EMD-8216
Ethernet Digital I/O module

Software Manual (V1.2)

健昇科技股份有限公司

JS AUTOMATION CORP.

新北市汐止區中興路 100 號 6 樓

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

Xizhi Dist., New Taipei City, Taiwan

TEL：+886-2-2647-6936

FAX：+886-2-2647-6940

http://www.automation.com.tw

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

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

Correction record

Version	Record
1.0	EMD-8216.dll v1.0
	for driver 1.2up
1.0 -> 1.1	1. **EMD8216_standalone_config_set** and and **EMD8216_standalone_config_read** add a variable number 2. Modify the IO state description: change ‘low’ to ‘inactive’ and ‘high’ to ‘active’
1.1->1.2	1. add MODBus function and descriptions

Contents

1. How to install the software of EMD8216 5
   1.1 Install the EMD driver 5
2. Where to find the file you need 6
3. About the EMD8216 software 7
   3.1 What you need to get started 7
   3.2 Software programming choices 7
4. EMD8216 Language support 8
   4.1 Building applications with the EMD8216 software library 8
5. Basic concept of the remote digital I/O module 9
6. Function format and language difference 11
   6.1 Function format 11
   6.2 Variable data types 12
   6.3 Programming language considerations 13
7. Software overview and dll function 15
   7.1 Initialization and close 15
      EMD8216_initial 15
      EMD8216_close 16
      EMD8216_firmware_version_read 16
   7.2 Digital I/O 17
      EMD8216_port_config_set 18
      EMD8216_port_config_read 18
      EMD8216_port_polarity_set 19
      EMD8216_port_polarity_read 19
      EMD8216_port_set 20
      EMD8216_port_read 20
      EMD8216_point_config_set 21
      EMD8216_point_config_read 21
      EMD8216_point_polarity_set 22
      EMD8216_point_polarity_read 22
      EMD8216_point_set 23
      EMD8216_point_read 23
   7.3 Counter function 24
      EMD8216_counter_mask_set 24
      EMD8216_counter_enable 25
      EMD8216_counter_disable 25
      EMD8216_counter_read 25
      EMD8216_counter_clear 26
   7.4 Miscellaneous function 27
      EMD8216_change_socket_port 27

EMD8216_change_IP 27
EMD8216_reboot 28
7.5 Software key function 29
EMD8216_security_unlock 29
EMD8216_security_status_read 30
EMD8216_password_change 30
EMD8216_password_set_default 30
7.6 WDT (watch dog timer) 31
EMD8216_WDT_set 31
EMD8216_WDT_read 32
EMD8216_WDT_enable 32
EMD8216_WDT_disable 32
7.7 Standalone function 33
EMD8216_standalone_enable 33
EMD8216_standalone_disable 33
EMD8216_standalone_config_set 34
EMD8216_standalone_config_read 36
7.8 Error codes and address 38
8. Standalone mode user configuration utility 39
8.1 Overview of user configuration utility 39
8.2 Configure a command 40
8.3 Edit function 43
8.4 Upload program 44
8.5 Download program 44
8.6 Save and load program with PC 45
8.7 Enable/Disable standalone function 45
9. Standalone mode application examples 46
9.1 Monitoring input if condition meets, trigger output 46
9.2 Monitoring the input if condition meets, delay to trigger output 47
9.3 Monitoring the input if condition meets, output pulse 48
9.4 Monitoring the input if condition meets, output periodically and stop by some special input condition 49
9.5 Don’t care the input if standalone enabled, trigger output 51
9.6 Don’t care the input if standalone enabled, trigger pulse 52
9.7 Don’t care the input if standalone enabled, output periodically 53
10. Communication protocol 54
10.1 Host to module command format 54
10.2 Module to host command format 55
10.3 Definition of IP header 56
10.4 Definition of UDP header 56
10.5 EMD-8216 communication commands 57

GET_CARD_TYPE....57
REBOOT....58
CHANGE_SOCKETPORT....59
CHANGE_PASSWORD....60
RESTORE_PASSWORD....61
CHANGE_IP....62
GET_FIRMWARE_VERSION....63
WRITE_MAC....64
SET_COUNTER_MASK....65
ENABLE_COUNTER_MODE....66
DISABLE_COUNTER_MODE....67
READ_COUNTER....68
CLEAR_COUNTER....69
SET_PORT_CONFIG....70
READ_PORT_CONFIG....71
SET_PORT....72
READ_PORT....73
SET_POLARITY....74
READ_POLARITY....75
SET_POINT_CONFIG....76
READ_POINT_CONFIG....77
SET_POINT....78
READ_POINT....79
SET_POINT_POLARITY....80
READ_POINT_POLARITY....81
ENABLE_STANDALONE....82
DISABLE_STANDALONE....83
SET_STANDALONE_CONFIG....84
READ_STANDALONE_CONFIG....86
ENABLE_WDT....88
DISABLE_WDT....89
SET_WDT....90
READ_WDT....91
11. DLL list....92
12. EMD8216 Error codes summary....93
12.1 EMD8216 Error codes table....93
13. UDP communication command list....94
14. UDP Error codes summary....95
15. MODBus....96

1. How to install the software of EMD8216

Please register as user's club member to download the
"EMD series Ethernet IO modules installation guide" document from http://automation.com.tw

1.1 Install the EMD driver

The ether net module cannot found by OS as PCI cards. You can just install the driver without the module installed. Execute the file ..\install\EMD8216_Install.exe to install the driver, Api and demo program automatically.

For a more detail descriptions, please refer "EMD series Ethernet IO modules installation guide".

2. Where to find the file you need

Windows XP and up

In Windows XP and later system, the demo program can be setup by EMD8216_Install.exe.

If you use the default setting, a new directory .. \JS Automation\EMD8216 will generate to put the associate files.

.. / JS Automation /EMD8216/API (header files and VB,VC lib files)

.. / JS Automation /EMD8216/Driver (copy of driver code)

.. / JS Automation /EMD8216/exe (demo program and source code)

The dll is located at ..\system.

3. About the EMD8216 software

EMD8216 software includes a set of dynamic link library (DLL) based on socket that you can utilize to control the interface functions.

Your EMD8216 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 EMD8216 functions within Windows' operation system environment.

3.1 What you need to get started

To set up and use your EMD8216 software, you need the following:

- EMD8216 software
- EMD8216 hardware

3.2 Software programming choices

You have several options to choose from when you are programming EMD8216 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 EMD8216 software.

4. EMD8216 Language support

The EMD8216 software library is a DLL used with Windows XP and later system. You can use these DLL with any Windows integrating development environment that can call Windows DLLs.

4.1 Building applications with the EMD8216 software library

The EMD8216 function reference section contains general information about building EMD8216 applications, describes the nature of the EMD8216 functions used in building EMD8216 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.

EMD8216 Windows Libraries

The EMD8216 for Windows function library is a DLL called **EMD8216.dll**. Since a DLL is used, EMD8216 functions are not linked into the executable files of applications. Only the information about the EMD8216 functions in the EMD8216 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 EMD8216 functions in EMD8216 .dll.

Header Files and Import Libraries for Different Development Environments		
Development Environment	Header File	Import Library
Microsoft C/C++	EMD8216.h	EMD8216VC.lib
Borland C/C++	EMD8216.h	EMD8216BC.lib
Microsoft Visual Basic	EMD8216.bas	

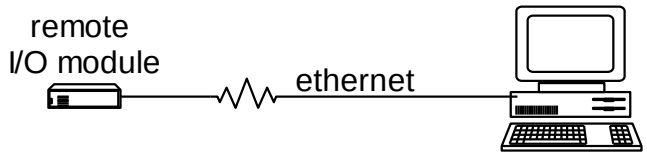
Table 1

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

I/O communicate via Ethernet

The remote digital I/O is the function extension of the card type digital I/O. 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 digital I/O will do.

JS automation keeps the remote digital I/O function as close to the card type digital I/O as possible. Users can port their application from card type to remote or from remote to card at the shortest working time.

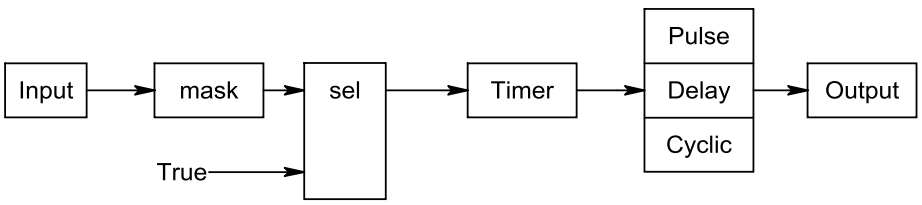


The module response or be commanded by the controller through Ethernet by UDP protocol. As a member on the Ethernet, it must have a distinguished IP and a predefined port. At factory, we set the default IP at 192.168.0.100 and the port at 6936 for the remote module. In spite of the IP addresses are same on the products shipped, JS Automation do provide a configuration program which enables you to connect the modules (without configured). The program scans all the on line modules and list the IP and socket. You can change on them on line. (refer the Ethernet Configuration using the EasyConfig software)

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.

Standalone I/O controller

The upgrade version of EMD8216 module has add new function to work as Ethernet controlled standalone controller, it can be download program or monitoring via Ethernet in standalone mode or fully controller via Ethernet. The EMD8216 provide internal timer to incorporate with the input and output that enables you to assign output delay or pulse on special input condition.



On the above figure, input condition can be mask or set to “True” to trigger timer and then timer can work in pulse mode, delay mode or cyclic mode to trigger output. In this configuration, you can program to process the I/O as you need and the standalone mode will work as a small PLC.

Work as SCADA I/O module

There are many SCADA software on the market, they provide the MODBUS protocol to communicate with the devices. The Ethernet I/O modules also provide the standard MODBUS protocol that enables the modules to talk with the SCADA software. The detail of SCADA configuration, please refer the SCADA software manual provided from the software company. (refer 15 modbus protocol)

6. Function format and language difference

6.1 Function format

Every EMD8216 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.

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 EMD8216 API. Read the following sections that apply to your programming language.

Note: Be sure to include the declaration functions of EMD8216 prototypes by including the appropriate EMD8216 header file in your source code. Refer to Chapter 4. EMD8216 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 = EMD8216_port_read (u32 CardID, u8 port, u8 *data);
```

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

```
u32 CardID=0
u8 port=0 ;    //assume CardID is 0 and port also 0
u8 data,
u32 Status;
Status = EMD8216_port_read ( CardID, port, &data);
```

6.3.2 Visual basic

The file EMD8216.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 EMD8216.bas, do not use the numerical values.
In Visual Basic, you can add the entire EMD8216.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 EMD8216.bas file for your project in Visual Basic 4.0, go to the **File** menu and select the **Add File...** **option**. Select EMD8216.bas, which is browsed in the EMD8216 \ api directory. Then, select **Open** to add the file to the project.

To add the EMD8216.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** EMD8216.bas, which is in the EMD8216 \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 EMD8216bc.lib EMD8216.dll

Then add the **EMD8216bc.lib** to your project and add

#include “EMD8216.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 = EMD8216_port_read ( CardID, port, &data);
```

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

```
u32 CardID=0;
u8 port=0 ;    //assume CardID is 0 and port also 0
u8 data,
u32 Status;
Status = EMD8216_port_read ( CardID, port, &data);
```

* 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 EMD8216 module and the detail of the dll function.

7.1 Initialization and close

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

EMD8216_initial( ) will do.

Once you want to close your application, call

EMD8216_close( ) to release all the resource.

To check the firmware version,

EMD8216_firmware_version_read( ) will do.

- **EMD8216_initial**

Format : u32 status =EMD8216_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 EMD8216 to a specified CardID number**.**

Parameters:

Input:

Name	Type	Description
CardID	u32	0~1999 Assign CardID to the EMD8216 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 EMD8216 Default: 6936
TimeOut	u16	Assign the max delay time of EMD8216 response message,1000~10000 ms.

Output:

Name	Type	Description
CardType	u8	not defined

- **EMD8216_close**

Format : **u32 status =EMD8216_close (u32 CardID)**

Purpose: Release the EMD8216 resource when closing the Windows applications.

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMD8216_initial

- **EMD8216_firmware_version_read**

Format : **u32 status =EMD8216_firmware_version_read(u32 CardID, u8 Version[2])**

Purpose: Read the firmware version.

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMD8216_initial

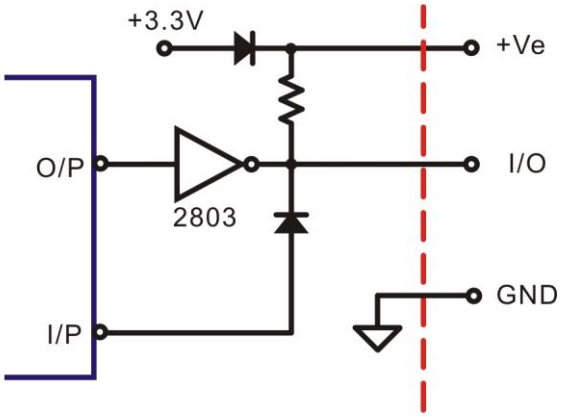
Output:

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

7.2 Digital I/O

Each digital I/O point can be configured as input or output. If it is configured as input, the input level can work up to the external terminal voltage Ve (not exceed the output buffer rating 45Vdc).

If it is configured as output, the output buffer can drive up to 450ma peak current and steady state up to 45ma, the working voltage up to 45Vdc. You can see the circuit diagram as follows.



First of all, each port must configure as input or output

EMD8216_port_config_set( ) and read back to verify by

EMD8216_port_config_read( )

Input and output polarity setting can give you the logic polarity as you need. Say, you use the positive logic in your application program and the input maybe short to ground as active, change the polarity to take the short to ground (active) input to be read as logic '1'.

To set the input and output polarity

EMD8216_port_polarity_set( ) and to read back the setting by

EMD8216_port_polarity_read( )

To control the output, use

EMD8216_port_set( )

To read input or output register status use

EMD8216_port_read( )

To configure a point as input or output, use

EMD8216_point_config_set( ) and read back to verify by

EMD8216_point_config_read( )

To set the input and output polarity by point

EMD8216_point_polarity_set( ) and to read back the setting by

EMD8216_point_polarity_read( )

To control a point of output, use

EMD8216_point_set( )

To read a point data of input or output register, use

EMD8216_point_read( )

- **EMD8216_port_config_set**

Format : **u32 status = EMD8216_port_config_set(u32 CardID ,u8 port, u8 config)**

Purpose: Set the Configure of the I/O port.

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMD8216_initial
Port	u8	port number 0: IO_00~IO_07 1: IO_10~IO_17
config	u8	Configure the IO as input or output bit0: 1: IO_n0 as input 0: IO_n0 as output … bit7: 1: IO_n7 as input 0:IO_n7 as output

- **EMD8216_port_config_read**

Format : **u32 status = EMD8216_port_config_read(u32 CardID ,u8 port, u8 *config)**

Purpose: Read back the Configure of the I/O port.

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMD8216_initial
port	u8	port number 0: IO_00~IO_07 1: IO_10~IO_17

Output:

Name	Type	Description
config	u8	Configure the IO as input or output bit0: 1:IO_n0 as input 0:IO_n0 as output … bit7: 1:IO_n7 as input 0:IO_n7 as output

- **EMD8216_port_polarity_set**

Format : u32 status = EMD8216_port_polarity_set (u32 CardID,u8 port,u8 data)

Purpose: Set the I/O port polarity.

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMD8216_initial
port	u8	port number 0: IO_00~IO_07 1: IO_10~IO_17
data	u8	b7: 0: normal polarity of IO_n7 1: invert polarity of IO_n7 … b0: 0: normal polarity of IO_n0 1: invert polarity of IO_n0

- **EMD8216_port_polarity_read**

Format : u32 status = EMD8216_port_polarity_read(u32 CardID ,u8 port, u8 *data)

Purpose: Read back the polarity setting of the I/O port.

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMD8216_initial
port	u8	port number 0: IO_00~IO_07 1: IO_10~IO_17

Output:

Name	Type	Description
data	u8	b7: 0: normal polarity of IO_n7 1: invert polarity of IO_n7 … b0: 0: normal polarity of IO_n0 1: invert polarity of IO_n0

- **EMD8216_port_set**

Format : u32 status = EMD8216_port_set (u32 CardID, u8 port,u8 data)

Purpose: Set the output port data.

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMD8216_initial
port	u8	port number 0: IO_00~IO_07 1: IO_10~IO_17
data	u8	b7: 0: IO_n7 in inactive state 1: IO_n7 in active state … b0: 0: IO_n0 in inactive state 1: IO_n0 in active state

Note: An output channel is active may be output high or low depends on the polarity it is configured. Say polarity is normal, active means output high and polarity is invert will have active output to be low state.

- **EMD8216_port_read**

Format : u32 status = EMD8216_port_read(u32 CardID ,u8 port, u8 *data)

Purpose: Read back the data of the I/O port.

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMD8216_initial
port	u8	port number 0: IO_00~IO_07 1: IO_10~IO_17

Output:

Name	Type	Description
data	u8	b7: 0: IO_n7 in inactive state 1: IO_n7 in active state … b0: 0: IO_n0 in inactive state 1: IO_n0 in active state

- **EMD8216_point_config_set**

Format : **u32 status =EMD8216_point_config_set(u32 CardID, u8 port, u8 point, u8 state)**

Purpose: Set bit Configure of I/O point

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMD8216_initial
port	u8	port number 0: IO_00~IO_07 1: IO_10~IO_17
point	u8	point number 0: IO_n0 .. 7:IO_n7
state	u8	1: input 0: output

Note: An output channel is active may be output high or low depends on the polarity it is configured. Say polarity is normal, active means output high and polarity is invert will have active output to be low state.

- **EMD8216_point_config_read**

Format : **u32 status =EMD8216_point_config_read(u32 CardID,u8 port,u8 point, u8 *state)**

Purpose: Read bit Configure of I/O point.

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMD8216_initial
port	u8	port number 0: IO_00~IO_07 1: IO_10~IO_17
point	u8	point number 0: IO_n0 .. 7:IO_n7

Output:

Name	Type	Description
state	u8	1: input 0: output

- **EMD8216_point_polarity_set**

Format : **u32 status = EMD8216_point_polarity_set (u32 CardID, u8 port, u8 point, u8 state)**

Purpose: Set the I/O point polarity.

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMD8216_initial
port	u8	port number 0: IO_00~IO_07 1: IO_10~IO_17
point	u8	point number 0: IO_n0 .. 7:IO_n7
state	u8	1 : invert 0 : normal.

Note: An output channel is active may be output high or low depends on the polarity it is configured. Say polarity is normal, active means output high and polarity is invert will have active output to be low state.

