags;                 /*データ方向（OUT/IN）の指定*/
        unsigned char   bCBWLUN;                    /*対象デバイスの番号*/
        unsigned char   bCBWCBLength;               /*CBWCB の有効バイト数*/
        unsigned char   CBWCB[16];                  /*CBWCB（コマンド）*/
} CBW_INFO,*PCBW_INFO;
```

(3) CSWデータ構造体

USBストレージ・クラス用ドライバが扱うCSW（Command Status Wrapper）用のデータ構造体の定義は，ヘッダ・ファイルnectools32¥USB_Storage¥inc¥types.hで行われています。CSWデータ構造体を次に示します。

```
typedef struct {
     unsigned char  dCSWSignature[4];    /*CSW シグネチャ*/
     unsigned char  dCSWTag[4];          /*CSW タグ*/
     unsigned char  dCSWDataResidue[4];  /*CBW 指定の転送データ長と処理したデータ長の差*/
     unsigned char  bmCSWStatus;         /*CBW 処理結果のステータス*/
} CSW_INFO,*PCSW_INFO;
```

3. 7. 6　関数解説

(1) 概　　要

この章で説明している各処理プログラムの一覧を次に示します。

表3－22　サンプル・プログラムの処理プログラム一覧（1/3）

処理プログラム名	関数名	ファイル名	備考
RX850 Pro依存処理プログラム			
CF定義ファイル	－	sys.cf	－
エントリ処理	－	entry.850	アセンブリ言語
ブート処理	boot	boot.850	アセンブリ言語
ハードウエア初期化部	__InitSystemTimer	init.c	C言語
初期化ハンドラ	varfunc	varfunc.c	C言語
ヘッダ・ファイル	－	init.h	－
ボード依存部処理プログラム			
ポートの初期化	port850_reset	port.c	C言語
ヘッダ・ファイル	－	port.h	－
ヘッダ・ファイル			
データ・タイプ宣言	－	types.h	－
戻り値宣言	－	errno.h	－
ビルド・ファイル	－	usb_bus.bld	－
セクション・マップ・ファイル	－	common.lx	－

表3－22　サンプル・プログラムの処理プログラム一覧（2/3）

処理プログラム名	関数名	ファイル名	備考
USBストレージ・クラス用ドライバ処理プログラム			
初期化関数	usbf850_init	usbf850.c	C言語
割り込みハンドラ（INTUSB0B信号用）	usbf850_inthdr	usbf850.c	C言語
割り込みハンドラ（INTUSB1B信号用）	usbf850_inthdr1	usbf850.c	C言語
割り込みハンドラ（INTUSB2B信号用）	usbf850_inthdr2	usbf850.c	C言語
割り込み処理用タスク（INTUSB0B信号用）	task_usb0b	usbf850.c	C言語
割り込み処理用タスク（INTUSB1B信号用）	task_usb1b	usbf850.c	C言語
割り込み処理用タスク（INTUSB2B信号用）	task_usb2b	usbf850.c	C言語
データ送信関数	usbf850_data_send	usbf850.c	C言語
データ受信関数	usbf850_data_receive	usbf850.c	C言語
Nullデータ送信関数（エンドポイント0）	usbf850_sendnullEP0	usbf850.c	C言語
Stall応答処理関数（エンドポイント0）	usbf850_sendstallEP0	usbf850.c	C言語
Stall応答処理関数（エンドポイント1）	usbf850_bulkin1_stall	usbf850.c	C言語
Stall応答処理関数（エンドポイント2）	usbf850_bulkout1_stall	usbf850.c	C言語
システム・コール呼び出し関数（loc_cpu）	usbf850_loc_cpu	usbf850.c	C言語
システム・コール呼び出し関数（unl_cpu）	usbf850_unl_cpu	usbf850.c	C言語
リクエスト処理関数	usbf850_rxreq	usbf850.c	C言語
リクエスト・データ読み出し関数	usbf850_rxreq_read	usbf850.c	C言語
標準リクエスト処理関数	usbf850_standardreq	usbf850.c	C言語
Get Descriptorリクエスト処理関数	usbf850_getdesc	usbf850.c	C言語
リクエスト処理関数設定用Stall応答処理関数（エンドポイント0）	usbf850_sstall_ctrl	usbf850.c	C言語
USB用ヘッダ・ファイル	－	usbf850.h	－
USB用ディスクリプタ宣言	－	usbf850desc.h	－
Bulk-Only Mass Storage Resetリクエスト処理関数（デバイス・クラス固有のリクエスト処理）	usbf850_blkonly_mass_storage_reset	usbf850_storage.c	C言語
Max LUNリクエスト処理関数（デバイス・クラス固有のリクエスト処理）	usbf850_max_lun	usbf850_storage.c	C言語
USBストレージ・クラス用デバイス・クラス固有のリクエスト処理関数の登録処理関数	usbf850_setfunction_storage	usbf850_storage.c	C言語
CBW受信処理関数	usbf850_rx_cbw	usbf850_storage.c	C言語
CBWチェック関数	usbf850_storage_cbwchk	usbf850_storage.c	C言語
CBWのエラー処理関数	usbf850_cbw_error	usbf850_storage.c	C言語
CBWのNO DATA系コマンド処理関数	usbf850_no_data	usbf850_storage.c	C言語
CBWのDATA IN系コマンド処理関数	usbf850_data_in	usbf850_storage.c	C言語
CBWのDATA OUT系コマンド処理関数	usbf850_data_out	usbf850_storage.c	C言語
CSW送信処理関数	usbf850_csw_ret	usbf850_storage.c	C言語
USB-ストレージ間インタフェース関数用ヘッダ・ファイル	－	usbf850_storage.h	－
USB用DMA初期化処理関数	usbf850_dma_init	usbf850_dma.c	C言語
USB用DMA開始処理関数	usbf850_dma_start	usbf850_dma.c	C言語
DMA用ヘッダ・ファイル	－	usbf850_dma.h	－

表3－22　サンプル・プログラムの処理プログラム一覧（3/3）

処理プログラム名	関数名	ファイル名	備考
ストレージ・デバイス処理プログラム			
ストレージ・デバイス初期化関数	storageDev_Init	ata_ctrl.c	C言語
ストレージ・デバイス用ヘッダ・ファイル	－	ata.h	－
CBWCBコマンド解析処理関数	scsi_command_to_ata	scsi_cmd.c	C言語
TEST UNIT READYコマンド処理関数	ata_test_unit_ready	scsi_cmd.c	C言語
SEEKコマンド処理関数	ata_seek	scsi_cmd.c	C言語
START STOP UNITコマンド処理関数	ata_start_stop_unit	scsi_cmd.c	C言語
SYNCHRONIZE CACHEコマンド処理関数	ata_synchronize_cache	scsi_cmd.c	C言語
REQUEST SENSEコマンド処理関数	ata_request_sense	scsi_cmd.c	C言語
INQUIRYコマンド処理関数	ata_inquiry	scsi_cmd.c	C言語
MODE SELECTコマンド処理関数	ata_mode_select	scsi_cmd.c	C言語
MODE SELECT（10）コマンド処理関数	ata_mode_select10	scsi_cmd.c	C言語
MODE SENSEコマンド処理関数	ata_mode_sense	scsi_cmd.c	C言語
MODE SENSE(10)コマンド処理関数	ata_mode_sense10	scsi_cmd.c	C言語
READ FORMAT CAPACITIESコマンド処理関数	ata_read_format_capacities	scsi_cmd.c	C言語
READ CAPACITYコマンド処理関数	ata_read_capacity	scsi_cmd.c	C言語
READ（6）コマンド処理関数	ata_read6	scsi_cmd.c	C言語
READ（10）コマンド処理関数	ata_read10	scsi_cmd.c	C言語
WRITE（6）コマンド処理関数	ata_write6	scsi_cmd.c	C言語
WRITE（10）コマンド処理関数	ata_write10	scsi_cmd.c	C言語
VERIFYコマンド処理関数	ata_verify	scsi_cmd.c	C言語
WRITE VERIFYコマンド処理関数	ata_write_verify	scsi_cmd.c	C言語
WRITE BUFFコマンド処理関数	ata_write_buff	scsi_cmd.c	C言語
SCSIからUSB向けデータ送信処理関数	scsi_to_usb	scsi_cmd.c	C言語
SCSIコマンド処理用ヘッダ・ファイル	－	scsi.h	－
関数マクロ			
V850E/ME2周辺I/Oレジスタ設定関数（1バイト単位：8ビット）	USBF850REG_SET	usbf850.h	C言語
V850E/ME2周辺I/Oレジスタ読み出し関数（1バイト単位：8ビット）	USBF850REG_READ	usbf850.h	C言語
V850E/ME2周辺I/Oレジスタ設定関数（1ワード単位：16ビット）	USBF850REG_SET_W	usbf850.h	C言語
V850E/ME2周辺I/Oレジスタ読み出し関数（1ワード単位：16ビット）	USBF850REG_READ_W	usbf850.h	C言語

(2) 関数ツリー

サンプル・プログラムの呼び出し関係（関数ツリー）を次に示します。

注意　usbf850_initおよびstorageDev_Initは，初期化ハンドラから呼び出されます。
また，usbf850_dma_initは，usbf850_initから呼び出されます。

図3－18　サンプル・プログラムの関数ツリー（1/4）

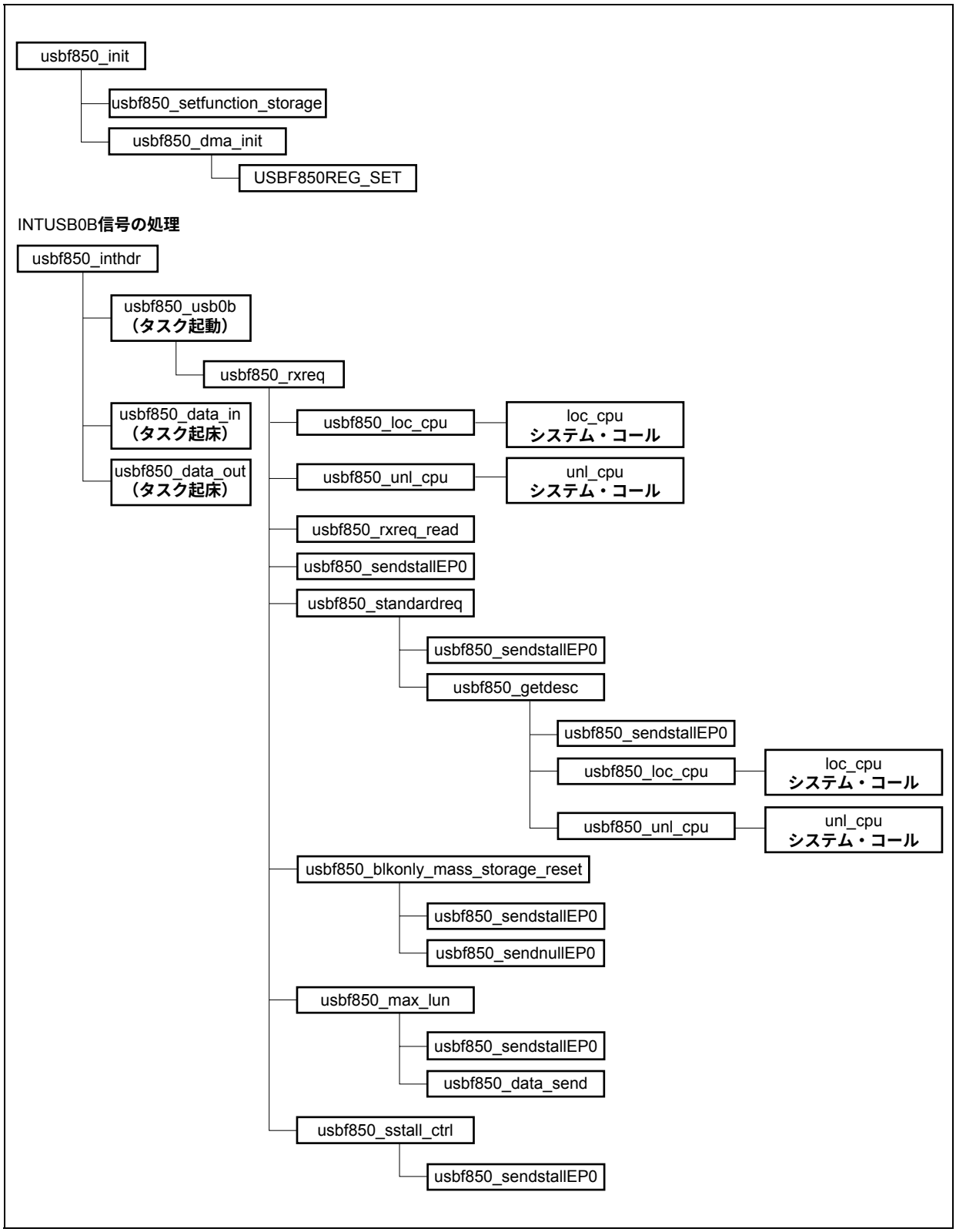


図3－18　サンプル・プログラムの関数ツリー（2/4）

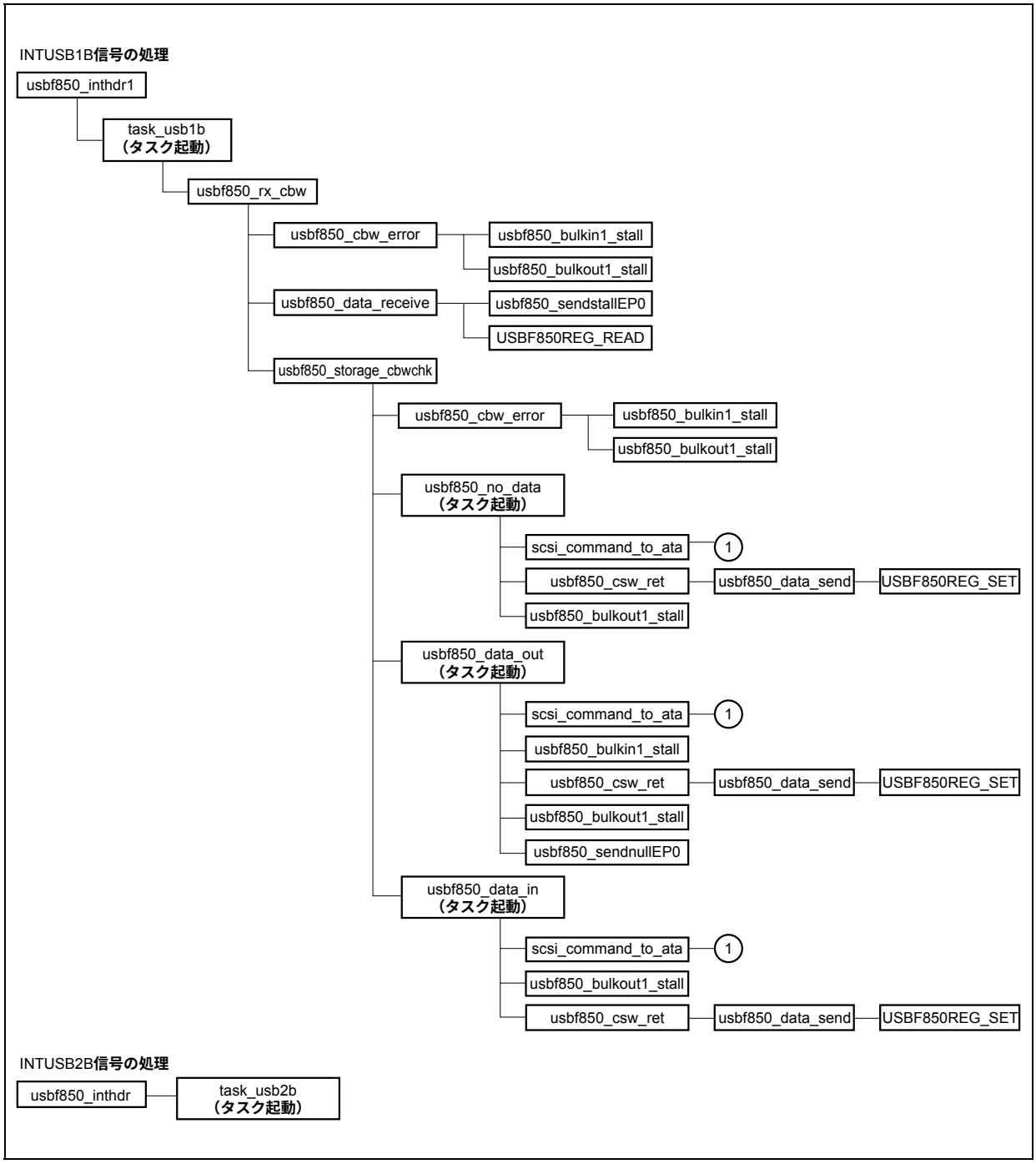


図3－18　サンプル・プログラムの関数ツリー（3/4）

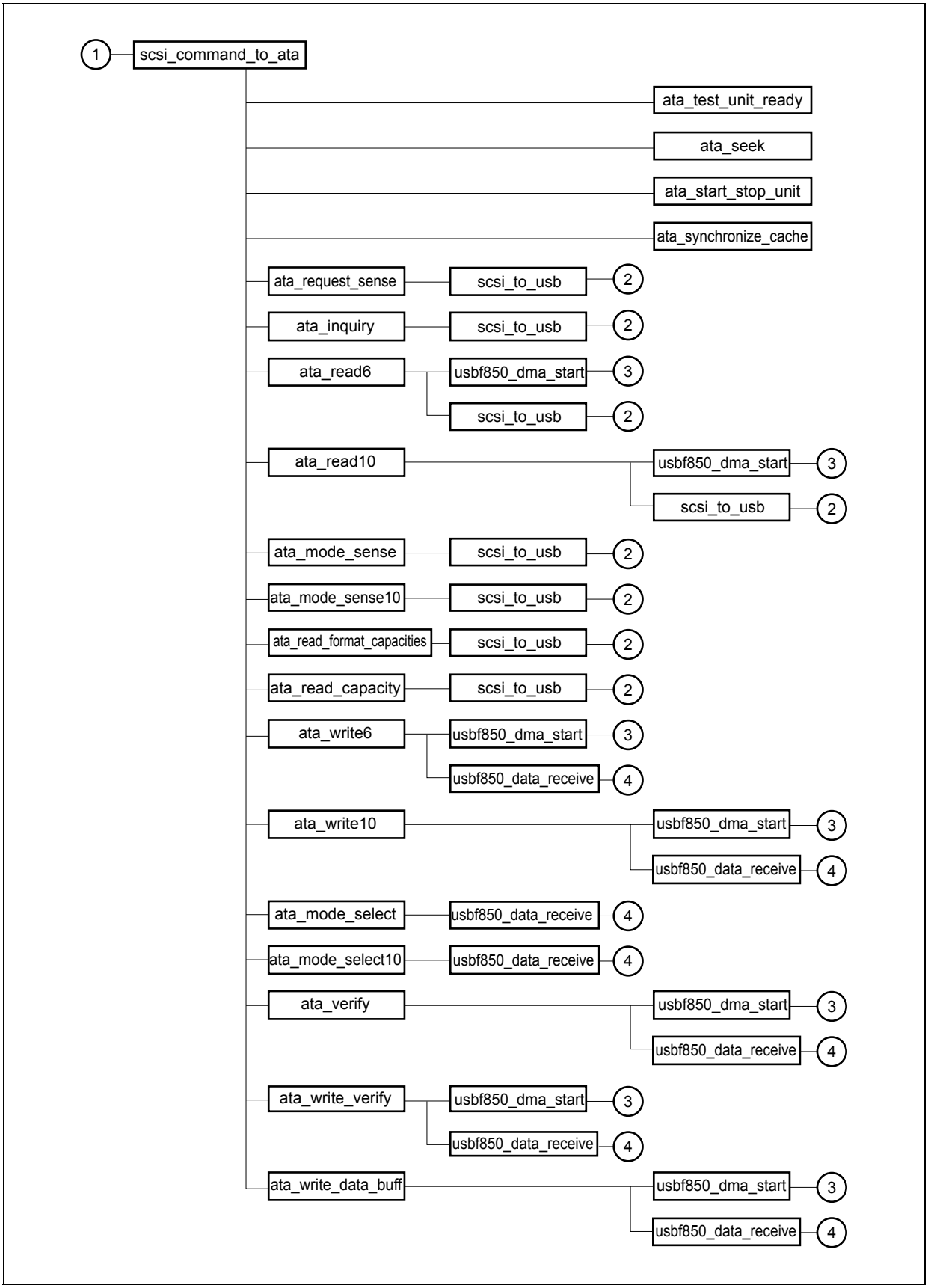
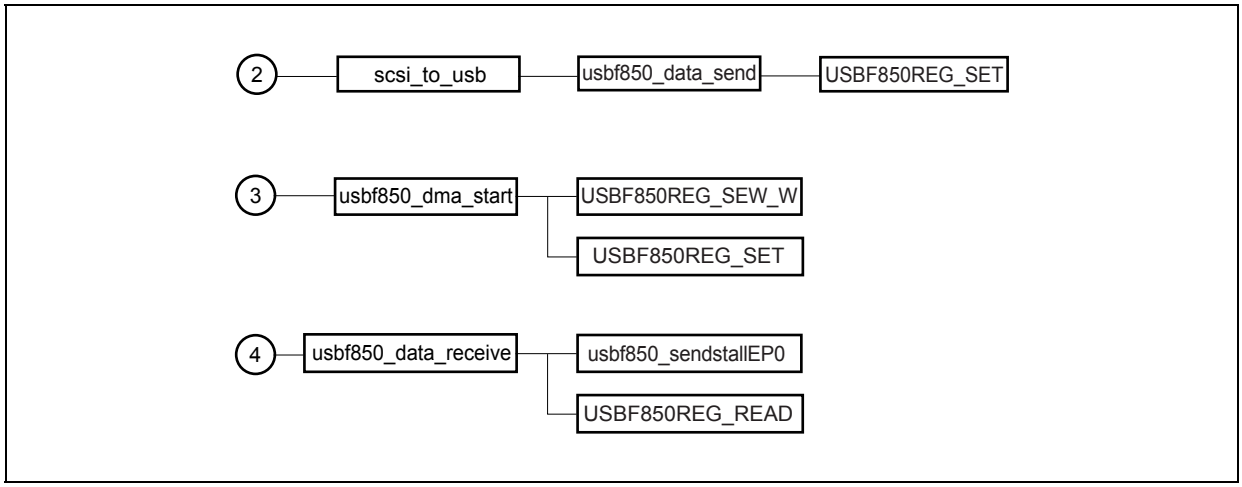


図3－18　サンプル・プログラムの関数ツリー（4/4）



(3) 関数解説

このサンプル・プログラムの関数群について，次の記述フォーマットにしたがって解説します。

XXXX … ①　　**発行有効範囲：－－－－ … ②**

[概　　要]　… ③

－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－

－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－

[C言語形式]　… ④

－－－－－－－－

[パラメータ]　… ⑤

I/O	パラメータ	説　　明

[機　　能]　… ⑥

－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－

－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－

[戻 り 値]　… ⑦

－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－

－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－

① 名称

関数の名称を示しています。

② 発行有効範囲

関数の発行が可能な処理プログラムの種別を示しています。

タスク	：タスクからだけ発行可能
非タスク	：非タスクからだけ発行可能
タスク｜非タスク	：タスク，非タスクのどちらからも発行可能
－	：割り込みハンドラまたはタスクのため，関数コールできない

③ 概要

関数の機能概要を示しています。

④ C言語形式

関数をC言語で記述された処理プログラムから発行するときの記述形式を示しています。

⑤ パラメータ

関数のパラメータを次の形式で示しています。

I/O	パラメータ	説　明
A	B	C

A ：パラメータの種類

I …USBファンクション・コントローラへの入力パラメータ

O…USBファンクション・コントローラからの出力パラメータ

B ：パラメータのデータ・タイプ

C ：パラメータの説明

⑥ 機能

関数の機能詳細を示しています。

⑦ 戻り値

関数からの戻り値を，データ・マクロおよび数値で示しています。

usbf850_init　　発行有効範囲：非タスク｜タスク

[概　　要]

V850E/ME2に内蔵しているUSBファンクション・コントローラの初期化処理を行う。

[C言語形式]

```
void usbf850_init (void)
```

[パラメータ]

I/O	パラメータ	説　　明
–	–	–

[機　　能]

ソフトウエア初期化部から呼び出され，V850E/ME2に内蔵しているUSBファンクション・コントローラの初期化処理を行います。

備考　初期化処理についての詳細は，3. 7. 2（1）**初期化処理**を参照してください。

[戻 り 値]

なし

usbf850_inthdr

発行有効範囲：－

[概　　要]

V850E/ME2に内蔵しているUSBファンクション・コントローラ用割り込みハンドラ（INTUSB0B信号用）。

[C言語形式]

```
ID usbf850_inthdr (void)
```

[パラメータ]

I/O	パラメータ	説　　明
－	－	－

[機　　能]

INTUSB0B信号（USBファンクション・ステータス0）により起動される割り込みハンドラです。

サンプル・プログラムでは，割り込み要因を調べCPUDEC割り込みだった場合だけ，割り込み処理用のタスク（task_usb0b）を起動します。DMAED割り込みまたはSHORT割り込みだった場合は，さらにUF0DMS1レジスタ（DMAステータス1レジスタ）を読み出し割り込み要因を調べ，該当するタスク（usbf850_data_inとusbf850_data_outはDMA起動後スリープ状態）を起床させます。このハンドラは，CF定義ファイルで定義されています。

備考　割り込み処理についての詳細は，3. 7. 2（2）割り込み処理を参照してください。

[戻 り 値]

オブジェクトID番号（タスクのID番号）

usbf850_inthdr1

発行有効範囲：－

[概　　要]

V850E/ME2に内蔵しているUSBファンクション・コントローラ用割り込みハンドラ（INTUSB1B信号用）。

[C言語形式]

```
ID usbf850_inthdr1 (void)
```

[パラメータ]

I/O	パラメータ	説　　明
－	－	－

[機　　能]

INTUSB1B信号（USBファンクション・ステータス1）により起動される割り込みハンドラです。

サンプル・プログラムでは，割り込み処理用のタスク(task_usb1b)を起動します。このハンドラは，CF定義ファイルで定義されています。

備考　割り込み処理についての詳細は，3. 7. 2（2）**割り込み処理**を参照してください。

[戻 り 値]

オブジェクトID番号（タスクのID番号）

usbf850_inthdr2　　発行有効範囲：－

[概　　要]

V850E/ME2に内蔵しているUSBファンクション・コントローラ用割り込みハンドラ（INTUSB2B信号用）。

[C言語形式]

```
ID usbf850_inthdr2 (void)
```

[パラメータ]

I/O	パラメータ	説　　明
－	－	－

[機　　能]

INTUSB2B信号（USBファンクション・ステータス2）により起動される割り込みハンドラです。

サンプル・プログラムでは，割り込み処理用のタスク（task_usb2b）を起動します。このハンドラは，CF定義ファイルで定義されています。

注意　このサンプル・プログラムでは，INTUSB2B信号により通知される割り込みについては，すべてマスクされているため，このハンドラは呼ばれません。

[戻 り 値]

オブジェクトID番号（タスクのID番号）

task_usb0b　　発行有効範囲：－

[概　　要]

INTUSB0B信号による割り込み処理を行う。

[C言語形式]

```
void task_usb0b (VP exinf)
```

[パラメータ]

I/O	パラメータ	説　　明
I	VP exinf	拡張情報

対象タスクに関する“ユーザ独自の情報”を格納するための領域で，ユーザが自由に利用できます。
exinfに設定された情報は，処理プログラム（タスク，非タスク）からref_tskシステム・コールを発行することにより，ダイナミックに獲得できます。

備考　システム・コールについての詳細は，RX850 Pro　**ユーザーズ・マニュアル　基礎編**を参照してください。

[機　　能]

INTUSB0B割り込み信号（USBファンクション・ステータス0割り込み）用の割り込みハンドラから起動されるタスクです。サンプル・プログラムでは，usbf850_rxreq関数を呼び出し，USB標準デバイス・リクエストおよびデバイス・クラス固有のリクエストの処理を行います。

注意　このサンプル・プログラムでは，V850E/ME2に内蔵しているUSBファンクション・コントローラで自動応答しない標準デバイス・リクエスト「Get Descriptor（String Descriptor）」だけを処理します。

備考　割り込み処理についての詳細は，3. 7. 2（2）**割り込み処理**を参照してください。

[戻 り 値]

なし

task_usb1b　　発行有効範囲：－

[概　　要]

INTUSB1B信号による割り込み処理を行う。

[C言語形式]

```
void task_usb1b (VP exinf)
```

[パラメータ]

I/O	パラメータ	説　　明
I	VP exinf	拡張情報

対象タスクに関する“ユーザ独自の情報”を格納するための領域で，ユーザが自由に利用できます。
exinfに設定された情報は，処理プログラム（タスク，非タスク）からref_tskシステム・コールを発行することにより，ダイナミックに獲得できます。

備考　システム・コールについての詳細は，RX850 Pro　**ユーザーズ・マニュアル　基礎編**を参照してください。

[機　　能]

INTUSB1B割り込み信号（USBファンクション・ステータス1割り込み）用の割り込みハンドラから起動されるタスクです。サンプル・プログラムでは，割り込み要因を確認し，割り込み要因がBKO1DTで受信データ長がCBWのデータ・サイズに等しければ，usbf850_rx_cbw関数を呼び出します。

備考　割り込み処理についての詳細は，3. 7. 2（2）**割り込み処理**を参照してください。

[戻 り 値]

なし

task_usb2b　　発行有効範囲：－

[概　　要]

INTUSB2B信号による割り込み処理を行う。

[C言語形式]

```
void task_usb2b (VP exinf)
```

[パラメータ]

I/O	パラメータ	説　　明
I	VP exinf	拡張情報

対象タスクに関する“ユーザ独自の情報”を格納するための領域で，ユーザが自由に利用できます。
exinfに設定された情報は，処理プログラム（タスク，非タスク）からref_tskシステム・コールを発行することにより，ダイナミックに獲得できます。

備考　システム・コールについての詳細は，RX850 Pro　**ユーザーズ・マニュアル　基礎編**を参照してください。

[機　　能]

INTUSB2B割り込み信号（USBファンクション・ステータス2割り込み）用の割り込みハンドラから起動されるタスクです。サンプル・プログラムでは該当する処理がないため，未処理で戻ります。

注意　このサンプル・プログラムでは，この関数を使用していません。

[戻 り 値]

なし

usbf850_data_send　　発行有効範囲：非タスク｜タスク

[概　　要]

USBファンクション・コントローラ用データ送信関数。

[C言語形式]

```
long usbf850_data_send (unsigned char* data, long len, char ep)
```

[パラメータ]

I/O	パラメータ	説　　明
I	unsigned char* data	送信データの先頭アドレス
I	long len	データ・サイズ
I	char ep	エンドポイント番号

[機　　能]

dataで指定されたアドレスから，lenで指定されたサイズ分のデータを，epで指定されたエンドポイントで送信処理を行います。

[戻 り 値]

送信時のステータス

DEV_ERROR ：エンドポイント番号が不正

DEV_OK　　：正常終了

usbf850_data_receive　　発行有効範囲：非タスク｜タスク

[概　　要]

USBファンクション・コントローラ用データ受信関数。

[C言語形式]

```
long usbf850_data_receive (unsigned char* data, long len, char ep)
```

[パラメータ]

I/O	パラメータ	説　　明
I	unsigned char* data	受信データ用バッファの先頭アドレス
I	long len	データ・サイズ
I	char ep	エンドポイント番号

[機　　能]

epで指定されたエンドポイントのバッファから，lenで指定されたサイズ分データを読み出し，dataで指定されたアドレスに格納します。

[戻 り 値]

受信時のステータス

DEV_ERROR ：受信データ・サイズが不正，またはエンドポイント番号が不正

DEV_OK　　：正常終了

usbf850_sendnullEP0

発行有効範囲：非タスク｜タスク

[概　　要]

コントロール・エンドポイント（エンドポイント0）用Nullデータ送信関数。

[C言語形式]

```
void usbf850_sendnullEP0 (void)
```

[パラメータ]

I/O	パラメータ	説　　明
–	–	–

[機　　能]

コントロール・エンドポイント（エンドポイント0）で，Nullデータ（データ・サイズ0のデータ）の送信処理を行います。

[戻 り 値]

なし

usb850_sendstallEP0

発行有効範囲：非タスク｜タスク

[概　　要]

コントロール・エンドポイント（エンドポイント0）用STALL応答関数。

[C言語形式]

