DIO8217
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 DIO8217

1.1 Install the PCI 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 later 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 (..\DIO8217\Software\WinXP_7\ or if you download from website please execute the file DIO8217_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 \DIO8217\API (header files and lib files for VB,VC,BCB,C#)

.. \ JS Automation \DIO8217\Driver (backup copy of DIO8217 drivers)

.. \ JS Automation \DIO8217\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 DIO8217 software

DIO8217 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 DIO8217 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 DIO8217 functions within Windows' operation system environment.

3.1 What you need to get started

To set up and use your DIO8217 software, you need the following:

- DIO8217 software
- DIO8217 hardware

  Main board

  Wiring board (Option)

3.2 Software programming choices

You have several options to choose from when you are programming DIO8217 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 DIO8217 software.

4. DIO8217 Language support

The DIO8217 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 DIO8217 software library

The DIO8217 function reference topic contains general information about building DIO8217 applications, describes the nature of the DIO8217 files used in building DIO8217 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 DIO8217 Windows libraries

The DIO8217 for Windows function library is a DLL called **DIO8217.dll**. Since a DLL is used, DIO8217 functions are not linked into the executable files of applications. Only the information about the DIO8217 functions in the DIO8217 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 DIO8217 functions in DIO8217.dll.

Header Files and Import Libraries for Different Development Environments		
Language	**Header File**	**Import Library**
Microsoft Visual C/C++	DIO8217.h	DIO8217VC.lib
Borland C/C++	DIO8217.h	DIO8217BC.lib
Microsoft Visual C#	DIO8217.cs	
Microsoft Visual Basic	DIO8217.bas	
Microsoft VB.net	DIO8217.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.

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.

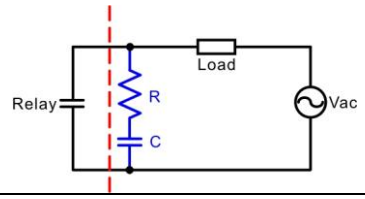
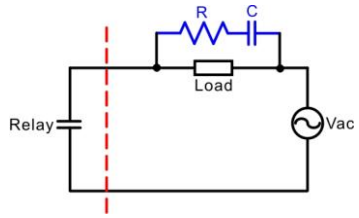
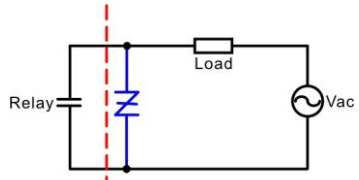
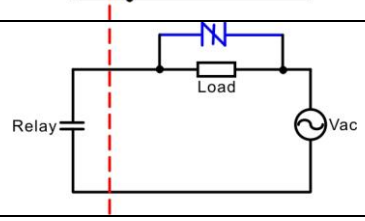
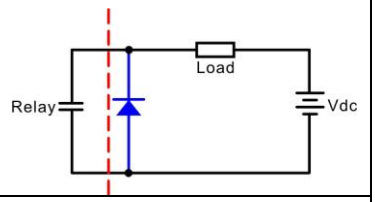
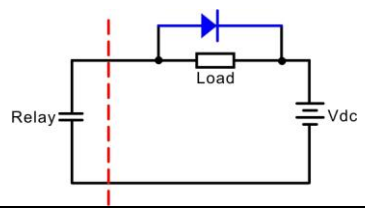
	Protection Connection	Comments	Design Note
A		Leakage current to load maybe cause malfunction. The effective release time will be lengthened.	Energy stored in the coil will dissipate by R and coil resistance. R: 0.5-1 Ohm per 1V contact voltage C: 0.5-1 uF per 1A current Use capacitor voltage of 200-300V 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.

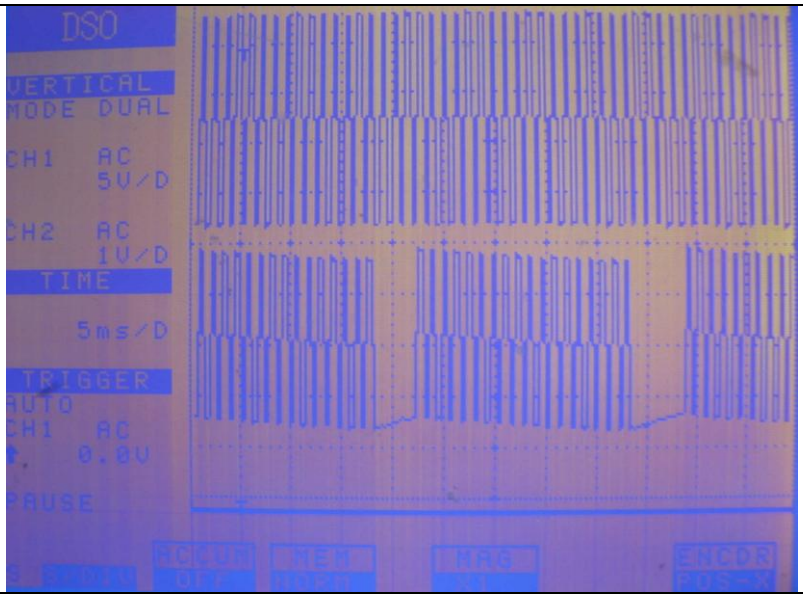
Type of load	Inrush current
Resistive load	Same as steady state current
Solenoid load	10-20 times of steady state current
Motor load	5-10 times of steady state current
Incandescent lamp load	10-15 times of steady state current
Mercury lamp load	~3 times of steady state current
Sodium vapor lamp load	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 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 DIO8217 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).



<- 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 to use a timer as time base 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. DIO8217 provides IN07 ~IN00 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 DIO8217 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 DIO8217 function is the parameter **CardID** which is located the driver of DIO8217 board you want to use those given operation. The **CardID** is assigned by rotary switch. 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 DIO8217 API. Read the following sections that apply to your programming language.

Note: Be sure to include the declaration functions of DIO8217 prototypes by including the appropriate DIO8217 header file in your source code. Refer to Building Applications with the DIO8217 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 = DIO8217_inport_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 = DIO8217_inport_read(CardID, port, &data);

6.3.2 Visual basic

The file DIO8217.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 DIO8217.bas, do not use the numerical values.

In Visual Basic, you can add the entire DIO8217.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 DIO8217.bas file for your project in Visual Basic 4.0, go to the **File** menu and select the **Add File... option**. Select DIO8217.bas, which is browsed in the DIO8217 \ API directory. Then, select **Open** to add the file to the project.

To add the DIO8217.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** DIO8217.bas, which is in the DIO8217 \ API directory. Then, select **Open** to add the file to the project.

6.3.3 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 DIO8217BC.lib DIO8217.dll

Then add the **DIO8217BC.lib** to your project and add

#include "DIO8217.h" to main program.

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

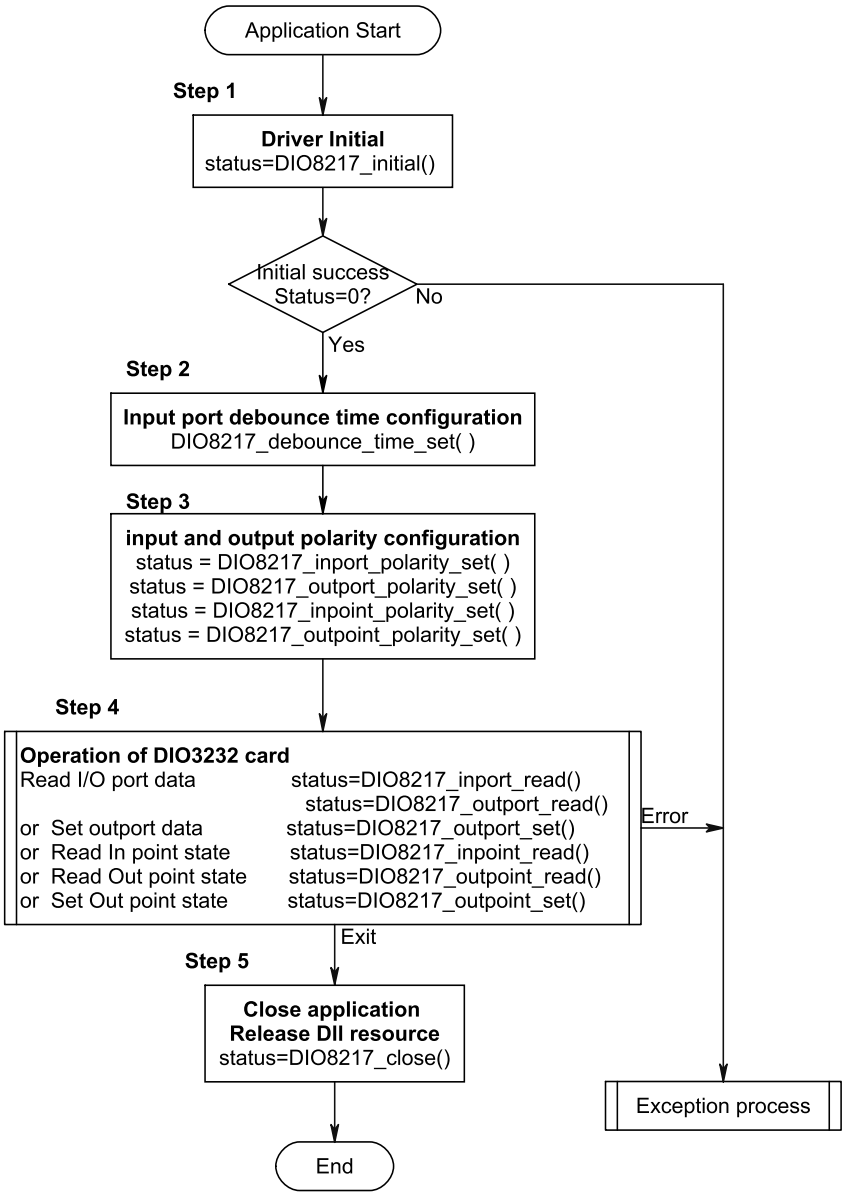
Status = DIO8217_inport_read(CardID, port, *data);

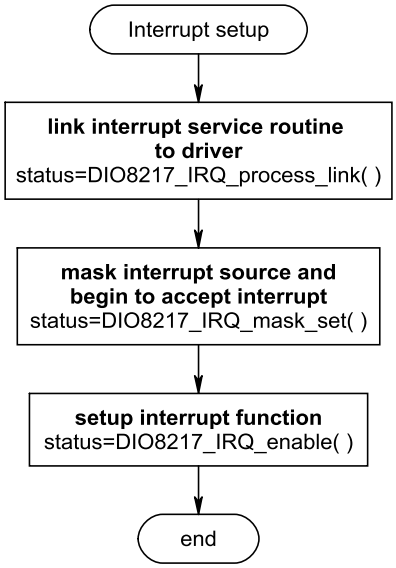
where **CardID** and **port** are input parameters, and **data** is an output parameter. Consider the following example:

