EMD-8216
Ethernet Digital I/O module

Software Manual (V1.0)

健昇科技股份有限公司

JS AUTOMATION CORP.

新北市汐止區中興路 100 號 6 樓

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

Xizhi Dist., New Taipei City, Taiwan

TEL：+886-2-2647-6936

FAX：+886-2-2647-6940

http://www.automation.com.tw

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

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

Correction record

Version	Record
1.0	EMD-8216.dll v1.0

Contents

1. How to install the software of EMD8216

Please register as user's club member to download the
"Step by step installation of EMD8216" document from http://automation.com.tw

1.1 Install the EMD driver

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

For a more detail descriptions, please refer "Step by step installation of EMD8216".

2. Where to find the file you need

Windows2000, XP and up

In Windows 2000,XP,Vista 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 2000/XP/Vista. 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	

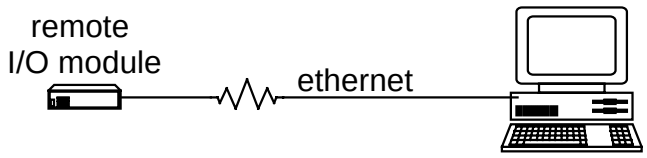
Table 1

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 ether net 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.

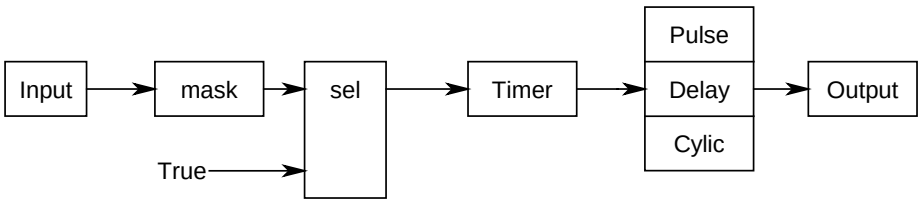


The module response or be commanded by the controller through Ethernet by UDP protocol. As a member on the Ethernet, it must have a distinguished IP and a predefined port. At factory, we set the default IP at 192.168.0.100 and the port at 6936 for the remote module.

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.

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

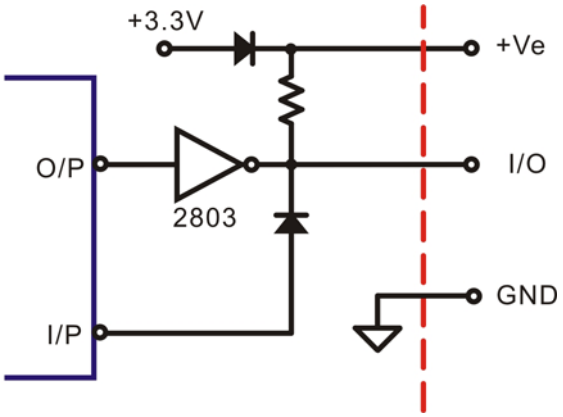
Output:

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 stae 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_00~07 1: IO_10~17
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_00~07 1: IO_10~17

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_00~07 1: IO_10~17
data	u8	b7 ~ b0, bitmap of output port polarity values, a value 1 means invert else normal. b7: polarity of IO_07 or IO_17 depend on port assignment … b0: polarity of IO_00 or IO_10 depend on port assignment

- **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_00~07 1: IO_10~17

Output:

Name	Type	Description
data	u8	b7 ~ b0, bitmap of I/O port polarity data b7: polarity of IO_07 or IO_17 depend on port assignment … b0: polarity of IO_00 or IO_10 depend on port assignment

- **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_00~07 1: IO_10~17
data	u8	b7 ~ b0, bitmap of output port values b7: IO_07 or IO_17 depend on port assignment … b0: IO_00 or IO_10 depend on port assignment

- **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_00~07 1: IO_10~17

Output:

Name	Type	Description
data	u8	b7 ~ b0, bitmap of I/O port data b7: IO_07 or IO_17 depend on port assignment … b0: IO_00 or IO_10 depend on port assignment

- **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_00~07 1: IO_10~17
point	u8	point number port = 0 Point 0x00~0x07 for IO_00~07 port = 1 Point 0x00~0x07 for IO_10~17
state	u8	1: input 0: output

- **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_00~07 1: IO_10~17
point	u8	point number port = 0 Point 0x00~0x07 for IO_00~07 port = 1 Point 0x00~0x07 for IO_10~17

Output:

Name	Type	Description
state	u8	Configure of designated point

- **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_00~07 1: IO_10~17
point	u8	point number port = 0 Point 0x00~0x07 for IO_00~07 port = 1 Point 0x00~0x07 for IO_10~17
state	u8	1 : invert 0 : normal.

- **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_00~07 1: IO_10~17
point	u8	point number port = 0 Point 0x00~0x07 for IO_00~07 port = 1 Point 0x00~0x07 for IO_10~17

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_00~07 1: IO_10~17
point	u8	point number port = 0 Point 0x00~0x07 for IO_00~07 port = 1 Point 0x00~0x07 for IO_10~17
state	u8	state of out port

- **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_00~07 1: IO_10~17
point	u8	point number port = 0 Point 0x00~0x07 for IO_00~07 port = 1 Point 0x00~0x07 for IO_10~17

Output:

Name	Type	Description
state	u8	state of designated point

7.3 Counter function

You can use the digital input as a low speed counter (no more than 200pps). 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 200Hz). 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_00~07 1: IO_10~17
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_00~07 1: IO_10~17

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_00~07 1: IO_10~17
Channel	u8	b7: 0: no function 1: clear IO_n7 counter … b0: 0: no function 1: clear IO_n0 counter

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 proceed 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 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
password[8]	u8	The new password to be set

- **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 can not always put the controller by guess. To ensure the controller will not harm to the system or human, people always put a WDT to monitor the controller, if the controller fail to reset it, WDT will latch the system to prevent further harm. EDM8216 also provide the hardware WDT function, you can enable or disable as your application requirement.

Use

EMD8216_WDT_set( ) to set up the WDT timer and the output state if the system 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 system failure.

- **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	read the output default state
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

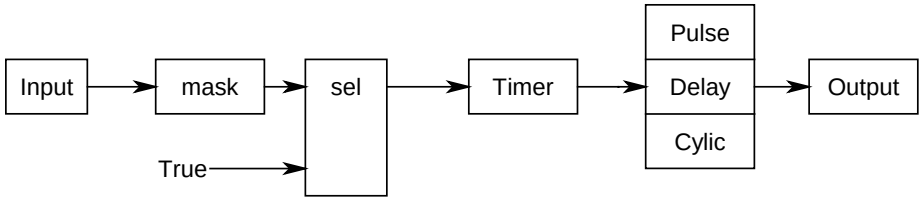
Standalone mode is the extension of EMD8216 module; it can work as I/O controller without the ethernet existing.

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

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=Low,
		// 0x1=High,
		// 0x2=Change

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

Note:

1. The StandaloneData isany 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. 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, StandaloneData data[32], u8 *enable, u8 *power_on_enable)**

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

Parameters:

Input:

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

Output:

Name	Type	Description

data	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=Low,
		// 0x1=High,
		// 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.

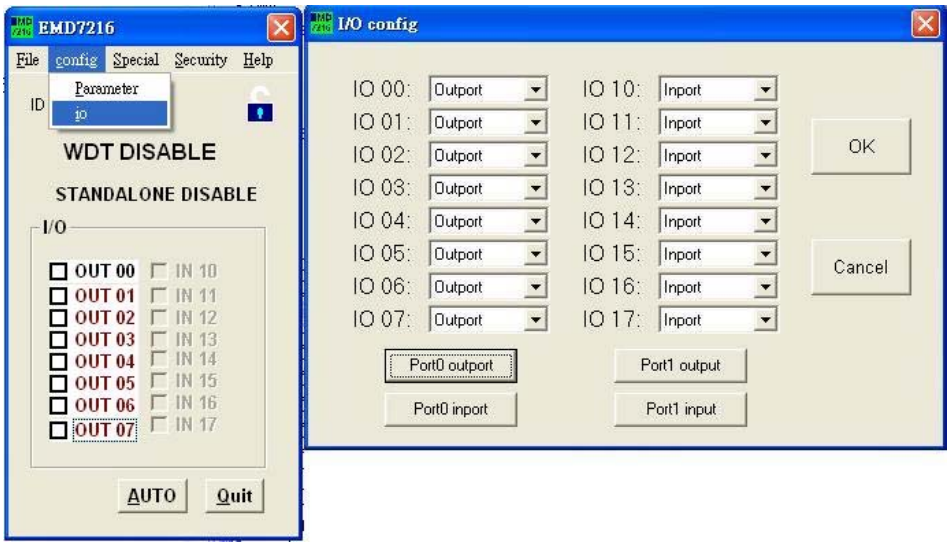
8. Standalone mode user configuration utility

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.

