AIO3315/AIO3315A

Analog I/O Card

Software Manual (V1.1)

健昇科技股份有限公司

JS AUTOMATION CORP.

新北市汐止區中興路 100 號 6 樓

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

Xizhi Dist., New Taipei City, Taiwan

TEL：+886-2-2647-6936

FAX：+886-2-2647-6940

http://www.automation.com.tw

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

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

Correction record

Manual Version	Record
1.0	Driver version → wdf3315.sys V1.0, AIO3315.dll V1.0
1.1	Driver version → wdf3315.sys V2.0, AIO3315.dll V2.0
	Add AIO3315_AD_all_read( )

Contents

1. How to install the software of AIO3315 4
   1.1 Install the PCI driver 4
2. Where to find the file you need 5
3. About the AIO3315 software 6
   3.1 What you need to get started 6
   3.2 Software programming choices 6
4. AIO3315 Language support 7
   4.1 Building applications with the AIO3315 software library 7
   4.2 AIO3315 Windows libraries 7
5. Basic concepts of analog I/O control 8
6. Basic concepts of digital I/O control 9
7. Software overview 12
   7.1 Initialization and close 12
   7.2 DA (Digital to analog) function 12
   7.3 AD (Analog to digital) function 12
   7.4 I/O Port R/W 12
   7.5 Timer function 13
   7.6 Interrupt function 13
   7.7 Error conditions 14
8. Flow chart of application implementation 15
   8.1 AIO3315 Flow chart of application implementation 15
9. Function reference 16
   9.1 Function format 16
   9.2 Variable data types 17
   9.3 Programming language considerations 18
   9.4 AIO3315 Functions 20
      Initialization and close 20
         AIO3315_initial 20
         AIO3315_close 20
         AIO3315_info 20
      DA BLOCK 21
         AIO3315_DA_set 21
         AIO3315_DA_read 21
      AD BLOCK 22
         AIO3315_AD_config_set 22
         AIO3315_AD_config_read 23
         AIO3315_AD_range_set 24
         AIO3315_AD_range_read 25
         AIO3315_AD_start 25

AIO3315_AD_read 26
AIO3315_AD_all_read 27
I/O Port R/W 28
AIO3315_port_config_set 28
AIO3315_port_config_read 28
AIO3315_debounce_time_set 29
AIO3315_debounce_time_read 29
AIO3315_port_set 30
AIO3315_port_read 30
AIO3315_point_set 31
AIO3315_point_read 31
Timer function 32
AIO3315_timer_set 32
AIO3315_timer_read 32
AIO3315_timer_start 32
AIO3315_timer_stop 33
AIO3315_TC_set 33
AIO3315_TC_read 33
Interrupt function 34
AIO3315_IRQ_polarity_set 34
AIO3315_IRQ_polarity_read 34
AIO3315_IRQ_mask_set 35
AIO3315_IRQ_mask_read 36
AIO3315_IRQ_process_link 37
AIO3315_IRQ_enable 37
AIO3315_IRQ_disable 37
AIO3315_IRQ_status_read 38
9.5 Dll list 39
10. AIO3315 Error codes table 40

1. How to install the software of AIO3315

1.1 Install the PCI driver

The PCI card is a plug and play card, once you add on a new card, 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 (..\AIO3315_A\Software\Win2K_up\ or if you download from website please execute the file AIO3315_Install.exe to get the file)
6. After installation, power off
7. Power on, it's ready to use

Note: AIO3315, AIO3315A 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 \AIO3315\API (header files and lib files for VB,VC,BCB,C#)

.. \ JS Automation \AIO3315\Driver (backup copy of AIO3315 drivers)

.. \ JS Automation \AIO3315\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 AIO3315 software

AIO3315 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 AIO3315 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 AIO3315 functions within Windows' operation system environment.

3.1 What you need to get started

To set up and use your AIO3315 software, you need the following:

- AIO3315 software
- AIO3315 hardware
  Main board
  Wiring board (Option)

3.2 Software programming choices

You have several options to choose from when you are programming AIO3315 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 AIO3315 software.

4. AIO3315 Language support

The AIO3315 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 AIO3315 software library

The AIO3315 function reference topic contains general information about building AIO3315 applications, describes the nature of the AIO3315 files used in building AIO3315 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 AIO3315 Windows libraries

The AIO3315 for Windows function library is a DLL called **AIO3315.dll**. Since a DLL is used, AIO3315 functions are not linked into the executable files of applications. Only the information about the AIO3315 functions in the AIO3315 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 AIO3315 functions in AIO3315.dll.

