《egPLC6888 User Manual》

Overview

egPLC6888 is a programmable PLC/RTU device applied to industrial application for data acquisition and control actuation, which is integrated with signal sampling, signal processing, and industrial control actuation. It is operated in low power with high efficiency, and anti-interference features, being able to do industrial process control and build customized equipment. egPLC6888 is embedded with a powerful 32bit microprocessor for application customization. Communication through RS485 provides a flexible and powerful topology for network building with local/remote sensing and control. Through simple and state machine based C language customization, egPLC6888 is much suitable to accomplish application target in traditional PLC application fields.

egPLC6888 PLC/RTU facilitates 28 physical IO ports

- 1 USB device port, used for egPLC6888 program download and PC windows API access to this device

- 2 RS485 master port, used for accessing local and remote RS485 slave devices and instruments

- 1 RS485 Modbus slave port (configurable as master port), used for connecting local/remote RS485 master or PC API access to this device

- 8 Current ADC channels, used to sample 4-20 mA analogue current signals (IO pins shared with analogue voltage ADC channels)

- 8 Voltage ADC channels, used to sample 0-5V analogue voltage signal (IO pins shared with analogue current ADC channels)

- 4 Counter sampling input channels, used for pulse signal counting (IO pins shared with digital input channels)

- 8 Digital signal input channels, used to sample digital voltage signal

- 8 Solid state relay output channels, used for control actuation. The first 4 out of which are common pins for 4 PWM (Pulse-Width-Modulation) channels

IO ports are separate from CPU with photo-electrical couple protection, and the maximal IO input tolerant voltage is no less than 24VDC.

egPLC6888 power supply is designed with extra wide range, it can be operated normally between 9VDC and 24VDC, and 12VDC power supply is recommended.

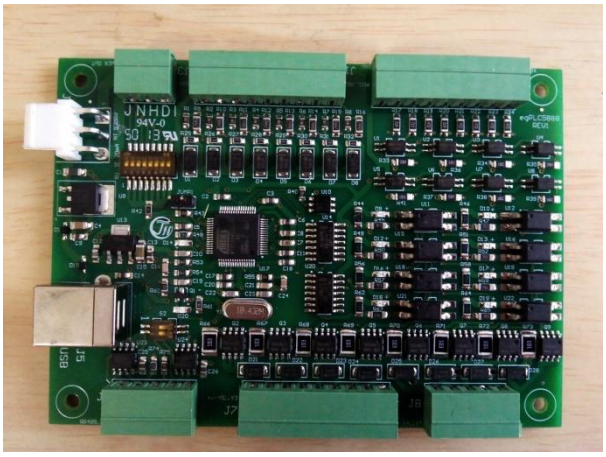


Figure 1. Real photo image of egPLC6888

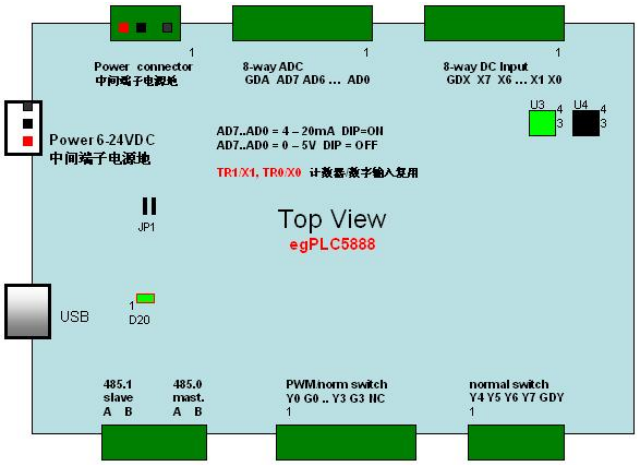


Figure 2. egPLC6888 and its IO ports layout

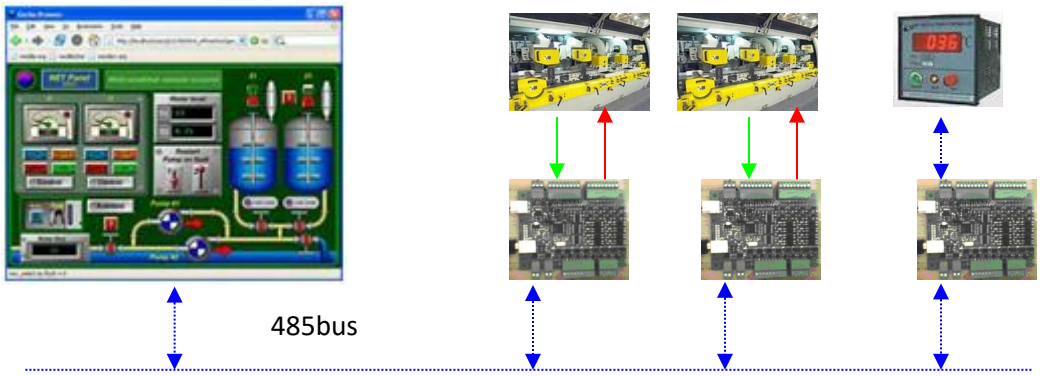


Figure 3. Application example of egPLC6888 network topology, also standalone equipment

Electrical character of egPLC6888 IO pins

RS485 Slave port: Differential IO signal input/output A and B, with 8-bit data format, 1-bit stop, no bit parity. The communication baud rate and other parameters can be set through "Visual State Machine" software utility. RS485 slave port is embedded with Modbus protocol for local/remote master machine to access any egPLC6888 internal register. RS485 port conforms to RS485 Modbus RTU protocol, which is robust and can be operated in harsh environment for industrial devices communication.

RS485 Master port: Differential IO signal input/output A and B, with 8-bit data format, 1-bit stop, no bit parity. The communication baud rate and other parameters can be set through "Visual State Machine" software utility and programmatically. The egPLC6888 can access slave RS485 devices clustered on the same RS485. RS485 master port conforms to RS485 bus protocol in hardware, which is robust and can be operated in harsh environment for bus signal communication.

