EMC8432/EMC8485
Ethernet to serial converter 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	EMC84xx.dll v1.0

Contents

1. How to install the software of EMC84xx

1.1 Install the EMC driver

The Ethernet module can not found by OS as PCI cards. You can just install the driver without the module installed. Execute the file ..\install\EMC84xx_Install.exe to install the driver, Api and demo program automatically.

For a more detail descriptions, please refer “Step by step installation of EMC84xx”.

2. Where to find the file you need

Windows2000 and up

In Windows 2000 and up, the demo program can be setup by EMC84xx_Install.exe.

If you use the default setting, a new directory .. \JS Automation\EMC84xx will generate to put the associate files.

.. / JS Automation /EMC84xx/API (header files and VB,VC lib files)

.. / JS Automation /EMC84xx /Driver (copy of driver code)

.. / JS Automation /EMC84xx /exe (demo program and source code)

The dll is located at ..\system.

3. About the EMC84xx software

EMC84xx software includes a set of dynamic link library (DLL) based on socket that you can utilize to control the interface functions.

Your EMC84xx 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 EMC84xx functions within Windows' operation system environment.

If you only want to use as a general COM port, the virtual COM driver only will do.

3.1 What you need to get started

To set up and use your EMC84xx software, you need the following:

- EMC84xx software
- EMC84xx hardware

3.2 Software programming choices

You have several options to choose from when you are programming EMC84xx 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 EMC84xx software.

4. Language support

The EMC84xx software library is a DLL used with Windows 2000 and up. You can use these DLL with any Windows integrating development environment that can call Windows DLLs.

4.1 Building applications with the EMC84xx software library

The EMC84xx function reference section contains general information about building EMC84xx applications, describes the nature of the EMC84xx functions used in building EMC84xx 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.

EMC84xx Windows Libraries

The EMC84xx for Windows function library is a DLL called **EMC84xx.dll**. Since a DLL is used, EMC84xx functions are not linked into the executable files of applications. Only the information about the EMC84xx functions in the EMC84xx 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 EMC84xx functions in EMC84xx.dll.

Header Files and Import Libraries for Different Development Environments		
Development Environment	Header File	Import Library
Microsoft C/C++	EMC84xx.h	EMC84xxVC.lib
Borland C/C++	EMC84xx.h	EMC84xxBC.lib
Microsoft Visual Basic	EMC84xx.bas	

Table 1

5. Function format and language difference

5.1 Function format

Every EMC84xx 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 EMC84xx function is the parameter **CardID** which is set by ***EMC84xx_initial*** . You can utilize multiple devices with different card ID within one application; to do so, simply pass the appropriate **CardID** to each function.

5.2 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

5.3 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 EMC84xx API. Read the following sections that apply to your programming language.

Note: Be sure to include the declaration functions of EMC84xx prototypes by including the appropriate EMC84xx header file in your source code. Refer to Chapter 4. EMC84xx Language Support for the header file appropriate to your compiler.