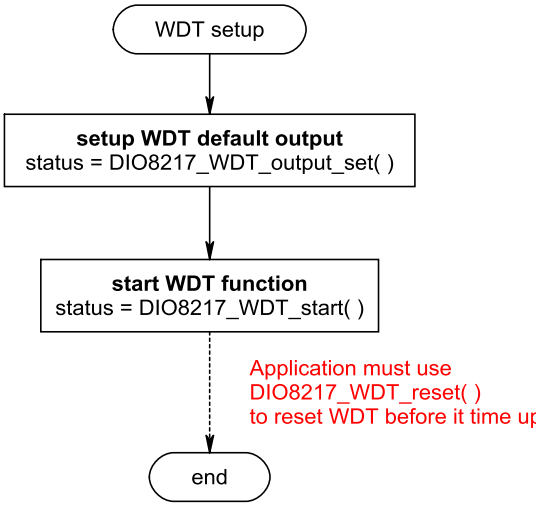
```
u16 CardID, port;
u8 data;
u32 Status;
Status = DIO8217_inport_read(CardID, port, &data);
```

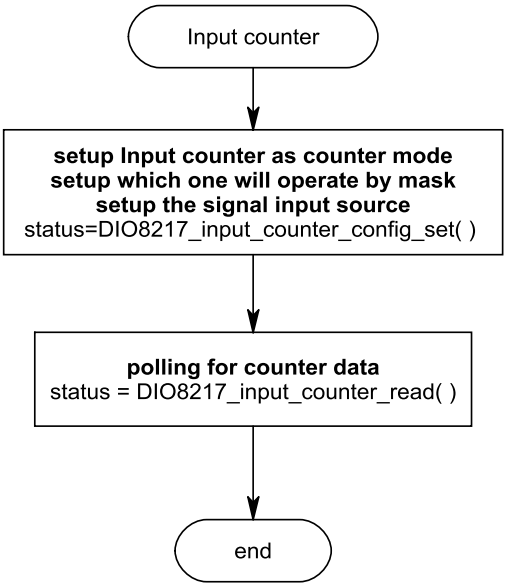
7. Flow chart of application implementation

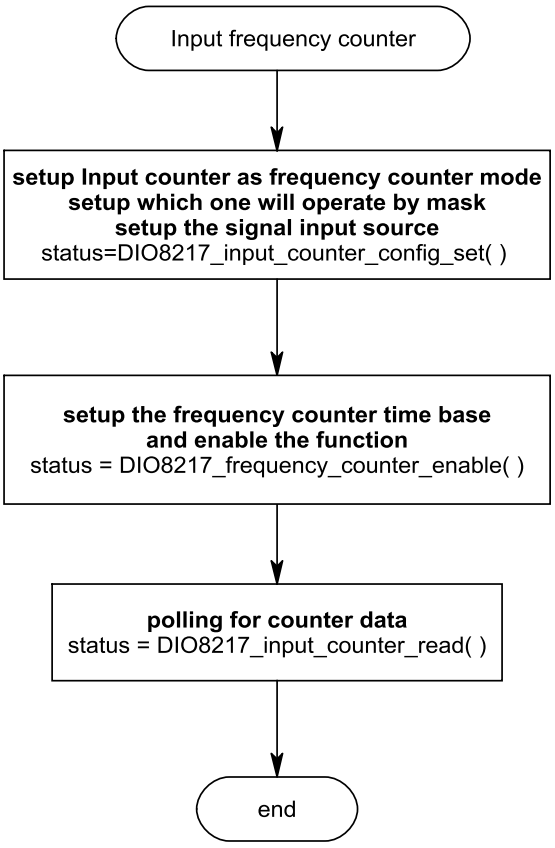
7.1 DIO8217 Flow chart of application implementation











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.

DIO8217_initial( ) to initial the resources of the driver.

DIO8217_close( ) to close the resources of the driver before you close your application.

DIO8217_info( ) get the information of address assigned by the OS.

To check the firmware version,

DIO8217_firmware_version_read( ) will do.

- **DIO8217_initial**

Format: **u32 status =DIO8217_initial (void)**

Purpose: Initial the DIO8217 resource when start the Windows applications.

- **DIO8217_close**

Format: **u32 status =DIO8217_close (void)**

Purpose: Release the DIO8217 resource when close the Windows applications.

- **DIO8217_info**

Format: **u32 status =DIO8217_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

- **DIO8217_firmware_version_read**

Format: **u32 status =DIO8217_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]

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.

Use ***DIO8217_ debounce_time_set( )*** to configure the debounce time.

DIO8217_debounce_time_read( ) to read back the configuration data.

To match the logic polarity of your software, DIO8217 also provides the input output polarity configuration; use

DIO8217_inport_polarity_set( ) to configure the polarity of each input of port,

DIO8217_inport_polarity_read( ) to read back the polarity of each input of port.

DIO8217_outport_polarity_set( ) to configure the polarity of each output of port,

DIO8217_outport_polarity_read( ) to read back the polarity of each output of port.

For the bitwise polarity setting or read back, use

DIO8217_inpoint_polarity_set( ) to configure the polarity of input;

DIO8217_inpoint_polarity_read( ) to read back the polarity of input.

DIO8217_outpoint_polarity_set( ) to configure the polarity of output;

DIO8217_outpoint_polarity_read( ) to read back the polarity of output.

For the port input, output, use:

DIO8217_outport_set( ) to output byte data to output port,

DIO8217_outport_read( ) to read a byte outport data from I/O port,

DIO8217_inport_read( ) to read a byte inport data from I/O port.

For bitwise control, use

DIO8217_outpoint_set( ) to set output bit,

DIO8217_outpoint_read( ) to read outpoint bit,

DIO8217_inpoint_read( ) to read inpoint bit.

The input points (IN07~IN00) 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 low speed/pulse counting. Refer 8.6 Input Counter for detail.

- **DIO8217_debounce_time_set**

Format: **u32 status = DIO8217_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 IN17-IN10
debounce_time	u8	0: no debounce 1: 50 Hz 2: 100 Hz (default) 3: 200 Hz 4: 1k Hz

- **DIO8217_debounce_time_read**

Format: **u32 status = DIO8217_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: input port0 for IN07-IN00 1: input port1 for IN17-IN10

Output:

Name	Type	Description
debounce_time	u8	0: no debounce 1: 50 Hz 2: 100 Hz (default) 3: 200 Hz 4: 1k Hz

- **DIO8217_inport_polarity_set**

Format: **u32 status = DIO8217_inport_polarity_set(u8 CardID, u8 port , u8 polarity)**

Purpose: Sets the inport 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 IN17-IN10
polarity	u8	bitmap of polarity values b7: INx7 ... b0: INx0 bit data =0, normal polarity (default) bit data =1, invert polarity

- **DIO8217_inport_polarity_read**

Format: **u32 status = DIO8217_inport_polarity_read(u8 CardID , u8 port , u8 *polarity)**

Purpose: Read the inport 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 IN17-IN10

Output:

Name	Type	Description
polarity	u8	bitmap of polarity values b7: INx7 ... b0: INx0 bit data =0, normal polarity (default) bit data =1, invert polarity

- **DIO8217_outport_polarity_set**

Format: **u32 status = DIO8217_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: output port0 for OUT07-OUT00 1: output port1 for OUT17-OUT10
polarity	u8	bitmap of polarity values b7: OUTx7 ... b0: OUTx0 bit data =0, normal polarity (default) bit data =1, invert polarity

- **DIO8217_outport_polarity_read**

Format: **u32 status = DIO8217_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: output port0 for OUT07-OUT00 1: output port1 for OUT17-OUT10

Output:

Name	Type	Description
polarity	u8	bitmap of polarity values b7: OUTx7 ... b0: OUTx0 bit data =0, normal polarity (default) bit data =1, invert polarity

- **DIO8217_inpoint_polarity_set**

Format: **u32 status = DIO8217_inpoint_polarity_set(u8 CardID, u8 port , u8 point , u8 state)**

Purpose: Sets the input 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 IN17-IN10
point	u8	point number 7~0 7: INx7 ... 0: INx0
state	u8	0: normal polarity (default) 1: invert polarity

- **DIO8217_inpoint_polarity_read**

Format: **u32 status = DIO8217_inpoint_polarity_read(u8 CardID , u8 port , u8 point , u8 *state)**

Purpose: Read the input 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 IN17-IN10
point	u8	point number 7~0 7: INx7 ... 0: INx0

Output:

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

- **DIO8217_outpoint_polarity_set**

Format: **u32 status = DIO8217_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: output port0 for OUT07-OUT00 1: output port1 for OUT17-OUT10
point	u8	point number 7~0 7: OUTx7 ... 0: OUTx0
state	u8	0: normal polarity (default) 1: invert polarity

- **DIO8217_outpoint_polarity_read**

Format: **u32 status = DIO8217_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: output port0 for OUT07-OUT00 1: output port1 for OUT17-OUT10
point	u8	point number 7~0 7: OUTx7 ... 0: OUTx0

Output:

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

- **DIO8217_outport_set**

Format: **u32 status = DIO8217_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: output port0 for OUT07-OUT00 1: output port1 for OUT17-OUT10
data	u8	output values: b7~b0 for OUTx7~OUTx0

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

- **DIO8217_outport_read**

Format: **u32 status =DIO8217_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: output port0 for OUT07-OUT00 1: output port1 for OUT17-OUT10

Output:

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

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

- **DIO8217_inport_read**

Format: **u32 status =DIO8217_inport_read(u8 CardID, u8 port, u8 *data)**

Purpose: Read input port 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 IN17-IN10

Output:

Name	Type	Description
data	u8	values: b7~b0 for INx7~INx0

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

- **DIO8217_outpoint_set**

Format: **u32 status = DIO8217_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: output port0 for OUT07-OUT00 1: output port1 for OUT17-OUT10
point	u8	point number 7~0 7: OUTx7 ... 0: OUTx0
state	u8	state of output point 0: inactive 1: active

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

- **DIO8217_outpoint_read**

Format: **u32 status =DIO8217_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: output port0 for OUT07-OUT00 1: output port1 for OUT17-OUT10
point	u8	point number 7~0 7: OUTx7 ... 0: 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.

- **DIO8217_inpoint_read**

Format: **u32 status =DIO8217_inpoint_read(u8 CardID , u8 port , u8 point , u8 *state)**

Purpose: Read input point 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 IN17-IN10
point	u8	point number 7~0 for INx7~INx0

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

DIO8217 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 DIO8217 user's manual)

To configure the port as input or output by:

DIO8217_TTL_IO_config_set( ) and read back the configuration by:

DIO8217_TTL_IO_config_read( ).

To change the polarity as you need by:

DIO8217_TTL_IO_port_polarity_set( ) and read back to verify by:

DIO8217_TTL_IO_port_polarity_read( ).

For the bitwise polarity set and read, use:

DIO8217_TTL_IO_point_polarity_set( ) and read back to verify by:

DIO8217_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:

DIO8217_TTL_IO_debounce_time_set( ) to set the adequate time constant to drop out the noise and read back to check the setting by:

DIO8217_TTL_IO_debounce_time_read( ).

To operate the TTL IO, you must enable the port first to release its tri-state

DIO8217_TTL_IO_enable( ) and you can also disable it to tri-state any time you want to go to the power on default by:

DIO8217_TTL_IO_disable( )

The TTL I/O port can use:

DIO8217_TTL_IO_port_set( ) to output data and input data by:

DIO8217_TTL_IO_port_read( ).

For the bitwise point output, use:

DIO8217_TTL_IO_point_set( ) and point input by:

DIO8217_TTL_IO_point_read( ).