8 solid state relays output Y0..Y7 with each channel sustaining 36VDC and 4A electrical power burst. However, a mechanical middle relay is recommended for power circuit control. A typical application is as follows (fuse is optional):

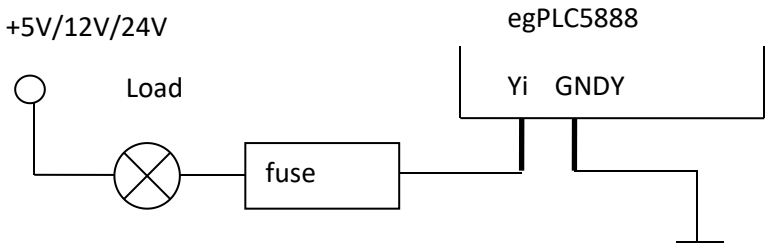


Figure 4. Connection diagram for output control

The load is selected according to application requirement. The lowest driving voltage for load is recommended +5VDC, user can raise the load voltage and power dissipation according to requirement. The optional fuse is for current overload protection. Proper selection of such measurements is for protection purpose.

egPLC6888 has 8 solid state relay outputs, 4 out of which has high-speed pulse PWM features. The output frequency and duty cycle can be controlled by VSM embedded C language. To control IO output one can realize this through PC USB API and 485 Modbus access.

egPLC6888 has 8 digital channels X0..X7. When external input voltage is greater than 3.6V, the input digital signal is interpreted as high. When the external voltage signal is less than 2V, the input digital signal is interpreted as low. The recommended input signal should be greater than 5V as to demonstrate high level and lower than 1V as to demonstrate low level. Each input can sustain DC voltage up to 36V.

Apart from this function, the first four channels, X0, X1, X2, X3 have pulse counting ability. This input pulse counting function is a parallel hardware design, although they physically share the same pin as digital level input. The counter can be read/set/reset via the access of connected internal registers by embedded C code. Moreover users can access this hardware function via USB port and RS485 Modbus slave port with remote PC API.

egPLC6888 has 8 analogue input channels AD0..AD7. egPLC6888 has both analogue current and voltage sample function. The input types can be selected by dip switch configuration. It is configured as voltage input by factory default. The voltage analogue input ranges from 0 to 5VDC and its resolution is 12 bits, which means the measurement can reach as accurate as 0.025% of 5V. The maximal tolerant input voltage is 20VDC and each channel input resistor 5K Ohm. For the analogue current input the maximal tolerant input voltage is 10VDC and its internal input resistance is 250 Ohm.

USB port can communicate with windows/Linux PC host via HID protocol. egPLC6888 firmware has built in USB HID protocol and performs plug and play without extra host device driver. Armed with egPLC6888 USB API, this device can be used to build PC-based standalone testing machine or various equipment. As such application has been demonstrated in many scenarios.

egPLC6888 IO space register address

egPLC6888 has facilitated 64K IO register space. All registers are 32-bit if they are not notably specified. All addresses are aligned with 4 byte address. All registers in memory storage is complied with big endian format.

Reg. address (hex)	Reg. Mnemonic	Interpretation	Read/write mode
0000 .. 0FFC	IO_MEM	1024 32-bit general registers	read/write
1000	AD0	12-bit ADC result reg.	Read only
1004	AD1	12-bit ADC result reg.	Read only
1008	AD2	12-bit ADC result reg.	Read only
100C	AD3	12-bit ADC result reg.	Read only
1010	AD4	12-bit ADC result reg.	Read only

1014	AD5	12-bit ADC result reg.	Read only
1018	AD6	12-bit ADC result reg.	Read only
101C	AD7	12-bit ADC result reg.	Read only
2000	TR[0].CNT	16-bit counter	Read/write, Write 0 to clear
200C	TR[0].RUN	Write 1 to start to count, 0 to remain	Read/Write, share with X0
2010	TR[1].CNT	16-bit counter	Read/write, Write 0 to clear
201C	TR[1].RUN	Write 1 to start to count, 0 to remain	Read/Write, share with X1
2020	TR[2].CNT	16-bit counter	Read/write, Write 0 to clear
202C	TR[2].RUN	Write 1 to start to count, 0 to remain	Read/Write, share with X2
2030	TR[3].CNT	16-bit counter	Read/write, Write 0 to clear
203C	TR[3].RUN	Write 1 to start to count, 0 to remain	Read/Write, share with X3
3000	X0	1-bit digital input reg.	Read only
3004	X1	1-bit digital input reg.	Read only
3008	X2	1-bit digital input reg.	Read only
300C	X3	1-bit digital input reg.	Read only
3010	X4	1-bit digital input reg.	Read only
3014	X5	1-bit digital input reg.	Read only
3018	X6	1-bit digital input reg.	Read only

301C	X7	1-bit digital input reg.	Read only
4000	Y0	1-bit digital output reg.	Read/write
4004	Y1	1-bit digital output reg.	Read/write
4008	Y2	1-bit digital output reg.	Read/write
400C	Y3	1-bit digital output reg.	Read/write
4010	Y4	1-bit digital output reg.	Read/write
4014	Y5	1-bit digital output reg.	Read/write
4018	Y6	1-bit digital output reg.	Read/write
401C	Y7	1-bit digital output reg.	Read/write
5000	PWM[0].FREQ	16-bit PWM frequency register, 50..1000	read/write
5004	PWM[0].DUTY	16-bit duty reg. 0..100	read/write
5008	PWM[0].POL	Waveform phase reg.	read/write
500C	PWM[0].RUN	1-bit run/stop register	read/write
5010	PWM[1].FREQ	16-bit PWM frequency register, 50..1000	read/write
5014	PWM[1].DUTY	16-bit duty reg. 0..100	read/write
5018	PWM[1].POL	Waveform phase reg.	read/write
501C	PWM[1].RUN	1-bit run/stop register	read/write
5020	PWM[2].FREQ	16-bit PWM frequency register, 50..1000	read/write
5024	PWM[2].DUTY	16-bit duty reg. 0..100	read/write
5028	PWM[2].POL	Waveform phase reg.	read/write
502C	PWM[2].RUN	1-bit run/stop register	read/write