Header Files and Import Libraries for Different Development Environments		
Language	**Header File**	**Import Library**
Microsoft Visual C/C++	AIO3315.h	AIO3315VC.lib
Borland C/C++	AIO3315.h	AIO3315BC.lib
Microsoft Visual C#	AIO3315.cs	
Microsoft Visual Basic	AIO3315.bas	
Microsoft VB.net	AIO3315.vb	

Table 1

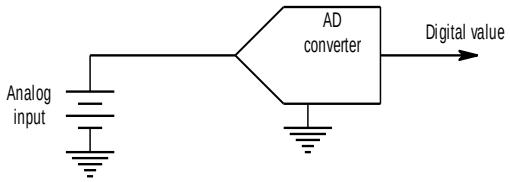
5. Basic concepts of analog I/O control

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

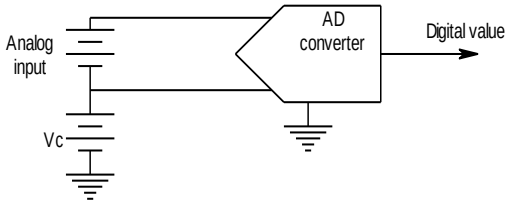
Types of analog input

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 AIO3315 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 AIO3315 has 12 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.
AIO3315 provide voltage type output and 12 bit resolution maps to –10V ~ +10V range.

6. Basic concepts of digital I/O control

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

Types of I/O classified by isolation

If the system and I/O are not electrically connected, we call it is isolated. There are many kinds of isolation: by transformer, by photo-coupler, by magnetic coupler,… Any kind of device, they can 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.**

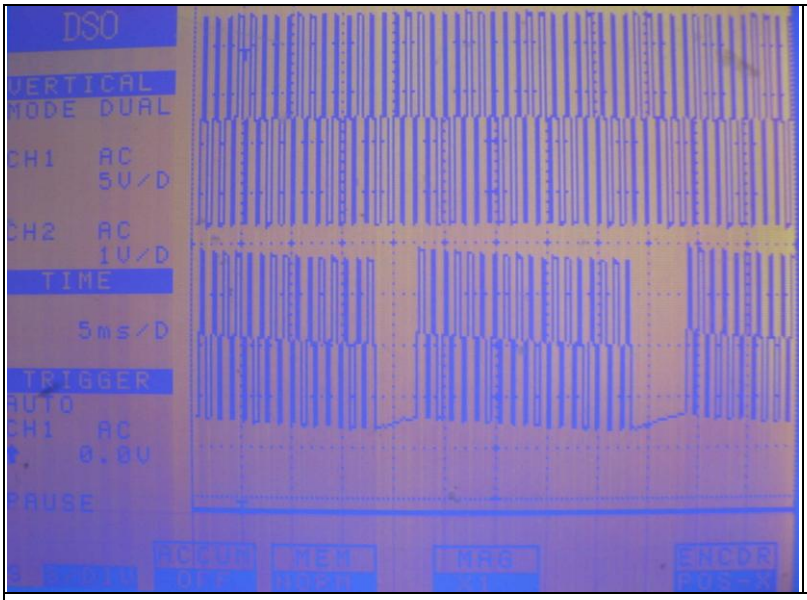
SCR (or triac) is seldom direct connect to digital output, but his relative SSR is the most often selection. In fact, SSR is a compact package of trigger circuit and triac. You can choose zero cross trigger (output command only turn on the output at power phase near zero to eliminate surge) or direct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

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 AIO3315 boards and briefly describes its functions.

7.1 Initialization and close

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

AIO3315_initial( ) will do.

Once you want to close your application, call

AIO3315_close( ) to release all the resource.

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

AIO3315_info( ) to get the address and CardType

7.2 DA (Digital to analog) function

The digital to analog conversion function is implemented by hardware, to output analog voltage just use:

AIO3315_DA_set( ), and you can also read back the settings by

AIO3315_DA_read( ).

7.3 AD (Analog to digital) function

The analog input maybe single end or differential, you can configure individual channel as single end input or the corresponding pair as differential input by:

AIO3315_AD_config_set( ) and read back to verify the configuration setting by

AIO3315_AD_config_read( ).

