AIO3320A/21A

Analog Input and Multi-Function Digital I/O Card

Software Manual (V1.0)

健昇科技股份有限公司

JS AUTOMATION CORP.

新北市汐止區中興路 100 號 6 樓

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

Xizhi Dist., New Taipei City, Taiwan

電話(TEL)：+886-2-2647-6936

傳真(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	First publish

Contents

1. Compatibility about the AIO3320 and AIO3320A

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

JS Automation tries to keep all the availability of its products before the market phase out. AIO3320A/1A is fully compatible with AIO3320/1, you can use it without any change of you old design but we strong recommend to use new driver and coding convention with new cards if you do not use it as just maintenance replacement of old version card.

card model	compatibility	
	hardware	software
AIO3320/1	----	use AIO3320.dll (new function do not support old version card)
AIO3320A/1A	direct compatible to AIO3320	AIO3320.dll
	direct compatible to AIO3320	use AIO3320A.dll with new coding convention that provides full functions

*1: old coding convention: the function call provides in AIO3320 software manual.

*2: new coding convention: the function call provides in AIO3320A software manual.

**We recommend to use new coding convention for AIO3320A for easier to upgrade if need.

2. How to install the software of AIO3320A

2.1 Install the PCI driver

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

In WinXP/7 and up system you should: (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 (..\AIO3320A\software\WinXP_7\ or if you download from website please execute the file AIO3320A_Install.exe to get the file)
6. After installation, power off
7. Power on, it's ready to use

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

http://automation.com.tw/

to download the complementary documents.

3. Where to find the file you need

WinXP/7 and up

The directory will be located at

.. \ JS Automation \AIO3320A\API (header files and lib files for VB,VC,BCB,C#)

.. \ JS Automation \AIO3320A\Driver (backup copy of AIO3320A drivers)

.. \ JS Automation \AIO3320A\exe (demo program and source code)

The system driver is located at **..\system32\Drivers** and the DLL is located at **..\system**.

For your easy startup, the demo program with source code demonstrates the card functions and help file.

4. About the AIO3320A software

AIO3320A 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 AIO3320A 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 AIO3320A functions within Windows' operation system environment.

4.1 What you need to get started

To set up and use your AIO3320A software, you need the following:

- AIO3320A software
- AIO3320A hardware
  Main board
  Wiring board (Option)

4.2 Software programming choices

You have several options to choose from when you are programming AIO3320A 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 AIO3320A software.

5. AIO3320A Language support

The AIO3320A software library is a DLL used with WinXP/7 and up. You can use these DLL with any Windows integrating development environment that can call Windows DLLs.

5.1 Building applications with the AIO3320A software library

The AIO3320A function reference topic contains general information about building AIO3320A applications, describes the nature of the AIO3320A files used in building AIO3320A 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.

5.2 AIO3320A Windows libraries

The AIO3320A for Windows function library is a DLL called **AIO3320A.dll**. Since a DLL is used, AIO3320A functions are not linked into the executable files of applications. Only the information about the AIO3320A functions in the AIO3320A 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 AIO3320A functions in AIO3320A.dll.

Header Files and Import Libraries for Different Development Environments		
Language	**Header File**	**Import Library**
Microsoft Visual C/C++	AIO3320A.h	AIO3320AVC.lib
Borland C/C++	AIO3320A.h	AIO3320ABC.lib
Microsoft Visual C#	AIO3320A.cs	
Microsoft Visual Basic	AIO3320A.bas	
Microsoft VB.net	AIO3320A.vb	

Table 1

6. Basic concepts of analog I/O

6.1 Analog input (analog to digital converter)

Analog input (analog to digital conversion) is used for digital system to access external world's physical quantity. The most common specifications are: 5V, 10V for unipolar voltage input; +-5V, +-10V for bipolar voltage input. Also for current measuring: 0-20ma or 4-20ma is common for industrial applications.

In the conversion of analog signal to digital, the conversion time is defined as: the time required for an A/D converter to complete a single conversion to specified resolution and linearity for a full scale analog input change. Resolution is defined as: the smallest change that can be distinguished by an A/D converter. Resolution may be stated in percentage of full scale but is commonly expressed as the number of bit n where the converter has 2^n possible states.

Selecting a A/D, you must know how fast you need and how accuracy you required. As a thumb of rule, the conversion speed is recommended as 10 times of the under control process will get good result to re-construct the original waveform.

6.2 Analog output (digital to analog converter)

Analog output (digital to analog converter) is used for the digital system to simulate analog signal to external physical quantity. The most common specifications are: 5V, 10V for unipolar voltage output; +-5V, +-10V for bipolar voltage output. Also for current output: 0-20ma or 4-20ma is common for industrial applications (source or sink also possible).