- **EMD8216_point_polarity_read**

Format : **u32 status = EMD8216_point_polarity_read(u32 CardID ,u8 port, u8 point, u8 *state)**

Purpose: Read back the polarity setting of the I/O point.

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMD8216_initial
port	u8	port number 0: IO_00~IO_07 1: IO_10~IO_17
point	u8	point number 0: IO_n0 .. 7:IO_n7

Output:

Name	Type	Description
state	u8	1 : invert 0 : normal.

- **EMD8216_point_set**

Format : **u32 status =EMD8216_point_set(u32 CardID, u8 port, u8 point, u8 state)**

Purpose: Set bit status of output point

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMD8216_initial
port	u8	port number 0: IO_00~IO_07 1: IO_10~IO_17
point	u8	point number 0: IO_n0 .. 7:IO_n7
state	u8	1 : active 0 : inactive

Note: An output channel is active may be output high or low depends on the polarity it is configured. Say polarity is normal, active means output high and polarity is invert will have active output to be low state.

- **EMD8216_point_read**

Format : **u32 status =EMD8216_point_read(u32 CardID,u8 port, u8 point, u8 *state)**

Purpose: Read bit state of I/O point.

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMD8216_initial
port	u8	port number 0: IO_00~IO_07 1: IO_10~IO_17
point	u8	point number 0: IO_n0 .. 7:IO_n7

Output:

Name	Type	Description
state	u8	1 : active 0 : inactive

7.3 Counter function

You can use the digital input as a low speed counter (no more than 100pps). First you can set which input channel you will want to work as counter by:

EMD8216_counter_mask_set( ) then enable the function by

EMD8216_counter_enable( ) and any time to stop by

EMD8216_counter_disable( ).

To read the counter value by

EMD8216_counter_read( ) and use

EMD8216_counter_clear( ) to clear counter.

Each point configured as input can work as low frequency counter (max 100Hz). The remote I/O module will count the input signal for you without any attention to the signal transition.

- **EMD8216_counter_mask_set**

Format : u32 status = EMD8216_counter_mask_set(u32 CardID,u8 port,u8 channel);

Purpose: To set the counter channel mask.

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMD8216_initial
port	u8	port number 0: IO_00~IO_07 1: IO_10~IO_17
Channel	u8	b7 ~ b0, b7: 0: IO_n7 counter disable 1: IO_n7 counter enable … b0: 0: IO_n0 counter disable 1: IO_n0 counter enable

- **EMD8216_counter_enable**

Format : **u32 status = EMD8216_counter_enable(u32 CardID);**

Purpose: To enable the counter.

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMD8216_initial

- **EMD8216_counter_disable**

Format : **u32 status = EMD8216_counter_disable(u32 CardID);**

Purpose: To disable the counter.

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMD8216_initial

- **EMD8216_counter_read**

Format : **u32 status = EMD8216_counter_read(u32 CardID, u8 port, u32 counter[8]);**

Purpose: To read all the counter value.

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMD8216_initial
port	u8	port number 0: IO_00~IO_07 1: IO_10~IO_17

Output:

Name	Type	Description
counter	u32	counter value counter[0] for IO_n0 … counter[7] for IO_n7

- **EMD8216_counter_clear**

Format : **u32 status = EMD8216_counter_clear (u32 CardID,u8 port,u8 channel);**

Purpose: To reset the counter value.

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMD8216_initial
port	u8	port number 0: IO_00~IO_07 1: IO_10~IO_17
Channel	u8	0:channel 0 1:channel 1 2:channel 2 3:channel 3 4:channel 4 5:channel 5 6:channel 6 7:channel 7

7.4 Miscellaneous function

The module IP and communication port must be confirmed with the gateway and software to ensure the correct Ethernet communication.

To change the communication port as you need by:

EMD8216_change_socket_port( ) [*1]

To change IP,

EMD8216_change_IP( ) [*1]

To reboot EMD8216 module for module alarm or to validate the system configuration change by:

EMD8216_reboot( ) [*1]

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

- **EMD8216_change_socket_port**

Format : u32 status = EMD8216_change_socket_port(u32 CardID,u16 Remote_port);

Purpose: To change the communicate port number of EMD8216.

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

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMD8216_initial
Remote_port	u16	The new port number to be set

- **EMD8216_change_IP**

Format : u32 status = EMD8216_change_IP(u32 CardID,u8 IP[4]);

Purpose: To change the communicate IP of EMD8216.

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

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMD8216_initial
IP[4]	U8	The new IP to be set

- **EMD8216_reboot**

Format : **u32 status = EMD8216_reboot(u32 CardID);**

Purpose: To reboot EMD8216(about 10s).

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMD8216_initial

7.5 Software key function

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

To operate the EMD8216, you must unlock the module first by

EMD8216_security_unlock( )

To verify the lock status by

EMD8216_security_status_read( )

You can change password for your convenience by

EMD8216_password_change( )

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

EMD8216_password_set_default( ) [*2]

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

- **EMD8216_security_unlock**

Format : u32 status = EMD8216_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 EMD8216_initial
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'};

- **EMD8216_security_status_read**

Format : **u32 status = EMD8216_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 EMD8216_initial

Output:

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

- **EMD8216_password_change**

Format : **u32 status = EMD8216_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 EMD8216_initial
Oldpassword [8]	u8	The previous password (ASCII)
password[8]	u8	The new password to be set (ASCII)

- **EMD8216_password_set_default**

Format : **u32 status = EMD8216_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 EMD8216_initial default : password[8] = {'1','2','3','4','5','6','7','8'};

7.6 WDT (watch dog timer)

In the industrial environment, we want the controller work as stable as possible but we are not God; we cannot 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 runs in abnormal state the system will fail to reset WDT then WDT will latch the system to prevent further harm. EDM8216 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 ***EMD8216_WDT_set( )*** to set up the WDT timer and the output state if the Ethernet connection fail to communicate.

EMD8216_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:

EMD8216_WDT_enable( ) and disable by:

EMD8216_WDT_disable( ).

- **EMD8216_WDT_set**

Format : **u32 status = EMD8216_WDT_set(u32 CardID,u16 time,u8 state[2])**

Purpose: Set WDT (watch dog timer) configuration.

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMD8216_initial
time	u16	Set the WDT wait time.(10~10000) based on 0.1 sec time base. default: 10 (1s)
state[2]	u8	Set the output default state, the state will keep while the connection failure. state[0]: IO_07~IO_00 predefined state at connection fail. state[1]: IO_17~IO_10 predefined state at connection fail.

Note: The predefined outputs will be complied with the polarity it is configured.

- **EMD8216_WDT_read**

Format : u32 status = EMD8216_WDT_read (u32 CardID, u16 *time, u8 state[2], u8 *enable)

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

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMD8216_initial

Output:

Name	Type	Description
time	u16	read the WDT wait time.
state[2]	u8	state[0]: IO_07~IO_00 predefined state at connection fail. state[1]: IO_17~IO_10 predefined state at connection fail.
enable	u8	0: disable 1: enable

- **EMD8216_WDT_enable**

Format : u32 status = EMD8216_WDT_enable(u32 CardID)

Purpose: enableWDT(watch dog timer) .

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMD8216_initial

- **EMD8216_WDT_disable**

Format : u32 status = EMD8216_WDT_disable(u32 CardID)

Purpose: disable WDT (watch dog timer) .

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMD8216_initial

7.7 Standalone function

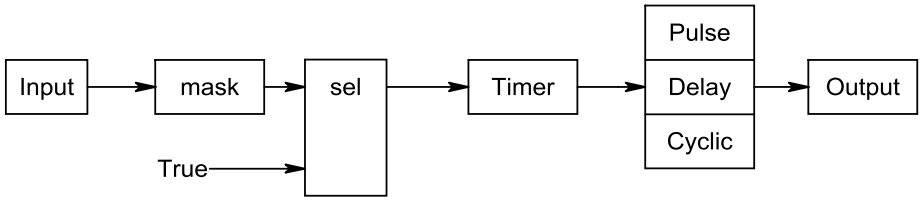
Standalone mode is the extension of EMD8216 module; it can work as I/O controller without the Ethernet existing.

The basic idea is the input, timer and output: 3 major elements. Input can be masked to select the desire state then timer accepts the trigger from input.

If timer works in delay mode, the output will not trigger until the time up.

If timer works in pulse mode, the output will trigger immediately on the input condition meets but inactive while time up.

If timer works in cyclic mode, the output will toggles immediately and stops until timer off.



The function blocks are as shown above.

- **EMD8216_standalone_enable**

Format : u32 status =EMD8216_standalone_enable(u32 CardID)

Purpose: Enable standalone mode**.**

Parameters:

Input:

Name	Type	Description
CardID	u32	0~255 CardID assigned by EMD8216_initial

- **EMD8216_standalone_disable**

Format : u32 status =EMD8216_standalone_disable(u32 CardID)

Purpose: Disable (stop) standalone mode**.**

Parameters:

Input:

Name	Type	Description
CardID	u32	0~255 CardID assigned by EMD8216_initial

- **EMD8216_standalone_config_set**

Format : **u32 status =EMD8216_standalone_config_set(u32 CardID, u8 number, StandaloneData data[32], u8 standalone_state)**

Purpose: To configure the process command.

Parameters:

Input:

Name	Type	Description
CardID	u32	0~255 CardID assigned by EMD8216_initial
number	u8	Number of data[ ]

data[32]	standalone_data	typedef struct _StandaloneData{
		u16 in_point_bit;
		u16 in_state_bit;
		u16 timer_value;
		u16 out_point_bit;
		u8 timer_mode;
		u8 out_mode;
		}
		in_point_bit
		//b7 ~ b0 = in07 ~ in00
		//b15 ~ b8 = in17 ~ in10
		in_state_bit
		//set input state
		timer_mode
		// 0x0 = Unused,
		// 0x1 = Input action and delay out,
		// 0x2 = Input action and pulse out
		// 0x3 = Input action and periodic out
		// 0x4 = Timer action and delay out
		// 0x5 = Timer action and periodic out
		// 0x6 = Timer off
		timer_value
		// timer tick is 5ms per tick
		// timer_mode = delay / periodic out
		// setting value is delay timer
		// timer_mode = pulse out
		// setting value is active time of pulse
		// if timer_value = 10 ,
		// the delay time is 10 * 5 ms = 50 ms

		out_point_bit
		//b7 ~ b0 = in07 ~ in00
		//b15 ~ b8 = in17 ~ in10
		out_mode
		// 0x0=inactive,
		// 0x1=active,
		// 0x2=Change

standalone_state	u8	0: power on do not run standalone mode (default) 1: power on run standalone mode

Note:

1. The Standalone Data is any array of 32 elements in which each element is a command of process. Each time you configure, you must prepare the 32 elements. If the command data is null (all elements are “0” in any of the 32 elements), the controller will take it as end of process.
2. Although the Standalone Data consist of 32 (max) elements, you also need to specify the number of the elements to accelerate the speed of instruction loading process (if less than 32, it will spend less time).
3. Standalone_state is used for configuration the function after the power-on. If standalone_state=1, after power on, the controller will run the pre-programmed command until it is commanded to stop from Ethernet interface or power off.

- **EMD8216_standalone_config_read**

Format : u32 status =EMD8216_standalone_config_read(u32 CardID,u8 *number, StandaloneData data[32], u8 *enable, u8 *power_on_enable)

Purpose: To read back the pre-programmed standalone process command**.**

Parameters:

Input:

Name	Type	Description
CardID	u32	0~255 CardID assigned by EMD8216_initial

Output:

Name	Type	Description
number	u8	Number of data[ ]

data	standalone_data	typedef struct _StandaloneData{
		u16 in_point_bit;
		u16 in_state_bit;
		u16 timer_value;
		u16 out_point_bit;
		u8 timer_value;
		u8 out_mode;
		}
		in_point_bit
		//b7 ~ b0 = in07 ~ in00
		//b15 ~ b8 = in17 ~ in10
		in_state_bit
		//set input state
		timer_mode
		// 0x0 = Unused,
		// 0x1 = Input action and delay out,
		// 0x2 = Input action and pulse out
		// 0x3 = Input action and periodic out
		// 0x4 = Timer action and delay out
		// 0x5 = Timer action and periodic out
		// 0x6 = Timer off
		timer_value
		// timer tick is 5ms per tick
		// timer_mode = delay / periodic out
		// setting value is delay timer
		// timer_mode = pulse out
		// setting value is active time of pulse
		// if timer_value = 10 ,
		// the delay time is 10 * 5 ms = 50 ms

		out_point_bit
		//b7 ~ b0 = in07 ~ in00
		//b15 ~ b8 = in17 ~ in10
		out_mode
		// 0x0=inactive,
		// 0x1=active,
		// 0x2=Change (toggle)

enable	u8	0: currently is standalone disabled 1: currently is standalone enabled
power_on_ enable	u8	0: power on do not run standalone mode (default) 1: power on run standalone mode

7.8 Error codes and address

Every EMD8216 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 EMD8216 function is the parameter **CardID** which is set by ***EMD8216_initial***. You can utilize multiple devices with different card ID within one application; to do so, simply pass the appropriate **CardID** to each function.

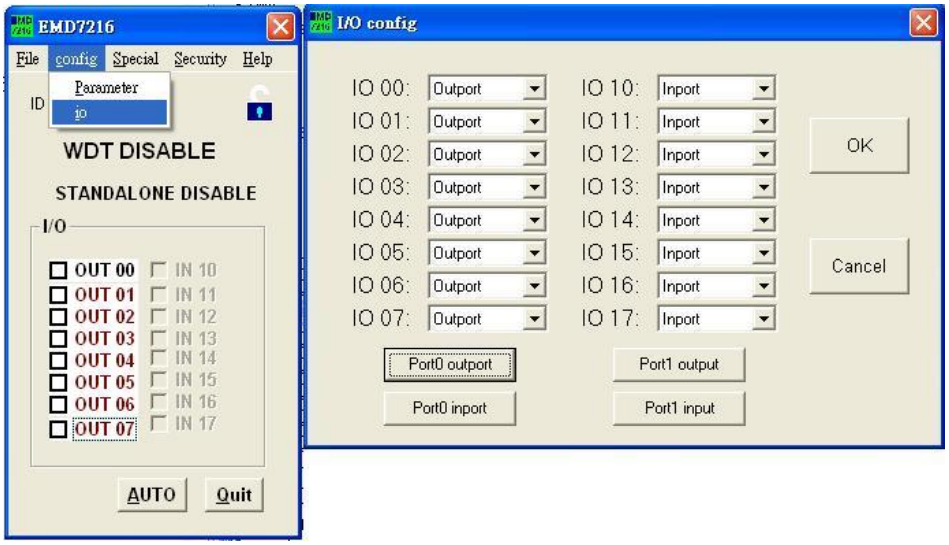
8. Standalone mode user configuration utility

Sometime you want to use the standalone mode without coding a program, it is easy to use the user configuration utility comes with the driver CD.

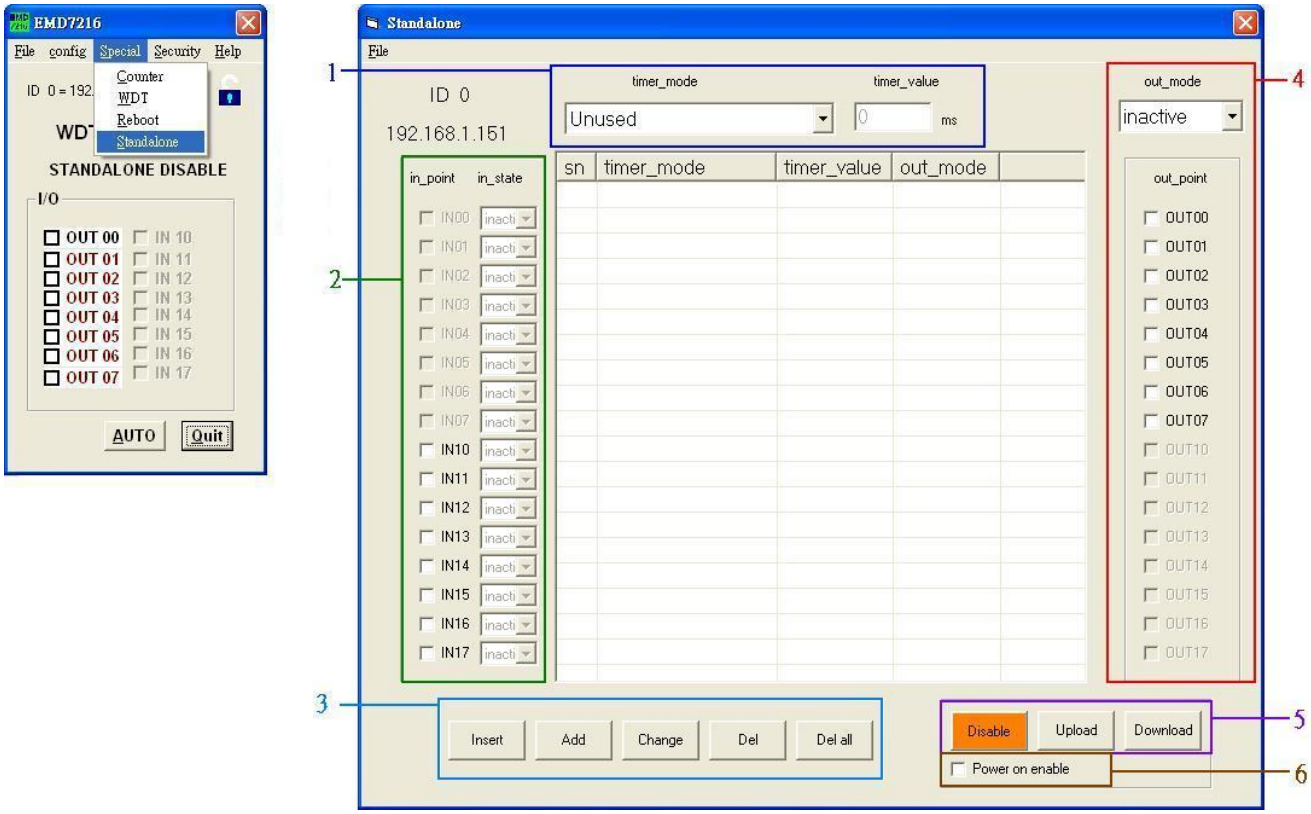
8.1 Overview of user configuration utility

-- After you have installed the driver and the demonstration program, run the EMD8216 demo program.

-- You must configure the I/O's as you need. Say which one is used as input and which one used as output.



