PIO-DIO

Software Manual

[For Windows 95/98, NT and 2000]

Table of Contents

1. Reference

1.1 USER'S MANUAL

Please refer to the following user manuals:

- **PnPInstall.pdf:**
  Describes how to install the PnP (Plug and Play) driver for PCI card under Windows 95/98/2000.

- **SoftInst.pdf:**
  Describes how to install the software package under Windows 95/98/NT/2000.

- **CallDll.pdf:**
  Describes how to call the DLL functions with VC++5, VB5, Delphi3 and Borland C++ Builder 3.

- **ResCheck.pdf:**
  Describes how to check the resources I/O Port address, IRQ number and DMA number for add-on cards under Windows 95/98/NT/2000.

- **PIO-D144.pdf:**
  PIO-D144 Hardware manual.
- **PIO-D48.pdf:**
  PIO-D48 Hardware manual.
- **PIO-D56.pdf:**
  PIO-D56/PIO-D24 Hardware manual.
- **PIO-D96.pdf:**
  PIO-D96 Hardware manual.

1.2 ERROR CODE

For the most errors, it is recommended to check:

1. Does the device driver installs successful?
2. Does the card have plugged?
3. Does the card conflicts with other device?
4. Close other applications to free the system resources.
5. Try to use another slot to plug the card.
6. Restart your system to try again.

Error Code	Error Description
PIODIO_NoError	OK
PIODIO_DriverOpenError	Device driver can't be opened.
PIODIO_DriverNoOpen	Users have to call the PIODIO_DriverInit() function firstly.
PIODIO_GetDriverVersionError	Can not communicate with device driver.
PIODIO_InstallIrqError	Can't communicate with device driver or the system has no enough system resources for interrupt services.
PIODIO_ClearIntCountError	Can not communicate with device driver.
PIODIO_GetIntCountError	Can not communicate with device driver.
PIODIO_RemoveIrqError	Can not communicate with device driver or the interrupt service routine not installed!
PIODIO_FindBoardError	Check your card.
PIODIO_ExceedBoardNumber	The Max. boards is: 8
PIODIO_ResetError	Can not communicate with device driver.
PIODIO_IrqMaskError	Irq-Mask is 1, 2, 4, 8 or 1 to 0xF.
PIODIO_ActiveModeError	Active-Mode is 1,2 or 1 to 3.
PIODIO_GetActiveFlagError	Can not communicate with device driver.
PIODIO_ActiveFlagEndOfQueue	The flag queue is empty.

2. Declaration Files

```
|--\Driver                ← some device driver
    |--\BCB3              ← for Borland C++ Builder 3
    |    |--\PIODIO.H     ← Header file
    |    |--\PIOD48u.CPP  ← Subroutine for PIO-D48
    |    +--\PIODIO.LIB   ← Linkage library for BCB3 only
    |
    |--\Delphi3           ← for Delphi 3
    |    |--\PIOD48u.PAS  ← Subroutine for PIO-D48
    |    +--\PIODIO.PAS   ← Declaration file
    |
    |--\VB5               ← for Visual Basic 5
    |     |--\PIOD48u.BAS ← Subroutine for PIO-D48
    |    +--\PIODIO.BAS   ← Declaration file
    |
    +--\VC5               ← for Visual C++ 5
        |--\PIODIO.H      ← Header file
        +--\PIODIO.LIB    ← Linkage library for VC5 only
```

2.1 PIODIO.H

```
#ifdef __cplusplus
    #define EXPORTS extern "C" __declspec (dllimport)
#else
    #define EXPORTS
#endif

// return code
#define PIODIO_NoError                  0
#define PIODIO_DriverOpenError          1
#define PIODIO_DriverNoOpen             2
#define PIODIO_GetDriverVersionError    3
#define PIODIO_InstallIrqError          4
#define PIODIO_ClearIntCountError       5
#define PIODIO_GetIntCountError         6
#define PIODIO_RegisterApcError         7
#define PIODIO_RemoveIrqError           8
#define PIODIO_FindBoardError           9
#define PIODIO_ExceedBoardNumber        10
#define PIODIO_ResetError               11
#define PIODIO_IrqMaskError             12
#define PIODIO_ActiveModeError          13
#define PIODIO_GetActiveFlagError       14
#define PIODIO_ActiveFlagEndOfQueue     15

// define the interrupt signal source
#define PIOD144_P2C0      0   // pin29 of CN1(37 pin D-type, pin1 to pin37)
#define PIOD144_P2C1      1   // pin28 of CN1(37 pin D-type, pin1 to pin37)
#define PIOD144_P2C2      2   // pin27 of CN1(37 pin D-type, pin1 to pin37)
#define PIOD144_P2C3      3   // pin26 of CN1(37 pin D-type, pin1 to pin37)

// Interrupt Channel for PIO-D48
#define PIOD48_INTCH0     1   // INT_CHAN_0
#define PIOD48_INTCH1     2   // INT_CHAN_1
#define PIOD48_INTCH2     4   // INT_CHAN_2
#define PIOD48_INTCH3     8   // INT_CHAN_3
// Interrupt ActiveMode for PIOD48_XXX functions
#define PIOD48_ActiveLow   1   // Active When Low
#define PIOD48_ActiveHigh  2   // Active When High

// to trigger a interrupt when high -> low
#define PIODIO_ActiveLow            0
// to trigger a interrupt when low -> high
#define PIODIO_ActiveHigh           1
```

```
// ID
#define PIO_D144                0x800100    // 144 * D/I/O
#define PIO_D96                 0x800110    // 96 * D/I/O
#define PIO_D64                 0x800120    // 64 * D/I/O
#define PIO_D56                 0x800140    // D24 + 16I + 16O
#define PIO_D48                 0x800130    // 48 * D/I/O
#define PIO_D24                 0x800140    // 24 * D/I/O

// Test functions
EXPORTS float       CALLBACK PIODIO_FloatSub(float fA, float fB);
EXPORTS short       CALLBACK PIODIO_ShortSub(short nA, short nB);
EXPORTS WORD        CALLBACK PIODIO_GetDllVersion(void);

// Driver functions
EXPORTS WORD        CALLBACK PIODIO_DriverInit(void);
EXPORTS void        CALLBACK PIODIO_DriverClose(void);
EXPORTS WORD        CALLBACK PIODIO_SearchCard
      (WORD *wBoards, DWORD dwPIOCardID);
EXPORTS WORD        CALLBACK PIODIO_GetDriverVersion
      (WORD *wDriverVersion);
EXPORTS WORD        CALLBACK PIODIO_GetConfigAddressSpace
      (WORD  wBoardNo, DWORD *wAddrBase, WORD *wIrqNo,
       WORD *wSubVendor, WORD *wSubDevice, WORD *wSubAux,
       WORD *wSlotBus,   WORD *wSlotDevice);
EXPORTS WORD    CALLBACK PIODIO_ActiveBoard( WORD wBoardNo );
EXPORTS WORD    CALLBACK PIODIO_WhichBoardActive(void);

// DIO functions
EXPORTS void    CALLBACK PIODIO_OutputWord
      (DWORD wPortAddress, DWORD wOutData);
EXPORTS void    CALLBACK PIODIO_OutputByte
      (DWORD wPortAddr, WORD bOutputValue);
EXPORTS DWORD   CALLBACK PIODIO_InputWord
      (DWORD wPortAddress);
EXPORTS WORD    CALLBACK PIODIO_InputByte(DWORD wPortAddr);

// Interrupt functions
EXPORTS WORD    CALLBACK PIODIO_IntInstall(WORD wBoardNo,
HANDLE *hEvent, WORD wInterruptSource, WORD wActiveMode);
EXPORTS WORD    CALLBACK PIODIO_IntRemove(void);
EXPORTS WORD    CALLBACK PIODIO_IntResetCount(void);
EXPORTS WORD    CALLBACK PIODIO_IntGetCount(DWORD *dwIntCount);
```

```
// PIOD48 Counter functions
EXPORTS void   CALLBACK PIOD48_SetCounter
      (DWORD dwBase, WORD wCounterNo, WORD bCounterMode,
DWORD wCounterValue);
EXPORTS DWORD  CALLBACK PIOD48_ReadCounter
      (DWORD dwBase, WORD wCounterNo, WORD bCounterMode);
EXPORTS void   CALLBACK PIOD48_SetCounterA
      (WORD wCounterNo, WORD bCounterMode, DWORD wCounterValue);
EXPORTS DWORD  CALLBACK PIOD48_ReadCounterA(WORD
wCounterNo, WORD bCounterMode);

// PIOD48 Interrupt functions
EXPORTS WORD   CALLBACK PIOD48_IntInstall
      (WORD wBoardNo, HANDLE *hEvent, WORD wIrqMask, WORD
wActiveMode);
EXPORTS WORD   CALLBACK PIOD48_IntRemove();
EXPORTS WORD   CALLBACK PIOD48_IntGetActiveFlag
      (WORD *bActiveHighFlag, WORD *bActiveLowFlag);
EXPORTS WORD   CALLBACK PIOD48_IntGetCount(DWORD *dwIntCount);

// PIOD64 Counter functions
EXPORTS void   CALLBACK PIOD64_SetCounter
      (DWORD dwBase, WORD wCounterNo, WORD bCounterMode,
DWORD wCounterValue);
EXPORTS DWORD  CALLBACK PIOD64_ReadCounter
      (DWORD dwBase, WORD wCounterNo, WORD bCounterMode);
EXPORTS void   CALLBACK PIOD64_SetCounterA
      (WORD wCounterNo, WORD bCounterMode, DWORD wCounterValue);
EXPORTS DWORD  CALLBACK PIOD64_ReadCounterA(WORD
wCounterNo, WORD bCounterMode);

// PIOD48 Frequency Measurement functions
EXPORTS DWORD  CALLBACK PIOD48_Freq(DWORD dwBase);
EXPORTS DWORD  CALLBACK PIOD48_FreqA();
```