```
void usbf850_sendstallEP0 (void)
```

[パラメータ]

I/O	パラメータ	説　　明
–	–	–

[機　　能]

コントロール・エンドポイント（エンドポイント0）でSTALL応答を設定します。

[戻 り 値]

なし

usbf850_bulkin1_stall　　発行有効範囲：非タスク｜タスク

[概　　要]

バルク・エンドポイント（エンドポイント1）用STALL応答関数。

[C言語形式]

```
void usbf850_bulkin1_stall (void)
```

[パラメータ]

I/O	パラメータ	説　　明
–	–	–

[機　　能]

バルク・エンドポイント（エンドポイント1）でSTALL応答を設定します。

[戻 り 値]

なし

usbf850_bulkout1_stall

発行有効範囲：非タスク｜タスク

[概　　要]

バルク・エンドポイント（エンドポイント2）用STALL応答関数。

[C言語形式]

```
void usbf850_bulkout1_stall (void)
```

[パラメータ]

I/O	パラメータ	説　　明
–	–	–

[機　　能]

バルク・エンドポイント（エンドポイント2）でSTALL応答を設定します。

[戻 り 値]

なし

usbf850_loc_cpu　　発行有効範囲：タスク

[概　　要]

マスカブル割り込みの受け付けとディスパッチ処理を禁止する。

[C言語形式]

```
void usbf850_loc_cpu (void)
```

[パラメータ]

I/O	パラメータ	説　　明
–	–	–

[機　　能]

loc_cpuシステム・コールを呼び出します。

備考　システム・コールについての詳細は，RX850 Pro　**ユーザーズ・マニュアル　基礎編**を参照してください。

[戻 り 値]

なし

usbf850_unl_cpu　　発行有効範囲：タスク

[概　　要]

マスカブル割り込みの受け付けとディスパッチ処理を許可する。

[C言語形式]

```
void usbf850_unl_cpu (void)
```

[パラメータ]

I/O	パラメータ	説　　明
–	–	–

[機　　能]

unl_cpuシステム・コールを呼び出します。

備考　システム・コールについての詳細は，RX850 Pro　**ユーザーズ・マニュアル　基礎編**を参照してください。

[戻 り 値]

なし

usbf850_rxreq

発行有効範囲：非タスク｜タスク

[概　　要]

USBリクエスト処理を行う。

[C言語形式]

```
void usbf850_rxreq (void)
```

[パラメータ]

I/O	パラメータ	説　　明
–	–	–

[機　　能]

INTUSB0B割り込み信号により起動されるtask_usb0bタスクによって，呼び出されます。SETUPデータの読み出し処理を呼び出し，読み出したデータを解析します。解析結果に基づき，USBのリクエスト処理を呼び出します。

[戻 り 値]

なし

usbf850_rxreq_read

発行有効範囲：非タスク｜タスク

[概　　要]

USBリクエスト・データを読み出す。

[C言語形式]

```
void usbf850_rxreq_read (void)
```

[パラメータ]

I/O	パラメータ	説　　明
–	–	–

[機　　能]

コントロール転送（エンドポイント0）で，Setupトークンに続いて受信されるSETUPデータを読み出します。

SETUPデータは，通常のデータと区別され専用のレジスタに格納されており，必ず8バイト・リードします。

[戻 り 値]

なし

usbf850_standardreq

発行有効範囲：非タスク｜タスク

[概　　要]

USB標準リクエストの処理を行う。

[C言語形式]

```
void usbf850_standardreq (void)
```

[パラメータ]

I/O	パラメータ	説　　明
–	–	–

[機　　能]

SETUPデータ読み出し後，リクエストの内容が標準リクエストであった場合に呼び出されます。Get Descriptor リクエストであることを確認し，usbf850_getdesc関数を呼び出します。

[戻 り 値]

なし

usbf850_getdesc

発行有効範囲：非タスク｜タスク

[概　　要]

USB標準リクエストのGet Descriptor（String Descriptor）の処理を行う。

[C言語形式]

```
void usbf850_getdesc (void)
```

[パラメータ]

I/O	パラメータ	説　　明
−	−	−

[機　　能]

usbf850_standardreq関数から呼び出され，USB標準リクエストのGet Descriptor（String Descriptor）の処理を行います。Get Descriptor（String Descriptor）リクエスト以外の場合は，STALL応答します。

[戻 り 値]

なし

usbf850_sstall_ctrl

発行有効範囲：非タスク｜タスク

[概　　要]

コントロール・エンドポイント（エンドポイント0）用STALL応答処理関数。

[C言語形式]

```
void usbf850_sstall_ctrl (void)
```

[パラメータ]

I/O	パラメータ	説　　明
–	–	–

[機　　能]

コントロール・エンドポイント（エンドポイント0）でSTALL応答を設定します。

usbf850_setfunction_storage関数で，クラス・リクエスト処理関数を関数ポインタとして配列に準備するときに，配列の添え字にリクエスト・コードを使用します。該当するリクエストがない部分にこの関数を登録することで，サポートしないリクエスト・コードが来た場合にSTALL応答するよう設定されます。

[戻 り 値]

なし

usbf850_blkonly_mass_storage_reset　　発行有効範囲：非タスク｜タスク

[概　　要]

USB Mass Storageクラス固有のリクエスト（Bulk-Only Mass Storage Reset）処理関数。

[C言語形式]

```
void usbf850_blkonly_mass_storage_reset (void)
```

[パラメータ]

I/O	パラメータ	説　　明
–	–	–

[機　　能]

Bulk-Only Mass Storage Resetリクエストの処理を行います。このリクエストを受け取ると，ストレージ・デバイスの初期化処理を行います。サンプル・プログラムでは，バルク・エンドポイント（エンドポイント番号1, 2）のバッファをクリアし，STALL応答を設定します。

[戻 り 値]

なし

usbf850_max_lun　　発行有効範囲：非タスク｜タスク

[概　　要]

USB Mass Storageクラス固有のリクエスト（Get Max LUN）処理関数。

[C言語形式]

```
void usbf850_max_lun (void)
```

[パラメータ]

I/O	パラメータ	説　　明
–	–	–

[機　　能]

Get Max LUNリクエストの処理を行います。このリクエストを受け取ると，デバイスがサポートする論理ユニットの総数を，1バイトのデータで返します。サンプル・プログラムでは，ストレージ・デバイスが仮想デバイスであるため，00Hをデータとして用意し，コントロール・エンドポイント（エンドポイント番号0）で送信します。

[戻 り 値]

なし

usbf850_setfunction_storage　　発行有効範囲：非タスク｜タスク

[概　　要]

USB Mass Storageクラス固有のリクエスト処理関数を関数ポインタとして配列に登録する処理を行う。

[C言語形式]

```
void usbf850_setfunction_storage (void)
```

[パラメータ]

I/O	パラメータ	説　　明
–	–	–

[機　　能]

USBの初期化処理から呼ばれ，USB Mass Storageクラス固有のリクエスト処理関数を関数ポインタとして，配列（配列名：Req_Func_C）に登録します。

サポートしないリクエスト・コードには，usbf850_sstall_ctrl関数を登録し，サポートしないリクエストが来た場合にSTALL応答するよう設定します。

[戻 り 値]

なし

usbf850_rx_cbw

発行有効範囲：非タスク｜タスク

[概　　要]

CBWデータの受信処理関数。

[C言語形式]

```
void usbf850_rx_cbw (void)
```

[パラメータ]

I/O	パラメータ	説　　明
–	–	–

[機　　能]

割り込み処理用タスクから呼び出され，CBWデータを読み出します。その後，usbf850_storage_cbwchk関数を呼び出します。

備考　CBWの受信処理についての詳細は，3. 7. 2（3）CBW**データの処理**を参照してください。

[戻 り 値]

なし

usbf850_storage_cbwchk　　発行有効範囲：非タスク｜タスク

[概　　要]

CBWデータのコマンド解析処理関数。

[C言語形式]

```
int usbf850_storage_cbwchk (void)
```

[パラメータ]

I/O	パラメータ	説　　明
–	–	–

[機　　能]

読み出したCBWデータを解析し，該当するデータ方向の処理タスクを起動します。

備考　CBWの受信処理についての詳細は，3. 7. 2（3）CBW**データの処理**を参照してください。

[戻 り 値]

CBWチェック時のステータス

DEV_ERROR ：CBWCBのレングスが不正

DEV_OK　　：正常終了

usbf850_cbw_error　　発行有効範囲：非タスク｜タスク

[概　　要]

CBWデータのエラー処理関数。

[C言語形式]

```
void usbf850_cbw_error (void)
```

[パラメータ]

I/O	パラメータ	説　　明
–	–	–

[機　　能]

CBWのエラー検出時，バルク・エンドポイント（エンドポイント番号1, 2）に対してSTALL応答を設定します。

[戻 り 値]

なし

usbf850_no_data　　発行有効範囲：非タスク｜タスク

[概　　要]

SCSIのNO DATA系コマンド処理タスク。

[C言語形式]

```
void usbf850_no_data (VP exinf)
```

[パラメータ]

I/O	パラメータ	説　　明
I	VP exinf	拡張情報

対象タスクに関する“ユーザ独自の情報”を格納するための領域で，ユーザが自由に利用できます。
exinfに設定された情報は，処理プログラム（タスク，非タスク）からref_tskシステム・コールを発行することにより，ダイナミックに獲得できます。

備考　システム・コールについての詳細は，RX850 Pro　**ユーザーズ・マニュアル　基礎編**を参照してください。

[機　　能]

SCSIのNO DATA系コマンド処理用のタスクです。scsi_command_to_ata関数を呼び出し，実行結果からCSW応答処理関数にCBWの処理状態（GOOD, FAIL, PHASE）を渡します。

備考　SCSIコマンドの処理についての詳細は，3. 7. 2（4）SCSI**コマンドの処理**を参照してください。

[戻 り 値]

なし

usbf850_data_in　　発行有効範囲：非タスク｜タスク

[概　　要]

SCSIのDATA IN系コマンド処理タスク。

[C言語形式]

```
void usbf850_data_in (VP exinf)
```

[パラメータ]

I/O	パラメータ	説　　明
I	VP exinf	拡張情報

対象タスクに関する“ユーザ独自の情報”を格納するための領域で，ユーザが自由に利用できます。
exinfに設定された情報は，処理プログラム（タスク，非タスク）からref_tskシステム・コールを発行することにより，ダイナミックに獲得できます。

備考　システム・コールについての詳細は，RX850 Pro　**ユーザーズ・マニュアル　基礎編**を参照してください。

[機　　能]

SCSIのDATA IN系コマンド処理用のタスクです。scsi_command_to_ata関数を呼び出し，実行結果からCSW応答処理関数にCBWの処理状態（GOOD, FAIL, PHASE）を渡します。

備考　SCSIコマンドの処理についての詳細は，3. 7. 2（4）SCSI**コマンドの処理**を参照してください。

[戻 り 値]

なし

usbf850_data_out　　発行有効範囲：非タスク｜タスク

[概　　要]

SCSIのDATA OUT系コマンド処理タスク。

[C言語形式]

```
void usbf850_data_out (VP exinf)
```

[パラメータ]

I/O	パラメータ	説　　明
I	VP exinf	拡張情報

対象タスクに関する“ユーザ独自の情報”を格納するための領域で，ユーザが自由に利用できます。
exinfに設定された情報は，処理プログラム（タスク，非タスク）からref_tskシステム・コールを発行することにより，ダイナミックに獲得できます。

備考　システム・コールについての詳細は，RX850 Pro　**ユーザーズ・マニュアル　基礎編**を参照してください。

[機　　能]

SCSIのDATA OUT系コマンド処理用のタスクです。scsi_command_to_ata関数を呼び出し，実行結果からCSW応答処理関数にCBWの処理状態（GOOD, FAIL, PHASE）を渡します。

備考　SCSIコマンドの処理についての詳細は，3. 7. 2（4）SCSI**コマンドの処理**を参照してください。

[戻 り 値]

なし

usbf850_csw_ret　　発行有効範囲：非タスク｜タスク

[概　　要]

CSW応答処理関数。

[C言語形式]

```
long usbf850_csw_ret (BYTE status)
```

[パラメータ]

I/O	パラメータ	説　　明
I	BYTE status	送信する状態（GOOD, FAIL, PHASE）

[機　　能]

CSW応答処理用の関数です。 引き数で渡されたCBWの処理状態（GOOD, FAIL, PHASE）をホストに送信します。

[戻 り 値]

送信時のステータス

DEV_OK　　：正常終了

usbf850_dma_init　　発行有効範囲：非タスク｜タスク

[概　　要]

DMA初期化関数。

[C言語形式]

```
void usbf850_dma_init (char ep)
```

[パラメータ]

I/O	パラメータ	説　　明
I	char ep	DMAを使用するエンドポイント番号

[機　　能]

引き数で指定されたエンドポイントに対して，使用するDMAの初期化処理を行います。

[戻 り 値]

なし

usbf850_dma_start　　発行有効範囲：非タスク｜タスク

[概　　要]

DMA起動関数。

[C言語形式]

```
void usbf850_dma_start (unsigned char* data, long len, char ep)
```

[パラメータ]

I/O	パラメータ	説　　明
I	unsigned char* data	送信データ，受信データを格納するバッファのポインタ
I	long len	データ長
I	char ep	DMAを使用するエンドポイント番号

[機　　能]

epが1ならば，lenで指定された長さ分，dataで指定されたバッファからDMAでUF0BI1レジスタにデータを転送します。

epが2ならば，lenで指定された長さ分，dataで指定されたバッファにUF0BO1レジスタからDMAでデータを読み出します。

[戻 り 値]

なし

storageDev_Init　　発行有効範囲：非タスク｜タスク

[概　　要]

ストレージ・デバイス初期化処理関数。

[C言語形式]

```
void storageDev_Init (void)
```

[パラメータ]

I/O	パラメータ	説　　明
–	–	–

[機　　能]

ストレージ・デバイスの初期化処理を行います。サンプル・プログラムでは，メモリ上にストレージ用の領域を確保しただけの仮想デバイスを扱うため，確保したメモリ領域を0クリアするだけです。

[戻 り 値]

なし

scsi_command_to_ata

発行有効範囲：非タスク｜タスク

[概　　要]

SCSIコマンド処理関数。

[C言語形式]

```
long scsi_command_to_ata
          ( BYTE *ScsiCommandBuf, BYTE *pbData, long lDataSize, long TransFlag )
```

[パラメータ]

I/O	パラメータ	説　　明
I	BYTE *ScsiCommandBuf	SCSIプロトコルの時のCBWCB
I	BYTE *pbData	各エンドポイント用データ・レジスタのアドレス
I	long lDataSize	送信／受信データ・サイズ
I	long TransFlag	転送方向

[機　　能]

CBWで通知されたSCSIコマンドから，コマンド処理関数を呼び出します。

[戻 り 値]

コマンド処理時のステータス

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で上記3つのステータス以外，またはリクエストが不正
DEV_OK　　：正常終了

ata_test_unit_ready　　発行有効範囲：非タスク｜タスク

[概　　要]

TEST UNIT READYコマンド処理関数。

[C言語形式]

```
long ata_test_unit_ready (long TransFlag)
```

[パラメータ]

I/O	パラメータ	説　　明
I	long TransFlag	データ転送方向

[機　　能]

TEST UNIT READYコマンドの処理を行います。サンプル・プログラムでは，仮想デバイスを扱うため，何もせずOKを返して終了します。

備考　SCSIコマンドの処理についての詳細は，3. 7. 2（4）SCSIコマンドの処理を参照してください。

[戻 り 値]

コマンド処理時のステータス

DEV_ERR_NODATA　：NO DATA系コマンドで転送方向エラー

DEV_OK　　：正常終了

ata_seek	発行有効範囲：非タスク｜タスク

[概　　要]

SEEKコマンド処理関数。

[C言語形式]

```
long ata_seek (long TransFlag)
```

[パラメータ]

I/O	パラメータ	説　　明
I	long TransFlag	データ転送方向

[機　　能]

SEEKコマンドの処理を行います。サンプル・プログラムでは，仮想デバイスを扱うため，何もせずOKを返して終了します。

備考　SCSIコマンドの処理についての詳細は，3. 7. 2（4）SCSI**コマンドの処理**を参照してください。

[戻 り 値]

コマンド処理時のステータス

DEV_ERR_NODATA　：NO DATA系コマンドで転送方向エラー

DEV_OK　　：正常終了

ata_start_stop_unit

発行有効範囲：非タスク｜タスク

[概　　要]

START STOP UNITコマンド処理関数。

[C言語形式]

```
long ata_start_stop_unit (long TransFlag)
```

[パラメータ]

I/O	パラメータ	説　　明
I	long TransFlag	データ転送方向

[機　　能]

START STOP UNITコマンドの処理を行います。サンプル・プログラムでは，仮想デバイスを扱うため，何もせずOKを返して終了します。

備考 SCSIコマンドの処理についての詳細は，3. 7. 2（4）SCSIコマンドの処理を参照してください。

[戻 り 値]

コマンド処理時のステータス

DEV_ERR_NODATA ：NO DATA系コマンドで転送方向エラー

DEV_OK ：正常終了

ata_synchronize_cache　　発行有効範囲：非タスク｜タスク

[概　　要]

SYNCHRONIZE CACHEコマンド処理関数。

[C言語形式]

```
long ata_synchronize_cache (long TransFlag)
```

[パラメータ]

I/O	パラメータ	説　　明
I	long TransFlag	データ転送方向

[機　　能]

SYNCHRONIZE CACHEコマンドの処理を行います。サンプル・プログラムでは，仮想デバイスを扱うため，何もせずOKを返して終了します。

備考　SCSIコマンドの処理についての詳細は，3. 7. 2（4）SCSIコマンドの処理を参照してください。

[戻 り 値]

コマンド処理時のステータス

DEV_ERR_NODATA　：NO DATA系コマンドで転送方向エラー

DEV_OK　　：正常終了

ata_request_sense　　発行有効範囲：非タスク｜タスク

[概　　要]

REQUEST SENSEコマンド処理関数。

[C言語形式]

```
long ata_request_sense
          (BYTE *ScsiCommandBuf, BYTE *pbData, long lDataSize, long TransFlag)
```

[パラメータ]

I/O	パラメータ	説　　明
I	BYTE *ScsiCommandBuf	SCSIプロトコルの時のCBWCB
I	BYTE *pbData	各エンドポイント用データ・レジスタのアドレス
I	long lDataSize	送信／受信データ・サイズ
I	long TransFlag	データ転送方向

[機　　能]

REQUEST SENSEコマンド処理を行います。サンプル・プログラムでは，仮想デバイスを扱うため，コマンドで指定された送信データ・サイズが0の場合は，何もせずOKを返して終了します。コマンドで指定されたデータ・サイズが0でない場合は，そのデータ・サイズ分，REQUEST SENSEデータを準備し送信します。用意したREQUEST SENSEデータのデータ長を越える場合は，用意したREQUEST SENSEデータのデータ長の分だけ送信します。

備考　SCSIコマンドの処理についての詳細は，3. 7. 2（4）SCSIコマンドの処理を参照してください。

[戻 り 値]

コマンド処理時のステータス

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_OK　　：正常終了

ata_inquiry　　発行有効範囲：非タスク｜タスク

[概　　要]

INQUIRYコマンド処理関数。

[C言語形式]

```
long ata_inquiry (BYTE *ScsiCommandBuf, BYTE *pbData, long lDataSize, long TransFlag)
```

[パラメータ]

I/O	パラメータ	説　　明
I	BYTE *ScsiCommandBuf	SCSIプロトコルの時のCBWCB
I	BYTE *pbData	各エンドポイント用データ・レジスタのアドレス
I	long lDataSize	送信／受信データ・サイズ
I	long TransFlag	データ転送方向

[機　　能]

INQUIRYコマンド処理を行います。サンプル・プログラムでは，コマンドで指定されたデータ・サイズ分，INQUIRYデータを準備し送信します。用意したINQUIRYデータのデータ長を越える場合は，用意したINQUIRYデータのデータ長の分だけ送信します。

備考　SCSIコマンドの処理についての詳細は，3. 7. 2（4）SCSIコマンドの処理を参照してください。

[戻 り 値]

コマンド処理時のステータス

DEV_ERR_READ　：READ系コマンドで転送方向エラー
DEV_ERROR　：リクエストが不正，またはscsi_to_usbの実行結果が異常終了
DEV_OK　：正常終了

ata_mode_select　　発行有効範囲：非タスク｜タスク

[概　　要]

MODE SELECT（6）コマンド処理関数。

[C言語形式]

```
long ata_mode_select
              (BYTE *ScsiCommandBuf, BYTE *pbData, long lDataSize, long TransFlag)
```

[パラメータ]

I/O	パラメータ	説　　明
I	BYTE *ScsiCommandBuf	SCSIプロトコルの時のCBWCB
I	BYTE *pbData	各エンドポイント用データ・レジスタのアドレス
I	long lDataSize	送信／受信データ・サイズ
I	long TransFlag	データ転送方向

[機　　能]

MODE SELECT（6）コマンド処理を行います。サンプル・プログラムでは，指定されたデータ・サイズ分，MODE SELECTテーブルにデータを読み込みます。用意したMODE SELECTテーブルのデータ長を越える場合は，用意したMODE SELECTテーブルのデータ長の分だけ読み込みます。

備考　SCSIコマンドの処理についての詳細は，3. 7. 2（4）SCSIコマンドの処理を参照してください。

[戻 り 値]

コマンド処理時のステータス

DEV_ERR_WRITE	：WRITE系コマンドで転送方向エラー
DEV_ERROR	：CDBの内容が不正
DEV_OK	：正常終了

ata_mode_select10　　発行有効範囲：非タスク｜タスク

[概　　要]

MODE SELECT（10）コマンド処理関数。

[C言語形式]

```
long ata_mode_select10
              (BYTE *ScsiCommandBuf, BYTE *pbData, long lDataSize, long TransFlag)
```

[パラメータ]

I/O	パラメータ	説　　明
I	BYTE *ScsiCommandBuf	SCSIプロトコルの時のCBWCB
I	BYTE *pbData	各エンドポイント用データ・レジスタのアドレス
I	long lDataSize	送信／受信データ・サイズ
I	long TransFlag	データ転送方向

[機　　能]

MODE SELECT（10）コマンド処理を行います。サンプル・プログラムでは，指定されたデータ・サイズ分，MODE SELECT（10）テーブルにデータを読み込みます。用意したMODE SELECT（10）テーブルのデータ長を越える場合は，用意したMODE SELECT（10）テーブルのデータ長の分だけ読み込みます。

備考　SCSIコマンドの処理についての詳細は，3. 7. 2（4）SCSIコマンドの処理を参照してください。

[戻 り 値]

コマンド処理時のステータス

DEV_ERR_WRITE　：WRITE系コマンドで転送方向エラー
DEV_ERROR　：CDBの内容が不正
DEV_OK　：正常終了

ata_mode_sense　　発行有効範囲：非タスク｜タスク

[概　　要]

MODE SENSE（6）コマンド処理関数。

[C言語形式]

```
long ata_mode_sense
              (BYTE *ScsiCommandBuf, BYTE *pbData, long lDataSize, long TransFlag)
```

[パラメータ]

I/O	パラメータ	説　　明
I	BYTE *ScsiCommandBuf	SCSIプロトコルの時のCBWCB
I	BYTE *pbData	各エンドポイント用データ・レジスタのアドレス
I	long lDataSize	送信／受信データ・サイズ
I	long TransFlag	データ転送方向

[機　　能]

MODE SENSE (6) コマンド処理を行います。サンプル・プログラムでは，指定されたデータ・サイズ分，MODE SENSEデータを準備し送信します。用意したMODE SENSEデータのデータ長を越える場合は，用意したMODE SENSEデータのデータ長の分だけ読み込みます。

備考　SCSIコマンドの処理についての詳細は，3. 7. 2（4）SCSIコマンドの処理を参照してください。

[戻 り 値]

コマンド処理時のステータス

DEV_ERR_READ	：READ系コマンドで転送方向エラー
DEV_ERROR	：scsi_to_usbの実行結果が異常終了
DEV_OK	：正常終了

ata_mode_sense10　　発行有効範囲：非タスク｜タスク

[概　　要]

MODE SENSE（10）コマンド処理関数。

[C言語形式]

```
long ata_mode_sense10
              (BYTE *ScsiCommandBuf, BYTE *pbData, long lDataSize, long TransFlag)
```

[パラメータ]

I/O	パラメータ	説　　明
I	BYTE *ScsiCommandBuf	SCSIプロトコルの時のCBWCB
I	BYTE *pbData	各エンドポイント用データ・レジスタのアドレス
I	long lDataSize	送信／受信データ・サイズ
I	long TransFlag	データ転送方向

[機　　能]

MODE SENSE（10）コマンド処理を行います。サンプル・プログラムでは，指定されたデータ・サイズ分，MODE SENSE（10）データを準備し送信します。用意したMODE SENSE（10）データのデータ長を越える場合は，用意したMODE SENSE（10）データのデータ長の分だけ読み込みます。

備考　SCSIコマンドの処理についての詳細は，3. 7. 2（4）SCSIコマンドの処理を参照してください。

[戻 り 値]

コマンド処理時のステータス

DEV_ERR_READ　：READ系コマンドで転送方向エラー
DEV_ERROR　：scsi_to_usbの実行結果が異常終了
DEV_OK　：正常終了

ata_read_format_capacities　　発行有効範囲：非タスク｜タスク

［概　　要］

READ FORMAT CAPACITIESコマンド処理関数。

［C言語形式］

```
long ata_read_format_capacities
              (BYTE *ScsiCommandBuf, BYTE *pbData, long lDataSize, long TransFlag)
```

［パラメータ］

I/O	パラメータ	説　　明
I	BYTE *ScsiCommandBuf	SCSIプロトコルの時のCBWCB
I	BYTE *pbData	各エンドポイント用データ・レジスタのアドレス
I	long lDataSize	送信／受信データ・サイズ
I	long TransFlag	データ転送方向

［機　　能］

READ FORMAT CAPACITIESコマンド処理を行います。サンプル・プログラムでは，指定されたデータ・サイズ分，READ FORMAT CAPACITIESデータを準備し送信します。用意したREAD FORMAT CAPACITIESデータのデータ長を越える場合は，用意したREAD FORMAT CAPACITIESデータのデータ長の分だけ送信します。

備考　SCSIコマンドの処理についての詳細は，3. 7. 2（4）SCSIコマンドの処理を参照してください。

［戻 り 値］

コマンド処理時のステータス

DEV_ERR_READ　：READ系コマンドで転送方向エラー
DEV_ERROR　：scsi_to_usbの実行結果が異常終了
DEV_OK　：正常終了

ata_read_capacity　　発行有効範囲：非タスク｜タスク

[概　　要]

READ CAPACITYコマンド処理関数。

[C言語形式]

```
long ata_read_capacity
              (BYTE *ScsiCommandBuf, BYTE *pbData, long lDataSize, long TransFlag)
```

[パラメータ]

I/O	パラメータ	説　　明
I	BYTE *ScsiCommandBuf	SCSIプロトコルの時のCBWCB
I	BYTE *pbData	各エンドポイント用データ・レジスタのアドレス
I	long lDataSize	送信／受信データ・サイズ
I	long TransFlag	データ転送方向

[機　　能]

READ CAPACITYコマンド処理を行います。サンプル・プログラムでは，指定されたデータ・サイズ分，READ CAPACITYデータを準備し送信します。用意したREAD CAPACITYデータのデータ長を越える場合は，用意したREAD CAPACITYデータのデータ長の分だけ送信します。

備考　SCSIコマンドの処理についての詳細は，3. 7. 2（4）SCSIコマンドの処理を参照してください。

[戻 り 値]

コマンド処理時のステータス

DEV_ERR_READ	：READ系コマンドで転送方向エラー
DEV_ERROR	：scsi_to_usbの実行結果が異常終了
DEV_OK	：正常終了

ata_read6　　発行有効範囲：非タスク｜タスク

[概　　要]

READ（6）コマンド処理関数。

[C言語形式]

```
long ata_read6 (BYTE *ScsiCommandBuf, BYTE *pbData, long lDataSize, long TransFlag)
```

[パラメータ]

I/O	パラメータ	説　　明
I	BYTE *ScsiCommandBuf	SCSIプロトコルの時のCBWCB
I	BYTE *pbData	各エンドポイント用データ・レジスタのアドレス
I	long lDataSize	送信／受信データ・サイズ
I	long TransFlag	データ転送方向

[機　　能]

READ（6）コマンド処理を行います。指定された場所から指定されたサイズ分，ストレージ・デバイス上のデータを読み出します。サンプル・プログラムでは，仮想デバイス上のデータを読み出します。

備考　SCSIコマンドの処理についての詳細は，3. 7. 2（4）SCSIコマンドの処理を参照してください。

[戻 り 値]

コマンド処理時のステータス

DEV_ERR_READ　：READ系コマンドで転送方向エラー
DEV_ERROR　：CDBの内容が不正，またはscsi_to_usbの実行結果が異常終了
DEV_OK　：正常終了

ata_read10　　発行有効範囲：非タスク｜タスク

[概　　要]

READ（10）コマンド処理関数。

[C言語形式]

```
long ata_read10 (BYTE *ScsiCommandBuf, BYTE *pbData, long lDataSize, long TransFlag)
```

[パラメータ]

I/O	パラメータ	説　　明
I	BYTE *ScsiCommandBuf	SCSIプロトコルの時のCBWCB
I	BYTE *pbData	各エンドポイント用データ・レジスタのアドレス
I	long lDataSize	送信／受信データ・サイズ
I	long TransFlag	データ転送方向

[機　　能]

READ（10）コマンド処理を行います。指定された場所から指定されたサイズ分，ストレージ・デバイス上のデータを読み出します。サンプル・プログラムでは，仮想デバイス上のデータを読み出します。

備考　SCSIコマンドの処理についての詳細は，3. 7. 2（4）SCSIコマンドの処理を参照してください。

[戻 り 値]

コマンド処理時のステータス

DEV_ERR_READ　：READ系コマンドで転送方向エラー
DEV_ERROR　：CDBの内容が不正，またはscsi_to_usbの実行結果が異常終了
DEV_OK　：正常終了

ata_write6　　発行有効範囲：非タスク｜タスク

[概　　要]

WRITE（6）コマンド処理関数。

[C言語形式]

```
long ata_write6 (BYTE *ScsiCommandBuf, BYTE *pbData, long lDataSize, long TransFlag)
```

[パラメータ]

I/O	パラメータ	説　　明
I	BYTE *ScsiCommandBuf	SCSIプロトコルの時のCBWCB
I	BYTE *pbData	各エンドポイント用データ・レジスタのアドレス
I	long lDataSize	送信／受信データ・サイズ
I	long TransFlag	データ転送方向

[機　　能]

WRITE（6）コマンド処理を行います。指定された場所から指定されたサイズ分，ストレージ・デバイス上に受け取ったデータを書き込みます。サンプル・プログラムでは，仮想デバイス上にデータを書き込みます。

備考　SCSIコマンドの処理についての詳細は，3. 7. 2（4）SCSIコマンドの処理を参照してください。

[戻 り 値]

コマンド処理時のステータス

DEV_ERR_WRITE　：WRITE系コマンドで転送方向エラー
DEV_ERROR　：CDBの内容が不正
DEV_OK　：正常終了

ata_write10　　発行有効範囲：非タスク｜タスク

[概　　要]

WRITE（10）コマンド処理関数。

[C言語形式]

```
long ata_write10 (BYTE *ScsiCommandBuf, BYTE *pbData, long lDataSize, long TransFlag)
```

[パラメータ]

I/O	パラメータ	説　　明
I	BYTE *ScsiCommandBuf	SCSIプロトコルの時のCBWCB
I	BYTE *pbData	各エンドポイント用データ・レジスタのアドレス
I	long lDataSize	送信／受信データ・サイズ
I	long TransFlag	データ転送方向

[機　　能]

WRITE（10）コマンド処理を行います。指定された場所から指定されたサイズ分，ストレージ・デバイス上に受け取ったデータを書き込みます。サンプル・プログラムでは，仮想デバイス上にデータを書き込みます。

備考　SCSIコマンドの処理についての詳細は，3. 7. 2（4）SCSIコマンドの処理を参照してください。

[戻 り 値]

コマンド処理時のステータス

DEV_ERR_WRITE　：WRITE系コマンドで転送方向エラー
DEV_ERROR　：CDBの内容が不正
DEV_OK　：正常終了

ata_verify　　発行有効範囲：非タスク｜タスク

[概　　要]

VERIFYコマンド処理関数。

[C言語形式]

