User’s Manual of ISaGRAF® Embedded Controllers

The "User’s Manual of ISaGRAF Embedded Controllers" is intended for integrators, programmers, and maintenance personnel who will be installing and maintaining an I-8417/8817/8437/8837, I-7188EG, I-7188XG & Wincon-8037/8337/8737 series controller system featuring the ISaGRAF Workbench software program.

Please refer to CD-ROM:\napdos\isagraf\wincon\Difference_between_i8437_w8337.pdf & ReadMe.pdf for Wincon-8037/8337/8737

ICP DAS CO., LTD. would like to congratulate you own your purchase of our ISaGRAF controller. The ease to integration of the controller system and the power of the IEC 61131-3 ISaGRAF software program combine to make a powerful, yet inexpensive industrial process control system.

Legal Liability

ICP DAS CO., LTD. assumes no liability for any and all damages that may be incurred by the user as a consequence of this product. ICP DAS CO., LTD. reserves the right to change this manual at any time without notice.

ICP DAS CO., LTD. constantly strives to provide our customers with the most reliable and accurate information possible regarding our products. However, ICP DAS CO., LTD. assumes no responsibility for its use, or for any infringements of patents or other rights of third parties resulting from its use.

Trademark & Copyright Notice

The names of products are used for identification purposes only, and are the registered trademarks of their respective owners or companies.

Table of Contents

Reference Guide

English manual:

I-8000 & I-7188 CD: \napdos\isagraf\8000\english_manu\ “user_manual_i_8xx7.pdf“
Wincon CD: \napdos\isagraf\wincon\english_manu\ “user_manual_i_8xx7.pdf”

中文使用手冊:

I-8000 & I-7188 CD: \napdos\isagraf\8000\chinese_manu\ “chinese_user_manual_i_8xx7.pdf”
Wincon CD: \napdos\isagraf\wincon\chinese_manu\ “chinese_user_manual_i_8xx7.pdf”

I-8000 Hardware Manual:

Please refer to I-8000 CD\NAPDOS\8000\index.htm .

Resource on the Internet:

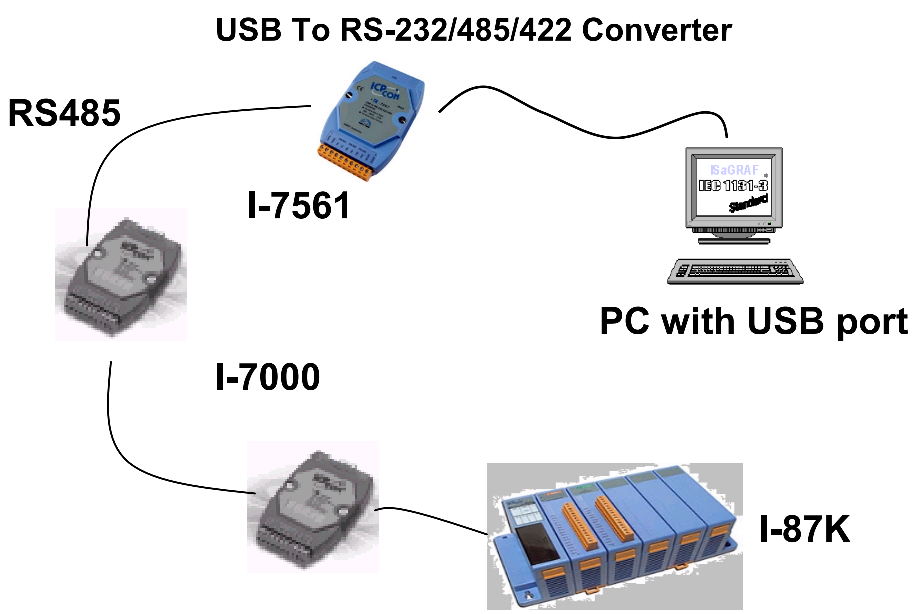
Newly updated ISaGRAF IO libraries, drivers and manuals can be found at
http://www.icpdas.com/products/8000/isagraf.htm

Technical Service:

Please contact local agent or email problem-report to service@icpdas.com
New information can be found at www.icpdas.com

USB To RS-232/485/422 Converter:

http://www.icpdas.com/products/7000/i-7561.htm



Specifications: I-8437 / 8837

Power supply	
Power requirements	10 to 30VDC (unregulated)
Power consumption	20W (when I/O slots are empty)
Protection	Built-in power protection & network protection circuit
General environment	
Operating temperature	-25°C to +75°C
Storage temperature	-30°C to +85°C
Humidity	0 to 95 % (non-condensed)
System	
CPU	Am188™ES,40MHz, or compatiable
Watchdog timer	0.8 second
Real time clock	Year-2000 compliance. Gives hour, minute, sec, date of week, date of month, month & year (1980 to 2079)
SRAM	512Kbytes
FLASH Memory	512Kbytes, Erase unit is 64K bytes, 100,000 erase/write cycles
NVSRAM	31 bytes, battery backup, data valid up to 10 years
EEPROM	2048 bytes, retention > 100 years. 1,000,000 erase/write cycles
SMMI	Five 7-Seg. Led, four push buttons & three Led on the front panel. It can display message, value, input value, simulate input & ouput.
I/O slots	4 empty slots for I-8437, 8 empty slots for I-8837 Accept parallel & serial I/O boards
NET ID	8 dip switch to set NET ID as 1 to 255
Serial ports	
COM1	RS232: TXD,RXD,GND, Speed: 115200 bps max. Program download port.
Ethernet	10M bps, NE2000 compatible, 10 BaseT, Program download port.
COM3	Can be configed as RS232 or S485, Speed: 115200 bps max. RS232: TXD,RXD,RTS,CTS,GND, RS485: Data+, Data-
COM4	RS232: Full modem signals, Speed: 115200 bps max. TXD,RXD,RTS,CTS,DSR,DTR,CD,RI,GND.
Development software	
ISaGRAF	IEC61131-3 standard. Languages: LD, ST, FBD, SFC, IL & FC
Motion control	
	The I-8417/8817/8437/8837 can integrate with one I-8091(2-axes) or two I-8091(4-axes) motion board to do motion control. When doing motion control, Ethernet communication is not available.
PWM output	
Pulse Width Modulation output	8 channels max. for one controller. 500Hz max. for Off=1ms & On=1 ms Output square curve: Off: 1 to 32767 ms, On: 1 to 32767 ms Optional parallel D/O boards: i-8037, 8041, 8042, 8054, 8055,

	8056, 8057, 8060, 8063, 8064, 8065, 8066,8068, 8069
Counters	
Parallel D/I counter	8 ch. max. for one controller. Counter value: 32 bit 500Hz max. Min. pulse width > 1ms Optional parallel D/I boards: i-8040, 8042, 8051, 8052, 8053, 8054, 8055, 8058, 8063, 8077
Serial D/I counter	Counter input: 100Hz max. Counter value: 0 to 65535 (16 bit) Optional serial I-87K D/I boards: i-87051, 87052, 87053, 87054, 87055, 87058, 87063
Remote D/I counter	All remote I-7000 & I-87K D/I modules support counters. 100Hz max. Counter value: 0 to 65535 (16 bit)
High speed counter	i-87082: 100kHz max. 32 bit, i-8080: 450kHz max. 32 bit
Protocols	
Modbus serial protocol	COM1 default for connecting ISaGRAF, PC/HMI & MMI panels.
Modbus TCP/IP protocol	Ethernet port for connecting ISaGRAF & PC/HMI.
Remote I/O	COM3 or COM4 supports I-7000 I/O modules & (I-87K base + I-87K serial I/O boards) as remote I/O. Max. 64 I/O module for one controller
Modbus slave I/O devices	COM1 or COM3 or COM4 (or COM5 if multi-serial port boards are plugged) supports Modbus master protocol to connect to other Modbus slave I/O devices
Fbus	A software mechanism built in COM3 port to exchange data between ICP DAS's ISaGRAF controllers.
Ebus	A software mechanism built in Ethernet port to exchange data between ICP DAS's ISaGRAF Ethernet controllers.
SMS: Short Message Service	COM4 or COM5 can link to a GSM modem to support SMS. User can request data or control the controller by cellular phone. The controller can also send data & alarms to user's cell. phone. Optional GSM modems: GM29:GSM 900/1800 MHz
User defined protocol	User can write his own protocol applied at COM1, COM3, COM4 (& COM5 to COM20 if multi-serial port boards are plugged).
Modem_Link	Supports PC remotely download & monitor & I-8417/8817/8437/8837 through a normal modem.
MMICON / LCD	COM3 or COM4 supports ICP DAS's MMICON. The MMICON is featured with a 240 x 64 dot LCD and a 4 x 4 Keyboard. It can display picture, string, integer, float, and input a character, string, integer and float.
Redundant Bus7000	Two ISaGRAF controllers can link to remote I-7000 & I-87K I/O modules at the same time. Only one controller is active to control these remote I/Os. If one is dead, the other one will take over the control of remote I/Os.
Battery backup SRAM	
	Data, date & time can be stored at S256/S512, and then PC can load these data via COM1 or COM2. PC can also download pre-defined data to the S256/S512. Optional: S256: 256kbytes, S512: 512kbytes

Specifications: I-8417 / 8817

Power supply	
Power requirements	10 to 30VDC (unregulated)
Power consumption	20W (when I/O slots are empty)
Protection	Built-in power protection & network protection circuit
General environment	
Operating temperature	-25°C to +75°C
Storage temperature	-30°C to +85°C
Humidity	0 to 95 % (non-condensed)
System	
CPU	Am188™ES,40MHz, or compatiable
Watchdog timer	0.8 second
Real time clock	Year-2000 compliance. Gives hour, minute, sec, date of week, date of month, month & year (1980 to 2079)
SRAM	512Kbytes
FLASH Memory	512Kbytes, Erase unit is 64K bytes, 100,000 erase/write cycles
NVSRAM	31 bytes, battery backup, data valid up to 10 years
EEPROM	2048 bytes, retention > 100 years. 1,000,000 erase/write cycles
SMMI	Five 7-Seg. Led, four push buttons & three Led on the front panel. It can display message, value, input value, simulate input & ouput.
I/O slots	4 empty slots for I-8417, 8 empty slots for I-8817 Accept parallel & serial I/O boards
NET ID	8 dip switch to set NET ID as 1 to 255
Serial ports	
COM1	RS232: TXD,RXD,GND, Speed: 115200 bps max. Program download port.
COM2	RS485: Data+, Data-, Speed: 115200 bps max. Self-tuner ASIC inside, Program download port.
COM3	Can be configed as RS232 or S485, Speed: 115200 bps max. RS232: TXD,RXD,RTS,CTS,GND, RS485: Data+, Data-
COM4	RS232: Full modem signals, Speed: 115200 bps max. TXD,RXD,RTS,CTS,DSR,DTR,CD,RI,GND.
Development software	
ISaGRAF	IEC61131-3 standard. Languages: LD, ST, FBD, SFC, IL & FC
Motion control	
	The I-8417/8817/8437/8837 can integrate with one I-8091(2-axes) or two I-8091(4-axes) motion board to do motion control. When doing motion control, Ethernet communication is not available.
PWM output	
Pulse Width Modulation output	8 channels max. for one controller. 500Hz max. for Off=1 & On=1 ms Output square curve: Off: 1 to 32767 ms, On: 1 to 32767 ms Optional parallel D/O boards: i-8037, 8041, 8042, 8054, 8055, 8056, 8057, 8060, 8063, 8064, 8065, 8066,8068, 8069

Counters	
Parallel D/I counter	8 ch. max. for one controller. Counter value: 32 bit 500Hz max. Min. pulse width > 1ms Optional parallel D/I boards: i-8040, 8042, 8051, 8052, 8053, 8054, 8055, 8058, 8063, 8077
Serial D/I counter	Counter input: 100Hz max. Counter value: 0 to 65535 (16 bit) Optional serial I-87K D/I boards: i-87051, 87052, 87053, 87054, 87055, 87058, 87063
Remote D/I counter	All remote I-7000 & I-87K D/I modules support counters. 100Hz max. Counter value: 0 to 65535 (16 bit)
High speed counter	i-87082: 100kHz max. 32 bit, i-8080: 450kHz max. 32 bit
Protocols	
Modbus serial protocol	COM1 & COM2 default supports Modbus serial protocol for connecting ISaGRAF, PC/HMI & MMI panels.
Remote I/O	COM3 or COM4 supports I-7000 I/O modules & (I-87K base + I-87K serial I/O boards) as remote I/O. Max. 64 I/O module for one controller
Modbus slave I/O devices	COM1 or COM3 or COM4 (or COM5 if multi-serial port boards are plugged) supports Modbus master protocol to connect to other Modbus slave I/O devices
Fbus	A software mechanism built in COM3 port to exchange data between ICP DAS's ISaGRAF controllers.
SMS: Short Message Service	COM4 or COM5 can link to a GSM modem to support SMS. User can request data or control the controller by cellular phone. The controller can also send data & alarms to user's cell. phone. Optional GSM modems: GM29:GSM 900/1800 MHz
User defined protocol	User can write his own protocol applied at COM1, COM3, COM4 (& COM5 to COM20 if multi-serial port boards are plugged).
Modem_Link	Supports PC remotely download & monitor I-8417/8817/8437/8837 through a normal modem.
MMICON / LCD	COM3 or COM4 supports ICP DAS's MMICON. The MMICON is featured with a 240 x 64 dot LCD and a 4 x 4 Keyboard. User can use it to display picture, string, integer, float, and input a character, string, integer and float.
Redundant Bus7000	Two ISaGRAF controllers can link to remote I-7000 & I-87K I/O modules at the same time. Only one controller is active to control these remote I/Os. If one is dead, the other one will take over the control of remote I/Os.
Battery backup SRAM	
	Data, date & time can be stored at S256/S512, and then PC can load these data via COM1 or COM2. PC can also download pre-defined data to the S256/S512. Optional: S256: 256kbytes, S512: 512kbytes

Specifications: I-7188EG

Power supply	
Power requirements	10 to 30VDC (unregulated)
Power consumption	7188EG:2W , 7188EGD: 3W
Protection	Built-in power protection & network protection circuit
General environment	
Operating temperature	-25°C to +75°C
Storage temperature	-40°C to +85°C
Humidity	0 to 95 % (non-condensed)
System	
CPU	Am188™ES,40MHz, or compatiable
Watchdog timer	1.6 second
Real time clock	Year-2000 compliance. Gives hour, minute, sec, date of week, date of month, month & year (1980 to 2079)
SRAM	512Kbytes
FLASH Memory	512Kbytes, Erase unit is 64K bytes, 100,000 erase/write cycles
NVSRAM	31 bytes, battery backup, data valid up to 10 years
EEPROM	2048 bytes, retention > 100 years. 1,000,000 erase/write cycles
Display for I-7188EGD	Five 7-Seg. Led on the front. It can display message & value.
Expansion I/O bus	One optional Xxxx series I/O board can be plugged inside I-7188EG/D.
NET ID	Set by software
Ethernet port	
	10M bps, NE2000 compatible, 10 BaseT, Program download port.
Serial ports	
COM1	RS232: TXD,RXD,RTS,CTS,GND, Speed: 115200 bps max. Program download port.
COM2	RS485: D+, D- , Speed: 115200 bps max. Self-tuner ASIC inside
Development software	
ISaGRAF	Supports IEC61131-3 standard. Programming languages: LD, ST, FBD, SFC, IL & FC
PWM output	
Pulse Width Modulation output	All Xxxx series D/O boards support PWM output. 8 channels max. for one controller. 500Hz max. for Off=1 & On=1 ms Output square curve: Off: 1 to 32767 ms, On: 1 to 32767 ms

Counters	
Parallel D/I counter	All Xxxx series D/I boards support D/I counter. 8 ch. max. for one controller. Counter value: 32 bit 500Hz max. Min. pulse width > 1ms
Remote D/I counter	All remote I-7000 & I-87K D/I modules support counters. 100Hz max. Counter value: 0 to 65535 (16 bit)
Remote high speed counter	Optional i-87082:100kHz max. , 32 bit
Protocols	
Modbus serial protocol	COM1 default supports Modbus serial protocol for connecting ISaGRAF, PC/HMI & MMI panels.
Modbus TCP/IP protocol	Ethernet port supports Modbus TCP/IP protocol for connecting ISaGRAF & PC/HMI.
Remote I/O	COM2 (or COM3:RS485 if found) supports I-7000 I/O modules & (I-87K base + I-87K serial I/O boards) as remote I/O.Max. 64 I/O modules for one controller
Modbus slave I/O devices	COM1 or COM2 (or COM3 if found) supports Modbus master protocol to connect to other Modbus slave I/O devices
Fbus	A software mechanism built in COM2 port to exchange data between ICP DAS's IsaGRAF controllers.
Ebus	A software mechanism built in Ethernet port to exchange data between ICP DAS's ISaGRAF Ethernet controllers.
SMS: Short Message Service	(COM3:RS232 or COM4:RS232 if found) can link to a GSM modem to support SMS. User can request data or control the controller by cellular phone. The controller can also send data & alarms to user's cell. phone. Optional GSM modems: GM29:GSM 900/1800 MHz
User defined protocol	User can write his own protocol applied at COM1, COM2 & (COM3 to COM8 if found).
MMICON / LCD	(COM3:RS232 if found) supports ICP DAS's MMICON. The MMICON is featured with a 240 x 64 dot LCD and a 4 x 4 Keyboard. User can use it to display picture, string, integer, float, and input a character, string, integer and float.
Redundant Bus7000	Two ISaGRAF controllers can link to remote I-7000 & I-87K I/O modules at the same time. Only one controller is active to control these remote I/Os. If one is dead, the other one will take over the control of remote I/Os.
Battery backup SRAM	
	Data, date & time can be stored at X607/X608, and then PC can load these data via COM1. PC can also download pre-defined data to the X607/X608. Optional: X607:128kbytes , X608:512kbytes

Specifications: I-7188XG

Power supply	
Power requirements	10 to 30VDC (unregulated)
Power consumption	7188XG:2W , 7188XGD: 3W
Protection	Built-in power protection & network protection circuit
General environment	
Operating temperature	-25°C to +75°C
Storage temperature	-40°C to +85°C
Humidity	0 to 95 % (non-condensed)
System	
CPU	Am188™ES,40MHz, or compatiable
Watchdog timer	1.6 second
Real time clock	Year-2000 compliance. Gives hour, minute, sec, date of week, date of month, month & year (1980 to 2079)
SRAM	512Kbytes
FLASH Memory	512Kbytes, Erase unit is 64K bytes, 100,000 erase/write cycles
NVSRAM	31 bytes, battery backup, data valid up to 10 years
EEPROM	2048 bytes, retention > 100 years. 1,000,000 erase/write cycles
Display for I-7188XGD	Five 7-Seg. Led on the front. It can display message & value.
Expansion I/O bus	One optional Xxxx series I/O board can be plugged inside I-7188XG/D.
NET ID	Set by software
Serial ports	
COM1	Can be used as RS232 or RS485 , Speed: 115200 bps max. RS232 TXD,RXD,RTS,CTS,GND RS485: D+, D-, self-tuner inside Program download port.
COM2	RS485: D+, D- , Self-tuner ASIC inside , Speed: 115200 bps max.
Development software	
ISaGRAF	Supports IEC61131-3 standard. Programming languages: LD, ST, FBD, SFC, IL & FC
PWM output	
Pulse Width Modulation output	All Xxxx series D/O boards support PWM output. 8 channels max. for one controller. 500Hz max. for Off=1 & On=1ms Output square curve: Off: 1 to 32767 ms, On: 1 to 32767 ms

Counters	
Parallel D/I counter	All Xxxx series D/I boards support D/I counter. 8 ch. max. for one controller. Counter value: 32 bit, 500Hz max. Min. pulse width > 1ms
Remote D/I counter	All remote I-7000 & I-87K D/I modules support counters. 100Hz max. , Counter value: 0 to 65535 (16 bit)
Remote high speed counter	Optional i-87082:100kHz max. 32 bit
Protocols	
Modbus serial protocol	COM1 supports Modbus serial protocol for connecting PC/HMI & MMI panels.
Remote I/O	COM2 (or COM3:RS485 if found) supports I-7000 I/O modules & (I-87K base + I-87K serial I/O boards) as remote I/O.Max. 64 I/O modules for one controller
Modbus slave I/O devices	COM2 (or COM3 if found) supports Modbus master protocol to connect to other Modbus slave I/O devices
Fbus	A software mechanism built in COM2 port to exchange data between ICP DAS's IsaGRAF controllers.
SMS: Short Message Service	(COM3:RS232 or COM4:RS232 if found) can link to a GSM modem to support SMS. User can request data or control the controller by cellular phone. The controller can also send data & alarms to user's cell. phone. Optional GSM modems: GM29:GSM 900/1800 MHz
User defined protocol	User can write his own protocol applied at COM2 & (COM3 to COM8 if found).
MMICON / LCD	(COM3:RS232 if found) supports ICP DAS's MMICON. The MMICON is featured with a 240 x 64 dot LCD and a 4 x 4 Keyboard. User can use it to display picture, string, integer, float, and input a character, string, integer and float.
Redundant Bus7000	Two ISaGRAF controllers can link to remote I-7000 & I-87K I/O modules at the same time. Only one controller is active to control these remote I/Os. If one is dead, the other one will take over the control of remote I/Os.
Battery backup SRAM	
	Data, date & time can be stored at X607/X608, and then PC can load these data via COM1. PC can also download pre-defined data to the X607/X608. Optional: X607:128kbytes , X608:512kbytes

Selection Guide

Power supply	
ACE-540A	24V/1.7A power supply(panel Mount)
DIN-540A	24V/1.7A power supply(DIN-Rail mount)
KA-52F	24V/1A power supply(no mounting)
DIN-KA52F	24V/1A power supply(DIN-Rail mountong)
KWM020-1824F	24V/0.75A power supply (No-mounting)
Development tools	
ISaGRAF-256	ISaGRAF Workbench Software, up to 256 I/O tags.
ISaGRAF Book-E	User's manual of ISaGRAF controllers (English)
ISaGRAF Book-C	User's manual of ISaGRAF controllers (Chinese, traditional)
ISaGRAF controller	
I-8417	ISaGRAF I-8000 controller, 4 empty slots
I-8817	ISaGRAF I-8000 controller, 8 empty slots
I-8437	ISaGRAF I-8000 ethernet controller, 4 empty slots
I-8837	ISaGRAF I-8000 ethernet controller, 8 empty slots
I-7188XG	ISaGRAF I-7188 controller
I-7188XGD	ISaGRAF I-7188 controller with display
I-7188EG	ISaGRAF I-7188 ethernet controller
I-7188EGD	ISaGRAF I-7188 ethernet controller with display
W-8037	ISaGRAF Wincon-8000 controller, No I/O slot
W-8337	ISaGRAF Wincon-8000 controller, 3 empty slots
W-8737	ISaGRAF Wincon-8000 controller, 7 empty slots
Battery backup SRAM	
S256	256Kbytes battery backup SRAM for I-8417 /8817/8437/8837
S512	512Kbytes battery backup SRAM for I-8417 /8817/8437/8837
X607	128Kbytes battery backup SRAM for I-7188XG/7188EG
X608	512Kbytes battery backup SRAM for I-7188XG/7188EG
MMICON / LCD	MMICON + 240x64 Graphic LCD
GSM modem	
GM29	900/1800 GSM/GPRS External Modem
I-87K expansion base	
I-87K4	Remote I-87K I/O base, 4 empty slots
I-87K5	Remote I-87K I/O base, 5 empty slots
I-87K8	Remote I-87K I/O base, 8 empty slots
I-87K9	Remote I-87K I/O base, 9 empty slots
Motion control board	
I-8091	2-axes stepping/servo motor control card
I-8090	3-axes encoder card

Timer/Counter board	
I-8080	4-ch. counter/frequency, 32 bit
I-87082	2 channel counter/Frequency, 32 bit
Multi-serial board	
I-8112	2 port RS232
I-8114	4 port RS232
I-8142	2 port RS485/422
I-8144	4 port RS485/422
Parallel analog I/O board	
I-8017H	8-ch. 14-bit analog input, each ch. can be different input type (V, mA) & range
I-8024	4-ch. 14-bit analog output, each ch. can be different output type (V,mA) & range
Parallel digital I/O board	
I-8037	16-ch. isolated open-drain output
I-8040	32-ch. isolated digital input
I-8041	32-ch. isolated digital output
I-8042	Isolated digital 16-ch. input & 16-ch. output
I-8051	16-ch. non-isolated digital input
I-8052	8-ch. isolated digital input (differential)
I-8053	16-ch. isolated digital input (single ended)
I-8054	Isolated digital 8-ch. input & 8-ch. output
I-8055	Non-isolated digital 8ch. input & 8ch. output
I-8056	16-ch. non-isolated O.C. output
I-8057	16-ch. isolated O.C. output
I-8058	8-ch. isolated digital input, AC/DC
I-8060	6-ch. relay output
I-8063	Isloated digital 4-ch. input & 4-ch. relay
I-8064	8-ch. power relay output
I-8065	8-ch. SSR-AC output
I-8066	8-ch. SSR-DC output
I-8068	8-ch. relay output
I-8069	8-ch. Photo Mos relay output
I-8077	8-ch. digital input & 8-ch. output simulator
Serial analog I/O board	
I-87013	4-ch. RTD input
I-87017	8-ch. analog input
I-87018	8-ch. thermocouple input
I-87022	2-ch. 12-bit analog output, each ch. can be different output type (V,mA) & range
I-87024	4-ch. 14-bit analog output
I-87026	2-ch. 16-bit analog output, each ch. can be different output type (V,mA) & range

Serial digital I/O board	
I-87051	16-ch. non-isolated digital input
I-87052	8-ch. isolated digital input (differential)
I-87053	16-ch. isolated digital input (single ended)
I-87054	Isolated digital 8-ch. input & 8-ch. output
I-87055	Non-isolated digital 8ch. input & 8ch. output
I-87057	16-ch. isolated O.C. output
I-87058	8-ch. isolated digital input, AC/DC
I-87063	Isloated digital 4-ch. input & 4-ch. relay
I-87064	8-ch. power relay output
I-87065	8-ch. SSR-AC output
I-87066	8-ch. SSR-DC output
I-87068	8-ch. relay output
Conveter & Repeater	
PCISA-7520R	PCI/ISA bus RS-232 to RS-485/422 card
PCISA-7520AR	RS-232 to RS-422/RS-485 card with D-sub 9-pin cable
I-7520	RS-232 to RS-485 converter
I-7520R	I-7520 with 3000V DC isolation at RS-485 side
I-7520A	RS-232 to RS-422/RS-485 converter
I-7520AR	I-7520A with 3000V DC isolation at RS-485 side
I-7561	USB to RS-232/422/485 Converter
I-7510	RS-485 isolated high speed repeater
I-7510R	RS485/RS422 isolated high speed repeater
I-7510AR	Three way Isolated RS-422/485 Repeater
RS485 Hub	3-way isolated RS485 to 3 ports RS485 hub
Man Machine Interface	
Touch506L	5.7" 4-Gray STN Panel display with touch
Touch506S	5.7" Color STN Panel display with touch
Touch510T	10.4" Color TFT Panel Display With Touch
Wireless Modem	
SST-2450	Wireless Modem Module with RS-232/RS-485 Interface
I-7000 analog I/O module	
I-7011	1-ch. thermo-couple input (16-bit), 1-ch. D/I & 2-ch. D/O
I-7011D	I-7011 with display
I-7011P	1-ch. thermo-couple input (16-bit), 1-ch. D/I & 2-ch. D/O
I-7011PD	I-7011P with display
I-7012	1-ch. analog input (16-bit), 1-ch. D/I & 2-ch. D/O
I-7012D	I-7012D with display
I-7012F	Fast mode I-7012 (12-bit), normal 16-bit
I-7012FD	I-7012F with display
I-7013	1-ch. RTD input (16-bit)

I-7013D	I-7013 with display
I-7033	3-ch. RTD input (16-bit)
I-7033D	I-7033 with display
I-7014D	1-ch. analog/transmitter input (16-bit) with display, 1-ch. D/I & 2-ch. D/O
I-7016	1-ch. strained gauge input (16-bit), 1-ch. D/I & 4-ch. D/O
I-7016D	I-7016 with display
I-7016P	1-ch. strained gauge input (16-bit), 1-ch. D/I & 4-ch. D/O
I-7016PD	I-7016 with display
I-7017	8-ch. analog input (16-bit)
I-7017F	Fast mode I-7017 (12-bit), normal (16-bit)
I-7018	8-ch. thermocouple input (16-bit)
I-7018P	8-ch. thermocouple input (16-bit)
I-7021	1-ch. analog output (12-bit)
I-7021P	1-ch. analog output (16-bit)
I-7022	2-ch. analog output (12-bit), each ch. can be different output type (V,mA) & range
I-7024	4-ch. analog output (14-bit)
I-7000 digital I/O module	
I-7041	14-ch. isolated digital input
I-7041D	I-7041 with LED display
I-7042	13-ch. isolated O.C. output
I-7042D	I-7042 with LED display
I-7043	16-ch. non-isolated O.C. output
I-7043D	I-7043 with LED display
I-7044	Isolated digital 4-ch. input & 8-ch. output
I-7044D	I-7044 with LED display
I-7050	7-ch. digital input & 8-ch. output
I-7050D	I-7050 with LED display
I-7050A	7 digital input & 8 output (current source)
I-7050AD	I-7050A with LED display
I-7052	8-ch. isolated digital input (6 differential + 2 single end)
I-7052D	I-7052 with LED display
I-7053	16-ch. digital input
I-7053D	I-7053 with LED display
I-7060	4-ch. isolated input & 4-ch. relay output
I-7060D	I-7060 with LED display
I-7063	8-ch. isolated input & 3ch. power relay
I-7063D	I-7063D with LED display
I-7063A	8-ch. isolated input & 3ch. AC-SSR output
I-7063AD	I-7063A with LED display
I-7063B	8-ch. isolated input & 3ch. DC-SSR output
I-7063BD	I-7063B with LED display
I-7065	4-ch. isolated input & 5ch. power relay
I-7065D	I-7065 with LED display
I-7065A	4-ch. isolated input & 5ch. AC-SSR relay

I-7065AD	I-7065A with LED display
I-7065B	4-ch. isolated input & 5ch. DC-SSR relay
I-7065BD	I-7065B with LED display
I-7066	7-ch. Photo Mos relay output
I-7066D	I-7066 with LED display
I-7067	7-ch. relay output
I-7067D	I-7067 with LED display
I-7000 counter module	
I-7080	2 high speed counter/frequency input
I-7080D	I-7080 with display
Parallel I/O board	For I-7188XG & I-7188EG
X107	6-ch. D/I and 7-ch. D/O
X109	7-ch. PhotoMos Relay
X110	14-ch. D/I
X111	13-ch. D/O
X119	7-ch. D/O and 7-ch. D/I
X202	7-ch. A/D (0~20mA)
X203	2-ch. A/D (0~20mA), 2-ch. D/I, 6-ch. D/O
X303	1-ch. A/D (+/-5V), 1-ch. D/A (+/-5V), 4-ch. D/I, 6-ch. D/O
X304	3-ch. A/D (+/-5V), 1-ch. D/A (+/-5V), 4-ch. D/I, 4-ch. D/O
X305	7-ch. A/D (+/-5V), 1-ch. D/A (+/-5V), 2-ch. D/I, 2-ch. D/O
X307	8-ch. A/D (+/-10V), 2-ch. D/I, 2-ch. D/O (will be available)
X308	4-ch. A/D (+/-10V), 6-ch. D/O (will be available)
X310	ch. A/D (0~10V), 1-ch. A/D (0~20mA), 2-ch. D/A (0~10V), 3-ch. D/I, 3-ch. D/O
RS232/422/485 board	For I-7188XG & I-7188EG
X503	1-Port RS-232 (5-Pin)
X504	2-Port RS-232 (5-Pin) and (9-Pin)
X505	3-Port RS-232 (5-Pin)
X506	6-Port RS-232 (3-Pin)
X507	1-Port RS-422/485, 4-ch. D/I, 4-ch. D/O
X508	1-Port RS-232 (5-Pin), 4-ch. D/I, 4-ch. D/O
X509	2-Port RS-232 (3-Pin), 4-ch. D/I, 4-ch. D/O
X510	1-Port RS-232 (3-Pin), 5-ch. D/I, 5-ch. D/O, EEPROM 128K x2
X511	3-Port RS-485
X512	4-Port RS-232 (3-Pin) ,1-ch. RS-485 (will be available)

Chapter 1: Software & Hardware Installation

NOTE:
The I-8xx7 abbreviation is for the I-8417, I-8437, I-8817 and I-8837 controllers, while W-8xx7 is the abbreviation for the Wincon-8037/8337/8737 controller.

1.1: Installing The ISaGRAF Workbench Software Program

Chapter 1 of the "User's Manual of ISaGRAF Embedded Controllers" manual details how to properly setup and run the I-8xx7, I-7188EG/XG & W-8xx7 controller system and the ISaGRAF Workbench software program.

Numerous illustrations and pictures are provided in this chapter to assist the integrator and programmer with the basics of how to properly setup the hardware and software for their system.

If you are not familiar with the setup of either the I-8xx7, I-7188EG/XG & W-8xx7 controller system or the ISaGRAF Workbench software program, please take the time to thoroughly read Chapter 1. The procedures detailed in this chapter are easy to understand, and will assist the user to quickly and easily setup and start running the controller and the ISaGRAF software program.

For the I-8xx7, I-7188EG/XG & W-8xx7 controller system and the ISaGRAF Workbench software to operate properly, it is imperative that each is setup correctly. This chapter covers the details of how to setup the controller system and the ISaGRAF Workbench software in a minimum of time.

Before you can start programming the I-8xx7, I-7188EG/XG & W-8xx7 embedded controller system with the ISaGRAF software program, you must first install the ISaGRAF Workbench software program on a target PC.