5030	PWM[3].FREQ	16-bit PWM frequency register, 50..1000	read/write
5034	PWM[3].DUTY	16-bit duty reg. 0..100	read/write
5038	PWM[3].POL	Waveform phase reg.	read/write
503C	PWM[3].RUN	1-bit run/stop register	read/write
6000	TIMER[0]	32-bit clock, in seconds	read/write
6004	TIMER[1]	32-bit clock, in seconds	read/write
6008	TIMER[2]	32-bit clock, in seconds	read/write
600C	TIMER[3]	32-bit clock, in seconds	read/write
6010	TIMER[4]	32-bit clock, in seconds	read/write
6014	TIMER[5]	32-bit clock, in seconds	read/write
6018	TIMER[6]	32-bit clock, in seconds	read/write
601C	TIMER[7]	32-bit clock, in seconds	read/write
6020..603C	TIMER[8..f]	32-bit clock, in seconds	Read/write
6100	TIMER[10]	32-bit clock, in ms	Read/wite
6104	TIMER[11]	32-bit clock, in ms	Read/wite
6108	TIMER[12]	32-bit clock, in ms	Read/wite
610C	TIMER[13]	32-bit clock, in ms	Read/wite
6110	TIMER[14]	32-bit clock, in ms	Read/wite
6114	TIMER[15]	32-bit clock, in ms	Read/wite
6118	TIMER[16]	32-bit clock, in ms	Read/wite
611C	TIMER[17]	32-bit clock, in ms	Read/wite
6120..613C	TIMER[18..1f]	32-bit clock, in ms	Read/wite

C000/C004/C008	IDAT (com0/1/2)	RS485 master read port	read only
C100..C3FF C400..C7FF C800..CBFF	ODAT(com0/1/2)	768/1024/1024-Byte RS485 output data write port buffer	write only
C030/C034/C038	CDAT(com0/1/2)	Number of output bytes to send	write only, trigger to send
C040/C044/C048	BAUD(com0/1/2)	Receive/send baud rate	write only, set baud rate
C050/C054/C058	MODE(com0/1/2)	Mode register, refer to following table 2	Write only
E000	AD0-AD1	RS485 AD condensed channel (16-bit each)	read only, with ADC1 higher word
E004	AD2-AD3	RS485 AD condensed channel (16-bit each)	read only, with ADC3 higher word
E008	AD4-AD5	RS485 AD condensed channel (16-bit each)	read only, with ADC5 higher word
E00C	AD6-AD7	RS485 AD condensed channel (16-bit each)	read only, with ADC7 higher word
E010	XY	------X7..X0------Y7..Y0 32bits, the bitmap for 8-input for Xi and 8-output for Yi.	read only
F000 .. FFFC	NVR	1024 32-bit non-volatile registers	read/write

Table 1. egPLC6888 OI space definition

Communication mode	Definition	Description	Examples,data:parity:stop
Bit<3..0>	The stop time for a datum transferred	0：for 1-bit stop	0x8020, 8:N:1
		1：for 1.5-bit stop	0x8021, 8:N:1.5
		2：for 2-bit stop	0x8022, 8:N:2
Bit<7..4>	Parity mode for a datum	0：for even parity	0x8000, 8:E:1
		1：for odd parity	0x8010, 8:O:1
		2：none parity	0x8020, 8:N:1
Bit<11..8>	Reserved		
Bit<15..12>	Bits for each atomic sending	8：8-bit for a datum	0x8020, 8:N:1
		7：7-bit for a datum	0x7020, 7:N:1

Table 2. RS485 communication mode definition

IO_MEM register space: 0x0000 – 0x0FFF, 4K bytes in total, 1024 32-bit registers, which have read and write functions. These can be accessed by customer application C code or by PC API, also can be used for data communication synchronous for multi-port/multi-device access.

ADC IO space AD0..AD7: 0x1000 – 0x101F, 32 bytes in total, 8 32-bit ADC sampling registers, read only. Data sampling process is automatically carried out by hardware. User can access them at any time, and what user read are the latest sampling results.

Counter register space: TR0 to TR3. Their IO space address is 0x1000 and 0x1030.

0x1000 and 0x100C are the first counter register and its enable register； 0x1010 and 0x101C are the second counter register and its enable register, and so on. The enable registers can be used to start/stop the counter register. Their IO Pins are shared with X0, X1, X2, X3, and their count registers can be read and written. The counting trigger of each channel is the rising edge of the IO pin.

Digital signal IO space X0..X7: 0x3000 – 0x301F, 32 bytes in total, 8 channels, 4-byte for each channel, read only, only the least significant bit is defined for a channel.

Digital signal output (solid state relay) IO space Y0..Y7: 0x4000 – 0x401F, 32 bytes in total, 8 channels, 4-byte for each channel, read/write, only the least significant bit is defined for a channel.

PWM output (solid state relay) IO space: 0x5000 – 0x503F, 64 bytes in total, 4 channels, 16 bytes for each channel, with 4 field registers for each channel, which are frequency, duty cycle, phase polarity, and start/stop register. The PWM output frequency ranges from 50Hz to 1kHz, the duty cycle ranges from 0 to 100. Although other numbers can be written to field frequency and duty registers, their register values will be auto clamped when field run register is written to 1, as a result this switches the pin output from Yi to PWMi with the wave form generated.

System clocks TIMER0..15 are 16 timers counting in seconds; TIMER16..TIMER31 are 16 timers counting in milliseconds. They can be read/write at any time and they keep counting all the time.

