DIO3217

Multi-function
Digital I/O Card

Software Manual (V1.1)

健昇科技股份有限公司

JS AUTOMATION CORP.

新北市汐止區中興路 100 號 6 樓

6F., No.100, Zhongxing Rd.,

Xizhi Dist., New Taipei City, Taiwan

TEL：+886-2-2647-6936

FAX：+886-2-2647-6940

http://www.automation.com.tw

http://www.automation-js.com/

E-mail：control.cards@automation.com.tw

Correction record

Version	Record
1.0	wdm3217.sys V1.0 drv3217.dll V1.0 DIO3217.dll V1.0
1.1	disable the software key function with return value always true

Contents

1. How to install the software of DIO3217

1.1 Install the PCI driver

The PCI card is a plug and play card, once you add a new card on the window system will detect while it is booting. Please follow the following steps to install your new card.

In Win2K/XP/7 and up system you should: (take Win XP as example)

1. Make sure the power is off
2. Plug in the interface card
3. Power on
4. A hardware install wizard will appear and tell you it finds a new PCI card
5. Do not response to the wizard, just Install the file (..\DIO3217\Software\Win2K_up\ or if you download from website please execute the file DIO3217_Install.exe to get the file)
6. After installation, power off
7. Power on, it's ready to use

For more detail of step by step installation guide, please refer the file "installation.pdf " on the CD come with the product or register as a member of our user's club at:

http://automation.com.tw/

to download the complementary documents.

2. Where to find the file you need

Win2K/XP/7 and up

The directory will be located at

.. \ JS Automation \DIO3217\API (header files and lib files for VB,VC,BCB,C#)

.. \ JS Automation \DIO3217\Driver (backup copy of DIO3217 drivers)

.. \ JS Automation \DIO3217\exe (demo program and source code)

The system driver is located at **..\system32\Drivers** and the DLL is located at **..\system**.

For your easy startup, the demo program with source code demonstrates the card functions and help file.

3. About the DIO3217 software

DIO3217 software includes a set of dynamic link library (DLL) and system driver that you can utilize to control the I/O card's ports and points separately.

Your DIO3217 software package includes setup driver, tutorial example and test program that help you how to setup and run appropriately, as well as an executable file which you can use to test each of the DIO3217 functions within Windows' operation system environment.

3.1 What you need to get started

To set up and use your DIO3217 software, you need the following:

- DIO3217 software
- DIO3217 hardware

  Main board

  Wiring board (Option)

3.2 Software programming choices

You have several options to choose from when you are programming DIO3217 software. You can use Borland C/C++, Microsoft Visual C/C++, Microsoft Visual Basic, or any other Windows-based compiler that can call into Windows dynamic link libraries (DLLs) for use with the DIO3217 software.

4. DIO3217 Language support

The DIO3217 software library is a DLL used with Win2K/XP/7 and up. You can use these DLL with any Windows integrating development environment that can call Windows DLLs.

4.1 Building applications with the DIO3217 software library

The DIO3217 function reference topic contains general information about building DIO3217 applications, describes the nature of the DIO3217 files used in building DIO3217 applications, and explains the basics of making applications using the following tools:

Applications tools

- Microsoft Visual C/C++
- Borland C/C++
- Microsoft Visual C#
- Microsoft Visual Basic
- Microsoft VB.net

If you are not using one of the tools listed, consult your development tool reference manual for details on creating applications that call DLLs.

4.2 DIO3217 Windows libraries

The DIO3217 for Windows function library is a DLL called **DIO3217.dll**. Since a DLL is used, DIO3217 functions are not linked into the executable files of applications. Only the information about the DIO3217 functions in the DIO3217 import libraries is stored in the executable files.

Import libraries contain information about their DLL-exported functions. They indicate the presence and location of the DLL routines. Depending on the development tools you are using, you can make your compiler and linker aware of the DLL functions through import libraries or through function declarations.

Refer to **Table 1** to determine to which files you need to link and which to include in your development to use the DIO3217 functions in DIO3217.dll.

Header Files and Import Libraries for Different Development Environments		
Language	**Header File**	**Import Library**
Microsoft Visual C/C++	DIO3217.h	DIO3217VC.lib
Borland C/C++	DIO3217.h	DIO3217BC.lib
Microsoft Visual C#	DIO3217.cs	
Microsoft Visual Basic	DIO3217.bas	
Microsoft VB.net	DIO3217.vb	

Table 1

5. Basic concepts of digital I/O control

The digital I/O control is the most common type of PC based application. For example, on the main board, printer port is the TTL level digital I/O.

5.1 Types of I/O calssified by isolation

If the system and I/O are not electrically connected, we call it is isolated. There are many kinds of isolation: by transformer, by photo-coupler, by magnetic coupler,… Any kind of device, they can brake the electrical connection without braking the signal is suitable for the purpose.

Currently, photo-coupler isolation is the most popular selection, isolation voltage up to 2000V or over is common. But the photo-coupler is limited by the response time, the high frequency type cost a lot. The new selection is magnetic coupler, it is design to focus on high speed application.

The merit of isolation is to avoid the noise from outside world to enter the PC system, if the noise comes into PC system without elimination, the system maybe get "crazy" by the noise disturbance. Of course the isolation also limits the versatile of programming as input or output at the same pin as the TTL does. The inter-connection of add-on card and wiring board maybe extend to several meters without any problem.

The non-isolated type is generally the TTL level input/output. The ground and power source of the input/output port come from the system. Generally you can program as input or output at the same pin as you wish. **The connection of wiring board and the add-on board is limited to 50cm or shorter** (depends on the environmental noise condition).

5.2 Types of Output calssified by driver device

There are several devices used as output driver, the relay, transistor or MOS FET, SCR and SSR. Relay is electric- mechanical device, it life time is about 1,000,000 times of switching. But on the other hand it has many selections such as high voltage or high current. It can also be used to switch DC load or AC load.

Transistor and MOS FET are basically semi-permanent devices. If you have selected the right ratings, it can work without switching life limit. But the transistor or MOS FET can only work in DC load condition.

The transistor or MOS FET also have another option is source or sink. For PMOS or PNP transistor is source type device, the load is one terminal connects to output and another connects to common ground, but NPN or NMOS is one terminal connects to output and the other connects to VCC+. **If you are concerned about hazard from high DC voltage while the load is floating, please choose the source type driver device.**

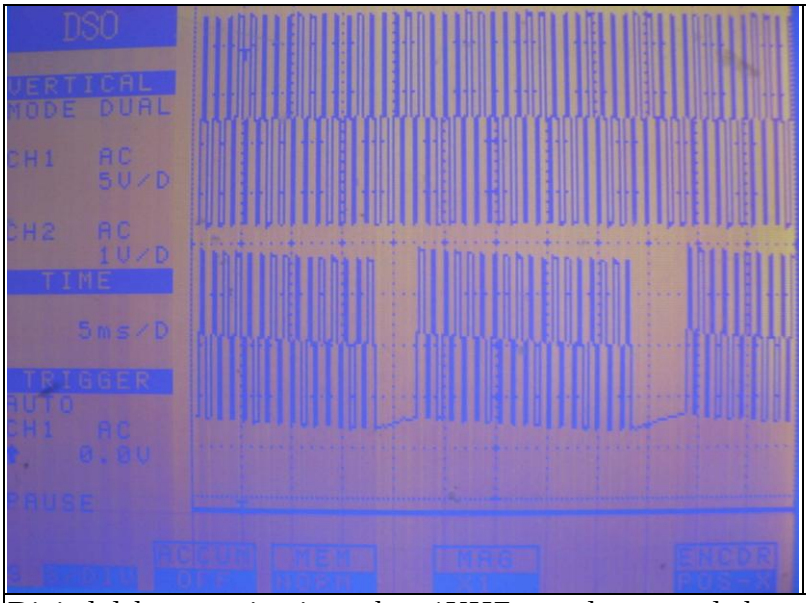
SCR (or triac) is seldom direct connect to digital output, but his relative SSR is the most often selection. In fact, SSR is a compact package of trigger circuit and triac. You can choose zero cross trigger (output command only turn on the output at power phase near zero to eliminate surge) or direct turn on type. SSR is working in AC load condition.