Hardware Requirements

- A Personal Computer With At Least A Pentium, 133 MHz Or Faster Processor
- 32 Mbytes Memory (Preferably 64 Mbytes RAM)
- A Hard Drive With At Least 128 Mbytes Of Storage Space (Preferably Larger)
- At Least One RS-232 Serial Port

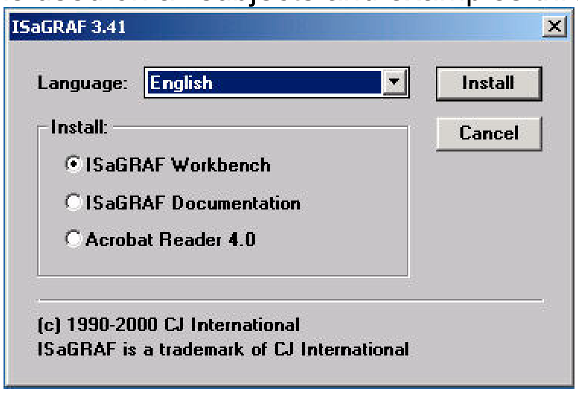
Software Requirements
One of the following computer operating systems must be installed on the target computer system before you can install the ISaGRAF Workbench software program.

- Windows 95
- Windows 98
- Windows NT Version 3.51 or Windows NT Version 4.0
- Windows 2000 Or Windows XP

Steps To Installing The ISaGRAF Workbench Program

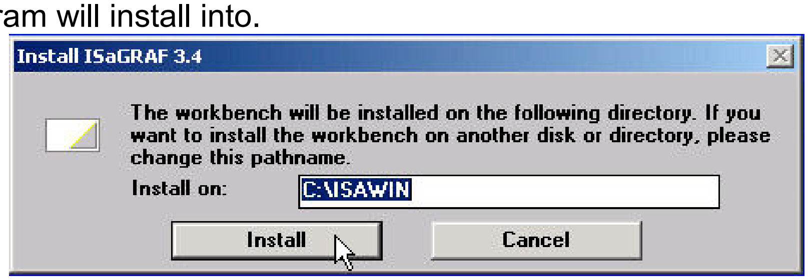
Insert the ISaGRAF Workbench CD into your CD-ROM drive. Normally the auto-start program will activate the "install.bat" file automatically. If your computer does not have the auto-start feature active, use the Windows Explorer and go to the CD-ROM drive where the Workbench CD is installed, then double-click on the "install.bat" file listed on the ISaGRAF CD. If the "install.bat" file is not found on your ISaGRAF CD, then double-click on the "ISaGRAF.exe" file to start the installation process.

Once you have started the "install.bat" file, a dialog box will appear as shown on the next page. Select the language version of the ISaGRAF software program you would like to use. The English version is used on all subjects and examples throughout this manual.

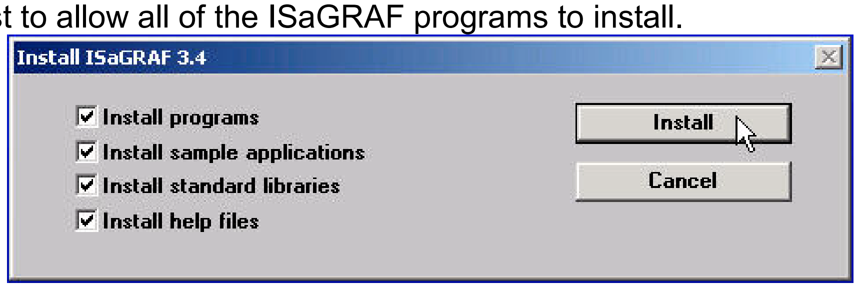


Once you have selected to install the ISaGRAF Workbench program and selected the desired language, just press the "Install" button, and follow the step-by-step directions of each dialog box as they appear to complete the installation process.

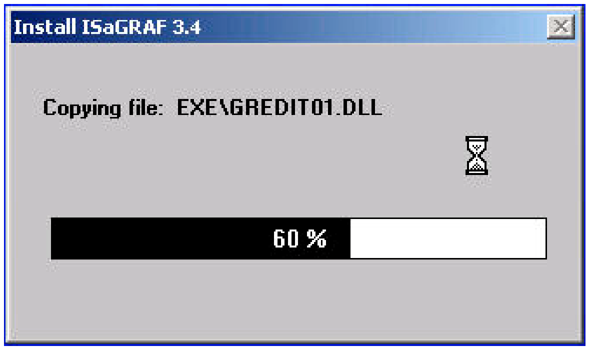
The first dialog box to appear allows the user to define what drive and subdirectory the ISaGRAF program will install into.



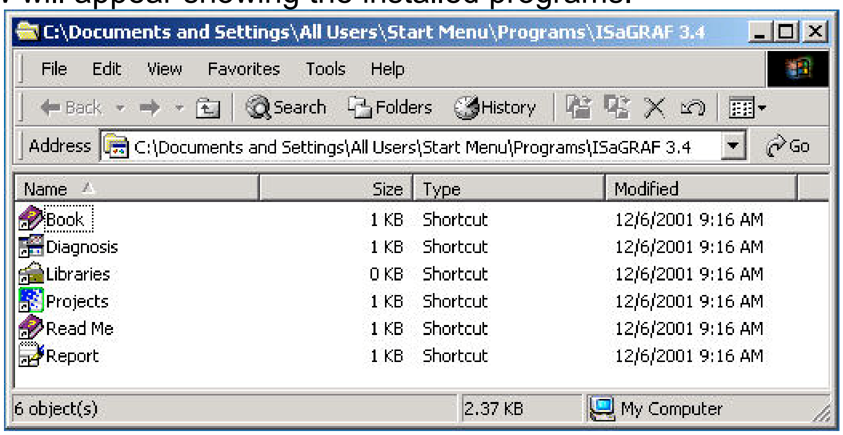
The next dialog box asks the user how much of the ISaGRAF program to you wish to install. By default, it is best to allow all of the ISaGRAF programs to install.



Once you have selected which programs and applications are to be installed, the installation process begins, and an installation progress dialog box will appear showing the installation progress.

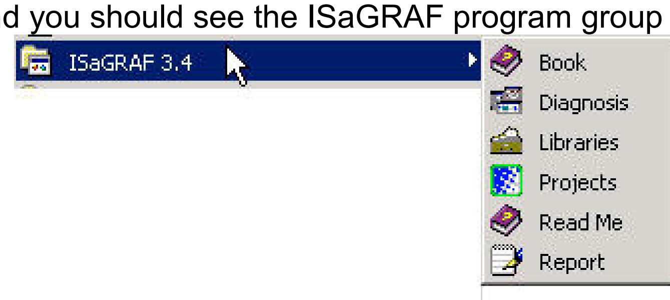


Once the ISaGRAF Workbench software installation process has been completed, a Windows Explorer window will appear showing the installed programs.



The installation process is now complete, and you can begin to use the ISaGRAF software program.

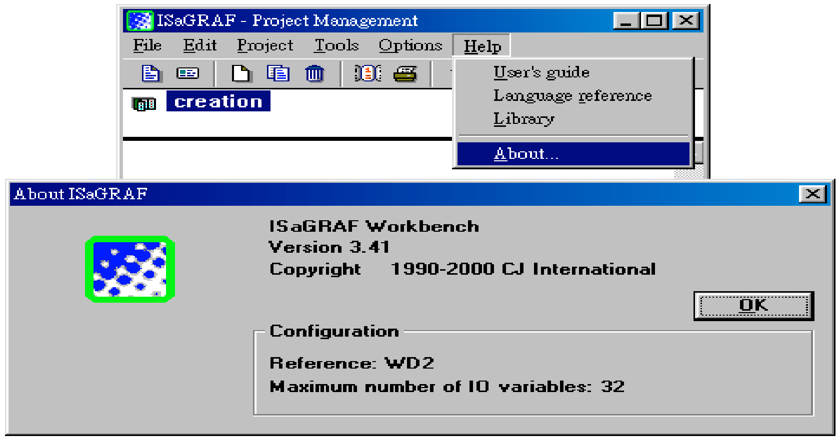
To begin the ISaGRAF 3.x software program, click on the Windows "Start" button, then on "Programs", and you should see the ISaGRAF program group as illustrated below.



You will see that six program icons are now associated with the ISaGRAF 3.x software group. You can select any of the icons to learn more about the ISaGRAF Workbench software program.

NOTE: You must install the hardware protection device (dongle) provided with the ISaGRAF software on your computers parallel port to for the ISaGRAF program to achieve fully authorized functionality.

While using ISaGRAF and the dongle is plugged well, if the “Help” – “About” says “Maximum number of IO variables: 32”, it means ISaGRAF workbench cannot find the dongle well. Please reset your PC and then check the “Help” – “About” again. If it still displays “Maximum number of



IO variables: 32”, the dongle driver may not be installed well. Please execute the ISaGRAF CD_ROM \Sentinel5382\setup.exe for ISaGRAF-80 or \Sentinel\setup.exe for other ISaGRAF version and then reset the PC again.

Important Notice For Window NT Users

If your computer is using the Windows NT operating system, you will need to add one line to the "isa.ini" file in the ISaGRAF Workbench "EXE" subdirectory. If the ISaGRAF program is installed on your computers "C" hard drive, you will find the required file in the following path:

C:\isawin\exe\isa.ini

You can use any ASCII based text editor (such as Notepad or UltraEdit32) to open the "isa.ini" file. Locate the [WS001] header in the "isa.ini" initialization file (it should be at the top of the file). Anywhere within the [WS001] header portion of the "isa.ini" initialization file, add the entry shown below within the [WS001] header:

[WS001]
NT=1
Isa=C:\ISAWIN
IsaExe=C:\ISAWIN\EXE
Group=Samples
IsaApl=c:\isawin\smp
IsaTmp=C:\ISAWIN\TMP

The [WS001] header should now look like the above example. The **NT=1** entry addition is absolutely required for the RS-232 communications to operate properly in the Windows NT operating environment.

1.2: Installing The ICP DAS Utilities For ISaGRAF

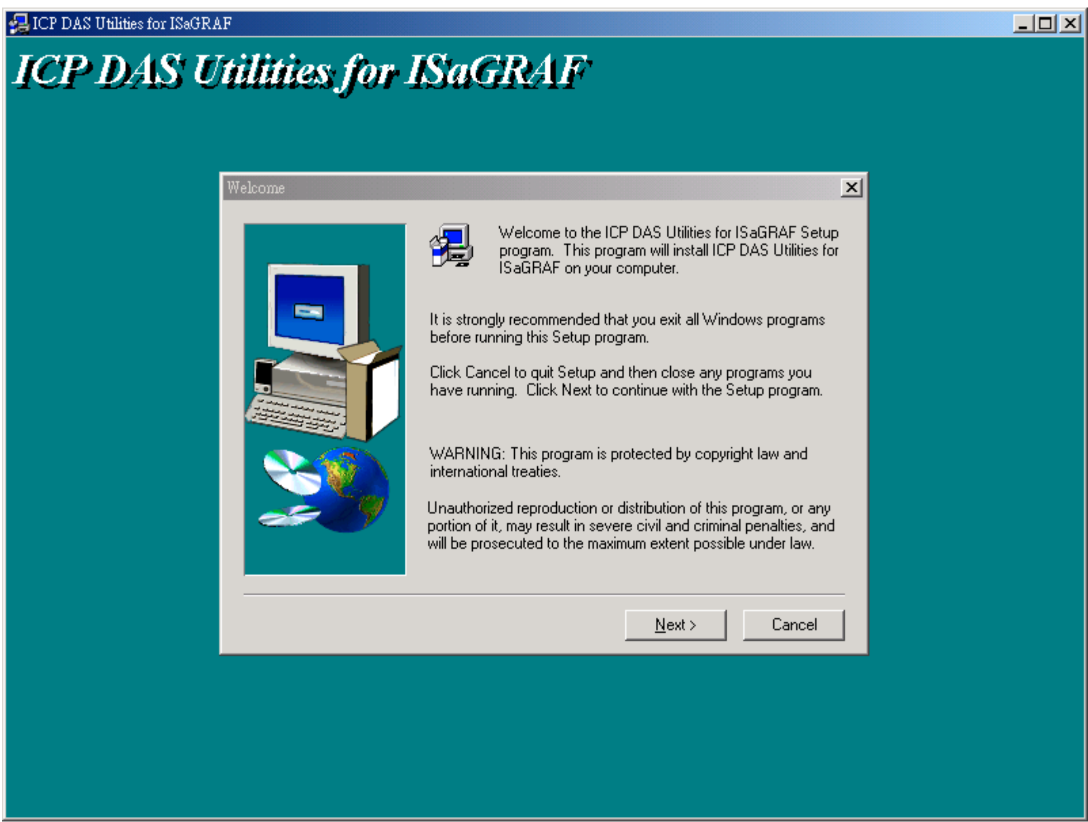
The "ICP DAS Utilities For ISaGRAF" consists of 3 major items.

I/O library (Include I/O libraries of I-8xx7, I-7188EG, I-7188XG & W-8xx7)
Modem_Link utility (Chapter 13)
Auto-scan I/O utility (Section 3.6)

The ISaGRAF Workbench software program must be installed before attempting to install the "ICP DAS Utilities for ISaGRAF". If you have not already installed the ISaGRAF Workbench program, please refer to section 1.1 before continuing.

When the ISaGRAF Workbench program is first installed, it contains only the basic I/O libraries from CJ International - the authors of the ISaGRAF software program. Users will have to install the appropriate I/O library files and some utilities before you can properly program the ISaGRAF controller.

There is a CD-ROM supplied with each of the ISaGRAF controllers with the "ICP DAS Utilities for ISaGRAF". Please insert the CD-ROM into your CD-ROM drive. Then run "setup.exe" in the folder of CD-ROM: \napdos\isagraf\ . Follow the steps to install it.



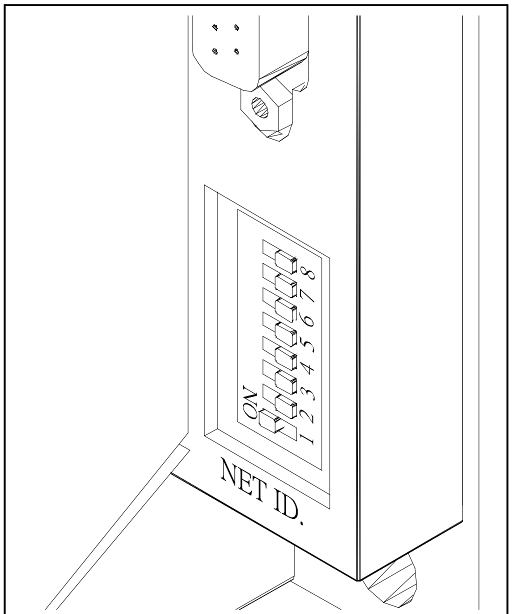
Note:
If "setup.exe" is not in your CD-ROM, please download "**ICP DAS Utilities For ISaGRAF.zip**" from http://www.icpdas.com/products/8000/isagraf.htm

1.3: Connecting Your PC To The Controller

Note:
Below sections are for the I-8417/8817/8437/8837 controller only, please refer to the respective "Getting Started" Manual which delivered with the controller for connecting PC to the I-7188EG/XG or W-8xx7 controller.

1.3.1: Setting The NET-ID Addresses For The I-8xx7 Controller System

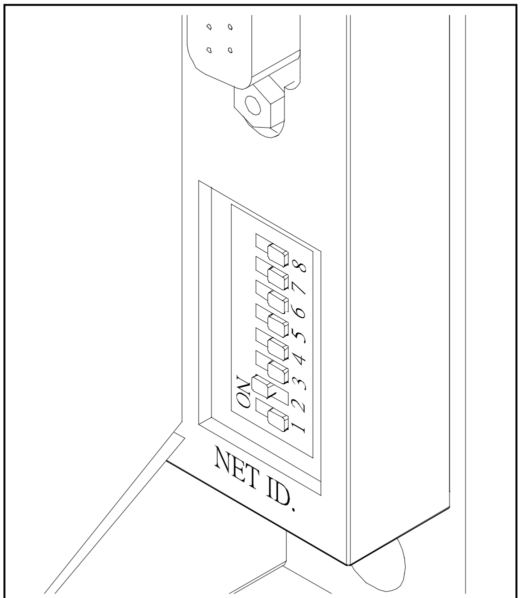
For the I-8xx7 controller to properly operate, it must first be addressed correctly.



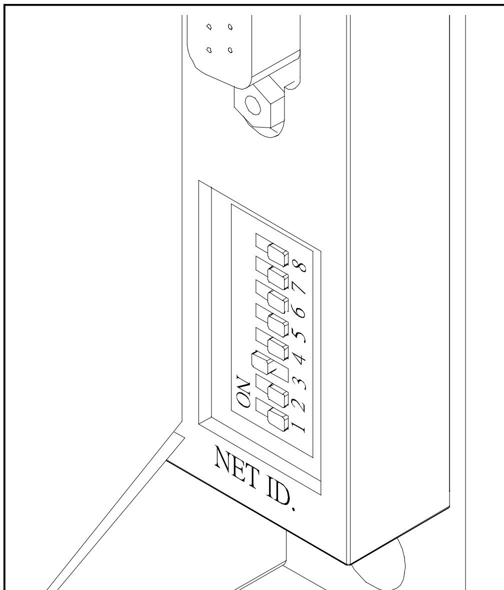
Default setting → NET-ID=1

	1	2	3	4	5	6	7	8
NET-ID=00								
NET-ID=01	ON							
NET-ID=02		ON						
NET-ID=03	ON	ON						
NET-ID=04			ON					
NET-ID=FF	ON	ON	ON	ON	ON	ON	ON	ON

Default setting → NET-ID=01
For ISaGRAF workbench , it can only recognize NET-ID from 01 to FF (1~255).
The NET-ID of every Main Control Unit in the same network must be unique (different from each other).



NET-ID=0x02



NET-ID=0x04

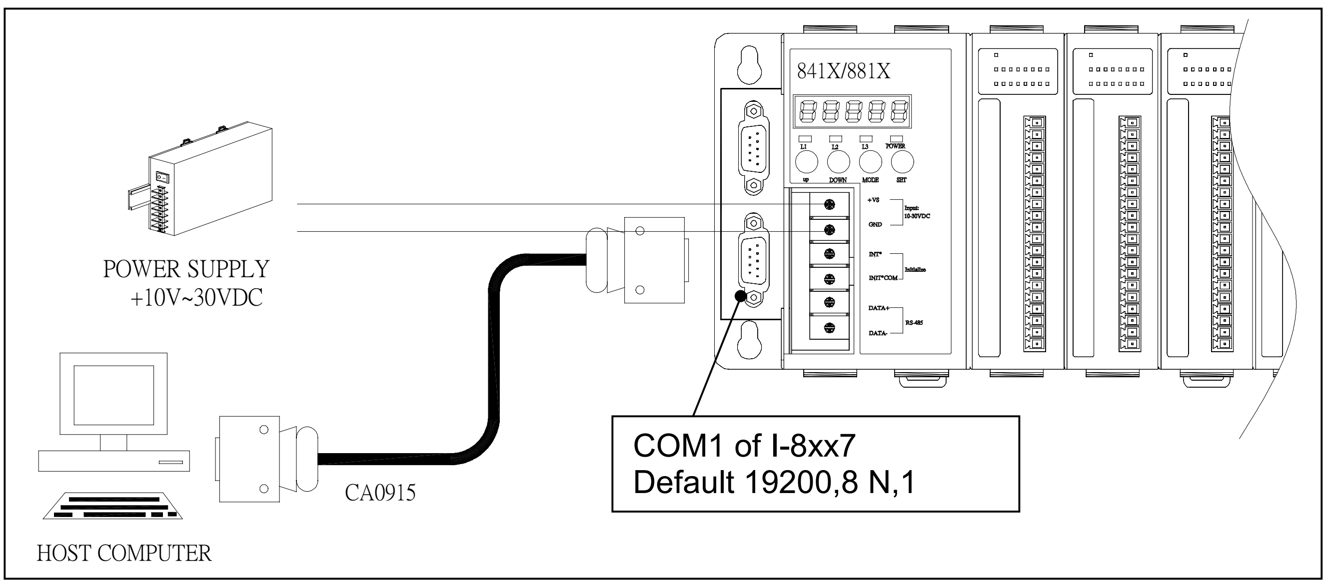
1.3.2: Downloading & Communicating Via Modbus With The I-8xx7

The I-8xx7 controller provides two COM ports standard for downloading the ISaGRAF program and debugging your application. The COM1 port is an RS-232 port and the COM2 port is an RS-485 port for the I-8417/8817 controller system, and the I-8437/8837 features an Ethernet port connection instead of a second COM port.

Both of the COM1 and COM2 ports of the I-8417/8817 controllers support the Modbus communications protocol. For I-8437/8837 controllers, COM1 support Modbus protocol while COM2 is an ethernet port support Modbus TCP/IP protocol. There are an abundant number of Human Machine Interface (HMI) and Man Machine Interface (MMI) software programs and additional hardware devices that support the Modbus or/and Modbus TCP/IP communications protocols. All of these programs and devices can access data from the I-8xx7 controller system through the two COM ports using the Modbus / Modbus TCP/IP protocol.

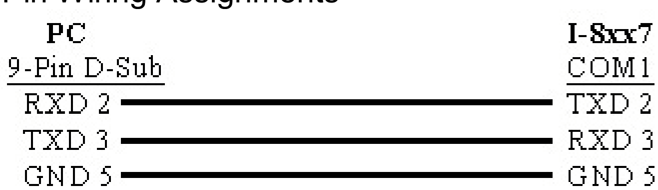
1.3.3: Connecting Your PC To The I-8xx7 COM1 Port

When you receive your I-8xx7 controller system, there is one (1) RS-232 communications cable provided with the system. The cable is used to connect your PC to the I-8xx7 controller or to an I-7520 RS-232/RS-485 converter that can be purchased from ICP DAS.



The communication parameters for the I-8xx7 COM1 port defaultly be set to 19200-baud rate, 8 data bits, no stop bits, and one parity bit ("19200, 8, N, 1").

Normal RS-232 Pin Wiring Assignments

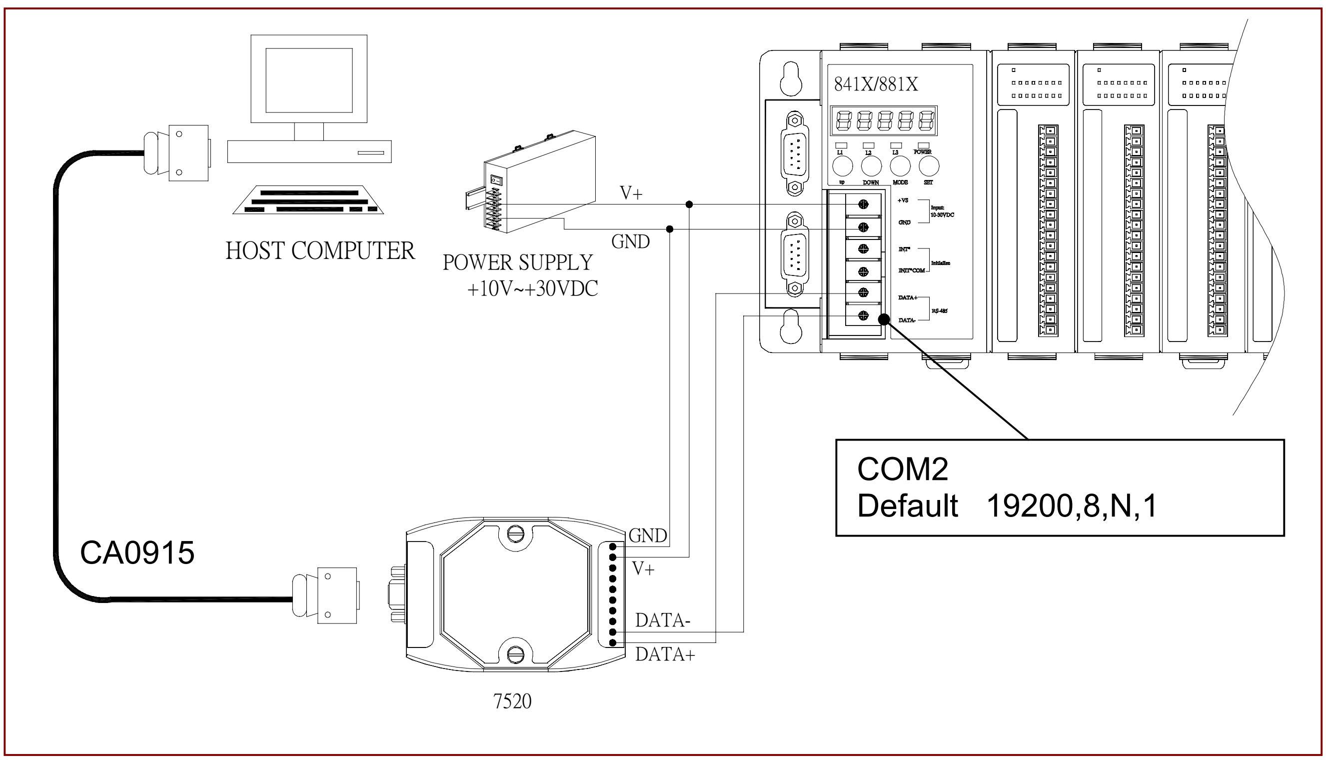


For the ISaGRAF Workbench RS-232 communications to operate properly, only the RXD, TXD, and the GND signals are used. If your PC is running a hardware device or software program that uses the CTS and DSR signals, you will need to wire the RTS-CTS and DTR-DSR signals together as shown below.

PC 9-Pin D-Sub		I-8xx7 COM1
RXD 2	———	TXD 2
TXD 3	———	RXD 3
GND 5	———	GND 5
DTR 4		
DSR 6		
RTS 7		
CTS 8		

1.3.4: Connecting Your PC To The I-8xx7 COM2 Port

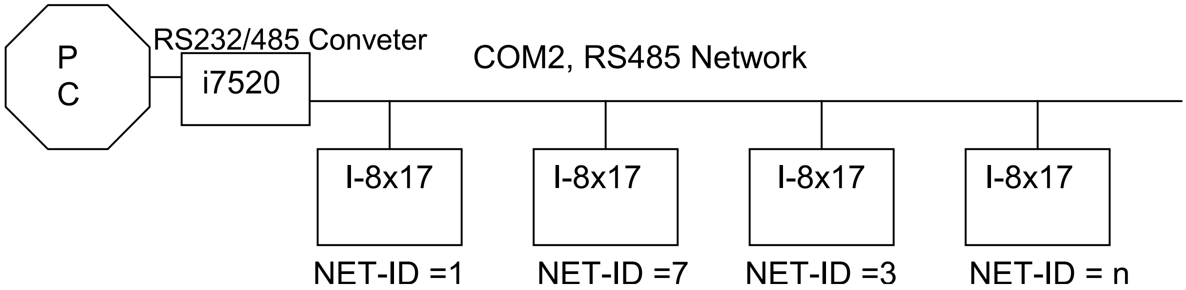
If your PC is connecting to an I-8417/8817's COM2 port (RS-485), the maximum distance between the I-7520 (the RS-232/RS-485 converter) and an I-8xx7 controller is up to 1,200 meters (4,000 feet). The distance between the two is dependent on the baud rate; the rule to follow is the lower you set the baud rate, the longer the distance can be.

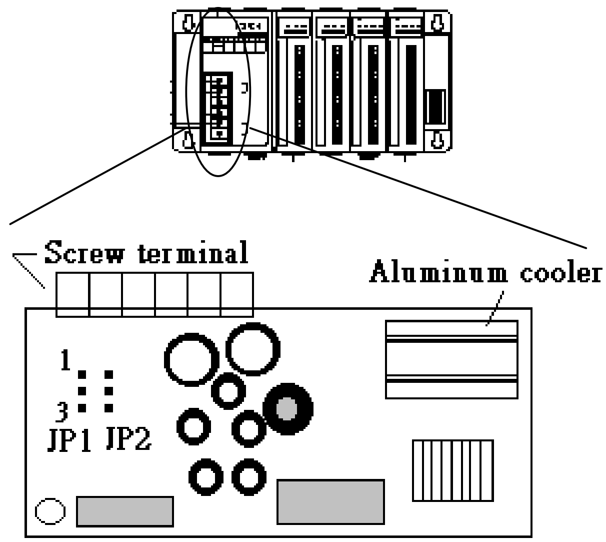


1.3.5: Connecting One PC To Several I-8417/8817 Controllers

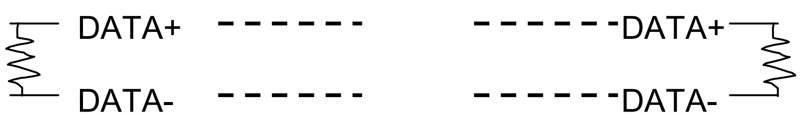
An additional feature of using the COM2 port of the I-8417/8817 is that you can configure an RS-485 network from one PC to link to numerous I-8417/8817 controllers. The PC can download ISaGRAF applications to each I-8417/8817 controller system on the RS-485 network. The maximum number of I-8417/8817 controllers that can be networked via the RS-485 network is 255 (Not recommended to use so many).
To create an RS-485 network you must first insure that each I-8417/8817 controller has a unique NET-ID address, and each of the controllers link the "DATA+" to the "DATA+" signal, and the "DATA-" to the "DATA-" signals.

Lastly, you must plug ONE of the I-8417/8817's JP-1 and JP-2 on the power board to position 1 to 2, (resistance applied to the network). The other I-8417/8817's JP-1 and JP-2 plugs should be left at the default setting of connecting 2 to 3 (no resistance).





It is recommended to add two terminal resistors (try 220Ω, then 110Ω, and then 330Ω) on the nearest I-8417/8817 and farest I-8417/8817 for long distance RS485 network.

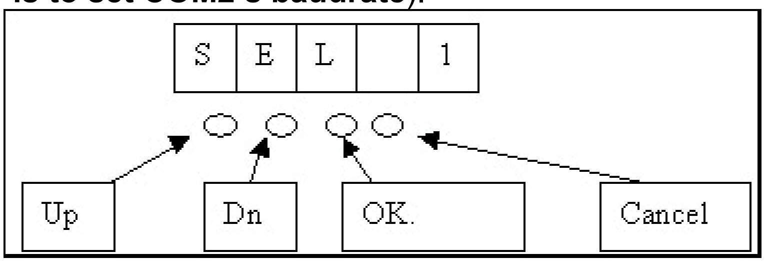


1.3.6: Changing The COM1 & COM2 Baud Rate Setting

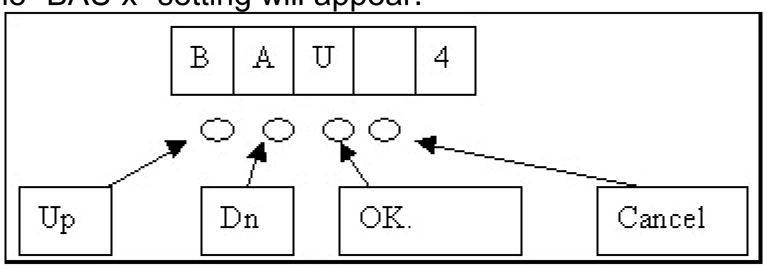
The baud rate for the I-8417/8817/8437/8837's COM1 port (RS-232) can be set between 300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200 bps(bit per second). Other parameter can not be changed, they are always - 8 data bits, No parity, and 1 stop bit . The default baud rate for I-8417/8817/8437/8837's COM1 & I-8417/8817's COM2 is 19200.

To change the baud rate setting on the COM1 & I-8417/8817's COM2 port, first power off the controller. Then press in and hold in the **first two buttons** on the front panel of the controller and then power back up the controller system as shown below.

The first read out to appear is the “SEL 0” or "SEL 1" (“**SEL 0**” **is to set COM1’s baudrate**, while “**SEL 1**” **is to set COM2’s baudrate**).

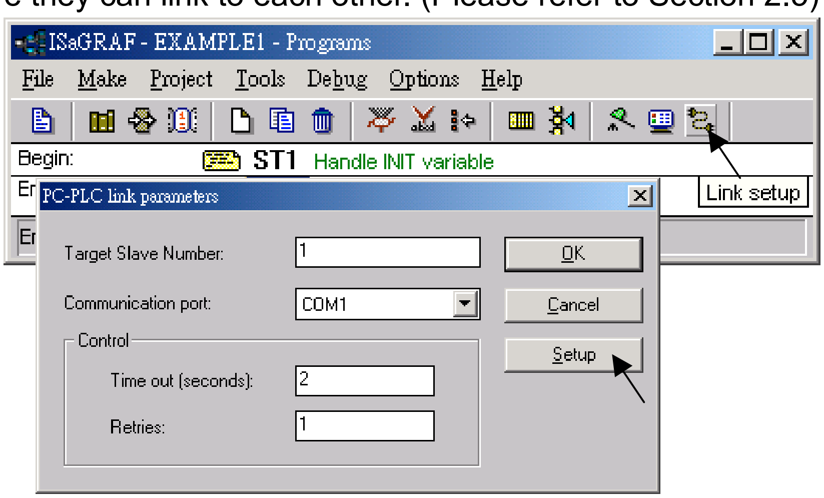


Press the “Up” or “Dn” to change selection, then press the "OK" button (third button on the panel), and the "BAU x" setting will appear.



You can now change the baud rate setting by pressing the "UP" or "Down" button to the desired baud rate setting. The settings for the baud rate are as follows: (0) 1200, (1) 2400, (2) 4800, (3) 9600, (4) 19200, (5) 38400, (6) 57600, (7) 115200, (8) 300 & (9) 600. Press “OK” to save the selected setting. And then press some “Cancel” to exit the hardware setting.

