PCI-3E Manual

PCI Interface Card for Three Incremental Encoders with I/O Port

Revision: 1.01
09/30/2010

Table of Contents

Amendments

Date	Comment(s)	Authors
1/19/06	PCI-3E Manual Revision 1.0	Sup Premvuti / Steve Smith
02/26/08	Fixed typo in PCI-3E_CaptureTimeAndCounts C example	Steve Smith

1 Introduction

1.1 Purpose

The purpose of this manual is to provide aid in understanding how to use the features of the PCI-3E, PCI Interface Card for 3 Incremental Encoders with I/O Port. The features of the PCI-3E card are made accessible by using the functions provided in the USD_PCI_3E.dll.

1.2 Scope

This document shall describe how to use each of the available interface methods provided by the PCI-3E card. The following chapters are included.

- Installation Instructions
- Troubleshooting
- Running Demonstration Program
- Architecture of PCI-3E
- PCI-3E Registers
- Trigger/Capture Feature
- Typical Usage Scenario
- Function Calls
- Using Java
- Input/Output Port Diagram
- Error Codes

2 Installation Instructions

2.1 Windows Operating Sytem

Please follow these five steps to install PCI-3E and its software.

1. Run the Setup.exe

   Note: The installation checks if an old version of windrvr.sys already exists in the ..system32\drivers directory. If an older version is found, a dialog will be displayed which presents the following three options:

   Option 1 - Install windrvr.sys version 5.2.2 and make a backup of the older version and place it in the C:\Program Files\PCI3E\Backup directory.

   Option 2 - Leave the older version installed and copy version 5.2.2 to C:\Program Files\PCI3E directory.

   Option 3 - Cancel the installation.

   The PCI-3E demos will not function properly with an older version of the windrvr.sys file.

   If you are running an application that requires a previous version of the windrvr.sys file, please contact US Digital for support.

   If you need to restore (or run with) the older version simply copy the windrvr.sys from the C:\Program Files\PCI3E\Backup directory to the ..\system32\drivers directory and then reboot.

2. Shutdown the computer and install the PCI-3E card(s).

   2.1 For additional safety, disconnect power from the computer at the main supply.

   2.2 Observe static handling procedures while working with the PCI-3E: wear an approved ground strap, and open the PCI-3E packaging only at a work surface with a grounded anti-static mat.

   2.3 Carefully insert the PCI-3E card into an available PCI slot. You can install one or more PCI-3E cards at the same time.

3. Power-On the computer. The amber LED on the PCI-3E card should be flashing. If the LED is not flashing see Chapter 3, Troubleshooting.

4. The PCI3E_rev01.inf is copied to the WINNT\inf directory and installed by the setup program

so the Found New Hardware Wizard should not appear. If it does appear, follow its instructions; when asked, specify the location of the PCI3E_rev01.inf file. A copy of this file has also been placed in the C:\Program Files\PCI-3E Demo directory.

5. Launch the PCI-3E VB Demo to test all installed PCI-3E cards. (See Chapter 4 Running Demonstration Program) After the PCI-3E VB Demo is running properly with PCI-3E card, LabVIEW users may proceed to install LabVIEW VIs for PCI-3E.

If the demo application fails to load or the device manager indicates that there is a problem with the Jungo\US DIGITAL PCI-3E Firmware Revision 00 device, then ensure the windrvr.sys file is located in the correct directory, especially if installing on a multi-boot system or if the OS is installed in a non-default directory. The windrvr.sys should be located in one of the directories specified for the OS below:

Windows 2000	- C:\WINNT\system32\drivers
Windows XP	- C:\Windows\system32\drivers

Please contact US Digital Customer Support if you have additional questions.

2.2 Linux Operating System

Please follow these steps to install PCI-3E and its software.

1. Shutdown the computer and install the PCI-3E card(s).

    1.1 For additional safety, disconnect power from the computer at the main supply.

    1.2 Observe static handling procedures while working with the PCI-3E: wear an approved ground strap, and open the PCI-3E packaging only at a work surface with a grounded anti-static mat.

    1.3 Carefully insert the PCI-3E card into an available PCI slot. You can install one or more PCI-3E cards at the same time.

2. Power-On the computer. The amber LED on the PCI-3E card should be flashing. If the LED is not flashing see Chapter 3, Trouble Shooting.

3. Create local directory to copy the pci3e-1.0.tar file to, i.e., /home/steve/pci3e and change to the directory.

```
$ mkdir /home/steve/pci3e
$ cd /home/steve/pci3e
```

4. After the pci3e-1.0.tar file has been copied to the local directory. Extract the contents of pci3e-1.0.tar file.

```
$ tar xzvf pci3e-1.0.tar
```

5. Change to root or an equivalent access level.

```
$ su
      password
```

6. The pci3e kernel module driver has only been tested on version 2.6.12-gentoo-r10 of Linux. In order to access the kernel source code, the makefile relies on a symbolic link to the kernel source directory. If one does not exist you may create one using the following command:

Note: the version of linux installed on your system may be different than the one specified in the following command. If you are using version 2.6.12-gentoo-r10 of Linux, then you may skip to step 8.

```
# ln -s /usr/src/linux-2.6.12-gentoo-r10 /usr/src/linux
```

7. Remove previously built object files and recompile the pci3e driver and demo.

```
# make clean
# make
# make demo
```

8. Allow the script files to be executed.

```
# chmod +x load
# chmod +x unload
```

```
# chmod +x pci3e*
```

9. Load the pci3e kernel module.

```
# ./load
```

10. Run the demo.

```
# ./demo  (press any key to terminate the demo)
```

3 Troubleshooting

Symptom:

The amber LED on the PCI-3E card does not come on after the computer is Powered-On.

Problem:

The board is not receiving any power.

The firmware initialization has failed.

Resolution:

Power-Off the computer and insure that the PCI-3E card is installed correctly.

Contact US Digital customer support, if all attempts fail.

Symptom:

The amber LED on the PCI-3E card is on but does not flash after the computer is Powered-On.

Problem:

The firmware initialization has succeeded but the PCI and PCI-3E board is not communicating.

Resolution:

Power-Off the computer and insure that the PCI-3E card is installed correctly.

Contact US Digital customer support, if all attempts fail.

4 Running Demonstration Programs

After PCI-3E hardware and driver software has been successfully installed, you should be able to run accompanied demonstration programs. The demo programs will give you an opportunity to explore features of PCI-3E.

Steps to run the demo programs (written in C, Visual Basic or LabVIEW):

(1) Connect at least one encoder to encoder connectors of PCI-3E.
(2) Launch a demo program.
(3) The demo program will display the number of existing PCI-3E board(s) on the PCI bus and automatically assign unique device numbers for each PCI-3E board. Use an option of the demo program to choose a PCI-3E board you want to access.
(4) On the demo screen, locate a command to set “Cycles Per Revolution”. This number should be set to match the connected encoder. When using the VB Demo, select View\Configuration menu item to display Cycles Per Revolution input text box.
(5) Turn the encoder and see if the number of counts and the graph display match the movement of the encoder’s shaft.
(6) Explore available features of each demo such as changing quadrature mode.
(7) The demo programs also allow you to directly access all 48 registers of PCI-3E. The detailed explanation of architecture of PCI-3E and its registers can be found in the following chapters.

5 Architecture of PCI-3E

5.1 Overview

Memory-mapped registers based system:

PCI-3E interfaces to an application program through 48 memory-mapped 32-bit registers. An application program reads or writes registers to set mode, select functions or get data from PCI-3E.

48 registers are divided into 8 groups (See Figure 5.1)

(1) Channel 0 registers group:	register 0 to register 6
(2) Channel 1 registers group:	register 8 to register 14
(3) Channel 2 registers group:	register 16 to register 22
(4) Miscellaneous registers group:	register 7, 15 and 23
(5) Interrupt registers group:	register 34 and 35
(6) FIFO registers group:	register 37, 38 and 39
(7) Digital input triggering group:	register 27 and 28
(8) Data Logging and I/O group:	register 30, 31, 40 ~ 47

(Note: register 24, 25, 26, 29, 32, 33 and 36 are reserved.)

All three channels are identical in number of registers and operation for each register. Each channel works independently based on its own set of registers.

Miscellaneous registers group consists of Command Register (reg.#7), Time Stamp Latch Register (reg.#15), and 33 MHz Time Stamp Counter (reg.#23).

Interrupt registers group consists of Interrupt Control Register (reg.#34) and Interrupt Status Register(reg.#35).

FIFO registers group consists of FIFO ON/OFF Register (reg.#37), FIFO Status/Control Register (reg.#38), and FIFO Read Register (reg.#39).

For the digital input triggering group, and the data logging and I/O group, see 6.6 and 6.7 for details of registers in these groups.

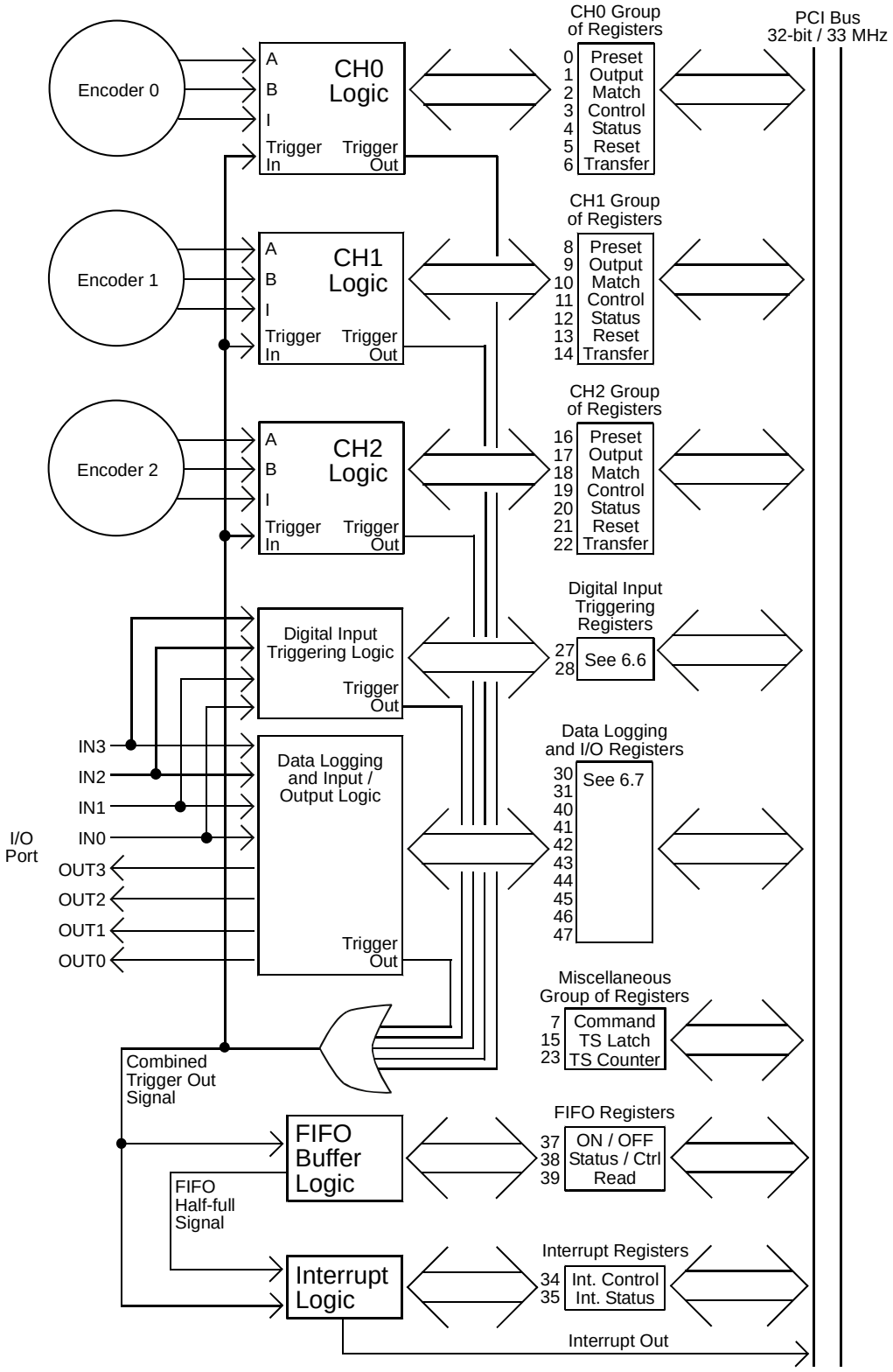


Figure 5.1 Block Diagram of PCI-3E

5.2 Principle of Operation for a Channel

The heart of each channel is a 24 bit up-down counter. (See Figure 5.2) It receives signals commanding it to count up or down from a state machine that watches the quadrature signals coming in. When the state machine sees the quadrature advance, it issues a pulse to increment the counter. When it sees the quadrature retard (move backward) it issues a pulse to decrement the counter. The various input modes such as X1, X2, X4 and pulse/direction mode are implemented via the input state machine.

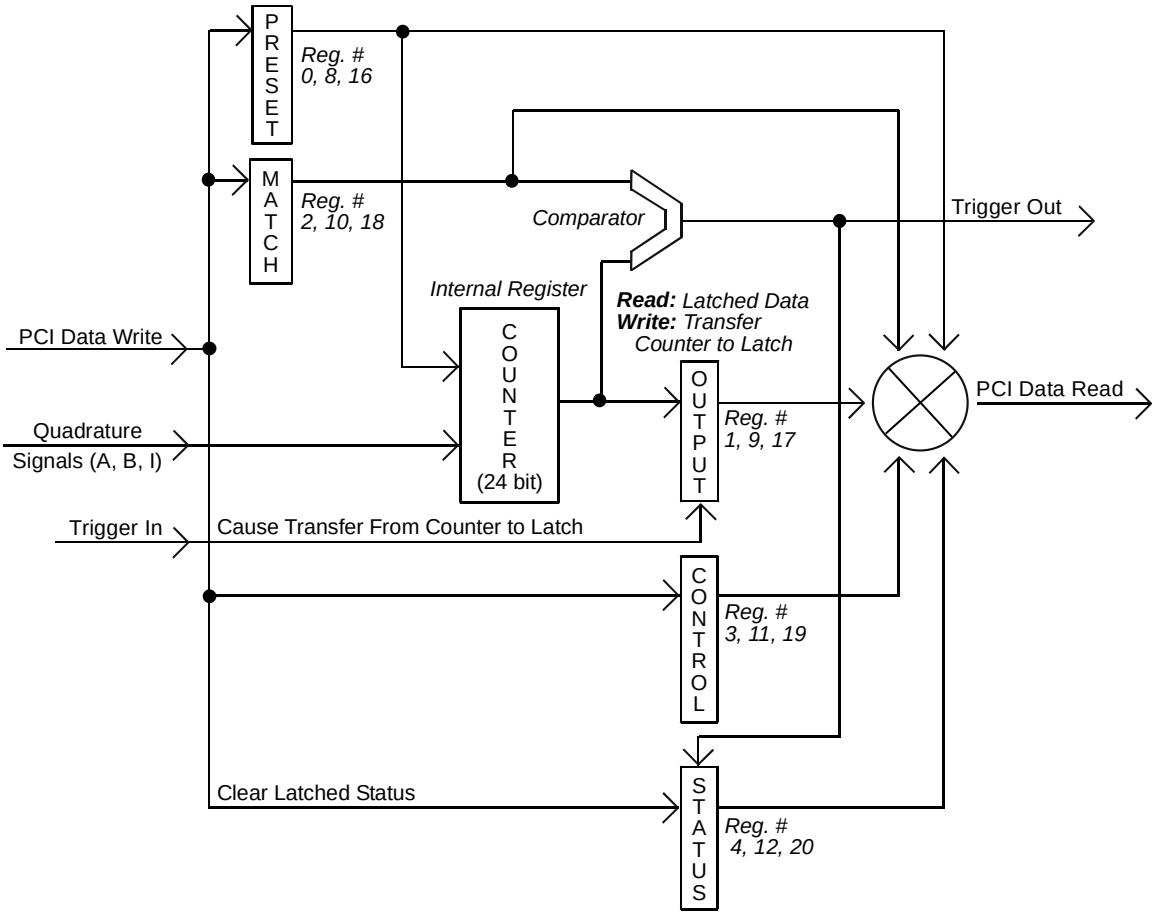
The output of the counter is made available to the host PCI bus through a latching register called the output latch. This was done so as to provide a way to capture and hold a snapshot of encoder position in hardware without requiring immediate software attention. There are two ways to transfer the counter value to the output latch: the host software can write (any value) to the output latch address, or the host can set up a triggering event that will use dedicated hardware to recognize a condition that will capture the counter value. Triggering is explained in detail in a later paragraph.

The counter is capable of counting in special modes that roll over from value N to value 0, or stop counting when a limit is reached. It does this with the help of a preset register, which defines the upper limit of the counting process. The value of the counter is continuously compared to the preset register, and in these special counting modes the counter is either disabled or reset or preset.

A separate match register is provided, to allow for comparisons against an arbitrary value even while the preset register is being used to implement a limited-range counting mode. The result of a match can be used to generate a trigger that will cause transfer of the counter value to the output latch on this channel and/or other channels simultaneously.

A control register is provided, to allow the various counting modes and input modes to be specified. A status register is also available to report on various conditions existing within the channel; some conditions are latched, and persist until cleared explicity by writing 0xFFFFFFFF to the status register.

The triggering capability allows the host to specify conditions that will cause a capture of counter values on multiple channels. The conditions include advance of quadrature, retard of quadrature, passing through zero, encountering an index, reaching a value that corresponds to the match register, carry condition, or borrow condition. The specified condition may be sensed on any channel, and sent out of the channel to a higher level logic block, where it is OR'ed with the triggers from other channels. (See Figure 5.1) The resulting "combined trigger" then re-enters all of the channels; a channel may be enabled to respond to this event by transferring the counter contents to the output latch.



Abbreviation of register names

Symbol	Name
COUNTER	Internal counter register
CONTROL	Control register
STATUS	Status register
MATCH	Match register
PRESET	Preset register
OUTPUT	Output latch register

Figure 5.2 Block Diagram of Channel

5.3 Minimum Programming for a Channel

Once the installation has been done successfully, all PCI-3E boards in a PC are ready to be accessed through provided function calls. After understanding features of 6 registers in a Channel Group, advanced users can operate PCI-3E with just five basic function calls.

PCI Initializing

9.4.43	PCI3E_Initialize
9.4.2	PCI3E_CardCount
9.4.81	PCI3E_Shutdown

Read/Write Registers

9.4.49	PCI3E_ReadRegister
9.4.85	PCI3E_WriteRegister

In addition to 5 basic function calls, “User friendly functions” are also provided in order to facilitate writing application programs with better readability. Names of user friendly functions refer directly to their functions or features. (See Chapter 9 “Function Calls” for details.) Each function call will be translated into reading, writing or combinations of reading and writing one or more of 48 registers of PCI-3E.

User friendly functions are used in the following example of a minimum program. Register numbers accessed by function calls are also provided as references.

A minimum program in C consists of four sections.
(Register numbers shown in this section are based on Channel 0. For accessing other channels, please refer to Chapter 6 PCI-3E Registers.)

1. Initialize PCI-3E cards
2. Configure counter
    2.1 Select value of Preset register (reg.#0)
2.2 Select value of Control register (reg.#3)
    - quadrature mode
    - count mode
    - direction of count (up/down)
    - master enable
3. Get count from Output Latch register (reg.#1)
4. Close PCI-3E

Description:

(1) Initialize PCI-3E cards and get number of total PCI-3E cards on PCI bus.

Use this function:

```
PCI3E_Initialize(short *piDeviceCount);
```

(2) Select value of Preset Register (reg.#0)

If you plan to select the following counter modes; Range-limit mode, Non-recycle mode, or Modulo-N mode (See 6.1 Control Registers); the preset register must be set to your desired value.

Use this function:

```
PCI3E_SetPresetValue(short iDeviceNo, short iEncoder, long lVal);
```

(3) Select quadrature mode in Control Register (reg.#3)

Bit 15 and 14 determine how the accumulator increments: These bits may be referred to as either quadrature mode or multiplier.

Mode	bit15, bit14	Description
0	00	A quadrature input = Clock B quadrature input = Direction Each rising edge of A input causes a counter increment or decrement, depending on the level of B input.
1	01	Accumulator increments once for every for four quadrature states (X1).
2	10	Accumulator increments once for every for two quadrature states (X2)
3	11	Accumulator increments once for every for quadrature state (X4)

Use this function:

```
PCI3E_SetMultiplier(short iDeviceNo, short iEncoder, short iMode);
```

(4) Select count mode in Control Register (reg.#3)

Bit 17 and 16 determine mode of internal counter.

Mode	bit17, bit16	Description
0	00	Simple 24-bit counter mode
1	01	Range-limit mode
2	10	Non-recycle mode
3	11	Modulo-N mode

Use this function:

```
PCI3E_SetCounterMode(short iDeviceNo, short iEncoder, short iMode);
```

(5) Set direction bit (swap quadrature A/B bit) in Control Register (reg.#3)

Bit 19 of Control Register controls the direction of count (up/down)

“0” Quadrature signals A and B are treated normally in a channel’s internal logic.
“1” Quadrature signals A and B are swapped in a channel’s internal logic.
As the result, the direction of count (up/down) will be reversed when bit 19 changes value.

Use this function:

```
PCI3E_SetForward(short iDeviceNo, short iEncoder, BOOL bVal);
```

Note that PCI3E_SetForward function sets bit 19 of Control register when its parameter, bVal, is ‘1’.

(6) Set master enable bit in Control Register (reg.#3)

Set bit 18 to ‘1’ to enable accumulator.

Use this function:

```
PCI3E_SetEnableAccumulator(short iDeviceNo, short iEncoder, BOOL bVal);
```

(7) Get count data from Output Latch Register (reg.#1)

The Output Latch Register is used to latch the count value from the internal counter register for reading by an application program. It is important to understand that the Output Latch Register will be updated ONLY after a WRITE action to the Output Latch Register (data is irrelevant). This means an application can read the Output Latch Register at any time. But its value will be updated to current count value only after it has been written.

To accommodate users who want to write a simple program that retrieves encoder counts, PCI3E_GetCount function is provided. When using this function, please be aware that write to and read from Output Latch Register are performed consecutively in one call of PCI3E_GetCount.

Use this function:

```
PCI3E_GetCount(short iDeviceNo, short iEncoder, long *plVal);
```

(8) Close PCI-3E before exiting application

The PCI3E_Shutdown function must be call in order to disconnect from the PCI3E driver.

Use this function:

```
PCI3E_Shutdown();
```

A minimum program in C

```
// C Hello World.cpp: Defines the entry point for the console application.
//

#include <conio.h>
#include "stdio.h"
#include "windows.h"
#include "..\PCI_3e.h"

int main(int argc, char* argv[])
{
    short devicecount = 0;
    short iResult = 0;
    unsigned long ctrlmode = 0;
    unsigned long ulCount;
    unsigned long ulPrevCount;

    printf("--------------------------------\n");
    printf("PCI-3E Hello World!\n");
    printf("--------------------------------\n");

    // Initialize the PCI-3E driver.
    iResult = PCI3E_Initialize(&devicecount);         // initialize the card

       // Check result code...
       if (iResult != S_OK) {
printf("Failed to initialize PCI-3E driver!  Result code = %d.\nPress any key to exit.\n",
iResult);
               while( !_kbhit() )
                      Sleep(100);

               goto done;
       }

    // Caution! The reset of the example is implemented without any error checking.

    // Configure encoder channel 0.
    PCI3E_SetPresetValue(0,0,499);          // Set the preset register to the CPR-1
    PCI3E_SetMultiplier(0,0,3);             // Set quadrature mode to X4.
    PCI3E_SetCounterMode(0,0,3);            // Set counter mode to modulo-N.
    PCI3E_SetForward(0,0,TRUE);             // Optional: determines the direction of counting.
    PCI3E_SetEnableAccumulator(0,0,TRUE);   // Enable the counter. **IMPORTANT**

    // PCI3E_SetControlMode(0,0,0xFC000);   // You may replace the previous five lines with
                                            // one call to PCI3E_SetControlMode using to correct
                                            // control mode value.

    printf("Reading encoder channel 0. Press any key to exit.\n");
    // Waits for the user to press any key, then exits.
    while( !_kbhit() ){
        PCI3E_GetCount(0,0,&ulCount);
        // Update display when value changes
        if (ulPrevCount != ulCount)
            printf("%d              \r", ulCount);
        ulPrevCount = ulCount;
        Sleep(1); // Don't want to hog all the CPU.
    }

done:
    // Close all open connections to the PCI3E devices.
    PCI3E_Shutdown();

    return 0;

}
```

6 PCI-3E Registers

The PCI-3E board has 48 32-bit registers which are divided into the following groups

(1) Encoder Channel 0 Group: 7 registers
(2) Encoder Channel 1 Group: 7 registers
(3) Encoder Channel 2 Group: 7 registers
(4) Miscellaneous Group: 3 registers
(5) Interrupt Group: 2 registers
(6) FIFO Group: 3 registers
(7) Digital Input Triggering Group: 2 registers
(8) Data Logging and Input/Output Group: 10 registers

Note: 7 registers are reserved.

Register name	Description	Register number		
		Encoder Channel 0 Group	Encoder Channel 1 Group	Encoder Channel 2 Group
Preset	Sets roll-over value for Modulo-N mode, and upper limit for non-recycle and range limit modes.	0	8	16
Output Latch	Counter contents are captured here by command from control register or by trigger capture. Writes cause capture of counter, reads return contents of output latch.	1	9	17
Match	Used as a reference to capture trigger when counter equals match register.	2	10	18
Control (See 6.1)	Contains bits that control the counting mode, quadrature mode, and other aspects of channel's operation.	3	11	19
Status (See 6.2)	Contains bits that describe the state of the counter, trigger system when read. Writing 1 to the bit that is set will clear that bit. Writing 0xFFFFFFFF to status register clears all status bits.	4	12	20
Reset	Reading returns the current counter value. Writing any value to this address causes the channel's counter to be reset to zero.	5	13	21
Transfer Preset	Writing any value to this address causes the counter to be set to the contents of the channel's preset register. (Write only register)	6	14	22

Miscellaneous Group (See 6.3)		
Register name	Register number	Description
Command	7	Command
TS Latch	15	Time Stamp Latch
TS Counter	23	33 MHz Time Stamp Counter

Interrupt Group (See 6.4)		
Register name	Register number	Description
Interrupt Control	34	Enable interrupt
Interrupt Status	35	Interrupt status

FIFO Group (See 6.5)		
Register name	Register number	Description
FIFO ON/OFF	37	Enable/Disable FIFO
FIFO Status/Control	38	Provides status of FIFO
FIFO Read	39	Returns an entry of the FIFO

Digital Input Triggering Group (See 6.6)		
Register name	Register number	Description
Digital Input Trigger Control	27	Select trigger input bits and their active level (active high or active low)
Digital Input Trigger Status	28	Hold the latched value of detected triggers.

Data Logging and Input/Output Group (See 6.7)		
Register name	Register number	Description
Sampling rate multiplier	30	Holds a 32-bit sampling rate multiplier, “N”.
Sampling rate counter	31	Sampling rate counter. Each count is equal to (“N”+1) times 30 microseconds.
Input Port	40	Input Port
Trigger Setup	41	Trigger Setup
Qualifier Setup	42	Qualifier Setup
Number of samples to collect	43	Number of samples to collect
Number of samples remaining to be collected	44	Number of samples remaining to be collected
Acquisition Control Register	45	This register is used to start/stop acquisition. It also indicates the acquisition status.
Output Port	46	Output Port
Output Port Setup	47	Output Port Setup

Note: register 24, 25, 26, 29, 32, 33 and 36 are reserved.