-- Open the standalone mode configuration window. EMD8216 -> Special -> Standalone.



From the above diagram, you will see

Block1: Timer operation mode and time constant setting.

Block2: standalone mode command input configuration.

Block3: command edit function, add/ delete/insert.

Block4: standalone mode command output configuration.

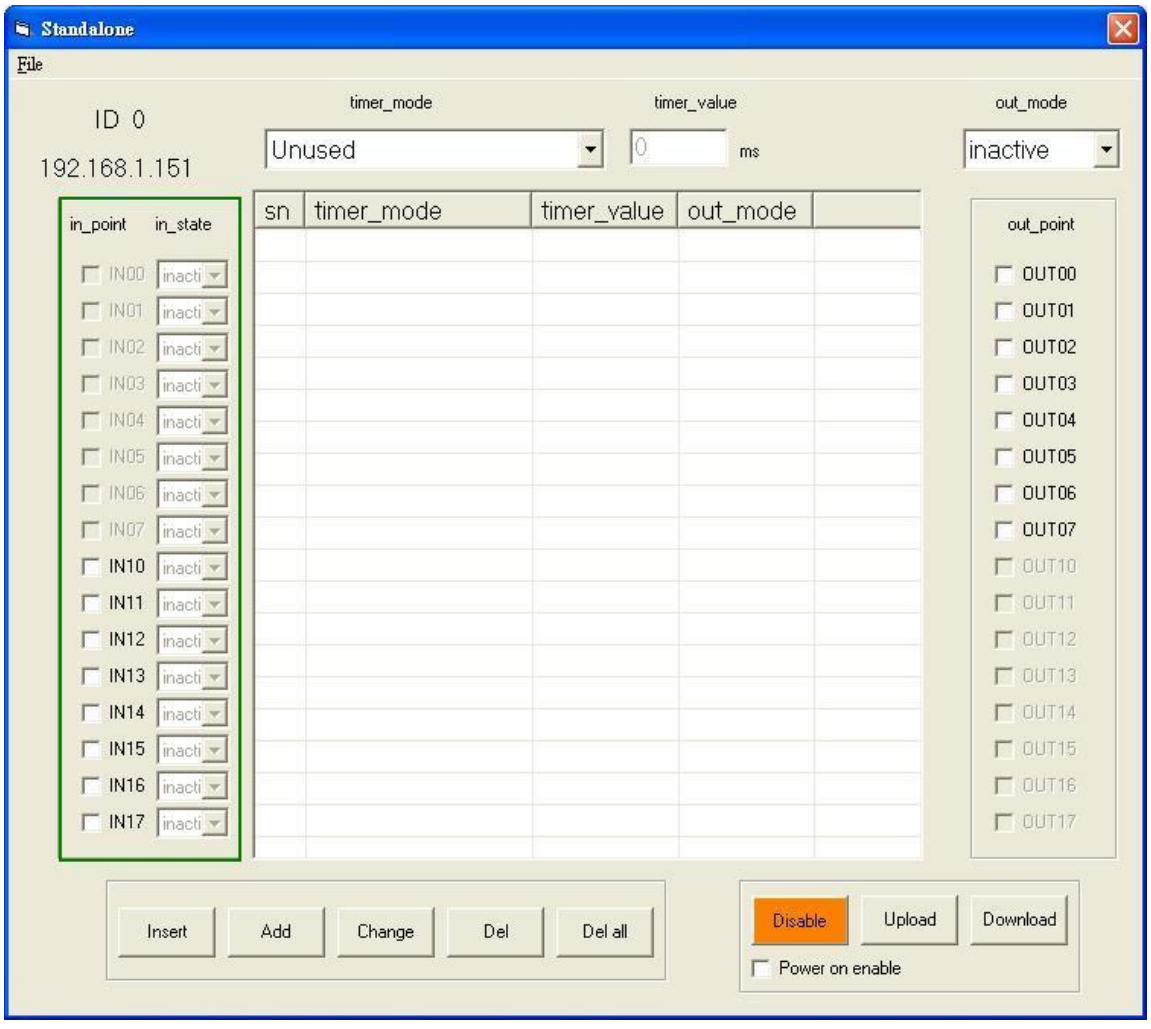
Block5: standalone mode command upload /download, start/stop.

Block6: power on standalone mode enable/ disable.

8.2 Configure a command

Each standalone command consists of input, timer and output. Generally we configure the input first.

-- input configuration

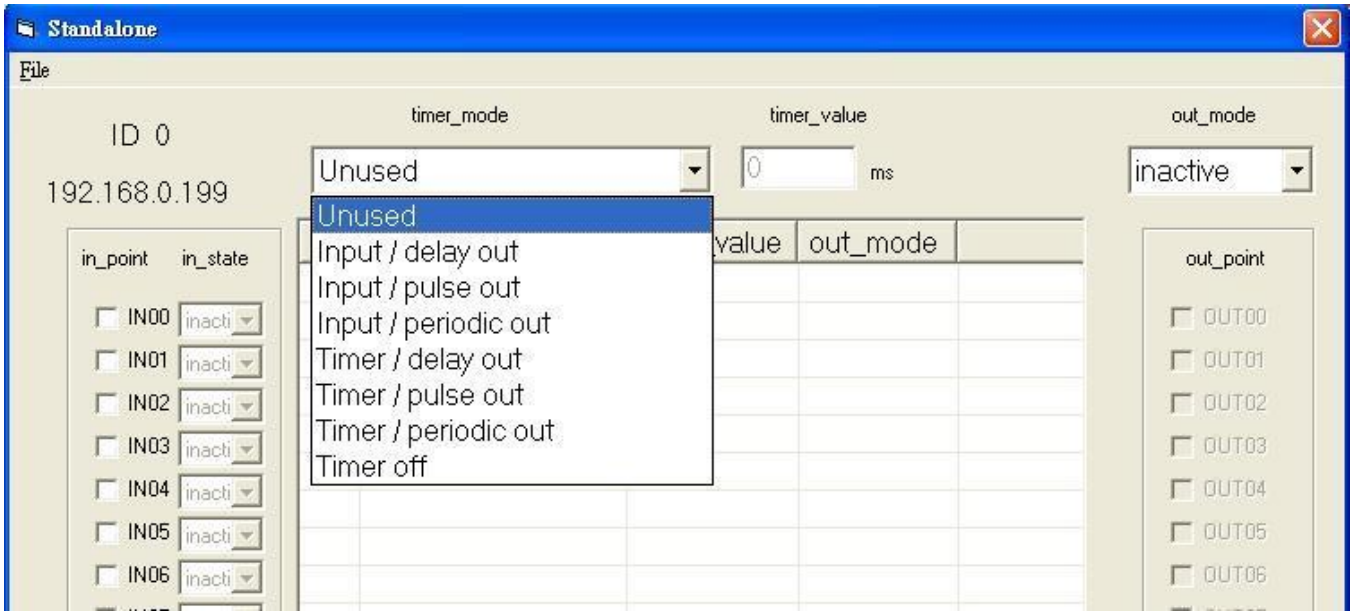


From the above diagram, check the input point and its state the current command will take care.

The above diagram shown that if you want the input monitor IN01 active and IN02 inactive as trigger source of the command. You can select and configure any of the inputs (the I/O have already configured as input) to monitor as trigger source.

Input debounce frequency is 100Hz, response faster than 100Hz maybe ignore as noise by the EMD8216 module.

-- timer configuration

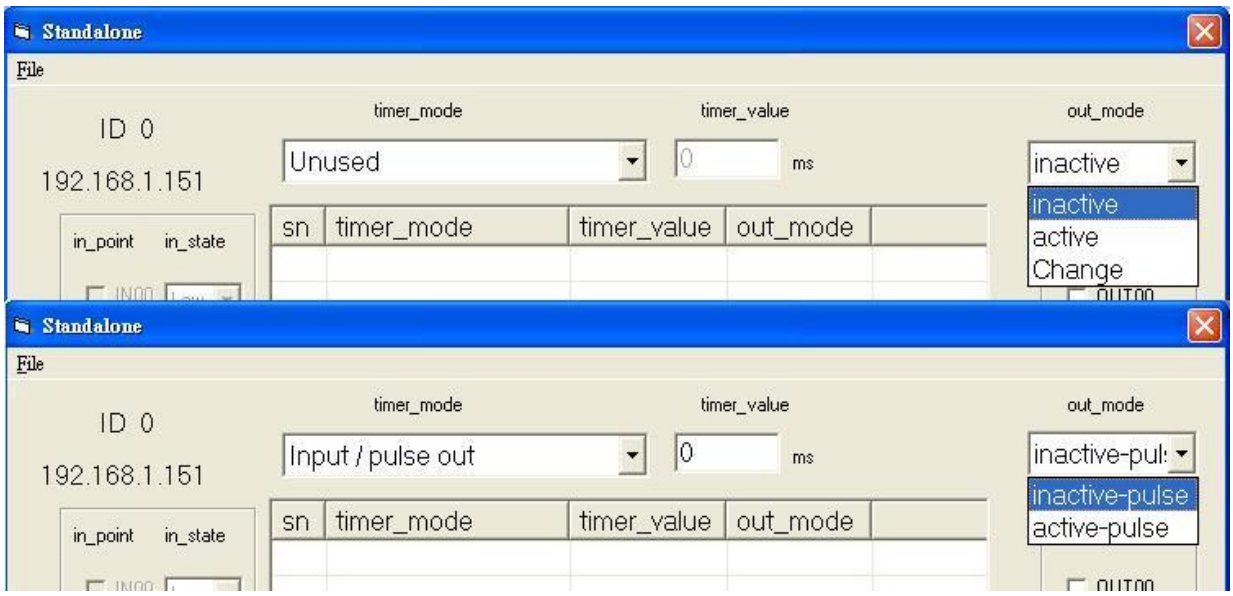


If the inputs meet the condition you configured, it will trigger the timer to operate. The time provides several kinds of working mode:

working mode	explanation
Unused	bypass the input trigger to output, timer do no work.
Input / delay out	input trigger the timer to work as delay timer (time up triggers output)
Input / pulse out	input trigger the timer to work as pulse timer (timing the output duty)
Input / periodic out	input trigger the timer to work as periodic timer (time up toggles output and reload to run)
Timer / delay out	power on or start standalone mode trigger the timer to work as delay timer (time up triggers output)
Timer / pulse out	power on or start standalone mode trigger the timer to work as pulse timer (timing the output duty)
Timer / periodic out	power on or start standalone mode trigger the timer to work as periodic timer (time up toggles output and reload to run)
Timer off	disable the timer function just followed by current command

The timer is based on 5ms time base, less than 5ms or not the multiples of 5ms is impossible to implement.

-- Output configuration

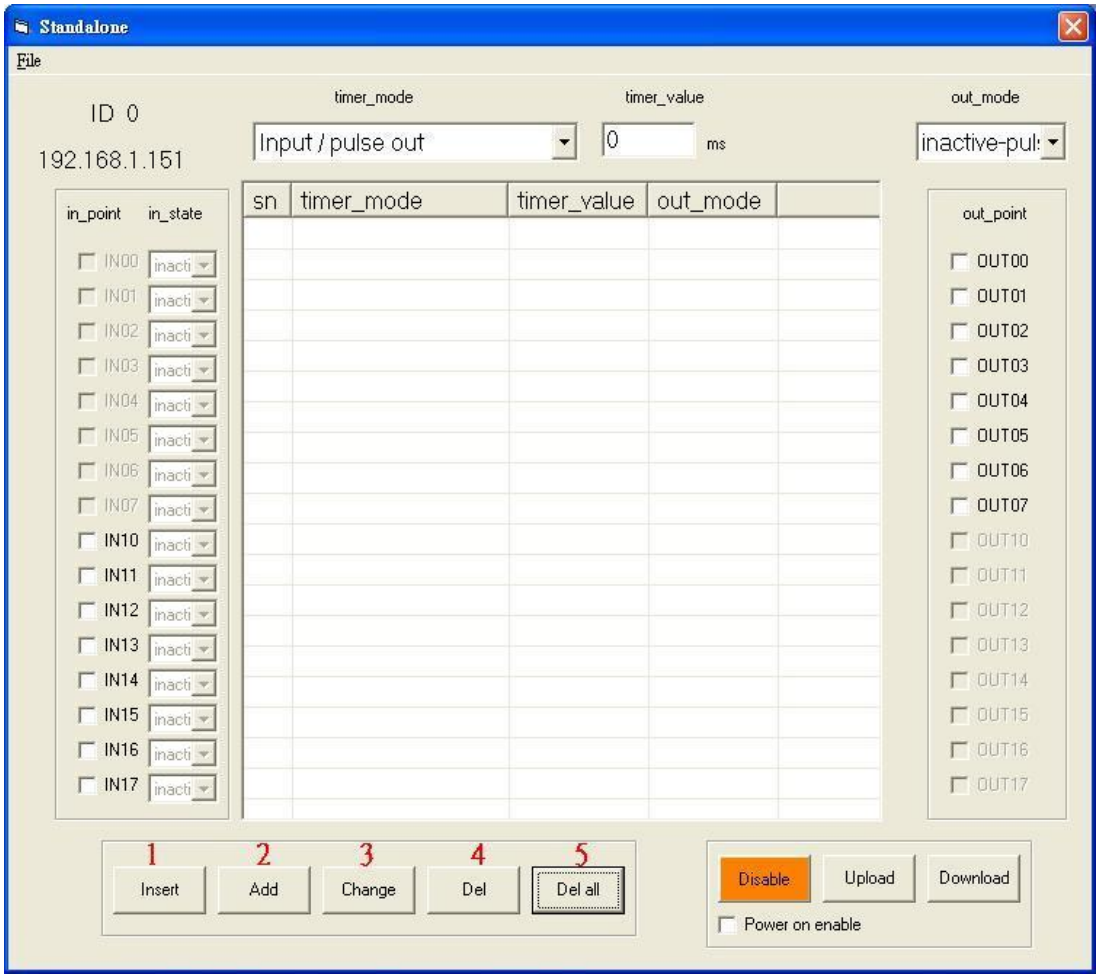


The timer depending on its working mode controls the output. The output can be configured as inactive , active or toggle.

Timer mode	output mode	explanation
Unused	inactive	output inactive
	active	output active
	Change	output toggles
Input action delay	inactive level	output inactive level
	active level	output active level
	Change	output toggles
Input action pulse	inactive-pulse	output inactive pulse (‾|__|‾)
	active-pulse	output active pulse (__|‾|__)
Input action periodic	Change	output toggles
Timer action delay	inactive level	output inactive level
	active level	output active level
	Change	output toggles
Timer action periodic	Change	output toggles
Timer off	none	output reset to its normal state

8.3 Edit function

To provide a good edit environment, some functions of editing are necessary: insert, add, change, delete and delete all are provided. Basically, a command is consist of input point and its state (on the left side of the following diagram); next, the timer operation mode, time constant (on the middle of the following diagram) and finally the output mode and output points (on the right side of the following diagram). The input and output block will update as the current command line highlighted.



1: Insert: insert a new command above the high lighted bar in the table.

2: Add: add a new command

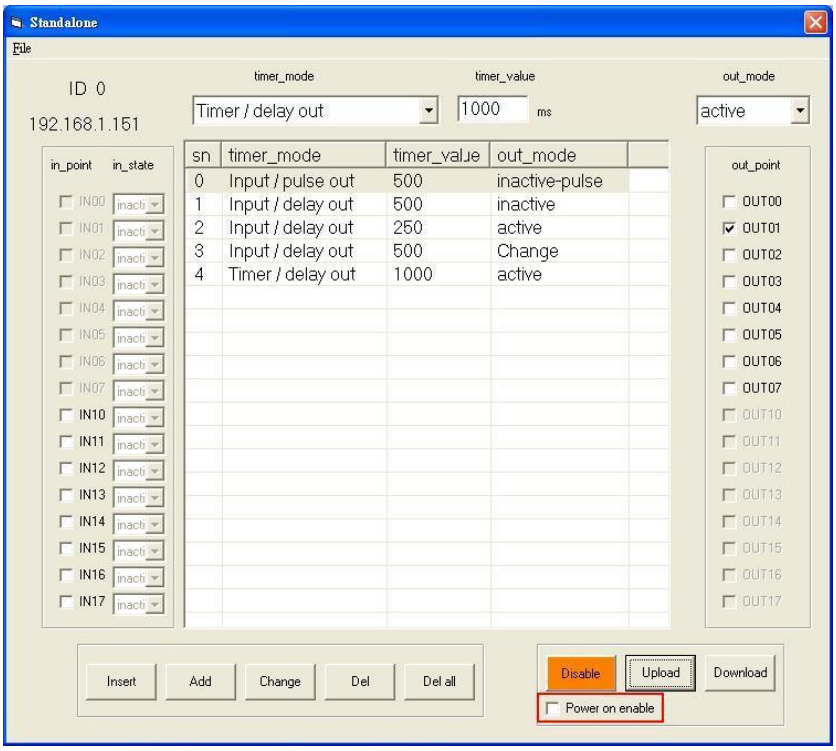
3: Change: modify the existing command line

4: Del: delete the highlighted command line

5: Del all: clear all the commands

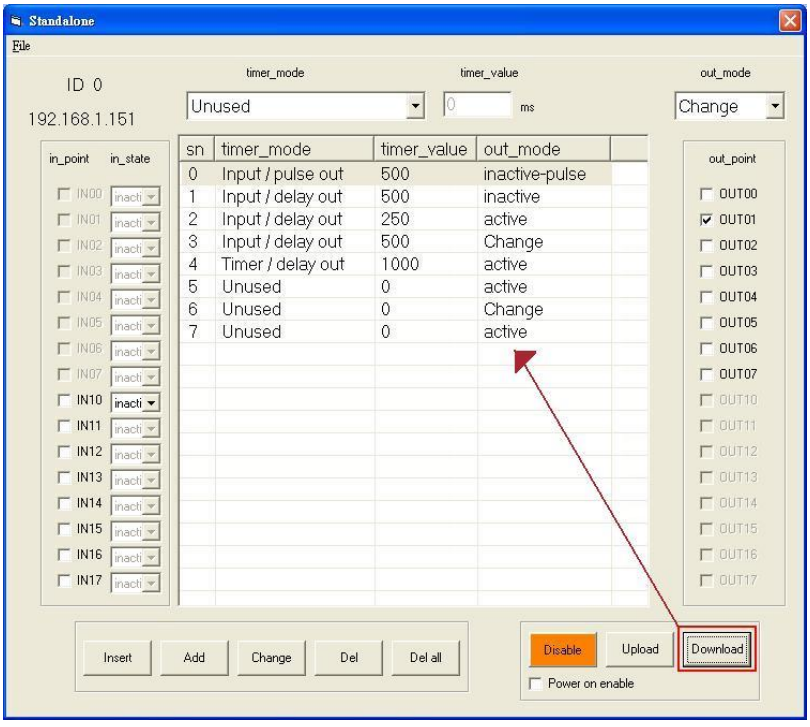
8.4 Upload program

There are totally 32 commands can be execute in EMD8216 module, after you edit the command sequence, you can upload to the module to store and execute immediately or store it and execute on next power on (select option: power on enable) or command to run via Ethernet.



