解説編

第3章

VisualBasicからの利用方法

…ドライバとファームウェアを呼び出すプログラミング

ここでは，Windows上のアプリケーション・プログラムからUSB汎用インターフェース・ボード（UCT-203）を制御するためのプログラミング方法を説明します．UCT-203上のファームウェアについては第2章を参照してください．

3-1　サイプレス社ドライバの使いかた

サイプレス社のソフトウェア・ツールをインストールしたときに組み込まれるUSBドライバは，単なるファームウェア・ダウンロード用のドライバではありません．開発ツールであるEZ-USBコントロール・パネルからUSB標準リクエストやベンダ・リクエストを発行することや，任意のエンド・ポイントのリード/ライトを行うなど，USBターゲットへの基本的なアクセスを一通りサポートした汎用のUSBドライバです．

このドライバのアプリケーションからの利用手順は次のようになります．

① ドライバをオープンする

Windows APIの一つである**CreateFile**を利用します．デバイス名は“¥¥.¥Ezusb-0”が1台目で，2台目，3台目以降は最後の“-0”が“-1”，“-2”になります．ただし，本書のサンプル・プログラムでは接続するデバイスは1台だけであることを前提にしています．

CreateFileで返されたハンドルを，デバイスとの通信やクローズ処理で使用します．

② デバイスと通信する

Windows APIの**DeviceIOControl**を利用して，ドライバとの間でコマンドやデータのやりとりを行います．

③ ドライバをクローズする

Widnows APIの一つである**CloseHandle**を使って，ドライバをクローズします．

＊　　＊

DeviceIOControlは汎用的なものであるため，多用された場合は，あまり読みやすくありませんし，プログラムを書くときの間違いも起きやすくなります．このことから，本書のサンプル・プログラムでは**DeviceIOControl**を呼び出すためのラッパ関数を，VisualBasicの標準モジュールとして用意しています．標準モジュールのラッパ関数だけを使うのであれば，以下の説明は読み飛ばしていただいてもかまいません．

標準モジュールで用意したラッパ関数については後で説明します．

● DeviceIOControl の方法

Windows API の **DeviceIOControl** では，次のような引数を与えます．サイプレス社ドライバでは，このうち **lpInBuffer** と **nInBufferSize** を使ってパラメータを与え，**lpOutBuffer** をデータ入出力用として利用するような使いかたをしています．

デバイスのハンドル（**CreateFile** でオープンしたときに取得したもの）
IOCTL 要求コード（マクロで値が定義）
InBuffer のアドレス（実体は 32 ビット長のパイプ番号データがあるだけ）
InBuffer のデータ長
OutBuffer の先頭アドレス
OutBuffer のデータ長
実際に転送されたデータ長を収めるアドレス
オーバーラップ（常に 0）

●サイプレス社ドライバの DeviceIOControl

サイプレス社のドライバはかなり汎用的に作られており，さまざまな IOCTL 要求が用意されています．ソース・コード（C:¥cypress¥USB¥Drivers¥ezusbdrv の中にある）を見ると，サイプレス社ドライバの用意している IOCTL 要求が行えるようになっていることがわかりますが，これらのうち，本書のサンプル・プログラムで使用したものは次の三つです．

▶ IOCTL_EZUSB_BULK_READ

バルク IN エンド・ポイントのリードを行います．GPIF モードやスレーブ FIFO モードによるデータ入力や，シリアル・ポートからのデータ入力に利用します．

▶ IOCTL_EZUSB_BULK_WRITE

バルク OUT エンド・ポイントのライトを行います．GPIF モードやスレーブ FIFO モードによるデータ出力や，シリアル・ポートへのデータ出力に利用します．

▶ IOCTL_EZUSB_VENDOR_OR_CLASS_REQUEST

ベンダ・リクエストを行います．PIO モードでのリード/ライトや動作モードの設定などはベンダ・リクエストで行いますので，これを利用します．

それぞれの使いかたは次のとおりです．

● IOCTL_EZUSB_BULK_READ

呼び出し方法の例は次のようになります．

```
Dim result As Long
Dim btc As BulkTransferControlType
```

```
result = DeviceIoControl(
            _hUSBDriver,
            _IOCTL_EZUSB_BULK_READ,
            _btc,
            _Len(btc),
            _buffer,
            _dataLen,
            _NofXfr,
            _0)
```

InBufferとして，**btc**（Bulk Transfer Control）構造体を渡します．**nInBufferSize**には**btc**構造体のサイズが入ります．**btc**の中身は1ロング・ワードのパイプ（エンド・ポイント）番号です．パイプ番号はエンド・ポイント・ディスクリプタに現れた順に0，1，2…と付けられる番号です．EzFirm/FX2では0がシリアル・ポート出力，1がシリアル・ポート入力，2がバルク・データOUT，3がバルク・データINとなります．