6.1 Control Registers (reg.#3, reg.#11, reg.#19)

32-bit register:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0																		0	0	0	0	0	0	0

'0' indicates unused or reserved bit which always returns '0' when being read.

Bit	Group name	Bit name	Default value after reset
31-24	**UNUSED**	31: unused31	
		30: unused30	
		29: unused29	
		28: unused28	
		27: unused27	
		26: unused26	
		25: unused25	
		24: unused24	
23	**CAPTURE**	23: ctrl_enable_capture	0
22-20	**INDEX**	22: ctrl_index_preset_not_reset	0
		21: ctrl_invert_index	0
		20: ctrl_enable_index	0
19	**COUNT DIRECTION**	19: ctrl_count_direction	0
18	**MASTER ENABLE**	18: ctrl_enable	0
17-16	**COUNT MODE**	17: count_mode1	0
		16: count_mode0	0
15-14	**QUAD MODE**	15: quad_mode1	0
		14: quad mode0	0
13-7	**TRIGGER**	13: ctrl_trigger_retard	0
		12: ctrl_trigger_advance	0
		11: ctrl_trigger_index	0
		10: ctrl_trigger_borrow	0
		9: ctrl_trigger_carry	0
		8: ctrl_trigger_match	0
		7: ctrl_trigger_zero	0
6-0	**RESERVED**	6: reserved6	
		5: reserved5	
		4: reserved4	
		3: reserved3	
		2: reserved2	
		1: reserved1	
		0: reserved0	

Description

Registers 3, 11, 19, 27 are used to hold a 24-bit value that determines the operation of their respective channel. The following table defines what each of 24 bits control.

Bit	Description of Control
31-24	Unused.
23	Allow trigger_in to cause transfer from accumulator to output latch register.
22	When set and index is enabled, causes preset; otherwise if this bit is low a reset will occur when index is true.
21	Set for active low index, (invert index); leave reset for active high index.
20	When set, an index event will either reset or preset accumulator.
19	Determines the count direction. See the table below.

Bit 19	Quadrature Mode (depends on bit 15 and 14)	Conditions of "A" input and "B" input → Count Direction
'0'	X1, X2, X4	"A" leads "B" → **UP** "B" leads "A" → **DOWN**
'1'	X1, X2, X4	"A" leads "B" → **DOWN** "B" leads "A" → **UP**
'0'	Clock and Direction	"A" = clock, "B" = '1' → **UP** "A" = clock, "B" = '0' → **DOWN**
'1'	Clock and Direction	"A" = clock, "B" = '1' → **DOWN** "A" = clock, "B" = '0' → **UP**

18	Master enable for accumulator.
17, 16	Governs the behavior of the internal counter at limits: • '00' accumulator acts like a simple 24 bit counter – counts from 0 up to 16,777,215 and then rolls over to 0 again and resumes counting upward; counting downwards, the counter goes from 0 to 16,777,215 and continues downwards. • '01' accumulator uses preset register in range-limit mode – when count reaches 0 or the preset value the counter freezes until the inputs cause a change in direction that keeps the counter within the bounds of 0 and preset value. • '10' accumulator uses preset register in non-recycle mode – when count reaches 0 going down or the preset value going upwards, the counter is frozen until a channel reset is performed. • '11' accumulator uses preset register in modulo-N mode – the counter counts upward until it matches the preset value, then rolls over to 0, and resumes counting upwards; when counting downward the counter rolls under from 0 to the preset value, and counts downward from there.
15, 14	Determines how the accumulator increments: This bit may be referred to as either quadrature mode or multiplier.

- '00' A input = clock and B input = direction. Each rising edge of A input causes a counter increment or decrement, depending on the level of B input.
- '01' accumulator increments once for every for four quadrature states (X1)
- '10' accumulator increments once for every for two quadrature states (X2)
- '11' accumulator increments once for every for quadrature state (X4)

13 Trigger signal is generated when accumulator decreases or retards.
12 Trigger signal is generated when accumulator increases or advances.
11 Trigger signal is generated when edge of index occurs.
10 Trigger signal is generated when rolling under 0 to N-1 in modulo-N mode.
9 Trigger signal is generated when rolling over N-1 to 0 in modulo-N mode.
8 Trigger signal is generated when accumulator equals match register.
7 Trigger signal is generated when accumulator equals zero.
5 - 0 Reserved for future use.

6.2 Status Registers (reg.#4, reg.#12, reg.#20)

32-bit register:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0		0	0														0	0	0	0	0	0	0

‘0’ indicates unused or reserved bit which always returns ‘0’ when being read.

Bit	Group name	Bit name
31-24	**UNUSED**	31: unused31
		30: unused30
		29: unused29
		28: unused28
		27: unused27
		26: unused26
		25: unused25
		24: unused24
23	**LAST DIRECTION INDICATOR**	23: last_direction
22-21	**RESERVED**	22: reserved22
		21: reserved21
20-14	**EVENT DETECTED**	20: retard_detected
		19: advance_detected
		18: index_detected
		17: borrow_detected
		16: carry_detected
		15: match_detected
		14: zero_detected
13-7	**LATCHED EVENT DETECTED**	13: latched_retard_detected
		12: latched_advance_detected
		11: latched_index_detected
		10: latched_borrow_detected
		9: latched_carry_detected
		8: latched_match_detected
		7: latched_zero_detected
6-0	**RESERVED**	6: reserved6
		5: reserved5
		4: reserved4
		3: reserved3
		2: reserved2
		1: reserved1
		0: reserved0

Description

The following defines bits of the status registers:

Bit	Status
31-24	Unused
23	Indicates the last counting direction
22-21	Reserved for future use
20	Retard of quadrature is detected
19	Advance of quadrature is detected
18	Index is detected
17	Borrow is detected
16	Carry is detected
15	Internal counter reaching a value that corresponds to the Match register.
14	Internal counter passing through zero.
13	Latched value of bit 20
12	Latched value of bit 19
11	Latched value of bit 18
10	Latched value of bit 17
9	Latched value of bit 16
8	Latched value of bit 15
7	Latched value of bit 14
6-0	Reserved for future use.

Writing 1 to the Status bit that is set will clear that bit.
Writing 0xFFFFFFFF to a Status register will clear all of its status bits to 0.

6.3 Miscellaneous Registers (reg.#7, reg.#15, reg.#23)

This section describes the features associated with each of the following registers:

- **Command (Register #7) : Command Register**

Bit	Description
31-24	ROM identification (ROM_ID). These bits are read only and their values may vary between boards.
23-7	Reserved.
6	This bit is used to run or stop 33 MHz Time Stamp Counter (TS Counter). 0: Time Stamp counter is running. 1: Time Stamp counter stops at count 0.
5	Bit 5 is used to cause the TS Counter to transfer to the Time Stamp Latch Register (TS Latch). Transition from 0 to 1 of this bit (write '0' then write '1') will generate a one-shot pulse that causes the transfer. (See Figure 7.1 Trigger and Time Stamp Circuit.)
4	Bit 4 is used to initiate a software-capture-all which causes the TS Counter to transfer to the TS Latch and then causes all accumulators with captured enabled to transfer to the Output Latch. Transition from 0 to 1 of this bit (write '0' then write '1') will generate a one-shot pulse that causes all the described transfers. (See Figure 7.1 Trigger and Time Stamp Circuit.)
3-0	Reserved

- **TS Latch (Register #15) : Time Stamp Latch Register**
  This register is used to hold the Time Stamp Output Latch value. When a trigger event occurs, the value of TS Counter will be transferred to TS Latch.
  Read: Return the 32-bit time stamp latched value
  Write (any value): Initiate a software-capture-all which causes the TS Counter to transfer to the TS Latch and then causes all accumulators with captured enabled to transfer to the Output Latch. It is equivalent to writing 0 then writing 1 to bit 4 of the Command register.

- **TS Counter (Register #23) : 33 MHz Time Stamp Counter**
  This register is used to hold a 32-bit counter running at 33 MHz.
  Read: Return the current counter value
  Write (any value): Initiate the transfer of the TS Counter to the Time Stamp Latch Register (TS Latch). It is equivalent to writing 0 then writing 1 to bit 5 of the Command register.
  Note: This counter is based on the PCI bus's 33.3333 MHz clock. One count is approximately equivalent to 30 nano-seconds. The actual clock frequency of each PC may vary.

6.4 Interrupt Registers (reg.#34, reg.#35)

- **Interrupt Control (Register #34)** (Read/Write)
  Bit 31: Enable interrupt caused by FIFO half-full
  Bit 30: Enable interrupt caused by trigger out signal

  This register controls the Interrupt Logic to accept or ignore two input signals, "Encoder Trigger Out" signal and "FIFO Half-full" signal. See **Figure 5.1 Block Diagram of PCI-3E.** The Interrupt Logic generates "Interrupt Out" signal to the PCI bus and asserts bit 31 of the Interrupt Status register when it detects a signal on the enabled inputs.

  Important note: A new "Trigger Out" signal cannot be generated from an encoder channel until its pending trigger status is cleared. Writing 0xFFFFFFFF to all status registers **right after** enabling the interrupts will clear any pending trigger status.

  When either bit 30 or bit 31 is set using the PCI3E_WriteRegister function, the Status register for each encoder must also be cleared.

  However, when using **PCI3E_SetInterruptControl**, the clearing of status registers is included in the function call. No additional action is needed to clear the status registers.

- **Interrupt Status (Register #35)** (Read/Write)
  Bit 31: Interrupt occurred / Write '1' to clear
  Bit 30: FIFO half-full status bit

  After an interrupt occurred, bit 31 must be cleared to lower the "Interrupt Out" signal on the PCI bus.

  FIFO half-full status bit is a copy of bit 19 of the FIFO Status/Control (reg.#38). This bit can be used to determine the cause of interrupt if either bit 30 and bit 31 of the Interrupt Control (reg.#34) is set.

6.5 FIFO Registers (reg.#37, reg.#38, reg.#39)

- **FIFO ON/OFF (Register #37)** (Read/Write)

  Bit 8: FIFO on-off
  1- ON: Using FIFO
  0- OFF: Not using FIFO

  When the FIFO is turned on, all encoder counts with the time stamp will be stored in the FIFO for every "Encoder Trigger Out" signal. See **Figure 5.1 Block Diagram of PCI-3E** for the block diagram of the FIFO logic. The "Encoder Trigger Out" signal is tapped from the output of the OR gate whose inputs are from "trigger out" of all channels. See also **7 Trigger/Capture Feature**.

  The size of the FIFO is 1024 x 32-bit. This FIFO can store 204 records of encoder data. Each record consists of 4 encoder counts with a 32-bit timestamp.

  Important note: A new "Trigger Out" signal cannot be generated from an encoder channel until its pending trigger status is cleared. Writing 0xFFFFFFFF to all status registers **right after** the FIFO is turned on will clear any pending trigger status. When a new "Encoder Trigger Out" signal is detected, the FIFO logic will store a new record in the FIFO and clear the status registers automatically.

  When bit 8 is set using the PCI3E_WriteRegister function, the status register for each encoder must also be cleared.

  However, when using **PCI3E_EnableFIFOBuffer**, the clearing of status registers is included in the function call. No additional action is needed to clear the status registers.

  The FIFO can be used with polling method (checking the FIFO Status/Control (reg.#38) then writing/reading the FIFO Read (reg.#39)) or interrupt driven method.

  Polling method:
  (1) Check the FIFO Status/Control (reg.#38) for the presence of data.
  (2) Read the FIFO Read (reg.#39).
  (3) Write the FIFO Read (reg.#39) with any value to update the FIFO Read for the next entry.
  (4) Repeat (2) and (3) until the FIFO is empty.

  For the interrupt driven method, please refer to the following sections:
  6.4 Interrupt Registers
  8 Example Programs
  9.3 User friendly functions / Interrupt Handling Group.

- **FIFO Status/Control (Register #38)** (Read/Write)
  Bit 20 ... bit 10: data_count(10..0) --status (read only) range: 00000000000 ~ 10000000000
  Bit 9: empty --status (read only)
  Bit 8: full --status (read only)
  Bit 1: init (clear FIFO) --command (read/write)
  1- reset
  0- ready
  Bit 0: Update the FIFO Read register to hold the next entry by writing 0 then writing 1 to this bit
  --command (read/write)

  When the FIFO is full (indicated by bit 8), no additional records will be stored. The application program should do the following steps.

  (1) Turn the FIFO off
  (2) Read all records or clear the FIFO.

  Caution: After the FIFO is full, do not read the FIFO (using reg.#39) without turning OFF the FIFO. The FIFO reading will clear up some spaces in the FIFO but the order of entries inside a new record cannot be maintained. (See "Order in the FIFO" below)

- **FIFO Read (Register #39)** (Read/Write)
  Read --- Returns an entry in the FIFO
  Write any value --- Updates this register to hold the next entry in the FIFO (Equivalent to writing 0 then writing 1 to bit 0 of FIFO Status/Control (Register #38))

- **Order in the FIFO**
  Each record contains five 32-bit data as follows.

FIFO OUT (reading from FIFO by the application program)

record i	Time	← entry number 0
	CH0 encoder data	← entry number 1
	CH1 encoder data	← entry number 2
	CH2 encoder data	← entry number 3
	4-bit input port data	← entry number 4
record i+1	Time	← entry number 5
	CH0 encoder data	← entry number 6
	CH1 encoder data	← entry number 7
	CH2 encoder data	← entry number 8
	4-bit input port data	← entry number 9
……	……	……

FIFO IN (writing to FIFO by the firmware)

Format of time entry
Bit 31~bit0: time ----------from TS Latch register (Register #15)

Format of encoder data entry
Bit 31: last direction----------from bit 23 of Status reg.
Bit 30: latched_retard_detected----------from bit 13 of Status reg.
Bit 29: latched_advance_detected--------from bit 12 of Status reg.
Bit 28: latched_index_detected----------from bit 11 of Status reg.
Bit 27: latched_borrow_detected---------from bit 10 of Status reg.
Bit 26: latched_carry_detected----------from bit 9 of Status reg.
Bit 25: latched_match_detected----------from bit 8 of Status reg.
Bit 24: latched_zero_detected----------from bit 7 of Status reg.
Bit 23~bit0: Encoder count----------from bit 23 ~ bit 0 of Output Latch

Format of 4-bit input port data entry
Bit 31~4: 0x0000000
Bit 3: Input port bit 3----------from bit 3 of Input Port (Register #40)
Bit 2: Input port bit 2----------from bit 2 of Input Port (Register #40)
Bit 1: Input port bit 1---------- from bit 1 of Input Port (Register #40)
Bit 0: Input port bit 0---------- from bit 0 of Input Port (Register #40)

6.6 Digital Input Triggering Registers (reg.#27and reg.#28)

- **Triggering Control (Register #27)** (Read/Write)
  Bit 31 ~ Bit 8: reserved
  Bit7: ctrl_trigger_at_rising_not_falling_edge_in3 (0 = falling edge, 1 = rising edge)
  Bit6: ctrl_trigger_at_rising_not_falling_edge_in2 (0 = falling edge, 1 = rising edge)
  Bit5: ctrl_trigger_at_rising_not_falling_edge_in1 (0 = falling edge, 1 = rising edge)
  Bit4: ctrl_trigger_at_rising_not_falling_edge_in0 (0 = falling edge, 1 = rising edge)
  Bit 3: ctrl_trigger_in3
  Bit 2: ctrl_trigger_in2
  Bit 1: ctrl_trigger_in1
  Bit 0: ctrl_trigger_in0

  ctrl_invert_inX (X = 0, 1, 2, 3)
  0: Asserted state is defined as the transition of inX input from logic high to logic low.
  1: Asserted state is defined as the transition of inX input from logic low to logic high.

  ctrl_trigger_inX (X = 0, 1, 2, 3)
  0: Disable trigger event detection.
  1: Enable trigger event detection. Trigger event occurs when the inX input entering asserted state.

- **Triggering Status (Register #28)** (Read/Write)
  Bit 31 ~ Bit 4: reserved
  Bit 3: latched_in3_detected
  Bit 2: latched_in2_detected
  Bit 1: latched_in1_detected
  Bit 0: latched_in0_detected

Bit 0, 1, 2, 3	Read:	0: No enabled trigger event detected. 1: Enabled trigger event detected.
Bit 0, 1, 2, 3	Write:	0: No effect. 1: Clear the detected event of that bit, if any.

  When an enabled trigger event is detected, its “latched_inX_detected” bit will be set to ‘1’.
  Software must clear the detected event by writing ‘1’ to its “latched_inX_detected” (X=0, 1, 2, 3), otherwise this bit cannot detect a new event.

  When the FIFO is ON, these triggering status bits will be cleared automatically.

6.7 Data Logging and Input/Output Registers (reg.#30, reg.#31, reg.#40 ~ reg.#47)

- **Sampling rate multiplier (Register #30)** (Read/Write)
  Holds a 32-bit sampling rate multiplier, "N". See 7.3 Data Logging for details.

- **Sampling rate counter (Register #31)** (Read/Write)
  Read: 32- bit sampling rate counter. Each count is equal to ("N"+1) times 30 microseconds.
  Write (any value): Reset count to '0' when the acquisition is not in progress.

- **Input Port (Register #40)** (Read only) – Active low convention
  Bit 31 ~ Bit 4: reserved
  Bit 3: Input Port – IN3
  Bit 2: Input Port – IN2
  Bit 1: Input Port – IN1
  Bit 0: Input Port – IN0

  0 = Logic HIGH at an input pin. (No connection or the input level is HIGH)
  1 = Logic LOW at an input pin. (Connected to ground or the input level is LOW)

  When nothing is connected, the voltage level at an input pin is pulled up to HIGH by a resistor.

- **Trigger Setup (Register #41)** (Read/Write)
  Bit 31: (r/w) 1 --- AND select, 0--- OR select
  Bit 30 ~ Bit 12: reserved
  Bit 11 ~ Bit 9: (r/w) Trigger setup code for IN3 see **NOTE
  Bit 8 ~ Bit 6: (r/w) Trigger setup code for IN2 see **NOTE
  Bit 5 ~ Bit 3: (r/w) Trigger setup code for IN1 see **NOTE
  Bit 2 ~ Bit 0: (r/w) Trigger setup code for IN0 see **NOTE

- **Qualifier Setup (Register #42)** (Read/Write)
  Bit 31: (r/w) 1 --- AND select, 0--- OR select
  Bit 30 ~ Bit 12: reserved
  Bit 11 ~ Bit 9: (r/w) Qualifier setup code for IN3 see **NOTE
  Bit 8 ~ Bit 6: (r/w) Qualifier setup code for IN2 see **NOTE
  Bit 5 ~ Bit 3: (r/w) Qualifier setup code for IN1 see **NOTE
  Bit 2 ~ Bit 0: (r/w) Qualifier setup code for IN0 see **NOTE

  **NOTE: 3-Bit Trigger/Qualifier setup codes
  000: Never. (Ignore)
  001: Rising. Logic level at an input pin is changing from LOW to HIGH.
  010: Falling. Logic level at an input pin is changing from HIGH to LOW.
  011: Change. Logic level at an input pin is changing from its previous value.
  100: High. Logic level at an input pin is HIGH.

101: Logic level at an input pin is LOW.
110: Unconditionally. (Always)
111: Unconditionally. (Always)

- **Number of samples to collect (Register #43)** (Read/Write)
  Number of samples to collect (32-bit)

- **Number of samples remaining to be collected (Register #44)** (Read only)
  Number of samples remaining to be collected (32-bit)

- **Acquisition Control Register (Register #45)** (Read/Write)
  This register is used to start/stop acquisition. It also indicates the acquisition status.

  Bit 31 ~ Bit 3: reserved
  Bit 2: (read only) Continuous mode in progress
  Bit 1: (read only) Acquisition has been triggered
  Bit 0: (r/w) Start|Stop acquisition

- **Output Port (Register #46)** (Read/Write)
  Bit 31 ~ Bit 4: reserved
  Bit 3: Output Port Bit 3
  Bit 2: Output Port Bit 2
  Bit 1: Output Port Bit 1
  Bit 0: Output Port Bit 0

  0 = OFF (The OFF transistor isolates the output pin.)
  1 = ON (The ON transistor connects the output pin to ground.)

- **Output Port Setup (Register #47)** (Read/Write)
  Bit 31 ~ Bit30: (r/w) Set the length of Trigger signal.

bit31	bit30	Length of Trigger signal
0	0	1 mS
0	1	200 µS
1	0	20 µS
1	1	5 µS

  Bit 29 ~ Bit 4: reserved
  Bit 3: 0 --- OUT3 is bit 3 of reg_46
  1 --- OUT3 is Combined Trigger Out signal
  Bit 2: 0 --- OUT2 is bit 2 of reg_46
  1 --- OUT2 is Trigger Out signal from Encoder Channel 2
  Bit 1: 0 --- OUT1 is bit 1 of reg_46
  1 --- OUT1 is Trigger Out signal from Encoder Channel 1
  Bit 0: 0 --- OUT0 is bit 0 of reg_46
  1 --- OUT0 is Trigger Out signal from Encoder Channel 0

7 Trigger / Capture / Data Logging Feature

7.1 Trigger Signal Generated by Encoders

The PCI-3E is organized into encoder channels. Each encoder channel may be programmed to recognize a variety of conditions (such as the counter passing through zero, or the counter being equal to the match register). This recognition of conditions is achieved by setting trigger bits in the control register, (bit 7 to bit 13 of Control register). When a trigger bit for a condition is enabled within a given channel, that triggering channel generates a hardware signal that goes out of the channel; the triggers from all the channels are OR'ed together to form a capture signal called "Combined Trigger Out".

This single capture signal goes into all the channels; each channel may be programmed, (bit 23 of Control register) to accept this signal (the channel is said to have "capture enabled"). When enabled, the capture signal causes a channel to transfer the contents of the accumulator (counter) to the output latch, without any software intervention. Recall that in normal operation the host software must issue a write to the "Output Latch" register to cause the accumulator contents to be copied to the output latch; all that the triggering system provides is a way for this transfer to be done by hardware when a condition is recognized on a triggering channel.

In order to facilitate analyzing or processing of encoder position data that require precise timing information, 33 MHz Time Stamp Counter (TS Counter) and Time Stamp Latch register (TS Latch) are incorporated into the triggering scheme. When a trigger signal is generated, the value of TS Counter is automatically latched to TS Latch register.

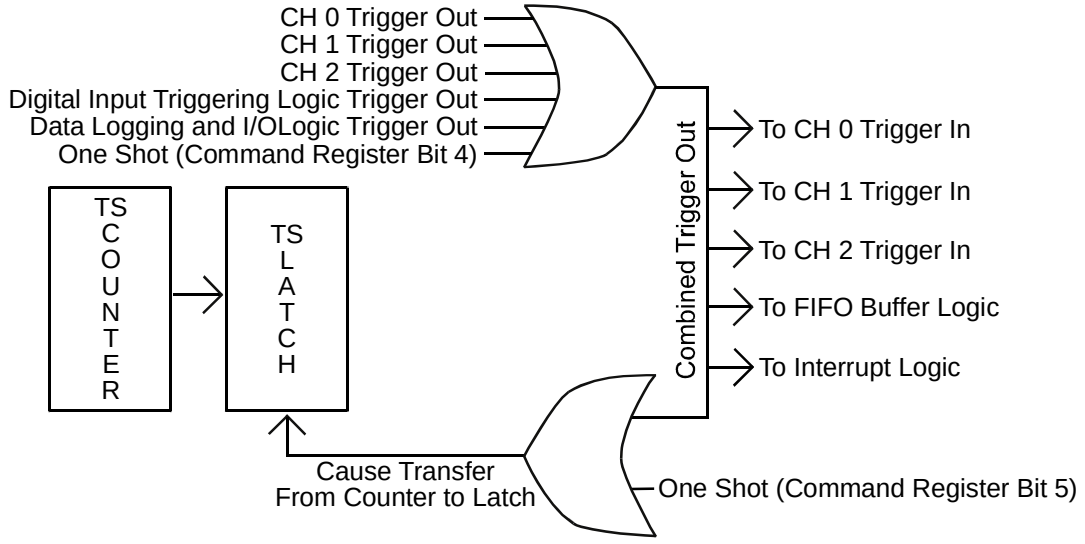


Figure 7.1 Trigger and Time Stamp Circuit

This triggering capability is most useful in the context of a data recording function. In a data recording function the user may desire to have an ordered sequence of samples stored to a disk file. Most often this sequence is generated by storing samples according to a periodic time function, but the triggering function described above allows a sample to be captured as a function of the data itself – when one channel's accumulator reaches zero, for instance, all channels may be programmed to capture the contents of their accumulators. Successive triggers would then capture successive samples.

So, the host software must be used to record a sequence of samples. In the case of periodic sampling, the host would set up a software timer; upon expiration of the timer the host would issue a write to the "Output Latch" register to cause the accumulator contents to be copied to the output latch of all channels, then read the output latch of each of the channels in order, and write the data to a disk file.

In the case of a triggered acquisition sequence (actually, a better term would be storage qualified sequence) the host must set up one or more channels to generate triggering conditions, and then set up one or more channels to respond to the triggers (capture enabled). The host software is responsible for reading the output latch at the right moment, just after a capture has occurred.

The way to accomplish a storage qualified acquisition is for the host software to poll the status register of that channel, looking for a capture detected bit to be set. When a capture detected bit is found to be set, the host knows that a trigger has been generated, and data is waiting to be read in the output latches of the capture-enabled channels. This data should be read and stored to disk file, and then the status should be cleared on the triggering channel by writing 0xFFFFFFFF to the status register. Then the host may go back to polling the status register for captured event activity. Note that when collecting storage-qualified samples the host never issues a write to the "Output Latch" register; the hardware performs that function.

Note: Additional usage of the "Combined Trigger Out" signal

Trigger Out signals from Digital Input Triggering Logic and from Data Logging and Input/Output Logic are also included in the "Combined Trigger Out" signal.

The "Combined Trigger Out" signal also goes to the FIFO buffer logic. The FIFO buffer logic stores the latched encoder counts, input port data and time stamp latch into the on-board FIFO buffers which can hold upto 204 records. (See 6.5 FIFO Registers.)

The Interrupt Logic can be programmed to generate the PCI interrupt signal when the "Combined Trigger Out" is detected. (See 6.4 Interrupt Registers.)

7.2 Trigger Signal Generated by Digital inputs

The PCI-3E may be programmed to generate a trigger signal for latching timestamp and encoder counts when rising or falling edge signals are detected at the 4-bit input port. This recognition of conditions is achieved by setting trigger bits of the Triggering Control Register (reg.#27). The generated trigger signal is OR'ed with other trigger signals from encoder channels. See Figure 5.1.

The "capture enabled" bit (bit 23 of Control registers) of encoder channels must be programmed, to accept the Trigger In signal in order to latch new counts to the output latches.

The FIFO buffer can be used to store the latched encoder counts, input port data and time stamp latch. (See 6.5 FIFO Registers.)

The Interrupt Logic can be programmed to generate the PCI interrupt signal when the "Combined Trigger Out" is detected. (See 6.4 Interrupt Registers.)

This triggering is suitable for collecting encoder data based on external signals.

An example program is provided in the "ConsoleInterruptUsingDigitalInputTrigger" folder. (See 8. Example Programs)

7.3 Data Logging

Overview

The encoder channel processes encoder signals, A, B and I. The encoder channel updates encoder registers at the rate of 33 MHz. The application program can read the encoder registers anytime; however, Windows is not a real-time operating system. Attempting to use the software timer to achieve consistent register read intervals is not a viable approach. To attain consisitent register read intervals, the PCI-3E incorporates a programmable clock with the period of (N+1) * 30 microseconds. The range of "N" is 0 ~ 4294967295. This allows us to select a wide range of sampling period starting from 30 microseconds with 30-microsecond increment upto 128849 seconds. The data logging is synchronized precisely to this programmable clock. The on-board FIFO buffer can store upto 204 samples.

