お客様各位

カタログ等資料中の旧社名の扱いについて

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

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

2010年4月1日
ルネサスエレクトロニクス株式会社

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

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

ご注意書き

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

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

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

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

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

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

RENESAS

アプリケーション・ノート

μPD78F0730

8 ビット・シングルチップ・マイクロコントローラ

USB-シリアル変換ソフトウエア編

資料番号 U19660JJ2V0AN00（第 2 版）
発行年月 February 2010 NS

［メ　　モ］

目次要約

MINICUBE は NEC エレクトロニクス株式会社の登録商標です。

Windows および Windows Vista は，米国 Microsoft Corporation の米国およびその他の国における登録商標または商標です。

PC/AT は，米国 IBM 社の商標です。

その他，この資料に記載されている会社名，製品名などは，各社の商標または登録商標です。

はじめに

対 象 者　このアプリケーション・ノートは，μPD78F0730 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。

目　　的　このアプリケーション・ノートは，μPD78F0730 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・ソフトウエア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。

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。

- μPD78F0730 の USB ファンクション・コントローラの概要
- USB 規格の概要
- サンプル・ソフトウエアの仕様
- 開発環境
- サンプル・ソフトウエアの応用

読 み 方　このアプリケーション・ノートの読者には，電気，論理回路，マイクロコンピュータの一般知識を必要とします。

- μPD78F0730 のハードウエア機能，および電気的特性を知りたいとき
  → 別冊の**μPD78F0730　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。

- μPD78F0730 の命令機能を知りたいとき
  → 別冊の **78K/0 シリーズ　ユーザーズ・マニュアル　命令編**を参照してください。

凡　　例

データ表記の重み：	左が上位桁，右が下位桁
注：	本文中につけた注の説明
注意：	気をつけて読んでいただきたい内容
備考：	本文中の補足説明
数の表記：	2 進数または 10 進数 … XXXX
	16 進数 … 0xXXXX
2 のべき数を示す接頭語（アドレス空間，メモリ容量）：	
	K（キロ） … $2^{10} = 1024$
	M（メガ） … $2^{20} = 1024^2$
	G（ギガ） … $2^{30} = 1024^3$
	T（テラ） … $2^{40} = 1024^4$
	P（ペタ） … $2^{50} = 1024^5$
	E（エクサ） … $2^{60} = 1024^6$

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

○ μPD78F0730 **に関する資料**

資 料 名	資料番号
μPD78F0730　**ユーザーズ・マニュアル**	U19014J
μPD78F0730　**ユーザーズ・マニュアル**　USB-**シリアル変換ドライバ編**	U19340J
μPD78F0730　**アプリケーション・ノート**　USB-**シリアル変換ソフトウエア編**	**このマニュアル**
78K/0 **シリーズ　ユーザーズ・マニュアル　命令編**	U12326J

○ **開発ツールに関する資料（ユーザーズ・マニュアル）**

資 料 名		資料番号
CC78K0 Ver.3.70　C **コンパイラ**	**操作編**	U17201J
	言語編	U17200J
RA78K0 Ver.3.80　**アセンブラ・パッケージ**	**操作編**	U17199J
	言語編	U17198J
	構造化アセンブリ言語編	U17197J
SM+　**システム・シミュレータ**	**操作編**	U18601J
	ユーザ・オープン・インタフェース編	U18212J
SM78K Ver.2.52　**システム・シミュレータ**	**操作編**	U16768J
PM plus Ver.5.10　**プロジェクト・マネージャ**		U16569J
ID78K0-NS Ver.2.70　**統合デバッガ**	**操作編**	U17729J
ID78K0-QB Ver.3.00　**統合デバッガ**	**操作編**	U18492J
QB-780731　**インサーキット・エミュレータ**		U17804J
QB-MINI2　**プログラミング機能付きオンチップ・デバッグ・エミュレータ**		U18371J
PG-FP5　**フラッシュ・メモリ・プログラマ**		U18865J
PG-FP4　**フラッシュ・メモリ・プログラマ**		U15260J

目　次

第1章 概　　説

このアプリケーション・ノートは，マイクロコントローラ μPD78F0730 内蔵の USB ファンクション・コントローラ用に作成された USB-シリアル変換用サンプル・ソフトウエアについて説明します。

主に次に示す内容で構成されます。

- サンプル・ソフトウエアの仕様
- サンプル・ソフトウエアを利用したアプリケーション・プログラム開発のための環境
- サンプル・ソフトウエアを利用するための参考情報

この章では，サンプル・ソフトウエアの概要と，適用対象となるマイクロコントローラについて説明します。

1. 1 概　　要

1. 1. 1 USB ファンクション・コントローラの特徴

μPD78F0730 内蔵の USB ファンクション・コントローラ（USBF）には，次の特徴があります。

- Universal Serial Bus Specification に準拠
- 12 Mbps（フル・スピード）転送に対応
- 転送用のエンドポイントを内蔵

表 1－1　μPD78F0730 内蔵 USB ファンクション・コントローラのエンドポイント構成

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

- 内部クロックと外部クロックを選択可能（f_{USB} = 48 MHz）注

  X1 発振回路で生成するクロック（f_X = 12 または 16 MHz）を 4 または 3 逓倍

  外部入力クロック（f_{EXCLK} = 12 または 16 MHz）を 4 または 3 逓倍

注　サンプル・ソフトウエアでは内部クロックを選択します。

1.1.2 サンプル・ソフトウエアの特徴

このサンプル・ソフトウエアには，次の特徴があります。機能や動作の詳細は第 3 章　サンプル・ソフトウエアの仕様を参照してください。

- 仮想 COM ポートとして動作
- 専用のホスト・ドライバを使用（詳細は1．1．4 ホスト・ドライバの構成 を参照）
- USB ファンクション・コントローラで受信したデータをそのまま UART から送信
- UART で受信したデータをそのまま USB ファンクション・コントローラから送信
- ターミナル・ソフトからボー・レート，ストップ・ビット，データ長，パリティ・ビットを変更可能
- ベンダ・クラスとして動作し，3 つのエンドポイント（Control，Bulk In，Bulk Out）を使用
- サスペンド／レジューム機能は非サポート
- バス・パワード・デバイスとして動作
- 次に示すサイズのメモリを占有（ベクタ・テーブルを除く）

  ROM：約 4.2 K バイト

  RAM：約 0.3 K バイト
- 対象 OS: Windows® 2000，Windows® XP，Windows Vista®

1.1.3 サンプル・ソフトウエアの構成

このサンプル・ソフトウエアは次のファイルで構成されています。

表 1−2　サンプル・ソフトウエアのファイル構成

フォルダ	ファイル	概　要
Src	main.c	初期化，メイン・ルーチン
	usbf78k.c	USBF 初期化，割り込み処理，バルク転送，コントロール転送
	uart_ctrl.c	UART 通信制御
	usbf78k_vendor.c	ベンダ・クラス処理
	boot.asm	ブート処理ルーチン
include	errno.h	エラー・コード定義
	main.h	main.c 関数プロトタイプ宣言
	Types.h	ユーザ型宣言
	uart_ctrl.h	uart_ctrl.c 関数プロトタイプ宣言
	usbf78k.h	usbf78k.c 関数プロトタイプ宣言
	usbf78k_desc.h	ディスクリプタ定義
	usbf78k_sfr.h	USB ファンクション・コントローラ用レジスタ・アクセス用マクロ定義
	usbf78k_vendor.h	usbf78k_vendor.c 関数プロトタイプ宣言

備考　このほか，USB-シリアル変換用ホスト・ドライバ，PM+（NEC エレクトロニクス製統合開発ツール）で開発環境を構築した場合に生成されるプロジェクト関連ファイル一式も同梱されています。詳細は4. 2. 2　ホスト環境整備を参照してください。

1. 1. 4 ホスト・ドライバの構成

このサンプル・ソフトウエアで使用するホスト・ドライバは次のファイルで構成されています。

表 1−3　ホスト・ドライバのファイル構成

フォルダ		ファイル	概　要
DRIVER	win2k	necelusbvcom.inf	USB シリアル変換用ホスト・ドライバ 対象 OS: Windows 2000，Windows XP(32 ビット版)，Windows Vista(32 ビット版)
		necelusbdv.sys	
		necelvcom.sys	
	wlh_amd64	necelusbvcom.inf	USB シリアル変換用ホスト・ドライバ 対象 OS: Windows XP(64 ビット版) ，Windows Vista (64 ビット版)
		necelusbdv.sys	
		necelvcom.sys	

1. 2 μPD78F0730 の概要

ここでは，サンプル・ソフトウエアの制御の対象であるμPD78F0730 について説明します。

μPD78F0730 は，NEC エレクトロニクスの 8 ビット・シングルチップ・マイクロコントローラです。ROM/RAM，タイマ／カウンタ，シリアル・インタフェース，A/D コンバータ，D/A コンバータ，DMA コントローラ，USB ファンクション・コントローラなどの周辺機能を内蔵しています。詳細はμPD78F0730　ユーザーズ・マニュアルを参照してください。

μPD78F0730 には，主に次の特徴があります。

○ 高速（0.125 μs：高速システム・クロック 16 MHz 動作時）で命令実行が可能

○ 汎用レジスタ：8 ビット×32 レジスタ（8 ビット×8 レジスタ×4 バンク）

○ ROM, RAM 容量

項 目 / 品 名	プログラム・メモリ（ROM）	データ・メモリ	
	フラッシュ・メモリ注	内部高速 RAM 注	内部拡張 RAM 注
μPD78F0730	16 K バイト	1 K バイト	2 K バイト

注　内部フラッシュ・メモリ，内部高速 RAM，内部拡張 RAM の容量は，レジスタにより変更可能です。

○ USB ファンクション・コントローラ（USBF）を搭載

○ 単一電源のフラッシュ・メモリ内蔵

○ セルフ・プログラミング内蔵（ブート・スワップ機能あり）

○ オンチップ・デバッグ機能内蔵

○ パワーオン・クリア（POC）回路，低電圧検出（LVI）回路内蔵

○ ウォッチドッグ・タイマ（低速内蔵発振クロックで動作可能）内蔵

○ I/O ポート：19 本（N-ch オープン・ドレーン：2 本）

○ タイマ：5 チャネル

- 16 ビット・タイマ／イベント・カウンタ　：1 チャネル
- 8 ビット・タイマ／イベント・カウンタ　：2 チャネル
- 8 ビット・タイマ　：1 チャネル
- ウォッチドッグ・タイマ　：1 チャネル

○シリアル・インタフェース：3 チャネル

- UART　：1 チャネル
- CSI　：1 チャネル
- USB　：1 チャネル

第2章　USB の概要

この章では，サンプル・ソフトウエアが準拠する USB 規格の概要を説明します。

USB（Universal Serial Bus）は共通のコネクタでさまざまな周辺機器をホスト・コンピュータに接続できるようにするためのインタフェース規格です。ハブと呼ばれる分岐点を追加することで最大 127 個の機器を接続でき，Plug&Play で機器を認識できるホットプラグに対応しているなど，従来のインタフェースより柔軟で使いやすくなっています。現在では PC の USB インタフェース搭載率はほぼ 100%になってきており，PC と周辺機器間の標準インタフェースとして定着したと言えます。

USB 規格の策定と管理は USB Implementers Forum（USB-IF）という団体が行っています。USB 規格の詳細は USB-IF の公式ウェブサイト（www.usb.org）を参照してください。

2. 1　転送方式

USB 規格では，4 種類の転送方式（コントロール，バルク，インタラプト，アイソクロナス）が定義されています。各転送方式には表 2－1に示す特徴があります。

表 2－1　USB の転送方式

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

2. 2　エンドポイント

エンドポイントはホスト・デバイスが通信相手を特定するための情報の 1 つで，0-15 の番号と方向（IN/OUT）で指定されます。エンドポイントは周辺機器で使用するデータ通信経路ごとに用意しなければならず，複数の通信経路で共用できません[注]。たとえば，SD カードへの書き込み／読み出しとプリント出力の機能を持った機器の場合，SD カードへの書き込み用エンドポイント，読み出し用エンドポイント，プリント出力用エンドポイントを個別に持つ必要があります。どのような機器でも必ず使用するコントロール転送には，エンドポイント 0 を使用します。

データ通信を行うとき，ホスト・デバイスは機器を特定する USB デバイス・アドレスとともにエンドポイント（番号と方向）を使用して，機器内部の通信先を特定します。

エンドポイントのための物理的な回路として周辺機器内にバッファ・メモリを装備し，USB と通信先（メモリなど）の速度差を吸収する FIFO の役割も果たします。

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

2. 3　デバイス・クラス

USB を介して接続する周辺機器（ファンクション・デバイス）には，その機能によりさまざまなデバイス・クラスが定義されています。代表的なクラスとしてマス・ストレージ・クラス（MSC），プリンタ・クラス，ヒューマン・インタフェース・デバイス・クラス（HID）などがあります。各デバイス・クラスにはプロトコルなどで標準仕様が定められているため，これに準拠していれば共通のホスト・ドライバを使用できます。デバイスごとにドライバを用意する必要がなくなるため，ユーザはどんな機器でも同じように使用でき，ベンダはアプリケーション・プログラムの開発工数を節減できます。

2. 4　リクエスト

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

2. 4. 1 種　　類

標準リクエスト，クラス・リクエスト，ベンダ・リクエストの 3 種類があります。

サンプル・ソフトウエアが対応するリクエストについては，3. 1. 4　**リクエストへの対応**を参照してください。

(1) 標準リクエスト

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

表 2−2　標準リクエスト一覧

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

(2) クラス・リクエスト

デバイス・クラス固有のリクエストです。共通のホスト・ドライバを使用することで，このリクエストに対応できます。bmRequestType フィールドのビット 6 の値が 0，ビット 5 の値が 1 のとき，そのリクエストはクラス・リクエストです。

(3) ベンダ・リクエスト

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

2. 4. 2　フォーマット

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

表 2－3　USB リクエストのフォーマット

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

2. 5　ディスクリプタ

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

2. 5. 1　種　　類

次に示す 5 種類のディスクリプタが定義されています。

- デバイス・ディスクリプタ
  どのデバイスにも必ず存在するディスクリプタで，対応している USB 仕様のバージョン，デバイス・クラス，プロトコル，Endpoint0 に対する転送で利用可能な最大パケット長，ベンダ ID，プロダクト ID などの基本情報が含まれています。
  GET_DESCRIPTOR_Device リクエストに応答して送信するディスクリプタです。
- コンフィギュレーション・ディスクリプタ
  すべてのデバイスに 1 つ以上存在するディスクリプタで，デバイスの属性（電源供給方法），消費電力などの情報を含みます。
  GET_DESCRIPTOR_Configuration リクエストに応答して送信するディスクリプタです。
- インタフェース・ディスクリプタ
  インタフェースごとに必要なディスクリプタで，インタフェース識別番号，インタフェース・クラス，サポートするエンドポイントの数などが含まれます。
  GET_DESCRIPTOR_Configuration リクエストに応答して送信するディスクリプタです。
- エンドポイント・ディスクリプタ
  インタフェース・ディスクリプタに指定されたエンドポイントごとに必要なディスクリプタで，転送タイプ（転送方向），転送で利用可能な最大パケット長，転送のインターバルを定義します。ただし，Endpoint0 はこのディスクリプタを持ちません。
  GET_DESCRIPTOR_Configuration リクエストに応答して送信するディスクリプタです。
- ストリング・ディスクリプタ
  任意の文字列を含むディスクリプタです。
  GET_DESCRIPTOR_String リクエストに応答して送信するディスクリプタです。

2. 5. 2　フォーマット

ディスクリプタのサイズとフィールドは，次のように種類ごとに異なります。

備考　各フィールドのデータ並びはリトル･エンディアンです。

表 2－4　デバイス・ディスクリプタのフォーマット

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

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

表 2－5　コンフィギュレーション・ディスクリプタのフォーマット

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

表 2－6　インタフェース・ディスクリプタのフォーマット

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

表 2－7　エンドポイント・ディスクリプタのフォーマット

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

表 2－8　ストリング・ディスクリプタのフォーマット

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

第3章　サンプル・ソフトウエアの仕様

この章では，μPD78F0730 向け USB シリアル変換サンプル・ソフトウエアの機能と処理内容の詳細，および実装している関数の仕様について説明します。

3. 1　概　　要

3. 1. 1　機　　能

このサンプル・ソフトウエアには次のような処理が実装されています。

(1) 初期化

μPD78F0730 のメモリ・サイズ，クロックの設定を行います。詳細は**3. 2　CPU 初期化処理**を参照してください。

(2) USB 制御処理

USBF の初期化，割り込みハンドラ，データ送信処理，およびデータ受信処理を行います。詳細は**3. 3　USB 制御処理**を参照してください。

また，μPD78F0730 が自動応答しない USB リクエストに対する応答処理を実装しています。詳細は**3. 1. 4 リクエストへの対応**を参照してください。

(3) UART 制御処理

UART の初期化，割り込みハンドラ，データ送信処理，およびデータ受信処理を行います。

詳細は**3. 4　UART 制御処理**を参照してください。

(4) UART-USB 間ブリッジ処理

UART 受信→USB 送信バッファ格納処理では，UART 受信完了割り込み処理で読み出した受信データを USB 送信バッファに格納します。

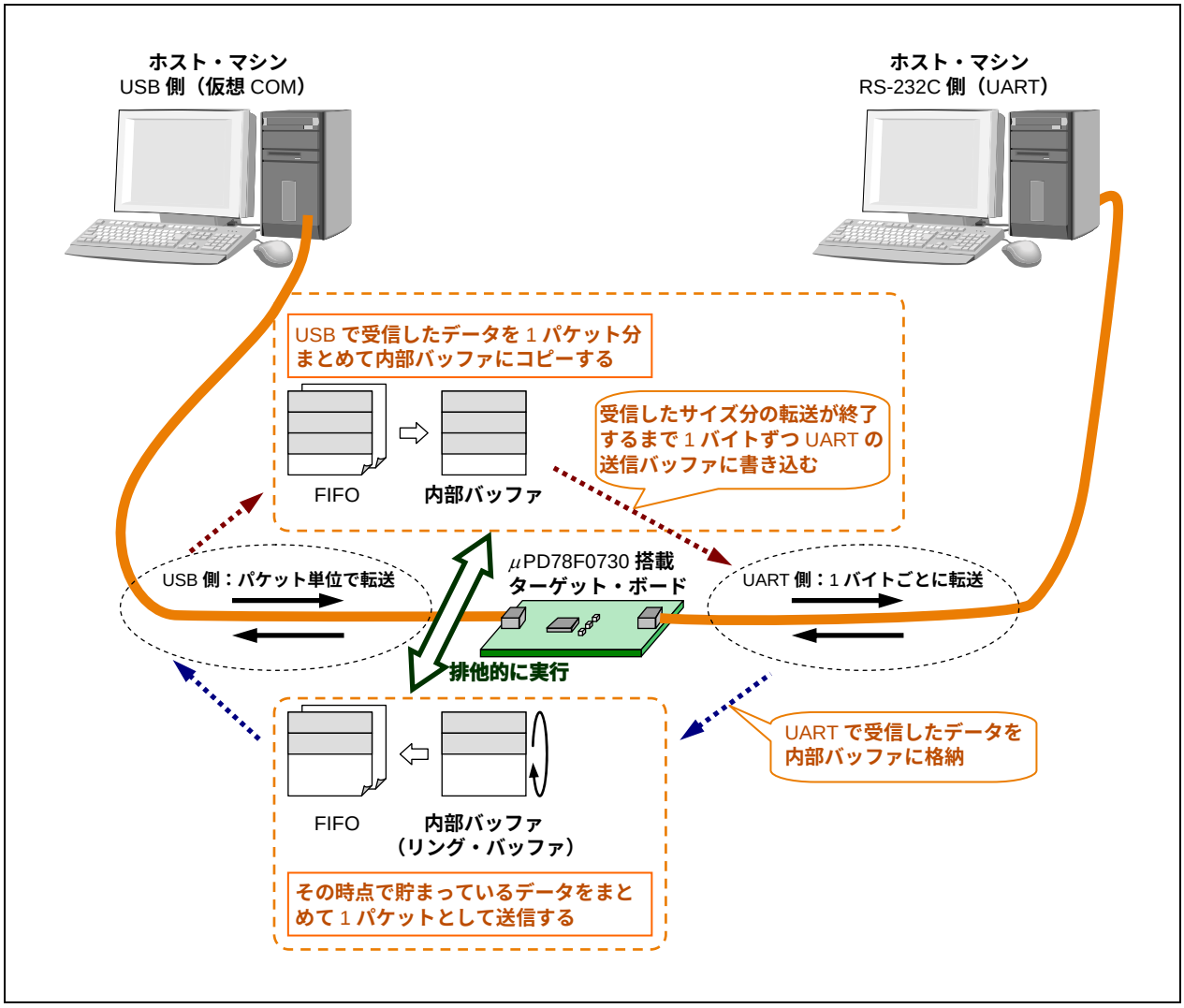
USB 受信→UART 送信処理では，バルク・アウト転送（受信）用エンドポイントにあるデータを読み出し，UART から送信します。

詳細は**3. 5　UART-USB 間ブリッジ処理**を参照してください。

3. 1. 2　システムの構成

サンプル・ソフトウエアを利用するシステムの構成と処理の流れを図 3－1に示します。

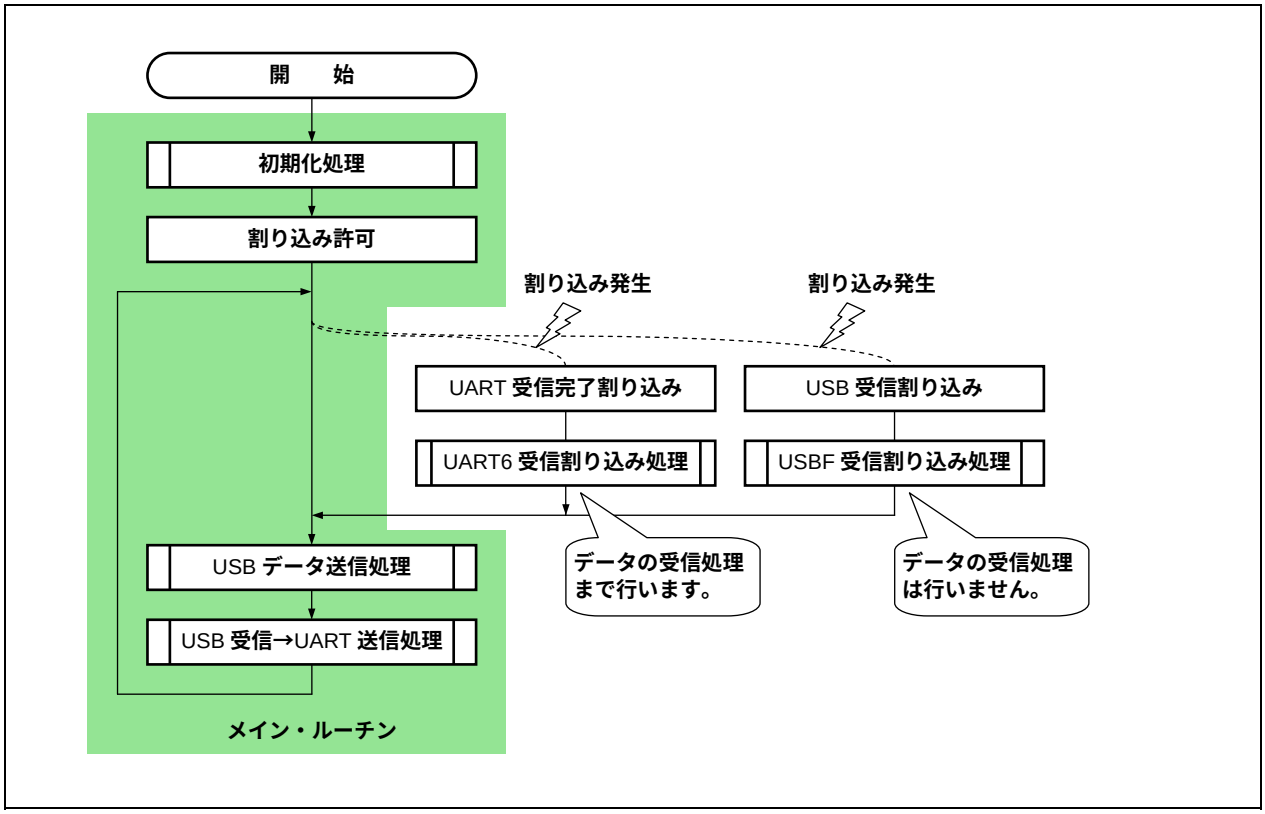
図 3－1　システムの構成と処理の流れ



3. 1. 3　処理の流れ

サンプル・ソフトウエアを実行すると図 3－2および図 3－3に示す処理を行います。

図 3－2　サンプル・ソフトウエアの処理の流れ 1

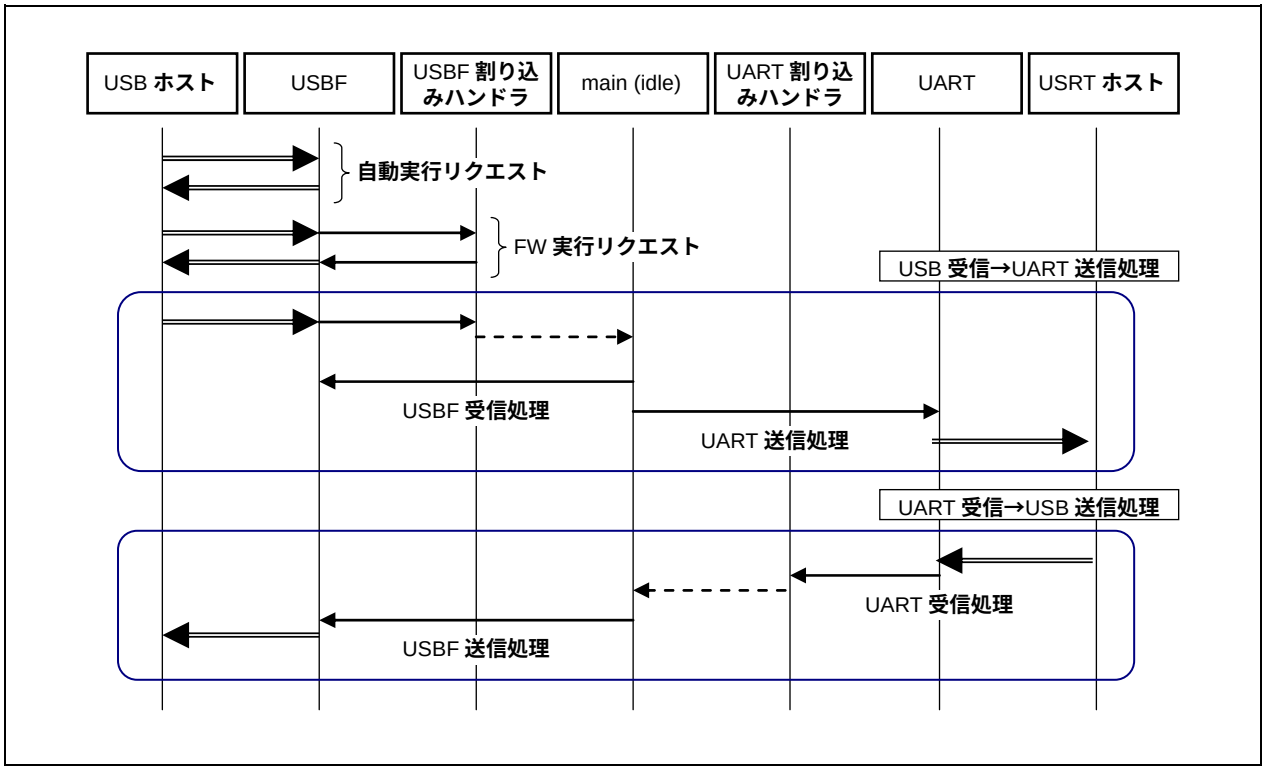


<1> ボード起動時の初期化が終了すると，ループ処理（メイン処理関数（main）内），および各種割り込み信号による割り込み処理を開始します。USB においてホスト・マシンとの接続確立後，UART-USB 間でデータの送受信が開始されます。

