EMD-8204/08
Ethernet Digital I/O module

Software Manual (V2.0)

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-820x.dll v1.0
2.0	1. Add Chapter 5.2 Types of I/O classified by isolation
	2. Add Chapter 5.3 Types of Output classified by driver device
	3. Add Chapter 5.4 Protection of output transient on inductive load
	4. Add Chapter 5.5 Inrush current consideration
	5. Add Chapter 5.6 Virtual IO
	6. Add Chapter 7 Flow chart of application implementation
	7. Add new function in Chapter 8
	EMD820x_dll_version_read
	EMD820x_subnet_mask_set
	EMD820x_subnet_mask_read
	EMD820x_counter_mask_read
	EMD820x_WDT_set
	EMD820x_WDT_read
	EMD820x_WDT_enable
	EMD820x_WDT_disable
	EMD820x_standalone_enable
	EMD820x_standalone_disable
	EMD820x_standalone_V_config_set
	EMD820x_standalone_V_config_read
	EMD820x_standalone_config_clear
	EMD820x_D2D_config_set
	EMD820x_D2D_config_read
	EMD820x_D2D_connection_clear
	EMD820x_D2D_connection_state_read
	8. Add Chapter 9 User configuration utility of Standalone mode
	9. Add Chapter 10 Standalone mode application examples
	10. Add and Modify communication commands in Chapter 11.5
	11. Add Chapter 15 Error codes table for UDP Success_flag

Contents

1. How to install the software of EMD820x

Please register as user's club member to download the
"Step by step installation of EMD820x" document from http://automation.com.tw

1.1 Install the EMD driver

The ether net module cannot found by OS as PCI cards. You can just install the driver without the module installed. Execute the file ..\install\EMD820x_Install.exe to install the driver, Api and demo program automatically.
For a more detail descriptions, please refer "Step by step installation of EMD820x".

2. Where to find the file you need

Windows2000, XP and up

In Windows 2000,XP,Win7 system, the demo program can be setup by EMD820x_Install.exe.

If you use the default setting, a new directory .. \JS Automation\EMD820x will generate to put the associate files.

.. / JS Automation /EMD820x/API (header files and VB,VC lib files)

.. / JS Automation /EMD820x/Driver (copy of driver code)

.. / JS Automation /EMD820x/exe (demo program and source code)

The dll is located at ..\system.

3. About the EMD820x software

EMD820x software includes a set of dynamic link library (DLL) based on socket that you can utilize to control the interface functions.

Your EMD820x 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 EMD820x functions within Windows' operation system environment.

3.1 What you need to get started

To set up and use your EMD820x software, you need the following:

- EMD820x software
- EMD820x hardware

3.2 Software programming choices

You have several options to choose from when you are programming EMD820x 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 EMD820x software.

4. EMD820x Language support

The EMD820x software library is a DLL used with Windows OS. You can use these DLL with any Windows integrating development environment that can call Windows DLLs.

4.1 Building applications with the EMD820x software library

The EMD820x function reference section contains general information about building EMD820x applications, describes the nature of the EMD820x functions used in building EMD820x 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.

EMD820x Windows Libraries

The EMD820x for Windows function library is a DLL called **EMD820x.dll**. Since a DLL is used, EMD820x functions are not linked into the executable files of applications. Only the information about the EMD820x functions in the EMD820x 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 EMD820x functions in EMD820x.dll.

Header Files and Import Libraries for Different Development Environments		
Development Environment	Header File	Import Library
Microsoft C/C++	EMD820x.h	EMD820xVC.lib
Borland C/C++	EMD820x.h	EMD820xBC.lib
Microsoft Visual Basic	EMD820x.bas	

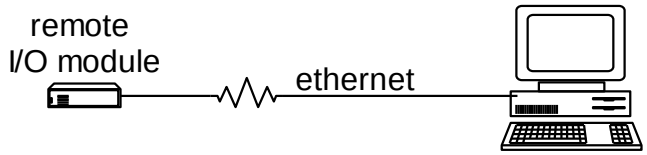
Table 1

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

5.1 I/O communicate via Ethernet

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

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



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

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.

5.2 Types of I/O classified by isolation

If the system and I/O are not electrically connected, we call it is isolated. There are many kinds of isolation: by transformer, by photo-coupler, by magnetic coupler,… Any kind of device, they can break the electrical connection without breaking the signal is suitable for the purpose.

Currently, photo-coupler isolation is the most popular selection, isolation voltage up to 2000V or over is common. But the photo-coupler is limited by the response time, the high frequency type cost a lot. The new selection is magnetic coupler, it is design to focus on high speed application.

The merit of isolation is to avoid the noise from outside world to enter the PC sysmodule, if the noise comes into module without elimination, the system maybe get "crazy" by the noise disturbance. Of course the isolation also limits the versatile of programming as input or output at the same pin as the TTL does. The inter-connection of add-on card and wiring board maybe extend to several meters without any problem.

The non-isolated type is generally the TTL level input/output. The ground and power source of the input/output port come from the system. Generally you can program as input or output at the same pin as you wish. **The connection wiring is limited to 50cm or shorter** (depends on the environmental noise condition).

5.3 Types of Output classified by driver device

There are several devices used as output driver, the relay, transistor or MOS FET, SCR and SSR. Relay is electric-mechanical device, it life time is about 1,000,000 times of switching. But on the other hand it has many selections such as high voltage or high current. It can also be used to switch DC load or AC load. For the practical application, using relay to switch an inductive load is common but the surge voltage and current during on or off will damage the relay contact and shorten its life.

Transistor and MOS FET are basically semi-permanent devices. If you have selected the right ratings, it can work without switching life limit. But the transistor or MOS FET can only work in DC load condition.

The transistor or MOS FET also have another option is source or sink. For PMOS or PNP transistor is source type device, the load is one terminal connects to output and another connects to common ground, but NPN or NMOS is one terminal connects to output and the other connects to VCC+. **If you are concerned about hazard from high DC voltage while the load is floating, please choose the source type driver device.**

SCR (or triac) is seldom direct connect to digital output, but his relative SSR is the most often selection. In fact, SSR is a compact package of trigger circuit and triac. You can choose zero cross trigger (output command only turn on the output at power phase near zero to eliminate surge) or direct turn on type. SSR is working in AC load condition.

5.4 Protection of output transient on inductive load

No matter what type of the output driver, to switch the inductive load is potentially to induce transient high voltage. The induced high voltage will reach several KV even the load voltage only

several volts. Semiconductor driver will punched through by the high voltage to cause the output a fatal error, even the rely contact can also results in micro-weld akin to spot welding. This will case the relay contact welding together (always make) or bad conduction (always break) or abnormally make or break. Owing to the popular wiring board driver components can be NMOS, PMOS or relay. The former two are semiconductor type and limit to DC load. The relay output may be AC or DC load, we will put emphasis on relay and apply the result to NMOS and PMOS.

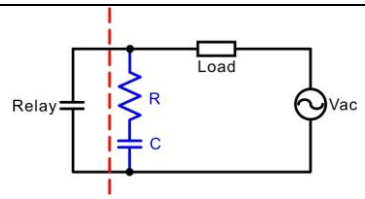
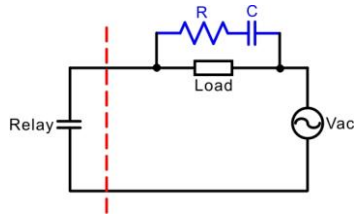
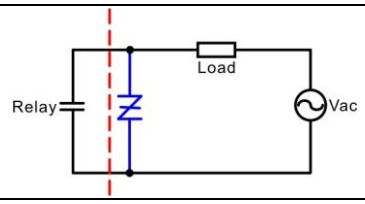
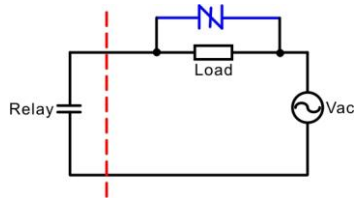
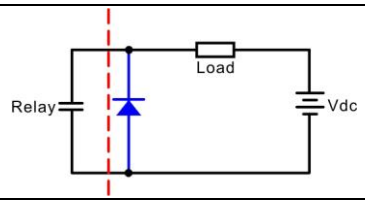
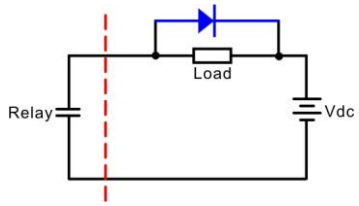
	Protection Connection	Comments	Design Note
A		Leakage current to load maybe cause malfunction. The effective release time will be lengthened.	Energy stored in the coil will dissipate by R and coil resistance. R: 0.5-1 Ohm per 1V contact voltage C: 0.5-1 uF per 1A current Use capacitor voltage of 200-300V and non-polarized type (AC capacitor) for DC operating voltage. If AC operating condition, the capacitor should at least 20% higher than working voltage.
B		The effective release time will be lengthened.	
C		To prevent the excess high voltage to damage the contact. There only slightly release delay at the release time.	ZNR rated voltage must select at least 20% higher than the working voltage. Selection of power of ZNR depends on the operation frequency, higher frequency needs larger power one. If possible, the diagram D is better than diagram C.
D			
E		The effective release time will be lengthened longer than RC snubber.	Energy stored in the coil will dissipate by coil resistance. (Only for DC working voltage) If possible, the diagram F is better than diagram E.
F			

fig. 5.4.1 Protection of output transient

The above list and example schematics show the possible solutions to protect the relay contact, it can also apply to NMOS or PMOS as they can only switch the DC working voltage.

5.5 Inrush current consideration

Inrush current (input surge current or switch-on surge) is the maximum, instantaneous input current drawn by an electrical device when first turned on.

The wiring board provides interface devices to drive the target under control device but the type of load, its inrush current and operating frequency are key factors to the contact life. You must take the inrush current into consideration to keep the interface in adequate working condition and life.
The following table gives you the design hints for your reference. Please note that a more conservative design will keep the drive circuit in a safer operating environment.

Type of load	Inrush current
Resistive load	Same as steady state current
Solenoid load	10-20 times of steady state current
Motor load	5-10 times of steady state current
Incandescent lamp load	10-15 times of steady state current
Mercury lamp load	~3 times of steady state current
Sodium vapor lamp load	1-3 times of steady state current
Capacitive load	20-40 times of steady state current
Transformer load	5-15 times of steady state current

5.6 Virtual IO

The EMD module has a special function call Virtual IO, it is an internal register, you can take it as input if you assign the input data to it or take it as output, if you assign the output data to it. The most valuable function is that the virtual IO can be a composed result of some input or output (may be real IO or Virtual IO) and then trigger the communication to transmit information to other modules. This is the very important base of device to device control function. Please refer the 9.4 Virtual I/O and broken line state configuration.

6. Function format and language difference

6.1 Error codes and address

Every EMD820x 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 EMD820x function is the parameter **CardID** which is set by ***EMD820x_IP_mapping*** . You can utilize multiple devices with different card ID within one application; to do so, simply pass the appropriate **CardID** to each function.

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 EMD820x API. Read the following sections that apply to your programming language.