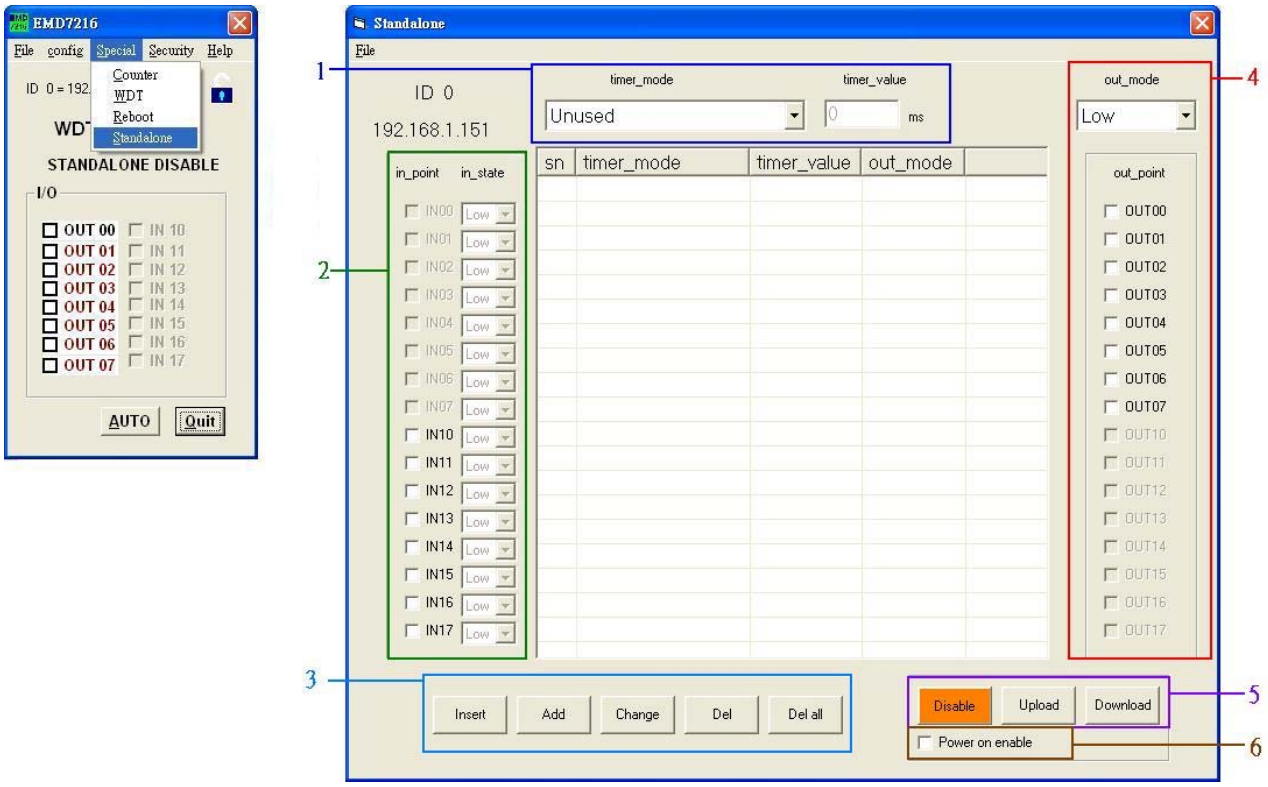
8.1 Overview of user configuration utility

-- 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.

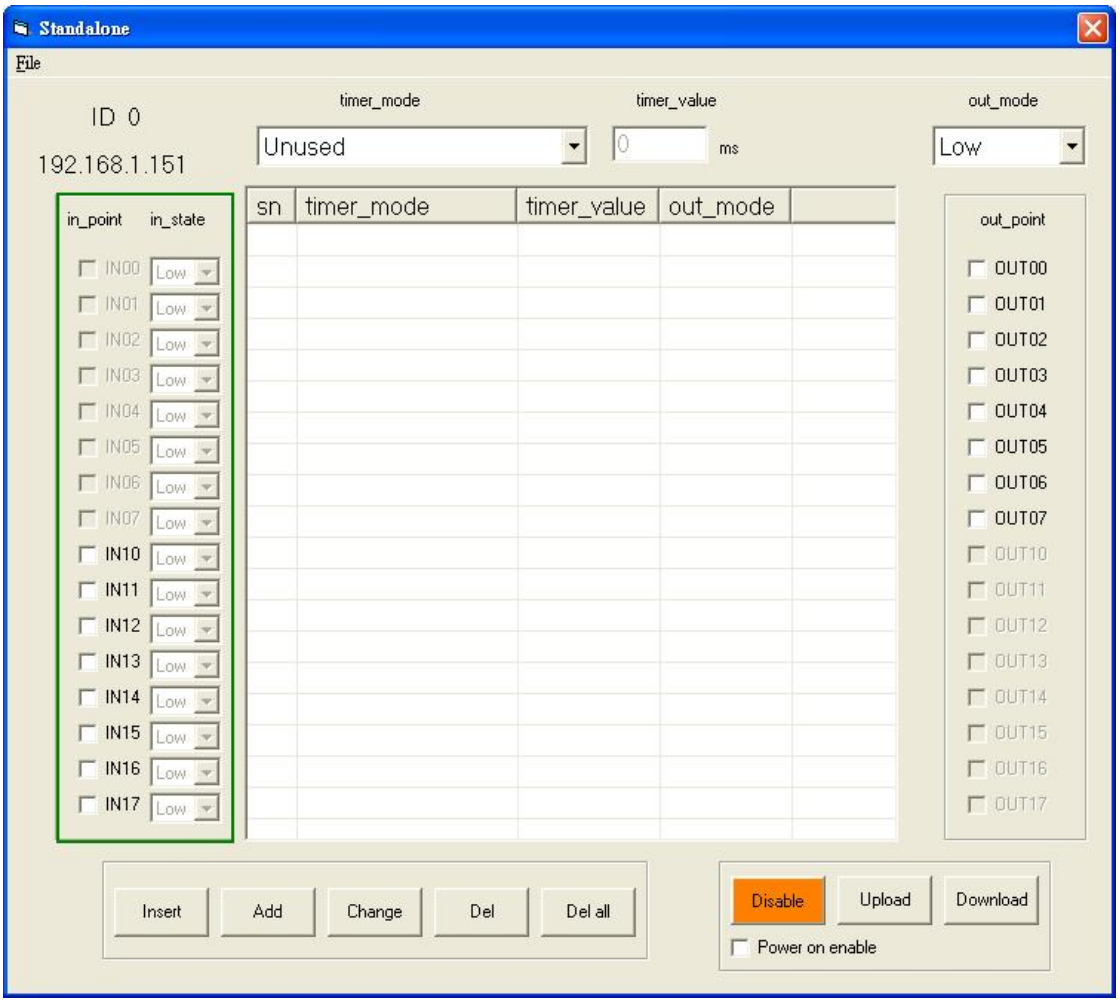
Block5: standalone mode command upload /download, start/stop.