OutBufferがデバイスから読み出したデータを格納するバッファで，バッファ・サイズとして読み出したいデータ長を与えます．転送完了後に，実際にリードされたデータ・バイト数が**NofXfr**変数に格納されます．

● IOCTL_EZUSB_BULK_WRITE

呼び出し方法の例は次のようになります．

```
Dim result As Long
Dim btc As BulkTransferControlType
btc.lPipeNum = pipe
result = DeviceIoControl(
            _hUSBDriver,
            _IOCTL_EZUSB_BULK_WRITE,
            _btc,
            _Len(btc),
            _buffer,
            _dataLen,
            _NofXfr,
            _0)
```

InBufferとして，**btc**（Bulk Transfer Control）構造体を渡します．**nInBufferSize**には**btc**構造体のサイズが入ります．**btc**の中身は1ロング・ワードのパイプ（エンド・ポイント）番号です．パイプ番号については**IOCTL_EZUSB_BULK_READ**の説明を参照してください．

OutBufferがデバイスへ送りたいデータを格納するバッファで，バッファ・サイズとして送りたいデータ・サイズを与えます．転送完了後に，実際に送られたデータ・バイト数が**NofXfr**変数に格納されます．

● IOCTL_EZUSB_VENDOR_OR_CLASS_REQUEST

呼び出し方法の例は次のようになります．

```
Dim result As Long
Dim myRequest As VENDOR_OR_CLASS_REQUEST_CONTROL
myRequest.direction_in = Dir_In
myRequest.requestType = 2          'ベンダ定義リクエスト
myRequest.recepient = recepient
myRequest.request = bRequest
myRequest.value = wValue
myRequest.index = wIndex
result = DeviceIoControl(
            _hUSBDriver,
            _IOCTL_EZUSB_VENDOR_OR_CLASS_REQUEST,
            _myRequest,
            _Len(myRequest),
            _buffer,
            _dataLen,
            _NofXfr,
            _0)
```

InBufferとして，USBのデバイス・リクエスト構造体（**myRequest**）を渡します．フィールドの並びやサイズは以下のようなもので，USB規格に準拠したものになっています．

```
Public Type VENDOR_OR_CLASS_REQUEST_CONTROL
    direction_in As Byte
    requestType As Byte
    recepient As Byte
    requestTypeReservedBits As Byte
    request As Byte
    padding_byte As Byte
    value As Integer
    index As Integer
End Type
```

3-2　標準モジュールとして用意したラッパ関数

サイプレス社ドライバの呼び出しを**DeviceIOControl**で直接行うことも可能ですが，**DeviceIOControl**を

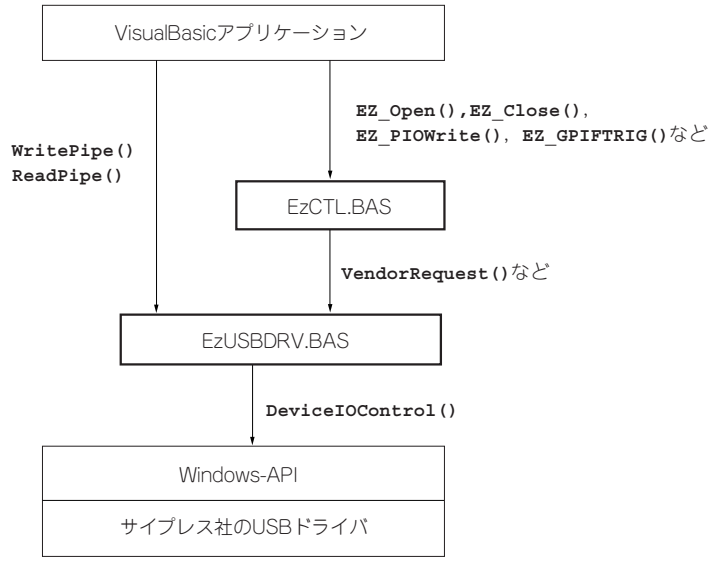

図1　標準モジュールの位置づけ

毎回記述することは面倒なことから，サンプル・プログラムではこれをラップしてアクセスしやすくしたラッパ関数を標準モジュールとして用意しました（付属CD-ROMに収録）．

用意した標準モジュールは，

EZUSBDRV.BAS

EZCTL.BAS

の二つがあります（**図1**）．

EZUSBDRV.BASは，サイプレス社ドライバの**DeviceIOControl**などのWindowsとのインターフェース部分をラップしたもので，デバイスのオープン/クローズやエンド・ポイント（パイプ）のリード/ライト，ベンダ・リクエストの発行などを行うものです．

EZCTL.BASのほうは，EZUSBDRV.BASの上にかぶさる形となり，ファームウェア（EzFirm/FX2）がもっているI/O操作などのベンダ・リクエストを簡単に行えるようにしたものです．

