DIO3264B

Digital I/O Card

Software Manual (V1.0)

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	

Contents

1. Compatibility about the DIO3264, DIO3264A and DIO3264B

The life cycle of industrial control cards normally remain several years but sometimes the chips phase out during its life cycle.

JS Automation tries to keep all the availability of its products before the market phase out. DIO3264A is fully compatible with DIO3264, you can use it without any change of you old design, it also work with the upgrade version DIO3264B. The following tables will give you clear information of all these cards.

card model	compatibility	
	hardware	software
DIO3264	----	use DIO3264.dll
DIO3264A	direct compatible to DIO3264	DIO3264.dll
	direct compatible to DIO3264B with limited hardware function that DIO3264B provides	use DIO3264B.dll but only limit functions that DIO3264 provides
DIO3264B	down compatible to DIO3264	use DIO3264B.dll but with old coding convention [*1] but only limit functions that DIO3264 provides
	down compatible to DIO3264A	use DIO3264B.dll with new or old coding convention[*2] but only limit functions that DIO3264 provides
	----	use DIO3264B.dll with new coding convention that provides full functions

*1: old coding convention: the function call provides in DIO3264 software manual.

*2: new coding convention: the function call provides in DIO3264B software manual.

**We recommend to use new coding convention for DIO3264A for easier to upgrade if need.

2. How to install the software of DIO3264B

2.1 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 WinXP/7 and up system you should:

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 (..\DIO3264B\Software\WinXP_7\ or if you download from website please execute the file DIO3264B_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.

3. Where to find the file you need

WinXP/7 and up

The directory will be located at

.. \ JS Automation \DIO3264B\API\   (header files and lib files for VB,VC,BCB,C#)

.. \ JS Automation \DIO3264B\Driver\   (backup copy of DIO3264B drivers)

.. \ JS Automation \DIO3264B\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.

4. About the DIO3264B software

DIO3264B 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 DIO3264B 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 DIO3264B functions within Windows' operation system environment.

4.1 What you need to get started

To set up and use your DIO3264B software, you need the following:

- DIO3264B software
- DIO3264B hardware

  Main board

  Wiring board (Option)

4.2 Software programming choices

You have several options to choose from when you are programming DIO3264B 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 DIO3264B software.

5. DIO3264B Language support

The DIO3264B software library is a DLL used with WinXP/7 and up. You can use these DLL with any Windows integrating development environment that can call Windows DLLs.

5.1 Building applications with the DIO3264B software library

The DIO3264B function reference topic contains general information about building DIO3264B applications, describes the nature of the DIO3264B files used in building DIO3264B 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

5.2 DIO3264B Windows libraries

The DIO3264B for Windows function library is a DLL called **DIO3264B.dll**. Since a DLL is used, DIO3264B functions are not linked into the executable files of applications. Only the information about the DIO3264B functions in the DIO3264B 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 DIO3264B functions in DIO3264B.dll.

Header Files and Import Libraries for Different Development Environments		
Language	**Header File**	**Import Library**
Microsoft Visual C/C++	DIO3264B.h	DIO3264BVC.lib
Borland C/C++	DIO3264B.h	DIO3264BBC.lib
Microsoft Visual C#	DIO3264B.cs	
Microsoft Visual Basic	DIO3264B.bas	
Microsoft VB.net	DIO3264B.vb	

Table 1

6. 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.

Types of I/O classified 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 break the electrical connection without breaking 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).

Types of Output classified 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.

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 less than half of the sample frequency, the output will generate a beat frequency. The DIO3264B has built-in digital debounce function by hardware, you can choose the debounce frequency from 100Hz, 200Hz, 1KHz up to 10KHz and if you need faster input frequency, you can program it no debounce (only limit by the photo-isolator response time).

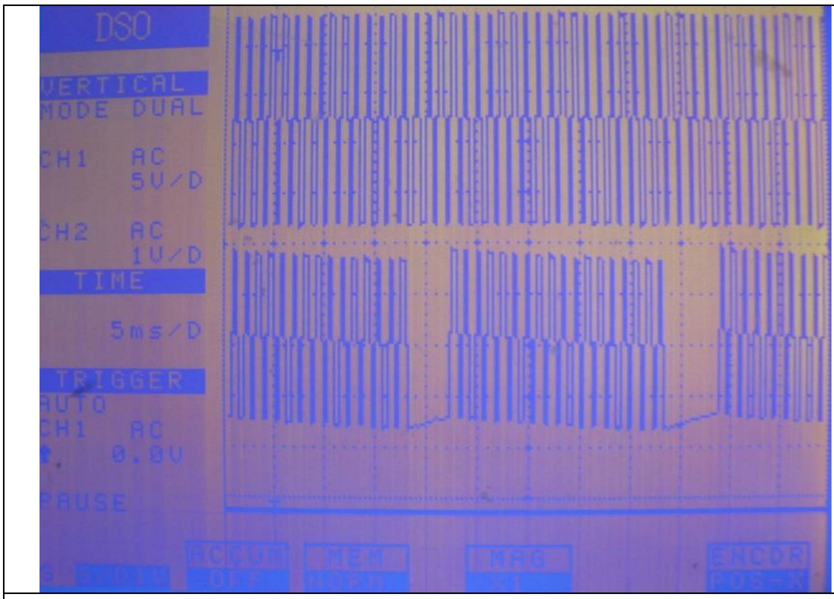
<table>
<tr>
<td>
</td>
<td><- Input frequency at 835Hz
<- Output of digital filter,
Please note the beat frequency.</td>
</tr>
<tr>
<td colspan="2">Digital debounce circuit work at 1KHZ sample rate and observe the output of filter from 835Hz input</td>
</tr>
</table>

fig. 6.1 Digital debounce

You can also implement debounce by software; of course it will consumes the CPU time a lot, we do not recommend to use except for you really know what you want.

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. DIO3264B provides IN0 ~IN15as interrupt input.

Read back of Output status

Some applications need to read back the output status, if the card does 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.**

7. Function format and language difference

7.1 Function format

Every DIO3264B 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.