RS485_0 IO space: 0xC000 input data read port. When a read attempt returns -1 no bytes available in input buffer. Every read access returns one byte.

0xC100 output data write port, with 768 bytes buffer.

0xC030 data send port, write only, to instruct the number of bytes to send from the output data buffer register from 0xC100 to 0xC3FF.

0xC040 the baud rate register, the baud rate setting here is temporarily stored in RAM.

0xC050 the mode register, serial data bits for a byte, parity type, and stop bits.

RS485_1 IO space: 0xC004 input data read port. When a read attempt returns -1 no bytes available in input buffer. Every read access returns one byte.

0xC400 output data write port, with 1K bytes buffer.

0xC034 data send port, write only, to instruct the number of bytes to send from the output data buffer register from 0xC400 to 0xC7FF.

0xC044 the baud rate register, the baud rate setting here is temporarily stored in RAM.

0xC054 the mode register, serial data bits for a byte, parity type, and stop bits.

RS485_2 IO space: 0xC008 input data read port. When a read attempt returns -1 no bytes available in input buffer. Every read access returns one byte.

0xC800 output data write port, with 1K bytes buffer.

0xC038 data send port, write only, to instruct the number of bytes to send from the output data buffer register from 0xC800 to 0xCBFF.

0xC048 the baud rate register, the baud rate setting here is temporarily stored in RAM.

0xC058 the mode register, serial data bits for a byte, parity type, and stop bits.

IO space access is via IN/OUT instructions of embedded microprocessor, the egPLC6888 system provide C language subroutine interface for user development.

RS485 ModbusRTU Protocol

egPLC6888 RS485 slave port has a ModbusRTU communication protocol built-in, which enables other master device to communicate with this device using RS485 data bus. egPLC6888 Modbus command is simplified to 2 function categories, 0x03 and 0x10 which are register read and register write operations. egPLC6888 RS485 Slave receives remote master PC Modbus commands, then processes it and returns immediately. 2 command formats, read command (function code 0x03) and write command (function code 0x10) are implemented.

The command and data frame are represented with byte code fashion in Modbus RTU format, using 2 hexadecimal digits to represent one byte. In the following examples, we will assume the egPLC6888 485 slave address 0xaa.

Remote PC read egPLC6888 register operation= {addr, 03, sah, sal, regsh, regsl, crcl, crch }

egPLC6888 returns = {addr, 03, nbytes, w1h, w1l, w0h, w0l, …, crcl, crch} where the sequential 4 bytes quadruples form one 32-bit long word (w1h,w1l,w0h,w0l), in big-endian format in both bus transmittal and register storage state.

Where sah is the higher byte of target register address, sal is the lower byte of target register address, regsh is the higher byte number of 16-bit registers to access, regsl is the lower byte number of 16-bit registers to access, crcl lower byte of CRC, crch is higher byte of CRC, w1h is the most significant byte of the 4-byte register content, w0l is the least significant byte of the 4-byte register content.

Example 1: master PC read egPLC6888 Y0 digital outputs（solid state relay）status.

PC outgoing data frame: aa 03 40 00 00 02 c8 10

Returned from egPLC data frame: aa 03 04 00 00 00 01 21 39

Example 2: master PC read egPLC6888 AD0 analogue input for current/voltage.

PC outgoing data frame: aa 03 10 00 00 02 d9 10

Returned from egPLC data frame: aa 03 04 00 00 01 ab a0 d6

Remote PC write egPLC register operation= {addr, 10, sah, sal, regsh, regsl, nbytes, datH, datL,…, crcl, crch }

egPLC returns= { addr, 10, sah, sal, regsh, regsl, crcl, crch }

where sah is the higher byte of register address, sal is the lower byte of the register address, crcl is the lower byte of CRC, crch is the higher byte of CRC, regsh, regsl are the numbers of the registers for access (here every 32-bit register is expressed as 2 16-bit registers formally), nbytes is the real number of data bytes to write.

Example 3: Master PC writes egPLC6888 Y0 digital output (solid state relay) status to 0 (turn on).

PC outgoing data frame: aa 10 40 00 00 02 04 00 00 00 00 e5 4a

Returned from egPLC data frame: aa 10 40 00 00 02 4d d3

Example 4: Master PC write egPLC6888 Y1 digital output (solid state relay) status to 0 (turn on).

PC outgoing data frame: aa 10 40 04 00 02 04 00 00 00 00 e4 b9

Returned from egPLC data frame: aa 10 40 04 00 02 0c 12

egPLC6888 RS485 communication baud rate can be set through “visual state machine” software utility. egPLC6888 RS485 both master and slave ports use the same default communication settings. However, for RS485 master port, the communication parameters can be altered by user embedded C code.

egPLC6888 Embedded C Programming

egPLC6888 is a very powerful PLC. Unlike most traditional PLC, where the application target is realized in ladder diagram egPLC6888 is completely realized in C language. It is not only good at combination logic network but also good at calculation, data management and protocol customization. egPLC6888 mimics a minicomputer, with a set of peripheral IO interfaces to do sensing control, and actuation. egPLC6888 employs user defined state machine by PC windows utility to support C language application development. egPLC6888 reserves 32KB RAM space for user program stack and variables, and also system reserves 128KB flash memory for user binary code downloading. The development tool chain is

