DIO8265

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	New

Contents

1. How to install the software of DIO8265

1.1 Install the PCIe driver

The PCIe 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 PCIe card
5. Do not response to the wizard, just Install the file (..\DIO8265\Software\WinXP_7\ or if you download from website please execute the file DIO8265_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

WinXP/7 and up

The directory will be located at

.. \ JS Automation \DIO8265\API\   (header files and lib files for VB,VC,BCB,C#)

.. \ JS Automation \DIO8265\Driver\   (backup copy of DIO8265 drivers)

.. \ JS Automation \DIO8265\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 DIO8265 software

DIO8265 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 DIO8265 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 DIO8265 functions within Windows’ operation system environment.

3.1 What you need to get started

To set up and use your DIO8265 software, you need the following:

- DIO8265 software
- DIO8265 hardware
  - Main board
  - Wiring board (Option)

3.2 Software programming choices

You have several options to choose from when you are programming DIO8265 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 DIO8265 software.

4. DIO8265 Language support

The DIO8265 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.

4.1 Building applications with the DIO8265 software library

The DIO8265 function reference topic contains general information about building DIO8265 applications, describes the nature of the DIO8265 files used in building DIO8265 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

4.2 DIO8265 Windows libraries

The DIO8265 for Windows function library is a DLL called **DIO8265.dll**. Since a DLL is used, DIO8265 functions are not linked into the executable files of applications. Only the information about the DIO8265 functions in the DIO8265 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 DIO8265 functions in DIO8265.dll.

Header Files and Import Libraries for Different Development Environments		
Language	**Header File**	**Import Library**
Microsoft Visual C/C++	DIO8265.h	DIO8265VC.lib
Borland C/C++	DIO8265.h	DIO8265BC.lib
Microsoft Visual C#	DIO8265.cs	
Microsoft Visual Basic	DIO8265.bas	
Microsoft VB.net	DIO8265.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 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).

5.2 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. For the practical application, using relay to switch an inductive load is common but the surge voltage and current during on or off will damage the relay contact and shorten its life. **To avoid the inductance induced high voltage, JS Automation supplies a special version wiring board with ZNR (a device will limit the induced voltage) to solve the relay contact problem.**

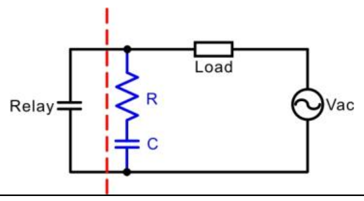
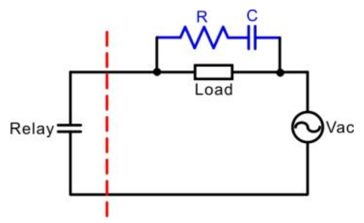
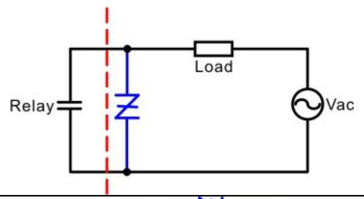
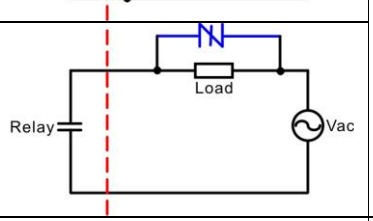
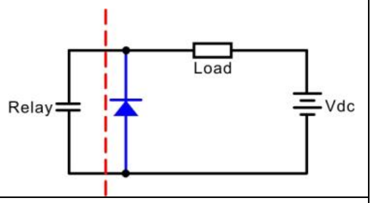
Transistor and MOS FET are basically semi-permanent devices. If you have selected the right ratings, it can work without switching life limit. But the transistor or MOS FET can only work in DC load condition.

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 Protection of output transient on inductive load

No matter what type of the output driver, to switch the inductive load is potentially to induce transient high voltage. The induced high voltage will reach several KV even the load voltage only several volts. Semiconductor driver will punched through by the high voltage to cause the output a fatal error, even the rely contact can also results in micro-weld akin to spot welding. This will case the relay contact welding together (always make) or bad conduction (always break) or abnormally make or break. Owing to the popular wiring board driver components can be NMOS, PMOS or relay. The former two are semiconductor type and limit to DC load. The relay output may be AC or DC load, we will put emphasis on relay and apply the result to NMOS and PMOS.

	Protection Connection	Comments	Design Note
A		Leakage current to load maybe cause malfunction. The effective release time will be lengthened.	Energy stored in the coil will dissipate by R and coil resistance. R: 0.5-1 Ohm per 1V contact voltage C: 0.5-1 uF per 1A current Use capacitor voltage of 200-300V and non-polarized type (AC capacitor) for DC operating voltage. If AC operating condition, the capacitor should at least 20% higher than working voltage.
B		The effective release time will be lengthened.	
C		To prevent the excess high voltage to damage the contact. There only slightly release delay at the release time.	ZNR rated voltage must select at least 20% higher than the working voltage. Selection of power of ZNR depends on the operation frequency, higher frequency needs larger power one. If possible, the diagram D is better than diagram C.
D			
E		The effective release time will be lengthened longer than RC snubber.	Energy stored in the coil will dissipate by coil resistance. (Only for DC working voltage) If possible, the diagram F is better than diagram E.
F			

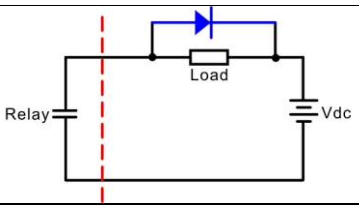
			

fig. 5.3.1 Protection of output transient

The above list and example schematics show the possible solutions to protect the relay contact, it can also apply to NMOS or PMOS as they can only switch the DC working voltage.

5.4 Inrush current consideration

Inrush current (input surge current or switch-on surge) is the maximum, instantaneous input current drawn by an electrical device when first turned on.

The wiring board provides interface devices to drive the target under control device but the type of load, its inrush current and operating frequency are key factors to the contact life. You must take the inrush current into consideration to keep the interface in adequate working condition and life.
The following table gives you the design hints for your reference. Please note that a more conservative design will keep the drive circuit in a safer operating environment.