```
long ata_verify (BYTE *ScsiCommandBuf, BYTE *pbData, long lDataSize, long TransFlag)
```

[パラメータ]

I/O	パラメータ	説　　明
I	BYTE *ScsiCommandBuf	SCSIプロトコルの時のCBWCB
I	BYTE *pbData	各エンドポイント用データ・レジスタのアドレス
I	long lDataSize	送信／受信データ・サイズ
I	long TransFlag	データ転送方向

[機　　能]

VERIFYコマンド処理を行います。指定された場所から指定されたサイズ分，ストレージ・デバイス上のデータをチェックします。サンプル・プログラムでは，仮想デバイスを扱うため，データのチェックは行いません。

備考　SCSIコマンドの処理についての詳細は，3. 7. 2（4）SCSIコマンドの処理を参照してください。

[戻 り 値]

コマンド処理時のステータス

DEV_ERR_NODATA　：NO DATA系コマンドで転送方向エラー
DEV_ERROR　　：CDBの内容が不正
DEV_OK　　：正常終了

ata_write_verify　　発行有効範囲：非タスク｜タスク

[概　　要]

WRITE VERIFYコマンド処理関数。

[C言語形式]

```
long ata_write_verify
          (BYTE *ScsiCommandBuf, BYTE *pbData, long lDataSize, long TransFlag)
```

[パラメータ]

I/O	パラメータ	説　　明
I	BYTE *ScsiCommandBuf	SCSIプロトコルの時のCBWCB
I	BYTE *pbData	各エンドポイント用データ・レジスタのアドレス
I	long lDataSize	送信／受信データ・サイズ
I	long TransFlag	データ転送方向

[機　　能]

WRITE VERIFYコマンド処理を行います。指定された場所から指定されたサイズ分，ストレージ・デバイス上にデータを書き込み，書き込まれたデータが正しいかチェックします。サンプル・プログラムでは，仮想デバイスを扱うため，指定されたメモリ上にデータを書き込むだけで，データのチェックは行いません。

備考　SCSIコマンドの処理についての詳細は，3. 7. 2（4）SCSIコマンドの処理を参照してください。

[戻 り 値]

コマンド処理時のステータス

DEV_OK　　：正常終了

ata_write_buff　　発行有効範囲：非タスク｜タスク

[概　　要]

WRITE BUFFコマンド処理関数。

[C言語形式]

```
long ata_write_buff (BYTE *ScsiCommandBuf, BYTE *pbData, long lDataSize, long TransFlag)
```

[パラメータ]

I/O	パラメータ	説　　明
I	BYTE *ScsiCommandBuf	SCSIプロトコルの時のCBWCB
I	BYTE *pbData	各エンドポイント用データ・レジスタのアドレス
I	long lDataSize	送信／受信データ・サイズ
I	long TransFlag	データ転送方向

[機　　能]

WRITE BUFFコマンド処理を行います。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，仮想デバイスを扱うため，何も処理せず正常終了します。

備考　SCSIコマンドの処理についての詳細は，3. 7. 2（4）SCSIコマンドの処理を参照してください。

[戻 り 値]

コマンド処理時のステータス

DEV_OK　　：正常終了

scsi_to_usb　　発行有効範囲：非タスク｜タスク

[概　　要]

仮想デバイスからUSB方向への送信処理関数。

[C言語形式]

```
long scsi_to_usb (BYTE *pbData, long TransFlag)
```

[パラメータ]

I/O	パラメータ	説　　明
I	BYTE *pbData	送信データの先頭アドレス
I	long TransFlag	データ転送方向

[機　　能]

仮想デバイスからUSB方向へのデータの送信処理を行います。

[戻 り 値]

コマンド処理時のステータス

DEV_ERR_READ　：READ系コマンドで転送方向エラー

DEV_OK　：正常終了

USBF850REG_SET　　発行有効範囲：非タスク｜タスク

[概　　要]

V850E/ME2周辺I/Oレジスタ設定関数（1バイト単位：8ビット）。

[C言語形式]

```
USBF850REG_SET (offset, val)
```

[パラメータ]

I/O	パラメータ	説　　明
I	offset	周辺I/Oレジスタのアドレス
I	val	設定用データ

[機　　能]

V850E/ME2周辺I/Oレジスタ（offsetで指定されたレジスタ・アドレス）へ，valで指定されたデータを設定します。なお，このマクロは，1バイト（8ビット）単位でアクセス可能なレジスタだけ使用できます。

[戻 り 値]

なし

USBF850REG_READ

発行有効範囲：非タスク｜タスク

[概　　要]

V850E/ME2周辺I/Oレジスタ読み出し関数（1バイト単位：8ビット）。

[C言語形式]

```
USBF850REG_READ (offset)
```

[パラメータ]

I/O	パラメータ	説　　明
I	offset	周辺I/Oレジスタのアドレス

[機　　能]

V850E/ME2周辺I/Oレジスタ（offsetで指定されたレジスタ・アドレス）の値を読み出します。なお，このマクロは，1バイト（8ビット）単位でアクセス可能なレジスタだけ使用できます。

[戻 り 値]

なし

USBF850REG_SET_W　　発行有効範囲：非タスク｜タスク

[概　　要]

V850E/ME2周辺I/Oレジスタ設定関数（1ワード単位：16ビット）。

[C言語形式]

```
USBF850REG_SET_W (offset, val)
```

[パラメータ]

I/O	パラメータ	説　　明
I	offset	周辺I/Oレジスタのアドレス
I	val	設定用データ

[機　　能]

V850E/ME2周辺I/Oレジスタ（offsetで指定されたレジスタ・アドレス）へ，valで指定されたデータを設定します。なお，このマクロは，1ワード（16ビット）単位でアクセス可能なレジスタだけ使用できます。

[戻 り 値]

なし

USBF850REG_READ_W　　発行有効範囲：非タスク｜タスク

[概　　要]

V850E/ME2周辺I/Oレジスタ読み出し関数（1ワード単位：16ビット）。

[C言語形式]

```
USBF850REG_READ_W (offset)
```

[パラメータ]

I/O	パラメータ	説　　明
I	offset	周辺I/Oレジスタのアドレス

[機　　能]

V850E/ME2周辺I/Oレジスタ（offsetで指定されたレジスタ・アドレス）の値を読み出します。なお，このマクロは，1ワード（16ビット）単位でアクセス可能なレジスタだけ使用できます。

[戻 り 値]

なし

第4章　USBコミュニケーション・クラス用ドライバ

4.1 概　　説

4.1.1 概　　要

USBコミュニケーション・クラス用ドライバは，V850E/ME2に内蔵しているUSBファンクション・コントローラ用のサンプル・プログラムです。Universal Serial Bus Specification Revision 1.1に準拠しており，組み込み型制御用リアルタイム・オペレーティング・システムRX850 Pro（μITRON3.0仕様準拠）上で動作します。

このサンプル・プログラムでは，コントロール・エンドポイント（エンドポイント番号0），バルク・エンドポイントのINとOUT（エンドポイント番号3, 4），およびインタラプト・エンドポイントのIN（エンドポイント番号7）の4つを使用します。その上で，Windows XPに標準のコミュニケーション・クラス用ホスト・ドライバと接続し，仮想的なCOMポートとして動作します。クラスとしては，コミュニケーション・クラスが定義されています。

このサンプル・プログラムは，ハードウエアの実行環境として評価ボードSolutionGear MINI（SG-703111-1）を使用しています。SolutionGear MINIとサンプル・プログラムをそのまま使用する場合は，4. 6　**ロード・モジュール**の手順に従って実行オブジェクトを作成し，4. 2　**ロード・モジュールの実行**に従って動作を確認してください。

SolutionGear MINIからほかのターゲット・ボードに変更して使用する場合は，4. 3　**システム構築**，4. 4 RX850 Pro**依存処理プログラム**，4. 5　**セクション・マップ・ファイル**を参照し，ボードの仕様に従って変更してください。

SolutionGear MINIおよびサンプル・プログラムの両方を変更して使用する場合は，4. 3　**システム構築**，4. 4 RX850 Pro**依存処理プログラム**，4. 5　**セクション・マップ・ファイル**，4. 6　**ロード・モジュール**，4. 7　USB**ドライバの機能**を参照し，必要な変更を行ってください。

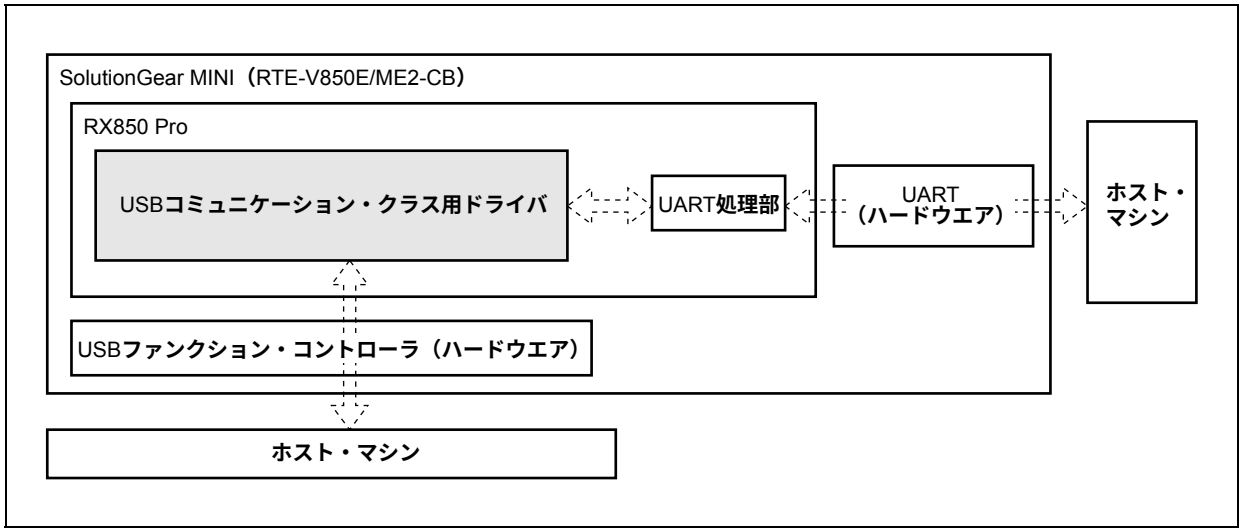
USBコミュニケーション・クラス用ドライバの位置付けを次に示します。

注意　Windows XP**標準のコミュニケーション・クラス用ホスト・ドライバは，正式にサポートされていないため，ドライバ・モジュールを呼び出すために**inf**ファイルを作成する必要があります。**inf**ファイルについては，**http://www.lvr.com/usbfaq.htm**の**USB Developers FAQ**にある**Device Classes**に説明がありますので，そちらを参照してください。**

備考1. USBコミュニケーション・クラスの詳細は，Universal Serial Bus Class Definitions for communication Devices Version 1.1を参照してください。

2. 4. 2. 1　**ロード・モジュールの実行手順**は，4. 1. 3　**実行環境**で示した環境を想定して記述しています。

図4－1　USBコミュニケーション・クラス用ドライバの位置付け



4. 1. 2　開発環境

サンプル・プログラムを使用してシステムを開発するうえで必要となるハードウエア環境およびソフトウエア環境として，次に示す環境を想定して記述しています。

・ハードウエア環境

ホスト・マシン	：PC/AT互換機（OS：Windows XP）

・ソフトウエア環境

リアルタイムOS	：RX850 Pro Version 3.15
USBコミュニケーション・クラス用ドライバ	：この章で説明するサンプル・プログラム一式
Cコンパイラ・パッケージ	：MULTI2000 （CCV850 Version 3.5（Green Hills Software, Inc.製））

注意　サンプルのビルド・ファイルで扱うディレクトリ構成と違う場合は，環境にあわせてビルド・ファイルを編集してください。

備考　ビルド・ファイルの記述内容については，MULTI（Green Hills Software, Inc.製）**のヘルプを参照してください。**

4.1.3　実行環境

サンプル・プログラムを使用したロード・モジュールを実行するときのハードウエア環境およびソフトウエア環境として，次に示す環境を想定して記述しています。

・ハードウエア環境

ホスト・マシン	：PC/AT互換機（OS：Windows XP）
IE制御用マシン	：PC/AT互換機（OS：Windows XP）
ターゲット・ボード	：SolutionGear MINI（SG-703111-1）
インサーキット・エミュレータ（IE）	：N-wire IE（RTE-2000-TP）（マイダス・ラボ社製）
JTAGプローブ	
USBケーブル	
RS-232Cケーブル	：SolutionGear MINI標準添付のRS-232Cケーブル

・ソフトウエア環境

IE用ソフトウエア	：PARTNER Setup Program Version 1.242

備考1. 実行環境のセットアップ方法についての詳細は，付録A　SG-703111-1**ボード**，および SG-703111-1　**ユーザーズ・マニュアル**を参照してください。

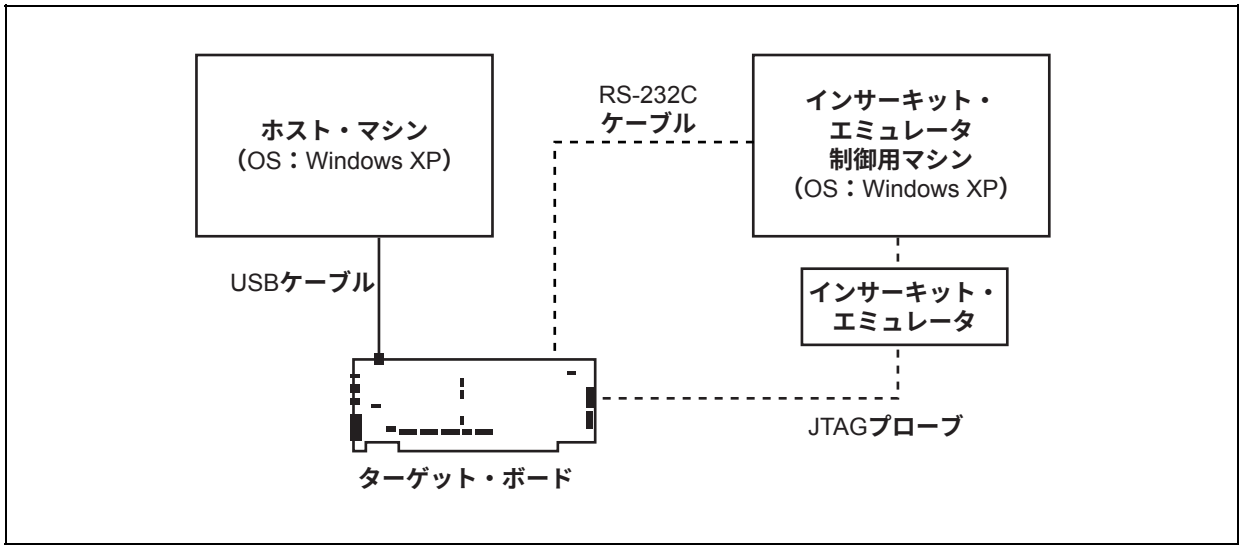
2. インサーキット・エミュレータ（RTE-2000-TP）のセットアップ方法についての詳細は，RTE-2000-TP　**ハードウエア・ユーザーズ・マニュアル**を参照してください。

3. PARTNERについての詳細は，PARTNER　**ユーザーズ・マニュアル**　V800**シリーズ「**V800**シリーズ共通編」，「**NB85E-CB**個別編」**を参照してください。

4. RS-232Cケーブルは，変換コネクタを使用して，ターゲット・ボード上のシリアル・コネクタJSIO2とUARTの対向側ホスト・マシンを接続します。適合コネクタについては，SG-703111-1　**ユーザーズ・マニュアル**を参照してください。

5. 図4－2では，UARTの対向側ホスト・マシンとして，IE制御用マシンを使用しています。

図4－2　実行環境



4. 2　ロード・モジュールの実行

4. 2. 1　ロード・モジュールの実行手順

このサンプル・プログラムを使用したロード・モジュールを例にとり，4. 1. 3　**実行環境**で示した環境において実行するときの手順を，次に示します。

① **インサーキット・エミュレータ（IE）制御用マシンの準備**

IE制御用マシンに電源を投入し起動しておきます。また，インサーキット・エミュレータにも電源を投入し準備しておきます。

② **ホスト・マシンの準備**

ホスト・マシンに電源を投入し起動しておきます（ホスト・マシンは，IE制御用マシンで兼用できますが，開発時には別のホスト・マシンを用意することを強く推奨します）。

③ SG-703111-1**ボードのリセット**

SG-703111-1**ボードのリセット・ボタン**RESET**を押下し，**SG-703111-1**ボードのリセットを行います。**

④ **インサーキット・エミュレータの起動**

インサーキット・エミュレータを起動します。

Windows**の「スタート」ボタンから「すべてのプログラム」－「**PARTNER**」－「**RPTSETUP (NB85ET)**」を選択します。**

[PARTNER**の起動例]**

次に示す画面が起動されます。

[PARTNERの起動画面]

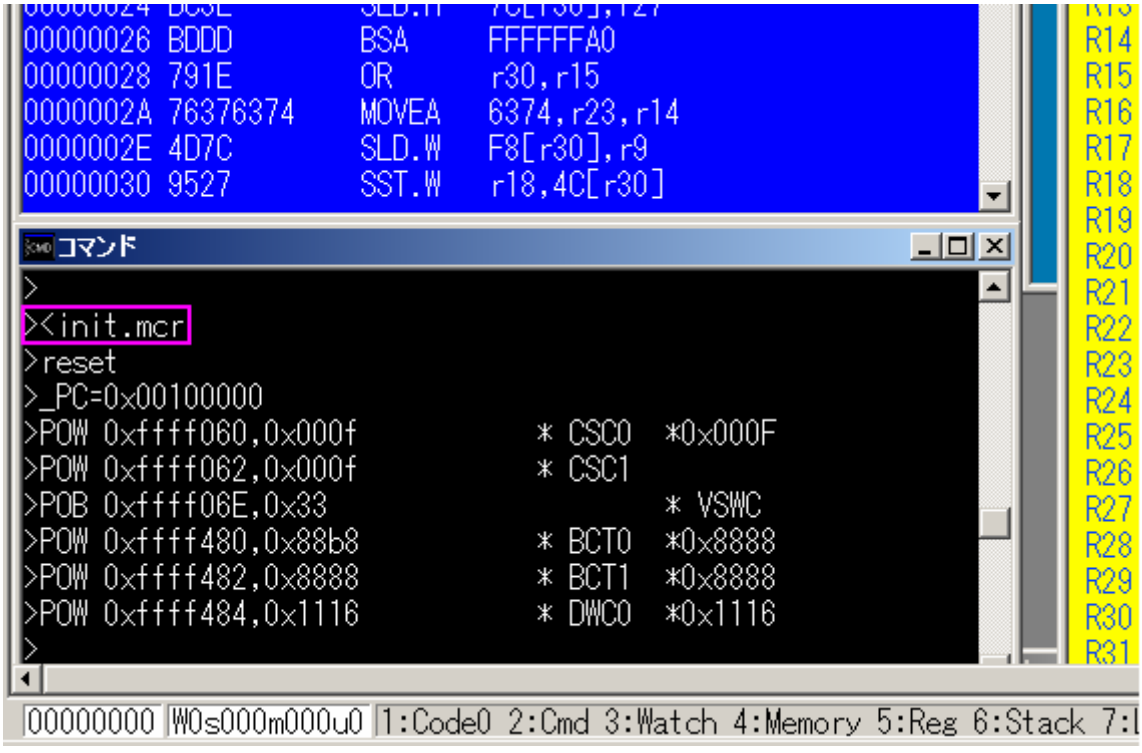
「開く」ボタンをクリックし，プロジェクト・ファイルを指定すると，「起動」ボタンが有効になります。次に，「起動」ボタンをクリックするとPARTNERが起動されます。起動後，ボードの設定を行います。このとき，起動時に読み込まれる設定ファイルをあらかじめ作成しておくと便利です。ここで説明するサンプル用の設定ファイルについては，**付録**A　SG-703111-1**ボード**，および，PARTNER　**ユーザーズ・マニュアル**　V800**シリーズ**「V800**シリーズ共通編**」，「NB85E-CB**個別編**」を参照してください。

注意1. **インサーキット・エミュレータの起動は，必ずターゲット・ボードの電源投入後に行ってください。**

2. **インサーキット・エミュレータの起動後，ターゲット・ボードのリセットのために設定ファイルを読み込みたい場合は，コマンド・ウィンドウから，次のコマンド入力例にならって設定ファイル（例では**init.mcr**）を読み込むことが出来ます。**

[コマンド入力例]

```
><init.mcr<Enter>
```



⑤　ロード・モジュールの読み込み

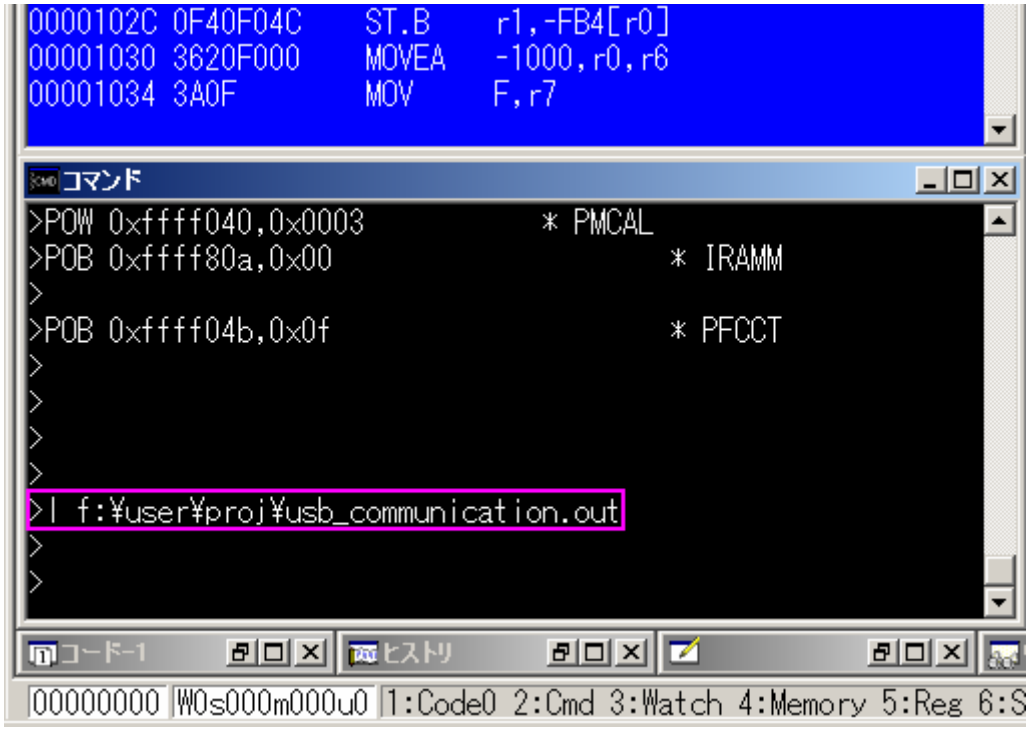
インサーキット・エミュレータの機能を使って，ロード・モジュールをボード上に読み込みます。

ツールバーの「ファイル」－「ロード」からロード・モジュールを読み込むか，コマンド・ウィンドウからL（ファイル読み込み）コマンドを使用してロード・モジュール（例ではusb_communication.out）を読み込みます。

[ファイルのロード例]

[コマンド入力例]

