

sampleSource
visualC++Polling
visualC++RingBuffer
visualBasicPolling

ADIOX-MK II

MULTIFUNCTION-I/O-X2 SERIES
ADIOX2-API
SAMPLE SOURCE
UPDATE 2011-10-28

SAYA INC.

目次

1. はじめに	3
2. VisualC++ ポーリング	5
3. VisualC++ リングバッファ	9
4. VisualBASIC ポーリング	14
5. VisualC# ポーリング/リングバッファ	17

1. はじめに

言語および対応ボード

サンプルソースは VisualC++、VisualBASIC、VisualC# で記述されています。

このサンプルプログラムは以下のボード、即ち全ての MultifunctionI/O-X2 シリーズに対応します。

ADX II 14-80M-PCI EX-BP75
ADX II 14-80M-PCI EX-UP75
ADX II 85-1M-PCI EX
ADX II 85-1M-PCI
ADX II 42-1K-Ethernet-BD5
ADX II 42-1K-Ethernet-ED12
ADX II 42-1K-Ethernet-EA100

ADX II 14-80M-PCI EX-BP50
ADX II 14-80M-PCI EX-UP50
ADX II 85-1M-PCI EX-P
ADX II 85-1M-PCI-P
ADX II 42-1K-Ethernet-ED5
ADX II 42-1K-Ethernet-ED24
ADX II 52-1K-Ethernet

VisualC++ クラス間の分離されたファイルの結合

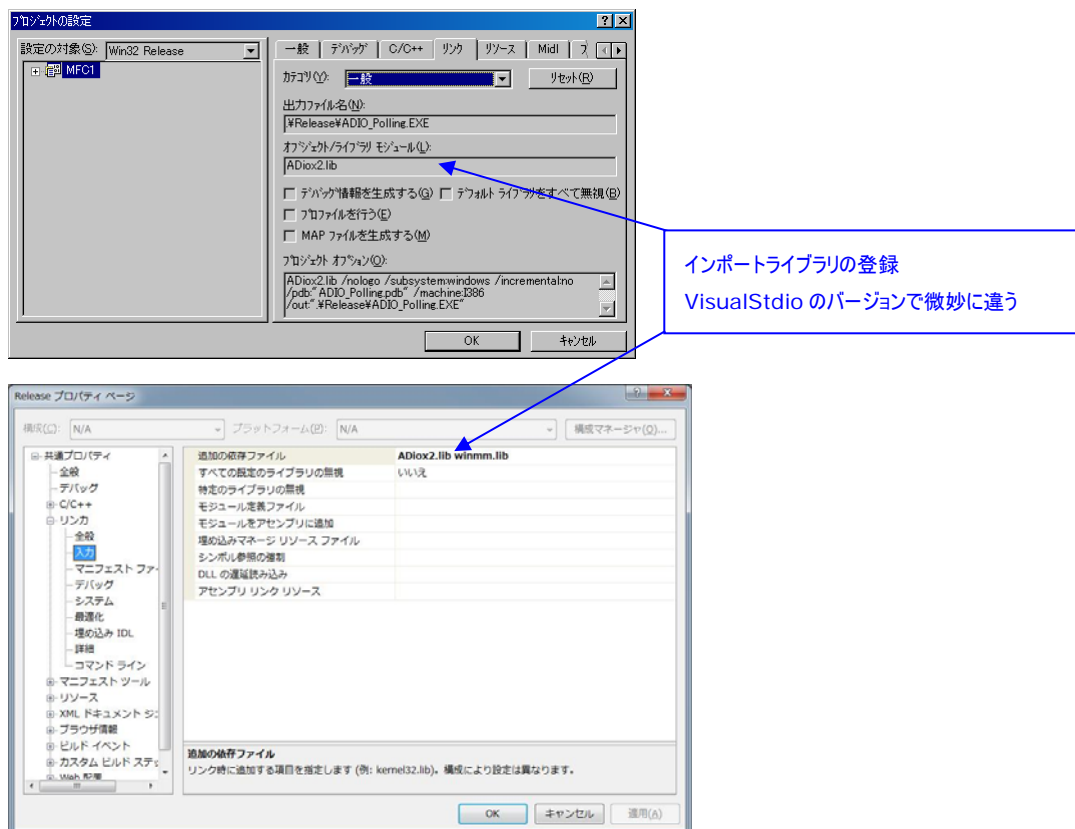
MFC では、クラス毎にクラス定義を記述したヘッダファイルと、本体の cpp ファイルが存在します。小規模のコードでは沢山のファイルの存在がコードの見通しを悪くする可能性があります。そこで、このサンプルソースでは複数のクラスのヘッダファイルと、本体 cpp ファイルを結合、非常に短いコードで記述しています。

VisualC++ インポートライブラリとヘッダファイル

VisualC++ で ADiox2-API を使うにはインポートライブラリを登録する必要があります。インポートライブラリは今回使う API であれば ADiox2.lib だけです。各種インポートライブラリとヘッダファイルはターゲット OS により以下のフォルダにあります。

- ◆ Windows7-64Bit、WindowsVista-64Bit **CDROM\MFIO_X2\sdk\VisualCPP_X64**
- ◆ Windows7-32Bit、WindowsVista-32Bit、WindowsXP-32Bit **CDROM\MFIO_X2\sdk\VisualCPP_X32**

サンプルソースのプロジェクトは既にインポートライブラリが登録されていますが、新規のアプリケーションでインポートライブラリを登録するには、プロジェクトにインポートライブラリを登録し、かつ ADiox2.h をインクルードします。



VisualBASIC

VisualBASIC では **CDROM\MFIO_X2\sdk\VisualBasic\ADIOX-API_VB.bas** をプロジェクトに加えてください。ADiox2-API の定義ファイルです。ちなみに最新の VisualBASIC では、変数に多くの型を用意しており、C/C++/VisualC++/VisualC# と同等に扱うことができますが、VisualBASIC6.0 ぐらいのリビジョンになると型はとても貧弱です。ADiox2-API は VisualBASIC6.0 にあわせているので型に short 型や DWORDLONG などの 16Bit や 64Bit の型を使っていません。

ソフト上のハードウェアの識別

ソフトウェアから見たハードウェアは論理的には 4 種類しかありません。そこで、ソフト上のハードウェアは以下のように命名しています。この名称は、SAYA_DEVICE_INFO.dwDeviceType で取得したハードウェアタイプを識別するためにヘッダファイルにも定義されています。

<ADX14_PCI>

ADX II 14-80M-PCIEX-BP75
ADX II 14-80M-PCIEX-BP50
ADX II 14-80M-PCIEX-UP75
ADX II 14-80M-PCIEX-UP50

<ADX85_PCI>

ADX II 85-1M-PCIEX
ADX II 85-1M-PCIEX-P
ADX II 85-1M-PCI
ADX II 85-1M-PCI-P

<ADX42_LAN>

ADX II 42-1K-Ethernet-BD5
ADX II 42-1K-Ethernet-ED5
ADX II 42-1K-Ethernet-ED12
ADX II 42-1K-Ethernet-ED24
ADX II 42-1K-Ethernet-EA100

<ADX52_LAN>

ADX II 52-1K-Ethernet

このサンプルソースで学べること

ここで紹介するサンプルソースはハードウェアの代表的な機能をほぼ網羅しますが、ADiox2-API の全ての機能を網羅しているわけではありません。ここで紹介した意外にも、ADiox2-API は、リングバッファ出力、プリライトやポストリード、様々なトリガ、ファイル処理、波形生成、高度な画面描画、信号解析、信号調節、設定ファイルを用いたドライバオープン・クローズ処理など、便利な機能がいろいろあります。これらを使いこなすにも、また、サンプルソースを完全に理解するためにも ADiox2-API リファレンスを熟読されることをお勧めします。

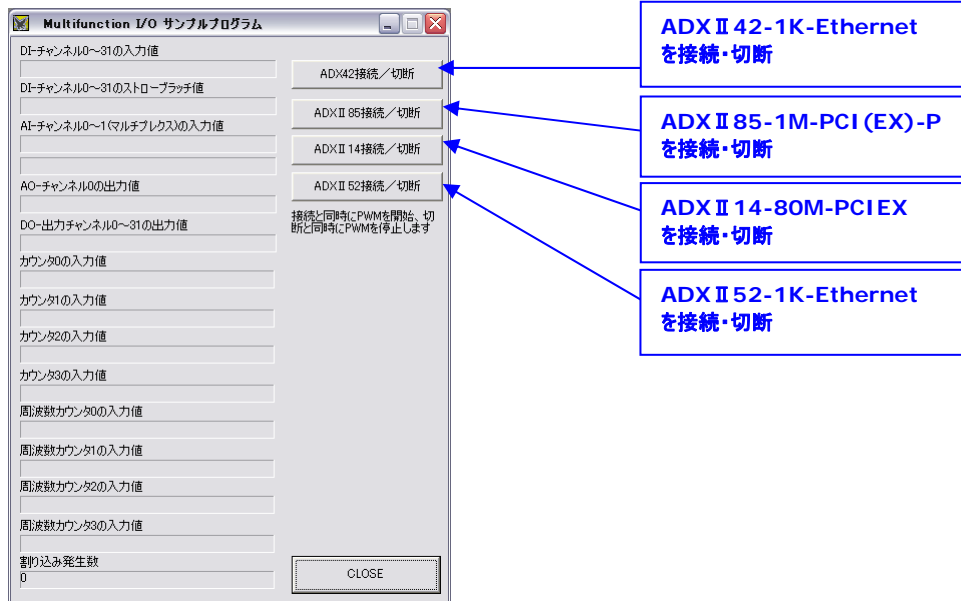
2. VisualC++ + ポーリング

収納場所 **CDROM¥MFIO_X2¥SAMPLE¥PollingInterruptVC6_X86**
CDROM¥MFIO_X2¥SAMPLE¥PollingInterruptVC2005_X86
CDROM¥MFIO_X2¥SAMPLE¥PollingInterruptVC2005_X64