2.2 PIOD48u.CPP

```
// ****************************************************************
// Initialize the INT2(COut0) Interrupt to High
// this will uses the Counter0 to trigger the interrupt.
//
// wAddrBase       : The base address of PIO-D48,
//                   please refer to function 'PIODIO_GetConfigAddressSpace()'.
// wClockIntConfig : The "Clock/Int Control Register" configuration code,
//                   refer to section "Read/Write Clock/Int Control Register"
//                   in the hardware's manaul.
// wCounter0Config : The configuration code of Counter0
// wCounter0Value  : 0 to &hFFFF, the value is used to set the Counter0
//                   Only the low WORD (16-bits) is valid.
// ****************************************************************
void PIOD48u_INT2InitialHigh(DWORD wAddrBase, WORD wClockIntConfig,
    WORD wCounter0Config, DWORD wCounter0Value)
{

  PIODIO_OutputByte(wAddrBase+0xf0, wClockIntConfig);

  //--- program the trigger freq as P2C0 div wCounter0Value   ---
  //--- For example: if freq of P2C0 is 100Hz, then the       ---
  //--- Freq for COut0 as P2C0/wCounter0Value                 ---
  wCounter0Config = (WORD)( (wCounter0Config >> 1) & 0x07 );  // counter
mode ?
  PIOD48_SetCounter(wAddrBase, 0, wCounter0Config, wCounter0Value); //
Counter 0

}
```

```
// ****************************************************************
// Initialize the INT3(COut2) Interrupt to High,
// this will uses the Counter1 and Counter2 to trigger the interrupt.
//
// wAddrBase       : The base address of PIO-D48,
//                   please refer to function 'PIODIO_GetConfigAddressSpace()'.
// wClockIntConfig : The "Clock/Int Control Register" configuration code,
//                   refer to section "Read/Write Clock/Int Control Register"
//                   in the hardware's manaul.
// wCounter1Config : The configuration code of Counter1
// wCounter1Value  : 0 to &hFFFF, the value is used to set the Counter1
//                   Only the low WORD (16-bits) is valid.
// wCounter2Config : The configuration code of Counter2
// wCounter2Value  : 0 to &hFFFF, the value is used to set the Counter2
//                   Only the low WORD (16-bits) is valid.
// ****************************************************************
void PIOD48u_INT3InitialHigh(DWORD wAddrBase, WORD wClockIntConfig,
    WORD wCounter1Config, DWORD wCounter1Value,
    WORD wCounter2Config, DWORD wCounter2Value )
{

  PIODIO_OutputByte(wAddrBase+0xf0, wClockIntConfig);

  // Cout2 as ?hz/( wCounter1Value * wCounter2Value)
  wCounter1Config =(WORD)( (wCounter1Config >> 1) & 0x07 );   // counter
mode
  wCounter2Config =(WORD)( (wCounter2Config >> 1) & 0x07 );   // counter
mode
  PIOD48_SetCounter(wAddrBase, 1, wCounter1Config, wCounter1Value); //
Counter 1
  PIOD48_SetCounter(wAddrBase, 2, wCounter2Config, wCounter2Value); //
Counter 2

  // wait for Cout2 to high
  for( ;  ;  )
  {
    if( (PIODIO_InputByte(wAddrBase+0x07)&0x08) != 0 )
        break;
  }
}
```