The analog inputs maybe at different voltage range, you can configure the adequate input range to fit the inputs by:

AIO3315_AD_range_set( ) and read back to verify the settings by:

AIO3315_AD_range_read( )

Once the input type and input range has been set, you can start AD conversion by:

AIO3315_AD_start ( ) and read the conversion data by

AIO3315_AD_read( ).

To read a specific port (contains 8 channels) use:

AIO3315_AD_all_read( )

7.4 I/O Port R/W

Before using a IO port, you must configure the port direction (as input or as output) first by

AIO3315_port_config_set( ) and any time you can read back configuration by

AIO3315_port_config_read( )

Mechanical contact or noisy environment always induced unstable state at digital inputs, the AIO3315 provides software selectable debounce function (the former digital IO cards use hardware debounce and fixed at one frequency). You can filter out the pulse width at 10ms (100Hz), 5ms (200Hz), 1ms (1KHz) or no filter as you need.

Use

AIO3315_debounce_time_set( ) to select the debounce frequency and read back the setting by ***AIO3315_debounce_tme_read( ).***

Then you can use the following functions for I/O port output, data reading and control:

AIO3315_port_set( ) to output byte data to output port,

AIO3315_port_read( ) to read a byte data from I/O port,

AIO3315_point_set ( ) to set output bit,

AIO3315_point_read( ) to read I/O bit,

7.5 Timer function

There is a build in 32 bit timer run on 1us time base, you can set the timer constant by

AIO3315_timer_set( ) and

AIO3315_timer_read( ) to read timer value on the fly.

AIO3315_timer_start( ) to star its operation,

AIO3315_timer_stop( ) to stop operation.

For the timer related registers use:

AIO3315_TC_set( ) to set registers,

AIO3315_TC_read( ) to read back registers.

7.6 Interrupt function

Sometimes you want your application to take care of the I/O while special event occurs, interrupt function is the right choice. AIO3315 provide IO00 ~ IO07 as external event trigger input. You may configure the trigger polarity by:

AIO3315_IRQ_polarity_set( ) and read back by

AIO3315_IRQ_polarity__read( )

For timer ans digital IO interrupts, you can mask off the source you don' want by

AIO3315_IRQ_mask_set( ) and read back by

AIO3315_IRQ_mask_read( ).

After all the above is prepared, you must first link your service routine to the driver by

AIO3315_IRQ_process_link( )

Now all is ready, you can enable the interrupt by

AIO3315_IRQ_enable( ) or disable by

AIO3315_IRQ_disable( ).

To read back the interrupt status (at interrupt service routine or polling routine) use

AIO3315_IRQ_status_read( ).

After reading the status register on card will be cleared.

7.7 Error conditions

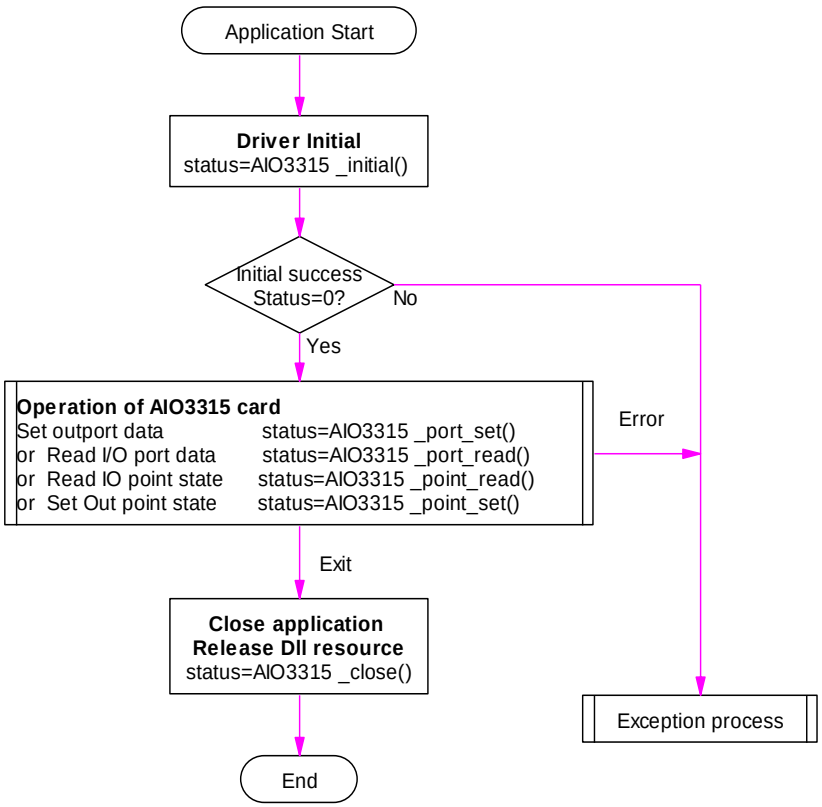
AIO3315 cards minimize error conditions. There are three possible fatal failure modes:

