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

RX77016

リアルタイム・オペレーティング・システム

HOST API 編

対象デバイス

μPD77016
μPD77017
μPD77018
μPD77018A
μPD77019
μPD77110
μPD77111
μPD77112
μPD77113
μPD77114

資料番号　U14371JJ1V0AN00（第 1 版）
発行年月　July 1999 NS　CP（K）

© NEC Corporation 1999

[メ　　モ]

目次要約

第 1 章　概　　要　…　15

第 2 章　HOST API の構成　…　17

第 3 章　HOST API の機能　…　21

第 4 章　入力ファイル　…　31

第 5 章　HOST API の仕組み　…　37

第 6 章　API ファンクション　…　39

第 7 章　BIOS ファンクション　…　69

第 8 章　HOST API 実行ファイルの作成　…　79

Windows 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。

- **本資料の内容は予告なく変更することがありますので，最新のものであることをご確認の上ご使用ください。**
- 文書による当社の承諾なしに本資料の転載複製を禁じます。
- 本資料に記載された製品の使用もしくは本資料に記載の情報の使用に際して，当社は当社もしくは第三者の知的財産権その他の権利に対する保証または実施権の許諾を行うものではありません。上記使用に起因する第三者所有の権利にかかわる問題が発生した場合，当社はその責を負うものではありませんのでご了承ください。
- 本資料に記載された回路，ソフトウエア，及びこれらに付随する情報は，半導体製品の動作例，応用例を説明するためのものです。従って，これら回路・ソフトウエア・情報をお客様の機器に使用される場合には，お客様の責任において機器設計をしてください。これらの使用に起因するお客様もしくは第三者の損害に対して，当社は一切その責を負いません。

M7A　98.8

巻末にアンケート・コーナを設けております。このドキュメントに対するご意見をお気軽にお寄せください。

はじめに

対象者　このアプリケーション・ノートは，μPD77016ファミリの機能を理解し，それを用いたアプリケーション・プログラムを設計するユーザを対象とします。

μPD77016ファミリは，μPD77016, 77017, 77018, 77018A, 77019, 77110, 77111, 77112, 77113, 77114の総称です。このマニュアルでは，特に機能面において違いがないかぎり，μPD77016を代表品種として説明しています。

目的　このアプリケーション・ノートは，RX77016を搭載したμPD77016ファミリとHOST Processor間のデータ通信を支援するためのAPI（Application Program Interface）であるHOST APIを，ユーザに理解していただくことを目的とします。なお，掲載のプログラム構成は例示的に示したものであり，量産設計を対象とするものではありません。

構成　このアプリケーション・ノートでは，HOST APIの構成，仕組みと，APIファンクション，BIOSファンクションについて説明しています。

読み方　このマニュアルの読者は，論理回路，マイクロコンピュータ，C言語およびWindows™に関する一般的知識が必要となります。

RX77016の機能について知りたいとき
→ **RX77016　ユーザーズ・マニュアル　機能編**を参照してください。

RX77016コンフィギュレーション・ツールの操作について知りたいとき
→ **RX77016　ユーザーズ・マニュアル　コンフィギュレーション・ツール編**を参照してください。

μPD77016ファミリのハードウエア機能について知りたいとき
→ **μPD7701×ファミリ　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。

μPD77016ファミリの命令機能について知りたいとき
→ **μPD7701×ファミリ　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
アクティブ・ロウの表記　：$\overline{\times\times\times}$（端子，信号の名称に上線）
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0b××××
10進数 … ××××
16進数 … 0x××××

表現形式

このアプリケーション・ノートでは，API ファンクションを表現する場合，C 言語の関数であることを示す “ () ” を付加し，BIOS ファンクションを表現する場合，μPD77016 のアセンブリ言語のレーベルであることを示す “ : ” を付加しています。

例 API ファンクションの “ OpenDEV ” → “ OpenDEV() ” と表します。
BIOS ファンクションの “ EchoWord ” → “ EchoWord: ” と表します。

HOST API for RX77016 ではターゲット・システム上の DSP（Digital Signal Processor）としてμPD77016 ファミリを想定していますが，このアプリケーション・ノート中では単に DSP と記述しています。

このアプリケーション・ノートでは HOST API for RX77016 の 1999 年 7 月時点のバージョンを現在のバージョンとして記述しています。

*×××は×××をアドレスとするメモリの内容を表します。

例 DP0 レジスタの内容が 0x1000 のとき，*DP0 はメモリの 0x1000 番地の内容を表します。

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。

●デバイスに関する資料

資料名／品名	パンフレット	データ・シート	ユーザーズ・マニュアル		アプリケーション・ノート	
			アーキテクチャ編	命令編	基本ソフトウエア編	ライブラリ編
μPD77016	U12395J	U10891J	U10503J	U13116J	U11958J	U12021J
μPD77017		U10902J				
μPD77018						
μPD77018A		U11849J				
μPD77019						
μPD77019-013		U13053J				
μPD77110		U12801J	作成中			
μPD77111						
μPD77112						
μPD77113		U14373J				
μPD77114						

●開発ツールに関する資料

資　料　名		資料番号
SM77016　ユーザーズ・マニュアル		U11602J
WB77016　ユーザーズ・マニュアル	言語編	U10078J
	操作編	U11506J
ID77016　ユーザーズ・マニュアル		U10118J
IE-77016-98/PC　ユーザーズ・マニュアル	ハードウエア編	U13044J
IE-77016-CM-EM6　ユーザーズ・マニュアル		EEU-984
EB-77017　ユーザーズ・マニュアル		EEU-983
μPD77016　スタータ・キット　ユーザーズ・マニュアル		U13032J
IE-77016-CM-LC　ユーザーズ・マニュアル		U14139J
RX77016　ユーザーズ・マニュアル	機能編	U14397J
	コンフィギュレーション・ツール編	U14371J
RX77016　アプリケーション・ノート	HOST API 編	このマニュアル

注意　上記関連資料は，予告なしに内容を変更することがあります。設計などには，必ず最新の資料をご使用ください。

[メ　　モ]

目　次

第 1 章　概　　要　…　15

1.1　API ファンクションの概要　…　15
1.2　BIOS ファンクションの概要　…　16

第 2 章　HOST API の構成　…　17

2.1　ソフトウエア構成　…　17
2.2　ハードウエア構成　…　19

第 3 章　HOST API の機能　…　21

3.1　通信経路の確立，遮断　…　21
3.2　OS 通信用領域へのデータ設定　…　22
3.3　リブート　…　23
3.4　データ・メモリ・アクセス　…　24
3.5　コミュニケーション・バッファ　…　25
3.5.1　コミュニケーション・バッファ領域の確保と解放　…　26
3.5.2　コミュニケーション・バッファ・アクセス　…　28
3.6　タスク管理　…　30

第 4 章　入力ファイル　…　31

4.1　Information File　…　31
4.1.1　Device セクション　…　31
4.1.2　IO RESOURCE セクション　…　31
4.1.3　Task セクション　…　34
4.1.4　TaskXX セクション　…　34
4.1.5　SubTaskXXYY セクション　…　34
4.1.6　Information File の記述例　…　35
4.2　HEX ファイル　…　36

第 5 章　HOST API の仕組み　…　37

5.1　BIOS モード　…　37
5.2　BIOS ファンクション・コール　…　38

第 6 章　API ファンクション　…　39

6.1　データ定義　…　39

6.2　API ファンクションの機能　…　41

6.2.1　**初期設定ファンクション**　…　41

6.2.2　**タスク管理ファンクション**　…　46

6.2.3　**メモリ・アクセス・ファンクション**　…　49

6.2.4　HOST API **管理ファンクション**　…　66

第 7 章　BIOS ファンクション　…　69

7.1　HI 割り込み発生時のアクション　…　69

7.2　BIOS ファンクションの機能　…　69

第 8 章　HOST API 実行ファイルの作成　…　79

8.1　API ファンクションを使用した実行ファイルの作成　…　79

8.1.1　hapiucfg.h　…　79

8.1.2　spx.c　…　80

8.2　BIOS ファンクションを使用した実行ファイルの作成　…　81

8.2.1　os_udef.h　…　81

8.2.2　**ターゲット・システム用初期化部（**_MOS_TargetSysInit:**）**　…　81

8.2.3　HI **割り込みハンドラ・コード部（**_MOS_ivHI:**）**　…　81

8.2.4　biosucfg.h　…　82

図の目次

図番号	タイトル，ページ
2－1	ソフトウエア構成イメージ　…　17
2－2	ハードウエア構成イメージ　…　19
2－3	バイト・アクセス・イメージ　…　20
3－1	通信経路の確立，遮断イメージ（マルチ・デバイス対応）　…　21
3－2	OS 通信領域へのデータ設定イメージ　…　22
3－3	リブート・イメージ　…　23
3－4	データ・メモリ・アクセス・イメージ　…　24
3－5	コミュニケーション・バッファ・イメージ　…　25
3－6	コミュニケーション・バッファ領域確保イメージ　…　26
3－7	使用領域開放イメージ　…　27
3－8	コミュニケーション・バッファ・アクセス・イメージ　…　28
3－9	タスク管理イメージ　…　30
4－1	Information File の記述例　…　35
4－2	拡張 Intel HEX File を利用したリブート・イメージ　…　36
5－1	OpenDEV()，CloseDEV()をコールしたときの BIOS モード変化　…　37
5－2	BIOS カーネルの動作フロー　…　38
6－1	リード・コミュニケーション・バッファの読み出し先アドレス　…　55
6－2	ライト・コミュニケーション・バッファの読み出し先アドレス　…　56

表の目次

表番号	タイトル，ページ
3－1	コミュニケーション・バッファ・アクセス状態一覧　…　29
6－1	データ型一覧　…　39
6－2	文字定数一覧　…　40
7－1	シフト量とジャンプ先　…　70
7－2	*_MHA_PCmdPtr:MEM の内容とジャンプ先　…　71
7－3	コール・アドレスとリブート・ルーチン　…　72

［メ　　モ］

第 1 章　概　　要

HOST API とは，μPD77016 ファミリと Host Processor 注間でデータ通信を行うために，ユーザ・アプリケーションに対して提供するプログラム・インタフェースのことです。ユーザ・アプリケーションはすべてのデータ通信を，HOST API を経由して行います。

注　現在のバージョンの HOST API for RX77016 では Host Processor にパーソナル・コンピュータを想定しています。

1.1　API ファンクションの概要

HOST API システムの提供する API ファンクションの概要を次に示します。API ファンクションの詳細は**第 6 章　API ファンクション**を参照してください。

(1) 初期設定ファンクション

初期設定ファンクションは，ターゲット・デバイスに対し次の機能を提供します。

OpenDEV()	…	通信経路の確立
CloseDEV()	…	通信経路の遮断
RebootDEV()	…	ホスト・ブート／セルフ・ブート
ResetDEV()	…	システム・リセット
SetCommunicateValue()	…	OS 通信用領域へのデータ設定

(2) タスク管理ファンクション

タスク管理ファンクションは，ターゲット・デバイスの特定のタスクに対し次の機能を提供します。

ShowTask()	…	タスク情報の出力
ResumeSyncTask()	…	フレーム・カウンタ A 値の更新
SuspendSyncTask()	…	フレーム・カウンタ A 値の 0 クリア

(3) メモリ・アクセス・ファンクション

メモリ・アクセス・ファンクションは，ターゲット・デバイスの特定のデータ・メモリ，またはコミュニケーション・バッファ（3.5　**コミュニケーション・バッファ**参照）に対し，次の機能を提供します。

CreateCBuf()	…	コミュニケーション・バッファ領域の確保
FreeCBuf()	…	コミュニケーション・バッファ領域の開放
GetCBufCaps()	…	コミュニケーション・バッファ情報の表示
InitCBuf()	…	コミュニケーション・バッファの 0 クリア
ReadFromTMem()	…	X or Y メモリから 1 データ・リード
WriteToTMem()	…	X or Y メモリに 1 データ・ライト
ReadFromCBuf()	…	コミュニケーション・バッファから 1 データ・リード
WriteToCBuf()	…	コミュニケーション・バッファに 1 データ・ライト
CopyTMemToHost()	…	ホスト上のメモリから X or Y メモリへコピー

CopyHostToTMem() … X or Y メモリからホスト上のメモリへコピー
CopyCBufToHost() … コミュニケーション・バッファからホスト上のメモリへコピー
CopyHostToCBuf() … ホスト上のメモリからコミュニケーション・バッファへコピー
CopyCBufToTmem() … コミュニケーション・バッファからホスト上のメモリへコピー
CopyTMemToCBuf() … ホスト上のメモリからコミュニケーション・バッファへコピー

（4）HOST API 管理ファンクション

HOST API 管理ファンクションでは，HOST API システムの使用を円滑に行うために次の機能を提供します。

LoadInformationFIle() … Information File の Load
ShowInfoContentsAll() … コンテンツ情報の表示
GetVersion() … バージョン情報の表示
HexToBuf() … HEX ファイルのバッファリング

1.2　BIOS ファンクションの概要

HOST API システムは,次の BIOS ファンクションを提供します。BIOS ファンクションの詳細は,**第 7 章　BIOS ファンクション**を参照してください。

EchoWord: … 通信経路不確立時の HI 割り込みに対応するアクション
Interpreter: … HI 割り込みに対応するファンクションの指定
SelPCmd: … ジャンプ先ファンクション・アドレスの指定
PuttoCmdBuf: … 指定した回数の HI 割り込みがあった時点でジャンプ
IDTagCmd: … 指定したユーザ定義ファンクションへジャンプ
Reboot: … コールされたアドレスにしたがってリブート
Direct_RX: … X メモリから 1 データ・リード
Direct_RY: … Y メモリから 1 データ・リード
Direct_WX: … X メモリに 1 データ・ライト
Direct_WY: … Y メモリに 1 データ・ライト
Copy_XtoX: … X メモリ内のデータ・コピー
Copy_YtoY: … Y メモリ内のデータ・コピー
Copy_XtoY: … X メモリから Y メモリへのデータ・コピー
Copy_YtoX: … Y メモリから X メモリへのデータ・コピー
ReadFromCBuf: … ライト・コミュニケーション・バッファからデータ・リード
WriteToCBuf: … リード・コミュニケーション・バッファにデータ・ライト

第 2 章　HOST API の構成

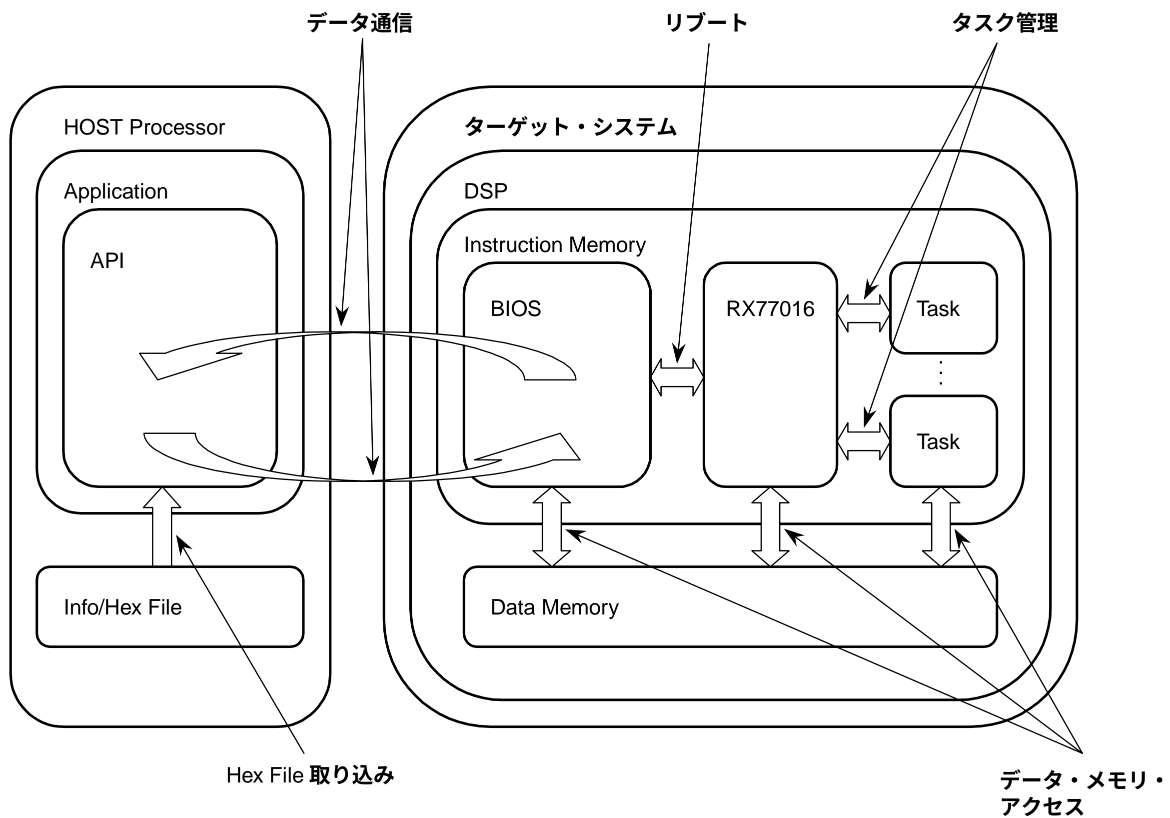
この章では，HOST API システムの構成について説明します。

2.1　ソフトウエア構成

HOST API システムは，Host Processor 上に存在し，ユーザ・アプリケーションと DSP 間のインタフェース部分である API ファンクションと，DSP 上に存在し，HOST API の機能の実体である BIOS ファンクションの 2 種類のコードで構成されています。それら 2 コード間のデータの受け渡しによってリブート，データ・メモリ・アクセス，タスク管理などを行います。

図 2－1 にソフトウエア構成のイメージ図を示します。

図 2－1　ソフトウエア構成イメージ



備考　イメージ図の Host Processor にはパーソナル・コンピュータを想定しています。

API ファンクションは，C 言語の関数として提供され，制御の母体となるユーザ・アプリケーションにコールされる形で使用されます。ユーザ・アプリケーションにコールされた API ファンクションは，その機能を実現するために，DSP に対してコマンドを送信します。

BIOS ファンクションは，μPD77016 ファミリのアセンブリ言語の関数として提供され，API ファンクションが発行するコマンドをデコードした結果にしたがって，どのファンクションをコールするか決定します。