● EZUSBDRV.BASの関数

EZUSBDRV.BASには次のような関数が含まれます．

▶ WritePipe

```
Function WritePipe(
    ByVal hUSBDriver As Long,
    ByVal pipe As Integer,
    ByRef buffer As Byte,
```

```
        ByVal dataLen As Long,
        ByRef NofXfr As Long
        )  As Long
```

指定されたデバイス（**hUSBDriver**）の指定されたパイプ（エンド・ポイント；**pipe**）へ書き込みを行います．データ・バッファ領域は**buffer**で，書き込みデータ・サイズ（バイト）は**dataLen**で指定します．

パイプ番号（**pipe**）はエンド・ポイント・ディスクリプタに現れた順で，EzFirm/FX2の場合には，

0：シリアル・ポートOUT（送信）
1：シリアル・ポートIN（受信）
2：バルクOUT（GPIFモード/スレーブFIFOモードによる出力方向）
3：バルクIN（GPIFモード/スレーブFIFOモードによる入力方向）

となっています．

書き込み動作の完了後，実際に転送が行われたサイズ（バイト数）が**NofXfr**に格納されます．

▶ ReadPipe

```
    Function ReadPipe(
        ByVal hUSBDriver As Long,
        ByVal pipe As Integer,
        ByRef buffer As Byte,
        ByVal dataLen As Long,
        ByRef NofXfr As Long
        ) As Long
```

指定されたデバイス（**hUSBDriver**）の指定されたパイプ（エンド・ポイント；**pipe**）から読み込みを行います．データ・バッファ領域は**buffer**で，読み込みデータ・サイズ（バイト）は**dataLen**で指定します．

パイプ番号（**pipe**）はエンド・ポイント・ディスクリプタに現れた順で，EzFirm/FX2の場合には，

0：シリアル・ポートOUT（送信）
1：シリアル・ポートIN（受信）
2：バルクOUT（GPIFモード/スレーブFIFOモードによる出力方向）
3：バルクIN（GPIFモード/スレーブFIFOモードによる入力方向）

となっています．

読み込み動作の完了後，実際に転送が行われたサイズ（バイト数）が**NofXfr**に格納されます．

▶ VendorRequest

```
    Function VendorRequest(
        ByVal hUSBDriver As Long,
        ByVal Dir_In As Byte,
        ByVal recepient As Byte,
        ByVal bRequest As Byte,
```

```
    ByVal wValue As Integer,
    ByVal wIndex As Integer,
    ByRef buffer As Byte,
    ByVal dataLen As Long,
    ByRef NofXfr As Long
    ) As Long
```

指定されたデバイス（**hUSBDriver**）に対してベンダ・リクエストを発行します．**Dir_In**はデータ・ステージのデータ転送方向で，IN方向データがあるときは‘1’，OUT方向のデータがあるとき（**EZ_WaveSet**など）には‘0’にします．

recepientは，USBのデバイス・リクエストの**bmRequestType**バイトの**recipient**フィールドの値になります．EzFirm/FX2では，このフィールドの値は使用していません．

bRequestがベンダ・リクエストを示すコードで，以下**wValue**，**wIndex**は付随するパラメータです．ベンダ・リクエスト・コードやパラメータの値については，EzFirm/FX2の説明（第2章）を参照してください．

VendorRequest()関数は引数も多く，見た目が複雑になることから，応用編のサンプル・プログラムではEzFirm/FX2用のベンダ・リクエストをより簡単に発行できるようにした標準モジュール（EZCTL.BAS）を用意しています．

▶ OpenDriver

```
Function OpenDriver( sDevname As String ) As Long
```

デバイスをオープンします．**sDevname**はデバイスの識別文字列になります．サイプレス社ドライバでは複数のデバイスを同時に利用できるようになっており，1台目が“Ezusb-0”，2台目が“Ezusb-1”…となります．本書のサンプル・プログラムは1台目だけを対象にしています．

戻り値として，デバイスのハンドルが返ります．ハンドルがマイナスの場合にはデバイスのオープンに失敗したことを示します．

▶ CloseDriver

```
Function CloseDriver( hUSBDriver As Long ) As Long
```

デバイスをクローズします．引数にはオープン時に受け取ったハンドルを与えます．

● EZCTL.BASの関数

EZCTL.BASは，EzFirm/FX2専用のベンダ・リクエスト（コマンド）などを簡単に扱えるようにしたものです．EZCTL.BASはEZUSBDRV.BASの上に乗る形で実装されています．EZCTL.BASで用意した関数は次のとおりです．

▶ EZ_Open

```
Public Sub EZ_Open()
```

FX2デバイスをオープンします．EZCTL.BASでは1台のUCT-203だけをターゲットにしていますので，“Ezusb-0”デバイスに決めうちしています．オープンされたデバイスのハンドルは，グローバル変数の**hUSB**に

