Hivertec Bridge Board Series

HPCIe-BRGGPI4INT

4チャンネル インテリジェントGPIBインターフェースボード

ソフトウェアマニュアル

2022-10-13 版　　株式会社ハイバーテック 発行

株式会社 ハイバーテック
東京都江東区新大橋 1-8-11
大樹生命新大橋ビル
TEL 03-3846-3801
FAX 03-3846-3773
sales@hivertec.co.jp

 本製品をご使用される前に「注意事項」を必ずご一読の上ご利用をお願い致します。

はじめに

保証範囲

本製品の保証期間は，お買い上げ頂いた日より 3 年間です．保証期間中に弊社の判断により欠陥が判明した場合には，本製品を弊社に引き取り，修理または交換を行います．保証期間内外に関わらず，弊社製品の使用，供給 (納期) または故障に起因する，お客様及び第三者が被った，直接，間接，二次的な損害あるいは，遺失利益の損害に付いて，弊社は本製品の販売価格以上の責任を負わないものとしますので，予めご了承ください．

免責事項

本書に記載された内容に沿わない，製品の取付，接続，設定，運用により生じた損害に対しましては，一切の責任を負いかねますので，予めご了承ください．本製品は，一般電子機器用 (工作機械・計測機器・FA/OA 機器・通信機器等) に製造された半導体製品を使用していますので，その誤作動や故障が直接，生命を脅かしたり，身体・財産等に危害を及ぼしたりする恐れのある装置 (医療機器・交通機器・燃焼機器・安全装置等) に適用できるような設計，意図，または，承認，保証もされていません．ゆえに本製品の安全性，品質および性能に関しては，本書 (またはカタログ) に記載してあること以外は明示的にも黙示的にも一切保証するものではありませんので，予めご了承ください．保証期間内外に関わらず，お客様が行った弊社の承認しない製品の改造または，修理が原因で生じた損害に対しましては，一切の責任を負いかねますので，予めご了承ください．本書に記載された内容について，弊社もしくは，第三者の特許権，著作権，商標権，その他の知的所有権の権利に対する保証または実施権の許諾を行うものではありません．また本書に記載された情報を使用したことにより第三者の知的所有権等の権利に関わる問題が生じた場合，弊社は，その責任を負いかねますので，予めご了承ください．

安全にお使いいただくために

 警告

本製品のご使用前に、必ず本マニュアル及び付属書類を全て熟読し、内容を理解してから正しくご使用下さい。

- 本製品の知識、安全の情報及び注意事項の全てに付いて習熟してからご使用下さい。
- 本製品には安全対策、対応規格の仕様以上の通信エラー回避等は含まれておりませんので、お客様の方でご配慮願います。

対象ユーザー

 警告

本製品およびマニュアルは、以下の様な、ユーザーを対象としています。* 制御用電子機器、製品が対応する通信規格についての基本的な知識、パソコン、Windows 及び開発環境等について基本的な知識を有している方。

添付ソフトウェア適合 OS


警告

デバイスドライバ，ドライバ関数等の添付ソフトウェアは，Windows10(64bit) においてボードの制御を行う為のソフトウェアです．上記以外の OS でのご使用については，弊社営業までお問合せ下さい．

試運転・調整


警告

本シリーズ製品を使用し装置を動作させる時は，プログラムのデバッグを充分行ってから動作させてください．プログラムに間違いがありますと，思わぬ動きをすることがあります．

動かしてみるプログラム


警告

本製品に添付される「動かしてみる」プログラムは，ボードが正しく設定・装着されているか，動作環境が正しく設定されているかを確認するとともに，ボードの機能・動作を理解して頂く為のものです．故に使用される機器毎に固有な安全対策処理等を含んでいませんので，「動かしてみる」プログラムを定常的に機器運転に使用しないで下さい．

サンプルプログラム


警告

本製品に添付されるサンプルプログラムは，ボードを制御する手順・制御プログラムの作成方法を理解して頂く為のものです．故に使用される機器毎に固有な安全対策処理等を含んでいませんので，サンプルプログラムを定常的に機器運転に使用しないで下さい．

ユーザープログラム


警告

本製品を使用し装置を動作させる際には，プログラムのデバッグを充分行ってから動作させて下さい．プログラムに間違いがありますと，思わぬ動きをすることがあります．本製品に添付されるサンプルプログラムまたはマニュアル内のコード例は，本製品のソフトウェア・ボードの機能・動作を理解して頂く為のものです．故に使用される機器毎に固有な安全対策処理・エラー処理・例外処理・排他処理等は省略されています．実際にプログラムを作成する場合は，十分に上記対策等を考慮し，必要な処理を追加してください．

目次

第 1 章

製品の概要

HPCIe-BRGGPI4INT は、１枚のボードで４ｃｈの GPIB コントローラーを搭載したインターフェースボードです。

1.1 基本仕様

- チャンネル数： 4 (制御チップ NAT9914 NATIONAL INSTRUMENTS 製)
- 送受信バッファサイズ ： 各チャンネル 送信：8000 byte 受信：8000 byte
- デリミタ ： CR(0x0d),LF(0x0a),CR+LF(0x0d,0x0a),EOI のみ

1.2 GPIB 機能

- T(Talker)
  - 基本トーカ機能
  - シリアルポーリング機能
  - EOI 送信、デリミタ制御
- L(Listener)
  - 基本リスナ機能
  - EOI 検出、デリミタ制御
- SR(Service Request)
  - SRQ 送信機能
- C(Controller)
  - GPIB システムコントローラー機能
  - IFC 信号送信
  - REN 信号送信
  - SRQ 応答
  - マルチラインインターフェースメッセージ送信

1.3 添付ソフトウェア

- デバイスドライバー：Windows10 64bit 用。ドライバー関数 DLL。
- HPCIe-BRGGPI4INT 用動かしてみる ： Windows10 64bit 用。
- サンプルプログラム：Windows10 64bit 用。VisualStudio2019 使用。

