EMA-8314R
Ethernet RTD I/O module

Software Manual (V1.0)

健昇科技股份有限公司

JS AUTOMATION CORP.

新北市汐止區中興路 100 號 6 樓

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

Xizhi Dist., New Taipei City, Taiwan

TEL：+886-2-2647-6936

FAX：+886-2-2647-6940

http://www.automation.com.tw

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

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

Correction record

Version	Record
1.0	EMA-8314.dll v1.0

Contents

1. How to install the software of EMA8314R

Please register as user's club member to download the
"Step by step installation of EMA8314" document from http://automation.com.tw

1.1 Install the EMA driver

The ether net module can not found by OS as PCI cards. You can just install the driver without the module installed. Execute the file ..\install\EMA8314_Install.exe to install the driver, API and demo program automatically.

For a more detail descriptions, please refer "Step by step installation of Ethernet IO module".

2. Where to find the file you need

Windows2000, XP and up

In Windows 2000,XP and up operating system, the demo program can be setup by EMA8314_Install.exe.

If you use the default setting, a new directory .. \JS Automation\EMA8314 will generate to put the associate files.

.. / JS Automation /EMA8314/API (header files and VB,VC lib files)

.. / JS Automation /EMA8314/Driver (copy of driver code)

.. / JS Automation /EMA8314/exe (demo program and source code)

The dll is located at ..\system.

3. About the EMA8314 software

EMA8314 software includes a set of dynamic link library (DLL) based on socket that you can utilize to control the interface functions.

Your EMA8314 software package includes setup driver, 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 EMA8314 functions within Windows' operation system environment.

3.1 What you need to get started

To set up and use your EMA8314 software, you need the following:

- EMA8314 software
- EMA8314 hardware

3.2 Software programming choices

You have several options to choose from when you are programming EMA8314 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 EMA8314 software.

4. EMA8314 Language support

The EMA8314 software library is a DLL used with Windows 2000/XP and up. You can use these DLL with any Windows integrating development environment that can call Windows DLLs.

4.1 Building applications with the EMA8314 software library

The EMA8314 function reference section contains general information about building EMA8314 applications, describes the nature of the EMA8314 functions used in building EMA8314 applications, and explains the basics of making applications using the following tools:

Applications tools

- Borland C/C++
- Microsoft Visual C/C++
- Microsoft Visual Basic

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

EMA8314 Windows Libraries

The EMA8314 for Windows function library is a DLL called **EMA8314.dll**. Since a DLL is used, EMA8314 functions are not linked into the executable files of applications. Only the information about the EMA8314 functions in the EMA8314 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 EMA8314 functions in EMA8314 .dll.

Header Files and Import Libraries for Different Development Environments		
Development Environment	Header File	Import Library
Microsoft C/C++	EMA8314.h	EMA8314VC.lib
Borland C/C++	EMA8314.h	EMA8314BC.lib
Microsoft Visual Basic	EMA8314.bas	

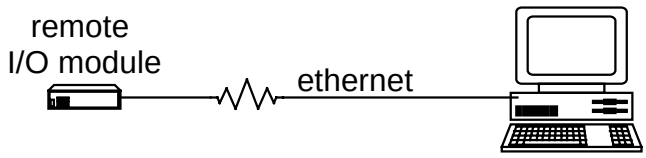
Table 1

5. Basic concept of the remote RTD I/O module

I/O communicate via Ethernet

The remote RTD I/O is the function extension of the card type device. If the under control target is at a long distance away, the card type is limited by the wiring, it is very difficult to go far away but the Ethernet remote analog I/O will do.

JS automation keeps the remote RTD I/O functions as the card type device does. The dll functions provided makes users can program the remote RTD device as if it is a card without caring the communication functions after setting up the device IP and communication port.



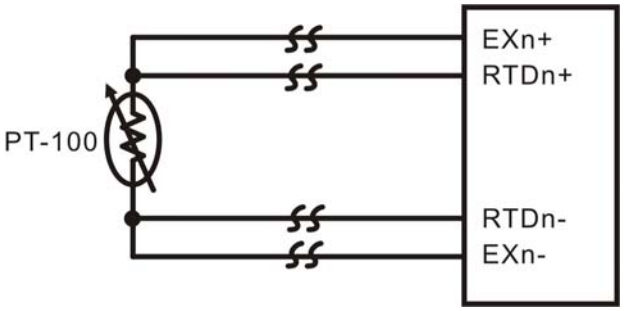
The module response or be commanded by the controller through Ethernet by UDP protocol. As a member on the Ethernet, it must have a distinguished IP and a predefined port. At factory, we set the default IP at 192.168.0.100 and the port at 6936 for the remote module.

If you want to control the module through internet, you must configure your network to pass the message to the module, say, your gateway allows the message from outside to go to the module and also the message from the module can go out to the internet. Please check with your internet engineer to set up the environment.

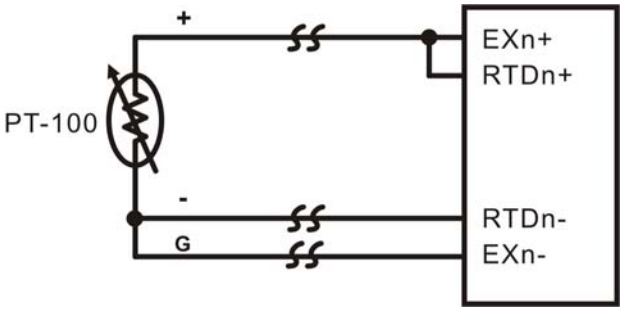
RTD and its connection conventions

An RTD (Resistance Temperature Detector) is basically a temperature sensitive resistor. It is a positive temperature coefficient device, which means that its resistance increases with temperature. Platinum is the primary choice for most industrial, commercial, laboratory and other critical RTD temperature measurements. It has fairly linear characteristic over the measuring temperature range.