収められます．

▶ EZ_Close

```
Public Sub EZ_Close()
```

FX2デバイスをクローズします．

▶ EZ_SetPortConfig

```
Public Function EZ_SetPortConfig(
    ByVal Mode As Byte,
    ByVal Brate As Byte,
    ByVal PE As Byte,
    ByVal PD As Byte,
    ByVal PC As Byte,
    ByVal PB As Byte,
    ByVal PA As Byte
    ) as Long
```

FX2デバイスの動作モードやPIOポートの設定を行います．**Mode**が全体の動作モード選択，**Brate**がシリアル・ポートのビットレート設定，**PE**～**PA**が各ポートの入出力方向の設定になります．各パラメータの詳細はEzFirm/FX2の説明（第2章）を参照してください．

▶ EZ_WaveSet

```
Public Function EZ_WaveSet(
    ByVal wavenum As Byte,
    ByRef wsw As Byte
    ) As Long
```

GPIF用のウェーブフォーム・ディスクリプタを設定します．**wavenum**は，

0：バースト・リード（バルクINエンド・ポイントを使った転送）

1：バースト・ライト（バルクOUTエンド・ポイントを使った転送）

2：シングル・リード（**EZ_SglRd()**を利用した転送）

3：シングル・ライト（**EZ_SglWt()**を利用した転送）

となっています．**wsw**は実際のウェーブフォーム・データで，32バイト長に固定です．

▶ EZ_PIOWrite

```
Public Function EZ_PIOWrite(
    ByVal Port As Byte,
    ByVal data As Byte
    ) As Long
```

PIOポート（8ビット）にデータをセットします．**Port**がポート番号，**data**が設定するデータです．

▶ EZ_PIORead

```
Public Function EZ_PIORead(
    ByVal Port As Byte
    ) As Long
```

PIOポートのデータを読み込み，戻り値として返します．下位8ビットのみが有効です．

▶ EZ_AdrsCTLSet

```
Public Function EZ_AdrsCTLSet(
    ByVal Address As Integer,
    ByVal CTL As Integer
    ) As Long
```

GPIFモードで動作させているときの，GPIFADR端子，およびCTL出力端子の状態を設定します．**Address**がGPIFADR端子，**CTL**がCTL端子への設定になります．

▶ EZ_GPIFAbort

```
Public Function EZ_GPIFAbort() As Long
```

動作中のGPIFを強制停止させます．

▶ EZ_GPIFTrig

```
Public Function EZ_GPIFTrig(
    ByVal EpNum As Byte,
    ByVal XfrSize As Long
    ) As Long
```

GPIFによるバースト転送を開始します．**EpNum**はエンド・ポイント識別用の番号で，0がOUT方向（ライト方向），2がIN方向（リード方向）になります．転送データ・サイズ（ワード数）は**XfrSize**で指定します．

GPIFによる転送は常に16ビット幅で行われるため，転送されるデータのバイト数は**XfrSize**の2倍になることに注意してください．

▶ EZ_SglRd

```
Public Function EZ_SglRd(
    ByVal Address As Integer
    ) As Long
```

GPIFを利用したシングル・リード動作を行います．FD［0：15］端子から読み込まれた1ワード（16ビット）のデータが戻り値として返されます．**Address**は，GPIFADR端子への設定値です．シングル・リード動作の完了後，読み出されたデータが戻り値として返ります．

▶ EZ_SglWt

```
Public Function EZ_SglWt(
    ByVal Address As Integer,
    ByVal data As Integer
```

```
    ) As Long
```

GPIF を利用したシングル・ライト動作を行い，動作完了後にリターンします．**Address** はGPIFADR 端子への設定値，**Data** は出力したいデータで，下位16 ビットが有効です．シングル動作の完了後にリターンします．ライト動作の完了を待たずに次の動作に移りたい場合には，次に説明する **EZ_SglWtNW()** を使用してください．

▶ EZ_SglWtNW

```
Public Function EZ_SglWtNW(
    ByVal Address As Integer,
    ByVal data As Integer
    ) As Long
```

GPIF を利用したシングル・ライト動作を行い，ライト動作の完了を待たずにリターンします．**Address** はGPIFADR 端子への設定値，**Data** は出力したいデータです．ライト動作の完了を待ってから次の動作に移りたい場合には，**EZ_SglWt()** を使用してください．

●PIO モードの使いかた

PIO モードは，PORTA ～ PORTE のI/O ピンを入出力ポートとして使うモードです．

基本的な使いかたは，次のような手順となります．

① **EZ_Open** をコールしてデバイスをオープン
② **EZ_SetPortConfig** を使ってPIO モードに設定
③ **EZ_PIOWrite/EZ_PIORead** でポートのライト/リードを実行
④ **EZ_Close** をコールしてデバイスをクローズ