Type of load	Inrush current
Resistive load	Same as steady state current
Solenoid load	10-20 times of steady state current
Motor load	5-10 times of steady state current
Incandescent lamp load	10-15 times of steady state current
Mercury lamp load	~3 times of steady state current
Sodium vapor lamp load	1-3 times of steady state current
Capacitive load	20-40 times of steady state current
Transformer load	5-15 times of steady state current

5.5 Input debounce

Debounce is the function to filter out 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 unsignificant 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 DIO8265 has built-in digital debounce function by hardware, you can choose the debounce frequency from 50Hz, 100Hz, 200Hz, 1KHz up to 10KHz and more higher, if you need faster input frequency, you can program it no debounce (only limit by the photo-isolator response time).

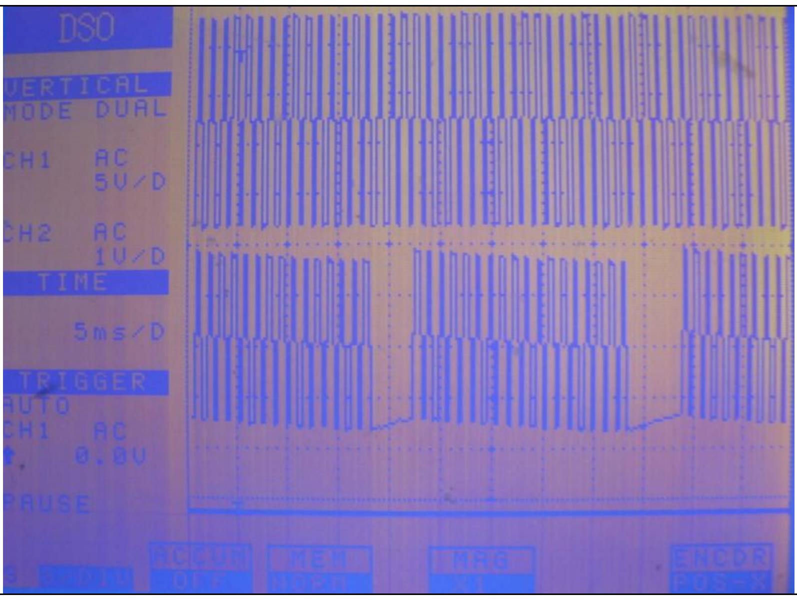
 DSO VERTICAL MODE DUAL CH1 AC 5V/D CH2 AC 1V/D TIME 5ms/D TRIGGER AUTO CH1 AC 0.0V PAUSE ACCUM MEM MAG ENCDR	<- Input frequency at 835Hz <- Output of digital filter, Please note the beat frequency.
Digital debounce circuit work at 1KHZ sample rate and observe the output of filter from 835Hz input	

fig. 5.5.1 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.

5.6 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. DIO8265 provides IN07 ~IN00 isolated input and TTL IO07~IO00 as interrupt input.

5.7 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.**

6. Function format and language difference

6.1 Function format

Every DIO8265 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 DIO8265 function is the parameter **CardID** which is located the driver of DIO8265 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**)

6.2 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

6.3 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 DIO8265 API. Read the following sections that apply to your programming language.

Note: Be sure to include the declaration functions of DIO8265 prototypes by including the appropriate DIO8265 header file in your source code. Refer to Building Applications with the DIO8265 Software Library for the header file appropriate to your compiler.

6.3.1 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 = DIO8265_outport_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 = DIO8265_outport_read(CardID, port, &data);

6.3.2 Visual basic

The file DIO8265.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 DIO8265.bas, do not use the numerical values.

In Visual Basic, you can add the entire DIO8265.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 DIO8265.bas file for your project in Visual Basic 4.0, go to the **File** menu and select the **Add File... option**. Select DIO8265.bas, which is browsed in the DIO8265 \ API directory. Then, select **Open** to add the file to the project.

To add the DIO8265.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** DIO8265.bas, which is in the DIO8265 \ API directory. Then, select **Open** to add the file to the project.

6.3.3 Vb.net

The file DIO8265.vb 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 DIO8265.vb, do not use the numerical values.

In x32 operation system Vb.net , you can add the entire DIO8265.vb 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 DIO8265.vb file for your project in Vb.net, go to the **Project** menu and select the **Add Existing item... option**. Select DIO8265.vb, which is browsed in the DIO8265\API directory. Then, select **Open** to add the file to the project.

In x64 operation system, the header file is under DIO8265\API\x64 directory.

6.3.4 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 DIO8265BC.lib DIO8265.dll

Then add the **DIO8265BC.lib** to your project and add

#include "DIO8265.h" to main program.

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

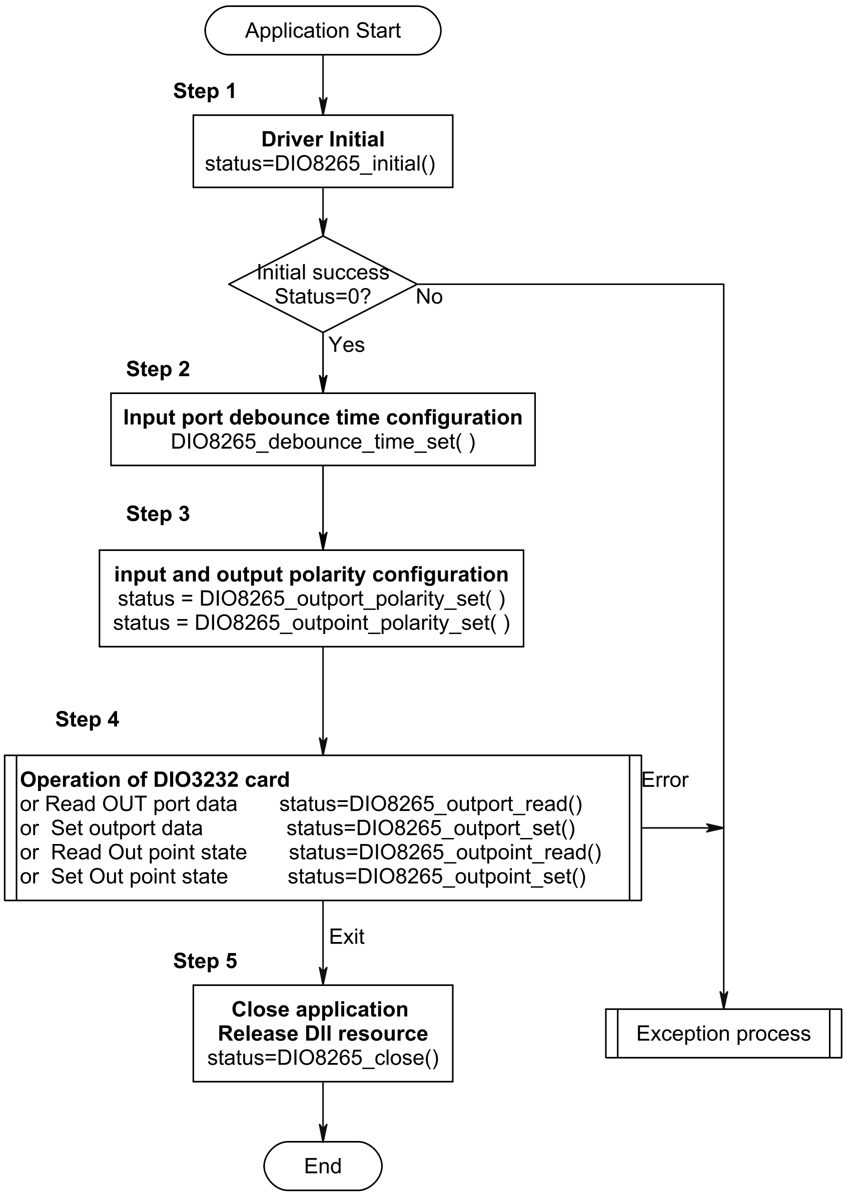
Status = DIO8265_outport_read(CardID, port, data);

