AIO6328/6328A

PCI-104
Analog I/O and digital I/O Card

Software Manual (V1.2)

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	Driver version → wdf6328.sys V1.0, AIO6328.dll V1.0
1.1	Driver version → wdf6328.sys V2.0, AIO6328.dll V2.0
	add new function ***AIO6328_AD_all_read*** to read all channels in one command
1.2	Add new function ***AIO6328_AD_integral_start, AIO6328_AD_integral_stop, AIO6328_AD_integral_all_read***

Contents

1. How to install the software of AIO6328

1.1 Install the PCI-104 driver

The PCI-104 card is a plug and play card, once you add a new card on the window system will detect while it is booting. Please follow the following steps to install your new card.

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

1. Make sure the power is off
2. Plug in the interface card
3. Power on
4. A hardware install wizard will appear and tell you it finds a new PCI card
5. Do not response to the wizard, just Install the file (..\AIO6328_A\Software\Win2K_up\ or if you download from website please execute the file AIO6328_Install.exe to get the file)
6. After installation, power off
7. Power on, it's ready to use

Note: AIO6328, AIO6328A use the same driver and dll.

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

http://automation.com.tw/

to download the complementary documents.

2. Where to find the file you need

Win2K/XP/7 and up

The directory will be located at

.. \ JS Automation \AIO6328\API (header files and lib files for VB,VC,BCB,C#)

.. \ JS Automation \AIO6328\Driver (backup copy of AIO6328 drivers)

.. \ JS Automation \AIO6328\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 AIO6328 software

AIO6328 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 AIO6328 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 AIO6328 functions within Windows' operation system environment.

3.1 What you need to get started

To set up and use your AIO6328 software, you need the following:

- AIO6328 software
- AIO6328 hardware

  Main board

  Wiring board (Option)

3.2 Software programming choices

You have several options to choose from when you are programming AIO6328 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 AIO6328 software.

4. AIO6328 Language support

The AIO6328 software library is a DLL used with Win2K/XP/7 and up. You can use these DLL with any Windows integrating development environment that can call Windows DLLs.

4.1 Building applications with the AIO6328 software library

The AIO6328 function reference topic contains general information about building AIO6328 applications, describes the nature of the AIO6328 files used in building AIO6328 applications, and explains the basics of making applications using the following tools:

Applications tools

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

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

4.2 AIO6328 Windows libraries

The AIO6328 for Windows function library is a DLL called **AIO6328.dll**. Since a DLL is used, AIO6328 functions are not linked into the executable files of applications. Only the information about the AIO6328 functions in the AIO6328 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 AIO6328 functions in AIO6328.dll.

Header Files and Import Libraries for Different Development Environments		
Language	**Header File**	**Import Library**
Microsoft Visual C/C++	AIO6328.h	AIO6328VC.lib
Borland C/C++	AIO6328.h	AIO6328BC.lib
Microsoft Visual C#	AIO6328.cs	
Microsoft Visual Basic	AIO6328.bas	
Microsoft VB.net	AIO6328.vb	

Table 1

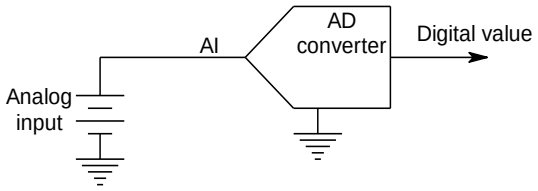
5. Basic concepts of analog I/O control

Analog input is used for quantizing the real world signals to digital values and analog output is vice versa for converting digital values to real world analog values.

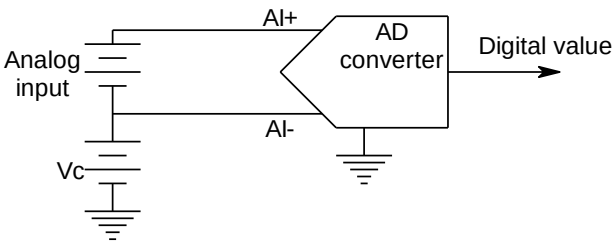
Types of analog input

We can classify the input by type as: current type, voltage type. But you can easily convert the current to voltage by insert a resistor. Generally, voltage type analog input can satisfy most applications.

On the other hand, the input type can also classify by the detection method. Single end input detects the input voltage by reference to a common point and differential input detects input voltage to the difference of the input.



Single end type



Differential type

From the above diagram, you can see the differential type must have a common connection between measured signal and the AD converter. The under measure voltage is riding on a common voltage Vc.

The AIO6328 card can measure the analog voltage without damage at Vc under –10V ~ +10V. Out of this common voltage range, the conversion maybe error or the on board chip will be damaged.

The AIO6328 has 12 bit (AIO6328A 16 bit) resolution and can accept 0 ~ 5V, 0 ~ 10V, -5V ~ +5V and –10V ~ +10V range. The adjacent channel can be programmed as differential or leave it as single end input.

Types of analog output

There are current type and voltage type in general. Current type mostly use 0~20ma or 4~20ma for remote analog signal transmission. Voltage type is the major on the market.

AIO6328 provide voltage type output and 12 bit amplitude to 0V ~ +10V range and an extra polarity bit to define the voltage polarity, the possible output will be –10V ~+10V.

