

# Windows 用 NI-488.2™ ユーザマニュアル

1993 年 11 月版  
製品番号 370886A-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

ナショナルインスト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トルメンツにあった場合、ナショナルインストルメンツは、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。ナショナルインストルメンツは、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。

ナショナルインスト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、ナショナルインスト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、ナショナルインスト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トルメンツは、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。

ナショナルインストルメンツは、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。ナショナルインスト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、ナショナルインストルメンツは、たとえそのようなことが生じる可能性があるとは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ト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。ナショナルインスト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。ナショナルインストルメンツは、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を負いません。ここに定めた限定保証では、ユーザがナショナルインストルメンツの設置・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

著作権法に基づき、ナショナルインスツルメンツ社の書面による事前の許可なく、複写、記録、情報検索システムへの保存および翻訳を含め、電子的であるか機械的であるかを問わず、いかなる形式であれ本書の一部あるいは全部を複製または伝送することを禁止します。

## 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

---

## このマニュアルについて

このマニュアルセットの使い方.....	xiii
このマニュアルの構成.....	xiv
このマニュアルで使う表記法.....	xv
関連文書.....	xvi
カスタマーコミュニケーション.....	xvi

## 第1章

### はじめに

GPIB の概要.....	1-1
トーカ、リスナ、コントローラ.....	1-1
コントローラインチャージとシステムコントローラ.....	1-1
GPIB アドレス指定.....	1-2
GPIB 上でのメッセージ送信.....	1-2
データ線.....	1-3
ハンドシェイク線.....	1-3
インタフェース管理線.....	1-3
システムのセットアップと構成.....	1-4
複数ボードの制御.....	1-5
必要なシステム構成.....	1-6
NI-488.2 ソフトウェアパッケージ.....	1-7
NI-488.2 ドライバとドライバユーティリティ.....	1-7
C 言語ファイル.....	1-8
Visual Basic 言語ファイル.....	1-8
サンプルアプリケーションファイル.....	1-8
NI-488.2 ソフトウェアと Windows との関係.....	1-9
NI-488.2 ドライバのアンロードと再ロード.....	1-10

## 第2章

### アプリケーション例

例 1: 基本的な通信.....	2-2
例 2: デバイスのクリアとトリガ.....	2-3
例 3: 非同期入出力.....	2-5
例 4: 文字列の終わり (EOS) モード.....	2-7
例 5: サービス要求.....	2-9
例 6: IEEE 488.2 準拠デバイスとの簡単な通信.....	2-14
例 7: NI-488.2 ルーチンを使ったシリアルポール.....	2-16
例 8: パラレルポール.....	2-18
例 9: コントローラでないデバイスのエミュレーション.....	2-21

**第 3 章****アプリケーションの開発**

NI-488.2 DLL へのアクセス方法の選択 .....	3-1
NI-488 関数または NI-488.2 ルーチンの選択 .....	3-1
NI-488 関数の使い方 : 1 ボードに 1 デバイスの場合 .....	3-2
NI-488 デバイス関数 .....	3-2
NI-488 ボード関数 .....	3-2
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 .....	3-3
グローバル変数を使ったステータス確認 .....	3-3
ステータスワード - ibsta .....	3-3
エラー変数 - iberr .....	3-5
カウント定数 - ibcnt と ibcntl .....	3-5
wibic を使ったデバイスとの通信 .....	3-6
NI-488 アプリケーションの書き方 .....	3-6
組み込む項目 .....	3-6
NI-488 プログラムシェル .....	3-7
一般的なプログラム手順とその例 .....	3-8
ステップ 1 デバイスのオープン .....	3-8
ステップ 2 デバイスのクリア .....	3-8
ステップ 3 デバイスの構成 .....	3-9
ステップ 4 デバイスのトリガ .....	3-9
ステップ 5 測定の待機 .....	3-10
ステップ 6 測定の読み取り .....	3-10
ステップ 7 データの処理 .....	3-11
ステップ 8 デバイスをオフラインにする .....	3-11
NI-488.2 アプリケーションの書き方 .....	3-11
組み込む項目 .....	3-11
NI-488.2 プログラムシェル .....	3-12
一般的なプログラム手順とその例 .....	3-14
ステップ 1 初期化 .....	3-14
ステップ 2 全リスナの検出 .....	3-14
ステップ 3 計測器の識別 .....	3-15
ステップ 4 計測器の初期化 .....	3-16
ステップ 5 計測器の構成 .....	3-16
ステップ 6 計測器のトリガ .....	3-17
ステップ 7 測定の待機 .....	3-17
ステップ 8 測定の読み取り .....	3-18
ステップ 9 データの処理 .....	3-19
ステップ 10 ボードをオフラインにする .....	3-19
アプリケーションのコンパイル、リンク、実行 .....	3-19
Microsoft C .....	3-19
Visual Basic .....	3-20

C 言語によるダイレクトエントリ .....	3-20
Visual Basic によるダイレクトエントリ .....	3-21

## 第 4 章

### アプリケーションのデバッグ

wibtest の実行 .....	4-1
ドライバの存在確認テスト .....	4-1
ボードの存在確認テスト .....	4-1
GPIB ケーブル接続 .....	4-2
GPIBInfo の実行 .....	4-2
グローバルステータス変数を使ったデバッグ .....	4-3
wibic を使ったデバッグ .....	4-4
GPIB Spy を使ったデバッグ .....	4-4
GPIB エラーコード .....	4-4
構成エラー .....	4-5
タイミングエラー .....	4-6
通信エラー .....	4-6
アドレス指定の繰り返し .....	4-6
終了方法 .....	4-7
Q&A .....	4-7

## 第 5 章

### wibic - Windows 用インタフェースバス対話式制御ユーティリティ

概要 .....	5-1
NI-488 関数を使用した例 .....	5-1
wibic 構文 .....	5-5
数値の構文 .....	5-5
文字列の構文 .....	5-5
アドレスの構文 .....	5-6
NI-488 関数の wibic 構文 .....	5-6
NI-488.2 ルーチンの wibic 構文 .....	5-9
ステータスワード .....	5-10
エラー情報 .....	5-11
カウント .....	5-11
一般的な NI-488 関数 .....	5-12
ibfind .....	5-12
ibdev .....	5-12
ibwrt .....	5-14
ibrd .....	5-14
wibic における一般的な NI-488.2 ルーチン .....	5-15
Se .....	5-15
Send と SendList .....	5-15
Receive .....	5-16

補助関数 .....	5-16
Set( デバイスまたはボードの選択 ) .....	5-17
Help( ヘルプ情報の表示 ) .....	5-17
! ( 直前の関数の繰り返し ) .....	5-17
- ( 画面表示 OFF)、+ ( 画面表示 ON) .....	5-17
n* ( 関数を n 回繰り返す ) .....	5-18
\$( 間接ファイルの実行 ) .....	5-19
Print(ASCII 文字列の表示) .....	5-19

## 第 6 章

### GPIB Spy

概要 .....	6-1
GPIB Spy の起動 .....	6-1
GPIB Spy の終了 .....	6-2
Call Details( 呼び出しの詳細 ) の表示 .....	6-2
GPIB Spy の構成オプション .....	6-3
Calls to Highlight( 強調表示する呼び出し ) .....	6-3
Calls to Trap On( トラップする呼び出し ) .....	6-3
Buffer History( バッファ履歴 ) .....	6-3
Set Highlight Color( 強調色の設定 ) .....	6-4
Store Configuration( 構成の保存 ) .....	6-4
GPIB Spy の出力オプション .....	6-4
To Screen( 画面へ ) .....	6-4
To File( ファイルへ ) .....	6-5
Clear Screen( 画面のクリア ) .....	6-5
GPIB Spy Help(GPIB Spy のヘルプ) .....	6-5
パフォーマンスについて .....	6-5

## 第 7 章

### GPIB プログラミングテクニ

#### ック

データ転送の終了 .....	7-1
高速データ転送 (HS488) .....	7-2
HS488 のイネーブル .....	7-2
システム構成の HS488 への影響 .....	7-3
GPIB 条件の待機 .....	7-3
デバイスレベル呼び出しとバス管理 .....	7-4
トーカー / リスナーアプリケーション .....	7-5
コントローラからのメッセージの待機 .....	7-5
イベント待ち行列の使い方 .....	7-5
サービス要求 .....	7-6



シリアルポーリング.....	7-6
IEEE 488 デバイスからのサービス要求.....	7-6
IEEE 488.2 デバイスからのサービス要求.....	7-6
自動シリアルポーリング.....	7-7
スタック SRQ 状態.....	7-7
自動ポーリングと割り込み.....	7-8
NI-488 デバイス関数を使った SRQ とシリアルポーリング.....	7-8
NI-488.2 ルーチンを使った SRQ とシリアルポーリング.....	7-9
例 1: FindRQS の使い方.....	7-10
例 2: AllSpoll の使い方.....	7-11
パラレルポーリング.....	7-12
パラレルポーリングの実行.....	7-12
NI-488 関数を使ったパラレルポーリング.....	7-12
NI-488.2 ルーチンを使ったパラレルポーリング.....	7-13

## 第 8 章

### wibconf - Windows 用インタフェースバス構成ユーティリティ

概要.....	8-1
wibconf の起動.....	8-1
wibconf の上位レベルと下位レベル.....	8-2
上位レベルデバイスマップ.....	8-3
ボードのデバイスマップ.....	8-3
Help( ヘルプ ).....	8-4
Rename( 名称変更 ).....	8-4
(Dis)connect( 接続 / 非接続 ).....	8-4
Edit( 編集 ).....	8-4
Exit( 終了 ).....	8-5
下位レベルのデバイス / ボードの特性.....	8-5
特性の変更.....	8-6
ボードまたはデバイスの変更.....	8-6
Help( ヘルプ ).....	8-6
Reset Value( 値のリセット ).....	8-6
Return to Map( マップに戻る ).....	8-6
ボードおよびデバイスの構成オプション.....	8-7
Primary GPIB Address(1 次 GPIB アドレス ).....	8-7
Secondary GPIB Address(2 次 GPIB アドレス ).....	8-7
Timeout Setting( タイムアウトの設定 ).....	8-8
Serial Poll Timeouts( シリアルポールタイムアウト、デバイス特性のみ ).....	8-8
Terminate Read on EOS(EOS による読み取りの終了 ).....	8-8
Set EOI with EOS on Writes( 書き込みで EOS とともに EOI を設定 ).....	8-8
Type of Compare on EOS(EOS での比較のタイプ ).....	8-8
EOS Byte(EOS バイト ).....	8-9
Send EOI at End of Write( 書き込みの終わりに EOI を送信 ).....	8-9

System Controller(システムコントローラ、ボード特性のみ).....	8-9
Assert REN when SC(SC時のRENのアサート、ボード特性のみ).....	8-9
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).....	8-9
Enable CIC Protocol(CICプロトコルのイネーブル、ボード特性のみ).....	8-10
Bus Timing(バスタイミング、ボード特性のみ).....	8-10
Cable Length for High Speed(高速用ケーブル長、ボード特性のみ).....	8-10
Parallel Poll Duration(パラレルポールの間隔、ボード特性のみ).....	8-11
Use This GPIB Interface(このGPIBインタフェースを使用、ボード特性のみ).....	8-11
Base I/O Address(ベースI/Oアドレス、ボード特性のみ).....	8-11
DMA Channel(DMAチャンネル、ボード特性のみ).....	8-11
Interrupt Jumper Setting(割り込みジャンパの設定、ボード特性のみ).....	8-12
Serial Poll Timeout(シリアルポールタイムアウト、デバイス特性のみ).....	8-12
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).....	8-12
GPIB-PCII/IIA Mode Switch(GPIB-PCII/IIAモード切替え).....	8-12
wibconfのデフォルト構成.....	8-13
wibconfの終了.....	8-13
エラーのチェック.....	8-13
wibconfのバッチモード.....	8-14

## 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

## 付録 C カスタマーコミュニケーション

## 用語集

## 索引

### 図

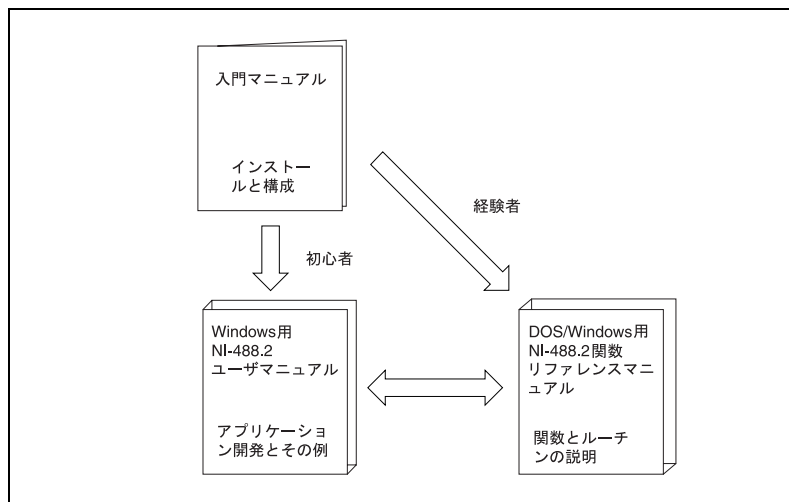
図 1-1	GPIB アドレスビット.....	1-2
図 1-2	システムのリニア構成とスター構成.....	1-5
図 1-3	マルチボードシステムのセットアップ例.....	1-6
図 1-4	NI-488.2 ソフトウェアと Windows との関係.....	1-9
図 2-1	例 1 のプログラムフローチャート.....	2-3
図 2-2	例 2 のプログラムフローチャート.....	2-5

図 2-3	例 3 のプログラムフローチャート .....	2-6
図 2-4	例 4 のプログラムフローチャート .....	2-8
図 2-5	例 5 のプログラムフローチャート .....	2-10
図 2-6	例 6 のプログラムフローチャート .....	2-11
図 2-7	例 7 のプログラムフローチャート .....	2-12
図 2-8	例 8 のプログラムフローチャート .....	2-14
図 2-9	例 9 のプログラムフローチャート .....	2-15
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エル .....	3-7
図 3-2	NI-488.2 ルーチンを使った一般的なプログラムシエル .....	3-12
図 6-1	GPIB Spy のメインウィンドウ .....	6-2
図 6-2	GPIB Spy の ibwrt の Call Details ウィンドウ .....	6-3
図 8-1	上位レベルの wibconf .....	8-3
図 8-2	下位レベルの wibconf .....	8-5
<b>表</b>		
表 1-1	GPIB ハンドシェイク線 .....	1-3
表 1-2	GPIB インタフェース管理線 .....	1-4
表 3-1	ステータスワード (ibsta) のレイアウト .....	3-4
表 4-1	GPIB エラーコード .....	4-5
表 5-1	wibic におけるデバイスレベル NI-488 関数の構文 .....	5-6
表 5-2	wibic におけるボードレベル NI-488 関数の構文 .....	5-7
表 5-3	wibic における NI-488.2 ルーチンの構文 .....	5-9
表 5-4	wibic の補助関数 .....	5-16
表 8-1	wibconf 起動オプション .....	8-2
表 8-2	wibconf のバッチモードのコマンドペア .....	8-15

# このマニュアルについて

このマニュアルでは、Windows 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 Windows バージョン 3.0 以上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 Windows 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 Windows 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、「Windows 用 NI-488.2 ユーザマニュアル (部品番号 320750-01)」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、プログラムの役割、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「wibic - Windows 用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 wibic について説明します。
- 第6章「GPIB Spy」では、GPIB Spy ユーティリティを使ってアプリケーションプログラムをモニタし、デバッグする方法を説明します。
- 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「wibconf-Windows 用インタフェースバス構成ユーティリティ」では、Windows 用 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 wibconf 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「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 ( <b>bold</b> )	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 ( <i>italic</i> )	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 ( <b>bold italic</b> )	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。
( <b>bold monospace</b> )	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。
( <i>italic monospace</i> )	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 C「カスタマーコミュニケーション」にあります。

---

## 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ー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 には電氣的、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。

## トーカー、リスナ、コントローラ

---

GPIB デバイスはトーカー、リスナ、コントローラのいずれかとして動作します。 トーカー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、いくつか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ー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ー、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)コントローラといえます。 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ナアドレスを 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 1-1 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 1-1 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。トーカーが高速転送 (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)	トーカー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。
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)	トーカーは 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 1-2 に示します。

