V850ES マイクロコントローラ USB ファンクション内蔵品

USB ファンクション

ファームウエア・アップデート

R20AN0052JJ0101
Rev.1.01
2010.12.10

要 旨

このアプリケーション・ノートでは，フラッシュ・メモリ・セルフ・プログラミング・ライブラリ（以降，セルフ・ライブラリ）を用いて，内蔵フラッシュ・メモリ上のデータを任意に書き換える処理（セルフ書き換え）の応用，および USB ファンクション・コントローラのコミュニケーション・デバイス・クラス（以降，CDC と記します）を用いた処理の応用を，理解することを目的としています。

処理の詳細は，USB ファンクション・ファームウエア・アップデート・サンプル・プログラムを例に述べていきます。

評価環境は，製品ごとに下記を使用します。なお，本ドキュメントでの処理の詳細の説明は，対象ＣＰＵは，「V850ES/JH3-U（UPD70F3769）」，ターゲットボードは，テセラ・テクノロジー株式会社製「TK-850/JH3U-SP」を例として記載しています。他の対象ＣＰＵ，ターゲットボードを使用する場合の変更点につきましては，第 7 章カスタマイズを参照してください。

動作確認デバイス

表 A-1　マイコンと評価環境ターゲットボード一覧

対象CPU	V850ES/JG3-H V850ES/JH3-H	V850ES/JG3-U V850ES/JH3-U	V850ES/JH3-E V850ES/JJ3-E	V850ES/JG3-L
評価環境 ターゲット ボード	TK-850/JG3H	TK-850/JH3U-SP	TK-850/JH3E+ NET	TK-850/JG3L+ USB

目 次

1. はじめに

1.1 注 意

このアプリケーション・ノートで使用するサンプル・プログラムはあくまで参考用のものであり，当社がこの動作を保証するものではありません。
サンプル・プログラムを使用する場合，ユーザのセット上で十分な評価をしたうえで使用してください。
また本サンプルではフラッシュ・メモリ・セルフ・プログラミング・ライブラリ Type04 Ver1.20 を使用しています。セットに組み込む際には、最新バージョンをご使用／評価の上、使用してください。(※)

(※) 最新版は以下の URL より入手可能です。ダウンロードするには ID/パスワードが必要です。
https://www5.renesas.com/micro/tool_reg/OdsListTop.do?lang=ja

1.2 対 象 者

このアプリケーション・ノートは，V850ES マイクロコントローラの機能を理解し，それを用いたアプリケーション・システムを開発しようとするユーザを対象とします。

1.3 目 的

このアプリケーション・ノートは，V850ES マイクロコントローラに内蔵の USB ファンクション・コントローラを使用するためのサンプル・ドライバの仕様をユーザに理解していただくことを目的とします。

1.4 構 成

このアプリケーション・ノートは，大きく分けて次の内容で構成しています。

- USB ファンクション・ファームウエア・アップデートの概要
- プログラム構成の説明
- アプリケーションの使用方法
- サンプル・プログラムの応用

1.5 読 み 方

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

— ハードウエア機能の詳細（特にレジスタ機能とその設定方法など），および電気的特性を知りたいとき
　→別冊の対象 V850ES マイクロコントローラの　ユーザーズ･マニュアル　ハードウエア編を参照してください。

— 命令機能の詳細を理解しようとするとき
　→別冊の V850ES　ユーザーズ・マニュアル　アーキテクチャ編を参照してください。

2. 概要

2.1 USBファンクション・ファームウエア・アップデート概要

USB ファンクション・ファームウエア・アップデート・サンプル・プログラムは，ホスト・マシン（以降，PC と記します）上のファイル転送アプリケーションで指定されたファイルを，USB のシリアル通信で評価ボードに転送し，セルフ・ライブラリを使用してユーザ・プログラムのブート領域への書き込みや，メモリ上の任意の場所にデータを書き込めます。
USB ファンクション・ファームウエア・アップデート・サンプル・プログラムは，次のプログラムで構成されます。

- ファームウエア・アップデート・プログラム
  評価ボードに実装します。USB でのシリアル通信，およびセルフ書き換えを行います。
- ファイル転送アプリケーション
  ホスト・マシン（PC）で動作し，指定ファイルをシリアル通信で評価ボードへ送ります。
- サンプル・ユーザ・プログラム
  動作確認のための HEX ファイルです。

  — TK-850/JH3U-SP 用
  　タッチ・パネル・プログラム　　：LCD 画面に触れると描画します。
  　フォト・フレーム・プログラム　　：2 枚の画像を一定間隔で繰り返し描画します。
  — TK-850/JG3H 用・TK-850/JH3E+NET・TK-850/JG3L+USB 用
  　キー入力エコーバック・プログラム　　：ターミナルからのキー入力をエコーバックします。
  　ストレージ・デバイス・プログラム　　：Windows にストレージ・デバイスとして認識されます。

次に，プログラムのデータの流れを表します。

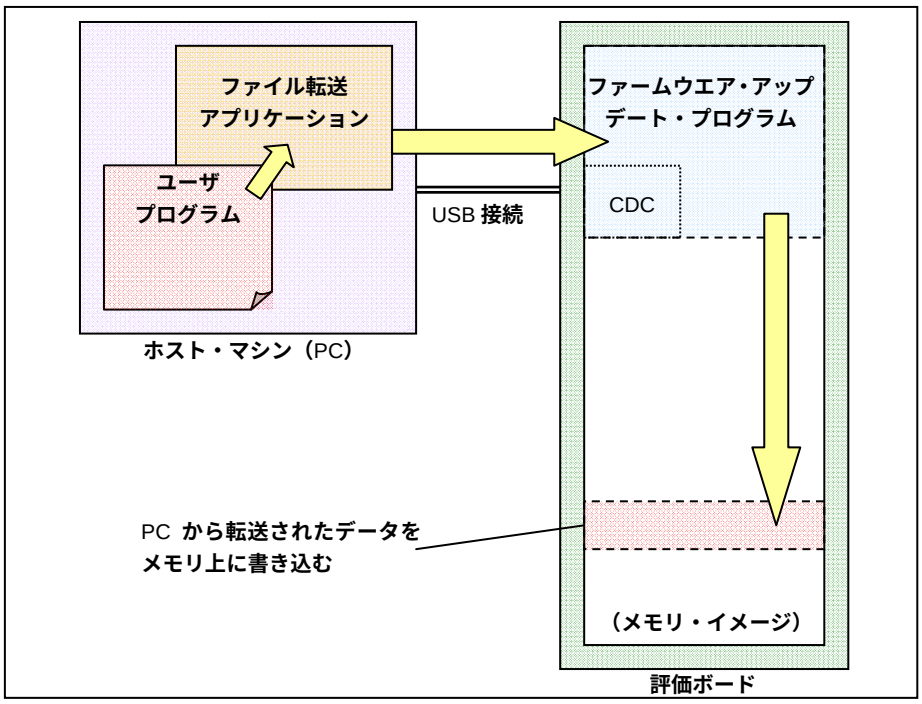


図 2-1　USB ファンクション・ファームウエア・アップデートのデータの流れ

評価ボードは，通常，起動すると書き込まれたユーザ・プログラムが動作しますが，特定条件下で起動（またはリセット）することで，ファームウエア・アップデート・プログラムが動作します。

2.2 特長

USB ファンクション・ファームウエア・アップデート・サンプル・プログラムには，次の特徴があります。

- ファームウエア・アップデート・プログラムは，内蔵フラッシュ・メモリを 4 ブロック（16 K バイト）使用します。
- ユーザ・プログラムの書き換え可能なフォーマット（HEX ファイル）は，モトローラ S フォーマットとインテル拡張フォーマットです。
- メモリ上のアドレスを指定して，任意の領域にデータを書き込めます。
- ユーザ・プログラムは，すべての割り込みを使用できます。
- 使用する内部資源を表 2−1 に示します。

表 2-1　マイコンと評価環境ターゲットボード一覧

資源名	セクション名	サイズ（バイト）							
		TK-850/ JH3U-SP		TK-850/ JG3-H		TK-850/ JH3E+NET		TK-850/ JG3L+USB	
ROM （CONST）	.const	24		24		24		24	
ROM（TEXT）	SelfLib_Rom.text	134	5,580	134	5,580	134	5,556	134	5,608
	.text	5,444		5,444		5,420		5472	
ROM	apstart	52		52		52		52	
RAM （ FLASHTEXT ）	SelfLib_ToRamUsrInt.text	8	1,938	8	1,938	8	1,942	8	1,898
	SelfLib_ToRamUsr.text	8		8		8		8	
	SelfLib_RomOrRam.text	974		974		974		974	
	SelfLib_ToRam.text	480		480		480		480	
	flash.text	466		466		470		426	
RAM（DATA）	.data	12	7,572	12	7,572	12	7,572	12	7,580
	.sdata	200		200		200		208	
	.sbss	5,280		5,280		5,280		5,280	
	.bss	2,048		2,048		2,048		2,048	
	SelfLib_RAM.bss	32		32		32		32	

2.3 フォルダ構成

USB ファンクション ファームウエア・アップデート・サンプル・プログラムのフォルダ構成を示します。

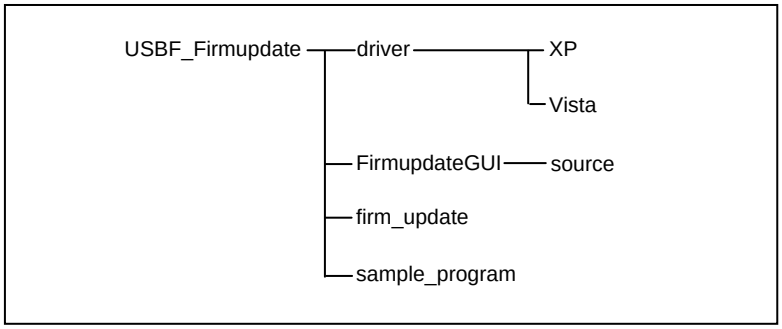


図 2-2　フォルダ構成

次に，各フォルダの説明を示します。

（1）driver¥XP
Windows XP®用の CDC ドライバが格納されているフォルダです。
xxxx_CDC_XP.inf：Windows XP 用の CDC ドライバ(xxxx には型番が入ります)

（2）driver¥VISTA
Windows Vista®用の CDC ドライバが格納されているフォルダです。
xxxx_CDC_VISTA.inf：Windows Vista 用の CDC ドライバ(xxxx には型番が入ります)

（3）FimupdateGUI
ファイル転送アプリケーションが格納されているフォルダです。
UsbfUpdate.exe　：ファイル転送アプリケーションの実行ファイル
UsbfUpdate.ini　：ファイル転送アプリケーションの設定ファイル

（4）FirmupdateGUI¥source
ファイル転送アプリケーションのソース・プログラムが格納されているフォルダです。第 5 章　ファイル転送アプリケーションの解説を参照してください。

（5）firm_update
ファームウエア・アップデート・プログラムが格納されているフォルダです。第 4 章　ファームウエア・アップデート・プログラムの解説を参照してください。

（6）sample_program
サンプル・ユーザ・プログラムが格納されているフォルダです。

- TK-850/JH3U-SP 用
  photo_sample.hex　：フォト・フレーム・プログラム
  touch_sample.hex　：タッチ・パネル・プログラム

- TK-850/JG3H・TK-850/JH3E+NET・TK-850/JG3L+USB 用（xxxx には型番が入ります）
  romp_cdc_xxxx.hex　：キー入力エコーバック・プログラム
  romp_msc_xxxx.hex　：ストレージ・デバイス・プログラム

3. USBファンクション・ファームウエア・アップデートの実行

USB ファンクション・ファームウエア・アップデート・サンプル・プログラムの実行方法について説明します。
ここでは TK-850/JH3U-SP ボードを用い，ユーザ・プログラムが，タッチ・パネル・プログラム，次にフォト・フレーム・プログラムの順番で書き換わることを確認します。

3.1 動作環境

ハードウエア環境を次に示します。

— 評価ボード	TK-850/JH3U-SP（テセラ・テクノロジー株式会社製）
— 評価ボード搭載 CPU	uPD70F3769（V850ES/JH3-U）
— インサーキット・エミュレータ	QB-V850MINI（MINICUBE®）
— USB ケーブル	評価ボードと PC 間でシリアル通信を行う
— PC	Windows XP 搭載

ソフトウエア環境を次に示します。

— 統合開発環境	PM+ V6.32
— コンパイラ	CA850 W3.40
— デバッガ	ID850QB V3.60

— USB ファンクション・ファームウエア・アップデート・サンプル・プログラム一式
- ファームウエア・アップデート・プログラム
- ファイル転送アプリケーション
- サンプル・ユーザ・プログラム：タッチ・パネル・プログラム
  フォト・フレーム・プログラム

3.2　サンプル・プログラムの実行

USB ファンクション・ファームウエア・アップデート・サンプル・プログラムの動作環境を実行するときの手順を示します。

3.2.1　ファームウエア・アップデート・プログラムの実装

(1) ファームウエア・アップデート・プログラムを実装するために，MINICUBE と評価ボードを接続します。次に TK-850/JH3U-SP の接続図を示します。

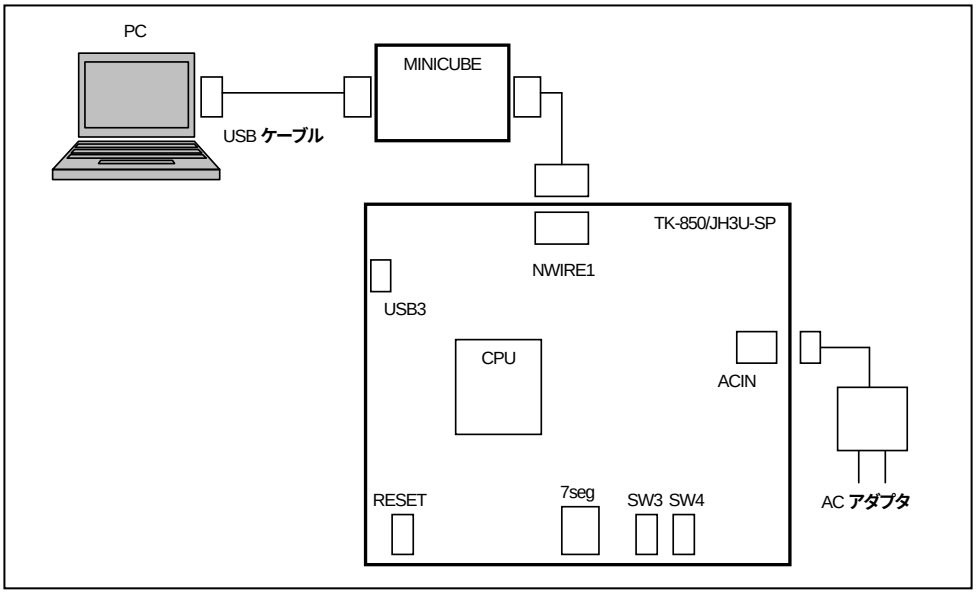


図 3-1　MINICUBE と評価ボードの接続図

(2) PM+を起動し，メニューの[ファイル]-[ワークスペースを開く]を選択してファームウエア・アップデート・プログラムのワークスペース・ファイル“firm_update.prw”を開きます。

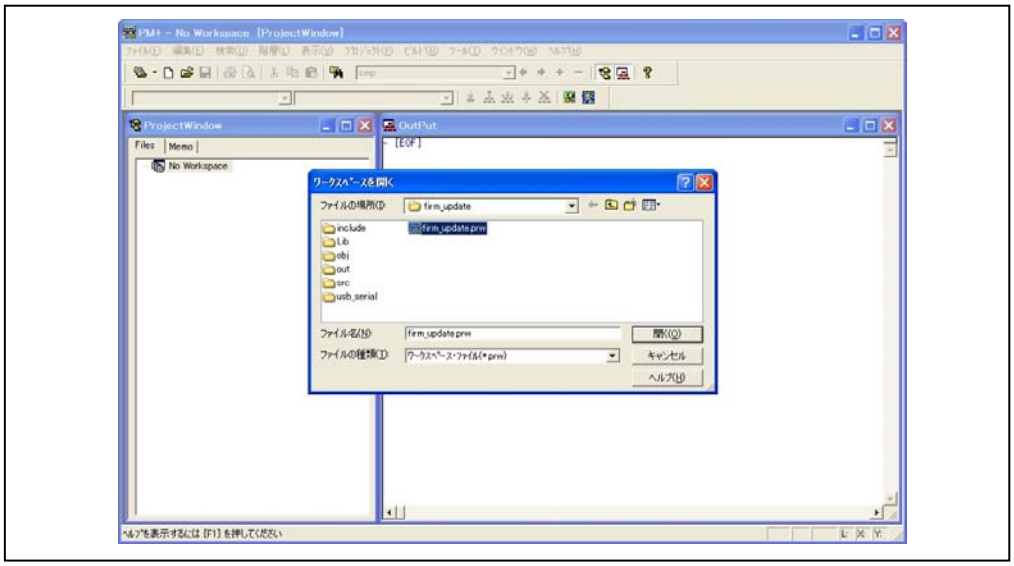


図 3-2　ワークスペース・ファイルの指定

(3) メニューの[ビルド]-[デバッグ]を選択すると，ファームウエア・アップデート・プログラムが評価ボードに書き込まれます。

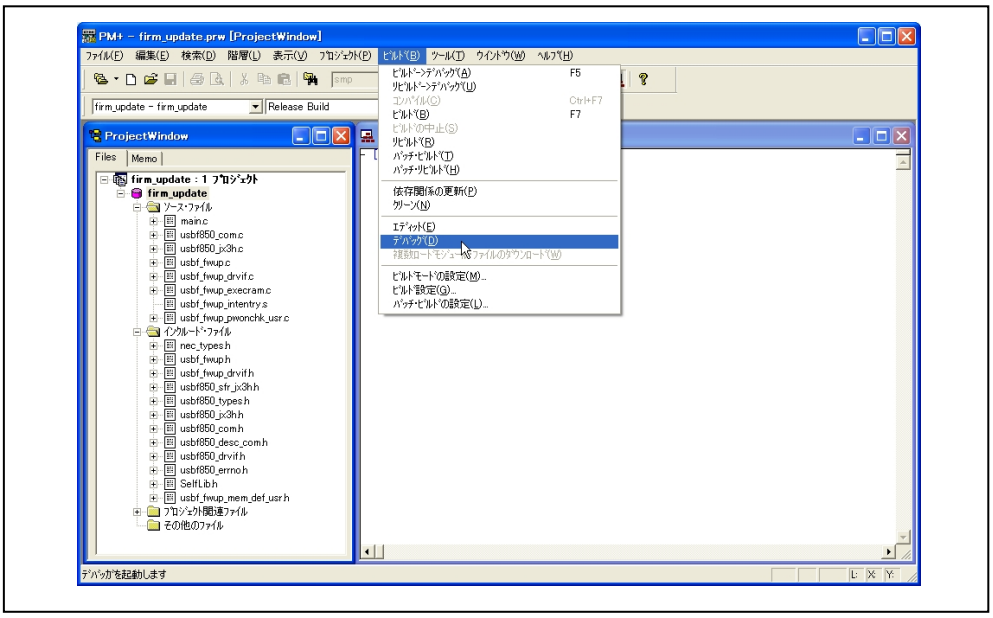


図 3-3　デバッグの選択

3.2.2　書き換え処理の実行手順

(1) 書き換え処理を実行するために，MINICUBE を外し，PC と評価ボードを USB ケーブルで接続します。
次に TK-850/JH3U-SP の接続図を示します。

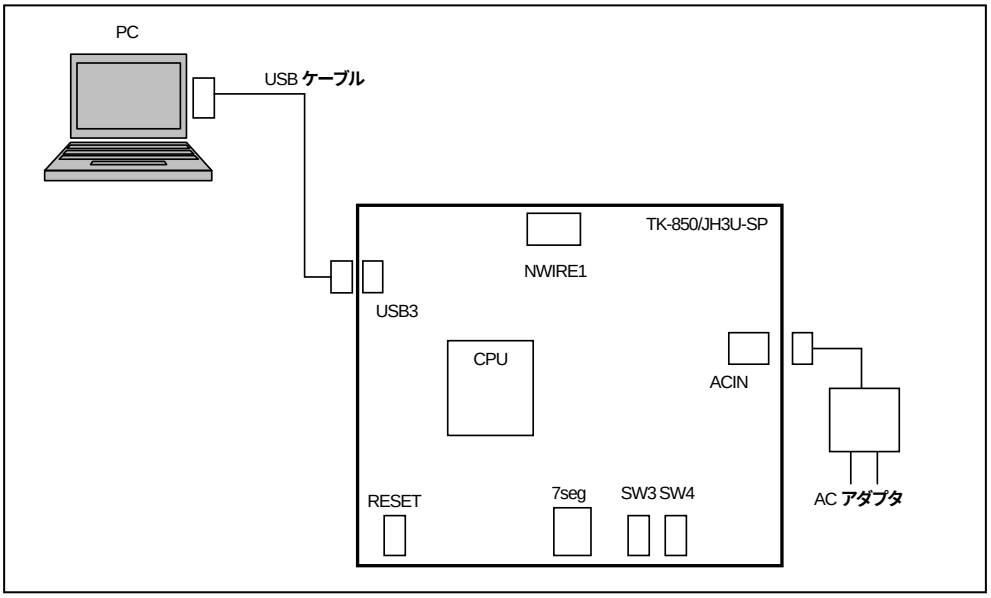


図 3-4　PC と評価ボードの接続図

(2) 評価ボード上の SW3, SW4 を押しながら，リセット・ボタンを押してください。書き換えモードとなり，PC からの転送データを待つ状態になります。

【注意】　初めて TK-850/JH3U-SP と PC を接続し書き換えモードにした場合は，CDC ドライバのインストールが必要になります。詳細については，2. 2. 3　CDC ドライバのインストールを参照してください。
他のターゲットボードをご使用の場合も，初めて接続する場合には CDC ドライバは同様にインストールが必要です。
書き換えモードの起動ボタンは，評価環境 TK-850/JG3H，　TK-850/JH3U-SP，　TK-850/JH3E+NET，TK-850/JG3L+USB 共に SW3, SW4 です。

(3) 評価ボードへ転送するサンプル・ユーザ・プログラムの HEX ファイル（ここではタッチ・パネル・プログラムの“touch_sample.hex”ファイルを指します）を PC 側に用意します。PC 上で，ファイル転送アプリケーションを起動します（図 3-5 参照）。

“Load File” ボタンをクリックして，対象とする HEX ファイルを選択します（ファイルのパスは，《Path》のテキスト・ボックスへの直接入力，またはファイルのウインドウ上へのドラッグ＆ドロップでも，《Path》のテキスト・ボックスに表示できます）。

《Mode》は《Chip》を選択します。《COM》は，デバイス・マネージャを確認し，接続した USB ポートを選択してください。

【注意】　COM の番号は環境によって変わります。

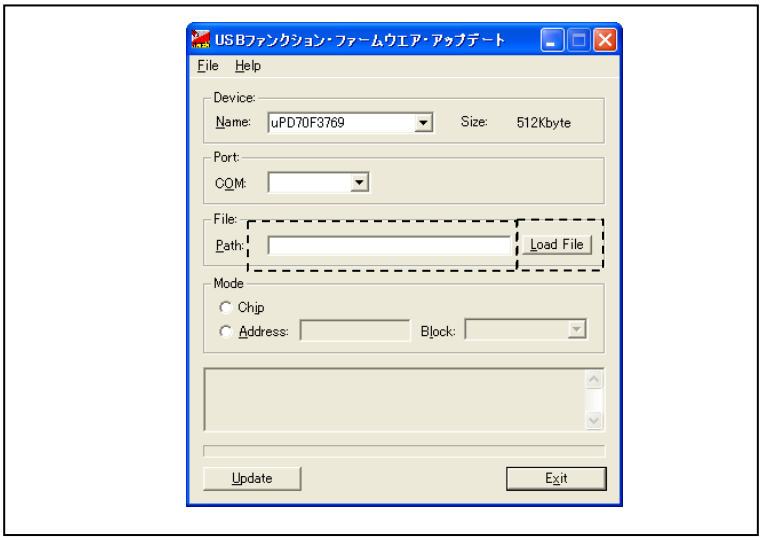


図 3-5　ファイル転送アプリケーションでファイルを選択

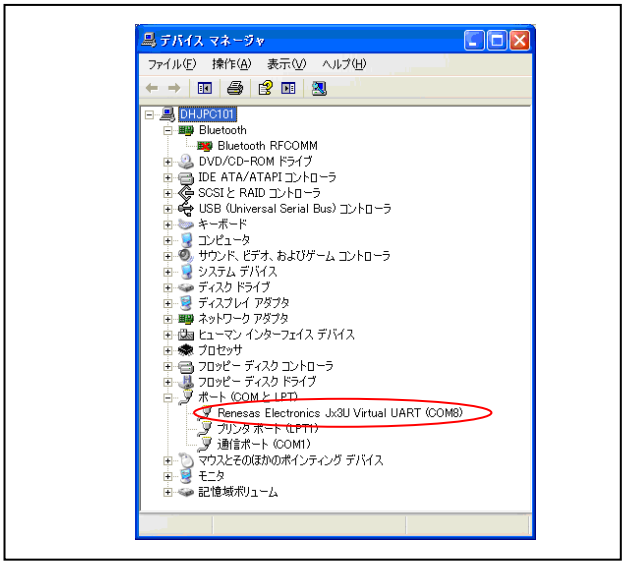


図 3-6　デバイス・マネージャ

(4) ファイル転送アプリケーションの “Update” ボタンをクリックします。開始のメッセージが表示され，ファイルの転送処理，および書き換え処理が開始されます。