次の例では，PORTA（PA）を入力，それ以外を出力ピンとして設定して，PA にデータ出力，PB からデータ入力しています．

```
Dim sts as Long
Dim indata as Integer
    Call EZ_Open                          'FX2デバイスをオープン
    sts = Ez_SetPortConfig(0,2,1,1,1,0)   'PIOモード，PAが入力，PB～PEが出力
    sts = EZ_PIOWrite(0,1)                'PORTAに01hをライト
    indata = EZ_PIORead(1)                'PORTBをリード
    Call EZ_Close                         'FX2デバイスをクローズ
```

●スレーブFIFO モードの使いかた

スレーブFIFO モードでは，PORTB/PORTD がデータ・ライン（FD［0 : 15］）となり，PORTA が制御用の入出力信号に変わります．そのほかのポートは，PIO モードと同じように使うことができます．次に説明するGPIF モードと異なり，OUT 方向とIN 方向を同時に動作させておくことが可能です．

スレーブFIFO モードの場合，バルクIN，およびバルクOUT エンド・ポイントが外部との間のFIFO メモリに

なったような動作になります.

FLAGA/B/Cは，それぞれEP2-Empty（バルクOUTエンド・ポイントのEmptyフラグ），EP6-Full（バルクINエンド・ポイントのFullフラグ），EP6-Empty（バルクINエンド・ポイントのEmptyフラグ）になっており，いずれもHアクティブです．通常，OUT方向（ホスト→ターゲット）はEP2-Empty，IN方向（ターゲット→ホスト）はEP6-Fullを利用し，Lアクティブの転送要求信号のように使うのが便利でしょう.

スレーブFIFOモードの基本的な使いかたは次のようになります.

① **EZ_Open**をコールしてデバイスをオープンする

② **EZ_SetPortConfig()**を使ってスレーブFIFOモードに設定

③ **WritePipe()/ReadPipe()**を使ってエンド・ポイントのライト/リードを実行

④ 外部機器がFX2のFLAG信号を見ながらデータ転送を実行

⑤ **EZ_Close**をコールしてデバイスをクローズする

以下に例を示します.

```
Dim sts as Long
Dim data(2048) as Byte
Dim xfflen as Long
    Call EZ_Open                                          ' FX2デバイスをオープン
    sts = Ez_SetPortConfig(3,2,1,1,1,1)                   ' スレーブFIFOモード
    …
    sts = WritePipe(hUSB, 2, Data(0), 2048, xfrlen)       ' EP2にデータ書き込み
    sts = ReadPipe(hUSB, 3, Data(0), 2048, xfrlen)        ' EP6からデータ読み込み
    …
    Call EZ_Close                                         ' FX2デバイスをクローズ
```

●GPIFモードの使いかた

スレーブFIFOモードの場合，外部機器側がFX2のFIFOステータスをチェックしながらアクセスしなくてはなりませんので，外部回路側でCPUやFPGAなどのインテリジェントなデバイスが必要です．このインテリジェント・デバイスの代行を行うのがGPIFです．GPIFの動作はウェーブフォーム・ディスクリプタと呼ばれる，波形定義テーブルでプログラミングします.

また，GPIFはエンド・ポイントと外部の間をとりもって行うバースト・リード/ライト転送のほか，1回だけ転送を行って停止するシングル・リード/ライト機能もあります.

EzFirm/FX2ではこれらの両方，四つの転送モードをサポートしています.

GPIFモードの基本的な使いかたは次のようになります.