5.3 Input debounce

Debounce is the function to filter the input jitters. From the microscope view of a switch input, you will see the contact does not come to close or release to open clearly. In most cases, it will contact-release-contact-release… for many times then go to steady state (ON or OFF). If you do not have the debounce function, you will read the input at high state and then next read will get low state, this maybe an error data for your decision of contact input.

Debounce can be implemented by hardware or software. Analog hardware debounce circuit will have fixed time constant to filter out the significant input signal, if you want to change the response time, the only way is to change the circuit device.

If digital debounce is implemented, maybe several filter frequency you can choose. To choose the filter frequency, please keep the Nyquist–Shannon sampling theorem in mind: filter sample frequency must at least twice of the input frequency. The following sample is a bad selection of debounce filter, the input frequency is not as low as les than half of the sample frequency, the output will generate a beat frequency.



<- Input frequency at 835Hz

<- Output of digital filter,
Please note the beat frequency.

Digital debounce circuit work at 1KHZ sample rate and observe the output of filter from 835Hz input

Software debounce will consumes the CPU time a lot, we do not recommend to use except for you really know you want.

5.4 Input interrupt

You can scan the input by polling, but the CPU will spend a lot of time to do null task. Another way is use a timer to sample the input at adequate time (remind the Nyquist–Shannon sampling theorem, at least double of the input frequency). The third one is directly allows the input to generate interrupt to CPU. To use direct interrupt from input, the noise coupled from input must take special care not to mal-trigger the interrupt.

5.5 Read back of Output status

Some applications need to read back the output status, if the card do not provide output status read back, you can use a variable to store the status of output before you really command it output. Some cards provide the read back function but please note that **the read back status is come from the output register, not from the real physical output.**

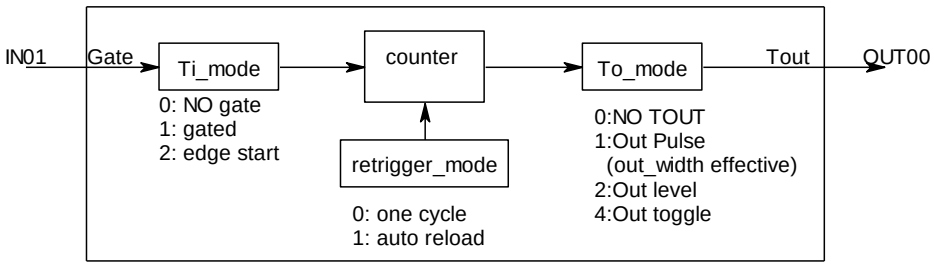
6. Basic concepts multifunction timer/counter

Many control applications need timer as time base for digital sampled data control systems. The timer consists a counter to count the time base clock on the fly and generate interrupt at a periodic time interval. If the counter do not count the time base but count the signals from external world, we call it "counter".

A timer/counter may be multi-functions, if the input signal and control mode and the output can be programmed as various kind of working mode.

6.1 Timer function

The timer model used in DIO3217 is as follows:



In this model, the timer can work in one cycle mode and auto reload mode.

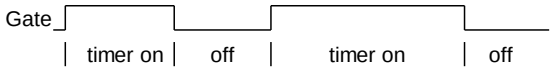
one cycle mode: the timer will stop when the timer time up.

auto reload mode (sometimes called continuous mode): the time will reload the time constant while time up.

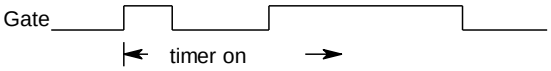
In the timer input control block:

NO gate: the timer do not control by any input.

gated : the timer only working on the gate input active and stops counting while gate is inactive.



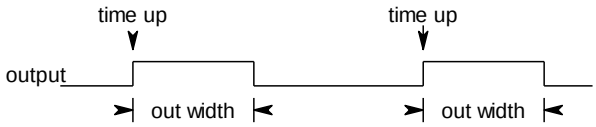
Edge start: the timer will start timing while the gate input transition from inactive to active.



In the timer output block:

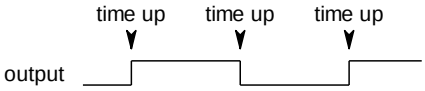
NO TOUT: the timer has no output to control (but timer time up interrupt is available).

Out_pulse: while the timer time up, it will trigger a output pulse and pulse width is controlled by out_width register at 1us time base.



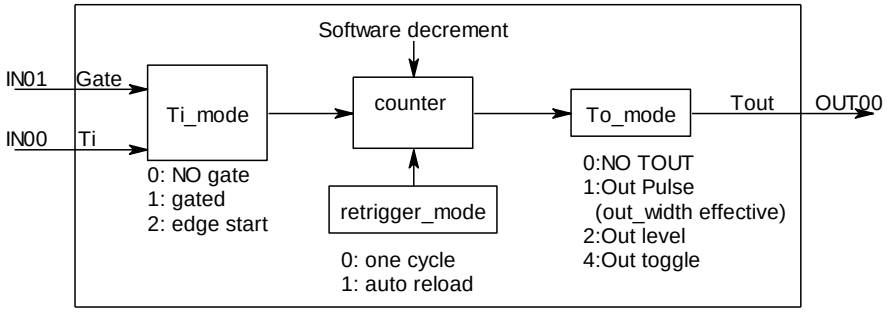
Out_level: while the timer time up, it will trigger the output active.

Out_toggle: while the timer time up, it will trigger the output toggled.



6.2 Counter function

The counter model used in DIO3217 is as follows:



In this model, the counter can work in one cycle mode and auto reload mode.

one cycle mode: the counter will stop when the counter cross zero.

auto reload mode (sometimes called continuous mode): the counter will reload the counter constant while time up.

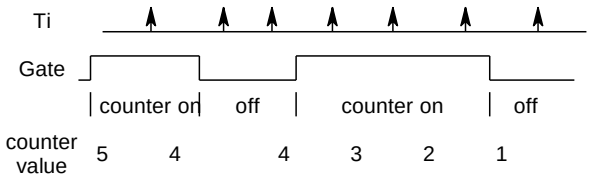
Software decrement: the counter value will decrement by software trigger.

In the counter input control block:

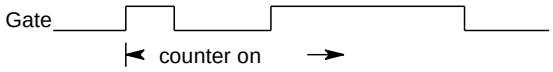
NO gate: the counter do not control by any input.

gated : while gate input is active and the counter signal input also active the counter will decrement by 1 and stops counting while gate is inactive.

Take the following diagram as example, the counter is initialized at 5 and working in gated mode, while the Ti(counter signal input) is active and gate is also active, the counter will decrease by one.



Edge start: the counter will start counting function while the gate input transition from inactive to active.



In the counter output block: (refer the timer function)

NO TOUT: the counter has no output to control (but counter cross zero interrupt is available).