(5) 転送処理，および書き換え処理が終了すると，ファイル転送アプリケーションにより，終了メッセージが表示されます。これで一連の書き換え処理は終了です。

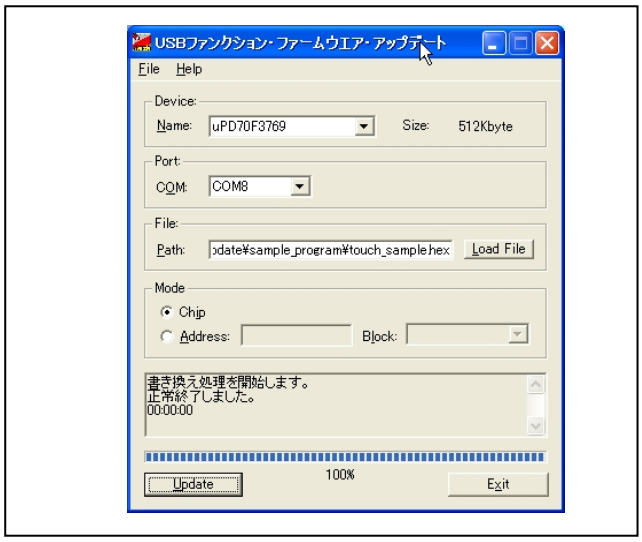


図 3-7　書き換え処理終了 1

(6) 評価ボードをリセットします。書き込んだユーザ・プログラムが起動します。
LCD の画面に触れることで，描画していきます。

(7) ユーザ・プログラムを書き換えます。フォト・フレーム・プログラム“photo_sample.hex”を用意し，(4) から同様の手順で書き換えを行います。

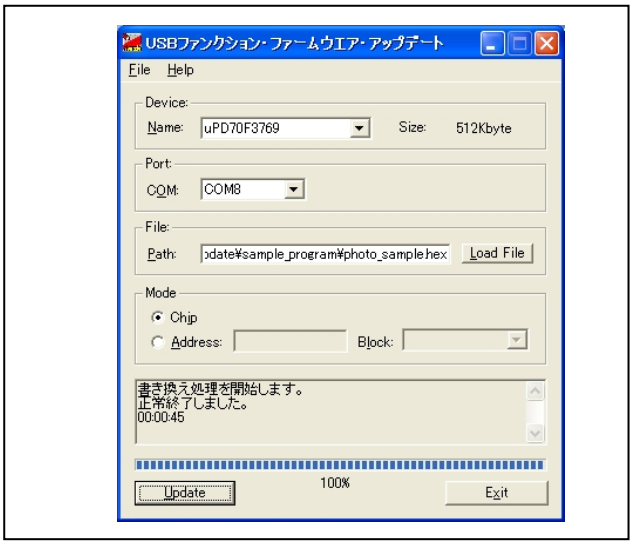


図 3-8　書き換え処理終了 2

(8) 評価ボードをリセットします。書き込んだユーザ・プログラムが起動します。
一定間隔で，LCD に画像を切り替え表示します。

3.2.3　CDCドライバのインストール

TK-850/JH3U-SP, TK-850/JG3H, TK-850/JH3E+NET, TK-850/JG3L+USB を初めて書き換えモードにした場合は，PC に CDC ドライバをインストールする必要があります。次に Windows XP に TK-850/JH3U-SP を接続した場合を例とした CDC ドライバのインストール手順を示します。

(1) 新しいハードウエアが検出され，＜新しいハードウエアの検出ウィザード＞ウインドウが表示されます。《一覧または特定の場所からインストールする（詳細）（S）》を選択し，“次へ（N）”をクリックしてください。

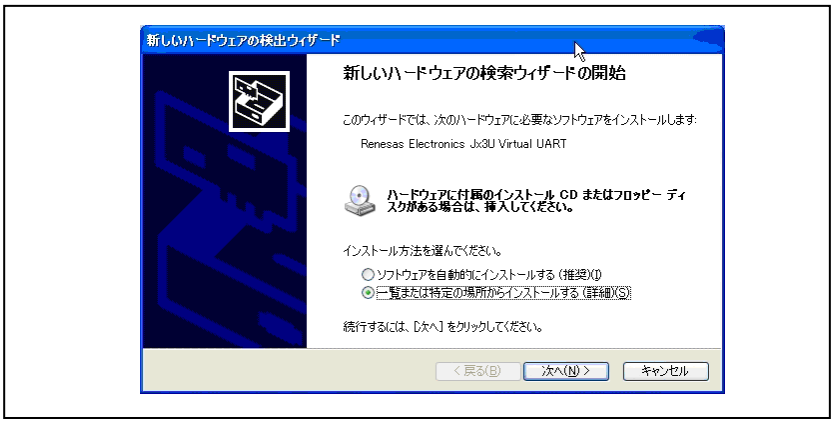


図 3-9　新しいハードウエアの検索ウィザード

(2) 《次の場所で最適のドライバを検索する（S）》，《次の場所を含める（O）》を選択します。“参照（R）” をクリックして“JG3H_CDC_XP.inf”の存在するフォルダを指定し，“次へ（N）” をクリックしてください。

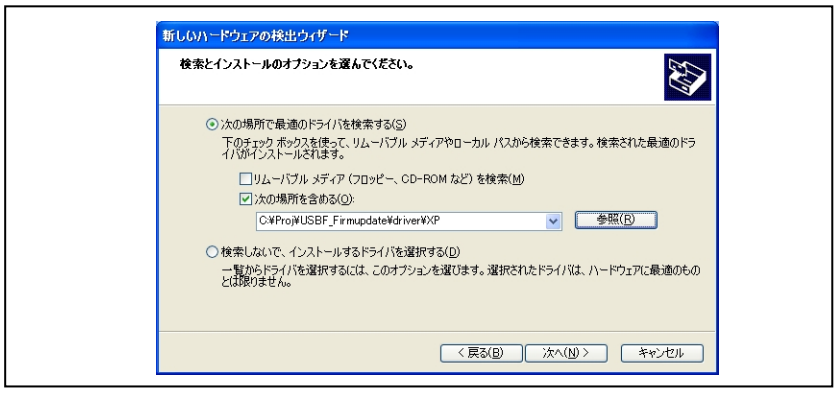


図 3-10　ドライバの場所の選択

(3) 次のインストール確認画面が表示される場合は，“続行（C）”をクリックしてください。

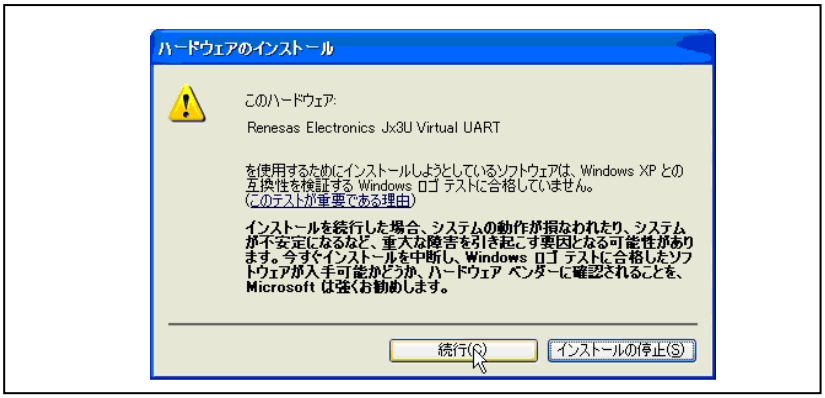


図 3-11　インストール確認

(4) 次のウインドウが表示されたら，CDC ドライバのインストールは完了です。”完了“ をクリックしてください。

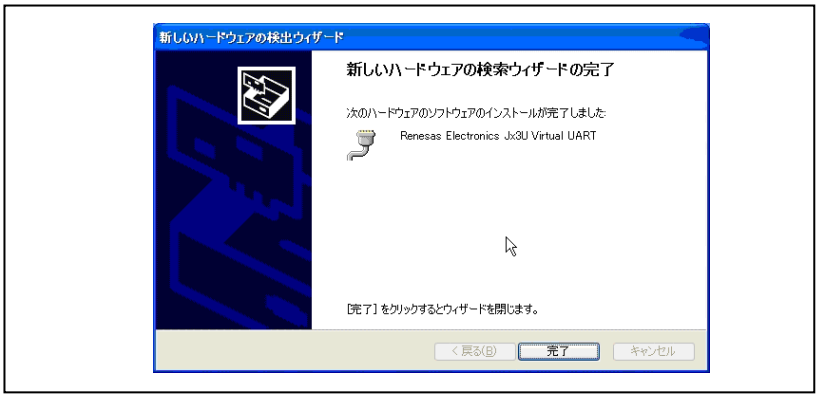


図 3-12　インストール完了

4. ファームウエア・アップデート・プログラムの解説

この章では，ファームウエア・アップデート・プログラムで使用している各ファイルについて説明します。

4.1 ファイル・フォルダ構成

ファームウエア・アップデート・プログラムのソース・ファイルとフォルダ構成を示します。

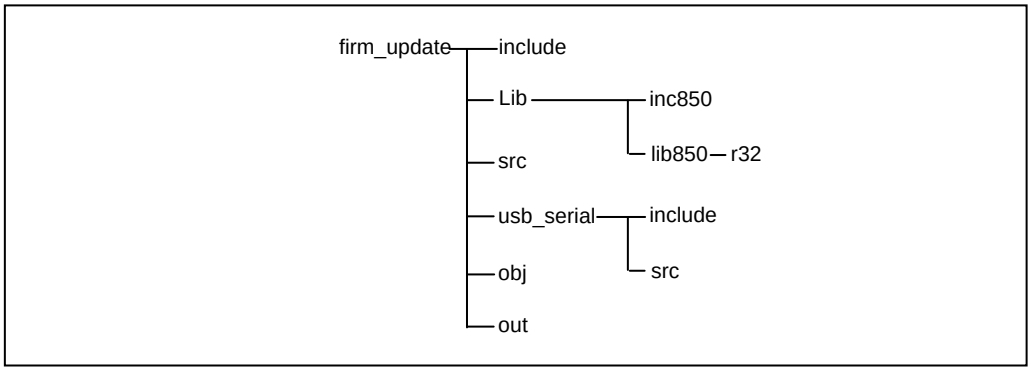


図 4-1　ファームウエア・アップデート・プログラムのフォルダ構成

4.1.1 firm_updateフォルダ

ファームウエア・アップデート・プログラムのフォルダ直下にはプロジェクト関連ファイルが格納されています。次に主なファイルを示します。

表 4-1　ファームウエア・アップデート・プログラムのプロジェクト関連ファイル

ファイル名	説　明
firm_update.prw	PM+ワークスペース・ファイル
firm_update.prj	PM+プロジェクト・ファイル
firm_update.pri	PM+プロジェクトのPRIファイル
firm_update.cld	PM+プロジェクトのCLDファイル
firm_update.mak	メイク・ファイル
firm_update.dir	リンク・ディレクティブ・ファイル

4.1.2 firm_update¥includeフォルダ

ファームウエア・アップデート・プログラムのヘッダ・ファイルを格納するフォルダです。

表 4-2　ファームウエア・アップデート・プログラムのヘッダ・ファイル

ファイル名	説　明
usbf_fwup.h	セルフ・アップデートを行うためのヘッダ・ファイル
usbf_fwup_drvif.h	USBファンクション・コントロール・ドライバ・インタフェースのヘッダ・ファイル
usbf_fwup_mem_def_usr.h	ファームウエア・アップデート・プログラムのメモリを指定するユーザ・カスタマイズ・ヘッダ・ファイル

4.1.3 firm_update¥libフォルダ

セルフ・ライブラリを格納するフォルダです。

表 4-3　セルフ・ライブラリとヘッダ・ファイル

ファイル名	説　明
inc850¥types.h	型を統一した形式に定義しているヘッダ・ファイル
inc850¥SelfLib.h	セルフ・ライブラリのヘッダ・ファイル
lib850¥r32¥libf.a	セルフ・ライブラリ本体

4.1.4 firm_update¥srcフォルダ

ファームウエア・アップデート・プログラムのソース・ファイルを格納するフォルダです。

表 4-4　ファームウエア・アップデート・プログラムのソース・ファイル

ファイル名	説　明
crtE.s	スタートアップ・ファイル
main.c	メイン・ルーチンのソース・ファイル
usbf_fwup_intentry.s	フラッシュ環境時の割り込みエントリのソース・ファイル （フラッシュ環境については，4. 5　割り込み処理参照）
usbf_fwup.c	セルフ・アップデートを行うソース・ファイル
usbf_fwup_execram.c	フラッシュ・メモリ書き込みを行うソース・ファイル
usbf_fwup_pwonchk_usr.c	ユーザ・カスタマイズ・ソース・ファイル （セルフ・アップデート・プログラム，ユーザ・プログラムのどちらを実行するかの判定処理を記述します）
usbf_fwup_drvif.c	CDCドライバとのインタフェースとなるソース・ファイル

4.1.5 firm_update¥usb_serialフォルダ

CDC のソース・ファイル，およびヘッダ・ファイルを格納します。

表 4-5　CDC プログラムのソース・ファイル

ファイル名	説　明
include¥usbf850_types.h	型を統一した形式に定義しているヘッダ・ファイル
include¥usbf850_error.h	終了コードおよびエラー・コードを定義したヘッダ・ファイル
include¥usbf850_jx3u.h	USBファンクション・レジスタ設定用マクロを定義したヘッダ・ファイル
include¥usbf850_sfr_jx3u.h※	USBファンクション・レジスタ制御用マクロを定義したヘッダ・ファイル
include¥usbf850_desc_com.h	ディスクリプタを定義したヘッダ・ファイル
include¥usbf850_com.h	CDC固有処理のためのヘッダ・ファイル
include¥usbf850_devif.h	CDCドライバとのインタフェースを定義したヘッダ・ファイル
src¥usbf850_jx3u.c	USB関連レジスタ初期化，エンドポイント制御，バルク転送，コントロール転送を行うソース・ファイル
src¥usbf850_com.c	CDC固有処理を行うソース・ファイル

※TK-850/JH3E+NET, TK-850/JG3L+USB 用サンプル・プログラムにはこのファイルはありません。

4.1.6 firm_update¥objフォルダ

ファームウエア・アップデート・プログラムのオブジェクト・ファイルを格納するフォルダです。

4.1.7 firm_update¥outフォルダ

ファームウエア･アップデート･プログラムの実行可能なオブジェクト･ファイルと HEX ファイルを格納するフォルダです。

表 4-6　実行可能オブジェクト・ファイル

ファイル名	説　明
romp.out	実行可能オブジェクト・ファイル
firm_update.hex	ヘキサ・フォーマットの実行可能オブジェクト・ファイル

4.2　メモリ・マップ

メモリ配置とリンク・ディレクティブ・ファイルの説明をします。

4.2.1　メモリ・マップ詳細

下記は，セルフ・アップデート・プログラムのメモリ・マップです。ここでは V850ES/JH3-U(μPD70F3769)を例に説明します。
ブロックとは，セルフ・ライブラリが内蔵フラッシュ・メモリを書き換える単位です。

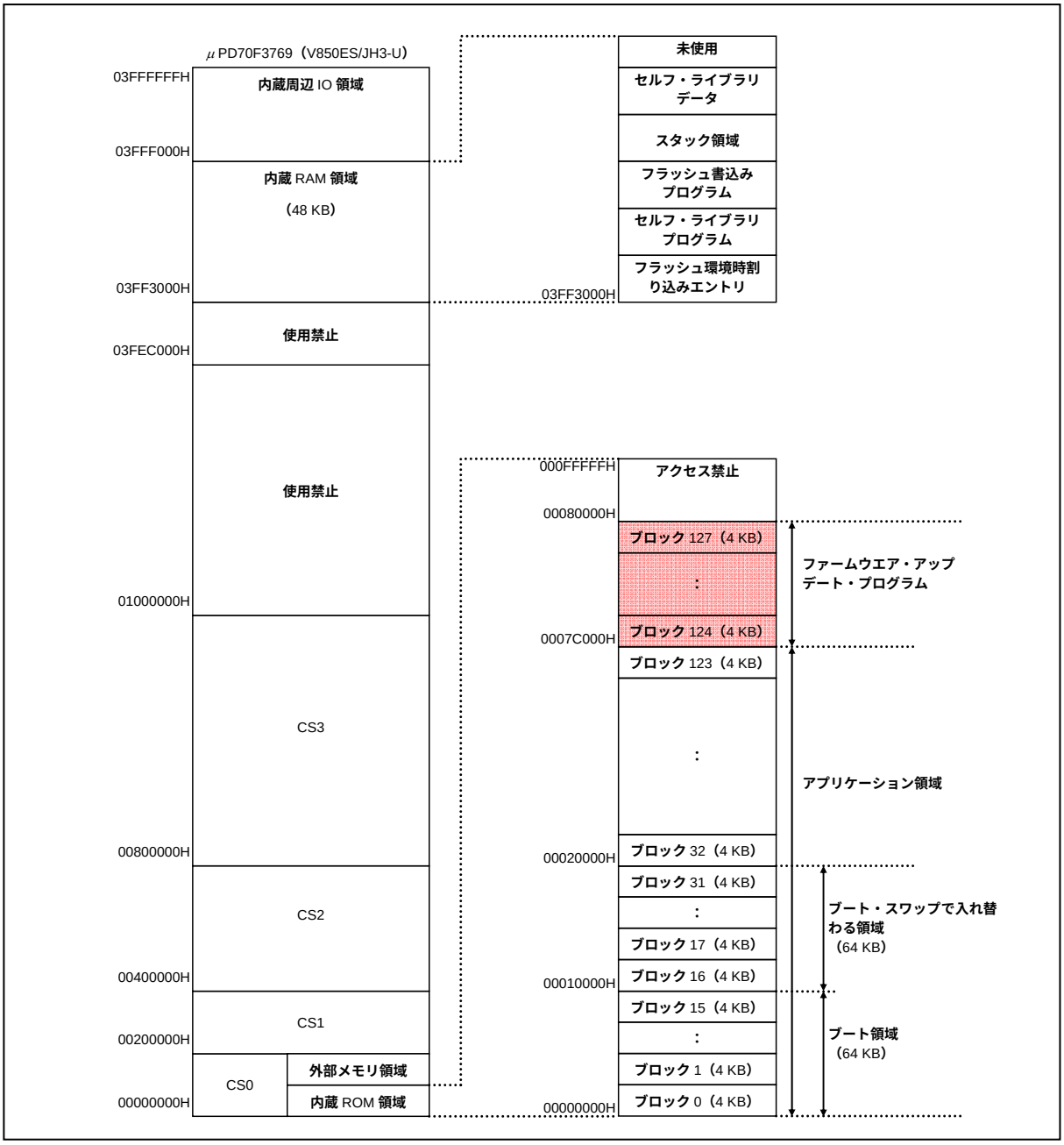


図 4-2　メモリ・マップ詳細図

4.2.2 リンク・ディレクティブ・ファイル（flash_update.dir）

リンク・ディレクティブ・ファイル（flash_update.dir）で領域の割り当てを行います（セグメントを定義し，マッピングを行います）。ここでは V850ES/JH3-U(uPD70F3769)を例に説明します。
実行可能セクション（.text：プログラム・データ），否実行セクション（.const：定数データ），RAM 領域配置情報等を uPD70F3769（V850ES/JH3-U）の内蔵メモリ空間へ配置します。

（1）ROM 領域の割り当て
ファームウエア・アップデート・プログラムが使用する ROM 空間配置情報は，0007C000H-0007FFFFH の 16 K バイトです。したがって，ユーザ・プログラムが使用する ROM 領域は，00000000H-0007BFFFH の 496 K バイト以内に収める必要があります。

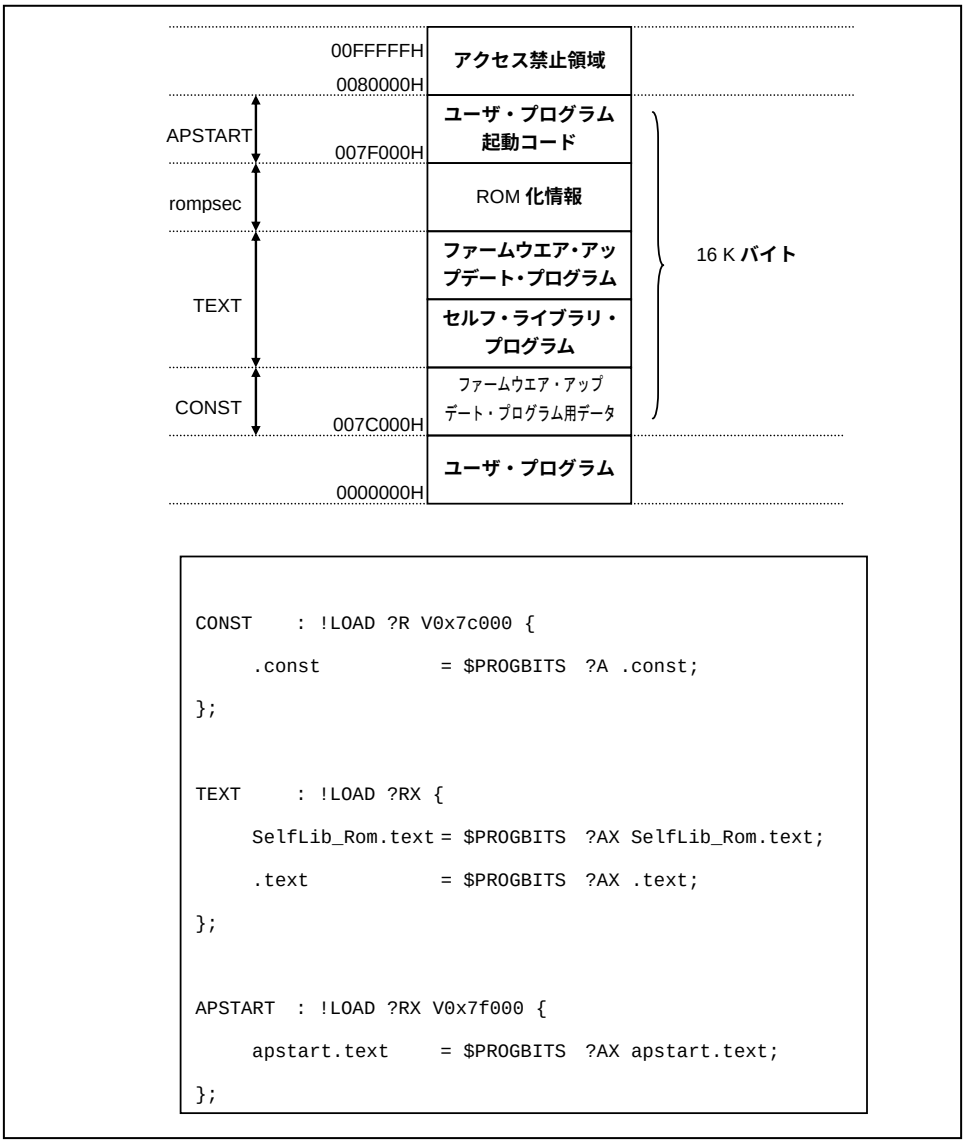


図 4-3　リンク・ディレクティブ（ROM）

追加したセクションの説明を次に示します。

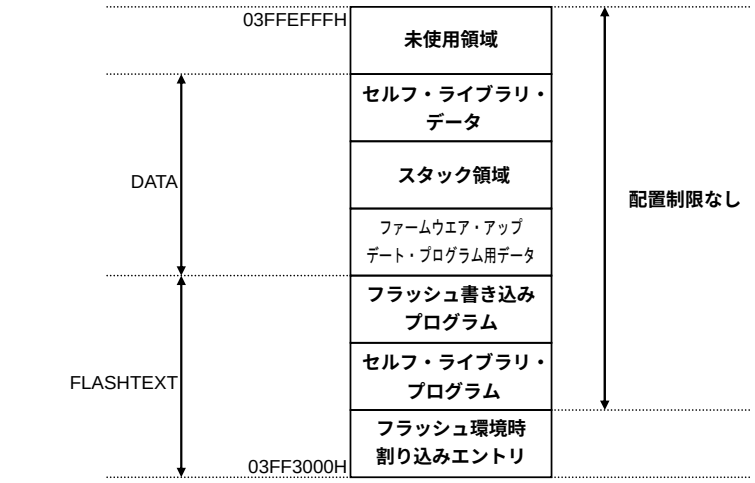
表 4-7　ROM 領域追加セクション

セクション	内　　容
SelfLib_Rom.text	セルフ・ライブラリ初期化処理を行う
.text	ファームウエア・アップデート・プログラム本体が配置される
apstart.text	ファームウエア・アップデート・プログラムが使用する，ユーザ・プログラムへのジャンプ・コードを書き込む領域

（2）RAM 領域の割り当て

3FF3000H-3FFEFFFH が RAM 空間となります。

3FF3000H から 8 バイトが，フラッシュ環境時の割り込みエントリとなりますが，ファームウエア・アップデート・プログラムでは，フラッシュ環境時の割り込みを使用していません。ただし，割り込みエントリの配置は行っています。詳細については，4. 5　割り込み処理を参照してください。




```
FLASHTEXT: !LOAD ?RX V0x3ff3000 {
    SelfLib_ToRamUsrInt.text = $PROGBITS  ?AX SelfLib_ToRamUsrInt.text;
    SelfLib_ToRamUsr.text    = $PROGBITS  ?AX SelfLib_ToRamUsr.text;
    SelfLib_RomOrRam.text    = $PROGBITS  ?AX SelfLib_RomOrRam.text;
    SelfLib_ToRam.text       = $PROGBITS  ?AX SelfLib_ToRam.text;
    flash.text               = $PROGBITS  ?AX flash.text;
};

DATA    : !LOAD ?RW {
    .data           = $PROGBITS    ?AW  .data;
    .sdata          = $PROGBITS    ?AWG .sdata;
    .sbss           = $NOBITS      ?AWG .sbss;
    .bss            = $NOBITS      ?AW  .bss;
    SelfLib_RAM.bss = $NOBITS      ?AW  SelfLib_RAM.bss;
};
```

図 4-4　リンク・ディレクティブ（RAM）

追加したセクションの説明を次に示します。

表 4-8　RAM 領域追加セクション

セクション	内　容
SelfLib_ToRamUsrInt.text	**セルフ・ライブラリの割り込みエントリ処理を行う**
SelfLib_ToRamUsr.text	**ユーザ用のプログラムが配置される**
SelfLib_RomOrRam.text	**セルフ・ライブラリのユーザ・インタフェース処理を行う**
SelfLib_ToRam.text	**セルフ・ライブラリのフラッシュ・マクロ・サービスの呼び出し処理を行う**
flash.text	**ファームウエア・アップデート・プログラムの**RAM**上での作業領域**
SelfLib_RAM.text	**セルフ・ライブラリの作業領域**

リンク・ディレクティブの詳細については，CA850　ユーザーズ・マニュアルを参照してください。
セルフ・ライブラリの詳細については，V850 マイクロコントローラ　ユーザーズ・マニュアル　フラッシュ・メモリ・セルフ・プログラミング・ライブラリ Type04 Ver.1.20 を参照してください。

4.3 ブート処理

ブート処理とは，V850 マイコンをリセット後，メイン関数（C 言語記述：main()）を実行する前に実行するプログラムを指します。
V850 マイコンでは，リセット後の初期化処理として，主に次のことを行います。