5.3.1 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 = EMC84xx_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, port=0 ;    //assume CardID is 0 and port also 0
u8 data,
u32 Status;
Status = EMC84xx_port_read ( CardID, port, &data);
```

5.3.2 Visual basic

The file EMC84xx.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 EMC84xx.bas, do not use the numerical values.

In Visual Basic, you can add the entire EMC84xx.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 EMC84xx.bas file for your project in Visual Basic 4.0, go to the **File** menu and select the **Add File...** **option**. Select EMC84xx.bas, which is browsed in the EMC84xx \ api directory. Then, select **Open** to add the file to the project.

To add the EMC84xx.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** EMC84xx.bas, which is in the EMC84xx \api directory. Then, select **Open** to add the file to the project.

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

5.3.3 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 EMC84xxbc.lib EMC84xx.dll

Then add the **EMC84xxBC.lib** to your project and add

#include “EMC84xx.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 = EMC84xx_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, port=0 ;    //assume CardID is 0 and port also 0
u8 data,
u32 Status;
Status = EMC84xx_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.

6. Software overview and dll function

6.1 Initialization and close

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

EMC84xx_initial( ) will do.

Once you want to close your application, call

EMC84xx_close( ) to release all the resource.

To check the firmware version,

EMC84xx_firmware_version_read( ) will do.

- **EMC84xx_initial**

Format : u32 status =EMC84xx_initial (u32 CardID,u8 IP_Address[4],u16 Host_Port,u16 Remote_port,u16 TimeOut_ms,u8 *CardType)

Purpose: To map IP and PORT of an existing EMC84xx to a specified CardID number**.**

Parameters:

Input:

Name	Type	Description
CardID	u32	0 ~ 1999 Assign CardID to the EMC84xx of a corresponding IP address.
IP_Address[4]	u8	4 words of IP address 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 Default:192.168.0.100
Host_Port	u16	Assign a communicate port of host PC Default: 25122
Remote_port	u16	Assign a communicate port of EMC84xx Default: 6936
TimeOut	u16	Assign the max delay time of EMC84xx response message,1000~10000 ms.

Output:

Name	Type	Description
CardType	u8	Get the Card Type of EMC84xx 1: EMC8485 2: EMC8432

- **EMC84xx_close**

Format : **u32 status =EMC84xx_close (u32 CardID)**

Purpose: Release the EMC84xx resource when closing the Windows applications.

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMC84xx_initial function

- **EMC84xx_firmware_version_read**

Format : **u32 status = EMC84xx_firmware_version_read(u32 CardID, u8 Version[2])**

Purpose: Read the firmware version.

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMC84xx_initial

Output:

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

6.2 Configuration function

To change the socket port by

EMC84xx_socket_port_change( ) and change IP by

EMC84xx_IP_change( )

Sometimes you need to reset the system (hot reset), you can commend by

EMC84xx_reboot( )

- **EMC84xx_socket_port_change**

Format : u32 status = EMC84xx_socket_port_change (u32 CardID,u16 Remote_port);

Purpose: To change the communicate port number of EMC84xx.

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

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMC84xx_initial function
Remote_port	u16	The new port number to be set Default port is: 6936

- **EMC84xx_IP_change**

Format : u32 status = EMC84xx_IP_change (u32 CardID,u8 IP[4]);

Purpose: To change the communicate IP of EMC84xx.

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

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMC84xx_initial function
IP[4]	u8	The new IP to be set Default IP is: 192.168.0.100 IP_Address[0]=192 IP_Address[1]=168 IP_Address[2]=0 IP_Address[3]=100