2.3 PIODIO.BAS

```
Attribute VB_Name = "PIODIO"

Global Const PIODIO_NoError                 = 0
Global Const PIODIO_DriverOpenError         = 1
Global Const PIODIO_DriverNoOpen            = 2
Global Const PIODIO_GetDriverVersionError   = 3
Global Const PIODIO_InstallIrqError         = 4
Global Const PIODIO_ClearIntCountError      = 5
Global Const PIODIO_GetIntCountError        = 6
Global Const PIODIO_RegisterApcError        = 7
Global Const PIODIO_RemoveIrqError          = 8
Global Const PIODIO_FindBoardError          = 9
Global Const PIODIO_ExceedBoardNumber       = 10
Global Const PIODIO_ResetError              = 11
Global Const PIODIO_IrqMaskError            = 12
Global Const PIODIO_ActiveModeError         = 13
Global Const PIODIO_GetActiveFlagError      = 14
Global Const PIODIO_ActiveFlagEndOfQueue    = 15

' define the interrupt signal source
Global Const PIOD144_P2C0 = 0 ' pin29 of CN1(37 pin D-type, pin1 to pin37)
Global Const PIOD144_P2C1 = 1 ' pin28 of CN1(37 pin D-type, pin1 to pin37)
Global Const PIOD144_P2C2 = 2 ' pin27 of CN1(37 pin D-type, pin1 to pin37)
Global Const PIOD144_P2C3 = 3 ' pin26 of CN1(37 pin D-type, pin1 to pin37)

' Interrupt Channel for PIO-D48
Global Const PIOD48_INTCH0 = 1   ' INT_CHAN_0
Global Const PIOD48_INTCH1 = 2   ' INT_CHAN_1
Global Const PIOD48_INTCH2 = 4   ' INT_CHAN_2
Global Const PIOD48_INTCH3 = 8   ' INT_CHAN_3
' Interrupt ActiveMode for PIOD48_XXX functions
Global Const PIOD48_ActiveLow = 1     ' Active When Low
Global Const PIOD48_ActiveHigh = 2    ' Active When High

' to trigger a interrupt when high -> low
Global Const PIODIO_ActiveLow = 0
' to trigger a interrupt when low -> high
Global Const PIODIO_ActiveHigh = 1
```

```
' ID
Global Const PIO_D144 = &H800100                    ' 144 * D/I/O
Global Const PIO_D96 = &H800110                     ' 96 * D/I/O
Global Const PIO_D64 = &H800120                     ' 64 * D/I/O
Global Const PIO_D56 = &H800140                     ' D24 + 16I + 16O
Global Const PIO_D48 = &H800130                     ' 48 * D/I/O
Global Const PIO_D24 = &H800140                     ' 24 * D/I/O


' The Test functions
Declare Function PIODIO_ShortSub Lib "PIODIO.dll" (ByVal a As Integer,
ByVal b As Integer) As Integer
Declare Function PIODIO_FloatSub Lib "PIODIO.dll" (ByVal a As Single,
ByVal b As Single) As Single
Declare Function PIODIO_GetDllVersion Lib "PIODIO.dll" () As Integer

' The Driver functions
Declare Function PIODIO_DriverInit Lib "PIODIO.dll" () As Integer
Declare Sub PIODIO_DriverClose Lib "PIODIO.dll" ()
Declare Function PIODIO_SearchCard Lib "PIODIO.dll" (wBoards As Integer,
ByVal dwPIOPISOCardID As Long) As Integer
Declare Function PIODIO_GetDriverVersion Lib "PIODIO.dll" (wDriverVersion
As Integer) As Integer
Declare Function PIODIO_GetConfigAddressSpace Lib "PIODIO.dll" ( _
    ByVal wBoardNo As Integer, wAddrBase As Long, wIrqNo As Integer, _
    wSubVendor As Integer, wSubDevice As Integer, wSubAux As Integer, _
    wSlotBus As Integer, wSlotDevice As Integer) As Integer

Declare Function PIODIO_ActiveBoard Lib "PIODIO.dll" (ByVal wBoardNo As
Integer) As Integer
Declare Function PIODIO_WhichBoardActive Lib "PIODIO.dll" () As Integer


' DIO functions
Declare Sub PIODIO_OutputByte Lib "PIODIO.dll" (ByVal address As Long,
ByVal dataout As Integer)
Declare Sub PIODIO_OutputWord Lib "PIODIO.dll" (ByVal address As Long,
ByVal dataout As Long)
Declare Function PIODIO_InputByte Lib "PIODIO.dll" (ByVal address As
Long) As Integer
Declare Function PIODIO_InputWord Lib "PIODIO.dll" (ByVal address As
Long) As Long

' Interrupt functions
Declare Function PIODIO_IntInstall Lib "PIODIO.dll" (ByVal wBoard As
Integer, hEvent As Long, _
      ByVal wInterruptSource As Integer, ByVal wActiveMode As Integer) As
Integer
Declare Function PIODIO_IntRemove Lib "PIODIO.dll" () As Integer
```

```
Declare Function PIODIO_IntGetCount Lib "PIODIO.dll" (dwIntCount As Long)
As Integer
Declare Function PIODIO_IntResetCount Lib "PIODIO.dll" () As Integer

' PIOD48 Counter functions
Declare Sub PIOD48_SetCounter Lib "PIODIO.dll" _
      (ByVal dwBase As Long, ByVal wCounterNo As Integer, _
       ByVal bCounterMode As Integer, ByVal wCounterValue As Long)
Declare Function PIOD48_ReadCounter Lib "PIODIO.dll" _
      (ByVal dwBase As Long, ByVal wCounterNo As Integer, _
       ByVal bCounterMode As Integer) As Long
Declare Sub PIOD48_SetCounterA Lib "PIODIO.dll" _
      (ByVal wCounterNo As Integer, ByVal bCounterMode As Integer, _
       ByVal wCounterValue As Long)
Declare Function PIOD48_ReadCounterA Lib "PIODIO.dll" _
      (ByVal wCounterNo As Integer, ByVal bCounterMode As Integer) As
Long

' PIOD48 Interrupt functions
Declare Function PIOD48_IntInstall Lib "PIODIO.dll" _
      (ByVal wBoardNo As Integer, hEvent As Long, _
       ByVal wIrqMask As Integer, ByVal wActiveMode As Integer) As Integer
Declare Function PIOD48_IntRemove Lib "PIODIO.dll" () As Integer
Declare Function PIOD48_IntGetActiveFlag Lib "PIODIO.dll" _
      (bActiveHighFlag As Integer, bActiveLowFlag As Integer) As Integer
Declare Function PIOD48_IntGetCount Lib "PIODIO.dll" _
      (dwIntCount As Long) As Integer

' PIOD64 Counter functions
Declare Sub PIOD64_SetCounter Lib "PIODIO.dll" _
      (ByVal dwBase As Long, ByVal wCounterNo As Integer, _
       ByVal bCounterMode As Integer, ByVal wCounterValue As Long)
Declare Function PIOD64_ReadCounter Lib "PIODIO.dll" _
      (ByVal dwBase As Long, ByVal wCounterNo As Integer, _
       ByVal bCounterMode As Integer) As Long
Declare Sub PIOD64_SetCounterA Lib "PIODIO.dll" _
      (ByVal wCounterNo As Integer, ByVal bCounterMode As Integer, _
       ByVal wCounterValue As Long)
Declare Function PIOD64_ReadCounterA Lib "PIODIO.dll" _
      (ByVal wCounterNo As Integer, ByVal bCounterMode As Integer) As
Long

' PIOD48 Frequency Measurement Functons
Declare Function PIOD48_Freq  Lib "PIODIO.dll" _
      (ByVal dwBase As Long) As Long
Declare Function PIOD48_FreqA Lib "PIODIO.dll" () As Long
```

2.4 PIOD48u.BAS

```
Attribute VB_Name = "PIOD48u"
Option Explicit


'// ***************************************************************
'// Initialize the INT2(COut0) Interrupt to High
'// this will uses the Counter0 to trigger the interrupt.
'//
'// wAddrBase       : The base address of PIO-D48,
'//                   please refer to function 'PIODIO_GetConfigAddressSpace()'.
'// wClockIntConfig : The "Clock/Int Control Register" configuration code,
'//                   refer to section "Read/Write Clock/Int Control Register"
'//                   in the hardware's manaul.
'// wCounter0Config : The configuration code of Counter0
'// wCounter0Value  : 0 to &hFFFF, the value is used to set the Counter0
'//                   Only the low WORD (16-bits) is valid.
'// ***************************************************************
Sub PIOD48u_INT2InitialHigh(ByVal wAddrBase As Long, ByVal
wClockIntConfig As Integer, _
    ByVal wCounter0Config As Integer, ByVal wCounter0Value As Long)

    PIODIO_OutputByte (wAddrBase + &HF0), wClockIntConfig

    '//--- program the trigger freq as P2C0 div wCounter0Value   ---
    '//--- For example: if freq of P2C0 is 100Hz, then the       ---
    '//--- Freq for COut0 as P2C0/wCounter0Value                  ---
    wCounter0Config = (wCounter0Config \ 2) And &H7       ' Counter mode
    PIOD48_SetCounter wAddrBase, 0, wCounter0Config, wCounter0Value
'Counter 0

End Sub
```

```
'// ***************************************************************
'// Initialize the INT3(COut2) Interrupt to High,
'// this will uses the Counter1 and Counter2 to trigger the interrupt.
'//
'// wAddrBase       : The base address of PIO-D48,
'//                   please refer to function 'PIODIO_GetConfigAddressSpace()'.
'// wClockIntConfig : The "Clock/Int Control Register" configuration code,
'//                   refer to section "Read/Write Clock/Int Control Register"
'//                   in the hardware's manaul.
'// wCounter1Config : The configuration code of Counter1
'// wCounter1Value  : 0 to &hFFFF, the value is used to set the Counter1
'//                   Only the low WORD (16-bits) is valid.
'// wCounter2Config : The configuration code of Counter2
'// wCounter2Value  : 0 to &hFFFF, the value is used to set the Counter2
'//                   Only the low WORD (16-bits) is valid.
'// ***************************************************************
Sub PIOD48u_INT3InitialHigh(ByVal wAddrBase As Long, ByVal
wClockIntConfig As Integer, _
        ByVal wCounter1Config As Integer, ByVal wCounter1Value As Long, _
        ByVal wCounter2Config As Integer, ByVal wCounter2Value As Long)

  PIODIO_OutputByte (wAddrBase + &HF0), wClockIntConfig

  '// Cout2 as ?hz/( wCounter1Value * wCounter2Value)
  wCounter1Config = (wCounter1Config \ 2) And &H7       ' Counter mode
  wCounter2Config = (wCounter2Config \ 2) And &H7       ' Counter mode
  PIOD48_SetCounter wAddrBase, 1, wCounter1Config, wCounter1Value
'Counter 1
  PIOD48_SetCounter wAddrBase, 2, wCounter2Config, wCounter2Value
'Counter 2

  '// wait for Cout2 to high
  While (True)
    If ((PIODIO_InputByte(wAddrBase + &H7) And &H8) <> 0) Then
      Exit Sub
    End If
  Wend
End Sub
```