The data logging feature is based on two logic blocks. (See Figure 5.1 Block Diagram of PCI-3E.)

(1) Data Logging and Input/Output Logic.
(2) FIFO Buffer Logic.

The "Data Logging and Input/Output Logic" generates "Trigger Out" signal which is used by the "FIFO Buffer Logic" as a write signal to store data into the FIFO buffer.

Buffering

The PCI-3E has on-board buffering, which allows the unit to buffer 204 samples. Each sample contains four 24-bit encoder counts, four 8-bit status data and a 32-bit time stamp. An interrupt can be generated when the buffer is half-full. See 6.5 FIFO Registers.

Sampling Period

Register 30 is used to set the sampling period for data logging. The sampling period is calculated by the following equation.

Equation: The sampling period = (N+1) * 30 microsecond.
Where, N is the value of the "sampling rate multiplier register" (reg.#30).

Example: How to find N for a 0.01 second sampling period.

Calculation: N = ((0.01 second)*(1000000)/30) - 1 = 333.3333 – 1 = 332.3333

Rounding: Round the result to the nearest integer: N = 332

Verification: The obtained sampling period = (332+1)* 30 microsecond = 0.00999 second.

Note: Since the resolution of sampling period is 30 microseconds, the obtained sampling period may differ from the desired value by +/- 30 microseconds.

Triggering
The PCI-3E may be set up to look for a trigger before storing data via its data logging capability. The trigger is a generalized combination of digital input bit conditions. The combination of digital input conditions may include multiple digital input bits, and the conditions may be of different types (high, low, rising edge, falling edge, either edge, always, or ignore). The digital input combinations may be combined with an OR gate or an AND gate. There are commands to start and stop acquisitions; each new start of the acquisition system resets the trigger to begin looking for a new trigger.

Storage Qualification
Storage qualification is a digital logic analyzer term that describes when samples of data are stored. Like a logic analyzer, the PCI-3E may be set up to constantly scan its inputs for samples that meet a certain criteria, and then store only those samples. Storage qualification applies only after a trigger has occurred. For instance, the user might set the trigger to start data collection when a machine start signal is detected, and then desire to only store data when a certain sensor is active, indicating some portion of the machine's cycle. The data is stored on selected digital input conditions. It is similar to the options available for triggering explained in above “Triggering”.

How to use the data logging feature.

General steps are as follows.

Step 1: Initialize PCI-3E.
Step 2: Set the bit 23 (capture) of control registers to ‘1’. (Other bits of control registers must also be set to your desired settings.)
Step 3: Select the sampling period.
Step 4: Select the number of samples to be collected and the condition for triggering and storage qualification.
Step 5: Clear the FIFO buffer.
Step 6: Turn on the FIFO buffer.
Step 7: Start acquisition.
Step 8: Read data from the FIFO until the specified number of data are collected.
Step 9: Display the collected data

For each step, refer to the following registers or functions.

Step 1: `9.4.43 PCI3E_Initialize`

Step 2: reg.#3, reg.#11, reg.#19
Use `9.4.57 PCI3E_SetControlMode` with appropriate bit selection, or call the following functions.
`9.4.56 PCI3E_SetCaptureEnabled`
`9.4.70 PCI3E_SetPresetValue`
`9.4.59 PCI3E_SetCounterMode`
`9.4.57 PCI3E_SetControlMode`
`9.4.63 PCI3E_SetForward`
`9.4.62 PCI3E_SetEnableIndex`

9.4.61 PCI3E_SetEnableAccumulator

Step 3: reg.#30
9.4.72 PCI3E_SetSamplingRateMultiplier

Step 4: reg.#41, reg.#42, reg.#43
9.4.73 PCI3E_SetTimeBasedLogSettings

Step 5: reg.#38
9.4.5 PCI3E_ClearFIFOBuffer

Step 6: reg.#37
9.4.7 PCI3E_EnableFIFOBuffer

Step 7: reg.#45
9.4.82 PCI3E_StartAcquisition

Step 8: reg.#44, reg.#39
9.4.16 PCI3E_GetFIFOBufferCount
9.4.46 PCI3E_ReadFIFOBufferStruct, or
9.4.45 PCI3E_ReadFIFOBuffer

Step 9: Display the collected data (User defined function.)

The complete C source codes are provided in the “ConsoleTimeBasedDataLogging” folder. (See 8. Example Programs)

8 Example Programs

The following example programs are provided. These programs will be stored at `C:\Program Files\PCI-3E\Source` after running `PCI3ESetup.EXE`. All programs are written in C except VB Demo (Visual Basic).

Source Folder	Description
C Hello World (A minimum program)	This example illustrates how to initialize a PCI-3E card and perform basic configuration. It displays the current encoder counter value to screen as the encoder changes position.
ConsoleFIFOPolling (Using polling method for FIFO reading)	This example initializes a PCI-3E card and polls the FIFO buffer until a predefined number of records (10) have been read. You may modify the demo to read a different number of records.
ConsoleInterruptLogger (PCI-3E Interrupt/FIFO Demo)	This example initializes a PCI-3E card and provides a menu selection whereby a user may enable or disable trigger-out and half-full interrupts. As the user turns an encoder and interrupts are generated the console displays the time, counts for each encoder channel and input port register value.
ConsoleInterruptLogger_EncoderTriggerOut (Using Encoder Trigger-Out Signal Interrupt)	This example initializes a PCI-3E card and automatically registers an interrupt handler that will display the time, counts for each encoder channel and input port register value as interrupts are generated. Interrupts are set to be generated each time an index signal is detected.
ConsoleInterruptLogger_FIFOHalfFull (Using FIFO Half-Full Interrupt)	This example initializes a PCI-3E card and automatically registers an interrupt handler that will display the time, counts for each encoder channel and input port register values after the internal FIFO buffer has reached half full.

ConsoleInterruptUsingDigitalInputTrigger (Using Digital Input Trigger to Generate Interrupt)	This example initializes a PCI-3E card and automatically registers an interrupt handler that will display the time, counts for each encoder channel and input port register value as interrupts are generated. Interrupts are set to be generated each time a falling edge on input 0 of I/O port is detected.
ConsoleTimeBasedDataLogging (Using Time-based Data Logging)	This example illustrates how to use the data logging feature of the PCI-3E card.
VBDemo (Demonstrating features of the PCI-3E)	The PCI-3E VB Demo provides an easy to use graphical interface. The user may configure encoder channels and perform event and time base data logging.

Programming:
You can use any of the supplied source code as part of your own control software, but we make no guarantees on any part of it.

9 Function Calls

User applications may utilize the PCI-3E by calling provided functions in the PCI-3E's Dynamic Link Library (DLL). Function calls are categorized into 3 groups as follows.

- Basic functions
- PCI-3E card information functions
- User friendly functions

9.1 Basic functions (5 functions)

After understanding the meanings and functions of the 6 registers in each Channel Group, advanced users can set up and access PCI-3E with just five basic function calls.

PCI Initalizing (important)

9.4.43 PCI3E_Initialize
9.4.2 PCI3E_CardCount
9.4.81 PCI3E_Shutdown

Registers Read/Write (important) (Access by device number and register number)

9.4.50 PCI3E_ReadRegister
9.4.86 PCI3E_WriteRegister

9.2 PCI-3E card information functions (3 functions)

Three functions are provided for acquiring information related to PCI-3E card.

Functions to get PCI-3E card information (optional).

9.3 User friendly functions (78 functions)

To facilitate programming with high readability, user friendly functions named with their features have been provided. Please note that advanced users can substitute all user friendly functions by reading/writing specific registers. A user friendly function that changes only a specific bit or bits of a register preserves value of other bits by writing back with the same value.

Registers Read/Write Group : (Access by device number and encoder number)

Register Name	Write functions	Read functions
Preset	9.4.70 PCI3E_SetPresetValue	9.4.24 PCI3E_GetPresetValue
Output Latch	9.4.10 PCI3E_GetCount (*Write & Read)	9.4.48 PCI3E_ReadOutputLatch(**)
Match	9.4.66 PCI3E_SetMatch	9.4.20 PCI3E_GetMatch
Control	9.4.57 PCI3E_SetControlMode	9.4.9 PCI3E_GetControlMode
Status	9.4.3 PCI3E_ClearCapturedStatus	9.4.31 PCI3E_GetStatus
Reset	9.4.54 PCI3E_ResetCount	N/A
Transfer Preset	9.4.44 PCI3E_PresetCount	N/A

Overview

Functions in this group read or write specific registers of a selected channel. Functions under "Write functions" are equivalent to `PCI3E_WriteRegister`, but using device number and encoder number as parameters for accessing registers. Also, functions under "Read functions" are equivalent to `PCI3E_ReadRegister`, but using device number and encoder number for accessing registers. Encoder number is equivalent to channel number. Also note the following:

* Write & Read

`PCI3E_GetCount`, first, writes to Output Latch register to transfer the value from the internal counter to the Output Latch register. Then, it immediately reads the Output Latch register to acquire the just transferred value. Use this function as a convenient way to get updated count of encoders when not using the trigger / capture feature.

**

When using the trigger/capture feature to transfer the internal counter value to the Ouput Latch register, use the `PCI3E_ReadOutputLatch` function to simply read the last latched counter value.

Counter Set-up Group

Write functions	Read functions
9.4.59 PCI3E_SetCounterMode	9.4.11 PCI3E_GetCounterMode
9.4.67 PCI3E_SetMultiplier	9.4.21 PCI3E_GetMultiplier
9.4.63 PCI3E_SetForward	9.4.17 PCI3E_GetForward
9.4.70 PCI3E_SetPresetValue (***)	9.4.24 PCI3E_GetPresetValue (***)
9.4.61 PCI3E_SetEnableAccumulator	9.4.14 PCI3E_GetEnableAccumulator

Overview
Functions in this group will help you set up the counter mode to match your system. Normal step involves calling PCI3E_SetCounterMode, PCI3E_SetMultiplier and PCI3E_SetForward. If a counter mode other than 'simple 24 bit counter' is selected, PCI3E_SetPresetValue must be called to specify preset value. Call PCI3E_SetEnableAccumulator to start the internal counter.

Function PCI3E_Get... under "Read functions" may be used to verify their PCI3E_Set... counterparts.

Note: *** These functions also belong to Registers Read/Write Group.

Index Set-up Group

Write functions	Read functions
9.4.69 PCI3E_SetPresetOnIndex	9.4.23 PCI3E_GetPresetOnIndex
9.4.65 PCI3E_SetInvertIndex	9.4.19 PCI3E_GetInvertIndex
9.4.62 PCI3E_SetEnableIndex	9.4.15 PCI3E_GetEnableIndex

Overview
If index is required for resetting or presetting counter value, functions in this group should be called. PCI3E_SetPresetOnIndex will determine the action when index signal is detected, either resetting counter to 0 or presetting counter value equal to the value in preset register. PCI3E_SetInvertIndex facilitates polarity changing of index signal. Call PCI3E_SetEnableIndex to start watching for index signal.

Function PCI3E_Get... under "Read functions" may be used to verify their PCI3E_Set... counterparts.

Count Data Handling Group

Write functions	Read functions
9.4.10 PCI3E_GetCount (***)	9.4.48 PCI3E_ReadOutputLatch (***)
9.4.54 PCI3E_ResetCount (***)	
9.4.44 PCI3E_PesetCount (***)	
9.4.58 PCI3E_SetCount	
	9.4.1 PCI3E_CaptureTimeAndCounts
	9.4.51 PCI3E_ReadTimeAndCounts

Overview

After PCI3E_SetEnableAccumulator is called, the internal counter will be updated continuously based on signals input into A, B and Index pins. The internal counter may be read directly using the PCI3E_ReadRegister function (reg.#5, reg.#13, reg.#21). The Output Latch register is used to relay the value of internal counter to external interface. To get the count value, two steps are needed. First, the Output Latch register must be written in order to transfer value from the internal counter to the Output Latch register. Second, the Output Latch register is read to retrieve the transferred value. These two steps are combined in PCI3E_GetCount function. This function is recommended when not using trigger / capture feature. PCI3E_ReadOutputLatch is normally called when the trigger / latch feature is in use. It's because a trigger event will automatically transfer the count value from the internal counter to the Output Latch register. For a detailed explanation, please refer to chapter 7 'Trigger / Capture / Data Logging Feature'.

PCI3E_ResetCount or PCI3E_PresetCount forces internal counter's value to zero or the same as Preset register's respectively.

PCI3E_SetCount forces internal counter's value to a specified value without permanently changing the Preset register. In fact, PCI3E_SetCount utilizes Preset register for transferring data to the internal counter, but the original value of Preset register is restored at the end of function call. When writing an application that always watches for changing of value of Preset register, the programmer must be aware of this temporary change of value.

PCI3E_ReadTimeAndCounts simply reads the TimeStamp Latch and each of the encoder's Output Latch while PCI3E_CaptureTimeAndCounts causes a synchronized capture of the TimeStamp counter and all channel accumulators that have captured enabled set true.

Note: *** These functions also belong to Registers Read/Write Group.

Time Stamp Group

Write functions	Read functions
	9.4.52 PCI3E_ReadTimeStamp
9.4.55 PCI3E_ResetTimeStamp	
9.4.34 PCI3E_GetTimeStamp	

Overview

PCI3E_ReadTimeStamp simply reads the Time Stamp Latch without causing the Time Stamp Counter to be transferred to the Time Stamp Latch. PCI3E_ResetTimeStamp sets the Time Stamp Counter value to zero. PCI3E_GetTimeStamp writes to the Command Register which causes the Time Stamp Counter to be latched to the Time Stamp Latch and then reads the Time Stamp Latch.

Trigger/Capture Feature Group

Capture Functions

Write functions	Read functions
9.4.56 PCI3E_SetCaptureEnabled	9.4.8 PCI3E_GetCaptureEnabled

Trigger Functions

Write functions	Read functions
9.4.66 PCI3E_SetMatch (***)	9.4.20 PCI3E_GetMatch (***)
9.4.75 PCI3E_SetTriggerOnIncrease	9.4.36 PCI3E_GetTriggerOnIncrease
9.4.76 PCI3E_SetTriggerOnIndex	9.4.37 PCI3E_GetTriggerOnIndex
9.4.77 PCI3E_SetTriggerOnMatch	9.4.38 PCI3E_GetTriggerOnMatch
9.4.71 PCI3E_SetTriggerOnDecrease	9.4.35 PCI3E_GetTriggerOnDecrease
9.4.78 PCI3E_SetTriggerOnRollover	9.4.39 PCI3E_GetTriggerOnRollover
9.4.79 PCI3E_SetTriggerOnRollunder	9.4.40 PCI3E_GetTriggerOnRollunder
9.4.80 PCI3E_SetTriggerOnZero	9.4.41 PCI3E_GetTriggerOnZero
9.4.3 PCI3E_ClearCapturedStatus (***)	9.4.31 PCI3E_GetStatus (***) 9.4.32 PCI3E_GetStatusEX

Overview

An encoder channel may be configured to generate a trigger signal when various conditions are met. This trigger signal is forwarded to all encoder channels. If a channel has capture enabled, it will then transfer the internal counter value to the Output Latch register. The trigger signal will also transfer the Time Stamp Counter to the Time Stamp Latch regardless of any channel having capture enabled.

Function PCI3E_Get... under "Read functions" may be used to verify their PCI3E_Set... counterparts. Please refer to section "Trigger/Capture Feature" and definitions of each function in this group.

Note: *** These functions also belong to Registers Read/Write Group.

Interrupt Handling Group

9.4.18 PCI3E_GetInterruptControl
9.4.53 PCI3E_RegisterInterruptHandler
PCI3E_UnRegisterInterruptHandler
9.4.64 PCI3E_SetInterruptControl
9.4.84

Overview

Four functions are provided for setup and control of interrupts related to the PCI-3E card. Use PCI3E_RegisterInterruptHandler to register your interrupt handler function before enabling the interrupt using PCI3E_SetInterruptControl. PCI3E_GetInterruptControl gets the current setting of two available interrupts, FIFO Half-Full interrupt and Encoder Trigger-Out interrupt. PCI3E_UnRegisterInterruptHandler is used to remove a registered interrupt handler function.

First-In-First-Out (FIFO) Buffer Handling Group

9.4.5 PCI3E_ClearFIFOBuffer
9.4.6 PCI3E_DisableFIFOBuffer
9.4.7 PCI3E_EnableFIFOBuffer
9.4.16 PCI3E_GetFIFOBufferCount
9.4.45 PCI3E_ReadFIFOBuffer
9.4.46 PCI3E_ReadFIFOBufferStruct

Overview

Five functions are provided that support the FIFO buffering feature related to PCI-3E card. The FIFO can be enabled using PCI3E_EnableFIFOBuffer. The PCI3E_GetFIFOBufferCount returns the number of records currently stored in the FIFO buffer. The FIFO buffer can hold up to 204 records. Each record consists of 5 entries. For details of the FIFO structure please see **6.5 FIFO Registers**. PCI3E_ClearFIFOBuffer resets the FIFO buffer. PCI3E_ReadFIFOBuffer or PCI3E_ReadFIFOBufferStruct is used to read stored records in the FIFO buffer. PCI3E_DisableFIFOBuffer disables the FIFO feature.

Digital Input Triggering Group

9.4.60 PCI3E_SetDigitalInputTriggerConfig
9.4.12 PCI3E_GetDigitalInputTriggerConfig
9.4.4 PCI3E_ClearDigitalInputTriggerStatus
9.4.13 PCI3E_GetDigitalInputTriggerStatus

Overview

Digital Input Triggering feature is implemented as a quick and easy way to capture encoder counts along with time stamp based on the rising or falling edge of external digital inputs. When the specified edge is detected on an input pin, the status of that input pin is set and the encoder counts

with time stamp are latched to the Output Latch registers(reg.#1, reg.#9, and reg.#17) and the Time Stamp Latch register (reg.#15). There are four input pins. Each input pin has its own status bit and works dependently. The status bit must be cleared using `PCI3E_ClearDigitalInputTriggerStatus` before the same pin can be used to detect the trigger signal. However, the status bits can also be cleared automatically when the FIFO buffer is enabled by `PCI3E_EnableFIFOBuffer`. While the FIFO buffer is enabled, the captured encoder counts and the time stamp are also stored in the FIFO.

Data Logging and Input/Output Group

9.4.71 PCI3E_SetSamplesToCollect
9.4.72 PCI3E_SetSamplingRateMultiplier
PCI3E_GetSamplingRateMultiplier
9.4.73 PCI3E_SetTimeBasedLogSettings
9.4.82 PCI3E_StartAcquisition
9.4.68 PCI3E_SetOutputPortConfig
9.4.85 PCI3E_WriteOutputPortRegister

9.4.27 PCI3E_GetSamplesToCollect
9.4.29
9.4.33 PCI3E_GetTimeBasedLogSettings
9.4.83 PCI3E_StopAcquisition
9.4.22 PCI3E_GetOutputPortConfig
9.4.49 PCI3E_ReadOutputPortRegister
9.4.47 PCI3E_ReadInputPortRegister
9.4.28 PCI3E_GetSamplingRateCounter
9.4.26 PCI3E_GetSamplesRemaining

Overview

`PCI3E_SetSamplingRateMultiplier` sets the 32 bit sampling rate multiplier (N) which is used to determine the sampling period. The data logging is synchronized precisely to this sampling period. `PCI3E_SetTimeBasedLogSettings` determines the input condition that must be satisfied in order to start a data acquisition. Once the acquisition is started, this function also evaluates the input condition at each sampling period to determine if the data should be stored or discarded. `PCI3E_SetSamplesToCollect` sets the number of samples to be collected when an acquisition is started. `PCI3E_StartAcquisition` starts the acquisition. The data acquisition will stop once the specified number of data has been reached. `PCI3E_StopAcquisition` can be used to abort the acquisition in progress. During the data acquisition, `PCI3E_GetSamplesRemaining` can be used to retrieve the number of samples remaining to be collected.

`PCI3E_ReadInputPortRegister` returns the value stored in the input port register. `PCI3E_WriteOutputPortRegister` sets the value stored in the output port register. `PCI3E_ReadOutputPortRegister` read back the valued stored in the output port register. `PCI3E_SetOutputPortConfig` is used to configure the output port setup. The output port pins may be driven by the output port register or trigger out signals. If the trigger out signals are used to drive the output port, then the length of the output trigger signal may also be specified. `PCI3E_GetSamplesToCollect`, `PCI3E_GetSamplingRateMultiplier`, `PCI3E_GetTimeBasedLogSettings`, `PCI3E_GetOutputPortConfig` retrieve values of each settings. `PCI3E_GetSamplingRateCounter` retrieves the current value of the sampling rate counter.

9.4 Function Definitions

9.4.1 PCI3E_CaptureTimeAndCounts

Description:

This function causes a synchronized capture of the TimeStamp counter and all channel accumulators which have captured enabled set true. Note: the `ulCounts` parameter of this function is pointer to an array of 3 unsigned longs. Each item in the array will contain a channel's output latch count value.

C Language Function Prototype:

```
int _stdcall PCI3E_CaptureTimeAndCounts(short iDeviceNo, unsigned long
*pulCounts, unsigned long *pulTimeStamp);
```

Returns:

Result code as an integer: See error code section for values other than zero. Zero implies function call is successful.

Parameters:

`iDeviceNo`: identifies the PCI-3E card (zero based).
`pulCounts`: see description above.
`pulTimeStamp`: in out parameter containing the TimeStamp value.

Example C Usage:

```
int iResult = S_OK;
short iDeviceNo = 0;
unsigned long ulCounts[3] = {0, 0, 0};
unsigned long ulTimeStamp = 0;
iResult = PCI3E_CaptureTimeAndCounts(iDeviceNo, ulCounts, &ulTimeStamp);
if ( iResult != S_OK ){ // Handle error...}
```

VB Language Function Declaration:

```
Public Declare Function PCI3E_CaptureTimeAndCounts Lib "USD_PCI_3E.dll" (ByVal
iDeviceNo As Integer, ByRef pulCounts As Long, ByRef pulTimeStamp As Long) As
Long
```

Example VB Usage:

```
Dim errCode       As Long
Dim iDeviceNo     As Integer
Dim lCounts(2)    As Long
Dim lTimeStamp    As Long

iDeviceNo = 0

errCode = PCI3E_CaptureTimeAndCounts(iDeviceNo, lCounts(0), lTimeStamp)
If errCode <> 0 then
      ‘ Handle error..
End If
```

9.4.2 PCI3E_CardCount

Description:
This function returns the number of PCI-3E cards detected on the PCI bus. The value returned should be the same value as returned in the in-out `piDeviceCount` parameter of the `PCI3E_Initialize` function.

C Language Function Prototype:
```
int _stdcall PCI3E_CardCount();
```

Returns:
See description above.

Parameters:
None

Example C Usage:
```
short iCards = PCI3E_CardCount();
```

VB Language Function Declaration:
```
Public Declare Function PCI3E_CardCount Lib "USD_PCI_3E.dll" () As Long
```

Example VB Usage:
```
Dim iCards As Integer
iCards = PCI3E_CardCount()
```

9.4.3 PCI3E_ClearCapturedStatus

Description:
This function clears the captured event status by writing 0xFFFFFFFF into the status register.

Note: Refer to section 6.2 Status Registers.

C Language Function Prototype:
```
int _stdcall PCI3E_ClearCapturedStatus(short iDeviceNo, short iEncoder);
```

Returns:
Result code as an integer: See error code section for values other than zero. Zero implies function call is successful.

Parameters:
`iDeviceNo`: identifies the PCI-3E card (zero based).
`iEncoder`: identifies the encoder channel (zero based).

Example C Usage:
```
int iResult = S_OK;
short iDeviceNo = 0;
short iEncoder = 0;

iResult = PCI3E_ClearCapturedStatus(iDeviceNo, iEncoder);
if ( iResult != S_OK ) { // Handle error...}
```

VB Language Function Declaration:
```
Public Declare Function PCI3E_ClearCapturedStatus Lib "USD_PCI_3E.dll" (ByVal
iDeviceNo As Integer, ByVal iEncoder As Integer) As Long
```

Example VB Usage:
```
Dim errCode As Long
Dim iDeviceNo As Integer
Dim iEncoder As Integer

iDeviceNo = 0
iEncoder = 0

errCode = PCI3E_ClearCapturedStatus(iDeviceNo, iEncoder)
If errCode <> 0 then
      ‘ Handle error..
End If
```

9.4.4 PCI3E_ClearDigitalInputTriggerStatus

Description:
This function clears the digital input detected status for each input by writing 0xFFFFFFFF to the digital input status register.

Note: Refer to section 6.6 Digital Input Triggering Registers.

C Language Function Prototype:
```
int _stdcall PCI3E_ClearDigitalInputTriggerStatus(short iDeviceNo);
```

Returns:
Result code as an integer: See error code section for values other than zero. Zero implies function call is successful.

Parameters:
`iDeviceNo`: identifies the PCI-3E card (zero based).

Example C Usage:
```
int iResult = S_OK;
short iDeviceNo = 0;

iResult = PCI3E_ClearDigitalInputTriggerStatus(iDeviceNo);
if ( iResult != S_OK ){ // Handle error...}
```

VB Language Function Declaration:
```
Public Declare Function PCI3E_ClearDigitalInputTriggerStatus Lib
"USD_PCI_3E.dll" (ByVal iDeviceNo As Integer) As Long
```

Example VB Usage:
```
Dim errCode As Long
Dim iDeviceNo As Integer

iDeviceNo = 0

errCode = PCI3E_ClearDigitalInputTriggerStatus(iDeviceNo)
If errCode <> 0 then
      ‘ Handle error..
End If
```

9.4.5 PCI3E_ClearFIFOBuffer

Description:
This function flushes the FIFO buffer.

C Language Function Prototype:
```
int _stdcall PCI3E_ClearFIFOBuffer(short iDeviceNo);
```

Returns:
Result code as an integer: See error code section for values other than zero. Zero implies function call is successful.

Parameters:
`iDeviceNo`: identifies the PCI-3E card (zero based).

Example C Usage:
```
int iResult = S_OK;
short iDeviceNo = 0;

iResult = PCI3E_ClearFIFOBuffer(iDeviceNo);
if ( iResult != S_OK ){ // Handle error...}
```

VB Language Function Declaration:
```
Public Declare Function PCI3E_ClearFIFOBuffer Lib "USD_PCI_3E.dll" (ByVal
iDeviceNo As Integer) As Long
```

Example VB Usage:
```
Dim errCode As Long
Dim iDeviceNo As Integer

iDeviceNo = 0

errCode = PCI3E_ClearFIFOBuffer (iDeviceNo)
If errCode <> 0 then
      ‘ Handle error..
End If
```

9.4.6 PCI3E_DisableFIFOBuffer

Description:
This function disables the FIFO buffering feature and disables auto clearing of captured event status and digital input trigger status.

C Language Function Prototype:
```
int _stdcall PCI3E_DisableFIFOBuffer(short iDeviceNo);
```

Returns:
Result code as an integer: See error code section for values other than zero. Zero implies function call is successful.

Parameters:
`iDeviceNo`: identifies the PCI-3E card (zero based).