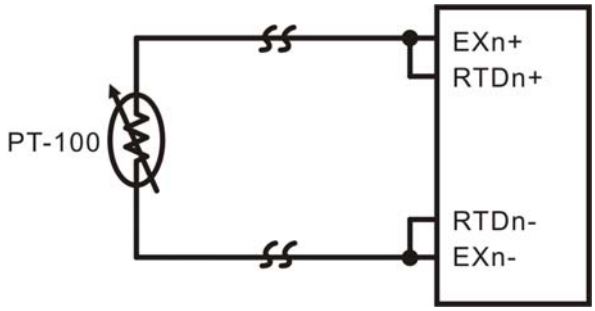
There are several connection conventions for the RTD sensors; we will give you some hints over each connection.



4 wire connection



3 wire connection



2 wire connection

The terminal EXn+ and EXn- are the excitation, RTDn+ and RTDn- are the sensor signal input, for a long distance wiring, to compensate the wire resistance; the 4 wire connection will be better. But in short distance, 2 wire, 3 wire or 4 wire connection seems no difference.

6. Function format and language difference

6.1 Function format

Every EMA8314 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 EMA8314 function is the parameter **CardID** which is set by ***EMA8314_IP_mapping*** . You can utilize multiple devices with different card ID within one application; to do so, simply pass the appropriate **CardID** to each function.

[*1] Command concerning the system rebooting, please wait for about 10s to precede the next communication.

6.2 Variable data types

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

Primary Type Names					
Name	**Description**	**Range**	**C/C++**	**Visual BASIC**	**Pascal (Borland Delphi)**
u8	8-bit ASCII character	0 to 255	char	Not supported by BASIC. For functions that require character arrays, use string types instead.	Byte
i16	16-bit signed integer	-32,768 to 32,767	short	Integer (for example: deviceNum%)	SmallInt
u16	16-bit unsigned integer	0 to 65,535	unsigned short for 32-bit compilers	Not supported by BASIC. For functions that require unsigned integers, use the signed integer type instead. See the i16 description.	Word
i32	32-bit signed integer	-2,147,483,648 to 2,147,483,647	long	Long (for example: count&)	LongInt
u32	32-bit unsigned integer	0 to 4,294,967,295	unsigned long	Not supported by BASIC. For functions that require unsigned long integers, use the signed long integer type instead. See the i32 description.	Cardinal (in 32-bit operating systems). Refer to the i32 description.
f32	32-bit single-precision floating-point value	-3.402823E+38 to 3.402823E+38	float	Single (for example: num!)	Single
f64	64-bit double-precision floating-point value	-1.797685123862315E+308 to 1.797685123862315E+308	double	Double (for example: voltage Number)	Double

Table 2

6.3 Programming language considerations

Apart from the data type differences, there are a few language-dependent considerations you need to be aware of when you use the EMA8314 API. Read the following sections that apply to your programming language.

Note: Be sure to include the declaration functions of EMA8314 prototypes by including the appropriate EMA8314 header file in your source code. Refer to Chapter 4. EMA8314 Language Support for the header file appropriate to your compiler.

6.3.1 C/C++

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

```
Status = EMA8314_output_read (u32 CardID, u8 port, u8 *value);
```

where **CardID** and **port** are input parameters, and **value** is an output parameter.
To use the function in C language, consider the following example:

```
u32 CardID=0, port=0 ;    //assume CardID is 0 and port also 0
u8 value,
u32 Status;
Status = EMA8314_port_read ( CardID, port, &value);
```

6.3.2 Visual basic

The file EMA8314.bas contains definitions for constants required for obtaining LSI Card information and declared functions and variable as global variables. You should use these constants symbols in the EMA8314.bas, do not use the numerical values.
In Visual Basic, you can add the entire EMA8314.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 EMA8314.bas file for your project in Visual Basic 4.0, go to the **File** menu and select the **Add File...** **option**. Select EMA8314.bas, which is browsed in the EMA8314 \ api directory. Then, select **Open** to add the file to the project.

To add the EMA8314.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** EMA8314.bas, which is in the EMA8314 \api directory. Then, select **Open** to add the file to the project.

If you want to use under .NET environment, please download “

6.3.3 Borland C++ builder

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

implib EMA8314bc.lib EMA8314.dll

Then add the **EMA8314bc.lib** to your project and add

#include “EMA8314.h” to main program.

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

```
Status = EMA8314_output_read ( CardID, port, &value);
```

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

```
u32 CardID=0, port=0 ;    //assume CardID is 0 and port also 0
u8 value,
u32 Status;
Status = EMA8314_port_read ( CardID, port, &value);
```

* If you are using Delphi, please refer to **http://www.drbob42.com/headconv/index.htm** for more detail about the difference of C++ and Delphi.

7. Software overview and dll function

These topics describe the features and functionality of the EMA8314 module and the detail of the dll function.

7.1 Initialization and close

You need to initialize system resource, port and IP each time you start to run your application,

EMA8314_initial ( ) will do.

Once you want to close your application, call

EMA8314_close ( ) to release all the resource.

To check the firmware version,

EMA8314_firmware_version_read ( ) will do.

- **EMA8314_initial**

Format : u32 status =EMA8314_initial(u32 CardID,u8 IP_Address[4] , u16 Host_Port, u16 Remote_port, u16 TimeOut, u8 *CardType)

Purpose: To map IP and port of an existing EMA8314 to a specified CardID number**.**

Parameters:

Input:

Name	Type	Description
CardID	u32	Range : 0 ~ 1999 Assign CardID to the EMA8314 of a corresponding IP address.
IP_Address[4]	u8	4 words of IP address, Default:192.168.0.100 For example : if IP address is “192.168.0.100” then IP_Address[0]=192 IP_Address[1]=168 IP_Address[2]=0 IP_Address[3]=100
Host_Port	u16	Assign a communicate port of host PC Default : 15120
Remote_port	u16	Assign a communicate port of EMA8314 Default : 6936
TimeOut	u16	Assign the max delay time of EMA8314 response message,1000~10000 ms.