In the conversion of digital to analog signal, the settling time is defined as: the time elapsed from the application of a full step input to a circuit to the time when the output has entered and remained within a specified error band around its final value. Resolution is defined as: the smallest change that can be distinguished by a D/A converter. Resolution may be stated in percentage of full scale but is commonly expressed as the number of bit n where the converter has 2^n possible states.

Selecting a D/A, you must know how fast the signal you will change and how accuracy you required. As a thumb of rule, the settling time decides the how fast your signal can be output. You cannot get accurate output shorter than the settling time.

6.3 Quantization error

On the fig. 6.1, if the vertical Y axis is the digital input you can see the output of D/A is step-wise. Same as if you take horizontal X axis as input, you can see the digital output that analog input quantized to digital output. The quantization error shown as fig 6.2

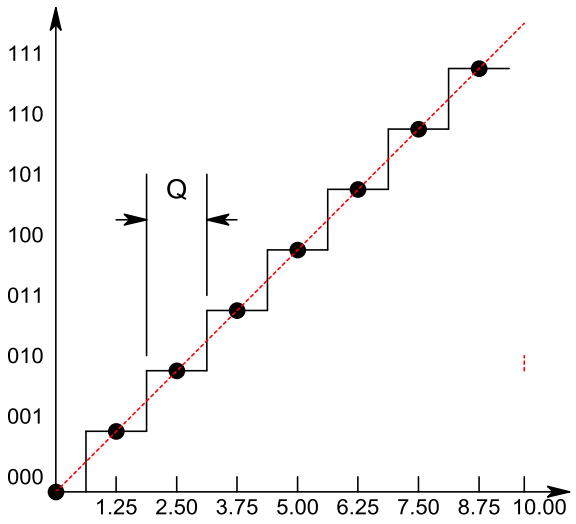


fig 6.1 Input-output relation for A/D or D/A

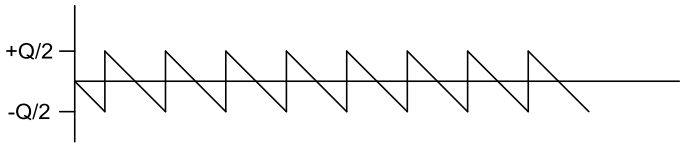


fig 6.2 Quantization error

6.4 Sample and hold

Sample and hold circuit often used to hold analog input to A/D, it will freeze the input and provide the A/D converter a near frozen state to achieve accuracy conversion. (Be sure that if the input analog signal varies at the conversion period, you cannot get accurate conversion A/D data at the sample time slice). You can also use the circuit to hold the D/A signal which you can expand D/A channels with only one fast D/A converter to multiplex outputs.

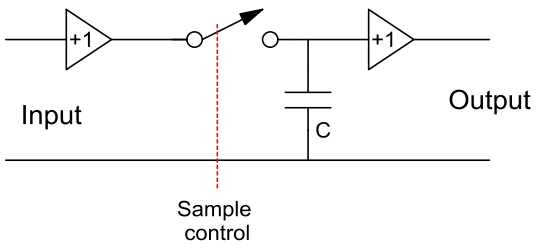


fig 6.3 Sample and hold

7. Function format and language difference

7.1 Function format

Every AIO3320A 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 AIO3320A function is the parameter **CardID** which is located the driver of AIO3320A 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**)

7.2 Variable data types

Every function description has a parameter table that lists the data types for each parameter. The following sections describe the notation used in those parameter tables and throughout the manual for variable data types.

Primary Type Names					
Name	**Description**	**Range**	**C/C++**	**Visual BASIC**	**Pascal (Borland Delphi)**
u8	8-bit ASCII character	0 to 255	char	Not supported by BASIC. For functions that require character arrays, use string types instead.	Byte
i16	16-bit signed integer	-32,768 to 32,767	short	Integer (for example: deviceNum%)	SmallInt
u16	16-bit unsigned integer	0 to 65,535	unsigned short for 32-bit compilers	Not supported by BASIC. For functions that require unsigned integers, use the signed integer type instead. See the i16 description.	Word
i32	32-bit signed integer	-2,147,483,648 to 2,147,483,647	long	Long (for example: count&)	LongInt
u32	32-bit unsigned integer	0 to 4,294,967,295	unsigned long	Not supported by BASIC. For functions that require unsigned long integers, use the signed long integer type instead. See the i32 description.	Cardinal (in 32-bit operating systems). Refer to the i32 description.
f32	32-bit single-precision floating-point value	-3.402823E+38 to 3.402823E+38	float	Single (for example: num!)	Single
f64	64-bit double-precision floating-point value	-1.797683134862315E+308 to 1.797683134862315E+308	double	Double (for example: voltage Number)	Double

Table 2

7.3 Programming language considerations

Apart from the data type differences, there are a few language-dependent considerations you need to be aware of when you use the AIO3320A API. Read the following sections that apply to your programming language.