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

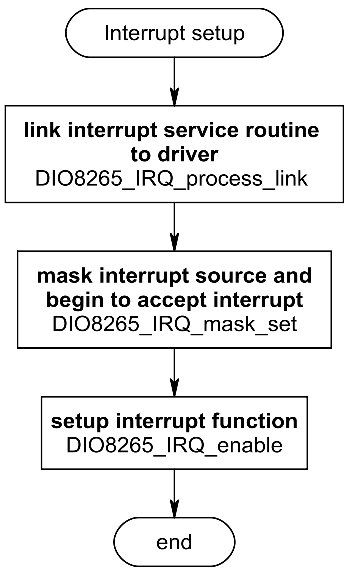
u16 CardID, port;
u8 data;
u32 Status;
Status = DIO8265_outport_read(CardID, port, &data);

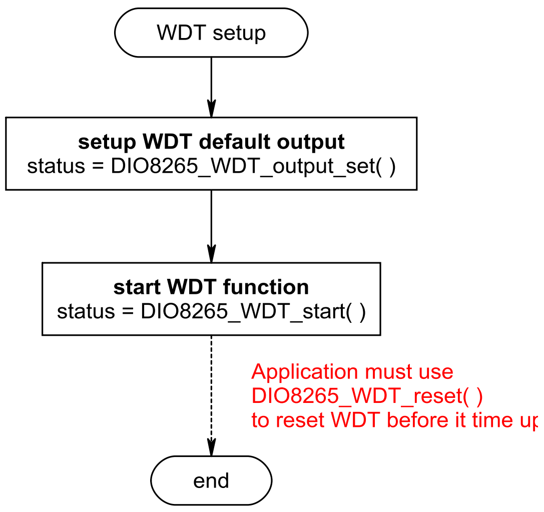
7. Flow chart of application implementation

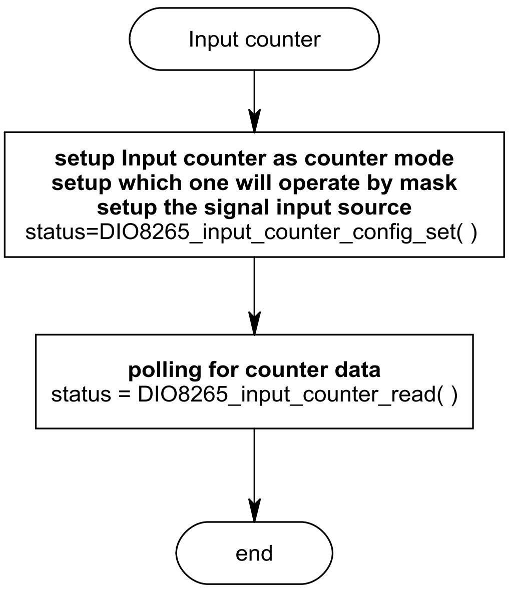
7.1 DIO8265 Flow chart of application implementation

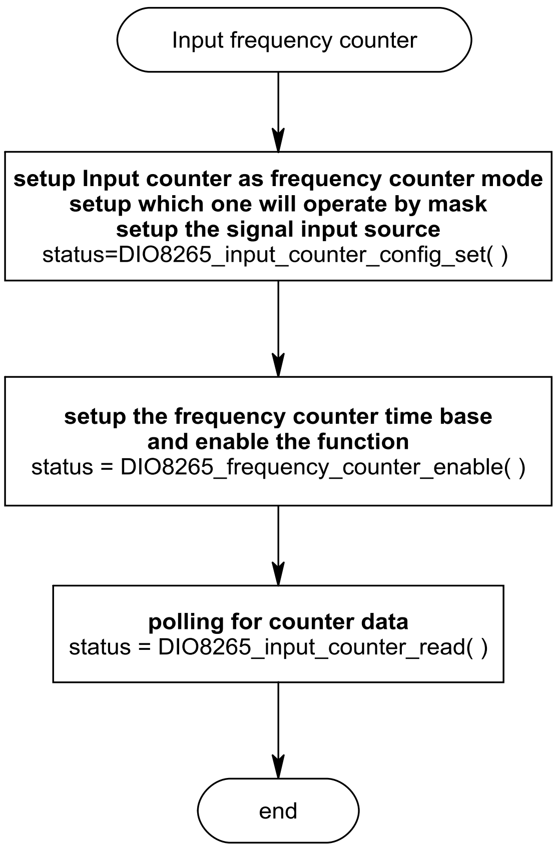


Application Start
Step 1
Driver Initial
status=DIO8265_initial()
Initial success
Status=0?
No
Yes
Step 2
Input port debounce time configuration
DIO8265_debounce_time_set( )
Step 3
input and output polarity configuration
status = DIO8265_outport_polarity_set( )
status = DIO8265_outpoint_polarity_set( )
Step 4
Operation of DIO3232 card
or Read OUT port data status=DIO8265_outport_read()
or Set outport data status=DIO8265_outport_set()
or Read Out point state status=DIO8265_outpoint_read()
or Set Out point state status=DIO8265_outpoint_set()
Error
Exit
Step 5
Close application
Release Dll resource
status=DIO8265_close()
Exception process
End






Input counter
setup Input counter as counter mode
setup which one will operate by mask
setup the signal input source
status=DIO8265_input_counter_config_set( )
polling for counter data
status = DIO8265_input_counter_read( )
end


Input frequency counter
setup Input counter as frequency counter mode
setup which one will operate by mask
setup the signal input source
status=DIO8265_input_counter_config_set( )
setup the frequency counter time base and enable the function
status = DIO8265_frequency_counter_enable( )
polling for counter data
status = DIO8265_input_counter_read( )
end

Note: The frequency timer will generate interrupt at its time base, if you enable the interrupt.
The alternative way to get the frequency is by IRQ service routine.