なお，このアプリケーション・ノートでは，API ファンクションを表現する場合，C 言語の関数であることを示す“ () ”を付加し，BIOS ファンクションを表現する場合，μPD77016 ファミリのアセンブリ言語のレーベルであることを示す“ : ”を付加しています。

例　API ファンクションの“ OpenDEV ”→“ OpenDEV() ”と表します。
BIOS ファンクションの“ EchoWord ”→“ EchoWord: ”と表します。

（1）API ファンクション

API ファンクションは次に示す 6 ファイルで構成されています。API ファンクションをコールしたユーザ・アプリケーションを C コンパイラでコンパイルし，mos_hapi.c，spx.c のオブジェクト・ファイル（先にコンパイルしておく必要があります）とリンクすると，実行ファイルを作成できます。

hostapi.h	…	ユーザ・アプリケーションが API ファンクションをコールする場合に必要なヘッダ・ファイル
hapiucfg.h	…	mos_hapi.c，spx.c のユーザ定義用ヘッダ・ファイル
mos_hapi.h	…	mos_hapi.c のヘッダ・ファイル
mos_hapi.c	…	API ファンクションのソース・ファイル
spx.h	…	spx.c のヘッダ・ファイル
spx.c	…	API ファンクションと DSP 間のインタフェース部分のソース・ファイル

（2）BIOS ファンクション

BIOS ファンクションは次に示す 4 ファイルで構成されています。RX77016 のターゲット・システム用初期化部（_MOS_TargetSysInit:）と HI 割り込みハンドラ・コード部（_MOS_ivHI:）に必要なコード（**8.2.3 HI 割り込みハンドラ・コード部（_MOS_ivHI:）**参照）を挿入し，WB77016 でアセンブルしたあと，bios_fns.asm のオブジェクト・ファイル（先にアセンブルしておく必要があります）とリンクすると，実行ファイルを作成できます。

bios_fns.h	…	RX77016 のターゲット・システム用初期化部（_MOS_TargetSysInit:）と HI 割り込みハンドラ・コード部（_MOS_ivHI:）用のヘッダ・ファイル
bios_mac.h	…	RX77016 のターゲット・システム用初期化部（_MOS_TargetSysInit:）と bios_fns.asm 用のマクロ定義ファイル
biosucfg.h	…	RX77016 のターゲット・システム用初期化部（_MOS_TargetSysInit:）と HI 割り込みハンドラ・コード部（_MOS_ivHI:）と bios_fns.asm 用のヘッダ・ファイル
bios_fns.asm	…	BIOS ファンクションのソース・ファイル

2.2　ハードウエア構成

HOST API システムでは，HOST Processor（パーソナル・コンピュータまたは V85x などのマイコン）からシングル・デバイス構成の DSP に対してのみ制御を行います。

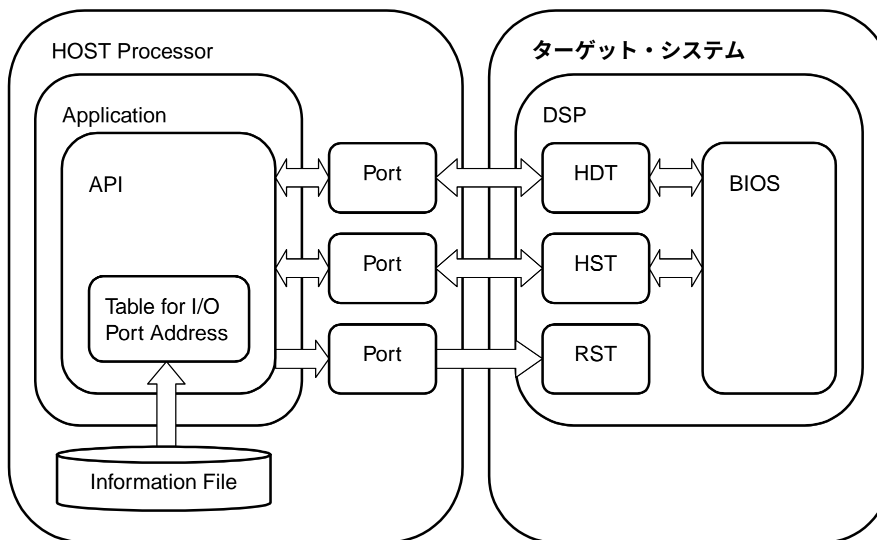
ただし，将来 RX77016 のマルチ・デバイス対応化に伴い，HOST API システムのバージョン・アップを行うとき，ユーザ・プログラムを変更することなく新バージョンへ移行できるように設計されています。

データ通信は，Host Processor の I/O ポートと DSP のホスト・インタフェース間[注]で行います。使用する I/O ポート・アドレスは，HOST API の情報ファイルである Information File（**4.1　Information File** 参照）の IO RESOURCES Section に記述しておき，LoadInformationFile()によって取得し，バッファに蓄積されます。DSP と何らかの通信を行う API ファンクションは，このバッファの情報を利用し，データ通信を行います。

図 2－2 に，ハードウエア構成のイメージ図を示します。

注　HOST API の BIOS ファンクションを DSP に搭載した場合，ホスト・データ・レジスタ（HDT レジスタ）を占有するため，他のアプリケーションが HDT レジスタを使用できません。HOST API では，Host Processor と DSP 間のすべてのデータ通信が HOST API を介して行われることを前提にしています。

図 2－2　ハードウエア構成イメージ



備考　イメージ図の Host Processor にはパーソナル・コンピュータを想定しています。

DSP のホスト・インタフェースであるホスト・データ・レジスタ（HDT レジスタ）とホスト・インタフェース・ステータス・レジスタ（HST レジスタ）にアクセスするには，バイト・アクセスとワード・アクセスの 2 種類の方法注があります。

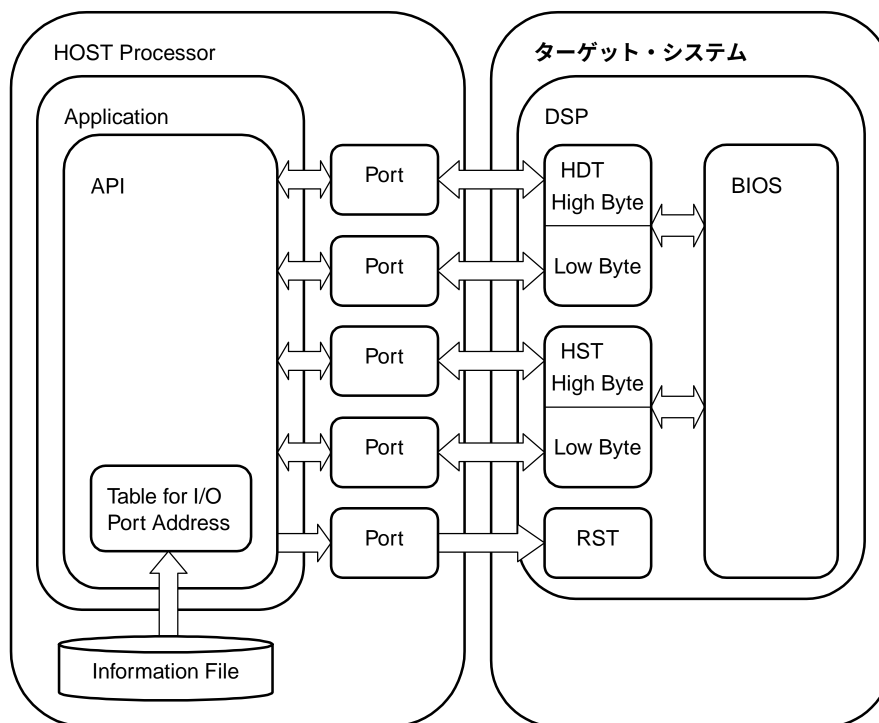
バイト・アクセスは，16 ビット（1 ワード）を上位 8 ビット（High Byte）と下位 8 ビット（Low Byte）に分割し，それぞれ別の I/O ポートを使用してデータのリード／ライトを行います。

ワード・アクセスは，16 ビット（1 ワード）を 1 つの I/O ポートを使用してデータのリード／ライトを行います。

図 2－3 にバイト・アクセス時のイメージ図を示します。

注　hapiucfg.h の HDTAccess，HSTAccess，RSTAccess 定義による条件コンパイルによってアクセス方法を切り替えるため，アクセス方法を切り替えるには再度コンパイルする必要があります。

図 2－3　バイト・アクセス・イメージ



備考　イメージ図の Host Processor にはパーソナル・コンピュータを想定しています。

第 3 章　HOST API の機能

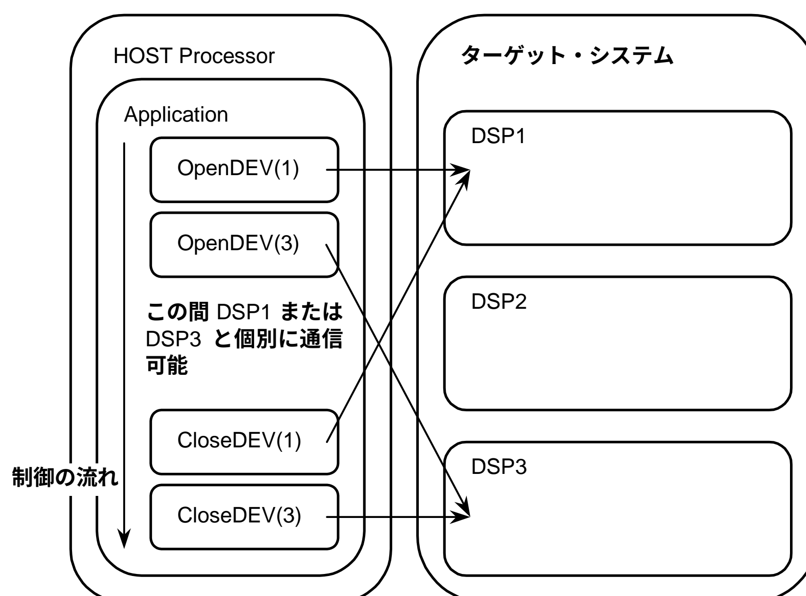
この章では，HOST API システムの機能について説明します。

3.1　通信経路の確立，遮断

HOST API システムでは，OpenDEV()により特定の DSP との通信経路の確立を行い，以降 CloseDEV()により遮断を行うまで，DSP とのデータ通信を行えます。OpenDEV()は，通信経路を確立した際，特定の DSP に対応するデバイス・ハンドルを割り当てます。すべての API ファンクションは，デバイス・ハンドルを使用して，特定の DSP とのデータ通信を行います。

図 3－1 に，RX77016，HOST API がマルチ・デバイス対応になった場合の通信経路の確立，遮断イメージ図を示します。

図 3－1　通信経路の確立，遮断イメージ（マルチ・デバイス対応）



備考　イメージ図の Host Processor にはパーソナル・コンピュータを想定しています。

注意　**本機能は将来 RX77016 がマルチ・デバイス対応になったときに真価を発揮するもので，現バージョンでは，そのときにユーザ・プログラムの変更を最小限に押さえるために存在します（現バージョンの HOST API はマルチ・デバイスに対応していません）。**

3.2　OS 通信用領域へのデータ設定

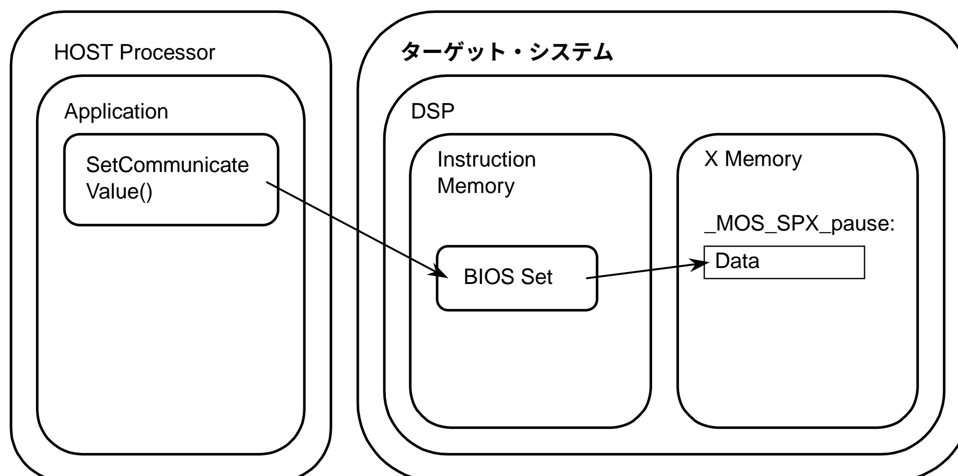
HOST API システムでは，SetCommunicateValue()により，RX77016 の OS 通信用領域（*_MOS_SPX_pause:X）へ任意のデータを設定できます。これにより，RX77016 のシステム起動後の一時停止[注]を解除できます。

OS 通信用領域へのデータ設定は，将来 RX77016 または個々のアプリケーションが外部から何らかのデータ通信を行う場合に必要になります。

図 3－2 に OS 通信領域へのデータ設定のイメージ図を示します。

注　RX77016 のシステム起動後の一時停止とは，OS のすべての初期処理が完了すると外部（HOST Processor など）からの指示があるまで，次の処理（アプリケーション・プログラム開始）の実行を保留している状態のことを表します。*_MOS_SPX_pause:X が 0xffff のとき一時停止を行い，それ以外の値を書き込んだとき，解除されます。ただし，初期状態に 0xffff 以外を記述した場合，一時停止することなく次の処理を実行します。

図 3－2　OS 通信領域へのデータ設定イメージ



備考　イメージ図の Host Processor にはパーソナル・コンピュータを想定しています。

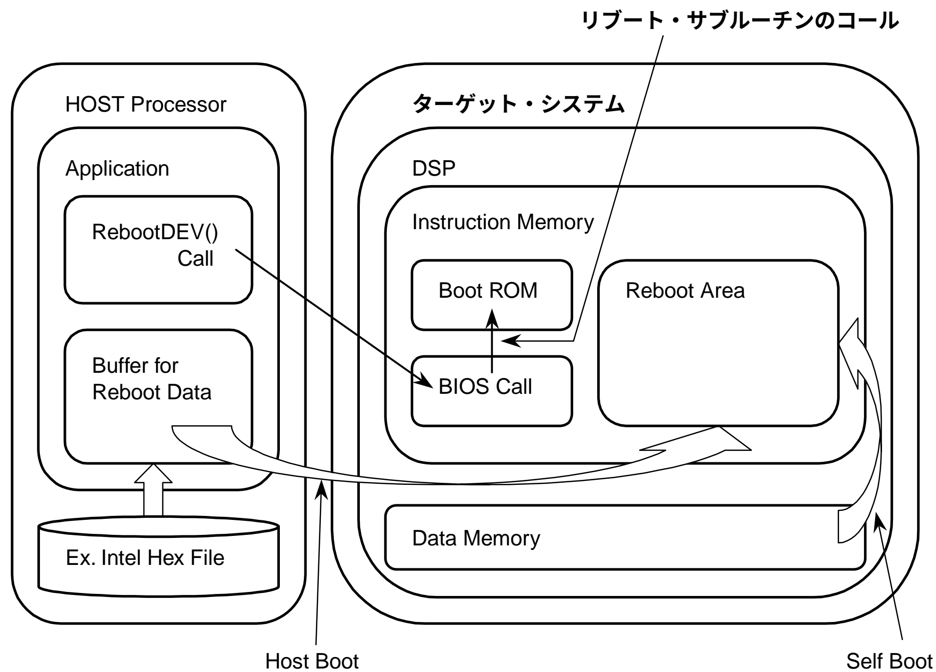
3.3　リブート

HOST API システムでは，RebootDEV()により，DSP のブート ROM のリブート・サブルーチンを利用して，X ワード・リブート，X バイト・リブート，Y ワード・リブート，Y ワード・リブート，ホスト・リブートを行えます。

また，ホスト・リブートでは，HexToBuf()ファンクションを使用して，拡張 Intel HEX File をデータ・ソースとすることも可能です（ただし，Host Processor にパーソナル・コンピュータを選択する場合にのみ可能です）。

図 3－3 にリブートのイメージ図を示します。

図 3－3　リブート・イメージ



備考　イメージ図の Host Processor にはパーソナル・コンピュータを想定しています。

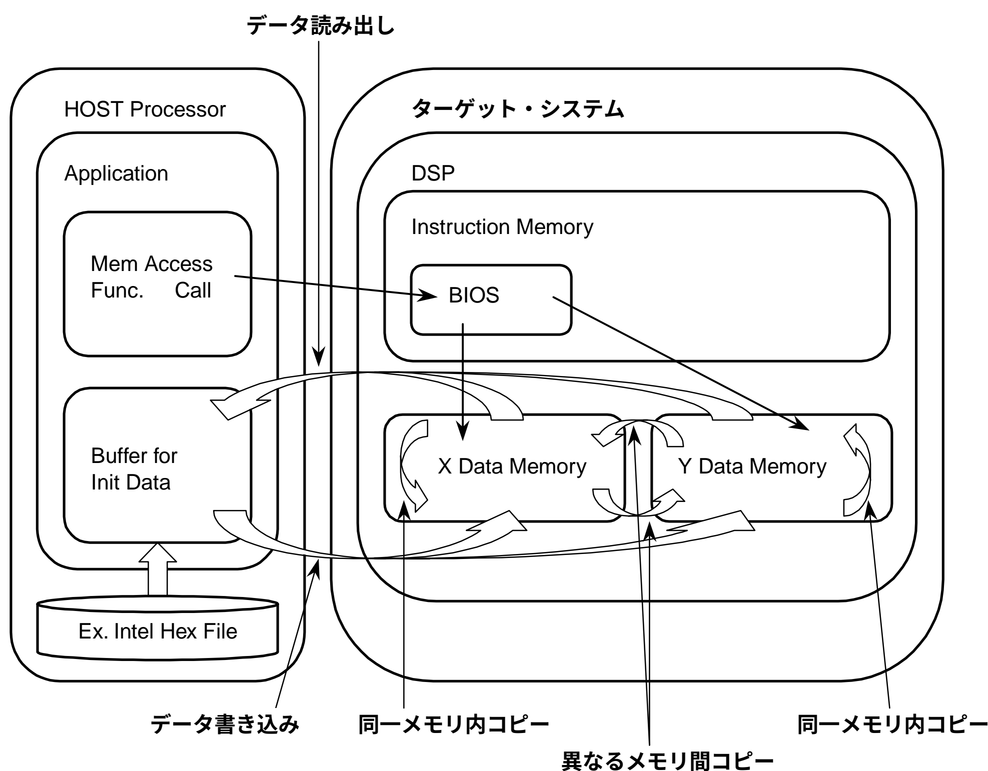
3.4　データ・メモリ・アクセス

HOST API システムでは，ReadFromTMem()，WriteToTMem()，CopyTMemToHost()，CopyHostToTMem()，CopyTMemToTMem()のいずれかにより，データ・メモリの任意のアドレスに対してデータの書き込み／読み出し，同一メモリ内のデータのコピー，異なるメモリ間のデータのコピーを行えます。

