Interface Board

IF-0145-2

（ＰＣＩバスタイプ）

アクセス・マニュアル

有限会社　図工

１．はじめに

PCI バスタイプ　Interface Board（以下ＩＦボードと略称）には、１６チャンネル AD コンバータ、１６チャンネル DA コンバータ、１６チャンネル PWM、１６チャンネルカウンター、６４チャンネル PI、６４チャンネル PO、４チャンネルの外部割込みが集約されている。PC 側のアプリケーションが PCI バスコントローラを通じてインタフェース IC チップを直接に制御する（図 1-1）。よって、アプリケーションがそれぞれの IC チップの内蔵レジスタに直接指示データを書き込んだり、結果データを読みこんだりする必要がある。IC チップによって内蔵レジスタのロケーションや内容定義やアクセス方法などが全く異なるので、ＩＦボード上の全ての IC チップのマニュアルとデータシートを読んで理解しなければならない。但し、ＩＦボードに高性能のドライバソフトが添付されているのでそれを利用すれば、IC チップのマニュアルやデータシートを読む必要がなく、インタフェース IC の制御が非常に簡単になる。詳しくは第 3 章を参照。

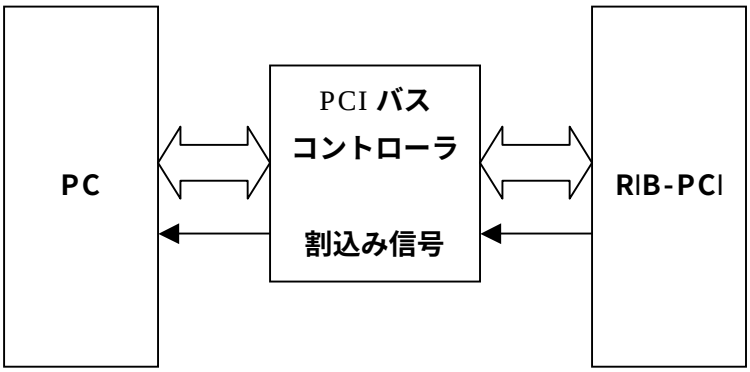

図 1-1

PCI ボードの IO アドレスは、PC が立ち上がるときに BIOS によって自動的に決められるので、ハード上での設定をする必要がない。ドライバソフトを利用すれば、ＩＦボードの IO アドレスを知る必要がないが、直接アクセスする場合はＩＦボードの IO アドレスを知らなければならない。ドライバソフトがＩＦボードの IO アドレスを取得する API 関数も提供している。ＩＦボードの IO アドレスが分かれば、各インタフェース IC のアドレスが次の通りになる。

表１-１

インタフェースIC	IC アドレス
AD コンバータ（CH０１～０８）	ベースアドレス＋０ｘ００
AD コンバータ（CH０９～１６）	ベースアドレス＋０ｘ１０
DA コンバータ（CH０１～０８）	ベースアドレス＋０ｘ２０
DA コンバータ（CH０９～１６）	ベースアドレス＋０ｘ３０
カウンター（CH01～０２）	ベースアドレス＋０ｘ４０
カウンター（CH0３～０４）	ベースアドレス＋０ｘ４８
カウンター（CH0５～０６）	ベースアドレス＋０ｘ５０
カウンター（CH0７～０８）	ベースアドレス＋０ｘ５８
PIO（CH０１～３２）	ベースアドレス＋０ｘ６０
PWM（CH０１～１６）	ベースアドレス＋０ｘ８０
PIO（CH３３～６４）	ベースアドレス＋０ｘC０
カウンター（CH0９～１０）	ベースアドレス＋０ｘE０
カウンター（CH１１～１２）	ベースアドレス＋０ｘE８
カウンター（CH１３～１４）	ベースアドレス＋０ｘF０
カウンター（CH１５～１６）	ベースアドレス＋０ｘF８

ここでベースアドレスはＩＦボードのIO アドレスで、IC アドレスは、IC 内蔵レジスタのスタートアドレスである。各内蔵レジスタのアドレスとスタートアドレスとの関係は、IC のマニュアルを参考にする。これらのマニュアルが製品には添付されていないので、必要な場合、弊社の技術サポートの方に問い合わせる。

ＩＦボードは、４チャンネルの外部割込みを発生する以外、他の割り込みを一切発生しない。４チャンネルの割り込みが一つの割り込み番号を共用し、番号の割振りは PC が立ち上がるときに BIOS によって自動的に決められる。ドライバソフトを利用する場合、割り込み番号を知らなくてもいいが、ドライバソフトが割り込み番号を取得する API 関数も提供している。

ＩＦボードのIO アドレスと割り込み番号は BIOS によって自動的に割り付けられるので、他の IO ボードのアドレスと割込み番号と衝突する可能性がほとんどない。また、ＩＦボードを二枚以上同時に使う場合も、IO アドレスと割込み番号が重複するような心配も要らない。