```
>l usb_communication.out<Enter>
```

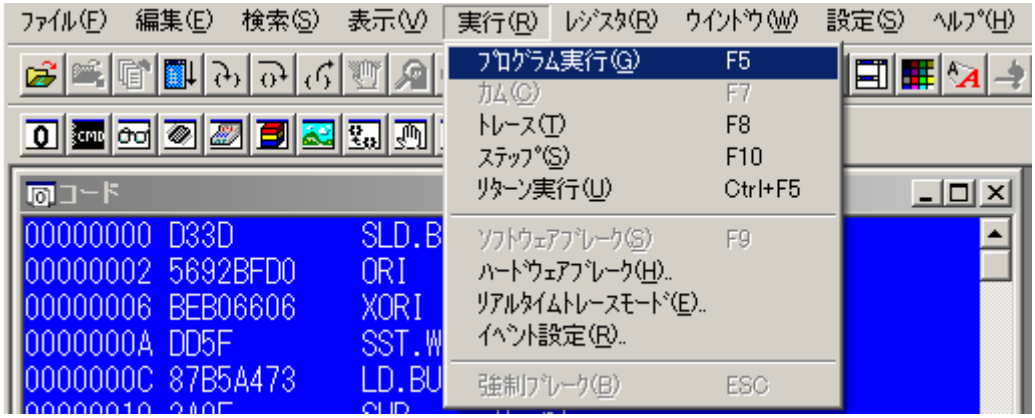


⑥　実行

F5キーまたは実行ボタンで，ボード上に読み込まれたコードを実行します。

注意　ツールバーの「実行」－「プログラムの実行」からも同じように実行できます。

[プログラム実行例]



⑦　USB接続

USBケーブルを接続します。

ボード側にBコネクタを接続し，ホスト・マシン側にAコネクタを接続します。

注意1.　ターゲット・ボードの起動前にUSBを接続しても問題ありません。

2.　ホスト・マシン側でデバイスが認識されると，次のようにソフトウエアのインストール画面が現れます。このサンプル・プログラムには，Windows XP標準のUSBコミュニケーション・クラス用ホスト・ドライバをインストールしてください。Windows XP標準のUSBコミュニケーション・クラス用ホスト・ドライバは，正式にサポートされていないため，ドライバ・モジュールを呼び出すためにinfファイルを作成する必要があります。infファイルについては，http://www.lvr.com/usbfaq.htmのUSB Developers FAQにあるDevice Classesに説明がありますので，そちらを参照してください。

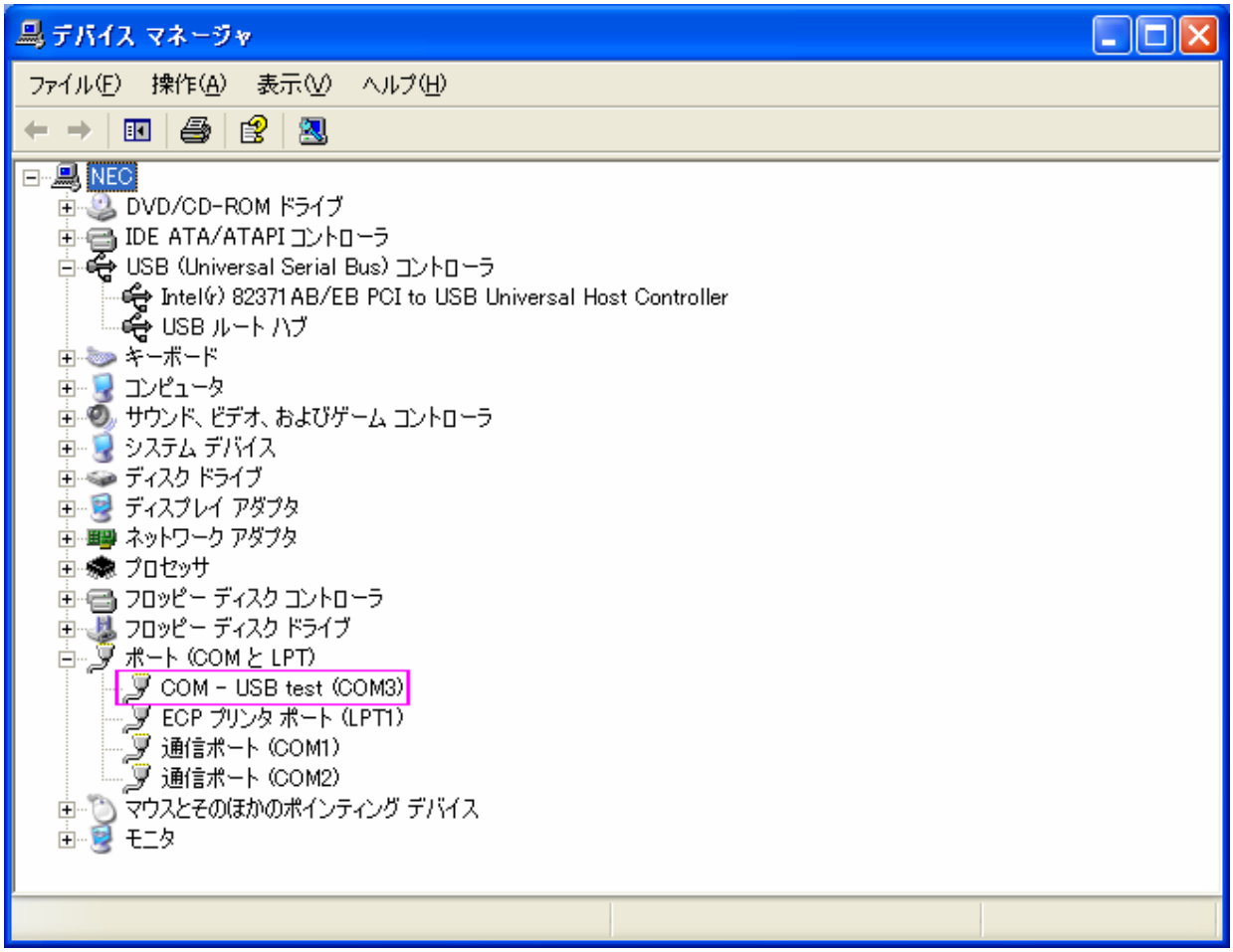
［ソフトウエアのインストール画面］

⑧　デバイス マネージャの起動

「マイ コンピュータ」の「プロパティ」から「ハードウェア」のタブを選択します。「デバイス マネージャ」の項目を選択し，「デバイス マネージャ」を起動します。

注意　「マイ コンピュータ」の「管理」や「コントロール パネル」からも起動できます。

[デバイス マネージャ表示例]



⑨　USBデバイスの接続の確認

「デバイス マネージャ」の画面上で，「ポート（COMとLPT）」の下に「COM-USB test (COM3)」が表示されていることを確認します。

注意　上記の［デバイス マネージャ表示例］では「COM-USB test (COM3)」ですが，COMポートの使用状況によっては「COM3」以外（COM4など）が表示されることがあります。

⑩　RS-232Cケーブルの接続

RS-232Cケーブルで，V850E/ME2に内蔵されているUARTB0とIE制御用マシンを接続します。

注意1.　**ターゲット・ボードの起動前に**RS-232C**ケーブルを接続しても問題ありません。**

2.　RS-232C**ケーブルでは，変換コネクタを使用してターゲット・ボード上のシリアル・コネクタ**JSIO2**とIE制御用マシンを接続します。適合コネクタについては，**SG-703111-1　**ユーザーズ・マニュアルを参照してください。**

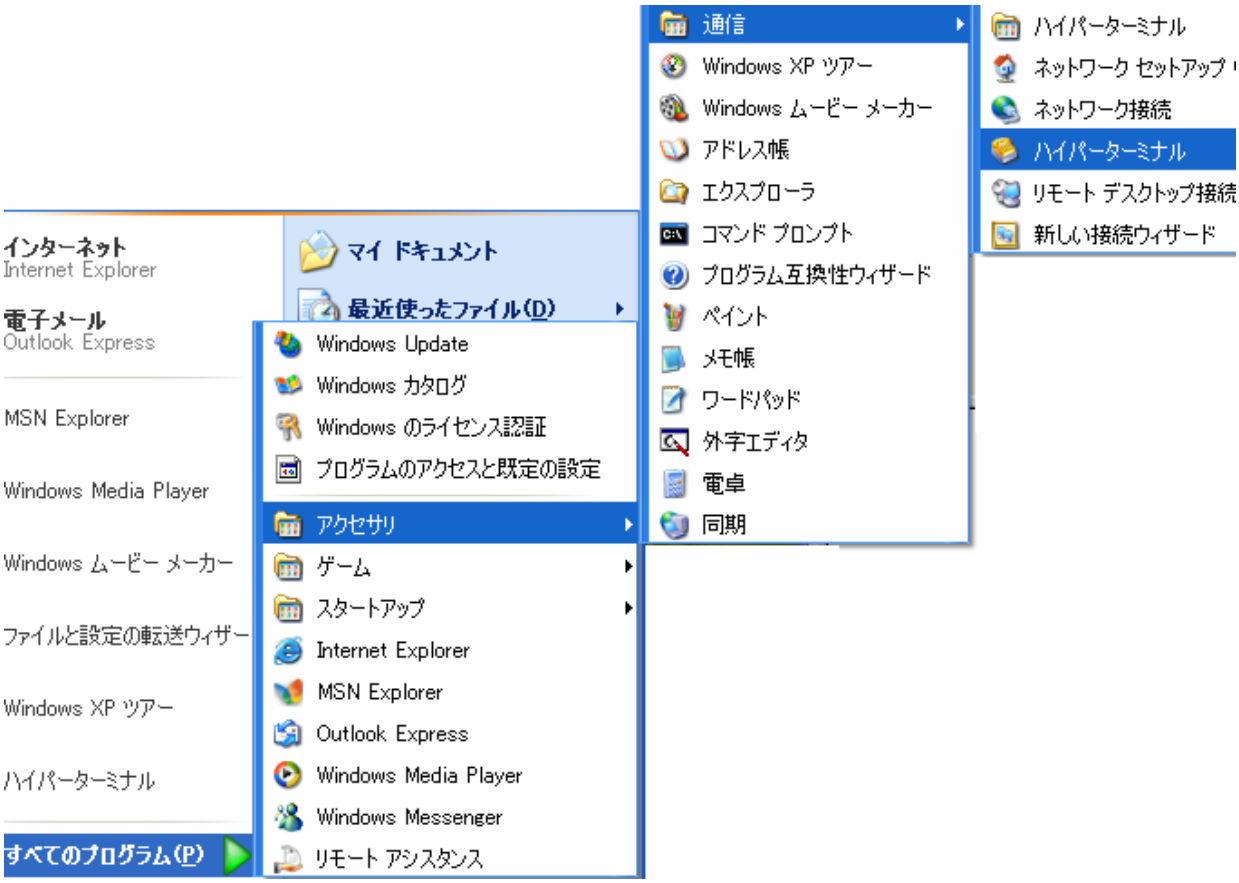
⑪　動作確認

ホスト・マシンおよびIE制御用マシンの両方で，ハイパーターミナルのようなターミナル・ソフトウエアを起動します。ここでは，例としてV850E/ME2に内蔵しているUARTB0をCOM1に接続するものとします。

ホスト・マシン側での設定方法を次に説明します。UARTB0側は，COM1に読み替えて設定してください。なお，転送速度などの設定は，ホスト・マシン側，IE制御用マシン側で，同じ設定値に設定してください。同じ設定値でないと正しく動作しません。

まず，「すべてのプログラム」―「アクセサリ」―「通信」―「ハイパーターミナル」を選択します。

［ハイパーターミナル起動例］

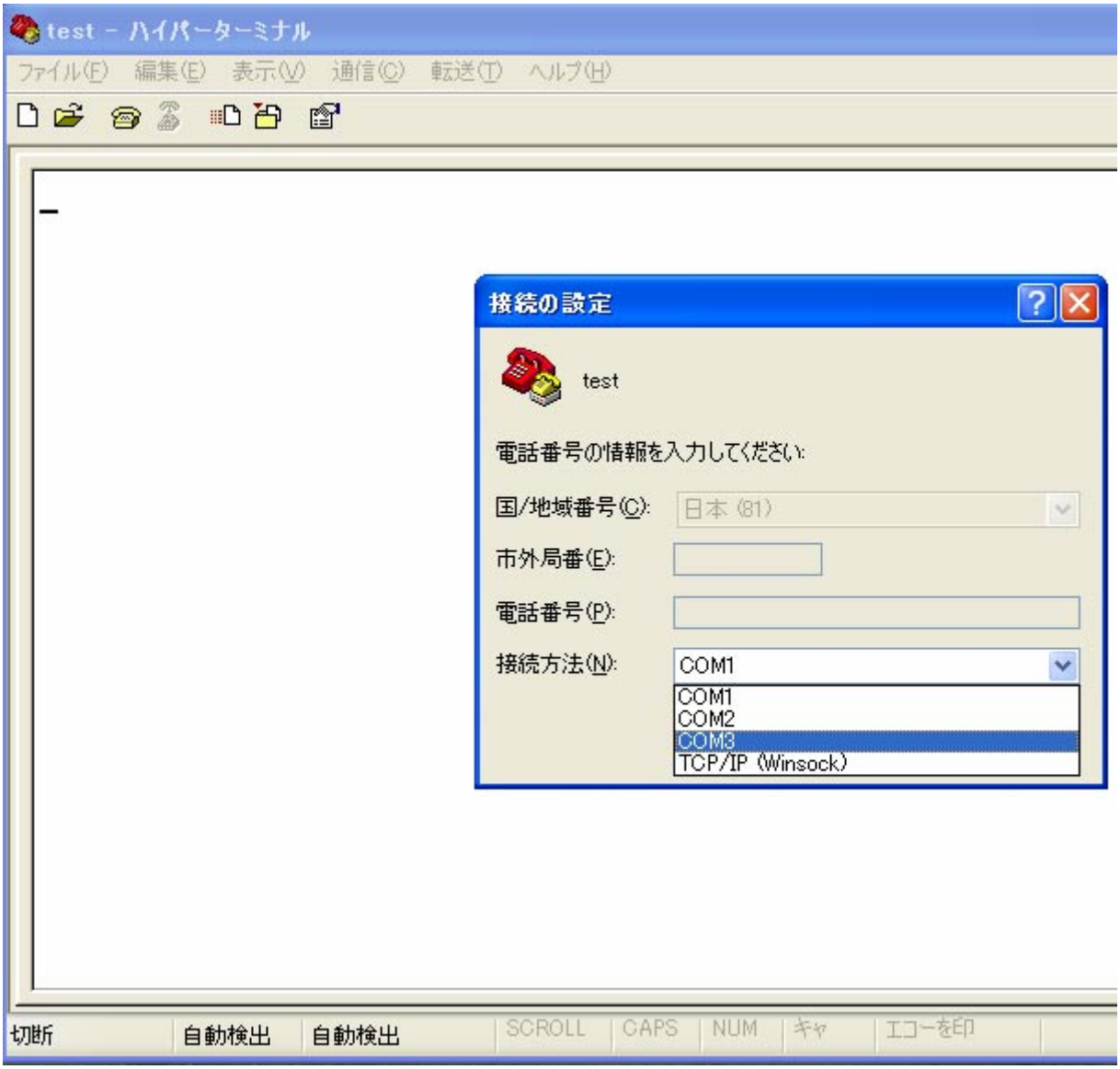


ハイパーターミナルが起動します。新しい接続の接続名を適当な名前で入力してください。

例では，testになっています。

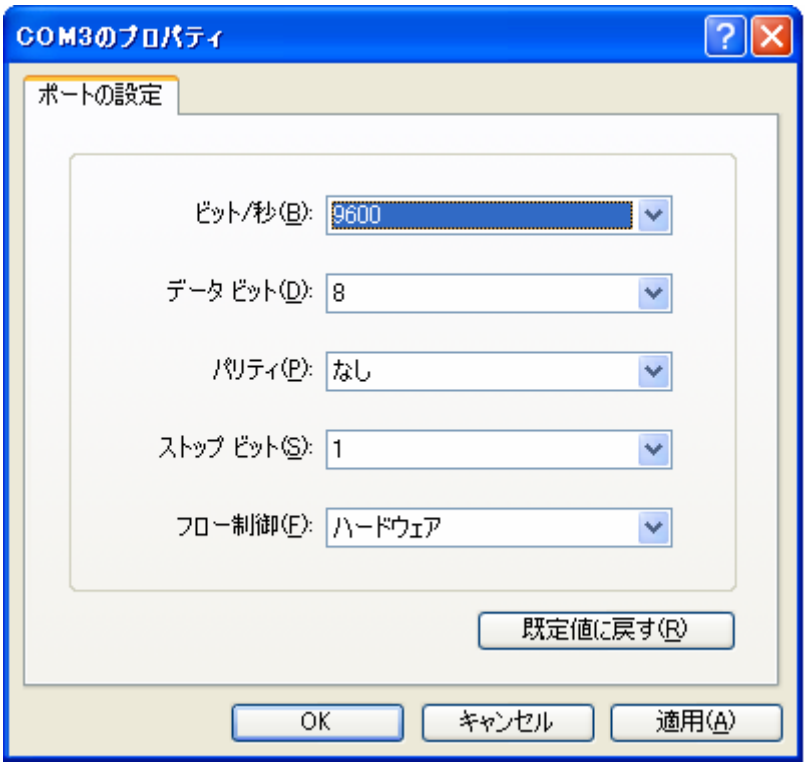
次に接続の設定画面が表示されます。USB側では，接続方法でCOM3を選択します。UARTB0側では，COM1を選択することになります。

［ハイパーターミナル起動画面］



接続方法が決定すると，「ポートの設定」画面が表示されます。「転送速度」，「データ長」，「パリティ」，「ストップ ビット」について，設定を行ってください。

[ハイパーターミナル設定画面]



設定が終了すると，データの送受信の動作確認ができます。

また，UARTの転送速度などの設定値を変更するときは，一度切断し，ハイパーターミナルの「プロパティ」－「モデムの構成」を選択し，設定画面から変更してください。このときにも，対向側の設定を同じように変更してください。

⑫　プログラムの終了方法

実行中のプログラムを終了します。

実行中のPARTNERの画面上にある強制終了ボタンをクリックするか，ツールバーの「実行」－「強制ブレーク」を選択し，プログラムの実行を停止します。

[強制終了ボタンによる終了例]

[強制ブレークによる終了例]

⑬　終了方法

インサーキット・エミュレータの終了を行い，ボードを①の手順でリセットします。

ツールバーの「ファイル」－「終了」を選択し，PARTNERを終了してください。

PARTNERを終了してから，ボードを①の手順でリセットします。

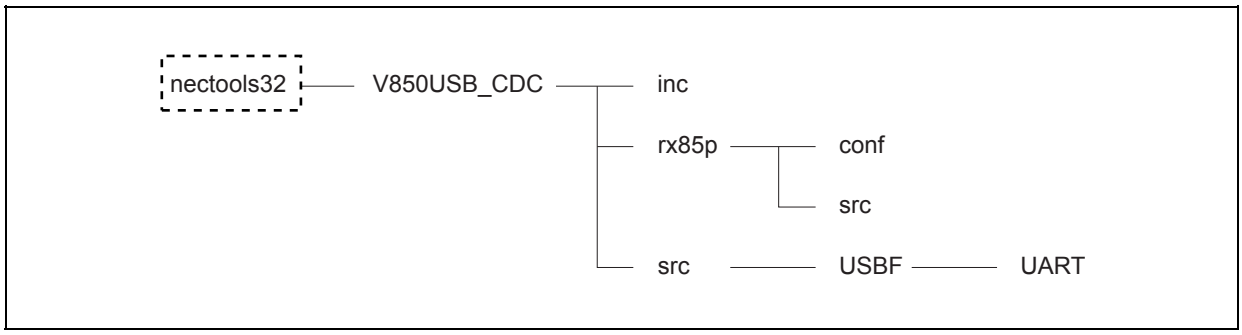
[PARTNERの終了例]

4.2.2　ディレクトリ構成

このサンプル・プログラム一式に含まれるファイル群のディレクトリ構成を，次に示します。

注意　USBコミュニケーション・クラス用ドライバ用ファイル郡のディレクトリは，RX850 Proをインストールしたディレクトリ（¥nectools32）の直下に置くことを推奨します。

図4－3　サンプル・プログラムのディレクトリ構成



各ディレクトリの概要を次に示します。

（1）nectools32

RX850 Proをインストールすると作成されるディレクトリです。このディレクトリの直下に，ドライバのディレクトリ（ディレクトリ名：V850USB_CDC）を置きます。

（2）nectools32¥V850USB_CDC

USBコミュニケーション・クラス用ドライバのディレクトリです。

- usb_communication.bld　：USBコミュニケーション・クラス用ドライバのビルド・ファイル
- common.lx　：セクション・マップ・ファイル

（3）nectools32¥V850USB_CDC¥inc

USBコミュニケーション・クラス用ドライバのヘッダ・ファイルが格納されているディレクトリです。

- errno.h　：戻り値用ヘッダ・ファイル
- types.h　：データ・タイプ用ヘッダ・ファイル
- sys.h　：システム情報ヘッダ・ファイル

注意　sys.h（システム情報ヘッダ・ファイル）は，ビルド時に生成されるファイルです。本来，このファイルは，コンフィギュレータを使用してコマンド操作により生成されます。しかし，サンプルのビルド・ファイルを使用すると，ビルド実行時にコマンドを実行する設定になっているため，あらかじめ生成しておく必要はありません。

（4）nectools32¥V850USB_CDC¥rx85p

RX850 Pro用のファイル群が格納されているディレクトリです。

(5) nectools32¥V850USB_CDC¥rx85p¥conf

RX850 Pro用のシステム・ファイルが格納されているディレクトリです。

- sit.850　：システム情報テーブル
- svc.850　：システム・コール・テーブル
- sysi.tbl　：システム情報テーブル
- sysc.tbl　：システム・コール・テーブル

注意1. **このディレクトリにあるファイルは，ビルド時に生成されるファイルです。本来，これらのファイルは，コンフィギュレータを使用してコマンド操作により生成されます。しかし，サンプルのビルド・ファイルを使用すると，ビルド実行時にコマンドを実行する設定になっているため，あらかじめ生成しておく必要はありません。**

2. sit.850とsysi.tbl，svc.850とsysc.tblは，それぞれファイルの拡張子が違うだけで，内容は同じものです。

(6) nectools32¥V850USB_CDC¥rx85p¥src

RX850 Pro用のファイルが格納されているディレクトリです。

- boot.850　：ブート処理用アセンブラ・ファイル
- entry.850　：エントリ処理用アセンブラ・ファイル
- init.c　：ハードウエア初期化部ソース・ファイル
- init.h　：ハードウエア初期化部ヘッダ・ファイル
- sys.cf　：CF定義ファイル
- varfunc.c　：ソフトウエア初期化部ソース・ファイル

(7) nectools32¥V850USB_CDC¥src

USBコミュニケーション・クラス用ドライバのボード依存部のファイルが格納されているディレクトリです。

- port.c　：ポート設定用ソース・ファイル
- port.h　：ポート設定用ヘッダ・ファイル

(8) nectools32¥V850USB_CDC¥src¥USBF

USBコミュニケーション・クラス用ドライバのUSB処理部のファイルが格納されているディレクトリです。

- usbf850.c　：USBデバイス用ソース・ファイル
- usbf850.h　：USBデバイス用ヘッダ・ファイル
- usbf850desc.h　：USB用ディスクリプタ定義ファイル
- usbf850_communication.c　：USB-UART間インタフェース用ソース・ファイル
- usbf850_communication.h　：USB-UART間インタフェース用ヘッダ・ファイル

(9) nectools32¥V850USB_CDC¥src¥USBF¥UART

USBコミュニケーション・クラス用ドライバのUART処理部のファイルが格納されているディレクトリです。

- uart_ctrl.c　　：UART処理部ソース・ファイル
- uart_ctrl.h　　：UART処理部ヘッダ・ファイル

注意　UART処理部についての詳細は，4．8　UART処理部を参照してください。サンプル・プログラムのUART処理部は，このサンプル・プログラムで必要となる最低限の機能だけをサポートしています。このため，このUART処理部は，汎用のUARTドライバとしての動作を保証しません。

4. 3　システム構築

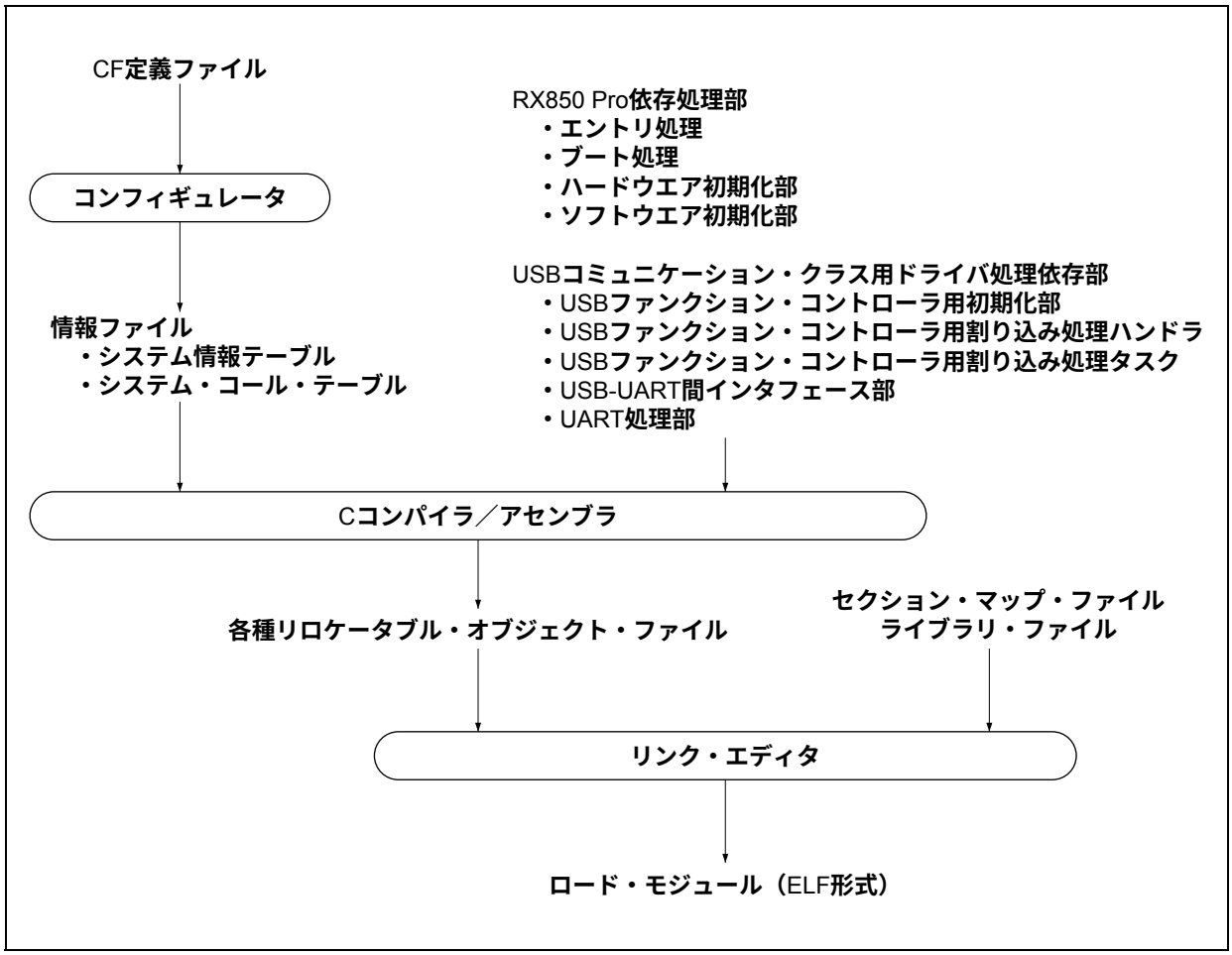
4. 3. 1　概　　要

システム構築とは，USBコミュニケーション・クラス用ドライバ提供媒体からユーザの開発環境（ホスト・マシン）上にインストールしたファイル群を使用して，ロード・モジュールを生成することです。

USBコミュニケーション・クラス用ドライバのシステム構築手順を，次に示します。

① RX850 Pro依存処理部の記述
② ボード依存部の記述
③ USBコミュニケーション・クラス用ドライバ処理依存部の記述
④ セクション・マップ・ファイルの記述
⑤ ロード・モジュールの生成

図4－4　システム構築手順



4. 3. 2　RX850 Pro依存処理部の記述

USBコミュニケーション・クラス用ドライバが提供する一部の機能は，リアルタイムOS（RX850 Pro）の機能を利用しています。また，ユーザが記述した処理プログラムは，RX850 Proの管理下で実行することになります。

したがって，RX850 Proを正常に動作させるために，RX850 Pro依存処理部の記述が必要となります。

RX850 Pro依存処理部の一覧を次に示します。

○CF定義ファイル
○エントリ処理
○システム初期化処理
・ブート処理
・ハードウエア初期化部
・ソフトウエア初期化部

備考　RX850 Pro依存処理プログラムについての詳細は，4. 4　RX850 Pro**依存処理プログラム**を参照してください。

4. 3. 3　ボード依存部の記述

USBコミュニケーション・クラス用ドライバのソース・プログラムは，ユーザの実行環境／アプリケーション・システムに依存した処理に関する初期化処理を，ボード依存部として切り出しています。

ボード依存部の一覧を次に示します。

・CPUボード依存部
USBコミュニケーション・クラス用ドライバで必要となるI/Oポート入出力操作については，CPUボード依存部として切り出しています。

注意　ポートの設定は，ほかのレジスタ設定と同じように扱うため，専用の関数は用意していません。レジスタの定義は，RX850 Pro**の標準ヘッダ・ファイル**¥nectools32¥inc850¥common¥SFR.h**を参照してください。処理方法は，ブート処理部（**boot.850**）およびソフトウエア初期化部から呼ばれるポート設定用ソース・プログラム（**port.c**）を参照してください。**

4.3.4　USBコミュニケーション・クラス用ドライバ処理依存部の記述

このサンプル・プログラムでは，USBコミュニケーション・クラス用ドライバ機能を実現するためのドライバ関数を，USBコミュニケーション・クラス用ドライバ処理依存部として切り出しています。

USBコミュニケーション・クラス用ドライバ処理依存部の一覧を，次に示します。

- USBファンクション・コントローラ初期化部
- USBファンクション・コントローラ割り込みハンドラ
- USBファンクション・コントローラの割り込み処理タスク
- USBファンクション・コントローラ用汎用関数
- USB-UART間インタフェース部
- UART処理部

備考　USBコミュニケーション・クラス用ドライバ処理依存部の詳細は，4.7　USB**コミュニケーション・クラス用ドライバの機能**を参照してください。また，UART処理部の詳細は，4.8　UART**処理部**を参照してください。

4.3.5　セクション・マップ・ファイルの記述

セクション・マップ・ファイルは，リンク・エディタが行うアドレス割り付けをユーザが固定化するためのファイルです。

RX850 Proを使用するとき，次の5つのテキスト領域が必須のセクションとなっています。

- 共通部分配置領域　：.systemセクション
- 割り込み処理関連配置領域　：.system_intセクション
- スケジューラ関連配置領域　：.system_cmnセクション
- システム情報領域　：.sitセクション
- インタフェース・ライブラリ／システム・コール配置領域　：.textセクション

備考　セクション・マップ・ファイルについての詳細は，4.5　**セクション・マップ・ファイル**を参照してください。

4.3.6　ロード・モジュールの生成

コーディングを終了したRX850 Pro依存処理プログラム，USBコミュニケーション・クラス用ドライバ処理依存部，セクション・マップ・ファイルに対して，Cコンパイラ，アセンブラ，リンカなどを実行することにより，ELF形式のロード・モジュールを生成します。

備考　ロード・モジュールの生成手順についての詳細は，4.6　**ロード・モジュール**を参照してください。

4. 4　RX850 Pro依存処理プログラム

4. 4. 1　概　　要

USBコミュニケーション・クラス用ドライバが提供する一部の機能は，リアルタイムOS（RX850 Pro）の機能を利用しています。また，ユーザが記述した処理プログラムは，RX850 Proの管理下で実行することになります。

したがって，RX850 Proを正常に動作させるために，RX850 Pro依存処理部の記述が必要となります。

RX850 Pro依存処理部の一覧を次に示します。

○CF定義ファイル

○エントリ処理

○システム初期化処理

・ブート処理

・ハードウエア初期化部

・ソフトウエア初期化部

4.4.2　CF定義ファイル

RX850 Proを使用したシステムを構築する場合，RX850 Proに提供する各種データを保持した情報ファイル（CF定義ファイル）が必要となります。

USBコミュニケーション・クラス用ドライバ機能を実現するうえで必要となる情報を，次に示します。

○リアルタイムOS情報
- RXシリーズ情報

○SIT情報
- システム情報
- システム最大値情報
- システム・メモリ情報
- タスク情報
- 割り込みハンドラ情報
- 初期化ハンドラ情報

○SCT情報
- タスク管理／タスク付属同期管理機能システム・コール情報
- 割り込み処理管理機能システム・コール情報
- 時間管理機能システム・コール情報

注意　このサンプル・プログラムでは，4個のタスク，6個の割り込みハンドラ，7種類のシステム・コールを使用して，各種機能を実現しています。このため，CF定義ファイルにおいては，システム最大値情報のタスクの最大生成数に“4個”を，割り込みハンドラの最大生成数に“6個”をUSBコミュニケーション・クラス用ドライバのために確保するほか，タスク管理／タスク付属同期管理機能システム・コール情報に“sta_tsk, ext_tsk, slp_tsk, wup_tskシステム・コール”を，割り込み処理管理機能システム・コール情報に“loc_cpu, unl_cpuシステム・コール”を，時間管理機能システム・コール情報に“dly_tskシステム・コール”を使用するものとして，定義する必要があります。ただし，割り込みハンドラ6個のうち3個は，UARTの割り込みによって使用されます。

備考　CF定義ファイルのコーディング方法についての詳細は，RX850 Pro　**ユーザーズ・マニュアル　インストレーション編**，およびサンプルのCF定義ファイル（sys.cf）を参照してください。

(1) 情報ファイルの生成手順

情報ファイル（システム情報テーブル，システム・コール・テーブル，システム情報ヘッダ・ファイル）の生成手順を，次に示します。

なお，情報ファイルの生成は，Windowsのコマンド・プロンプト上で行います。

注意　サンプルのビルド・ファイルを使用する場合，情報ファイルはビルド時に生成されます。このため，次の手順で生成する必要はありません。

① ディレクトリの移動

Windowsのcdコマンドを使用して，CF定義ファイルが格納されているディレクトリに移動します。なお，CF定義ファイルが格納されているディレクトリがC:¥sampleの場合の入力例を，次に示します。

[コマンド入力例]

```
C:>cd C:¥sample¥rx850<Enter>
```

② 情報ファイルの生成

コンフィギュレータcf850pro.exeを使用して，独自の記述形式で作成されたCF定義ファイルから情報ファイルを生成します。

なお，入力ファイル（CF定義ファイル名：sys.cf）から3つの情報ファイル（システム情報テーブル：sit.850，システム・コール・テーブル：svc.850，システム情報ヘッダ・ファイル：sys.h）を生成する場合の入力例を，次に示します。

[コマンド入力例]