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

The first parameter to almost every DIO3264B function is the parameter **CardID** which is located the driver of DIO3264B board you want to use those given operation. The **CardID** is assigned by DIP 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 SW (**0x0-0xF**)

7.2 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

7.3 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 DIO3264B API. Read the following sections that apply to your programming language.

Note: Be sure to include the declaration functions of DIO3264B prototypes by including the appropriate DIO3264B header file in your source code. Refer to Building Applications with the DIO3264B Software Library for the header file appropriate to your compiler.

7.3.1 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 = DIO3264B_port_read(CardID, port, 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 = DIO3264B_port_read(CardID, port, &data);
```

7.3.2 Visual basic

The file DIO3264B.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 DIO3264B.bas, do not use the numerical values.

In Visual Basic, you can add the entire DIO3264B.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 DIO3264B.bas file for your project in Visual Basic 4.0, go to the **File** menu and select the **Add File... option**. Select DIO3264B.bas, which is browsed in the DIO3264B \ API directory. Then, select **Open** to add the file to the project.

To add the DIO3264B.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** DIO3264B.bas, which is in the DIO3264B \ API directory. Then, select **Open** to add the file to the project.

7.3.3 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 DIO3264Bbc.lib DIO3264B.dll

Then add the **DIO3264Bbc.lib** to your project and add

#include "DIO3264B.h" to main program.

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

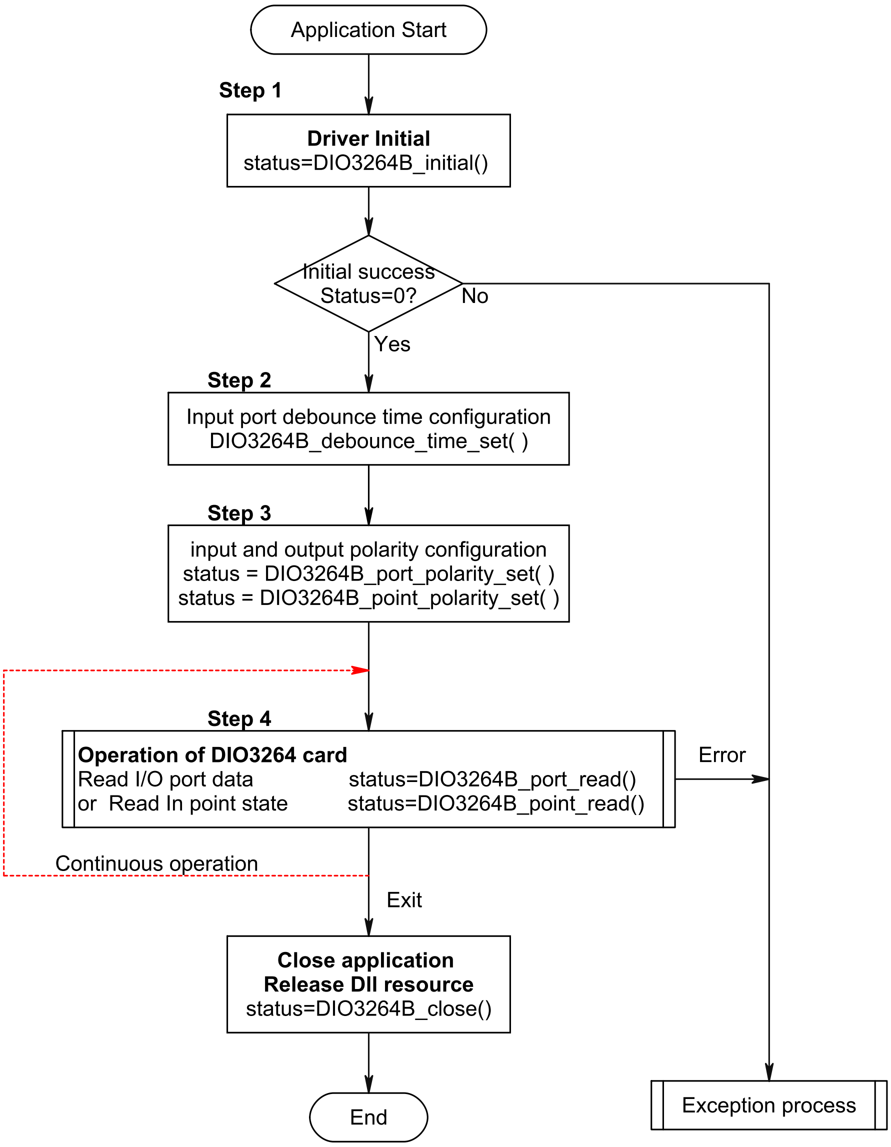
Status = DIO3264B_port_read(CardID, port, data);

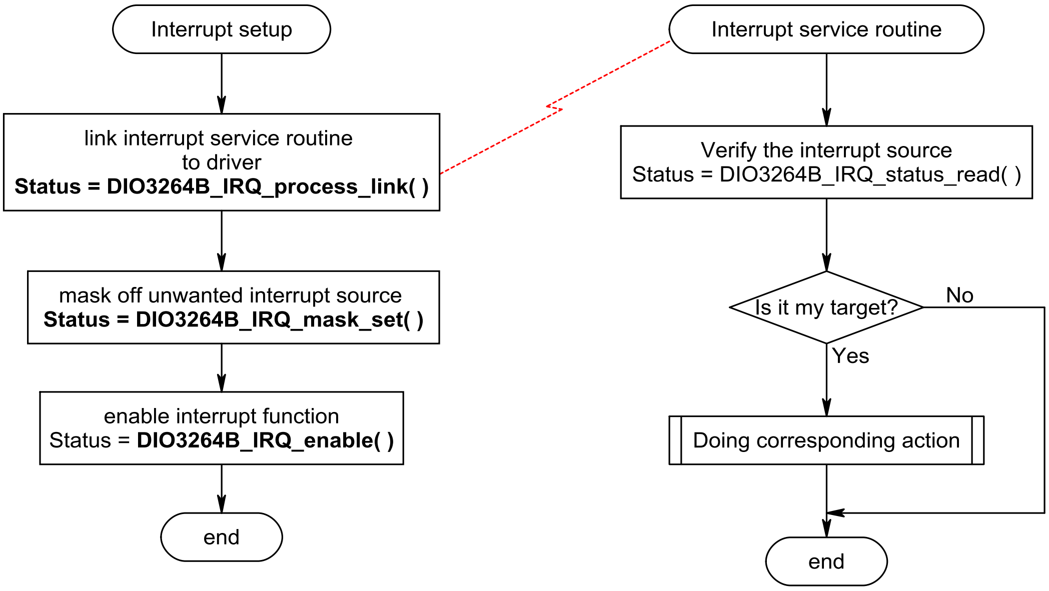
where **CardID** and **port** are input parameters, and **data** is an output parameter. Consider the following example:

