DOS 用 NI-488.2™
ユーザマニュアル

1993 年 11 月版
製品番号 370885A-01

日本ナショナルインスツルメンツ (株)
〒 142 東京都品川区戸越 5-14-24
ITO ビル 2 階
TEL:　(03)3788-1921
FAX:　(03)3788-1923

National Instruments Corporate Headquarters
6504 Bridge Point Parkway
Austin, TX 78730-5039
(512) 794-0100
Technical support fax: (800) 328-2203
(512) 794-5678

Branch Offices:
Australia (03) 879 9422, Austria (0662) 435986, Belgium 02/757.00.20,
Canada (Ontario) (519) 622-9310, Canada (Quebec) (514) 694-8521,
Denmark 45 76 26 00, Finland (90) 527 2321, France (1) 48 14 24 24,
Germany 089/741 31 30, Italy 02/48301892, Japan (03) 3788-1921,
Mexico 95 800 010 0793, Netherlands 03480-33466, Norway 32-84 84 00,
Singapore 2265886, Spain (91) 640 0085, Sweden 08-730 49 70,
Switzerland 056/20 51 51, Taiwan 02 377 1200, U.K. 0635 523545

限定保証

ナショナルインスツルメンツのソフトウェアが入った媒体は、出荷日から90日間は、媒体材料やソフトウェア制作上の欠陥が原因でプログラミング用の命令が実行不可能にならないことが保証されています。出荷日はソフトウェアの受領書または他の文書(登録カード)によって証明されます。ソフトウェア媒体がプログラミング用の命令を実行しない欠陥がある旨の通知が、上記保証期間中にナショナルインスツルメンツにあった場合、ナショナルインスツルメンツは、当社の判断によりそのソフトウェア媒体を修理または交換します。ナショナルインスツルメンツは、ソフトウェアの動作が中断しないことや、エラーが起きないことは保証しません。

ナショナルインスツルメンツは本マニュアルの内容が正確であると考えています。本書は技術的に正しいかどうか入念に見直されています。技術上または印刷上の間違いがあった場合、ナショナルインスツルメンツは本書の所持者に事前に通告することなく次回以降の版に変更を加える権利を有します。本マニュアルに間違いと思われる個所を発見された場合には、ナショナルインスツルメンツにお問合せください。いかなる場合もナショナルインスツルメンツは、本書およびその内容に関連した原因から生じた損害に対して責任を負いません。

ナショナルインスツルメンツは、保証の明示、暗示を問わず、ここに記載された以外の保証は行わず、特に、特定用途に対する市場性や適性に対する保証は行いません。ナショナルインスツルメンツの過失または不注意により発生した損害に対するユーザの賠償権は、ユーザがそれまでに支払った金額を限度とします。データ紛失から生じた損失や、利益、製品の使用、付随的または結果的に生じた損害に対しては、ナショナルインスツルメンツは、たとえそのようなことが生じる可能性があると告知されていた場合でも、責任を負いません。このナショナルインスツルメンツの限定責任は、契約が遵守された場合でも、契約に違反した場合でも、不注意の場合でも、訴訟方式に関係なく適用されます。ナショナルインスツルメンツに対する訴訟は、訴訟の原因が生じてから1年以内に起こす必要があります。ナショナルインスツルメンツは、妥当な管理限界を超えた原因により発生した履行遅延に対する責任は負いません。ここに定めた限定保証では、ユーザがナショナルインスツルメンツの設置・操作・保守に関連する指示を守らなかったために生じた損害、欠陥、誤作動、動作故障は対象となりません。さらに、ユーザが製品を改造した場合、ユーザによる酷使・誤操作・不注意の場合、停電・電源サージ・火事・洪水・事故・第三者の行為・その他、妥当な管理の範囲外の事象も、本限定保証の対象とはなりません。

著作権

登録商標

NI-488®, NI-488.2™, TNT4882C™ はナショナルインスツルメンツの登録商標です。

リストされた製品名および会社名は、それぞれ該当する会社の商標または商標名です。

ナショナルインスツルメンツの製品を医療用、臨床用として使用する場合の警告

ナショナルインスツルメンツの製品は、人体の治療や診断に使うことを目的としていません。ナショナルインスツルメンツの製品を医療用または臨床用の目的で使用した場合、製品の故障、またはユーザやアプリケーション設計者のミスにより、怪我を招く恐れがあります。ナショナルインスツルメンツの製品を医療用または臨床用として使用する場合は、適切な訓練を受け資格を有する医療専門家が行うものとし、また当該製品を使用する場合は、重大な怪我や死亡の危険を避けるため、従来の医療安全策、機器、および手順を引き続き実施してください。ナショナルインスツルメンツの製品は、医療または臨床治療における人体の健康と安全を監視もしくは保護するための既定のプロセス、手順、または機器の代わりに使用するものではありません。

目次

第 3 章
アプリケーションの開発

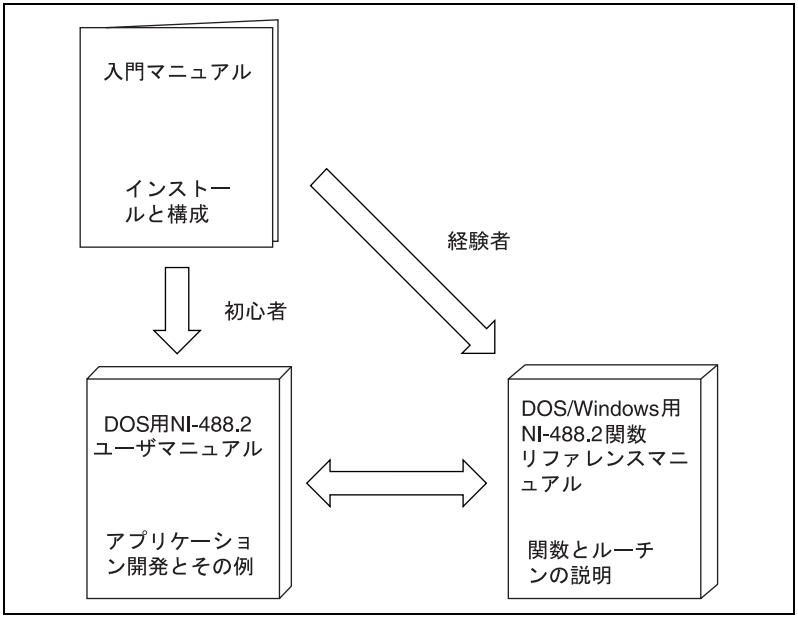
第 8 章
ibconf - インタフェースバス構成ユーティリティ

表

このマニュアルについて

このマニュアルでは、DOS 用 NI-488.2 ソフトウェアの特徴と機能について説明します。NI-488.2 ソフトウェアは、Microsoft MS-DOS(バージョン 3.0 以上) またはそれと同等のシステムで使用されることを前提としています。また、このマニュアルでは DOS の知識が必要になります。

このマニュアルセットの使い方



GPIB ハードウェアと DOS 用 NI-488.2 ソフトウェアのインストールと構成については、入門マニュアルをお読みください。

GPIB の基礎とアプリケーションプログラムの開発方法については、「DOS 用 NI-488.2 ユーザマニュアル (部品番号 320749-01)」をお読みください。

各 NI-488 関数および NI-488.2 ルーチンのフォーマット、パラメータ、エラーに関する詳細な説明については、「DOS/Windows 用 NI-488.2 関数リファレンスマニュアル (部品番号 320702-01)」をお読みください。

このマニュアルの構成

このマニュアルは次のように構成されています。

- 第 1 章「はじめに」では、GPIB と NI-488.2 ソフトウェアの概要について説明します。
- 第 2 章「アプリケーション例」では、9 つのアプリケーション例を取り上げ、ユーザ独自のアプリケーションを書く際に役に立つ、GPIB の特定の概念やテクニックについて説明します。各例の説明には、プログラマの役割、プログラムフローチャート、およびフローチャートのブロック番号に対応した番号順のステップが含まれます。
- 第 3 章「アプリケーションの開発」では、NI-488 関数と NI-488.2 ルーチンを使用して GPIB アプリケーションプログラムを開発する方法について説明します。
- 第 4 章「アプリケーションのデバッグ」では、アプリケーションプログラムのデバッグ方法をいくつか紹介します。
- 第 5 章「ibic - インタフェースバス対話式制御ユーティリティ」では、GPIB デバイスと対話式の通信を行う際に使用できる対話式制御プログラム ibic について説明します。
- 第 6 章「appmon - GPIB アプリケーションモニタ」では、役に立つデバッグ用ツール、GPIB アプリケーションモニタ `appmon` のインストール、構成および使用方法について説明します。
- 第 7 章「GPIB プログラミングテクニック」では、アプリケーションプログラムで NI-488 関数や NI-488.2 ルーチンを使用する際のテクニックについて説明します。
- 第 8 章「ibconf - インタフェースバス構成ユーティリティ」では、NI-488.2 ソフトウェアを構成する際に使用するソフトウェア構成プログラム `ibconf` について説明します。
- 付録 A「ステータスワード状況」では、ステータスワード `ibsta` に返される状態について詳しく説明します。
- 付録 B「エラーコードと対処」では、各エラーとその原因および対処をリストし、説明します。
- 付録 C「ユニバーサル言語インタフェース」では、ユニバーサル言語インタフェース (ULI) のインストールおよび使用方法について説明します。
- 付録 D「カスタマーコミュニケーション」には、ご不明な点をナショナルインスツルメンツに質問したり、当社の製品とマニュアルについてコメントをお寄せいただくための用紙があります。
- 「用語集」では、このマニュアルで使用している用語を、英語や略語などはアルファベット順に、日本語は 50 音順にリストし、それぞれの用語について説明しています。

- 「索引」では、このマニュアルで使用している主な用語とトピックを、英語や記号などはアルファベット順に、日本語は 50 音順にリストし、該当ページを示してあります。

このマニュアルで使う表記法

このマニュアルでは次の表記法に従います。

太字 (**bold**)	太字のテキストは、Windows、メニュー、メニューやダイアログボックスのオプションを表します。
イタリック体 (*italic*)	イタリック体のテキストは、強調、相互参照、フィールド名、または重要な概念の紹介を表します。
太字のイタリック体 (***bold italic***)	太字のイタリック体は、メモ、注意、または警告を表します。
(monospace)	モノスペースのテキストは、キーボードから入力するテキストまたは文字を表します。コードの一部、プログラム例、構文例もこのフォントで表します。また、ディスクデバイス、パス、ディレクトリ、デバイス名、変数の正しい名前およびプログラムコードから取り出される文にもこのフォントを使用します。
(**bold monospace**)	モノスペースの太字の小文字テキストは、コンピュータからスクリーンに自動的に表示されるメッセージおよび応答を表します。
(*italic monospace*)	モノスペースのイタリック体の小文字テキストは、使用されている用語の代わりに適切なワードや値を入力することを示します。
<>	キーボード上のキーの名前は角括弧で囲みます。例 :<PageDown>
<Enter>	キーの名前は頭が大文字になっています。
-	2 つ以上のキーの名前をハイフンでつなぎ、角括弧で囲んでいる場合は、最初のキーを押しながら次のキーを押すことを示します。 例 : <Control-C>
IEEE 488, IEEE 488.2	このマニュアルでは、IEEE 488 と IEEE 488.2 は、それぞれ GPIB を定義した ANSI/IEEE 規格 488.1-1987 と ANSI/IEEE 規格 488.2-1987 を意味します。
NI-488.2 ソフトウェア	このマニュアルでは、NI-488.2 ソフトウェアという用語は、他に指定のない場合は Windows 用の NI-488.2 ソフトウェアを意味します。 略語、頭辞語、メートル法の接頭辞、ニモニック、シンボル、および用語は用語集にリストしてあります。

関連文書

このマニュアルを読むにあたって有益な関連情報が、次の文書に記載されています。

- ANSI/IEEE Standard 488.1-1987, IEEE Standard Digital Interface for Programmable Instrumentation(ANSI/IEEE 規格 488.1-1987。IEEE 規格によるプログラム可能計測器デジタルインタフェース)
- ANSI/IEEE Standard 488.1-1987, IEEE Standard Codes, Formats, Protocols, and Common Commands(ANSI/IEEE 規格 488.1-1987。IEEE 規格によるコード、フォーマット、プロトコル、共通コマンド)
- Microsoft Windows User's Guide(Microsoft Windows ユーザガイド)

カスタマーコミュニケーション

ナショナルインスツルメンツでは、当社の製品とマニュアルについて、お客様のご意見をお聞きしたいと考えています。お客様が当社の製品を使い、どのようなアプリケーションを開発しているのか教えていただければ、何か問題があったときでもお役に立てます。当社に簡単に連絡できるよう、このマニュアルにはコメント用紙とシステム構成用紙が添付されています。用紙は巻末の付録 D「カスタマーコミュニケーション」にあります。

1

はじめに

本章では、GPIB と NI-488.2 ソフトウェアの概要について説明します。

GPIB の概要

ANSI/IEEE 規格 488.1-1987 は、通常、GPIB(汎用インタフェースバス) と呼ばれますが、この規格は、市販の多数のメーカの機器やコントローラの通信に使う標準インタフェースを規定したものです。GPIB には電気的、機械的、機能的な仕様がすべて含まれます。GPIB とはデジタル式の 8 ビット・パラレル通信用インタフェースで、1Mbytes/s 以上のデータ転送速度があります。GPIB がサポートするのは、1 台のシステムコントローラ (通常はコンピュータ) と 14 台までの計測器です。IEEE 規格 488.1 を拡張した ANSI/IEEE 規格 488.2-1987 では、通信プロトコル、一般的なデータコード / データフォーマット、汎用のデバイスコマンドなどが定義されました。

トーカ、リスナ、コントローラ

GPIB デバイスはトーカ、リスナ、コントローラのいずれかとして動作します。トーカはデータメッセージを送信します。リスナはデータメッセージを受信します。コントローラ (通常はコンピュータ) はバス上の情報の流れを管理します。コントローラは通信リンクを定義し、GPIB コマンドをデバイスに送信します。

また、いくつもの役割を果たせるデバイスもあります。たとえばデジタル電圧計はトーカにもリスナにもなれます。お使いのパソコンにナショナルインスツルメンツ製 GPIB インタフェースボードと NI-488.2 ソフトウェアをインストールしてあれば、そのパソコンはトーカ、リスナ、コントローラのいずれにもなることができます。

コントローラインチャージとシステムコントローラ

GPIB 上にコントローラが複数あっても構いませんが、一度にでアクティブコントローラになれるコントローラは 1 つだけです。このコントローラをコントローラインチャージ (CIC) と呼びます。コントローラがアクティブでないときはアイドル (活動休止中の) コントローラといいます。アクティブ制御権は、現在の CIC からアイドルコントローラに渡すこと

ができます。システムコントローラ (通常は GPIB インタフェースボード) は、バス上で自分自身を CIC にすることができる唯一のデバイスです。

GPIB アドレス指定

GPIB に接続したデバイスやボードには、必ずそれぞれ固有の GPIB アドレスを指定しなければなりません。コントローラがデータの送受信を行うときには、アドレスで各デバイスを識別します。GPIB アドレスは、1 次アドレスと、2 次アドレス (任意) の 2 つの部分からなっています。アドレスを設定するには、普通はボードやデバイス上のスイッチを使います。

1 次アドレスとは 0 ～ 30 の範囲の数です。GPIB コントローラがデバイスと通信するときには、GPIB で送信するトークアドレスまたはリスンアドレスを 1 次アドレスを使って作成します。

トークアドレスを作成するには、GPIB アドレスのビット 6、つまり TA(Talk Active) ビットを設定します。リスナアドレスを作成するには、GPIB アドレスのビット 5、つまり LA(Listen Active) ビットを設定します。デバイスがアドレス 1 にある場合を例に考えてみると、コントローラは 16 進 "41"(ビット 6 を設定したアドレス 1) を送信して、そのデバイスをトーカにします。コントローラの 1 次アドレスは通常は 0 なので、コントローラは 16 進 "20"(ビット 5 を設定したアドレス 0) を送信して自分自身をリスナにします。GPIB アドレスビットの構成を図 2-1 に示します。

ビット位置	7	6	5	4	3	2	1	0
意味	0	TA	LA	GPIB の 1 次アドレス (0 ～ 30 の範囲)				

図 1-1　GPIB アドレスビット

デバイスによっては 2 次アドレスも使えます。2 次アドレスは 16 進 "60" ～ 16 進 "7E" の範囲の数です。2 次アドレスを使う場合、コントローラはデバイスの 1 次トーク (またはリスン) アドレスの後にその 2 次アドレスを続けて送信します。

GPIB 上でのメッセージ送信

GPIB 上のデバイス同士は、メッセージを送信しあって通信を行います。信号や線は GPIB インタフェースを介してメッセージを転送します。GPIB インタフェースは 16 本の信号線と 8 本の接地帰還 (シールドドレーン) 線で構成されています。16 本の信号線については、本書に説明があります。

データ線

DIO0 ～ DIO8 までの 8 本のデータ線を使って、データとコマンドメッセージの両方を送信します。

ハンドシェイク線

ハードウェアとのハンドシェイク線が 3 本あり、デバイス間のメッセージバイトの転送を非同期的に制御します。このプロセスは 3 線インタロックハンドシェイクといいます。このプロセスによって、デバイスはデータ線上のメッセージバイトを転送エラーなく送受信できます。GPIB ハンドシェイク線についてのまとめを表 2-1 に示します。

表 1-1　GPIB ハンドシェイク線

線	説明
NRFD (not ready for data)	リスナとなったデバイスがメッセージバイトを受信する準備ができているかどうかを示します。トーカが高速転送 (HS488) であることを知らせるのもこの線です。
NDAC (not data accepted)	リスナとなったデバイスがメッセージバイトを受信したかどうかを示します。
DAV (data valid)	トーカとなったデバイスが、データ線上の信号が安定した (有効な) データかどうかを示します。

インタフェース管理線

5 本の GPIB ハードウェア線が GPIB バス上の情報の流れを管理します。GPIB インタフェース管理線についてのまとめを表 1-2 に示します。

表 1-2　GPIB インタフェース管理線

線	説明
ATN (attention)	コントローラはコマンドの送信時には ATN 線を TRUE(真) に、データメッセージの送信時には FALSE(偽) に設定します。
IFC (interface clear)	システムコントローラは IFC 線を設定することで GPIB バスを初期化し、自分自身を CIC に設定します。
REN (remote enable)	システムコントローラは REN 線を設定することでデバイスをリモートプログラムモードまたはローカルプログラムモードにします。

表 1-2　GPIB インタフェース管理線（続き）

SRQ (service request)	どのデバイスも、SRQ 線を設定してコントローラからのサービスを非同期に要求することができます。
EOI (end or identify)	トーカは EOI 線を使ってデータメッセージの最後にマークを付けます。コントローラはパラレルポールを行うときに EOI 線を使います。

システムのセットアップと構成

通常、デバイスはケーブルアセンブリで接続します。ケーブルアセンブリは、シールド 24 芯ケーブルの両端にプラグとレセプタクルコネクタを取り付けたものです。こうすることでデバイス同士のリンクを、リニア (直線) 構成、スター (星状) 構成、またはその両者を組み合わせた構成に接続することができます。リニア構成とスター構成を図 2-2 に示します。

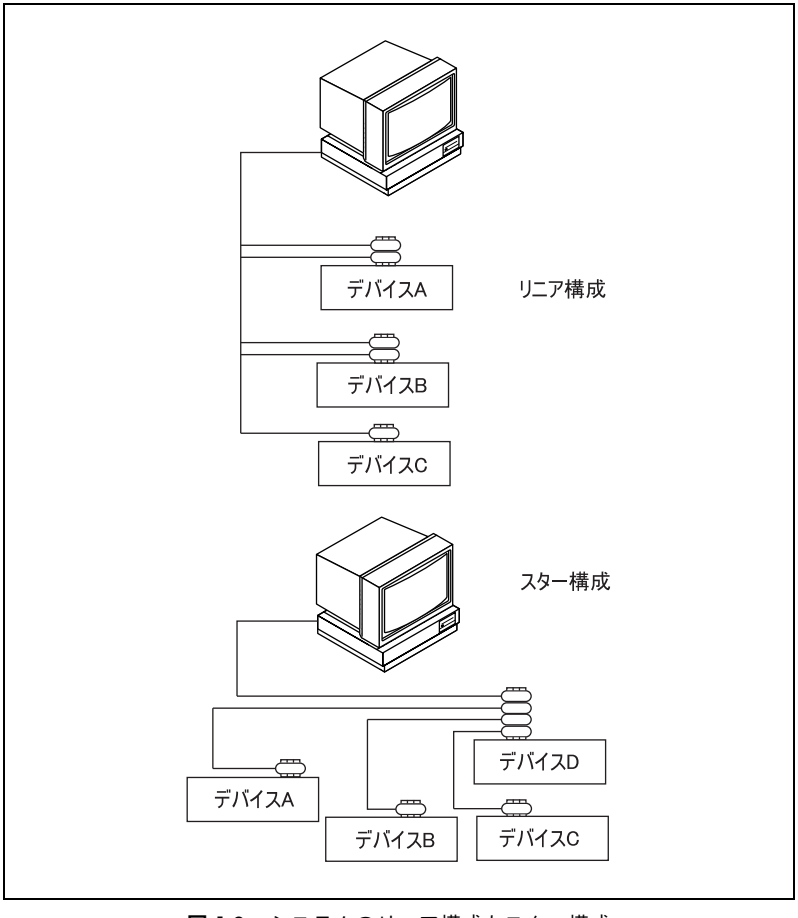


図 1-2　システムのリニア構成とスター構成

複数ボードの制御

DOS 用 NI-488.2 ドライバなどのマルチボードドライバは、複数のインタフェースボードを制御することができます。図 2-3 にマルチボードシステム構成の例を示します。`gpib0` は電圧計のアクセスボード、`gpib1` はプロッタとプリンタのアクセスボードです。デバイスの制御関数は、自動的にそれぞれのボードにアクセスします。

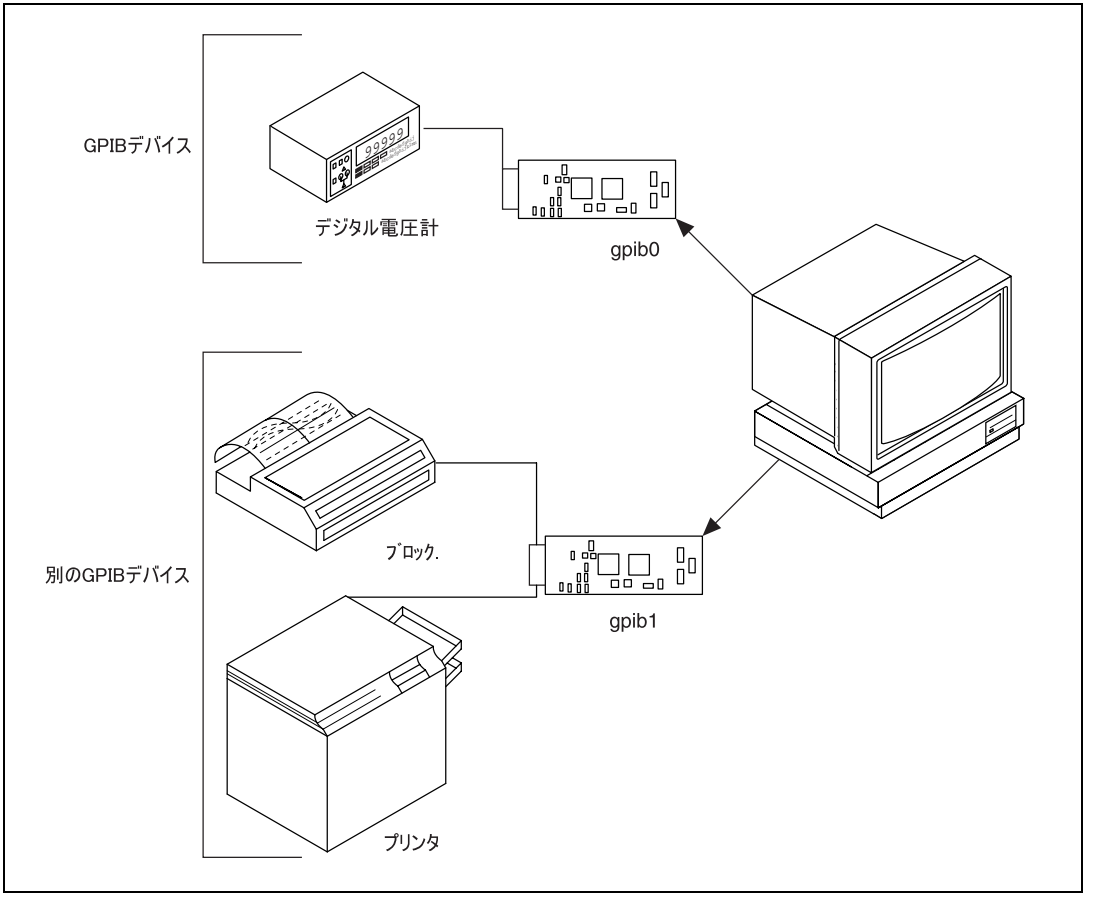


図 1-3　マルチボードシステムのセットアップ例

必要なシステム構成

GPIB の目的である高速データ転送を達成するためには、デバイス間の物理的距離を短くし、バス上のデバイス数を制限しなければなりません。一般的に、以下のような制約があります。

- デバイス間の距離は、どこをとっても 4 メートルを超えてはならず、バス全体でのデバイス同士の平均距離は 2 メートルを超えないこと。
- ケーブルの合計長は最大で 20 メートルとする。
- 1 つのバスに最大で 15 のデバイスを接続できるが、最低でもその 3 分の 2 の電源がオンであること。

処理を高速にするため、以下のような制約があります。

- システムのすべてのデバイスの電源がオンであること。

- ケーブル長は可能な限り短くし、最大でも 1 システムで合計 15 メートルとする。
- ケーブル 1 メートルごとに少なくとも 1 つのデバイスと同等の負荷があること。

この制約外の条件で GPIB を使用する場合は、バスエクステンダでケーブル長を延長したり、エクスパンダでデバイス負荷の数を増やすことができます。エクステンダやエクスパンダが必要な場合は、ナショナルインスツルメンツにご連絡ください。

次に NI-488.2 ソフトウェアについて説明します。NI-488.2 ソフトウェアは GPIB 上の通信の流れを制御します。

NI-488.2 ソフトウェアパッケージ

以下に、DOS 用 NI-488.2 ソフトウェアの重要な要素に焦点を当て、そのそれぞれの機能を説明します。

NI-488.2 ドライバとドライバユーティリティ

NI-488.2 ソフトウェアには以下のドライバファイル、ユーティリティファイルがあります。

- `readme.txt` : NI-488.2 ソフトウェアに関する重要情報と、新規機能についての説明が入った文書ファイルです。NI-488.2 ソフトウェアを使う前にこのファイルを読んで、最新の情報を確認してください。
- `install.exe` : メニューにしたがって NI-488.2 ソフトウェアをインストールするプログラムです。
- `gpib.com` : NI-488.2 ドライバです。システム起動時に DOS がロードします。
- `ibdiag.exe` : GPIB ハードウェアをテストし、正しく動作するかどうか確認するのに使うプログラムです。
- `ibtest.exe` : NI-488.2 ソフトウェアが正しくインストールされているかどうかテストします。
- `gpibinfo.exe` : NI-488.2 ソフトウェアのバージョン、使用インタフェースボードのタイプなど、GPIB ハードウェア / ソフトウェアについて知るためのユーティリティです。
- `ibic.exe` : 対話式制御プログラムで、NI-488.2 ソフトウェアの関数やルーチンを使って GPIB デバイスとの通信を対話式に行います。NI-488.2 ソフトウェアのルーチンの勉強にもなり、計測器や GPIB デバイスのプログラム作成にも参考になります。
- `ibconf.exe` : ソフトウェア構成プログラムで、NI-488.2 ソフトウェアの構成パラメータを変更するのに使います。

- appmon.exe : GPIB アプリケーションモニタプログラム。デバッグツールであり、お使いの DOS アプリケーションからの NI-488.2 ソフトウェアの呼び出しをモニタできます。
- ibtrap.exe : 上記のアプリケーションモニタプログラムの構成を変更するするプログラムです。

C 言語ファイル

NI-488.2 ソフトウェアには以下の C 言語ファイルが入っています。

- readme.mc : C 言語インタフェースについての情報が入った文書ファイルです。
- mcib.lib : Microsoft C(バージョン 5.1 以上) 言語インタフェースライブラリです。このライブラリをアプリケーションプログラムとリンクさせなければ、プログラムは NI-488.2 ドライバにアクセスできるようになりません。
- decl.h : インクルードファイルです。NI-488.2 関数と NI-488.2 ルーチンのプロトタイプ、初期設定定数、などが入っています。

BASIC 言語ファイル

NI-488.2 ソフトウェアには、Microsoft Professional BASIC(バージョン 4.0 以上)、Microsoft Visual BASIC for DOS(バージョン 1.0 以上)、QuickBASIC(バージョン 4.0 以上)、BASICA、GWBASIC の言語インタフェースファイルが入っています。以下にそのファイルについて説明します。

- readme.mb : Microsoft BASIC と Microsoft Visual BASIC の言語インタフェースに関する情報の入った文書ファイルです。
- mbib.obj : バイナリ言語インタフェースファイルです。Microsoft BASIC(バージョン 7.0 以上) や Microsoft Visual BASIC(バージョン 1.0 以上) で書かれたアプリケーションプログラムが NI-488.2 ドライバにアクセスできるようにします。
- mbdecl.bas : Microsoft BASIC や Microsoft Visual BASIC のアプリケーションプログラムの最初に置くコードが入った宣言ファイルです。
- readme.qb : QuickBASIC 言語インタフェースに関する情報が入った文書ファイルです。
- qbib.obj : バイナリ言語インタフェースファイルです。QuickBASIC(バージョン 4.0 以上) で書かれたアプリケーションプログラムが NI-488.2 ドライバにアクセスできるようにします。
- qbdecl.bas : QuickBASIC のアプリケーションプログラムの最初に置くコードが入った宣言ファイルです。

- `readme.ba` : BASICA/GWBASIC 言語インタフェースに関する情報が入った文書ファイルです。
- `bib.m` : バイナリ言語インタフェースファイルです。BASICA/GWBASIC のプログラムが NI-488.2 ドライバにアクセスできるようにします。
- `decl.bas` : BASICA/GWBASIC のアプリケーションプログラムの最初に置くコードが入った宣言ファイルです。

ユニバーサル言語インタフェースファイル

NI-488.2 ソフトウェアにはユニバーサル言語インタフェース (ULI) ファイル、`uli.com` が入っています。このファイルを使うのは、HP スタイルの呼び出しを使った既存のアプリケーションをお使いの場合だけにしてください。

サンプルアプリケーションファイル

NI-488.2 ソフトウェアには、C, Microsoft BASIC, QuickBASIC, BASICA 用ソースコードだけでなく、サンプルアプリケーションも 9 つ入っています。サンプルアプリケーションファイルの詳細については第 2 章の「アプリケーション例」の項を参照してください。

NI-488.2 ソフトウェアと DOS との関係

NI-488.2 ドライバは、システム起動時にロードされ、標準 DOS デバイスドライバとして動作します。

NI-488.2 ソフトウェアが DOS および GPIB ハードウェアに対してどのような動作をするかを図 2-4 に示します。

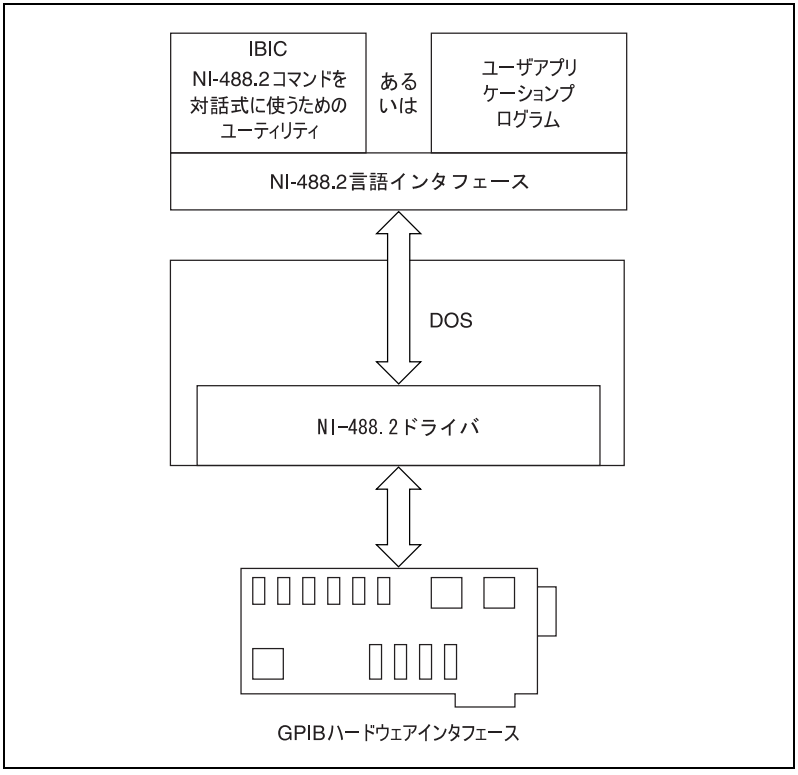


図 1-4　NI-488.2 ソフトウェアと DOS との関係

NI-488.2 ドライバのアンロードと再ロード

DOS が起動されると、`config.sys` ファイル内のコマンドによって NI-488.2 ドライバがロードされます。NI-488.2 ドライバをアンロードしたいときは、先頭に "`rem`" を加え、このコマンドを注釈行に変更します。このコマンドを注釈に変更すると、システムを再起動したとき NI-488.2 ドライバはロードされません。NI-488.2 ドライバをロードさせないためには、`config.sys` ファイルを以下のように変更してください。

1. `config.sys` ファイルを起動ドライブ (通常は `C` ドライブ) のルートディレクトリに置きます。
2. コマンド行を次のような形で入力します。

   `device = drive:\path\gpib.com`

   ここで、*`drive`* はドライブ名 (通常は C ドライブ)、*`path`* は NI-488.2 ソフトウェアをインストールしたディレクトリへのパス (例 `c:\at-gpib`) です。

3. 次のようにコマンド行の頭に "rem" をつけ加え、注釈行に変更します。

   ```
   rem device = drive:\path\gpib.com
   ```

4. コンピュータを再起動すれば、今行った変更が有効になります。

再び NI-488.2 ドライバを使用する必要があれば、今の行から "rem" を削除し、システムを再起動してください。

2

アプリケーション例

本章では 9 つのアプリケーション例を使って GPIB の概念と手法を説明し、ユーザがアプリケーションを作成できるようにします。例はプログラマの作業、プログラムフローチャート、手順からできており、手順の各番号はフローチャート内の番号に対応しています。

本章を読む際には、NI-488.2 ソフトウェアのパッケージを手に取って本文に合わせてご確認ください。NI-488.2 ソフトウェアには 9 例すべての C 言語および BASIC のソースコードが収めてあります。プログラムは、最初はやさしく、段々と高度になるように配置されています。GPIB のプログラミングが初めての場合は、まず第 1 例 `simple.c` の内容と概念を理解してから第 2 例以降に進んでください。

NI-488.2 ソフトウェアには、以下のプログラム例が入っています。

- `simple.c` : 第 1 例のソースコードファイルです。ホストコンピュータと GPIB デバイスの間で通信を始める方法を説明します。
- `clr_trg.c` : 第 2 例のソースコードファイルです。GPIB デバイスをクリアする方法とトリガする方法を説明します。
- `asynch.c` : 第 3 例のソースコードファイルです。GPIB を使ってデータを転送しながら GPIB 以外のタスクを実行する方法を説明します。
- `eos.c` : 第 4 例のソースコードファイルです。EOS(ファイル終わり) 文字の考え方を説明します。
- `rqs.c` : 第 5 例のソースコードファイルです。GPIB SRQ 線を使ってサービスを要求している GPIB デバイスとの通信方法を説明します。NI-488 関数を使用しています。
- `easy4882.c` : 第 6 例のソースコードファイルです。NI-488.2 ルーチンの入門になっています。
- `rqs4882.c` : 第 7 例のソースコードファイルです。GPIB SRQ 線を使ってサービスを要求している GPIB デバイスと通信するには、どのように NI-488.2 ルーチンを使えばよいか説明します。
- `ppoll.c` : 第 8 例のソースコードファイルです。NI-488.2 ルーチンを使ってパラレルポールを実行する方法を説明します。
- `non_cic.c` : 第 9 例のソースコードファイルです。非コントローラアプリケーションで NI-488.2 ドライバをどのように使用するか説明します。

例 1:　基本的な通信

この例では、ホストコンピュータと GPIB デバイスとの通信を開始する上での基本事項に焦点を当てて説明します。

ある技術者が、GPIB マルチメータで電圧値を読み取って監視しなければなりません。コンピュータには IEEE 488.2 インタフェースボードを取り付けられています。NI-488.2 ソフトウェアがインストールされており、GPIB ケーブルでマルチメータの GPIB ポートがコンピュータと接続されています。

この技術者は、マルチメータのリモートプログラミングコマンドについての十分な知識があります。このコマンドはそれぞれのマルチメータに特有で、マルチメータのメーカに連絡すれば入手できます。

マルチメータで計測を行い、同時にその結果がコンピュータで記録されるように設定します。方法としては、まず簡単な高レベル GPIB コマンドを使ったアプリケーションを書きます。次に示す手順の番号は、図 2-1 のプログラムフローチャート内の番号に対応しています。

1. アプリケーションはコンピュータのインタフェースボードをオンラインにすることによって、GPIB を初期化します。
2. アプリケーションは次にマルチメータに命令を送り、自動レンジ設定モードで電圧を測定するよう設定します。
3. 次にアプリケーションは電圧測定をするようにマルチメータに命令を送ります。
4. アプリケーションは次に、マルチメータに対して、収集した測定データをコンピュータに送信するように命令を送信します。

   読み取り値が存在する限り、マルチメータに対する測定要求と読み取り (ステップ 3 ～ 4) は繰り返されます。
5. アプリケーションは、手順終了時の処理としてインタフェースボードをオフラインにし、元の状態に戻します。

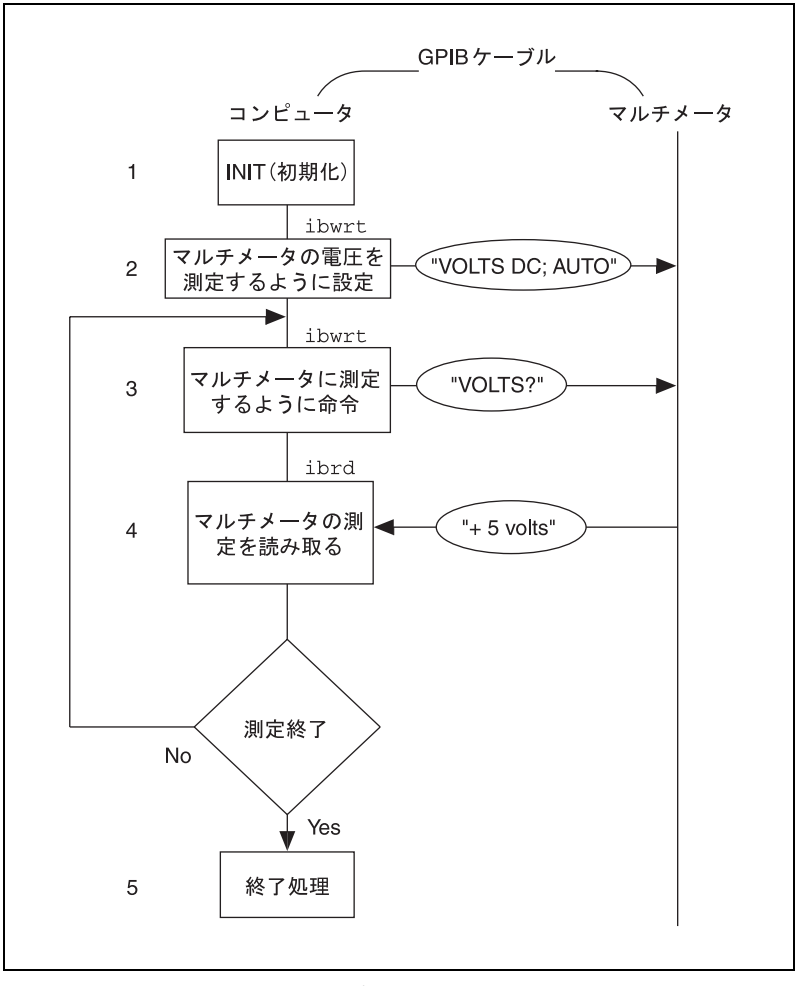


図 2-1　例 1 のプログラムフローチャート

例 2:　デバイスのクリアとトリガ

この例では、GPIB デバイスのクリア方法とトリガ方法について説明します。

新人の物理研究員が二人、GPIB デジタルオシロスコープの使い方を勉強している場合を想定しましょう。二人はパソコンに NI-488.2 ソフトウェアをロードし、GPIB ボードを GPIB デジタルオシロスコープに接続するところまではうまくできました。今、課題として渡されているのは、オシロスコープとそのコマンドセットの使い方を練習するための小さなアプリケーションプログラムを、高レベル GPIB コマンドで書くことです。次に示す手順の番号は、図 2-2 のプログラムフローチャート内の番号に対応しています。