```
C:>cf850pro –i sit.850 –c svc.850 –d sys.h sys.cf<Enter>
```

上記の操作により，CF定義ファイルから情報ファイルを生成できます。

注意　このサンプル・プログラムでは，情報ファイルを生成するためのサンプル・ファイル（CF定義ファイル）を提供しています。

備考　コンフィギュレータcf850pro.exeの起動オプションおよび実行方法についての詳細は，RX850 Pro　ユーザーズ・マニュアル　インストレーション編を参照してください。

4.4.3　エントリ処理

マスカブル割り込みが発生したときに，プロセッサが強制的に制御を移すハンドラ・アドレスに対して，割り込みハンドラへの分岐処理を割り付けています。

なお，RX850 Proの管理下で実行される割り込みハンドラ（CF定義ファイルの割り込みハンドラ情報で定義した割り込みハンドラ）に対応したハンドラ・アドレスに対しては，RX850 Proが提供するマクロRTOS_IntEntry_Indirect（RX850 Proが提供する割り込み処理管理機能への分岐処理）を割り付けてください。

備考　エントリ処理のコーディング方法についての詳細は，添付プログラムentry.850を参照してください。

4.4.4　システム初期化処理

システム初期化処理は，RX850 Proが正常に動作するうえで必要となるハードウエアの初期化処理（ブート処理，ハードウエア初期化部），およびソフトウエアの初期化処理（ニュークリアス初期化部，初期化ハンドラ）から構成されています。

したがって，システムが起動したとき，最初に行われる処理がシステム初期化処理となります。

注意　4種類のシステム初期化処理のうち，ニュークリアス初期化部については，RX850 Proが提供する機能の一部であるため，ユーザがその処理を記述する必要はありません。

ニュークリアス初期化部で行われる処理を次に示します。

○CF定義ファイルで定義されたシステム・メモリの確保
- System Pool 0
- User Pool 0

○CF定義ファイルで定義された管理オブジェクトの生成／初期化
- タスクの生成／起動
- 割り込みハンドラの登録

○初期タスクの起動

○アイドル・タスクの生成／起動

○ソフトウエア初期化部の呼び出し

○スケジューラに制御を移す

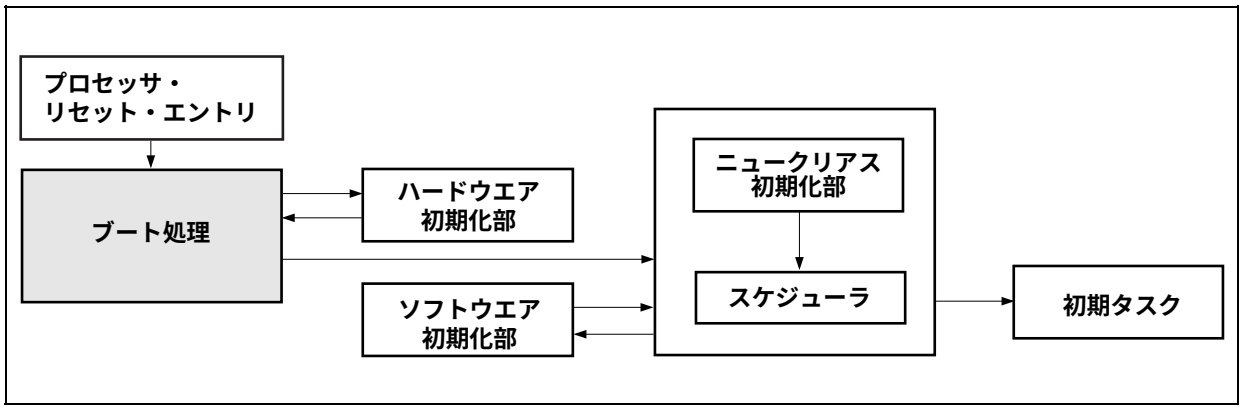
なお，アイドル・タスクとは，RX850 Proの管理下にある処理プログラム（タスク）がrun状態またはready状態でなくなったとき，すなわち，RX850 Proのスケジューリング対象となる処理プログラムがシステム内に1つも存在しなくなったときに，スケジューラから起動される処理ルーチンであり，HALT命令の発行を行っています。

(1) ブート処理

ブート処理は，プロセッサのリセット・エントリに割り付けられた関数であり，システム初期化処理の中でも最初に処理が実行されます。

ブート処理の位置付けを次に示します。

図4－5　ブート処理の位置付け



ブート処理で実行すべき処理内容を次に示します。

備考　ブート処理のコーディング方法についての詳細は，サンプルのboot.850を参照してください。

・tp, gp, epレジスタの設定

システム起動時，各種処理プログラム（ブート処理を含む）を実行するうえで必要となるテキスト・ポインタtp，グローバル・ポインタgp，スタック・ポインタepの値は不定です。そこで，ブート処理の最初の処理として，これらのレジスタの初期設定を行います。

注意　この章では，tpに“0”を，gpに“コンパイラが出力するグローバル・ポインタ・シンボル_gp”を，epに“コンパイラが出力するエレメント・ポインタ・シンボル_ep”を設定することを推奨します。

・ハードウエア初期化部の呼び出し

ターゲット・システム上のハードウエアを初期化するために用意された関数（ハードウエア初期化部）を呼び出します。

なお，ハードウエア初期化部で実行すべき処理内容（内部ユニットの初期化）をほかのところで行う場合は，“ハードウエア初期化部の呼び出し”は不要となります。

注意　この章では，ハードウエア初期化部で実行すべき処理内容（内部ユニットの初期化）をソフトウエア初期化部で行うため，“ハードウエア初期化部の呼び出し”は不要です。詳細は，RX850 Pro　ユーザーズ・マニュアル　インストレーション編を参照してください。

・ニュークリアス初期化部への制御の移行

ニュークリアス初期化部では，システム情報テーブルに記述された情報をもとに，システム・メモリ（System Pool 0, User Pool 0）の確保，および管理オブジェクトの生成／初期化などを行っています。そこで，ニュークリアス初期化部に制御を移す前には，r10レジスタにシステム情報テーブルの先頭アドレス_sitを設定しておく必要があります。

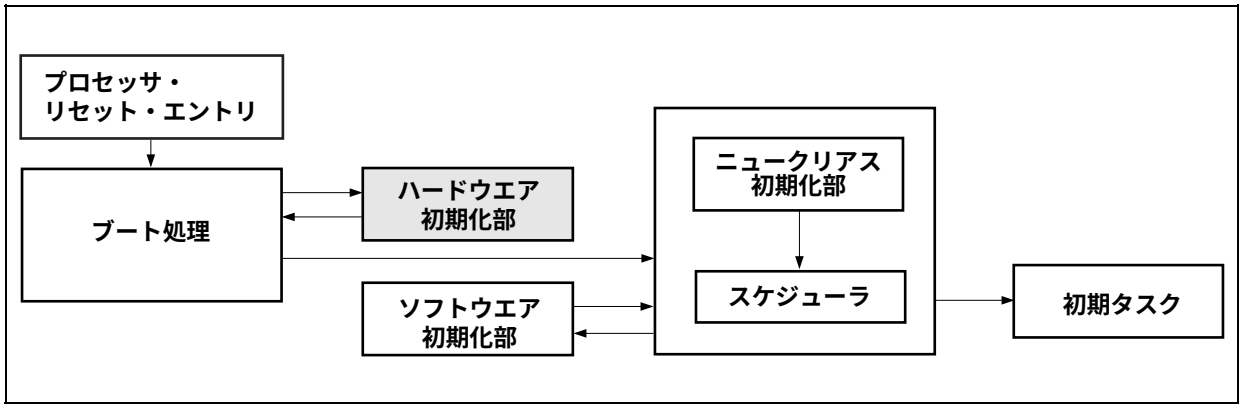
注意　システム情報テーブルとは，独自の記述形式で作成されたCF定義ファイルを，RX850 Proが提供するユーティリティ・ツール（コンフィギュレータcf850pro.exe）を使用して，アセンブリ言語形式に変換したものです。

(2) ハードウエア初期化部

ハードウエア初期化部は，ターゲット・システム上のハードウエアを初期化するために用意された関数であり，ブート処理から呼び出されます。

ハードウエア初期化部の位置付けを次に示します。

図4－6　ハードウエア初期化部の位置付け



ハードウエア初期化部で実行すべき処理内容を次に示します。

注意1. **マスカブル割り込みは，初期化時ディフォールトでマスクされていますので，特に禁止設定をする必要はありません。**

2. **サンプル・プログラムでは，ハードウエアの初期化をソフトウエア初期化部で行っています。ハードウエア初期化部の詳細は，**RX850 Pro　**ユーザーズ・マニュアル　インストレーション編を参照してください。**

・ブート処理に制御を戻す

ブート処理の呼び出し関数であるハードウエア初期化部からブート処理に制御を戻す場合，ブート処理におけるハードウエア初期化部の呼び出し時，lpレジスタに対する戻りアドレスの設定が行われているため，“ return( ); ”命令を発行することにより実現されます。

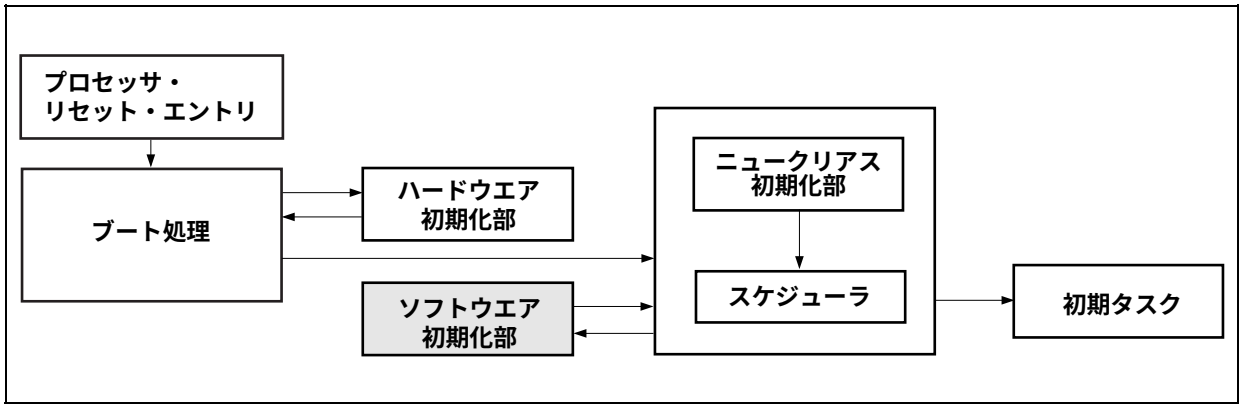
なお，ハードウエア初期化部をアセンブリ言語で記述した場合は，“ jmp [lp] ”命令を発行することにより実現されます。

(3) ソフトウエア初期化部

初期化ハンドラは，ユーザのソフトウエア環境を快適なものとするために用意された関数であり，ニュークリアス初期化部から呼び出されます。

ソフトウエア初期化部の位置付けを次に示します。

図4－7　ソフトウエア初期化部の位置付け



ソフトウエア初期化部で実行すべき処理内容を次に示します。

備考　ソフトウエア初期化部のコーディング方法についての詳細は，サンプルのvarfunc.cを参照してください。

・内部ユニット（リアルタイム・パルス・ユニット（RPU））の初期化

RX850 Proでは，一定周期で発生するタイマ割り込みを利用してタイマ・オペレーション機能（タスクの遅延起床，周期起動ハンドラの起動，タイムアウトなど）を実現しています。

このため，RX850 Proが処理を開始する前にリアルタイム・パルス・ユニットを初期化しておく必要があります。

なお，リアルタイム・パルス・ユニットのコンペア・レジスタCMD0には，CF定義ファイルのシステム情報で定義した基本クロック周期でタイマ割り込みが発生するような値を設定する必要があります。

・タイマ割り込みの受け付け許可

タイマ割り込みの受け付けを許可します。これにより，ニュークリアス初期化部の処理が終了したとき，RX850 Proが提供するタイマ・オペレーション機能（タスクの遅延起床，タイムアウト，周期起動ハンドラの起動など）の利用が可能となります。

・ニュークリアス初期化部に制御を戻す

ニュークリアス初期化部の呼び出し関数である初期化ハンドラからニュークリアス初期化部に制御を戻す場合，ニュークリアス初期化部における初期化ハンドラの呼び出し時に，lpレジスタに対する戻りアドレスの設定が行われているため，“ return( ); ”命令を発行することにより実現されます。

なお，初期化ハンドラをアセンブリ言語で記述した場合は，“ jmp [lp] ”命令を発行することにより実現されます。

4.4.5　時間管理機能

RX850 Proにおける時間管理は，ハードウエア（クロック・コントローラなど）により一定周期で発生するクロック割り込みを利用しています。

クロック割り込みが発生すると，RX850 Proのシステム・クロック処理が呼び出され，システム・クロックの更新やタスクの遅延起床，周期ハンドラの起動などの，時間に関連した処理が行われます。

システム・クロックは，RX850 Proが時間管理のときに使用する時刻（48ビット幅，単位：ms）を保持したソフトウエア・タイマです。

システム・クロックは，システム初期化処理で“0H”に設定されたあと，システム・クロック処理で基本クロック周期（コンフィギュレーション時に指定）を単位として更新されます。

注意　RX850 Proが管理するシステム・クロックは，48ビット幅で構成されています。このため，RX850 Proでは，オーバフローした数値（48ビットでは表せない数値）は無視されます。RX850 Proの時間管理機能の詳細は，RX850 Pro　ユーザーズ・マニュアル　基礎編を参照してください。

4.5　セクション・マップ・ファイル

4.5.1　概　　要

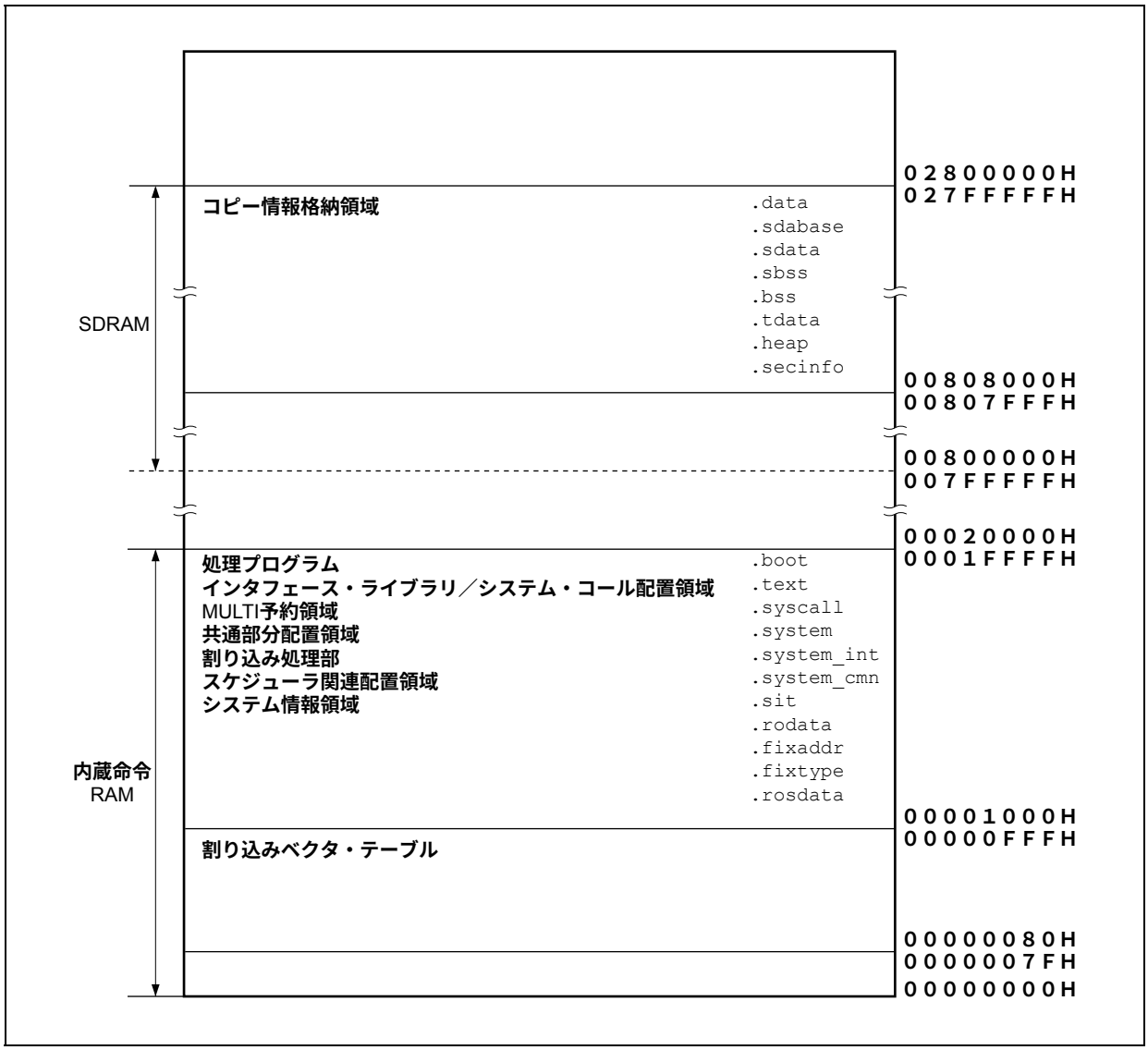
セクション・マップ・ファイルとは，リンク・エディタが行うアドレス割り付けを，ユーザが固定化するためのファイルです。

4.5.2　RX850 Proのアドレス割り付け，4.5.3　その他のアドレス割り付けに，ユーザの処理プログラム（.data, .bssセクションなど）以外に必要となるアドレス割り付けについて示します。

サンプルのcommon.lxで行われているアドレス割り付けを次に示します。

備考　セクション・マップ・ファイルのコーディング方法についての詳細は，サンプルのcommon.lxを参照してください。

図4－8　アドレス割り付け例



4. 5. 2　RX850 Proのアドレス割り付け

RX850 Proは，5つのテキスト領域（共通部分配置領域，割り込み処理関連配置領域，スケジューラ関連配置領域，システム情報領域，インタフェース・ライブラリ／システム・コール配置領域）から構成されています。これにより，大きなサイズが要求されるメモリ領域については外部RAMに，高速なアクセスが要求されるメモリ領域（割り込み処理部，スケジューリング処理部）については内蔵命令RAM（00000000H-0001FFFFH）に割り付けるといったことが可能となります。

注意　サンプル・プログラムでは，5つのテキスト領域のすべてを内蔵命令RAMに配置しています。

・共通部分配置領域（.systemセクション）

RX850 Proの本体処理（タスク管理機能，タスク付属同期機能など）が割り付けられる領域です。

・割り込み処理関連配置領域（.system_intセクション）

RX850 Proが提供する割り込み処理管理機能のうち，割り込みハンドラに制御を移すときに行われる割り込み前処理，およびマスカブル割り込みが発生した処理プログラムに制御を戻すときに行われる割り込み後処理から構成されています。

したがって，割り込み処理部を内蔵命令RAMに割り付けることにより，割り込みハンドラに対する応答性能が向上します。

注意　この章では，割り込み処理部を内蔵命令RAMに割り付けることを推奨します。

・スケジューラ関連配置領域（.system_cmnセクション）

RX850 Proが提供するスケジューリング機能のうち，タスクの起床処理およびタスクのスケジューリング処理から構成されています。

したがって，スケジューリング処理部を内蔵命令RAMに割り付けることにより，タスクの起床処理およびタスクのスケジューリング処理が高速化されるほか，スケジューリング処理を伴ったシステム・コールの処理も高速化されます。

注意　この章では，スケジューリング処理部を内蔵命令RAMに割り付けることを推奨します。

・システム情報領域（.sitセクション）

CF定義ファイルに対してコンフィギュレータcf850.exeを実行することにより生成されたシステム情報テーブルが割り付けられる領域です。

なお，システム情報テーブルは，ニュークリアス初期化部（システム・メモリの確保，管理オブジェクトの生成／初期化）を実行するうえで必要となる各種データから構成されています。

・インタフェース・ライブラリ／システム・コール配置領域（.textセクション）

システム・コールを含む命令が割り付けられる領域です。

・システム・メモリ

RX850 Proが提供する機能を実現するうえで必要となる各種管理ブロック（タスク管理ブロック，セマフォ管理ブロックなど），割り込みハンドラ用スタック，タスク用スタックが割り付けられる領域（System Pool 0），および処理プログラムからダイナミックなメモリ操作（メモリ・ブロックの獲得／開放）を可能とした領域（User Pool 0）から構成されています。

注意1. CF定義ファイル作成時，“システム・メモリの先頭アドレス”を指定する必要があります。したがって，セクション・マップ・ファイルにシステム・メモリの定義を行うときには，必ずアドレスを指定してください。

2. システム・メモリのセクション名については，ユーザが自由に指定できます。

4.5.3　その他のアドレス割り付け

次に，RX850 Pro以外にアドレス割り付けが必要となるセクションについて示します。

・MULTI予約領域（.syscallセクション）

ディバッガMULTI（Green Hills Software, Inc.製）がワーク・エリアとして使用する領域です。

注意1. MULTI使用の有無にかかわらず，.syscallセクションの定義を行う必要があります。

2. .syscallセクションの定義を行うときには，必ず4バイト・アライン指定を行ってください。

・コピー情報格納領域（.secinfoセクション）

セクション・マップ・ファイルで，ROM識別子の指定が行われているセクションのプログラム（データ，テキスト）をROMからRAMに転送するときに必要となる情報（先頭アドレス，サイズ）を，リンク・エディタが出力するための領域です。

なお，ROM識別子の指定は，処理プログラムをROM化するときに必要となるものです。したがって，ROM化を行わない場合には，.secinfoセクションの定義は不要となります。

注意　このサンプル・プログラムでは，ROM識別子の指定を行っていないため，空のセクションとなります。

4.6　ロード・モジュール

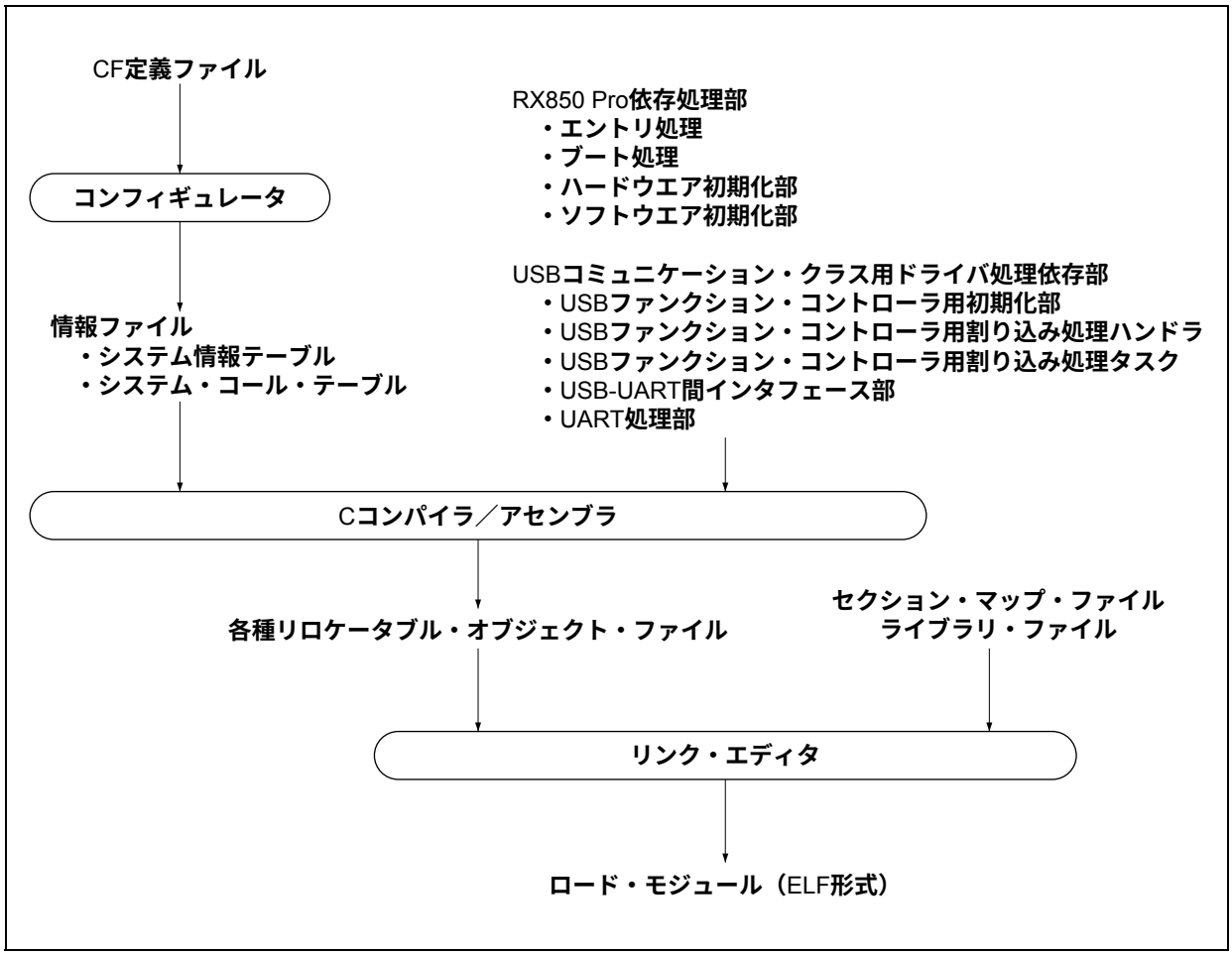
4.6.1　概　　要

コーディングを終了したRX850 Pro依存処理プログラム，USBコミュニケーション・クラス用ドライバ処理依存部，セクション・マップ・ファイルに対して，Cコンパイラ，アセンブラ，リンカなどを実行することにより，ELF形式のロード・モジュールを生成します。

ロード・モジュールの生成手順を次に示します。

注意　サンプルの.bldファイルを実行することにより，サンプル・プログラムに対応したロード・モジュールを生成できます。
ただし，.bldファイルのパスの定義については，ユーザの開発環境にあわせた修正が必要です。

図4－9　ロード・モジュールの生成手順



4. 6. 2　ロード・モジュールの生成

コーディングを終了したRX850 Pro依存処理プログラム，USBコミュニケーション・クラス用ドライバ処理依存部，セクション・マップ・ファイルは，次の手順によりELF形式のロード・モジュールとなります。

① システム情報テーブル，システム・コール・テーブルの生成

独自の記述形式で作成されたCF定義ファイルは，ロード・モジュール作成時，リンク・エディタが行うリンク処理の対象外のファイル形式です。

そこで，RX850 Proが提供するユーティリティ・ツール（コンフィギュレータcf850.exe）を使用して，アセンブル可能なファイル（システム情報テーブル，システム・コール・テーブル）を生成します。

備考　システム情報テーブル，システム・コール・テーブルの生成方法についての詳細は，4. 4. 2（1）**情報ファイルの生成手順**を参照してください。

② オブジェクト・ファイルの生成

次に示す各種処理プログラム（C言語／アセンブリ言語形式のファイル）に対して，Cコンパイラ／アセンブラを実行することにより，リロケータブル・オブジェクト・ファイルを生成します。

○RX850 Pro依存処理プログラム
- システム情報テーブル
- システム・コール・テーブル
- エントリ処理
- ブート処理
- ハードウエア初期化部
- 初期化ハンドラ

○USBコミュニケーション・クラス用ドライバ処理依存部

○UART処理部

③ ロード・モジュールの生成

②で生成したリロケータブル・オブジェクト・ファイル，および，各種ライブラリ・ファイル，セクション・マップ・ファイルに対して，リンク・エディタを実行することにより，ELF形式のロード・モジュールを生成します。

libansi.a	ANSI Cライブラリ
libind.a	Green Hills Software, Inc.製Cライブラリ（ターゲットCPUに依存しないルーチン群）
libarch.a	Green Hills Software, Inc.製Cライブラリ（ターゲットCPUに依存するルーチン群）
libsys.a	Green Hills Software, Inc.製Cライブラリ（システム・コール，初期化ルーチンなど）

rxcore.o	ニュークリアス共通部分オブジェクト
librxp.a	ニュークリアス・ライブラリ
libchp.a	インタフェース・ライブラリ

なお，rxcore.o, librxp.a, libchp.aは“ RX850 Pro ”から，libansi.a, libind.a, libarch.a, libsys.aは“ CCV850（Green Hills Software, Inc.製） ” から提供されています。

4. 7　USBコミュニケーション・クラス用ドライバの機能

4. 7. 1　概　　要

USBコミュニケーション・クラス用ドライバでは，USBコミュニケーション・クラス用ドライバ処理を実現するためのタスク，割り込みハンドラのほかに，USBファンクション・コントローラの初期化処理の記述が必要となります。

USBコミュニケーション・クラス用ドライバ処理依存部の一覧を，次に示します。

・USBファンクション・コントローラの初期化処理

RX850 Proのソフトウエア初期化部から呼び出され，USBファンクション・コントローラの初期化処理を行います。

・USBファンクション・コントローラの割り込みハンドラ

USBファンクション・コントローラの割り込み発生ごとに呼び出される割り込み処理専用ルーチンであり，CF定義ファイルに定義されています。

注意　このサンプル・プログラムでは，必要な割り込み以外はマスクされています。
このサンプル・プログラムで使用する割り込みは，次の3つです。

- INTUSB0B信号で通知されるSETRQ割り込み
（自動処理対象のSET_XXXXリクエストを受信し，自動処理を行ったことを示します）
- INTUSB0B信号で通知されるCPUDEC割り込み
（UF0E0STレジスタにFWでデコードを行うリクエストがあることを示します）
- INTUSB1B信号で通知されるBKO2DT割り込み
（UF0BO2レジスタにデータが正常受信されたことを示します）

・USBファンクション・コントローラの割り込み処理タスク

USBファンクション・コントローラの割り込みハンドラから呼び出され，割り込み要因ごとの処理（レジスタ設定，データの送信／受信処理など）を行います。

・USBファンクション・コントローラ用汎用関数

USBコミュニケーション・クラス用ドライバで使用される汎用関数として，エンドポイントごとのSTALL応答の設定や，送信，受信の処理を行う関数が用意されています。

備考　USBコミュニケーション・クラス用ドライバ処理依存部のコーディング方法についての詳細は，サンプルのusbf850.cを参照してください。

・USB-UART間インタフェース部

USBコミュニケーション・クラス用のデバイス・クラス固有のリクエスト処理，およびUSB-UART間のデータ送信／受信処理を行います。

注意　このサンプル・プログラムで受け付けるデバイス・クラス固有のリクエストは，次の5つです。各リクエストの詳細は，Universal Serial Bus Class Definitions for communication Devices Version 1.1**を参照してください。**

- SEND ENCAPSULATED COMMANDリクエスト
- GET ENCAPSULATED RESPONSEリクエスト
- SET LINE CODINGリクエスト
- GET LINE CODINGリクエスト
- SET CONTROL LINE STATEリクエスト

備考　USB-UART間インタフェース部のコーディング方法についての詳細は，サンプルのusbf850_communication.cを参照してください。

・USBのサスペンド／レジュームの処理

USBのサスペンド／レジュームの処理はシステムに依存するため，このサンプル・プログラムではサポートしていません。システム上，処理が必要な場合は，次のことに注意して処理を追加してください。

V850E/ME2に内蔵しているUSBファンクション・コントローラでは，サスペンド／レジュームが割り込み（INTUSB0B信号）により通知されます。このため，割り込みハンドラ（INTUSB0B信号用）内でUF0IS0.RSUSPDビットを確認し，セット（1）されていれば，UF0EPS1.RSUMビットを確認してサスペンド状態なのかレジューム状態なのかを判断できます。

処理を追加する一例として，割り込みハンドラ（INTUSB0B信号用）に上記の状態判断のコードを追加し，そこから必要な処理を行うタスクを起床するようにすることで実現できます。

4.7.2　処理の流れ

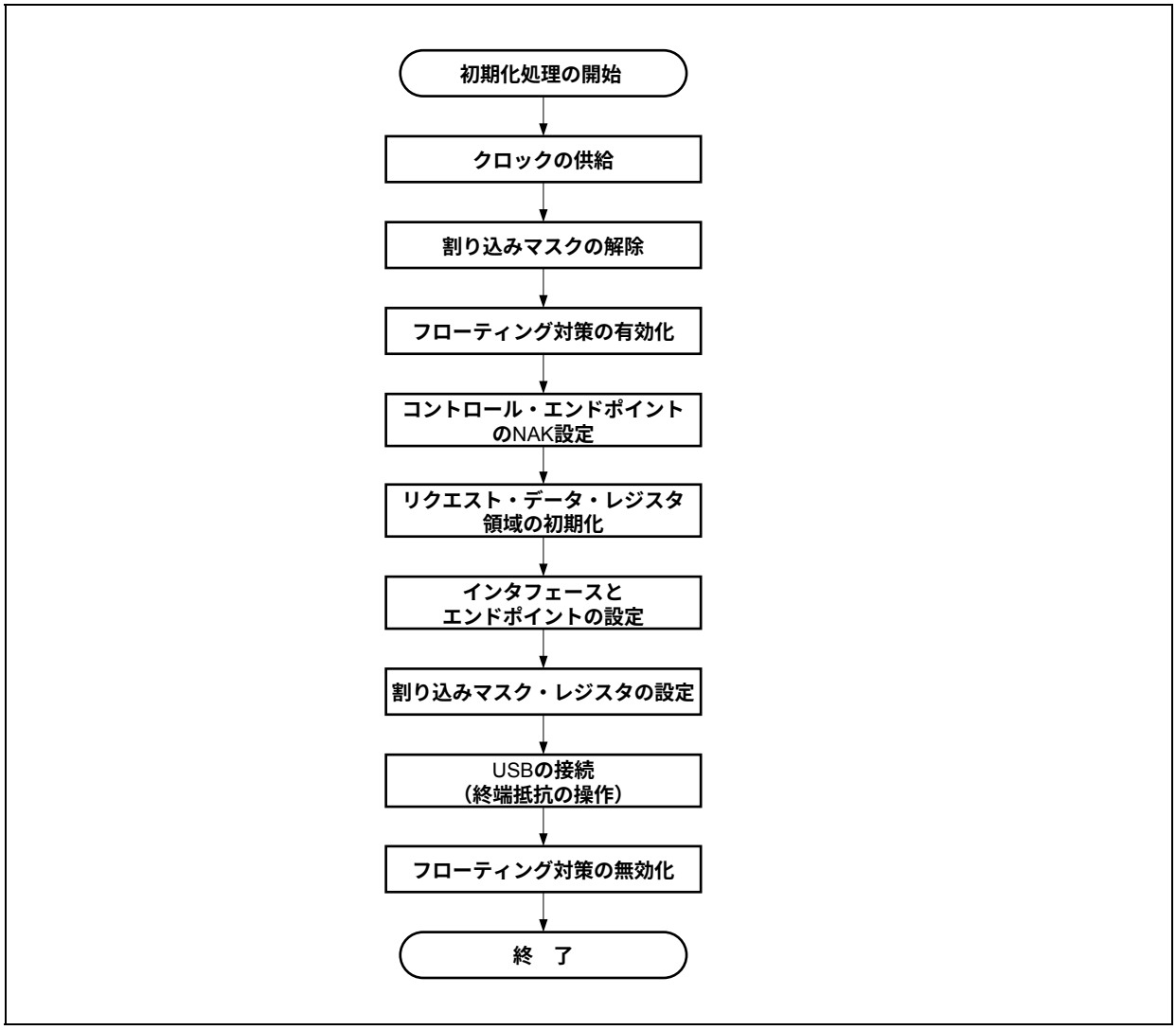
サンプル・プログラムの初期化処理，割り込み処理についての処理の流れを，次に示します。

(1) 初期化処理

USBデバイスの初期化は，ソフトウエア初期化部から呼び出され実行されます。

サンプル・プログラムにおけるUSBデバイス初期化処理（電源投入時）の流れを，次に示します。

図4－10　初期化処理のフロー・チャート



初期化処理で実行すべき処理内容を次に示します。

注意　ポートの処理以外では，初期化処理が必要です。ターゲット・ボードが違う場合，端子の割り付けが違うことがありますので，ターゲット・ボードの仕様にあわせて読み替えてください。

・クロックの供給

USBファンクション・コントローラのレジスタを設定する前に，必ずUCKC.UCKCNTビットをセット（1）する必要があります。セット（1）することにより，USBへのクロック供給を許可します。なお，P10端子をクロック入力として使用するので，P10端子を入出力ポート・モードの入力モードに設定し，クロックの入力を許可します。

・割り込みマスクの解除

割り込み制御レジスタでUSB関連の割り込み信号のマスクを解除します。

・フローティング対策の有効化

UF0BC.UBFIORビットをクリア（0）し，ケーブル未接続時の不定値によるBus Resetなどの誤認識を防止します。

・コントロール・エンドポイントのNAK設定

自動実行リクエストを含むすべてのリクエストにNAK応答します。
自動実行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実行リクエストに意図しないデータを返さないように設定します。

・リクエスト・データ・レジスタ領域の初期化

Get Descriptorリクエストに応答するためのディスクリプタ・データなどをレジスタに登録します。登録するデータは，デバイス・ステータス，エンドポイント0ステータス，Device Descriptor, Configuration Descriptor, Interface Descriptor, Endpoint Descriptorです。

注意　クラスによっては，そのクラス用のディスクリプタの登録が必要な場合があります。このサンプル・プログラムでは，USBコミュニケーション・クラスを定義し，USBの標準ディスクリプタだけを使用します。

・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，Alternative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，レジスタに設定します。

・コントロール・エンドポイントのNAK設定の解除

自動実行リクエスト用のデータ登録が終わったところで，コントロール・エンドポイント（エンドポイント番号0）のNAK設定を解除します。

・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ステータス・レジスタに示される割り込み要因ごとのマスクを設定します。

・USBの接続（終端抵抗の操作）

D+信号をプルアップします。

・フローティング対策の無効化

UF0BC.UBFIORビットをセット（1）し，フローティング対策を無効化します。

（2）割り込み処理

サンプル・プログラムでは，初期化完了後，割り込みイベントによって動作します。イベントがない場合は，常にアイドル状態です。ただし，割り込みはUSBファンクション・コントローラからだけでなく，UARTからも通知されます。

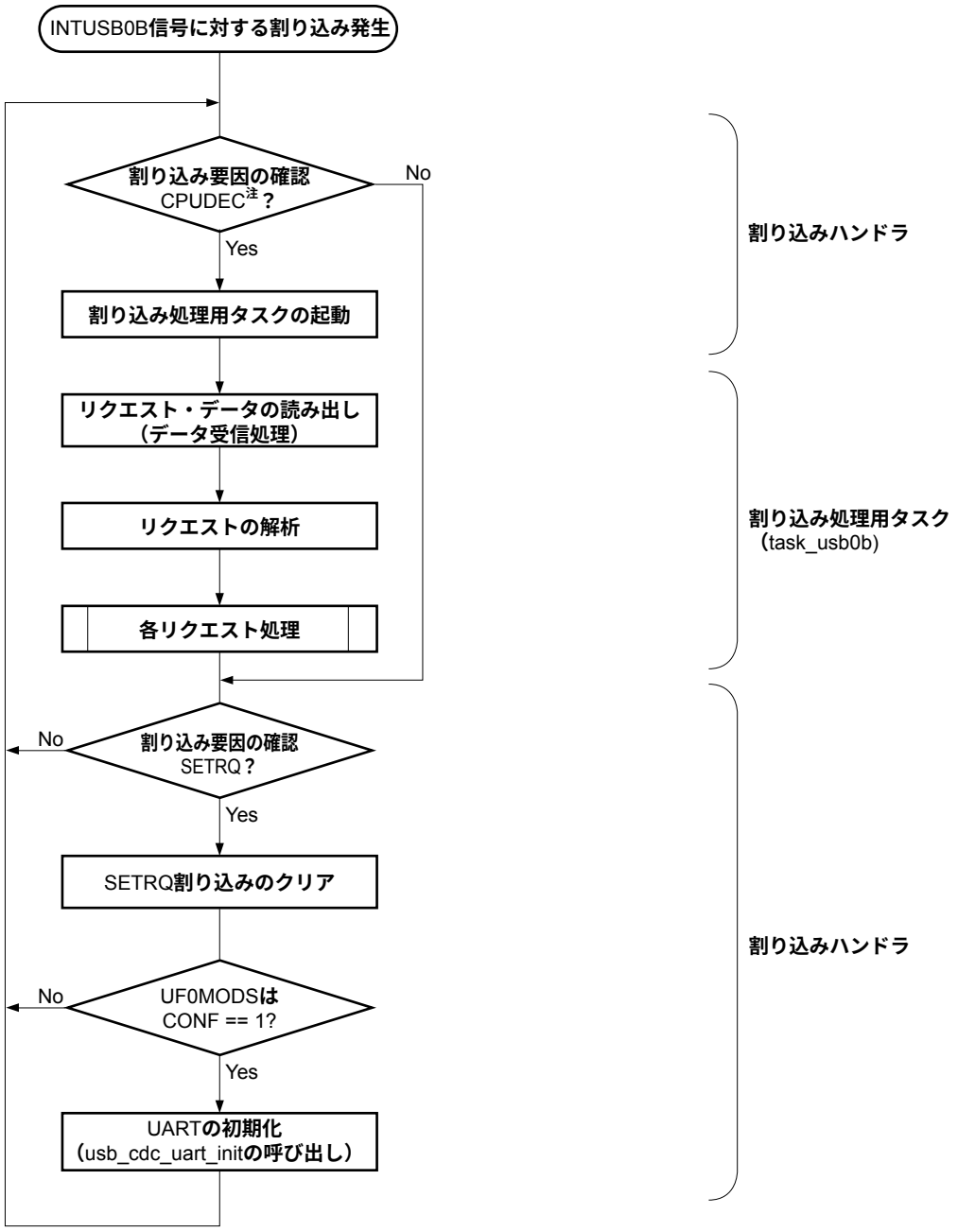
サンプル・プログラムでの割り込み処理の流れを，図4－11，図4－12に示します。

注意　図4－11に示すフロー・チャートは，USBファンクション・コントローラのINTUSB0B信号により通知される割り込み処理の流れを示しています。

図4－12に示すフロー・チャートは，USBファンクション・コントローラのINTUSB1B信号により通知される割り込み処理の流れを示しています。

UARTの割り込み処理の詳細については，4. 8　UART処理部を参照してください。

図4－11　割り込み処理のフロー・チャート（1）



注　USBコミュニケーション・クラス用ドライバのINTUSB0B信号では，CPUDEC, SETRQ割り込みを受け付けます。

INTUSB0B信号による割り込みでのサンプル・プログラムの処理内容を，次に示します。

[割り込みハンドラ内での処理]

・割り込み要因の確認

サンプル・プログラムでは，実行される割り込みハンドラにより，解析する割り込みステータスが違います。

INTUSB0B信号で通知される割り込みについては，CPUDEC, SETRQ割り込みに対応しています。これらの割り込みが発生すると，INTUSB0B信号による割り込みハンドラが起動します。 この割り込みハンドラの中で，UF0IS1レジスタを読み，割り込み要因がCPUDEC割り込みかどうかを確認します。さらに，UF0IS0レジスタを読み，CONFビットがセット（1）されているかどうかを確認します。

注意　このサンプル・プログラムでは，使用する割り込みハンドラをCF定義ファイルであらかじめ登録してあります。

・割り込み処理用のタスクの起動

割り込み要因がCPUDECだった場合，task_usb0bタスクを起動します。

注意　このサンプル・プログラムでは，起動するタスクをCF定義ファイルであらかじめ登録してあります。

・UARTの初期化

SETRQ割り込みだった場合，さらにUF0IS0レジスタを読み，CONFビットがセット（1）されているかどうかを確認します。CONFビットがセット（1）されていれば，usb_cdc_uart_init関数を呼び出し，UARTの初期化を行います。

[task_usb0bタスクでの処理]

・リクエスト・データの読み出し

UF0E0STレジスタからSETUPデータを読み出します。

・リクエストの解析

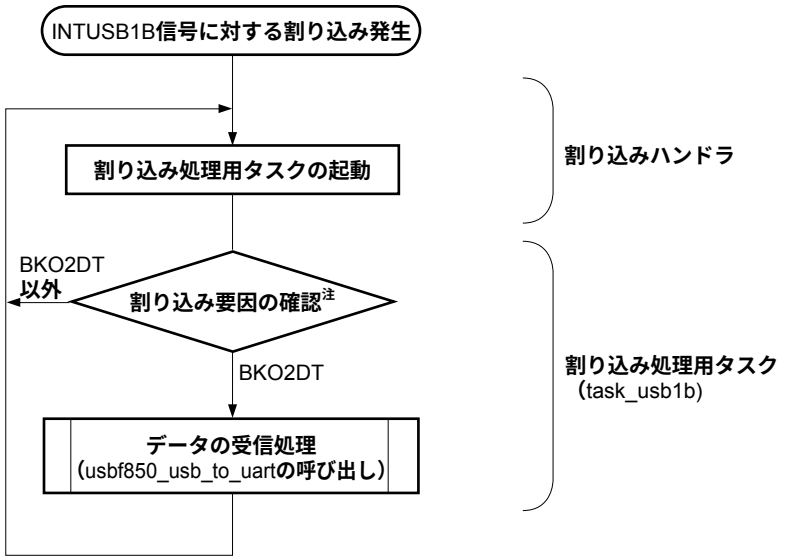
読み出したSETUPデータを解析し，リクエスト内容を確認します。

・各リクエスト処理

解析したリクエスト内容に対応した処理を行います。

サンプル・プログラムでは，標準デバイス・リクエストのGet Descriptor（String Descriptor）リクエスト，およびデバイス・クラス固有のリクエストの処理を行います。

図4－12　割り込み処理のフロー・チャート（2）



注　USBコミュニケーション・クラス用ドライバのINTUSB1B信号では，BKO2DT割り込みだけを受け付けます。

INTUSB1B信号による割り込みでのサンプル・プログラムの処理内容を，次に示します。

・割り込み処理用のタスクの起動

割り込み要因を確認せず，task_usb1bタスクを起動します。

注意　このサンプル・プログラムでは，起動するタスクをCF定義ファイルであらかじめ登録してあります。

・割り込み要因の確認

INTUSB1B信号で通知される割り込みについては，BKO2DT割り込みだけに対応しています。割り込みが発生すると，INTUSB1B信号による割り込みハンドラが起動します。

割り込みハンドラ内では割り込み要因を確認せず，起動されたタスクで割り込み要因がBKO2DTであるかを確認します。

注意　このサンプル・プログラムでは，使用する割り込みハンドラをCF定義ファイルであらかじめ登録してあります。

・データの受信処理

割り込み要因がBKO2DTであれば，USBからUARTへのデータ転送関数（usbf850_usb_to_uart関数）を呼び出します。

備考　UARTの処理については，4. 8　UART処理部を参照してください。

4. 7. 3　USBコミュニケーション・クラス用ドライバのディスクリプタ情報

サンプル・プログラムで定義されているUSB標準ディスクリプタについて，次に示します。

(a) - (d) までのディスクリプタは，最低限用意するべきディスクリプタです。

備考　詳細は，Universal Serial Bus Specification Revision 1.1を参照してください。

(a) Device Descriptor

デバイスの一般的な情報を持ち，デバイスごとに1つのDevice Descriptorを用意する必要があります。この情報は，デバイスのコンフィギュレーションで唯一のデバイスを認識するために使用されます。このサンプル･プログラムでは，USBコミュニケーション・クラスが定義されています。

表4－1　Device Descriptor

オフセット	サイズ（バイト）	値	説　明
0	1	12H	このディスクリプタの Length 値（バイト）
1	1	01H	ディスクリプタ・タイプ（デバイス）
2	2	10H/01H	USB のバージョン（USB1.1）
4	1	02H	クラス・コード（Communication Device Class）
5	1	00H	サブクラス・コード
6	1	00H	プロトコル・コード
7	1	40H	Endpoint0 の最大パケット・サイズ
8	2	09H/04H	ベンダ ID（NEC エレクトロニクス）
10	2	FDH/FFH	プロダクト ID
12	2	01H/00H	デバイス・リリース番号
14	1	01H	ストリング・ディスクリプタへのインデクス（Manufacturer）
15	1	02H	ストリング・ディスクリプタへのインデクス（Product）
16	1	03H	ストリング・ディスクリプタへのインデクス（Serial Number）
17	1	01H	可能なコンフィギュレーションの数

（b）Configuration Descriptor

具体的なデバイス・コンフィギュレーションについての情報を保持しています。

表4－2　Configuration Descriptor

オフセット	サイズ（バイト）	値	説　明
0	1	09H	このディスクリプタの Length 値（バイト）
1	1	02H	ディスクリプタ・タイプ（コンフィギュレーション）
2	2	30H/00H	Get Descriptor 要求でコンフィギュレーション・ディスクリプタと一緒に返されるディスクリプタのトータル Length 値
4	1	02H	この構成でサポートされているインタフェース数
5	1	01H	コンフィギュレーション値
6	1	00H	ストリング・ディスクリプタへのインデクス（Configuration）
7	1	C0H	デバイスの構成（Self-powered/Remote Wakeup 機能）
8	1	00H	デバイスの最大消費電力

（c）Interface Descriptor

コンフィギュレーション内の具体的なインタフェース情報を保持しています。

サンプル・プログラムでは，コンフィギュレーションは2つのインタフェースを提供します。

Interface Descriptorは，常にConfiguration Descriptorの一部として戻され，Interface Descriptorにおいて，Get DescriptorおよびSet Descriptor要求による直接アクセスはされません。

表4－3　Interface Descriptor（1）

オフセット	サイズ（バイト）	値	説　明
0	1	09H	このディスクリプタの Length 値（バイト）
1	1	04H	ディスクリプタ・タイプ（インタフェース）
2	1	00H	インタフェース値
3	1	00H	Alternate 設定値
4	1	01H	Endpoint 数（Endpoint0 を除く）
5	1	02H	インタフェース・クラス（Communication Interface Class）
6	1	02H	インタフェース・サブクラス（Abstract Control Model）
7	1	00H	インタフェース・プロトコル（No Class：USB Specification）
8	1	00H	ストリング・ディスクリプタへのインデクス（インタフェース）

表4－4　Interface Descriptor（2）

オフセット	サイズ（バイト）	値	説　明
0	1	09H	このディスクリプタの Length 値（バイト）
1	1	04H	ディスクリプタ・タイプ（インタフェース）
2	1	01H	インタフェース値
3	1	00H	Alternate 設定値
4	1	02H	Endpoint 数（Endpoint0 を除く）
5	1	0aH	インタフェース・クラス（Data Interface Class）
6	1	00H	インタフェース・サブクラス（Data Class）
7	1	00H	インタフェース・プロトコル（No Class：USB Specification）
8	1	00H	ストリング・ディスクリプタへのインデクス（インタフェース）

（d）Endpoint Descriptor

ホストが個々のEndpointのバンド幅要件を決定するために必要な情報を保持しています。

Endpoint Descriptorは，常にコンフィギュレーション・ディスクリプタの一部として戻され，Endpoint Descriptorにおいて，Get DescriptorおよびSet Descriptor要求による直接アクセスはされません。

表4－5　Endpoint Descriptor（Interrupt IN）

オフセット	サイズ（バイト）	値	説　明
0	1	07H	このディスクリプタのLength値（バイト）
1	1	05H	ディスクリプタ・タイプ（Endpoint）
2	1	87H	Endpointのアドレス値
3	1	03H	Endpointの転送タイプ
4	2	08H/00H	Endpointの最大パケット・サイズ
6	1	0AH	インターバル（ms）：Isochronous, Interrupt Endpointだけ有効

表4－6　Endpoint Descriptor（Bulk IN）

オフセット	サイズ（バイト）	値	説　明
0	1	07H	このディスクリプタのLength値（バイト）
1	1	05H	ディスクリプタ・タイプ（Endpoint）
2	1	83H	Endpointのアドレス値
3	1	02H	Endpointの転送タイプ
4	2	40H/00H	Endpointの最大パケット・サイズ
6	1	00H	インターバル（ms）：Isochronous, Interrupt Endpointだけ有効

表4－7　Endpoint Descriptor（Bulk OUT）

オフセット	サイズ（バイト）	値	説　明
0	1	07H	このディスクリプタのLength値（バイト）
1	1	05H	ディスクリプタ・タイプ（Endpoint）
2	1	04H	Endpointのアドレス値
3	1	02H	Endpointの転送タイプ
4	2	40H/00H	Endpointの最大パケット・サイズ
6	1	00H	インターバル（ms）：Isochronous, Interrupt Endpointだけ有効

（e）String Descriptor

このサンプル・プログラムでは，デバイスの製造社名などの情報を保持します。

表4－8　String Descriptor（1）

オフセット	サイズ（バイト）	値	説　明
0	1	04H	このディスクリプタのLength値（バイト）
1	1	03H	ディスクリプタ・タイプ（ストリング）
2	2	09H/04H	ストリング・ディスクリプタで使用する言語タイプ（English/U.S.）

表4－9　String Descriptor（2）

オフセット	サイズ（バイト）	値	説　明
0	1	2AH	このディスクリプタのLength値（バイト）
1	1	03H	ディスクリプタ・タイプ（ストリング）
2	40	'N','E','C','','E','l','e','c','t','r','o','n','i','c','s',' ','C','o','.'	製造社名（Manifacturer） NEC Electronics Co.

表4－10　String Descriptor（3）

オフセット	サイズ（バイト）	値	説　明
0	1	16H	このディスクリプタのLength値（バイト）
1	1	06H	ディスクリプタ・タイプ（ストリング）
2	20	'C','o','m','m','u','n','i','D','r','v'	製品名（Product） CommuniDrv

表4－11　String Descriptor（4）

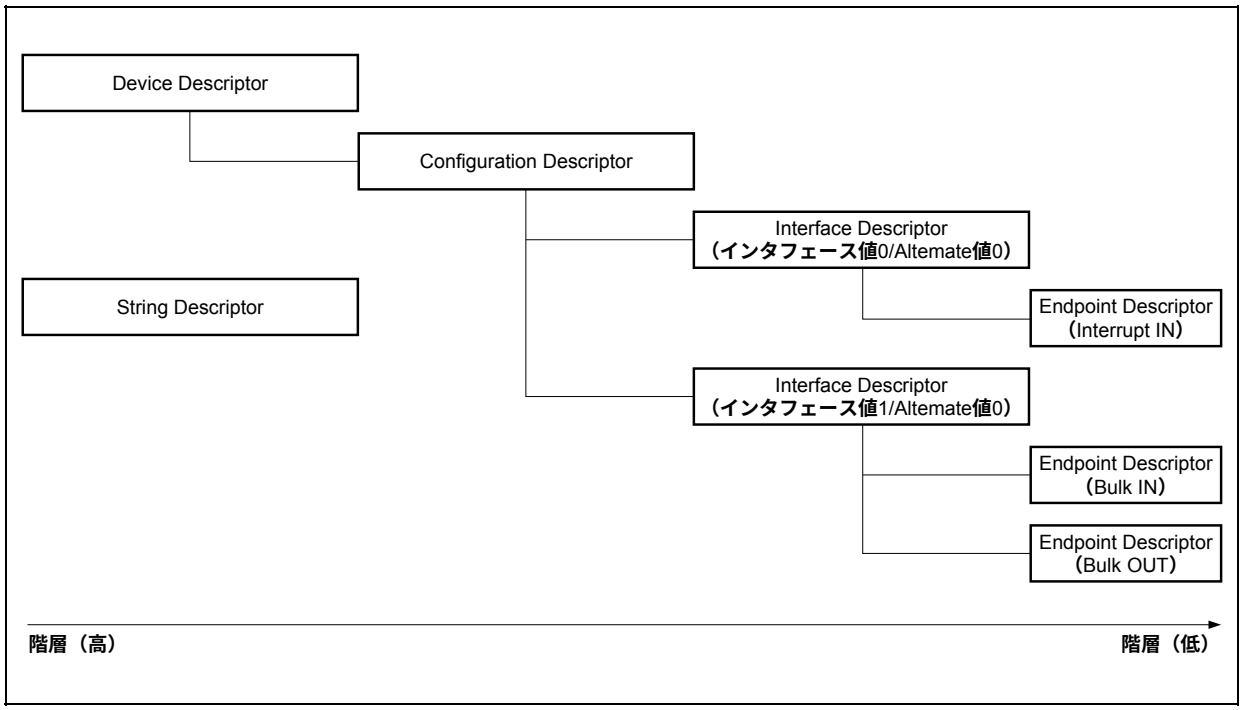
オフセット	サイズ（バイト）	値	説　明
0	1	16H	このディスクリプタのLength値（バイト）
1	1	06H	ディスクリプタ・タイプ（ストリング）
2	20	'0','_','9','8','7','6','5','4','3','2'	シリアル番号（Serial Number） 0_98765432

(1) ディスクリプタの構成

サンプル・プログラムにおけるディスクリプタの構成を次に示します。前述したディスクリプタ類を次の構成で準備します。

注意　Device Descriptor/Configuration Descriptor/String Descriptorは，それぞれ別々のGet Descriptorリクエストによってアクセスされます。Interface DescriptorおよびEndpoint Descriptorは，Configuration Descriptorの一部としてアクセスされます。

図4－13　ディスクリプタ構成図



4.7.4　データ・マクロ

USBコミュニケーション・クラス用ドライバ内で使用する各種データ・マクロ（データ・タイプ，戻り値など）について，次に示します。

(1) データ・タイプ

USBコミュニケーション・クラス用ドライバ関数を発行するときに指定する各種パラメータのデータ・タイプのマクロ定義は，ヘッダ・ファイルnectools32¥USB_CDC¥inc¥types.hで行われています。

データ・タイプ一覧を次に示します。

表4－12　データ・タイプ一覧

マクロ	型	意　味
(*PFV) ()	void	処理プログラムの起動アドレス

(2) 戻 り 値

USBコミュニケーション・クラス用ドライバ関数からの戻り値のマクロ定義は，ヘッダ・ファイルnectools32¥USB_CDC¥inc¥errno.hで行われています。

戻り値一覧を次に示します。

表4－13　戻り値一覧

マクロ	数　値	意　味
DEV_OK	0	正常終了
DEV_ERROR	－1	異常終了

4.7.5　データ構造体

USBコミュニケーション・クラス用ドライバが使用するデータ構造体について，次に示します。

(1) USBデバイス・リクエスト構造体

USBデバイス・リクエスト構造体の定義は，USB用ヘッダ・ファイルnectools32¥V850USB_CDC¥src¥USBF¥usbf850.hで行われています。USBデバイス・リクエスト構造体USB_SETUPを次に示します。

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

(2) UARTモード・テーブル構造体

UARTモード・テーブル構造体の定義は，ヘッダ・ファイルnectools32¥V850USB_CDC¥inc¥types.hで行われています。UARTモード・テーブル構造体UART_MODE_TBLを次に示します。