8.5 Download program

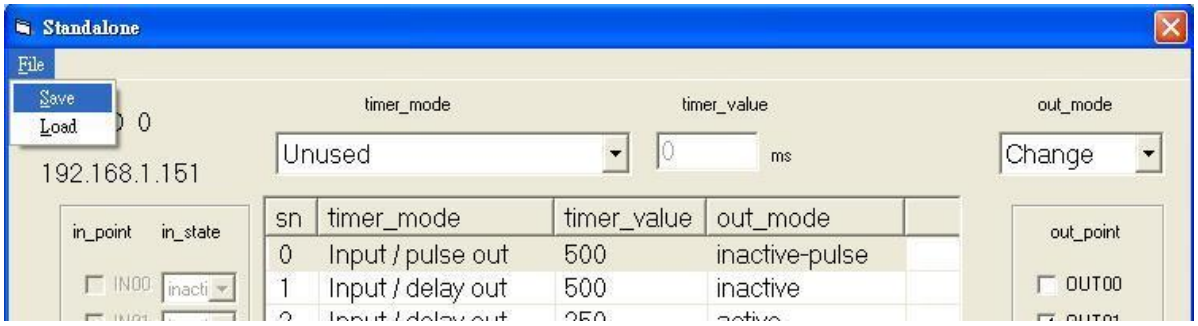
If you have connected with EMD8216 module via Ethernet, you can download the stored program from the module.



8.6 Save and load program with PC

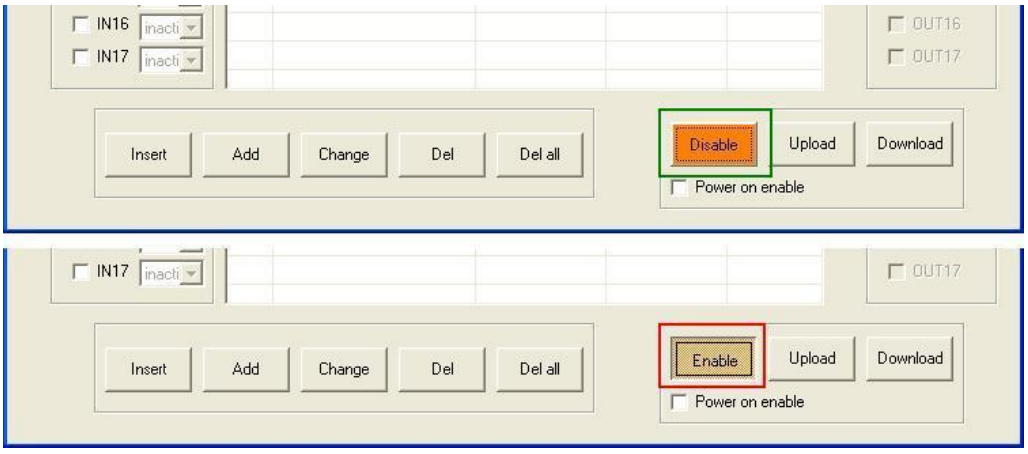
You can save the under edit or finished program to PC by click the File->Save to save the file as a specific file name and place.

To retrieve the stored program from PC by click File->Load and select the file you want to retrieve.



8.7 Enable/Disable standalone function

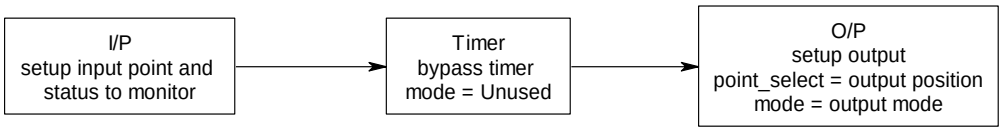
Standalone mode can enable or disable by the button as following shown.

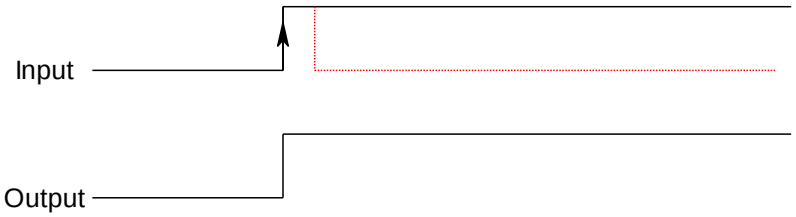


Whether the module standalone mode is enabled or disabled can be verified shown on the main form.

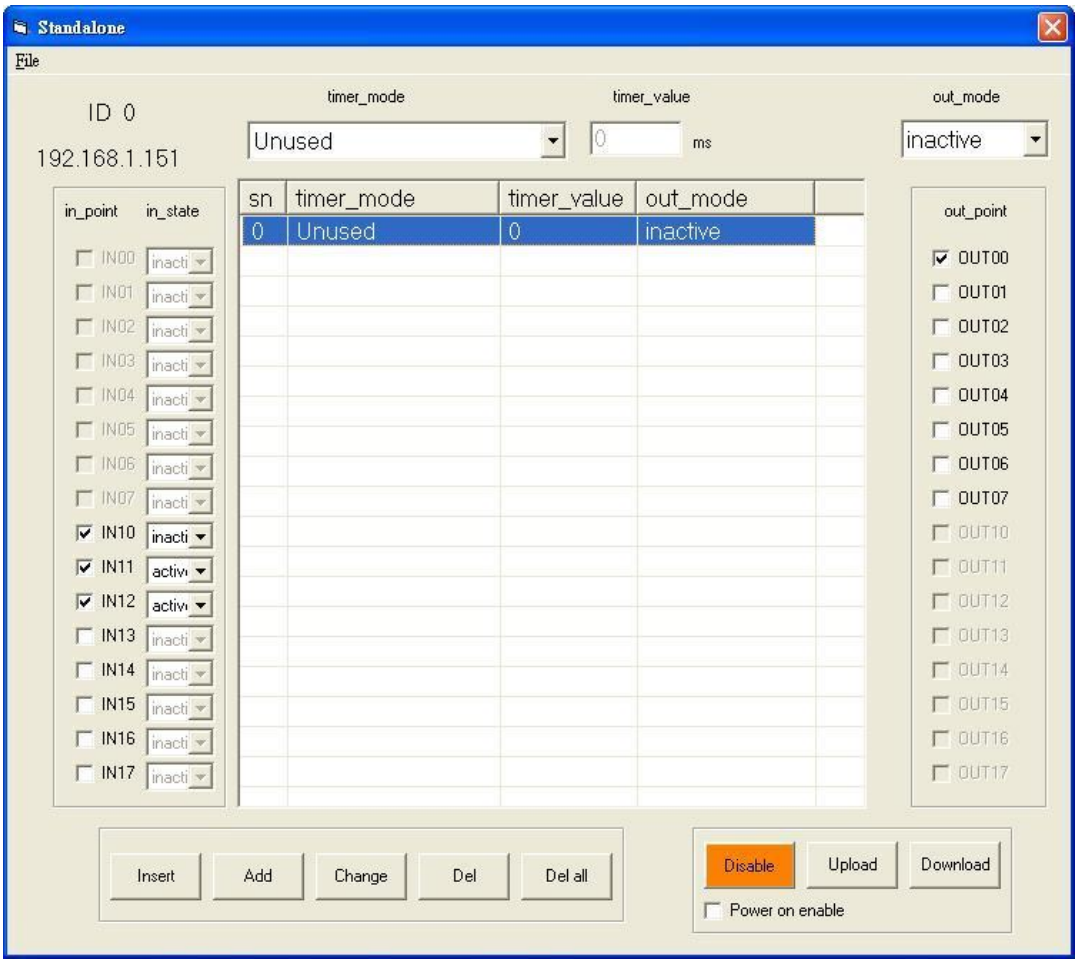
9. Standalone mode application examples

9.1 Monitoring input if condition meets, trigger output

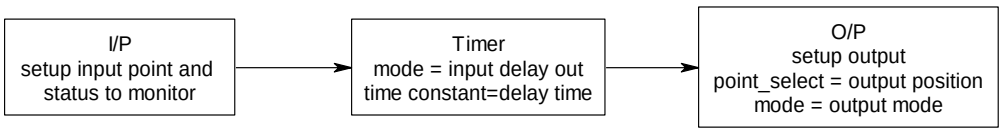


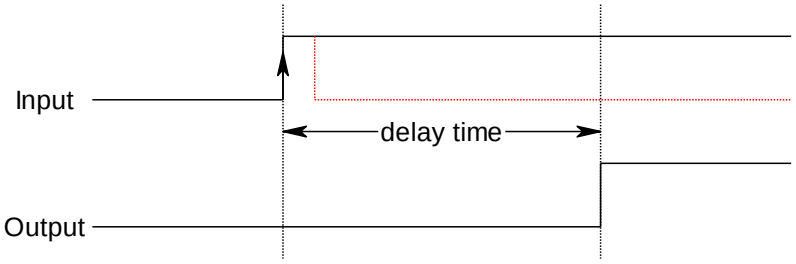


Say, you want to watch IN10 inactive, IN11 and IN12 active to trigger output OUT00 to inactive. Program as the following shown.

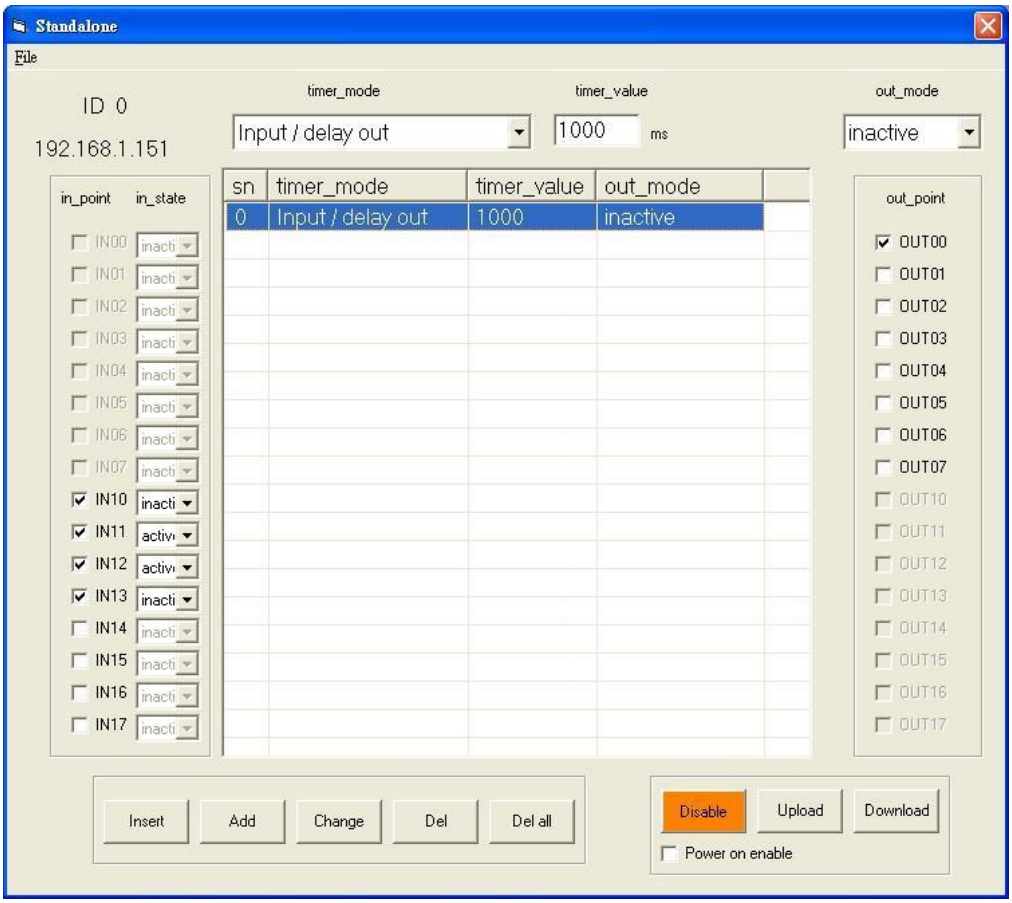


9.2 Monitoring the input if condition meets, delay to trigger output

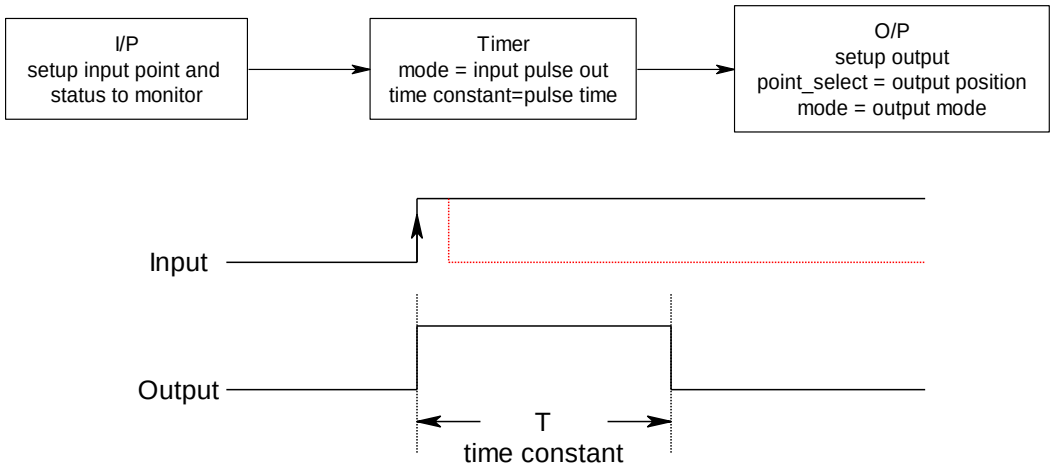




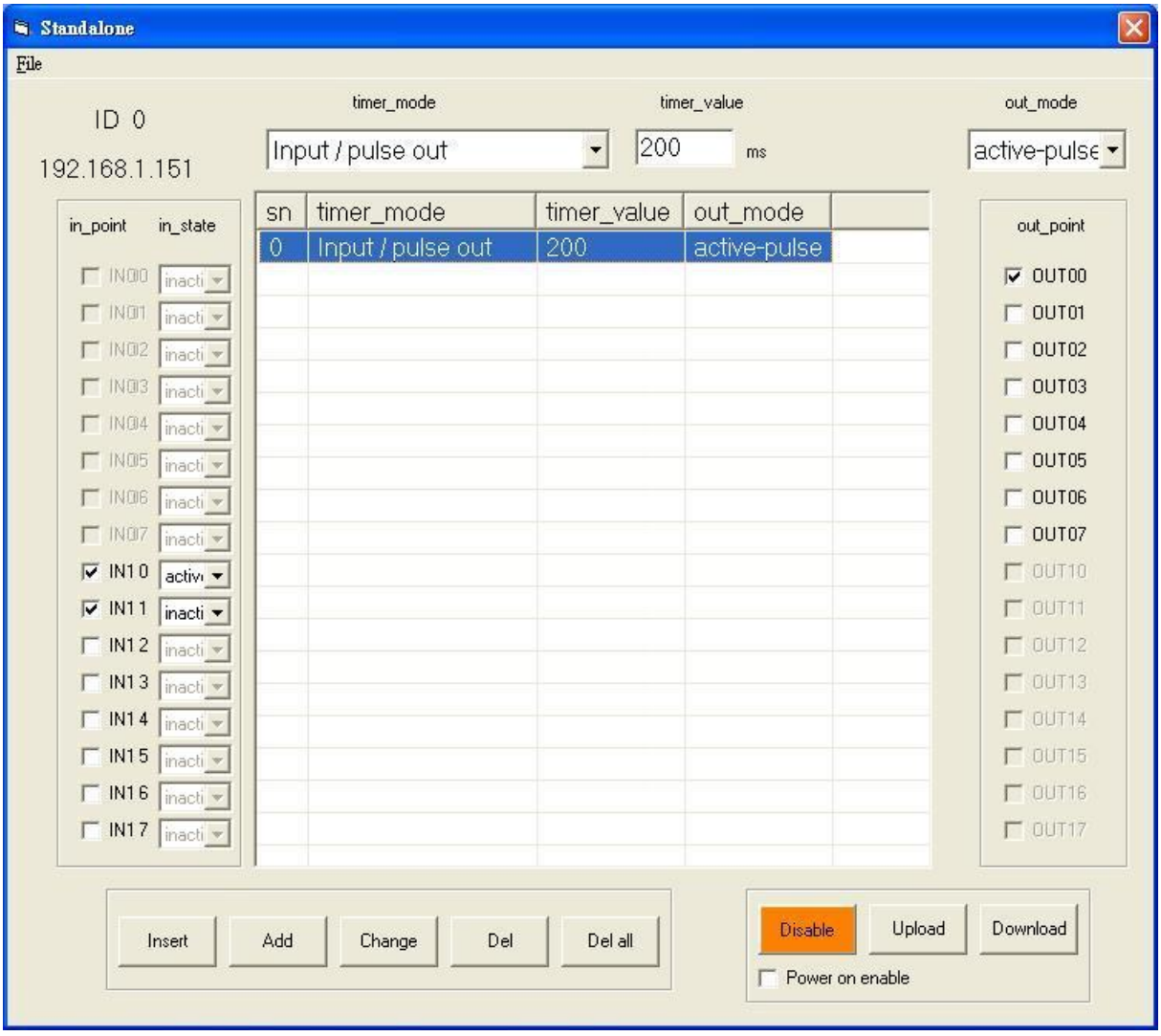
Say, you want to watch IN10 and IN13 are inactive and IN11 and IN12 are active to trigger output OUT00 to inactive. Program as the following shown.