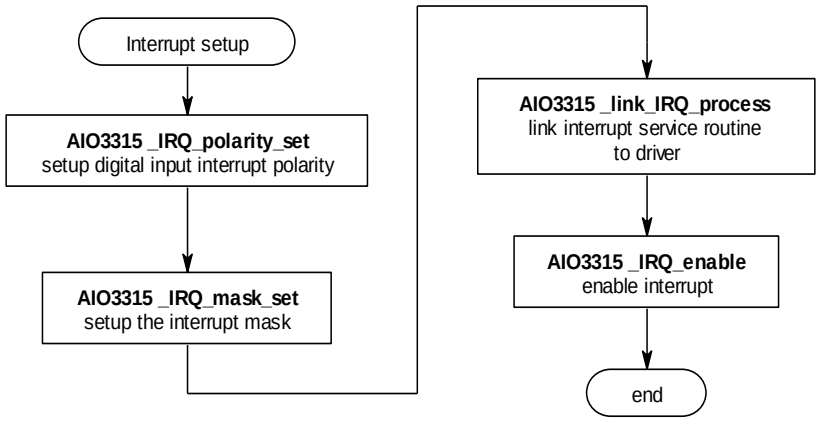
- System Fail Status Bit Valid
- Communication Loss
- Hardware not ready

These error types may indicate an internal hardware problem on the board. Error Codes contains a detailed listing of the error status returned by AIO3315 functions.

8. Flow chart of application implementation

8.1 AIO3315 Flow chart of application implementation





9. Function reference

9.1 Function format

Every AIO3315 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 AIO3315 function is the parameter **CardID** which is located the driver of AIO3315 board you want to use those given operation. The **CardID** is assigned by DIP/ROTARY SW. You can utilize multiple devices with different card CardID within one application; to do so, simply pass the appropriate **CardID** to each function.

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

These topics contain detailed descriptions of each AIO3315 function. The functions are arranged alphabetically by function name. Refer to AIO3315 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	-1.797683134862315E+308 to 1.797683134862315E+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 AIO3315 API. Read the following sections that apply to your programming language.

Note: Be sure to include the declaration functions of AIO3315 prototypes by including the appropriate AIO3315 header file in your source code. Refer to Building Applications with the AIO3315 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 = AIO3315_port_read (u8 CardID, u8 port, u8*data);

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

```
u8 CardID, port;
u8 data,
u32 Status;
Status = AIO3315_port_read (CardID, port, &data);
```

9.3.2 Visual basic

The file AIO3315.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 AIO3315.bas, do not use the numerical values.

In Visual Basic, you can add the entire AIO3315.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 AIO3315.bas file for your project in Visual Basic 4.0, go to the **File** menu and select the **Add File... option**. Select AIO3315.bas, which is browsed in the AIO3315 \ API directory. Then, select **Open** to add the file to the project.

To add the AIO3315.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** AIO3315.bas, which is in the AIO3315 \ 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 AIO3315BC.lib AIO3315.dll

Then add the **AIO3315BC.lib** to your project and add

#include "AIO3315.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 = AIO3315_port_read (u8 CardID, u8 port, u8*data);

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

```
u8 CardID, port;
u8 data;
u32 Status;
Status = AIO3315_port_read (CardID, port, &data);
```

9.4 AIO3315 Functions

Initialization and close

- **AIO3315_initial**

Format : u32 status =AIO3315_initial (void)

Purpose: Initial the AIO3315 resource when start the Windows applications**.**

- **AIO3315_close**

Format : u32 status =AIO3315_close (void);

Purpose: Release the AIO3315 resource when close the Windows applications.