6. Basic concepts of digital I/O control

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

Types of I/O calssified by isolation

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

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

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

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

Types of Output calssified by driver device

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

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

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

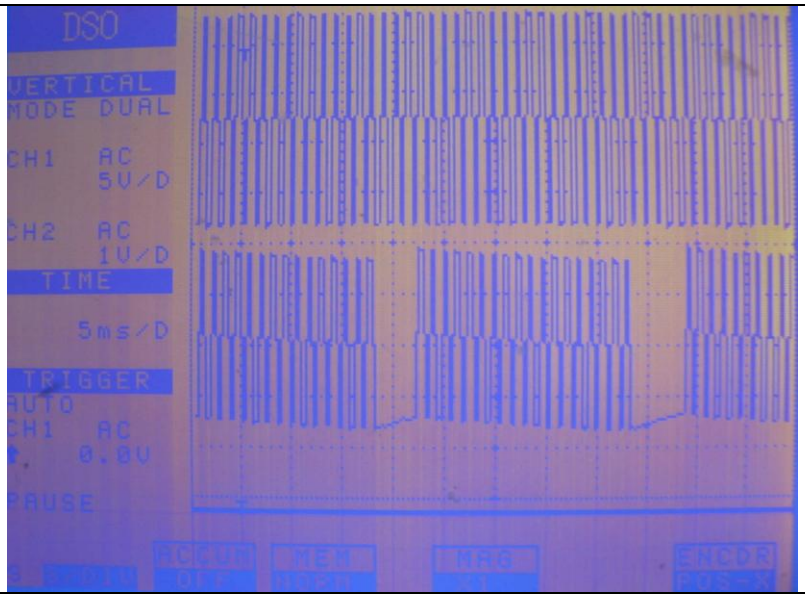
SCR (or triac) is seldom direct connect to digital output, but his relative SSR is the most often selection. In fact, SSR is a compact package of trigger circuit and triac. You can choose zero cross trigger (output command only turn on the output at power phase near zero to eliminate surge) or direct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

Input debounce

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

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

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



<- Input frequency at 835Hz

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

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

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

AIO6328/AIO6328A has hardware build-in digital filters to drop out the <10ms (100Hz), <5ms(200Hz) and <1ms (1KHz) signal by software programmable command.

Input interrupt

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

Read back of Output status

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

7. Software overview

These topics describe the features and functionality of the AIO6328 boards and briefly describes the AIO6328 dll functions.

Owing to the PCI-104 use stack method to connect the bus, the lower layer stack will be closest to the PCI signal. To compensate the distance, the AIO6328 requires the CardID setting and clock selection according to the physical stacking layer, i.e. the closest one CardID=0 and CLK also select 0, the next to the bottom layer will be set jumper to CardID=1 and CLK also to 1….

7.1 Initialization and close

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

AIO6328_initial( ) will do.

Once you want to close your application, call

AIO6328_close( ) to release all the resource.

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

AIO6328_info( ) to get the address and CardType

7.2 Analog I/O function

The AIO6328 card has 2 channels of DA converters, depending on the model option, it can be 12 bits or 16 bits in voltage amplitude and one polarity bit to decide positive or negative voltage. You can command the DA output voltage and polarity by

AIO6328_DA_set( ) and read back the setting data by

AIO6328_DA_read( )

For the AD converters, totally 8 analog switch multiplexed channels with programmable single end or differential mode and programmable voltage range.

To configure as single end or differential mode input by

AIO6328_AD_config_set( ) and read back the configuration data by

AIO6328_AD_config_read( ).

To setup the analog input range by

AIO6328_AD_range_set( ) and read back the range setting data by

AIO6328_AD_range_read( ).

The AD previous conversion data will be received at the same time as the current conversion command send out.

AIO6328_AD_start( ) will start a new conversion and receive the last conversion data and you must use ***AIO6328_AD_read( )*** to read the conversion data, ***AIO6328_AD_all_read*** to read all channels.

If your application needs long time integration to filter out the environment and power line noise, special functions allow you to access data for 1sample per second up to 10 sample per second. Use

AIO6328_AD_integral_start( ) to configure the sample time you need, tostop the integration by ***AIO6328_AD_integral_stop( )***
and get data by
AIO6328_AD_integral_all_read( ).

7.3 Digital I/O function

Before using a input port, if you already know the maximum response time of the input signal you can setup the debounce time to filter out the undesired noise signal and get a noise-free signal. If you do not know the exact response, please use the conservative setting i.e. 100Hz (sample rate 200Hz) is a common choice.

Use ***AIO6328_debounce_time_set ( )*** to configure the debounce time.

AIO6328_debounce_time_read ( ) to read back the configuration data.

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

AIO6328_port_set ( ) to output byte data to output port,

AIO6328_port_read ( ) to read a byte data from I/O port,

AIO6328_point_set ( ) to set output bit,

AIO6328_point_read ( ) to read I/O bit.

7.4 Timer / Counter function

The on card timer is a 32 bits counter based on 1MHZ time base. To configure the working mode use ***AIO6328_timer_set ( )***

Read timer on the fly

AIO6328_timer_read( )

To start/stop the operation by:

AIO6328_timer_start ( )

AIO6328_timer_stop ( )

To read or load dedicated timer/counter registers for advanced application, use

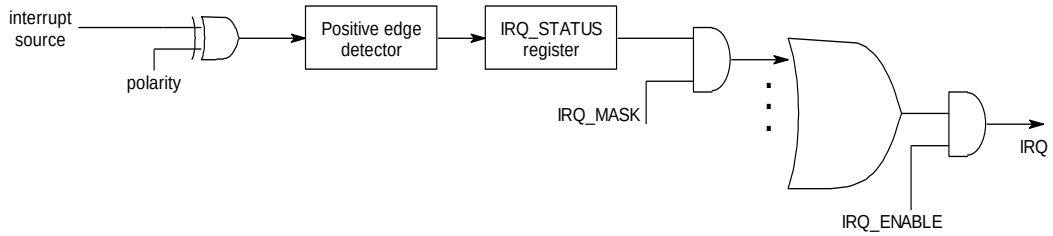
AIO6328_TC_set ( ) set TC dedicated registers

AIO6328_TC_read ( ) read TC dedicated registers

7.5 Interrupt function

Sometimes you want your application to take care of the I/O while special event occurs, interrupt function is the right choice. The AIO6328 card can generate interrupt from DIO block, AD block and Timer block.

The AIO6328 card interrupt model is as follows:



For digital input, the interrupt source is IN00~IN07, the physical input is changed polarity by POLARITY_SET, then the on board hardware detect the positive edge transition to trigger the IRQ_STATUS register, it is irrelevant to the IRQ_MASK. If the IRQ_MASK is set to 1 and IRQ_ENABLE are also set, the interrupt will generate. By this model, you can see that you can check the fast changing input by IRQ_STATUS without using interrupt.

