EPCIO Series Device Driver Library User Manual

Version: V.3.01

Date: 2008.07

http://www.epcio.com.tw

Table of Contents

I. INTRODUCTION TO THE DEVICE DRIVER LIBRARY 2

II. SETTING THE INTERRUPT AND RESET FUNCTIONS OF A MOTION CONTROL CARD 3

III. PULSE OUTPUT CONTROL 6

III.1 Basic Pulse Output Control 6

III.2 Controlling Pulse Command Registers (FIFOs) 9

III.3 Controlling Pulse Commands Being Sent 10

III.4 Emergency Stop of Pulse Output 10

III.5 Counting the Total Number of Pulses Already Output 11

III.6 Cyclic Interrupt Function 13

III.7 FIFO Minimum Stock Interrupt 17

IV. ENCODER CONTROL 22

IV.1 Basic Settings and Functions 22

IV.2 Encoder Count-Triggered Interrupt Service Routine 24

IV.3 Index Interrupt 28

IV.4 Count Latch 31

V. LOCAL I/O CONNECTION CONTROL 34

V.1 Basic Settings and Functions 34

V.2 Hardware Limit Switch Interrupt 41

V.3 Timer Interrupt 44

V.4 Watchdog 46

I. Introduction to the Device Driver Library

The EPCIO Series Device Driver Library can be used to drive motion control cards which are designed and developed with EPCIO ASIC and which have a PCI-Bus interface, such as EPCIO-4000, EPCIO-4005, EPCIO-6000 and EPCIO-6005.

The method of use of the EPCIO Series Device Driver Library will be detailed in the following chapters which are divided based on the functions listed below.

▲Bus Interface	Setting the interrupt and reset functions of a motion control card
▲DDA Control Interface	Pulse output control
▲Encoder Counter Interface	Encoder control
▲Local I/O Control Interface	Local input/output control
▲Remote Digital I/O Interface	Remote input/output control
▲ADC Control Interface	Analog-to-digital input control
▲PCL Control Interface	Setting a hardware position closed-loop
▲DAC Control Interface	Digital-to-analog output control

Related reference user manuals:

1. Hardware information.

- EPCIO –4000/4005 Hardware User Manual
- EPCIO –6000/6005 Hardware User Manual

2. Device driver user guides.

- EPCIO Series Device Driver Library Reference Manual
- EPCIO Series Device Driver Library Example Manual
- EPCIO Series Device Driver Library Integrated Testing Environment User Manual

II. Setting the Interrupt and Reset Functions of a Motion Control Card

The first step of using the EPCIO Series Device Driver Library is to initialize the motion control card. A motion control card can be initialized with the following functions:

EPCIO4000_Init()	applicable to EPCIO-4000, EPCIO-4005
EPCIO6000_Init()	applicable to EPCIO-6000, EPCIO-6005

For example, EPCIO6000_Init(), which can be used to initialize the EPCIO-6000 motion control card, is described below to demonstrate how to initialize a motion control card. The function declaration is as follows:

```
BOOL EPCIO6000_Init(    DDAISR      fnDDA_ISR,
                        ENCISR      fnENC012_ISR,
                        ENCISR      fnENC345_ISR,
                        ENCISR      fnENC678_ISR,
                        RIOISR      fnRIO0_ISR,
                        RIOISR      fnRIO1_ISR,
                        ADCISR      fnADC_ISR,
                        LIOISR      fnLIO_ISR,
                        PCLISR      fnPCL_ISR,
                        WORD        wCardIndex)
```

fnDDA_ISR ~ fnPCL_ISR are customized interrupt service routines. The interrupt function of a certain module can be disabled by inputting NULL into the corresponding parameter position.

wCardIndex is the motion control card index, which ranges from 0 to 11 and is to be selected by the user. This index is used in the EPCIO Series Device Driver Library to identify motion control cards. Therefore, different indices must be selected for different motion control cards respectively. Due to the limited range of the index, a PC can use a maximum of only 12 EPCIO series motion control cards at the same time. The initialization of motion control card is shown below in the example.

```
EPCIO6000_Init(   DDA_ISR_Function, NULL, NULL, NULL, NULL, NULL,
                  NULL, NULL, NULL, 0);
```

DDA_ISR_Function is the customized DDA interrupt service routine. Apart from being add through an initialization function (e.g., EPCIO6000_Init()), the customized interrupt service routine can be added via EPCIO_SetISRFunction(), which function, however, must be used before the initialization function is called. For more details, please refer to "**EPCIO Series Device Driver Library Reference Manual**".

If the return value of EPCIO6000_Init() is TRUE(1), it means that the motion control card has been successfully initialized, just after that other functions can be implemented.

To disable the EPCIO series motion control card, EPCIO_Close() must be used. This function disables all the functions of the EPCIO modules. If the interrupt function of the motion control card has been enabled, the interrupt vector will be restored, too.

In systems which operate in a relatively harsh environment, it may be necessary to call EPCIO_SetIntPeriod() in order to set the number of system clock cycles (25ns) the low active period of a PCI Bus interrupt signal will occupy when the interrupt signal is generated. For a general user, the default setting will do; the above function need not be used.

The EPCIO Series Device Driver Library also provides EPCIO_SetIntMode() for setting how the interrupt service function is triggered when PCI Bus generates an interrupt.

To further demonstrate the use of the EPCIO Series Device Driver Library, the following programming code shows how EPCIO_ResetModule() is applied to reset the specified EPCIO module. This function is often used in combination with an initial function.

```
if (EPCIO6000_Init(      DDA_ISR_Function, NULL, NULL, NULL, NULL, NULL,
                         NULL, NULL, NULL, 0)    )
{
        //Reset the EPCIO series motion control card of index 0.
        EPCIO_ResetModule(RESET_ALL, 0);

        //The following function is applicable EPCIO Series motion control cards.
        EPCIO_SetIntPeriod(250, 0);

        /*

        The programming code that the user wishes to execute.
        User-defined program

        */

        EPCIO_Close(0);//Disable the EPCIO series motion control card of index 0.
}
```

III. Pulse Output Control

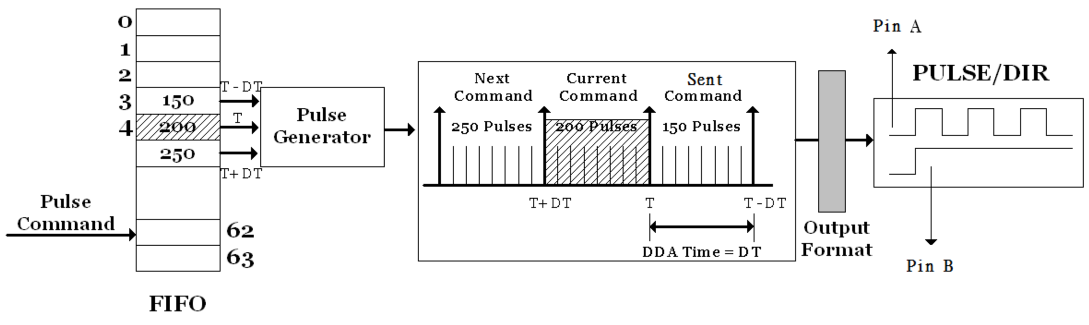
III.1 Basic Pulse Output Control

The EPCIO series motion control card has up to 6 output channels (channel 0 ~ channel 5) and uses the DDA module to control pulse output. The sequence consists essentially of the following two steps:

1. Send the pulse command to the **pulse command register (FIFO)** of the specified channel, in which the FIFO can store a maximum of 64 pulse commands.

2. At a time interval equal to the DDA Time (the DDA Time can be flexibly set), the pulse generator engine automatically reads one pulse command from the FIFO at a time and sends out pulses in the set output format through the specified channel evenly within the DDA Time.

The two steps are shown in the following diagram. Please note that each output channel has its own FIFO. On the EPCIO-6000 motion control card for example, there are 6 output channels and therefore 6 FIFOs. The diagram also shows that each DDA Time consumes one pulse command.