Example C Usage:
```
int iResult = S_OK;
short iDeviceNo = 0;

iResult = PCI3E_DisableFIFOBuffer(iDeviceNo);
if ( iResult != S_OK ){ // Handle error...}
```

VB Language Function Declaration:
```
Public Declare Function PCI3E_DisableFIFOBuffer Lib "USD_PCI_3E.dll" (ByVal
iDeviceNo As Integer) As Long
```

Example VB Usage:
```
Dim errCode As Long
Dim iDeviceNo As Integer

iDeviceNo = 0
errCode = PCI3E_DisableFIFOBuffer (iDeviceNo)
If errCode <> 0 then
      ‘ Handle error..
End If
```

9.4.7 PCI3E_ EnableFIFOBuffer

Description:
This function enables the FIFO buffering feature and enables auto clearing of captured event status and digital input trigger status.

C Language Function Prototype:
```
int _stdcall PCI3E_EnableFIFOBuffer(short iDeviceNo);
```

Returns:
Result code as an integer: See error code section for values other than zero. Zero implies function call is successful.

Parameters:
`iDeviceNo`: identifies the PCI-3E card (zero based).

Example C Usage:
```
int iResult = S_OK;
short iDeviceNo = 0;

iResult = PCI3E_EnableFIFOBuffer(iDeviceNo);
if ( iResult != S_OK ){ // Handle error...}
```

VB Language Function Declaration:
```
Public Declare Function PCI3E_EnableFIFOBuffer Lib "USD_PCI_3E.dll" (ByVal
iDeviceNo As Integer) As Long
```

Example VB Usage:
```
Dim errCode As Long
Dim iDeviceNo As Integer

iDeviceNo = 0

errCode = PCI3E_EnableFIFOBuffer(iDeviceNo)
If errCode <> 0 then
      ‘ Handle error..
End If
```

9.4.8 PCI3E_GetCaptureEnabled

Description:

This function retrieves a boolean value that identifies if trigger_in causes a transfer from accumulator (counter) to output.

C Language Function Prototype:

```
int _stdcall PCI3E_GetCaptureEnabled(short iDeviceNo, short iEncoder, BOOL
*pbVal);
```

Returns:

Result code as an integer: See error code section for values other than zero. Zero implies function call is successful.

Parameters:

`iDeviceNo`: identifies the PCI-3E card (zero based).
`iEncoder`: identifies the encoder channel (zero based).
`pbVal`: inout parameter that indentifies if the capture feature is enabled.

Example C Usage:

```
int iResult = S_OK;
short iDeviceNo = 0;
short iEncoder = 0;
BOOL bVal = 0;

iResult = PCI3E_GetCaptureEnabled(iDeviceNo, iEncoder, &bVal);
if ( iResult != S_OK ){ // Handle error...}
```

VB Language Function Declaration:

```
Public Declare Function PCI3E_GetCaptureEnabled Lib "USD_PCI_3E.dll" (ByVal
iDeviceNo As Integer, ByVal iEncoder As Integer, ByRef pbVal As Long) As Long
```

Example VB Usage:

```
Dim errCode As Long
Dim iDeviceNo As Integer
Dim iEncoder As Integer
Dim bVal As Long

iDeviceNo = 0
iEncoder = 0

errCode = PCI3E_GetCaptureEnabled(iDeviceNo, iEncoder, bVal)
If errCode <> 0 then
      ‘ Handle error..
End If
```

9.4.9 PCI3E_GetControlMode

Description:

This function retrieves a 32 bit value from the control register. This value is used to control the operation of a channel. *See section 6.1 Control Registers*

C Language Function Prototype:

```
int _stdcall PCI3E_GetControlMode(short iDeviceNo, short iEncoder, unsigned
long *pulVal);
```

Returns:

Result code as an integer: See error code section for values other than zero. Zero implies function call is successful.

Parameters:

`iDeviceNo`: identifies the PCI-3E card (zero based).
`iEncoder`: identifies the encoder channel (zero based).
`pulVal`: 32 bit in-out parameter containing the control mode.

Example C Usage:

```
int iResult = S_OK;
short iDeviceNo = 0;
short iEncoder = 0;
unsigned long ulVal = 0;

iResult = PCI3E_GetControlMode(iDeviceNo, iEncoder, &ulVal );
if ( iResult != S_OK ){ // Handle error...}
```

VB Language Function Declaration:

```
Public Declare Function PCI3E_GetControlMode Lib "USD_PCI_3E.dll" (ByVal
iDeviceNo As Integer, ByVal iEncoder As Integer, ByRef pulVal As Long) As Long
```

Example VB Usage:

```
Dim errCode As Long
Dim iDeviceNo As Integer
Dim iEncoder As Integer
Dim lVal As Long

iDeviceNo = 0
iEncoder = 0

errCode = PCI3E_GetControlMode(iDeviceNo, iEncoder, lVal)
If errCode <> 0 then
      ‘ Handle error..
End If
```

9.4.10 PCI3E_GetCount

Description:

This function retrieves the count value.

Notes: This function performs the following two steps.

(1) Write to Output Latch registers (reg.#1, reg.#9, or reg.#17 based on channel selected). This action will transfer the value from internal counter register to the Output Latch registers.

(2) Read from Output Latch registers (reg.#1, reg.#9, or reg.#17 based on channel selected). The result of this read is the updated value from the Output Latch register which is passed to pulVal.

Caveats: This PCI3E_GetCount is a convenient function to easily get encoder counts from PCI-3E. However, if you want to use triggering features of PCI-3E to transfer data from internal counter to Output Latch register, you should use PCI3E_ReadRegister instead of PCI3E_GetCount. In this case, using PCI3E_GetCount to read data will result in overwriting the output latch count value when a trigger event occurs.

C Language Function Prototype:

```
int _stdcall PCI3E_GetCount(short iDeviceNo, short iEncoder, unsigned long
*pulVal);
```

Returns:

Result code as an integer: See error code section for values other than zero. Zero implies function call is successful.

Parameters:

`iDeviceNo`: identifies the PCI-3E card (zero based).

`iEncoder`: identifies the encoder channel (zero based).

`pulVal`: in/out parameter that will receive the encoder count value.

Example C Usage:

```
int iResult = S_OK;
short iDeviceNo = 0;
short iEncoder = 0;
unsigned long ulVal = 0;

iResult = PCI3E_GetCount(iDeviceNo, iEncoder, &ulVal);
if ( iResult != S_OK ){ // Handle error...}
```

VB Language Function Declaration:

```
Public Declare Function PCI3E_GetCount Lib "USD_PCI_3E.dll" (ByVal iDeviceNo As
Integer, ByVal iEncoder As Integer, ByRef pulVal As Long) As Long
```

Example VB Usage:

```
Dim errCode As Long
Dim iDeviceNo As Integer
Dim iEncoder As Integer
Dim lVal As Long

iDeviceNo = 0
iEncoder = 0
```

```
errCode = PCI3E_GetCount(iDeviceNo, iEncoder, lVal)
If errCode <> 0 then
      ‘ Handle error..
End If
```

9.4.11 PCI3E_GetCounterMode

Description:

This function retrieves a control counter mode that governs the counter behavior and limits. See parameters sections for description of the possible counter modes.

C Language Function Prototype:

```
int _stdcall PCI3E_GetCounterMode(short iDeviceNo, short iEncoder, short
*piVal);
```

Returns:

Result code as an integer: See error code section for values other than zero. Zero implies function call is successful.

Parameters:

`iDeviceNo`: identifies the PCI-3E card (zero based).
`iEncoder`: identifies the encoder channel (zero based).
`piVal`: in-out parameter containing the counter mode.

0 = acc. acts like a 24 bit counter
1 = acc. uses preset register in range-limit mode
2 = acc. uses preset register in non-recycle mode
3 = acc. uses preset register in modulo-N mode.

See 6.1 Control Registers for explanation of modes.

Example C Usage:

```
int iResult = S_OK;
short iDeviceNo = 0;
short iEncoder = 0;
short iVal = 0;

iResult = PCI3E_GetControlMode(iDeviceNo, iEncoder, &iVal);
if ( iResult != S_OK ){ // Handle error... }
```

VB Language Function Declaration:

```
Public Declare Function PCI3E_GetCounterMode Lib "USD_PCI_3E.dll" (ByVal
iDeviceNo As Integer, ByVal iEncoder As Integer, ByRef piVal As Integer) As
Long
```

Example VB Usage:

```
Dim errCode As Long
Dim iDeviceNo As Integer
Dim iEncoder As Integer
Dim iVal As Integer

iDeviceNo = 0
iEncoder = 0

errCode = PCI3E_GetCounterMode(iDeviceNo, iEncoder, iVal)
If errCode <> 0 then
      ‘ Handle error..
End If
```

9.4.12 PCI3E_GetDigitalInputTriggerConfig

Description:
This function retrieves the digital input trigger configuration settings.

Note: Refer to section 6.6 Digital Input Triggering Registers.

C Language Function Prototype:
```
int _stdcall PCI3E_GetDigitalInputTriggerConfig(short iDeviceNo, BOOL
*pbEnableTrigger, BOOL *pbTriggerOnRisingEdge);
```

Returns:
Result code as an integer: See error code section for values other than zero. Zero implies function call is successful.

Parameters:
`iDeviceNo`: identifies the PCI-3E card (zero based).
`pbEnableTrigger`: an array of four booleans which determine if input signals can generate a trigger out signal.
`pbTriggerOnRisingEdge`: an array of four booleans which determine the type of tigger signal, rising or falling edge. 1 = rising edge, 0 = falling edge.

Example C Usage:
```
int iResult = S_OK;
int iDeviceNo = 0;
BOOL bEnableTrigger[4] = {FALSE, FALSE, FALSE, FALSE};
BOOL bTriggerOnRisingEdge[4] = {FALSE, FALSE, FALSE, FALSE};

iResult = PCI3E_GetDigitalInputTriggerConfig(iDeviceNo, bEnableTrigger,
      bTriggerOnRisingEdge);
if ( iResult != S_OK ){ // Handle error...}
```

VB Language Function Declaration:
```
Public Declare Function PCI3E_GetDigitalInputTriggerConfig Lib "USD_PCI_3E.dll"
(ByVal iDeviceNo As Integer, ByRef bEnableTrigger As Long, ByRef
bTriggerOnRisingEdge As Long) As Long
```

Example VB Usage:
```
Dim errCode As Long
Dim iDeviceNo As Integer
Dim bEnableTrigger(3) As Long
Dim bTriggerOnRisingEdge (3) As Long

iDeviceNo = 0

errCode = PCI3E_GetDigitalInputTriggerConfig(iDeviceNo, bEnableTrigger(0),
      bTriggerOnRisingEdge(0))
If errCode <> 0 then
      ‘ Handle error..
End If
```

9.4.13 PCI3E_GetDigitalInputTriggerStatus

Description:
This function retrieves the digital input trigger event detected status.

Note: Refer to section 6.6 Digital Input Triggering Registers.

C Language Function Prototype:
```
int _stdcall PCI3E_GetDigitalInputTriggerStatus(short iDeviceNo, BOOL *pbVal);
```

Returns:
Result code as an integer: See error code section for values other than zero. Zero implies function call is successful.

Parameters:
`iDeviceNo`: identifies the PCI-3E card (zero based).
`pbVal`: an array of four booleans that identifying if input signals have been detected.

Example C Usage:
```
int iResult = S_OK;
int iDeviceNo = 0;

BOOL bVal[4] = {FALSE, FALSE, FALSE, FALSE};

iResult = PCI3E_GetDigitalInputTriggerStatus(iDeviceNo, bVal);
if ( iResult != S_OK ){ // Handle error...}
```

VB Language Function Declaration:
```
Public Declare Function PCI3E_GetDigitalInputTriggerStatus Lib "USD_PCI_3E.dll"
(ByVal iDeviceNo As Integer, ByRef pbVal As Long) As Long
```

Example VB Usage:
```
Dim errCode As Long
Dim iDeviceNo As Integer
Dim bVal(3) As Long

iDeviceNo = 0

errCode = PCI3E_GetDigitalInputTriggerStatus(iDeviceNo, bVal(0))
If errCode <> 0 then
      ‘ Handle error..
End If
```

9.4.14 PCI3E_GetEnableAccumulator

Description:

This function retrieves a boolean value that indicates whether the master enable for accumulator is set, (must be set to true to count).

C Language Function Prototype:

```
int _stdcall PCI3E_GetEnableAccumulator(short iDeviceNo, short iEncoder, BOOL
*pbVal);
```

Returns:

Result code as an integer: See error code section for values other than zero. Zero implies function call is successful.

Parameters:

`iDeviceNo`: identifies the PCI-3E card (zero based).
`iEncoder`: identifies the encoder channel (zero based).
`pbVal`: in-out boolean parameter identifying whether the counter is enabled.

Example C Usage:

```
int iResult = S_OK;
short iDeviceNo = 0;
short iEncoder = 0;
BOOL bVal = FALSE;

iResult = PCI3E_GetEnableAccumulator(iDeviceNo, iEncoder, &bVal);
if ( iResult != S_OK ){ // Handle error... }
```

VB Language Function Declaration:

```
Public Declare Function PCI3E_GetEnableAccumulator Lib "USD_PCI_3E.dll" (ByVal
iDeviceNo As Integer, ByVal iEncoder As Integer, ByRef pbVal As Long) As Long
```

Example VB Usage:

```
Dim errCode As Long
Dim iDeviceNo As Integer
Dim iEncoder As Integer
Dim bVal As Long

iDeviceNo = 0
iEncoder = 0

errCode = PCI3E_GetEnableAccumulator(iDeviceNo, iEncoder, bVal)
If errCode <> 0 then
      ‘ Handle error..
End If
```

9.4.15 PCI3E_GetEnableIndex

Description:

This function retrieves a boolean value that indicates whether index detection is enabled. When enabled, you can use the PCI3E_SetPresetOnIndex to determine how to respond to an index signal.

C Language Function Prototype:

```
int _stdcall PCI3E_GetEnableIndex(short iDeviceNo, short iEncoder, BOOL
*pbVal);
```

Returns:

Result code as an integer: See error code section for values other than zero. Zero implies function call is successful.

Parameters:

`iDeviceNo`: identifies the PCI-3E card (zero based).
`iEncoder`: identifies the encoder channel (zero based).
`pbVal`: in-out boolean parameter identifying whether the index is enabled.

Example C Usage:

```
int iResult = S_OK;
short iDeviceNo = 0;
short iEncoder = 0;
BOOL bVal = FALSE;

iResult = PCI3E_GetEnableIndex(iDeviceNo, iEncoder, &bVal);
if ( iResult != S_OK ){ // Handle error... }
```

VB Language Function Declaration:

```
Public Declare Function PCI3E_GetEnableIndex Lib "USD_PCI_3E.dll" (ByVal
iDeviceNo As Integer, ByVal iEncoder As Integer, ByRef pbVal As Long) As Long
```

Example VB Usage:

```
Dim errCode As Long
Dim iDeviceNo As Integer
Dim iEncoder As Integer
Dim bVal As Long

iDeviceNo = 0
iEncoder = 0

errCode = PCI3E_GetEnableIndex(iDeviceNo, iEncoder, bVal)
If errCode <> 0 then
      ‘ Handle error..
End If
```

9.4.16 PCI3E_GetFIFOBufferCount

Description:
This function retrieves the number of records currently stored in the FIFO buffer.

C Language Function Prototype:
```
int _stdcall PCI3E_GetFIFOBufferCount(short iDeviceNo, short *piVal);
```

Returns:
Result code as an integer: This function will return FIFO_BUFFER_FULL if the FIFO buffer is full when the call is made. See error code section for values other than zero. Zero implies function call is successful.

Parameters:
iDeviceNo: identifies the PCI-3E card (zero based).
piVal: contains the number of records stored in the FIFO buffer.

Example C Usage:
```
int iResult = S_OK;
short iDeviceNo = 0;
short iVal = 0;

iResult = PCI3E_GetFIFOBufferCount(iDeviceNo, &iVal);
if ( iResult != S_OK ){ // Handle error...}
```

VB Language Function Declaration:
```
Public Declare Function PCI3E_GetFIFOBufferCount Lib "USD_PCI_3E.dll" (ByVal
iDeviceNo As Integer, ByRef piVal As Integer) As Long
```

Example VB Usage:
```
Dim errCode   As Long
Dim iDeviceNo As Integer
Dim iVal      As Integer

iDeviceNo = 0

errCode = PCI3E_GetFIFOBufferCount(iDeviceNo, iVal)
If errCode <> 0 then
      ‘ Handle error..
End If
```

9.4.17 PCI3E_GetForward

Description:
This function retrieves a boolean value that indicates whether the B input of quadrature signal is inverted. 1 = inverted, 0 = not inverted.

C Language Function Prototype:
```
int _stdcall PCI3E_GetForward(short iDeviceNo, short iEncoder, BOOL *pbVal);
```

Returns:
Result code as an integer: See error code section for values other than zero. Zero implies function call is successful.

Parameters:
`iDeviceNo`: identifies the PCI-3E card (zero based).
`iEncoder`: identifies the encoder channel (zero based).
`pbVal`: in-out boolean parameter identifying if the B signal is inverted or not.

Example C Usage:
```
int iResult = S_OK;
short iDeviceNo = 0;
short iEncoder = 0;
BOOL bVal = FALSE;

iResult = PCI3E_GetForward(iDeviceNo, iEncoder, &bVal);
if ( iResult != S_OK ){ // Handle error... }
```

VB Language Function Declaration:
```
Public Declare Function PCI3E_GetForward Lib "USD_PCI_3E.dll" (ByVal iDeviceNo
As Integer, ByVal iEncoder As Integer, ByRef pbVal As Long) As Long
```

Example VB Usage:
```
Dim errCode As Long
Dim iDeviceNo As Integer
Dim iEncoder As Integer
Dim bVal As Long

iDeviceNo = 0
iEncoder = 0

errCode = PCI3E_GetForward(iDeviceNo, iEncoder, bVal)
If errCode <> 0 then
      ‘ Handle error..
End If
```

9.4.18 PCI3E_GetInterruptControl

Description:

This function gets the current enable state of the FIFO Half-Full interrupt and the Trigger-Out interrupt.

Note: if the FIFO Half-Full interrupt is enabled, the FIFO buffer should also be enabled using the `PCI3E_EnableFIFOBuffer` function.

C Language Function Prototype:

```
int _stdcall PCI3E_GetInterruptControl(short iDeviceNo, BOOL
*pbEnableFIFOHalfFullInterrupt, BOOL *pbEnableTriggerOutInterrupt);
```

Returns:

Result code as an integer: See error code section for values other than zero. Zero implies function call is successful.

Parameters:

`iDeviceNo`: identifies the PCI-3E card (zero based).
`pbEnableFIFOHalfFullInterrupt`: in-out boolean parameter that holds the enable state of the FIFO Half-Full interrupt. 1 = enabled, 0 = disabled
`pbEnableTriggerOutInterrupt`: in-out boolean parameter that holds the enable state of the trigger-out interrupt. 1 = enabled, 0 = disabled

Example C Usage:

```
int iResult = S_OK;
short iDeviceNo = 0;
BOOL bFIFOHalfFullInterrupt = FALSE;
BOOL bEncoderTriggerOutInterrupt = FALSE;

iResult = PCI3E_GetInterruptControl(iDeviceNo, &bFIFOHalfFullInterrupt,
     &bEncoderTriggerOutInterrupt);
if ( iResult != S_OK ){ // Handle error...}
```

VB Language Function Declaration:

```
Public Declare Function PCI3E_GetInterruptControl Lib "USD_PCI_3E.dll" (ByVal
iDeviceNo As Integer, ByRef pbFIFOHalfFullInterrupt As Long, ByRef
pbEncoderTriggerOutInterrupt As Long) As Long
```

Example VB Usage:

```
Dim errCode As Long
Dim iDeviceNo As Integer
Dim bFIFOHalfFullInterrupt As Long
Dim bEncoderTriggerOutInterrupt As Long

iDeviceNo = 0

errCode = PCI3E_GetInterruptControl(iDeviceNo, bFIFOHalfFullInterrupt,
bEncoderTriggerOutInterrupt)
If errCode <> 0 then
      ‘ Handle error..
End If
```

9.4.19 PCI3E_GetInvertIndex

Description:

This function retrieves a boolean value that determines the active level of the index.
`bVal = TRUE` when the index is active low. Default is active high.

C Language Function Prototype:

```
int _stdcall PCI3E_GetInvertIndex(short iDeviceNo, short iEncoder, BOOL
*pbVal);
```

Returns:

Result code as an integer: See error code section for values other than zero. Zero implies function call is successful.

Parameters:

`iDeviceNo`: identifies the PCI-3E card (zero based).
`iEncoder`: identifies the encoder channel (zero based).
`pbVal`: in-out boolean parameter. See description above.

Example C Usage:

```
int iResult = S_OK;
short iDeviceNo = 0;
short iEncoder = 0;
BOOL bVal = FALSE;

iResult = PCI3E_GetInvertIndex(iDeviceNo, iEncoder, &bVal);
if ( iResult != S_OK ){ // Handle error... }
```

VB Language Function Declaration:

```
Public Declare Function PCI3E_GetInvertIndex Lib "USD_PCI_3E.dll" (ByVal
iDeviceNo As Integer, ByVal iEncoder As Integer, ByRef pbVal As Long) As Long
```

Example VB Usage:

```
Dim errCode As Long
Dim iDeviceNo As Integer
Dim iEncoder As Integer
Dim bVal As Long

iDeviceNo = 0
iEncoder = 0

errCode = PCI3E_GetInvertIndex(iDeviceNo, iEncoder, bVal)
If errCode <> 0 then
      ‘ Handle error..
End If
```

9.4.20 PCI3E_GetMatch

Description:

This function retrieves the match register value. The match register is used as a reference to signal a capture when the counter equals the match register value.

C Language Function Prototype:

```
int _stdcall PCI3E_GetMatch(short iDeviceNo, short iEncoder, unsigned long
*pulVal);
```

Returns:

Result code as an integer: See error code section for values other than zero. Zero implies function call is successful.

Parameters:

`iDeviceNo`: identifies the PCI-3E card (zero based).
`iEncoder`: identifies the encoder channel (zero based).
`pulVal`: in-out parameter containing the match register value.

Example C Usage:

```
int iResult = S_OK;
short iDeviceNo = 0;
short iEncoder = 0;
unsigned long ulVal = 0;

iResult = PCI3E_GetMatch(iDeviceNo, iEncoder, &ulVal);
if ( iResult != S_OK ){ // Handle error... }
```

VB Language Function Declaration:

```
Public Declare Function PCI3E_GetMatch Lib "USD_PCI_3E.dll" (ByVal iDeviceNo As
Integer, ByVal iEncoder As Integer, ByRef pulVal As Long) As Long
```

Example VB Usage:

```
Dim errCode As Long
Dim iDeviceNo As Integer
Dim iEncoder As Integer
Dim lVal As Long

iDeviceNo = 0
iEncoder = 0

errCode = PCI3E_GetMatch(iDeviceNo, iEncoder, lVal)
If errCode <> 0 then
      ‘ Handle error..
End If
```

9.4.21 PCI3E_GetMultiplier

Description:

This function retrieves the multiplier mode that determines when the counter is to be incremented based on the number of quadrature state transitions. See possible values in parameters description.

C Language Function Prototype:

```
int _stdcall PCI3E_GetMultiplier(short iDeviceNo, short iEncoder, short
*piVal);
```

Returns:

Result code as an integer: See error code section for values other than zero. Zero implies function call is successful.

Parameters:

`iDeviceNo`: identifies the PCI-3E card (zero based).

`iEncoder`: identifies the encoder channel (zero based).

`piVal`: identifies when the counter increments.

Possible values are: 0 = in_a is clock, in_b is direction
1 = count increments once every four quad states, X1
2 = count increments once every two quad states, X2
3 = count increments once every quad state, X4

Example C Usage:

```
int iResult = S_OK;
short iDeviceNo = 0;
short iEncoder = 0;
short iVal = 0;

iResult = PCI3E_GetMultiplier(iDeviceNo, iEncoder, &iVal);
if ( iResult != S_OK ){ // Handle error... }
```

VB Language Function Declaration:

```
Public Declare Function PCI3E_GetMultiplier Lib "USD_PCI_3E.dll" (ByVal
iDeviceNo As Integer, ByVal iEncoder As Integer, ByRef piVal As Integer) As
Long
```

Example VB Usage:

```
Dim errCode As Long
Dim iDeviceNo As Integer
Dim iEncoder As Integer
Dim iVal As Integer

iDeviceNo = 0
iEncoder = 0

errCode = PCI3E_GetMultiplier(iDeviceNo, iEncoder, iVal)
If errCode <> 0 then
      ‘ Handle error..
End If
```

9.4.22 PCI3E_GetOutputPortConfig

Description:
This function retrieves the output port setup configuration.

The output port pins may be driven by the output port register or trigger out signals.

If the trigger out signal is used to drive the output port, then the `pucTriggerSignalLengthCode` parameter indicates the length of the output trigger signal.

C Language Function Prototype:
```
int _stdcall PCI3E_GetOutputPortConfig(short iDeviceNo, BOOL
*pbTriggerOutSignalDrivesOutputPin, unsigned char *pucTriggerSignalLengthCode);
```

Returns:
Result code as an integer: See error code section for values other than zero. Zero implies function call is successful.

Parameters:
`iDeviceNo`: identifies the PCI-3E card (zero based).
`pbTriggerOutSignalDrivesOutputPin`: pointer to an array of 4 booleans which are used indicate if the cooresponding output port pins are driven by the output port register or trigger out signals.

array element 0:	0 --- OUT0 is driven by bit 0 of reg.#46
	1 --- OUT0 is driven by Trigger Out signal from Encoder Channel 0
array element 1:	0 --- OUT1 is driven by bit 1 of reg.#46
	1 --- OUT1 is driven by Trigger Out signal from Encoder Channel 1
array element 2:	0 --- OUT2 is driven by bit 2 of reg.#46
	1 --- OUT2 is driven by Trigger Out signal from Encoder Channel 2
array element 3:	0 --- OUT3 is driven by bit 3 of reg.#46
	1 --- OUT3 is driven by Combined Trigger Out signal

`pucTriggerSignalLengthCode`: identifies the length of the signal generated on the output pin. This only applies when the output is driven by Trigger Out signal.

Code	Length of Trigger Signal
0	1 mS
1	200 µS
2	20 µS
3	5 µS
4	Toggle

Example C Usage:
```
int iResult = S_OK;
short iDeviceNo = 0;
BOOL bTriggerOutSignalDrivesOutputPin[4] = {0, 0, 0, 0};
```

```
unsigned char ucTriggerSignalLengthCode = 0;
iResult = PCI3E_GetOutputPortConfig(iDeviceNo,
                                    bTriggerOutSignalDrivesOutputPin,
                                    &ucTriggerSignalLengthCode);
if ( iResult != S_OK ){ // Handle error...}
```

VB Language Function Declaration:

```
Public Declare Function PCI3E_GetOutputPortConfig Lib "USD_PCI_3E.dll" (ByVal
iDeviceNo As Integer, ByRef pbTriggerOutSignalDrivesOutputPin As Long, ByRef
ucTriggerSignalLengthCode As Byte) As Long
```

Example VB Usage:

```
Dim errCode As Long
Dim iDeviceNo As Integer
Dim bTriggerOutSignalDrivesOutputPin(3) As Long
Dim bytTriggerSignalLengthCode As Byte

iDeviceNo = 0
bytTriggerSignalLengthCode = 0

errCode = PCI3E_GetOutputPortConfig(iDeviceNo, _
                                    bTriggerOutSignalDrivesOutputPin, _
                                    bytTriggerSignalLengthCode)
If errCode <> 0 then
      ‘ Handle error..
End If
```

9.4.23 PCI3E_GetPresetOnIndex

Description:

This function retrieves a boolean value that indicates whether index will either reset or preset accumulator (counter). 1 = preset occurs when index is detected, 0 = reset will occur when index is detected. This function requires that the index is enabled using PCI3E_SetEnableIndex.