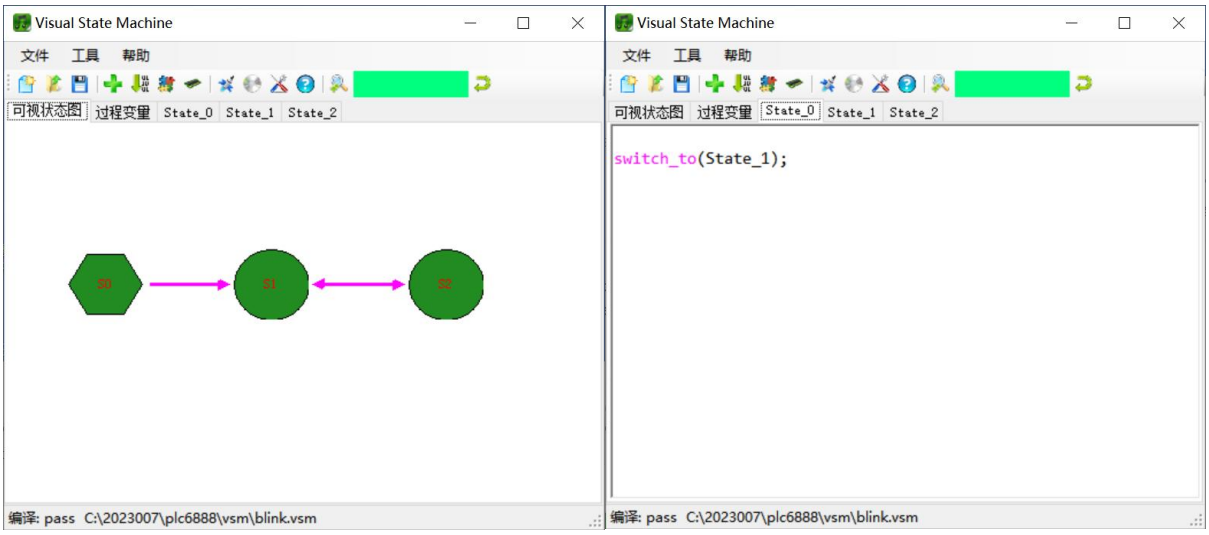
C Compiler: cc105.exe

Assembly language assembler: as105.exe

Simulator: sim105.exe

Binary code downloader: fsh4888.exe

Users can use visual state machine tool set to develop egPLC6888 embedded application, to realize the board-level customization, a graphics interface IDE utility “Visual State Machine” to develop the customized program. The following is the introduction of this IDE development utility.



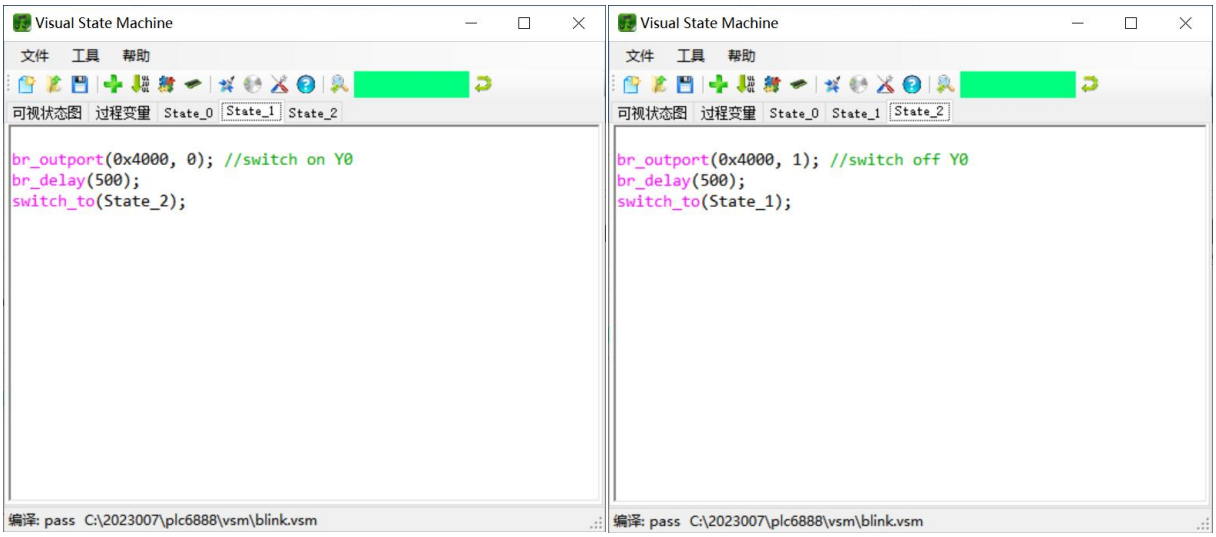


Figure 5. a glimpse of user program development using visual state machine.

This very classic blink program can be compiled and downloaded egPLC6888 via GUI utility. Once this program is downloaded through USB cable, the egPLC6888 is running and performing user-defined task.

The above program C code is equivalent to the following code:

```
#include <syslib.h>
void State_0(void);
void State_1(void);
void State_2(void);

void main(void) { switch_to(State_0);}

void State_0(void) {
switch_to(State_1);
}

void State_1(void) {
br_outport(0x4000, 0); //switch on Y0
br_delay(500);
switch_to(State_2);
}

void State_2(void) {
br_outport(0x4000, 1); //switch off Y0
br_delay(500);
switch_to(State_1);
}
```

The core mechanism of the egPLC6888 is the finite state machine. In each state, the feature and task is expressed in C language. The underneath multi-task os running inside egPLC6888 supports and manages the state transfer switch and IO request.

The C development tool set provide a mini set of embedded library. The following table list API functions:

prototype	parameters	Functionality	comment
void switch_to(long addr)	long addr -- the target address of destination state	This function realizes the state machine transition from source state to target state, where user program running from one state jumping to next state. The addr is the address for the target state, being a label or a function name.	
void sleep_1_tick(void)	none	Gives up cpu execution for 1 tick time. 1 tick time is 10ms interval in egPLC6888	
void br_delay(int ms)	int ms -- to sleep current program in ms millisecond amount	This function realizes the user program controlled delay in milliseconds. egPLC6888 user delay resolution is 10 milliseconds. The parameter “ms” is recommended to be multiple of 10.	
long br_inport(long addr)	long addr -- the register address of egPLC6888	This function reads a register content from IO register space. Parameter “addr” is 32-bit integer for the IO space register address.	
void br_outport(long addr, long dat)	long addr -- the register address of egPLC6888 long dat -- the data to store the register	This function writes a 32-bit integer to register content from IO space. The parameter “addr” is the address of the target IO space register.	
void qdat_mem_copy(char *desc_io, char	char *desc_io -- IO space register address char *src_mem -- C program	This function copies a block of data from source area to destination area, the address can be memory address and IO	

*src_mem, int cnt)	variable address int cnt -- the total number of bytes to copy	buffer address.	