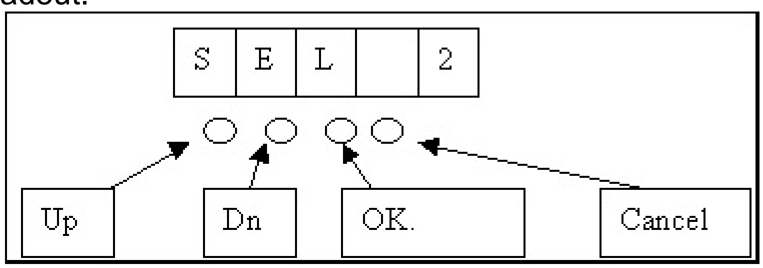
Important Notice: The ISaGRAF workbench’s default setting for PC’s COM1 & COM2 is 19200, 8, N, 1. If you have changed the I-8417/8817/8437/8837 COM1/COM2’s baud rate to other value. You should change your ISaGRAF Workbench’s COMM to the same setting before they can link to each other. (Please refer to Section 2.5)



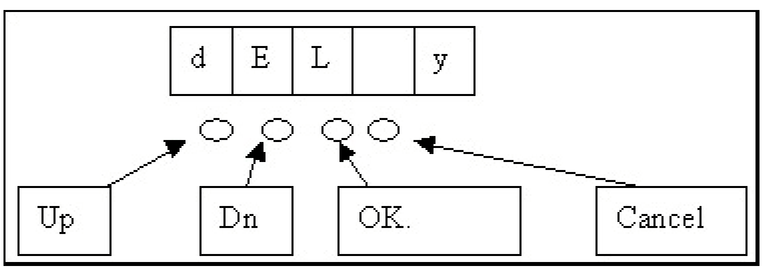
1.3.7: Deleting An ISaGRAF Project From The I-8xx7 Controller

There may be occasions when you will need to delete the ISaGRAF project from the controller system. To begin this, you follow the same control start up routine as changing the baud rate. You first press in and hold in the **first two buttons** on the front panel of the controller and then power back up the I-8417/8817/8437/8837 controller to gain the ability to change the parameters.

When the first display appear, press the "Up" or "Down" button until "SEL 2" (Select 2) appears in the LED readout.



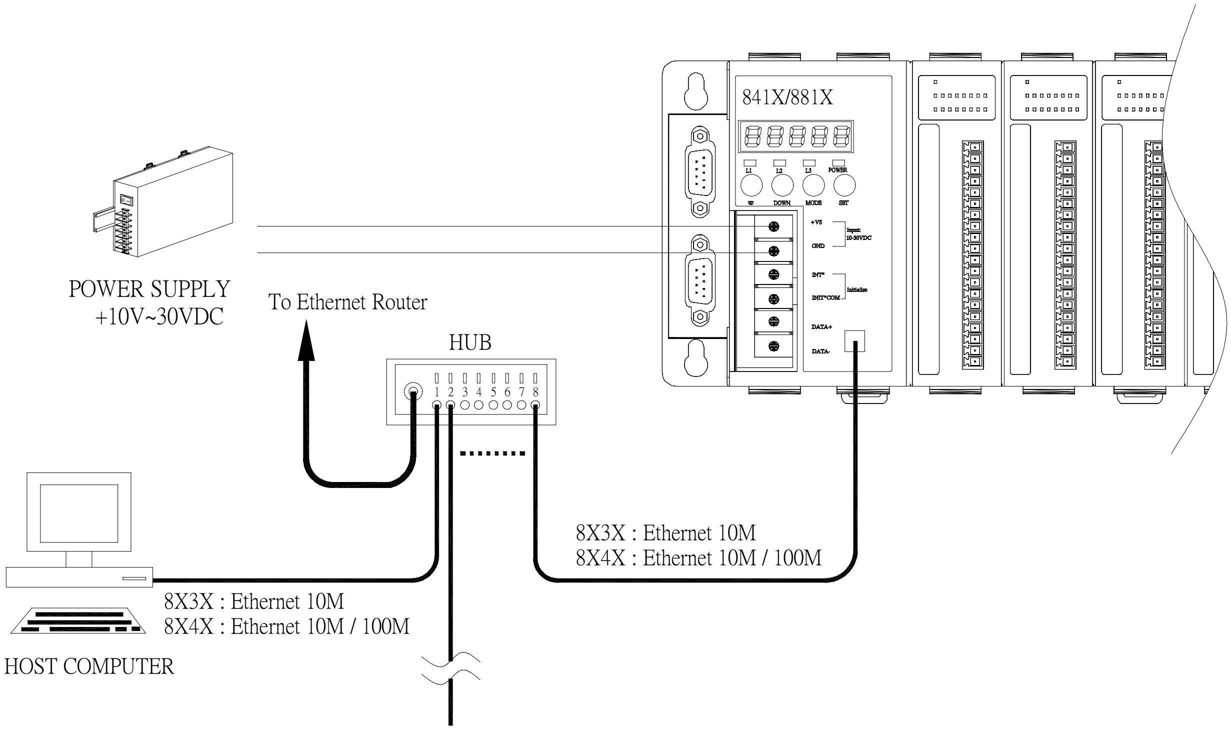
Press the "Up" or "Down" buttons until "dEL" appears in the LED read out.



Press the "Up" or "Down" buttons until "y" appears in the LED readout then press the "OK" button. This will delete the currently installed ISaGRAF project from the controller system. After that press some “Cancel” to exit the hardware setting.

1.3.8: Connecting Your PC To The I-8437/8837 Ethernet Port

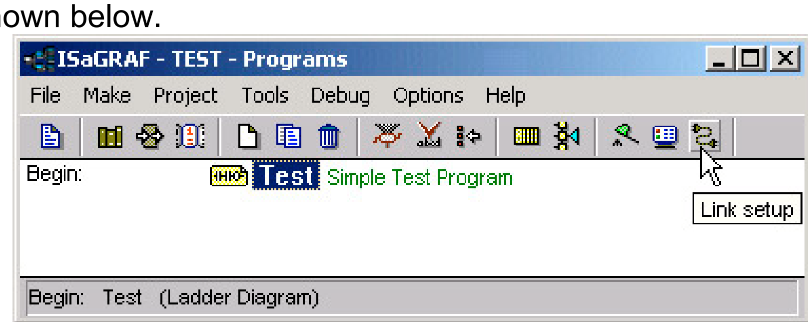
The I-8437 and I-8847 controller systems feature a built in Ethernet port. The COM2 port is replaced from an RS-485 to Ethernet.



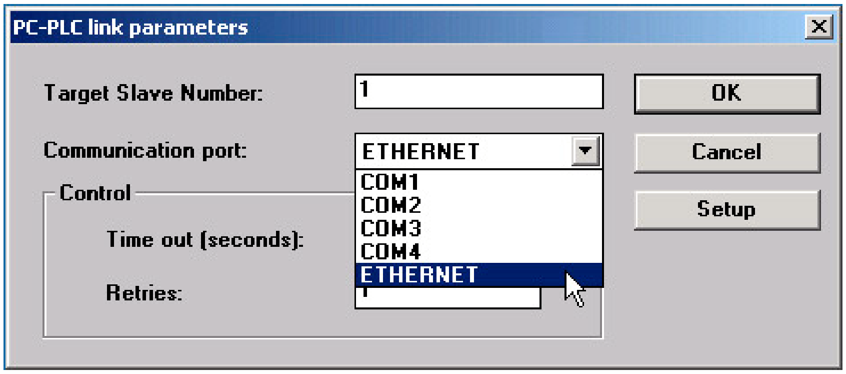
Before you can download an ISaGRAF application to the I-8437/8837 controller system using the Ethernet port, you must first setup the Ethernet port to properly communicate with the host PC.

On the I-8437/8837, Set IP, Mask and Gateway address:
Refer to **Appendix B** or CD_ROM:\NAPDOS\ISaGRAF\8000\driver\setip.txt

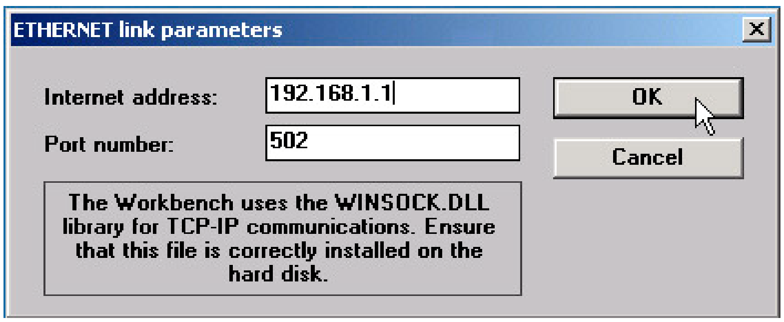
On your PC:
First open an ISaGRAF project and select a program you wish to communicate between your PC and the I-8437/8837 controller system. Next, select the "Link Setup" button on the project screen as shown below.



A "PC-PLC Link Parameters" dialog box will appear as shown below. From here select the "Ethernet" communications option and click on the "Setup" button.



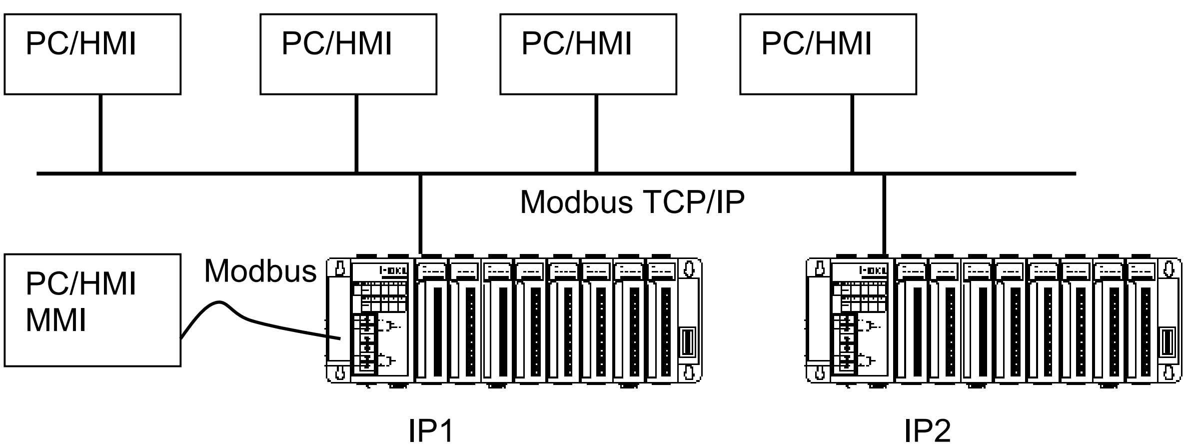
Once you have clicked on the "Setup" button, an "Ethernet Link Parameters" dialog box will appear. Set the "Port Number" to "**502**" and enter in the **Internet address (IP) of the I-8437/8837** controller.



Once you have entered the appropriate information, click on the "OK" button, and now you have configured your PC to communicate with the I-8437/8837 through the Ethernet port.

1.3.9: Multi-Clients Connection to The I-8437/8837 Ethernet Port

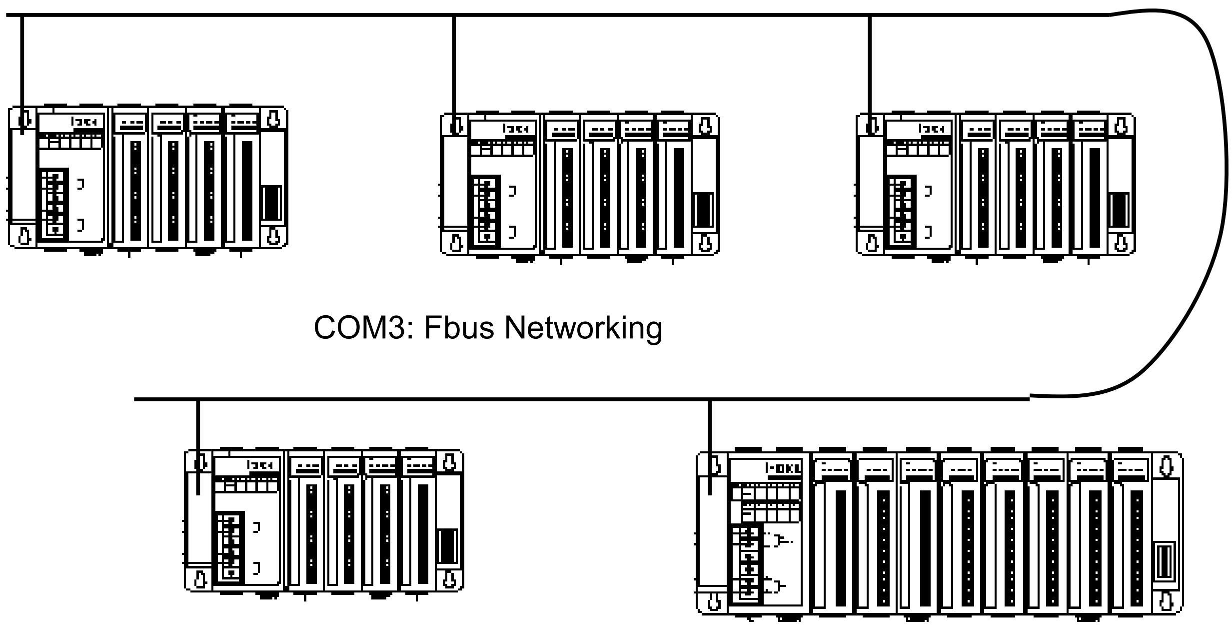
Each I-8437 / 8837 has an IP address and with a fixed Ethernet port No. **502.** Up to 4 PCs can link to one I-8437 / 8837 throughout Ethernet (Modbus TCP/IP protocol). Another PC or MMI can link to COM1: RS232 port (Modbus protocol) of the I-8437/8837. Therefore the maximum number of clients can be linked is 5.



1.4: Controller to Controller Data Exchange: Fbus

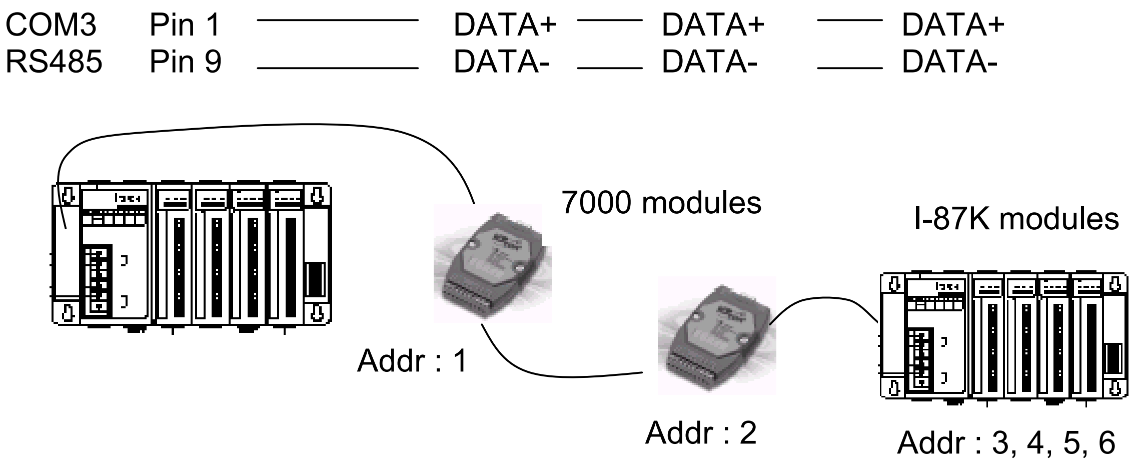
Connect all COM3's Pin 1 together and Pin 9 together and then one of these I-8xx7 controllers should set its **JP1 and JP2 of the power board to position "1 to 2"** (refer to section 1.3.5). The maximum distance for the Fbus data exchange network is 1200 meters (4,000 feet) depending on the communication baud rate. The distance between the PC and the I-8xx7 controller system is dependent on the baud rate; the rule to follow is the lower you set the baud rate, the longer the distance can be.

Pin 1 ——— Pin 1 ——— Pin 1 ——— Pin 1
Pin 9 ——— Pin 9 ——— Pin 9 ——— Pin 9

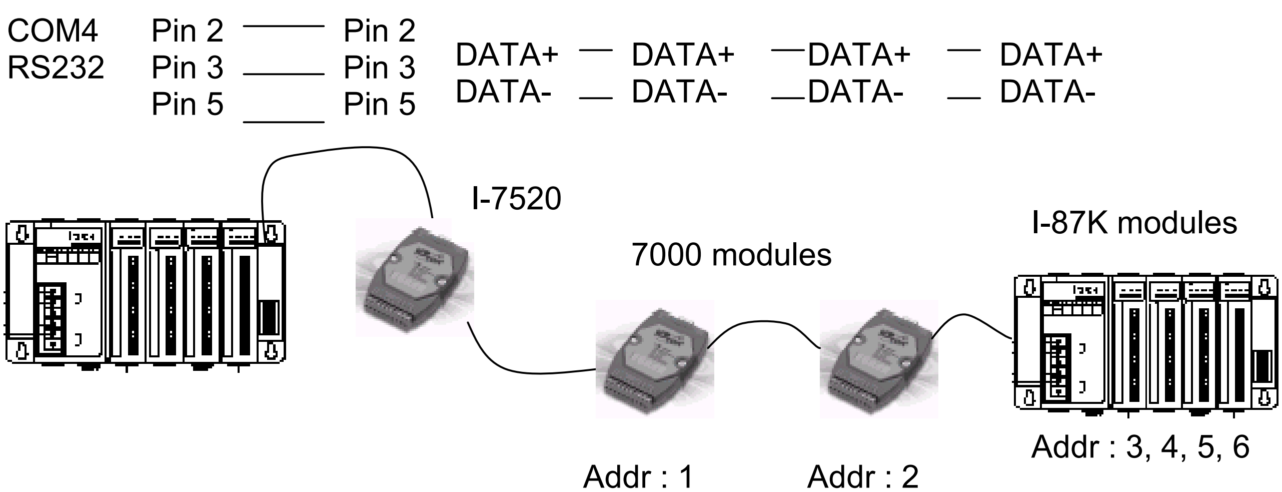


1.5: Linking I-7000 and I-87K Modules For Remote I/O

The I-8xx7 controller system can use one of its COM3 or COM4 ports, wile COM2 or COM3 for I-7188EG/XG, to link to ICP DAS's "I-7000" and “I-87K” series of remote I/O modules. This configuration can be very useful in applications that require distributed remote I/O throughout the system.



If you choose to utilize the COM4 port, connect the COM4 port to the I-7520's RS-232 port, and also connect the "DATA+" to the "DATA+" signal, and the "DATA-" to the "DATA-" signal as shown below.



You can link up to 64 I-7000 or I-87K series remote modules to one I-8xx7 controller system. You must remember to set each I-7000 and I-87K remote module must have a unique address, and be set to the same baud rate as the I-8xx7 controller system.

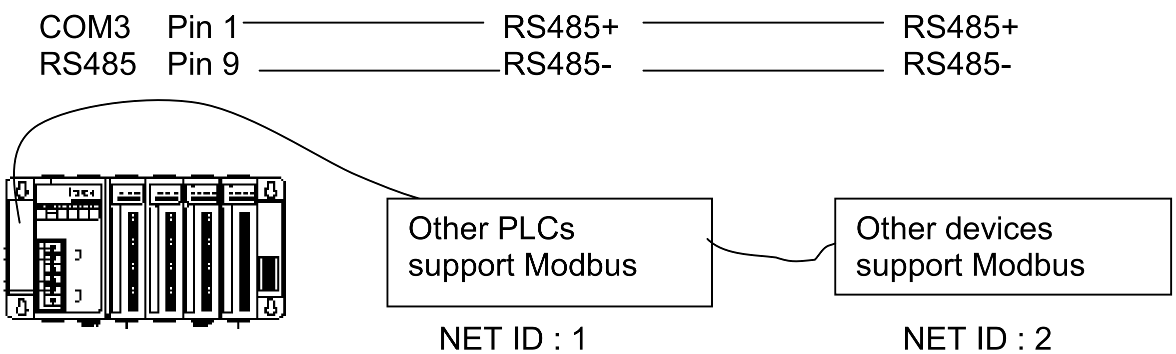
For more information regarding setting up and programming an I-7000 / I-87K remote module, please refer to Chapter 6 - "Linking To I-7000 and I-87K Modules".

1.6: Creating A Modbus Link With The I-8xx7 Controller

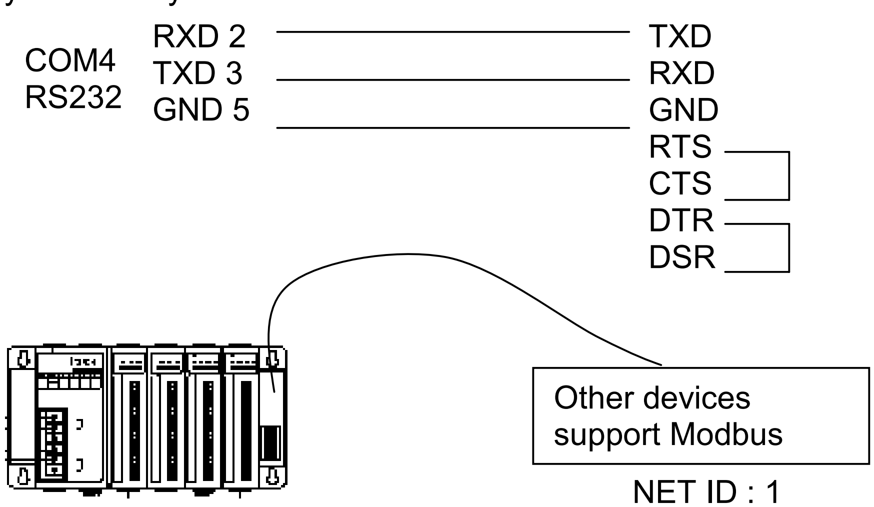
The I-8xx7 controller system can be a Modbus "Slave" and/or a Modbus "Master" controller depending on the application. Through this method you can use the COM1 and COM2 ports of the I-8xx7 controller system to link to a PC or other HMI products. In this type of configuration, the I-8xx7 controller system becomes a Modbus slave controller system. For more information about setting up and programming for Modbus slave, please refer to Chapter 4 – “Linking The I-8xx7 To An HMI Program”.

If COM3 or COM4 is used to link to other devices that support the Modbus protocol, the I-8xx7 controller system will be the Modbus master controller. For more information about setting up and programming for Modbus master, please refer to Chapter 8 - "Linking To A Modbus RTU Or Other Devices".

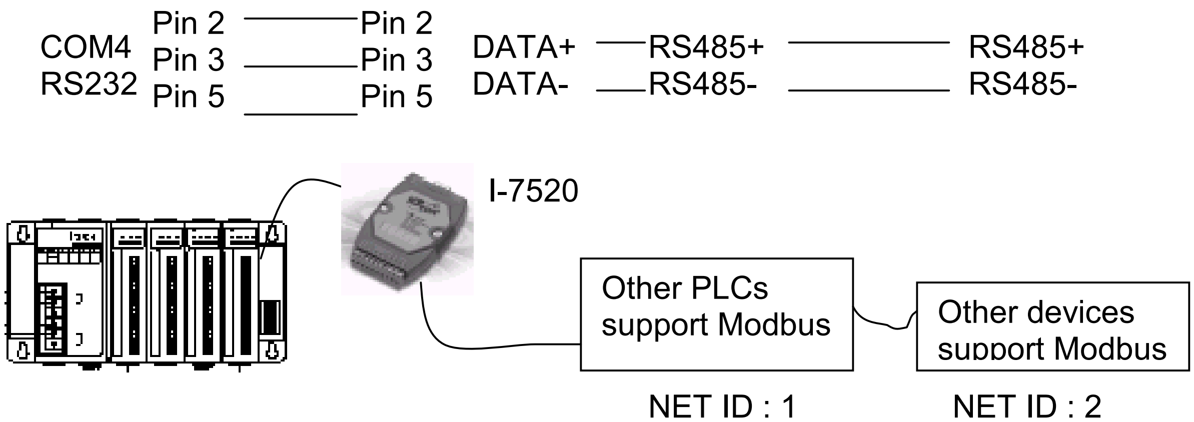
If the COM3:RS485 port is used for Modbus master, one I-8xx7 can connect to many other devices. Each device on the link must have a unique NET ID (1 ~ 255) address, and communicate at same baud rate settings.



If COM4 is used, you can only link one I-8xx7 to one other Modbus device.



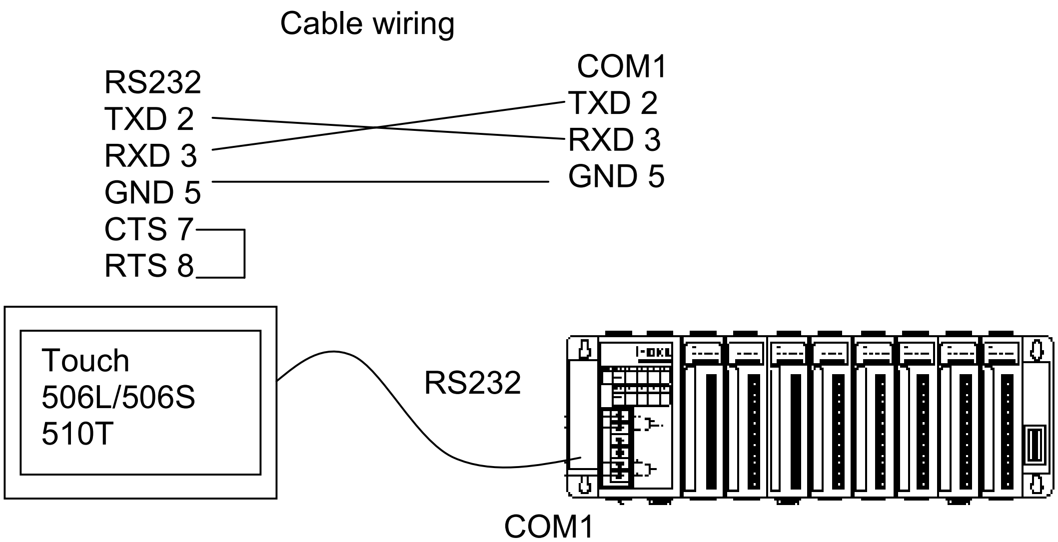
If the COM4 port of the I-8xx7 controller system is used to connect to one I-7520 remote device, then the I-8xx7 controller can network to numerous Modbus devices.



1.7: Linking To An MMI Interface Device

The COM1 (RS-232) and COM2 (RS-485) ports of the I-8xx7 controller system can be used to interface with additional Man Machine Interface (MMI) devices such as touch screen displays. ICP DAS provides a full line of touch screen displays, such as the "Touch" series screens. The models in the product line include the Touch 506L/506S and Touch 510T MMI products.

If you are using any of the “Touch” series of MMI devices to connect to an I-8xx7 controller, you can only interface the devices to the COM1 port on the I-8xx7 controller.



For more information regarding interfacing the Touch series of MMI devices to the I-8xx7 controller system, please refer to Chapter 4- "Linking The I-8xx7 To HMI Devices".

1.8: Using N-Port COM

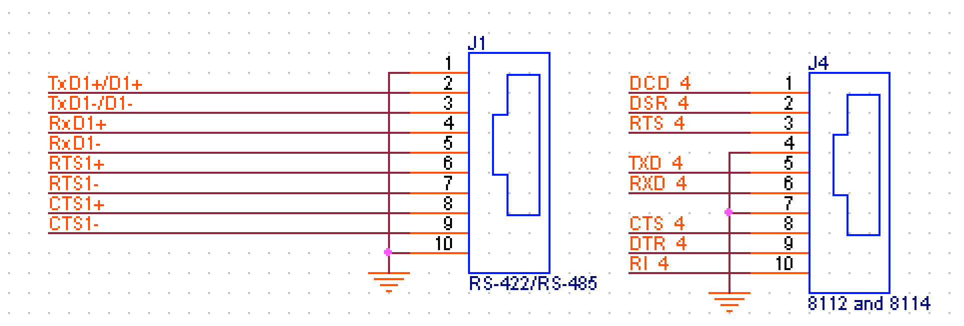
There are some N-Port COM boards that can be used to extend communication ability of the I-8xx7 controller. The model No. available are as below.

I-8112:	2-channel RS232 Module
I-8114:	4-channel RS232 Module
I-8142:	2-channel RS422/485 Module
I-8144:	4-channel RS422/485 Module

Note:
These N-Port COM boards can only be plugged into slot 0 to slot 3. It doesn't support slot 4 to slot 7. That means user can use only Com5 to Com20 of N-Port COM boards.

Some functions can be used to read/write these COM ports. Please refer to Appendix A.4 for "COMOPEN" , "COMCLOSE" , "COMREADY" , "COMARY_R" , "COMARY_W" , "COMREAD" , "COMSTR_W" , "COMWRITE" and "COMCLEAR".

Pin assignment:



Chapter 2: Getting Started

This chapter provides simple yet effective program examples of how you can use the different ISaGRAF programming languages available with the I-8417/8817/8437/8837, I-7188EG/XG & W-8037/8337/8737 controller system. The ISaGRAF programming environment provides a powerful and flexible way to create industrial control software.

For more extensive information regarding all of the capabilities of the ISaGRAF programming system, please refer to **Appendix E: “Language Reference”** of this manual or the “**ISaGRAF USER’S GUIDE**” manual which can be found from the CD_ROM of the ISaGRAF workbench. Its file name is either “ISaGRAF.pdf” or “ISaGRAF.doc”.

This manual provides some program examples and its description, please refer to Chapter 11.

2.1: A Simple Ladder Logic (LD) Program

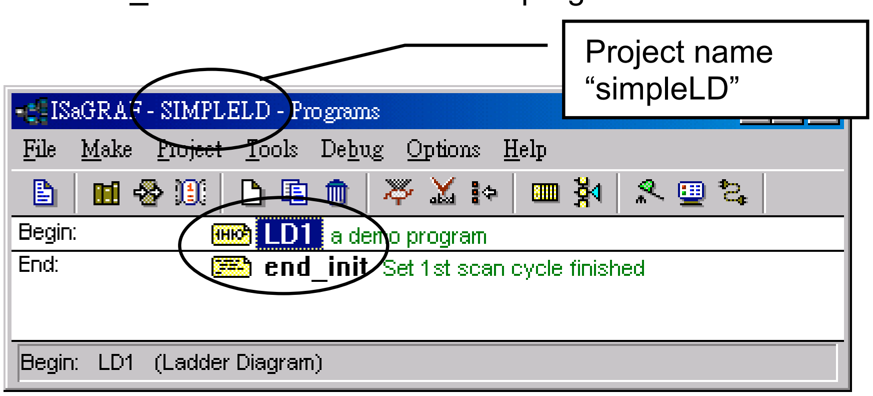
Ladder Logic Basics
"Ladder Logic" programming (LD) is a graphical representation of Boolean equations, combining **contacts** (input arguments) and **coils** (output results). Ladder Logic most closely resembles the electrical schematics that an electrician or technician may use to diagnose and troubleshoot an industrial process controller system.

The LD language enables the programmer to describe the conditions and modifications to Boolean data by placing "graphical symbols" to represent hardware devices used in a process control application.

A Simple Ladder Example Program
The following is a step-by-step example on how to create a ladder logic (hence forth referred as "LD") program using the ISaGRAF Workbench software program provided with the ISaGRAF controller system.

We will create one another Structured Text (hence forth referred as “ST”) program to indicate the first PLC scan cycle. That means in this example ISaGRAF project, we have two programs inside it. One is written in LD and the other is written in ST.

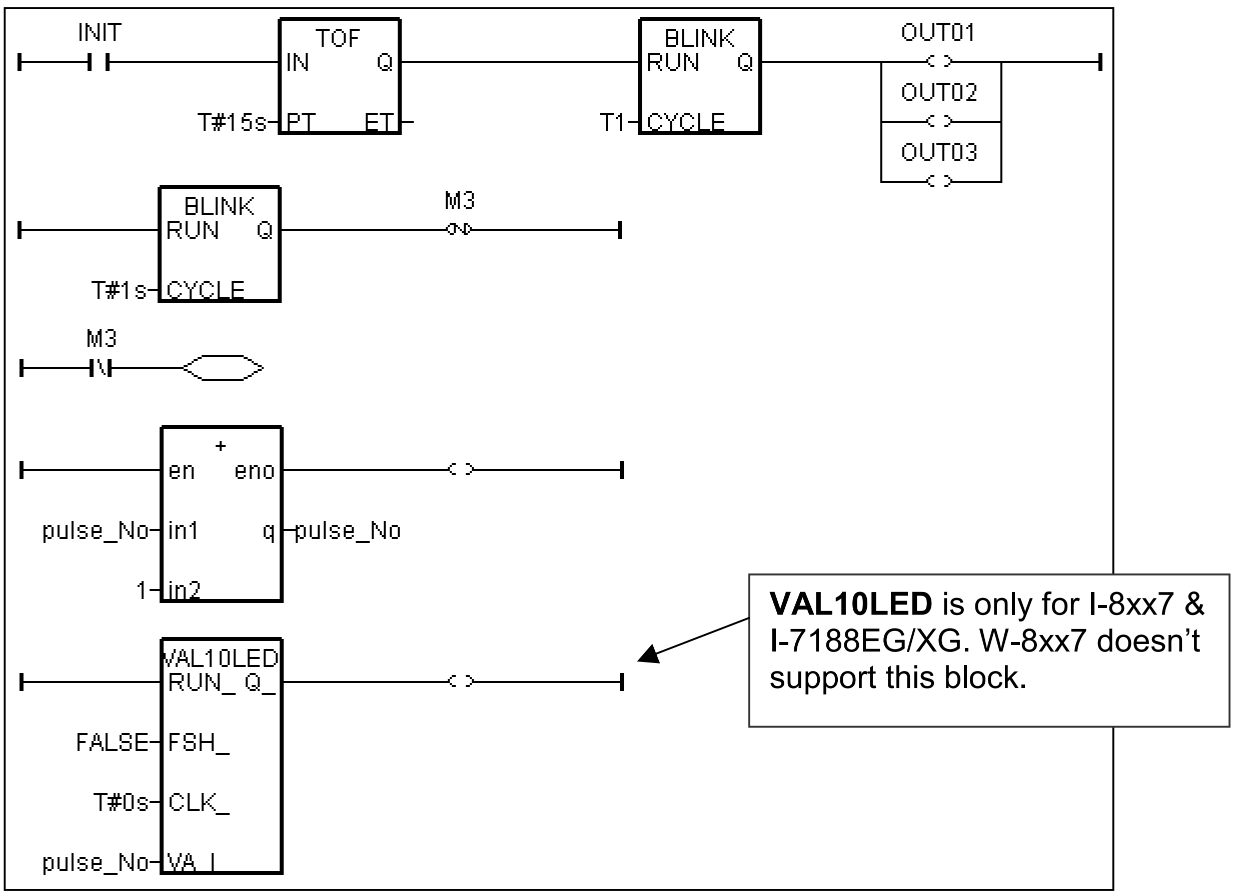
The example project name is “simpleLD”. The name of the LD program of this example project is “LD1” and “end_init” is the name of the ST program .