ＩＦボードを PC に取り付けて利用する前に、まず付属のフロッピーディスクのフォルダ「A: \Win98 \Test」にあるテストプログラムを実行して、ボード上の全てのインタフェースチップが正常に稼動しているかどうかを確認する。

２．Windows ドライバ・ソフト仕様

この節は、Windows 95/98 を利用しているユーザーがＩＦボードを簡単にアクセスできるために提供されているドライバソフトについて説明する。Windows 95/98 では、アプリケーションが直接ハードウェアのリソースをアクセスするのではなく、ドライバソフトを通じてアクセスする方法が推薦されている。特に割込みイベントをキャッチするのはドライバソフトしかできない。従って、本ボードもドライバソフトを提供している。

一、Windows ドライバのファイル構成

ドライバソフトは次のファイルから構成されている。

ファイル名	役割説明
Jwribpci.inf	ドライバのインストール情報が格納されている。
Jwribpci.vxd	ＩＦボードをアクセスするドライバ本体である。
Jwribpci.dll	アプリケーションとのインタフェースを提供する動的ライブラリで、API 関数が含まれる。
Jwribpci.lib	API 関数のリンク情報を含める静的なライブラリで、アプリケーションとリンクする必要がある。
Jwribpci.h	API 関数のプロトタイプが含まれている。アプリケーションのソースプログラムにインクルードする必要がある。

上記のファイルが付属のフロッピーディスクのフォルダ「\Win98 \Driver」に格納されている。これらのプログラムが全て３２ビットなので、３２ビットのアプリケーションでしか利用できない。

二、Windows ドライバのインストール

ＩＦボードを PC に取り付けて、PC を起動すると、本製品がプラグ＆プレイで認識され、新しいハードウェアの追加ウィザードが起動する。以下の手順に従ってドライバをインストールする。

●「次の新しいドライバを検索しています」が表示されている画面で＜次へ＞ボタンを押す。

●「検索方法を選択して下さい」が表示されている画面で、「使用中のデバイスに最適なドライバを検索する（推奨）」を選んで＜次へ＞ボタンを押す。

●「検索場所の指定」に「A: \Win98 \Driver」を入力して＜次へ＞ボタンを押す。

●PC が起動したら、付属のフロッピーディスクのフォルダ「\Win98 \Driver」に格納されているファイル「Jwribpci.vxd」と「Jwribpci.dll」をアプリケーションと同じフォルダにコピーするか、Windows のシステムフォルダ（通常「C: \Windows \System」）にコピーする。

●最後に付属のフロッピーディスクのフォルダ「\Win98 \Driver」に格納されているファイル「Jwribpci.lib」と「Jwribpci.h」をアプリケーションのソースフォルダにコピーする。

以上、ドライバのインストールが終わる。

三、Windows ドライバの利用方法

Windows ドライバは、アプリケーションが簡単にインタフェースボードをアクセスするために、１１種類の API 関数を用意している。アプリケーションが Jwribpci.lib とリンクすれば、API 関数を使ってＩＦボードを制御することができる。以降、これらの API 関数について詳しく説明する。

１．アドレス取得関数

関数名	int GetRegAdrEx(int iRibNo)
機　能	ＩＦボードの IO アドレスを取得する。
引　数	iRibNo: ＩＦボードの番号（0 から始まる）
戻り値	Not -1: IO アドレス　　-1: 異常終了

２．割込み番号取得関数

関数名	int GetIrqNoEx(int iRibNo)
機　能	ＩＦボードが使う割込みの番号を取得する。
引　数	iRibNo: ＩＦボードの番号（0 から始まる）
戻り値	Not -1: 割込み番号　　-1: 異常終了

３．割込みイベントメッセージ設定関数

関数名	int SetWndMsgEx(int iRibNo, HANDLE hWnd, UINT uMsg)
機　能	ＩＦボードに割込みが発生したときに Windows アプリケーションに知らせるためのメッセージを設定する。
引　数	iRibNo: ＩＦボードの番号（0 から始まる） hWnd: メッセージを受け取る Window のハンドル uMsg: 割込みイベントに対応するメッセージ（0xC000 以上の整数）
戻り値	0: 正常終了　　Not 0: 異常終了

４．割込みイベントメッセージ許可関数

関数名	int EnableIntMsgEx(int iRibNo, BOOL bEnable)
機　能	割込みイベントが発生したら Windows メッセージを生成するかどうかを設定する。割込みを禁止するかどうかを設定する関数でないことに注意。
引　数	iRibNo: ＩＦボードの番号（0 から始まる） bEnable: メッセージ許可フラグ（TRUE：許可、FALSE：禁止）
戻り値	0: 正常終了　　Not 0: 異常終了

5．パラメータのないコマンド送信関数

関数名	BOOL SendCmdEx(int iRibNo, int iChNo, WORD wCmd)
機　能	パラメータを持っていないコマンドをＩＦボードに送信する。
引　数	iRibNo: ＩＦボードの番号（0 から始まる） iChNo: チャンネル番号（0 から始まる） wCmd: 送信するコマンド（第３章を参考）
戻り値	TRUE: 正常終了　　FALSE: 異常終了