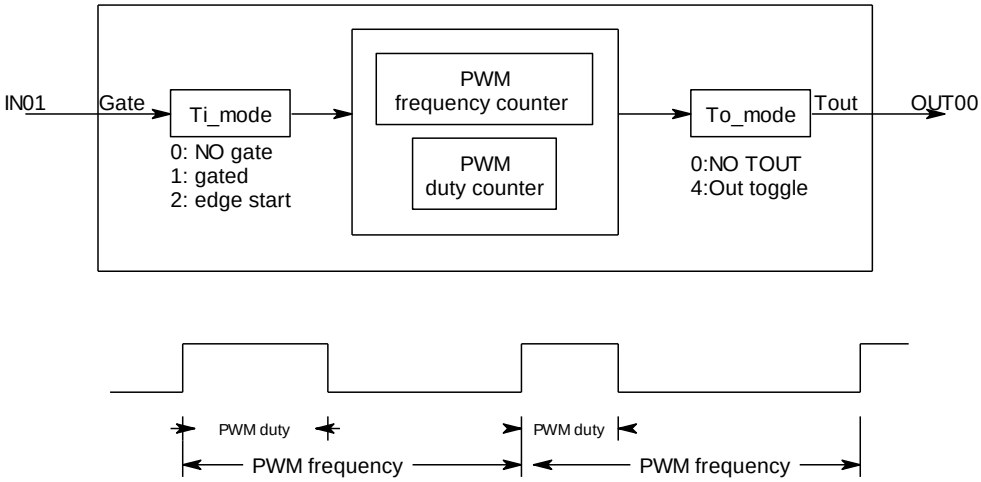
Out_pulse: while the counter cross zero, it will trigger a output pulse and pulse width is controlled by out_width register at 1us time base.

Out_level: while the counter cross zero, it will trigger the output active.

Out_toggle: while the counter cross zero, it will trigger the output toggled.

6.3 PWM function

The PWM model used in DIO3217 is as follows:



In this model, the PWM counter can only work in auto reload mode.

auto reload mode (sometimes called continuous mode): the time will reload the time constant while time up.

In the PWM counter input control block: (refer the counter function)

NO gate: the PWM counter do not control by any input.

gated : while gate input is active the PWM counter will start working and stops while gate is inactive.

Edge start: the PWM counter will start counting function while the gate input transition from inactive to active.

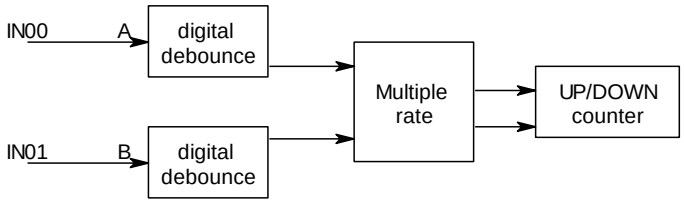
In the PWM counter output block: (refer the timer function)

NO TOUT: the PWM counter has no output to control.

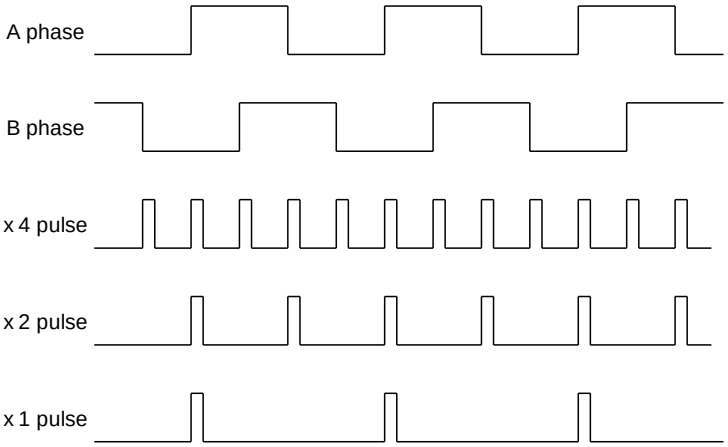
Out_toggle: while the PWM counter cross zero, it will trigger the output toggled.

7. Basic concepts quadrature encoder counter

In spite of the flexible multi-function timer/counter, the quadrature encoder counter is another type of application. The DIO3217 also has the build in function for quadrature encoder input counting.



On the above diagram, you can see the digital debounce function filter out the unwanted high frequency then pass the signal to the multiple rate circuit to determine the pulse and direction, finally the counter counts the pulses.



The left diagram shown that A phase leads B, if we take A leads B as up count and the counting pulse of up count will depends on the multiple rate.

DIO3217 card provides wide range of filter frequency from 1K up to 8M (to drop out noise pulse from 1ms to 0.125us), even the quadrature signals comes from mechanical contacts the counter can still operate very nice.

8. Function format and language difference

8.1 Function format

Every DIO3217 function is consist of the following format:

Status = function_name (parameter 1, parameter 2, … parameter n);

Each function returns a value in the **Status** global variable that indicates the success or failure of the function. A returned **Status** equal to zero that indicates the function executed successfully. A non-zero status indicates failure that the function did not execute successfully because of an error, or executed with an error. (refer chapter12 DIO3217 Error codes summary)

Note : **Status** is a 32-bit unsigned integer.

The first parameter to almost every DIO3217 function is the parameter **CardID** which is located the driver of DIO3217 board you want to use those given operation. The **CardID** is assigned by DIP/ROTARY SW. You can utilize multiple devices with different card CardID within one application; to do so, simply pass the appropriate **CardID** to each function.

Note: **CardID** is set by DIP/ROTARY SW (**0x0**-**0xF**)

These topics contain detailed descriptions of each DIO3217 function. The functions are arranged alphabetically by function name. Refer to DIO3217 Function Reference for additional information.

8.2 Variable data types

Every function description has a parameter table that lists the data types for each parameter. The following sections describe the notation used in those parameter tables and throughout the manual for variable data types.

Primary Type Names					
Name	**Description**	**Range**	**C/C++**	**Visual BASIC**	**Pascal (Borland Delphi)**
u8	8-bit ASCII character	0 to 255	char	Not supported by BASIC. For functions that require character arrays, use string types instead.	Byte
I16	16-bit signed integer	-32,768 to 32,767	short	Integer (for example: deviceNum%)	SmallInt
U16	16-bit unsigned integer	0 to 65,535	unsigned short for 32-bit compilers	Not supported by BASIC. For functions that require unsigned integers, use the signed integer type instead. See the i16 description.	Word
I32	32-bit signed integer	-2,147,483,648 to 2,147,483,647	long	Long (for example: count&)	LongInt
U32	32-bit unsigned integer	0 to 4,294,967,295	unsigned long	Not supported by BASIC. For functions that require unsigned long integers, use the signed long integer type instead. See the i32 description.	Cardinal (in 32-bit operating systems). Refer to the i32 description.
F32	32-bit single-precision floating-point value	-3.402823E+38 to 3.402823E+38	float	Single (for example: num!)	Single
F64	64-bit double-precision floating-point value	-1.797683134862315E+308 to 1.797683134862315E+308	double	Double (for example: voltage Number)	Double

Table 2

8.3 Programming language considerations

Apart from the data type differences, there are a few language-dependent considerations you need to be aware of when you use the DIO3217 API. Read the following sections that apply to your programming language.

Note: Be sure to include the declaration functions of DIO3217 prototypes by including the appropriate DIO3217 header file in your source code. Refer to Building Applications with the DIO3217 Software Library for the header file appropriate to your compiler.