第 2 章

ソフトウェアのセットアップ

2.1　OS の設定

本ボードを使用するにあたり、OS の高速スタートアップを OFF にする必要があります。
1) コントロールパネルから、” 電源オプション ” を選択します。
2) ” 電源ボタンの動作を選択する ” をクリックします。
3) ” 現在利用可能ではない設定を変更します ” をクリックします。
3) ” 高速スタートアップを有効にする（推奨）” のチェックボックスのチェックを外します。
4) ” 変更の保存 ” ボタンを押します。
5) PC を再起動します。

2.2　デバイスドライバーのインストール

本ボードを WindowsOS で使用するために、デバイスドライバーをインストールします。
デバイスドライバーのインストール方法は 2 種類あります。どちらの方法でもインストールできます。

2.2.1　インストーラーを使用したインストール

1) drivers フォルダにある setup.exe を右クリックします。
2) ポップアップメニューで、” 管理者として実行 ” を選択します。
3) 画面に表示されるダイアログに従ってデバイスドライバーをインストールします。
4) PC を再起動します。

2.2.2　inf ファイルを使用したインストール

1) driver_x64 フォルダにある inf ファイルを右クリックします。
2) ポップアップメニューで ” インストール ” を選択します。
3) 正常にインストールされましたという旨のメッセージが出ればインストール完了です。
4) PC を再起動します。

2.3　添付ソフトウェアのインストール

実行ファイルが圧縮されていますので、任意の場所で展開してください。

2.4　デバイスドライバーのアンインストール

2.4.1　インストーラーでインストールした場合

1) drivers フォルダにある setup.exe を右クリックします。
2) ポップアップメニューで、” 管理者として実行 ” を選択します。
3) "修復または削除"の選択画面で、削除の方にチェックを入れます。

4) 画面に表示されるダイアログに従ってデバイスドライバーをアンインストールします。
5) PC を再起動します。
または、Windows の"アプリと機能一覧"から、HPCIe-BRGGPI4INT を探して、アンインストールします。

2.4.2 inf ファイルを使用した方法でインストールした場合

1) コントロールパネルから、デバイスマネージャーを選択します。
2) デバイスマネージャーのデバイスツリーから ” Hivertec Board ” をクリックします。
3) HPCIe-BRGGPI4INT を右クリックしてポップアップメニューの ” 削除 ” を選択します。
4) ” デバイスのアンインストールの確認 ” ダイアログが表示されるので、” このデバイスのドライバーソフトウェアを削除する ” にチェックを入れて OK ボタンを押します。
5) PC を再起動します。

第 3 章

基本情報

3.1　この文書は

HPCIe-BRGGPI4INT の API を使用したプログラミングについて、基本情報を説明するドキュメントです。

3.2　HPCIe-BRGGPI4INT の制御プログラミング概要

HPCIe-BRGGPI4INT では、メインコントローラー、チャンネルコントローラーという 2 つのコントローラーが実装されています。メインコントローラーは、各チャンネルコントローラーに指令を伝えます。チャンネルコントローラーは 4 つあり、それぞれ物理的な GPIB チャンネルを管理します。

HPCIe-BRGGPI4INT を制御するためのプログラミングの基本的な流れは以下のようになります。

基本的な流れ

1. インストールされているボード数を取得
2. デバイス ID を指定して操作したいデバイスをオープン
3. ボードリセットを行う
4. 送信バッファへコマンドやデータを書き込む
5. 送信設定完了のステータスを設定する
6. 実行ステータスを確認する
7. 受信完了となったら受信バッファからデータを読み出す
8. デバイスをクローズ

以下の説明は、トラブルが発生した時に送信バッファ、受信バッファの内容を確認する事で問題の解決に役立つ情報として記載します。

3.3　ステータス

ステータスは 2 種類のステータスがあります。１つは、メインコントローラーのステータス、もう１つはチャンネルコントローラーのステータスです。ステータスは、実行状況やエラー状態などを表します。チャンネルコントローラーステータスは、送信ステータスと受信ステータスの２つがあります。

▼リスト 3.1　**メインコントローラーのステータス定義**

```
enum MC_ChCmdStatus
        MC_CH_STAT_NONE = 0,            // なし
        MC_CH_STAT_ACCEPTED,            // 受け付けた
        MC_CH_STAT_CANCELLED,           // キャンセルされた
        MC_CH_STAT_ILLEGALCMD,          // 不正なコマンド
        MC_CH_STAT_SET_SENDING,         // 送信設定中
        MC_CH_STAT_SENDING,             // 送信中
        MC_CH_STAT_SEND_DONE,           // 送信完了
        MC_CH_STAT_SET_RECEIVING,       // 受信設定中
        MC_CH_STAT_SET_RECEIVE_DONE,    // 受信設定完了
        MC_CH_STAT_RECEIVING,           // 受信中
        MC_CH_STAT_RECEIVE_DONE,        // 受信完了
```

```
;
```

▼リスト 3.2　チャンネルコントローラーステータス定義：送信

```
#define FLAG_CLEAR                      0
#define FLAG_TX_SET_START               0x01    // 設定開始
#define FLAG_TX_SET_DONE                0x02    // 設定完了
#define FLAG_TX_REQ_SET_DONE_CLR        0x04    // 完了クリア要求
#define FLAG_TX_REQ_ABORT               0x08    // 送信中断要求

#define FLAG_TX_START                   0x10    // 送信開始
#define FLAG_TX_DONE                    0x20    // 送信完了
```

▼リスト 3.3　チャンネルコントローラーステータス定義：受信

```
#define FLAG_RX_SET_START               0x01    // 設定開始
#define FLAG_RX_SET_DONE                0x02    // 設定完了
#define FLAG_RX_SET_REQ_CLR_DONE        0x04    // 完了クリア要求
#define FLAG_RX_SET_REQ_ABORT           0x08    // 受信中断要求

#define FLAG_RX_START                   0x10    // 受信開始
#define FLAG_RX_DONE                    0x20    // 受信完了
```