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

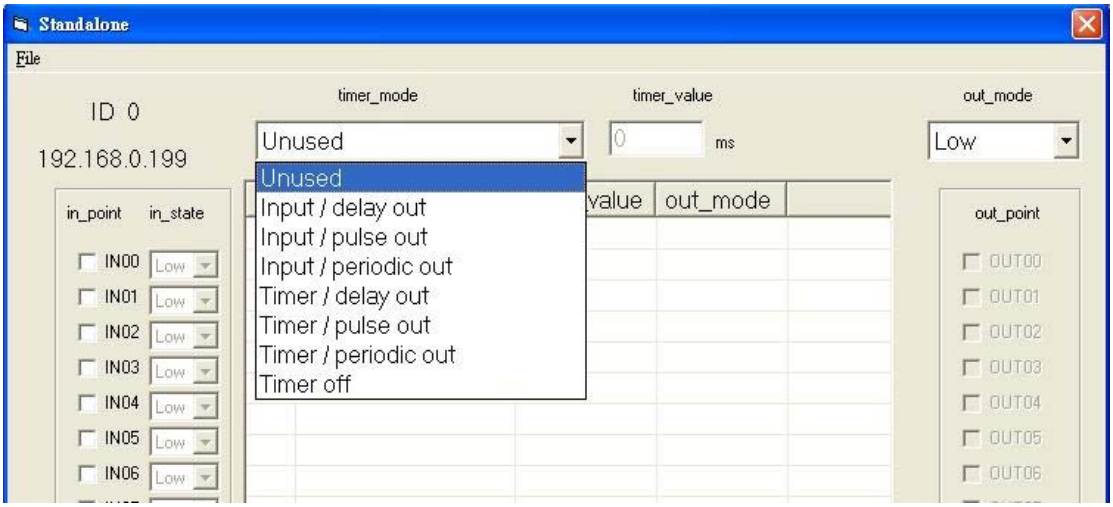


From the above diagram, check the input point and its state the current command will take care.

The above diagram shown that if you want the input monitor IN01 high and IN02 low 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 200Hz, response faster than 200Hz maybe ignore as noise by the EMD8216 module.

-- timer configuration

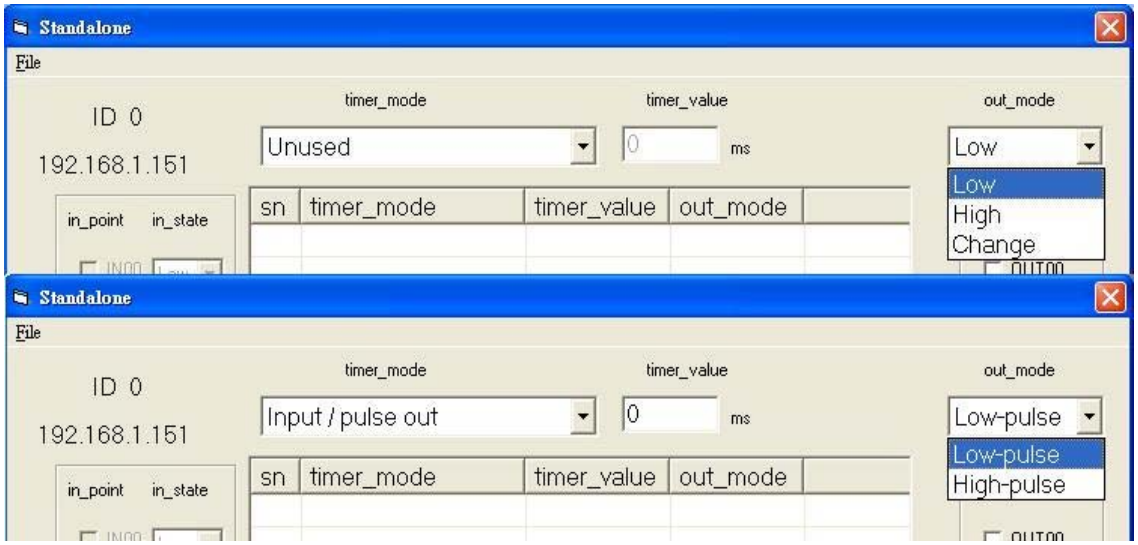


If the inputs meet the condition you configured, it will trigger the timer to operate. The time provides several kinds of working mode:

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

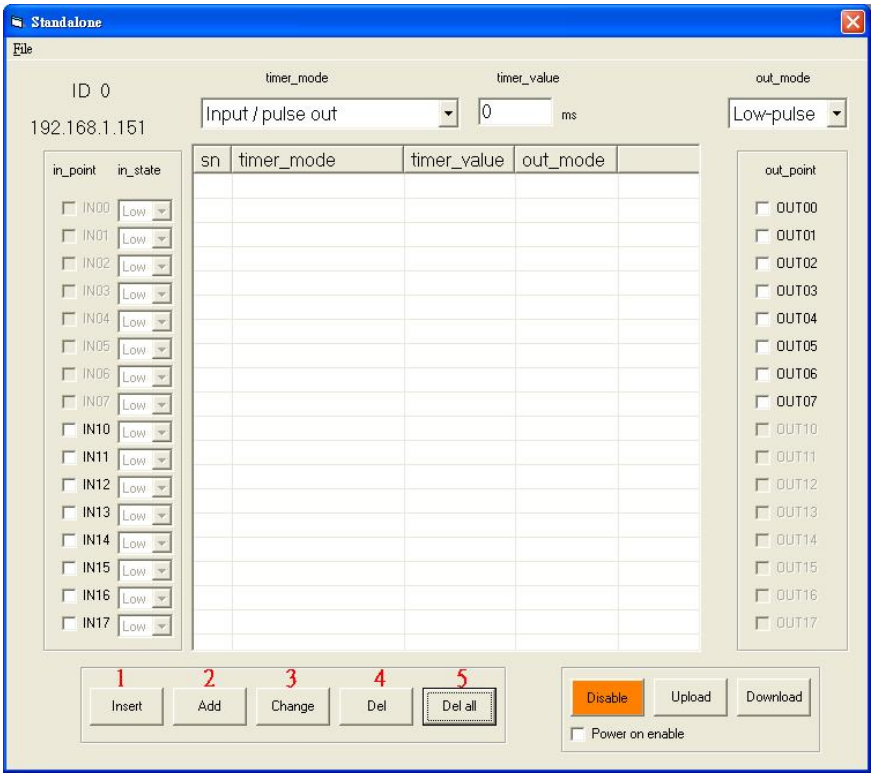


The timer depending on its working mode controls the output. The output can be configured as active low, active high or toggle.

Timer mode	output mode	explanation
Unused	Low	output active low
	High	output active high
	Change	output toggles
Input action delay	Low level	output active low
	High level	output active high
	Change	output toggles
Input action pulse	Low-pulse	output active low pulse (‾|__|‾)
	High-pulse	output active high pulse (__|‾|__)
Input action periodic	Change	output toggles
Timer action delay	Low	output active low
	High level	output active high
	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, 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

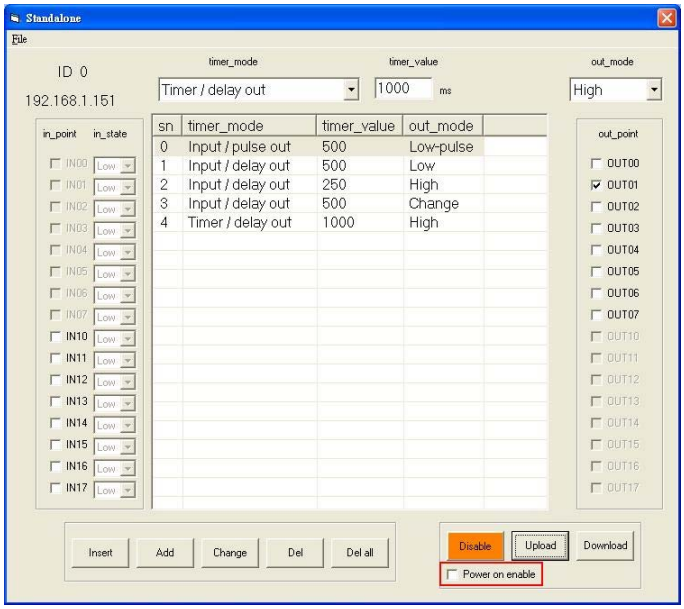
3: Change: modify the existing command line