- **DIO8217_TTL_IO_config_set**

Format: **u32 status =DIO8217_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)

- **DIO8217_TTL_IO_config_read**

Format: **u32 status =DIO8217_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

- **DIO8217_TTL_IO_port_polarity_set**

Format: u32 status =DIO8217_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

- **DIO8217_TTL_IO_port_polarity_read**

Format: u32 status = DIO8217_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

- **DIO8217_TTL_IO_point_polarity_set**

Format: **u32 status =DIO8217_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

- **DIO8217_TTL_IO_point_polarity_read**

Format: **u32 status = DIO8217_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

- **DIO8217_TTL_IO_debounce_time_set**

Format: **u32 status = DIO8217_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

- **DIO8217_TTL_IO_debounce_time_read**

Format: **u32 status = DIO8217_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

- **DIO8217_TTL_IO_enable**

Format: u32 status =DIO8217_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

- **DIO8217_TTL_IO_disable**

Format: u32 status =DIO8217_TTL_IO_disable(u8 CardID, u8 port)

Purpose: Disable TTL IO.

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

Note: The output will be high or low depends on the JP4, JP5 setting. (refer hardware manual)

- **DIO8217_TTL_IO_port_set**

Format: **u32 status = DIO8217_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.

- **DIO8217_TTL_IO_port_read**

Format: **u32 status = DIO8217_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.

- **DIO8217_TTL_IO_point_set**

Format: **u32 status =DIO8217_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 7~0 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.

- **DIO8217_TTL_IO_point_read**

Format: **u32 status =DIO8217_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 7~0 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 Multi-function Counter / Timer function

The multi-function counter / timer can work as timer (based on 1us system clock), simple counter, up down counter, encoder quadrature counter and PWM generator (based on 33MHz clock).

Timer

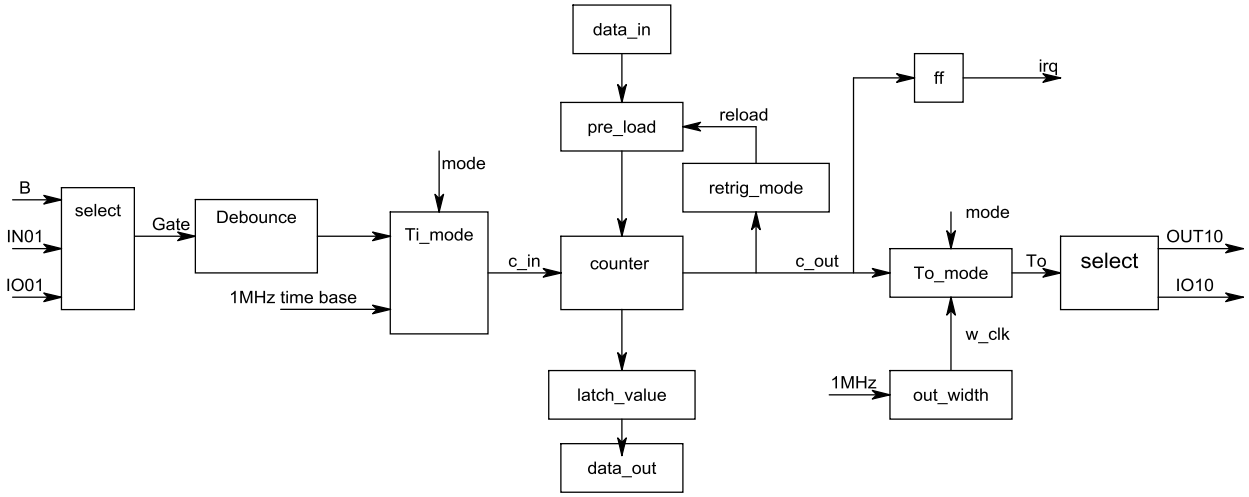


fig. 8.4.1 Timer working model

From the fig. 8.4.1, the timer has a Gate input, gate can be disabled, gated or edge trigger to start timer/counter.

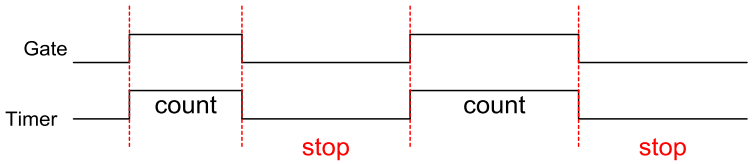


fig. 8.4.2 Gated function

At Gated mode, the gate controls the counting pulse (on timer it is the 1us time base) to go to timer/counter, if gate is active and if timer is also enabled, the timer will run else stops.

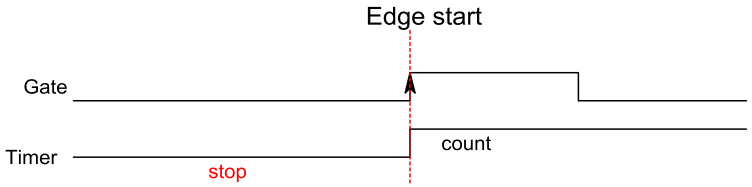


fig. 8.4.3 Gate edge start function

At edge start mode, the gate from space to mark (low to high edge) will trigger the timer/counter to run. If no gate trigger signal input it will keep at idle state to wait.

The Gate input can come from IN01, TTL IO01 or dedicated input B.

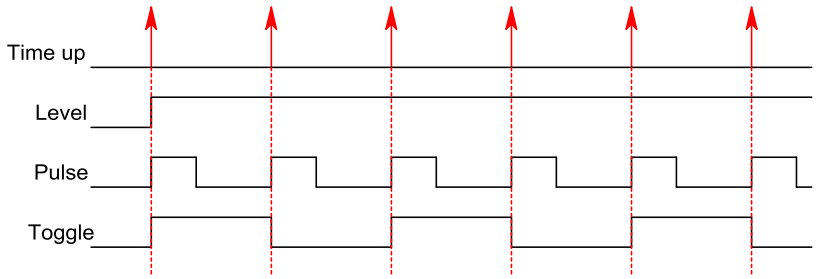


fig. 8.4.4 Output mode

The timer/counter also provides an output; it can be configured as no output (disabled) or output from TL IO10 or OUT10. If the output is valid, the output mode can be level, pulse (duty can be configured) or toggle. Fig. 8.4.4 explains the output according the output mode.

Use ***DIO8217_TC_timer_set( )*** to configure as timer and its output mode.

DIO8217_TC_timer_read( ) to read the timer on the fly.

To start/stop the operation by:

DIO8217_TC_start( )

DIO8217_TC_stop( )

After starting the timer/counter operation, you may want to know the timer/counter value on the fly or the preset value, using

DIO8217_TC_counter_value_read( ) to read for application.

For the advanced users, if they need dedicated function programming, use

DIO8217_TC_debounce_set( ) to configure the Gate input debounce time and read back to verify by

DIO8217_TC_debounce_read( ).

To setup the input (Gate input and its source) by:

DIO8217_TC_input_mode_set( ) and read back for verification by:

DIO8217_TC_input_mode_read( ).

To setup the output mode (T out) by:

DIO8217_TC_output_mode_set( ) and read back for verification by:

DIO8217_TC_output_mode_read( ).

To configure the timer as one cycle timer or continuous timer by:

DIO8217_TC_retriger_mode_set( ) and read back for verification by:

DIO8217_TC_retriger_mode_read( ).

To verify the status of Gate input and T out by:

DIO8217_TC_status_read( ).

To read or load dedicated timer/counter registers, use

DIO8217_TC_set( ) set TC dedicated registers,

DIO8217_TC_read( ) read TC dedicated registers.

Counter

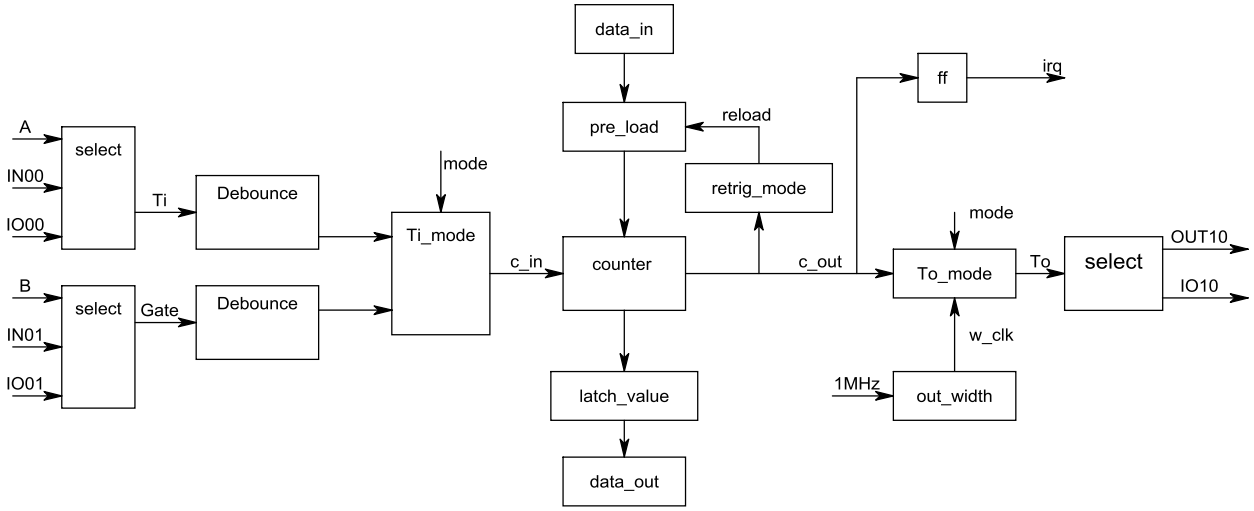


fig. 8.4.5 Counter working model

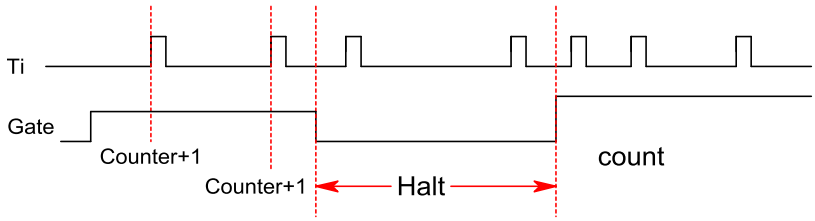


fig. 8.4.6 Counter gated function

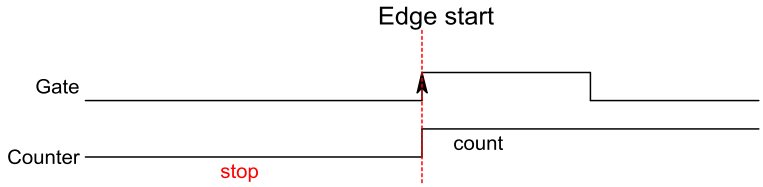


fig. 8.4.7 Counter gate to start

The counter has a Gate input to control the start. (refer fig. 8.4.6 Counter gated function, fig 8.4.7 fig. .1 frequency counter function model

). The counter can be no output or output from TTL IO10 or OUT10, the output mode can be level, pulse (duty can be configured) or toggle (refer fig. 8.4.4 Output mode).