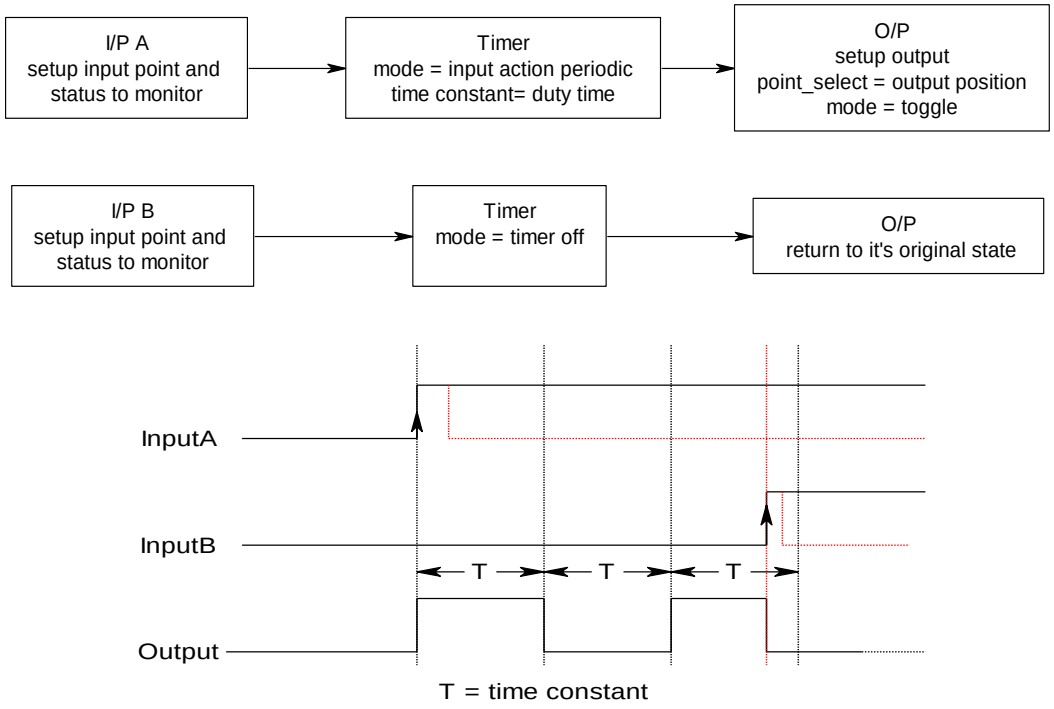
9.3 Monitoring the input if condition meets, output pulse



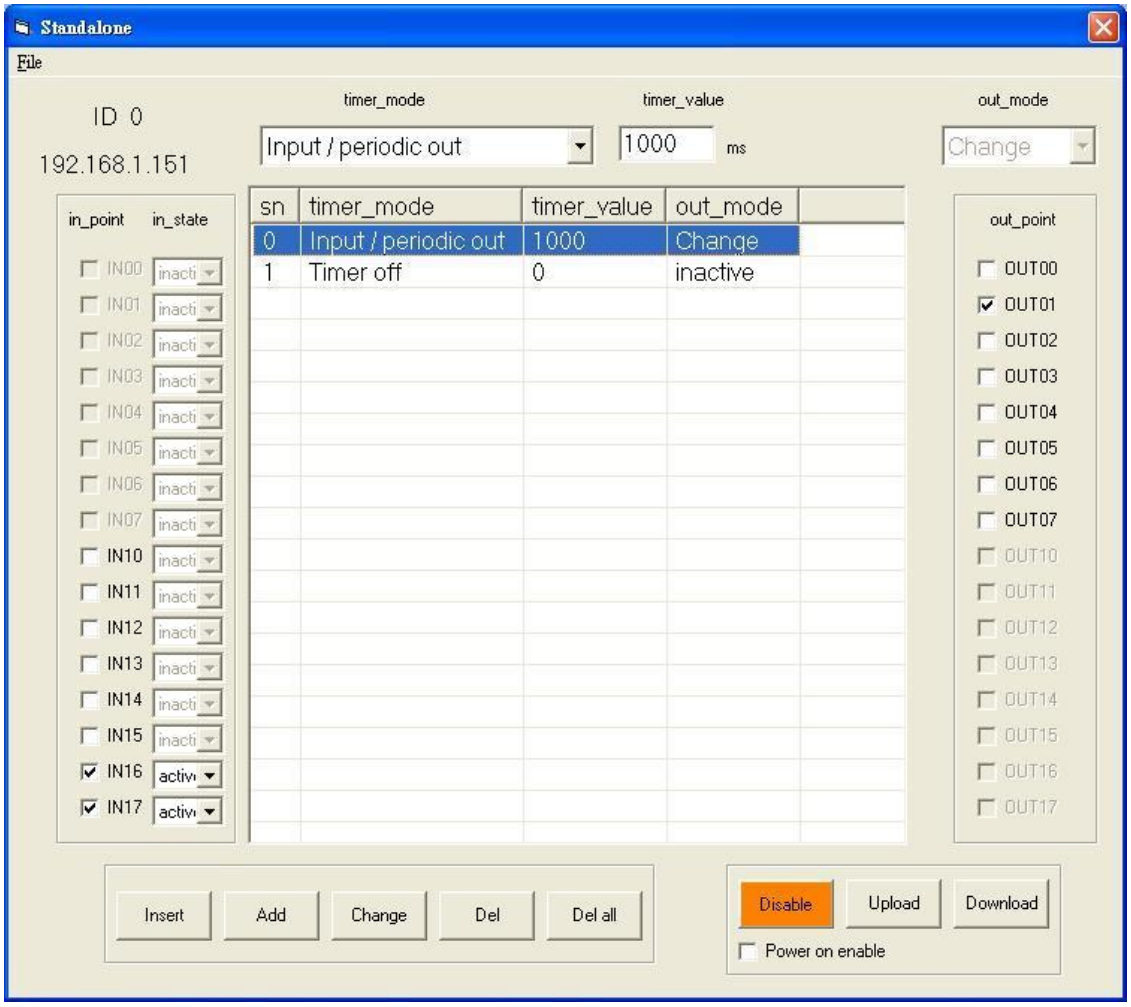
Say, you want to watch IN10 is active and IN11 is inactive to trigger output OUT00 to active-pulse. Program as the following shown.



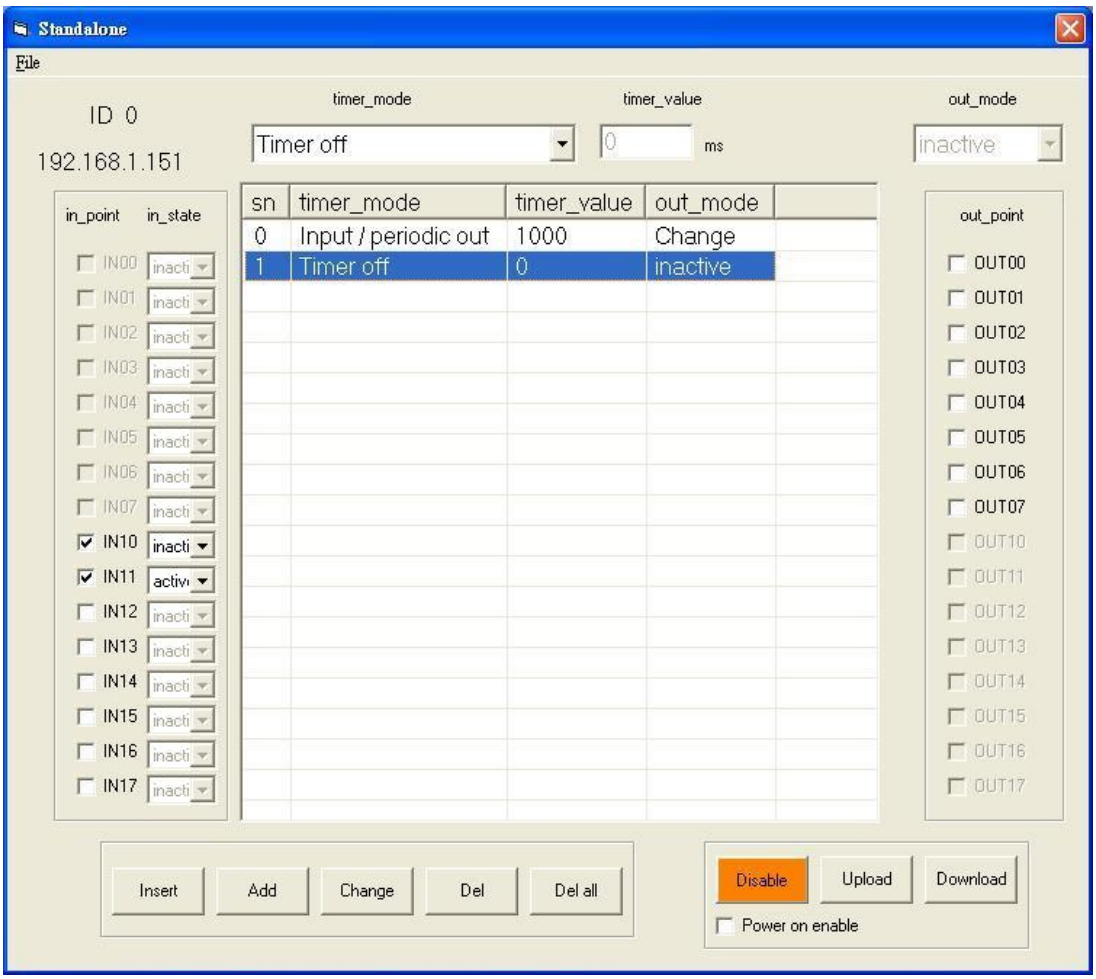
9.4 Monitoring the input if condition meets, output periodically and stop by some special input condition



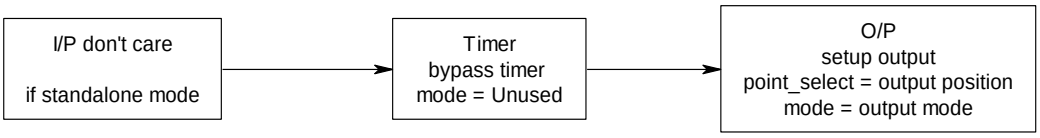
Say, you want to watch IN10 is inactive and IN11 is active to trigger output OUT00 to toggle. Program as the following shown.

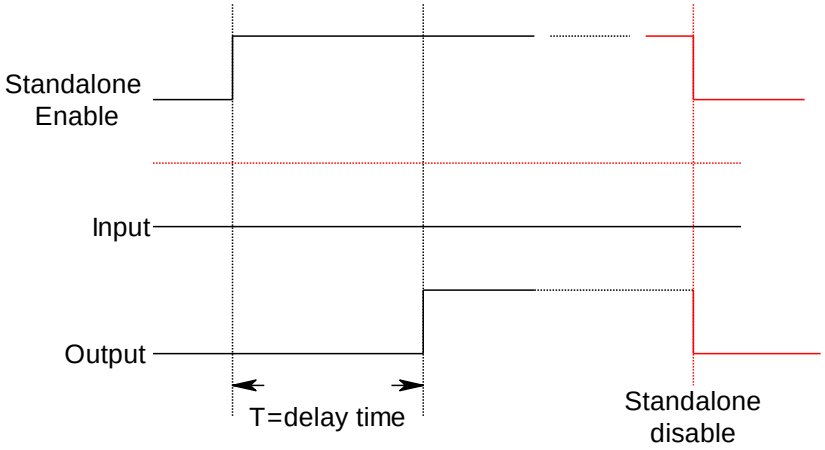


Then, if we want IN10 is inactive and IN11 is active to trigger to stop the timer. Program as the following shown.

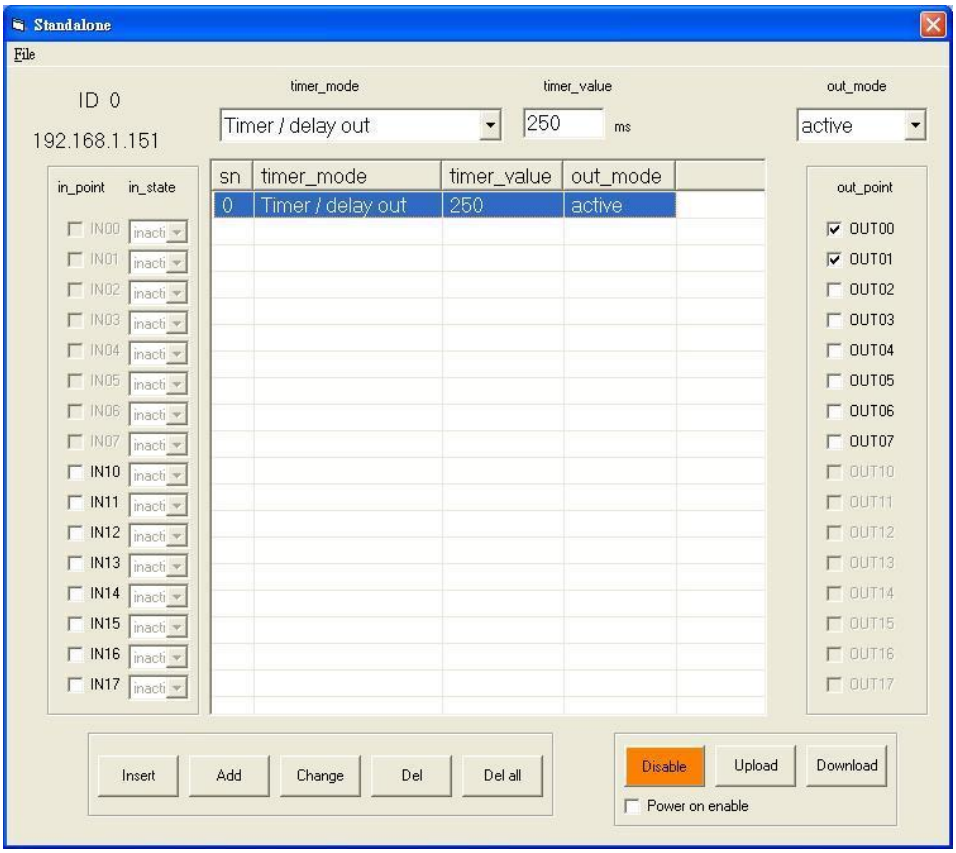


9.5 Don’t care the input if standalone enabled, trigger output

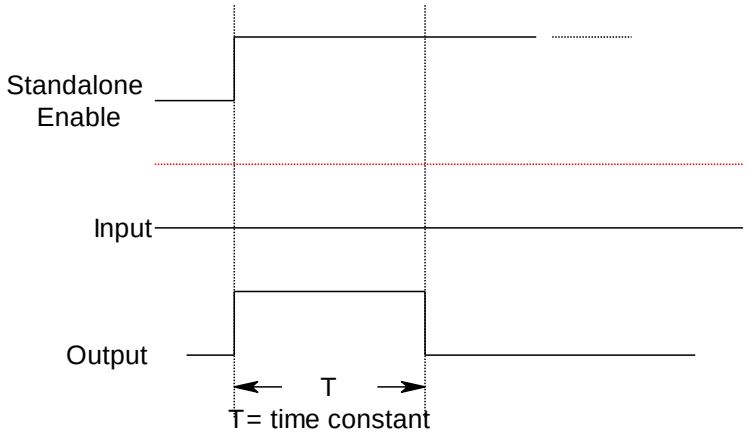




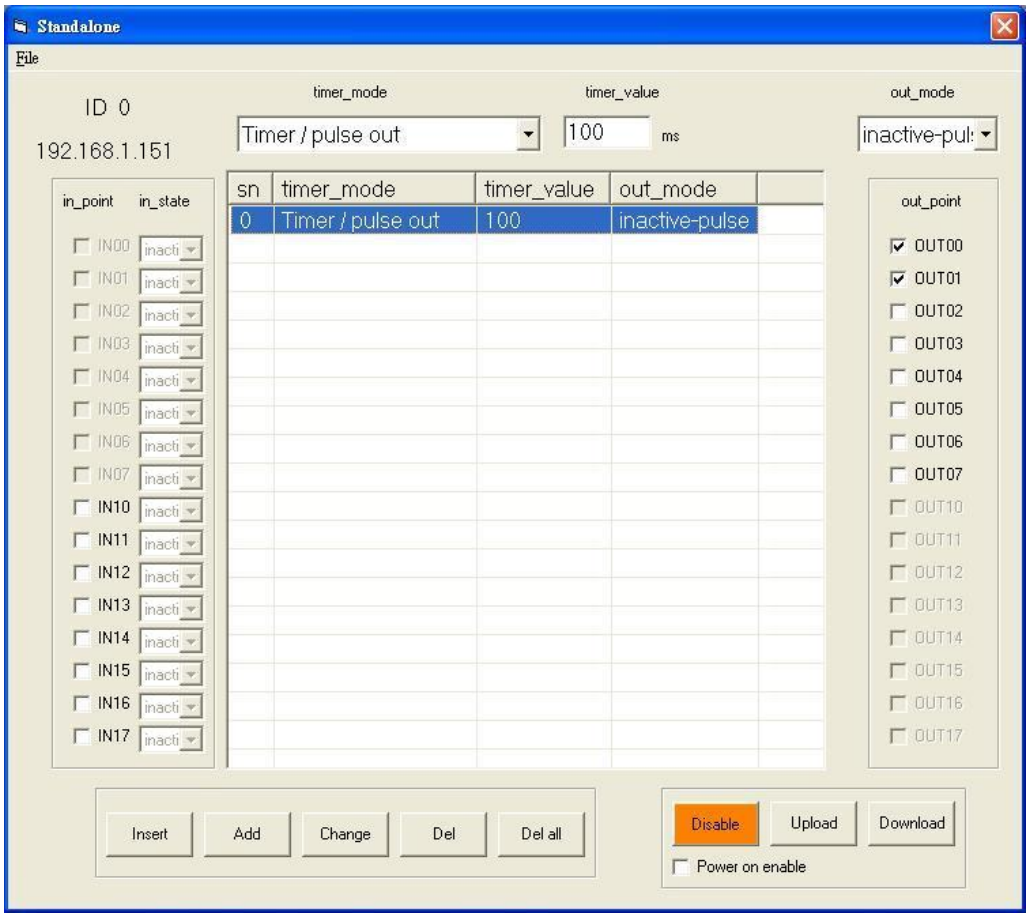
Say, don’t care any input just output OUT00 OUT01 active as the standalone mode enabled. Program as the following shown.



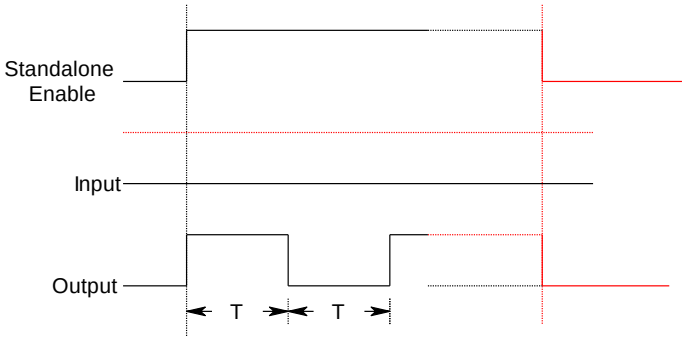
9.6 Don’t care the input if standalone enabled, trigger pulse



Say, don’t care any input just output OUT00 OUT01 inactive-pulse as the standalone mode enabled. Program as the following shown.