2.5 PIODIO.PAS

```
unit  PIODIO;         { PIODIO.dll interface unit }

interface

const
  PIODIO_NoError                =0;
  PIODIO_DriverOpenError        =1;
  PIODIO_DriverNoOpen           =2;
  PIODIO_GetDriverVersionError  =3;
  PIODIO_InstallIrqError        =4;
  PIODIO_ClearIntCountError     =5;
  PIODIO_GetIntCountError       =6;
  PIODIO_RegisterApcError       =7;
  PIODIO_RemoveIrqError         =8;
  PIODIO_FindBoardError         =9;
  PIODIO_ExceedBoardNumber      =10;
  PIODIO_ResetError             =11;
  PIODIO_IrqMaskError           =12;
  PIODIO_ActiveModeError        =13;
  PIODIO_GetActiveFlagError     =14;
  PIODIO_ActiveFlagEndOfQueue   =15;

  // define the interrupt signal source
  PIOD144_P2C0      =0;   // pin29 of CN1(37 pin D-type, pin1 to pin37)
  PIOD144_P2C1      =1;   // pin28 of CN1(37 pin D-type, pin1 to pin37)
  PIOD144_P2C2      =2;   // pin27 of CN1(37 pin D-type, pin1 to pin37)
  PIOD144_P2C3      =3;   // pin26 of CN1(37 pin D-type, pin1 to pin37)

  // Interrupt Channel for PIO-D48
  PIOD48_INTCH0      =1;   // INT_CHAN_0
  PIOD48_INTCH1      =2;   // INT_CHAN_1
  PIOD48_INTCH2      =4;   // INT_CHAN_2
  PIOD48_INTCH3      =8;   // INT_CHAN_3
  // Interrupt ActiveMode for PIOD48_XXX functions
  PIOD48_ActiveLow    =1;   // Active When Low
  PIOD48_ActiveHigh   =2;   // Active When High

  // to trigger a interrupt when high -> low
  PIODIO_ActiveLow              =0;
  // to trigger a interrupt when low -> high
  PIODIO_ActiveHigh             =1;
```

```
  // ID
  PIO_D144                = $800100;    // 144 * D/I/O
  PIO_D96                 = $800110;    // 96 * D/I/O
  PIO_D64                 = $800120;    // 64 * D/I/O
  PIO_D56                 = $800140;    // D24 + 16I + 16O
  PIO_D48                 = $800130;    // 48 * D/I/O
  PIO_D24                 = $800140;    // 24 * D/I/O


// Test functions
function  PIODIO_ShortSub(nA : smallint; nB : smallint)     : smallint; StdCall;
function  PIODIO_FloatSub(fA :   single; fB : single)       :   single; StdCall;
function  PIODIO_GetDllVersion                              :     word; StdCall;

// Driver functions
function  PIODIO_DriverInit                                 :     word; StdCall;
procedure PIODIO_DriverClose                                        ; StdCall;
function  PIODIO_SearchCard(var wBoards:WORD;
dwPIOPISOCardID:LongInt):WORD; StdCall;
function  PIODIO_GetDriverVersion(var wDriverVer: word)  :  word; StdCall;
function  PIODIO_GetConfigAddressSpace(wBoardNo:word;
      var wAddrBase:LongInt;  var wIrqNo:word;
      var wSubVerdor:word; var wSubDevice:word;    var wSubAux:word;
      var wSlotBus:word;   var wSlotDevice:word ): word; StdCall;
function  PIODIO_ActiveBoard(wBoardNo:Word)       :     WORD; StdCall;
function  PIODIO_WhichBoardActive                           :     WORD; StdCall;

// DIO functions
procedure PIODIO_OutputByte(wPortAddr : LongInt; bOutputVal : Word);
StdCall;
procedure PIODIO_OutputWord(wPortAddr : LongInt; wOutputVal : LongInt);
StdCall;
function  PIODIO_InputByte(wPortAddr  : LongInt )          : word;        StdCall;
function  PIODIO_InputWord(wPortAddr  : LongInt )   : LongInt;      StdCall;

// Interrupt functions
function  PIODIO_IntInstall(wBoard:Word; var hEvent:LongInt;
wInterruptSource:Word; wActiveMode:Word):Word; StdCall;
function  PIODIO_IntRemove                          : WORD; StdCall;
function  PIODIO_IntGetCount(var dwIntCount:LongInt)       : WORD; StdCall;
function  PIODIO_IntResetCount                              : WORD; StdCall;
```

```
// PIOD48 Counter functions
procedure PIOD48_SetCounter( dwBase:LongInt;    wCounterNo:WORD;
                    bCounterMode:WORD; wCounterValue:LongInt); StdCall;
function PIOD48_ReadCounter( dwBase:LongInt;    wCounterNo:WORD;
                    bCounterMode:WORD):LongInt; StdCall;
procedure PIOD48_SetCounterA(wCounterNo:WORD;
                    bCounterMode:WORD; wCounterValue:LongInt); StdCall;
function PIOD48_ReadCounterA(wCounterNo:WORD;
bCounterMode:WORD):LongInt; StdCall;


// PIOD48 Interrupt functions
function PIOD48_IntInstall( wBoardNo:WORD; var hEvent:LongInt;
wIrqMask:WORD;
                   wActiveMode:WORD) : WORD; StdCall;
function PIOD48_IntRemove : WORD; StdCall;
function PIOD48_IntGetActiveFlag( var bActiveHighFlag:WORD; var
bActiveLowFlag:WORD) : WORD; StdCall;
function PIOD48_IntGetCount(      var dwIntCount:LongInt) : WORD; StdCall;

// PIOD64 Counter functions
procedure PIOD64_SetCounter( dwBase:LongInt;    wCounterNo:WORD;
                    bCounterMode:WORD; wCounterValue:LongInt); StdCall;
function PIOD64_ReadCounter( dwBase:LongInt;    wCounterNo:WORD;
                    bCounterMode:WORD):LongInt; StdCall;
procedure PIOD64_SetCounterA(wCounterNo:WORD;
                    bCounterMode:WORD; wCounterValue:LongInt); StdCall;
function PIOD64_ReadCounterA(wCounterNo:WORD;
bCounterMode:WORD):LongInt; StdCall;

// PIOD48 Frequency Measurement functions
function  PIOD48_Freq(dwBase:LongInt):LongInt; StdCall;
function  PIOD48_FreqA:LongInt; StdCall;

implementation

// Test functions
function  PIODIO_ShortSub;              external 'PIODIO.DLL' name
'PIODIO_ShortSub';
function  PIODIO_FloatSub;              external 'PIODIO.DLL' name
'PIODIO_FloatSub';
function  PIODIO_GetDllVersion;         external 'PIODIO.DLL' name
'PIODIO_GetDllVersion';
```

```
// Driver functions
function  PIODIO_DriverInit;            external 'PIODIO.DLL' name
'PIODIO_DriverInit';
procedure PIODIO_DriverClose;           external 'PIODIO.DLL' name
'PIODIO_DriverClose';
function  PIODIO_SearchCard;        external 'PIODIO.DLL' name
'PIODIO_SearchCard';
function  PIODIO_GetDriverVersion;      external 'PIODIO.DLL' name
'PIODIO_GetDriverVersion';
function  PIODIO_GetConfigAddressSpace;  external 'PIODIO.DLL' name
'PIODIO_GetConfigAddressSpace';

function  PIODIO_ActiveBoard;               external 'PIODIO.DLL' name
'PIODIO_ActiveBoard';
function  PIODIO_WhichBoardActive;     external 'PIODIO.DLL' name
'PIODIO_WhichBoardActive';

// DIO functions
procedure PIODIO_OutputByte;            external 'PIODIO.DLL' name
'PIODIO_OutputByte';
procedure PIODIO_OutputWord;            external 'PIODIO.DLL' name
'PIODIO_OutputWord';
function  PIODIO_InputByte;             external 'PIODIO.DLL' name
'PIODIO_InputByte';
function  PIODIO_InputWord;             external 'PIODIO.DLL' name
'PIODIO_InputWord';

// Interrupt functions
function  PIODIO_IntInstall;            external 'PIODIO.DLL' name
'PIODIO_IntInstall';
function  PIODIO_IntRemove;             external 'PIODIO.DLL' name
'PIODIO_IntRemove';
function  PIODIO_IntGetCount;           external 'PIODIO.DLL' name
'PIODIO_IntGetCount';
function  PIODIO_IntResetCount;         external 'PIODIO.DLL' name
'PIODIO_IntResetCount';

// PIOD48 Counter functions
procedure PIOD48_SetCounter;            external 'PIODIO.DLL' name
'PIOD48_SetCounter';
function  PIOD48_ReadCounter;           external 'PIODIO.DLL' name
'PIOD48_ReadCounter';
procedure PIOD48_SetCounterA;           external 'PIODIO.DLL' name
'PIOD48_SetCounterA';
function  PIOD48_ReadCounterA;          external 'PIODIO.DLL' name
'PIOD48_ReadCounterA';

// PIOD48 Interrupt functions
function PIOD48_IntInstall;             external 'PIODIO.DLL' name
```

```
'PIOD48_IntInstall';
function PIOD48_IntRemove;              external 'PIODIO.DLL' name
'PIOD48_IntRemove';
function PIOD48_IntGetActiveFlag;       external 'PIODIO.DLL' name
'PIOD48_IntGetActiveFlag';
function PIOD48_IntGetCount;            external 'PIODIO.DLL' name
'PIOD48_IntGetCount';

// PIOD64 Counter functions
procedure PIOD64_SetCounter;           external 'PIODIO.DLL' name
'PIOD64_SetCounter';
function  PIOD64_ReadCounter;          external 'PIODIO.DLL' name
'PIOD64_ReadCounter';
procedure PIOD64_SetCounterA;          external 'PIODIO.DLL' name
'PIOD64_SetCounterA';
function  PIOD64_ReadCounterA;         external 'PIODIO.DLL' name
'PIOD64_ReadCounterA';

// PIOD48 Frequency Measurement functions
function  PIOD48_Freq;                  external 'PIODIO.DLL' name
'PIOD48_Freq';
function  PIOD48_FreqA;                 external 'PIODIO.DLL' name
'PIOD48_FreqA';

end.
```