It can be known from the above that, in order to send out a pulse command, at least the following steps must be implemented:

1. Use EPCIO_DDA_SetTime() to set the DDA Time.

2. Set the pulse output format of the specified channel, including:
   A. Use EPCIO_DDA_SetPulseWidth() to set the pulse output width, multiply by the system clock(unit:25ns).
   B. Set the signal line of output pin A as inverted or otherwise.
      ➔ EPCIO_DDA_EnableOutAInverse()
      EPCIO_DDA_DisableOutAInverse()
   C. Set the signal line of output pin B as inverted or otherwise.
      ➔ EPCIO_DDA_EnableOutBInverse()
      EPCIO_DDA_DisableOutBInverse()
   D. Swap the two signal lines of output pins A and B or not.
      ➔ EPCIO_DDA_EnableABSwap()
      EPCIO_DDA_DisableABSwap()

3. Use EPCIO_DDA_EnableOutputChannel() to enable the output function of the specified channel.
   ➔ *See also* EPCIO_DDA_DisableOutputChannel()

4. Use EPCIO_DDA_StartEngine() to activate the DDA module.
   ➔ *See also* EPCIO_DDA_StopEngine()

5. Use EPCIO_DDA_SendPulse() to send the pulse command to the FIFO of the specified channel.

The following programming code shows how to send out a pulse command after a motion control card is successfully initialized. Whenever a position (pulse) or velocity (voltage) command is to be output from an EPCIO series motion control card, **EPCIO_LIO_EnablePulseDAC() must be used to enable the output function**. In certain servo systems, it may be necessary to call EPCIO_LIO_ServoOn() in order to enable the servo on connection of a servo motor drive; otherwise, the system cannot work properly. The complete calling procedure is as follows:

```
//Enable the pulse and velocity command output function of Card 0.
EPCIO_LIO_EnablePulseDAC(0);
```

```
//Enable the servo on connection of Channel 0 of Card 0.
EPCIO_LIO_ServoOn(0,0);

//Set DDA Time = 10 ms and DDA Length = 15 bits.
EPCIO_DDA_SetTime(10, DDA_LEN15, 0);

//Set the pulse output format of Channel 0 of Card 0 to Pulse/Dircetion.
EPCIO_DDA_SetOutputFormat(0, DDA_FMT_PD, 0);

//Set the pulse output width of Channel 0 of Card 0 to 100 multiply by the system
//clock.
EPCIO_DDA_SetPulseWidth(0, 100, 0);

//Enable the output function of Channel 0 of Card 0.
EPCIO_DDA_EnableOutputChannel(0, 0);

//Activate the DDA module.
EPCIO_DDA_StartEngine(0);

//Send out a pulse command, 200 pulses to be sent from Channel 0 of Card
//0 within 1 DDA Time.
EPCIO_DDA_SendPulse(0, 200, 0);
```

EPCIO_DDA_SetTime() and EPCIO_DDA_SetBitLength() can be used to set the upper limit of the total number of pulses that can be sent within each DDA time. Therefore, the total number of pulses in each pulse command sent via EPCIO_DDA_SendPulse() is subject to this limitation.

III.2 Controlling Pulse Command Registers (FIFOs)

The EPCIO Series Device Driver Library provides the following functions for controlling and acquiring the status of each FIFO.

1. EPCIO_DDA_CheckFIFOEmpty() can be used to check if the FIFO of the specified channel stores no pulse command at the present time.

2. EPCIO_DDA_CheckFIFOFull() can be used to check if the FIFO of the specified channel has no more register for storing pulse commands. Each FIFO has a maximum of 64 storage spaces.

3. EPCIO_DDA_GetStockCount() can be used to acquire the number of pulse commands that are currently stored in the FIFO of the specified channel but have not been executed yet.

4. EPCIO_DDA_EraseFIFOCmd() can be used to remove the pulse commands that are currently stored in the FIFO of the specified channel but have not been executed yet. Up to 64 commands can be deleted at a time. **Please note that the command currently being executed will not be affected.**

5. EPCIO_DDA_ShiftOutFIFOCmd() can be used to remove the next to-be-executed command in the FIFO of the specified channel. To remove a command from the FIFO with this function, it is necessary to disable the output function of the channel (i.e., EPCIO_DDA_DisableOutputChannel() must be called in advance). The output function of the channel is disabled, and then a FIFO command could be removed. All the channels in operation will not be affected.

The functions mentioned above allow sufficient use of the FIFO storage spaces. Pulse commands can be pre-stored into the FIFOs to prevent discontinuous motion attributable to a lack of pulse commands. This helps increase the stability of system operation, especially when a WINDOWS operating system is used without a real-time library.

III.3 Controlling Pulse Commands Being Sent

The EPCIO Series Device Driver Library provides the following functions for controlling and reading pulse commands which are being sent and therefore no longer in the FIFOs.

1. EPCIO_DDA_GetCurrentCmd() can be used to read the pulse command (including the positive/negative sign) currently being sent from the specified channel. The current direction of motion can be determined according to the positive/negative sign read with this function.

2. EPCIO_DDA_SetOutputFormat (channel, **DDA_FMT_NO**) can be used to nullify the output of the specified channel. All the FIFOs commands will be inhibited.

III.4 Emergency Stop of Pulse Output

Under certain circumstances, pulse output must be stopped at once. The EPCIO Series Device Driver Library provides the following functions to achieve this.

1. EPCIO_DDA_DisableOutputChannel() can be used to disable the output function of the specified channel, and the currently running command will not be affected and will be completed. After the executing command is finished, the commands in stock will stop sending.

2. EPCIO_DDA_StopEngine() can be used to disable the DDA module. This function will disable the output function of all channels.
   ➔ *See also* EPCIO_DDA_StartEngine()

3. EPCIO_DDA_EnableEmgcStop() can be used to enable the emergency stop function. This function can stop all channels from pulse output even if a command is currently being executed. The command in execution will be kept from output immediately. EPCIO will nevertheless keep on calculating pulse commands internally. Once the emergency stop function is canceled, pulse output from pulse commands will resume in the next DDA cycle.

➔ *See also* EPCIO_DDA_DisableEmgcStop()

4. When it is desired to immediately stop outputting the commands in a FIFO and the command currently being output, EPCIO_DDA_SetOutputFormat(the specified channel, DDA_FMT_NO) can be used to nullify the output of the specified channel.

EPCIO_DDA_EraseFIFOCmd() is often used in combination with the aforementioned output emergency stop function in order to remove pulse commands which are stored in the FIFO but have not been executed yet, as illustrated by the following programming code.

```
//Nullify the output of Channel 0 of Card 0.
EPCIO_DDA_SetOutputFormat (0, DDA_FMT_NO, 0);

//Disable the output function of Channel 0 of Card 0.
EPCIO_DDA_DisableOutputChannel(0, 0);

//Remove all the pulse commands stored in the FIFO of Channel 0 of Card 0.
EPCIO_DDA_EraseFIFOCmd(0, 64, 0);
```

III.5 Counting the Total Number of Pulses Already Output

The EPCIO Series Device Driver Library provides pulse counting functions for obtaining the total number of pulses actually output. These functions include:

1. EPCIO_DDA_EnablePulseCounter() for enabling the pulse counting function of the specified channel.
   ➔ *See also* EPCIO_DDA_DisablePulseCounter()

2. EPCIO_DDA_ClearCounter() for resetting the count of the pulse counter of the specified channel to zero.

3. EPCIO_DDA_GetOutputPulse() for acquiring the count of the pulse counter of the specified channel.

   ➔ *See also* EPCIO_DDA_GetOutputPulse()

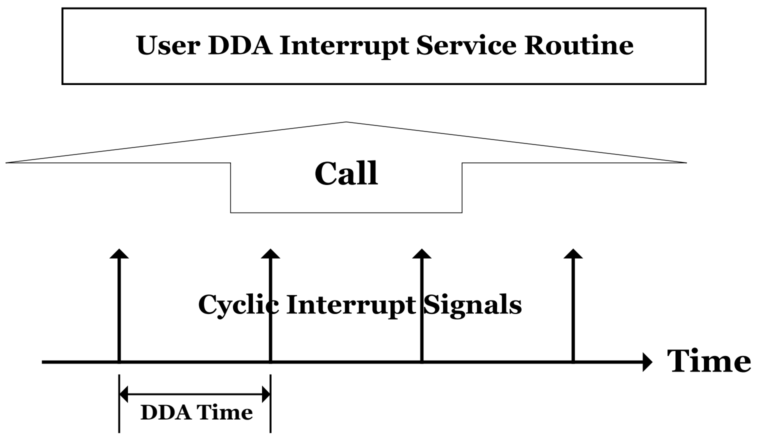
Before using EPCIO_DDA_GetOutputPulse(), the counting function must be enabled, i.e., EPCIO_DDA_EnablePulseCounter() must be called first. The following programming code shows how to acquire the total number of pulses output from Channel 0.

```
//Clear the count of the pulse counter of Channel 0 of Card 0.
EPCIO_DDA_ClearPulseCounter(0, 0);

//Enable the pulse counting function of Channel 0 of Card 0.
EPCIO_DDA_EnablePulseCounter(0, 0);
 .
 .
 .
long lPulseCount;
//Get the total number of pulses actually sent from Channel 0 of Card 0.
EPCIO_DDA_GetOutputPulse(0, &lPulseCount, 0);
```

It is common to compare the count of the pulse counter with the number of pulses actually sent via EPCIO_DDA_SendPulse() in order to verify whether EPCIO_DDA_SetPulseWidth(), EPCIO_SetTime() and EPCIO_SetBitLength() are correctly used.