Note: Be sure to include the declaration functions of EMD820x prototypes by including the appropriate EMD820x header file in your source code. Refer to Chapter 4 EMD820x 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 = EMD820x_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 = EMD820x_port_read ( CardID, port, &data);
```

6.3.2 Visual basic

The file EMD820x.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 EMD820x.bas, do not use the numerical values.
In Visual Basic, you can add the entire EMD820x.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 EMD820x.bas file for your project in Visual Basic 4.0, go to the **File** menu and select the **Add File...** **option**. Select EMD820x.bas, which is browsed in the EMD820x \ api directory. Then, select **Open** to add the file to the project.

To add the EMD820x.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** EMD820x.bas, which is in the EMD820x \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 EMD820xbc.lib EMD820x.dll

Then add the **EMD820xbc.lib** to your project and add

#include “EMD820x.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 = EMD820x_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 = EMD820x_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. Flow chart of application implementation

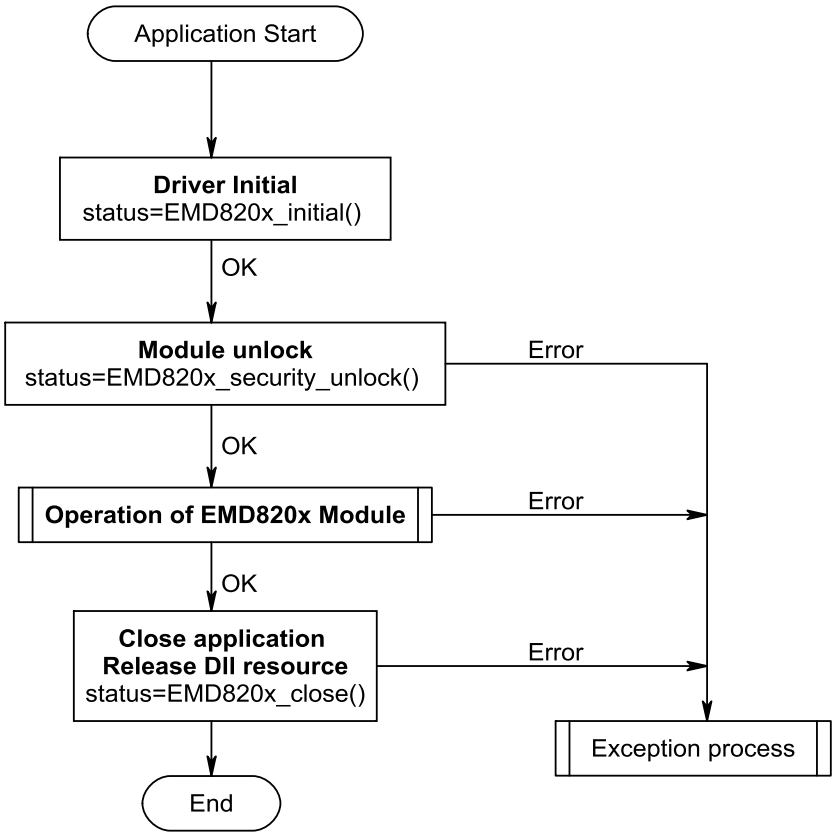


fig. 7.1 main flow

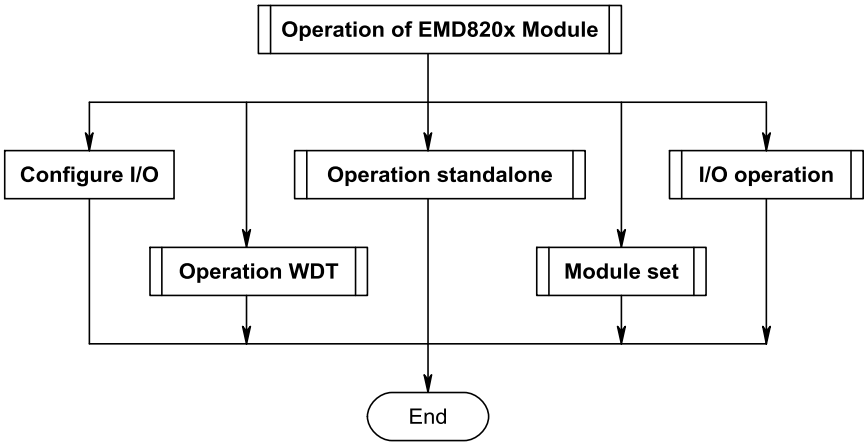


fig. 7.2 various operation modules

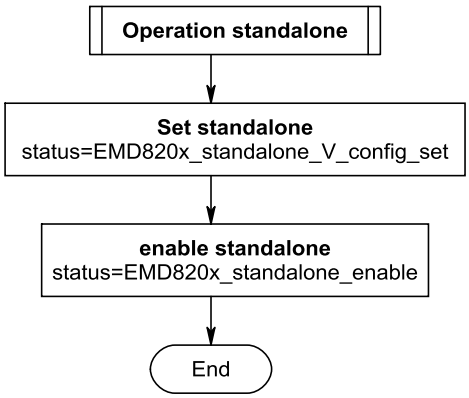


fig. 7.3 standalone operation modules

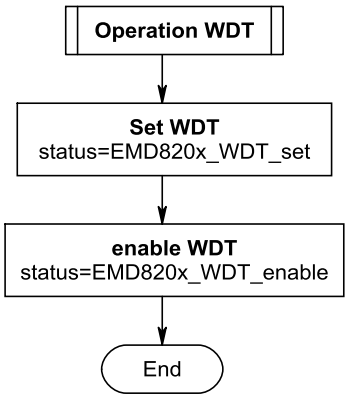


fig. 7.4 WDT operation modules

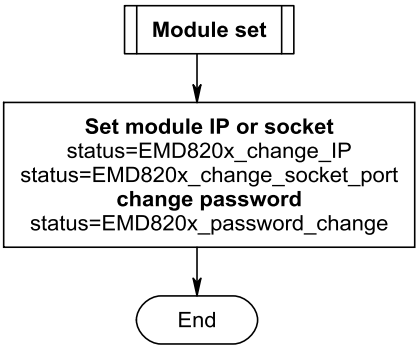


fig. 7.5 IP, socket port and password setting

8. Software overview and dll function

These topics describe the features and functionality of the EMD820x module and the detail of the dll function.

8.1 Initialization and close

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

EMD820x_initial( ) will do.

Once you want to close your application, call

EMD820x_close( ) to release all the resource.

To check the firmware version,

EMD820x_firmware_version_read( ) will do.

To check the dll version,

EMD820x_dll_version_read( )

To change the subnet mask use

EMD820x_subnet_mask_set( ) and read use

EMD820x_subnet_mask_read( )

- **EMD820x_initial**

Format : u32 status =EMD820x_initial (u32 CardID,u8 IP_Address[4] ,
u16 host_port, u16 remote_port, u16 TimeOut_ms,
u8 *module_type)

Purpose: To map IP and PORT of an existing EMD820x to a specified CardID **number.**

Parameters:

Input:

Name	Type	Description
CardID	u32	0~1999 Assign CardID to the EMD820x of a corresponding IP address.
IP_Address[4]	u8	4 bytes 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 PC 0: assign by os Default:15120
remote_port	u16	Assign a communicate port of EMD820x Default:6936
TimeOut_ms	u16	Assign the max delay time of EMD820x response message,1000~10000 ms.

Output:

Name	Type	Description
module_type	u8	Get the Module Type of EMD820x 1: 4 in / 4 out (EMD8204) 2: 8 in / 8 out (EMD8208)

- **EMD820x_close**

Format : u32 status =EMD820x_close (u32 CardID)

Purpose: Release the EMD820x resource when closing the Windows applications.

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMD820x_initial

- **EMD820x_firmware_version_read**

Format : u32 status =EMD820x_firmware_version_read(u32 CardID, u8 Version[2])

Purpose: Read the firmware version.

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMD820x_initial

Output:

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

- **EMD820x_dll_version_read**

Format : u32 status =EMD820x_dll_version_read(u8 *maj_ver,u8 *sub_ver)

Purpose: Read dll version.

Parameters:

Output:

Name	Type	Description
maj_ver	u8	major version
sub_ver	u8	sub version

- **EMD820x_subnet_mask_set**

Format : **u32 status =EMD820x_subnet_mask_set(u32 CardID,u8 subnet_mask[4])**

Purpose: set the subnet mask

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMD820x_initial
subnet_mask[4]	u8	4 byte of subnet mask, Default:255.255.255.0 For example: if subnet_mask is “255.255.255.0” then subnet_mask[0]=255 subnet_mask[1]=255 subnet_mask[2]=255 subnet_mask[3]=0

- **EMD820x_subnet_mask_read**

Format : **u32 status =EMD820x_subnet_mask_read(u32 CardID,u8 subnet_mask[4])**

Purpose: read the subnet mask

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMD820x_initial

Output:

Name	Type	Description
subnet_mask[4]	u8	4 byte of subnet mask, Default:255.255.255.0 For example: if subnet_mask is “255.255.255.0” then subnet_mask[0]=255 subnet_mask[1]=255 subnet_mask[2]=255 subnet_mask[3]=0

8.2 Input/Output function

Physical port

Physical port is the port supports real input or output function.

On physical port 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'.

Virtual port

Virtual port is the port that has input or output data but do not have the real input or output. But you can map the real input or real output to it. The virtual ports can also map as real input or output of another modules. The module has 2 virtual ports (16bits). (Refer the 8.7Standalone function)

To set the polarity (physical port only) setting by

EMD820x_port_polarity_set( )

To read back the input and output polarity (physical port only) setting by

EMD820x_port_polarity_read( )

To control the output, use

EMD820x_port_set( )

To read input or output register status use

EMD820x_port_read( )

To set the point polarity (physical port only) setting by

EMD820x_point_polarity_set( )

To read back the point of input and output polarity (physical port only) setting by

EMD820x_point_polarity_read( )

To control a point of output, use

EMD820x_point_set( )

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

EMD820x_point_read( )

Note: EMD8204 has only IN0 ~ IN3 and OUT0 ~ OUT3;

EMD8208 has only IN0 ~ IN7 and OUT0 ~ OUT7.

Any command over the hardware restriction is not available.

Both modules have the 2 virtual port, Virtual IO17~IO00

- **EMD820x_port_polarity_set**

Format : u32 status =EMD820x_port_polarity_set(u32 CardID, u8 port, u8 polarity)

Purpose: Set polarity of input port or output port.

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMD820x_initial
port	u8	port number 0: OUT PORT 1: IN PORT
polarity	u8	polarity of designated port b7: 0: normal polarity of IN7 or OUT7 1: invert polarity of IN7 or OUT7 … b0: 0: normal polarity of IN0 or OUT0 1: invert polarity of IN0 or OUT0 **(only OUT3~OUT0 and IN3~IN0 valid for EMD8204)**

- **EMD820x_port_polarity_read**

Format : u32 status =EMD820x_port_polarity_read(u32 CardID, u8 port, u8 * polarity)

Purpose: Read polarity of input port or output port.

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMD820x_initial
port	u8	port number 0: OUT PORT 1: IN PORT

Output:

Name	Type	Description
polarity	u8	polarity of designated port b7: 0: normal polarity of IN7 or OUT7 1: invert polarity of IN7 or OUT7 … b0: 0: normal polarity of IN0 or OUT0 1: invert polarity of IN0 or OUT0 **(only b3~b0 valid for EMD8204)**

- **EMD820x_port_set**

Format : u32 status = EMD820x_port_set (u32 CardID , u8 port , u8 data)

Purpose: Set the output or virtual port data.

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMD820x_initial
port	u8	port number 0: OUT PORT 1: null 2: virtual port0 3: virtual port1
data	u8	state of designated port b7: 0: inactive of OUT7 or Virtual IOn7 1: active of OUT7 or Virtual IOn7 … b0: 0: inactive of OUT0 or Virtual IOn0 1: active of OUT0 or Virtual IOn0 **(only OUT3~OUT0 valid for EMD8204)**

Note: The output rely will be close or open depends on the polarity and the state set.

- **EMD820x_port_read**

Format : u32 status = EMD820x_port_read(u32 CardID , u8 port , u8 *data)

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

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMD820x_initial
port	u8	port number 0: OUT PORT 1: IN PORT 2: virtual port0 3: virtual port1

Output:

Name	Type	Description
data	u8	state of designated port b7: 0: inactive of OUT7 or IN7 or Virtual IOn7 1: active of OUT7 or IN7 or Virtual IOn7 … b0: 0: inactive of OUT0 or IN0 or Virtual IOn0 1: active of OUT0 or IN0 or Virtual IOn0 **(only OUT3~OUT0 and IN3~IN0 valid for EMD8204)**

- **EMD820x_point_polarity_set**

Format : u32 status =EMD820x_point_polarity_set(u32 CardID, u8 port, u8 point, u8 state)

Purpose: Set polarity of input point or output point.

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMD820x_initial
port	u8	port number 0: OUT PORT 1: IN PORT
point	u8	point number of input or output 7~0 for IN7 ~ IN0 or OUT7 ~ OUT0 **(only OUT3~OUT0 and IN3~IN0 valid for EMD8204)**
state	u8	polarity of designated point 0: normal 1: invert

- **EMD820x_point_polarity_read**

Format : u32 status =EMD820x_point_polarity_read(u32 CardID, u8 port, u8 point, u8 * state)

Purpose: Read polarity of input point or output point.

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMD820x_initial
Port	u8	port number 0: OUT PORT 1: IN PORT
point	u8	point number of inport or outport 7~0 for IN7 ~ IN0 or OUT7 ~ OUT0 **(only OUT3~OUT0 and IN3~IN0 valid for EMD8204)**

Output:

Name	Type	Description
state	u8	polarity of designated point 0: normal 1: invert

- **EMD820x_point_set**

Format : u32 status =EMD820x_point_set(u32 CardID, u8 port, u8 point, u8 state)

Purpose: Set bit status of output point or virtual point

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMD820x_initial
port	u8	port number 0: OUT PORT 2: virtual port0 3: virtual port1
point	u8	Output point number 0~7 for OUT0~OUT7 or VIOn0~VIOn7 **(only OUT3~OUT0 valid for EMD8204)**
state	u8	state of designated point 0: inactive 1: active

Note: The output rely will be close or open depends on the polarity and the state set.

- **EMD820x_point_read**

Format : u32 status =EMD820x_point_read(u32 CardID, u8 port, u8 point, u8 *state)

Purpose: Read bit state of input, output and virtual I/O point.

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMD820x_initial
port	u8	port number 0: OUT PORT 1: IN PORT 2: virtual port0 3: virtual port1
point	u8	point number of input or output 0~7 for IN0 ~ IN7 or OUT0 ~ OUT7 or VIOn0~VIOn7 **(only OUT3~OUT0 and IN3~IN0 valid for EMD8204)**

Output:

Name	Type	Description
state	u8	state of designated point 0: inactive 1: active

8.3 Counter function

You can use the digital input as a low speed counter (no more than 100pps @ 50% duty). First you can set which input channel you will want to work as counter by:

EMD820x_counter_mask_set( ) then enable the function by

To read the mask value by

EMD820x_counter_mask_read( )

EMD820x_counter_enable( ) and any time to stop by

EMD820x_counter_disable( ).

To read the counter value by

EMD820x_counter_read( ) and use

EMD820x_counter_clear( ) to clear counter.

If the digital input speed is faster than 100 pps (50% duty) the counter might be error.

- **EMD820x_counter_mask_set**

Format : u32 status = EMD820x_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 EMD820x_initial
port	u8	unused
channel	u8	b7 ~ b0, b7: 0: IN7 counter disable 1: IN7 counter enable … b0: 0: IN0 counter disable 1: IN0 counter enable **(only IN3~IN0 valid for EMD8204)**

- **EMD820x_counter_mask_read**

Format : u32 status = EMD820x_counter_mask_read(u32 CardID, u8 port, u8 *channel);

Purpose: To read the counter channel mask.

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMD820x_initial
port	u8	unused

Output:

Name	Type	Description
channel	u8	b7 ~ b0, b7: 0: IN7 counter disable 1: IN7 counter enable … b0: 0: IN0 counter disable 1: IN0 counter enable **(only IN3~IN0 valid for EMD8204)**

- **EMD820x_counter_enable**

Format : u32 status = EMD820x_counter_enable(u32 CardID);

Purpose: To enable the counter function.

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMD820x_initial

- **EMD820x_counter_disable**

Format : u32 status = EMD820x_counter_disable(u32 CardID);

Purpose: To disable the counter function.