```
u16 CardID, port;
u8 data;
u32 Status;
Status = DIO3264B_port_read(CardID, port, &data);
```

8. Flow chart of application implementation

8.1 DIO3264B Flow chart of application implementation


Application Start
Step 1
Driver Initial
status=DIO3264B_initial()
Initial success
Status=0?
No
Yes
Step 2
Input port debounce time configuration
DIO3264B_debounce_time_set( )
Step 3
input and output polarity configuration
status = DIO3264B_port_polarity_set( )
status = DIO3264B_point_polarity_set( )
Step 4
Operation of DIO3264 card
Read I/O port data
status=DIO3264B_port_read()
or Read In point state
status=DIO3264B_point_read()
Error
Continuous operation
Exit
Close application
Release Dll resource
status=DIO3264B_close()
End
Exception process


Interrupt setup
link interrupt service routine
to driver
Status = DIO3264B_IRQ_process_link( )
mask off unwanted interrupt source
Status = DIO3264B_IRQ_mask_set( )
enable interrupt function
Status = DIO3264B_IRQ_enable( )
end
Interrupt service routine
Verify the interrupt source
Status = DIO3264B_IRQ_status_read( )
Is it my target?
No
Yes
Doing corresponding action
end

9. Software overview and dll function

9.1 Initialization

You need to initialize each time you run your application.

DIO3264B_initial( ) to initial the resources of the driver.

DIO3264B_close( ) to close the resources of the driver before you close your application.

DIO3264B_info( ) get the information of address assigned by the OS.

To check the firmware version,

DIO3264B_firmware_version_read( ) will do.

- **DIO3264B_initial**

Format: u32 status =DIO3264B_initial (void)

Purpose: Initial the DIO3264B resource when start the Windows applications.

- **DIO3264B_close**

Format: u32 status =DIO3264B_close (void)

Purpose: Release the DIO3264B resource when close the Windows applications.

- **DIO3264B_info**

Format: u32 status =DIO3264B_info(u8 CardID, u16 *address)

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

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY switch(**0x0-0xF**)

Output:

Name	Type	Description
address	u16	physical I/O address assigned by OS

- **DIO3264B_firmware_version_read**

Format: u32 status =DIO3264B_firmware_version_read(u8 CardID, u8 Version[2])

Purpose: Read the firmware version.

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY switch(**0x0-0xF**)

Output:

Name	Type	Description

Version[2]	u8	the firmware version x.y
		x = Version[1]
		y = Version[0]

9.2 I/O Port R/W

Before using an 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 debounce (sample rate 200Hz) is a common choice.

Use ***DIO3264B_debounce_time_set( )*** to configure the debounce time.

DIO3264B_debounce_time_read( ) to read back the configuration data.

To match the logic polarity of your software, DIO3264B also provides the input polarity configuration; use

DIO3264B_port_polarity_set( ) to configure the polarity of each input of port,

DIO3264B_port_polarity_read( ) to read back the polarity of each input of port.

For the bitwise polarity setting or read back, use

DIO3264B_point_polarity_set( ) to configure the polarity of input;

DIO3264B_point_polarity_read( ) to read back the polarity of input.

For the port input, use:

DIO3264B_port_read( ) to read a byte data from I/O port,

For bitwise control, use

DIO3264B_point_read( ) to read I/O bit,

The input points (IN0~IN15) provide interrupt function to have fast response of input transition. Please refer the 8.7 interrupt function section for detail.

- **DIO3264B_debounce_time_set**

Format: **u32 status = DIO3264B_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 DIP/ROTARY switch(**0x0-0xF**)
port	u8	port number 0: input port0 for IN07-IN00 1: input port1 for IN15-IN08 2: input port2 for IN23-IN16 3: input port3 for IN31-IN24 4: input port4for IN39-IN32 5: input port5 for IN47-IN40 6: input port6 for IN55-IN48 7: input port7 for IN63-IN56
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) 4: drop under 0.1ms pulse(10KHz)

- **DIO3264B_debounce_time_read**

Format: **u32 status = DIO3264B_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 DIP/ROTARY switch(**0x0-0xF**)
port	u8	port number 0~7

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) 4: drop under 0.1ms pulse(10KHz)

- **DIO3264B_port_polarity_set**

Format: **u32 status = DIO3264B_port_polarity_set(u8 CardID, u8 port , u8 polarity)**

Purpose: Sets the I/O port polarity.

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY switch(**0x0-0xF**)
port	u8	port number 0: input port0 for IN07-IN00 1: input port1 for IN15-IN08 2: input port2 for IN23-IN16 3: input port3 for IN31-IN24 4: input port4for IN39-IN32 5: input port5 for IN47-IN40 6: input port6 for IN55-IN48 7: input port7 for IN63-IN56
polarity	u8	bitmap of polarity values take port 1 as example: bit7: IN15 … bit0: IN08 bit data =0, normal polarity (default) bit data =1, invert polarity

- **DIO3264B_port_polarity_read**

Format: **u32 status = DIO3264B_port_polarity_read(u8 CardID , u8 port , u8 *polarity)**

Purpose: Read the I/O port polarity.

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY switch(**0x0-0xF**)
port	u8	port number 0~7

Output:

Name	Type	Description
polarity	u8	bitmap of polarity values take port 6 as example: bit7: IN48 … bit0: IN55 bit data =0, normal polarity (default) bit data =1, invert polarity