III.6 Cyclic Interrupt Function

The EPCIO Series Device Driver Library provides a cyclic interrupt function. After the cyclic interrupt function is enabled, the device driver library automatically triggers the customized DDA interrupt service routine at an interval equal to the DDA time, as shown in the diagram below.



To use the cyclic interrupt functions, the following steps must be completed:

1. Declare and define the customized DDA interrupt service routine. The routine must declare as follows:

   typedef void(_stdcall *DDAISR)(DDAINT*);

   Therefore, the customized DDA interrupt service routine can be defined as follows:

```
void _stdcall DDA_ISR_Function(DDAINT *pstINTSource)
{
        if (pstINTSource->CYCLE)//Determine whether a cyclic interrupt occurs.
        {
            /*
```

```
                The programming code will be executed after the occurrence of a
                cyclic interrupt.
                */
            }
        }
```

The DDA interrupt service routine must determine whether it is triggered by a cyclic interrupt.

2. Add the customized DDA interrupt service routine.
   The DDA interrupt service routine must be added during initialization of the motion control card to be used. Taking the initialization of EPCIO-6000 for example, the function pointer of the interrupt service routine should be input into EPCIO6000_init() as follows:

   ```
   EPCIO6000_Init(    DDA_ISR_Function, NULL, NULL, NULL, NULL,
                      NULL, NULL, NULL, NULL, 0);
   ```

3. Use EPCIO_DDA_EnableCycleInt() to enable the cyclic interrupt function.
   ➔ *See also* EPCIO_DDA_DisableCycleInt()

The following programming code shows how the cyclic interrupt function is used

```
void _stdcall DDA_ISR_Function(DDAINT *pstINTSource)
{
        if (pstINTSource->CYCLE)//Determine whether a cyclic interrupt occurs.
        {
            /*
            The programming code will be executed after the occurrence of a
            cyclic interrupt.
```

```
                    */
            }
}
    •
    •
    •

if (EPCIO6000_Init(DDA_ISR_Function, NULL, NULL, NULL, NULL,
                         NULL, NULL, NULL, NULL, 0))
{
    •
    EPCIO_DDA_EnableCycleInt(0);
    •
}
```

Cyclic interrupt is a hardware interrupt having a relatively precise trigger cycle. It is typically used in tasks having a cyclic feature and requiring punctual execution. The cyclic interrupt function, when used in combination with a function for checking a FIFO status, guarantees that the number of commands in the FIFO can satisfy the need of the desired motion so that the motion will not stop due to an absence of commands in the FIFO.

Assume a total of 200 pulse commands need to be sent out. Since a FIFO can store maximum 64 commands at most, a cyclic interrupt can be used to trigger an interrupt service routine which acquires the number of commands currently stored in the FIFO, counts the remaining storage spaces in the FIFO, and thereby determines the number of commands that should be sent out of or can be sent into the FIFO. The foregoing actions of the interrupt service routine will be repeated until the 200 commands are all sent out. The following programming code illustrates the process described above.

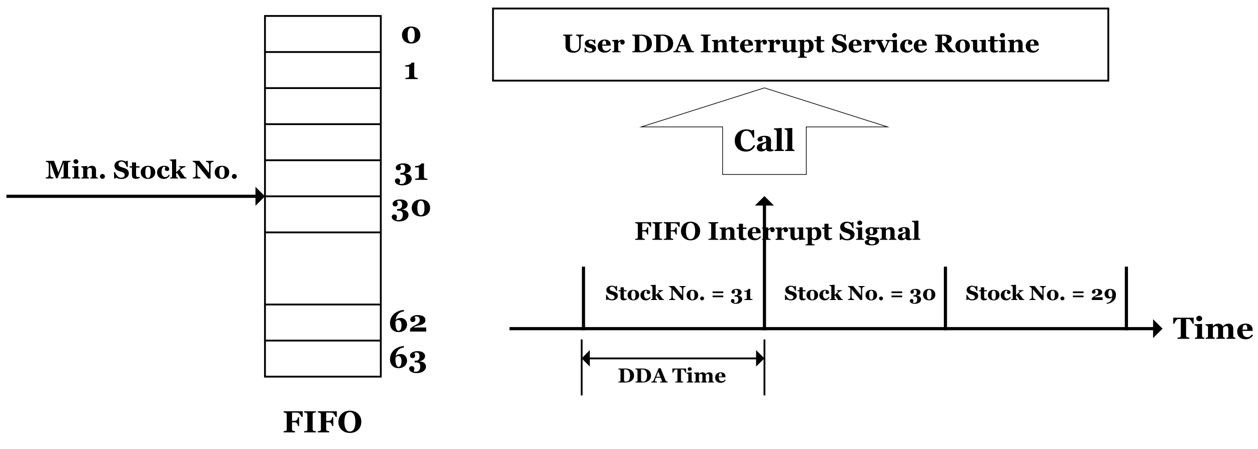
```
int nCount = 200;           //A total of 200 pulse commands need to be sent out.
int nPulse[200] = 150;      //The array contains 200 commands, which can be
                            //preprogrammed.

void _stdcall DDA_ISR_Function(DDAINT *pstINTSource)
{
        WORD wSockNo;

        if (pstINTSource->CYCLE)//Determine whether a cyclic interrupt occurs.
        {
           if (nCount)//Until nCount is equal to 0 when all the 200 commands
                            //have been sent out.
           {
                //Acquire the number of commands currently stored in the FIFO
                //of Channel 0 of Card 0.
                EPCIO_DDA_GetStockNo(0, &wStockNo, 0);
                //Determine the FIFO is exhausted or not.
                //Because the FIFO has only 64 storage spaces.
                for (int i = 0;i < 64 - wStockNo && nCount;i++)
                {
                    EPCIO_DDA_SendPulse(0, nPulse[200 - nCount], 0)
                    nCount--;
                }
           }
        }
    }
```

Because of its cyclic occurrence, cyclic interrupts can also be used to check a system's various statuses, such as the input/output states of I/O connections. Please note that an interrupt latency period exists between the time at which a cyclic interrupt occurs and the time at which an interrupt service routine is triggered for execution.

The impact of this time delay on system performance should be taken into consideration, especially when a WINDOWS operating system is used.

III.7 FIFO Minimum Stock Interrupt

The EPCIO Series Device Driver Library provides a FIFO minimum stock interrupt function (or FIFO interrupt for short). Once the minimum stock of each FIFO is set, and the FIFO interrupt function of the specified channel is enabled, the customized DDA interrupt service routine will be triggered when the commands stored in the FIFO of the specified channel are reduced, by being consumed, to as few as the minimum stock, as shown in the diagram below.



To use the FIFO interrupt function, the following steps must be completed:

1. Declare and define the customized DDA interrupt service routine. The routine must declare as follows:

   typedef void(_stdcall *DDAISR)(DDAINT*);

   Therefore, the customized DDA interrupt service routine can be defined as follows:

   void ***_stdcall*** DDA_ISR_Function(DDAINT *pstINTSource)

```
{
        //Determine whether a FIFO interrupt of Channel 0 occurs.
        if (pstINTSource->FIFO0)
        {
            /*
            The programming code will be executed after the occurrence of a FIFO
            interrupt of Channel 0.
            */
        }
        //Determine whether a FIFO interrupt of Channel 1 occurs.
        if (pstINTSource->FIFO1)
        {
            /*
            The programming code will be executed after the occurrence of a FIFO
            interrupt of Channel 1
            */
        }
 }
```

The DDA interrupt service routine must determine whether it is triggered by a FIFO interrupt. pstINTSource->FIFO0 ~ pstINTSource->FIFO5 are respectively used to determine whether FIFO interrupts of Channel 0 ~ Channel 5 occur.