```
typedef struct _UART_MODE_TBL{
         char  DTERate[4]; /*transfer rate(bps)*/
         char  STOPBIT;    /*length of the stop bit - 0:1bit (1:1.5bits) 2:2bits*/
         char  PARITYType; /*parity bit - 0:None 1:Odd 2:Even (3:Mark) 4:Space */
         char  DATABits;   /*data size (number of the bits:5,6,7,8,16) */
} UART_MODE_TBL , *PUART_MODE_TBL;
```

4.7.6　関数解説

(1) 概　　要

この章で説明している各処理プログラムの一覧を次に示します。

注意　“usbf850**”で始まる関数は，**V850E/ME2**に内蔵している**USB**ファンクション・コントローラ用です。“**uartb0850**”で始まる関数は，**V850E/ME2**に内蔵している**UARTB0**用の関数です。**UARTB0**用の関数の詳細は，**4.8.4　**関数解説を参照してください。**

表4－14　サンプル・プログラムの処理プログラム一覧（1/3）

処理プログラム名	関数名	ファイル名	備考
RX850 Pro依存処理プログラム			
CF定義ファイル	－	sys.cf	－
エントリ処理	－	entry.850	アセンブリ言語
ブート処理	boot	boot.850	アセンブリ言語
ハードウエア初期化部	__InitSystemTimer	init.c	C言語
初期化ハンドラ	varfunc	varfunc.c	C言語
ヘッダ・ファイル	－	init.h	－
ボード依存部処理プログラム			
ポートの初期化	port850_reset	port.c	C言語
ヘッダ・ファイル	－	port.h	－
ヘッダ・ファイル			
データ・タイプ宣言	－	types.h	－
戻り値宣言	－	errno.h	－
ビルド・ファイル	－	usb_bus.bld	－
セクション・マップ・ファイル	－	common.lx	－

表4－14　サンプル・プログラムの処理プログラム一覧（2/3）

処理プログラム名	関数名	ファイル名	備考
USBコミュニケーション・クラス用ドライバ処理プログラム（USB処理部）			
初期化関数	usbf850_init	usbf850.c	C言語
割り込みハンドラ（INTUSB0B信号）	usbf850_inthdr	usbf850.c	C言語
割り込みハンドラ（INTUSB1B信号）	usbf850_inthdr1	usbf850.c	C言語
割り込みハンドラ（INTUSB2B信号）	usbf850_inthdr2	usbf850.c	C言語
割り込み処理用タスク（INTUSB0B信号）	task_usb0b	usbf850.c	C言語
割り込み処理用タスク（INTUSB1B信号）	task_usb1b	usbf850.c	C言語
割り込み処理用タスク（INTUSB2B信号）	task_usb2b	usbf850.c	C言語
データ送信関数	usbf850_data_send	usbf850.c	C言語
データ受信関数	usbf850_data_receive	usbf850.c	C言語
Nullデータ送信関数（エンドポイント0）	usbf850_sendnullEP0	usbf850.c	C言語
Stall応答処理関数（エンドポイント0）	usbf850_sendstallEP0	usbf850.c	C言語
Stall応答処理関数（エンドポイント1）	usbf850_bulkin1_stall	usbf850.c	C言語
Stall応答処理関数（エンドポイント2）	usbf850_bulkout1_stall	usbf850.c	C言語
システム・コール呼び出し関数（loc_cpu）	usbf850_loc_cpu	usbf850.c	C言語
システム・コール呼び出し関数（unl_cpu）	usbf850_unl_cpu	usbf850.c	C言語
リクエスト処理関数	usbf850_rxreq	usbf850.c	C言語
リクエスト・データ読み出し関数	usbf850_rxreq_read	usbf850.c	C言語
標準リクエスト処理関数	usbf850_standardreq	usbf850.c	C言語
Get Descriptorリクエスト処理関数	usbf850_getdesc	usbf850.c	C言語
リクエスト処理関数設定用Stall応答処理関数（エンドポイント0）	usbf850_sstall_ctrl	usbf850.c	C言語
USB用ヘッダ・ファイル	－	usbf850.h	－
USB用ディスクリプタ宣言	－	usbf850desc.h	－

表4－14　サンプル・プログラムの処理プログラム一覧（3/3）

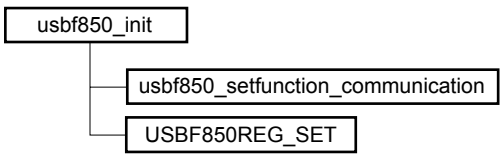
処理プログラム名	関数名	ファイル名	備考
USBコミュニケーション・クラス用ドライバ処理プログラム（USB-UARTインタフェース部）			
USB-UART間インタフェース用初期化関数	usb_cdc_init	usbf850_communication.c	C言語
SEND ENCAPSULATED COMMAND リクエスト処理関数	usbf850_send_encapsulated_command	usbf850_communication.c	C言語
GET ENCAPSULATED RESPONSE リクエスト処理関数	usbf850_get_encapsulated_response	usbf850_communication.c	C言語
SET LINE CODINGリクエスト処理関数	usbf850_set_line_coding	usbf850_communication.c	C言語
GET LINE CODINGリクエスト処理関数	usbf850_get_line_coding	usbf850_communication.c	C言語
SET CONTROL LINE STATEリクエスト処理関数	usbf850_set_control_line_state	usbf850_communication.c	C言語
USB-UARTデータ送信関数	usbf850_usb_to_uart	usbf850_communication.c	C言語
USB-UARTデータ送信関数	usbf850_uart_to_usb	usbf850_communication.c	C言語
USBコミュニケーション・クラス用デバイス・クラス固有のリクエスト処理関数の登録処理関数	usbf850_setfunction_communication	usbf850_communication.c	C言語
USB-UART間インタフェース関数用ヘッダ・ファイル	－	usbf850_ communication.h	－
関数マクロ			
V850E/ME2周辺I/Oレジスタ設定関数（1バイト単位：8ビット）	USBF850REG_SET	usbf850.h	C言語
V850E/ME2周辺I/Oレジスタ読み出し関数（1バイト単位：8ビット）	USBF850REG_READ	usbf850.h	C言語
V850E/ME2周辺I/Oレジスタ設定関数（1ワード単位：16ビット）	USBF850REG_SET_W	usbf850.h	C言語
V850E/ME2周辺I/Oレジスタ読み出し関数（1ワード単位：16ビット）	USBF850REG_READ_W	usbf850.h	C言語

（2）関数ツリー

USBコミュニケーション・クラス用ドライバ処理依存部の呼び出し関係（関数ツリー）を，次に示します。

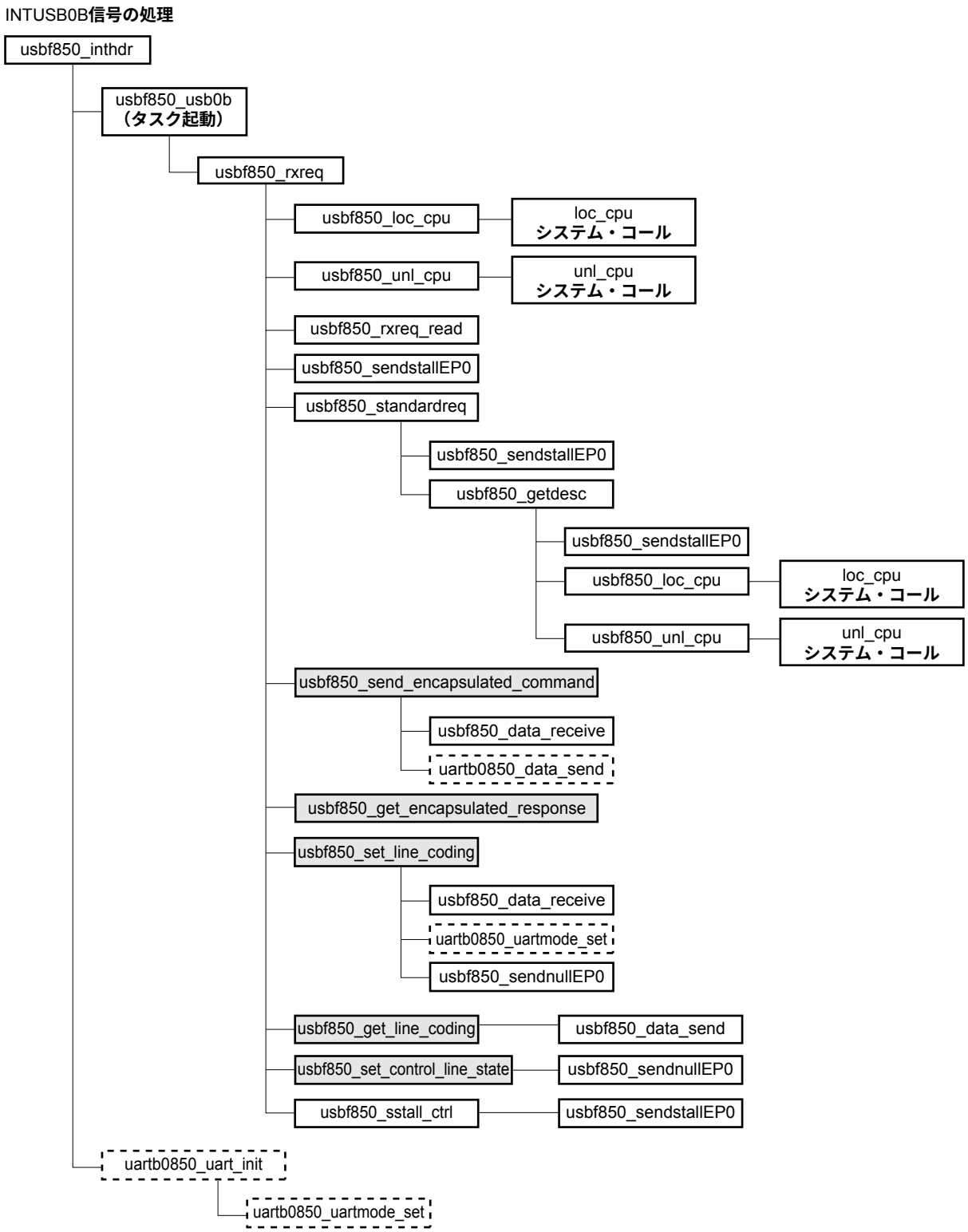
備考　UART処理部の呼び出し関係については，4. 8. 4（2）関数ツリーを参照してください。

図4－14　サンプル・プログラムの関数ツリー（1/3）



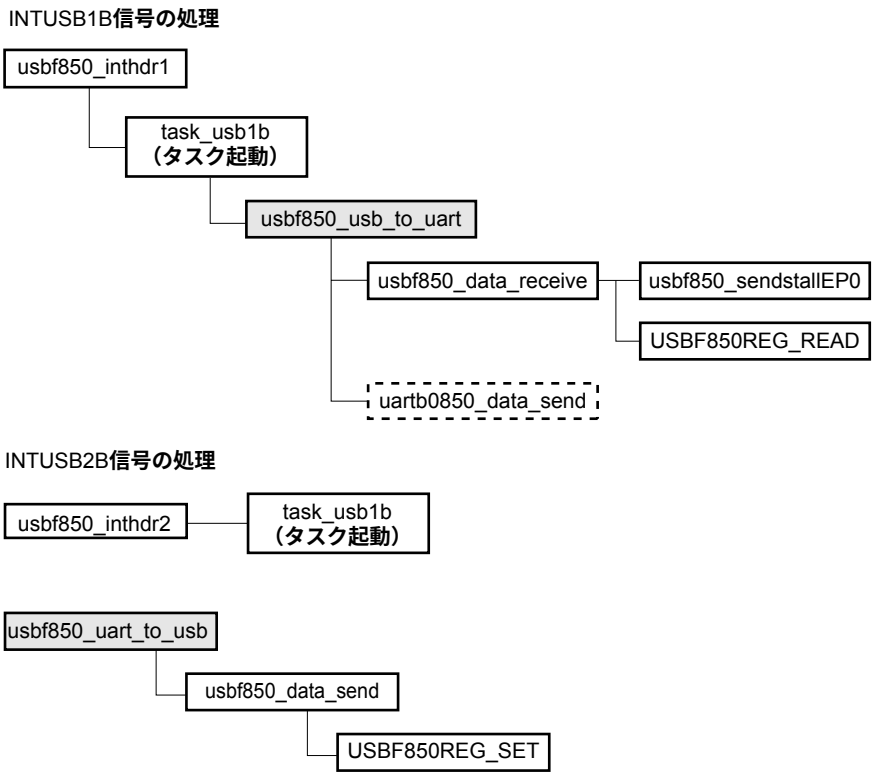
備考　usbf850_initは，初期化ハンドラから呼び出されます。

図4－14　サンプル・プログラムの関数ツリー（2/3）



備考　図中の網掛け部分は，この章で説明しているUSB-UARTインタフェース関数です。図中の破線は，UART処理部用の関数です。UART処理部の関数については，4．8　UART**処理部を参照してください。**

図4－14　サンプル・プログラムの関数ツリー（3/3）



備考　**図中の網掛け部分は，この章で説明しているUSB-UARTインタフェース関数です。図中の破線は，UART処理部用の関数です。UART処理部の関数については，**4. 8　UART**処理部を参照してください。**

(3) 関数解説

このUSBコミュニケーション・クラス用ドライバ処理依存部の関数群について，次の記述フォーマットに従って解説します。

備考　UART処理部の関数群については，4. 8. 4 (3) 関数解説に同じフォーマットで解説します。

XXXX … ①	**発行有効範囲：－－－－** … ②

[概　　要] … ③

－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－

－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－

[C言語形式] … ④

－－－－－－－－－

[パラメータ] … ⑤

I/O	パラメータ	説　　明

[機　　能] … ⑥

－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－

－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－

[戻 り 値] … ⑦

－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－

－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－－

① 名称

関数の名称を示しています。

② 発行有効範囲

関数の発行が可能な処理プログラムの種別を示しています。

タスク	：タスクからだけ発行可能
非タスク	：非タスクからだけ発行可能
タスク｜非タスク	：タスク，非タスクのどちらからも発行可能
－	：割り込みハンドラまたはタスクのため，関数コールできない

③ 概要

関数の機能概要を示しています。

④ C言語形式

関数をC言語で記述された処理プログラムから発行するときの記述形式を示しています。

⑤ パラメータ

関数のパラメータを次の形式で示しています。

I/O	パラメータ	説　明
A	B	C

A ：パラメータの種類

I …USBファンクション・コントローラへの入力パラメータ

O …USBファンクション・コントローラからの出力パラメータ

B ：パラメータのデータ・タイプ

C ：パラメータの説明

⑥ 機能

関数の機能詳細を示しています。

⑦ 戻り値

関数からの戻り値を，データ・マクロおよび数値で示しています。

usbf850_init　　発行有効範囲：非タスク｜タスク

[概　　要]

V850E/ME2に内蔵しているUSBファンクション・コントローラの初期化処理を行う。

[C言語形式]