Output:

Name	Type	Description
CardType	u8	Get the Card Type of EMA8314 1 : EMA-8314R 2 : EMA-8314T

- **EMA8314_close**

Format : **u32 status =EMA8314_close(u32 CardID)**

Purpose: Release the EMA8314 resource when closing the Windows applications.

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMA8314_IP_mapping

- **EMA8314_firmware_version_read**

Format : **u32 status =EMA8314_firmware_version_read(u32 CardID, u8 version[2])**

Purpose: Read the firmware version.

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMA8314_initial

Output:

Name	Type	Description
version[2]	u8	The firmware version x.y x = version[1] y = version[0]

7.2 Output function

The EMA8314 module has 4 relay output (OUT0~OUT3), it can work as general purpose output or controlled under the temperature condition. First of all, you must decide its function. Choose to work as general-purpose output, you can command it on or off as you need any time, if it works as temperature control output, it will be controlled under the temperature condition (the module work as temperature controller).

To configure the output as general purpose or temperature related function by:

EMA8314_output_mode_set ( ) and read back the configuration by:

EMA8314_output_mode_read ( )

To control the output (as general purpose output), use

EMA8314_output_set ( )

To read output register status use

EMA8314_output_read ( )

- **EMA8314_output_mode_set**

Format : u32 status =EMA8314_output_mode_set(u32 CardID,u8 port,u8 mode)

Purpose: To set output mode.

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMA8314_initial function
port	u8	always 0
mode	u8	bit0: 0: OUT0 as general purpose output (default) 1: OUT0 as temperature control output bit1: 0: OUT1 as general purpose output (default) 1: OUT1 as temperature control output bit2: 0: OUT2 as general purpose output (default) 1: OUT2 as temperature control output bit3: 0: OUT3 as general purpose output (default) 1: OUT3 as temperature control output

- **EMA8314_output_mode_read**

Format : **u32 status =EMA8314_output_mode_read(u32 CardID,u8 port,u8 *mode)**

Purpose: To read output mode.

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMA8314_initial function
port	u8	always 0

Output:

Name	Type	Description
mode	u8	bit0: 0: OUT0 as general purpose output (default) 1: OUT0 as temperature control output bit1: 0: OUT1 as general purpose output (default) 1: OUT1 as temperature control output bit2: 0: OUT2 as general purpose output (default) 1: OUT2 as temperature control output bit3: 0: OUT3 as general purpose output (default) 1: OUT3 as temperature control output

- **EMA8314_output_set**

Format : **u32 status =EMA8314_output_set(u32 CardID,u8 port,u8 value)**

Purpose: To set output status*.

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMA8314_initial function
port	u8	always 0
value	u8	bit 0 ~ bit 3 : channel 0 ~ channel 3 status of output 0: output de-active 1: output active

*** Only valid for output work as general-purpose output.**

- **EMA8314_output_read**

Format : **u32 status =EMA8314_output_read(u32 CardID,u8 port,u8 *value)**

Purpose: To read output status.

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMA8314_initial function
port	u8	always 0

Output:

Name	Type	Description
value	u8	bit 0 ~ bit 3 : channel 0 ~ channel 3 status of output 0: output de-active 1: output active

7.3 Sensor setup

The module can use PT100 and PT1000 as sensor input, to configure for correct temperature conversion; you need to tell the module which kind of sensor connects to the input.

To set one channel sensor type by

EMA8314_channel_sensor_type_set ( )

To read one channel sensor type by

EMA8314_channel_sensor_type_read ( )

To set one port sensor type by

EMA8314_port_sensor_type_set ( )

To read one port sensor type by

EMA8314_port_sensor_type_read ( )

You can also check the sensor status by:

EMA8314_port_sensor_status_read ( )

- **EMA8314_channel_sensor_type_set**

Format : u32 status =EMA8314_channel_sensor_type_set(u32 CardID,u8 port , u8 channel,u8 type)

Purpose: To set one channel sensor type.

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMA8314_initial function
port	u8	always 0
channel	u8	0 ~ 3 = channel 0 ~ channel 3
type	u8	1 : Pt-1000 (default) 2 : Pt-100

- **EMA8314_channel_sensor_type_read**

Format : **u32 status =EMA8314_channel_sensor_type_read(u32 CardID,u8 port , u8 channel,u8 *type)**

Purpose: To read one channel sensor type.

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMA8314_initial function
port	u8	always 0
channel	u8	0 ~ 3 = channel 0 ~ channel 3

Output:

Name	Type	Description
type	u8	1 : Pt-1000 (default) 2 : Pt-100

- **EMA8314_port_sensor_type_set**

Format : **u32 status =EMA8314_port_sensor_type_set(u32 CardID,u8 port ,u8 type[4])**

Purpose: To set one channel sensor type.

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMA8314_initial function
port	u8	always 0
type[4]	u8	type[0]: channel 0 =1 : Pt-1000 (default) =2 : Pt-100 … type[3]: channel 3 =1 : Pt-1000 (default) =2 : Pt-100

- **EMA8314_port_sensor_type_read**

Format : u32 status =EMA8314_port_sensor_type_read(u32 CardID,u8 port, u8 *type[4])

Purpose: To read one channel sensor type.