1. アプリケーションはコンピュータのインタフェースボードをオンラインにすることによって、GPIB を初期化します。
2. 次にアプリケーションはオシロスコープに GPIB clear コマンドを送信します。このコマンドでオシロスコープの内部レジスタがクリアされ、デフォルト値 / デフォルト設定に初期化されます。
3. アプリケーションはオシロスコープに対し、トリガされるたびに波形を読み取るように命令するコマンドを送信します。このようにタスクをあらかじめ定義しておくことで、その実行にかかる時間を減らせます。こうすればオシロスコープをトリガするだけで、実行までできます。
4. アプリケーションはオシロスコープに GPIB trigger コマンドを送信し、オシロスコープはデータを収集します。
5. アプリケーションはオシロスコープに収集したデータを問い合わせます。オシロスコープはデータを送信します。
6. アプリケーションはオシロスコープからのデータを読み取ります。
7. アプリケーションは外部グラフィックルーチンを呼び出し、収集した波形を表示します。

   オシロスコープが必要なデータを全て収集し、それをコンピュータが受信し終わるまで、上記のステップ 4、5、6、7 が繰り返されます。
8. アプリケーションは、手順終了時の処理として、インタフェースボードをオフラインにし、元の状態に戻します。

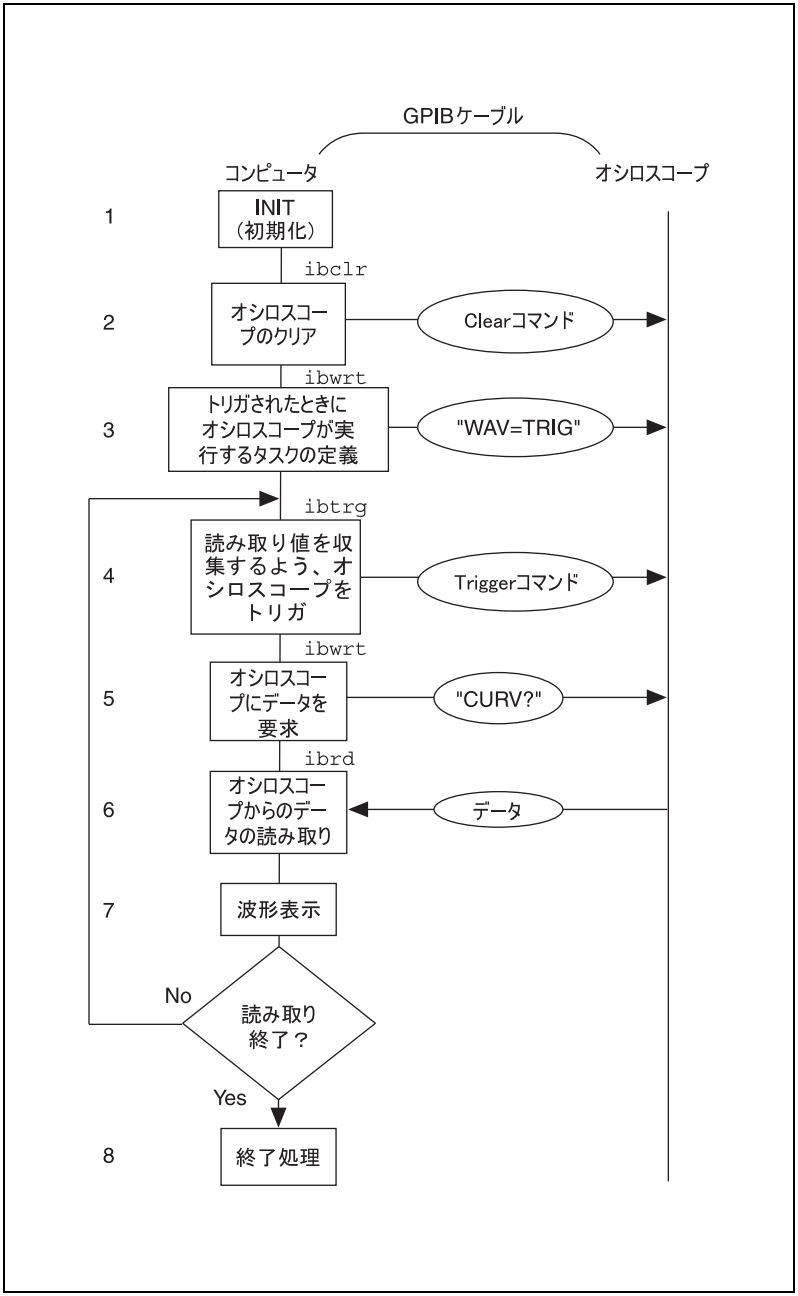


図 2-2　例 2 のプログラムフローチャート

例 3:　非同期入出力

この例では、アプリケーションがどのように GPIB デバイスとデータ転送を実行し、次に、直ちに非 GPIB 関連タスクの実行に移りながら、どのようにバックグラウンドで GPIB 入出力を実行するかを説明します。非同期動作モードと呼ばれるこのような動作は、要求した GPIB 作業が完了するまでにある程度の時間がかかる場合に特に有効です。

ここでは、神経科医が患者の脳に CAT スキャンを行い、患者の怪我の程度を調べるものと仮定します。神経科医はスキャンが 1 回済むごとに結果をカラー印刷します。作業はすべてコンピュータ制御されています。CAT スキャン装置は収集した画像を、GPIB カラープリンタが接続してあり、しかも NI-488.2 ソフトウェアをインストールしてあるコンピュータに送信します。神経科医はプリンタのユーザマニュアルに説明のあるコマンドを熟知しているものとします。そして、高レベル GPIB コマンドを使って書いたアプリケーションプログラムにより、スキャン画像の収集と印刷を実行します。次に示す手順の番号は、図 2-3 のプログラムフローチャート内の番号に対応しています。

1. アプリケーションはコンピュータのインタフェースボードをオンラインにすることによって、GPIB を初期化します。
2. スキャンにより画像が読み取られます。
3. アプリケーションは GPIB カラープリンタに今得られた画像を印刷するようコマンドを送信し、その後、入出力動作の終了を待たずにただちに復帰します。
4. アプリケーションは、得た画像をファイルに保存します。
5. アプリケーションは GPIB wait コマンドを発し、印刷動作が終了したかどうかを問い合わせます。wait コマンドが返したステータスが完了を意味するものであり (CMPL が返されます)、しかもスキャンをまだ続けるのであれば、スキャン画像の収集がすべて終了するまでステップ 2 ～ 5 が繰り返されます。ステップ 5 で wait コマンドが返したステータスが、印刷の終了を示すものでなければ、収集したスキャン画像に対して統計計算が実行され、ステップ 5 が繰り返されます。
6. アプリケーションは、手順終了時の処理として、インタフェースボードをオフラインにし、元の状態に戻します。

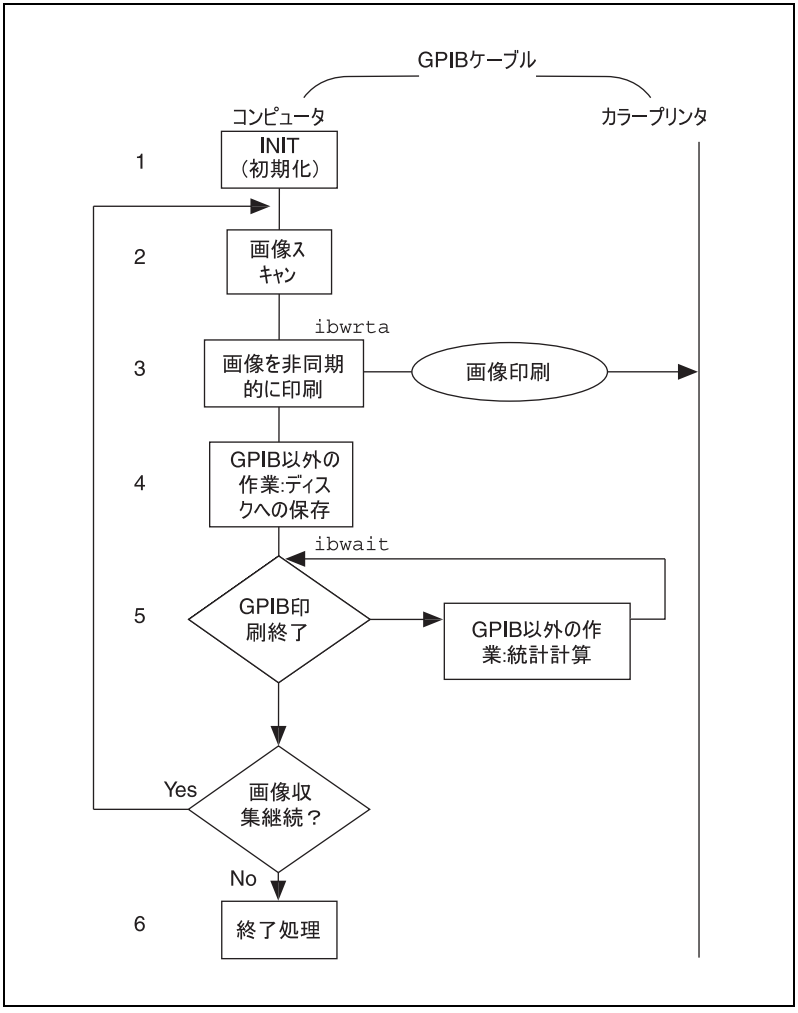


図 2-3　例 3 のプログラムフローチャート

例 4:　文字列の終わり (EOS) モード

この例では、文字列の終わりモードをどのように使えば、GPIB デバイスがデータ送信を終了したかどうかを検出できるかを説明します。

新聞記者が GPIB スキャナを使って写真をパソコンに読み取り、ニュース記事に使いたいと思っています。スキャナとコンピュータは GPIB ケーブルで接続してあります。記者の使っているアプリケーションは、同部署の見習い記者がスキャナの取扱説明書を読み、プログラム条件も理解して作成したものです。次に示す手順の番号は、図 2-4 のプログラムフローチャート内の番号に対応しています。

1. アプリケーションはコンピュータのインタフェースボードをオンラインにすることによって、GPIB を初期化します。
2. アプリケーションは GPIB clear メッセージをスキャナに送信し、スキャナを電源オン時のデフォルト状態に初期化します。
3. スキャナは、コマンドの終わりを示す区切り文字を検出しなければなりません。この場合、スキャナは、コマンドの最後に <CR><LF>(復帰を示す \r と改行を示す \n) があるものと考えて動作を決定します。アプリケーションは文字列の終わり (EOS) バイトを <LF> に設定します。改行コードは、スキャナに対して、もうデータがないことを伝えるもので、EOS バイトと呼ばれます。改行コードは、使用中の GPIB スキャナに対して文字列の終わり状態のフラグを立てます。コマンドを送信したときに EOI 線をアサートしても同じ結果が得られます。
4. スキャン解像度を別にすれば、デフォルト設定はすべて現在のタスク用として適当です。アプリケーションはスキャナにコマンドを送信し、スキャン解像度を変更します。
5. スキャナは change resolution コマンドのステータスを表した情報を送り返します。この情報はバイト列で、最後に文字列の終わり文字があり、アプリケーションに解像度の変更が済んだことを知らせます。
6. アプリケーションはスキャンコマンドをスキャナに送り、スキャンを開始します。
7. アプリケーションはスキャンデータを読み取り、コンピュータに送ります。
8. アプリケーションは、手順終了時の処理として、インタフェースボードをオフラインにし、元の状態に戻します。

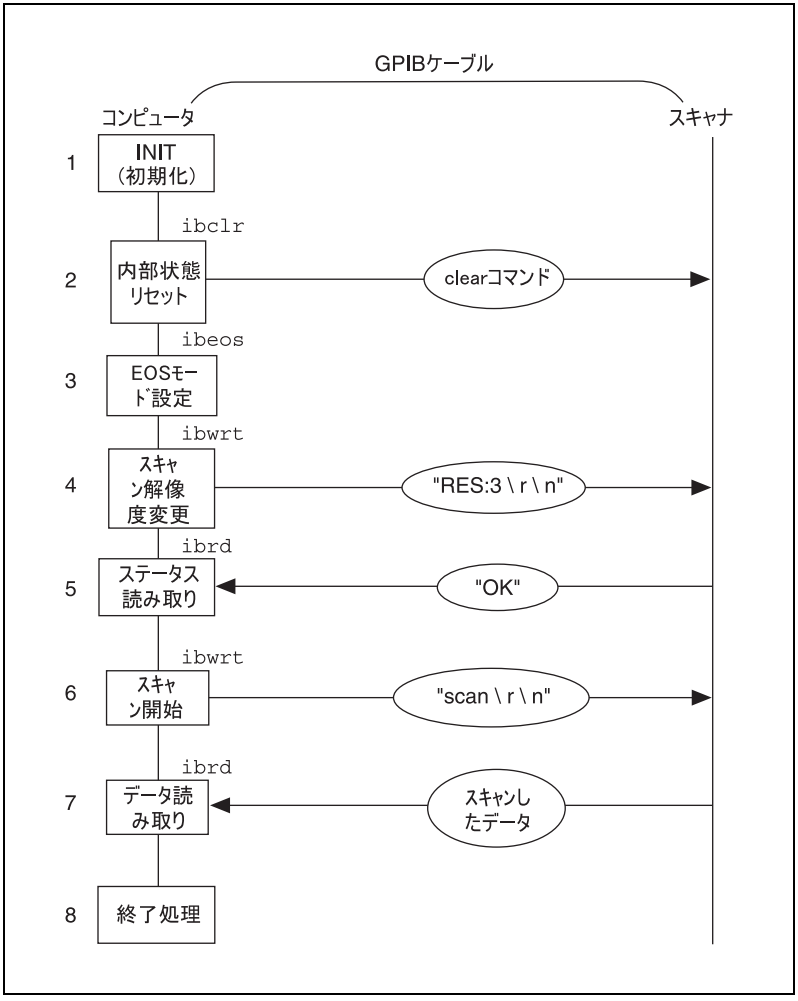


図 2-4　例 4 のプログラムフローチャート

例 5:　サービス要求

この例では、アプリケーションが、GPIB サービス要求 (SRQ) 線を使ってサービス処理を求めている GPIB デバイスとどのように通信するかを説明します。

グラフィックアートデザイナーが、コンピュータに保存してあるデジタル画像を GPIB デジタルフィルムレコーダを使ってカラーフィルムに転送する場合を考えます。フィルムレコーダの GPIB ポートとコンピュータの IEEE 488.2 インタフェースボードは GPIB ケーブルで接続してあります。コンピュータには NI-488.2 ソフトウェアをインストールしてあり、フィルムレコーダのユーザマニュアルを読んでプログラム規則についてもよく理解しています。カメラに新しいフィルムを入れ、高レベル GPIB コマンドを使って書いた簡単なアプリケーションを起動します。次にアプリケーションを使い、フィルムに画像を記録します。次に示す手順の番号は、図 2-5 のプログラムフローチャート内の番号に対応しています。

1. アプリケーションはコンピュータのインタフェースボードをオンラインにすることによって、GPIB を初期化します。
2. アプリケーションはデバイスクリア命令を発し、フィルムレコーダを準備完了状態にします。これでフィルムレコーダはデフォルト値を使って動作する準備が完了しました。(グラフィックアートデザイナーは以前にもフィルムレコーダをデフォルト値で操作したことがあるので、デフォルト値のままでフィルムのタイプに合っていることが分かっています。)
3. アプリケーションはフィルムを正規の位置にセットし、1 番目の画像が 1 番目のコマに割り当てられるようにします。これには、フィルムレコーダのプログラミングガイドに説明のある命令を使います。
4. アプリケーションは、フィルムレコーダが RQS(サービス要求) を送信してフィルム装填終了を知らせるのを待ちます。フィルムレコーダは、フィルム装填が終了すると GPIB SRQ 線をアサートします。
5. フィルムレコーダが GPIB SRQ 線をアサートすると、ただちにアプリケーションの RQS イベント待ちが終了します。アプリケーションは、フィルムレコーダに対し、シリアルポールステータスバイトの形で応答を返すよう指示する特殊コマンドメッセージを送信し、シリアルポールを実行します。このバイトに入っている情報は、フィルムレコーダがどのようなサービスを要求しているのかや、どの状態のフラグを立てているのかを表します。この例では、コマンドの終了を表しています。
6. デジタルフィルムレコーダへカラー画像を送信する際には、パスを 3 回連続して行い、それぞれのパスは画像の赤、緑、青の成分です。パス 1 回ごとに次のステップ 6a、6b、6c が繰り返されます。

a. アプリケーションはフィルムレコーダにコマンドを送信し、データを受信してパス画像を 1 つ作成させます。フィルムレコーダはパスが完了すると、ただちに SRQ 線をアサートします。

b. アプリケーションは RQS を待ちます。

c. アプリケーションは、SRQ 線がアサートされるとフィルムレコーダとシリアルポールを行い、ステップ 5 で説明したようにサービスの要求をしているかどうかを確認します。

7. アプリケーションはフィルムレコーダにコマンドを送信し、フィルムを 1 つコマ送りします。コマ送りはフィルムが終わらない限り失敗することはありません。

8. アプリケーションは RQS を待ちます。フィルムレコーダが SRQ 線をアサートしてコマ送りの完了を知らせると、RQS 待ちを終了します。

9. アプリケーションの RQS 待ちが終了すると、ただちにアプリケーションはフィルムレコーダとのシリアルポールを行い、ステップ 5 のサービス要求をしているかどうか確認します。シリアルポールによって返されたステータスバイトを見れば、フィルムレコーダが要求通りコマ送りを完了したのか、またはフィルムが終わりに来たのでコマ送りできなかったのかが分かります。ステップ 6 ～ 9 は、カメラにフィルムが残っており、しかも画像の記録を継続する必要がある限り繰り返されます。

10. アプリケーションは、手順終了時の処理として、インタフェースボードをオフラインにし、元の状態に戻します。

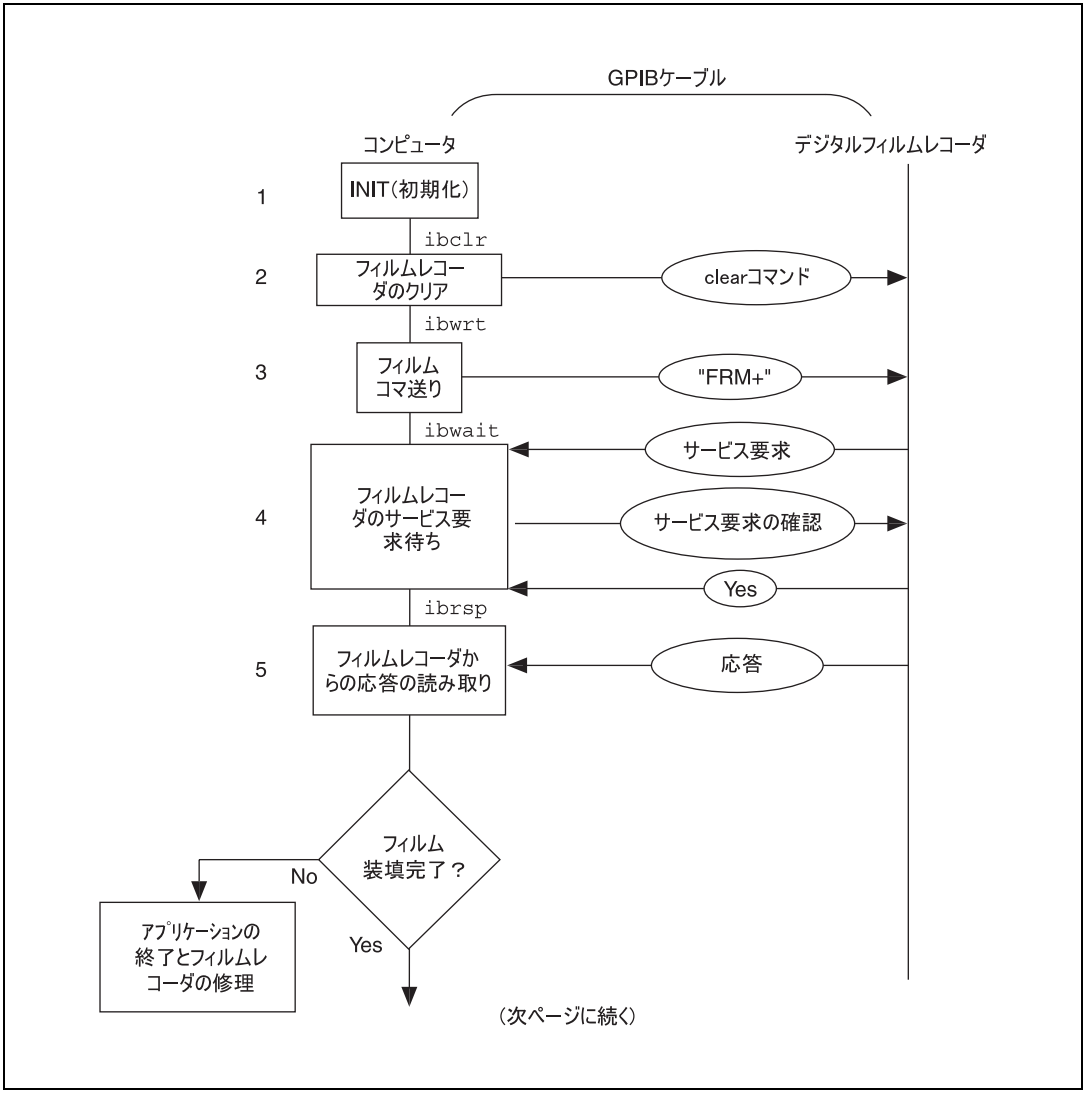


図 2-5　例 5 のプログラムフローチャート

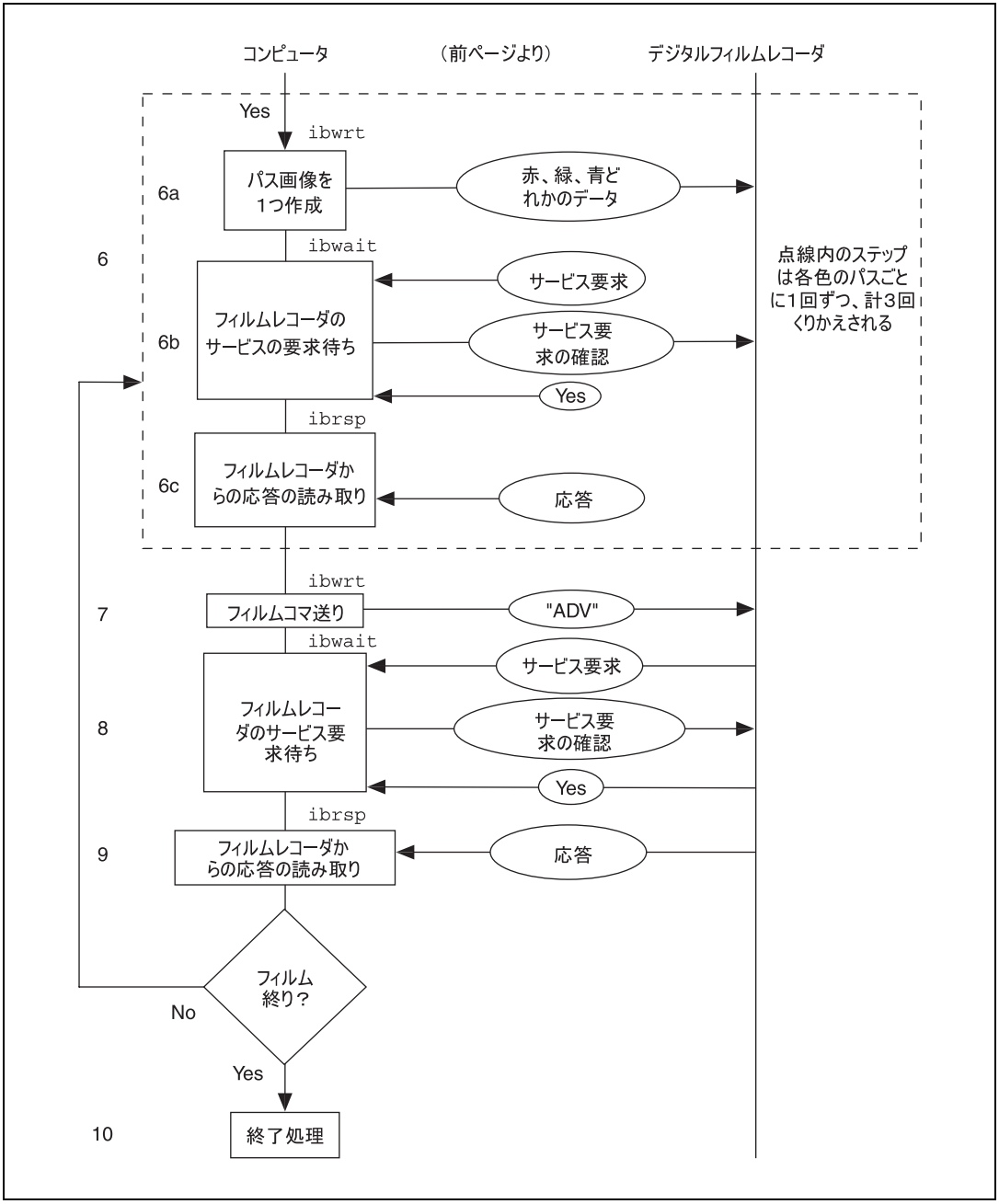


図 2-5　例 5 のプログラムフローチャート (続き)

例 6:　IEEE 488.2 準拠デバイスとの簡単な通信

この例では IEEE 488.2 準拠デバイスとの通信の基本的な説明をします。

金属工場の試験技師が、生産した金属棒の強度を測定するのに IEEE 488.2 規格に準拠した引張り強度テスタを使う場合を考えます。引張り強度テスタは何台もあり、それがすべて、IEEE 488.2 インタフェースボードを付けた中央コンピュータに接続されています。テスタが大きいので、技師がそれぞれのテスタのアドレススイッチに手を伸ばすのは大変です。ずっとこのテスタを使っていくことを考えると、各テスタの GPIB アドレススイッチを一度に設定できるようにしなければなりません。自分で書いた簡単なアプリケーションプログラムを使えば、それが可能になります。次に示す手順の番号は、図 2-6 のプログラムフローチャート内の番号に対応しています。

1. アプリケーションはコンピュータのインタフェースボードをオンラインにすることによって、GPIB を初期化します。
2. アプリケーションは、コマンドを送信し、GPIB 上でリスナ状態にあるデバイスがあるかどうか検出し、リスナデバイスすべてのアドレスのリストを作成します。
3. アプリケーションはステップ 2 で GPIB 上に検出したリスナデバイスに ID 問い合わせ ("`IDN?`") を送信します。
4. アプリケーションは、ステップ 3 の問い合わせに応答して返された ID 情報を読み取ります。
5. アプリケーションは、手順終了時の処理として、インタフェースボードをオフラインにし、元の状態に戻します。

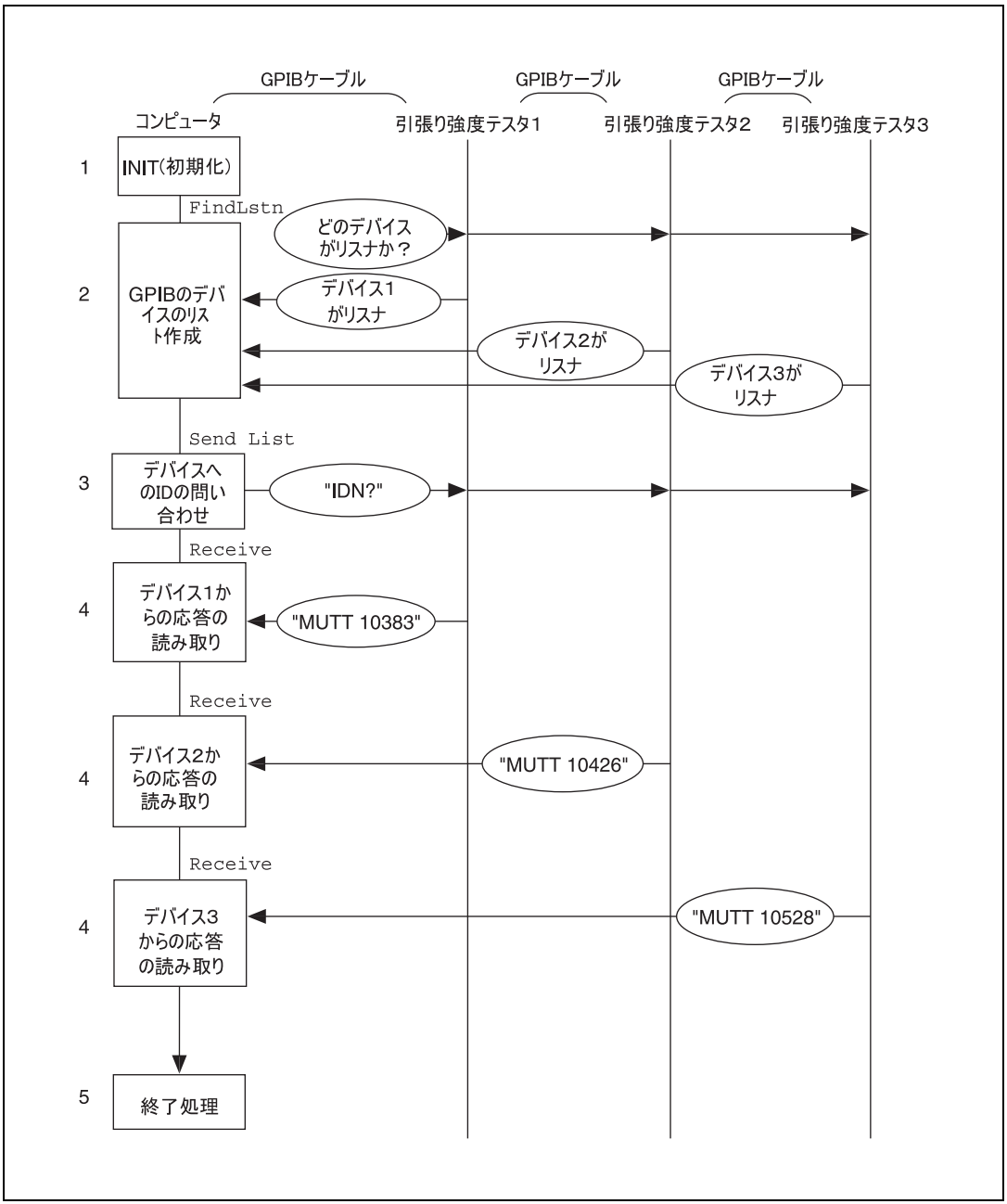


図 2-6　例 6 のプログラムフローチャート

例 7:　NI-488.2 ルーチンを使ったシリアルポール

この例では NI-488.2 ルーチンをどのように利用すれば複数デバイスのシリアルポールが簡単になるかを説明します。

製菓会社で、キャンディーのシロップの粘度を測定するのに GPIB ひずみ計を使う場合を想定します。工場にはシロップ用の大きなミキサーが 4 台あります。高品質のキャンディーを作るにはシロップにある程度の粘度がなくてはなりません。シロップの硬度は、ミキサーのアームを動かすのに必要な圧力をひずみ計で測定すれば分かります。シロップがある粘度に達したらミキサーから取り出して、代わりに次回分のシロップを入れます。GPIB ひずみ計は、IEEE 488.2 インタフェースボードを付け、NI-488.2 ソフトウェアをインストールしたコンピュータに接続されています。このプロセスを制御するアプリケーションは、NI-488.2 ルーチンを使って IEEE 488.2 準拠ひずみ計との通信を行います。次に示す手順の番号は、図 2-7 のプログラムフローチャート内の番号に対応しています。

1. アプリケーションはコンピュータのインタフェースボードをオンラインにすることによって、GPIB を初期化します。
2. アプリケーションはひずみ計の設定を行い、圧力読み取り値が大きくなったり機械故障が発生したときに、ひずみ計がサービスを要求するようにします。ひずみ計は SRQ 線をアサートすることでサービス要求を表します。
3. アプリケーションはひずみ計から圧力読み取り値が大きくなったことが示されるまで待ちます。待ち状態は SRQ 線がアサートされるとすぐに終了します。
4. アプリケーションは各ひずみ計とシリアルポールを行い、サービスを要求したかどうかを確認します。
5. どのひずみ計がサービスを要求しているかが分かると、アプリケーションはそのひずみ計から読み取り値を受信します。
6. 読み取り値が指定の粘度に達していれば、コンピュータ画面にはダイアログウィンドウが表示され、ミキサーのオペレータに現在のシロップを取り出して次回分のシロップを入れるように指示します。もし希望の値に達していなければ、ダイアログウィンドウは別の指示を表示します。

   ステップ 3 ～ 6 はミキサーが作動している限り繰り返されます。
7. 最終回分のシロップの加工が終了すると、アプリケーションは手順終了時の処理として、インタフェースボードをオフラインにし、元の状態に戻します。

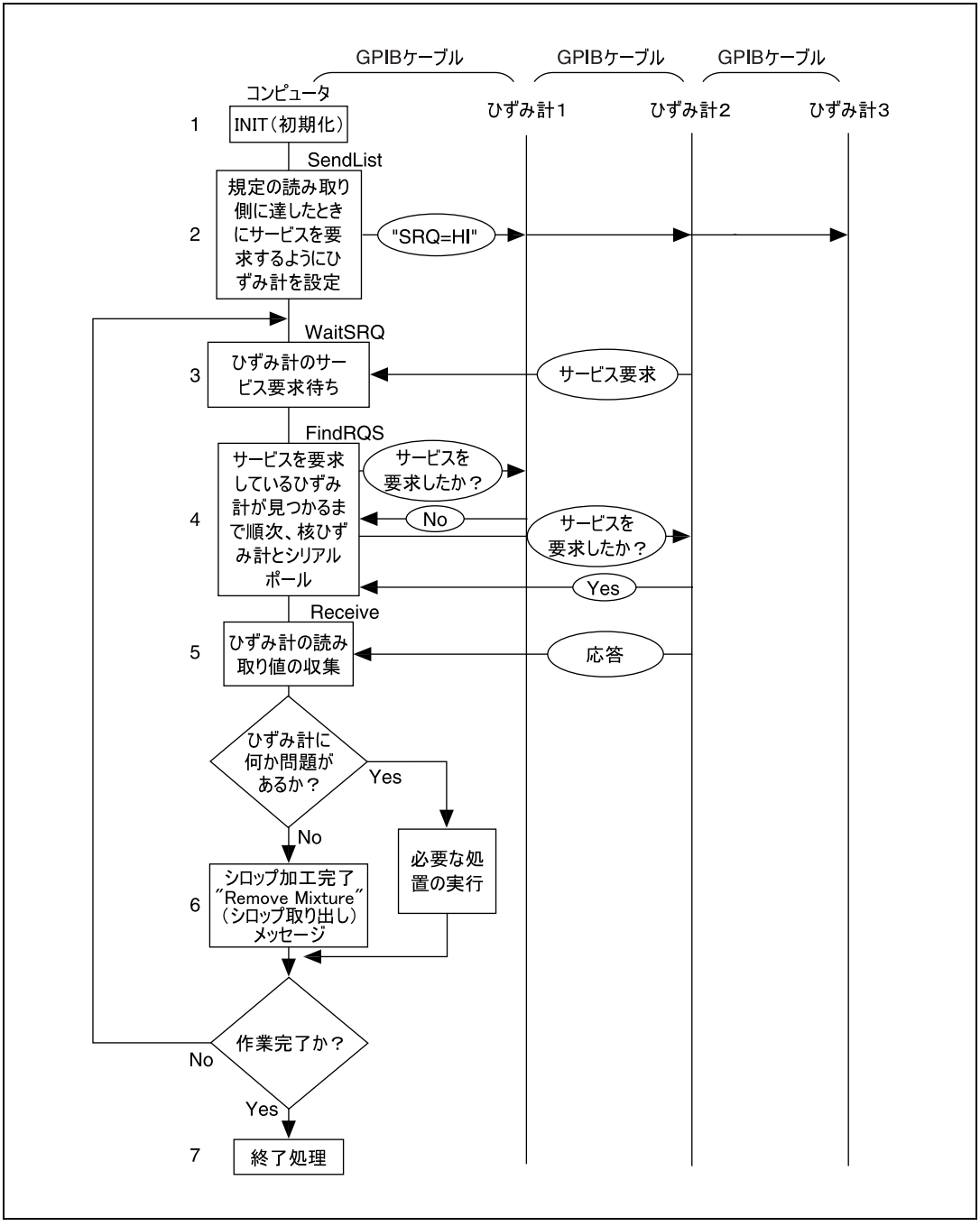


図 2-7　図 2-7　例 7 のプログラムフローチャート

例 8:　パラレルポール

この例では、NI-488.2 ルーチンを使ってパラレルポールという方法で複数の IEEE 488.2 準拠デバイスから 1 度に情報を収集する方法を説明します。

ある合金を製造するには、3 種類の金属をそれぞれ別の温度に熱した後に混合します。バットを 3 つ使い、それぞれに別の金属を入れます。バット毎に GPIB 溶融金属監視装置を 1 台配置します。監視装置は GPIB 温度変換器と GPIB 電源で構成されています。温度変換器で金属の温度を測定します。電源は、溶融金属温度が規定値に達したときにモータを起動して、溶融金属を型に流し込むのに使います。3 台の監視装置はどれも、NI-488.2 ソフトウェアをインストールしたコンピュータの IEEE 488.2 インタフェースボードに接続されています。NI-488.2 ルーチンを使ったアプリケーションがこの 3 台の監視装置を作動させます。アプリケーションはパラレルポールによって監視装置 3 台から情報を収集し、溶融金属をいつ混合タンクに注ぎ込むかを決めます。次に示す手順の番号は、図 2-8 のプログラムフローチャート内の番号に対応しています。

1. アプリケーションはコンピュータのインタフェースボードをオンラインにすることによって、GPIB を初期化します。
2. アプリケーションは、1 台目の監視装置の温度変換器について、パラレルポール時にその温度変換器が 8 本の GPIB データ線のどれを使うかを設定します。また、温度スレッショルドも設定します。温度変換器は、規定の温度スレッショルドに達するとステータス (ist) ビットが true(真) になるように温度変換器メーカーがあらかじめ設定しています。また、温度変換器のステータスモードは、「データ線をアサート」と構成されています。したがって、パラレルポールを実行した時、温度がスレッショルドを超えていると、温度変換器はデータ線をアサートします。
3. アプリケーションは、2 台目の監視装置の温度変換器についてもパラレルポール時の設定を行います。
4. アプリケーションは、3 台目の監視装置の温度変換器についてもパラレルポール時の設定を行います。
5. アプリケーションは、金属加熱中は GPIB 以外の作業を行います。
6. アプリケーションは 3 台の温度変換器とパラレルポールを行い、それぞれの金属が規定の温度に達しているかどうか調べます。温度変換器は金属が温度スレッショルドに達していれば、構成ステップでデータ線をアサートします。
7. パラレルポールへの応答を調べた結果、3 種類の金属がどれも規定温度に達していれば、アプリケーションは 3 台の電源にコマンドを送信し、その電源を入れます。これでモータが作動して金属は型に流し込まれます。