6．パラメータのあるコマンド送信関数

関数名	BOOL SendCmdDataEx(int iRibNo, int iChNo, WORD wCmd, WORD wData)
機　能	パラメータを持っているコマンドをＩＦボードに送信する。
引　数	iRibNo: ＩＦボードの番号（0 から始まる） iChNo: チャンネル番号（0 から始まる） wCmd: 送信するコマンド（第３章を参考） wData: コマンドが持っているパラメータ
戻り値	TRUE: 正常終了　　FALSE: 異常終了

7．ダブルワード・パラメータのあるコマンド送信関数

関数名	BOOL SendCmdDWDataEx(int iRibNo, int iChNo, WORD wCmd, DWORD dwData)
機　能	ダブルワード・パラメータを持っているコマンドをＩＦボードに送信する。
引　数	iRibNo: ＩＦボードの番号（0 から始まる） iChNo: チャンネル番号（0 から始まる） wCmd: 送信するコマンド（第３章を参考） dwData: コマンドが持つダブルワード・パラメータ（例えばカウンターの値）
戻り値	TRUE: 正常終了　　FALSE: 異常終了

8．データ受信関数

関数名	BOOL ReadDataEx(int iRibNo, WORD *pwData, int iCount)
機　能	ＩＦボードからデータを受信する。
引　数	iRibNo: ＩＦボードの番号（0 から始まる） pwData: 受信データを受取るバッファーのポイント。バッファーの長さは iCount ワード以上でなければならない。 iCount: 受信データのワード数
戻り値	TRUE: 正常終了　　FALSE: 異常終了

9．ダブルワード・データ受信関数

関数名	BOOL ReadDWDataEx(int iRibNo, DWORD *pdwData)
機　能	ＩＦボードからダブルワード・データを受信する。
引　数	iRibNo: ＩＦボードの番号（0 から始まる） pdwData: ダブルワード・受信データ（例えばカウンターの値）を受取るバッファーのポイント。バッファーの長さはダブルワードである。
戻り値	TRUE: 正常終了　　FALSE: 異常終了

10．割り込み時のコールバック関数の設定関数

関数名	int SetVxdCallBackEx(int iRibNo, RIBVXDCALLBACK *pUserProc, DWORD dwCallBackSize, void *pUserData, DWORD dwUserDataSize)
機　能	割り込み発生時に自動的に呼び出される関数（割り込み処理関数）を設定する。
引　数	iRibNo: ＩＦボードの番号（0 から始まる） pUserProc: 割り込み処理関数（コールバック関数）のポインタ dwCallBackSize: コールバック関数のサイズ。正確にわからない場合に多めに設定する。 pUserData: 割り込み処理に必要なデータを格納するバッファーのポインタ dwUserDataSize: データバッファーのサイズ
戻り値	0: 正常終了　　Not 0: 異常終了

11．ボード番号取得関数

関数名	int GetRibNoEx(int iRibID)
機　能	ＩＦボードの番号を取得する。
引　数	iRibID: ＩＦボードの ID（製造シリアル番号）
戻り値	Not -1: ボード番号　　-1: 異常終了

補足説明：

●引数 iRibNo はＩＦボードの番号を各関数に渡す。１枚のＩＦボードのみを使う場合、番号を常に０に設定すればいい。２枚以上のＩＦボードを同時に使う場合、GetRibNoEx 関数を使って、ＩＦボードの製造シリアル番号の下 4 桁を引数として渡して、当該ボードの番号を取得する。PC の拡張ボードの構成によって番号が変わる可能性があるので、各ＩＦボードの番号を定数として定義するのではなく、アプリケーションの初期化ルーチンに GetRibNoEx を呼び出して動的にボードの番号を決める。

●引数 iChNo はチャンネル番号である。ＩＦボードでは各種インタフェースがそれぞれ複数のチャンネルを持っているので、インタフェースをアクセスする多くのコマンドはチャンネルを指定しなければならない。チャンネルを指定する必要のないコマンドの場合、チャンネル番号を０にする。また、チャンネル番号が０番から始まるので、チャンネル N をアクセスしたい場合、iChNo に N-1 をセットしなければならない。

●使いたい機能が標準コマンドでサポートしていない場合､又はコマンドを利用せずに直接 IC のレジスタをアクセスしたい場合、GetRegAdr と GetIrqNo を使ってＩＦボードの IO アドレスと割り込み番号を取得する。

●外部割込みが発生するとき、ドライバソフトがそれを検出し、外部割込み番号（１から始まる）をパラメータ wParam のローバイトにセットしてメッセージ uMsg と一緒にアプリケーションに送る。従って、アプリケーション側が uMsg を受け取ったら wParam によって外部割込み番号を知り、対応する処理プログラムを実行すればいいです。二枚以上のＩＦボードを利用する場合、それぞれ異なる割込みイベントメッセージを設定すれば、どのＩＦボードに割込みが発生したかを判断できる。一方、パラメータ lParam に割込みの発生回数をセットしている。