```
void usbf850_init (void)
```

[パラメータ]

I/O	パラメータ	説　　明
–	–	–

[機　　能]

ソフトウエア初期化部から呼び出され，V850E/ME2に内蔵しているUSBファンクション・コントローラの初期化処理を行います。

備考　初期化処理についての詳細は，4. 7. 2（1）**初期化処理**を参照してください。

[戻 り 値]

なし

usbf850_inthdr

発行有効範囲：－

[概　　要]

V850E/ME2に内蔵しているUSBファンクション・コントローラ用割り込みハンドラ（INTUSB0B信号用）。

[C言語形式]

```
ID usbf850_inthdr (void)
```

[パラメータ]

I/O	パラメータ	説　　明
－	－	－

[機　　能]

INTUSB0B信号（USBファンクション・ステータス0）により起動される割り込みハンドラです。

サンプル・プログラムでは，割り込み要因を調べCPUDEC割り込みだった場合だけ，割り込み処理用のタスク（task_usb0b）を起動します。SETRQ割り込みだった場合は，USB-UARTインタフェース用初期化処理関数（uartb0850_uart_init）を呼び出します。このハンドラは，CF定義ファイルで定義されています。

備考　割り込み処理についての詳細は，4. 7. 2（2）**割り込み処理**を参照してください。

[戻 り 値]

オブジェクトID番号（タスクのID番号）

usbf850_inthdr1

発行有効範囲：－

[概　　要]

V850E/ME2に内蔵しているUSBファンクション・コントローラ用割り込みハンドラ（INTUSB1B信号用）。

[C言語形式]

```
ID usbf850_inthdr1 (void)
```

[パラメータ]

I/O	パラメータ	説　　明
－	－	－

[機　　能]

INTUSB1B信号（USBファンクション・ステータス1）により起動される割り込みハンドラです。

サンプル・プログラムでは，割り込み処理用のタスク（task_usb1b）を起動します。このハンドラは，CF定義ファイルで定義されています。

備考　割り込み処理についての詳細は，4. 7. 2（2）**割り込み処理**を参照してください。

[戻 り 値]

オブジェクトID番号（タスクのID番号）

usbf850_inthdr2

発行有効範囲：－

[概　　要]

V850E/ME2に内蔵しているUSBファンクション・コントローラ用割り込みハンドラ（INTUSB2B信号用）。

[C言語形式]

```
ID usbf850_inthdr2 (void)
```

[パラメータ]

I/O	パラメータ	説　　明
－	－	－

[機　　能]

INTUSB2B信号（USBファンクション・ステータス2）により起動される割り込みハンドラです。

サンプル・プログラムでは，割り込み処理用のタスク（task_usb2b）を起動します。このハンドラは，CF定義ファイルで定義されています。

注意　このサンプル・プログラムでは，INTUSB2B信号により通知される割り込みについては，すべてマスクされているため，このハンドラは呼ばれません。

[戻 り 値]

オブジェクトID番号（タスクのID番号）

task_usb0b　　発行有効範囲：－

[概　　要]

INTUSB0B信号による割り込み処理を行う。

[C言語形式]

```
void task_usb0b (VP exinf)
```

[パラメータ]

I/O	パラメータ	説　　明
I	VP exinf	拡張情報

対象タスクに関する“ユーザ独自の情報”を格納するための領域で，ユーザが自由に利用できます。
exinfに設定された情報は，処理プログラム（タスク，非タスク）からref_tskシステム・コールを発行することにより，ダイナミックに獲得できます。

備考　システム・コールについての詳細は，RX850 Pro　**ユーザーズ・マニュアル　基礎編**を参照してください。

[機　　能]

INTUSB0B割り込み信号（USBファンクション・ステータス0割り込み）用の割り込みハンドラから起動されるタスクです。サンプル・プログラムでは，usbf850_rxreq関数を呼び出し，USB標準デバイス・リクエストおよびデバイス・クラス固有のリクエストの処理を行います。

注意　このサンプル・プログラムでは，V850E/ME2に内蔵しているUSBファンクション・コントローラで自動応答しない標準デバイス・リクエスト「Get Descriptor（String Descriptor）」だけを処理します。

備考　割り込み処理についての詳細は，4. 7. 2（2）**割り込み処理**を参照してください。

[戻 り 値]

なし

task_usb1b

発行有効範囲：－

[概　　要]

INTUSB1B信号による割り込み処理を行う。

[C言語形式]

```
void task_usb1b (VP exinf)
```

[パラメータ]

I/O	パラメータ	説　　明
I	VP exinf	拡張情報

対象タスクに関する“ユーザ独自の情報”を格納するための領域で，ユーザが自由に利用できます。
exinfに設定された情報は，処理プログラム（タスク，非タスク）からref_tskシステム・コールを発行することにより，ダイナミックに獲得できます。

備考　システム・コールについての詳細は，RX850 Pro　**ユーザーズ・マニュアル　基礎編**を参照してください。

[機　　能]

INTUSB1B割り込み信号（USBファンクション・ステータス1割り込み）用の割り込みハンドラから起動されるタスクです。サンプル・プログラムでは，割り込み要因を確認し，割り込み要因がBKO2DTの場合，USBからUARTへのデータ転送関数（usbf850_usb_to_uart）を呼び出します。

備考　割り込み処理についての詳細は，4. 7. 2（2）**割り込み処理**を参照してください。

[戻 り 値]

なし

task_usb2b　　発行有効範囲：－

[概　　要]

INTUSB2B信号による割り込み処理を行う。

[C言語形式]

```
void task_usb2b (VP exinf)
```

[パラメータ]

I/O	パラメータ	説　　明
I	VP exinf	拡張情報

対象タスクに関する“ユーザ独自の情報”を格納するための領域で，ユーザが自由に利用できます。
exinfに設定された情報は，処理プログラム（タスク，非タスク）からref_tskシステム・コールを発行することにより，ダイナミックに獲得できます。

備考　システム・コールについての詳細は，RX850 Pro　**ユーザーズ・マニュアル　基礎編**を参照してください。

[機　　能]

INTUSB2B割り込み信号（USBファンクション・ステータス2割り込み）用の割り込みハンドラから起動されるタスクです。サンプル・プログラムでは該当する処理がないため，未処理で戻ります。

注意　このサンプル・プログラムでは，この関数を使用していません。

[戻 り 値]

なし

usbf850_data_send　　発行有効範囲：非タスク｜タスク

[概　　要]

USBファンクション・コントローラ用データ送信関数。

[C言語形式]

```
long usbf850_data_send (unsigned char* data, long len, char ep)
```

[パラメータ]

I/O	パラメータ	説　　明
I	unsigned char* data	送信データの先頭アドレス
I	long len	データ・サイズ
I	char ep	エンドポイント番号

[機　　能]

dataで指定されたアドレスから，lenで指定されたサイズ分のデータを，epで指定されたエンドポイントで送信処理を行います。

[戻 り 値]

送信時のステータス

DEV_ERROR ：エンドポイント番号が不正

DEV_OK　　：正常終了

usbf850_data_receive　　発行有効範囲：非タスク｜タスク

[概　　要]

USBファンクション・コントローラ用データ受信関数。

[C言語形式]

```
long usbf850_data_receive (unsigned char* data, long len, char ep)
```

[パラメータ]

I/O	パラメータ	説　　明
I	unsigned char* data	受信データ用バッファの先頭アドレス
I	long len	データ・サイズ
I	char ep	エンドポイント番号

[機　　能]

epで指定されたエンドポイントのバッファから，lenで指定されたサイズ分データを読み出し，dataで指定されたアドレスに格納します。

[戻 り 値]

受信時のステータス

DEV_ERROR ：受信データ・サイズが不正，またはエンドポイント番号が不正

DEV_OK　　：正常終了

usbf850_sendnullEP0

発行有効範囲：非タスク｜タスク

[概　　要]

コントロール・エンドポイント（エンドポイント0）用Nullデータ送信関数。

[C言語形式]

```
void usbf850_sendnullEP0 (void)
```

[パラメータ]

I/O	パラメータ	説　　明
–	–	–

[機　　能]

コントロール・エンドポイント（エンドポイント0）で，Nullデータ（データ・サイズ0のデータ）の送信処理を行います。

[戻 り 値]

なし

usb850_sendstallEP0

発行有効範囲：非タスク｜タスク

[概　　要]

コントロール・エンドポイント（エンドポイント0）用STALL応答関数。

[C言語形式]

```
void usbf850_sendstallEP0 (void)
```

[パラメータ]

I/O	パラメータ	説　　明
–	–	–

[機　　能]

コントロール・エンドポイント（エンドポイント0）でSTALL応答を設定します。

[戻 り 値]

なし

usbf850_bulkin1_stall

発行有効範囲：非タスク｜タスク

[概　　要]

バルク・エンドポイント（エンドポイント1）用STALL応答関数。

[C言語形式]

```
void usbf850_bulkin1_stall (void)
```

[パラメータ]

I/O	パラメータ	説　　明
−	−	−

[機　　能]

バルク・エンドポイント（エンドポイント1）でSTALL応答を設定します。

注意　このサンプル・プログラムでは，この関数を使用していません。

[戻 り 値]

なし

usbf850_bulkout1_stall　　発行有効範囲：非タスク｜タスク

[概　　要]

バルク・エンドポイント（エンドポイント2）用STALL応答関数。

[C言語形式]

```
void usbf850_bulkout1_stall (void)
```

[パラメータ]

I/O	パラメータ	説　　明
−	−	−

[機　　能]

バルク・エンドポイント（エンドポイント2）でSTALL応答を設定します。

注意　このサンプル・プログラムでは，この関数を使用していません。

[戻 り 値]

なし

usbf850_loc_cpu　　発行有効範囲：タスク

[概　　要]

マスカブル割り込みの受け付けとディスパッチ処理を禁止する。

[C言語形式]

```
void usbf850_loc_cpu (void)
```

[パラメータ]

I/O	パラメータ	説　　明
–	–	–

[機　　能]

loc_cpuシステム・コールを呼び出します。

備考　システム・コールについての詳細は，RX850 Pro　**ユーザーズ・マニュアル　基礎編**を参照してください。

[戻 り 値]

なし

usbf850_unl_cpu　　発行有効範囲：タスク

[概　　要]

マスカブル割り込みの受け付けとディスパッチ処理を許可する。

[C言語形式]

```
void usbf850_unl_cpu (void)
```

[パラメータ]

I/O	パラメータ	説　　明
–	–	–

[機　　能]

unl_cpuシステム・コールを呼び出します。

備考　システム・コールについての詳細は，RX850 Pro　**ユーザーズ・マニュアル　基礎編**を参照してください。

[戻 り 値]

なし

usbf850_rxreq

発行有効範囲：非タスク｜タスク

[概　　要]

USBリクエスト処理を行う。

[C言語形式]

```
void usbf850_rxreq (void)
```

[パラメータ]

I/O	パラメータ	説　　明
–	–	–

[機　　能]

INTUSB0B割り込み信号により起動されるtask_usb0bタスクによって，呼び出されます。SETUPデータの読み出し処理を呼び出し，読み出したデータを解析します。解析結果に基づき，USBのリクエスト処理を呼び出します。

[戻 り 値]

なし

usbf850_rxreq_read

発行有効範囲：非タスク｜タスク

[概　　要]

USBリクエスト・データを読み出す。

[C言語形式]

```
void usbf850_rxreq_read (void)
```

[パラメータ]

I/O	パラメータ	説　　明
–	–	–

[機　　能]

コントロール転送（エンドポイント0）で，Setupトークンに続いて受信されるSETUPデータを読み出します。

SETUPデータは，通常のデータと区別され専用のレジスタに格納されており，必ず8バイト・リードします。

[戻 り 値]

なし

usbf850_standardreq

発行有効範囲：非タスク｜タスク

[概　　要]

USB標準リクエストの処理を行う。

[C言語形式]

```
void usbf850_standardreq (void)
```

[パラメータ]

I/O	パラメータ	説　　明
–	–	–

[機　　能]

SETUPデータ読み出し後，リクエストの内容が標準リクエストであった場合に呼び出されます。Get Descriptor リクエストであることを確認し，usbf850_getdesc関数を呼び出します。

[戻 り 値]

なし

usbf850_getdesc　　発行有効範囲：非タスク｜タスク

[概　　要]

USB標準リクエストのGet Descriptor（String Descriptor）の処理を行う。

[C言語形式]

```
void usbf850_getdesc (void)
```

[パラメータ]

I/O	パラメータ	説　　明
–	–	–

[機　　能]

usbf850_standardreq関数から呼び出され，USB標準リクエストのGet Descriptor（String Descriptor）の処理を行います。Get Descriptor（String Descriptor）リクエスト以外の場合は，STALL応答します。

[戻 り 値]

なし

usbf850_sstall_ctrl　　発行有効範囲：非タスク｜タスク

[概　　要]

コントロール・エンドポイント（エンドポイント0）用STALL応答処理関数。

[C言語形式]

```
void usbf850_sstall_ctrl (void)
```

[パラメータ]

I/O	パラメータ	説　　明
–	–	–

[機　　能]

コントロール・エンドポイント（エンドポイント0）でSTALL応答を設定します。

usbf850_setfunction_communication**関数で，クラス・リクエスト処理関数を関数ポインタとして配列に準備するときに，配列の添え字にリクエスト・コードを使用します。該当するリクエストがない部分にこの関数を登録することで，サポートしないリクエスト・コードが来た場合に**STALL**応答するよう設定されます。**

[戻 り 値]

なし

usbf850_send_encapsulated_command　　発行有効範囲：非タスク｜タスク

[概　　要]

USBコミュニケーション・クラス固有のリクエスト（SEND ENCAPSULATED COMMAND）処理関数。

[C言語形式]

```
void usbf850_send_encapsulated_command (void)
```

[パラメータ]

I/O	パラメータ	説　　明
–	–	–

[機　　能]

SEND ENCAPSULATED COMMANDリクエストの処理を行います。ホストから受けたコマンド・データをCDCデバイスに引き渡します。サンプル・プログラムでは，このリクエストを受け取ると，リクエストに続いてホストから通知されるデータを受信し，そのデータをUARTへ送信しNULL応答します。

[戻 り 値]

なし

usbf850_get_encapsulated_response　　発行有効範囲：非タスク｜タスク

[概　　要]

USBコミュニケーション・クラス固有のリクエスト（GET ENCAPSULATED RESPONSE）処理関数。

[C言語形式]

```
void usbf850_get_encapsulated_response (void)
```

[パラメータ]

I/O	パラメータ	説　　明
–	–	–

[機　　能]

GET ENCAPSULATED RESPONSEリクエストの処理を行います。CDCデバイスから受けたコマンド・データをホストに引き渡します。サンプル・プログラムでは，このリクエストを受け取ると，何も処理せず正常終了します。

[戻 り 値]

なし

usbf850_set_line_coding　　発行有効範囲：非タスク｜タスク

[概　　要]

USBコミュニケーション・クラス固有のリクエスト（SET LINE CODING）処理関数。

[C言語形式]

```
void usbf850_set_line_coding (void)
```

[パラメータ]

I/O	パラメータ	説　　明
–	–	–

[機　　能]

SET LINE CODINGリクエストの処理を行います。転送速度，ストップ・ビット，パリティ・ビット，データ長を設定します。サンプル・プログラムでは，このリクエストを受け取ると，リクエストに続いてホストから通知されるデータを受信し，そのデータをUART_MODE_INFO構造体に書き込みます。さらに，UART_MODE_INFO構造体に書き込まれた値をもとにUARTのモードを設定し，NULL応答します。

[戻 り 値]

なし

usbf850_get_line_coding

発行有効範囲：非タスク｜タスク

[概　　要]

USBコミュニケーション・クラス固有のリクエスト（GET LINE CODING）処理関数。

[C言語形式]

```
void usbf850_get_line_coding (void)
```

[パラメータ]

I/O	パラメータ	説　　明
–	–	–

[機　　能]

GET LINE CODINGリクエストの処理を行います。転送速度，ストップ・ビット，パリティ・ビット，データ長の現在の設定値をホストへ送信します。サンプル・プログラムでは，このリクエストを受け取ると，UART_MODE_INFO構造体に設定されている現在のUARTの設定値をホストへ送信します。

[戻 り 値]

なし

usbf850_set_control_line_state　　発行有効範囲：非タスク｜タスク

[概　　要]

USBコミュニケーション・クラス固有のリクエスト（SET CONTROL LINE STATE）処理関数。

[C言語形式]

```
void usbf850_set_control_line_state (void)
```

[パラメータ]

I/O	パラメータ	説　　明
–	–	–

[機　　能]

SET CONTROL LINE STATEリクエストの処理を行います。RS-232/V.24のコントロール信号（RTS/DTR）を設定します。ただし，V850E/ME2ではRTS/DTR端子を持っていないため，設定できません。このため，サンプル・プログラムでは，NULL応答し正常終了します。

[戻 り 値]

なし

usbf850_usb_to_uart

発行有効範囲：非タスク｜タスク

[概　　要]

USBからUART方向へのデータ転送関数。

[C言語形式]

```
void usbf850_usb_to_uart (void)
```

[パラメータ]

I/O	パラメータ	説　　明
–	–	–

[機　　能]

USBのデータ受信処理関数（usbf850_data_receive）を呼び出し，データの受信処理を行います。その後，UARTへのデータ送信関数（usbf850_cdc_data_send）を呼び出し，UARTへデータを送信します。

[戻 り 値]

なし

usbf850_uart_to_usb　　発行有効範囲：非タスク｜タスク

[概　　要]

UARTからUSB方向へのデータ転送関数。

[C言語形式]

```
void usbf850_uart_to_usb (unsigned char* data, int len)
```

[パラメータ]

I/O	パラメータ	説　　明
I	unsigned char* data	送信データの先頭アドレス
I	int	lenデータ・サイズ

[機　　能]

USBのデータ送信関数（usbf850_data_send）を呼び出し，UARTの受信データをUSBへ送信します。

[戻 り 値]

なし

usbf850_setfunction_communication　　発行有効範囲：非タスク｜タスク

[概　　要]

USBコミュニケーション・クラス固有のリクエスト処理関数を，関数ポインタとして配列に登録する処理を行う。

[C言語形式]

```
void usbf850_setfunction_communication (void)
```

[パラメータ]

I/O	パラメータ	説　　明
−	−	−

[機　　能]

USBの初期化処理から呼ばれ，USBコミュニケーション・クラス固有のリクエスト処理関数を，関数ポインタとして配列（配列名：Req_Func_C）に登録します。

サポートしないリクエスト・コードには，usbf850_sstall_ctrl関数を登録し，サポートしないリクエストが来た場合にSTALL応答するよう設定します。

[戻 り 値]

なし

USBF850REG_SET　　発行有効範囲：非タスク｜タスク

[概　　要]

V850E/ME2周辺I/Oレジスタ設定関数（1バイト単位：8ビット）。

[C言語形式]

```
USBF850REG_SET (offset, val)
```

[パラメータ]

I/O	パラメータ	説　　明
I	offset	周辺I/Oレジスタのアドレス
I	val	設定用データ

[機　　能]

V850E/ME2周辺I/Oレジスタ（offsetで指定されたレジスタ・アドレス）へ，valで指定されたデータを設定します。なお，このマクロは，1バイト（8ビット）単位でアクセス可能なレジスタだけ使用できます。

[戻 り 値]

なし

USBF850REG_READ

発行有効範囲：非タスク｜タスク

[概　　要]

V850E/ME2周辺I/Oレジスタ読み出し関数（1バイト単位：8ビット）。

[C言語形式]

```
USBF850REG_READ (offset)
```

[パラメータ]

I/O	パラメータ	説　　明
I	offset	周辺I/Oレジスタのアドレス

[機　　能]

V850E/ME2周辺I/Oレジスタ（offsetで指定されたレジスタ・アドレス）の値を読み出します。なお，このマクロは，1バイト（8ビット）単位でアクセス可能なレジスタに限ります。

[戻 り 値]

なし

USBF850REG_SET_W

発行有効範囲：非タスク｜タスク

[概　　要]

V850E/ME2周辺I/Oレジスタ設定関数（1ワード単位：16ビット）。

[C言語形式]

```
USBF850REG_SET_W (offset, val)
```

[パラメータ]

I/O	パラメータ	説　　明
I	offset	周辺I/Oレジスタのアドレス
I	val	設定用データ

[機　　能]

V850E/ME2周辺I/Oレジスタ（offsetで指定されたレジスタ・アドレス）へ，valで指定されたデータを設定します。なお，このマクロは，1ワード（16ビット）単位でアクセス可能なレジスタだけ使用できます。

[戻 り 値]

なし

USBF850REG_READ_W

発行有効範囲：非タスク｜タスク

[概　　要]

V850E/ME2周辺I/Oレジスタ読み出し関数（1ワード単位：16ビット）。

[C言語形式]