3 種類すべての金属の温度がそれぞれの規定値に達しない限りは、ステップ 5 ～ 6 が繰り返されます。そうしないと、3 種類の金属がうまく混合されません。

8. アプリケーションはすべての温度変換器の設定を解除し、パラレルポールに関与しないようにします。
9. アプリケーションは、手順終了時の処理として、インタフェースボードをオフラインにし、元の状態に戻します。

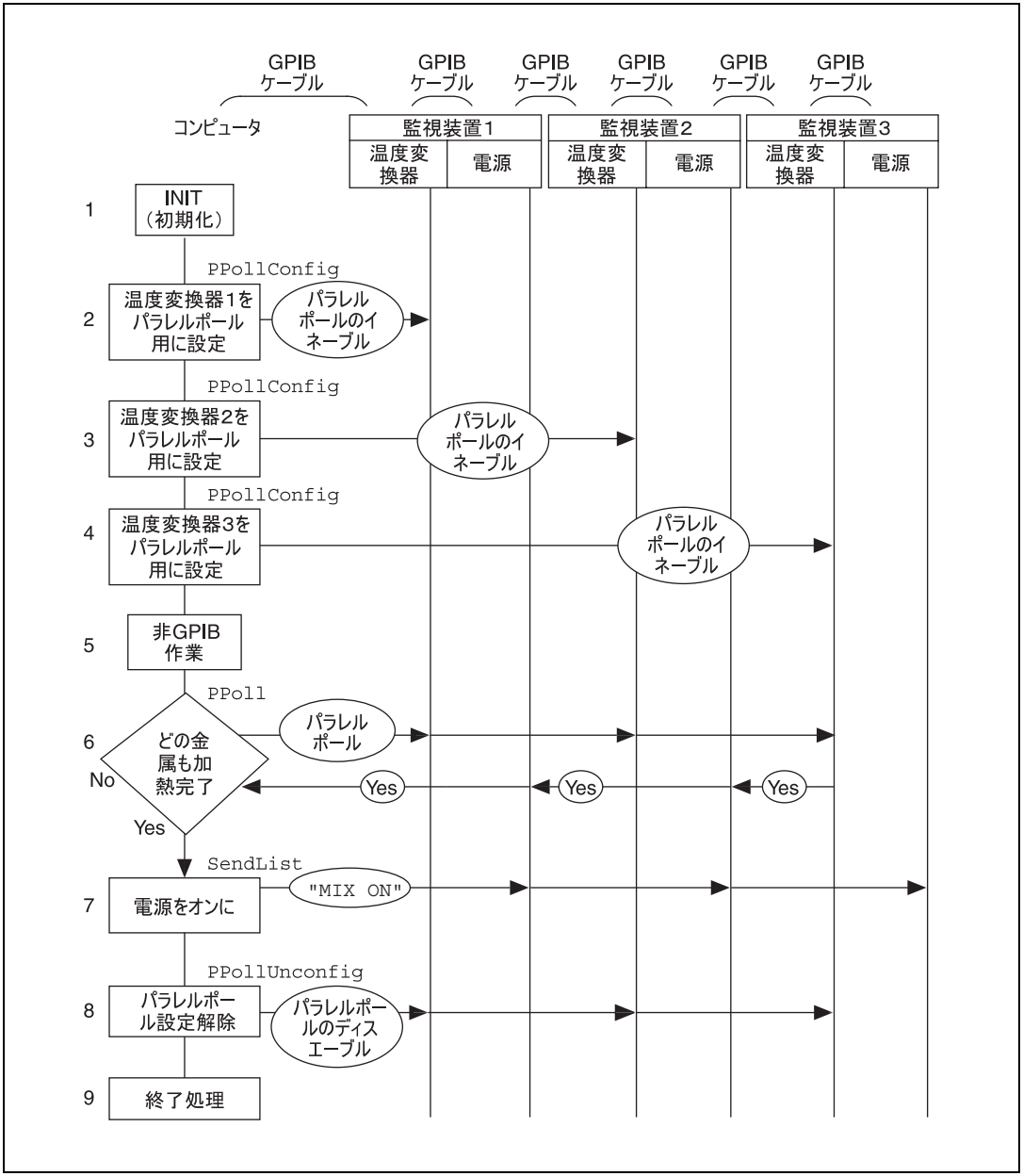


図 2-8　例 8 のプログラムフローチャート

例 9:　コントローラでないデバイスのエミュレーション

この例では、NI-488.2 ソフトウェアを使って、GPIB コントローラでない GPIB デバイスをエミュレートする方法を説明します。

ソフトウェア技術者が調査プロジェクト用に GPIB デバイスをエミュレートするファームウェアを作成し、簡単な GPIB 呼び出しを行うアプリケーションを使ってそれをテストしていると想定します。次に示す手順の番号は、図 2-9 のプログラムフローチャートの番号に対応しています。

1. アプリケーションは、デバイスをオンラインにします。
2. アプリケーションは、デバイスがリスナとしてアドレスされるか、トーカとしてアドレスされるか、GPIB クリアメッセージを受信するか、の 3 つのうちいずれかのイベントが発生するのを待ちます。
3. いずれかのイベントが発生すると、ただちにアプリケーションはそのイベントに合った動作を実行します。デバイスがクリアされたら、アプリケーションはデバイスの内部状態をリセットしてデフォルト値に戻します。デバイスがトーカとしてアドレスされたら、コントローラにデータを書き込みます。リスナとしてアドレスされたら、コントローラから新しいデータを読み取ります。

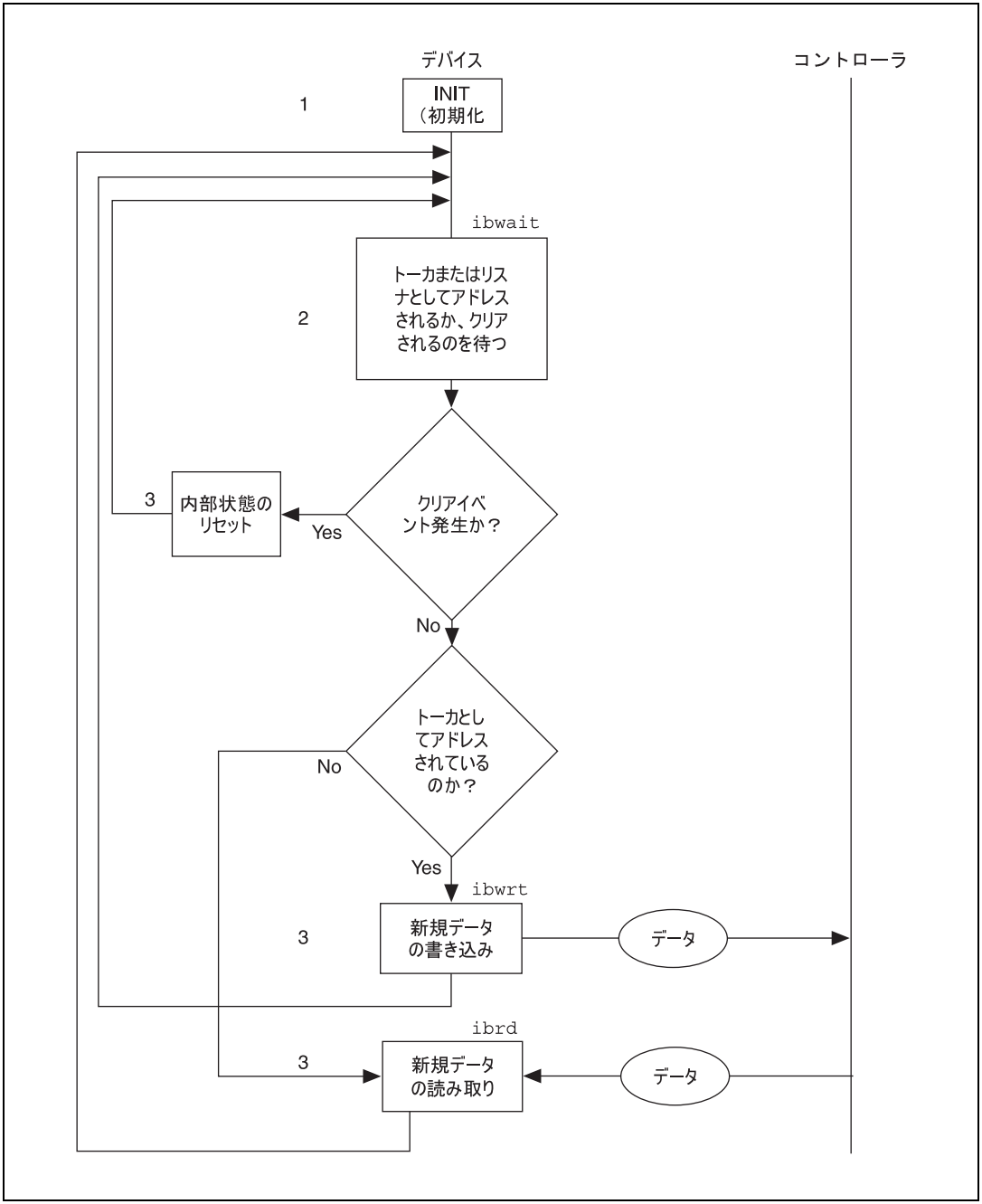


図 2-9　例 9 のプログラムフローチャート

3

アプリケーションの開発

本章では、NI-488 関数と NI-488.2 ルーチンを使って GPIB アプリケーションプログラムを開発する方法を説明します。

プログラミング方法の選択

GPIB を使った通信が必要なプログラムは、NI-488.2 言語インタフェースまたはユニバーサル言語インタフェース (ULI) を使えば NI-488 ドライバにアクセスできます。GPIB アプリケーションを新しく作成する場合は、NI-488.2 言語インタフェースをお勧めします。

NI-488.2 言語インタフェースの使い方

NI-488.2 ソフトウェアには、アプリケーションのニーズに応えられるように、サブルーチンが 2 種類入っています。ほとんどのアプリケーションには NI-488 関数だけで十分ですが、インタフェースボードと複数のデバイスを使った複雑なシステム構成の場合には、NI-488.2 ルーチンが必要です。

次に、NI-488 関数と NI-488.2 ルーチンとの違いについて説明します。

NI-488 関数の使い方：1 ボードに 1 デバイスの場合

お使いのシステムでボード 1 枚に対してデバイス 1 台しか使っていないときは、プログラミングにはたいてい NI-488 関数だけで十分です。
NI-488 関数を使った方が便利な理由には、次のようなものもあります。

- NI-488 非同期入出力関数 (ibcmda, ibrda, ibwrta) を使えば、CPU で非 GPIB タスクの実行を制御しながら、入出力シーケンスを起動できます。
- NI-488 関数にはファイル転送関数 (ibrdf, ibwrtf) が組み込まれています。
- NI-488 関数を使えば、GPIB バスを特殊な方法で制御したり、IEEE 488.2 に準拠しないデバイスと通信したりすることが可能です。

NI-488 関数は、GPIB 管理動作の多くを含んだハイレベル (つまりデバイス) 関数と、NI-488.2 ルーチンより優れた GPIB 制御能力を持つローレベル (つまりボード) 関数からなっています。次に、この異なった関数タイプについて説明します。

NI-488 デバイス関数

デバイス関数はハイレベル関数で、デバイスからの読み取り、デバイスへの書き込み、ステータスを尋ねるポール、などのバス管理動作を処理するコマンドを自動的に実行します。デバイス関数を使うならば、GPIB プロトコルやバス管理を勉強する必要はありません。デバイスレベル呼び出しと、それがどのように GPIB を管理するかについては、第 7 章「GPIB プログラミングテクニック」の「デバイスレベル呼び出しとバス管理」の項を参照してください。

NI-488 ボード関数

ボード関数はローレベル関数で、基本的な GPIB 動作を実行します。ボード関数はインタフェースボードに直接アクセスし、ユーザにアドレッシングとバス管理プロトコルを処理するように求めます。ハイレベルデバイス関数が使用条件に合わない場合、柔軟性と制御能力の高いローレベル関数ならば、以下のような状況にも対処できます。

- 非準拠デバイス (非 IEEE 488.2 デバイス) との通信。
- 様々なローレベルボード構成の変更。
- 特殊な方法を使ったバス管理。

NI-488 ボード関数は NI-488.2 ルーチンのシーケンスと互換性があり、その中に併存させることができます。NI-488.2 ルーチンのシーケンス中でボード関数を使うと、ボード記述子を得るために ibfind を呼び出す必要がなくなります。単にボード関数呼び出しの第 1 パラメータとしてボード指標を使うだけで済みます。このような柔軟性があるので、NI-488.2 ルーチンだけでは解決できない非標準つまり特殊な状況にも対処できます。

NI-488.2 ルーチンの使い方：ボードまたはデバイスが複数の場合

お使いのシステムに複数のデバイスにアクセスしなければならないボードがある場合には、NI-488.2 ルーチンを使ってください。NI-488.2 ルーチンは 1 回の呼び出しで以下のようなタスクが実行できます。

- バス上の全リスナの検出。
- サービスを要求しているデバイスの検出。
- SRQ 線のステータスの決定、SRQ 線アサート待ち
- 複数デバイスをリスナとしてアドレス。

ユニバーサル言語インタフェース (ULI) の使い方

新規に GPIB アプリケーションを書く場合、NI-488.2 言語インタフェースを使ってください。HP スタイルの呼び出しを使った既存アプリケーションがあるなら、ユニバーサル言語インタフェース (ULI) を使えば NI-488.2 ソフトウェアにアクセスできます。ULI の詳細については、付録 C「ユニバーサル言語インタフェース」を参照してください。

グローバル変数を使ったステータス確認

NI-488 関数と NI-488.2 ルーチンは、使用中のデバイスやボードのステータスを反映させてグローバル変数を更新します。ステータスワー (ibsta)、エラー変数 (`iberr`)、カウント変数 (`ibcnt`, `ibcntl`) は、アプリケーションプログラムの動作についての有益な情報です。プログラムは、これらの変数を頻繁にチェックするように作成してください。以下に、これらのグローバル変数について説明し、アプリケーションプログラムでどのように使えばよいかを述べます。グローバル変数の値は、アプリケーションの実行中いつでも表示することができます。

ステータスワード - ibsta

関数は全てグローバルステータスワード `ibsta` を更新します。`ibsta` には GPIB と GPIB ハードウェアの状態に関する情報が入っています。NI-488 関数は多くが `ibsta` の値を返します。`ibsta` に返された状況を確認して処理を継続するかどうかを決めたり、呼び出しの後で `ibsta` を確認してプログラムをデバッグしたりできます。

`ibsta` は 16 ビットの値です。ビット値が 1 であれば、ある特定の条件が発生しています。ビット値がゼロであれば、その条件は発生していません。`ibsta` の各ビットは、NI-488 デバイス呼び出し (dev)、NI-488 ボード呼び出しおよび NI-488.2 呼び出し (brd)、またはその両方の呼び出し (dev, brd) で設定されます。

それぞれのビット位置が表す条件、ビットのニモニック、ビットが設定される呼び出しのタイプを表 3-1 に示します。ステータス条件の詳細については、付録 A「ステータスワード状況」を参照してください。

表 3-1　ステータスワード (ibsta) のレイアウト

ニモニック	ビット位置	16 進値	タイプ	説明
ERR	15	8000	dev,brd	GPIB エラー
TIMO	14	4000	dev,brd	制限時間を超過しました
END	13	2000	dev,brd	END または EOS が検出されました
SRQI	12	1000	brd	SRQ 割り込みを受信しました
RQS	11	800	dev	デバイスがサービスを要求しています
SPOLL	10	400	brd	コントローラがボードをシリアルポールしました
EVENT	9	200	brd	DCAS, DTAS, IFC のいずれかのイベントが発生しました
CMPL	8	100	dev,brd	入出力が完了しました
LOK	7	80	brd	ロックアウト状態です
REM	6	40	brd	リモート状態です
CIC	5	20	brd	コントローラインチャージです
ATN	4	10	brd	アテンションがアサートされています
TACS	3	8	brd	トーカです
LACS	2	4	brd	リスナです
DTAS	1	2	brd	デバイストリガ状態です
DCAS	0	1	brd	デバイスクリア状態です

添付のディスクに入っている言語ヘッダファイルには、`ibsta` のニモニック定数が記録されています。その数値またはニモニック定数を使えば、`ibsta` の中でのビット位置を確認することができます。たとえば、ビット位置 15(16 進 "8000") の場合、これは GPIB エラーの検出を示します。このビットのニモニックは ERR です。GPIB エラーが発生しているかどうかを確認するには、NI-488 関数や NI-488.2 ルーチンの後に以下の文のどちらかを入れます。

```
if (ibsta & ERR) gpiberr();
```

または

```
if (ibsta & 0x8000) gpiberr();
```

ここで gpiberr() はエラー処理ルーチンです。

エラー変数 - iberr

ステータスワード (ibsta) に ERR ビットが設定されたら、GPIB エラーが発生しています。エラーが発生したときには、iberr の値がエラーのタイプを表します。

メモ　iberr の値は、ERR ビットが設定されたとき、つまりエラーが発生したときのみ有効で、エラーのタイプを示します。

エラーコードやその対策の詳細については、第 4 章「アプリケーションのデバッグ」または付録 B「エラーコードと対処」を参照してください。

カウント定数 - ibcnt と ibcntl

カウント定数は、読み取り関数、書き込み関数、コマンド関数のいずれかが実行されると更新されます。ibcnt は 16 ビットの整数で、ibcntl は 32 ビットの整数です。データを読み取るときには、カウント定数は読み取り済みバイト数を示します。データやコマンドを送信するときには、カウント定数は送信済みバイト数を表示に反映させます。

アプリケーションプログラムでカウント定数を使い、計測器から受け取った ASCII 文字列のデータをヌル終了することができます。たとえばデータを文字配列の形で受け取ったときには、ibcntl を使ってその文字配列をヌル終了し、測定値を画面上に次のように表示することができます。

```
char rdbuf[512]
ibrd (ud,rdbuf, 20L);
if (!(ibsta & ERR)){
      rdbuf[ibcntl] = '\0';
       printf (Read:   %s\n", rdbuf);
       }
       else {
            error();
}
```

ibcntl は、受信済みバイト数です。データは指標ゼロの配列から始まるので、ibcntl は、文字列の終わりを示すヌル文字の位置になります。

ibic を使ったデバイスとの通信

アプリケーションプログラムを書き始める前に ibic ユーティリティを使う必要がある場合があります。ibic(インタフェースバス対話式制御) を使えば、アプリケーションプログラムを使わなくても、キーボードで計測器と通信できます。GPIB アプリケーションを開発する前に ibic を使ってみれば、計測器との通信方法や、プログラミングに必要な条件を知ることができます。デバイス通信における個別の指示については、その計測器デバイスの取扱説明書を参照してください。ibic の使用方法や細かな使用例については、第 5 章「ibic - インタフェースバス対話式制御ユーティリティ」を参照してください。

デバイスとの ibic による通信方法が理解できたら、アプリケーションプログラムを書く準備が整ったことになります。

NI-488 アプリケーションの書き方

ここでは、アプリケーションプログラムに組み込まなければならない項目、一般的なプログラム手順、NI-488 プログラム例について説明します。このマニュアルでは、例題のコードは標準的な C 言語インタフェースを使った C 言語で書いてあります。NI-488.2 ソフトウェアには、C 言語で書いたこの例のソースコード (devsamp.c) と、同じ例を BASIC で書いたソースコード (devsamp.bas) が入っています。

組み込む項目

- GPIB ヘッダファイル。このファイルには、アプリケーションプログラムで使う NI-488 関数と定数のプロトタイプ (原型) が入っています。
- NI-488 関数呼び出しの後のエラーチェック。
- GPIB エラーを処理する関数の宣言と定義。この関数はデバイスをオフラインにし、アプリケーションを閉じます。関数が以下のように宣言されると、

```
void gpiberr (char *msg);   /* function prototype */
```

アプリケーションは関数を次のように呼び出します。

```
if (ibsta & ERR) {
    gpiberr("GPIB error");
}
```

NI-488 プログラムシェル

図 3-1 に、デバイスレベル NI-488 関数を使ってアプリケーションプログラムを作成するための手順のフローチャートを示します。

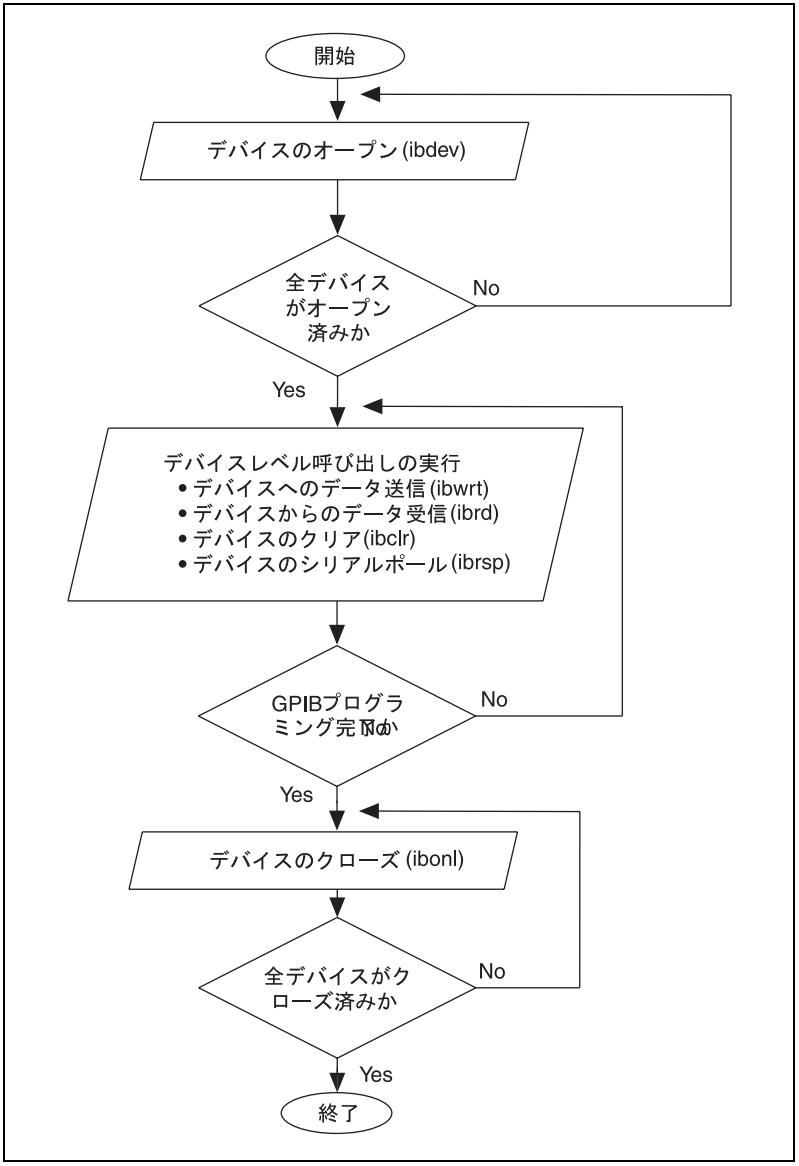


図 3-1　NI-488 デバイス関数を使った一般的なプログラムシェル

一般的なプログラム手順とその例

次の手順は、アプリケーションでの NI-488 デバイス関数の使い方を示したものです。この例ではデジタルマルチメータを構成し、電圧値を 10 回読み取り、その測定値の平均を計算します。

ステップ 1　デバイスのオープン

最初の NI-488 関数呼び出しは ibdev に対して行い、デバイスをオープンします。

```
ud = ibdev(0, 1, 0, T10s, 1, 0);

if (ibsta & ERR) {
       gpiberr("ibdev error");

}
```

ibdev 関数に入力した引数には以下のような意味があります。

0	-	GPIB0 のボード指標
1	-	デバイスの 1 次 GPIB アドレス
0	-	デバイスの 2 次 GPIB アドレスは、なし
T10s	-	入出力タイムアウト値 (10 秒)
1	-	デバイスに書き込むとき、最終バイトと一緒に END メッセージを送信
0	-	EOS 検出モードをディスエーブル

ibdev を呼び出すと、ドライバは自動的にインタフェースクリア (IFC) メッセージを送信してデバイスをリモートプログラミング状態にし、GPIB の初期化を行います。

ステップ 2　デバイスのクリア

デバイスをアプリケーションに合わせて構成する前に、必ずそのデバイスを 1 回クリアしてください。クリアすると内部機能がリセットされ、デフォルト状態に戻ります。

```
ibclr(ud);
if (ibsta & ERR) {
    gpiberr("ibclr error");
}
```

ステップ 3　デバイスの構成

デバイスをオープンしてクリアすれば、そのデバイスはコマンドを受け取る準備が完了したことになります。計測器を構成するには、ibwrt 関数で、そのデバイス特有のコマンドを送信します。お使いの計測器で使用できるコマンドバイトについては、その計測器のユーザマニュアルを参照してください。

```
ibwrt(ud, "*RST; VAC; AUTO; TRIGGER 2; *SRE 16", 35L);
if (ibsta & ERR) {
    gpiberr("ibwrt error");
}
```

この例のプログラミング命令はマルチメータをリセットします (*RST)。マルチメータは、自動範囲設定 (AUTO) を使って交流電圧 (VAC) を測定し、GPIB インタフェースボードからのトリガを待った上で計測を始め (TRIGGER 2)、測定が完了してマルチメータが測定結果送信の準備ができると SRQ 線をアサート (*SRE 16) するように指示されています。

ステップ 4　デバイスのトリガ

トリガを待つようにデバイスを構成した場合、測定値を読み取る前にデバイスにトリガコマンドを送信しなければなりません。そして、デバイスに、次にトリガされた読み取り値を GPIB 出力バッファに送信するように指示します。

```
ibtrg(ud);
if (ibsta & ERR) {
    gpiberr("ibtrg error");
}

ibwrt(ud,"VAL1?", 5L);
if (ibsta & ERR) {
    gpiberr("ibwrt error");
}
```

ステップ 5　測定の待機

デバイスをトリガした後、デバイスが測定値の送信準備を完了すると RQS ビットが設定されます。RQS を検出するには、ibwait 関数を使い

ます。2 番目のパラメータは、ユーザが何を待っているのかを表します。ibwait 関数は入出力タイムアウト値を超過したときにも値を返すことに注意してください。

```
printf("Waiting for RQS...\n");
ibwait (ud, TIMO | RQS);
if (ibsta & (ERR | TIMO)) {
    gpiberr("ibwait error");
}
```

SRQ が検出されたら、計測器をシリアルポールして、測定データが有効かどうか、または障害が発生しているかどうかを検出します。IEEE 488.2 計測器では、メッセージ有効 (MAV) ビット、つまり計測器から受信するステータスバイトのビット 4 を確認すれば、その状態を調べられます。

```
ibrsp (ud, &StatusByte);
if (ibsta & ERR) {
  gpiberr("ibrsp error");
}

if ( !(StatusByte & MAVbit)) {
    gpiberr("Improper Status Byte");
    printf("  Status Byte = 0x%x\n", StatusByte);
}
```

ステップ 6　測定の読み取り

データが有効であれば、計測器から測定値を読み取ります。
(AsciiToFloat はヌル終了の文字列を入力として読み取り、それが意味する浮動小数点数を出力します。)

```
ibrd (ud, rdbuf, 10L);
if (ibsta & ERR) {
    gpiberr("ibrd error");
}

rdbuf[ibcntl] = '\0';
printf("Read: %s\n", rdbuf);
    /*  Output ==> Read: +10.98E-3  */

sum += AsciiToFloat(rdbuf);
```

ステップ 7　データの処理

測定値を 10 回読み取るまで、ステップ 4 ～ 6 をループして繰り返します。それから、読み取り値の平均を次のように表示します。

```
printf("The average of the 10 readings is %f\n", sum/10.0);
```

ステップ 8　デバイスをオフラインにする

最終ステップとして、ibonl 関数でデバイスをオフラインにします。

```
ibonl (ud, 0);
```

NI-488.2 アプリケーションの書き方

ここでは、NI-488.2 ルーチンを使ったアプリケーションプログラムに組み込まなければならない項目や、一般的なプログラム手順、そして NI-488.2 の例について説明します。このマニュアルでは、例題のコードは標準の C 言語インタフェースを使った C 言語で書いてあります。NI-488.2 ソフトウェアには、C 言語で書かれたこの例のソースコード (`samp4882.c`) と同じ例を BASIC で書いたソースコード (`samp4882.bas`) が入っています。

組み込む項目

- GPIB ヘッダファイル。このファイルには、アプリケーションプログラムで使う NI-488.2 ルーチンと定数のプロトタイプ (原型) が入っています。
- NI-488.2 ルーチン呼び出しの後のエラーチェック。
- GPIB エラーを処理する関数の宣言と定義。この関数はデバイスをオフラインにし、アプリケーションを閉じます。関数が以下のように宣言されると、

  ```
  void gpiberr (char *msg);  /* function prototype */
  ```

  アプリケーションは関数を次のように呼び出します。

  ```
  if (ibsta & ERR) {
      gpiberr("GPIB error");
  }
  ```

NI-488.2 プログラムシェル

図 3-2 に、NI-488.2 ルーチンを使ってアプリケーションプログラムを作成するための手順のフローチャートを示します。

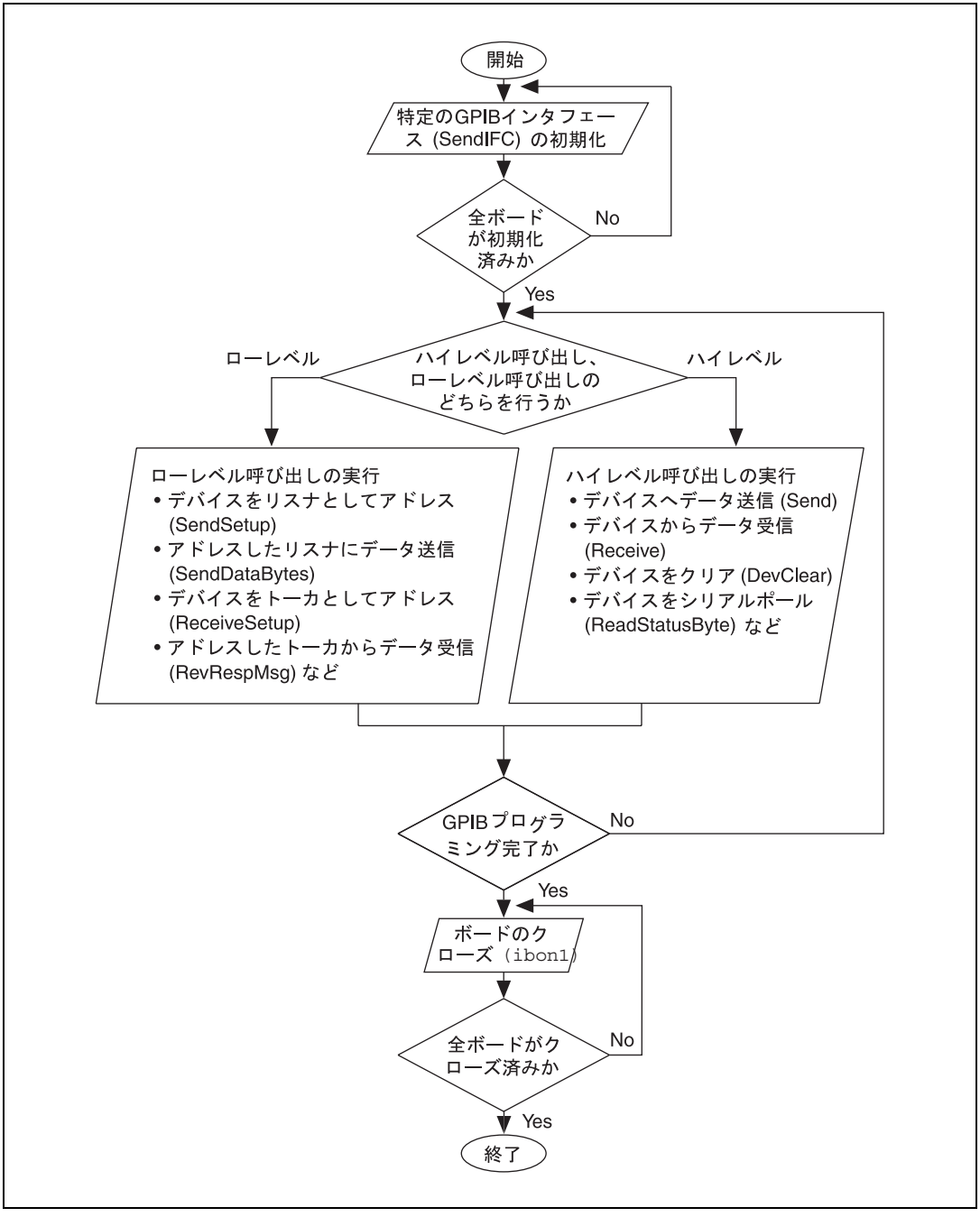


図 3-2　NI-488.2 ルーチンを使った一般的なプログラムシェル

一般的なプログラム手順とその例

以下の手順では、プログラムで NI-488.2 ルーチンをどのように使えばよいかを説明します。この例では、デジタルマルチメータを構成し、電圧値を 10 回読み取り、その測定値の平均を計算します。

ステップ 1　初期化

SendIFC ルーチンを使って GPIB バスと GPIB インタフェースボードを初期化し、GPIB ボードをコントローラインチャージ (CIC) にします。SendIFC の引数は GPIB インタフェースボード番号だけです。

```
SendIFC(0);
if (ibsta & ERR) {
    gpiberr("SendIFC error");
}
```

ステップ 2　全リスナの検出

GPIB に接続された計測器すべてが入った配列を作成するには、FindLstn ルーチンを使います。第 1 引数はインタフェースボード番号です。第 2 引数は作成した計測器リスト、第 3 引数は実際に検出できた計測器のアドレス、最終の第 4 引数は検出可能な最大デバイス数 (つまり上限に達したら中止しなければならない数) です。アドレスの最後には NOADDR 定数を付けてマークしなければなりません。この定数は、プログラムの始めに組み込んだヘッダに定義されています。

```
for (loop = 0; loop <=30; loop++){
   instruments[loop] = loop;
}
instruments[31] = NOADDR;

printf("Finding all Listeners on the bus...\n");

Findlstn(0, instruments, result, 30);
if (ibsta & ERR) {
   gpiberr("FindLstn error");
}
```

ステップ 3　計測器の識別

デバイスに ID 問い合わせを送信し、デバイスを識別します。この例では、どの計測器も IEEE 488.2 互換であり、ID 問い合わせ *IDN? を読み取ることができるものと仮定しています。さらに、FindLstn によって GPIB インタフェースボードが 1 次アドレス 0(デフォルト) で検出されたので result 配列の第 1 エントリは省略できるものとします。

```
for (loop = 1; loop <= num_Listeners; loop++) {
    Send(0, result[loop], "*IDN?", 5L, NLend);
    if  (ibsta & ERR) {
            gpiberr("Send error");
    }
    Receive(0, result[loop], buffer, 10L, STOPend);
    if  (ibsta & ERR) {
        gpiberr("Receive error");
    }
    buffer[ibcntl] = '\0';
    printf("The instrument at address %d is a %s\n",
            result[loop], buffer);
    if (strncmp(buffer, "Fluke, 45", 9) == 0) {
        fluke = result[loop];
        printf("****  Found the Fluke ****\n");
        break;
    }
}

if (loop > num_Listeners) {
    printf("Did not find the Fluke!\n");
    ibonl(0,0);
    exit(1);
}
```

定数 `NLend` は、EOI のついた改行文字が、送信するデータの最後に自動的に追加されたことを示しています。

定数 `STOPend` は EOI が検出されると読み取りが終了することを示します。

ステップ 4　計測器の初期化

マルチメータを検出できたら、DevClear ルーチンでそれをクリアします。第 1 引数は GPIB ボード番号です。第 2 引数はマルチメータの GPIB アドレスです。次に、IEEE 488.2 リセットコマンドをマルチメータに送信します。

```
DevClear(0, fluke);
if (ibsta & ERR) {
```

```
    gpiberr("DevClear error")
}

Send(0, fluke, "*RST", 4L, NLend);
if  (ibsta & ERR) {
    gpiberr("Send *RST error");
}
sum = 0.0;
for(m =0; m<10; m++){
/* start of loop for Steps 5 through 8 */
```

ステップ 5　計測器の構成

計測器は、初期化が済めばコマンドを受け取る準備ができています。マルチメータを構成するには、Send ルーチンで、そのデバイスに特有のコマンドを送信します。第 1 引数はアクセスボードの番号です。第 2 引数はマルチメータの GPIB アドレスです。第 3 引数は、マルチメータに送信するひと続きのデータです。

この例では、マルチメータに対して、自動範囲設定 (AUTO) で交流電圧 (VAC) を測定し、コントローラからトリガが来るのを待って測定を開始し (TRIGGER 2)、測定が終了してマルチメータが測定結果送信の準備ができると SRQ 線をアサートする (*SRE 16) ように指示します。第 4 引数は送信するバイト数です。最終引数である NLend はヘッダファイルで定義された定数で、Send に対して、マルチメータに送信するメッセージの最後に改行文字を追加するとともに EOI をアサートするように指示します。

```
Send (0, fluke, "VAC; AUTO; TRIGGER 2; *SRE 16", 29L,
NLend);
if (ibsta & ERR) {
    gpiberr("Send setup error");
}
```

ステップ 6　計測器のトリガ

ステップ 5 では、マルチメータはトリガを待ってから測定を開始するように指示されました。ここでトリガをマルチメータに送信します。`Trigger` ルーチンを使えばこれができますが、Fluke 45 は IEEE 488.2 互換なので直接マルチメータにトリガコマンド `*TRG` を送信することもできます。`VAL1?` コマンドはメータに、トリガされた次の読み取り値を出力バッファに送信するように指示します。

```
Send(0, fluke, "*TRG; VAL1?", 11L, NLend);
if (ibsta & ERR) {
```

```
    gpiberr("Send trigger error");
}
```

ステップ 7　測定の待機

マルチメータは、トリガされると測定値を読み取り、その値をフロントパネルに表示し、SRQ 線をアサートします。SRQ 線のアサートを検出するには、TestSRQ ルーチンまたは WaitSRQ ルーチンを使います。測定を待つ間に実行したいプロセスがある場合は、TestSRQ ルーチンを使ってください。この例では、WaitSRQ ルーチンを使います。WaitSRQ の第 1 引数は GPIB ボード番号です。第 2 引数は WaitSRQ が返すフラグで、そのフラグは SRQ 線がアサートされたかどうかを示します。

```
WaitSRQ(0, &SRQasserted);
if (!SRQasserted) {
    gpiberr("WaitSRQ error");
}
```

SRQ を検出したら、`ReadStatusByte` ルーチンでマルチメータをポールし、そのステータスを確認します。第 1 引数は GPIB ボード番号、第 2 引数は計測器の GPIB アドレス、最終引数は、`ReadStatusByte` ルーチンが計測器のステータスバイトの保存に使う変数です。

```
ReadStatusByte(0, fluke, &statusByte);
if (ibsta & ERR) {
    gpiberr("ReadStatusByte error");
}
```

ステータスバイトが分かったら、マルチメータに送信メッセージがあるかどうかを必ず確認してください。これを行うには、ステータスバイトのビット 4 のメッセージ利用可能 (MAV) ビットをチェックします。

```
if (!(statusByte & MAVbit) {
    gpiberr("Improper Status Byte");
    printf("Status Byte = 0x%x\n", statusByte);
}
```

ステップ 8　測定の読み取り

`Receive` 関数を使い、GPIB を介して測定値を読み取ります。第 1 引数は GPIB インタフェースボード番号、第 2 引数はマルチメータの GPIB アドレスです。第 3 引数は、`Receive` 関数がマルチメータから受け取ったデータバイトを入れる文字列です。第 4 引数は受信するバイト数です。最後の引数は、`Receive` メッセージが END メッセージの入ったバイトを受信することで終了することを示します。

```
Receive(0, fluke, buffer, 10L, STOPend);
if (ibsta & ERR) {
    gpiberr("Receive error");
}

buffer[ibcntl] = '\0';
printf (Reading : %s\n", buffer);
sum += AsciiToFloat(buffer);
}  /*  end of loop started in Step 5 */
```