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMD820x_initial

- **EMD820x_counter_read**

Format : u32 status = EMD820x_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 EMD820x_initial
port	u8	unused

Output:

Name	Type	Description
counter[8]	u32	counter value counter[0] for IN0 … counter[7] for IN7 **(only IN3~IN0 valid for EMD8204)**

- **EMD820x_counter_clear**

Format : u32 status = EMD820x_counter_clear (u32 CardID, u8 port,u8 channel);

Purpose: To reset the counter value.

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMD820x_initial
port	u8	unused
channel	u8	b7~b0, b7: 0: no function 1: clear IN7 counter … b0: 0: no function 1: clear IN0 counter **(only IN3~IN0 valid for EMD8204)**

8.4 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:

EMD820x_change_socket_port( ) *1

To change IP,

EMD820x_change_IP( ) *1

To reboot EMD820x module for module alarm or to validate the system configuration change by:

EMD820x_reboot( ) *1

*1: Normally the module needs to spend about 10s to restart.

- **EMD820x_change_socket_port**

Format : u32 status = EMD820x_change_socket_port(u32 CardID,u16 module_port);

Purpose: To change the communicate port number of EMD820x.

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

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMD820x_initial
module_port	u16	The new port number to be set to the module 1~65535 but suggest to exclude the traditional default ports.

- **EMD820x_change_IP**

Format : **u32 status = EMD820x_change_IP(u32 CardID, u8 module_IP[4]);**

Purpose: To change the communicate IP of EMD820x.

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

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMD820x_initial
module_IP[4]	u8	The new IP to be set to the module 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

- **EMD820x_reboot**

Format : **u32 status = EMD820x_reboot(u32 CardID);**

Purpose: To reboot EMD820x (wait about 10s to reset).

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMD820x_initial

8.5 Software key function

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

To operate the EMD820x, you must unlock the module first by

EMD820x_security_unlock( )

To verify the lock status by

EMD820x_security_status_read( )

You can change password for your convenience by

EMD820x_password_change( )

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

EMD820x_password_set_default( ) [*1]

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

- **EMD820x_security_unlock**

Format : u32 status = EMD820x_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 EMD820x_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'};

- **EMD820x_security_status_read**

Format : u32 status = EMD820x_security_status_read(u32 CardID,u8 *lock_status);

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

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMD820x_initial

Output:

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

- **EMD820x_password_change**

Format : u32 status = EMD820x_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 EMD820x_initial
Oldpassword[8]	u8	The previous password
password[8]	u8	The new password to be set

- **EMD820x_password_set_default**

Format : u32 status = EMD820x_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 EMD820x_initial default : password[8] = {'1','2','3','4','5','6','7','8'};

8.6 WDT (watch dog timer)

In the industrial environment, we want the controller work as stable as possible but we are not God; we cannot always put the controller by guess. To ensure the controller will not harm to the system or human, people always put a WDT to monitor the controller, if the controller runs in abnormal state the system will fail to reset WDT then WDT will latch the system to prevent further harm. EMD820x also provide the WDT function, which will detect the Ethernet connection state, once the connection is fail for a predefined period, the module will output the predefined status to the ports. You can enable or disable as your application required.

Use ***EMD820x_WDT_set( )*** to set up the WDT timer and the output state if the Ethernet connection fail to communicate.

EMD820x_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:

EMD820x_WDT_enable( ) and disable by:

EMD820x_WDT_disable( ).

- **EMD820x_WDT_set**

Format : u32 status = EMD820x_WDT_set(u32 CardID,u16 time,u8 state)

Purpose: Set WDT (watch dog timer) configuration.

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMD820x_initial
time	u16	Set the WDT wait time.(10~10000) based on 0.1 sec time base. default: 10 (1s)
state	u8	Set the output default state while the connection failed. state: OUT7~OUT0 predefined state while connection fail. **(only OUT3~OUT0 valid for EMD8204)**

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

- **EMD820x_WDT_read**

Format : u32 status = EMD820x_WDT_read (u32 CardID, u16 *time, u8 *state, u8 *enable)

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

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMD820x_initial

Output:

Name	Type	Description
time	u16	read the WDT wait time.
state	u8	state: OUT7~OUT0 predefined state while connection fail. **(only OUT3~OUT0 valid for EMD8204)**
enable	u8	0: disable 1: enable

- **EMD820x_WDT_enable**

Format : u32 status = EMD820x_WDT_enable(u32 CardID)

Purpose: enable WDT (watch dog timer).

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMD820x_initial

- **EMD820x_WDT_disable**

Format : u32 status = EMD820x_WDT_disable(u32 CardID)

Purpose: disable WDT (watch dog timer).

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMD820x_initial

8.7 Standalone function

The above mentioned functions are majorly support for master slave control model which needs the Ethernet always active. The standalone mode is the application extension of EMD820x module; it can work as I/O controller with or without the Ethernet existing.

The standalone mode consists of 3 function blocks: input configuration, control block (input to output relationship) and output configuration.

8.7.1 Input configuration

Physical input, the input of the module to which external circuit to connect. For example: IN7 ~IN0 for EMD8208 and IN3~IN0 for EMD8204.

Virtual input, the logic layer input, no external wire to connect.

Inputs can be masked to select the desire points (both Physical or Virtual input) and state to trigger control. Standalone mode startup and communication broken are the two special input triggers. (Fig 8.7.1 Standalone mode function block diagram) The startup input is an always true input while the module power up and ready to run standalone mode.

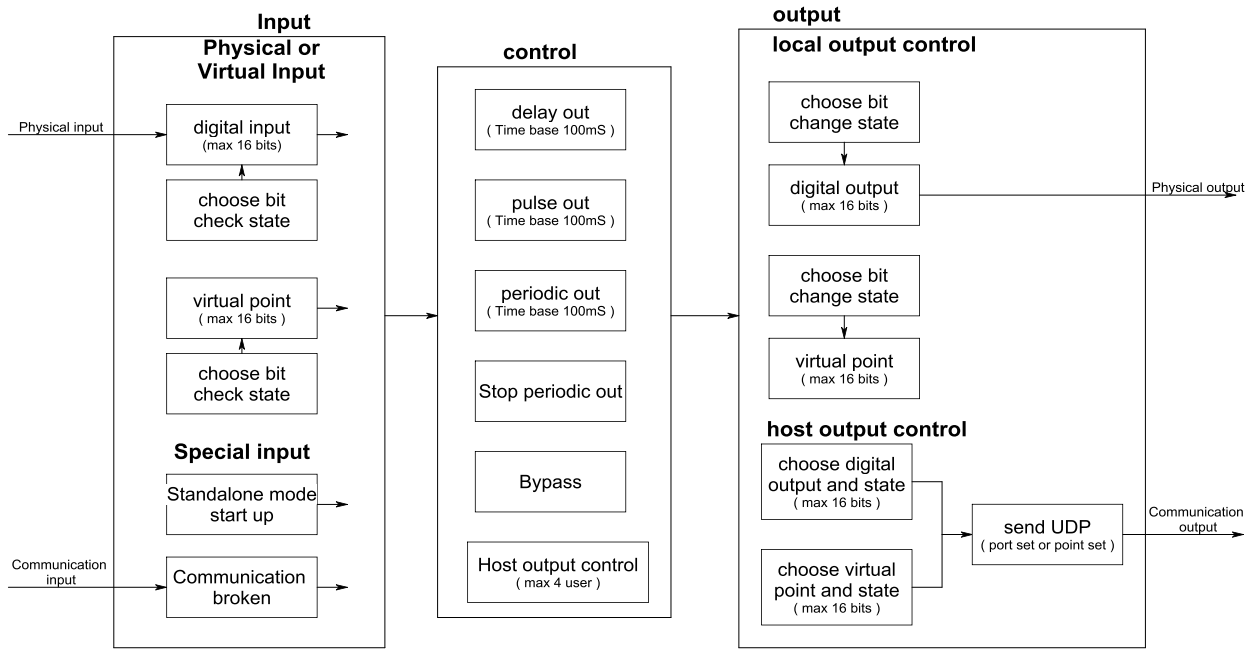


fig 8.7.1 Standalone mode function block diagram

8.7.2 Control block

There are several control modes to generate output:

- If working in delay mode, the output will not trigger until the timer time up.
- If working in pulse mode, the output will trigger immediately while the input condition meets but inactive while timer time up.
- If working in periodic mode, the output will toggle at each timer time up then timer reloads

itself for next toggle until stop periodic out (timer off).

- If working in bypass mode, there is no interaction of timer to generate output (output immediately).
- If working in send to remote output control mode, the output will send to the remote module via Ethernet. (refer 8.8 Remote output control (Device to device control))

8.7.3 Output block

Physical output, the output of the module is used to control the external devices. For EMD8204 OUT3~OUT0 are physical output and EMD8208 are OUT7~OUT0.

Virtual output, the logic layer output, they are common to Virtual input. (i.e. the virtual inputs are also virtual outputs)

If we use the device to device function, the outputs maybe the remote module's physical or virtual outputs. The output mode can be active or inactive and toggle, if toggle mode, the output will change state each tine the logical result of the input is true.

8.7.4 Standalone mode startup

When the module is programmed in standalone mode, a virtual startup input will be true at module ready to run standalone mode. The typical application, say, you want to generate a square wave. Now program the control mode as "Startup Input action and periodic out", you will get a square wave at designated point while the module works in standalone mode.

8.7.5 Communication broken

A special status bit called communication broken can be used to detect the PC to module or module to module communication. It can be used to generate an output or a device to device control (refer. 8.8 Remote output control (Device to device control))

8.7.6 DLL of standalone function

If you have preprogrammed step sequence command in the module, you can command it to start by:

EMD820x_standalone_enable( ) and stop by

EMD820x_standalone_disable( )

There are total 32 steps can be programmed, to program the step sequence command,

EMD820x_standalone_V_config_set( ) and you can read back for verification by

EMD820x_standalone_V_config_read( )

To clear all the saved step sequence command of the module,

EMD820x_standalone_config_clear( )

- **EMD820x_standalone_enable**

Format : u32 status =EMD820x_standalone_enable(u32 CardID)

Purpose: Enable standalone mode.

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMD820x_initial

- **EMD820x_standalone_disable**

Format : u32 status =EMD820x_standalone_disable(u32 CardID)

Purpose: Disable (stop) standalone mode.

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMD820x_initial

- **EMD820x_standalone_V_config_set**

Format : u32 status =EMD820x_standalone_V_config_set(u32 CardID,u8 step_number, _V_StandaloneData data[32], u8 standalone_state)

Purpose: To configure the step command.

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMD820x_initial
step_number	u8	Number of process steps

data[32]	_V_StandaloneData	struct _V_StandaloneData{
		u32 timer_value;
		u8 in_virtual_point_bit[2];
		u8 in_virtual_state_bit[2];
		u8 in_point_bit[2];
		u8 in_state_bit[2];
		u8 remote_detection [4];
		u8 control_mode;
		u8 out_mode;
		u8 out_virtual_point_bit[2];
		u8 out_point_bit[2];
		}

		timer_value
		// timer tick is 100ms per tick
		// control_mode = delay or periodic out mode
		// setting value is delay timer
		// control_mode = pulse out
		// setting value is active time of pulse
		// if timer_value = 10 ,
		// the delay time is 10 * 100 ms = 1000 ms
		
		in_virtual_point_bit: select the virtual point which you want to watch the state
		//[0].b7 ~ b0 = virtual IO07 ~ virtual IO00
		//[1].b7 ~ b0 = virtual IO17 ~ virtual IO10
		
		in_virtual_state_bit
		//set virtual IO state
		//0: inactive, 1:active
		//[0].b7 ~ b0 = virtual IO07 ~ virtual IO00
		//[1].b7 ~ b0 = virtual IO17 ~ virtual IO10
		
		in_point_bit: select the input point which you want to watch the state.
		//[0].b7 ~ b0 = IN7 ~ IN0
		//[1].b7 ~ b0 = unused,
		// **(only IN3~IN0 valid for EMD8204)**
		
		in_state_bit:
		//set input state
		//0: inactive, 1:active
		//[0].b7 ~ b0 = IN7 ~ IN0
		//[1].b7 ~ b0 = unused,
		// **(only IN3~IN0 valid for EMD8204)**
		
		remote_detection
		//detection the connection status from remote to
		//the module
		//1: enable 0: disable
		//[0] : detect remote device0
		//[1] : detect remote device1
		//[2] : detect remote device2

		//[3] : detect remote device3
		
		control_mode
		// 0x0 = Bypass,
		// 0x1 = Input action and delay out
		// 0x2 = Input action and pulse out
		// 0x3 = Input action and periodic out
		//0x4 = Startup action and delay out
		// 0x5 = Startup action and pulse out
		// 0x6 = Startup action and periodic out
		// 0x7 = Stop periodic out
		// 0x10 = send to remote device0
		// 0x11 = send to remote device1
		// 0x12 = send to remote device2
		// 0x13 = send to remote device3
		
		out_mode
		// control_mode = Bypass,
		// 0x0= INACTIVE, 0x1= ACTIVE,
		// 0x2=Change state(toggle)
		// control_mode = delay out,
		// 0x0= INACTIVE, 0x1= ACTIVE,