3.4　コマンドについて

コマンドは、本ボードを制御するために使用します。
各コマンドは、対応する API が存在しています。詳しくは本ボードの API リファレンスマニュアルをご覧ください。
API は、コマンド番号 + 必要なパラメータを送信バッファに書き込みます。
コマンドの処理結果などは、以下に説明するコマンドフォーマットで受信バッファに書き込まれます。

3.5　コマンドフォーマット

コマンドは、送信バッファにコマンドヘッダとデータをセットで書き込みます。

▼表 3.1　コマンドヘッダフォーマット

オフセット	意味	サイズ (byte)	備考
0x00	コマンド No	2	
0x02	データサイズ	2	
0x04	パラメータ	4	
0x08	データ	x	データもパラメータも入る

3.6　コマンド

コマンドは、1byte で表される番号です。HPCIe-BRGGPI4INT は、送信バッファにコマンドが書き込まれ、送信ステータスが送信設定完了となると実行します。
以下に、コマンド No とパラメータを示します。

▼表 3.2　コマンド一覧

コマンド No	意味	パラメータ１	パラメータ２	パラメータ３	パラメータ４
0x0d	GET コマンドを受信したかを確認	なし	なし	なし	なし
0x0f	DCL/SDC/IFC コマンドを受信したかを確認	なし	なし	なし	なし
0x11	スレーブとして初期化	なし	なし	なし	なし
0x12	スレーブとしてほかの GBIB デバイスへデータ送信	uint8_t 送信先アドレス	uint8_t 送信データ文字列	なし	なし
0x13	スレーブとしてほかの GBIB デバイスからデータ受信	uint8_t 送信元アドレス	なし	なし	なし
0x14	SRQ 発行、シリアルポーリング要求	uint8_t statusbyte	なし	なし	なし
0x21	マスターとして初期化	なし	なし	なし	なし
0x22	マスターとしてほかの GBIB デバイスへデータ送信	uint8_t 送信先アドレス	uint8_t 送信データ文字列	なし	なし
0x23	マスターとしてほかの GBIB デバイスからデータ受信	uint8_t 送信元アドレス	なし	なし	なし
0x24	マルチラインメッセージを送信	uint8_t CommandByte	なし	なし	なし
0x25	シリアルポーリング処理、SRQ 発行元からのステータスバイトを取得	なし	なし	なし	なし
0x29	すべてのデバイスの初期化	なし	なし	なし	なし
0x2a	GET コマンドを送りトリガーをかける	uint8_t TargetAddr	なし	なし	なし

3.7　戻り値

戻り値は、受信バッファにコマンドヘッダ＋戻り値データで書き込まれます。コマンドヘッダには、実行したコマンド No、サイズ、戻り値が書き込まれます。

▼表 3.3　コマンド戻り値一覧

コマンド No	意味	戻り値１	戻り値２
0x0d	GET コマンドを受信したかを確認	0:受信していない 1:受信した	なし
0x0f	DCL/SDC/IFC コマンドを受信したかを確認	0:受信していない 1:受信した	なし
0x11	スレーブとして初期化	なし	なし
0x12	スレーブとしてほかの GBIB デバイスへデータ送信	なし	なし
0x13	スレーブとしてほかの GBIB デバイスからデータ受信	受信データ	なし
0x14	SRQ 発行、シリアルポーリング要求	なし	なし
0x21	マスターとして初期化	なし	なし
0x22	マスターとしてほかの GBIB デバイスへデータ送信	なし	なし
0x23	マスターとしてほかの GBIB デバイスからデータ受信	受信データ	なし
0x24	マルチラインメッセージを送信	なし	なし
0x25	シリアルポーリング処理､SRQ 発行元からのステータスバイトを取得	uint8_t statusbyte	uint8_t 要求した機器のアドレス
0x29	すべてのデバイスの初期化	なし	なし
0x2a	GET コマンドを送りトリガーをかける	なし	なし

3.8　送信バッファ

コマンドや送信データを設定するバッファです。ユーザーのプログラムで使用できるサイズは 8000byte です。各チャンネル毎にバッファがあります。自由に読み書き可能です。

3.9　受信バッファ

コマンドの戻り値や受信データを格納するバッファです。サイズは 8000byte です。各チャンネル毎にバッファがあります。読込専用です。

3.10　エラーコード

エラーコードは API のエラーコードとコマンドのエラーコードがあります。
API のエラーコードは、API の戻り値で返ります。

▼リスト 3.4　API エラーコード定義

```
enum ErrorCode
        RET_SUCCESS         = 0x00000000, //!< 正常.
        NOT_FOUND           = 0x00000001, //!< デバイスが見つからない
        ALREADY_OPENED      = 0x00000002, //!< 既にオープン済のデバイスをオープン.
        INSUFFICIENT_MEMORY = 0x00000004, //!< デバイス情報格納メモリが不足.
        INVALID_HANDLE      = 0x00000008, //!< 無効なデバイスハンドルを指定.

        LOST_DEVICE         = 0x00000010, //!< デバイスとの通信ができない
        ILLEGAL_PARAM       = 0x00000020, //!< 関数の引数の値が異常.
        ILLEGAL_DEVICE      = 0x00000040, //!< デバイスオープン時にボード情報が正しく読み出せない
```

```
        DEVICE_BUSY                     = 0x00001000, //!< デバイスはBusy状態
        FAILED_API                      = 0x00002000, //!< APIエラー
        DATA_EMPTY                      = 0x00004000, //!< データはない

        UNDEFINED_ERROR     = 0x000FFFFF  //!< 未定義エラー
;
```

コマンドのエラーコードは、API の

▼**リスト** 3.5　**コマンドのエラーコード定義**

```
enum CmdErrorCode
        OK = 0,                 // OK
        TIMEOUT,                // タイムアウト
        ILLEGAL_PARAM,  // パラメータ異常
        NOT_READY,              // 準備ができていない
        UNKWON,                 // 不明なエラー
;
```