- **DIO3264B_point_polarity_set**

Format: **u32 status = DIO3264B_point_polarity_set(u8 CardID, u8 port , u8 point , u8 state)**

Purpose: Sets the I/O point polarity.

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY switch(**0x0-0xF**)
port	u8	port number 0: input port0 for IN07-IN00 1: input port1 for IN15-IN08 2: input port2 for IN23-IN16 3: input port3 for IN31-IN24 4: input port4for IN39-IN32 5: input port5 for IN47-IN40 6: input port6 for IN55-IN48 7: input port7 for IN63-IN56
point	u8	point number 0~7 take port 5 as example, 7: IN40 ... 0: IN47
state	u8	0, normal polarity (default) 1, invert polarity

- **DIO3264B_point_polarity_read**

Format: **u32 status = DIO3264B_point_polarity_read(u8 CardID , u8 port , u8 point , u8 *state)**

Purpose: Read the I/O point polarity.

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY switch(**0x0-0xF**)
port	u8	port number 0~7
point	u8	point number 0~7 take port 4 as example, 7: IN32 ... 0: IN39

Output:

Name	Type	Description
state	u8	0, normal polarity (default) 1, invert polarity

- **DIO3264B_port_read**

Format: **u32 status =DIO3264B_port_read(u8 CardID, u8 port, u8 *data)**

Purpose: Read input port register data.

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY switch(**0x0-0xF**)
port	u8	port number 0: input port0 for IN07-IN00 1: input port1 for IN15-IN08 2: input port2 for IN23-IN16 3: input port3 for IN31-IN24 4: input port4for IN39-IN32 5: input port5 for IN47-IN40 6: input port6 for IN55-IN48 7: input port7 for IN63-IN56

Output:

Name	Type	Description
data	u8	input values: b7~b0

- **DIO3264B_point_read**

Format: **u32 status =DIO3264B_point_read(u8 CardID, u8 port , u8 point , u8 *state)**

Purpose: Read input point register data.

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY switch(**0x0-0xF**)
port	u8	port number 0: input port0 for IN07-IN00 1: input port1 for IN15-IN08 2: input port2 for IN23-IN16 3: input port3 for IN31-IN24 4: input port4for IN39-IN32 5: input port5 for IN47-IN40 6: input port6 for IN55-IN48 7: input port7 for IN63-IN56
point	u8	point number 0~7

Output:

Name	Type	Description
state	u8	state of output point 0: inactive 1: active

9.3 Input Counter (Only valid for DIO3264B)

The DIO3264B IN0~IN7 inputs can work as counter input to 16bit COUNTER0 ~ COUNTER7. The counter model is shown as follows:

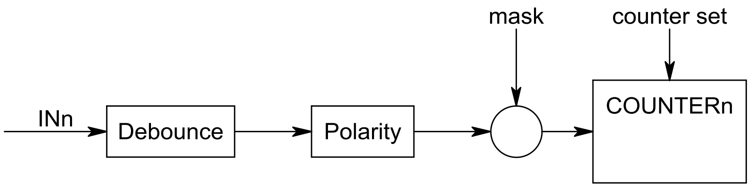


fig. 9.1 counter function model

From the model, you can see the input polarity and debounce block, they are the same as general purpose input (Refer ***DIO3264B_port_polarity_set( ), DIO3264B_debounce_time_set( )***). Mask and counter set are dedicated function of counter. Each counter can be mask off function (stop counting input signal and keep the counter value) or set it to any value.

To use the counter function, the most common is read or set the counter value.

DIO3264B_input_counter_all_set( ) to set values to all counters (set 0 value functions as counter reset) or set single counter by:

DIO3264B_input_counter_all_read( ) or single counter value by:

DIO3264B_input_counter_set( ) and read back all counters' value by:

DIO3264B_input_counter_read( )

If any counter you want to temporary stop or work, you can switch the mask ON/OFF by:

DIO3264B_input_counter_mask_set( ) and read back by

DIO3264B_input_counter_mask_read( )

All the counter function can be enable or disable (similar to real counter power off and the counter value will set to 0) by:

DIO3264B_input_counter_control_set( ) and read back to check the status by:

DIO3264B_input_counter_control_read( )

- **DIO3264B_input_counter_all_set**

Format: **u32 status = DIO3264B_input_counter_all_set(u8 CardID, u16 data[8])**

Purpose: set input counters' data.

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY switch(**0x0-0xF**)
data[8]	u16	data to be set to counters data[7]: for COUNTER7 ... data[0]: for COUNTER0

*Set counter to '0' is equivalent to counter clear.

- **DIO3264B_input_counter_all_read**

Format: **u32 status = DIO3264B_input_counter_all_read(u8 CardID,u16 data[8])**

Purpose: To read input counters' data.

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY switch(**0x0-0xF**)

Output:

Name	Type	Description
data[8]	u16	data set to counters data[7]: for COUNTER7 ... data[0]: for COUNTER0

- **DIO3264B_input_counter_set**

Format: **u32 status = DIO3264B_input_counter_set(u8 CardID,u8 index, u16 data)**

Purpose: set input counter's datum.

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY switch(**0x0-0xF**)
index	u8	counter index 7: for COUNTER7 ... 0: for COUNTER0
data	u16	datum to be set to counter

*Set counter to '0' is equivalent to counter clear.

- **DIO3264B_input_counter_read**

Format: u32 status = DIO3264B_input_counter_read(u8 CardID,u8 index, u16 *data)

Purpose: To read input counter's datum.

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY switch(**0x0-0xF**)
index	u8	counter index 7: for COUNTER7 ... 0: for COUNTER0

Output:

Name	Type	Description
data	u16	counter's datum

- **DIO3264B_input_counter_mask_set**

Format: u32 status = DIO3264B_input_counter_mask_set(u8 CardID, u8 mask)

Purpose: set input counters' operation mask.