ステップ 9　データの処理

測定値が 10 回読み取られるまで、ステップ 5 ～ 8 をループして繰り返します。それから、読み取り値の平均を次のように表示します。

```
printf (" The average of the 10 readings is :  %f\n",
sum/10);
```

ステップ 10　ボードをオフラインにする

アプリケーションプログラムを終了する前に、ibonl 関数でボードをオフラインにします。

```
ibonl(0,0);
```

C 言語のアプリケーションのコンパイル、リンク、実行

アプリケーションプログラムをコンパイルする前に、次の行がプログラムの最初に組み込まれているかどうか必ず確認してください。

```
#include "decl.h"
```

C 言語のアプリケーションプログラムを書き終えたら、Microsoft C(バージョン 5.1 以上) のコンパイラで、そのプログラムをコンパイルしてください。コンパイルした後、プログラムを C 言語インタフェース `mcib.lib` とリンクします。Microsoft C で C 言語アプリケーション `cprog` のコンパイルとリンクには、次のコマンドを使います。

```
cl cprog.c mcib.lib
```

アプリケーションを実行するには、次のコマンドを入力します。

```
cprog
```

プログラムを実行してみてエラーが見つかったら、第 4 章「アプリケーションのデバッグ」を参照してください。アプリケーション内の NI-488

や NI-488.2 の呼び出しをチェックするには、GPIB アプリケーションモニタ appmon を使います。appmon については第 6 章「appmon - GPIB アプリケーションモニタ」に説明があります。

BASIC のアプリケーションのコンパイル、リンク、実行

NI-488.2 ソフトウェアに含まれる言語インタフェースには、Microsoft Professional BASIC(バージョン 7.0 以上)、Microsoft Visual BASIC (バージョン 1.0 以上)、QuickBASIC(バージョン 4.0 以上)、BASICA、GWBASIC があります。

Microsoft BASIC

アプリケーションプログラムをコンパイルする前に、プログラムの最初に次の行が組み込まれているかどうか必ず確認してください。

```
'$include: 'mbdecl.bas'
```

BASIC のアプリケーションプログラムを書き終えたら、Microsoft Professional BASIC(バージョン 7.0 以上) のコンパイラでそのプログラムをコンパイルしてください。コンパイルが終わったら、BASIC 言語インタフェース mbib.obj とリンクします。

DOS コマンドラインから BASIC プログラム bprog のコンパイル、リンク、実行をするには、次のコードを使います。

```
bc bprog;
link bprog mbib;
bprog
```

BASIC 言語インタフェース mbib.qlb を QBX 環境用にするには、次のコマンドを使います。

```
lib mbib.lib + mbib.obj;
link /q mbib.obj, mbib.qlb,, qbxqlb.lib;
```

BASIC プログラム bprog を QBX 環境で実行するには、次のコマンドを入力します。

```
qbx bprog /l mbib.qlb
```

Run メニューから Start を選択します。

プログラムを実行してエラーが見つかったら、第 4 章「アプリケーションのデバッグ」を参照してください。アプリケーション内の NI-488 や NI-488.2 の呼び出しをチェックするには、GPIB アプリケーションモニタ

appmon を使います。appmon については第 6 章「appmon - GPIB アプリケーションモニタ」に説明があります。

Microsoft Visual BASIC

Visual BASIC で書いたプログラムのコンパイルとリンクを行う前に、次の行をモジュールレベルコードに組み込まなければなりません。

```
'$include: 'mbdecl.bas'
```

mbdecl.bas をプロジェクトの .mak ファイル内にリストしないでください。

BASIC 言語インタフェース mbib.obj を Visual BASIC 環境用にするには、次のコマンドを使います。

```
lib vbib.lib + mbib.obj + vbdos.lib;
link /q mbib.obj vbdos.lib,vbib.qlb,,vbdosqlb.lib;
```

BASIC プログラム bprog を Visual BASIC 環境で実行するには、次のコマンドを入力します。

```
vbdos bprog /l vbib.qlb
```

Run メニューから Start を選択します。

プログラムを実行してエラーが見つかったら、第 4 章「アプリケーションのデバッグ」を参照してください。アプリケーション内の NI-488 や NI-488.2 の呼び出しをチェックするには、GPIB アプリケーションモニタ appmon を使います。appmon については第 6 章「appmon - GPIB アプリケーションモニタ」に説明があります。

QuickBASIC

アプリケーションプログラムをコンパイルする前に、プログラムの最初に次の行が組み込まれているかどうか必ず確認してください。

```
'$include: 'qbdecl.bas'
```

QuickBASIC のアプリケーションプログラムを書き終えたら、Microsoft QuickBASIC(バージョン 4.0 以上) のコンパイラでそのプログラムをコンパイルしてください。コンパイルが終わったら、QuickBASIC 言語インタフェース qbib.obj とリンクします。

DOS コマンドラインから QuickBASIC のプログラム bprog のコンパイル、リンク、実行をするには、次のコードを使います。

```
bc bprog;
link bprog qbib;
bprog
```

QuickBASIC 対話環境から QuickBASIC のプログラムのコンパイル、リンク、実行をするには、次のコードを使います。

```
link /q qbib.obj, qbib.qlb,, bqlb45.lib
qb bprog /l qbib.qlb
```

QuickBASIC 4.0 コンパイラでは、bqlb45.lib ではなく bqlb40.lib を使います。

プログラムを実行してエラーが見つかったら、第 4 章「アプリケーションのデバッグ」を参照してください。アプリケーション内の NI-488 や NI-488.2 の呼び出しをチェックするには、GPIB アプリケーションモニタ appmon を使います。appmon については第 6 章「appmon - GPIB アプリケーションモニタ」に説明があります。

BASICA/GWBASIC

BASICA 言語インタフェース bib.m は、必ずカレントディレクトリに置いてください。

BASICA のプログラム bprog のコンパイル、リンク、実行をするには、次のコードを使います。

```
load "bprog.bas
merge "decl.bas
run
```

第 1 行の decl.bas の中の clear 文には、BASICA 言語インタフェース bib.m をロードした状態で使用可能なメモリ量を示す定数が入っています。ほとんどの場合は、この定数はそのままで使用できます。ただし、BASICA を起動したときに使用可能領域が 59KB より少ないという表示が出たら、定数の値を小さくしてください。

プログラムを実行してエラーが見つかったら、第 4 章「アプリケーションのデバッグ」を参照してください。アプリケーション内の NI-488 や NI-488.2 の呼び出しをチェックするには、GPIB アプリケーションモニタ appmon を使います。appmon については第 6 章「appmon - GPIB アプリケーションモニタ」に説明があります。

4

アプリケーションのデバッグ

本章では、アプリケーションプログラムのデバッグ方法をいくつか説明します。

ibtest の実行

アプリケーションプログラムを実行する前に、NI-488.2 ソフトウェアに付属のソフトウェア診断テスト ibtest を実行してください。ibtest プログラムは NI-488.2 アプリケーションで、ドライバを呼び出すことができます。ibtest の結果が正常なら、GPIB ハードウェアと NI-488.2 ソフトウェアは互いに正しく動作しています。以下に ibtest の実行中に表示されるメッセージについて説明し、問題点への対処も合わせて述べます。gpib*x* とは、ボード gpib0, gpib1, gpib2, gpib3 のいずれかを意味します。

ドライバの存在確認テスト

ibtest プログラムは NI-488.2 ドライバが存在するかどうかテストし、問題があれば以下のメッセージを表示します。

<<< No driver present for GPIBx. >>>

このメッセージが表示されたら、GPIB ドライバがインストールされているかどうか確認します。また、次の行が config.sys ファイルに入っているかどうかチェックします。

```
device = drive:\path\gpib.com
```

ここで *drive* は NI-488.2 ソフトウェアをインストールしたドライブ名 (通常 c ドライブ)、path はそのドライブ内の NI-488.2 ソフトウェアまでのパス (例 : at-gpib) です。

ボードの存在確認テスト

gpib*x* がインストールされていなかったり、NI-488.2 ソフトウェアの構成が間違っていると、以下のエラーメッセージが表示されます。

<<< No board present for GPIBx. >>>

このメッセージが表示されたら、以下のいずれかの状態になっています。

- ibconf の Use this GPIB Interface(この GPIB インタフェースを使用) フィールドでボード GPIB*x* の設定が no になっています。ボードを使用するときは、このフィールドを yes に設定しなければなりません。
- ボードのインストールまたは構成が間違っています。詳細については「入門マニュアル」を参照してください。
- ソフトウェアとハードウェアの設定に不一致があります。ibconf を実行すればソフトウェアの現在の構成が確認できます。

GPIB ケーブル接続

ibtest の実行時に GPIB ケーブルがボードに接続されたままになっていると、次のようなエラーメッセージが表示されます。

```
Call(25) 'ibcmd " "' failed, ibsta (0x134) not what was
expected (0x8130)

Call(25) 'ibcmd " "' failed, expected ibsta (0x100) to have
the ERR bit set.
```

全 GPIB ケーブルを外してから、ibtest を再実行してください。

ULI ドライバロード済み

ULI ドライバ uli.com をロードしたまま、NI-488 関数や NI-488.2 ルーチンを使ったり ibtest を実行すると、次のエラーメッセージが表示され、場合によってはコンピュータがハングアップすることもあります。

```
Syntax Error
```

ULI ドライバがロードされていると、標準 NI-488 関数や NI-488.2 ルーチンは使えません。それらを使用するには、コンピュータを再起動し、ULI がロードされていない状態にしてください。autoexec.bat ファイルで uli.com をロードしているのであれば、uli.com をロードする行を注釈行に変更してからコンピュータを再起動します。

GPIBInfo の実行

GPIBInfo ユーティリティプログラムは簡単な診断ツールで、お使いの NI-488.2 ソフトウェアや、システムの GPIB インタフェースボードについての情報を得ることができます。この情報を見れば、NI-488.2 ソフトウェアの機能が分かり、問題が発生してナショナルインスツルメンツのテクニカルサポートへ問合せる場合にも便利です。

GPIBInfo をパラメータなしで実行すると、GPIBInfo は、お使いの GPIB ソフトウェアの名前とバージョン、GPIB インタフェースボードのタイプ、GPIB ソフトウェアで使用可能な関数、HS488 高速プロトコル使用の可否などを表示します。また GPIBInfo は、システムにインストールしてある GPIB インタフェースボードについて、ボード名、コントローラチップ、ハードウェア設定、使用可能な機能の種類、HS488 高速通信プロトコル使用の可否も表示します。GPIBInfo 出力の一般的な例を次に示します。

```
GPIBInfo (Sep 29 1993)
Copyright (c) 1993 National Instruments Corp.  All rights
reserved.

Software Information:
    The NI-488.2 Software for MS-DOS is loaded.
    You are running Version 2.5 for the AT-GPIB/TNT board.
    It supports both the NI-488 functions and the NI-488.2
    routines.
    It supports the HS488 high-speed protocol.

Hardware Information:
  GPIB0: an AT-GPIB/TNT board using the TNT4882C chip.
        It supports both the NI-488 functions and NI-488.2
        routines.
        It supports the HS488 high-speed protocol.
        It uses base I/O address 0x2C0.
        It uses interrupt level 11.
        It uses DMA channel 5.
```

GPIBInfo は、パラメータを 1 つ付けて実行することもできます。パラメータは、システムの GPIB ボードのベース I/O アドレスです。GPIBInfo は、指定のアドレスにボードを検出すると、そのボードについての情報を表示します。GPIBInfo のこの機能は、ある特定のアドレスに GPIB ボードをインストールしてあるかどうかを調べるのに便利です。ボードをベース I/O アドレス 2C0(16 進) にインストールしてあるのであれば、gpibinfo 0x2c0 と入力すれば次のような出力が得られます。

```
GPIBInfo (Sep 29 1993)
Copyright (c) 1993 National Instruments Corp.  All rights
reserved.

The board at base I/O address 0x2C0 appears to be an
AT-GPIB/TNT.
```

```
It uses the TNT4882C GPIB Controller chip.
It supports both the NI-488 functions and the NI-488.2
routines.
It supports the HS488 high-speed protocol.
The NI-488.2 software is configured to access this board
as GPIB0.
```

グローバルステータス変数を使ったデバッグ

NI-488.2 ドライバへの関数呼び出しがあると、関数呼び出しは ibsta, iberr, ibcnt, ibcntl を更新してからアプリケーションに戻ります。GPIB 呼び出しの後にはエラーがないか必ず確認してください。プログラムでエラーの自動確認を行う際のグローバル変数の使い方については、第 3 章「アプリケーションの開発」を参照してください。

どの GPIB 呼び出しに障害があるかと、グローバル変数の対応値も分かったら、付録 A「ステータスワード状況」、付録 B「エラーコードと対処」を参照してください。ドライバの状態が理解できます。

ibic を使ったデバッグ

エラーの検出と表示を自動的に行わないアプリケーションをお使いの場合は、ibic を使えばエラーが検出できます。同じ関数またはルーチンを、アプリケーションプログラムにあるのと同じように 1 つずつ実行するだけです。ibic はステータス値とエラーコードを返すので、どの GPIB 呼び出しに障害があるかが分かります。ibic の詳細については、第 5 章「ibic - インタフェースバス対話式制御ユーティリティ」を参照してください。

どの GPIB 呼び出しに障害があるかと、グローバル変数の対応値が分かったら、付録 A「ステータスワード状況」、付録 B「エラーコードと対処」を参照してください。ドライバの状態が理解できます。

appmon を使ったデバッグ

NI-488.2 ソフトウェアには、デバッグツールとして便利な appmon というアプリケーションモニタユーティリティが入っています。appmon では、DOS アプリケーションからの NI-488 や NI-488.2 の呼び出しを監視することができます。ibsta にエラービットが設定されるとアプリケーションの実行を中断してエラー情報画面を表示するように appmon を構成することができます。appmon の詳細については、第 6 章「appmon - GPIB アプリケーションモニタ」を参照してください。

GPIB エラーコード

GPIB エラーコードを表 4-1 に示します。エラー変数に意味があるのはステータス変数の ERR ビットが設定されているときだけだということに注意してください。各エラーの詳しい説明と対処については、付録 A「エラーコードと対処」を参照してください。

表 4-1　GPIB エラーコード

エラーニモニック	iberr 値	意味
EDVR	0	DOS エラー
ECIC	1	この関数では GPIB ボードが CIC でなければなりません
ENOL	2	GPIB 上にリスナがありません
EADR	3	GPIB ボードのアドレスが間違っています
EARG	4	関数呼び出しの引数が無効です
ESAC	5	GPIB ボードがシステムコントローラであることが必要です
EABO	6	入出力処理が打ち切られました (タイムアウト)
ENEB	7	GPIB ボードがありません
EOIP	10	非同期入出力実行中
ECAP	11	資格がありません
EFSO	12	ファイルシステムエラー
EBUS	14	GPIB バスエラー
ESTB	15	シリアルポールのステータスバイト待ち行列オーバフロー
ESRQ	16	SRQ がオフになりません
ETAB	20	表の異常

構成エラー

ハードウェアとソフトウェアの設定が一致していないと、以下のどちらかの問題点が発生することがあります。

- 入出力関数でアプリケーションがハングアップする。
- データが破壊される。

このような問題が発生したら、割り込み要求レベルと DMA チャネルについて、GPIB ハードウェアの設定が NI-488.2 ソフトウェアの設定と一致し

ているかどうか確認してください。ハードウェアとソフトウェアのデフォルト設定については、製品に付属する「入門マニュアル」を参照してください。NI-488.2 ソフトウェアの構成を表示したり変更する方法については、第 8 章「ibconf - インタフェースバス構成ユーティリティ」を参照してください。

ハードウェアをテストするには、ibdiag プログラムを使います。ibdiag については、「入門マニュアル」を参照してください。

アプリケーションによっては、GPIB ドライバの構成を変更しなければならないことがあります。たとえば、ある特定の EOS(文字列の終わり) 文字で読み取りを終了する時や、2 次アドレス指定が必要な時などです。このような場合は、ibconf ユーティリティでドライバを完全に再構成するか、NI-488 の ibconfig 関数を使ってアプリケーションの動作中にプログラムでドライバを変更することができます。

メモ　ベース I/0 アドレス、割り込みレベル、DMA チャネル以外の設定を変更するときは、ibconf ユーティリティを実行せずに、ibconfig 関数を使うことをお勧めします。

アプリケーションで ibconfig を使えば、アプリケーションはドライバのその時点での構成に関係なく、いつでも動作します。詳細については、「DOS/Windows 用 NI-488 関数リファレンスマニュアル」の説明を参照してください。

タイミングエラー

アプリケーションに障害が発生しているにも関わらず ibic で同じ呼び出しが成功するようであれば、プログラムの NI-488.2 呼び出しが速すぎて、デバイスの処理と応答ができないことが考えられます。このような障害はデータが壊れたり、不完全だった場合にも発生します。

正常な IEEE 488 デバイスはハンドシェイキングを延期して適切な転送速度を設定します。そうでない場合、プログラムをシングルステップ実行して、それぞれの GPIB 呼び出しの間にある程度の長さのある遅延を挿入すれば、タイミングエラーの検出と修復ができます。また、方法の 1 つとして、できるだけ頻繁にデバイスにそのステータスを報告させることがあります。これが可能なデバイスは多くはありませんが、普通はこれが最善の方法です。遅延はデバイスに制御されますが、アプリケーション自身が調整を行い、どんなプラットフォームでも動作します。これ以外の遅延機構では、プラットフォームによって遅延時間が変化してしまう可能性が高くなります。

通信エラー

アドレス指定の繰り返し

デバイスによっては、GPIB 動作を実行する前に GPIB アドレス指定が必要なものがあります。IEEE 488.2 に準拠したデバイスは、状態を変更させるコマンドが GPIB を通じて送信されない限り現在の状態のまま変わりません。デバイスがアドレス指定したときの状態に留まらないのであれば、アドレス指定を繰り返すように NI-488.2 ドライバを構成しなければなりません。NI-488.2 ソフトウェアの構成変更の詳細については、第 8 章「ibconf - インタフェースバス構成ユーティリティ」または「DOS/Windows 用 NI-488 関数リファレンスマニュアル」の `ibconfig`(オプション `IbcREADDR`) の説明を参照してください。

終了方法

ユーザはデバイスが使っているデータ終了方法を知っておいてください。NI-488.2 ソフトウェアはデフォルトでは、書き込み時には EOI を送信し、読み取りは EOI を検出するか指定のバイトカウント数に達したときに終了するよう構成されています。デバイスにコマンド文字列を送信しても応答がないときは、そのデバイスがコマンドの終わりを認識できていないのかも知れません。書き込みの後で、次に示すように <CR><LF> などの終了メッセージを送信する必要があるのかもしれません。

```
ibwrt(dev, "COMMAND\x0A\x0D",9);
```

Q&A

Q:DOS 用と Windows 用のドライバを両方インストールしても構いませんか？

A: はい、構いません。両方ともインストールしても大丈夫です。ただし、両方に同時にアクセスすることは避けてください。

Q: エラーが発生したときに `ibdiag` や `ibtest` を実行しても原因が分からないときはどうすればよいのですか？

A: テストが失敗した原因については、本章の「ibtest の実行」と「入門マニュアル」の `ibdiag` に関する項目を参照してください。

Q: どうすれば GPIB を使って計測器と通信できるのですか？

A: 計測器に付属する取扱説明書を参照してください。どのようなコマンドシーケンスを使うべきかは、それぞれの計測器によって全く異なりま

す。その取扱説明書に、計測器との通信に必要な GPIB コマンドが記載されています。ほとんどの場合には、NI-488 デバイスレベル呼び出しを使えば計測器との通信ができます。詳細については、第 3 章「アプリケーションの開発」を参照してください。

Q:1 つのアプリケーションの中で NI-488 と NI-488.2 の呼び出しを両方使っても構いませんか？

A: はい、構いません。NI-488 関数と NI-488.2 ルーチンは一緒に使うことができます。

Q: アプリケーションの名前と、`ibconf` にリストされる GPIB ボード名またはデバイス名が同じでも構いませんか？

A: いいえ、それは問題があります。DOS デバイスの使うネーム空間は、ファイルやディレクトリと同じです。ファイルまたはディレクトリの名前が GPIB ボードまたはデバイスの名前と同じだと、DOS は正しく動作しません。ファイル拡張子の部分を変えても、この問題には何の効果もありません。つまりファイル名が `gpib0` と `gpib0.txt` であっても、問題が起きてしまいます。DOS ドライバが使う名前は、デフォルトでは、`gpib0, gpib1, gpib2, gpib3, dev1, dev2, dev3.....dev32` です。

Q:GPIB アプリケーションでエラーをチェックするにはどうすればいいのでしょうか？

A:NI-488 や NI-488.2 の呼び出しの後に必ず `ibsta` の値をチェックしてください。呼び出しが失敗すれば、`ibsta` の ERR ビットが設定され、エラーコードが iberr に格納されます。グローバルステータス変数の詳細については、第 3 章「アプリケーションの開発」を参照してください。

Q:`ibic` はどうやって使うのですか？

A:`ibic` を使えば、計測器との通信や、トラブルシュート、アプリケーションプログラムの作成などができます。詳細については、第 5 章「ibic - インタフェースバス対話式制御ユーティリティ」を参照してください。

Q: インストールしてある GPIB ボードがどんなタイプなのか、どうすれば分かるのですか？

A:`GPIBInfo` ユーティリティを実行してください。パラメータを指定せずに `GPIBInfo` を実行すれば、現在、そのシステムで使われている GPIB ボードに関する情報が表示されます。GPIB インタフェースボードのベース I/0 アドレスが分かっていれば、それをパラメータとして入力して、そのボードについての詳しい説明が得られます。たとえば `gpibinfo` 2c0 と入力すれば、ベース I/0 アドレスが 2C0 の GPIB ボードに関する情報が表示されます。

Q: インストールしてある NI-488.2 ソフトウェアのバージョン番号はどうすれば分かるのですか？

A:GPIBInfo ユーティリティを実行してください。パラメータを指定せずに実行すれば、現在インストールされている NI-488.2 ソフトウェアのバージョンが表示されます。

Q: 問題が発生してナショナルインスツルメンツに電話するときには、何が分かっていれば解決が早いのでしょうか？

A: ナショナルインスツルメンツに電話するときには、診断テスト ibdiag および ibtest の結果と、GPIBInfo ユーティリティの出力をご用意ください。また、付録 D「カスタマーコミュニケーション」にあるテクニカルサポートフォームの各項目をご記入になっておいてください。

5

ibic - インタフェースバス対話式制御ユーティリティ

本章では、ibic(インタフェースバス対話式制御ユーティリティ) について基本的な説明をします。ibic では、GPIB デバイスと対話式に通信を行うことができます。

概要

ibic プログラムでは、キーボードから関数を入力して GPIB デバイスと通信を行うことができます。ある特定のデバイスとの通信方法については、そのデバイスの取扱説明書を参照してください。ibic を使えば、計測器との通信、トラブルシュート、アプリケーションプログラム作成などの練習ができます。

ibic は計測器についての情報を知らせ、トラブルシュートを助ける方法の 1 つとして、コマンドを入力すれば以下のような情報を画面に表示します。

- 16 進表記によるステータスワード (ibsta) の内容。
- ibsta の各ビットの設定に対するニモニック定数。
- エラーが発生した (ibsta の ERR ビットが設定された) 場合には、エラー変数 (iberr) のニモニック値。
- 読み取り関数、書き込み関数、コマンド関数のカウント値。
- 計測器から受信したデータ。

NI-488 関数を使用した例

ここでは、ibic を使って NI-488 デバイス関数呼び出しのシーケンスをテストする方法について説明します。関数のパラメータを覚える必要はありません。関数名だけを入力すれば、ibic が必要なパラメータを要求してきます。

1. ibic を実行するため、NI-488.2 ソフトウェアをインストールしてあるディレクトリ (この場合は c:\at-gpib) に移動します。次にコマンド ibic を入力します。スクリーンに以下の表示が現れます。

```
C:\AT-GPIB> ibic

National Instruments
IEEE-488 Interface Bus Interactive Control Program
(IBIC)
Copyright (c) 1993 National Instruments Corp. Version
3.0 (DOS)
Version Date: May 28 1993 Version Time: 09:42:25
All rights reserved

Type 'help' for help or 'q' to quit

:
```

2. 次に示した例では、ibdev を使ってデバイスをオープンし、それをアクセスボード gpib0 に割り当て、さらに 1 次アドレス 6(2 次アドレスなし) を選んでいます。また、タイムアウトは 10ms に設定し、END メッセージをイネーブルし、EOS モードはディスエーブルしています。

```
:ibdev
    enter board index: 0
    enter primary address: 6
    enter secondary address: 0
    enter timeout: 13
    enter 'EOI on last byte' flag: 1
    enter end-of-string mode/byte: 0
id = 32256
ud0:
```

ibdev コマンドを使えば、同じ情報を次のように入力できます。

```
:ibdev 0 6 0 13 1 0
id = 32256
ud0:
```

3. 次のようにデバイスをクリアします。

```
ud0: ibclr
[0100] (cmpl)
```

4. ファンクション、レンジ、トリガソースの命令をデバイスに書き込みます。計測器で使用できるコマンドバイトについては、計測器の取扱説明書を参照してください。

```
ud0: ibwrt
    enter string: "*RST; VAC; AUTO; TRIGGER 2; *SRE 16"
[0100] (cmpl)
count: 35
```

あるいは

```
ud0: ibwrt "*RST; VAC; AUTO; TRIGGER 2; *SRE 16"
[0100] (cmpl)
count: 35
```

5. 次のようにデバイスをトリガします。

```
ud0: ibtrg
[0100] (cmpl)
```

6. タイムアウトが発生するか、デバイスがサービスを要求するまで待機します。現在のタイムアウト制限時間が短すぎるようであれば、ibtmo で調整してください。ibwait コマンドは次のように使います。

```
ud0: ibwait
    enter wait mask: TIMO RQS
[0900] (rqs cmpl)
```

あるいは

```
ud0: ibwait TIMO RQS
[0900] (rqs cmpl)
```

7. シリアルポールステータスバイトを読み取ります。シリアルポールステータスバイトは、使用するデバイスによって異なります。

```
ud0: ibrsp
[0100] (cmpl)
Poll: 0x40 (decimal : 64)
```

8. 読み取りコマンドを使って、16 進値とそれに対応する ASCII 文字を画面に表示します。

```
ud0: ibrd
    enter byte count: 18
```

```
[0100] (cmpl)
count: 18
4e 44 43 56 20 30 30 30     N D C V   0 0 0
2e 30 30 34 37 45 2b 30     . 0 0 4 7 E + 0
0a 0a                       . .
```

あるいは

```
ud0: ibrd 18
[0100] (cmpl)
count: 18
4e 44 43 56 20 30 30 30     N D C V   0 0 0
2e 30 30 34 37 45 2b 30     . 0 0 4 7 E + 0
0a 0a                       . .
```

9. 次のようにしてデバイスをオフラインにします。

```
ud0: ibonl
   enter value: 0
[0100] (cmpl)
```

あるいは

```
ud0: ibonl 0
[0100] (cmpl)
```

10. プロンプトが表示された状態で q と入力し、ibic プログラムを終了します。

ibic 構文

ibic でコマンドを入力するときは、パラメータを同時に入力することも、パラメータなしで入力してプログラムに値を要求させることもできます。コマンドによって、入力値が数値であったり文字列であったりします。

数値の構文

数値には 16 進表記、8 進表記、10 進表記の整数が入力できます。

16 進数 - 16 進値の前にゼロと x を置きます。例 : 0xD

8 進数 - 8 進値の前にゼロだけを置きます。例 : 015

10 進数 - 10 進数の数字だけを入力します。

文字列の構文

文字列は、ASCII 文字シーケンス、8 進バイト、16 進バイト、特殊記号、の形で入力することができます。

ASCII 文字シーケンス - シーケンス全体を引用符でくくります (例 : "*tst")。文字列の中に引用符を入れたいときは、引用符の前にバックスラッシュをつけます (例 : "ab\"cd")。

8 進バイト - 8 進値は、前にバックスラッシュを置きます。たとえば 8 進 "40" は、\40 と示され、文字列の中では "ab\40cd" のように使います。

16 進バイト - 16 進値は、前にバックスラッシュと x を置きます。たとえば 16 進 "40" は、\x40 と示され、文字列の中では "ab\x40cd" のように使います。

特殊記号 - 計測器によっては、デバイスに対して転送が終了したことを示す特殊な終了方法として、文字列の終わり (EOS) 文字が必要です。一般的な EOS 文字は、\r と \n です。\r は復帰文字、\n は改行文字を表します。これらの特殊文字を使えば、"F3R5T1\r\n" のように文字列の中に復帰文字と改行文字を入れることができます。

アドレスの構文

NI-488.2 ルーチンの多くは、アドレスまたはアドレスリストパラメータを使います。アドレスとは、デバイスの GPIB アドレスを示す 16 ビット表現です。1 次アドレスは下位バイトに格納され、2 次アドレスがあれば上位バイトに格納されます。たとえば、1 次アドレスが 6 で 2 次アドレスが 0x67 のデバイスの場合、アドレスは 0x6706 になります。NULL アドレスは、0xffff と表されます。

NI-488 関数の ibic 構文

表 5-1 と表 5-2 は、ibic における NI-488 関数の構文を示しています。v は入力する数値を、string は同じく文字列を表します。関数パラメータの詳細については ibic のヘルプ機能を使うか、「DOS/Windows 用 NI-488.2 関数リファレンスマニュアル」を参照してください。

表 5-1　ibic におけるデバイスレベル NI-488 関数の構文

構文	説明
ibask mn	構成の情報を返します。mn は構成のパラメータを表すニモニックまたはそれに対応する整数値です。
ibbna brdname	デバイスのアクセスボードを変更します。brdname は新たなボードのシンボリック名です。
ibclr	指定したデバイスをクリアします。
ibconfig mn v	構成のパラメータを変更します。mn は構成のパラメータを表すニモニックまたはそれに対応する整数値です。
ibdev v v v v v v	使用されていないデバイスをオープンします。ibdev パラメータは board id, pad, sad, tmo, eos, eot です。
ibeos v	EOS メッセージを変更 / ディスエーブルします。
ibeot v	END メッセージをイネーブル / ディスエーブルします。
iblines	全ての GPIB 管理線の状態を読み取ります。
ibln v v	GPIB 上のアドレス pad, sad にデバイスが存在するかどうか確認します。
ibloc	ローカルにします。
ibonl v	デバイスをオンラインまたはオフラインにします。
ibpad v	1 次アドレスを変更します。
ibpct	制御を渡します。
ibppc v	パラレルポール構成。
ibrd v	データを読み取ります。v は読み取るバイト数です。
ibrda v	データを非同期に読み取ります。v は読み取るバイト数です。
ibrdf flname	データをファイルに読み取ります。flname はデータを入れるファイルのパス名です。
ibrpp	パラレルポールを実行します。
ibrsp	シリアルポールバイトを返します。
ibsad v	2 次アドレスを変更します。
ibstop	非同期処理を打ち切ります。
ibtmo v	制限時間を変更 / ディスエーブルします。
ibtrg	選択したデバイスをトリガします。

表 5-1　ibic におけるデバイスレベル NI-488 関数の構文（続き）

構文	説明
ibwait mask	選択したイベントが発生するのを待ちます。mask は 16 進、8 進、10 進の整数、またはマスクビットのニモニックです。
ibwrt string	データを書き込みます。
ibwrta string	データを非同期的に書き込みます。
ibwrtf flname	ファイルからデータを書き込みます。flname はデータの入っているファイルのパス名です。

表 5-2　ibic におけるボードレベル NI-488 関数の構文

構文	説明
ibask mn	構成の情報を返します。mn は構成のパラメータを表すニモニックまたはそれに対応する整数値です。
ibcac v	アクティブコントローラにします。
ibcmd string	コマンドを送信します。
ibcmda string	コマンドを非同期的に送信します。
ibconfig mn v	構成パラメータを変更します。mn は構成のパラメータを表すニモニックまたはそれに対応する整数値です。
ibdma v	DMA をイネーブル / ディスエーブルします。
ibeos v	EOS メッセージを変更 / ディスエーブルします。
ibeot v	END メッセージをイネーブル / ディスエーブルします。
ibevent	記録されている一番古いイベントを返します。
ibfind udname	ユニット記述子を返します。udname はボードのシンボリック名 (例 : gpib0) です。
ibgts v	アクティブコントローラをスタンバイ状態にします。
ibist v	ist を設定 / クリアします。
iblines	全ての GPIB 管理線の状態を読み取ります。
ibln v v	GPIB 上のアドレス pad，sad にデバイスが存在するかどうか確認します。
ibloc	ローカルにします。
ibonl v	デバイスをオンラインまたはオフラインにします。

表 5-2　ibic におけるボードレベル NI-488 関数の構文

構文	説明
ibpad v	1 次アドレスを変更します。
ibppc v	パラレルポール構成。
ibrd v	データを読み取ります。v は読み取るバイト数です。
ibrda v	データを非同期的に読み取ります。v は読み取るバイト数です。
ibrdf flname	データをファイルに読み取ります。flname はデータを入れるファイルのパス名です。
ibrpp	パラレルポールを実行します。
ibrsc v	システム制御の要求 / リリースを行います。
ibrsv v	サービスを要求します。
ibsad v	2 次アドレスを変更します。
ibsic	インタフェースクリアを送信します。
ibsre v	リモートイネーブル線を設定 / クリアします。
ibstop	非同期処理を打ち切ります。
ibtmo v	制限時間を変更 / ディスエーブルします。
ibtrap v v	appmon の構成を変更します。
ibwait mask	選択したイベントが発生するのを待ちます。mask は 16 進、8 進、10 進の整数、またはマスクビットのニモニックです。
ibwrt string	データを書き込みます。
ibwrta string	データを非同期的に書き込みます。
ibwrtf flname	ファイルからデータを書き込みます。flname はデータの入っているファイルのパス名です。

NI-488.2 ルーチンの ibic 構文

以下の表 5-3 は ibic における NI-488.2 ルーチンの構文をまとめたものです。v は数値を、string は文字列を表します。address はアドレス、addrlist はカンマで区切ったアドレスリストを表します。NI-488.2 ルーチンのパラメータの詳細については、ibic のヘルプ機能を使うか、「DOS/Windows 用 NI-488.2 関数リファレンスマニュアル」を参照してください。

表 5-3　ibic における NI-488.2 ルーチンの構文

ルーチン構文	説明
AllSpoll addrlist	複数のデバイスとのシリアルポールを行います。
DevClear address	デバイスをクリアします。
DevClearList addrlist	複数のデバイスをクリアします。
EnableLocal addrlist	ローカル制御をイネーブルします。
EnableRemote addrlist	リモート制御をイネーブルします。
FindLstn addrlist v	全リスナを検出します。
FindRQS addrlist	SRQ をアサートしているデバイスを検出します。
PassControl address	制御をデバイスに渡します
PPoll	デバイスのパラレルポールを行います。
PPollConfig address v v	デバイスをパラレルポール用に構成します。
PPollUnconfig address	デバイスのパラレルポール構成を解除します。
RcvRespMsg address string v	応答メッセージを受信します。
ReadStatusByte address	デバイスのシリアルポールを行います。
Receive address string v	デバイスからデータを受信します。
ReceiveSetup address	セットアップを受信します。
ResetSys addrlist	複数のデバイスをリセットします。
Send address string v	デバイスにデータを送信します。
SendCmds string	コマンドバイトを送信します。
SendDataBytes addrlist string v	データバイトを送信します。
SendIFC	インタフェースクリアを送信します。
SendList addrlist string v	複数のデバイスにデータを送信します。
SendLLO	デバイスをローカルロックアウト状態にします。
SendSetup addrlist	セットアップを送信します。
Set 488.2 v	ボード v を 488.2 モードにします。
SetRWLS addrlist	リモートのデバイスをロックアウト状態にします。
TestSys addrlist	複数のデバイスに自己テストを実施させます。
TestSRQ	サービス要求が出ているかどうかテストします。

表 5-3　ibic における NI-488.2 ルーチンの構文（続き）

ルーチン構文	説明
`Trigger address`	デバイスにトリガを送信します。
`TriggerList addrlist`	複数のデバイスにトリガを送信します。
`WaitSRQ`	サービス要求を待ちます。

ステータスワード

`ibic` では、全ての NI-488 関数 (`ibfind` と `ibdev` を除く) と NI-488.2 ルーチンはステータスワードを返し、それらは角括弧内の 16 進値と括弧内のニモニックのリストという 2 つの形式で表示されます。次の例では、ステータスワードは第 2 行にあり、デバイス関数の書き込み動作が正常に終了したことを示しています。

```
ud0: ibwrt "f2t3x"
[0100] (cmpl)
count: 5

ud0:
```

ステータスワードの詳細については、第 3 章「アプリケーションの開発」を参照してください。

エラー情報

NI-488 関数や NI-488.2 ルーチンの終了時にエラーが発生していると、`ibic` は該当するエラーニモニックを表示します。次の例では、データ転送中にエラー状況 EBUS が発生しました。

```
ud0: ibwrt "f2t3x"
[8100] (err cmpl)
error: EBUS
count: 1

ud0:
```

この例ではアドレス指定のコマンドバイトがデバイスに転送されませんでした。これは dev1 の電源がオフであったか、GPIB ケーブルが接続されていなかったことを表しています。

エラーコードとその意味の詳細については、第 4 章「アプリケーションのデバッグ」を参照してください。

カウント

入出力関数が完了すると、ibic はエラーの有無に関係なく、送信または受信が済んだバイト数を表示します。

NI-488.2 ルーチンのアドレスリストに無効なアドレスが入っていると、エラー EARG が発生し、ibic は無効アドレスの指標をカウントで表示します。

カウントは、どの NI-488 関数または NI-488.2 ルーチンを呼び出したかによって、意味が異なります。カウント戻り値の正しい意味については、「DOS/Windows 用 NI-488.2 関数リファレンスマニュアル」を参照してください。

一般的な NI-488 関数

ibfind

ボードをオープンするには ibfind 関数を使います。次の例では、gpib0 をオープンしています。

```
:ibfind gpib0
id = 32000

gpib0:
```

id はボードのユニット記述子です。プロンプトの gpib0: はボードがオープンされていることを表します。

ibfind 関数に使う名前は、ドライバにある有効なシンボリック名でなければなりません。有効な名前の詳細については、第 8 章「ibconf - インタフェースバス制御ユーティリティ」を参照してください。

ibdev

ibdev コマンドは、入力した情報を使ってデバイス記述子を初期化します。

ibdev では、次の値を指定してください。

- デバイスのアクセスボード
- 1 次アドレス
- 2 次アドレス
- タイムアウト設定

- EOT モード
- EOS モード

次の例では、ibdev が有効なデバイスをオープンし、それが 1 次アドレス 6(pad=6)、2 次アドレス 16 進 67(sad=0x67) にある gpib0(board=0) にアクセスするように割当てています。また、タイムアウトは 10 秒 (tmo=13)、END メッセージはイネーブル (eot=1)、EOS モードはディスエーブル (eos=0) という設定をしています。

```
:ibdev 0 6 0x67 13 1 0
id = 32256

ud0:
```

パラメータを指定せずに ibdev を使うと、次の例に示すように ibic が入力パラメータを要求します。

```
:ibdev
    enter board index: 0
    enter primary address: 6
    enter secondary address: 0x67
    enter timeout: 13
    enter 'EOI on last byte' flag: 1
    enter end-of-string mode/byte: 0
id = 32256

ud0:
```

ibdev 呼び出しにおいては、特に次の 3 種類のエラーが発生することがあります。

- EDVR - デバイスが利用できません。入力されたボード指標にボードがない (つまりボード指標が 0, 1, 2, 3 以外である) か、ドライバがインストールされていません。次に EDVR エラーの例を示します。

  ```
  :ibdev 4 6 0x67 7 1 0
  id = -1
  [8000] (err)
  error: EDVR (2)
  :
  ```

- ENEB - 入力されたボード指標は存在するボードのもの (例 : 0) ですが、ドライバがボードを検出できません。この場合、ibconf を実行して、ボードのベースアドレスが正しく設定してあるかどうかと、