2. Add the customized DDA interrupt service routine.
   The DDA interrupt service routine must be added during initialization of the motion control card to be used. Taking the initialization of EPCIO-6000 for example, the function pointer of the interrupt service routine should be input into EPCIO6000_Init() as follows:

```
EPCIO6000_Init(     DDA_ISR_Function, NULL, NULL, NULL, NULL,
                    NULL, NULL, NULL, NULL, 0);
```

3. Use EPCIO_DDA_SetMinStockNo() to set the minimum stock of each FIFO. For example, EPCIO_DDA_SetMinStockNo(30, 0) is called so that a FIFO interrupt occurs only when the commands stored in the FIFO are reduced from 31 to 30. **In other words, as long as the number of commands stored in the FIFO is equal to or smaller than 30, no interrupt will ever occur.**

4. Use EPCIO_DDA_EnableStockInt() to enable the FIFO interrupt function of the specified channel.

   ➔ *See also* EPCIO_DDA_DisableStockInt()

The following programming code shows how to use the FIFO interrupt function.

```
void _stdcall DDA_ISR_Function(DDAINT *pstINTSource)
{
            //Determine whether a FIFO interrupt of Channel 0 occurs.
            if (pstINTSource->FIFO0)
            {
                /*
                The programming code will be executed after the occurrence of a FIFO
                interrupt.
                */
            }
           }

if (EPCIO6000_Init(     DDA_ISR_Function, NULL, NULL, NULL, NULL,
                        NULL, NULL, NULL, NULL, 0))
{
    //Set the minimum stock of the FIFO of channel 0 of card 0 to 30.
    EPCIO_DDA_SetMinStockNo(30, 0);
    EPCIO_DDA_EnableStockInt(0);
}
```

FIFO interruption is a hardware interrupt and just happens when interrupt conditions is satisfied, in other hand cyclic interrupt occurs in a cycle way. Therefore, FIFO interruption has less computing cost than cyclical interruption. Generally, the FIFO interrupt function, when used in combination with a function for checking a FIFO's status, can guarantee that the number of commands in the FIFO will not be smaller than a preset value (i.e., the minimum stock of the FIFO), and that therefore the motion will not stop due to an absence of commands in the FIFO.

Assume a total of 200 pulse commands need to be sent out. Since a FIFO can store maximum 64 commands, a FIFO interrupt can be used to trigger an interrupt service routine which get the number of commands currently stored in the FIFO, counts the remaining storage spaces in the FIFO, and thereby determines the number of commands that should be sent out of or can be sent into the FIFO. The foregoing actions of the interrupt service routine will be repeated until the 200 commands are all sent out.

To trigger the FIFO interrupt, 64 commands must be sent to the FIFO in the first place. Only then can the condition of triggering the FIFO interrupt (i.e, the number of commands stored in the FIFO is reduced from 31 to 30) take place. The following programming code illustrates the process described above.

```
int nCount = 200;          //A total of 200 pulse commands need to be sent out.
int nPulse[200] = 150;     //A array contains 200 commands, which can be
                           //preprogrammed.

for (int i = 0;i < 64;i++)  //Send 64 commands to the FIFO of Channel 0 of Card 0.
{
        EPCIO_DDA_SendPulse(0, nPulse[200 - nCount], 0)
        nCount--;
}
void _stdcall DDA_ISR_Function(DDAINT *pstINTSource)
{
            WORD wStockNo;
            if (pstINTSource->FIFO0)
            {
                if (nCount)//The nCount is equal to 0 when all the 200 commands
                           //have been sent out.
                 {
                    //Get the number of commands currently stored in the FIFO
                    //of Channel 0 of Card 0.
                    EPCIO_DDA_GetStockNo(0, &nStockNo, 0);

                    //The FIFO has spaces for storing up to 64 commands.
                    for (int i = 0;i < 64 - nStockNo && nCount;i++)
                    {
                        EPCIO_DDA_SendPulse(0, nPulse[200 - nCount] , 0)
                        nCount--;
                    }
                }
            }
    }
```

As with cyclic interrupt, an interrupt latency period exists between the time at which a FIFO interrupt condition is satisfied and the time at which an interrupt service routine is triggered for execution. The impact of this time delay on system performance should be taken into consideration, especially when a WINDOWS operating system is used.

IV. Encoder Control

IV.1 Basic Settings and Functions

An EPCIO series motion control card has up to 9 channels (channel 0 ~ channel 8) for encoder signal input. To use functions related to encoder control, the following steps must be completed first:

1. Use EPCIO_ENC_SetFilterClock() to enable the encoder filtering, sampling function and set the sampling rate. The sampling rate is System Clock / (Divider + 1). Once the sampling rate is set, the input signal must be identical (HIGH or LOW) in three consecutive samples to be considered as an effective input.

2. Use EPCIO_ENC_SetInputType() to set the input signal format for the specified channel. The input signal format must match hardware settings. When the input signal is a motor encoder feedback signal, please refer to motor or drive settings; when a common MPG handwheel is used, set the input signal format to A/B phase (Default: A/B phase input).

3. Use EPCIO_ENC_SetInputRate() to set the counter signal-decoding multiplier for the specified channel. The decoding multiplier is effective only when the inputted encoder format is A/B phase. **To use this function, EPCIO_ENC_SetInputType() must be set to A/B phase.**

4. Use EPCIO_ENC_StartInput() to enable the counting function of the counter. Before using this function, EPCIO_ENC_ClearCounter() is typically called to reset the encoder counting number to zero.

Once the above settings are made, EPCIO_ENC_GetValue() can be used to get the count of the specified channel.

The following programming code shows how to get the count of Channel 0.

```
long lCounter
```

```
//Set the filtering and sampling clock of Card 0.
EPCIO_ENC_SetFilterClock(2, 0);


//Set the input format of Channel 0 of Card 0 to A/B phase.
EPCIO_ENC_SetInputType(0, ENC_TYPE_AB, 0);


//Set the signal-decoding multiplier of Channel 0 of Card 0 to multiply by 4.
EPCIO_ENC_SetInputRate(0, ENC_RATE_X4, 0);


//Reset the encoder counting number of Channel 0 of Card 0 to zero.
EPCIO_ENC_ClearCounter(0, 0);


//Enable the counting function of Card 0.
EPCIO_ENC_StartInput(0);


//Get the encoder counting number of Channel 0 of Card 0.
EPCIO_ENC_GetValue(0, & lCounter, 0);
```

To satisfy the actual wiring needs, the EPCIO Series Device Driver Library provides the following functions for inverting the signals sent to the counter input pins.

1. Use EPCIO_ENC_EnableInAInverse() to invert the counter input signal at pin A of the specified channel. The default setting is "**No** Inverse".
   ➔ *See also* EPCIO_ENC_DisableInAInverse()

2. Use EPCIO_ENC_EnableInBInverse() to invert the counter input signal at pin B of the specified channel. The default setting is "**No** Inverse".
   ➔ *See also* EPCIO_ENC_DisableInBInverse()

3. Use EPCIO_ENC_EnableInCInverse() to invert the counter input signal at pin C

of the specified channel. The default setting is "**No** Inverse".

➔ *See also* EPCIO_ENC_DisableInCInverse()

4. Use EPCIO_ENC_EnableInABSwap() to swap the counter input signals at pins A and B of the specified channel before the signals enter the counter. The default setting is "**Non-**Swapping".