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMA8314_initial function
port	u8	always 0

Output:

Name	Type	Description
type[4]	u8	type[0]: channel 0 =1 : Pt-1000 (default) =2 : Pt-100 … type[3]: channel 3 =1 : Pt-1000 (default) =2 : Pt-100

- **EMA8314_port_sensor_status_read**

Format : u32 status =EMA8314_port_sensor_status_read(u32 CardID,u8 port, u8 *status)

Purpose: To check sensors status.

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMA8314_initial function
port	u8	always 0

Output:

Name	Type	Description
status	u8	Bit 0: channel 0 =0 : normal =1 : broken … Bit 3: channel 3 =0 : normal =1 : broken

Note: On standalone control mode, any sensor line broken will make the power LED flick at 3 Hz speed to signal error.

7.4 Temperature control function

The EMA-8314 module provide build-in temperature control, which you can just configure it as one of the following modes:

mode 0: Over high temperature limit ON, under low temperature limit OFF
(It is generally used in cooler control, air condition control)

mode 1: Over high temperature limit OFF, under low temperature limit ON
(It is generally used in heater control)

mode 2: Within high temperature limit and low temperature limit ON else OFF

mode 3: Within high temperature limit and low temperature limit OFF else ON
(Mode2 and mode3 is generally used in temperature monitoring and alarm system)

To set one channel limit data by

EMA8314_channel_temperature_limit_set ( )

To read one channel limit data by

EMA8314_channel_temperature_limit_read ( )

To set one port limit data by

EMA8314_port_temperature_limit_set ( )

To read one port limit data by

EMA8314_port_temperature_limit_read ( )

To set control mode by

EMA8314_control_mode_set ( )

To read control mode by

EMA8314_control_mode_read ( )

To set all channel temperature mask by

EMA8314_control_mask_set ( )

To read all channel temperature mask by

EMA8314_control_mask_read ( )

To enable the temperature comparison by

EMA8314_control_enable ( )

To disable the temperature comparison by

EMA8314_control_disable ( )

To read control status by

EMA8314_control_status_read ( )

To read the one channel analog data by

EMA8314_channel_temperature_read ( )

To read all analog data by

EMA8314_port_temperature_read ( )

- **EMA8314_channel_temperature_limit_set**

Format : u32 status =EMA8314_channel_temperature_limit_set(u32 CardID,u8 port, u8 channel,f32 temperature[2],u8 type)

Purpose: To set one channel limit data.

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMA8314_initial function
port	u8	always 0
channel	u8	0 ~ 3 = channel 0 ~ channel 3
temperature[2]	f32	limit of temperature temperature[0]: low temperature limit temperature[1]: high temperature limit
type	u8	0: Centigrade (default) 1: Fahrenheit

- **EMA8314_channel_temperature_limit_read**

Format : u32 status =EMA8314_channel_temperature_limit_read(u32 CardID,u8 port, u8 channel,f32 temperature[2],u8 *type)

Purpose: To read one channel limit data.

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMA8314_initial function
port	u8	always 0
channel	u8	0 ~ 3 = channel 0 ~ channel 3

Output:

Name	Type	Description
temperature[2]	f32	limit of temperature temperature[0]: low temperature limit temperature[1]: high temperature limit
type	u8	0: Centigrade (default) 1: Fahrenheit

- **EMA8314_port_temperature_limit_set**

Format : u32 status =EMA8314_port_temperature_limit_set(u32 CardID,u8 port, f32 temperature[8],u8 type[4])

Purpose: To set one port (all channels) limit data.

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMA8314_initial function
port	u8	always 0
temperature[8]	f32	temperature[0] : channel 0 low temperature limit temperature[1] : channel 0 high temperature limit temperature[2] : channel 1 low temperature limit temperature[3] : channel 1 high temperature limit temperature[4] : channel 2 low temperature limit temperature[5] : channel 2 high temperature limit temperature[6] : channel 3 low temperature limit temperature[7] : channel 3 high temperature limit
type[4]	u8	type[0]: 0: centigrade of channel 0 (default) 1: Fahrenheit of channel 0 … type[3]: 0: centigrade of channel 3 (default) 1: Fahrenheit of channel 3

- **EMA8314_port_temperature_limit_read**

Format : u32 status =EMA8314_port_temperature_limit_read(u32 CardID,u8 port, f32 temperature [8],u8 type[4])

Purpose: To read one port (all channels) limit data.

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMA8314_initial function
port	u8	always 0

Output:

Name	Type	Description
temperature[8]	f32	temperature[0] : channel 0 low temperature limit temperature[1] : channel 0 high temperature limit temperature[2] : channel 1 low temperature limit temperature[3] : channel 1 high temperature limit temperature[4] : channel 2 low temperature limit temperature[5] : channel 2 high temperature limit temperature[6] : channel 3 low temperature limit temperature[7] : channel 3 high temperature limit
type[4]	u8	type[0]: 0: centigrade of channel 0 (default) 1: Fahrenheit of channel 0 … type[3]: 0: centigrade of channel 3 (default) 1: Fahrenheit of channel 3

- **EMA8314_control_mode_set**

Format : **u32 status =EMA8314_control_mode_set(u32 CardID,u8 port, u8 channel, u8 mode)**

Purpose: To setup one channel limit data.

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMA8314_initial function
port	u8	always 0
channel	u8	0 ~ 3 = channel 0 ~ channel 3
mode	u8	mode 0: Over high temperature limit ON, under low temperature limit OFF mode 1: Over high temperature limit OFF, under low temperature limit ON mode 2: Within high temperature limit and low temperature limit ON else OFF mode 3: Within high temperature limit and low temperature limit OFF else ON

Note: Channel0 control output on OUT0, … channel3 control output on OUT3

- **EMA8314_control_mode_read**

Format : **u32 status =EMA8314_control_mode_read(u32 CardID,u8 port, u8 channel, u8 *mode)**

Purpose: To read back the channel control mode.