- リセットが入ったときの RESET ハンドラの設定
- スタートアップ・ルーチンのレジスタ・モード設定
- スタック領域の確保とスタック・ポインタの設定
- main 関数の引数領域の確保
- tp, gp, ep の設定，マスク・レジスタへマスク値を設定
- main 関数実行前に行う必要のある周辺 I/O レジスタの初期化
- sbss 領域，bss 領域，sebss 領域，tibss.byte 領域，tibss.word 領域，sibss 領域の初期化
- main 関数への分岐

ブート処理は，スタートアップ・ファイル（crtE.s）で定義しています。
ブート処理の詳細については，CA850　ユーザーズ・マニュアルを参照してください。
ファームウエア・アップデート・プログラムでは，上記に加え，ユーザ・プログラムへの分岐，V850 マイコンの初期化処理を行っています。次にブート処理の大まかな流れを示します。

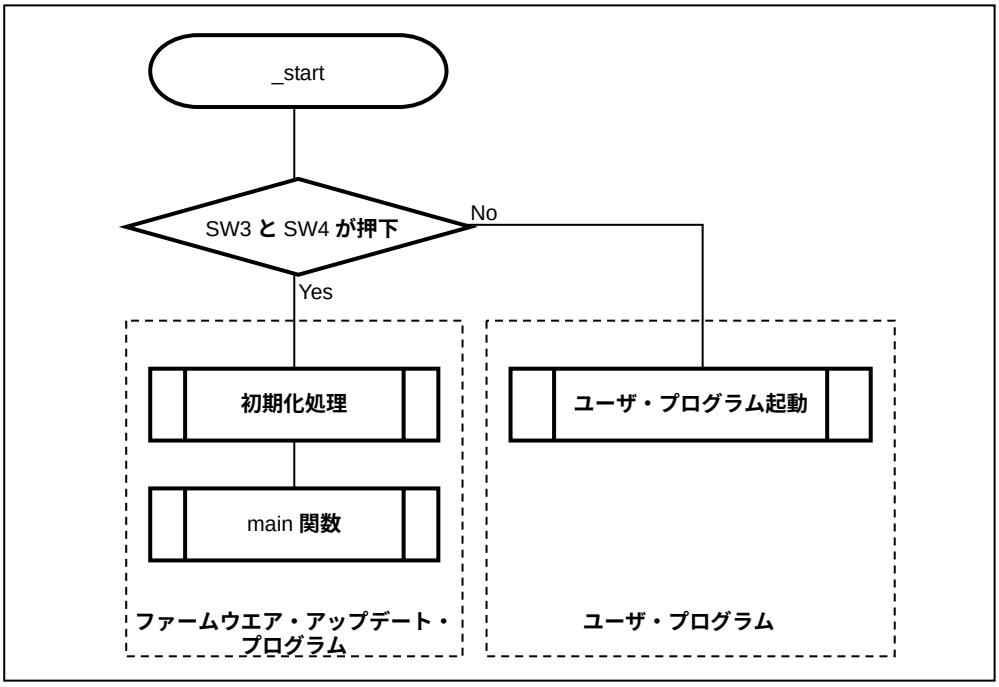


図 4-5　ファームウエア・アップデート・プログラムのブート処理の流れ

4.3.1　スタートアップ・ファイル（crtE.s）

次に，ファームウエア・アップデート・プログラムのスタートアップ・ファイルについて説明します。

```
#-------------------------------------------------------------------------------
#         special symbols
#-------------------------------------------------------------------------------
          .extern   __tp_TEXT, 4
          .extern   __gp_DATA, 4
          .extern   __ep_DATA, 4
          .extern   __ssbss, 4
          .extern   __esbss, 4
          .extern   __sbss, 4
          .extern   __ebss, 4

#-------------------------------------------------------------------------------
#          C program main function
#-------------------------------------------------------------------------------
          .extern   _main
          .extern   _usbf_fwup_pwonchk_usr

#-------------------------------------------------------------------------------
#         for argv
#-------------------------------------------------------------------------------
          .data
          .size     __argc, 4
          .align    4
__argc:
          .word     0
          .size     __argv, 4
__argv:
          .word     #.L16
.L16:
          .byte     0
          .byte     0
          .byte     0
          .byte     0

#-------------------------------------------------------------------------------
#         dummy data declaration for creating sbss section
#-------------------------------------------------------------------------------
          .sbss
          .lcomm    __sbss_dummy, 0, 0

#-------------------------------------------------------------------------------
#         system stack
#-------------------------------------------------------------------------------
          .set      STACKSIZE, 0x800
          .bss
          .lcomm    __stack, STACKSIZE, 4

#-------------------------------------------------------------------------------
#         RESET handler
#-------------------------------------------------------------------------------
          .section  "RESET", text
          jr        __start
```

2048 バイトのスタック領域を確保

リセット・ベクタ（0000H）
リセット後__start へ分岐

図 4-6　スタートアップ（1/4）

```
#-------------------------------------------------------------------------------
#         application start routine
#-------------------------------------------------------------------------------
                    .section "apstart.text", text
                    .align    4
                    .globl    __apstart
__apstart:
                    nop
                    nop
                    nop
                    nop
                    nop
                    nop
                    nop
                    nop
                    nop
                    nop
                    nop
                    nop
                    nop
                    nop
                    nop
                    nop
                    nop
                    nop
                    nop
                    nop
                    nop
                    nop
                    nop
                    halt


#-------------------------------------------------------------------------------
#         start up
#                   pointers: tp - text pointer
#                             gp - global pointer
#                             sp - stack pointer
#                             ep - element pointer
#                   mask reg: r20 - 0xff
#                             r21 - 0xffff
#         exit status is set to r10
#-------------------------------------------------------------------------------
                    .text
                    .align    4
                    .globl    __start
                    .globl    __exit
                    .globl    __startend

__start:
                    mov       #__tp_TEXT, tp              -- set tp register
                    mov       #__gp_DATA, gp              -- set gp register offset
                    add       tp, gp                      -- set gp register
                    mov       #__stack+STACKSIZE, sp      -- set sp register
                    mov       #__ep_DATA, ep              -- set ep register
```

このセクションは，ユーザ・プログラムを動作させるための分岐するコードを書き込む領域です。セルフ・アップデート・プログラムが，ユーザ・プログラムを書き込むときに，この領域を更新します。

tp，gp，ep，sp を設定

図 4-6　スタートアップ（2/4）

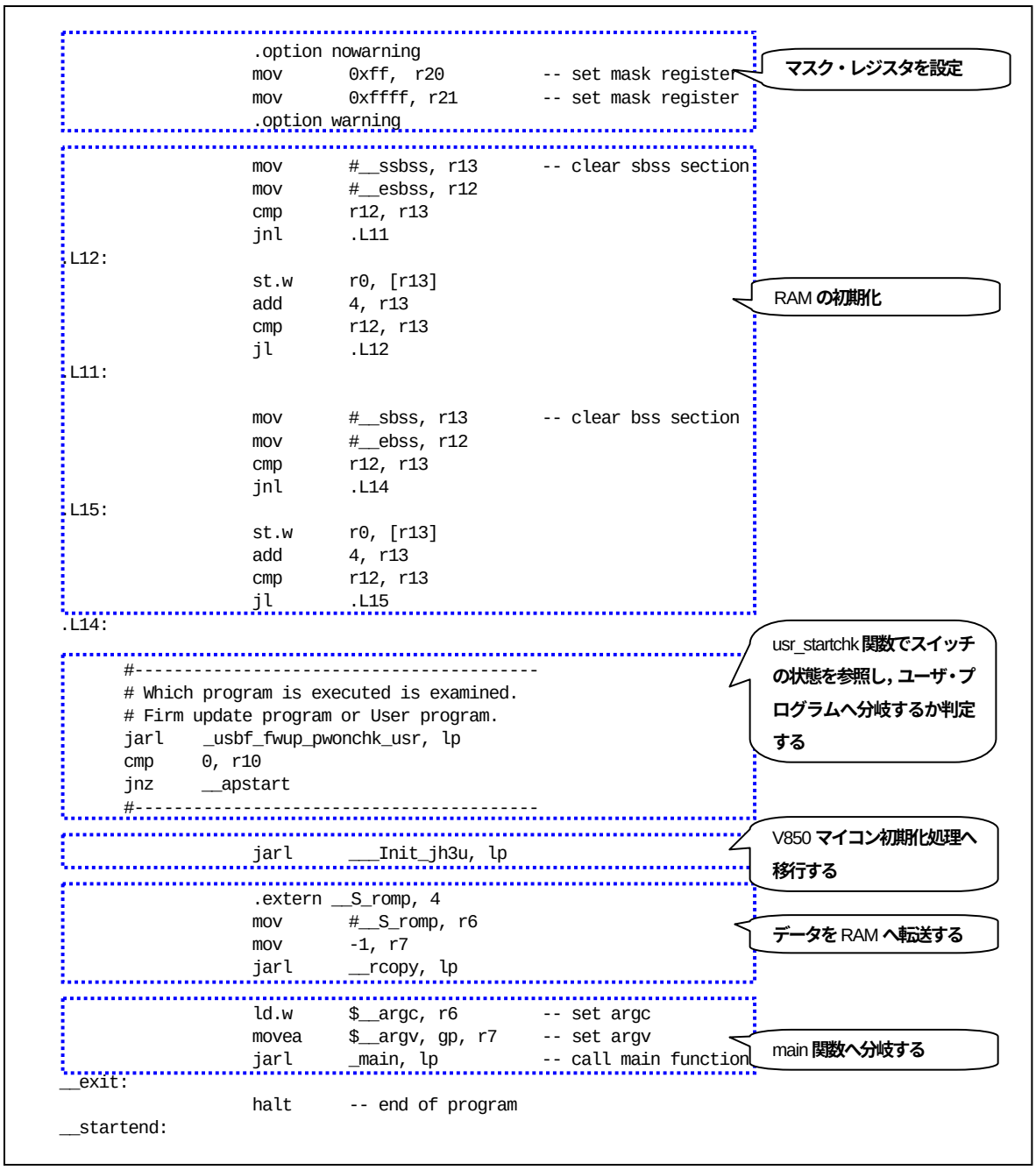


図 4-6　スタートアップ（3/4）

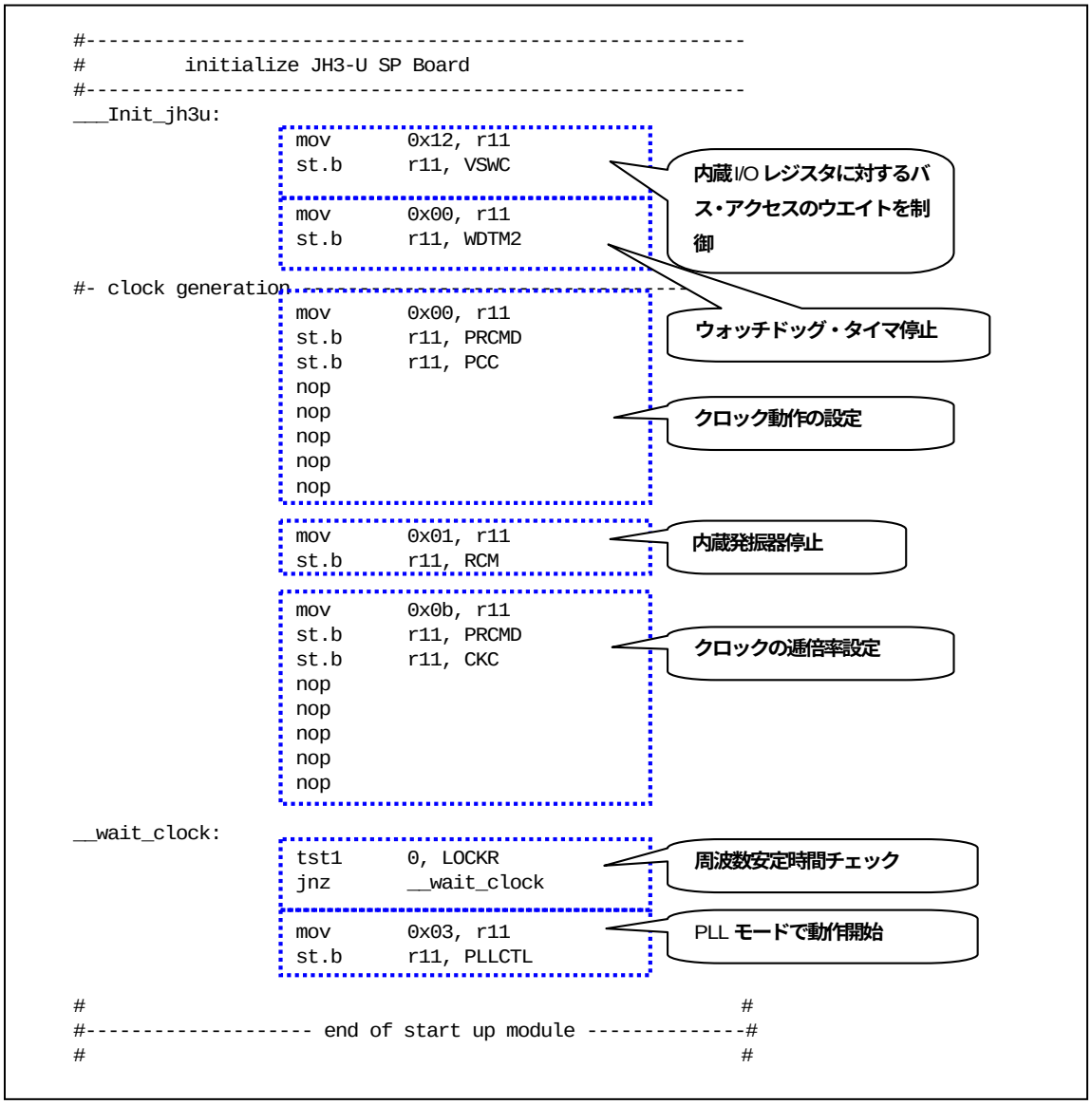


図 4-6　スタートアップ（4/4）

初期化処理では，CPU クロック，使用する周辺機能について設定します。
評価ボードについては，TK-850/JH3U-SP, TK-850/JG3H, TK-850/JH3E+NET, TK-850/JG3L+USB　の各ユーザーズ・マニュアルを参照してください。
CPU については，V850ES/JG3-U, V850ES/JH3-U, V850ES/JG3-H, V850ES/JH3-H, V850ES/JH3-E, V850ES/JJ3-E, V850ES/JG3-L の各ユーザーズ・マニュアル　ハードウエア編を参照してください。

4.3.2　パワーオン・チェック処理

スタートアップ・ファイルからコールされる usr_startchk 関数の中で，ファームウエア・アップデート・プログラムへ分岐するか，ユーザ・プログラムへ分岐するかの判定を行います。
TK-850/JH3U-SP の SW3，SW4 の状態を判定します。両方押されていた場合はファームウエア・アップデート・プログラムを実行，それ以外の場合はユーザ・プログラムを実行します。

```
#pragma ioreg

#define SW_PUSHED   0x00      /* pushed switch SW3 and SW4 */
#define SW_STATUS   0x03      /* switch status SW3 and SW4 */

s32 usbf_fwup_pwonchk_usr(void);

s32 usbf_fwup_pwonchk_usr(void){
          int       ret = -1;
          unsigned char  sts;

          sts = P9H;
          if ((sts & SW_STATUS) == SW_PUSHED) {
                    ret = 0;
          }

          return ret;
}
```

SW3, SW4 の状態を判定します

図 4-7　パワーオン・チェック処理

4.4 メイン・ルーチン

ブート処理が終了すると，main 関数へ分岐してメイン・ルーチンが実行されます。
メイン・ルーチンでは，CDC のシリアル通信処理の初期化を行ったあと，ファームウエア・アップデート・プログラムを実行します。

```
#define   SERIAL_BUF_SIZE     512

static unsigned char          serial_buf[SERIAL_BUF_SIZE];

int
main(int argc, char **argv)
{
                    __EI();

                    /* Initialize */
                    usbf_fwup_drvif_init(serial_buf, SERIAL_BUF_SIZE);

                    /* Update flash memory */
                    usbf_fwup();

                    return 0;
}
```

CDC のシリアル通信処理の初期化を行う

ファームウエア・アップデート処理を実行

図 4-8　メイン・ルーチン

4.4.1 USB通信の初期化処理（usbf_fwup_drvif.c）

USB でのシリアル通信のための初期化処理を行います。usbf850_devif_init 関数で，受信処理で使用する関数を定義した構造体を渡します。また，戻り値で CDC 処理内での動作関数を定義した構造体のポインタをもらい，その中の初期化関数を呼び出します。

```
void usbf_fwup_drvif_init(u08 *buf, s32 buf_len)
{
        recv_buf = buf;
        recv_buf_size = buf_len;

        cdc_funcs = usbf850_devif_init(&serial_funcs);
        cdc_funcs->init();

        usbf_fwup_drvif_clear_buffer_();
}
```

受信バッファのポインタとサイズを設定

CDC の初期化

受信バッファのクリア

図 4-9　USB 通信処理の初期化

cdc_funcs と serial_funcs の定義は，同ソース内で行っています。
usbf_fwup_drvif_read 関数については，4. 7. 3　EP1 監視処理を参照してください。

```
static const struct usb_cdc_funcs_st *cdc_funcs = (const struct usb_cdc_funcs_st *)0;

static const struct usb_serial_funcs_st serial_funcs = {
    usbf_fwup_drvif_read
};
```

受信処理に usbf_fwup_drvif_read 関数を設定

図 4-10　cdc_funcs と serial_funcs の定義

usb_serial_func_st 構造体，usb_cdc_func_st 構造体は usbf850_drvif.h で定義されています。

```
#ifndef __USBF850_DRVIF_H__
#define __USBF850_DRVIF_H__

struct usb_serial_funcs_st {
    void    (*read)(UINT8 len);
};

struct usb_cdc_funcs_st {
    void    (*init)(void);
    void    (*int0b)(void);
    void    (*int1b)(void);
    INT32   (*datasend)(UINT8* data, INT32 len, INT8 ep);
    INT32   (*datareceive)(UINT8* data, INT32 len, INT8 ep);
};

const struct usb_cdc_funcs_st *usbf850_devif_init(const struct usb_serial_funcs_st *funcs);

#endif/* __USBF850_DRVIF_H__ */
```

受信処理をするための構造体

CDC 動作のための構造体

図 4-11　usb_serial_funcs_st 構造体と usb_cdc_funcs_st 構造体

usbf850_devif_init 関数は usbf850_jx3u.c 内で定義しています。

```
const struct usb_cdc_funcs_st *usbf850_devif_init(const struct usb_serial_funcs_st *funcs)
{
    serial_funcs = funcs;

    return &cdc_funcs;
}
```

cdc_funcs のポインタを返す

図 4-12　usbf850_devif_init 関数

serial_funcs と cdc_funcs の定義は，同ソース内で行っています。

```
static const struct usb_serial_funcs_st *serial_funcs = (const struct usb_serial_funcs_st *)0;

static const struct usb_cdc_funcs_st cdc_funcs = {
    usbf850_init,
    usbf850_intusb0b,
    usbf850_intusb1b,
    usbf850_data_send,
    usbf850_data_receive
};
```

CDC 用関数定義構造体

図 4-13　serial_fincs と cdc_funcs の定義

usbf_fwup_drvif_init 関数内の“cdc_funcs->init();”は，上記定義より，usbf850_init 関数が呼ばれます。

usbf850_init 関数を次に示します。usbf850_intusb0b 関数，usbf850_intusb1b 関数，usbf850_data_send 関数，usbf850_data_receive 関数については，4. 7　CDC（Communication Device Class）を参照してください。

```
void usbf850_init(void)
{
    INT32 i;

    UF0E0NA = C_EP0NKA;
    while (UF0E0NA != C_EP0NKA) {
        UF0E0NA = C_EP0NKA;
    }

    /* The initialization of the request data register area */
    UF0DSTL = 0x00;        /* Bus Powered */
    UF0E0SL = 0x00;
    UF0E1SL = 0x00;
    UF0E2SL = 0x00;

    /* The total byte of the UF0CIEa register is long. */
    UF0DSCL = (C_CONF_DSC_wTotalLength_L - 1);
    for (i = 0; i < T_DEV_DSC[0]; i++) {
        USBF850REG_SET((UF0DD0_ADDRESS + (i*sizeof(INT16))), T_DEV_DSC[i]);
    }
    for (i = 0; i < T_CONF_DSC[2]; i++) {
        USBF850REG_SET((UF0CIE0_ADDRESS + (i*sizeof(INT16))), T_CONF_DSC[i]);
    }

    /* The initialization of the request data register area (The ending) */
    UF0MODC = 0x00;    /* SET GET_DESCRIPTOR REQ. AUTO */

    /* The setting of Interface and Endpoint */
    UF0AIFN = 0x80;    /* Interface0,1 Support */
    UF0AAS = 0x00;     /* It is not in the Alternate setting. */

    /* SFR_UF0EnIM = xx// (It sets EP not to use to 0x00.) */
    UF0E1IM = 0x40;
    UF0E2IM = 0x40;
    UF0E7IM = 0x20;

    /* The setting of Interface and Endpoint (The ending) */
    UF0E0NA = 0x00;    /* RESET EP0 NAK SEND */   /* RESET EP0 NAK SEND */

    /* The interrupt and FIFO relation register initialization */
    UF0IC0 = C_IC0_ALL;    /* interrupt clear */
    UF0IC1 = C_IC1_ALL;    /* interrupt clear */
    UF0IC2 = C_IC2_ALL;    /* interrupt clear */
    UF0IC3 = C_IC3_ALL;    /* interrupt clear */
    UF0IC4 = C_IC4_ALL;    /* interrupt clear */

    UF0FIC0 = C_FIC0_ALL;  /* The FIFO clearness, the counter reset */
    UF0FIC1 = C_FIC1_ALL;  /* The FIFO clearness */

    /* The setting of a interrupt mask */
    UF0IM0 = C_IM0_ALL;                    /* ALL MASK */
    UF0IM1 = (C_IM1_ALL & (~C_CPUDEC));       /* CPUDEC mask clear */
    UF0IM2 = C_IM2_ALL;                    /* ALL Mask */
    UF0IM3 = (C_IM3_ALL & (~C_BKO1DT));       /* BKO1DT mask clear */
    UF0IM4 = C_IM4_ALL;                    /* ALL Mask */

    usbf850_setfunction_communication();

    /* D+ Pullup */
    PM4 = 0xFC;
    P4  = 0x02;
}
```

自動リクエストを含むすべてのリクエストに NAK 応答する

リクエスト格納レジスタの初期化

GetDiscriptor 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する

サポートするインタフェースの数，オルタナティブ設定の状態，エンドポイント情報をレジスタに設定，エンドポイントの設定

NAK 設定解除

割り込み要因ごとのマスクを設定

CDC リクエスト登録

D+信号をプルアップ設定

図 4-14　usbf850_init 関数

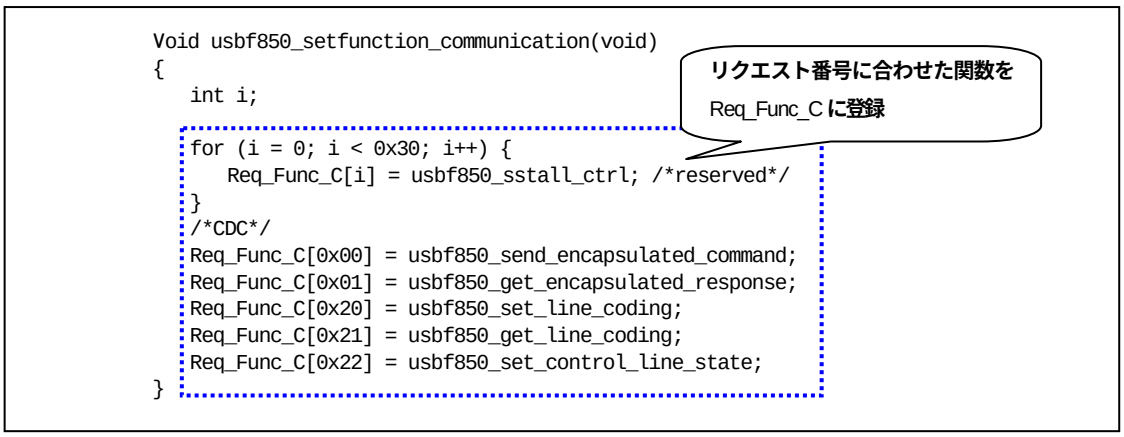


図 4-15　CDC リクエストを登録

CDC リクエストについては，4.7　CDC（Communication Device Class）を参照してください。

4.5 割り込み処理

4.5.1 フラッシュ環境時の割り込み（usbf_fwup_intentry.s）

フラッシュ環境とは，内蔵フラッシュ・メモリを操作（書き換え等）が行える状態を指します。処理内でセルフ・ライブラリの FlashEnv 関数が呼ばれることで，フラッシュ環境の開始／終了が実現されます。
フラッシュ環境時は，内蔵フラッシュ・メモリの参照ができないため，ノンマスカブル割り込みが発生した場合は内蔵 RAM の先頭へ，マスカブル割り込み，ソフトウエア例外，例外トラップが発生した場合は内蔵 RAM の先頭＋4 バイトの領域へジャンプします。フラッシュ環境時の割り込みエントリは，usbf_fwup_intentry.s に記述しています。
ただし，ファームウエア・アップデート・プログラムでは，フラッシュ環境時に割り込みは使用しないので，ここで行う処理はありません。しかし，セルフ・ライブラリは該当処理がなくても，ダミー・セクションとして，定義が必要のため，下記の記述を行っています。

```
-----------------------------------------------------
--  flash_int.s                                    --
-----------------------------------------------------
        .section "SelfLib_ToRamUsrInt.text", text

        .globl __SELFLIB_NMI_VECTOR
        .globl __SELFLIB_INT_VECTOR

        .align 4
__SELFLIB_NMI_VECTOR:
        jr __nmi_check_entry

        .align 4
__SELFLIB_INT_VECTOR:
        jr __int_check_entry
-----------------------------------------------------
        .section "SelfLib_ToRamUsr.text", text
        .align 4
__nmi_check_entry:
        reti

        .align 4
__int_check_entry:
        reti
-----------------------------------------------------
```