Variables Used In The Example LD Program:

Name	Type	Attribute	Description
INIT	Boolean	**Internal**	**initial value at "TRUE"**. TRUE means 1st scan cycle
M3	Boolean	**Internal**	Indicate a pulse is generated or not.
OUT01	Boolean	**Output**	Output 1
OUT02	Boolean	**Output**	Output 2
OUT03	Boolean	**Output**	Output 3
T1	Timer	**Internal**	Time Period of blinking, **initial value is set at "T#1s"**
Pulse_No	Integer	**Internal**	To puls one when M3 pulse is generated **initial value is set at "0"**

Ladder Logic Program "LD1" Outline:



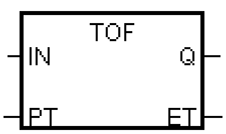
ST program "end_init" Outline:

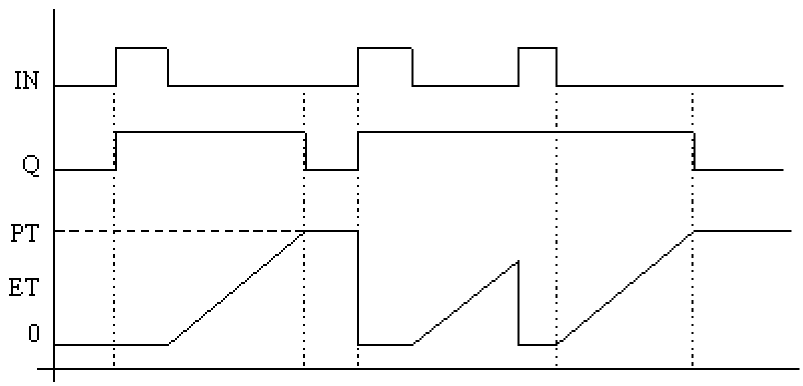
```
INIT := FALSE ;
```

Process Operation Actions:
Ladder Logic Program “LD1” :
Blink Outputs 1, 2, & 3 with a period of “T1” in the first 15 seconds, “T1” has initial value equal to 1 second. Atfer these 15 seconds, Outputs 1, 2, & 3 will be turned OFF.
Generate a pulse output every 1 second to the internal boolean variable “M3”.
Plus integer variable “pulse_No” by 1 every time when “M3” pulse is generated.
Display the value of “pulse_No” to the 7-Seg leds of the I-8xx7 or I-7188EG/XG controller.

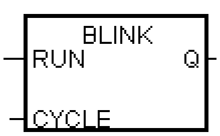
ST Program “end_init” :
Set boolean variable “INIT” to FALSE at the end of the PLC scan cycle. So that “INIT” will be TRUE only at the first scan cycle.

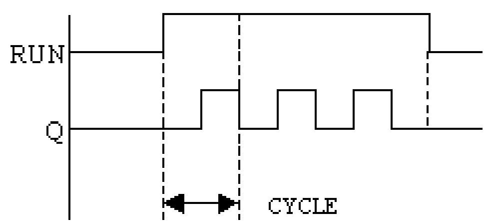
Description of block and some basic LD item:
TOF:To turn off a boolean however delay a time of “PT”.
“IN” is a boolean parameter, if falling from TRUE to FALSE. The timer ticks from 0 to “PT”
“PT” is a timer parameter, it defines the delay time of output.
“Q” is the boolean output of this block. It will be turned OFF when “PT” is reached.
“ET” is the timer output of this block. (We don’t use it in this example)



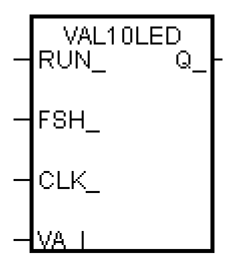


BLINK: To blink a boolean with a period of “CYCLE”.
“RUN” is a boolean parameter, if it is TRUE, the boolean output “Q” will be blinking at period of the timer parameter “CYCLE”.

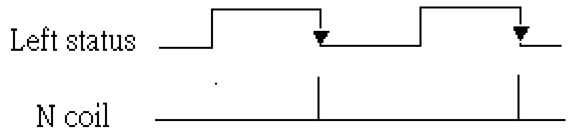




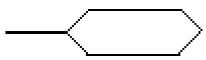
VAL10LED: Display a interger value to the 7-Seg leds of the controller.
“RUN_” is a boolean parameter. TRUE to display.
“FSH_” is a boolean parameter. TRUE to blink the display.
“CLK_” is a timer parameter. It defines the blinking period.
“VA_I_” is the integer to display.



“N” coil : Coil with N type means it will be set to a pulse TRUE when the left status is just falling from TRUE to FALSE.

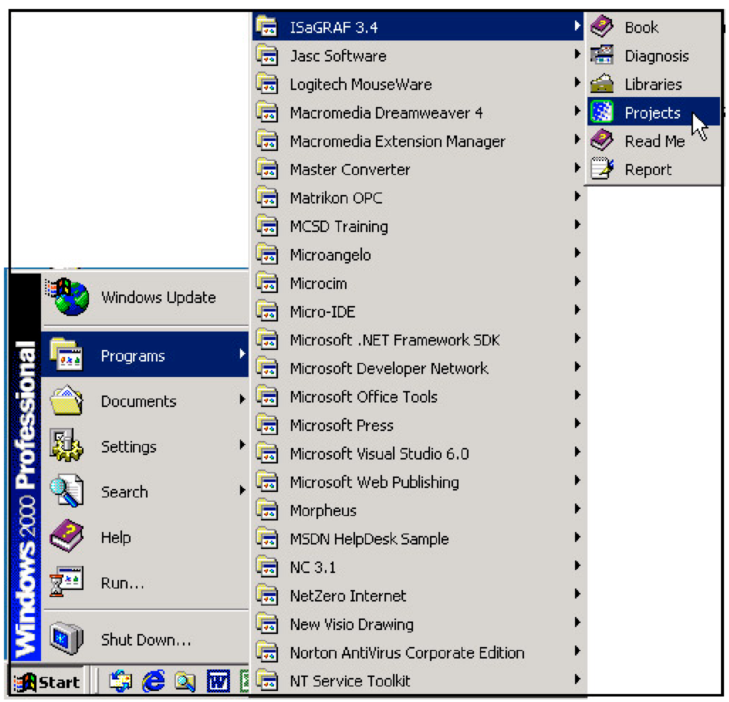


“Retrun” : To return from the excution if the left status is TRUE, that is, the reset LD rungs of the program below this “return” will not excute when the left status is TRUE.



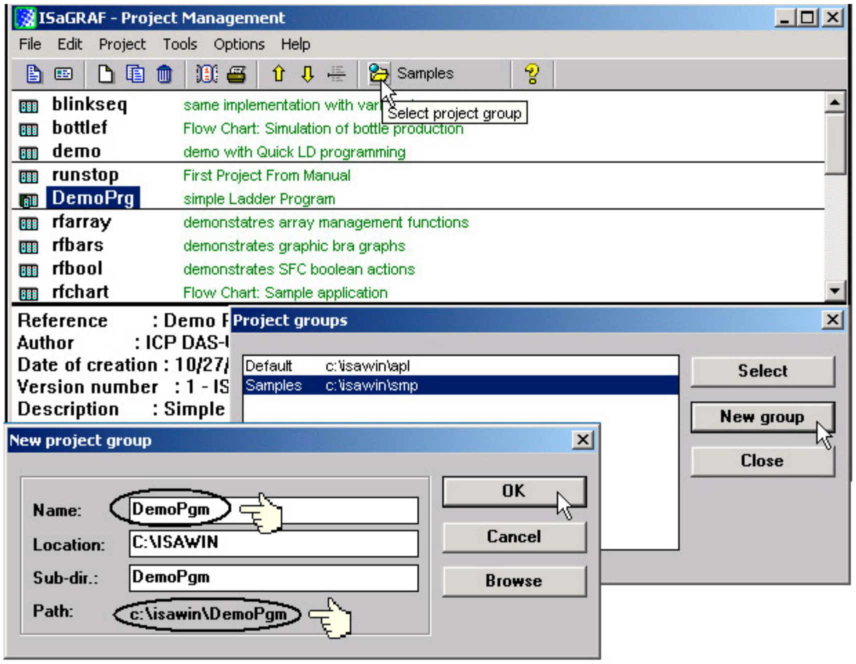
2.1.1: Programming LD

Starting & Running The ISaGRAF Workbench Program
Click on the Windows "Start" button, then click on "Programs", then click on "ISaGRAF 3.4", then click on "Projects" as shown below.



2.1.1.1: Creating An ISaGRAF User's Group

Click on the "Select Project Group", and then click on "New Group", then type in the name for the new user's group you wish to create, and last click on "OK".

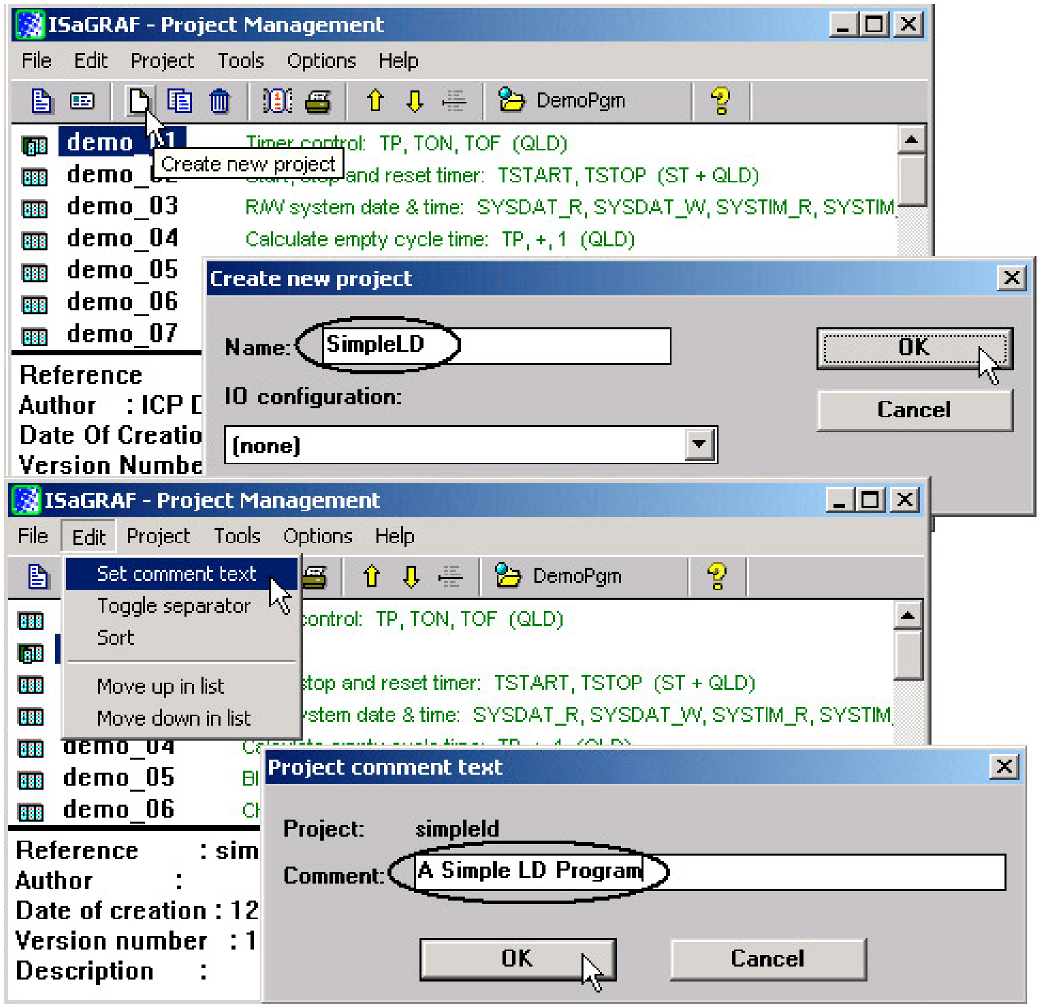


Note that the name that you give the "New Project Group" also creates a new sub-directory corresponding to the project group name in the "c:\isawin" sub-directory.

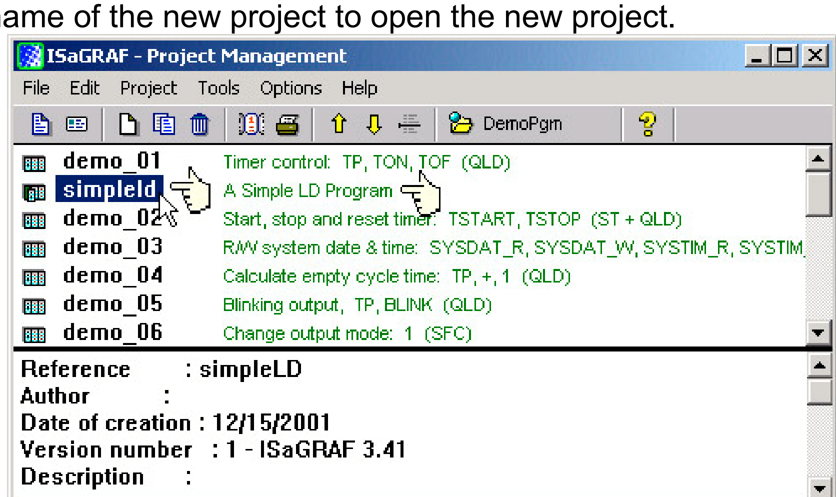
To get into the new project group, either double click on the new group name, or click on the new group name (the name will be highlighted) to select the new project group and click on the "Select" button.

2.1.1.2: Creating A New ISaGRAF Project

To start a new ISaGRAF project, click on the "Create New Project" icon and then enter in the name for the new project. You can then enter additional information for your project by clicking on the "Edit" and then "Set Comment Text" menu as illustrated below.

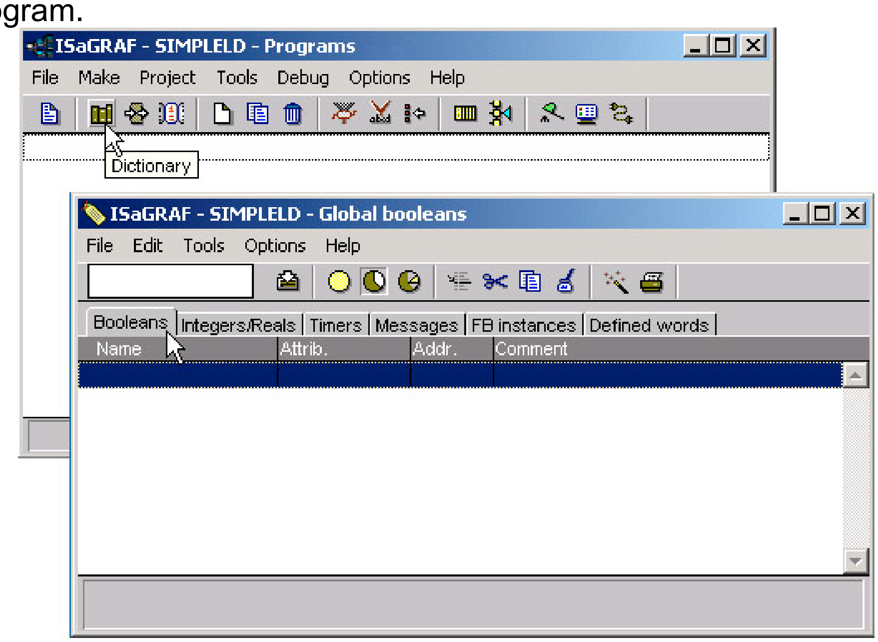


You will now see the name of the new project in the "Project Management" window. Double click on the name of the new project to open the new project.

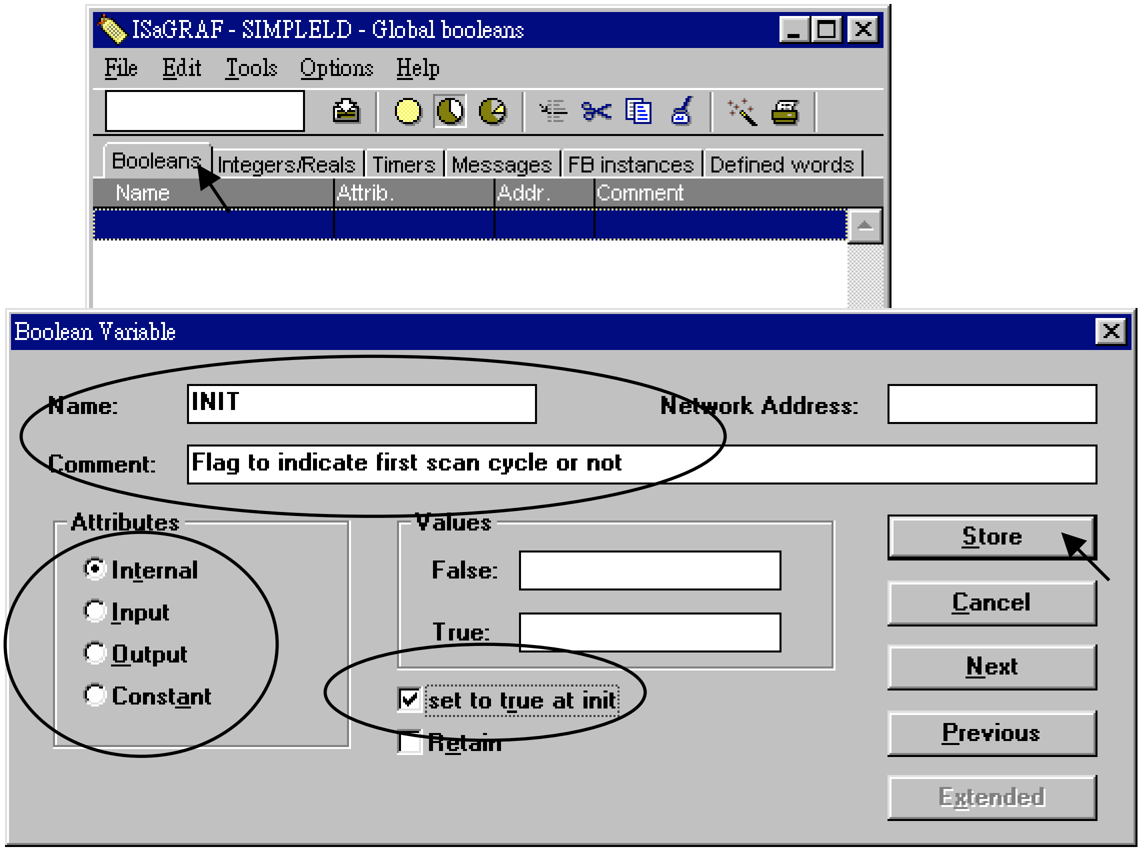


2.1.1.3: Declaring The ISaGRAF Project Variables

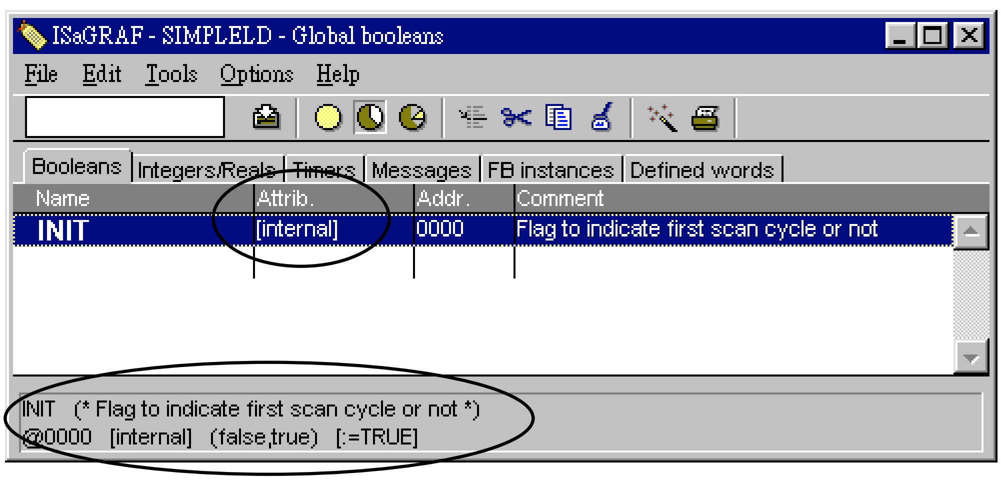
Before you can start creating an ISaGRAF program, you must first declare the variables that will be used in the ISaGRAF program. To begin this process, first click on the "Dictionary" icon and then click on the "Boolean" tab to declare the Boolean variables that will be used in our example program.



To declare the program variables for the ISaGRAF project, double click on the colored area below the "Boolean" tab, and a "Boolean Variable" window will open. Enter in the name of the variable to be used in the project. For the purpose of this example program the variable "Boolean Variable Name" is "INIT", and "Flag to indicate first scan cycle or not" is added to the "Comment Section". The next item that must be declared is what type of "Attribute" the variable will possess. In this example program, INIT's attribute will be an "Internal". Lastly, check on the "set to true at init" since we need INIT has its initial value as TRUE when the project is just power up to run. Then press the "Store" button to save the Boolean variable that has been created.

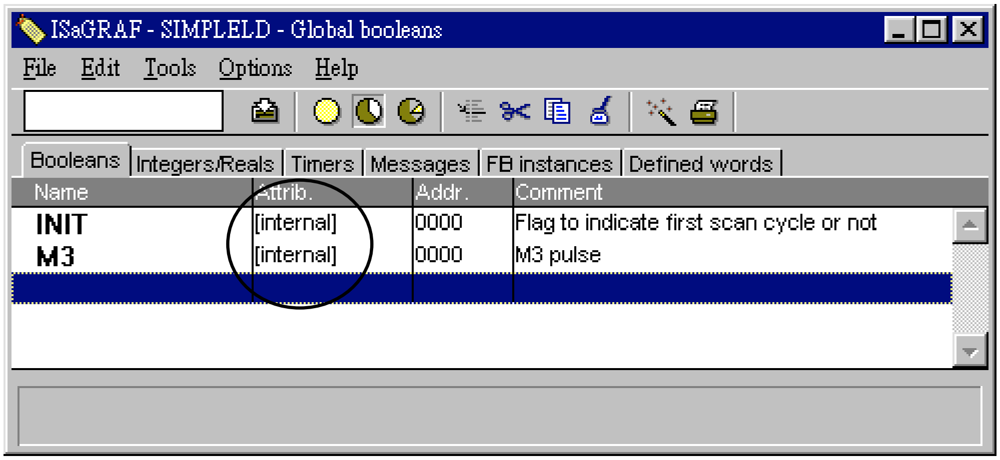


The new Boolean variable has now been declared. Note the other information areas that are provided for the programmer to fully explain how the variable will be handled.

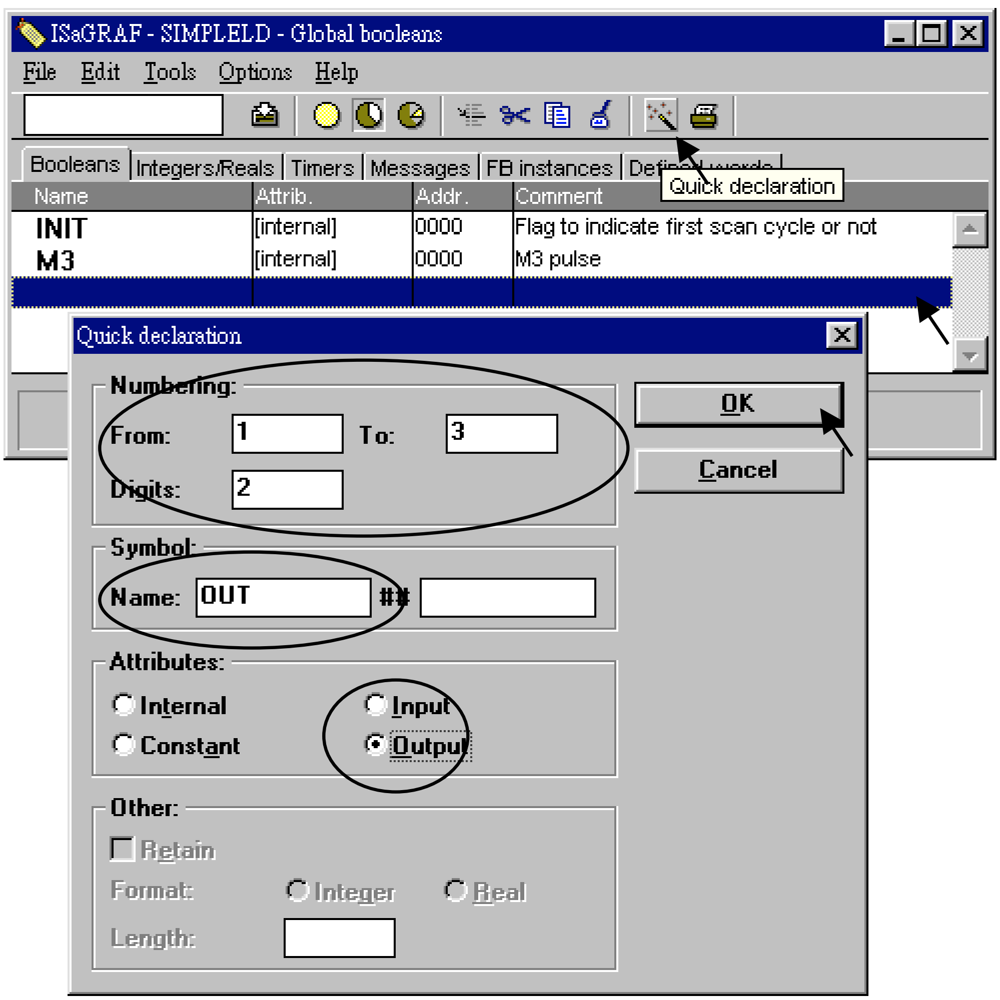


NOTE: You MUST make sure that the variable you have declared has the desired Attribute assigned. If you decide that you want to change a project variable's attribute, just double click on the variable name and you can reassign the attribute for the variable.

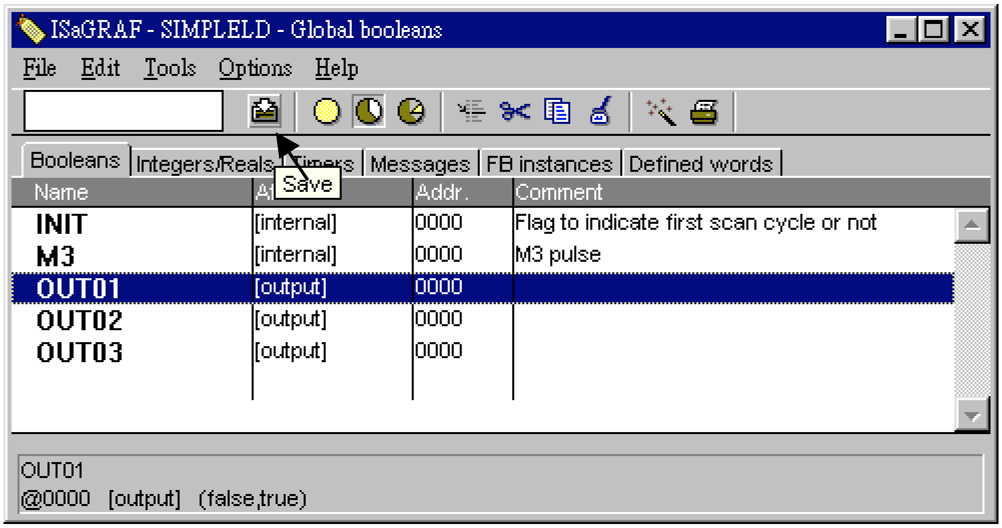
Using the same method described above, declare the additional Boolean variables for this example program, "M3". When you have completed the Boolean variable assignments, the Global Boolean window should look like the example below.



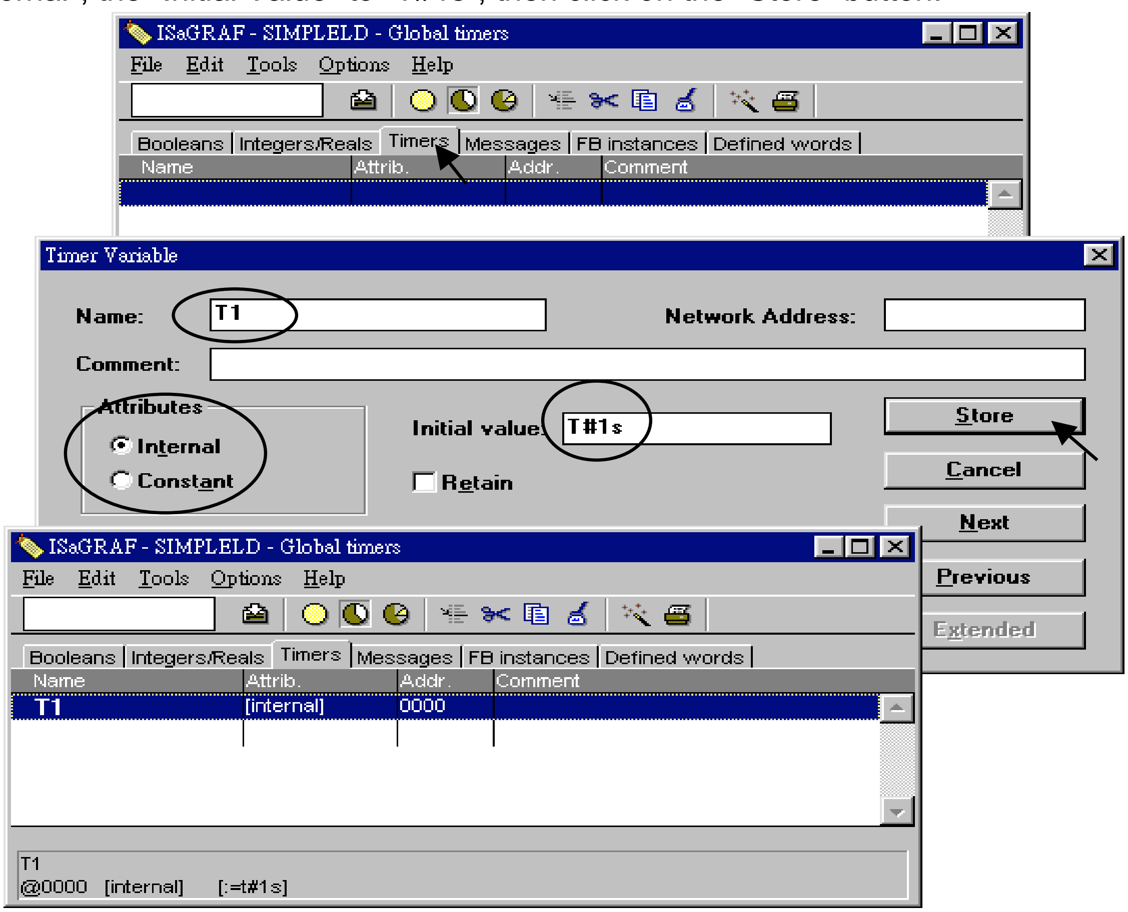
There are three outputs used in this example program named "OUT01, OUT02, and OUT03". ISaGRAF provides a quick and easy way to declare like variables that are sequentially ordered. To begin this process, click on the "Quick Declaration" icon, and enter in the output number that you will start with in the "Numbering" from and "To" field (this example uses from 1 to 3). Enter the "Symbol" name for the output variables being declared, and lastly, set the attribute to "Output".



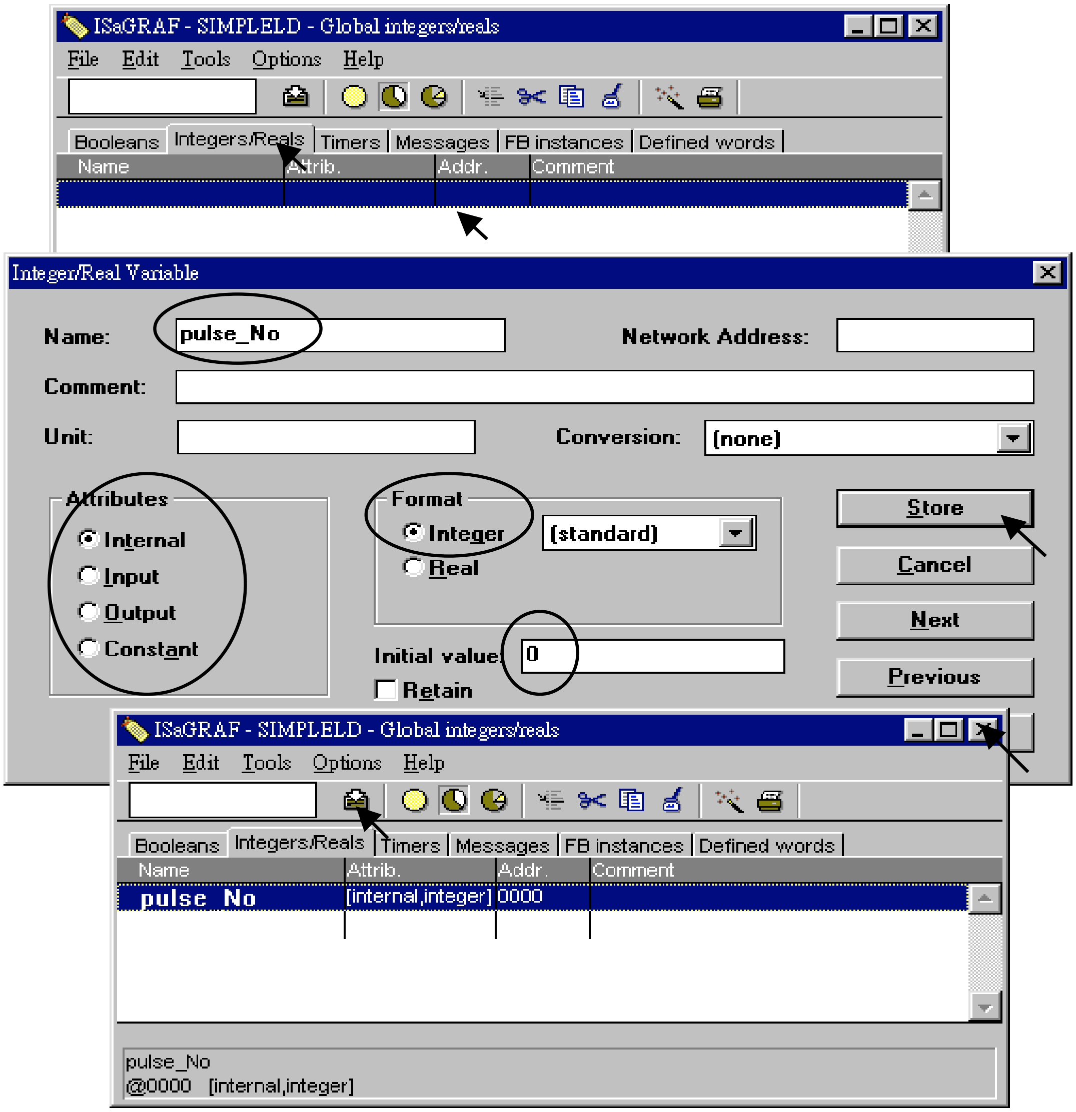
When you click on the "OK" button, all three outputs will be immediately added to the "Global Boolean" window.