C Language Function Prototype:

```
int _stdcall PCI3E_GetPresetOnIndex(short iDeviceNo, short iEncoder, BOOL
*pbVal);
```

Returns:

Result code as an integer: See error code section for values other than zero. Zero implies function call is successful.

Parameters:

`iDeviceNo`: identifies the PCI-3E card (zero based).
`iEncoder`: identifies the encoder channel (zero based).
`pbVal`: in-out parameter that will receive the preset on index enable value.

Example C Usage:

```
int iResult = S_OK;
short iDeviceNo = 0;
short iEncoder = 0;
BOOL bVal = False;

iResult = PCI3E_GetPresetOnIndex(iDeviceNo, iEncoder, &bVal);
if ( iResult != S_OK ){ // Handle error...}
```

VB Language Function Declaration:

```
Public Declare Function PCI3E_GetPresetOnIndex Lib "USD_PCI_3E.dll" (ByVal
iDeviceNo As Integer, ByVal iEncoder As Integer, ByRef pbVal As Long) As Long
```

Example VB Usage:

```
Dim errCode As Long
Dim iDeviceNo As Integer
Dim iEncoder As Integer
Dim bVal As Long

iDeviceNo = 0
iEncoder = 0

errCode = PCI3E_GetPresetOnIndex(iDeviceNo, iEncoder, bVal)
If errCode <> 0 then
      ‘ Handle error..
End If
```

9.4.24 PCI3E_GetPresetValue

Description:
This function retrieves the Preset register value.

C Language Function Prototype:
```
int _stdcall PCI3E_GetPresetValue(short iDeviceNo, short iEncoder, unsigned
long *pulVal);
```

Returns:
Result code as an integer: See error code section for values other than zero. Zero implies function call is successful.

Parameters:
`iDeviceNo`: identifies the PCI-3E card (zero based).
`iEncoder`: identifies the encoder channel (zero based).
`pulVal`: the Preset register value may also be referred to as upper-limit, range-limit, max count, or resolution -1.

Example C Usage:
```
int iResult = S_OK;
short iDeviceNo = 0;
short iEncoder = 0;
unsigned long ulVal = 0;

iResult = PCI3E_GetPresetValue(iDeviceNo, iEncoder, &ulVal);
if ( iResult != S_OK ){ // Handle error...}
```

VB Language Function Declaration:
```
Public Declare Function PCI3E_GetPresetValue Lib "USD_PCI_3E.dll" (ByVal
iDeviceNo As Integer, ByVal iEncoder As Integer, ByRef pulVal As Long) As Long
```

Example VB Usage:
```
Dim errCode As Long
Dim iDeviceNo As Integer
Dim iEncoder As Integer
Dim lVal As Long

iDeviceNo = 0
iEncoder = 0

errCode = PCI3E_GetPresetValue(iDeviceNo, iEncoder, lVal)
If errCode <> 0 then
      ‘ Handle error..
End If
```

9.4.25 PCI3E_GetROM_ID

Description:
This function retrieves the ROM_ID which is contained in bits 24 through 31 of the Command register.

C Language Function Prototype:
```
int _stdcall PCI3E_GetROM_ID(short iDeviceNo, unsigned char *pucVal);
```

Returns:
Result code as an integer: See error code section for values other than zero. Zero implies function call is successful.

Parameters:
`iDeviceNo`: identifies the PCI-3E card (zero based).
`pucVal`: an eight bit value that identifies the ROM ID.

Example C Usage:
```
int iResult = S_OK;
short iDeviceNo = 0;
unsigned char ucVal = 0;

iResult = PCI3E_GetROM_ID(iDeviceNo, &ucVal);
if ( iResult != S_OK ){ // Handle error...}
```

VB Language Function Declaration:
```
Public Declare Function PCI3E_GetROM_ID Lib "USD_PCI_3E.dll" (ByVal iDeviceNo
As Integer, ByRef pucVal As Byte) As Long
```

Example VB Usage:
```
Dim errCode As Long
Dim iDeviceNo As Integer
Dim bytVal As Byte

iDeviceNo = 0
iEncoder = 0

errCode = PCI3E_GetROM_ID(iDeviceNo, bytVal);
If errCode <> 0 then
      ‘ Handle error..
End If
```

9.4.26 PCI3E_GetSamplesRemaining

Description:
This function retrieves the number of samples remaining to be collected.

C Language Function Prototype:
```
int _stdcall PCI3E_GetSamplesRemaining(short iDeviceNo, unsigned long *pulVal);
```

Returns:
Result code as an integer: See error code section for values other than zero. Zero implies function call is successful.

Parameters:
`iDeviceNo`: identifies the PCI-3E card (zero based).
`pulVal`: in-out parameter that identifies the number of samples remaining to be collected.

Example C Usage:
```
int iResult = S_OK;
short iDeviceNo = 0;
unsigned long ulVal = 0;

iResult = PCI3E_GetSamplesRemaining(iDeviceNo, &ulVal);
if ( iResult != S_OK ){ // Handle error... }
```

VB Language Function Declaration:
```
Public Declare Function PCI3E_GetSamplesRemaining Lib "USD_PCI_3E.dll" (ByVal
iDeviceNo As Integer, ByRef pulVal As Long) As Long
```

Example VB Usage:
```
Dim errCode As Long
Dim iDeviceNo As Integer
Dim lVal As Long

iDeviceNo = 0

errCode = PCI3E_GetSamplesRemaining(iDeviceNo, lVal)
If errCode <> 0 then
      ‘ Handle error..
End If
```

9.4.27 PCI3E_GetSamplesToCollect

Description:
This function retrieves the number of samples to be collected when an acquisition is started.

C Language Function Prototype:
```
int _stdcall PCI3E_GetSamplesToCollect(short iDeviceNo, unsigned long *pulVal);
```

Returns:
Result code as an integer: See error code section for values other than zero. Zero implies function call is successful.

Parameters:
`iDeviceNo`: identifies the PCI-3E card (zero based).
`pulVal`: in-out parameter that identifies the number of samples to be collected.

Example C Usage:
```
int iResult = S_OK;
short iDeviceNo = 0;
unsigned long ulVal = 0;

iResult = PCI3E_GetSamplesToCollect(iDeviceNo, &ulVal);
if ( iResult != S_OK ){ // Handle error...}
```

VB Language Function Declaration:
```
Public Declare Function PCI3E_GetSamplesToCollect Lib "USD_PCI_3E.dll" (ByVal
iDeviceNo As Integer, ByRef pulVal As Long) As Long
```

Example VB Usage:
```
Dim errCode As Long
Dim iDeviceNo As Integer
Dim lVal As Long

iDeviceNo = 0
iEncoder = 0

errCode = PCI3E_GetSamplesToCollect(iDeviceNo, lVal)
If errCode <> 0 then
      ‘ Handle error..
End If
```

9.4.28 PCI3E_GetSamplingRateCounter

Description:

This function retrieves the number of sample periods that have expired since the data acquisition was last started.

C Language Function Prototype:

```
int _stdcall PCI3E_GetSamplingRateCounter(short iDeviceNo, unsigned long
*pulVal);
```

Returns:

Result code as an integer: See error code section for values other than zero. Zero implies function call is successful.

Parameters:

`iDeviceNo`: identifies the PCI-3E card (zero based).

`pulval:` in-out parameter that identifies the number of sample periods that have expired since the data acquisition was last started.

Example C Usage:

```
int iResult = S_OK;
short iDeviceNo = 0;
unsigned long ulVal = 0;

iResult = PCI3E_GetSamplingRateCounter(iDeviceNo, &ulVal);
if ( iResult != S_OK ){ // Handle error... }
```

VB Language Function Declaration:

```
Public Declare Function PCI3E_GetSamplingRateCounter Lib "USD_PCI_3E.dll"
(ByVal iDeviceNo As Integer, ByRef pulVal As Long) As Long
```

Example VB Usage:

```
Dim errCode As Long
Dim iDeviceNo As Integer
Dim lVal As Long

iDeviceNo = 0

errCode = PCI3E_GetSamplingRateCounter(iDeviceNo, lVal)
If errCode <> 0 then
      ‘ Handle error..
End If
```

9.4.29 PCI3E_GetSamplingRateMultiplier

Description:

This function retrieves the 32 bit sampling rate multiplier (N) which is used to determine the sampling period. The sampling period is calculated by the following equations.

N: the value of the "sampling rate multiplier register"
The sampling period = (N+1) * 30 microsecond

C Language Function Prototype:

```
int _stdcall PCI3E_GetSamplingRateMultiplier(short iDeviceNo, unsigned long
*pulVal);
```

Returns:

Result code as an integer: See error code section for values other than zero. Zero implies function call is successful.

Parameters:

`iDeviceNo`: identifies the PCI-3E card (zero based).
`pulVal`: in-out parameter that contains the sampling rate multiplier used to calculate the sampling period.

Example C Usage:

```
int iResult = S_OK;
short iDeviceNo = 0;
unsigned long ulVal = 0;

iResult = PCI3E_GetSamplingRateMultiplier(iDeviceNo, &ulVal);
if ( iResult != S_OK ){ // Handle error... }
```

VB Language Function Declaration:

```
Public Declare Function PCI3E_GetSamplingRateMultiplier Lib "USD_PCI_3E.dll"
(ByVal iDeviceNo As Integer, ByRef pulVal As Long) As Long
```

Example VB Usage:

```
Dim errCode As Long
Dim iDeviceNo As Integer
Dim lVal As Long

iDeviceNo = 0

errCode = PCI3E_GetSamplingRateMultiplier(iDeviceNo, lVal)
If errCode <> 0 then
      ‘ Handle error..
End If
```

9.4.30 PCI3E_GetSlotNumber

Description:
This function retrieves the slot number assigned to the PCI-3E card via the PCI bus.

C Language Function Prototype:
```
int _stdcall PCI3E_GetSlotNumber(short iDeviceNo, short *piVal);
```

Returns:
Result code as an integer: See error code section for values other than zero. Zero implies function call is successful.

Parameters:
`iDeviceNo`: identifies the PCI-3E card (zero based).
`piVal`: identifies the assigned PCI bus slot number.

Example C Usage:
```
int iResult = S_OK;
short iDeviceNo = 0;
short iVal = 0;

iResult = PCI3E_GetSlotNumber(iDeviceNo, &iVal);
if ( iResult != S_OK ){ // Handle error... }
```

VB Language Function Declaration:
```
Public Declare Function PCI3E_GetSlotNumber Lib "USD_PCI_3E.dll" (ByVal
iDeviceNo As Integer, ByRef piVal As Integer) As Long
```

Example VB Usage:
```
Dim errCode As Long
Dim iDeviceNo As Integer
Dim iVal As Integer

iDeviceNo = 0

errCode = PCI3E_GetSlotNumber(iDeviceNo, iVal)
If errCode <> 0 then
      ‘ Handle error..
End If
```

9.4.31 PCI3E_GetStatus

Description:
This function retrieves the status register value for an encoder channel.

C Language Function Prototype:
```
int _stdcall PCI3E_GetStatus(short iDeviceNo, short iEncoder, unsigned long
*pulVal);
```

Returns:
Result code as an integer: See error code section for values other than zero. Zero implies function call is successful.

Parameters:
`iDeviceNo`: identifies the PCI-3E card (zero based).
`iEncoder`: identifies the encoder channel (zero based).
`pulVal`: in out parameters which contains the status register value. *Refer to section 6.2 Status Registers.*

Example C Usage:
```
int iResult = S_OK;
short iDeviceNo = 0;
short iEncoder = 0;
unsigned long ulVal = 0;

iResult = PCI3E_GetStatus(iDeviceNo, iEncoder, &ulVal);
if ( iResult != S_OK ){ // Handle error... }
```

VB Language Function Declaration:
```
Public Declare Function PCI3E_GetStatus Lib "USD_PCI_3E.dll" (ByVal iDeviceNo
As Integer, ByVal iEncoder As Integer, ByRef pulVal As Long) As Long
```

Example VB Usage:
```
Dim errCode As Long
Dim iDeviceNo As Integer
Dim iEncoder As Integer
Dim lVal As Long

iDeviceNo = 0
iEncoder = 0

errCode = PCI3E_GetStatus(iDeviceNo, iEncoder, lVal)
If errCode <> 0 then
      ‘ Handle error..
End If
```

9.4.32 PCI3E_GetStatusEx

Description:
This function retrieves the status of each trigger on event for a specified encoder channel.

C Language Function Prototype:
```
int _stdcall PCI3E_GetStatusEx(short iDeviceNo, short iEncoder, BOOL
*pbDecreaseDetected, BOOL *pbIncreaseDetected, BOOL *pbIndexDetected, BOOL
*pbRollunderDetected, BOOL *pbRolloverDetected, BOOL *pbMatchDetected, BOOL
*pbZeroDetected);
```

Returns:
Result code as an integer: See error code section for values other than zero. Zero implies function call is successful.

Parameters:
`iDeviceNo`: identifies the PCI-3E card (zero based).
`iEncoder`: identifies the encoder channel (zero based).
`pbDecreaseDetected`: indicates if the accumulator has decreased in value.
`pbIncreaseDetected`: indicates if the accumulator has increased in value.
`pbIndexDetected`: indicates if an index signal has been detected.
`pbRollunderDetected`: indicates if a rollunder has occurred.
`pbRolloverDetected`: indicates if a rollover has occurred.
`pbMatchDetected`: indicates if a match has occurred.
`pbZeroDetected`: indicates if the accumulator was equal to zero.

Refer to section *6.2 Status Registers.*

Example C Usage:
```
int iResult = S_OK;
short iDeviceNo = 0;
Short iEncoder = 0;
BOOL bDecreaseDetected = FALSE;
BOOL bIncreaseDetected = FALSE;
BOOL bIndexDetected = FALSE;
BOOL bRollunderDetected = FALSE;
BOOL bRolloverDetected = FALSE;
BOOL bMatchDetected = FALSE;
BOOL bZeroDetected = FALSE;

iResult = PCI3E_GetStatusEx(iDeviceNo, iEncoder, &bDecreaseDetected,
                            &bIncreaseDetected, &bIndexDetected,
                            &bRollunderDetected, &bRolloverDetected,
                            &bMatchDetected, &bZeroDetected);
if ( iResult != S_OK ){ // Handle error... }
```

VB Language Function Declaration:
```
Public Declare Function PCI3E_GetStatusEx Lib "USD_PCI_3E.dll" (ByVal iDeviceNo
As Integer, ByVal iEncoder As Integer, ByRef pbDecreaseDetected As Long, ByRef
pbIncreaseDetected As Long, ByRef pbIndexDetected As Long, ByRef
pbRollunderDetected As Long, ByRef pbRolloverDetected As Long, ByRef
pbMatchDetected As Long, ByRef pbZeroDetected As Long) As Long
```

Example VB Usage:

```
Dim errCode As Long
Dim iDeviceNo As Integer
Dim iEncoder As Integer
Dim bDecreaseDetected As Long
Dim bIncreaseDetected As Long
Dim bIndexDetected As Long
Dim bRollunderDetected As Long
Dim bRolloverDetected As Long
Dim bMatchDetected As Long
Dim bZeroDetected As Long

iDeviceNo = 0
iEncoder = 0

errCode = PCI3E_GetStatusEx(iDeviceNo, iEncoder, bDecreaseDetected,
                            bIncreaseDetected, bIndexDetected,
                            bRollunderDetected, bRolloverDetected,
                            bMatchDetected, bZeroDetected)
If errCode <> 0 then
      ‘ Handle error..
End If
```

9.4.33 PCI3E_GetTimeBasedLogSettings

Description:

This function retrieves the settings used to determine the condition that must be satisfied in order to start a data acquisistion. Once the acquisition is started, the PCI-3E will evaluate the `pucQualifier` parameter on each sampling period to determine if the data should be stored or discarded.

C Language Function Prototype:

```
int _stdcall PCI3E_GetTimeBasedLogSettings(short iDeviceNo,
unsigned char *pucTrigger, unsigned char *pucTrigAnd,
unsigned char *pucQualifier, unsigned char *pucQualAnd,
unsigned long *pulNumberOfSamples);
```

Returns:

Result code as an integer: See error code section for values other than zero. Zero implies function call is successful.

Parameters:

`iDeviceNo`: identifies the PCI-3E card (zero based).
pucTrigger: an array of 4 triggering codes.
pucTrigAnd: in-out parameter that determines if the array of trigger codes are AND'ed or OR'ed.
1 = AND'ed, 0 = OR'ed
pucQualifier: and array of 4 qualifier codes.
pucQualAnd: in-out parameter that determines if the array of qualifier codes are AND'ed or OR'ed.
1 = AND'ed, 0 = OR'ed
pulNumberOfSamples: identifies the number of samples to be collected.

Triggering / Qualifier Codes

Trigger or qualify never (ignore)	0
Trigger or qualify on rising edge	1
Trigger or qualify on falling edge	2
Trigger or qualify on either edge	3
Trigger or qualify on high condition	4
Trigger or qualify on low condition	5
Trigger or qualify unconditionally (always)	6
Trigger or qualify unconditionally (always)	7

Example C Usage:

```
int iResult = S_OK;
short iDeviceNo = 0;
unsigned char ucTrigger[4] = {0,0,0,0};
unsigned char ucTrigAnd = FALSE;
unsigned char ucQualifier[4] = {0,0,0,0};
unsigned char ucQualAnd = FALSE;
unsigned long ulNumSamples = 0;

iResult = PCI3E_GetTimeBasedLogSettings(iDeviceNo, ucTrigger, &ucTrigAnd,
```

```
      ucQualifier, &ucQualAnd, &ulNumSamples);
if ( iResult != S_OK ){ // Handle error... }
```

VB Language Function Declaration:

```
Public Declare Function PCI3E_GetTimeBasedLogSettings Lib "USD_PCI_3E.dll"
(ByVal iDeviceNo As Integer, ByRef pucTrigger As Byte, ByRef pucTrigAnd As
Byte, ByRef pucQualifier As Byte, ByRef pucQualAnd As Byte, ByRef
pulNumberOfSamples As Long) As Long
```

Example VB Usage:

```
Dim errCode As Long
Dim iDeviceNo As Integer
Dim bytTrigger(3) As Byte
Dim bytTriggerAnd As Byte
Dim bytQualifier(3) As Byte
Dim bytQualifierAnd As Byte
Dim lNumberOfSamples As Long

iDeviceNo = 0

errCode = PCI3E_GetTimeBasedLogSettings(iDeviceNo, bytTrigger(0),
bytTriggerAnd, bytQualifier(0), bytQualifierAnd, lNumberOfSamples)
    If errCode <> 0 then
      ‘ Handle error..
End If
```

9.4.34 PCI3E_GetTimeStamp

Description:

This function writes to the CMD_Register which causes the TimeStamp counter to be latched to the TimeStamp Latch and then reads the TimeStamp Latch. Refer to the ReadTimeStamp function to simply read the TimeStamp Latch without causing the TimeStamp counter to be transferred to the TimeStamp Latch.

C Language Function Prototype:

```
int _stdcall PCI3E_GetTimeStamp(short iDeviceNo, unsigned long *pulVal);
```

Returns:

Result code as an integer: See error code section for values other than zero. Zero implies function call is successful.

Parameters:

`iDeviceNo`: identifies the PCI-3E card (zero based).
`pulVal`: in-out parameter containing the TimeStamp Latch value.

Example C Usage:

```
int iResult = S_OK;
short iDeviceNo = 0;
unsigned long ulVal = 0;

iResult = PCI3E_GetTimeStamp(iDeviceNo, &ulVal);
if ( iResult != S_OK ){ // Handle error...}
```

VB Language Function Declaration:

```
Public Declare Function PCI3E_GetTimeStamp Lib "USD_PCI_3E.dll" (ByVal
iDeviceNo As Integer, ByRef pulVal As Long) As Long
```

Example VB Usage:

```
Dim errCode As Long
Dim iDeviceNo As Integer
Dim lVal As Long

iDeviceNo = 0

errCode = PCI3E_GetTimeStamp(iDeviceNo, lVal)
If errCode <> 0 then
      ‘ Handle error…
End If
```

9.4.35 PCI3E_GetTriggerOnDecrease

Description:

This function retrieves a boolean value that indicates whether a trigger signal is generated when the count decreases.

C Language Function Prototype:

```
int _stdcall PCI3E_GetTriggerOnDecrease(short iDeviceNo, short iEncoder, BOOL
*pbVal);
```

Returns:

Result code as an integer: See error code section for values other than zero. Zero implies function call is successful.

Parameters:

`iDeviceNo`: identifies the PCI-3E card (zero based).
`iEncoder`: identifies the encoder channel (zero based).
`pbVal`: in out parameters. See description above.

Example C Usage:

```
int iResult = S_OK;
short iDeviceNo = 0;
short iEncoder = 0;
BOOL bVal = False;

iResult = PCI3E_GetTriggerOnDecrease(iDeviceNo, iEncoder, &bVal);
if ( iResult != S_OK ){ // Handle error... }
```

VB Language Function Declaration:

```
Public Declare Function PCI3E_GetTriggerOnDecrease Lib "USD_PCI_3E.dll" (ByVal
iDeviceNo As Integer, ByVal iEncoder As Integer, ByRef pbVal As Long) As Long
```

Example VB Usage:

```
Dim errCode As Long
Dim iDeviceNo As Integer
Dim iEncoder As Integer
Dim bVal As Long

iDeviceNo = 0
iEncoder = 0

errCode = PCI3E_GetTriggerOnDecrease(iDeviceNo, iEncoder, bVal)
If errCode <> 0 then
      ‘ Handle error…
End If
```

9.4.36 PCI3E_GetTriggerOnIncrease

Description:

This function retrieves a boolean value that indicates whether a trigger signal is generated when a count increases.

C Language Function Prototype:

```
int _stdcall PCI3E_GetTriggerOnIncrease(short iDeviceNo, short iEncoder, BOOL
*pbVal);
```

Returns:

Result code as an integer: See error code section for values other than zero. Zero implies function call is successful.

Parameters:

`iDeviceNo`: identifies the PCI-3E card (zero based).
`iEncoder`: identifies the encoder channel (zero based).
`pbVal`: in-out boolean parameter. See description above.

Example C Usage:

```
int iResult = S_OK;
short iDeviceNo = 0;
short iEncoder = 0;
BOOL bVal = False;

iResult = PCI3E_GetTriggerOnIncrease(iDeviceNo, iEncoder, &bVal);
if ( iResult != S_OK ){ // Handle error... }
```

VB Language Function Declaration:

```
Public Declare Function PCI3E_GetTriggerOnIncrease Lib "USD_PCI_3E.dll" (ByVal
iDeviceNo As Integer, ByVal iEncoder As Integer, ByRef pbVal As Long) As Long
```

Example VB Usage:

```
Dim errCode As Long
Dim iDeviceNo As Integer
Dim iEncoder As Integer
Dim bVal As Long

iDeviceNo = 0
iEncoder = 0

errCode = PCI3E_GetTriggerOnIncrease(iDeviceNo, iEncoder, bVal)
If errCode <> 0 then
      ‘ Handle error…
End If
```

9.4.37 PCI3E_GetTriggerOnIndex

Description:

This function retrieves a boolean value that indicates whether a trigger signal is generated when the edge of index is detected.

See also `PCI3E_SetEnableIndex`, `PCI3E_SetInvertIndex` and, `PCI3E_SetPresetOnIndex`.

C Language Function Prototype:

```
int _stdcall PCI3E_GetTriggerOnIndex(short iDeviceNo, short iEncoder, BOOL
*pbVal);
```

Returns:

Result code as an integer: See error code section for values other than zero. Zero implies function call is successful.

Parameters:

`iDeviceNo`: identifies the PCI-3E card (zero based).
`iEncoder`: identifies the encoder channel (zero based).
`pbVal`: in-out boolean parameter. See description above.

Example C Usage:

```
int iResult = S_OK;
short iDeviceNo = 0;
short iEncoder = 0;
BOOL bVal = False;

iResult = PCI3E_GetTriggerOnIndex(iDeviceNo, iEncoder, &bVal);
if ( iResult != S_OK ){ // Handle error... }
```

VB Language Function Declaration:

```
Public Declare Function PCI3E_GetTriggerOnIndex Lib "USD_PCI_3E.dll" (ByVal
iDeviceNo As Integer, ByVal iEncoder As Integer, ByRef pbVal As Long) As Long
```

Example VB Usage:

```
Dim errCode As Long
Dim iDeviceNo As Integer
Dim iEncoder As Integer
Dim bVal As Long

iDeviceNo = 0
iEncoder = 0

errCode = PCI3E_GetTriggerOnIndex(iDeviceNo, iEncoder, bVal)
If errCode <> 0 then
      ‘ Handle error…
End If
```

9.4.38 PCI3E_GetTriggerOnMatch

Description:

This function retrieves a boolean value that indicates whether a trigger signal is generated when count = match.

C Language Function Prototype:

```
int _stdcall PCI3E_GetTriggerOnMatch(short iDeviceNo, short iEncoder, BOOL
*pbVal);
```

Returns:

Result code as an integer: See error code section for values other than zero. Zero implies function call is successful.

Parameters:

`iDeviceNo`: identifies the PCI-3E card (zero based).
`iEncoder`: identifies the encoder channel (zero based).
`pbVal`: in out boolean parameter. See description above.

Example C Usage:

```
int iResult = S_OK;
short iDeviceNo = 0;
short iEncoder = 0;
BOOL bVal = FALSE;

iResult = PCI3E_GetTriggerOnMatch(iDeviceNo, iEncoder, &bVal);
if ( iResult != S_OK ){ // Handle error... }
```

VB Language Function Declaration:

```
Public Declare Function PCI3E_GetTriggerOnMatch Lib "USD_PCI_3E.dll" (ByVal
iDeviceNo As Integer, ByVal iEncoder As Integer, ByRef pbVal As Long) As Long
```

Example VB Usage:

```
Dim errCode As Long
Dim iDeviceNo As Integer
Dim iEncoder As Integer
Dim bVal As Long

iDeviceNo = 0
iEncoder = 0

errCode = PCI3E_GetTriggerOnMatch(iDeviceNo, iEncoder, bVal)
If errCode <> 0 then
      ‘ Handle error…
End If
```

9.4.39 PCI3E_GetTriggerOnRollover

Description:

This function retrieves a boolean value that indicates whether a trigger signal is generated when rolling over N-1 to 0 in modulo-N mode.

C Language Function Prototype:

```
int _stdcall PCI3E_GetTriggerOnRollover(short iDeviceNo, short iEncoder, BOOL
*pbVal);
```

Returns:

Result code as an integer: See error code section for values other than zero. Zero implies function call is successful.

Parameters:

`iDeviceNo`: identifies the PCI-3E card (zero based).
`iEncoder`: identifies the encoder channel (zero based).
`pbVal`: in out parameters. See description above.

Example C Usage:

```
int iResult = S_OK;
short iDeviceNo = 0;
short iEncoder = 0;
BOOL bVal = False;

iResult = PCI3E_GetTriggerOnRollover(iDeviceNo, iEncoder, &bVal);
if ( iResult != S_OK ){ // Handle error... }
```

VB Language Function Declaration:

```
Public Declare Function PCI3E_GetTriggerOnRollover Lib "USD_PCI_3E.dll" (ByVal
iDeviceNo As Integer, ByVal iEncoder As Integer, ByRef pbVal As Long) As Long
```

Example VB Usage:

```
Dim errCode As Long
Dim iDeviceNo As Integer
Dim iEncoder As Integer
Dim bVal As Long

iDeviceNo = 0
iEncoder = 0

errCode = PCI3E_GetTriggerOnRollover(iDeviceNo, iEncoder, bVal)
If errCode <> 0 then
      ‘ Handle error…
End If
```

9.4.40 PCI3E_GetTriggerOnRollunder

Description:

This function retrieves a boolean value that indicates whether a trigger signal is generated when rolling under 0 to N-1 in modulo-N mode.