<2> USB でデータを受信すると，USB の受信完了割り込みにより USBF 受信割り込みハンドラが起動されます。このハンドラでは，受信完了のフラグを設定するのみで，データの読み出しを行いません。割り込み処理が終了したあと，メイン処理関数（main）内で USB 受信→UART 送信処理関数（usbf78k_usb_to_uart）を呼び出し，USB 受信処理および UART 送信処理を実行します。

<3> UART でデータを受信すると，UART の受信完了割り込みにより UART 受信完了割り込みハンドラが起動されます。このハンドラでは，受信したデータを読み出し，さらに UART 受信→USB 送信バッファ格納処理関数（usbf78k_uart_to_usb）を呼び出し，内部バッファにデータを格納します。格納したデータは，メイン処理関数（main）内で USB データ送信処理関数（usbf78k_send_txbuf）を呼び出すことで，USB から送信します。

図 3−3　サンプル・ソフトウエアの処理の流れ 2



<1>　**自動実行リクエスト**

ハードウエア（μPD78F0730**）が自動的に応答します。**

<2>　FW **実行リクエスト**

ファームウエア（サンプル・ソフトウエア）が USBF **割り込みハンドラ処理内で応答します。**

<3>　USB **受信→**UART **送信処理**

USBF **割り込みハンドラ処理で受信を判別し，**USB **で受信したデータを** UART **から送信します。**

<4>　UART **受信→**USB **送信処理**

UART **受信完了割り込みハンドラ処理で受信データをバッファに格納し，**USB **から送信します。**

3. 1. 4　リクエストへの対応

表 3－1にハードウエア（μPD78F0730）およびファームウエア（サンプル・ソフトウエア）で定義されている USB リクエストを示します。

表 3－1　USB リクエストの処理

リクエスト名	コード								処　　理
	0	1	2	3	4	5	6	7	
標準リクエスト									
GET_INTERFACE	0x81	0x0A	0x00	0x00	0xXX	0xXX	0x01	0x00	HW 自動応答
GET_CONFIGURATION	0x80	0x08	0x00	0x00	0x00	0x00	0x01	0x00	HW 自動応答
GET_DESCRIPTOR Device	0x80	0x06	0x00	0x01	0x00	0x00	0xXX	0xXX	HW 自動応答
GET_DESCRIPTOR Configuration	0x80	0x06	0x00	0x02	0x00	0x00	0xXX	0xXX	HW 自動応答
GET_DESCRIPTOR String	0x80	0x06	0x00	0x03	0x00	0x00	0xXX	0xXX	FW 応答
GET_STATUS Device	0x80	0x00	0x00	0x00	0x00	0x00	0x02	0x00	HW 自動応答
GET_STATUS Interface	0x81	0x00	0x00	0x00	0xXX	0xXX	0x02	0x00	HW 自動 STALL 応答
GET_STATUS Endpoint n	0x82	0x00	0x00	0x00	0xXX	0xXX	0x02	0x00	HW 自動応答
CLEAR_FEATURE Device	0x00	0x01	0x01	0x00	0x00	0x00	0x00	0x00	HW 自動応答
CLEAR_FEATURE Interface	0x01	0x01	0x00	0x00	0xXX	0xXX	0x00	0x00	HW 自動 STALL 応答
CLEAR_FEATURE Endpoint n	0x02	0x01	0x00	0x00	0xXX	0xXX	0x00	0x00	HW 自動応答
SET_DESCRIPTOR	0x00	0x07	0xXX	0xXX	0xXX	0xXX	0xXX	0xXX	FW STALL 応答
SET_FEATURE Device	0x00	0x03	0x01	0x00	0x00	0x00	0x00	0x00	HW 自動応答
SET_FEATURE Interface	0x02	0x03	0xXX	0xXX	0xXX	0xXX	0x00	0x00	HW 自動 STALL 応答
SET_FEATURE Endpoint n	0x02	0x03	0x00	0x00	0xXX	0xXX	0x00	0x00	HW 自動応答
SET_INTERFACE	0x01	0x0B	0xXX	0xXX	0xXX	0xXX	0x00	0x00	HW 自動応答
SET_CONFIGURATION	0x00	0x09	0xXX	0xXX	0x00	0x00	0x00	0x00	HW 自動応答
SET_ADDRESS	0x00	0x05	0xXX	0xXX	0x00	0x00	0x00	0x00	HW 自動応答
ベンダ・リクエスト									
LINE_CONTROL	0x40	0x0B	0x00	0x00	0x00	0x00	0x06	0x00	FW 応答
SET_DTR_RTS	0x40	0x0B	0x00	0x00	0x00	0x00	0x02	0x00	FW 応答
SET_XON_XOFF_CHR	0x40	0x0B	0x00	0x00	0x00	0x00	0x03	0x00	FW 応答
OPEN_CLOSE	0x40	0x0B	0x00	0x00	0x00	0x00	0x02	0x00	FW 応答
SET_ERR_CHR	0x40	0x0B	0x00	0x00	0x00	0x00	0x03	0x00	FW 応答
その他のリクエスト	**上記以外**								FW STALL 応答

備考　HW：ハードウエア（μPD78F0730）

FW：ファームウエア（サンプル・ソフトウエア）

0xXX：不定値

(1) 標準リクエスト

ハードウエア（μPD78F0730）が自動的に応答しないリクエストに対し，サンプル・ソフトウエアは次のような応答処理を行います。

(a) GET_DESCRIPTOR_string

ホストがファンクション・デバイスのストリング・ディスクリプタを取得するためのリクエストです。

このリクエストを受信すると，サンプル・ソフトウエアは要求されたストリング・ディスクリプタの送信処理（コントロール・リード転送）を行います。

(b) SET_DESCRIPTOR

ホストがファンクション・デバイスのディスクリプタを設定するためのリクエストです。

このリクエストを受信すると，サンプル・ソフトウエアは STALL 応答を返します。

(2) ベンダ・リクエスト

サンプル・ソフトウエアには，次の 5 種類のリクエストへの応答処理が実装されています。

- LINE_CONTROL
- SET_DTR_RTS
- SET_XON_XOFF_CHR
- OPEN_CLOSE
- SET_ERR_CHR

各リクエストの詳細は**3. 6　ベンダ・リクエストのフォーマット**を参照してください。

(3) 定義されていないリクエスト

定義されていないリクエストを受信すると，サンプル・ソフトウエアは STALL 応答を返します。

3. 1. 5　ディスクリプタの設定

サンプル・ソフトウエアでの各ディスクリプタの設定を次に示します。各ディスクリプタの設定は，ヘッダ・ファイル "usbf78k_desc.h" に記述されています。

(1) デバイス・ディスクリプタ

GET_DESCRIPTOR_device リクエストに応答して送信されるディスクリプタです。

GET_DESCRIPTOR_device リクエストにはハードウエアが自動的に応答するため，設定内容は USB ファンクション・コントローラの初期化時に UF0DDn レジスタ（n = 0-17）に格納されます。

表 3−2　デバイス・ディスクリプタの設定

フィールド	サイズ（バイト）	設定値	説　明
bLength	1	0x12	ディスクリプタのサイズ：18 バイト
bDescriptorType	1	0x01	ディスクリプタの種類：デバイス
bcdUSB	2	0x0200	USB 仕様リリース番号：USB 2.0
bDeviceClass	1	0xFF	クラス・コード：ベンダ・クラス
bDeviceSubClass	1	0x00	サブクラス・コード：なし
bDeviceProtocol	1	0x00	プロトコル・コード：固有プロトコル未使用
bMaxPacketSize0	1	0x40	Endpoint0 の最大パケット・サイズ：64
idVendor	2	0x0409	ベンダ ID：NEC
idProduct	2	0x01CD	プロダクト ID：μPD78F0730
bcdDevice	2	0x0001	デバイスのリリース番号：第 1 版
iManufacturer	1	0x01	製造者を表すストリング・ディスクリプタへのインデックス：1
iProduct	1	0x02	製品を表すストリング・ディスクリプタへのインデックス：2
iSerialNumber	1	0x03	デバイスの製造番号を表すストリング・ディスクリプタへのインデックス：3
bNumConfigurations	1	0x01	コンフィギュレーションの数：1

(2) コンフィギュレーション・ディスクリプタ

GET_DESCRIPTOR_configuration リクエストに応答して送信されるディスクリプタです。

GET_DESCRIPTOR_configuration リクエストにはハードウエアが自動的に応答するため，設定内容は USB ファンクション・コントローラの初期化時に UF0CIEn レジスタ（n = 0-255）に格納されます。

表 3−3　コンフィギュレーション・ディスクリプタの設定

フィールド	サイズ（バイト）	設定値	説　明
bLength	1	0x09	ディスクリプタのサイズ：9 バイト
bDescriptorType	1	0x02	ディスクリプタの種類：コンフィギュレーション
wTotalLength	2	0x0020	コンフィギュレーション，インタフェース，およびエンドポイント・ディスクリプタの総バイト数：32 バイト
bNumInterfaces	1	0x01	このコンフィギュレーションが持つインタフェースの数：1
bConfigurationValue	1	0x01	このコンフィギュレーションの識別番号：1
iConfiguration	1	0x00	このコンフィギュレーションを記述するストリング・ディスクリプタへのインデックス：0
bmAttributes	1	0x80	このコンフィギュレーションの特徴：バス・パワード，リモート・ウエイクアップなし
bMaxPower	1	0x32	このコンフィギュレーションの最大消費電流：100 mA

(3) インタフェース・ディスクリプタ

GET_DESCRIPTOR_configuration リクエストに応答して送信されるディスクリプタです。

GET_DESCRIPTOR_configuration リクエストにはハードウエアが自動的に応答するため，設定内容は USB ファンクション・コントローラの初期化時に UF0CIEn レジスタ（n = 0-255）に格納されます。

表 3−4　Interface0 のインタフェース・ディスクリプタの設定

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

(4) エンドポイント・ディスクリプタ

GET_DESCRIPTOR_configuration リクエストに応答して送信されるディスクリプタです。

GET_DESCRIPTOR_configuration リクエストにはハードウエアが自動的に応答するため，設定内容は USB ファンクション・コントローラの初期化時に UF0CIEn レジスタ（n = 0-255）に格納されます。

サンプル・ソフトウエアではエンドポイントを 2 つ使用するため，ディスクリプタも 2 種類設定しています。

表 3－5　Endpoint2 のエンドポイント・ディスクリプタの設定

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

表 3－6　Endpoint1 のエンドポイント・ディスクリプタの設定

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

(5) ストリング・ディスクリプタ

GET_DESCRIPTOR_string リクエストに応答して送信されるディスクリプタです。

GET_DESCRIPTOR_string リクエストを受信すると，サンプル・ソフトウエアはこのディスクリプタの設定をヘッダ・ファイル "usbf78k_desc.h" から取り出して，USB ファンクション・コントローラの UF0E0W レジスタに格納します。

表 3−7　ストリング・ディスクリプタの設定

(a) String 0

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

(b) String 1

フィールド	サイズ（バイト）	設定値	説　明
bLength 注1	1	0x2A	ディスクリプタのサイズ：42 バイト
bDescriptorType	1	0x03	ディスクリプタの種類：ストリング
bString 注2	40	–	ベンダ：NEC Electronics Co.

注 1. bString フィールドのサイズにより設定値が異なります。

2. ベンダにより任意に設定できる領域のため，サイズや設定値は一定ではありません。

(c) String 2

フィールド	サイズ（バイト）	設定値	説　明
bLength 注1	1	0x16	ディスクリプタのサイズ：22 バイト
bDescriptorType	1	0x03	ディスクリプタの種類：ストリング
bString 注2	12	–	製品の種類：VirtualCom（仮想 COM ドライバ）

注 1. bString フィールドのサイズにより設定値が異なります。

2. ベンダにより任意に設定できる領域のため，サイズや設定値は一定ではありません。

(d) String 3

フィールド	サイズ（バイト）	設定値	説　明
bLength 注1	1	0x16	ディスクリプタのサイズ：22 バイト
bDescriptorType	1	0x03	ディスクリプタの種類：ストリング
bString 注2	20	–	シリアル番号：0_98765432

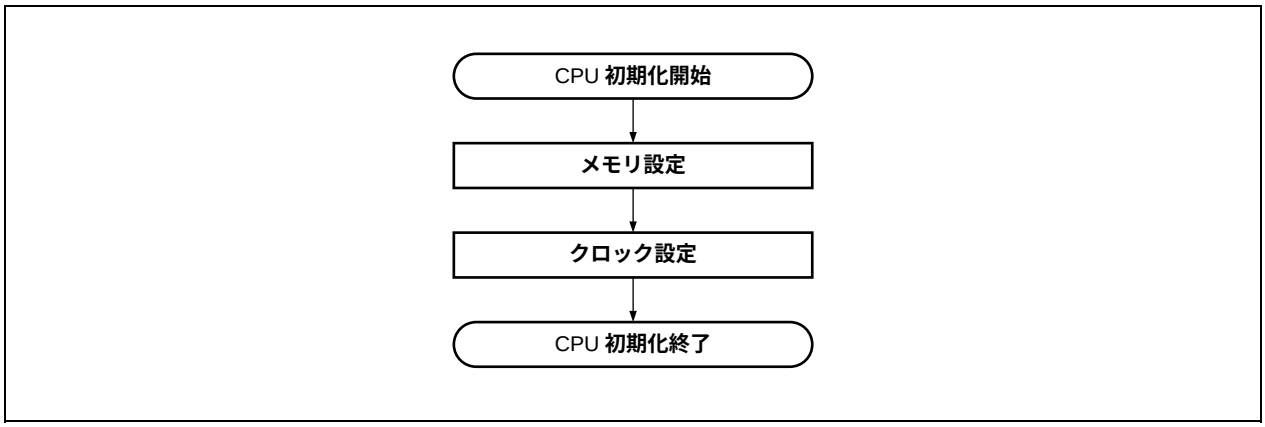
注 1. bString フィールドのサイズにより設定値が異なります。

2. ベンダにより任意に設定できる領域のため，サイズや設定値は一定ではありません。

3. 2　CPU 初期化処理

μPD78F0730 を使用するために必要な項目を設定します。

図 3－4　CPU 初期化処理フロー



(1) メモリ設定

メモリ・サイズ（ROM，内部拡張 RAM サイズ）の設定を行います。

① IMS レジスタに "0xC4" を書き込みます。この設定により，使用する ROM が 16 KB になります。

② IXS レジスタに "0x08" を書き込みます。この設定により，使用する RAM が 3 KB（ハイスピード：1 KB，拡張 2 KB）になります。

(2) クロック設定

高速システム・クロック，CPU クロック，PLL への供給クロックの設定を行います。

① OSCCTL レジスタに "0x41" を書き込みます。

② MOC レジスタに "0x00" を書き込みます。

③ OSTS レジスタに "0x01" を書き込みます。

④ PCC レジスタに "0x00" を書き込みます。

⑤ RCM レジスタに "0x00" を書き込みます。

⑥ PLLC レジスタに "0x01" を書き込みます。

⑦ MCM レジスタに "0x05" を書き込みます。

⑧ PLLC レジスタに "0x03" を書き込みます。

⑨ PLLC レジスタに "0x02" を書き込みます。

この設定により，高速内蔵発振クロック（f_{RH} = 16 MHz）で動作し，USB 動作クロック（f_{USB}）が 48 MHz になります。

3. 3　USB 制御処理

USB ファンクション・コントローラ（USBF）の初期化，割り込みハンドラ，データ送信処理，およびデータ受信処理を行います。

USBF の初期化，割り込みハンドラ，データ送信処理，およびデータ受信処理を行います。

サンプル・ソフトウエアでは，μPD78F0730 内蔵の USBF を動作させるための簡易ドライバを用意しています。

注意　μPD78F0730 内蔵の USBF の処理プログラムを用意していますが，サンプル・ソフトウエア用に最低限の処理を行うものです。汎用の USB ドライバとしての動作は保障しません。

- USBF の初期化

  メイン・ルーチンから呼び出し，μPD78F0730 内蔵の USBF の初期化を行います。

- USBF の割り込みハンドラ

  USBF の割り込みが発生するたびに呼び出される割り込み処理専用ルーチンです。

  注意　このサンプル・ソフトウエアでは，必要な割り込み以外をマスクします。
  このサンプル・ソフトウエアで使用する割り込みは，次の 2 つです。
  ・INTUSB0B で通知される RSUSPD，BUSRST，SETRQ，CPUDEC 割り込み
  ・INTUSB1B で通知される BKO1DT 割り込み

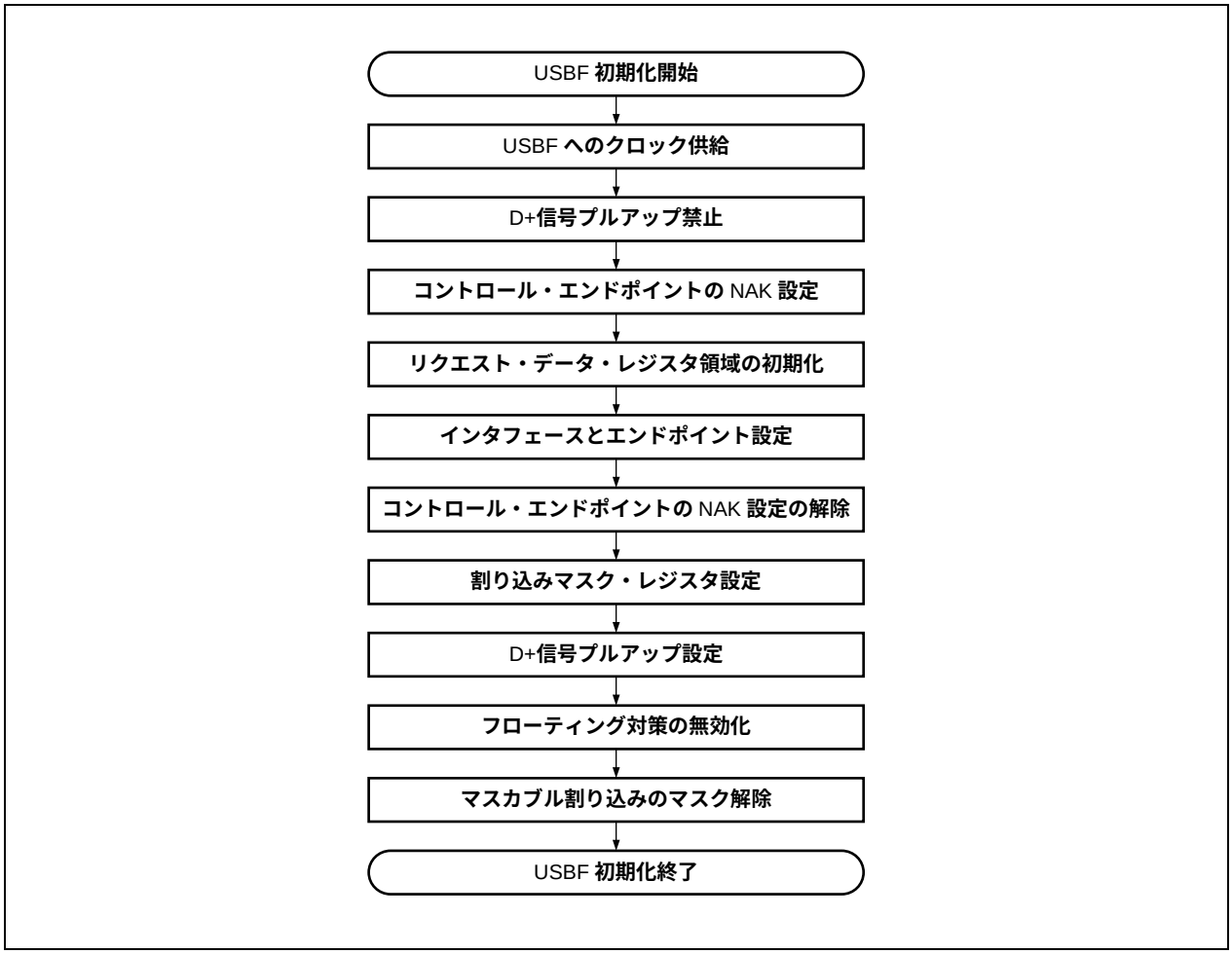
- USB 用汎用関数

  USB 制御処理で使用する汎用関数として USB データ受信／送信関数を用意しています。

3. 3. 1　USBF 初期化処理

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

図 3−5　USBF 初期化処理フロー



(1) USBF へのクロック供給

USB ファンクション・コントローラ（USBF）へのクロック供給を許可します。

- UCKC レジスタの UCKCNT ビットに "1" を書き込みます。

(2) D+信号プルアップ禁止

D+信号のプルアップをオフにすることで USB ホスト／HUB への接続通知を禁止します。

- UF0GPR レジスタの CONNECT ビットに "0" を書き込みます。

(3) コントロール・エンドポイントの NAK 設定

自動実行リクエストを含むすべてのリクエストに NAK 応答するように設定します。

- UF0E0NA レジスタの EP0NKA ビットに "1" を書き込みます。
  この設定により，自動応答リクエストで使用するデータの登録が完了するまで，ハードウエアが自動応答リクエストに対して意図しないデータを返さないようにします。

(4) リクエスト・データ・レジスタ領域の初期化

GET_DESCRIPTOR リクエストに応答するためのディスクリプタ・データなどをレジスタに登録します。

登録するデータは，デバイス・ステータス，エンドポイント 0 ステータス，デバイス・ディスクリプタ，コンフィギュレーション・ディスクリプタ，インタフェース・ディスクリプタ，およびエンドポイント・ディスクリプタです。

① UF0DSTL レジスタに "0x00" を書き込みます。

この設定により，リモート・ウエイクアップ機能の使用が禁止され，USB ファンクション・コントローラはバス・パワード・デバイスとして動作します。

② UF0EnSL レジスタ（n = 0-2）に "0x00" を書き込みます。

この設定により，Endpoint n が正常に動作していることを示します。

③ UF0DSCL レジスタに必要なディスクリプタのデータ長の合計（バイト数）を書き込みます。

この設定により，使用する UF0CIEn レジスタ（n = 0-255）の範囲が決まります。

④ UF0DDn レジスタ（n = 0-17）にデバイス・ディスクリプタのデータを書き込みます。

⑤ UF0CIEn レジスタ（n = 0-255）にコンフィギュレーション・ディスクリプタ，インタフェース・ディスクリプタ，およびエンドポイント・ディスクリプタのデータを書き込みます。

⑦ UF0MODC レジスタに "0x00" を書き込みます。

この設定により，GET_DESCRIPTOR_configuration リクエストへの自動応答を許可します。

(5) インタフェースとエンドポイントの設定

サポートするインタフェースの数，オルタナティブ設定の状態，インタフェースとエンドポイントの関係などの情報をレジスタに設定します。

① UF0AIFN レジスタに "0x00" を書き込みます。

この設定により，Interface0 のみを有効にします。

② F0AAS レジスタに "0x00" を書き込みます。

この設定により，オルタナティブ設定を無効にします。

③ UF0E1IM レジスタに "0x20"，UF0E2IM レジスタに "0x20" を書き込みます。

この設定により，Endpoint1 および Endpoint2 が Interface0 にリンクされます。

(6) コントロール・エンドポイントの NAK 設定の解除

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

- UF0E0NA レジスタの EP0NKA ビットに "0" を書き込みます。

  この設定により，自動応答リクエストを含むすべてのリクエストに対して，それぞれに応じた応答が再開されます。

(7) 割り込みマスク・レジスタの設定

USB ファンクション・コントローラの割り込みステータス・レジスタに示される割り込み要因ごとのマスクを設定します。

① UF0ICn レジスタ（n = 0-4）のすべての有効ビットに "1" を書き込みます。
この設定により，すべての割り込み要因がクリアされます。

② UF0FIC0，UF0FIC1 レジスタのすべての有効ビットに "1" を書き込みます。
この設定により，すべての転送用 FIFO がクリアされます。

③ UF0IM0 レジスタに "0x07" を書き込みます。
この設定により，UF0IS0 レジスタに示される割り込み要因のうち，RSUSPDM，BUSRSTM 割り込み以外の要因がマスクされます。

④ UF0IM1 レジスタに "0x7E" を書き込みます。
この設定により，UF0IS1 レジスタに示される割り込み要因のうち，CPUDEC 割り込み以外の要因がマスクされます。

⑤ UF0IM2 レジスタに "0x30" を書き込みます。この設定により，UF0IS2 レジスタに示される割り込み要因がすべてマスクされます。

⑥ UF0IM3 レジスタに "0x0E" を書き込みます。この設定により，UF0IS3 レジスタに示される割り込み要因のうち，BKO1DT 割り込み以外の要因がすべてマスクされます。

⑦ UF0IM4 レジスタに "0x20" を書き込みます。この設定により，UF0IS4 レジスタに示される割り込み要因がすべてマスクされます。

(8) D+信号プルアップ設定

D+信号をプルアップし，ホスト側にデバイスが接続されたことを認識させます。

- UF0GPR レジスタの CONNECT ビットに "1" を書き込みます。

(9) フローティング対策の無効化

フローティング対策を無効化します。

- UF0BC レジスタに "0x03" を書き込みます。
  この設定により，フローティング対策が無効になり，USB 用バッファが有効になります。

(10) マスカブル割り込みのマスク解除

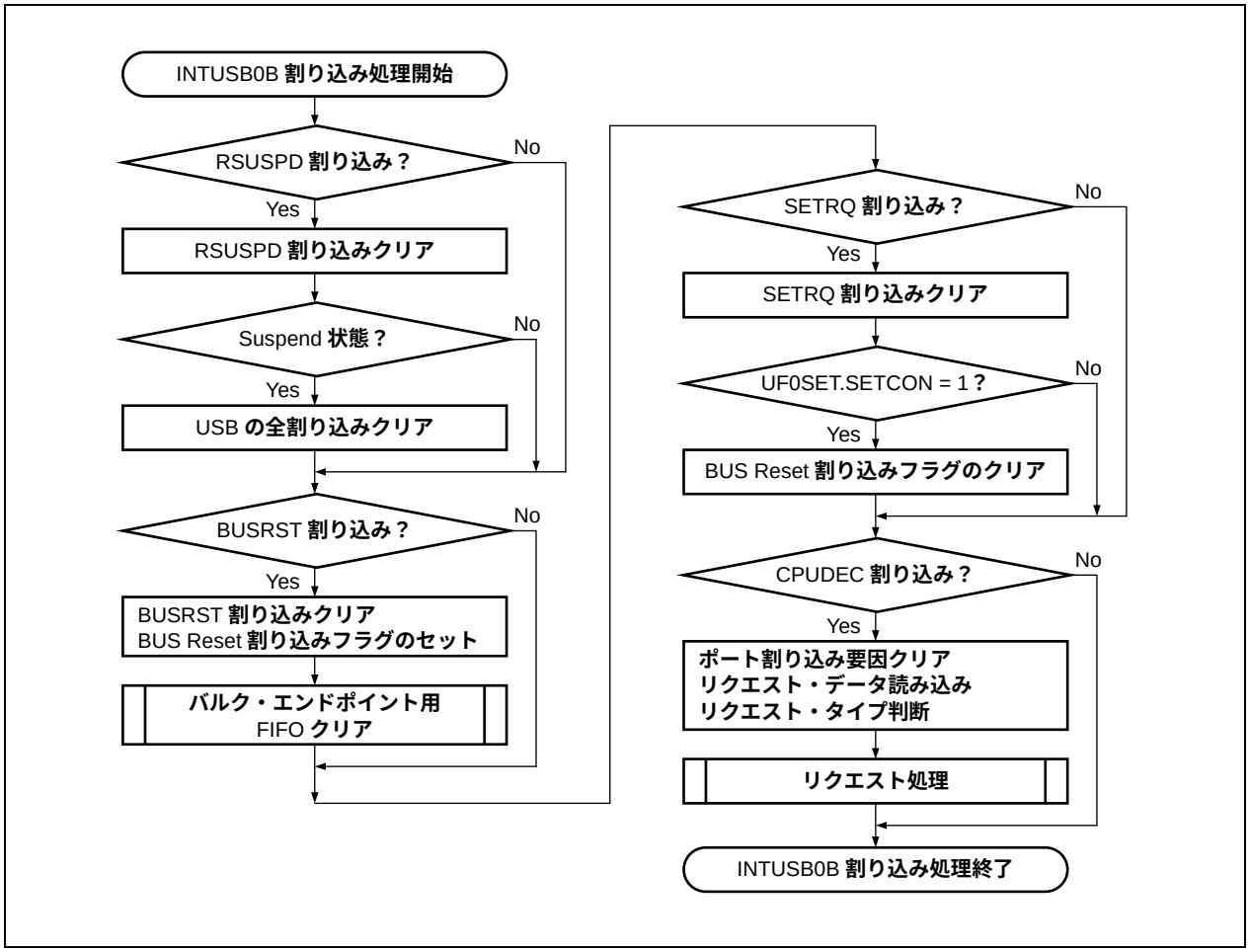
割り込み要因 INTUSB0，INTUSB1 のマスクを解除します。

- 割り込みマスク・フラグ MK0L の USBMK0，USBMK1 ビットに "0" を書き込みます。

3. 3. 2　USBF 割り込み処理（INTUSB0B）

INTUSB0B 割り込みハンドラでは，RSUSPD，BUSRST，SETRQ，CPUDEC 割り込みの処理を行います。

図 3－6　INTUSB0B 割り込みハンドラ処理フロー



（1）RSUSPD 割り込みの処理

UF0IS0 レジスタの RSUSPD ビットが "1" のとき，RSUSPD 割り込みが発生していると判断します。

RSUSPD 割り込みが発生している場合，次の処理を行います。

- 割り込み要因をクリア（UF0IC0 レジスタの RSUSPDC ビットに "0" を書き込む）
- Suspend/Resume 状態の判定

（2）Suspend 時の処理

UF0EPS1 レジスタの RSUM ビットが "1" のとき，Suspend 状態にあると判断します。

Suspend 状態の場合，USB の割り込み要因をすべてクリアします。これにより，以降の INTUSB0B 割り込み処理は省略されます。

(3) BUSRST 割り込みの処理

UF0IS0 レジスタの BUSRST ビットが "1" のとき，BUSRST 割り込みが発生していると判断します。

BUSRST 割り込みが発生している場合，次の処理を行います。

- 割り込み要因をクリア（UF0IC0 レジスタの BUSRST ビットに "0" を書き込む）
- BUS Reset 割り込みフラグ（usbf78k_busrst_flg）に "1" を設定
- バルク・エンドポイント用 FIFO クリア