① **EZ_Open**を呼び出してデバイスをオープンする

② **EZ_SetPortConfig()**でGPIFモードに設定

③ **EZ_WaveSet()**で，ウェーブフォームを設定

＜バースト・リード/ライトの場合＞

④ **EZ_AdrsCTLSet** で，GPIFADR や CTL 信号の初期値を設定（省略可）

⑤ **EZ_GPIFTrig()** で，バースト・リード/ライトのいずれかを開始

⑥ **ReadPipe** または **WritePipe** でエンド・ポイントのリード/ライトを実行

⑦ **EZ_Close** を呼び出してデバイスをクローズする

＜シングル・リード/ライトの場合＞

④ **EZ_SglRd()** または **EZ_SglWt()** でシングル・リード/ライトを実行

⑤ **EZ_Close** を呼び出してデバイスをクローズする

実際のプログラムでは以下の例のようになります.

```
Dim sts as Long
Dim data(2047) as Byte
Dim xfrlen as Long
Dim wsw(31) as Byte
    Call EZ_Open        'FX2デバイスをオープン
    …
    sts = Ez_SetPortConfig(2,2,1,1,1,1)        'GPIF モード
    wsw(0)=xx:wsw(1)=xx:  …  :wsw(31)=xx
    sts = EZ_WaveSet(0, wsw(0))                'バースト・リード動作用
    wsw(0)=xx:wsw(1)=xx:  …  :wsw(31)=xx
    sts = EZ_WaveSet(1, wsw(0))                'バースト・ライト動作用
    wsw(0)=xx:wsw(1)=xx:  …  :wsw(31)=xx
    sts = EZ_WaveSet(2, wsw(0))                'シングル・リード動作用
    wsw(0)=xx:wsw(1)=xx:  …  :wsw(31)=xx
    sts = EZ_WaveSet(3, wsw(0))                'シングル・ライト動作用
    …
    sts = EZ_GPIFTrig(0, 1024)
    sts = WritePipe(hUSB, 2, Data(0), 2048, xfrlen)  'EP2データ書き込み
    …                                                '（バースト・ライト）
    sts = EZ_GPIFTrig(0, 1024)
    sts = ReadPipe(hUSB, 3, Data(0), 2048, xfrlen)   'EP6データ読み込み
    …                                                '（バースト・リード）
    sts = EZ_SglWt(1,2)              'GPIFADR ピンを 001h にして 0002h をライト
    data(0)=EZ_SglRd(2)              'GPIFADR ピンを 002h にして 1 ワード・リード
    sts = EZ_SglWtNW(1,2)            'SglWt と同様だが GPIF の動作完了を待たない
    Call EZ_Close                    'FX2デバイスをクローズ
```

3-3　ウェーブフォーム・ディスクリプタの構造と作りかた

GPIFはウェーブフォーム・ディスクリプタと呼ばれる，GPIF動作用のテーブルを参照しながら動作します．GPIFは8ステートのステート・マシンになっており，このうちステート7は停止状態を示すアイドル・ステートです．GPIFは起動されると，ステート0に移行し，動作を開始します．

ウェーブフォーム・ディスクリプタでは，ステート0～ステート6の7個のステートについて，それぞれGPIFがどのような動作を行うのかを記述します．各ステートごとの記述を「ステート・インストラクション」と呼んでいます．

ステート・インストラクションでは，FD［0：15］を使ったデータ入出力やCTL出力信号の状態設定，RDY入力信号の状態判定と分岐などを指定します．

● ディシジョン・ポイントとノンディシジョン・ポイント

ステート・インストラクションは，ステートを移動する条件によって大きく二つの種類に分かれます．

一つはRDY入力の状態を判断して，次にどのステートに移行するかを決定するもので，「ディシジョン・ポイント」と呼ばれています．BasicのIF文やGOTO文などのように，条件によって分岐したり任意のステート番号に移動するときに利用します．

もう一つは，あらかじめ指定した時間（クロック数）だけ経過したら次のステート（たとえばステート2にいたならステート3）に移動するというもので，「ノンディシジョン・ポイント」と呼ばれます．こちらは，パルスの幅や，アクセス時のセットアップ時間やホールド時間を確保したい場合に利用されます．

図2の例は，接続した機器との間でGPIFモードを使って2線式ハンドシェークを行った例です．

・ステート0（S0）

データをFD［0：15］に出力して，セットアップ時間を待ちます．

・ステート1（S1）

CTL0をLレベルにして，RDY0がHレベルになるのを待ちます．外部機器側では，CTL0がLレベルになったのを見てデータを引き取り，RDY0をHレベルにしてデータ取得の完了を通知します．GPIFは，RDY0がHレベルになったことを検出して，ステート2に移動します．

・ステート2（S2）

CTL0をHレベルに戻すとともにデータ（FD［0：15］）を次のデータに更新して，RDY0がLレベルに戻るのを待ちます．外部機器側は，CTL0がHレベルに戻ったのを確認してRDY0をLレベルに戻します．GPIFは，RDY0がLレベルになったらステート7に移動します．

●ウェーブフォーム・ディスクリプタとステート・インストラクションの構造

表1に示すように，ウェーブフォーム・ディスクリプタは32バイトあり，ステート0からステート6におけるGPIFの動作を定義します．**`EZ_WaveSet()`**で設定するウェーブフォーム・データは，ステート7（アイドル・ステート）に相当する部分は予約済みになっており，データは無効です．