VisualC++ + 6.0/32BIT
 VisualStudio2005/32BIT
 VisualStudio2005/64BIT

目的 リングバッファを除く主要なハードウェア機能(以下)を操作し、タイマーによるポーリングと割り込みメッセージの処理について学習します。
 アナログ入出力のポーリング、 デジタル入出力のポーリング
 エンコーダカウンタのポーリング、 周波数カウンタのポーリング
 PWM 出力の設定、 DI 割り込みの受信

概要 本サンプルソースは、全ての MultifunctionI/O-X2 シリーズに対応するため起動時にハードウェアを初期化せず、接続・切断ボタンをクリックすることで、ハードウェアにアクセスします。



初期化 接続すると、ハードウェアの初期化を行い、PWM を初期化・開始します。(ADX II 42 には実装されないので無視される)また DI に立ち上がりエッジ割り込みと、ストローブラッチを割り当てて、割り込みを許可します。そしてタイマー割り込みをスタートします。

タイマー割り込み アナログ入力、デジタル入力、ラッチされたデジタル入力、エンコーダカウンタ、周波数カウンタをタイマーで読み出し、同時に、アナログ出力に階段状のランプ波形を、デジタル出力に 1Bit シフト波形を出力します。波形出力はあくまでもサンプルなのでシンプルなものになっており振幅やオフセットや周波数、位相などは調節できません。ポーリングの結果は表示されます。

ハードウェア割り込み DI に立ち上がりエッジ割り込みの回数を数えて表示します。

終了処理 切断、アプリケーションの終了でハードウェアの終了処理を行います。

主に使用する ADIOX-API の関数と構造体

関数

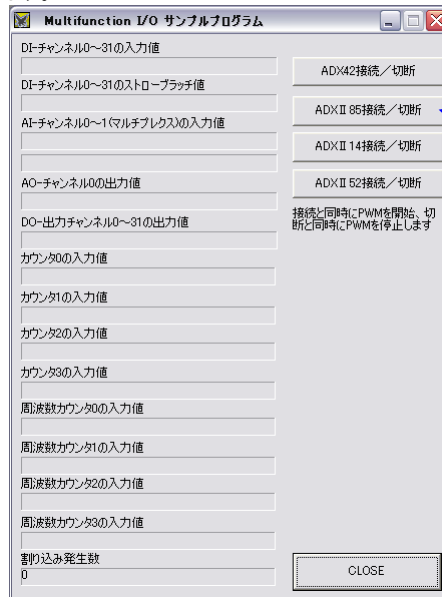
vSetupTcplp	IP アドレス・ポート番号を任意のカード ID に割り付ける(リモート I/O のみ)
bADioxOpen2	ドライバのオープン、ハードウェアの初期化
bADioxDefaultInit	ADiox2-API 主要構造体の安全なデフォルト値を生成し、ハードウェアに設定します
bADioxClose	ドライバのクローズ、ハードウェアの終了処理
bADioxDioMisc2	カウンタ、ストローブラッチ、DI 割り込み、PWM 割付の設定
bADioxConfigPWM2	PWM の位相、発振サイクル数の設定
bADioxSetupPWM2	PWM のデューティの設定
bADioxAnalogConfiguration2	アナログデジタル入出力インターフェース部分の設定
bADioxADIO2	アナログデジタル入出力およびカウンタの一斉ポーリング
bADioxInterruptStart	割り込みの ON/OFF
bADioxInterruptStatus	割り込みステータスの取得

構造体

TDIO_MISC	DIO に割り付けられた機能(DIO インテリジェント機能)設定項目を格納
TConfigPWM	PWM の位相、発振サイクル数の設定項目を格納
TSetupPWM	PWM のデューティの設定項目を格納
IOGEOSUP2	アナログデジタル入出力インターフェースの設定項目を格納
TADIO2	アナログデジタル入出力およびカウンタの一斉ポーリングを行うための構造体
IRQ_BUFFER	割り込みステータスを格納します

ClassWizard で新しく追加した関数

終了処理を記述するため、WM_DESTROY に対する関数 `OnDestroy`、タイマー割り込みを記述するため WM_TIMER に対する関数 `OnTimer` を追加しています。初期化 WM_INITDIALOG に対する関数 `OnInitDialog` も必要です。また操作ボタン IDC_ADX42、IDC_ADX85、IDC_ADX14、IDC_ADX52 に対する関数 `OnAdx42`、`OnAdx85`、`OnAdx14`、`OnAdx52` も追加しています。



IDC_ADX42、IDC_ADX85、IDC_ADX14、IDC_ADX52 に対する関数 `OnAdx42`、`OnAdx85`、`OnAdx14`、`OnAdx52` を追加

マニュアルで追加した関数 1

ハードウェアの起動処理・終了処理をまとめたヘルパ関数、`OnHelper()` を追加しています。

クラス定義の class `CYOKOMFCDIg` : public `CDialog` でプロトタイプ宣言

```
void OnHelper();
```

実体を記述

```
void CYOKOMFCDIg::OnHelper()
{
}
```

マニュアルで追加した関数 2

ハードウェア割り込みのカスタムメッセージに対するコールバック関数 `OnIntrEx()` を追加しています。

クラス定義の class `CYOKOMFCDIg` : public `CDialog` でプロトタイプ宣言

```
LRESULT OnIntrEx ( WPARAM wParam,LPARAM lParam );
```

実体を記述

```
LRESULT CYOKOMFCDIg::OnIntrEx ( WPARAM wParam,LPARAM lParam )
{
    return(0);
}
```

更に関数 `OnIntrEx` を、割り込みメッセージに割り付けます。これはメッセージマップに以下の青字の行を追加します。メッセージマップは通常 ClassWizard で自動的に生成され、イベントメッセージと関数(イベントハンドラ)の割付けを行う部分です。

```
BEGIN_MESSAGE_MAP(CYOKOMFCDIg, CDialog)
   //{{AFX_MSG_MAP(CYOKOMFCDIg)
    ...中省略
    ON_MESSAGE(WM_HWINTR0,OnIntrEx)
   //}}AFX_MSG_MAP
END_MESSAGE_MAP()
```

マニュアルで追加した変数

class `CYOKOMFCDIg` : public `CDialog` クラス定義に以下の変数を追加します。構造体は `ADiox2-API` を使うためです。

```
BYTE    CARD_ID;           // カード ID
HWND    hWnd;             // ウインドハンドル
BOOL    bConnect;        // 接続状態
DWORD   dwIntrrupt;      // 割り込み発生回数
TXBUFSETUP2  sTBUFSETUP;
IOGEOSETUP2  sIOGEOSETUP;
TDIO_MISC  sTDIO_MISC;
TConfigPWM  sTConfigPWM;
TSetupPWM  sTSetupPWM;
```

```
TADIO2          sTADIO;
```

初期化 OnInitDialog

割り込みを使う場合には、必須ですので、ウィンドウハンドルを取得します。

```
hWnd = GetSafeHwnd ();
```

そのほか割り込み発生回数を 0 でリセット、ハードウェア接続フラグを未接続に定義します。

```
dwInterrupt = 0;
bConnect = FALSE;
```

接続切断ボタンのイベントハンドラ

機種を識別する CARD_ID を定義します。**ADX II 42-1K-Ethernet** は CARD_ID は、4~28 ですがここでは 4 を、**ADX II 85-1M-PCIEX/PCI** では 0~3 ですがここでは 0 を、**ADX II 14-80M-PCIEX** では 28 を割り当てます。**ADX II 42-1K-Ethernet** だけはリモート I/O なので vSetupTcpIp で、CADRD_ID に IP アドレスとポート番号を割り当てます。最後に OnHelper() を呼び出して、初期化処理に入ります。

```
void CYOKOMFCDlg::OnAdx42()
{
    CARD_ID = 4;
    vSetupTcpIp ( 192,168,0,2,9004,CARD_ID );
    OnHelper();
}

void CYOKOMFCDlg::OnAdx85()
{
    CARD_ID = 0;
    OnHelper();
}

void CYOKOMFCDlg::OnAdx14()
{
    CARD_ID = 28;
    OnHelper();
}

void CYOKOMFCDlg::OnAdx52()
{
    CARD_ID = 4;
    vSetupTcpIp ( 192,168,1,31,9004,CARD_ID );
    OnHelper();
}
```

OnHelper

まず、bConnect によって処理が分かれます。この変数が FALSE の場合未接続と定義しているので初期化を、TRUE なら接続中と定義しているので終了処理を行います。

```
if ( bConnect == FALSE ) { 初期化処理 }
else { 終了処理 }
```