C Language Function Prototype:

```
int _stdcall PCI3E_GetTriggerOnRollunder(short iDeviceNo, short iEncoder, BOOL
*pbVal);
```

Returns:

Result code as an integer: See error code section for values other than zero. Zero implies function call is successful.

Parameters:

`iDeviceNo`: identifies the PCI-3E card (zero based).
`iEncoder`: identifies the encoder channel (zero based).
`pbVal`: in-out parameters. See description above.

Example C Usage:

```
int iResult = S_OK;
short iDeviceNo = 0;
short iEncoder = 0;
BOOL bVal = False;

iResult = PCI3E_GetTriggerOnRollunder(iDeviceNo, iEncoder, &bVal);
if ( iResult != S_OK ){ // Handle error... }
```

VB Language Function Declaration:

```
Public Declare Function PCI3E_GetTriggerOnRollunder Lib "USD_PCI_3E.dll" (ByVal
iDeviceNo As Integer, ByVal iEncoder As Integer, ByRef pbVal As Long) As Long
```

Example VB Usage:

```
Dim errCode As Long
Dim iDeviceNo As Integer
Dim iEncoder As Integer
Dim bVal As Long

iDeviceNo = 0
iEncoder = 0

errCode = PCI3E_GetTriggerOnRollunder(iDeviceNo, iEncoder, bVal)
If errCode <> 0 then
      ‘ Handle error…
End If
```

9.4.41 PCI3E_GetTriggerOnZero

Description:

This function retrieves a boolean value that indicates whether a trigger signal is generated when count = 0.

C Language Function Prototype:

```
int _stdcall PCI3E_GetTriggerOnZero(short iDeviceNo, short iEncoder, BOOL
*pbVal);
```

Returns:

Result code as an integer: See error code section for values other than zero. Zero implies function call is successful.

Parameters:

`iDeviceNo`: identifies the PCI-3E card (zero based).
`iEncoder`: identifies the encoder channel (zero based).
`bVal`: in-out parameters. See description above.

Example C Usage:

```
int iResult = S_OK;
short iDeviceNo = 0;
short iEncoder = 0;
BOOL bVal = False;

iResult = PCI3E_GetTriggerOnZero(iDeviceNo, iEncoder, &bVal);
if ( iResult != S_OK ){ // Handle error... }
```

VB Language Function Declaration:

```
Public Declare Function PCI3E_GetTriggerOnZero Lib "USD_PCI_3E.dll" (ByVal
iDeviceNo As Integer, ByVal iEncoder As Integer, ByRef pbVal As Long) As Long
```

Example VB Usage:

```
Dim errCode As Long
Dim iDeviceNo As Integer
Dim iEncoder As Integer
Dim bVal As Long

iDeviceNo = 0
iEncoder = 0

errCode = PCI3E_GetTriggerOnZero(iDeviceNo, iEncoder, bVal)
If errCode <> 0 then
      ‘ Handle error…
End If
```

9.4.42 PCI3E_GetVersion

Description:
This function retrieves the version number associated with a specified device.

C Language Function Prototype:
```
int _stdcall PCI3E_GetVersion(short iDeviceNo, short *piVal);
```

Returns:
Result code as an integer: See error code section for values other than zero. Zero implies function call is successful.

Parameters:
`iDeviceNo`: identifies the PCI-3E card (zero based).
`piVal`: identifies the version number of the PCI-3E card.

Example C Usage:
```
int iResult = S_OK;
short iDeviceNo = 0;
int iVal = 0;
iResult = PCI3E_GetVersion(iDeviceNo, &iVal);
if ( iResult != S_OK ){ // Handle error...}
```

VB Language Function Declaration:
```
Public Declare Function PCI3E_GetVersion Lib "USD_PCI_3E.dll" (ByVal iDeviceNo
As Integer, ByRef piVal As Integer) As Long
```

Example VB Usage:
```
Dim errCode As Long
Dim iDeviceNo As Integer
Dim iVal As Integer

iDeviceNo = 0

errCode = PCI3E_GetVersion(iDeviceNo, iVal)
If errCode <> 0 then
      ‘ Handle error..
End If
```

9.4.43 PCI3E_Initialize

Description:

This function is used to open a connection with all installed and detected PCI-3E encoder interface cards. This function returns the number of cards detected in the in-out parameter piDeviceCount. This function must be called before any other function. Almost all other function calls require a device number. If there are two boards detected, then the first board will be device number 0 and the second device number 1.

This function sets the control registers and the preset registers of all channels to the following values:

The Control Register is loaded with 0x074000

Description	Master enable: enable	bit 18 = `1',
	Counter mode: modulo-N	bit 17 = `1', bit 16 = `1'
	Quadrature mode: X1	bit 15 = `0', bit 14 = `1'

The Preset Register is loaded with 0x0001F3 (499 in decimal) for an encoder with 500 CPR.

After PCI3E_Initialize is called, functions in "Counter Set-up Group" (See 9.3 User friendly functions) can be used to change the configuration if needed.

C Language Function Prototype:

```
int _stdcall PCI3E_Initialize(short *piDeviceCount);
```

Returns:

Result code as an integer: See error code section for values other than zero. Zero implies function call is successful.

Parameters:

`piDeviceCount`: an in-out parameter used to return the number of boards detected.

Example C Usage:

```
int iResult = 0;
short iDeviceCount = 0;

iResult = PCI3E_Initialize(&iDeviceCount);
if ( iResult != S_OK ){ // Handle error...}
```

VB Language Function Declaration:

```
Public Declare Function PCI3E_Initialize Lib "USD_PCI_3E.dll" (ByRef
piDeviceCount As Integer) As Long
```

Example VB Usage:

```
Dim errCode As Long
Dim iDeviceCount As Integer

errCode = PCI3E_Initialize(iDeviceCount)
```

```
If errCode <> 0 then
      ‘ Handle error..
End If
```

9.4.44 PCI3E_PresetCount

Description:
This function sets a channel's counter to its preset register value.

C Language Function Prototype:
```
int _stdcall PCI3E_PresetCount(short iDeviceNo, short iEncoder);
```

Returns:
Result code as an integer: See error code section for values other than zero. Zero implies function call is successful.

Parameters:
`iDeviceNo`: identifies the PCI-3E card (zero based).
`iEncoder`: identifies the encoder channel (zero based).

Example C Usage:
```
int iResult = S_OK;
short iDeviceNo = 0;
short iEncoder = 0;

iResult = PCI3E_PresetCount(iDeviceNo, iEncoder);
if ( iResult != S_OK ){ // Handle error...}
```

VB Language Function Declaration:
```
Public Declare Function PCI3E_PresetCount Lib "USD_PCI_3E.dll" (ByVal iDeviceNo
As Integer, ByVal iEncoder As Integer) As Long
```

Example VB Usage:
```
Dim errCode As Long
Dim iDeviceNo As Integer
Dim iEncoder As Integer

iDeviceNo = 0
iEncoder = 0

errCode = PCI3E_PresetCount(iDeviceNo, iEncoder)
If errCode <> 0 then
      ‘ Handle error..
End If
```

9.4.45 PCI3E_ReadFIFOBuffer

Description:

This function reads the FIFO buffer records and copies the data into user allocated arrays. The user is responsible for creating the arrays and passing their pointer to this function.

The piSize parameter identifies the number of records to read. Each of the allocated arrays must be at least piSize in length.

If the specified number of records is greater than the number of records in the FIFO buffer, then only the records in the FIFO buffer are read and copied. The piSize parameter will be changed to the number of records that were copied.

This function returns when (1) the number of records read equals piSize, or (2) the FIFO buffer is empty, or (3) the FIFO buffer is full (possible overflow).

C Language Function Prototype:

```
int _stdcall PCI3E_ReadFIFOBuffer(short iDeviceNo,
                                  short *piSize,
                                  unsigned long *pTime,
                                  unsigned long *pCount0,
                                  unsigned long *pCount1,
                                  unsigned long *pCount2,
                                  unsigned char *pStatus0,
                                  unsigned char *pStatus1,
                                  unsigned char *pStatus2,
                                  unsigned char *pInput);
```

Returns:

Result code as an integer: See error code section for values other than zero. Zero implies function call is successful.

Parameters:

`iDeviceNo`: identifies the PCI-3E card (zero based).
`piSize`: refer to description above.
`pTime`: an array of TimeStamp.
`pCount0`: an array of encoder channel 0 counts.
`pCount1`: an array of encoder channel 1 counts.
`pCount2`: an array of encoder channel 2 counts.
`pStatus0`: an array of encoder channel 0 status codes.
`pStatus1`: an array of encoder channel 1 status codes.
`pStatus2`: an array of encoder channel 2 status codes.
`pInput`: an array of input port register values.

Bit 7: last direction----------from bit 23 of Status reg.
Bit 6: latched_retard_detected----------from bit 13 of Status reg.
Bit 5: latched_advance_detected--------from bit 12 of Status reg.
Bit 4: latched_index_detected----------from bit 11 of Status reg.
Bit 3: latched_borrow_detected---------from bit 10 of Status reg.

Bit 2: latched_carry_detected----------from bit 9 of Status reg.
Bit 1: latched_match_detected----------from bit 8 of Status reg.
Bit 0: latched_zero_detected----------from bit 7 of Status reg.

Example C Usage:

```
int iResult = S_OK;
short iDeviceNo = 0;
short iSize = PCI3E_FIFO_MAX_COUNT;
unsigned long Time[PCI3E_FIFO_MAX_COUNT];
unsigned long Count0[PCI3E_FIFO_MAX_COUNT];
unsigned long Count1[PCI3E_FIFO_MAX_COUNT];
unsigned long Count2[PCI3E_FIFO_MAX_COUNT];
unsigned char Status0[PCI3E_FIFO_MAX_COUNT];
unsigned char Status1[PCI3E_FIFO_MAX_COUNT];
unsigned char Status2[PCI3E_FIFO_MAX_COUNT];
unsigned char Input[PCI3E_FIFO_MAX_COUNT];

iResult = PCI3E_ReadFIFOBuffer(iDeviceNo, &iSize, Time,
                               Count0, Count1, Count2,
                               Status0, Status1, Status2,
                               Input);
if ( iResult != S_OK ){ // Handle error...}
```

VB Language Function Declaration:

```
Public Declare Function PCI3E_ReadFIFOBuffer Lib "USD_PCI_3E.dll" ( _
      ByVal iDeviceNo As Integer, _
      ByRef piSize As Integer, _
      ByRef pCount0 As Long, _
      ByRef pCount1 As Long, _
      ByRef pCount2 As Long, _
      ByRef pStatus0 As Byte, _
      ByRef pStatus1 As Byte, _
      ByRef pStatus2 As Byte, _
      ByRef pInput As Byte)
```

Example VB Usage:

```
Dim errCode                         As Long
Dim iDeviceNo                       As Integer
Dim iSize                           As Integer
Dim Count0(PCI3E_FIFO_MAX_COUNT-1)  As Long
Dim Count1(PCI3E_FIFO_MAX_COUNT-1)  As Long
Dim Count2(PCI3E_FIFO_MAX_COUNT-1)  As Long
Dim Status0(PCI3E_FIFO_MAX_COUNT-1) As Byte
Dim Status1(PCI3E_FIFO_MAX_COUNT-1) As Byte
Dim Status2(PCI3E_FIFO_MAX_COUNT-1) As Byte
Dim Input(PCI3E_FIFO_MAX_COUNT-1)   As Byte

iSize = PCI3E_FIFO_MAX_COUNT
iDeviceNo = 0

errCode = PCI3E_ReadFIFOBuffer(iDeviceNo, iSize, Time(0),
                               Count0(0), Count1(0), Count2(0),
                               Status0(0), Status1(0), Status2(0),
                               Input(0))
If errCode <> 0 then
      ‘ Handle error..
End If
```

9.4.46 PCI3E_ReadFIFOBufferStruct

Description:

This function reads the FIFO buffer records and copies the data into the user allocated array of `PCI3E_FIFOBufferRecord` stucture. The user is responsible for creating the array and passing it's pointer to this function.

The piSize parameter identifies the number of records to read. The allocated array of `PCI3E_FIFOBufferRecord` structure must be at least piSize in length.

If the specified number of records are greater than the number of records in the FIFO buffer, then only the records in the FIFO buffer are read and copied. The piSize parameter will be changed to the number of records that were copied.

This function returns when (1) the number of records read equals piSize, or (2) the FIFO buffer is empty, or (3) the FIFO buffer is full (possible overflow).

C Language Function Prototype:

```
int _stdcall PCI3E_ReadFIFOBuffer(short iDeviceNo, short *piSize,
PCI3E_FIFOBufferRecord *pCBR);
```

Returns:

Result code as an integer: See error code section for values other than zero. Zero implies function call is successful.

Parameters:

`iDeviceNo`: identifies the PCI-3E card (zero based).
`piSize`: refer to description above.
`pCBR`: an array of `PCI3E_FIFOBufferRecord`.

C – Definition of Channel Buffer Record	VB Definition of Channel Buffer Record

`struct PCI3E_FIFOBufferRecord`
`{`
`  unsigned long Time;`
`  unsigned long Count[3];`
`  unsigned char Status[3];`
`  unsigned char Input;`
`};`
	`Public Type`
	`PCI3E_FIFOBufferRecord`
	`  Time As Long`
	`  Count(2) As Long`
	`  Status(2) As Byte`
	`  Input As Byte`
	`End Type`

Example C Usage:

```
int iResult = S_OK;
short iDeviceNo = 0;
short iSize = PCI3E_FIFO_MAX_COUNT;
PCI3E_FIFOBufferRecord cbr[PCI3E_FIFO_MAX_COUNT];

iResult = PCI3E_ReadFIFOBuffer(iDeviceNo, &iSize, cbr);
if ( iResult != S_OK ){ // Handle error...}
```

VB Language Function Declaration:

```
Public Declare Function PCI3E_ReadFIFOBuffer Lib "USD_PCI_3E.dll" (ByVal
iDeviceNo As Integer, ByRef piSize As Integer, ByRef pCBR As
PCI3E_FIFOBufferRecord) As Long
```

Example VB Usage:

```
Dim errCode   As Long
Dim iDeviceNo As Integer
Dim iSize     As Integer
Dim cbr(PCI3E_FIFO_MAX_COUNT -1) As PCI3E_FIFOBufferRecord
iSize = FIFO_MAX_COUNT
iDeviceNo = 0

errCode = PCI3E_ReadFIFOBuffer(iDeviceNo, iSize, cbr)
If errCode <> 0 then
      ‘ Handle error..
End If
```

9.4.47 PCI3E_ReadInputPortRegister

Description:
This function returns the value stored in the input port register.

C Language Function Prototype:
```
int _stdcall PCI3E_ReadInputPortRegister(short iDeviceNo, unsigned char
*pucVal);
```

Returns:
Result code as an integer: See error code section for values other than zero. Zero implies function call is successful.

Parameters:
`iDeviceNo`: identifies the PCI-3E card (zero based).
`pucVal`: in-out parameter containing the value read from the input port register.

Example C Usage:
```
int iResult = S_OK;
short iDeviceNo = 0;
unsigned char ucVal;

iResult = PCI3E_ReadInputPortRegister(iDeviceNo, &ucVal);
if( iResult != S_OK ){  // Handle error...}
```

VB Language Function Declaration:
```
Public Declare Function PCI3E_ReadInputPortRegister Lib "USD_PCI_3E.dll" (ByVal
iDeviceNo As Integer, ByRef pucVal As Byte) As Long
```

Example VB Usage:
```
Dim errCode As Long
Dim iDeviceNo As Integer
Dim bytVal As Byte

iDeviceNo = 0

errCode =  PCI3E_ReadInputPortRegister(iDeviceNo, bytVal)
If errCode <> 0 Then
      ‘ Handle error...
End If
```

9.4.48 PCI3E_ReadOutputLatch

Description:
This function returns the value from Output Latch Register.

C Language Function Prototype:
```
int _stdcall PCI3E_ReadOutputLatch(short iDeviceNo, short iEncoder, unsigned
long *pulVal);
```

Returns:
Result code as an integer: See error code section for values other than zero. Zero implies function call is successful.

Parameters:
`iDeviceNo`: identifies the PCI-3E card (zero based).
`iEncoder`: identifies the encoder channel (zero based).
`pulVal`: in-out parameter that contains the Output Latch Register value.

Example C Usage:
```
int iResult = S_OK;
short iDeviceNo = 0;
short iEncoder = 0;
unsigned long ulVal = 0;

iResult = PCI3E_ReadOutputLatch(iDeviceNo, iEncoder, &ulVal);
if ( iResult != S_OK ){ // Handle error...}
```

VB Language Function Declaration:
```
Public Declare Function PCI3E_ReadOutputLatch Lib "USD_PCI_3E.dll" (ByVal
iDeviceNo As Integer, ByVal iEncoder As Integer, ByRef pulVal As Long) As Long
```

Example VB Usage:
```
Dim errCode As Long
Dim iDeviceNo As Integer
Dim iEncoder As Integer
Dim lVal As Long


iDeviceNo = 0
iEncoder = 0

errCode = PCI3E_ReadOutputLatch(iDeviceNo, iEncoder, lVal)
If errCode <> 0 then
      ‘ Handle error..
End If
```

9.4.49 PCI3E_ReadOutputPortRegister

Description:
This function returns the value stored in the output port register.

C Language Function Prototype:
```
int _stdcall PCI3E_ReadOutputPortRegister(short iDeviceNo, unsigned char
*pucVal);
```

Returns:
Result code as an integer: See error code section for values other than zero. Zero implies function call is successful.

Parameters:
`iDeviceNo`: identifies the PCI-3E card (zero based).
`pucVal`: in-out parameter containing value read from the output port register.

Bit 7-4: always 0
Bit 3: Output Port – OUT3
Bit 2: Output Port – OUT2
Bit 1: Output Port – OUT1
Bit 0: Output Port – OUT0

Example C Usage:
```
int iResult = S_OK;
short iDeviceNo = 0;
unsigned char ucVal;

iResult = PCI3E_ReadOutputPortRegister(iDeviceNo, &ucVal);
if( iResult != S_OK ){  // Handle error...}
```

VB Language Function Declaration:
```
Public Declare Function PCI3E_ReadOutputPortRegister Lib "USD_PCI_3E.dll"
(ByVal iDeviceNo As Integer, ByRef pucVal As Byte) As Long
```

Example VB Usage:
```
Dim errCode As Long
Dim iDeviceNo As Integer
Dim bytVal As Byte

iDeviceNo = 0

errCode =  PCI3E_ReadOutputPortRegister(iDeviceNo, bytVal)
If errCode <> 0 Then
      ‘ Handle error...
End If
```

9.4.50 PCI3E_ReadRegister

Description:
This function returns the value stored in a specified PCI-3E register.

C Language Function Prototype:
```
int _stdcall PCI3E_ReadRegister(short iDeviceNo, short iRegister, unsigned long *pulVal);
```

Returns:
Result code as an integer: See error code section for values other than zero. Zero implies function call is successful.

Parameters:
`iDeviceNo`: identifies the PCI-3E card (zero based).
`iRegister`: identifies the specific register to read. Valid registers are 0 – 47.
`pulVal`: in-out parameter containing value read from the specified register.

Example C Usage:
```
int iResult = S_OK;
short iDeviceNo = 0;
short iRegister = 0;
unsigned long ulVal = 0;

iResult = PCI3E_ReadRegister(iDeviceNo, iRegister, &ulVal);
if( iResult != S_OK ){  // Handle error...}
```

VB Language Function Declaration:
```
Public Declare Function PCI3E_ReadRegister Lib "USD_PCI_3E.dll" (ByVal
iDeviceNo As Integer, ByVal iRegister As Integer, ByRef pulVal As Long) As Long
```

Example VB Usage:
```
Dim errCode As Long
Dim iDeviceNo As Integer
Dim iRegister As Integer
Dim lVal As Long

iDeviceNo = 0
iRegister = 0

errCode =  PCI3E_ReadRegister(iDeviceNo, iRegister, lVal)
If errCode <> 0 Then
      ‘ Handle error...
End If
```

9.4.51 PCI3E_ReadTimeAndCounts

Description:
This function reads the TimeStamp Latch and each encoder's Output Latch.

C Language Function Prototype:
```
int _stdcall PCI3E_ReadTimeAndCounts(short iDeviceNo, unsigned long *pulCounts,
unsigned long *pulTimeStamp);
```

Returns:
Result code as an integer: See error code section for values other than zero. Zero implies function call is successful.

Parameters:
`iDeviceNo`: identifies the PCI-3E card (zero based).
`pulCounts`: array of 3 longs containing the Output Latch value for each channel.
`pulTimeStamp`: in-out parameter containing the TimeStamp Latch value.

Example C Usage:
```
int iResult = S_OK;
short iDeviceNo = 0;
unsigned long ulCounts [3] = {0, 0, 0};
unsigned long ulTimeStamp = 0;

iResult = PCI3E_ReadTimeAndCounts(iDeviceNo, ulCounts, &ulTimeStamp);
if ( iResult != S_OK ){ // Handle error...}
```

VB Language Function Declaration:
```
Public Declare Function PCI3E_ReadTimeAndCounts Lib "USD_PCI_3E.dll" (ByVal
iDeviceNo As Integer, ByRef pulCounts As Long, ByRef pulTimeStamp As Long) As
Long
```

Example VB Usage:
```
Dim errCode As Long
Dim iDeviceNo As Integer
Dim lCounts(2) As Long
Dim lTimeStamp As Long

iDeviceNo = 0

errCode = PCI3E_ReadTimeAndCounts(iDeviceNo, lCounts(0), lTimeStamp)
If errCode <> 0 then
      ‘ Handle error…
End If
```

9.4.52 PCI3E_ReadTimeStamp

Description:
This function reads the TimeStamp Latch register.

C Language Function Prototype:
```
int _stdcall PCI3E_ReadTimeStamp(short iDeviceNo, unsigned long *pulVal);
```

Returns:
Result code as an integer: See error code section for values other than zero. Zero implies function call is successful.

Parameters:
`iDeviceNo`: identifies the PCI-3E card (zero based).
`pulVal`: in-out parameter containing the TimeStamp Latch value.

Example C Usage:
```
int iResult = S_OK;
short iDeviceNo = 0;
unsigned long ulVal = 0;

iResult = PCI3E_ReadTimeStamp(iDeviceNo, &ulVal);
if ( iResult != S_OK ){ // Handle error...}
```

VB Language Function Declaration:
```
Public Declare Function PCI3E_ReadTimeStamp Lib "USD_PCI_3E.dll" (ByVal
iDeviceNo As Integer, ByRef pulVal As Long) As Long
```

Example VB Usage:
```
Dim errCode As Long
Dim iDeviceNo As Integer
Dim lVal As Long

iDeviceNo = 0

errCode = PCI3E_ReadTimeStamp(iDeviceNo, lVal)
If errCode <> 0 then
      ‘ Handle error…
End If
```

9.4.53 PCI3E_RegisterInterruptHandler

Description:

This function takes a pointer to a callback function that is called when an interrupt is detected.

Below is an example implementation of an interrupt callback function, where the function name may be changed to user's preference:

C code:

```
void __stdcall InterruptHandler(short iDeviceNo)
{
      usigned long ulCounts[3] = {0,0,0};
      unsigned long ulTimeStamp = 0;
      short iFIFOCount = 0;
      int i = 0;

      PCI3E_FIFOBufferRecord cbr[PCI3E_FIFO_MAX_COUNT];

      // determine if interrupt came from FIFO Half-Full
      // or encoder trigger-out
      if (m_bFIFOEnabled) {
            printf("*** Half-Full Interrupt ***\n");
            iFIFOCount = PCI3E_FIFO_MAX_COUNT;
            while(iFIFOCount)
            {
                  iFIFOCount = PCI3E_FIFO_MAX_COUNT;
                  // iFIFOCount gets number of records copied.
                  PCI3E_ReadFIFOBuffer(iDeviceNo, &iFIFOCount, cbr);
                  for (i = 0; i < iFIFOCount; i++) {
                        printf("%u\t%d\t%d\t%d\t%d\n", cbr[i].Time,
                        cbr[i].Count[0],
                        cbr[i].Count[1],
                        cbr[i].Count[2],
                        cbr[i].Count[3]);
                  }
            }
      } else {
            // Get the encoder counts and timestamp.
            PCI3E_ReadTimeAndCounts (iDeviceNo, ulCounts, &ulTimeStamp);

            // Clear captured status for each encoder
            for (i = 0; i<PCI3E_MAX_ENCODERS; i++) {
                  PCI3E_ClearCapturedStatus(iDeviceNo, i);
            }

            printf("%u\t%d\t%d\t%d\n", ulTimeStamp, ulCounts[0],
                   ulCounts[1], ulCounts[2]);
      }
}
```

VB code:

```
' The VB interrupt handler should be place in a VB module.
Public Sub InterruptHandler (ByVal iDeviceNo As Integer)
      'TODO: Add code to handle interrupt
End Sub
```

C Language Function Prototype:

```
int _stdcall PCI3E_RegisterInterruptHandler(short iDeviceNo, unsigned long
ulInterruptHandler);
```

Returns:

Result code as an integer: See error code section for values other than zero. Zero implies function call is successful.

Parameters:

`iDeviceNo`: identifies the PCI-3E card (zero based).

`ulInterruptHandler:` void pointer to interrupt handler function.

Example C Usage:

```
int iResult = S_OK;
short iDeviceNo = 0;

iResult = PCI3E_RegisterInterruptHandler(iDeviceNo, (unsigned
long)InterruptHandler);
if ( iResult != S_OK ){ // Handle error...}
```

VB Language Function Declaration:

```
Public Declare Function PCI3E_RegisterInterruptHandler Lib "USD_PCI_3E.dll"
(ByVal iDeviceNo As Integer, ByVal ulInterruptHandler As Long) As Long
```

Example VB Usage:

```
Dim errCode As Long
Dim iDeviceNo As Integer

iDeviceNo = 0

errCode = PCI3E_RegisterInterruptHandler(iDeviceNo, AddressOf InterruptHandler)
‘ Refer to Public Sub InterruptHandler
If errCode <> 0 then
      ‘ Handle error..
End If
```

9.4.54 PCI3E_ResetCount

Description:
This function sets the counter value to zero.

C Language Function Prototype:
```
int _stdcall PCI3E_ResetCount(short iDeviceNo, short iEncoder);
```

Returns:
Result code as an integer: See error code section for values other than zero. Zero implies function call is successful.

Parameters:
`iDeviceNo`: identifies the PCI-3E card (zero based).
`iEncoder`: identifies the encoder channel (zero based).

Example C Usage:
```
int iResult = S_OK;
short iDeviceNo = 0;
short iEncoder = 0;

iResult = PCI3E_ResetCount(iDeviceNo, iEncoder);
if ( iResult != S_OK ){ // Handle error...}
```

VB Language Function Declaration:
```
Public Declare Function PCI3E_ResetCount Lib "USD_PCI_3E.dll" (ByVal iDeviceNo
As Integer, ByVal iEncoder As Integer) As Long
```

Example VB Usage:
```
Dim errCode As Long
Dim iDeviceNo As Integer
Dim iEncoder As Integer

iDeviceNo = 0
iEncoder = 0

errCode = PCI3E_ResetCount(iDeviceNo, iEncoder)
If errCode <> 0 then
      ‘ Handle error…
End If
```

9.4.55 PCI3E_ResetTimeStamp

Description:
This function sets the TimeStamp counter value to zero.