Note: Be sure to include the declaration functions of AIO3320A prototypes by including the appropriate AIO3320A header file in your source code. Refer to Building Applications with the AIO3320A Software Library for the header file appropriate to your compiler.

7.3.1 C/C++

For C or C++ programmers, parameters listed as Input/Output parameters or Output parameters are pass-by-reference parameters, which means a pointer points to the destination variable should be passed into the function. For example, the Read Port function has the following format:

```
Status = AIO3320A_port_read(CardID, port, *data);
```

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

```
u8 CardID, port;
u8 data,
u32 Status;
Status = AIO3320A_port_read (CardID, port, data);
```

7.3.2 Visual basic

The file AIO3320A.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 AIO3320A.bas, do not use the numerical values.

In Visual Basic, you can add the entire AIO3320A.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 AIO3320A.bas file for your project in Visual Basic 4.0, go to the **File** menu and select the **Add File... option**. Select AIO3320A.bas, which is browsed in the AIO3320A \ API directory. Then, select **Open** to add the file to the project.

To add the AIO3320A.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** AIO3320A.bas, which is in the AIO3320A \ API directory. Then, select **Open** to add the file to the project.

7.3.3 Borland C++ builder

To use Borland C++ builder as development tool, you should generate a .lib file from the .dll file by implib.exe.

implib AIO3320ABC.lib AIO3320A.dll

Then add the **AIO3320ABC.lib** to your project and add

#include “AIO3320A.h” to main program.

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

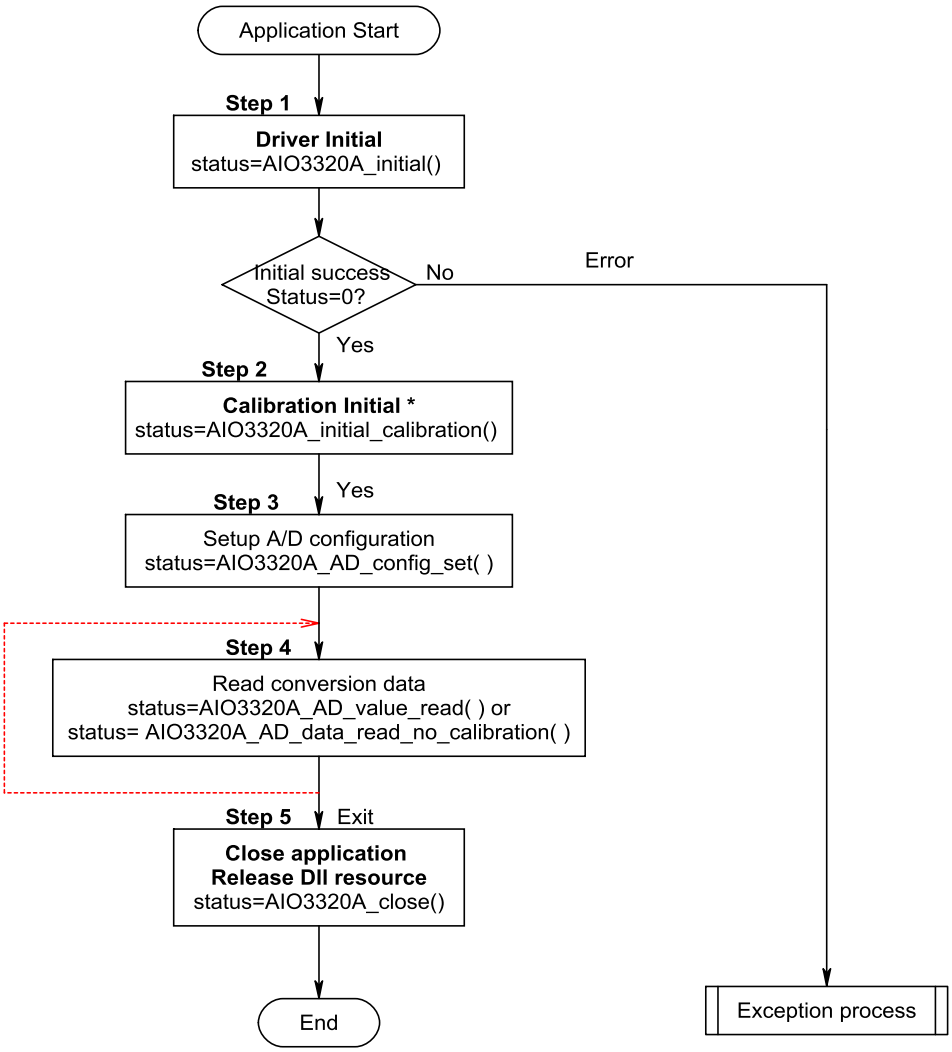
```
Status = AIO3320A_port_read(CardID, port, *data);
```