   ➔ *See also* EPCIO_ENC_DisableInABSwap()

IV.2 Encoder Count-Triggered Interrupt Service Routine

The "encoder count-triggered interrupt service routine" function (or count-triggered interrupt for short) provided in the EPCIO Series Device Driver Library allows the user to set a comparison value for the specified channel. So that, after the function is enabled for the specified channel, a comparator automatically triggers the customized interrupt service routine upon the channel's count equaling the comparison value. To use the count-triggered interrupt function, the following steps must be completed:

1. Define and declare the customized ENC interrupt service routine. The declaration of an ENC interrupt service routine is as follows:

```
typedef void(_stdcall *ENCISR)(ENC INT*);
```

Therefore, the customized ENC interrupt service routine can be defined as follows:

```
void _stdcall ENC_ISR_Function1(ENCINT *pstINTSource)
{
        //Determine whether the interrupt service routine is triggered by the count
         //of the 0-th channel of this channel group.
        if (pstINTSource->COMP0)
        {
            /*
            The programming code will be executed after the count triggers an
            interrupt.
            */
        }
}
```

The ENC interrupt service routine must determine whether it is triggered by a count. In encoder-related interrupt functions, Channel 0 ~ Channel 8 are grouped into groups of three (i.e., divided into three groups: Channel 0 ~ Channel 2, Channel 3 ~ Channel 5 and Channel 6 ~ Channel 8). Each group has a corresponding ENC interrupt service routine and, when this ENC interrupt service routine is triggered by a count, uses COMP0 ~ COMP2 to determine which channel’s encoder count has triggered the routine. The relationships between the channels and COMP0 ~ COMP2 are as follows.

```
Channel 0：pstINTSource -> COMP0      ----
Channel 1：pstINTSource -> COMP1      --- ENC_ISR_Function1()
Channel 2：pstINTSource -> COMP2      ----

Channel 3：pstINTSource -> COMP0      ----
Channel 4：pstINTSource -> COMP1      |--- ENC_ISR_Function2()
Channel 5：pstINTSource -> COMP2      ----

Channel 6：pstINTSource -> COMP0      ----
Channel 7：pstINTSource -> COMP1      |--- ENC_ISR_Function3()
Channel 8：pstINTSource -> COMP2      ----
```

pstINTSource->COMP0 ~ pstINTSource->COMP2 are used to determine whether the interrupt service routine is triggered by the encoder count of the 0-th, 1-st or 2-nd channel of the channel group. The number of the channel (range from 0 to 8) depends on where the function pointer of the ENC interrupt service routine is input in the initialization function (e.g., EPCIO6000_Init()), as explained in more detail in the next step.

2. Add the customized ENC interrupt service routine.

   The ENC interrupt service routine must be added during initialization of the motion control card to be used. Taking the initialization of EPCIO-6000 for example, the function pointer of the interrupt service routine should be input into EPCIO6000_Init() as follows:

```
EPCIO6000_Init(   NULL,
                  ENC_ISR_Function1,//   for Channel 0 ~ Channel 2
                  ENC_ISR_Function2,//   for Channel 3 ~ Channel 5
                  ENC_ISR_Function3,//   for Channel 6 ~ Channel 8
                  NULL, NULL, NULL, NULL, NULL, 0);
```

   where ENC_ISR_Function1, ENC_ISR_Function2, and ENC_ISR_Function3 are the interrupt service routines for use by Channel 0 ~ Channel2, Channel 3 ~ Channel 5 and Channel 6 ~ Channel 8 respectively.

3. Use EPCIO_ENC_SetCompValue() to set the comparison value of the counter of the specified channel.

4. Use EPCIO_ENC_EnableCompInt() to enable the “encoder count-triggered interrupt service routine” function of the specified channel.

   ➔ *See also* EPCIO_ENC_DisableCompInt()

In the following programming code, the “encoder count-triggered interrupt service routine” function is enabled for channel 4. Please note where the function pointer of ENC_ISR_Function() is input in EPCIO6000_Init().

```
void _stdcall ENC_ISR_Function2(ENCINT *pstINTSource)
    {
        //Determine whether the routine is triggered by the encoder count of the 4-th
        //channel.
            if (pstINTSource->COMP1)
            {
                /*
                The programming code will be executed after the encoder count
                triggers the interrupt service routine.
                */
            }
    }
  .
  .
if (EPCIO6000_Init(      NULL, NULL, ENC_ISR_Function, NULL, NULL,
                         NULL, NULL, NULL, NULL, 0))
{
      .
      .
      //Set the comparison value of the counter of channel 4 of Card 0 to 10000.
      EPCIO_ENC_SetCompValue(4, 10000, 0);

      //Enable the "encoder count-triggered interrupt service routine" function of
      //channel 4 of card 0.
      EPCIO_ENC_EnableCompInt(4, 0);
      .
      .
}
```

IV.3 Index Interrupt

The EPCIO Series Device Driver Library provides an encoder index interrupt function. So that, when an encoder index (Z phase) signal is input, the customized interrupt service routine will be triggered. To use the index interrupt function, the following settings must be completed:

1. Define and declare the customized ENC interrupt service routine. The declaration of an ENC interrupt service routine is as follows:

```
typedef void(_stdcall *ENCISR)(ENC INT*);
```

Therefore, the customized ENC interrupt service routine can be defined as follows:

```
void _stdcall ENC_ISR_Function1(ENCINT *pstINTSource)
{
        //Determine whether the interrupt service routine is triggered by the index
        //signal of the 0-th channel of this channel group.
         if (pstINTSource->INDEX0)
         {
             /*
             The programming codes will be executed after the occurrence of an
             index interrupt.
             */
         }
 }
```

The ENC interrupt service routine must determine whether it is triggered by an index interrupt. In encoder related interrupt functions, Channel 0 ~ Channel 8 are grouped into groups of three (i.e., divided into three groups: Channel 0 ~ Channel 2, Channel 3 ~ Channel 5 and Channel 6 ~ Channel 8). Each group has a corresponding ENC interrupt service routine. When an index interrupt occurs, uses INDEX0 ~ INDEX2 to determine which channel's index signal has triggered the interrupt. The relationships between the channels and INDEX0 ~ INDEX2 are as follows.

```
Channel 0：pstINTSource -> INDEX0      ----
Channel 1：pstINTSource -> INDEX1      |--- ENC_ISR_Function1()
Channel 2：pstINTSource -> INDEX2      ----

Channel 3：pstINTSource -> INDEX0      ----
Channel 4：pstINTSource -> INDEX1      |--- ENC_ISR_Function2()
Channel 5：pstINTSource -> INDEX2      ----

Channel 6：pstINTSource -> INDEX0      ----
Channel 7：pstINTSource -> INDEX1      |--- ENC_ISR_Function3()
Channel 8：pstINTSource -> INDEX2      ----
```

pstINTSource->INDEX0 ~ pstINTSource->INDEX2 are used to determine whether the index interrupt is associated with the 0-th, 1-st, or 2-nd channel of the channel group. The number of the channel (ranging from 0 to 8) depends on where the function pointer of the ENC interrupt service routine is input in the initialization function (e.g., EPCIO6000_Init()), as explained in more detail in the next step.

In addition, the declaration of the ENC interrupt service routine must include the keyword _stdcall.

2. Add the customized ENC interrupt service routine.
   The ENC interrupt service routine must be added during initialization of the motion control card to be used. Taking the initialization of EPCIO-6000 for example, the function pointer of the interrupt service routine should be input into EPCIO6000_Init() as follows:

```
EPCIO6000_Init(     NULL,
                    ENC_ISR_Function1,//   for Channel 0 ~ Channel 2
                    ENC_ISR_Function2,//   for Channel 3 ~ Channel 5
                    ENC_ISR_Function3,//   for Channel 6 ~ Channel 8
                    NULL, NULL, NULL, NULL, NULL, 0);
```

where ENC_ISR_Function1, ENC_ISR_Function2 and ENC_ISR_Function3 are the index interrupt service routines for used by Channel 0 ~ Channel 2, Channel 3 ~ Channel 5 and Channel 6 ~ Channel 8 respectively.

3. Use EPCIO_ENC_EnableIndexInt() to enable the index interrupt triggering function of the specified channel.

   ➔ *See also*   EPCIO_ENC_DisableIndexInt()
                  EPCIO_ENC_GetIndexStatus()

In the following programming code, the index interrupt function is enabled for Channel 5 alone. Please note where the function pointer of ENC_ISR_Function() is input in EPCIO6000_Init().

```
void _stdcall ENC_ISR_Function(ENCINT *pstINTSource)
{
    //Determine whether the interrupt service routine is triggered by the index signal
    //of the 5-th channel of Card 0.
    if (pstINTSource->INDEX2)
    {
            /*
            The programming codes will be executed after the occurrence of an index
             interrupt.
            */
    }
}
```
.