●Windows メッセージによる割込み処理がどうしても遅くて間に合わない場合、CallBack 関数を利用して、割り込み発生時に必要最低限の処理を CallBack 関数に移せば、割込みに対する高速の反応が可能になる。CallBack 関数の定義が次の通りである。

```
typedef int (RIBVXDCALLBACK)(WORD wIntNo, BOOL*
bEnableMessage, void* pUserData, DWORD dUserDataSize);
```

ここで、wIntNo は外部割込み番号で、ドライバが CallBack 関数に渡す。bEnableMessage は、割込み発生に対応するメッセージをアプリケーションに送信するかどうかを決めるフラグで、CallBack 関数がドライバに渡す。TRUE の場合、ドライバが割込みメッセージをアプリに送信し、FALSE の場合、送信しないことになる。 PUserData と dUserDataSize は 関数 SetVxdCallBackEx により渡されたデータバッファーのポインタとサイズをドライバがそのまま CallBack 関数に渡す。CallBack 関数で処理したいデータを SetVxdCallBackEx により渡すことになる。

●アプリケーションは一度に一つの外部割込みをしか処理できないので、ＩＦボードでは割込みが発生すると、割込み発生フラグ INT がセットされ、次の割込みを待たせる。従って、アプリケーション側が割込み処理を完了したら、速やかにコマンド INT_Clear をドライバに送信してフラグ INT をクリアしなければならない。詳しくは第 3 章のコマンド説明を参照。

付属フロッピーディスクのフォルダ「 \Win98 \Sample」にドライバが提供している API 関数の使い方を示すサンプルプログラムが格納されている。

３．コマンド仕様

ＩＦボードは、ユーザーのアプリケーションが簡単に入出力インタフェースをアクセスするために作られたコマンドをサポートする。コマンドの数は表３-１ に示されている２３種類のコマンドになる。

表３-１

ｺｰﾄﾞ	記　号	説　　明	ﾁｬﾝﾈﾙ	渡すﾃﾞｰﾀ	貰うﾃﾞｰﾀ
10H	AD_Read	AD からﾃﾞｰﾀを読み込む	あり	なし	変換結果
20H	DA_Write	DA へﾃﾞｰﾀを書き込む	あり	変換ﾃﾞｰﾀ	なし
30H	PWM_Start	PWM 出力を開始する	あり	なし	なし
31H	PWM_Stop	PWM 出力を停止する	あり	なし	なし
32H	PWM_Rate	出力ﾊﾟﾙｽの周波数を設定する	あり	周波数	なし
33H	PWM_Duty	出力ﾊﾟﾙｽのﾃﾞｭｰﾃｨｰを設定する	あり	ﾃﾞｭｰﾃｨｰ	なし
40H	ENC_Start	ｴﾝｺｰﾀﾞｰｶｳﾝﾀを開始する	あり	なし	なし
41H	ENC_Stop	ｴﾝｺｰﾀﾞｰｶｳﾝﾀを停止する	あり	なし	なし
42H	ENC_Mode	入力ﾓｰﾄﾞと逓倍数を設定する	あり	ﾓｰﾄﾞ/逓倍数	なし
43H	ENC_Read	ｶｳﾝﾀの値を読み出す	あり	なし	ｶｳﾝﾀ値
44H	ENC_Clear	ｶｳﾝﾀの値をｸﾘｱする	あり	なし	なし
45H	ENC_Zphase	ｴﾝｺｰﾀﾞｰ Z 相の動作を設定する	なし	ﾁｬﾝﾈﾙ･ﾌﾗｸﾞ	なし
46H	ENC_Zploe	ｴﾝｺｰﾀﾞｰ Z 相の極性を設定する	なし	極性	なし
47H	ENC_Set	ｶｳﾝﾀの値を設定する	あり	ｶｳﾝﾀ値	なし
50H	PIO_Read	１６Bit 単位でﾎﾟｰﾄから入力する	あり	なし	入力ﾃﾞｰﾀ
51H	PIO_Write	１６Bit 単位でﾎﾟｰﾄへ出力する	あり	出力ﾃﾞｰﾀ	なし
52H	PIO_Set	指定ﾎﾟｰﾄをﾊｲにｾｯﾄする	あり	なし	なし
53H	PIO_Clear	指定ﾎﾟｰﾄをﾛｰにｾｯﾄする	あり	なし	なし
60H	INT_Enable	外部割込みを許可する	あり	なし	なし
61H	INT_Disable	外部割込みを禁止する	あり	なし	なし
62H	INT_Mode	外部割込みﾄﾘｶﾞｰﾓｰﾄﾞを設定する	あり	ﾄﾘｶﾞｰﾓｰﾄﾞ	なし
63H	INT_Clear	割込みﾌﾗｸﾞをｸﾘｱする	なし	なし	なし
FFH	ALL_Reset	IO ﾎﾞｰﾄﾞをﾘｾｯﾄし初期状態に戻す	なし	なし	なし