(4) バルク・エンドポイント用 FIFO クリア

FIFO クリア関数（usbf78k_clearFIFO）を呼び出し，バルク・エンドポイント用 FIFO をすべてクリアします。

(5) SETRQ 割り込みの処理

UF0IS0 レジスタの SETRQ ビットが "1" のとき，割り込みが発生していると判断します。

SETRQ 割り込みが発生している場合，次の処理を行います。

- 割り込み要因をクリア（UF0IC0 レジスタの SETRQ ビットに "0" を書き込む）
- 自動応答リクエスト（SET_XXXX）の処理

(6) 自動応答リクエスト（SET_XXXX）の処理

UF0SET レジスタの SETCON ビットが "1" のとき，SET_CONFIGURATION リクエストを受信し，自動処理を行った状態にあることを判断します。

自動処理を行った場合，BUS Reset 割り込みフラグ（usbf78k_busrst_flg）を "0" にします。

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

(7) CPUDEC 割り込みの処理

UF0IS1 レジスタの CPUDEC ビットが "1" のとき，割り込みが発生していると判断します。

CPUDEC 割り込みが発生している場合，次の処理を行います。

- ポート割り込み要因クリア（UF0IC1 レジスタの PORT ビットに "0" を書き込む）
- 受信データを FIFO から読み込み，リクエスト・データを構成
- リクエスト処理

(8) リクエスト処理

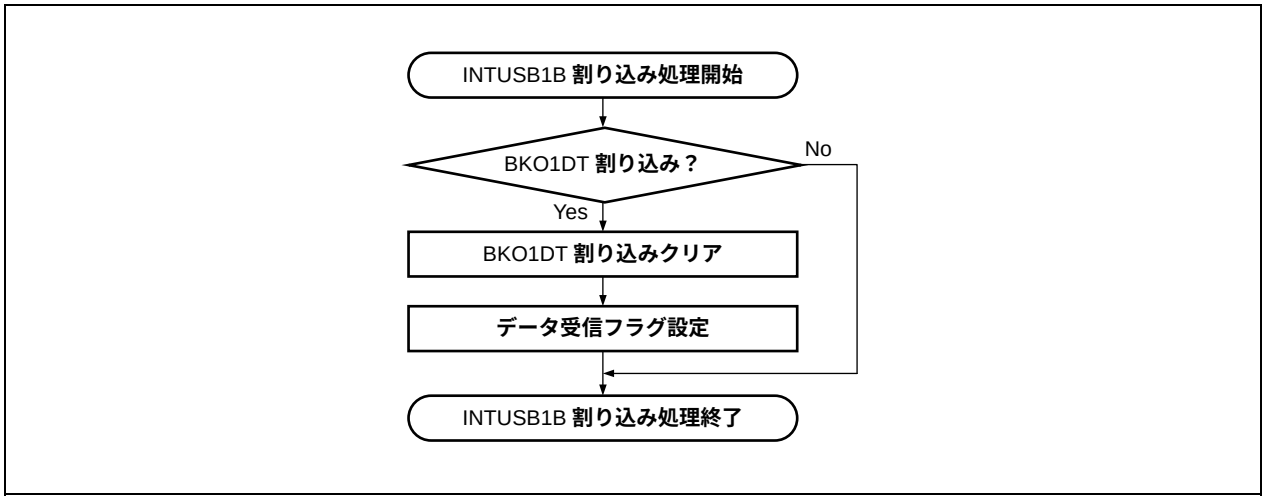
リクエスト・データが，ハードウエアで自動応答しないリクエスト（標準，クラス，ベンダ）かどうかを判断し，リクエスト・タイプに応じて，各リクエストの処理を実行します。

エンドポイント 0 は，コントロール転送用のエンドポイントです。プラグイン時のエニュメレーション処理では，ほとんどの標準デバイス・リクエストがハードウエアによって自動処理されます。ここでは，自動処理対象外の標準リクエスト，クラス・リクエストおよびベンダ・リクエストについて処理します。

3. 3. 3　USBF 受信割り込み処理（INTUSB1B）

INTUSB1B 割り込みハンドラでは，BKO1DT 割り込みの処理を行います。

図 3－7　INTUSB1B 割り込みハンドラ処理フロー



（1）BKO1DT 割り込み判断

UF0IS3 レジスタの BKO1DT ビットが "1" のとき，割り込みが発生していると判断します。

（2）BKO1DT 割り込みクリア

UF0IC3 レジスタの BKO1DTC ビットに "0" を書き込むことで割り込み要因をクリアします。

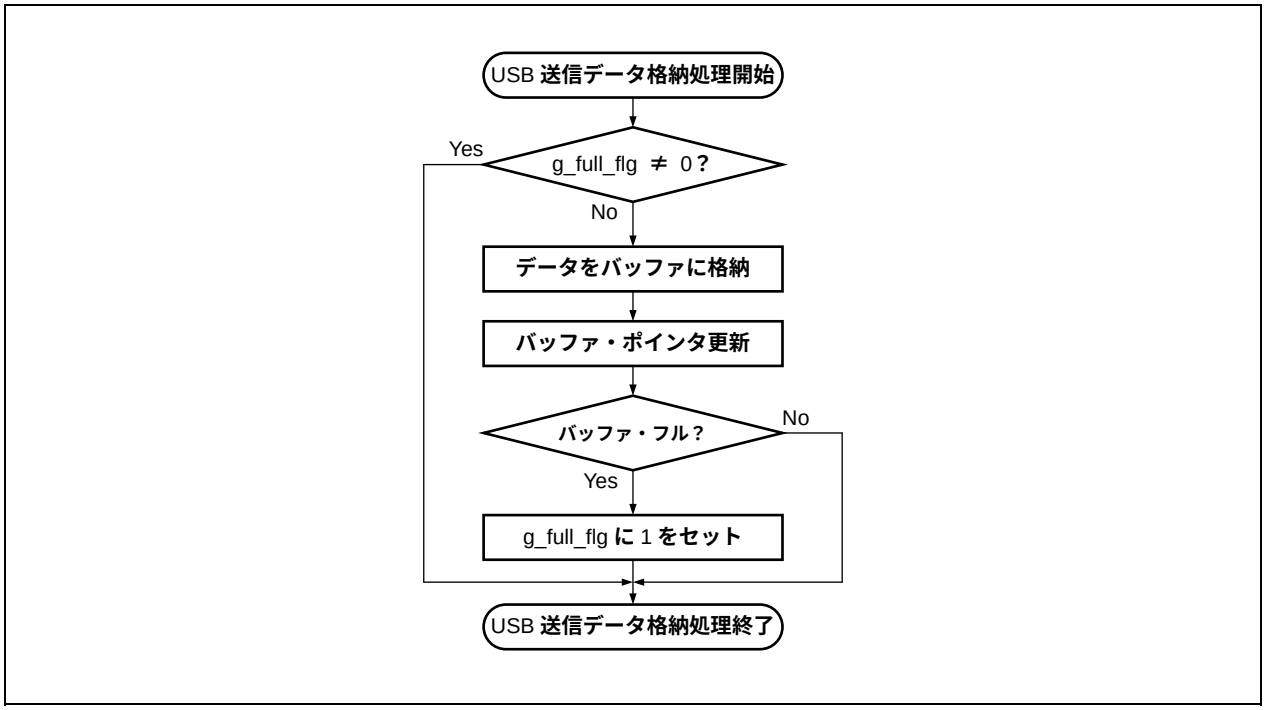
（3）データ受信フラグ設定

データ受信フラグ（usbf78k_rdata_flg）を "1" にします。

3. 3. 4　USB 送信データ格納処理

USB へ送信するデータを送信用リング・バッファへ格納します。

図 3－8　USB 送信データ格納処理フロー



（1）USB 送信データ格納バッファに空きがある場合（g_full_flg = 0）

データを USB 送信データ格納バッファに格納します。また，バッファ・ポインタを更新します。

データの格納後にバッファ・フルになった場合，バッファ・フル・フラグ（g_full_flg）を "1" にします。

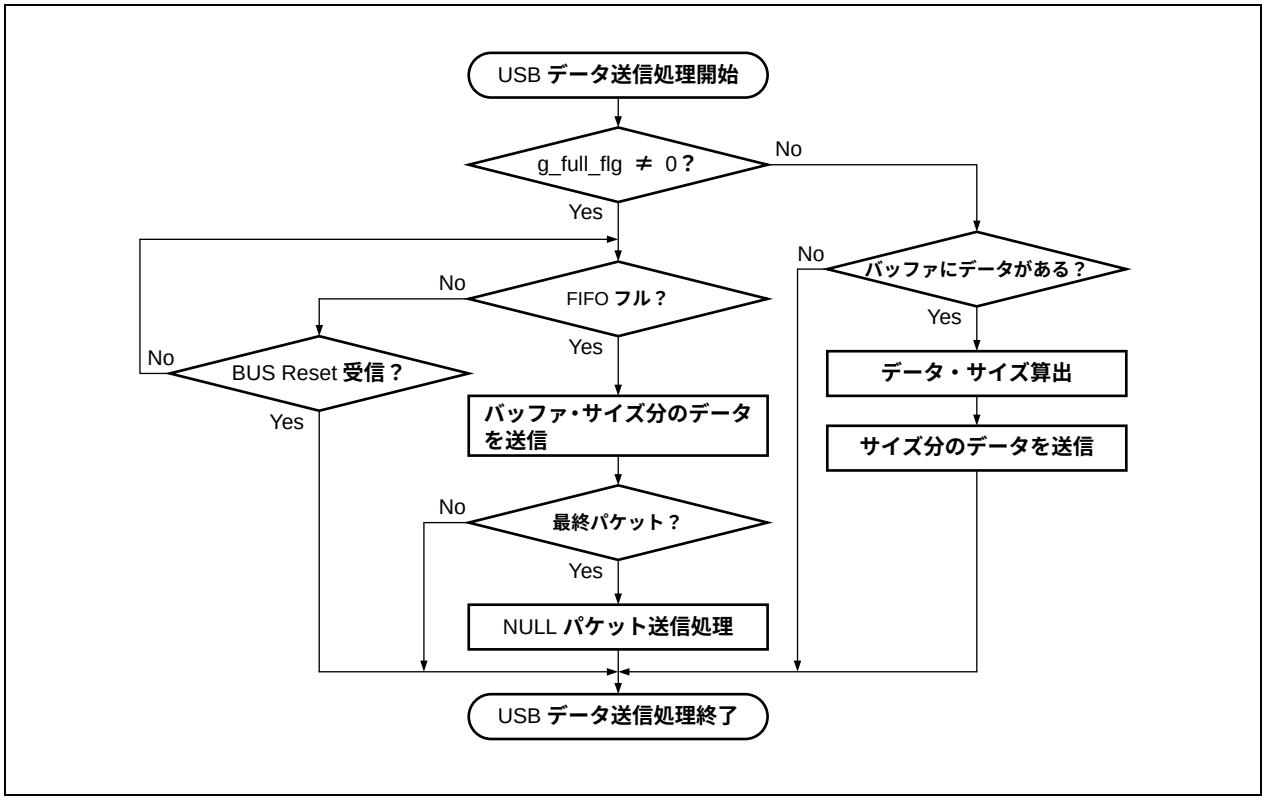
（2）USB 送信データ格納バッファに空きがない場合（g_full_flg ≠ 0）

データを格納せずに USB 送信データ格納処理を終了します。

3. 3. 5　USB データ送信処理

USB 送信用リング・バッファに格納されたデータを送信します。

図 3－9　USB データ送信処理フロー



(1) 送信用リング・バッファに空きがある場合（g_full_flg = 0）

送信用リング・バッファに格納されているデータのサイズを算出し，サイズ分のデータを送信します。

(2) 送信用リング・バッファに空きがない場合（g_full_flg ≠ 0）

送信用リング・バッファに格納されているデータをすべて送信します。

送信データが最終パケットの場合，データを送信したあとに NULL パケットを送信します。

(3) BUS Reset の場合

送信用リング・バッファに空きがなく，かつ送信用 FIFO がフルの場合のみ判定を行います。

BUS Reset 割り込みフラグ（usbf78k_busrst_flg）が "1" の場合，BUS Reset を受信していると判断します。

BUS Reset を受信している場合，データを送信せずに USB データ送信処理を終了します。

3. 4　UART 制御処理

シリアル・インタフェース（UART6）の初期化，割り込みハンドラ，データ送信処理，およびデータ受信処理を行います。

サンプル・ソフトウエアでは，μPD78F0730 内蔵の UART6 を動作させるための簡易ドライバを用意しています。

注意　μPD78F0730 内蔵の UART の処理プログラムを用意していますが，サンプル・ソフトウエア用に最低限の処理を行うものです。汎用の UART ドライバとしての動作は保障しません。

- UART の初期化

  メイン・ルーチンから呼び出し，μPD78F0730 内蔵の UART6 の初期化を行います。

- UART の割り込みハンドラ

  UART の割り込みが発生するたびに呼び出される割り込み処理専用ルーチンです。

  注意　このサンプル・ソフトウエアでは，必要な割り込み以外をマスクします。
  このサンプル・ソフトウエアで使用する割り込みは，次の 2 つです。
  ・INTSR6 で通知される受信完了割り込み
  ・INTSRE6 で通知される受信エラー割り込み

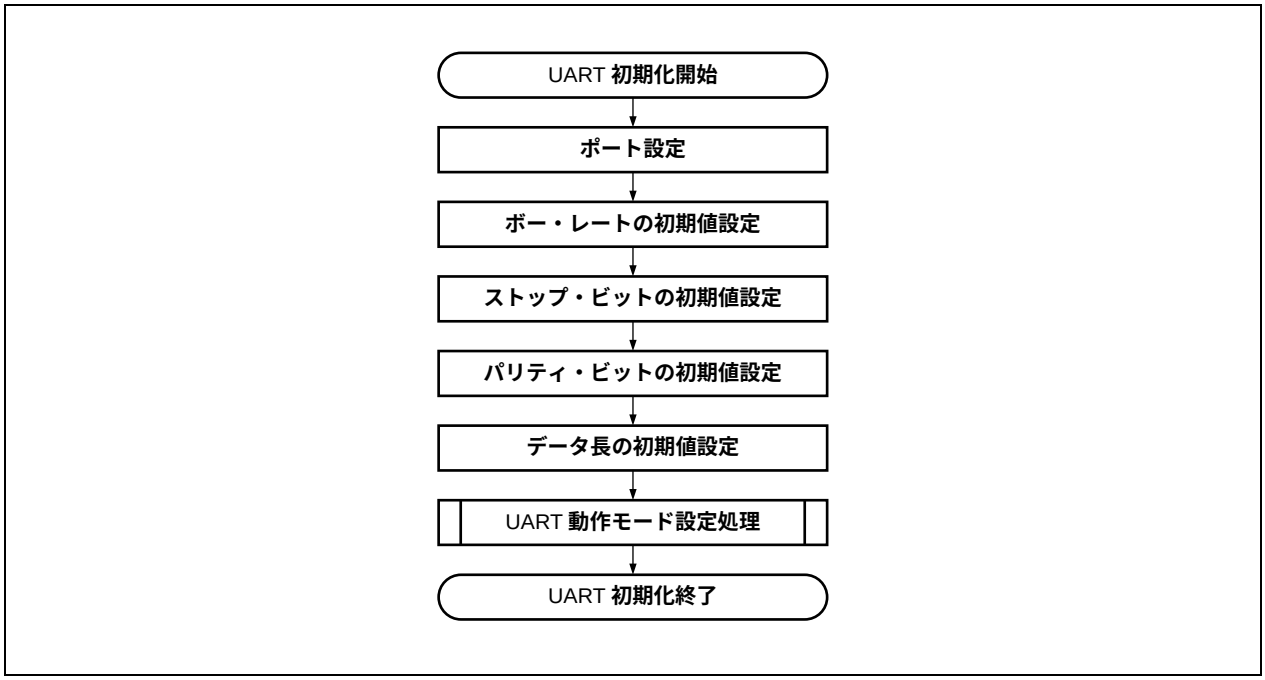
- UART 用汎用関数

  UART 処理部で使用する汎用関数として UART のデータ送信関数や，動作モードの設定関数を用意しています。

3. 4. 1　UART 初期化処理

μPD78F0730 のポートおよび UART6 の動作モードの初期値を設定します。

図 3－10　UART 初期化処理フロー



(1) ポート設定

シリアル・インタフェース UART6 のシリアル・データ入出力用ポートの設定を行います。

(2) ボー・レートの初期値設定

ボー・レートの初期設定値を UART 通信用設定値構造体（UART_MODE_TBL）に格納します。

(3) ストップ・ビットの初期値設定

ストップ・ビットの初期設定値を UART 通信用設定値構造体（UART_MODE_TBL）に格納します。

(4) パリティ・ビットの初期値設定

パリティ・ビットの初期設定値を UART 通信用設定値構造体（UART_MODE_TBL）に格納します。

(5) データ長の初期値設定

データ長の初期設定値を UART 通信用設定値構造体（UART_MODE_TBL）に格納します。

(6) UART 動作モード設定処理

UART 動作モード設定処理関数（uart78k_uartmode_set）を呼び出し，UART6 のレジスタ設定を行います。

3. 4. 2　UART 動作モード設定処理

UART 通信用設定値構造体（UART_MODE_TBL）に格納されている設定値に合わせて UART の動作モードを設定します。設定内容は「転送速度」,「パリティ・ビット」,「データ長」,「ストップ・ビット」の 4 つです。

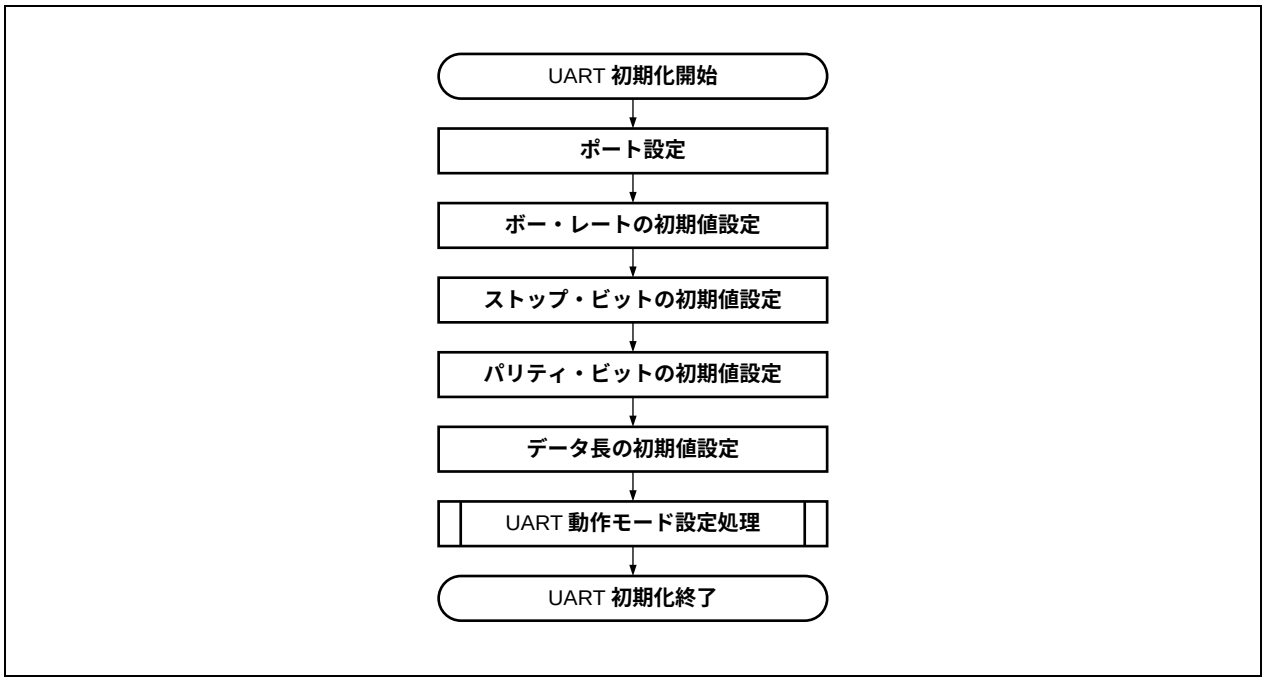
この処理は，UART 初期化処理および LINE_CONTROL リクエストの処理から呼び出されます。

UART 初期化処理では「初期設定値」を UART 通信用設定値構造体（UART_MODE_TBL）に格納し，UART 動作モード設定処理を呼び出します。

LINE_CONTROL リクエストの処理では，ホストが指定する動作モードの値を UART 通信用設定値構造体（UART_MODE_TBL）に格納し，UART 動作モード設定処理を呼び出すことで，UART の動作モードを変更します。

備考　UART 通信用設定値構造体（UART_MODE_TBL）およびサンプル・ソフトウエアで有効な UART の設定値については，**3. 4. 6　UART の動作モード**を参照してください。
LINE_CONTROL リクエストについては**3. 6. 1　LINE_CONTROL**を参照してください。

図 3－11　UART 動作モード設定処理フロー



(1) 送受信動作禁止

アシンクロナス・シリアル・インタフェース動作モード・レジスタ 6（ASIM6）の POWER6/TXE6/RXE6 に（0）をセットし，UART6 の送受信動作を禁止します。

(2) ボー・レートの設定

ボー・レートの設定値を UART 通信用設定値構造体から読み出し，クロック選択レジスタ 6（CKSR6）とボー・レート・ジェネレータ・コントロール・レジスタ 6（BRGC6）を設定します。

(3) ストップ・ビット設定

ストップ・ビットの設定値を UART 通信用設定値構造体から読み出し，アシンクロナス・シリアル・インタフェース動作モード・レジスタ 6（ASIM6）の SL6 ビットの設定を行います。

(4) パリティ・ビット設定

パリティ・ビットの設定値を UART 通信用設定値構造体から読み出し，アシンクロナス・シリアル・インタフェース動作モード・レジスタ 6（ASIM6）の PS60/PS61 ビットの設定を行います。

(5) データ長設定

データ長の設定値を UART 通信用設定値構造体から読み出し，アシンクロナス・シリアル・インタフェース動作モード・レジスタ 6（ASIM6）の CL6 ビットの設定を行います。

(6) 受信エラー割り込み信号の設定

アシンクロナス・シリアル・インタフェース動作モード・レジスタ 6（ASIM6）の ISRM6 ビットに（0）をセットし，エラー発生時の受信完了割り込み発生を許可します。

(7) 送受信動作許可

アシンクロナス・シリアル・インタフェース動作モード・レジスタ 6（ASIM6）の POWER6/TXE6/RXE6 に（1）をセットし，UART6 の送受信動作を許可します。