また，CopyHostToTMem()では，HexToBuf()ファンクションを使用して，拡張 Intel HEX File をデータ・ソースとすることも可能です（ただし，Host Processor にパーソナル・コンピュータを選択する場合のみ可能です）。

図 3－4 にデータ・メモリ・アクセスのイメージ図を示します。

図 3－4　データ・メモリ・アクセス・イメージ



備考　イメージ図の Host Processor にはパーソナル・コンピュータを想定しています。

3.5　コミュニケーション・バッファ

HOST API システムでは，DSP の X または Y メモリ上にコミュニケーション・バッファと呼ばれる領域を作成して，ユーザ・アプリケーションと DSP 上のアプリケーション間でデータ通信を行えます。

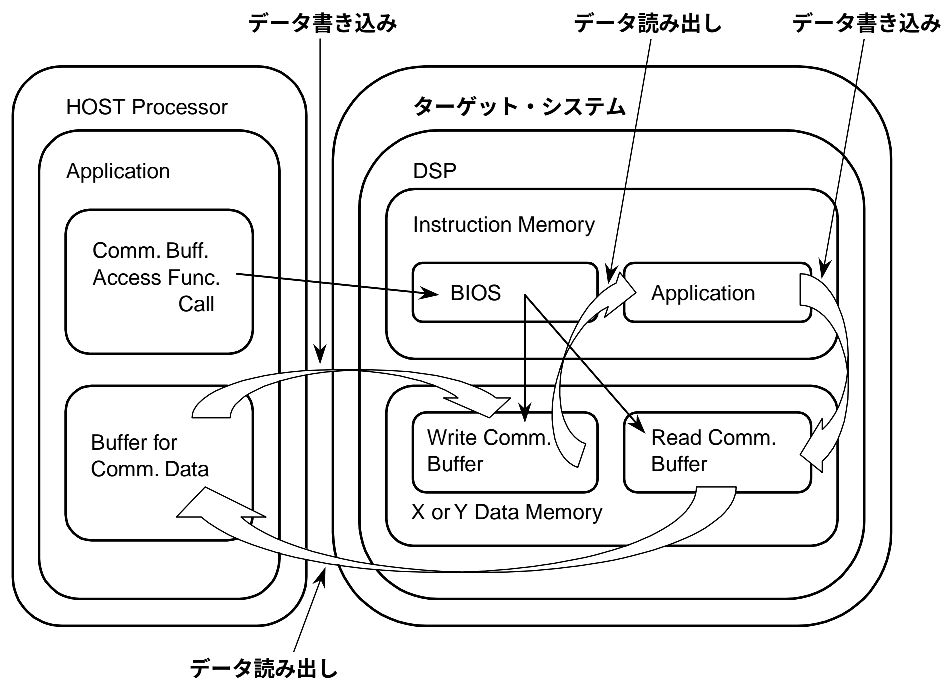
コミュニケーション・バッファには，ユーザ・アプリケーションから DSP 上のアプリケーションへのデータ転送に使用するライト・コミュニケーション・バッファ，DSP 上のアプリケーションからユーザ・アプリケーションへのデータ転送に使用するリード・コミュニケーション・バッファの 2 種類があり，そのサイズ[注]はユーザが自由に設定できます。

また，コミュニケーション・バッファは X または Y データ・メモリの任意のアドレスに対してもデータ通信を行えます。

図 3－5 にコミュニケーション・バッファのイメージ図を示します。

注　biosucfg.h の HICBUF_HEAP_SIZE，HOCBUF_HEAP_SIZE を定義することにより，コミュニケーション・バッファ・サイズを指定します。

図 3－5　コミュニケーション・バッファ・イメージ



備考　イメージ図の Host Processor にはパーソナル・コンピュータを想定しています。

3.5.1　コミュニケーション・バッファ領域の確保と解放

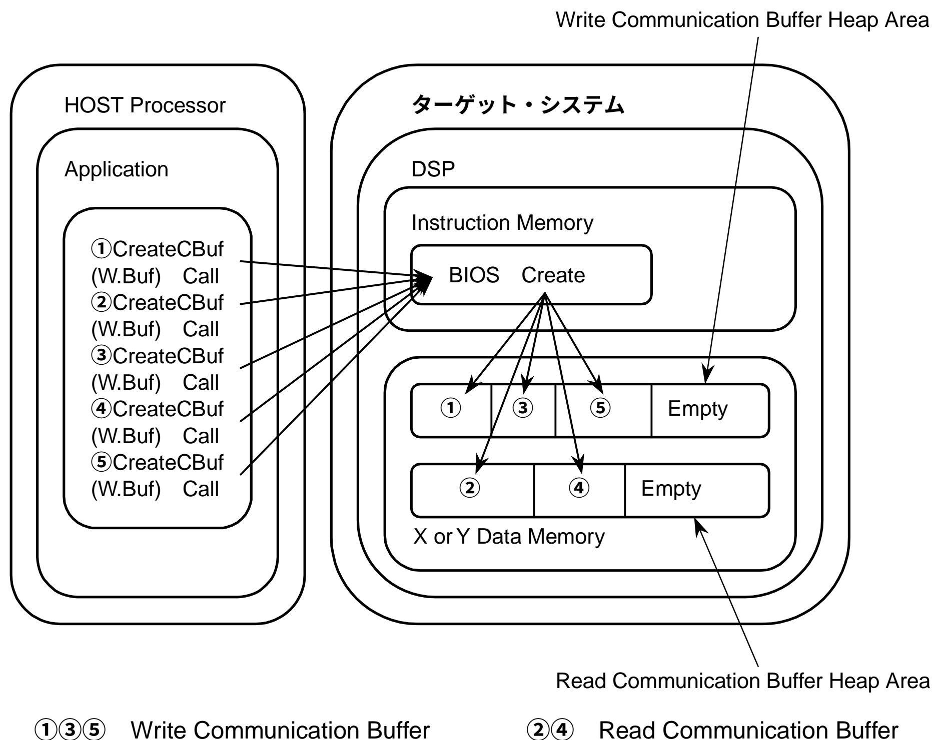
コミュニケーション・バッファでは，あらかじめ確保[注]しておいたヒープ領域（リード用／ライト用別に存在する）内に，使用領域を必要に応じて小出しに確保／開放できます。CreateCBuf()を使用して使用領域確保を，FreeCBuf()を使用して使用領域解放を行います。

CreateCBuf()による使用領域確保では，すでに確保済みのサイズと新規に確保するサイズの和が残りのヒープ領域のサイズを越えない範囲，かつすでに確保済みのバッファ数＋1 が予定したバッファ数を越えない範囲であれば，自由に範囲を設定できます。また，領域を確保すると，その領域に対応するハンドルを割り当てます。以降，そのハンドルを使用して特定の領域をアクセスします。

図 3－6 にコミュニケーション・バッファ領域確保のイメージ図を示します。

注　biosucfg.h の HICBUF_HEAP_SIZE，NUM_HICBUF，HOCBUF_HEAP_SIZE，NUM_HOCBUF を定義してアセンブルすると自動的に必要な領域を確保します。

図 3－6　コミュニケーション・バッファ領域確保イメージ



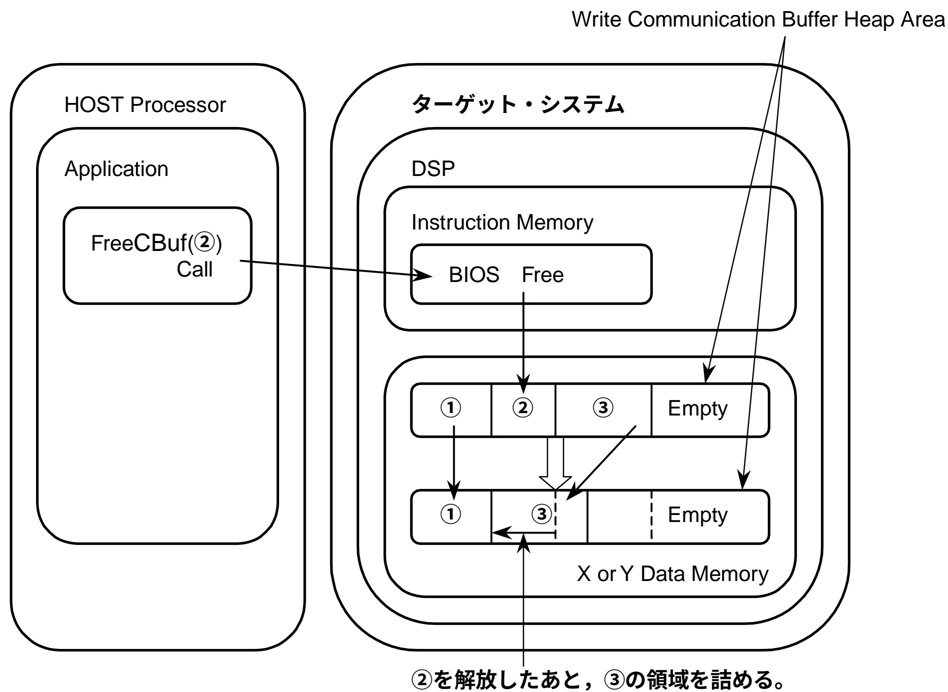
備考　イメージ図の Host Processor にはパーソナル・コンピュータを想定しています。

FreeCBuf()による領域解放では，既存の使用領域を解放し，再利用可能な領域に変更します。また，領域を開放したあと，使用領域がヒープ領域の先頭から連続して配置されるように，メモリ・コンパクションを行います。

たとえば，図 3－7 の②の使用領域を開放すると①と③の間にブランクができるので，③の位置を移動してもともと②があった位置のブランクを埋めます。

図 3－7 に使用領域開放のイメージ図を示します。

図 3－7　使用領域開放イメージ



備考　イメージ図の Host Processor にはパーソナル・コンピュータを想定しています。

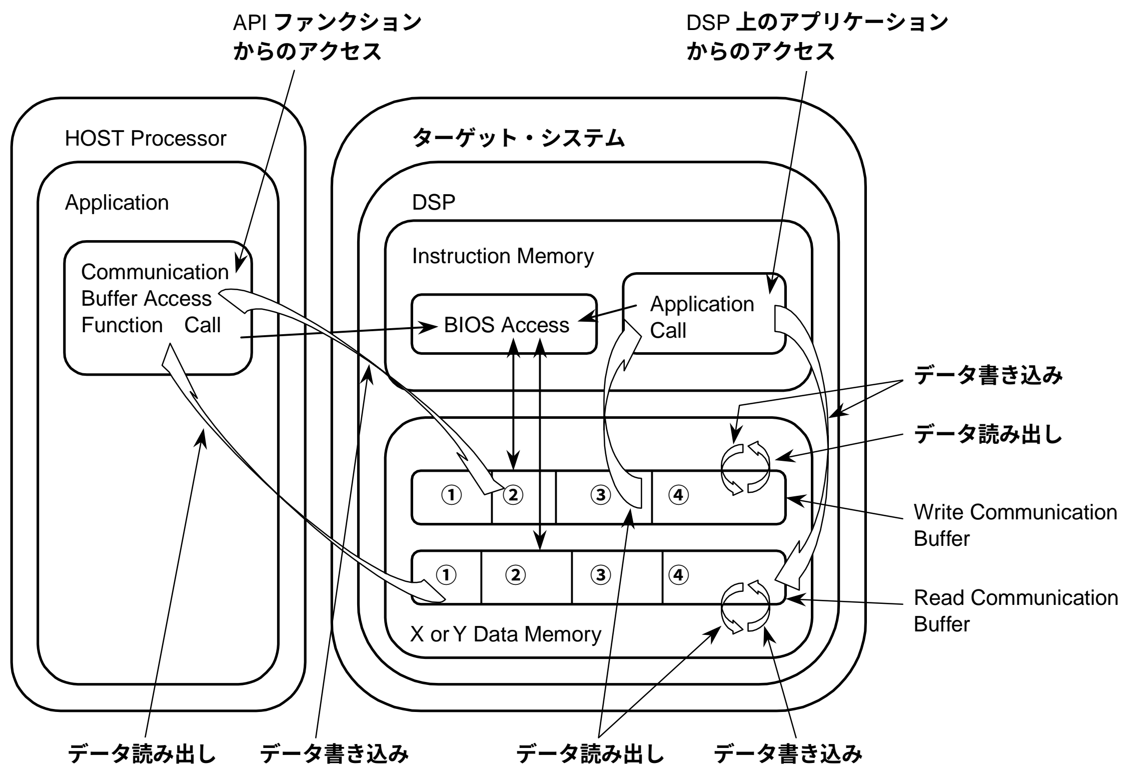
3.5.2　コミュニケーション・バッファ・アクセス

コミュニケーション・バッファには，2 種類（API ファンクションからのアクセスと DSP 上のアプリケーションからのアクセス）のアクセス方法があります。

API ファンクションからのアクセスには，InitCBuf()，ReadFromCBuf()，WriteToCBuf()，CopyCBufToHost()，CopyHostToCBuf()，CopyCBufToTMem()，CopyTMemToCBuf()のいずれかをコールし，DSP 上のアプリケーションからのアクセスには，ReadFromCBuf:または WriteToCBuf:をコールします（それぞれのファンクションの詳細は第 6 章　API ファンクション，第 7 章　BIOS ファンクションを参照してください）。

図 3－8 にコミュニケーション・バッファ・アクセス（カレントが②の場合）のイメージ図を示します。

図 3－8　コミュニケーション・バッファ・アクセス・イメージ



備考　イメージ図の Host Processor にはパーソナル・コンピュータを想定しています。

ライト・コミュニケーション・バッファの基本的なアクセス手順は，次のとおりです。

① CreateCBuf()にて，使用領域を確保する。

② InitCBuf()，WriteToCBuf()，CopyHostToCBuf()，CopyTMemToCBuf()のいずれかにて，①で確保した使用領域に対しデータを書き込む。

③ CopyCBufToTMem()，ReadFromCBuf:のいずれかにて②でデータを書き込んだ使用領域のデータを読み出す。

また，リード・コミュニケーション・バッファの基本的なアクセス手順は，次のとおりです。

① CreateCBuf()にて，使用領域を確保する。

② CopyTMemToCBuf()，WriteToCBuf:のいずれかにて①で確保した使用領域に対してデータを書き込む。

③ ReadFromCBuf()，CopyCBufToHost()，CopyCBufToTMem()のいずれかにて②でデータを書き込んだ使用領域のデータを読み出す。

ただし，コミュニケーション・バッファにはその性格上，アクセス可能な場合とアクセス不可能な場合があります。次にその一覧表を示します。

表3－1　コミュニケーション・バッファ・アクセス状態一覧

使用領域の状態 / ファンクション	ライト・コミュニケーション・バッファ		リード・コミュニケーション・バッファ	
	書き込んだデータは読み出されている	書き込んだデータは読み出されていない	書き込んだデータは読み出されている	書き込んだデータは読み出されていない
InitCBuf()	○	×	－	－
ReadFromCBuf()	－	－	×	○
WriteToCBuf()	○	×	－	－
CopyCBufToHost()	－	－	×	○
CopyHostToCBuf()	○	×	－	－
CopyCBufToTMem()	×	○	×	○
CopyTMemToCBuf()	○	×	○	×
ReadFromCBuf:	×	○	－	－
WriteToCBuf:	－	－	○	×

備考　○はアクセス可能，×はアクセス不可能，－はアクセス禁止を示します。

API ファンクションまたは BIOS ファンクションが，アクセス不可能なコミュニケーション・バッファをアクセスしようとした場合，戻り値にエラー値を返すため，その値を判断してその後の処理を決定できます。

3.6　タスク管理

HOST API システムでは，SuspendSyncTask()，ResumeSyncTask()を使用して，動作タスクを停止タスクに，停止タスクを動作タスクに変更できます。

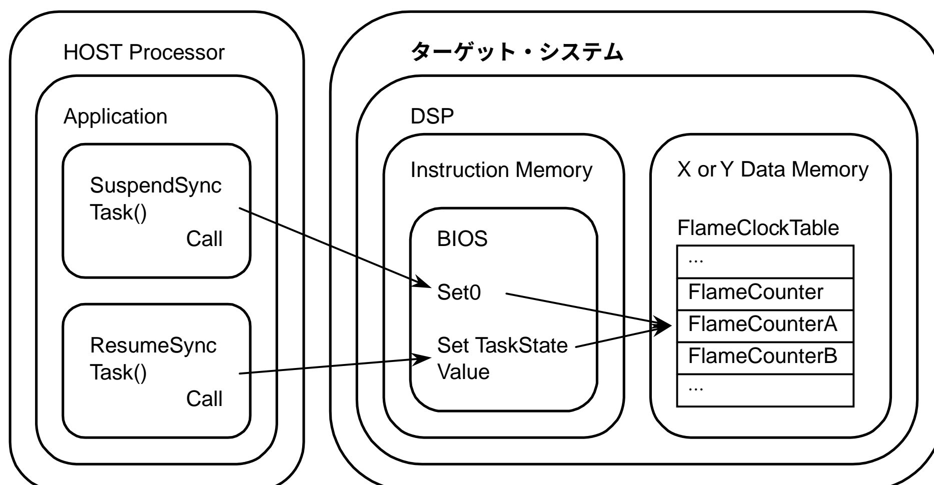
SuspendSyncTask()により，指定タスクのフレーム・カウンタ A 注に 0 を設定します。設定前のフレーム・カウンタ値が 0 以外であった場合，動作タスクを停止タスクに変更したことになります。

ResumeSyncTask()により，RX77016 のフレーム・クロック・テーブルのフレーム・カウンタ A に Information File の SubTaskXXYY セクションの TaskState キーで指定した値を設定します。設定前のフレーム・カウンタ A 値が 0 であった場合，停止タスクを動作タスクに変更したことになります。

図 3－9 にタスク管理のイメージ図を示します。

注　フレーム・カウンタ値に対して，減算する値です。減算はタイマを参照し，一定周期で行われます（詳細は **RX77016　ユーザーズ・マニュアル　機能編**を参照してください）。