8.3.1 C/C++

For C or C++ programmers, parameters listed as Input/Output parameters or Output parameters are pass-by-reference parameters, which means a pointer points to the destination variable should be passed into the function. For example, the Read Port function has the following format:

Status = DIO3217_port_read(u8 CardID, u8 port, u8*data);

where **CardID** and **port** are input parameters, and **data** is an output parameter. Consider the following example:

```
u8 CardID, port;
u8 data,
u32 Status;
Status = DIO3217_port_read (CardID, port, &data);
```

8.3.2 Visual basic

The file DIO3217.bas contains definitions for constants required for obtaining DIO Card information and declared functions and variable as global variables. You should use these constants symbols in the DIO3217.bas, do not use the numerical values.

In Visual Basic, you can add the entire DIO3217.bas file into your project. Then you can use any of the constants defined in this file and call these constants in any module of your program. To add the DIO3217.bas file for your project in Visual Basic 4.0, go to the **File** menu and select the **Add File... option**. Select DIO3217.bas, which is browsed in the DIO3217 \ API directory. Then, select **Open** to add the file to the project.

To add the DIO3217.bas file to your project in Visual Basic 5.0 and 6.0, go to the **Project** menu and select **Add Module**. Click on the Existing tab page. **Select** DIO3217.bas, which is in the DIO3217 \ API directory. Then, select **Open** to add the file to the project.

8.3.3 Borland C++ builder

To use Borland C++ builder as development tool, you should generate a .lib file from the .dll file by implib.exe.

implib DIO3217BC.lib DIO3217.dll

Then add the **DIO3217BC.lib** to your project and add

#include "DIO3217.h" to main program.

Now you may use the dll functions in your program. For example, the Read Port function has the following format:

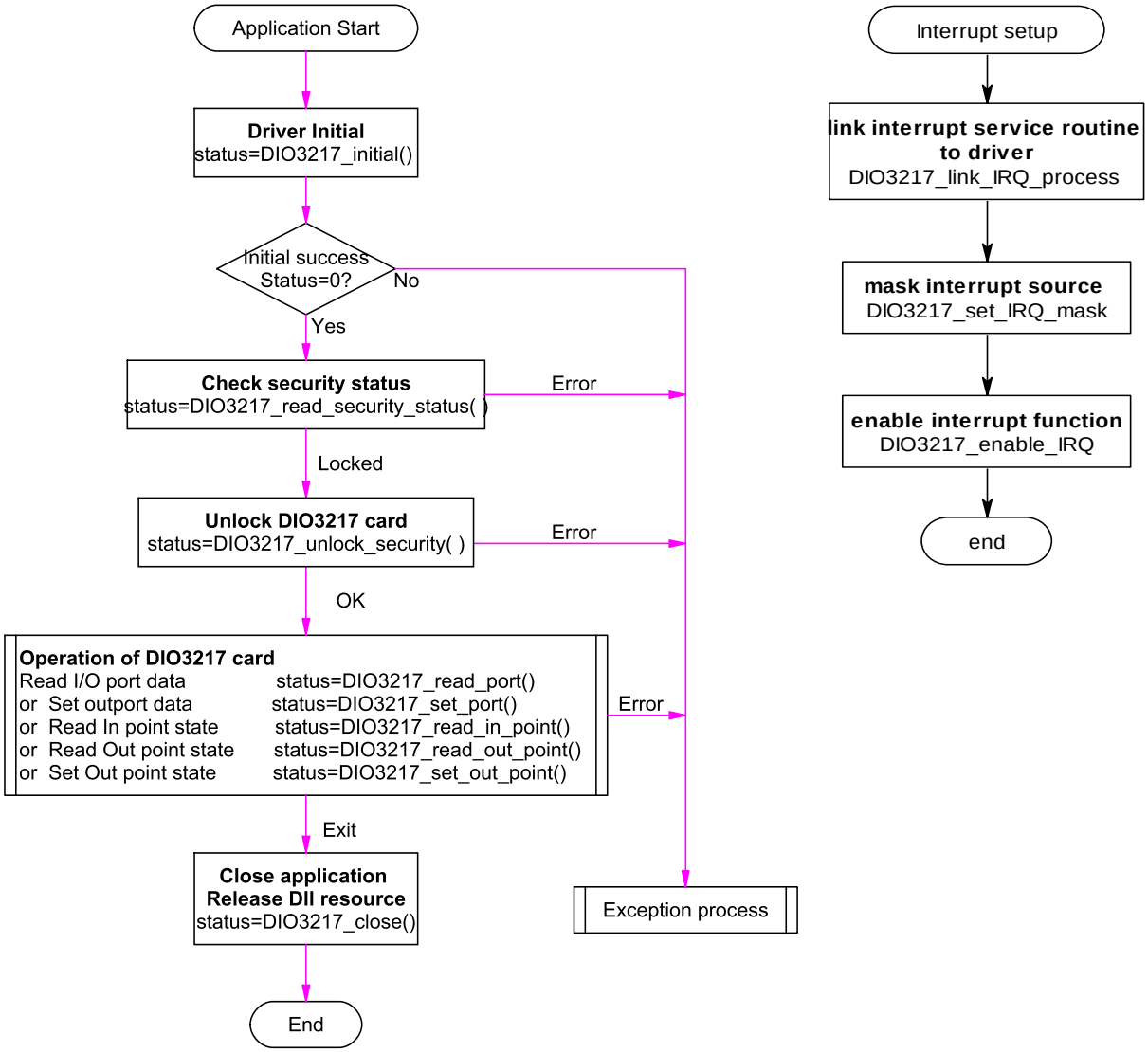
Status = DIO3217_port_read(u8 CardID, u8 port, u8*data);

where **CardID** and **port** are input parameters, and **data** is an output parameter. Consider the following example:

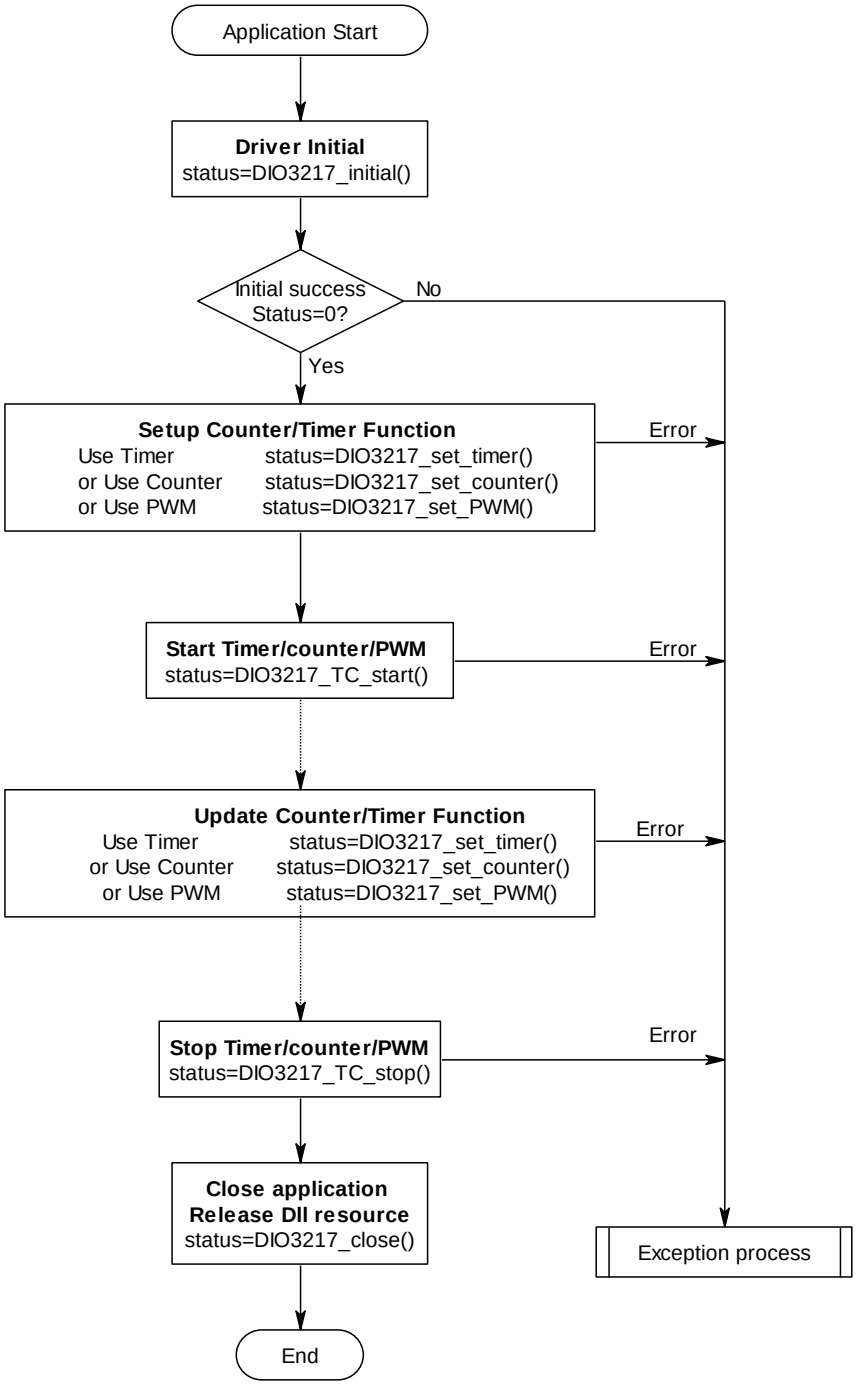
u8 CardID, port;
u8 data;
u32 Status;
Status = DIO3217_port_read(CardID, port, &data);