4: Del: delete the highlighted command line

5: Del al: clear all the commands

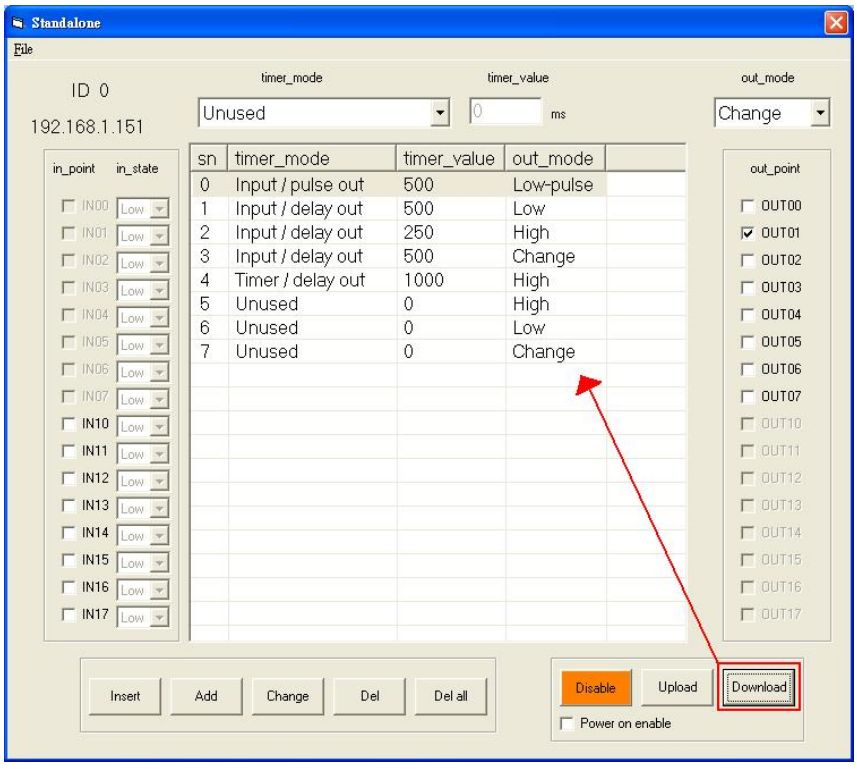
8.4 Upload program

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

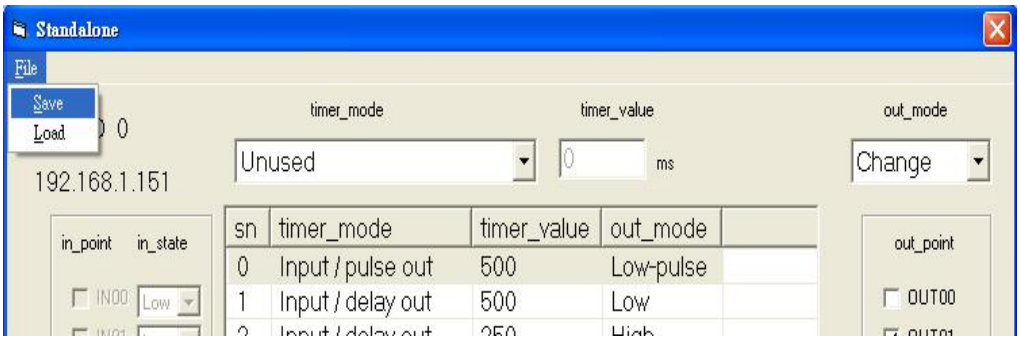
If you have connected with EMD8216 module via Ethernet, you can download the stored program from the module.



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.



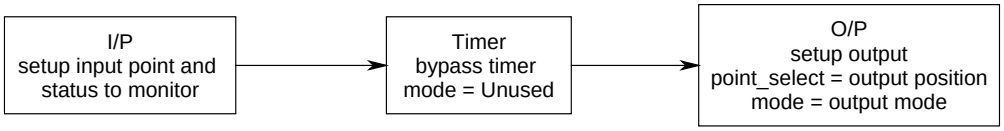
8.7 Enable/Disable standalone function

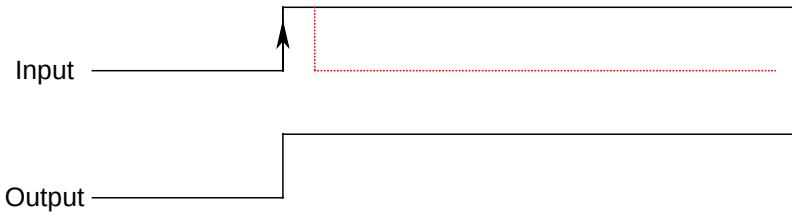
Standalone mode can enable or disable by the button as following shown.

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

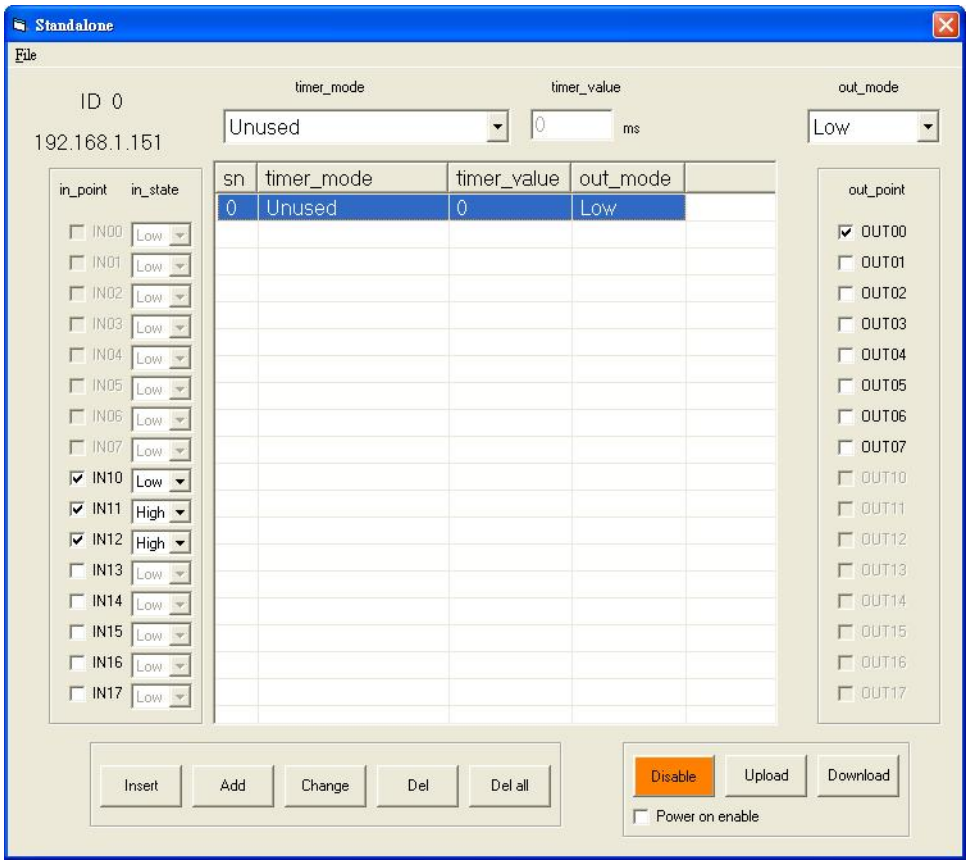
9. Standalone mode application examples

9.1 Monitoring input if condition meets, trigger output

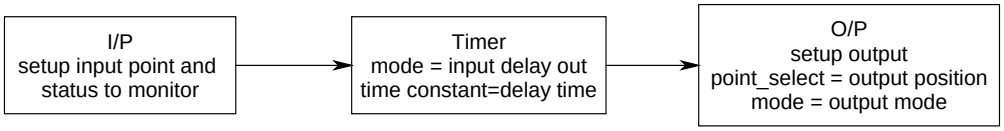


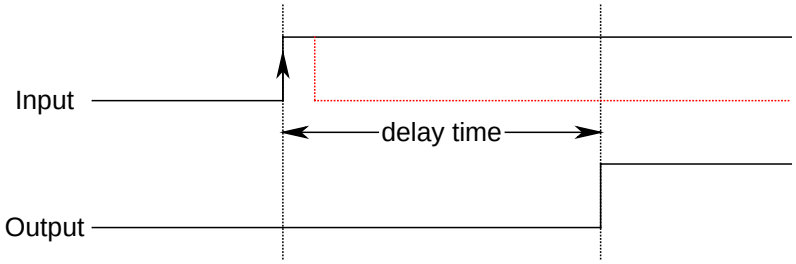


Say, you want to watch IN10 low, IN11 and IN12 high to trigger output OUT00 to low. Program as the following shown.