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

```
u8 CardID, port;
u8 data;
u32 Status;
Status =AIO3320A_port_read (CardID, port, data);
```

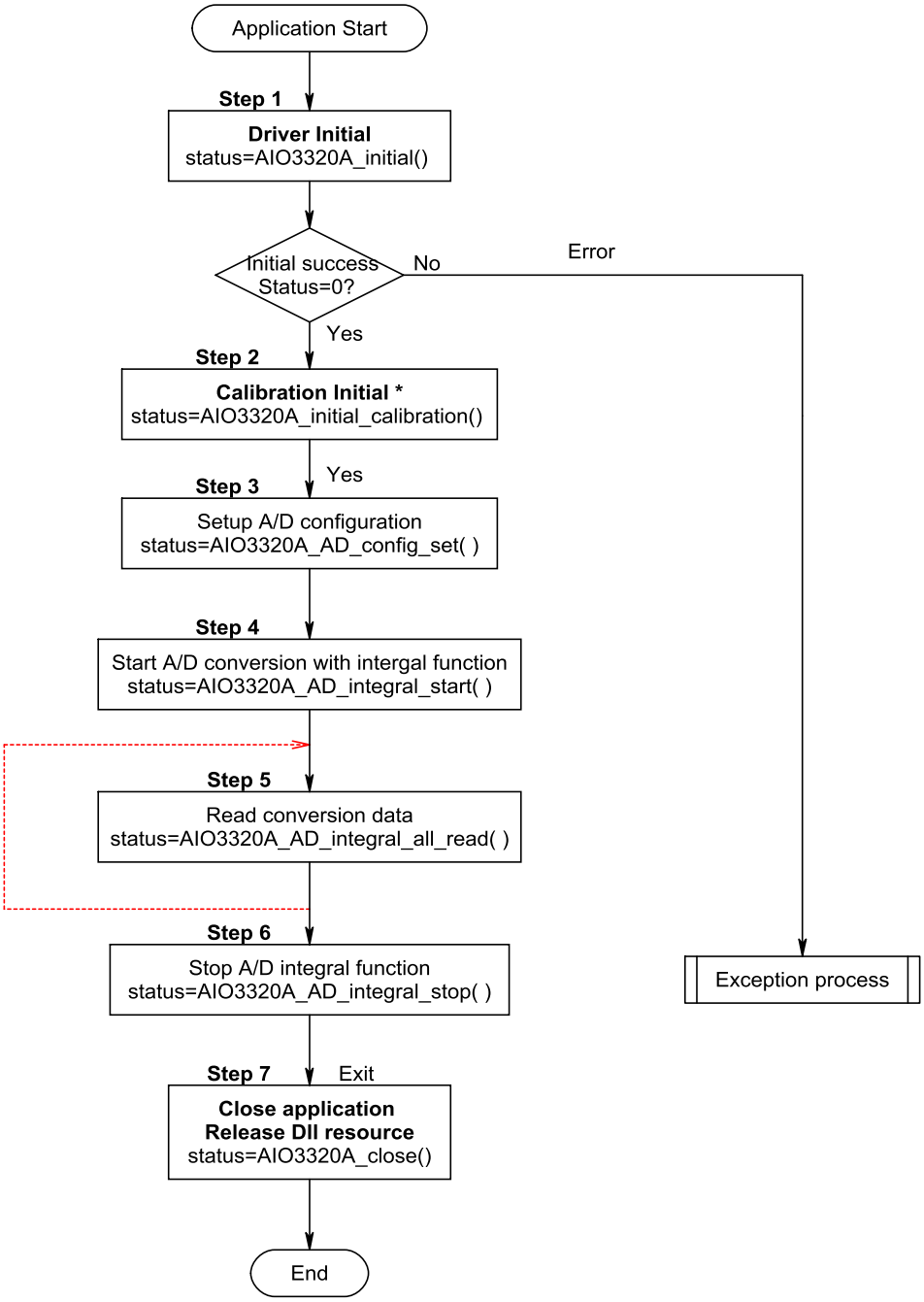
8. Flow chart of application implementation