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMA8314_initial function
port	u8	always 0
channel	u8	0 ~ 3 = channel 0 ~ channel 3

Output:

Name	Type	Description
mode	u8	mode 0: Over high temperature limit ON, under low temperature limit OFF mode 1: Over high temperature limit OFF, under low temperature limit ON mode 2: Within high temperature limit and low temperature limit ON else OFF mode 3: Within high temperature limit and low temperature limit OFF else ON

- **EMA8314_control_mask_set**

Format : **u32 status =EMA8314_control_mask_set(u32 CardID,u8 port,u8 value)**

Purpose: To mask off channel temperature comparison.

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMA8314_initial function
port	u8	always 0
value	u8	bit 0 ~ bit 3 : channel 0 ~ channel 3 0 = unmask (activate operation) 1 = mask off (no operation)

- **EMA8314_control_mask_read**

Format : **u32 status =EMA8314_control_mask_read(u32 CardID,u8 port,u8 *value)**

Purpose: To read unmask channel temperature comparison.

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMA8314_initial function
port	u8	always 0

Output:

Name	Type	Description
value	u8	bit 0 ~ bit 3 : channel 0 ~ channel 3 0 = unmask (activate operation) 1 = mask off (no operation)

- **EMA8314_control_enable**

Format : **u32 status =EMA8314_control_enable(u32 CardID)**

Purpose: To enable the temperature comparison control.

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMA8314_initial function

- **EMA8314_control_disable**

Format : **u32 status =EMA8314_control_disable(u32 CardID)**

Purpose: To disable the temperature comparison control.

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMA8314_initial function

- **EMA8314_control_status_read**

Format : **u32 status = EMA8314_control_status_read(u32 CardID,u8 *status)**

Purpose: To read temperature comparison status.

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMA8314_initial function

Output:

Name	Type	Description
status	u8	1 = disable temperature comparison 2 = enable temperature comparison

- **EMA8314_channel_temperature_read**

Format : **u32 status =EMA8314_channel_temperature_read(u32 CardID, u8 port, u8 channel, f32 *temperature, u8 *type)**

Purpose: To read one channel temperature data.

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMA8314_initial function
port	u8	always 0
channel	u8	0 = channel 0 1 = channel 1 2 = channel 2 3 = channel 3

Output:

Name	Type	Description
temperature	f32	measured temperature
type	u8	0: Centigrade (default) 1: Fahrenheit

- **EMA8314_port_temperature_read**

Format : **u32 status =EMA8314_port_temperature_read(u32 CardID, u8 port, f32 temperature[4], u8 type[4])**

Purpose: To read one port (all channel) A/D data.

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMA8314_initial function
port	u8	always 0

Output:

Name	Type	Description
temperature[4]	f32	measured temperature
type[4]	u8	0: Centigrade (default) 1: Fahrenheit

7.5 WDT (watch dog timer) function

In the industrial environment, we want the controller work as stable as possible but we are not God; we can not always put the controller by guess. To ensure the controller will not harm to the system or human, people always put a WDT to monitor the controller, if the controller run in abnormal state the system will fail to reset WDT then WDT will latch the system to prevent further harm. EMA8314 also provide the WDT function, which will detect the Ethernet connection state, once the connection is fail for a predefined period, the module will output the predefined status to the ports. You can enable or disable as your application required.

Use

EMA8314_WDT_set( ) to set up the WDT timer and the output state if the Ethernet connection fail to communicate.

EMA8314_WDT_read( ) to read back the configuration.

To enable the WDT function to monitor the communication (you must periodically communicate with the remote I/O module to keep it alive) by:

EMA8314_WDT_enable( ) and disable by:

EMA8314_WDT_disable( ).

- **EMA8314_WDT_set**

Format : u32 status = EMA8314_WDT_set(u32 CardID,u8 output_status,u16 wait_time)

Purpose: Set WDT (watch dog timer) configuration.

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMA8314_initial
output_status	u8	Set the output default state, the state will keep while the connection failed. bit 0 ~ bit 3 : OUT0 ~ OUT3
wait_time	u16	Set the WDT wait time (10~10000) based on 0.1 sec time base. default: 10 (1s)

- **EMA8314_WDT_read**

Format : **u32 status = EMA8314_WDT_read(u32 CardID,u8 *output_status, u16 *wait_time,u8 *wdt_status)**

Purpose: Read back WDT (watch dog timer) configuration.

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMA8314_initial

Output:

Name	Type	Description
output_status	u8	bit 0 ~ bit 3 : OUT0 ~ OUT3
wait_time	u16	Read the WDT wait time.
wdt_status	u8	1 : WDT disable 2 : WDT enable

- **EMA8314_WDT_enable**

Format : **u32 status = EMA8314_WDT_enable(u32 CardID)**

Purpose: Enable WDT (watch dog timer) .

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMA8314_initial

- **EMA8314_WDT_disable**

Format : **u32 status = EMA8314_WDT_disable(u32 CardID)**

Purpose: Disable WDT (watch dog timer) .

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMA8314_initial

7.6 Miscellaneous function

To change the communication port as you need by:

EMA8314_change_socket_port( ) [*1]

To change IP,

EMA8314_change_IP( ) [*1]

To reboot EMA8314 module for module alarm or to validate the system configuration change by:

EMA8314_reboot( ) [*1]

- **EMA8314_change_socket_port**

Format : u32 status =EMA8314_change_socket_port(u32 CardID,u16 Remote_port)

Purpose: To change the communicate port number of EMA8314.

After using this function, please wait for reboot (about 10s) to validate the change.

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMA8314_IP_mapping
Remote_port	u16	The new port number to be set

- **EMA8314_change_IP**

Format : u32 status =EMA8314_change_IP(u32 CardID, u8 IP[4])

Purpose: To change the communicate IP of EMA8314.