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY switch(**0x0-0xF**)
mask	u8	bitmap of input mask value bit7: for COUNTER7 ... bit0: for COUNTER0 If corresponding bit =0, mask off the input, counter will stop and keep the counting value If corresponding bit =1, counter counts the input (if the counter is enabled)

*counter mask off is equivalent to 'Halt' counter operation.

- **DIO3264B_input_counter_mask_read**

Format: u32 status = DIO3264B_input_counter_mask_read(u8 CardID,u8 * mask)

Purpose: To read input counter operation mask.

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY switch(**0x0-0xF**)

Output:

Name	Type	Description
mask	u8	bitmap of input mask value bit7: for COUNTER7 ... bit0: for COUNTER0 If corresponding bit =0, mask off the input, counter will stop and keep the counting value If corresponding bit =1, counter counts the input (if the counter is enabled)

- **DIO3264B_input_counter_control_set**

Format: u32 status = DIO3264B_input_counter_control_set(u8 CardID, u8 control)

Purpose: set input counter control.

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY switch(**0x0-0xF**)
control	u8	COUNTER control: 0: disable, all counter stops and clear to 0 1: enable, counters work as input trigger (if no mask)

- **DIO3264B_input_counter_control_read**

Format: u32 status = DIO3264B_input_counter_control_read(u8 CardID, u8 *control)

Purpose: To read input counter control status

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY switch(**0x0-0xF**)

Output:

Name	Type	Description
control	u8	COUNTER control: 0: disable, all counter stops and clear to 0 1: enable, counters work as input trigger (if no mask)

9.4 TTL I/O Port R/W

DIO3264B has not only isolated input/output ports but it also provides 2 TTL I/O ports which are more flexible for non-isolated application. The ports can be configured as input or output on port base. The port can be set at normal high or normal low voltage during power on by the on card jumper JP1 and JP2. (Refer Hardware manual, chapter 8 Hardware settings)

To configure the port as input or output by:

DIO3264B_TTL_IO_config_set( ) and read back the configuration by:

DIO3264B_TTL_IO_config_read( ).

To change the polarity as you need by:

DIO3264B_TTL_IO_polarity_set( ) and read back to verify by:

DIO3264B_TTL_IO_polarity_read( ).

For the bitwise polarity set and read, use:

DIO3264B_TTL_IO_point_polarity_set( ) and read back to verify by:

DIO3264B_TTL_IO_point_polarity_read( ).

At noisy environment, maybe you need debounce function to keep the signal integrity; TTL IO also provides digital input debounce function. There are 4 ranges: 100Hz, 200Hz, 1KHz ... up to 10MHz and no debounce to select for your application, use:

DIO3264B_TTL_IO_debounce_time_set( ) to set the adequate time constant to drop out the noise and read back to check the setting by:

DIO3264B_TTL_IO_debounce_time_read( ).

After the configuration is complete, you can enable the port to function or disable it any time you want, the output of the disabled port will remain at high or low depends on the jumper setting. Use:

DIO3264B_TTL_IO_enable( ) to enable the port and

DIO3264B_TTL_IO_disable( ) to disable the port.

The TTL I/O port can use:

DIO3264B_TTL_IO_port_set( ) to output data and input data by:

DIO3264B_TTL_IO_port_read( ).

For the bitwise point output, use:

DIO3264B_TTL_IO_point_set( ) and point input by:

DIO3264B_TTL_IO_point_read( ).

- **DIO3264B_TTL_IO_config_set**

Format: **u32 status =DIO3264B_TTL_IO_config_set (u8 CardID, u8 port, u8 config)**

Purpose: Set port configuration.

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY switch(**0x0-0xF**)
port	u8	port number 0: port0 , IO07~IO00 1: port1 , IO17~IO10
config	u8	0: output port (default) 1: input port

- **DIO3264B_TTL_IO_config_read**

Format: **u32 status =DIO3264B_TTL_IO_config_read (u8 CardID, u8 port, u8 *config, u8 *control)**

Purpose: read port configure and control status.

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY switch(**0x0-0xF**)
port	u8	port number 0: port0 , IO07~IO00 1: port1 , IO17~IO10

Output:

Name	Type	Description
config	u8	0: output port (default) 1: input port
control	u8	0: Disable 1: Enable

- **DIO3264B_TTL_IO_polarity_set**

Format: **u32 status =DIO3264B_TTL_IO_polarity_set (u8 CardID, u8 port, u8 polarity)**

Purpose: Sets the I/O polarity of port0~ port1

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY switch(**0x0-0xF**)
port	u8	port number 0: port0 , IO07~IO00 1: port1 , IO17~IO10
polarity	u8	polarity values. take port1 as example: bit7: IO17 ... bit0: IO10 bit data =0, normal polarity bit data =1, invert polarity

- **DIO3264B_TTL_IO_polarity_read**

Format: **u32 status = DIO3264B_TTL_IO_polarity_read (u8 CardID, u8 port, u8 * polarity)**

Purpose: Read the I/O polarity of the port0~port1.

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY switch(**0x0-0xF**)
port	u8	port number 0: port0 , IO07~IO00 1: port1 , IO17~IO10

Output:

Name	Type	Description
polarity	u8	polarity values. take port0 as example: bit7: IO07 ... bit0: IO00 bit data =0, normal polarity bit data =1, invert polarity

- **DIO3264B_TTL_IO_point_polarity_set**

Format: **u32 status =DIO3264B_TTL_IO_point_polarity_set (u8 CardID, u8 port, u8 point, u8 state)**