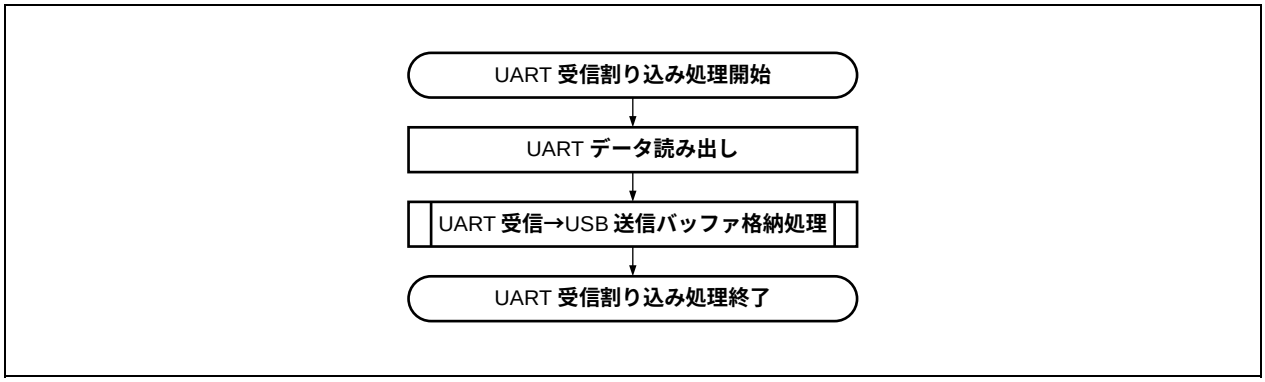
(8) 割り込み許可

割り込み要求フラグのクリアとマスク解除を行い，SRIF6/SREIF6 の割り込みを許可します。

3. 4. 3　UART 受信割り込み処理

UART 受信完了割り込み発生時は，UART データ受信処理が起動するようにベクタ登録されています。

図 3－12　UART 受信割り込み処理フロー



(1) UART データ読み出し

UART の受信バッファ・レジスタからデータを読み出します。これにより，新しいデータを受信できる状態になります。

(2) UART 受信→USB 送信バッファ格納処理

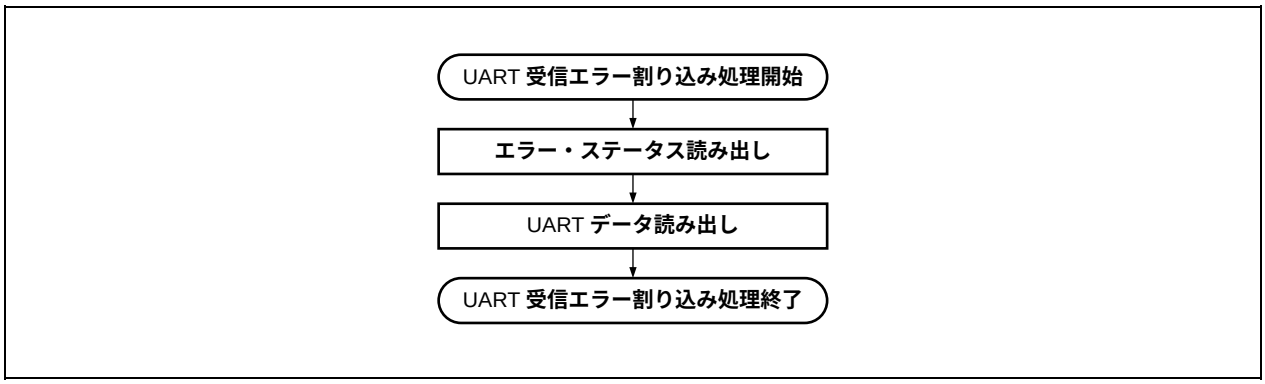
読み出したデータを USB 送信用リング・バッファに格納します。

3. 5. 1　UART 受信→USB 送信バッファ格納処理を参照してください。

3. 4. 4　UART 受信エラー割り込み処理

UART 受信時にエラーが発生した場合，エラーとなった受信データを破棄します。

図 3－13　UART 受信割り込み処理フロー



(1) エラー・ステータス読み出し

受信エラー・ステータス・レジスタを読み出します。これにより，エラー・フラグがクリアされます。

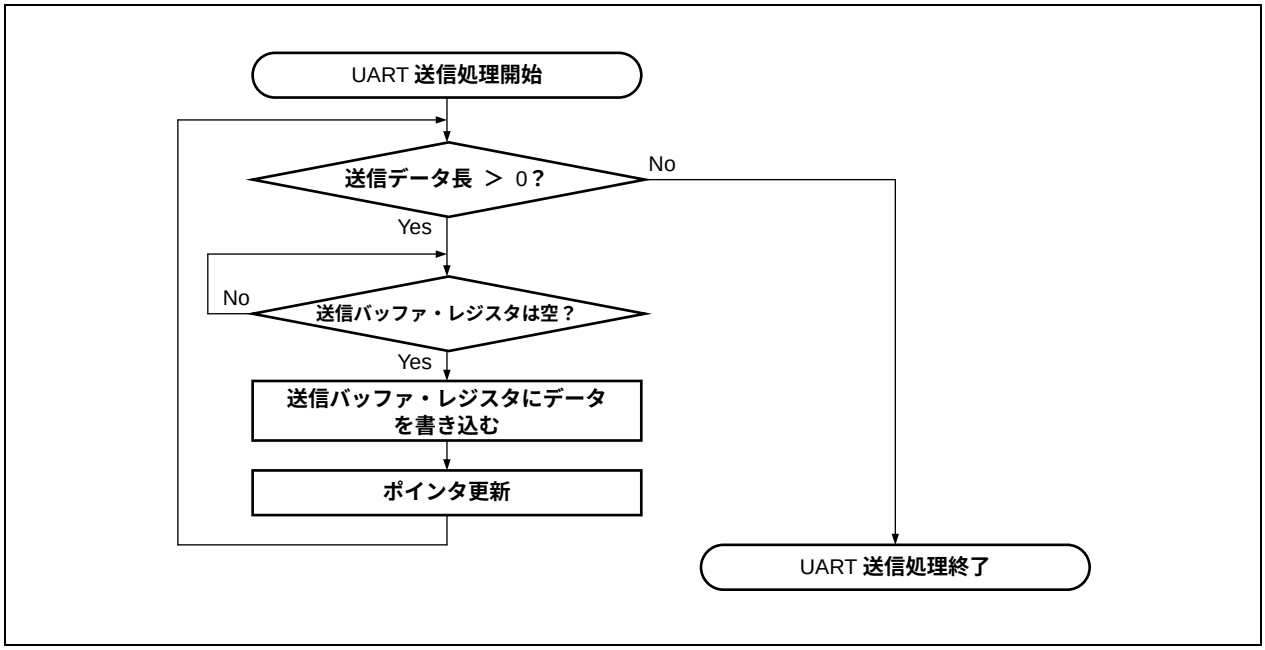
(2) UART データ読み出し（読み捨て）

UART の受信バッファ・レジスタからデータを読み出します。これにより，新しいデータを受信できる状態になります。

3. 4. 5　UART データ送信処理

UART から指定したサイズ分のデータを送信します。

図 3－14　UART データ送信処理フロー



(1) 送信データ長の確認

送信データがある間，送信処理を行います。

(2) 送信バッファ・レジスタの確認

UART の送信バッファ・レジスタに空きができるまで待ちます。

(3) データの送信

UART の送信バッファ・レジスタに送信データを書き込みます。送信バッファ・レジスタへの書き込みは，1 バイト単位で行います。

(4) ポインタ更新

送信データのポインタ，および送信データ長を更新します。

3. 4. 6　UART の動作モード

サンプル・ソフトウエアの UART 動作モード設定では，有効な設定値を限定しています。表 3－8にサンプル・ソフトウエアで有効な設定値を示します。

表 3－8　UART の設定値一覧

設定内容	設定値	備　　考
転送速度	2400 bps	－
	4800 bps	－
	9600 bps	初期設定値
	19200 bps	－
	38400 bps	－
	57600 bps	－
	76800 bps	－
	115200 bps	－
	上記以外	「9600 bps」に設定されます
パリティ・ビット	なし	初期設定値
	奇数	－
	偶数	－
	（スペース）	ホスト・ドライバ側に該当機能なし
	上記以外	「パリティなし」として設定されます
データ長	7 ビット	－
	8 ビット	初期設定値
	上記以外	「8 ビット」に設定されます
ストップ・ビット	1 ビット	初期設定値
	2 ビット	－
	上記以外	「1 ビット」に設定されます

UART の動作モードを UART 通信用設定値構造体（UART_MODE_TBL）に保持します。UART 通信用設定値構造体（UART_MODE_TBL）は，次のように定義されています。

リスト 3－1　UART 通信用設定値構造体（UART_MODE_TBL）

```
typedef struct UART_MODE_TBL{
   UINT8       DTERate[4];         /* transfer rate(bps) */
   UINT8       STOPBIT;            /* length of the stop bit - 0:1bit 2:2bits */
   UINT8       PARITYType;         /* parity bit - 0:None 1:Odd 2:Even 4:Space */
   UINT8       DATABits;           /* data size (number of the bits:7 or 8) */
} UART_MODE_TBL , *PUART_MODE_TBL;
```

3. 5　UART-USB 間ブリッジ処理

UART-USB 間のデータ受け渡しを行います。

3. 5. 1　UART 受信→USB 送信バッファ格納処理

UART 受信→USB 送信バッファ格納処理は，UART 受信完了割り込み処理から呼び出されます。

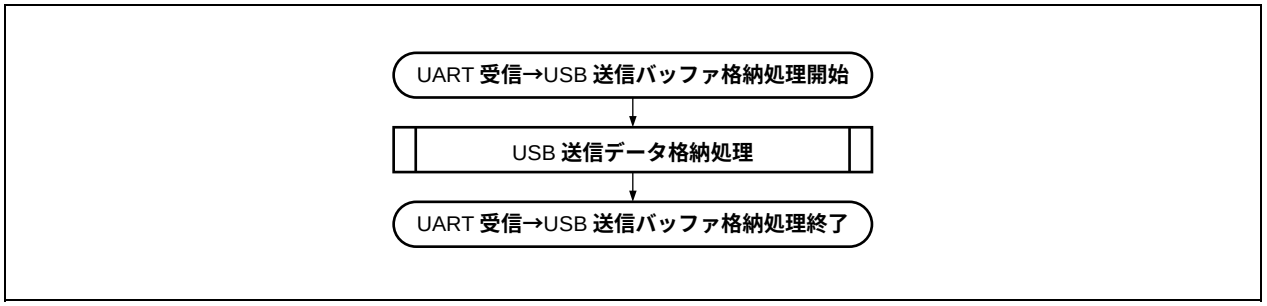
この処理では，UART で受信したデータを USB の送信用リング・バッファに格納します。

UART 受信完了割り込み処理については，3. 4. 3　UART 受信割り込み処理を参照してください。

送信用リング・バッファ内のデータを送信する処理については，3. 3. 5　USB データ送信処理を参照してください。

USB 送信データ格納処理については，3. 3. 4　USB 送信データ格納処理を参照してください。

図 3－15　UART 受信→USB 送信バッファ格納処理フロー



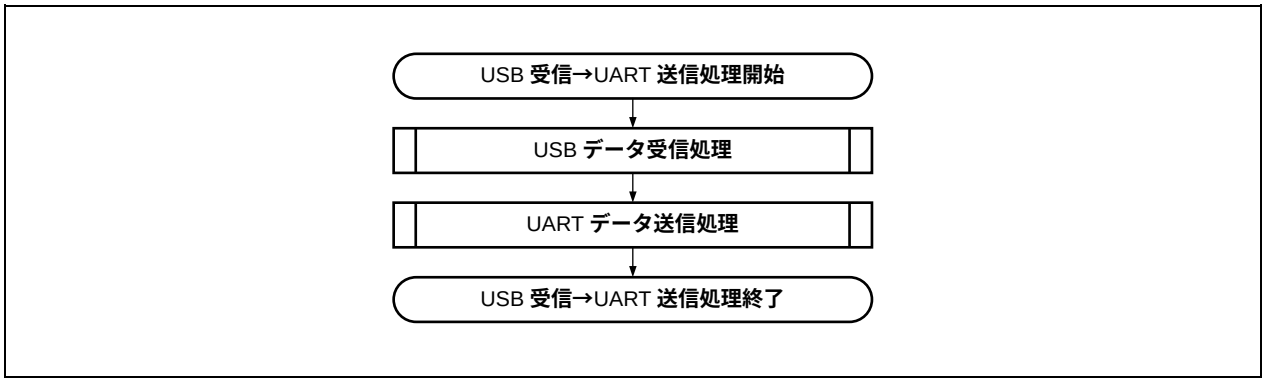
3. 5. 2　USB 受信→UART 送信処理

USB 受信→UART 送信処理では，バルク・アウト転送（受信）用エンドポイント（FIFO）にあるデータを読み出し，UART から送信します。

USB データ受信処理では，FIFO にあるデータを読み出し，受信データ格納バッファに格納します。

UART データ送信処理では，受信データ格納バッファ内のデータを 1 バイトずつ UART から送信します。詳細は3. 4. 5　UART データ送信処理を参照してください。

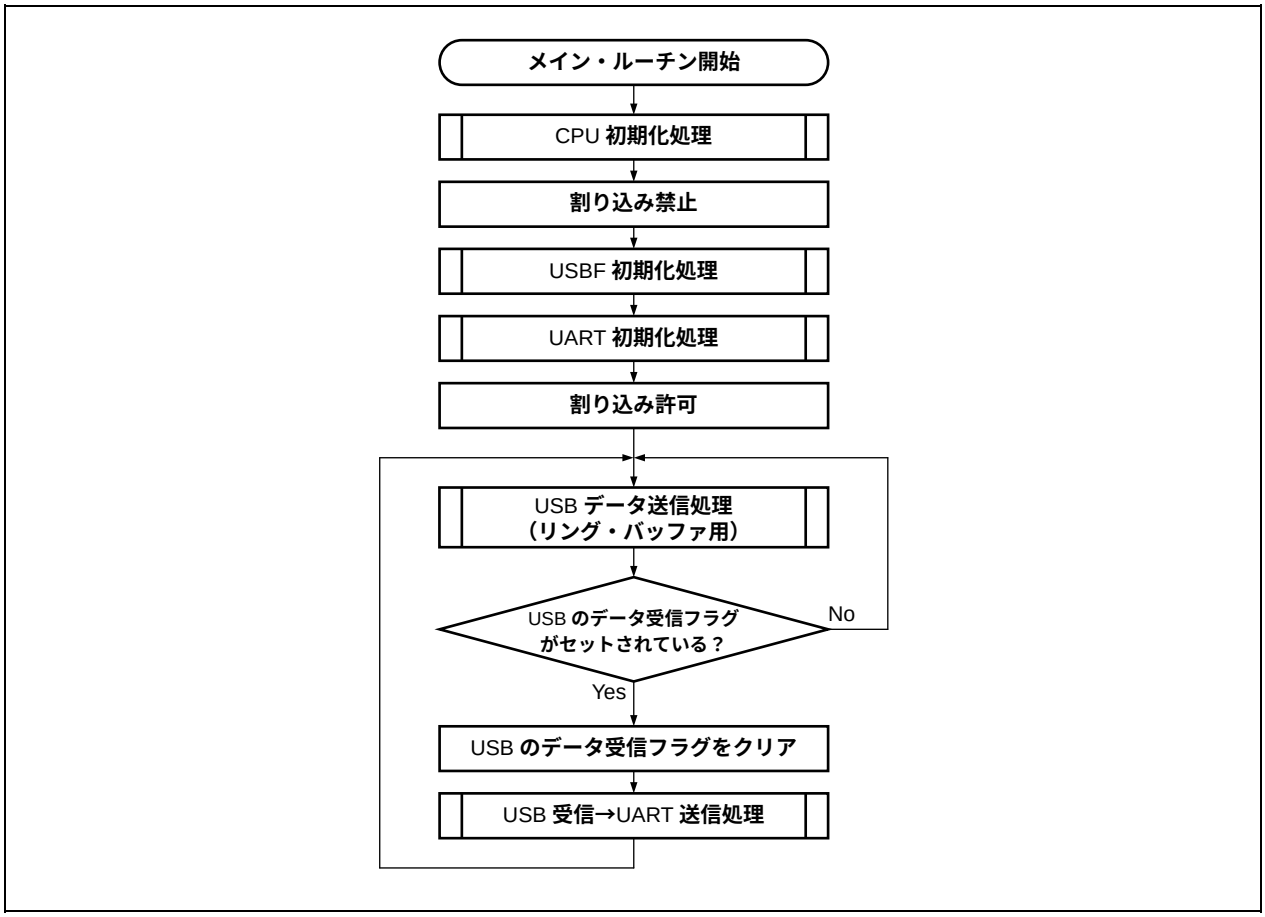
図 3－16　USB 受信→UART 送信処理フロー



3. 5. 3　メイン・ルーチン

メイン・ルーチンは，USB ファンクション・コントローラ（USBF）で受信したデータをシリアル・インタフェース（UART6）から送信する処理，およびシリアル・インタフェースで受信したデータを USB ファンクション・コントローラから送信する処理を行います。

図 3－17　メイン・ルーチンの処理フロー



（1）USB データ送信処理

リング・バッファ用 USB データ送信処理関数（usbf78k_send_txbuf）を呼び出し，USB 送信用リング・バッファに格納されているデータを送信します。

（2）USB データ受信の判定

サンプル・ソフトウエアにより設定されるデータ受信フラグ（usbf78k_rdata_flg）を監視します。このフラグがセット（1）された場合，USB ファンクション・コントローラ内に受信データがあることを示します。

受信データがある場合，データ受信フラグ（usbf78k_rdata_flg）をクリア（0）したあと，USB 受信→UART 送信処理を実行します。

（3）USB 受信→UART 送信処理

USB 受信→UART 送信処理関数（usbf78k_usb_to_uart）を呼び出し，USB の受信 FIFO にあるデータを UART から送信します。

3. 6　ベンダ・リクエストのフォーマット

サンプル・ソフトウエアには，次の 5 種類のリクエストへの応答処理が実装されています。
ここでは各ベンダ・リクエストのフォーマットについて説明します。

3. 6. 1　LINE_CONTROL

ホストがボー・レート，フロー制御，パリティ・ビット，データ・サイズの設定をデバイスに通知するためのリクエストです。

表 3－9　LINE_CONTROL リクエストのフォーマット

（a）リクエスト・コード

フィールド	サイズ	設定値
bmRequestType	1	**リクエスト・タイプ：**0x40
bRequest	1	**リクエスト識別子：**0x00
wValue	2	**未使用：**0x0000
wIndex	2	**未使用：**0x0000
wLength	2	**データ長：**0x0006（データ・ステージのバイト数）

（b）データ

フィールド	サイズ	設定値
bRequest	1	**リクエスト識別子：**0x00（LINE_CONTROL）
bBaud	4	**ボー・レート注：** 0x00000960（2400 bps） 0x000012c0（4800bps） 0x00002580（9600bps） 0x00004b00（19200bps） 0x00009600（38400bps） 0x0000e100（57600bps） 0x00012c00（76800bps） 0x0001c200（115200bps）
bParams	1	D7-D6（予約）：　00 D5-D4（フロー制御）：　00（なし） 01（ハードウエア（RTS/CTS）） 10（ソフトウエア（Xon/Xoff）） D3-D2（パリティ）：　00（なし） 01（偶数） 10（奇数） D1（ストップ・ビット）：　0（1 ビット） 1（2 ビット） D0（データ・サイズ）：　0（7 ビット） 1（8 ビット）

注　bBaud には任意の値を設定できますが，実際に動作するボー・レートはデバイスに依存します。

3. 6. 2　SET_DTR_RTS

ホストが DTR/RTS 設定の ON/OFF 切り替えを通知するためのリクエストです。

表 3－10　SET_DTR_RTS リクエストのフォーマット

（a）リクエスト・コード

フィールド	サイズ	設定値
bmRequestType	1	リクエスト・タイプ：0x40
bRequest	1	リクエスト識別子：0x00
wValue	2	未使用：0x0000
wIndex	2	未使用：0x0000
wLength	2	データ長：0x0002（データ・ステージのバイト数）

（b）データ

フィールド	サイズ	設定値
bRequest	1	リクエスト識別子：0x01（SET_DTR_RTS）
bParams	1	D7-D2（予約）： 000000 D1（DTR）： 0（OFF） 1（ON） D0（RTS）： 0（OFF） 1（ON）

3. 6. 3　SET_XON_XOFF_CHR

ホストが Xon/Xoff キャラクタ・コード設定を通知するためのリクエストです。

表 3－11　SET_XON_XOFF_CHR リクエストのフォーマット

（a）リクエスト・コード

フィールド	サイズ	設定値
bmRequestType	1	リクエスト・タイプ：0x40
bRequest	1	リクエスト識別子：0x00
wValue	2	未使用：0x0000
wIndex	2	未使用：0x0000
wLength	2	データ長：0x0003（データ・ステージのバイト数）

（b）データ

フィールド	サイズ	設定値
bRequest	1	リクエスト識別子：0x02（SET_XON_XOFF_CHR）
XonChr	1	Xon キャラクタ・コード
XoffChr	1	Xoff キャラクタ・コード

3. 6. 4　OPEN_CLOSE

ホストがポートのオープン／クローズ状態を通知するためのリクエストです。

表 3－12　OPEN_CLOSE リクエストのフォーマット

（a）リクエスト・コード

フィールド	サイズ	設定値
bmRequestType	1	リクエスト・タイプ：0x40
bRequest	1	リクエスト識別子：0x00
wValue	2	未使用：0x0000
wIndex	2	未使用：0x0000
wLength	2	データ長：0x0002（データ・ステージのバイト数）

（b）データ

フィールド	サイズ	設定値
bRequest	1	リクエスト識別子：0x03（OPEN_CLOSE）
bOpen	1	ポートの状態：0x00（ポートがクローズされた。） 0x01（ポートがオープンされた。）

3. 6. 5　SET_ERR_CHR

ホストがエラー発生時の置き換えキャラクタ・コードの設定を通知するためのリクエストです。

表 3－13　SET_ERR_CHR リクエストのフォーマット

（a）リクエスト・コード

フィールド	サイズ	設定値
bmRequestType	1	リクエスト・タイプ：0x40
bRequest	1	リクエスト識別子：0x00
wValue	2	未使用：0x0000
wIndex	2	未使用：0x0000
wLength	2	データ長：0x0003（データ・ステージのバイト数）

（b）データ

フィールド	サイズ	設定値
bRequest	1	リクエスト識別子：0x04（SET_ERR_CHR）
bOpen	1	ポートの状態：0x00（ErrChr の置き換え無効） 0x01（ErrChr の置き換え有効）
ErrChr	1	パリティ・エラーなどが発生したときの置き換えキャラクタ・コード

3. 7　関数の仕様

ここでは，サンプル・ソフトウエアに実装されている各種関数について説明します。

3. 7. 1　関数一覧

サンプル・ソフトウエアでは，ソース・ファイルそれぞれに次のような関数が実装されています。

表 3－14　サンプル・ソフトウエア内の関数

ソース・ファイル	関数名	説　　明
main.c	main	メイン・ルーチン
	cpu_init	CPU 初期化
usbf78k.c	usbf78k_init	USB ファンクション・コントローラの初期化
	usbf78k_standardreq	標準リクエストの処理
	usbf78k_getdesc	GET_DISCRIPTOR リクエストの処理
	usbf78k_data_send	USB ホストへデータ送信
	usbf78k_rdata_length	USB 受信データ長の取得
	usbf78k_data_receive	USB ホストからデータ受信
	usbf78k_clearFIFO	エンドポイント（FIFO）データのクリア
	usbf78k_sendnullEP0	Endpoint0 の NULL パケット発行
	usbf78k_sendstallEP0	Endpoint0 の STALL 応答
	usbf78k_put_txbuf	UART 受信データを専用バッファにコピー
	usbf78k_send_txbuf	usbf78k_data_send 関数のコール
	usbf78k_intusb0b	INTUSB0B 割り込みの処理
	usbf78k_intusb1b	INTUSB1B 割り込みの処理（BULK/INTERRUPT Endpoint 使用）
usbf78k_vendor.c	usbf78k_vendorreq	ベンダ独自リクエストへの応答
	usbf78k_line_control	LINE_CONTROL リクエストへの処理応答
	usbf78k_set_dtr_rts	SET_DTR_RTS リクエストへの NULL 応答
	usbf78k_set_xon_xoff_chr	SET_XON_XOFF_CHR リクエストへの NULL 応答
	usbf78k_open_close	OPEN_CLOSE リクエストへの NULL 応答
	usbf78k_set_err_chr	SET_ERR_CHR リクエストへの NULL 応答
	usbf78k_usb_to_uart	USB ホストから UART へのデータ転送
	usbf78k_uart_to_usb	UART から USB ホストへのデータ転送
uart_ctrl.c	uart78k_init	シリアル・インタフェース UART6 の初期化
	uart78k_data_send	uart_send 関数のコール
	uart78k_uartmode_set	SET_LINE_CODING リクエストへの応答処理 （UART パラメータの設定）
	uart_send	UART 端末へデータ送信
	uart_receive	UART 端末からデータ受信（割り込み応答処理）
	uart_receive_error	UART 端末からデータ受信失敗（割り込みの応答処理）
boot.asm	－	ブート処理ルーチン

3. 7. 2　関数の相関関係

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

図 3－18　USB 割り込み処理での関数の呼び出し

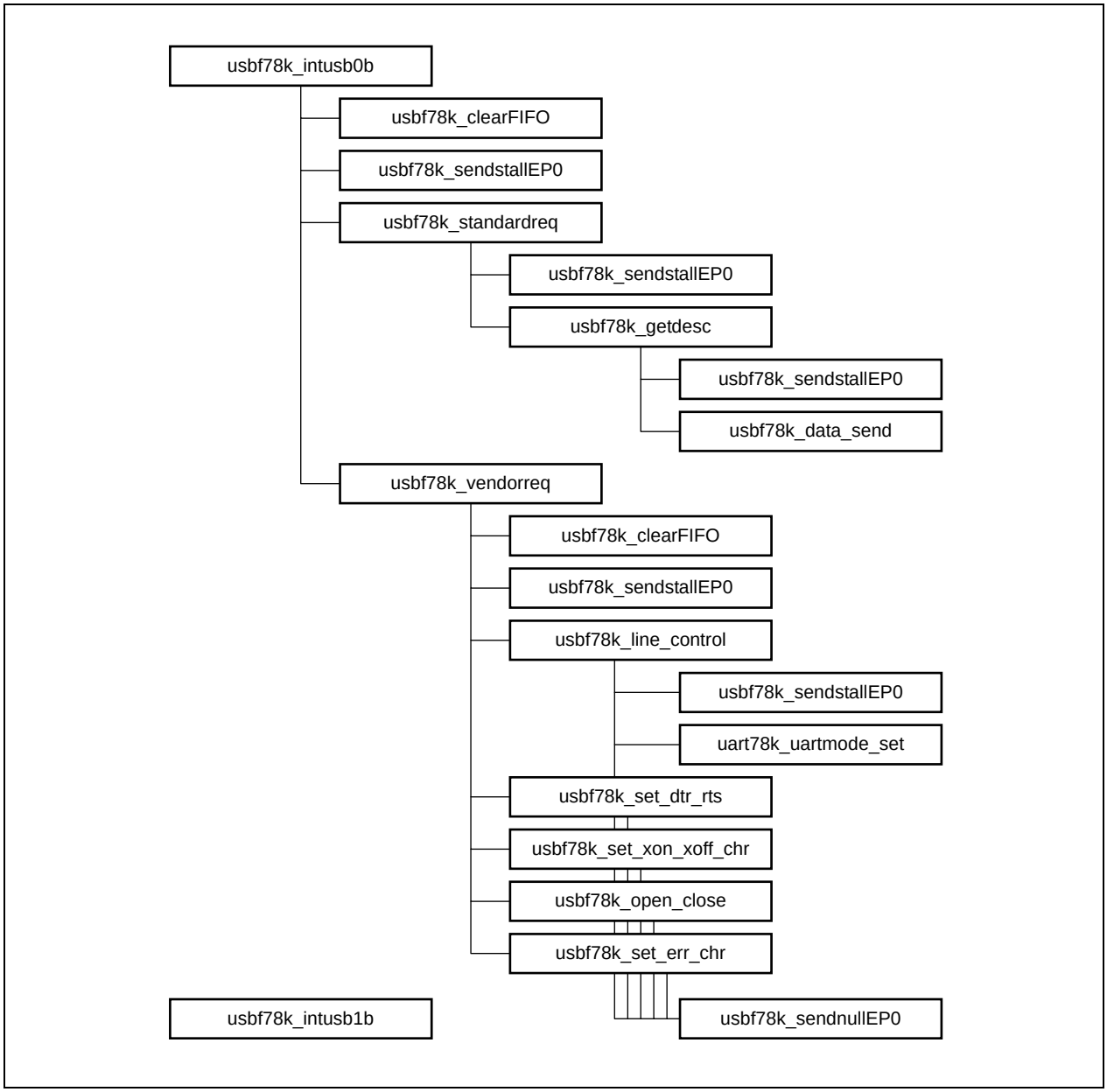


図 3－19　UART 割り込み処理での関数の呼び出し

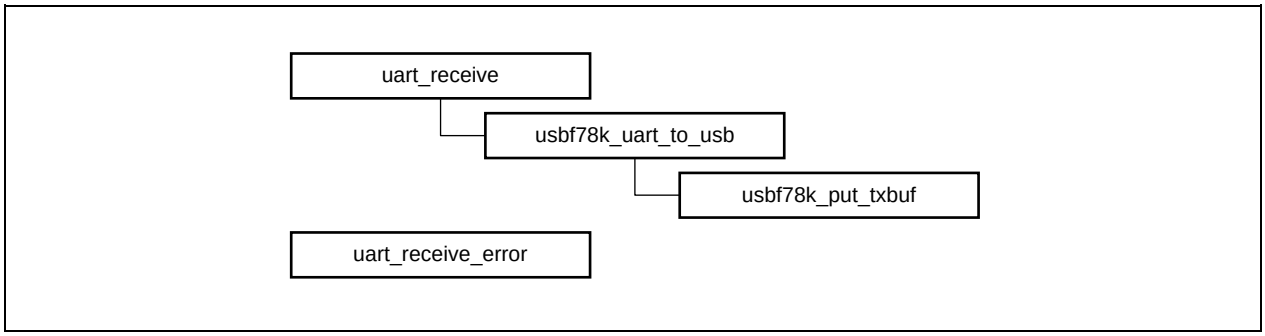
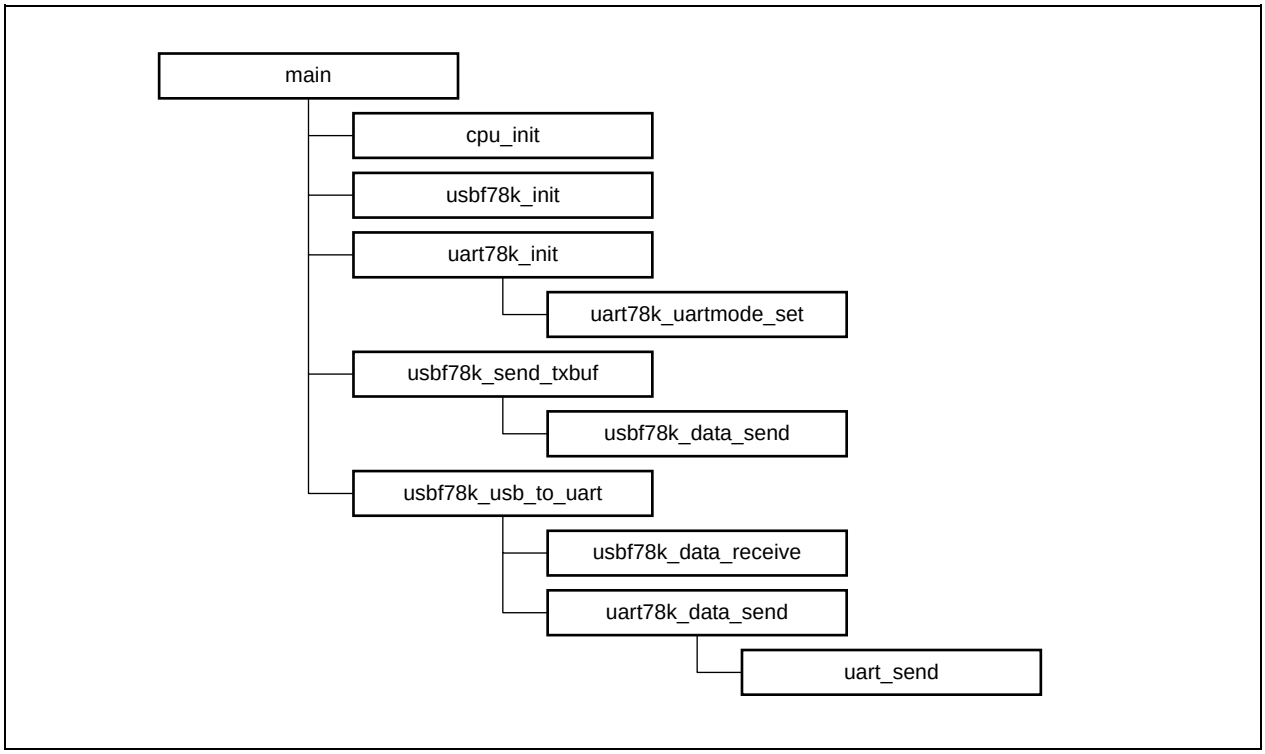