```
USBF850REG_READ_W (offset)
```

[パラメータ]

I/O	パラメータ	説　　明
I	offset	周辺I/Oレジスタのアドレス

[機　　能]

V850E/ME2周辺I/Oレジスタ（offsetで指定されたレジスタ・アドレス）の値を読み出します。なお，このマクロは，1ワード（16ビット）単位でアクセス可能なレジスタに限ります。

[戻 り 値]

なし

4. 8　UART処理部

4. 8. 1　概　　要

このサンプル・プログラムでは，UART処理部として，V850E/ME2に内蔵しているUARTB0を動作させるための簡易ドライバが用意されています。そこで，この節章では，UART処理部について示します。

UART処理部の一覧を次に示します。

注意　V850E/ME2に内蔵しているUARTの処理プログラムを用意していますが，サンプル・プログラム用に最低限の処理を行うものです。汎用のUART用ドライバとしては，動作を保証しません。

備考　USBコミュニケーション・クラス用ドライバ処理依存部の詳細については，4. 7　USBコミュニケーション・クラス用ドライバの機能を参照してください。

・UARTの初期化処理

RX850 Proのソフトウエア初期化部から呼び出され，UARTB0の初期化処理を行います。

・UARTの割り込みハンドラ

UARTの割り込みが発生するごとに呼び出される割り込み処理専用ルーチンであり，CF定義ファイルに定義されています。

注意　このサンプル・プログラムでは，必要な割り込み以外はマスクしてあります。
このサンプル・プログラムで使用する割り込みは，次の3つです。

- UBTIRE信号で通知されるUARTB0受信エラー割り込み
- UBTIR0信号で通知されるUARTB0受信完了割り込み
- UBTITO0信号で通知されるUARTB0受信タイムアウト割り込み

・UARTの割り込み処理タスク

UARTの割り込みハンドラから呼び出され，データの受信処理を行います。

・UART用汎用関数

UART処理部で使用される汎用関数として，UARTのデータ送信関数や，動作モードの設定関数が用意されています。

4. 8. 2　処理の流れ

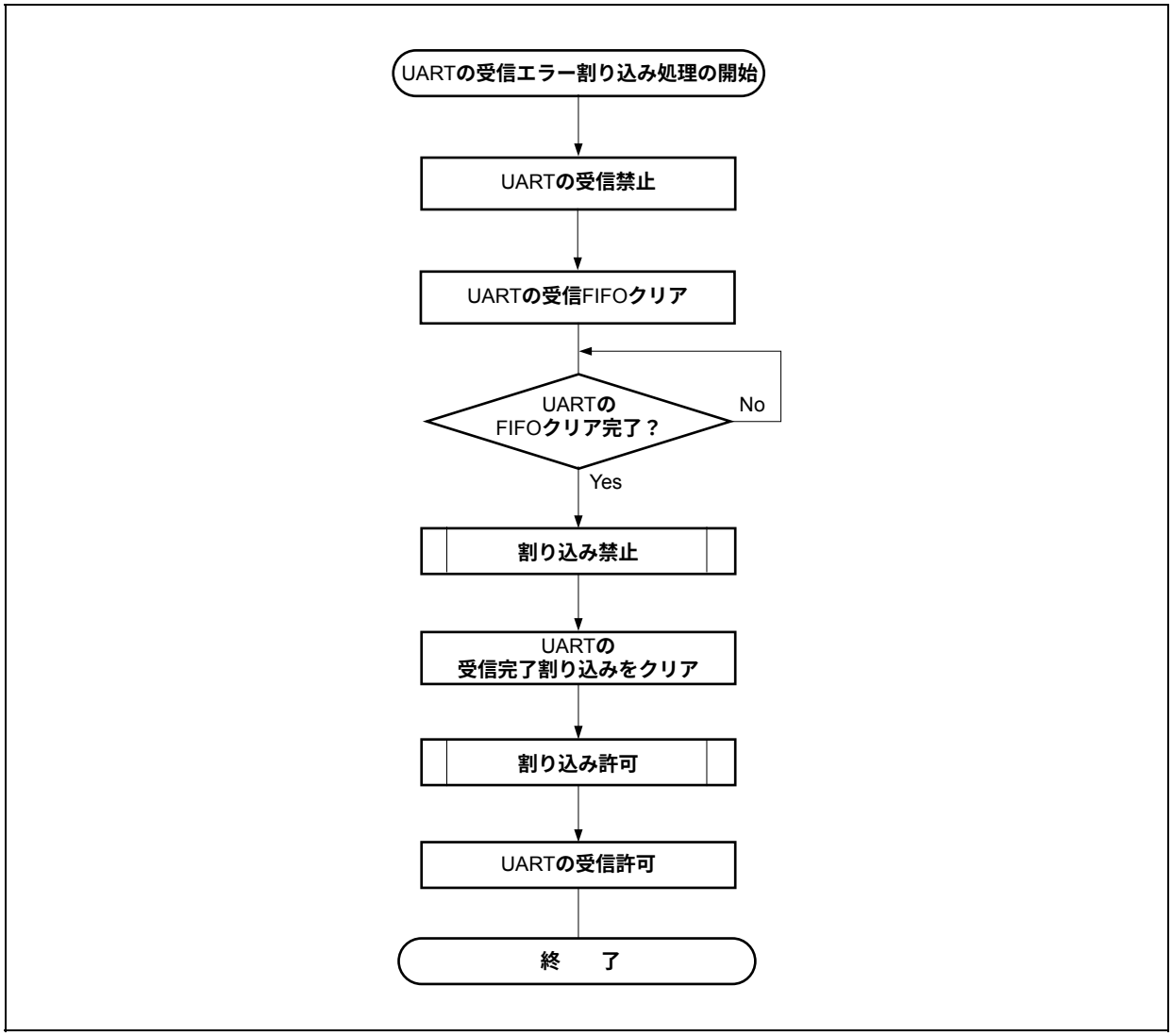
（1）割り込み処理

サンプル･プログラムでは，初期化完了後，割り込みイベントによって動作します。イベントがない場合は，常にアイドル状態です。ただし，割り込みはUARTからだけでなく，USBファンクション・コントローラからも通知されます。

サンプル・プログラムにおけるUARTの割り込み処理の流れを，次に示します。

備考　**USBファンクション・コントローラの割り込み処理については，**4. 7. 2**（2）割り込み処理を参照してください。**

図4－15　割り込み処理のフロー・チャート（1）



UBTIRE0信号による割り込みでのサンプル・プログラムの処理内容を，次に示します。

・UARTの受信禁止

UB0CTL0.UB0RXEビットにより，UARTの受信動作を禁止します。

・UARTの受信FIFOクリア

UB0FIC0レジスタにより，受信FIFOをクリアします。

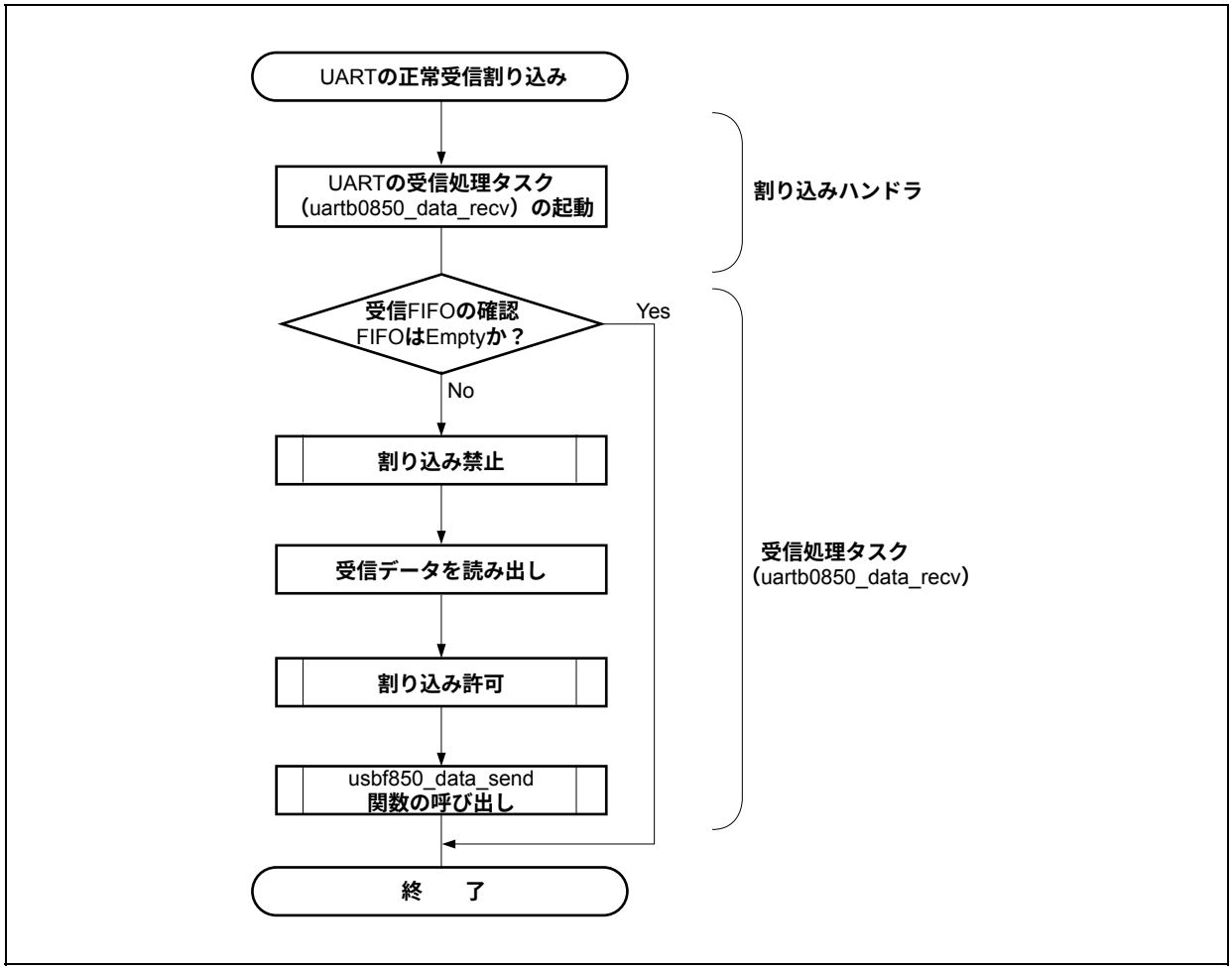
・UARTの受信完了割り込みのクリア

URIC0レジスタにより，UARTの受信完了割り込みをクリアします。

・UARTの受信許可

UB0CTL0.UB0RXEビットにより，UARTの受信動作を許可します。

図4－16　割り込み処理のフロー・チャート（2）



UBTIR0信号による割り込みでのサンプル・プログラムの処理内容を，次に示します。

・UARTの受信処理タスクの起動

UARTの受信処理タスク（uartb0850_data_recv）を起動します。

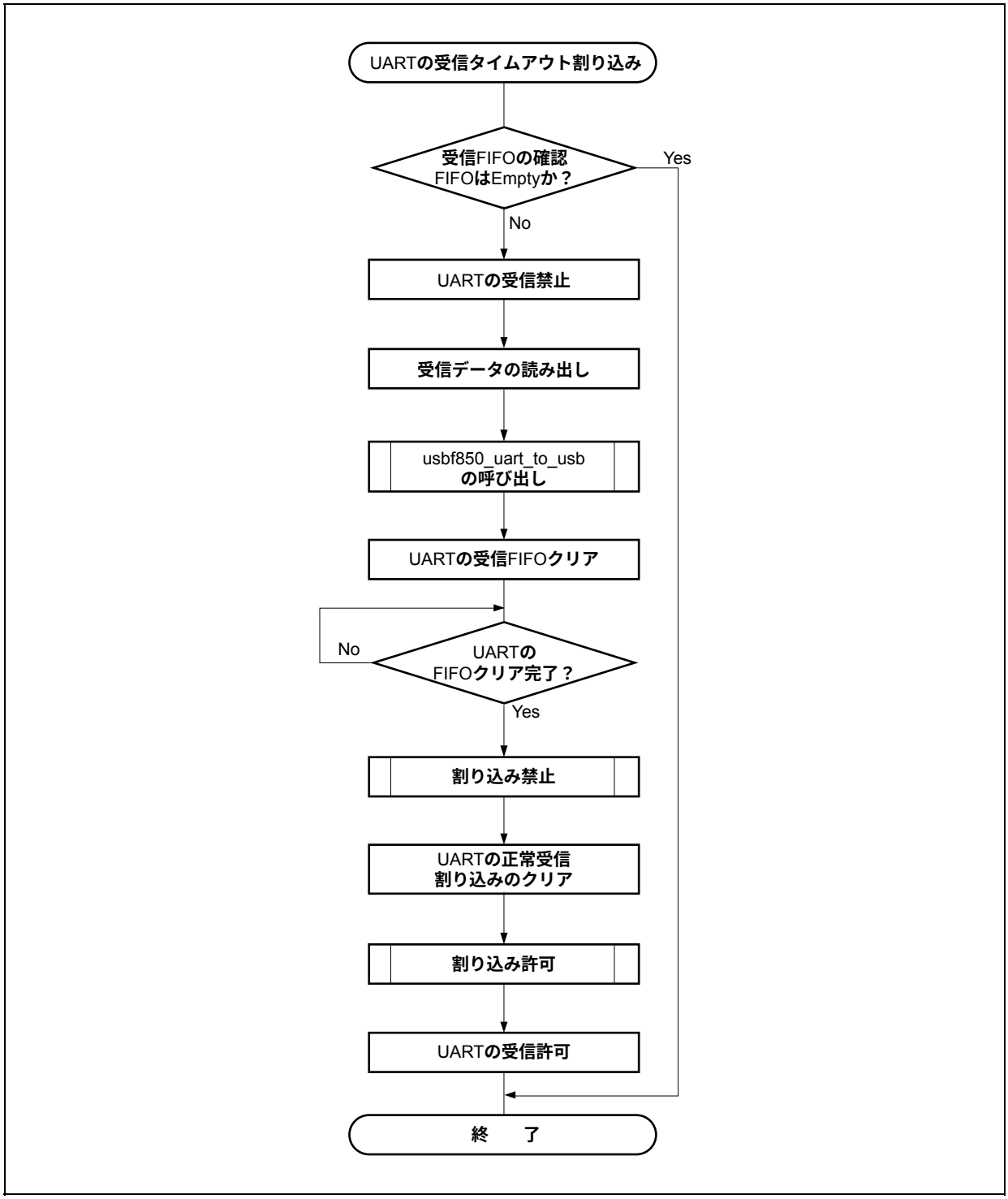
・UARTの受信FIFOの確認

UARTの受信FIFOにデータが格納されているかを確認し，受信データがあれば，データを読み出します。受信データがなければ処理を終了します。

・usbf850_data_send関数の呼び出し

読み出した受信データをホストへ送信します。

図4－17　割り込み処理のフロー・チャート（3）



UBTITO0信号による割り込みでのサンプル・プログラムの処理内容を，次に示します。

・UARTの受信FIFOの確認

UARTの受信FIFOにデータが格納されているかを確認し，受信データがあれば，UARTの受信を禁止します。データがなければ処理を終了します。

・UARTの受信禁止

UB0CTL0.UB0RXEビットにより，UARTの受信動作を禁止します。

・UARTの受信データの読み出し

UARTの受信FIFOからデータを読み出します。

・usbf850_uart_to_usb関数の呼び出し

読み出した受信データをホストへ送信します。

・UARTの受信FIFOクリア

UB0FIC0レジスタにより，受信FIFOをクリアします。

・UARTの受信完了割り込みのクリア

URIC0レジスタにより，UARTの受信完了割り込みをクリアします。

・UARTの受信許可

UB0CTL0.UB0RXEビットにより，UARTの受信動作を許可します。

4. 8. 3　動作モード

サンプル・プログラムでは，UARTの動作モードとして有効な設定値を一部限定しています。

サンプル・プログラムでの有効な設定値を次に示します。

表4－15　UARTの設定値一覧

設定内容	設定値	備　　考
転送速度	9,600 bps	－
	19,200 bps	初期設定値
	38,400 bps	－
	57,600 bps	－
	115,200 bps	－
	上記以外	9,600 bps に設定されます
パリティ・ビット	なし	初期設定値
	奇数	－
	偶数	－
	スペース	－
	上記以外	スペースとして設定されます
データ長	7 ビット	－
	8 ビット	初期設定値
	上記以外	8 ビットに設定されます
ストップ・ビット	1 ビット	初期設定値
	2 ビット	
	上記以外	2 ビットに設定されます

4.8.4　関数解説

(1) 概　　要

サンプル・プログラムにおけるUART処理部の各処理プログラムの一覧を，次に示します。

注意　“usbf850”で始まる関数は，V850E/ME2に内蔵しているUSBファンクション・コントローラ用です。“uartb0850”で始まる関数は，V850E/ME2に内蔵しているUARTB0用の関数です。USBファンクション・コントローラ用の関数の詳細は，4.7.6　関数解説を参照してください。

表4－16　UART処理部の各処理プログラム一覧

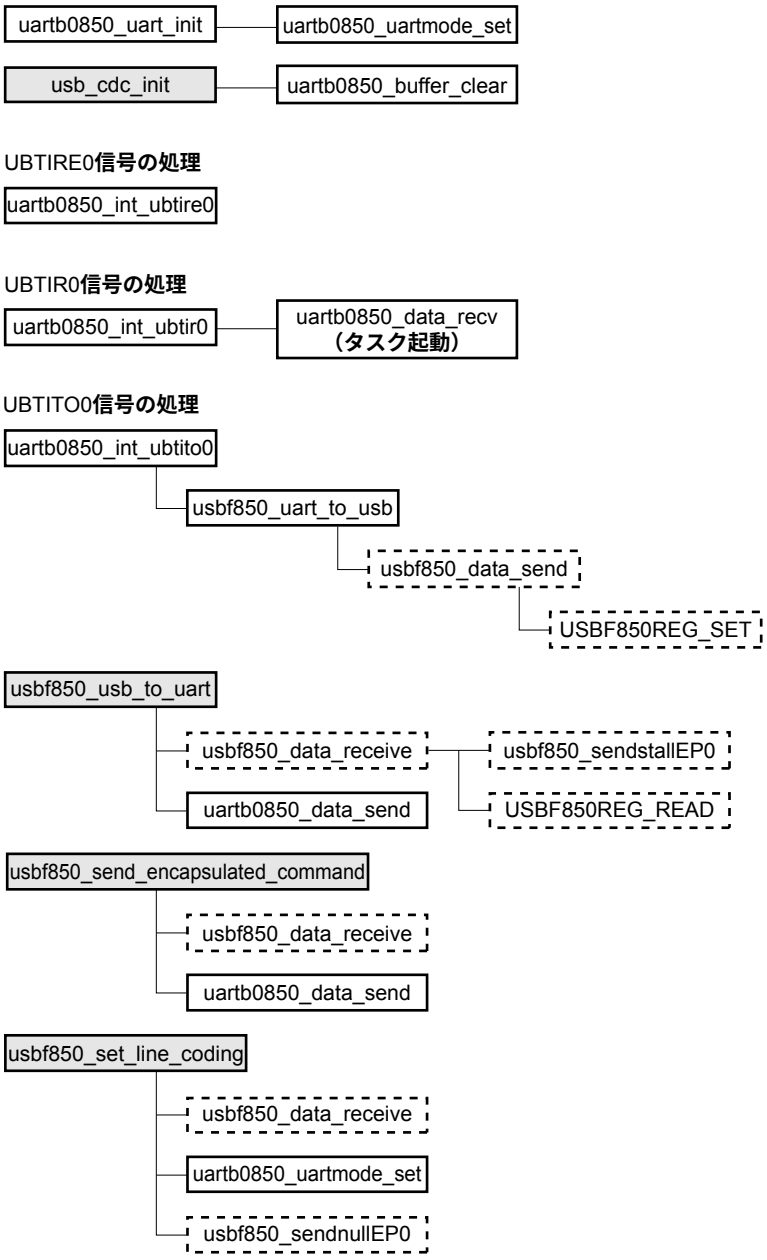
処理プログラム名	関数名	ファイル名	備考
UART処理プログラム			
UART初期化関数	uartb0850_enable	uart_ctrl.c	C言語
UART設定初期化関数	uartb0850_uart_init	uart_ctrl.c	C言語
割り込み処理用タスク（UBTIR0信号処理用）	uartb0850_data_recv	uart_ctrl.c	C言語
データ送信関数	uartb0850_data_send	uart_ctrl.c	C言語
UARTのFIFOクリア関数	uartb0850_buffer_clear	uart_ctrl.c	C言語
UARTのモード設定関数	uartb0850_uartmode_set	uart_ctrl.c	C言語
割り込みハンドラ（UBTIRE0信号：UARTB0受信エラー）	uartb0850_int_ubtire0	uart_ctrl.c	C言語
割り込みハンドラ（UBTIR0信号：UARTB0受信完了）	uartb0850_int_ubtir0	uart_ctrl.c	C言語
割り込みハンドラ(UBTITO0信号:UARTB0受信タイムアウト)	uartb0850_int_ubtito0	uart_ctrl.c	C言語
UART用ヘッダ・ファイル	－	uart_ctrl.h	－

(2) 関数ツリー

サンプル・プログラムのUART処理部の呼び出し関係（関数ツリー）を，次に示します。

備考　USBコミュニケーション・クラス用ドライバ処理依存部の呼び出し関係については，4.7.6 (2) 関数ツリーを参照してください。

図4－18　サンプル・プログラムのUART処理部の関数ツリー



注意　uartb0850_enable**および**usb_cdc_init**は，初期化ハンドラから呼び出されます。**

備考　**図中の網掛け部分は，**USB-UART**インタフェース関数です。図中の破線は，**USB**コミュニケーション・クラス用ドライバ処理依存部の関数です。**USB**コミュニケーション・クラス用ドライバ処理依存部の関数および**USB-UART**インタフェース関数については，**4. 7　USB**コミュニケーション・クラス用ドライバの機能を参照してください。**

(3) 関数解説

サンプル・プログラムのUART処理部の関数群について，4. 7. 6**(3) 関数解説の記述フォーマットにしたがって解説します。**

uartb0850_enable　　発行有効範囲：非タスク｜タスク

[概　　要]

UARTの初期化関数。

[C言語形式]

```
void uartb0850_enable (void)
```

[パラメータ]

I/O	パラメータ	説　　明
–	–	–

[機　　能]

UARTの初期化処理として，ポートの設定，割り込みの許可などを行います。

[戻 り 値]

なし

uartb0850_uart_init　　発行有効範囲：非タスク｜タスク

[概　　要]

UARTの設定初期化関数。

[C言語形式]

```
void uartb0850_uart_init (void)
```

[パラメータ]

I/O	パラメータ	説　　明
–	–	–

[機　　能]

UARTを次に示した初期設定値で設定します。

- 転送速度　　：19,200 bps
- データ・ビット　：8ビット
- パリティ　　：なし
- ストップ・ビット　：1ビット

[戻 り 値]

なし

uartb0850_data_recv　　発行有効範囲：－

[概　　要]

UARTのUBTIR0信号による割り込み処理を行う。

[C言語形式]

```
void uartb0850_data_recv (VP exinf)
```

[パラメータ]

I/O	パラメータ	説　　明
I	VP exinf	拡張情報

対象タスクに関する“ユーザ独自の情報”を格納するための領域で，ユーザが自由に利用できます。
exinfに設定された情報は，処理プログラム（タスク，非タスク）からref_tskシステム・コールを発行することにより，ダイナミックに獲得できます。

備考　システム・コールについての詳細は，RX850 Pro　**ユーザーズ・マニュアル　基礎編**を参照してください。

[機　　能]

UBTIR0割り込み信号用の割り込みハンドラから起動されるタスクです。サンプル・プログラムでは，受信FIFOにデータがあればデータを読み出し，USBのデータ送信関数（usbf850_data_send）を呼び出します。データがない場合は，何もせず終了します。

備考　割り込み処理についての詳細は，4. 8. 2（1）**割り込み処理**を参照してください。

[戻 り 値]

なし

uartb0850_data_send　　発行有効範囲：非タスク｜タスク

[概　　要]

UARTのデータ送信関数。

[C言語形式]

```
void uartb0850_data_send (unsigned char* buffer, int size)
```

[パラメータ]

I/O	パラメータ	説　　明
I	unsigned char* buffer	送信データの先頭アドレス
I	int size	データ・サイズ

[機　　能]

UARTに対して，bufferで指定されたアドレスから，sizeで指定されたサイズ分のデータを送信処理します。

[戻 り 値]

なし

uartb0850_buffer_clear　　発行有効範囲：非タスク｜タスク

[概　　要]

UARTの送受信用バッファのクリア関数。

[C言語形式]

```
void uartb0850_buffer_clear (char* buffer, int size)
```

[パラメータ]

I/O	パラメータ	説　　明
I	char* buffer	バッファの先頭アドレス
I	int size	クリアするバッファのサイズ

[機　　能]

bufferで指定されたアドレスから，sizeで指定されたサイズ分，送信／受信用データ・バッファの0クリアを行います。

[戻 り 値]

なし

uartb0850_uartmode_set　　発行有効範囲：非タスク｜タスク

［概　　要］

UARTの動作モード設定関数。

［C言語形式］

```
void uartb0850_cdc_uartmode_set (void)
```

［パラメータ］

I/O	パラメータ	説　　明
–	–	–

［機　　能］

UART_MODE_INFO構造体に格納された現在の設定値で，UARTの動作モードを設定します。設定内容は，転送速度，パリティ・ビット，データ長，ストップ・ビットの4つです。UART_MODE_INFO構造体は，表4－15 UARTの設定値一覧に示した「初期設定値」を初期値として持っています。この値は，SET LINE CODINGリクエストによって，ホストから変更されます。また，このリクエストが来るとこの関数が呼ばれ，UARTの動作モードを変更します。

備考1. UART_MODE_INFO構造体の詳細については，4. 7. 5（2）UARTモード・テーブル構造体を参照してください。

2. サンプル・プログラムでの有効なUARTの設定値については，4. 8. 3　動作モードを参照してください。

［戻 り 値］

なし

uartb0850_int_ubtire0

発行有効範囲：－

[概　　要]

V850E/ME2に内蔵しているUART用割り込みハンドラ（UBTIRE0信号用）。

[C言語形式]

```
ID uartb0850_int_ubtire0 (void)
```

[パラメータ]

I/O	パラメータ	説　　明
－	－	－

[機　　能]

UBTIRE0信号（UARTB0受信エラー割り込み）によって起動される割り込みハンドラです。ハンドラ内でエラー処理を行います。このハンドラは，CF定義ファイルで定義されています。

備考　割り込み処理についての詳細は，4. 8. 2（1）**割り込み処理**を参照してください。

[戻 り 値]

オブジェクトID番号（タスクのID番号）

uartb0850_int_ubtir0

発行有効範囲：－

[概　　要]

V850E/ME2に内蔵しているUART用割り込みハンドラ（UBTIR0信号用）。

[C言語形式]

```
ID uartb0850_int_ubtir0 (void)
```

[パラメータ]

I/O	パラメータ	説　　明
－	－	－

[機　　能]

UBTIR0信号（UARTB0受信完了割り込み）によって起動される割り込みハンドラです。このサンプル・プログラムでは，割り込み処理用のタスク（usbf850_cdc_data_recv）を起動します。このハンドラは，CF定義ファイルで定義されています。

備考　割り込み処理についての詳細は，4. 8. 2（1）**割り込み処理**を参照してください。

[戻 り 値]

オブジェクトID番号（タスクのID番号）

uartb0850_int_ubtito0

発行有効範囲：－

[概　　要]

V850E/ME2に内蔵しているUART用割り込みハンドラ（UBTITO0信号用）。

[C言語形式]

```
ID uartb0850_int_ubtito0 (void)
```

[パラメータ]

I/O	パラメータ	説　　明
－	－	－

[機　　能]

UBTITO0信号（UARTB0受信タイムアウト割り込み）によって起動される割り込みハンドラです。ハンドラ内で受信タイムアウト処理を行います。このハンドラは，CF定義ファイルで定義されています。

備考　割り込み処理についての詳細は，4. 8. 2（1）**割り込み処理**を参照してください。

[戻 り 値]

オブジェクトID番号（タスクのID番号）

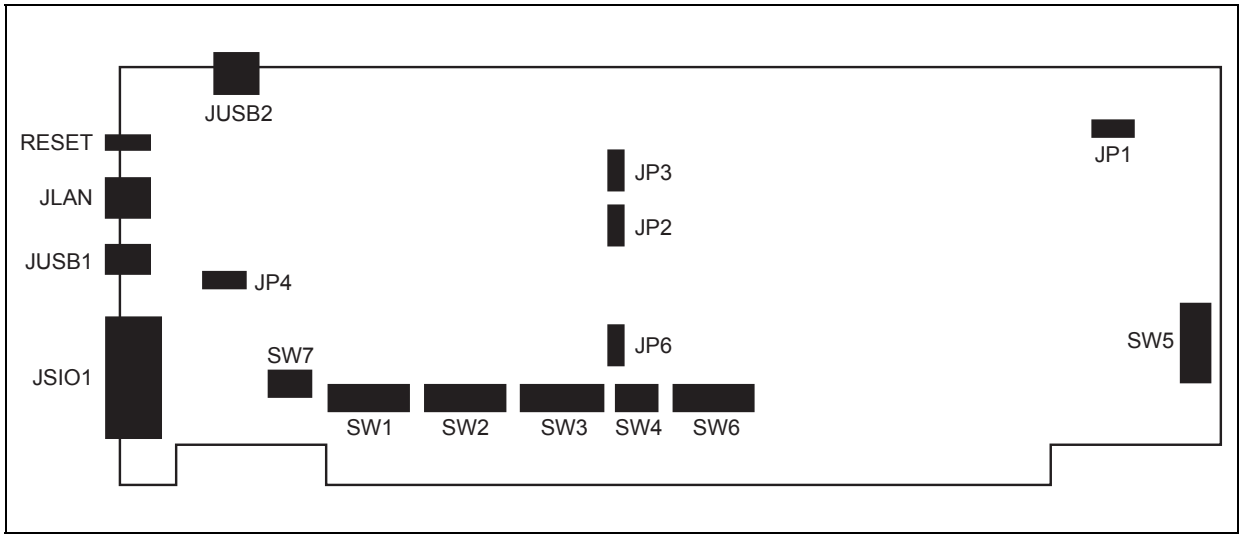
付録A　SG-703111-1ボード

A. 1　概　　要

これらのサンプル・プログラムは，SG-703111-1**ボード上で動作します。**

ここでは，SG-703111-1**ボードのセットアップ方法について示します。**

図A－1　SG-703111-1**の構成**



A. 2　ディップ・スイッチ（SW1-SW7）の設定

SG-703111-1ボードには，7個のディップ・スイッチ（SW1-SW7）があります。

ディップ・スイッチの設定例を次に示します。

表A－1　ディップ・スイッチ（SW1-SW7）の設定（1/2）

スイッチ名	設定	備　考
SW1	1：OFF	モニタROMに関する設定（このアプリケーション・ノートでは，意味を持ちません）。
	2：ON	
	3：ON	
	4：ON	
	5：OFF	
	6：OFF	
	7：OFF	
	8：OFF	
SW2	1：OFF	通常のモニタを起動します（このアプリケーション・ノートでは，意味を持ちません）。
	2：OFF	常時OFFで使用してください。
	3：OFF	NMIを使用します。
	4：ON	$\overline{CS0}$-$\overline{CS2}$をuncacheに設定します。
	5：ON	SDRAMのバス・サイズを32ビットに指定します。
	6：OFF	常時OFFで使用してください。
	7：ON	7：ON, 8：ONで，ユーザ・プログラムを起動します。
	8：ON	
SW3	1：ON	通常は出荷時の状態で使用してください。
	2：ON	
	3：ON	
	4：OFF	
	5：OFF	
	6：OFF	
	7：OFF	
	8：OFF	
SW4	1：ON	通常は出荷時の状態で使用してください。
	2：ON	
	3：OFF	
	4：ON	

備考　ディップ・スイッチ（SW1-SW7）の設定に関する詳細は，SG-703111-1　ユーザーズ・マニュアルを参照してください。

表A－1　ディップ・スイッチ（SW1-SW7）の設定（2/2）

スイッチ名	設定	備　考
SW5	1：OFF	1：OFF, 2：ONで，起動時のバス幅を16ビットに指定します。
	2：ON	
	3：OFF	V850E/ME2のSSEL0端子の入力レベルをハイ・レベルに指定します。
	4：ON	V850E/ME2のSSEL1端子の入力レベルをロウ・レベルに指定します。
	5：OFF	V850E/ME2のJIT0端子の入力レベルをハイ・レベルに指定します。
	6：OFF	V850E/ME2のJIT1端子の入力レベルをハイ・レベルに指定します。
	7：ON	V850E/ME2のPLLSEL端子の入力レベルをロウ・レベルに指定します。
	8：OFF	常時OFFで使用してください。
SW6	1：OFF	通常は出荷時の状態で使用してください。
	2：OFF	
	3：OFF	
	4：OFF	
	5：ON	
	6：ON	
	7：OFF	
	8：OFF	
SW7	1：ON	出荷時の設定から変更しないでください。
	2：ON	
	3：ON	
	4：OFF	

備考　ディップ・スイッチ（SW1-SW7）の設定に関する詳細は，SG-703111-1　ユーザーズ・マニュアルを参照してください。

A. 3　ジャンパ・スイッチ（JP1-JP4, JP6）の設定

SG-703111-1ボードには，5個のジャンパ・スイッチ（JP1-JP4, JP6）があります。
ジャンパ・スイッチの設定例を次に示します。

表A－2　ジャンパ・スイッチ（JP1-JP4, JP6）の設定（1/2）

スイッチ名	設定	備　考
JP1	1－2：Short	V850E/ME2に供給するA/Dコンバータ用電源（AV_{DD}）をボードから供給
JP2	1－2：Short	出荷時の設定
JP3	1－2：Open	出荷時の設定
JP4	1－2：Open	出荷時の設定
JP6	1－2：Short	出荷時の設定

備考　ジャンパ・スイッチ（JP1-JP4, JP6）の設定に関する詳細は，SG-703111-1　ユーザーズ・マニュアルを参照してください。

A. 4　インサーキット・エミュレータ起動時のボード初期化用ファイル

インサーキット・エミュレータを使用して実行ファイルをターゲット・ボード上のメモリに書き込むため，プログラム実行の前にターゲット・ボードを初期化（特にメモリ関連）する必要があります。

インサーキット・エミュレータ起動時に，この初期化を自動的に読み込んで行うための自動実行ファイル（init.mcr）を準備します。 このファイルは，インサーキット・エミュレータのプロジェクト・ファイルと同じディレクトリに置く必要があります。

V850E/ME2用のinit.mcrの例を次に示します。

備考　ファイルの書式についての詳細は，PARTNER　ユーザーズ・マニュアル　V800シリーズ「V800シリーズ共通編」，「NB85E-CB個別編」を参照してください。

```
reset
_PC=0x00100000
POW 0xffff060,0x000f            * CSC0
POW 0xffff062,0x000f            * CSC1
POB 0xffff06E,0x33              * VSWC
POW 0xffff480,0x88b8            * BCT0
POW 0xffff482,0x8888            * BCT1
POW 0xffff484,0x1116            * DWC0
POW 0xffff486,0x1111            * DWC1
POW 0xffff488,0x0002            * BCC
POW 0xffff48A,0x0000            * ASC
POW 0xffff48e,0x6aa9            * LBS
POB 0xffff06E,0x37              * VSWC
POB 0xffff498,0x02              * BMC
POB 0xffff06E,0x33              * VSWC

POB 0xffff6c0,0x00              * OSTS

POW 0xffff4A4,0x20a5            *SCR1
POW 0xffff4A6,0x8203            *RFS1

POB 0xffff1fc,0xff              * PRCMD
POB 0xffff822,0x03              * CKC
POB 0xffff1fc,0xff              * PRCMD
POB 0xffff82c,0x01              * CKS

POW 0xffff056,0x01              * PFCDH *0x00
POW 0xffff040,0x0003            * PMCAL
POB 0xffff80a,0x00              * IRAMM

POB 0xffff04b,0x0f              * PFCCT
```

付録B　関数索引

(1/4)

関数名称	ドライバ名	ページ数
ata_inquiry	USB ストレージ・クラス用ドライバ	195
ata_mode_select	USB ストレージ・クラス用ドライバ	196
ata_mode_select10	USB ストレージ・クラス用ドライバ	197
ata_mode_sense	USB ストレージ・クラス用ドライバ	198
ata_mode_sense10	USB ストレージ・クラス用ドライバ	199
ata_read_capacity	USB ストレージ・クラス用ドライバ	201
ata_read_format_capacities	USB ストレージ・クラス用ドライバ	200
ata_read10	USB ストレージ・クラス用ドライバ	203
ata_read6	USB ストレージ・クラス用ドライバ	202
ata_request_sense	USB ストレージ・クラス用ドライバ	194
ata_seek	USB ストレージ・クラス用ドライバ	191
ata_start_stop_unit	USB ストレージ・クラス用ドライバ	192
ata_synchronize_cache	USB ストレージ・クラス用ドライバ	193
ata_test_unit_ready	USB ストレージ・クラス用ドライバ	190
ata_verify	USB ストレージ・クラス用ドライバ	206
ata_write_buff	USB ストレージ・クラス用ドライバ	208
ata_write_verify	USB ストレージ・クラス用ドライバ	207
ata_write10	USB ストレージ・クラス用ドライバ	205
ata_write6	USB ストレージ・クラス用ドライバ	204
scsi_command_to_ata	USB ストレージ・クラス用ドライバ	189
scsi_to_usb	USB ストレージ・クラス用ドライバ	209
storageDev_Init	USB ストレージ・クラス用ドライバ	188
task_usb0b	USB バス・ドライバ	70
	USB ストレージ・クラス用ドライバ	160
	USB コミュニケーション・クラス用ドライバ	273
task_usb1b	USB バス・ドライバ	71
	USB ストレージ・クラス用ドライバ	161
	USB コミュニケーション・クラス用ドライバ	274
task_usb2b	USB バス・ドライバ	72
	USB ストレージ・クラス用ドライバ	162
	USB コミュニケーション・クラス用ドライバ	275
uartb0850_buffer_clear	USB コミュニケーション・クラス用ドライバ	313
uartb0850_cdc_uartmode_set	USB コミュニケーション・クラス用ドライバ	314
uartb0850_data_recv	USB コミュニケーション・クラス用ドライバ	311
uartb0850_data_send	USB コミュニケーション・クラス用ドライバ	312

(2/4)

関数名称	ドライバ名	ページ数
uartb0850_enable	USB コミュニケーション・クラス用ドライバ	309
uartb0850_int_ubtir0	USB コミュニケーション・クラス用ドライバ	316
uartb0850_int_ubtire0	USB コミュニケーション・クラス用ドライバ	315
uartb0850_int_ubtito0	USB コミュニケーション・クラス用ドライバ	317
uartb0850_uart_init	USB コミュニケーション・クラス用ドライバ	310
usbf850_blkonly_mass_storage_reset	USB ストレージ・クラス用ドライバ	176
usbf850_bulkin1_stall	USB バス・ドライバ	77
	USB ストレージ・クラス用ドライバ	167
	USB コミュニケーション・クラス用ドライバ	280
usbf850_bulkout1_stall	USB バス・ドライバ	78
	USB ストレージ・クラス用ドライバ	168
	USB コミュニケーション・クラス用ドライバ	281
usbf850_cbw_error	USB ストレージ・クラス用ドライバ	181
usbf850_csw_ret	USB ストレージ・クラス用ドライバ	185
usbf850_data_in	USB ストレージ・クラス用ドライバ	183
usbf850_data_out	USB ストレージ・クラス用ドライバ	184
usbf850_data_receive	USB バス・ドライバ	74
	USB ストレージ・クラス用ドライバ	164
	USB コミュニケーション・クラス用ドライバ	277
usbf850_data_send	USB バス・ドライバ	73
	USB ストレージ・クラス用ドライバ	163
	USB コミュニケーション・クラス用ドライバ	276
usbf850_dma_init	USB ストレージ・クラス用ドライバ	186
usbf850_dma_start	USB ストレージ・クラス用ドライバ	187
usbf850_get_encapsulated_response	USB コミュニケーション・クラス用ドライバ	290
usbf850_get_line_coding	USB コミュニケーション・クラス用ドライバ	292
usbf850_getdesc	USB バス・ドライバ	84
	USB ストレージ・クラス用ドライバ	174
	USB コミュニケーション・クラス用ドライバ	287
usbf850_init	USB バス・ドライバ	66
	USB ストレージ・クラス用ドライバ	156
	USB コミュニケーション・クラス用ドライバ	269
usbf850_inthdr	USB バス・ドライバ	67
	USB ストレージ・クラス用ドライバ	157
	USB コミュニケーション・クラス用ドライバ	270
usbf850_inthdr1	USB バス・ドライバ	68
	USB ストレージ・クラス用ドライバ	158
	USB コミュニケーション・クラス用ドライバ	271

(3/4)

(4/4)

関数名称	ドライバ名	ページ数
USBF850REG_READ	USB ストレージ・クラス用ドライバ	211
	USB コミュニケーション・クラス用ドライバ	298
USBF850REG_READ_W	USB ストレージ・クラス用ドライバ	213
	USB コミュニケーション・クラス用ドライバ	300
USBF850REG_SET	USB ストレージ・クラス用ドライバ	210
	USB コミュニケーション・クラス用ドライバ	297
USBF850REG_SET_W	USB ストレージ・クラス用ドライバ	212
	USB コミュニケーション・クラス用ドライバ	299

【発　行】
NECエレクトロニクス株式会社
〒211-8668　神奈川県川崎市中原区下沼部1753
電話（代表）：044(435)5111

お問い合わせ先

【ホームページ】

NECエレクトロニクスの情報がインターネットでご覧になれます。
URL（アドレス）　http://www.necel.co.jp/

【営業関係，技術関係お問い合わせ先】

半導体ホットライン
（電話：午前 9:00～12:00，午後 1:00～5:00）
電　話　：044−435−9494
E-mail　：info@necel.com

【資料請求先】

NECエレクトロニクスのホームページよりダウンロードいただくか，NECエレクトロニクスの販売特約店へお申し付けください。