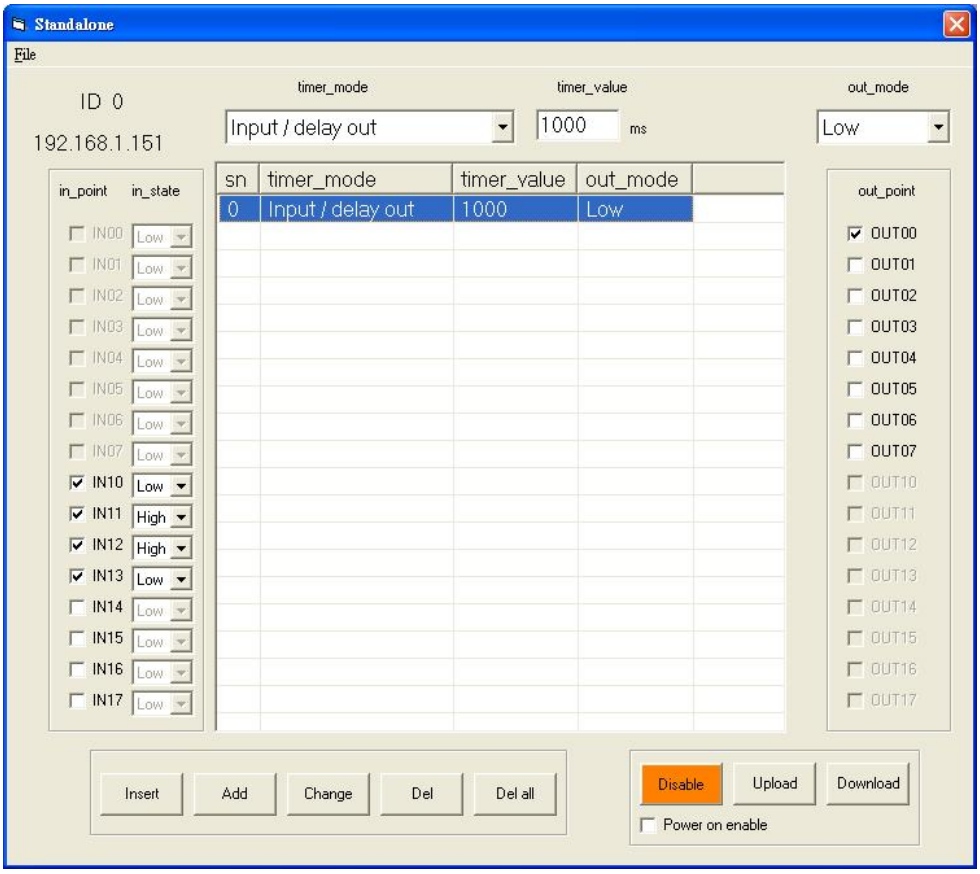


9.2 Monitoring the input if condition meets, delay to trigger output

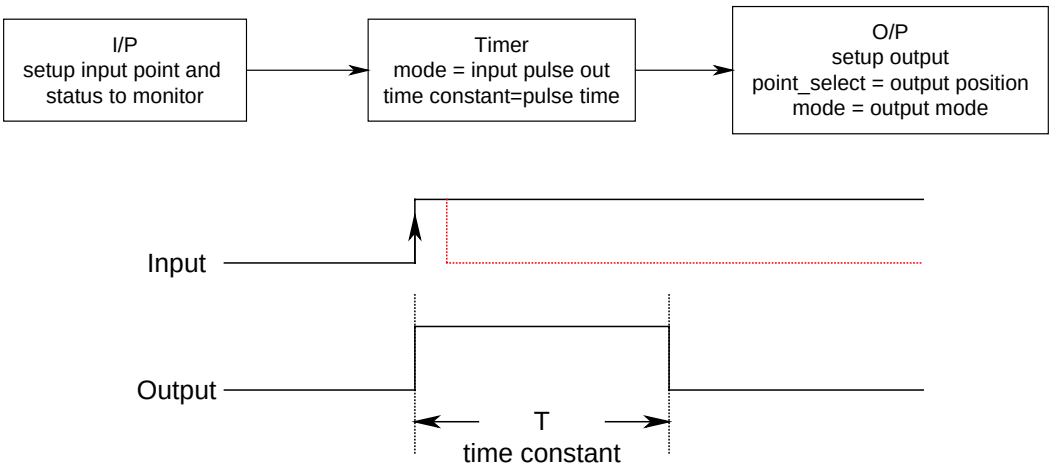




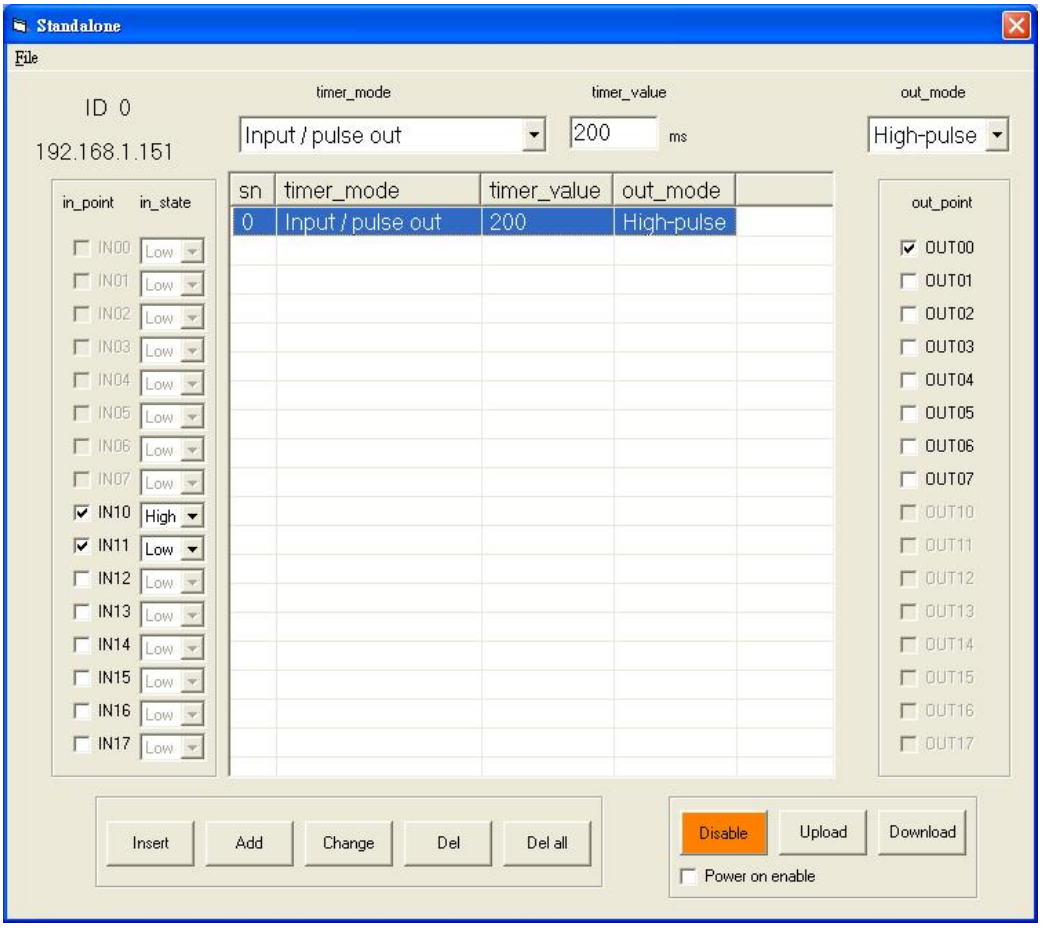
Say, you want to watch IN10 and IN13 are low and IN11 and IN12 are high to trigger output OUT00 to low. Program as the following shown.