Table 4. system library for visual state machine C programming

The above 6 functions are basic API for users to develop C program by Visual State Machine utility. These tiny C prototypes are defined in syslib.h, and syslib.asm is their assembly implementation. The file is automatically linked to build a final binary target. Once this binary file blink.hex is downloaded over USB, the user program is running flashed and won't be lost for new power cycles.

C language programming development environment is provided by Visual State Machine software package.

egPLC6888 Graphic User Interface development tool — Visual State Machine

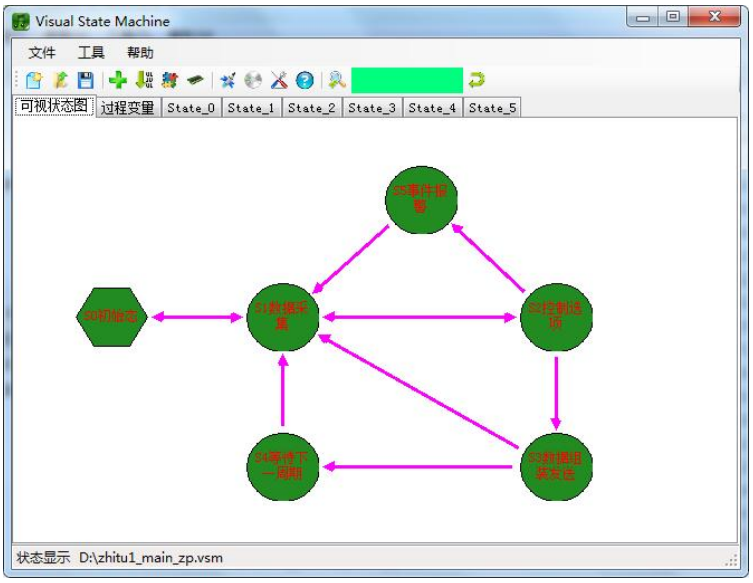


Figure 6. State transition diagram for a real application

Visual State Machine (VSM in brief) provides a GUI C language development environment. Compared with other IDE C language development environment, this software package provides users with more freedom and simplicity for C language development. Users do not need to code a complete C program, instead, users are only required to focus on the state transition and their logic, the complete C program of the finite state machine is automatically generated by the software system. Finite state machine is the software architecture of egPLC6888 series product, which offers a clear and easy way to code and customize various data acquisition and control actuation and communication programs. Finally this will do users great benefit for clear logic and efficiency improvement for problem solving.

VSM is especially prepared for egPLC compatible series devices with built-in features for board-level function customization in C language. egPLC6888 is very much suitable for industrial process control and remote data acquisition, and all the same building standalone mechanical and electrical equipment.

VSM treats every state a sub-routine. The transition from state to state is through switch_to() function call. The foreground state transition diagram and the code body for every state is encapsulated and completed with main() entry point and <syslib.h> header in background by the software system. The complete program is eventually compiled and assembled into binary machine code.

VSM software function includes state graph edit and C code edit. VSM also integrates C language development tool chain, including C grammar check, C program compilation, assembly, and binary code downloading. For more details, please refer to VSM software for more egPLC6888 C language programming discussion.

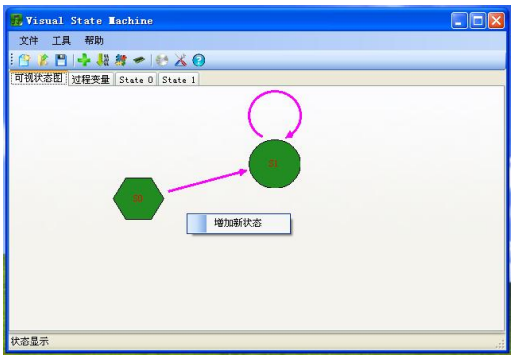


Figure 7. To add a new state by right-click

The above VSM diagram shows the new state is created and added to the state machine. The “visual state graph” tab display the finite state machine with transition from state to state. We can edit attributes for every state, setting the state name or transitional path.

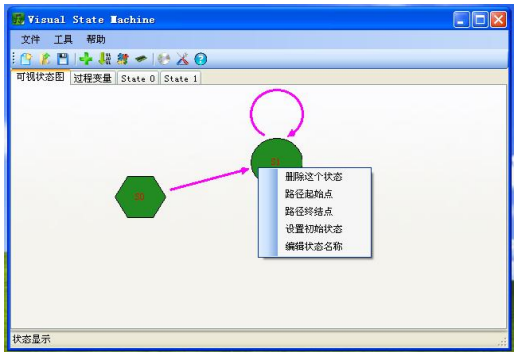


Figure 8. Change the property of a states

VSM menu function includes cleaning program from egPLC6888 and setting the baud rate and Modbus slave address of RS485 parameters.

Figure 9. Setting RS485 communication parameters

After VSM project file is created, we can edit the visual state graph and C code through the software utility menu functions. They are grammar checkup, compilation, and assembly and program download.

VSM programming example 1 is to control the digital output Y7, output '0' (close the relay) for 1 second, then output '1' (open the relay) for 0.5 seconds. And then repeat this behavior.

State	Output reg. addr.	Output data val.	Delay time	Next State	remark
State_0				State_1	initialization
State_1	0x401c	0	1000ms	State_2	relay on
State_2	0x401c	1	500ms	State_1	relay off

Table 5. State transition of the example

The visual state graph and code for each state are as follows.