割り込みエントリ

割り込み前処理

図 4-16　フラッシュ環境時の割り込み

4.6　内蔵フラッシュ・メモリへの書き込み

ファームウエア・アップデート・プログラムは，内蔵フラッシュ・メモリの内容を書き換えることにより，ファームウエア，およびメモリ上の任意の領域を更新します。
ファームウエア・アップデート・プログラムでは，内蔵フラッシュ・メモリの書き込みを行うために，セルフ・ライブラリを使用します。
セルフ・ライブラリは，現在 Type01 から Type04 まであり，使用するデバイスに対応したものが必要で，本評価ボードでは，Type04 のセルフ・ライブラリを使用します。
セルフ・ライブラリの詳細については，V850 マイクロコントローラ　ユーザーズ・マニュアル　フラッシュ・メモリ・セルフ・プログラミング・ライブラリ　Type04 Ver.1.20 を参照してください。

4.6.1　フラッシュ・メモリの書き込み処理

評価ボードで使用されるそれぞれの製品の内蔵フラッシュ・メモリは，下記に示すブロックで構成されます。

- uPD70F3769（V850ES/JH3-U）　：　ブロック 0-127
- uPD70F3760（V850ES/JG3-H）　：　ブロック 0-63
- uPD70F3783（V850ES/JH3-E）　：　ブロック 0-127
- uPD70F3796（V850ES/JG3-L）　：　ブロック 0-127

セルフ書き換えは，ブロック単位で内蔵フラッシュ・メモリへ書き込みを行います。
usbf_fwup.c で定義している usbf_fwup_from_write 関数は，内蔵フラッシュ・メモリの指定されたブロックへ書き込み処理を行います。書き込みを行うブロック，書き込むデータは，flash_data_st 構造体の形式で指定します。
flash_data_st 構造体は，usbf_fwup.h で宣言しています。

```
#ifndef __USBF_FWUP_H__
#define __USBF_FWUP_H__

#define FLASH_BLOCK_SIZE          (4096)

#define BOOT_FLAG_SETINFO         (0x01)
#define BOOT_FLAG_BOOTSWAP        (0x02)

struct flash_data_st {
    u32     block;
    u32     data_length;
    u08     data[FLASH_BLOCK_SIZE];
};

void    usbf_fwup(void);
u32     usbf_fwup_from_write(struct flash_data_st *data, u08 flag);

#endif/* __USBF_FWUP_H__ */
```

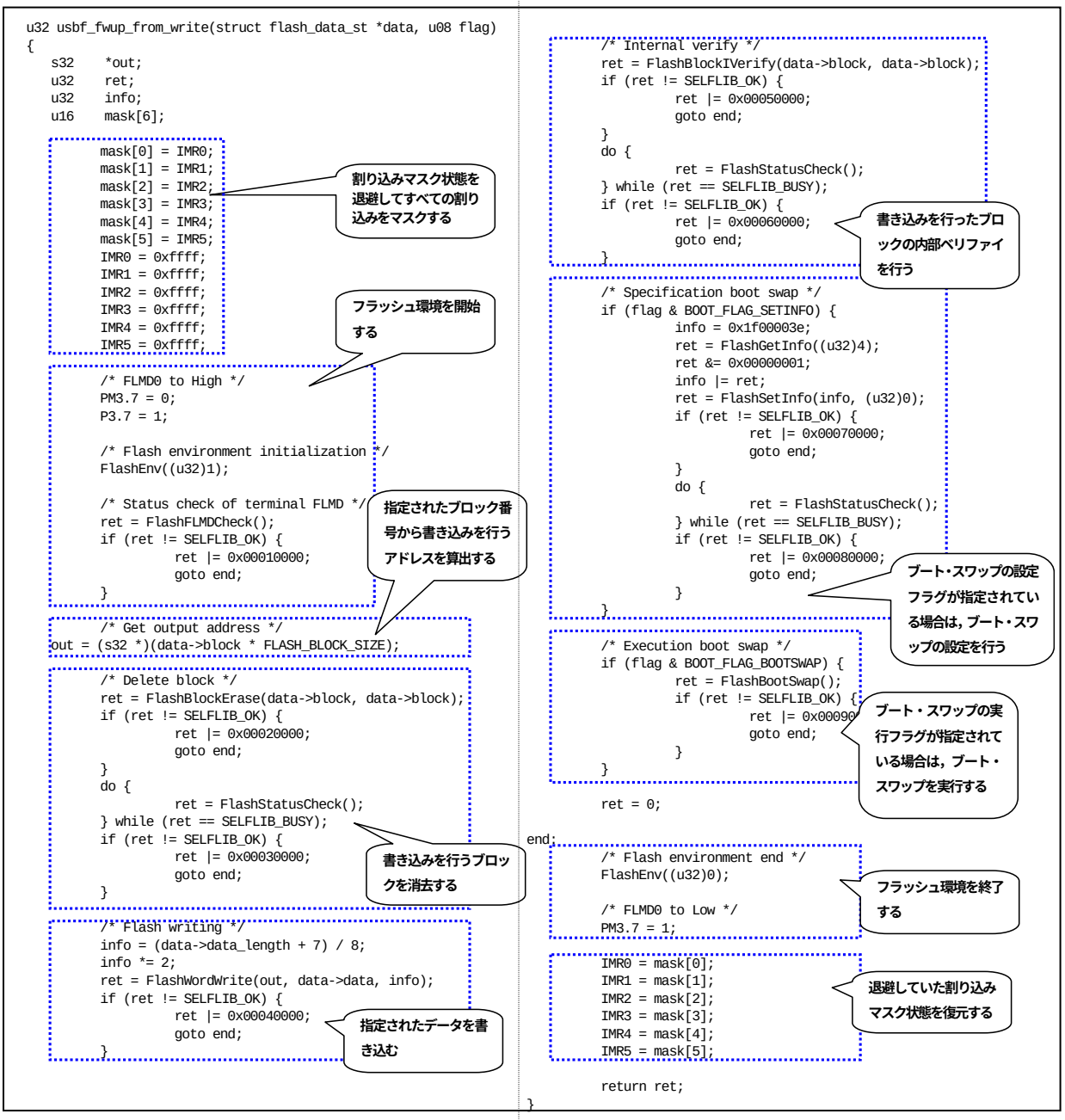
1 ブロックあたりのバイト数

ブート・スワップ処理フラグ

usbf_fwup_from_write 関数にデータを渡すための構造体

図 4-17　flash_data 構造体の定義

flash_data_st 構造体の block メンバは書き込みを行うブロックの番号，data メンバは書き込むデータ，data_length メンバは書き込むデータのバイト数を指定します。
usbf_fwup_from_write 関数は，セルフ・ライブラリが提供しているフラッシュ関数を使用して内蔵フラッシュ・メモリへデータを書き込みます。




```
u32 usbf_fwup_from_write(struct flash_data_st *data, u08 flag)
{
    s32    *out;
    u32    ret;
    u32    info;
    u16    mask[6];

        mask[0] = IMR0;
        mask[1] = IMR1;
        mask[2] = IMR2;
        mask[3] = IMR3;
        mask[4] = IMR4;
        mask[5] = IMR5;
        IMR0 = 0xffff;
        IMR1 = 0xffff;
        IMR2 = 0xffff;
        IMR3 = 0xffff;
        IMR4 = 0xffff;
        IMR5 = 0xffff;

        /* FLMD0 to High */
        PM3.7 = 0;
        P3.7 = 1;

        /* Flash environment initialization */
        FlashEnv((u32)1);

        /* Status check of terminal FLMD */
        ret = FlashFLMDCheck();
        if (ret != SELFLIB_OK) {
                ret |= 0x00010000;
                goto end;
        }

        /* Get output address */
    out = (s32 *)(data->block * FLASH_BLOCK_SIZE);

        /* Delete block */
        ret = FlashBlockErase(data->block, data->block);
        if (ret != SELFLIB_OK) {
                ret |= 0x00020000;
                goto end;
        }
        do {
                ret = FlashStatusCheck();
        } while (ret == SELFLIB_BUSY);
        if (ret != SELFLIB_OK) {
                ret |= 0x00030000;
                goto end;
        }

        /* Flash writing */
        info = (data->data_length + 7) / 8;
        info *= 2;
        ret = FlashWordWrite(out, data->data, info);
        if (ret != SELFLIB_OK) {
                ret |= 0x00040000;
                goto end;
        }

        /* Internal verify */
        ret = FlashBlockIVerify(data->block, data->block);
        if (ret != SELFLIB_OK) {
                ret |= 0x00050000;
                goto end;
        }
        do {
                ret = FlashStatusCheck();
        } while (ret == SELFLIB_BUSY);
        if (ret != SELFLIB_OK) {
                ret |= 0x00060000;
                goto end;
        }

        /* Specification boot swap */
        if (flag & BOOT_FLAG_SETINFO) {
                info = 0x1f00003e;
                ret = FlashGetInfo((u32)4);
                ret &= 0x00000001;
                info |= ret;
                ret = FlashSetInfo(info, (u32)0);
                if (ret != SELFLIB_OK) {
                        ret |= 0x00070000;
                        goto end;
                }
                do {
                        ret = FlashStatusCheck();
                } while (ret == SELFLIB_BUSY);
                if (ret != SELFLIB_OK) {
                        ret |= 0x00080000;
                        goto end;
                }
        }

        /* Execution boot swap */
        if (flag & BOOT_FLAG_BOOTSWAP) {
                ret = FlashBootSwap();
                if (ret != SELFLIB_OK) {
                        ret |= 0x00090000;
                        goto end;
                }
        }

        ret = 0;

end:
        /* Flash environment end */
        FlashEnv((u32)0);

        /* FLMD0 to Low */
        PM3.7 = 1;

        IMR0 = mask[0];
        IMR1 = mask[1];
        IMR2 = mask[2];
        IMR3 = mask[3];
        IMR4 = mask[4];
        IMR5 = mask[5];

        return ret;
}
```

図 4-18　内蔵フラッシュ・メモリへの書き込み処理

4.6.2 ブート・スワップ機能

対象としている製品は，ユーザ・プログラムを書き換えるときに，ブート領域の書き換え中に電源遮断等の事故が発生しても，ブート領域を保障し，正常に動作するためにブート・スワップ機能をサポートしています。
対象としている製品は，ブロック番号 0-15 とブロック番号 16-31(※)を入れ替えること（スワップ）ができます。

※V850ES/JG3-L ではブロック番号 0-7 とブロック番号 8-15 が入れ替わります。

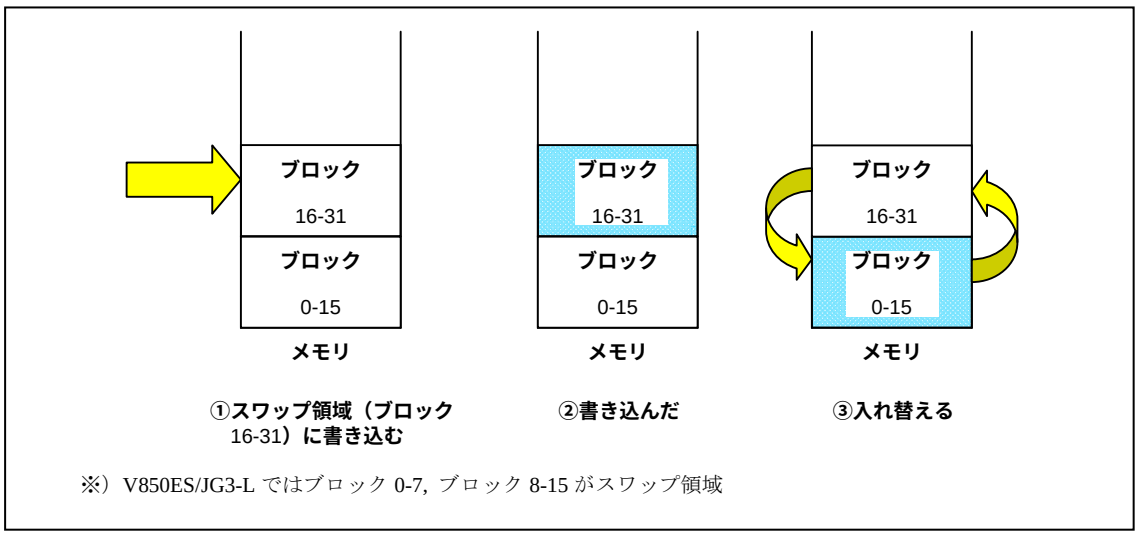


図 4-19　ユーザ・プログラム書き換え時のブート・スワップ機能

4.6.3 ファームウエア・アップデート処理

内蔵フラッシュ・メモリを書き換える場合は，ブロック単位で行います。ファームウエア・アップデート・プログラムでは，書き換える領域のブロックをコピーし，コピーしたブロックを書き換えてから，そのブロックを書き込みます。これにより，1 バイト単位の書き換えを実現します。

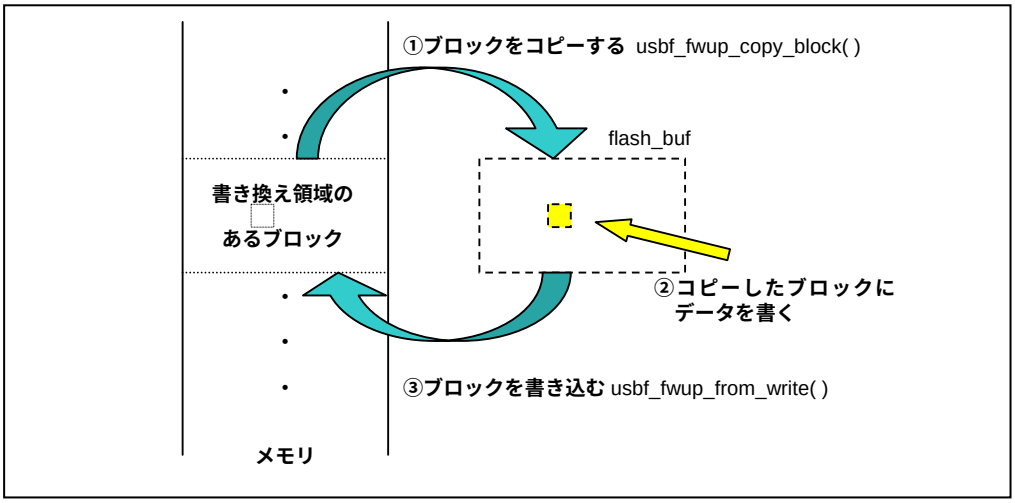


図 4-20　書き換え処理イメージ

usbf_fwup.c の usbf_fwup 関数で，PC からのデータ受信／応答，データの書き換えを行います。

```
void usbf_fwup(void)
{
    u32     first_addr;
    s32     ret;
    u08     code;

    /* start */
    ret = usbf_fwup_recv_record();
    while (ret != RECORD_TYPE_START) {
        usbf_fwup_send_startres(RESPONSE_NAK);

        ret = usbf_fwup_recv_record();
    }

    FlashInit();
    usbf_fwup_send_startres(RESPONSE_ACK);

    /* first block */
    ret = usbf_fwup_recv_record();
    while (ret != RECORD_TYPE_DATA) {
        if (ret == RECORD_TYPE_START) {
            usbf_fwup_send_startres(RESPONSE_ACK);
        }
        else if (ret == RECORD_TYPE_END) {
            goto end;
        }
        else {
            usbf_fwup_send_datares(RESPONSE_NAK);
        }

        ret = usbf_fwup_recv_record();
    }
    flash_addr = usbf_fwup_get_addr();
    flash_block = (u32)(flash_addr / FLASH_BLOCK_SIZE);

    if (flash_block < 16) {
        first_addr = flash_addr;
        flash_buf.block = flash_block + 16;
        flash_block = usbf_fwup_recv_block();
        if (first_addr == 0) {
            usbf_fwup_replace_apstart();
        }
        while (flash_block < 16) {
            __DI();
            ret = (s32)usbf_fwup_from_write(&flash_buf, 0);
            __EI();
            if (ret != 0) {
                code = ERROR_FLASH_WRITE;
                goto error;
            }
            flash_buf.block = flash_block + 16;
            flash_block = usbf_fwup_recv_block();
        }
        __DI();
        ret = (s32)usbf_fwup_from_write(&flash_buf, BOOT_FLAG_SETINFO);
        __EI();
        if (ret != 0) {
            code = ERROR_FLASH_WRITE;
            goto error;
        }
```

PC からのデータを受信し，開始レコードでなかったら，NAK を送信する

セルフ・ライブラリを初期化

ACK を送信

データ・レコードでなかった場合の処理。
開始レコードなら ACK を返して，再度データ受信する。
終了レコードなら終了する。
それ以外なら NAK を返す。

データ・レコード内のロード・アドレスを取得し，ブロック番号を取得する。

ブート領域の場合

書き換えのブロックをスワップ領域とし，1 ブロック分の受信をする。
先頭アドレスが 0 からなら，ユーザ・プログラムへのジャンプ処理を変更する

1 ブロックずつ受信し，書き込んでいく。書き込みブロックが 16 を越えたらループを抜ける。
抜けた後は，書き込み終了後ブート・スワップを行う。

図 4-21　ファームウエア・アップデート処理（1/2）

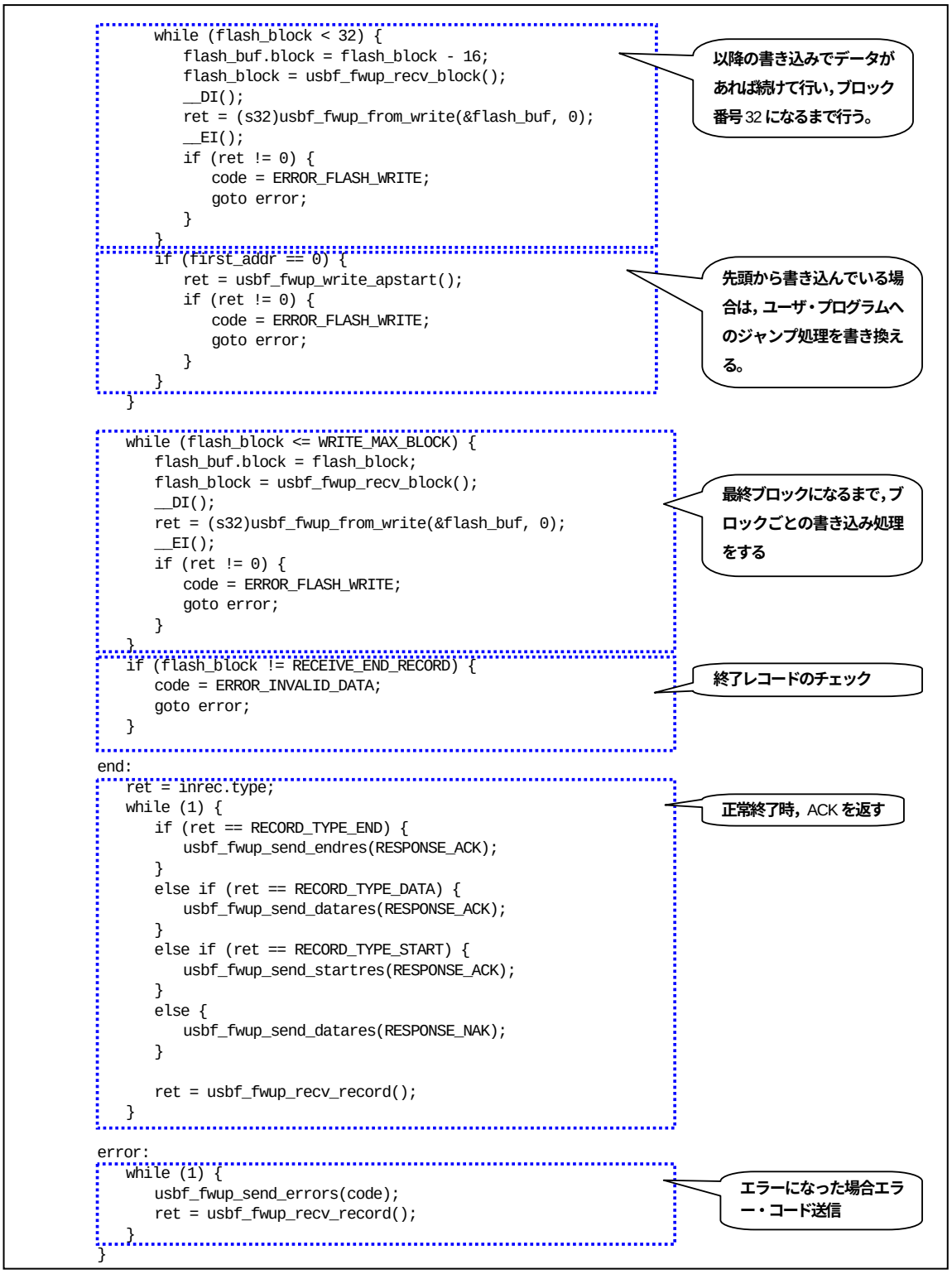


図 4-21　ファームウエア・アップデート処理（2/2）

4.6.4 ユーザ・プログラムの書き換え

ユーザ・プログラムを書き込むときは，apstart セクション内を変更し，起動時にユーザ・プログラムが動作するようにします。
受信したユーザ・プログラムのブート処理（リセット・セクション）を従来（ファームウエア・アップデート）のブート処理での分岐先（ユーザ・プログラムへジャンプするコード：apstart セクション）として取り出して書き込み，元のブート処理を従来（ファームウエア・アップデート）のブート処理に変更します。これによりブート処理部分を保持し，ファームウエア・アップデートの操作を再度実施できようになります。

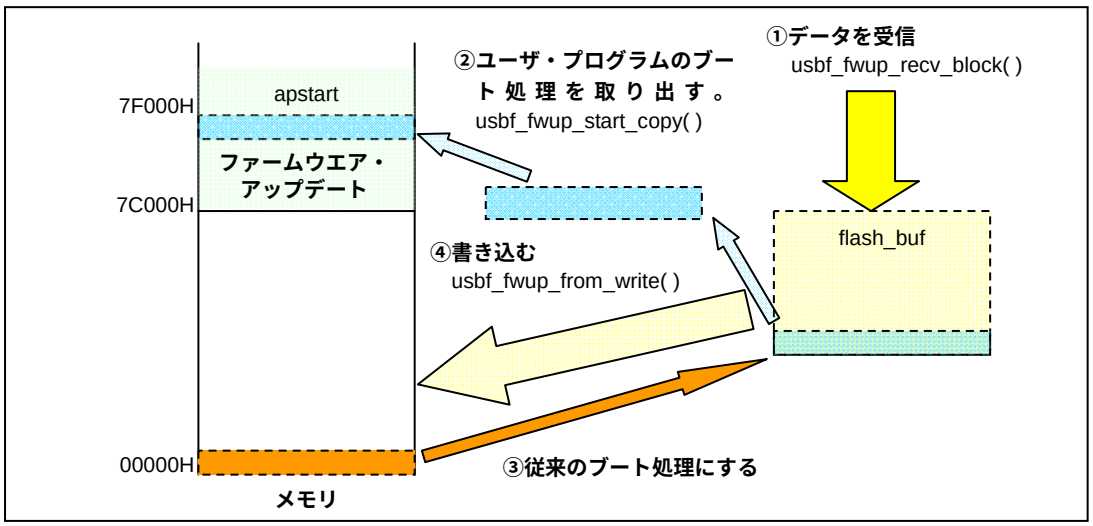


図 4-22　ユーザ・プログラム書き換え時のブート処理書き換えイメージ

起動処理では，最初にファームウエア・アップデート・プログラムのブート処理へ移動し，その処理内で起動条件（SW3, SW4 の状態）を判定し，apstart へ移動し，そこからユーザ・プログラムの先頭へ移動します。

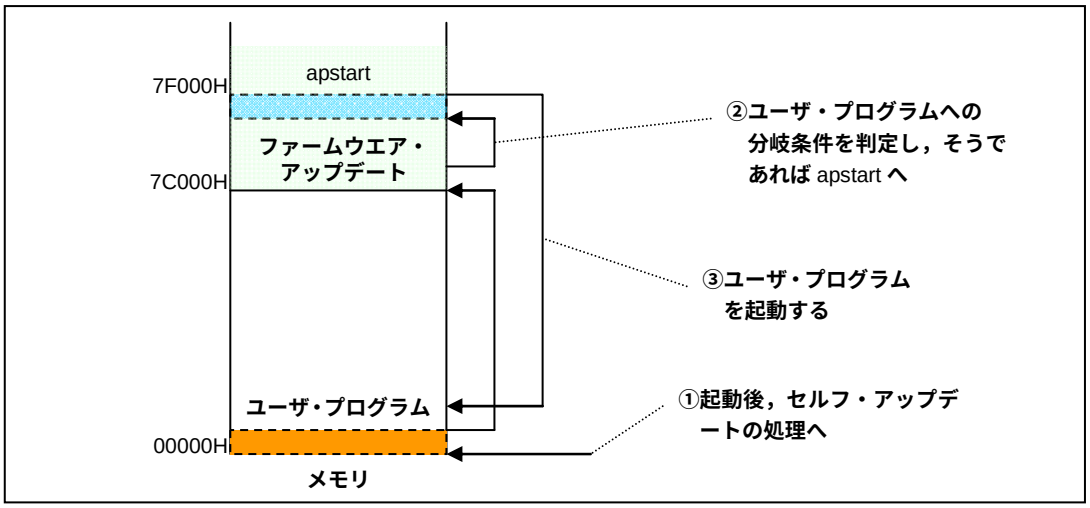


図 4-23　ユーザ・プログラムへの分岐イメージ

```
static void usbf_fwup_replace_apstart(void)
{
    u16    *in = (u16 *)0;
    u16    *out;
    s32    i;

    out = (u16 *)flash_buf.data;
    usbf_fwup_start_copy(out);
    for (i = 7; i >= 0; i--) {
        out[i] = in[i];
    }
}
```