Use This GPIB Interface(この GPIB インタフェースを使用) フィールドが yes に設定してあるかどうかを確認してください。

- EARG - 最後の 5 つのパラメータのどれかが無効な値です。ibdev 呼び出しを実行してもまたプロンプトが出て、EARG エラー (無効な関数引数) が表示されます。ibdev 呼び出しの結果 EARG エラーが出たら、どのパラメータが間違っているかを調べ、適切なコマンドを使ってその間違いを正さなければなりません。次の例では、pad に無効な値が入っています。ibpad 呼び出しを使って訂正します。

```
:ibdev 0 66 0x67 7 1 0
id = 32256
[8100] (err cmpl)
error: EARG

ud0: ibpad 6
previous value: 16
```

ibwrt

ibwrt コマンドはデータを GPIB デバイスから別の GPIB デバイスに送信します。たとえば 6 文字のデータ文字列 F3R5T1 をコンピュータからデバイスに送信するには、プロンプトに続けて次の例に示すように入力します。

```
ud0: ibwrt "F3R5T1"
[0100] (cmpl)
count: 6
```

戻ってきたステータスワードには cmpl ビットが入っています。cmpl ビットは、入出力動作が正常に終了したことを示します。バイトカウント 6 というのは、6 文字が全てコンピュータから送信され、デバイスがそれを受信したことを示します。

ibrd

ibrd コマンドは GPIB デバイスに他の GPIB デバイスからのデータを受信させます。次の例では、GPIB デバイスからデータを受け取り、それを画面に 16 進値とそれに対応する ASCII 文字で表示します。ステータスワードとバイトカウントも一緒に表示されます。

```
ud0: ibrd 20
[2100] (end cmpl)
count: 18
4e 44 43 56 28 30 30 30    N D C V 9 0 0 0
```

```
2e 30 30 34 37 45 2b 30    . 0 0 4 7 E + 0
0d 0a                      . .
```

ibic における一般的な NI-488.2 ルーチン

Set

ibic では、必ず set コマンドを実行してから NI-488.2 ルーチンを使ってください。この場合の set コマンドの構文は次のようになります。

```
set 488.2  n
```

ここで n はボード番号を表します (例 : gpib0 ならば n=0)。

488.2 プロンプトが表示されていれば、ボード n での NI-488.2 モードです。次の例では、ボード gpib0 の NI-488.2 モードに移る方法を示します。

```
set 488.2 0

488.2 (0):
```

Send と SendList

Send ルーチンは 1 台の GPIB デバイスにデータを送信します。SendList コマンドを使えば、複数の GPIB デバイスにデータを送信できます。1 例として、5 文字の文字列 *IDN? の後に EOI 付きの改行文字を続けたものを送信したい場合を考えます。コンピュータからこのメッセージを、1 次アドレスが 2 と 17 のデバイスに送信するものとします。これを実行するには、次に示すように、488.2 (0) プロンプトが出ている状態で SendList コマンドを入力します。

```
488.2 (0): SendList 2, 17 "*IDN?" NLend
[0128] (cmpl cic tacs)
count: 6
```

返されたステータスワードには cmpl ビットが入っています。cmpl ビットは入出力動作が正常に終了したことを示します。バイトカウント 6 というのは、6 文字 (改行文字を含む) がコンピュータから送信され、2 台のデバイスに受信されたことを示します。

Receive

Receive ルーチンは、GPIB ボードが他の GPIB ボードからのデータを受信するように指示します。次の例では、1 次アドレス 5 のデバイスから 10 バイトのデータを受け取ります。Receive ルーチンは、10 文字の受信が完了するか、END メッセージが受信されると停止します。受け取ったデータは 16 進値とそれに対応する ASCII 文字の両方で表示されます。ibic プログラムはステータスワードと、転送済みバイト数も表示します。

```
488.2 (0): Receive 5 10 STOPend
[2124] (end cmpl cic lacs)
count: 5
48 65 6c 6c 6f      Hello
```

補助関数

ibic で使える補助関数を表 5-4 にまとめて示します。

表 5-4　ibic の補助関数

関数	説明
set udname	アクティブなデバイスまたはボードを選択します。udname は新しいデバイスまたはボードのシンボリック名です (例 : devl または gpib0)。最初に ibfind または ibdev を呼び出してボードやデバイスをオープンします。
help [option]	ヘルプ情報を表示します。option は NI-488 または NI-488.2 の呼び出しです。option を入力しないと、オプションのメニューが表示されます。
!	直前の関数を繰り返します。
-	画面表示をオフにします。
+	画面表示をオンにします。
n* function	function が正しい ibic 関数構文だった場合、関数を n 回実行します。
n* !	直前の関数を n 回繰り返します。
$ filename	間接ファイルの実行を行います。filename は、実行する ibic 関数の入ったファイルのパス名です。

表 5-4　ibic の補助関数（続き）

関数	説明
print string	文字列を画面に表示します。string は、ASCII 文字シーケンス、8 進値、16 進値、特殊記号のいずれかです。
e	終了
q	終了

Set(デバイスまたはボードの選択)

NI-488.2 モードを選択したり、現在通信中以外のデバイスと通信するには、set コマンドを使います。次の例では、NI-488.2 ルーチンを使用して gpib0 と通信している状態から、事前に ibdev 呼び出しで受け取ったユニット記述子 (ud0) を使って通信する状態へと切り替えています。

```
488.2 (0): set ud0

ud0:
```

Help(ヘルプ情報の表示)

ヘルプ機能では、選択可能な項目のメニューが表示されます。各項目には、関連の関数などの情報が入っています。help に続けて呼び出し名を入力 (例 : help ibwrt) すれば、特定の NI-488 関数や NI-488.2 ルーチンに直接アクセスできます。ヘルプには、NI-488 関数と NI-488.2 ルーチンの全ての関数構文についての説明があります。

!(直前の関数の繰り返し)

! 関数は、直前に実行した関数を繰り返します。次の例では、ibsic コマンドを発信した後、同じコマンド (ここでは ibsic) を繰り返します。

```
gpib0: ibsic
[0130] (cmpl cic atn)

gpib0: !
[0130] (cmpl cic atn)
```

–(画面表示 OFF)、+(画面表示 ON)

– 関数は、プロンプトを除く画面表示を全てオフにします。この関数は、画面出力の表示を待たずに素早く入出力関数を繰り返したいときに便利です。

+ 関数は画面出力をオンにします。

次の例では、ibrd 呼び出しを 3 回使って、連続するアルファベット 24 文字を読み取っています。

```
ud0: ibrd 8
[2100] (end cmpl)
count: 8
61 62 63 64 65 66 67 68    a b c d e f g h

ud0: -

ud0: ibrd 8

ud0: +

ud0: ibrd 8
[2100] (end cmpl)
count: 8
71 72 73 74 75 76 77 78    q r s t u v w x
```

n*(関数を n 回繰り返す)

n* 関数は、指定した関数を n 回繰り返して実行します。n は整数です。次の例では、ud0 で表されるデバイスにメッセージ "Hello" が 5 回送信されます。

```
ud0: 5*ibwrt "Hello"
```

次の例では、"Hello" という語が 5 回、20 回と送信された後、さらに 10 回送信されます。

```
ud0: 5*ibwrt "Hello"
ud0: 20* !
ud0: 10* !
```

注意が必要ですが、乗算子 (*) は関数名の一部分として機能することはなく、あくまで独立して働きます。つまり、「ibwrt "Hello"」が 20 回繰り返されるのであって、「5*ibwrt "Hello"」が 20 回繰り返されるのではありません。

$(間接ファイルの実行)

$ 関数は、指定されたファイルを読み取り、そのファイルにリストされている複数の ibic 関数を連続して実行し、キーボードからその順番に入力したのと同じように動作します。次の例では、userfile ファイルにリストされている ibic 関数を実行します。

```
gpib0: $ userfile
```

次の例では、同じ処理を 3 回繰り返します。

```
gpib0: 3*$ userfile
```

間接ファイル内の関数で、関数実行前の表示モードを変更することもできます。

Print(ASCII 文字列の表示)

print 関数では、文字列を画面に表示することができます。次の例では、プリントコマンドで ASCII 文字や 16 進値を表示させる方法を示します。

```
dev1: print "hello"
hello

dev1: print "and\r\n\x67\x6f\x6f\x64\x62\x79\x65"
and
goodbye
```

また print 関数では、間接ファイルからのコメントを表示することもできます。プリントされた文字列は、- 関数で表示をオフにしてある場合でも表示されます。

6

appmon - GPIB アプリケーションモニタ

本章では、GPIB アプリケーションモニタ appmon のインストール、構成、使用方法について説明します。appmon は、便利なデバッグツールです。

概要

appmon では、アプリケーションプログラムが実行するいくつもの GPIB 呼び出しのデバッグが便利に行えます。ある特定のイベントが発生したときにプログラムの実行が一旦停止するように appmon を構成することができます。appmon がプログラム実行を一旦停止すると、戻り値、グローバル変数、エラーコードなどのステータス情報の入ったポップアップ画面が表示されます。

appmon のインストール

appmon は NI-488.2 ソフトウェアに appmon.exe というファイルとして入っています。appmon をインストールするには、DOS プロンプトで次のように入力してください。

```
appmon
```

GPIB ドライバがなかったり、appmon が既にインストールされている場合には、エラーメッセージが表示されます。

appmon は、システムを再起動しない限りインストールされたままになります。

appmon の構成

appmon では、ある条件が発生するとプログラムをトラップし、実行を一時中断させることができます。プログラムをトラップするための条件は、次の中から選択することができます。

- NI-488.2 ルーチンまたは NI-488 関数から値が戻るたびに一時中断する。
- エラーを示す値が返されると一時中断する。

- 呼び出しを行い、GPIB ステータスワード ibsta の内の選択したビットパターンが戻り値に入っていれば一時中断する。

トラップ条件のオプションを設定するには、アプリケーションプログラムから ibtrap を呼び出すか、ibtrap.exe という特殊プログラムを実行します。アプリケーションプログラムから ibtrap を呼び出すときには、「DOS/Windows 用

NI-488.2 関数リファレンスマニュアル」を参照して正しい構文を使ってください。

ibtrap.exe ユーティリティを使うには、DOS プロンプトから ibtrap と入力します。次に、<F5> を押してトラップマスクを選択するか、<F6> を押してモニタモードを選択します。マスクまたはモニタモードを選択するときには、矢印キーとスペースバーで変更を行います。変更した内容を保存するには <Enter> を、変更を無効にする場合は <Esc> を押します。ibtrap のマスクオプションを表 6-1 にまとめます。

表 6-1　ibtrap のマスクオプション

マスクフラグ	マスクオプション
-all	全ての GPIB 呼び出し
-err	GPIB エラー
-timo	タイムアウト
-end	GPIB ボードが END または EOS を検出
-srqi	SQR オン
-rqs	デバイスがサービスを要求
-spoll	GPIB ボードがシリアルポールされた
-event	DTAS メッセージまたは DCAS メッセージを受信
-cmpl	入出力完了
-lok	GPIB ボードがロックアウト状態
-rem	GPIB ボードがリモート状態
-cic	GPIB ボードがコントローラインチャージ
-atn	アテンションがアサートされた
-tacs	GPIB ボードがトーカ
-lacs	GPIB ボードがリスナ

表 6-1　ibtrap のマスクオプション（続き）

マスクフラグ	マスクオプション
-dtas	GPIB ボードがデバイストリガ状態
-dcas	GPIB ボードがデバイスクリア状態

表 6-2　ibtrap のモニタモードオプション

モニタモー ドフラグ	モニタモード
-off	appmon オフ - 何も記録されず、トラップ動作も起こりません
-rec	appmon は全ての GPIB ドライバ呼び出しを記録するが、トラップは起こりません
-dis	appmon は全ての GPIB ドライバ呼び出しを記録し、トラップ条件が発生すると情報を表示します

フラグなしで ibtrap を呼び出すと、有効なフラグとその現在の状態がリスト表示されます。appmon の構成は変化しません。

mask パラメータと monitor パラメータに色々なフラグを選択することで、多様なトラップ構成が可能になります。次に、例をいくつか示します。

- ibtrap -cic -atn -dis　アテンションがアサートされているか GPIB ボードがコントローラインチャージのときに、全ての GPIB 呼び出しを記録し、アプリケーションモニタを表示します。
- ibtrap -srq -rec　SRQ がオンのときに、全ての GPIB ドライバ呼び出しを記録し、トラップが行われるようにトラップマスクを設定します。トラップ条件が発生してもアプリケーションモニタは表示されません。
- ibtrap -dis　トラップ条件が発生すると、全ての GPIB ドライバ呼び出しを記録し、アプリケーションモニタが表示されます。トラップマスクは変更されません。

  -dis オプションを選択し、パソコンのディスプレイをグラフィックモードに変

えるアプリケーションプログラムを実行すると、アプリケーションモニタは全ての GPIB ドライバ呼び出しは記録しますがディスプレイがキャラクタモードに戻るまでは appmon は表示されません。

- ibtrap -off　アプリケーションモニタをオフにします。記録もトラップも行われません。

appmon の使い方

アプリケーションモニタが表示されると、現在の GPIB 呼び出しのパラメータを表示させたり、表示モードとトラップモードを変更したり、それまでの GPIB 呼び出しに関する情報を調べることができます。appmon が各 GPIB 呼び出しに関して記録した情報をを図 6-1 に示します。

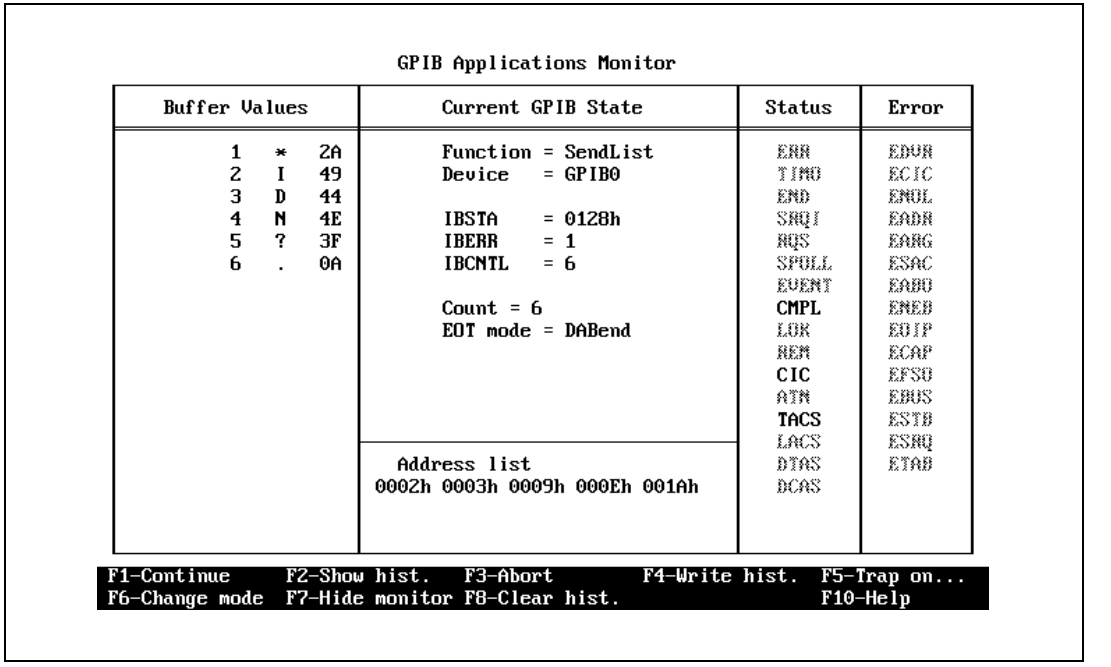


図 6-1　appmon ポップアップ画面

- Function　NI-488.2 ルーチンまたは NI-488 関数の説明
- Device　デバイスのシンボリック名

•ibsta	GPIB ステータス情報
•iberr	GPIB エラー情報。エラーが発生していなければ直前の value パラメータの値
•ibcntl	転送済みバイト数の 32 ビット表現
•Address List	アドレスリストをパラメータに持つ関数のアドレスリスト
•Buffer Value	バッファをパラメータに持つ関数のバッファの内容。バッファの各バイトは、指標、文字イメージ、ASCII 値で表示されます。
•Status	ibsta の個々のビットの状態 - アクティブビットは全て強調表示されます。
•Error	iberr の状態 - エラーが発生すると、そのエラーのニモニックが強調表示されます。

メモ　ibsta 値と、バッファ値の ASCII コードは、常に 16 進表記です。他の数字は、末尾に h が付いていない限りは 10 進表記です。

コマンドキーの使い方

appmon のメイン画面が表示されているときに使用可能なファンクションキーを表 6-3 に示します。

表 6-3　appmon のファンクションキー

キー	コマンド
<F1>	アプリケーションプログラムの実行を継続します
<F2>	セッションの概要を表示します
<F3>	実行中のプログラムを打ち切り、DOS に戻ります
<F4>	ヒストリ (履歴) をファイルに書き込みます
<F5>	トラップマスクを構成します
<F6>	モニタモードを構成します
<F7>	画面の表示 / 非表示

表 6-3　appmon のファンクションキー

キー	コマンド
<F8>	GPIB ヒストリバッファをクリアします
<F10>	コマンドキーのリストを表示します
<Cursor Up>	バッファを 1 文字前にスクロールします
<Cursor Down>	バッファを 1 文字後ろにスクロールします
<Page Up>	バッファを 1 ページ前にスクロールします
<Page Down>	バッファを 1 ページ後ろにスクロールします
<Home>	バッファの最初までスクロールします
<End>	バッファの最後までスクロールします
<Cursor Right>	アドレスリストを 2 文字右にスクロールします
<Cursor Left>	アドレスリストの最初までスクロールします

GPIB ヒストリ画面の表示

<F2> を押すと、それまでの GPIB 呼び出しに関する情報を表示することができます。情報が表示されたら、カーソルキーを使って表示を操作してください。<Esc> または <F2> を押すと、GPIB ヒストリ画面を終了して appmon のメイン画面に戻ります。

アプリケーションモニタの表示 / 非表示

アプリケーションモニタを非表示にしたり、画面の内容を再び表示させるには、<F7> を押します。<F7> をもう一度押すと、アプリケーションモニタのポップアップ画面が表示されます。

7

GPIB プログラミングテクニック

本章では、アプリケーションプログラムでの NI-488 関数と NI-488.2 ルーチンの使い方について説明します。

それぞれの関数やルーチンの詳細については「DOS/Windows 用 NI-488.2 関数リファレンスマニュアル」を参照してください。

データ転送の終了

GPIB データ転送が終了するのは、転送された最終のバイトによって GPIB EOI 線がアサートされるか、あらかじめ設定しておいた EOS(文字列の終わり) 文字が転送されたときです。NI-488.2 ドライバは、デフォルトでは、書き込みの最終バイトで EOI をアサートし、EOS モードはディスエーブルされています。

`ibeot` 関数では、EOT(転送の終わり) モードのイネーブル / ディスエーブルができます。EOT モードをイネーブルすると、NI-488.2 ドライバは、書き込みの最終バイトが GPIB 上に送信されたときに GPIB EOI 線をアサートします。EOT モードをディスエーブルすると、書き込みの最終バイトで EOI 線がアサートされなくなります。

`ibeos` 関数では、EOS モードのイネーブル / ディスエーブルしたり、構成を変えたりできます。EOS モードの構成には、次の情報が含まれます。

- EOS バイト
- EOS の比較方法 - EOS バイトの意味のあるビットが 7 ビットなのか 8 ビットなのかを示します。7 ビット EOS バイトでは、EOS バイトの第 8 ビットは無視されます。
- EOS 書き込み方法 - これがイネーブルされていると、NI-488.2 ドライバは、EOS バイトが GPIB に書き込まれると自動的に GPIB EOI 線をアサートします。たとえば `ibwrt` 呼び出しに渡されるバッファに EOS バイトが 5 つ入っている場合を考えると、EOI 線は、EOS バイトが GPIB に書き込まれる度に計 5 回アサートされます。`ibwrt` バッファに EOS バイトが入っていなければ、EOI 線はアサートされません (ただし、EOT モードがイネーブルされている場合は、書き込みの最終バイトで EOI 線がアサートされます)。
- EOS 読み取り方法 - これがイネーブルされている場合は、NI-488.2 ドライバは、EOS バイトが GPIB 上で検出されたとき、GPIB EOI 線が

アサートされたとき、指定のカウントに達したときに、ibrd, ibrda, ibrdf の各呼び出しを終了します。EOS 読み取り方法がディスエーブルされている場合は、GPIB EOI 線がアサートされたときと、指定のカウント数が読み取られたときにだけ、ibrd, ibrda, ibrdf の各呼び出しを終了します。

ibconfig 関数を使えば、EOS バイトを読み取ったときに GPIB EOI 線がアサートされたかどうかを、NI-488.2 ソフトウェアが知らせるように構成することができます。IbcEndBitIsNormal オプションを使えば、GPIB EOI 線がアサートされたときだけ ibsta の END ビットを設定して返すように NI-488.2 ソフトウェアを構成することができます。NI-488.2 ドライバは、デフォルトでは、EOS バイトを読み取るか、読み取り中に EOI 線がアサートされたときに ibsta の END を返します。

高速データ転送 (HS488)

ナショナルインスツルメンツは IEEE 488 用の高速データ転送プロトコル、HS488 を開発しました。このプロトコルは、使用システムによって異なりますが、GPIB 読み取り / 書き込みの性能を最大 8Mbytes/s まで向上させられます。

HS488 は IEEE 488 規格の上位互換規格です。したがって IEEE 488.1 デバイス、IEEE 488.2 デバイス、HS488 デバイスを一緒に 1 つのシステムで使うことができます。TNT4882C ハードウェアでは、HS488 をイネーブルすると、HS488 計測器と通信を行うときに自動的に高速転送が実現します。お使いの GPIB インタフェースボードに TNT4882C ハードウェアが付いているかどうか確かめるには、GPIBInfo ユーティリティを使います。TNT4882C チップの入っていない GPIB ボードで HS488 をイネーブルしようとすると、ECAP エラーが返されます。

HS488 のイネーブル

GPIB ボードの HS488 をイネーブルするには、ibconfig 関数 (オプション IbcHSCableLength) を使います。GPIB 構成上のケーブルが何メートルあるかを示す値を、ibconfig に渡さなければなりません。ケーブル長を実際よりもずっと小さく指定してしまうと、転送データが壊れることがあります。逆に実際よりも長く指定すると、データの転送はうまくいきますが、正しい長さを指定した場合と比較して転送速度は遅くなります。

ibconfig を使って GPIB ボードを HS488 に設定するだけでなく、コントローラインチャージは GPIB コマンドバイト (インタフェースメッセージ) を送信して、他のデバイスも HS488 転送用に構成しなければなりません。

デバイスレベル呼び出しを使っているのならば、NI-488.2 ソフトウェアは自動的に HS488 構成メッセージをデバイスに送信します。ibconf で HS488 プロトコルをイネーブルした場合には、ibdev でデバイスをオンラインにすると NI-488.2 ソフトウェアは HS488 構成メッセージを送信します。ibconfig を呼び出して GPIB ケーブル長を変更した場合には、次にデバイスレベル関数を呼び出した時、NI-488.2 ソフトウェアは HS488 メッセージを送信します。

ボードレベル関数または NI-488.2 ルーチンを使っていて、しかもデバイスを高速に設定する場合は、ibcmd または SendCmds で HS488 構成メッセージを送信する必要があります。HS488 構成メッセージは、2 バイトの GPIB コマンドです。第 1 バイトである設定イネーブル (CFE) メッセージ (16 進 "1F") は、すべての HS488 デバイスを構成モードにします。非 HS488 デバイスはこのメッセージを無視します。第 2 バイトは GPIB2 次コマンドで、システムのケーブル長が何メートルあるかを示します。第 2 バイトは設定 (CFGn) メッセージと呼ばれます。HS488 はケーブル長が 1 ～ 15 メートルの範囲にないと動作できないので、CFGn の値は 1 ～ 15(16 進 "61" ～ "6F") の範囲だけが有効です。ibconf でのケーブル長の設定が正しければ、アプリケーションプログラムで ibask(オプション IbaHSCableLength) を呼び出せばシステムで使われているケーブル長を知ることができます。CFE メッセージと CFGn メッセージについては、「DOS/Windows 用 NI-488.2 関数リファレンスマニュアル」の付録 A「Multiline Interface Messages(マルチラインインタフェースメッセージ)」を参照してください。

システム構成の HS488 への影響

どれだけ高いデータ転送速度が可能になるかは、ホストコンピュータと GPIB システムの設定によって制限を受けます。HS488 を使った最大転送速度は理論的には 8Mbytes/s ですが、ISA バスを使った IBM PC/AT 互換のパソコンで実際に可能な最大転送速度は 2Mbytes/s です。350ns の T1 遅延という IEEE 488 ケーブルにおける制限は、HS488 にも当てはまります。GPIB 構成で使っているケーブル長が増えると、HS488 での最大データ転送速度は減少します。つまり、2 つの HS488 デバイスを 2 メートルのケーブルで接続した場合には、3 つのデバイスを 4 メートルのケーブルで接続した場合よりも高速でデータ転送ができます。

GPIB 条件の待機

ibwait 関数を使うと、現在の ibsta 値を表示したり、指定した条件が GPIB に発生するまでアプリケーションを一時中断することができます。パラメータをゼロにして ibwait 関数を使うと、ibwait はただちに ibsta を更新して、返します。ibwait 関数を使って、イベントが発生するのを待機する場合、ibwait 関数に待機マスクを渡してください。待機

マスクには常に TIMO イベントを入れてください。そうでないと、待機マスクイベントが発生するまでアプリケーションはいつまでも中断したままになります。

デバイスレベル呼び出しとバス管理

NI-488 デバイスレベル呼び出しは、アプリケーションプログラムの GPIB 管理をすべて実行できるように設計されています。しかし NI-488.2 ドライバがバス管理を処理できるのは、GPIB インタフェースボードが CIC(コントローラインチャージ) のときだけです。コマンドバイトを GPIB バス上のデバイスに送信して、デバイスアドレス指定などのバス管理作業ができるのは、CIC だけです。GPIB ボードを CIC にするには、次のいずれかの方法を使います。

- GPIB ボードがシステムコントローラとして構成されていれば (デフォルト状態)、最初にデバイスレベル呼び出しを行ったときに、自動的に IFC 線をアサートして GPIB ボード自身が CIC になります。
- 複数のコントローラを設定してある場合や、GPIB インタフェースボードをシステムコントローラとして設定してない場合には、CIC プロトコル方法を使います。このプロトコルを使うには、ibconfig 関数 (オプション IbcCICPROT) を実行するか、ibconf 構成ユーティリティを使って、CIC プロトコルを起動します。GPIB インタフェースボードが CIC でない時に CIC プロトコルをイネーブルしてデバイスレベル呼び出しを行うと、次のような一連の処理が実行されます。

  1. GPIB インタフェースボードが SRQ 線をアサートします。
  2. 現在の CIC がボードとシリアルポーリングを行います。
  3. インタフェースボードは 16 進 "42" という応答バイトを返します。
  4. 現在の CIC が制御を GPIB インタフェースボードに渡します。

  現在の CIC が制御を渡さない場合は、NI-488.2 ドライバがアプリケーションに ECIC エラーコードを返します。このようなエラーは、現在の CIC が CIC プロトコルを認識しないときに発生します。この場合、GPIB ボードの制御を要求するデバイス固有コマンドを送信することができます。それから、ボードレベルの `ibwait` コマンドを使って、CIC となるまで待ちます。

トーカ / リスナアプリケーション

NI-488.2 ソフトウェアは、コントローラインチャージ用に設計されてはいますが、非コントローラ状況においてもほとんどの場合に利用できます。非コントローラ状況は、通常トーカ / リスナアプリケーションと呼ばれますが、これはインタフェースボードが GPIB コントローラではないからです。一般的なトーカ / リスナアプリケーションは、コントローラからの何らかのイベントを待ち、それに合った応答をするものです。以下に、トーカ / リスナアプリケーションのプログラミングテクニックを説明します。

コントローラからのメッセージの待機

トーカ / リスナアプリケーションは、通常、GPIB インタフェースボードのステータスをモニタするには、マスクを 0 にした `ibwait` 関数を使います。次にアプリケーションは、`ibsta` に設定されたステータスに基づいて適切な動作を実行します。1 例としては、アプリケーションがステータスビット TACS(トーカアクティブ状態) と LACS(リスナアクティブ状態) をモニタして、いつコントローラにデータを送信するか、いつコントローラからデータを受信するかを決める場合があります。また、アプリケーションが DCAS(デバイスクリアアクティブ状態) ビットと DTAS(デバイストリガアクティブ状態) ビットをモニタして、コントローラがデバイスクリア (DCL または SDC) メッセージやトリガ (GET) メッセージを GPIB インタフェースボードに送信したかどうかを調べることもできます。アプリケーションは、コントローラからデバイスクリアを検出すると、メッセージバッファの内部状態をリセットするかもしれません。検出したのがコントローラからのトリガメッセージであれば、アプリケーションが電圧計として動作しているときは、電圧を読み取るなどの動作を開始するかもしれません。

イベント待ち行列の使い方

アプリケーションによっては、コントローラからメッセージがどんな順序で送信されるか知っている必要がある場合があります。メッセージの順序をモニタするには、アプリケーションプログラムは、`ibconfig`(オプション `IbcEventQueue`) を使って EVENT ビットをイネーブルしておかなければなりません。EVENT ビットがイネーブルされると、DCAS ビットと DTAS ビットは非活動状態になります。その代わりにすべての DCAS メッセージと DTAS メッセージが、受信された順番で待ち行列に格納されます。イベント待ち行列には、インタフェースクリア (IFC) メッセージも格納されます。イベント待ち行列に何らかの情報があるときは、NI-488.2 ソフトウェアは `ibsta` の EVENT ビットを設定します。アプリケーションプログラムが EVENT を検出すると、`ibevent` 関数を呼び出して、最初に発生したイベントを取り出します。イベントを、イベント待ち

行列から取り出すことで、アプリケーションはデバイスクリア、デバイストリガ、インタフェースクリアの各メッセージに正しい順序で応答することができるようになります。

サービス要求

トーカ / リスナアプリケーションでもう 1 つ重要なイベントは、シリアルポールです。トーカ / リスナアプリケーションは、コントローラにサービス要求をするには、シリアルポール応答バイトを使って `ibrsv` を呼び出すことができます。アプリケーションが、コントローラがいつシリアルポール応答バイトを読んだのか知る必要のあるときには、`ibconfig`(オプション `IbcSPollBit`) で `ibsta` の SPOLL ビットをイネーブルしておきます。SPOLL ビットは、GPIB インタフェースボードがサービスを要求し、コントローラがシリアルポールを行ったときに NI-488.2 ソフトウェアによって設定されます。

複数アドレスのシミュレーション

NI-488.2 ソフトウェアでは、トーカ / リスナアプリケーションは複数の GPIB アドレスアドレスを仮想化することができます。複数の GPIB アドレスを仮想化することで、トーカ / リスナアプリケーションはコントローラに対して複数の別個の GPIB デバイスであるかのようにふるまうことができます。この機能を使えば、コントローラがどの GPIB アドレスをアプリケーションに送信したかによって、いくつかの異なった役割をこなすアプリケーションを作成することができます。

`GotoMultAddr` 関数を使うと、アプリケーションは NI-488.2 ドライバに 2 つのルーチンを登録します。NI-488.2 ドライバは、コントローラがバスを通じて GPIB アドレスを送信すると、第 1 ルーチンを呼び出します。第 1 ルーチンは、そのアドレスを受け入れるか拒否するかを決めなければなりません。アドレスを受け入れたときは、インタフェースボードは後続の処理でその新しいアドレスに対して応答します。コントローラがもう一方のアドレスを送信すると、同じプロセスが行われます。第 2 ルーチンが呼び出されるのは、コントローラが仮想化アドレスの 1 つとシリアルポールを行ったときです。第 2 ルーチンは適切なシリアルポール応答バイトを返さなければなりません。このバイトを NI-488.2 ドライバがコントローラに送信します。

シリアルポーリング

シリアルポーリングでは、GPIB デバイスがサービスを要求しているとき特定の情報を得ることができます。GPIB SRQ 線がアサートされているときは、コントローラに対してサービス要求への対応がまだであることを示しています。コントローラはどのデバイスが SRQ 線をアサートしている

かを調べ、それぞれに適切に応答する必要があります。SRQ 線の検出とそれへの応答に一番多く使われている方法は、シリアルポールです。この節では、GPIB デバイスからのサービス要求を検出し、それに応答できるように設定する方法について説明します。

IEEE 488 デバイスからのサービス要求

IEEE 488 デバイスは、GPIB SRQ 線をアサートして GPIB コントローラにサービスを要求します。コントローラは SRQ 線を確認すると、バス上のオープンデバイス 1 つ 1 つとシリアルポールを行い、どのデバイスがサービスを要求しているのか調べます。ここでサービスを要求しているデバイスは、ビット 6 が設定されたステータスバイトを返し、SRQ 線のアサートを解除します。サービスを要求していないデバイスはビット 6 がクリアされたステータスバイトを返します。IEEE 488 デバイスのメーカーは、サービス要求の理由を伝えたりデバイスの状態を報告するためには、ビット 6 より下位のビットを使っています。

IEEE 488.2 デバイスからのサービス要求

IEEE 488.2 規格ではステータスバイトのビット割り当てが高度化されました。IEEE 488.2 デバイスは、サービスを要求するときにビット 6 を設定するだけでなく、他にも 2 つのビットを使って自分のステータスを特定します。MAV(メッセージ利用可能ビット) であるビット 4 は、直前に問い合わせを受けたデータの送信準備ができていると設定されます。ESB(イベントステータスビット) であるビット 5 は、イネーブルされた IEEE 488.2 イベントが発生すると設定されます。IEEE 488.2 イベントには、電源オン、ユーザ要求、コマンドエラー、実行エラー、デバイス依存エラー、問い合わせエラー、要求制御、処理完了などがあります。IEEE 488.2 デバイスは、ESB か MAV が設定されたときと、メーカー指定の条件が発生したときに SRQ 線をアサートします。

自動シリアルポーリング

SRQ 線がアサートされた時にいつでも自動的にシリアルポールを実行したい場合、自動シリアルポーリングをイネーブルします。自動ポーリングが可能なのは、NI-488 デバイスレベル呼び出しだけです。自動ポーリングは次のような手順で行われます。

1. 自動ポーリングをイネーブルするには、構成ユーティリティ `ibconf` を使うか、構成関数 `ibconfig` を、オプション `IbcAUTOPOLL` で使います (自動ポーリングはデフォルトではイネーブルされています)。
2. SRQ 線がアサートされると、NI-488.2 ドライバが自動的にオープンデバイスとのシリアルポールを行います。

3. 肯定のシリアルポール応答 (ビット 6 つまり 16 進 "40" が設定されている) は、送信元のデバイスに対応した待ち行列に格納されます。デバイスステータスワード ibsta の RQS ビットが設定されます。
4. ポーリングは、SRQ 線のアサートが解除されるか、エラー状況が検出されるまで続きます。
5. 待ち行列を空にするには、`ibrsp` 関数を使います。`ibrsp` は待ち行列に格納された最初の応答を返します。残りの応答は先入れ先出し (FIFO) 方式で読み取られます。`ibrsp` を呼び出したときにステータスワードの RQS ビットが設定されていなければ、シリアルポールが実行され、そこでデバイスから受信した応答を返します。自動シリアルポールが行われたら、すぐに待ち行列を空にしてください。もし待ち行列が一杯だと、応答が格納できず捨てられてしまうことがあります。
6. `ibrsp` を呼び出した後もステータスワードの RQS ビットが設定されたままの場合は、応答バイト待ち行列に応答バイトが少なくとも 1 つは残っています。このようなときは、RQS がクリアされるまで `ibrsp` を呼び出し続けてください。

スタック SRQ 状態

自動ポーリングがイネーブルで、GPIB インタフェースボードが SRQ を検出すると、ドライバはボードに接続されているすべてのオープンデバイスとシリアルポールを行います。シリアルポールは、SRQ のアサートが解除されるか、すべてのデバイスとのポールが終了するまで続きます。

シリアルポーリングに肯定応答をするデバイスがなかった場合や、計測器やケーブルの障害のために SRQ がアサートされたままの場合には、スタック SRQ 状態が発生します。`ibwait` で RQS を待っているときにこのような状態が発生すると、ドライバは ESRQ エラーを返します。スタック SRQ 状態が発生したときには、もう一度 RQS を待つ `ibwait` を実行するまで、それ以上のポールは行われません。`ibwait` を実行すれば、スタック SRQ 状態が終了し、ドライバはシリアルポールを新たに開始します。

自動ポーリングと割り込み

自動ポーリングと割り込みを両方ともイネーブルしてあれば、NI-488.2 ソフトウェアは、GPIB 入出力の実行中でない限りは、デバイスレベル NI-488 呼び出しの後で自動ポーリングを実行することができます。つまり、アプリケーションが NI-488.2 ソフトウェアに呼び出しをしていないときでも、自動シリアルポールは実行されるということです。自動ポーリングは、RQS を待つデバイスレベル `ibwait` の実行中でも行うことができます。ただし、アプリケーションがボードレベル NI-488 関数または

NI-488.2 ルーチンのどれかを呼び出しているときと、スタック SRQ 状態 (ESRQ) が発生しているときは、自動ポーリングは行えません。

自動ポーリングをイネーブルし、割り込みをディスエーブルしてあるときに、自動ポーリングを行えるのは次の場合だけです。

- RQS 待ちのデバイスレベル `ibwait` の実行中
- デバイスレベル NI-488 関数の完了直後で、制御がアプリケーションプログラムに戻される前

"ON SRQ" 機能

NI-488.2 ソフトウェアでは、アプリケーションは GPIB SRQ 線のアサートに非同期的に応答することができます。アプリケーションが NI-488.2 ソフトウェアの ON SRQ 機能をイネーブルした場合は、GPIB SRQ 線がアサートされると、ユーザが作成した関数が呼び出されます。この機能により、デバイスがサービスを要求したときのアプリケーションの応答をカスタマイズすることができます。

メモ　ON SRQ 機能が正しく動作するためには、`ibconfig`(オプション `IbcAUTOPOLL`) で自動シリアルポーリングをディスエーブルしておかなければなりません。

C 言語の "ON SRQ" 機能

C 言語では、NI-488 `ibsrq` 関数を使って SRQ に非同期的に応答することができます。アプリケーションは NI-488 `ibsrq` 関数を使って SRQ 処理ルーチンを指定することができます。このルーチンは、NI-488.2 ドライバが SRQ 線のアサートを検出すると呼び出されます。SRQ 処理ルーチンは、割り込み処理ルーチンではありません。NI-488.2 ドライバは、NI-488 関数や NI-488.2 ルーチンの完了後に GPIB SRQ 線をチェックし、もし SRQ がアサートされていて、しかもアプリケーションが `ibsrq` を呼び出していれば、ユーザ定義 SRQ 処理ルーチンが呼び出されます。

BASIC/QuickBASIC/BASICA の "ON SRQ" 機能

BASIC/QuickBASIC/BASICA アプリケーションも SRQ に非同期的に応答することができます。BASIC 言語の ON PEN 機能は、NI-488.2 ドライバによって自動的に ON SRQ 機能に変換されます。BASIC アプリケーションで ON SRQ 機能をイネーブルするには、次に示すサンプルコードを実行してください。

```
100  REM Define srq-handling routine (MySRQRoutine)
200  ON PEN GOSUB MySRQRoutine
300  REM Enable the ON SRQ functionality
```

```
400  PEN ON
```

BASIC アプリケーションでは、SRQ 処理ルーチンである MySRQRoutine を定義しなければなりません。このルーチンは、BASIC 環境で SRQ 線のアサートが検出されると呼び出されます。この SRQ 処理ルーチンは、割り込み処理ルーチンではありません。BASIC 環境では、BASIC コードの実行部の行と行との間でしか SRQ に応答しません。

NI-488 デバイス関数を使った SRQ とシリアルポーリング

デバイスレベル NI-488 関数 ibrsp を使ってシリアルポールを行うことができます。ibrsp はシングルシリアルポールを実行し、シリアルポール応答バイトをアプリケーションプログラムに返します。自動シリアルポーリングをイネーブルしてあれば、アプリケーションプログラムは、ibwait を使ってステータスワード ibsta に RQS が設定されるまでプログラムの実行を一時中断することができます。ここでプログラムが ibrsp を呼び出せばシリアルポール応答バイトを受け取ることができます。