2.6 PIOD48u.PAS

```
unit  PIOD48u;

interface

procedure PIOD48u_INT2InitialHigh(wAddrBase:LongInt;
wClockIntConfig:WORD;
        wCounter0Config:WORD; wCounter0Value:LongInt);
procedure PIOD48u_INT3InitialHigh(wAddrBase:LongInt;
wClockIntConfig:WORD;
        wCounter1Config:WORD; wCounter1Value:LongInt;
        wCounter2Config:WORD; wCounter2Value:LongInt);

implementation

uses PIODIO;



// ****************************************************************
// Initialize the INT2(COut0) Interrupt to High
// this will uses the Counter0 to trigger the interrupt.
//
// wAddrBase       : The base address of PIO-D48,
//                   please refer to function 'PIODIO_GetConfigAddressSpace()'.
// wClockIntConfig : The "Clock/Int Control Register" configuration code,
//                   refer to section "Read/Write Clock/Int Control Register"
//                   in the hardware's manaul.
// wCounter0Config : The configuration code of Counter0
// wCounter0Value  : 0 to &hFFFF, the value is used to set the Counter0
//                   Only the low WORD (16-bits) is valid.
// ****************************************************************
procedure PIOD48u_INT2InitialHigh(wAddrBase:LongInt;
wClockIntConfig:WORD;
        wCounter0Config:WORD; wCounter0Value:LongInt);
begin
  PIODIO_OutputByte(wAddrBase+$f0, wClockIntConfig);

  //--- program the trigger freq as P2C0 div wCounter0Value   ---
  //--- For example: if freq of P2C0 is 100Hz, then the       ---
  //--- Freq for COut0 as P2C0/wCounter0Value                 ---
  wCounter0Config := ( wCounter0Config shr 1 ) and $7; // Counter mode
  PIOD48_SetCounter(wAddrBase, 0, wCounter0Config, wCounter0Value); //
Counter 0
end;
```

```
// ****************************************************************
// Initialize the INT3(COut2) Interrupt to High,
// this will uses the Counter1 and Counter2 to trigger the interrupt.
//
// wAddrBase       : The base address of PIO-D48,
//                   please refer to function 'PIODIO_GetConfigAddressSpace()'.
// wClockIntConfig : The "Clock/Int Control Register" configuration code,
//                   refer to section "Read/Write Clock/Int Control Register"
//                   in the hardware's manaul.
// wCounter1Config : The configuration code of Counter1
// wCounter1Value  : 0 to &hFFFF, the value is used to set the Counter1
//                   Only the low WORD (16-bits) is valid.
// wCounter2Config : The configuration code of Counter2
// wCounter2Value  : 0 to &hFFFF, the value is used to set the Counter2
//                   Only the low WORD (16-bits) is valid.
// ****************************************************************
procedure PIOD48u_INT3InitialHigh(wAddrBase:LongInt;
wClockIntConfig:WORD;
        wCounter1Config:WORD; wCounter1Value:LongInt;
        wCounter2Config:WORD; wCounter2Value:LongInt);
begin

  PIODIO_OutputByte(wAddrBase+$f0, BYTE(wClockIntConfig) );

  // Cout2 as ?hz/( wCounter1Value * wCounter2Value)
  wCounter1Config := ( wCounter1Config shr 1 ) and $7; // Counter mode
  wCounter2Config := ( wCounter2Config shr 1 ) and $7; // Counter mode
  PIOD48_SetCounter(wAddrBase, 1, wCounter1Config, wCounter1Value); //
Counter 1
  PIOD48_SetCounter(wAddrBase, 2, wCounter2Config, wCounter2Value); //
Counter 2

  // wait for Cout2 to high
  while ( true ) do
  begin
    if( (PIODIO_InputByte(wAddrBase+$07) and $08) <> 0 ) then
       exit;
  end;
end;


end.
```

3. Function Descriptions

In this chapter, we use some keywords to indicate the attribute of Parameters.

Keyword	Setting parameter by user before calling this function ?	Get the data/value from this parameter after calling this function ?
[Input]	Yes	No
[Output]	No	Yes
[Input, Output]	Yes	Yes

Note: All of the parameters need to be allocated spaces by the user.

3.1 FUNCTIONS OF TEST

3.1.1 PIODIO_GetDllVersion

- **Description:**
  To get the version number of PIODIO.DLL

- **Syntax:**
  WORD PIODIO_GetDllVersion(Void)

- **Parameter:**
  None

- **Return:**
  200(hex) for version 2.00

3.1.2 PIODIO_ShortSub

- **Description:**
  To perform the subtraction as nA - nB in short data type. This function is provided for testing DLL linkage purpose.