図 3－9　タスク管理イメージ



備考　イメージ図の Host Processor にはパーソナル・コンピュータを想定しています。

第 4 章　入力ファイル

4.1　Information File

HOST API は，I/O ポート・アドレス，コミュニケーション・バッファ情報，タスク情報などを得るために，Information File を取り込みます。Information File は Windows の INI 形式のファイルであり，次に示すセクションとキーによって定義されます。

4.1.1　Device セクション

(1) デバイス識別名の指定

• DeviceDescription キー

デバイス識別名を 31 文字以内（2 バイト文字は 2 文字と数える）で指定します。ここで指定した識別名は BIOS を通信可能モードに変更（OpenDEV()のコール）する際のターゲット・デバイス名となります。32 文字以上のデバイス識別名を指定した場合，32 文字以降は切り捨てられます。

次の記述例では，デバイス識別名に“SPX0”を指定しています。

DeviceDescription = SPX0

4.1.2　IO RESOURCE セクション

(1) HDT レジスタの I/O ポート・アドレスの指定

HDT レジスタに対してバイト・アクセスを行うか，ワード・アクセスを行うかによって IOPortHDTLow，IOPortHDTHigh キーの両方，または IOPortHDT キーのいずれかを使用します。

• IOPortHDTLow，IOPortHDTHigh キー

HDT レジスタに対しバイト・アクセスを行う場合の I/O ポート・アドレスを 16 進数 4 桁以内で指定します。IOPortHDTLow に HDT レジスタの 0～7 ビット，IOPortHDTHigh に 8～15 ビットの I/O ポート・アドレスを指定します。

次の記述例では，IOPortHDTLow に 0x8020，IOPortHDTHigh に 0x8021 を指定しています。

IOPortHDTLow = 0x8020

IOPortHDTHigh = 0x8021

• IOPortHDT キー

HDT レジスタに対しワード・アクセスを行う場合の I/O ポート・アドレスを 16 進数 4 桁以内で指定します。IOPortHDT キー定義以降の IOPortHDTLow キー定義と IOPortHDTHigh キー定義は無効となります。

次の記述例では，IOPortHST に 0x8020 を指定しています。

IOPortHDT = 0x8020

（2）HST レジスタの I/O ポート・アドレスの指定

HST レジスタに対してバイト・アクセスを行うか，ワード・アクセスを行うかによって IOPortHSTLow，IOPortHSTHigh キーの両方，または IOPortHST キーのいずれかを使用します。

・IOPortHSTLow，IOPortHSTHigh キー

HST レジスタに対しバイト・アクセスを行う場合の I/O ポート・アドレスを 16 進数 4 桁以内で指定します。IOPortHSTLow に HST レジスタの 0〜7 ビット，IOPortHSTHigh に 8〜15 ビットの I/O ポート・アドレスを指定します。

次の記述例では，IOPortHSTLow に 0x8022，IOPortHSTHigh に 0x8023 を指定しています

```
IOPortHSTLow = 0x8022
IOPortHSTHigh = 0x8023
```

・IOPortHST キー

HST レジスタに対しワード・アクセスを行う場合の I/O ポート・アドレスを 16 進数 4 桁以内で指定します。

次の記述例では，IOPortHST に 0x8022 を指定しています。

```
IOPortHST = 0x8022
```

（3）RST 端子をアクセスする I/O ポート・アドレスの指定

RST 端子をアクセスする I/O ポートに対してバイト・アクセスを行うか，ワード・アクセスを行うかによって IOPortRSTLow，IOPortRSTHigh キーの両方，または IOPortRST キーのいずれかを使用します。

・IOPortRSTLow，　IOPortRSTHigh キー

RST 端子をアクセスする I/O ポートに対し，バイト・アクセスを行う場合の I/O ポート・アドレスを 16 進数 4 桁以内で指定します。IOPortRSTLow に RST 端子を含むワードの 0〜7 ビットを，IOPortRSTHigh に 8〜15 ビットを指定します。

次の記述例では，IOPortRSTLow に 0x8024，IOPortRSTHigh に 0x8025 を指定しています。

```
IOPortRSTLow = 0x8024
IOPortRSTHigh = 0x8025
```

・IOPortRST キー

RST 端子をアクセスする I/O ポートに対し，ワード・アクセスを行う場合の I/O ポート・アドレスを 16 進数 4 桁以内で指定します。

次の記述例では，IOPortRST に 0x8024 を指定しています。

```
IOPortRST=0x8024
```

（4）RST 端子のビット・アサインの指定

RST 端子をアクセスする I/O ポートに対してバイト・アクセスを行うか，ワード・アクセスを行うかによって IOPortRSTLow，IOPortRSTHigh キーの両方，または IOPortRST キーのいずれかを使用します。

・BitAssignRSTLow，BitAssignRSTHigh キー

RST 端子をアクセスする I/O ポートに対し，バイト・アクセスを行う場合の RST 端子のビット位置を 16 進数 2 桁以内で指定します。BitAssignRSTLow に RST 端子を含むワードの 0〜7 ビットのビット・アサインを，BitAssignRSTHigh に 8〜15 ビットのビット・アサインを指定します。該当する位置に 1 を立て，それ以外は 0 を指定します。

次の記述例では，BitAssignRSTLow に 0x01 を BitAssignRSTHigh に 0x00 を指定しています。

```
BitAssignRSTLow = 0x01
BitAssignRSTHigh = 0x00
```

・BitAssignRST キー

RST 端子をアクセスする I/O ポートに対し，ワード・アクセスを行う場合の RST 端子のビット位置を 16 進数 4 桁以内で指定します。該当する位置に 1 を立てそれ以外は 0 を指定します。

次の記述例では，BitAssignRST に 0x0001 を指定しています。

```
BitAssignRST = 0x0001
```

（5）RST 端子をアクセスする I/O ポートのマスク・データ

RST 端子をアクセスする I/O ポートに対してバイト・アクセスを行うか，ワード・アクセスを行うかによって MaskDataRSTLow，MaskDataRSTHigh キーの両方，または MaskDataRST キーのいずれかを使用します。

・MaskDataRSTLow，MaskDataRSTHigh キー

RST 端子をアクセスする I/O ポートに対し，バイト・アクセスを行う場合の RST 端子以外の OR マスク・データを 16 進数 2 桁以内で指定します。MaskDataRSTLow に RST 端子を含むワードの 0〜7 ビットのマスク・データを，MaskDataRSTHigh に 8〜15 ビットのマスク・データを指定します（RST 端子へのアクセスは，RST 端子のビット位置に 1 を立ててもマスクされません）。

次の記述例では，MaskDataRSTLow に 0x30，MaskDataRSTHigh に 0x05 を指定しています。

```
MaskDataRSTLow = 0x30
MaskDataRSTHigh = 0x05
```

・MaskDataRST キー

RST 端子をアクセスする I/O ポートに対し，ワード・アクセスを行う場合の RST 端子以外の OR マスク・データを 16 進数 4 桁以内で指定します（RST 端子へのアクセスは，RST 端子のビット位置に 1 を立ててもマスクされません）。

次の記述例では，MaskDataRST に 0x3005 を指定しています。

```
MaskDataRST = 0x3005
```

4.1.3　Task セクション

（1）タスク数指定

• nTask キー

ターゲット・デバイス上の RX77016 が制御するタスク数を 10 進数 4 桁以内で指定します。

次の記述例では，nTask に 2 を指定しています。

```
nTask = 2
```

4.1.4　TaskXX セクション

TaskXX セクションでは，nTask キーで指定した数のセクションを記述する必要があります。XX には 01 を先頭とする連続した数を割り当てられます。nTask キーに 3 を指定した場合，XX には 01～03 が割り当てられ，Task01，Task02，Task03 セクションを記述することになります。また，TaskXX は，RX77016 の TaskX と同じタスクを差していなければなりません。

（1）サブタスク数指定

• nSubTask キー

TaskX のサブタスク数を 10 進数 4 桁以内で指定します。

次の記述例では，Task01 の nSubTask に 1，Task02 の nSubTask に 3 を指定しています。

```
[Task01]
  nSubTask = 1
[Task02]
  nSubTask = 3
```

4.1.5　SubTaskXXYY セクション

SubTaskXXYY セクションでは，TaskXX セクションの nSubTask キーで指定した数のセクションを記述する必要があります。XX には TaskXX セクションと同じ数値が割り当てられます。YY には nSubTask キーに指定した数に応じて 00 を先頭とする連続した数が割り当てられます。

Task セクションの nTask キーに 2 を指定し，Task01 セクションの nSubTask キーに 2，task02 セクションの nSubTask キーに 3 を指定した場合，XXYY に 0100，0101，0200，0201，0202 が割り当てられ，SubTask0100，SubTask0101，SubTask0200，SubTask0201，SubTask0202 セクションを記述することになります。また，SubTaskXXYY は，RX77016 の SubTaskXY と同じタスクを差していなければなりません。

（1）サブタスク名の指定

• TaskName キー

SubTaskXXYY のサブタスク名を 31 文字以内（2 バイト文字は 2 文字と数える）で指定します。32 文字以上のサブタスク名を指定した場合は，それ以降を切り捨てます。

次の記述例では，SubTask0101 セクションの TaskName に " SWAPIO " を指定しています。

```
[SubTask0101]
  TaskName = SWAPIO
```

（2）サブタスク・ステートの指定

・TaskState **キー**

SubTaskXXYY **のサブタスク・ステート**[注]**を** 16 **進数** 4 **桁以内で指定します。**

次の記述例では，SubTask0101 **セクションの** TaskState **に** 1 **を指定しています。**

```
[SubTask0101]
  TaskState = 1
```

注　ResumeSyncTask()**が** RX77016 **のフレーム・クロック・テーブルのフレーム・カウンタ** A **に設定する値です。**

4.1.6　Information File の記述例

図 4－1 **に，**Information File **の記述例を示します。**

図 4－1　Information File の記述例

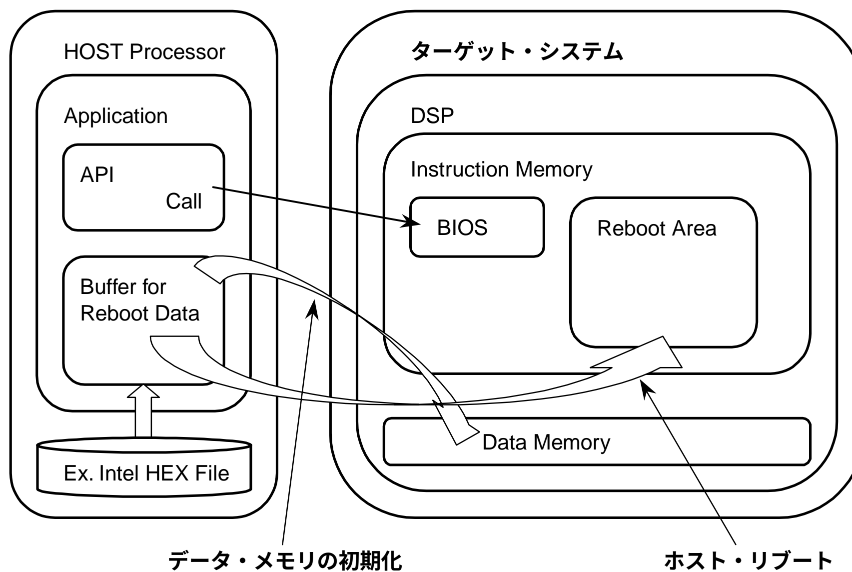
```
;**********************************************************
;* uPD77016 series processor information file For example  *
;**********************************************************
[Device]
DeviceDescription=SPX0
[IO RESOURCE]
IOPortHDT=0x8020
IOPortHST=0x8022
IOPortRST=0x8024
BitAssignRST=0x0001
MaskDataRST=0x3005
[Task]
nTask =2
[Task01]
nSubTask=1
[Task02]
nSubTask=3
[SubTask0100]
TaskName=SWAPIO
TaskState=1
[SubTask0200]
TaskName=REVERB
TaskState=1
[SubTask0201]
TaskName=CHORUS
TaskState=1
[SubTask0202]
TaskName=SURROUND
TaskState=1
```

4.2　HEX ファイル

HOST API は DSP に対して，インストラクションまたはデータをリブートする際に，ロード・データをあらかじめバッファに蓄積しておく必要があります。そのデータ・ソースとして拡張 Intel HEX File（WB77016 の出力である.HXI，.HDX，.HDY ファイルなど）を指定することができます。HOST API システムでは，拡張 Intel HEX File のデータ・ソースを指定バッファに蓄積する API ファンクションをサポートしています。

図 4－2 に拡張 Intel HEX File を利用したリブートのイメージ図を示します。

図 4－2　拡張 Intel HEX File を利用したリブート・イメージ



備考　イメージ図の Host Processor にはパーソナル・コンピュータを想定しています。

第 5 章　HOST API の仕組み

5.1　BIOS モード

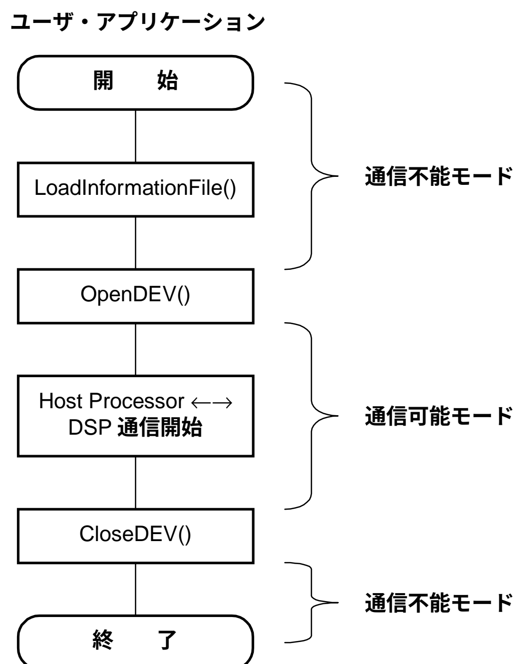
BIOS には，通信可能モードと通信不能モードの 2 種類のモードが存在します。

通信可能モードとは，API ファンクションと BIOS ファンクション間で自由に通信を行えるモードのことです。通信不能時に OpenDEV()をコールすることにより通信経路が確立され，このモードへ変化します。このとき，API ファンクションによって書き込まれた HDT レジスタのデータを受け取った BIOS ファンクションは，そのデータをコマンドまたはパラメータと判断し，適切な処理を行います。

通信不能モードとは，API ファンクションと BIOS ファンクション間で通信を行えないモードのことです。通信可能時に CloseDEV()をコールすることにより通信経路が遮断され，このモードへ変化します。また，BIOS の初期状態も通信不能モードです。このとき，HDT レジスタに書き込まれたデータは意味のないものと判断され，デコードは行われません。

図 5－1 にユーザ・アプリケーションが OpenDEV()，CloseDEV()をコールしたときの BIOS モードの変化を示します。

図 5－1　OpenDEV()，CloseDEV()をコールしたときの BIOS モード変化



5.2　BIOSファンクション・コール

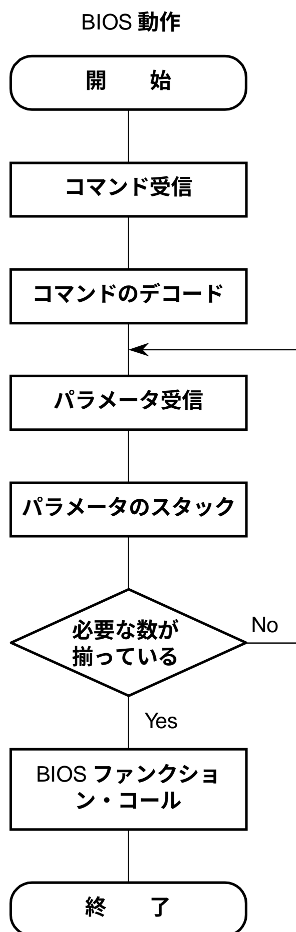
HOST APIシステムでは，HOST API管理ファンクションを除くすべてのAPIファンクションがその機能を実現するBIOSファンクションをコールするために，コマンドとパラメータをDSPに対して送信します。

コマンドとは，意図したBIOSファンクションをコールするためのコードのことで，デコードすることにより，コールするファンクションと必要なパラメータ数が判明します。

パラメータとは，コマンドに対応するファンクションをコールする際に必要なデータのことです。たとえば，メモリ・ライトのライト・アドレスとライト・データがこれに当たります。BIOSは，パラメータをコマンド・バッファと呼ばれるスタック領域にスタックしていき，必要な数が揃った時点で，BIOSファンクション・コールを行います。

図5－2にBIOSカーネルの動作フローを示します。

図5－2　BIOSカーネルの動作フロー



第 6 章　API ファンクション

6.1　データ定義

API ファンクションでは，表 6－1 のデータ型を定義しています。

表 6－1　データ型一覧

定義データ型	C 言語表現のデータ型	用　　途
BYTE	unsigned char	符号なし 1 バイト・データ
LPBYTE	unsigned char*	符号なし 1 バイト・データのポインタ
SWORD	short	符号付き 2 バイト・データ
WORD	unsigned short	符号なし 2 バイト・データ
LPWORD	unsigned short*	符号なし 2 バイト・データのポインタ
DWORD	unsigned long	符号なし 4 バイト・データ
devHnd	unsigned short	デバイス・ハンドル
cbHnd	unsigned short	CB ハンドル
lpcbHnd	unsigned short FAR*	CB ハンドルのポインタ
SubTaskInfo	typedef struct { WORD ID ; BYTE Name[MAX_NAMES] ; WORD FrameSubValue ; WORD FrameSubAddress ; } SubTaskInfo ;	 サブタスク ID サブタスク名 FrameSub 値 FrameSub のアドレス
RebootParam	typedef struct { WORD CmdID ; WORD NumInstSize ; WORD SrcStarAddr ; WORD *SrcPointer ; WORD DestStartAddr ; } RebootParam ;	 リブート・コマンド ID ブート・サイズ ソースの先頭アドレス（セルフ・ブート時） ソースのポインタ（ホスト・ブート時） デスティネーションの先頭アドレス
CBCaps	Typedef struct { WORD StartAddr ; WORD nSize ; WORD RWFlag ; DWORD StructAddr ; } CBCaps ;	 CB スタート・アドレス CB サイズ CB R/W フラグ CB 情報のアドレス