Purpose: Sets the I/O point polarity of port0~ port1

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY switch(**0x0-0xF**)
port	u8	port number 0: port0 , IO00~IO00 1: port1 , IO17~IO10
point	u8	point number 0~7 take port0 as example: 7: IO07 ... 0: IO00
state	u8	polarity value. bit data =0, normal polarity bit data =1, invert polarity

- **DIO3264B_TTL_IO_point_polarity_read**

Format: **u32 status = DIO3264B_TTL_IO_point_polarity_read (u8 CardID, u8 port, u8 point, u8 * state)**

Purpose: Read the I/O point polarity of the port0~port1.

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY switch(**0x0-0xF**)
port	u8	port number 0: port0 , IO07~IO00 1: port1 , IO17~IO10
point	u8	point number 0~7 take port1 as example: 7: IO17 ... 0: IO10

Output:

Name	Type	Description
state	u8	polarity value. bit data =0, normal polarity bit data =1, invert polarity

- **DIO3264B_TTL_IO_debounce_time_set**

Format: **u32 status = DIO3264B_TTL_IO_debounce_time_set (u8 CardID,u8 port , u8 debounce_time)**

Purpose: debounce time of the TTL I/O port signal

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY switch(**0x0-0xF**)
port	u8	port number 0: port0 , IO07~IO00 1: port1 , IO17~IO10
debounce_time	u8	Debounce time selection: 0: no debounce 1: filter out duration less than 10ms(100Hz) 2: filter out duration less than 5ms(200Hz) 3: filter out duration less than 1ms(1KHz) 4: filter out duration less than 100us(10KHz) (default) 5: filter out duration less than 20us(50KHz) 6: filter out duration less than 10us(100KHz) 7: filter out duration less than 2us(500KHz) 8: filter out duration less than 1us(1MHz) 9: filter out duration less than 0.5us(2MHz) 10: filter out duration less than 0.25us(4MHz) 11: filter out duration less than 0.125us(8MHz) 12: filter out duration less than 0.01us(10MHz)

Note: only valid for TTL port configured as input

- **DIO3264B_TTL_IO_debounce_time_read**

Format: u32 status = DIO3264B_TTL_IO_debounce_time_read (u8 CardID,u8 port , u8 *debounce_time)

Purpose: To read back configuration of debounce mode

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY switch(**0x0-0xF**)
port	u8	port number 0: port0 , IO07~IO10 1: port1 , IO17~IO10

Output:

Name	Type	Description
debounce_time	u8	Debounce time selection: 0: no debounce 1: filter out duration less than 10ms(100Hz) 2: filter out duration less than 5ms(200Hz) 3: filter out duration less than 1ms(1KHz) 4: filter out duration less than 100us(10KHz) (default) 5: filter out duration less than 20us(50KHz) 6: filter out duration less than 10us(100KHz) 7: filter out duration less than 2us(500KHz) 8: filter out duration less than 1us(1MHz) 9: filter out duration less than 0.5us(2MHz) 10: filter out duration less than 0.25us(4MHz) 11: filter out duration less than 0.125us(8MHz) 12: filter out duration less than 0.01us(10MHz)

- **DIO3264B_TTL_IO_enable**

Format: u32 status =DIO3264B_TTL_IO_enable (u8 CardID, u8 port)

Purpose: Enable TTL IO. Only enabled port can be input or output.

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY switch(**0x0-0xF**)
port	u8	port number 0: port0 , IO07~IO00 1: port1 , IO17~IO10

- **DIO3264B_TTL_IO_disable**

Format: **u32 status =DIO3264B_TTL_IO_disable (u8 CardID, u8 port)**

Purpose: Disable TTL IO. The output will be high or low depends on the JP1, JP2 setting.

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY switch(**0x0-0xF**)
port	u8	port number 0: port0 , IO07~IO00 1: port1 , IO17~IO10

- **DIO3264B_TTL_IO_port_set**

Format: **u32 status = DIO3264B_TTL_IO_port_set (u8 CardID,u8 port, u8 data)**

Purpose: Sets the output data.

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY switch(**0x0-0xF**)
port	u8	port number 0: port0 , IO07~IO00 1: port1 , IO17~IO10
data	u8	bitmap of output values take port0 as example: bit7: IO07 ... bit0: IO00

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

- **DIO3264B_TTL_IO_port_read**

Format: **u32 status = DIO3264B_TTL_IO_port_read (u8 CardID , u8 port , u8 *data)**

Purpose: Read the output data.

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY switch(**0x0-0xF**)
port	u8	port number 0: port0 , IO07~IO00 1: port1 , IO17~IO10

Output:

Name	Type	Description
data	u8	bitmap of port values port1 as example: bit7: IO17 ... bit0: IO10

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

- **DIO3264B_TTL_IO_point_set**

Format: **u32 status =DIO3264B_TTL_IO_point_set (u8 CardID, u8 port , u8 point, u8 state)**

Purpose: Sets the bit data of output port.

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY switch(**0x0-0xF**)
port	u8	port number 0: port0 , IO07~IO00 1: port1 , IO17~IO10
point	u8	point number 0~7 take port0 as example: 7: IO07 ... 0: IO00
state	u8	point of output state 0: inactive 1: active

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

DIO3264B_TTL_IO_point_read

Format: **u32 status =DIO3264B_TTL_IO_point_read (u8 CardID, u8 port , u8 point, u8 *state)**

Purpose: Read the output port state.

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY switch(**0x0-0xF**)
port	u8	port number 0: port0 , IO07~IO00 1: port1 , IO17~IO10
point	u8	point number 0~7 take port1 as example: 7: IO17 ... 0: IO10

Output:

Name	Type	Description
state	u8	point of output state 0: inactive 1: active