C Language Function Prototype:
```
int _stdcall PCI3E_ResetTimeStamp(short iDeviceNo);
```

Returns:
Result code as an integer: See error code section for values other than zero. Zero implies function call is successful.

Parameters:
`iDeviceNo`: identifies the PCI-3E card (zero based).

Example C Usage:
```
int iResult = S_OK;
short iDeviceNo = 0;

iResult = PCI3E_ResetTimeStamp(iDeviceNo);
if ( iResult != S_OK ){ // Handle error...}
```

VB Language Function Declaration:
```
Public Declare Function PCI3E_ResetTimeStamp Lib "USD_PCI_3E.dll" (ByVal
iDeviceNo As Integer) As Long
```

Example VB Usage:
```
Dim errCode As Long
Dim iDeviceNo As Integer

iDeviceNo = 0

errCode = PCI3E_ResetTimeStamp(iDeviceNo)
If errCode <> 0 then
      ‘ Handle error…
End If
```

9.4.56 PCI3E_SetCaptureEnabled

Description:

This function sets a boolean value that determines whether a trigger_in will cause a transfer from counter (accumulator) to output.

C Language Function Prototype:

```
int _stdcall PCI3E_SetCaptureEnabled(short iDeviceNo, short iEncoder, BOOL
bVal);
```

Returns:

Result code as an integer: See error code section for values other than zero. Zero implies function call is successful.

Parameters:

`iDeviceNo`: identifies the PCI-3E card (zero based).
`iEncoder`: identifies the encoder channel (zero based).
`bVal`: indicates whether a trigger_in will cause a transfer from accumulator to output.

Example C Usage:

```
int iResult = S_OK;
short iDeviceNo = 0;
short iEncoder = 0;
BOOL  bVal = TRUE;

iResult = PCI3E_SetCaptureEnabled(iDeviceNo, iEncoder, bVal);
if ( iResult != S_OK ){ // Handle error...}
```

VB Language Function Declaration:

```
Public Declare Function PCI3E_SetCaptureEnabled Lib "USD_PCI_3E.dll" (ByVal
iDeviceNo As Integer, ByVal iEncoder As Integer, ByVal bVal As Long) As Long
```

Example VB Usage:

```
Dim errCode As Long
Dim iDeviceNo As Integer
Dim iEncoder As Integer
Dim bVal as Long

iDeviceNo = 0
iEncoder = 0
bVal = True

errCode = PCI3E_SetCaptureEnabled(iDeviceNo, iEncoder, bVal)
if errCode <> 0 then
      ‘ Handle error…
End If
```

9.4.57 PCI3E_SetControlMode

Description:
This function sets the Control Register value which controls the operation of a channel.

C Language Function Prototype:
```
int _stdcall PCI3E_SetControlMode(short iDeviceNo, short iEncoder, unsigned
long ulVal);
```

Returns:
Result code as an integer: See error code section for values other than zero. Zero implies function call is successful.

Parameters:
`iDeviceNo`: identifies the PCI-3E card (zero based).
`iEncoder`: identifies the encoder channel (zero based).
`ulVal`: value containing the control mode. *See section 6.1 Control Registers.*

Example C Usage:
```
int iResult = S_OK;
short iDeviceNo = 0;
short iEncoder = 0;
unsigned long ulVal = 0xF4000; // Obtain the correct control mode.

iResult = PCI3E_SetControlMode (iDeviceNo, iEncoder, ulVal);
if ( iResult != S_OK ){ // Handle error...}
```

VB Language Function Declaration:
```
Public Declare Function PCI3E_SetControlMode Lib "USD_PCI_3E.dll" (ByVal
iDeviceNo As Integer, ByVal iEncoder As Integer, ByVal ulVal As Long) As Long
```

Example VB Usage:
```
Dim errCode As Long
Dim iDeviceNo As Integer
Dim iEncoder As Integer
Dim lVal As Long

iDeviceNo = 0
iEncoder = 0
lVal = &hF4000    ’ Obtain the correct control mode.

errCode = PCI3E_SetControlMode(iDeviceNo, iEncoder, lVal)
if errCode <> 0 then
      ‘ Handle error…
End If
```

9.4.58 PCI3E_SetCount

Description:
This function sets the count to a specified value.

Caveats: PCI3E_SetCount forces internal counter's value to a specified value without permanently changing the Preset register. In fact, PCI3E_SetCount utilizes the Preset Register for transferring data to the internal counter, but the original value of Preset Register is restored at the end of function call. When writing an application that always watches for changing of value of Preset Register, the programmer must be aware of this temporary change of value.

C Language Function Prototype:
```
int _stdcall PCI3E_SetCount(short iDeviceNo, short iEncoder, unsigned long
ulVal);
```

Returns:
Result code as an integer: See error code section for values other than zero. Zero implies function call is successful.

Parameters:
`iDeviceNo`: identifies the PCI-3E card (zero based).
`iEncoder`: identifies the encoder channel (zero based).
`ulVal`: the new value to be written to the counter register.

Example C Usage:
```
int iResult = S_OK;
short iDeviceNo = 0;
short iEncoder = 0;
unsigned long ulVal= 0; //  Note: choose your value here.

iResult = SetCount(iDeviceNo, iEncoder, ulVal);
if ( iResult != S_OK ){ // Handle error...}
```

VB Language Function Declaration:
```
Public Declare Function PCI3E_SetCount Lib "USD_PCI_3E.dll" (ByVal iDeviceNo As Integer,
ByVal iEncoder As Integer, ByVal ulVal As Long) As Long
```

Example VB Usage:
```
Dim errCode As Long
Dim iDeviceNo As Integer
Dim iEncoder As Integer
Dim lVal As Long

iDeviceNo = 0
iEncoder = 0
lVal = 0

errCode =  PCI3E_SetCount(iDeviceNo, iEncoder, lVal)
If errCode <> 0 then
       ‘ Handle error..
End If
```

9.4.59 PCI3E_SetCounterMode

Description:
This function sets the control counter mode that governs the counter behavior and limits. See parameters sections for description of the possible counter modes.

C Language Function Prototype:
```
int _stdcall PCI3E_SetCounterMode(short iDeviceNo, short iEncoder, short iVal);
```

Returns:
Result code as an integer: See error code section for values other than zero. Zero implies function call is successful.

Parameters:
`iDeviceNo`: identifies the PCI-3E card (zero based).
`iEncoder`: identifies the encoder channel (zero based).
`iVal`: 32 bit in-out parameter containing the counter mode.
0 = acc. acts like a 24 bit counter
1 = acc. uses preset register in range-limit mode
2 = acc. uses preset register in non-recycle mode
3 = acc. uses preset register in modulo-N mode.
See 6.1 Control Registers.

Example C Usage:
```
int iResult = S_OK;
short iDeviceNo = 0;
short iEncoder = 0;
short iVal = 0;

iResult = PCI3E_SetCounterMode(iDeviceNo, iEncoder, iVal);
if ( iResult != S_OK ){ // Handle error... }
```

VB Language Function Declaration:
```
Public Declare Function PCI3E_SetCounterMode Lib "USD_PCI_3E.dll" (ByVal
iDeviceNo As Integer, ByVal iEncoder As Integer, ByVal iVal As Integer) As Long
```

Example VB Usage:
```
Dim errCode As Long
Dim iDeviceNo As Integer
Dim iEncoder As Integer
Dim iVal As Integer

iDeviceNo = 0
iEncoder = 0

errCode = PCI3E_SetCounterMode(iDeviceNo, iEncoder, iVal)
If errCode <> 0 then
      ‘ Handle error..
End If
```

9.4.60 PCI3E_SetDigitalInputTriggerConfig

Description:
This function is used to configure the digital input trigger settings.

Note: Refer to section 6.6 Digital Input Triggering Registers.

C Language Function Prototype:
```
int _stdcall PCI3E_SetDigitalInputTriggerConfig(short iDeviceNo, BOOL
*pbEnableTrigger, BOOL *pbTriggerOnRisingEdge);
```

Returns:
Result code as an integer: See error code section for values other than zero. Zero implies function call is successful.

Parameters:
`iDeviceNo`: identifies the PCI-3E card (zero based).
`pbEnableTrigger`: an array of four booleans which determine if input signals can generate a trigger out signal.
`pbTriggerOnRisingEdge`: an array of four booleans which determine the type of tigger signal, rising or falling edge. 1 = rising edge, 0 = falling edge.

Example C Usage:
```
int iResult = S_OK;
int iDeviceNo = 0;

// enable trigger on input 0
BOOL bEnableTrigger[4] = {TRUE, FALSE, FALSE, FALSE};

// look for rising edge on input 0
BOOL bTriggerOnRisingEdge[4] = {TRUE, FALSE, FALSE, FALSE};

iResult = PCI3E_SetDigitalInputTriggerConfig(iDeviceNo, bEnableTrigger,
      bTriggerOnRisingEdge);
if ( iResult != S_OK ){ // Handle error...}
```

VB Language Function Declaration:
```
Public Declare Function PCI3E_SetDigitalInputTriggerConfig Lib "USD_PCI_3E.dll"
(ByVal iDeviceNo As Integer, ByRef bEnableTrigger As Long, ByRef
bTriggerOnRisingEdge As Long) As Long
```

Example VB Usage:
```
Dim errCode As Long
Dim iDeviceNo As Integer
Dim bEnableTrigger(3) As Long
Dim bTriggerOnRisingEdge (3) As Long

iDeviceNo = 0
bEnableTrigger(0) = True            ‘ enable trigger on input 0
bTriggerOnRisingEdge(0) = True      ‘look for rising edge on input 0

errCode = PCI3E_GetDigitalInputTriggerConfig (iDeviceNo, bEnableTrigger(0),
      bTriggerOnRisingEdge(0))
```

```
If errCode <> 0 then
      ‘ Handle error..
End If
```

9.4.61 PCI3E_SetEnableAccumulator

Description:

This function sets a boolean value that determines whether the master enable for accumulator is set, (must be set to true to count).

C Language Function Prototype:

```
int _stdcall PCI3E_SetEnableAccumulator(short iDeviceNo, short iEncoder, BOOL
bVal);
```

Returns:

Result code as an integer: See error code section for values other than zero. Zero implies function call is successful.

Parameters:

`iDeviceNo`: identifies the PCI-3E card (zero based).
`iEncoder`: identifies the encoder channel (zero based).
`bVal`: in-out boolean parameter identifying whether the counter is enabled.

Example C Usage:

```
int iResult = S_OK;
short iDeviceNo = 0;
short iEncoder = 0;
BOOL bVal = FALSE;

iResult = PCI3E_SetEnableAccumulator(iDeviceNo, iEncoder, bVal);
if ( iResult != S_OK ){ // Handle error... }
```

VB Language Function Declaration:

```
Public Declare Function PCI3E_SetEnableAccumulator Lib "USD_PCI_3E.dll" (ByVal
iDeviceNo As Integer, ByVal iEncoder As Integer, ByVal bVal As Long) As Long
```

Example VB Usage:

```
Dim errCode As Long
Dim iDeviceNo As Integer
Dim iEncoder As Integer
Dim bVal as Long

iDeviceNo = 0
iEncoder = 0
bVal = True

errCode = PCI3E_SetEnableAccumulator(iDeviceNo, iEncoder, bVal)
If errCode <> 0 then
      ‘ Handle error..
End If
```

9.4.62 PCI3E_SetEnableIndex

Description:

This function sets a boolean value that indicates whether index detection is enabled. When enabled, you can use the `PCI3E_SetPresetOnIndex` to determine how to respond to an index signal.

C Language Function Prototype:

```
int _stdcall PCI3E_SetEnableIndex(short iDeviceNo, short iEncoder, BOOL bVal);
```

Returns:

Result code as an integer: See error code section for values other than zero. Zero implies function call is successful.

Parameters:

`iDeviceNo`: identifies the PCI-3E card (zero based).
`iEncoder`: identifies the encoder channel (zero based).
`bVal`: in-out boolean parameter identifying whether the index is enabled.

Example C Usage:

```
int iResult = S_OK;
short iDeviceNo = 0;
short iEncoder = 0;
BOOL bVal = FALSE;

iResult = PCI3E_SetEnableIndex(iDeviceNo, iEncoder, bVal);
if ( iResult != S_OK ){ // Handle error... }
```

VB Language Function Declaration:

```
Public Declare Function PCI3E_SetEnableIndex Lib "USD_PCI_3E.dll" (ByVal
iDeviceNo As Integer, ByVal iEncoder As Integer, ByVal bVal As Long) As Long
```

Example VB Usage:

```
Dim errCode As Long
Dim iDeviceNo As Integer
Dim iEncoder As Integer
Dim bVal as Long

iDeviceNo = 0
iEncoder = 0
bVal = True

errCode = PCI3E_SetEnableIndex(iDeviceNo, iEncoder, bVal)
If errCode <> 0 then
      ‘ Handle error..
End If
```

9.4.63 PCI3E_SetForward

Description:

This function sets a boolean value that indicates whether the B input of quadrature signal is inverted. 1 = inverted, 0 = not inverted.

C Language Function Prototype:

```
int _stdcall PCI3E_SetForward(short iDeviceNo, short iEncoder, BOOL bVal);
```

Returns:

Result code as an integer: See error code section for values other than zero. Zero implies function call is successful.

Parameters:

`iDeviceNo`: identifies the PCI-3E card (zero based).
`iEncoder`: identifies the encoder channel (zero based).
`pbVal`: in-out boolean parameter identifying if the B signal is inverted or not.

Example C Usage:

```
int iResult = S_OK;
short iDeviceNo = 0;
short iEncoder = 0;
BOOL bVal = TRUE;

iResult = PCI3E_SetForward(iDeviceNo, iEncoder, bVal);
if ( iResult != S_OK ){ // Handle error... }
```

VB Language Function Declaration:

```
Public Declare Function PCI3E_SetForward Lib "USD_PCI_3E.dll" (ByVal iDeviceNo
As Integer, ByVal iEncoder As Integer, ByVal bVal As Long) As Long
```

Example VB Usage:

```
Dim errCode As Long
Dim iDeviceNo As Integer
Dim iEncoder As Integer
Dim bVal As Long

iDeviceNo = 0
iEncoder = 0
bVal = True

errCode = PCI3E_SetForward(iDeviceNo, iEncoder, bVal)
If errCode <> 0 then
      ‘ Handle error..
End If
```

9.4.64 PCI3E_SetInterruptControl

Description:

This function sets the current enable state of the FIFO Half-Full interrupt and the Encoder Trigger-Out interrupt.

Note: if the FIFO Half-Full interrupt is enabled, the FIFO buffer should also be enabled using the PCI3E_EnableFIFOBuffer function.

C Language Function Prototype:

```
int _stdcall PCI3E_SetInterruptControl(short iDeviceNo, BOOL
bEnableFIFOHalfFullInterrupt, BOOL bEnableTriggerOutInterrupt);
```

Returns:

Result code as an integer: See error code section for values other than zero. Zero implies function call is successful.

Parameters:

`iDeviceNo:` identifies the PCI-3E card (zero based).

`bEnableFIFOHalfFullInterrupt:` enable state of the FIFO Half-Full interrupt.
1 = enabled, 0 = disabled

`bEnableTriggerOutInterrupt:` enable state of the trigger-out signal interrupt.
1 = enabled, 0 = disabled

Example C Usage:

```
int iResult = S_OK;
short iDeviceNo = 0;
BOOL bEnableFIFOHalfFullInterrupt= FALSE;
BOOL bEnableTriggerOutInterrupt = TRUE;

iResult = PCI3E_SetInterruptControl(iDeviceNo, bEnableFIFOHalfFullInterrupt,
bEnableTriggerOutInterrupt);
if ( iResult != S_OK ){ // Handle error...}
```

VB Language Function Declaration:

```
Public Declare Function PCI3E_SetInterruptControl Lib "USD_PCI_3E.dll" (ByVal
iDeviceNo As Integer, ByVal bEnableFIFOHalfFullInterrupt As Long, ByVal
bEnableTriggerOutInterrupt As Long) As Long
```

Example VB Usage:

```
Dim errCode As Long
Dim iDeviceNo As Integer
Dim bEnableFIFOHalfFullInterruptAs Long
Dim bEnableTriggerOutInterrupt As Long

iDeviceNo = 0
bEnableFIFOHalfFullInterruptAs = False
bEnableTriggerOutInterrupt = True

errCode = PCI3E_SetInterruptControl(iDeviceNo, bEnableFIFOHalfFullInterrupt,
      bEnableTriggerOutInterrupt)
If errCode <> 0 then
```

```
      ‘ Handle error..
End If
```

9.4.65 PCI3E_SetInvertIndex

Description:

This function takes a boolean value that determines the active level of the index.
`bVal = TRUE` when the index is active low. Default is active high.

C Language Function Prototype:

```
int _stdcall PCI3E_SetInvertIndex(short iDeviceNo, short iEncoder, BOOL bVal);
```

Returns:

Result code as an integer: See error code section for values other than zero. Zero implies function call is successful.

Parameters:

`iDeviceNo`: identifies the PCI-3E card (zero based).
`iEncoder`: identifies the encoder channel (zero based).
`bVal`: in-out boolean parameter. See description above.

Example C Usage:

```
int iResult = S_OK;
short iDeviceNo = 0;
short iEncoder = 0;
BOOL bVal = FALSE;

iResult = PCI3E_SetInvertIndex(iDeviceNo, iEncoder, bVal);
if ( iResult != S_OK ){ // Handle error... }
```

VB Language Function Declaration:

```
Public Declare Function PCI3E_SetInvertIndex Lib "USD_PCI_3E.dll" (ByVal
iDeviceNo As Integer, ByVal iEncoder As Integer, ByVal bVal As Long) As Long
```

Example VB Usage:

```
Dim errCode As Long
Dim iDeviceNo As Integer
Dim iEncoder As Integer
Dim bVal as Long

iDeviceNo = 0
iEncoder = 0
bVal = False

errCode = PCI3E_SetInvertIndex(iDeviceNo, iEncoder, bVal)
If errCode <> 0 then
      ‘ Handle error..
End If
```

9.4.66 PCI3E_SetMatch

Description:

This function sets the Match Register value. It is used as a reference to signal a capture when the counter equals the match register value.

C Language Function Prototype:

```
int _stdcall PCI3E_SetMatch(short iDeviceNo, short iEncoder, unsigned long
ulVal);
```

Returns:

Result code as an integer: See error code section for values other than zero. Zero implies function call is successful.

Parameters:

`iDeviceNo`: identifies the PCI-3E card (zero based).
`iEncoder`: identifies the encoder channel (zero based).
`ulVal`: contains the value to be written to the Match Register.

Example C Usage:

```
int iResult = S_OK;
short iDeviceNo = 0;
short iEncoder = 0;
unsigned long ulVal = 500;    //  Note: choose your value here.

iResult = PCI3E_SetMatch(iDeviceNo, iEncoder, ulVal);
if ( iResult != S_OK ){ // Handle error... }
```

VB Language Function Declaration:

```
Public Declare Function PCI3E_SetMatch Lib "USD_PCI_3E.dll" (ByVal iDeviceNo As
Integer, ByVal iEncoder As Integer, ByVal ulVal As Long) As Long
```

Example VB Usage:

```
Dim errCode As Long
Dim iDeviceNo As Integer
Dim iEncoder As Integer
Dim lVal As Long

iDeviceNo = 0
iEncoder = 0
lVal = 499

errCode = PCI3E_SetMatch(iDeviceNo, iEncoder, lVal)
If errCode <> 0 then
      ‘ Handle error..
End If
```

9.4.67 PCI3E_SetMultiplier

Description:

This function set the multiplier mode that determines when the counter is to be incremented based on the number of quadrature state transitions. See possible values in parameters description.

C Language Function Prototype:

```
int _stdcall PCI3E_SetMultiplier(short iDeviceNo, short iEncoder, short iVal);
```

Returns:

Result code as an integer: See error code section for values other than zero. Zero implies function call is successful.

Parameters:

`iDeviceNo`: identifies the PCI-3E card (zero based).
`iEncoder`: identifies the encoder channel (zero based).
`iVal`: identifies when the count increments.

Possible values are: 0 = in_a is clock, in_b is direction
1 = count increments once every four quad states, X1
2 = count increments once every two quad states, X2
3 = count increments once every quad state, X4

Example C Usage:

```
int iResult = S_OK;
short iDeviceNo = 0;
short iEncoder = 0;
short iVal = 1;

iResult = PCI3E_SetMultiplier(iDeviceNo, iEncoder, iVal);
if ( iResult != S_OK ){ // Handle error...}
```

VB Language Function Declaration:

```
Public Declare Function PCI3E_SetMultiplier Lib "USD_PCI_3E.dll" (ByVal
iDeviceNo As Integer, ByVal iEncoder As Integer, ByVal iVal As Integer) As Long
```

Example VB Usage:

```
Dim errCode As Long
Dim iDeviceNo As Integer
Dim iEncoder As Integer
Dim iVal As Integer

iDeviceNo = 0
iEncoder = 0
iVal = 1

errCode = PCI3E_SetMultiplier(iDeviceNo, iEncoder, iVal)
If errCode <> 0 then
      ‘ Handle error..
End If
```

9.4.68 PCI3E_SetOutputPortConfig

Description:
This function is used to configure the output port setup.

The output port pins may be driven by the output port register or trigger out signals.

If the trigger out signal is used to drive the output port, then the length of the output trigger signal may be specified.

C Language Function Prototype:
```
int _stdcall PCI3E_SetOutputPortConfig(short iDeviceNo, BOOL
*pbTriggerOutSignalDrivesOutputPin, unsigned char ucTriggerSignalLengthCode);
```

Returns:
Result code as an integer: See error code section for values other than zero. Zero implies function call is successful.

Parameters:
`iDeviceNo`: identifies the PCI-3E card (zero based).
`pbTriggerOutSignalDrivesOutputPin`: pointer to an array of 4 booleans used to determine if the cooresponding output port pins are to be driven by the output port register or trigger out signals.

array element 0:	0 --- OUT0 is driven by bit 0 of reg.#46
	1 --- OUT0 is driven by Trigger Out signal from Encoder Channel 0
array element 1:	0 --- OUT1 is driven by bit 1 of reg.#46
	1 --- OUT1 is driven by Trigger Out signal from Encoder Channel 1
array element 2:	0 --- OUT2 is driven by bit 2 of reg.#46
	1 --- OUT2 is driven by Trigger Out signal from Encoder Channel 2
array element 3:	0 --- OUT3 is driven by bit 3 of reg.#46
	1 --- OUT3 is driven by Combined Trigger Out signal

`ucTriggerSignalLengthCode`: is used to specify the length of the signal generated on an output pin when an output pin is driven by a Trigger Out signal.

Code	Length of Trigger Signal
0	1 mS
1	200 µS
2	20 µS
3	5 µS
4	Toggle

Example C Usage:
```
int iResult = S_OK;
short iDeviceNo = 0;
BOOL bTriggerOutSignalDrivesOutputPin[4] = {1, 0, 0, 0};
```

```
unsigned char ucTriggerSignalLengthCode = 1;
iResult = PCI3E_SetOutputPortConfig(iDeviceNo,
                                    bTriggerOutSignalDrivesOutputPin,
                                    ucTriggerSignalLengthCode);
if ( iResult != S_OK ){ // Handle error...}
```

VB Language Function Declaration:

```
Public Declare Function PCI3E_SetOutputPortConfig Lib "USD_PCI_3E.dll" (ByVal
iDeviceNo As Integer, ByRef pbTriggerOutSignalDrivesOutputPin As Long, ByVal
ucTriggerSignalLengthCode As Byte) As Long
```

Example VB Usage:

```
Dim errCode As Long
Dim iDeviceNo As Integer
Dim bTriggerOutSignalDrivesOutputPin(3) As Long
Dim bytTriggerSignalLengthCode As Byte

iDeviceNo = 0
bTriggerOutSignalDrivesOutputPin(0) = 1
bytTriggerSignalLengthCode = 1

errCode = PCI3E_SetOutputPortConfig(iDeviceNo, _
                                    bTriggerOutSignalDrivesOutputPin, _
                                    bytTriggerSignalLengthCode)
If errCode <> 0 then
      ‘ Handle error..
End If
```

9.4.69 PCI3E_SetPresetOnIndex

Description:

This function sets a boolean value that indicates whether index will either reset or preset accumulator (counter). 1 = preset occurs when index is detected, 0 = reset will occur when index is detected. This function requires that the index is enabled using PCI3E_SetEnableIndex.

C Language Function Prototype:

```
int _stdcall PCI3E_SetPresetOnIndex(short iDeviceNo, short iEncoder, BOOL
bVal);
```

Returns:

Result code as an integer: See error code section for values other than zero. Zero implies function call is successful.

Parameters:

`iDeviceNo`: identifies the PCI-3E card (zero based).
`iEncoder`: identifies the encoder channel (zero based).
`bVal`: See description above.

Example C Usage:

```
int iResult = S_OK;
short iDeviceNo = 0;
short iEncoder = 0;
BOOL bVal = False;

iResult = PCI3E_SetPresetOnIndex(iDeviceNo, iEncoder, bVal);
if ( iResult != S_OK ){ // Handle error...}
```

VB Language Function Declaration:

```
Public Declare Function PCI3E_SetPresetOnIndex Lib "USD_PCI_3E.dll" (ByVal
iDeviceNo As Integer, ByVal iEncoder As Integer, ByVal bVal As Long) As Long
```

Example VB Usage:

```
Dim errCode As Long
Dim iDeviceNo As Integer
Dim iEncoder As Integer
Dim bVal As Long

iDeviceNo = 0
iEncoder = 0
bVal = True

errCode = PCI3E_SetPresetOnIndex(iDeviceNo, iEncoder, bVal)
If errCode <> 0 then
      ‘ Handle error..
End If
```

9.4.70 PCI3E_SetPresetValue

Description:
This function sets the Preset Register with a new value.

C Language Function Prototype:
```
int _stdcall PCI3E_SetPresetValue(short iDeviceNo, short iEncoder, unsigned
long ulVal);
```

Returns:
Result code as an integer: See error code section for values other than zero. Zero implies function call is successful.

Parameters:
`iDeviceNo`: identifies the PCI-3E card (zero based).
`iEncoder`: identifies the encoder channel (zero based).
`ulVal`: the new preset register value may also be referred to as upper-limit, range-limit, max count, or resolution -1.

Example C Usage:
```
int iResult = S_OK;
short iDeviceNo = 0;
short iEncoder = 0;
unsigned long ulVal = 499;          //  Note: choose your value here.

iResult = PCI3E_SetPresetValue(iDeviceNo, iEncoder, ulVal);
if ( iResult != S_OK ){ // Handle error...}
```

VB Language Function Declaration:
```
Public Declare Function PCI3E_SetPresetValue Lib "USD_PCI_3E.dll" (ByVal
iDeviceNo As Integer, ByVal iEncoder As Integer, ByVal ulVal As Long) As Long
```

Example VB Usage:
```
Dim errCode As Long
Dim iDeviceNo As Integer
Dim iEncoder As Integer
Dim lVal As Long

iDeviceNo = 0
iEncoder = 0
lVal = 499

errCode = PCI3E_SetPresetValue(iDeviceNo, iEncoder, lVal)
If errCode <> 0 then
      ‘ Handle error..
End If
```

9.4.71 PCI3E_SetSamplesToCollect

Description:
This function sets the number of samples to be collected when an acquisition is started.

C Language Function Prototype:
```
int _stdcall PCI3E_SetSamplesToCollect(short iDeviceNo, unsigned long ulVal);
```

Returns:
Result code as an integer: See error code section for values other than zero. Zero implies function call is successful.