第 4 章

プログラミング

4.1　この文書は

HPCIe-BRGGPI4INT の API を使用したプログラミングについて、プログラミングを説明するドキュメントです。

4.2　プログラミングのための準備

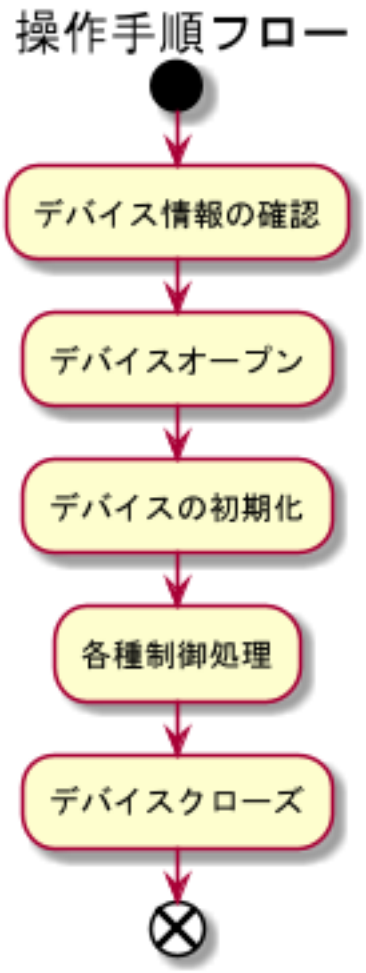
本ボードを操作するプログラミングをするために、以下のヘッダを include し、以下のライブラリをリンクしてください。

必要なインクルードファイル、ライブラリファイル

- brggpi4Intdll.h
- command_def.h
- error_code.h
- BRGGPI4INT_Dll.lib

4.3　操作手順

本ボードを制御するための基本的な処理手順を以下に示します。



▲図 4.1　操作手順フロー

4.4　デバイスを操作するための準備

本ホードを操作するためには、まず PC にインストールされている本ボードの枚数や情報を得る必要があります。
本ボードは、ボード ID スイッチを変更する事によって 1 台の PC に最大 16 枚インストールする事ができます。
プログラムでは、まず PC にインストールされている本ボードの枚数などを取得します。
次にインストールされているボードの枚数を使用して、デバイス情報を取得します。
デバイス情報は、BRGGPI4INTINFO 構造体へ格納します。
デバイス情報構造体には、API を使用するために必要なデバイスハンドルが格納されています。

▼リスト 4.1　BRGGPI4INTINFO 構造体

```
typedef struct _BRGGPI4INTINFO
        HANDLE hDeviceHandle;           //!< デバイスハンドル
        HANDLE hInterruptHandler;       //!< 割り込みハンドル
        DWORD ManagedNumber;            //!< デバイスハンドル作成の為の番号
        DWORD BoardID;                  //!< ボードID
        DWORD IsOpen;                   //!< デバイスオープンされたかのフラグ
        DWORD IsCanceled;               //!< 未使用
 BRGGPI4INTINFO, *PBRGGPI4INTINFO;
```

PC にインストールできる本ボード枚数は最大 16 枚なので、BRGGPI4INTINFO 構造体の配列を要素数 16 で定義します。
以下のコードでは、ボード数の取得、デバイス情報の取得を行っています。

▼リスト 4.2　getdeviceinfo()

```
DWORD g_totalBoard = 0; // 装着ボード数
BRGGPI4INTINFO g_infoDat[16];   // デバイス情報構造体

bool getDeviceInfo()

    brggpi4Int_CountDevice(&g_totalBoard);  // ボード枚数の確認
    if(g_totalBoard == 0)
        return false;   // ボードが1枚もインストールされていない！


    memset(&g_infoDat[0], 0, sizeof(BRGGPI4INTINFO) * 16);  // デバイス情報構造体の初期化
    brggpi4Int_LoadDeviceInfo(g_totalBoard, g_infoDat); // インストールされているすべてのボード情報を得る

    return true;
```

4.5　デバイスのオープン

制御したいデバイスのハンドルを API に渡す事でデバイスを制御できます。デバイスのハンドルは BRGGPI4INTINFO 構造体 BRGGPI4INTINFO 構造体のメンバ変数として定義されています。

デバイスを制御するためには、制御したいデバイスをオープンする必要があります。

以下の例では、取得したデバイス情報の 0 番目のデバイスをオープンしています

▼リスト 4.3　デバイスオープン

```
    // デバイスオープン
    if ((ret = brggpi4Int_OpenDevice(&g_infoDat[0])) != 0)
        return 0xffffffff;
```

4.6　デバイスの初期化

デバイスの制御を開始する前にデバイスを初期化しておきます。

API の brggpi4Int_SetBoardReset() を使用するとボードリセットをかけデバイスを初期化します。

ボードリセット後は、500ms~1s 程プログラムを待機します。

▼リスト 4.4　デバイスの初期化

```
    brggpi4Int_SetBoardReset(g_infoDat[0].DeviceHandle);   // ボードリセットをかけておく
    ::Sleep(500);   // 初期化処理が終わるまで待つ
```

4.7　デバイスのクローズ

デバイスの制御を終了する時は、デバイスをクローズします。

以下の例では、取得したデバイス情報の 0 番目のデバイスをクローズしています。

▼リスト 4.5　デバイスクローズ

```
    brggpi4Int_CloseDevice(&g_infoDat[0]);
```

4.8 制御プログラミング概要

制御プログラミングには、コマンドを使用します。コマンドは、API 関数によってボードへ送信されます。コマンドには、送信系と受信系の 2 つの種別があります。送信系は、送信完了で処理が完了するコマンドです。受信系は、コマンド送信、データ受信で完了するコマンドです。