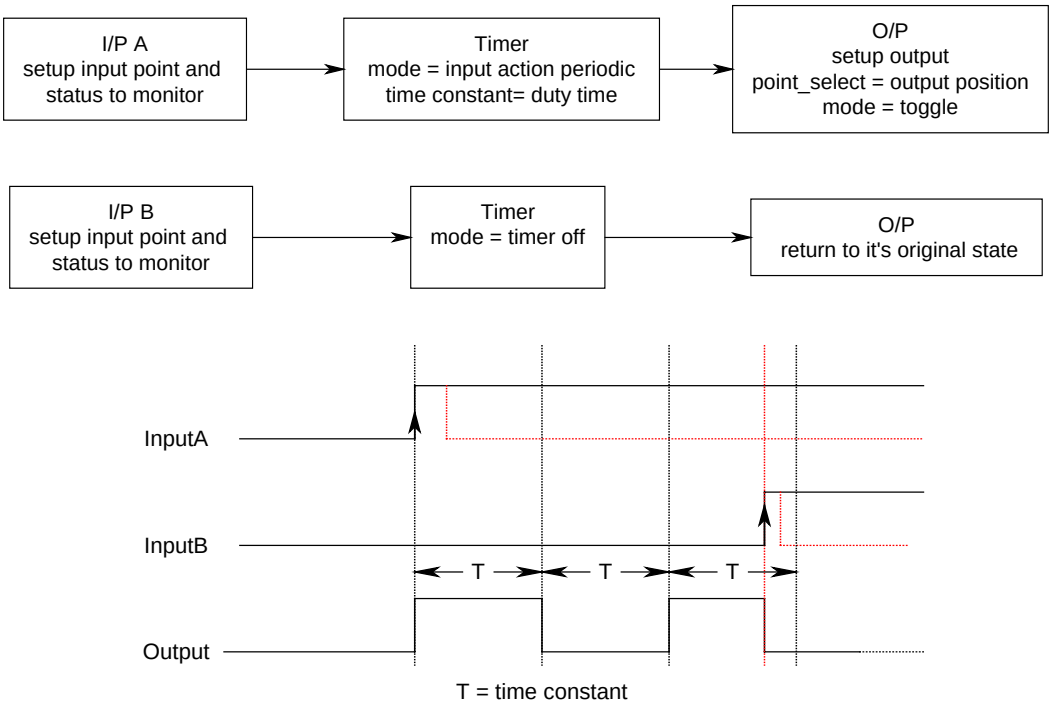
9.3 Monitoring the input if condition meets, output pulse



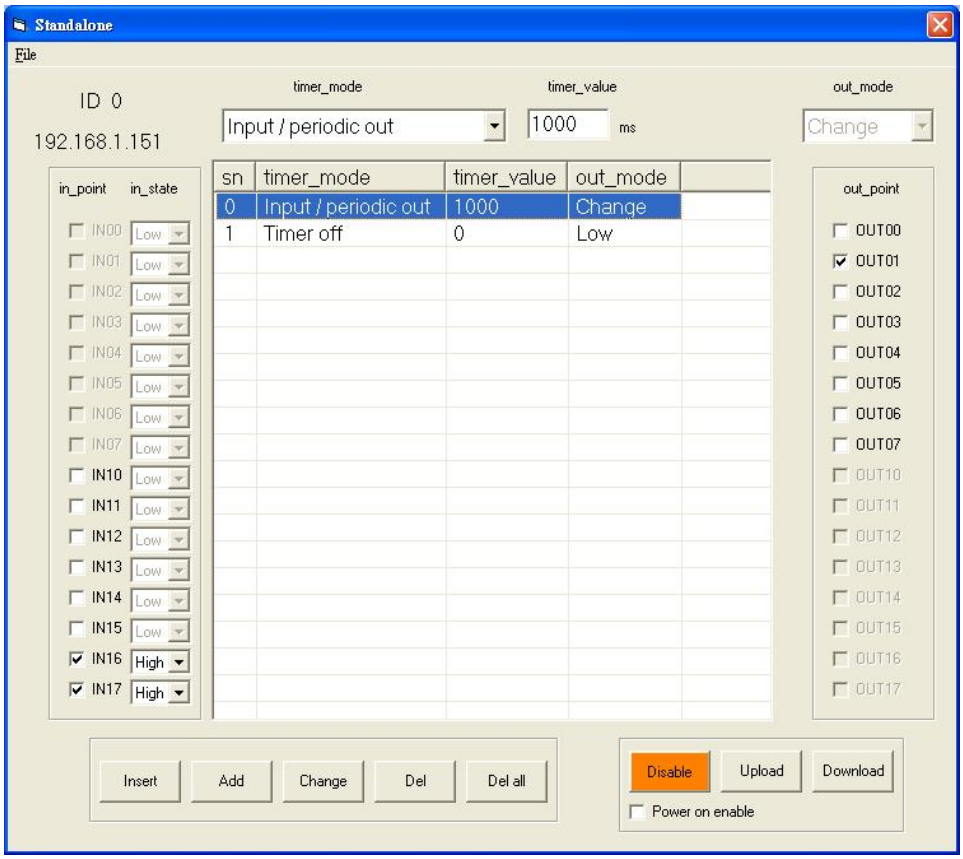
Say, you want to watch IN10 is high low and IN11 is low to trigger output OUT00 to pulse high. Program as the following shown.



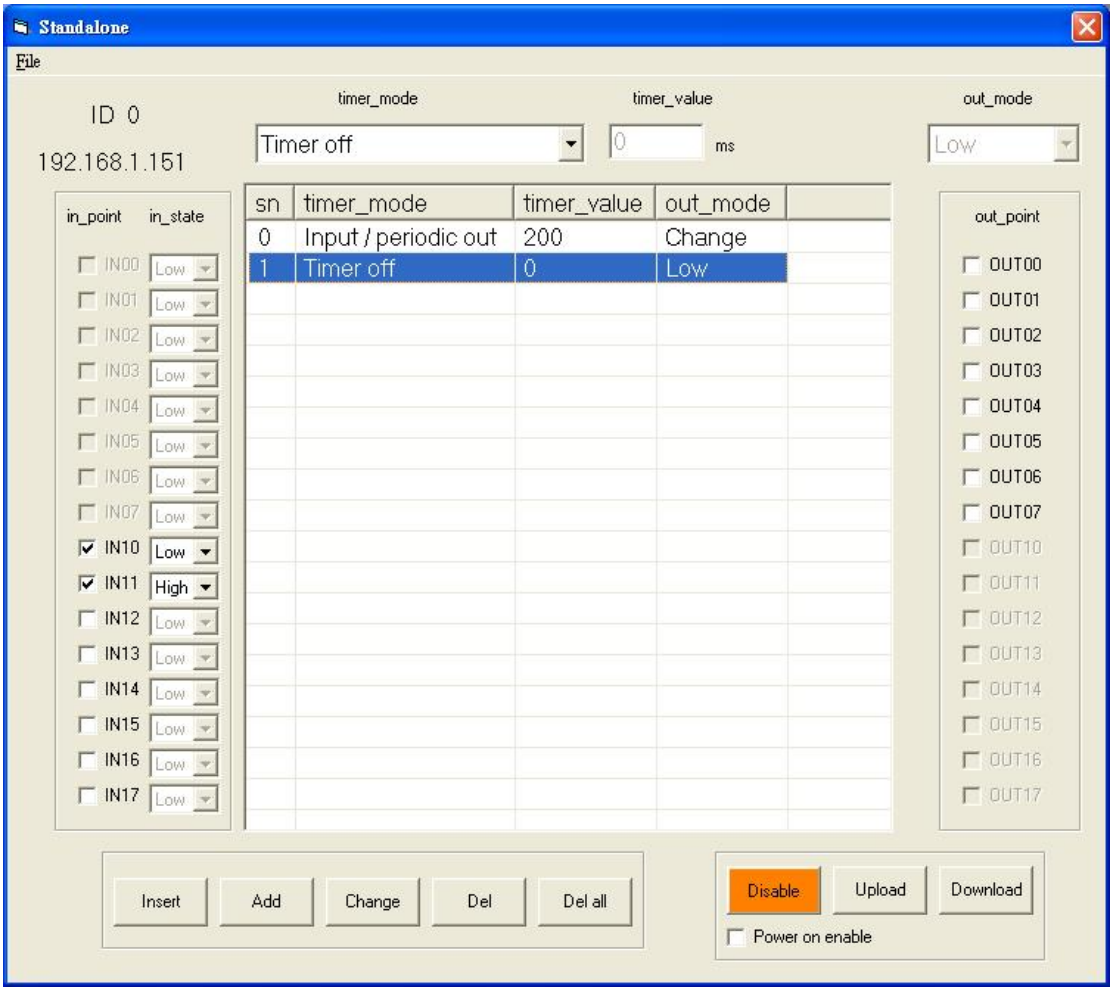
9.4 Monitoring the input if condition meets, output periodically and stop by some special input condition