8. Software overview and dll function

8.1 Initialization

You need to initialize each time you run your application.

DIO8265_initial( ) to initial the resources of the driver.

DIO8265_close( ) to close the resources of the driver before you close your application.

DIO8265_info( ) get the information of address assigned by the OS.

To check the firmware version,

DIO8265_firmware_version_read( ) will do.

- **DIO8265_initial**

Format: **u32 status =DIO8265_initial (void)**

Purpose: Initial the DIO8265 resource when start the Windows applications.

- **DIO8265_close**

Format: **u32 status =DIO8265_close (void)**

Purpose: Release the DIO8265 resource when close the Windows applications.

- **DIO8265_info**

Format: **u32 status =DIO8265_info(u8 CardID, u32 *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	u32	physical I/O address assigned by OS

- **DIO8265_firmware_version_read**

Format: **u32 status =DIO8265_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] example： version: 3.50 Version[1] == 3 Version[0] == 50

8.2 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. The isolated digital input normally limited by the response time of photo-coupler,

To match the logic polarity of your software, DIO8265 also provides the input output polarity configuration; use

DIO8265_outport_polarity_set( ) to configure the polarity of each output of port,

DIO8265_outport_polarity_read( ) to read back the polarity of each output of port.

For the bitwise polarity setting or read back, use

DIO8265_outpoint_polarity_set( ) to configure the polarity of output;

DIO8265_outpoint_polarity_read( ) to read back the polarity of output.

For the port input, output, use:

DIO8265_outport_set( ) to output byte data to output port,

DIO8265_outport_read( ) to read a byte outport data from I/O port,

For bitwise control, use

DIO8265_outpoint_set( ) to set output bit,

DIO8265_outpoint_read( ) to read outpoint bit,

- **DIO8265_outport_polarity_set**

Format: **u32 status = DIO8265_outport_polarity_set(u8 CardID, u8 port , u8 polarity)**

Purpose: Sets the outport polarity.

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY switch(**0x0-0xF**)
port	u8	port number 0: Port 0 1: Port 1 2: Port 2 3: Port 3 4: Port 4 5: Port 5 6: Port 6 7: Port 7
polarity	u8	bitmap of polarity values 7: OUTx7 or INx7 ... 0: OUTx0 or INx0 bit data =0, normal polarity (default) bit data =1, invert polarity

- **DIO8265_outport_polarity_read**

Format: **u32 status = DIO8265_outport_polarity_read(u8 CardID , u8 port , u8 *polarity)**

Purpose: Read the outport polarity.

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY switch(**0x0-0xF**)
port	u8	port number 0: Port 0 1: Port 1 2: Port 2 3: Port 3 4: Port 4 5: Port 5 6: Port 6 7: Port 7

Output:

Name	Type	Description
polarity	u8	bitmap of polarity values 7: OUTx7 or INx7 ... 0: OUTx0 or INx0 bit data =0, normal polarity (default) bit data =1, invert polarity

- **DIO8265_outpoint_polarity_set**

Format: **u32 status = DIO8265_outpoint_polarity_set(u8 CardID, u8 port , u8 point , u8 state)**

Purpose: Sets the output point polarity.

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY switch(**0x0-0xF**)
port	u8	port number 0: Port 0 1: Port 1 2: Port 2 3: Port 3 4: Port 4 5: Port 5 6: Port 6 7: Port 7
point	u8	point number 0~7 7: OUTx7 or INx7 ... 0: OUTx0 or INx0
state	u8	0, normal polarity (default) 1, invert polarity

- **DIO8265_outpoint_polarity_read**

Format: **u32 status = DIO8265_outpoint_polarity_read(u8 CardID , u8 port , u8 point , u8 *state)**

Purpose: Read the output point polarity.

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY switch(**0x0-0xF**)
port	u8	port number 0: Port 0 1: Port 1
point	u8	point number 0~7 7: OUTx7 or INx7 ... 0: OUTx0 or INx0

Output:

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

DIO8265_outport_set

Format: **u32 status = DIO8265_outport_set(u8 CardID, u8 port , u8 data)**

Purpose: Set the output port data.

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY switch(**0x0-0xF**)
port	u8	port number 0: Port 0 1: Port 1 2: Port 2 3: Port 3 4: Port 4 5: Port 5 6: Port 6 7: Port 7
data	u8	output values: b7~b0 for OUTx7~OUTx0

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

DIO8265_outport_read

Format: **u32 status =DIO8265_outport_read(u8 CardID, u8 port, u8 *data)**

Purpose: Read back the output port register data.

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY switch(**0x0-0xF**)
port	u8	port number 0: Port 0 1: Port 1 2: Port 2 3: Port 3 4: Port 4 5: Port 5 6: Port 6 7: Port 7

Output:

Name	Type	Description
data	u8	values: b7~b0 for OUTx7~OUTx0

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

- **DIO8265_outpoint_set**

Format: **u32 status = DIO8265_outpoint_set(u8 CardID, u8 port , u8 point, u8 state)**

Purpose: Set the output point.

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY switch(**0x0-0xF**)
port	u8	port number 0: Port 0 1: Port 1 2: Port 2 3: Port 3 4: Port 4 5: Port 5 6: Port 6 7: Port 7
point	u8	point number OUTx7~OUTx0
state	u8	state of output point 0: inactive 1: active

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

- **DIO8265_outpoint_read**

Format: **u32 status =DIO8265_outpoint_read(u8 CardID, u8 port , u8 point , u8 *state)**

Purpose: Read back the output point register data.

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY switch(**0x0-0xF**)
port	u8	port number 0: Port 0 1: Port 1 2: Port 2 3: Port 3 4: Port 4 5: Port 5 6: Port 6 7: Port 7
point	u8	point number 7~0 for OUTx7~OUTx0

Output:

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

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

8.3 TTL I/O Port R/W

DIO8265 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 JP0 and JP1(refer DIO8265 user's manual).