To declare the timer (T1) variable used in this example program, click on the "Timers" tab in the setup screen. Double click on the colored area and enter the Name as "T1", set the "Attributes" to "Internal", the "Initial Value" to "T#1s", then click on the "Store" button.



To declare the Integer (pulse_No) variable used in this example program, click on the "Integers/Reals" tab in the setup screen. Double click on the colored area and enter the Name as "pulse_No", set the "Attributes" to "Internal", the “Format” to “Integer”, and the "Initial Value" to "0", then click on the "Store" button.

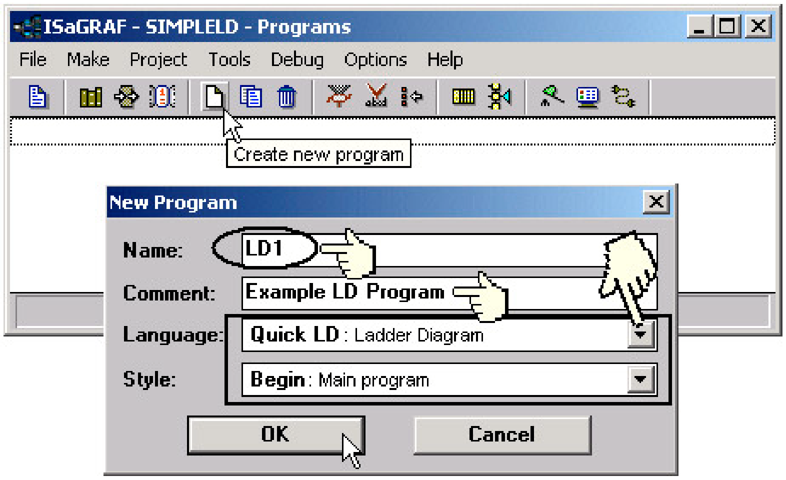


Once all of the variable characteristics have been properly setup, click on “save” and then click on "X" at the top right of the setup window to close the variable dictionary for this example project.

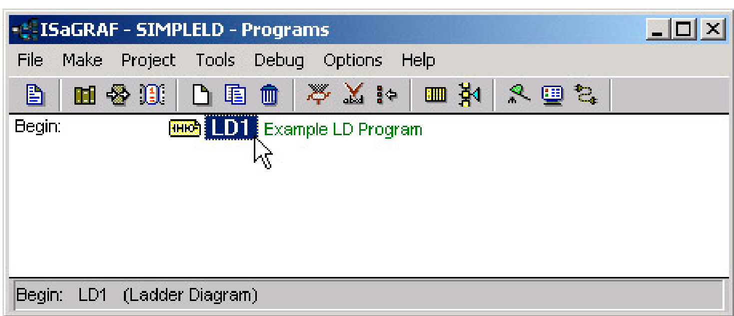
2.1.1.4: Creating The Example LD Program

Once all of the variables have been properly declared, you are now ready to create the example LD program. To start this process, click on the "Create New Program" icon and the "New Program" window will appear.

Enter the "Name" as "LD1" (the name of our example program), next, click on the "Language" scroll button and select "Quick LD: Ladder Diagram", and make sure the "Style" is set to "Begin: Main Program". You can add any desired text to the "Comment" section for the LD program, but it isn't required.

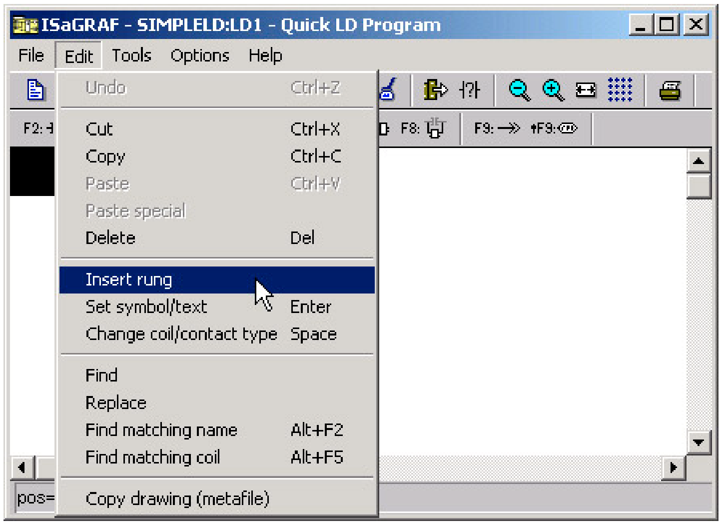


The "LD1" program has now been created. To open the "LD1" program, double click on the "LD1" name.

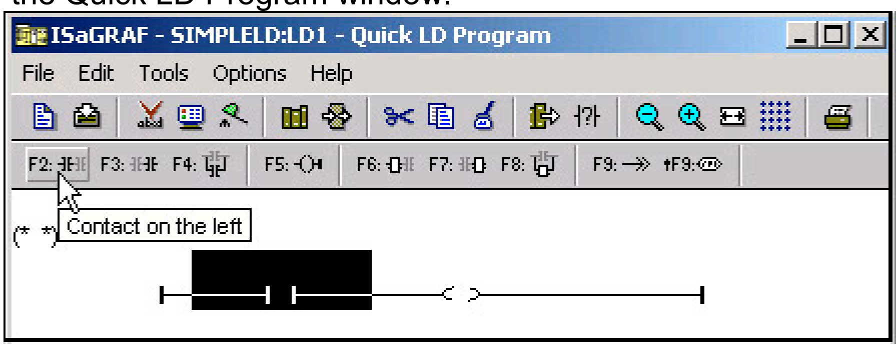


2.1.1.5: Editing The Example "LD1" Program

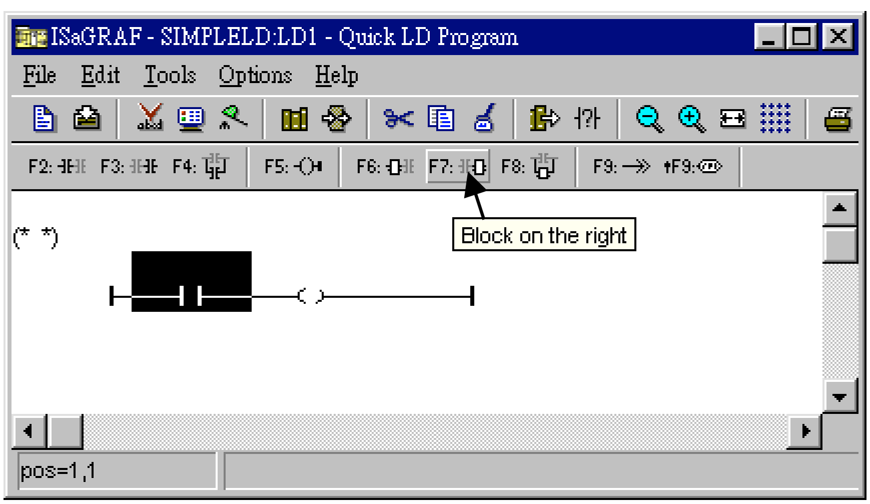
When you double click on the "LD1" name the "Quick LD Program" window will appear. To start programming our LD program, click on "Edit" from the main menu bar, then click on "Insert Rung" as shown below. "Insert Rung" means to insert a basic LD rung just above the current position.



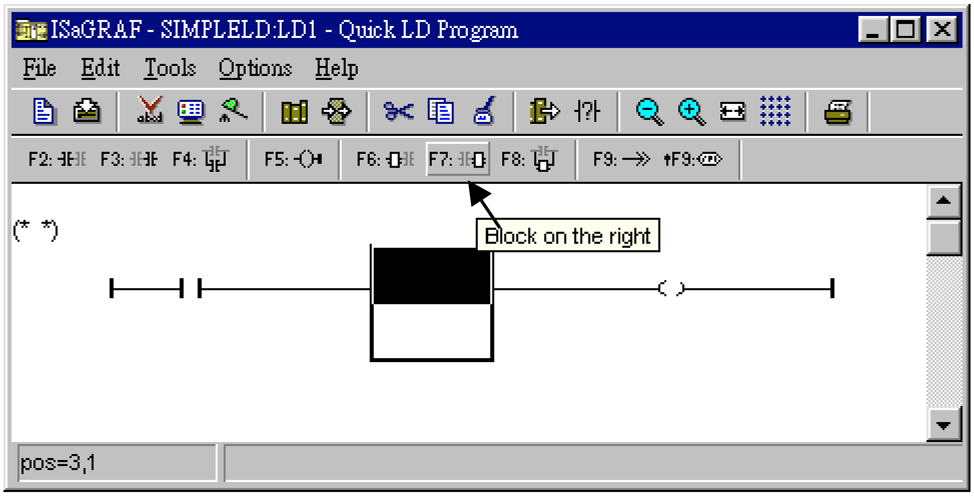
Or, you may just simply click on the "F2 (Contact On The Left)" icon, and the following will appear within the Quick LD Program window.



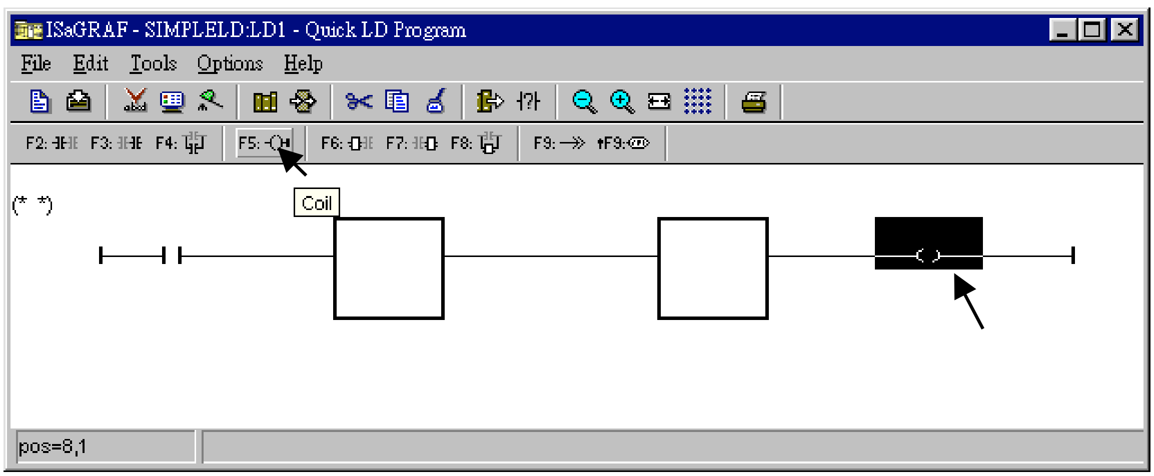
Click on the "F7 (Block on the right)" icon and you will create a block on the right of the first input contact.



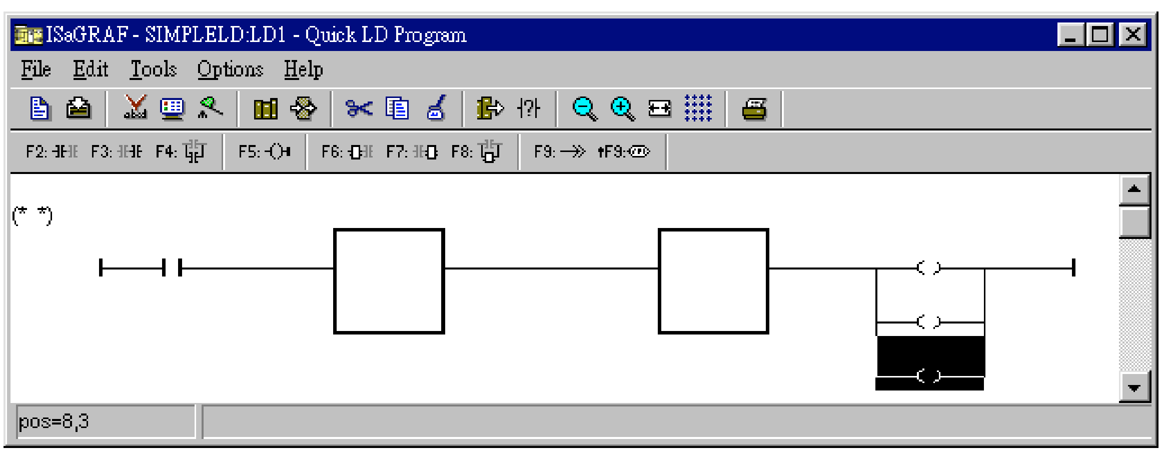
Click on "F7 (Block on the right)" icon again to create one another block on the right of the first block.



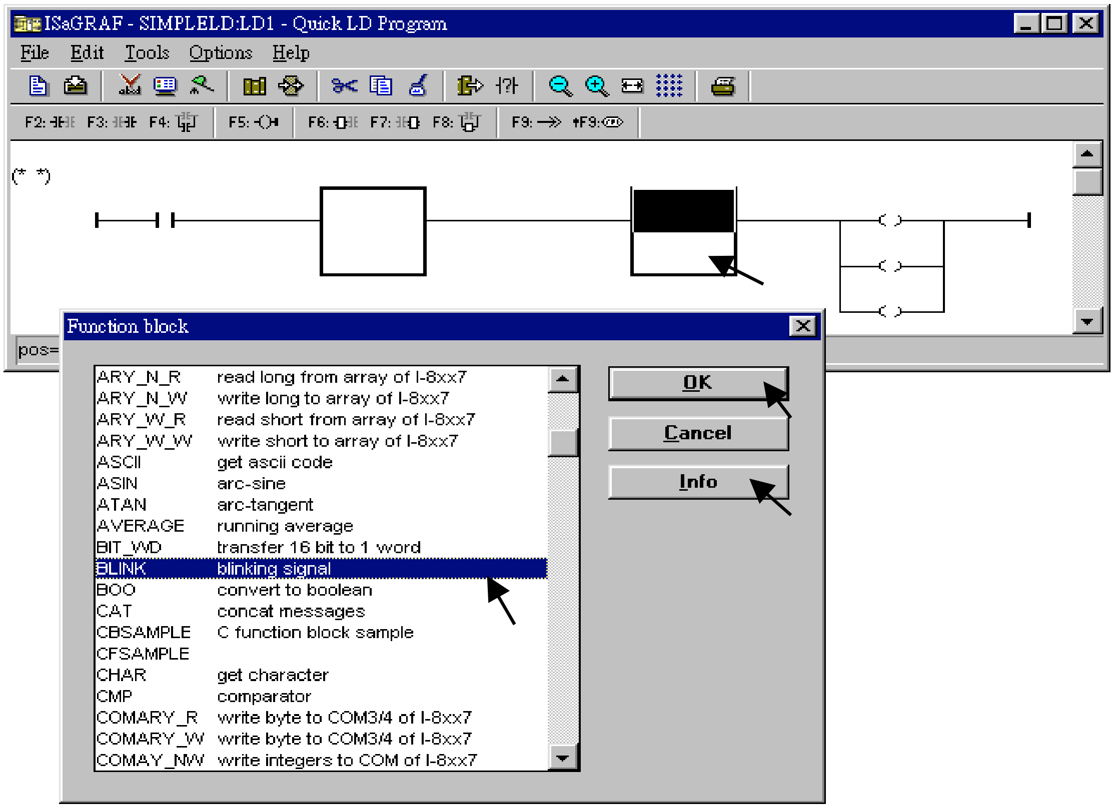
Then you will get the window as below. Move the cursor to the Coil on the right. Then click on “F5 (Coil)” to add one coil just below the first coil. And then click on “F5 (Coil)” again to add the third coil.



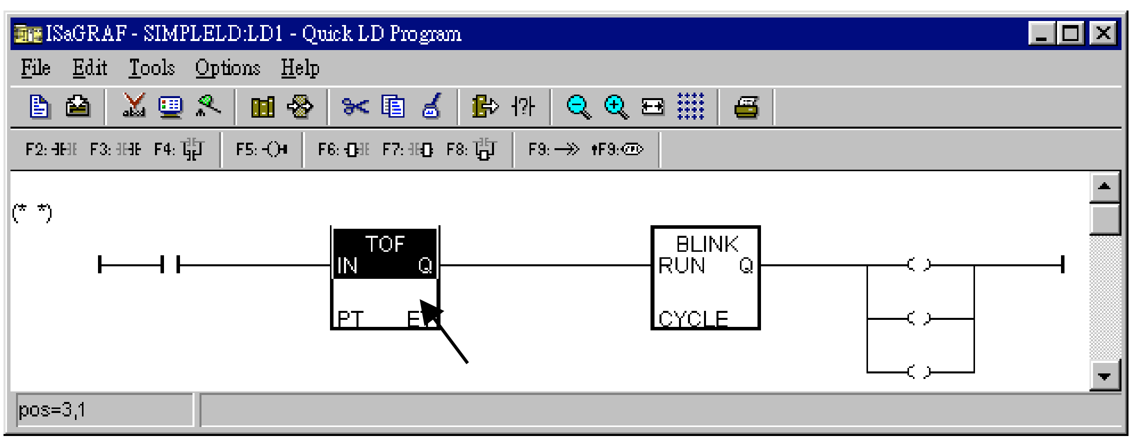
Then the window will look like below.



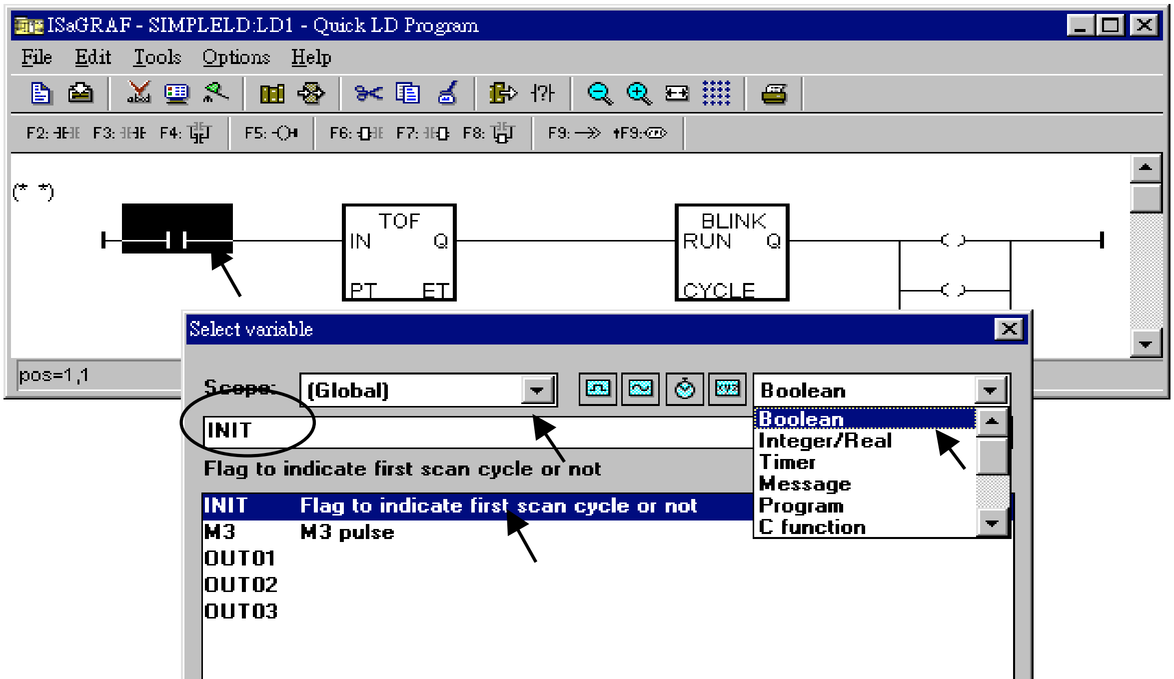
Double click anywhere inside of the second block and the "Function Block" assignment window appears. Select the "BLINK" type function block are using in our example program. To learn how the "BLINK" function operates you can click on the "Info" button for a detailed explanation of its functionality.

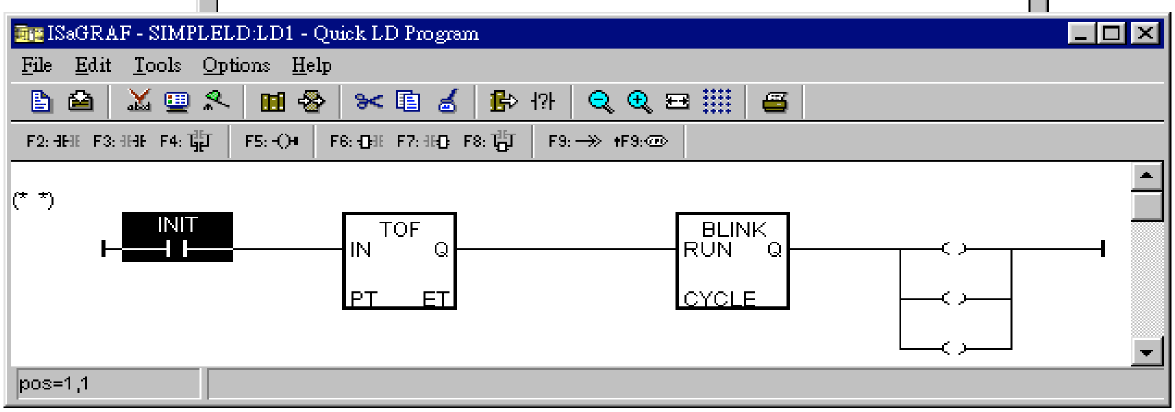


Using the same procedure to assign the first block to “TOF” as below.

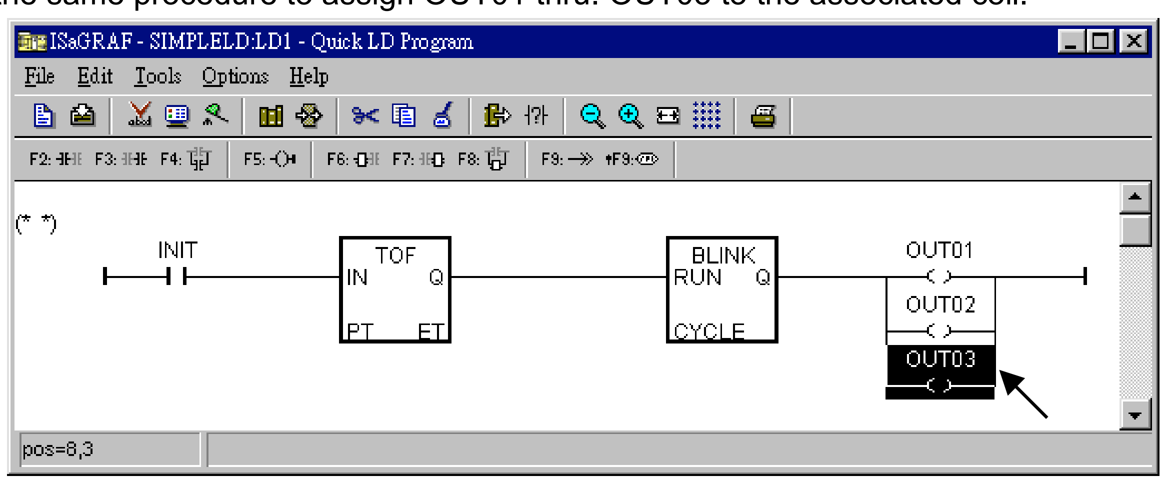


Now we are going to assign the associated variable & constant to each item. Double click on the first contact, a “Select variable” screen appeared. First select the “Scope” to “(Global)” and the proper type to “Boolean”. Then double click on “INIT” or you may use the keyboard to type “INIT”.

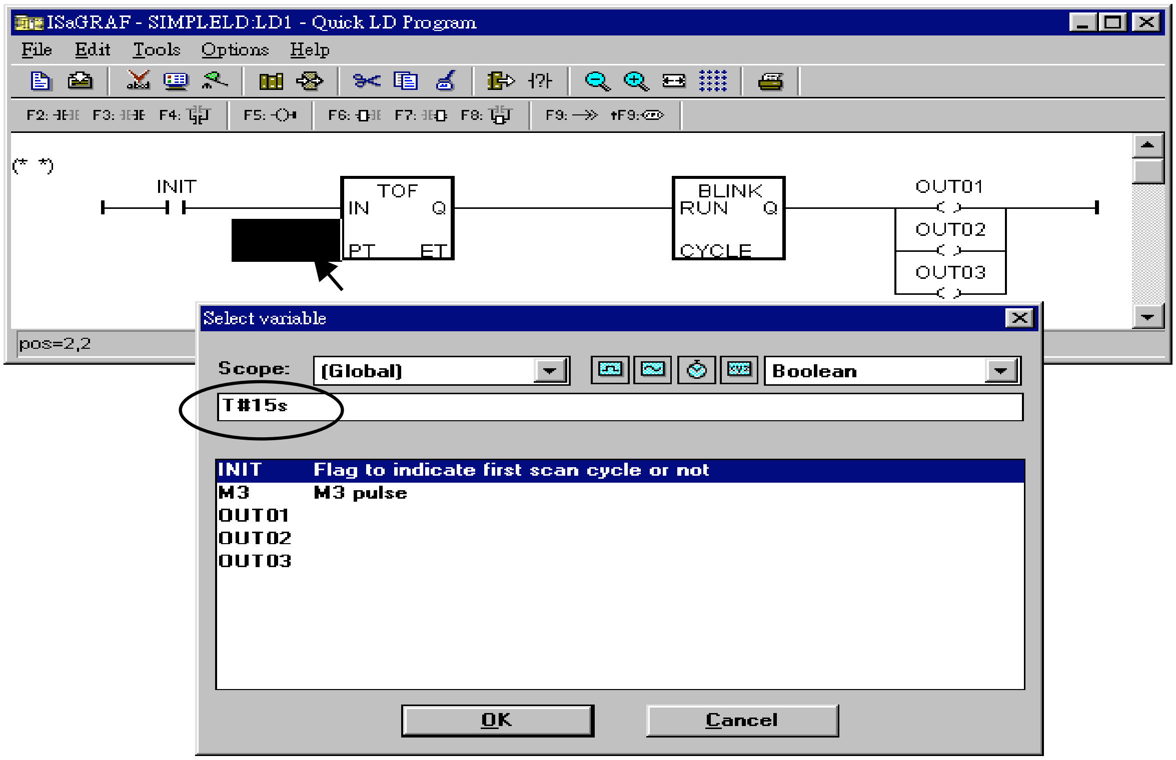




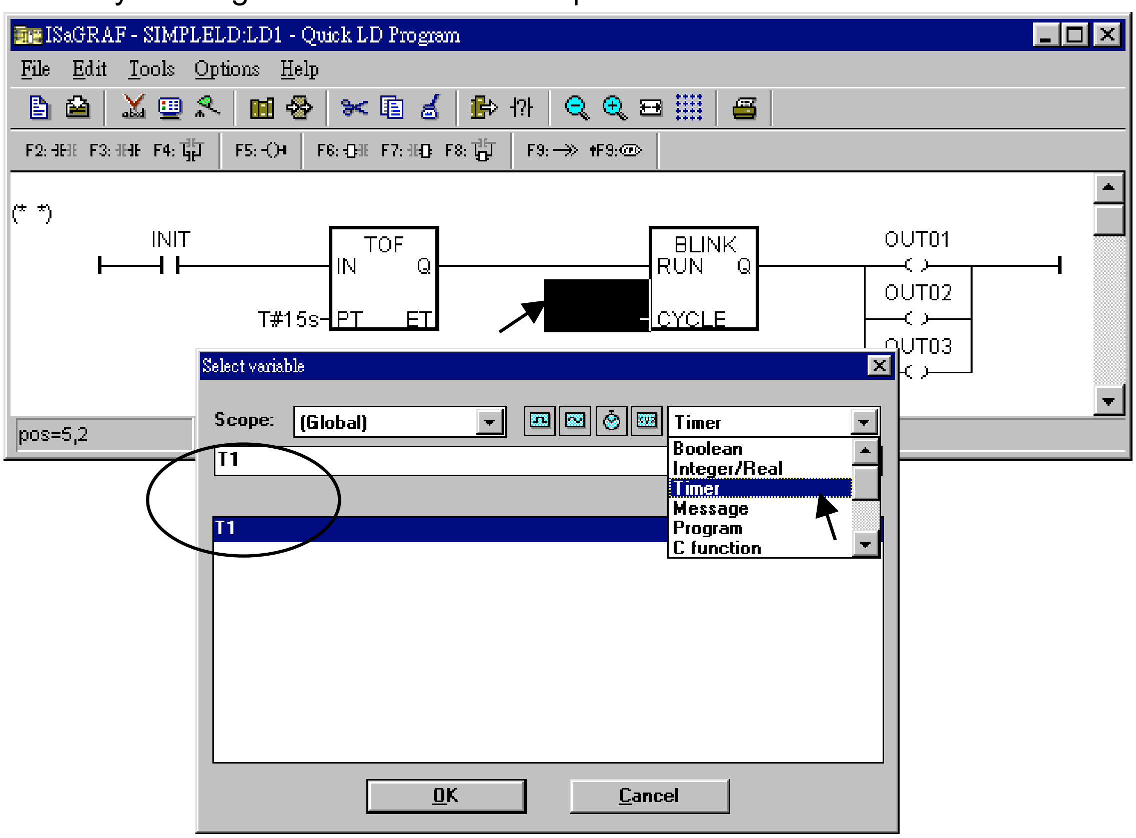
Using the same procedure to assign OUT01 thru. OUT03 to the associated coil.



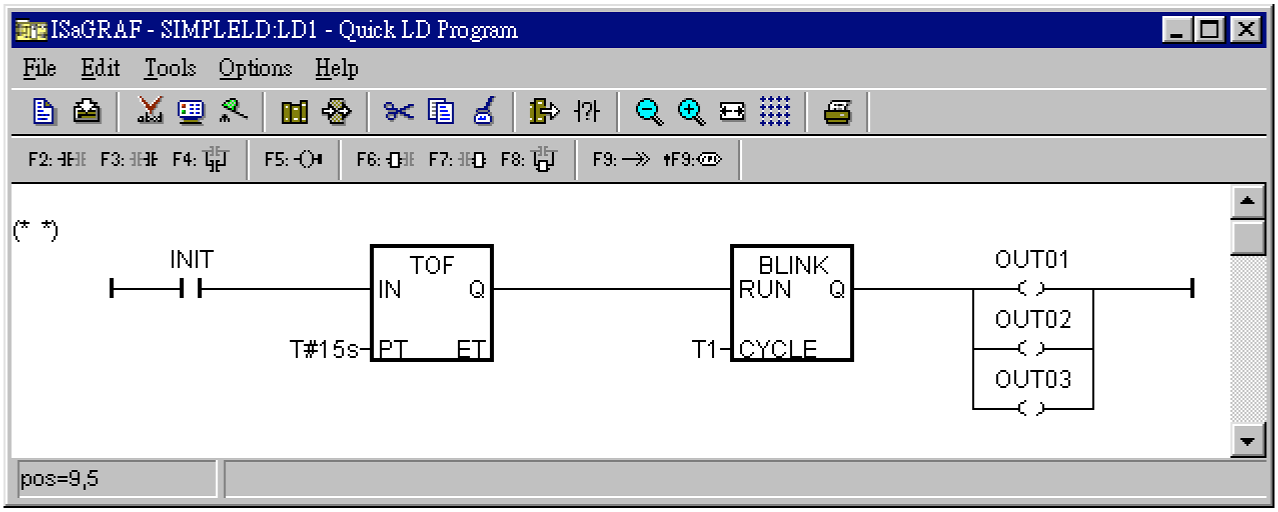
Now move your cursor to the left of the parameter “PT” of the “TOF” block. Double click on it, type “T#15s” (it means 15 second), then press “OK”.