		// 0x2=Change state
		// control_mode = pulse out,
		// 0x0 = L_PULSE, 0x1 = H_PULSE
		// control_mode = periodic out,
		// 0x2 = Change state
		// control_mode = remote control,
		// 0x0= INACTIVE, 0x1= ACTIVE
		
		out_virtual_point_bit
		//[0].b7 ~ b0 = Virtual IO07 ~ Virtual IO00
		//[1].b7 ~ b0 = Virtual IO17 ~ Virtual IO10
		//virtual output may be on the module itself or the
		// remote device. If **control_mode** is
		// configured as send to remote device.
		//The output will be on the remote.
		
		out_point_bit
		//[0].b7 ~ b0 = OUT07 ~ OUT00

		//[1].b7 ~ b0 = OUT17 ~ OUT10
		//output may be on the module itself or the
		// remote device. If **control_mode** is
		// configured as send to remote device. The
		// output will be on the remote.
		//The point will be limited by the designated
		//module.

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

Note:

1. The Standalone Data is an array of 32 elements in which each element is a step command of process. Each time you configure, you must prepare the 32 elements. If the command data is null (all elements are "0" in any of the 32 elements), the controller will take it as end of process.
2. Although the Standalone Data consist of 32 (max) elements, you also need to specify the number of the elements (steps) to accelerate the speed of instruction loading process (if less than 32, it will spend less time).
3. Standalone_state is used for configuration the function after the power-on. If standalone_state=1, after power on, the controller will run the pre-programmed command until it is commanded to stop from Ethernet interface or power off.
4. If the returned code is error (any number except for 0), you must try to re-download until the returned code is no error (or by any means to recover it) to confirm the correct standalone working.

Special note on periodic output and stop periodic output

If the output is configured as periodic out, there are three possible situations to stop the output.

1. Stop by module power off.
2. Stop by disable standalone mode.
3. Stop by command with control mode 'stop periodic out' that follows the periodic output.

At the stop periodic out, the output state will fixed at inactive.

- **EMD820x_standalone_V_config_read**

Format : u32 status =EMD820x_standalone_V_config_read(u32 CardID,
u8 *step_number, _V_StandaloneData data[32],
u8 *standalone_state, u8 *enable)

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

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMD820x_initial

Output:

Name	Type	Description
step_number	u8	Number of process steps

data[32]	_V_StandaloneData	struct _V_StandaloneData{
		u32 timer_value;
		u8 in_virtual_point_bit[2];
		u8 in_virtual_state_bit[2];
		u8 in_point_bit[2];
		u8 in_state_bit[2];
		u8 remote_detection [4];
		u8 control_mode;
		u8 out_mode;
		u8 out_virtual_point_bit[2];
		u8 out_point_bit[2];
		}
		timer_value
		// timer tick is 100ms per tick
		// control_mode = delay or periodic out mode
		// setting value is delay timer
		// control_mode = pulse out
		// setting value is active time of pulse
		// if timer_value = 10 ,
		// the delay time is 10 * 100 ms = 1000 ms
		
		in_virtual_point_bit: select the virtual point which you want to watch the state
		//[0].b7 ~ b0 = virtual IO07 ~ virtual IO00
		//[1].b7 ~ b0 = virtual IO17 ~ virtual IO10
		
		in_virtual_state_bit

		//set virtual IO state
		//0: inactive, 1:active
		//[0].b7 ~ b0 = virtual IO07 ~ virtual IO00
		//[1].b7 ~ b0 = virtual IO17 ~ virtual IO10
		
		in_point_bit: select the input point which you want to watch the state.
		//[0].b7 ~ b0 = IN7 ~ IN0
		//[1].b7 ~ b0 = unused,
		// **(only IN3~IN0 valid for EMD8204)**
		
		in_state_bit:
		//set input state
		//0: inactive, 1:active
		//[0].b7 ~ b0 = IN7 ~ IN0
		//[1].b7 ~ b0 = unused,
		// **(only IN3~IN0 valid for EMD8204)**
		
		remote_detection
		//detection the connection status from remote to
		//the module
		//1: enable 0: disable
		//[0] : detect remote device0
		//[1] : detect remote device1
		//[2] : detect remote device2
		//[3] : detect remote device3
		
		control_mode
		// 0x0 = Bypass,
		// 0x1 = Input action and delay out
		// 0x2 = Input action and pulse out
		// 0x3 = Input action and periodic out
		//0x4 = Startup action and delay out
		// 0x5 = Startup action and pulse out
		// 0x6 = Startup action and periodic out
		// 0x7 = Stop periodic out
		// 0x10 = send to remote device0
		// 0x11 = send to remote device1
		// 0x12 = send to remote device2
		// 0x13 = send to remote device3

		out_mode
		// control_mode = Bypass,
		// 0x0= INACTIVE, 0x1= ACTIVE,
		// 0x2=Change state(toggle)
		// control_mode = delay out,
		// 0x0= INACTIVE, 0x1= ACTIVE,
		// 0x2=Change state
		// control_mode = pulse out,
		// 0x0 = L_PULSE, 0x1 = H_PULSE
		// control_mode = periodic out,
		// 0x2 = Change state
		// control_mode = remote control,
		// 0x0= INACTIVE, 0x1= ACTIVE
		
		out_virtual_point_bit
		//[0].b7 ~ b0 = Virtual IO07 ~ Virtual IO00
		//[1].b7 ~ b0 = Virtual IO17 ~ Virtual IO10
		//virtual output may be on the module itself or the
		// remote device. If **control_mode** is
		// configured as send to remote device. The output
		// will be on the remote.
		
		out_point_bit
		//[0].b7 ~ b0 = OUT07 ~ OUT00
		//[1].b7 ~ b0 = OUT17 ~ OUT10
		//output may be on the module itself or the
		// remote device. If **control_mode** is
		// configured as send to remote device. The output
		// will be on the remote. The point will be limited
		// by the designated module.

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

- **EMD820x_standalone_config_clear**

Format : u32 status =EMD820x_standalone_config_clear(u32 CardID)

Purpose: clear the standalone step program.

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMD820x_initial

8.8 Remote output control (Device to device control)

Remote output control is used for device to device control, i.e. a device can generate a command to trigger the other modules (i.e. remote module).

Let's take 2 modules as example; module A wants to monitor the Ethernet communication from remote device 0 and input from a smoke sensor, if any of the conditions is active, the module A will output alarm on point OUT0 and trigger the module B (maybe far away to generate alarm on pint OUT0). The program is very simple, refer 10.10 Integration example

.

Before you send command to the remote module, the information of the remote module must setup ready by: ***EMD820x_D2D_config_set( )*** and read back for verification by

EMD820x_D2D_config_read( )

To stop and clear the configuration (for new configuration) by

EMD820x_D2D_connection_clear( )

To check the connection status by

EMD820x_D2D_connection_state_read( )

- **EMD820x_D2D_config_set**

Format : u32 status =EMD820x_D2D_config_set(u32 CardID, u8 remote_ID, u8 IP_Address[4] , u8 password[8],u16 remote_port, u8 remote_type)

Purpose: To configure the basic communication information of the remote module.

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMD820x_initial
remote_ID	u8	0~3, assign the ID of remote modules
IP_Address[4]	u8	4 bytes of IP address of remote module (which is already or will be set by **EMD820x_change_IP**) 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
password[8]	u8	the logging password of the remote module
remote_port	u16	Assign a communicate port of the remote module (the remote module is already or will be set by **EMD820x_change_socket_port**) Default: 6936
remote_type	u8	type of remote module // 1 : EMD8204, 2 : EMD8208, // 3 : EMD8216, 4 : EMC8485, // 5 : EMC8432, 6 : EMA8314R, // 7 : EMA8308, 8 : EMA8308D, // 9 : PC

- **EMD820x_D2D_config_read**

Format : u32 status =EMD820x_D2D_config_read(u32 CardID, u8 remote_ID, u8 IP_Address[4] ,u16 *remote_port,u8 *remote_type)

Purpose: To read the basic communication information of the remote module.

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMD820x_initial
remote_ID	u8	0~3

Output:

Name	Type	Description
IP_Address[4]	u8	4 bytes of IP address of remote module (which is already or will be set by **EMD820x_change_IP**) 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
remote_port	u16	The communicate port of the remote module (the remote module is already or will be set by **EMD820x_change_socket_port**) Default: 6936
remote_type	u8	// 1 : EMD8204, 2 : EMD8208, // 3 : EMD8216, 4 : EMC8485, // 5 : EMC8432, 6 : EMA8314R, // 7 : EMA8308, 8 : EMA8308D, // 9 : PC

- **EMD820x_D2D_connection_clear**

Format : u32 status =EMD820x_D2D_connection_clear(u32 CardID,u8 state[4])

Purpose: To clear the connect information. It will disconnect before clear.

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMD820x_initial
state[4]	u8	state[n] : remote n connection state If set to 1: it will clear the connect information and disable the connection.

Note: If the module is in standalone mode enabled state, the connection clear command is valid.

- **EMD820x_D2D_connection_state_read**

Format : u32 status =EMD820x_D2D_connection_state_read(u32 CardID,u8 state[4], u8 error_state[4])

Purpose: Read the connect state.

Parameters:

Input:

Name	Type	Description
CardID	u32	CardID assigned by EMD820x_initial

Output:

Name	Type	Description
state[4]	u8	connection state state[n] : remote n connection state 1:connected 0:broken
error_state[4]	u8	error_state[n] : (n: remote_ID) 1: remote n return error 0: remote n no error.

9. User configuration utility of Standalone mode

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.

9.1 Overview of user configuration utility

-- Open the standalone mode configuration window. EMD820x -> Special -> Standalone.

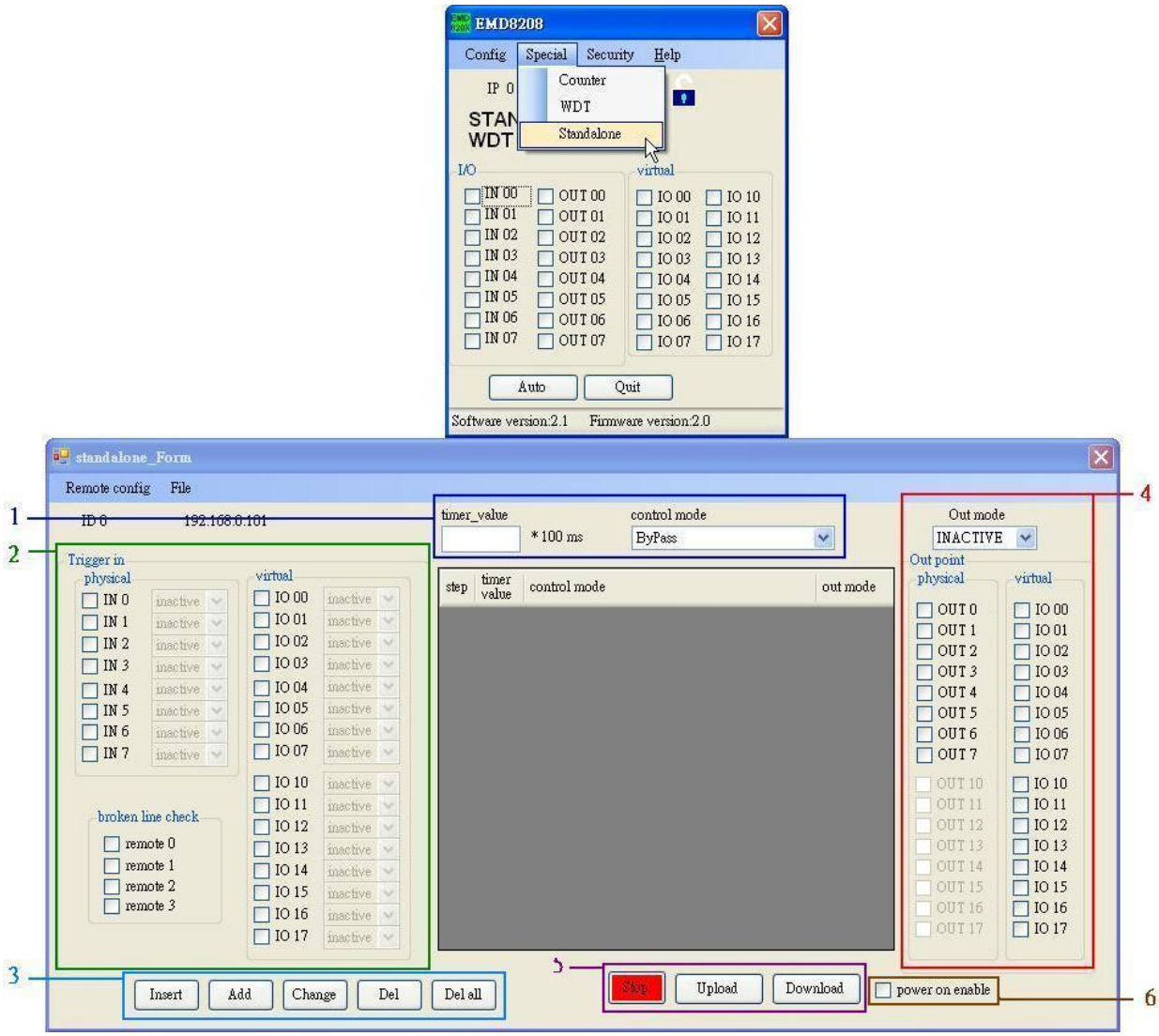


fig. 9.1.1 Standalone mode configuration panel

Basically, a command is consist of input point and its state (on the left side of fig. 9.1.1); next, the control operation mode, time constant (on the middle of fig. 9.1.1) and finally the output mode and output points (on the right side of fig. 9.1.1). The input and output block will update as the current command line highlighted. From the above diagram, you will see

Block1: Control operation mode and time constant setting.

Block2: standalone mode command input configuration.

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

Block4: standalone mode command output configuration.

Block5: standalone mode command upload/download, Run/ Stop.

Block6: power on standalone mode enable/ disable.

9.2 Configure a command

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

-- input configuration

fig 9.2.1 Input block

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

The above diagram shown that if you want the input monitor IN1 active and IN2 inactive as trigger source of the command. You can select and configure any of the inputs to monitor as trigger source.

Input debounce frequency is 100Hz (50% duty), response faster than 100Hz maybe ignore as noise by the EMD820X module.

-- control configuration

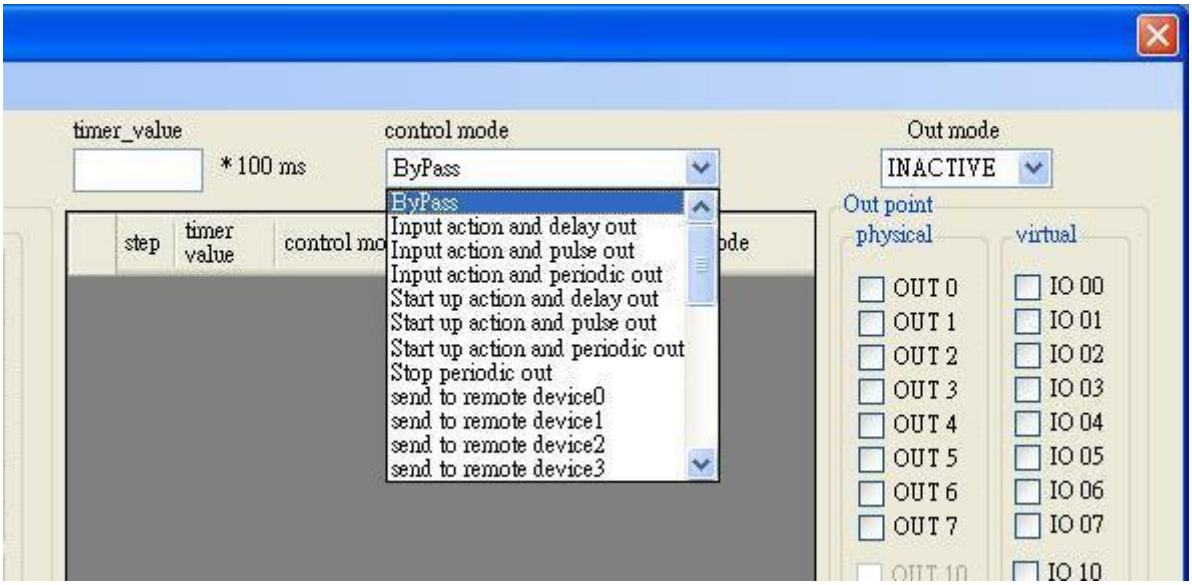


fig. 9.2.2 Control mode block

The control block consists of a timer and its associate functions. If the inputs meet the condition you configured, it will trigger the timer to operate. The control block provides several kinds of working mode:

working mode	explanation
Bypass	bypass the input trigger to output, timer do not work.
Input action and delay out	input triggers the timer to work as delay timer (time up triggers output)
Input action and pulse out	input triggers the timer to work as pulse timer (timing the output duty)
Input action and periodic out	input triggers the timer to work as periodic timer (time up toggles output and reload to run)
Startup action and delay out	startup triggers the timer to work as delay timer (time up triggers output)
Startup action and pulse out	startup triggers the timer to work as pulse timer (timing the output duty)
Startup action and periodic out	startup triggers the timer to work as periodic timer (time up toggles output and reload to run)