9. Flow chart of application implementation

9.1 DIO3217 Flow chart of application implementation



9.2 DIO3217 Flow chart of Timer / Counter / PWM application



10. Software overview and dll function

10.1 Initialization and close

You need to initialize system resource each time you run your application.

DIO3217_initial( ) will do.

Once you want to close your application, call

DIO3217_close( ) to release all the resource.

If you want to know the physical address assigned by OS. use

DIO3217_info( ) to get the address .

- **DIO3217_initial**

Format : **u32 status =DIO3217_initial (void)**

Purpose: Initial the DIO3217 resource when start the Windows applications.

- **DIO3217_close**

Format : **u32 status =DIO3217_close (void);**

Purpose: Release the DIO3217 resource when close the Windows applications.

- **DIO3217_info**

Format : **u32 status =DIO3217_info(u8 CardID, u16 *DIO_address, u16 *TC_address);**

Purpose: Read the physical I/O address assigned by O.S.

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY SW

Output:

Name	Type	Description
DIO_address	u16	physical I/O address of DIO assigned by OS
TC_address	u16	physical I/O address of TC assigned by OS

10.2 I/O Port R/W

Before using a input port, if you already know the maximum response time of the input signal you can setup the debounce time to filter out the undesired noise signal and get a noise-free signal. If you do not know the exact response, please use the conservative setting i.e. 100Hz is a common choice.

Use ***DIO3217_debounce_time_set ( )*** to configure the debounce time.

DIO3217_debounce_time_read ( ) to read back the configure of debounce time

Use the following functions for I/O port output value reading and control:

DIO3217_port_polarity_set( ) to set port polarity

DIO3217_port_polarity_read( ) to read port polarity

DIO3217_port_set( ) to output byte data to output port,

DIO3217_port_read( ) to read input port data or read back the output port data,

DIO3217_point_set( ) to set output point,

DIO3217_point_read( ) to read input point data or read back the output point data,

- **DIO3217_debounce_time_set**

Format : **u32 status = DIO3217_debounce_time_set(u8 CardID , u8 port , u8 debounce_time)**

Purpose: Set the input port debounce time

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by Rotary SW
port	u8	port number 0: input port0 for IN00-IN07 1: input port1 for IN10-IN17
debounce_time	u8	Debounce time selection: 0: no debounce 1: drop under 10ms pulse (default,100Hz) 2: drop under 5ms pulse(200Hz) 3: drop under 1ms pulse(1KHz)

- **DIO3217_debounce_time_read**

Format : **u32 status = DIO3217_debounce_time_read (u8 CardID , u8 port , u8 * debounce_time)**

Purpose: Read back the input port debounce time

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by Rotary SW
port	u8	port number 0: input port0 for IN00-IN07 1: input port1 for IN10-IN17

Output:

Name	Type	Description
debounce_time	u8	Debounce time selection: 0: no debounce 1: drop under 10ms pulse (default,100Hz) 2: drop under 5ms pulse(200Hz) 3: drop under 1ms pulse(1KHz)

- **DIO3217_port_polarity_set**

Format : **u32 status = DIO3217_port_polarity_set(u8 CardID, u8 port , u8 polarity)**

Purpose: Set the I/O port polarity.

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by Rotary SW
port	u8	port number 0: input port0 for IN00-IN07 1: input port1 for IN10-IN17 2: output port0 for OUT07~OUT00 3: output port1 for OUT17~OUT10
polarity	u8	polarity values: b7~b0 bit data =0, normal polarity (default) bit data =1, invert polarity

Note:

For example, your card ID is 4 and you want to program output port1 OUT10~OUT11 as invert polarity, you should set CardID=4, port=3,polarity=0000 0011b.

- **DIO3217_port_polarity_read**

Format : **u32 status = DIO3217_port_polarity_read(u8 CardID , u8 port , u8 *polarity)**

Purpose: Read the polarity of the I/O port.

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by Rotary SW
port	u8	port number 0: input port0 for IN07~IN07 1: input port1 for IN17~IN10 2: output port0 for OUT07~OUT00 3: output port1 for OUT17~OUT10

Output:

Name	Type	Description
polarity	u8	polarity data: b7~b0 bit data =0, normal polarity bit data =1, invert polarity

- **DIO3217_port_set**

Format : **u32 status = DIO3217_port_set(u8 CardID, u8 port , u8 state)**

Purpose: Set the output port data.

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by Rotary SW
port	u8	port number 0: invalid 1: invalid 2: output port0 for OUT07~OUT00 3: output port1 for OUT17~OUT10
state	u8	output values: b7~b0

Note: The physical output will depend on the polarity you configured.

- **DIO3217_port_read**

Format : u32 status = DIO3217_port_read (u8 CardID , u8 port , u8 * state)

Purpose: read input port data or read back the output port register data.

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by Rotary SW
port	u8	port number 0: input port0 for IN07~IN07 1: input port1 for IN17~IN10 2: output port0 for OUT07~OUT00 3: output port1 for OUT17~OUT10

Output:

Name	Type	Description
state	u8	I/O data: b7~b0

Note: The physical input state depends on the polarity you configured.

- **DIO3217_point_set**

Format : u32 status =DIO3217_point_set(u8 CardID, u8 port, u8 point, u8 state)

Purpose: set output point.

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by Rotary SW
port	u8	port number 0: invalid 1: invalid 2: output port0 for OUT07~OUT00 3: output port1 for OUT17~OUT10
point	u8	point number 0~7 for OUTx0~OUTx7
state	u8	state of output point 0: make 1: break

Note: The physical output depends on the polarity you configured.