次の例では、自動シリアルポーリングをイネーブルしてある場合の代表的な SRQ 処理状況での ibwait 関数と ibrsp 関数の使い方を示してあります。

```
#include "decl.h"

char GetSerialPollResponse ( int DeviceHandle )
{
    char SerialPollResponse = 0;
    ibwait ( DeviceHandle, TIMO | RQS );
    if ( ibsta & RQS )  {
        printf ( "Device asserted SRQ.\n" );
        /*  Use ibrsp to retrieve the serial poll
           response.  */
        ibrsp ( DeviceHandle, &SerialPollResponse );
    }
    return SerialPollResponse;
}
```

NI-488.2 ルーチンを使った SRQ とシリアルポーリング

NI-488.2 ソフトウェアには、SRQ 処理とシリアルポールを行うための NI-488.2 ルーチンが入っています。SRQ 処理とシリアルポーリングに関連するルーチンには、AllSpoll, FindRQS, ReadStatusByte, TestSRQ, WaitSRQ があります。

AllSpoll ルーチンは、1 回の呼び出しで複数デバイスとのシリアルポールが行えます。このルーチンは、ポール先の各計測器から受け取ったステータスバイトを、設定しておいた配列に保存します。ここで、各ステータスバイトの RQS ビットを確認し、そのデバイスがサービスを要求しているのかどうか調べなければなりません。

ReadStatusByte ルーチンは、AllSpoll によく似ていますが、デバイス 1 台としかシリアルポールを行わない点が違います。また、デバイスレベル NI-488 関数 ibrsp にもよく似ています。

FindRQS ルーチンはリストに指定されたデバイスとシリアルポールを行います。シリアルポールは、サービスを要求しているデバイスを検出するか、そのデバイス全てとのポールが終了するまで続けられます。このルーチンは、サービスを要求しているデバイスの指標とステータスバイトの値を返します。

TestSRQ ルーチンは、SRQ 線がアサートされているかアサート解除されているかを調べ、その結果をただちにアプリケーションプログラムに返します。

WaitSRQ ルーチンは TestSRQ ルーチンとよく似ていますが、SRQ がアサートされるかタイムアウト時間が経過するまでアプリケーションの実行を一時中断する点が違います。

次の例では、NI-488.2 ルーチンを使って SRQ を検出し、どのデバイスがサービスを要求しているか調べます。この例では 3 つのデバイスが GPIB 上のアドレス 3, 4, 5 にあり、GPIB インタフェースはバス指標 0 に設定されています。最初の例では FindRQS ルーチンを使い、どのデバイスがサービスを要求しているか調べ、2 番目の例では AllSpoll を使い、3 つのデバイスすべてとシリアルポーリングを行います。どちらの例でも WaitSRQ を使って、GPIB SRQ 線がアサートされるのを待ちます。

メモ　この例では NI-488.2 ルーチンと同時には使えないので、自動シリアルポーリングは使用していません。

例 1:　FindRQS の使い方

この例では、どのデバイスがサービスを要求しているかを FindRQS を使って調べる方法を示します。

```
void GetASerialPollResponse ( char *DevicePad, char
*DeviceResponse )
{
    char SerialPollResponse = 0;
    int WaitResult;
```

```
    Addr4882_t Addrlist[4] = {3,4,5,NOADDR};

    WaitSRQ (0, &WaitResult);

    if (WaitResult) {
        printf ("SRQ is asserted.\n");
/* Use FindRQS to find a device that requested service. */

        FindRQS ( 0, AddrList, &SerialPollResponse );
        if (!(ibsta & ERR))  {
            printf ("Device at pad %x returned byte %x.\n",
                   AddrList[ibcnt],(int)
            SerialPollResponse);
            *DevicePad = AddrList[ibcnt];
            *DeviceResponse = SerialPollResponse;
        }
    }

    return;
}
```

例 2:　AllSpoll の使い方

この例では、AllSpoll を使って 3 台のデバイスとシリアルポールを行う方法を示します。

```
void GetAllSerialPollResponses ( Addr4882_t AddrList[],
short ResponseList[] )
{
    int WaitResult;

    WaitSRQ (0, &WaitResult);

    if ( WaitResult ) {
        printf ( "SRQ is asserted.\n" );


/* Use Allspoll to serial poll all the devices at once. */

        AllSpoll ( 0, AddrList, ResponseList );
        if (!(ibsta & ERR))  {
```

```
            for ( i = 0; AddrList[i] != NOADDR; i++ )  {
                  printf ("Device at pad %x returned byte
                    %x.\n",AddrList[i], ResponseList[i] );
            }
      }
   }

   return;
}
```

パラレルポーリング

パラレルポーリングはあまり広くは使われませんが、複数のデバイスのステータスを一度に収集するのには便利な方法です。パラレルポールのメリットとして、最大 8 台までのデバイスを 1 回で検査することができます。比較のために述べておくと、8 台のデバイスのシリアルポール応答バイトを検査するためには、デバイス 8 台を別々にシリアルポールしなければなりません。

パラレルポーリングの実行

パラレルポーリングは、NI-488 関数と NI-488.2 ルーチンのどちらかを使って実行できます。パラレルポールを実行するのに NI-488.2 ルーチンを使えば、多くのパラレルポーリングメッセージを覚える必要はありません。ただし、GPIB ボードがコントローラではなくて、自分自身をパラレルポール用に構成した上で個々のステータスビット (`ist`) を設定しなければならないときには、パラレルポールには NI-488 関数を使ってください。

NI-488 関数を使ったパラレルポーリング

NI-488 関数を使ってパラレルポーリングを行うには、次の手順にしたがってください。各ステップごとにコード例を示してあります。

1. `ibppc` 関数を使ってデバイスをパラレルポール用に構成します。ただし、デバイスがデバイス自身をパラレルポール用に構成できる場合は除きます。

   `ibppc` は 8 ビットの値でデータ線の番号と `ist` センスを割り当て、さらに関数がデバイスをパラレルポール用に構成するのか構成解除するのかを決めます。ビットパターンは次の通りです。

   0 1 1 E S D2 D1 D0

   E の設定は、指定のデバイスのパラレルポーリングをディスエーブルするときには 1 に、イネーブルするときには 0 にします。

S の設定は、割り当てたデータ線を ist=1 のときにデバイスにアサートさせたいときには 1 に、ist=0 のときにアサートさせたいときには 0 にします。

D2 ～ D0 は、割り当てるデータ線の番号です。物理的な線番号は 2 進表記の線番号に 1 を加えたものです。たとえば DIO3 の 2 進表記ビットパターンは 010 になります。

次のコード例では、NI-488 関数を使ってデバイスをパラレルポール用に構成しています。デバイスは、ist=0 のときに DIO7 をアサートします。

この例で ibdev コマンドを使ってオープンするデバイスは、1 次アドレスが 3、2 次アドレスなし、タイムアウトが 3 秒です。また、書き込み処理の最終バイトで EOI をアサートし、EOS 文字はディスエーブルされています。

```
#include "decl.h"
char ppr;

dev = ibdev(0,3,0,T3s,1,0);

/* Pass the binary bit pattern, 0110 0110 or hex 66, to
ibppc. */

ibppc(dev, 0x66);
```

GPIB インタフェースボードが自分自身をパラレルポール用に構成する場合でも、やはり ibppc 関数を使わなければなりません。ibppc の第 1 引数としてボード指標かボードユニット記述子の値を渡してください。また、GPIB インタフェースボードの個々のステータスビット (ist) を変更する必要のあるときは、ibist 関数を使います。

次の例では、GPIB ボードは GPIB ボード自身がパラレルポールに参加できるように構成します。パラレルポールが実行されると、GPIB ボードは ist=1 のときに DIO5 をアサートします。

```
ibppc(0, 0x6C);
ibist(0, 1);
```

2. ibrpp を使ってパラレルポールを実行し、応答がある値かどうかを検査します。次のコード例では、パラレルポールを実行し、応答を 16 進 "10"(DIO5 に対応) と比較します。検査したビットが設定されていれば、デバイスの ist は 0 です。

```
ibrpp(dev, &ppr);
if (ppr & 0x10) printf("ist = 0\n");
```

3. ibpcc を使って、デバイスのパラレルポール構成を解除します。ビットパターンのパラレルポールのディスエーブルビット (ビット 4) が設定されていればどんな値でも、パラレルポール構成をディスエーブルするので、16 進 "70" ～ "7E" の範囲ならばどの値でも使用できることに注意してください。

```
ibppc(dev, 0x70);
```

NI-488.2 ルーチンを使ったパラレルポーリング

NI-488.2 ルーチンを使ってパラレルポーリングを行うには、次の手順にしたがってください。各ステップごとにコード例を示してあります。

1. PPollConfig ルーチンを使って、デバイスをパラレルポール用に構成します。ただし、デバイスがデバイス自身をパラレルポール用に構成できる場合は除きます。次の例では、アドレスが 3 のデバイスを、ist=1 のときにデータ線 5(DIO5) をアサートするように設定します。

```
#include "decl.h"
char response;
Addr4882_t AddressList[2];

/* The following command clears the GPIB.    */

SendIFC(0);

/* The value of sense is compared with the ist bit of
the device
and determines whether the data line is asserted.    */

PPollConfig(0,3,5,1);
```

2. PPoll を使ってパラレルポールを実行し、応答を保存し、その応答がある値かどうかを検査します。次の例では、ist=1 のときにデバイスが DIO5 をアサートするので、プログラムは応答のビット 4(16 進 "10") を検査して ist の値を求めます。

```
PPoll(0, &response);

/* If response has bit 4 (hex 10) set, the ist bit of
   the device at that time is equal to 0.  If it does
   not appear, the ist bit is equal to 1.  Check the
   bit in the following statement. */

if (response & 0x10) {
      printf("The ist equals 0.\n");
```

```
}
else {
       printf("The ist equals 1.\n");
}
```

3. 次の例に示すように PPollUnconfig ルーチンを使い、デバイスのパラレルポール構成を解除します。この例では、NOADDR 定数を配列の最後に置き、アドレスリストの終わりであることを示さなければなりません。配列に NOADDR 以外の値が何もないときは、すべてのデバイスがパラレルポールのディスエーブルメッセージを受信します。

```
AddressList[0] = 3;
AddressList[1] = NOADDR;
PPollUnconfig(0, AddressList);
```

8

ibconf - インタフェースバス構成ユーティリティ

この章では、NI-488.2 ソフトウェアの構成に使用するソフトウェア構成プログラム、ibconf について説明します。

概要

ibconf ユーティリティは、GPIB インタフェースボードおよびそれらに接続されている GPIB デバイスの構成パラメータの表示や変更に使用する、画面指向の対話式プログラムです。通常、GPIB インタフェースボードのハードウェアの設定には ibconf を使用します。

ibconf ユーティリティは、ディスク上の NI-488.2 ドライバファイルの構成パラメータを読み込み、表示することができます。ユーザはそれから変更を行い、ディスクファイルを保存できます。さらに、ibconf をバッチモードで使用して、非対話方式で構成することもできます。

ibconf を使用せずに、ドライバをプログラムで構成することもできます。この場合、ibconfig 関数を使用して、プログラムの実行中にボードやデバイスの特性を変更します。ダイナミック構成の場合、アプリケーション開始前に ibconf を実行しておく必要はありません。また、適切な NI-488.2 ソフトウェアを使用するコンピュータであれば、アプリケーションがドライバを必要に応じて構成するため、構成が異なっていてもアプリケーションを実行することができます。GPIB アプリケーションの開発には、プログラムによる構成が適しています。

ibconf の起動

ibconf を使用するには、NI-488.2 ソフトウェアがインストールされたディレクトリに移り、次のコマンドを入力するのが最も簡単です。

```
ibconf
```

現在のディレクトリにドライバ gpib.com がない場合、ibconf は変更する gpib.com を探します。一般に ibconf は、以下のプロセスで構成する gpib.com ファイルを見つけます。

1. c:\config.sys ファイルが存在し、フォーマット行 device=<path>gpib.com がある場合、その gpib.com ファイルが構成されます。
2. config.sys ファイルが現在のドライブのルートディレクトリに存在し、フォーマット行 device=<path>gpib.com がある場合、その gpib.com ファイルが構成されます。
3. gpib.com ファイルが現在のディレクトリにある場合、そのファイルが構成されます。

表 8-1 に、ibconf を起動するときに選択できるオプションを示します。

表 8-1　ibconf の起動オプション

ibconf のオプション	動作
driver	指定のドライバを構成します。 ibconf は gpib.com ファイルを探す代わりに、指定されたドライバファイルを構成します (例 : ibconf d:\at-gpib\at-gpib.com)。
-h	ヘルプ。 このオプションを使用すると、オプションの要約が表示されます。
-b *filename*	バッチモード構成。 ibconf は、所定のファイルに入っている構成情報を使用してバッチモードで実行されます (例 : ibconf -b gpib.cfg)。本章後半の「ibconf のバッチモード」の節を参照してください。
-d	ダイナミック構成。 このオプションを使用すると、ibconf を終了するときに、メモリにロードされたドライバが自動的に更新されます。本章後半の「ibconf の終了」の節を参照してください。
-e	エキスパートモード。 このオプションを使用すると、ibconf を終了するときに、ibconf の警告メッセージがディスエーブルされます。本章後半の「ibconf の終了」の節を参照してください。
-f	ダイナミック構成のディスエーブル。 このオプションを使用すると、ibconf を終了するときに、メモリにロードされているドライバが無視されます。ibconf はロードされているドライバを更新しません。

表 8-1　ibconf の起動オプション（続き）

ibconf のオプション	動作
-m	白黒モード。 このオプションでは、カラーモニタを使用していても、ibconf は白黒モードで実行されます。
-vb	BIOS からのビデオアクセス。 このオプションを使用すると、ibconf は画面への表示に、システムの BIOS ルーチンを使用します。この場合、画面に直接アクセスするデフォルト設定より処理が遅くなりますが、一定の非標準システムとの互換性が高まります。

ibconf の上位レベルと下位レベル

ibconf には、上位レベルと下位レベルがあります。上位レベルは、GPIB システムを図で示したデバイスマップからなっています。下位レベルはシステムを構成する個々のボードとデバイスについて説明する画面からなります。

上位レベルデバイスマップ

図 8-1 に、上位レベルの ibconf を示します。

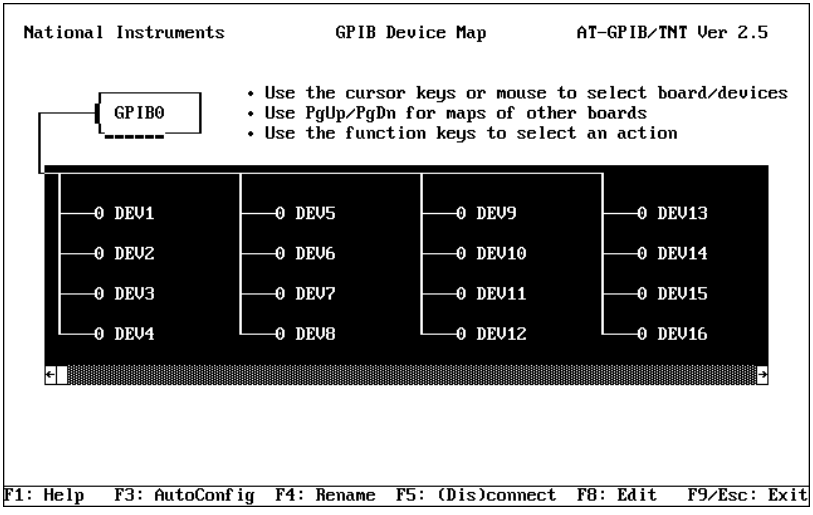


図 8-1　上位レベルの ibconf

図 8-1 に示すとおり、ibconf の上位レベル画面には、ドライバで制御されるすべてのデバイス名が表示されます。また、各インタフェースボード

からどのデバイスにアクセスできるかも示します。マップ内の移動にはカーソル制御キーまたはマウスを使用します。

上位レベルには以下のオプションがあります。

- ボードのデバイスマップ
- Help(ヘルプ)
- Rename(名称変更)
- (Dis)connect(接続 / 非接続)
- Edit(編集)
- GPIB ドライバ構成の出力
- Autoconfigure(自動構成)
- Exit(終了)

ボードのデバイスマップ

異なる GPIB インタフェースボードのデバイスマップをトグルするには、<PageUp> または <PageDown> を使用します。これらのボードは、アクセスボードと呼ばれます。マップは、各ボードにどのデバイスが割り当てられているかを示します。

Help(ヘルプ)

`ibconf` 全体についてのオンラインヘルプ機能にアクセスするには、ファンクションキー <F1> を使用します。ヘルプには、上位レベル `ibconf` 関連の関数と一般的な用語についての説明があります。

Rename(名称変更)

デバイス名を変更するには、ファンクションキー <F4> を使用します。カーソル制御キーで名前を変更するデバイスに移動します。<F4> を押して、デバイスの新しい名前を入力します。デバイス名は小文字または大文字で 8 文字以内です。

デバイス名を変更する場合、以下のような制約があります。

- 拡張子 (`.xxx`) は使用できません。
- DOS での指定と同様、デバイス名には以下の文字は使用できません。

. " / \ ( ) :

| < > + = ; ,

- デバイスには予約名 con または nul は使用しないでください。
- GPIB デバイスには、ファイル、ディレクトリ、サブディレクトリと同じ名前を付けないでください。たとえば、システムに pltr.dat というファイルや pltr というサブディレクトリがあるのに、GPIB デバイスに pltr という名前を付けると矛盾が生じます。
- アクセスボード名、gpib0, gpib1,gpib2, gpib3 は固定されており、変更することはできません。

(Dis)connect(接続 / 非接続)

デバイスを特定のアクセスボードに接続または非接続するには、ファンクションキー <F5> を使用します。カーソル制御キーで接続 / 非接続するデバイスにカーソルを動かし、<F5> キーを押します。

Edit(編集)

特定のボードまたはデバイスの特性を変更したり調べたりするときは、ファンクションキー <F8> または <Enter> を使用します。カーソル制御キーで変更するボードまたはデバイスに移動し、<F8> キーを押します。これで ibconf は下位レベルに入り、変更するボードまたはデバイスの特性をリストします。Edit を終了するには、ファンクションキー <F9> または <Escape> を押します。

GPIB ドライバ構成の出力

GPIB ドライバを構成すると、ドライバ構成のテキストバージョンをディスクのファイルに書き込むことができます。ファンクションキー <F2> を使用して、ibconf で現在のディレクトリに gpib.txt という名前のテキストファイルを作成することができます。このファイルには、現在の GPIB ドライバの説明が入りますが、参照の目的のみに使用します。

Autoconfigure(自動構成)

自動構成にはファンクションキー <F3> を使用します。特定の GPIB ボードを自動構成すると、ibconf はすべてのリスンデバイスを検出し、それらのデバイスだけが接続されるようにそのボードのデバイスマップを調整します。また、デバイスの 1 次および 2 次アドレスフィールドを調整して、リスナとして応答したアドレスに一致させます。すべての処理にかかる時間はほんの数秒です。自動構成の前に、システムのすべてのデバイスが接続され電源が ON になっていることを確認してください。

注意：　<F3> を押して自動構成をしてしまうと、新しい構成を取り消すことはできません。

複数のインタフェースボードがあるシステムで自動構成機能を使用するときは、一番小さい番号のボードから昇順で自動構成を行います。ibconf は、番号の小さいインタフェースボードの自動構成中に、大きな番号のボードからデバイスを切り離すことがあるため、必ず昇順 (gpib0, gpib1,...) で行ってください。

Exit(終了)

ibconf を終了するには、ファンクションキー <F9> または <Escape> を使用します。変更を行うと、ibconf は終了する前に変更を保存するよう促すプロンプトを表示します。詳しくは、本章後半の「ibconf の終了」の節を参照してください。

下位レベルのデバイス / ボードの特性

ibconf の下位レベル画面には、図 8-2 に示すとおり、アドレッシングやタイムアウト情報など、デバイスやボードの特性について現在定義されている値が表示されます。

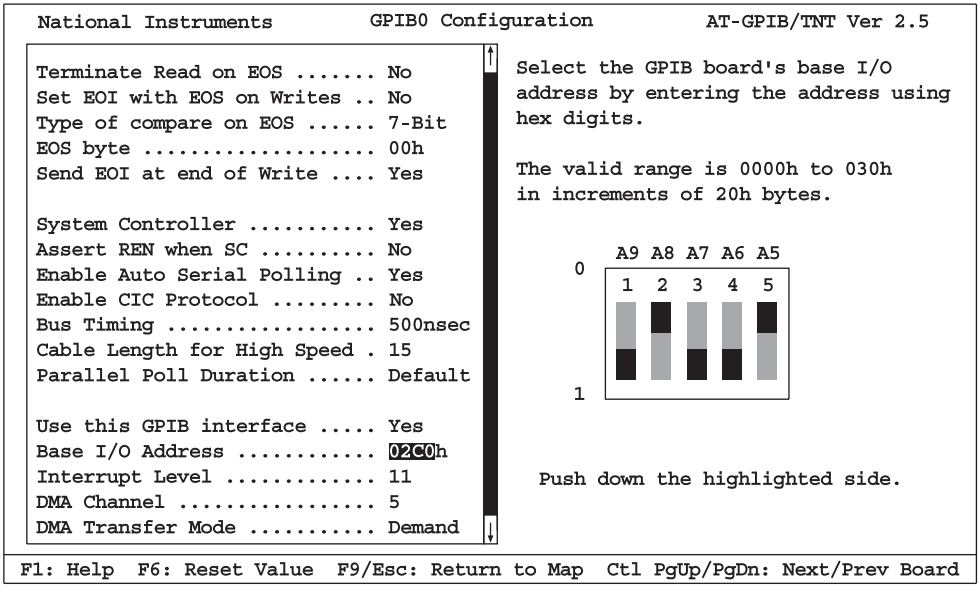


図 8-2　下位レベルの ibconf

下位レベルの画面には、ibconf のマップレベルでボードまたはデバイスを選択して <F8> を押すとアクセスできます。特性を選択するには、

<Up>, <Down>, <PageUp> および <PageDown> カーソルキーを使用します。すぐに参照できるように、画面の下にカーソル制御キーとファンクションキーが表示されます。

各デバイスまたはボードの構成の設定を選択すると、そのボードまたはデバイスで使用する通信や他のオプションをカスタマイズすることになります。アクセスボードは、デバイスをプログラムするようデバイス関数が呼び出されたとき、あるいはボードをプログラムするようボード関数が呼び出されたときに、これらの設定を使用します。

下位レベルには、以下のオプションがあります。

- 特性の変更
- ボードまたはデバイスの変更
- Help(ヘルプ)
- Reset Value(値のリセット)
- Return to Map(マップへ戻る)

特性の変更

デバイスまたはボードの特定の特性を変更するには、その特性のフィールドにカーソルを移動するか、その上でマウスをクリックします。また、デバイスまたはボードの他の特性を選択するには、<PageUp>, <PageDown>, <Home>, <End> を使用します。カーソルが特性の上にあるときに、左右の矢印キーでオプションを選択するか、キーボードから直接オプションを入力します。画面右側の指示は、選択した特性にはどの方法が適しているかを示します。

ボードまたはデバイスの変更

次の、あるいは前の GPIB ボードまたはデバイスに移るには、<Control-PageUp> および <Control-PageDown> を使用します。たとえば、dev3 を変更しているときに <Control-PageUp> を押すと、`dev4` に移ります。

Help(ヘルプ)

`ibconf` 全体についてのオンラインヘルプ機能にアクセスするには、ファンクションキー <F1> を使用します。ヘルプには、下位レベル `ibconf` 関連の関数と一般的な用語についての説明があります。

Reset Value(値のリセット)

特性のオプションを直前の値に戻すには、ファンクションキー <F6> を使用します。

Return to Map(マップに戻る)

下位レベルでファンクションキー <F9> または <Escape> を押すと ibconf の上位レベルデバイスマップに戻ります。

ボードおよびデバイスの構成オプション

各特性の詳しい情報を表示するには、カーソルをその特性のフィールド上に置きます。あるドライバに特有の特性の情報については、インタフェースボードの「入門マニュアル」をお読みください。以下の節では、DOS 用 NI-488.2 ソフトウェアの ibconf で使用できるオプションについて説明します。

Primary GPIB Address(1 次 GPIB アドレス)

すべてのデバイスおよびボードには、16 進 "00" から "1E"(10 進値 0 から 30 まで) の範囲でユニークな 1 次アドレスを指定しなければなりません。すべての GPIB ボードのデフォルトの 1 次アドレスは 0 です。

デバイスの 1 次 GPIB アドレスは、ハードウェアスイッチまたはソフトウェアプログラムによってデバイス内に設定されます。デバイス内に設定されたアドレスは、メモリ常駐のドライバのアドレスと一致していなければなりません。DOS 用 NI-488.2 ドライバでは、dev1 から dev16 および dev17 から dev32 のデフォルトの 1 次アドレスはそれぞれ 1 から 32 です。デバイスアドレスの設定に関しては、デバイスのマニュアルを参照してください。GPIB ボードには、GPIB アドレスを選択するためのハードウェアスイッチはありません。

1 次 GPIB アドレスは、デバイスとボードのトークアドレスおよびリスンアドレスを計算するために使用されます。NI-488.2 ドライバは、1 次アドレスに 16 進 "20" を加えてリスンアドレスを、1 次アドレスに 16 進 "40" を加えてトークアドレスを自動的に設定します。たとえば、1 次アドレスが 16 進 "10" の場合、リスンアドレスは 16 進 "30"、トークアドレスは 16 進 "50" になります。

Secondary GPIB Address(2 次 GPIB アドレス)

拡張アドレッシングを使用しているデバイスまたはボードには、16 進 "60" から "7E"(10 進値 96 から 126 まで) の範囲の 2 次アドレスを指定しなければなりません。あるいは、NONE を選択して 2 次アドレッシングをディスエーブルすることもできます。

1 次アドレスと同様、デバイスの 2 次 GPIB アドレスは、ハードウェアスイッチまたはソフトウェアプログラムによってデバイス内に設定されます。デバイス内に設定されたアドレスは、メモリ常駐のドライバのアドレ

スと一致していなければなりません。2 次アドレスの設定に関しては、デバイスのマニュアルを参照してください。すべてのボードおよびデバイスのこの特性のデフォルト設定は NONE です。

Timeout Setting(タイムアウトの設定)

タイムアウト値は、GPIB 関数がデータやコマンドが送信されるのを待つおおよその時間です。また、TIMO ビットがイベントマスクに設定されている場合は、ibwait 関数がイベントを待つ時間を意味します。たとえば、イベントマスクの SRQI ビットと TIMO ビットが ibwait 関数に渡され、SRQ が検出されずにタイムアウト時間が経過すると、関数はタイムアウトになります。この特性のデフォルトのオプションは 10 秒です。

Serial Poll Timeouts(シリアルポールタイムアウト、デバイス特性のみ)

このタイムアウト値は、ドライバがデバイスからのシリアルポール応答を待つ時間を制御します。IEEE 488.1 の仕様では、コントローラが応答バイトを待つ時間は指定されていません。ほとんどのデバイスには、デフォルトの 1 秒で十分なはずです。シリアルポールに問題がある場合は、タイムアウトを長めに設定してみてください。

Terminate Read on EOS(EOS による読み取りの終了)

デバイスによっては、データメッセージの最終バイトを示す EOS バイトを送信するものがあります。このフィールドに yes を設定すると、GPIB ボードは EOS バイトを受け取ると読み取り処理を終了します。この特性のデフォルトのオプションは no です。

Set EOI with EOS on Writes(書き込みで EOS とともに EOI を設定)

このフィールドに yes を設定すると、GPIB ボードは書き込み処理で EOS バイトが検出されると、EOI 線をアサートします。この特性のデフォルトのオプションは no です。

Type of Compare on EOS(EOS での比較のタイプ)

このフィールドでは、EOS バイトで行う比較の種類を指定します。ASCII または ISO(国際標準化機構) フォーマットにあるように、8 ビットすべてを比較するか、下位の 7 ビットだけを比較するかを選択します。この特性のデフォルトのオプションは 7-bit です。

メモ　このフィールドは、`Set EOI with EOS on Write` フィールドまたは `Terminate Read on EOS` フィールドに yes を設定した場合にのみ意味があります。

EOS Byte(EOS バイト)

デバイスによっては、選択された文字が検出されたら読み取り処理を終了するようにプログラムできるものがあります。改行文字 (16 進 "0A") が一般的な EOS バイトです。この特性のデフォルトのオプションは 00H です。

メモ　ドライバが、書き込み処理でデータ文字列の終わりに自動的に EOS を付けることはありません。データ文字列に明示的にこのバイトを入れる必要があります。EOS バイトは、ドライバに読み取り処理を正しく終了させるためにのみ使用します。

Send EOI at End of Write(書き込みの終わりに EOI を送信)

リスナのデバイスの中には、トーカに最終バイトで EOI 線をアサートしてデータメッセージを終了するよう要求するものがあります。`yes` を設定すると、GPIB ボードは最終データバイトで EOI 線をアサートします。この特性のデフォルトのオプションは `yes` です。

System Controller(システムコントローラ、ボード特性のみ)

このフィールドはボード特性の画面にのみ表示されます。GPIB システムのシステムコントローラは、バスの制御を行う最高位のデバイスです。GPIB でリンクされているコンピュータネットワークなどの場合、他のデバイスをシステムコントローラにすることもできます。このような場合、このフィールドに `no` を設定して GPIB ボードのシステムコントローラの資格を取り消す必要があります。`yes` を設定すると、システムコントローラの資格が与えられます。通常、GPIB ボードがシステムコントローラとして指定されています。この特性のデフォルトのオプションは `yes` です。

メモ　GPIB システムで複数のシステムコントローラを指定することはできません。

Assert REN when SC(SC 時の REN のアサート、ボード特性のみ)

このフィールドに yes を設定すると、ボードがシステムコントローラのときにそのボードがオンラインになると、自動的に Remote Enable(REN) がアサートされます。`no` を設定すると、REN をアサートする場合、明示的に `ibsre` を呼び出す必要があります。この特性のデフォルトのオプションは `no` です。

Enable Auto Serial Polling(自動シリアルポーリングのイネーブル、ボード特性のみ)

このオプションで、GPIB Service Request(SRQ) 線がアサートされたときに、デバイスの自動シリアルポールをイネーブルまたはディスエーブルします。ポール応答があると、呼び出し後保存され、ibrsp デバイス関数で読み取ることができます。通常、この機能は IEEE 488.1 仕様に準拠したデバイスと矛盾しません。矛盾する場合は、このフィールドに no を設定して、この機能をディスエーブルしてください。この特性のデフォルトのオプションは yes です。

Enable CIC Protocol(CIC プロトコルのイネーブル、ボード特性のみ)

制御資格を他のデバイスに渡した後でデバイスレベルの NI-488 関数が呼び出された場合、このプロトコルをイネーブルされていると、ボードはシリアルポールステータスバイトに 16 進 "42" を設定して SRQ をアサートします。現在のコントローラはボードが制御資格を必要としていることを認識することになります。現在のコントローラが制御資格をボードに返すと、デバイス呼び出しは通常どおり実行されます。制御資格がタイムアウト時間内に受け渡されない場合、または CIC プロトコルがディスエーブルされている場合は、ECIC エラーが返されます。この特性のデフォルトのオプションは no です。

Bus Timing(バスタイミング、ボード特性のみ)

このフィールドには、ボードのソースハンドシェイク機能の T1 遅延を指定します。この遅延で、データがバスに送られてから、ボードが書き込みまたはコマンド処理中に DAV をアサートするまでの最少時間を決定します。システムの GPIB ケーブルの全長が 15 メートル未満の場合は、350ns が適当です。

他にも T1 遅延の選択に影響する要因はありますが、システムのセットアップに影響することはないでしょう。これらの要因について詳しくは、ANSI/IEEE 規格 488.1-1987 の 5.2 節を参照してください。このオプションのデフォルトは 500ns です。

Cable Length for High Speed(高速用ケーブル長、ボード特性のみ)

このフィールドには、システムで使用する GPIB ケーブルの長さをメートルで指定します。HS488 高速プロトコルを使用して HS488 準拠のデバイスと通信している場合は、システムの GPIB ケーブルの全長を指定します。システムコントローラは、高速転送がエラーなく行われるよう、

GPIB の初期化時にこの情報を HS488 デバイスに送信する必要があります。

Parallel Poll Duration(パラレルポールの間隔、ボード特性のみ)

このフィールドには、パラレルポール実行時のデバイスの待ち時間を指定します。通常のバス構成 (コントローラとデバイスが同じバス上にある) では、デフォルトの 2μs を使用します。表示なしのパラレルポールモードで GPIB バスエクステンダを使用している場合、バスエクステンダがアプリケーションに影響を与えずに動作できるよう、ポール間隔を 10μs に延長する必要があります。

Use This GPIB Interface(この GPIB インタフェースを使用、ボード特性のみ)

指定されたベースアドレスのボードにドライバがアクセスしないようにするには (システムにボードがない時など)、このオプションに `no` を設定します。このフィールドに `no` を設定しておけば、プログラムがボードにアクセスしようとすると、ドライバは EDVR エラーを返します。デフォルトでは、アクセスボード `gpib0` はイネーブルされ、`gpib1`, `gpib2`, `gpib3` はディスエーブルされています。

Base I/O Address(ベース I/O アドレス、ボード特性のみ)

このフィールドには、GPIB ボードの I/O アドレスを指定します。GPIB ボード自身のベース I/O アドレスと同じ値を設定してください。ベース I/O アドレスレベルの設定については、GPIB インタフェースボードの「入門マニュアル」に説明があります。

メモ　システムによっては、このフィールドは読み取り専用で、`ibconf` で変更することができません。I/O アドレスの変更方法については、「入門マニュアル」を参照してください。

DMA Channel(DMA チャネル、ボード特性のみ)

このフィールドには、GPIB ボードが使用する DMA チャネルを指定します。GPIB ボード自身の DMA チャネル (Micro Channel システムではアービトレーションレベル) と同じ値を設定してください。DMA チャネルの設定については、GPIB インタフェースボードの「入門マニュアル」に説明があります。

メモ　システムによっては、`ibconf` ユーティリティで変更できるのは、DMA のイネーブルとディスエーブルのみで、DMA チャネルの設定は変更できません。DMA チャネルの変更方法については、「入門マニュアル」を参照してください。

Interrupt Jumper Setting(割り込みジャンパの設定、ボード特性のみ)

このフィールドには、GPIB ボードが使用する割り込み線を設定します。GPIB ボード自身の割り込みレベルと同じ値を設定してください。割り込みレベルの設定については、GPIB インタフェースボードの「入門マニュアル」に説明があります。

メモ　システムによっては、ibconf ユーティリティで変更できるのは、割り込みのイネーブルとディスエーブルのみで、IRQ 要求線の設定は変更できません。割り込みレベルの変更方法については、「入門マニュアル」を参照してください。

Serial Poll Timeout(シリアルポールタイムアウト、デバイス特性のみ)

このタイムアウト値で、ドライバがデバイスからのシリアルポート応答を待つ時間を制御します。IEEE 488.1 仕様では、コントローラが応答バイトを待つ時間は指定されていません。ほとんどのデバイスでは、デフォルト値の 1 秒で十分です。シリアルポールに問題がある場合は、タイムアウトを長めに設定してください。

Enable Repeat Addressing(アドレス指定の繰り返しイネーブル、デバイス特性のみ)

通常、各書き込みまたは読み取り処理ごとに、デバイスがアドレスされることはありません。no を選択すると、選択したデバイスで同じ処理が実行されたばかりのときは、読み取りまたは書き込み処理の際にそのデバイスは再アドレスされません。再アドレス指定を避けることによって、いくつかの GPIB 処理を実行する場合には時間を節約することができます。ただし、入出力処理ごとに GPIB アドレスを送信しなければならない古い IEEE 488.1 デバイスでは問題があるかもしれません。その場合は、yes を選択してアドレス指定の繰り返しをイネーブルしてください。この特性のデフォルトのオプションは no です。

GPIB-PCII/IIA Mode Switch(GPIB-PCII/IIA モード切替え)

GPIB-PCII と GPIB-PCIIA のインタフェースボードキットのドライバは同じなので、両方のボードに使用することができます。このフィールドを使用して、システムにインストールされているボードの種類を指定してください。システムには GPIB-PCII と GPIB-PCIIA の両方のインタフェースボードを同時にインストールすることができます。

ibconf のデフォルト構成

この節では、NI-488.2 ドライバのデフォルトの構成値を示します。

- デバイス 32 台 (シンボリック名 dev1 ～ dev32)。
- アクセスボード 4 枚 (シンボリック名 gpib0, gpib1, gpib2, gpib3)。アクセスボード名は変更できません。
- アクセスボード gpib0 はイネーブル。gpib1, gpib2, gpib3 はディスエーブル。
- 最初の 16 台のデバイスの GPIB アドレスはデバイス番号と同じ。たとえば、dev1 はアドレス 1 です。これらの 16 台のデバイスはアクセスボード gpib0 に割り当てられています。
- 残りの 16 台のデバイス (デバイス 17 から 32) はアクセスボード gpib1 に割り当てられています。これらの GPIB アドレスはそれぞれ 1 から 16 です。たとえば、dev17 はアドレス 1 にあります。
- 各 GPIB インタフェースボードは、個々のバスのシステムコントローラで、GPIB アドレスは 0 です。
- デバイスへの各データメッセージの最終バイトとともに END メッセージが送信されます。文字列の終わり (EOS) 文字は認識されません。
- 入出力および待ち関数呼び出しのタイムアウト値は 10 秒に設定されています。
- 各 GPIB ボードとデバイスは入出力転送を DMA を使用して実行するように設定されています。
- 自動シリアルポールがイネーブルされています。
- 各 NI-488.2 ルーチンが終わると、NI-488.2 ドライバはバスを現在アドレスされている状態に保ちます (IEEE 488.2 規格)。

ibconf の終了

すべての変更を行ったら、<F9> または <Esc> を押して ibconf を終了します。プログラムは、ドライバのディスクコピーに変更を保存するかどうかを聞いてきます。変更を保存する場合は yes を、無視する場合は no を、ibconf を続ける場合は cancel を選択します。システムにドライバがロードされていれば、ibconf は現在ロードされているドライバに変更を保存するかどうかを聞いてきます。ロードされているドライバを変更する場合は yes を、変更せずに終了する場合は no を、ibconf を続ける場合は cancel を選択します。ibconf がロードされているドライバを更新できない場合、ibconf はその理由を表示し、コンピュータを再起動するよう促す最終プロンプトを表示します。

エラーのチェック

エキスパートモードで起動していなければ、ibconf は終了する前に問題が発生していないかチェックし、以下の状況を知らせます。

- デバイスとそのアクセスボードの GPIB アドレス指定が一致していない。
- 指定されたアドレスのホストマシンに GPIB ボードがない。
- デバイスまたはボードでタイムアウトがディスエーブルされている。

ibconf の起動時に自動チェックをディスエーブルするときは、-e オプションを使用します。

ロードされたドライバへの変更の保存

ロードされたバージョンと編集したバージョンに互換性がある場合、ibconf を使用してメモリにロードされたドライバを変更することができます。ロードされたバージョンと互換で、ibconf を -f または -d オプションで起動していない場合は、ロードされたドライバを変更するかどうかを聞くプロンプトが表示されます。y を入力すると、現在ロードされたドライバが、選択したパラメータで変更されます。ibconf がロードされたドライバを見つけられない場合、プロンプトは表示されません。ibconf を -d オプションで起動すると、プロンプトは表示されず、ドライバが互換ならばロードされたバージョンが自動的に構成されます。-f オプションで ibconf を起動すると、ロードされたドライバは変更されません。

ボードの割り込みレベルを変更したり、デバイス名が lpt1, lpt2, lpt3, com1, または com2 に変更されていると、ibconf はロードされたドライバを構成しません。変更を加えて保存してもロードされたドライバを変更しなければ、コンピュータを再起動しないかぎり変更は有効になりません。

ibconf のバッチモード

ibconf のバッチモードは、NI-488.2 ドライバの構成を変更するもう 1 つの方法です。バッチモードでは、ロードされたドライバも変更することができます。