8.1 AIO3320A Flow chart of application implementation (Basic AD operation)



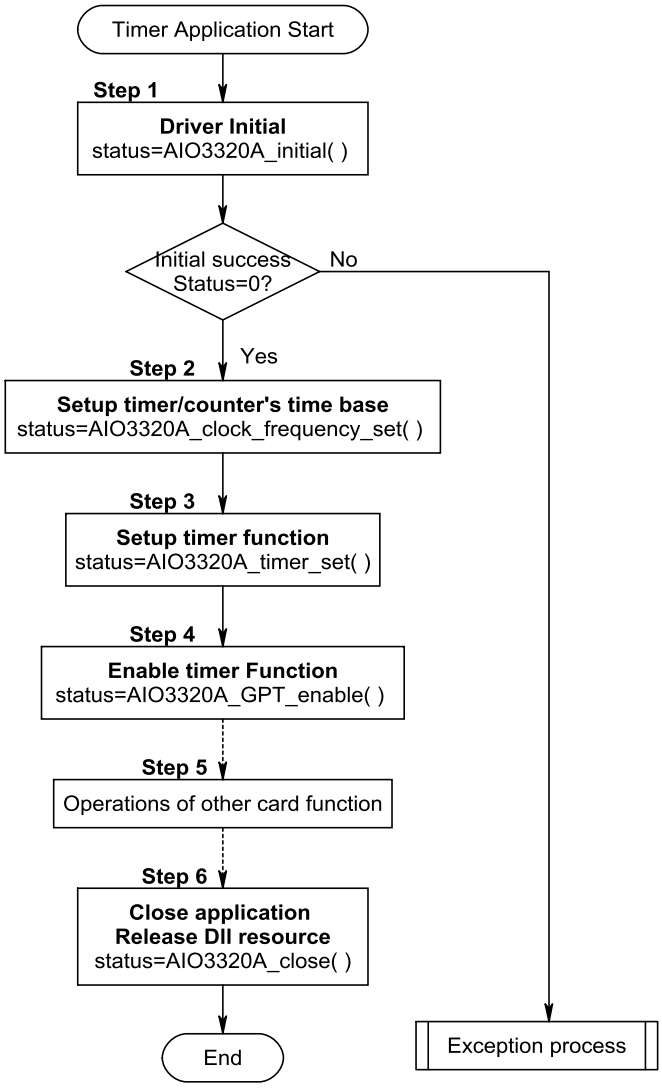
* If you need the data use factory pre-calibrated data to calibrate.

8.2 AIO3320A Flow chart of application implementation (Embedded integral AD operation)

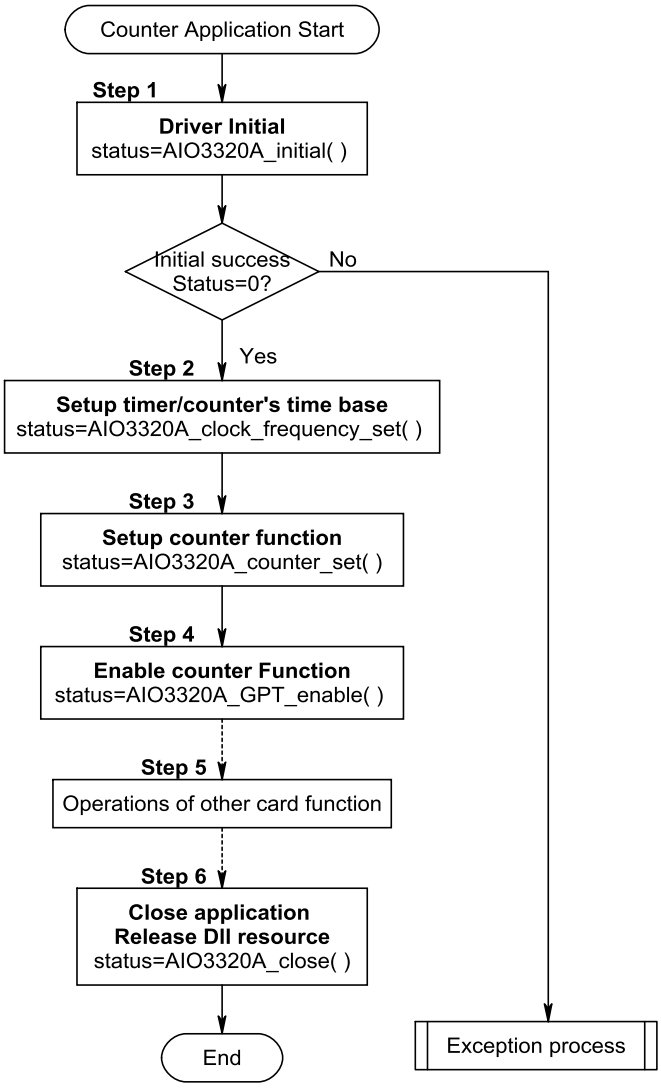


* If you need the data use factory pre-calibrated data to calibrate.