- **DIO3217_point_read**

Format : **u32 Status =DIO3217_point_read(u8 CardID, u8 port, u8 point, u8 *state)**

Purpose: read input point data or read back the output point register data.

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by Rotary SW
port	u8	port number 0: input port0 for IN07~IN07 1: input port1 for IN17~IN10 2: output port0 for OUT07~OUT00 3: output port1 for OUT17~OUT10
point	u8	point number of input 0~7 for INx0~INx7 or OUTx0~OUTx7

Output:

Name	Type	Description
state	u8	state of output point 0: make 1:break

Note: The physical input state depends on the polarity you configured.

10.3 Timer / Counter function

Timer/counter function can work general mode: as timer, as counter or as PWM generator and special mode: quadrature counter mode.

In the general timer/counter mode, IN00, IN01 and OUT00 can be configured as dedicated I/O (disable general purpose I/O).

The gate signal or counter input signal maybe noisy and adequate filtering is recommended to achieve a accuracy result. The 2 special inputs, if used in the Timer/counter function, can be configured the input filtering time constant in a wide range from 1KHz(1ms) to 8MHz(0.125us).

Use: ***DIO3217_TC_debounce_time_set( )*** to configure and read back the configuration by ***DIO3217_TC_debounce_time_read( )***

To configure the working mode use

DIO3217_timer_set( ) to configure as timer and its output mode

DIO3217_counter_set( ) to configure as counter and its input and output mode

DIO3217_PWM_set( ) to configure as PWM generator.

DIO3217_quadrature_set( ) to configure as quadrature counter.

To start/stop the operation by:

DIO3217_TC_start( )

DIO3217_TC_stop( )

To read or load dedicated timer/counter registers, use

DIO3217_TC_set( ) set TC dedicated registers

DIO3217_TC_read( ) read TC dedicated registers

Note1: For timer / counter function, Tin means IN00, Gate mean IN01 for timer / counter.

Note2: For timer / counter function, Tout means OUT00 for timer / counter.

Note3: For quadrature counter mode, IN00 is A phase input IN01 is B phase input

- **DIO3217_TC_debounce_time_set**

Format : **u32 status = DIO3217_TC_debounce_time_set (u8 CardID , u8 port , u8 data)**

Purpose: Set the gate and Ti input debounce time

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by Rotary SW
port	u8	port number 0: Ti (IN00) 1: gate (IN01)
data	u8	Debounce time selection: 0: drop under 1ms pulse(1KHz) 1: drop under 0.5ms pulse(2KHz) (default) 2: drop under 0.25ms pulse(4KHz) 3: drop under 0.125ms pulse(8KHz) 4: drop under 62.5us pulse(16KHz) 5: drop under 31.25us pulse(32KHz) 6: drop under 15.625us pulse(64KHz) 7: drop under 7.8125us pulse(128KHz) 8: drop under 3.9us pulse(256KHz) 9: drop under 1.95us pulse(512KHz) 10: drop under 1us pulse(1MHz) 11: drop under 0.5us pulse(2MHz) 12: drop under 0.25us pulse(4MHz) 13: drop under 0.125us pulse(8MHz)

- **DIO3217_TC_debounce_time_read**

Format : **u32 status = DIO3217_TC_debounce_time_read (u8 CardID , u8 port , u8 *data)**

Purpose: Read back the input port debounce time

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by Rotary SW
port	u8	port number 0: Ti (IN00) 1: gate (IN01)

Output:

Name	Type	Description
data	u8	Debounce time selection: 0: drop under 1ms pulse(1KHz) 1: drop under 0.5ms pulse(2KHz) (default) 2: drop under 0.25ms pulse(4KHz) 3: drop under 0.125ms pulse(8KHz) 4: drop under 62.5us pulse(16KHz) 5: drop under 31.25us pulse(32KHz) 6: drop under 15.625us pulse(64KHz) 7: drop under 7.8125us pulse(128KHz) 8: drop under 3.9us pulse(256KHz) 9: drop under 1.95us pulse(512KHz) 10: drop under 1us pulse(1MHz) 11: drop under 0.5us pulse(2MHz) 12: drop under 0.25us pulse(4MHz) 13: drop under 0.125us pulse(8MHz)

- **DIO3217_timer_set**

Format : u32 status = DIO3217_timer_set (u8 CardID,Timer_struct *TC_struct)

Purpose: To setup timer operation mode or update timer

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY switch

TC_struct	struct	struct TC_struct
		{
		u8 TiGate_MODE,
		// 0: NO_GATE
		//Always count without gate function,
		//IN00 is digital input. //IN01 is digital input
		// 1:GATED
		//IN00 is gate input, after command start_TC,
		//if internal logic active high will start timer and
		//low will halt the timer counting.
		//IN01 is digital input
		// 2: EDGE_START
		//IN00 is gate input, after command start_TC,
		//if internal logic active high will start timer
		//IN01 is digital input
		
		u32 time_constant,
		// Timer constant based on 1us clock
		
		u8 Tout_mode,
		// 0: NO_TOUT ,
		// OUT00 use as general digital output
		// 1: OUT_PULSE
		//OUT00:timer cross zero output pulse.
		//(out_width effective)
		// 2: OUT_LEVEL
		//OUT00: timer cross zero output will make.
		// 4:OUT_TOGGLE
		//OUT00: timer cross zero toggles output
		
		u16 Tout_width,
		// Output pulse width based on 1us clock, only
		//valid in Tout_mode is OUT_PULSE
		
		u8 cont_single
		// 0: SINGLE_CYCLE
		//single cycle mode, timer will stop operation
		//when time constant count down to zero.
		// 1: ALWAYS_RUN
		//continuous operation mode, timer will reload
		//time constant and continue operation when
		//time constant count down to zero.
		}

- **DIO3217_counter_set**

Format : u32 status = DIO3217_counter_set (u8 CardID, Counter_struct *TC_struct)

Purpose: To setup counter operation mode or update counter

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY switch

TC_struct	struct	struct TC_struct
		{
		u8 TiGate_MODE,
		// 0: NO_GATE
		//Always count without gate function,
		//IN00 is digital input. //IN01 is counter pulse input
		// 1:GATED
		//IN00 is gate input, internal logic active high //will start to pass the counter Ti pulse to counter //after command start_TC
		//IN01 is counter pulse input (Ti)
		// 2: EDGE_START
		//IN00 is gate input, internal logic active high //will start to pass the counter Ti pulse to counter //after command start_TC
		//IN01 is counter pulse input (Ti)
		
		u32 counter_constant,
		// Counter constant
		
		u8 Tout_mode,
		// 0: NO_TOUT
		// OUT00 use as general digital output
		// 1: OUT_PULSE
		//OUT00: timer cross zero output pulse. //(out_width effective)
		// 2: OUT_LEVEL
		//OUT00: timer cross zero output will make.
		// 4:OUT_TOGGLE
		//OUT00: timer cross zero toggles output
		
		u16 Tout_width,
		// Output pulse width based on 1us clock, only //valid in Tout_mode is OUT_PULSE
		
		u8 cont_single
		// 0: SINGLE_CYCLE
		//single cycle mode, counter will stop operation //when time constant count down to zero.
		// 1: ALWAYS_RUN
		// continuous operation mode, counter will reload //time constant and continue operation when time //constant count down to zero.
		}

- **DIO3217_PWM_set**

Format : **u32 status = DIO3217_PWM_set(u8 CardID, PWM_struct *PWM_struct)**

Purpose: To setup PWM operation mode or update PWM.