Use ***DIO8217_TC_counter_set( )*** to configure as counter and its output mode.

DIO8217_TC_counter_read( ) to read the counter on the fly.

To start/stop the operation by:

DIO8217_TC_start( )

DIO8217_TC_stop( )

After starting the timer/counter operation, you may want to know the timer/counter value on the fly or the preset value, using

DIO8217_TC_counter_value_read( ) to read for application.

For the advanced users, if they need dedicated function programming, use

DIO8217_TC_debounce_set( ) to configure the Ti, Gate input debounce time and read back to verify by

DIO8217_TC_debounce_read( ).

To setup the input (Ti, Gate input and its source) by:

DIO8217_TC_input_mode_set( ) and read back for verification by:

DIO8217_TC_input_mode_read( ).

To setup the output mode (T out) by:

DIO8217_TC_output_mode_set( ) and read back for verification by:

DIO8217_TC_output_mode_read( ).

To configure the timer as one cycle counter or continuous counter by:

DIO8217_TC_retriger_mode_set( ) and read back for verification by:

DIO8217_TC_retriger_mode_read( ).

To verify the status of Ti, Gate input and T out by:

DIO8217_TC_status_read( ).

To read or load dedicated timer/counter registers, use

DIO8217_TC_set( ) set TC dedicated registers,

DIO8217_TC_read( ) read TC dedicated registers.

Encoder quadrature counter/Up down counter

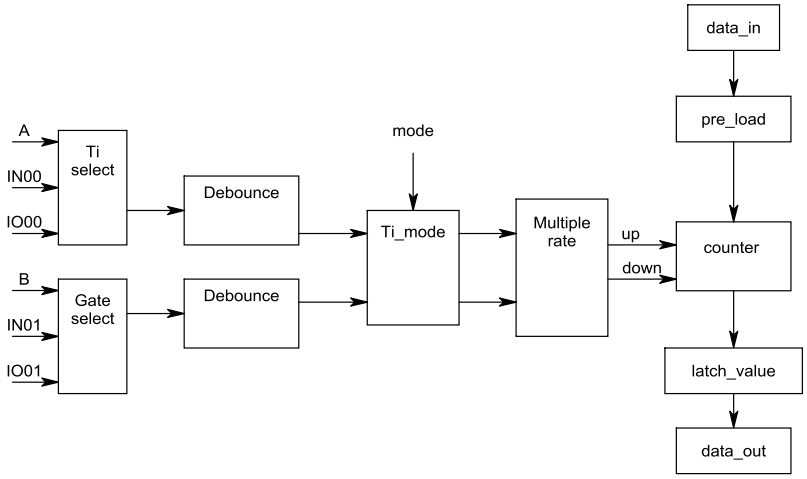
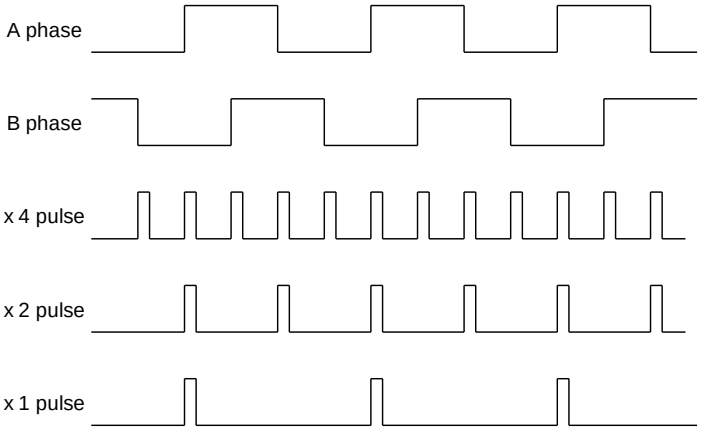


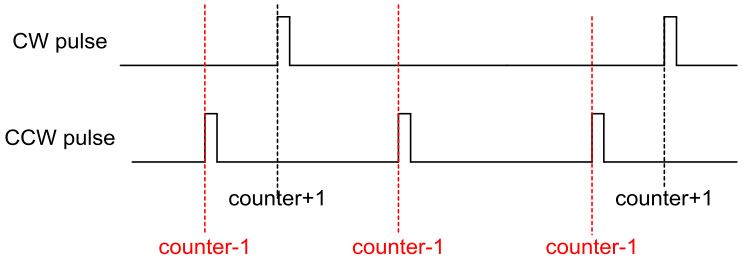
fig. 8.4.8 Quadrature/ Up down counter model



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

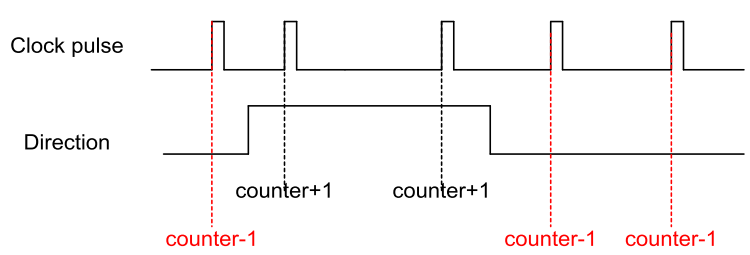
On the other hand, if B phase leads A phase, the counter will be down count.

fig. 8.4.9 Quadrature input and multiple rate



The left diagram shown that CW and CCW pulses. Any CW pulse input will increase counter by 1 and any CCW pulse input will decrease counter by 1.

fig. 8.4.10 CW,CCW input and counting



The left diagram shown that Clock and Direction pulses. Any Clock pulse input will increase counter by 1 while the Direction signal is mark and any Clock pulse input will decrease counter by 1 while Direction signal is space.

fig. 8.4.11 Clock, direction input and counting

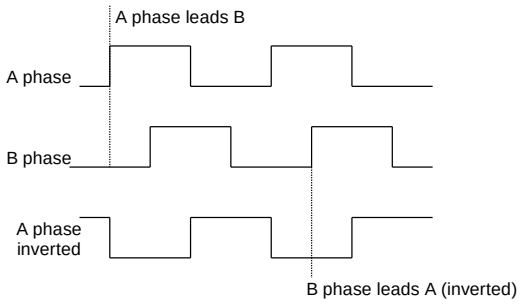


fig. 8.4.12 A,B phase input and phase inversion

The encoder quadrature counter/ up down counter has 2 inputs to control the counter counting and its direction (up count or down count).

To use the integrated function call, configure the working mode use

DIO8217_TC_up_down_counter_set( ) to configure its working mode.

DIO8217_TC_up_down_counter_read( ) to read the counter on the fly.

To start/stop the operation by:

DIO8217_TC_start( )

DIO8217_TC_stop( )

After starting the timer/counter operation, you may want to know the timer/counter value on the fly or the preset value, using

DIO8217_TC_counter_value_read( ) to read for application.

For the advanced users, if they need dedicated function programming, use

DIO8217_TC_debounce_set( ) to configure the Ti, Gate input debounce time and read back to verify by

DIO8217_TC_debounce_read( ).

To setup the input (Ti, Gate input and its source) by:

DIO8217_TC_input_mode_set( ) and read back for verification by:

DIO8217_TC_input_mode_read( ).

To verify the status of inputs by:

DIO8217_TC_status_read( ).

To read or load dedicated timer/counter registers, use

DIO8217_TC_set( ) set TC dedicated registers,

DIO8217_TC_read( ) read TC dedicated registers.

PWM generator

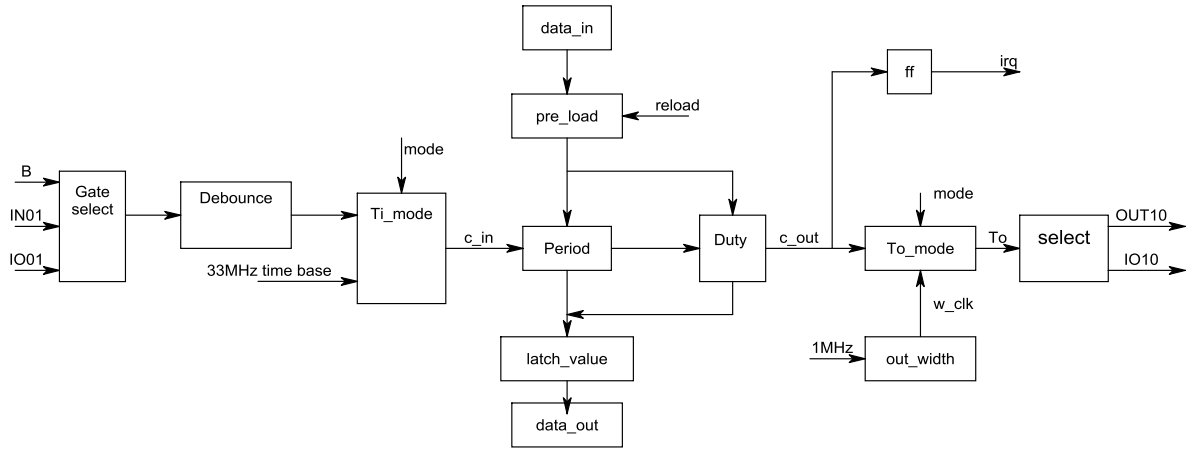


fig. 8.4.13 PWM working model

The PWM function is based on 33MHZ time base and the counter of PWM period is 16 bit and the duty also 16 bit width.

To use the integrated function call, configure the working mode use

DIO8217_TC_PWM_set( ) to configure as PWM generator.

DIO8217_TC_PWM_read( ) to read the counter on the fly.

To start/stop the operation by:

DIO8217_TC_start( )

DIO8217_TC_stop( )

After starting the timer/counter operation, you may want to know the timer/counter value on the fly or the preset value, using

DIO8217_TC_counter_value_read( ) to read for application.

For the advanced users, if they need dedicated function programming, use

DIO8217_TC_debounce_set( ) to configure the Gate input debounce time and read back to verify by

DIO8217_TC_debounce_read( ).

To setup the input (Gate input and its source) by:

DIO8217_TC_input_mode_set( ) and read back for verification by:

DIO8217_TC_input_mode_read( ).

To setup the output mode (T out) by:

DIO8217_TC_output_mode_set( ) and read back for verification by:

DIO8217_TC_output_mode_read( ).

To configure the timer as one cycle counter or continuous counter by:

DIO8217_TC_retriger_mode_set( ) and read back for verification by:

DIO8217_TC_retriger_mode_read( ).

To verify the status of Gate input by:

DIO8217_TC_status_read( ).

To read or load dedicated timer/counter registers, use

DIO8217_TC_set( ) set TC dedicated registers,

DIO8217_TC_read( ) read TC dedicated registers.

- **DIO8217_TC_timer_set**

Format: u32 status = DIO8217_TC_timer_set(u8 CardID, timer_struct *timer)

Purpose: set timer parameter.