- **EMC84xx_reboot**

Format : **u32 status = EMC84xx_reboot(u32 CardID);**

Purpose: To reboot EMC84xx (about 10s).

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMC84xx_initial function

6.3 Software key function

To prevent un-authorized person to change the settings and outputs, software key is an essential protection. If you want to command to change settings or output, you must unlock first by

EMC84xx_security_unlock( ) and read back the status of security by

EMC84xx_security_status_read( )

If you want to change password, use

EMC84xx_password_change( ) will do.

If you forget the password and you want to reset password to factory default value remotely,

EMC84xx_password_set_default( ) [*1] will do.

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

- **EMC84xx_security_unlock**

Format : **u32 status = EMC84xx_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 EMC84xx_initial function
password[8]	u8	The password previous set Default: password[8] = {'1','2','3','4','5','6','7','8'};

- **EMC84xx_security_status_read**

Format : **u32 status = EMC84xx_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 EMC84xx_initial function

Output:

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

- **EMC84xx_password_change**

Format : **u32 status = EMC84xx_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 EMC84xx_initial function
Oldpassword [8]	u8	The previous password
password[8]	u8	The new password to be set

- **EMC84xx_password_set_default**

Format : **u32 status = EMC84xx_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 EMC84xx_initial function default :password[8] = {'1','2','3','4','5','6','7','8'};

6.4 Digital I/O

The Ethernet to serial converter module provides extra 8-bit Digital I/O (IO_0 ~ IO_7) for compact integration of various applications need to control on/off or detect external signals.

First of all, you must setup each pin as input or output.

EMC84xx_port_config_set( ) will do.

EMC84xx_port_config_read( ) to read back for verification.

Digital 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'.

EMC84xx_port_polarity_set( )

EMC84xx_port_polarity_read( )

To read write the port by:

EMC84xx_port_set( ) to set the output data;

EMC84xx_port_read( ) to read the input status.

The point operation is convenience function if you just want to operate bit data, use

EMC84xx_point_config_set( ) to setup bit configuration and read back by:

EMC84xx_point_config_read( )

Also bit polarity can be set by:

EMC84xx_point_polarity_set( )

EMC84xx_point_polarity_read( )

To write the point data by:

EMC84xx_point_set( ) and read back by:

EMC84xx_point_read( )

- **EMC84xx_port_config_set**

Format : **u32 status = EMC84xx_port_config_set(u32 CardID, u8 port, u8 config);**

Purpose: To setup the IO configuration.

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMC84xx_initial function
port	u8	unused
IO_config	u8	Configure the I/O as input or output bit0: 0: IO_0 as output 1: IO_0 as input … bit7: 0: IO_7 as output 1: IO_7 as input

- **EMC84xx_port_config_read**

Format : **u32 status = EMC84xx_port_config_read(u32 CardID, u8 port, u8 *IO_config);**

Purpose: To read the IO configuration.

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMC84xx_initial function
port	u8	unused

Output:

Name	Type	Description
IO_config	u8	bit0: 0: IO_0 as output 1: IO_0 as input … bit7: 0: IO_7 as output 1: IO_7 as input

- **EMC84xx_port_polarity_set**

Format : **u32 status = EMC84xx_port_polarity_set(u32 CardID,u8 port ,u8 polarity);**

Purpose: Sets the I/O polarity of IO_0 ~ IO_7

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMC84xx_initial function
port	u8	unused
polarity	u8	polarity data: b7~b0 b0: =0, IO_0 normal polarity =1, IO_0 invert polarity … b7: =0, IO_7 normal polarity =1, IO_7 invert polarity

- **EMC84xx_port_polarity_read**

Format : **u32 status = EMC84xx_port_polarity_read(u32 CardID,u8 port, u8 * polarity);**