To configure the port as input or output by:

DIO8265_TTL_IO_config_set( ) and read back the configuration by:

DIO8265_TTL_IO_config_read( ).

To change the polarity as you need by:

DIO8265_TTL_IO_port_polarity_set( ) and read back to verify by:

DIO8265_TTL_IO_port_polarity_read( ).

For the bitwise polarity set and read, use:

DIO8265_TTL_IO_point_polarity_set( ) and read back to verify by:

DIO8265_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 15 ranges: 50Hz, 100Hz, 200KHz ... up to 8MHz and no debounce to select for your application, use:

DIO8265_TTL_IO_debounce_time_set( ) to set the adequate time constant to drop out the noise and read back to check the setting by:

DIO8265_TTL_IO_debounce_time_read( ).

After system reset, the TTL port is hardware disabled, the JP0, JP1 jumper preset its state to high or low. **Before you begin to operate the input output function, you must enable it.**

DIO8265_TTL_IO_enable( ), you can also disable it to keep the TTL port at its default setting state(by JP0,JP1 setting):

DIO8265_TTL_IO_disable( )

The TTL I/O port can use:

DIO8265_TTL_IO_port_set( ) to output data and input data by:

DIO8265_TTL_IO_port_read( ).

For the bitwise point output, use:

DIO8265_TTL_IO_point_set( ) and point input by:

DIO8265_TTL_IO_point_read( ).

The TTL input points (IO07~IO00) provide interrupt function to have fast response of input transition. Please refer the 8.8 Interrupt function section for detail. It can also be used as counter input to the 16bit counter for speed/pulse counting. Refer 8.6 Input Counter for detail.

- **DIO8265_TTL_IO_config_set**

Format: **u32 status =DIO8265_TTL_IO_config_set (u8 CardID, u8 port, u8 config)**

Purpose: Set TTL 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 1: input port (default)

- **DIO8265_TTL_IO_config_read**

Format: **u32 status =DIO8265_TTL_IO_config_read (u8 CardID, u8 port, u8 *config ,**
u8 *control)

Purpose: read TTL port configure 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 1: input port (default)
control	u8	0: Disable 1: Enable

- **DIO8265_TTL_IO_port_polarity_set**

Format: **u32 status =DIO8265_TTL_IO_port_polarity_set (u8 CardID, u8 port, u8 polarity)**

Purpose: Sets the TTL 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 value. take port1 as example: bit7: IO17 ... bit0: IO10 bit data =0, normal polarity bit data =1, invert polarity

- **DIO8265_TTL_IO_port_polarity_read**

Format: **u32 status = DIO8265_TTL_IO_port_polarity_read (u8 CardID, u8 port, u8 * polarity)**

Purpose: Read the TTL 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 value. take port0 as example: bit7: IO07 ... bit0: IO00 bit data =0, normal polarity bit data =1, invert polarity

● **DIO8265_TTL_IO_point_polarity_set**

Format: u32 status =DIO8265_TTL_IO_point_polarity_set (u8 CardID, u8 port, u8 point, u8 state)

Purpose: Sets the TTL 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 7~0 take port0 as example: 7: IO07 ... 0: IO00
state	u8	polarity value. bit data =0, normal polarity bit data =1, invert polarity

● **DIO8265_TTL_IO_point_polarity_read**

Format: u32 status = DIO8265_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 7~0 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

- **DIO8265_TTL_IO_debounce_time_set**

Format: **u32 status = DIO8265_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: 50 Hz 2: 100 Hz 3: 200 Hz 4: 1kHz 5: 2KHz 6: 10KHz 7: 20KHz 8: 50KHz 9: 100KHz (default) 10: 200KHz 11: 500KHz 12: 1MHz 13: 2MHz 14: 4MHz 15: 8MHz

Note: only valid for TTL port configured as input

- **DIO8265_TTL_IO_debounce_time_read**

Format: **u32 status = DIO8265_TTL_IO_debounce_time_read (u8 CardID,u8 port , u8 *debounce_time)**

Purpose: To read back configuration of TTL 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: 50 Hz 2: 100 Hz 3: 200 Hz 4: 1kHz 5: 2KHz 6: 10KHz 7: 20KHz 8: 50KHz 9: 100KHz (default) 10: 200KHz 11: 500KHz 12: 1MHz 13: 2MHz 14: 4MHz 15: 8MHz

- **DIO8265_TTL_IO_enable**

Format: **u32 status =DIO8265_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

- **DIO8265_TTL_IO_disable**

Format: **u32 status =DIO8265_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

- **DIO8265_TTL_IO_port_set**

Format: **u32 status = DIO8265_TTL_IO_port_set (u8 CardID,u8 port, u8 data)**

Purpose: Sets the TTL 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.

● DIO8265_TTL_IO_port_read

Format: **u32 status = DIO8265_TTL_IO_port_read (u8 CardID , u8 port , u8 *data)**

Purpose: Read the TTL 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.

● DIO8265_TTL_IO_point_set

Format: **u32 status =DIO8265_TTL_IO_point_set (u8 CardID, u8 port , u8 point, u8 state)**

Purpose: Sets the bit data of TTL 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.

- **DIO8265_TTL_IO_point_read**

Format: **u32 status =DIO8265_TTL_IO_point_read (u8 CardID, u8 port , u8 point, u8 *state)**

Purpose: Read the TTL 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.

8.4 Timer function

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

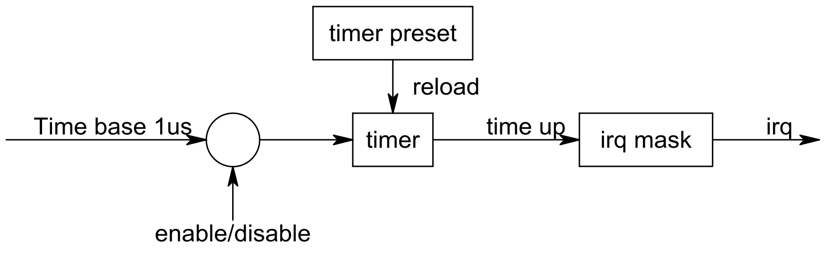


fig. 8.4.1 timer function model

You can set the timer constant by

DIO8265_timer_set( ) and use

DIO8265_timer_start( ) to star its operation,

DIO8265_timer_stop( ) to stop operation.