- **AIO3315_info**

Format : u32 status =AIO3315_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: AIO3315 (12 bit version) 1: AIO3315A (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 timer block by OS

DA BLOCK

- **AIO3315_DA_set**

Format : u32 status = AIO3315_DA_set(u8 CardID,u8 channel, u16 data)

Purpose: DA output

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY SW
channel	u8	0: DA0 channel 1: DA1 channel
data	u16	0~0xfff (AIO3315)、0~0xffff (AIO3315A) for analog output amplitude 0V ~ 10V 0V output is 0x7ff (AIO3315) 0x7fff (AIO3315A) -10V output is 0, 10V output is 0xfff (AIO3315) 0xffff (AIO3315A)

- **AIO3315_DA_read**

Format : u32 status = AIO3315_DA_read(u8 CardID,u8 channel, u16 *data)

Purpose: read back DA setting data

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY SW
channel	u8	0: DA0 channel 1: DA1 channel

Output:

Name	Type	Description
data	u16	0~0xfff (AIO3315)、0~0xffff (AIO3315A) for analog output –10V ~ 10V 0V output is 0x7ff (AIO3315) 0x7fff (AIO3315A) -10V output is 0, 10V output is 0xfff (AIO3315) 0xffff (AIO3315A)

AD BLOCK

- **AIO3315_AD_config_set**

Format : **u32 status = AIO3315_AD_config_set (u8 CardID, u8 port, AD_config *AD_config)**

Purpose: configure each channel as differential or single end.

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY SW
port	u8	0: port0, AD0x 1: port1, AD1x 2: port2, AD2x 3: port3, AD3x

AD_config	AD_config	struct _AD_config{
		u8 ch01_config,
		u8 ch23_config,
		u8 ch45_ config,
		u8 ch67_ config
		}
		// ch01: AIx0~AIx1
		// ch23: AIx2~AIx3
		// ch45: AIx4~AIx5
		// ch67: AIx6~AIx7
		// 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}

- **AIO3315_AD_config_read**

Format : **u32 status = AIO3315_AD_config_read (u8 CardID, u8 port, AD_config*AD_config)**

Purpose: read back configuration of each channel.

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY SW
port	u8	0: port0, AD0x 1: port1, AD1x 2: port2, AD2x 3: port3, AD3x

Output:

Name	Type	Description

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

- **AIO3315_AD_range_set**

Format : **u32 status = AIO3315_AD_range_set(u8 CardID, u8 port, AD_range *AD_range)**

Purpose: set up each group conversion range

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY SW
port	u8	0: port0, AD0x 1: port1, AD1x 2: port2, AD2x 3: port3, AD3x

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 **AIO3315_AD_config_set,** then the AD_Range.ch1_range is of no use.

- **AIO3315_AD_range_read**

Format : **u32 status = AIO3315_AD_range_read(u8 CardID, u8 port, AD_range *AD_range)**

Purpose: read back each group conversion range setting

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY SW
port	u8	0: port0, AD0x 1: port1, AD1x 2: port2, AD2x 3: port3, AD3x

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

- **AIO3315_AD_start**

Format : **u32 status = AIO3315_AD_start(u8 CardID,u8 port,u8 channel)**

Purpose: start AD conversion of designated port and channel

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY SW
port	u8	0: port0, AD0x 1: port1, AD1x 2: port2, AD2x 3: port3, AD3x
channel	u8	0~7, channel no for portN

- **AIO3315_AD_read**

Format : **u32 status = AIO3315_AD_read(u8 CardID,u8 port,u16 *data)**

Purpose: read AD conversion data of previous designated port and channel

Parameters:

Input:

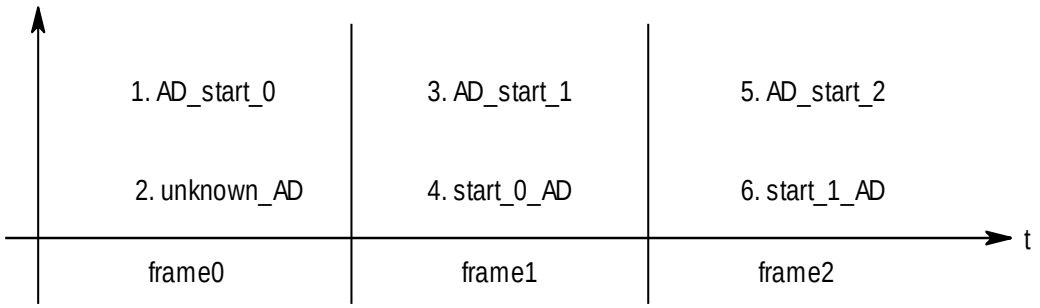
Name	Type	Description
CardID	u8	assigned by DIP/ROTARY SW
port	u8	0: port0, AD0x 1: port1, AD1x 2: port2, AD2x 3: port3, AD3x