- **Syntax:**
  short PIODIO_ShortSub(short nA, short nB)

- **Parameter:**
  nA :[Input] 2 bytes short data type value
  nB :[Input] 2 bytes short data type value

- **Return:**
  The value of nA - nB

3.1.3 PIODIO_FloatSub

- **Description:**
  To perform the subtraction as fA - fB in float data type. This function is provided for testing DLL linkage purpose.

- **Syntax:**
  float PIODIO_FloatSub(float fA, float fB)

- **Parameter:**
  fA : [Input] 4 bytes floating point value
  fB : [Input] 4 bytes floating point value

- **Return:**
  The value of fA - fB

3.2 FUNCTIONS OF I/O

3.2.1 PIODIO_OutputByte

- **Description :**
  This subroutine will send the 8 bits data to the desired I/O port.

- **Syntax :**
  void PIODIO_OutputByte(DWORD wPortAddr, WORD bOutputVal);

- **Parameter :**
  wPortAddr : [Input] I/O port addresses, please refer to function PIODIO_GetConfigAddressSpace.
  Only the low WORD is valid.
  bOutputVal : [Input] 8 bit data send to I/O port.
  Only the low BYTE is valid.

- **Return:**
  None

3.2.2 PIODIO_InputByte

- **Description :**
  This subroutine will input the 8 bit data from the desired I/O port.

- **Syntax :**
  WORD PIODIO_InputByte(DWORD wPortAddr);

- **Parameter :**
  wPortAddr : [Input] I/O port addresses, please refer to function
  PIODIO_GetConfigAddressSpace().
  Only the low WORD is valid.

- **Return:**
  16 bits data with the leading 8 bits are all 0.
  (Only the low BYTE is valid.)

3.2.3 PIODIO_OutputWord

- **Description :**
  This subroutine will send the 16 bits data to the desired I/O port.

- **Syntax :**
  void PIODIO_OutputWord(DWORD wPortAddr, DWORD wOutputVal);

- **Parameter :**
  wPortAddr : [Input] I/O port addresses, please refer to function PIODIO_GetConfigAddressSpace().
  Only the low WORD is valid.
  wOutputVal : [Input] 16 bit data send to I/O port.
  Only the low WORD is valid.

- **Return:**
  None

3.2.4 PIODIO_InputWord

- **Description :**
  This subroutine will input the 16 bit data from the desired I/O port.

- **Syntax :**
  DWORD PIODIO_InputWord(DWORD wPortAddr);

- **Parameter :**
  wPortAddr : [Input] I/O port addresses, please refer to function
  PIODIO_GetConfigAddressSpace().
  Only the low WORD is valid.

- **Return:**
  16 bit data. Only the low WORD is valid.

3.3 FUNCTIONS OF DRIVER

3.3.1 PIODIO_GetDriverVersion

- **Description :**
  This subroutine will read the version number of PIODIO driver.

- **Syntax :**
  WORD PIODIO_GetDriverVersion(WORD *wDriverVersion);

- **Parameter :**
  wDriverVersion : [Output] address of wDriverVersion

- **Return:**
  Please refer to "Section 1.2 Error Code".

3.3.2 PIODIO_DriverInit

- **Description :**
  This subroutine will open the PIODIO driver and allocate the resource for the device. This function must be called once before calling other PIODIO functions.

- **Syntax :**
  WORD PIODIO_DriverInit();

- **Parameter :**
  None

- **Return:**
  Please refer to "Section 1.2 Error Code".

3.3.3 PIODIO_DriverClose

- **Description :**
  This subroutine will close the PIODIO Driver and release the resource from the device. This function must be called once before exit the user's application.

- **Syntax :**
  void PIODIO_DriverClose();

- **Parameter :**
  None

- **Return:**
  None

3.3.4 PIODIO_GetConfigAddressSpace

- **Description :**
  Get the I/O address of PIODIO board n.

- **Syntax :**

```
WORD PIODIO_GetConfigAddressSpace
        ( WORD wBoardNo,   DWORD *wAddrBase,     WORD *wIrqNo,
        WORD *wSubVendor, WORD *wSubDevice,     WORD *wSubAux,
        WORD *wSlotBus,     WORD *wSlotDevice);
```

- **Parameter :**

```
wBoardNo     : [Input]  PIODIO board number
wAddrBase    : [Output]  The base address of PIODIO board.
               Only the low WORD is valid.
wIrqNo       : [Output]  The IRQ number that the PIODIO board using.
wSubVendor   : [Output]  Sub Vendor ID.
wSubDevice   : [Output]  Sub Device ID.
wSubAux      : [Output]  Sub Aux ID.
wSlotBus     : [Output]  Slot Bus number.
wSlotDevice  : [Output]  Slot Device ID.
```

- **Return:**
  Please refer to "Section 1.2 Error Code".

3.4 INTERRUPT FUNCTION

3.4.1 PIODIO_IntResetCount

- **Description:**
  This function will clear the counter value on the device driver for the interrupt.

- **Syntax:**
  WORD PIODIO_IntResetCount(void);

- **Parameter:**
  None

- **Return:**
  Please refer to "Section 1.2 Error Code".

3.4.2 PIODIO_IntGetCount

- **Description:**
  This subroutine will read the d**wIntCount** defined in device driver.

- **Syntax :**
  WORD PIODIO_IntGetCount(DWORD *dwIntCount);

- **Parameter:**
  dwIntCount : [Output] Address of dwIntCount, which will stores the counter value of interrupt.

- **Return:**
  Please refer to "Section 1.2 Error Code".

3.4.3 PIODIO_IntInstall

- **Description:**
  This subroutine will install the IRQ service routine.

- **Syntax:**
  WORD PIODIO_IntInstall(WORD wBoardNo, HANDLE *hEvent,
  WORD wInterruptSource, WORD wActiveMode);

- **Parameter:**
  wBoardNo : [Input] Which board to be used.
  hEvent : [Input] Address of a Event handle. The user's program must call the Windows API function "CreateEvent()" to create the event-object.
  wInterruptSource : [Input] What the Interrupt-Source to be used ? Please refer to hardware's manual for the detail information.

Card No.	wInterruptSource	Description
PIO-D48	0	PC3/PC7 from Port-2
	1	PC3/PC7 from Port-5
	2	Cout0
	3	Cout2
PIO-D56 PIO-D24	0	PC0
	1	PC1
	2	PC2
	3	PC3
PIO-D64	0	EXTIRQ
	1	EVTIRQ
	2	TMRIRQ
PIO-D96	0	P2C0
	1	P5C0
	2	P8C0
	3	P11C0
PIO-D144	0	P2C0
	1	P2C1
	2	P2C2
	3	P2C3

  wActiveMode : [Input] When to trigger the interrupt ?
  This can be PIODIO_ActiveHigh or PIODIO_ActiveLow.

- **Return:**
  Please refer to "Section 1.2 Error Code".