図 3－20　メイン・ルーチンでの関数の呼び出し



3. 7. 3　関数の機能

ここでは，サンプル・ソフトウエアに実装されている各種関数について解説します。

(1) 関数解説フォーマット

解説は，関数ごとに次の形式で記述されます。

関数名称

【 概　要 】

概要説明

【C 言語記述形式】

C 言語上の記述形式

【パラメータ】

パラメータ（引数）の説明

パラメータ	説　　明
パラメータ型，名称	*パラメータ概要説明*

【 戻り値 】

戻り値の説明

シンボル	説　　明
戻り値型，名称	*戻り値概要説明*

【 機　能 】

機能説明

(2) メイン処理用の関数

main

【 概　要 】
メイン処理

【C 言語記述形式】
```
void main(void)
```

【パラメータ】
なし

【 戻り値 】
なし

【 機　能 】
サンプル・ソフトウエアを実行すると最初に呼び出される関数です。
CPU 初期化処理関数（cpu_init），USBF 初期化処理関数（usbf78k_init），UART 初期化処理関数（uart78k_init）を順に呼び出して初期化を実行します。
初期化後はループ処理で次の関数を呼び出します。

- USB からデータを送信（リング・バッファ用 USB データ送信処理関数（usbf78k_send_txbuf））
- USB で受信したデータを UART から送信（USB 受信→UART 送信処理関数（usbf78k_usb_to_uart））

cpu_init

【 概　要 】
CPU 初期化処理

【C 言語記述形式】
```
void cpu_init(void)
```

【パラメータ】
なし

【 戻り値 】
なし

【 機　能 】
メイン処理で呼び出される関数です。
メモリ・サイズやクロック周波数など，μPD78F0730 を使用するために必要な項目を設定します。

(3) USB ファンクション・コントローラ用の関数

usbf78k_init

【 概　要 】

USB ファンクション・コントローラ初期化処理

【C 言語記述形式】

```
void usbf78k_usbf_init(void)
```

【パラメータ】

なし

【 戻り値 】

なし

【 機　能 】

メイン処理で呼び出される関数です。

データ領域の確保と設定，割り込み要求のマスクなど，USB ファンクション・コントローラを使用するために必要な項目を設定します。

usbf78k_intusb0b

【 概　要 】

INTUSB0B 割り込みハンドラ処理

【C 言語記述形式】

```
void usbf78k_intusb0b(void)
```

【パラメータ】

なし

【 戻り値 】

なし

【 機　能 】

INTUSB0B 割り込みが発生したときに実行する関数です。

割り込み要因が RSUSPD，BUSRST，SETRQ および CPUDEC の場合，各要求に合わせて処理を実行します。

割り込み要因が CPUDEC の場合，リクエスト・データ（8 バイト）を取り込んでデコードします。デコードの結果からリクエスト・タイプを判別し，該当する関数を呼び出して応答処理を行います。

usbf78k_intusb1b

【 概　要 】

INTUSB1B 割り込みハンドラ処理

【C 言語記述形式】

```
void usbf78k_intusb1b(void)
```

【パラメータ】

なし

【 戻り値 】

なし

【 機　能 】

INTUSB1B 割り込みが発生したときに実行する関数です。

割り込み要因が BKO1DT の場合，データ受信フラグ（usbf78k_rdata_flg）をセット（1）します。

usbf78k_data_send

【 概　要 】

USB データ送信処理

【C 言語記述形式】

```
INT32 usbf78k_data_send(UINT8 *data, INT32 len, INT8 ep)
```

【パラメータ】

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

【 戻り値 】

シンボル	説　明
DEV_OK	正常終了
DEV_ERROR	異常終了

【 機　能 】

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

usbf78k_rdata_length

【 概　要 】

USB 受信データ長取得

【C 言語記述形式】

```
void usbf78k_rdata_length(INT32 *len , INT8 ep)
```

【パラメータ】

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

【 戻り値 】

なし

【 機　能 】

指定したエンドポイントの受信データ長を読み出します。

usbf78k_data_receive

【 概　要 】

USB データ受信処理

【C 言語記述形式】

```
INT32 usbf78k_data_receive(UINT8 *data, INT32 len, INT8 ep)
```

【パラメータ】

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

【 戻り値 】

シンボル	説　明
DEV_OK	正常終了
DEV_ERROR	異常終了

【 機　能 】

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

usbf78k_clearFIFO

【 概　要 】

FIFO クリア処理

【C 言語記述形式】

```
void usbf78k_clearFIFO(INT8 ep)
```

【パラメータ】

パラメータ	説　　明
INT8 ep	エンドポイント番号

【 戻り値 】

なし

【 機　能 】

指定したエンドポイントのすべての FIFO をクリアします。

usbf78k_sendnullEP0

【 概　要 】

Endpoint0 用 NULL パケット送信処理

【C 言語記述形式】

```
void usbf78k_sendnullEP0(void)
```

【パラメータ】

なし

【 戻り値 】

なし

【 機　能 】

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

usbf78k_sendstallEP0

【 概　要 】

Endpoint0 用 STALL 応答処理

【C 言語記述形式】

```
void usbf78k_sendstallEP0(void)
```

【パラメータ】

なし

【 戻り値 】

なし

【 機　能 】

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

usbf78k_standardreq

【 概　要 】

USB ファンクション・コントローラが自動応答しない標準リクエストの処理

【C 言語記述形式】

```
void usbf78k_standardreq(void)
```

【パラメータ】

なし

【 戻り値 】

なし

【 機　能 】

USB 割り込み処理（INTUSB0B）から呼び出される関数です。

デコードされたリクエストが GET_DESCRIPTOR の場合，GET_DESCRIPTOR リクエスト処理関数（usbf78k_getdesc）を呼び出します。それ以外のリクエストの場合は Endpoint0 用 STALL 応答処理関数（usbf78k_sendstallEP0）を呼び出します。

usbf78k_getdesc

【 概　要 】

GET_DESCRIPTOR リクエスト処理

【C 言語記述形式】

```
void usbf78k_getdesc(void)
```

【パラメータ】

なし

【 戻り値 】

なし

【 機　能 】

USB ファンクション・コントローラが自動応答しない標準リクエストの処理で呼び出される関数です。
デコードされたリクエストがストリング・ディスクリプタを要求している場合，USB データ送信処理関数（usbf78k_data_send）を呼び出して，Endpoint0 からストリング・ディスクリプタを送信します。それ以外のディスクリプタを要求している場合は Endpoint0 用 STALL 応答処理関数（usbf78k_sendstallEP0）を呼び出します。

usbf78k_put_txbuf

【 概　要 】

送信用バッファへのデータ格納処理

【C 言語記述形式】

```
void usbf78k_put_txbuf(UINT8 *data)
```

【パラメータ】

パラメータ	説　　明
UINT8 *data	格納するデータのポインタ

【 戻り値 】

なし

【 機　能 】

UART 受信→USB 送信バッファ格納処理関数（usbf78k_uart_to_usb）から呼び出される関数です。
UART の受信データを USB 送信用リング・バッファに格納します。

usbf78k_send_txbuf

【 概　要 】

USB データ送信処理（リング･バッファ用）

【C 言語記述形式】

```
INT32 usbf78k_send_txbuf(void)
```

【パラメータ】

なし

【 戻り値 】

シンボル	説　　明
DEV_OK	正常終了
DEV_ERROR	異常終了

【 機　能 】

USB 送信用リング･バッファに格納されているデータを送信します。

(4) ベンダ・リクエスト処理，UART-USB 間通信用の関数

usbf78k_vendorreq

【 概　要 】

ベンダ・リクエスト処理

【C 言語記述形式】

```
void usbf78k_vendorreq(void)
```

【パラメータ】

なし

【 戻り値 】

なし

【 機　能 】

リクエスト・データ受信時に INTUSB0B 割り込みハンドラ処理関数（usbf78k_intusb0b）から呼び出されます。

ベンダ・リクエストの種類を判別し，該当処理を実行します。

usbf78k_line_contrl

【 概　要 】

LINE_CONTROL リクエスト処理

【C 言語記述形式】

```
void usbf78k_line_contrl(void)
```

【パラメータ】

なし

【 戻り値 】

なし

【 機　能 】

LINE_CONTROL リクエストのデータ・ステージで指定されている UART の動作設定パラメータを UART 通信用設定値構造体（UART_MODE_TBL）に格納します。

UART 動作モード設定処理関数（uart78k_uartmode_set）を呼び出し，UART の動作モードを変更します。

動作モードの変更が完了すると，Endpoint0 用 NULL 応答処理関数（usbf78k_sendnullEP0）を呼び出し，NULL 応答を行います。

usbf78k_set_dtr_rts

【 概　要 】

SET_DTR_RTS **リクエスト処理**

【C 言語記述形式】

```
void usbf78k_set_dtr_rts(void)
```

【パラメータ】

なし

【 戻り値 】

なし

【 機　能 】

Endpoint0 **用** NULL **応答処理関数（**usbf78k_sendnullEP0**）を呼び出します。**

サンプル・ソフトウエアでは未使用の機能のため，正常応答して終了します。

usbf78k_set_xon_xoff_chr

【 概　要 】

SET_XON_XOFF_CHR **リクエスト処理**

【C 言語記述形式】

```
void usbf78k_set_xon_xoff_chr(void)
```

【パラメータ】

なし

【 戻り値 】

なし

【 機　能 】

Endpoint0 **用** NULL **応答処理関数（**usbf78k_sendnullEP0**）を呼び出します。**

サンプル・ソフトウエアでは未使用の機能のため，正常応答して終了します。

usbf78k_open_close

【 概　要 】

OPEN_CLOSE リクエスト処理

【C 言語記述形式】

```
void usbf78k_open_close(void)
```

【パラメータ】

なし

【 戻り値 】

なし

【 機　能 】

Endpoint0 用 NULL 応答処理関数（usbf78k_sendnullEP0）を呼び出します。
サンプル・ソフトウエアでは未使用の機能のため，正常応答して終了します。

usbf78k_set_err_chr

【 概　要 】

SET_ERR_CHR リクエスト処理

【C 言語記述形式】

```
void usbf78k_set_err_chr(void)
```

【パラメータ】

なし

【 戻り値 】

なし

【 機　能 】

Endpoint0 用 NULL 応答処理関数（usbf78k_sendnullEP0）を呼び出します。
サンプル・ソフトウエアでは未使用の機能のため，正常応答して終了します。

usbf78k_usb_to_uart

【 概　要 】

USB 受信→UART 送信処理

【C 言語記述形式】

```
void usbf78k_usb_to_uart(UINT8 len)
```

【パラメータ】

シンボル	説　明
UINT8 len	データ長

【 戻り値 】

なし

【 機　能 】

メイン処理で呼び出される関数です。

USB データ受信処理関数（usbf78k_data_receive）を呼び出し，受信データをバッファに格納します。

UART データ送信処理関数（uart78k_data_send）を呼び出し，バッファに格納したデータを UART から送信します。

usbf78k_uart_to_usb

【 概　要 】

UART 受信→USB 送信バッファ格納処理

【C 言語記述形式】

```
void usbf78k_uart_to_usb(UINT8 *data)
```

【パラメータ】

シンボル	説　明
UINT8 *data	受信データ・バッファのポインタ

【 戻り値 】

なし

【 機　能 】

UART 受信完了割り込みハンドラ関数（uart_receive）から呼び出されます。

送信用バッファへのデータ格納処理関数（usbf78k_put_txbuf）を呼び出し，UART の受信データを USB 送信用データ・バッファ（リング・バッファ）に格納します。

(5) UART 用の関数

uart78k_init

【 概　要 】

UART 通信用デフォルト設定処理（UART 初期化処理）

【C 言語記述形式】

```
void uart78k_init(void)
```

【パラメータ】

なし

【 戻り値 】

なし

【 機　能 】

UART のポート，ボー・レート，STOP ビット，パリティ・ビット，データ長などの UART 通信設定を構造体に登録します。

UART 動作モード設定処理関数（uart78k_uartmode_set）を呼び出し，UART を起動します。

uart78k_uartmode_set

【 概　要 】

UART 動作モード設定処理

【C 言語記述形式】

```
void uart78k_uartmode_set(void)
```

【パラメータ】

なし

【 戻り値 】

なし

【 機　能 】

UART 通信用デフォルト設定処理関数（uart78k_init）および LINE_CONTROL リクエスト処理関数（usbf78k_line_contrl）から呼び出されます。

UART を停止し，構造体に保持している UART 通信設定を元にレジスタの設定を行います。

レジスタの設定後，UART を起動し，UART の割り込みを許可します。

uart78k_data_send

【 概　要 】

UART **データ送信処理**

【C 言語記述形式】

```
void uart78k_data_send(UINT8 *buffer, UINT32 size)
```

【パラメータ】

シンボル	説　明
UINT8 *buffer	**データ・バッファのポインタ**
UINT32 size	**データ・サイズ**

【 戻り値 】

なし

【 機　能 】

UART **送信処理関数（**uart_send**）を呼び出し，**UART **送信を実行します。**

uart_send

【 概　要 】

UART **送信処理**

【C 言語記述形式】

```
void uart_send(UINT8 *sendString, UINT32 len)
```

【パラメータ】

なし

シンボル	説　明
UINT8 *sendString	**送信データ・バッファのポインタ**
UINT32 len	**データ長**

【 戻り値 】

なし

【 機　能 】

データ長が 1 **以上の場合，**1 **バイトごとに送信処理を実行します。**

uart_receive

【 概　要 】

UART 受信完了割り込みハンドラ

【C 言語記述形式】

```
__interrupt void uart_receive(void)
```

【パラメータ】

なし

【 戻り値 】

なし

【 機　能 】

UART 受信データを読み出します。さらに UART 受信→USB 送信バッファ格納処理関数（usbf78k_uart_to_usb）を呼び出し，UART の受信データを USB 送信バッファに格納します。

uart_receive_error

【 概　要 】

UART 受信エラー用割り込みハンドラ

【C 言語記述形式】

```
__interrupt void uart_receive_error(void)
```

【パラメータ】

なし

【 戻り値 】

なし

【 機　能 】

エラー・フラグをクリアし，受信データを読み出します（読み捨てます）。

3. 8　データ構造体

サンプル・ソフトウエアでは，次の構造体を使用します。

（1）USB デバイス・リクエスト構造体

USB デバイス・リクエスト構造体は，"usbf78k.h" ファイルで定義されています。

プログラム中ではグローバル変数 "UsbSetup_Data"として使用します。

```
/*-------------------------------------------------------------
 * SETUP DATA structure
 *-------------------------------------------------------------*/
typedef struct {
     UINT8  RequstType;          /* bmRequestType  */
     UINT8  Request;             /* bRequest       */
     UINT16 Value;               /* wValue         */
     UINT16 Index;               /* wIndex         */
     UINT16 Length;              /* wLength        */
     UINT8* Data;                /* index to Data  */
} Usb_Setup_st;

/*-------------------------------------------------------------
 *   global variable
 *-------------------------------------------------------------*/
extern Usb_Setup_st    UsbSetup_Data;
```

（2）UART 通信設定構造体

UART 通信設定構造体は，"usbf78k_vendor.h" ファイルで定義されています。

```
typedef struct _UART_MODE_TBL{
     UINT8 DTERate[4];   /* transfer rate(bps) */
     UINT8 STOPBIT;      /* length of the stop bit - 0:1bit (1:1.5bits) 2:2bits */
     UINT8 PARITYType;   /* parity bit - 0:None 1:Odd 2:Even (3:Mark) 4:Space */
     UINT8 DATABits;     /* data size (number of the bits:5,6,7,8,16) */
} UART_MODE_TBL , *PUART_MODE_TBL;
```

第4章　開発環境

この章では，μPD78F0730 向け USB-シリアル変換用サンプル・ソフトウエアを利用したアプリケーション・プログラムを開発する際の環境構築の例と，そこでのデバッグの手順について説明します。

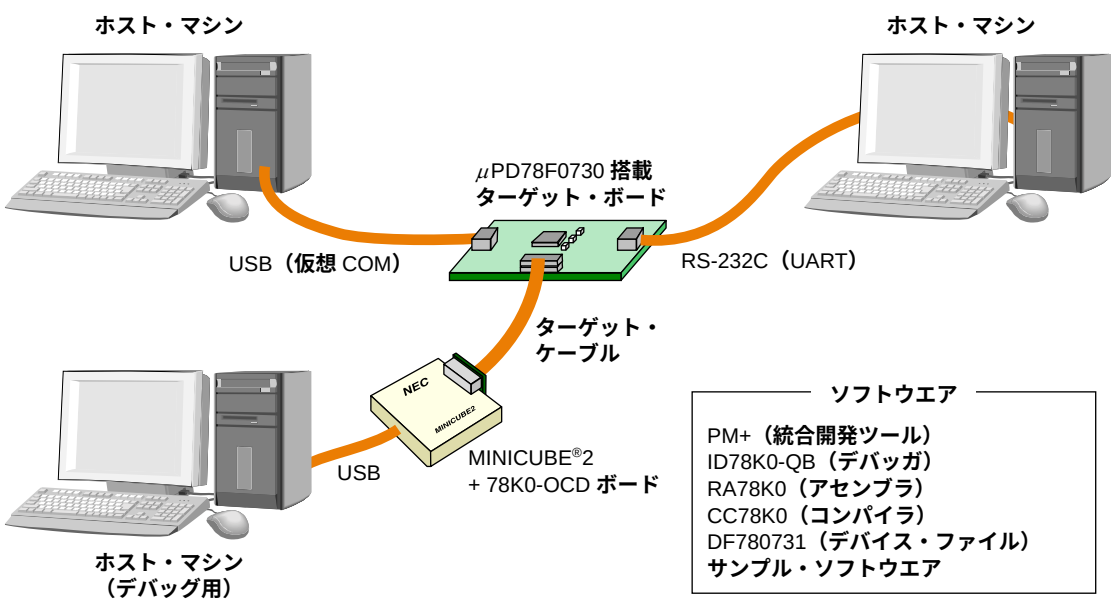
4. 1　製品構成

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

4. 1. 1　システム構成

サンプル・ソフトウエアを利用するシステムの構成を図 4－1に示します。

図 4－1　システム構成例（デバッグ時）



備考 1. デバッグ用ホスト・マシンに USB（仮想 COM），RS-232C（UART）を接続し，1 台のホスト・マシンを使ってデバッグを行うことも可能です。

2. ターゲット・ボードの外形やポートについては4. 1. 3　デバッグおよび付録 A　ターゲット・ボードを参照してください。

4. 1. 2　プログラム開発

サンプル・ソフトウエアを利用したシステムを開発する際には，次のハードウエアとソフトウエアが必要です。

表 4－1　プログラム開発環境構成例

構成品		製品例	備　　考
ハードウエア	ホスト・マシン	－	PC/AT®互換機（OS：Windows® XP または Windows Vista®）
ソフトウエア	統合開発ツール	PM+	V6.30
	アセンブラ	RA78K0	W4.00
	コンパイラ	CC78K0	W4.01
ファイル	デバイス・ファイル	DF780731	V1.10（μPD78F0730 用）
	ソース・ファイル	サンプル・ソフトウエア	
	インクルード・ファイル		

4. 1. 3　デバッグ

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

表 4－2　デバッグ環境構成例

構成品		製品例	備　　考
ハードウエア	ホスト・マシン	－	PC/AT 互換機（OS：Windows XP または Windows Vista）
	ターゲット	QB-78F0730-TB 注1	USB インタフェースと RS-232C インタフェース（UART）を持つμPD78F0730 搭載ボード
	オンチップ・デバッグ・エミュレータ	MINICUBE2	
	ケーブル類	－	USB ケーブル，RS-232C ケーブルなど
ソフトウエア	統合開発ツール	PM+	V6.30
	デバッガ	ID78K0-QB	V3.00
	ターミナル	－	注 2
ファイル	USB-シリアル変換用ホスト・ドライバ	－	サンプル・ソフトウエアに同梱
	デバイス・ファイル	DF780731	V1.10（μPD78F0730 用）
	ソース・ファイル	サンプル・ソフトウエア	
	インクルード・ファイル		
	プロジェクト関連ファイル		注 3

注 1. QB-78F0730-TB は RS-232C インタフェースを搭載していないため，別途用意して接続する必要があります。

2. ハイパーターミナル（Windows に標準で搭載されているソフトウエア）など，Windows 上で動作するターミナル・エミュレータを使用できます。

3. PM+で構築した場合のファイルがサンプル・ソフトウエアに同梱されています。

4. 2　環境設定

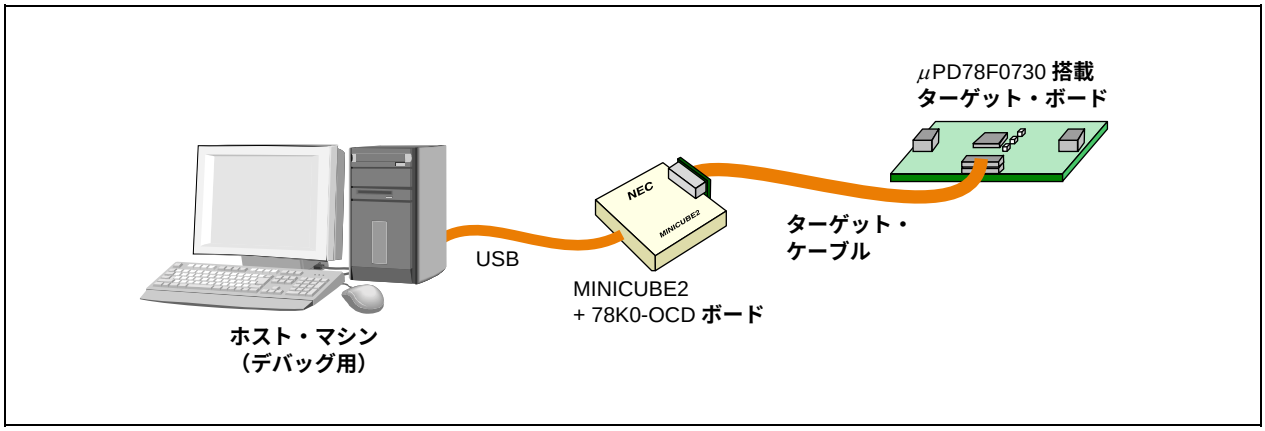
ここでは，4. 1　**製品構成**に示した製品構成で開発やデバッグを行うための準備について説明します。

4. 2. 1　ターゲット環境整備

ターゲット・ボードと MINICUBE2，MINICUBE2 とデバッグ用ホスト・マシンを接続します。

MINICUBE2 の接続方法については **QB-MINI2　ユーザーズ・マニュアル（U18371J）**を参照してください。

図 4−2　ターゲット・ボードの接続



ターゲット・ボードとして QB-78F0730-TB を使用する場合の手順を次に示します。

(1) スイッチの設定

MINICUBE2 のモード選択スイッチを M2（78K0 マイクロコントローラ），電源スイッチを 5（ターゲット・システムへ 5 V を供給）に設定します。

注意　USB ケーブル接続時に MINICUBE2 のスイッチを切り替えないでください。スイッチを切り替える場合は，USB ケーブルを取り外してから行ってください。

(2) 78K0-OCD ボードの設定

20 MHz の発振器を実装します（工場出荷時は実装されています）。

MINICUBE2 と 78K0-OCD ボードを接続します。

(3) ターゲット・ボードの接続

QB-78F0730-TB と 78K0-OCD ボードを 10 ピン・ターゲット・ケーブルで接続します。

(4) USB の接続

MINICUBE2 とデバッグ用ホスト・マシンを接続します。

4. 2. 2　ホスト環境整備

デバッグ用ホスト・マシン上に専用のワークスペースを作成します。

(1) 統合開発ツールのインストール

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

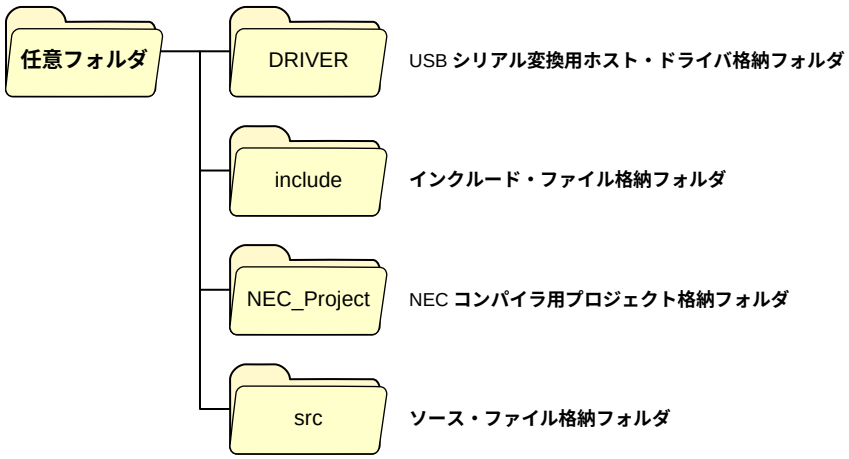
(2) デバッガのインストール

ID78K0-QB をインストールします。詳細は ID78K0-QB のユーザーズ・マニュアルを参照してください。

(3) サンプル・ソフトウエアのダウンロード

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

図 4−3　サンプル・ソフトウエアのフォルダ構成

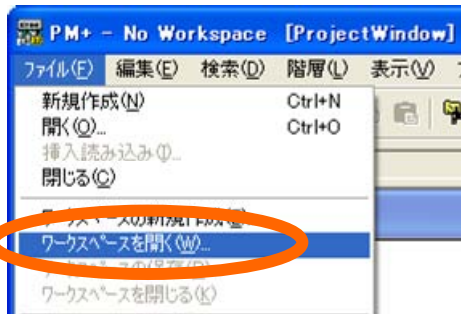


備考　デバッグ用ホスト・マシンと USB ポート接続用ホスト・マシンが異なる場合，USB シリアル変換用ホスト・ドライバ格納フォルダを USB ポート接続用ホスト・マシンの任意のディレクトリに格納します。また，RS-232C ポート用ホスト・ドライバを RS-232C ポート接続用ホスト・マシンの任意のディレクトリに格納します。