Parameters:

Input:

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

timer	timer_struct	struct timer_struct
		{
		u32 time_constant;
		// Timer constant based on 1us clock
		
		u8 Tout_mode;
		//0: NO_TOUT (default)
		//1: OUT_PULSE
		//(Counter cross zero output pulse.
		//(Tout_width effective))
		//2: OUT_LEVEL
		//(Counter cross zero output edge transition.)
		//3: OUT_TOGGLE
		//(Counter cross zero toggles output)
		u8 Tout_source;
		//0: No output(default)
		//1: timer output is from OUT00
		//2: timer output is from TTL IO10
		//(TTL Port1 is outport)
		u16 Tout_width,
		// Output pulse width based on 1us clock, only
		//valid in Tout_mode is OUT_PULSE
		
		u8 Gate_mode;
		//0: NO_GATE(Always count without gate
		// function, default)
		//1:GATED (Count on Gate input active.)
		//2:EDGE_START
		//(Start count on Gate input edge transition.)
		u8 Gate_source;
		//0: use dedicate input B (default)
		//1: Gated is from IN01
		//2: Gated is from TTL IO01
		//(TTL Port0 is inport)
		
		u8 Gate_debounce;
		//0: no debounce
		//1: 50 Hz
		//2: 100 Hz (default)
		//3: 200 Hz
		//4: 1K Hz
		//5: 2K Hz

		//6: 10K Hz
		//7: 20K Hz
		//8: 50K Hz
		//9: 100K Hz
		//10: 200K Hz
		//11: 500K Hz
		//12: 1M Hz
		//13: 2M Hz
		//14: 4M Hz
		//15: 8M Hz
		
		u8 Retrigger_mode;
		//0: one cycle
		//1: continuous mod, auto reload counter
		// from pre_load.
		}

Note:

1. It is recommend that debounce time selection 0~6 for IN01, 0~15 for TTL IO01
2. Timer output will be active on the time count cross zero.

- **DIO8217_TC_timer_read**

Format: u32 status = DIO8217_TC_timer_read(u8 CardID, timer_struct *timer)

Purpose: read timer parameter.

Parameters:

Input:

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

Output:

Name	Type	Description

timer	timer_struct	struct timer_struct
		{
		u32 time_constant;
		// Timer constant based on 1us clock
		
		u8 Tout_mode;
		//0: NO_TOUT (default)
		//1: OUT_PULSE
		//(Counter cross zero output pulse.
		//(Tout_width effective))
		//2: OUT_LEVEL
		//(Counter cross zero output edge transition.)
		//3: OUT_TOGGLE
		//(Counter cross zero toggles output)
		u8 Tout_source;
		//0: No output(default)
		//1: timer output is from OUT00
		//2: timer output is from TTL IO10
		//(TTL Port1 is outport)

		u16 Tout_width,
		// Output pulse width based on 1us clock, only
		//valid in Tout_mode is OUT_PULSE
		
		u8 Gate_mode;
		//0: NO_GATE(Always count without gate
		// function, default)
		//1:GATED (Count on Gate input active.)
		//2:EDGE_START
		//(Start count on Gate input edge transition.)
		u8 Gate_source;
		//0: use dedicated input B (default)
		//1: Gated is from IN01
		//2: Gated is from TTL IO01
		//(TTL Port0 is inport)
		
		u8 Gate_debounce;
		//0: no debounce
		//1: 50 Hz
		//2: 100 Hz (default)
		//3: 200 Hz
		//4: 1K Hz
		//5: 2K Hz
		//6: 10K Hz
		//7: 20K Hz
		//8: 50K Hz
		//9: 100K Hz
		//10: 200K Hz
		//11: 500K Hz
		//12: 1M Hz
		//13: 2M Hz
		//14: 4M Hz
		//15: 8M Hz
		
		u8 Retrigger_mode;
		//0: one cycle
		//1: continuous mod, auto reload counter
		// from pre_load.
		}

Note:

1. It is recommend that debounce time selection 0~6 for IN01, 0~15 for TTL IO01
2. Timer output will be active on the time count cross zero.

- **DIO8217_TC_counter_set**

Format: **u32 status = DIO8217_TC_counter_set(u8 CardID, counter_struct *counter)**

Purpose: set counter parameter.

Parameters:

Input:

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

counter	counter_struct	struct counter_struct
		{
		u32 counter_constant;
		// counter constant to be set
		u8 Ti_source;
		//0: use dedicated input A (default)
		//1: Ti is from IN0
		//2: Ti is from TTL IO00
		//(TTL Port0 is inport)
		
		u8 Ti_debounce;
		//0: no debounce
		//1: 50 Hz
		//2: 100 Hz(default)
		//3: 200 Hz
		//4: 1K Hz
		//5: 2K Hz
		//6: 10K Hz
		//7: 20K Hz
		//8: 50K Hz
		//9: 100K Hz
		//10: 200K Hz
		//11: 500K Hz
		//12: 1M Hz
		//13: 2M Hz
		//14: 4M Hz
		//15: 8M Hz
		
		u8 Tout_mode;
		//0: NO_TOUT (default)
		//1: OUT_PULSE
		//(Counter cross zero output pulse.
		//(Tout_width effective))
		//2: OUT_LEVEL
		//(Counter cross zero output edge transition.)
		//3: OUT_TOGGLE
		//(Counter cross zero toggles output)
		u8 Tout_source;
		//0: No output(default)
		//1: counter output is from OUT00
		//2: counter output is from TTL IO10
		//(TTL Port1 is outport)

		u16 Tout_width,
		// Output pulse width based on 1us clock, only
		//valid in Tout_mode is OUT_PULSE
		
		u8 Gate_mode;
		//0: NO_GATE(Always count without gate
		// function, default)
		//1:GATED (Count on Gate input active.)
		//2:EDGE_START
		//(Start count on Gate input edge transition.)
		u8 Gate_source;
		//0: use dedicated input B (default)
		//1: Gated is from IN01
		//2: Gated is from TTL IO01
		//(TTL Port0 is inport)
		
		u8 Gate_debounce;
		//0: no debounce
		//1: 50 Hz
		//2: 100 Hz(default)
		//3: 200 Hz
		//4: 1K Hz
		//5: 2K Hz
		//6: 10K Hz
		//7: 20K Hz
		//8: 50K Hz
		//9: 100K Hz
		//10: 200K Hz
		//11: 500K Hz
		//12: 1M Hz
		//13: 2M Hz
		//14: 4M Hz
		//15: 8M Hz
		
		u8 Retrigger_mode;
		//0: one cycle
		//1: auto reload counter from pre_load.
		//(continuous mode)
		}

Note:

1. It is recommend that debounce time selection 0~6 for IN01, 0~15 for TTL IO01
2. Counter output will be active on the counter count cross zero.

- **DIO8217_TC_counter_read**

Format: **u32 status = DIO8217_TC_counter_read(u8 CardID, counter_struct *counter)**

Purpose: read counter parameter.

Parameters:

Input:

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

Output:

Name	Type	Description

counter	counter_struct	struct counter_struct
		{
		u32 counter_constant;
		// counter constant to be set
		u8 Ti_source;
		//0: use dedicated input A (default)
		//1: Ti is from IN0
		//2: Ti is from TTL IO00
		//(TTL Port0 is inport)
		
		u8 Ti_debounce;
		//0: no debounce
		//1: 50 Hz
		//2: 100 Hz(default)
		//3: 200 Hz
		//4: 1K Hz
		//5: 2K Hz
		//6: 10K Hz
		//7: 20K Hz
		//8: 50K Hz
		//9: 100K Hz
		//10: 200K Hz
		//11: 500K Hz
		//12: 1M Hz
		//13: 2M Hz
		//14: 4M Hz
		//15: 8M Hz
		
		u8 Tout_mode;
		//0: NO_TOUT (default)
		//1: OUT_PULSE
		//(Counter cross zero output pulse.
		//(Tout_width effective))
		//2: OUT_LEVEL
		//(Counter cross zero output edge transition.)
		//3: OUT_TOGGLE
		//(Counter cross zero toggles output)
		u8 Tout_source;
		//0: No output(default)
		//1: counter output is from OUT00

		//2: counter output is from TTL IO10
		//(TTL Port1 is outport)
		
		u16 Tout_width,
		// Output pulse width based on 1us clock, only
		//valid in Tout_mode is OUT_PULSE
		
		u8 Gate_mode;
		//0: NO_GATE(Always count without gate
		// function, default)
		//1:GATED (Count on Gate input active.)
		//2:EDGE_START
		//(Start count on Gate input edge transition.)
		u8 Gate_source;
		//0: use dedicated input B (default)
		//1: Gated is from IN01
		//2: Gated is from TTL IO01
		//(TTL Port0 is inport)
		
		u8 Gate_debounce;
		//0: no debounce
		//1: 50 Hz
		//2: 100 Hz(default)
		//3: 200 Hz
		//4: 1K Hz
		//5: 2K Hz
		//6: 10K Hz
		//7: 20K Hz
		//8: 50K Hz
		//9: 100K Hz
		//10: 200K Hz
		//11: 500K Hz
		//12: 1M Hz
		//13: 2M Hz
		//14: 4M Hz
		//15: 8M Hz
		
		u8 Retrigger_mode;
		//0: one cycle
		//1: auto reload counter from pre_load.
		//(continuous mode)
		}

Note:

1. It is recommend that debounce time selection 0~6 for IN01, 0~15 for TTL IO01
2. Counter output will be active on the counter count cross zero.

- **DIO8217_TC_up_down_counter_set**

Format: **u32 status = DIO8217_TC_up_down_counter_set(u8 CardID,**
up_down_counter_struct * up_down_counter)

Purpose: setup **up_down_counter** parameter.

Parameters:

Input:

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

up_down_ counter	up_down_ counter_struct	struct up_down_counter_struct
		{
		u32 counter_constant;
		// counter constant to be set
		u8 control_mode
		0: quardature mode
		1: single pulse: A as clock, B as direction
		2: daul pulse: A as CW, B as CCW
		u8 multiple_rate;(only quardature mode use)
		0: multiple rate x4 (default)
		1: multiple rate x2
		2: multiple rate x1
		
		u8 A_source;
		//0: use dedicated input A (default)
		//1: A is from IN00
		//2: A is from TTL IO00
		//(TTL Port0 is inport)
		u8 A_debounce;
		//0: no debounce
		//1: 50 Hz
		//2: 100 Hz
		//3: 200 Hz
		//4: 1K Hz
		//5: 2K Hz
		//6: 10K Hz
		//7: 20K Hz
		//8: 50K Hz
		//9: 100K Hz
		//10: 200K Hz
		//11: 500K Hz
		//12: 1M Hz
		//13: 2M Hz
		//14: 4M Hz
		//15: 8M Hz
		
		u8 B_source;
		//0: use dedicated input B (default)
		//1: B is from IN01
		//2: B is from TTL IO01
		//(TTL Port0 is inport)

		u8 B_debounce; //0: no debounce //1: 50 Hz //2: 100 Hz //3: 200 Hz //4: 1K Hz //5: 2K Hz //6: 10K Hz //7: 20K Hz //8: 50K Hz //9: 100K Hz //10: 200K Hz //11: 500K Hz //12: 1M Hz //13: 2M Hz //14: 4M Hz //15: 8M Hz }

Note: The debounce time selection will depends on the speed of your encoder, low speed maybe several hundreds, medium maybe in the range of several KHz and high speed will be MHz range.

- **DIO8217_TC_up_down_counter_read**

Format: **u32 status = DIO8217_TC_up_down_counter_read(u8 CardID, up_down_counter_struct * up_down_counter)**

Purpose: read back **up_down_counter** parameter.