After using this function, please wait for reboot (about 10s) to validate the change.

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMA8314_IP_mapping
IP[4]	u8	The new IP to be set

- **EMA8314_reboot**

Format : u32 status =EMA8314_reboot(u32 CardID)

Purpose: To reboot EMA8314 (about 10s).

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMA8314_IP_mapping

7.7 Software key function

Software key is used to prevent the modification of IO state and system configuration by un-authorized person.

To operate the EMA8314, you must unlock the module first by

EMA8314_security_unlock( )

To verify the lock status by

EMA8314_security_status_read( )

You can change password for your convenience by

EMA8314_password_change( )

If you forget the password you set, you can recover the factory default password by:

EMA8314_password_set_default( ) *1

- **EMA8314_security_unlock**

Format : u32 status =EMA8314_security_unlock(u32 CardID,u8 password[8])

Purpose: To unlock security function and enable the further operation.

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMA8314_IP_mapping
password[8]	u8	The password previous set (ASCII) use a-z, A-Z, 0-9 characters. For example: u8 password[8] = {'1','2','3','4','5','6','7','8'}; u8 password[8] = {'1','2','3','a', 'A',NULL,NULL,NULL}; Default : password[8] = {'1','2','3','4','5','6','7','8'};

- **EMA8314_security_status_read**

Format : **u32 status =EMA8314_security_status_read(u32 CardID,u8 *lock_status)**

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

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMA8314_IP_mapping

Output:

Name	Type	Description
lock_status	u8	0 : security unlocked 1 : locked

- **EMA8314_password_change**

Format : **u32 status =EMA8314_password_change(u32 CardID, u8 Oldpassword[8], u8 password[8])**

Purpose: To replace old password with new password.

After using this function, please wait for reboot (about 10s) to validate the change.

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMA8314_IP_mapping
Oldpassword[8]	u8	The previous password(ASCII)
password[8]	u8	The new password to be set(ASCII)

- **EMA8314_password_set_default**

Format : **u32 status =EMA8314_password_set_default(u32 CardID)**

Purpose: Set password to default.

After using this function, please wait for reboot (about 10s) to validate the change.

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMA8314_IP_mapping Default : password[8] = {'1','2','3','4','5','6','7','8'};

8. Communication protocol

Although the dll have provide various function for the user, which enables the users to take the EMA module as if it is a non-ethernet I/O interface. But some users may want to coding their own software from the Ethernet basic functions, this chapter provides the detail of the communication protocol.

8.1 Host to module command format

IP header	UDP header	UDP data			
		Card_name	Password	Command	Data
20bytes	8bytes	7bytes	8bytes	1byte	32bytes

As shown above, the command format from PC to module is an UDP format, the first 20 bytes is the IP header, the next 8 byes is the UDP header then follows 48 bytes UDP data. The UDP data is defined as follows:

```
typedef struct _EMA8314_Send
{
    u8   card_name[7];          // card_name={‘E’,‘M’,‘A’,‘8’,‘3’,‘1’,‘4’ }
    u8   password[8];           // password（8 byte）
    u8   command;               // command（EMA8314 UDP COMMAND）
    Send_Data    Data;          // Send_Data
}EMA8314_Send;

typedef union Send_Data
{
    u8   data_b[32];            // data for byte
    //u16      data_w[16];      // data for word
    //u32      data_l[8];       // data for long
    u8   IP[4];                 // IP address
    u16  socket_port;           // socket port
    u8   new_password[8];       // new password, 8 words
    u8   MAC[6];                // MAC address
    Analog_Data Data_a;
    Digital_Data Data_d;
    No_Regulate_Data Data_n;
    Eeprom_Write_Read_Data Eep_data;
    WDT_Data    data_wdt;
}Send_Data;
```

```
typedef struct _Analog_Data
{
    u8      port[2];        // A/D port
    u8      channel[2];     // A/D channels
    f32     ad_data[4];     // A/D data
    u8      type[4];        // type
    u8      status[4];      // status
}Analog_Data;

typedef struct _Digital_Data
{
    u8  port;               // port
    u8  value;              // channels
    u8  type;
} Digital_Data;

typedef struct _Eeprom_Write_Read_Data
{
    u8  read_write_data[20];    // datta
    u32 eeprom_address;         // address
    u32 eeprom_data_count;      // length
} Eeprom_Write_Read_Data

typedef struct _No_Regulate_Data
{
    u8  port[2];            // A/D port
    u8  channel[2];         // A/D channels
    u32 ad_data;            // A/D data
    u8  status;             // status
}No_Regulate_Data;

typedef struct _WDT_Data
{
    u16 wait_time;          // WDT time
    u8  output_status;      // output status
    u8  wdt_status;         // WDT status
}WDT_Data;
```

8.2 Module to host command format

IP header	UDP header	UDP data		
		Data	Flag	Command
20bytes	8bytes	32bytes	1byte	1byte

As shown above, the command format from module to host is an UDP format, the first 20 bytes is the IP header, the next 8 byes is the UDP header then follows 34 bytes UDP data. The UDP data is defined as follows:

```
typedef struct _EMA8314_Receive
{
    Receive_Data Data;      // Receive_Data
    u8   success_flag;          // flag：  0 = send command failed
                                        99 = send command successfully
    u8   command;               // command（EMA8314 UDP COMMAND）
}EMA8314_Receive;
```

The Receive_Data is defined as:

```
typedef union Receive_Data
{
    u8   data_b[32];            // data for byte
    //u16      data_w[16];      // data for word
    //u32      data_l[8];       // data for long
    u8   card_type;             // card_type： 0x1=RTD,
                                //             0x2=Thermocouple
    u8   version[2];            // firmware version
    Analog_Data Data_a;
    Digital_Data Data_d;
    No_Regulate_Data Data_n;
    Eeprom_Write_Read_Data Eep_data;
    WDT_Data    data_wdt;
}Receive_Data;
```