Output:

Name	Type	Description
data	u16	0~0xfff (AIO3315), 0~0xffff (AIO3315A). AD converted data

Note:

1. **AIO3315_AD_start** will select the port and channel for the next AD operation.
2. Before read back the data by **AIO3315_AD_read,** you must check the status by **AIO3315_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 follows:



At the same time frame, the command start the designated AD channel and collect the converted data. In order to confirm the operation is complete, we suggest to use ***AIO3315_IRQ_status_read*** to verify the completeness of conversion then use ***AIO3315_AD_read*** to read the converted data.

- **AIO3315_AD_all_read**

Format : **u32 status = AIO3315_AD_all_read(u8 CardID,u8 port, u16 data[8])**

Purpose: read AD conversion data of all channels of a specific port.

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by jumper setting
port	u8	0: port0, AD0x 1: port1, AD1x 2: port2, AD2x 3: port3, AD3x

Output:

Name	Type	Description
data[8]	u16	0~0xfff (AIO3315), 0~0xffff (AIO3315A). AD converted data

Note:

To read all channels, please follow the swquence:

1. Set up start channel at **channel 0** by **AIO3315_AD_start.**
2. Read all channels by **AIO3315_AD_all_read**.

I/O Port R/W

- **AIO3315_port_config_set**

Format : u32 status =AIO3315_port_config_set (u8 CardID, u8 port, u8 configuration)

Purpose: Sets port configuration.

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by Rotary SW
port	u8	port number 0: port0 1: port1
configuration	u8	b0: 0: port0 as input port (default) 1: port0 as output port b1: 0: port1 as input port (default) 1: port1 as output port

- **AIO3315_port_config_read**

Format : u32 status =AIO3315_port_config_read (u8 CardID, u8 port,u8 *configuration)

Purpose: read port configuration.

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by Rotary SW
port	u8	port number 0: port0 1: port1

Output:

Name	Type	Description
configuration	u8	b0: 0: port0 as input port (default) 1: port0 as output port b1: 0: port1 as input port (default) 1: port1 as output port

- **AIO3315_debounce_time_set**

Format : **u32 status = AIO3315_debounce_time_set (u8 CardID,u8 port ,u8 debounce_time)**

Purpose: set the input port debounce time

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by Rotary SW
port	u8	port number 0: port0 1: port1
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

Note: only valid for port configured as input

- **AIO3315_debounce_time_read**

Format : **u32 status = AIO3315_debounce_time_read (u8 CardID,u8 port , u8 *debounce_time)**

Purpose: To read back configuration of debounce mode

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by Rotary SW
port	u8	port number 0: port0 1: port1

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

- **AIO3315_port_set**

Format : **u32 status = AIO3315_port_set (u8 CardID,u8 port, u8 data)**

Purpose: Sets the output data.

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by Rotary SW
port	u8	port number 0: port0 1: port1
data	u8	bitmap of output values If port is configured as input, the data is registered and do not output. If port is configured as output, the data is registered and output.

Note: If you change the configuration from input to output, the previous registered data will be output.

- **AIO3315_port_read**

Format : **u32 status = AIO3315_port_read (u8 CardID , u8 port , u8 *data)**

Purpose: Read the register or input values of the I/O port.

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by Rotary SW
port	u8	port number 0: port0 1: port1

Output:

Name	Type	Description
data	u8	I/O data If port is configured as input, the data is external input data. If port is configured as output, the data is the output register data.

- **AIO3315_point_set**

Format : **u32 status =AIO3315_point_set (u8 CardID, u8 port, u8 point, u8 state)**

Purpose: Sets the bit data of output port.

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by Rotary SW
port	u8	port number 0: port0 1: port1
point	u8	point number 0~7 for bit0~bit7
state	u8	state of output point If port is configured as input, the data is registered and do not output. If port is configured as output, the data is registered and output.

Note: If you change the configuration from input to output, the previous registered data will be output.