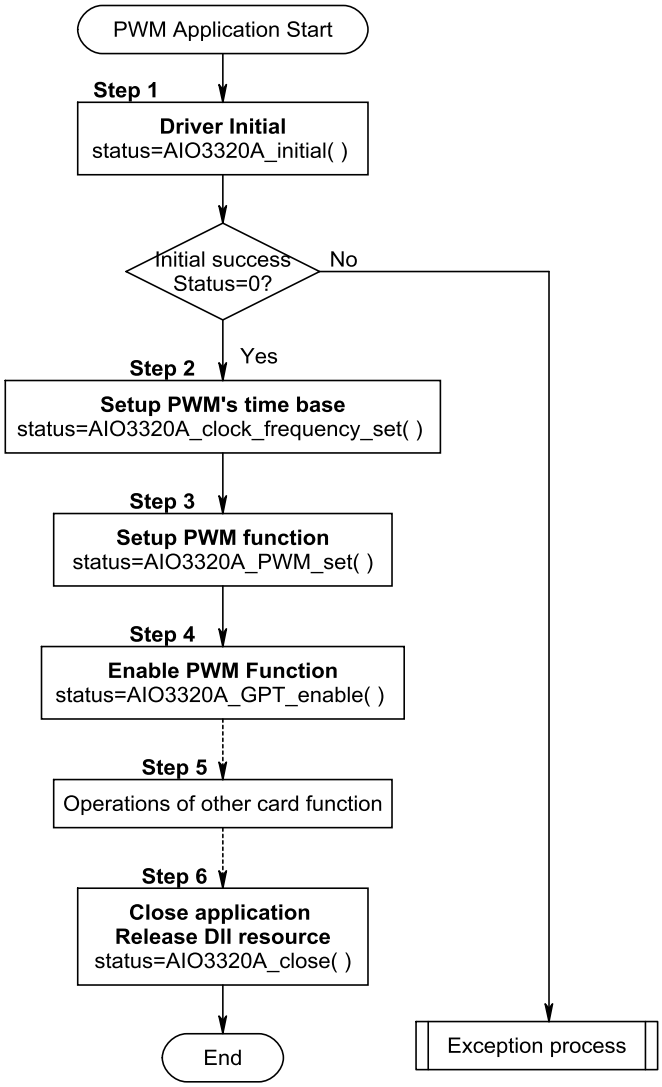
8.3 AIO3320A Flow chart of Timer / Counter / PWM application