8.3 Definition of IP header

The IP header is defined as follows:

```
struct ipheader
{
      unsigned char ip_hl:4, ip_v:4; /* this means that each member is 4 bits */
      unsigned char ip_tos;    // type of service
      unsigned short int ip_len; //IP header total length
      unsigned short int ip_id; // identification
      unsigned short int ip_off; //fragment offset
      unsigned char ip_ttl; // time to live
      unsigned char ip_p; //protocol
      unsigned short int ip_sum; //header checksum
      unsigned int ip_src; //source ip address
      unsigned int ip_dst; //destination ip address
}; /* total ip header length: 20 bytes (=160 bits) */
```

8.4 Definition of UDP header

The UDP header is defined as follows:

```
struct udpheader
{
      unsigned short int uh_sport; // source port number
      unsigned short int uh_dport; //destination port number
      unsigned short int uh_len; // UDP package length
      unsigned short int uh_check; //UDP checksum
}; /* total udp header length: 8 bytes (=64 bits) */
```

8.5 EMA-8314 communication commands

- **GET_CARD_TYPE**

Function: ask the EMA8314 module type

Host to Ethernet module:

IP header	UDP header	UDP data			
		Card_name	Password	0x1	Data
20bytes	8bytes	7bytes	8bytes	1byte	32bytes

Card_name: ‘E’, ‘M’, ‘A’, ‘8’, ‘3’, ‘1’, ‘4’

Password: un-used

Command: 0x1

Data: un-used

Ethernet module to Host:

IP header	UDP header	UDP data		
		Data	Flag	0x1
20bytes	8bytes	32bytes	1byte	1byte

Command: 0x1

Flag: = 0x0 //command fail (this is not EMD8216)

= 0x63 //command successful (this is EMD8216)

Data: un-used

- **REBOOT**

Function: reboot EMA8314 module

Host to Ethernet module:

IP header	UDP header	UDP data			
		Card_name	Password	0x2	Data
20bytes	8bytes	7bytes	8bytes	1byte	32bytes

Card_name: ‘E’, ‘M’, ‘A’, ‘8’, ‘3’, ‘1’, ‘4’

Password: un-used

Command: 0x2

Data: un-used

Ethernet module to Host:

IP header	UDP header	UDP data		
		Data	Flag	0x2
20bytes	8bytes	32bytes	1byte	1byte

Command: 0x2

Flag: = 0x0 //command fail

= 0x63 //command successful

Data: un-used

- **CHANGE_SOCKETPORT**

Function: change socket port of EMA8314 module

Host to Ethernet module:

IP header	UDP header	UDP data			
		Card_name	Password	0x3	Data
20bytes	8bytes	7bytes	8bytes	1byte	32bytes

Card_name: ‘E’, ‘M’, ‘A’, ‘8’, ‘3’, ‘1’, ‘4’

Password: your password

Command: 0x3

Data: socket_port //your new socket port number

other structure members are un-used.

Parameter	Type	Description
socket_port	u16	socket port number

Ethernet module to Host:

IP header	UDP header	UDP data		
		Data	Flag	0x3
20bytes	8bytes	32bytes	1byte	1byte

Command: 0x3

Flag: = 0x0 //command fail

= 0x63 //command successful

Data: // unused

- **CHANGE_PASSWORD**

Function: change password of EMA8314 module

Host to Ethernet module:

IP header	UDP header	UDP data			
		Card_name	Password	0x4	Data
20bytes	8bytes	7bytes	8bytes	1byte	32bytes

Card_name: 'E', 'M', 'A', '8', '3', '1', '4'

Password: your password （8 bytes）

Command: 0x4

Data: password[8] //your new password

other structure members are un-used.

Parameter	Type	Description
password[8]	u8	new password to be set

Ethernet module to Host:

IP header	UDP header	UDP data		
		Data	Flag	0x4
20bytes	8bytes	32bytes	1byte	1byte

Command: 0x4

Flag: = 0x0 //command fail

= 0x63 //command successful

Data: // unused

- **RESTORE_PASSWORD**

Function: restore password of EMA8314 module

Host to Ethernet module:

IP header	UDP header	UDP data			
		Card_name	Password	0x5	Data
20bytes	8bytes	7bytes	8bytes	1byte	32bytes

Card_name: ‘E’, ‘M’, ‘A’, ‘8’, ‘3’, ‘1’, ‘4’

Password: your password （8 bytes）

Command: 0x5

Data: un-used.

Ethernet module to Host:

IP header	UDP header	UDP data		
		Data	Flag	0x5
20bytes	8bytes	32bytes	1byte	1byte

Command: 0x5

Flag: = 0x0 //command fail

= 0x63 //command successful

Data: // unused

- **CHANGE_IP**

Function: change IP of EMA8314 module

Host to Ethernet module:

IP header	UDP header	UDP data			
		Card_name	Password	0x6	Data
20bytes	8bytes	7bytes	8bytes	1byte	32bytes

Card_name: ‘E’, ‘M’, ‘A’, ‘8’, ‘3’, ‘1’, ‘4’

Password: your password （8 bytes）

Command: 0x6

Data: IP[4] //new IP address

other structure members are un-used.