- **AIO3315_point_read**

Format : **u32 status =AIO3315_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 Rotary SW
port	u8	port number 0: port0 1: port1
point	u8	point number of input 0~7 for bit0~bit7

Output:

Name	Type	Description
state	u8	state of point of input If port is configured as input, the data is external input data. If port is configured as output, the data is the output register data.

Timer function

- **AIO3315_timer_set**

Format : **u32 status = AIO3315_timer_set (u8 CardID,u32 Timer_constant)**

Purpose: set time constant.

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY SW
Timer_constant	u32	Timer _constant based on 1us time base

Note:

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

- **AIO3315_timer_read**

Format : **u32 status = AIO3315_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

- **AIO3315_timer_start**

Format : **u32 status = AIO3315_timer_start (u8 CardID)**

Purpose: start timer function.

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY SW

- **AIO3315_timer_stop**

Format : **u32 status = AIO3315_timer_stop (u8 CardID)**

Purpose: stop timer function.

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY SW

- **AIO3315_TC_set**

Format : **u32 status= AIO3315_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 DIP/ROTARY SW
index	u8	0: TC_CONTROL 1: PRELOAD 2: TIMER
data	u32	For index = TC_CONTROL 0: stop timer operation 1: timer run For index = 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**.**

- **AIO3315_TC_read**

Format : **u32 status= AIO3315_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 DIP/ROTARY SW
index	u8	0: TC_CONTROL 1: PRELOAD 2: TIMER

Output:

Name	Type	Description
data	u32	Data read back

Interrupt function

- **AIO3315_IRQ_polarity_set**

Format : u32 status = AIO3315_IRQ_polarity_set (u8 CardID, u8 polarity)

Purpose: Sets the IRQ polarity of port0 (IO00~IO07)

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by Rotary SW
polarity	u8	Data to be set, 0x0 ~ 0xff bit0: IO00 0:normal (default) 1:invert … bit7: IO07 0:normal (default) 1:invert

Note: Port0 must configured as input port for IO00~IO07 IRQ function.

- **AIO3315_IRQ_polarity_read**

Format : u32 status = AIO3315_IRQ_polarity_read (u8 CardID, u8 *polarity)

Purpose: Read the IRQ polarity of the IO00~IO07

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by Rotary SW

Output:

Name	Type	Description
polarity	u8	Data to be set, 0x0 ~ 0xff bit0: IO00 0:normal (default) 1:invert … bit7: IO07 0:normal (default) 1:invert

- **AIO3315_IRQ_mask_set**

Format : **u32 status = AIO3315_IRQ_mask_set (u8 CardID,u8 source, u8 mask)**

Purpose: Mask interrupt from port0 IO07~IO00 or timer

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by Rotary SW
source	u8	0:digital I/O block 1: AD block 2: timer block
mask	u8	Digital IO block: b0=0, IO00 input disable irq b0=1, IO00 input can generate irq … b7=0, IO07 input disable irq b7=1, IO07input can generate irq AD block: b0=1 means AD0 end of conversion can generate interrupt b0=0 AD0 will not generate interrupt while end of conversion b1=1 means AD1 end of conversion can generate interrupt b1=0 AD1 will not generate interrupt while end of conversion b2=1 means AD2 end of conversion can generate interrupt b2=0 AD2 will not generate interrupt while end of conversion b3=1 means AD3 end of conversion can generate interrupt b3=0 AD3 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

- **AIO3315_IRQ_mask_read**

Format : **u32 status = AIO3315_IRQ_mask_read (u8 CardID,u8 source,u8 *mask)**

Purpose: read back interrupt Mask of IO07~IO00 or ADC or timer

Parameters:

Input:

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

Output:

Name	Type	Description
mask	u8	Digital IO block: b0=0, IO00 input disable irq b0=1, IO00 input can generate irq … b7=0, IO07 input disable irq b7=1, IO07input can generate irq AD block: b0=1 means AD0 end of conversion can generate interrupt b0=0 AD0 will not generate interrupt while end of conversion b1=1 means AD1 end of conversion can generate interrupt b1=0 AD1 will not generate interrupt while end of conversion b2=1 means AD2 end of conversion can generate interrupt b2=0 AD2 will not generate interrupt while end of conversion b3=1 means AD3 end of conversion can generate interrupt b3=0 AD3 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