備考　CB：コミュニケーション・バッファ

API ファンクションでは，次の文字定数を定義しています。

表 6－2　文字定数一覧

定義文字列	C 言語表現のデータ	用　　途
SUCCESS	(SWORD)0x0000	API ファンクションの戻り値
FILE_NOTFOUND	(SWORD)0xfff5	〃
SUBTASKTBL_FULL	(SWORD)0xfff6	〃
CBTBL_FULL	(SWORD)0xfff7	〃
CB_FULL	(SWORD)0xfff8	〃
CB_EMPTY	(SWORD)0xfff9	〃
FUNCID_ERROR	(SWORD)0xfffa	〃
ALREADY_OPEN	(SWORD)0xfffb	〃
CBHND_ERROR	(SWORD)0xfffc	〃
DEVHND_ERROR	(SWORD)0xfffd	〃
DEVNAME_ERROR	(SWORD)0xfffe	〃
FAIL	(SWORD)0xffff	〃
_MHA_SELPCMD_CMD	0x8000	コマンド ID
_MHA_IDTAGCMD_CMD	0x4000	〃
_MHA_REBOOT_SUBCMD	0x0000	サブコマンド ID
_MHA_DIRECT_RX_SUBCMD	0x0002	〃
_MHA_DIRECT_RY_SUBCMD	0x0004	〃
_MHA_DIRECT_WX_SUBCMD	0x0006	〃
_MHA_DIRECT_WY_SUBCMD	0x0008	〃
_MHA_COPY_XTOX_SUBCMD	0x000a	〃
_MHA_COPY_YTOY_SUBCMD	0x000c	〃
_MHA_COPY_XTOY_SUBCMD	0x000e	〃
_MHA_COPY_YTOX_SUBCMD	0x0010	〃
_MHA_REBOOT_HOST_CMD	0x0005	リブート・コマンド ID
_MHA_REBOOT_X_BYTE_CMD	0x0004	〃
_MHA_REBOOT_Y_BYTE_CMD	0x0003	〃
_MHA_REBOOT_X_WORD_CMD	0x0002	〃
_MHA_REBOOT_Y_WORD_CMD	0x0001	〃
READ	0x0000	R/W フラグ
WRITE	0x0001	〃
CB_NOTUSED	0xffff	CB 使用状況フラグ
WAIT	0x0000	CB アクセス制限フラグ
NON_WAIT	0x0001	〃
INST	0x0000	Hex ファイル判別フラグ
DATA	0x0001	〃

備考　CB：コミュニケーション・バッファ

6.2　API ファンクションの機能

6.2.1　初期設定ファンクション

（1）OpenDEV()

[Description]

DEV_Name に指定されたデバイスへの通信経路を確立し，そのデバイス・ハンドルを device へ書き込みます。

```
SWORD OpenDEV(
    devHnd *device          /* デバイス・ハンドル格納先のポインタ */
    LPBYTE DEV_Name         /* ターゲット・デバイス名のポインタ */
);
```

本ファンクションをコールすると，BIOS は通信可能モードになります。ターゲット・デバイスと何らかのデータ通信を行う API ファンクションを使用する場合，あらかじめ本ファンクションをコールしておかなければなりません。

DEV_Name には，Information File の Device セクションの DeviceDes キーで指定したデバイス名を記述します。

本ファンクションは将来 HOST API がマルチ・デバイス対応になったときに真価を発揮するものです。現バージョンでは，そのときにユーザ・プログラムの変更を最小限に押さえるために存在します。

[Return Value]

SUCCESS	…	通信経路の確立成功
DEVNAME_ERROR	…	認識不可能なデバイス名の指定
ALREADY_OPEN	…	指定デバイスはすでにオープンしている
FAIL	…	DSP との通信に失敗

[Code Example]

```
int main()
{
   devHnd device1 ;
   if(LoadInformationFile(“devinfo.ini”)!=SUCCESS) return -1 ;
   if(OpenDEV(&device1,”Device1”)!=SUCCESS ){
      puts(“It failed to Open device.”) ;
      return -1 ;
   }
   if(SetCommunicateValue(device1,0x0000)!=SUCCESS) return -1 ;

   if(CloseDEV(device1)!=SUCCESS) return -1 ;
   return 0;
}
```

（2）CloseDEV()

[Description]

ターゲット・デバイスとの通信経路を遮断します。

```
SWORD CloseDEV(
    devHnd device              /* ターゲット・デバイス・ハンドル */
);
```

本ファンクションをコールすると，BIOS は通信不能モードになります。ターゲット・デバイスと何らかのデータ通信を行う API を以降使用しない場合，本ファンクションを使用します。

本ファンクションは将来 HOST API がマルチ・デバイス対応になったときに真価を発揮するものです。現バージョンでは，そのときにユーザ・プログラムの変更を最小限に押さえるために存在します。

[Return Value]

SUCCESS	…	通信経路の遮断成功
DEVHND_ERROR	…	認識不可能なデバイス・ハンドルの指定
FAIL	…	DSP との通信に失敗

[Code Example]

```
int main()
{
   devHnd device1 ;
   if(LoadInformationFile(“devinfo.ini”)!=SUCCESS)  return -1 ;
   if(OpenDEV(&device1,“Device1”)!=SUCCESS )  return -1 ;
   if(SetCommunicateValue(device1,0x0000)!=SUCCESS) return -1 ;

   if(CloseDEV(device1)!=SUCCESS){
      puts(“It failed to Close device.”) ;
      return -1 ;
   }
   return 0;
}
```

(3) RebootDEV()

[Description]

ターゲット・デバイスに対して X ワード／バイト・リブート，Y ワード／バイト・リブートまたは，セルフ・ブートを行います。

```
SWORD RebootDEV(
    devHnd device,                /* ターゲット・デバイス・ハンドル */
    RebootParam * SpxRbtParam     /* RebootParam 構造体のポインタ */
) ;
```

本ファンクションによって，ホスト・ブートを行う場合は，あらかじめ SpxRbtParam → SrcStartAddr を先頭アドレスとするバッファにブート・データを蓄積しておく必要があります。データ・ソースとして拡張 Intel Hex File を使用したい場合，HexToBuf()ファンクションを使用してバッファリングすることが可能です。

[Return Value]

SUCCESS	…	リブート成功
DEVHND_ERROR	…	認識不可能なデバイス・ハンドルの指定
FAIL	…	DSP との通信に失敗

[Code Example]

```
WORD IBuff[]={ 0x0000,0x0000,0x0000,0x0000 } ;
RebootParam RebootParam1={
   _MHA_REBOOT_HOST_CMD, // Reboot Function ID
   2, // Size of reboot instruction
   0, // Start address of source
   IBuff, // Pointer of source
   1024, // Start address of destination
} ;
int main()
{
   devHnd device1 ;
   if(LoadInformationFile("devinfo.ini")!=SUCCESS) return -1 ;
   if(OpenDEV(&device1,"Device1")!=SUCCESS ) return -1 ;
   if(SetCommunicateValue(device1,0x0000)!=SUCCESS) return -1 ;

   if(RebootDEV(device1,&RebootParam1)!=SUCCESS){
      puts("It failed to reboot device.") ;
   }

   if(CloseDEV(device1)!=SUCCESS) return -1 ;
   return 0;
}
```

(4) ResetDEV()

[Description]

ターゲット・デバイスに対し，システム・リセットを行います。

```
SWORD ResetDEV(
    devHnd device                    /* ターゲット・デバイス・ハンドル */
);
```

[Return Value]

SUCCESS　　　…　システム・リセット成功

DEVHND_ERROR　…　認識不可能なデバイス・ハンドルの指定

[Code Example]

```
int main()
{
   devHnd device1 ;
   if(LoadInformationFile("devinfo.ini")!=SUCCESS) return -1 ;
   if(OpenDEV(&device1,"Device1")!=SUCCESS ) return -1 ;
   if(SetCommunicateValue(device1,0x0000)!=SUCCESS) return -1 ;

   if(ResetDev(device1)!=SUCCESS){
      puts("It failed to reset device.") ;
      return -1 ;
   }

   return 0;
}
```

(5) SetCommunicateValue()

[Description]

RX77016 の OS 通信用領域（*_MOS_SPX_pause:X）へ指定したデータを設定します。

```
SWORD SetCommunicateValue(
    devHnd device,          /* ターゲット・デバイス・ハンドル */
    WORD Value              /* 設定データ */
);
```

[Return Value]

SUCCESS	…	設定成功
DEVHND_ERROR	…	認識不可能なデバイス・ハンドルの指定
FAIL	…	DSP との通信に失敗

[Code Example]

```
int main()
{
   devHnd device1 ;
   if(LoadInformationFile("devinfo.ini")!=SUCCESS) return -1 ;
   if(OpenDEV(&device1,"Device1")!=SUCCESS ) return -1 ;
   if(SetCommunicateValue(device1,0x0000)!=SUCCESS){
      puts("It failed to set Communicate Value.") ;
      return -1 ;
   }

   if(CloseDEV(device)!=SUCCESS) return -1 ;
   return 0;
}
```

6.2.2　タスク管理ファンクション

（1）ShowTask()

[Description]

loadInformationFile()で取り込んだタスク情報を表示します。RETURN_TO_WINDOW が定義されていれば Windows 上のメッセージ・ボックスに，定義されていなければコンソール上に表示し，pFunc に Func 構造体のポインタをコピーします。

```
SWORD ShowTask(
    devHnd device,              /* ターゲット・デバイス・ハンドル */
    WORD SubTaskID,             /* サブタスク ID */
    SubTaskInfo* pSubTask       /* Func 構造体のコピー先のポインタ */
) ;
```

表示する項目は，サブタスク名，サブタスク ID，カレント・ステート，フレーム・カウンタ A です。

[Return Value]

SUCCESS	…	表示成功
DEVHND_ERROR	…	認識不可能なデバイス・ハンドルの指定
FUNCID_ERROR	…	認識不可能なサブタスク ID の指定

[Code Example]

```
int main()
{
   Func Func1 ;
   devHnd device1 ;
   if(LoadInformationFile(“devinfo.ini”)!=SUCCESS) return -1 ;
   if(OpenDEV(&device1,“Device1”)!=SUCCESS ) return -1 ;
   if(SetCommunicateValue(device1,0x0000)!=SUCCESS) return -1 ;

   if(ShowTask(&device1,100,&Func1)!=SUCCESS ){
      puts(“It failed to show task information.”) ;
   }

   if(CloseDEV(device1)!=SUCCESS) return -1 ;
   return 0;
}
```

（2）ResumeSyncTask()

[Description]

指定タスクに対応する，RX77016 のフレーム・クロック・テーブルのフレーム・カウンタ A に devInf[device].subtask[SubTaskID]→FrameSubValue 値を書き込みます。元のフレーム・カウンタ A 値が 0 であった場合に 0 以外を書き込むと，そのタスクは実行可能タスクとなります。

```
SWORD ResumeSyncTask(
    devHnd device,            /* ターゲット・デバイス・ハンドル */
    WORD SubTaskID            /* サブタスク ID */
) ;
```

[Return Value]

SUCCESS … フレーム・カウンタ A 値の更新成功
DEVHND_ERROR … 認識不可能なデバイス・ハンドルの指定
FUNCID_ERROR … 認識不可能なサブタスク ID の指定
FAIL … DSP との通信に失敗

[Code Example]

```
int main()
{
    devHnd device1 ;
    if(LoadInformationFile(“devinfo.ini”)!=SUCCESS) return -1 ;
    if(OpenDEV(&device1,“Device1”)!=SUCCESS ) return -1 ;
    if(SetCommunicateValue(device1,0x0000)!=SUCCESS) return -1 ;

    if(ResumeSyncTask(device1,100)!=SUCCESS ){
        puts(“It failed to resume task.”) ;
    }

    if(CloseDEV(device1)!=SUCCESS) return -1 ;
    return 0;
}
```

（3）SuspendSyncTask()

[Description]

指定タスクに対応する，RX77016 のフレーム・クロック・テーブルのフレーム・カウンタ A に 0 を書き込みます。元のフレーム・カウンタ A 値が 0 以外であった場合，そのタスクは停止タスクとなります。

```
SWORD SuspendSyncTask(
    devHnd device,        /* ターゲット・デバイス・ハンドル */
    WORD SubTaskID        /* サブタスク ID */
) ;
```

[Return Value]

SUCCESS	…	フレーム・カウンタ値 A の更新成功
DEVHND_ERROR	…	認識不可能なデバイス・ハンドルの指定
FUNCID_ERROR	…	認識不可能なサブタスク ID の指定
FAIL	…	DSP との通信に失敗

[Code Example]

```
int main()
{
   devHnd device1 ;
   if(LoadInformationFile(“devinfo.ini”)!=SUCCESS) return -1 ;
   if(OpenDEV(&device1,“Device1”)!=SUCCESS ) return -1 ;
   if(SetCommunicateValue(device1,0x0000)!=SUCCESS) return -1 ;

   if(SuspendSyncTask(device1,100)!=SUCCESS ){
      puts(“It failed to suspend task.”) ;
   }

   if(CloseDEV(device1)!=SUCCESS) return -1 ;
   return 0;
}
```

6.2.3　メモリ・アクセス・ファンクション

（1）CreateCBuf()

[Description]

あらかじめ確保しておいたコミュニケーション・バッファ用のヒープ領域から，nSize 分の使用領域を作成します。また，作成した使用領域に対応するコミュニケーション・バッファ・ハンドルを割り当てます。

```
SWORD CreateCBuf(
    devHnd device,      /* ターゲット・デバイス・ハンドル */
    WORD nSize,         /* コミュニケーション・バッファ・サイズ */
    WORD RWFlag,        /* R/W フラグ */
    cbHnd *cbuf         /* コミュニケーション・バッファ・ハンドル格納先のポインタ */
) ;
```

nSIZE 値が残りのヒープ領域より大きい場合，または予定したバッファ数を越える場合は，新たな領域を作成しません。

RWFlag には，リード用コミュニケーション・バッファの場合“ READ ”を，ライト用コミュニケーション・バッファの場合 “ WRITE ” を指定します。

[Return Value]

SUCCESS	…	作成成功
DEVHND_ERROR	…	認識不可能なデバイス・ハンドルの指定
CBHND_ERROR	…	biosucfg.h で指定したバッファ数を越えた
CBTBL_FULL	…	hapiucfg.h で指定したバッファ数を越えた
FAIL	…	DSP との通信に失敗

[Code Example]

```
int main()
{
   devHnd device1 ;
   cbHnd cbHnd1 ;
   if(LoadInformationFile(“devinfo.ini”)!=SUCCESS) return -1 ;
   if(OpenDEV(&device1,“Device1”)!=SUCCESS ) return -1 ;
   if(SetCommunicateValue(device1,0x0000)!=SUCCESS) return -1 ;

   if(CreateCBuf(device1,10,READ,&cbHnd1)!=SUCCESS ){
      puts(“It failed to create communication buffer.”) ;
   }

   if(CloseDEV(device1)!=SUCCESS) return -1 ;
   return 0;
}
```

（2）FreeCBuf()

[Description]

不要となった，コミュニケーション・バッファの使用領域を開放します。

```
SWORD FreeCBuf(
    devHnd device,          /* ターゲット・デバイス・ハンドル */
    cbHnd cbuf              /* 領域を開放するコミュニケーション・バッファ・ハンドル */
) ;
```

[Return Value]

SUCCESS …… 開放成功
DEVHND_ERROR …… 認識不可能なデバイス・ハンドルの指定
CBHND_ERROR …… 認識不可能なコミュニケーション・バッファ・ハンドルの指定
FAIL …… DSP との通信に失敗

[Code Example]

```
int main()
{
   devHnd device1 ;
   cbHnd cbHnd1 ;
   if(LoadInformationFile("devinfo.ini")!=SUCCESS) return -1 ;
   if(OpenDEV(&device1,"Device1")!=SUCCESS ) return -1 ;
   if(SetCommunicateValue(device1,0x0000)!=SUCCESS) return -1 ;

   if(CreateCBuf(device1,10,READ,&cbHnd1)!=SUCCESS ){
      puts("It failed to create communication buffer.") ;
   }

   if(CloseDEV(device1)!=SUCCESS) return -1 ;
   return 0;
}
```

(3) GetCBufCaps()

[Description]

コミュニケーション・バッファ情報を，CBCaps 構造体にコピーします。RETURN_TO_WINDOW が定義されていれば，Windows 上のメッセージ・ボックスに表示します。

```
SWORD GetCbufCaps(
    devHnd device,              /* ターゲット・デバイス・ハンドル */
    cbHnd cbuf,                 /* ターゲット・コミュニケーション・ハンドル */
    CBCaps *pCommBufCaps        /* CBCaps 構造体のコピー先のポインタ */
);
```

[Return Value]

SUCCESS … コピーまたは表示の成功

DEVHND_ERROR … 認識不可能なデバイス・ハンドルの指定

CBHND_ERROR … 認識不可能なコミュニケーション・バッファ・ハンドルの指定

FAIL … DSP との通信に失敗

[Code Example]

```
int main()
{
   devHnd device1 ;
   CBCaps CBCaps2 ;
   if(LoadInformationFile(“devinfo.ini”)!=SUCCESS) return -1 ;
   if(OpenDEV(&device1,“Device1”)!=SUCCESS ) return -1 ;
   if(SetCommunicateValue(device1,0x0000)!=SUCCESS) return -1 ;
   if(CreateCBuf(device1,10,READ,&cbHnd1)!=SUCCESS ) return -1 ;

   if(GetCBufCaps(device1,cbHnd1,&CBCaps2)!=SUCCESS){
      puts(“It failed to get communication buffer capabilities.”) ;
   }

   if(CloseDEV(device1)!=SUCCESS) return -1 ;
   return 0;
}
```

(4) InitCBuf()

[Description]

ライト・コミュニケーション・バッファの，指定した使用領域の内容を 0 クリアします。

```
SWORD InitCBuf(
    devHnd device,          /* ターゲット・デバイス・ハンドル */
    cbHnd cbuf              /* ターゲット・コミュニケーション・バッファ・ハンドル */
);
```

以前書き込んだデータがまだ読み出されていない場合，本ファンクションは CB_FULL を返し，クリアを行うことはできません。

[Return Value]

SUCCESS	…	0 クリア成功
DEVHND_ERROR	…	認識不可能なデバイス・ハンドルの指定
CBHND_ERROR	…	認識不可能なコミュニケーション・バッファ・ハンドルの指定
CB_FULL	…	以前書き込んだデータが読み出されていない
FAIL	…	DSP との通信に失敗