Purpose: Read the I/O polarity of the IO_0~IO_7.

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMC84xx_initial function
port	u8	unused

Output:

Name	Type	Description
polarity	u8	polarity data: b7~b0 b0: =0, IO_0 normal polarity =1, IO_0 invert polarity … b7: =0, IO_7 normal polarity =1, IO_7 invert polarity

- **EMC84xx_port_set**

Format : **u32 status = EMC84xx_port_set(u32 CardID,u8 port, u8 data);**

Purpose: To set the output value.

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMC84xx_initial function
port	u8	unused
data	u8	Set the IO output value. bit0: IO_0 value … bit7: IO_7 value

- **EMC84xx_port_read**

Format : **u32 status = EMC84xx_port_read(u32 CardID, u8 port, u8 *data);**

Purpose: To read all the IO port value.

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMC84xx_initial function
port	u8	unused

Output:

Name	Type	Description
data	u8	Read the IO port value. bit0: IO_0 value … bit7: IO_7 value

- **EMC84xx_point_config_set**

Format : **u32 status = EMC84xx_point_config_set(u32 CardID, u8 port, u8 point, u8 state);**

Purpose: To setup the IO configuration.

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMC84xx_initial function
port	u8	unused
point	u8	point number 0: IO_0 1: IO_1 … 7: IO_7
state	u8	Configure the IO as input or output 0: IO as output 1: IO as input

- **EMC84xx_point_config_read**

Format : **u32 status = EMC84xx_point_config_read(u32 CardID, u8 port, u8 point, u8 *state);**

Purpose: To read the IO configuration.

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMC84xx_initial function
port	u8	unused
point	u8	point number 0: IO_0 1: IO_1 … 7: IO_7

Output:

Name	Type	Description
state	u8	Configure the IO as input or output 0: IO as output 1: IO as input

- **EMC84xx_point_polarity_set**

Format : **u32 status = EMC84xx_point_polarity_set(u32 CardID,u8 port,u8 point, u8 polarity);**

Purpose: Sets the I/O polarity of point IO_0~IO_7

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMC84xx_initial function
port	u8	unused
point	u8	point number 0: IO_0 1: IO_1 … 7: IO_7
polarity	u8	Polarity state: 0, normal polarity 1, invert polarity

- **EMC84xx_point_polarity_read**

Format : **u32 status = EMC84xx_point_polarity_read(u32 CardID, u8 port, u8 point, u8 *polarity);**

Purpose: Read the I/O polarity of point IO_0~IO_7

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMC84xx_initial function
port	u8	unused
point	u8	point number 0: IO_0 1: IO_1 … 7: IO_7

Output:

Name	Type	Description
polarity	u8	Polarity state: 0, normal polarity 1, invert polarity

- **EMC84xx_point_set**

Format : **u32 status = EMC84xx_point_set(u32 CardID,u8 port,u8 point, u8 state);**

Purpose: To set the output value.

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMC84xx_initial function
port	u8	unused
point	u8	point number 0: IO_0 1: IO_1 … 7: IO_7
state	u8	state of designated output point

- **EMC84xx_point_read**

Format : **u32 status = EMC84xx_point_read(u32 CardID, u8 port, u8 point, u8 *state);**

Purpose: To read the IO point value.

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMC84xx_initial function
port	u8	unused
point	u8	point number 0: IO_0 1: IO_1 … 7: IO_7

Output:

Name	Type	Description
state	u8	state of designated point

6.5 Counter function

Inputs (IO_0 ~ IO_7) can be used as low frequency counter (less than 200 pulses per second), you can mask off the counter function on unwanted inputs by:

EMC84xx_counter_mask_set( ), then enable or disable the counter function:

EMC84xx_counter_enable( ) to enable counter function;

EMC84xx_counter_disable( ) to disable counter function.

The counter can be read or clear by using:

EMC84xx_counter_read( ) to read counter on the fly;

EMC84xx_counter_clear( ) to clear counter data.

- **EMC84xx_counter_mask_set**

Format : u32 status = EMC84xx_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 EMC84xx_initial function
port	u8	unused
Channel	u8	b0: 0: IO_0 counter disable 1: IO_0 counter enable … b7: 0: IO_7 counter disable 1: IO_7 counter enable