- **AIO3315_IRQ_process_link**

Format : **u32 status = AIO3315_IRQ_process_link (u8 CardID,**
void (__stdcall *callbackAddr)(u8 CardID))

Purpose: Link irq service routine to driver

Parameters:

Input:

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

- **AIO3315_IRQ_enable**

Format : **u32 status = AIO3315_IRQ_enable (u8 CardID, HANDLE *phEvent)**

Purpose: Enable interrupt from selected source

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by Rotary SW

Output:

Name	Type	Description
phEvent	HANDLE	event handle

- **AIO3315_IRQ_disable**

Format : **u32 status = AIO3315_IRQ_disable (u8 CardID)**

Purpose: Disable interrupt from selected source

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by Rotary SW

- **AIO3315_IRQ_status_read**

Format : **u32 status = AIO3315_IRQ_status_read (u8 CardID,u8 source, u8 *Event_Status)**

Purpose: To read back the interrupt status to identify the source

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by Rotary SW
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, IO00 input generate irq … b7=1, IO07 input generate irq AD block: b0=1, AD0 end of conversion and data is ready b0=0, AD0 is under conversion b1=1, AD1 end of conversion and data is ready b1=0, AD1 is under conversion b2=1, AD2 end of conversion and data is ready b2=0, AD2 is under conversion b3=1, AD3 end of conversion and data is ready b3=0, AD3 is under conversion Timer block: b0=1 means timer count up occurred. b0=0 means timer not count up.

Note:

1. Status read back will also clear the on board status register.
2. The status will reflect the on board digital input or timer count up status are irrelevant to the IRQ_MASK

9.5 Dll list

	Function Name	Description
1.	AIO3315_initial( )	AIO3315 Initial
2.	AIO3315_close( )	AIO3315 Close
3.	AIO3315_info( )	get OS. Assigned address
4.	AIO3315_DA_set( )	DA output
5.	AIO3315_DA_read( )	read back DA setting data
6.	AIO3315_AD_config_set( )	configure each channel as differential or single end
7.	AIO3315_AD_config_read( )	read back configuration of each channel
8.	AIO3315_AD_range_set( )	set up each group conversion range
9.	AIO3315_AD_range_read( )	Read back each group conversion range setting
10.	AIO3315_AD_start( )	start AD conversion of designated channel
11.	AIO3315_AD_read( )	read AD conversion data
12.	AIO3315_AD_all_read( )	Read a specific port AD data
13.	AIO3315_port_config_set( )	Port direction configuration
14.	AIO3315_port_config_read( )	Read back port configuration
15.	AIO3315_debounce_time_set( )	Set input port debounce time
16.	AIO3315_debounce_time_read( )	Read back input port debounce time
17.	AIO3315_port_set( )	Set Output port
18.	AIO3315_port_read( )	Read the register or input values of the I/O port
19.	AIO3315_point_set ( )	Set the bit data of output port
20.	AIO3315_point_read( )	Read the state of the input points or output register
21.	AIO3315_timer_set( )	Set timer constant
22.	AIO3315_timer_read( )	Read timer on the fly
23.	AIO3315_timer_start( )	Start timer operation
24.	AIO3315_timer_stop( )	Stop timer operation
25.	AIO3315_TC_set( )	load data to timer related registers
26.	AIO3315_TC_read( )	Read back data of timer related registers
27.	AIO3315_IRQ_polarity_set( )	Sets the IRQ polarity of port0
28.	AIO3315_IRQ_polarity_read( )	Read back the setting of IRQ polarity
29.	AIO3315_IRQ_mask_set( )	Mask off the IRQ
30.	AIO3315_IRQ_mask_read( )	Read back the mask
31.	AIO3315_IRQ_process_link( )	Link irq service routine
32.	AIO3315_IRQ_enable( )	Enable interrupt function
33.	AIO3315_IRQ_disable( )	Disable interrupt function
34.	AIO3315_IRQ_status_read( )	Read back the IRQ status

10. AIO3315 Error codes table

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
100	DEVICE_IO_ERROR	Device Read/Write error
101	DRV_NO_CARD	No AIO3315 card on the system.
102	DRV_DUPLICATE_ID	AIO3315 CardID duplicate error.
103	DRV_NOT_INSTALL	AIO3315 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
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
700	SOURCE_ERROR	IRQ source error
701	POLARITY_ERROR	IRQ polarity error
702	MASK_ERROR	IRQ mask error