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY switch

PWM_struct	struct	PWM_struct
		{
		u8 TiGate_MODE,
		
		// 0: NO_GATE
		//Always count without gate function,
		//IN00 is digital input.
		//IN01 is digital input
		// 1:GATED
		//IN00 is gate input, internal logic active high
		//will start to pass the counter Ti pulse to counter
		//after command start_TC
		//IN01 is counter pulse input (Ti)
		// 2: OUT_LEVEL
		//OUT00: timer cross zero output will make.
		
		u16 PWM_freq;
		// PWM frequency clock count based on 33MHz
		// clock
		
		u16 PWM_duty;
		//PWM duty clock count based on 33MHz clock
		
		//u16 TO_MODE,
		// **0: NO_TOUT**
		// OUT00 use as general digital output
		// **4:OUT_TOGGLE**
		//OUT00: timer cross zero toggles output
		}

Note:

1. PWM base clock is based on33MHz, say if you want your PWM frequency is 20KHz, please put the PWM_freq = (33MHz/20KHz) = 1650
2. PWM duty must less than PWM_freq for proper operation, from the example above, the PWM_duty value can be 1 ~ 1649. For 50% duty, the PWM_duty will be 1650/2 = 825

- **DIO3217_quadrature_set**

Format : **u32 status = DIO3217_quadrature_set (u8 CardID,u8 Multiple_rate)**

Purpose: To setup quadrature counter operation mode

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY switch
Multiple_rate	u8	Only valid for quadrature mode, in other mode, this parameter is ignored. 0: MULTIPLE_4 (default) A,B phase input multiple rate is 4 1: MULTIPLE_2 A,B phase input multiple rate is 2 2: MULTIPLE_1 A,B phase input multiple rate is 1

- **DIO3217_TC_start**

Format : **u32 status = DIO3217_TC_start (u8 CardID)**

Purpose: To start timer/counter/PWM/quadrature counter operation mode

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY switch

- **DIO3217_TC_stop**

Format : **u32 status = DIO3217_TC_stop (u8 CardID)**

Purpose: To stop timer/counter/PWM/quadrature counter operation mode

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY switch

- **DIO3217_TC_set**

Format : **u32 status=DIO3217_TC_set (u8 CardID,u8 index,u32 data)**

Purpose: To set data to counter/timer register

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY SW
index	u8	0: TC_CONTROL 1: TC_MODE 2: TiGate_MODE 3: To_MODE 4: RETRIGGER_MODE 5: PRELOAD 6: COUNTER 7:OUT_WIDTH 8: MULTIPLE_RATE
data	u32	register data to be set

Note: please refer the next segment "Note: Meaning of setting or return value of different index"

- **DIO3217_TC_read**

Format : u32 status=DIO3217_TC_read (u8 CardID,u8 index,u32 *data)

Purpose: To read data from counter/timer

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY SW
index	u8	0: TC_CONTROL 1: TC_MODE 2: TiGate_MODE 3: Tout_MODE 4: RETRIGGER_MODE 5: PRELOAD 6: COUNTER 7:OUT_WIDTH 8: MULTIPLE_RATE

Output:

Name	Type	Description
data	u32	Data read back

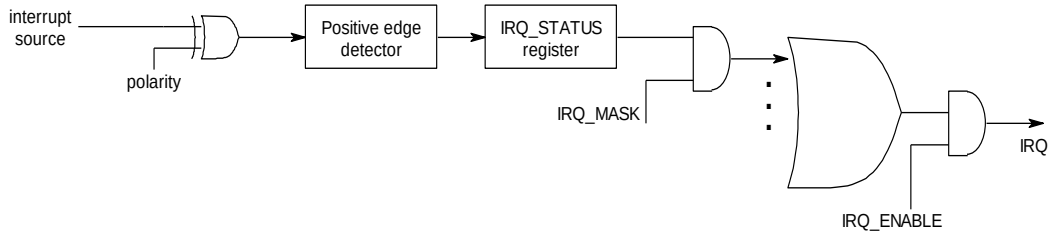
Note: Meaning of setting or return value of different index

index	register	value	meaning
0	TC_CONTROL	0	STOP, stop operation of TC
		1	START, start operation of TC
1	TC_MODE	0	TIMER_MODE
		1	COUNTER_MODE
		3	SW_DEC (a write will software decrease counter by 1 and return to COUNTER_MODE.)
		4	PWM_MODE
2	TiGate_MODE	0	NO_GATE
		1	GATED
		2	EDGE_START
3	Tout_MODE	0	NO_TOUT
		1	OUT_PULSE
		2	OUT_LEVEL
		3	OUT_TOGGLE
4	RETRIGGER_MODE	0	SINGLE_CYCLE
		1	ALWAYS_RUN
5	PRELOAD	1~0xffffffff	Counter or timer or PWM preload value
6	COUNTER	1~0xffffffff	Set (write): will write preload and counter Read : will read counter on the fly
7	OUT_WIDTH	1~0xffff	Output pulse width
8	MULTIPLE_RATE	0	0: MULTIPLE_4 (default) A,B phase input multiple rate is 4
		1	1: MULTIPLE_2 A,B phase input multiple rate is 2
		2	2: MULTIPLE_1 A,B phase input multiple rate is 1

10.4 Interrupt function

Sometimes you want your application to take care of the I/O while special event occurs, interrupt function is the right choice.

The DIO3217 card interrupt model is as follows:



The interrupt source maybe digital input or timer/counter function. For digital input, the interrupt source is IN00~IN07, the physical input is changed polarity by

DIO3217_port_polarity_set**,** then the on board hardware detect the positive edge transition to trigger the IRQ_STATUS register, it is irrelevant to the IRQ_MASK. If the IRQ_MASK is set to 1 and IRQ_ENABLE are also set, the interrupt will generate. By this model, you can see that you can check the fast changing input by IRQ_STATUS without using interrupt.

DIO3217_IRQ_process_link( ) to link the user irq service program to system

DIO3217_IRQ_mask_set( ) to set irq MASK of DIO or TC

DIO3217_IRQ_mask_read( ) to read irq MASK of DIO or TC

DIO3217_IRQ_enable( ) enable interrupt function

DIO3217_IRQ_disable( ) disable interrupt function

DIO3217_IRQ_status_read( ) to read irq STATUS of DIO or TC

- **DIO3217_IRQ_process_link**

Format : u32 status = DIO3217_IRQ_process_link (u8 CardID,

void (__stdcall *callbackAddr)(u8 CardID));

Purpose: Link irq service routine to driver

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by Rotary SW
callbackAddr	void	callback address of service routine

- **DIO3217_IRQ_mask_set**

Format : **u32 status = DIO3217_IRQ_mask_set (u8 CardID, u8 source, u8 mask)**

Purpose: Select interrupt source

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by Rotary SW
source	u8	0:DIO 1:TC
mask	u8	For DIO b7: 0, disable IN07 as interrupt source 1, enable IN07 as interrupt source b6: 0, disable IN06 as interrupt source 1, enable IN06 as interrupt source … b0: 0, disable IN00 as interrupt source 1, enable IN00 as interrupt source for TC b0: 0, disable TC counter counts to zero as interrupt source 1, enable TC counter counts to zero as interrupt source

- **DIO3217_IRQ_mask_read**

Format : **u32 status = DIO3217_IRQ_mask_read (u8 CardID, u8 source, u8 *mask)**

Purpose: read back Selected interrupt source

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by Rotary SW
source	u8	0:DIO 1:TC

Output:

Name	Type	Description
mask	u8	Return the setting value of mask register

- **DIO3217_IRQ_enable**

Format : **u32 status = DIO3217_IRQ_enable (u8 CardID, HANDLE *phEvent)**

Purpose: Enable interrupt from unmasked source

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by Rotary SW

Output:

Name	Type	Description
phEvent	HANDLE	event handle

- **DIO3217_IRQ_disable**

Format : **u32 status = DIO3217_IRQ_disable (u8 CardID)**

Purpose: Disable interrupt from unmasked source

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by Rotary SW

- **DIO3217_IRQ_status_read**

Format : u32 status = DIO3217_IRQ_status_read (u8 CardID, u8 source, u8 * Event_Status)

Purpose: To read back the interrupt status and clears the on board status register

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by Rotary SW
source	u8	0:DIO 1:TC

Output:

Name	Type	Description
Event_Status	u8	For DIO b7: 1, IN07 interrupted b6: 1, IN06 interrupted … b0: 1, IN00 interrupted for TC 1, TC counter counts to zero interrupted

Note: This command will also clear the on board IRQ_status register, the second read will not be correct.