For the timer related registers use:

DIO8265_timer_read( ) to read back registers.

The timer can also trigger interrupt at timer up, please refer to 8.8 Interrupt function for detail.

- **DIO8265_timer_set**

Format: **u32 status = DIO8265_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 1~ 4294967295

Note:

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

- **DIO8265_timer_start**

Format: **u32 status = DIO8265_timer_start (u8 CardID)**

Purpose: start timer function.

Parameters:

Input:

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

- **DIO8265_timer_stop**

Format: **u32 status = DIO8265_timer_stop (u8 CardID)**

Purpose: stop timer function.

Parameters:

Input:

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

- **DIO8265_timer_read**

Format: **u32 status= DIO8265_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

8.5 WDT (Watch Dog Timer)

In the industrial application, computer abnormal function can be improved by many treatments but the last and most often method is the watch-dog timer. A watch-dog timer is a timer that counts the time at preset value, the user's program or application must reset it before the time up. In normal condition, the user's program or application will not fail to reset it but while it is abnormal, rest watch dog timer will fail. On this special occasion, the system must take some special action to prevent further disaster. Generally a predefined output by hardware is a good choice. The function block of watch dog timer shown as follows:

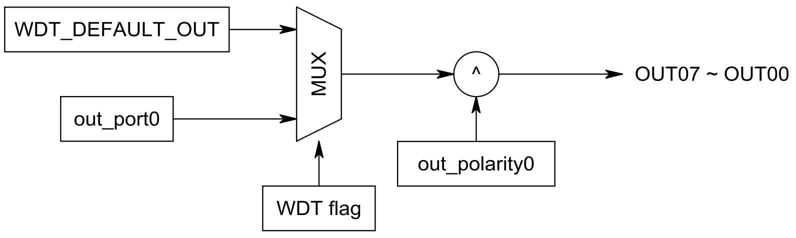


fig 8.5.1 watch dog timer

To setup the hardware forced output while user's program or application fail to reset WDT (watch dog timer), using:

DIO8265_WDT_output_set( ) to setup the default output and read back for verification by ***DIO8265_WDT_output_read( )***

To start the monitoring of WDT (watch dog timer),

DIO8265_WDT_start( ) will do. Once you start it, you must reset it before time up by: ***DIO8265_WDT_reset( )***

If you want to quit from the WDT, you must stop it by

DIO8265_WDT_stop( ) and on any time you can check the WDT status by ***DIO8265_WDT_read( )***

DIO8265_WDT_output_set

Format: **u32 status = DIO8265_WDT_output_set(u8 CardID, u8 output)**

Purpose: To set WDT default output on OUT07~OUT00.

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY switch(**0x0-0xF**)
output	u8	WDT default output data on OUT07~OUT00 bit7 : OUT07 … bit0 : OUT00 point of output state 0: inactive 1: active

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

DIO8265_WDT_output_read

Format: **u32 status = DIO8265_WDT_output_read(u8 CardID,u8 *output)**

Purpose: To read back WDT output on OUT07~OUT00.

Parameters:

Input:

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

Output:

Name	Type	Description
output	u8	WDT default output data on OUT07~OUT00 bit7 : OUT07 … bit0 : OUT00 point of output state 0: inactive 1: active

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

- **DIO8265_WDT_start**

Format: **u32 status = DIO8265_WDT_start(u8 CardID, u16 time_constant, u8 mode)**

Purpose: To start WDT function.

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY switch(**0x0-0xF**)
time_constant	u16	time constant of WDT timer at 1ms time base, the time constant is recommend **no less than 150ms**
mode	u8	0: auto mode, user no need to reset WDT, the driver will auto reset WDT on every 0.5*WDT_time_constant 1: manual mode, user must reset WDT before its time up

- **DIO8265_WDT_reset**

Format: **u32 status = DIO8265_WDT_reset(u8 CardID)**

Purpose: To reset WDT timer, used for manual mode.

Parameters:

Input:

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

- **DIO8265_WDT_stop**

Format: **u32 status = DIO8265_WDT_stop(u8 CardID)**

Purpose: To stop WDT function.

Parameters:

Input:

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

- **DIO8265_WDT_read**

Format: **u32 status = DIO8265_WDT_read(u8 CardID, u8 index, u16 *data)**

Purpose: To read back WDT related registers.

Parameters:

Input:

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

Output:

Name	Type	Description

data	u16	if index=0,
		data=0, WDT stops
		data=1, WDT run
		if index=1,
		data=1~65535,
		the preset WDT time constant
		if index=2,
		data=0~65535, the WDT time on the fly

8.6 Input Counter

The DIO8265 TTL IO07~IO00 inputs can work as counter input to 16bit COUNTER7 ~ COUNTER0. The counter model is shown as follows:

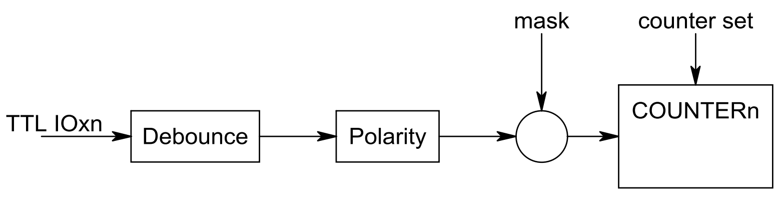


fig. 8.6.1 counter function model

From the model, you can see the input polarity and debounce block, they are the same as general purpose input (Refer ***DIO8265_port_polarity_set( ), DIO8265_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.

Owing to TTL IO07~IO00 inputs can work as counter input, it can also work as input of frequency counter (to count the input signal frequency). You must configure what function you need before put it into function.

DIO8265_input_counter_config_set( ) to setup function as input counter or input frequency counter ***DIO8265_input_counter_config_read( )*** to read back the configuration.

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

DIO8265_input_counter_all_set( ) to set values to all counters (set 0 value functions as counter reset) or read all counters by:

DIO8265_input_counter_all_read( ) or set single counter's value by:

DIO8265_input_counter_set( ) and read back single counter's value by:

DIO8265_input_counter_read( )

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

DIO8265_input_counter_mask_set( ) and read back by

DIO8265_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:

DIO8265_input_counter_control_set( ) and read back to check the status by:

DIO8265_input_counter_control_read( )

● DIO8265_input_counter_config_set

Format: **u32 status = DIO8265_input_counter_config_set(u8 CardID, u8 mask, u8 source, u8 mode)**

Purpose: set the inport or TTL/IO to in_counter or frequency_counter

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)
source	u8	null
mode	u8	bit7~bit0 bit7 for counter7 0: in_counter 1: frequency_counter ... bit0 for counter0 0: in_counter 1: frequency_counter

*The signal source only TTL port0 at input mode, no need to program.