Stop periodic out	periodic out is stopped just followed by this command
Send to remote device0	input trigger to remote device0 output, timer do not work.
Send to remote device1	input trigger to remote device1 output, timer do not work.
Send to remote device2	input trigger to remote device2 output, timer do not work.
Send to remote device3	input trigger to remote device3 output, timer do not work.

The timer is based on 100ms time base, less than 100ms or not the multiple of 100ms is impossible to implement.

-- Output configuration

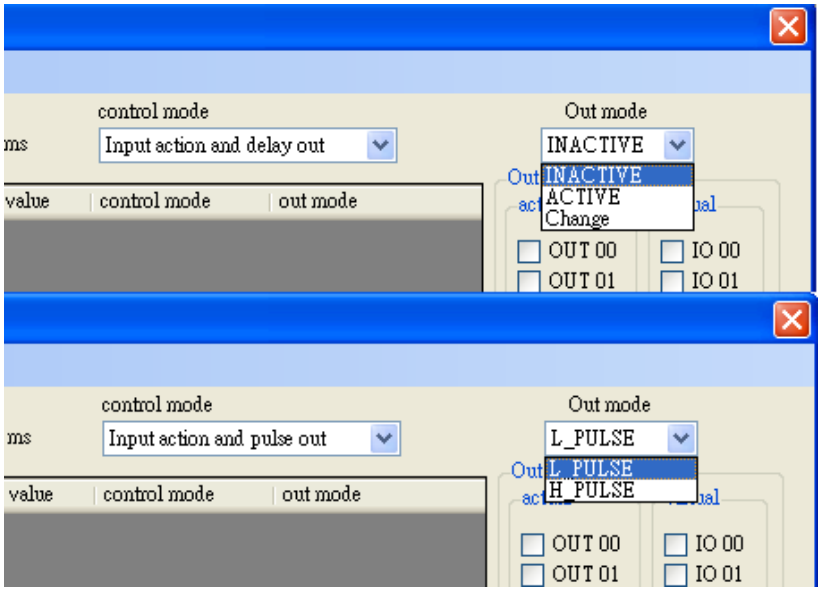


fig. 9.2.3 Out mode block

The control block depends on its working mode to control the output. The output can be configured as inactive, active or toggle.

control mode	output mode	explanation
Bypass	inactive	output inactive level
	active	output active level
	Change	output toggles
Input action and delay out	inactive	output inactive level
	active	output active level
	Change	output toggles
Input action and pulse out	L-pulse	output inactive pulse (‾|__|‾)
	H-pulse	output active pulse (__|‾|__)
Input action and periodic out	Change	output toggles
Startup action and delay out	inactive	output inactive level
	active	output active level
	Change	output toggles
Startup action and pulse out	L-pulse	output inactive pulse (‾|__|‾)
	H-pulse	output active pulse (__|‾|__)
Startup action and periodic out	Change	output toggles
Stop periodic out	none	output reset to its normal state
Send to remote device0 ~ device3	inactive	Remote output inactive
	active	Remote output active

9.3 Remote configure

Each module can control 4 remote modules. To use the device to device communication, you must setup the basic information of each remote module then select control mode to send to remote device to control its output or virtual IO.

-- Open the remote configuration window. EMD820x -> Special -> Standalone -> Remote config.

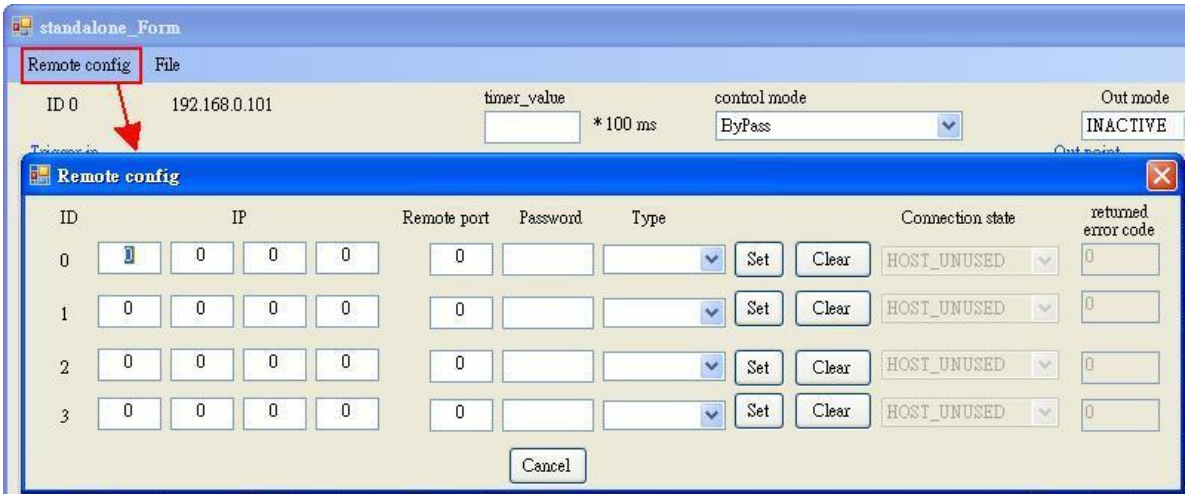


fig. 9.3.1 Remote configuration block

From the above diagram, you will see

ID : remote device index.

IP: remote device IP address.

Remote port: remote device socket port, default is 6936.

Password: remote device connection password, default is “12345678”.

Type: remote device module type.

Connection state: after the above information filled, the demo program will try to connect the remote module and display its connection state in this column.

returned error code: if the remote has connection error, the returned code will be displayed in this column for you to debug. Error code refers Chapt. 13 EMD820x Error codes summary .

9.4 Virtual I/O and broken line state configuration

Virtual IO00 ~ IO17can be input or output since it is the virtual IO port. You can trigger the virtual IO of the remote module and then the remote module takes the virtual IO as input to decide the output with its local inputs or other virtual IO from different modules. The modules on the Ethernet connection will work as a big controller.

To detect the connection of remote modules, the broken line detection function will be easy with the special input; you can monitor the broken line status to generate an alarm or take some action.

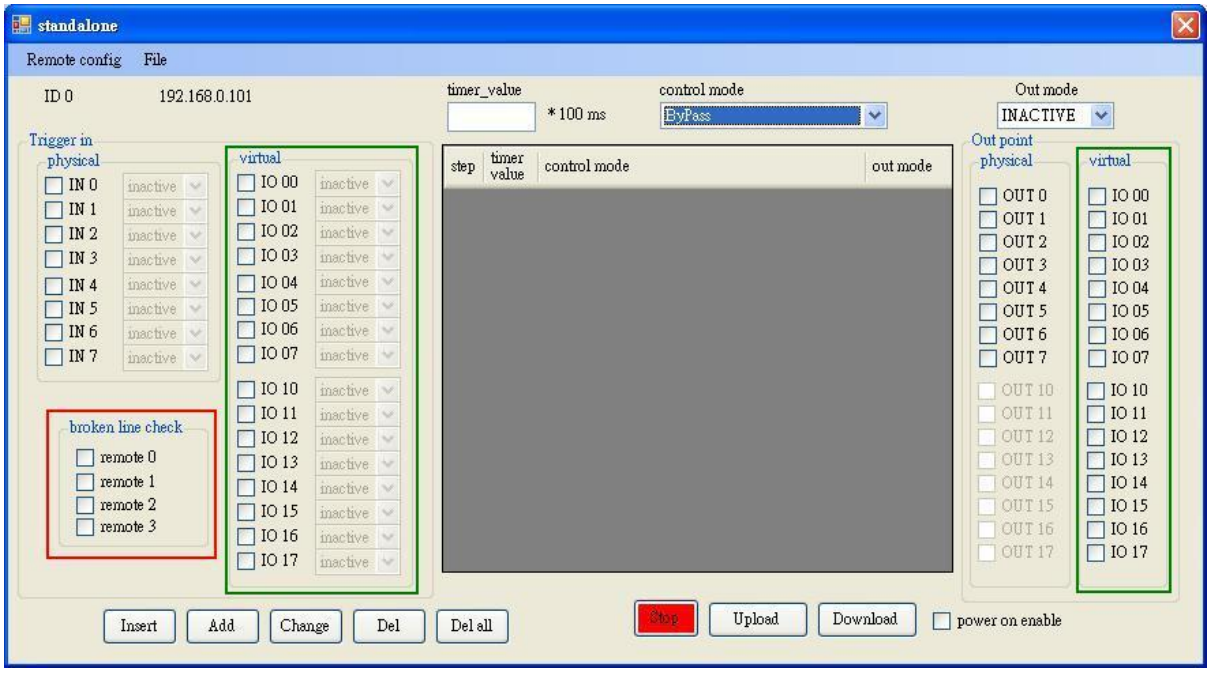


fig. 9.4.1 Virtual IO and broken line block

9.5 Edit function

To provide a good edit environment, some functions of editing are necessary: insert, add, change, delete and delete all are provided.

fig. 9.5.1 Edit function

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

2: Add: add a new command

3: Change: modify the existing command line

4: Del: delete the highlighted command line

5: Del all: clear all the commands

9.6 Power on enable

Power on enable function will determine the module to standby mode or run the stored pre-programmed command steps. If enabled, run the step command, else keep Ethernet connection alive for communication only.

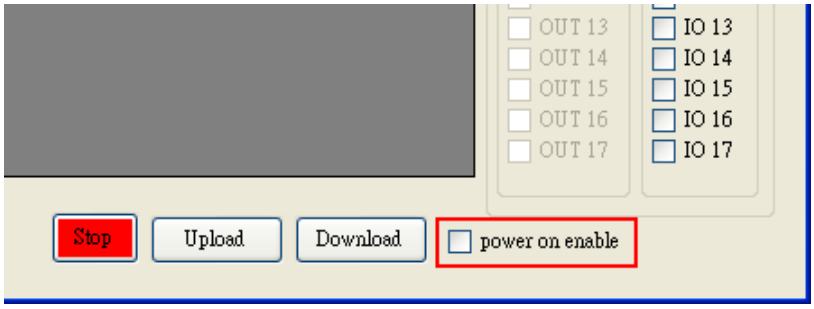


fig. 9.6.1 Power on anable

9.7 Upload program

There are totally 32 step commands can be execute in EMD820x module, after you edit the step command sequence, you can upload to the module to store and execute immediately (click Stop button to Run state) or store it and execute on next power on (select option: power on enable) or command to run via Ethernet.

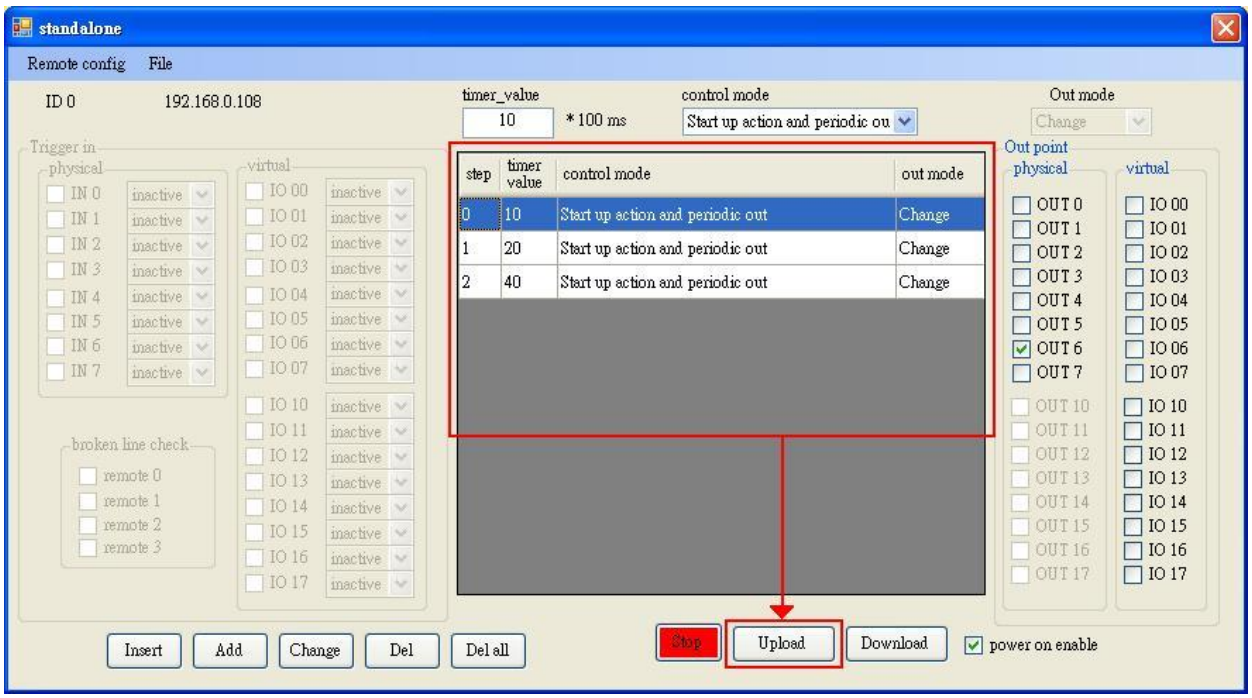


fig. 9.7.1 Upload

9.8 Download program

If you have connected with EMD820x module via Ethernet, you can download the stored program from the module.

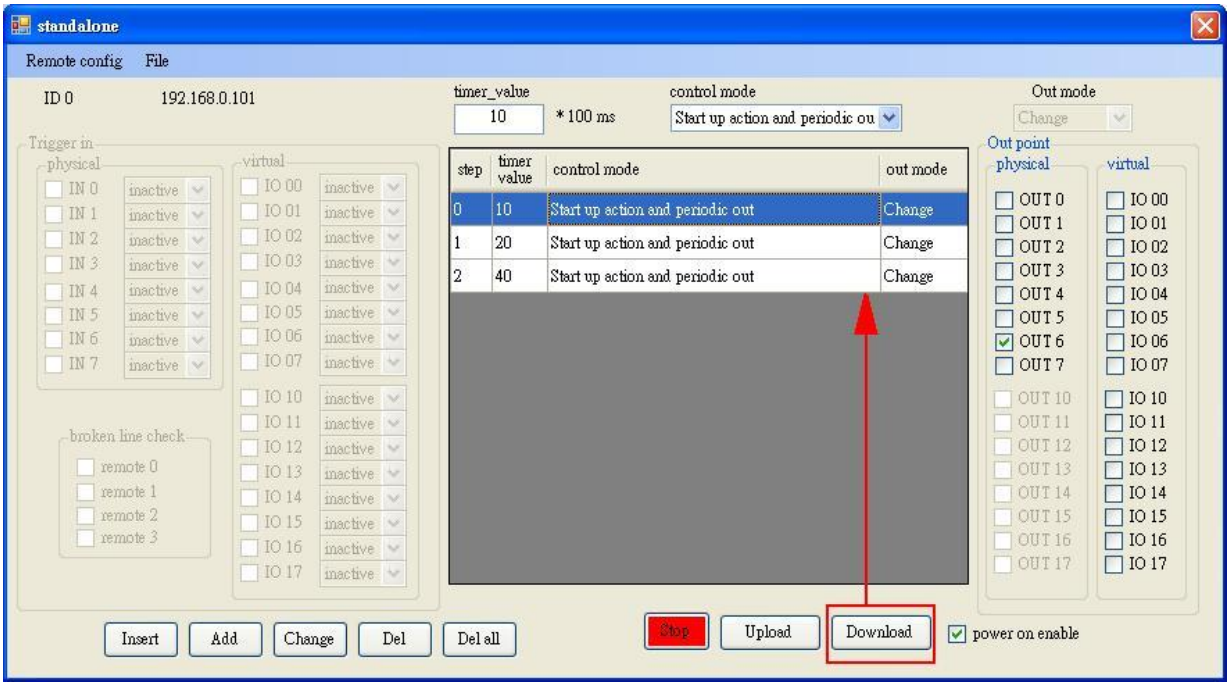


fig. 9.8.1 Download

9.9 Save or 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.

fig. 9.9.1 Save or load program from PC

9.10 Run / Stop standalone function

Standalone mode can Run or Stop by the button as following shown.

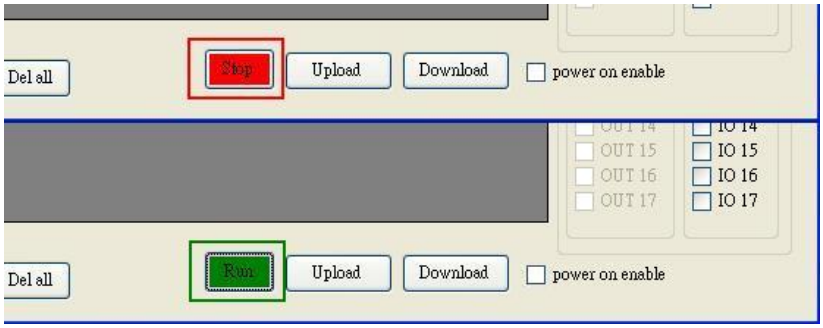


fig. 9.10.1 Run or Stop standlone mode

Whether the module standalone mode is Run or Stop can be verified shown on the main form.

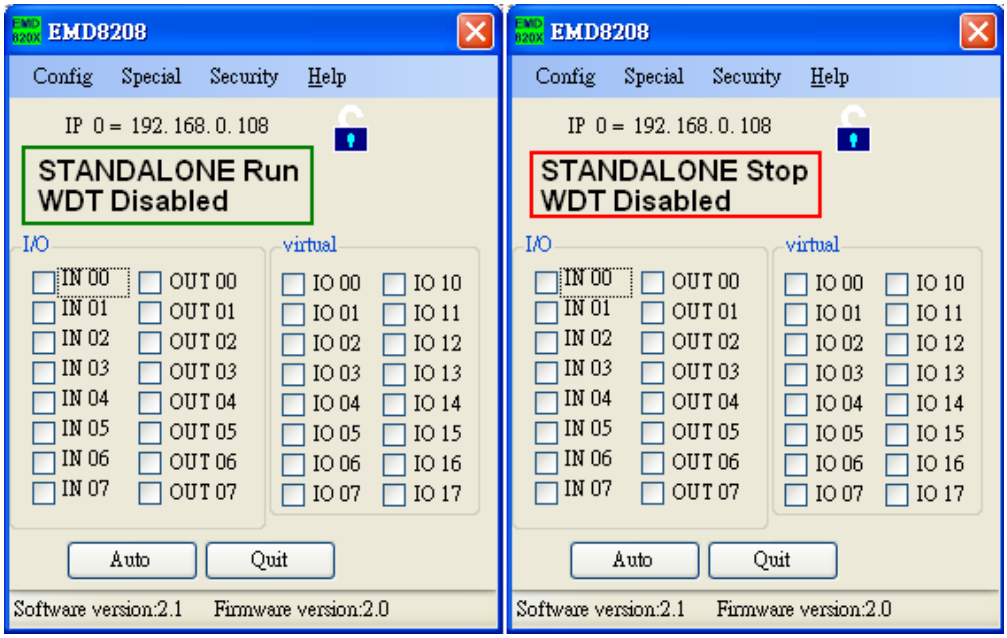


fig. 9.10.2 Run / Stop indication

10. Standalone mode application examples

10.1 Example 1: Monitoring inputs if condition meets, trigger output

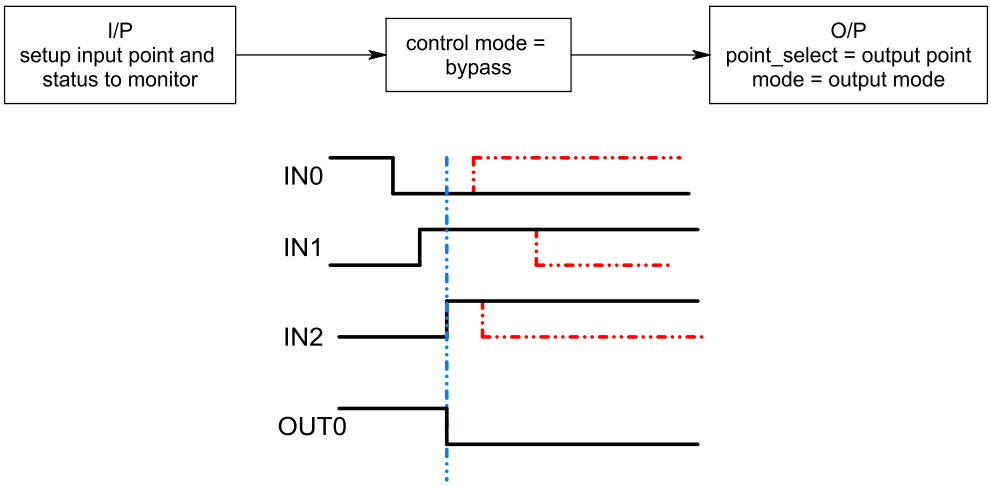


fig. 10.1.1 Example 1, block diagram and timing

Say, you want to watch IN0 inactive, IN1 and IN2 active to trigger output OUT0 to inactive. Program as the following shown.

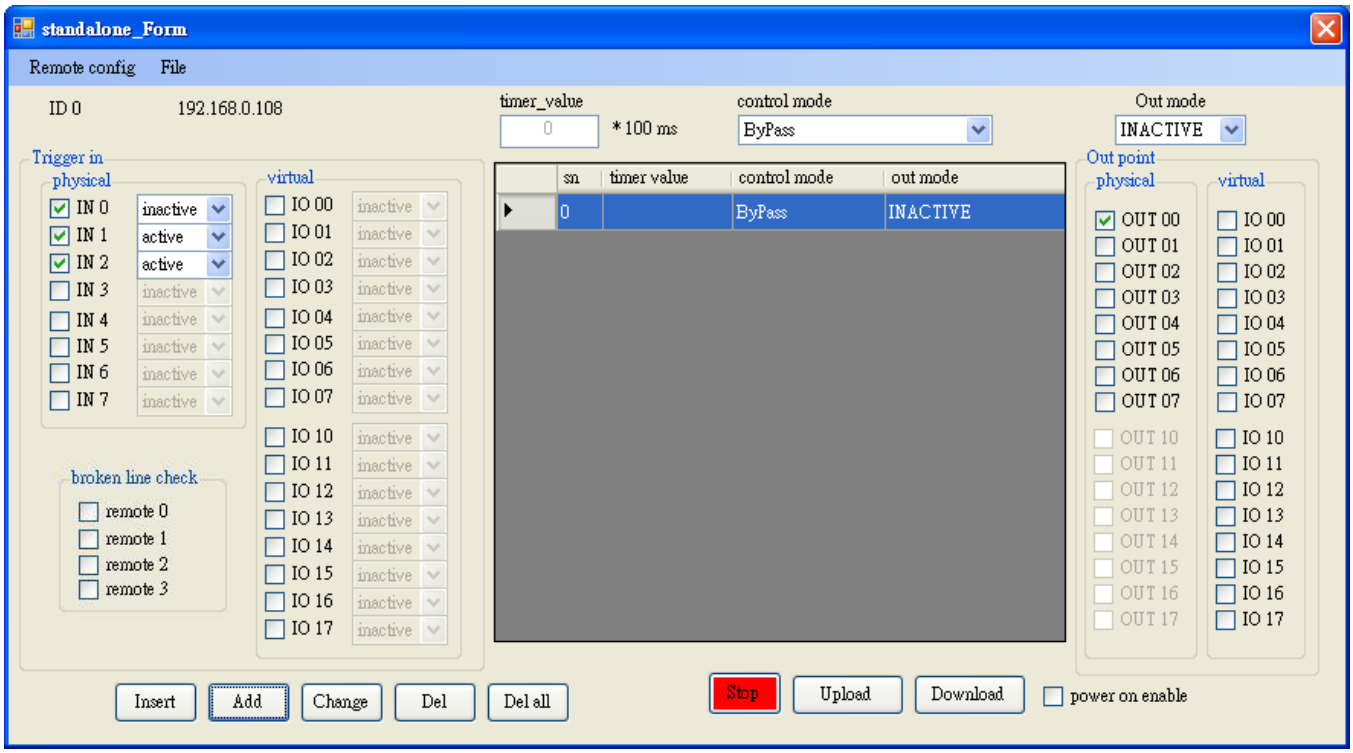


fig. 10.1.2 Example 1 program

Note: The output change state (form active to inactive or inactive to active depends on your program) only if the input condition meet and if not meet, keep the previous state.

10.2 Example 2: Monitoring the inputs if condition meets, delay to trigger output

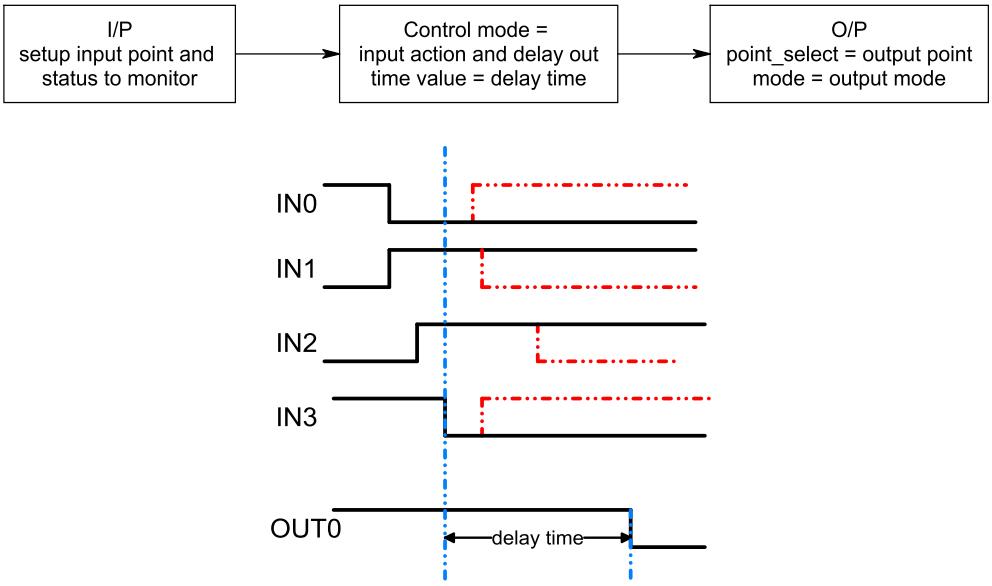


fig. 10.2.1 Example 2, block diagram and timing

Say, you want to watch IN0 and IN3 are inactive and IN1 and IN2 are active to trigger output OUT00 to inactive with delay. Program as the following shown.

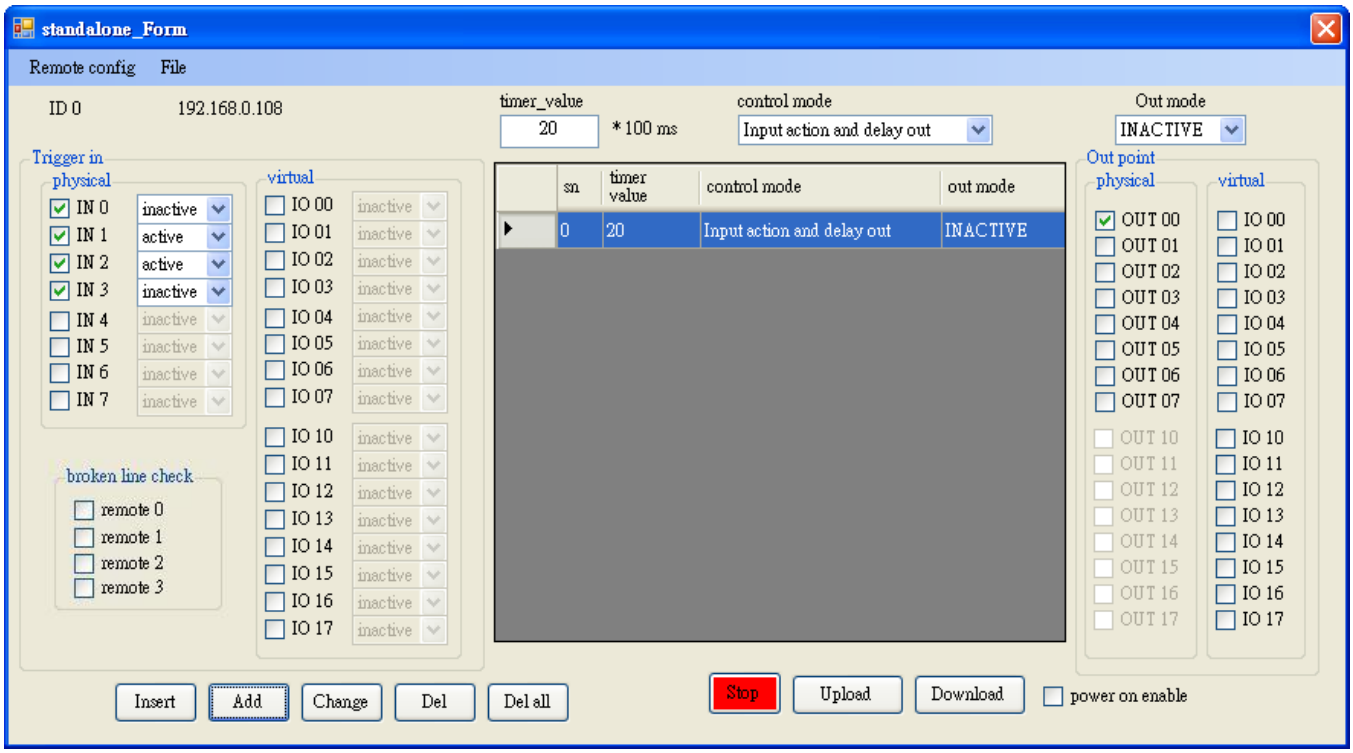


fig. 10.2.2 Example 2 program

Note: While the delay interval is in progress, no further condition meet triggers will be valid.

10.3 Example 3: Monitoring the inputs if condition meets, output pulse

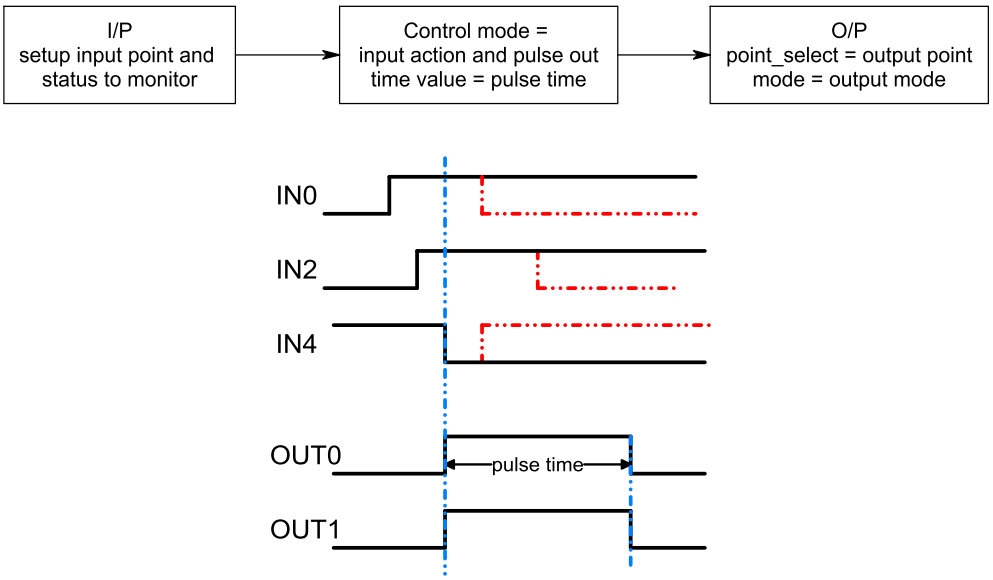


fig. 10.3.1 Example 3, block diagram and timing

Say, you want to watch IN0 and IN2 is active and IN4 is inactive to trigger output OUT0 and OUT1 to H-pulse. Program as the following shown.

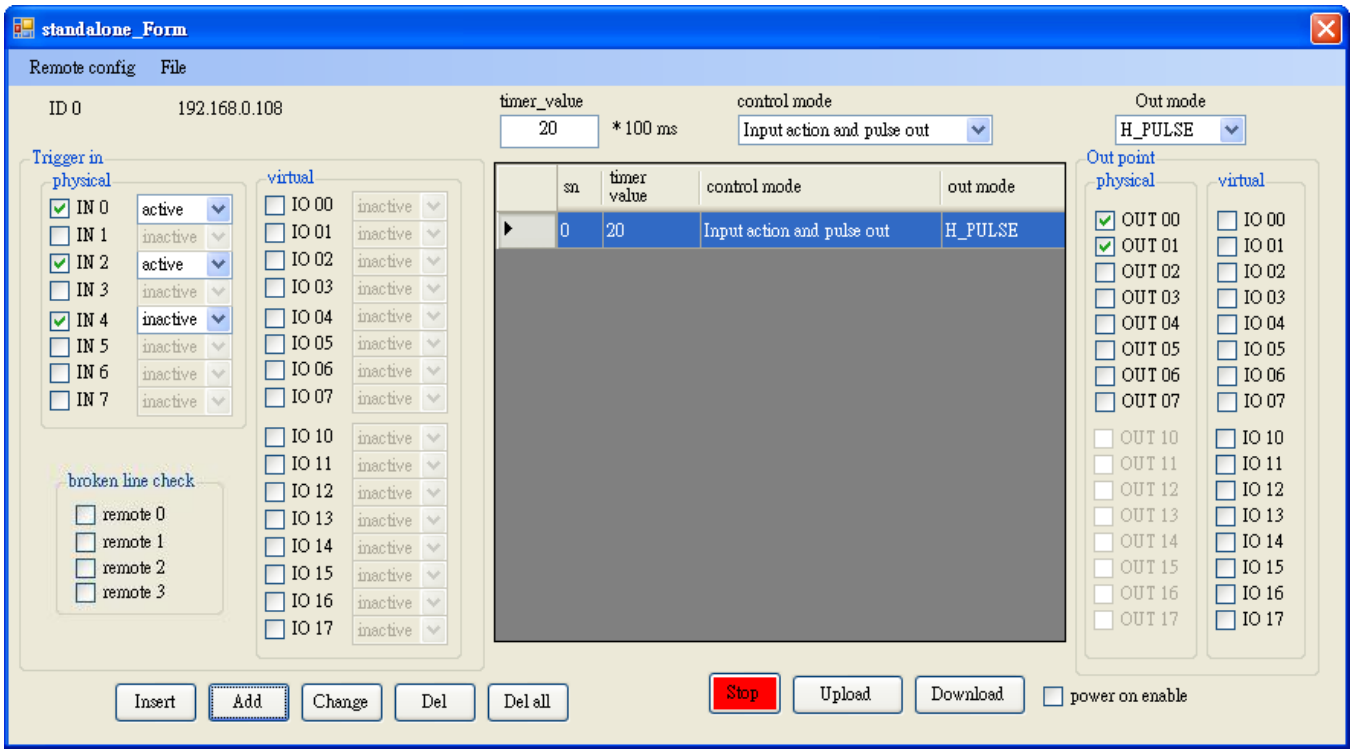


fig. 10.3.2 Example 3 program

Note: While the pulse timing is in progress, no further condition meet triggers will be valid.

10.4 Example 4: Monitoring the inputs if condition meets, output periodically and stop by some special input condition

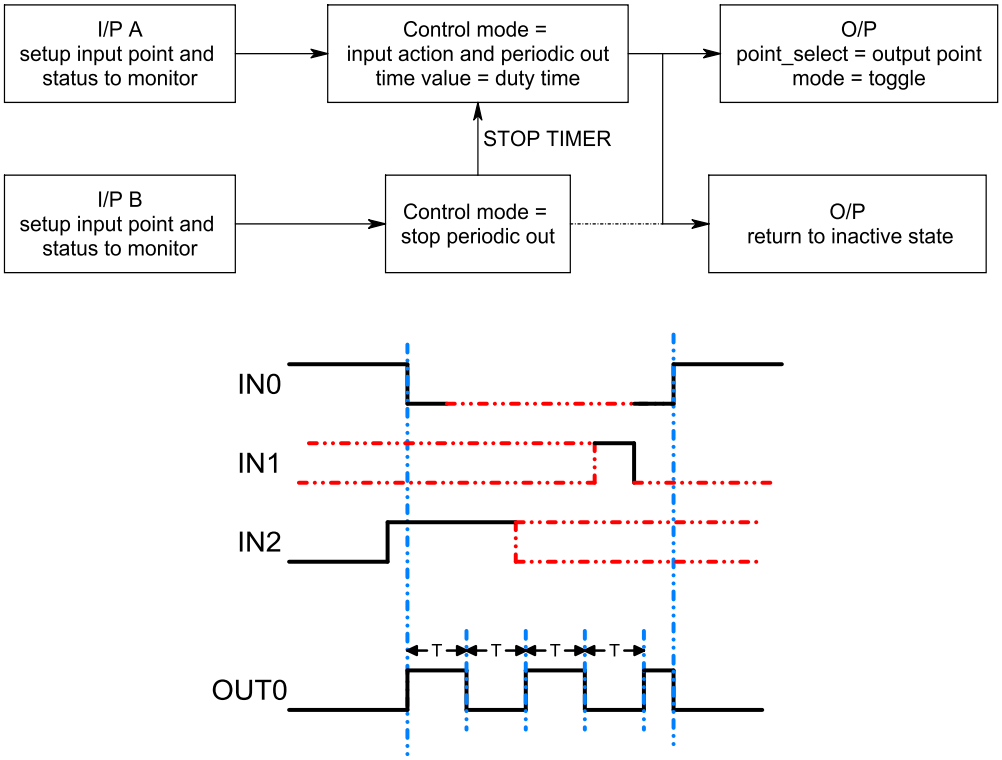


fig.10.4.1 Example 4, block diagram and timing

Say, you want to watch IN00 is inactive and IN02 is active to trigger output OUT00 to toggle. Program as the following shown.

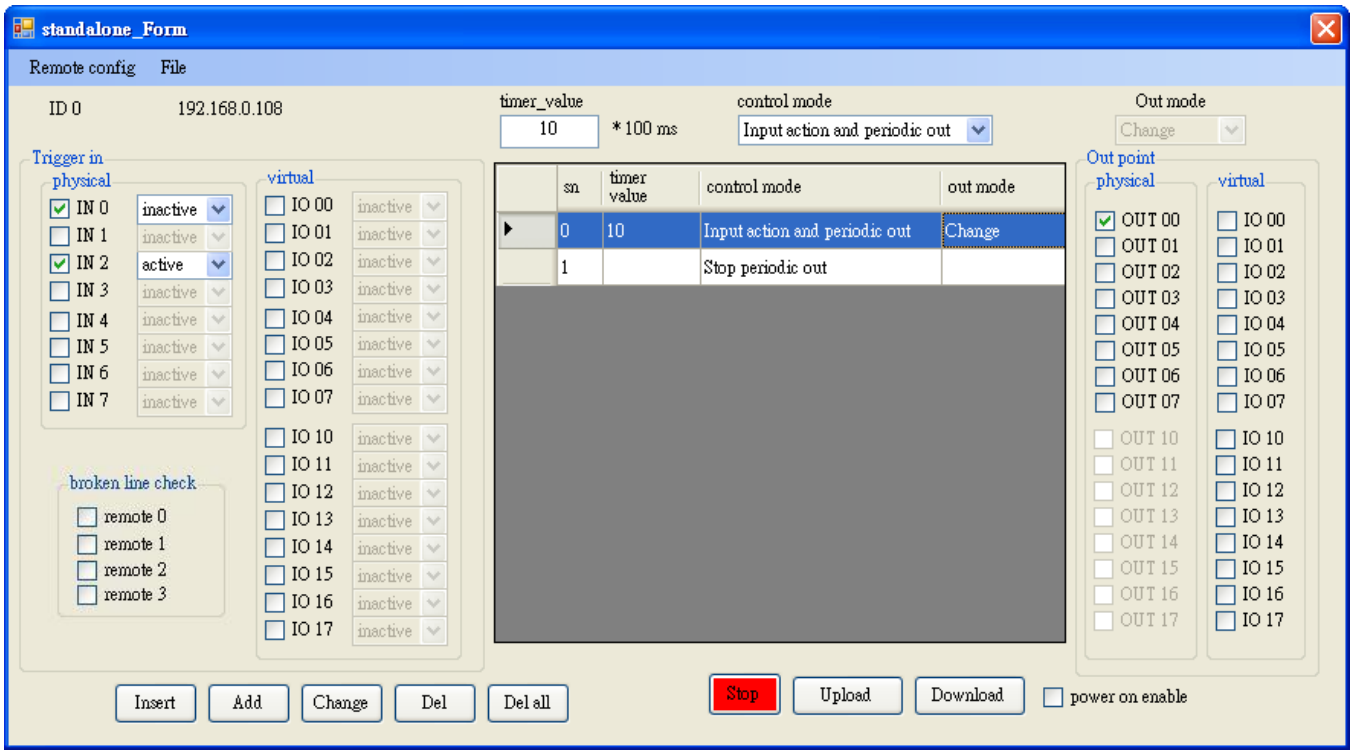


fig. 10.4.2 Example 4 program

Then, if we want IN0 is active and IN1 is inactive to trigger to stop the timer. Program as the following shown.

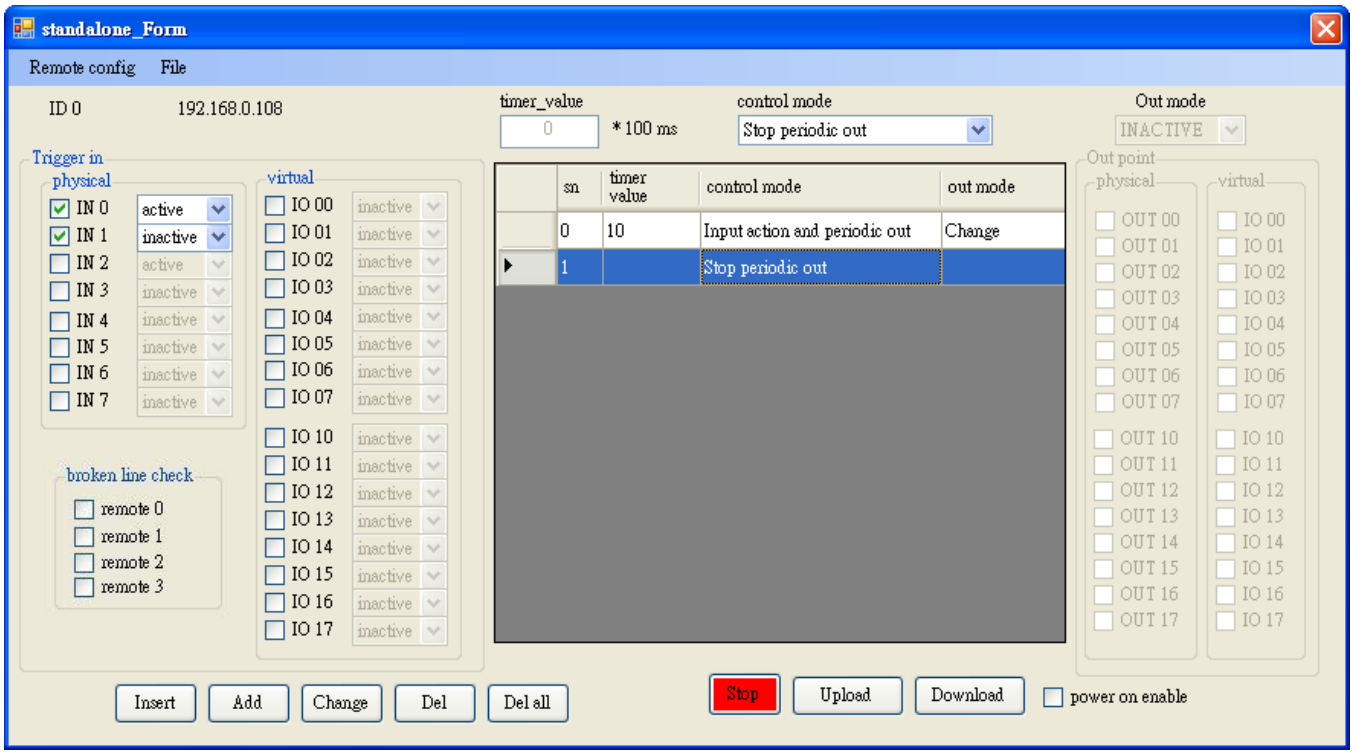


fig. 10.4.3 Example 4 program-1

10.5 Example 5: Don’t care the inputs if standalone enabled, trigger output with delay

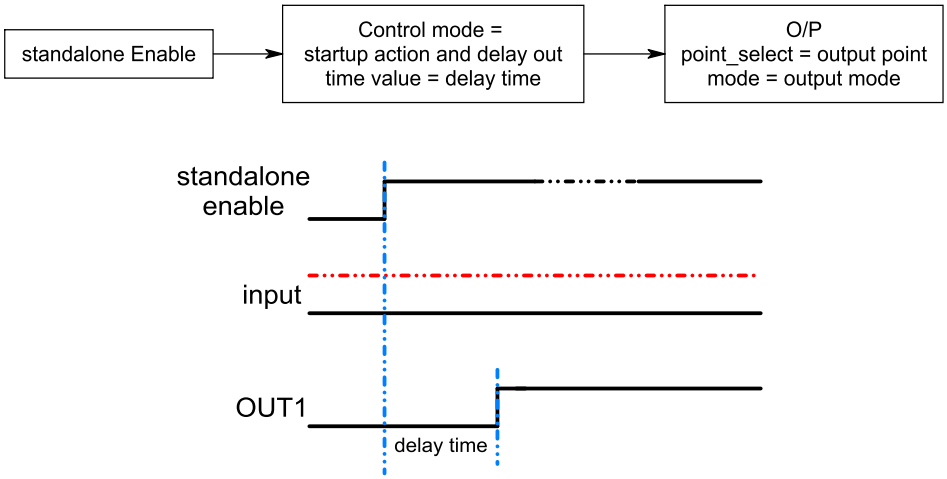


fig. 10.5.1 Example 5, block diagram and timing

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

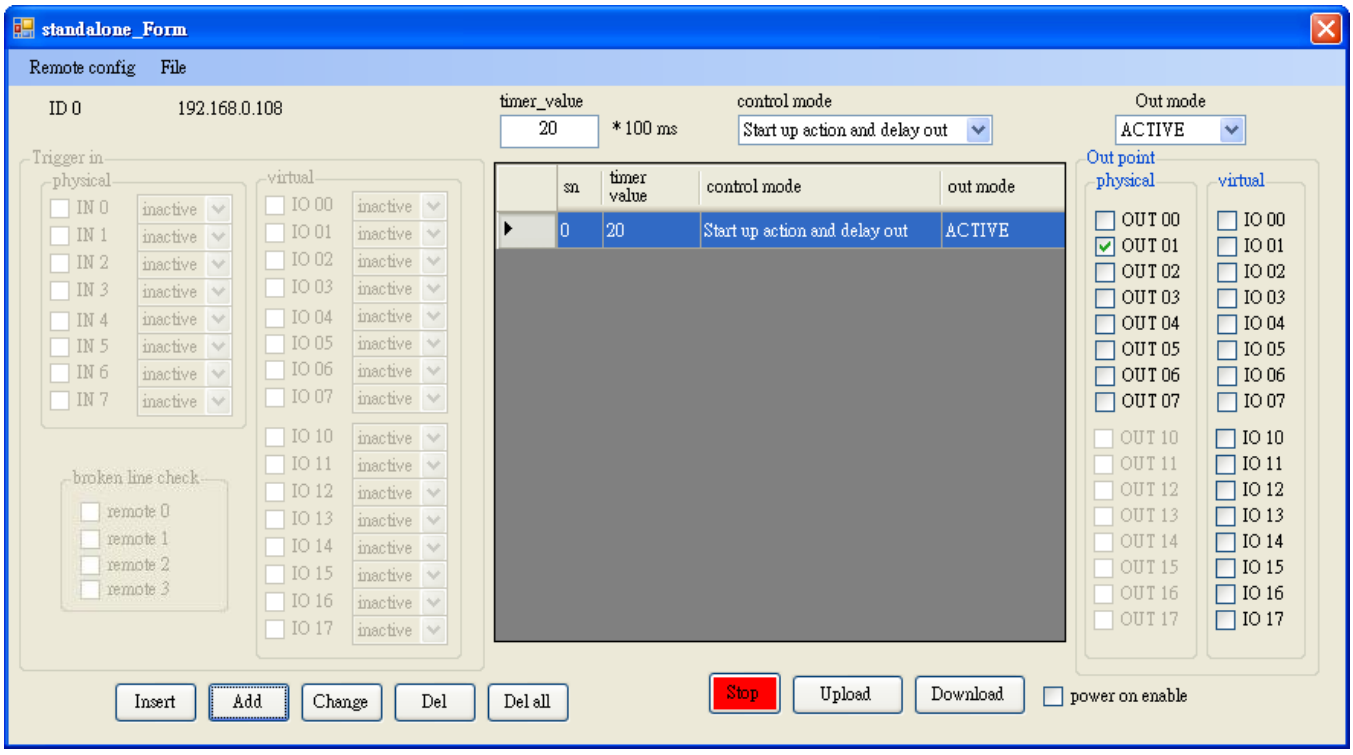


fig. 10.5.2 Example 5 program

10.6 Example 6: Don’t care the inputs if standalone enabled, trigger pulse

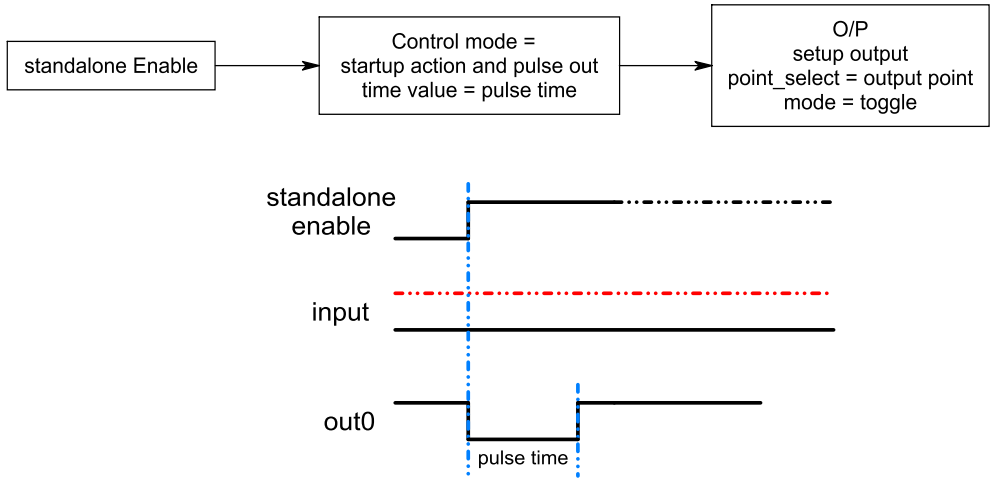


fig. 10.6.1 Example 6, block diagram and timing

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

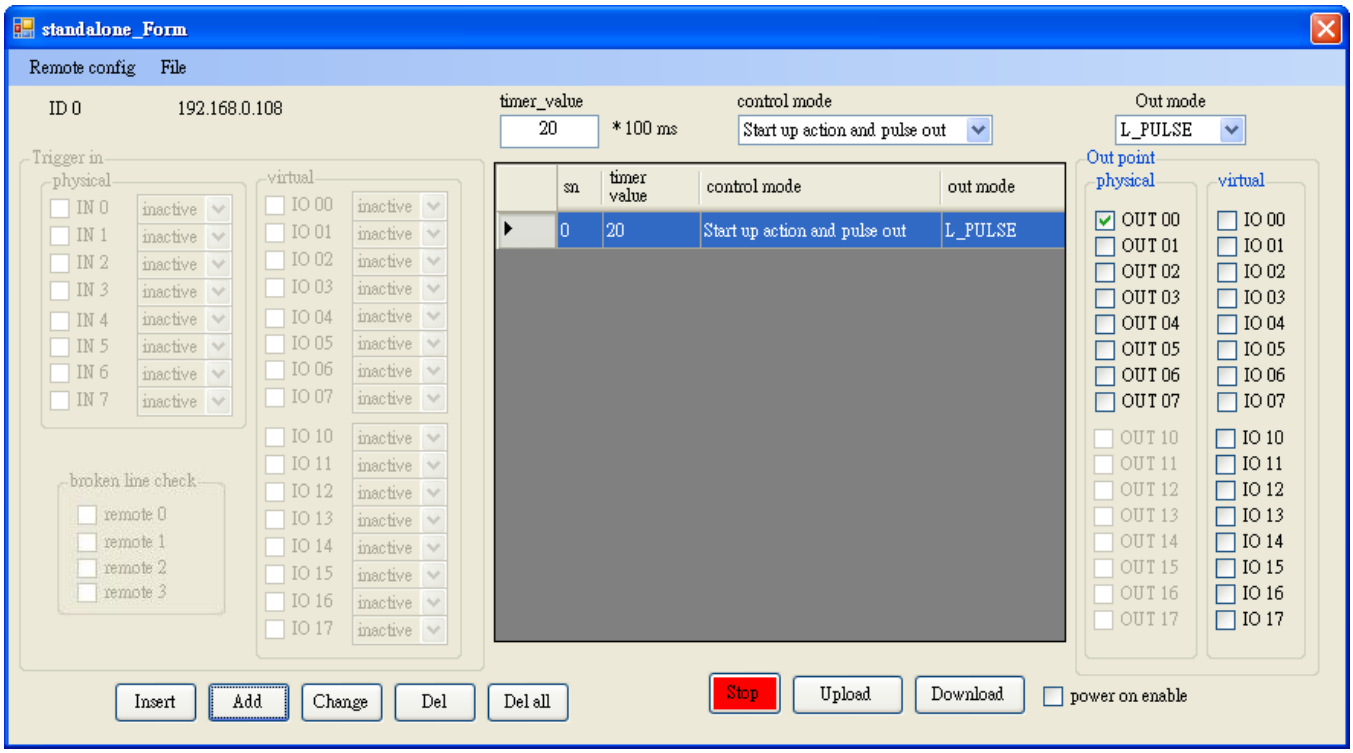


fig. 10.6.2 Example 6 program

10.7 Example 7: Don’t care the inputs if standalone enabled, output periodically and stop if input condition meets

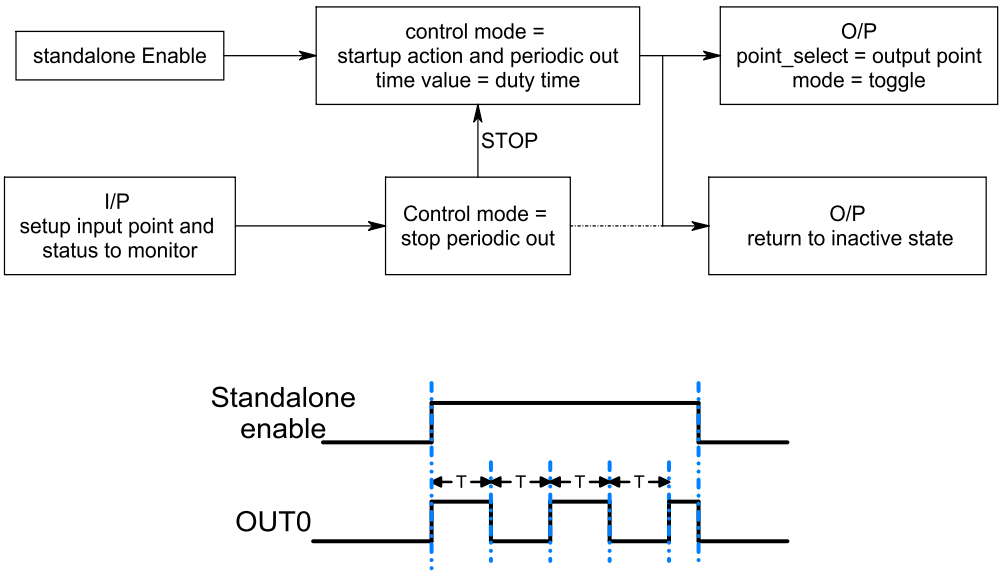


fig. 10.7.1 Example 7, block diagram and timing

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.

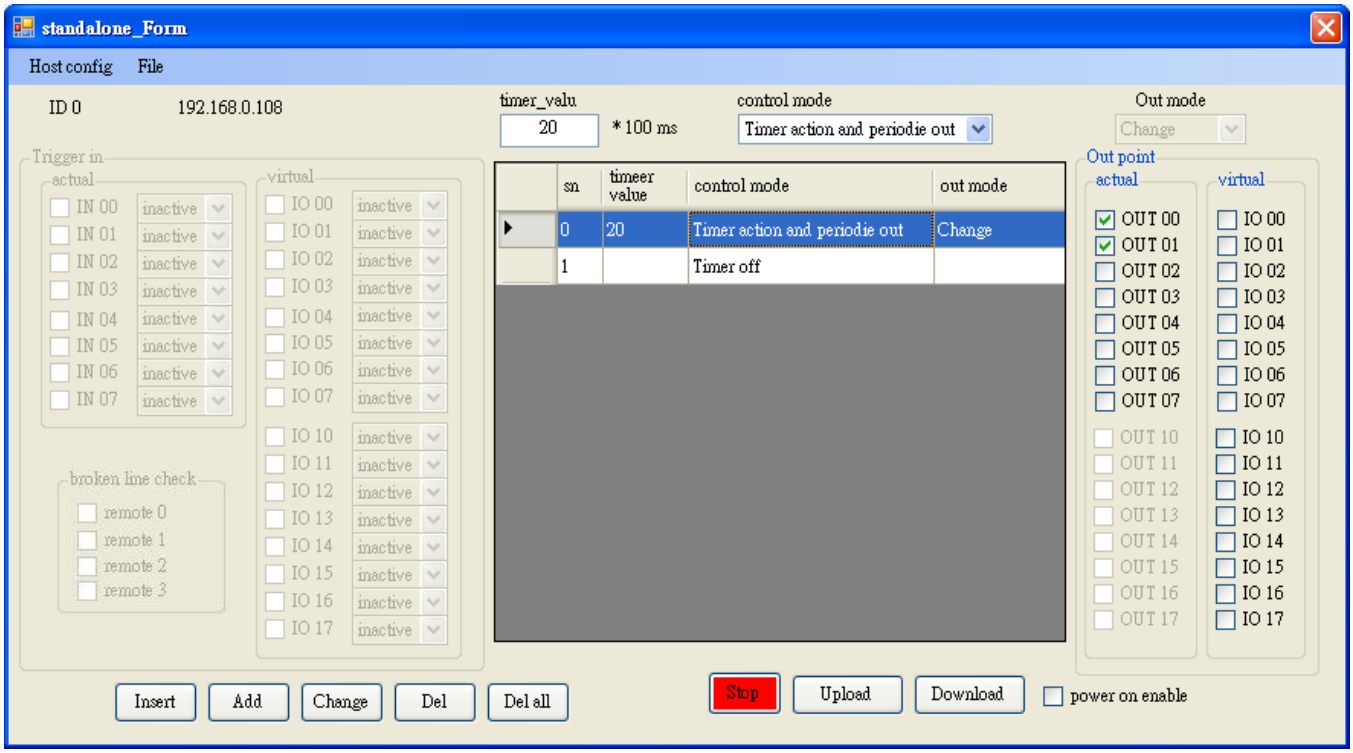


fig. 10.7.2 Example 7 program

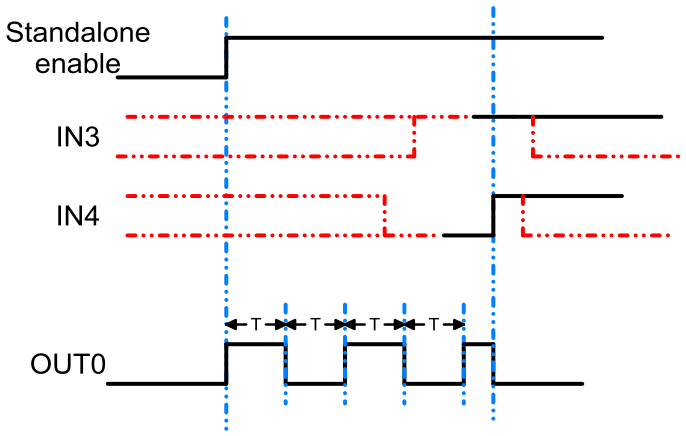


fig. 10.7.3 Example 7, timing of stop standalone mode

Say, start the periodic output on the standalone mode enabled and stop the output on IN3 and IN4 are active. The program is shown as followings.

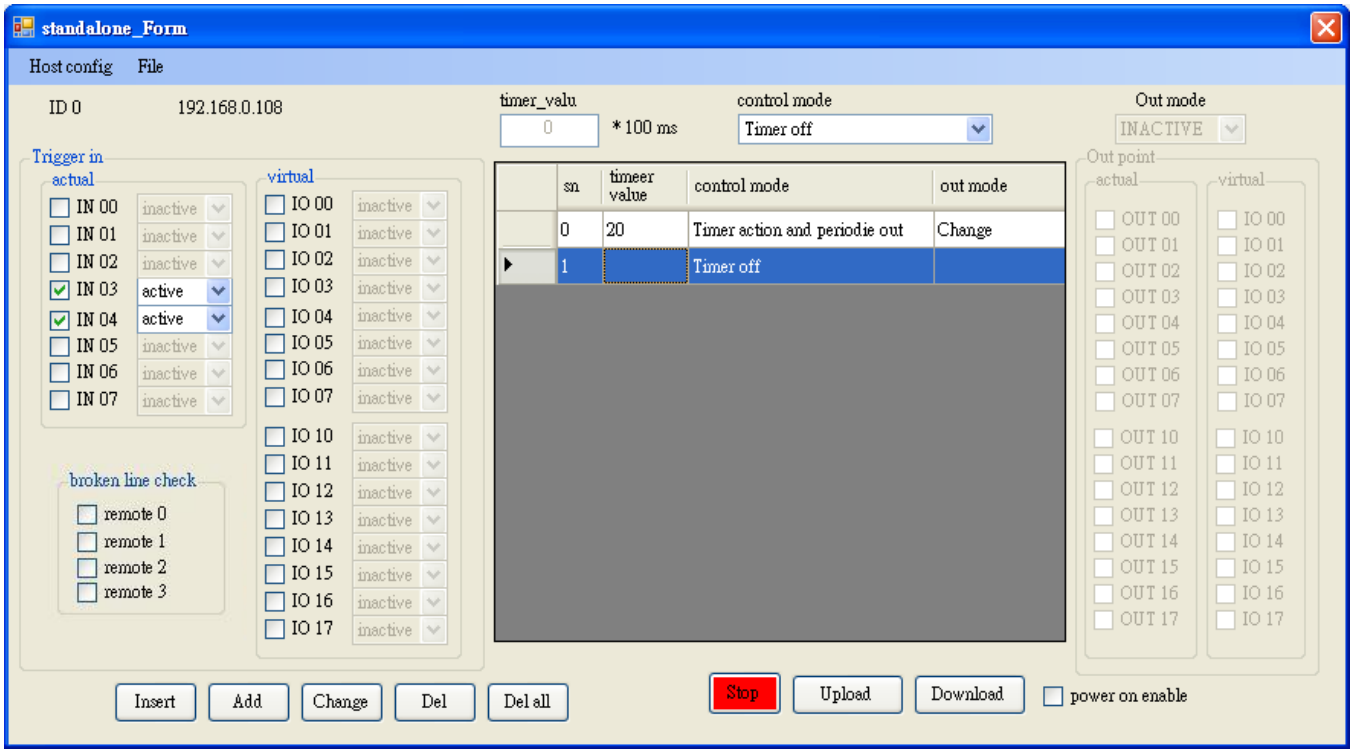


fig. 10.7.4 Example 7, program of stop standalone mode

10.8 Example 8: Monitoring inputs if condition meets, trigger remote output

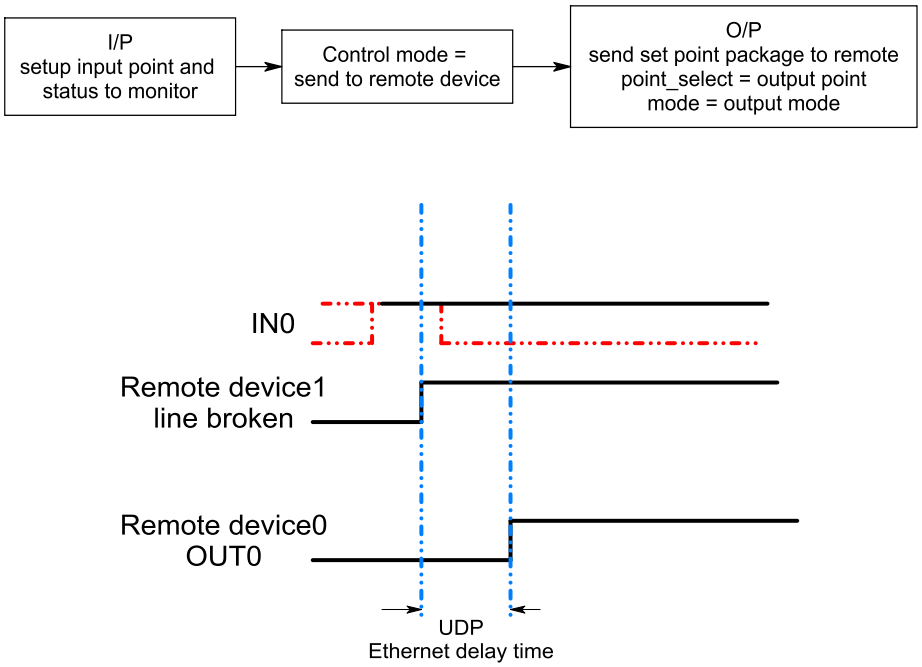


fig. 10.8.1 Example 8, block diagram and timing

Say, you want to watch IN0 is active and remote device 1 is line broken to trigger remote device0's output OUT0 to active. Program as the following shown.

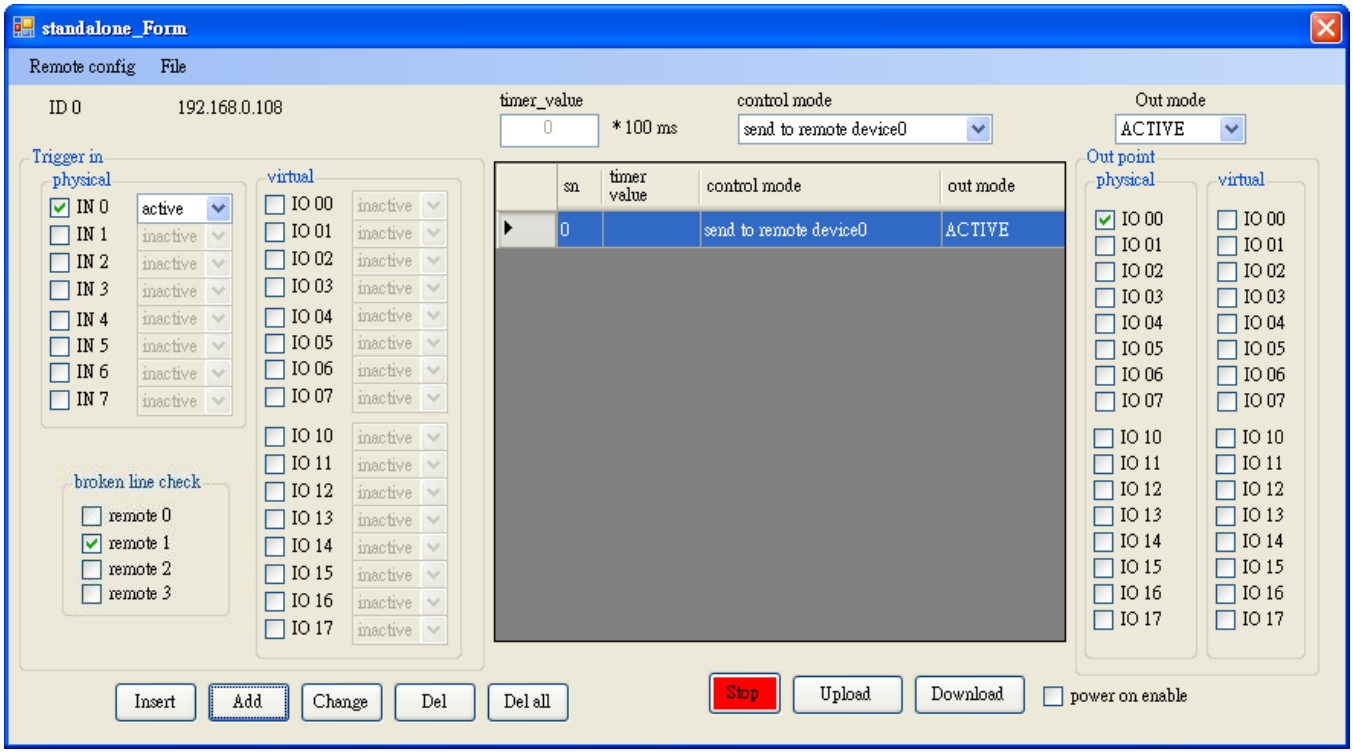


fig. 10.8.2 Example 8 program

Please note that you must already configure the remote device information before using the send to remote Control mode (refer 9.3 Remote configure) .

10.9 Example 9: Monitoring inputs if condition meets, trigger delayed pulse output

On a single module it seems difficult to generate an event triggered delayed pulse but in fact it is easy with the virtual IO concept. Say for example, you want to output a pulse 0.5s on OUT0 which is delayed 1s to IN0 is active.

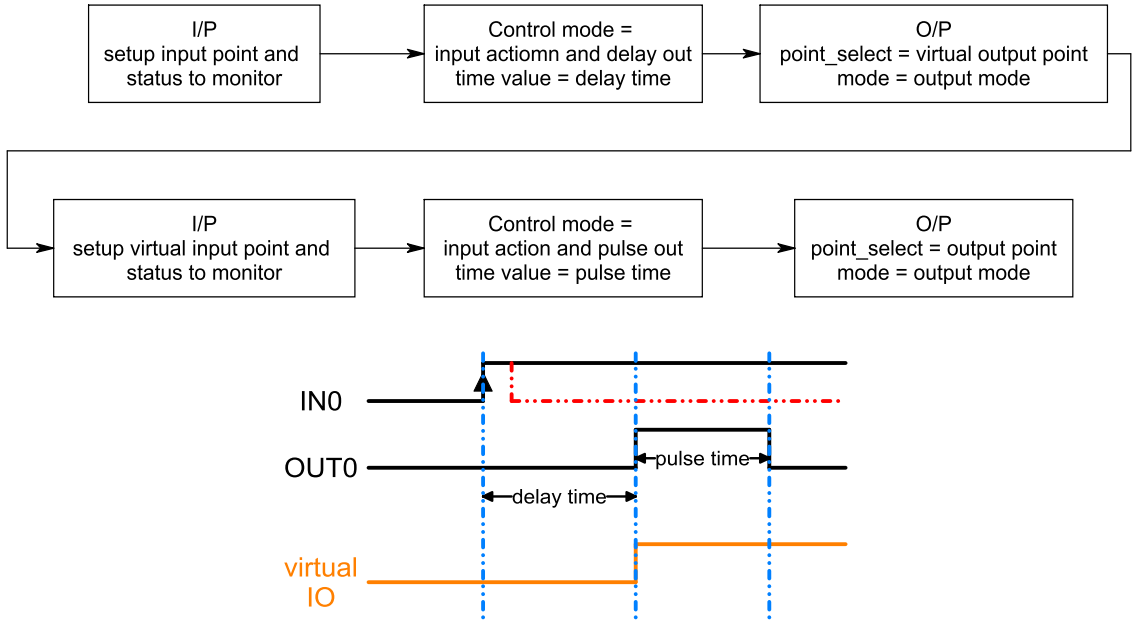


fig. 10.9.1 Example 9, block diagram and timing

Program as the following shown.

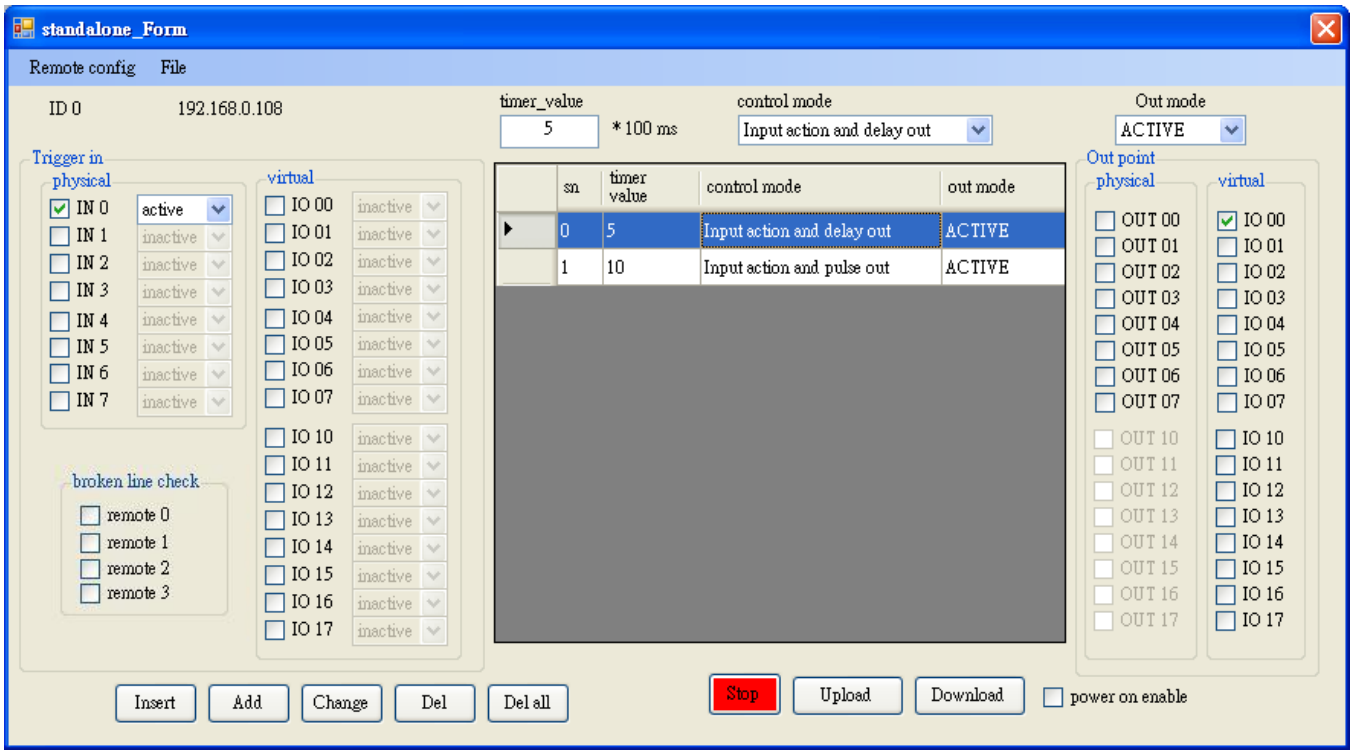


fig. 10.9.2 Example 9 program-1

Then, you can to watch virtual IO00 is active to trigger output OUT00 to H-pulse. Program as the following shown.

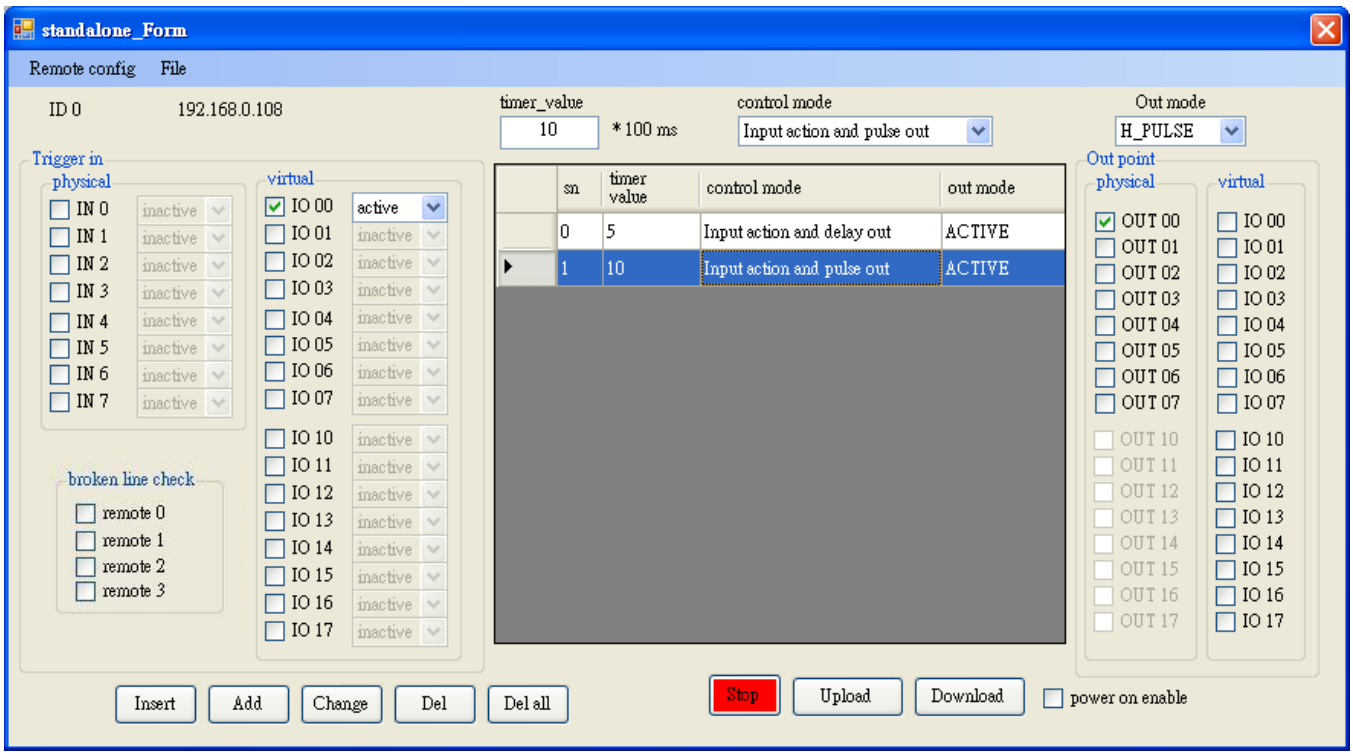


fig. 10.9.3 Example 9 program-2

10.10 Integration example

On a 2 or more modules system, module A put at field site is in charge of monitoring the Ethernet from remote device0 (say, another Ethernet module C at ip:192.168.0.101 for example) and a smoke sensor (or other many inputs but we take one for example). Module B (ip:192.168.0.118 for example) is put at office to receive abnormal event trigger to alarm for the technician.

While smoke sensor detection is active, the module A needs to generate alarm and also the office site must alarm. But if only line broken, the office site must flash light (0.5s on/off flash) and field site no signal out.

Let us assign the I/O points,

- IN0 (at module A) : smoke sensor / active
- OUT0 (at module A) : alarm / active
- line broken detection (at module A detect remote device0 (module C)) : active
- Virtual IO0 (at module B) : event trigger from module A /active
- OUT0 (at module B) : alarm / active (smoke)
- OUT1 (at module B) : flash / active (broken line of module C)

To do the device to device control, the first thing to do is to setup the remote information to the device (host) which will generate the trigger signal, current example is the module A. Module B and C are the remote modules.

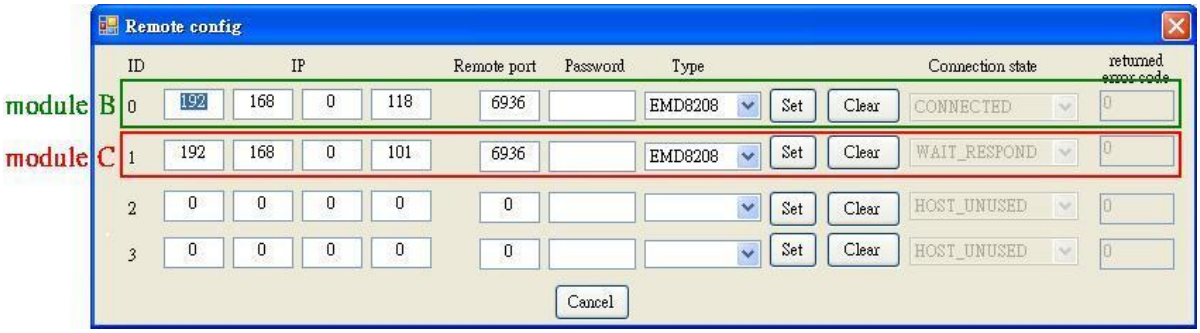


fig. 10.10.1 Remote configuration information for the module A

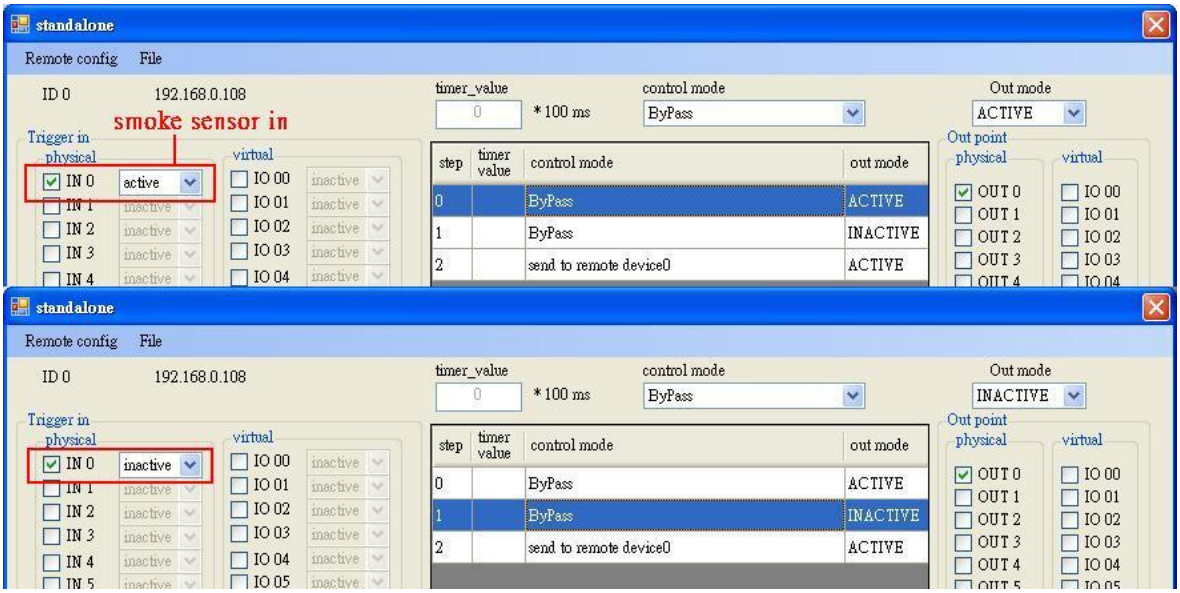


fig. 10.10.2 Input monitoring and its output (program on module A)

Note: Owing to the input condition meets will trigger the output (else keep the previous output state), you need to monitor the input active and input inactive to trigger the output.

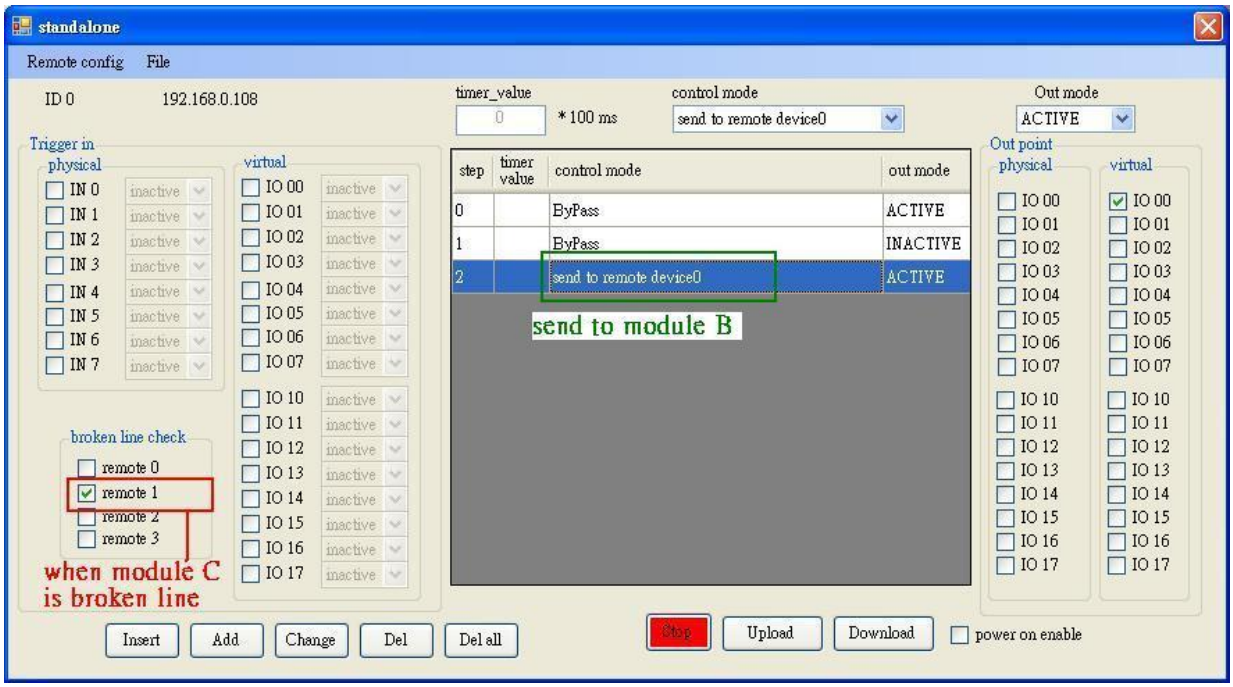


fig. 10.10.3 Broken line monitoring and its output (program on module A)

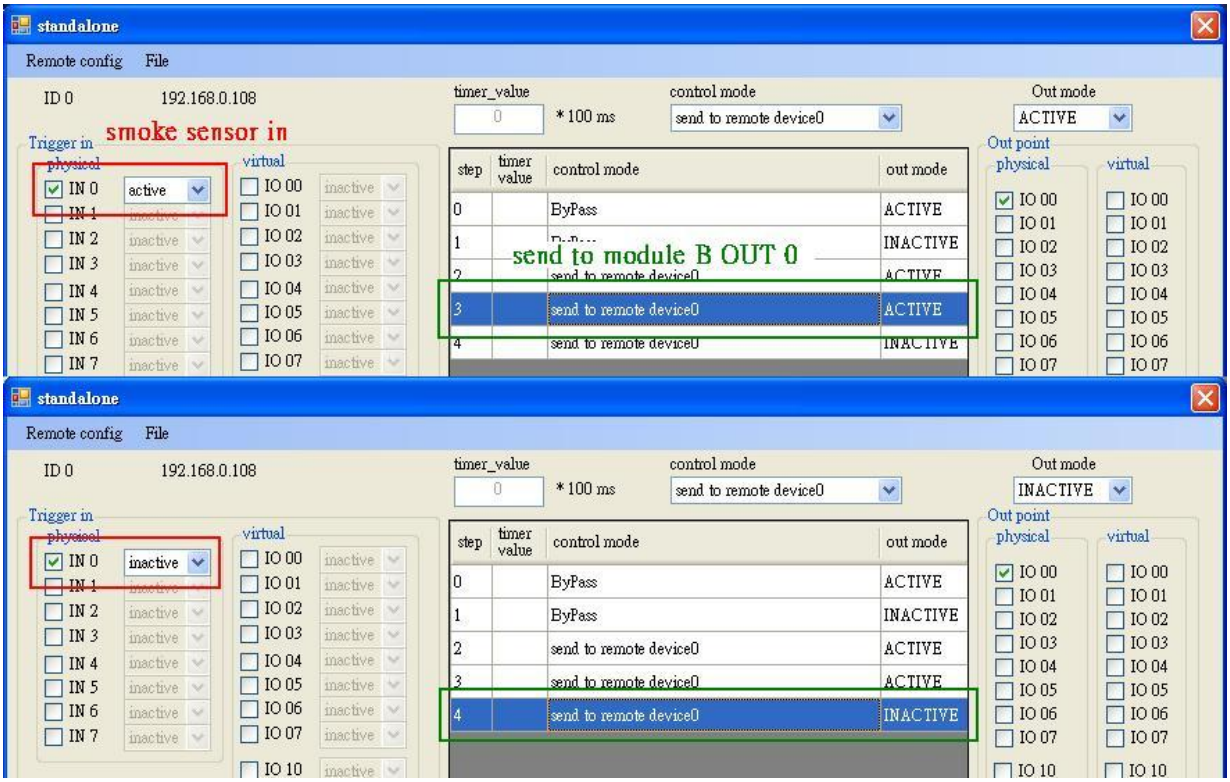


fig. 10.10.4 Input monitoring and output to trigger remote output (program on module A)

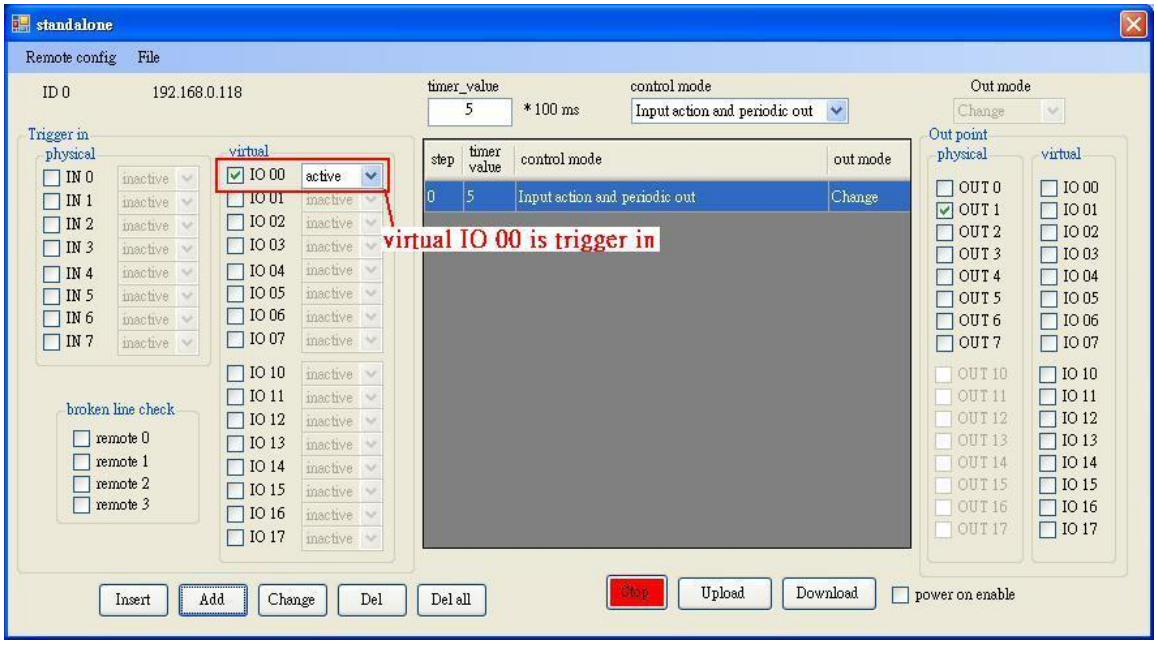


fig. 10.10.5 Virtual input monitoring to trigger toggle output (program on module B)

11. Communication protocol

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

11.1 Host to module command format

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

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