以下、これらのコマンドについて詳しく説明する。

(1) AD_Read

これは AD コンバーターから変換データを読み込むコマンドである。ドライバがこのコマンドを受信したら、直ちに AD 変換を開始し変換終了を待つ。AD 変換が終了したらアプリケーションに戻る。戻ってきたらアプリケーションは直ちに ReadDataEx 関数を呼び出して変換結果を取得する。AD 変換が 1.6μs しかかからないので、このコマンドが非常に効率が良く使いやすい。

変換結果は 12Bit のデータで、0000H から 0FFFH までの範囲に入る。電圧 V と変換データ D との関係は次の式で計算できる。

$$V=(D-2048.0)\times 20.0/4096.0$$

例 12 チャンネルの AD 変換データを読み込みたい場合、次のようにを書けばいい。

```
SendCmdEx(0, 11, 0x10);   ReadDataEx(0, &wData,1);
```

(2) DA_Write

これは DA コンバーターへ変換データを書き込むコマンドである。ドライバがこのコマンドを受信したら、直ちに変換データを DA にセットし変換を開始する。

変換データは 12Bit で、0000H から 0FFFH までの範囲に入る。変換データ D と電圧 V との関係は次の式で計算できる。

$$D=INT(V\times 4096.0/20.0+2047.5)$$

ここで、INT は整数部のみを取る関数を示す。

初期状態では全ての DA の出力が０Ｖになっている。

例 12 チャンネルの DA に 1V を出力したい場合、次のように書けばいい。

```
SendCmdDataEx(0, 11, 0x20, 0x08CD);
```

(3) PWM_Start

このコマンドは指定したチャンネルの PWM に対してパルス信号出力の開始を指示する。

例 チャンネル 4 の PWM を開始させたい場合、次のように書けばいい。

```
SendCmdEx(0, 3, 0x30);
```

(4) PWM_Stop

このコマンドは指定したチャンネルの PWM に対してパルス信号出力の停止を指示し、出力をハイ・インピーダンスにする。初期状態では全ての PWM の出力がハイ・インピーダンスになっている。

例 チャンネル 4 の PWM を停止させたい場合、次のように書けばいい。

```
SendCmdEx(0, 3, 0x31);
```

(5) PWM_Rate

このコマンドは指定したチャンネルの PWM に対してパルス信号の周波数を設定する。16 チャンネルの PWM の周波数をそれぞれ設定できる。設定可能な範囲は 156.25Hz～40kHz である。ドライバに渡すデータ D（0～255）とパルス周波数 F の関係は次のようになっている。

F＝40000／(256-D)（Hz）

デフォルト周波数は、F＝156.25Hz（D＝0）になっている。

例 チャンネル 4 の周波数を 2kHz に設定したい場合、次のように書けばいい。

```
SendCmdDataEx(0, 3, 0x32, 0xEC);
```

(6) PWM_Duty

このコマンドは指定したチャンネルの PWM に対してパルス信号のデューティーを設定する。16 チャンネルの PWM のデューティーをそれぞれ設定できる。ドライバに渡すデータはパルス・デューティーである。デューティーの設定可能な範囲は 0～100 である。100 以上の数値を渡しても 100%のデューティーになる。

例 チャンネル 4 の PWM のデューティーを 16%に設定したい場合、次のように書けばいい。

```
SendCmdDataEx(0, 3, 0x33, 0x10);
```

(7) ENC_Start

このコマンドは指定したチャンネルのカウンタにカウントの開始を指示する。

例 チャンネル 3 のカウンタを開始させたい場合、次のように書けばいい。

```
SendCmdEx(0, 2, 0x40);
```

(8) ENC_Stop

このコマンドは指定したチャンネルのカウンタにカウントの停止を指示する。

例 チャンネル 3 のカウンタを停止させたい場合、次のように書けばいい。

```
SendCmdEx(0, 2, 0x41);
```

(9) ENC_Mode

このコマンドは指定したチャンネルのカウンタに対し、入力モードと逓倍数を設定する。ドライバに渡すデータと入力モード・逓倍数との関係は表３－２のように示す。デフォルトの入力モードは「1 相入力、Up／Down カウント」になっている。

例 チャンネル 3 のカウンタを 2 相入力、4 逓倍、差異カウントモードに設定したい場合、次のように書けばいい。

```
SendCmdDataEx(0, 2, 0x42, 0x03);
```

表 3-2

渡すデータ	入力モード・逓倍数
0	1 相入力、Up／Down カウント
1	2 相入力、1 逓倍、差異カウント
2	2 相入力、2 逓倍、差異カウント
3	2 相入力、4 逓倍、差異カウント