メモリの先頭を示す

受信データのブート処理を取り出す

従来の処理に変更する

図 4-24　ブート処理の変更

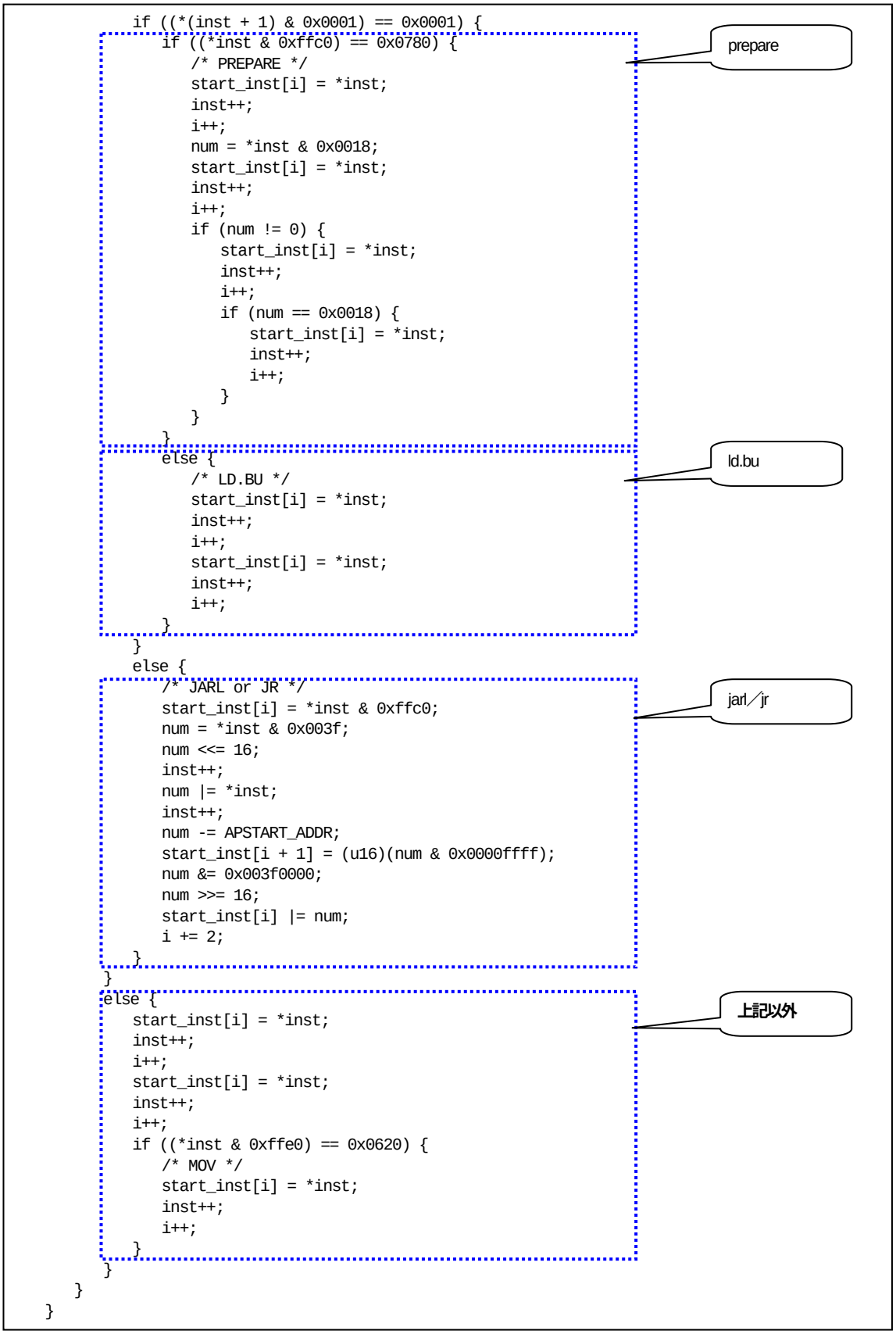
apstart へ書き込む処理を編集します。

```
static void usbf_fwup_start_copy(u16 *inst)
{
    s32    num;
    s32    i;

    for (i = 23; i >= 0; i--) {
        start_inst[i] = 0xffff;
    }

    i = 0;
    while (i < 8) {
        if (*inst == 0xffff) {
            break;
        }
        if ((*inst & 0x0700) < 0x0600) {
            if ((*inst & 0x0780) == 0x0580) {
                /* Bcond */
                num = *inst & 0xf800;
                num >>= 4;
                num |= (*inst & 0x0070);
                num >>= 3;
                num -= APSTART_ADDR;
                if (num > 255 || num < -256) {
                    usbf_fwup_add_jr(i, num);
                    num = i * 2 + 16;
                }
                start_inst[i] = *inst & 0x078f;
                num <<= 3;
                start_inst[i] |= num & 0x00000070;
                num <<= 4;
                start_inst[i] |= num & 0x0000f800;
            }
            else {
                start_inst[i] = *inst;
            }
            inst++;
            i++;
        }
        else if ((*inst & 0x07c0) == 0x0780) {
```

start_inst に apstart へ書き込むコードを入れておく。
最初は 0xffff で初期化する。

jcond／bcound

図 4-25　ブート処理の書き換え（1/2）




```
        if ((*(inst + 1) & 0x0001) == 0x0001) {
            if ((*inst & 0xffc0) == 0x0780) {
                /* PREPARE */
                start_inst[i] = *inst;
                inst++;
                i++;
                num = *inst & 0x0018;
                start_inst[i] = *inst;
                inst++;
                i++;
                if (num != 0) {
                    start_inst[i] = *inst;
                    inst++;
                    i++;
                    if (num == 0x0018) {
                        start_inst[i] = *inst;
                        inst++;
                        i++;
                    }
                }
            }
            else {
                /* LD.BU */
                start_inst[i] = *inst;
                inst++;
                i++;
                start_inst[i] = *inst;
                inst++;
                i++;
            }
        }
        else {
            /* JARL or JR */
            start_inst[i] = *inst & 0xffc0;
            num = *inst & 0x003f;
            num <<= 16;
            inst++;
            num |= *inst;
            inst++;
            num -= APSTART_ADDR;
            start_inst[i + 1] = (u16)(num & 0x0000ffff);
            num &= 0x003f0000;
            num >>= 16;
            start_inst[i] |= num;
            i += 2;
        }
    }
    else {
        start_inst[i] = *inst;
        inst++;
        i++;
        start_inst[i] = *inst;
        inst++;
        i++;
        if ((*inst & 0xffe0) == 0x0620) {
            /* MOV */
            start_inst[i] = *inst;
            inst++;
            i++;
        }
    }
  }
}
```

図 4-25　ブート処理の書き換え（2/2）

4.6.5 データ受信処理

ファームウエア・アップデート・プログラムは，PC とのシリアル通信を行い，書き換えるデータを受信します。
通信インタフェース仕様については，8.1　書き換え通信インタフェース仕様を参照してください。

```
static s32 usbf_fwup_recv_block(void)
{
    s32    ret;
    u32    in_addr;
    u32    out_addr;
    s32    in_len;
    s32    in_idx;
    s32    out_idx;

    usbf_fwup_copy_block(flash_block);
    flash_buf.data_length = FLASH_BLOCK_SIZE;

    out_addr = flash_block * FLASH_BLOCK_SIZE;
    do {
        if (out_addr == flash_addr) {
            out_idx = 0;
            in_idx  = 4;
        }
        else if (out_addr > flash_addr) {
            in_idx = out_addr - flash_addr + 4;
            out_idx = 0;
        }
        else {
            out_idx = flash_addr - out_addr;
            in_idx = 4;
        }
        in_len = inrec.len - 1;
        while (in_idx < in_len) {
            if (out_idx >= FLASH_BLOCK_SIZE) {
                ret = flash_block + 1;
                goto end;
            }
            flash_buf.data[out_idx] = inrec.data[in_idx];
            out_idx++;
            in_idx++;
        }
        usbf_fwup_send_datares(RESPONSE_ACK);
        ret = usbf_fwup_recv_record();

        while (1) {
            if (ret == RECORD_TYPE_DATA) {
                in_addr = usbf_fwup_get_addr();
                if (in_addr > flash_addr) {
                    break;
                }
                usbf_fwup_send_datares(RESPONSE_ACK);
            }
            else if (ret == RECORD_TYPE_END) {
                ret = RECEIVE_END_RECORD;
                goto end;
            }
            else if (ret == RECORD_TYPE_START) {
                usbf_fwup_send_startres(RESPONSE_ACK);
            }
            else {
                usbf_fwup_send_datares(RESPONSE_NAK);
            }

            ret = usbf_fwup_recv_record();
        }
```

指定のブロックをコピーする

1 レコード分を受信する。
最後まで受信するか，ブロックがいっぱいになったらループを抜ける。

ACK を送信後
再度レコードを受信する

受信レコードを判定する

図 4-26　1 ブロック・データ受信（1/2）

```
        flash_addr = in_addr;
        ret = flash_addr / FLASH_BLOCK_SIZE;

    } while (ret == flash_block);

end:
    return ret;
}
```

ブロックが変わったらループを抜ける

図 4-26　1 ブロック・データ受信（2/2）

次に示すのは，1 レコードを受信する処理です。

```
static s32 usbf_fwup_recv_record(void)
{
    s32    ret;
    s32    i;
    u16    chk;


    /* Read record */
    usbf_fwup_drvif_clear_buffer();
    ret = usbf_fwup_drvif_recv(&inrec.type, 1);
    ret = usbf_fwup_drvif_recv(&inrec.len, 1);
    if (inrec.len == 0) {
        ret = -1;
        goto end;
    }
    chk = inrec.len;
    if (inrec.len > 1) {
        ret = usbf_fwup_drvif_recv(inrec.data, inrec.len - 1);
        for (i = inrec.len - 2; i >= 0; i--) {
            chk += inrec.data[i];
        }
    }
    ret = usbf_fwup_drvif_recv(&inrec.sum, 1);

    /* Check sum */
    chk ^= 0xffff;
    chk &= 0x00ff;
    if (chk != inrec.sum) {
        ret = -1;
        goto end;
    }
    ret = inrec.type;

end:
    return ret;
}
```

レコード・タイプと長さを受信する

指定の長さに達するまでループする。
また，受信データはチェック・サム用に計上する。

チェック・サム

図 4-27　1 レコード受信

4.7 CDC（Communication Device Class）

ファームウエア・アップデート・プログラムで使用されている CDC 処理について記載します。
USB の CDC 仕様については，Universal Serial Bus Class Definitions for Communications Devices を参照してください。
ファームウエア・アップデート・プログラムで使用されている CDC は，Abstract Control Model で，対応するクラス・リクエストは次のようになります。

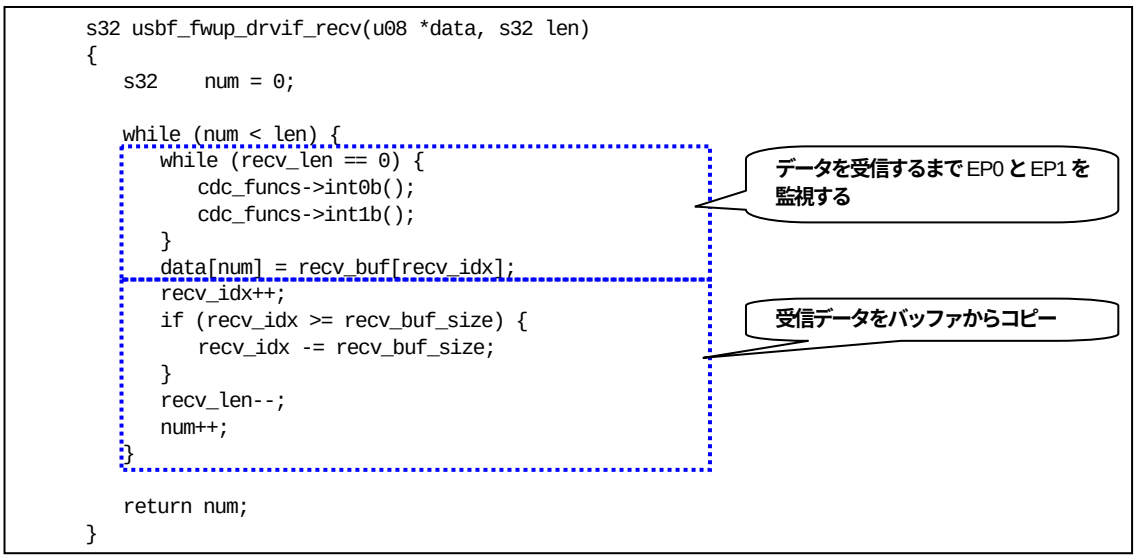
備考　USB 規格の策定と管理は，USB Implementers Forum（USB-IF）という団体が行っています。
Universal Serial Bus Class Definitions for Communications Devices の詳細については，USB-IF の公式ウェブサイト（www.usb.org）を参照してください。

表 4-9　対応するクラス・リクエスト

クラス・リクエスト	内　　容
SendEncapsulatedCommand	コミュニケーション・クラス・インタフェースの制御プロトコルのフォーマットでコマンドを発行するためのリクエスト
GetEncapsulatedRespons	コミュニケーション・クラス・インタフェースの制御プロトコルのフォーマットで応答を要求するためのリクエスト
SetLineCoding	シリアル通信の通信フォーマットを指定するためのリクエスト
GetLineCoding	デバイス側の現在の通信フォーマット設定を取得するためのリクエスト
SetControlLineState	RS-232/V.24形式の制御信号

4.7.1 ポーリングによるEP監視処理

EP 監視処理は，割り込みベクタではなく，ポーリングで EP（エンドポイント）の割り込みフラグを監視することで，EP0（コントロール転送用エンドポイント），EP1（バルクイン用エンドポイント）の FIFO にデータがあるかどうかを監視します。
次に，受信時における EP 監視処理を示します。




```
s32 usbf_fwup_drvif_recv(u08 *data, s32 len)
{
    s32     num = 0;

    while (num < len) {
        while (recv_len == 0) {
            cdc_funcs->int0b();
            cdc_funcs->int1b();
        }
        data[num] = recv_buf[recv_idx];
        recv_idx++;
        if (recv_idx >= recv_buf_size) {
            recv_idx -= recv_buf_size;
        }
        recv_len--;
        num++;
    }

    return num;
}
```

図 4-28　データ受信時の EP 監視

関数内の“cdc_funcs->int0b();”は，初期設定により usbf850_intusb0b 関数が呼ばれます。同様に“cdc_funcs->int1b();”は，usbf850_intusb1b 関数が呼ばれます。

4.7.2 EP0 監視処理

EP0 はコントロール転送用のエンドポイントです。ここでは，ハードウエアでは対応しない標準リクエスト，クラス・リクエスト，ベンダ・リクエストを監視します。
次に EP0 の監視処理を示します。

```
void usbf850_intusb0b(void)
{
    UINT8 request;
    volatile UINT8 tmpl = 0;
    volatile UINT8 tmph = 0;

    if(UF0IS0 & C_RSUSPD){
        UF0IC0 = C_RSUSPDC;
        if(UF0EPS1 & C_RSUM){
            UF0IC0 = 0x00;          /* interrupt clear */
            UF0IC1 = 0x00;          /* interrupt clear */
            UF0IC2 = 0x00;          /* interrupt clear */
            UF0IC3 = 0x00;          /* interrupt clear */
            UF0IC4 = 0x00;          /* interrupt clear */
            return;
        }
    }

    if (UF0IS1 & C_CPUDEC) {
        UF0IC1 = (UINT8)~C_PROT; /*PROT interrupt clear*/

        UsbSetup_Data.RequstType = UF0E0ST;
        UsbSetup_Data.Request    = UF0E0ST;
        tmpl = UF0E0ST;
        tmph = UF0E0ST;
        UsbSetup_Data.Value      = (tmpl | ((tmph << 8) & 0xff00));
        tmpl = UF0E0ST;
        tmph = UF0E0ST;
        UsbSetup_Data.Index      = (tmpl | ((tmph << 8) & 0xff00));
        tmpl = UF0E0ST;
        tmph = UF0E0ST;
        UsbSetup_Data.Length     = (tmpl | ((tmph << 8) & 0xff00));

        if (UsbSetup_Data.RequstType & C_CLASS_REQUEST) {
            if (UsbSetup_Data.Index != C_IF0_DSC_bInterfaceNumber) {
                usbf850_sendstallEP0(); /*error*/
            }
            request = (UsbSetup_Data.Request & 0xff);

            /*Request Decode*/
            (*Req_Func_C[request])();
        }
        else if(UsbSetup_Data.RequstType & C_VENDER_REQUEST){
            usbf850_sendstallEP0(); /*error*/
        }
        else {
            usbf850_standardreq();
        }
    }
}
```

Resume または Suspend 状態の判定

UF0E0ST にデータがあったら割り込み要因をクリアする

リクエスト・データを構成する

クラス・リクエストの処理初期化の際に登録した関数を実行

ベンダ・リクエストか判定。STALL とする。

標準リクエストの処理

図 4-29　EP0 監視処理

（1）標準リクエスト

標準リクエストでは，ディスクプリタを取得します。

```
void usbf850_standardreq(void)
{
   if (UsbSetup_Data.Request == GETDESC) { /*GetDescriptor[String/Class]*/
      usbf850_getdesc();
   }
   else { /*error*/
      usbf850_sendstallEP0();
   }
}
```

リクエストがディスクプリタなら取得する。そうでない場合は STALL 制御処理。

図 4-30　標準リクエストの処理

```
void usbf850_getdesc(void)
{
   UINT8 len;
   UINT8 value;
   UINT8* tmp;

   if ((UsbSetup_Data.Value & 0xff00) == STRDESC) { /*String Descriptor*/
      value = (UINT8)(UsbSetup_Data.Value & 0xff);
      if (value >= (sizeof(USB_strings)/sizeof(USB_strings[0]))) {
         /*EP0 STALL*/
         usbf850_sendstallEP0();
         return ;
      }
      len = USB_strings[value][0];
      tmp = &(USB_strings[value][0]);
   }
   else {
      /*error*/
      usbf850_sendstallEP0();
      return ;
   }
   if (UsbSetup_Data.Length < len) {
      len = UsbSetup_Data.Length;
   }
   usbf850_data_send(tmp,len,C_EP0);
}
```

文字列によるディスクリプタを設定

長さを取得

EP0 から送信

図 4-31　ディスクリプタ送信処理

ディスクリプタ・データ（USB_string）は，次のように定義されています。DSTR, USTR はロケール設定，および Unicode 設定のためのマクロです。

```
/* 0 : Language Code*/
DSTR(LangString, 2, (0x09,0x04));
/* 1 : Manufacturer*/
USTR(ManString, 23, ('R','e','n','e','s','a','s',' ','E','l','e','c','t','r','o','n','i','c','s',' ','C','o','.'));
/* 2 : Product*/
USTR(ProductString, 10, ('U','S','B',' ','C','o','m','D','r','v'));
/* 3 : Serial Number*/
USTR(SerialString,  10, ('0','_','9','8','7','6','5','4','3','2'));

unsigned char *USB_strings[]={LangString,ManString,ProductString,SerialString};
```

図 4-32　ディスクリプタ・データの定義

（2）クラス・リクエスト

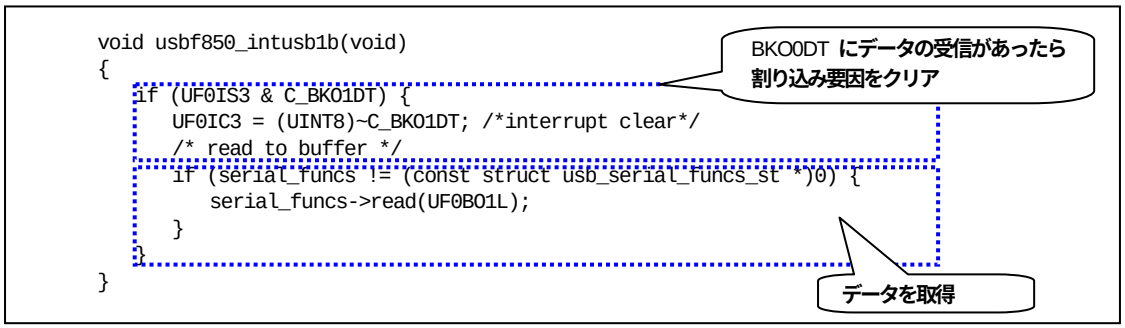
Req_Func_C に登録されているクラス・リクエストの一覧を示します。リクエストにより次に示す関数が実行されます。

表 4-10　クラス・リクエスト一覧

関数名	対応するリクエストと動作
usbf850_send_encapsulated_command	SendEncapsulatedCommand ：EP0からデータを受信する
usbf850_get_encapsulated_response	GetEncapsulatedRespons ：処理なし
usbf850_set_line_coding	SetLineCoding ：EP0でUART通信用設定データを受信 ：EP0NULLパケット送信処理実行
usbf850_get_line_coding	GetLineCoding ：UART通信用設定データをEP0から送信
usbf850_set_control_line_state	SetControlLineState ：EP0NULLパケット送信処理実行
usbf850_sstall_ctrl	：STALL制御処理

4.7.3 EP1 監視処理

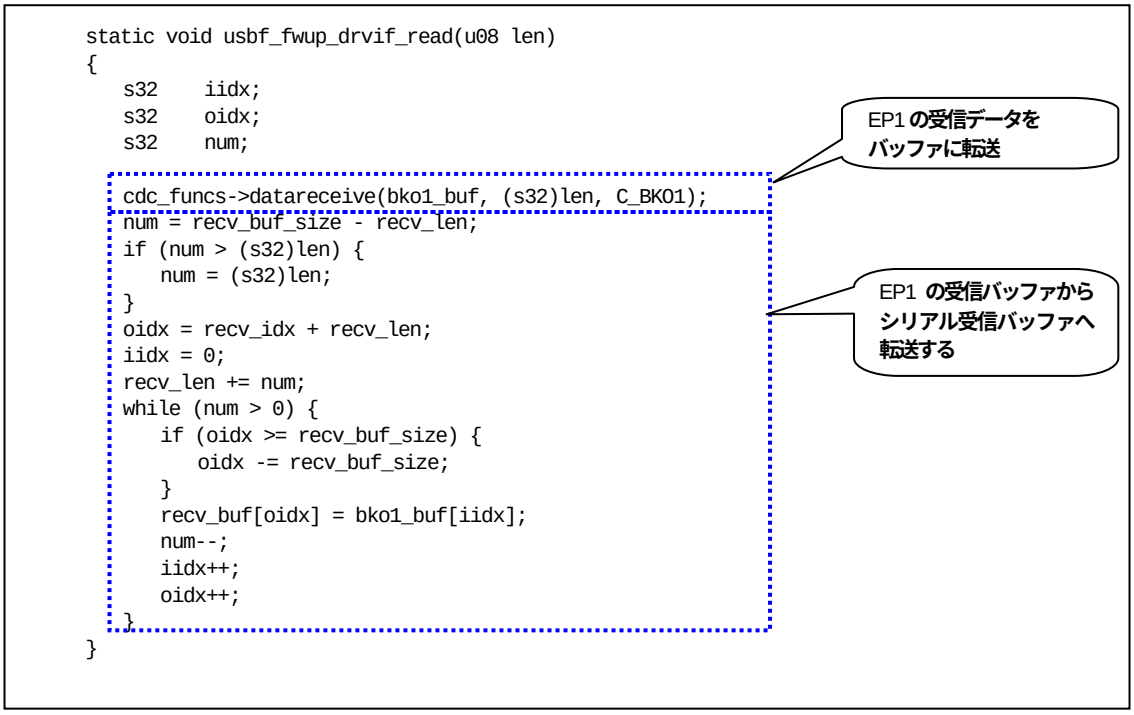
次に EP1 の監視処理を示します。




```
void usbf850_intusb1b(void)
{
    if (UF0IS3 & C_BKO1DT) {
        UF0IC3 = (UINT8)~C_BKO1DT; /*interrupt clear*/
        /* read to buffer */
        if (serial_funcs != (const struct usb_serial_funcs_st *)0) {
            serial_funcs->read(UF0BO1L);
        }
    }
}
```

図 4-33　EP1 監視処理

関数内の“serial_funcs->read(UF0BO1L);”は，初期設定により usbf_fwup_drvif_read 関数が呼ばれます。




```
static void usbf_fwup_drvif_read(u08 len)
{
    s32    iidx;
    s32    oidx;
    s32    num;

    cdc_funcs->datareceive(bko1_buf, (s32)len, C_BKO1);
    num = recv_buf_size - recv_len;
    if (num > (s32)len) {
        num = (s32)len;
    }
    oidx = recv_idx + recv_len;
    iidx = 0;
    recv_len += num;
    while (num > 0) {
        if (oidx >= recv_buf_size) {
            oidx -= recv_buf_size;
        }
        recv_buf[oidx] = bko1_buf[iidx];
        num--;
        iidx++;
        oidx++;
    }
}
```

図 4-34　EP1 受信処理

関数内の“cdc_funcs->datareceive(bko0_buf, IINT32)len, C_BKO1);”は，初期設定により usbf850_data_receive 関数が呼ばれます。

4.7.4　USBデータ送受信

次に，USB データ送受信処理，NULL パケット送信処理，STALL 制御処理を示します。

（1）USB データ送信処理

```
INT32 usbf850_data_send(UINT8* data, INT32 len, INT8 ep)
{
    INT32 i;
    UINT32 addr;
    INT32 dlen = len;

    INT8 dend;
    INT8 ep_status;
    INT8 max_packet_size;

    switch (ep) {
    case C_EP0: /*For the data stage*/
        addr = UF0E0W_ADDRESS;
        dend = C_E0DED;
        ep_status = C_EP0W;
        max_packet_size = C_MAXP0;
        break;
    case C_BKI1:
        addr = UF0BI1_ADDRESS;
        dend = C_BKI1DED;
        ep_status = C_BKIN1;
        max_packet_size = C_MAXP1;
        break;
    case C_BKI2:
        addr = UF0BI2_ADDRESS;
        dend = C_BKI2DED;
        ep_status = C_BKIN2;
        max_packet_size = C_MAXP3;
        break;
    case C_INT1:
        addr = UF0INT1_ADDRESS;
        dend = C_IT1DED;
        ep_status = C_IT1;
        max_packet_size = C_MAXP7;
        break;
    default: /*error*/
        return DEV_ERROR;
    }
```