•

```
if (EPCIO6000_Init(      NULL, NULL, ENC_ISR_Function, NULL, NULL,
                         NULL, NULL, NULL, NULL, 0))
{
     //Enable the index interrupt triggering function of Channel 5 of Card 0.
     EPCIO_ENC_EnableIndexInt(5, 0);
}
```

IV.4 Count Latch

The EPCIO Series Device Driver Library provides a "count latch" function that allows users to set the signal sources that trigger the encoder count to be recorded in the latch register. Users can also use functions in the device driver library to get the recorded value in the latch register.

To use the count latch function, the sources of trigger signals must be set in advance with EPCIO_ENC_SetTrigSource(). The declaration of this function is as follows:

```
     BOOL EPCIO_ENC_SetTrigSource(WORD wChannel,
                                  WORD wSource,
                                  WORD wCardIndex);
```

wChannel is the number of a channel and ranges from 0 to 8. wSource is the source(s) of trigger signals. There is a total of 15 eligible trigger source signals for triggering the latch of an encoder count. The source setting can be a union of a number of sources. The eligible sources of trigger signals include:

ENC_TRIG_NO	No trigger signal source selected
ENC_TRIG_INDEX0	Index signal in encoder Channel 0
ENC_TRIG_INDEX1	Index signal in encoder Channel 1
ENC_TRIG_INDEX2	Index signal in encoder Channel 2
ENC_TRIG_INDEX3	Index signal in encoder Channel 3
ENC_TRIG_INDEX4	Index signal in encoder Channel 4

ENC_TRIG_INDEX5	Index signal in encoder Channel 5
ENC_TRIG_INDEX6	Index signal in encoder Channel 6
ENC_TRIG_INDEX7	Index signal in encoder Channel 7
ENC_TRIG_INDEX8	Index signal in encoder Channel 8
ENC_TRIG_LIO0	Interrupt request from local I/O connection DI 0
ENC_TRIG_LIO1	Interrupt request from local I/O connection DI 1
ENC_TRIG_RDI0	Interrupt request from remote I/O connection Set 0 Slave 0 DI 0
ENC_TRIG_RDI1	Interrupt request from remote I/O connection Set 0 Slave 0 DI 1
ENC_TRIG_ADC0	ADC comparator INT of Channel 0
ENC_TRIG_ADC1	ADC comparator INT of Channel 1

Once the source(s) of trigger signals are set, the encoder count will be recorded in the latch register upon occurrence of the trigger signal(s). However, before EPCIO_ENC_StartInput() is used to start the count latch function, EPCIO_ENC_SetTrigMode() must be called to set the latch mode. The declaration of this function is as follows:

```
EPCIO_ENC_SetTrigMode( WORD wChannel,
                       WORD wMode,
                       WORD wCardIndex)
```

wChannel is the number of a channel and ranges from 0 to 8. wMode is the latch trigger mode, which can be either of the following:

ENC_TRIG_FIRST	When the trigger condition is satisfied for the first time, the encoder count is latched and will no longer be changed.
ENC_TRIG_LAST	The encoder count is latched when the trigger condition is satisfied, but if the trigger condition is satisfied again, the latched count will be updated.

EPCIO_ENC_GetLatchValue() can be used to get from the latch register the recorded value of the specified channel.

The following programming code shows how the trigger signal source is set as serially connected to the index signal of encoder channel 0. The programming code also shows that the latched, recorded value is acquired after the index signal occurs, in order to the user to know from this value the actual location of the index triggered.

```
void _stdcall ENC_ISR_Function(ENCINT *pstINTSource)
{
    //Determine whether the function is triggered by the index signal of the 0-th
    //channel.
    if (pstINTSource->INDEX0)
    {
        //The programming code will be executed after the occurrence of an index
        //interrupt.
        long lLatchValue;

        //Get the encoder count from the latch register.
        EPCIO_ENC_GetLatchValue(0, &lLatchValue, 0);
    }
}
if (EPCIO6000_Init(    NULL, ENC_ISR_Function, NULL, NULL, NULL,
                       NULL, NULL, NULL, NULL, 0))
{
    //Set the counter latch trigger source of Channel 0 of Card 0 to the encoder
    //index signal.
    EPCIO_ENC_SetTrigSource(0, ENC_TRIG_INDEX0, 0);

    //Set the counter latch trigger mode of Channel 0 of Card 0 to continuous trigger.
    EPCIO_ENC_SetTrigMode(0, ENC_TRIG_LAST, 0);

}
```

V. Local I/O Connection Control

There is a total of 28 local I/O connections, which can be programmed for input or output. However, for an EPCIO series motion control card, the specific uses of these I/O connections are already programmed as summarized below:

Total of 20 input connections, which include:

Home Sensor	6 input connections.
Limit Switch Plus (+)	6 input connections.
Limit Switch Minus (-)	6 input connections.
Status for Getting 24V	1 input connection.
Status for Emgergency Stop	1 input connection.

Total of 8 output connections, which include:

Servo On/Off	6 output connections.
Enabling Position Ready	1 output connection.
Enabling Pulse DAC	1 output connection.

The following sections explain how these local I/O connections are used.

V.1 Basic Settings and Functions

Before using any local I/O connection, EPCIO_LIO_EnableLDOOutput() must be called to enable the output function of the connection. The declaration of this function is as follows:

```
BOOL EPCIO_LIO_EnableLDOOutput(WORD wPort,
                               WORD wCardIndex);
```

The local digital output connections are grouped into groups of four. The 28 local I/O connections are divided into Port 0 ~ Port 6, each consisting of 4 connections. This function can enable and disable the output function of each port independently. The default output statuses of all the ports are disabled. This function defines each port as including 4 connections, and the parameter port can be any of the following:

LIO_OUT_EN0	representing Port 0	(LDO 0 ~ LDO 3)
LIO_OUT_EN1	representing Port 1	(LDO 4 ~ LDO 7)
LIO_OUT_EN2	representing Port 2	(LDO 8 ~ LDO 11)
LIO_OUT_EN3	representing Port 3	(LDO 12 ~ LDO 15)
LIO_OUT_EN4	representing Port 4	(LDO 16 ~ LDO 19)
LIO_OUT_EN5	representing Port 5	(LDO 20 ~ LDO 23)
LIO_OUT_EN6	representing Port 6	(LDO 24 ~ LDO 27)

➔ *See also* EPCIO_LIO_DisableLDOOutput()

EPCIO_LIO_GetLDIInput() can be used to acquire the digital signal input values of Local LDI 0 ~ LDI 27. The declaration of this function is as follows:

```
BOOL EPCIO_LIO_GetLDIInput(DWORD *pdwInput, WORD wCardIndex);
```

Bit 0 ~ bit 27 of *pdwInput obtained with this function represent the input statuses of Local LDI 0 ~ LDI 27, whereas bit 28 ~ bit 31 have no meaning.

Therefore, if the input value obtained with EPCIO_LIO_GetLDIInput(&dwInput, 0) is 0x0002, it means that the digital signal input value of LDI 1 of Card 0 is currently 1. This is because 0x0002 can be converted into the binary number 0b0000000000000010, in which the bit corresponding to LDI 1 has a value of 1.

Similarly, EPCIO_LIO_SetLDOOutput() can be used to set the digital signal output values of Local LDO 0 ~ LDO 27. The declaration of this function is as follows:

```
BOOL EPCIO_LIO_SetLDOOutput(DWORD dwValue,
                            WORD wCardIndex);
```

Bit 0 ~ bit 27 of the parameter dwValue in this function represent the output statuses of Local LDO 0 ~ LDO 27, whereas bit 28 ~ bit 31 have no meaning.

Therefore, EPCIO_LIO_SetLDOOutput(0x0021, 0) means to output signals to LDO 0 and LDO 5 of Card 0. This is because 0x0021 can be converted into the binary number 0b0000000000100001, in which the bits corresponding to LDO 0 and LDO 5 are set to 1. Please refer to the following tables for the meaning of each bit when using the foregoing functions.

Applicable to EPCIO-4000 and EPCIO-4005 control cards.