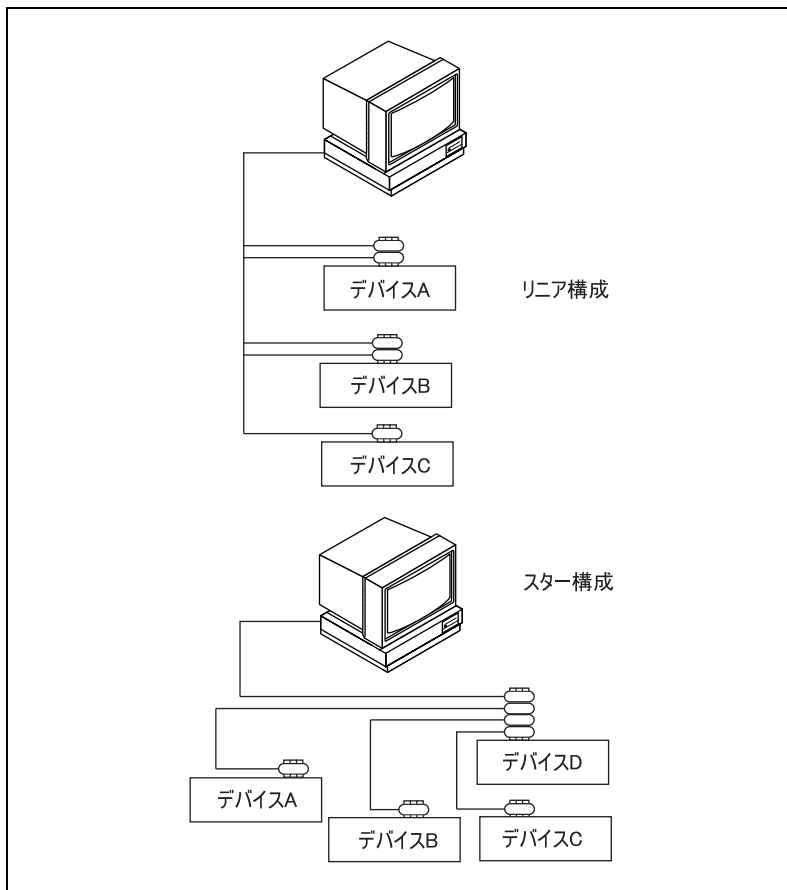


図 1-2 システムのリニア構成とスター構成

## 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 1-3 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。

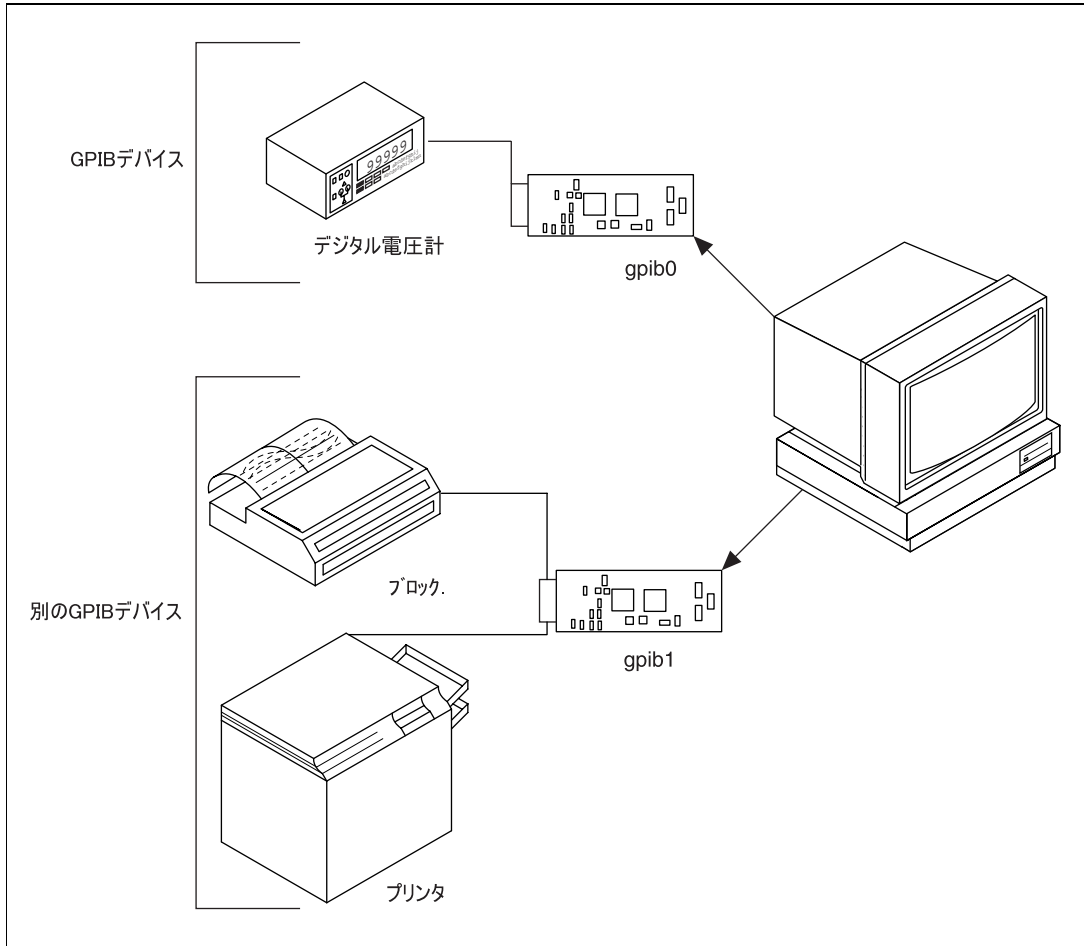


図 1-3 マルチボードシステムのセットアップ例

## 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、Windows 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 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。
- `setup.exe`: NI-488.2 ソフトウェアをインストールする Windows セットアップ実行ファイルです。
- `gpib.dll`: NI-488 関数および NI-488.2 関数を実行する 16 ビットのダイナミックリンクライブラリです。
- `gpib.ini`: NI-488.2 ソフトウェアのデフォルト構成が入った GPIB 構成情報ファイルです。
- `nivgpibd.386`: ナショナルインスツルメンツの仮想 GPIB デバイスです。DMA を使用して、メモリの 16MB を超えた位置にあるバッファを転送することができます。セットアッププログラムでこの仮想デバイスをインストールします。
- `nivdmad.386`: Windows 3.0 の 386 拡張モードで使用する仮想 DMA デバイス (`vdmad`) の改良バージョンです。Windows 3.0 を使用している場合は、セットアッププログラムでこの仮想デバイスをインストールします。
- `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を確認するのに使うプログラムです。

- `wibtest.exe` : Windows 用 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。
- `wibic.exe` : 対話式制御プログラムで、NI-488.2 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 のルーチンの勉強にもなり、計測器や他の GPIB デバイスのプログラム作成にも参考になります。
- `wibconf.exe` : ソフトウェア構成プログラムで、Windows 用 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。
- `gpibspy.exe` : GPIB の動作モニタプログラム。デバッグツールであり、お使いの Windows アプリケーションからの NI-488.2 の呼び出しをモニタできます。

## 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。
- `gpib.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しなければ、プログラムは NI-488.2 ダイナミックリンクライブラリにアクセスできるようになりません。
- `windecl.h` : インクルードファイルです。NI-488 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。

## Visual Basic 言語ファイル

付属のディスクには以下の Microsoft Visual Basic 用の言語ファイルが含まれています。

- `readme.vb` : Visual Basic 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。
- `vbib.bas` : `gpib.dll` を呼び出す NI-488 関数および NI-488.2 ルーチンを定義する、Visual Basic(バージョン 1.0 以上) 言語ソースファイルです。
- `niglobal.bas` : いくつかの初期設定定数の宣言を含み、グローバル変数を定義する Visual Basic のグローバルモジュールです。

## 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、NI-488.2 ソフトウェアがサポートする各言語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 Windows との関係

NI-488.2 ドライバは、Windows 用 GPIB ドライバの実行時にロードされる、標準 16 ビット Windows DLL です。Windows 用 NI-488.2 ドライバは実行していた GPIB アプリケーションの最後の 1 つが終了するとアンロードされます。

NI-488.2 ソフトウェアが Windows 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 1-4 に示します。

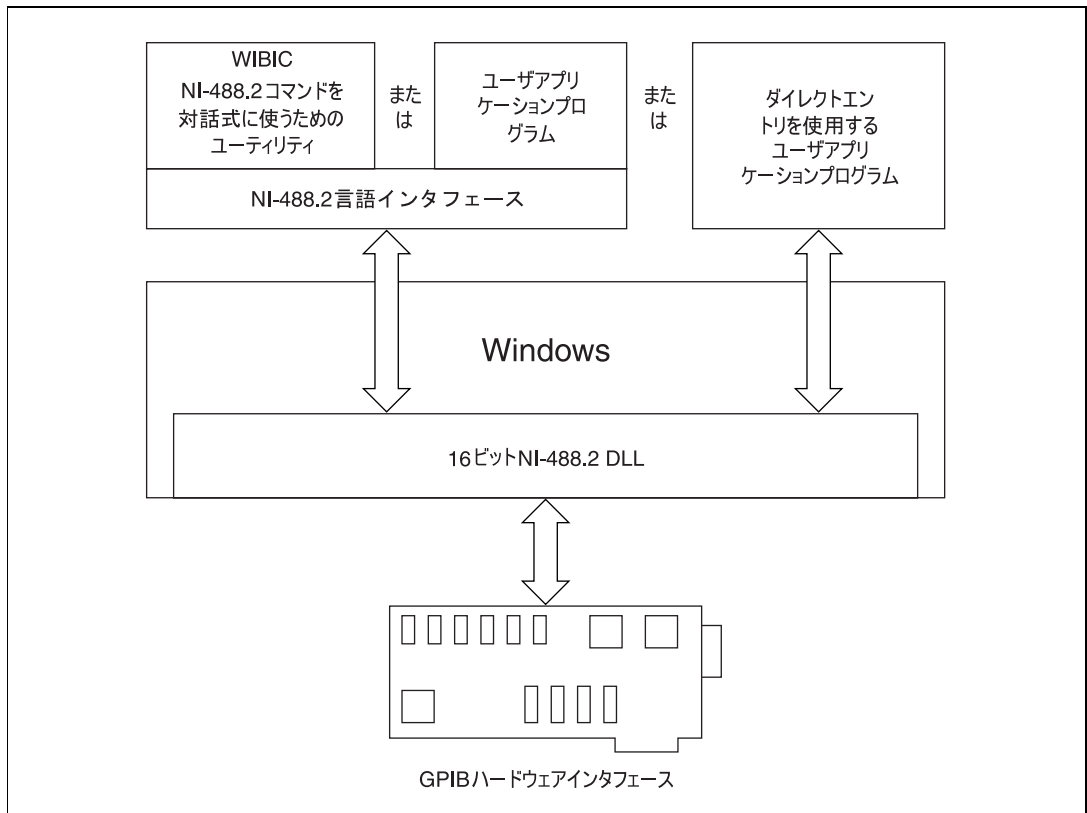


図 1-4 NI-488.2 ソフトウェアと Windows との関係



## NI-488.2 ドライバのアンロードと再ロード

---

Windows 用 NI-488.2 ドライバはダイナミックリンクライブラリ (DLL) で、自動的にロードおよびアンロードされます。最初の Windows アプリケーションが NI-488.2 DLL にアクセスすると、DLL がメモリにロードされます。NI-488.2 DLL を使用している最後の Windows アプリケーションが終了すると、DLL はメモリからアンロードされます。

NI-488.2 DLL は、仮想デバイスドライバ `nivgpibd.386` のサービスを使用することもあります。コンピュータに 16MB を超える RAM があり、DMA を使用して GPIB 計測器への書き込みや読み取りを行う場合、`nivgpibd.386` デバイスをインストールする必要があります。このデバイスがインストールされていないと、EDMA エラーが返されます。

以下の手順で `system.ini` ファイルを変更して `nivgpibd.386` デバイスを取り除くこともできます。

1. Windows ディレクトリ (通常 `c:\windows`) にある `system.ini` ファイルを探します。
2. `(386Enh)` セクションから次の行を見つけます。  
`device = drive:\path\nivgpibd.386`  
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です (例 `c:\at-gpibw`)。
3. 次のようにコマンド行の頭にセミコロンを追加して、注釈行に変更します。  
`;device = drive:\path\nivgpibd.386`
4. Windows を再起動すれば、`system.ini` に行った変更が有効になります。

## 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と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 Visual 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ー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。