```
typedef struct _EMD820x_UDP_T_data
{
    u8   Module_name[7];        // Module_name={‘E’, ‘M’, ‘D’, ‘8’, ‘2’, ‘0’, ‘x’ }
    u8   password[8];           // password, 8 alphabets
    u8   command;               // command（EMD820x UDP COMMAND LIST）
    Data  data_in;              // data maximum 32 bytes
}EMD820x_data;

typedef union _Data
{
    u8   data_b[32];            //Data for byte
    u16  data_w[16];            //Data for word
    u32  data_l[8];             //Data for long
    u8   IP[4];                 //Data（IP Address）
    u16  socket_port;           //Data（Socket Port）
    u8   New_Password[8];       //Data（New password,8 words）
    u8   MAC[6];                //Data（MAC Address）
    u8   Port_Value[2];         //Data（Port Value）Port_Value[0] = Port,
                                //                 Port_Value[1] = value
    u8   Point_Value[3];        //Data（Point Select）Point_Value[0] = Port,
                                //                    Point_Value[1] = Point
                                //                    Point_Value[2] = Point
    _WDTData            WDT;            // Data（WDT wait time & output state）
    _NEW_ST_DATA        new_st_data;    // maximum 24byte for standalone mode
    _MULTI              multi;          // multi point R/W
    _REMOTE_CONFIG      remote_config;      // remote device configuration data
```