[Code Example]

```
int main()
{
   devHnd device1 ;
   if(LoadInformationFile("devinfo.ini")!=SUCCESS) return -1 ;
   if(OpenDEV(&device1,"Device1")!=SUCCESS ) return -1 ;
   if(SetCommunicateValue(device1,0x0000)!=SUCCESS) return -1 ;
   if(CreateCBuf(device1,10,WRITE,&cbHnd1)!=SUCCESS ) return -1 ;

   if(InitCBuf(device1,cbHnd1)!=SUCCESS ){
      puts("It failed to initialize communication buffer.") ;
   }

   if(CloseDEV(device1)!=SUCCESS) return -1 ;
   return 0;
}
```

(5) ReadFromTMem()

[Description]

ターゲット・デバイス上のデータ・メモリの SrcAddr 番地のデータを，HOST Processor 上のメモリの lpRData 番地へ書き込みます。

```
SWORD ReadFromTMem(
    devHnd device,          /* ターゲット・デバイス・ハンドル */
    DWORD SrcAddr,          /* ソース・アドレス (YMEM のとき SrcAddr+0x10000) */
    LPWORD lpRData          /* リード・データ書き込み先のポインタ */
);
```

SrcAddr は，X メモリから読み出す場合，アドレスをそのまま指定し，Y メモリから読み出す場合，アドレス＋0x10000 を指定します。

[Return Value]

SUCCESS	…	読み出し成功
DEVHND_ERROR	…	認識不可能なデバイス・ハンドルの指定
FAIL	…	DSP との通信に失敗，またはパラメータが異常

[Code Example]

```
int main()
{
   devHnd device1 ;
   unsigned short RData ;
   if(LoadInformationFile("devinfo.ini")!=SUCCESS) return -1 ;
   if(OpenDEV(&device1,"Device1")!=SUCCESS ) return -1 ;
   if(SetCommunicateValue(device1,0x0000)!=SUCCESS) return -1 ;

   if(ReadFromTMem(device1,0x10000,&RData)!=SUCCESS ){
      puts("It failed to read data .") ;
   }

   if(CloseDEV(device1)!=SUCCESS) return -1 ;
   return 0;
}
```

（6） WriteToTMem()

[Description]

ターゲット・デバイス上のデータ・メモリの DestAddr 番地に WData を書き込みます。

```
SWORD WriteToTMem(
    devHnd device,       /* ターゲット・デバイス・ハンドル */
    DWORD DestAddr,      /* デスティネーション・アドレス (YMEM のとき DestAddr+0x10000) */
    WORD WData           /* ライト・データ */
);
```

DestAddr は，X メモリに書き込む場合は，アドレスをそのまま指定し，Y メモリに書き込む場合は，アドレス＋0x10000 を指定します。

[Return Value]

SUCCESS … 書き込み成功

DEVHND_ERROR … 認識不可能なデバイス・ハンドルの指定

FAIL … DSP との通信に失敗，またはパラメータが異常

[Code Example]

```
int main()
{
   devHnd device1 ;
   if(LoadInformationFile("devinfo.ini")!=SUCCESS) return -1 ;
   if(OpenDEV(&device1,"Device1")!=SUCCESS ) return -1 ;
   if(SetCommunicateValue(device1,0x0000)!=SUCCESS) return -1 ;

   if(WriteToTMem(device1,0x0000,0x369c)!=SUCCESS ){
      puts("It failed to write data .") ;
   }

   if(CloseDEV(device1)!=SUCCESS) return -1 ;
   return 0;
}
```

（7）ReadFromCBuf()

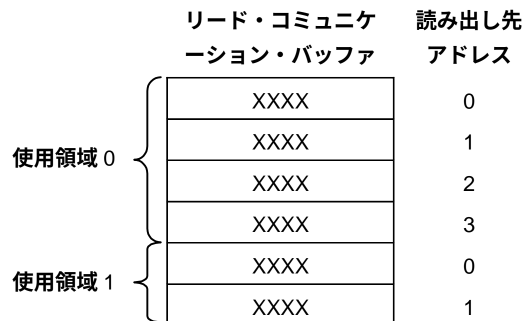
[Description]

リード・コミュニケーション・バッファの，指定した使用領域の任意アドレスのデータを，HOST Processor 上のメモリの lpRData 番地へ書き込みます。

```
SWORD ReadFromCBuf(
    devHnd device,         /* ターゲット・デバイス・ハンドル */
    cbHnd cbuf,            /* ターゲット・コミュニケーション・バッファ・ハンドル */
    WORD RelAddr,          /* 読み出し先アドレス */
    LPWORD lpRData         /* デスティネーションのポインタ */
    WORD Flag              /* アクセス制限フラグ */
) ;
```

読み出し先アドレス（RelAddr）は，指定した使用領域内の何番目のデータを読み出すかを指定します（図 6－1 参照）。

図 6－1　リード・コミュニケーション・バッファの読み出し先アドレス



アクセス制限フラグ（Flag）には，WAIT もしくは NON_WAIT を指定します。

WAIT の場合，CopyTMemToCBuf()または WriteToCBuf:によるデータの書き込みを許可しません。

NON_WAIT の場合，CopyTMemToCBuf()または WriteToCBuf:によるデータの書き込みを許可します。

たとえば図 6－1 の場合，NON_WAIT 指定により，使用領域 0（サイズ 4）でも，1 データを読み出した時点で書き込みを許可することが可能です。サイズ 4 の使用領域のデータをすべて読み出すには，次の 4 ステップが必要です。

```
ReadFromCBuf(device,cbuf,0,RData++,WAIT) ;
ReadFromCBuf(device,cbuf,1,RData++,WAIT) ;
ReadFromCBuf(device,cbuf,2,RData++,WAIT) ;
ReadFromCBuf(device,cbuf,3,RData++,NON_WAIT) ;
```

しかし，ReadFromCBuf()の Flag 検出では，0 以外なら WAIT，0 なら NON_WAIT と判断しています。したがって，Flag 検出法の性質を利用すれば，次のように記述できます。

```
for(i=0 ; i<=3 ; i++) ReadFromCBuf(device,cbuf,i,RData++,3-i) ;
```

現在のデータがすでに読み出し済みの場合，本ファンクションは CB_EMPTY を返し，読み出しを行いません。

[Return Value]

SUCCESS	…	読み出し成功
DEVHND_ERROR	…	認識不可能なデバイス・ハンドルの指定
CBHND_ERROR	…	認識不可能なコミュニケーション・バッファ・ハンドルの指定
CB_EMPTY	…	コミュニケーション・バッファの内容は読み出し済みか，空である
FAIL	…	DSP との通信に失敗，またはパラメータが異常

[Code Example]

```
int main()
{
    devHnd device1 ;
    unsigned short RData[10] ;
    if(LoadInformationFile("devinfo.ini")!=SUCCESS) return -1 ;
    if(OpenDEV(&device1,"Device1")!=SUCCESS ) return -1 ;
    if(SetCommunicateValue(device1,0x0000)!=SUCCESS) return -1 ;
    if(CreateCBuf(device1,10,READ,&cbHnd1)!=SUCCESS ) return -1 ;

    for(int i=9 ; i>=0 ; i--) {
        if(ReadFromCBuf(device1,cbHnd1,i,RData++,i)!=SUCCESS ){
            puts("It failed to read communication buffer's data.") ;
        }
    }

    if(CloseDEV(device1)!=SUCCESS) return -1 ;
    return 0;
}
```

（8）WriteToCBuf()

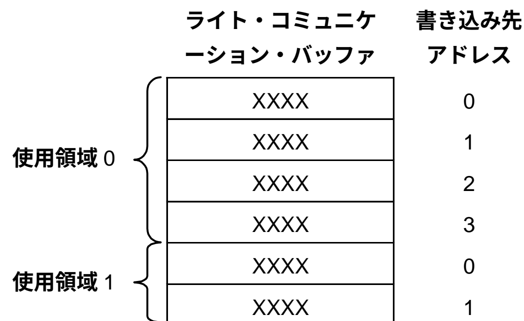
[Description]

ライト・コミュニケーション・バッファの，指定した使用領域の任意のアドレスに WData を書き込みます。

```
SWORD WriteToCBuf(
    devHnd device,          /* ターゲット・デバイス・ハンドル */
    cbHnd cbuf,             /* ターゲット・コミュニケーション・ハンドル */
    WORD RelAddr,           /* 書き込み先アドレス */
    SWORD WData             /* ライト・データ */
    WORD Flag               /* アクセス制限フラグ */
);
```

書き込み先アドレス（RelAddr）は，指定した使用領域内の何番目にデータを書き込むかを指定します（図 6－2 参照）。

図 6－2　ライト・コミュニケーション・バッファの書き込み先アドレス



アクセス制限フラグ（Flag）には，WAIT もしくは NON_WAIT を指定します。

WAIT の場合，CopyCBufToTMem()または ReadFromCBuf:によるデータの読み出しを許可しません。

NON_WAIT の場合，CopyCBufToTMem()または ReadFromCBuf:による読み出しを許可します。

NON_WAIT 指定により，たとえば上記の使用領域 0（サイズ 4）でも，1 データを書き込んだ時点で読み出しを許可することも可能です。サイズ 4 の使用領域のデータをすべて書き込むには，次の 4 ステップが必要です。

```
WriteToCBuf(device,cbuf,0,*WData++,WAIT) ;
WriteToCBuf(device,cbuf,1,*WData++,WAIT) ;
WriteToCBuf(device,cbuf,2,*WData++,WAIT) ;
WriteToCBuf(device,cbuf,3,*WData++,NON_WAIT) ;
```

しかし，WriteToCBuf()では，Flag 検出に 0 以外なら WAIT，0 なら NON_WAIT と判断しています。したがって，Flag 検出法の性質を利用すれば，次のように記述できます。

```
for(i=0 ; i<=3 ; i++) WriteToCBuf(device,cbuf,i,*WData++,3-i) ;
```

以前書き込んだデータがまだ読み出されていない場合，本ファンクションは CB_FULL を返し，書き込みを行いません。

[Return Value]

SUCCESS … 書き込み成功
DEVHND_ERROR … 認識不可能なデバイス・ハンドルの指定
CBHND_ERROR … 認識不可能なコミュニケーション・バッファ・ハンドルの指定
CB_FULL … 以前書き込んだデータが読み出されていない
FAIL … DSP との通信に失敗，またはパラメータが異常

[Code Example]

```
int main()
{
   devHnd device1 ;
   if(LoadInformationFile("devinfo.ini")!=SUCCESS) return -1 ;
   if(OpenDEV(&device1,"Device1")!=SUCCESS ) return -1 ;
   if(SetCommunicateValue(device1,0x0000)!=SUCCESS) return -1 ;
   if(CreateCBuf(device1,10,WRITE,&cbHnd1)!=SUCCESS ) return -1 ;

   for(int i=9 ; i>=0 ; i--){
      if(WriteToCBuf(device1,cbHnd1,i,0x369c,i)!=SUCCESS ){
         puts("It failed to write data to communication buffer.") ;
      }
   }

   if(CloseDEV(device1)!=SUCCESS) return -1 ;
   return 0;
}
```

(9) CopyTMemToHost()

[Description]

ターゲット・デバイス上のデータ・メモリの SrcAddr 番地から SrcAddr＋nSize－1 番地のデータを，HOST Processor 上のメモリの DestPtr 番地から DestPtr＋nSize－1 番地の領域にコピーします。

```
SWORD CopyTMemToHost(
    devHnd device,          /* ターゲット・デバイス・ハンドル */
    DWORD SrcAddr,          /* ソース・アドレス (YMEM のとき SrcAddr+0x10000) */
    LPWORD DestPtr,         /* デスティネーションのポインタ */
    WORD nSize              /* コピー・サイズ */
);
```

SrcAddr は，X メモリからコピーする場合は，アドレスをそのまま指定し，Y メモリからコピーする場合は，アドレス＋0x10000 を指定します。

[Return Value]

SUCCESS	…	コピー成功
DEVHND_ERROR	…	認識不可能なデバイス・ハンドルの指定
FAIL	…	DSP との通信に失敗，またはパラメータが異常

[Code Example]

```
int main()
{
   devHnd device1 ;
   unsigned short RData[10] ;
   if(LoadInformationFile(“devinfo.ini”)!=SUCCESS) return -1 ;
   if(OpenDEV(&device1,“Device1”)!=SUCCESS ) return -1 ;
   if(SetCommunicateValue(device1,0x0000)!=SUCCESS) return -1 ;

   if(CopyTMemToHost(device1,0x10000,RData,0x10)!=SUCCESS ){
      puts(“It failed to copy data.”) ;
   }

   if(CloseDEV(device1)!=SUCCESS) return -1 ;
   return 0;
}
```

（10）CopyHostToTMem()

[Description]

HOST Processor 上のメモリの SrcPtr 番地から SrcPtr＋nSize－1 番地のデータを，ターゲット・デバイス上のデータ・メモリの DestAddr から DestAddr＋nSize－1 番地の領域へコピーします。

```
SWORD CopyHostToTMem(
    devHnd device,       /* ターゲット・デバイス・ハンドル */
    LPWORD SrcPtr,       /* ソース・アドレス */
    DWORD DestAddr,      /* デスティネーション・アドレス (YMEM のとき DestAddr+0x10000) */
    WORD nSize           /* コピー・サイズ */
) ;
```

DestAddr は，X メモリにコピーする場合は，アドレスをそのまま指定し，Y メモリにコピーする場合は，アドレス＋0x10000 を指定します。

[Return Value]

SUCCESS … コピー成功
DEVHND_ERROR … 認識不可能なデバイス・ハンドルの指定
FAIL … DSP との通信に失敗，またはパラメータが異常

[Code Example]

```
int main()
{
   devHnd device1 ;
   unsigned short WData[]={ 0,1,2,3,4,5,6,7,8,9 } ;
   if(LoadInformationFile("devinfo.ini")!=SUCCESS) return -1 ;
   if(OpenDEV(&device1,"Device1")!=SUCCESS ) return -1 ;
   if(SetCommunicateValue(device1,0x0000)!=SUCCESS) return -1 ;

   if(CopyHostToTMem(device1,WData,0x10000,10)!=SUCCESS ){
      puts("It failed to OpenDEVice.") ;
      return -1 ;
   }

   if(CloseDEV(device1)!=SUCCESS) return -1 ;
   return 0;
}
```

（11）CopyCBufToHost()

[Description]

リード・コミュニケーション・バッファの，指定した使用領域のデータを，HOST Processor 上のメモリの DestPtr 番地から DestPtr＋バッファ・サイズ－1 番地の領域へコピーしたあと，データの書き込みを許可します。

```
SWORD CopyCBufToHost(
    devHnd device,          /* ターゲット・デバイス・ハンドル */
    cbHnd cbuf,             /* ターゲット・コミュニケーション・ハンドル */
    LPWORD DestPtr,         /* デスティネーションのポインタ */
);
```

現在のデータがすでに読み出し済みの場合，本ファンクションは CB_EMPTY を返し，読み出しを行いません。

[Return Value]

SUCCESS	…	コピー成功
DEVHND_ERROR	…	認識不可能なデバイス・ハンドルの指定
CBHND_ERROR	…	認識不可能なコミュニケーション・バッファ・ハンドルの指定
CB_EMPTY	…	コミュニケーション・バッファの内容は読み出し済みか，空である
FAIL	…	DSP との通信に失敗

[Code Example]

```
int main()
{
   devHnd device1 ;
   WORD RData[10] ;
   if(LoadInformationFile("devinfo.ini")!=SUCCESS) return -1 ;
   if(OpenDEV(&device1,"Device1")!=SUCCESS ) return -1 ;
   if(SetCommunicateValue(device1,0x0000)!=SUCCESS) return -1 ;
   if(CreateCBuf(device1,10,READ,&cbHnd1)!=SUCCESS ) return -1 ;

   if(CopyCBufToHost(device1,cbHnd1,RData)!=SUCCESS ){
      puts("It failed to copy communication buffer's data to host.") ;
   }

   if(CloseDEV(device1)!=SUCCESS) return -1 ;
   return 0;
}
```

（12）CopyHostToCBuf()

[Description]

HOST Processor 上のメモリの SrcPtr 番地から SrcPtr＋バッファ･サイズ－1 番地のデータを，ライト・コミュニケーション・バッファの指定した使用領域へコピーしたあと，データの読み出しを許可します。

```
SWORD CopyHostToCBuf(
    devHnd device,          /* ターゲット・デバイス・ハンドル */
    cbHnd cbuf,             /* ターゲット・コミュニケーション・ハンドル */
    SWORD *SrcPtr           /* ソース・アドレス */
);
```

以前書き込んだデータがまだ読み出されていない場合，本ファンクションは CB_FULL を返し，書き込みを行いません。

[Return Value]

SUCCESS	…	コピー成功
DEVHND_ERROR	…	認識不可能なデバイス・ハンドルの指定
CBHND_ERROR	…	認識不可能なコミュニケーション・バッファ・ハンドルの指定
CB_FULL	…	以前書き込んだデータが読み出されていない
FAIL	…	DSP との通信に失敗

[Code Example]

```
int main()
{
  devHnd device1 ;
  unsigned short WData[]={ 0,1,2,3,4,5,6,7,8,9 } ;
  if(LoadInformationFile(“devinfo.ini”)!=SUCCESS) return -1 ;
  if(OpenDEV(&device1,“Device1”)!=SUCCESS ) return -1 ;
  if(SetCommunicateValue(device1,0x0000)!=SUCCESS) return -1 ;
  if(CreateCBuf(device1,10,WRITE,&cbHnd1)!=SUCCESS ) return -1 ;

  if(CopyHostToCBuf(device1,cbHnd1,&WData)!=SUCCESS ){
    puts(“It failed to copy data to communication buffer.”) ;
  }

  if(CloseDEV(device1)!=SUCCESS) return -1 ;
  return 0;
}
```

（13）CopyCBufToTMem()

[Description]

リードまたはライト・コミュニケーション・バッファの，指定した使用領域の先頭から nSize 個のデータを，同デバイス上のメモリの DestAddr 番地から DestAddr＋nSize－1 番地へコピーしたあと，データの書き込みを許可します。

```
SWORD CopyCBufToTMem(
    devHnd device,        /* ターゲット・デバイス・ハンドル */
    cbHnd cbuf,           /* ターゲット・コミュニケーション・ハンドル */
    DWORD DestAddr,       /* デスティネーション・アドレス (YMEM のとき DestAddr+0x10000) */
    WORD nSize            /* コピー・サイズ */
);
```