- **EMC84xx_counter_enable**

Format : u32 status = EMC84xx_counter_enable(u32 CardID);

Purpose: To enable the counter function.

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMC84xx_initial function

- **EMC84xx_counter_disable**

Format : **u32 status = EMC84xx_counter_disable(u32 CardID);**

Purpose: To disable the counter function.

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMC84xx_initial function

- **EMC84xx_counter_read**

Format : **u32 status = EMC84xx_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 EMC84xx_initial function
port	u8	unused

Output:

Name	Type	Description
counter	u32	counter value counter[0] for IO_0 … counter[7] for IO_7

- **EMC84xx_counter_clear**

Format : **u32 status = EMC84xx_counter_clear(u32 CardID, u8 port,u8 channel);**

Purpose: To reset the counter value.

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMC84xx_initial function
port	u8	unused
Channel	u8	b0 = 0: no function b0 = 1: clear IO_0 counter … b7 = 0: no function b7 = 1: clear IO_7 counter

6.6 RS232/422/485 setup

As a serial to Ethernet converter, we must setup the serial port parameters of the module to meet the communication protocol.

EMC84xx_serial_port_set( ) is used for parameters setting and

EMC84xx_serial_port_read( ) is used to read back for verification.

- **EMC84xx_serial_port_set**

Format : u32 status = EMC84xx_serial_port_set(u32 CardID, u8 baud_rate, u8 data_bit, u8 parity, u8 stop_bits, u8 flow_control, u8 mode);

Purpose: To set the serial port configuration.

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMC84xx_initial function
baud_rate	u8	Set the baud rate. 0: 1200 1: 2400 2: 4800 3: 9600(default) 4: 19200 5: 38400 6: 57600 7: 115200 8: 921600
data_bit	u8	Communication data bit setting 0: 5 bits 1: 6 bits 2: 7 bits 3: 8 bits (default)
parity	u8	Communication parity setting 0:Odd 1:Even 2:None(default)
stop_bits	u8	Communication stop bit setting 0: 1 bit (default) 1: 1.5 bit
flow_control	u8	Flow control setting 0:Xon/Xoff 1: Hardware(default) 2:None
mode	u8	Mode setting for EMC8485 set as 1: RS422 (default) 2: RS485 for EMC8432 this parameter is of no use.

- **EMC84xx_serial_port_read**

Format : **u32 status = EMC84xx_serial_port_read(u32 CardID, u8 &baud_rate, u8 &data_bit, u8 &parity, u8 &stop_bits, u8 &flow_control, u8 &mode);**

Purpose: To read the serial port configuration.

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMC84xx_initial function

Output:

Name	Type	Description
baud_rate	u8	Set the baud rate. 0: 1200 1: 2400 2: 4800 3: 9600(default) 6: 57600 5: 38400 4: 19200 7: 115200 8: 921600
data_bit	u8	Communication data bit setting 0: 5 bits 1: 6 bits 2: 7 bits 3: 8 bits
parity	u8	Communication data bit setting 0:Odd 1:Even 2:None(default)
stop_bits	u8	Communication stop bit setting 0: 1 bit (default) 1: 1.5 bit
flow_control	u8	Flow control setting 0: Xon/Xoff 1: Hardware(default) 2: None
mode	u8	Mode setting for EMC8485 set as 1: RS422 (default) 2: RS485 for EMC8432 this parameter is of no use.

6.7 Standalone function

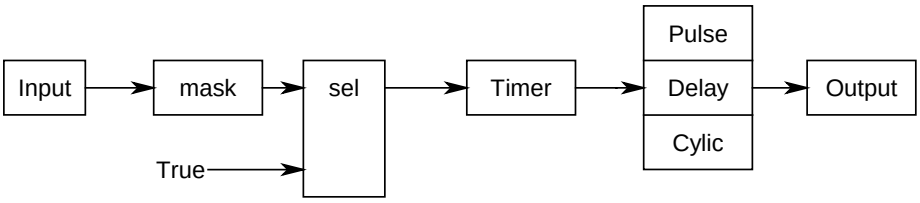
Standalone mode is the extension of EMC84xx module; it can work as I/O controller without the Ethernet existing.

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.

- **EMC84xx_standalone_enable**

Format : u32 status =EMC84xx_standalone_enable(u32 CardID)

Purpose: Enable standalone mode**.**

Parameters:

Input:

Name	Type	Description
CardID	u32	0~255 CardID assigned by EMC84XX_initial

- **EMC84xx_standalone_disable**

Format : u32 status =EMC84xx_standalone_disable(u32 CardID)

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

Parameters:

Input:

Name	Type	Description
CardID	u32	0~255 CardID assigned by EMC84XX_initial

- **EMC84xx_standalone_config_set**

Format : **u32 status =EMC84xx_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 EMC84XX_initial

data[32]	standalone_data	typedef struct _StandaloneData{
		u8 in_point_bit;