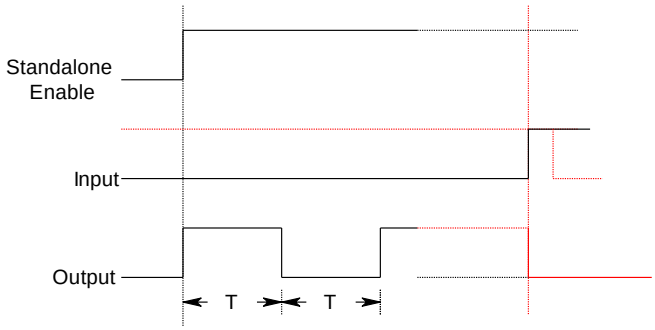


9.7 Don’t care the input if standalone enabled, output periodically



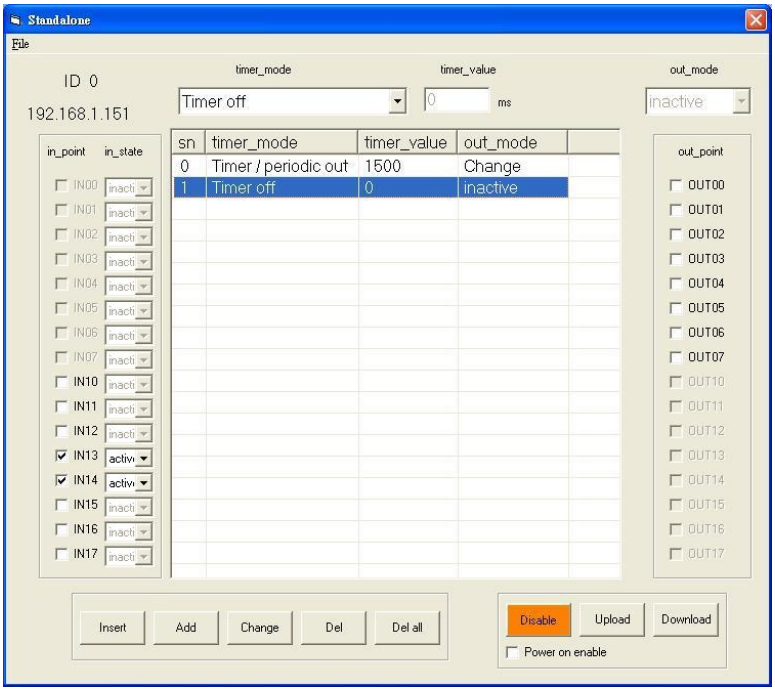
T= time constant

The above diagram shows the output will be active while standalone mode is enabled, if the standalone mode is disabled, the output will be reset. Another method to stop the periodic working output is to use some input to trigger to stop it. Please refer the diagram as follows.



T= time constant

Say, start the periodic output on the standalone mode enabled and stop the output on IN13 and IN14 are active. The program is shown as followings.



10. Communication protocol

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

10.1 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 _EMD8216_UDP_Tdata
{
    u8   card_name[7];          // card_name={'E', 'M', 'D', '8', '2', '1', '6' };
                                // card_name[0]='E' …card_name[6]='6'
    u8   password[8];           // password,8 words
    u8   command;               // command（EMD8216 UDP COMMAND LIST）
    Data      data_in;          // maximum 32 bytes
}EMD8216_data;

typedef union _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];                 //Data（IP Address）
    u16  socket_port;           //Data（Socket Port）
    u8   New_password[8];       //Data（New password,8 words）
    u8   MAC[6];                //Data（MAC Address）
    u8   Port_Value[2];         //Data（Port Value）Port_Value[0] = Port,
                                //                  Port_Value[1] = value
    u8   Point_value[3];        //Data（Point Select）Point_Value[0] = Port,
                                //                    Point_Value[1] = Point
                                //                    Point_Value[2] = Value
```

```
StandaloneData    standalone_data;        // maximum 25 bytes
_WDTData          WDT;                    //Data（WDT wait time & output state）
}Data;

typedef struct _StandaloneData
{
    u8   function_index;          // start function index
    u8   function_number;         // be used max function number
    u8   timer_mode[2];           // set timer mode
    u16  timer_value[2];          // set time constant
    u16  input_point[2];          // choose input IO_00 ~ IO_17
    u16  input_state[2];          // set input state
    u16  output_point[2];         // choose output IO_00 ~ IO_17
    u8   out_mode[2];             // set out mode
    u8   standalone_state;        // 1: standalone Enable; 0: standalone Disable
}StandaloneData

struct _WDTData
{
    u16      Timer_value;         // WDT timer value
    u8       output[2];           // WDT timer out, output data
    u8       state;               // 1: WDT Enable, 0: WDT Disable
};
```

10.2 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 _EMD8216_Rdata
{
    Receive_Data  Data      // Receive Data maximum 32byte
    u8   success_flag;      // Flag （0:Send command Failed;
                            //        0x63:Send command successfully）
    u8   command            // command（EMD8216 UDP COMMAND LIST）
}EMD8216_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    WDT[5]               //Data ( WDT wait time and output state )
    u8    Port_Value[2];       //Data（Port Value）Port_Value[0] = Port,
                               //                  Port_Value[1] = value
    u8    Point_value[3];      //Data（Point Select）Point_Value[0] = Port,
                               //                    Point_Value[1] = Point
                               //                    Point_Value[2] = Point
    u8 Version[2]              //Data ( firmware version )
    StandaloneData    standalone_data;        // maximum 32 bytes
    _WDTData          WDT;                    //Data（WDT wait time & output state）
} Receive_Data
```

10.3 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) */
```

10.4 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) */
```

10.5 EMD-8216 communication commands

- **GET_CARD_TYPE**

Function: ask the EMD8216 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’, ‘D’, ‘8’, ‘2’, ‘1’, ‘6’

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 EMD8216 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’, ‘D’, ‘8’, ‘2’, ‘1’, ‘6’

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 EMD8216 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', 'D', '8', '2', '1', '6'

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 EMD8216 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', 'D', '8', '2', '1', '6'

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 EMD8216 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', 'D', '8', '2', '1', '6'

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 EMD8216 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’, ‘D’, ‘8’, ‘2’, ‘1’, ‘6’

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 EMD8216 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', 'D', '8', '2', '1', '6'

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]	u16	Version x.y
Version[0]		Version[1]: x
		Version[0]: y

- **WRITE_MAC**

Function: the MAC address of EMD8216 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’, ‘D’, ‘8’, ‘2’, ‘1’, ‘6’

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	0xfa
20bytes	8bytes	32bytes	1byte	1byte

Command: 0xfa

Flag: = 0x0 //command fail

= 0x63 //command successful

Data: un-used

- **SET_COUNTER_MASK**

Function: set the counter input mask (enable or disable counter function per channel) of EMD8216 module

Host to Ethernet module:

IP header	UDP header	UDP data			
		Card_name	Password	0x20	Data
20bytes	8bytes	7bytes	8bytes	1byte	32bytes

Card_name: 'E', 'M', 'D', '8', '2', '1', '6'

Password: your password （8 bytes）

Command: 0x20

Data: Port_Value[2] //choose I/O port and counter mask

Parameter	Type	Description
Port_Value[0]	u8	Select port number 0: choose port 0, i.e. select IO_0m 1: choose port 1, i.e. select IO_1m
Port_Value[1]	u8	counter mask Any one work as output, the mask can only set to 0 bit 7: 0: IO_n7 mask off (disable) counter function 1: IO_n7 counter function enabled …. bit 0: 0: IO_n0 mask off (disable) counter function 1: IO_n0 counter function enabled

Notes: Any I/O bit configured as output will not effect by the SET_COUNTER_MASK function.

Ethernet module to Host:

IP header	UDP header	UDP data		
		Data	Flag	0x20
20bytes	8bytes	32bytes	1byte	1byte

Command: 0x20

Flag: = 0x0 //command fail

= 0x63 //command successful

Data: un-used

- **ENABLE_COUNTER_MODE**

Function: global enable counter function of EMD8216 module

Host to Ethernet module:

IP header	UDP header	UDP data			
		Card_name	Password	0x21	Data
20bytes	8bytes	7bytes	8bytes	1byte	32bytes

Card_name: ‘E’, ‘M’, ‘D’, ‘8’, ‘2’, ‘1’, ‘6’

Password: your password （8 bytes）

Command: 0x21

Data: un-used

Ethernet module to Host:

IP header	UDP header	UDP data		
		Data	Flag	0x21
20bytes	8bytes	32bytes	1byte	1byte

Command: 0x21

Flag: = 0x0 //command fail

= 0x63 //command successful

Data: un-used

- **DISABLE_COUNTER_MODE**

Function: global disable counter function of EMD8216 module

Host to Ethernet module:

IP header	UDP header	UDP data			
		Card_name	Password	0x22	Data
20bytes	8bytes	7bytes	8bytes	1byte	32bytes

Card_name: ‘E’, ‘M’, ‘D’, ‘8’, ‘2’, ‘1’, ‘6’

Password: your password （8 bytes）

Command: 0x22

Data: un-used

Ethernet module to Host:

IP header	UDP header	UDP data		
		Data	Flag	0x22
20bytes	8bytes	32bytes	1byte	1byte

Command: 0x22

Flag: = 0x0 //command fail

= 0x63 //command successful

Data: un-used

- **READ_COUNTER**

Function: read the counter data on the fly of EMD8216 module

Host to Ethernet module:

IP header	UDP header	UDP data			
		Card_name	Password	0x23	Data
20bytes	8bytes	7bytes	8bytes	1byte	32bytes

Card_name: 'E', 'M', 'D', '8', '2', '1', '6'

Password: your password （8 bytes）

Command: 0x23

Data: Port_Value[0] // 0 : choose Port0; 1 : choose Port1

other structure members are un-used.

Parameter	Type	Description
Port_Value[0]	u8	Select port number 0: choose port 0, i.e. select IO_0m 1: choose port 1, i.e. select IO_1m

Ethernet module to Host:

IP header	UDP header	UDP data		
		Data	Flag	0x23
20bytes	8bytes	32bytes	1byte	1byte

Command: 0x23

Flag: = 0x0 //command fail

= 0x63 //command successful

Data: data_l[8] //counter value of selected port（Read back Counter value）

Parameter	Type	Description
data_l[8]		counter value of selected port data_l[0] : counter value of IO_n0 … data_l [7] : counter value of IO_n7

- **CLEAR_COUNTER**

Function: clear counter data per channel of EMD8216 module

Host to Ethernet module:

IP header	UDP header	UDP data			
		Card_name	Password	0x24	Data
20bytes	8bytes	7bytes	8bytes	1byte	32bytes

Card_name: 'E', 'M', 'D', '8', '2', '1', '6'

Password: your password （8 bytes）

Command: 0x24

Data: Port_Value[0], Port_Value[1]

other structure members are un-used.

Parameter	Type	Description
Port_Value[0]	u8	Select port number 0: choose port 0, i.e. select IO_0m 1: choose port 1, i.e. select IO_1m
Port_Value[1]	u8	bit 7: 0: unchanged of counter of IO_n7 1: clear counter of IO_n7 …. bit 0: 0: unchanged of counter of IO_n0 1: clear counter of IO_n0

Ethernet module to Host:

IP header	UDP header	UDP data		
		Data	Flag	0x24
20bytes	8bytes	32bytes	1byte	1byte

Command: 0x24

Flag: = 0x0 //command fail

= 0x63 //command successful

Data: un-used

- **SET_PORT_CONFIG**

Function: configure as input or output per channel of EMD8216 module

Host to Ethernet module:

IP header	UDP header	UDP data			
		Card_name	Password	0x30	Data
20bytes	8bytes	7bytes	8bytes	1byte	32bytes

Card_name: 'E', 'M', 'D', '8', '2', '1', '6'

Password: your password （8 bytes）

Command: 0x30

Data: Port_Value[0], Port_Value[1]

other structure members are un-used.

Parameter	Type	Description
Port_Value[0]	u8	Select port number 0: choose port 0, i.e. select IO_0m 1: choose port 1, i.e. select IO_1m
Port_Value[1]	u8	I/O configuration bit 7: 0: IO_n7 configured as output 1: IO_n7 configured as input …. bit 0: 0: IO_n0 configured as output 1: IO_n0 configured as input

Ethernet module to Host:

IP header	UDP header	UDP data		
		Data	Flag	0x30
20bytes	8bytes	32bytes	1byte	1byte

Command: 0x30

Flag: = 0x0 //command fail

= 0x63 //command successful

Data: un-used

- **READ_PORT_CONFIG**

Function: read back the configuration per channel of EMD8216 module

Host to Ethernet module:

IP header	UDP header	UDP data			
		Card_name	Password	0x31	Data
20bytes	8bytes	7bytes	8bytes	1byte	32bytes

Card_name: 'E', 'M', 'D', '8', '2', '1', '6'

Password: your password （8 bytes）

Command: 0x31

Data: Port_Value[0] // 0 : choose Port0; 1 : choose Port1

other structure members are un-used.

Parameter	Type	Description
Port_Value[0]	u8	Select port number 0: choose port 0, i.e. select IO_0m 1: choose port 1, i.e. select IO_1m

Ethernet module to Host:

IP header	UDP header	UDP data		
		Data	Flag	0x31
20bytes	8bytes	32bytes	1byte	1byte

Command: 0x31

Flag: = 0x0 //command fail

= 0x63 //command successful

Data: Port_Value[0] , Port_Value[1]

other structure members are un-used.

Parameter	Type	Description
Port_Value[0]	u8	Port number 0: choose port 0, i.e. select IO_0m 1: choose port 1, i.e. select IO_1m
Port_Value[1]	u8	I/O configuration bit 7: 0: IO_n7 configured as output 1: IO_n7 configured as input …. bit 0: 0: IO_n0 configured as output 1: IO_n0 configured as input

- **SET_PORT**

Function: set the output state per channel of EMD8216 module

Host to Ethernet module:

IP header	UDP header	UDP data			
		Card_name	Password	0x32	Data
20bytes	8bytes	7bytes	8bytes	1byte	32bytes

Card_name: 'E', 'M', 'D', '8', '2', '1', '6'

Password: your password （8 bytes）

Command: 0x32

Data: Port_Value[0] , Port_Value[1]

other structure members are un-used.

Parameter	Type	Description
Port_Value[0]	u8	Port number 0: choose port 0, i.e. select IO_0m 1: choose port 1, i.e. select IO_1m
Port_Value[1]	u8	I/O state (only valid for pins configured as outputs) bit 7: 0: IO_n7 output inactive 1: IO_n7 output active …. bit 0: 0: IO_n0 output inactive 1: IO_n0 output active

Note: An output channel is active may be output active or inactive depends on the polarity it is configured. Say polarity is normal, active means output active and polarity is invert will have active output to be inactive state.

Ethernet module to Host:

IP header	UDP header	UDP data		
		Data	Flag	0x32
20bytes	8bytes	32bytes	1byte	1byte

Command: 0x32

Flag: = 0x0 //command fail

= 0x63 //command successful

Data: un-used

- **READ_PORT**

Function: read the port state per channel of EMD8216 module

Host to Ethernet module:

IP header	UDP header	UDP data			
		Card_name	Password	0x33	Data
20bytes	8bytes	7bytes	8bytes	1byte	32bytes

Card_name: 'E', 'M', 'D', '8', '2', '1', '6'

Password: your password （8 bytes）

Command: 0x33

Data: Port_Value[0] // 0 : choose Port0; 1 : choose Port1

other structure members are un-used.

Parameter	Type	Description
Port_Value[0]	u8	Select port number 0: choose port 0, i.e. select IO_0m 1: choose port 1, i.e. select IO_1m

Ethernet module to Host:

IP header	UDP header	UDP data		
		Data	Flag	0x33
20bytes	8bytes	32bytes	1byte	1byte

Command: 0x33

Flag: = 0x0 //command fail

= 0x63 //command successful

Data: Pot_Value[0], Port_Value[1]

other structure members are un-used.

Parameter	Type	Description
Port_Value[0]	u8	Port number 0: choose port 0, i.e. select IO_0m 1: choose port 1, i.e. select IO_1m
Port_Value[1]	u8	I/O state bit 7: 0: IO_n7 state is inactive 1: IO_n7 state is active …. bit 0: 0: IO_n0 state is inactive 1: IO_n0 state is active

- **SET_POLARITY**

Function: set the I/O polarity per channel of EMD8216 module

Host to Ethernet module:

IP header	UDP header	UDP data			
		Card_name	Password	0x34	Data
20bytes	8bytes	7bytes	8bytes	1byte	32bytes

Card_name: 'E', 'M', 'D', '8', '2', '1', '6'

Password: your password （8 bytes）

Command: 0x34

Data: Pot_Value[0], Port_Value[1]

other structure members are un-used.

Parameter	Type	Description
Port_Value[0]	u8	Port number 0: choose port 0, i.e. select IO_0m 1: choose port 1, i.e. select IO_1m
Port_Value[1]	u8	I/O polarity bit 7: 0: IO_n7 normal polarity 1: IO_n7 invert polarity …. bit 0: 0: IO_n0 normal polarity 1: IO_n0 invert polarity

Note: An output channel is active may be output high or low depends on the polarity it is configured. Say polarity is normal, active means output high and polarity is invert will have active output to be low state.

Ethernet module to Host:

IP header	UDP header	UDP data		
		Data	Flag	0x34
20bytes	8bytes	32bytes	1byte	1byte

Command: 0x34

Flag: = 0x0 //command fail

= 0x63 //command successful

Data: un-used

- **READ_POLARITY**

Function: read the I/O polarity per channel of EMD8216 module

Host to Ethernet module:

IP header	UDP header	UDP data			
		Card_name	Password	0x35	Data
20bytes	8bytes	7bytes	8bytes	1byte	32bytes

Card_name: 'E', 'M', 'D', '8', '2', '1', '6'

Password: your password （8 bytes）

Command: 0x35

Data: Port_Value[0] // 0 : choose Port0; 1 : choose Port1

other structure members are un-used.