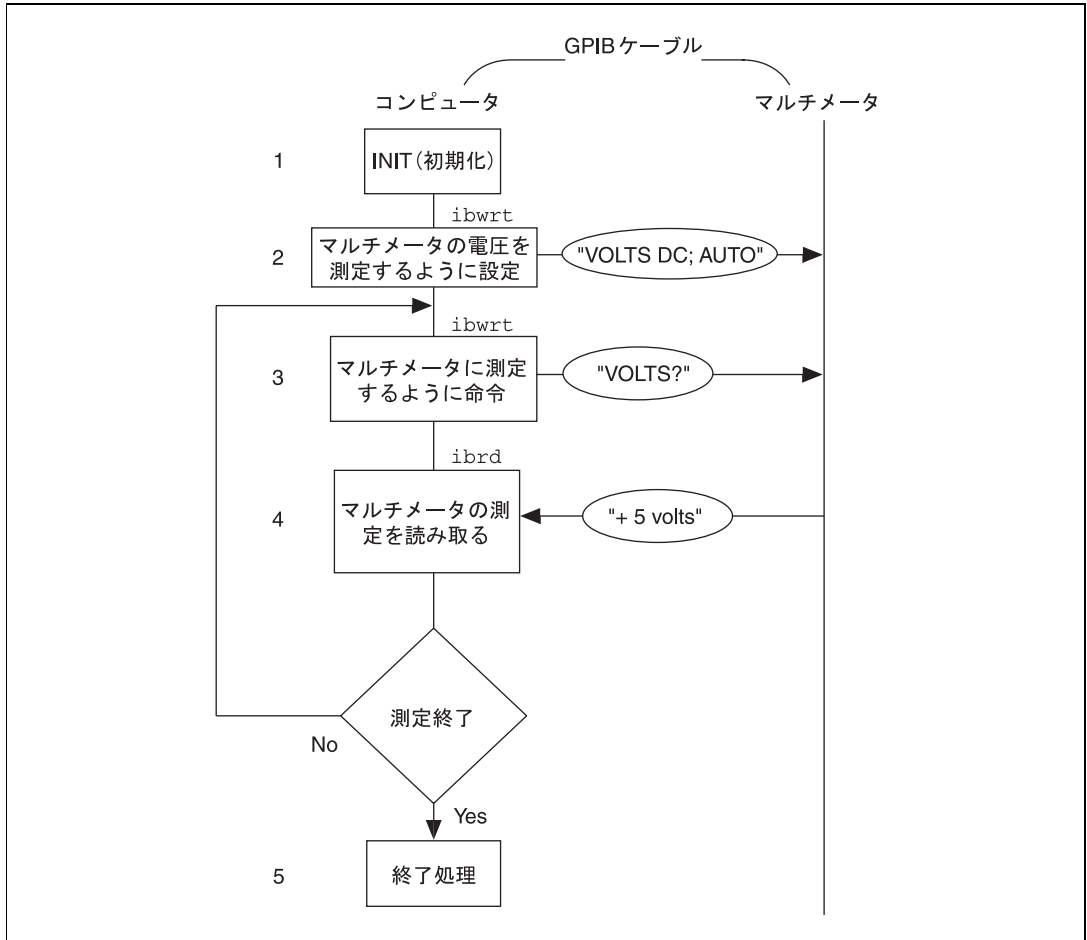


図 2-1 例 1 のプログラムフローチャート

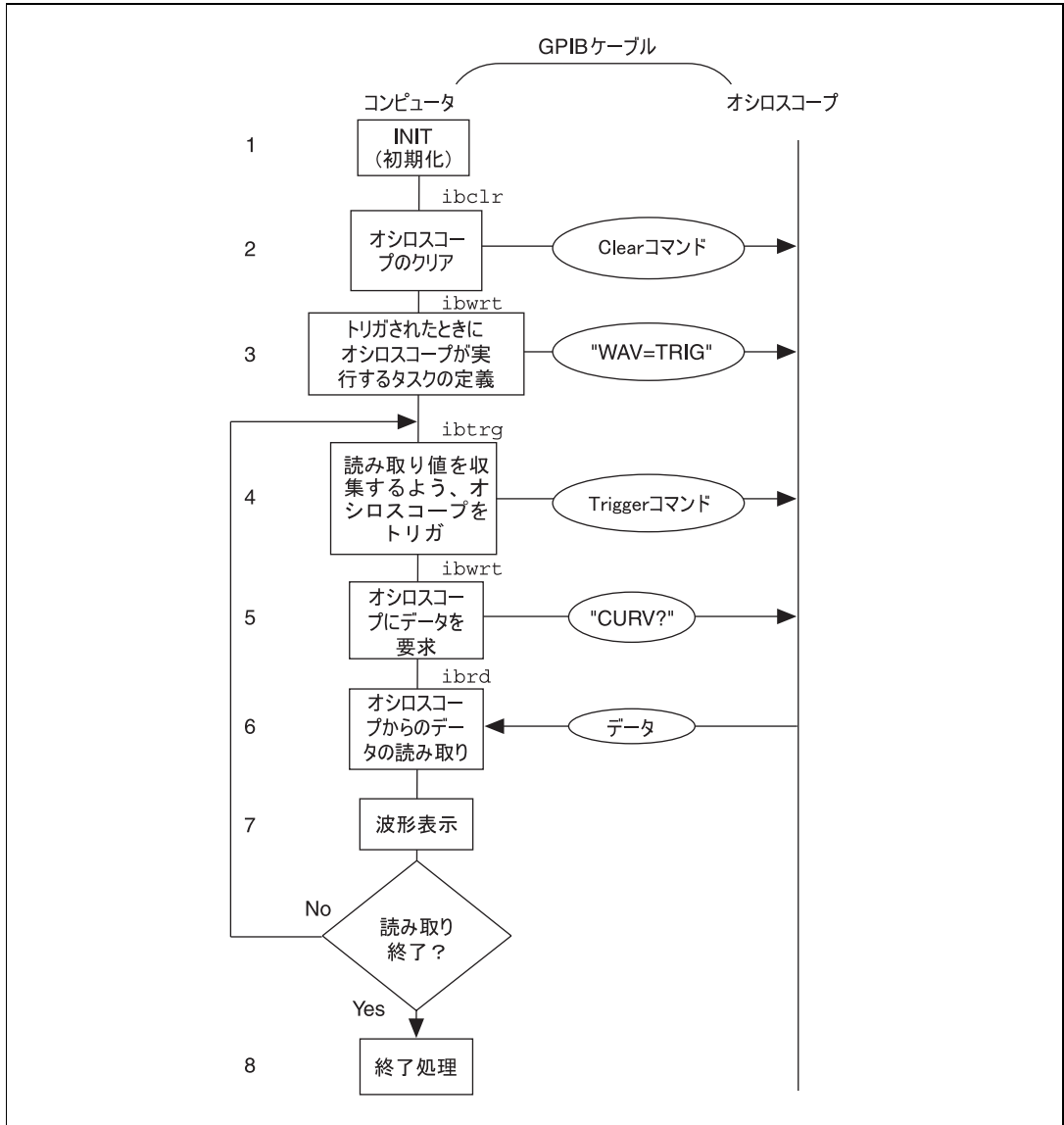
## 例 2: デバイスのクリアとトリガ

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。



### 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おり、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。

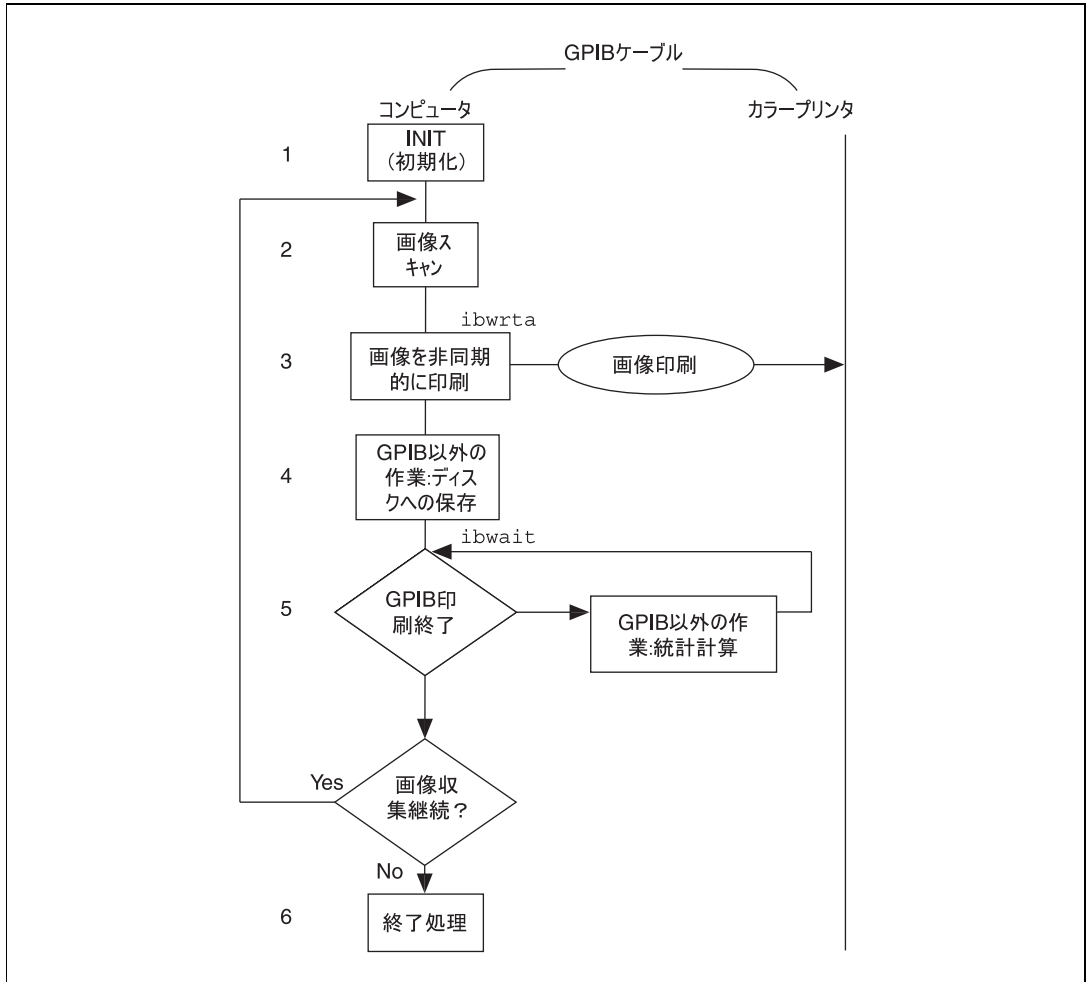


図 2-3 例 3 のプログラムフローチャート

## 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取り込み、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。

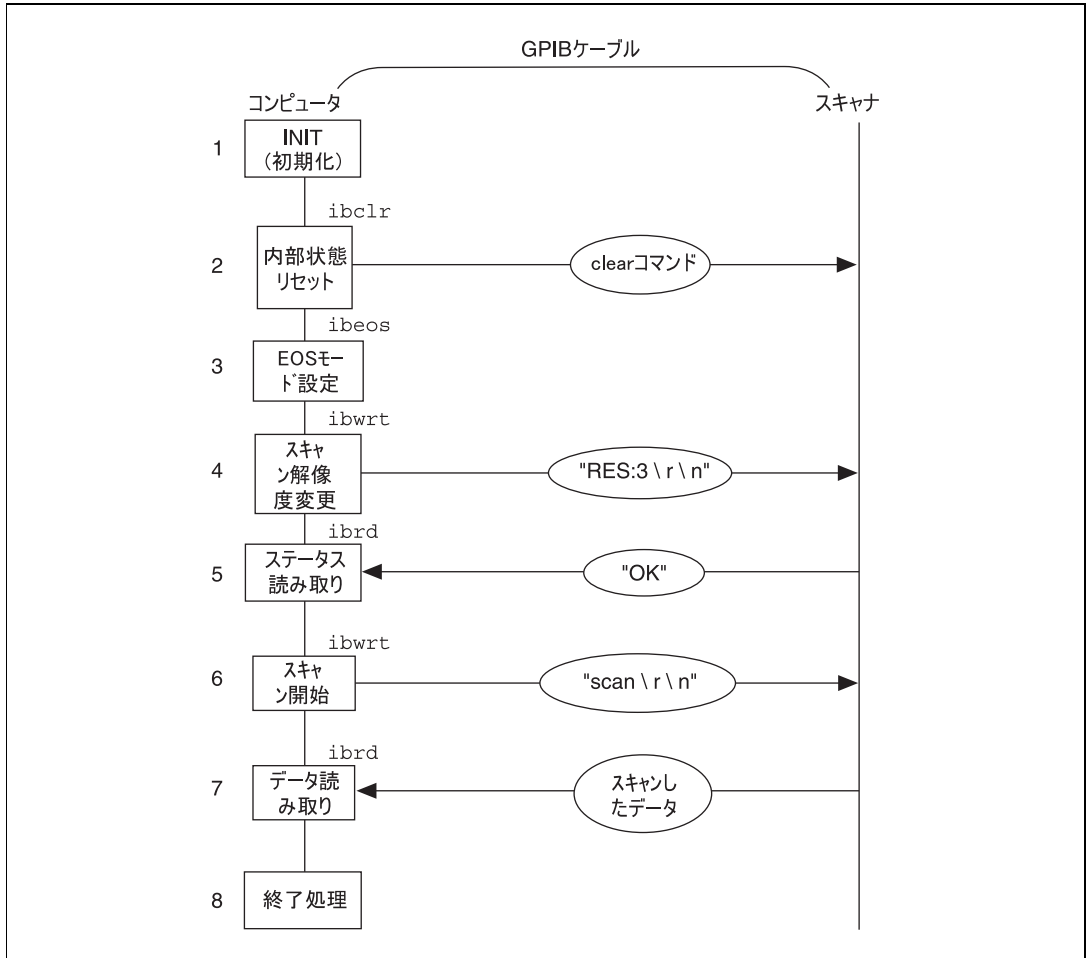


図 2-4 例 4 のプログラムフローチャート

## 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

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生し、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フィルム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リングを行い、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。シリアルポーリング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。

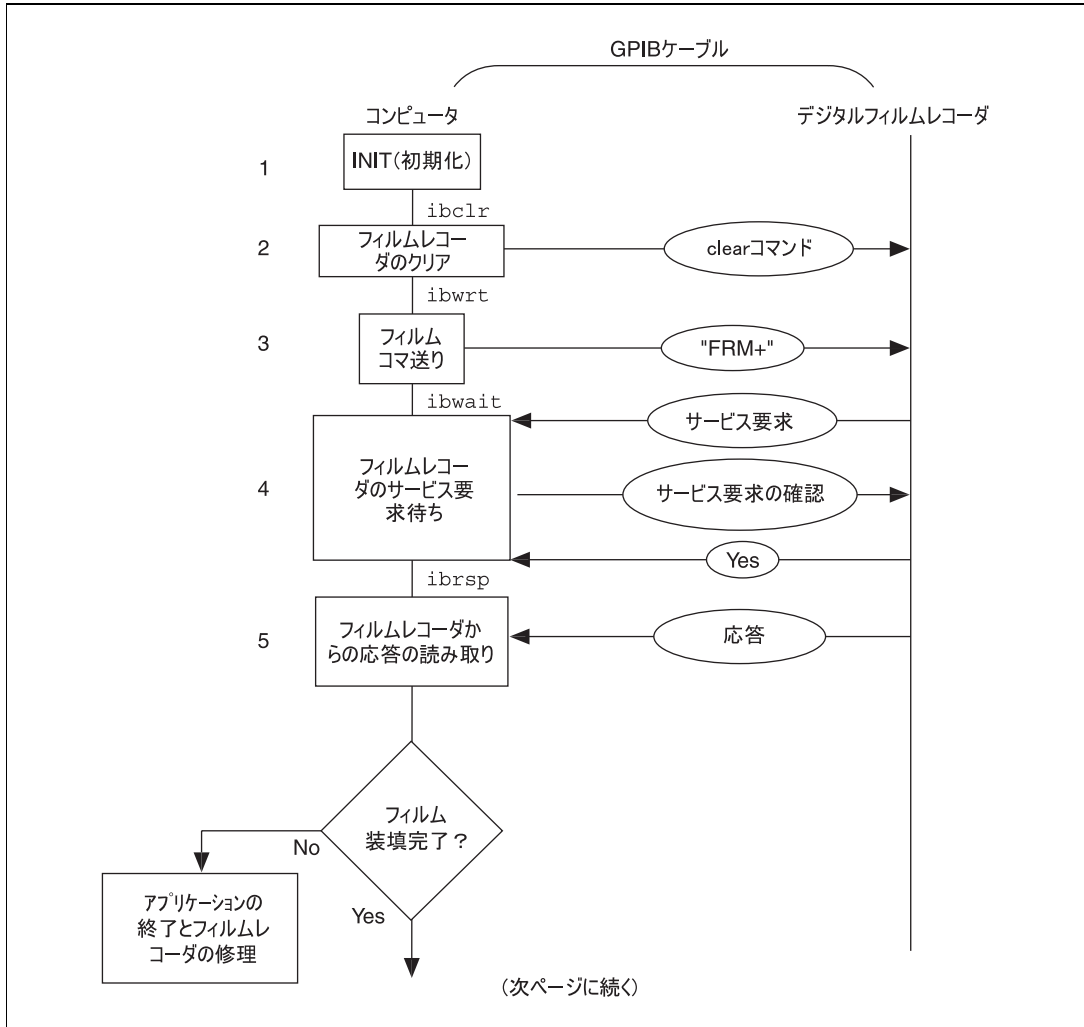


図 2-5 例5のプログラムフローチャート

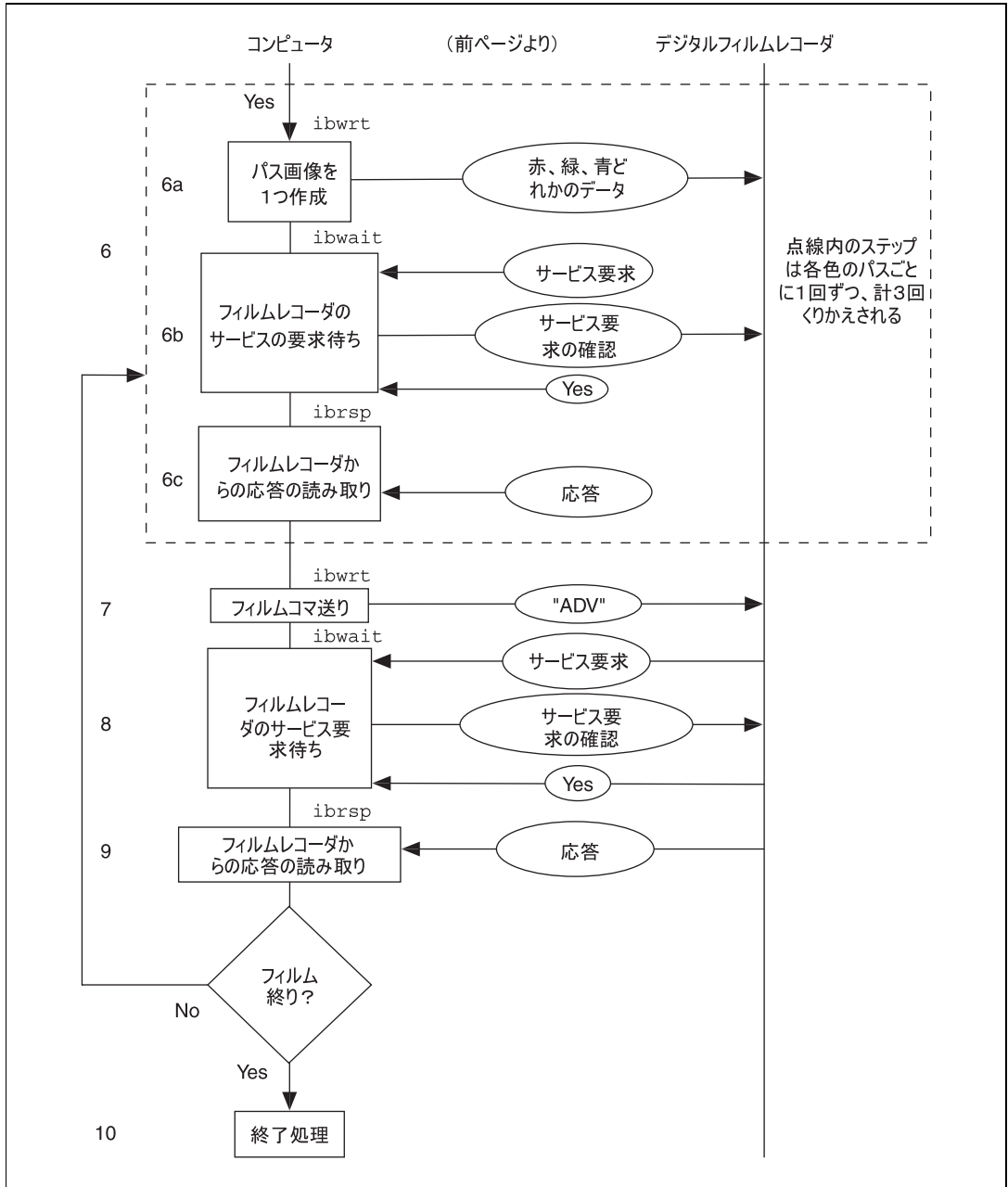


図 2-5 例 5 のプログラムフローチャート

## 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トを使う場合を考えます。引張り強度テストは何台もあり、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。テストが大きいので、技師がそれぞれのテストのアドレススイッチに手を伸ばすのは大変です。ずっとこのテストを使っていくことを考えると、各テストの 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を検出し、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。

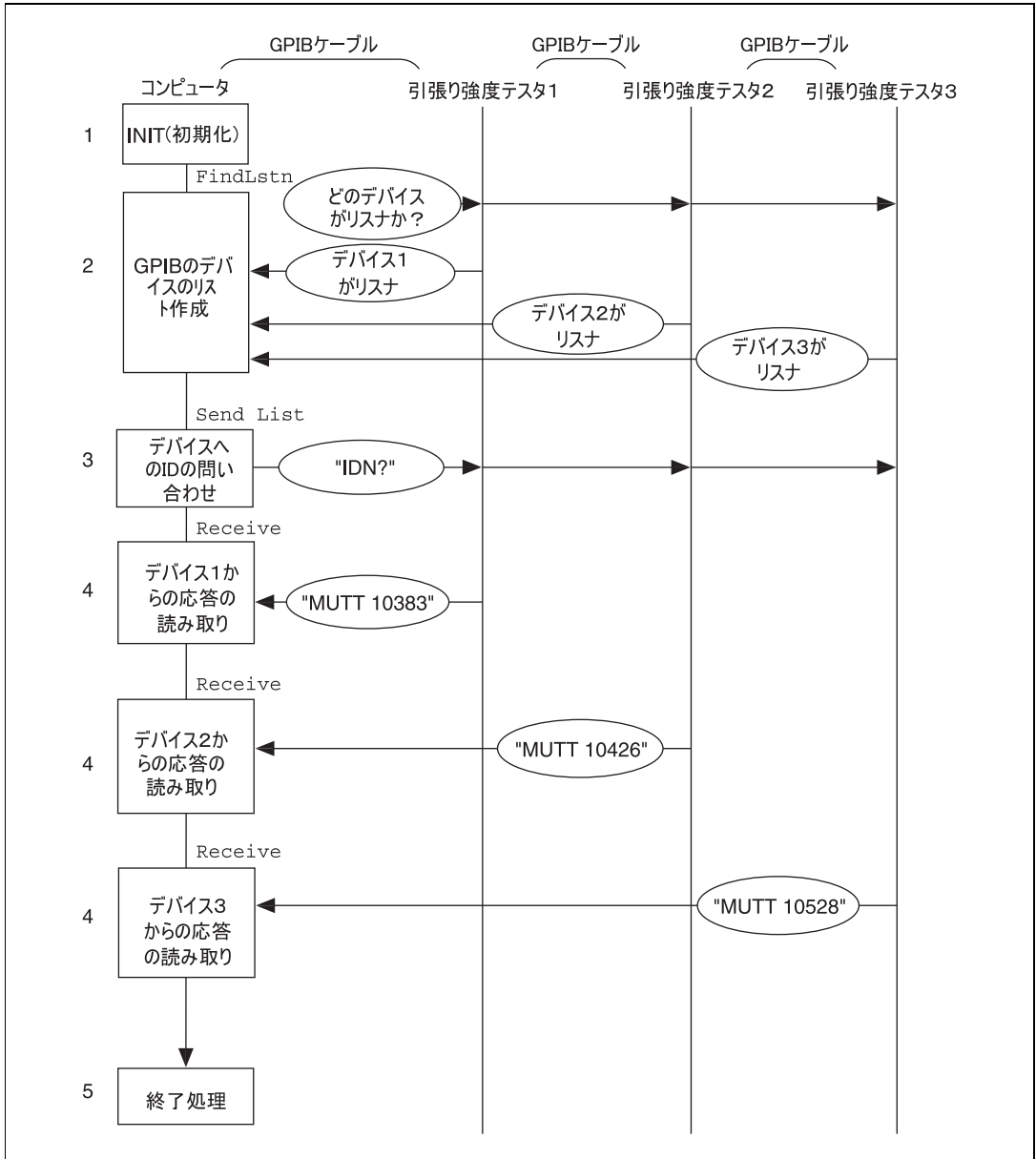


図 2-6 例6のプログラムフローチャート



## 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。

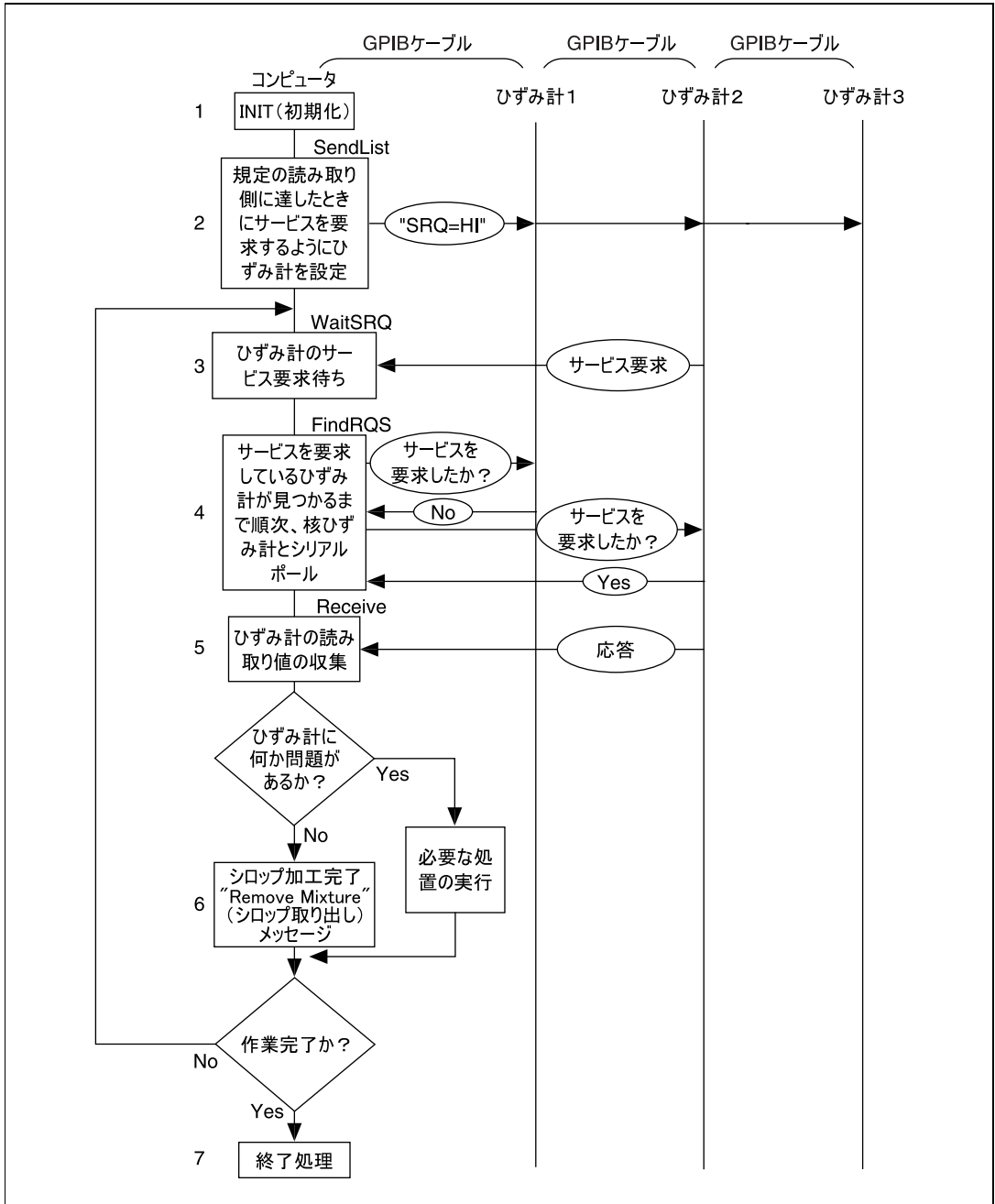


図 2-7 例 7 のプログラムフローチャート

## 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、温度スレッシュホールドも設定します。温度変換器は、規定の温度スレッシュホー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、温度がスレッシュホー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ュホー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、パラレルポールに参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。