現在のデータがすでに読み出し済みの場合は，本ファンクションは CB_EMPTY を返し，読み出しを行いません。

[Return Value]

SUCCESS … コピー成功
DEVHND_ERROR … 認識不可能なデバイス・ハンドルの指定
CBHND_ERROR … 認識不可能なコミュニケーション・バッファ・ハンドルの指定
CB_EMPTY … コミュニケーション・バッファの内容は読み出し済みか，空である
FAIL … DSP との通信に失敗，またはパラメータが異常

[Code Example]

```
int main()
{
   devHnd device1 ;

   if(LoadInformationFile("devinfo.ini")!=SUCCESS) return -1 ;
   if(OpenDEV(&device1,"Device1")!=SUCCESS ) return -1 ;
   if(SetCommunicateValue(device1,0x0000)!=SUCCESS) return -1 ;
   if(CreateCBuf(device1,10,READ,&cbHnd1)!=SUCCESS ) return -1 ;

   if(CopyCBufToTMem(device1,cbHnd1,0x10000,10)!=SUCCESS ){
      puts("It failed to copy communication buffer's data to memory.") ;
   }

   if(CloseDEV(device1)!=SUCCESS) return -1 ;
   return 0;
}
```

（14）CopyTMemToCBuf()

[Description]

ターゲット・デバイス上のメモリの SrcAddr 番地から SrcAddr＋nSize－1 番地のデータを，リードまたはライト・コミュニケーション・バッファの，指定した使用領域の先頭アドレスからコピーしたあと，データの読み出しを許可します。

```
SWORD CopyTMemToCBuf(
    devHnd device,          /* ターゲット・デバイス・ハンドル */
    cbHnd cbuf,             /* ターゲット・コミュニケーション・ハンドル */
    DWORD SrcAddr,          /* ソース・アドレス (YMEM のとき SrcAddr+0x10000) */
    WORD nSize              /* コピー・サイズ */
) ;
```

以前書き込んだデータがまだ読み出されていない場合，本ファンクションは CB_FULL を返し，書き込みを行いません。

[Return Value]

SUCCESS	…	コピー成功
DEVHND_ERROR	…	認識不可能なデバイス・ハンドルの指定
CBHND_ERROR	…	認識不可能なコミュニケーション・バッファ・ハンドルの指定
CB_FULL	…	以前書き込んだデータが読み出されていない
FAIL	…	DSP との通信に失敗，またはパラメータが異常

[Code Example]

```
int main()
{
   devHnd device1 ;
   unsigned short WData[]={ 0,1,2,3,4,5,6,7,8,9 } ;
   if(LoadInformationFile("devinfo.ini")!=SUCCESS) return -1 ;
   if(OpenDEV(&device1,"Device1")!=SUCCESS ) return -1 ;
   if(SetCommunicateValue(device1,0x0000)!=SUCCESS) return -1 ;

   if(CreateCBuf(device1,10,WRITE,&cbHnd1)!=SUCCESS ) return -1 ;

   if(CopyTMemToCBuf(device1,cbHnd1,&WData,10)!=SUCCESS ){
      puts("It failed to copy data to communication buffer.") ;
   }

   if(CloseDEV(device1)!=SUCCESS) return -1 ;
   return 0;
}
```

（15）CopyTMemToTMem()

[Description]

ターゲット・デバイス上のメモリの SrcAddr 番地から SrcAddr＋nSize－1 番地のデータを，DstAddr 番地から DstAddr＋nSize－1 番地のへコピーします。

```
SWORD CopyTMemToCBuf(
    devHnd device,         /* ターゲット・デバイス・ハンドル */
    DWORD SrcAddr,         /* ソース・アドレス (YMEM のとき SrcAddr+0x10000)*/
    DWORD DstAddr,         /* デスティネーション・アドレス (YMEM のとき SrcAddr+0x10000)*/
    WORD nSize             /* コピー・サイズ */
) ;
```

[Return Value]

SUCCESS	…	コピー成功
DEVHND_ERROR	…	認識不可能なデバイス・ハンドルの指定
FAIL	…	DSP との通信に失敗，またはパラメータが異常

[Code Example]

```
int main()
{
   devHnd device1 ;
   if(LoadInformationFile(“devinfo.ini”)!=SUCCESS) return -1 ;
   if(OpenDEV(&device1,“Device1”)!=SUCCESS ) return -1 ;
   if(SetCommunicateValue(device1,0x0000)!=SUCCESS) return -1 ;

   if(CopyTMemToTMem(device1,0x0000,0x0010,0x10)!=SUCCESS ){
      puts(“It failed to copy data.”) ;
   }

   if(CloseDEV(device1)!=SUCCESS) return -1 ;
   return 0;
}
```

6.2.4　HOST API 管理ファンクション

（1）LoadInformationFile()

[Description]

InFile によって指定された Information File を解析し，あらかじめ確保して置いた InfFile 型の devInf にその解析データを保存します。

```
SWORD LoadInformationFile(
    LPCSTR InFile                /* Information File 名のポインタ */
);
```

API ファンクションは，Information File から得た情報をもとに，ターゲット・デバイスとデータ通信を行うため，使用する前に必ず本ファンクションをコールしなければなりません。

また，InFile で指定するファイル名をフルパスで記述しない場合，Windows の System フォルダ（ドライブ名:¥Windows¥System）に存在すると判断します。

[Return Value]

SUCCESS	…	正常終了
FILE_NOTFOUND	…	指定したファイルが見つからない
SUBTASKTBL_FULL	…	hapiucfg.h の MAX_SUB_TASK 数を越えるサブタスク数
FAIL	…	異常終了

[Code Example]

```
int main()
{
   devHnd device1 ;

   if(LoadInformationFile(“devinfo.ini”)!=SUCCESS){
      puts(“It failed to get Information.”) ;
      return -1 ;
   }

   if(OpenDEV(&device1,“Device1”)!=SUCCESS ) return -1 ;
   if(SetCommunicateValue(device1,0x0000)!=SUCCESS) return -1 ;
   if(CloseDEV(device1)!=SUCCESS) return -1 ;
   return 0;
}
```

（2）ShowInfoContentsAll()

[Description]

LoadInformationFile()で取り込んだ情報を表示します。RETURN_TO_WINDOW が定義されていれば Windows 上のメッセージ・ボックスに，定義されていなければ標準出力に出力します。

```
void ShowInfoContentsAll(
    devHnd device                /* ターゲット・デバイス・ハンドル */
) ;
```

表示項目は，デバイス名，HDT/HST レジスタの I/O Port Address とそのビット・アサイン，リード用/ライト用コミュニケーション・バッファ情報です。

[Return Value]

戻り値はありません。

[Code Example]

```
int main()
{
   devHnd device1 ;
   if(LoadInformationFile(“devinfo.ini”)!=SUCCESS) return -1 ;
   if(OpenDEV(&device1,“Device1”)!=SUCCESS ) return -1 ;
   if(SetCommunicateValue(device1,0x0000)!=SUCCESS) return -1 ;

   ShowInfoContentsAll(device1) ;

   if(CloseDEV(device1)!=SUCCESS) return -1 ;
   return 0;
}
```

（3）HexToBuf()

[Description]

拡張 Intel Hex 形式のファイルを解析し，SrcAddr 番地から SrcAddr＋nSize－1 番地のデータを，HOST Processor 上のメモリの DestPtr 番地にから DestPtr＋nSize－1 番地にコピーします。

```
SWORD HexToBuf(
    WORD HexType        /* Hex ファイル・タイプ */
    LPCSTR InFile,      /* ファイル名のポインタ */
    WORD SrcAddr,       /* ソース・アドレス */
    LPWORD DestPtr,     /* デスティネーションのポインタ */
    WORD nSize          /* コピー・サイズ */
);
```

Hex ファイル・タイプには，インストラクション（たとえば.HXI ファイル）の場合は INST を，データ（たとえば.HDX または.HDY ファイル）の場合は DATA を指定します。

[Return Value]

SUCCESS　…　保存成功

FAIL　…　保存失敗

[Code Example]

```
int main()
{
   devHnd device1 ;
   unsigned short Dest[10] ;
   if(LoadInformationFile("devinfo.ini")!=SUCCESS) return -1 ;
   if(OpenDEV(&device1,"Device1")!=SUCCESS ) return -1 ;
   if(SetCommunicateValue(device1,0x0000)!=SUCCESS) return -1 ;

   if(HexToBuf("Device1.hdx",0x10,Dest,10)!=SUCCESS ){
      puts("It failed to OpenDEVice.") ;
      return -1 ;
   }

   if(CloseDEV(device1)!=SUCCESS) return -1 ;
   return 0;
}
```

第 7 章　BIOS ファンクション

7.1　HI 割り込み発生時のアクション

BIOS は，HI 割り込み，または HO 割り込み発生時のアクションに，特定の BIOS ファンクションを割り当てることで，その機能を実現します。

BIOS の初期状態（通信不可モード）では，HI 割り込み発生時のアクションに EchoWord:を割り当てています。このとき，ユーザ・アプリケーション上で API ファンクションである OpenDEV()をコールすることにより， HI 割り込み発生時のアクションが Interpreter:に変更されます。この状態以降，HI 割り込み発生時のアクションを再度 EchoWord:に変更するまでの間が通信可能モードです。

Interpreter:では，受け取った HDT 値（コマンド）によって，次回の HI 割り込み発生時のアクションを EchoWord:，SelPCmd:に設定するか，もしくは IDTagCmd:を実行します。

SelPCmd:ではあらかじめ決められた数のパラメータを受け取ったあと，指定した BIOS ファンクションを実行します。

IDTagCmd:ではユーザが作成した BIOS ファンクションを実行します。

7.2　BIOS ファンクションの機能

（1）EchoWord:

[Description]

通信経路不確立時の HI 割り込みに対応するアクションです。

*HDT:X が 0x5555（通信経路の確立要求）のとき，次回以降の HI 割り込みに対応するアクションを本ファンクションから Interpreter:に変更します。*HDT:X が 0x5555 以外の場合，1 の補数値を HDT に書き込み，HI 割り込みの対応アクションは変わりません。

[Parameters]

*HDT:X　…　0x5555（通信経路の確立要求）または，それ以外のデータ

[Return Value]

*HDT:X　…　入力値に対する 1 の補数値

（2）Interpreter:

[Description]

通信経路確立時の HI 割り込みに対応するアクションです。

*HDT:X の上位 8 ビットにコマンド，下位 8 ビットにオペランドを入力します。

コマンドは 32 ビットに正規化するためのシフト量によって，次のファンクションへジャンプします。

表 7−1　シフト量とジャンプ先

シフト量	ジャンプ先
0	SelPCmd:
1	IDTagCmd:
2〜7	空き

また，表 7−1 の空きの部分にはユーザが自由に新たなファンクションを定義できます。その場合，mos_hapi.asm の MEDIAOS_HIF_ID_Vect_10 に jmp xxxx（xxxx はファンクションの先頭アドレス）を記述します。

[Parameters]

*HDT:X　…　コマンド，オペランド

[Return Value]

R0L　　…　オペランド

（3）SelPCmd:

[Description]

R0L で示されるジャンプ先ファンクションのアドレスを*_MHA_PCmdPtr:MEM に，そのファンクションのパラメータ数を*_MHA_nRstPCmdPrm:MEM に代入し，次回の HI 割り込み発生時のアクションを Interpreter:から PuttoCmdBuf:に変更します。

[Parameters]

R0L　…　オペランド

[Return Value]

*_MHA_PCmdPtr:MEM　…　ジャンプ先ファンクションのアドレス

*_MHA_nRstPCmdPrm:MEM …　ジャンプ先ファンクションのパラメータ数

（4）PuttoCmdBuf:

[Description]

*HDT 値を_MHA_CmdBuf_Base:MEM を先頭とする領域へスタックしていきます。

*_MHA_nRstPCmdPrm:MEM 回の HI 割り込みがあった時点で，次回の HI 割り込み発生時のアクションを PuttoCmdBuf:から Interpreter:に変更したあと，*_MHA_PCmdPtr:MEM 番地へジャンプします。

表 7－2　*_MHA_PCmdPtr:MEM の内容とジャンプ先

*_MHA_PCmdPtr:MEM の内容	ジャンプ先	パラメータ数
0	Reboot:	4
2	Direct_RX:	1
4	Direct_RY:	1
6	Direct_WX:	2
8	Direct_WY:	2
10	Copy_XtoX:	3
12	Copy_YtoY:	3
14	Copy_XtoY:	3
16	Copy_YtoX:	3

また，表 7－2 の 18 以降の部分には，ユーザが自由に新たなファンクションを定義できます。その場合，mos_hapi.asm の MEDIAOS_HIF_PCmdStruct に DW xxxx（xxxx はファンクションの先頭アドレス）と DW yyyy（yyyy はパラメータ数）を記述します。

[Parameters]

*_MHA_PCmdPtr:MEM　…　ジャンプ先ファンクションのアドレス

*_MHA_nRstPCmdPrm:MEM　…　ジャンプ先ファンクションのパラメータ数

[Return Value]

*_MHA_CmdBuf_Base+?:MEM　…　パラメータ数に応じたスタック・データ

備考　? = 0～パラメータ数

(5) IDTagCmd:

[Description]

R0L で示されるユーザ定義ファンクションへジャンプします。定義方法は，mos_hapi.asm の MEDIAOS_HIF_ID_Vect_20 に jmp xxxx（xxxx はファンクションの先頭アドレス）を記述します。ID_Tag_20XX:＋R0L で示される番地の命令を実行します。

[Parameters]

R0L　…　ジャンプ先ファンクション ID

[Return Value]

戻り値はありません。

(6) Reboot:

[Description]

*_MHA_CmdBuf_Base+2:MEM を DP7 に，*_MHA_CmdBuf_Base＋0:MEM を DP3 に，*_MHA_CmdBuf_Base+1:MEM を DP0 に，*HDT:x を r7l に設定して call DP0 を実行します。

表 7−3　コール・アドレスとリブート・ルーチン

コール・アドレス	リブート・ルーチン
1	Y メモリ → 命令メモリ・ワード・リブート
2	X メモリ → 命令メモリ・ワード・リブート
3	Y メモリ → 命令メモリ・バイト・リブート
4	X メモリ → 命令メモリ・バイト・リブート
5	HDT → 命令メモリ・リブート

[Parameters]

*_MHA_CmdBuf_Base+2:MEM　…　dp7 値
*_MHA_CmdBuf_Base+1:MEM　…　コール・アドレス
*_MHA_CmdBuf_Base+0:MEM　…　dp3 値

[Return Value]

戻り値はありません。

(7) Direct_RX:

[Description]

X メモリの*HDT:X 番地のデータを読み出し*HDT:X に書き込みます。

[Parameters]

*HDT:X … リード・アドレス

[Return Value]

*HDT:X … リード・データ

(8) Direct_RY:

[Description]

Y **メモリの***HDT:X **番地のデータを読み出し***HDT:X **に書き込みます。**

[Parameters]

*HDT:X　… **リード・アドレス**

[Return Value]

*HDT:X … **リード・データ**

(9) Direct_WX:

[Description]

X **メモリの***_MHA_CmdBuf_Base+0:MEM **番地へ***HDT:X **値を書き込みます。**

[Parameters]

*_MHA_CmdBuf_Base+0:MEM … **ライト・アドレス**

*HDT:X … **ライト・データ**

[Return Value]

戻り値はありません。

(10) Direct_WY:

[Description]

Y **メモリの***_MHA_CmdBuf_Base+0:MEM **番地へ***HDT:X **値を書き込みます。**

[Parameters]

*_MHA_CmdBuf_Base+0:MEM … **ライト・アドレス**

*HDT:X … **ライト・データ**

[Return Value]

戻り値はありません。

（11）Copy_XtoX:

[Description]

X メモリの*_MHA_CmdBuf_Base+0:MEM 番地から*HDT:X サイズの領域のデータを，同メモリの*_MHA_CmdBuf_Base+1:MEM 番地から*HDT:X サイズの領域にコピーします。

[Parameters]

*_MHA_CmdBuf_Base+1:MEM　…　ソースの先頭アドレス
*_MHA_CmdBuf_Base+0:MEM　…　デスティネーションの先頭アドレス
*HDT:X　…　コピー・サイズ

[Return Value]

戻り値はありません。

（12）Copy_YtoY:

[Description]

Y メモリの*_MHA_CmdBuf_Base+0:MEM 番地から*HDT:X サイズの領域のデータを，同メモリの*_MHA_CmdBuf_Base+1:MEM 番地から*HDT:X サイズの領域にコピーします。

[Parameters]

*_MHA_CmdBuf_Base+1:MEM+0　…　ソースの先頭アドレス
*_MHA_CmdBuf_Base+0:MEM+1　…　デスティネーションの先頭アドレス
*HDT:X　…　コピー・サイズ

[Return Value]

戻り値はありません。

（13）Copy_XtoY:

[Description]

X メモリの*_MHA_CmdBuf_Base+0:MEM 番地から*HDT:X サイズの領域のデータを，Y メモリの*_MHA_CmdBuf_Base+1:MEM 番地から*HDT:X サイズの領域にコピーします。

[Parameters]

*_MHA_CmdBuf_Base+1:MEM+0　…　ソースの先頭アドレス
*_MHA_CmdBuf_Base+0:MEM+1　…　デスティネーションの先頭アドレス
*HDT:X　…　コピー・サイズ

[Return Value]

戻り値はありません。

（14）Copy_YtoX:

[Description]

Y メモリの*_MHA_CmdBuf_Base+0:MEM 番地から*HDT:X サイズの領域のデータを，X メモリの*_MHA_CmdBuf_Base+1:MEM 番地から*HDT:X サイズの領域にコピーします。

[Parameters]

*_MHA_CmdBuf_Base+1:MEM+0 … ソースの先頭アドレス
*_MHA_CmdBuf_Base+0:MEM+1 … デスティネーションの先頭アドレス
*HDT:X … コピー・サイズ

[Return Value]

戻り値はありません。

（15）ReadFromCBuf:

[Description]

ライト・コミュニケーション・バッファの，R0L で示す使用領域から 1 データを読み出し，R1L に代入します。本ファンクションは DSP 上のサブタスクから直接コールします。

また，複数のタスクで本ファンクションをコールする場合，タイマとして使用している割り込み（os_udef.h の_USED_TimerINT で定義している割り込み）を前もって禁止しておく必要があります。