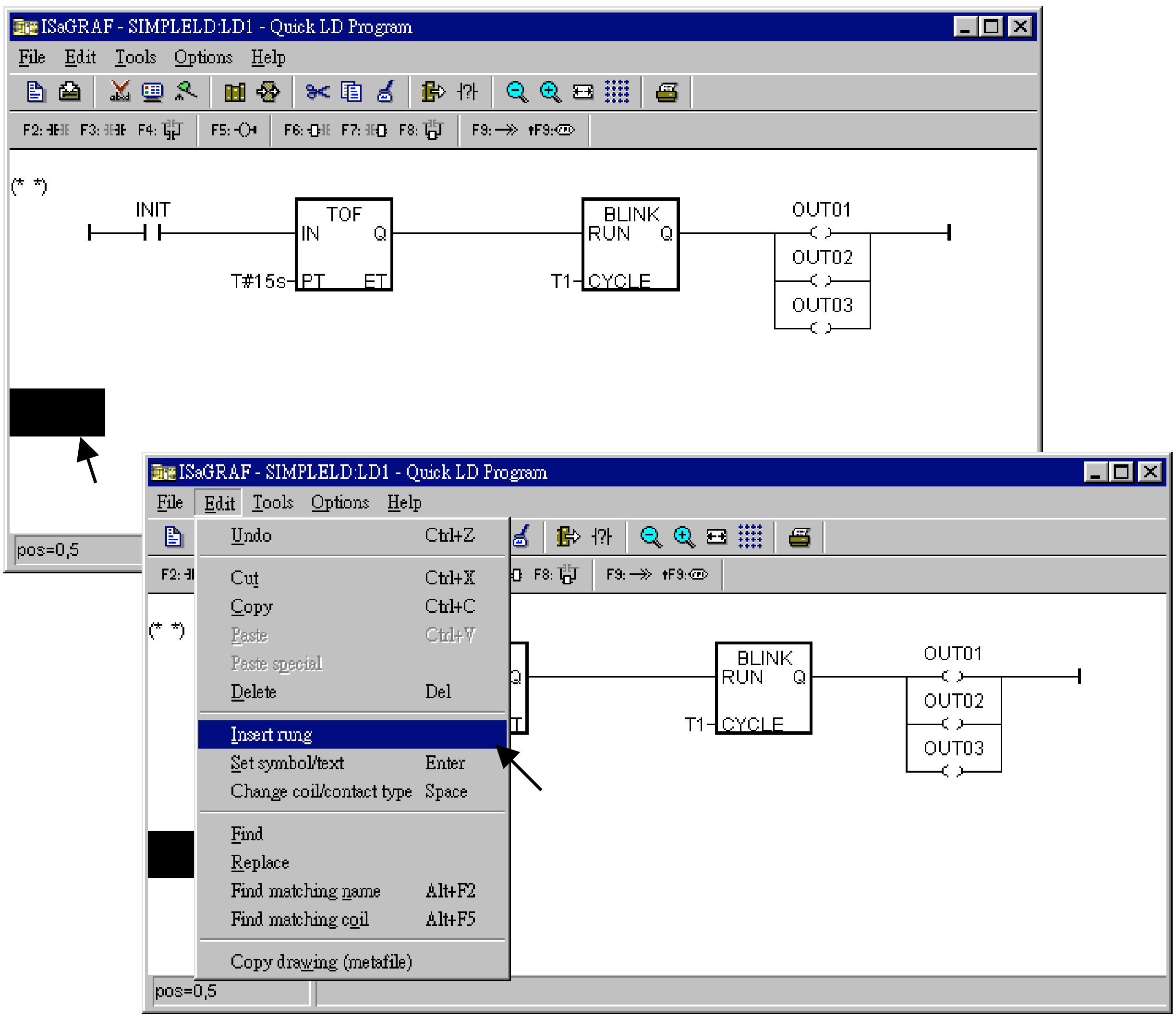
Do the same way to assign “T1” to the left of the parameter “CYCLE” of the “BLINK” block.



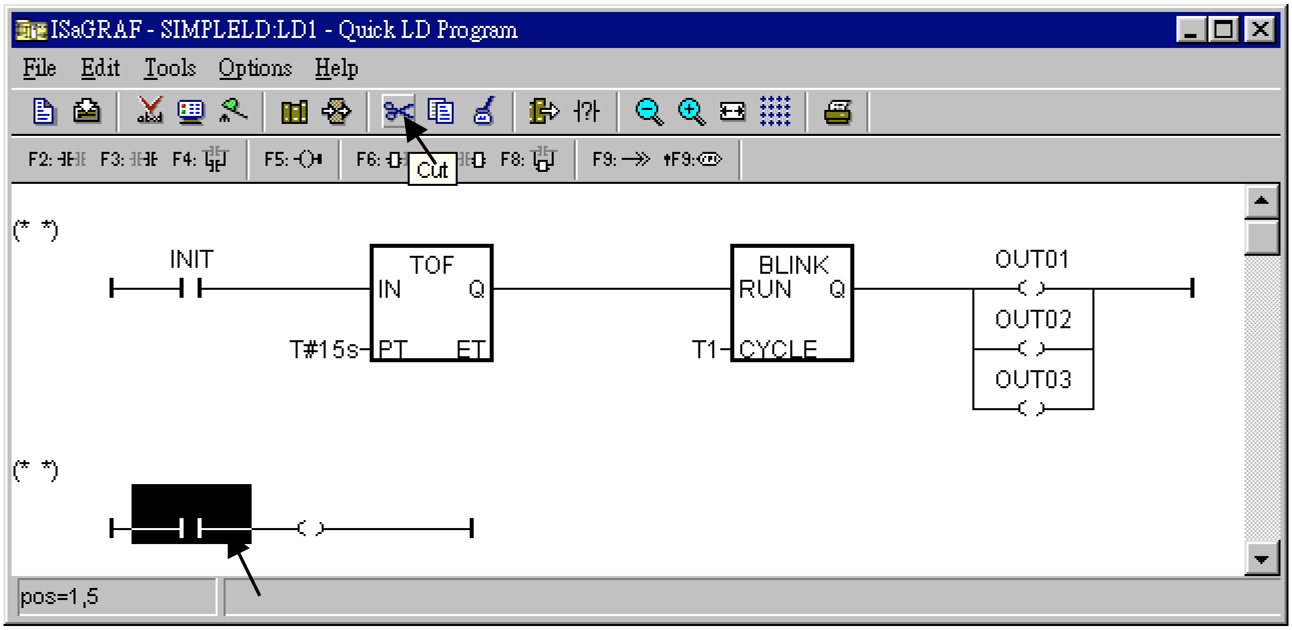
Now the window will look like below.



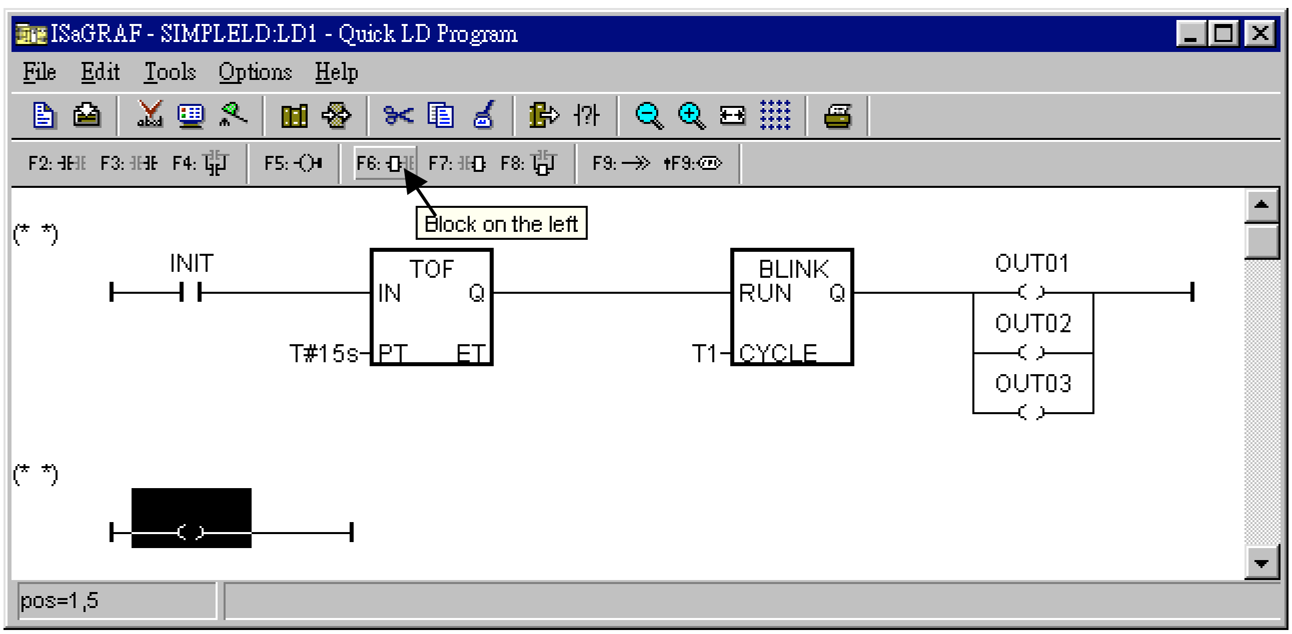
To add a new LD rung, first move the cursor to the proper position below the first rung. Then click on “Edit – Insert rung”



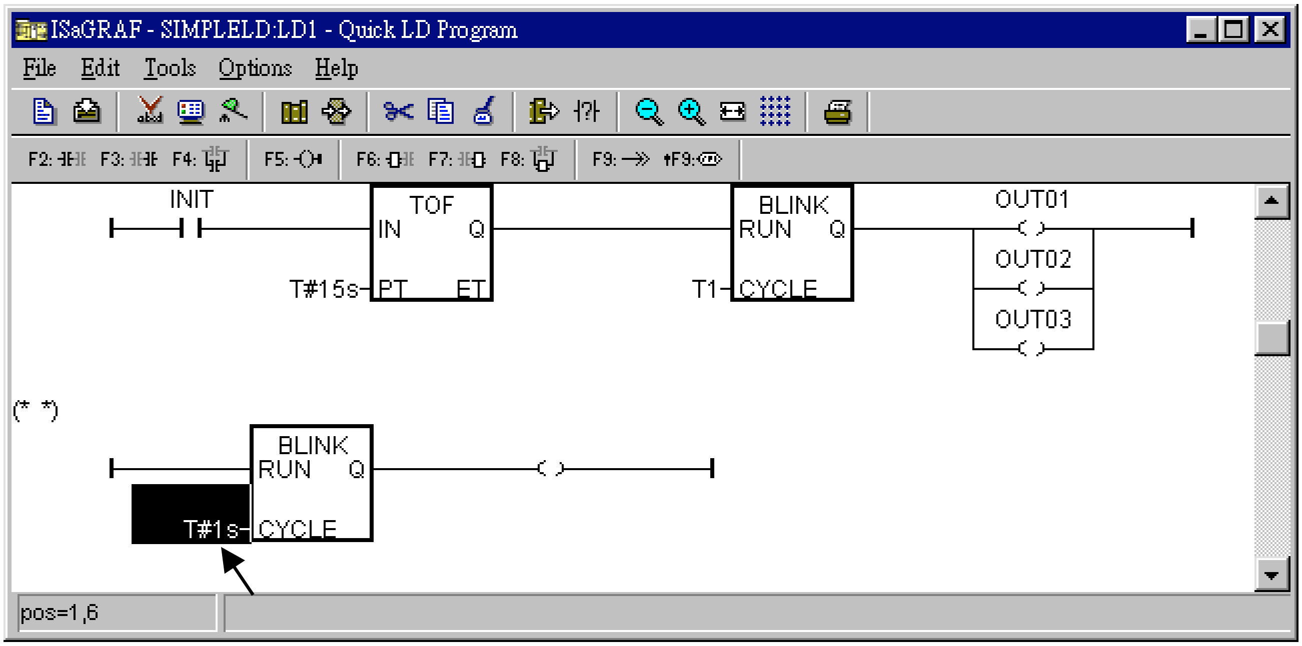
We don’t need the contact in the new rung, move cursor to it, then click on “Cut”.



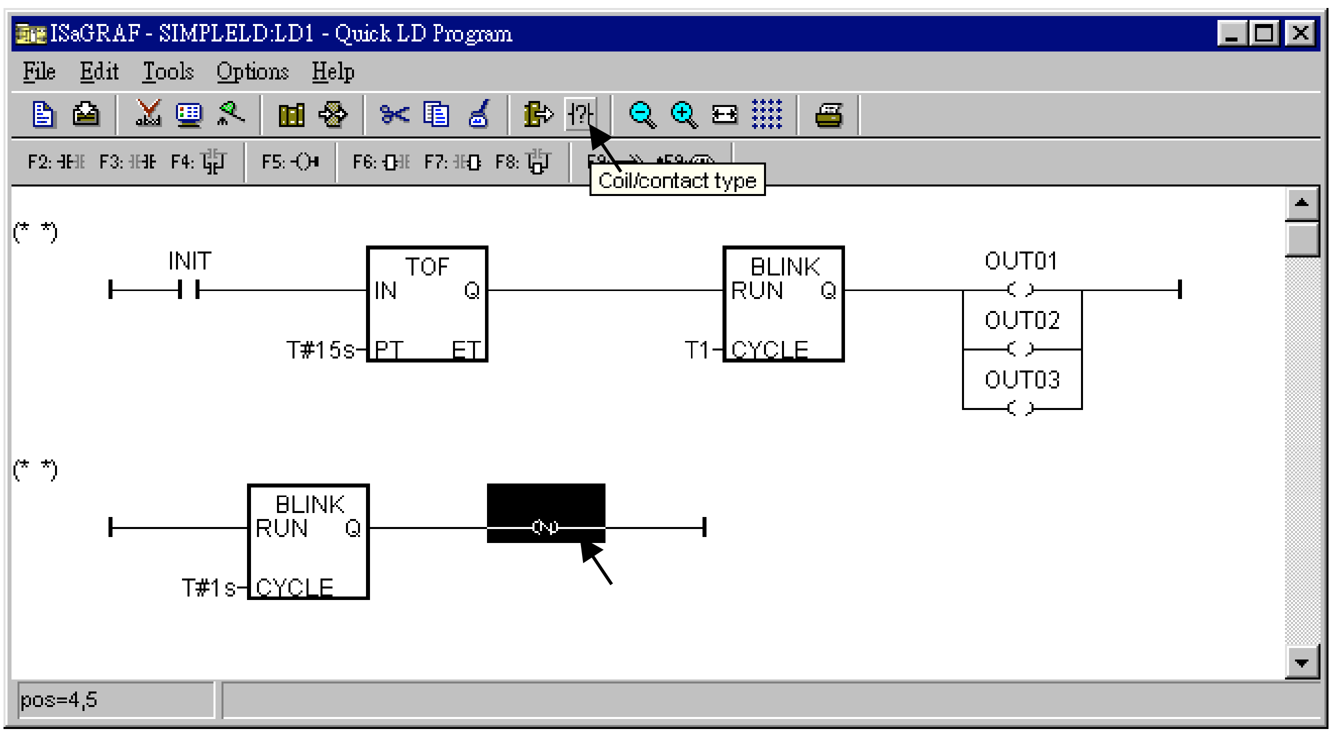
Now click on “F6 (Block on the left)” , and then double click on inside the block to create an “BLINK” block.



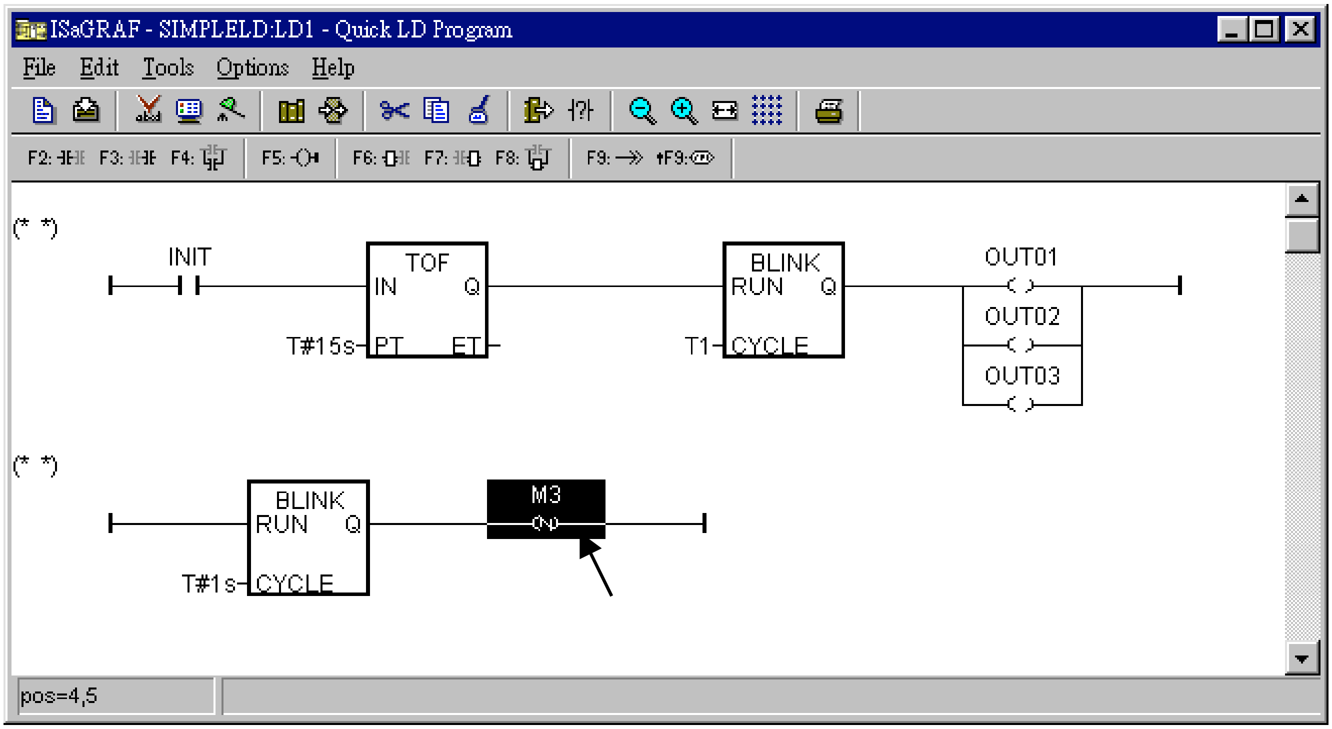
Assign “T#1s” to the parameter of “CYCLE”, then we got the below window.



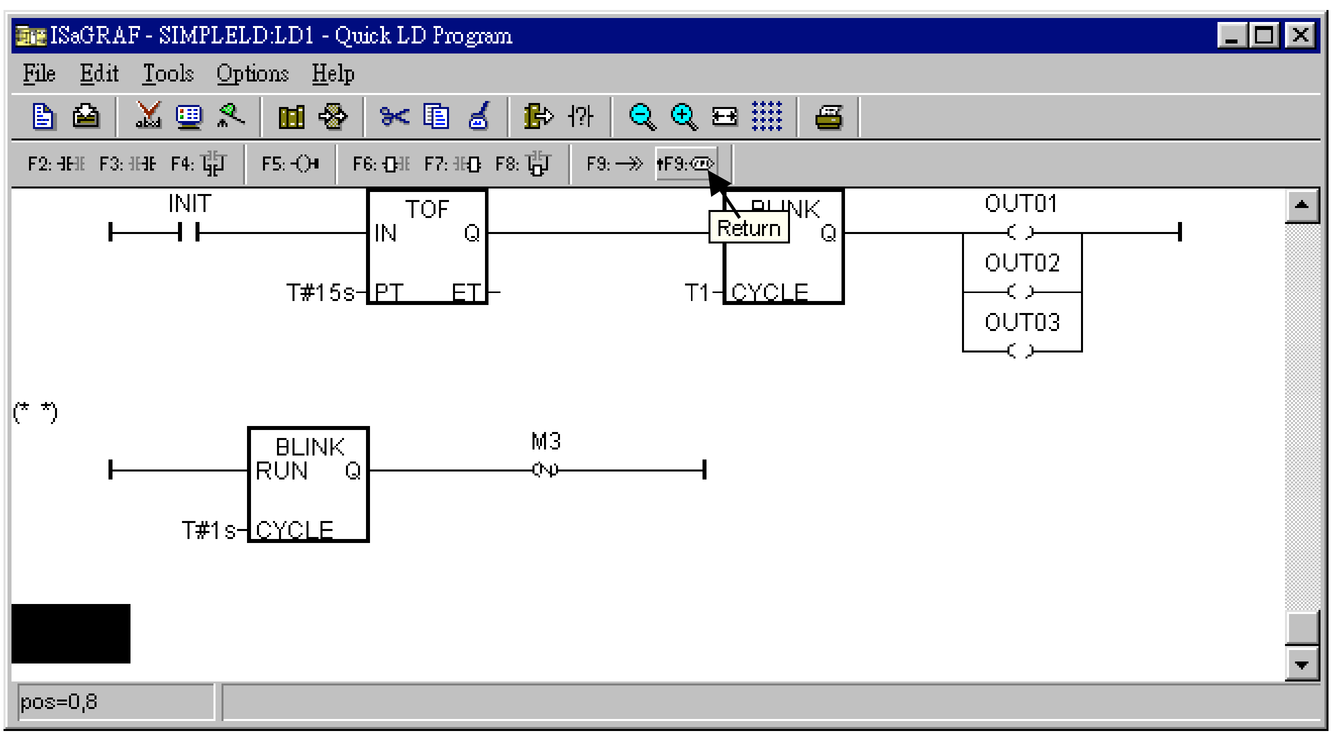
Move the cursor to the right coil, then click on “Coil/contact type” some times to assign the type to “N”.



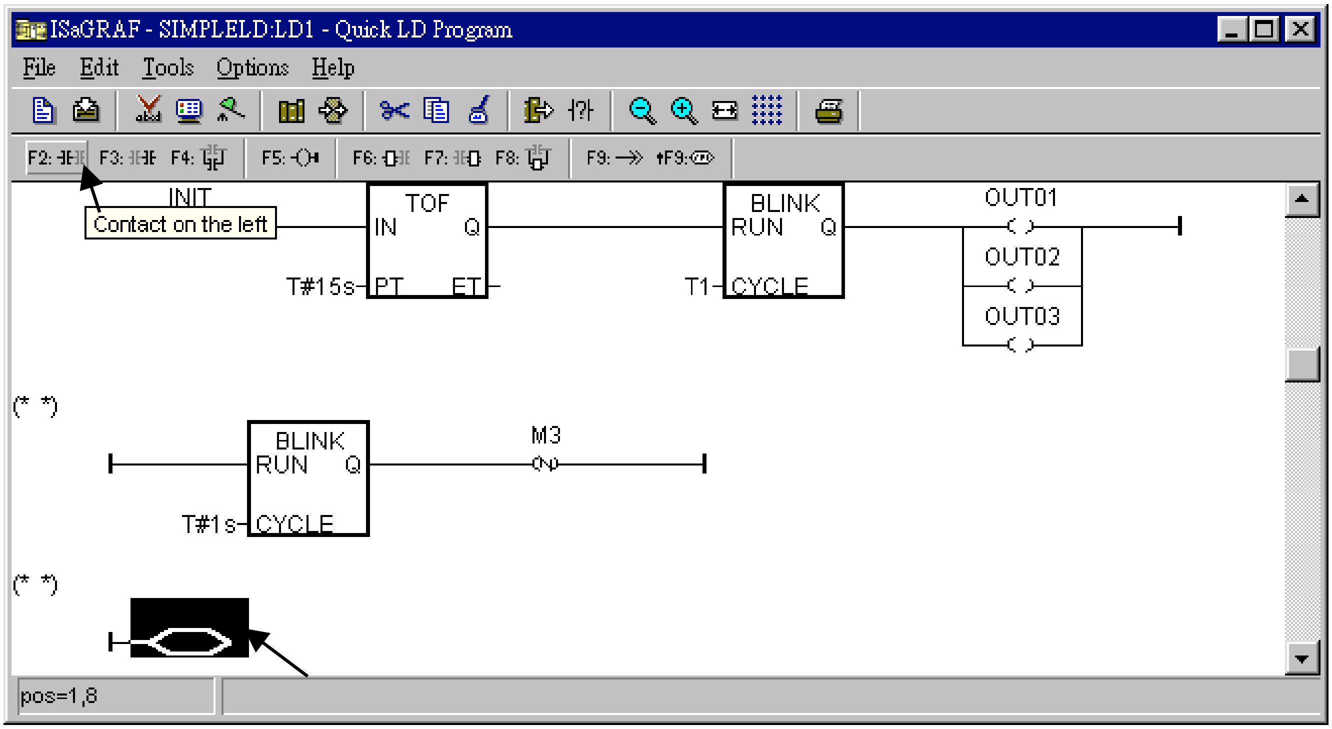
Double click on the N coil to assign “M3” to it.



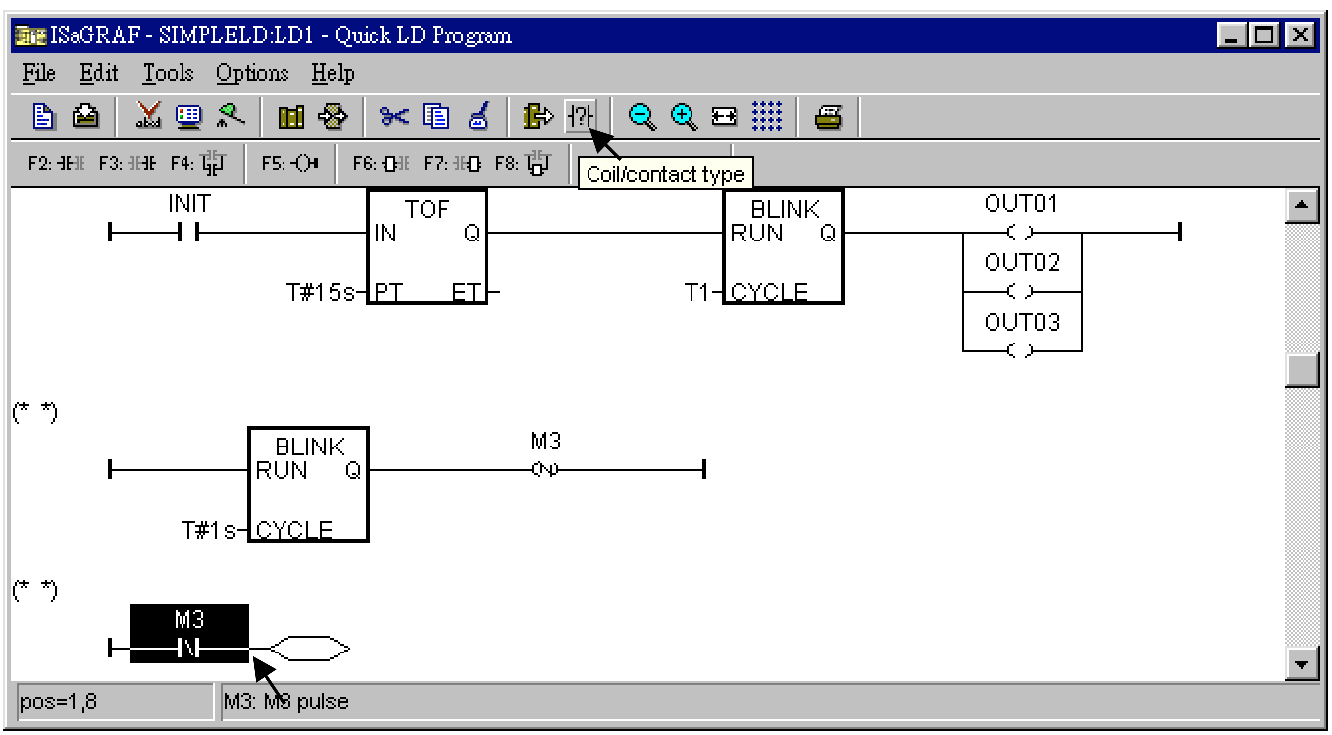
Now we are going to add another LD rung. Move the cursor to the below position of the second rung. And click on “F9 (Return)”.



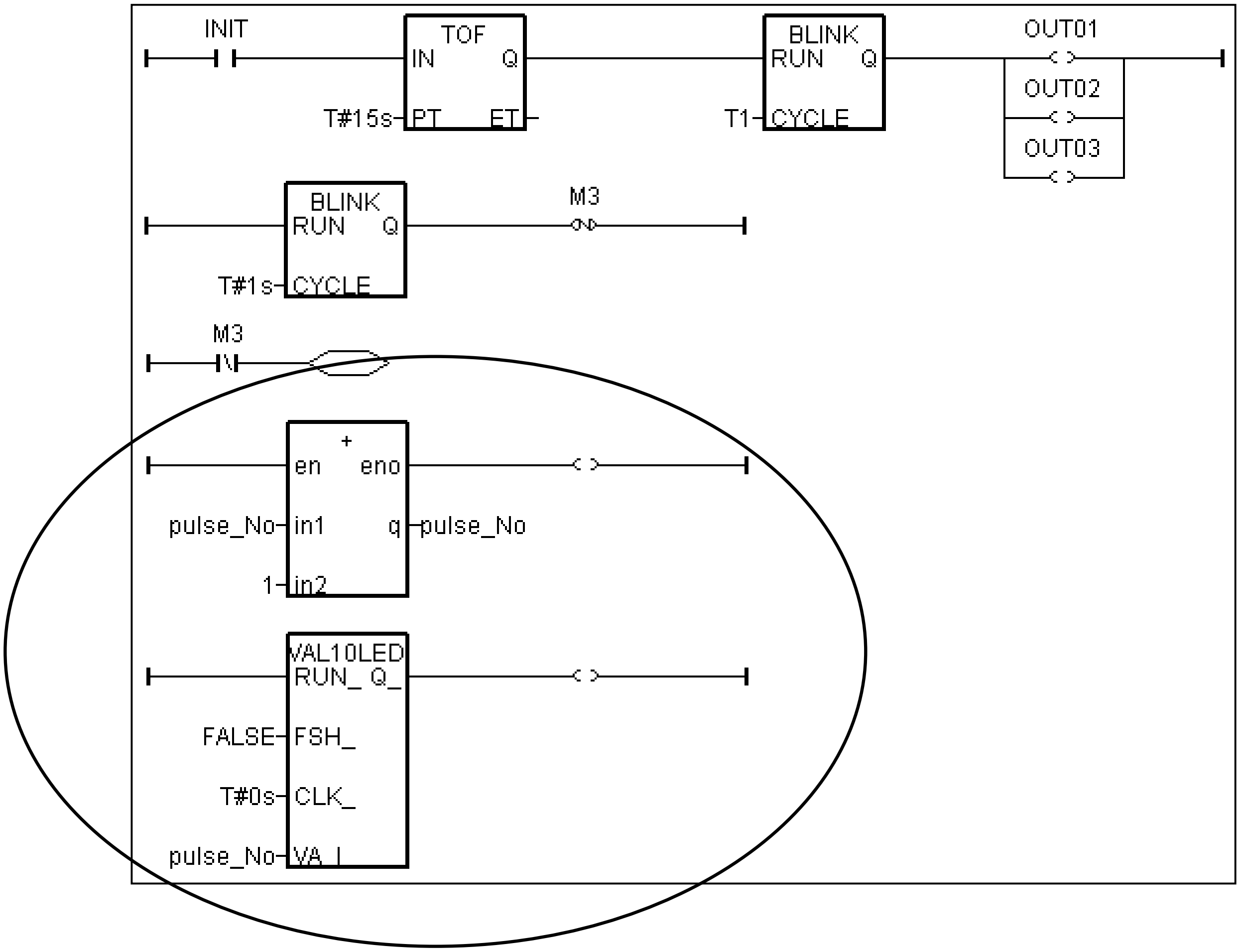
Move the cursor to “return” and then click on “F2 (Contact on the left)” to add a contact on the left.



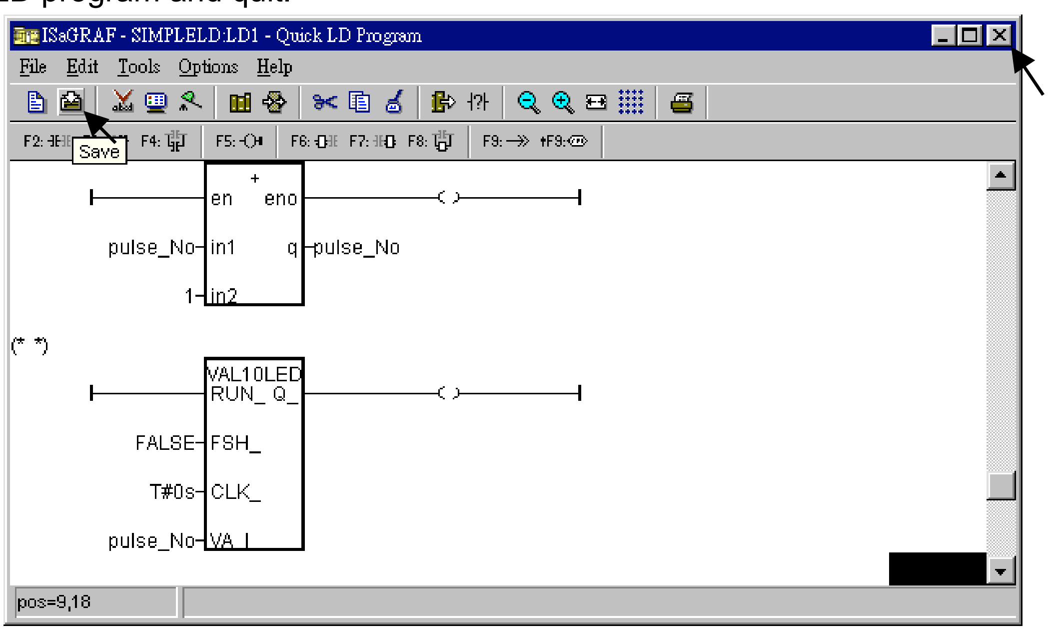
Then double click on the contact to assign “M3” to it. And change its type to “\” (inverted contact).



The procedure to create the forth & the last LD rung is similar as former steps. Please do it by yourself. The final LD program should look like the below.



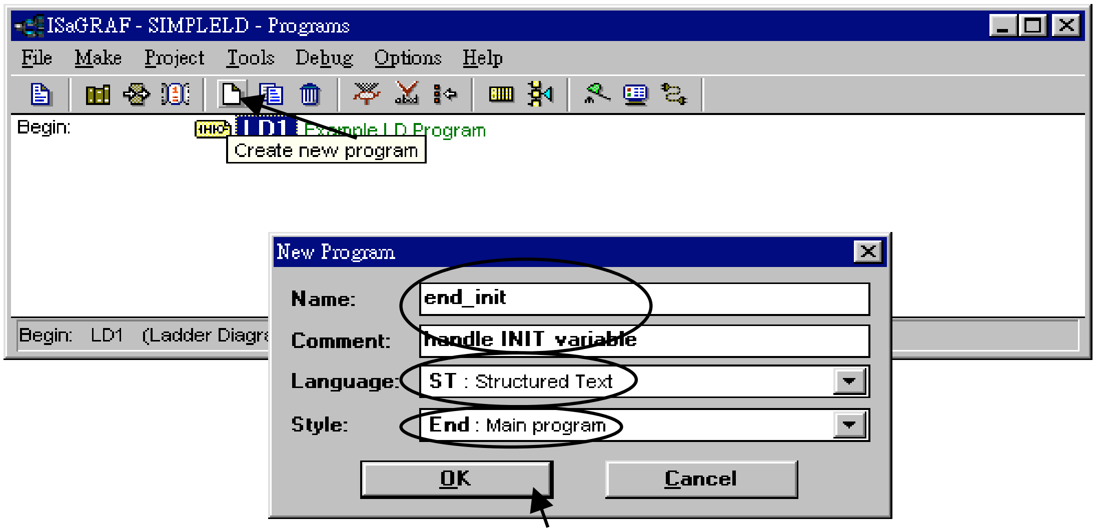
Save this LD program and quit.