<初期化処理>

bADioxOpen2~bADioxDefaultInit でドライバをオープンし、デフォルトの設定を行い、デフォルトの構造体群を取得します。次に以下のように、構造体の中で今回使用する箇所だけを設定します。

```
sTDIO_MISC.dwStrobeInterval= 1; // スローブラッチを割り当てる
sTADIO.dwAo0= 0; // AO の初期化
sTADIO.dwDOS= 1; // DO の初期化
sTDIO_MISC.dwDinIntMode16= POSEDGE_INT; // DI を割り込み要因に割り付ける
sTDIO_MISC.dwSetFreq= 5; // PWM 発信周期
sTDIO_MISC.dwLinkPwmToDO= 8; // PWM3 チャンネル ON
sTConfigPWM.dwCycleMax3s= 0; // PWM3 パルス回数=無制限
sTSetupPWM.dwPwm3Value= 2048; // PWM3 デューティ比約 50% (0-4096 の 50%位置)
sTConfigPWM.dwStart= 8; // PWM3(=8)開始コマンド
```

次に結果をハードウェアに反映します。

```
bADioxDioMisc2 ( &sTDIO_MISC,CARD_ID ); // DIO,PWM の設定をハードウェアに反映
bADioxConfigPWM2 ( &sTConfigPWM,CARD_ID ); // パルス発生回数・位相の反映
bADioxSetupPWM2 ( &sTSetupPWM,CARD_ID ); // デューティ比の反映
bADioxInterruptStart ( TRUE,CARD_ID ); // 割り込み許可
```

最後にタイマーを開始、bConnect フラグを接続状態にして終了します。

```
SetTimer ( 1,700,NULL );
bConnect = TRUE;
```

<終了処理>

ドライバを終了できるよう、タイマー割り込み停止します。本来は完全に停止したのを待つべきです。

```
KillTimer ( 1 );
```

PWM3 を停止させます。但し最後に 5 サイクル走らせて止めるようにしています。PWM 機能はハードウェアに実装されているので、この処理をしなければ、PWM 出力はコンピュータの電源を切るまで止まりません。

```
TConfigPWM sTConfigPWM_LOC;
sTConfigPWM_LOC = sTConfigPWM;
sTConfigPWM_LOC.dwCycleMax0s = 5;
sTConfigPWM_LOC.dwCycleMax1s = 5;
sTConfigPWM_LOC.dwCycleMax2s = 5;
sTConfigPWM_LOC.dwCycleMax3s = 5;
bADioxConfigPWM2 ( &sTConfigPWM_LOC,CARD_ID );
sTConfigPWM.dwStart = 0x0;
bADioxSetupPWM2 ( &sTSetupPWM,CARD_ID );
```

最後にドライバをクローズ、bConnect フラグを未接続とします。

```
bADioxClose ( CARD_ID );
bConnect = FALSE;
```

タイマー割り込み OnTimer

タイマー割り込みでは、まず、アナログ出力値、デジタル出力値を以下のように作成します。

```
sTADIO.dwAo0 = ( sTADIO.dwAo0 >= 0xF800 ) ? 0 : sTADIO.dwAo0 + 0x3E8 ;
sTADIO.dwDOS = ( sTADIO.dwDOS == 0x80000000 ) ? 1 : ( sTADIO.dwDOS << 1 );
```

続いて、ポーリング関数を呼び、上記値を出力するとともに、アナログデジタル入力値。カウンタ値などを取得します。

```
sTADIO.dwMode = 0; // 必ず 0 に
bADioxADIO2 ( &sTADIO,CARD_ID );
```

ここからはひたすら取得した値と、出力した値を表示し、最後にアナログ入力チャンネルを切り替えます。チャンネルは 0 と 1 を交互に繰り返します。同時サンプルの ADX II 14-80M-PCIEX はチャンネルを切り替える必要がないのでスキップします。

```
if ( CARD_ID != 28 )
{
    sIOGEOSSETUP.dwMux = ( sIOGEOSSETUP.dwMux == 0 ) ? 1 : 0;
    bADioxAnalogConfigration2 ( &sIOGEOSSETUP,CARD_ID );
}
```

ハードウェア割り込み OnIntrEx

本サンプルでは DI 立ち上がりエッジ割り込みを設定しているので、DI 割り込みが入ると、OnIntrEx が呼び出されます。

OnIntrEx を割り込みメッセージに結びつける方法は、『マニュアルで追加した関数 2』を参照してください。

OnIntrEx では、まず割り込みステータスを取得します。

```
bADioxInterruptStatus ( &sIrbuf,CARD_ID );
```

次にこれが DI 割り込みならば、割り込み発生回数をアップカウントしながら表示します。

```
if ( ( sIrbuf.dwRequestPostmessage & DI_EVENT ) == DI_EVENT )
{
    csText.Format ( "%x", ++dwIntrrupt );
    ::SetDlgItemText ( hWnd,IDC_IRQ,csText );
}
```

終了処理: OnDestroy

本サンプルでは GUI 上のボタンで接続・切断しますので、終了処理でハードウェア処理は必要なさそうですが、このボタンで切断をしないまま終了した時のために、もし接続状態ならば、必要最低限の終了処理を実施するようにしています。

```
if ( bConnect == TRUE )
{
    KillTimer ( 1 );
    bADioxClose ( CARD_ID );
}
```

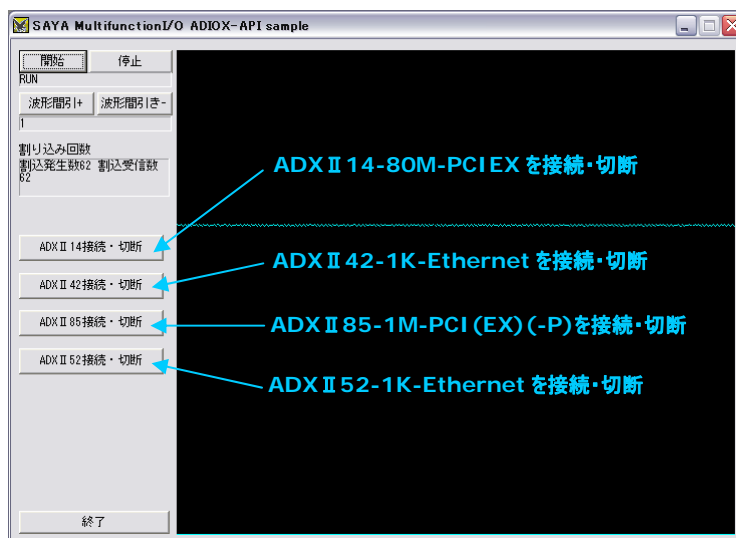
3. VisualC++ + リングバッファ

収納場所 **CDROM\MFIO_X2\SAMPLE\TriggerBufferVC6_X86**
CDROM\MFIO_X2\SAMPLE\TriggerBufferVC2005_X86
CDROM\MFIO_X2\SAMPLE\TriggerBufferVC2005_X64

VisualC++ 6.0/32BIT
 VisualStudio2005/32BIT
 VisualStudio2005/64BIT

目的 リングバッファアクセスによる連続データ収集に必要な最低限を学習します。
 アナログ入力のリングバッファ読み出し
 デジタル入力のリングバッファ読み出し

概要 本サンプルソースは、全ての MultifunctionI/O-X2 シリーズに対応するため起動時にハードウェアを初期化せず、接続・切断ボタンをクリックすることで、ハードウェアにアクセスします。



初期化 接続すると、ハードウェアの初期化を行い、割り込みを許可、トリガを無条件にしてリングバッファを稼働させます。
タイマー割り込み ステータスを取得表示します。
ハードウェア割り込み リングバッファの値を読み出して波形表示を行います。また割り込み発生回数・受信数を表示します。
終了処理 切断、アプリケーションの終了でハードウェアの終了処理を行います。

主に使用する ADIOX-API の関数と構造体

関数

vSetupTcplp	IP アドレス・ポート番号を任意のカード ID に割り付ける(リモート I/O のみ)
bADioxOpen2	ドライバのオープン、ハードウェアの初期化
vADioxDeviceInfo	デバイス情報を取得します
bADioxDefaultInit	ADiox2-API 主要構造体の安全なデフォルト値を生成し、ハードウェアに設定します
bADioxClose	ドライバのクローズ、ハードウェアの終了処理
bADioxSetupSymmetryEngine2	トリガ、リングバッファの設定を行います
bADioxRingBufferMode	セカンダリリングバッファサイズを設定します
bADioxInterruptStart	割り込みの ON/OFF
bADioxInterruptStatus	割り込みステータスの取得
bADioxMessageCount	割り込み発生回数を取得します
bADioxDmaReadEX2	A1,DI リングバッファを読み出します
bADioxStatus2	ステータス(トリガ状況など)を取得します

構造体

TXBUFSETUP2	トリガ、リングバッファの設定を格納します
ADIOX_EXTENTION2	ファイル保存・波形生成の設定を格納します
IRQ_BUFFER	割り込みステータスを格納します
TStatusPack2	ステータスを格納します
SAYA_DEVICE_INFO	デバイス情報を格納します

ClassWizard で新しく追加した関数

終了処理を記述するため、WM_DESTROY に対する関数 `OnDestroy`、タイマー割り込みを記述するため WM_TIMER に対する関数 `OnTimer`、初期化 WM_INITDIALOG に対する関数 `OnInitDialog` を追加します。また操作ボタンクリックに対して以下のように関数を作成しています。