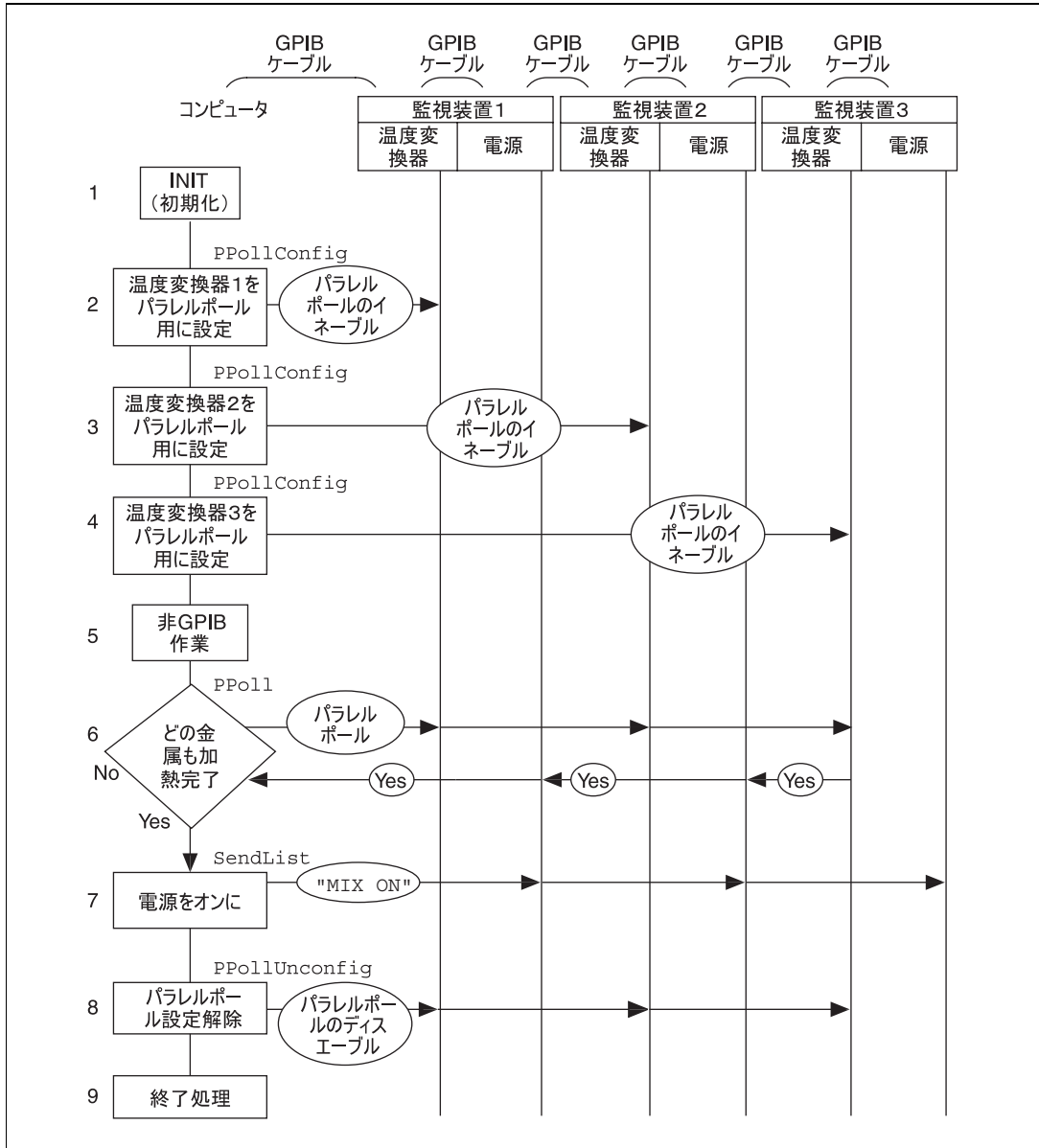


図 2-8 例 8 のプログラムフローチャート

## 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、トーカー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。デバイスがトーカー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。

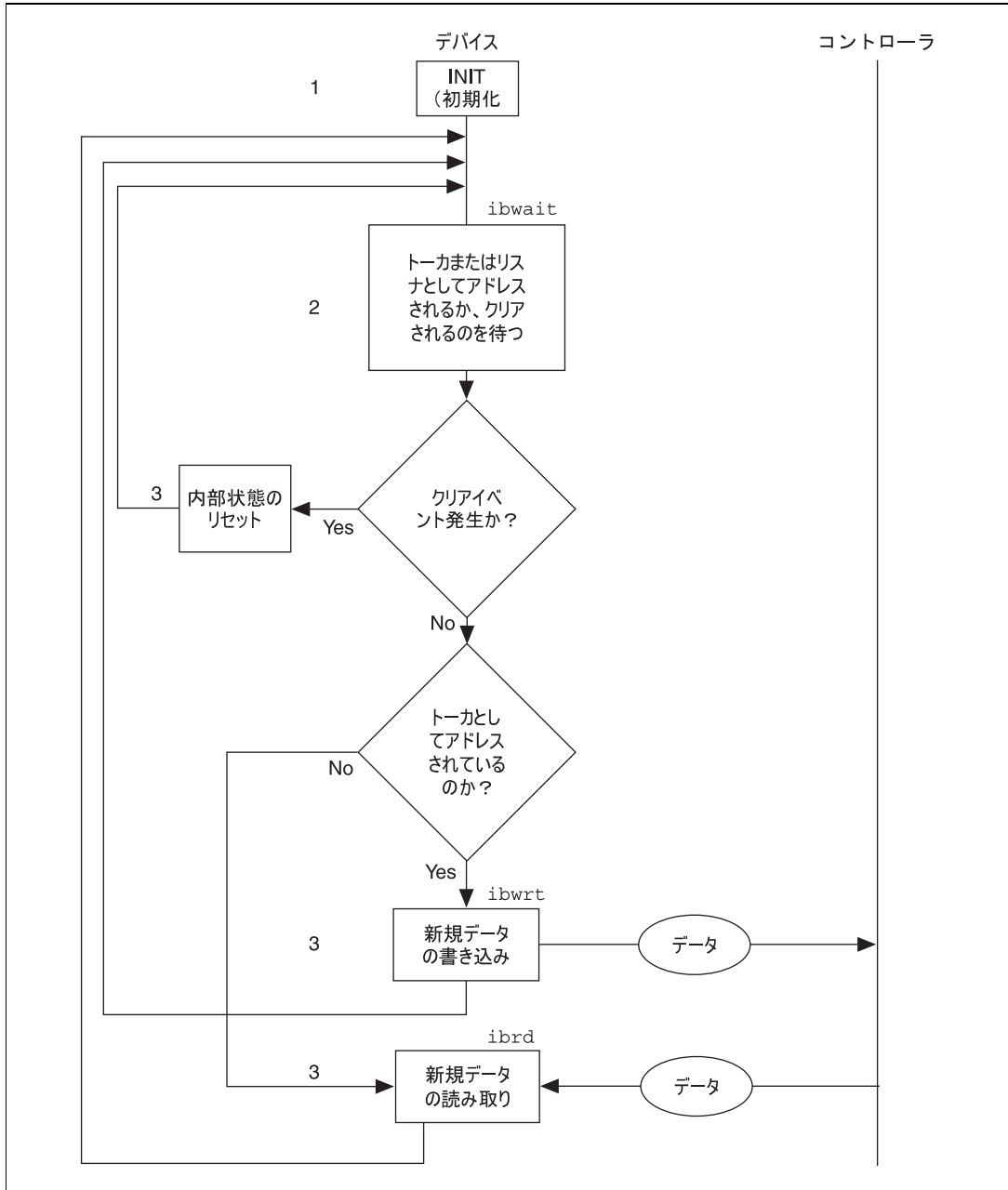


図 2-9 例9のプログラムフローチャート

---

# 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。

## NI-488.2 DLL へのアクセス方法の選択

---

アプリケーションは、C 言語または Visual Basic 用 NI-488.2 言語インタフェースを使用して、あるいは直接 `gpib.dll` にアクセスすることによって、NI-488.2 ダイナミックリンクライブラリ (`gpib.dll`) にアクセスできます。NI-488.2 言語インタフェースを使用すれば、最も速く簡単にアクセスすることができ、ほとんどのアプリケーションで使用できます。

C 言語や Visual Basic 以外の言語で `gpib.dll` にアクセスする場合、または `gpib.dll` にアクセスするために特別な条件がある場合は、ダイレクトエントリ関数を使用する必要があります。これらの関数は、次の 2 つの点を除けば NI-488 および NI-488.2 呼び出しと同じです。

- 各関数名の前に `DLL` という文字を付ける必要があります。たとえば、NI-488 関数 `ibwrt` を使用する代わりに、ダイレクトエントリ関数 `DLLibwrt` を使用します。
- 各関数の取る引き数の数が 3 つ増えます。最後の 3 つのパラメータは、順に、`ibsta`、`iberr`、`ibcntl` 変数を示すポインタです。アプリケーションでダイレクトエントリ関数を使用するには、独自の `ibsta`、`iberr` および `ibcntl` 変数を宣言する必要があります。

## NI-488 関数または NI-488.2 ルーチンの選択

---

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リング、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せん。

## 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。

## 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。ステータスワード (*ibsta*)、エラー変数 (*iberr*)、カウント変数 (*ibcnt*, *ibcnt1*) 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。

### 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
SROLL	10	400	brd	コントローラがボードをシリアルポーリングしました
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	トークです
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 のニモニク定数が記録されています。その数値またはニモニク定数を使えば、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();

}
```

ibcnt1 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、ibcnt1 は、文字列の終わりを示すヌル文字の位置になります。

## wibic 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 wibic ユーティリティを使う必要がある場合があります。wibic(Windows 用インタフェース対話式制御)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。wibic を使って、NI-488 関数や NI-488.2 ルーチンによる計測器との通信方法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。wibic の使用方法や詳しい使用例については、第5章「wibic - Windows 用インタフェー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 wibic 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 C 言語で書いてあります。NI-488.2 ソフトウェアには、C 言語のソースコード (devsamp.c) と、Visual Basic のソースコード (devsamp.bas) および C 言語のダイレクトエントリ関数 (dlldev.c) と Visual Basic のダイレクトエントリ関数 (dlldev.bas) が入っています。

### 組み込む項目

- GPIB ヘッドファイル。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) が入っていません。
- 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 プログラムシエル

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。

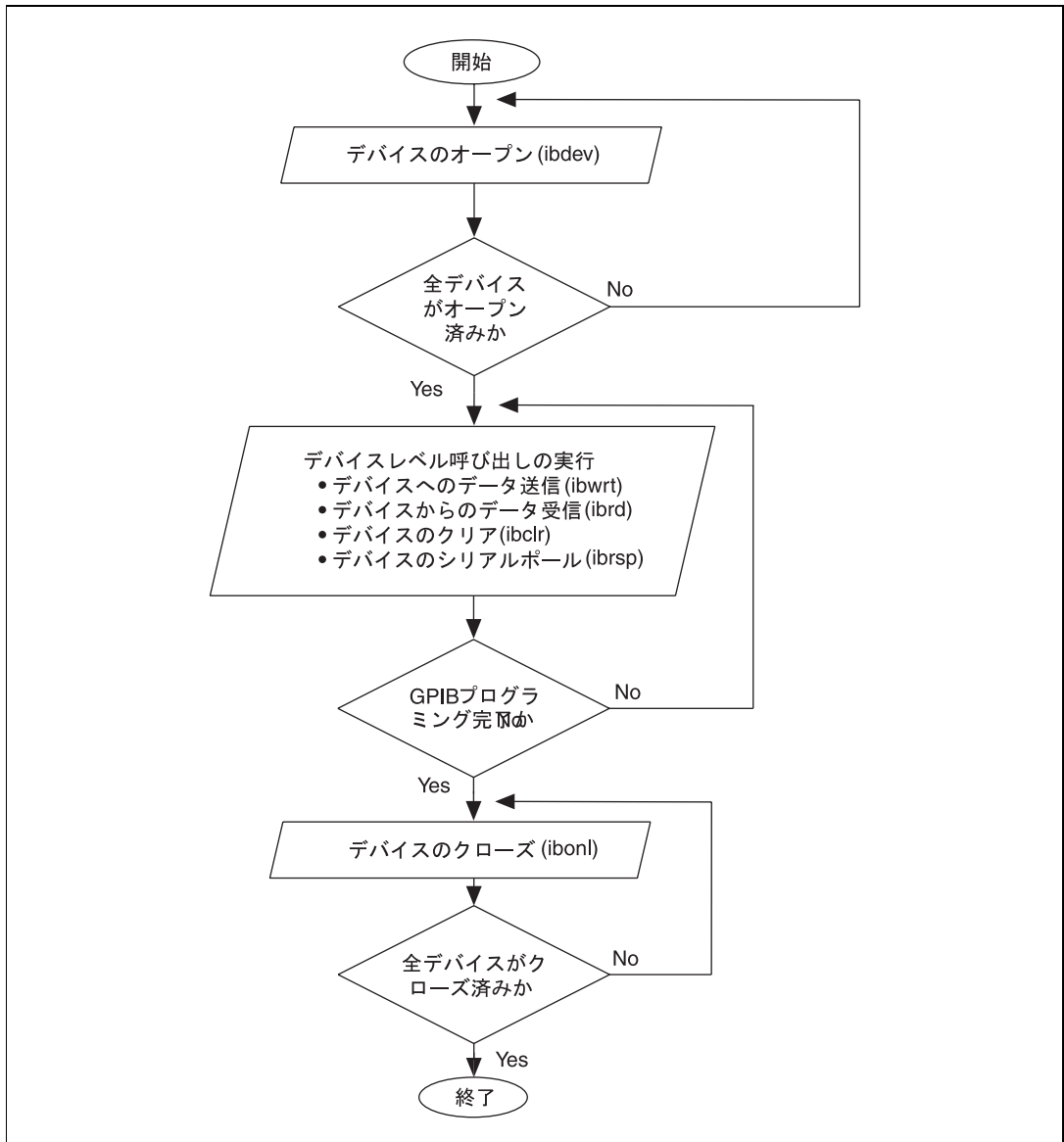


図 3-1 NI-488 デバイス関数を使った一般的なプログラムシエル

## 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ラして、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[ibcnt1] = '\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、`ibon1` 関数でデバイスをオフラインにします。

```
ibon1 (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 C 言語で書いてあります。NI-488.2 ソフトウェアには、C 言語のソースコード (`samp4882.c`) と Visual Basic のソースコード (`samp4882.bas`) および C 言語のダイレクトエントリ関数 (`d114882.c`) と Visual Basic のダイレクトエントリ関数 (`d114882.bas`) が入っています。

## 組み込む項目

- GPIB ヘッドファイル。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。

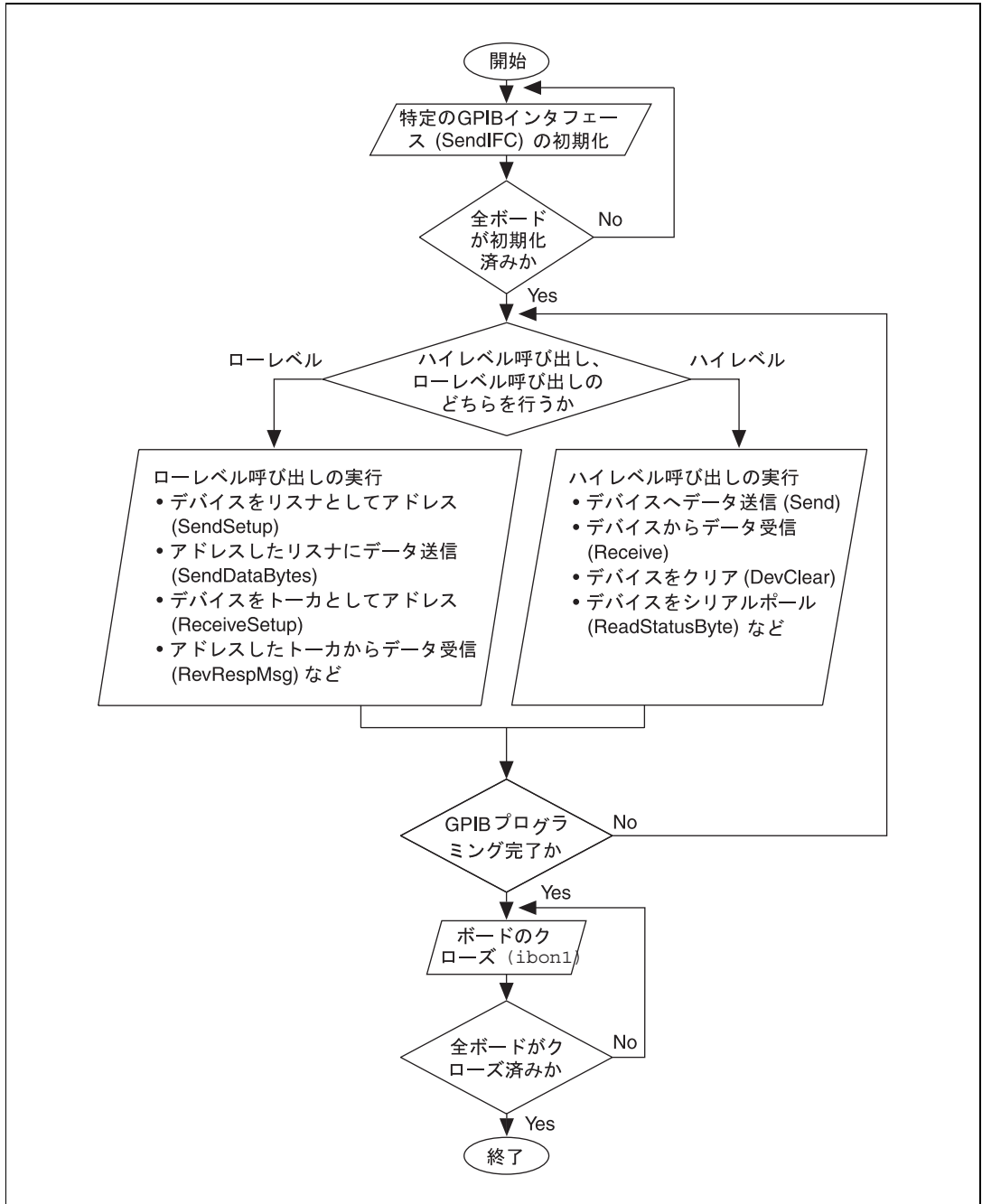


図 3-2 NI-488.2 ルーチンを使った一般的なプログラムシェル

## 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 (strcmp(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リングし、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[ibcctl] = '\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);
```

# アプリケーションのコンパイル、リンク、実行

---

## Microsoft C

C言語の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 "windecl.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 `gpib.lib` とリンクします。Microsoft CでQuickWin C言語アプリケーション `cprog` をコンパイルしてリンクするには、次のコマンドを使います。