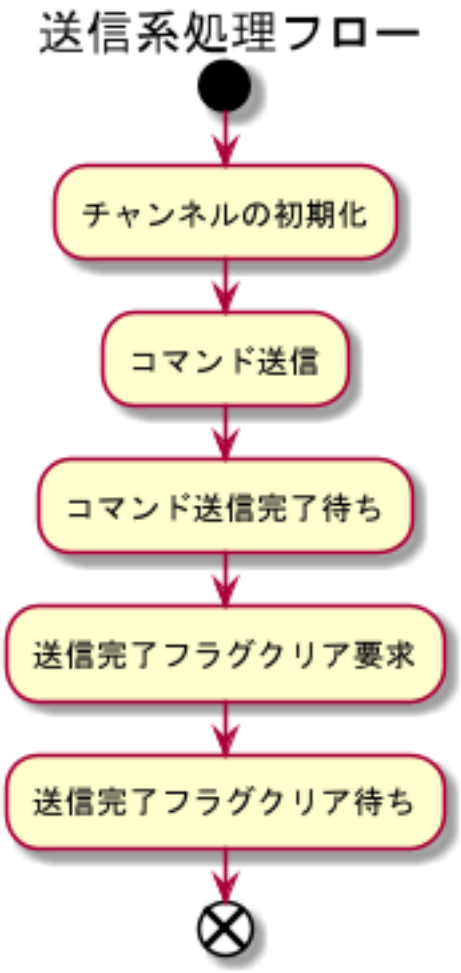
▼**リスト** 4.6　**送信系コマンド**

```
        ・指定チャンネルをスレーブとして初期化する
BRGGPI4INT_API DWORD WINAPI brggpi4Int_InitializeSlave(HANDLE deviceHandle, BYTE channel);

・指定チャンネルをマスターとして初期化する
BRGGPI4INT_API DWORD WINAPI brggpi4Int_InitializeMaster(HANDLE deviceHandle, BYTE channel);

・スレーブとしてデータを送信する
BRGGPI4INT_API DWORD WINAPI brggpi4Int_SlaveSendMessage(HANDLE deviceHandle, BYTE channel,
                                                WORD datasize, BYTE* pSendData,BYTE addr);

・マスターとしてデータを送信する
BRGGPI4INT_API DWORD WINAPI brggpi4Int_MasterSendMessage(HANDLE deviceHandle, BYTE channel,
                                                WORD datasize, BYTE* pSendData,BYTE addr);
・指定したチャンネルにステータスバイトを設定しSRQを発行する
BRGGPI4INT_API DWORD WINAPI brggpi4Int_RequestSerialPoll(HANDLE deviceHandle, BYTE channel, BYTE Statusbyte);

・マルチラインインターフェースメッセージを送信する
BRGGPI4INT_API DWORD WINAPI brggpi4Int_SendMultilineMessage(HANDLE deviceHandle, BYTE channel,BYTE CommandByte);

・GET指令を送る
BRGGPI4INT_API DWORD WINAPI brggpi4Int_SendGET(HANDLE deviceHandle, BYTE channel, BYTE TargetAddr);

・デバイスALLクリア指令を出す
BRGGPI4INT_API DWORD WINAPI brggpi4Int_ClearAllDevice(HANDLE deviceHandle, BYTE channel);
```

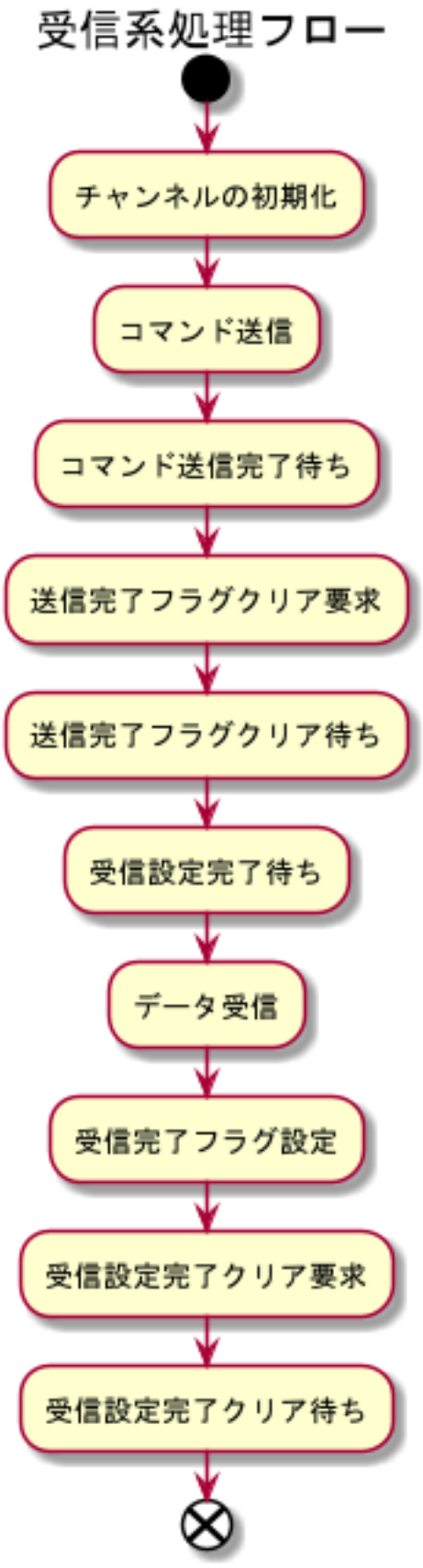
▼**リスト** 4.7　**受信系コマンド**

```
・スレーブとしてデータを受信する
BRGGPI4INT_API DWORD WINAPI brggpi4Int_SlaveReceiveMessage(HANDLE deviceHandle, BYTE channel, BYTE addr);
・マスターとしてデータを受信する
BRGGPI4INT_API DWORD WINAPI brggpi4Int_MasterReceiveMessage(HANDLE deviceHandle, BYTE channel, BYTE addr);
・指定したチャンネルでシリアルポーリングを行い、SRQを発行したGPIB機器から応答を取得する
BRGGPI4INT_API DWORD WINAPI brggpi4Int_ConductSerialPoll(HANDLE deviceHandle, BYTE channel);
・GET指令を受信したか確認する
BRGGPI4INT_API DWORD WINAPI brggpi4Int_QueryGETReceived(HANDLE deviceHandle, BYTE channel);
・デバイスクリア指令を受信したか確認する
BRGGPI4INT_API DWORD WINAPI brggpi4Int_QueryCleared(HANDLE deviceHandle, BYTE channel);
```