IDC_ADX42 に対する関数 `OnAdx42` (接続・切断)
 IDC_ADX85 に対する関数 `OnAdx85` (接続・切断)
 IDC_ADX14 に対する関数 `OnAdx14` (接続・切断)
 IDC_ADX52 に対する関数 `OnAdx52` (接続・切断)
 IDC_START に対する関数 `OnStart` (リングバッファ開始)
 IDC_STOP に対する関数 `OnStop` (リングバッファ停止)
 IDC_TIMEP に対する関数 `OnTimep` (波形間引き+)
 IDC_TIMEN に対する関数 `OnTimen` (波形間引き-)

マニュアルで追加した関数 1

ハードウェアの起動処理・終了処理をまとめたヘルパ関数、`OnHelper()` を追加しています。

クラス定義の class `CYOKOMFCDlg` : public `CDialog` でプロトタイプ宣言

```
void OnHelper();
```

実体を記述

```
void CYOKOMFCDlg::OnHelper()
{
}
```

マニュアルで追加した関数 2

ハードウェア割り込みのカスタムメッセージに対するコールバック関数 `OnIntrEx()` を追加しています。

クラス定義の class `CYOKOMFCDlg` : public `CDialog` でプロトタイプ宣言

```
LRESULT OnIntrEx ( WPARAM wParam,LPARAM lParam );
```

実体を記述

```
LRESULT CYOKOMFCDlg::OnIntrEx ( WPARAM wParam,LPARAM lParam )
{
    return(0);
}
```

更に関数 `OnIntrEx` を、割り込みメッセージに割り付けます。これはメッセージマップに以下の青字の行を追加します。メッセージマップは通常 ClassWizard で自動的に生成され、イベントメッセージと関数(イベントハンドラ)の割付を行う部分です。

```
BEGIN_MESSAGE_MAP(CYOKOMFCDlg, CDialog)
   //{{AFX_MSG_MAP(CYOKOMFCDlg)
    ... 中省略
    ON_MESSAGE(WM_HWINTRO,OnIntrEx)
   //}}AFX_MSG_MAP
END_MESSAGE_MAP()
```

マニュアルで追加した変数

class `CYOKOMFCDlg` : public `CDialog` クラス定義に以下の変数を追加します。構造体は ADiox2-API を使うためです。

<code>TXBUFSETUP2</code>	<code>sTBUFSETUP;</code>	// ADiox2-API 用構造体
<code>ADIOX_EXTENTION2</code>	<code>sADIOX_EXTENTION2;</code>	// ADiox2-API 用構造体
<code>HWND</code>	<code>hWnd;</code>	// ウィンドウハンドル
<code>DWORD</code>	<code>dwBufInt;</code>	// 割り込み発生回数
<code>DWORD</code>	<code>dwSkip;</code>	// 波形間引き比率
<code>LPDWORD</code>	<code>dwReadBufFa;</code>	// リングバッファデータ取得用配列
<code>int</code>	<code>iStartX;</code>	// 波形書き込み位置
<code>DWORD</code>	<code>dwBufferSize;</code>	// リングバッファサイズ
<code>BYTE</code>	<code>CARD_ID;</code>	// CARD_ID
<code>BOOL</code>	<code>bConnect;</code>	// 接続・切断状態のフラグ
<code>CDC *</code>	<code>cdPpb;</code>	// 波形描画用のデバイスコンテキスト
<code>CRect</code>	<code>pPpbRect;</code>	// 波形描画領域の座標

初期化 OnInitDialog

割り込みを使う場合には、必須ですので、ウィンドウハンドルを取得します。

```
hWnd = GetSafeHwnd ();
```

そのほか波形間引きしない、ハードウェア接続フラグを未接続にするなどの定義してタイマー割り込みを開始させます。

```
bConnect = FALSE;
dwSkip = 1;
...実際にはここで画面の初期化が入る
SetTimer ( 1,700,NULL );
```

接続切断ボタンのイベントハンドラ

機種を識別する CARD_ID を定義します。**ADX II 42-1K-Ethernet**、**ADX II 52-1K-Ethernet** は CARD_ID は、4~28 ですがここでは 4 を、**ADX II 85-1M-PCIEX/PCI** では 0~3 ですがここでは 0 を、**ADX II 14-80M-PCIEX** では 28 を割り当てます。**ADX II 42-1K-Ethernet**、**ADX II 52-1K-Ethernet** はリモート I/O なので vSetupTcpIp で、CADRD_ID に IP アドレスとポート番号を割り当てます。最後に OnHelper() を呼び出して、初期化処理に入ります。

```
void CYOKOMFCDlg::OnAdx42()
{
    CARD_ID = 4;
    vSetupTcpIp ( 192,168,0,2,9004,CARD_ID );
    OnHelper();
}

void CYOKOMFCDlg::OnAdx85()
{
    CARD_ID = 0;
    OnHelper();
}

void CYOKOMFCDlg::OnAdx14()
{
    CARD_ID = 28;
    OnHelper();
}

void CYOKOMFCDlg::OnAdx52()
{
    CARD_ID = 4;
    vSetupTcpIp ( 192,168,1,31,9004,CARD_ID );
    OnHelper();
}
```

OnHelper

まず、bConnect によって処理が分かれます。この変数が FALSE の場合未接続と定義しているので初期化を、TRUE なら接続中と定義しているので終了処理を行います。

```
if ( bConnect == FALSE ) { 初期化処理 }
else { 終了処理 }
```

<初期化処理>

bADioxOpen2~bADioxDefaultInit でドライバをオープンし、デフォルトの設定を行い、デフォルトの構造体群を取得します。また、vADioxDeviceInfo でリングバッファサイズを取得しています。そして以下のようにリングバッファの受信用配列を初期化します。ここでは ADX II 85-1M-PCIEX/PCI にはプライマリオンチップリングバッファの 20 倍のサイズを設定しています。

```
DWORD dwRingbufferSize = ( CARD_ID==0 ) ? 20 : 1;
dwBufferSize = ( sSAYA_DEVICE_INFO.dwBufferSizeOfDWORD * dwRingbufferSize );
dwReadBuffa = new DWORD[dwBufferSize * 2];
```

前記処理と記述位置は前後しますが、以下のように無条件トリガ、ファイル保存なし、AO,DO 側リングバッファ未使用、各ボードで最適なサンプリング速度で初期化します。

```
sTBUFSETUP.dwTrigStartMode= BURST; // 無条件トリガ
sTBUFSETUP.dwTrigStopMode= RESET; // 停止トリガなし
sTBUFSETUP.dwClockScall = (CARD_ID==28) ? 4 : sSAYA_DEVICE_INFO.dwSAMPLE_FAST * 6;
sTBUFSETUP.dwAoHspBufferd= 0; // 出力側はリングバッファ使用しない
sTBUFSETUP.dwDoHspBufferd= 0; // 出力側はリングバッファ使用しない
sADIOX_EXTENTION2.dwADI_style = 0; // ファイル保存なし
sTBUFSETUP.dwClockScall = ( sSAYA_DEVICE_INFO.dwDeviceType == ADX14_PCI ) ? 7
: ( sSAYA_DEVICE_INFO.dwDeviceType == ADX85_PCI ) ? 80
: ( sSAYA_DEVICE_INFO.dwDeviceType == ADX42_LAN ) ? 160000
: ( sSAYA_DEVICE_INFO.dwDeviceType == ADX52_LAN ) ? 10000
: 10000;
bADioxSetupSymmetryEngine2 ( &sTBUFSETUP,&sADIOX_EXTENTION2,CARD_ID );
```

最後に割り込みを許可し、bConnect フラグを接続状態にして終了します。

```
bADioxInterruptStart ( TRUE,CARD_ID );
bConnect = TRUE;
```

<終了処理>

ドライバを終了できるよう、タイマー割り込み停止します。本来は完全に停止したのを待つべきです。

```
KillTimer ( 1 );
```

最後にドライバをクローズ、リングバッファの受信用配列を開放、bConnect フラグを未接続とします。

```
bADioxClose ( CARD_ID );
delete dwReadBufa;
bConnect = FALSE;
```

開始ボタンのイベントハンドラ

まず接続していない場合にはすぐさま終了します。

```
if ( bConnect == FALSE )
    return;
```

次に波形描画位置や、割り込み受信回数を初期化します。

```
iStartX          = pPpbRect.left;
dwBufInt         = 0;
```

最後にリングバッファを開始させます。(トリガ待ちにする⇒無条件なのですぐさま開始となる)

```
sTBUFSETUP.dwInterruptMode = DMA_INT;
bADioxSetupSymmetryEngine2 ( &sTBUFSETUP,&sADIOX_EXTENTION2,CARD_ID );
```

停止ボタンのイベントハンドラ

まず接続していない場合にはすぐさま終了します。

```
if ( bConnect == FALSE )
    return;
```

最後にリングバッファを停止させます。

```
sTBUFSETUP.dwInterruptMode = NOT_INT;
bADioxSetupSymmetryEngine2 ( &sTBUFSETUP,&sADIOX_EXTENTION2,CARD_ID );
```

タイマー割り込み OnTimer

まず接続していない場合にはすぐさま終了します。

```
if ( bConnect == FALSE )
    return;
```

次にステータスを取得し、トリガ状態 sTStatus.dwTrigSeq を表示させます。csTrigSequence はコード先頭で定義された文字列で、sTStatus.dwTrigSeq に対する文字=0 なら"IDLE"、1 なら"RUN"が記載されています。

```
TStatusPack2    sTStatus;
bADioxStatus2 ( &sTStatus,CARD_ID );
::SetDlgItemText ( hWnd,IDC_TRIGGER_STATUS,csTrigSequence[sTStatus.dwTrigSeq]);
```