		u8 in_state_bit;
		u16 timer_value;
		u8 out_point_bit;
		u8 timer_mode;
		u8 out_mode;
		}
		in_point_bit
		//b7 ~ b0 = IO_7 ~ IO_0
		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 = IO_7 ~IO_0
		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 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. 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.

- **EMC84xx_standalone_config_read**

Format : **u32 status =EMC84xx_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 EMC84XX_initial

Output:

Name	Type	Description

data	standalone_data	typedef struct _StandaloneData{
		u8 in_point_bit;
		u8 in_state_bit;
		u16 timer_value;
		u8 out_point_bit;
		u8 timer_mode;
		u8 out_mode;
		}
		in_point_bit
		//b7 ~ b0 = IO_7 ~ IO_0
		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

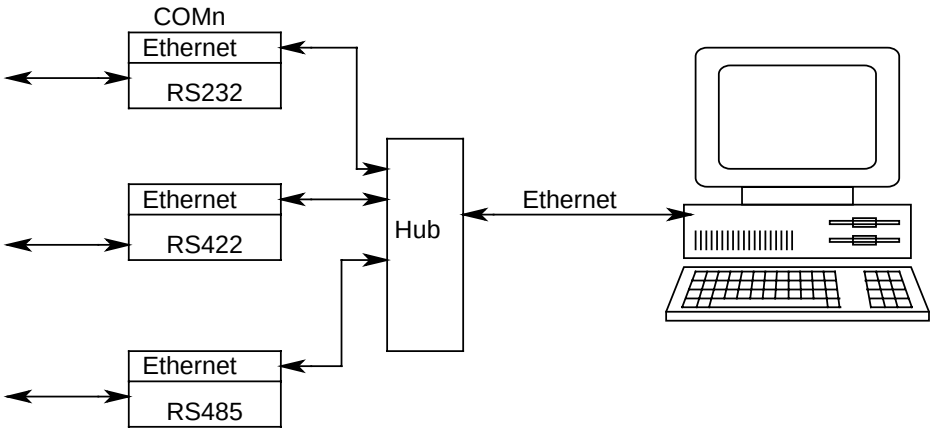
		timer_value
		// timer tick is 5ms per tick
		// timer_mode = delay / periodic out
		// setting value is delay timer
		// timer_mode = pulse out
		// setting value is active time of pulse
		// if timer_value = 10 ,
		// the delay time is 10 * 5 ms = 50 ms
		out_point_bit
		//b7 ~ b0 = IO_7 ~ IO_0
		out_mode
		// 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

6.8 Virtual COM port

The serial to Ethernet module is in fact use Ethernet to connect to computer but from the user side, we would rather take it as a COM port for the existing program or the traditional COM port programmers.

As the followings shown, the virtual COM port will take the RS232 (RS422 or RS485) as the computer inside COM port but it really connect to the serial port via Ethernet.



Using

EMC84xx_VSPM_install( ) to add a virtual COM port to the system,

EMC84xx_VSPM_remove( ) to remove the virtual COM port and release resource.

EMC84xx_VSPM_set( ) to setup the Ethernet IP of the converter module to the virtual COM port.

EMC84xx_VSPM_connect( ) connect the virtual COM port (logic device) to the converter module (physical device).

EMC84xx_VSPM_info( ) to read the Ethernet IP of the converter module from the virtual COM port.

EMC84xx_VSPM_close( ) to close the connection.

- **EMC84xx_VSPM_install**

Format : **u32 Status = EMC84xx_VSPM_install(u8 *vID);**

Purpose: To add a virtual com port module.

Parameters:

Output:

Name	Type	Description
vID	u8	Return the number vID virtual com port module.

- **EMC84xx_VSPM_remove**

Format : **u32 Status =EMC84xx_VSPM_remove(u8 vID);**

Purpose: To remove a virtual com port module.

Parameters:

Input:

Name	Type	Description
vID	u8	vID assigned by EMC84xx_VSPM_install function.

- **EMC84xx_VSPM_set**

Format : **u32 Status = EMC84xx_VSPM_set(u8 vID,u8 ip[4]);**

Purpose: To set the virtual COM port IP.

Parameters:

Input:

Name	Type	Description
vID	u8	vID assigned by EMC84xx_VSPM_install function.
ip[4]	u8	IP of virtual COM device (the converter module) Default IP is: 192.168.0.100 IP[0]=192 IP[1]=168 IP[2]=0 IP [3]=100

- **EMC84xx_VSPM_connect**

Format : **u32 Status = EMC84xx_VSPM_connect(u8 vID);**

Purpose: The virtual COM (logic device) connect to the remote IP (converter module)

Parameters:

Input:

Name	Type	Description
vID	u8	vID assigned by EMC84xx_VSPM_install function.

Note: use EMC84xx_VSPM_set to set the IP first.

- **EMC84xx_VSPM_info**

Format : **u32 Status = EMC84xx_VSPM_info(u8 vID, u8 *status, u8 remote_IP[4]);**

Purpose: Get the virtual COM information.

Parameters:

Input:

Name	Type	Description
vID	u8	vID assigned by EMC84xx_VSPM_install function.

Output:

Name	Type	Description
status	u8	Return the virtual device status 0:idle 1:connect
remote_IP[4]	u8	IP of virtual COM port (converter module) Default IP is: 192.168.0.100 IP[0]=192 IP[1]=168 IP[2]=0 IP [3]=100

- **EMC84xx_VSPM_close**

Format : **u32 Status = EMC84xx_VSPM_close(u8 vID);**

Purpose: To close the virtual COM (logic device) connection.

Parameters:

Input:

Name	Type	Description
vID	u8	vID assigned by EMC84xx_VSPM_install function.

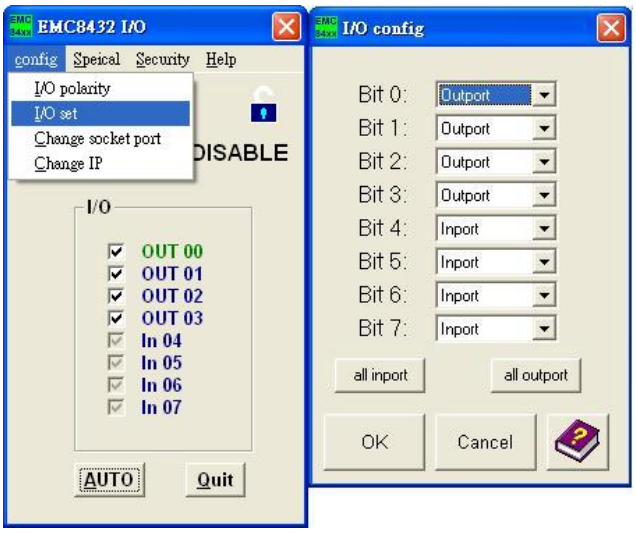