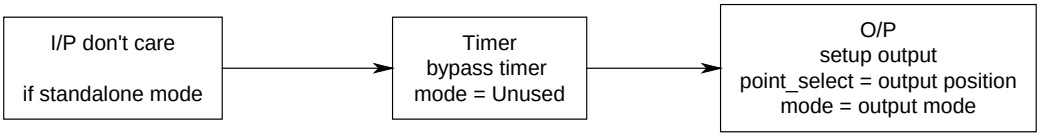
Say, you want to watch IN10 is low and IN11 is high to trigger output OUT00 to toggle. Program as the following shown.

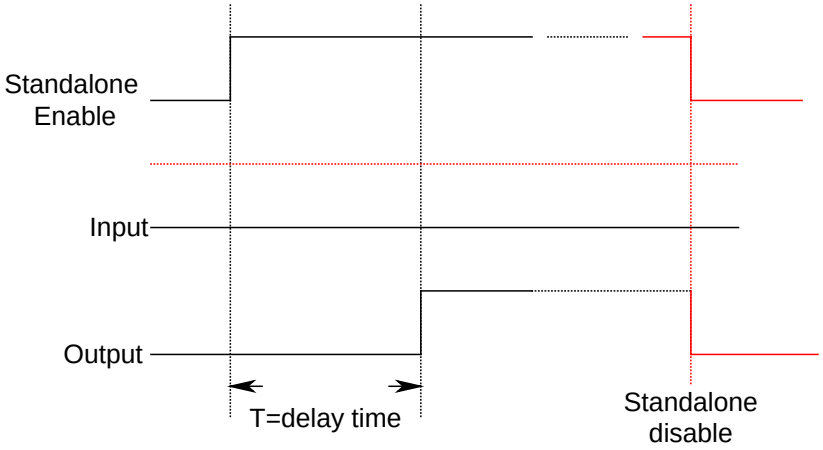


Then, if we want IN10 is low and IN11 is high to trigger to stop the timer. Program as the following shown.