制御プログラミングのための処理フローを以下に示します。処理フローは、送信系のフローと受信系のフローがあります。送信系は、送信完了で処理が完了するコマンドでのフローです。受信系は、コマンド送信ーデータ受信で完了するコマンドのフローです。



▲図 4.2　送信系処理フロー



▲図 4.3　送信系処理フロー

4.9 コマンドの送信

コマンドを送信しボードへ処理を依頼します。
コマンドは、API を使用して送信します。
API はデバイスハンドル、チャンネル番号などをパラメータとして指定して呼び出します。
以下の例では、取得したデバイス情報の 0 番目のデバイスのチャンネル 0 をアドレス 1, デリミタを CR に設定しています。

▼**リスト** 4.8　API **の呼び出し**

```
    brggpi4Int_WriteAddress(g_infoDat[0].hDeviceHandle, 0, 1); // アドレス指定
    brggpi4Int_WriteDelimiter(g_infoDat[0].hDeviceHandle, 0, 0x01); // デリミタ設定
```

4.10 コマンド処理待ち

コマンド送信後、処理終了を待ちます。コマンドによって、処理待ちの手順が違います。

- 送信系のコマンド
  - 送信完了で処理が完了するコマンドです。送信完了フラグ (0x02) を待ちます。
  - 送信完了フラグ (0x02) が立ったら、完了クリア要求フラグ (0x04) を設定し送信完了フラグがクリアされるのを待ちます。
  - 送信完了フラグがクリアされたら、完了クリア要求フラグをクリアします。

コマンド送信後は、コマンドが終了したかを確認する必要があります。

▼**リスト** 4.9　chkSendCmdDone()

```
// 送信系コマンドが終了したか？
bool chkSendCmdDone(HANDLE hDeviceHandle, uint8_t ch,BYTE* pChkLv)

    bool result = true;
    uint8_t data;

    switch (*pChkLv)

    case 0: // 送信完了チェック
        brggpi4Int_GetSendStatus(hDeviceHandle, ch, &data);
        if (data & FLAG_TX_DONE)       // 送信完了？
            data = FLAG_TX_REQ_SET_DONE_CLR;    // 完了クリア要求を出す
            brggpi4Int_SetSendSettingStatus(hDeviceHandle, ch, data);
            *pChkLv += 1;

        break;
    case 1: // 完了クリア
        brggpi4Int_GetSendStatus(hDeviceHandle, ch, &data);
        if (data == 0)    // クリアされた？
            data = 0x00;    // 完了 フラグをクリアする
            brggpi4Int_SetSendSettingStatus(hDeviceHandle, ch, data);
            *pChkLv += 1;

        break;
    case 2: // エラー確認
        brggpi4Int_ReadChannelCtrlCmdState(hDeviceHandle, ch, &data);
        if (data == CMD_STATE_ERROR)  // エラーステート
            result = false;

        brggpi4Int_ReadChannelCtrlErrorInfo(hDeviceHandle, ch, &data);
        if (data)
            result = false;


        brggpi4Int_ReadMainCtrlErrorInfo(hDeviceHandle, &data);
        if (data)
```

```
            result = false;


        brggpi4Int_ReadChannelCtrlErrorInfo(hDeviceHandle, ch, &data);
        if (data)
            result = false;

        *pChkLv += 1;
        break;

    return result;
```

- 受信系のコマンド
  - 受信設定完了で処理が完了するコマンドです。コマンドの送信完了 (0x02) を待ち、次に受信設定完了 (0x20) まで待ちます。
  - 送信完了フラグが立ったら、完了クリア要求フラグ (0x04) を設定し送信完了フラグがクリアされるのを待ちます。
  - 送信完了フラグがクリアされたら、完了クリア要求フラグをクリアします。
  - 受信設定完了フラグが立ったら、受信開始フラグ (0x10) を立てて受信バッファからデータを読み出します。
  - 受信バッファからデータを読み出し完了したら、受信終了フラグ (0x20) を立てます。
  - 完了クリア要求フラグ (0x04) を待ち、完了クリア要求フラグが立ったら受信終了フラグをクリアします。

▼**リスト** 4.10　recive_Done()

```
// 受信系コマンドが終了したか？
bool chkRecvCmdDone(HANDLE hDeviceHandle, uint8_t ch, BYTE* pChkLv)

    bool result = true;
    uint8_t data;

    switch (*pChkLv)

    case 0: // 送信完了チェック
        brggpi4Int_GetSendStatus(hDeviceHandle, ch, &data);
        if (data & FLAG_TX_DONE)       // 送信完了？
            data = FLAG_TX_REQ_SET_DONE_CLR;    // 完了クリア要求を出す
            brggpi4Int_SetSendSettingStatus(hDeviceHandle, ch, data);
            *pChkLv += 1;

        break;
    case 1: // 完了クリア
        brggpi4Int_GetSendStatus(hDeviceHandle, ch, &data);
        if (data == 0)    // クリアされた？
            data = 0x00;    // 完了 フラグをクリアする
            brggpi4Int_SetSendSettingStatus(hDeviceHandle, ch, data);
            *pChkLv += 1;

        break;
    case 2: // 設定完了チェック
        brggpi4Int_GetRecvSettingStatus(hDeviceHandle, ch, &data);
        if (data & FLAG_RX_SET_DONE)    // 受信設定完了？
            // 受信開始
            brggpi4Int_Receive(hDeviceHandle, ch, &g_reciveData[ch][0]);    // データ受信
            *pChkLv += 1;

        break;
    case 3: // 完了クリア要求待ち
        brggpi4Int_GetRecvSettingStatus(hDeviceHandle, ch, &data);
        if (data & FLAG_RX_SET_REQ_CLR_DONE)    // 完了クリア要求か？
            data = 0x00;    // 完了 フラグをクリアする
            brggpi4Int_SetRecvDataStatus(hDeviceHandle, ch, data);
            *pChkLv += 1;

        break;
    case 4: //
        brggpi4Int_GetRecvSettingStatus(hDeviceHandle, ch, &data);
        if (data == 0)    // クリアされた？
            *pChkLv += 1;
```

```
        break;
    case 5:
        brggpi4Int_ReadChannelCtrlCmdState(hDeviceHandle, ch, &data);
        if (data == CMD_STATE_ERROR)  // エラーステート
            result = false;

        brggpi4Int_ReadChannelCtrlErrorInfo(hDeviceHandle, ch, &data);
        if (data)
            result = false;


        brggpi4Int_ReadMainCtrlErrorInfo(hDeviceHandle, &data);
        if (data)
            result = false;


        brggpi4Int_ReadChannelCtrlErrorInfo(hDeviceHandle, ch, &data);
        if (data)
            result = false;

        *pChkLv += 1;
        break;


    return result;
```