(10) ENC_Read

このコマンドは指定したチャンネルのカウンタからカウント値を読み出す。コマンドを送信した後に、ReadDWDataEx により２４ビットのカウンタ値を読み出す。

例 チャンネル 3 のカウンタ値を読み出したい場合、次のように書けばいい。

```
SendCmdEx(0, 2, 0x43);   ReadDWDataEx(0, &dwData);
```

(11) ENC_Clear

このコマンドは指定したチャンネルのカウンタをクリアする（カウント値＝０）。

例 チャンネル 3 のカウンタをクリアしたい場合、次のように書けばいい。

```
SendCmdEx(0, 2, 0x44);
```

(12) ENC_Zphase

このコマンドは 16 チャンネルのエンコーダーの Z 相信号に対して、Z 相信号が入力される（ハイからローに又はローからハイに変わる）とカウンタの値を自動クリアするかどうかを設定する。Z 相信号の極性はコマンド ENC_Zpole で設定する。マイコンに渡すデータは 16Bit で、1Bit は１チャンネルのエンコーダーと対応する。Bit0 はチャンネル 1 と、Bit15 はチャンネル１６と対応する。ビットをセットすると、自動クリアを許可し、ビットをクリアすると、自動クリアを禁止する。デフォルトでは全ての Z 相の自動クリアが禁止されている状態になっている。

例 チャンネル１、2 の Z 相自動クリアを許可し、それ以外の Z 相自動クリアを禁止したい場合、次のように書けばいい。

```
SendCmdDataEx(0, 0, 0x45, 0x0003);
```

(13) ENC_Zpole

このコマンドは 16 チャンネルのエンコーダーの Z 相信号の極性を一括で設定する。チャンネル毎には設定できない。ドライバに０渡すと、Z 相がアクティブ・ローに、１を渡すと、Z 相がアクティブ・ハイになる。

例 使うエンコーダーZ 相信号がアクティブ・ハイの場合、次のように書けばいい。

```
SendCmdDataEx(0, 0, 0x46, 0x0001);
```

(14) ENC_Set

このコマンドは指定したチャンネルのカウンタにカウント値を設定する。カウンタ値は 24 ビットなので、SendCmdDWDataEx によりコマンドを送信する。

例 チャンネル 3 のカウンタにカウント値 123456H を設定したい場合、次のように書けばいい。

```
SendCmdDWDataEx(0, 2, 0x47, 0x123456);
```

(15) PIO_Read

このコマンドは指定したポート・グループから 16Bit のデータを読み出す。64 チャンネルのポートは次のように四つのポート・グループに分けられる（表 3-3 をご参照）。このコマンドはチャンネルでなくグループ毎に指定するので、16 チャンネルのポートから同時にデータを読み出す。

表 3-3

ポート・グループ No	1	2	3	4
チャンネル範囲	1～16	17～32	33～48	49～64

例 チャンネル 17～32 のポートの値を読み出したい場合、グループ No-2 をセットする。

```
SendCmdEx(0, 1, 0x50);   ReadDataEx(0, &wData,1);
```

(16) PIO_Write

このコマンドは指定したポート・グループ（表 3-3 をご参照）に 16Bit のデータを書き出す。このコマンドはチャンネルでなくグループ毎に指定するので、16 チャンネルのポートに同時にデータを書き出すのでご注意する。1 チャンネル毎に出力したい場合にコマンド PIO_Set と PIO_Clear を利用する。

初期状態では全ての出力ポートがローになっている。

例 チャンネル 17～32 のポートを全てハイにセットしたい場合、グループ No-２をセットする。

```
SendCmdDataEx(0, 1, 0x51, 0xFFFF);
```

(17) PIO_Set

このコマンドは指定したチャンネルのポートをハイにセットする。

例 チャンネル 16 のポートをハイにセットしたい場合、次のように書けばいい。

```
SendCmdEx(0, 15, 0x52);
```

(18) PIO_Clear

このコマンドは指定したチャンネルのポートをローにセットする。

例 チャンネル 16 のポートをローにセットしたい場合、次のように書けばいい。

```
SendCmdEx(0, 15, 0x53);
```

(19) INT_Enable

このコマンドは指定したチャンネルの外部割込みを有効にする。その割込み入力端子にローからハイになる信号を与えると割込みが発生する。詳しくは「４．割込み仕様」を参照。

例 チャンネル 1 の外部割込みを許可したい場合、次のように書けばいい。

```
SendCmdEx(0, 0, 0x60);
```

(20) INT_Disable

このコマンドは指定したチャンネルの外部割込みを禁止する。その割込み入力端子に信号を与えても割込みが発生しない。

初期状態では全ての外部割込みが禁止状態になっている。

例 チャンネル 1 の外部割込みを禁止したい場合、次のように書けばいい。

```
SendCmdEx(0, 0, 0x61);
```

(21) INT_Mode

このコマンドは指定したチャンネルの外部割込みのトリガーモードを設定する。マイコンに渡すデータとトリガーモードとの関係は次のように示す。デフォルトモードはロー・レベル・センス・モードになっている。