EP0 用の書き込み用のレジスタ・アドレス、書き込み終了ビット，状態ビット，最大サイズをセット

EP1 用の書き込み用のレジスタ・アドレス，終了ビット，状態ビット，最大サイズをセット

EP3 用の書き込み用のレジスタ・アドレス，終了ビット，状態ビット，最大サイズをセット

EP7 用の書き込み用のレジスタ・アドレス，終了ビット，状態ビット，最大サイズをセット

図 4-35　データ送信処理（1/2）

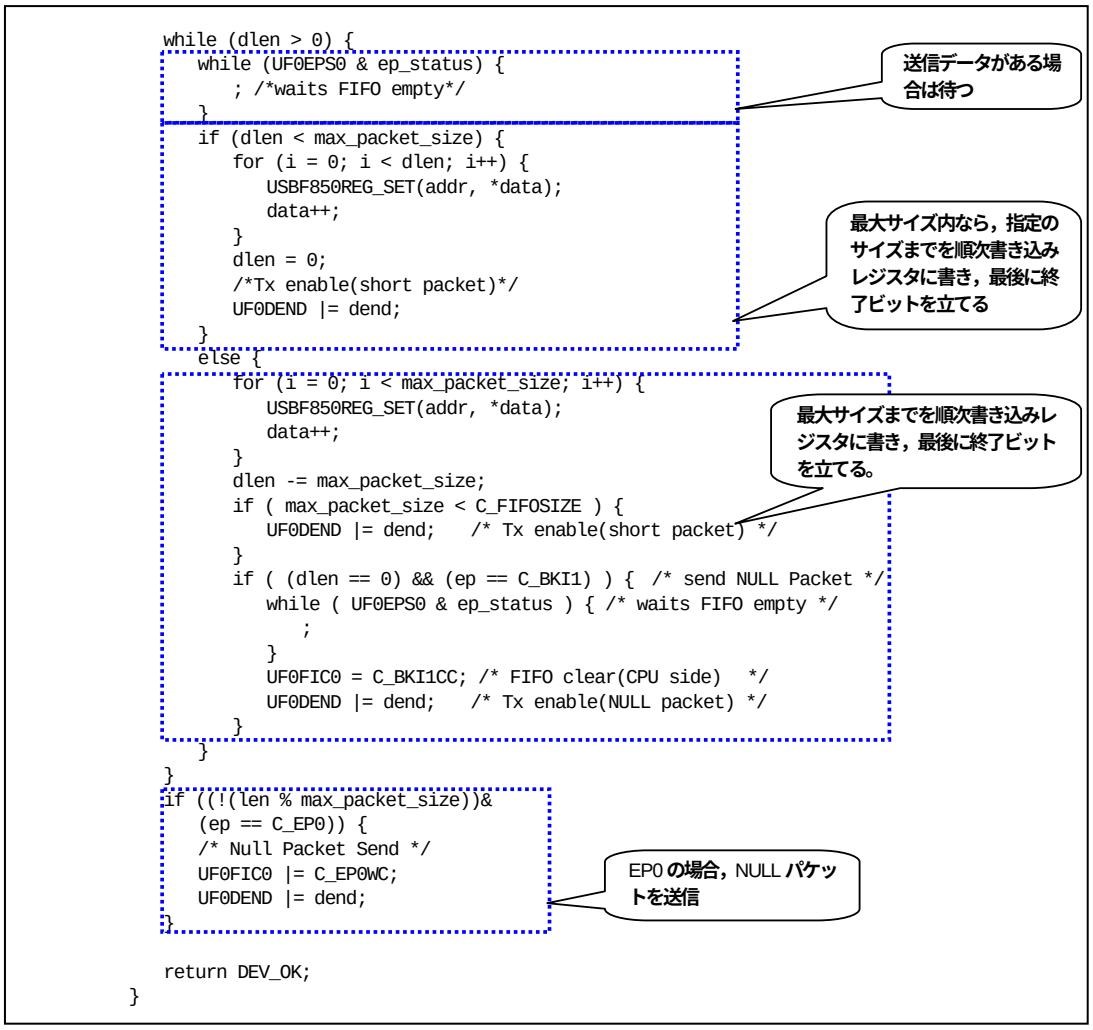


図 4-35　データ送信処理（2/2）

（2）USB データ受信処理

```
INT32 usbf850_data_receive(UINT8* data, INT32 len, INT8 ep)
{
    INT32 i = 0;
    INT32 j = 0;
    UINT32 addr;
    UINT32 len_addr;
    UINT8 size;
    INT8 ep_status;
    UINT8 tmp;

    switch (ep) {
    case C_EP0: /*For the data stage*/
        while ((UF0IS1 & C_E0ODT) == 0) {
            /*Control OUT interrupt wait*/
        }
        UF0IC1 = (UINT8)~C_E0ODT;
        size = UF0E0L;
        if (size != len) {      /*error*/
            UF0FIC0 = C_EP0RC; /*FIFO Clear*/
            usbf850_sendstallEP0();
            return DEV_ERROR;
        }
        for (i = 0; i < len; i++) {
            *data = UF0E0R;
            data++;
        }
        if (UF0EPS0 & C_EP0R) { /*Rx data reading completion*/
            /*error:begins to see in the rereading*/
            data -= len;
            len = UF0E0L;
            for (i = 0; i < len; i++) {
                *data = UF0E0R;
                data++;
            }
        }
        return DEV_OK;
    case C_BKO1:
        addr = UF0BO1_ADDRESS;
        len_addr = UF0BO1L_ADDRESS;
        ep_status = C_BKO1DT;
        break;
    case C_BKO2:
        addr = UF0BO2_ADDRESS;
        len_addr = UF0BO2L_ADDRESS;
        ep_status = C_BKO2DT;
        break;
    default: /*error*/
        return DEV_ERROR;
    }

    while (i < len) {
        size = USBF850REG_READ(len_addr);
        j += size;
        for ( ; i < j; i++) {
            if( i < len ){
                *data = USBF850REG_READ(addr);
                data++;
            }
            else{   /* read and thrown away. */
                tmp = USBF850REG_READ(addr);
            }
        }
        if ((len - j)>0) {
            while ( (UF0EPS0 & C_BKOUT1) == 0 ) {
               /*data wait*/
            }
            UF0IC3 = ~ep_status;
        }
    }
    return DEV_OK;
}
```

受信データがない場合待つ

レジスタに格納されているデータの長さが指定の長さと不一致の場合エラーとする

受信データをバッファに転送する

データがある場合は，再度，データの長さを取得し，バッファに転送する

EP0 でデータが残っている，または EP1 の場合のデータ取得処理

図 4-36　データ受信処理

（3）EP0NULL パケット送信処理

```
void usbf850_sendnullEP0(void)
{
  UF0FIC0 = C_EP0WC;  /*FIFO Clear*/
  UF0DEND |= C_E0DED; /*data send(Null Packet)*/
}
```

FIFO をクリアし，送信処理を行う

図 4-37　EP0NULL パケット送信

（4）STALL 制御処理

```
void usbf850_sendstallEP0(void)
{
  UF0SDS = C_SNDSTL; /*send STALL*/
}
```

STALL ハンドシェーク応答

図 4-38　STALL 制御処理

5. ファイル転送アプリケーションの解説

この章では，PC 上で動作するファイル転送アプリケーションについて記載します。

5.1 開発環境

ファイル転送アプリケーションは，次に示す環境で構築されています。

OS　　：Windows XP
開発言語：Microsoft Visual C++ 6.0（MFC）

5.2 動作概要

ファイル転送アプリケーションは，起動時に引数として，書き換え対象のファイル名（およびオプション）を受け取ると，直接書き換え処理に移行します。ファイルの指定がない場合は，設定画面を表示します。

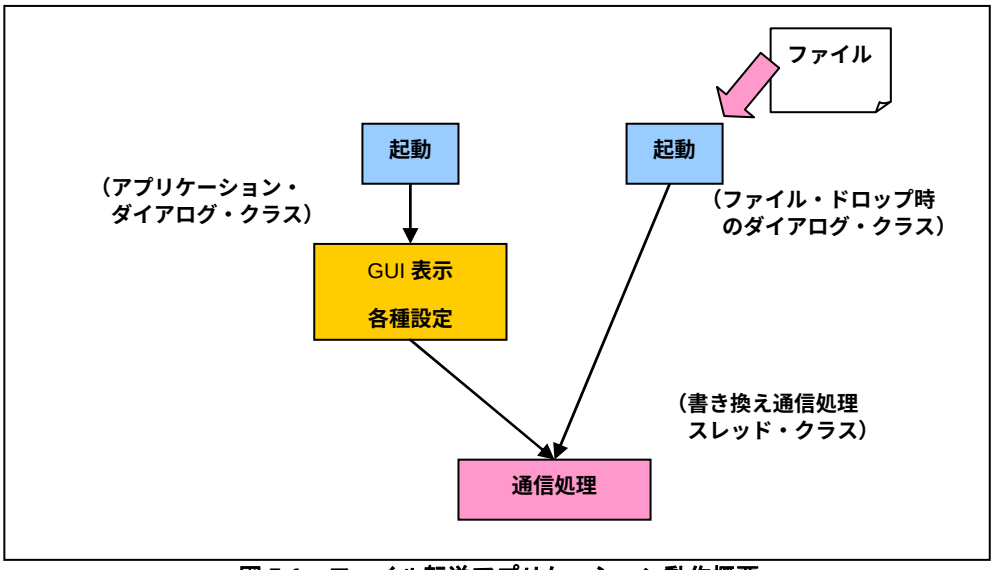


図 5-1　ファイル転送アプリケーション動作概要

5.3 ファイル構成

ファイル転送アプリケーションのファイル一覧を次に示します（主要なファイルのみを記載しています）。

表 5-1　ファイル転送アプリケーションのファイル一覧

ファイル名	説　明
FlashSelfRewriteGUI.dsw	ワークスペース・ファイル
FlashSelfRewriteGUI.dsp	プロジェクト・ファイル
FlashSelfRewriteGUI.clw	クラス・ウィザード用ファイル
FlashSelfRewriteGUI.rc	リソース・ファイル
FlashSelfRewriteGUI.cpp	アプリケーション・クラスの処理ファイル
FlashSelfRewriteGUI.h	アプリケーション・クラスの定義ファイル
FlashSelfRewriteGUIDlg.cpp	アプリケーションのダイアログ・クラスの処理ファイル
FlashSelfRewriteGUIDlg.h	アプリケーションのダイアログ・クラスの定義ファイル
FlashSelfRewriteGUIDrop.cpp	ファイル・ドロップ時のダイアログ・クラスの処理ファイル
FlashSelfRewriteGUIDrop.h	ファイル・ドロップ時のダイアログ・クラスの定義ファイル
CommandThread.cpp	書き換え通信処理スレッド・クラス処理ファイル
CommandThread.h	書き換え通信処理スレッド・クラス定義ファイル
CommonProc.cpp	共通処理クラス処理ファイル
CommonProc.h	共通処理クラス定義ファイル
SerialPort.cpp	シリアルCOMポート通信クラスの処理ファイル
SerialPort.h	シリアルCOMポート通信クラスの定義ファイル
Resource.h	リソース・ヘッダ・ファイル
UsbfUpdate.ini	アプリケーション動作設定ファイル

5.3.1 アプリケーション･クラス（FlashSelfRewriteGUI）

初回起動時に引数（オプション）をチェックし，ファイルが指定してあればファイル・ドロップ時のダイアログ・クラスを呼び，そうでなければ通常のダイアログ・クラスを呼び出します。
次にアプリケーションの起動オプションを示します。

表 5-2　アプリケーション起動オプション

オプション	説　明
/M [chip|address]	動作モードの指定 Chipかaddressを指定
/S nnnnnn	書き込み開始アドレスを16進数で指定
/C nn	接続COMポート番号の指定
filename	書き換え対象のファイル・パス

5.3.2 アプリケーション・ダイアログ・クラス（FlashSelfRewriteGUIDlg）

書き換え指定のダイアログ画面を表示します（第 2 章　USB ファンクション・ファームウエア・アップデートの実行参照）。この画面で動作モード，書き換えアドレス，書き換えファイル，接続 COM ポートを指定します。また，画面表示の際に，アプリケーション動作設定ファイルを読み込み，先の指定がしてあればデフォルト値として画面に反映されます。
“Update” ボタンをクリックすると，書き換え通信処理スレッド・クラスが呼び出されます。
追加したメンバ変数を示します。

表 5-3　アプリケーション・ダイアログ・クラスのメンバ変数の一覧

メンバ変数		説　明
型	メンバ名	
int	m_nCOM	**接続するCOMポート番号**
TCHAR	m_tcAppDir[_MAX_PATH]	**アプリケーションの実行ディレクトリ**
int	m_nCurTargetID	**現在のターゲットのID**
CString	m_strCurTarget	**現在のターゲット名**
CString	m_strCurDevice	**現在のデバイス**
CStringArray	m_arDeviceVal	**デバイスのリスト**
CStringArray	m_arDeviceText	**デバイス名のリスト**
int	m_nDevSize	**現在のデバイスのROMサイズ**
CWinThrread*	m_pCommandThread	**スレッド・クラスのポインタ**
BOOL	m_bExistThread	**スレッドの動作状態**
BOOL	m_bStartUp	**初回起動を表す**
CArray<int,int>	m_arBlockStart	**先頭ブロック番号の配列**
CArray<int,int>	m_arBlockEnd	**末尾ブロック番号の配列**
CArray<int,int>	m_arBlockUnit	**1ブロックごとのバイト数の配列**
COleDateTime	m_dtStart	**書き換え処理開始日付**
COleDateTime	m_dtEnd	**書き換え処理終了日付**

メンバ関数を次に示します。

表 5-4　Read_DeviceInfo 関数

関数名		Read_DeviceInfo
記述形式		BOOL Read_DeviceInfo (VOID)
機能		**アプリケーション動作設定ファイルから情報を取得**
入出力	**入力**	**なし**
	出力	TRUE(**成功**)／FALSE(**失敗**)

表 5-5　Write_DeviceInfo 関数

関数名		Write_DeviceInfo
記述形式		BOOL Write_DeviceInfo (VOID)
機能		**アプリケーション動作設定ファイルを更新する**
入出力	**入力**	**なし**
	出力	TRUE(**成功**)／FALSE(**失敗**)

表 5-6　Update_Message 関数

関数名		Update_Message
記述形式		VOID Update_Message (LPCTSTR)
機能		メッセージ表示欄にメッセージを表示する
入出力	入力	メッセージ文字列のポインタ
	出力	なし

表 5-7　Get_BlockAddress 関数

関数名		Get_BlockAddress
記述形式		DWORD Get_BlockAddress(int nBlk, EnBlockAddress opt)
機能		指定したブロック番号のメモリ上のアドレスを返す
入出力	入力	nBlk：ブロック番号 Opt：START／END(ブロックの先頭か末尾を現す)
	出力	メモリ上のアドレス

表 5-8　Get_AddressBlock 関数

関数名		Get_AddressBlock
記述形式		int Get_AddressBlock(DWORD dwAddress)
機能		指定したアドレスがあるブロック番号を返す
入出力	入力	dwAddress：メモリ上のアドレス
	出力	ブロック番号

表 5-9　Initialize_Device 関数

関数名		Initialize_Device
記述形式		VOID Initialize_Device(VOID)
機能		初期化処理
入出力	入力	なし
	出力	なし

表 5-10　AppStatus 関数

関数名		AppStatus
記述形式		VOID AppStatus(BOOL stu)
機能		書き換え動作時の状態を設定する
入出力	入力	stu：TRUE(画面操作を有効にする) FALSE(画面操作を無効にする)
	出力	なし

5.3.3 ファイル・ドロップ時のダイアログ・クラス（FlashSelfRewriteGUIDrop）

表示後，すぐに書き換え通信処理スレッド・クラスを呼び出し，書き換え処理を開始します。画面はプログレス・バーのみです。
追加したメンバ変数を示します（アプリケーション・ダイアログ・クラスと同じ部分は，省略しています）。

表 5-11　ファイル・ドロップ時のダイアログ・クラスのメンバ変数の一覧

メンバ変数		説　明
型	メンバ名	
CString	m_strFileName	対象とするファイル・パス
EnMode	m_enMode	書き換えモード
DWORD	m_dwStartAddress	書き換え開始アドレス

追加したメンバ関数を次に示します。

表 5-12　Execute 関数

関数名		Execute
記述形式		VOID Execute(VOID)
機能		書き換え処理を実行します
入出力	入力	なし
	出力	なし

5.3.4 書き換え通信処理スレッド・クラス（CommandThread）

シリアル COM ポート通信クラスを使用して，ターゲットとなる評価ボードに接続し，指定のファイルをインタフェース仕様に沿って送受信を行います。ファイルが HEX ファイルの場合は，その解析も行います。
追加したメンバ変数を示します（アプリケーション・ダイアログ・クラスと同じ部分は，省略しています）。

表 5-13　書き換え通信処理スレッド・クラスのメンバ変数一覧

メンバ変数		説　明
型	メンバ名	
CDialog*	m_pAppDlg	呼び出し元のダイアログ・クラスのポインタ
CString	m_strAppDir	アプリケーションのあるディレクトリ
BOOL*	m_pbExistThread	スレッドの動作状況のポインタ
CSerialPort	m_Serial	シリアルCOMポート通信クラス
int	m_nCOM	接続するCOMポート番号
CString	m_strFileName	対象とするファイル・パス
EnMode	m_enMode	書き換えモード
DWORD	m_dwStartAddress	書き換え開始アドレス
DWORD	m_dwROMStartAddress	ROMの先頭アドレス
DWORD	m_dwROMEndAddress	ROMの末尾アドレス

追加したメンバ関数を次に示します。

表 5-14　Cal_CheckSum 関数

関数名		Cal_CheckSum
記述形式		BYTE Cal_CheckSum(LPBYTE bytes, LONG size)
機能		**チェック・サムを算出します**
入出力	**入力**	bytes：**データ列のポインタ** size：**データ列の長さ**
	出力	**チェック・サム算出値**

表 5-15　Chage_strHex2Bibary 関数

関数名		Change_strHex2Binary
記述形式		VOID Change_strHex2Binary(LPCSTR strHex, LPBYTE pbytes, LONG size)
機能		**16進数であらわされた文字列をバイナリのデータ列に変換します**
入出力	**入力**	strHex：**16進数で表された文字列のポインタ** pbyte：**データ列の先頭ポインタ** size：**変換するデータの数**
	出力	**なし**

表 5-16　Upsets_DWORD 関数

関数名		Upsets_DWORD
記述形式		DWORD Upsets_DWORD(DWORD dwVal)
機能		DWORD**型の値をバイトごとに反転する** **(**ex.**)**　　0xaabbccdd → 0xddccbbaa
入出力	**入力**	dwVal：**反転する**DWORD**の値**
	出力	**反転された値**

表 5-17　SET_StartRecord 関数

関数名		SET_StartRecord
記述形式		VOID SET_StartRecord (LPVOID lpRecord)
機能		**書き換え開始レコードを作製する**
入出力	**入力**	lpRecord：**レコード格納ポインタ**
	出力	**なし**

表 5-18　SET_EndRecord 関数

関数名		SET_EndRecord
記述形式		VOID SET_EndRecord (LPVOID lpRecord)
機能		**書き換え終了レコードを作製する**
入出力	**入力**	lpRecord：**レコード格納ポインタ**
	出力	**なし**

5.3.5 共通処理クラス（CommonProc）

共通で使用される処理を定義しています。
追加したメンバ関数を次に示します。

表 5-19　GetAppDir 関数

関数名		GetAppDir
記述形式		static VOID GetAppDir(LPTSTR path, int sw = 0)
機能		**アプリケーションの実行アドレスを取得します。**
入出力	**入力**	path：**取得する文字列のポインタ** sw：0　**パスをそのまま取得** 1　**ショート・パスに変更したパスを取得**
	出力	**なし**

表 5-20　Change_Hex2Val 関数

関数名		Change_Hex2Val
記述形式		static DWORD Change_Hex2Val(LPCSTR pHex)
機能		**1バイト（16進数2桁）で表された文字列を数値に変換する**
入出力	**入力**	pHex：**16進数2桁で表された文字列のポインタ**
	出力	**変換された値**

表 5-21　IsNumeric 関数

関数名		IsNumeric
記述形式		static BOOL IsNumeric(LPCTSTR lpNum, LONG size, int type = 10)
機能		**数値チェック処理**
入出力	**入力**	lpNum：**数値を表した文字列のポインタ** size：**チェックする数値の桁数** type：10　**10進数としてチェックする** 16　**16進数としてチェックする**
	出力	TRUE(**数値であることを表す**)／FALSE(**数値ではないことを表す**)

表 5-22　IsExistFile 関数

関数名		IsExistFile
記述形式		static BOOL IsExistFile(LPCTSTR lpszFileName, BOOL bDirectory = FALSE)
機能		**ファイルの存在チェック**
入出力	**入力**	lpszFileName：**確認するファイル・パス** bDirectory：FALSE(**ファイルをチェックする**) TRUE(**ディレクトリをチェックする**)
	出力	TRUE(**存在する**)／FALSE(**存在しない**)

5.3.6 シリアルCOMポート通信クラス（SerialPort）

このクラスを使用して，COM ポートでのシリアル通信を行います。通信設定は固定で次のようになっています。

表 5-23　シリアル通信設定値

設定項目	値
ボー・レート	115200 bps
データ・サイズ	8ビット
パリティ	なし
ストップ・ビット	1ビット
スタート・ビット	LSB
フロー制御	なし

追加したメンバ変数を次に示します。

表 5-24　シリアル COM ポート通信クラスのメンバ変数一覧

メンバ変数		説　明
型	メンバ名	
HANDLE	m_hCom	接続時に取得するハンドル
DCB	m_Dcb	デバイス制御ブロック構造体
COMMTIMEOUTS	m_TimeoutSts	タイムアウト設定用の構造体
INT	m_nCOM	接続するポート番号

メンバ関数を次に示します。

表 5-25　Port_Open 関数

関数名		Port_Open
記述形式		LONG Port_Open(INT com)
機能		指定のCOMポートに接続します
入出力	入力	com：COMポート番号
	出力	0　接続成功 －1　接続失敗

表 5-26　Port_Close 関数

関数名		Port_Close
記述形式		VOID Port_Close(VOID)
機能		接続中のポートを切断します。
入出力	入力	なし
	出力	なし

表 5-27　Port_Write 関数

関数名		Port_Write
記述形式		LONG Port_Write(LPCVOID buf, LONG cnt)
機能		シリアル通信によるデータの送信を行う。
入出力	入力	buf：送信データの列のポインタ cnt：送信データの長さ（バイト）
	出力	送信したバイト数。−1で送信の失敗。

表 5-28　Port_Read 関数

関数名		Port_Read
記述形式		LONG Port_Read(LPVOID buf, LONG cnt)
機能		シリアル通信によるデータの受信を行う。
入出力	入力	buf：受信データを格納するデータ列のポインタ cnt：受信するデータの長さ（バイト）
	出力	受信したバイト数。−1で受信の失敗を表す。

表 5-29　Get_PortNumber 関数

関数名		Get_PortNumber
記述形式		INT Get_PortNumber(VOID)
機能		接続中のポート番号を取得
入出力	入力	なし
	出力	接続中のポート番号

表 5-30　AutoScanCom 関数

関数名		AutoScanCom
記述形式		INT AutoScanCom (LPCTSTR pszService, LPCTSTR pszInterface, INT nNo = 0)
機能		接続可能なCOMポートを検出
入出力	入力	pszService：COMポートが動作しているサービス名 pszInterface：インタフェース名 nNo：この番号以降を検索する
	出力	検出したCOMポート番号。見つからなかったら0が返る。

5.3.7 アプリケーション動作設定ファイル（UsbfUpdate.ini）

アプリケーション動作設定ファイルは ini ファイル形式で，設定値の保持，またはデバイスの情報を保持します。
このファイルは exe ファイルと同一のフォルダに置きます。
次に ini ファイル内の定義を示します。

表 5-31　アプリケーション動作設定ファイル説明（セクション）

セクション	説　明
Application	アプリケーションで設定中の値を表す
Tartget1	ターゲットIDを指す
Device.70F3769	デバイスの情報を表す 複数設定が可能

表 5-32　アプリケーション動作設定ファイル内項目一覧

セクション	キー	値	説　明
Application	Target	1～	指定中のターゲットのID番号
	COM	1～20	接続中または接続するCOMポート番号
	Mode	chip/address	指定中の動作モードを表す chip：ブート・スワップを使用したユーザ・プログラムの書き換え address：指定アドレスからの書き換え
	Address	FFFFFFFF	書き込む先頭のアドレス（16進数）
Tartget1	Name	XXX	このターゲットの名前を表す
	Device	XXX	このターゲットで指定しているデバイス
Device.70F3769	Target	1～	このデバイスが属するターゲットID
	Name	XXX	このデバイスの名前
	Size	999	このデバイスのROMサイズ
	Block0	XXX|XXX|XXX|XXX	ブロックの情報。"|"で区切る 先頭ブロック番号｜最終ブロック番号｜1ブロックのサイズ（Kバイト）｜ブート領域かどうか Block1～とすることで複数指定可能

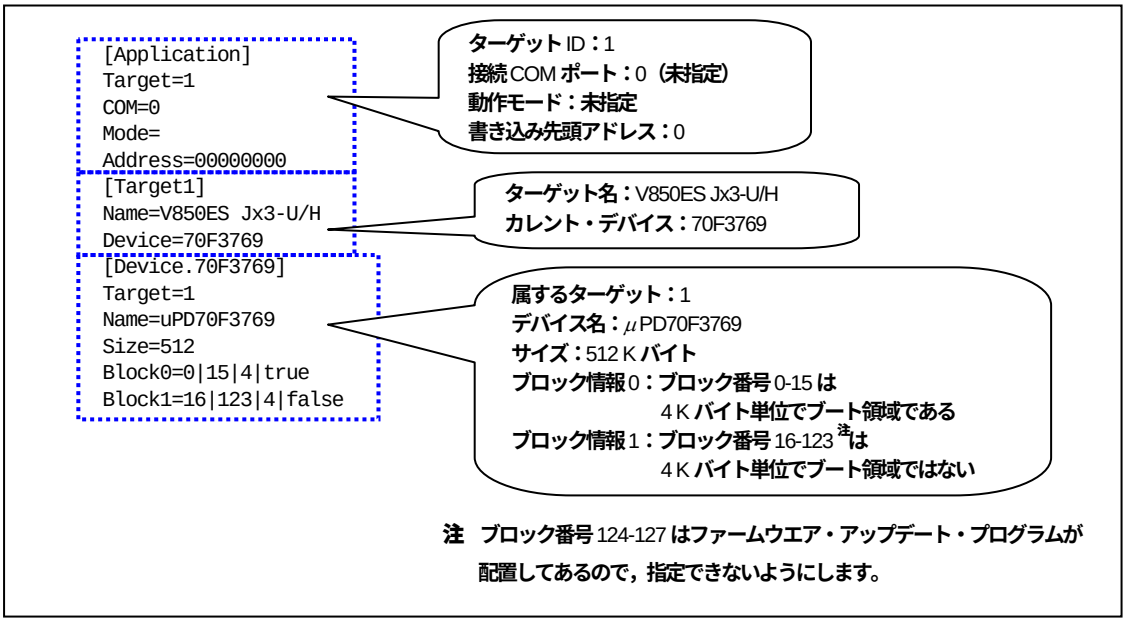


図 5-2　アプリケーション動作設定 ini ファイル

5.4　動作モード

動作モードの内容を次に示します。

（1）Chip

指定のファイルは，モトローラ S フォーマット，およびインテル拡張フォーマットの HEX ファイルでなければなりません。解析処理でそれ以外のフォーマットであればエラーとなります。書き込む先はメモリの先頭からなので，アドレスの指定は無視されます。