Parameters:

Input:

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

Output:

Name	Type	Description

up_down_ counter	up_down_ counter_struct	struct up_down_counter_struct
		{
		u32 counter_constant;
		// counter constant to be set
		u8 control_mode
		0: quardature mode
		1: single pulse: A as clock, B as direction
		2: daul pulse: A as CW, B as CCW
		u8 multiple_rate;(only quardature mode use)
		0: multiple rate x4 (default)
		1: multiple rate x2
		2: multiple rate x1
		
		u8 A_source;
		//0: use dedicated input A (default)
		//1: A is from IN00
		//2: A is from TTL IO00

		//(TTL Port0 is inport) u8 A_debounce; //0: no debounce //1: 50 Hz //2: 100 Hz //3: 200 Hz //4: 1K Hz //5: 2K Hz //6: 10K Hz //7: 20K Hz //8: 50K Hz //9: 100K Hz //10: 200K Hz //11: 500K Hz //12: 1M Hz //13: 2M Hz //14: 4M Hz //15: 8M Hz u8 B_source; //0: use dedicated input B (default) //1: B is from IN01 //2: B is from TTL IO01 //(TTL Port0 is inport) u8 B_debounce; //0: no debounce //1: 50 Hz //2: 100 Hz //3: 200 Hz //4: 1K Hz //5: 2K Hz //6: 10K Hz //7: 20K Hz //8: 50K Hz //9: 100K Hz //10: 200K Hz //11: 500K Hz //12: 1M Hz //13: 2M Hz //14: 4M Hz //15: 8M Hz }

Note: The debounce time selection will depends on the speed of your encoder,low speed maybe several hundreds, medium maybe in the range of several KHz and high speed will be MHz range.

- **DIO8217_TC_PWM_set**

Format: u32 status = DIO8217_TC_PWM_set(u8 CardID, PWM_struct *PWM)

Purpose: set PWM parameter.

Parameters:

Input:

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

PWM	PWM_struct	struct pwm_struct
		{
		u16 pwm_period;
		//period = (1/33MHz) * pwm_period
		u16 pwm_duty;
		//duty = (1/33MHz) * pwm_duty
		
		u8 Tout_source;
		//0: No output(default)
		//1: counter output is from OUT00
		//2: counter output is from TTL IO10
		//(TTL Port1 is outport)
		
		u8 Gate_mode;
		//0: NO_GATE(Always count without gate
		// function, default)
		//1:GATED (Count on Gate input active.)
		//2:EDGE_START
		//(Start count on Gate input edge transition.)
		u8 Gate_source;
		//0: use dedicated input B (default)
		//1: Gated is from IN01
		//2: Gated is from TTL_port01
		//(TTL Port0 is inport)
		u8 Gate_debounce;
		//0: no debounce
		//1: 50 Hz
		//2: 100 Hz
		//3: 200 Hz
		//4: 1K Hz
		//5: 2K Hz
		//6: 10K Hz
		//7: 20K Hz
		//8: 50K Hz
		//9: 100K Hz
		//10: 200K Hz
		//11: 500K Hz
		//12: 1M Hz
		//13: 2M Hz
		//14: 4M Hz
		//15: 8M Hz
		}

Note: It is recommend that debounce time selection 0~6 for IN01, 0~15 for TTL IO01

- **DIO8217_TC_PWM_read**

Format: **u32 status = DIO8217_TC_PWM_read(u8 CardID, PWM_struct *PWM)**

Purpose: read PWM parameter.

Parameters:

Input:

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

Output:

Name	Type	Description

PWM	PWM_struct	struct pwm_struct
		{
		u16 pwm_period;
		//period = (1/33MHz) * pwm_period
		u16 pwm_duty;
		//duty = (1/33MHz) * pwm_duty
		
		u8 Tout_source;
		//0: No output(default)
		//1: counter output is from OUT00
		//2: counter output is from TTL IO10
		//(TTL Port1 is outport)
		
		u8 Gate_mode;
		//0: NO_GATE(Always count without gate function, default)
		//1:GATED (Count on Gate input active.)
		//2:EDGE_START
		//(Start count on Gate input edge transition.)
		u8 Gate_source;
		//0: use dedicated input B (default)
		//1: Gated is from IN01
		//2: Gated is from TTL_port01
		//(TTL Port0 is inport)
		u8 Gate_debounce;
		//0: no debounce
		//1: 50 Hz
		//2: 100 Hz
		//3: 200 Hz
		//4: 1K Hz
		//5: 2K Hz
		//6: 10K Hz
		//7: 20K Hz
		//8: 50K Hz
		//9: 100K Hz
		//10: 200K Hz
		//11: 500K Hz

		//12: 1M Hz //13: 2M Hz //14: 4M Hz //15: 8M Hz }

Note: It is recommend that debounce time selection 0~6 for IN01, 0~15 for TTL IO01

- **DIO8217_TC_start**

Format: u32 status = DIO8217_TC_start(u8 CardID)

Purpose: start timer / counter / up down counter / PWM function.

Parameters:

Input:

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

- **DIO8217_TC_stop**

Format: u32 status = DIO8217_TC_stop(u8 CardID)

Purpose: stop timer / counter / up down counter / PWM function.

Parameters:

Input:

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

- **DIO8217_TC_counter_value_read**

Format: **u32 status = DIO8217_TC_counter_value_read(u8 CardID, u8 index, u32 *value)**

Purpose: Read TC timer/counter working value on the fly

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY switch(**0x0-0xF**)
index	u8	0: working counter on the fly 1: counter preset

Output:

Name	Type	Description
value	u32	Counter preset or working value on the fly Depends on the working mode, the working counter value on the fly may be: -- u32 working timer (in timer mode) -- u32 working counter(in counter mode) -- u32 working up down counter (in up down counter mode) -- u16 working PWM period(high word), u16 working PWM duty(low word) (in PWM mode)

- **DIO8217_TC_mode_set**

Format: **u32 status= DIO8217_TC_mode_set(u8 CardID, u8 mode)**

Purpose: set TC mode parameter

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY switch(**0x0-0xF**)
mode	u8	0: TIMER_MODE (internal clock based on 1MHz(as timer).) 1: COUNTER_MODE (external(as counter).) 2: PWM_MODE. 3: QUADRATURE_MODE (A,B phase input) 4: CW, CCW mode 5: Clock, Direction mode

- **DIO8217_TC_mode_read**

Format: u32 status= DIO8217_TC_mode_read (u8 CardID, u8 *mode)

Purpose: Read TC mode parameter

Parameters:

Input:

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

Output:

Name	Type	Description
mode	u8	0: TIMER_MODE (internal clock based on 1MHz(as timer).) 1: COUNTER_MODE (external(as counter).) 2: PWM_MODE. 3: QUADRATURE_MODE (A,B phase input) 4: CW, CCW mode 5: Clock, Direction mode

- **DIO8217_TC_debounce_set**

Format: u32 status= DIO8217_TC_debounce_set (u8 CardID, u8 index, u8 debounce)

Purpose: set Ti/Gate debounce parameter

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY switch(**0x0-0xF**)
index	u8	0: Ti 1:Gate
debounce	u8	0: no debounce 1: 50 Hz 2: 100 Hz 3: 200 Hz 4: 1K Hz 5: 2K Hz 6: 10K Hz 7: 20K Hz 8: 50K Hz 9: 100K Hz 10: 200K Hz 11: 500K Hz 12: 1M Hz(default) 13: 2M Hz 14: 4M Hz 15: 8M Hz

- **DIO8217_TC_debounce_read**

Format: u32 status= DIO8217_TC_debounce_read (u8 CardID, u8 index, u8 *debounce)

Purpose: Read Ti/Gate debounce parameter

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY switch(**0x0-0xF**)
index	u8	0: Ti 1:Gate

Output:

Name	Type	Description
debounce	u8	0: no debounce 1: 50 Hz 2: 100 Hz 3: 200 Hz 4: 1K Hz 5: 2K Hz 6: 10K Hz 7: 20K Hz 8: 50K Hz 9: 100K Hz 10: 200K Hz 11: 500K Hz 12: 1M Hz(default) 13: 2M Hz 14: 4M Hz 15: 8M Hz

- **DIO8217_TC_input_mode_set**

Format: **u32 status= DIO8217_TC_input_mode_set (u8 CardID , u8 gate_mode , u8 gate_source , u8 Ti_source)**

Purpose: set TC input mode parameter

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY switch(**0x0-0xF**)
gate_mode	u8	0: NO_GATE (Always count without gate function, default) 1:GATED (Count on Gate input active.) 2:EDGE_START (Start count on Gate input edge transition.)
gate_source	u8	0: Use dedicated input B 1: Gated is from IN01 2: Gated is from TTL_port01
Ti_source	u8	0: Use dedicated input A 1: Ti is from IN0 2: Ti is from TTL_port00

- **DIO8217_TC_input_mode_read**

Format: **u32 status= DIO8217_TC_input_mode_read (u8 CardID,u8 *gate_mode , u8 *gete_source , u8 *Ti_source)**

Purpose: Read TC input mode parameter

Parameters:

Input:

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

Output:

Name	Type	Description
gate_mode	u8	0: NO_GATE (Always count without gate function, default) 1:GATED (Count on Gate input active.) 2:EDGE_START (Start count on Gate input edge transition.)
gate_source	u8	0: Use dedicated input B 1: Gated is from IN01 2: Gated is from TTL_port01
Ti_source	u8	0: Use dedicated input A 1: Ti is from IN0 2: Ti is from TTL_port00

- **DIO8217_TC_output_mode_set**

Format: u32 status= DIO8217_TC_output_mode_set (u8 CardID, u8 Tout_mode , u8 Tout_source , u16 Tout_width)

Purpose: set TC output mode parameter

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY switch(**0x0-0xF**)
Tout_mode	u8	0: NO_TOUT (default) 1: OUT_PULSE (TC cross zero output pulse. (Tout_width effective)) 2: OUT_LEVEL (TC cross zero output edge transition.) 3: OUT_TOGGLE (TC cross zero toggles output)
Tout_source	u8	0: No output(default) 1: TC output is from OUT00 2: TC output is from TTL_port10
Tout_width	u16	Output pulse width based on 1us clock, only valid in Tout_mode is OUT_PULSE

- **DIO8217_TC_output_mode_read**

Format: u32 status= DIO8217_TC_output_mode_read (u8 CardID,u8 *Tout_mode , u8 *Tout_source , u16 *Tout_width)

Purpose: Read TC output mode parameter

Parameters:

Input:

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

Output:

Name	Type	Description
Tout_mode	u8	0: NO_TOUT (default) 1: OUT_PULSE (TC cross zero output pulse. (Tout_width effective)) 2: OUT_LEVEL (TC cross zero output edge transition.) 3: OUT_TOGGLE (TC cross zero toggles output)
Tout_source	u8	0: No output(default) 1: TC output is from OUT00 2: TC output is from TTL_port10
Tout_width	u16	Output pulse width based on 1us clock, only valid in Tout_mode is OUT_PULSE

- **DIO8217_TC_retrigger_mode_set**

Format: **u32 status= DIO8217_TC_retrigger_mode_set (u8 CardID, u8 retrigger)**

Purpose: set TC retrigger mode parameter

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY switch(**0x0-0xF**)
retrigger	u8	0: one cycle 1: continuous mode , auto reload counter from pre_load.