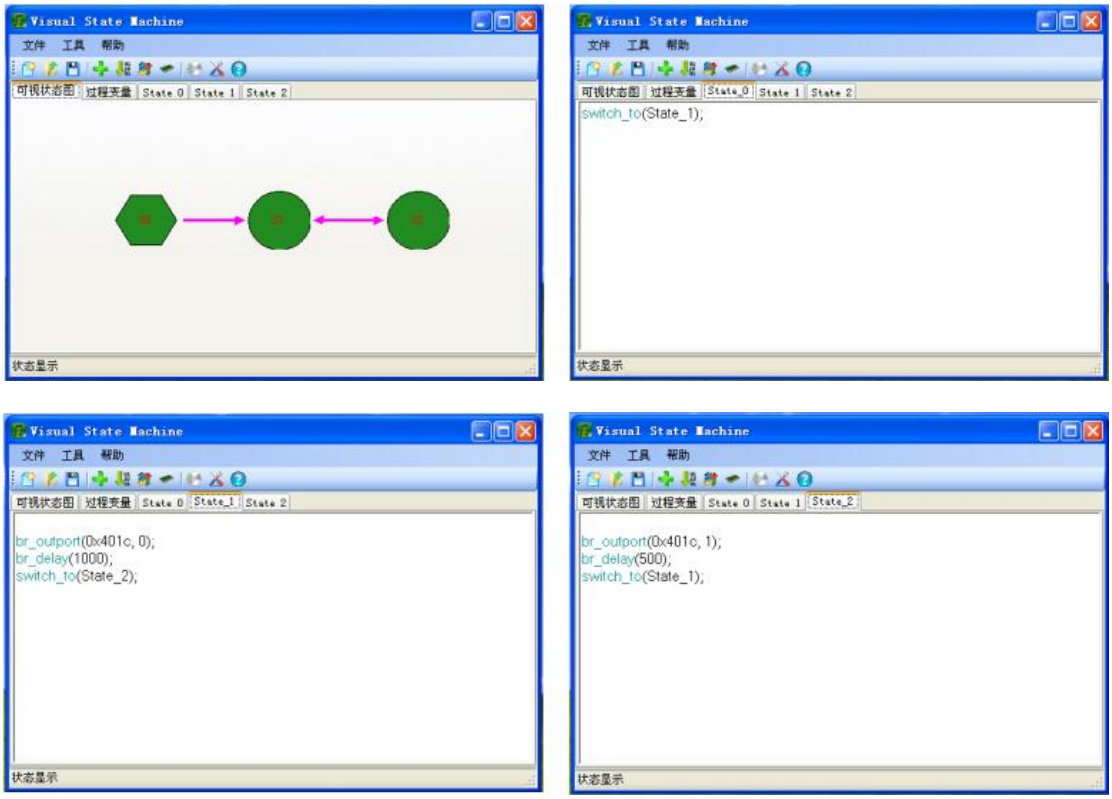


Figure 10. Example 1, blinking

One can watch the PLC6888 output indicator led for digital channel 7 (Y7)

VSM programming example 2, watch the digital input channel X7 status, and output to Y7.

State	Input reg. addr.	Output reg. addr.	Variable name	remark
State 1	0x301c	0x401c	tmp	

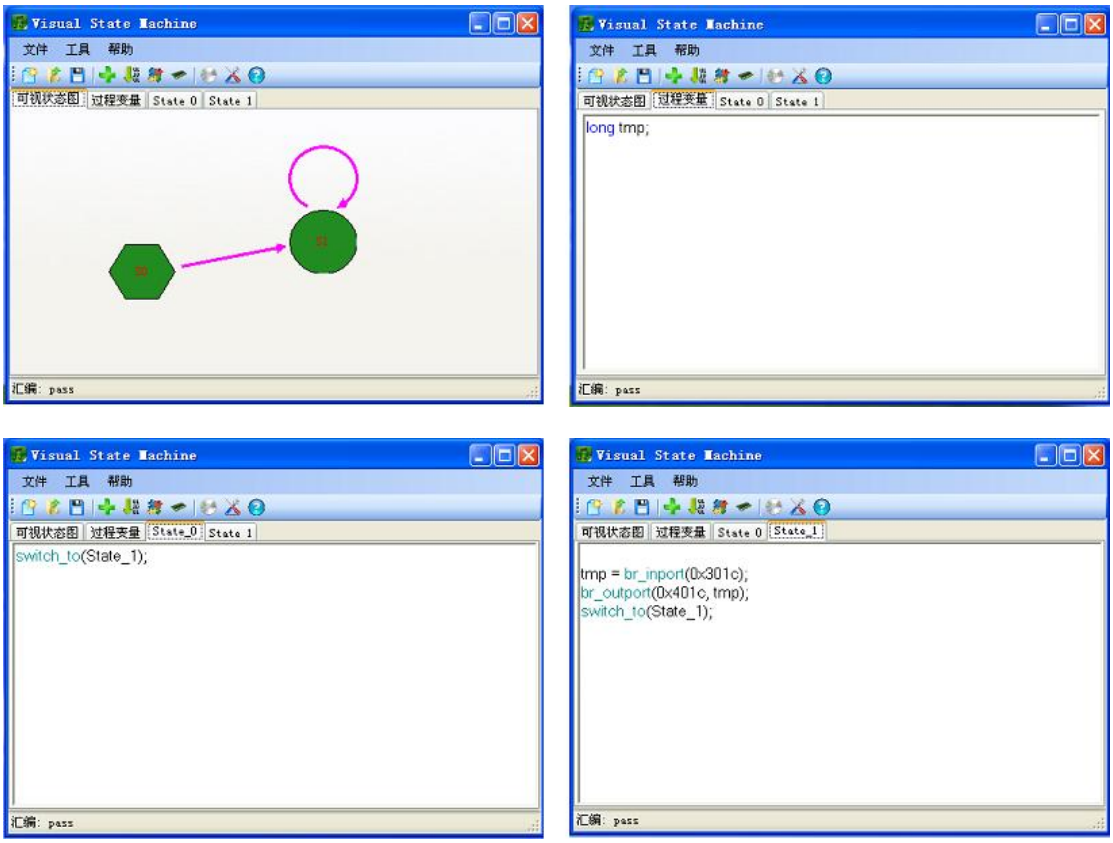


Figure 11. Example 2, PWM output

When the input X7 is connected to 5V voltage or more, the return result is “1”. When X7 is connected to 1V voltage or less, the return result is “0”. The output Y7 reflects input X7.