7. Standalone mode user configuration utility

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.

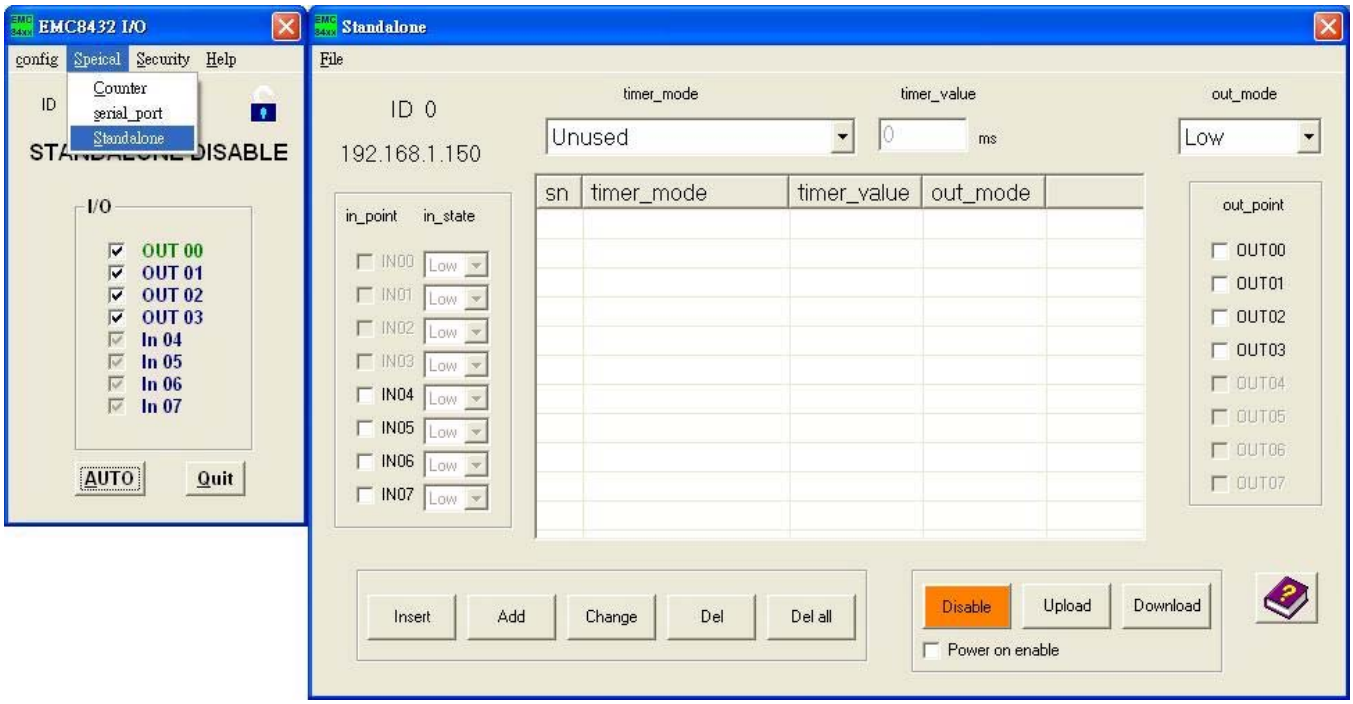
7.1 Overview of user configuration utility

-- 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.

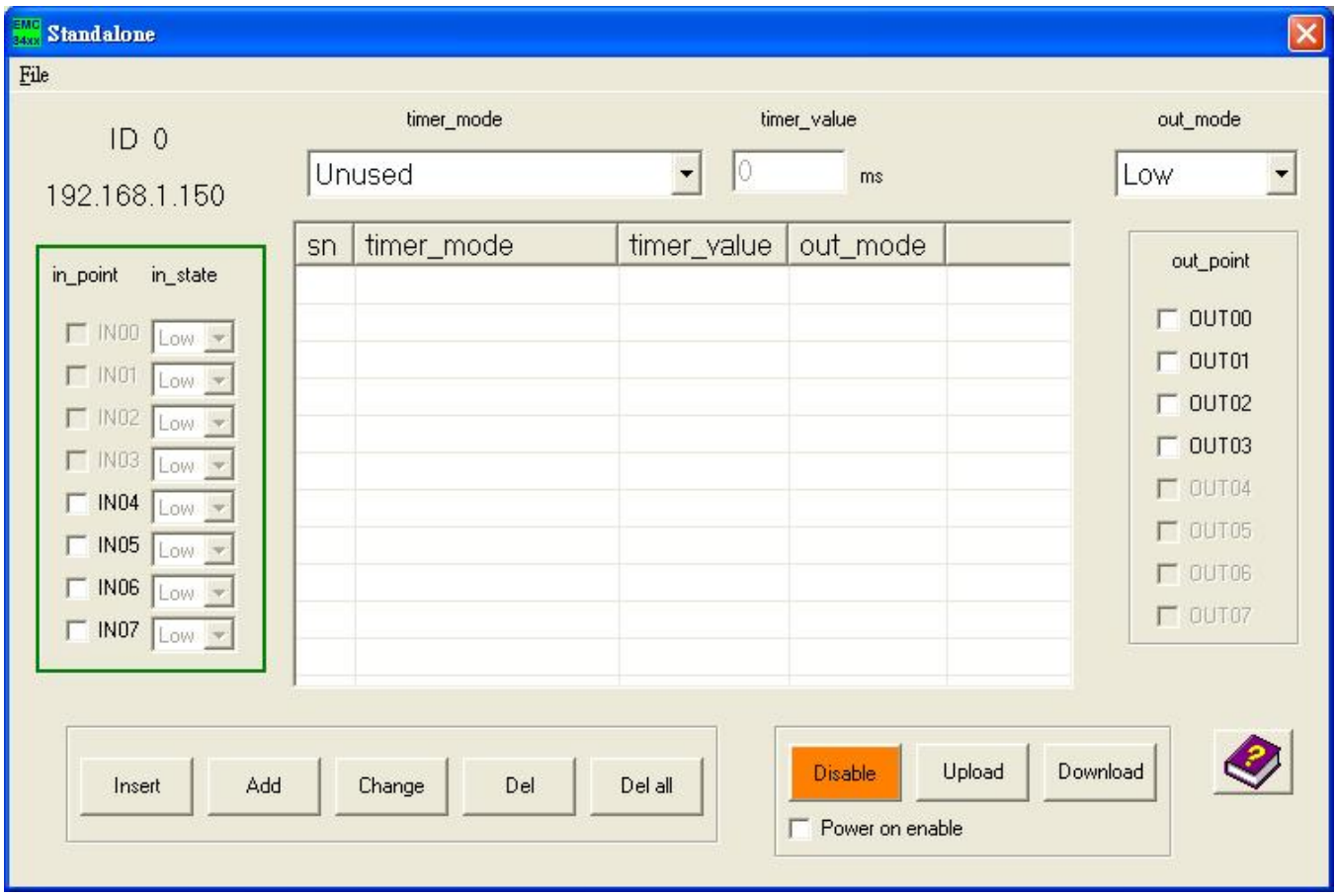
Block5: standalone mode command upload /download, start/stop.

Block6: power on standalone mode enable/ disable.

7.2 Configure a command

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

-- input configuration

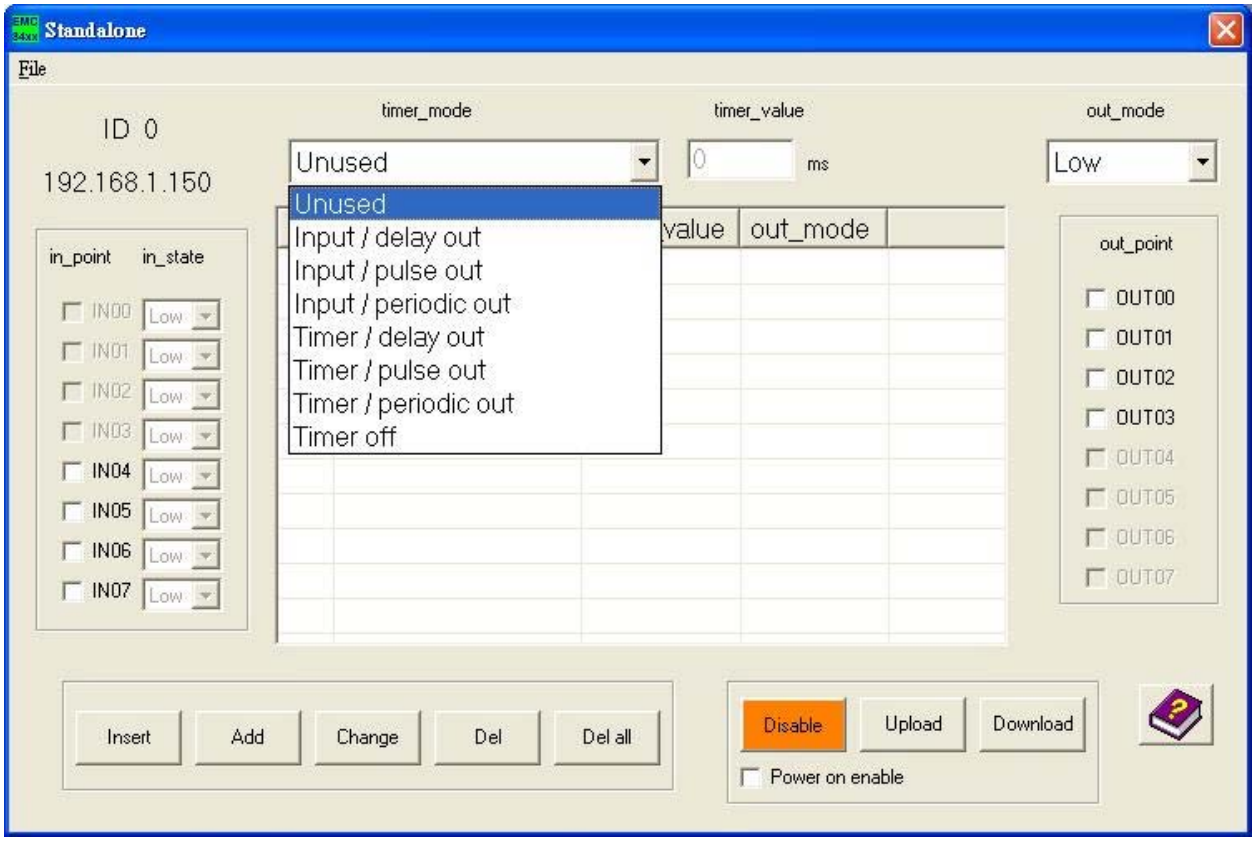


From the above diagram, check the input point and its state the current command will take care.

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 EMC84xx module.

-- timer configuration

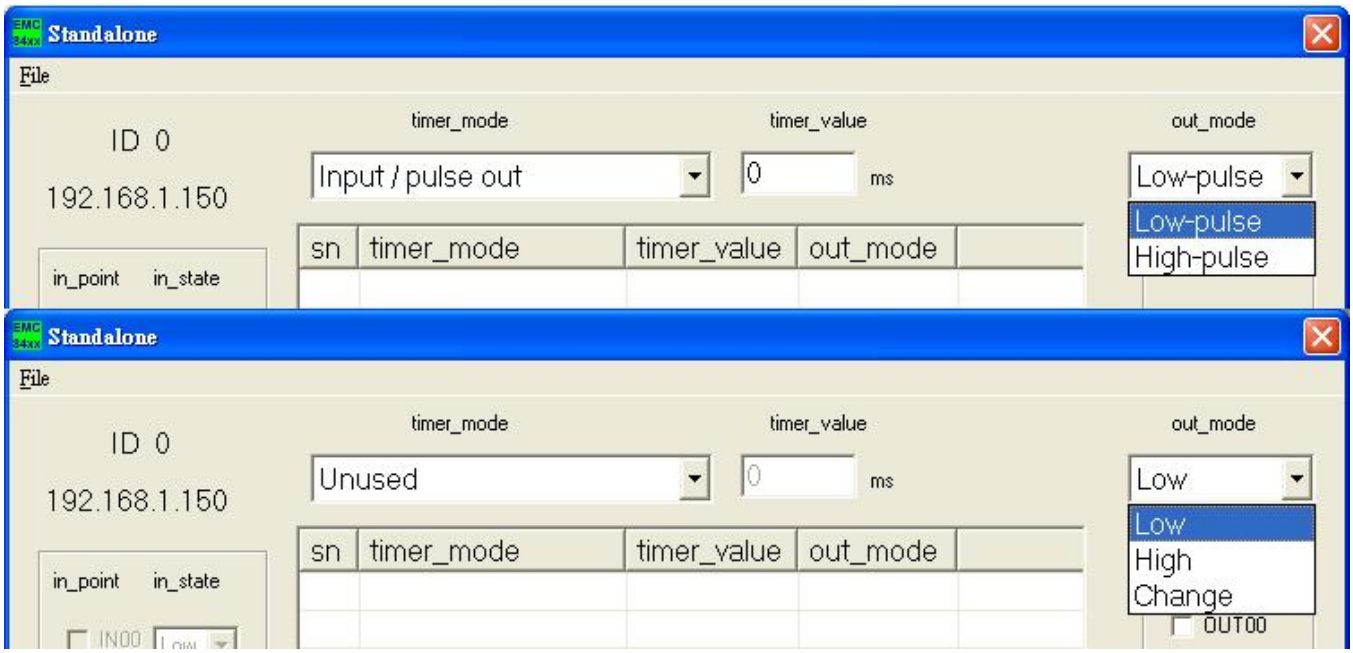


If the inputs meet the condition you configured, it will trigger the timer to operate. The time provides several kinds of working mode:

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

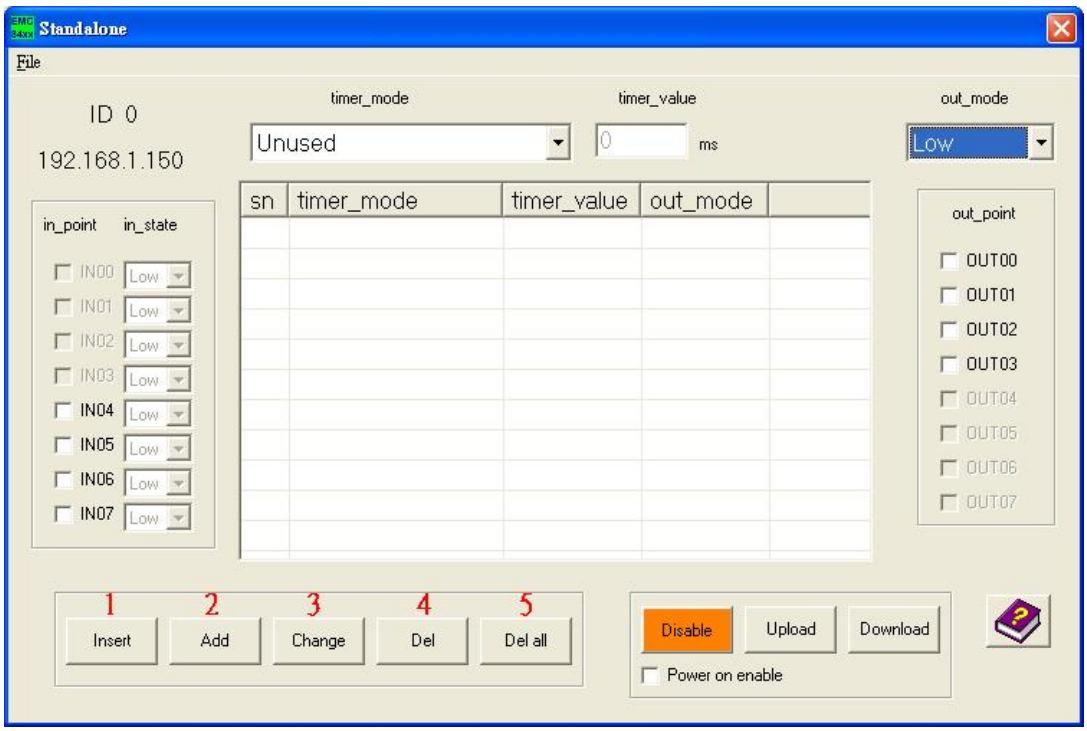


The timer depending on its working mode controls the output. The output can be configured as active low, active high or toggle.

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

7.3 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

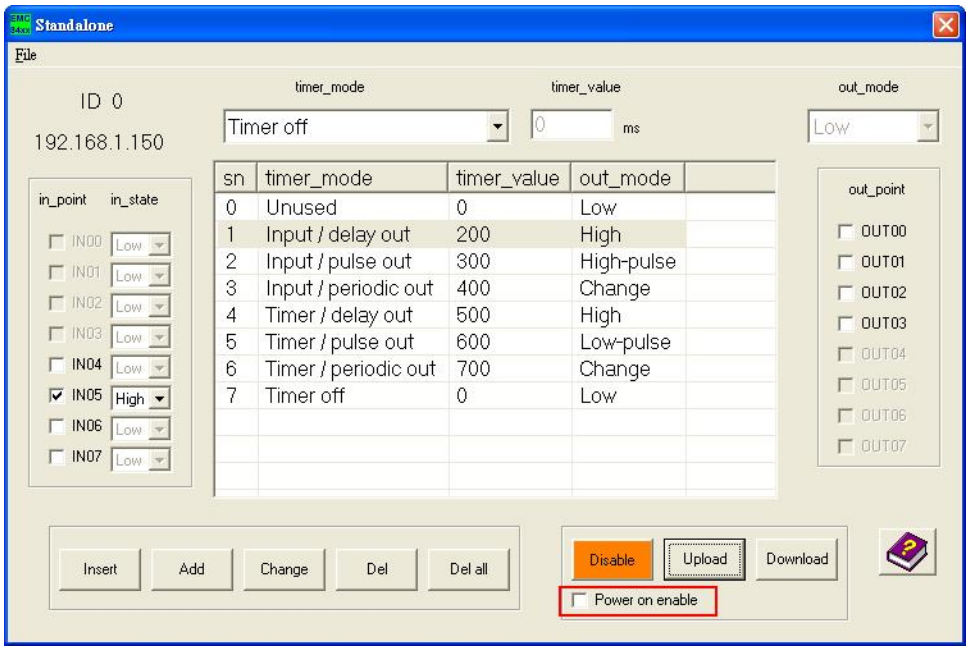
3: Change: modify the existing command line

4: Del: delete the highlighted command line

5: Del al: clear all the commands

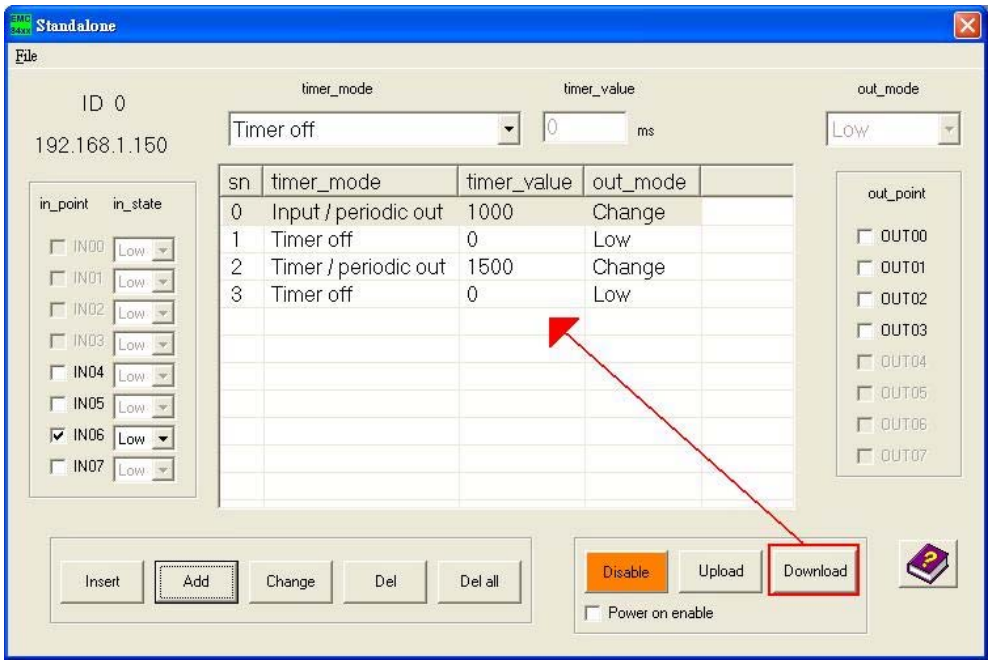
7.4 Upload program

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.



7.5 Download program

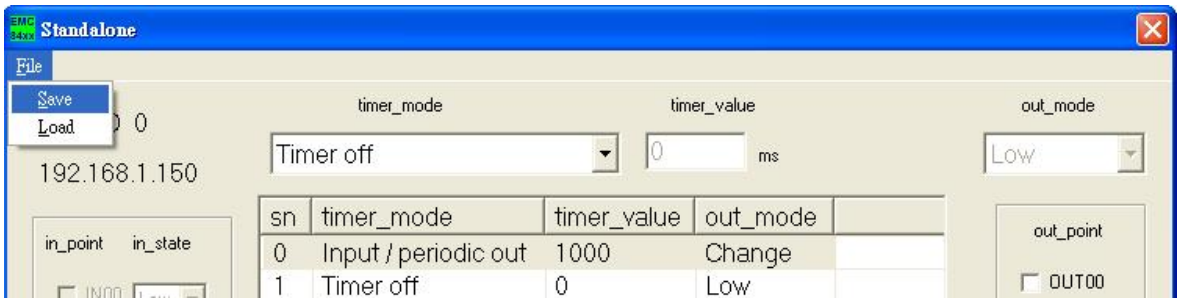
If you have connected with EMD8216 module via Ethernet, you can download the stored program from the module.



7.6 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.



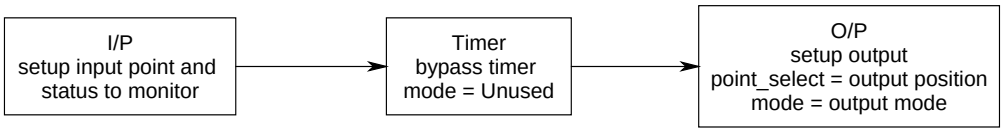
7.7 Enable/Disable standalone function

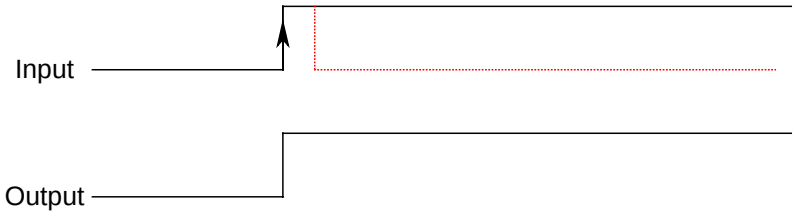
Standalone mode can enable or disable by the button as following shown.

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

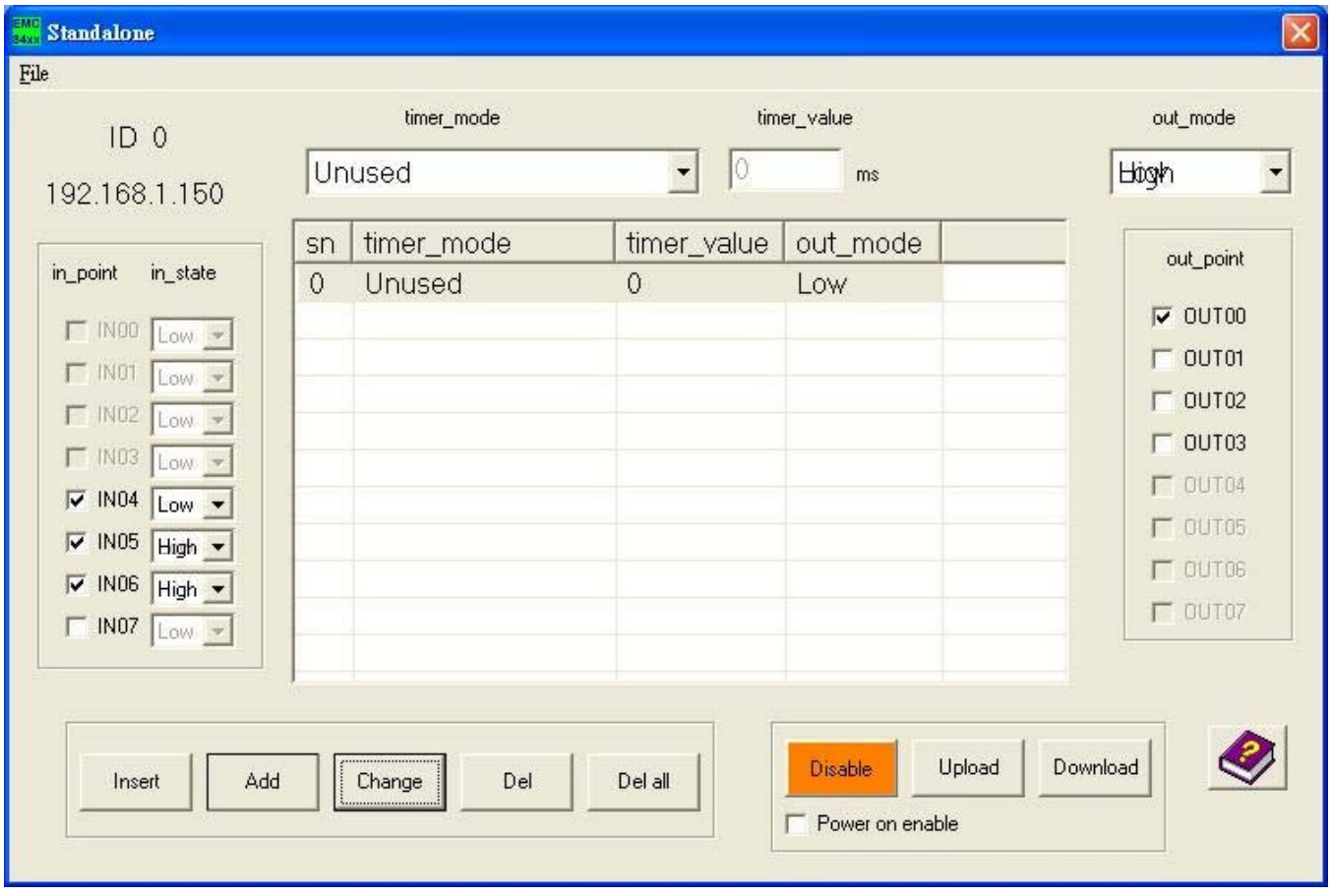
8. Standalone mode application examples

8.1 Monitoring input if condition meets, trigger output

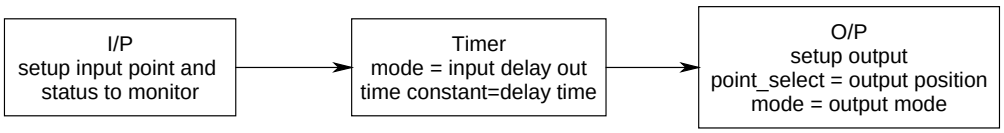


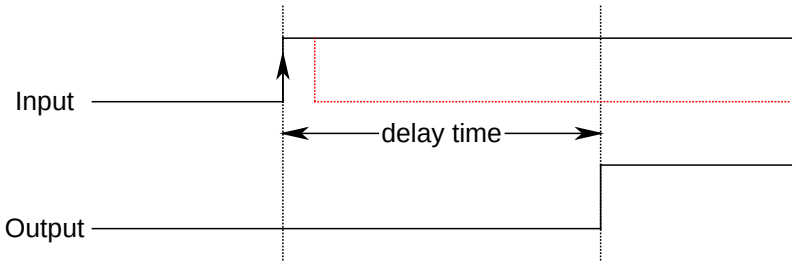


Say, you want to watch IN04 low, IN05 and IN06 high to trigger output OUT00 to low. Program as the following shown.

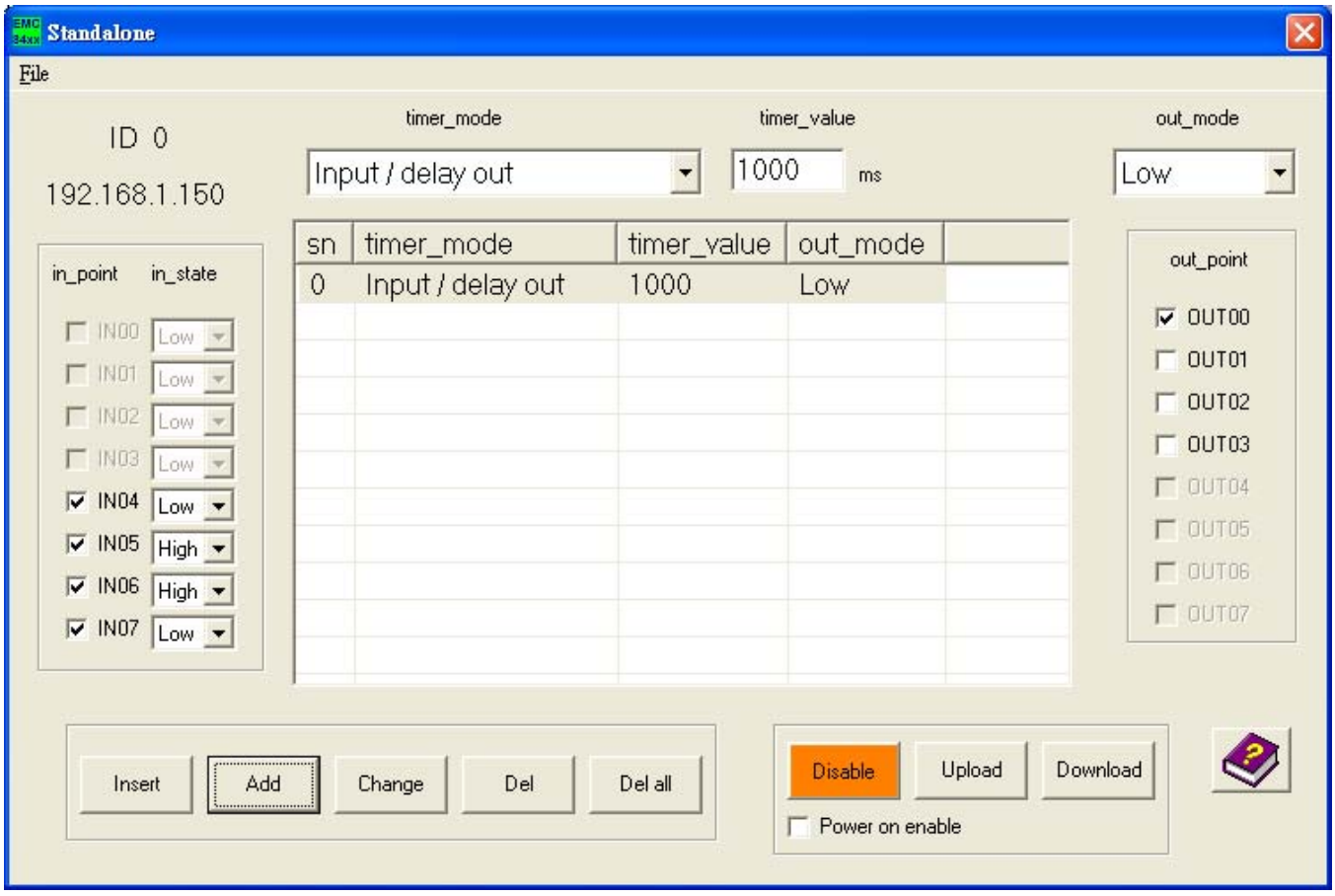


8.2 Monitoring the input if condition meets, delay to trigger output

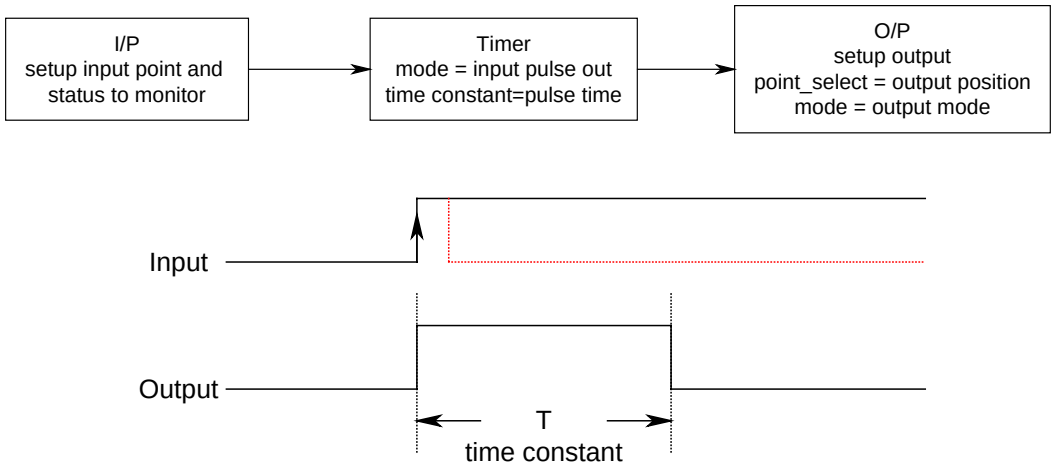




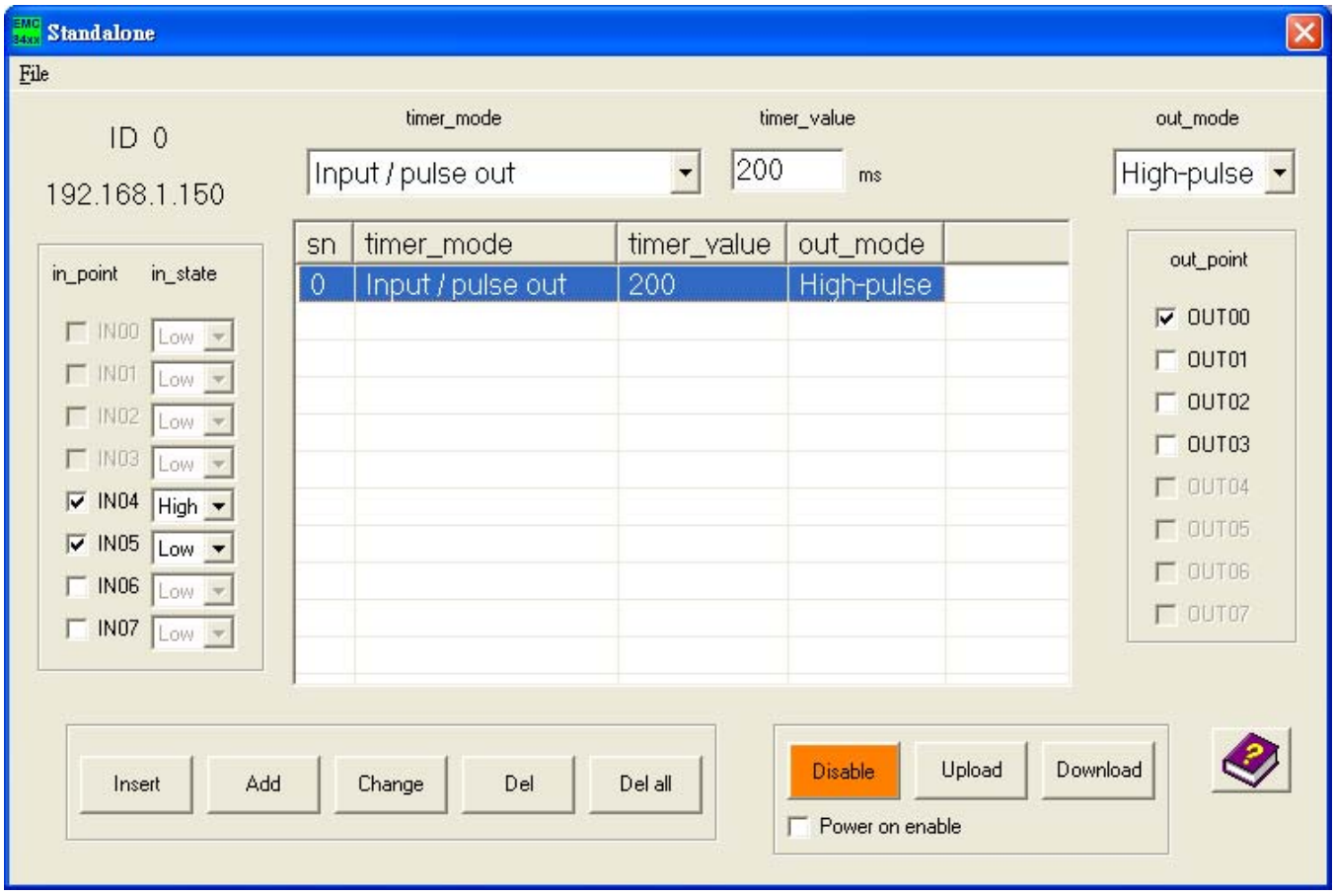
Say, you want to watch IN04 and IN07 are low and IN05 and IN06 are high to trigger output OUT00 to low. Program as the following shown.



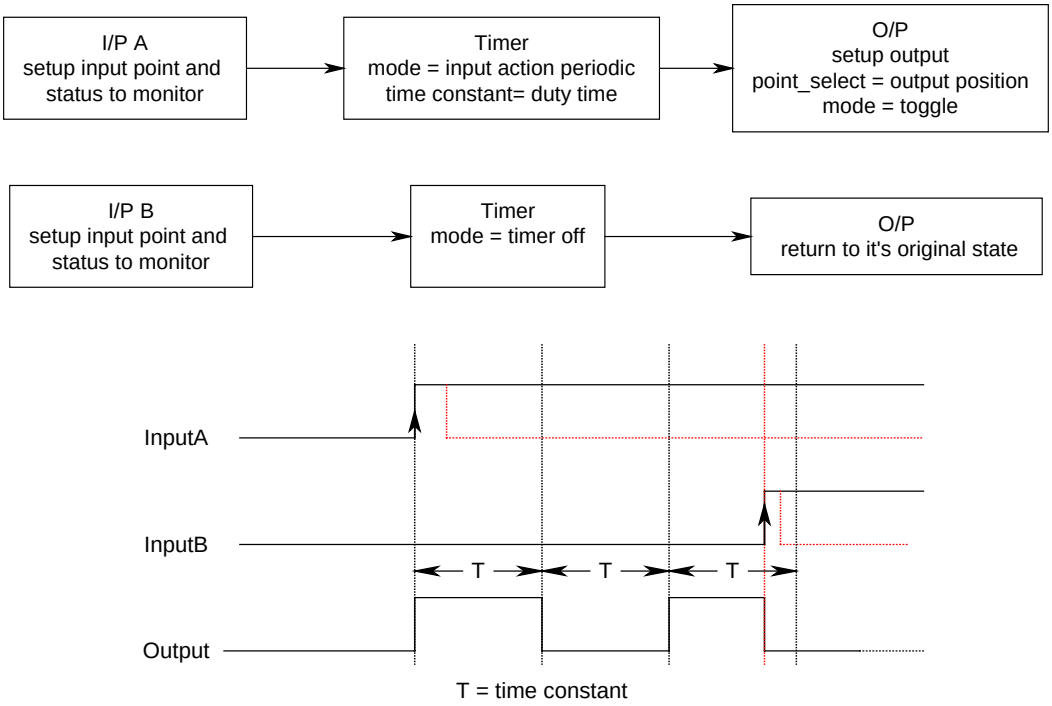
8.3 Monitoring the input if condition meets, output pulse



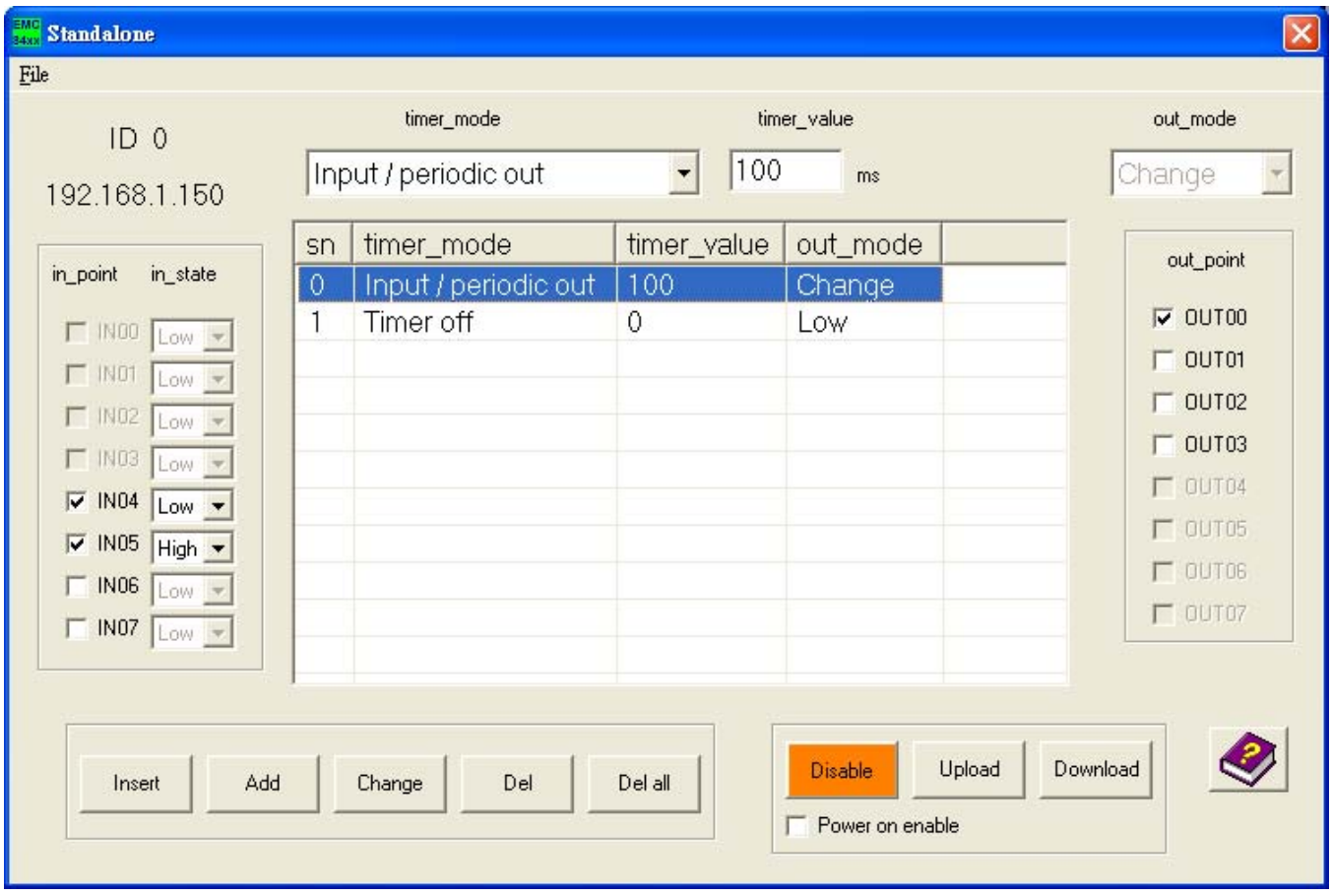
Say, you want to watch IN04 is high low and IN05 is low to trigger output OUT00 to pulse high. Program as the following shown.



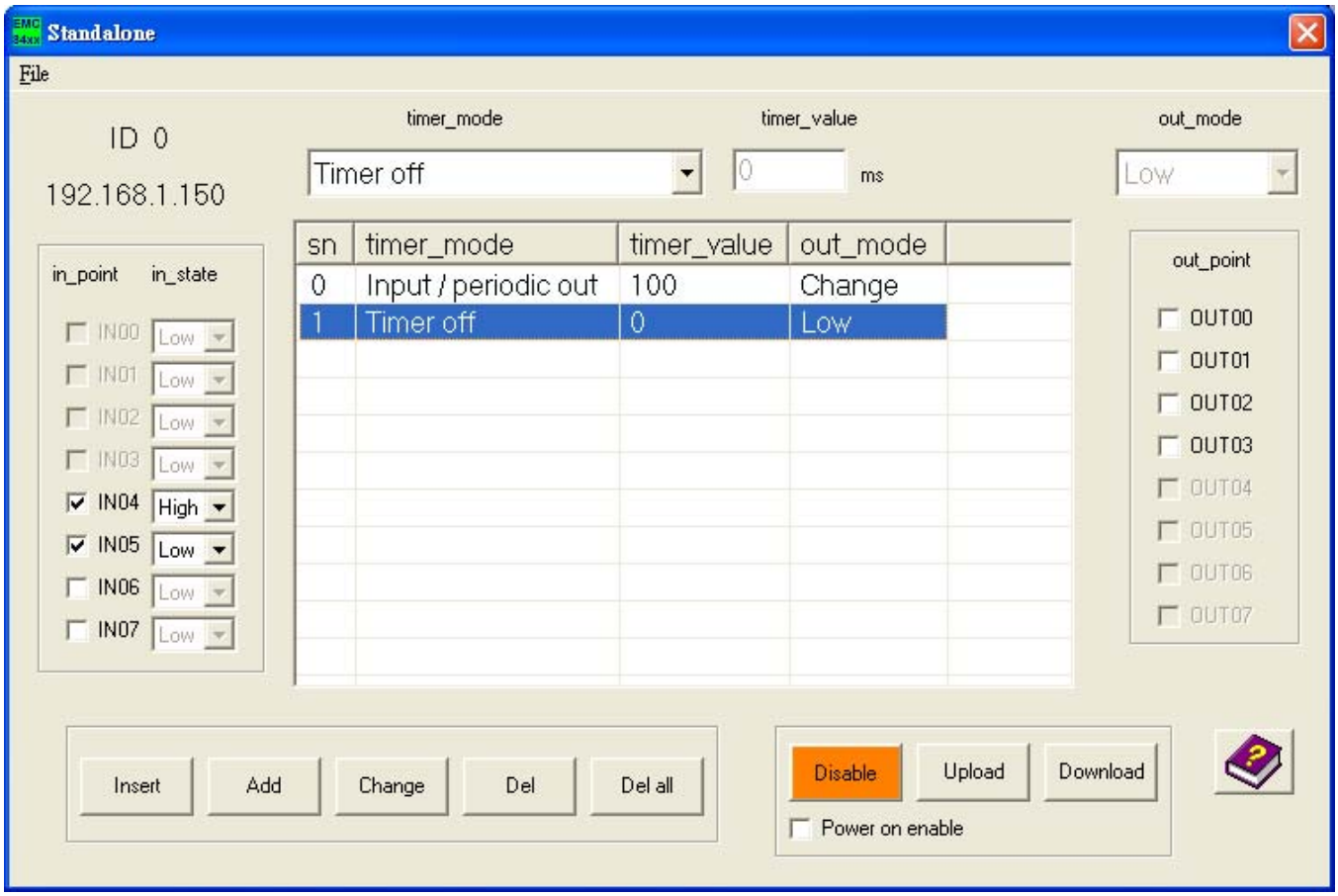
8.4 Monitoring the input if condition meets, output periodically and stop by some special input condition



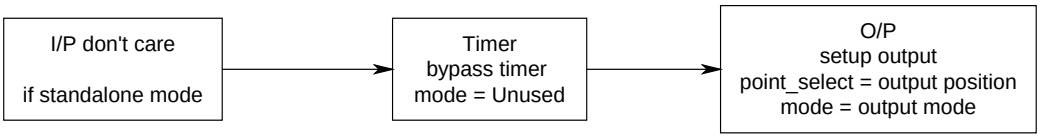
Say, you want to watch IN04 is low and IN05 is high to trigger output OUT00 to toggle. Program as the following shown.

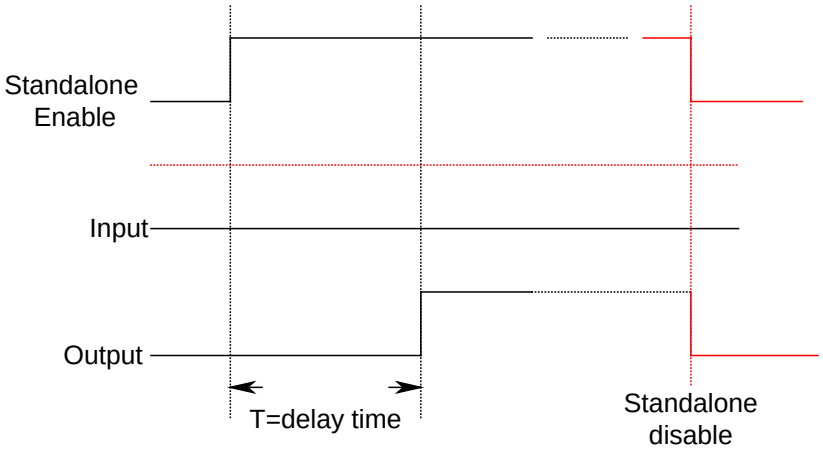


Then, if we want IN10 is high and IN11 is low to trigger to stop the timer. Program as the following shown.

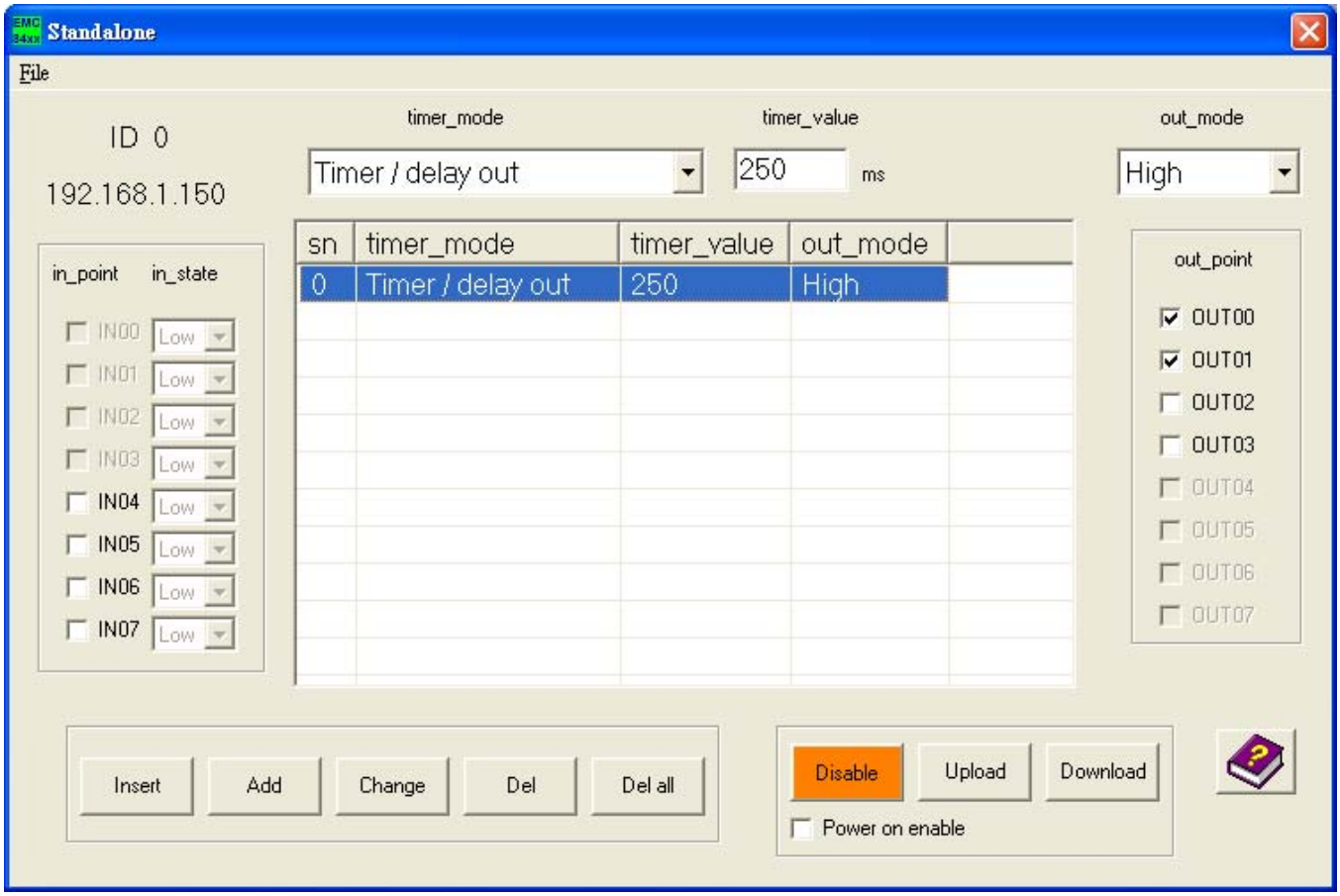


8.5 Don’t care the input if standalone enabled, trigger output

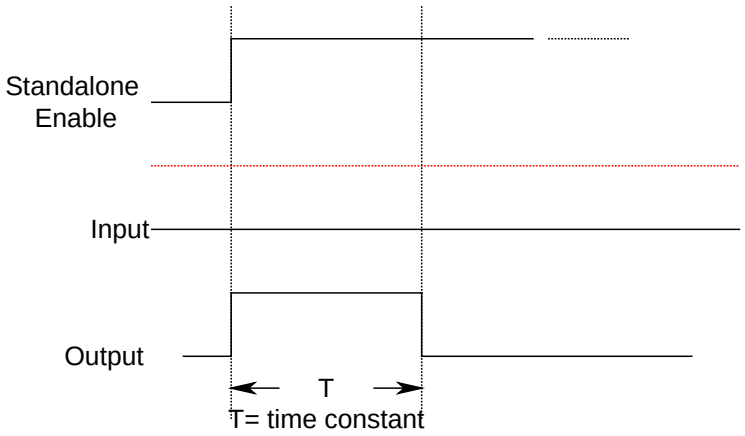




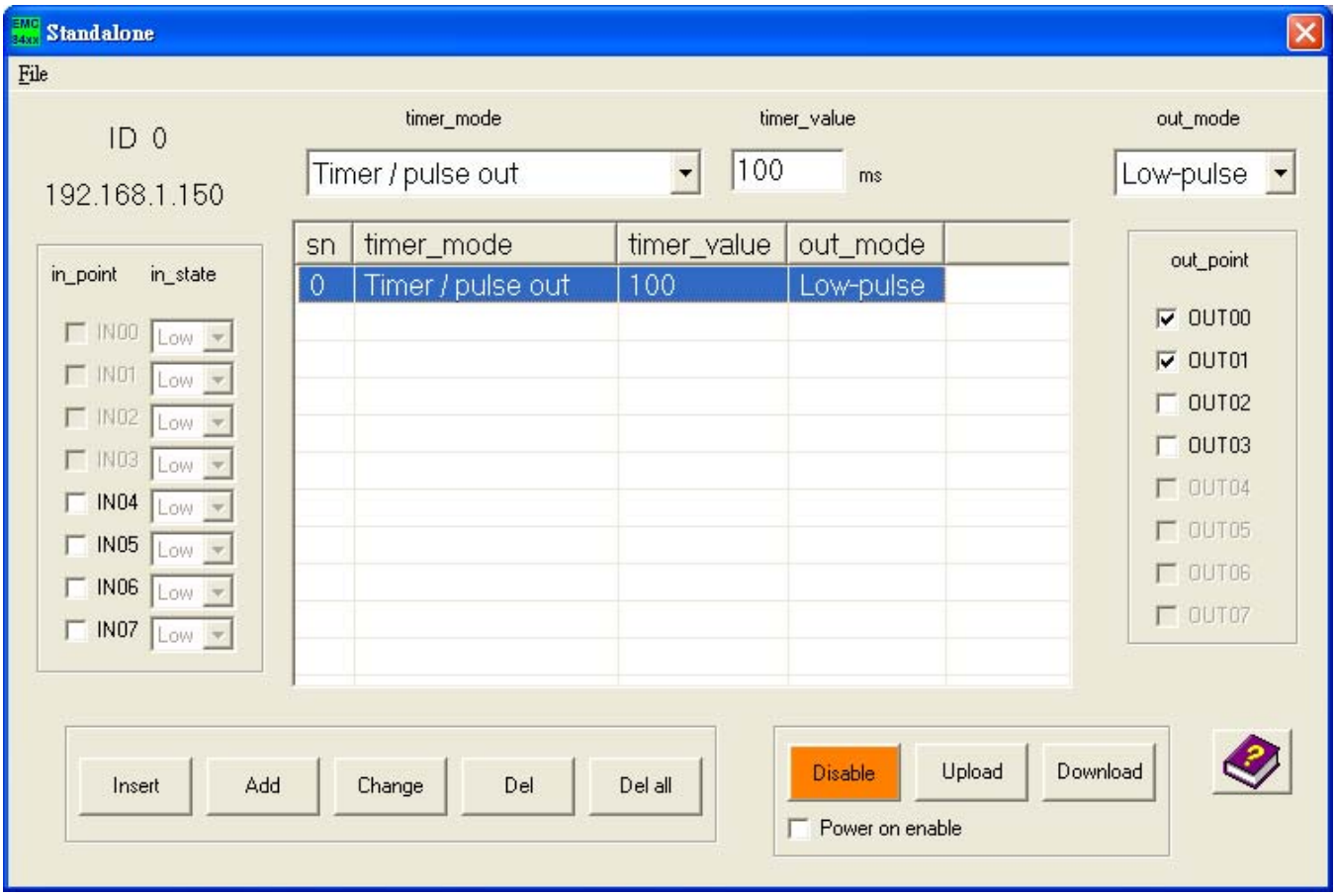
Say, don’t care any input just output OUT00 OUT01 high as the standalone mode enabled. Program as the following shown.



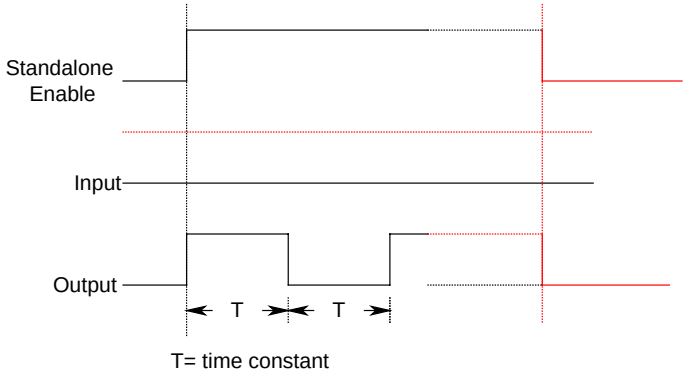
8.6 Don’t care the input if standalone enabled, trigger pulse



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

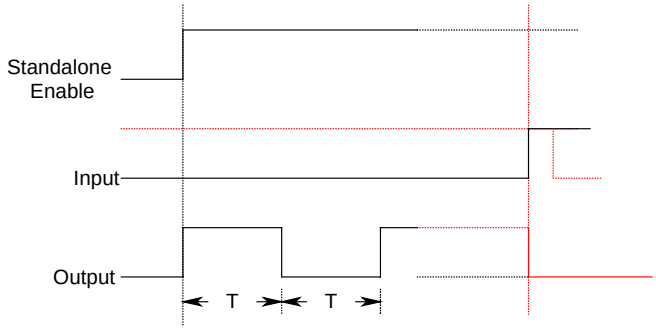