LDIO	Definition	Corresponding SCSI II Pin	Note
0	Channel 0 OT+	8 (to connect with external connection)	Can trigger an interrupt
1	Channel 1 OT+	42 (to connect with external connection)	Can trigger an interrupt
2	Channel 2 OT+	12 (to connect with external connection)	Can trigger an interrupt
3	Channel 3 OT+	46 (to connect with external connection)	Can trigger an interrupt
4	Channel 0 OT-	9 (to connect with external connection)	Can trigger an interrupt
5	Channel 1 OT-	43 (to connect with external connection)	Can trigger an interrupt
6	Channel 2 OT-	13 (to connect with external connection)	Can trigger an interrupt
7	Channel 3 OT-	47 (to connect with external connection)	
8	Channel 0 HOME	7 (to connect with external connection)	
9	Channel 1 HOME	41 (to connect with external connection)	
10	Channel 2 HOME	11 (to connect with external connection)	
11	Channel 3 HOME	45 (to connect with external connection)	
12 ~ 15	Reserved (unused)		
16	INH_O0	10 (to connect with external connection)	
17	INH_O1	44 (to connect with external connection)	
18	INH_O2	14 (to connect with external connection)	
19	INH_O3	48 (to connect with external connection)	
20	P_RDY (PCI Bus)	40 (to connect with external connection)	
21	Reserved (unused)		

22	Reserved (for internal use)		
23	PULSE_DA_OUTPUT_ENABLE	Not to connect with external connection	
24	Reserved (for internal use)	Not to connect with external connection	
25	Reserved (for internal use)	Not to connect with external connection	
26	Reserved (for internal use)	Not to connect with external connection	
27	Reserved (for internal use)	Not to connect with external connection	

Applicable to EPCIO-6000 and EPCIO-6005 control cards.

LDIO	Definition	Corresponding SCSI II Pin	Note
0	Channel 0 OT+	10 (to connect with external connection)	Can trigger an interrupt
1	Channel 1 OT+	60 (to connect with external connection)	Can trigger an interrupt
2	Channel 2 OT+	14 (to connect with external connection)	Can trigger an interrupt
3	Channel 3 OT+	64 (to connect with external connection)	Can trigger an interrupt
4	Channel 4 OT+	18 (to connect with external connection)	Can trigger an interrupt
5	Channel 5 OT+	68 (to connect with external connection)	Can trigger an interrupt
6	Channel 0 OT-	11 (to connect with external connection)	Can trigger an interrupt
7	Channel 1 OT-	61 (to connect with external connection)	
8	Channel 2 OT-	15 (to connect with external connection)	
9	Channel 3 OT-	65 (to connect with external connection)	
10	Channel 4 OT-	19 (to connect with external connection)	
11	Channel 5 OT-	69 (to connect with external connection)	
12	Channel 0 HOME	9 (to connect with external connection)	
13	Channel 1 HOME	59 (to connect with external connection)	
14	Channel 2 HOME	13 (to connect with external connection)	

15	Channel 3 HOME	63 (to connect with external connection)	
16	INH_O0	12 (to connect with external connection)	
17	INH_O1	62 (to connect with external connection)	
18	INH_O2	16 (to connect with external connection)	
19	INH_O3	66 (to connect with external connection)	
20	INH_O4	20 (to connect with external connection)	
21	INH_O5	70 (to connect with external connection)	
22	P_RDY	58 (to connect with external connection)	
23	PULSE_DA_ OUT_PUT_ENABLE	Not to connect with external connection	
24	PASS CHECK	Not to connect with external connection	
25	PASS CHECK	Not to connect with external connection	
26	PASS CHECK	Not to connect with external connection	
27	PASS CHECK	Not to connect with external connection	

As the specific uses of these I/O connections are already programmed for EPCIO series motion control cards, one who uses the EPCIO-400-1, EPCIO-400-2, EPCIO-601-1, or EPCIO-601-2 adapter board can use the following functions to facilitate reading from or inputting into the I/O connections of the adapter board. When these functions are used to set the statuses of the output connections, the output functions of the output connections are automatically enabled, so EPCIO_LIO_EnableLDOOutput() need not be used.

The EPCIO Series Device Driver Library provides the following functions for acquiring the statuses of input connections.

1. EPCIO_LIO_GetHomeSensor() for acquiring the HOME sensor status of the specified channel. When the HOME sensor status is changed, no interrupt signal will be generated. The HOME sensor status of the specified channel can only be

checked with this function.

2. EPCIO_LIO_GetOverTravelUp() for checking whether the hardware limit switch plus of the specified channel is triggered. If the hardware limit switch plus is triggered, collision may occur; the user should take necessary measures immediately. A customized interrupt service routine can be triggered when the hardware limit switch plus of any of Channel 0 ~ Channel 5 of EPCIO-6000 or EPCIO-6005, or of Channel 0 ~ Channel 3 of EPCIO-4000 or EPCIO-4005 is triggered.

3. EPCIO_LIO_GetOverTravelDown() for checking whether the hardware limit switch minus of the specified channel is triggered. If the hardware limit switch minus is touched, collision may occur; the user should take necessary measures immediately. A customized interrupt service routine can be triggered when the hardware limit switch minus of Channel 0 ~ Channel 5 of EPCIO-6000 or EPCIO-6005, or any of Channel 0 ~ Channel 3 of EPCIO-4000 or EPCIO-4005 is triggered.

4. EPCIO_LIO_Get24VSensor() for acquiring the status of 24V voltage input.

5. EPCIO_LIO_GetEmgcStopStatus() for acquiring the status of the emergency stop switch.

The EPCIO Series Device Driver Library provides the following functions for setting the statuses of output connections.

1. EPCIO_LIO_ServoOff() for enabling input inhibition of the specified channel. The connection in question can be connected with the input-inhibited connection point of the motor drive. Once this function is called, the specified channel can no longer receive position or velocity commands. After the initialization function (e.g., EPCIO4000_Init()) is successfully called, input inhibition is enabled by default.

2. EPCIO_LIO_ServoOn() for disabling input inhibition of the specified channel. The connection in question can be connected with the input-inhibited motor drive point. Once this function is called and set, the specified channel can receive position or velocity commands from an EPCIO series motion control cards.

3. EPCIO_LIO_DisablePrdy() for disabling position ready output. The output connection point in question can be connected with the plug of the power switch control. Once this function is called, the latter connection point is open-circuited. After the initialization function (e.g., EPCIO4000_Init()) is successfully called, position ready output is disabled by default.

4. EPCIO_LIO_EnblePrdy() for enabling position ready output. The connection point in question can be connected with the corresponding power switch control point. Once this function is called, the latter connection point is closed.

5. EPCIO_LIO_DisablePulseDAC() for disabling the position (pulse) and velocity (voltage) command output function of an EPCIO series motion control card. Once this function is called, the output function is disabled. After the initialization function (e.g., EPCIO4000_Init()) is successfully called, the output function is disabled by default.

6. EPCIO_LIO_EnablePulseDAC() for enabling the position (pulse) and velocity (voltage) command output function of an EPCIO Series motion control card. Once this function is called, the output function is enabled.

While all the output connections of an EPCIO series motion control card have specific uses, the output connections can also be used for general output. For example, a channel connected with a stepping motor does not require Servo On/Off signal control, so the Servo On/Off output connection of this channel can be used for general output.

V.2 Hardware Limit Switch Interrupt

The limit switch plus and limit switch minus (also known as over-travel limit switches) of certain channels of an EPCIO series motion control card provide the hardware limit switch interrupt function (or limit interrupt for short). When a limit switch is triggered, a customized interrupt service routine will be executed. This function can be used to program the measures to be taken in an emergency.