10.5 Software key function

From the dll version 3.0 and later, we remove the software key function owing to some customers complained about the card locked on some unknown occasion. We only remain the functions to comply with the existing programs but the returned value always true.

Since DIO3217 is a general purpose card, anyone who can buy from JS automation Corp. or her distributors. Your program is the fruit of your intelligence, un-authorized copy maybe prevent by the security function enabled.

You can use

DIO3217_password_set( ) to set password and start the security function.

DIO3217_password_change( ) to change it.

If you don't want to use security function after the password being setup,

DIO3217_password_clear( ) will reset to the virgin state.

Once the password is set, any function call of the dll's (except for the security functions) will be blocked until the

DIO3217_security_unlock( ) unlock the security.

You can also use

DIO3217_security_status_read( ) to check the current status of security.

- **DIO3217_password_set**

Format : **u32 status = DIO3217_password_set (u8 CardID,u16 password[5]);**

Purpose: To set password and if the password is not all "0", security function will be enabled.

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY SW
password[5]	u16	Password, 5 words

Note on password: If the password is all "0", the security function is disabled**.**

- **DIO3217_password_change**

Format : **u32 status = DIO3217_password_change (u8 CardID,u16 Oldpassword[5], u16 password[5]);**

Purpose: To replace old password with new password.

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY SW
Oldpassword [5]	u16	The previous password
password[5]	u16	The new password to be set

- **DIO3217_password_clear**

Format : **u32 status = DIO3217_password_clear (u8 CardID,u16 password[5])**

Purpose: To clear password, to set password to all "0", i.e. disable security function.

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY SW
password[5]	u16	The password previous set

- **DIO3217_security_unlock**

Format : **u32 status = DIO3217_security_unlock (u8 CardID,u16 password[5])**

Purpose: To unlock security function and enable the further operation of this card

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY SW
password[5]	u16	The password previous set

- **DIO3217_security_status_read**

Format : **u32 status = DIO3217_security_status_read (u8 CardID,u8 *lock_status, u8 *security_enable);**

Purpose: To read security status for checking if the card security function is unlocked.

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY SW

Output:

Name	Type	Description
lock_status	u8	0: security unlocked 1: locked 2: dead lock (must return to original maker to unlock)
security_enable	u8	0: security function disabled 1: security function enabled

Note on security status:

The security should be unlocked before using any other function of the card, and any attempt to unlock with the wrong passwords more than 10 times will cause the card at dead lock status. Any further operation even with the correct password will not unlock the card. The only way is to send back to the card distributor or the original maker to unlock to virgin state.

10.6 Error conditions

DIO3217 cards minimize error conditions. There are three possible fatal failure modes:

- System Fail Status Bit Valid
- Communication Loss
- Hardware not ready

These error types may indicate an internal hardware problem on the board. Error Codes contains a detailed listing of the error status returned by DIO3217 functions.

11. Dll list

	Function Name	Description
1.	DIO3217_initial( )	DIO3217 Initial
2.	DIO3217_close( )	DIO3217 Close
3.	DIO3217_info( )	get OS. assigned address
4.	DIO3217_debounce_time_set( )	Set the input port debounce time
5.	DIO3217_debounce_time_read( )	Read back the input port debounce time
6.	DIO3217_port_polarity_set( )	Set the I/O port polarity
7.	DIO3217_port_polarity_read( )	Read back the polarity of the I/O port
8.	DIO3217_port_set( )	Set the output port data
9.	DIO3217_port_read( )	read input port or output port register data
10.	DIO3217_point_set( )	set output point
11.	DIO3217_point_read( )	read input point or output point register data
12.	DIO3217_TC_debounce_time_set( )	Set the gate and Ti input debounce time
13.	DIO3217_TC_debounce_time_read( )	Read back the input port debounce time
14.	DIO3217_timer_set( )	setup timer operation mode or update timer
15.	DIO3217_counter_set( )	setup counter operation modeor update counter
16.	DIO3217_PWM_set( )	setup PWM operation mode or update PWM
17.	DIO3217_quadrature_set( )	setup quadrature counter operation mode
18.	DIO3217_TC_start( )	start timer/counter/PWM/quadrature counter operation mode
19.	DIO3217_TC_stop( )	stop timer/counter/PWM/quadrature counter operation mode
20.	DIO3217_TC_set( )	set data to counter/timer register
21.	DIO3217_TC_read( )	read data from counter/timer register
22.	DIO3217_IRQ_process_link( )	Link interrupt service routine to driver
23.	DIO3217_IRQ_mask_set( )	Select interrupt source
24.	DIO3217_IRQ_mask_read( )	read back Selected interrupt source
25.	DIO3217_IRQ_enable( )	Enable interrupt function
26.	DIO3217_IRQ_disable( )	Disable interrupt function
27.	DIO3217_IRQ_status_read( )	Read back irq status
28.	DIO3217_password_set( )	Set software key
29.	DIO3217_password_change( )	Change software key
30.	DIO3217_password_clear( )	Clear software key
31.	DIO3217_security_unlock( )	Unlock software key
32.	DIO3217_security_status_read( )	Read software key status

12. DIO3217 Error codes summary

12.1 DIO3217 Error codes table

Error Code	Symbolic Name	Description
0	JSDRV_NO_ERROR	No error.
1	JSDRV_READ_DATA_ERROR	Read data error
2	JSDRV_INIT_ERROR	Driver initial error
3	JSDRV_UNLOCK_ERROR	Software key unlock error
4	JSDRV_LOCK_COUNTER_ERROR	Software key unlock error count over
5	JSDRV_SET_SECURITY_ERROR	Software key setting error
100	DEVICE_RW_ERROR	Device Read/Write error
101	JSDRV_NO_CARD	No DIO3217 card on the system.
102	JSDRV_DUPLICATE_ID	DIO3217 CardID duplicate error.
300	JSDIO_ID_ERROR	Function input parameter error. CardID setting error, CardID doesn't match the DIP SW setting
301	JSDIO_PORT_ERROR	Function input parameter error. Parameter out of range.
302	JSDIO_IN_POINT_ERROR	Function input parameter error. Parameter out of range.
303	JSDIO_OUT_POINT_ERROR	Function input parameter error. Parameter out of range.
304	JSDIO_VERSION_ERROR	Hardware version can not match with software version
305	JSDIO_SOURCE_ERROR	Interrupt source select error
406	JSDIO_INDEX_ERROR	TC register index error
407	JSDIO_TO_MODE_ERROR	Timer output mode error
408	JSDIO_TI_MODE_ERROR	Timer input mode error
501	DEBOUNCE_MODE_ERROR	Debounce mode error