（2）Address

ファイルのイメージをそのまま転送し，指定のアドレスから書き込みます。

5.5　メッセージ表示

メッセージ表示欄に表示されるメッセージとその表示タイミングの一覧を次に示します。

表 5-33　表示メッセージ一覧

	メッセージ	表示タイミング
1	書き換え処理を開始します	書き換え処理開始時
2	正常終了しました	書き換え処理正常終了時
3	ファイルを指定してください	書き換え処理時に指定のファイルが指定されていない。 または，指定ファイルが存在しない
4	モードを指定してください	書き換え処理時にモードが設定されていない
5	アドレスを正しく指定してください	Addressモードで書き換え処理時に，アドレスが正常に指定されていない。
6	COMポートを指定してください	COMポートが正しく指定されていない
7	ERR：ファイルオープンエラー	ファイルのオープンに失敗した
8	ERR：ファイルフォーマットエラー	Mode=Chipのときに，モトローラSフォーマット，インテル拡張フォーマット以外のファイルを指定した
9	ERR：COMポートnに接続できません。	COMポートnの接続に失敗した
10	ERR：データ送信エラー	データの送信に失敗した
11	ERR：データ受信エラー	データの受信に失敗した（リトライ3回も同様）
12	ERR：書き換え処理停止	評価ボード側からNAK（エラー）を受信した
13	ERR：ファイルサイズエラー	ファイル・サイズ・チェックの際にデータのサイズがROM領域をはみ出てしまう

6. ユーザ・プログラムの作成

この章では，ユーザ・プログラムを作成するうえで留意すべき事項について説明します。

6.1 PM+設定（HEXファイルのフォーマット設定）

USB ファームウエア・アップデート・プログラム（ファイル転送アプリケーション）は，ユーザ・プログラムを書き換えるとき，モトローラタイプ S（32 ビット），モトローラタイプ S（スタンダード），インテル拡張フォーマットの HEX ファイルしか扱いません。＜ヘキサコンバータオプションの設定＞ダイアログの《オプション》タブからフォーマットを指定してください。

例　《フォーマット（F）》に“モトローラタイプ S（32 ビット）[-fs]”を選択します。

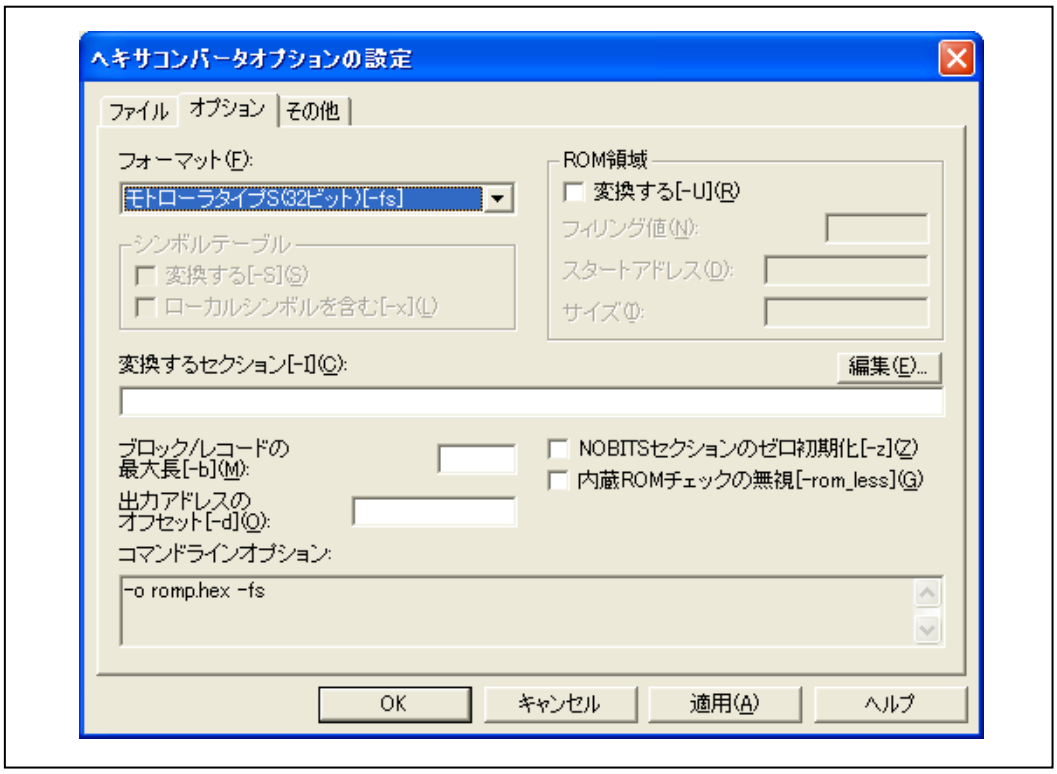


図 6-1　HEX ファイル・フォーマット指定例

6.2 ブート処理（リセット・ベクタ・セクション）

セルフ・アップデート・プログラムは，リセット時のベクタ処理をメモリの先頭（00000000 番地から）を前提に動作するため，作成するユーザ・プログラムも，リセットのセクションはメモリの先頭にしてください。

6.3 リンク・ディレクティブ（ユーザ・プログラムの配置制限）

ユーザ・プログラムは，ファームウェアアップデートプログラムの位置には配置できません。そのため，リンク・ディレクティブ・ファイルに記述するセグメントには，セルフ・アップデート・プログラムに配置されないようにアドレスを指定する必要があります（4. 2　メモリ・マップ参照）。
各製品でのファームウェアアップデートプログラムの位置は次のとおりです。

表 6-1　ファームウエア・アップデート・プログラム配置アドレス

対象CPU	配置アドレス
V850ES/JG3-U, V850ES/JH3-U	0007C000H
V850ES/JG3-H, V850ES/JH3-H	0003C000H
V850ES/JH3-E, V850ES/JJ3-E	0007C000H
V850ES/JG3-L	0007C000H

7. カスタマイズ

この章では，USB ファンクション・ファームウエア・アップデート・プログラムを他の環境へ移植する方法を説明します。

7.1 TK-850/JG3-Hへの移植

TK-850/JH3U-SP ボードより TK-850/JG3H ボードへの移殖の際に変更を行った箇所を説明します。TK-850/JG3H ボードで使用されている CPU（uPD70F3760）のメモリ容量は次のようになります。

・内蔵フラッシュ・メモリ：256 K バイト（ブロック 0-63）
・内蔵 RAM　　：32 K バイト

変更が必要なファイルを次に示します。

・firm_update.dir
・usbf_fwup_mem_def_usr.h
・usbf_fwup_pwonchk_usr.c
・UsbfUpdate.ini

7.1.1 セルフ・アップデート・プログラムの変更

firm_update ディレクトリ以下にあるファイルを移植環境に合わせてカスタマイズを行い，ファームウエア・アップデート・プログラムを変更します。

表 7-1　ファームウエア・アップデート・プログラムのカスタマイズ・ファイル

ファイル名	説　明
firm_update.dir	リンク・ディレクティブ・ファイル
include¥usbf_fwup_mem_def_usr.h	フラッシュ・メモリ環境の定義
src¥usbf_fwup_pwonchk_usr.c	実行プログラムの選択処理

（1）firm_update.dir

TK-850/JG3H ボードに使用されている CPU（uPD70F3760）に合わせて, セグメントの配置アドレスを変更します。ファームウエア・アップデート・プログラムはフラッシュ・メモリの最後に配置する必要があり，4 ブロック（16 K バイト）使用します。CONST, TEXT セグメントは合わせて 3 ブロック，APSTART セグメントは 1 ブロック使用します。FLASHTEXT, DATA セグメントは内蔵 RAM の先頭から配置します。

```
CONST   : !LOAD ?R:V0x3c000 {
        .const                  = $PROGBITS   ?A .const;
};

TEXT    : !LOAD ?RX {
        SelfLib_Rom.text        = $PROGBITS   ?AX SelfLib_Rom.text;
        .text                   = $PROGBITS   ?AX .text;
};

APSTART : !LOAD ?RX:V0x3f000 {
        apstart.text            = $PROGBITS   ?AX apstart.text;
};

FLASHTEXT: !LOAD ?RX:V0x3ff7000 {
        SelfLib_ToRamUsrInt.text = $PROGBITS  ?AX SelfLib_ToRamUsrInt.text;
        SelfLib_ToRamUsr.text    = $PROGBITS  ?AX SelfLib_ToRamUsr.text;
        SelfLib_RomOrRam.text    = $PROGBITS  ?AX SelfLib_RomOrRam.text;
        SelfLib_ToRam.text       = $PROGBITS  ?AX SelfLib_ToRam.text;
        flash.text               = $PROGBITS  ?AX flash.text;
};

DATA    : !LOAD ?RW {
        .data                   = $PROGBITS   ?AW  .data;
        .sdata                  = $PROGBITS   AWG .sdata;
        .sbss                   = $NOBITS     ?AWG .sbss;
        .bss                    = $NOBITS     ?AW  .bss;
        SelfLib_RAM.bss         = $NOBITS     ?AW  SelfLib_RAM.bss;
};

__tp_TEXT @ %TP_SYMBOL;
__gp_DATA @ %GP_SYMBOL &__tp_TEXT{DATA};
__ep_DATA @ %EP_SYMBOL;
```

ブロック 60 の開始アドレスを記述します

ブロック 63 の開始アドレスを記述します

内蔵 RAM の開始アドレスを記述します

図 7-1　flash_update.dir

（2）usbf_fwup_mem_def_usr.h

ファームウエア・アップデート・プログラムで使用するフラッシュ・メモリ環境を定義します。

APSTART_ADDR は，リンク・ディレクティブ・ファイルで指定した APSTART セグメントの配置アドレスを記述します。WRITE_MAX_BLOCK は，ユーザ・プログラムが使用できる最終ブロック番号を記述します。ファームウエア・アップデート・プログラムは，ブロック 60-63 を使用するため，ユーザ・プログラムが使用できるブロックは，ブロック 0-59 となります。

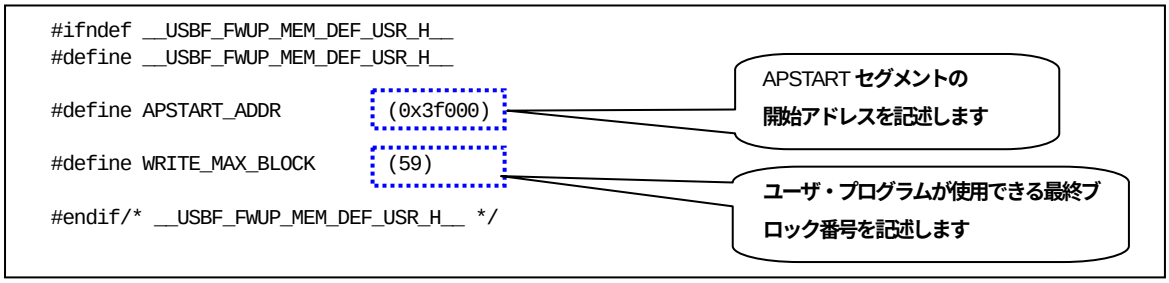


図 7-2　usbf_fwup_mem_def_usr.h

（3）usbf_fwup_pwonchk_usr.c

電源投入時，またはリセット時に，ファームウエア・アップデート・プログラム，またはユーザ・プログラムを実行するための判定処理を記述します。

TK-850/JG3H ボード上にある SW3, SW4 は，TK-850/JH3U-SP ボードと同じ構成であるため，変更の必要はありません。

```
#pragma ioreg

#include "nec_types.h"

#define SW_PUSHED   (0x00)     /* pushed switch SW3 and SW4 */
#define SW_STATUS   (0x03)     /* switch status SW3 and SW4 */

s32 usbf_fwup_pwonchk_usr(void);

s32 usbf_fwup_pwonchk_usr(void)
{
   s32    ret = -1;
   u08    sts;

   sts = P9H;
   if ((sts & SW_STATUS) == SW_PUSHED) {
      ret = 0;
   }

   return ret;
}
```

図 7-3　usbf_fwup_pwonchk_usr.c

7.1.2　ファイル転送アプリケーションiniファイルの変更

FirmupdateGUI ディレクトリ以下にある UsbfUpdateini ファイルを移植環境に合わせてカスタマイズします。

表 7-2　ファイル転送アプリケーションのカスタマイズ・ファイル

ファイル名	説　明
UsbfUpdate.ini	ファイル転送アプリケーションの設定

（1）UsbfUpdate.ini

uPD70F3760 の設定を追加します。

```
[Application]
Target=1
COM=8
Mode=chip
Address=00000000
[Target1]
Name=V850ES Jx3-U/H
Device=70F3760
[Device.70F3769]
Target=1
Name=uPD70F3769
Size=512
Block0=0|15|4|true
Block1=16|123|4|false
[Device.70F3760]
Target=1
Name=uPD70F3760
Size=256
Block0=0|15|4|true
Block1=16|59|4|false
```

属するターゲット:1
デバイス名:μPD70F3760
サイズ:256 K バイト
ブロック情報 0:ブロック番号 0-15 は
4 K バイト単位でブート領域である
ブロック情報 1:ブロック番号 16-59 は
4 K バイト単位でブート領域ではない

図 7-4　UsbUpdate.ini

7.2 TK-850/JH3E+NETへの移植

TK-850/JH3U-SP ボードより TK-850/JH3E+NET ボードへの移植の際に変更を行った箇所を説明します。
TK-850/JH3E+NET ボードで使用されている CPU（uPD70F3783）のメモリ容量は次のようになります。

・内蔵フラッシュ・メモリ：512 K バイト（ブロック 0-127）
・内蔵 RAM　　：60 K バイト

変更が必要なファイルを次に示します。

・firm_update.dir
・usbf_fwup_drvif.c
・usbf_fwup_execram.c
・usbf_fwup_pwonchk_usr.c
・UsbfUpdate.ini

7.2.1 セルフ・アップデート・プログラムの変更

firm_update ディレクトリ以下にあるファイルを移植環境に合わせてカスタマイズを行い，ファームウエア・アップデート・プログラムを変更します。

表 7-3　ファームウエア・アップデート・プログラムのカスタマイズ・ファイル

ファイル名	説　明
firm_update.dir	**リンク・ディレクティブ・ファイル**
src¥usbf_fwup_drvif.c	CDC**ドライバとのインタフェース**
src¥usbf_fwup_execram.c	**フラッシュ・メモリ書き込み処理**
src¥usbf_fwup_pwonchk_usr.c	**実行プログラムの選択処理**

（1）firm_update.dir

TK-850/JH3E+NET ボードに使用されている CPU（uPD70F3783）に合わせて，セグメントの配置アドレスを変更します。

ファームウエア・アップデート・プログラムはフラッシュ・メモリの最後に配置する必要があり，4 ブロック（16 K バイト）使用します。CONST, TEXT セグメントは合わせて 3 ブロック，APSTART セグメントは 1 ブロック使用します。FLASHTEXT, DATA セグメントは内蔵 RAM の先頭から配置します。

```
CONST   : !LOAD ?R V0x7c000 {
        .const                  = $PROGBITS   ?A .const;
};

TEXT    : !LOAD ?RX {
        SelfLib_Rom.text        = $PROGBITS   ?AX SelfLib_Rom.text;
        .text                   = $PROGBITS   ?AX .text;
};

APSTART : !LOAD ?RX V0x7f000 {
        apstart.text            = $PROGBITS   ?AX apstart.text;
};

FLASHTEXT: !LOAD ?RX V0x3ff0000 {
        SelfLib_ToRamUsrInt.text = $PROGBITS  ?AX SelfLib_ToRamUsrInt.text;
        SelfLib_ToRamUsr.text    = $PROGBITS  ?AX SelfLib_ToRamUsr.text;
        SelfLib_RomOrRam.text    = $PROGBITS  ?AX SelfLib_RomOrRam.text;
        SelfLib_ToRam.text       = $PROGBITS  ?AX SelfLib_ToRam.text;
        flash.text               = $PROGBITS  ?AX flash.text;
};

DATA    : !LOAD ?RW {
        .data                   = $PROGBITS   ?AW  .data;
        .sdata                  = $PROGBITS   AWG .sdata;
        .sbss                   = $NOBITS     ?AWG .sbss;
        .bss                    = $NOBITS     ?AW  .bss;
        SelfLib_RAM.bss         = $NOBITS     ?AW  SelfLib_RAM.bss;
};

__tp_TEXT @ %TP_SYMBOL;
__gp_DATA @ %GP_SYMBOL &__tp_TEXT{DATA};
__ep_DATA @ %EP_SYMBOL;
```

ブロック 124 の開始アドレスを記述します

ブロック 127 の開始アドレスを記述します

内蔵 RAM の開始アドレスを記述します

図 7-5　flash_update.dir

（2）usbf_fwup_drvif.c

CDC ドライバとのインタフェースとなる処理を記述します。

TK-850/JH3E+NET では不要なインクルードファイルの指定があるので，これを削除します。

```
/*-------------------------------------------------------------------------*/
/* File    : usbf_fwup_drvif.c                                             */
/* Content : USB driver interface to application.                          */
/*                                                                         */
/* Copyright(c) 2009 Renesas Electronics Corporation                       */
/*-------------------------------------------------------------------------*/

#include "types.h"
#include "usbf850_sfr_jx3u.h"
#include "usbf850_jx3u.h"
#include "usbf850_drvif.h"
#include "usbf_fwup_drvif.h"

void       usbf_fwup_drvif_init(u08 *buf, s32 buf_len);
void       usbf_fwup_drvif_clear_buffer(void);
s32        usbf_fwup_drvif_recv(u08 *data, s32 len);
void       usbf_fwup_drvif_send(u08 *data, s32 len);
static void usbf_fwup_drvif_read(u08 len);
       .
       .
       .
```

不要なので削除します。

図 7-6　usbf_fwup_drvif.c

（3）usbf_fwup_execram.c

フラッシュ・メモリ書き込みを行う処理を記述します。セルフ・ライブラリを用いる際に必要な FLMD0 端子の操作を変更します。

TK-850/JH3E+NET では TK-850/JH3U-SP と操作するポートが異なるため，以下の場所を変更します。

```
u32 usbf_fwup_from_write(struct flash_data_st *data, u08 flag)
{
    s32    *out;
    u32    ret;
    u32    info;
    u16    mask[6];

          mask[0] = IMR0;
          mask[1] = IMR1;
          mask[2] = IMR2;
          mask[3] = IMR3;
          mask[4] = IMR4;
          mask[5] = IMR5;
          IMR0 = 0xffff;
          IMR1 = 0xffff;
          IMR2 = 0xffff;
          IMR3 = 0xffff;
          IMR4 = 0xffff;
          IMR5 = 0xffff;

          /* FLMD0 to High */
          PM0.2 = 0;
          P0.2 = 1;

          /* Flash environment initialization */
          FlashEnv((u32)1);

          /* Status check of terminal FLMD */
          ret = FlashFLMDCheck();
          if (ret != SELFLIB_OK) {
                    ret |= 0x00010000;
                    goto end;
          }

          /* Get output address */
    out = (s32 *)(data->block * FLASH_BLOCK_SIZE);

          /* Delete block */
          ret = FlashBlockErase(data->block, data->block);
          if (ret != SELFLIB_OK) {
                    ret |= 0x00020000;
                    goto end;
          }
          do {
                    ret = FlashStatusCheck();
          } while (ret == SELFLIB_BUSY);
          if (ret != SELFLIB_OK) {
                    ret |= 0x00030000;
                    goto end;
          }

          /* Flash writing */
          info = (data->data_length + 7) / 8;
          info *= 2;
          ret = FlashWordWrite(out, data->data, info);
          if (ret != SELFLIB_OK) {
                    ret |= 0x00040000;
                    goto end;
          }

          /* Internal verify */
          ret = FlashBlockIVerify(data->block, data->block);
          if (ret != SELFLIB_OK) {
                    ret |= 0x00050000;
                    goto end;
          }
          do {
                    ret = FlashStatusCheck();
          } while (ret == SELFLIB_BUSY);
          if (ret != SELFLIB_OK) {
                    ret |= 0x00060000;
                    goto end;
          }

          /* Specification boot swap */
          if (flag & BOOT_FLAG_SETINFO) {
                    info = 0x1f00003e;
                    ret = FlashGetInfo((u32)4);
                    ret &= 0x00000001;
                    info |= ret;
                    ret = FlashSetInfo(info, (u32)0);
                    if (ret != SELFLIB_OK) {
                              ret |= 0x00070000;
                              goto end;
                    }
                    do {
                              ret = FlashStatusCheck();
                    } while (ret == SELFLIB_BUSY);
                    if (ret != SELFLIB_OK) {
                              ret |= 0x00080000;
                              goto end;
                    }
          }

          /* Execution boot swap */
          if (flag & BOOT_FLAG_BOOTSWAP) {
                    ret = FlashBootSwap();
                    if (ret != SELFLIB_OK) {
                              ret |= 0x00090000;
                              goto end;
                    }
          }

          ret = 0;

end:
          /* Flash environment end */
          FlashEnv((u32)0);

          /* FLMD0 to Low */
          P0.2 = 0;
          PM0.2 = 1;

          IMR0 = mask[0];
          IMR1 = mask[1];
          IMR2 = mask[2];
          IMR3 = mask[3];
          IMR4 = mask[4];
          IMR5 = mask[5];

          return ret;
}
```

FLMD0 制御ポートを変更します。

FLMD0 制御ポートを変更します。

図 7-7　usbf_fwup_execram.c

（4）usbf_fwup_pwonchk_usr.c

電源投入時，またはリセット時に，ファームウエア・アップデート・プログラム，またはユーザ・プログラムを実行するための判定処理を記述します。

TK-850/JH3E+NET ボード上にある SW3, SW4 は，TK-850/JH3U-SP ボードとフラグ位置が異なるため，以下の場所を変更します。

```
#pragma ioreg

#include "types.h"

#define SW_PUSHED   (0x00)      /* pushed switch SW3 and SW4 */
#define SW_STATUS   (0x11)      /* switch status SW3 and SW4 */

s32 usbf_fwup_pwonchk_usr(void);

s32 usbf_fwup_pwonchk_usr(void)
{
   s32    ret = -1;
   u08    sts;

   sts = P9H;
   if ((sts & SW_STATUS) == SW_PUSHED) {
      ret = 0;
   }

   return ret;
}
```

SW3, SW4 のビット位置を合わせます

図 7-8　usbf_fwup_pwonchk_usr.c

7.2.2 ファイル転送アプリケーションiniファイルの変更

FirmupdateGUI ディレクトリ以下にある UsbfUpdateini ファイルを移植環境に合わせてカスタマイズします。

表 7-4　ファイル転送アプリケーションのカスタマイズ・ファイル

ファイル名	説　明
UsbfUpdate.ini	ファイル転送アプリケーションの設定

（1）UsbfUpdate.ini

uPD70F3783 の設定を追加します。

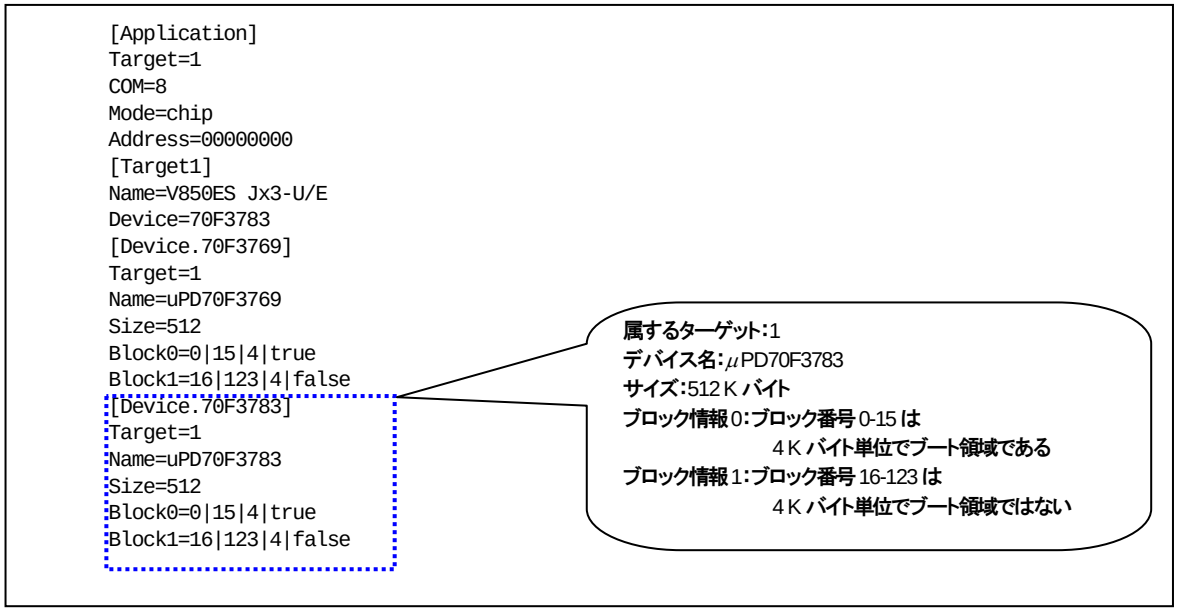


図 7-9　UsbUpdate.ini

7.3 TK-850/JG3+USBへの移植

TK-850/JH3U-SP ボードより TK-850/JG3L+USB ボードへの移植の際に変更を行った箇所を説明します。
TK-850/JG3L+USB ボードで使用されている CPU（uPD70F3796）のメモリ容量は次のようになります。

・内蔵フラッシュ・メモリ：512 K バイト（ブロック 0-127）
・内蔵 RAM　　：40 K バイト

変更が必要なファイルを次に示します。

・firm_update.dir
・usbf_fwup.c
・usbf_fwup_drvif.c
・usbf_fwup_execram.c
・usbf_fwup_pwonchk_usr.c
・UsbfUpdate.ini