ハードウェア割り込み OnIntrEx

まず接続していない場合にはすぐさま終了します。

```
if ( bConnect == FALSE )
    return;
```

次に、割り込みステータスと、割り込み発生回数を取得します。

```
IRQ_BUFFER      sIrqbuf;
TStatusPack2    sTStatus;
bADioxInterruptStatus ( &sIrqbuf,CARD_ID );
bADioxMessageCount ( &dwMessageCount,CARD_ID );
...ここで割り込み発生回数と割り込み受信回数を表示
```

次にリングバッファ割り込みか否かを確認し、リングバッファ割り込みであれば、リングバッファデータ取得 (bADioxDmaReadEX2) ~ 波形描画となります。bADioxDmaReadEX2 の戻り値は停止フラグであるので、これを見て停止処理 OnStop() を呼び出していますが、このコードでは停止トリガなどを使っていないので、あくまで参考です。もし出力側リングバッファを使う場合に bADioxWriteMemoryEX を波形描画の前か、bADioxDmaReadEX2 の前に記述します。MultifunctionI/O-X2 シリーズは全機種ともに入力側リングバッファと出力側リングバッファが同期しているので、割り込みメッセージは DMA_SIGNAL か DMA_BUFFER_HALF_EMPTY のいずれか一つしか転送されないのです。従って、入力側リングバッファ割り込みと出力側リングバッファ割り込みは区別されません。

```
if ((( sIrqbuf.dwRequestPostmessage & DMA_SIGNAL ) == DMA_SIGNAL )
    || (( sIrqbuf.dwRequestPostmessage & DMA_BUFFER_HALF_EMPTY ) == DMA_BUFFER_HALF_EMPTY ))
{
    // リングバッファデータの取得
    if ( bADioxDmaReadEX2 ( &dwReadBufa[0],&sTStatus,CARD_ID ) == TRUE )
        OnStop();
    ...ここで波形描画
}
```

終了処理: OnDestroy

本サンプルでは GUI 上のボタンで接続・切断しますので、終了処理でハードウェア処理は必要なさそうですが、このボタンで切断をしないまま終了した時のために、もし接続状態ならば、必要最低限の終了処理を実施するようにしています。

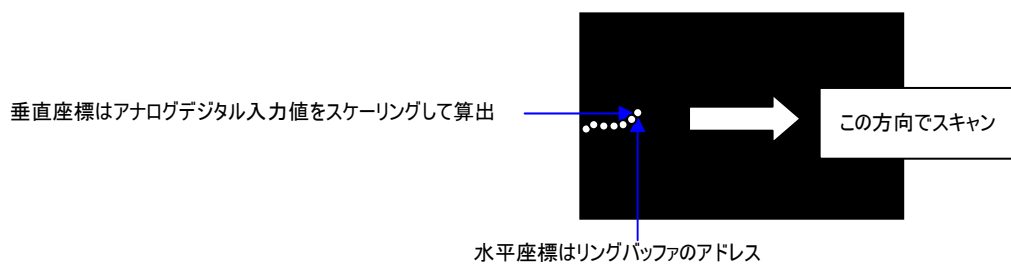
```
if ( bConnect == TRUE )
{
    KillTimer ( 1 );
    bADioxClose ( CARD_ID );
}
```

波形間引きボタンのイベントハンドラ

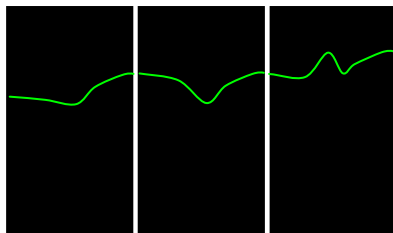
OnTimep, OnTimen では、間引き数 `dwSkip` を変更しています。この結果は、ハードウェア割り込みハンドラ `OnIntrEx` の波形描画処理に反映されます。波形描画はあくまでサンプルなので、単純なものになっています。より本格的な波形描画を実現すると、コードはかなり複雑になりますが、ADiox2-API は波形描画アシスト関数が充実しており、Sドライバ内部で波形イメージを作成することができます。このため、アプリケーションはわずかのコードで高度な波形描画を実現できます。

OnPaint に追加したコード、及び波形描画

このサンプルでは波形描画のために `OnPaint` 関数にもコードを追加しています。このコードは ADiox2-API の使用方法とは直接関係ないので解説は省略します。このサンプルの波形描画は垂直座標は計測データをピクチャボックスにスケールした値 `iAD` (アナログ) で、水平座標はリングバッファのアドレスです。リングバッファのアドレスを `dwSkip` で間引くことで水平方向の縮小が可能です。

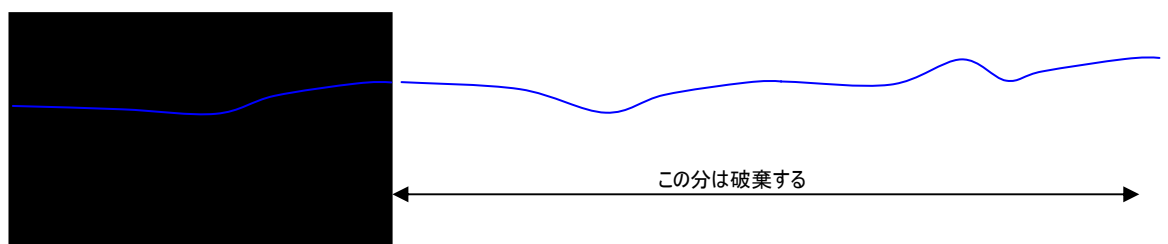


もし `dwSkip` を大きくしたことでピクチャボックスの水平サイズにリングバッファのデータが不足した場合には、以下のように複数のバッファのデータを時間順で水平方向に継ぎ足していきます。



3 つのバッファデータで 1 ピクチャボックスを構成

逆に `dwSkip` を小さくすると、ピクチャボックスの水平サイズよりもバッファのデータが多すぎて描画できなくなってしまいます。描画できなくなったデータをここでは破棄しています。

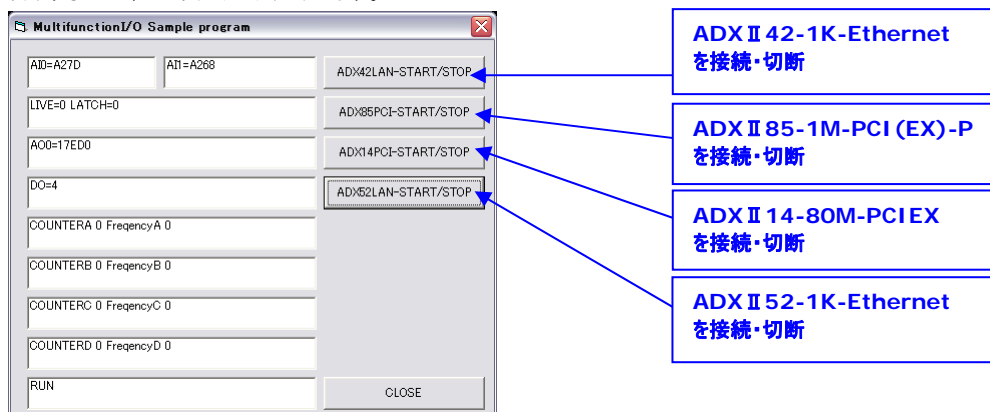


4. VisualBASIC ポーリング

収納場所 **CDROM¥MFIO_X2¥SAMPLE¥PollingInterruptVB6**

目的 内容は VisualC++ 版の PollingInterruptVB6 を VisualBASIC 版に相当します。但し割り込みはありません。割り込みとリングバッファを除く主要なハードウェア機能(以下)を操作し、タイマーによるポーリングについて学習します。
 アナログ入出力のポーリング
 デジタル入出力のポーリング
 エンコーダカウンタのポーリング
 周波数カウンタのポーリング
 PWM 出力の設定
 DI 割り込みの受信

概要 本サンプルソースは、全ての MultifunctionI/O-X2 シリーズに対応するため起動時にハードウェアを初期化せず、接続・切断ボタンをクリックすることで、ハードウェアにアクセスします。



初期化 接続すると、ハードウェアの初期化を行い、PWM を初期化・開始します。(ADX II 42 には実装されないで無視される)またストローブラッチを割り当てます。そしてタイマー割り込みをスタートします。

タイマー割り込み アナログ入力、デジタル入力、ラッチされたデジタル入力、エンコーダカウンタ、周波数カウンタをタイマーで読み出し、同時に、アナログ出力に階段状のランプ波形を、デジタル出力に 1Bit シフト波形を出力します。波形出力はあくまでもサンプルなのでシンプルなものになっており振幅やオフセットや周波数、位相などは調節できません。ポーリングの結果は表示されます。