To configure the IN00~IN07 interrupt polarity, use

AIO6328_IRQ_polarity_set( ) and read back the setting data by

AIO6328_IRQ_polarity_read( ).

Next, you should enable / disable the hardware of the interrupt source by,

AIO6328_IRQ_mask_set ( )

AIO6328_IRQ_mask_read ( ) to read back the IRQ mask status.

After all is prepared, tell the driver your interrupt service routine by

AIO6328_IRQ_process_link ( )

To enable the IRQ function,

AIO6328_IRQ_enable ( ) to start waiting the interrupt.

If you do not use interrupt any more and you will close your application program, be sure to use

AIO6328_IRQ_disable ( ) to release the resource.

In interrupt service routine, if you want to know the interrupt status, use

AIO6328_IRQ_status_read ( ) to identify the source of interrupt. If you do not use the interrupt function (IRQ disabled) and want to scan the interrupt generated only, this command can also use to verify the external trigger status.

7.6 Software key function

Since AIO6328 is a general purpose card, anyone who can buy from JS automation Corp. or her distributors. Your program is the fruit of your intelligence, un-authorized copy maybe prevent by the security function enabled.

You can use

AIO6328_password_set ( ) to set password and start the security function.

AIO6328_password_change ( ) to change it.

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

AIO6328_password_clear ( ) will reset to the virgin state.

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

AIO6328_security_unlock ( ) unlock the security.

You can also use

AIO6328_security_status_read ( ) to check the current status of security.

7.7 Error conditions

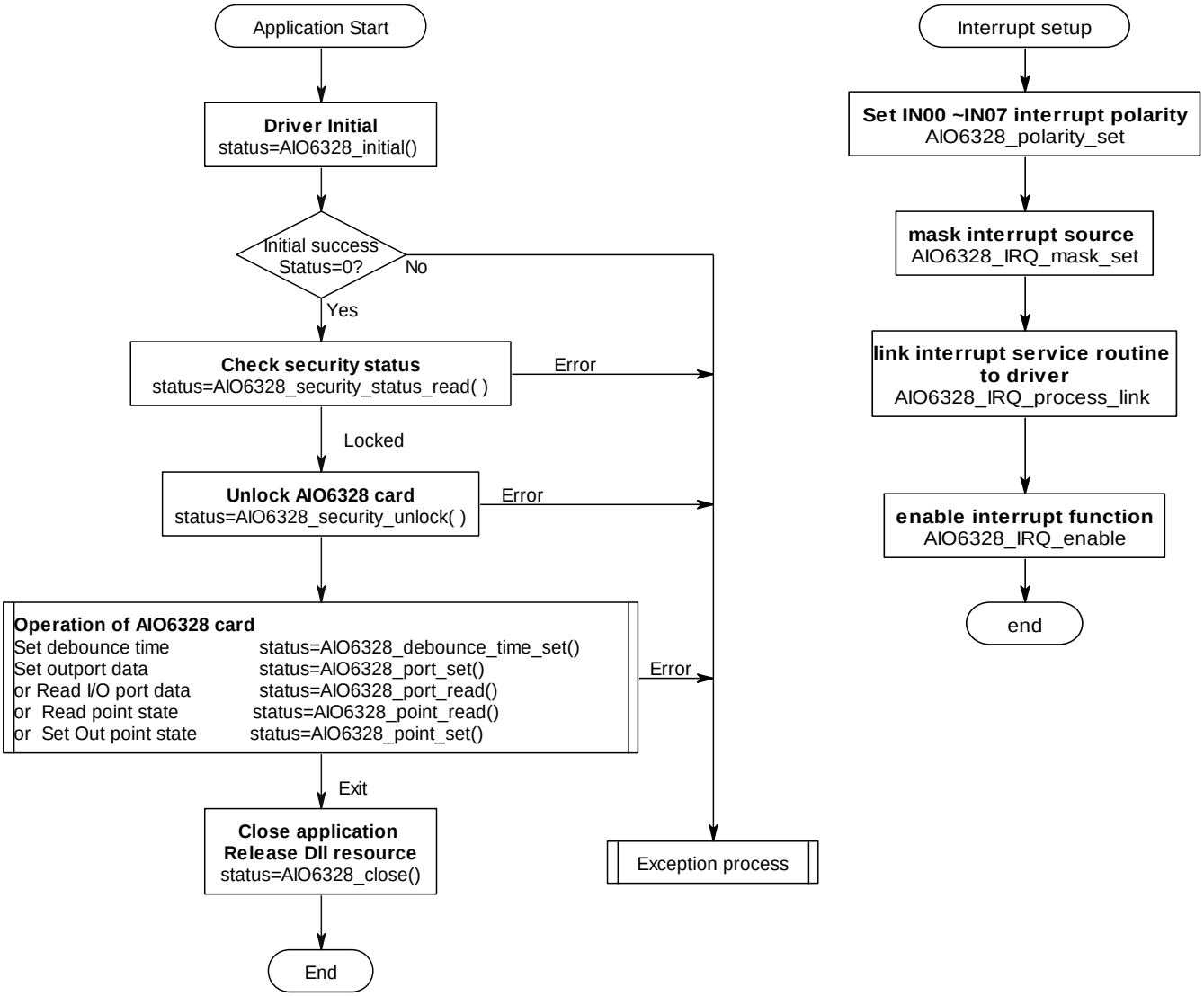
AIO6328 cards minimize error conditions. There are three possible fatal failure modes:

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

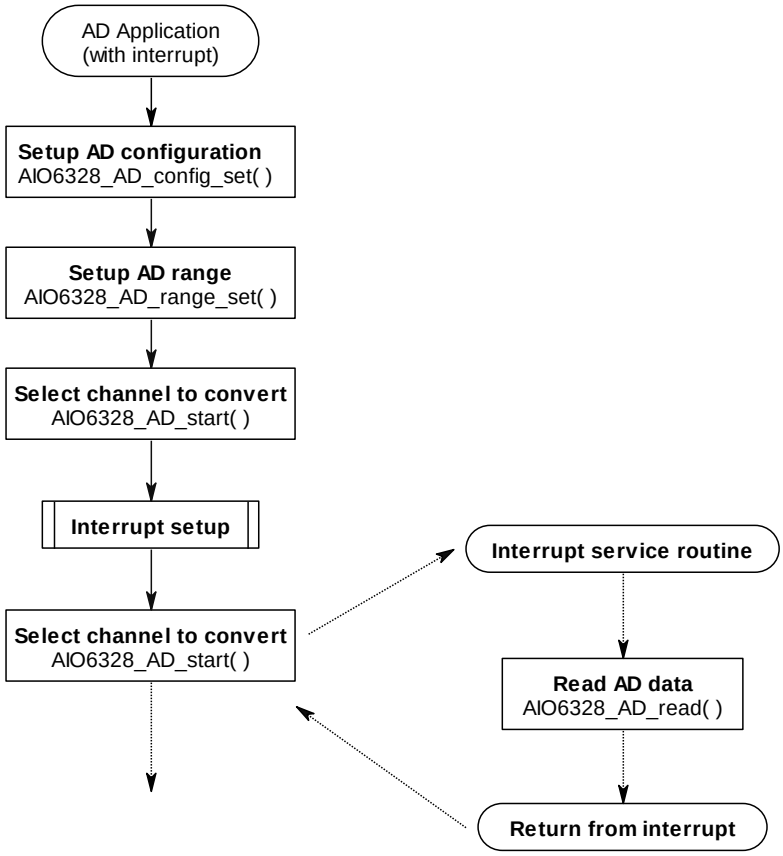
These error types may indicate an internal hardware problem on the board. Error Codes contains a detailed listing of the error status returned by AIO6328 functions.

8. Flow chart of application implementation

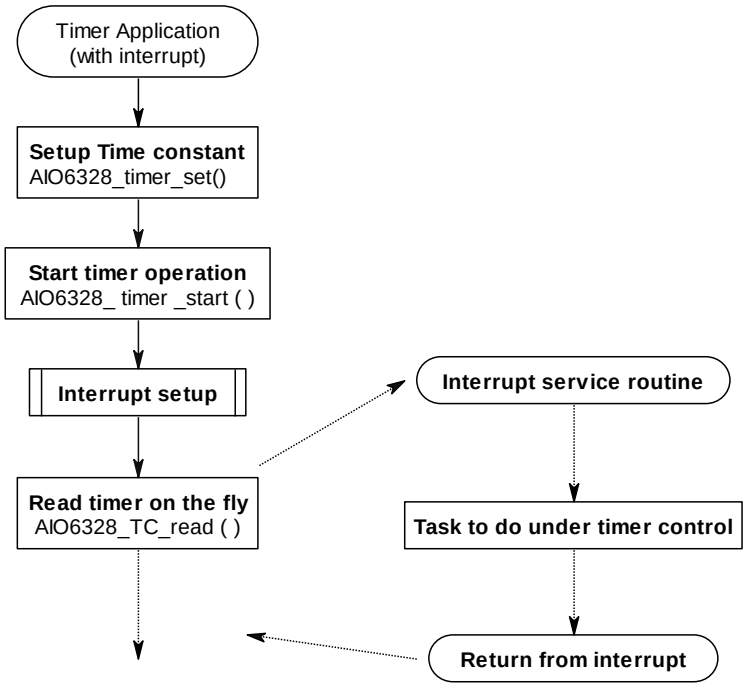
8.1 AIO6328 Flow chart of application implementation



8.2 AIO6328 Flow chart of AD application



8.3 AIO6328 Flow chart of Timer application



9. Function reference

9.1 Function format

Every AIO6328 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 AIO6328 function is the parameter **CardID** which is located the driver of AIO6328 board you want to use those given operation. The **CardID** is assigned by DIP/ROTARY SW. You can utilize multiple devices with different card CardID within one application; to do so, simply pass the appropriate **CardID** to each function.

Note: **CardID** is set by jumper (**0x0-0x3**)

These topics contain detailed descriptions of each AIO6328 function. The functions are arranged alphabetically by function name. Refer to AIO6328 Function Reference for additional information.

9.2 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	-1.797685123862315E+308 to 1.797685123862315E+308	double	Double (for example: voltage Number)	Double

Table 2

9.3 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 AIO6328 API. Read the following sections that apply to your programming language.

Note: Be sure to include the declaration functions of AIO6328 prototypes by including the appropriate AIO6328 header file in your source code. Refer to Building Applications with the AIO6328 Software Library for the header file appropriate to your compiler.

9.3.1 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 = AIO6328_ port_read (u8 CardID, u8 port, u8*data);

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

```
u8 CardID=0, port=0;      //assume CardId=0 and port=0
u8 data,
u32 Status;
Status = AIO6328_port_read (CardID, port, &data);
```

9.3.2 Visual basic

The file AIO6328.bas contains definitions for constants required for obtaining AIO Card information and declared functions and variable as global variables. You should use these constants symbols in the AIO6328.bas, do not use the numerical values.

In Visual Basic, you can add the entire AIO6328.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 AIO6328.bas file for your project in Visual Basic 4.0, go to the **File** menu and select the **Add File... option**. Select AIO6328.bas, which is browsed in the AIO6328 \ API directory. Then, select **Open** to add the file to the project.

To add the AIO6328.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** AIO6328.bas which is under the AIO6328 \ API directory. Then, select **Open** to add the file to the project.