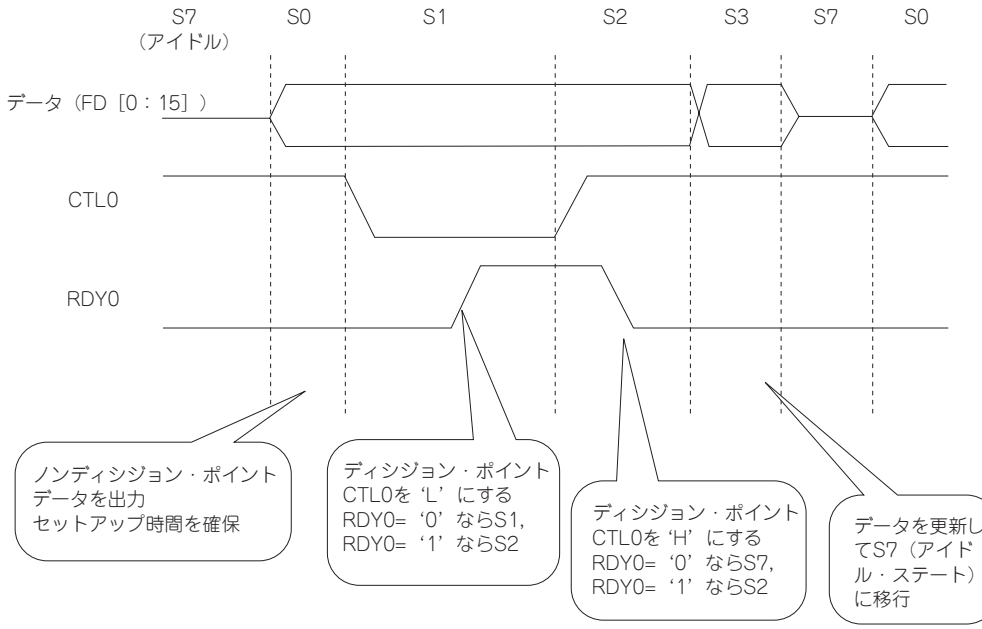

図2　ディシジョン・ポイントとノンディシジョン・ポイント

同一ステートにあたるLENGTH/BRANCH，OPCODE，OUTPUT，LOGIC FUNCTIONの4個のフィールドがセットになって1個のステート・インストラクションになります．たとえば，ステート0用のステート・インストラクションはオフセット＋0，＋8，＋16，＋24の位置にある4バイトのデータで構成されます．

●ステート・インストラクションの区別

ステート・インストラクションのフィールドの使われかたは，ディシジョン・ポイントであるか，ノンディシジョン・ポイントであるかによって異なってきます．

ディシジョン・ポイント用のステート・インストラクションか，ノンディシジョン・ポイント用のステート・インストラクションであるかは，OPCODEフィールドのDPビット（ビット0）で区別されます．

DPビットが‘1’ならばディシジョン・ポイント，‘0’ならばノンディシジョン・ポイントになります．

▶ディシジョン・ポイントのステート・インストラクション

ディシジョン・ポイントのステート・インストラクションの構造は**図3**のようになっています．

OUTPUTフィールドで，このステートにおけるCTL信号（出力）の状態を決定します．

ディシジョン・ポイントでの条件判断は，RDY0～RDY5信号のなかから任意の2本を選び，その間で論理演算を行った結果が‘0’であるか‘1’であるかによって，次のステートを決定するようになっています．

入力信号の選択は，LOGIC FUNCTIONフィールドのTERMA，TERMBで指定し，実行する論理演算はLFUNCで指定します．たとえば，LFUNC = ‘01’，TERMA = ‘001’，TERMB = ‘010’ならば，RDY1と

表1　ウェーブフォーム・ディスクリプタの構造

オフセット	フィールド名	
0	LENGTH/BRANCH	ステート0
1	LENGTH/BRANCH	ステート1
2	LENGTH/BRANCH	ステート2
3	LENGTH/BRANCH	ステート3
4	LENGTH/BRANCH	ステート4
5	LENGTH/BRANCH	ステート5
6	LENGTH/BRANCH	ステート6
7	（予約）	——
8	OPCODE	ステート0
9	OPCODE	ステート1
10	OPCODE	ステート2
11	OPCODE	ステート3
12	OPCODE	ステート4
13	OPCODE	ステート5
14	OPCODE	ステート6
15	（予約）	——

オフセット	フィールド名	
16	OUTPUT	ステート0
17	OUTPUT	ステート1
18	OUTPUT	ステート2
19	OUTPUT	ステート3
20	OUTPUT	ステート4
21	OUTPUT	ステート5
22	OUTPUT	ステート6
23	（予約）	——
24	LOGIC FUNCTION	ステート0
25	LOGIC FUNCTION	ステート1
26	LOGIC FUNCTION	ステート2
27	LOGIC FUNCTION	ステート3
28	LOGIC FUNCTION	ステート4
29	LOGIC FUNCTION	ステート5
30	LOGIC FUNCTION	ステート6
31	（予約）	——

フィールド名	ビット7	ビット6	ビット5	ビット4	ビット3	ビット2	ビット1	ビット0
LENGTH/BRANCH	Re-Execute	X	BRANCH ON 1(演算結果が‘1’の時の飛び先)			BRANCH ON 0(演算結果が‘0’の時の飛び先)		
OPCODE	X	X	SGL	GINT(未使用)	INCAD	NEXT/SGLCRC	DATA	DP(‘1’固定)
OUTPUT	X	X	CTL5	CTL4	CTL3	CTL2	CTL1	CTL0
LOGIC FUNCTION	LFUNC		TERMA			TERMB		