```
cl /Mq cprog.c gpib.lib
```

Microsoft Cの `/Mq` オプションを使用すると、QuickWinアプリケーションが作成されます。アプリケーションを実行するには、Program Manager ウィンドウのFileメニューからRunオプションを選択します。ダイアログボックスが現れたら、コンパイルしたプログラムのパスと名前を入力します。

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、Windows用GPIBアプリ

ケーションモニタ GPIB Spy を使います。GPIB Spy については、第6章「GPIB Spy」に説明があります。

## Visual Basic

Visual Basic のアプリケーションを実行する前に、アプリケーションプロジェクトファイルに `niglobal.bas` ファイルと `vbib.bas` ファイルを入れてください。

Microsoft Visual Basic 1.0 を使用している場合、`niglobal.bas` を変更して、定数 `true` と `false` を次のように定義する必要があります。

```
global const true = -1
```

```
global const false = 0
```

Microsoft Visual Basic 2.0 以上では `true` と `false` は予約語ですから、これらの定数は `niglobal.bas` では定義されていません。

Visual Basic のアプリケーションを書き終えたら、Run メニューから Start オプションを選択して、プログラムを実行します。

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、Windows 用 GPIB アプリケーションモニタ GPIB Spy を使います。GPIB Spy については、第6章「GPIB Spy」に説明があります。

## C 言語によるダイレクトエントリ

C 言語の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。

```
#define GPIB_DIRECT_ACCESS

#ifdef __cplusplus

extern "C" {

#endif

#include "windecl.h"

#include "dllfunc.h"          /* Direct entry prototypes */

int  ibsta, iberr;          /* NI-488.2 global status
variables */
```

```

long  ibcntl;

#ifdef  __cplusplus
}

#endif

```

変数 `GPIOB_DIRECT_ACCESS` を定義して、`windecl.h` の関数のプロトタイプやグローバルステータス変数の外部宣言がアプリケーションプログラムに含まれないようにします。グローバルステータス変数はアプリケーションプログラムで定義する必要があります。

ユーザは、アプリケーションプログラムで使用する関数のダイレクトエントリのプロトタイプを含むファイル (例: `dllfunc.h`) を作成する必要があります。NI-488 関数および NI-488.2 ルーチンの C 言語によるダイレクトエントリ構文については、「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。定義済の C++ macro `__cplusplus` により、ANSI C と C++ 間の移植が可能になります。

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。Microsoft C で QuickWin C 言語アプリケーション `cprog` をコンパイルおよびリンクするには、次のコマンドを使います。

```
cl /Mq cprog.c cprog.def
```

Microsoft C の /Mq オプションを使用すると、QuickWin アプリケーションが作成されます。モジュール定義ファイル `cprog.def` には、アプリケーションプログラムで使用するダイレクトエントリポイントをリストした `IMPORTS` セクションがあります。

アプリケーションを実行するには、Program Manager ウィンドウの File メニューから Run オプションを選択します。ダイアログボックスが現れたら、コンパイルしたプログラムのパスと名前を入力します。

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、Windows 用 GPIB アプリケーションモニタ GPIB Spy を使います。GPIB Spy については、第 6 章「GPIB Spy」に説明があります。

## Visual Basic によるダイレクトエントリ

アプリケーションプロジェクトファイルには、`niglobal.bas` ファイルを入れてください。アプリケーションプログラムで使用する関数のダイレクトエントリのプロトタイプを含むファイル (例: `dllfunc.bas`) を作成

する必要があります。アプリケーションプロジェクトファイルにダイレクトエントリファイルを入れてください。NI-488 および NI-488.2 関数の Visual Basic によるダイレクトエントリ構文については、「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。

Microsoft Visual Basic 1.0 を使用している場合、niglobal.bas を変更して、定数 true と false を次のように定義する必要があります。

```
global const true = -1
```

```
global const false = 0
```

Microsoft Visual Basic 2.0 以上では true と false は予約語ですから、これらの定数は niglobal.bas では定義されていません。

Visual Basic のアプリケーションを書き終えたら、Run メニューから Start オプションを選択して、プログラムを実行します。

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、Windows 用 GPIB アプリケーションモニタ GPIB Spy を使います。GPIB Spy については、第 6 章「GPIB Spy」に説明があります。

# 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。

## wibtest 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 wibtest を実行してください。wibtest 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。wibtest 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 wibtest 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。gpibx 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

wibtest プログラムは NI-488.2 ドライバ gpib.dll および NI-488.2 構成情報ファイル gpib.ini が存在するかどうかテストします。gpib.dll が Windows ディレクトリにないと、次のような内容の警告ボックスが現れます。

```
File Error: Cannot find GPIB.DLL
```

Close ボタンを押してください。wibtest は、テストがエラーの時のようなエラー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.dll ファイルが Windows ディレクトリ (通常 c:\windows) にあるかどうか確認してください。

gpib.ini ファイルが Windows ディレクトリにない場合も同じエラーメッセージが表示されます。gpib.dll または gpib.ini が見つからない場合は、NI-488.2 ソフトウェアをもう 1 度インストールしてください。

### ボードの存在確認テスト

gpibx 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。

- wibconf 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 gpidx 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。wibconf 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。

## 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ト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`, `ibcnt1` 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。

## wibic 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、wibic 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。wibic 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。wibic の詳細については、第 5 章「wibic - Windows 用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。

## GPIB Spy 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 GPIB Spy 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。GPIB Spy では、Windows 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 GPIB Spy を構成することもできます。GPIB Spy の詳細については、第 6 章「GPIB Spy」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

エラーモニック	iberr 値	意味
EDVR	0	Windows 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)
EDMA	8	DMA エラー
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ーフロー
ESRQ	16	SRQ がオフになりません
ETAB	20	表の異常

## 構成エラー

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、wibconf 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/O アドレス、割り込みレベル、DMA チャンネル以外の設定を変更するときは、wibconf 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-488.2 関数リファレンスマニュアル」の `ibconfig` の説明を参照してください。

ハードウェアの構成をテストするには、`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。

## 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 `wibic` 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ク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、方法の一つとして、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できます。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場合があります。IEEE488.2 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。ソフトウェアの構成変更の詳細については、第 8 章「`wibconf - Windows 用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-488.2 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ない。

```
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 `wibtest` 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「`wibtest` 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: 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:`wibic` はどうやって使うのですか？

A:`wibic` 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「`wibic` - Windows 用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:`wibconf` を起動すると現れるポップアップウィンドウには、何を入力すればいいのでしょうか？

A:このポップアップウィンドウは、`gpib.ini` ファイルのあるドライブとパスを入力するよう促しています。Windows がデフォルトのディレクトリ (`c:\windows`) にインストールされている場合は、`<Return>` キーを押すだけです。Windows がデフォルト以外のディレクトリにインストールされている場合は、Windows がインストールされているディレクトリへのパスとそのディレクトリを入力します。これで、`wibconf` は変更を行う正しい `gpib.ini` ファイルを見つけることができます (例 :`c:\win31\gpib.ini`)。

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/O アドレスが分かっている場合は、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/O 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:Windows3.0 の上に Windows3.1 をインストールすると Windows を起動できないのはなぜですか？

A:system.ini ファイルに、ロードする仮想デバイスについて矛盾する情報が入っていると、Windows は起動できません。この場合、system.ini ファイルはたいてい 2 台の仮想 DMA デバイスをロードしようとしています。system.ini ファイルでロードする仮想 DMA デバイスは 1 台だけでなければなりません。不必要な行を削除するか、行の先頭にセミコロンを追加して注釈行に変更してください。

仮想 DMA デバイスは、system.ini ファイルの (386Enh) セクションにロードされます。Windows 3.1 で使用するのに適したデバイスをロードするには、次の行を入力します。

```
device=*vdmad
```

Windows 3.0 で使用するのに適したデバイスをロードするには、次の行を入力します。

```
device=drive:\path\nivdmad.386
```

ここで、drive と path はハードディスク上の nivdmad.386 ファイルの位置を示します (例 :c:\at-gpibw)。

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、何が分かっている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 `wibttest` 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 C 「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。

---

# wibic - Windows 用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、wibic(Windows 用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。wibic 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

---

wibic 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。wibic 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。

wibic 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、wibic 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、wibic が必要なパラメータを要求してきます。

1. wibic を実行するため、Program Manager の GPIB グループにある wibic アイコンをダブルクリックします。スクリーンに以下の表示が現れます。

National Instruments

IEEE-488 Interface Bus Interactive Control Program  
(IBIC)

Copyright (c) 1993 National Instruments Corp. Version  
3.0 (Win16)

Version Date: May 28 1993 Version Time: 09:42:25

All rights reserved

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

:

2. `ibdev` を使って、`wibconf` プログラムでデバイスに割り当てられているデバイス名を見つけてみます。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] (cml)
```

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] (cml)
```

```
count: 35
```

あるいは

```
ud0: ibwrt "*RST; VAC; AUTO; TRIGGER 2; *SRE 16"
```

```
[0100] (cml)
```

```
count: 35
```

5. 次のようにデバイスをトリガします。

```
ud0: ibtrg
```

```
[0100] (cml)
```

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 cml)
```

あるいは

```
ud0: ibwait TIMO RQS
```

```
[0900] (rqs cml)
```

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] (cml)  
  
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] (cml)  
  
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] (cml)  
  
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] (cml)
```

あるいは

```
ud0: ibonl 0
```

```
[0100] (cml)
```

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。

## wibic 構文

---

wibic 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。

8 進バイト - 8 進値は、前にバックslash を置きます。たとえば 8 進 "40" は、\40 と示されます。

16 進バイト - 16 進値は、前にバックslash と x を置きます。たとえば 16 進 "40" は、\x40 と示されます。

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 wibic 構文

表 5-1 と表 5-2 は、wibic 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 wibic 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 wibic 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。
ibl n 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。

表 5-1 wibic におけるデバイスレベル NI-488 関数の構文

構文	説明
<code>ibpct</code>	制御を渡します。
<code>ibppc v</code>	パラレルポール構成。
<code>ibrd v</code>	データを読み取ります。v は読み取るバイト数です。
<code>ibrda v</code>	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。
<code>ibrdf filename</code>	データをファイルに読み取ります。filename はデータを入れるファイルのパス名です。
<code>ibrpp</code>	パラレルポールを実行します。
<code>ibrsp</code>	シリアルポールバイトを返します。
<code>ibsad v</code>	2 次アドレスを変更します。
<code>ibstop</code>	非同期的処理を打ち切ります。
<code>ibtmo v</code>	制限時間を変更 / ディスエーブルします。
<code>ibtrg</code>	選択したデバイスをトリガします。
<code>ibwait mask</code>	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。
<code>ibwrt string</code>	データを書き込みます。
<code>ibwrta string</code>	データを非同期的に書き込みます。
<code>ibwrtf filename</code>	ファイルからデータを書き込みます。filename はデータの入っているファイルのパス名です。

表 5-2 wibic におけるボードレベル NI-488 関数の構文

構文	説明
<code>ibask mn</code>	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。
<code>ibcac v</code>	アクティブコントローラにします。
<code>ibcmd string</code>	コマンドを送信します。
<code>ibcmda string</code>	コマンドを非同期的に送信します。

表 5-2 wibic におけるボードレベル NI-488 関数の構文

構文	説明
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。
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 filename	データをファイルに読み取ります。filename 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。

表 5-2 wibic におけるボードレベル NI-488 関数の構文

構文	説明
<code>ibstop</code>	非同期処理を打ち切ります。
<code>ibtmo v</code>	制限時間を変更 / ディスエーブルします。
<code>ibwait mask</code>	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二モニックです。
<code>ibwrt string</code>	データを書き込みます。
<code>ibwrta string</code>	データを非同期的に書き込みます。
<code>ibwrtf flname</code>	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 wibic 構文

以下の表 5-3 は wibic 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、wibic 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 wibic における NI-488.2 ルーチンの構文

ルーチン構文	説明
<code>AllSpoll addrlist</code>	複数のデバイスとのシリアルポールを行います。
<code>DevClear address</code>	デバイスをクリアします。
<code>DevClearList addrlist</code>	複数のデバイスをクリアします。
<code>EnableLocal addrlist</code>	ローカル制御をイネーブルします。
<code>EnableRemote addrlist</code>	リモート制御をイネーブルします。
<code>FindLstn addrlist v</code>	全リスナを検出します。
<code>FindRQS addrlist</code>	SRQ をアサートしているデバイスを検出します。
<code>PassControl address</code>	制御をデバイスに渡します。
<code>PPoll</code>	デバイスのパラレルポールを行います。
<code>PPollConfig address v v</code>	デバイスをパラレルポール用に構成します。

表 5-3 wibic における NI-488.2 ルーチンの構文

ルーチン構文	説明
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。
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

wibic 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] (cml)
```

```
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、wibic は該当するエラーモニタックを表示します。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 cml)
```

```
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、wibic 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、wibic 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 ルーチン呼び出しかによって、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「wibconf - Windows 用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せん。

`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 `wibic` 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、Windows が `gpib.dll` ファイルまたは `gpib.ini` ファイル、あるいはその両方を見つけれられません。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、`wibconf` 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                          . .

```

## wibic における一般的な NI-488.2 ルーチン

---

### Set

wibic 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] (cml c ic tacs)
```

```
count: 6
```

返されたステータスワードには `cml` ビットが入っています。`cml` 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

wibic で使える補助関数を表 5-4 にまとめて示します。

表 5-4 wibic 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: dev1 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 wibic 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 wibic 関数の入ったファイルのパス名です。

表 5-4 wibic の補助関数

関数	説明
<code>print string</code>	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。
<code>e</code>	終了または強制終了
<code>q</code>	終了または強制終了

## 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] (cml c ic atn)
```

```
gpib0: !
```

```
[0130] (cml c ic 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 回繰り返されるわけではありません。



## \$( 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\x6e\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。

# GPIB Spy

この章では、GPIB Spy ユーティリティを使用して、アプリケーションプログラムのモニタおよびデバッグを行う方法について説明します。

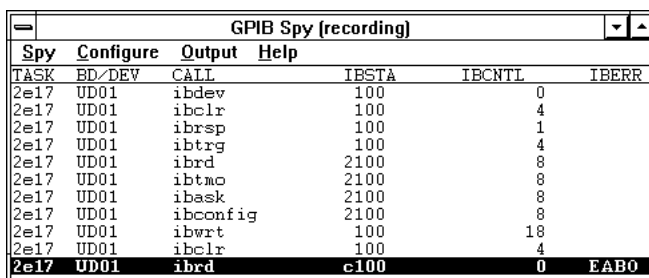
## 概要

GPIB Spy は、Windows アプリケーション用の GPIB アプリケーションモニタユーティリティです。このユーティリティは、Windows 用 NI-488.2 ドライバに対して実行された NI-488 および NI-488.2 呼び出しをモニタし、それらの呼び出しについての情報を記録し表示します。GPIB Spy アプリケーションを使用して、ご使用のアプリケーションおよび NI-488.2 ソフトウェア構成のエラーを検出し、分析することができます。

## GPIB Spy の起動

GPIB Spy を起動するには、Program Manager の GPIB グループで GPIB Spy アイコンをダブルクリックします。これで GPIB Spy が起動され、GPIB Spy のメインウィンドウが表示されます。GPIB Spy の稼働中に実行されたすべての NI-488 および NI-488.2 呼び出しが、GPIB Spy のメインウィンドウに記録されます。GPIB Spy のオンオフを切り換えるには、Spy メニューから SpyOn を選択します。GPIB Spy をオンにすると、GPIB Spy のメインウィンドウのタイトルは「GPIB Spy(recording)」に、オフのときは、「GPIB Spy」になります。

図 6-1 に、いくつかの NI-488 関数が記録された GPIB Spy のメインウィンドウを示します。



GPIB Spy (recording)					
Spy	Configure	Output	Help		
TASK	BD/DEV	CALL	IBSTA	IBCNTL	IBERR
2e17	UD01	ibdev	100		0
2e17	UD01	ibclr	100		4
2e17	UD01	ibrsp	100		1
2e17	UD01	ibtrg	100		4
2e17	UD01	ibrdr	2100		8
2e17	UD01	ibtmo	2100		8
2e17	UD01	ibask	2100		8
2e17	UD01	ibconfig	2100		8
2e17	UD01	ibwrt	100		18
2e17	UD01	ibclr	100		4
2e17	UD01	ibrdr	c100	0	EAB0

図 6-1 GPIB Spy のメインウィンドウ

## GPIB Spy の終了

---

GPIB Spy を終了するには、Spy メニューから Exit を選択します。

注意 :GPIB Spy を終了すると、記録された情報はすべて失われます。情報を保存する場合は、この章の「GPIB Spy の出力オプション」の節を参照してください。

## Call Details( 呼び出しの詳細 ) の表示

---

GPIB Spy に記録された呼び出しの詳細情報を見るには、Call Details ウィンドウを使用します。Call Details ウィンドウは 2 つの方法で起動することができます。1 つは、GPIB Spy のメインウィンドウに記録された呼び出しのどれかをダブルクリックする方法で、その呼び出しに関する詳しい情報が Call Details ウィンドウに表示されます。もう 1 つは、Spy メニューから View Details オプションを選択する方法です。この場合、Call Details ウィンドウがトップレベルウィンドウになります。