Channels which do not provide limit interrupt can only be checked as frequently as needed in order to know whether their limit switches are triggered. Those limit switches plus and limit switches minus which are not in use may serve as the over-travel limit switches of other channels. Of course, software must also work in order to correctly determine the source channel of a limit interrupt and the cause of the interrupt (whether the limit switch plus or limit switch minus is triggered). To use the limit interrupt function, the following steps must be completed:

1. Define and declare the customized LIO interrupt service routine. The declaration of a LIO interrupt service routine must be designed following the definitions below:

   ```
   typedef void(_stdcall *LIOISR)(LIO INT*);
   ```

   Therefore, the customized LIO interrupt service routine can be defined as follows:

   ```
   void _stdcall LIO_ISR_Function(LIOINT *pstINTSource)
   {
           //Determine whether a limit interrupt occurs.
           if (pstINTSource-> LDI0)
           {
               /*
               Emergency measures is taken when an over-travel limit is reached.
               */
           }
   ```

```
}
```

The LIO interrupt service routine uses LDI0 ~ LDI6 to determine whether the routine is triggered by a limit interrupt. The meanings of LDI0 ~ LDI6 are as follows:

a. For EPCIO-4000 and EPCIO-4005 control cards.

pstINTSource-> LDI0	OT+ (limit switch plus) of Channel 0
pstINTSource-> LDI1	OT+ of Channel 1
pstINTSource-> LDI2	OT+ of Channel 2
pstINTSource-> LDI3	OT+ of Channel 3
pstINTSource-> LDI4	OT- (limit switch minus) of Channel 0
pstINTSource-> LDI5	OT- of Channel 1
pstINTSource-> LDI6	OT- of Channel 2

b. For EPCIO-6000 and EPCIO-6005 control cards.

pstINTSource-> LDI0	OT+ (limit switch plus) of Channel 0
pstINTSource-> LDI1	OT+ of Channel 1
pstINTSource-> LDI2	OT+ of Channel 2
pstINTSource-> LDI3	OT+ of Channel 3
pstINTSource-> LDI4	OT+ of Channel 4
pstINTSource-> LDI5	OT+ of Channel 5
pstINTSource-> LDI6	OT- (limit switch minus) of Channel 0

2. Add the customized LIO interrupt service routine.
   The LIO interrupt service routine must be added during initialization of the motion control card to be used. Taking the initialization of EPCIO-6000 for example, the function pointer of the interrupt service routine should be input into EPCIO6000_Init() as follows:

```
EPCIO6000_Init(     NULL, NULL, NULL,
                             NULL, NULL, NULL,
                             NULL, LIO_ISR_Function, NULL, 0);
```

3. Use EPCIO_LIO_SetLDIIntType() to set the interrupt trigger type of LDI 0 ~ LDI 6 to rising edge trigger, falling edge trigger, or level change trigger.

4. Use EPCIO_LIO_EnableLDIInt() to enable the limit interrupt function.

The following programming code shows how a limit interrupt is used. Please note where the function pointer of LIO_ISR_Function() is input in EPCIO6000_Init().

```
void _stdcall LIO_ISR_Function(LIOINT *pstINTSource)
{
    //Determine whether a limit interrupt occurs.
    if (pstINTSource->LDI0)
    {
        /*
        Emergency measures is taken when an over-travel limit is reached.
        */
    }
}
 .
 .
if (EPCIO6000_Init(      NULL, NULL, NULL, NULL, NULL
                         NULL, NULL, LIO_ISR_Function, NULL, 0))
{
      .
      .
     //Enable the interrupt triggering functions of LDO 0 of Card 0.
    EPCIO_LIO_SetLDIIntType(LIO_LDI0, PSTINTSOURCE_FALL, 0);
    EPCIO_LIO_EnableLDIInt(LIO_LDI0, 0);
    .
    .
}
```

V.3 Timer Interrupt

EPCIO series motion control cards provide a 24-bit timer which can be set by the user. When the timer expires, a timer interrupt will be triggered, and the timer will be restarted. This process will continue until the function is disabled. To use timer interrupt, the following setting steps must be completed:

1. Define and declare the customized timer interrupt service routine. The declaration of a timer interrupt service routine must be designed following the definitions below:

   ```
   typedef void(_stdcall *LIOISR)(LIO INT*);
   ```

   Therefore, the customized timer interrupt service routine can be defined as follows:

```
void _stdcall Timer_ISR_Function(LIOINT *pstINTSource)
{
        //Determine whether a timer interrupt occurs.
        if (pstINTSource->TIMER)
        {
            /*
            The programming code will be executed, when the timer expires.
            */
        }
}
```

The timer interrupt service routine uses pstINTSource->TIMER to determine whether the routine is triggered by a timer interrupt.

2. Add the customized timer interrupt service routine.
   The timer interrupt service routine must be added during initialization of the motion control card to be used. Taking the initialization of EPCIO-6000 for example, the function pointer of the interrupt service routine should be input into EPCIO6000_Init() as follows:

   ```
   EPCIO6000_Init(    NULL, NULL, NULL,
                      NULL, NULL, NULL,
                      NULL, Timer_ISR_Function, NULL, 0);
   ```

3. Use EPCIO_LIO_SetTimer() to set the timer in units of System Clock (25ns).

4. Use EPCIO_LIO_EnableTimerInt() to enable the timer interrupt function.
   ➔ *See also* EPCIO_LIO_DisableTimerInt()

5. Use EPCIO_LIO_EnableTimer() to enable the timer function.
   ➔ *See also* EPCIO_LIO_DisableTimer()

The following programming code shows how to use the timer interrupt function.

```
void _stdcall LIO_ISR_Function(LIOINT *pstINTSource)
{
    //Determine whether a timer interrupt occurs.
    if (pstINTSource->TIMER)
    {
        /*
        The programming code will be executed, when the timer expires.
        */
    }
```

```
}
 .
 .
if (EPCIO6000_Init(     NULL, NULL, NULL, NULL, NULL
                        NULL, NULL, Timer_ISR_Function, NULL, 0))
     {
      .
      .
      //Set the LIO timer of card 0 for 25ns x 1000000 = 25ms.
      EPCIO_LIO_SetTimer(1000000, 0);
     EPCIO_LIO_EnableTimerInt(0);   //Enable the timer interrupt function of card 0.
     EPCIO_LIO_EnableTimer(0);     //Enable the timer function of card 0.
      .
      .
     }
```

V.4 Watchdog

EPCIO series motion control cards provide the watchdog function. After the user has enabled the watchdog function, the time of the watchdog timer must be cleared before the watchdog timer expires (i.e., the watchdog timer's time equals a pre-set value). Otherwise, once the watchdog timer expires, the hardware will be reset. To use the watchdog function, the following steps must be completed:

1. Use EPCIO_LIO_SetTimer() to set the timer in units of System Clock (25ns).

2. Use EPCIO_LIO_SetWDogTimer() to set the watchdog timer comparison value. The watchdog timer comparison value is a 16-bit numerical value using the time setting of the timer as the time base. In other words, if the following programming code is used:

```
EPCIO_LIO_SetTimer(1000000, 0);
```

```
EPCIO_LIO_SetWDogTimer(2000, 0);
```

The Card 0 watchdog timer comparison value is set at (25ns×1000000)×2000 = 50s.

3. Use EPCIO_LIO_SetWDogReset() to set the reset signal duration of the watchdog timer. Hardware Reset will be triggered upon timeout of the watchdog timer. The duration of reset can be programmed with this function, in units of system clock.

4. Use EPCIO_LIO_EnableWDogTimer() to enable the watchdog function.
   ➔ *See also* EPCIO_LIO_DisableWDogTimer()

5. Use EPCIO_LIO_EnableTimer() to enable the timer function.
   ➔ *See also* EPCIO_LIO_DisableTimer()

After the watchdog function is enabled, EPCIO_LIO_RefreshWDogTimer() must be used to clear the count of the watchdog timer before the watchdog timer expires. Once this function is used, the count of the watchdog timer will reset to zero, and the watchdog timer will start timing again. The following example shows how to use the watchdog.

```
//Set the timer of card 0 for 25ns x 1000000 = 25ms (time base).
EPCIO_LIO_SetTimer(1000000, 0);

//Set the watchdog timer comparison value of Card 0 to 2000 × 25ms = 50s.
EPCIO_LIO_SetWDogTimer(2000, 0);

EPCIO_LIO_EnableWDogTimer(0);    //Enable the watchdog function of Card 0.
EPCIO_LIO_EnableTimer(0);        //Enable the timer function of Card 0.
 .
 .
//The count of Card 0 watchdog timer must be cleared before the timer expires.
EPCIO_LIO_RefreshWDogTimer(0);
```

-
-

The user can combine this function with the timer interrupt function. The user will be alerted before the watchdog generates the reset signal and will have to deal with the issue within the timer interrupt service routine.