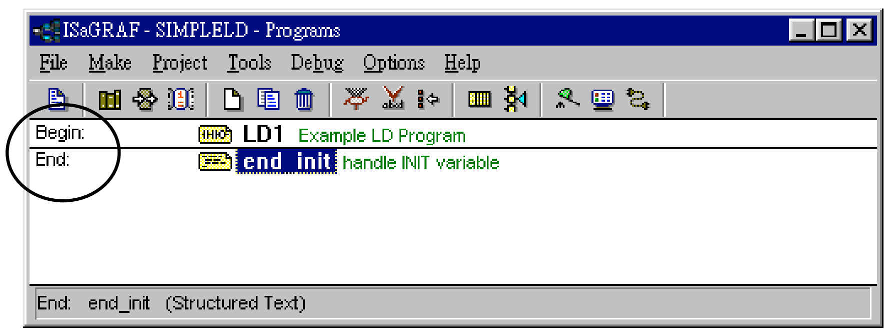
2.1.1.6: Create The ST "end_init" Program

In this project we need an extra ST program to handle the “INIT” variable.

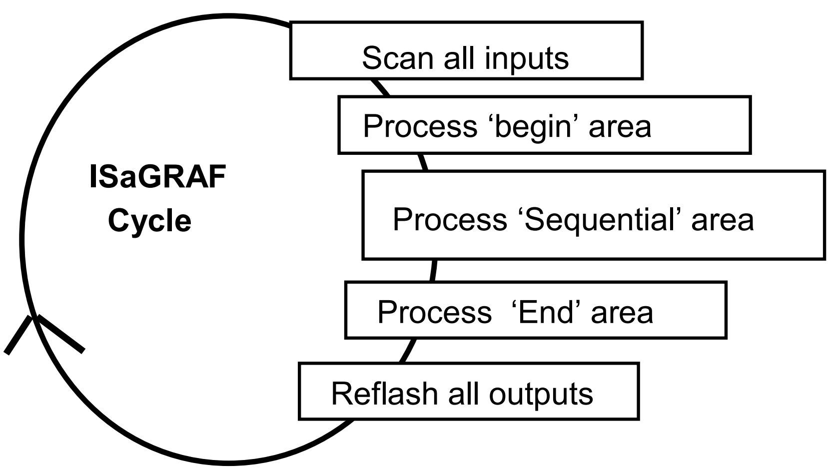
Click on “Create new program” in the “ … - Programs” window to add a ST program. Given the Name as “end_init” , Comment as “Handle INIT variable” , Language as “ST: Structured Text” , & Style as “End: Main program”. Then click on “OK”.



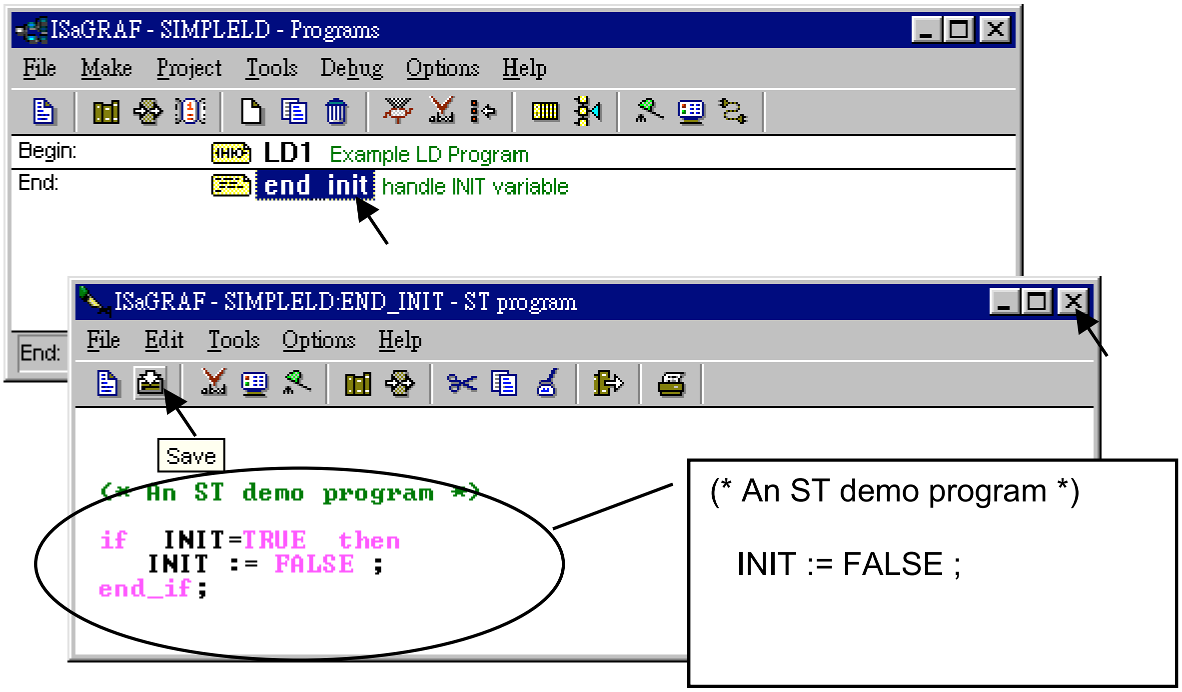
Now we have two programs inside this project.



ISaGRAF will run these two programs one time in each PLC scan cycle. Programs in the “begin” area will run first, then the “Sequential” area, and last the “End” area. An ISaGRAF cycle run in the way as the below scheme.



Double click on “end_init” program to edit it. Click on “save” and then exit when you finish it. (Any character inside between “(*” and “*)” is the comment.)



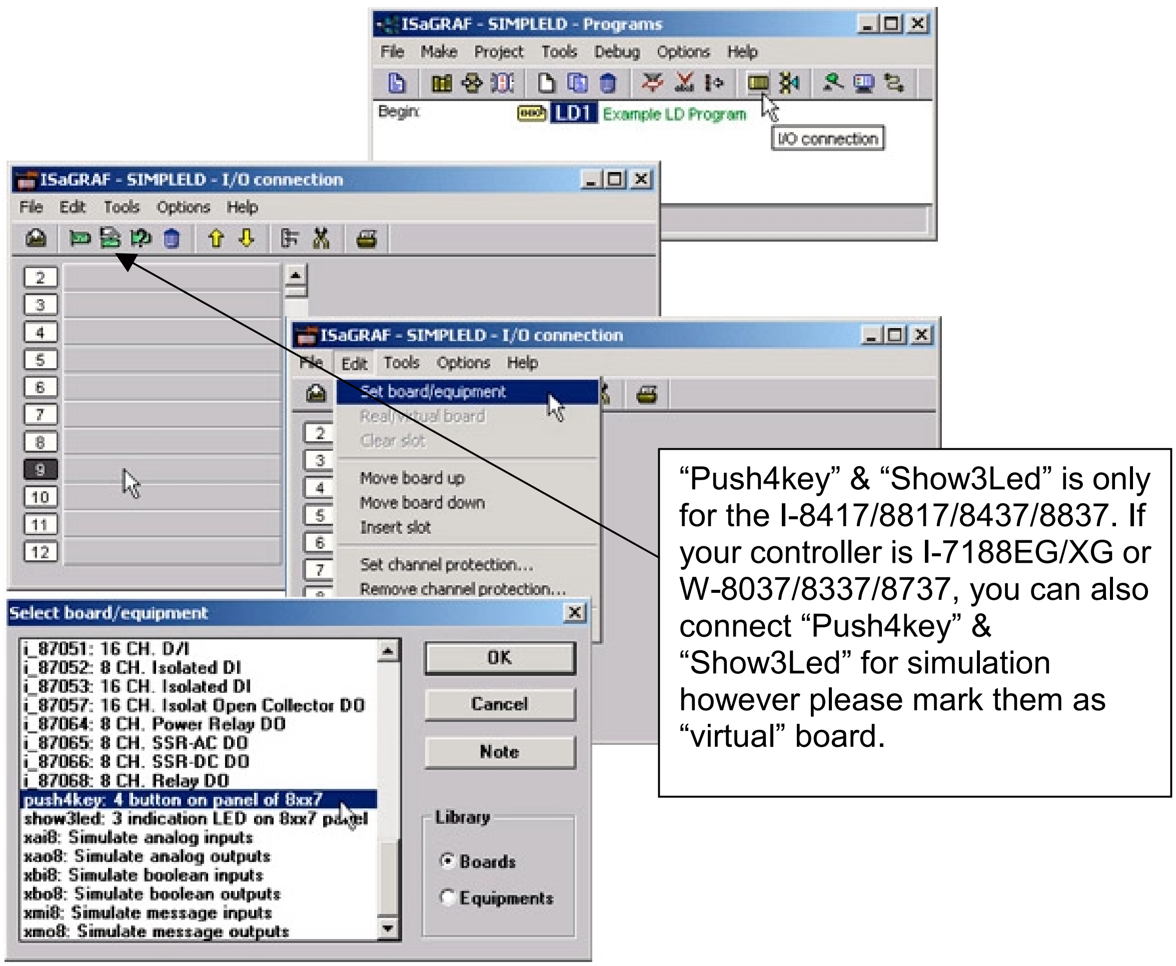
Since “INIT” is declared with an initial value “TRUE”, this ST program will let “INIT” set to “FALSE” at the end of the first scan cycle. In other word, “INIT” will indicate this project is running in the first scan cycle or not (TRUE: first scan cycle, FALSE: other cycles).

Now we have finished the programming, now we are going to the next step – “Connect the I/O”.

2.1.2: Connecting The I/O

The ISaGRAF Workbench software program is an open programming system. This allows the user to create an ISaGRAF program that can operate a large number of different PLC controller systems. It is the responsibility of the PLC hardware manufacturer to embed the ISaGRAF "driver" in their respective controller for the ISaGRAF program to operate properly. The ICP DAS line of I-8417/8817/8437/8837, I-7188EG, I-7188XG & W-8037/8337/8737 series of controllers have the ISaGRAF driver embedded, creating a powerful and flexible industrial controller system.

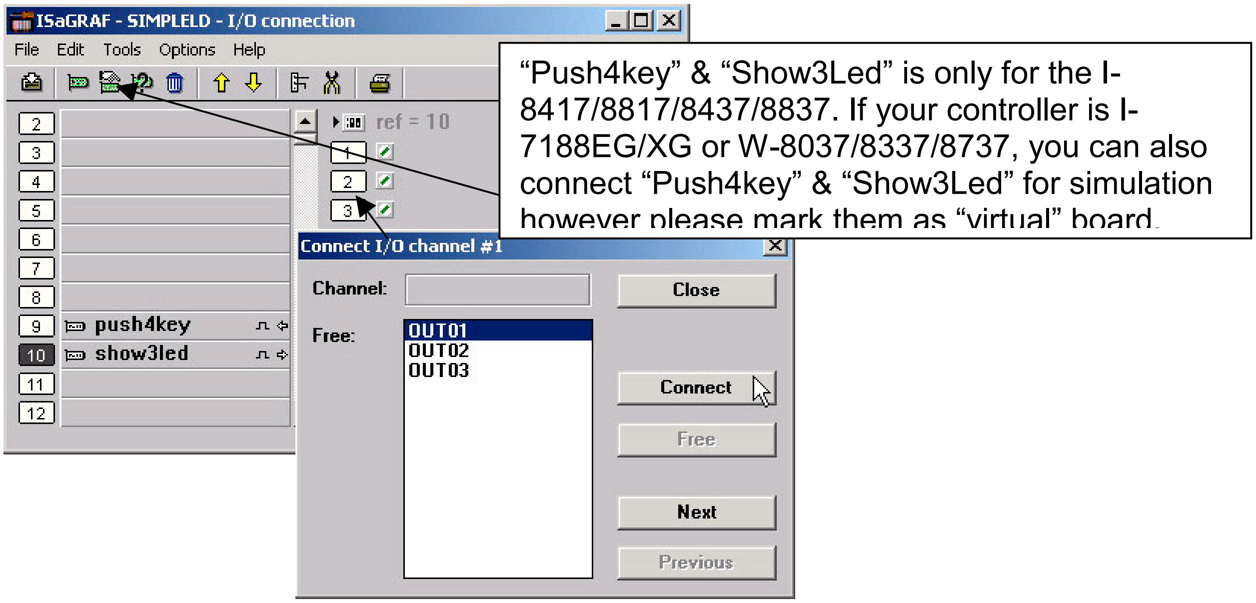
Now that you have created the ISaGRAF example program, now you must connect the I/O to the controller system. A useful feature of the I-8xx7 controller system is that you can run program we have created WITHOUT having any I/O boards plugged into the I-8xx7 controller system. The four pushbuttons on the I-8xx7 controller system can be used as four digital inputs, and the three left LED's above the control panel pushbuttons can be used as outputs.



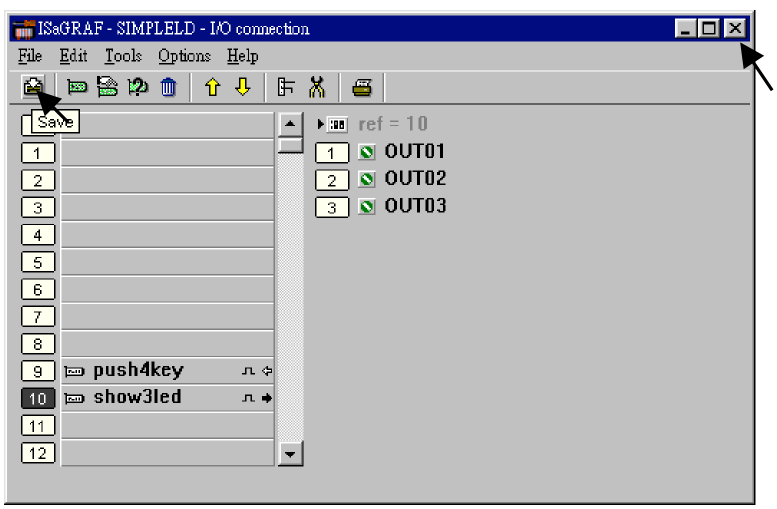
Click on the "I/O Connection" icon as shown in the top picture and the "I/O Connection" window will appear as shown in the next illustration. For the purpose of this example, you can either double click on the "9" slot, or just click on the "9" slot, then click on "Edit" and then "Set Board/Equipment" and then the "I/O Connection" window will appear. This now associates the four control panel pushbuttons - "push4key" as four digital inputs. (We don't use it in this example program since there is no boolean variable declared with "Input" attribution).

IMPORTANT NOTICE: I/O Slots 0 through 7 are reserved for REAL I/O boards that will be used in the I-8xx7 controller (W-8337/8737 doesn't have slot 0). You can use slots 8 and above for additional functionality as illustrated by the example program.

To create the I/O connections for the outputs, double click on the "10" slot, then click on the "Show3led: 3 indication LED on 8xx7 panel" selection. This will now associate the three LED's above the four control panel pushbuttons as the three outputs for the example program. Your "I/O Connection" window should now look like the screen below.



Remember to click on the "SAVE" icon to save the I/O connections that have been created for the example program. And click on the "X" to exit the window.

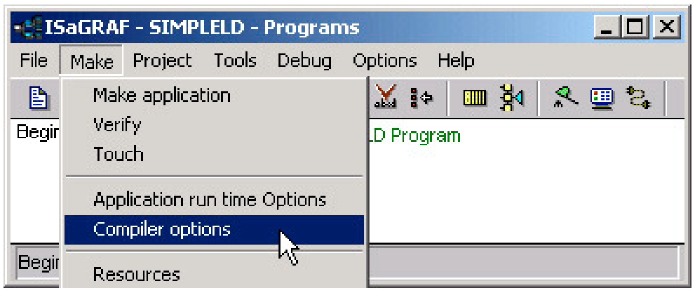


IMPORTANT NOTE: All of the variables with Input and Output attribute MUST be connected through the I/O connection as described above for any program to be successfully compiled. Only the Input and Output attributed variables will appear in the "I/O Connections" window. In this example we have only 3 boolean output variables, they are OUT01, OUT02 & OUT03.

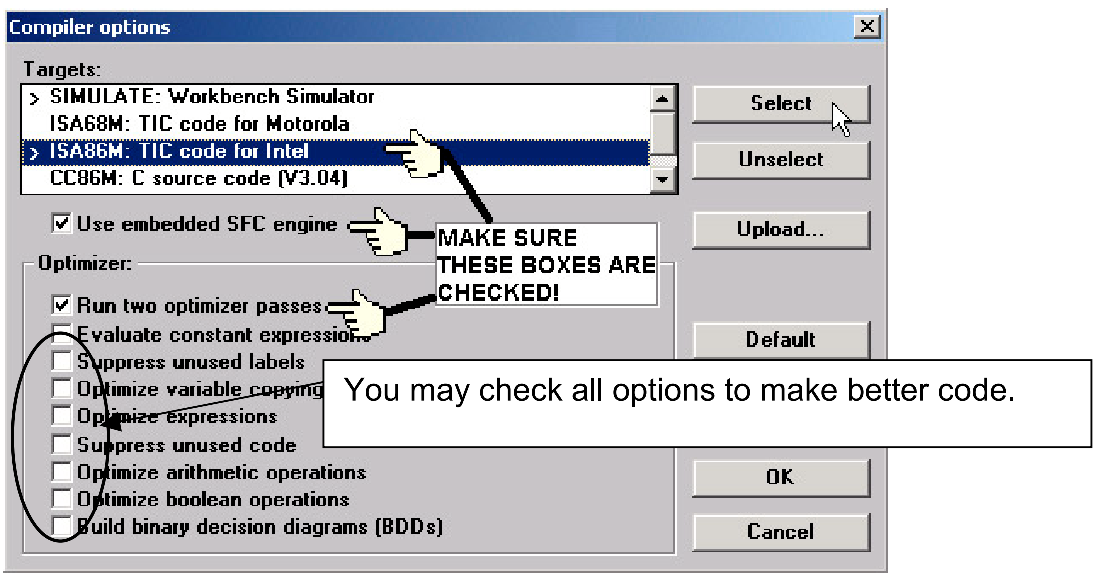
2.1.3: Compiling The Example LD Project

For ANY AND EVERY ISaGRAF program to work properly with any of the I-8xx7, I-7188EG, 7188XG & W-8xx7 controller systems, it is the responsibility of the programmer to properly select the correct "Compiler Options". You MUST select the "ISA86M: TIC Code For Intel" option as described below.

To begin the compilation process, first click on the "MAKE" option from the main menu bar, and then click on "Compiler Options" as shown below.

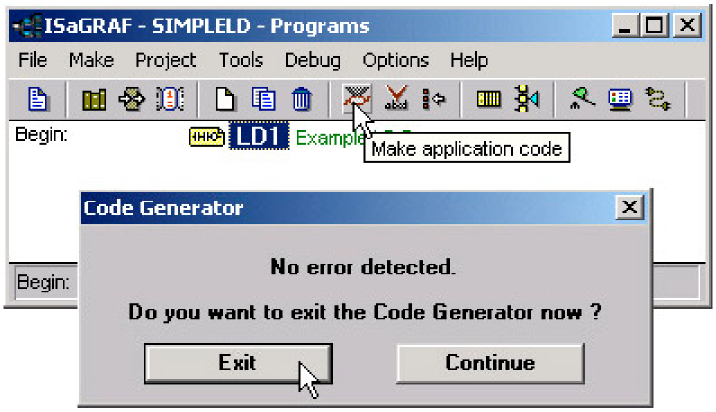


The "Compiler Options" window will now appear. Make sure to select the options as shown below then press the "OK" button to complete the compiler option selections.



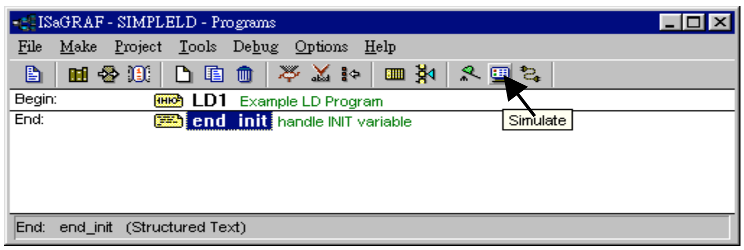
TIME TO COMPILE THE PROJECT!
Now that you have selected the proper compiler options, click on the "Make Application Code" icon to compile the example LD project. If there are no compiler errors detected during the compilation process, CONGRATULATIONS, you have successfully created our example LD program.

If errors are detected during the compilation process, just click on the "CONTINUE" button to review the error messages. Return to the Project Editor and correct the errors as outlined in the error message window.



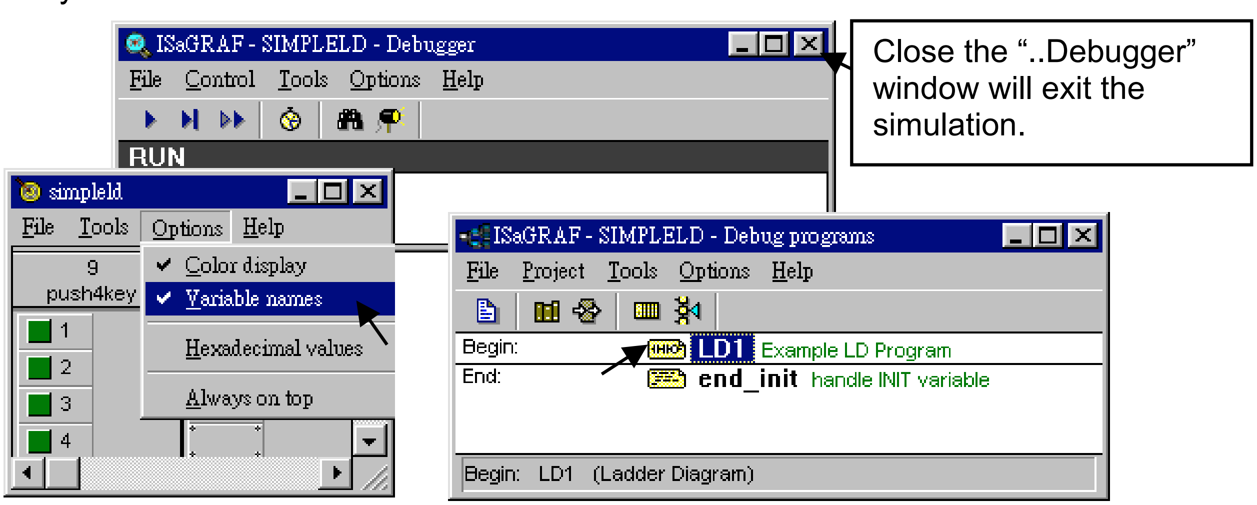
2.1.4: Simulating The LD Project

A powerful program-debugging feature of the ISaGRAF software program is the ability to "SIMULATE" the program you have developed before loading it into the ISaGRAF controller system. After successfully compiling the example LD program, click on the "SIMULATE" icon as shown below.



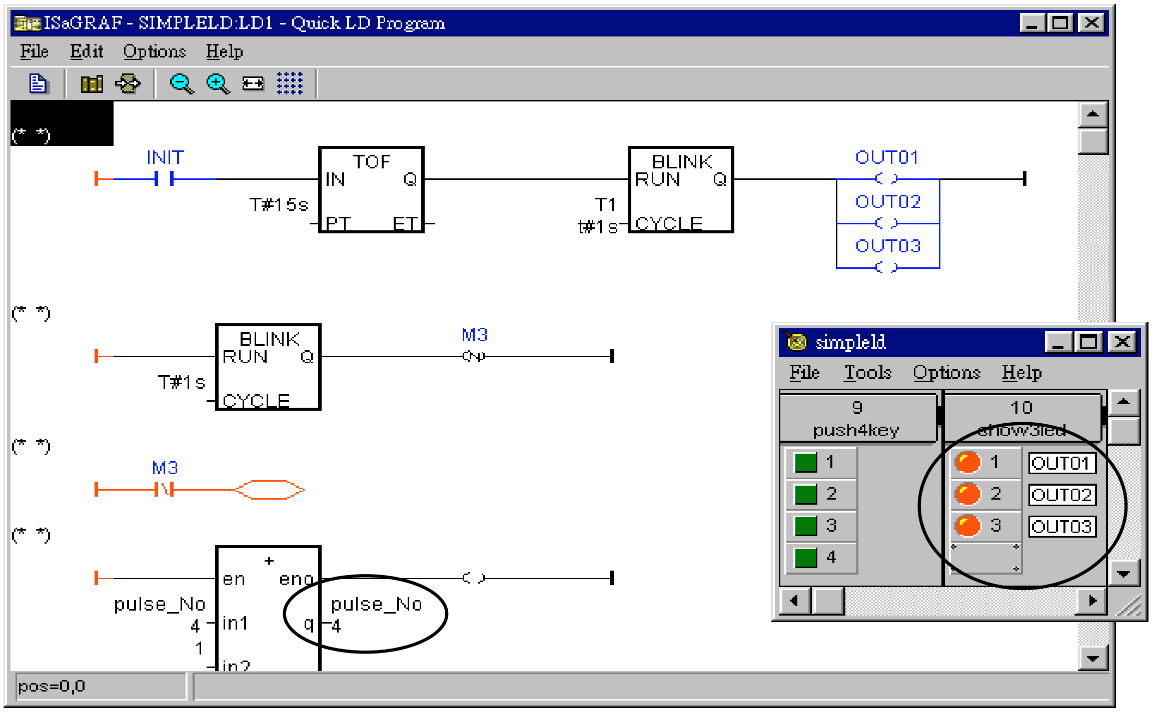
When you click on the "Simulate" icon three windows will appear. The windows are the "ISaGRAF Debugger", the "ISaGRAF Debug Programs", and the "I/O Simulator" windows. If the I/O variable names you have created DO NOT appear in the I/O simulator window, just click on the "Options" and "Variable Names" selection and the variable names you have created will now appear next to each of the I/O's in the simulator window.

In the "ISaGRAF Debug Program" window, double click on the "LD1" where the cursor below is positioned. This will open up the ISaGRAF Quick LD Program window and you can see the LD program you have created.



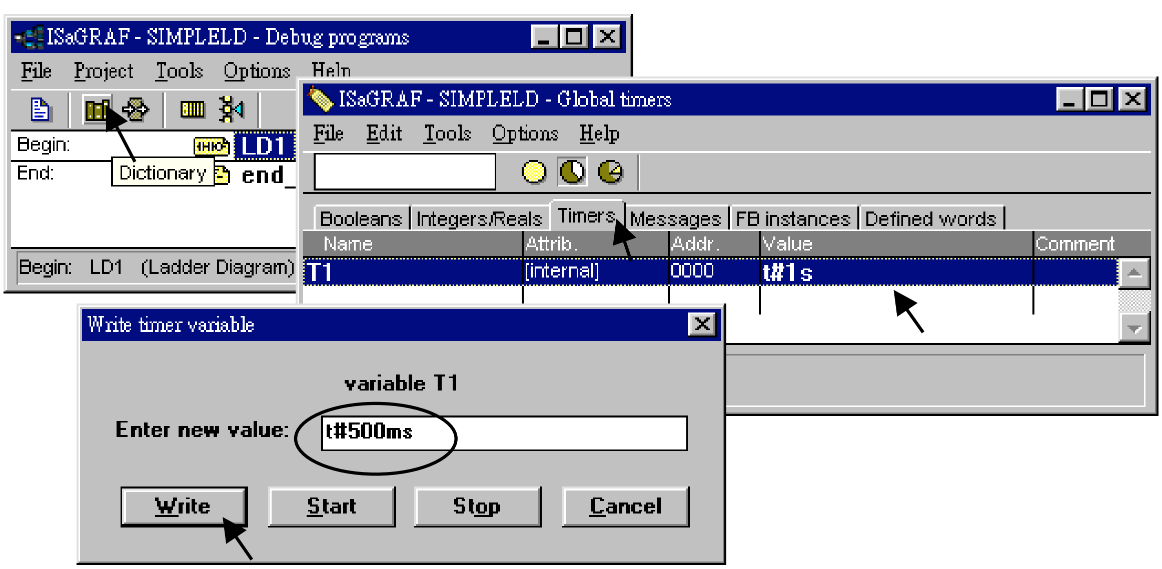
Running The Simulation Program

When you double click on "LD1" in the "ISaGRAF Debug Programs" window, the follow window should appear.



You can see outputs “OUT01” thru. “OUT03” will blink in the first 15 seconds. And the “pulse_No” continuously plus one every second.

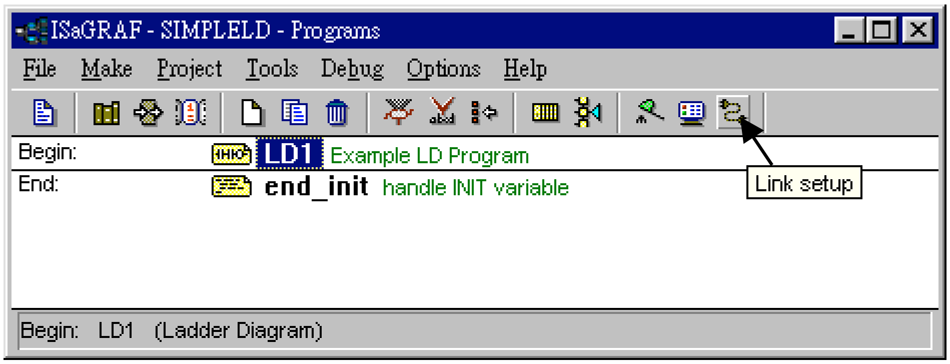
You can adjust the "T1" variable while the program is running. To accomplish this, click on the "Dictionary" icon which will open the "ISaGRAF Global Variables" window as shown in the first two pictures below. Click on “Timer” tab and then double click on “T1” to change the timer value to “T#500ms” (this means 0.5 second). Then click on “Write”.



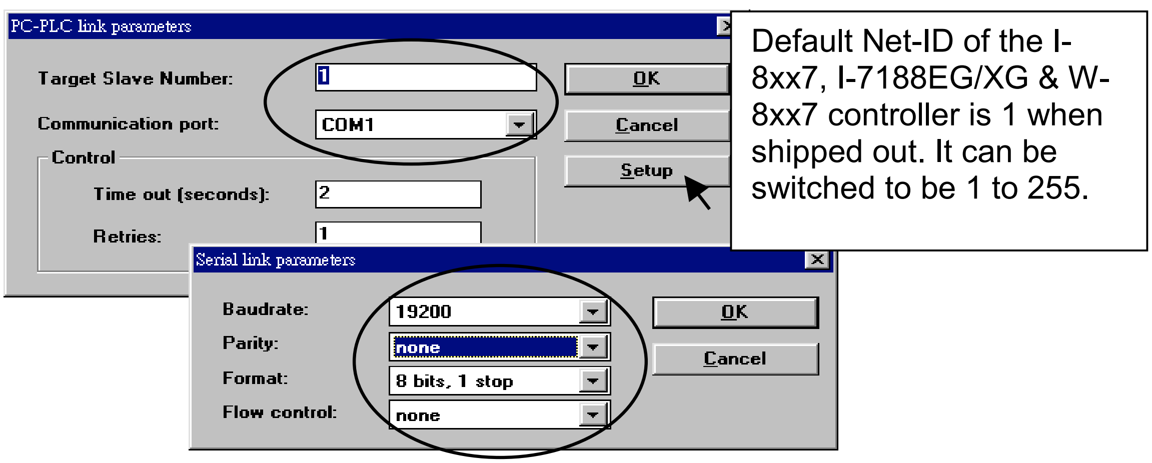
2.1.5: Download & Debugging The Example LD Project

The last step required to running the example LD program on the ISaGRAF controller system is to download the project to the controller (frequently referred to as the "Target" platform"). Before this download can be accomplished you must first establish communications between your development PC and the controller.

To begin this process, click on the "Link Setup" icon in the "ISaGRAF Programs" window. When you click on the "Link Setup" icon, the following window will appear.



The "Target Slave Number" is the Node-ID address for the I-8xx7 controller as defined by the dipswitch settings outlined in Chapter 1, Section 1.3.1. The Node-ID dipswitch is located in the bottom right portion of the I-8xx7 controller. If your I-8xx7 controller is the first one, the Node-ID address should be set to "1". The "Communication Port" is the serial port connection on your development PC, and this is normally either COM1 or COM2.

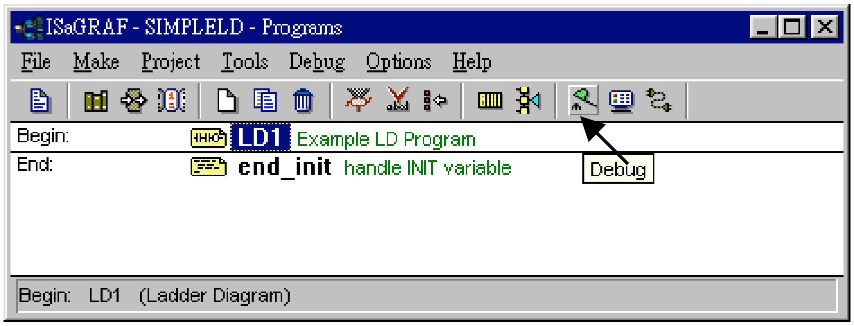


The communication parameters for the target I-8xx7 controller MUST be set to the same serial communication parameters for the development PC. For I-8417 and I-8817 controllers (serial port communications), the default parameters for COM1 (RS232) and COM2 (RS485) ports are:

Baudrate:	19200
Parity:	none
Format:	8 bits, 1 stop
Flow control:	none

IMPORTANT NOTE
It may be necessary to change the COM port settings for the development PC. Depending on which computer operating system you are using, you will need to make sure that the COM port can properly communicate to the I-8xx7, I-7188EG/XG & W-8xx7 controller system.

DOWNLOADING THE EXAMPLE PROJECT
Before you can download the project to the controller, you must first verify that your development PC and the controller are communicating with each other. To verify proper communication, click on the "Debug" icon in the "ISaGRAF Programs" window as shown below.