4.11 処理結果の読み出し

処理結果は受信バッファへコマンドヘッダ＋データで格納されます。
コマンドヘッダには、コマンド番号、受信データサイズが書かれています。

▼表 4.1 コマンドヘッダフォーマット

オフセット	意味	サイズ (byte)	備考
0x00	コマンド No	2	
0x02	データサイズ	2	
0x04	パラメータ	4	
0x08	データ	x	データもパラメータも入る

戻り値のあるコマンドの場合、コマンドヘッダのデータ以降に戻り値が格納されています。
以下のメモリイメージは、” シリアルポーリング処理、SRQ 発行元からのステータスバイトを取得 ” のコマンドで、statusbyte = 0x41, 要求機器アドレスが 0x01 の場合を表しています。

▼表 4.2 SRQ 受信コマンド実行後の受信バッファイメージ

オフセット	内容	意味
0x0000	0x25	コマンド番号
0x0002	0x06	データサイズ
0x0004	0x00	パラメータ 0
0x0005	0x00	パラメータ 1
0x0006	0x00	パラメータ 2
0x0007	0x00	パラメータ 3
0x0008	0x41	status byte
0x0009	0x01	要求機器アドレス

受信バッファは、brggpi4Int_RX_ReadByte(),brggpi4Int_RX_ReadWoad(),brggpi4Int_RX_ReadDWord() の API で読み出す事ができます。

4.12 データ送受信処理説明

ここからは、より具体的に処理内容を説明します。

チャンネルを初期化しデータの送信、受信の手順を説明します。

4.13 チャンネルの初期化

初期化処理は、マスターとして初期化するか、スレーブとして初期化するかを選択します。
本ボードは電源投入時にはスレーブとして初期化されます。
初期化には、チャンネルに設定するアドレス、デリミタの設定値を設定する必要があります。

- アドレス:0-30
- デリミタ
- bit 0：CR
- bit 1：LF
- 0 の場合 EOI

▼**リスト** 4.11　init_ch()

```
brggpi4Int_WriteAddress(hDeviceHandle, ch, addr); // アドレス指定
brggpi4Int_WriteDelimiter(hDeviceHandle, ch, delim); // デリミタ設定
if (master_f)  // マスターの場合
    brggpi4Int_InitializeMaster(hDeviceHandle, ch); // マスターとして初期化

else
    brggpi4Int_InitializeSlave(hDeviceHandle, ch); // スレーブとして初期化


// コマンドの処理が終わるのを待つ
bool result = true;
BYTE chkLv = 0;
while (1)
    ::Sleep(1);
    result = chkSendCmdDone(hDeviceHandle, ch, &chkLv);
    if (chkLv == 3)  // 終了か？
        break;
```

4.14 データ送信

データ送信は、マスターとして送信するか、スレーブとして送信するかで使用する API が変わります。送信するデータの格納されているアドレスと送信データサイズを指定します。送信データの先頭の 1byte は、送信先の機器のアドレスを設定します。
[送信条件]

- Master として Slave へ送信
- 送信先アドレス 1
- 送信データサイズ 256byte
- デリミタ CR+LF

上記送信条件時の送信バッファ中のバイトイメージは、以下のようになります。

▼表 4.3　送信データのイメージ

オフセット	意味	値	サイズ (byte)	備考
0x00	コマンド No	0x22	2	Master として送信した場合
0x02	データサイズ	264	2	コマンド No、データサイズ、パラメータのサイズも含む
0x04	パラメータ 0	1	1	送信先の機器アドレス
0x05	パラメータ 1	1	0	なし
0x06	パラメータ 2	1	0	なし
0x07	パラメータ 3	1	0	なし
0x08	データ 0 バイト	x	1	実際のデータの始まり
0x09	データ 1 バイト	x	1	実際のデータの始まり
:				
0x0102	データ 258 バイト	0x0d	1	デリミタ CR
0x0103	データ 259 バイト	0x0a	1	デリミタ LF

▼リスト 4.12　send_data()

```
if (master_f)  // マスターとして送信
    brggpi4Int_MasterSendMessage(hDeviceHandle, ch, size, pSendData,addr);
else    // スレーブとして送信
    brggpi4Int_SlaveSendMessage(hDeviceHandle, ch, size, pSendData,addr);

// 相手先のGPIB機器を受信状態にする
// :
// :
// :

// コマンドの処理が終わったかを確認
bool result = true;
BYTE chkLv = 0;
::Sleep(1);
result = chkSendCmdDone(hDeviceHandle, ch, &chkLv);
if (chkLv == 3)  // 終了か？
    // 送信完了の処理


// 他のチャンネルの処理など
// :
// :
// :
```

API 呼び出し後は、コマンド処理完了を待機します。送信完了 (0x02) フラグが立てば、次のデータを送信できます。次のデータを送信する前に、送信完了フラグをクリアするようにボードへ依頼します (完了クリア要求:0x04) 上記の例では、これらの処理はすべて chkSendCmdDone() 関数で行っています。先に説明した送信系コマンド終了待ち処理を行っています。