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

9.5 Timer function (Only valid for DIO3264B)

The timer works based on the 1us clock to give an accuracy counting of time period. It is a 32bit counter. At the end of time period, it can generate an interrupt to trigger event to request service. The timer block function shown as follows:

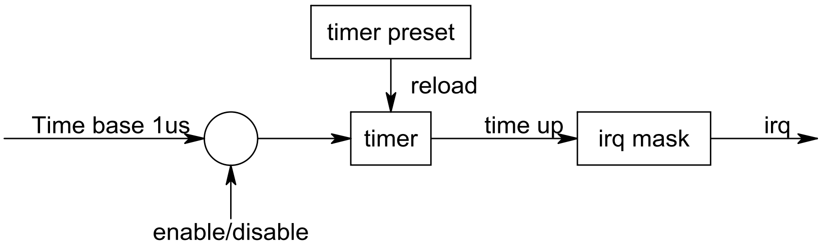


fig. 9.2 timer function model

You can set the timer constant by

DIO3264B_timer_set( ) and use

DIO3264B_timer_start( ) to star its operation,

DIO3264B_timer_stop( ) to stop operation.

For the timer related registers use:

DIO3264B_timer_read( ) to read back registers.

The timer can also trigger interrupt at timer up, please refer to 8.7 interrupt function for detail.

- **DIO3264B_timer_set**

Format: u32 status = DIO3264B_timer_set(u8 CardID, u32 time_constant)

Purpose: set time constant.

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY switch(**0x0-0xF**)
time_constant	u32	time_constant based on 1us time base

Note:

1. Time constant is based on 1us clock, period T= (time_constant +1) * 1us

2. If you also enable the timer interrupt, the period T must at least larger than the system interrupt response time else the system will be hanged by excess interrupts.

- **DIO3264B_timer_start**

Format: u32 status = DIO3264B_timer_start (u8 CardID)

Purpose: start timer function.

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY switch(**0x0-0xF**)

- **DIO3264B_timer_stop**

Format: **u32 status = DIO3264B_timer_stop (u8 CardID)**

Purpose: stop timer function.

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY switch(**0x0-0xF**)

- **DIO3264B_timer_read**

Format: **u32 status= DIO3264B_timer_read (u8 CardID, u8 index, u32 * data)**

Purpose: Read back the setting of timer related registers

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY switch(**0x0-0xF**)
index	u8	0: Timer start status 1: Time constant (preset data) 2: Current time data

Output:

Name	Type	Description

data	u32	if index=0,
		data=0, Timer stops
		data=1, Timer run
		if index=1,
		data=1~4294967295,
		the preset time constant
		if index=2,
		data=0~4294967295, the time on the fly

9.6 Interrupt function (**superset of DIO3264A**)

The DIO3264B card provides timer and inputs IN0 ~ IN15 as interrupt sources. The interrupt will trigger the system to get a quick service.

To use interrupt function, we must have some ideas of the function in advance:

1. The interrupt function is simulated by the system event. It can run on user's level.
2. The interrupt function is suggested to be as quick as possible and not to disturb the normal system operation.

In practice, we must prepare interrupt service routine first. It is the function you want to run after the interrupt occurs. Use

DIO3264B_IRQ_process_link( ) then setup the IRQ mask for the interrupt by:

DIO3264B_IRQ_mask_set( ) to select IN0~IN15 as source of IRQ.

DIO3264B_IRQ_mask_read( )

After setup, you can enable the IRQ by:

DIO3264B_IRQ_enable( ) and also you can disable IRQ by:

DIO3264B_IRQ_disable( )

On the service routine, you can check the interrupt source (if multiple interrupt source) by:

DIO3264B_IRQ_status_read( ), if you do not use IRQ function or the sources you have mask off, you can still get the information for polling process.

- **DIO3264B_IRQ_process_link**

Format: **u32 status = DIO3264B_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 DIP/ROTARY switch(**0x0-0xF**)
callbackAddr	void	callback address of service routine

- **DIO3264B_IRQ_mask_set**

Format: **u32 status = DIO3264B_IRQ_mask_set(u8 CardID, u32 Data)**

Purpose: Mask interrupt from timer and IN0~IN15

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY switch(**0x0-0xF**)
Data	u32	bit00: 1, irq source from IN0 bit01: 1, irq source from IN1 bit02: 1, irq source from IN2 bit03: 1, irq source from IN3 bit04: 1, irq source from IN4 bit05: 1, irq source from IN5 bit06: 1, irq source from IN6 bit07: 1, irq source from IN7 bit08: 1, irq source from IN8 bit09: 1, irq source from IN9 bit10: 1, irq source from IN10 bit11: 1, irq source from IN11 bit12: 1, irq source from IN12 bit13: 1, irq source from IN13 bit14: 1, irq source from IN14 bit15: 1, irq source from IN15 bit16: 1, irq source from T/C If corresponding bit =0, mask off the interrupt

- **DIO3264B_IRQ_mask_read**

Format: **u32 status = DIO3264B_IRQ_mask_read(u8 CardID, u32 *Data)**

Purpose: Read Mask interrupt from timer and IN0~IN15

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY switch(**0x0-0xF**)

Output:

Name	Type	Description
Data	u32	the same as above

- **DIO3264B_IRQ_enable**

Format: **u32 status = DIO3264B_IRQ_enable(u8 CardID, HANDLE *phEvent)**

Purpose: Enable interrupt from timer and IN0~IN15

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY switch(**0x0-0xF**)

Output:

Name	Type	Description
phEvent	HANDLE	event handle

- **DIO3264B_IRQ_disable**

Format: **u32 status = DIO3264B_IRQ_disable(u8 CardID)**

Purpose: Disable interrupt from timer and IN0~IN15

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY switch(**0x0-0xF**)

- **DIO3264B_IRQ_status_read**

Format: **u32 status = DIO3264B_IRQ_status_read(u8 CardID, u32 *Event_Status)**

Purpose: To read back the interrupt source to identify