- **DIO8217_TC_retrigger_mode_read**

Format: **u32 status= DIO8217_TC_output_mode_read (u8 CardID,u8 *retrigger)**

Purpose: Read TC retrigger mode parameter

Parameters:

Input:

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

Output:

Name	Type	Description
retrigger	u8	0: one cycle 1: continuous mode , auto reload counter from pre_load.

- **DIO8217_TC_status_read**

Format: **u32 status= DIO8217_TC_status_read (u8 CardID,u8 *status)**

Purpose: Read TC status mode parameter

Parameters:

Input:

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

Output:

Name	Type	Description
status	u8	bit 0: TC output status bit 1: Ti input status bit 2: Gate input status

- **DIO8217_TC_set**

Format: u32 status= DIO8217_TC_set (u8 CardID, u8 index, u32 data)

Purpose: set TC parameter

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY switch(**0x0-0xF**)
index	u8	0: TC_CONTROL 1: TC_MODE 2: COUNTER 3: PRELOAD 4: Ti_SOURCE 5: Ti_DEBOUNCE 6: Gate_MODE 7: Gate_SOURCE 8: Gate_DEBOUNCE 9: Tout_MODE 10: Tout_SOURCE 11: Tout_WIDTH 12: RETRIGGER_mode 13: MULTIPLE_RATE
data	u32	register data to be set

- **DIO8217_TC_read**

Format: u32 status= DIO8217_TC_read (u8 CardID, u8 index, u32 *data)

Purpose: Read TC parameter

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY switch(**0x0-0xF**)
index	u8	0: TC_CONTROL 1: TC_MODE 2: COUNTER 3: PRELOAD 4: Ti_SOURCE 5: Ti_DEBOUNCE 6: Gate_MODE 7: Gate_SOURCE 8: Gate_DEBOUNCE 9: Tout_MODE 10: Tout_SOURCE 11: Tout_WIDTH 12: RETRIGGER_mode 13: MULTIPLE_RATE

Output:

Name	Type	Description
data	u32	register data to be set

index	register	value	meaning
0	TC_CONTROL	0	STOP, stop operation of TC
		1	START, start operation of TC
1	TC_MODE	0	TIMER_MODE
		1	COUNTER_MODE
		2	PWM_MODE
		3	QUADRATURE_MODE
		4	CW_CCW_MODE
		5	CLOCK_DIR_MODE
2	COUNTER	1~0xffffffff	Set (write): will write preload and counter Read : will read counter on the fly
3	PRELOAD	1~0xffffffff	Counter or timer or PWM preload value
4	Ti_SOURCE	0	dedicated A input (default)
		1	Ti/A is from IN00
		2	Ti/A is from TTL_port00
5	Ti_DEBOUNCE	0	no debounce
		1	50 Hz
		2	100 Hz(default)
		3	200 Hz
		4	1KHz
		5	2KHz
		6	10KHz
		7	20KHz
		8	50KHz
		9	100KHz
		10	200KHz
		11	500KHz
		12	1MHz
		13	2MHz
		14	4MHz
		15	8MHz
6	Gate_MODE	0	NO_GATE
		1	GATED
		2	EDGE_START
7	Gate_SOURCE	0	dedicated B input (default)
		1	Gated is from IN01
		2	Gated is from TTL_port01
8	Gate_DEBOUNCE	with Ti_DEBOUNCE same	
9	Tout_MODE	0	NO_TOUT
		1	OUT_PULSE
		2	OUT_LEVEL
		3	OUT_TOGGLE
10	Tout_SOURCE	0	0: No output(default)
		1	tc output is from OUT00

		2	tc output is from TTL_port10
11	Tout_WIDTH	1~0xffff	Output pulse width based on 1us
12	RETRIGGER_MODE	0	SINGLE_CYCLE
		1	ALWAYS_RUN
13	MULTIPLE_RATE	0	0: MULTIPLE_4 (default) A,B phase input multiple rate is 4
		1	1: MULTIPLE_2 A,B phase input multiple rate is 2
		2	2: MULTIPLE_1 A,B phase input multiple rate is 1

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 a 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:

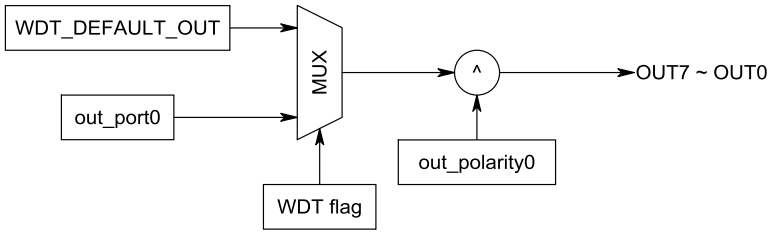


fig 8.5.1 watch dog timer

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

DIO8217_WDT_output_set( ) to setup the default output and read back for verification by ***DIO8217_WDT_output_read()***

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

DIO8217_WDT_start( ) will do. Once you start it, you must reset it before time up by: ***DIO8217_WDT_reset( ).***

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

DIO8217_WDT_stop( ) and on any time you can check the WDT status by ***DIO8217_WDT_read( )***

- **DIO8217_WDT_output_set**

Format: **u32 status = DIO8217_WDT_output_set(u8 CardID, u8 output)**

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

Parameters:

Input:

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

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

- **DIO8217_WDT_output_read**

Format: **u32 status = DIO8217_WDT_output_read(u8 CardID,u8 *output)**

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

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.

- **DIO8217_WDT_start**

Format: **u32 status = DIO8217_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

- **DIO8217_WDT_reset**

Format: **u32 status = DIO8217_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**)

- **DIO8217_WDT_stop**

Format: **u32 status = DIO8217_WDT_stop(u8 CardID)**

Purpose: To stop WDT function.

Parameters:

Input:

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

- **DIO8217_WDT_read**

Format: **u32 status = DIO8217_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

You can configure the input counter as pure counter function or work with the timer as frequency counter.

Input Counter

The DIO8217 IN07~IN00 inputs or TTL IO07~IO00 can work as counter input to 16bit COUNTER7 ~ COUNTER0. The counter model is shown as follows:

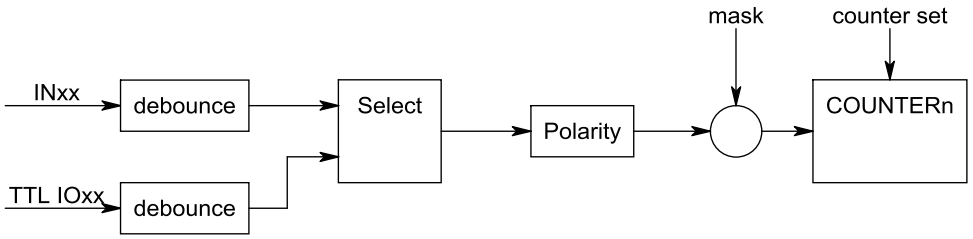


fig. 8.6.1 counter function model

From the model, you can see the input polarity and debounce block, they are the same as general purpose input (Refer ***DIO8217_IO_port_polarity_set( ), DIO8217_IO_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.

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 or from isolated input IN07~IN00 , mask off the unused channels by

DIO8217_input_counter_config_set( ) and read back for verification by

DIO8217_input_counter_config_read( )

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

DIO8217_input_counter_all_set( ) to set values to all counters (set 0 value functions as counter reset) or read all counters by:

DIO8217_input_counter_all_read( ) or set single counter's value by:

DIO8217_input_counter_set( ) and read back single counter's value by:

DIO8217_input_counter_read( )

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

DIO8217_input_counter_mask_set( ) and read back by

DIO8217_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:

DIO8217_input_counter_control_set( ) and read back to check the status by:

DIO8217_input_counter_control_read( )

- **DIO8217_input_counter_config_set**

Format: u32 status = DIO8217_input_counter_config_set(u8 CardID, u8 mask, u8 sourc, u8 mode)

Purpose: set the inport or TTL/IO to work as 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	bit7~bit0 bit7 0 : inport IN07 1: IO07* ... bit0 0 : inport IN00 1: IO00*
mode	u8	bit7~bit0 bit7 for counter7 work as in_counter or frequency counter 0: in_counter 1: frequency_counter ... bit0 for counter0 0: in_counter 1: frequency_counter

*To select the signal source from TTL IO07~IO00 will also set the TTL port0 to input mode.

- **DIO8217_input_counter_config_read**

Format: u32 status = DIO8217_input_counter_config_read(u8 CardID, u8 *mask, u8 *sourc, u8 *mode)

Purpose: read the inport or TTL/IO configuration as input of 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	bit7~bit0 bit7 0 : inport IN07 1: IO07* ... bit0 0 : inport IN00 1: IO00*
mode	u8	bit7~bit0 bit7 for counter7 0: in_counter 1: frequency_counter ... bit0 for counter0 0: in_counter 1: frequency_counter

- **DIO8217_input_counter_all_set**

Format: u32 status = DIO8217_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.

- **DIO8217_input_counter_all_read**

Format: u32 status = DIO8217_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

- **DIO8217_input_counter_set**

Format: u32 status = DIO8217_input_counter_set(u8 CardID,u8 index, u16 data)

Purpose: set input counter's data.

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	data to be set to counter

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

- **DIO8217_input_counter_read**

Format: u32 status = DIO8217_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 data

- **DIO8217_input_counter_mask_set**

Format: u32 status = DIO8217_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 continues to count the input (if the counter is enabled)

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

- **DIO8217_input_counter_mask_read**

Format: u32 status = DIO8217_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 continues to count the input (if the counter is enabled)

- **DIO8217_input_counter_control_set**

Format: u32 status = DIO8217_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)

- **DIO8217_input_counter_control_read**

Format: u32 status = DIO8217_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

You can configure the input counter as pure counter function or work with the timer as frequency counter.

Frequency Counter

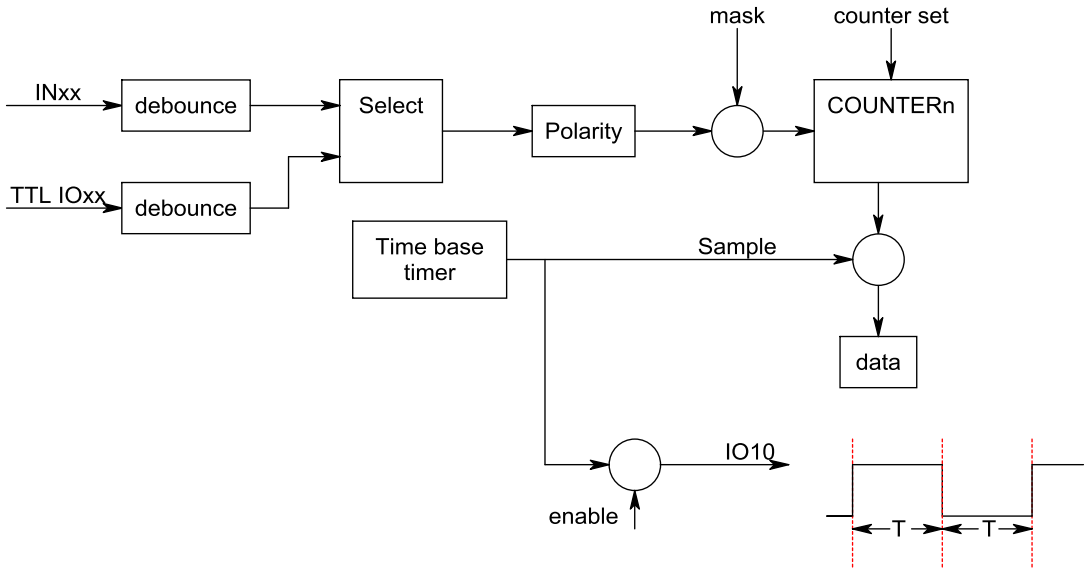


fig. .1 frequency counter function model

If the input counter work with the time base timer, you can count the frequency on the base of timer time constant (fig. 8.7.1 frequency counter function model).