9.3.3 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 AIO6328BC.lib AIO6328.dll

Then add the **AIO6328BC.lib** to your project and add

#include "AIO6328.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 = AIO6328_port_read (u8 CardID, u8 port, u8*data);

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

```
u8 CardID=0, port=0;      //assume CardId=0 and port=0
u8 data;
u32 Status;
Status = AIO6328_port_read (CardID, port, &data);
```

9.4 AIO6328 Functions

Note:

Owing to the PCI-104 use stack method to connect the bus, the lower layer stack will be closest to the PCI signal. To compensate the distance, the AIO6328 requires clock selection according to the physical stacking layer, i.e. the closest one CLK also select 0, the next to the bottom layer will be set CLK jumper to 1….

Initialization and close

- **AIO6328_initial**

Format : u32 status =AIO6328_initial (void)

Purpose: Initial the AIO6328 resource when start the Windows applications**.**

- **AIO6328_close**

Format : u32 status =AIO6328_close (void);

Purpose: Release the AIO6328 resource when close the Windows applications.

- **AIO6328_info**

Format : u32 status =AIO6328_info(u8 CardID, u8 *CardType, u16 *DIO_address,u16 *TC_address);

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

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY SW

Output:

Name	Type	Description
CardType	u8	0: AIO6328 (12 bit version) 1: AIO6328A (16 bit version)
DIO_address	u16	physical I/O address assigned to DIO block by OS
TC_address	u16	physical I/O address assigned to ADC and timer block by OS

Analog I/O function

- **AIO6328_DA_set**

Format : u32 status = AIO6328_DA_set(u8 CardID,u8 channel, u16 data,u16 sign)

Purpose: DA output

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by jumper setting
channel	u8	0: DA0 channel 1: DA1 channel
data	u16	0~0xfff (AIO6328)、0~0xffff (AIO6328A) for analog output amplitude 0V ~ 10V 0V output is 0x0, 10V output is 0xfff (AIO6328) 0xffff (AIO6328A)
sign	u16	0: output is positive voltage (default after hardware reset) 1: output is negative voltage

- **AIO6328_DA_read**

Format : u32 status = AIO6328_DA_read(u8 CardID,u8 channel, u16 *data, u16 * sign)

Purpose: read back DA data

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by jumper setting
channel	u8	0: DA0 channel 1: DA1 channel

Output:

Name	Type	Description
data	u16	0~0xfff (AIO6328)、0~0xffff (AIO6328A) for analog output amplitude 0V ~ 10V 0V output is 0x0, 10V output is 0xfff (AIO6328) 0xffff (AIO6328A)
sign	u16	0: output is positive voltage 1: output is negative voltage

- **AIO6328_AD_config_set**

Format : u32 status = AIO6328_AD_config_set (u8 CardID, AD_config *AD_config)

Purpose: configure each port as differential or single end.

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by jumper setting

AD_config	AD_config	struct _AD_config{
		u8 ch01_config,
		u8 ch23_config,
		u8 ch45_ config,
		u8 ch67_ config
		}
		// ch01: AI0~AI1
		// ch23: AI2~AI3
		// ch45: AI4~AI5
		// ch67: AI6~AI7
		// chNM_config:
		//0: chNM is paired differential and polarity is normal
		//1: chNM is paired differential and polarity is inverse
		//2: invalid
		//3: chNM is single end
		For example, if you will configure
		channel 0,1 as differential with polarity normal,
		channel 2,3 as single end
		channel 4,5 , channel 6,7 as differential with inverse polarity then struct AD_config is {0,3,1,1}

Note:

AD input differential connection please refer to chap 5 Basic concepts of analog I/O control.

- **AIO6328_AD_config_read**

Format : **u32 status = AIO6328_ AD_config_read (u8 CardID, AD_config *AD_config)**

Purpose: configure each channel as differential or single end.

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by jumper setting

Output:

Name	Type	Description

AD_config	AD_config	struct _AD_config{
		u8 ch01_config,
		u8 ch23_config,
		u8 ch45_ config,
		u8 ch67_ config
		}

- **AIO6328_AD_range_set**

Format : **u32 status = AIO6328_AD_range_set(u8 CardID, AD_Range *AD_range)**

Purpose: set up each group conversion range

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by jumper setting

AD_range	AD_range	struct _AD_Range{
		u8 ch0_ range,
		u8 ch1_ range,
		u8 ch2_ range,
		u8 ch3_ range
		u8 ch4_ range,
		u8 ch5_ range,
		u8 ch6_ range,
		u8 ch7_ range
		}
		// chN_range
		//0: +-5V
		//1: 0-5V
		//2: +-10V
		//3: 0-10V

Note: If the even channel is configured as differential input, the next odd number channel member is invalid.

For example ch0 is configured as differential input by **AIO6328_AD_config_set,** then the AD_Range.ch1_range is of no use.

- **AIO6328_AD_range_read**

Format : **u32 status = AIO6328_AD_range_read(u8 CardID, AD_Range *AD_range)**

Purpose: read back each group conversion range setting

Input:

Name	Type	Description
CardID	u8	assigned by jumper setting

Output:

Name	Type	Description

AD_range	AD_range	struct _AD_Range{
		u8 ch0_ range,
		u8 ch1_ range,
		u8 ch2_ range,
		u8 ch3_ range
		u8 ch4_ range,
		u8 ch5_ range,
		u8 ch6_ range,
		u8 ch7_ range
		}
		// chN_range
		//0: +-5V
		//1: 0-5V
		//2: +-10V
		//3: 0-10V

- **AIO6328_AD_start**

Format : **u32 status = AIO6328_AD_start(u8 CardID,u8 channel)**

Purpose: start AD conversion of designated channel and trigger to receive the previous conversion data.

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by jumper setting
channel	u8	0~7, channel no.

- **AIO6328_AD_read**

Format : **u32 status = AIO6328_AD_read(u8 CardID,u16 *data)**

Purpose: read AD conversion data. Please refer the Note of this command.