```
    _REMOTE_STATE       remote_state;       // remote device connect state and error code
}Data;


typedef struct    WDT_Data
{
    u16     Timer_Value;            // WDT timer value
    u8      output;                 // WDT timer out, output data
    u8      state;                  // 1: WDT Enable, 0: WDT Disable
} _WDT_Data;

typedef struct    NEW_ST_DATA
{
    u8    step_index;           // start function index
    u8    step _number;         // be used max function number
    u8    standalone_state;
    u8    power_on_state;
    _STEP_DATA config_data;    //step command
}_NEW_ST_DATA;

typedef struct  MULTI
{
    u8    point[4];             // choose points
    u8    state[4];             // points state
} _MULTI;

typedef struct    REMOTE_CONFIG
{
    u8    IP[4];                // remote IP address
    u8    password[8];          // remote device password
    u16   remote_port;          // remote device socket port
    u8    module_type;          // remote device type, other: error
                                    // 1: EMD8204, 2: EMD8208, 3: EMD820X
                                    // 4: EMC8432, 5: EMC8485, 6: EMA8314R
                                    // 7: EMA8308, 8: EMA8308D, 9: PC
    u8    index;                // remote device ID, 0~3
}_REMOTE_CONFIG;

typedef struct    REMOTE_STATE
{
    u8    connection_state[4];      //Data (connection state and mask)
```

```
    u8    connection_ERROR_code[4];       //Data (connection state and mask)
}_REMOTE_STATE;



typedef struct _STEP_DATA
{
    u8    control_mode;           // set control mode
    u8    out_mode;               // set out mode
    u8    timer_value[4];         // set timer value
    u8    input_point[2];         // choose IN00 ~ IN17 (depends on module)
    u8    input_V_point[2];       // choose virtual IO00 ~ IO17
    u8    input_state[2];         // set input state of IN00~IN17
    u8    input_V_state[2];       // set input state of virtual IO00 ~ IO17
    u8    output_point[2];        // choose OUT00 ~ OUT17
    u8    output_V_point[2];      // choose virtual IO00 ~ IO17
    u8    remote_detection[4];    // set remote connect detection
}STEP_DATA;
```

11.2 Module to remote command format

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

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

```
typedef struct _EMD820x_R_data
{
    Receive_Data  Data      // Receive Data maximum 32byte
    u8   success_flag;      // Flag （0:Send command Failed   0x63:Send command successfully）
    u8   command            // command （EMD8204/08 UDP COMMAND LIST）
}EMD820x_receive;
```

The Receive_Data is defined as:

```
typedef   union _Receive_Data
{
    u8   data_b[32];        //Data for byte
    u16  data_w[16];        //Data for word
    u32  data_l[8];         //Data for long
    u8   module_type;       //module type
                            // 1: EMD8204, 2: EMD8208, 3: EMD8216
                            // 4: EMC8432, 5: EMC8485, 6: EMA8314R
                            // 7: EMA8308, 8: EMA8308D, 9: PC
    u8   Port_Value[2];     //Data（Port Value）Port_Value[0] = Port,
                            //                  Port_Value[1] = value
    u8   Point_Value[3];    //Data（Point Select）Point_Value[0] = Port,
                            //                    Point_Value[1] = Point
                            //                    Point_Value[2] = Value
    WDT_DATA   WDT;         //Data（WDT wait time & output state）
    u8   Version[2]         //Data ( firmware version )

} Receive_Data


    Typedef struct   WDT_DATA
    {
        u16  Timer_Value;           // WDT timer value
```