Call Details ウィンドウの使い方は 2 通りあります。View メニューで Continuously 項目をチェックすると、Call Details ウィンドウは、新しい GPIB が記録されるたびに更新され、ユーザは記録された各呼び出し間で詳細をスクロールすることはできません。View メニューの Continuously( 連続 ) 項目のチェックを取り消すと、<Page Up> または <Page Down> キーで、記録されたすべての呼び出しの詳細をスクロールすることができます。ドライバへの呼び出しによって GPIB エラーが発生すると、対応する Call Details ウィンドウは iberr Help ボタンを表示します。このボタンをクリックすると、対話型のヘルプユーティリティが起動され、これを使用して原因を突き止め、解決することができます。

図 6-2 に、NI-488 関数 `ibwrt` についての GPIB Spy の Call Details ウィンドウを示します。

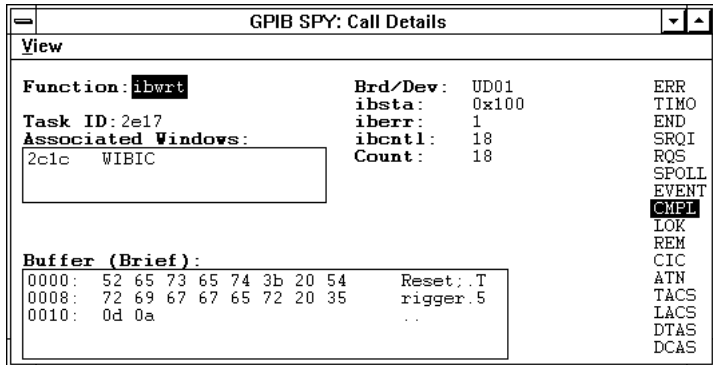


図 6-2 GPIB Spy の ibwrt の Call Details ウィンドウ

## GPIB Spy の構成オプション

### Calls to Highlight( 強調表示する呼び出し )

GPIB Spy では、ibsta に設定されたビットに基づいて、一定の呼び出しを強調表示することができます。このオプションを変更するには、Configure メニューから Calls to Highlight( 強調表示する呼び出し ) メニュー項目を選択します。ダイアログボックスを使用して、GPIB Spy のメインウィンドウでどの ibsta ビットがオンになった時に記録された呼び出しを強調表示するかを選択します。デフォルトでは、GPIB Spy は ibsta の ERR ビットがオンで返された NI-488 および NI-488.2 呼び出しを強調表示するよう構成されています。

### Calls to Trap On( トラップする呼び出し )

実行中の NI-488 または NI-488.2 呼び出しの ibsta の指定したトラップビットがオンになったらアプリケーションの実行を中断するよう、GPIB Spy を構成することができます。このオプションを変更するには、Configure メニューから Calls to Trap On( トラップする呼び出し ) メニュー項目を選択します。呼び出しによってトラップが発生すると、ダイアログボックスが現れ、トラップされた呼び出しに関する簡単な説明を表示します。このダイアログボックスが現れると、ユーザが OK ボタンをクリックするまで、実行中の Windows のすべてのアプリケーションが中断されます。デフォルトでは、GPIB Spy は呼び出しのトラップを全く行ないません。

### Buffer History( バッファ履歴 )

GPIB Spy が、ibrd, ibwrt, Send などのバッファデータを含む呼び出しを記録すると、Call Details ウィンドウはそのバッファの内容を表示し

ます。デフォルトでは、GPIB Spy は各入出力バッファを短いフォーマットで保存するよう構成されています。このオプションを変更するには、Configure メニューから Buffer History( バッファ履歴 )メニュー項目を選択します。バッファ履歴には、brief, full, none の 3 つの設定があります。これらの設定で、記録中の NI-488 および NI-488.2 呼び出しについて保存するバッファの大きさを決定します。一旦記録された呼び出しについては、バッファ履歴は変更できません。brief は、64 バイトまでのバッファ情報を保存します。バッファが 64 バイトより長い場合は、バッファの最初の 32 バイトと最後の 32 バイトが保存されます。none は、バッファ情報を保存しません。full は、入出力バッファ全体を記録します。full を選択すると、アプリケーションのパフォーマンス、および特にアプリケーションが大きな GPIB バッファを転送している場合は、ウィンドウズシステム全体のパフォーマンスが低下する可能性があるので注意してください。

## Set Highlight Color( 強調色の設定 )

このオプションを使用して、GPIB Spy ウィンドウで情報の強調表示に使用する強調色を構成することができます。強調色を変更するには、Configure メニューから Set Highlight Color( 強調色の設定 )メニュー項目を選択します。Windows の標準の「画面の色」ダイアログボックスが現れます。新しい色を選択して OK をクリックします。デフォルトの強調色は赤です。

## Store Configuration( 構成の保存 )

このオプションを使用して、GPIB Spy の現在の構成をデフォルトの構成にすることができます。GPIB Spy のデフォルト構成を設定するには、Configure メニューから Store Configuration( 構成の保存 )を選択します。

# GPIB Spy の出力オプション

---

## To Screen( 画面へ )

このオプションを使用して、GPIB Spy に表示する NI-488 および NI-488.2 呼び出しの数を選択することができます。デフォルトでは、GPIB Spy は 100 個の呼び出しを表示します。最新の情報だけが保存されるため、記録された呼び出しの数が制限を超えると、古い情報から捨てられます。表示する呼び出しの数を変更するには、Output メニューから To Screen( 画面へ )メニュー項目を選択します。そこで新しい値を入力して OK をクリックし、変更を保存します。

## To File( ファイルへ )

このオプションを使用して、現在 GPIB Spy に記録されている情報を保存するための出力ファイルを指定することができます。情報をファイルに保存するには、Output メニューから To File( ファイルへ ) メニュー項目を選択します。Windows の標準のファイル選択ダイアログボックスが現れます。情報を保存するドライブ、ディレクトリおよびファイル名を選択します。OK をクリックすると、GPIB Spy に記録された情報が指定ファイルに保存されます。

GPIB Spy 情報が記録されたらすぐに保存する場合は、ダイアログボックスの Update Continuously チェックボックスをチェックします。NI-488 または NI-488.2 呼び出しが記録されるたびに、出力ファイルが開かれ、情報を書き込んで閉じられます。GPIB Spy を閉じる前に、システムクラッシュを引き起こす Windows アプリケーションをデバッグしている時などに、このオプションを使用するとよいでしょう。Windows のすべてのアプリケーションのパフォーマンスが低下する可能性があるため、必要としないときはこのオプションは使用しないでください。さらに、ディスクスペースの空きが少ないときにこのオプションを選択すると、スペースが足りなくなるかもしれません。連続更新機能をオフにするには、ファイル選択ダイアログボックスの Update Continuously( 連続更新 ) チェックボックスの選択を解除してください。

## Clear Screen( 画面のクリア )

このオプションは、表示と GPIB Spy に保存された情報をクリアします。クリアされた情報は復元できません。

## GPIB Spy Help(GPIB Spy のヘルプ)

---

Help メニューから GPIB Spy Help(GPIB Spy のヘルプ) メニュー項目を選択して、GPIB Spy のオンラインヘルプにアクセスすることができます。この章で説明した情報はすべてオンラインヘルプで読むことができます。オンラインヘルプには、GPIB Spy の各機能の詳しい説明と Windows の GPIB アプリケーションを開発する際に役立つ情報がはいっています。

## パフォーマンスについて

---

GPIB Spy が GPIB アプリケーションのパフォーマンスおよびウィンドウズシステム全体のオペレーティング環境を低下させる可能性があることを心に留めておいてください。入出力バッファをすべて記録したり、出力ファイルを継続的に更新したり、多数の GPIB 呼び出しの情報を記録するように GPIB Spy を設定すると、パフォーマンスに重大な影響を及ぼすこ

とがあります。このため、アプリケーションのデバッグ中やパフォーマンスがそれほど重要でないときにのみ GPIB Spy を使用してください。

---

# 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。

- 7ビットまたは8ビットの 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いければ、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 - これがい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、`ibrdr`、`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、`ibrdr`、`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、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せんが、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。wibconf 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 にも当てはまりません。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いない場合には、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せん。
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。

## トーカー / 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、通常トーカー / 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。一般的なトーカー / 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、トーカー / 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

トーカー / 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(トーカーアクティブ状態)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

トーカー / 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。トーカー / 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 `S POLL` ビットをイネーブルしておきます。`S POLL` 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。

## 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 `wibconf` 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

スとシリアルポーリングを行います。シリアルポーリングは、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リング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、それ以上のポーリングは行われません。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リング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、自動シリアルポーリング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

## 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リングを行うことができます。`ibrsp` はシングルシリアルポーリングを実行し、シリアルポーリング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リング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リングを行うための 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リングが行えます。このルーチンは、ポーリング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リング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リングを行います。シリアルポーリングは、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リング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戻します。

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異なります。

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ラ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ぬときには、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isableするときには 1 に、イenable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`) を変更する必要があるときは、`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. `ibppc` 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);
```

---

# wibconf - Windows 用インタフェースバス構成ユーティリティ

この章では、Windows 用 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、wibconf について説明します。

## 概要

---

wibconf 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を使用する、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 wibconf を使用します。

wibconf 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、 wibconf 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。

wibconf 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、 wibconfig 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 wibconf 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。

## wibconf の起動

---

wibconf を使用するには、 Windows の Program Manager で GPIB グループの wibconf アイコンをダブルクリックするのが最も簡単です。 Parameters というラベルのついたダイアログボックスが表示されます。

デフォルトでは、 wibconf は c:\windows ディレクトリで NI-488.2 ソフトウェア構成ファイル gpib.ini を探します。 c:\windows 以外のディレクトリに Windows をインストールした場合、 Parameters のボック

スに gpib.ini ファイルへの完全なパスを指定してください。たとえば、ディレクトリ d:\win3 に Windows をインストールした場合は、Parameters ダイアログボックスに d:\win3\gpib.ini と入力します。Windows がデフォルトのディレクトリにインストールされている場合は、ただ <Return> キーを押してください。

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

表 8-1 wibconf 起動オプション

wibconf のオプション	動作
<i>driver</i>	特定のドライバを構成します。wibconf は、指定されたドライバ構成ファイルを構成します (例: wibconf d:\win3\gpib.ini)。
-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 <i>filename</i>	バッチモード構成。wibconf 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: wibconf -b gpib.cfg)。本章後半の「wibconf のバッチモード」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、wibconf を終了するときに、wibconf の警告メッセージがディスプレイされます。本章後半の「wibconf の終了」の節を参照してください。
-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る場合でも、wibconf 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、wibconf は画面への書き込みの際に、システムの BIOS ルーチンを使用します。このオプションを使用すると、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。

## wibconf の上位レベルと下位レベル

wibconf 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 wibconf を示します。

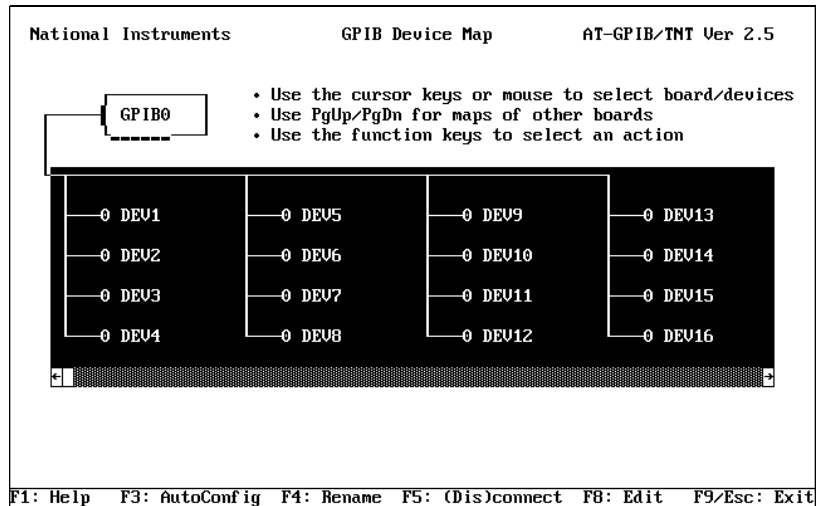


図 8-1 上位レベルの wibconf

図 8-1 に示すとおり、wibconf の上位レベル画面は、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)
- 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 )

wibconf 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 wibconf 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、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 wibconf 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。

## Exit(終了)

wibconf 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、wibconf 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「wibconf 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

wibconf 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構成について現在定義されている値が表示されます。

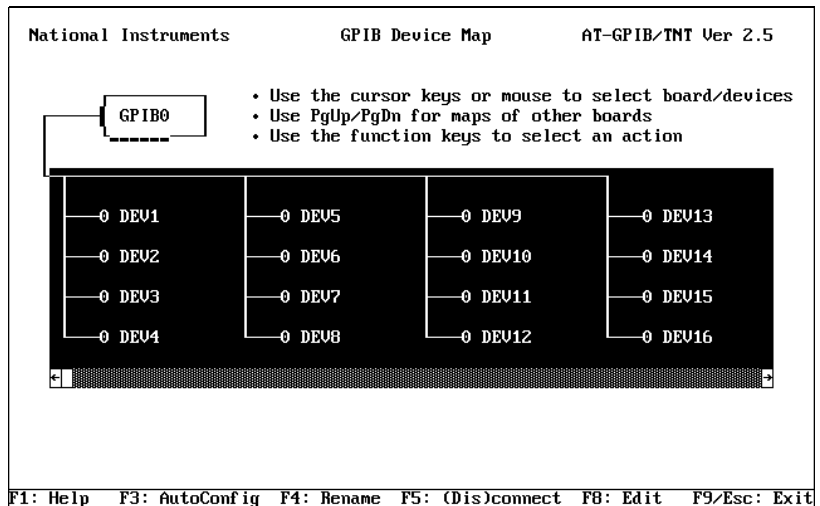


図 8-2 下位レベルの wibconf

下位レベルの画面には、wibconf 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、コンピュータ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)

wibconf 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 wibconf 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 wibconf 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、Windows 用 NI-488.2 ソフトウェアの wibconf 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。Windows 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 16 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セ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が、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、トーカーに最終バイトで 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 `ibsr` 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戻すと、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  $2\mu\text{s}$  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に、ポール間隔を  $10\mu\text{s}$  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、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、`wibconf` 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 チャンネル、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 チャンネルを指定します。GPIB ボード自身の DMA チャンネル (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 チャン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、`wibconf` 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 チャンネルの設定は変更できません。DMA チャン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、 wibconf 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。

## wibconf 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)。

## wibconf の終了

---

すべての変更を行ったら、<F9> または <Esc> を押して wibconf を終了します。プログラムは、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、wibconf を続ける場合は cancel を選択します。

## 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、wibconf 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。

wibconf 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。

## wibconf のバッチモード

---

wibconf 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。

```
wibconf -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 wibconf を実行すると、wibconf 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。wibconf 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 つでも見つかり、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 wibconf 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
<code>find</code>	ボードまたはデバイスを見つける	<code>board#</code> または <code>device#</code>
<code>pad</code>	1 次 GPIB アドレス	数値
<code>sad</code>	2 次 GPIB アドレス	数値
<code>tmo</code>	タイムアウト設定	ニモニック
<code>xeos</code>	書き込みで EOS とともに EOI を設定	<code>yes</code> または <code>no</code>
<code>bin</code>	EOS での比較のタイプ	7-bit または 8-bit
<code>eot</code>	書き込みの最終バイトに EOI を設定	<code>yes</code> または <code>no</code>
<code>sc</code>	システムコントローラ (ボードのみ)	<code>yes</code> または <code>no</code>
<code>sre</code>	SC 時の REN のアサート (ボードのみ)	<code>yes</code> または <code>no</code>

表 8-2 wibconf のバッチモードのコマンドペア

ニモニク	第 1 項目の説明	第 2 項目
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 チャンネル (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)	ニモニク
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	ニモニク

## 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

ニモニク	ビット位置	16進数値	タイプ	説明
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`, `ibrdr`, `ibwrtr`, `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。 シリアルポールに回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ナ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せん。

- 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 ボードがトーカー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。

## 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	Windows 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を使用されました
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
EDMA	8	DMA エラー
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)

---

EDVR は、以下の場合に返されます。

- `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」が入ります。
- 関数呼び出しに無効なユニット記述子が受け渡された場合。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.dll`) がインストールされていない場合。
- ドライバ構成ファイル `gpib.ini` がウィンドウズディレクトリにない場合。変数 `ibcntl` には値 -1 が入ります。
- ドライバファイル `gpib.ini` はウィンドウズディレクトリにあるが、使用中のドライバファイル `gpib.dll` と互換性がない場合。変数 `ibcntl` には -1 以外の負の値が入ります。

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 `wibconf` 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。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。
- `gpib.dll` と `gpib.ini` がウィンドウズディレクトリ (通常 `c:\windows`) にあるかどうかをチェックして、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てください。

## 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

- `ibrdr`, `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 `wibconf` 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。

## EDMA (8)

---

EDMA は、データ転送に DMA を使用しているときにエラーが発生した場合に返されます。ご使用のコンピュータに 16MB を超える RAM があり、ナショナルインスツルメンツの仮想 GPIB デバイス (`nivgpibd.386`) がインストールされていないときに、DMA を使用し、データバッファが



16MB を超えたメモリに作成されると NI-488.2 ソフトウェアは EDMA を返します。

Windows 3.0 でデータ転送に DMA を使用し、ナショナルインスツルメンツの仮想 DMA デバイス (`nivdmad.386`) がインストールされていないときに、データ転送に DMA を使用すると、NI-488.2 ソフトウェアは EDMA を返します。

#### 対処

- Windows ディレクトリ内の `system.ini` ファイルにある `(386Enh)` セクションに適切な仮想デバイスをインストールします。次の行で仮想 GPIB デバイスをインストールできます。

```
device = drive:\path\nivgpib.386
```

ここでは、`drive` と `path` でハードドライブ上の `nivgpib.386` の位置を表します。

- デフォルトでは、1 度に 1 つの GPIB ボードしか DMA を実行できません。同時に複数の GPIB ボードで DMA 転送を行う必要がある場合は、`system.ini` ファイルに新しいセクション (`vgpibd`) を追加してください。このセクションには、オプション `NumBoardsUsingDMA` を追加し、DMA を実行するボード数に等しい値を設定します。たとえば、同時に 2 つのボードで DMA を実行する場合は、`system.ini` ファイルの終わりに次の行を追加します。

```
[vgpibd]
```

```
NumBoardsUsingDMA=2
```

- 仮想 DMA デバイスをインストールするには、まずデフォルトの仮想 DMA の行を、セミコロンを追加してコメント行に変更します。それから、次のような行を追加して、ナショナルインスツルメンツの仮想 DMA デバイスをインストールします。

```
;device = *vdmad
```

```
device = drive:\path\nivdmad.386
```

ここでは、`drive` と `path` でハードドライブ上の `nivdmad.386` の位置を表します。

メモ: `system.ini` ファイルを変更した後は、必ず Windows を再起動してください。

- 上記以外の方法として、ソフトウェアで DMA をディスエーブルして EDMA のエラーを訂正することもできます。DMA のディスエーブルには `ibdma` を使用します。