3.4.4 PIODIO_IntRemove

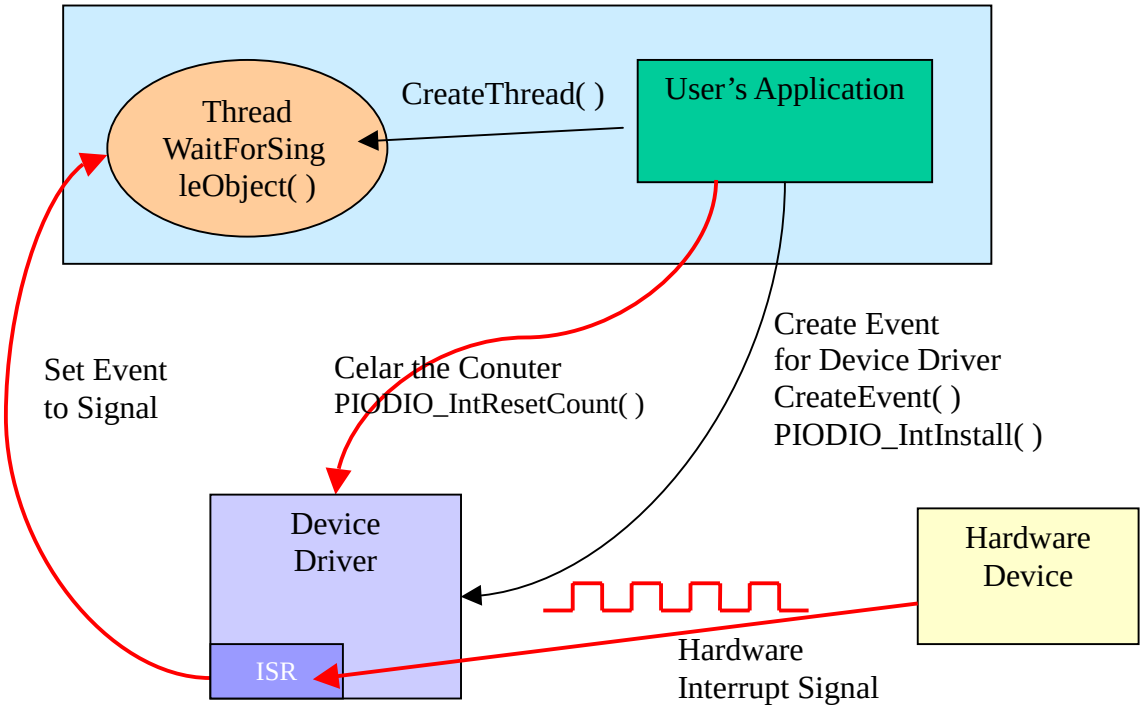
- **Description:**
  This subroutine will remove the IRQ service routine.

- **Syntax:**
  WORD PIODIO_IntRemove(void);

- **Parameter:**
  None

- **Return:**
  Please refer to "Section 1.2 Error Code".

3.4.5 Architecture of Interrupt mode



Please refer to the following Windows API functions:

The following portion description of these functions was copied from MSDN. For the detailed and completely information, please refer to MSDN.

CreateEvent( )

The CreateEvent function creates or opens a named or unnamed event object.

```
HANDLE CreateEvent(
    // pointer to security attributes
    LPSECURITY_ATTRIBUTES lpEventAttributes,
    BOOL bManualReset,     // flag for manual-reset event
    BOOL bInitialState,    // flag for initial state
    LPCTSTR lpName         // pointer to event-object name
);
```

CreateThread( )

The CreateThread function creates a thread to execute within the virtual address space of the calling process.

To create a thread that runs in the virtual address space of another process, use the CreateRemoteThread function.

```
HANDLE CreateThread(
    // pointer to security attributes
    LPSECURITY_ATTRIBUTES lpThreadAttributes,
    DWORD dwStackSize,              // initial thread stack size
     // pointer to thread function
    LPTHREAD_START_ROUTINE lpStartAddress,
    LPVOID lpParameter,             // argument for new thread
    DWORD dwCreationFlags,          // creation flags
    LPDWORD lpThreadId              // pointer to receive thread ID
);
```

WaitForSingleObject( )

The WaitForSingleObject function returns when one of the following occurs:

- The specified object is in the signaled state.
- The time-out interval elapses.

To enter an alertable wait state, use the WaitForSingleObjectEx function. To wait for multiple objects, use the WaitForMultipleObjects.

```
DWORD WaitForSingleObject(
        HANDLE hHandle,          // handle to object to wait for
        DWORD dwMilliseconds   // time-out interval in
    milliseconds
);
```

3.5 PIO-D48 INTERRUPT

The following PIOD48_XXX series function is designed for PIO-D48 card only. They cannot be used with other cards.

The most different between the PIO-DIO and PIO-D48 interrupt functions is the PIO-DIO supports only one interrupt-source at a time and the PIO-D48 supports 4 interrupt-source at a time.

3.5.1 PIOD48_IntInstall

- **Description:**
  This subroutine will install the IRQ service routine. This function supports multiple interrupt-source and the Active-Mode can setting to "Active-Low only", "Active-High only" and "Active-Low or Active-High".
- **Syntax:**
  WORD PIOD48_IntInstall(WORD wBoardNo, HANDLE *hEvent,
  WORD wIrqMask, WORD wActiveMode);
- **Parameter:**
  wBoardNo : [Input] Which board to be used.
  hEvent : [Input] Address of a Event handle. The user's program must call the Windows API function "CreateEvent()" to create the event-object.
  wIrqMask : [Input] What the Interrupt-Source to be used ? Please refer to hardware's manual for the detail information.

wIrqMask	Description
1	INT_CHAN_0: PC3/PC7 from Port-2
2	INT_CHAN_1: PC3/PC7 from Port-5
4	INT_CHAN_2: Cout0
8	INT_CHAN_3: Cout2

This function supports 4 interrupt-source at a time, thus users can use multiple interrupt-source like 1 +2 +4 +8.

wActiveMode : [Input] When the ISR will service the interrupt ?

wActiveMode	Description
1	PIOD48_ActiveLow The interrupt is occurred when the Interrupt-Source status is low.
2	PIOD48_ActiveHigh The interrupt is occurred when the Interrupt-Source status is high.

This can be 1 (Active-High), 2(Active-Low) or 1 + 2 (Both of the High and Low will active the interrupt).

- **Return:**
  Please refer to "Section 1.2 Error Code".

3.5.2 PIOD48_IntRemove

- **Description:**
  This subroutine will remove the IRQ service routine.

- **Syntax:**
  WORD PIOD48_IntRemove(void);

- **Parameter:**
  None

- **Return:**
  Please refer to "Section 1.2 Error Code".

3.5.3 PIOD48_IntGetCount

- **Description:**
  This subroutine will read the **Interrupt-Counter** value in the device driver. The Interrupt-Counter will be increased (in the ISR) when the interrupt is triggered. When the interrupt setting to Active-High only or Active-Low only, some of the interrupt signal will be ignored and the Interrupt-Counter will not increased.

- **Syntax :**
  WORD PIOD48_IntGetCount(DWORD *dwIntCount);

- **Parameter:**
  dwIntCount : [Output] Address of dwIntCount, which will stores the counter value of interrupt.

- **Return:**
  Please refer to "Section 1.2 Error Code".

3.5.4 PIOD48_IntGetActiveFlag

- **Description:**

  This subroutine will read the Active-High and Active-Low flag from the device driver's memory queues (First-in-First-out, Buffer Size: 2000 flags for High/Low).

  The Active-Flag is used to records the Active-State-change of interrupt-source when the interrupt occurred. The Active-High-Flag records which interrupt-source changed to high state and the Active-Low-Flag records which interrupt-source changed to low state. Users can uses these flags to indicate which interrupt-source has changed.

  If the Active-Mode is set to Active-Low(/Active-High) only and the state for the Active-Low(/Active-High) is equal to zero, then the ISR will not increased the interrupt-counter, and the Active-Flag for High and Low will not recorded.

  If users have not calling this function to retrieve the flags from device driver's memory queues, these queues will stop record the flags (lost data) while the buffer is full. But the interrupt-counter will still counting while the ISR services the interrupt.

- **Syntax :**

  WORD PIOD48_IntGetActiveFlag
  (WORD *bActiveHighFlag, WORD *bActiveLowFlag);

- **Parameter:**

  bActiveHighFlag : [Output] Returns a flag that indicates which interrupt-source changed to High-State.
  bActiveLowFlag : [Output] Returns a flag that indicates which interrupt-source changed to Low-State.

- **Return:**

  Please refer to "Section 1.2 Error Code".

3.6 PIO-D48 COUNTER

The following PIOD48_XXX series function is designed for PIO-D48 card only.

3.6.1 PIOD48_SetCounter

- **Description :**
  This subroutine is used to set the 8254 counter's mode and value.