終了処理 切断、アプリケーションの終了でハードウェアの終了処理を行います。

主に使用する ADIOX-API の関数と構造体

関数

vSetupTcplp	IP アドレス・ポート番号を任意のカード ID に割り付ける(リモート I/O のみ)
bADioxOpen2	ドライバのオープン、ハードウェアの初期化
bADioxDefaultInit	ADiox2-API 主要構造体の安全なデフォルト値を生成し、ハードウェアに設定します
bADioxClose	ドライバのクローズ、ハードウェアの終了処理
bADioxDioMisc2	カウンタ、ストローブラッチ、DI 割り込み、PWM 割付の設定
bADioxConfigPWM2	PWM の位相、発振サイクル数の設定
bADioxSetupPWM2	PWM のデューティーの設定
bADioxAnalogConfiguration2	アナログデジタル入出力インターフェース部分の設定
bADioxADIO2	アナログデジタル入出力およびカウンタの一斉ポーリング

構造体

TDIO_MISC	DIO に割り付けられた機能(DIO インテリジェント機能)設定項目を格納
TConfigPWM	PWM の位相、発振サイクル数の設定項目を格納
TSetupPWM	PWM のデューティーの設定項目を格納
IOGEOSUP2	アナログデジタル入出力インターフェースの設定項目を格納
TADIO2	アナログデジタル入出力およびカウンタの一斉ポーリングを行うための構造体

新しく追加したメソッド(関数)

初期化に対するメソッド	Form_Load
“ADX42LAN-START/STOP” ボタンクリックに対するメソッド	Command2_Click
“ADX85PCI-START/STOP” ボタンクリックに対するメソッド	Command3_Click
“ADX14PCI-START/STOP” ボタンクリックに対するメソッド	Command4_Click
開始、停止のヘルプ関数	start_adx
タイマー割り込み	Timer1_Timer
終了に対するメソッド	Form_Unload

追加したグローバル変数

ADiox2-API の関数で ByRef 型の引数にグローバル変数を直接代入してはいけません。ByRef 型の引数に構造体を代入させる場合にはローカル変数とすべきです。

Dim bConnect	As Integer	接続・未接続を表すフラグ
Dim bCardID	As Byte	CARD_ID を保持する変数
Dim dwAo	As Long	AO 値
Dim dwDO	As Long	DO 値
Dim dwChannel	As Long	AI チャンネル
Dim sgIOGEOSETUP	As IOGEOSETUP2	ADiox2-API を使うためです
Dim sgTADIO	As TADIO2	ADiox2-API を使うためです

初期化 Form_Load

ここではハードウェア接続フラグを未接続に定義するだけです。

```
bConnect = 0
```

接続切断ボタンのイベントハンドラ(Command_Click2_Click、Command3_Click、Command4_Click)

機種を識別する bCardID を定義します。**ADX II 42-1K-Ethernet**、**ADX II 52-1K-Ethernet** は bCardID は、4~28 ですがここでは 4 を、**ADX II 85-1M-PCIEX/PCI** では 0~3 ですがここでは 0 を、**ADX II 14-80M-PCIEX** では 28 を割り当てます。**ADX II 42-1K-Ethernet**、**ADX II 52-1K-Ethernet** はリモート I/O なので vSetupTcpIp で、CADRD_ID に IP アドレスとポート番号を割り当てます。最後に start_adx を呼び出して、初期化処理に入ります。

```
Private Sub Command2_Click()
    bCardID = 4
    Call vSetupTcpIp(192, 168, 0, 2, 9004, bCardID)
    start_adx
End Sub

Private Sub Command3_Click()
    bCardID = 0
    start_adx
End Sub

Private Sub Command4_Click()
    bCardID = 28
    start_adx
End Sub

Private Sub Command5_Click()
    bCardID = 4
    Call vSetupTcpIp(192, 168, 1, 31, 9004, bCardID)
    start_adx
End Sub
```

開始・停止ヘルパ start_adx

まず、bConnect によって処理が分かれます。この変数が 0 場合未接続と定義しているので初期化を、TRUE なら接続中と定義しているので終了処理を行います。

```
If bConnect = 0 Then { 初期化処理 }
Else { 終了処理 }
```

<初期化処理>

bADioxOpen2~bADioxDefaultInit でドライバをオープンし、デフォルトの設定を行い、デフォルトの構造体群を取得します。次に、ここで出てくる ADiox2-API はローカル変数なので、以下の 2 つの構造体をグローバル変数に代入しています。

```
sgIOGEOSETUP = sIOGEOSETUP
sgTADIO = sTADIO
```

次に以下のように、構造体の中で今回使用する箇所だけを設定します。

```
sTDIO_MISC.dwStrobeInternal= 1 // ストローブラッチを割り当てる
sTADIO.dwAo0= 0 // AO の初期化
sTADIO.dwDOS= 1 // DO の初期化
sTDIO_MISC.dwSetFreq= 5 // PWM 発信周期
sTDIO_MISC.dwLinkPwmToDO= 8 // PWM3 チャンネル ON
sTConfigPWM.dwCycleMax3s= 0 // PWM3 パルス回数=無制限
sTSetupPWM.dwPwm3Value= 2048 // PWM3 デューティ比約 50% (0-4096 の 50%位置)
sTConfigPWM.dwStart= 8; // PWM3(=8)開始コマンド
```

また AO、DO、AI チャンネルのグローバル変数を初期化します。

```
dwAo = 0
dwDO = 1
dwChannel = 0
```

最後にこれらをハードウェアに反映し bConnect フラグを接続状態にして終了します。

```
Call bADioxDioMisc2(sTDIO_MISC, bCardID) '発信周期の反映
Call bADioxSetupPWM2(sTSetupPWM, bCardID) 'デューティー比の反映
Call bADioxConfigPWM2(sTConfigPWM, bCardID) 'パルス発生回数・位相の反映
bConnect = 1
```

<終了処理>

bConnect フラグを未接続とします。

```
bConnect = 0
```

PWM3 を停止させます。但し最後に 5 サイクル走らせて止めるようにしています。PWM 機能はハードウェアに実装されているので、この処理をしなければ、PWM 出力はコンピュータの電源を切るまで止まりません。

```
TConfigPWM sTConfigPWM_LOC
sTConfigPWM_LOC = sTConfigPWM
sTConfigPWM_LOC.dwCycleMax0s = 5
sTConfigPWM_LOC.dwCycleMax1s = 5
sTConfigPWM_LOC.dwCycleMax2s = 5
sTConfigPWM_LOC.dwCycleMax3s = 5
Call bADioxConfigPWM2(sTConfigPWM_LOC, bCardID)
sTConfigPWM.dwStart = 0
Call bADioxSetupPWM2(sTSetupPWM, bCardID)
```

最後にドライバをクローズします。

```
bADioxClose (bCardID)
```

タイマー割り込み OnTimer

まず接続していない場合にはすぐさま終了します。

```
If bConnect = 0 Then
Exit Sub
```

次にアナログ出力値、デジタル出力値を以下のように作成します。

```
If dwAo > &HF8000 Then
dwAo = 0
Else: dwAo = dwAo + &H3E8
End If
```

```
If dwDO > 32767 Then
dwDO = 1
Else: dwDO = dwDO * 2
End If
```

```
sTADIO.dwAo0 = dwAo
sTADIO.dwDOS = dwDO
```

続いて、ポーリング関数を呼び、上記値を出力するとともに、アナログデジタル入力値。カウンタ値などを取得します。

```
sTADIO.dwMode = 0 '必ず 0 に
Call bADioxADIO2(sTADIO, bCardID)
```

ここからはひたすら取得した値と、出力した値を表示し、最後に次にアナログ入力チャンネルを切り替えます。チャンネルは 0 と 1 を交互に繰り返します。同時サンプルの ADX II 14-80M-PCIEX はチャンネルを切り替える必要がないのでスキップします。

```
If bCardID <> 28 Then
If dwChannel = 1 Then
dwChannel = 0
Else: dwChannel = 1
End If
```

```
sIOGEOSSETUP.dwMux = dwChannel
Call bADioxAnalogConfiguration2(sIOGEOSSETUP, bCardID)
```

終了処理: OnDestroy

本サンプルでは GUI 上のボタンで接続・切断しますので、終了処理でハードウェア処理は必要なさそうですが、このボタンで切断をしないまま終了した時のために、もし接続状態ならば、必要最低限の終了処理を実施するようにしています。

```
If bConnect = 1 Then
bConnect = 0
Call bADioxClose(bCardID)
End If
```

5. VisualC# ポーリング/リングバッファ

収納場所 CDROM¥MFIO_X2¥SAMPLE¥ MfioX2_SampleCS

- 目的**
- ①タイマーによるポーリングと割り込みメッセージの処理
 - アナログ入出力のポーリング
 - デジタル入出力のポーリング
 - エンコーダカウンタのポーリング
 - 周波数カウンタのポーリング
 - PWM 出力の設定
 - DI 割り込みの受信
 - ②リングバッファアクセスによる連続データ収集
 - アナログ入力のリングバッファ読み出し
 - デジタル入力のリングバッファ読み出し

概要 本サンプルソースは、全ての MultifunctionI/O-X2 シリーズに対応するため起動時にハードウェアを初期化せず、接続・切断ボタンをクリックすることで、ハードウェアにアクセスします。



●接続/切断 初期化

- ①機種別カード ID を設定
- ②ドライバのオープン、ハードウェアの初期化
- ③デフォルト値をドライバ&ハードウェアに反映
- ④構造体配列・2次元配列をもつ構造体 SCP_SETUP メンバの配列の領域確保(C#では必須)
- ⑤割り込み許可

終了処理 ハードウェアの終了処理を行います。

●RingBuffer-Start 初期化

- ①デバイス情報を取得
- ②入力値バッファ確保
- ③各種パラメタを設定(無条件開始トリガ)