## 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。

## EFSSO (12)

---

`ibrdf` または `ibwrdf` 呼び出しで、ファイル処理の実行に問題が発生すると、EFSSO 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、アドレスリ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 `wibconf` 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ー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。



---

# 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 時間いつでも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





# 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様のご意見を心よりお待ちしております。皆様の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：Windows 用 NI-488.2™ ユーザマニュアル

編集：1993 年 11 月

部品番号：370886A-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

---

接頭語	意味	数値
n-	ナノ	$10^{-9}$
$\mu$ -	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せん。
DLL	ダイナミックリンクライブラリ。
DMA( ダイレクトメモリアクセス )	GPIB ボードとメモリ間的高速データ転送で、直接には 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。
iberr	失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。
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せん。
ist	Parallel Poll Configure(パラレルポーリング構成)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。

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。

## W

wibconf Windows 用 NI-488.2 ドライバ構成プログラム

wibic Windows 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。

## 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リング。
シリアルポーリング	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。「パラレルポーリング」の項も参照してください。
ステータスバイト	シリアルポーリング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	「libsta」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ーはデータを、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。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。
トーカー	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。

**パラレルポーラ** 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。「シリアルポーラ」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。

ハンドシェイク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

!(直前の関数の繰り返し)関数、wibic 5-17  
\$(間接ファイルの実行)関数、wibic 5-19  
+(画面表示 ON)関数、wibic 5-17  
-(画面表示 OFF)関数、wibic 5-17  
1次 GPIB アドレス 「Primary GPIB Address(1次 GPIB アドレス)」の項を参照してください。  
2次 GPIB アドレス 「Secondary GPIB Address(2次 GPIB アドレス)」の項を参照してください。

## A

ALLSpoll ルーチン 7-9, 7-12  
ANSI/IEEE 規格 488.1-1987 「GPIB」の項を参照してください。  
ATN(アテンション)線 1-4  
ATN ステータスワード状況  
説明 A-5  
ビット位置、16進値、タイプ(表) 3-4

## B

Base I/O Address(ベース I/O アドレス)、設定 8-11  
Buffer History(バッファ履歴)、表示 6-3  
Bus Timing(バスタイミング)、設定 8-10

## C

Call Details(呼び出しの詳細)ウィンドウ、GPIB SPY ユーティリティ 6-2  
Calls 「呼び出し」の項を参照してください。  
CIC 「コントローラインチャージ(CIC)」の項を参照してください。  
CIC プロトコル  
GPIB ボードをコントローラインチャージにする 7-4  
イネーブル 8-10  
CIC ステータスワード状況  
説明 A-4  
ビット位置、16進値、タイプ(表) 3-4  
CMPL ステータスワード状況  
説明 A-4

ビット位置、16進値、タイプ(表) 3-4  
Configure(CFGn)メッセージ 7-3  
Configure Enable(CFE)メッセージ 7-3  
C 言語  
NI-488.2 ソフトウェアで利用可能なファイル 1-8  
アプリケーションのコンパイル、リンク、実行 3-19  
ダイレクトエントリ 3-20

## D

DAV(データ有効)線 1-3  
DCAS ステータスワード状況  
コントローラからのメッセージのモニタ 7-5  
説明 A-6  
ビット位置、16進値、タイプ(表) 3-4  
DevClear(デバイスクリア)ルーチン 3-16  
DMA Channel(DMA チャネル)、設定 8-11  
DTAS ステータスワード状況  
コントローラからのメッセージのモニタ 7-5  
説明 A-5  
ビット位置、16進値、タイプ(表) 3-4

## E

EABO エラーコード  
説明 B-5  
定義(表) 4-5  
EADR エラーコード  
説明 B-3  
定義(表) 4-5  
EARG エラーコード  
wibic の例 5-14  
説明 B-4  
定義(表) 4-5  
EBUS エラーコード  
説明 B-8  
定義(表) 4-5

- ECAP エラーコード  
 説明 B-7  
 定義(表) 4-5
- ECIC エラーコード  
 説明 B-2 ~ B-3  
 定義(表) 4-5
- EDMA エラーコード  
 説明 B-5 ~ B-6  
 定義(表) 4-5
- EDVR エラーコード  
 wibic の例 5-13  
 説明 B-2  
 定義(表) 4-5
- EFSO エラーコード  
 説明 B-7  
 定義(表) 4-5
- END ステータスワード状況  
 説明 A-2  
 ビット位置、16 進値、タイプ(表) 3-4
- ENEB エラーコード  
 wibic の例 5-13  
 説明 B-5  
 定義(表) 4-5
- ENOL エラーコード  
 説明 B-3  
 定義(表) 4-5
- EOI(終了または識別コードの要求)線  
 Send EOI at End of Write(書き込みの終わりに EOI を送信) 8-9  
 Set EOI with EOS on Writes(書き込みに EOS とともに EOI を設定) 8-8  
 データ転送の終了 7-1 ~ 7-2  
 目的(表) 1-4
- EOIP エラーコード  
 説明 B-7  
 定義(表) 4-5
- EOS  
 EOS バイト用の文字の設定 8-9  
 EOS モードの構成 7-1 ~ 7-2  
 Set EOI with EOS on Writes(書き込みに EOS とともに EOI を設定) 8-8  
 Terminate Read on EOS(EOS での読み取りの終了) 8-8  
 Type of Compare on EOS(EOS での比較タイプ) 8-8  
 文字列の終わり (EOS) モードのアプリ  
 ケーション例 2-7 ~ 2-8
- ERR ステータスワード状況  
 説明 A-2  
 ビット位置、16 進値、タイプ(表) 3-4
- ESAC エラーコード  
 説明 B-4  
 定義(表) 4-5
- ESRQ エラーコード  
 説明 B-9  
 定義(表) 4-5
- ESTB エラーコード  
 説明 B-8  
 定義(表) 4-5
- ETAB エラーコード  
 説明 B-9  
 定義(表) 4-5
- Event Status ビット (ESB) 7-6
- EVENT ステータスワード状況  
 説明 A-3  
 ビット位置、16 進値、タイプ(表) 3-4
- EVENT ビット、イネーブル 7-5
- ## F
- FindLstn(全リスナの検出)ルーチン 3-14  
 FindRSQ(サービス要求検出)ルーチン  
 7-10 ~ 7-11
- ## G
- GPIB  
 概要 1-1  
 構成 1-5 ~ 1-7 「wibconf ユーティリティ」の項も参照してください。  
 システムのリニア構成とスター構成(図) 1-5  
 複数ボードの制御 1-5  
 必要な構成 1-6 ~ 1-7  
 定義 1-1  
 トーカ、リスナ、コントローラ 1-1  
 メッセージの送信 1-2 ~ 1-3  
 インタフェース管理線 1-3  
 データ線 1-3  
 ハンドシェイク線 1-3

gpib.dll ファイル 1-7 「NI-488.2 DLL」の項も参照してください。

GPIBInfo ユーティリティ

実行 4-2 ~ 4-3

目的 1-7

gpib.ini ファイル 1-7, B-2

GPIB-PCII/IIA Mode Switch(GPIB-PCII/IIA モード切替え) 8-12

GPIB アドレス

1 次アドレス 1-2, 8-7

2 次アドレス 1-2, 8-7

wibic の構文 5-6

Enable Repeat Addressing(アドレス指定の繰り返し)のイネーブル) 8-12

アドレス指定の繰り返し 4-6

アドレスビット構成(図) 1-2

トークアドレス 1-2

目的 1-2

リスンアドレス 1-2

GPIB SPY ユーティリティ 「デバッグ」の項も参照してください。

Call Details(呼び出しの詳細)の表示 6-2

Help メニュー 6-5

概要 6-1

起動 6-1

構成オプション

Buffer History(バッファ履歴) 6-3

Calls To Highlight(強調表示する呼び出し) 6-3

Calls To Trap On(トラップする呼び出し) 6-3

Set Highlight Color(強調色の設定) 6-4

Store Configuration(構成の保存) 6-4

終了 6-2

出力オプション

Clear Screen(画面のクリア) 6-5

To File(ファイルへ) 6-5

To Screen(画面へ) 6-4

デバッグアプリケーション 4-4

メインウィンドウ(図) 6-1

目的 1-8

パフォーマンスについて 6-5

GPIB プログラミングテクニック

GPIB 条件の待機 7-3

高速データ転送 7-2 ~ 7-3

HS488 のイネーブル 7-2 ~ 7-3

システム構成の影響 7-3

シリアルポーリング 7-6 ~ 7-12

SRQ とシリアルポーリング

NI-488.2 ルーチンで 7-9 ~ 7-12

NI-488 デバイス関数で 7-9

サービス要求

IEEE 488.2 デバイスから 7-6

IEEE 488 デバイスから 7-6

自動シリアルポーリング 7-7 ~ 7-8

自動ポーリングと割り込み 7-8

スタック SRQ 状態 7-7

データ転送の終了 7-1 ~ 7-2

デバイスレベル呼び出しとバス管理 7-4

トーカー / リスナアプリケーション 7-5 ~ 7-6

イベント待ち行列 7-5

コントローラからのメッセージの待機 7-5

サービス要求 7-6

パラレルポーリング 7-12 ~ 7-14

NI-488.2 ルーチンで 7-13 ~ 7-14

NI-488 関数で 7-12 ~ 7-13

実行 7-12 ~ 7-14

## H

Help(ヘルプ情報の表示) 関数 5-17

Highlight Color(強調色)、設定 6-4

## I

ibask 関数 7-3

ibclr 関数

wibic での使用、例 5-2

デバイスのクリア 3-8

ibcmd 関数 7-3

ibcnt と ibcntI 変数 「カウント変数 - ibcnt と ibcntI」の項を参照してください。

ibconfig 関数

EOI 線のアサートの決定 7-2

EVENT ビットのイネーブル 7-5

GPIB ドライバの再構成 4-8  
 GPIB ボードを CIC として構成 7-4  
 高速データ転送のイネーブル 7-2 ~ 7-3  
 自動ポーリングのイネーブル 7-7 ~ 7-8  
**ibdev** 関数  
   wibic での使用 5-12 ~ 5-13  
     例 5-2  
   デバイスのオープン 3-8  
   パラレルポールの実行 7-12 ~ 7-14  
**ibdiag** ユーティリティ  
   ハードウェア構成のテスト 4-7  
   目的 1-7  
**ibeos** 関数 7-1  
**ibeof** 関数 7-1  
**iberr** 「エラー変数 - iberr」の項を参照してください。  
**ibevent** 関数 7-5  
**ibfind** 関数 5-12  
**iboni** 関数  
   wibic での使用、例 5-4  
   デバイスをオフラインにする 3-11  
   ボードをオフラインにする 3-19  
**ibppc** 関数  
   パラレルポーリングのためのデバイスの  
   構成解除 7-13  
   パラレルポールの実行 7-12  
**ibrd** 関数  
   wibic での使用 5-14  
     例 5-4  
**ibrpp** 関数 7-13  
**ibrsp** 関数  
   wibic での使用、例 5-4  
   シリアルポールの実行 7-7  
**ibsta** 「ステータスワード - ibsta」の項を参照してください。  
**ibtrg** 関数  
   wibic での使用、例 5-3  
   デバイスのトリガ 3-9  
**ibwait** 関数  
   GPIB 条件の待機 7-4  
   wibic での使用、例 5-3  
   シリアルポールの実行 7-7 ~ 7-8  
   スタック SRQ 状態の終了 7-7 ~ 7-8  
   測定の待機 3-10  
   トーカー / リスナアプリケーション 7-5

**ibwrt** 関数  
   wibic での使用 5-14  
     例 5-3  
   デバイスの構成 3-9  
 \*IDN? 問い合わせ 3-14 ~ 3-16  
 IEEE 規格 488.1-1987 「GPIB」の項を参照してください。  
 IFC(インタフェースクリア)線 1-4  
 Interrupt Jumper(割り込みジャンパ)、設定 8-12

## L

**LACS** ステータスワード状況  
   コントローラからのメッセージのモニタ  
   7-5  
   説明 A-5  
   ビット位置、16 進値、タイプ(表) 3-4  
**LOK** ステータスワード状況  
   説明 A-4  
   ビット位置、16 進値、タイプ(表) 3-4

## M

**Message Available(MAV)** ビット 7-7  
**Microsoft C** 「C 言語」の項を参照してください。

## N

**n\*(関数を n 回繰り返す)** 関数、wibic  
 5-18  
**NDAC(データ未受信)** 線 1-3  
**NI-488.2 DLL**  
   NI-488.2 ドライバのアンロードまたは再  
   ロード 1-10  
   アクセス方法の選択 3-1  
   直接アクセスの要件 3-1  
**NI-488.2 アプリケーション、プログラミング**  
   一般的な手順と例 3-14 ~ 3-19  
   組み込む項目 3-11  
   計測器の構成 3-16  
   計測器の識別 3-15  
   計測器の初期化 3-14  
   計測器のトリガ 3-17  
   初期化 3-14  
   全リスナの検出 3-14

- 測定の待機 3-17
  - 測定の読み取り 3-18
  - データの処理 3-19
  - プログラムシェル(図) 3-13
  - ボードをオフラインにする 3-19
  - ルーチンを使ったプログラミングのフローチャート 3-12
  - NI-488.2 ルーチン「NI-488.2 アプリケーション、プログラミング」の項も参照してください。
  - wibic 構文 5-9 ~ 5-10
  - wibic での使用
    - Receive 5-16
    - Send 5-15
    - SendList 5-15
    - 使用前に set コマンドを発行 5-15
  - シリアルポーリング 7-9 ~ 7-12
  - シリアルポーリングの例
    - AllSpoll の使用 7-11
    - FindRQS の使用 7-10
  - パラレルポーリング 7-12 ~ 7-14
  - プログラミング上の考慮点 3-3
  - NI-488.2 ソフトウェア 1-7 ~ 1-9 「アプリケーション開発」「NI-488 関数」「NI-488.2 ルーチン」の項も参照してください。
  - C 言語ファイル 1-8
  - Windows との対話 1-9
  - Visual Basic ファイル 1-8
  - アンロード 1-10
  - 再ロード 1-10
  - ドライバとドライバユーティリティ 1-7 ~ 1-8
  - NI-488.2 ダイナミックリンクライブラリへの直接アクセス
    - アプリケーションのコンパイル、リンク、実行
      - C 言語で 3-19
      - Visual Basic で 3-20
    - 必要条件 3-1
  - NI-488 アプリケーション、プログラミング
    - 一般的な手順と例 3-8 ~ 3-11
    - 組み込む項目 3-6
    - 測定の待機 3-10
    - 測定の読み取り 3-10
    - データの処理 3-11
    - デバイスのオープン 3-8
    - デバイスのクリア 3-8
    - デバイスの構成 3-9
    - デバイスのトリガ 3-9
    - デバイスレベル関数を使用したプログラミングのフローチャート 3-7
    - デバイスをオフラインにする 3-11
    - プログラムシェル(図) 3-7
  - NI-488 関数「補助関数、wibic」の項も参照してください。
    - 1 ボードに 1 デバイス 3-2 ~ 3-3
    - wibic での使用
      - ibdev 5-12 ~ 5-14
      - ibfind 5-12
      - ibrd 5-14
      - ibwrt 5-14
      - 構文 5-6 ~ 5-9
      - 例 5-1 ~ 5-5
    - シリアルポーリング 7-8
    - パラレルポーリング 7-12 ~ 7-13
    - プログラミング上の考慮点
      - 使用上の利点 3-2
      - デバイス関数 3-2
      - ボード関数 3-2 ~ 3-3
  - nivdmad.386 デバイス 1-7
  - nivgpibd.386 デバイス 1-7
  - NRFD(データ受信準備未完)線 1-3
- ## P
- PPollConfig ルーチン 7-14
  - PPollUnconfig ルーチン 7-14
  - PPoll ルーチン 7-14
  - Primary GPIB Address(1 次 GPIB アドレス)
    - 設定 8-7
    - 定義 1-2
    - 目的 8-7
  - print(ASCII 文字列の表示) 関数、wibic 5-19
- ## R
- readme.txt ファイル 1-7
  - ReadStatusByte ルーチン 7-9
  - Receive ルーチン
    - wibic での使用 5-16
    - 測定の読み取り 3-18

REM ステータスワード状況  
 説明 A-4  
 ビット位置、16 進値、タイプ(表) 3-4  
 REN(リモートイネーブル)線  
 自動アサートの設定 8-9  
 目的(表) 1-4  
 RQS ステータスワード状況  
 説明 A-3  
 ビット位置、16 進値、タイプ(表) 3-4

## S

Secondary GPIB Address(2 次 GPIB アドレス)  
 設定 8-7  
 定義 1-2  
 SendCmds 関数 7-3  
 SendIFC ルーチン 3-14  
 SendList ルーチン 5-15  
 Send ルーチン  
 wibic での使用 5-15  
 計測器の構成 3-16  
 set コマンド 5-15  
 Set(デバイスまたはボードの選択)関数  
 5-15  
 SPOLL ステータスワード状況  
 説明 A-3  
 ビット位置、16 進値、タイプ(表) 3-4  
 SRQ(サービス要求)線 「サービス要求」の  
 項も参照してください。  
 シリアルポーリング  
 NI-488.2 ルーチンで 7-9 ~ 7-12  
 NI-488 デバイス関数で 7-8  
 スタック SRQ 状態 7-7 ~ 7-8  
 目的(表) 1-4  
 SRQI ステータスワード状況  
 説明 A-3  
 ビット位置、16 進値、タイプ(表) 3-4  
 System Controller 「システムコントロー  
 ラ」の項を参照してください。

## T

TACS ステータスワード状況  
 コントローラからのメッセージのモニタ  
 7-5

説明 A-5  
 ビット位置、16 進値、タイプ(表) 3-4  
 Terminate Read on EOS(EOS での読み取り  
 の終了) 8-8  
 TestSRQ ルーチン 7-9  
 TIMO ステータスワード状況  
 説明 A-2  
 ビット位置、16 進値、タイプ(表) 3-4  
 TNT4882C ハードウェア 7-2  
 \*TRG コマンド 3-17

## V

Visual Basic  
 NI-488.2 ソフトウェアで利用可能なファ  
 イル 1-8  
 アプリケーションのコンパイル、リンク、  
 実行 3-20  
 ダイレクトエントリ 3-21

## W

WaitSRQ ルーチン  
 シリアルポールの実行 7-10  
 測定の待機 3-17 ~ 3-18  
 wait 関数 「ibwait 関数」の項を参照してく  
 ださい。  
 wibconf ユーティリティ  
 GPIB ドライバの再構成 4-6  
 概要 8-1  
 下位レベルのデバイス / ボードの特性  
 8-5 ~ 8-6  
 Help(ヘルプ) オプション(<F1>)  
 8-6  
 Reset Value(値のリセット)(<F6>)  
 8-6  
 Return to Map(マップに戻  
 る)(<F9> または <Escape>)  
 8-6  
 画面の図 8-5  
 特性の変更(<PageUp>,  
 <PageDown>, <Home>, または  
 <End>) 8-6  
 ボードまたはデバイスの変更  
 (<Control-PageUp> および  
 <Control-PageDown>) 8-6  
 起動 8-1 ~ 8-2