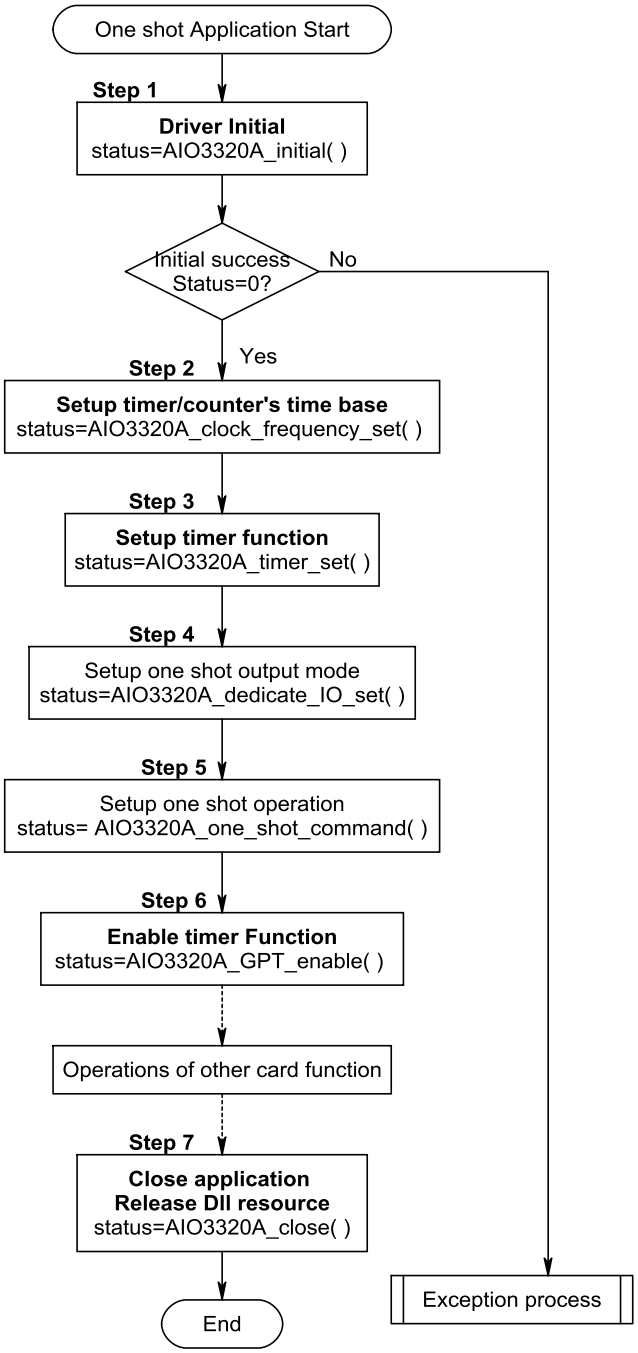
Application of timer



Application of counter

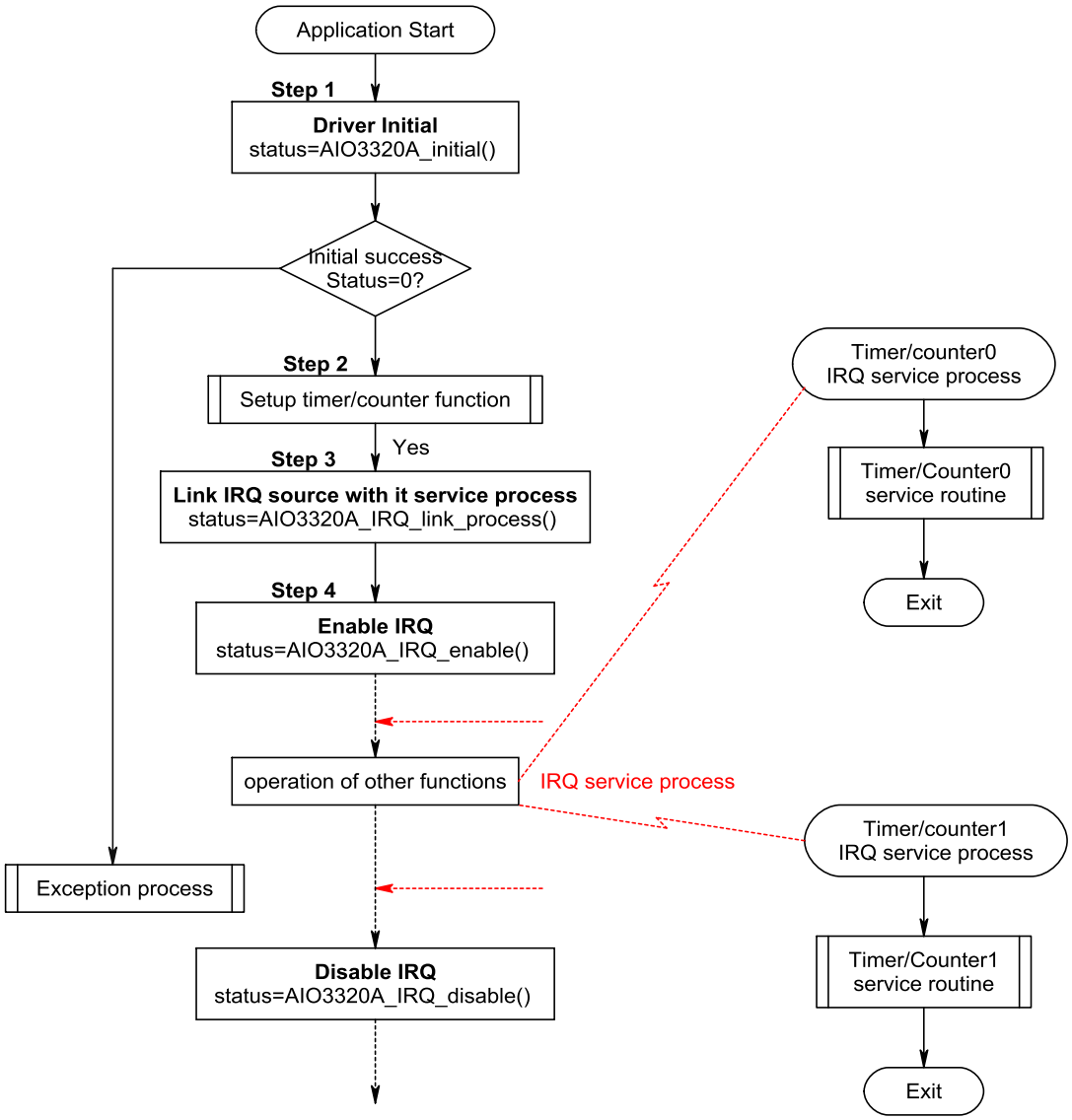


PWM application



One shot output application

8.4 AIO3320A Flow chart of interrupt



9. Software overview and dll function

9.1 Initialization and close

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

AIO3320A_initial() will do.

Before you want to A/D conversion, the factory pre-calibrated data should be initialized for A/D conversion.

AIO3320A_initial_calibration() will read factory calibrated data to working area.

Once you want to close your application, call

AIO3320A_close( ) to release all the resource.

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

AIO3320A_info( ) to get the address.

- **AIO3320A_initial**

Format : **u32 status =AIO3320A_initial(void)**

Purpose: Initial the AIO3320A resource when start the Windows applications.

- **AIO3320A_initial_calibration**

Format : **u32 status =AIO3320A_initial_calibration(u8 CardID)**

Purpose: Read the factory pre-calibrated data for the future calibration.