- **Syntax :**
  void PIOD48_SetCounter(DWORD dwBase, WORD wCounterNo,
  WORD bCounterMode, DWORD wCounterValue);

- **Parameter :**

dwBase	: [Input] I/O port addresses, please refer to function PIODIO_GetConfigAddressSpace. Only the low WORD is valid.
wCounterNo	: [Input] The 8254 Counter-Number: 0 to 2.
wCounterMode	: [Input] The 8254 Counter-Mode: 0 to 5. Please refer to the hardware manual for details.
wCounterValue	: [Input] The 16 bits value for the timer/counter to count. Only the low WORD is valid.

- **Return:**
  None

3.6.2 PIOD48_ReadCounter

- **Description :**
  This subroutine is used to read the 8254 counter's value.

- **Syntax :**
  DWORD PIOD48_ReadCounter
  (DWORD dwBase, WORD wCounterNo, WORD bCounterMode);

- **Parameter :**

dwBase	: [Input] I/O port addresses, please refer to function PIODIO_GetConfigAddressSpace. Only the low WORD is valid.
wCounterNo	: [Input] The 8254 Counter-Number: 0 to 2.
wCounterMode	: [Input] The 8254 Counter-Mode: 0 to 5. Please refer to the hardware manual for details.

- **Return:**
  Returns the 16 bits counter's value. (Only the low WORD is valid.)

3.6.3 PIOD48_SetCounterA

- **Description :**
  This subroutine is used to set the 8254 counter's mode and value.
  Users have to call the PIODIO_ActiveBoard() function before calling this function.

- **Syntax :**
  void PIOD48_SetCounterA(WORD wCounterNo,
  WORD bCounterMode, DWORD wCounterValue);

- **Parameter :**
  wCounterNo : [Input] The 8254 Counter-Number: 0 to 2.
  wCounterMode : [Input] The 8254 Counter-Mode: 0 to 5.
  Please refer to the hardware manual for details.
  wCounterValue : [Input] The 16 bits value for the counter to count.
  Only the low WORD is valid.

- **Return:**
  None

3.6.4 PIOD48_ReadCounterA

- **Description :**
  This subroutine is used to read the 8254 counter's value.
  Users have to call the PIODIO_ActiveBoard() function before calling this function.

- **Syntax :**
  DWORD PIOD48_ReadCounterA
  (WORD wCounterNo, WORD bCounterMode);

- **Parameter :**
  wCounterNo : [Input] The 8254 Counter-Number: 0 to 2.
  wCounterMode : [Input] The 8254 Counter-Mode: 0 to 5.
  Please refer to the hardware manual for details.

- **Return:**
  Returns the 16 bits counter's value. (Only the low WORD is valid.)

3.7 PIO-D64 COUNTER

The following PIOD64_XXX series function is designed for PIO-D64 card only.

3.7.1 PIOD64_SetCounter

- **Description :**
  This subroutine is used to set the 8254 counter's mode and value.
- **Syntax :**
  void PIOD64_SetCounter(DWORD dwBase, WORD wCounterNo,
  WORD bCounterMode, DWORD wCounterValue);
- **Parameter :**

dwBase	: [Input] I/O port addresses, please refer to function PIODIO_GetConfigAddressSpace. Only the low WORD is valid.
wCounterNo	: [Input] The 8254 Counter-Number: 0 to 5. (0 to 2: Chip-0, 3 to 5: Chip-1)
wCounterMode	: [Input] The 8254 Counter-Mode: 0 to 5. Please refer to the hardware manual for details.
wCounterValue	: [Input] The 16 bits value for the counter to count. Only the low WORD is valid.

- **Return:**
  None

3.7.2 PIOD64_ReadCounter

- **Description :**
  This subroutine is used to read the 8254 counter's value.
- **Syntax :**
  DWORD PIOD64_ReadCounter
  (DWORD dwBase, WORD wCounterNo, WORD bCounterMode);
- **Parameter :**

dwBase	: [Input] I/O port addresses, please refer to function PIODIO_GetConfigAddressSpace. Only the low WORD is valid.
wCounterNo	: [Input] The 8254 Counter-Number: 0 to 5. (0 to 2: Chip-0, 3 to 5: Chip-1)
wCounterMode	: [Input] The 8254 Counter-Mode: 0 to 5. Please refer to the hardware manual for details.

- **Return:**
  Returns the 16 bits counter's value. (Only the low WORD is valid.)

3.7.3 PIOD64_SetCounterA

- **Description :**
  This subroutine is used to set the 8254 counter's mode and value.
  Users have to call the PIODIO_ActiveBoard() function before calling this function.

- **Syntax :**
  void PIOD64_SetCounterA(WORD wCounterNo,
  WORD bCounterMode, DWORD wCounterValue);

- **Parameter :**
  wCounterNo : [Input] The 8254 Counter-Number: 0 to 5.
  (0 to 2: Chip-0, 3 to 5: Chip-1)
  wCounterMode : [Input] The 8254 Counter-Mode: 0 to 5.
  Please refer to the hardware manual for details.
  wCounterValue : [Input] The 16 bits value for the counter to count.
  Only the low WORD is valid.

- **Return:**
  None

3.7.4 PIOD64_ReadCounterA

- **Description :**
  This subroutine is used to read the 8254 counter's value.
  Users have to call the PIODIO_ActiveBoard() function before calling this function.

- **Syntax :**
  DWORD PIOD64_ReadCounterA
  (WORD wCounterNo, WORD bCounterMode);

- **Parameter :**
  wCounterNo : [Input] The 8254 Counter-Number: 0 to 5.
  (0 to 2: Chip-0, 3 to 5: Chip-1)
  wCounterMode : [Input] The 8254 Counter-Mode: 0 to 5.
  Please refer to the hardware manual for details.

- **Return:**
  Returns the 16 bits counter's value. (Only the low WORD is valid.)

3.8 PIO-D48 FREQUENCY

The following PIOD48_XXX series function is designed for PIO-D48 card only.

3.8.1 PIOD48_Freq

- **Description :**

  This subroutine is used to measurement the signal frequency. Users have to connect the signal(+) with CN1.Pin29, and connect the signal(-) with CN1.Pin19.

  It will uses the Counter-0 and Counter-1 to measure the frequency, thus users shouldn't use Counter-0 and Counter-1 for other purposes.

- **Syntax :**

  DWORD PIOD48_Freq(DWORD dwBase);

- **Parameter :**

  dwBase : [Input] I/O port addresses, please refer to function PIODIO_GetConfigAddressSpace.
  Only the low WORD is valid.

- **Return:**

  Returns the frequency value. (Only the low WORD is valid.)

3.8.2 PIOD48_FreqA

- **Description :**

  Please refer to the description of "PIOD48_Freq()" function.

  Users have to calling the "PIODIO_ActiveBoard()" function before calling this function.

- **Syntax :**

  DWORD PIOD48_FreqA();

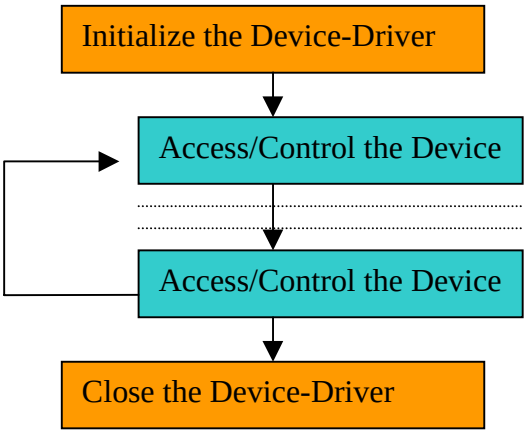
- **Parameter :**

  None

- **Return:**

  Returns the frequency value. (Only the low WORD is valid.)

4. Program Architecture




```
PIODIO_DriverInit()
Enable All DI/DO
….
    PIODIO_InputByte( … )
    ……..
    ……..
    PIODIO_OutputByte(…)
…..

PIODIO_DriverClose()
```