Parameter	Type	Description
Port_Value[0]	u8	Port number 0: choose port 0, i.e. select IO_0m 1: choose port 1, i.e. select IO_1m

Ethernet module to Host:

IP header	UDP header	UDP data		
		Data	Flag	0x35
20bytes	8bytes	32bytes	1byte	1byte

Command: 0x35

Flag: = 0x0 //command fail

= 0x63 //command successful

Data: Pot_Value[0], Port_Value[1]

other structure members are un-used.

Parameter	Type	Description
Port_Value[0]	u8	Port number 0: choose port 0, i.e. select IO_0m 1: choose port 1, i.e. select IO_1m
Port_Value[1]	u8	I/O polarity bit 7: 0: IO_n7 normal polarity 1: IO_n7 invert polarity …. bit 0: 0: IO_n0 normal polarity 1: IO_n0 invert polarity

Note: An output channel is active may be output high or low depends on the polarity it is configured. Say polarity is normal, active means output high and polarity is invert will have active output to be low state.

- **SET_POINT_CONFIG**

Function: configure channel as input or output of EMD8216 module

Host to Ethernet module:

IP header	UDP header	UDP data			
		Card_name	Password	0x36	Data
20bytes	8bytes	7bytes	8bytes	1byte	32bytes

Card_name: 'E', 'M', 'D', '8', '2', '1', '6'

Password: your password （8 bytes）

Command: 0x36

Data: Point_Value[0], Point_Value[1],Point_value[2]
other structure members are un-used.

Parameter	Type	Description
Point_Value[0]	u8	Select port number 0: choose port 0, i.e. select IO_0m 1: choose port 1, i.e. select IO_1m
Point_Value[1]	u8	point (channel) selection 0: select IO_n0 …. 7: select IO_n7
Point_Value[2]	u8	point (channel) configuration 0: configure as output 1: configure as input

Ethernet module to Host:

IP header	UDP header	UDP data		
		Data	Flag	0x36
20bytes	8bytes	32bytes	1byte	1byte

Command: 0x36

Flag: = 0x0 //command fail
= 0x63 //command successful

Data: un-used

- **READ_POINT_CONFIG**

Function: read back the channel configuration of EMD8216 module

Host to Ethernet module:

IP header	UDP header	UDP data			
		Card_name	Password	0x37	Data
20bytes	8bytes	7bytes	8bytes	1byte	32bytes

Card_name: 'E', 'M', 'D', '8', '2', '1', '6'

Password: your password （8 bytes）

Command: 0x37

Data: Point_Value[0], Point_Value[1]
other structure members are un-used.

Parameter	Type	Description
Point_Value[0]	u8	Select port number 0: choose port 0, i.e. select IO_0m 1: choose port 1, i.e. select IO_1m
Point_Value[1]	u8	point (channel) selection 0: select IO_n0 …. 7: select IO_n7

Ethernet module to Host:

IP header	UDP header	UDP data		
		Data	Flag	0x37
20bytes	8bytes	32bytes	1byte	1byte

Command: 0x37

Flag: = 0x0 //command fail
= 0x63 //command successful

Data: Point_Value[0], Point_Value[1], Point_value[2]
other structure members are un-used.

Parameter	Type	Description
Point_Value[0]	u8	Select port number 0: choose port 0, i.e. select IO_0m 1: choose port 1, i.e. select IO_1m
Point_Value[1]	u8	point (channel) selection 0: select IO_n0 …. 7: select IO_n7
Point_Value[2]	u8	point (channel) configuration 0: configure as output 1: configure as input

- **SET_POINT**

Function: set the output state of a channel of EMD8216 module

Host to Ethernet module:

IP header	UDP header	UDP data			
		Card_name	Password	0x38	Data
20bytes	8bytes	7bytes	8bytes	1byte	32bytes

Card_name: 'E', 'M', 'D', '8', '2', '1', '6'

Password: your password （8 bytes）

Command: 0x38

Data: Point_Value[0], Point_Value[1], Point_value[2]
other structure members are un-used.

Parameter	Type	Description
Point_Value[0]	u8	Port number 0: choose port 0, i.e. select IO_0m 1: choose port 1, i.e. select IO_1m
Point_Value[1]	u8	point (channel) selection 0: select IO_n0 …. 7: select IO_n7
Point_Value[2]	u8	point (channel) configuration 0: inactive 1: active

Note: An output channel is active may be output high or low depends on the polarity it is configured. Say polarity is normal, active means output high and polarity is invert will have active output to be low state.

Ethernet module to Host:

IP header	UDP header	UDP data		
		Data	Flag	0x38
20bytes	8bytes	32bytes	1byte	1byte

Command: 0x38

Flag: = 0x0 //command fail
= 0x63 //command successful

Data: un-used

- **READ_POINT**

Function: read the channel state of EMD8216 module

Host to Ethernet module:

IP header	UDP header	UDP data			
		Card_name	Password	0x39	Data
20bytes	8bytes	7bytes	8bytes	1byte	32bytes

Card_name: 'E', 'M', 'D', '8', '2', '1', '6'

Password: your password （8 bytes）

Command: 0x39

Data: Point_Value[0], Point_Value[1]
other structure members are un-used.

Parameter	Type	Description
Point_Value[0]	u8	Port number 0: choose port 0, i.e. select IO_0m 1: choose port 1, i.e. select IO_1m
Point_Value[1]	u8	point (channel) selection 0: select IO_n0 …. 7: select IO_n7

Ethernet module to Host:

IP header	UDP header	UDP data		
		Data	Flag	0x39
20bytes	8bytes	32bytes	1byte	1byte

Command: 0x39

Flag: = 0x0 //command fail
= 0x63 //command successful

Data: Point_Value[0], Point_Value[1], Point_value[2]
other structure members are un-used.

Parameter	Type	Description
Point_Value[0]	u8	Port number 0: choose port 0, i.e. select IO_0m 1: choose port 1, i.e. select IO_1m
Point_Value[1]	u8	point (channel) selection 0: select IO_n0 …. 7: select IO_n7
Point_Value[2]	u8	point (channel) configuration 0: inactive 1: active

- **SET_POINT_POLARITY**

Function: set the I/O polarity of a channel of EMD8216 module

Host to Ethernet module:

IP header	UDP header	UDP data			
		Card_name	Password	0x3A	Data
20bytes	8bytes	7bytes	8bytes	1byte	32bytes

Card_name: 'E', 'M', 'D', '8', '2', '1', '6'

Password: your password （8 bytes）

Command: 0x3A

Data: Point_Value[0], Point_Value[1], Point_value[2]
other structure members are un-used.

Parameter	Type	Description
Point_Value[0]	u8	Port number 0: choose port 0, i.e. select IO_0m 1: choose port 1, i.e. select IO_1m
Point_Value[1]	u8	point (channel) selection 0: select IO_n0 …. 7: select IO_n7
Point_Value[2]	u8	point (channel) polarity 0: normal 1: invert

Note: An output channel is active may be output high or low depends on the polarity it is configured. Say polarity is normal, active means output high and polarity is invert will have active output to be low state.

Ethernet module to Host:

IP header	UDP header	UDP data		
		Data	Flag	0x3A
20bytes	8bytes	32bytes	1byte	1byte

Command: 0x3A

Flag: = 0x0 //command fail
= 0x63 //command successful

Data: un-used

- **READ_POINT_POLARITY**

Function: read the channel polarity of EMD8216 module

Host to Ethernet module:

IP header	UDP header	UDP data			
		Card_name	Password	0x3B	Data
20bytes	8bytes	7bytes	8bytes	1byte	32bytes

Card_name: 'E', 'M', 'D', '8', '2', '1', '6'

Password: your password （8 bytes）

Command: 0x3B

Data: Point_Value[0], Point_Value[1]

other structure members are un-used.

Parameter	Type	Description
Point_Value[0]	u8	Port number 0: choose port 0, i.e. select IO_0m 1: choose port 1, i.e. select IO_1m
Point_Value[1]	u8	point (channel) selection 0: select IO_n0 …. 7: select IO_n7

Ethernet module to Host:

IP header	UDP header	UDP data		
		Data	Flag	0x3B
20bytes	8bytes	32bytes	1byte	1byte

Command: 0x3B

Flag: = 0x0 //command fail

= 0x63 //command successful

Data: Point_Value[0], Point_Value[1], Point_value[2]

other structure members are un-used.

Parameter	Type	Description
Point_Value[0]	u8	Port number 0: choose port 0, i.e. select IO_0m 1: choose port 1, i.e. select IO_1m
Point_Value[1]	u8	point (channel) selection 0: select IO_n0 …. 7: select IO_n7
Point_Value[2]	u8	point (channel) polarity 0: normal 1: invert

- **ENABLE_STANDALONE**

Function: enable standalone mode operation of EMD8216 module

Host to Ethernet module:

IP header	UDP header	UDP data			
		Card_name	Password	0x50	Data
20bytes	8bytes	7bytes	8bytes	1byte	32bytes

Card_name: ‘E’, ‘M’, ‘D’, ‘8’, ‘2’, ‘1’, ‘6’

Password: your password （8 bytes）

Command: 0x50

Data: un-used

Ethernet module to Host:

IP header	UDP header	UDP data		
		Data	Flag	0x50
20bytes	8bytes	32bytes	1byte	1byte

Command: 0x50

Flag: = 0x0 //command fail

= 0x63 //command successful

Data: un-used

- **DISABLE_STANDALONE**

Function: disable standalone mode operation of EMD8216 module

Host to Ethernet module:

IP header	UDP header	UDP data			
		Card_name	Password	0x51	Data
20bytes	8bytes	7bytes	8bytes	1byte	32bytes

Card_name: 'E', 'M', 'D', '8', '2', '1', '6'

Password: your password （8 bytes）

Command: 0x51

Data: un-used

Ethernet module to Host:

IP header	UDP header	UDP data		
		Data	Flag	0x51
20bytes	8bytes	32bytes	1byte	1byte

Command: 0x51

Flag: = 0x0 //command fail

= 0x63 //command successful

Data: un-used

- **SET_STANDALONE_CONFIG**

Function: configure standalone mode operation of EMD8216 module

Host to Ethernet module:

IP header	UDP header	UDP data			
		Card_name	Password	0x52	Data
20bytes	8bytes	7bytes	8bytes	1byte	32bytes

Card_name: 'E', 'M', 'D', '8', '2', '1', '6'

Password: your password （8 bytes）

Command: 0x52

Data: StandaloneData standalone_data; //standalone instruction and operation mode

Parameter	Type	Description
function_index	u8	command line index of current communication. Actually one communication data will carry 2 commands. The function_index is the first command's line number. allowable range: 0 ~ 31 (command line index)
function_number	u8	Standalone mode total commands. allowable range: 1 ~ 32 (commands)
timer_mode[0] timer_mode[1]	u8	timer working mode of current command: =0x0 :timer unused =0x1 : Input action and delay out, =0x2 : Input action and pulse out =0x3 : Input action and periodic out =0x4 : Timer action and delay out =0x5 : Timer action and pulse out =0x6 : Timer action and periodic out =0x7 : Timer off please refer: chapter 9 Standalone mode application examples
timer_value[0] timer_value[1]	u16	time constant based on 5ms click tick. allowable range: 1 ~ 65535 (5ms ~ 327675ms)
input_point[0] input_point[1]	u16	The input points your process wants to watch. The high byte data (b7~b0) is IO_17 ~ IO_10, low byte data (b7~b0) is IO_07 ~ IO_00
input_state[0] input_state[1]	u16	The input states that your process will trigger timer or output. The high byte data (b7~b0) is the state of IO_17 ~ IO_10, low byte data (b7~b0) is the state of IO_07 ~ IO_00

output_point[0] output_point[1]	u16	The output states that your process will trigger while the input states meet your preset. The high byte data (b7~b0) is the state of IO_17 ~ IO_10, low byte data (b7~b0) is the state of IO_07 ~ IO_00
output_mode[0] output_mode[1]	u8	output modes, which depends on the timer mode set. If timer_mode = unused or delay out, =0x0 : output inactive =0x1 : output active =0x2 : output change state If timer_mode = pulse out, =0x0 : inactive state pulse =0x1 : active state pulse If timer_mode = periodic out, =0x2 : output change state
standalone_state	u8	=0x0 : power on standalone disable =0x1 : power on standalone enable

Note: The communication data will maximum carries 2 commands, the first command consists of: timer_mode[0], timer_value[0], input_point[0], input_state[0], output_point[0], output_mode[0]; the second command consist of: timer_mode[1], timer_value[1], input_point[1], input_state[1], output_point[1], output_mode[1].

Ethernet module to Host:

IP header	UDP header	UDP data		
		Data	Flag	0x52
20bytes	8bytes	32bytes	1byte	1byte

Command: 0x52

Flag: = 0x0 //command fail

= 0x63 //command successful

Data: un-used

- **READ_STANDALONE_CONFIG**

Function: read back standalone commands from EMD8216 module

Host to Ethernet module:

IP header	UDP header	UDP data			
		Card_name	Password	0x53	Data
20bytes	8bytes	7bytes	8bytes	1byte	32bytes

Card_name: 'E', 'M', 'D', '8', '2', '1', '6'

Password: your password （8 bytes）

Command: 0x53

Data: function_index

other structure members are un-used.

Parameter	Type	Description
function_index	u8	command line index of requested read back command. allowable range: 0 ~ 31 (command line index)

Ethernet module to Host:

IP header	UDP header	UDP data		
		Data	Flag	0x53
20bytes	8bytes	32bytes	1byte	1byte

Command: 0x53

= 0x63 //command successful

Flag: = 0x0 //command fail

Data: StandaloneData standalone_data; //standalone instruction and operation mode

Parameter	Type	Description
function_index	u8	command line index of current communication. Actually one communication data will carry 2 commands. The function_index is the first command's line number. allowable range: 0 ~ 31 (command line index)
function_number	u8	Total commands on the module. allowable range: 1 ~ 32 (commands)
timer_mode[0] timer_mode[1]	u8	timer working mode of current command: =0x0 :timer unused =0x1 : Input action and delay out, =0x2 : Input action and pulse out =0x3 : Input action and periodic out =0x4 : Timer action and delay out =0x5 : Timer action and pulse out

		=0x6 : Timer action and periodic out =0x7 : Timer off please refer: chapter 9 Standalone mode application examples
timer_value[0] timer_value[1]	u16	time constant based on 5ms click tick. allowable range: 1 ~ 65535 (5ms ~ 327675ms)
input_point[0] input_point[1]	u16	The input points your process wants to watch. The high byte data (b7~b0) is IO_17 ~ IO_10, low byte data (b7~b0) is IO_07 ~ IO_00
input_state[0] input_state[1]	u16	The input states that your process will trigger timer or output. The high byte data (b7~b0) is the state of IO_17 ~ IO_10, low byte data (b7~b0) is the state of IO_07 ~ IO_00
output_point[0] output_point[1]	u16	The output states that your process will trigger while the input states meet your preset. The high byte data (b7~b0) is the state of IO_17 ~ IO_10, low byte data (b7~b0) is the state of IO_07 ~ IO_00
output_mode[0] output_mode[1]	u8	output modes, which depends on the timer mode set. If timer_mode = unused or delay out, =0x0 : output inactive =0x1 : output active =0x2 : output change state If timer_mode = pulse out, =0x0 : inactive state pulse =0x1 : active state pulse If timer_mode = periodic out, =0x2 : output change state
standalone_state	u8	=0x0 : power on standalone disable =0x1 : power on standalone enable

- **ENABLE_WDT**

Function: enable WDT (watchdog timer) operation of EMD8216 module

Host to Ethernet module:

IP header	UDP header	UDP data			
		Card_name	Password	0x60	Data
20bytes	8bytes	7bytes	8bytes	1byte	32bytes

Card_name: ‘E’, ‘M’, ‘D’, ‘8’, ‘2’, ‘1’, ‘6’

Password: your password （8 bytes）

Command: 0x60

Data: un-used

Ethernet module to Host:

IP header	UDP header	UDP data		
		Data	Flag	0x60
20bytes	8bytes	32bytes	1byte	1byte

Command: 0x60

Flag: = 0x0 //command fail

= 0x63 //command successful

Data: un-used

- **DISABLE_WDT**

Function: disable WDT (watchdog timer) operation of EMD8216 module

Host to Ethernet module:

IP header	UDP header	UDP data			
		Card_name	Password	0x61	Data
20bytes	8bytes	7bytes	8bytes	1byte	32bytes

Card_name: ‘E’, ‘M’, ‘D’, ‘8’, ‘2’, ‘1’, ‘6’

Password: your password （8 bytes）

Command: 0x61

Data: un-used

Ethernet module to Host:

IP header	UDP header	UDP data		
		Data	Flag	0x61
20bytes	8bytes	32bytes	1byte	1byte

Command: 0x61

Flag: = 0x0 //command fail

= 0x63 //command successful

Data: un-used

- **SET_WDT**

Function: setup WDT (watchdog timer) operation of EMD8216 module

Host to Ethernet module:

IP header	UDP header	UDP data			
		Card_name	Password	0x62	Data
20bytes	8bytes	7bytes	8bytes	1byte	32bytes

Card_name: 'E', 'M', 'D', '8', '2', '1', '6'

Password: your password （8 bytes）

Command: 0x62

Data: _WDTData WDT; //standalone instruction and operation mode.