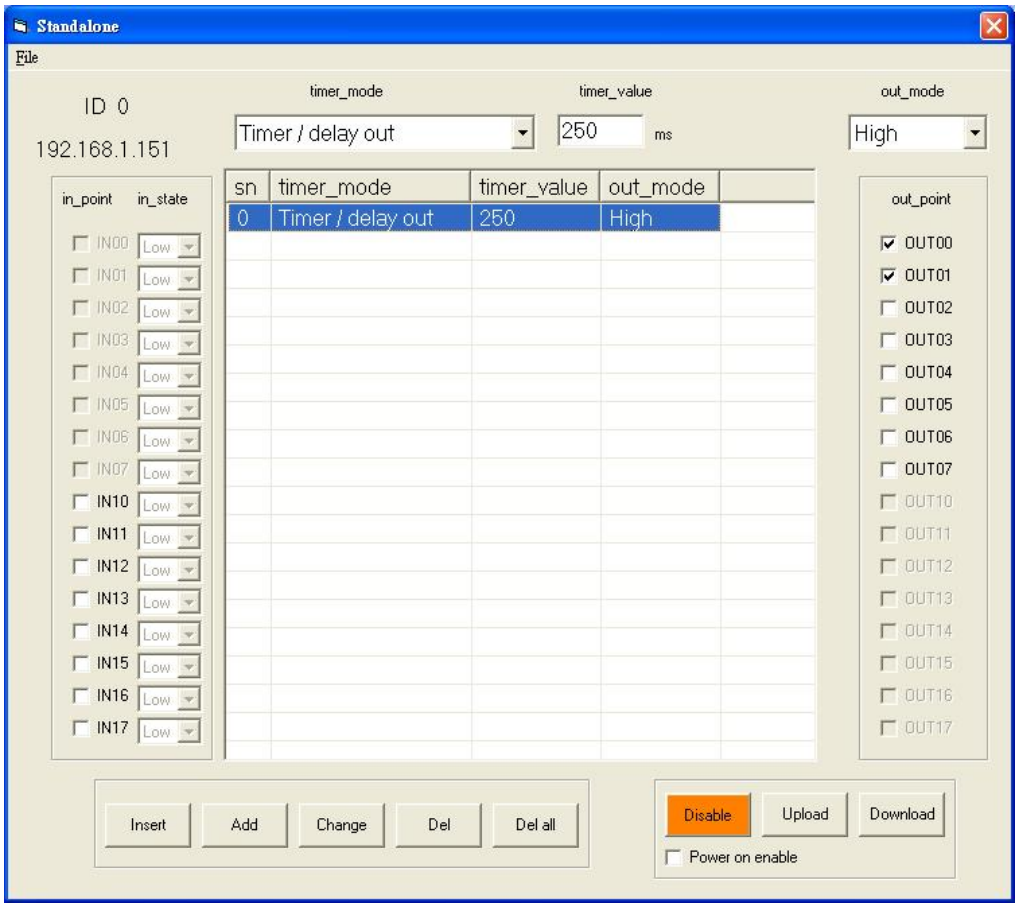


9.5 Don’t care the input if standalone enabled, trigger output

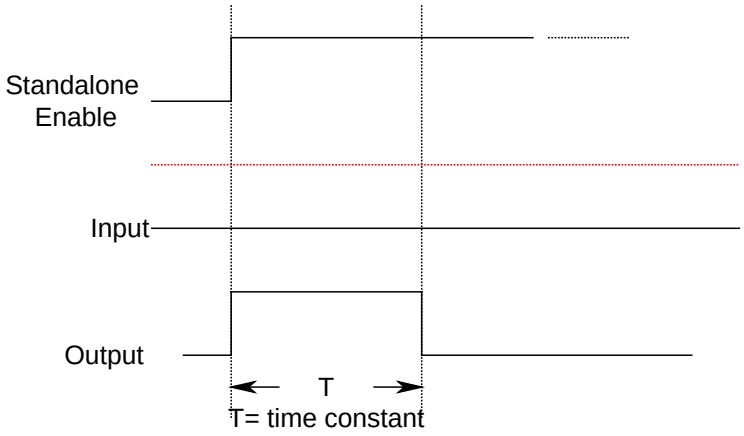




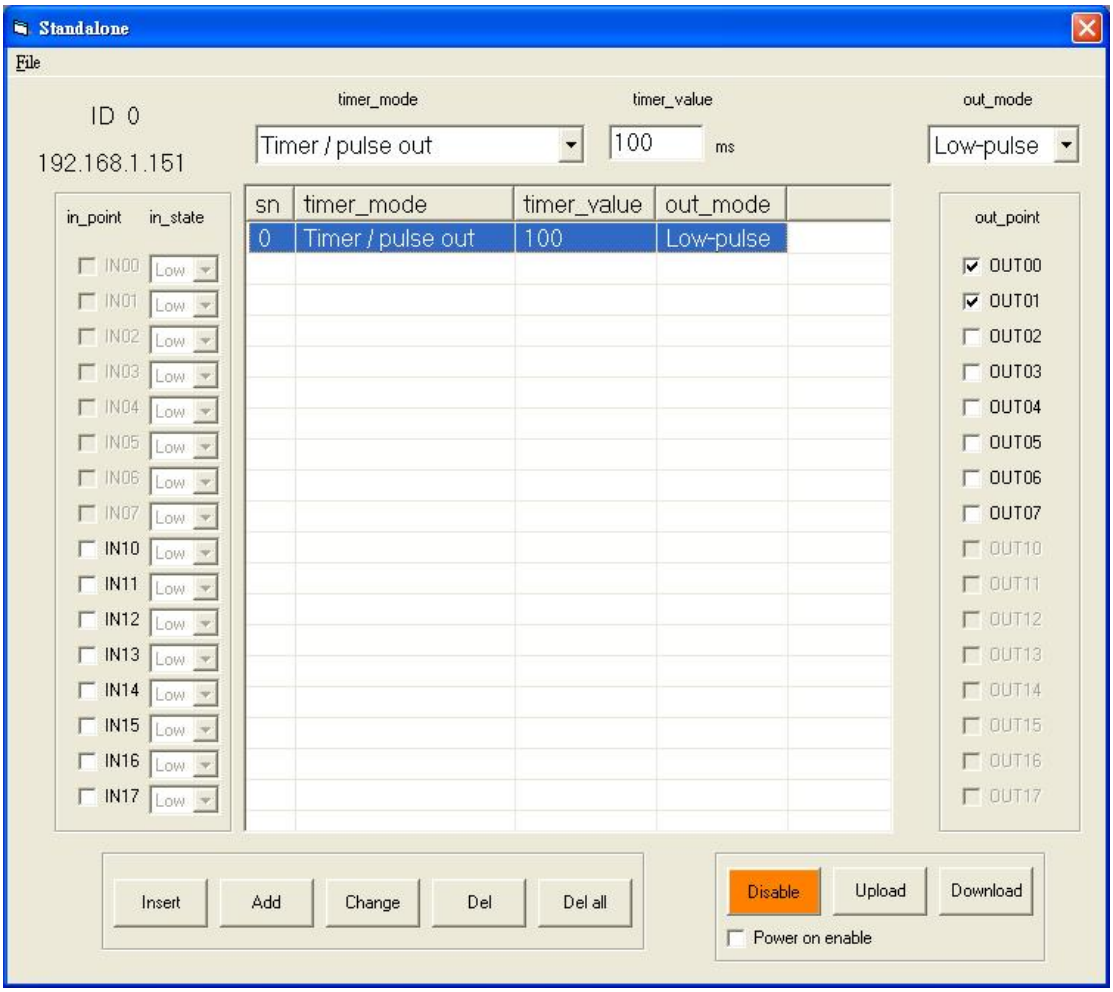
Say, don’t care any input just output OUT00 OUT01 high as the standalone mode enabled. Program as the following shown.



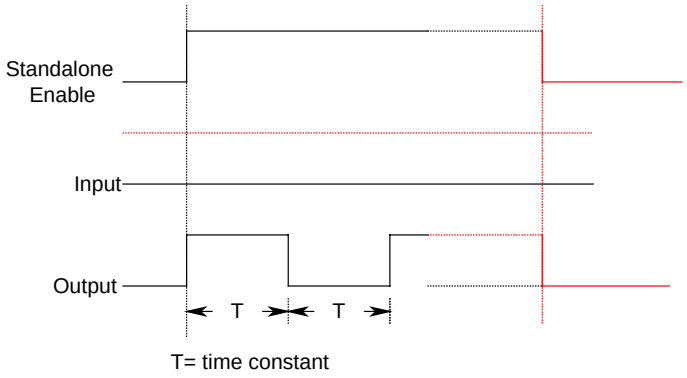
9.6 Don’t care the input if standalone enabled, trigger pulse



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

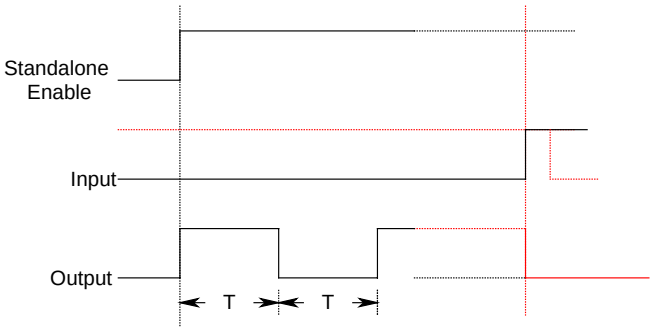


9.7 Don’t care the input if standalone enabled, output periodically



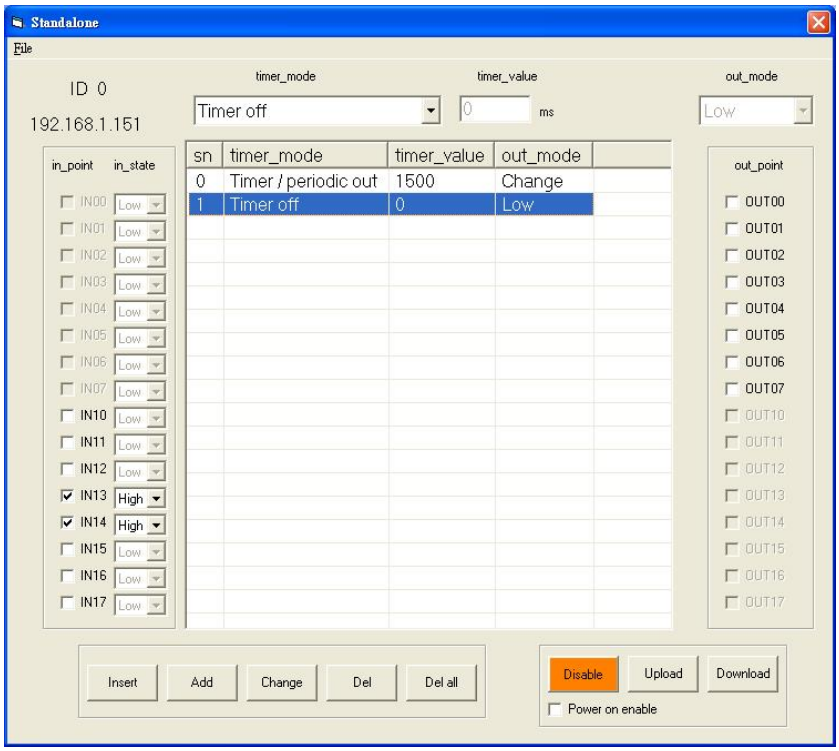
T= time constant

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 high. The program is shown as followings.



10. 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

11. EMD8216 Error codes summary

11.1 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	Cardid already used
300	ID_ERROR	Cardid is not acceptable
301	PORT_ERROR	Port parameter unacceptable or unreachable
302	IN_POINT_ERROR	Input point unreachable
303	OUT_POINT_ERROR	Output point unreachable
305	PARAMETERS_ERROR	Parameter error
306	CHANGE_SOCKET_ERROR	Can not change socket
307	UNLOCK_SECURITY_ERROR	Fail to unlock security
308	PASSWORD_ERROR	Password mismatched
309	REBOOT_ERROR	Can not reboot
310	TIME_OUT_ERROR	Too long to response
311	CREAT_SOCKET_ERROR	Socket can not create
312	CHANGE_IP_ERROR	Change IP error
313	MASK_ERROR	Set mask error
314	COUNTER_ENABLE_ERROR	Can not enable counter
315	COUNTER_DISABLE_ERROR	Can not 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	Can not read firmware version
321	STANDALONE_ENABLE_ERROR	Can not enable standalone
322	STANDALONE_DISABLE_ERROR	Can not disable standalone
323	STANDALONE_CONFIG_ERROR	Can not setup / read standalone configuration