LFUNC　00: A AND B
01: A OR B
10: A XOR B
11: (NOT A) AND B

TERMAおよびTERMB　000: RDY0
001: RDY1
010: RDY2
011: RDY3
100: RDY4
101: RDY5
101: FIFO FLAG(使用不可)
110: INTRDY(使用不可)

図3　ディシジョン・ポイントのステート・インストラクションの構造

RDY2の論理和（OR）になります．入力が1本だけでよい場合には，TERMA，TERMBの両方に同じものを指定し，LFUNCをANDなどにしておけばよいことになります．

論理演算の結果による飛び先は，LENGTH/BRANCHフィールドのBRANCH ON 1，BRANCH ON 0で指定します．論理演算の結果が‘1’のときにはBRANCH ON 1が，‘0’のときにはBRANCH ON 0側が利用されます．飛び先として現在と同じステート値を指定すれば，条件が成立するまでウェイトすることになります．

フィールド名	ビット7	ビット6	ビット5	ビット4	ビット3	ビット2	ビット1	ビット0
LENGTH/BRANCH	このステートに留まるクロック数（0は256クロック），クロックは48MHz							
OPCODE	X	X	SGL	GINT(未使用)	INCAD	NEXT/SGLCRC	DATA	DP(‘0’固定)
OUTPUT	X	X	CTL5	CTL4	CTL3	CTL2	CTL1	CTL0
LOGIC FUNCTION	未使用							

図4　ノンディシジョン・ポイントのステート・インストラクションの構造

表2　図2の動作タイムチャートを指定するウェーブフォーム・ディスクリプタの値

	LENGTH/BRANCH	OPCODE	OUTPUT	LOGIC FUNCTION	動作概要
ステート0	02h	02h	3Fh	00h	データ出力して2クロック待つ
ステート1	11h	03h	3Eh	00h	CTL0=‘0’，RDY0=‘0’ならステート1，‘1’ならステート2へ
ステート2	13h	03h	3Fh	00h	CTL0=‘1’，RDY0=‘0’ならステート3，‘1’ならステート2へ
ステート3	3Fh	07h	3Fh	00h	データ更新してステート7へ
ステート4	00h	00h	00h	00h	未使用
ステート5	00h	00h	00h	00h	未使用
ステート6	00h	00h	00h	00h	未使用
ステート7	00h	00h	00h	00h	未使用

データの入出力は，OPCODEフィールドのDATAビット（ビット1）で指定します．OUT方向（ホスト→ターゲット）のときには，DATAが‘1’になっているときデータ・バス（FD［0：15］）にデータが出力されます．DATAが‘0’になっているとハイ・インピーダンスになります．IN方向の場合には，DATAが‘1’になっているステートでデータの取り込みが行われます．

OUT動作では，OPCODEフィールドのSGLビット（ビット5）が‘0’の場合，NEXT/SGLCRCビット（ビット2）が‘1’になっているとFIFOポインタが進み，次のデータがデータ・バスFD［0：15］に出力されます．IN方向で使っている場合，このビットは無効です．

INCADビットが‘1’になっていると，GPIFADR端子の値をインクリメントします．GPIFADRピンの初期値は，**`EZ_AdrsCTLSet()`**によってCTL端子の初期値とともに指定します．

Re-Execute，SGLは通常は‘0’にしておきます．これらは，FX2デバイスをATAインターフェース用に使ったときにUDMAモードをサポートするためのものです．

▶ノンディシジョン・ポイントのステート・インストラクション

ノンディシジョン・ポイントのステートインストラクションの構成は**図4**のようになっています．

ノンディシジョン・ポイントの場合，LENGTH/BRANCHフィールドが，このステートに留まるクロック数を指定するフィールドになります．ゼロ（0x00）は256クロックの意味になります．EzFirm/FX2では，GPIFのクロックは48 MHzを与えるようにしています．

また，LOGIC FUNCTIONフィールドは，ノンディシジョン・ポイントでは使用されません．

●ウェーブフォーム・ディスクリプタのサンプル

先ほど**図2**に示した動作波形をウェーブフォーム・ディスクリプタにすると**表2**のようになります．

ステート0で確保する時間を仮に2クロックぶんとしてみました．今回，入力はRDY0だけですので，LOGIC FUNCTIONフィールドはRDY0同士のAND条件としていますので，00hばかりになりました．

ステート3は，データの更新とともにステート7へ無条件ジャンプさせています．RDY0同士のANDを取り，‘1’でも‘0’でもステート7へ移行するようにすることで，RDY0の状態に関係なく無条件ジャンプするようにしています．