- 起動オプション(表) 8-2
- 構成オプション 8-7 ~ 8-12
  - Assert REN when SC(SC 時の REN のアサート) 8-9
  - Base I/O Address(ベース I/O アドレス) 8-11
  - Bus Timing(バスタイミング) 8-10
  - Cable Length for High Speed(高速用ケーブル長) 8-10
  - DMA Channel(DMA チャンネル) 8-11
  - Enable Auto Serial Polling(自動シリアルポーリングのイネーブル) 8-9
  - Enable CIC Protocol(CIC プロトコルのイネーブル) 8-10
  - Enable Repeat Addressing(アドレス指定の繰り返しのイネーブル) 8-12
  - EOS Byte(EOS バイト) 8-9
  - GPIB-PCII/IIA Mode
    - Switch(GPIB-PCII/IIA モード切替え) 8-12
  - Interrupt Jumper Setting(割り込みジャンパ設定) 8-12
  - Parallel Poll Duration(パラレルポールの間隔) 8-11
  - Primary GPIB Address(1 次 GPIB アドレス) 8-7
  - Secondary GPIB Address(2 次 GPIB アドレス) 8-7
  - Send EOI at End of Write(書き込みの終わりに EOI を送信) 8-9
  - Serial Poll Timeout(シリアルポールタイムアウト) 8-8, 8-12
  - Set EOI with EOS on Writes(書き込みに EOS とともに EOI を設定) 8-8
  - System Controller(システムコントローラ) 8-9
  - Terminate Read on EOS(EOS での読み取りの終了) 8-8
  - Timeout Setting(タイムアウトの設定) 8-8
  - Type of Compare on EOS(EOS での比較タイプ) 8-8
  - Use This GPIB Interface(この GPIB インタフェースを使用) 8-11
- 終了(<F9> または <Escape>) 8-13
  - エラーのチェック 8-13
  - 上位レベルデバスマップ 8-3 ~ 8-5
  - 上位レベルデバスマップオプション
    - (Dis)connect(接続 / 非接続)<F5> 8-4
    - Edit(編集)<F8> 8-4
    - Exit(終了)<F9> または <Escape> 8-5
    - Help<F1> 8-4
    - Rename(名前の変更)<F4> 8-4
    - 画面の図 8-3
    - ボードのデバスマップ 8-3
  - デフォルトの構成 8-13
  - バッチモード 8-14 ~ 8-16
    - 構成ファイル 8-14 ~ 8-15
    - コマンドペア(表) 8-15
  - 目的 1-8
- wibic ユーティリティ
  - NI-488.2 ルーチン
    - Receive 5-16
    - Send 5-15
    - SendList 5-15
    - 使用前に set コマンドを発行 5-15
  - NI-488 関数
    - ibdev 5-12
    - ibfind 5-12
    - ibrd 5-14
    - ibwrt 5-14
    - 例 5-1 ~ 5-5
  - アプリケーションのデバッグ 4-4
  - エラー情報 5-11
  - 概要 5-1
  - カウント 5-11
  - 構文 5-5 ~ 5-10
    - NI-488.2 ルーチン 5-9 ~ 5-10
    - NI-488 関数 5-5 ~ 5-9
    - アドレス 5-6
    - 数値 5-5
    - デバイスレベル関数(表) 5-6
    - ボードレベル関数(表) 5-7 ~ 5-9
    - 文字列 5-5
  - ステータスワード 5-10
  - デバイスとの通信 3-6
  - 表示エラーのチェック 4-4
  - プログラミング上の考慮点 3-6



- 補助関数
    - !(直前の関数の繰り返し) 5-17
    - \$(間接ファイルの実行) 5-19
    - +(画面表示 ON) 5-17
    - (画面表示 OFF) 5-17
    - Help(ヘルプ情報の表示) 5-17
    - n\*(関数を n 回繰り返し) 5-18
    - print(ASCII 文字列の表示) 5-19
    - Set(デバイスまたはボードの選択) 5-14
    - 関数表 5-16
    - 目的 1-8
  - wibtest ユーティリティ
    - 診断メッセージ 4-1 ~ 4-2
      - ケーブル接続 4-2
      - ドライバの存在確認 4-1
      - ボードの存在確認 4-1 ~ 4-2
    - 目的 1-8
  - Windows と NI-488.2 ソフトウェアとの関係 1-9
  - Windows 用インタフェースバス対話式制御ユーティリティ 「wibic ユーティリティ」の項を参照してください。
- あ**
- アクティブコントローラ 「コントローラインチャージ (CIC)」の項を参照してください。
  - アドレス 「GPIB アドレス」の項を参照してください。
  - アドレス指定の繰り返し
    - イネーブル 8-12
    - 通信エラー 4-6
  - アプリケーション開発 「デバッグ」の項も参照してください。
  - NI-488.2 DLL へのアクセス 3-1
  - アプリケーション例
    - IEEE 488.2 準拠デバイスとの簡単な通信 2-14 ~ 2-15
    - NI-488.2 ルーチンを使ったシリアルポール 2-16 ~ 2-17
    - 基本的な通信 2-2 ~ 2-3
    - コントローラでないデバイスのエミュレーション 2-21 ~ 2-22
    - サービス要求 2-9 ~ 2-13
    - ソースコードファイル 2-1 ~ 2-2
    - デバイスのクリアとトリガ 2-3 ~ 2-5
    - パラレルポール 2-18 ~ 2-20
    - 非同期入出力 2-5 ~ 2-7
    - 文字列の終わり (EOS) モード 2-7 ~ 2-9
  - NI-488.2 アプリケーション
    - 一般的な手順と例 3-14 ~ 3-19
    - 組み込む項目 3-11
    - 計測器の構成 3-16
    - 計測器の識別 3-15
    - 計測器の初期化 3-16
    - 計測器のトリガ 3-17
    - 初期化 3-14
    - 全リスナの検出 3-14
    - 測定の待機 3-17
    - 測定の読み取り 3-18
    - データの処理 3-19
    - プログラムシェル (図) 3-13
    - ボードをオフラインにする 3-19
    - ルーチンを使ったプログラミングのフローチャート 3-12
  - NI-488.2 ルーチン 3-3
  - NI-488 アプリケーション
    - 一般的な手順と例 3-8 ~ 3-11
    - 組み込む項目 3-6
    - 測定の待機 3-10
    - 測定の読み取り 3-10
    - データの処理 3-11
    - デバイスのオープン 3-8
    - デバイスのクリア 3-8
    - デバイスの構成 3-9
    - デバイスのトリガ 3-9
    - デバイスレベル関数を使ったプログラミングのフローチャート 3-7
    - デバイスをオフラインにする 3-11
    - プログラムシェル (図) 3-7
  - NI-488 関数 3-2 ~ 3-3
    - 1 ボードに 1 デバイス 3-2 ~ 3-3
    - デバイス関数 3-2
    - ボード関数 3-2 ~ 3-3
    - 利点 3-2
  - NI-488 関数または NI-488.2 ルーチンの選択 3-1 ~ 3-3
  - アプリケーションのコンパイル、リンク、実行

C 言語によるダイレクトエントリ  
 3-20 ~ 3-21  
 Visual Basic によるダイレクトエ  
 ントリ 3-21  
 Microsoft C 3-19  
 Visual Basic アプリケーション  
 3-20  
 ステータス確認のためのグローバル変数  
 3-3 ~ 3-6  
 エラー変数 - iberr 3-5  
 カウント変数 - ibcnt と ibcntl 3-5  
 ステータスワード - ibsta 3-3 ~ 3-5  
 デバイスと通信を行うための wibic 3-6  
 アプリケーションのコンパイル、リンク、実  
 行

C 言語によるダイレクトエントリ 3-20  
 ~ 3-21  
 Visual Basic によるダイレクトエントリ  
 3-21  
 Microsoft C 3-19  
 Visual Basic 3-20  
 アプリケーションの実行「アプリケーション  
 のコンパイル、リンク、実行」の項を参照し  
 てください。  
 アプリケーションのリンク「アプリケーシ  
 ョンのコンパイル、リンク、実行」の項を参照  
 してください。  
 イベント待ち行列 7-5  
 インタフェース管理線 1-3  
 インタフェースバス対話式制御ユーティリ  
 ティ「wibic ユーティリティ」の項を参照し  
 てください。  
 エラーコードと対処

EABO B-5  
 EADR B-3  
 EARG B-4  
 EBUS B-8  
 ECAP B-7  
 ECIC B-2 ~ B-3  
 EDMA B-5 ~ B-6  
 EDVR B-2  
 EFSO B-7  
 ENEB B-5  
 ENOL B-3  
 EOIP B-7  
 ESAC B-4

ESRQ B-9  
 ESTB B-8  
 ETAB B-10  
 コード表 4-5  
 エラー状況  
 wibconf 終了前のエラーのチェック  
 8-13  
 wibic エラー情報 5-11  
 構成エラー 4-5  
 タイミングエラー 4-6  
 通信エラー 4-6 ~ 4-7  
 アドレス指定の繰り返し 4-6  
 終了方法 4-7  
 エラー変数 - iberr 3-5

## か

カウント、wibic の 5-11  
 カウント変数 - ibcnt と ibcntl 3-5  
 カスタマーコミュニケーション xvii, C-1  
 画面表示 OFF(-) 関数、wibic 5-17  
 画面表示 ON(+) 関数、wibic 5-17  
 関数「補助関数、wibic」「NI-488 関数」の  
 項を参照してください。  
 関数を n 回繰り返す (n\*) 関数、wibic 5-18  
 間接ファイルの実行 (\$) 関数、wibic 5-19  
 強調色 (Highlight Color)、設定 6-4  
 グローバル変数 3-3 ~ 3-6  
 アプリケーションのデバッグ 4-3 ~ 4-4  
 エラー変数 - iberr 3-5  
 カウント変数 - ibcnt と ibcntl 3-5  
 ステータスワード - ibsta 3-3 ~ 3-5, A-1  
 ~ A-6  
 ケーブル  
 wibtest によるチェック 4-2  
 高速データ転送用のケーブル長の設定  
 8-10  
 構成 1-4 ~ 1-6 「 GPIB SPY ユーティリ  
 ティ」「wibconf ユーティリティ」の項も参  
 照してください。  
 システムのリニア構成とスター構成  
 (図) 1-4  
 必要な構成 1-6 ~ 1-7  
 複数ボードの制御 1-5  
 構成エラー 4-5 ~ 4-6

構成ファイル、wibconf バッチモード用  
8-14 ~ 8-16  
高速データ転送 (HS488) 7-2 ~ 7-3  
HS488 のイネーブル 7-2 ~ 7-3  
ケーブル長の設定 8-10  
システム構成の影響 7-3  
コントローラ  
アイドルコントローラ 1-1  
システムコントローラ (System  
Controller) 1-1, 8-9  
定義 1-1  
コントローラでない GPIB のエミュレー  
ション、例 2-21 ~ 2-22  
コントローラインチャージ (CIC)  
CIC としてのアクティブコントローラ  
1-1  
CIC プロトコル 7-4, 8-10  
GPIB ボードを CIC として構成する  
7-4, 8-10  
~としてのシステムコントローラ 1-1

## さ

サービスの要求 「サービス要求」の項を参照  
してください。  
サービス要求 「SRQ(サービス要求)線」の  
項も参照してください。  
アプリケーション例 2-9 ~ 2-13  
シリアルポーリング  
IEEE 488.2 デバイス 7-6  
IEEE 488 デバイス 7-6  
スタック SRQ 状態 7-7  
トーカー / リスナアプリケーション 7-5  
システムコントローラ (System Controller)  
構成 8-9  
コントローラインチャージとして 1-1  
システムのセットアップ 「構成」の項を参照  
してください。  
自動シリアルポーリング 「シリアルポーリン  
グ」の項を参照してください。  
終了方法、~によるエラー 4-7  
出力オプション、GPIB SPY ユーティリティ  
6-4 ~ 6-5  
シリアルポーリング 7-6 ~ 7-12  
NI-488.2 ルーチンを使ったアプリケー  
ション例 2-16 ~ 2-17

SRQ とシリアルポーリング  
NI-488.2 ルーチンで 7-9 ~ 7-12  
NI-488 デバイス関数で 7-8  
サービス要求  
IEEE 488.2 デバイスから 7-6  
IEEE 488 デバイスから 7-6  
自動シリアルポーリング 7-7 ~ 7-8  
イネーブル 8-10  
自動ポーリングと割り込み 7-8  
スタック SRQ 状態 7-7  
タイムアウト値の設定 8-8, 8-12  
数値の構文、wibic で 5-5  
スタック SRQ 状態 7-7  
ステータスワード - ibsta 3-3 ~ 3-5  
ATN A-5  
CIC A-4  
CMPL A-4  
DCAS 7-5, A-6  
DTAS 7-5, A-5  
END A-2  
ERR A-2  
EVENT A-3  
LACS 7-5, A-4  
LOK A-4  
REM A-4  
RQS A-3  
SPOLL A-3  
SRQI A-3  
TACS 7-5, A-5  
TIMO A-2  
wibic の例 5-9  
プログラミング上の考慮点 3-3 ~ 3-6  
目的および使用 3-3 ~ 3-6  
レイアウト (表) 3-4  
セットアップユーティリティ 1-5  
ソフトウェア 「NI-488.2 ソフトウェア」の  
項を参照してください。

## た

ダイナミックリンクライブラリ、GPIB  
「NI-488.2 DLL」の項を参照してください。  
タイミングエラー 4-6  
タイムアウト値、設定 8-8  
シリアルポールのタイムアウト 8-8,  
8-12

直前の関数の繰り返し (!) 関数、wibic 5-17  
 通信アプリケーション例  
   基本的な通信 2-2 ~ 2-3  
     IEEE 488.2 準拠デバイスで 2-14 ~ 2-15  
 通信エラー 4-6  
   アドレス指定の繰り返し 4-6  
   終了方法 4-7  
 データ転送  
   高速 (HS488) 7-2 ~ 7-3  
     イネーブル 7-2 ~ 7-3  
     システム構成の影響 7-3  
   終了 7-1 ~ 7-2  
 データ転送の終了 7-1 ~ 7-2  
 データ線 1-3  
 テクニカルサポート C-1  
 デバイスのトリガ、例 2-3 ~ 2-5  
 デバイス構成 「wibconf ユーティリティ」の項を参照してください。  
 デバイス関数 「NI-488 関数」の項を参照してください。  
 デバイスレベル呼び出しとバス管理 7-4  
 デバイスのクリアおよびトリガ、例 2-3 ~ 2-5  
 デバッグ 「GPIB SPY ユーティリティ」の項も参照してください。  
   GPIBInfo ユーティリティ 4-2 ~ 4-3  
   GPIB エラーコード (表) 4-4 ~ 4-5  
   GPIB SPY ユーティリティ 4-4  
   wibic ユーティリティ 4-4  
   wibtest 診断 4-1 ~ 4-2  
     ケーブル接続 4-2  
     ドライバの存在確認 4-1  
     ボードの存在確認 4-1 ~ 4-2  
 Q&A 4-7 ~ 4-9  
 構成エラー 4-5 ~ 4-6  
 グローバルステータス変数 4-3 ~ 4-4  
 タイミングエラー 4-6  
 通信エラー 4-6  
   アドレス指定の繰り返し 4-6  
   終了方法 4-7  
 デフォルト構成、wibconf 8-13  
 トーカ 1-1  
 トーカ / リスナアプリケーション 7-5 ~ 7-6  
   イベント待ち行列 7-5  
   コントローラからのメッセージの待機 7-6

サービスの要求 7-5  
 トークアドレス、設定 1-2  
 ドライバ  
   NI-488.2 ソフトウェア用ドライバとドライバユーティリティ 1-7 ~ 1-8  
   wibtest によるテスト 4-1  
   再構成 4-5 ~ 4-6  
 トラブルシュート 「デバッグ」「wibic ユーティリティ」の項を参照してください。

## は

バスタイミング 「Bus Timing (バスタイミング)」の項を参照してください。  
 バッチモード、wibconf ユーティリティ 8-14 ~ 8-16  
   構成ファイル 8-14 ~ 8-15  
   コマンドペア (表) 8-15  
 バッファ履歴 (Buffer History)、表示 6-3  
 パラレルポーリング 7-12 ~ 7-14  
   NI-488.2 ルーチンで 7-13 ~ 7-14  
   NI-488 関数で 7-12 ~ 7-13  
   アプリケーション例 2-18 ~ 2-20  
   間隔の設定 8-11  
   実行 7-12 ~ 7-14  
 ハンドシェイク線 1-3  
 汎用インタフェースバス 「GPIB」の項を参照してください。  
 非同期入出力アプリケーション例 2-5 ~ 2-7  
 ファックスによるユーザサポート C-1  
 プログラミング 「アプリケーション開発」「デバッグ」「GPIB プログラミングテクニック」の項を参照してください。  
 文書  
   関連文書 xvi  
   このマニュアルで使う表記法 xv  
   マニュアルセットの使い方 xiii  
   マニュアルの構成 xiv  
 ベース I/O アドレス (Base I/O Address)、設定 8-11  
 ボード  
   GPIB ボードへのアクセスをディスエーブルする 8-11  
   wibtest によるテスト 4-1 ~ 4-2  
 ボード関数 「NI-488 関数」の項を参照してください。

ボードの構成 「wibconf ユーティリティ」の項を参照してください。

補助関数、wibic

- !(直前の関数の繰り返し) 5-17
- \$(間接ファイルの実行) 5-19
- +(画面表示 ON) 5-17
- (画面表示 OFF) 5-17
- Help(ヘルプ情報の表示) 5-17
- n\*(関数を n 回繰り返す) 5-18
- print(ASCII 文字列の表示) 5-19
- Set(デバイスまたはボードの選択)  
5-17
- 関数表 5-16

## ま

マニュアル 「文書」の項を参照してください。

- メッセージ、 GPIB での送信 1-2 ~ 1-4
- インタフェース管理線 1-3
- データ線 1-3
- ハンドシェイク線 1-3
- 文字列の終わり文字 「EOS」の項を参照してください。

文字列の構文、wibic で 5-5  
問題解決 「デバッグ」の項を参照してください。

## や

- 呼び出し
- 強調表示する 6-3
- トラップする 6-3

## ら

- リスナ 1-1 「トーカー / リスナアプリケーション」の項も参照してください。
- リスナアドレス、設定 1-2
- ルーチン 「NI-488.2 ルーチン」の項を参照してください。

## わ

- 割り込みジャンパ (Interrupt Jumper)、設定 8-12
- 割り込みと自動ポーリング 7-8