R0L に指定する値は使用領域を作成した順番で決まります。使用領域を削除したあと，新たな使用領域を作成した場合，その削除した値を割り当てます。

たとえば，API ファンクションの CreateCBuf()で 3 つの使用領域を作成した場合，作成した順に 0，1，2 となります。

CreateCBuf(device,3,WRITE,&cbHndA) ; … この使用領域をリードしたい場合 R0L = 0
CreateCBuf(device,5,WRITE,&cbHndB) ; … この使用領域をリードしたい場合 R0L = 1
CreateCBuf(device,4,WRITE,&cbHndC) ; … この使用領域をリードしたい場合 R0L = 2

その後，2 番目に作成した使用領域(cbHndB)を削除した場合でも，1 番目と 3 番目に作成した使用領域の R0L に指定する値は 0，2 のままです。

FreeCBuf(cbHndB) ; … 1 番目と 3 番目に作成したバッファをリードしたい場合 R0L = 0, 2 のまま

そして，新たに 4 番目の使用領域を作成した場合，R0L に指定する値は 1 となります。

CreateCBuf(device,6,WRITE,&cbHndD) ; … このバッファをリードしたい場合 R0L = 1

さらに，新たに 5 番目の使用領域を作成した場合，R0L に指定する値は 3 となります。

CreateCBuf(device,7,WRITE,&cbHndE) ; … このバッファをリードしたい場合 R0L = 3

以下同様に，使用領域を削除しても，それまでの R0L に指定する値は変わらず，その後新たに使用領域を作成した場合，空いている値を埋めていきます。空いている値がない場合は，新たな値を割り当てます。

R0L 値割り当ては，使用領域作成時の cbHnd の内容と一致しているため，その値を利用してサブタスクに指示することも可能です。

CreateCBuf(device,3,WRITE,&cbHndA) ;　… このバッファをリードしたい場合，R0L = cbHndA となるので，なんらかの手段でサブタスクに指示するプログラムを作成してください。

本ファンクションでは使用領域サイズ分のデータをすべて読み出さなければなりません。本ファンクションは使用領域サイズ分のデータ読み出しを行ったあと，その使用領域へのデータの書き込みを許可します。次の例では，サイズ 5 の使用領域の内容を読み出し，dp0 で示すアドレスに格納しています。

```
R0L=0 ;
Loop 0x5{
   Call ReadFromCBuf ;
   *dp0++=R1L ;
}
```

本ファンクションは，読み出そうとする使用領域のデータがすでに読み出されている，または書き込まれていない場合，戻り値として R2L に 1 をセットし，使用領域のデータの読み出しを行いません。その場合は，使用領域サイズ分の回数，本ファンクションをコールする必要はありません。次の例では，読み出せない場合にループを抜け出しています。

```
CLR(R2L) ;
R0L=0 ;
Loop 0x5{
   Call ReadFromCBuf ;
   if(R2==0) jmp $+2
   LPOP ;
   *dp0++=R1L ;
   NOP ;
}
```

[Parameters]

R0L …　ライト・コミュニケーション・バッファ・ハンドル

[Return Value]

R1L …　リード・データ

R2L …　0: 正常終了，1: 異常終了（データはすでに読み出しているか空であるため，読み出せない）

（16）WriteToCBuf:

[Description]

リード・コミュニケーション・バッファの，R0L で示す使用領域に R1L の内容を書き込みます。本ファンクションは DSP 上のサブタスクから直接コールします。

また，複数のタスクで本ファンクションをコールする場合，タイマとして使用している割り込み（os_udef.h の_USED_TimerINT で定義している割り込み）を前もって禁止しておく必要があります。

R0L に指定する値は，使用領域を作成した順番で決まります。使用領域を削除したあと，新たな使用領域を作成した場合は，その削除した値を割り当てます。

たとえば，API ファンクションの CreateCBuf()で 3 つのリード・コミュニケーション・バッファ使用領域を作成した場合，作成した順に 0，1，2 となります。

CreateCBuf(device,3,READ,&cbHndA)；…　このバッファにライトしたい場合 R0L = 0
CreateCBuf(device,5,READ,&cbHndB)；…　このバッファにライトしたい場合 R0L = 1
CreateCBuf(device,4,READ,&cbHndC)；…　このバッファにライトしたい場合 R0L = 2

その後，2 番目に作成した使用領域(cbHndB)を削除した場合でも，1 番目と 3 番目に作成した使用領域の R0L に指定する値は 0，2 のままです。

FreeCBuf(cbHndB)；…　1 番目と 3 番目に作成したバッファにライトしたい場合 R0L = 0, 2 のまま

そして，新たに 4 番目の使用領域を作成した場合，R0L に指定する値は１となります。

CreateCBuf(device,6,READ,&cbHndD)；…　このバッファにライトしたい場合 R0L = 1

さらに，新たに 5 番目の使用領域を作成した場合，R0L に指定する値は 3 となります。

CreateCBuf(device,7,READ,&cbHndE)；…　このバッファにライトしたい場合 R0L = 3

以下同様に，使用領域を削除しても，それまでの R0L に指定する値は変わらず，その後新たに使用領域を作成した場合，空いている値を埋めていきます。空いている値がない場合は，新たな値を割り当てます。

以上の値割り当ては，使用領域の作成時の cbHnd に 0x7fff で AND マスクした内容と一致しますので，その値を利用してサブタスクに指示することも可能です。

CreateCBuf(device,3,READ,&cbHndA)；　…　このバッファにライトしたい場合，
R0L = cbHndA & 0x7fff となるので，なんらかの手段で
サブタスクに指示するプログラムを作成してください。

本ファンクションは使用領域のサイズ分のデータをすべて書き込まなければなりません。本ファンクションは使用領域のサイズ分の読み出しを行ったあと，その使用領域のデータの書き込みを許可します。次の例では，dp0 で示すアドレスの内容をサイズ 5 の使用領域の内容を書き込んでいます。

```
R0L=0 ;
Loop 0x5{
   R1L=*dp0++ ;
   Call WriteToCBuf ;
}
```

本ファンクションは，書き込もうとする使用領域のデータがまだ読み出されていない場合，戻り値として R2L に 1 をセットし，データの書き込みを行いません。その場合は，使用領域サイズ分の回数，本ファンクションをコールする必要はありません。次の例では，書き込めない場合ループを抜け出しています。

```
CLR(R2L) ;
R0L=0 ;
Loop 0x5{
   R1L=*dp0++ ;
   Call WriteToCBuf ;
   if(R2==0) jmp $+2
   LPOP ;
   NOP ;
   NOP ;
}
```

[Parameters]

R0L …　リード・コミュニケーション・バッファ・ハンドル

R1L …　ライト・データ

[Return Value]

R2L …　0: 正常終了，1: 異常終了（以前書き込んだデータが残っているため，書き込めない）

第 8 章　HOST API 実行ファイルの作成

8.1　API ファンクションを使用した実行ファイルの作成

API ファンクションを使用した実行ファイルを作成するには，API ファンクションをコールしたユーザ・アプリケーションを C コンパイラでコンパイルし，mos_hapi.c，spx.c のオブジェクト・ファイル（先にコンパイルしておく必要があります）とリンクします。

また，次のファイルでは，ターゲット・システムに合わせたユーザの設定またはプログラムの作成が必要です。

8.1.1　hapiucfg.h

hapiucfg.h は，mos_hapi.c，spx.c のユーザ定義用ヘッダ・ファイルです。ターゲット・システムに合わせて次の設定が必要です。

（1）WAIT_TIME，TIM_CONST の定義

#define WAIT_TIME xx　…　xx には数値を記述します。

#define TIM_CONST yy　…　yy には数値を記述します。

WAIT_TIME と TIM_CONST 定義では，spx.c 内で DSP の HDT レジスタをアクセスする関数（inhdtw()，outhdtw()）を用いて，HST レジスタのホスト･リード・イネーブル・フラグまたはホスト・ライト・イネーブル・フラグが 1（リード，ライト許可）に変化するまでウエイトする時間を定義します。ウエイト時間は WAIT_TIME * TIM_CONST で決まります。inhdtw()，outhdtw()で WAIT_TIME，TIM_CONST の文字定数を使用しない場合，これらの定義文を削除できます。

（2）HDTAccess，HSTAccess，RSTAccess の定義

#define HDTAccess x　…　x には 0:バイト・アクセス または 1:ワード・アクセス を記述します。

#define HSTAccess y　…　y には 0:バイト・アクセス または 1:ワード・アクセス を記述します。

#define RSTAccess z　…　z には 0:バイト・アクセス または 1:ワード・アクセス を記述します。

HDTAccess，HSTAccess，RSTAccess 定義では，HDT レジスタ，HST レジスタ，RST 端子をバイト・アクセスするかワード・アクセスするかを定義します。HDT レジスタ，HST レジスタ，RST 端子をアクセスする関数（inhdtw()，outhdtw()，systemreset()）で，HDTAccess，HSTAccess，RSTAccess の文字定数を使用しない場合，これらの定義文は削除できます。

（3）MAX_SUB_TASK 定義

#define MAX_SUB_TASK xx　…　xx には数値を記述します。

MAX_SUB_TASK 定義では，使用するサブタスクの最大数を定義します。ここでの定義値は，実際に使用するサブタスク数より大きくてもかまいませんが，小さいと LoadInformationFile()実行時に SUBTASKTBL_FULL エラーとなります。

（4）NUM_HICBUF，NUM_HOCBUF の定義

#define NUM_HICBUF xx　…　xx には数値を記述します。
#define NUM_HOCBUF yy　…　yy には数値を記述します。

NUM_HICBUF，NUM_HOCBUF 定義では，それぞれ作成可能なライト・コミュニケーション・バッファとリード・コミュニケーション・バッファの使用領域数を定義します。ここでの定義値は，実際に作成するコミュニケーション・バッファ数より大きくてもかまいませんが，小さいと CreateCBuf()実行時に CBTBL_FULL エラーとなります。

8.1.2　spx.c

spx.c は，API ファンクションと DSP 間のインタフェース部分のソース・ファイルです。次に示す関数についてターゲット・システムに合わせたプログラムの作成する必要があります。

WORD inhdtw(Config cmConfig)　…　HDT にデータを書き込む関数です。
WORD outhdtw(Config cmConfig, WORD dataword)　…　HDT からデータを読み出す関数です。
void systemreset(Config cmConfig)　…　システム・リセットを行う関数です。

8.2　BIOS ファンクションを使用した実行ファイルの作成

BIOS ファンクションを使用した実行ファイルを作成するには，RX77016 のターゲット・システム用初期化部（_MOS_TargetSysInit:）と HI 割り込みハンドラ・コード部（_MOS_ivHI:）に必要なコードを挿入し，WB77016 でアセンブルしたあと，RX77016 と bios_fns.asm のオブジェクト・ファイル（先にアセンブルしておく必要があります）とリンクすると，実行ファイルを作成できます。

RX77016 上でコードの挿入または定義の変更が必要な個所は次のとおりです。

8.2.1　os_udef.h

（1）_USED_ivHI の定義

```
#DEFINE _USED_ivHI   TRUE          ; HI    (ITN9)   TRUE:Used   FALSE:Not Used
```

_USED_ivHI 定義は TRUE になっている必要があります。

（2）_IVAL_EIR，_IVAL_SRorMASK，_IVAL_SRandMASK の定義

```
#DEFINE _IVAL_EIR      xx            ; EIR
#DEFINE _IVAL_SRorMASK   xx          ; OR mask value for SR.
#DEFINE _IVAL_SRandMASK   xx         ; AND mask value for SR.
```

_IVAL_EIR，_IVAL_SRorMASK，_IVAL_SRandMASK 定義は，HI 割り込みを許可する定義になっている必要があります。

8.2.2　ターゲット・システム用初期化部（_MOS_TargetSysInit:）

ターゲット・システム用初期化部では，HST レジスタの HDT アクセス・ウエイト許可ビットを立て，%InitHiState(EchoWord)を実行します。また，%InitHiState(EchoWord)以前に biosucfg.h，bios_fns.h，bios_mac.h をインクルードする必要があります。次に，%InitHiState(EchoWord)を実行する記述例を示します。

```
R0L = *HST:x ;
R0 = R0 | 0x0400 ;
*HST:x = R0L ;

%InitHiState(EchoWord) ;
```

8.2.3　HI 割り込みハンドラ・コード部（_MOS_ivHI:）

HI 割り込みハンドラ・コード部（_MOS_ivHI:）では，次に示す 4 ステップを記述します。また，r0l=*_MHA_MdRg_In:MEM 以前に biosucfg.h，bios_fns.h をインクルードする必要があります。

```
clr(r0) ;
r0l = *_MHA_MdRg_In:MEM ;
dp0 = r0l;
jmp dp0;;
```

8.2.4　biosucfg.h

biosucfg.h は，RX77016 のターゲット・システム用初期化部（_MOS_TargetSysInit:），HI 割り込みハンドラ・コード部（_MOS_ivHI:），bios_fns.asm 用のヘッダ・ファイルで，ターゲット・システムに合わせた次の設定が必要です。

（1）_MHA_DATAMEM の定義

#define _MHA_DATAMEM x　…　x には 0:xmem または 1:ymem を記述します。

_MHA_DATAMEM 定義では，コミュニケーション・バッファ情報やヒープ領域，その他 BIOS ファンクションの実行に必要なデータ領域の xmem または ymem への配置を選択できます。

（2）HAVE_EXRAM の定義

#define HAVE_EXRAM x　…　x には 0:内部 RAM または 1:外部 RAM を記述します。

HAVE_EXRAM 定義では，BIOS ファンクションの実行に必要なインストラクション，データのすべてを内部 RAM または外部 RAM への配置を選択できます。

（3）CMD_BUF_SIZE の定義

#define CMD_BUF_SIZE x　…　x には数値を記述します。

CMD_BUF_SIZE 定義では，PuttoCmdBuf:で使用するパラメータのスタック領域のサイズを定義します。BIOS ファンクションの実行に必要なパラメータのスタック数の最大値を記述します。

（4）HICBUF_HEAP_SIZE，NUM_HICBUF の定義

#define HICBUF_HEAP_SIZE x　…　x には数値を記述します。
#define NUM_HICBUF y　…　y には数値を記述します。

HICBUF_HEAP_SIZE，NUM_HICBUF 定義は，ライト・コミュニケーション・バッファのヒープ領域のサイズと，作成可能なバッファ数を定義します。

（5）HOCBUF_HEAP_SIZE，NUM_HOCBUF の定義

#define HOCBUF_HEAP_SIZE x　…　x には数値を記述します。
#define NUM_HOCBUF y　…　y には数値を記述します。

HOCBUF_HEAP_SIZE，NUM_HOCBUF 定義は，リード・コミュニケーション・バッファのヒープ領域のサイズと，作成可能なバッファ数を定義します。

[メ　　モ]

お問い合わせ先

【技術的なお問い合わせ先】

ＮＥＣ半導体テクニカルホットライン（インフォメーションセンター） （電話：午前 9:00～12:00，午後 1:00～5:00）	電　話　：044−548−8899 FAX　：044−548−7900 E-mail　：s-info＠saed.tmg.nec.co.jp

【営業関係お問い合わせ先】

半導体第一販売事業部 半導体第二販売事業部 半導体第三販売事業部	〒108-8001	東京都港区芝５－７－１　（日本電気本社ビル）	(03)3454−1111
中部支社 半導体第一販売部 半導体第二販売部	〒460-8525	愛知県名古屋市中区錦１－17－１　（日本電気中部ビル）	(052)222−2170 (052)222−2190
関西支社 半導体第一販売部 半導体第二販売部 半導体第三販売部	〒540-8551	大阪府大阪市中央区城見１－４－24　（日本電気関西ビル）	(06)6945−3178 (06)6945−3200 (06)6945−3208

北海道支社	札　幌	(011)251−5599
東北支社	仙　台	(022)267−8740
岩手支店	盛　岡	(019)651−4344
郡山支店	郡　山	(024)923−5511
いわき支店	いわき	(0246)21−5511
長岡支店	長　岡	(0258)36−2155
水戸支店	水　戸	(029)226−1717
土浦支店	土　浦	(0298)23−6161
群馬支店	高　崎	(027)326−1255
太田支店	太　田	(0276)46−4011
宇都宮支店	宇都宮	(028)621−2281
小山支店	小　山	(0285)24−5011
甲府支店	甲　府	(055)224−4141
長野支社	松　本	(0263)35−1662
静岡支社	静　岡	(054)254−4794
立川支社	立　川	(042)526−5981,6167
埼玉支社	大　宮	(048)649−1415
千葉支社	千　葉	(043)238−8116
神奈川支社	横　浜	(045)682−4524
三重支店	津	(059)225−7341
北陸支社	金　沢	(076)232−7303
京都支社	京　都	(075)344−7824
神戸支社	神　戸	(078)333−3854
中国支社	広　島	(082)242−5504
鳥取支店	鳥　取	(0857)27−5311
岡山支店	岡　山	(086)225−4455
松山支店	松　山	(089)945−4149
九州支社	福　岡	(092)261−2806

キリトリ

アンケート記入のお願い

お手数ですが，このドキュメントに対するご意見をお寄せください。今後のドキュメント作成の参考にさせていただきます。

［ドキュメント名］　RX77016　アプリケーション・ノート　HOST API編

（U14371JJ1V0AN00（第1版））

［お名前など］（さしつかえのない範囲で）

御社名（学校名，その他）　（　　　　　　　　）
ご住所　（　　　　　　　　）
お電話番号　（　　　　　　　　）
お仕事の内容　（　　　　　　　　）
お名前　（　　　　　　　　）

1．ご評価（各欄に○をご記入ください）

項　　目	大変良い	良　い	普　通	悪　い	大変悪い
全体の構成					
説明内容					
用語解説					
調べやすさ					
デザイン，字の大きさなど					
その他（　　　　　　）					
（　　　　　　）					

2．わかりやすい所（第　章，第　章，第　章，第　章，その他　　　　）

理由［　　　　］

3．わかりにくい所（第　章，第　章，第　章，第　章，その他　　　　）

理由［　　　　］

4．ご意見，ご要望

5．このドキュメントをお届けしたのは

NEC販売員，特約店販売員，NEC半導体ソリューション技術本部員，

その他（　　　　　　）

ご協力ありがとうございました。

下記あてにFAXで送信いただくか，最寄りの販売員にコピーをお渡しください。

NEC半導体テクニカルホットライン

FAX：（044）548-7900