Parameter	Type	Description
IP[4]	u8	new IP address to be set for example: 192.168.0.5 IP[0]=192 IP[1]=168 IP[2]=0 IP[3]=5

Ethernet module to Host:

IP header	UDP header	UDP data		
		Data	Flag	0x6
20bytes	8bytes	32bytes	1byte	1byte

Command: 0x6

Flag: = 0x0 //command fail

= 0x63 //command successful

Data: // unused

- **GET_FIRMWARE_VERSION**

Function: read the firmware version of EMA8314 module

Host to Ethernet module:

IP header	UDP header	UDP data			
		Card_name	Password	0x7	Data
20bytes	8bytes	7bytes	8bytes	1byte	32bytes

Card_name: ‘E’, ‘M’, ‘A’, ‘8’, ‘3’, ‘1’, ‘4’

Password: your password （8 bytes）

Command: 0x7

Data: un-used

Ethernet module to Host:

IP header	UDP header	UDP data		
		Data	Flag	0x7
20bytes	8bytes	32bytes	1byte	1byte

Command: 0x7

Flag: = 0x0 //command fail

= 0x63 //command successful

Data: version[0 ~ 1] // firmware version

Parameter	Type	Description
version[1] version[0]	u16	version x.y version[1]: x version[0]: y

- **WRITE_MAC**

Function: the MAC address of EMA8314 module

Host to Ethernet module:

IP header	UDP header	UDP data			
		Card_name	Password	0xfa	Data
20bytes	8bytes	7bytes	8bytes	1byte	32bytes

Card_name: ‘E’, ‘M’, ‘A’, ‘8’, ‘3’, ‘1’, ‘4’

Password: your password （8 bytes）

Command: 0xfa

Data: MAC[6] //new MAC address

other structure members are un-used.

Parameter	Type	Description
MAC[6]	u8	MAC address to be set Say mac=112233445566 MAC[0]=11 .. MAC[5]=66

Ethernet module to Host:

IP header	UDP header	UDP data		
		Data	Flag	0x7
20bytes	8bytes	32bytes	1byte	1byte

Command: 0xfa

Flag: = 0x0 //command fail

= 0x63 //command successful

Data: un-used

9. DLL list

	Function Name	Description
1.	EMA8314_Initial( )	Map IP and get model parameter
2.	EMA8314_close( )	EMA8314 close
3.	EMA8314_firmware_version_read( )	Read the modules' firmware information
4.	EMA8314_output_mode_set( )	To set output mode
5.	EMA8314_output_mode_read( )	To read output mode
6.	EMA8314_output_set( )	To set the output value
7.	EMA8314_output _read( )	To read the output value
8.	EMA8314_channel_sensor_type_set( )	To set one channel sensor type
9.	EMA8314_channel_sensor_type_read( )	To read one channel sensor type
10.	EMA8314_port_sensor_type_set( )	To set one port sensor type
11.	EMA8314_port_sensor_type_read( )	To read one port sensor type
12.	EMA8314_port_sensor_status_read( )	To read break detection status
13.	EMA8314_channel_temperature_limit_set( )	To set a channel temperature comparison
14.	EMA8314_channel_temperature_limit_read( )	To read a channel temperature comparison
15.	EMA8314_port_temperature_limit_set( )	To set one port temperatures comparison
16.	EMA8314_port_temperature_limit_read( )	To read one port temperatures comparison
17.	EMA8314_control_mode_set( )	To set one port control mode
18.	EMA8314_control_mode_read( )	To read one port control mode
19.	EMA8314_control_mask_set( )	To set temperature control mask
20.	EMA8314_control_mask_read( )	To read temperature control mask
21.	EMA8314_control_enable( )	Enable temperature comparison
22.	EMA8314_control_disable( )	Disable temperature comparison
23.	EMA8314_control_status_read( )	To read temperature comparison status
24.	EMA8314_channel_temperature_read( )	To read one channel temperature data
25.	EMA8314_port_temperature_read( )	To read one port temperature data
26.	EMA8314_WDT_set( )	Set WDT (watch dog timer) configuration
27.	EMA8314_WDT_read( )	Read back WDT (watch dog timer) configuration
28.	EMA8314_WDT_enable( )	Enable WDT (watch dog timer)
29.	EMA8314_WDT_disable( )	Disable WDT (watch dog timer)
30.	EMA8314_change_socket_port( )	To change the communicate port number of EMA8314
31.	EMA8314_change_IP( )	To change the communicate IP of EMA8314
32.	EMA8314_reboot( )	To reboot EMA8314

33.	EMA8314_security_unlock( )	Unlock security
34.	EMA8314_security_status_read( )	Read lock status
35.	EMA8314_password_change( )	Change password
36.	EMA8314_password_set_default( )	Rest to factory default password

10. EMA8314 Error codes summary

10.1 EMA8314 Error codes table

Error Code	Symbolic Name	Description
0	JSDRV_NO_ERROR	No error.
1	INITIAL_SOCKET_ERROR	Sock can not initialized, maybe Ethernet hardware problem
2	IP_ADDRESS_ERROR	IP address is not acceptable
3	UNLOCK_ERROR	Unlock fail
4	LOCK_COUNTER_ERROR	Unlock error too many times
5	SET_SECURITY_ERROR	Fail to set security
100	DEVICE_RW_ERROR	Can not reach module
101	NO_CARD	Can not reach module
102	DUPLICATE_ID	CardID already used
300	ID_ERROR	CardID is not acceptable
301	PORT_ERROR	Port parameter unacceptable or unreachable
302	IN_POINT_ERROR	Input point unreachable
303	OUT_POINT_ERROR	Output point unreachable
305	PARAMETERS_ERROR	Parameter error
306	CHANGE_SOCKET_ERROR	Can not change socket
307	UNLOCK_SECURITY_ERROR	Fail to unlock security
308	PASSWORD_ERROR	Password mismatched
309	REBOOT_ERROR	Can not reboot
310	TIME_OUT_ERROR	Too long to response
311	CREAT_SOCKET_ERROR	Socket can not create
401	POWER_12V_ERROR	Power detection error