タイマー割り込み トリガ状況の確認。

ハードウェア割り込み リングバッファの値を読み出して波形表示を行います。また割り込み発生回数・受信数を表示。

終了処理 タイマー & ハードウェア割り込みの終了処理。

● Poling-Start

初期化

①各種パラメタを設定

- 1: PWM を初期化。(ADX II 4.2 には実装されないで無視)
- 2: DI に立ち上がりエッジ割り込み
- 3: トローブラッチを割り当て

タイマー割り込み

アナログ入力、デジタル入力、ラッチされたデジタル入力、エンコーダカウンタ、周波数カウンタをタイマーで読み出し、同時に、アナログ出力に階段状のランプ波形を、デジタル出力に 1Bit シフト波形を出力します。波形出力はあくまでもサンプルなのでシンプルなものになっており振幅やオフセットや周波数、位相などは調節できません。ポーリングの結果は表示されます。

ハードウェア割り込み

DI に立ち上がりエッジ割り込みの回数を数えて表示

終了処理

タイマー & ハードウェア割り込みの終了処理。

主に使用する ADIOX-API の関数と構造体

メソッド

vSetupTcplp	IP アドレス・ポート番号を任意のカード ID に割り付ける(リモート I/O のみ)
bADioxOpen2	ドライバのオープン、ハードウェアの初期化
vADioxDeviceInfo	デバイス情報を取得します
bADioxDefaultInit	ADiox2-API 主要構造体の安全なデフォルト値を生成し、ハードウェアに設定します
bADioxClose	ドライバのクローズ、ハードウェアの終了処理
bADioxSetupSymmetryEngine2	トリガ、リングバッファの設定を行います
bADioxRingBufferMode	セカンダリリングバッファサイズを設定します
bADioxInterruptStart	割り込みの ON/OFF
bADioxInterruptStatus	割り込みステータスの取得
bADioxMessageCount	割り込み発生回数を取得します
bADioxDmaReadEX2	AI, DI リングバッファを読み出します
bADioxStatus2	ステータス(トリガ状況など)を取得します

構造体

TXBUFSETUP2	トリガ、リングバッファの設定を格納します
ADIOX_EXTENTION2	ファイル保存・波形生成の設定を格納します
IRQ_BUFFER	割り込みステータスを格納します
TStatusPack2	ステータスを格納します
SAYA_DEVICE_INFO	デバイス情報を格納します

AdioxLibrary2.cs のプロジェクトへの追加 (.netFramework2.0 以降対応)

ADiox2-API を使うには、ADiox2-API のインターフェース定義ファイルをインポートする必要があります。また、ADiox2-API はアンセーフコードでないと動作しないため、アンセーフコーディングを許可しておく必要があります。まず以下の要領でこれらの作業を行います。

- ① "sdk\%c#\%d\AdioxLibrary2.cs" ファイルを、プロジェクトのフォルダ内にコピーして、プロジェクトに「既存項目の追加」で追加してください。
- ② 「public unsafe partial class Form1 : Form, IMessageFilter」という、ネームスペースを追加してください。
- ③ プロジェクトのプロパティ(ソリューションエクスプローラのプロジェクト名を右クリック>プロパティ選択)のビルドタブ内の、「アンセーフコードの許可」をチェックします。

フォーム起動・終了時のイベントハンドラの追加

```
void Form1_Load(object sender, EventArgs e)
```

フォームが、はじめて起動する直前に発生するイベントを追加します。

アプリケーション名からウィンドウハンドルをサーチ(Win32 API→AdioxLibrary2.cs でラッピング)します。

```
hWnd = mx.FindWindow(null, this.Text);
```

メッセージフィルタを開始(割り込みメッセージをトラップするため)します。

```
Application.AddMessageFilter(this);
```

```
void Form1_FormClosed(object sender, FormClosedEventArgs e)
```

フォームが、閉じている間に発生するイベントを追加します。

タイマーイベントを停止します。

```
timer1.Stop();
```

MultifunctionI/O-X2 シリーズ停止します。

```
mx.bADioxClose(byCARD_ID);
```

ボタンコントロールとイベントハンドラの追加

void button 接続切断_Click(object sender, EventArgs e)

MultifunctionI/O-X2 シリーズの接続・切断イベントを追加します。

■接続イベント

MultifunctionI/O-X2 シリーズは、3 機種ありますので、「radioButton」で切り替えます。ユーザの「radioButton」入力値にしたがい、ドライバのオープンメソッド-bADioxOpen2 の第 3 引数にカード ID を代入します。実際の機種と、選択番号が違っていた場合など、オープンに失敗した場合は、リターンします。

```
if (mx.bADioxOpen2(hWnd, 0, byCARD_ID) == FALSE)
{
    labelStatus.Text = "I/O オープンエラー";
    return;
}
```

メッセージフィルタを開始(割り込みメッセージをトラップするため)します。

```
Application.AddMessageFilter(this);
```

構造体配列・2次元配列をもつ構造体 SCP_SETUPメンバの配列の領域確保します。C#では必須メソッドです。C#では、2次元配列のサイズを宣言と同時に定義できませんので、このメソッドによる領域確保が必要になります。プログラム上、構造体 SCP_SETUP を使用しない場合は、使用しなくても問題ありません。

```
mx.bADioxArrayInit(ref sSCP_SETUP);
```

ハードウェア割り込みの許可をします。

```
mx.bADioxInterruptStart(TRUE, byCARD_ID);
```

■切断イベント

ドライバのクローズ、ハードウェアの終了処理をします。

```
mx.bADioxClose(byCARD_ID);
```

タイマー停止をします。

```
timer1.Stop();
```

void buttonRingBuffer_Start_Click(object sender, EventArgs e)

リングバッファの設定とハードウェア割り込みの開始をします。(詳細は、3:ヘルパーメソッド参照)

```
vStart_RingBuffer();
```

void buttonRingBuffer_Stop_Click(object sender, EventArgs e)

ハードウェア割り込みの停止をします。(詳細は、3:ヘルパーメソッド参照)

```
vStart_RingBuffer();
```

void buttonPolling_Start_Click(object sender, EventArgs e)

ポーリングの設定とタイマーの開始、ハードウェア割り込み設定をします。(詳細は、3:ヘルパーメソッド参照)

```
vStart_RingBuffer();
```

void buttonPolling_Stop_Click(object sender, EventArgs e)

タイマー&ハードウェア割り込みの停止をします。(詳細は、3:ヘルパーメソッド参照)

```
vStart_RingBuffer();
```

ヘルパーメソッドの追加

void vStart_RingBuffer()

リングバッファの設定とハードウェア割り込みの開始をします。

デバイス情報を取得します。ここでは、使用しているデバイスの、2ステージリングバッファ(プライマリオンチップリングバッファ)またはリングバッファ DWORD(32Bit)単位サイズを取得し、次にデバイスごとのセカンダリリングバッファの設定をします。

```
mx.vADioxDeviceInfo(ref sSAYA_DEVICE_INFO, byCARD_ID);
```

ここでは ADX II 85 にはプライマリオンチップリングバッファの 20 倍のサイズを設定(ADX II 85 のみ可能)しています

```
mx.bADioxRingBufferMode(uiRingbufferSize, byCARD_ID)
```

リングバッファの受信用配列を初期化します。

```
uiBufferSize = (sSAYA_DEVICE_INFO.dwBufferSizeOfDWORD * uiRingbufferSize);
```

```
uiReadBuffa = new uint[uiBufferSize + α];
```

無条件トリガ、ファイル保存なし、AO,DO 側リングバッファ未使用で初期化し、リングバッファを開始します。

```
sTXBUFSETUP.dwClockScall = (byCARD_ID == 28) ? 7 : sSAYA_DEVICE_INFO.dwSAMPLE_FAST * 6;
```

```
sTXBUFSETUP.dwTrigStartMode = mx.BURST; //無条件開始トリガ
```

```
sTXBUFSETUP.dwTrigStopMode = mx.RESET; //停止トリガなし
```

```
sTXBUFSETUP.dwAoHspBufferd = 0; //出力側はリングバッファ使用しない
```

```
sTXBUFSETUP.dwDoHspBufferd = 0; //出力側はリングバッファ使用しない
```

```
sTXBUFSETUP.dwIntrruptMode = mx.DMA_INT; //リングバッファ開始
```

```
mx.bADioxSetupSymmetryEngine2(ref sTXBUFSETUP, ref sADIOX_EXTENTION2, byCARD_ID);
```

タイマーを開始をします。

```
timer1.Start();
```

vStop_RingBuffer()

ハードウェア割り込みの停止をします。

リングバッファ停止をします。

```
sTXBUFSETUP.dwInterruptMode = mx.NOT_INT;
mx.bADioxSetupSymmetryEngine2(ref sTXBUFSETUP, ref sADIOX_EXTENTION2, byCARD_ID);
```

タイマー停止をします。

```
vStart_RingBuffer();
```

void vStart_Polling()