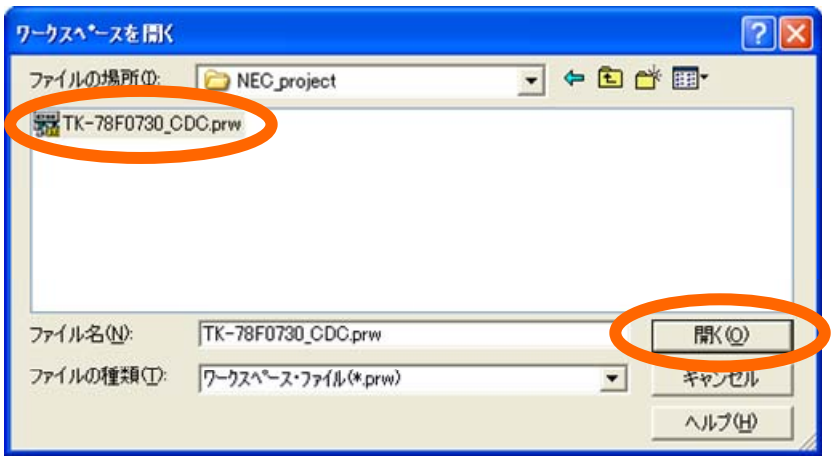
(4) ワークスペースの設定

ここではサンプル・ソフトウエアに同梱のプロジェクト関連ファイルを使用する場合の手順を示します。

<1>　PM+を起動し，「ファイル」メニューから「ワークスペースを開く」を選択します。



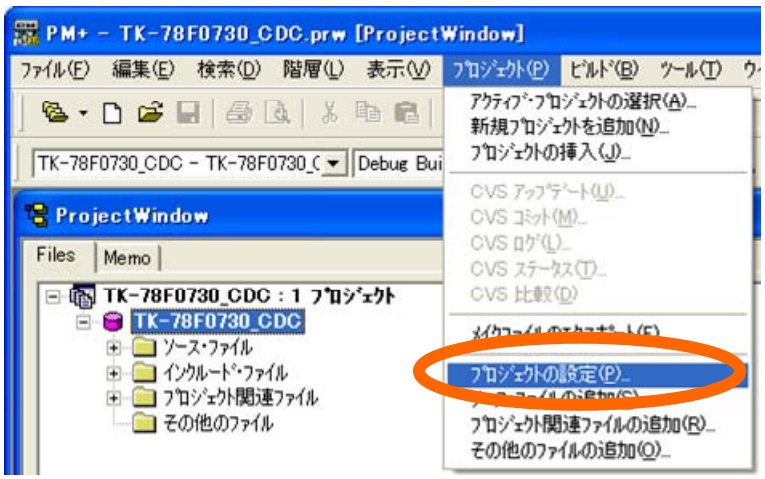
<2>　「ワークスペースを開く」ダイアログが開きます。サンプル・ソフトウエアを格納したディレクトリの「NEC_project」フォルダにあるワークスペース・ファイルを指定します。



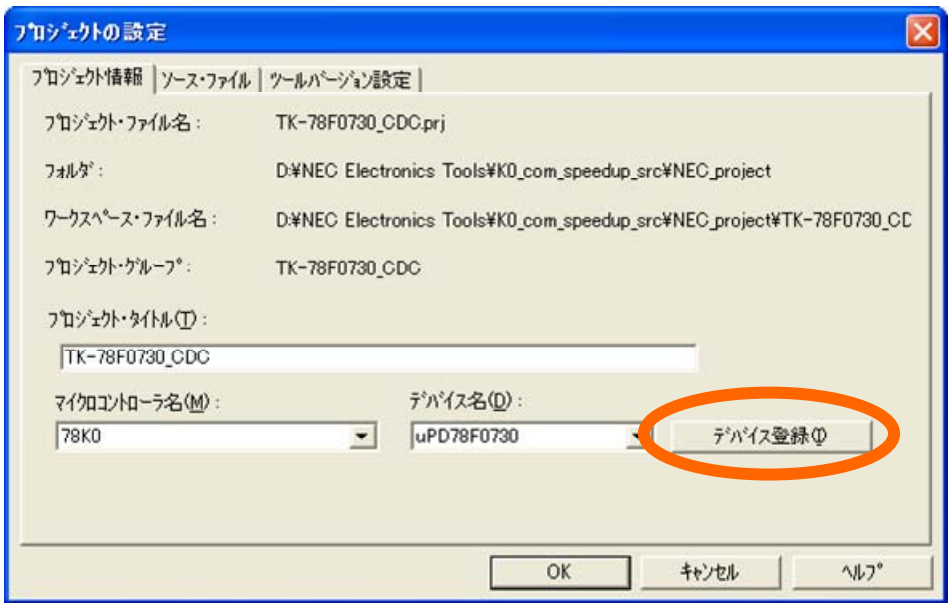
(5) デバイス・ファイルのインストール

ここではμPD78F0730 用のデバイス・ファイルを使用する場合の手順を示します。

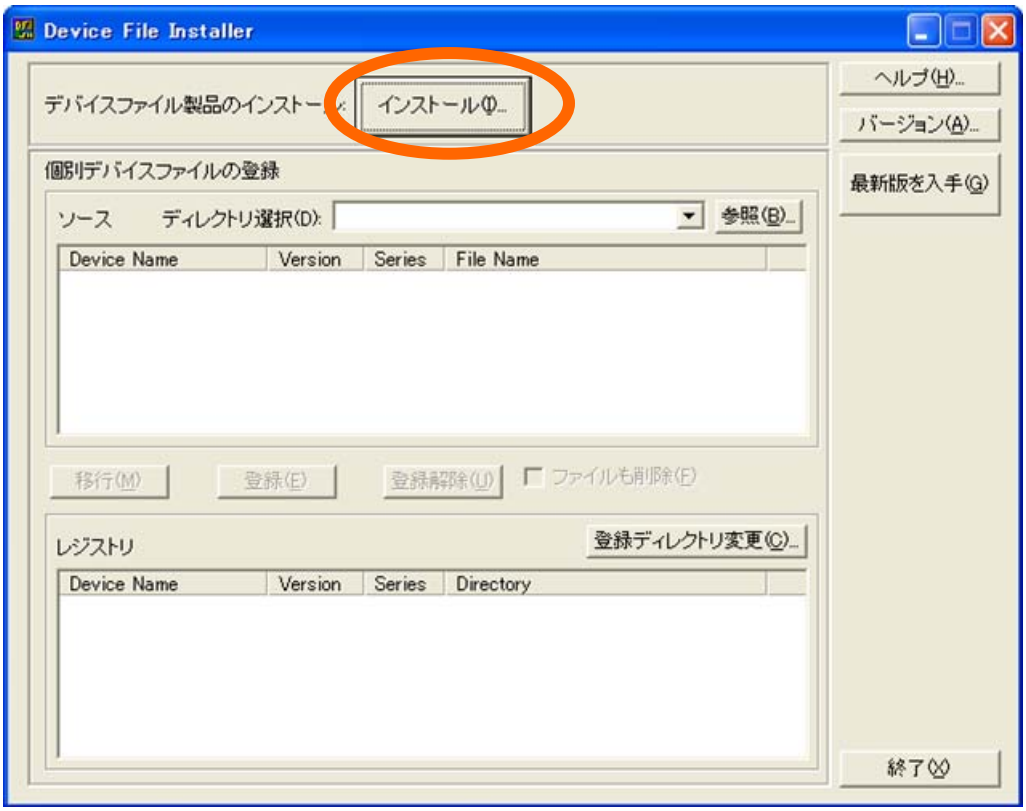
<1>　PM+の「プロジェクト」メニューから「プロジェクトの設定」を選択します。



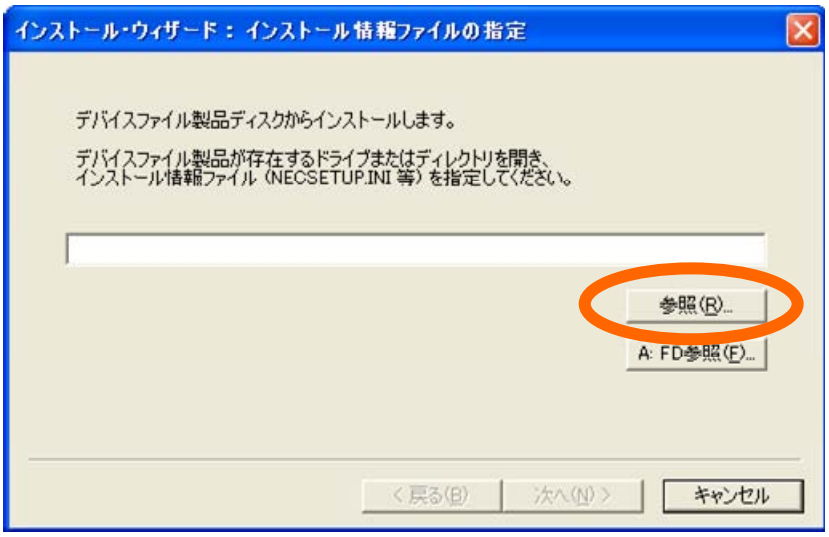
<2>　「プロジェクトの設定」ダイアログが開きます。「プロジェクト情報」タブの「デバイス登録」ボタンを押下して Device File Installer を起動します。



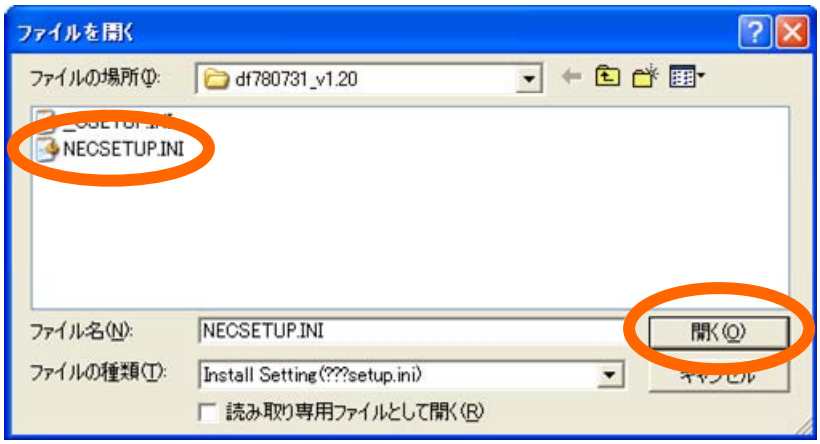
<3>　「Device File Installer」ダイアログが開きます。「インストール」ボタンを押下してインストール・ウィザードを起動します。



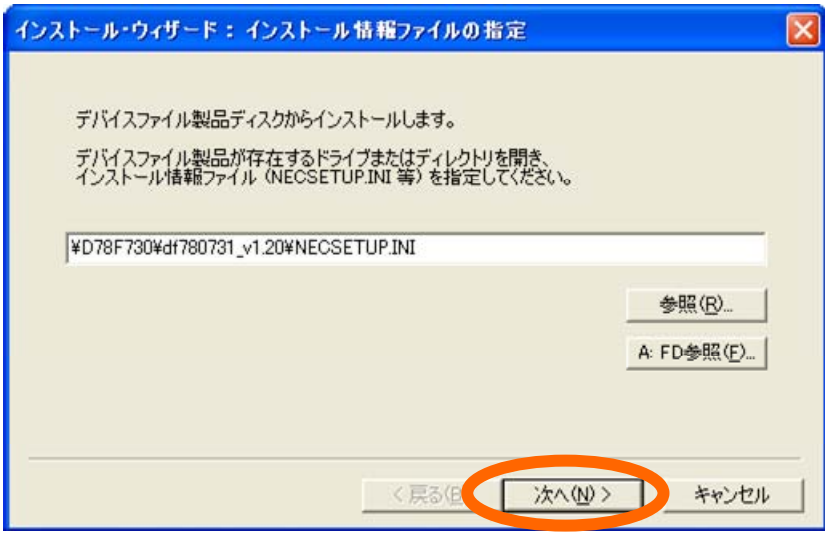
<4>　「インストール情報ファイルの指定」ダイアログが開きます。「参照」ボタンを押下します。



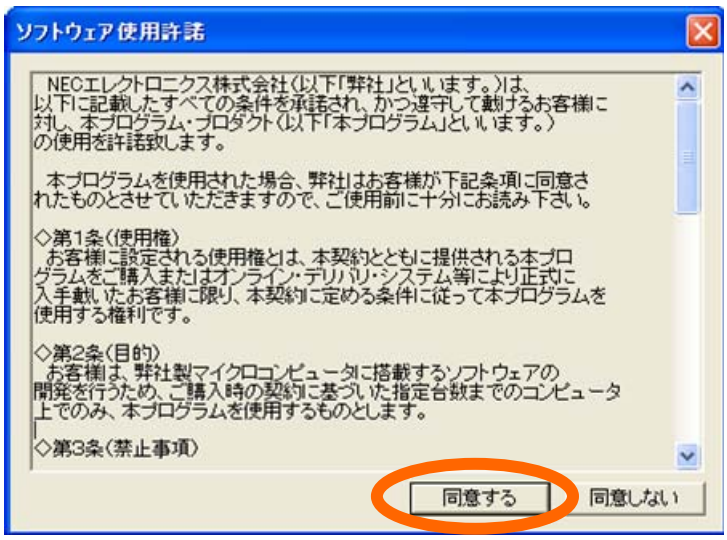
<5>　「ファイルを開く」ダイアログが表示されます。デバイス・ファイルを格納したディレクトリを開き，"NECSETUP.INI" ファイルを選択して「開く」ボタンを押下します。



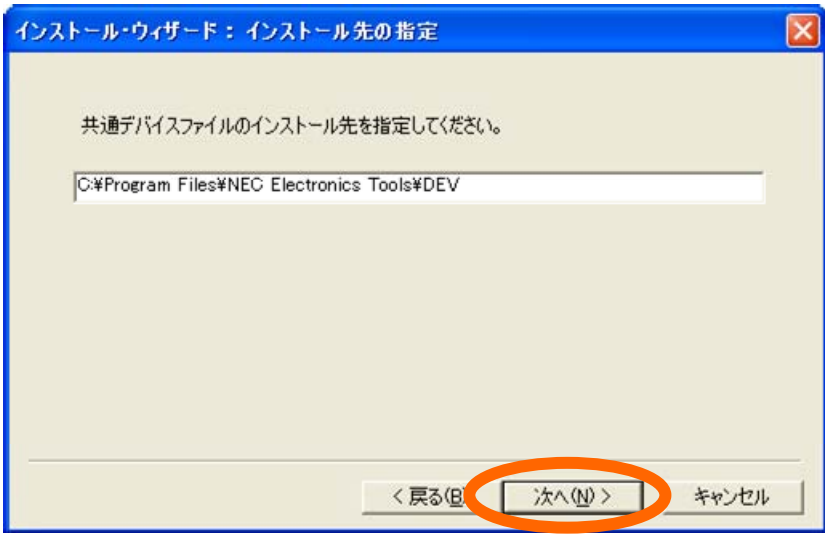
<6>　「インストール情報ファイルの指定」ダイアログに戻ります。「次へ」ボタンを押下します。



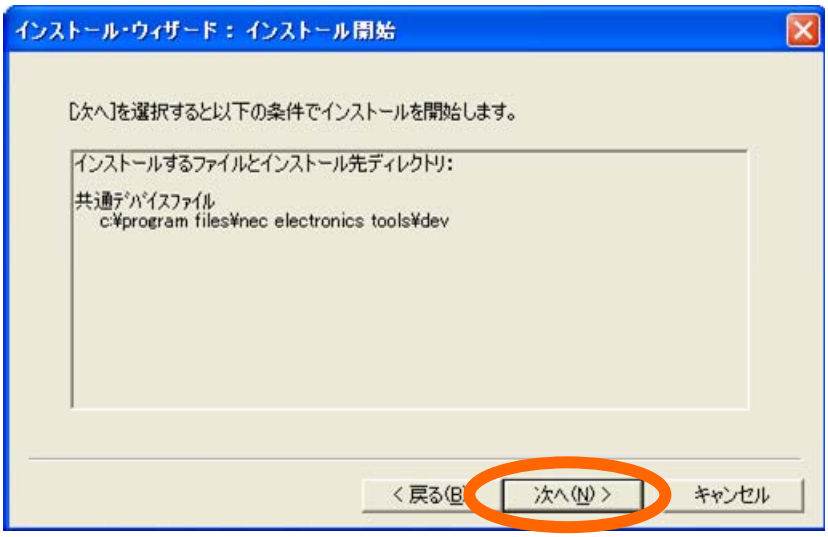
<6> 使用許諾に関するメッセージが表示されます。使用許諾に同意する場合は「同意する」ボタンを押下します。



<8> 「インストール先の指定」ダイアログが開きます。パスが表示されていますので，そのまま「次へ」ボタンを押下します。

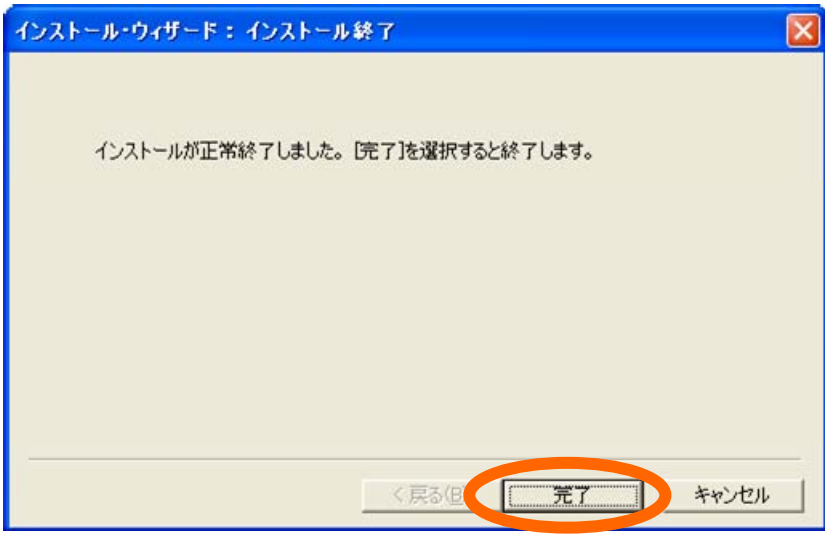


<9>　「インストール開始」ダイアログが開きます。「次へ」ボタンを押下します。



<10> デバイス・ファイルがプロジェクトにインストールされます。環境により，時間がかかる場合があります。

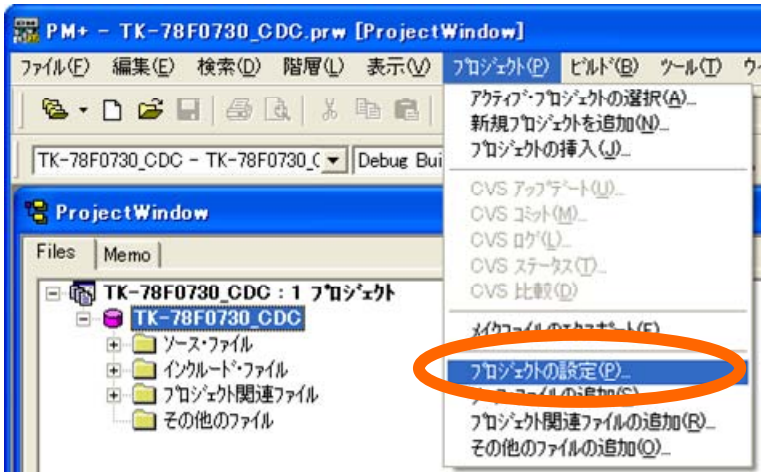
<11> インストールが終わると「インストール終了」ダイアログが開きます。「完了」ボタンを押下します。



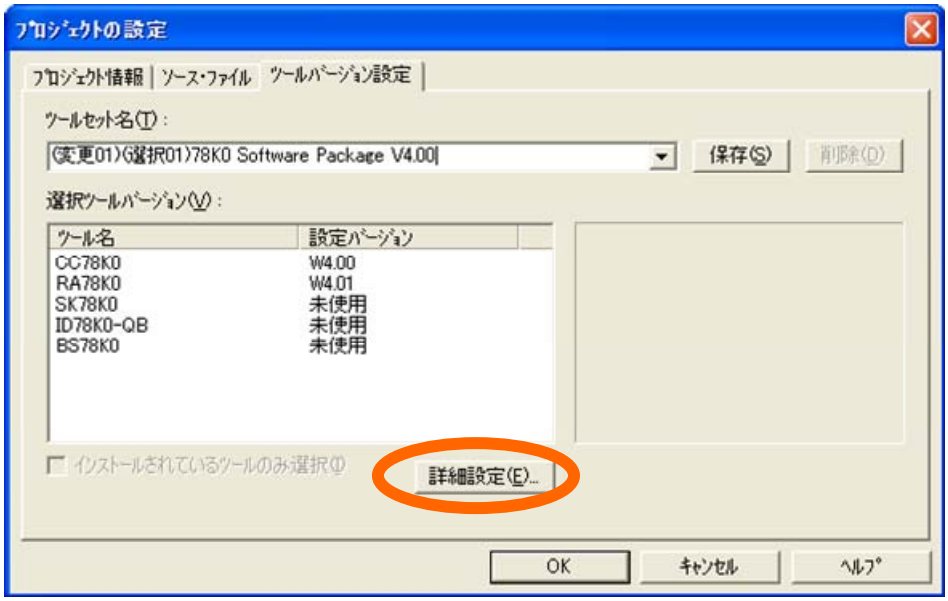
(6) ビルド・ツールの設定

ここではビルド・ツールとして CC78K0 を，デバッグ・ツールとして ID78K0-QB を使用する場合の手順を示します。

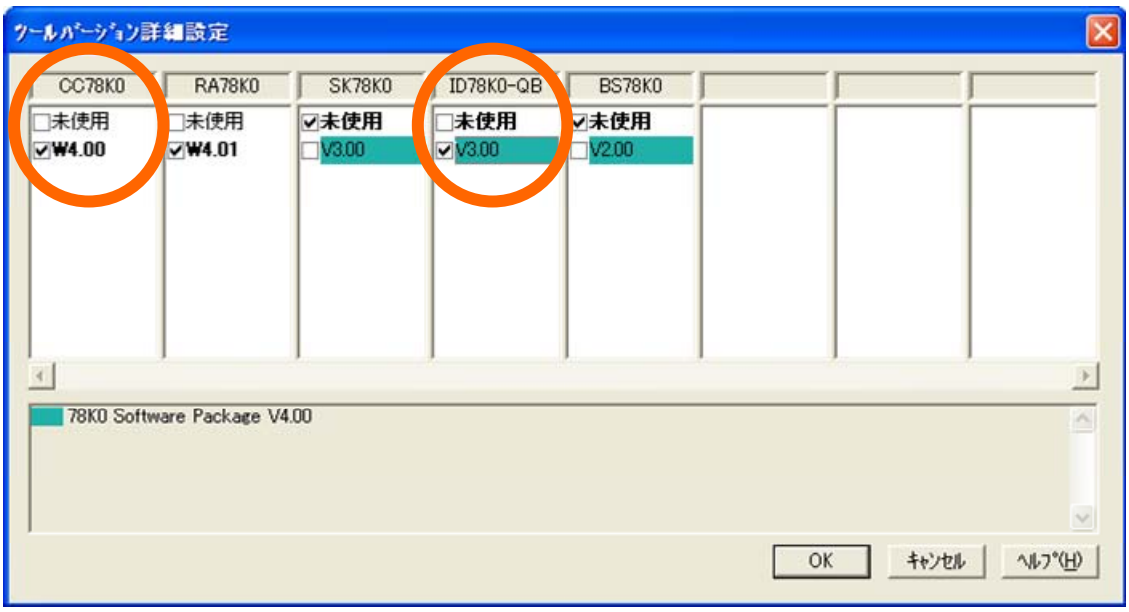
<1> PM+の「プロジェクト」メニューから「プロジェクトの設定」を選択します。



<2> 「プロジェクトの設定」ダイアログが開きます。「ツールバージョン設定」タブの「詳細設定」ボタンを押下します。



<3>　「ツールバージョン詳細設定」ダイアログが開きます。「CC78K0」の欄で使用するコンパイラのバージョンを，「ID78K0-QB」の欄で使用するデバッガのバージョンを選択します。



4. 3　オンチップ・デバッグ

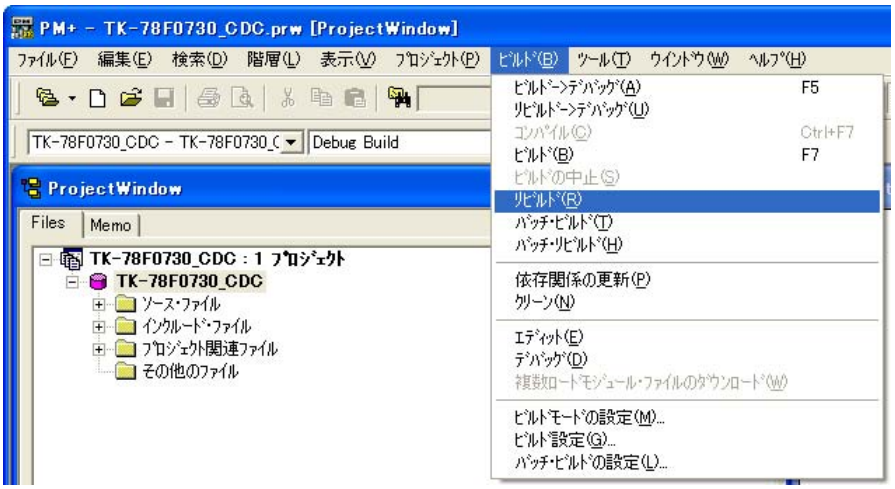
ここでは，4. 2　**環境設定**に示したワークスペースで開発したアプリケーション・プログラムのデバッグ手順について説明します。