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY switch(**0x0-0xF**)

Output:

Name	Type	Description
Event_Status	u32	bit00: 1, irq source from IN0 bit01: 1, irq source from IN1 bit02: 1, irq source from IN2 bit03: 1, irq source from IN3 bit04: 1, irq source from IN4 bit05: 1, irq source from IN5 bit06: 1, irq source from IN6 bit07: 1, irq source from IN7 bit08: 1, irq source from IN8 bit09: 1, irq source from IN9 bit10: 1, irq source from IN10 bit11: 1, irq source from IN11 bit12: 1, irq source from IN12 bit13: 1, irq source from IN13 bit14: 1, irq source from IN14 bit15: 1, irq source from IN15 bit16: 1, irq source from T/C

Note: The status does not affect by the IRQ mask on or off If you do not use the interrupt function, you can use the status for polling purpose to take action.

10. Dll list

	Function Name	Description
1.	DIO3264B_initial( )	DIO3264B initial
2.	DIO3264B_close( )	DIO3264B close
3.	DIO3264B_info( )	get OS. assigned address
4.	DIO3264B_firmware_version_read( )	Read device firmware version
5.	DIO3264B_debounce_time_set( )	setup input port debounce time
6.	DIO3264B_debounce_time_read( )	read back input port debounce time
7.	DIO3264B_port_polarity_set( )	setup port polarity
8.	DIO3264B_port_polarity_read( )	read back port polarity
9.	DIO3264B_point_polarity_set( )	setup point polarity
10.	DIO3264B_point_polarity_read( )	read back point polarity
11.	DIO3264B_port_read( )	read port data (byte)
12.	DIO3264B_point_read( )	read output point state (bit)
13.	DIO3264B_input_counter_all_set( )	set all input counters’ data
14.	DIO3264B_input_counter_all_read( )	read all input counters’ data
15.	DIO3264B_input_counter_set( )	set one input counters’ datum
16.	DIO3264B_input_counter_read( )	read one input counters’ datum
17.	DIO3264B_input_counter_mask_set( )	set input counters’ operation mask
18.	DIO3264B_input_counter_mask_read( )	read back input counters’ operation mask
19.	DIO3264B_input_counter_control_set( )	set input counter control
20.	DIO3264B_input_counter_control_read( )	read back input counter control status
21.	DIO3264B_TTL_IO_config_set( )	setup TTL port I/O configuration
22.	DIO3264B_TTL_IO_config_read( )	read back TTL port I/O configuration
23.	DIO3264B_TTL_IO_polarity_set( )	setup TTL port polarity
24.	DIO3264B_TTL_IO_polarity_read( )	read back TTL port polarity
25.	DIO3264B_TTL_IO_point_polarity_set( )	setup TTL point polarity
26.	DIO3264B_TTL_IO_point_polarity_read( )	read back TTL point polarity
27.	DIO3264B_TTL_IO_debounce_time_set( )	setup TTL port input debounce time
28.	DIO3264B_TTL_IO_debounce_time_read( )	read back TTL port input debounce time
29.	DIO3264B_TTL_IO_enable( )	enable TTL IO function
30.	DIO3264B_TTL_IO_disable( )	disable TTL IO function
31.	DIO3264B_TTL_IO_port_set( )	set TTL IO port data
32.	DIO3264B_TTL_IO_port_read( )	read TTL IO port data
33.	DIO3264B_TTL_IO_point_set( )	set TTL IO point data
34.	DIO3264B_TTL_IO_point_read( )	read TTL IO point data
35.	DIO3264B_timer_set( )	set timer time constant
36.	DIO3264B_timer_start( )	start timer function
37.	DIO3264B_timer_stop( )	stop timer function
38.	DIO3264B_timer_read( )	read timer related registers
39.	DIO3264B_IRQ_process_link( )	link interrupt service routine to driver
40.	DIO3264B_IRQ_mask_set( )	set interrupt mask
41.	DIO3264B_IRQ_mask_read( )	read interrupt mask
42.	DIO3264B_IRQ_enable( )	enable interrupt function
43.	DIO3264B_IRQ_disable( )	disable interrupt function
44.	DIO3264B_IRQ_status_read( )	read back IRQ status

11. Port-point reference table

11.1 DIO3264B I/O Port-Point table

DIO3264B I/O Port table

Bit Port	***Bit 7***	***Bit 6***	***Bit 5***	***Bit 4***	***Bit 3***	***Bit 2***	***Bit 1***	***Bit 0***
Port 0	IN 7	IN 6	IN 5	IN 4	IN 3	IN 2	IN 1	IN 0
Port 1	IN 15	IN 14	IN 13	IN 12	IN 11	IN 10	IN 9	IN 8
Port 2	IN 23	IN 22	IN 21	IN 20	IN 19	IN 18	IN 17	IN 16
Port 3	IN 31	IN 30	IN 29	IN 28	IN 27	IN 26	IN 25	IN 24
Port 4	IN 39	IN 38	IN 37	IN 36	IN 35	IN 34	IN 33	IN 32
Port 5	IN 47	IN 46	IN 45	IN 44	IN 43	IN 42	IN 41	IN 40
Port 6	IN 55	IN 54	IN 53	IN 52	IN 51	IN 50	IN 49	IN 48
Port 7	IN 63	IN 62	IN 61	IN 60	IN 59	IN 58	IN 57	IN 56

DIO3264B TTL I/O Port table

Bit Port	***Bit 7***	***Bit 6***	***Bit 5***	***Bit 4***	***Bit 3***	***Bit 2***	***Bit 1***	***Bit 0***
Port 0	IO 7	IO 6	IO 5	IO 4	IO 3	IO 2	IO 1	IO 0
Port 1	IO 15	IO 14	IO 13	IO 12	IO 11	IO 10	IO 9	IO 8

12. DIO3264B Error codes summary

12.1 DIO3264B Error codes table

Error Code	Symbolic Name	Description
0	JSDRV_NO_ERROR	No error.
2	JSDRV_INIT_ERROR	Driver initial error
3	JSDRV_UNLOCK_ERROR	Security unclock failure
4	JSDRV_LOCK_COUNTER_ERROR	Dead lock, unclock failure more than 10 times
5	SDRV_SET_SECURITY_ERROR	Password overwrite error
100	DEVICE_RW_ERROR	Device Read/Write error
101	JSDRV_NO_CARD	No DIO3264B card on the system.
102	JSDRV_DUPLICATE_ID	DIO3264B CardID duplicate error.
300	JSDIO_ID_ERROR	Function input parameter error. CardID setting error, CardID doesn’t match the DIP SW setting
301	PORT_ERROR	port parameter
302	POINT_ERROR	point parameter
303	DATA_ERROR	data parameter
304	STATE_ERROR	state parameter
305	MODE_ERROR	mode parameter
306	INDEX_ERROR	index parameter
307	CONFIG_ERROR	config parameter
308	CONTROL_ERROR	control parameter
309	TIME_ERROR	time parameter
310	POLARITY_ERROR	polarity parameter