```
    u8    output;                    // WDT timer out I/O output state
    u8    state;                     // 1: WDT Enable, 0: WDT Disable
}_WDT_Data;
```

11.3 Definition of IP header

The IP header is defined as follows:

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

11.4 Definition of UDP header

The UDP header is defined as follows:

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

11.5 EMD-820x communication commands

- **GET_MODULE_TYPE**

Function: ask the EMD820x module type

Host to Ethernet module:

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

Module_name: ‘E’, ‘M’, ‘D’, ‘8’, ‘2’, ‘0’, ‘x’

Password: un-used

Command: 0x1

Data: un-used

Ethernet module to Host:

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

Command: 0x1

Flag: = 0x0 //command fail (this is not EMD820x)

= 0x63 //command successful (this is EMD820x)

Data: Module_Type= 0x0; //= 0 is EMD8204

module_type= 0x1 //= 1 is EMD8208

- **REBOOT**

Function: reboot EMD820x module

Host to Ethernet module:

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

Module_name: ‘E’, ‘M’, ‘D’, ‘8’, ‘2’, ‘0’, ‘x’

Password: your password

Command: 0x2

Data: un-used

Ethernet module to Host:

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

Command: 0x2

Flag: = 0x0 //command fail

= 0x63 //command successful

Data: un-used

- **CHANGE_SOCKETPORT**

Function: change socket port of EMD820x module

Host to Ethernet module:

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

Module_name: ‘E’, ‘M’, ‘D’, ‘8’, ‘2’, ‘0’, ‘x’

Password: your password

Command: 0x3

Data: socket_port //your new socket port number

other structure members are un-used.

Parameter	Type	Description
socket_port	u16	socket port number

Ethernet module to Host:

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

Command: 0x3

Flag: = 0x0 //command fail

= 0x63 //command successful

Data: // unused

- **CHANGE_PASSWORD**

Function: change password of EMD820x module

Host to Ethernet module:

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

Module_name: ‘E’, ‘M’, ‘D’, ‘8’, ‘2’, ‘0’, ‘x’

Password: your password （8 bytes）

Command: 0x4

Data: password[8] //your new password

other structure members are un-used.

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

Ethernet module to Host:

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

Command: 0x4

Flag: = 0x0 //command fail

= 0x63 //command successful

Data: // unused

- **RESTORE_PASSWORD**

Function: restore password of EMD820x module

Host to Ethernet module:

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

Module_name: ‘E’, ‘M’, ‘D’, ‘8’, ‘2’, ‘0’, ‘x’

Password: un-used

Command: 0x5

Data: un-used.

Ethernet module to Host:

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

Command: 0x5

Flag: = 0x0 //command fail

= 0x63 //command successful

Data: // unused

- **CHANGE_IP**

Function: change IP of EMD820x module

Host to Ethernet module:

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

Module_name: ‘E’, ‘M’, ‘D’, ‘8’, ‘2’, ‘0’, ‘x’

Password: your password （8 bytes）

Command: 0x6

Data: IP[4] //new IP address

other structure members are un-used.

Parameter	Type	Description
IP[4]	u8	new IP address to be set

Ethernet module to Host:

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

Command: 0x6

Flag: = 0x0 //command fail

= 0x63 //command successful

Data: // unused

- **GET_FIRMWARE_VERSION**

Function: read the firmware version of EMD820x module

Host to Ethernet module:

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

Module_name: ‘E’, ‘M’, ‘D’, ‘8’, ‘2’, ‘0’, ‘x’

Password: your password （8 bytes）

Command: 0x7

Data: un-used

Ethernet module to Host:

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

Command: 0x7

Flag: = 0x0 //command fail

= 0x63 //command successful

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

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

- **CHANGE_SUBNET_MASK**

Function: set the subnet mask

Host to Ethernet module:

IP header	UDP header	UDP data			
		Module_name	Password	0x8	Data
20bytes	8bytes	7bytes	8bytes	1byte	32bytes

Module_name: ‘E’, ‘M’, ‘D’, ‘8’, ‘2’, ‘0’, ‘x’

Password: your password （8 bytes）

Command: 0x8

Data: IP[4] //new subnet mask

other structure members are un-used.

Parameter	Type	Description
IP[4]	u8	new subnet mask to be set

Ethernet module to Host:

IP header	UDP header	UDP data		
		Data	Flag	0x8
20bytes	8bytes	32bytes	1byte	1byte

Command: 0x8

Flag: = 0x0 //command fail

= 0x63 //command successful

Data: un-used

- **READ_SUBNET_MASK**

Function: read the subnet mask

Host to Ethernet module:

IP header	UDP header	UDP data			
		Module_name	Password	0x9	Data
20bytes	8bytes	7bytes	8bytes	1byte	32bytes

Module_name: ‘E’, ‘M’, ‘D’, ‘8’, ‘2’, ‘0’, ‘x’

Password: your password （8 bytes）

Command: 0x9

Data: un-used

Ethernet module to Host:

IP header	UDP header	UDP data		
		Data	Flag	0x9
20bytes	8bytes	32bytes	1byte	1byte

Command: 0x9

Flag: = 0x0 //command fail

= 0x63 //command successful

Data: IP[4] //subnet mask

other structure members are un-used.

Parameter	Type	Description
IP[4]	u8	subnet mask to be set

- **WRITE_MAC**

Function: the MAC address of EMD820x module

Host to Ethernet module:

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

Module_name: ‘E’, ‘M’, ‘D’, ‘8’, ‘2’, ‘0’, ‘x’

Password: your password （8 bytes）

Command: 0xfa

Data: MAC[6] //new MAC address

other structure members are un-used.

Parameter	Type	Description
MAC[6]	u8	MAC address to be set

Ethernet module to Host:

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

Command: 0xfa

Flag: = 0x0 //command fail

= 0x63 //command successful

Data: un-used

- **SET_COUNTER_MASK**

Function: set the counter input mask (enable or disable counter function per channel) of EMD820x module

Host to Ethernet module:

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

Module_name: 'E', 'M', 'D', '8', '2', '0', 'x'

Password: your password （8 bytes）

Command: 0x20

Data: Port_Value[1] // counter mask

Parameter	Type	Description
Port_Value[0]	u8	un-used
Port_Value[1]	u8	counter mask Any one work as output, the mask can only set to 0 bit 7: 0: IN7 mask off (disable) counter function 1: IN7 counter function enabled …. bit 0: 0: IN0 mask off (disable) counter function 1: IN0 counter function enabled

Notes: EMD8204 has only IN0 ~ IN3

Ethernet module to Host:

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

Command: 0x20

Flag: = 0x0 //command fail

= 0x63 //command successful

Data: un-used

- **ENABLE_COUNTER_MODE**

Function: global enable counter function of EMD820x module

Host to Ethernet module:

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

Module_name: ‘E’, ‘M’, ‘D’, ‘8’, ‘2’, ‘0’, ‘x’

Password: your password （8 bytes）

Command: 0x21

Data: un-used

Ethernet module to Host:

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

Command: 0x21

Flag: = 0x0 //command fail

= 0x63 //command successful

Data: un-used

- **DISABLE_COUNTER_MODE**

Function: global disable counter function of EMD820x module

Remote to Ethernet module:

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

Module_name: ‘E’, ‘M’, ‘D’, ‘8’, ‘2’, ‘0’, ‘x’

Password: your password （8 bytes）

Command: 0x22

Data: un-used

Ethernet module to Remote:

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

Command: 0x22

Flag: = 0x0 //command fail

= 0x63 //command successful

Data: un-used

- **READ_COUNTER**

Function: read the counter data on the fly of EMD820x module

Remote to Ethernet module:

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

Module_name: ‘E’, ‘M’, ‘D’, ‘8’, ‘2’, ‘0’, ‘x’

Password: your password （8 bytes）

Command: 0x23

Data: un-used

Ethernet module to Remote:

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

Command: 0x23

Flag: = 0x0 //command fail

= 0x63 //command successful

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

Parameter	Type	Description
data_l[8]	u32	counter value of selected port data_l[7] : counter value of IN7 … data_l [0] : counter value of IN0

- **CLEAR_COUNTER**

Function: clear counter data per channel of EMD820x module

Remote to Ethernet module:

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

Module_name: ‘E’, ‘M’, ‘D’, ‘8’, ‘2’, ‘0’, ‘x’

Password: your password （8 bytes）

Command: 0x24

Data: Port_Value[1]

other structure members are un-used.

Parameter	Type	Description
Port_Value[0]	u8	un-used
Port_Value[1]	u8	bit 7: 0: unchanged of counter of IN7 1: clear counter of IN7 …. bit 0: 0: unchanged of counter of IN0 1: clear counter of IN0

Ethernet module to Remote:

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

Command: 0x24

Flag: = 0x0 //command fail

= 0x63 //command successful

Data: un-used

- **SET_PORT**

Function: set the output state per channel of EMD820x module

Remote to Ethernet module:

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

Module_name: 'E', 'M', 'D', '8', '2', '0', 'x'

Password: your password （8 bytes）

Command: 0x32

Data: Port_Value[0] , Port_Value[1]

other structure members are un-used.

Parameter	Type	Description
Port_Value[0]	u8	always 0,2,3 Port number 0: choose port 0, i.e. select OUTn 1: choose port 1, i.e. select INn (not available) 2: virtual port0 3: virtual port1
Port_Value[1]	u8	I/O state (only valid for pins configured as outputs) bit 7: 0: OUT7 output inactive 1: OUT7 output active …. bit 0: 0: OUT0 output inactive 1: OUT0 output active

Note: An output channel is active, the relay output may be close or open depends on the polarity it is configured.

Ethernet module to Remote:

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

Command: 0x32

Flag: = 0x0 //command fail

= 0x63 //command successful

Data: un-used

- **READ_PORT**

Function: read the port state per channel of EMD820x module

Remote to Ethernet module:

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

Module_name: 'E', 'M', 'D', '8', '2', '0', 'x'

Password: your password （8 bytes）

Command: 0x33

Data: Pot_Value[0]

other structure members are un-used.

Parameter	Type	Description
Port_Value[0]	u8	Port number 0: choose port 0, i.e. select OUTn 1: choose port 1, i.e. select INn 2: virtual port0 3: virtual port1

Ethernet module to Remote:

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

Command: 0x33

Flag: = 0x0 //command fail

= 0x63 //command successful

Data: Port_Value[0], Port_Value[1]

other structure members are un-used.

Parameter	Type	Description
Port_Value[0]	u8	Port number 0: choose port 0, i.e. select OUTn 1: choose port 1, i.e. select INn 2: virtual port0 3: virtual port1
Port_Value[1]	u8	I/O state bit 7: 0: IN7 or OUT7 or IO17 or IO07 state is inactive (depends on port number) 1: IN7 or OUT7 or IO17 or IO07 state is

		active (depends on port number) …. bit 0: 0: IN0 or OUT0 or IO10 or IO00 state is inactive (depends on port number) 1: IN0 or OUT0 or IO10 or IO00 state is active (depends on port number)

- **SET_PORT_POLARITY**

Function: set the I/O polarity per channel of EMD820x module

Remote to Ethernet module:

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

Module_name: 'E', 'M', 'D', '8', '2', '0', 'x'

Password: your password （8 bytes）

Command: 0x34

Data: Pot_Value[0], Port_Value[1]

other structure members are un-used.

Parameter	Type	Description
Port_Value[0]	u8	Port number 0: choose port 0, i.e. select OUTn 1: choose port 1, i.e. select INn
Port_Value[1]	u8	I/O polarity bit 7: 0: IN7 or OUT7 normal polarity 1: IN7 or OUT7 invert polarity …. bit 0: 0: IN0 or OUT0 normal polarity 1: IN0 or OUT0 invert polarity

Note: An output channel is active, the relay output may be close or open depends on the polarity it is configured.

Ethernet module to Remote:

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

Command: 0x34

Flag: = 0x0 //command fail

= 0x63 //command successful

Data: un-used

- **READ_PORT_POLARITY**

Function: read the I/O polarity per channel of EMD820x module

Remote to Ethernet module:

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

Module_name: 'E', 'M', 'D', '8', '2', '0', 'x'

Password: your password （8 bytes）

Command: 0x35

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

other structure members are un-used.

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

Ethernet module to Remote:

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

Command: 0x35

Flag: = 0x0 //command fail

= 0x63 //command successful

Data: Pot_Value[0], Port_Value[1]

other structure members are un-used.

Parameter	Type	Description
Port_Value[0]	u8	Port number 0: choose port 0, i.e. select OUTn 1: choose port 1, i.e. select INn
Port_Value[1]	u8	I/O polarity bit 7: 0: IN7 or OUT7 normal polarity 1: IN7 or OUT7 invert polarity …. bit 0: 0: IN0 or OUT0 normal polarity 1: IN0 or OUT0 invert polarity

Note: An output channel is active, the relay output may be close or open depends on the polarity it is configured.

- **SET_POINT**

Function: set the output state of a channel of EMD820x module

Remote to Ethernet module:

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

Module_name: 'E', 'M', 'D', '8', '2', '0', 'x'

Password: your password （8 bytes）

Command: 0x38

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

Parameter	Type	Description
Point_Value[0]	u8	always 0, 2, 3 Port number 0: choose port 0, i.e. select OUTn 1: choose port 1, i.e. select INn (not available) 2: virtual port0 3: virtual port1
Point_Value[1]	u8	point (channel) selection 7: select OUT7 or IO17 or IO07 (depends on port number) …. 0: select OUT0 or IO10 or IO00 (depends on port number)
Point_Value[2]	u8	point (channel) configuration 0: inactive 1: active

Note: An output channel is active, the relay output may be close or open depends on the polarity it is configured.

Ethernet module to Remote:

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

Command: 0x38

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

Data: un-used

- **READ_POINT**

Function: read the channel state of EMD820x module

Remote to Ethernet module:

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

Module_name: 'E', 'M', 'D', '8', '2', '0', 'x'

Password: your password （8 bytes）

Command: 0x39

Data: Point_Value[0], Point_Value[1]

other structure members are un-used.

Parameter	Type	Description
Point_Value[0]	u8	Port number 0: choose port 0, i.e. select OUTn 1: choose port 1, i.e. select INn 2: virtual port0 3: virtual port1
Point_Value[1]	u8	point (channel) selection 7: select OUT7 or IO17 or IO07 (depends on port number) …. 0: select OUT0 or IO10 or IO00 (depends on port number)

Ethernet module to Remote:

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

Command: 0x39

Flag: = 0x0 //command fail

= 0x63 //command successful

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

other structure members are un-used.

Parameter	Type	Description
Point_Value[0]	u8	Port number 0: choose port 0, i.e. select OUTn 1: choose port 1, i.e. select INn 2: virtual port0

		3: virtual port1
Point_Value[1]	u8	point (channel) selection 7: select OUT7 or IO17 or IO07 (depends on port number) …. 0: select OUT0 or IO10 or IO00 (depends on port number)
Point_Value[2]	u8	point (channel) configuration 0: inactive 1: active

- **SET_POINT_POLARITY**

Function: set the I/O polarity of a channel of EMD820x module

Remote to Ethernet module:

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

Module_name: 'E', 'M', 'D', '8', '2', '0', 'x'

Password: your password （8 bytes）

Command: 0x3A

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

other structure members are un-used.

Parameter	Type	Description
Point_Value[0]	u8	Port number 0: choose port 0, i.e. select OUTn 1: choose port 1, i.e. select INn
Point_Value[1]	u8	point (channel) selection 0: select IN0 or OUT0 …. 7: select IN7 or OUT7
Point_Value[2]	u8	point (channel) polarity 0: normal 1: invert

Note: An output channel is active, the relay output may be close or open depends on the polarity it is configured.

Ethernet module to Remote:

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

Command: 0x3A

Flag: = 0x0 //command fail

= 0x63 //command successful

Data: un-used

- **READ_POINT_POLARITY**

Function: read the channel polarity of EMD820x module

Remote to Ethernet module:

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

Module_name: ‘E’, ‘M’, ‘D’, ‘8’, ‘2’, ‘0’, ‘x’

Password: your password （8 bytes）

Command: 0x3B

Data: Point_Value[0], Point_Value[1]

other structure members are un-used.

Parameter	Type	Description
Point_Value[0]	u8	Port number 0: choose port 0, i.e. select OUTn 1: choose port 1, i.e. select INn
Point_Value[1]	u8	point (channel) selection 0: select IN0 or OUT7 …. 7: select IN7 or OUT0

Ethernet module to Remote:

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

Command: 0x3B

Flag: = 0x0 //command fail

= 0x63 //command successful

Data: Pot_Value[0], Port_Value[1], Point_value[2]

other structure members are un-used.

Parameter	Type	Description
Point_Value[0]	u8	Port number 0: choose port 0, i.e. select OUTn 1: choose port 1, i.e. select INn
Point_Value[1]	u8	point (channel) selection 0: select IN0 or OUT7 …. 7: select IN7 or OUT0
Point_Value[2]	u8	point (channel) polarity

		0: normal 1: invert

- **WRITE_MULTI_POINT**

Function: set the I/O state of multi point of EMD820X module

Remote to Ethernet module:

IP header	UDP header	UDP data			
		Module_name	Password	0x3C	Data
20bytes	8bytes	7bytes	8bytes	1byte	32bytes

Module_name: ‘E’, ‘M’, ‘D’, ‘8’, ‘2’, ‘0’, ‘x’

Password: your password （8 bytes）

Command: 0x3C

Data: mulit. point [0~3], mulit. state [0~3]

other structure members are un-used.

Parameter	Type	Description
mulit. point [0~3]	u8	point (channel) selection [0]: outport, i.e. select OUT_m [1]: un-used [2]: choose virtual port 0, i.e. select IO0m [3]: choose virtual port 1, i.e. select IO1m
mulit. state [0~3]	u8	point (channel) configuration [0]: choose outport, i.e. select OUT_m [1]: un-used [2]: choose virtual port0, i.e. select IO0m [3]: choose virtual port1, i.e. select IO1m Any bit = 0: inactive = 1: active

Ethernet module to Remote:

IP header	UDP header	UDP data		
		Data	Flag	0x3C
20bytes	8bytes	32bytes	1byte	1byte

Command: 0x3C

Flag: = 0x0 //command fail

= 0x63 //command successful

Data: un-used

- **READ_MULTI_POINT**

Function: set the I/O state of multi point of EMD820X module

Remote to Ethernet module:

IP header	UDP header	UDP data			
		Module_name	Password	0x3D	Data
20bytes	8bytes	7bytes	8bytes	1byte	32bytes

Module_name: 'E', 'M', 'D', '8', '2', '0', 'x'

Password: your password （8 bytes）

Command: 0x3D

Data: mulit. point [0~3]

other structure members are un-used.

Parameter	Type	Description
mulit. point [0~3]	u8	point (channel) selection [0]: choose outport, i.e. select OUT_m [1]: choose inport, i.e. select IN_m [2]: choose virtual port0, i.e. select IO0m [3]: choose virtual port1, i.e. select IO1m

Ethernet module to Remote:

IP header	UDP header	UDP data		
		Data	Flag	0x3D
20bytes	8bytes	32bytes	1byte	1byte

Command: 0x3D

Flag: = 0x0 //command fail

= 0x63 //command successful

Data: mulit. point [0~3], mulit. state [0~3]

other structure members are un-used.

Parameter	Type	Description
mulit. point [0~3]	u8	point (channel) selection [0]: choose outport, i.e. select OUT_m [1]: choose inport, i.e. select IN_m [2]: choose virtual port0, i.e. select IO0m [3]: choose virtual port1, i.e. select IO1m
mulit. state [0~3]	u8	point (channel) configuration [0]: choose outport, i.e. select OUT_m [1]: choose inport, i.e. select IN_m [2]: choose virtual port0, i.e. select IO0m [3]: choose virtual port1, i.e. select IO1m

		Any bit = 0: inactive 1: active

- **ENABLE_STANDALONE**

Function: enable standalone mode operation of EMD820X module

Remote to Ethernet module:

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

Module_name: ‘E’, ‘M’, ‘D’, ‘8’, ‘2’, ‘0’, ‘x’

Password: your password （8 bytes）

Command: 0x50

Data: un-used

Ethernet module to Remote:

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

Command: 0x50

Flag: = 0x0 //command fail

= 0x63 //command successful

Data: un-used

- **DISABLE_STANDALONE**

Function: disable standalone mode operation of EMD820X module

Remote to Ethernet module:

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

Module_name: ‘E’, ‘M’, ‘D’, ‘8’, ‘2’, ‘0’, ‘x’

Password: your password （8 bytes）

Command: 0x51

Data: un-used

Ethernet module to Remote:

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

Command: 0x51

Flag: = 0x0 //command fail

= 0x63 //command successful

Data: un-used

- **SET_STANDALONE_CONFIG_NEW**

Function: set new standalone commands from EMD820X module

Remote to Ethernet module:

IP header	UDP header	UDP data			
		Module_name	Password	0x54	Data
20bytes	8bytes	7bytes	8bytes	1byte	32bytes

Module_name: 'E', 'M', 'D', '8', '2', '0', 'x'

Password: your password （8 bytes）

Command: 0x54

Data: _NEW_ST_DATA new_standalone_data; //new standalone instruction

Data: _NEW_ST_DATA new_standalone_data; // and operation mode

Parameter	Type	Description
function_index	u8	command line index of current communication. Actually one communication data will carry 2 commands. The function_index is the first command's line number. allowable range: 0 ~ 31 (command line index)
function_number	u8	Standalone mode total commands. allowable range: 1 ~ 32 (commands)
standalone_state	u8	Unused
power_on_state	u8	=0x0 : power on standalone disable =0x1 : power on standalone enable
control_mode	u8	timer working mode of current command: =0x0 : Bypass =0x1 : Input action and delay out, =0x2 : Input action and pulse out =0x3 : Input action and periodic out =0x4 : Timer action and delay out =0x5 : Timer action and pulse out =0x6 : Timer action and periodic out =0x7 : Timer off =0x8 : send to remote device0 =0x9 : send to remote device1 =0xA : send to remote device2 =0xB : send to remote device3 please refer: chapter 9 Standalone mode application examples
timer_value[0~3]	u8	time constant based on 100ms click tick. allowable range: 1 ~ 2^{32}(100ms ~ 100*2^{32}ms)

input_point[0] input_point[1]	u16	The input points your process wants to watch. The input_point[0] data high byte is un-used, low byte data (b7~b0) is IN7 ~ IN0 [0] : physical inport, IN7 ~ IN0 point [1] : virtual IO, IO17 ~ IO00 point
input_state[0] input_state[1]	u16	The input states that your process will trigger timer or output. The input_ state [0] data high byte is un-used, low byte data (b7~b0) is the state of IN7 ~ IN0 [0] : physical IO, IN7 ~ IN0 state [1] : virtual IO, IO17 ~ IO00 state
output_point[0] output_point[1]	u16	The output states that your process will trigger while the input states meet your preset. The output_point[0] high byte is un-used, low byte data (b7~b0) is the state of OUT7 ~ OUT0 [0] : physical outport, OUT7 ~ OUT0 state [1] : virtual IO, IO17 ~ IO00 state
output_mode	u8	output modes, which depends on the timer mode set. If control_mode = bypass or delay out, =0x0 : output inactive =0x1 : output active =0x2 : output change state If control_mode = pulse out, =0x0 : inactive state pulse =0x1 : active state pulse If control_mode = periodic out, =0x2 : output change state If control_mode = send to remote, =0x0 : output inactive =0x1 : output active
standalone_state	u8	=0x0 : power on standalone disable =0x1 : power on standalone enable

Ethernet module to Remote:

IP header	UDP header	UDP data		
		Data	Flag	0x54
20bytes	8bytes	32bytes	1byte	1byte

Command: 0x54

Flag: = 0x0 //command fail

= 0x63 //command successful

Data: un-used

- **READ_STANDALONE_CONFIG_NEW**

Function: read new standalone commands from EMD820X module

Remote to Ethernet module:

IP header	UDP header	UDP data			
		Module_name	Password	0x55	Data
20bytes	8bytes	7bytes	8bytes	1byte	32bytes

Module_name: 'E', 'M', 'D', '8', '2', '0', 'x'

Password: your password （8 bytes）

Command: 0x55

Data: _NEW_ST_DATA new_standalone_data; //new standalone instruction

Data: _NEW_ST_DATA new_standalone_data; // and operation mode

Parameter	Type	Description
function_index	u8	command line index of current communication. Actually one communication data will carry 2 commands. The function_index is the first command's line number. allowable range: 0 ~ 31 (command line index)

Ethernet module to Remote:

IP header	UDP header	UDP data		
		Data	Flag	0x55
20bytes	8bytes	32bytes	1byte	1byte

Command: 0x55

Flag: = 0x0 //command fail

= 0x63 //command successful

Data: _NEW_ST_DATA new_standalone_data; //new standalone instruction

Data: _NEW_ST_DATA new_standalone_data; // and operation mode

Parameter	Type	Description
function_index	u8	command line index of current communication. Actually one communication data will carry 2 commands. The function_index is the first command's line number. allowable range: 0 ~ 31 (command line index)
function_number	u8	Standalone mode total commands. allowable range: 1 ~ 32 (commands)
standalone_state	u8	=0x0 : power on standalone disable =0x1 : power on standalone enable
power_on_state	u8	=0x0 : power on standalone disable =0x1 : power on standalone enable
control_mode	u8	timer working mode of current command:

		=0x0 : Bypass =0x1 : Input action and delay out, =0x2 : Input action and pulse out =0x3 : Input action and periodic out =0x4 : Timer action and delay out =0x5 : Timer action and pulse out =0x6 : Timer action and periodic out =0x7 : Timer off =0x8 : send to remote device0 =0x9 : send to remote device1 =0xA : send to remote device2 =0xB : send to remote device3 please refer: chapter 9 Standalone mode application examples
timer_value[0~3]	u8	time constant based on 100ms click tick. allowable range: 1 ~ 2^{32}(100ms ~ 100*2^{32}ms)
input_point[0] input_point[1]	u16	The input points your process wants to watch. The input_point[0] data high byte is un-used, low byte data (b7~b0) is IN7 ~ IN0 [0] : physical inport, IN7 ~ IN0 point [1] : virtual IO, IO17 ~ IO00 point
input_state[0] input_state[1]	u16	The input states that your process will trigger timer or output. The input_ state [0] data high byte is un-used, low byte data (b7~b0) is the state of IN7 ~ IN0 [0] : physical IO, IN7 ~ IN0 state [1] : virtual IO, IO17 ~ IO00 state
output_point[0] output_point[1]	u16	The output states that your process will trigger while the input states meet your preset. The output_point[0] high byte is un-used, low byte data (b7~b0) is the state of OUT7 ~ OUT0 [0] : physical outport, OUT7 ~ OUT0 state [1] : virtual IO, IO17 ~ IO00 state
output_mode	u8	output modes, which depends on the timer mode set. If control_mode = bypass or delay out, =0x0 : output inactive =0x1 : output active =0x2 : output change state If control_mode = pulse out, =0x0 : inactive state pulse =0x1 : active state pulse

		If control_mode = periodic out, =0x2 : output change state If control_mode = send to remote, =0x0 : output inactive =0x1 : output active

- **CLEAR_STANDALONE_CONFIG**

Function: clear all standalone config

Remote to Ethernet module:

IP header	UDP header	UDP data			
		Module_name	Password	0x56	Data
20bytes	8bytes	7bytes	8bytes	1byte	32bytes

Module_name: ‘E’, ‘M’, ‘D’, ‘8’, ‘2’, ‘0’, ‘x’

Password: your password （8 bytes）

Command: 0x56

Data: un-used

Ethernet module to Remote:

IP header	UDP header	UDP data		
		Data	Flag	0x56
20bytes	8bytes	32bytes	1byte	1byte

Command: 0x56

Flag: = 0x0 //command fail

= 0x63 //command successful

Data: un-used

- **ENABLE_WDT**

Function: enable WDT (watchdog timer) operation of EMD820x module

Remote to Ethernet module:

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

Module_name: ‘E’, ‘M’, ‘D’, ‘8’, ‘2’, ‘0’, ‘x’

Password: your password （8 bytes）

Command: 0x60

Data: un-used

Ethernet module to Remote:

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

Command: 0x60

Flag: = 0x0 //command fail

= 0x63 //command successful

Data: un-used

- **DISABLE_WDT**

Function: disable WDT (watchdog timer) operation of EMD820x module

Remote to Ethernet module:

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

Module_name: ‘E’, ‘M’, ‘D’, ‘8’, ‘2’, ‘0’, ‘x’

Password: your password （8 bytes）

Command: 0x61

Data: un-used

Ethernet module to Remote:

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

Command: 0x61

Flag: = 0x0 //command fail

= 0x63 //command successful

Data: un-used

- **SET_WDT**

Function: setup WDT (watchdog timer) operation of EMD820x module

Remote to Ethernet module:

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

Module_name: ‘E’, ‘M’, ‘D’, ‘8’, ‘2’, ‘0’, ‘x’

Password: your password （8 bytes）

Command: 0x62

Data: Data.WDT. output , Data.WDT.timer_value
other structure members are un-used.

Parameter	Description
Data.WDT. output	OUT7 ~ OUT0 state at WDT time out bit7: 0: OUT7 is inactive at WDT time out 1: OUT7 is active at WDT time out … bit0: 0: OUT0 is inactive at WDT time out 1: OUT0 is active at WDT time out
Data.WDT.timer_value	WDT time constant at 0.1second time base allowable range: 10 (1/.0s) ~ 10000 (1000.0s)

Ethernet module to Remote:

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

Command: 0x62

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

Data: un-used

- **READ_WDT**

Function: read WDT (watchdog timer) counter of EMD820x module

Remote to Ethernet module:

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

Module_name: 'E', 'M', 'D', '8', '2', '0', 'x'

Password: your password （8 bytes）

Command: 0x63

Data: un-used

Ethernet module to Remote:

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

Command: 0x63

Flag: = 0x0 //command fail

= 0x63 //command successful

Data: Data.WDT.state , Data.WDT.output , Data.WDT.Timer_value

other structure members are un-used.

Parameter	Description
Data.WDT.output	OUT7 ~ OUT0 state at WDT time out bit7: 0: OUT7 is inactive at WDT time out 1: OUT7 is active at WDT time out … bit0: 0: OUT0 is inactive at WDT time out 1: OUT0 is active at WDT time out
Data.WDT.Timer_value	WDT time constant at 0.1second time base allowable range: 10 (1/.0s) ~ 10000 (1000.0s)
Data.WDT.state	0:WDT Disable 1:WDT Enable

- **SET_REMOTE_CONFIG**

Function: To map IP and PORT of an remote module to a specified remote_ID number.

Remote to Ethernet module:

IP header	UDP header	UDP data			
		Module_name	Password	0x70	Data
20bytes	8bytes	7bytes	8bytes	1byte	32bytes

Module_name: ‘E’, ‘M’, ‘D’, ‘8’, ‘2’, ‘0’, ‘x’

Password: your password （8 bytes）

Command: 0x70

Data: _REMOTE_CONFIG remote_config;

Parameter	type	Description
remote_config.IP[4]	u8	Remote device IP address Device default is 192.168.0.100
remote_config.remote_port	u16	Remote device socket port Device default is 6936
remote_config.module_type	u8	remote device type, // 1 : EMD8204, 2 : EMD8208, // 3 : EMD8216, 4 : EMC8485, // 5 : EMC8432, 6 : EMA8314R, // 7 : EMA8308, 8 : EMA8308D, // 9 : PC
remote_config.index	u8	Choose remote device ID ID = 0 ~ 3

Ethernet module to Remote:

IP header	UDP header	UDP data		
		Data	Flag	0x70
20bytes	8bytes	32bytes	1byte	1byte

Command: 0x70

Flag: = 0x0 //command fail

= 0x63 //command successful

Data: un-used

- **READ_REMOTE_CONFIG**

Function: To read IP and PORT mapped to an remote module of specified remote_ID number

Remote to Ethernet module:

IP header	UDP header	UDP data			
		Module_name	Password	0x71	Data
20bytes	8bytes	7bytes	8bytes	1byte	32bytes

Module_name: ‘E’, ‘M’, ‘D’, ‘8’, ‘2’, ‘0’, ‘x’

Password: your password （8 bytes）

Command: 0x71

Data: _REMOTE_CONFIG remote_config;

Parameter	type	Description
remote_config.index	u8	Choose remote device ID ID = 0 ~ 3

Ethernet module to Remote:

IP header	UDP header	UDP data		
		Data	Flag	0x71
20bytes	8bytes	32bytes	1byte	1byte

Command: 0x71

Flag: = 0x0 //command fail

= 0x63 //command successful

Data: _REMOTE_CONFIG remote_config;

Parameter	type	Description
remote_config.IP[4]	u8	Remote device IP address Device default is 192.168.0.100
remote_config.remote_port	u16	Remote device socket port Device default is 6936
remote_config.module_type	u8	remote device type, // 1 : EMD8204, 2 : EMD8208, // 3 : EMD8216, 4 : EMC8485, // 5 : EMC8432, 6 : EMA8314R, // 7 : EMA8308, 8 : EMA8308D, // 9 : PC
remote_config.index	u8	Choose remote device ID ID = 0 ~ 3

- **CLEAR_CONNECTION**

Function: To clear the connect information. It will disconnect communication before clear. If the standalone mode is enabled, this command is valid.

Remote to Ethernet module:

IP header	UDP header	UDP data			
		Module_name	Password	0x74	Data
20bytes	8bytes	7bytes	8bytes	1byte	32bytes

Module_name: 'E', 'M', 'D', '8', '2', '0', 'x'

Password: your password （8 bytes）

Command: 0x74

Data: connection_state[4];

Parameter	type	Description
remote_state.connection_state[4]	u8	Remote device IP address Device default is 192.168.0.100

Ethernet module to Remote:

IP header	UDP header	UDP data		
		Data	Flag	0x74
20bytes	8bytes	32bytes	1byte	1byte

Command: 0x74

Flag: = 0x0 //command fail

= 0x63 //command successful

Data: un-used

- **READ_CONNECTION_STATE**

Function: Read the connect state.

Remote to Ethernet module:

IP header	UDP header	UDP data			
		Module_name	Password	0x75	Data
20bytes	8bytes	7bytes	8bytes	1byte	32bytes

Module_name: ‘E’, ‘M’, ‘D’, ‘8’, ‘2’, ‘0’, ‘x’

Password: your password （8 bytes）

Command: 0x75

Data: un-used

Ethernet module to Remote:

IP header	UDP header	UDP data		
		Data	Flag	0x75
20bytes	8bytes	32bytes	1byte	1byte

Command: 0x75

Flag: = 0x0 //command fail

= 0x63 //command successful

Data: _REMOTE_STATE remote_state;

Parameter	type	Description
remote_state.connection_state[4]	u8	[0 ~ 3] is remote device ID connection_state = 0x0 is REMOTE_UNUSED 0x1 is CONNECTED 0x2 is RE_CONNECTION 0x3 is DISCONNECTION 0x4 is WAIT_RESPOND 0x5 is WAIT_TO_SEND 0x6 is SEND_TO_REMOTE
remote_state .connection_ERROR_code[4]	u8	[0 ~ 3] is remote device ID Error code

12. DLL list

	Function Name	Description
1.	EMD820x_initial( )	Assign IP and get model parameter
2.	EMD820x_close( )	EMD820x close
3.	EMD820x_firmware_version_read( )	Read the firmware version
4.	EMD820x_dll_version_read	Read the dll version
5.	EMD820x_subnet_mask_set	Set the subnet mask
6.	EMD820x_subnet_mask_read	Read the subnet mask
7.	EMD820x_port_polarity_set( )	Set polarity of input port or output port
8.	EMD820x_port_polarity_read( )	Read polarity of input port or output port.
9.	EMD820x_port_set ( )	Set the output port data.
10.	EMD820x_port_read( )	Read back the data of the I/O port.
11.	EMD820x_point_polarity_set( )	Set polarity of input port or output port.
12.	EMD820x_point_polarity_read( )	Read polarity of input port or output port
13.	EMD820x_point_set( )	Set bit status of output port
14.	EMD820x_point_read( )	Read bit state of input port.
15.	EMD820x_counter_mask_set( )	Set counter channel mask
16.	EMD820x_counter_mask_read( )	Read counter channel mask
17.	EMD820x_counter_enable( )	Enable counter function
18.	EMD820x_counter_disable( )	Disable counter function
19.	EMD820x_counter_read( )	Read counter value
20.	EMD820x_counter_clear( )	Clear designated counter
21.	EMD820x_change_socket_port( )	change the communication port
22.	EMD820x_change_IP( )	change the IP of EMD820x
23.	EMD820x_reboot( )	reboot EMD820x module
24.	EMD820x_security_unlock( )	Unlock security
25.	EMD820x_security_status_read( )	Read lock status
26.	EMD820x_password_change( )	Change password
27.	EMD820x_password_set_default( )	Rest to factory default password
28.	EMD820x_WDT_set( )	Set WDT(watch dog timer) configuration
29.	EMD820x_WDT_read( )	Read back WDT(watch dog timer) configuration.
30.	EMD820x_WDT_enable( )	enable WDT(watch dog timer)
31.	EMD820x_WDT_disable( )	disable WDT (watch dog timer)
32.	EMD820x_standalone_enable( )	Enable standalone
33.	EMD820x_standalone_disable( )	Disable standalone
34.	EMD820x_standalone_V_config_set( )	Set standalone configuration

35.	EMD820x_standalone_V_config_read( )	Read standalone configuration
36.	EMD820x_standalone_config_clear( )	Clear standalone configuration
37.	EMD820x_D2D_config_set( )	Set the remote configuration
38.	EMD820x_D2D_config_read( )	Read the remote configuration
39.	EMD820x_D2D_connection_clear( )	Clear the remote configuration setting
40.	EMD820x_D2D_connection_state_read( )	Read the remote configuration connection state

13. EMD820x Error codes summary

13.1 EMD820x 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_ERROR	The module locked
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	INPOINT_ERROR	Input point unreachable
303	OUTPOINT_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_OUTERROR	Too long to response
311	CREAT_SOCKET_ERROR	Socket can not create
312	CHANGEIP_ERROR	Can not change IP
313	MASK_CHANNEL_ERROR	Can not set counter mask
314	COUNTER_ENABLE_ERROR	Can not enable counter
315	COUNTER_DISABLE_ERROR	Can not disable counter
316	COUNTER_READ_ERROR	Can not read counter data
317	COUNTER_CLEAR_ERROR	Can not clear counter data
318	TIME_ERROR	connection response over time
321	STANDALONE_ENABLE_ERROR	Can not enable standalone mode
322	STANDALONE_DISABLE_ERROR	Can not disable standalone mode
323	STANDALONE_CONFIG_ERROR	Can not configure standalone mode
324	INDEX_ERROR	Standalone command index error
325	NUMBER_ERROR	Standalone command step number error
326	TIMER_MODE_ERROR	Standalone timer mode error

327	OUT_MODE_ERROR	Standalone output mode error
328	POWER_ON_ERROR	Standalone power on mode error
340	CARD_TYPE_ERROR	Standalone remote type error
341	REMOTE_DEVICE_ERROR	Standalone remote device connection error

14. UDP communication command list

command code	R/W	Mnemonics	Descriptions
0x1	R	GET_MODULE_TYPE	Get Module Type
0x2	W	REBOOT	Soft Reboot
0x3	W	CHANGE_SOCKETPORT	Change Socket Port
0x4	W	CHANGE_PASSWORD	Change password
0x5	W	RESTORE_PASSWORD	Restore password
0x6	W	CHANGE_IP	Change IP Address
0x7	R	GET_FIRMWARE_VERSION	Get firmware version
0x8	W	CHANGE_SUBNET_MASK	Change subnet mask
0x9	R	READ_SUBNET_MASK	Read subnet mask
0xFA	W	WRITE_MAC	Write MAC Address to EEPROM
0x20	W	SET_COUNTER_MASK	Set counter mask
0x21	W	ENABLE_COUNTER_MODE	Enable counter mode
0x22	W	DISABLE_COUNTER_MODE	Disable counter mode
0x23	R	READ_COUNTER	Read counter
0x24	W	CLEAR_COUNTER	Clear counter
0x32	W	SET_PORT	Set Port
0x33	R	READ_PORT	Read Port
0x34	W	SET_PORT_POLARITY	Set port polarity
0x35	R	READ_PORT_POLARITY	Read port polarity
0x38	W	SET_POINT	Set Point
0x39	R	READ_POINT	Read Point
0x3A	W	SET_POINT_POLARITY	Set Point polarity
0x3B	R	READ_POINT_POLARITY	Read Point polarity
0x3C	W	WRITE_MULTI_POINT	Write multi point
0x3D	R	READ_MULTI_POINT	Read multi point
0x50	W	ENABLE_STANDALONE	Enable standalone function
0x51	W	DISABLE_STANDALONE	Disable standalone function
0x54	W	SET_STANDALONE_CONFIG_NEW	Set new standalone config
0x55	R	READ_STANDALONE_CONFIG_NEW	Read new standalone config
0x56	W	CLEAR_STNADALONE_CONFIG	Clear standalone config
0x60	W	ENABLE_WDT	enable WDT operation
0x61	W	DISABLE_WDT	disable WDT operation
0x62	W	SET_WDT	setup WDT
0x63	R	READ_WDT	read WDT

0x70	W	SET_REMOTE_CONFIG	Set remote device config
0x71	R	READ_REMOTE_CONFIG	Read remote device config
0x74	W	CLEAR_CONNECTION	Abort and clear Connection
0x75	R	READ_CONNECTION_STATE	Read Connection state

15. Error codes table for UDP Success_flag

Error code	Symbolic Name	Description
99	SUCCESS	No Error
100	COMMAND_ERROR	Command Error
101	PASSWORD_ERROR	Password Error
102	CHANGE_IP_ERROR	Set IP value error out of range Range：LSB = 1 ~ 254
103	CHANGE_SOCKET_ERROR	Set socket port error out of range Range：value > 1000
104	CHANGE_MAC_ERROR	Set mac address error mac != FF FF FF FF FF FF
120	PORT_ERROR	Choose Port error out of range Range：port < port_max
121	POINT_ERROR	Choose port error out of range Range：port < port_max
122	STATE_ERROR	Set state error State = 1 or 0
123	TIMER_VALUE_ERROR	Set Timer value error out of range Range：value_min < value < value_max
124	MODE_ERROR	Choose mode error out of range Range：mode < mode_max