Parameter	Description
output[0]	IO_00 ~ IO_07 state at WDT time out bit0: 0: IO_00 is inactive at WDT time out 1: IO_00 is active at WDT time out … bit7: 0: IO_07 is inactive at WDT time out 1: IO_07 is active at WDT time out
output[1]	IO_10 ~ IO_17 state at WDT time out bit0: 0: IO_10 is inactive at WDT time out 1: IO_10 is active at WDT time out … bit7: 0: IO_17 is inactive at WDT time out 1: IO_17 is active at WDT time out
Timer_value	WDT time constant at 0.1second time base allowable range: 10 (1.0s) ~ 10000 (1000.0s)

Ethernet module to Host:

IP header	UDP header	UDP data		
		Data	Flag	0x62
20bytes	8bytes	32bytes	1byte	1byte

Command: 0x62

Flag: = 0x0 //command fail

= 0x63 //command successful

Data: un-used

- **READ_WDT**

Function: read WDT (watchdog timer) counter of EMD8216 module

Host to Ethernet module:

IP header	UDP header	UDP data			
		Card_name	Password	0x63	Data
20bytes	8bytes	7bytes	8bytes	1byte	32bytes

Card_name: 'E', 'M', 'D', '8', '2', '1', '6'

Password: your password （8 bytes）

Command: 0x63

Data: un-used

Ethernet module to Host:

IP header	UDP header	UDP data		
		Data	Flag	0x63
20bytes	8bytes	32bytes	1byte	1byte

Command: 0x63

Flag: = 0x0 //command fail

= 0x63 //command successful

Data: _WDTData WDT; //standalone instruction and operation mode

Parameter	Description
output[0]	IO_00 ~ IO_07 state at WDT time out bit0: 0: IO_00 is inactive at WDT time out 1: IO_00 is active at WDT time out … bit7: 0: IO_07 is inactive at WDT time out 1: IO_07 is active at WDT time out
output[1]	IO_10 ~ IO_17 state at WDT time out bit0: 0: IO_10 is inactive at WDT time out 1: IO_10 is active at WDT time out … bit7: 0: IO_17 is inactive at WDT time out 1: IO_17 is active at WDT time out
Timer_value	WDT time constant at 0.1second time base allowable range: 10 (1.0s) ~ 10000 (1000.0s)
state	0:WDT Disable 1:WDT Enable

11. DLL list

	Function Name	Description
1.	EMD8216_initial( )	Assign IP and get model parameter
2.	EMD8216_close( )	EMD8216 close
3.	EMD8216_firmware_version_read( )	Read the firmware version
4.	EMD8216_port_config_set( )	Set the Configure of the I/O port.
5.	EMD8216_port_config_read( )	Read back the Configure of the I/O port.
6.	EMD8216_port_polarity_set( )	Set the I/O port polarity.
7.	EMD8216_port_polarity_read( )	Read back the polarity setting of the I/O port
8.	EMD8216_port_set( )	Set the output port data.
9.	EMD8216_port_read( )	Read back the data of the I/O port.
10.	EMD8216_config_point_set( )	Set bit Configure of I/O point
11.	EMD8216_config_point _read( )	Read bit Configure of I/O point
12.	EMD8216_point_polarity_set( )	Set the I/O point polarity.
13.	EMD8216_point_polarity_read( )	Read back the polarity setting of the I/O point.
14.	EMD8216_point_set( )	Set bit status of output point
15.	EMD8216_point_read( )	Read bit state of I/O point.
16.	EMD8216_counter_mask_set( )	Set counter channel mask
17.	EMD8216_counter_enable( )	Enable counter function
18.	EMD8216_counter_disable( )	Disable counter function
19.	EMD8216_counter_read( )	Read counter value
20.	EMD8216_counter_clear( )	Clear designated counter
21.	EMD8216_change_socket_port( )	change the communication port
22.	EMD8216_change_IP( )	change the IP of EMD8216
23.	EMD8216_reboot( )	reboot EMD8216 module
24.	EMD8216_security_unlock( )	Unlock security
25.	EMD8216_security_status_read( )	Read lock status
26.	EMD8216_password_change( )	Change password
27.	EMD8216_password_set_default( )	Rest to factory default password
28.	EMD8216_WDT_set( )	Set up WDT timer and output states
29.	EMD8216_WDT_read( )	Read WDT timer and output state setting
30.	EMD8216_WDT_enable( )	Enable WDT function
31.	EMD8216_WDT_disable( )	Disable WDT function
32.	EMD8216_standalone_enable( )	Enable standalone mode
33.	EMD8216_standalone_disable( )	Disable (stop) standalone mode
34.	EMD8216_standalone_config_set( )	Set the standalone configuration
35.	EMD8216_standalone_config_read( )	Read the standalone configuration

12. EMD8216 Error codes summary

12.1 EMD8216 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	Card ID already used
300	ID_ERROR	Card ID 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	Cannot change socket
307	UNLOCK_SECURITY_ERROR	Fail to unlock security
308	PASSWORD_ERROR	Password mismatched
309	REBOOT_ERROR	Cannot reboot
310	TIME_OUT_ERROR	Too long to response
311	CREAT_SOCKET_ERROR	Socket cannot create
312	CHANGE_IP_ERROR	Change IP error
313	MASK_ERROR	Set mask error
314	COUNTER_ENABLE_ERROR	Cannot enable counter
315	COUNTER_DISABLE_ERROR	Cannot disable counter
316	COUNTER_READ_ERROR	Fail to read counter
317	COUNTER_CLEAR_ERROR	Fail to clear counter
318	TIME_ERROR	Set the time error
320	CARD_VERSION_ERROR	Cannot read firmware version
321	STANDALONE_ENABLE_ERROR	Cannot enable standalone
322	STANDALONE_DISABLE_ERROR	Cannot disable standalone
323	STANDALONE_CONFIG_ERROR	Cannot setup / read standalone configuration

13. UDP communication command list

command code	R/W	Mnemonics	Descriptions
0x1	R	GET_CARD_TYPE	Get Card Type
0x2	W	REBOOT	Soft Reboot
0x3	W	CHANGE_SOCKETPORT	Change Socket Port
0x4	W	CHANGE_PASSWORD	Change password
0x5	W	RESTORE_PASSWORD	Restore password
0x6	W	CHANGE_IP	Change IP Address
0x7	R	GET_FIRMWARE_VERSION	Get firmware version
0xFA	W	WRITE_MAC	Write MAC Address to EEPROM
0x20	W	SET_COUNTER_MASK	Set counter mask
0x21	W	ENABLE_COUNTER_MODE	Enable counter mode
0x22	W	DISABLE_COUNTER_MODE	Disable counter mode
0x23	R	READ_COUNTER	Read counter
0x24	W	CLEAR_COUNTER	Clear counter
0x30	W	SET_PORT_CONFIG	Set Port configuration
0x31	R	READ_PORT_CONFIG	Read Port configuration
0x32	W	SET_PORT	Set Port
0x33	R	READ_PORT	Read Port
0x34	W	SET_POLARITY	Set polarity
0x35	R	READ_POLARITY	Read polarity
0x36	W	SET_POINT_CONFIG	Set Point configuration
0x37	R	READ_POINT_CONFIG	Read Point configuration
0x38	W	SET_POINT	Set Point
0x39	R	READ_POINT	Read Point
0x3A	W	SET_POINT_POLARITY	Set Point polarity
0x3B	R	READ_POINT_ POLARITY	Read Point polarity
0x50	W	ENABLE_STANDALONE	Enable standalone function
0x51	W	DISABLE_STANDALONE	Disable standalone function
0x52	W	SET_STANDALONE_CONFIG	Setup standalone configuration
0x53	R	READ_STANDALONE_CONFIG	Read standalone configuration
0x60	W	ENABLE WDT	Enable WDT
0x61	W	DISABLE WDT	Disable WDT
0x62	W	SET WDT	Set WDT
0x63	R	READ WDT	Read WDT

14. UDP Error codes summary

Error code	Symbolic Name	Description
99	SUCCESS	No Error
100	COMMAND_ERROR	Command Error
101	PASSWORD_ERROR	Password Error
102	CHANGE_IP_ERROR	Set IP value error out of range Range：LSB = 1 ~ 254
103	CHANGE_SOCKET_ERROR	Set socket port error out of range Range：value > 1000
104	CHANGE_MAC_ERROR	Set mac address error mac != FF FF FF FF FF FF
120	PORT_ERROR	Choose Port error out of range Range：port < port_max
121	POINT_ERROR	Choose point error out of range Range：point < point_max
122	STATE_ERROR	Set state error State = 1 or 0
123	TIMER_VALUE_ERROR	Set Timer value error out of range Range：value_min < value < value_max
124	MODE_ERROR	Choose mode error out of range Range：mode < mode_max
150	INDEX_ERROR	Standalone index error Out of range Range：0 ≦ index ＜ number
151	NUMBER_ERROR	Standalone number error Out of range Range：1 ≦ number ≦ 32
152	TIMER_MODE_ERROR	Standalone Timer mode error Out of range Range：0x0 ~ 0x7
153	OUT_MODE_ERROR	Standalone output mode error out of range Range：0x0 ~ 0x2
154	INPUT_SETTING_ERROR	Function input must be choose Lost input setting
155	OUTPUT_SETTING_ERROR	Function output must be choose Lost output setting
156	POWER_ON_ERROR	Power_on state set error Out of range State = 0 or 1

15. MODBus

The MODBus protocol is widely accepted in SCADA application software, the EMD8216 also provide the protocol that enables the module to connect to existing software without much effort. The MODBus TCP protocol requires to communicate through port 502 as default. The MODBus function code and the address number are list as followings.

Input register: (function code is 04)

Address Number	data type	R/W	mnemonic	description
30001	2 byte	RO	INPUT	low byte is port0, high byte is port1 any bit = 1 active state = 0 inactive state
30002	2byte	RO	COUNTER00_H	Port0 counter data
30003	2byte	RO	COUNTER00_L	
30004	2byte	RO	COUNTER01_H	
30005	2byte	RO	COUNTER01_L	
30006	2byte	RO	COUNTER02_H	
30007	2byte	RO	COUNTER02_L	
30008	2byte	RO	COUNTER03_H	
30009	2byte	RO	COUNTER03_L	
30010	2byte	RO	COUNTER04_H	
30011	2byte	RO	COUNTER04_L	
30012	2byte	RO	COUNTER05_H	
30013	2byte	RO	COUNTER05_L	
30014	2byte	RO	COUNTER06_H	
30015	2byte	RO	COUNTER06_L	
30016	2byte	RO	COUNTER07_H	
30017	2 byte	RO	COUNTER07_L	
30018	2byte	RO	COUNTER10_H	Port1 counter data
30019	2byte	RO	COUNTER10_L	
30020	2byte	RO	COUNTER11_H	
30021	2byte	RO	COUNTER11_L	
30022	2byte	RO	COUNTER12_H	
30023	2byte	RO	COUNTER12_L	
30024	2byte	RO	COUNTER13_H	
30025	2byte	RO	COUNTER13_L	
30026	2byte	RO	COUNTER14_H	
30027	2byte	RO	COUNTER14_L	
30028	2byte	RO	COUNTER15_H	

30029	2byte	RO	COUNTER15_L	
30030	2byte	RO	COUNTER16_H	
30031	2byte	RO	COUNTER16_L	
30032	2byte	RO	COUNTER17_H	
30033	2 byte	RO	COUNTER17_L	

Configure Register: (function code is 03、06、16)

Address Number	data type	R/W	Mnemonic	description
40001	2 byte	R/W	PORT_CONFIG	b0~ b7: DIO00 ~ DIO07 b8 ~ b15: DIO10 ~ DIO17 any bit of PORT_CONFIG =1, configured as input =0, configured as output
40002	2 byte	R/W	POLARITY	b0~ b7: DIO00 ~ DIO07 b8 ~ b15: DIO10 ~ DIO17 any bit of POLARITY = 1, invert the corresponding DIO bit = 0, pass the corresponding DIO bit

WDT Register: (function code is 03、06、16)

Address Number	data type	R/W	Mnemonic	description
40003	2byte	R/W	WDT_TIMER_ CONSTANT	Time = timer_constant * 100(mS) 10 ≦ timer_constant ≦10000
40004	2byte	R/W	WDT_OUTPUT_ STATE	b0~ b7: DIO00 ~ DIO07 b8 ~ b15: DIO10 ~ DIO17 any bit = 1, active state = 0, inactive state at WDT time out
40005	2byte	R/W	WDT_ENABLE	0: disable 1: enable

Counter Register: (function code is 03、06、16)

Address Number	data type	R/W	mnemonic	description
40006	2byte	R/W	COUNTER_ENABLE	0: disable 1: enable
40007	2byte	W	COUNTER_CLEAR	b0 ~ b7: counter00 ~ counter07 b8 ~ b15: counter10 ~ counter17 any bit=1 will clear the corresponding counter
40008	2byte	R/W	COUNTER_MASK	b0 ~ b7: counter00 ~ counter07 b8 ~ b15: counter10 ~ counter17 any bit = 0 mask off the corresponding counter

Standalone register: (function code is 03、06、16)

Address Number	data type	R/W	mnemonic	Description
40009	2 byte	R/W	STANDALONE_ENABLE	standalone mode enable/disable 0: disable 1: enable
40010	2 byte	R/W	STANDALONE_NUMBER	Standalone function max number.
STANDALONE_FUNCTION0 (12 bytes)				**Timer mode** b2~b0 is choose mode: 0x0 = Unused, 0x1 = Input action and delay out, 0x2 = Input action and pulse out 0x3 = Input action and periodic out 0x4 = Timer action and delay out 0x5 = Timer action and pulse out 0x6 = Timer action and periodic out 0x7 = Timer off b15~b3 is unused **Timer value** if timer_mode = delay / periodic out the value is setting for delay timer if timer_mode = pulse out the value is setting for working time **Input point** b0 ~ b7: DIO00 ~ DIO07 b8 ~ b15: DIO10 ~ DIO17 any bit=1 will select as concerned input **Input state** b0 ~ b7: DIO00 ~ DIO07 b8 ~ b15: DIO10 ~ DIO17 0: INACTIVE, for the correspond input bit 1: ACTIVE, for the correspond input bit **Output point** b0 ~ b7: DIO00 ~ DIO07
40011	2 byte	R/W	STANDALONE_FUNCTION0_ TIMER_MODE	
40012	2 byte	R/W	STANDALONE_FUNCTION0_ TIMER_VALUE	
40013	2 byte	R/W	STANDALONE_FUNCTION0_ INPUT_POINT	
40014	2 byte	R/W	STANDALONE_FUNCTION0_ INPUT_STATE	
40015	2 byte	R/W	STANDALONE_FUNCTION0_ OUTPUT_POINT	
40016	2 byte	R/W	STANDALONE_FUNCTION0_ OUTPUT_MODE	
40017 ~ 40022	Total 12 byte	R/W	STANDALONE_FUNCTION1	
40023 ~ 40028	Total 12 byte	R/W	STANDALONE_FUNCTION2	
40029 ~ 40034	Total 12 byte	R/W	STANDALONE_FUNCTION3	
40035 ~ 40040	Total 12 byte	R/W	STANDALONE_FUNCTION4	
40041 ~ 40046	Total 12 byte	R/W	STANDALONE_FUNCTION5	
40047 ~ 40052	Total 12 byte	R/W	STANDALONE_FUNCTION6	
40053 ~ 40058	Total 12 byte	R/W	STANDALONE_FUNCTION7	
40059 ~ 40064	Total 12 byte	R/W	STANDALONE_FUNCTION8	
40065 ~ 40070	Total 12 byte	R/W	STANDALONE_FUNCTION9	

40071 ~ 40076	Total 12 byte	R/W	STANDALONE_FUNCTION10	b8 ~ b15: DIO10 ~ DIO17 any bit=1 will select as concerned output **Output mode** if timer_mode = Unused 0x0= INACTIVE 0x1= ACTIVE 0x2=Change if timer_mode = delay out, 0x0= INACTIVE 0x1= ACTIVE 0x2=Change if timer_mode = pulse out, not use if timer_mode = periodic out, 0x2 = Change
40077 ~ 40082	Total 12 byte	R/W	STANDALONE_FUNCTION11	
40083 ~ 40088	Total 12 byte	R/W	STANDALONE_FUNCTION12	
40089 ~ 40094	Total 12 byte	R/W	STANDALONE_FUNCTION13	
40095 ~ 40100	Total 12 byte	R/W	STANDALONE_FUNCTION14	
40101 ~ 40106	Total 12 byte	R/W	STANDALONE_FUNCTION15	
40107 ~ 40112	Total 12 byte	R/W	STANDALONE_FUNCTION16	
40113 ~ 40118	Total 12 byte	R/W	STANDALONE_FUNCTION17	
40119 ~ 40124	Total 12 byte	R/W	STANDALONE_FUNCTION18	
40125 ~ 40130	Total 12 byte	R/W	STANDALONE_FUNCTION19	
40131 ~ 40136	Total 12 byte	R/W	STANDALONE_FUNCTION20	
40137 ~ 40142	Total 12 byte	R/W	STANDALONE_FUNCTION21	
40143 ~ 40148	Total 12 byte	R/W	STANDALONE_FUNCTION22	
40149 ~ 40154	Total 12 byte	R/W	STANDALONE_FUNCTION23	
40155 ~ 40160	Total 12 byte	R/W	STANDALONE_FUNCTION24	
40161 ~ 40166	Total 12 byte	R/W	STANDALONE_FUNCTION25	
40167 ~ 40172	Total 12 byte	R/W	STANDALONE_FUNCTION26	
40173 ~	Total 12 byte	R/W	STANDALONE_FUNCTION27	

40178				
40179 ~ 40184	Total 12 byte	R/W	STANDALONE_FUNCTION28	
40185 ~ 40190	Total 12 byte	R/W	STANDALONE_FUNCTION29	
40191 ~ 40196	Total 12 byte	R/W	STANDALONE_FUNCTION30	
40197 ~ 40202	Total 12 byte	R/W	STANDALONE_FUNCTION31	
40203	2 byte	R/W	STANDALONE_POWER_ON	Enable/disable standalone mode working after power on (without external command) 0: disable 1: enable

Output register: (function code is 01、05、15)

Address number	data type	R/W	Mnemonic	Description
00001	1 bit	R/W	OUTPUT_00	while write 0: output inactive 1: output active while read if the point is output 0: output state is inactive 1: output state is active
00002	1 bit	R/W	OUTPUT_01	
00003	1 bit	R/W	OUTPUT_02	
00004	1 bit	R/W	OUTPUT_03	
00005	1 bit	R/W	OUTPUT_04	
00006	1 bit	R/W	OUTPUT_05	
00007	1 bit	R/W	OUTPUT_06	
00008	1 bit	R/W	OUTPUT_07	
00009	1 bit	R/W	OUTPUT_10	
00010	1 bit	R/W	OUTPUT_11	
00011	1 bit	R/W	OUTPUT_12	
00012	1 bit	R/W	OUTPUT_13	
00013	1 bit	R/W	OUTPUT_14	
00014	1 bit	R/W	OUTPUT_15	
00015	1 bit	R/W	OUTPUT_16	
00016	1 bit	R/W	OUTPUT_17	

Input register: (function code is 02)

Address number	data type	R/W	Mnemonic	Description
10001	1 bit	RO	INPUT_00	bit = 1, active state = 0, inactive state
10002	1 bit	RO	INPUT_01	
10003	1 bit	RO	INPUT_02	
10004	1 bit	RO	INPUT_03	
10005	1 bit	RO	INPUT_04	
10006	1 bit	RO	INPUT_05	
10007	1 bit	RO	INPUT_06	
10008	1 bit	RO	INPUT_07	
10009	1 bit	RO	INPUT_10	
10010	1 bit	RO	INPUT_11	
10011	1 bit	RO	INPUT_12	
10012	1 bit	RO	INPUT_13	
10013	1 bit	RO	INPUT_14	
10014	1 bit	RO	INPUT_15	
10015	1 bit	RO	INPUT_16	
10016	1 bit	RO	INPUT_17	