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by jumper setting

Output:

Name	Type	Description
data	u16	0~0xfff (AIO6328), 0~0xffff (AIO6328A). AD converted data

Note:

1. **AIO6328_AD_start** will select the channel and trigger to receive the conversion data of previous command for the AIO6328_AD_read( ).
2. Before read back the data by **AIO6328_AD_read,** you can check the status by **AIO6328_IRQ_status_read** (no matter you use interrupt or not) to confirm the AD data is ready.
3. The AD conversion time frame is as

```
AIO6328_AD_start(CardID,Ch0)      //start conversion of channel 0 and trigger to receive the
                                  //previous converted data
AIO6328_AD_read(CardID,*data0)    //unknown data
AIO6328_AD_start(CardID,Ch1)      //start conversion of channel 1 and trigger to receive the
                                  //previous converted data (i.e. ch0 data)
AIO6328_AD_read(CardID,*data0)    //read back the converted data of ch0
AIO6328_AD_start(CardID,Ch2)      //start conversion of channel 2 and trigger to receive the
                                  //previous converted data (i.e. ch1 data)
AIO6328_AD_read(CardID,*data1)    //read back the converted data of ch1
```

- **AIO6328_AD_all_read**

Format : **u32 status = AIO6328_AD_all_read(u8 CardID,u16 data[8])**

Purpose: read all AD conversion data.

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by jumper setting

Output:

Name	Type	Description
data[8]	u16	0~0xfff (AIO6328), 0~0xffff (AIO6328A). AD converted data

Note:

To read all channels

1. Set up start channel at **channel 0** by **AIO6328_AD_start.**
2. Read all channels by **AIO6328_AD_all_read**.

- **AIO6328_AD_integral_start**

Format : **u32 status = AIO6328_AD_integral_start(u8 CardID,u8 mode)**

Purpose: start AD conversion with integral constant.

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by jumper setting
mode	u8	0: immediately access, no integration 1: integration time 100ms 2: integration time 200ms 3: integration time 300ms 4: integration time 400ms 5: integration time 500ms 6: integration time 600ms 7: integration time 700ms 8: integration time 800ms 9: integration time 900ms 10: integration time 1s

Note: Owing to the integration function use the interrupt function to trigger, using the ***AIO6328_AD_integral_start*** can not also use interrupt function of AD.

- **AIO6328_AD_integral_stop**

Format : **u32 status = AIO6328_AD_integral_stop(u8 CardID)**

Purpose: stop AD integral conversion.

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by jumper setting

- **AIO6328_AD_integral_all_read**

Format : **u32 status = AIO6328_AD_integral_all_read(u8 CardID,u16 data[8])**

Purpose: read all integral result of AD conversion data.

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by jumper setting

Output:

Name	Type	Description
data[8]	u16	For uni-polar input (0~5V or 0~10V) AD converted data will be: 0~0xfff (AIO6328), 0~0xffff (AIO6328A). For bi-polar input (-5V~5V or –10V~10V) AD converted data will be in 2's complement form, the positive will be in 0~0x7ff (AIO6328), 0~0x7fff (AIO6328A). the negative will be in 0x800~0xfff (AIO6328), 0x8000~0xffff (AIO6328A).

Note:

To read all channels in integral

1. Start integral mode by **AIO6328_AD_integral_start.** (AD interrupt is not allowed to use)
2. Read all channels by **AIO6328_AD_integral_all_read**.
3. Stop AD integration function by **AIO6328_AD_integral_stop.**

Digital I/O function

- **AIO6328_debounce_time_set**

Format : u32 status = AIO6328_debounce_time_set (u8 CardID , u8 debounce_time)

Purpose: Set the input port debounce time

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by jumper setting
debounce_time	u8	Debounce time selection: 0: no debounce 1: filter out duration less than 10ms (default) 2: filter out duration less than 5ms 3: filter out duration less than 1ms

- **AIO6328_debounce_time_read**

Format : u32 status = AIO6328_debounce_time_read (u8 CardID , u8 * debounce_time)

Purpose: Read back the input port debounce time configuration

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by jumper setting

Output:

Name	Type	Description
debounce_time	u8	Debounce time selection: 0: no debounce 1: filter out duration less than 10ms (default) 2: filter out duration less than 5ms 3: filter out duration less than 1ms

- **AIO6328_port_set**

Format : **u32 status = AIO6328_port_set (u8 CardID, u8 data)**

Purpose: Sets the output data.

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by jumper setting
data	u8	bitmap of output values ,0x0 ~0xff bit0: 0: OUT00 break 1: OUT00 make … bit7: 0: OUT07 break 1: OUT07 make

- **AIO6328_port_read**

Format : **u32 status = AIO6328_port_read (u8 CardID , u8 port , u8 *data)**

Purpose: Read the register of output port or input status of the input port

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by jumper setting
port	u8	port number 0: INPORT 1: OUTPORT

Output:

Name	Type	Description
data	u8	I/O data For INPORT bit0: IN00 .. bit7:IN07 For OUTPORT bit0: OUT00 .. bit7: OUT07

Note:

If the port to be read is output, the data read back will be the output register data not physical output data.

- **AIO6328_point_set**

Format : **u32 status =AIO6328_point_set (u8 CardID, u8 point, u8 state)**

Purpose: Sets the bit data of output port.

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by jumper setting
point	u8	point number 0~7 for OUT00~OUT07
state	u8	state of output point 0: break 1: make

- **AIO6328_point_read**

Format : **u32 status =AIO6328_point_read (u8 CardID, u8 port , u8 point, u8 *state)**

Purpose: Read the state of the input points or output register.

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by jumper setting
port	u8	0: input port for IN00-IN07 1: output port for OUT00-OUT07
point	u8	point number of input 0~7 for IN00~IN07 or 0~7 for OUT00~OUT07

Output:

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

Note:

If the point to be read is output, the data read back will be the output register data not physical output data.

Timer function

- **AIO6328_timer_set**

Format : u32 status = AIO6328_timer_set (u8 CardID, u32 Timer_constant)

Purpose: To setup timer operation mode or update timer

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by jumper setting
Timer_constant	u32	Timer constant based on 1us clock

Note:

1. Time constant is based on 1us clock, period T= (time_constant +1) * 1us
2. If you also enable the timer interrupt, the period T must at least larger than the system interrupt response time else the system will be hanged by excess interrupts.