表 3-4

渡すデータ	トリガーモード
0	ロー・レベル・センス・モード
1	立下りエッジ・トリガー・モード

例 チャンネル 2 の外部割込みを立下りエッジトリガーに設定したい場合、次のように書けばいい。

```
SendCmdEx(0, 1, 0x62);
```

(22) INT_Clear

このコマンドはＩＦボードにある割込み発生フラグ INT をクリアする。フラグ INT をクリアしないと新しい割込みが入れなくなるので、PC 側が割込みを受け付けて処理を済ませたら必ずこのコマンドを送って INT をクリアする。

例 割込み発生フラグ INT をクリアしたい場合、次のように書けばいい。

```
SendCmdEx(0, 0, 0x64);
```

(23) ALL_Reset

このコマンドはＩＦボードをリセットし、電源を入れた直後の初期状態に戻させる。緊急の場合、又は異常が発生した場合、このコマンドを送信すれば全ての出力を停止できる。

例 ＩＦボードの全てのチャンネルの出力を停止したい場合、次のように書けばいい。

```
SendCmdEx(0, 0, 0xFF);
```

４．Linux ドライバ・ソフト仕様

Linux ドライバは Windows ドライバ違って、ソースコードの形で提供している。従って、次の二点に注意する。

●ドライバソフトを利用する前に、ソースコードをコンパイルして実行モジュールを作成し、システムにインストールする作業を行う必要がある。

●提供するソースコードをカスタマイズして利用することも可能であるが、カスタマイズにより生じる不具合などについてはサポート外になる。

一、Linux ドライバのソースファイル構成

ドライバのソースファイル一覧が次のようになっている。

ファイル名	役割説明
README	ドライバに関する説明（使用する前に必ず読む）
Makefile	ドライバのコンパイルとインストール
JwRib.mk	標準 LINUX 用ドライバの作成時にインクルードするファイル
RibPciMod.c	ドライバ・モジュールのソースファイル
RibPciMod.h	ドライバ・モジュール用ヘッドファイル
Rib_Intr.c	ユーザーが作成する割込み処理ルーチンのソールファイル
RibDef.h	ドライバやその他のアプリケーションが利用する定義ファイル
Hardware.h	ハードウェアに関するヘッドファイル
RibApic.h	PCI コントロールに関するヘッドファイル
RibFunc.h	関数の宣言用ヘッドファイル
RibMain.c	ＩＦボード仮想 I/O インタフェースのソースファイル
RibFunc.c	各種共通関数のソースファイル
RibApic.c	APIC コントロール制御関数のソースファイル
RibInt.c	割込み制御関数のソースファイル
RibAd.c	A/D コンバータ制御関数のソースファイル
RibDa.c	D/A コンバータ制御関数のソースファイル
RibEnc.c	エンコーダカウンター制御関数のソースファイル
RibPio.c	PIO 制御関数のソースファイル
RibPwm.c	PWM 制御関数のソースファイル

上記のファイルが全て付属のフロッピーディスクのフォルダ「/Linux/Driver」に格納されている。

二、Linux ドライバの構築方法

以下の手順に従ってドライバのソースコードをコンパイルしてインストールする。

●まず、スーパーユーザーとしてログインする。

●フロッピーディスクの Linux ディレクトリをそのままハードディスクのあるディレクトリにコピーして、Linux/Driver のディレクトリにカレントディレクトリにする。

●コマンド make clean を実行して、古いドライバを削除する。

●コマンド make を実行して、ソースファイルをコンパイルする。

●コマンド make dev を実行して、デバイスファイルを作成する。コマンド「ls/dev/jwribpci*」を実行してデバイスファイルが作られたことを確認する。

●コマンド make insmod を実行して、ドライバをインストールする。

注意事項：

●ドライバごとに割り付けられる番号――メジャー番号は、デフォルトで９９になっている。この番号が既に使われている場合に、別の番号に変更する必要がある。そのとき、ファイル Makefile と RibPciMod.h の中身も一緒に変更しなければならない。

●インストールしたドライバを削除したい場合、コマンド make rmmod を実行する。

三、Linux ドライバの利用方法

Windows ドライバと同じように、アプリケーションが簡単にインタフェースボードをアクセスするために、１０種類の API 関数を用意している。これらの API 関数の定義と宣言はそれぞれ付属のフロッピーディスクのフォルダ「/Linux/lib」に格納されているファイル「Riblo.c」と「Riblo.h」に入っている。これらの API 関数を利用するアプリケーションは必ずこの二つのファイルと一緒にコンパイルしリンクしなければならない。以降、これらの API 関数について詳しく説明する。

１．ドライバ開始関数

関数名	int OpenRibEx(int iRibNo)
機　能	指定した番号のＩＦボードをアクセスするドライバを開始する。
引　数	iRibNo: ＩＦボードの番号（0 から始まる）
戻り値	>=0: 正常終了　　<0: 異常終了