Visual state machine programming example 3 is to realize high frequency signal output via PWM through the pin Y1. Y0..Y3 output pins can be used for normal solid state relay output and also can be used for high speed switch frequency output. We use PWM interface to implement the high speed switch output. An external source of power is required to work with the circuit loop.

Every PWM channel has 4 registers to define PWM characters. The 4 channels are independent and being able to work simultaneously.

Output frequency register: address offset is 0x00, to set output waveform frequency.

Duty cycle ratio register: address offset is 0x04, value range from 0 to 100 in percentage.

Wave polarity phase register: address offset is 0x08, which is to complement the duty cycle percentage.

Run/stop register: offset address is 0x0c, where 0 for stop, 1 for start to output PWM waveform.

What we want here is to output a symmetric square waveform with frequency set to 100Hz from output Y1 pin. To realize this experiment we just replace the State_1 code from the previous example with the following code and then compile, assemble, and download the program.

```
br_outport (0x5010,  100);
br_outport (0x5014,  50);
br_outport (0x5018,  1);
br_outport (0x501c,  1);
while (1) br_delay(1000);
```

The PWM waveform drives led indicator of Y1 to light up and turn off very quickly (“0” to light up, “1” to turn off), as a result, it becomes dim.

Communication interface and master PC API

egPLC6888 has 4 communication ports, 1 USB and 3 RS485 ports. USB communication port is used by master PC to access egPLC6888 IO space registers via API. USB port is also used to download user program and to set RS485 communication mode settings. RS485 ports include 2 master and 1 slave ports. The slave port is embedded with Modbus protocol, and slave port can also be accessed by RS485 API. In either way the IO space registers can be read/written through C/.Net programming language.

Windows API supports PC host C/C++, C#, Vb development programming interface. The API lib supports USB, RS485 (slave port), and TCP/IP network communication. Although egPLC6888 only supports USB and RS485 modbus RTU slave.

The software architecture united all egPLC series and other egDevice products. The following diagram summarizes the software component and exported API for C/C++ and .Net.

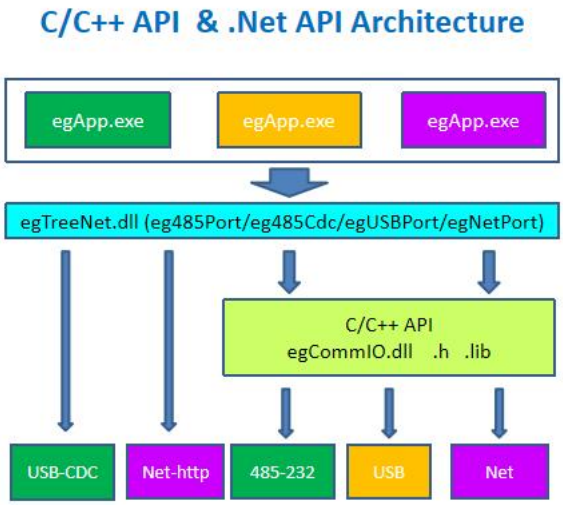


Figure 12. Windows C/C++ API and .Net API architecture

For egPLC6888, egCommIO.h, egCommIO.lib, egCommIO.dll are used for C/C++ API; and egTreeNet.dll is used for .Net which is used in C# / VB.

.Net API encapsulates 3 library classes, they are egUsbPort, eg485Port, and egNetPort. egTreeIO.dll integrates all the above 3 library classes into a single one under .Net architecture, which provides an easy way for .Net Windows GUI programming development. It is shown as follows:

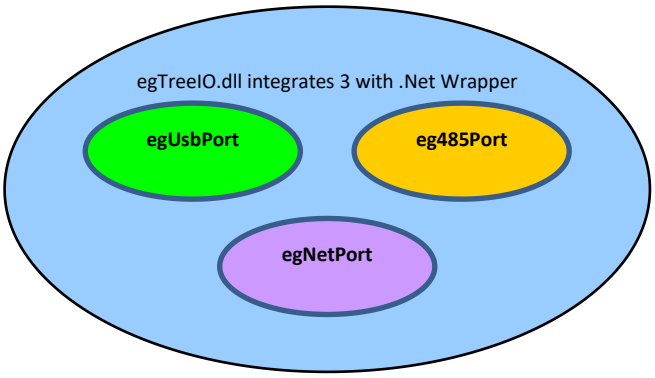


Figure 13. PC API software components

The following table summarizes the product variants and their API classes. All APIs are associated with communication ports.

Product and Communication Class	USB Port connection	485 port connection	Net port connection
egTreePIO32AD	yes		
egPLC	yes	yes	
egDAA		yes	yes
Common API Prototypes	egConnect egInput egOutput egSampleAll	eg485Open eg485Input eg485Output eg485SampleAll	Net_egInput Net_egOutput Net_egSampleAll
C/C++ library files	egCommIO.dll egCommIO.lib egCommIO.h	egCommIO.dll egCommIO.lib egCommIO.h	egCommIO.dll egCommIO.lib egCommIO.h
DotNet C#/VB file and Class	egTreeIO.dll class eg485Port class egUSBPort class egNetPort	egTreeIO.dll class eg485Port class egUSBPort class egNetPort	egTreeIO.dll class eg485Port class egUSBPort class egNetPort egOpen485 egRead485 egWrite485

Table 6. summary of C/C++ and .Net API for egPLC6888 communication

For more information about Windows PC API for egPLC6888, please refer to 《egTreeIO API》