μPD78F0730 では，内蔵のフラッシュ・メモリにプログラムを書き込み，デバッガなどから直接実行させて動作を検証すること（オンチップ・デバッグ）が可能です。

4. 3. 1　ロード・モジュール生成

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

PM+では，「ビルド」メニューから「リビルド」を選択すると，ロード・モジュールが生成されます。



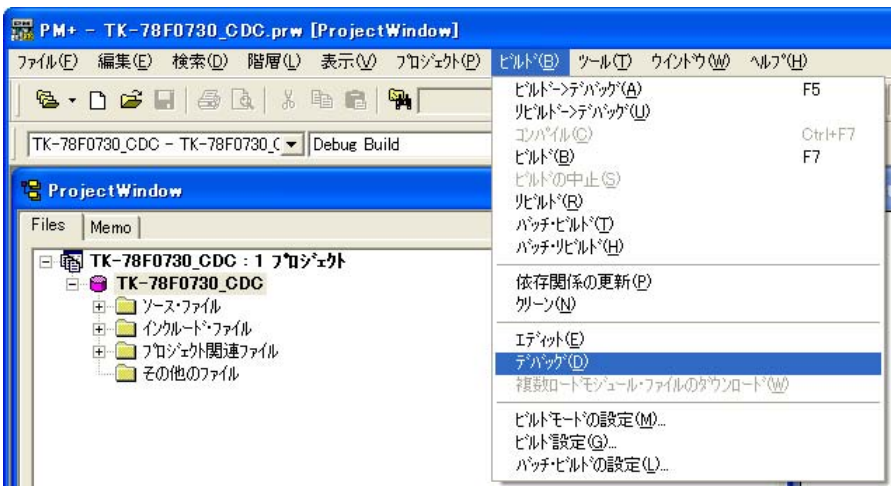
4. 3. 2　ロードと実行

生成したロード・モジュールをターゲットに書き込んで（ロード）実行させます。

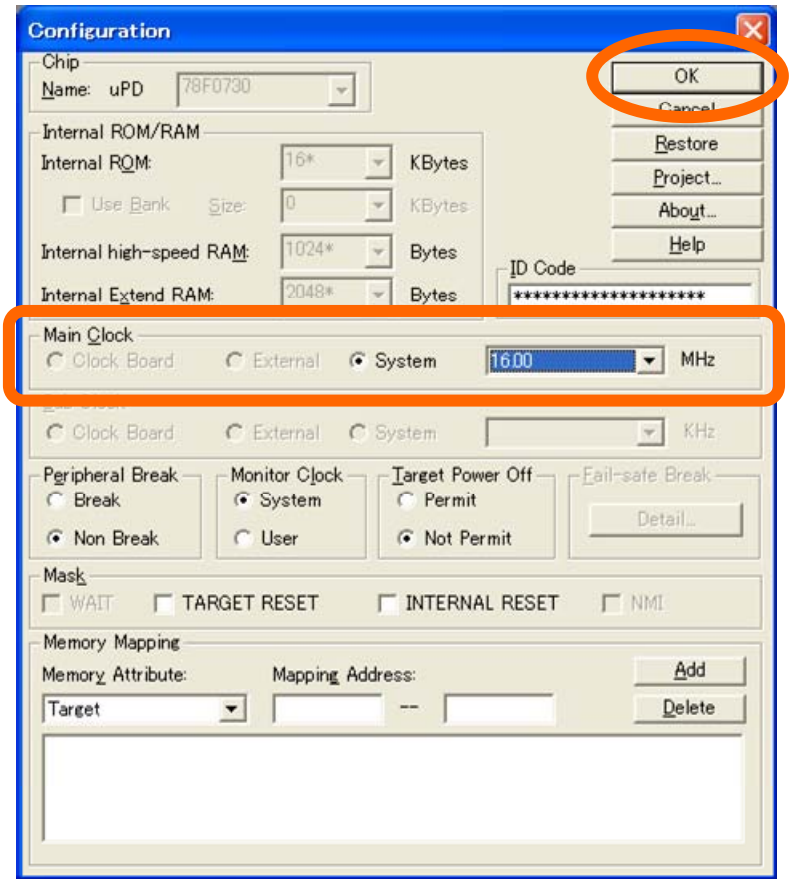
(1) ロード・モジュールの書き込み

ターゲット・ボード上のμPD78F0730 に PM+を介してロード・モジュールを書き込む手順を示します。

<1>　「ビルド」メニューから「デバッグ」を選択して ID78K0-QB-EZ を起動します。

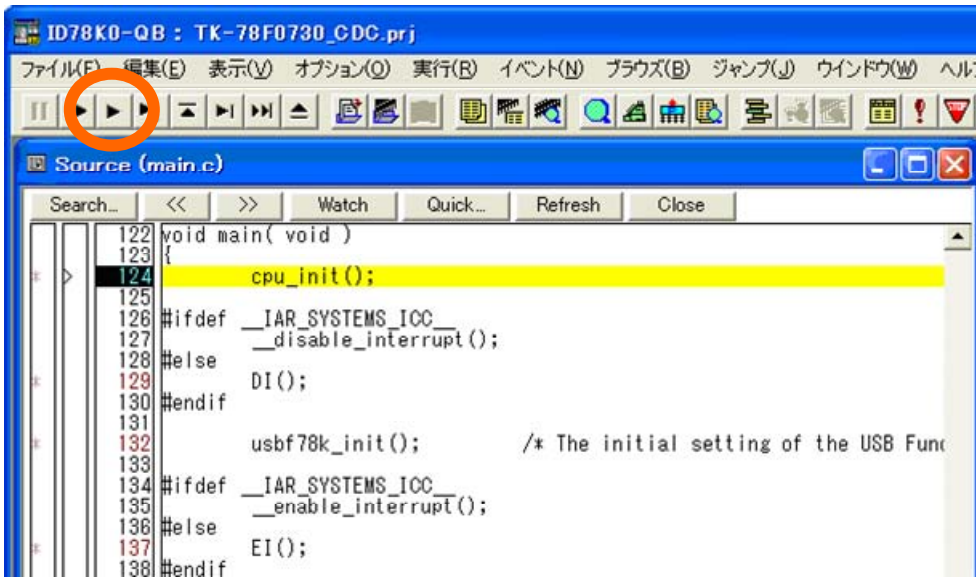


<2>　「Configuration」ダイアログが開きます。「Main Clock」の設定を確認し，「OK」ボタンを押下します。



(2) プログラムの実行

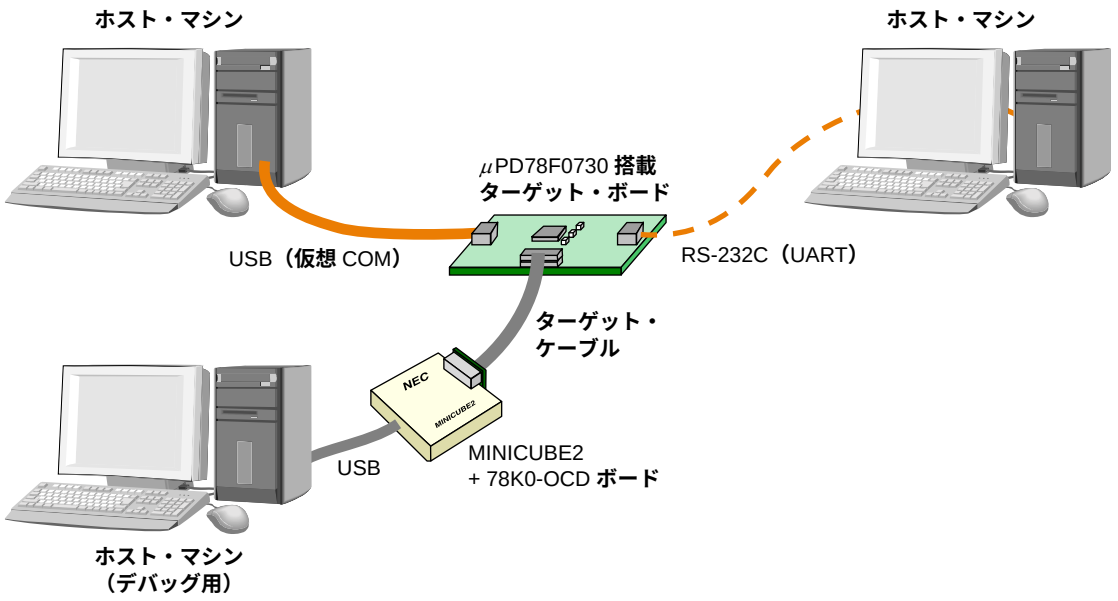
ID78K0-QB の ▶ ボタンを押下します。または「実行」メニューから「継続して実行」を選択します。



4. 3. 3　USB ポート（仮想 COM ポート）の接続

サンプル・プログラムが動作している状態で，ターゲット・ボードの USB ポートとホスト・マシンの USB ポートを接続します。

図 4−4　USB ポート（仮想 COM ポート）の接続



備考　デバッグ用ホスト・マシンに USB（仮想 COM），RS-232C（UART）を接続し，1 台のホスト・マシンを使ってデバッグを行うことも可能です。

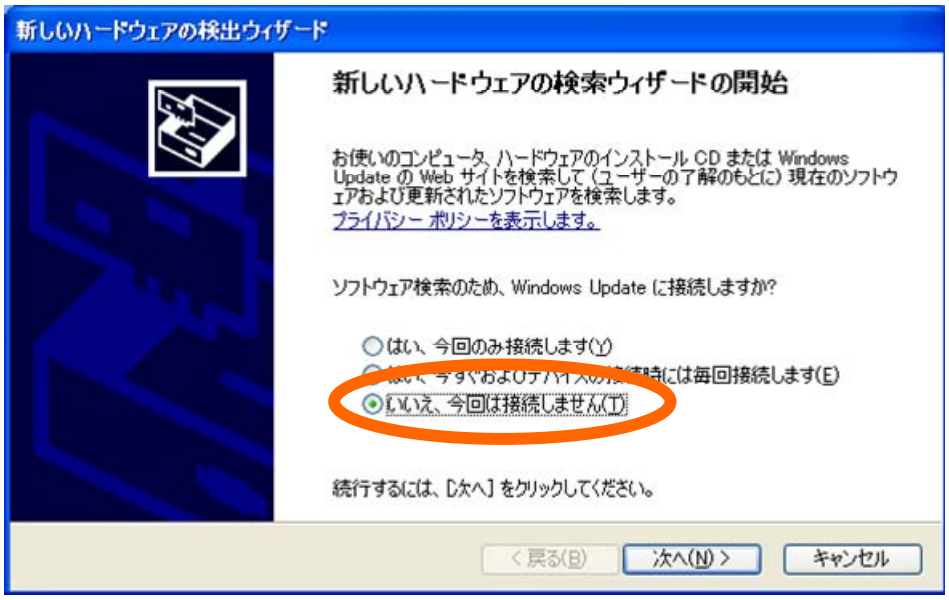
(1) ホスト・ドライバのインストール

ホスト・マシンには USB-シリアル変換用ホスト・ドライバをインストールする必要があります。

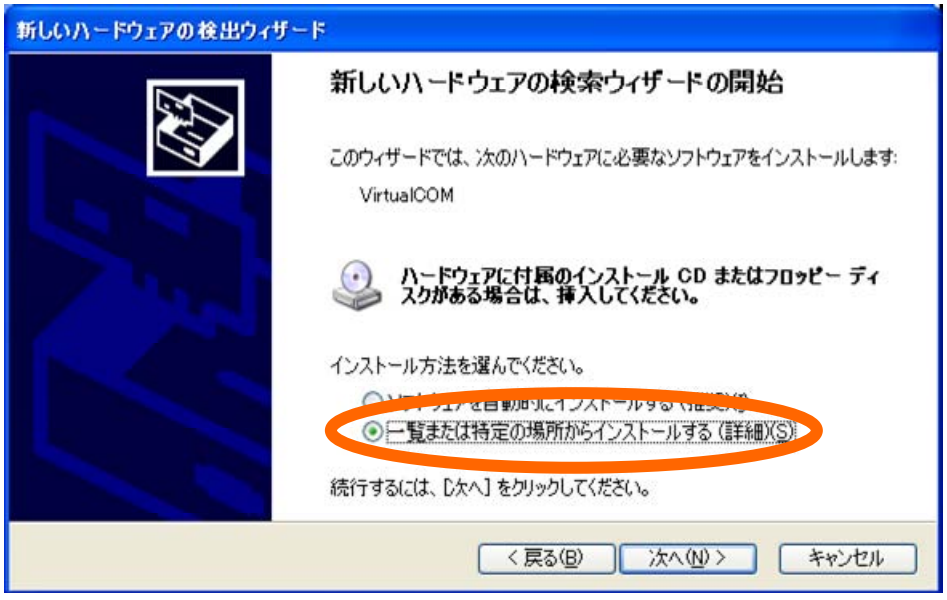
ここではサンプル・ソフトウエアに同梱の USB-シリアル変換用ホスト・ドライバを使用する場合の手順を示します。

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

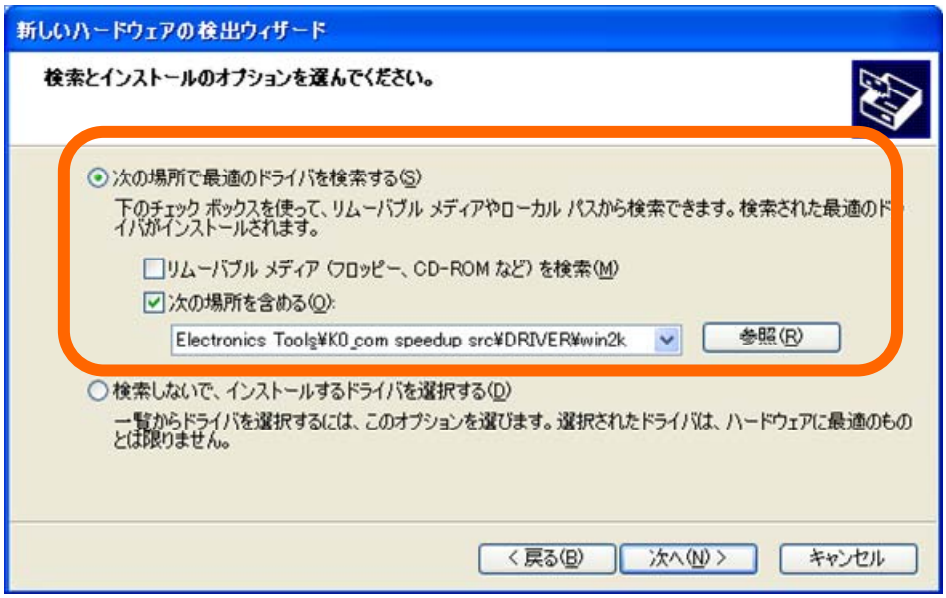
<2> 「新しいハードウエアの検出ウィザード」ダイアログが開きます。「いいえ，今回は接続しません」を選択して「次へ」ボタンを押下します。



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

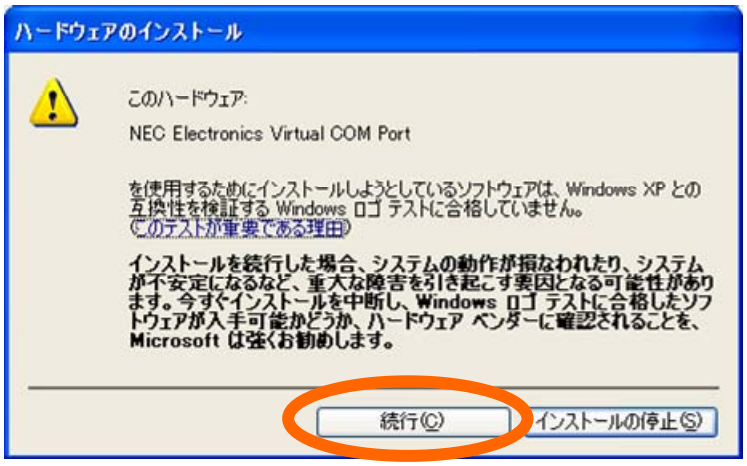


<4> 次の画面が表示されます。「次の場所で最新のドライバを検索する」を選択します。また，「次の場所を含める」のチェックを ON ☑ にし，ホスト・ドライバのフォルダを指定します。
ホスト・ドライバのフォルダを指定したら「次へ」ボタンを押下します。

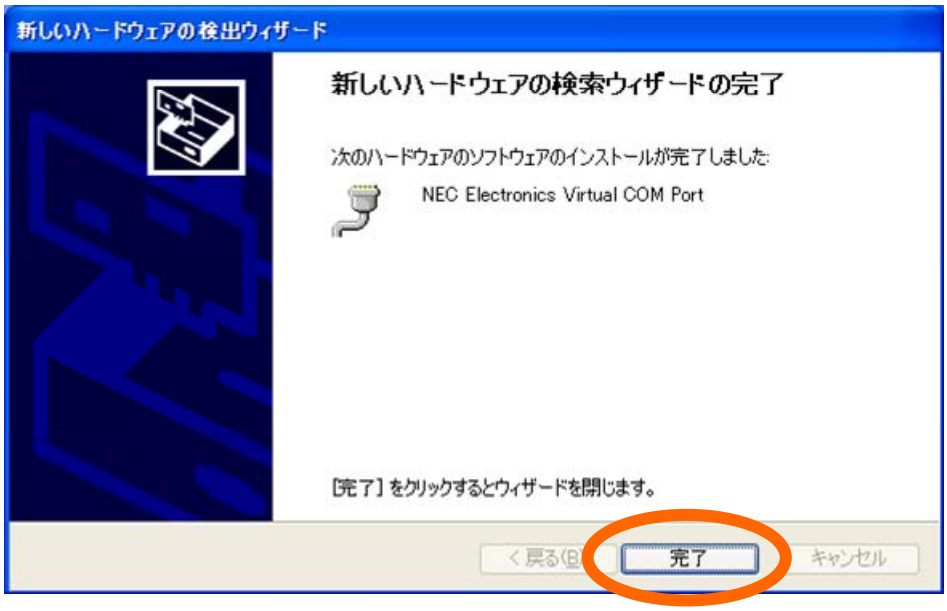


備考　ホスト・ドライバはホスト・マシンの OS の種類に合わせて選択してください。たとえば，32 ビット版 Windows XP の場合は DRIVER フォルダ内の「win2k」，64 ビット版 Windows XP の場合は DRIVER フォルダ内の「wlh_amd64」を選択してください。

<5> 「ハードウエアのインストール」ダイアログが表示されます。「続行」を押下します。

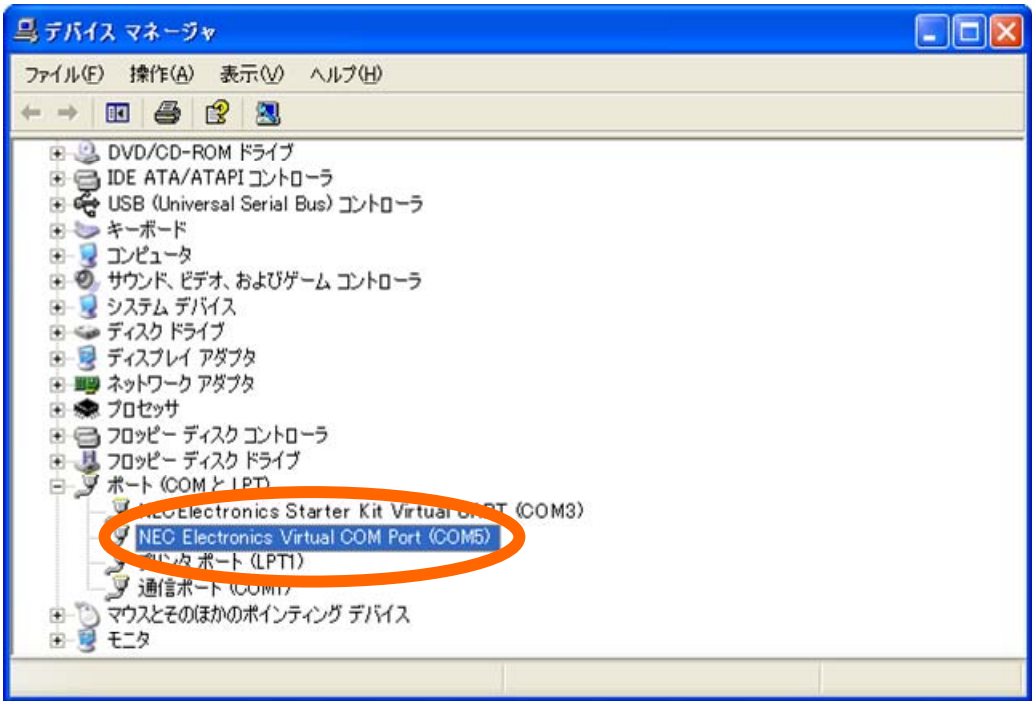


<6> 正常にインストールが完了すると次の画面が表示されます。「完了」ボタンを押下します。



(2) デバイス割り当ての確認

Windows のデバイスマネージャを開きます。デバイスの一覧表示の「ポート」のツリーを展開し，「NEC Electronics Virtual COM Port」が表示されていること，また割り当てられた COM ポート番号を確認します。

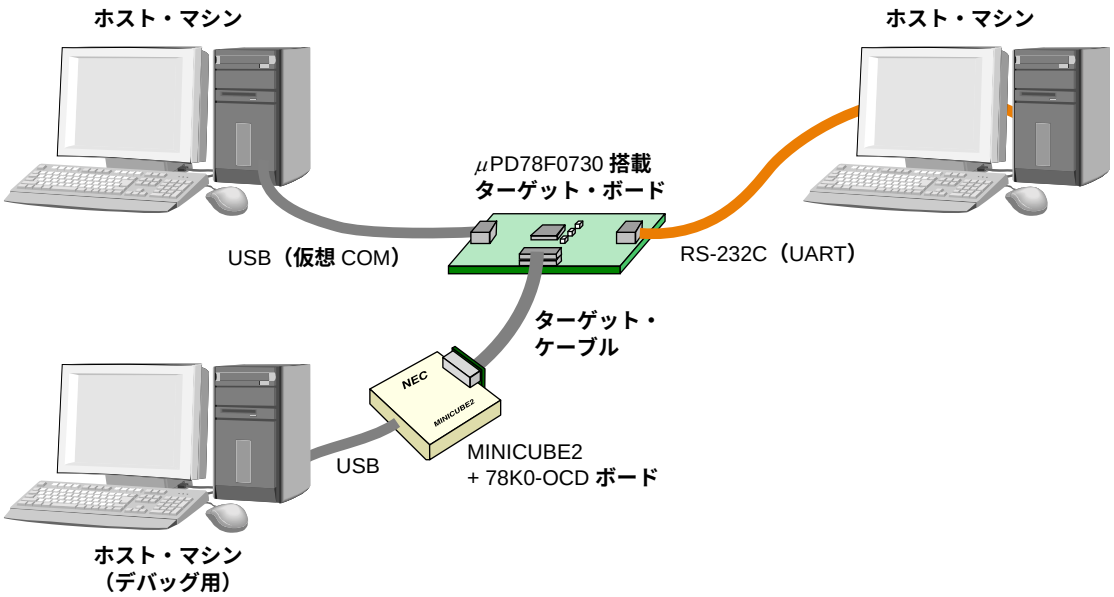


備考　デバイス名やポート番号は任意のものに変更できます。詳細は5. 2　**カスタマイズ**を参照してください。

4. 3. 4　RS-232C ポートの接続

ターゲット・ボードの RS-232C ポートをホスト・マシンに接続します。

図 4−5　RS-232C ポートの接続



備考 1.　デバッグ用ホスト・マシンに USB（仮想 COM），RS-232C（UART）を接続し，1 台のホスト・マシンを使ってデバッグを行うことも可能です。

2.　QB-78F0730-TB は RS-232C ポートを搭載していません。UART 信号レベル（TTL/CMOS レベル）を RS-232C 信号レベルに変換する回路を用いて接続する必要があります。

(1) デバイス割り当ての確認

Windows のデバイスマネージャを開きます。デバイスの一覧表示の「ポート」のツリーを展開し，RS-232C ポートの割り当てられた COM ポート番号を確認します。

表示される名称は，インストールするドライバによって異なります。

4. 3. 5　動作確認

動作の確認は，ホスト・マシン上でターミナル・ソフトウエアを使用して行います。

ここでは，Windows に標準でインストールされている「ハイパーターミナル」を使用した場合の例を示します。

（1）USB 側のハイパーターミナルの設定

USB でターゲット・ボードと接続されているホスト・マシンのポートを設定します。

<1> Windows の［スタート］ボタンから［すべてのプログラム（P)］→［アクセサリ］→［通信］→［ハイパーターミナル］の順に選択し，ハイパーターミナルを起動します。

<2> ハイパーターミナルを起動すると，「接続の設定」ダイアログが開きます。
名前に「USB ポート」と入力し，「OK」ボタンを押下します。アイコンは任意のものを選択できます。

<3> 接続方法に USB ポートの仮想 COM ポート番号を設定し，「OK」ボタンを押下します。
仮想 COM ポートの番号は，デバイスマネージャに表示される NEC Electronics Virtual COM Port の PORT 番号です。確認方法については，4. 3. 3 （2）デバイス割り当ての確認を参照してください。

<4> **転送速度（ビット/秒）を選択し，「**OK**」ボタンを押下します。**

転送速度は 115200，76800，38400，19200，9600，4800，2400 bps **のいずれかを選択してください。「ビット/秒」の項目以外の箇所はデフォルト設定のまま使用してください。**

<5> USB **ポート側のターミナル画面ができあがります。**

(2) RS-232C 側のハイパーターミナルの設定

RS-232C でターゲット・ボードと接続されているホスト・マシンのポートを設定します。

備考　1 台のホスト・マシンとターゲット・ボードを USB と RS-232C の両方で接続することも可能です。

<1> Windows の［スタート］ボタンから［すべてのプログラム（P)］→［アクセサリ］→［通信］→［ハイパーターミナル］の順に選択し，ハイパーターミナルを起動します。

<2> ハイパーターミナルを起動すると，「接続の設定」ダイアログが開きます。
名前に「USB ポート」と入力し，「OK」ボタンを押下します。アイコンは任意のものを選択できます。

<3> 接続方法に RS-232C の COM ポート番号を設定し，「OK」ボタンを押下します。
COM ポートの番号は，デバイスマネージャに表示されるデバッグ・ポートの PORT 番号です。確認方法については，4. 3. 3 (2) デバイス割り当ての確認を参照してください。

<4> **転送速度（ビット/秒）を選択し，「OK」ボタンを押下します。**

注意　USB ポートとシリアル・ポートの転送速度は必ず同じ値に設定してください。

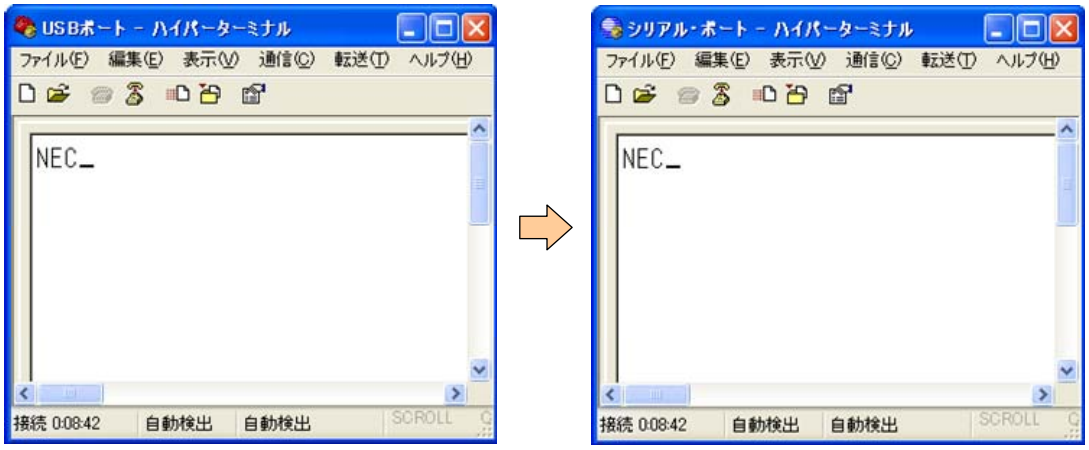
<5> **シリアル・ポート側のターミナル画面ができあがります。**

(3) 通信の確認

ハイパーターミナルにアルファベットの文字（1 バイト文字）を入力して通信状態を確認します。

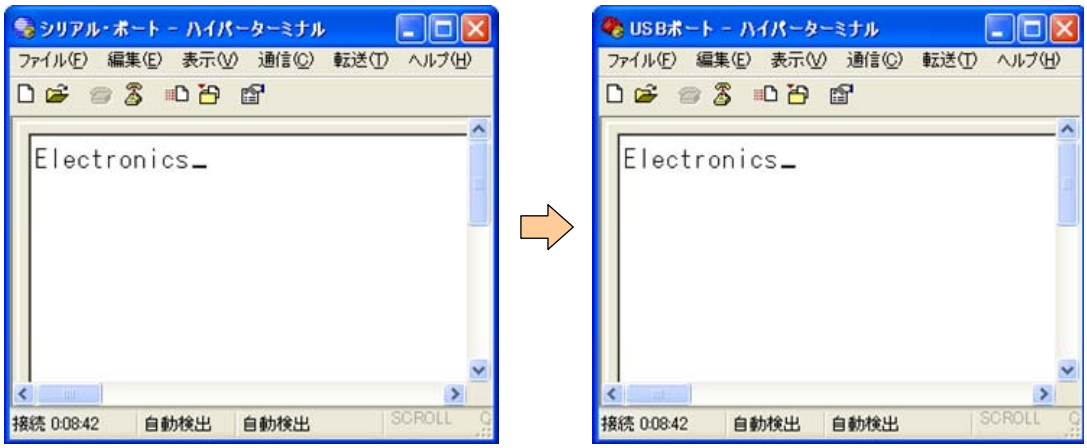
USB ポート側のハイパーターミナルに文字を入力したとき，入力した文字がシリアル・ポート側のハイパーターミナル画面に現れれば，USB からシリアルへの転送は正常です。