バッチモードでは、構成情報はユーザが作成した構成ファイルに入っています。作成した構成ファイルを使用するには、次のコマンドを入力します。

```
ibconf -b filename
```

ここで、***filename*** は構成ファイルの名前です。*-b* と *filename* の間に少なくともスペースを 1 つ空けてください。

構成ファイルは、いくつかのペアになった項目が入った自由形式のテキストファイルです。各項目は少なくとも 1 つのスペースまたは復帰改行文

字で区切ります。ペアの最初の項目は、構成するボードまたはデバイスの特性あるいはデバイスマップの構成関数 (rename, connect, disconnect など) を表すニモニックです。2 つめの項目は最初の項目 (ニモニック) に対して設定する値です。

ボードやデバイスを構成する前に、そのボードまたはデバイスを find name# で指定しなければなりません。ここで name は board または device, # は構成する GPIB ボードまたはデバイスのインデックスです。

バッチモードで ibconf を実行すると、ibconf はペアの最初の項目の構文と 2 番目の項目の値の範囲をチェックし、エラーがあった場合はそれを返します。ibconf が値の範囲に関してエラーを発見したときは、正しい範囲を表示します。エラーが 1 つでも見つかると、ドライバは構成されません。

以下に、構成ファイルのサンプルを示し、各項目について説明します。

```
find device1 name plotter
find device2 disconnect
find device3 connect board1
find board0 pad 2 tmo 30sec eos 0x1E sc no
                bin 8-bit tmng 350nsec
```

この例では、以下のような変更を行っています。

1. 1 番目のデバイスの名前を plotter に変更。
2. 2 番目のデバイスを非接続。
3. 3 番目のデバイスをボード 1(gpib1) に接続。
4. ボード 0(gpib0) の構成を変更。
   - 1 次アドレスを 2 に変更 (pad 2)。
   - タイムアウトの設定を 30s に変更 (tmo 30sec)。
   - EOS バイトを 16 進 "1E" に変更 (eos 0x1E)。
   - システムコントローラ資格を NO に変更 (sc no)。
   - EOS での比較のタイプを 8-bit に変更 (bin 8-bit)。
   - GPIB バスタイミングを 350ns に設定 (tmng 350nsec)。

表 8-2 に、有効な項目のペアを示します。数値には、10 進値または 16 進値を入力できます。16 進数の前には 0x を付けてください。たとえば、10 進数 64 は 0x40 となります。有効な入力値について詳しくは、「DOS/Windows 用 NI-488.2 関数リファレンスマニュアル」の該当する関数の説明を参照してください。

表 8-2　ibconf のバッチモードのコマンドペア

ニモニック	第 1 項目の説明	第 2 項目
find	ボードまたはデバイスを見つける	board# または device#
pad	1 次 GPIB アドレス	数値
sad	2 次 GPIB アドレス	数値
tmo	タイムアウト設定	ニモニック
xeos	書き込みで EOS とともに EOI を設定	yes または no
bin	EOS での比較のタイプ	7-bit または 8-bit
eot	書き込みの最終バイトに EOI を設定	yes または no
sc	システムコントローラ (ボードのみ)	yes または no
sre	SC 時の REN のアサート (ボードのみ)	yes または no
spoll	自動シリアルポーリングのイネーブル (ボードのみ)	yes または no
tmng	タイミング (ボードのみ)	2msec, 500nsec または 350nsec
cic_prot	CIC プロトコル (ボードのみ)	yes または no
int	割り込み設定 (ボードのみ)	数値
port	ベース I/O アドレス (ボードのみ)	数値
dma	DMA チャネル (ボードのみ)	数値
raddr	アドレス設定の繰り返し (デバイスのみ)	yes または no
name	デバイスの名称変更 (デバイスのみ)	デバイス名
connect	デバイスをボードに接続 (デバイスのみ)	board#
disconnect	デバイスをボードから非接続 (デバイスのみ)	値なし
type	現在のボードを PCII または PCIIA モードに切替え	PCII または PCIIA
pplength	パラレルポールの間隔 (ボードのみ)	ニモニック

表 8-2　ibconf のバッチモードのコマンドペア（続き）

ニモニック	第 1 項目の説明	第 2 項目
useboard	このインタフェースボードを使用 (ボードのみ)	yes または no
spolltmo	シリアルポールのタイムアウト (デバイスのみ)	ニモニック

A

ステータスワード状況

この付録では、ステータスワード ibsta に返される状態について詳しく説明します。

ご使用のアプリケーションプログラムで ibsta を使用する方法については、第 3 章「アプリケーションの開発」を参照してください。

関数呼び出しが ENEB または EDVR エラーを返してきたときは、ERR ビット以外のすべてのステータスワードビットはクリアされるため、GPIB ボードのステータスを知ることはできません。

ibsta の各ビットは、デバイス呼び出し (dev)、ボード呼び出し (brd)、またはその両方 (dev, brd) に設定できます。

以下に、ステータスワードビットを示します。

ニモニック	ビット位置	16 進数値	タイプ	説明
ERR	15	8000	dev, brd	GPIB エラー
TIMO	14	4000	dev, brd	タイムリミット超過
END	13	2000	dev, brd	END または EOS 検出
SRQI	12	1000	brd	SRQ 割り込み受信
RQS	11	800	dev	デバイスがサービスを要求
SPOLL	10	400	brd	ボードがコントローラによってシリアルポールされました
EVENT	9	200	brd	DCAS, DTAS、または IFC イベント発生
CMPL	8	100	dev, brd	入出力完了
LOK	7	80	brd	ロックアウト状態
REM	6	40	brd	リモート状態
CIC	5	20	brd	コントローラインチャージ
ATN	4	10	brd	アテンションがアサートされています
TACS	3	8	brd	トーカ
LACS	2	4	brd	リスナ
DTAS	1	2	brd	デバイストリガ状態
DCAS	0	1	brd	デバイスクリア状態

ERR(dev, brd)

呼び出しがエラーに終わると、ステータスワードには ERR が設定されます。エラー変数 `iberr` を調べて、エラーの種類を確認することができます。付録 B「エラーコードと対処」では、`iberr` に記録されるエラーコードについて説明し、さらにどのように対処すればよいか解説しています。ERR は、呼び出しが成功するとクリアされます。

TIMO(dev, brd)

TIMO は、タイムアウト時間を超過したことを示します。ibwait マスクパラメータの TIMO ビットが設定されているときにタイムリミットを超過すると、ibwait 呼び出しの後のステータスワードには TIMO が設定されます。また、同期入出力関数 (ibcmd, ibrd, ibwrt, Receive, Send, SendCmds など) の呼び出し中にタイムアウトが発生しても TIMO が設定されます。TIMO はそれ以外の場合はクリアされます。

END(dev, brd)

END は、GPIB EOI 線がアサートされたか、ソフトウェアが EOS バイトを受け取ったら読み取りを終了するように構成されている場合、EOS バイトが受信されたことを示します。GPIB ボードが ibgts 関数の結果シャドウハンドシェイキングを実行しており、他の関数の呼び出し前あるいはその間に END 状態が発生した場合、その関数はステータスワードに END ビットを立てて返します。END は、入出力操作を開始するとクリアされます。

アプリケーションによっては、読み取り操作の正確な入出力読み取り終了モード、つまり EOI のみ、EOS 文字のみ、または EOI と EOS 文字、を知る必要があるものもあります。ibconfig 関数 (オプション IbcEndBitIsNormal) を使用して、EOI がアサートされたときのみ END ビットを設定するモードをイネーブルすることができます。このモードでは、EOS 文字のみにより入出力操作が終了した場合は、END は設定されません。アプリケーションは、受信したバッファの最終バイトが EOS 文字かどうかをチェックする必要があります。

SRQI(brd)

SRQI は、GPIB デバイスがサービスを要求していることを示します。GPIB ボードが CIC で、GPIB SRQ 線がアサートされ、自動シリアルポール機能がディスエーブルされていると SRQI が設定されます。GPIB ボードが CIC でなくなるか GPIB SRQ 線がアサート解除されると SRQI はクリアされます。

RQS(dev)

RQS はデバイスレベル呼び出しの後にのみステータスワードに現れ、デバイスがサービスを要求していることを示します。デバイスのシリアルポールステータスバイトでビット 6 がアサートされると RQS が設定されます。シリアルポールは、ibrsp 呼び出しによって、または自動シリア

ルポーリングがイネーブルされている場合は自動ポーリングによってステータスビットを獲得します。シリアルポールに応答しないデバイスには RQS で ibwait を発行しないでください。ibrsp が RQS を設定したシリアルポールステータスバイトを読み取ると RQS はクリアされます。

SPOLL(brd)

コントローラがいつ GPIB ボードをシリアルポールしたかを確認するには、トーカ / リスナアプリケーションで SPOLL を使用してください。SPOLL ビットは、デフォルトではディスエーブルされています。イネーブルするときは ibconfig 関数 (オプション IbcSPollBit) を使用してください。このビットをイネーブルすると、ボードがポールされ、かつその時点でそのボードがサービスを要求している場合のみ SPOLL ビットが設定されます。SPOLL ビットがウェイトマスクに設定されている場合は、ibwait 呼び出しのすぐ後の呼び出しで、そうでない場合は ibrsv 呼び出しの直後にクリアされます。

EVENT(brd)

GPIB デバイスクリア、グループ実行トリガおよびインタフェースクリア送信のコマンドの順序をモニタするには、トーカ / リスナアプリケーション (GPIB インタフェースがコントローラでないアプリケーション) で EVENT を使用してください。ibsta の通常の DCAS および DTAS ビットでは不十分な場合があります。

EVENT ビットは、デフォルトではディスエーブルされています。このビットを使用する場合は、ibconfig 関数 (オプション IbcEventQueue) を使用して、EVENT をイネーブルしてください。このビットをイネーブルすると、DCAS と DTAS ビットはディスエーブルされます。イベントが発生すると、このビットが設定され、進行中の入出力は中止されます。アプリケーションは ibevent 関数を呼び出して、イベントの内容を確認することができます。

CMPL(dev, brd)

CMPL は入出力操作の状況を示し、入出力操作が完了すると設定されます。入出力操作中は CMPL はクリアされています。

LOK(brd)

LOK は、ボードがロックアウト状態にあるかどうかを示します。LOK が設定されている間は、そのボードでは EnableLocal ルーチンや ibloc 関数は使用できません。GPIB ボードが、GPIB ボードまたは他のコントローラによって Local Lockout(LLO) メッセージが送信されたことを検出すると LOK が設定されます。システムコントローラが Remote Enable(REN)GPIB 線をアサート解除すると、LOK はクリアされます。

REM(brd)

REM は、ボードがリモート状態にあるかどうかを示します。Remote Enable(REN)GPIB 線がアサートされ、GPIB ボードが、GPIB ボードまたは他のコントローラによって自分のリスンアドレスが送信されたことを検出すると、REM が設定されます。REM は以下の状況でクリアされます。

- REN がアサート解除された場合。
- リスナの GPIB ボードが、GPIB ボードまたは他のコントローラによって Go to Local(GTL) コマンドが送信されたことを検出した場合。
- ステータスワードで LOK ビットがクリアされているときに、ibloc 関数が呼び出された場合。

CIC(brd)

CIC は、GPIB ボードがコントローラインチャージであるかどうかを示します。GPIB ボードがシステムコントローラであるとき、または他のコントローラが GPIB ボードに制御資格を渡したときに、SendIFC ルーチンまたは ibsic 関数を実行すると、CIC が設定されます。GPIB ボードがシステムコントローラからの Interface Clear(IFC) を検出するか、GPIB ボードが制御資格を他のデバイスに渡すと CIC はクリアされます。

ATN(brd)

ATN は GPIB Attention(ATN) 線の状態を示します。ATN は、GPIB ATN 線がアサートされると設定され、ATN 線がアサート解除されるとクリアされます。

TACS(brd)

TACS は、GPIB ボードがトーカとしてアドレスされているかどうかを示します。GPIB ボードが、GPIB ボード自身または他のコントローラによって自分のトークアドレス (および該当する場合は 2 次アドレス) が送信されたことを検出すると、TACS が設定されます。GPIB ボードが Untalk(UNT) コマンド、自分自身のリスンアドレス、自分以外のトークアドレスまたは Interface Clear(IFC) を検出すると、TACS はクリアされます。

LACS(brd)

LACS は、GPIB ボードがリスナとしてアドレスされているかどうかを示します。GPIB ボードが、GPIB ボード自身または他のコントローラによって自分のリスンアドレス (および該当する場合は 2 次アドレス) が送信されたことを検出すると、LACS が設定されます。GPIB ボードが `ibgts` 関数の結果シャドウハンドシェイキングを行う場合にも、LACS が設定されます。GPIB ボードが Unlisten(UNL) コマンド、自分自身のトークアドレス、Interface Clear(IFC) を検出した場合、または `ibgts` 関数がシャドウハンドシェイキングなしで呼び出された場合、LACS はクリアされます。

DTAS(brd)

DTAS は、GPIB ボードがデバイストリガコマンドを検出したかどうかを示します。リスナの GPIB ボードが、他のコントローラによって Group Execute Trigger(GET) コマンドが送信されたことを検出すると、DTAS が設定されます。DTAS ビットが `ibwait` マスクパラメータに設定されている場合、このビットは `ibwait` 呼び出しの直後の呼び出しでクリアされます。

DCAS(brd)

DCAS は、GPIB ボードがデバイスクリアコマンドを検出したかどうかを示します。GPIB ボードが、他のコントローラによって Device Clear(DCL) コマンドが送信されたことを検出した場合、またはリスナの GPIB ボードが、他のコントローラによって Selected Device Clear(SDC) コマンドが送信されたことを検出すると、DCAS が設定されます。DCAS ビットが `ibwait` マスクパラメータに設定されている場合、このビットは `ibwait` 呼び出しの直後の呼び出しでクリアされます。また、読み取り書き込みの直後の呼び出しでもクリアされます。

B

エラーコードと対処

この付録では、各エラーの意味と原因、およびどのように対処すればよいかについて説明します。

次の表に、GPIB エラーコードをリストします。

エラーのニモニック	iberr 値	意味
EDVR	0	DOS エラー
ECIC	1	この関数では、GPIB ボードが CIC であることが必要です
ENOL	2	GPIB 上にリスナが存在しません
EADR	3	GPIB ボードへのアドレスが間違っています
EARG	4	関数呼び出しに無効な引数が使用されました
ESAC	5	GPIB ボードが要求されたシステムコントローラになっていません
EABO	6	入出力処理が打ち切られました (タイムアウト)
ENEB	7	存在しない GPIB ボードです
EOIP	10	非同期入出力が進行中です
ECAP	11	処理を行う資格がありません
EFSO	12	ファイルシステムエラー
EBUS	14	GPIB バスエラー
ESTB	15	シリアルポールステータスバイト待ち行列オーバーフロー
ESRQ	16	SRQ アサートされた状態から動きません
ETAB	20	表の問題

EDVR (0)

ibfind に受け渡されたボード名またはデバイス名がソフトウェアで構成されていないと、EDVR が返されます。この場合、変数 ibcntl には DOS エラーコード 2「デバイスが見つかりません」または 110「開けませんでした」が入ります。また、関数呼び出しに無効なユニット記述子が受け渡されたときも EDVR が返されます。この場合、変数 ibcntl には DOS エラーコード 6「無効な操作です」が入ります。

ドライバ (gpib.com) がインストールされていない場合にも EDVR が返されます。

対処

- ibdev を使用して、シンボリック名を指定せずにデバイスを開きます。
- ibfind 関数のパラメータとして、ユーティリティプログラム ibconf で構成されているデバイス名またはボード名のみを使用します。
- 後続の NI-488 関数の第 1 パラメータとして ibfind 関数で返されたユニット記述子を使用します。ibfind の後と失敗した関数の前の変数を調べ、変数が破壊されていないことを確認してください。
- ルートディレクトリの config.sys ファイルをチェックして、NI-488.2 ドライバがインストールされていることを確認します。次の行があるかどうか確かめてください。

  ```
  device=dir\gpib.com
  ```

  ここで、*dir* は gpib.com が入っているディレクトリです。

ECIC (1)

ボードが CIC でないときに以下のボード関数またはルーチンを呼び出すと、ECIC が返されます。

- GPIB に影響するデバイスレベルの NI-488 関数
- ibcmd, ibcmda, ibln, ibrpp などの GPIB コマンドバイトを発行するボードレベルの NI-488 関数
- ibcac, ibgts
- SendCmds, PPoll, Send, Receive などの GPIB コマンドバイトを発行する NI-488.2 ルーチン

対処

- ibsic または SendIFC を使用して、GPIB ボードを GPIB の CIC にします。
- "ibrsc 1" を使用して、ユーザの GPIB ボードがシステムコントローラであることを確認します。
- CIC が複数ある場合、呼び出しを試みる前にステータスワード ibsta に CIC ビットがあることを必ず確認してください。CIC ビットがない場合、(CIC への)ibwait 呼び出しを実行して、制御資格がボードに受け渡されるまで処理を遅らせることができます。

ENOL (2)

ENOL は通常、リスナをアドレスせずに書き込み操作を試みた場合に発生します。デバイス書き込みの場合、このエラーは、ソフトウェアでそのデバイスに構成された GPIB アドレスが、バスに接続されているどのデバイスの GPIB アドレスとも一致しない、GPIB ケーブルがデバイスに接続されていない、またはデバイスの電源が入っていないことを示します。

GPIB ボードが CIC ではなく、進行中の書き込み呼び出しが終了する前にコントローラが ATN をアサートした場合にも ENOL が発生することがあります。

対処

- デバイスの GPIB アドレスが、データを書き込むデバイスの GPIB アドレスと一致していることを確認します。
- ibcmd で適切な 16 進コードを使用して、デバイスをアドレスします。
- ケーブル接続をチェックして少なくとも 3 分の 2 のデバイスの電源が ON になっていることを確認してください。
- ibpad(または必要な場合は ibsad) を呼び出して、構成されたアドレスをデバイスのスイッチ設定に合わせます。
- 書き込みバイト数をコントローラが期待する値まで減らします。

EADR (3)

EADR は、GPIB ボードが CIC で、読み取りおよび書き込み関数の前に正しくアドレスされていない場合に発生します。このエラーは通常ボードレベル関数に関連しています。

シャドウハンドシェイク機能が要求され、GPIB ATN 線がすでにアサート解除されている場合にも、関数 ibgts によって EADR が返されます。この場合、シャドウハンドシェイクは不可能であり、それを通知するためにエラーが返されます。

対処

- ibrd, ibwrt, RcvRespMsg または SendDataBytes を呼び出す前に GPIB ボードが正しくアドレスされていることを確認してください。
- ibcmd 呼び出しの直後を除いて、ibgts の呼び出しを避けます (ibcmd によって ATN がアサートされます)。

EARG (4)

EARG は、無効な引数が関数呼び出しに受け渡された場合に発生します。以下にその例を示します。

- 0 ～ 17 の範囲外の値を使用して ibtmo が呼び出された。
- 無効なアドレスを使用して ibpad または ibsad が呼び出された。
- 無効なパラレルポール構成を使用して ibppc が呼び出された。
- 有効なデバイス記述子を使用してボードレベル NI-488 呼び出しが、またはボード記述子を使ってデバイスレベル NI-488 呼び出しが実行された。
- 無効なアドレスを使用して NI-488.2 ルーチンが呼び出された。
- 無効なデータ線またはセンスビットを使用して PPollConfig が呼び出された。

対処

- NI-488 関数または NI-488.2 ルーチンに渡されたパラメータが有効であることを確認してください。
- ボード関数にデバイス記述子を、デバイス関数にボード記述子を使用しないでください。

ESAC (5)

ESAC は、GPIB ボードがシステムコントローラの機能をもっていないときに ibsic, ibsre, SendIFC または EnableRemote が呼び出された場合に発生します。

対処

"ibrsc 1" を呼び出して、または ibconf を使用してソフトウェアを構成することによって、GPIB ボードにシステムコントローラの機能を与えます。

EABO (6)

EABO は、通常タイムアウトによって入出力処理がキャンセルされたことを示します。また、入出力処理中に ibstop を呼び出したり、CIC から Device Clear メッセージを受け取った場合にもこのエラーが発生します。

時々、入出力が進行しなくなったり (リスナがハンドシェイクしなくなる、またはトーカが送信をやめる)、タイムアウトになった呼び出しのバイト数が他方のデバイスが予期した値より大きくなっています。

対処

- 入力関数に正しいバイト数を使用するか、トーカに転送の終わりを示す END メッセージを使用させます。
- ibtmo を使用して、入出力処理のタイムアウト時間を延長します。
- データを要求する前に、そのデバイスがデータを送信するように構成されていることを確認してください。

ENEB (7)

ENEB は、構成プログラムで指定された入出力アドレスに GPIB ボードが存在しないときに発生します。ボードが実際にシステムに接続されていない場合、構成時に指定された入出力アドレスがボードの実際の設定と一致していない場合、ベース I/O アドレスにシステム上の矛盾がある場合、または ibconf の Use This GPIB Interface(この GPIB インタフェースを使用) フィールドの設定が正しくない場合に、このエラーが発生します。

対処

- システムに、有効なベース I/O アドレスを使用してハードウェアとソフトウェアの両方で正しく構成された GPIB ボードがあることを確認してください。
- ibconf の Use This GPIB Interface フィールドが yes に設定されていることを確認してください。

EOIP (10)

EOIP は、他の呼び出しが実行される前に、非同期入出力処理が終了しなかった場合に発生します。非同期入出力中は、ibstop, ibwait, ibonl または非 GPIB 処理しか実行できません。非同期入出力が始まると、ibstop, ibwait, または ibonl 以外の GPIB 呼び出しは厳しく制限されます。呼び出しによって進行中の入出力処理が妨害される可能性がある場合、ドライバは EOIP を返します。

対処

GPIB 呼び出しを実行する場合は、まずドライバとアプリケーションを再同期化してください。以下の 3 つの関数のいずれかを使用して再同期化を行います。

- ibwait　返された ibsta に CMPL がある場合、ドライバとアプリケーションは再同期化されます。
- ibstop　入出力はキャンセルされ、ドライバとアプリケーションが再同期化されます。
- ibonl　入出力はキャンセル、インタフェースはリセットされ、ドライバとアプリケーションが再同期化されます。

ECAP (11)

ECAP は、GPIB ボードに処理を実行する能力がない場合、またはソフトウェアで特定の機能がディスエーブルされているときに、その機能が必要な呼び出しが実行されると発生します。

対処

呼び出しの妥当性をチェックするか、GPIB インタフェースボードとドライバの両方が必要な機能を備えていることを確認してください。

EFSO (12)

ibrdf または ibwrtf 呼び出しで、ファイル処理の実行に問題が発生すると、EFSO が報告されます。このエラーは、特に関数がアクセスしたファイルのオープン、作成、シーク、書き込み、クローズができないことを示します。この状態を示す DOS エラーコードが ibcntl に入っています。

対処

- 指定したファイル名、パス、ドライブが正しいことを確認してください。
- ファイルのアクセスモードが正しいことを確認してください。
- ディスクにファイルを記憶するだけの容量があることを確認してください。

EBUS(14)

デバイス関数の実行中に一定の GPIB バスエラーが発生すると、EBUS が報告されます。すべてのデバイス関数はコマンドバイトを送信して、アドレッシングや他のバス管理を実行します。デバイスは、デフォルト構成または `ibtmo` 関数で指定された制限時間内にこれらのコマンドバイトを受け取るようになっています。これらのコマンドバイトの送信中にタイムアウトになると、EBUS が発生します。

対処

- 計測器が正しく動作していることを確認してください。
- ケーブルがゆるんだり損傷していないか、GPIB 上で電源の切れた計測器がないかをチェックしてください。
- タイムアウト時間が短くて、ドライバがコマンドバイトを送信できない場合は、タイムアウト時間を延長してください。

ESTB (15)

ESTB を返すのは、`ibrsp` 関数だけです。ESTB は、自動シリアルポールで受け取ったいくつかのシリアルポールステータスバイトが、記憶スペースの不足のために捨てられたことを示します。古いステータスバイトのいくつかは残っており、一番古いものが `ibrsp` 呼び出しで返されています。

対処

- `ibrsp` 呼び出しの回数を増やして、待ち行列を空にします。
- `ibconfig` 関数または `ibconf` ユーティリティで自動ポーリングをディスエーブルします。

ESRQ (16)

ESRQ は、ibwait 関数または WaitSRQ ルーチンの実行中にのみ発生します。ESRQ は、GPIB SRQ 線がアサートされた状態から動かないために RQS を待機できないことを示します。以下のイベントによってこのような状態が発生します。

- 通常、ソフトウェアが認識していないデバイスが SRQ をアサートしています。ソフトウェアはこのデバイスを知らないため、このデバイスをシリアルポールして SRQ をアサート解除することができません。
- GPIB バステスターまたは同等の装置が SRQ 線のアサートを強制しています。
- SRQ 線に関わるケーブルの問題が発生しています。

ESRQ は、GPIB に問題が発生していることを警告するものですが、この状態が続く限り RSQ ビットを使用できないことを除いて、GPIB の動作には影響しません。

対処

アプリケーションで使用していない他のデバイスが、SRQ をアサートしていないかチェックします。必要なら、それらを GPIB から切断します。

ETAB (20)

ETAB は、FindLstn, FindRQS および ibevent 関数の実行中にのみ発生します。ETAB は、これらの関数が使用する表に問題があることを示します。

- FindLstn の場合、ETAB は、所定の表にあるすべてのリスナのアドレスを記憶するだけの容量がないことを示します。
- FindRQS の場合、ETAB は、所定の表の中にサービスを要求しているデバイスがないことを示します。
- ibevent の場合、ETAB は、イベント待ち行列がオーバフローしてイベント情報が失われたことを示します。

対処

FindLstn の場合、結果配列のサイズを大きくします。FindRQS の場合、ユーザのアプリケーションで使用していない他のデバイスが SRQ をアサートしていないかどうかをチェックし、必要に応じて GPIB から切断します。ibevent から ETAB が返された場合は、ibevent を呼び出す回数を増やして待ち行列を空にしてください。

C

ユニバーサル言語インタフェース

この付録では、ユニバーサル言語インタフェース (ULI) のインストールおよび使用法について説明します。

概要

HP 式の呼び出しを使用する既存のプログラムをお持ちの場合、ユニバーサル言語インタフェース (ULI) を使用して NI-488.2 ドライバにアクセスすることができます。DOS ファイル呼び出しが、プログラミング言語の標準入出力コマンドを使って直接 ULI にアクセスするため、言語インタフェースはあまり必要ありません。ただし、ULI は NI-488.2 ドライバに比べると低性能のインタフェースです。

NI-488.2 言語インタフェースを使用すれば、高い性能と最大限の柔軟性が得られます。

ULI ドライバファイル `uli.com` は、GPIB インタフェースボードに含まれています。ULI ドライバは TSR(終了しても常駐する) ユーティリティで、DOS と NI-488.2 ドライバ `gpib.com` の間のインタフェースです。

ULI のインストール

ULI ソフトウェアをインストールする前に、GPIB ボードと NI-488.2 ソフトウェアをインストールし構成しておく必要があります。

ULI ソフトウェアをインストールするには、以下のステップに従ってください。

1. NI-488.2 ソフトウェアのインストール時に作成した、ULI サブディレクトリに移ります。
2. 次の `uli` コマンドを入力します。

   ```
   uli
   ```

ULI ドライバをメモリにロードした結果を示すメッセージがコンピュータ画面に現れます。ULI ドライバは、コンピュータの電源を切るか再起動するまでメモリに常駐します。

メモ ULI ドライバがロードされている間は、標準の NI-488 関数と NI-488.2 ルーチンは使用できません。

ULI シーケンス例

以下では、BASIC プログラムでの、汎用計測器で ULI を使用する方法について説明します。この計測器は 1 次アドレス 2 を使用し、2 次アドレスはありません。このプログラムで使われているテクニックは、ほとんどの計測器に応用できる一般的なものです。

システムの初期化

1. BASIC などのプログラミング言語から ULI ドライバとの通信を確立するには、OPEN 文を使用します。BASIC では、出力用と入力用に 2 つの OPEN 文が必要です。ULI ドライバは NI-488.2 ドライバ用に ibconf で定義されたボード名を使用します。この BASIC プログラムでは次の 2 つのコマンドを使用します。

```
100    OPEN "GPIB0" FOR OUTPUT AS #1
110    OPEN "GPIB0" FOR INPUT AS #2
```

2. 次に、Interface Clear(IFC) メッセージを送信してバスを初期化します。次のコマンドを入力します。

```
120    PRINT #1, "ABORT"
```

3. バスのステータスを調べるには、次のステータスコマンドを使用します。

```
130 PRINT #1, "STATUS"
140 INPUT #2, IBSTA%, IBERR%, IBCNT%
150 PRINT IBSTA%, IBERR%, IBCNT%
```

IBSTA% には、最後に実行したバス呼び出しのステータスが入っています。エラーが発生した場合、IBERR% にはエラーメッセージが入ります。IBCNT% には、バスで送信されたデータのバイト数が入ります。これらの変数については、第 3 章「アプリケーションの開発」に詳しい説明があります。

デバイスの構成

バスを初期化したら、デバイスを初期化します。

1. 計測器をリモートモードにするには、REMOTE コマンドを入力します。

```
160    PRINT #1, "REMOTE 2"
```

REMOTE の次の 2 は、リモートモードにするデバイスのアドレスです。

2. これで計測器はコマンドを受け取ることができます。OUTPUT コマンドでは特定のデバイス用のコマンドを送信します。たとえば、F1R0S2 を計測器に送信するときは次のようにします。

```
170  PRINT #1, "OUTPUT 2#6; F1R0S2"
```

OUTPUT コマンドは、指定されたデバイス (この場合、アドレス 2 のデバイス) に 6 バイトのデータ F1R0S2 を送信します。アドレスは、2 や 5 のような 1 次アドレスだけでも、0201 のように 2 次アドレスを含めてもかまいません。

読み取り

計測器が操作モードになったら、読み取りを行い、次のように表示することができます。

```
180    PRINT #1, "ENTER 2"
190    INPUT #2, RD$
200    PRINT RD$
```

ENTER コマンドにはアドレスを付け、これには 2 次アドレスも含めることができます。ENTER コマンドはドライバが計測器から読み取りを行うようプログラムします。INPUT 文が実行されると、データは BASIC の変数 RD$ に保存されます。BASIC では、最大 255 バイトのデータを文字列変数に読み込むことができます。デバイスが 255 バイトを超えて返すときは、複数の INPUT 文を使用して BASIC にデータを取り込みます。

計測器からの読み取りが終わったら、BASIC の文字列演算を使用してデータを処理することができます。

エラーの処理

BASIC では、ON ERROR GOTO コマンドによって IEEE 488 バスの操作中に発生したエラーを処理します。ON ERROR コマンドを使用して独自のエラー処理ルーチンを書く場合は、ULI 自動エラー検出をオフにして、アプリケーションモニタがインストールされていないことを確認してください。ULI 自動エラー検出は、次のコマンドでオフにすることができます。

```
10     PRINT #1, "ERRTRAP OFF"
```

エラーが発生したときに処理するためのサービスルーチンとコードを入れてください。次のコードセグメントを使用して ON ERROR を実行することができます。エラー値 24 は、GPIB エラーを示します。他のすべてのエラー値は非 GPIB エラーを表します。

```
50     ON ERROR GOTO 4000

4000   'SERVICE ROUTINE FOR ERRORS
4010   IF ERR <> 24 THEN GOTO 4090
4020   ELSE CLOSE
4030   OPEN "GPIB0" FOR INPUT AS #1
```

```
4040   OPEN "GPIB0" FOR OUTPUT AS #2
4050   PRINT #1, "STATUS"
4060   INPUT #2, IBSTA%, IBERR%, IBCNT%
4070   PRINT IBSTA%, IBERR%, IBCNT%
4080   RESUME NEXT
4090   PRINT "NON-GPIB ERROR"
5000   RESUME NEXT
```

データ転送の終了

GPIB のデータ転送は 2 つの部分に分かれます。各部分にはターミネータを付けて、転送の終了を示す必要があります。1 つは、ユーザのアプリケーションプログラムと NI-488.2 ドライバの間で行われ、言語ターミネータが使用されます。転送のもう一方の部分は、ドライバと GPIB デバイスの間で行われ、GPIB ターミネータを使用します。ユーザは ULI を使用して自分の転送ターミネータを構成し、プログラミング言語とデバイスの条件に合わせることができます。以下に、各ターミネータと言語アプリケーションでの役割について説明します。

OUTPUT ターミネータ

言語ターミネータ

アプリケーションプログラムから入出力コマンドを受け取るとき、ULI は言語ターミネータを使用して、文字列の終わりを認識します。BASIC では、入出力呼び出しで言語ターミネータを送信するか、出力文字列に言語ターミネータを付けるかのいずれかの方法で、すべての文字列と数字を終了することができます。

言語ターミネータの長さは、1 文字または 2 文字です。デフォルトの言語ターミネータ `CR LF` はほとんどの言語で使用できます。これを変更する必要がある場合は、`langeos` 関数を使用してください。言語ターミネータをご使用のプログラミング言語で機能しない値に変更すると、アプリケーションはデータを正しく読み取らなかったり、データ転送を終了しなくなります。

GPIB ターミネータ

ULI がデバイスに入出力コマンドを渡すときに、言語ターミネータは GPIB ターミネータと置き換えられます。言語ターミネータと同じ文字をデバイスに送信するには、GPIB ターミネータを言語ターミネータと同じ値に設定します。たとえば、CR LF をデバイスに送信するときは、次の関数呼び出しを使用してください。

```
PRINT #1, "GPIBEOS OUT CR LF"
```

GPIB ターミネータのデフォルトの設定はディスエーブルです。つまり、終了文字は送信されないということです。

図 C-1 に、言語ターミネータを指定された GPIB ターミネータに置き換える OUTPUT コマンドの例を示します。

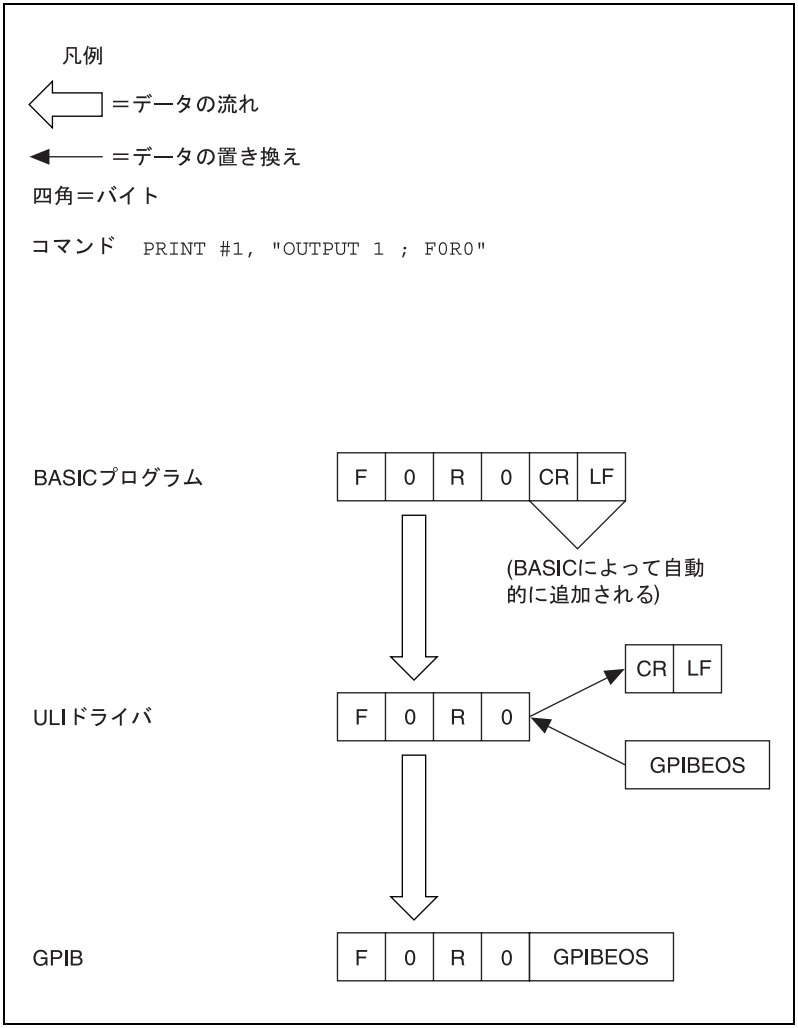


図 C-1　OUTPUT 文に文字カウントを含めない場合

GPIB ターミネータを使用する代わりに、出力文でカウント値を指定することもできます。バイト数を指定すると、GPIB ターミネータは自動的に

ディスエーブルされます。次の BASIC 文は、GPIB デバイスに文字 F, 0, R, 2 を送信します。

```
PRINT #1, "OUTPUT 1#4;F0R2"
```

カウント値が 4 で終了文字のスペースを残さないため、このコマンドは GPIB ターミネータを送信しません。カウントが 6 以上であれば (1 文字の GPIB ターミネータであれば 5)、GPIB ターミネータが含まれます。

INPUT ターミネータ

言語ターミネータ

INPUT 言語ターミネータは、データの流れが反対である点を除けば、OUTPUT 言語ターミネータと似た働きをします。GPIB ターミネータは、アプリケーションプログラムに送信される前に、言語ターミネータに置き換えられます。

データを、フォーマットされた ASCII 文字列や数値ではなく、バイナリ情報として扱わなければならない場合があります。たとえば、アプリケーションでカウント値を指定した場合、要求した文字数しか受け取りませんから、プログラミング言語で必要な言語ターミネータを受け取らないかもしれません。この場合、プログラミング言語のバイナリ転送関数 (BASIC では INPUT$) を使用します。詳しくは、次の「GPIB ターミネータ」の節を参照してください。

GPIB ターミネータ

計測器のメーカーによってデータ文字列の終了を示す方法が異なるため、システムに接続したデバイスの種類に応じて異なる GPIB ターミネータを使用する必要があるかもしれません。

デバイス入力は、以下のいずれかが発生すると終了します。

- 事前に定義されたカウントに達した。
- 1 文字の GPIB ターミネータが定義されており、それを受け取った。
- 2 文字の GPIB ターミネータが定義されており、それらを正しい順序で受け取った。
- GPIB EOI 線が最後のデータバイトでアサートされた。

以下にそれぞれの状況について説明します。

メモ　それぞれの状況で、データ行の文字は INPUT 文字列に入力することができるデータバイトを示します。

状況 1.　事前に定義されたカウントに達した。

図 2 に、ULI ドライバによる状況 1 の処理の流れを示します。

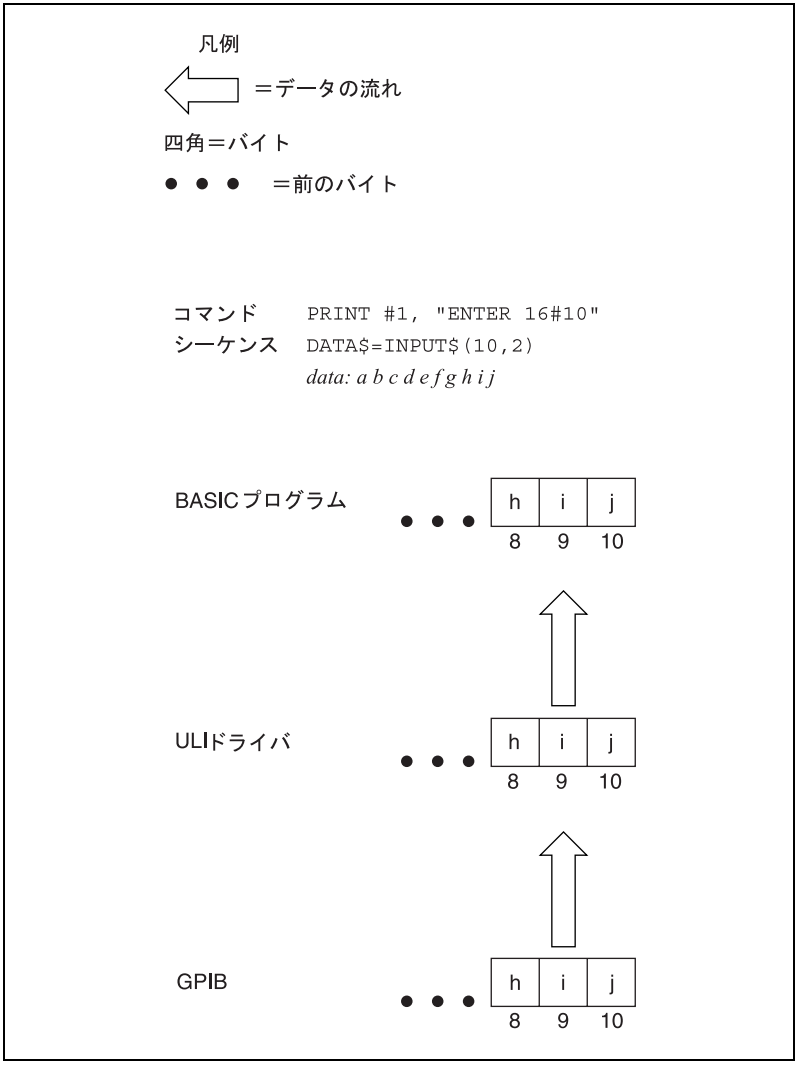


図 C-2　ENTER 文に文字カウントを含める場合

状況 2.　1 文字の GPIB ターミネータが定義されており、それを受け取った。

例

```
PRINT #1, "GPIBEOS IN CR"
```

```
PRINT #1, "ENTER 16"
INPUT #2,DATA$
```

data: a b c d e CR

状況 3.　　2 文字の GPIB ターミネータが定義されており、それらを正しい順序で受け取った。

例

```
PRINT #1, "GPIBEOS IN CHR(\X30)
   CHR(\X31)"
PRINT #1, "ENTER 16"
INPUT #2,DATA$
```

data: a b c d e f 0 1 (ASCII 0 = hex 30, ASCII 1 = hex 31)