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY SW

- **AIO3320A_close**

Format : **u32 status =AIO3320A_close(void);**

Purpose: Release the AIO3320A resource when close the Windows applications.

- **AIO3320A_info**

Format : **u32 status =AIO3320A_info(u8 CardID,u16 *address)**

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

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY SW

Output:

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

9.2 Analog input

The AIO3320A now are 8 channels 16 bit AD cards. There are different channel D/A numbers for AIO3320A (2 channels) and AIO3320A (4 channels).

You must configure the input range of the specific channel by:

AIO3320A_AD_config_set( ) and read back the configuration for verification by:

AIO3320A_AD_config_read( )

To read the input voltage value with the factory pre-calibrated data by:

AIO3320A_AD_value_read( ), it can be also read without the calibration by

AIO3320A_AD_value_read_no_calibration( )

If your application only need the raw AD data, you can read AD data by:

AIO3320A_AD_data_read_no_calibration( )

If your environment is noisy or you need to get more accurate data, you can use the integral function built in dll by:

AIO3320A_AD_integral_start( ) to start the embedded integration of the system.

AIO3320A_AD_integral_all_read( ) to read all channels.

AIO3320A_AD_integral_stop( ) to stop integration function.

- **AIO3320A_AD_config_set**

Format : u32 status = AIO3320A_AD_config_set(u8 CardID,u8 channel,u8 mode)

Purpose: Set A/D configuration.

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY SW
channel	u8	A/D channel number 0~7: 8 channels AD
mode	u8	scale range: 0: 0V ~ 5V 1: -5V ~ +5V 2: 0V ~ 10V 3: -10V ~ +10V 255 : AD stop operation.

Note: 0~5V, 0~20mA or 4~20mA are set by hardware but software must configure to meet. For the first time after power on or the first time after application opened, you should configure the AD mode as the hardware setting else **incorrect** reading will be.

- **AIO3320A_AD_config_read**

Format : **u32 status = AIO3320A_AD_config_read(u8 CardID,u8 channel,u8 *mode)**

Purpose: Read A/D config.

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY SW
channel	u8	A/D channel number 0~7: 8 channels AD

Output:

Name	Type	Description
mode	u8	scale range: 0: 0V ~ 5V 1: -5V ~ +5V (Default) 2: 0V ~ 10V 3: -10V ~ +10V 255 : AD stop operation.

- **AIO3320A_AD_value_read**

Format : **u32 status = AIO3320A_AD_value_read(u8 CardID, u8 channel, f32 *value)**

Purpose: Read voltage value with pre-calibration data.

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY SW
channel	u8	A/D channel number 0~7: 8 channels AD

Output:

Name	Type	Description
value	f32	Voltage value based on the AD converted and calibrated data. Say if the AD scale range is set at 0~5V then the voltage value returned will be in the 0~5 range.

Note: Use *AIO3320A_initial_calibration( )* to load calibration data first then the further calibration will be effective.

- **AIO3320A_AD_value_read_no_calibration**

Format : **u32 status = AIO3320A_AD_value_read_no_calibration (u8 CardID, u8 channel, f32 *value)**

Purpose: Read voltage value.

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY SW
channel	u8	A/D channel number 0~7: 8 channels AD

Output:

Name	Type	Description
value	f32	Voltage value based on the AD converted data only. Say if the AD scale range is set at 0~10V then the voltage value returned will be in the 0~10 range.

- **AIO3320A_AD_data_read_no_calibration**

Format : **u32 status = AIO3320A_AD_data_read_no_calibration (u8 CardID, u8 channel, u16 *data)**

Purpose: Read A/D data.

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by DIP/ROTARY SW
channel	u8	A/D channel number 0~7: 8 channels AD

Output:

Name	Type	Description
data	u16	AD 16 bit data, 0~65535: if unipolar AD mode, 0~5V or 0~10V range 0~32767: if bipolar AD mode, in 0~5V or 0~10V range 65535~32768: if bipolar AD mode, in 0~ -5V or 0 ~ -10V range

- **AIO3320A_AD_integral_start**

Format : u32 status = AIO3320A_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

- **AIO3320A_AD_integral_all_read**

Format : u32 status = AIO3320A_AD_integral_all_read(u8 CardID,u16 data[8])

Purpose: read integral result of AD conversion data.

Parameters:

Input:

Name	Type	Description
CardID	u8	assigned by jumper setting

Output:

Name	Type	Description
data[8]	u16	Channel 0 ~7 AD data

Note:

To read all channels in integral

Start integral mode by **AIO3320A_AD_integral_start.** (AD interrupt is not allowed to use)

Read all channels by AIO3320A_AD_integral_all_read.

Stop AD integration function by **AIO3320A_AD_integral_stop.**