8.7 Don’t care the input if standalone enabled, output periodically



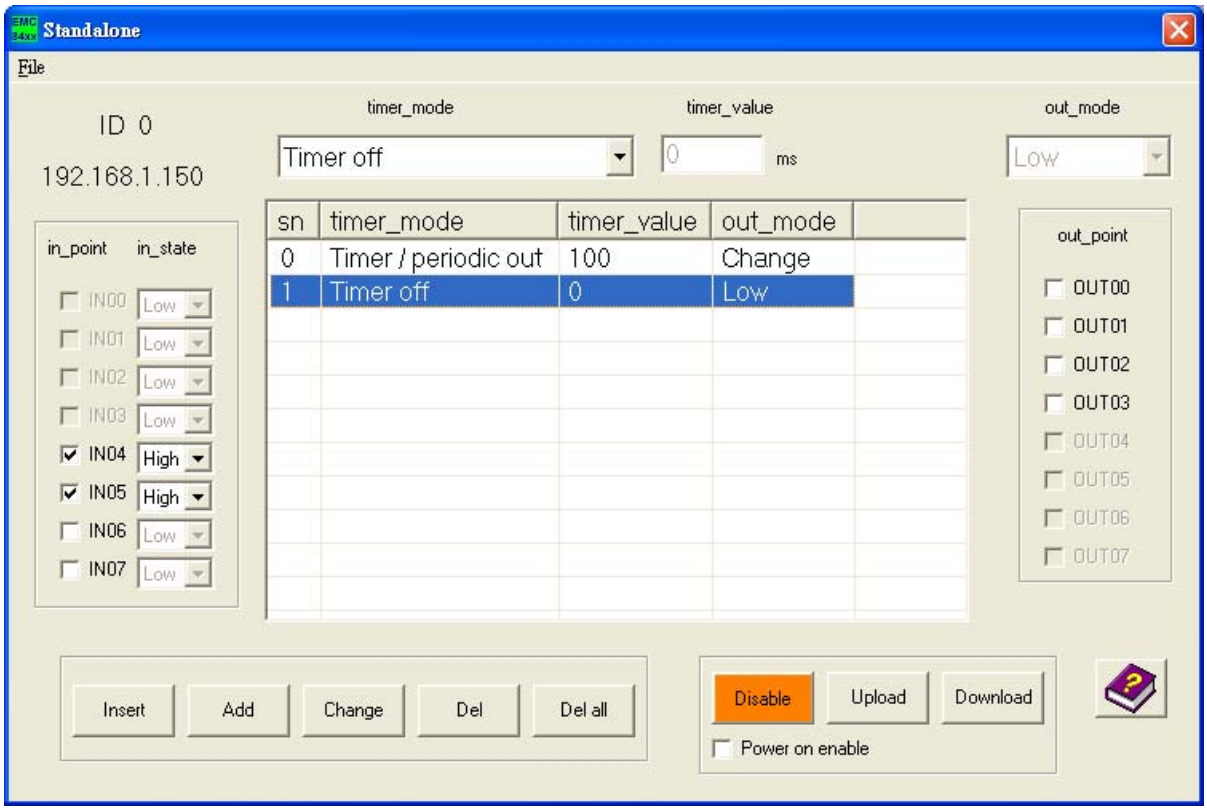
T= time constant

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 IN04 and IN05 are high. The program is shown as followings.



9. DLL list

	Function Name	Description
1.	EMC84xx_initial( )	Map IP and get model parameter
2.	EMC84xx_close( )	EMC84xx close
3.	EMC84xx_firmware_version_read( )	Read the firmware version
4.	EMC84xx_socket_port_change( )	To change the communicate port number of EMC84xx
5.	EMC84xx_IP_change( )	To change the communicate IP of EMC84xx
6.	EMC84xx_reboot( )	To reboot EMC84xx
7.	EMC84xx_security_unlock( )	Unlock security
8.	EMC84xx_security_status_read( )	Read lock status
9.	EMC84xx_password_change( )	Change password
10.	EMC84xx_password_set_default( )	Rest to factory default password
11.	EMC84xx_port_config_set( )	To setup the configuration
12.	EMC84xx_port_config_read( )	To read the configuration
13.	EMC84xx_port_polarity_set()	To set the polarity value
14.	EMC84xx_port_polarity_read( )	To read all the polarity value
15.	EMC84xx_port_set( )	To set the out port value
16.	EMC84xx_port_read( )	To read all the port value
17.	EMC84xx_point_config_set( )	To setup the configuration
18.	EMC84xx_point_config_read( )	To read the configuration
19.	EMC84xx_point_polarity_set( )	To set the polarity value
20.	EMC84xx_point_polarity_read( )	To read all the polarity value
21.	EMC84xx_point_set( )	To set the outport value
22.	EMC84xx_point_read( )	To read all the port value
23.	EMC84xx_counter_mask_set( )	To set the counter channel mask
24.	EMC84xx_counter_enable( )	To enable the counter function
25.	EMC84xx_counter_disable( )	To disable the counter function
26.	EMC84xx_counter_read( )	To read all the counter value
27.	EMC84xx_counter_clear( )	To reset the counter value
28.	EMC84xx_serial_port_set( )	To set the serial port configuration
29.	EMC84xx_serial_port_read( )	To read the serial port configuration
30.	EMC84xx_standalone_enable( )	Enable standalone mode
31.	EMC84xx_standalone_disable( )	Disable (stop) standalone mode
32.	EMC84xx_standalone_config_set( )	Set the standalone configuration
33.	EMC84xx_standalone_config_read( )	Read the standalone configuration

34.	EMC84xx_VSPM_install( )	To add a virtual com port module
35.	EMC84xx_VSPM_remove( )	To remove a virtual com port module
36.	EMC84xx_VSPM_set( )	To set the virtual COM port IP
37.	EMC84xx_VSPM_connect( )	The virtual COM connect to the remote IP
38.	EMC84xx_VSPM_info( )	Get the virtual COM information
39.	EMC84xx_VSPM_close( )	To close the virtual COM connection

10. EMC84xx Error code table

10.1 EMC84xx 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 is already used
300	ID_ERROR	CardID is not acceptable
301	PORT_ERROR	Port parameter unacceptable or unreachable
305	PARAMETERS_ERROR	Parameters 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	CREATE_SOCKET_ERROR	Socket can not create
312	CHANGEIP_ERROR	Error while change IP
313	COUNTER_MASK_SET_ERROR	Count mask error
314	COUNTER_ENABLE_ERROR	Can not enable counter function
315	COUNTER_DISABLE_ERROR	Can not disable counter function
316	COUNTER_READ_ERROR	Can not read counter data
317	COUNTER_CLEAR_ERROR	Can not clear counter data
318	PORT_CONFIG_SET_ERROR	Can not setup port configuration
319	PORT_CONFIG_READ_ERROR	Can not read port configuration
320	PORT_SET_ERROR	Can not set port data
321	PORT_READ_ERROR	Can not read port data
322	SERIAL_PORT_SET_ERROR	Can not set serial port configuration
323	SERIAL_PORT_READ_ERROR	Can not read serial port configuration
330	VSPM_REMOTE_ERROR	Can not talk with physical device
331	VSPM_CONNECT_ERROR	Can not connect virtual COM
332	VSPM_CLOSE_ERROR	Can not closed virtual COM
333	VSPM_INFO_ERROR	Can not get virtual COM info
334	VSPM_NOT_FOUND	Can not find virtual COM

340	STANDALONE_ENABLE_ERROR	Can not enable standalone function
341	STANDALONE_DISABLE_ERROR	Can not disable standalone function
342	STANDALONE_CONFIG_ERROR	Can not setup / read standalone configuration