You can configure the input counter as frequency counter and choose the signal source from TTL IO07~IO00 or from isolated input IN07~IN00 , mask off the unused channels by

DIO8217_input_counter_config_set( ) and read back for verification by

DIO8217_input_counter_config_read( )

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.

DIO8217_frequency_counter_enable( ), to stop the frequency counter by:

DIO8217_frequency_counter_disable( ).

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

DIO8217_input_counter_mask_set( ) and read back by

DIO8217_input_counter_mask_read( )

Owing to the time base clock is generated from PCI 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:

DIO8217_frequency_counter_test_enable( ), also use the same function, you can disable it. Refer fig. .1 frequency counter function model, you can see the wave form on a scope and measure the T(period) on an accurate instrument then adjust it by re-program the time base by **DIO8217_frequency_counter_enable( )**. After fine adjust, you can get an accurate time base then get accurate frequency counter at very low cost and effort.

- **DIO8217_frequency_counter_enable**

Format: u32 status = DIO8217_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 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

- **DIO8217_frequency_counter_disable**

Format: u32 status = DIO8217_frequency_counter_disable(u8 CardID)

Purpose: disable frequency counter

Parameters:

Input:

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

- **DIO8217_frequency_counter_test_enable**

Format: u32 status =DIO8217_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 DIO8217 card provides timer and inputs IN07 ~ IN00 or TTL IO07~IO00 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:

DIO8217_IRQ_process_link( ) then setup the IRQ mask for the interrupt by:

DIO8217_IRQ_mask_set( ) to select timer/counter, IN07~IN00 or TTL IO07~IO00 as source of IRQ.

DIO8217_IRQ_mask_read( )

After setup, you can enable the IRQ by:

DIO8217_IRQ_enable( ) and also you can disable IRQ by:

DIO8217_IRQ_disable( )

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

DIO8217_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.

- **DIO8217_IRQ_process_link**

Format: **u32 status = DIO8217_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

- **DIO8217_IRQ_mask_set**

Format: **u32 status = DIO8217_IRQ_mask_set(u8 CardID, u8 index , u8 Data)**

Purpose: Mask interrupt from timer/counter, IN07~IN00 or TTLIO07~IO00.

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY switch(**0x0-0xF**)
index	u8	0: input port0 1: TTL port 0 2: TC
Data	u8	If index = 0 then bit 0: irq source from IN00 bit 1: irq source from IN01 bit 2: irq source from IN02 bit 3: irq source from IN03 bit 4: irq source from IN04 bit 5: irq source from IN05 bit 6: irq source from IN06 bit 7: irq source from IN07 ElseIf 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 Else If index = 2 then bit 0: irq source from T/C bit 1: irq source from frequency counter time base If corresponding bit =0, mask off the interrupt

- **DIO8217_IRQ_mask_read**

Format: **u32 status = DIO8217_IRQ_mask_read(u8 CardID, u8 index , u8 *Data)**

Purpose: Read Mask interrupt from timer, IN07~IN00 or TTLIO07~IO00.

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY switch(**0x0-0xF**)
index	u8	0: input port0 1: TTL port 0 2: TC

Output:

Name	Type	Description
Data	u8	the same as above

- **DIO8217_IRQ_enable**

Format: **u32 status = DIO8217_IRQ_enable(u8 CardID, HANDLE *phEvent)**

Purpose: Enable interrupt from timer, IN07~IN00 or TTLIO07~IO00

Parameters:

Input:

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

Output:

Name	Type	Description
phEvent	HANDLE	event handle

- **DIO8217_IRQ_disable**

Format: **u32 status = DIO8217_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**)

- **DIO8217_IRQ_status_read**

Format: u32 status = DIO8217_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 IN00 bit01: 1, irq source from IN01 bit02: 1, irq source from IN02 bit03: 1, irq source from IN03 bit04: 1, irq source from IN04 bit05: 1, irq source from IN05 bit06: 1, irq source from IN06 bit07: 1, irq source from IN07 bit08: 1, irq source from TTL00 bit09: 1, irq source from TTL01 bit10: 1, irq source from TTL02 bit11: 1, irq source from TTL03 bit12: 1, irq source from TTL04 bit13: 1, irq source from TTL05 bit14: 1, irq source from TTL06 bit15: 1, irq source from TTL07 bit16: 1, irq source from T/C bit17: 1, irq source from frequency counter time base timer

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.	DIO8217_initial( )	DIO8217 initial
2.	DIO8217_close( )	DIO8217 close
3.	DIO8217_info( )	get OS. assigned address
4.	DIO8217_firmware_version_read( )	Read device firmware version
5.	DIO8217_debounce_time_set( )	setup input port debounce time
6.	DIO8217_debounce_time_read( )	read back input port debounce time
7.	DIO8217_inport_polarity_set( )	setup input port polarity
8.	DIO8217_inport_polarity_read( )	read back input port polarity
9.	DIO8217_outport_polarity_set( )	setup output port polarity
10.	DIO8217_outport_polarity_read( )	read back output port polarity
11.	DIO8217_inpoint_polarity_set( )	setup input point polarity
12.	DIO8217_inpoint_polarity_read( )	read back input point polarity
13.	DIO8217_outpoint_polarity_set( )	setup output point polarity
14.	DIO8217_outpoint_polarity_read( )	read back output point polarity
15.	DIO8217_outport_set( )	set output port (byte)
16.	DIO8217_outport_read( )	read port data (byte)
17.	DIO8217_inport_read( )	read input port data (byte)
18.	DIO8217_outpoint_set( )	set output point state (bit)
19.	DIO8217_outpoint_read( )	read back output point state (bit)
20.	DIO8217_inpoint_read( )	read input point state (bit)
21.	DIO8217_TTL_IO_config_set( )	setup TTL port I/O configuration
22.	DIO8217_TTL_IO_config_read( )	read back TTL port I/O configuration
23.	DIO8217_TTL_IO_port_polarity_set( )	setup TTL port polarity
24.	DIO8217_TTL_IO_port_polarity_read( )	read back TTL port polarity
25.	DIO8217_TTL_IO_point_polarity_set( )	setup TTL point polarity
26.	DIO8217_TTL_IO_point_polarity_read( )	read back TTL point polarity
27.	DIO8217_TTL_IO_debounce_time_set( )	setup TTL port input debounce time
28.	DIO8217_TTL_IO_debounce_time_read( )	read back TTL port input debounce time
29.	DIO8217_TTL_IO_enable( )	enable TTL IO function
30.	DIO8217_TTL_IO_disable( )	disable TTL IO function
31.	DIO8217_TTL_IO_port_set( )	set TTL IO port data
32.	DIO8217_TTL_IO_port_read( )	read TTL IO port data
33.	DIO8217_TTL_IO_point_set( )	set TTL IO point data
34.	DIO8217_TTL_IO_point_read( )	read TTL IO point data
35.	DIO8217_TC_timer_set( )	set timer mode function
36.	DIO8217_TC_timer_read( )	read timer mode function
37.	DIO8217_TC_counter_set( )	set counter mode function
38.	DIO8217_TC_counter_read( )	read counter mode function
39.	DIO8217_TC_up_down_counter_set( )	setup up_down_counter mode function
40.	DIO8217_TC_up_down_counter_read( )	read back up_down_counter mode function
41.	DIO8217_TC_PWM_set( )	set PWM mode function
42.	DIO8217_TC_PWM_read( )	read mode function
43.	DIO8217_TC_start( )	start Timer/counter/PWM

44.	DIO8217_TC_stop( )	stop Timer/counter/PWM
45.	DIO8217_TC_counter_value_read( )	read TC timer/counter working value on the fly
46.	DIO8217_TC_mode_set( )	set TC mode
47.	DIO8217_TC_mode_read( )	read TC mode
48.	DIO8217_TC_debounce_set( )	set timer debouncer
49.	DIO8217_TC_debounce_read( )	read timer debouncer
50.	DIO8217_TC_input_mode_set( )	set TC input mode
51.	DIO8217_TC_input_mode_read( )	read TC input mode
52.	DIO8217_TC_output_mode_set( )	set TC output mode
53.	DIO8217_TC_output_mode_read( )	read TC output mode
54.	DIO8217_TC_retrigger_mode_set( )	set counter retrigger mode
55.	DIO8217_TC_retrigger_mode_read( )	read counter retrigger mode
56.	DIO8217_TC_status_read( )	read TC status
57.	DIO8217_TC_set( )	set timer related registers
58.	DIO8217_TC_read( )	read timer related registers
59.	DIO8217_WDT_output_set( )	set WDT default output
60.	DIO8217_WDT_output_read( )	read WDT default output
61.	DIO8217_WDT_start( )	start WDT function
62.	DIO8217_WDT_reset( )	reset WDT function
63.	DIO8217_WDT_stop( )	stop WDT function
64.	DIO8217_WDT_read( )	read WDT related registers
65.	DIO8217_input_counter_config_set( )	setup the frequency counter
66.	DIO8217_input_counter_config_read( )	read back the setup of frequency counter
67.	DIO8217_input_counter_all_set( )	set all input counters' data
68.	DIO8217_input_counter_all_read( )	read all input counters' data
69.	DIO8217_input_counter_set( )	set one input counters' datum
70.	DIO8217_input_counter_read( )	read one input counters' datum
71.	DIO8217_input_counter_mask_set( )	set input counters' operation mask
72.	DIO8217_input_counter_mask_read( )	read back input counters' operation mask
73.	DIO8217_input_counter_control_set( )	set input counter control
74.	DIO8217_input_counter_control_read( )	read back input counter control status
75.	DIO8217_frequency_counter_enable( )	enable operation of frequency counter
76.	DIO8217_frequency_counter_disable( )	disable operation of frequency counter
77.	DIO8217_frequency_counter_test_enable( )	check the time base accuracy
78.	DIO8217_IRQ_process_link( )	link interrupt service routine to driver
79.	DIO8217_IRQ_mask_set( )	set interrupt mask
80.	DIO8217_IRQ_mask_read( )	read interrupt mask
81.	DIO8217_IRQ_enable( )	enable interrupt function
82.	DIO8217_IRQ_disable( )	disable interrupt function
83.	DIO8217_IRQ_status_read( )	read back IRQ status

10. DIO8217 Error codes summary

10.1 DIO8217 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 DIO8217 card on the system.
102	JSDRV_DUPLICATE_ID	DIO8217 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