7.3.1 セルフ・アップデート・プログラムの変更

firm_update ディレクトリ以下にあるファイルを移植環境に合わせてカスタマイズを行い，ファームウエア・アップデート・プログラムを変更します。

表 7-5　ファームウエア・アップデート・プログラムのカスタマイズ・ファイル

ファイル名	説　　明
firm_update.dir	リンク・ディレクティブ・ファイル
src¥usbf_fwup.c	セルフ・アップデートを行うソース・ファイル
src¥usbf_fwup_drvif.c	CDCドライバとのインタフェース
src¥usbf_fwup_execram.c	フラッシュ・メモリ書き込み処理
src¥usbf_fwup_pwonchk_usr.c	実行プログラムの選択処理

（1）firm_update.dir

TK-850/JG3L+USB ボードに使用されている CPU（uPD70F3796）に合わせて，セグメントの配置アドレスを変更します。

ファームウエア・アップデート・プログラムはフラッシュ・メモリの最後に配置する必要があり，4 ブロック（16 K バイト）使用します。CONST, TEXT セグメントは合わせて 3 ブロック，APSTART セグメントは 1 ブロック使用します。FLASHTEXT, DATA セグメントは内蔵 RAM の先頭から配置します。

```
CONST   : !LOAD ?R:V0x7c000 {
        .const                  = $PROGBITS   ?A .const;
};

TEXT    : !LOAD ?RX {
        SelfLib_Rom.text        = $PROGBITS   ?AX SelfLib_Rom.text;
        .text                   = $PROGBITS   ?AX .text;
};

APSTART : !LOAD ?RX:V0x7f000 {
        apstart.text            = $PROGBITS   ?AX apstart.text;
};

FLASHTEXT: !LOAD ?RX:V0x3ff5000 {
        SelfLib_ToRamUsrInt.text = $PROGBITS  ?AX SelfLib_ToRamUsrInt.text;
        SelfLib_ToRamUsr.text    = $PROGBITS  ?AX SelfLib_ToRamUsr.text;
        SelfLib_RomOrRam.text    = $PROGBITS  ?AX SelfLib_RomOrRam.text;
        SelfLib_ToRam.text       = $PROGBITS  ?AX SelfLib_ToRam.text;
        flash.text               = $PROGBITS  ?AX flash.text;
};

DATA    : !LOAD ?RW {
        .data                   = $PROGBITS   ?AW  .data;
        .sdata                  = $PROGBITS   AWG .sdata;
        .sbss                   = $NOBITS     ?AWG .sbss;
        .bss                    = $NOBITS     ?AW  .bss;
        SelfLib_RAM.bss         = $NOBITS     ?AW  SelfLib_RAM.bss;
};

__tp_TEXT @ %TP_SYMBOL;
__gp_DATA @ %GP_SYMBOL &__tp_TEXT{DATA};
__ep_DATA @ %EP_SYMBOL;
```

ブロック 124 の開始アドレスを記述します

ブロック 127 の開始アドレスを記述します

内蔵 RAM の開始アドレスを記述します

図 7-10　flash_update.dir

（2）usbf_fwup.c

TK-850/JG3L+USB ボードに使用されている CPU（uPD70F3796）に合わせて，ブート・スワップ領域への書き込み処理を変更します。

TK-850/JG3L+USB ボードに使用されている CPU（uPD70F3796）では，ブート・スワップ領域がブロック番号 0-7，ブロック 8-15 となっているため，以下の場所を変更します。

```
void usbf_fwup(void)
{
  .
  .
  .
   if (flash_block < 8) {
      first_addr = flash_addr;
      flash_buf.block = flash_block + 8;
      flash_block = usbf_fwup_recv_block();
      if (first_addr == 0) {
         usbf_fwup_replace_apstart();
      }
      while (flash_block < 8) {
         __DI();
         ret = (s32)usbf_fwup_from_write(&flash_bu    );
         __EI();
         if (ret != 0) {
            code = ERROR_FLASH_WRITE;
            goto error;
         }
         flash_buf.block = flash_block + 8;
         flash_block = usbf_fwup_recv_block();
      }
      __DI();
      ret = (s32)usbf_fwup_from_write(&flash_buf, BOOT_FLAG_SETINFO);
      __EI();
      if (ret != 0) {
         code = ERROR_FLASH_WRITE;
         goto error;
      }
      while (flash_block < 16) {
         flash_buf.block = flash_block - 8;
         flash_block = usbf_fwup_recv_block();
         __DI();
         ret = (s32)usbf_fwup_from_write(&flash_buf, 0);
         __EI();
         if (ret != 0) {
            code = ERROR_FLASH_WRITE;
            goto error;
         }
      }
      if (first_addr == 0) {
         ret = usbf_fwup_write_apstart();
         if (ret != 0) {
            code = ERROR_FLASH_WRITE;
            goto error;
         }
      }
   }
  .
  .
  .
```

ブート領域のブロック数を変更します

スワップ領域のブロック数を変更します

図 7-11　usbf_fwup.c

（3）usbf_fwup_drvif.c

CDC ドライバとのインタフェースとなる処理を記述します。

TK-850/JG3L+USB では不要なインクルードファイルの指定があるので，これを削除します。

```
/*-----------------------------------------------------------------------------*/
/*  File    : usbf_fwup_drvif.c                                                */
/*  Content : USB driver interface to application.                             */
/*                                                                             */
/*  Copyright(c) 2009 Renesas Electronics Corporation                          */
/*-----------------------------------------------------------------------------*/

#include "types.h"
#include "usbf850_sfr_jx3u.h"
#include "usbf850_jx3u.h"
#include "usbf850_drvif.h"
#include "usbf_fwup_drvif.h"

void       usbf_fwup_drvif_init(u08 *buf, s32 buf_len);
void       usbf_fwup_drvif_clear_buffer(void);
s32        usbf_fwup_drvif_recv(u08 *data, s32 len);
void       usbf_fwup_drvif_send(u08 *data, s32 len);
static void usbf_fwup_drvif_read(u08 len);
       .
       .
       .
```

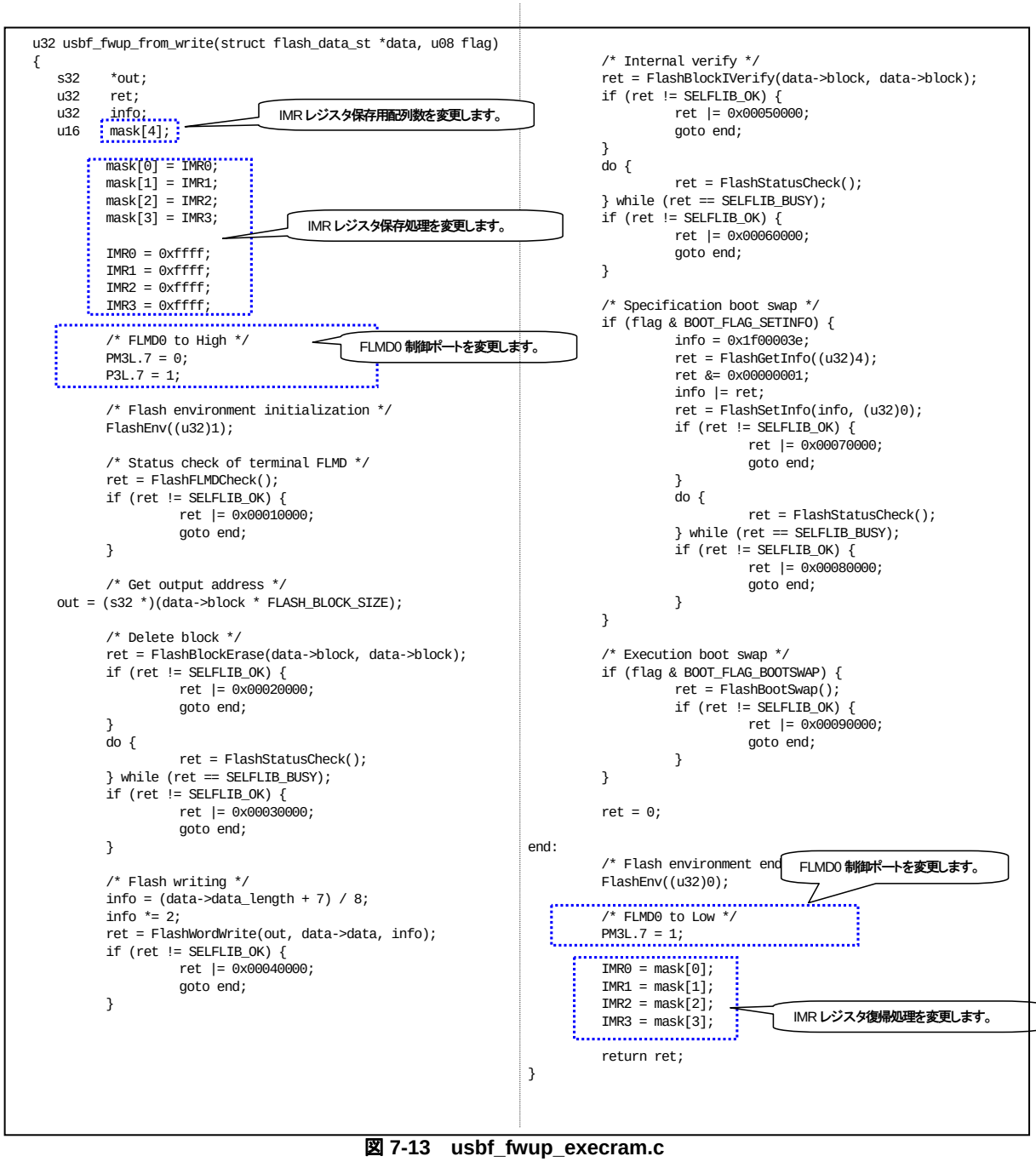
不要なので削除します。

図 7-12　usbf_fwup_drvif.c

（4）usbf_fwup_execram.c

フラッシュ・メモリ書き込みを行う処理を記述します。セルフ・ライブラリを用いる際に必要な FLMD0 端子の操作および割り込みマスク・レジスタ操作を変更します。

TK-850/JH3E+NET では TK-850/JH3U-SP と操作する割り込みマスク・レジスタ及び FLMD0 ポートが異なるため，以下の場所を変更します。




```
u32 usbf_fwup_from_write(struct flash_data_st *data, u08 flag)
{
    s32    *out;
    u32    ret;
    u32    info;
    u16    mask[4];

        mask[0] = IMR0;
        mask[1] = IMR1;
        mask[2] = IMR2;
        mask[3] = IMR3;

        IMR0 = 0xffff;
        IMR1 = 0xffff;
        IMR2 = 0xffff;
        IMR3 = 0xffff;

        /* FLMD0 to High */
        PM3L.7 = 0;
        P3L.7 = 1;

        /* Flash environment initialization */
        FlashEnv((u32)1);

        /* Status check of terminal FLMD */
        ret = FlashFLMDCheck();
        if (ret != SELFLIB_OK) {
                ret |= 0x00010000;
                goto end;
        }

        /* Get output address */
    out = (s32 *)(data->block * FLASH_BLOCK_SIZE);

        /* Delete block */
        ret = FlashBlockErase(data->block, data->block);
        if (ret != SELFLIB_OK) {
                ret |= 0x00020000;
                goto end;
        }
        do {
                ret = FlashStatusCheck();
        } while (ret == SELFLIB_BUSY);
        if (ret != SELFLIB_OK) {
                ret |= 0x00030000;
                goto end;
        }

        /* Flash writing */
        info = (data->data_length + 7) / 8;
        info *= 2;
        ret = FlashWordWrite(out, data->data, info);
        if (ret != SELFLIB_OK) {
                ret |= 0x00040000;
                goto end;
        }

        /* Internal verify */
        ret = FlashBlockIVerify(data->block, data->block);
        if (ret != SELFLIB_OK) {
                ret |= 0x00050000;
                goto end;
        }
        do {
                ret = FlashStatusCheck();
        } while (ret == SELFLIB_BUSY);
        if (ret != SELFLIB_OK) {
                ret |= 0x00060000;
                goto end;
        }

        /* Specification boot swap */
        if (flag & BOOT_FLAG_SETINFO) {
                info = 0x1f00003e;
                ret = FlashGetInfo((u32)4);
                ret &= 0x00000001;
                info |= ret;
                ret = FlashSetInfo(info, (u32)0);
                if (ret != SELFLIB_OK) {
                        ret |= 0x00070000;
                        goto end;
                }
                do {
                        ret = FlashStatusCheck();
                } while (ret == SELFLIB_BUSY);
                if (ret != SELFLIB_OK) {
                        ret |= 0x00080000;
                        goto end;
                }
        }

        /* Execution boot swap */
        if (flag & BOOT_FLAG_BOOTSWAP) {
                ret = FlashBootSwap();
                if (ret != SELFLIB_OK) {
                        ret |= 0x00090000;
                        goto end;
                }
        }

        ret = 0;

end:
        /* Flash environment end */
        FlashEnv((u32)0);

        /* FLMD0 to Low */
        PM3L.7 = 1;

        IMR0 = mask[0];
        IMR1 = mask[1];
        IMR2 = mask[2];
        IMR3 = mask[3];

        return ret;
}
```

図 7-13　usbf_fwup_execram.c

（5）usbf_fwup_pwonchk_usr.c

電源投入時，またはリセット時に，ファームウエア・アップデート・プログラム，またはユーザ・プログラムを実行するための判定処理を記述します。

TK-850/JG3L+USB ボード上にある SW3, SW4 は，TK-850/JH3U-SP ボードとフラグ位置が異なるため，以下の場所を変更します。

```
#pragma ioreg

#include "types.h"

#define SW_PUSHED   (0x00)      /* pushed switch SW3 and SW4 */
#define SW_STATUS   (0x30)      /* switch status SW3 and SW4 */

s32 usbf_fwup_pwonchk_usr(void);

s32 usbf_fwup_pwonchk_usr(void)
{
   s32    ret = -1;
   u08    sts;

   sts = P9H;
   if ((sts & SW_STATUS) == SW_PUSHED) {
      ret = 0;
   }

   return ret;
}
```

SW3, SW4 のビット位置を合わせます

図 7-14　usbf_fwup_pwonchk_usr.c

7.3.2　ファイル転送アプリケーションiniファイルの変更

FirmupdateGUI ディレクトリ以下にある UsbfUpdateini ファイルを移植環境に合わせてカスタマイズします。

表 7-6　ファイル転送アプリケーションのカスタマイズ・ファイル

ファイル名	説　明
UsbfUpdate.ini	ファイル転送アプリケーションの設定

（1）UsbfUpdate.ini

uPD70F3796 の設定を追加します。

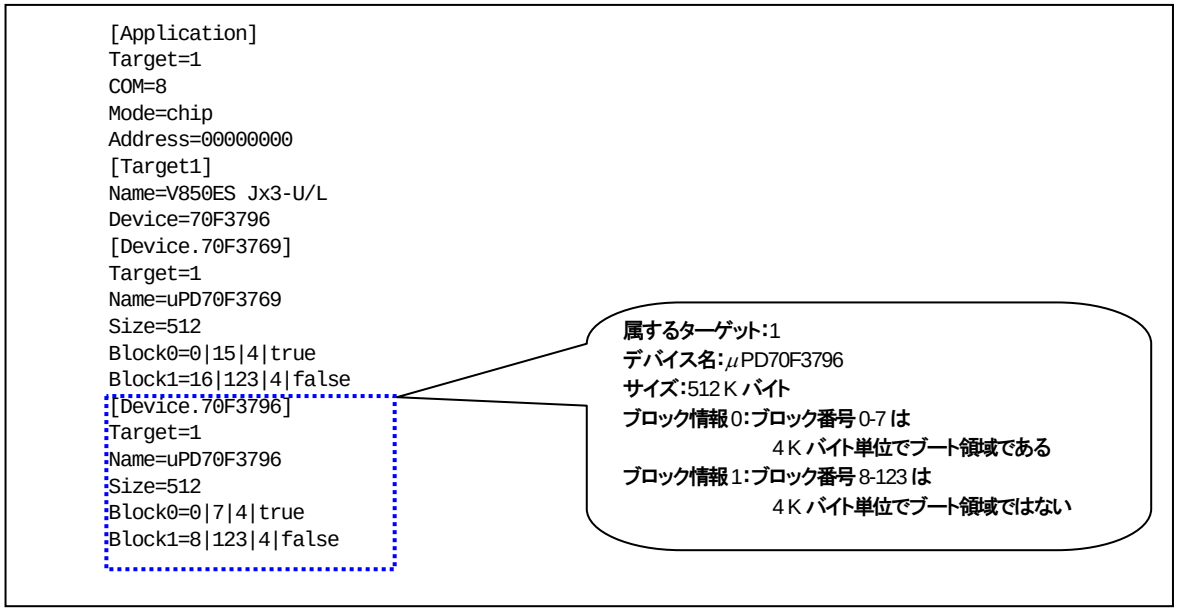


図 7-15　UsbUpdate.ini

8. データ通信仕様

8.1 書き換え通信インタフェース仕様

ファームウエア・アップデート・プログラムの PC と評価ボード間で通信する内容を示します。

8.1.1 通信データの構成

PC は最初に開始レコード，最後に終了レコードを送信します。フラッシュ・メモリに書き込むデータは，データ・レコードの形式で送信します。

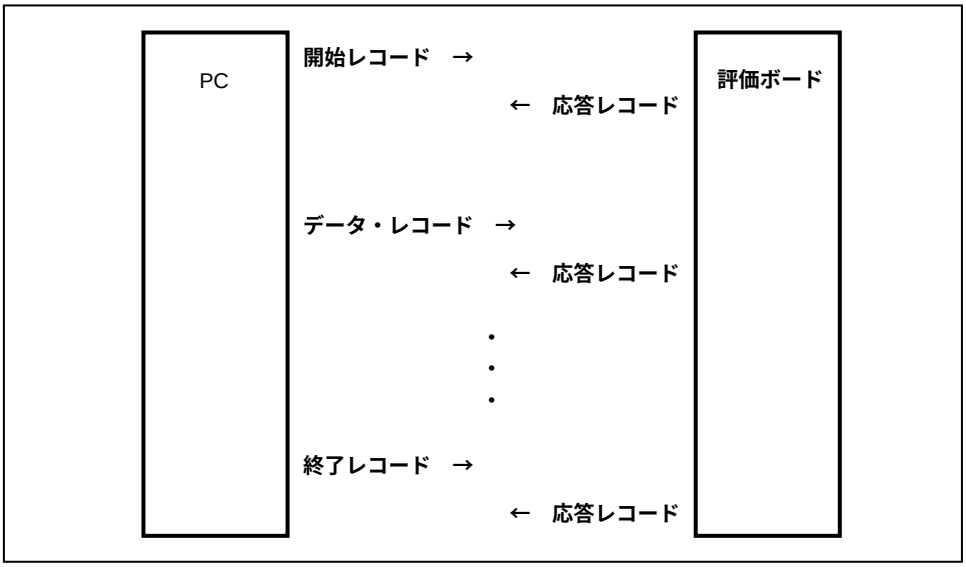


図 8-1 通信データ・シーケンス

8.1.2 PC側送信データ

PC 側は，開始レコード，データ・レコード，終了レコードを送信します。
各レコードは 1 レコードずつ送信し，応答レコードを受信するまで，次のレコードの送信は行いません。

（1）開始レコード
書き換えの実行時に，最初に送信するレコードです。

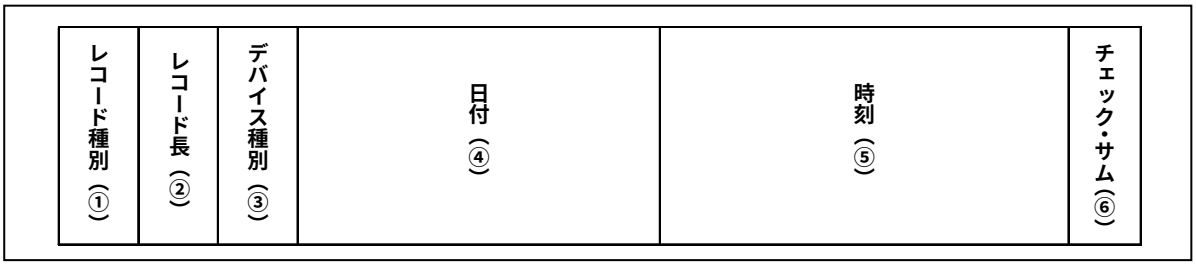


図 8-2　開始レコードの形式

①レコード種別
　レコードの種類
　1 バイト
　開始レコードのレコード種別は，0x00

②レコード長
　デバイス種別以降のバイト数
　1 バイト

③デバイス種別
　デバイスの種類
　1 バイト

④日付
　現在の日付
　年，月，日が各 1 バイト
　年は西暦年の下 2 桁

⑤時刻
　現在の時刻
　時，分，秒が各 1 バイト

⑥チェック・サム
　レコードのチェック・サム
　1 バイト
　レコード長とデバイス種別と日付と時刻のチェック・サム
　各バイトの値を加算した合計値の 1 の補数の下位 8 ビット

(2) データ・レコード
書き込むデータのレコードです。

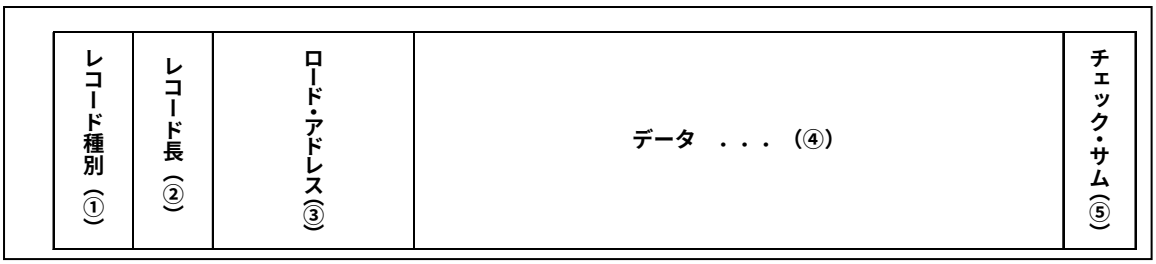


図 8-3　データ・レコードの形式

①レコード種別
　レコードの種類
　1 バイト
　データ・レコードのレコード種別は，0x0f

②レコード長
　ロード・アドレス以降のバイト数
　1 バイト

③ロード・アドレス
　フラッシュ・メモリのアドレス
　4 バイト
　このアドレスからデータが書き込まれる
　ロード・アドレスは，32 ビット数値でリトル・エンディアンの形式

④データ
　フラッシュ・メモリに書き込むデータ
　1 レコードあたり最大で 256 バイト

⑤チェック・サム
　レコードのチェック・サム
　1 バイト
　レコード長とロード・アドレスとデータのチェック・サム
　各バイトの値を加算した合計値の 1 の補数の下位 8 ビット

(3) 終了レコード

すべてのデータを送信後，最後に送信するレコードです。

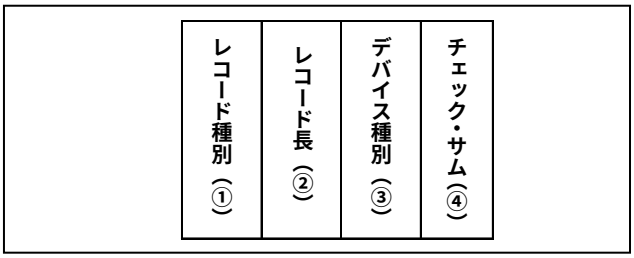


図 8-4　終了レコードの形式

①レコード種別

レコードの種類

1 バイト

終了レコードのレコード種別は，0xf0

②レコード長

デバイス種別以降のバイト数

1 バイト

③デバイス種別

デバイスの種類

1 バイト

④チェック・サム

レコードのチェック・サム

1 バイト

レコード長とデバイス種別のチェック・サム

各バイトの値を加算した合計値の 1 の補数の下位 8 ビット

8.1.3 評価ボード側送信データ

評価ボードは，PC からのレコードに対して，応答レコードを送信します。

(1) 応答レコード

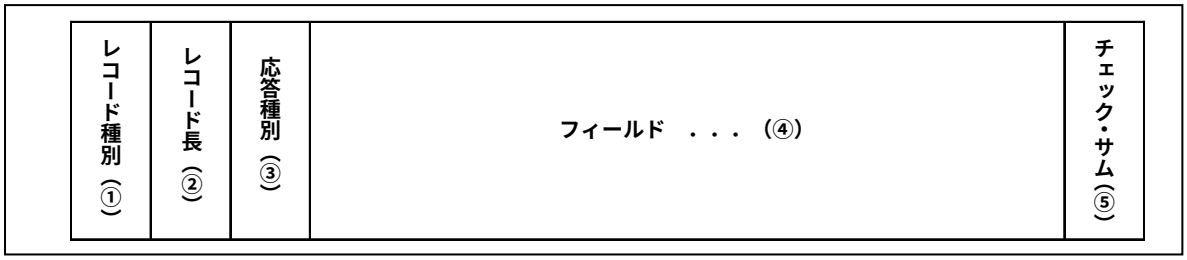


図 8-5　応答レコードの形式

①レコード種別
　レコードの種類
　1 バイト
　応答を返す対象のレコードのレコード種別

②レコード長
　応答種別以降のバイト数
　1 バイト

③応答種別
　応答種別
　1 バイト
　以下の 3 種類
　0x00　　：ACK
　0x0f　　：NAK（再送要求）
　0xf0　　：NAK（エラー終了）

④フィールド
　エラーの場合は，エラー・コード 1 バイト
　エラーでない場合は，レコード種別によって内容が異なる
　開始レコード　　：デバイス種別
　データ・レコード　：ロード・アドレス
　終了レコード　　：デバイス種別

⑤チェック・サム
　レコードのチェック・サム
　1 バイト
　レコード長と応答種別とフィールドのチェック・サム
　各バイトの値を加算した合計値の 1 の補数の下位 8 ビット

ホームページとサポート窓口

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

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

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2010.12.10	—	初版発行

製品ご使用上の注意事項

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

1. 未使用端子の処理

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

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

2. 電源投入時の処置

   【注意】電源投入時は，製品の状態は不定です。

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

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

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

3. リザーブアドレスのアクセス禁止

   【注意】リザーブアドレスのアクセスを禁止します。

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

4. クロックについて

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

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

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

5. 製品間の相違について

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

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

ご注意書き

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

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

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

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

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

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

ルネサス エレクトロニクス株式会社

■営業お問合せ窓口 http://www.renesas.com

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

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

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

Colophon 1.0