Parameters:
`iDeviceNo`: identifies the PCI-3E card (zero based).
`ulVal:` identifies the number of samples to collect.

Example C Usage:
```
int iResult = S_OK;
short iDeviceNo = 0;
unsigned long ulVal = 100000;

iResult = PCI3E_SetSamplesToCollect(iDeviceNo, ulVal);
if ( iResult != S_OK ){ // Handle error...}
```

VB Language Function Declaration:
```
Public Declare Function PCI3E_SetSamplesToCollect Lib "USD_PCI_3E.dll" (ByVal
iDeviceNo As Integer, ByVal ulVal As Long) As Long
```

Example VB Usage:
```
Dim errCode As Long
Dim iDeviceNo As Integer
Dim lVal As Long

iDeviceNo = 0
lVal = 100000

errCode = PCI3E_SetSamplesToCollect(iDeviceNo, lVal)
If errCode <> 0 then
      ‘ Handle error..
End If
```

9.4.72 PCI3E_SetSamplingRateMultiplier

Description:

This function sets the 32 bit sampling rate multiplier (N) which is used to determine the sampling period. The sampling period is calculated by the following equations.

N: the value of the "sampling rate multiplier register"
The sampling period = (N+1) * 30 microsecond

C Language Function Prototype:

```
int _stdcall PCI3E_SetSamplingRateMultiplier(short iDeviceNo, unsigned long
ulVal);
```

Returns:

Result code as an integer: See error code section for values other than zero. Zero implies function call is successful.

Parameters:

`iDeviceNo`: identifies the PCI-3E card (zero based).
`ulVal:` contains the sampling rate multiplier used to calculate the sampling period.

Example C Usage:

```
int iResult = S_OK;
short iDeviceNo = 0;
unsigned long ulVal = 33332; // Apx. 1 sec period

iResult = PCI3E_SetSamplingRateMultiplier(iDeviceNo, ulVal);
if ( iResult != S_OK ){ // Handle error... }
```

VB Language Function Declaration:

```
Public Declare Function PCI3E_SetSamplingRateMultiplier Lib "USD_PCI_3E.dll"
(ByVal iDeviceNo As Integer, ByVal ulVal As Long) As Long
```

Example VB Usage:

```
Dim errCode As Long
Dim iDeviceNo As Integer
Dim lVal As Long

iDeviceNo = 0
lVal = 33332 ‘ Apx. 1 second period

errCode = PCI3E_SetSamplingRateMultiplier(iDeviceNo, lVal)
If errCode <> 0 then
      ‘ Handle error..
End If
```

9.4.73 PCI3E_SetTimeBasedLogSettings

Description:
This function is used to configure the settings used to determine the condition that must be satisfied in order to start a data acquisition. Once the acquisition is started, the PCI-3E will evaluate the `pucQualifier` parameter on each sampling period to determine if the data should be stored or discarded.

C Language Function Prototype:
int _stdcall PCI3E_SetTimeBasedLogSettings(short iDeviceNo,
unsigned char *pucTrigger, unsigned char ucTrigAnd,
unsigned char *pucQualifier, unsigned char ucQualAnd,
unsigned long ulNumberOfSamples);

Returns:
Result code as an integer: See error code section for values other than zero. Zero implies function call is successful.

Parameters:
`iDeviceNo`: identifies the PCI-3E card (zero based).
pucTrigger: pointer to 4 byte array of triggering codes.
ucTrigAnd: determines if the array of trigger codes are AND'ed or OR'ed.
1 = AND'ed, 0 = OR'ed
pucQualifier: pointer to 4 byte array of qualifier codes.
ucQualAnd: determines if the array of qualifier codes are AND'ed or OR'ed.
1 = AND'ed, 0 = OR'ed
ulNumberOfSamples: identifies the number of samples to be collected.

Triggering / Qualifier Codes

Trigger or qualify never (ignore)	0
Trigger or qualify on rising edge	1
Trigger or qualify on falling edge	2
Trigger or qualify on either edge	3
Trigger or qualify on high condition	4
Trigger or qualify on low condition	5
Trigger or qualify unconditionally (always)	6
Trigger or qualify unconditionally (always)	7

Example C Usage:
```
int iResult = S_OK;
short iDeviceNo = 0;

// trigger on rising edge of input bit 0.
unsigned char ucTrigger[4] = {1,0,0,0};

// trigger conditions are AND'ed
unsigned char ucTrigAnd = TRUE;
```

```
// qualifier condition set to store always
unsigned char ucQualifier[4] = {6,6,6,6};

// qualifier conditions are OR'ed
unsigned char ucQualAnd = FALSE;
unsigned long ulNumberOfSamples = 100000;

iResult = PCI3E_SetTimeBasedLogSettings(iDeviceNo, ucTrigger, ucTrigAnd,
      ucQualifier, ucQualAnd, ulNumberOfSamples);
if ( iResult != S_OK ){ // Handle error... }
```

VB Language Function Declaration:

```
Public Declare Function PCI3E_SetTimeBasedLogSettings Lib "USD_PCI_3E.dll"
(ByVal iDeviceNo As Integer, ByRef pucTrigger As Byte, ByVal ucTrigAnd As Byte,
ByRef pucQualifier As Byte, ByVal ucQualAnd As Byte, ByVal ulNumberOfSamples As
Long) As Long
```

Example VB Usage:

```
Dim errCode As Long
Dim iDeviceNo As Integer
Dim bytTrigger(3) As Byte
Dim bytTriggerAnd As Byte
Dim bytQualifier(3) As Byte
Dim bytQualifierAnd As Byte
Dim lNumberOfSamples As Long

iDeviceNo = 0
bytTrigger(0) = 1
bytTrigAnd = False
bytQualifier(0) = 6
bytQualifier(1) = 6
bytQualifier(2) = 6
bytQualifier(3) = 6
bytQualAnd = False
lNumberOfSamples = 100000

errCode = PCI3E_SetTimeBasedLogSettings(0, bytTrigger(0), bytTriggerAnd,
bytQualifier(0), bytQualifierAnd, lNumberOfSamples)
    If errCode <> 0 then
      ' Handle error..
End If
```

9.4.74 PCI3E_SetTriggerOnDecrease

Description:

This function takes a boolean value that determines whether a trigger signal is generated when the count decreases.

C Language Function Prototype:

```
int _stdcall PCI3E_SetTriggerOnDecrease(short iDeviceNo, short iEncoder, BOOL
bVal);
```

Returns:

Result code as an integer: See error code section for values other than zero. Zero implies function call is successful.

Parameters:

`iDeviceNo`: identifies the PCI-3E card (zero based).
`iEncoder`: identifies the encoder channel (zero based).
`bVal`: See description above.

Example C Usage:

```
int iResult = S_OK;
short iDeviceNo = 0;
short iEncoder = 0;
BOOL bVal = FALSE;

iResult = PCI3E_SetTriggerOnDecrease(iDeviceNo, iEncoder, bVal);
if ( iResult != S_OK ){ // Handle error... }
```

VB Language Function Declaration:

```
Public Declare Function PCI3E_SetTriggerOnDecrease Lib "USD_PCI_3E.dll" (ByVal
iDeviceNo As Integer, ByVal iEncoder As Integer, ByVal bVal As Long) As Long
```

Example VB Usage:

```
Dim errCode As Long
Dim iDeviceNo As Integer
Dim iEncoder As Integer
Dim bVal As Long

iDeviceNo = 0
iEncoder = 0
bVal = False

errCode = PCI3E_SetTriggerOnDecrease(iDeviceNo, iEncoder, bVal)
If errCode <> 0 then
      ‘ Handle error…
End If
```

9.4.75 PCI3E_SetTriggerOnIncrease

Description:

This function sets a boolean value that determines whether a trigger signal is generated when a count increases.

C Language Function Prototype:

```
int _stdcall PCI3E_SetTriggerOnIncrease(short iDeviceNo, short iEncoder, BOOL
bVal);
```

Returns:

Result code as an integer: See error code section for values other than zero. Zero implies function call is successful.

Parameters:

`iDeviceNo`: identifies the PCI-3E card (zero based).
`iEncoder`: identifies the encoder channel (zero based).
`bVal`: See description above.

Example C Usage:

```
int iResult = S_OK;
short iDeviceNo = 0;
short iEncoder = 0;
BOOL bVal = False;

iResult = PCI3E_SetTriggerOnIncrease(iDeviceNo, iEncoder, bVal);
if ( iResult != S_OK ){ // Handle error... }
```

VB Language Function Declaration:

```
Public Declare Function PCI3E_SetTriggerOnIncrease Lib "USD_PCI_3E.dll" (ByVal
iDeviceNo As Integer, ByVal iEncoder As Integer, ByVal bVal As Long) As Long
```

Example VB Usage:

```
Dim errCode As Long
Dim iDeviceNo As Integer
Dim iEncoder As Integer
Dim bVal As Long

iDeviceNo = 0
iEncoder = 0
bVal = False

errCode = PCI3E_SetTriggerOnIncrease(iDeviceNo, iEncoder, bVal)
If errCode <> 0 then
      ‘ Handle error…
End If
```

9.4.76 PCI3E_SetTriggerOnIndex

Description:

This function takes a boolean value that determines whether a trigger signal is generated when the edge of an index is detected.

C Language Function Prototype:

```
int _stdcall PCI3E_SetTriggerOnIndex(short iDeviceNo, short iEncoder, BOOL
bVal);
```

Returns:

Result code as an integer: See error code section for values other than zero. Zero implies function call is successful.

Parameters:

`iDeviceNo`: identifies the PCI-3E card (zero based).
`iEncoder`: identifies the encoder channel (zero based).
`bVal`: See description above.

Example C Usage:

```
int iResult = S_OK;
short iDeviceNo = 0;
short iEncoder = 0;
BOOL bVal = False;

iResult = PCI3E_SetTriggerOnIndex(iDeviceNo, iEncoder, bVal);
if ( iResult != S_OK ){ // Handle error... }
```

VB Language Function Declaration:

```
Public Declare Function PCI3E_SetTriggerOnIndex Lib "USD_PCI_3E.dll" (ByVal
iDeviceNo As Integer, ByVal iEncoder As Integer, ByVal bVal As Long) As Long
```

Example VB Usage:

```
Dim errCode As Long
Dim iDeviceNo As Integer
Dim iEncoder As Integer
Dim bVal As Long

iDeviceNo = 0
iEncoder = 0
bVal = True

errCode = PCI3E_SetTriggerOnIndex(iDeviceNo, iEncoder, bVal)
If errCode <> 0 then
      ‘ Handle error…
End If
```

9.4.77 PCI3E_SetTriggerOnMatch

Description:

This function takes a boolean value that determines whether a trigger signal is generated when count = match.

C Language Function Prototype:

```
int _stdcall PCI3E_SetTriggerOnMatch(short iDeviceNo, short iEncoder, BOOL
bVal);
```

Returns:

Result code as an integer: See error code section for values other than zero. Zero implies function call is successful.

Parameters:

`iDeviceNo`: identifies the PCI-3E card (zero based).
`iEncoder`: identifies the encoder channel (zero based).
`bVal`: See description above.

Example C Usage:

```
int iResult = S_OK;
short iDeviceNo = 0;
short iEncoder = 0;
BOOL bVal = FALSE;

iResult = PCI3E_SetTriggerOnMatch(iDeviceNo, iEncoder, bVal);
if ( iResult != S_OK ){ // Handle error... }
```

VB Language Function Declaration:

```
Public Declare Function PCI3E_SetTriggerOnMatch Lib "USD_PCI_3E.dll" (ByVal
iDeviceNo As Integer, ByVal iEncoder As Integer, ByVal bVal As Long) As Long
```

Example VB Usage:

```
Dim errCode As Long
Dim iDeviceNo As Integer
Dim iEncoder As Integer
Dim bVal As Long

iDeviceNo = 0
iEncoder = 0
bVal = False
errCode = PCI3E_SetTriggerOnMatch(iDeviceNo, iEncoder, bVal)
If errCode <> 0 then
      ‘ Handle error…
End If
```

9.4.78 PCI3E_SetTriggerOnRollover

Description:

This function takes a boolean value that determines whether a trigger signal is generated when rolling over N-1 to 0 in modulo-N mode.

C Language Function Prototype:

```
int _stdcall PCI3E_SetTriggerOnRollover(short iDeviceNo, short iEncoder, BOOL
bVal);
```

Returns:

Result code as an integer: See error code section for values other than zero. Zero implies function call is successful.

Parameters:

`iDeviceNo`: identifies the PCI-3E card (zero based).
`iEncoder`: identifies the encoder channel (zero based).
`bVal`: See description above.

Example C Usage:

```
int iResult = S_OK;
short iDeviceNo = 0;
short iEncoder = 0;
BOOL bVal = False;

iResult = PCI3E_SetTriggerOnRollover(iDeviceNo, iEncoder, bVal);
if ( iResult != S_OK ){ // Handle error... }
```

VB Language Function Declaration:

```
Public Declare Function PCI3E_SetTriggerOnRollover Lib "USD_PCI_3E.dll" (ByVal
iDeviceNo As Integer, ByVal iEncoder As Integer, ByVal bVal As Long) As Long
```

Example VB Usage:

```
Dim errCode As Long
Dim iDeviceNo As Integer
Dim iEncoder As Integer
Dim bVal As Long

iDeviceNo = 0
iEncoder = 0
bVal = False

errCode = PCI3E_SetTriggerOnRollover(iDeviceNo, iEncoder, bVal)
If errCode <> 0 then
      ‘ Handle error…
End If
```

9.4.79 PCI3E_SetTriggerOnRollunder

Description:

This function takes a boolean value that determines whether a trigger signal is generated when rolling under 0 to N-1 in modulo-N mode.

C Language Function Prototype:

```
int _stdcall PCI3E_SetTriggerOnRollunder(short iDeviceNo, short iEncoder, BOOL
bVal);
```

Returns:

Result code as an integer: See error code section for values other than zero. Zero implies function call is successful.

Parameters:

`iDeviceNo`: identifies the PCI-3E card (zero based).
`iEncoder`: identifies the encoder channel (zero based).
`bVal`: See description above.

Example C Usage:

```
int iResult = S_OK;
short iDeviceNo = 0;
short iEncoder = 0;
BOOL bVal = FALSE;

iResult = PCI3E_SetTriggerOnRollunder(iDeviceNo, iEncoder, bVal);
if ( iResult != S_OK ){ // Handle error... }
```

VB Language Function Declaration:

```
Public Declare Function PCI3E_SetTriggerOnRollunder Lib "USD_PCI_3E.dll" (ByVal
iDeviceNo As Integer, ByVal iEncoder As Integer, ByVal bVal As Long) As Long
```

Example VB Usage:

```
Dim errCode As Long
Dim iDeviceNo As Integer
Dim iEncoder As Integer
Dim bVal As Long

iDeviceNo = 0
iEncoder = 0
bVal = False

errCode = PCI3E_GetTriggerOnRollunder(iDeviceNo, iEncoder, bVal)
If errCode <> 0 then
      ‘ Handle error…
End If
```

9.4.80 PCI3E_SetTriggerOnZero

Description:

This function takes a boolean value that determines whether a trigger signal is generated when count = 0.

C Language Function Prototype:

```
int _stdcall PCI3E_SetTriggerOnZero(short iDeviceNo, short iEncoder, BOOL
bVal);
```

Returns:

Result code as an integer: See error code section for values other than zero. Zero implies function call is successful.

Parameters:

`iDeviceNo`: identifies the PCI-3E card (zero based).
`iEncoder`: identifies the encoder channel (zero based).
`bVal`: See description above.

Example C Usage:

```
int iResult = S_OK;
short iDeviceNo = 0;
short iEncoder = 0;
BOOL bVal = FALSE;      // TODO: Set this parameter to TRUE or FALSE;

iResult = PCI3E_SetTriggerOnZero(iDeviceNo, iEncoder, bVal);
if ( iResult != S_OK ){ // Handle error... }
```

VB Language Function Declaration:

```
Public Declare Function PCI3E_SetTriggerOnZero Lib "USD_PCI_3E.dll" (ByVal
iDeviceNo As Integer, ByVal iEncoder As Integer, ByVal bVal As Long) As Long
```

Example VB Usage:

```
Dim errCode As Long
Dim iDeviceNo As Integer
Dim iEncoder As Integer
Dim bVal As Long

iDeviceNo = 0
iEncoder = 0
bVal = False

errCode = PCI3E_SetTriggerOnZero(iDeviceNo, iEncoder, bVal)
If errCode <> 0 then
      ‘ Handle error…
End If
```

9.4.81 PCI3E_Shutdown

Description:
This function must be call in order to disconnect from PCI3E driver.

C Language Function Prototype:
```
void _stdcall PCI3E_Shutdown();
```

Returns:
None

Parameters:
None

Example C Usage:
```
PCI3E_Shutdown();
```

VB Language Function Declaration:
```
Public Declare Sub PCI3E_Shutdown Lib "USD_PCI_3E.dll" ()
```

Example VB Usage:
```
PCI3E_Shutdown
```

9.4.82 PCI3E_StartAcquisition

Description:

This function starts a data acquisition. The data acquisition will stop once the specified number of data has been reached. `PCI3E_StopAcquisition` can be used to abort the acquisition in progress.

C Language Function Prototype:

```
int _stdcall PCI3E_StartAcquisition(short iDeviceNo);
```

Returns:

None

Parameters:

`iDeviceNo`: identifies the PCI-3E card (zero based).

Example C Usage:

```
int iResult = S_OK;
short iDeviceNo = 0;

iResult = PCI3E_StartAcquisition(iDeviceNo);
if ( iResult != S_OK ){ // Handle error... }
```

VB Language Function Declaration:

```
Public Declare Function PCI3E_StartAcquisition Lib "USD_PCI_3E.dll" (ByVal
iDeviceNo As Integer) As Long
```

Example VB Usage:

```
Dim errCode As Long
Dim iDeviceNo As Integer

iDeviceNo = 0

errCode = PCI3E_StartAcquisition(iDeviceNo)
If errCode <> 0 then
      ‘ Handle error…
End If
```

9.4.83 PCI3E_StopAcquisition

Description:
This function aborts the data acquisition in progress.

C Language Function Prototype:
```
int _stdcall PCI3E_StopAcquisition(short iDeviceNo);
```

Returns:
None

Parameters:
`iDeviceNo`: identifies the PCI-3E card (zero based).

Example C Usage:
```
int iResult = S_OK;
short iDeviceNo = 0;

iResult = PCI3E_StopAcquisition(iDeviceNo);
if ( iResult != S_OK ){ // Handle error... }
```

VB Language Function Declaration:
```
Public Declare Function PCI3E_StopAcquisition Lib "USD_PCI_3E.dll" (ByVal
iDeviceNo As Integer) As Long
```

Example VB Usage:
```
Dim errCode As Long
Dim iDeviceNo As Integer

iDeviceNo = 0

errCode = PCI3E_StopAcquisition(iDeviceNo)
If errCode <> 0 then
      ‘ Handle error…
End If
```

9.4.84 PCI3E_UnRegisterInterruptHandler

Description:

This function removes a registered callback function reference that is called when an interrupt is detected.

C Language Function Prototype:

```
int _stdcall PCI3E_UnRegisterInterruptHandler(short iDeviceNo);
```

Returns:

Result code as an integer: See error code section for values other than zero. Zero implies function call is successful.

Parameters:

`iDeviceNo`: identifies the PCI-3E card (zero based).

Example C Usage:

```
int iResult = S_OK;
short iDeviceNo = 0;

iResult = PCI3E_UnRegisterInterruptHandler(iDeviceNo);
if ( iResult != S_OK ){ // Handle error...}
```

VB Language Function Declaration:

```
Public Declare Function PCI3E_UnRegisterInterruptHandler Lib "USD_PCI_3E.dll"
(ByVal iDeviceNo As Integer) As Long
```

Example VB Usage:

```
Dim errCode As Long
Dim iDeviceNo As Integer

iDeviceNo = 0

errCode = PCI3E_UnRegisterInterruptHandler(iDeviceNo)
If errCode <> 0 then
      ‘ Handle error..
End If
```

9.4.85 PCI3E_WriteOutputPortRegister

Description:
This function sets the value stored in the Output Port Register.

C Language Function Prototype:
```
int _stdcall PCI3E_WriteOutputPortRegister(short iDeviceNo, unsigned char
ucVal);
```

Returns:
Result code as an integer: See error code section for values other than zero. Zero implies function call is successful.

Parameters:
`iDeviceNo`: identifies the PCI-3E card (zero based).
`ucVal`: value value to be written to the output port register.

Bit 7-4: always 0
Bit 3: Output Port – OUT3
Bit 2: Output Port – OUT2
Bit 1: Output Port – OUT1
Bit 0: Output Port – OUT0

Example C Usage:
```
int iResult = S_OK;
short iDeviceNo = 0;
unsigned char ucVal = 0x3; // Turn on ouput bits 0 and 1.

iResult = PCI3E_WriteOutputPortRegister(iDeviceNo, ucVal);
if( iResult != S_OK ){  // Handle error...}
```

VB Language Function Declaration:
```
Public Declare Function PCI3E_WriteOutputPortRegister Lib "USD_PCI_3E.dll"
(ByVal iDeviceNo As Integer, ByVal ucVal As Byte) As Long
```

Example VB Usage:
```
Dim errCode As Long
Dim iDeviceNo As Integer
Dim bytVal As Byte

iDeviceNo = 0
bytVal = &H3 ‘ Turn on output bits 0 and 1.

errCode =  PCI3E_WriteOutputPortRegister(iDeviceNo, bytVal)
If errCode <> 0 Then
      ‘ Handle error...
End If
```

9.4.86 PCI3E_WriteRegister

Description:
This function allows a value to be written into a specified PCI-3E register.

C Language Function Prototype:
```
int _stdcall PCI3E_WriteRegister(short iDeviceNo, short iRegister, unsigned
long ulVal);
```

Returns:
Result code as an integer: See error code section for values other than zero. Zero implies function call is successful.

Parameters:
`iDeviceNo`: identifies the PCI-3E card (zero based).
`iRegister`: identifies the specific register to read. Valid registers are 0 - 47.
`ulVal` : the value to be written to the specified register.

Example C Usage:
```
int iResult = S_OK;
short iDeviceNo = 0;
short iRegister = 0;
unsigned long ulVal = 0;

iResult = PCI3E_WriteRegister(iDeviceNo,  iRegister, ulVal);
if( iResult != S_OK ){  // Handle error...}
```

VB Language Function Declaration:
```
Public Declare Function PCI3E_WriteRegister Lib "USD_PCI_3E.dll" (ByVal
iDeviceNo As Integer, ByVal iRegister As Integer, ByVal ulVal As Long) As Long
```

Example VB Usage:
```
Dim errCode As Long
Dim iDeviceNo As Integer
Dim iRegister As Integer
Dim lVal As Long

iDeviceNo = 0
iRegister = 0
lVal = 0

errCode = PCI3E_WriteRegister(iDevice, iRegister, lVal)
If errCode <> 0 Then
      ‘ Handle error...
End If
```

10 Interface Circuits

10.1 Encoder Channel Input

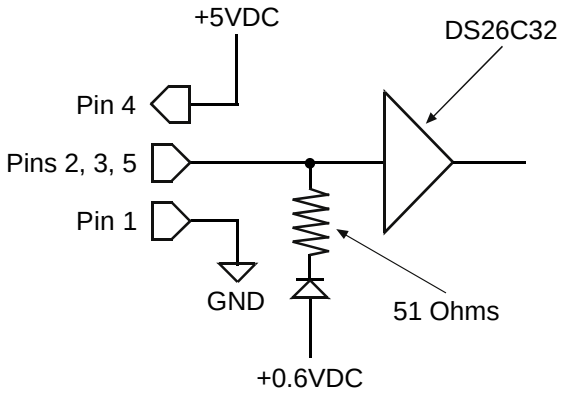


Figure 10.1.1 Single-ended input

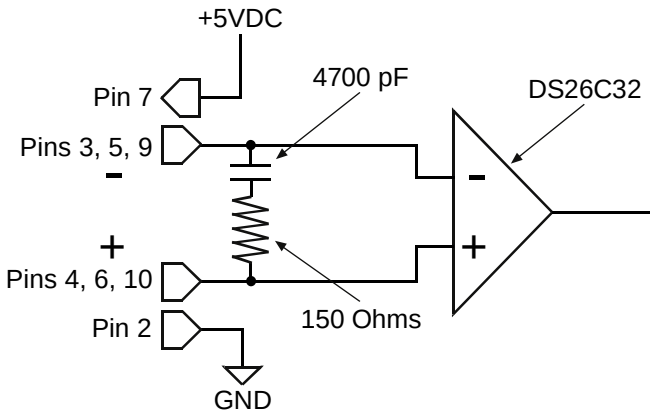
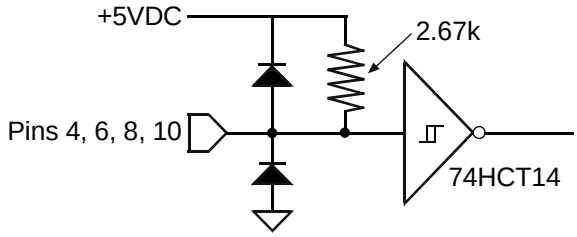


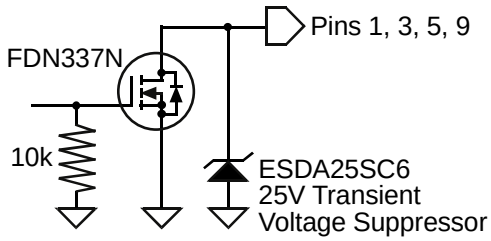
Figure 10.1.2 Differential input

10.2 Input/Output Port



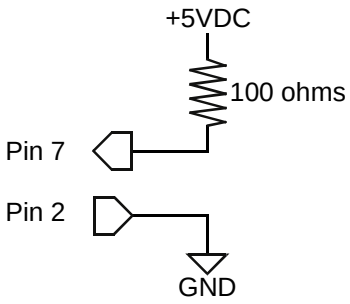
Note: (1) This diagram applies to the following pins of the "I/O PORT" connector:
IN0 – pin 4
IN1 – pin 6
IN2 – pin 8
IN3 – pin 10
(2) Input pins have built-in digital noise filters. Pulses with less than 2 microsecond width are considered noises and rejected.

Figure 10.2.1 Input port diagram



Note: This diagram applies to the following pins of the "I/O PORT" connector:
OUT0 – pin 1
OUT1 – pin 3
OUT2 – pin 5
OUT3 – pin 9

Figure 10.2.2 Output port diagram



Note: A current limiting resistor protects an encoder accidentally connected to the I/O port.

Figure 10.2.3 Power and Ground pins of the “I/O PORT” connector

11 Error Codes

- NO_CARDS_DETECTED -1
- INVALID_DEVICE_NUMBER -2
- DEVICE_NOT_OPEN -3
- INVALID_ENCODER_NUMBER -4
- INVALID_REGISTER_NUMBER -5
- INVALID_SLOT_NUMBER -6
- INVALID_QUADRATURE_MODE -7
- FEATURE_NOT_SUPPORTED -8
- INVALID_COUNTER_MODE -9
- FAILED_TO_OPEN_DRIVER -10
- CONTROL_MODE_IS_ZERO -11
- INCORRECT_WINDRIVER_VERSION -12
- OVERRUN_ERROR_DETECTED -13
- FRAMING_ERROR_DETECTED -14
- RX_BUFFER_FULL -15
- INVALID_PARAMETER -16
- FATAL_ERROR -17
- FAILED_TO_ENABLE_INTERRUPT -18
- INVALID_BUFFER_SIZE -19
- INSUFFICIENT_MEMORY -20
- FIFO_BUFFER_FULL -21