● DIO8265_input_counter_config_read

Format: **u32 status = DIO8265_input_counter_config_read(u8 CardID, u8 *mask, u8 *source, u8 *mode)**

Purpose: read the inport or TTL/IO is in_counter or frequency_counter

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)
source	u8	null
mode	u8	bit7~bit0 bit7 for counter7 0: in_counter 1: frequency_counter ... bit0 for counter0 0: in_counter 1: frequency_counter

● DIO8265_input_counter_all_set

Format: **u32 status = DIO8265_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.

- **DIO8265_input_counter_all_read**

Format: **u32 status = DIO8265_input_counter_all_read(u8 CardID,u16 data[8])**

Purpose: To read input counters' data on the fly.

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

- **DIO8265_input_counter_set**

Format: **u32 status = DIO8265_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.

- **DIO8265_input_counter_read**

Format: **u32 status = DIO8265_input_counter_read(u8 CardID,u8 index, u16 *data)**

Purpose: To read input counter's datum on the fly.

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

- **DIO8265_input_counter_mask_set**

Format: **u32 status = DIO8265_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.

- **DIO8265_input_counter_mask_read**

Format: **u32 status = DIO8265_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)

- **DIO8265_input_counter_control_set**

Format: **u32 status = DIO8265_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 1: enable, all counter clear to 0 before counters response to input trigger (if no mask)

- **DIO8265_input_counter_control_read**

Format: **u32 status = DIO8265_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 response to input trigger (if no mask)

8.7 Frequency counter

If the input counter work with the timer, you can count the frequency on the base of timer time constant. (Note: The frequency timer use another timer, not the timer mentioned on 8.4 Timer function)

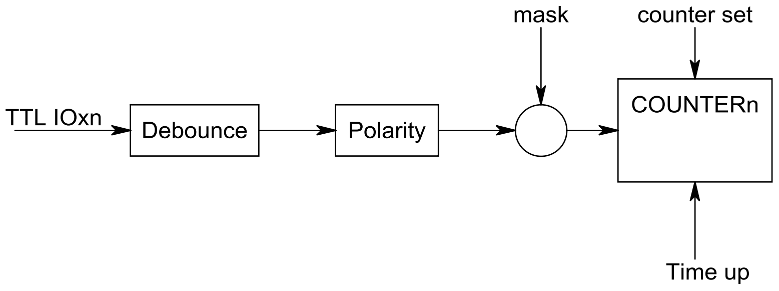


fig. 8.7.1 Frequency counter model

You can configure the input counter as pure counter function or work with the timer as frequency counter, both functions can choose the signal source from TTL IO07~IO00 , mask off the unused channels by

DIO8265_input_counter_config_set( ) and read back for verification by

DIO8265_input_counter_config_read( ) (refer 8.6 Input Counter)

To start the frequency counter operation, you need to enable the function. It will control the on-board timer and set the time constant as you specified.

DIO8265_frequency_counter_enable( ), to stop the frequency counter by:

DIO8265_frequency_counter_disable( )

To crop the frequency data, you just

DIO8265_input_counter_all_read( ) or

DIO8265_input_counter_read( ) to read the counter and the input signal frequency will be

frequency = counter data / timer time constant

Owing to the time base clock is generate from PCIe bus clock, the timing accuracy is sometimes not meet your requirement, you can use an accurate frequency counter to count the test output of the timer by:

DIO8265_frequency_counter_test_enable( ) the TTL IO10 will generate a square wave at 50-50 duty and the duty is the time constant you program. Using the output, you can measure with a high accuracy oscilloscope or meter to trimming the time constant.

- **DIO8265_frequency_counter_enable**

Format: **u32 status = DIO8265_frequency_counter_enable(u8 CardID, u32 timer)**

Purpose: enable frequency counter

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY switch(**0x0-0xF**)
timer	u32	set the timer time constant (1~4294967295), on 1us time base for the input counter. Say, set timer=10000, you will have the counter data on 10ms time base then the frequency per second will be: counter data *100

- **DIO8265_frequency_counter_disable**

Format: **u32 status = DIO8265_frequency_counter_disable(u8 CardID)**

Purpose: disable frequency counter

Parameters:

Input:

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

- **DIO8265_frequency_counter_test_enable**

Format: **u32 status =DIO8265_frequency_counter_test_enable(u8 CardID,u8 enable)**

Purpose: the TTL IO11 will have a toggled signal output of time base when frequency counter enabled

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY switch(**0x0-0xF**)
enable	u8	0: disable test out 1: enable test out TTL IO11

Note: At the test mode, the TTL port1 will automatically set to output mode and the timer time up will toggles the IO11. You can use a scope or frequency counter to measure the time and fine adjust the time constant. The TTL port1 will return to its original setting (as input or output) while the test is disabled.

8.8 Interrupt function

The DIO8265 card provides inputs IO07~IO00 and timers as interrupt sources. The interrupt will trigger the system to get a quick service. To use the external interrupt (IRQ) function you must link the service routine by:

DIO8265_IRQ_process_link( ) then setup the IRQ mask for the interrupt by:

DIO8265_IRQ_mask_set( ) to select IN07~IN00 or timers as source of IRQ.

DIO8265_IRQ_mask_read( )

After setup, you can enable the IRQ by:

DIO8265_IRQ_enable( ) and also you can disable IRQ by:

DIO8265_IRQ_disable( )

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

DIO8265_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.

● **DIO8265_IRQ_process_link**

Format: **u32 status = DIO8265_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

● **DIO8265_IRQ_mask_set**

Format: **u32 status = DIO8265_IRQ_mask_set(u8 CardID, u8 index , u8 Data)**

Purpose: Mask interrupt from timers, IN07~IN00 or IO07~IO00.