メモ　16 進 "30" と 16 進 "31" の両方ではなく、いずれかのみが受け取られた場合、あるいは 16 進 "31"、16 進 "30" の順序で受け取られた場合、入力は終了しません。16 進 "30" と 16 進 "31" がこの順序で受け取られた場合のみ、入力は終了します。

図 C-3 に、ULI ドライバによる状況 2 および 3 の処理の流れを示します。

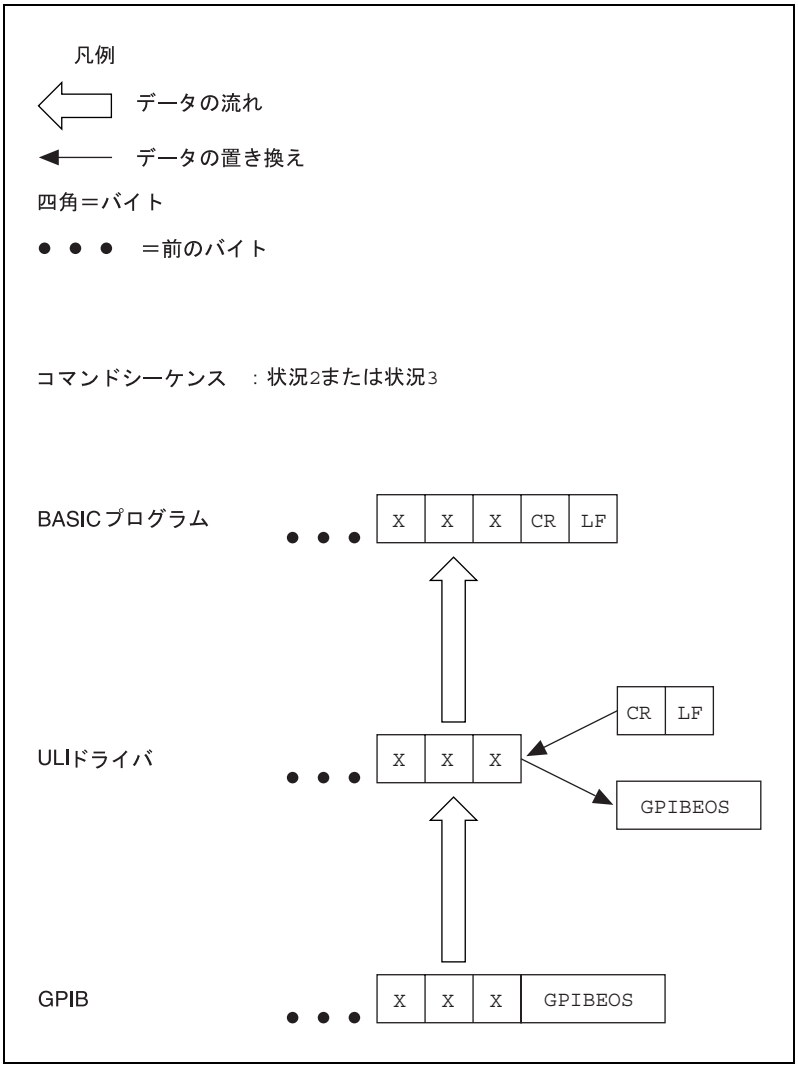


図 C-3　ENTER 文に文字カウントを含めない場合

状況 4.　GPIB EOI 線が最後のデータバイトでアサートされた。

例

```
print #1,enter 16"
    input #2,dATA$
    data: a b c d e f g h
```

EOI は、最後の文字 (h) でアサートされます。

ULI 関数

表 C-1 に、ULI 関数とそれぞれの簡単な説明をリストします。

表 C-1　ユニバーサル言語インタフェース関数

関数	説明
ABORT	バスを初期化し、バス転送が進行中の場合はそれを終了します。
CLEAR	選択されたデバイスをクリアします。
ENTER	GPIB からデータを読み取ります。
ERRTRAP	自動エラートラッピングをイネーブルまたはディスエーブルします。
GPIBEOS	GPIB データの終了文字を選択します。
LANGEOS	アプリケーションプログラムに受け渡されるデータの終了文字を設定します。
LOCAL	アドレスされたデバイスまたはすべてのデバイスカルモードにします。
LOCAL LOCKOUT	すべてのデバイスに対するフロントパネルからの制御をローを禁止します。
OFFLINE	デバイスまたはインタフェースボードをオフラインにします。
ONLINE	インタフェースボードをオンラインにします。
OUTPUT	デバイスにデータを送信します。
PASS CONTROL	制御資格をバス上の他のコントローラに渡します。
PPOLL	デバイスのパラレルポールを実行します。
PPOLL CONFIGURE	デバイスのパラレルポール応答を構成します。
PPOLL RESPONSE	パラレルポール応答用に GPIB インタフェースボード上の ist ビットを設定します。
PPOLL UNCONFIGURE	デバイスのパラレルポール応答をディスエーブルします。

表 C-1　ユニバーサル言語インタフェース関数（続き）

関数	説明
REMOTE	REN バス線を設定し、デバイスをリモートモードにします。
REQUEST	コントローラからのサービスを要求します。
RESET	構成パラメータをデフォルト値に設定します。
SEND	GPIB 管理コマンドを送信します。
SPOLL	デバイスをシリアルポールします。
STATUS	直前のバス呼び出しのステータスを返します。
TIMEOUT	タイムリミットを設定します。
TRIGGER	選択されたデバイスをトリガします。

関数構文の規約

ULI 関数の構文の説明に使用されている規約は以下のとおりです。

255 文字　コマンドまたはコマンドのデータ部分は 255 文字を超えてはいけません。

大文字　ENTER や ABORT など大文字の項目は、そのとおりに入力します。

空白　コマンドの空白は一般に無視されます。たとえば、LOCAL LOCKOUT は LOCAL LOCKOUT と同じです。次の 2 つの部分では空白は無視されません。1 つは、OUTPUT コマンドのデータ部分、もう 1 つは SEND コマンドの引数の間の空白です。

と ;　# 文字と ; 文字はそのとおりに入力します。

[ ]　大括弧で囲まれた項目はオプションです。

|　縦線で区切られた大かっこ内の項目 (`[item 1] | [item 2] | [item 3]`) はオプションです。その中から 1 つだけ選べます。また、何も選ばなくてもかまいません。

< >　角括弧で囲まれた項目は適切な値で置き換える必要があります。

関数の説明

これ以降の部分では、各 ULI 関数について例を挙げながら詳しく説明します。関数は、参照しやすいようにアルファベット順にリストしてあります。

ABORT

目的 :　バスを初期化し、バス上のデータ転送が進行中の場合はそれを終了します。

フォーマット :　`ABORT`

応答 :　なし

説明 :　`ABORT` コマンドは、GPIB ボードをシステムコントローラ (SC) およびコントローラインチャージ (CIC) にします。そして Interface Clear(IFC) 線と Attention(ATN) 線を順次アサートし、バストランザクションを中止します。

バスの初期化は、`ABORT` コマンドを使用すると通常の手続きよりも簡単に行えます。

例 :　バスを初期化します。またはバス上のデータ転送を終了します。

```
PRINT #1, "ABORT"
```

CLEAR

目的 :　選択されたデバイスをクリアします。

フォーマット :　`CLEAR [<addr>[,<addr>]]`

`addr` はデバイスの 1 次アドレス (およびオプションの 2 次アドレス) です。複数のアドレスがある場合は、コンマ (,) で区切ります。

応答 :　なし

説明 :　`CLEAR` コマンドは、Selected Device Clear(SDC) メッセージまたは Device Clear(DCL) メッセージ (アドレスを指定しない場合) を送信して特定のデバイスの内部関数またはデバイス関数をクリアします。

例 :　1. すべてのデバイスをクリアします。

```
PRINT #1, "CLEAR"
```

2. デバイス 2 をクリアします。

```
PRINT #1, "CLEAR 2"
```

ENTER

目的 :　GPIB からデータを読み取ります。

フォーマット :　`ENTER [<addr>] [# <count>] [DMA] Preserve Blanks`

`addr` は IEEE バスデバイスのアドレスです。`count` は読み取る文字数です。DMA はダイレクトメモリアクセス (DMA) をオンにします。`Preserve Blanks` は、デバイスの出力フォーマットの一部である可能性がある文字列変数内の先行ブランクを保持します。通常 BASIC では、これらのブランクは取り除かれます。

応答 :　デバイス固有のデータ

説明 :　デバイスのアドレスを指定すると、そのデバイスはトークにアドレスされます。アドレスを指定しない場合は、直前の `ENTER` コマンドまたは `SEND` コマンドの結果として、データを受け取るように GPIB ボードが構成されている必要があります。カウントを指定すると、その文字数のデータがデバイスから読み取られます。それ以外の場合、後続の `INPUT` コマンドは `GPIBEOS` 入力ターミネータを検出して読み取りを終了します。

例 :　1. デバイス 2 からデータを 1 行読み取ります。

```
PRINT #1, "ENTER 2"
LINE INPUT #2, A$
```

`Line Input` 文は入力ストリームの区切記号を無視します。詳しくは、ご使用のバージョンの BASIC リファレンスマニュアルを参照してください。

2. デバイス 16 から 10 バイトのデータを読み取ります。

```
PRINT #1, "ENTER 16#10"
B$ = INPUT$(10,2)
```

3. GPIBEOS 入力ターミネータを使用してデータを読み取ります。

```
PRINT #1, "ENTER 2"
INPUT #2, A$
```

ERRTRAP

目的 : 自動エラートラッピングをイネーブルまたはディスエーブルします。

フォーマット : `ERRTRAP [ON|OFF]`

応答 : なし

説明 : ERRTRAP がディスエーブルされていると、画面に表示されるエラーメッセージは「Device I/O Error(デバイス入出力エラー)」だけです。GPIB 固有のエラーメッセージ (No Listener(リスナなし) など) を受け取る場合は、ERRTRAP をイネーブルしてください。この機能のデフォルトの設定は ON(イネーブル) です。

例 : 自動エラー検出をディスエーブルします。

```
PRINT #1, "ERRTRAP OFF"
```

GPIBEOS

目的 : GPIB データの終了文字を選択します。

フォーマット : `GPIBEOS [IN|OUT] <term>`

IN は入力ターミネータ、OUT は出力ターミネータです。IN と OUT のどちらも指定しないと、両方とも設定されます。term はターミネータとして使用する文字です。

応答 : なし

説明 : この付録の「データ転送の終了」の節を参照してください。term には、以下のいずれかを使用できます。

CR	復帰文字
LF	改行文字

CHR(#)　# は 0 から 255 までの整数
'X　X は印刷可能な ASCII 文字

このコマンドがパラメータなしで実行された場合、GPIBEOS ターミネータの割当はキャンセルされます。

例 :

1. 両方のターミネータを CR に設定します。

```
PRINT #1, "GPIBEOS CR"
```

2. 入力ターミネータを LF に設定します。

```
PRINT #1, "GPIBEOS IN LF"
```

3. 出力ターミネータを CR LF に設定します。

```
PRINT #1, "GPIBEOS OUT CR LF"
```

LANGEOS

目的 :　アプリケーションプログラムで受け渡されるデータの終了文字を設定します。

フォーマット :　`LANGEOS [IN|OUT] <term>`

IN は入力ターミネータ、OUT は出力ターミネータです。IN と OUT のどちらも指定しないと、両方とも同じターミネータに設定されます。term はターミネータとして使用する文字です。

応答 :　なし

説明 :　この付録の「データ転送の終了」の節を参照してください。term には、以下のいずれかを使用できます。

CR　復帰文字
LF　改行文字
CHR(#)　# は 0 から 255 までの整数
'X　X は印刷可能な ASCII 文字

例 :

1. 両方のターミネータを CR に設定します。

```
PRINT #1, "LANGEOS CR"
```

2. 入力ターミネータを LF に設定します。

```
PRINT #1, "LANGEOS IN LF"
```

3. 出力ターミネータを CR LF に設定します。

```
PRINT #1, "LANGEOS OUT CR LF"
```

LOCAL

目的 :　アドレスされたデバイスまたはすべてのデバイスをローカルモードにします。

フォーマット :　LOCAL [<addr>[,<addr>]]

addr はデバイスの 1 次アドレス (およびオプションの 2 次アドレス) です。複数のアドレスがある場合は、コンマ (,) で区切ります。

応答 :　なし

説明 :　アドレスを指定すると、Go To Local(GTL) メッセージがそのデバイスに送信され、マニュアル操作モードになります。それ以外の場合は、Remote Enable 線がアサート解除されます。

例 :　1. Remote Enable 線をアサート解除します。

```
PRINT #1, "LOCAL"
```

2. Go To Local をデバイス 2 と 5 に送信します。

```
PRINT #1, "LOCAL 2,5"
```

LOCAL LOCKOUT

目的 :　すべてのデバイスに対するフロントパネルからの制御を禁止します。

フォーマット :　LOCAL LOCKOUT

応答 :　なし

例 :　Local Lockout バスコマンドを送信します。

```
PRINT #1, "LOCAL LOCKOUT"
```

OFFLINE

目的 :　デバイスまたはインタフェースボードをオフラインにします。これは、他のデバイスから GPIB ケーブルを切断することと同じです。

フォーマット :　`OFFLINE`

応答 :　なし

例 :　デバイスをオフラインにします。

```
PRINT #1, "OFFLINE"
```

ONLINE

目的 :　インタフェースボードをオンラインにします。これにより、デバイスまたはインタフェースボードのデフォルトの構成文字列が復元されます。

フォーマット :　`ONLINE`

応答 :　なし

例 :　インタフェースボードをオンラインにします。

```
PRINT #1, "ONLINE"
```

OUTPUT

目的 :　デバイスにデータを送信します。

フォーマット :　`OUTPUT [<addr>[,<addr>]] [# <count>] [dma] [END|NOEND]; <data>`

`addr` はデバイスのアドレス (およびオプションの 2 次アドレス) です。`count` は送信する文字数です。`DMA` はダイレクトメモリアクセス (`DMA`) をオンにします。`END | NOEND` でデータ伝送の終わりに EOI 線をアサートするかどうかを決定します。`data` はデバイスに送信される情報です。セミコロン (;) でデバイス依存のデータとコマンドを区切ります。

応答 :　なし

説明 :　デバイスのアドレスを指定すると、そのデバイスはリスンにアドレスされます。アドレスを指定しない場合は、

直前の OUTPUT コマンドまたは SEND コマンドの結果として、データを伝送するように GPIB ボードが構成されている必要があります。カウントを指定すると、その文字数だけデバイスに送信されます。それ以外の場合は、OUTPUT コマンドが GPIBEOS 出力ターミネータを追加します。

例 :

1. デバイス 2 に、"CURV? <GPIBEOS>" と送信します。

メモ : 事前に GPIBEOS コマンドを送信して、EOS 終了文字をセットアップしておく必要があります。

```
PRINT #1, "OUTPUT 2;CURV?"
```

2. デバイス 10,11,12 に F0R0 を送信します。

```
PRINT #1, "OUTPUT 10,11,12#4;F0R0"
```

PASS CONTROL

目的 : 制御資格をバス上の他のコントローラに渡します。

フォーマット : PASS CONTROL <addr>

addr はデバイスのアドレスです。

応答 : なし

説明 : GPIB ボードは ABORT コマンドを発行して制御資格を取り戻すことができます。

例 : デバイス 2 に制御資格を渡します。

```
PRINT #1, "PASS CONTROL 2"
```

PPOLL

目的 : デバイスのパラレルポールを実行します。

フォーマット : PPOLL

応答 : パラレルポール応答 (0 から 255 の数字)

説明 :　GPIB バス上のデバイスがパラレルポールされます (8 デバイスまで)。パラレルポールをサポートしていない GPIB デバイスもあります。

例 :　パラレルポールを実行します。

```
PRINT #1 "PPOLL"
INPUT #2, PPRESP%
```

PPOLL CONFIGURE

目的 :　デバイスのパラレルポール応答を構成します。

フォーマット :　`PPOLL CONFIGURE <addr> ; <number>`

`addr` は構成するデバイスの 1 次アドレスです。セミコロン (;) でデバイス依存のデータとコマンドを区切ります。`number` は、パラレルポールでデバイスがどの線をアサートするか、およびパラレルポールをイネーブルするかディスエーブルするかを指定するパターンです。

応答 :　なし

説明 :　パラレルポーリングについて詳しくは、第 7 章「GPIB プログラミングテクニック」を参照してください。

例 :　デバイス 2 に 10 進値 41 を送信して、パラレルポールを構成します。

```
PRINT #1, "PPOLL CONFIGURE 2;41"
```

PPOLL RESPONSE

目的 :　パラレルポール応答用に GPIB インタフェースボード上の個々のステータス (ist) ビットを設定します。

フォーマット :　`PPOLL RESPONSE <number>`

`number` は、GPIB インタフェースボードのステータスを示します。`number` が 0 以外の場合、ist ビットが設定されます。0 の場合 `ist` ビットはクリアされます。

応答 :　なし

説明 :　パラレルポーリングについて詳しくは、第 7 章「GPIB プログラミングテクニック」を参照してください。

例 :　個々のステータスビットを設定します。

```
PRINT #1, "PPOLL RESPONSE 2"
```

PPOLL UNCONFIGURE

目的 :　デバイスのパラレルポール応答をディスエーブルします。

フォーマット :　`PPOLL UNCONFIGURE [<addr>[,<addr>]]`

応答 :　なし

例 :　1. デバイス 2 の構成を解除します。

```
PRINT #1, "PPOLL UNCONFIGURE 2"
```

2. すべてのデバイスの構成を解除します。

```
PRINT #1, "PPOLL UNCONFIGURE"
```

REMOTE

目的 :　Remote Enable(REN) バス線をアサートして、デバイスをリモートモードにします。

フォーマット :　`REMOTE [<addr>[,<addr>]]`

`addr` はデバイスの 1 次アドレス (およびオプションの 2 次アドレス) です。複数の 1 次アドレスがある場合は、コンマ (,) で区切ります。

応答 :　なし

説明 :　REN バス線がアサートされます。デバイスのアドレスを指定すると、そのデバイスはさらにリスンにアドレスされ、リモートプログラミングモードに入ります。

例 :　1. Remote Enable をアサートします。

```
PRINT #1, "REMOTE"
```

2. Remote Enable をアサートし、デバイス 2 と 5 をリスンにアドレスします。

```
PRINT #1, "REMOTE 2, 5"
```

REQUEST

目的 :　サービスを要求し、シリアルポールステータスバイトの設定または変更を行います。

フォーマット :
```
REQUEST <number>
```

number は、GPIB の CIC である他のデバイスからシリアルポールされたときに GPIB ボードが提供するステータスバイトです。number にビット 6(16 進 40 のビット) を設定すると、GPIB ボードは GPIB SRQ 線をアサートしてさらにコントローラからのサービスを要求します。

応答 :　なし

説明 :　REQUEST コマンドは、Service Request(SRQ) シグナルを使用してコントローラからのサービスを要求し、コントローラが GPIB ボードをシリアルポールしたときにシステム依存のステータスバイトを提供するために使用します。

例 :　ステータスバイト &H41 でサービスを要求します。

```
PRINT #1, "REQUEST &H41";
```

RESET

目的 :　構成パラメータ (1 次および 2 次アドレス、LANGEOS 文字、GPIBEOS 文字) をデフォルト値に設定します。

フォーマット :
```
RESET
```

応答 :　なし

例 :　アドレスと EOS パラメータを復元します。

```
PRINT #1, "RESET"
```

SEND

目的 :　GPIB 管理コマンドを送信します。

フォーマット :
```
SEND [MTA|TALK <addr>] [MLA|LISTEN <addr>]
[UNL] [UNT] [DATA <number>
[, <number>, <...>]]
```

MTA は GPIB ボードのトークアドレスです。TALK addr は、デバイスのトークアドレスと 1 次アドレス addr を組み合わせます。MLA は GPIB ボードのリスンアドレスです。LISTEN addr は、デバイスのリスンアドレスと 1 次アドレス addr を組み合わせます。UNL はアンリスンコマンドメッセージです。これは常に &H3F と定義します。UNT はアントークコマンドメッセージです。これは常に &H5F と定義します。DATA は転送されるデバイス固有のデータです。number は 0 から 255 までの整数です。複数の 1 次アドレスがある場合は、コンマ (,) で区切ります。

応答 :　なし

説明 :　データをバイト単位で制御し、バス上で転送します。これらのコマンドは、GPIB コントローラに最大限の柔軟性と制御を提供します。

例 :　アドレス 2 のデバイスからアドレス 1 のデバイスに、2 つのデータバイト、10 進値 1 と 10 進値 2 を送ります。

```
PRINT #1, "SEND LISTEN 1 talk 2 data 1,2"
```

SPOLL

目的 :　デバイスをシリアルポールします。

フォーマット :　SPOLL [<addr>]

addr はデバイスの 1 次アドレスです。

応答 :　デバイスのシリアルポール応答 (0 から 255 までの整数)

説明 :　シリアルポールについて詳しくは、第 7 章「GPIB プログラミングテクニック」を参照してください。

例 :　1.　デバイス 2 をシリアルポールします。

```
PRINT #1, "SPOLL 2"
```

2.　シリアルポールステータスを受け取ります。

```
INPUT #2, SP%
```

STATUS

目的 :　直前のバス呼び出しのステータスを返します。

フォーマット : `STATUS`

応答 : ステータス変数 `IBSTA%, IBERR%, IBCNT%` を返します。

説明 : `IBSTA%, IBERR%, IBCNT%` は、それぞれステータス変数、エラー変数、カウント変数に対応します。詳しくは、第 3 章「アプリケーションの開発」を参照してください。

例 : ステータスをチェックします。

```
PRINT #1,"STATUS"

INPUT #2, IBSTA%, IBERR%, IBCNT%.
```

TIMEOUT

目的 : 制限時間を設定します。

フォーマット : `TIMEOUT <number>`

`number` はタイムアウト状態が発生するまでの時間を表すコードです。

応答 : なし

説明 : バス上のデータまたは制御トランザクションは一定の時間内に完了させる必要があります。この制限は表 C-2 に示す制限時間のいずれかに設定することができます。

表 C-2　ULI タイムアウトコード値

数値	最低限のタイムアウト
0	ディスエーブル
1	10 µs
2	30 µs
3	100 µs
4	300 µs
5	1 ms
6	3 ms
7	10 ms

表 C-2　ULI タイムアウトコード値（続き）

数値	最低限のタイムアウト
8	30 ms
9	100 ms
10	300 ms
11	1 s
12	3 s
13	10 s
14	30 s
15	100 s
16	300 s
17	1000 s

例 :　　タイムアウト時間を 10 秒に設定します。

```
PRINT #1, "TIMEOUT 13"
```

TRIGGER

目的 :　　選択されたデバイスをトリガします。

フォーマット :　　`TRIGGER [<addr>[,<addr>]]`

`addr` はデバイスの 1 次アドレス (およびオプションの 2 次アドレス) です。複数のアドレスがある場合は、コンマ (,) で区切ります。

応答 :　　なし

説明 :　　`TRIGGER` コマンドは指定された各デバイスに Group Execute Trigger(GET) メッセージを送ります。デバイスを指定しないと、それまでにリスンにアドレスされたデバイスだけが GET を受け取ります。

例 :　　デバイス `2` と `5` に `TRIGGER` コマンドを発行します。

```
PRINT #1 "TRIGGER 2, 5"
```

D

カスタマーコミュニケーション

本章には、ユーザの皆さんの便宜のために、技術的サポートに必要な情報を記入する書式だけでなく、製品やマニュアル類についてのご意見をお寄せいただくための書式も取り入れました。まずテクニカルサポートフォーム (サポート用紙) にご記入になってからナショナルインスツルメンツにご連絡ください。素早く、確実に問題を解決できるようになります。

ナショナルインスツルメンツは、広範な技術支援を世界中のユーザの皆さんに提供しています。米国およびカナダでは、アプリケーションエンジニアが月曜から金曜の 9:00 ～ 17:00 まで待機しております。これ以外の国では、お近くの支社にご連絡ください。ご相談のファックスは 24 時間いつご送信くださっても結構です。

日本ナショナルインスツルメンツ (株)

TEL:　(03) 3788-1921

FAX:　(03) 3788-1923

テクニカルサポートフォーム

テクニカルサポートは 24 時間ファックスで受け付けております。「入門マニュアル」の構成記録フォームの情報も記入してください。ページが足りないときは、適宜追加してください。

氏名 _____

会社名 _____

住所 _____

_____

ファックス (___) _____ 電話 (___) _____

コンピュータメーカ名 _____

機種 _____ CPU タイプ _____

オペレーティングシステム (バージョン番号も必要) _____

クロック速度 _____MHz RAM(“□›“”ÿ)_____MB ディスプレイアダプタ _____

マウス ___ 有 ___ 無 他のアダプタ類 _____

ハードディスク容量 _____MB メーカ名 _____

計測器 _____

ナショナルインスツルメンツ社のハードウェア製品モデル _____

レビジョン _____

構成 _____

ナショナルインスツルメンツ社のソフトウェア製品 _____

バージョン _____

構成 _____

問題点を詳しく説明してください _____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

表示されるエラーメッセージを記入してください

どのような手順で操作すると問題点が再現しますか

マニュアルについてのご意見をお聞かせください

ナショナルインスツルメンツでは、製品のマニュアルについての皆さんのご意見を心よりお待ちしております。皆さんがご意見をお寄せくださることが、ニーズに応える高品質の製品作りにつながります。

書名：　DOS 用 NI-488.2TTM ユーザマニュアル

編集：　1993 年 11 月

部品番号：　370885A-01

マニュアルの完成度、分かりやすさ、構成についてのご意見をお書きください。

_____

_____

_____

_____

_____

マニュアルに誤りを発見された場合は、そのページ番号を明記して、誤りの内容をご説明ください。

_____

_____

_____

_____

_____

ご協力どうも有り難うございました。

氏名 _____

役職名 _____

会社名 _____

住所 _____

_____

_____

電話番号　(___) _____

郵送宛先：

日本ナショナルインスツルメンツ (株)

〒 142 東京都品川区戸越 5-14-24

ITO ビル 2 階

ファックス宛先：

日本ナショナルインスツルメンツ(株)

ファックス　(03) 3788-1923

用語集

接頭辞	意味	数値
n-	ナノ	10^{-9}
µ-	マイクロ	10^{-6}
m-	ミリ	10^{-3}
k-	キロ	10^{3}
M-	メガ	10^{6}

A

ANSI　米国規格協会。

ASCII　情報交換用米国標準コード。

B

BIOS　基本入出力システム。

C

CFE　Configuration Enable(構成イネーブル)。CFGn に先行し、デバイスを構成モードにする GPIB コマンド。

CFGn　CFGn コマンド (CFG1 から CFG15) は CFE の後に続き、システムでその長さ (メートル) のケーブルに接続されているすべてのデバイスを構成し、HS488 転送がエラーなく行われるようにします。

CIC　「コントローラインチャージ」の項を参照してください。

CPU　中央処理装置。

D

DAV(Data Valid)　GPIB の 3 本のハンドシェイク線の 1 つ。「ハンドシェイク」の項を参照してください。

DCL　Device Clear(デバイスクリア)。あるデバイス、またはすべてのデバイスの内部機能をリセットする GPIB コマンド。「IFC」「SDC」の項を参照してください。

Device Clear　「DCL」の項を参照してください。

DIO1 から DIO8　GPIB 線。デバイスからデバイスへコマンドやデータバイトを転送します。

DMA(ダイレクトメ　GPIB ボードとメモリ間の高速データ転送で、直モリアクセス) 接には CPU の管理を受けません。DMA が利用できないシステムもあります。「プログラムド I/O」の項も参照してください。

E

END または END メッセージ　データ文字列の終わりを知らせるメッセージ。END は、最後のデータバイトで GPIB End or Identify(終了または識別コードの要求 :EOI) 線をアサートすることによって送信されます。

EOI　データメッセージの最終バイト (END) またはパラレルポール識別 (IDY) メッセージのシグナルを出すために使用される GPIB 線。

EOS または EOS バイト　7 ビットまたは 8 ビットの文字列の終わり文字で、データメッセージの最終バイトとして送信されます。

EOT　転送終了。

ESB　Event Status(イベントステータス) ビット。シリアルポールに応答するデバイスから受け取る、IEEE 488.2 で定義されたステータスバイトの一部。

G

GET　Group Execute Trigger(グループ実行トリガ)。デバイスまたはアドレスされたリスナの内部機能をトリガする GPIB コマンド。

Go To Local　「GTL」の項を参照してください。

GPIB　汎用インタフェースバス。ANSI/IEEE 規格 488.1-1987 および ANSI/IEEE 規格 488.2-1987 で定義された通信インタフェースシステムの一般名。

GPIB アドレス　GPIB 上のデバイスのアドレス。1 次アドレス (MLA および MTA) と場合によりさらに 2 次アドレス (MSA) を加えて構成されます。GPIB ボードには GPIB アドレスと I/O アドレスが 1 つずつあります。

GPIB ボード　ナショナルインスツルメンツの一連の GPIB インタフェースボードの呼び名。

Group Execute Trigger　「GET」の項を参照してください。

GTL　Go to Local(ローカル状態に設定)。アドレスしたリスナをローカル (フロントパネル) 制御モードにする GPIB コマンド。

H

hex　16 進法。16 を基数として表される数。例 :10 進数 16=16 進数 10。

Hz　ヘルツ。

I

ibcnt　各 NI-488.2 入出力関数の実行後、このグローバル変数には伝送された実際のバイト数が入ります。

ibconf　DOS 用 NI-488.2 ドライバ構成プログラム。

iberr　敗した関数呼出しに対応する特定のエラーコードが入ったグローバル変数。

ibic　DOS 用インタフェースバス対話式制御プログラム。GPIB デバイスとの通信、トラブルシュート、およびアプリケーションの開発に使用されます。

ibsta　各関数呼出しが終了すると、このグローバル変数 (ステータスワード) にはステータス情報が入ります。

IEEE　Institute of Electrical and Electronic Engineers

I/O アドレス　CPU から見た場合の GPIB ボードのアドレス。GPIB ボードの GPIB アドレスとは異なります。ポートアドレスまたはボードアドレスとも呼ばれます。

ist　Parallel Poll Configure(パラレルポール構成) 関数で使用されるステータスバイトの Individual Status(個々のステータス) ビット。

K

KB　キロバイト。

L

LAD(Listen Address)　「MLA」の項を参照してください。

M

m　メートル。

MAV　Message Available(メッセージ可能) ビット。シリアルポールに応答するデバイスから受け取る、IEEE 488.2 で定義されたステータスバイトの一部。

MB　メガバイト。

MLA(My Listen Address)　デバイスをリスナとしてアドレスする GPIB コマンド。31 の 1 次アドレスから 1 つを選ぶことができます。

MSA(My Secondary Address)　My Secondary Address(自分の 2 次アドレス)。拡張（2 バイト) アドレス指定が使用された場合、デバイスをリスナまたはトーカにアドレスする GPIB コマンド。完全なアドレスは MLA または MTA アドレスの後ろに MSA アドレスがきます。2 次アドレスは 31 あるので、デバイスのリスンまたはトークアドレスは合計 961 種類になります。

MTA(My Talk Address)　デバイスをトーカとしてアドレスする GPIB コマンド。31 の 1 次アドレスから 1 つを選ぶことができます。

N

NDAC(Not Data Accepted)　3 本の GPIB ハンドシェイク線の 1 つ。「ハンドシェイク」の項を参照してください。

NRFD(Not Ready for Data)　3 本の GPIB ハンドシェイク線の 1 つ。「ハンドシェイク」の項を参照してください。

P

PIO　「プログラムド I/O」の項を参照してください。

PPC (Parallel Poll Configure)　Parallel Poll Configure(パラレルポール構成)。アドレスされたリスナをパラレルポールに参加できるように構成する GPIB コマンド。

PPD (Parallel Poll Disable)　Parallel Poll Disable(パラレルポールディスエーブル)。ポーリングに参加するように構成されたデバイスのポーリング参加をディスエーブルする GPIB コマンド。PPD コマンドは 16 種類あります。

PPE (Parallel Poll Enable)　Parallel Poll Enable(パラレルポールイネーブル)。構成されたデバイスのポーリング参加をイネーブルし、DIO 応答線に割り当てる GPIB コマンド。PPD コマンドは 16 種類あります。

PPU (Parallel Poll Unconfigure)　Parallel Poll Unconfigure(パラレルポール構成解除)。どのようなデバイスもポーリングに参加できなくする GPIB コマンド。

R

RAM　ランダムアクセスメモリ。

RQS　Request Service(要求サービス)。

S

s　秒。

SDC　Selected Device Clear(選択デバイスクリア)。アドレスされたリスナの内部またはデバイス機能をリセットする GPIB コマンド。「DCL」「IFC」の項も参照してください。

Service Request　「SRQ」を参照してください。

SPD(Serial Poll Disable)　Serial Poll Disable(シリアルポールディスエーブル)。SPE コマンドをキャンセルする GPIB コマンド。

SPE(Serial Poll Enable)　Serial Poll Enable(シリアルポールイネーブル)。特定のデバイスのポーリングをイネーブルする GPIB コマンド。このデバイスはトーカにアドレスされている必要があります。「SPD」も参照してください。

SRQ(Service Request)　デバイスが CIC にサービスを要求するときにアサートする GPIB 線。

T

TAD(Talk Address)　「MTA」を参照してください。

TCT　Take Control(制御資格取得)。バスの制御資格を現在のコントローラからアドレスされたトーカに受け渡す GPIB コマンド。

TLC　ほとんどの GPIB トーカ、リスナおよびコントローラ機能を実行するハードウェアの集積回路。

U

ud(ユニット記述子)　関数の目的である GPIB インタフェースボードまたは他の GPIB デバイスのユニット記述子を含む、各関数呼出しの変数名と最初の引数。

ULI　ユニバーサル言語インタフェース。

UNL　unlisten(全リスナ解除)。アクティブなリスナに対するアドレスを解除する GPIB コマンド。

UNT　Untalk(全トーカ解除)。アクティブなトーカに対するアドレスを解除する GPIB コマンド。

あ

アクセスボード　バス上の接続デバイスを制御し、それらと通信を行う GPIB ボード。

アクセプタハンドシェイク　この GPIB インタフェース関数を使用して、リスナはデータを、すべてのデバイスはコマンドを受け取ります。「ソースハンドシェイク」「ハンドシェイク」の項も参照してください。

インタフェースメッセージ　コントローラからすべてのデバイスに送信され、GPIB を管理するために使用される同報通信メッセージ。

か

言語インタフェース　NI-488 関数または NI-488.2 ルーチンを使用してドライバにアクセスするアプリケーションプログラムをイネーブルするコード。

コントローラインチャージ　インタフェースメッセージを他のデバイスに送信することにより、GPIB を管理するデバイス。

さ

再同期　NI-488.2 ソフトウェアとユーザアプリケーションは、非同期入出力操作が完了したら再同期化する必要があります。

システムコントローラ　Interface Clear(IFC) メッセージを送信して制御資格をアサートする (GPIB の CIC になる) ことができるただ 1 つの指定されたコントローラ。他のデバイスは、制御資格を受け渡されない限り CIC になることはできません。

自動シリアルポーリング (自動ポーリング)　NI-488.2 ソフトウェアの特徴の 1 つ。デバイスが GPIB SRQ 線をアサートすると、ドライバによって自動的に実行されるシリアルポール。

シリアルポール　1 度に 1 台のデバイスのステータスバイトをポーリングして読み取るプロセス。「パラレルポール」の項も参照してください。

ステータスバイト　シリアルポールされたデバイスが送信する、IEEE 488.2 で定義されたデータバイト。

ステータスワード　「ibsta」を参照してください。

ソースハンドシェイク　データおよびコマンドを転送する GPIB インタフェース関数。この関数を使用してトーカはデータを、コントローラはコマンドを送信します。「アクセプタハンドシェイク」「ハンドシェイク」の項も参照してください。

た

タイムアウト　NI-488.2 ドライバの特徴の 1 つ。入出力関数の使用中に GPIB に問題が発生すると、関数の動作がいつまでも終了しないことがありますが、タイムアウトを設定することで防ぐことができます。

デバイスレベル関数　基本的なボード動作をいくつか一緒にまとめて、1 つの関数の動作として実行する関数。デバイスレベル関数を使用すれば、ユーザはバスの管理やその他の GPIB プロトコルなどを考慮する必要はありません。

トーカ　データメッセージをリスナに送信する GPIB デバイス。

同期　ドライバ関数がいつ実行されるかが予測できる場合の、NI-488.2 ドライバ関数とプロセスの関係を意味します。ドライバが関数を完了するまでプロセスはブロックされます。

ドライバ　オペレーティングシステム内にインストールされているデバイスドライバソフトウェア。

な

入出力 (入力 / 出力)　このマニュアルでは、入出力とは GPIB ボードを介してコンピュータと GPIB 上のデバイス間で行われる、コマンドまたはメッセージの伝送を意味します。

は

ハイレベル関数　「デバイスレベル関数」の項を参照してください。

パラレルポール　構成されたデバイスをすべて一度にポーリングし、複合ポーリング応答を読むプロセス。「シリアルポール」の項も参照してください。

ハンドシェイク　あるデバイスのソースハンドシェイク機能から別のデバイスのアクセプタハンドシェイク機能にバイトを転送するメカニズム。DAV, NRFD, NDAC の 3 本の GPIB 線をインターロックのやり方で使用して転送の各状況を伝え、一番遅いデバイスのスピードに合わせてバイトを非同期に (たとえばクロックなしで) 送るようになっています。

ハンドシェイキングについて詳しくは、ANSI/IEEE 規格 488.1-1987 を参照してください。

非同期　プログラムの実行中、いつ発生するか予測できない動作またはイベント。

プログラムド I/O　GPIB ボードとメモリ間の低速データ転送。CPU はプログラムの命令に従って各データバイトを送ります。「DMA」の項も参照してください。

ベース I/O アドレス　「I/O アドレス」の項を参照してください。

ボードレベル関数　単一の動作を実行する基本的な関数。

ま

マルチタスキング　複数のプログラムまたはタスクを同時に処理すること。

メモリ常駐　RAM 常駐。

ら

リスナ　トーカから送られたデータメッセージを受け取る GPIB デバイス。

リスンアドレス　「MLA」の項を参照してください。

ローレベル関数　「ボードレベル関数」の項を参照してください。

索引

F

G

H

I

O

P

Q

R

S

T

V

W

あ

か

さ

た

は

ま

や

ら

わ