図 4−6　USB ポートからシリアル・ポートへの転送確認



シリアル・ポート側のハイパーターミナルに文字を入力したとき，入力した文字が USB ポート側のハイパーターミナル画面に現れれば，シリアルから USB への転送は正常です。

図 4−7　シリアル・ポートから USB ポートへの転送確認



4. 4　注意事項

シリアル・ポート（RS-232C）から USB ポートへ向けてデータを送信する場合，通信速度やホスト・マシン側 OS の状態などの条件によってはデータの一部が欠落することがあります。

4. 4. 1　推奨通信速度

通信速度は，ホスト・マシンの OS に合わせて，次の範囲内で設定してください。

- Windows 2000 使用時　　：　38,400 bps 以下
- Windows XP, Vista 使用時　：　115,200 bps 以下

この範囲を越える速度で通信を行うと，データの抜け落ち（欠落）が発生しやすくなります。また USB ポートからシリアル・ポートへの転送とシリアル・ポートから USB ポートへの転送を同時に行うとデータの欠落が発生する可能性が高くなります。

4. 4. 2　データの欠落原因

μPD78F0730 の USB ファンクション・コントローラのバルク・イン転送（送信）用エンドポイントには，FIFO が複数（64 バイト×2）用意されています。この FIFO は，1 つの FIFO に USB パケットとして送信する 1 パケット分のデータを格納します。このため，シリアル通信のようなデータ・サイズが一定でない転送の場合，最大サイズ分バッファリングされるとは限りません。

サンプル・ソフトウエアでは，データをバッファリングするための内部バッファを用意しています。このバッファはリング・バッファとなっており，データ転送の単位に関わらず，最大でバッファ・サイズ分のシリアル通信データを格納できます（バッファ・サイズは "usbf78k.h" ファイルに定義されており，変更可能です）。

(1) リクエスト発行の遅れ

USB の通信はすべてホスト・マシンからのリクエストに応答して行います。サンプル・ソフトウエアでは，シリアル通信で受信したデータを FIFO へ格納し，USB ホスト・マシンへの転送に備えます。しかし，USB ホスト・マシンの OS が何らかの処理負荷を抱えてリクエスト発行が遅れると，USB への転送が未完了のうちに次のシリアル通信データが届いてしまいます。こうして，FIFO と内部バッファの両方がフルになると，シリアル通信データの読み捨てが生じます。シリアル・ポート側へはμPD78F0730 から任意のタイミングで送信することができるため，USB ポートからシリアル・ポートへ向けた転送ではデータの欠落はほとんど発生しません。

(2) 双方向通信

双方向通信中はサンプル・ソフトウエアの構造上，USB データ受信中に USB データの送信処理が待たされてしまうことがあります。通信速度が速ければ速いほど，この期間にサンプル・ソフトウエア内のバッファがフルになる可能性が高くなります。

(3) UART の受信エラー

「リクエスト発行の遅れ」と「双方向通信」のどちらの場合も，バッファがフルになるとシリアルの受信データが読み捨てられ，データ欠落が生じます。シリアル・ポートのデータを読み捨てずに待たせてしまうと，UART の受信エラー（オーバラン・エラー）が起きてしまい，この場合もデータが欠落することになります。サンプル・ソフトウエアでは，UART の受信エラーを起こさないためにデータの読み捨てを実行しています（通信速度によっては読み捨てが間に合わず，エラーが起きてしまうこともあります）。

第5章　サンプル・ソフトウエアの応用

この章では，μPD78F0730 向け USB-シリアル変換用サンプル・ソフトウエアを利用する際に，知っておいていただきたい情報について説明します。

5. 1　概　　要

サンプル・ソフトウエアの利用には，主に次の 2 つの方法が考えられます。

(1) カスタマイズ

次に示す部分を必要に応じて書き換えます。

- "main.c" ファイル内のアプリケーション部
- "usbf78k_sfr.h" ファイル内の各種レジスタの設定値
- "usbf78k_desc.h" ファイル内の各種ディスクリプタの内容
- 仮想 COM ポート用ホスト・ドライバ（INF ファイル）内のデバイス名やプロバイダ情報

備考　サンプル・ソフトウエアのファイル構成については**1. 1. 3　サンプル・ソフトウエアの構成**を参照してください。

(2) 関数の利用

アプリケーション・プログラム内で必要に応じて呼び出します。実装されている関数の詳細は，**3. 7　関数の仕様**を参照してください。

5.2　カスタマイズ

ここでは，サンプル・ソフトウエアの利用にあたり，必要に応じて書き換える部分について説明します。

5.2.1　アプリケーション部

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

リスト 5－1　メイン・ルーチンの記述

```
1       /*============================================================
2        Main function
3        void main( void )
4
5        Arguments:
6                N/A
7        Return values:
8                N/A
9        Overview:
10               main routine.
11      ============================================================*/
12      void main( void )
13      {
14              cpu_init();
15
16      #ifdef __IAR_SYSTEMS_ICC__
17              __disable_interrupt();
18      #else
19              DI();
20      #endif
21
22              usbf78k_init();      /* The initial setting of the USB Function */
23              uart78k_init();      /* The initial setting of the CDC device */
24
25      #ifdef __IAR_SYSTEMS_ICC__
26              __enable_interrupt();
27      #else
28              EI();
29      #endif
30
31              while ( 1 ) {
32                 usbf78k_send_txbuf();
33
34                 if ( usbf78k_rdata_flg ) {
35                      usbf78k_rdata_flg = 0;
36                      /* transfers to UART */
37                      usbf78k_usb_to_uart(UF0BO1L);
38                 }
39              }
40      }
```

5. 2. 2　レジスタの設定

サンプル・ソフトウエアが使用する（書き込みを行う）レジスタとその設定値は，"usbf78k_sfr.h" ファイルに定義されています。このファイル内の値を実際のアプリケーションでの使用法に合わせて書き換えることで，サンプル・ソフトウエアを介してターゲット・デバイスの動作を設定できます。

（1）"usbf78k_sfr.h" ファイル

USB ファンクション・コントローラのレジスタの定義が記述されています。また，各種処理で使用するレジスタ・ビットとその設定値も定義されています（3. 3. 1　USBF 初期化処理参照）。

5. 2. 3　ディスクリプタの内容

ディスクリプタ情報は "usbf78k_desc.h" ファイルに定義されています（3. 1. 5　ディスクリプタの設定参照）。サンプル・ソフトウエアを通してターゲット・デバイスの属性を設定するときは，このファイル内の値を実際のアプリケーションに合わせて書き換えてください。

なお，デバイス・ディスクリプタのベンダ ID やプロダクト ID を書き換えた場合，ターゲット・デバイス接続の際にインストールするホスト・ドライバ（INF ファイル）でも同様に書き換える必要があります（5. 2. 4　(3) ベンダ ID とプロダクト ID の変更参照）。

また，ストリング・ディスクリプタには任意の情報を登録できますので，適宜書き換えてください。

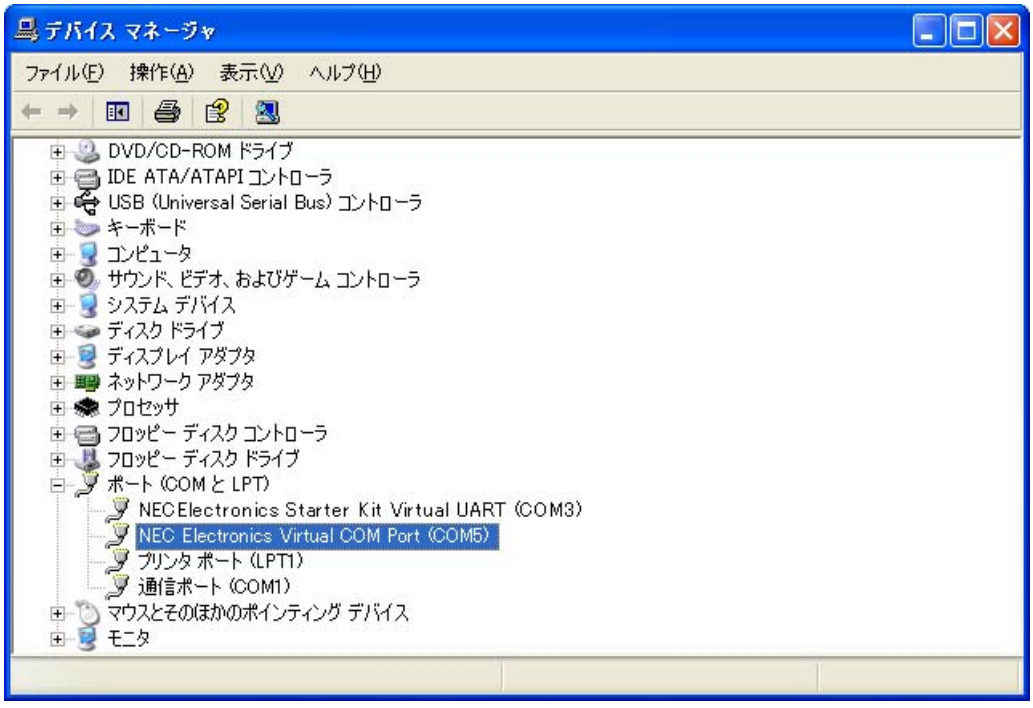
5. 2. 4　仮想 COM ポート用ホスト・ドライバの設定

USB ポート（仮想 COM ポート）用ドライバに関連して，次のようなカスタマイズが可能です。

（1）COM ポート番号の変更

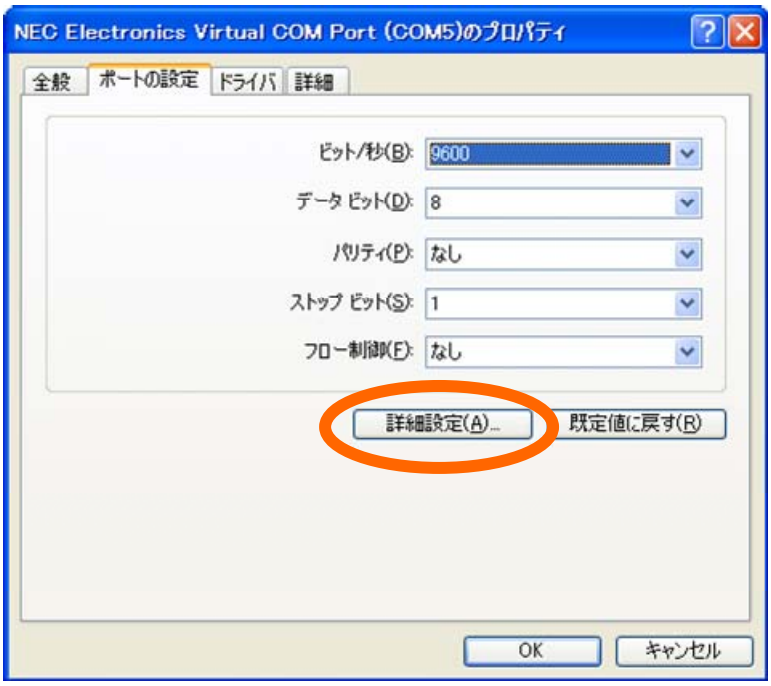
USB デバイスの接続を認識すると，そのデバイスの COM ポート番号をホストが自動的に割り付けますが，任意の番号に変更することもできます。ホスト・マシンで COM ポート番号を変更する手順は次のとおりです。

<1>　Windows のデバイスマネージャを開き，デバイスの一覧表示の「ポート」のツリーを展開します。

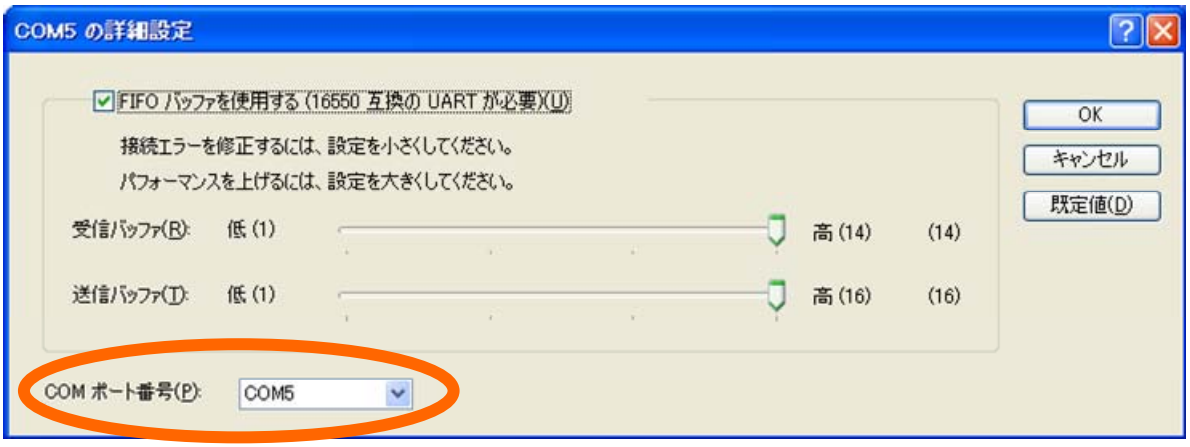


<2>　「NEC Electronics Virtual COM Port (COMn)」（n はホストが割り付けた番号）を選択してプロパティを表示します。

<3>　「ポートの設定」タブの「詳細設定」ボタンを押下します。



<4>　「COMn の詳細設定」ダイアログ（n はホストが割り付けた番号）が開きます。「COM ポート番号」欄のドロップダウン・リストから任意のポート番号を選択します。



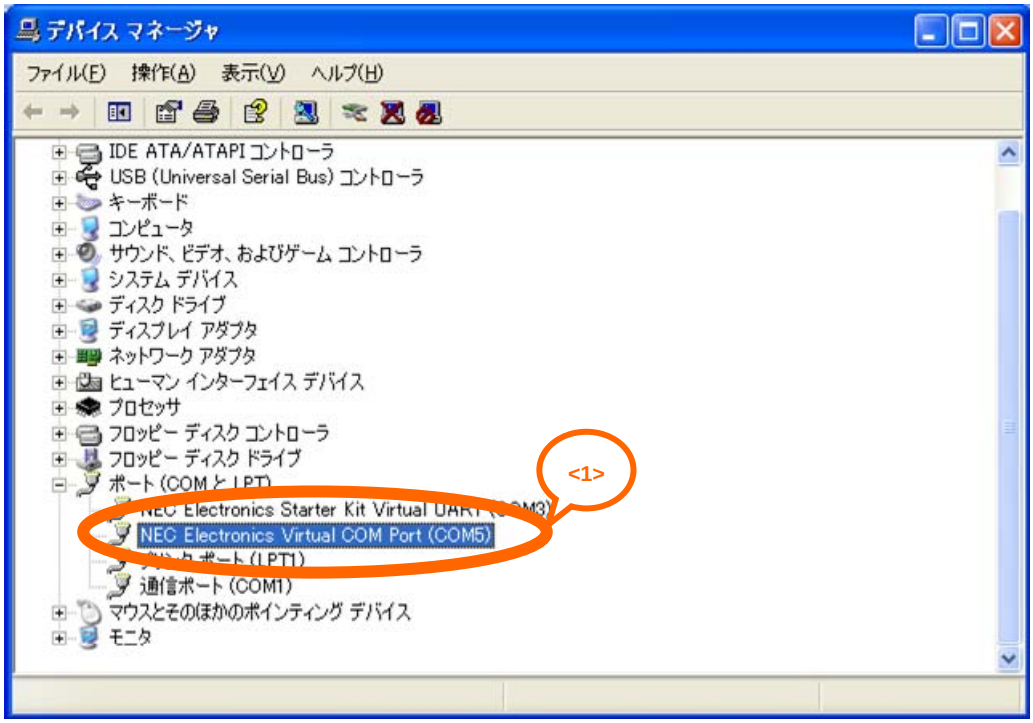
備考 1. ほかのデバイスで使用するポート番号と重ならないようにしてください。

2. この変更後はすぐに新しいポート番号が有効になりますが，デバイスマネージャの一覧表示にはすぐに反映されないことがあります。

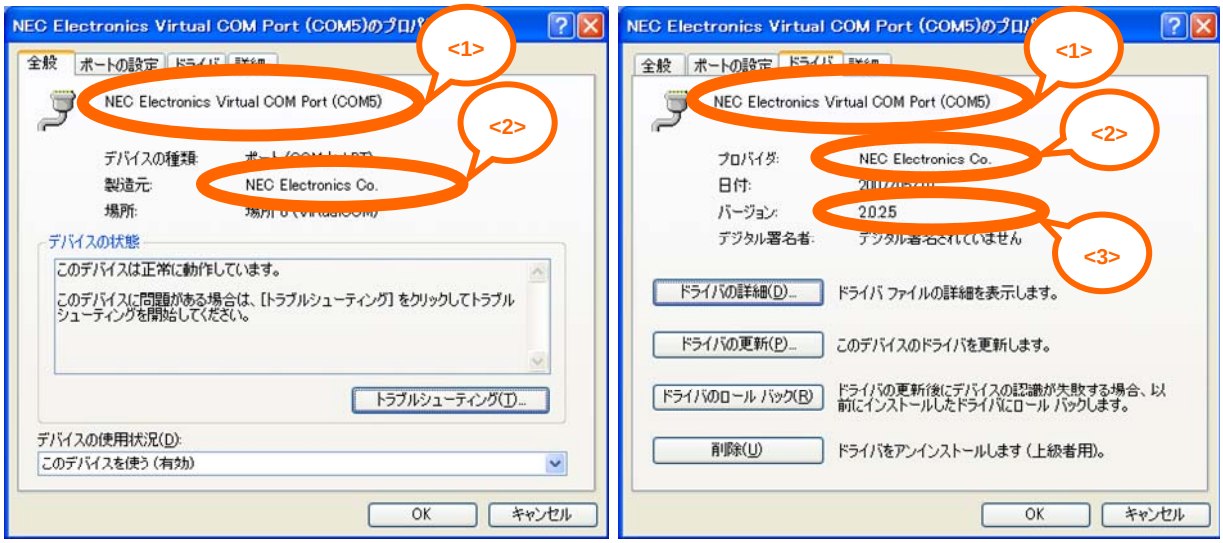
(2) プロパティの変更

Windows のデバイスマネージャで使用されるデバイスの属性など一部の情報は，任意のものに変更できます。変更可能な部分を次に示します。

(a) デバイス名（デバイス一覧）



(b) デバイス名，製造元名，バージョン（デバイスのプロパティ）



これらは，ホスト・ドライバ（INF ファイル）に記述されている情報を元に表示されるため，INF ファイルを書き換えることで変更できます。INF ファイル内で前述の例の番号に対応する部分は次のとおりです。

リスト 5−2　INF ファイル "necelusbvcom.inf" の記述（1/2）

```
1   ;/*++
2   ;
3   ;Copyright (c) NEC Electronics co. All rights Reserved
4   ;
5   ;Module Name:
6   ;
7   ;    NECELUSBVCOM.INF
8   ;
9   ;Abstract:
10  ;    NEC Electronics Virtual COM Port Driver for Windows 2000/Xp
11  ;
12  ;--*/
13  [Version]
14  Signature="$Windows NT$"
15  Class=Ports
16  ClassGuid = {4d36e978-e325-11ce-bfc1-08002be10318}
17  Provider=%MfgName%
18  DriverVer=05/10/2007,2.0.2.5                                          <3>
19
20  [SourceDisksNames]
21  1=%disk1.desc%
22
23  [SourceDisksFiles]
24  NECELUSBDV.sys=1
25  NECELVCOM.sys=1
26
27  [Manufacturer]
28  %MfgName%=SECTION_0
29
30  [SECTION_0]
31  %USB¥VID_0409&PID_01CD.DeviceDesc%=NECELUSBDV_V2.Dev, USB¥VID_0409&PID_01CD
32
33  [DestinationDirs]
34  NECELUSBDV_COPYFILES = 10,System32¥Drivers
35
36  [NECELUSBDV_V2.Dev.NT]
37  CopyFiles=NECELUSBDV_COPYFILES
38  AddReg=NECELUSBDV.AddReg
39
40  [NECELUSBDV.AddReg]
41  HKR,,PortSubClass,1,01
42  HKR,,EnumPropPages32,,"MsPorts.dll,SerialPortPropPageProvider"
43
44  [NECELUSBDV_COPYFILES]
45  NECELUSBDV.sys
46  NECELVCOM.sys
47
48  [NECELUSBDV_V2.Dev.NT.HW]
49  AddReg=NECELUSBDV_Common.Dev.NT.HW.AddReg, NECELUSBDV_V2.Dev.NT.HW.AddReg
50
51  [NECELUSBDV_Common.Dev.NT.HW.AddReg]
52  HKR,,"UpperFilters",0x00010000,"NECELVCOM_FILTER"
53  HKR,,"EndPointCaps",0x00010001,4
54  HKR,,"DebugLevel",0x00010001,3
55  HKR,,"RawDump",0x00010001,0
56
```

リスト 5－2　INF ファイル "necelusbvcom.inf" の記述（2/2）

```
57 [NECELUSBDV_V2.Dev.NT.HW.AddReg]
58 HKR,,"CtlPollPeriod",0x00010001,0
59 HKR,,"RxPollPeriod",0x00010001,0
60 HKR,,"RxReqSize",0x00010001,512
61 HKR,,"EP1_SegmentSize",0x00010001,512
62
63 [NECELUSBDV_V2.Dev.NT.Services]
64 Addservice = NECELUSBDV,2,NECELUSBDV.AddService
65 Addservice = NECELVCOM_FILTER,,NECELVCOM_FILTER.AddService
66
67 [NECELUSBDV.AddService]
68 DisplayName = %NECELVCOM_USB.SvcDesc%
69 ServiceType = 1
70 StartType   = 3
71 ErrorControl   = 1
72 ServiceBinary  = %12%¥NECELUSBDV.sys
73 LoadOrderGroup = Base
74
75 [NECELVCOM_FILTER.AddService]
76 DisplayName    = %NECELVCOM_FILTER.SvcDesc%
77 ServiceType = 1
78 StartType   = 3
79 ErrorControl   = 1
80 ServiceBinary  = %12%¥NECELVCOM.sys
81 LoadOrderGroup = PNP Filter
82
83 [Strings]
84 MfgName="NEC Electronics Co."                                          <2>
85 disk1.desc="NEC Electronics virtual COM port driver install disk"
86 USB¥VID_0409&PID_01CD.DeviceDesc="NEC Electronics Virtual COM Port"    <1>
87 NECELVCOM_USB.SvcDesc="NECEL USBLIB"
88 NECELVCOM_FILTER.SvcDesc="Virtual COM Port for NECEL USB"
```

(3) ベンダ ID とプロダクト ID の変更

デバイス・ディスクリプタ内のベンダ ID とプロダクト ID を変更した場合は，ホスト・ドライバ（INF ファイル）にも同じ内容を設定する必要があります。

INF ファイルでは，**リスト 5－2の** 31 **行目に次のような形式でベンダ** ID **とプロダクト** ID **を記述してください。**

ベンダ ID： "VID_" **に続けて** 4 **桁の** 16 **進数で表記**

プロダクト ID： "PID_" **に続けて** 4 **桁の** 16 **進数で表記**

付録A　ターゲット・ボード

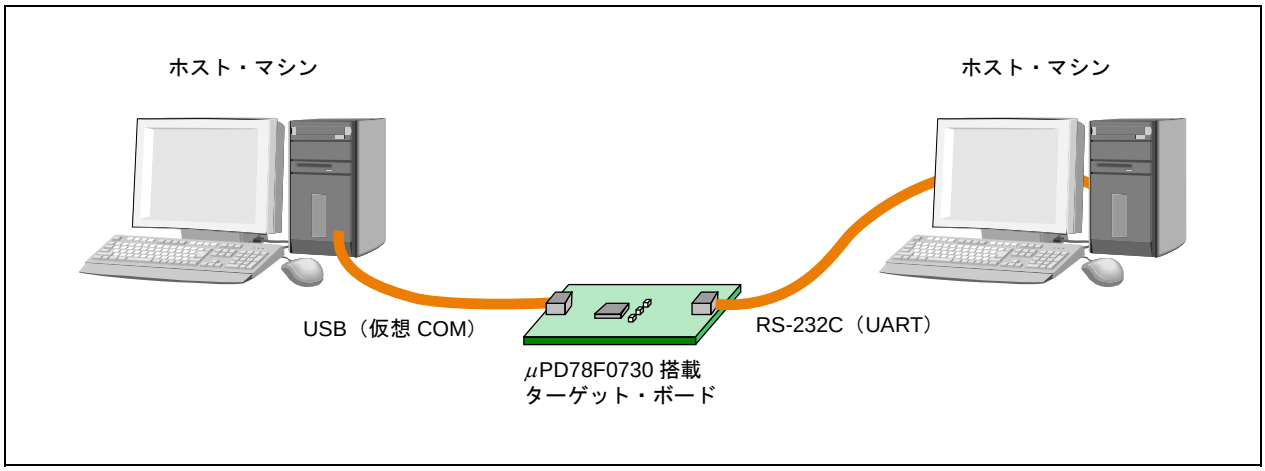
この章では，ターゲット・ボードについて説明します。

A.1　概　　要

このサンプル・ソフトウエアは，USB ファンクション・コントローラで受信したデータをそのまま UART から送信する，または UART で受信したデータをそのまま USB ファンクション・コントローラから送信するためのソフトウエアです。

この機能を使用するには，USB ポートと UART ポートを持つターゲット・ボードが必要となります。μPD78F0730 の UART 入出力信号を直接 PC に接続することはできないため，信号レベルを変換して RS-232C インタフェースを介して PC に接続する必要があります。

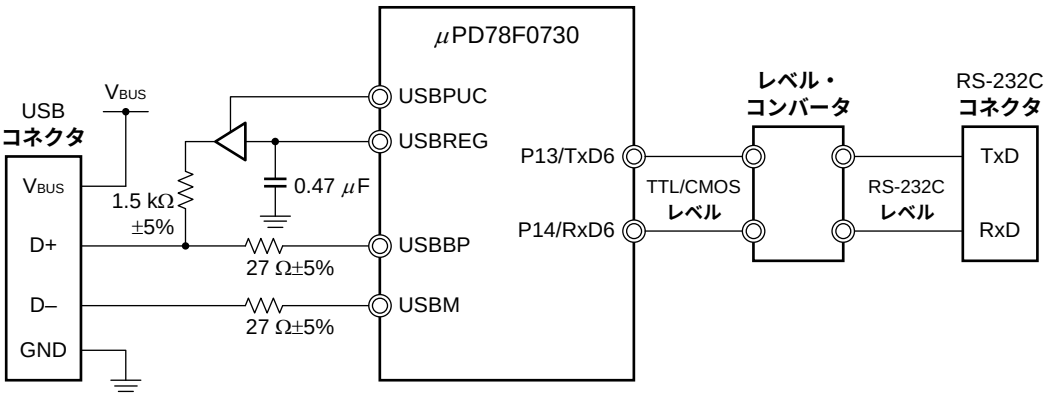
図 A−1　ターゲット・ボードの接続イメージ



A. 2　回路例

ターゲット・ボードの USB，RS-232C 部分の回路構成例を次に示します。

図 5－1　USB，RS-232C 部分の回路構成例



注意　回路例および回路定数は，例示的に示したものです。実際にターゲット・ボードを作成する場合は，十分に評価した上で回路および回路定数を決定してください。

［メ　　モ］

【発　行】

NEC**エレクトロニクス株式会社**

〒211-8668　**神奈川県川崎市中原区下沼部**1753

電話（代表）：(044)435−5111

【ホームページ】

NEC**エレクトロニクスの情報がインターネットでご覧になれます。**

URL**（アドレス）**　http://www.necel.co.jp/

【資料請求先】

NEC**エレクトロニクスのホームページよりダウンロードいただくか，**NEC**エレクトロニクスの販売特約店へお申し付けください。**

お問い合わせ先

【営業関係，デバイスの技術関係お問い合わせ先】

半導体ホットライン
（電話：午前 9:00〜12:00，**午後** 1:00〜5:00）

電　話　：(044)435−9494
E-mail　：info@necel.com

【マイコン開発ツールの技術関係お問い合わせ先】

開発ツールサポートセンター

E-mail　：toolsupport-micom@ml.necel.com

C0901-A-01