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY switch(**0x0-0xF**)
index	u8	1: TTL port 0 2: TC
Data	u8	index = 1 then bit 0: irq source fromTTL IO00 bit 1: irq source from TTL IO01 bit 2: irq source from TTL IO02 bit 3: irq source from TTL IO03 bit 4: irq source from TTL IO04 bit 5: irq source from TTL IO05 bit 6: irq source from TTL IO06 bit 7: irq source from TTL IO07 ElseIf index = 2 then bit 0: irq source from Timer bit 1: irq source from timer of frequency counter If corresponding bit =0, mask off the interrupt

- **DIO8265_IRQ_mask_read**

Format: **u32 status = DIO8265_IRQ_mask_read(u8 CardID, u8 index , u8 *Data)**

Purpose: Read Mask interrupt from timer, IN07~IN00 or IO07~IO00.

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY switch(**0x0-0xF**)
index	u8	1: TTL port 0 2: TC

Output:

Name	Type	Description
Data	u8	If index = 1 then bit 0: irq source fromTTL IO00 bit 1: irq source from TTL IO01 bit 2: irq source from TTL IO02 bit 3: irq source from TTL IO03 bit 4: irq source from TTL IO04 bit 5: irq source from TTL IO05 bit 6: irq source from TTL IO06 bit 7: irq source from TTL IO07 ElseIf index = 2 then bit 0: irq source from Timer bit 1: irq source from timer of frequency counter If corresponding bit =0, mask off the interrupt

- **DIO8265_IRQ_enable**

Format: **u32 status = DIO8265_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

● DIO8265_IRQ_disable

Format: **u32 status = DIO8265_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**)

● DIO8265_IRQ_status_read

Format: **u32 status = DIO8265_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: null bit01: null bit02: null bit03: null bit04: null bit05: null bit06: null bit07: null bit08: 1, irq source from TTL IO00 bit09: 1, irq source from TTL IO01 bit10: 1, irq source from TTL IO02 bit11: 1, irq source from TTL IO03 bit12: 1, irq source from TTL IO04 bit13: 1, irq source from TTL IO05 bit14: 1, irq source from TTL IO06 bit15: 1, irq source from TTL IO07 bit16: 1, irq source from Timer bit17: 1, irq source from timer of frequency counter

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.

9. Dll list

	Function Name	Description
1.	DIO8265_initial( )	DIO8265 initial
2.	DIO8265_close( )	DIO8265 close
3.	DIO8265_info( )	get OS. assigned address
4.	DIO8265_firmware_version_read ( )	Read device firmware version
5.	DIO8265_outport_polarity_set( )	setup outport polarity
6.	DIO8265_outport_polarity_read	read back outport polarity
7.	DIO8265_outpoint_polarity_set( )	setup outpoint polarity
8.	DIO8265_outpoint_polarity_read( )	read back outpoint polarity
9.	DIO8265_outport_set( )	set output port (byte)
10.	DIO8265_outport_read( )	read outport data (byte)
11.	DIO8265_outpoint_set( )	set output point state (bit)
12.	DIO8265_outpoint_read( )	read output point state (bit)
13.	DIO8265_TTL_IO_config_set( )	setup TTL port I/O configuration
14.	DIO8265_TTL_IO_config_read( )	read back TTL port I/O configuration
15.	DIO8265_TTL_IO_port_polarity_set( )	setup TTL port polarity
16.	DIO8265_TTL_IO_port_polarity_read( )	read back TTL port polarity
17.	DIO8265_TTL_IO_point_polarity_set( )	setup TTL point polarity
18.	DIO8265_TTL_IO_point_polarity_read( )	read back TTL point polarity
19.	DIO8265_TTL_IO_debounce_time_set( )	setup TTL port input debounce time
20.	DIO8265_TTL_IO_debounce_time_read( )	read back TTL port input debounce time
21.	DIO8265_TTL_IO_enable( )	enable TTL IO function
22.	DIO8265_TTL_IO_disable( )	disable TTL IO function
23.	DIO8265_TTL_IO_port_set( )	set TTL IO port data
24.	DIO8265_TTL_IO_port_read( )	read TTL IO port data
25.	DIO8265_TTL_IO_point_set( )	set TTL IO point data
26.	DIO8265_TTL_IO_point_read( )	read TTL IO point data
27.	DIO8265_timer_set( )	set timer time constant
28.	DIO8265_timer_start( )	start timer function
29.	DIO8265_timer_stop( )	stop timer function
30.	DIO8265_timer_read( )	read timer related registers
31.	DIO8265_WDT_output_set( )	set WDT default output
32.	DIO8265_WDT_output_read( )	read WDT default output
33.	DIO8265_WDT_start( )	start WDT function
34.	DIO8265_WDT_reset( )	
35.	DIO8265_WDT_stop( )	stop WDT function
36.	DIO8265_WDT_read( )	read WDT related registers
37.	DIO8265_input_counter_config_set( )	setup the frequency counter
38.	DIO8265_input_counter_config_read( )	read back the setup of frequency counter
39.	DIO8265_input_counter_all_set( )	set all input counters' data
40.	DIO8265_input_counter_all_read( )	read all input counters' data
41.	DIO8265_input_counter_set( )	set one input counters' datum
42.	DIO8265_input_counter_read( )	read one input counters' datum
43.	DIO8265_input_counter_mask_set( )	set input counters' operation mask
44.	DIO8265_input_counter_mask_read( )	read back input counters' operation mask
45.	DIO8265_input_counter_control_set( )	set input counter control

46.	DIO8265_input_counter_control_read( )	read back input counter control status
47.	DIO8265_frequency_counter_enable( )	enable operation of frequency counter
48.	DIO8265_frequency_counter_disable( )	disable operation of frequency counter
49.	DIO8265_frequency_counter_test_enable( )	check the time base accuracy
50.	DIO8265_IRQ_process_link( )	link interrupt service routine to driver
51.	DIO8265_IRQ_mask_set( )	set interrupt mask
52.	DIO8265_IRQ_mask_read( )	read interrupt mask
53.	DIO8265_IRQ_enable( )	enable interrupt function
54.	DIO8265_IRQ_disable( )	disable interrupt function
55.	DIO8265_IRQ_status_read( )	read back IRQ status

10. DIO8265 Error codes summary

10.1 DIO8265 Error codes table

Error Code	Symbolic Name	Description
0	JSDRV_NO_ERROR	No error.
2	JSDRV_INIT_ERROR	Driver initial error
10	CARD_ERROR	Card ver
100	DEVICE_RW_ERROR	Device Read/Write error
101	JSDRV_NO_CARD	No DIO8265 card on the system.
102	JSDRV_DUPLICATE_ID	DIO8265 CardID duplicate error.
300	JSDIO_ID_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