ポーリングの設定とタイマーの開始、ハードウェア割り込み設定をします。

構造体の中で今回使用する箇所だけを設定します。

```
sTDIO_MISC.dwStrobeInternal = 1;           //ストローブラッチを割り当てる
sTADIO.dwAo0 = 0;                         //AOの初期化
sTADIO.dwDOS = 1;                          //DOの初期化
sTDIO_MISC.dwDinIntMode16 = mx.POSEGE_INT; //DI16(ADX II 85)/DIO(ADX II 42)のエッジ割り込み要因を指定。

sTDIO_MISC.dwSetFreq = 5;                  //PWM発信周期
sTDIO_MISC.dwLinkPwmToDo = 8               //PWM3チャンネルON
sTConfigPWM.dwCycleMax3s = 0;              //PWM3パルス回数=無制限
sTSetupPWM.dwPwm3Value = 2048;             //PWM3デューティ比約50%
sTConfigPWM.dwStart = 8;                   //PWM3開始コマンド
sIOGEOSETUP.dwDifferential = 0;            //シングルエンド16チャンネルアナログ入力(ADX II 85専用)
```

前記構造体をハードウェア・ドライバに反映します。

```
mx.bADioxDioMisc2(ref sTDIO_MISC, byCARD_ID); //DIO,PWMの設定をハードウェアに反映
mx.bADioxConfigPWM2(ref sTConfigPWM, byCARD_ID); //パルス発生回数・位相の反映
mx.bADioxSetupPWM2(ref sTSetupPWM, byCARD_ID); //デューティ比の反映
```

タイマーを開始をします。

```
timer1.Start();
```

void vStop_Polling()

タイマー&ハードウェア割り込みの停止をします。

PWM3を停止させます。但し最後に5サイクル走らせて止めるようにしています。PWM機能はハードウェアに実装されているので、この処理をしなれば、PWM出力はコンピュータの電源を切るまで止まりません。

```
mx.TConfigPWM sTConfigPWM_LOC;
sTConfigPWM_LOC = sTConfigPWM;
sTConfigPWM_LOC.dwCycleMax0s = 5;
sTConfigPWM_LOC.dwCycleMax1s = 5;
sTConfigPWM_LOC.dwCycleMax2s = 5;
sTConfigPWM_LOC.dwCycleMax3s = 5;
mx.bADioxConfigPWM2(ref sTConfigPWM_LOC, byCARD_ID);
sTConfigPWM.dwStart = 0x0;
mx.bADioxSetupPWM2(ref sTSetupPWM, byCARD_ID);
```

タイマー停止をします。

```
vStart_RingBuffer();
```

割り込み**bool PreFilterMessage(ref Message msg)**

メッセージフィルタでハードウェア割り込みをキャッチします。

「AdioxLibrary2.cs」ファイルに記述されているメソッドで、ハードウェア割り込みをキャッチします。

```
mx.dnbADioxIrq(ref msg)
```

ハードウェア割り込みのメソッドを記述します。

```
OnIntrEx();
```

void OnIntrEx()

メッセージフィルタでハードウェア割り込みをキャッチします。

ステートマシンを利用して、ハードウェア割り込みが、このメソッドの処理よりも速い場合の、再突入防止をします。

```
if (IntrSEQ == RUN) return;
```

```
IntrSEQ = RUN;
```

```
·
```

```
·
```

```
IntrSEQ = IDLE; //ステートマシン(アイドル状態):メソッド末尾
```

割り込みステータスを確認します。

```
mx.bADioxInterruptStatus(ref sIRQ_BUFFER, byCARD_ID);
```

割り込み発生回数を取得し、メソッド処理の回数との比較を表示します。

```
mx.bADioxMessageCount(ref uiMessageCount, byCARD_ID);
```

```
label 割込回数.Text
= "割込発生数: " + uiMessageCount.ToString() + "%n 割込受信数: " + (++ui バッファ割込回数).ToString();
```

■vStart_RingBuffer()で開始した場合

リングバッファ割り込みか否かを確認し、リングバッファ割り込みであれば、リングバッファデータ取得 (bADioxDmaReadEX2) ~ 波形描画となります。bADioxDmaReadEX2 の戻り値は停止フラグであるので、これを見て停止処理 OnStop() を呼び出していますが、このコードでは停止トリガなどを使っていないので、あくまで参考です。もし出力側リングバッファを使う場合に bADioxWriteMemoryEX を波形描画の前か、bADioxDmaReadEX2 の前に記述します。MultifunctionI/O-X2 シリーズは全機種ともに入力側リングバッファと出力側リングバッファが同期しているので、割り込みメッセージは DMA_SIGNAL か DMA_BUFFER_HALF_EMPTY のいずれか一つしか転送されないのです。従って、入力側リングバッファ割り込みと出力側リングバッファ割り込みは区別されません。

```
if (((sIRQ_BUFFER.dwRequestPostmessage & mx.DMA_SIGNAL) == mx.DMA_SIGNAL)
    ||((sIRQ_BUFFER.dwRequestPostmessage & mx.DMA_BUFFER_HALF_EMPTY)
    == mx.DMA_BUFFER_HALF_EMPTY))
{
    if (mx.bADioxDmaReadEX2(ref uiReadBuffa[0], ref sTStatus, byCARD_ID) == TRUE)
        vStop_RingBuffer();
        ...波形描画&ラベル表示
}
}
```

■vStart_Polling()で開始した場合

DI16 (ADX II 85)/DIO (ADX II 42) のエッジ割り込みをキャッチします。

```
if ((sIRQ_BUFFER.dwRequestPostmessage & mx.DI_EVENT) == mx.DI_EVENT)
{
    ui 割込発生数++;
    label 割込発生数.Text = ui 割込発生数.ToString(); //割り込み発生回数を表示
}
}
```

void timer1_Tick(object sender, EventArgs e)

タイマー割り込みのメソッドを記述します。

■vStart_RingBuffer()で開始した場合

トリガ状況の確認をします。

```
mx.bADioxStatus2(ref sTStatus, byCARD_ID);
```

■vStart_Polling()で開始した場合

アナログ出力にランプ波を生成します。

```
sTADIO.dwAo0 = (sTADIO.dwAo0 >= 0xF800) ? 0 : sTADIO.dwAo0 + 0x3E8;
```

デジタル出力で、順番に 1ch づつ on していきます。

```
if (byCARD_ID == 0) DOnumber = 0x80000000; //ADX85
else if (byCARD_ID == 28) DOnumber = 0x8000; //ADX14
else DOnumber = 0x8; //ADX42
sTADIO.dwDOS = (sTADIO.dwDOS == DOnumber) ? 1 : (sTADIO.dwDOS << 1);
```

アナログ入出力・デジタル入出力・カウンタ入力の一斉ポーリングを指定します。

```
sTADIO.dwMode = 0; //必ず 0 に
mx.bADioxADIO2(ref sTADIO, byCARD_ID);
```

■label 表示

アナログ入力を表示します。

```
if (byCARD_ID == 28) //ADX14
{
    //ADX14 は同時サンプルなので一気に 2CH 取り込める
    labelAI0.Text = "0ch: " + Math.Round(sTADIO.dAi0, 3).ToString()
        + " ( 0x" + sTADIO.dwAi0.ToString("X") + " )";
    labelAI1.Text = "1ch: " + Math.Round(sTADIO.dAi1, 3).ToString()
        + " ( 0x" + sTADIO.dwAi1.ToString("X") + " )";
}
else //ADX85 と ADX42
{
    //ADX85 と ADX42 はマルチプレクス方式なので順次取り込む
    labelAI0.Text = Math.Round(sTADIO.dAi0, 3).ToString()
        + " ( 0x" + sTADIO.dwAi0.ToString("X") + " )";
}
}
```

デジタル入力-チャンネル 0~31 の入力値を表示します。

```
labelDI.Text = sTADIO.dwDI.ToString("X");
```

DI-チャンネル 0~31 のストローバッチ値を表示します。

```
labelDI_Latch.Text = sTADIO.dwDI_Latch.ToString("X");
```

アナログ出力-チャンネル 0 の出力値を表示します。

```
labelAO.Text = sTADIO.dwAo0.ToString("X");
```

デジタル出力-チャンネル 0~31 の出力値を表示します。

```
labelDO.Text = sTADIO.dwDOS.ToString("X");
```

エンコーダ or パルスカウンタチャンネル 0～チャンネル 3 の値を表示します。

```
label カウンタ 0.Text = sTADIO.dwCounterA.ToString("X");
    label カウンタ 1.Text = sTADIO.dwCounterB.ToString("X");
    label カウンタ 2.Text = sTADIO.dwCounterC.ToString("X");
    label カウンタ 3.Text = sTADIO.dwCounterD.ToString("X");
```

周波数カウンタチャンネル 0～チャンネル 3 の値を表示します。

```
label 周波数カウンタ 0.Text = sTADIO.dwFreqLatch_A.ToString("X");
    label 周波数カウンタ 1.Text = sTADIO.dwFreqLatch_B.ToString("X");
    label 周波数カウンタ 2.Text = sTADIO.dwFreqLatch_C.ToString("X");
    label 周波数カウンタ 3.Text = sTADIO.dwFreqLatch_D.ToString("X");
```

ADX II 85 と ADX II 42 のアナログ入力 CH の切り替えを表示します。

```
if (byCARD_ID != 28)
{
    if (uiAICH_keep != (uint)numericUpDownAIO_CH.Value)
    {
        sLOGESETUP.dwMux = (uint)numericUpDownAIO_CH.Value;

        //アナログデジタル入出力インターフェース部の機能設定
        mx.bADioxAnalogConfigration2(ref sLOGESETUP, byCARD_ID);

        //アナログ入力 CH を記憶
        uiAICH_keep = (uint)numericUpDownAIO_CH.Value;
    }
}
```