２．ドライバ終了関数

関数名	int CloseRibEx(int iRibNo)
機　能	指定した番号のＩＦボードをアクセスするドライバを終了する。
引　数	iRibNo: ＩＦボードの番号（0 から始まる）
戻り値	>=0: 正常終了　　<0: 異常終了

３．アドレス取得関数

関数名	int GetRegAdrEx(int iRibNo)
機　能	ＩＦボードの IO アドレスを取得する。
引　数	iRibNo: ＩＦボードの番号（0 から始まる）
戻り値	>=0: IO アドレス　　<0: 異常終了

４．割込み番号取得関数

関数名	int GetIrqNoEx(int iRibNo)
機　能	ＩＦボードが使う割込みの番号を取得する。
引　数	iRibNo: ＩＦボードの番号（0 から始まる）
戻り値	>=0: 割込み番号　　<0: 異常終了

５．パラメータのないコマンド送信関数

関数名	BOOL SendCmdEx(int iRibNo, int iChNo, WORD wCmd)
機　能	パラメータを持っていないコマンドをＩＦボードに送信する。
引　数	iRibNo: ＩＦボードの番号（0 から始まる） iChNo: チャンネル番号（0 から始まる） wCmd: 送信するコマンド（第３章を参考）
戻り値	TRUE: 正常終了　　FALSE: 異常終了

６．パラメータのあるコマンド送信関数

関数名	BOOL SendCmdDataEx(int iRibNo, int iChNo, WORD wCmd, WORD wData)
機　能	パラメータを持っているコマンドをＩＦボードに送信する。
引　数	iRibNo: ＩＦボードの番号（0 から始まる） iChNo: チャンネル番号（0 から始まる） wCmd: 送信するコマンド（第３章を参考） wData: コマンドが持っているパラメータ
戻り値	TRUE: 正常終了　　FALSE: 異常終了

7．ダブルワード・パラメータのあるコマンド送信関数

関数名	BOOL SendCmdDWDataEx(int iRibNo, int iChNo, WORD wCmd, DWORD dwData)
機　能	ダブルワード・パラメータを持っているコマンドをＩＦボードに送信する。
引　数	iRibNo: ＩＦボードの番号（0 から始まる） iChNo: チャンネル番号（0 から始まる） wCmd: 送信するコマンド（第３章を参考） dwData: コマンドが持つダブルワード・パラメータ（例えばカウンターの値）
戻り値	TRUE: 正常終了　　FALSE: 異常終了

8．データ受信関数

関数名	BOOL ReadDataEx(int iRibNo, WORD *pwData, int iCount)
機　能	ＩＦボードからデータを受信する。
引　数	iRibNo: ＩＦボードの番号（0 から始まる） pwData: 受信データを受取るバッファーのポイント。バッファーの長さは iCount ワード以上でなければならない。 iCount: 受信データのワード数
戻り値	TRUE: 正常終了　　FALSE: 異常終了

9．ダブルワード・データ受信関数

関数名	BOOL ReadDWDataEx(int iRibNo, DWORD *pdwData)
機　能	ＩＦボードからダブルワード・データを受信する。
引　数	iRibNo: ＩＦボードの番号（0 から始まる） pdwData: ダブルワード・受信データ（例えばカウンターの値）を受取るバッファーのポイント。バッファーの長さはダブルワードである。
戻り値	TRUE: 正常終了　　FALSE: 異常終了

10．ボード番号取得関数

関数名	int GetRibNoEx(int iRibID)
機　能	ＩＦボードの番号を取得する。
引　数	iRibID: ＩＦボードの ID（製造シリアル番号）
戻り値	>=0: ボード番号　　<0: 異常終了

補足説明：

●Windows ドライバと違って、Linux ドライバを利用するときに、必ず関数 OpenRibEx を呼び出してドライバを開始しなければならないし、利用後に必ず関数 CloseRibEx を呼び出してドライバを終了しなければならない。また、Linux ドライバでは、同時に使えるＩＦボードの数が４枚であることに注意する。

●Linux ドライバがソースファイルの形で提供されるので、割込み処理関数は CallBack 関数という形でなく、直接ドライバの中にある割込み処理関数の内容をカスタマイズして利用する形になる。割込み処理関数がソースファイル rib_intr.c に入っていて次のような形になっている。

```
static int rib_intr_routinue
   (int extintr,                  // 外部割込み番号(0-3)
    JWRIBISA_DEV *dev,   // カレント dev struct へのポインタ
    struct pt_regs* regs)    // プロセッサ・スナップショット(あまり使わない)
{
   // ここにユーザーが割込み処理プログラムを書き込む。
   // この割込み処理関数では時間のかかる処理を避けなければならない。
   return 0;
}
```

付属フロッピーディスクのフォルダ「/Linux/Sample」にドライバが提供している API 関数の使い方を示すサンプルプログラムが格納されている。