- **AIO6328_timer_read**

Format : u32 status = AIO6328_timer_read (u8 CardID, u32 * Timer_constant)

Purpose: To read timer value on the fly

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY SW

Output:

Name	Type	Description
Timer_constant	u32	timer value on the fly

- **AIO6328_timer_start**

Format : u32 status = AIO6328_timer_start (u8 CardID)

Purpose: To start timer operation mode

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by Jumper setting

- **AIO6328_timer_stop**

Format : **u32 status = AIO6328_timer_stop (u8 CardID)**

Purpose: To stop timer operation mode

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by Jumper setting

- **AIO6328_TC_set**

Format : **u32 status=AIO6328_TC_set (u8 CardID,u8 index,u32 data)**

Purpose: To load data to timer related registers

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by jumper setting
index	u8	0: TC_CONTROL 1: PRELOAD 2: TIMER
data	u32	For TC_CONTROL 0: stop timer operation 1: timer run For PRELOAD or TIMER Data is the constant to be load

Note:

PRELOAD is the register for timer to re-load, the value will be valid while timer count to zero and reload the data.

- **AIO6328_TC_read**

Format : **u32 status=AIO6328_TC_read (u8 CardID,u8 index,u32 *data)**

Purpose: To read data from timer related registers

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by jumper setting
index	u8	0: TC_CONTROL 1: PRELOAD 2: TIMER

Output:

Name	Type	Description
data	u32	Data read back

Note: Meaning of setting or return value of different index

index	register	value	meaning
0	TC_CONTROL	0~1	0:timer stops operation 1: timer runs
1	PRELOAD	0~0xffffffff	timer preload value
2	TIMER	0~0xffffffff	Timer value on the fly

Note:

1. For example, you want to watch the counting on the fly, use
   AIO6328_TC_read (CardID, TIMER,*data) //CardID as you assign, TIMER =2
   To read back the timer value.

Interrupt function

- **AIO6328_IRQ_polarity_set**

Format : u32 status = AIO6328_IRQ_polarity_set (u8 CardID, u8 polarity)

Purpose: Set the IN00~IN07 interrupt polarity

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by jumper setting
polarity	u8	Data to be set, 0x0 ~ 0xff bit0: IN00 0:normal (default) 1:invert … bit7: IN07 0:normal (default) 1:invert

- **AIO6328_IRQ_polarity_read**

Format : u32 status = AIO6328_IRQ_polarity_read (u8 CardID, u8 *polarity)

Purpose: Read back the IN00~IN07 interrupt polarity

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by jumper setting

Output:

Name	Type	Description
polarity	u8	Data read back, 0x0 ~ 0xff bit0: IN00 0:normal (default) 1:invert … bit7: IN07 0:normal (default) 1:invert

- **AIO6328_IRQ_mask_set**

Format : **u32 status = AIO6328_IRQ_mask_set (u8 CardID, u8 source, u8 mask)**

Purpose: Select interrupt source

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by jumper setting
source	u8	0: digital I/O block 1: AD block 2: timer block
mask	u8	**Digital IO block:** B0=0, disable IN00 irq B0=1, IN00 can generate irq … B7=0, disable IN07 irq B7=1, IN07 can generate irq **AD block:** B0=1 means AD end of conversion can generate interrupt B0=0 AD will not generate interrupt while end of conversion **Timer block:** B0=1 means timer count up can generate interrupt B0=0 timer will not generate interrupt while time up

- **AIO6328_IRQ_mask_read**

Format : **u32 status = AIO6328_IRQ_mask_read (u8 CardID, u8 source, u8 *mask)**

Purpose: Select interrupt source

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by jumper setting
source	u8	0: digital I/O block 1: AD block 2: timer block

Output:

Name	Type	Description
mask	u8	**Digital IO block:** B0=0, disable IN00 irq B0=1, IN00 can generate irq … B7=0, disable IN07 irq B7=1, IN07 can generate irq **AD block:** B0=1 means AD end of conversion can generate interrupt B0=0 AD will not generate interrupt while end of conversion **Timer block:** B0=1 means timer count up can generate interrupt B0=0 timer will not generate interrupt while time up

- **AIO6328_IRQ_process_link**

Format : **u32 status = AIO6328_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 jumper setting
callbackAddr	void	callback address of service routine

- **AIO6328_IRQ_enable**

Format : **u32 status = AIO6328_IRQ_enable (u8 CardID, HANDLE *phEvent)**

Purpose: Enable interrupt from unmasked source

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by jumper setting

Output:

Name	Type	Description
phEvent	HANDLE	event handle

- **AIO6328_IRQ_disable**

Format : **u32 status = AIO6328_IRQ_disable (u8 CardID)**

Purpose: Disable interrupt from unmasked source

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by jumper setting

- **AIO6328_IRQ_status_read**

Format : **u32 status = AIO6328_IRQ_status_read (u8 CardID, u8 source, u8 *Event_Status)**

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

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by jumper setting
source	u8	0: digital I/O block 1: AD block 2: timer block

Output:

Name	Type	Description
Event_Status	u8	**Digital IO block:** B0=1, port0 bit0 input request irq … B7=1, port0 bit7 input request irq **AD block:** B0=1, AD end of conversion and data is ready B0=0, Ad is under conversion **Timer block:** B0=1 means timer count up occurred. B0=0 means timer not count up.

Note:

1.AIO6328_IRQ_status_read( ) function can be used in the user's interrupt service routine to identify the irq source if multiple source is allowed.

2.If you do not use interrupt function, you can still use AIO6328_IRQ_status_read( ) to catch the fast changing input status.

3.Before return from AIO6328_IRQ_status_read( ), the status hardware will be cleared.

4. status read back will also clear the on board status register.