4.15　データ受信

データ受信は、マスターとして受信するか、スレーブとして受信するかで使用する API が変わります。受信データのバッファのアドレスと機器のアドレスを指定します。

▼リスト 4.13　recive_data()

```
if (master_f)  // マスターとして受信
    brggpi4Int_MasterReceiveMessage(hDeviceHandle, ch, addr);
else    // スレーブとして受信
    brggpi4Int_SlaveReceiveMessage(hDeviceHandle, ch, addr);

// 相手先のGPIB機器を送信状態にする
// :
// :
// :
```

```
// コマンドの処理が終わったかを確認
bool result = true;
BYTE chkLv = 0;
::Sleep(1);
result = chkRecvCmdDone(hDeviceHandle, ch, &chkLv);
if (chkLv == 6)  // 終了か？
    // 受信完了処理


// 他のチャンネルの処理など
// :
// :
// :
```

受信の処理では、本ボードを受信状態に設定します。そののち、相手先の機器を送信状態とします。受信処理が完了までは、先に説明した受信系コマンド終了待ち処理をおこないます。

[受信条件]

- Master として受信
- 送信元アドレス 2
- 送信データサイズ 256byte
- デリミタ CR+LF

上記送信条件時の送信バッファ中のバイトイメージは、以下のようになります。受信時の受信バッファ中のバイトイメージは、以下のようになります。

▼表 4.4　受信データのイメージ

オフセット	意味	値	サイズ (byte)	備考
0x00	コマンド No	0x23	2	Master として受信した場合
0x02	データサイズ	264	2	コマンド No、データサイズ、パラメータのサイズも含む
0x04	パラメータ 0	1	2	送信元の機器アドレス
0x05	パラメータ 1	1	0	なし
0x06	パラメータ 2	1	0	なし
0x07	パラメータ 3	1	0	なし
0x08	データ 0 バイト	x	1	実際のデータの始まり
0x09	データ 1 バイト	x	1	実際のデータの始まり
:				
0x0102	データ 258 バイト	0x0d	1	デリミタ CR
0x0103	データ 259 バイト	0x0a	1	デリミタ LF

第 5 章

サンプルプログラム

5.1 概要

添付サンプルプログラム"SampGPI4INT"は、HPCIe-BRGGPI4INT　2 枚による相互データ通信を行うサンプルプログラムです。

本製品による基本的な受信または送信処理の参考にしてください。

5.2 動作環境

- OS : Windows10 64bit
- Tool : Visual Studio2019
- 言語 : C++
- HPCIe-BRGGPI4INT × 2 枚 (送信用と受信用でそれぞれ 1 枚ずつ使用)

5.3 機能

- 下記の機能が実行可能です。
  - デバイスオープン
  - GPIB ボードの初期化設定
  - データ送信
  - データ受信

サンプルプログラム起動後、自動的にボードの初期化を行います。マスター・スレーブやチャンネル・デリミタの設定についてはソース内で決め打ちの値を入力しています。

送信データは、プロジェクトフォルダ内のファイルを読み出して送信します。

受信データは、受信した後、データ内容をコンソール画面に表示します。

付録 A

用 語 集

HPCIe-BRGGPI4INT を使用する上で必要な用語集です。

GPIB

汎用インターフェースバス。IEEE488.1、IEEE488.2 バスとも言われます。デジタル 8bit パラレル通信インターフェースです。

コントローラー

すべてのデバイスは、コントローラーか、トーカー、リスナーのいずれかの状態になります。コントローラーは、SRQ の応答、GPIB コマンドの送信、制御の受け渡しをおこなえます。

トーカー

コントローラーからトーカーとして設定されます。トーカーはデータ送信ができます。1 度に 1 つだけトーカーになる事ができます。

リスナー

コントローラーからリスナーとして設定されます。トーカーが送信したデータを読み取ります。複数のデバイスがリスナーになる事ができます。

マスター

コントローラーと同じような役割を持ちます。SRQ の応答、GPIB コマンドの送信、制御の受け渡しをおこなえます。

スレーブ

トーカーもしくはリスナーになります。

データライン

GPIB 信号の DIO1-DIO8 までデータラインといいます。データラインはコマンドメッセージとデータメッセージを伝送します。

インターフェース管理ライン

GPIB 全体の情報を管理する 5 つの信号ラインをインターフェース管理ラインと言います。インターフェース管理ラインには、IFC、ATN、REN、EOI、SRQ があります。

IFC(インターフェースクリア)

すべてのデバイスを静止状態にします。システムコントローラーによってアサートされます。

ATN(Attention:注意)

現在のデータタイプをデバイスに通知します。ATN がアサートされている場合はバス上のデータはコマンドメッセージと解釈されます。

REN(リモートイネーブル)

REN がアサートされている場合、デバイスはリモートプログラミングできます。システムコントローラーによってアサートされます。

EOI(End or Identify:完了または識別)

データの終わりを通知します。現在のトーカーによってアサートされます。

SRQ(サービスリクエスト)

デバイスからコントローラーへ通知されます。コントローラーでポーリングされるとアサートが解除されます。

ハンドシェイクライン

NRFD,NDAC,DAV の 3 つの信号ラインをハンドシェイクラインと呼びます。ハンドシェイクラインはデバイス

間のメッセージを制御します。

NRFD(not ready for data)

データの準備が出来ていない状態でアサートされます。

NDAC(not data accepted)

リスナーがデータの受け入れを完了していない状態でアサートされます。

DAV(data valid)

DIO ラインに送りだされたデータが有効である事を示します。

GPIB コマンド

GPIB の規格で定められたコマンドとデバイス固有のコマンドがあります。GPIB の規格で定められたコマンドであっても対応していない機器があります。

マルチラインメッセージ

デバイスに依存しない共通メッセージ。GPIB コマンド、バスラインコマンドとも言います。

更新履歴

2022/06/15 Ver. 1.0

初版。基本的な項目を記述。

2022/08/23 Ver. 1.0.1

要望対応で関数の引数変更、ファームパラメータ追加。受信時のバッファイメージを追加。

HPCIe-BRGGPI4INT software Manual

2022 年 6 月 15 日　新規作成。 v1.0.0
2022 年 8 月 23 日　 v1.0.1

発行所　株式会社ハイバーテック
連絡先　株式会社 ハイバーテック、東京都江東区新大橋 1-8-11 大樹生命新大橋ビル、TEL 03-3846-3801、FAX 03-3846-3773、sales@hivertec.co.jp