5. the status will reflect the on board digital input or timer count up status, all these status are irrelevant to the Interrupt mask register.

Software key function

- **AIO6328_password_set**

Format : **u32 status = AIO6328_password_set (u8 CardID,u16 password[5]);**

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

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by jumper setting
password[5]	u16	Password, 5 words

Note on password:

If the password is all "0", the security function is disabled.

- **AIO6328_password_change**

Format : **u32 status = AIO6328_password_change (u8 CardID,u16 Oldpassword[5], u16 password[5]);**

Purpose: To replace old password with new password.

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by jumper setting
Oldpassword [5]	u16	The previous password
password[5]	u16	The new password to be set

- **AIO6328_password_clear**

Format : **u32 status = AIO6328_password_clear (u8 CardID,u16 password[5])**

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

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by jumper setting
password[5]	u16	The password previous set

- **AIO6328_security_unlock**

Format : **u32 status = AIO6328_security_unlock (u8 CardID,u16 password[5])**

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

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by jumper setting
password[5]	u16	The password previous set

- **AIO6328_security_status_read**

Format : **u32 status = AIO6328_security_status_read (u8 CardID,u8 *lock_status, u8 *security_enable);**

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

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by jumper setting

Output:

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

Note on security status:

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

9.5 Dll list

	Function Name	Description
1	AIO6328_initial( )	AIO6328 Initial
2	AIO6328_close( )	AIO6328 Close
3	AIO6328_info( )	get OS. assigned address and card information
4	AIO6328_DA_set( )	DA output
5	AIO6328_DA_read( )	read back DA data
6	AIO6328_AD_config_set( )	configure each port as differential or single end
7	AIO6328_AD_config_read( )	configure each channel as differential or single end
8	AIO6328_AD_range_set( )	set up each group conversion range
9	AIO6328_AD_range_read( )	readback each group conversion range setting
10	AIO6328_AD_start( )	start AD conversion of designated channel and trigger to receive the previous conversion data
11	AIO6328_AD_read( )	read AD conversion data. Please refer the Note of this command
12	AIO6328_AD_all_read( )	read all AD conversion data
13	AIO6328_AD_integral_start( )	start AD conversion with integral constant
14	AIO6328_AD_integral_stop( )	stop AD integral conversion
15	AIO6328_AD_integral_all_read( )	read all integral result of AD conversion data
16	AIO6328_debounce_time_set ( )	Set input port digital debounce time
17	AIO6328_debounce_time_read ( )	Read back input port digital debounce time
18	AIO6328_port_set ( )	Set Output port
19	AIO6328_port_read ( )	Read Port Data
20	AIO6328_point_set ( )	Set Output Point State(bit)
21	AIO6328_point_read ( )	Read Input Point State(bit)
22	AIO6328_timer_set ( )	Setup or up date timer
23	AIO6328_timer_read ( )	Read timer on the fly
24	AIO6328_timer_start ( )	Start timer operation
25	AIO6328_timer_stop ( )	Stop timer operation
26	AIO6328_TC_set ( )	Set TC registers
27	AIO6328_TC_read ( )	Read TC registers
28	AIO6328_IRQ_polarity_set( )	Set the IN00~IN07 irq polarity
29	AIO6328_IRQ_polarity_read( )	Read back the IN00~IN07 irq polarity
30	AIO6328_IRQ_mask_set ( )	Set interrupt source mask
31	AIO6328_IRQ_mask_read ( )	Readback interrupt source mask
32	AIO6328_IRQ_process_link ( )	Link interrupt service routine to driver
33	AIO6328_IRQ_enable ( )	Enable interrupt function
34	AIO6328_IRQ_disable ( )	Disable interrupt function
35	AIO6328_IRQ_status_read( )	Read back irq status
36	AIO6328_password_set ( )	Set software key
37	AIO6328_password_change ( )	Change software key
38	AIO6328_password_clear ( )	Clear software key

39	AIO6328_security_unlock ( )	Unlock software key
40	AIO6328_security_status_read ( )	Read software key status

10. AIO6328 Error codes summary

Error Code	Symbolic Name	Description
0	DRV_NO_ERROR	No error.
1	DRV_READ_DATA_ERROR	Read data error
2	DRV_INIT_ERROR	Driver initial error
3	DRV_UNLOCK_ERROR	Software key unlock error
4	DRV_LOCK_COUNTER_ERROR	Software key unlock error count over
5	DRV_SET_SECURITY_ERROR	Software key setting error
100	DEVICE_IO_ERROR	Device I/O Read/Write error
101	DRV_NO_CARD	No AIO6328 card on the system.
102	DRV_DUPLICATE_ID	AIO6328 CardID duplicate error.
103	DRV_NOT_INSTALL	AIO6328 driver not installed completely
300	ID_ERROR	Function input parameter error. CardID setting error, CardID doesn't match the DIP SW setting
301	PORT_ERROR	Function input parameter error. Parameter out of range.
302	POINT_ERROR	Function input parameter error. Parameter out of range.
303	DATA_ERROR	Function input parameter error. Parameter out of range.
304	CONFIGURATION_ERROR	Hardware version can not match with software version
305	DEBOUNCE_TIME_ERROR	Debounce timer setting error
310	EVENT_ERROR	Event error
400	INDEX_ERRROR	TC register index error
401	CONSTANT_ERROR	Time constant error
402	TC_CONTROL_ERROR	TC control register setting error
500	DA_DATA_ERROR	DA setting data error
501	DA_CHANNEL_ERROR	DA channel selection error
600	AD_PORT_ERROR	AD port selection error
601	AD_CHANNEL_ERROR	AD channel selection error
602	AD_CONFIG_ERROR	AD channel configuration error
603	AD_RANGE_ERROR	AD range setting error
604	AD_MODE_ERROR	AD mode setting error
605	AD_RUNNING_ERROR	AD is occupied by other function
700	SOURCE_ERROR	IRQ source error
701	POLARITY_ERROR	IRQ polarity error
702	MASK_ERROR	IRQ mask error