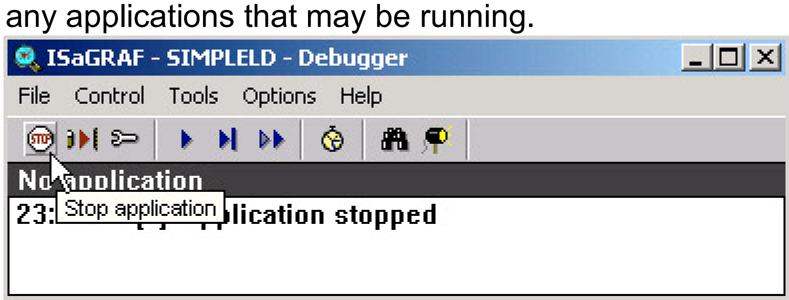


If the development PC and the I-8xx7, I-7188EG/XG & W-8xx7 controller system are communicating properly with each other, the following window displayed below will appear (or if a program is already loaded in the controller system, the name of the project will be displayed with the word "Active" following it.
If the message in the "ISaGRAF Debugger" says "Disconnected", it means that the development PC and the controller system have not established communications with each other.

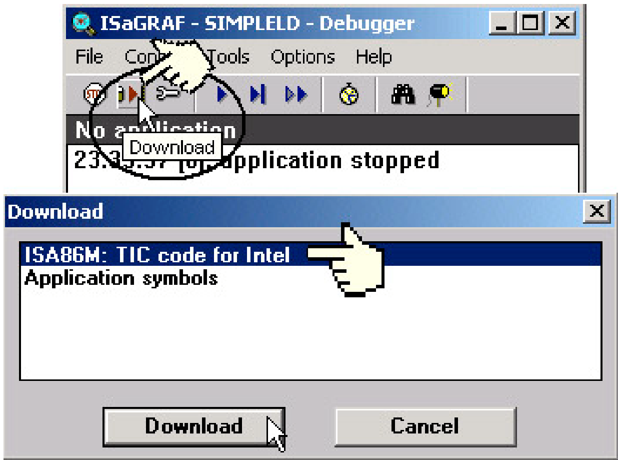
The most common causes for this problem is either the serial port cable not being properly configured, or the development PC's serial port communications DO NOT match that of the controller system.

You may have to either change the serial port communication settings for the development PC (which may require changing a BIOS setting) or change the "Serial Link Parameters" in the ISaGRAF program.

If there is a project already loaded in the controller system you will need to stop that project before you can download the example project. Click on the "STOP" icon as illustrated above to halt any applications that may be running.

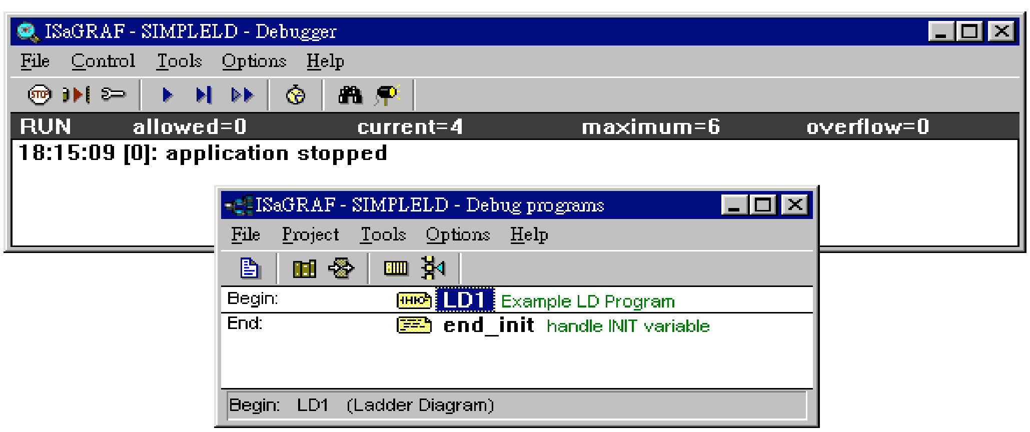


STARTING THE DOWNLOADING PROCESS
From the "ISaGRAF Debugger" window click on the "Download" icon, then click on "ISA86M: TIC Code For Intel" from the "Download" window as shown below.



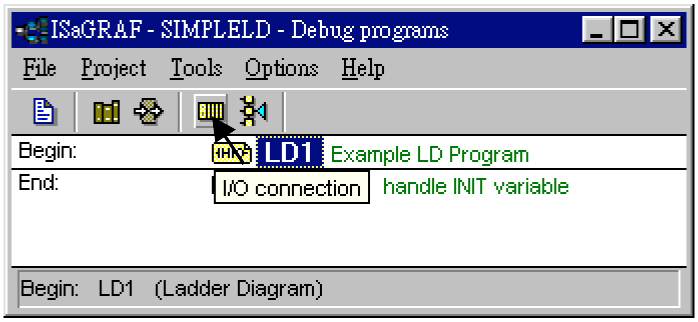
The example project will now start downloading to the I-8xx7, I-7188EG/XG & W-8xx7 controller system. A progress bar will appear in the "ISaGRAF Debugger" window showing the project downloading progress.

When the example project has successfully completed the downloading process to the controller system the following two windows will appear.

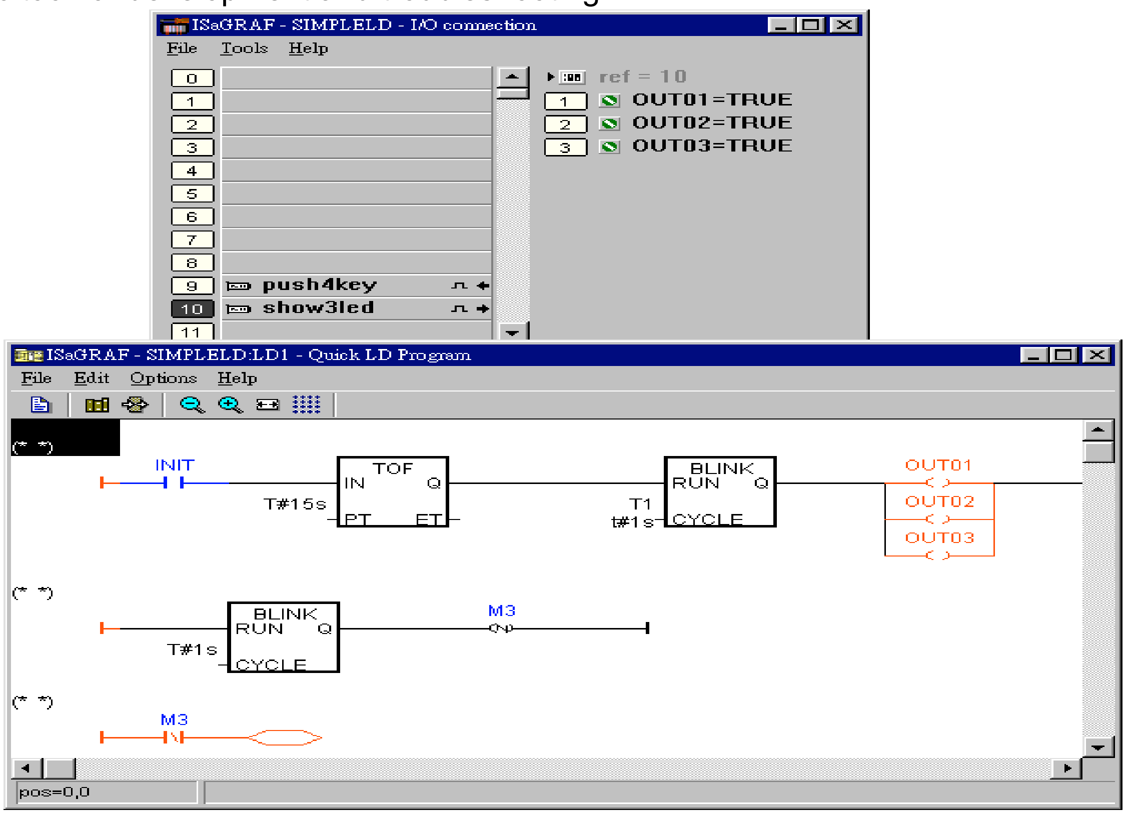


RUNNING THE EXAMPLE LD PROGRAM
You can observe the real time I/O status from several ISaGRAF windows while you are running the example project. One of the windows is the "I/O Connections" window, which shows each of the inputs and outputs as assigned. Click on the "I/O Connections" icon in the ISaGRAF

Debugger window to open the "I/O Connections" screen. Another VERY helpful window you can open is the "Quick LD Program" window. From this window you can observe the LD program being executed in real time.



In the window below, the OUT01 thru. OUT03 is blinking in the first 15 seconds. The "Quick LD Program" window shows the entire ladder logic program in REAL TIME and is an excellent diagnostic tool for development and troubleshooting.



Though there are numerous steps involved in creating and downloading an ISaGRAF program, each step is quick and easy to accomplish, and the end result is a powerful and flexible control development environment for the ISaGRAF controller systems.

PRACTICE, PRACTICE, PRACTICE!

Now that you have successfully created and ran your first ISaGRAF program with the I-8xx7, I-7188EG/XG & Wincon-8xx7 controller system, you should practice creating more elaborate and powerful programs. Like any other computer development environment, practice and experimentation is the key to understanding and success, GOOD LUCK!

2.2: A Simple Structured Text (ST) Program

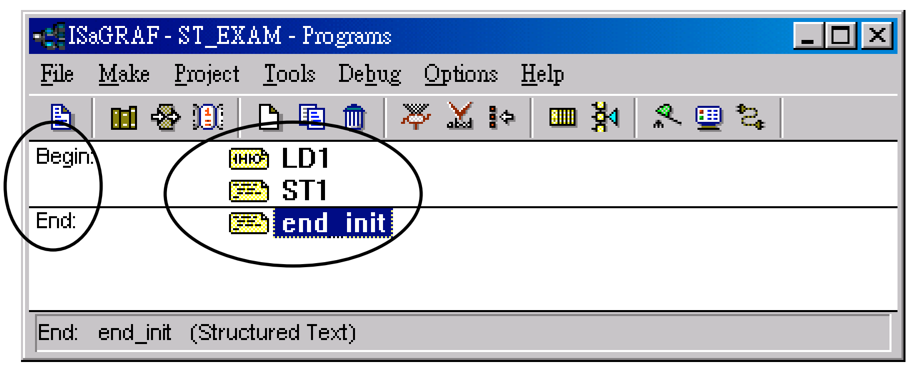
A "Structured Text" program is a high-level program language that is designed for automation process control applications. The "Structured Text (henceforth referred to as "ST") is primarily used to implement complex procedures that cannot be easily expressed by a graphical language such as LD or FBD.

An ST program is comprised by a list of "ST Statements", and each "ST Statement" MUST end with a semi-colon “;”. All characters inside between “(*” and “*)” is comment.

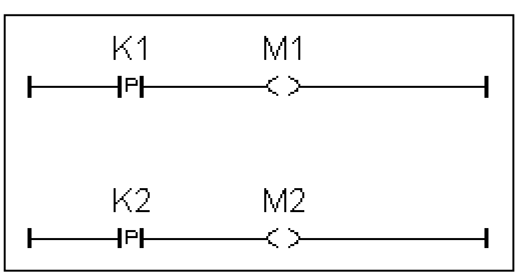
Variables Used In The Example ST Project:

Name	Type	Attribute	Description
INIT	Boolean	**Internal**	**initial value at “TRUE”**. TRUE means 1st scan cycle
K1	Boolean	**Input**	The first pushbutton on the front panel of the I-8xx7
K2	Boolean	**Input**	The second pushbutton on the front panel of the I-8xx7
M1	Boolean	**Internal**	Indicate pushbutton K1 is just pushed.
M2	Boolean	**Internal**	Indicate pushbutton K2 is just pushed.
TEMP	Boolean	**Internal**	A boolean variable for temporary use
COUNT	Integer	**Internal**	A integer value generated by push K1 & K2 **initial value is set at "0"**

Three programs are used in this example. One is LD program named “LD1”, The other two are ST programs named respectively as “ST1” & “end_init”.



LD program “LD1” Outline:



ST program “ST1” Outline:

```
(* Open Com3 with 9600 baud rate, 8 char. size, no parity, 1 stop bit at first scan cycle *)
if  INIT=TRUE  then
  TEMP := comopen(3, 9600, 8, 0, 1) ;
end_if ;

(* Do something when K1 or K2 is pushed *)
if  (M1=TRUE) or (M2=TRUE)  then

  (* COUNT plus 1 when K1 is pushed *)
  if  M1=TRUE  then
    COUNT := COUNT+1 ;
  end_if ;

  (* COUNT plus 10 when K2 is pushed *)
  if  M2=TRUE  then
    COUNT := COUNT+10 ;
  end_if ;

  (* save COUNT value to the 5th Pos. of No.2 integer arry *)
  TEMP := ARY_N_W(2, 5, COUNT) ;

  (* write one byte = 2 (hex.) to Com3 *)
  TEMP := COMWRITE(3, 16#2) ;

  (* write 1 integer (1 long integer contains 4 bytes) of Pos. 5 inside No.2 array to Com3 *)
  TEMP := COMAY_NW(3, 2, 1, 5) ;

  (* write one byte = 3 (hex.) to Com3 *)
  TEMP := COMWRITE(3, 16#3) ;

end_if ;
```

ST program “end_init” Outline:

```
If  INIT=TRUE  then
   INIT  :=  FALSE  ;
End_if ;
```

Process Operation Actions:

LD Program “LD1” :
Catch the rising edge status when pushbutton K1 is just pushed and save it into a internal boolean variable “M1”
Catch the rising edge status when pushbutton K2 is just pushed and save it into a internal boolean variable “M2”

ST Program “ST1” :
Open Com3 of the I-8xx7 controller with 9600 baud rate, 8 char. size, no parity, 1 stop bit at the first scan cycle.
Plus “COUNT” value by 1 every time when pushbutton K1 is pushed.
Plus “COUNT” value by 10 every time when pushbutton K2 is pushed.
Send “Count” value to a PC via Com3 of the I-8xx7 controller in the below frame format.

STX	Value of COUNT	ETX

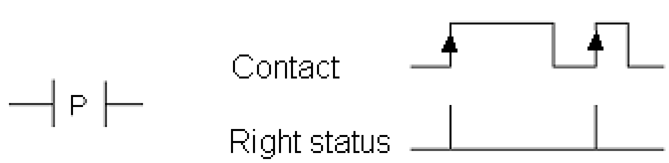
Lowest byte	2nd lower byte	3rd lower byte	Highest byte

STX : Start of frame, byte value = 2
ETX : End of frame, byte value = 3

ST Program “end_init” :
Set boolean variable “INIT” to FALSE at the end of the PLC scan cycle. So that “INIT” will be TRUE only at the first scan cycle.

Function description:

“P” contact : Contact with P type means the right status will be set to a pulse TRUE when the contact is just rising from FALSE to TRUE.



Comopen(PORT, BAUD, CHAR, PARI, STOP) : To open a Com port of the I-8xx7 controller

Parameter

PORT :	Integer	3:COM3 ,4:COM4, ..., 20:COM20
BAUD :	Integer	baud rate, 2400, 4800, 9600, 19200, 38400, 57600, 115200
CHAR :	Integer	char. size, 7 or 8
PARI :	Integer	parity, 0:none, 1:even, 2:odd
STOP :	Integer	stop bit, 1 or 2

Return : boolean ok.: TRUE , fail: FALSE

Ary_N_W(NUM, ADR, DATA) : Save one long integer into an integer array.

Parameter

NUM :	Integer	save to which array (1-6)
ADR :	Integer	save to which Pos. in this array (1-256)
DATA :	Integer	the integer value to save

Return : boolean ok.: TRUE , fail: FALSE

ComWrite(PORT, DATA) : Write one byte to a Com port

Parameter

PORT :	Integer	3:COM3 ,4:COM4, ..., 20:COM20
DATA :	Integer	the byte value (0 - 255) to write

Return : boolean ok.: TRUE , fail: FALSE

ComAy_NW(PORT, ARY_NO, NUM, POS) : Write an integer array to a Com port

Parameter

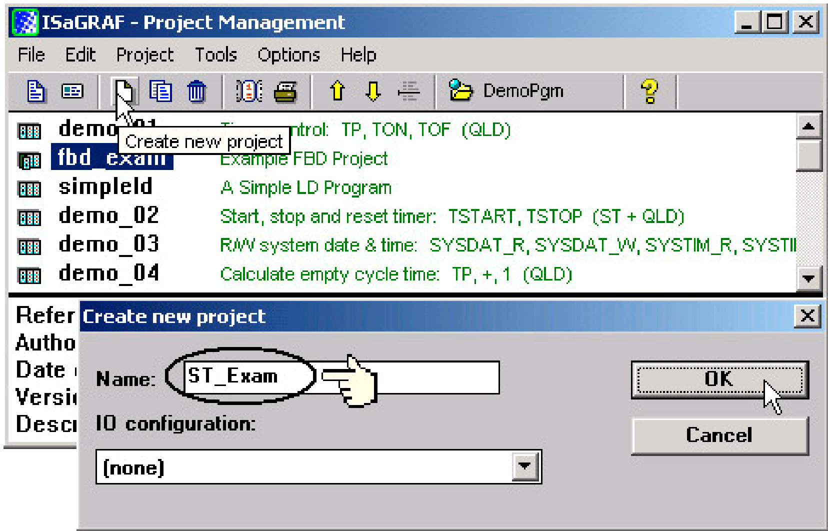
PORT :	Integer	3:COM3 ,4:COM4, ..., 20:COM20
ARY_NO :	Integer	the array No. to write (1-6)
NUM :	Integer	number of integers to write (0-256)
POS :	Integer	start position inside the array to write (1-256)

Return : boolean ok.: TRUE , fail: FALSE

2.2.1: Example ST Program

The first step is to create a new project for the example ST program.

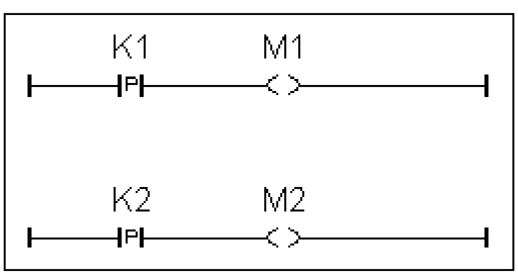
Creating The Example ST Project
From the "ISaGRAF Project Management" window click on the "Create New Project" icon and enter "ST_Exam" for the name for the example ST project.



Declaring The Example ST Variables as below content
Refer to Section 2.1.1.3. "Declaring The Variables" for assistance.

Name	Type	Attribute	Description
INIT	Boolean	**Internal**	**initial value at "TRUE"**. TRUE means 1st scan cycle
K1	Boolean	**Input**	The first pushbutton on the front panel of the I-8xx7
K2	Boolean	**Input**	The second pushbutton on the front panel of the I-8xx7
M1	Boolean	**Internal**	Indicate pushbutton K1 is just pushed.
M2	Boolean	**Internal**	Indicate pushbutton K2 is just pushed.
TEMP	Boolean	**Internal**	A boolean variable for temporary use.
COUNT	Integer	**Internal**	A integer value generated by push K1 & K2 **initial value is set at "0"**

Creating a LD program "LD1" with the below content.
Refer to Section 2.1.1.4. and 2.1.1.5 for assistance.



Follow the same steps as 2.1.1.6. to create a ST program “end_init” with the below content.

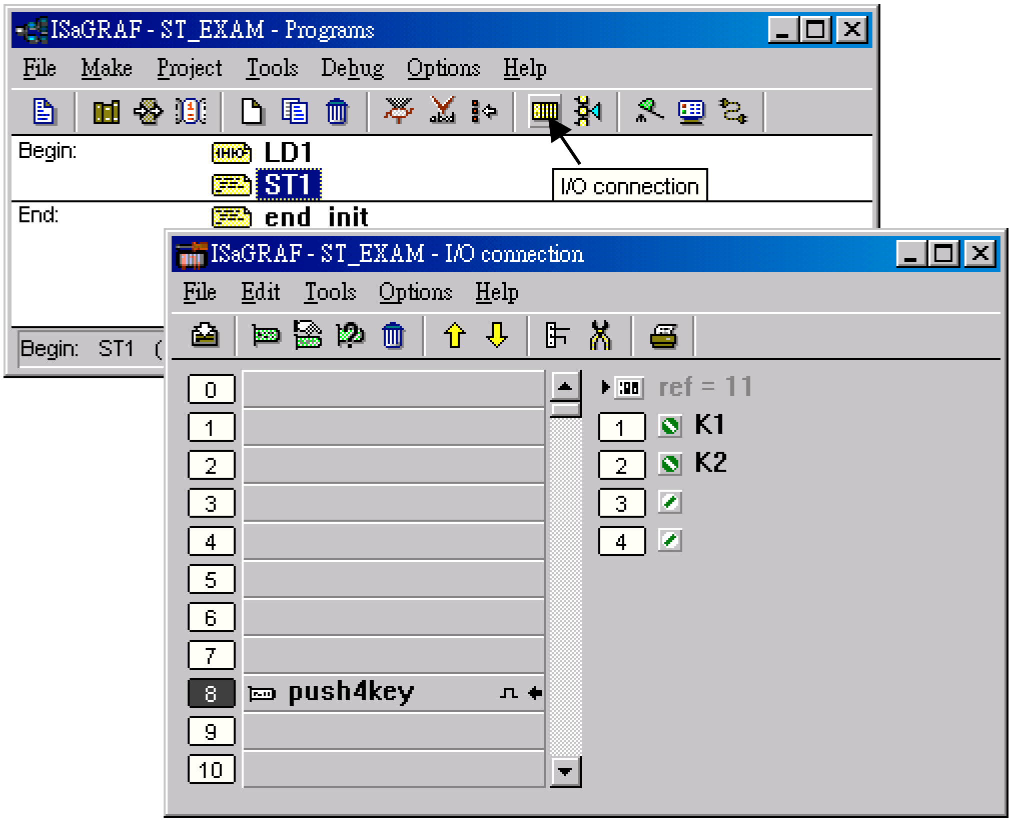
```
If  INIT=TRUE  then
   INIT  :=  FALSE  ;
End_if ;
```

Creating a ST program “ST1” with the below content.
Refer to Section 2.1.1.6. for assistance.

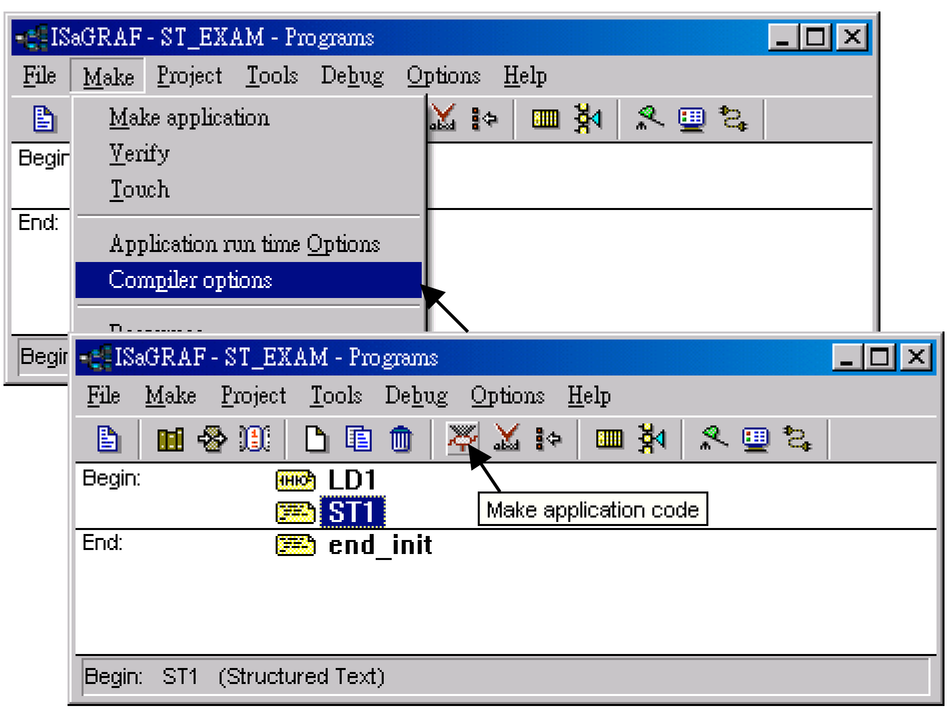
```
(* Open Com3 with 9600 baud rate, 8 char. size, no parity, 1 stop bit at first scan cycle *)
if  INIT=TRUE  then
  TEMP := comopen(3, 9600, 8, 0, 1) ;
end_if ;

(* Do something when K1 or K2 is pushed *)
if  (M1=TRUE) or (M2=TRUE)  then

  (* COUNT plus 1 when K1 is pushed *)
  if  M1=TRUE  then
    COUNT := COUNT+1 ;
  end_if ;

  (* COUNT plus 10 when K2 is pushed *)
  if  M2=TRUE  then
    COUNT := COUNT+10 ;
  end_if ;

  (* save COUNT value to the 5th Pos. of No.2 integer arry *)
  TEMP := ARY_N_W(2, 5, COUNT) ;

  (* write one byte = 2 (hex.) to Com3 *)
  TEMP := COMWRITE(3, 16#2) ;

  (* write 1 integer (1 long integer contains 4 bytes) of Pos. 5 inside No.2 array to Com3 *)
  TEMP := COMAY_NW(3, 2, 1, 5) ;

  (* write one byte = 3 (hex.) to Com3 *)
  TEMP := COMWRITE(3, 16#3) ;

end_if ;
```

IMPORTANT NOTE
Each ST statement line MUST end with a semi-colon ";" as shown above. After entering in the above example program remember to click on the "Save" icon to save the program, then click on "Exit".

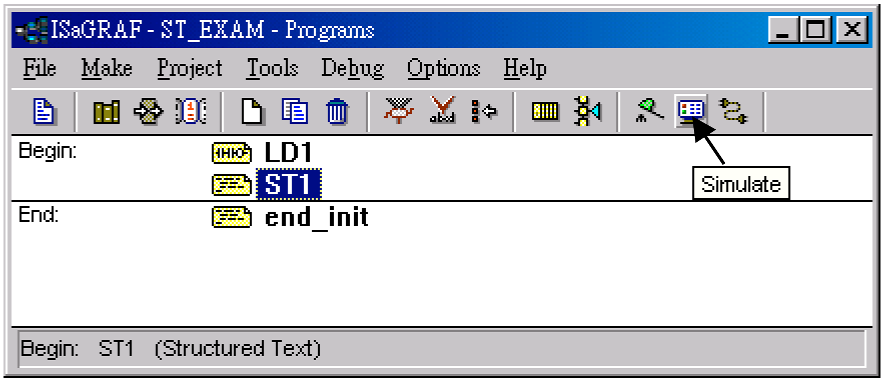
Use the similar procedure for the "Connecting I/O" as detailed in Section 2.1.2



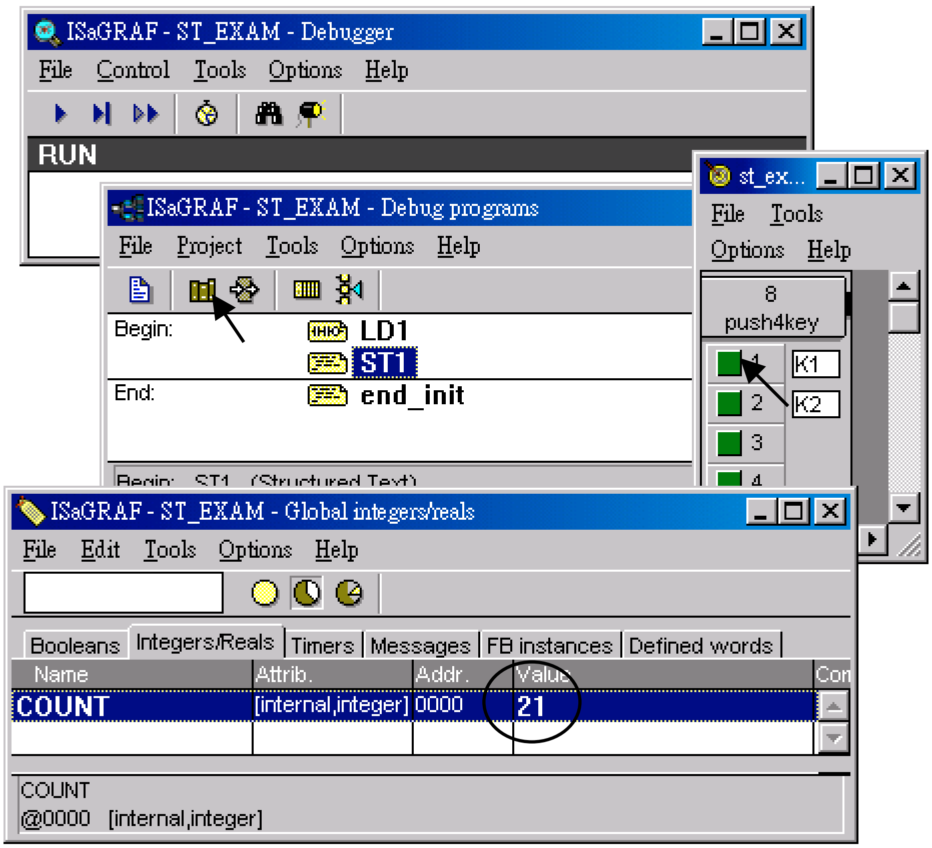
Use the similar procedure for the "Compilling the project" as detailed in Section 2.1.3



After compiling the example ST project click on the "Simulate" icon to observe the ST program running.



You may open the dictionary window to see the “COUNT” value. Click on “K1” or “K2”, you will see the “COUNT” value is changed.



You can now download this example project to the I-8xx7 controller system. Please follow the same procedure as outlined in Section 2.1.5 .

After downloading to the controller, the program will send 6 bytes via Com3 of the controller whenever K1 or K2 is pushed. If you have your RS232 monitoring program running on your PC, you can connect Com3 to your PC to see how it works.

2.3: A Simple Function Block Diagram (FDB) Program

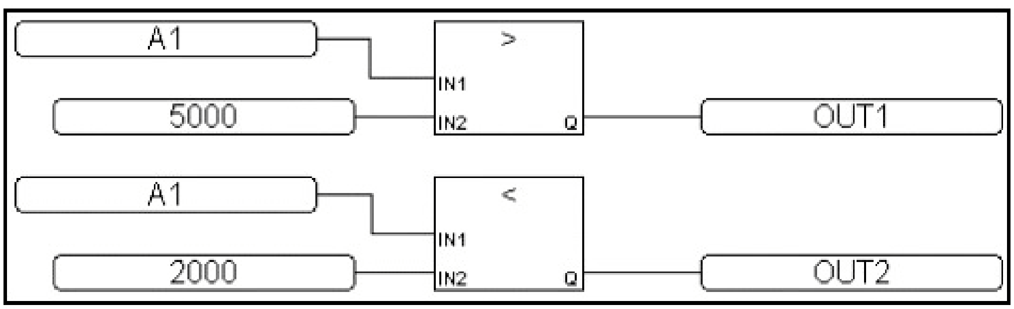
The "Function Block Diagram (FBD) is a graphical programming language that allows a programmer to build complex procedures by taking existing "Functions" from the ISaGRAF library and "Wiring" them together graphically to create powerful process control applications.

The following section details how to build a "Function Block Diagram" program with ISaGRAF. Function Block Diagram programs are extremely useful for managing several control process programs from a single source.

Example FBD Control Specification:
The following details the variables that will be used in our example Function Block Diagram program.

Name	Type	Attribute	Description
OUT1	Boolean	Output	High alarm
OUT2	Boolean	Output	Low alarm
A1	Integer	Internal	Simulate a temperature input, **initial value is 0**

FBD Program Outline:



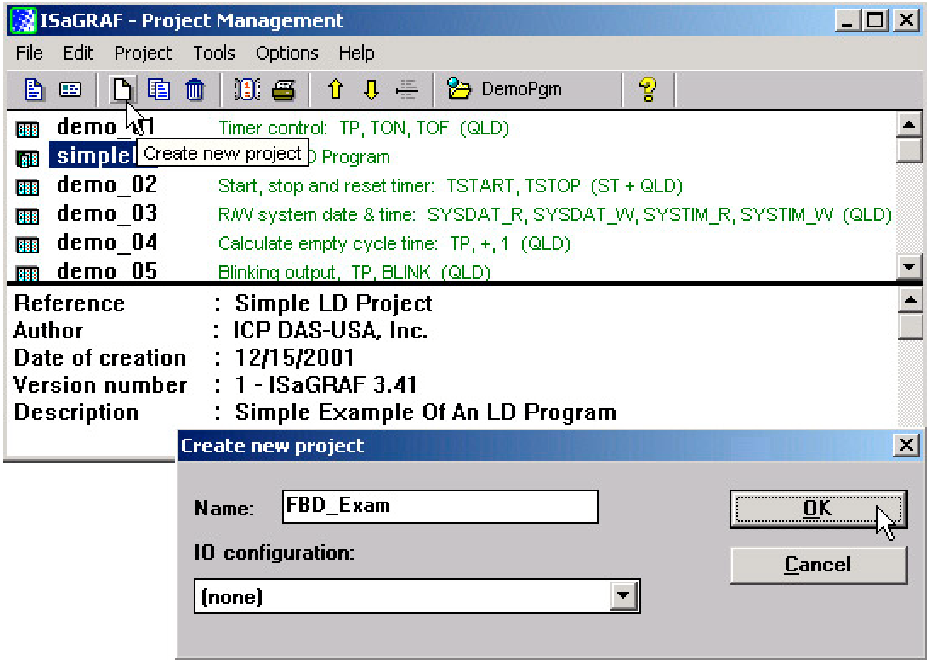
FBD Program Action:

If “A1” > 5000, output “OUT1” is "TRUE".
If “A1” < 2000, output “OUT2” is "TRUE".
Other situation, output “OUT1” and “OUT2” are "FALSE"

2.3.1: Programming The Example FBD Program

Creating a Function Block Diagram (henceforth referred to as "FBD") program is very similar to creating a LD program as outlined in Section 2.1. The following steps detail how easy it is to create a FBD program.

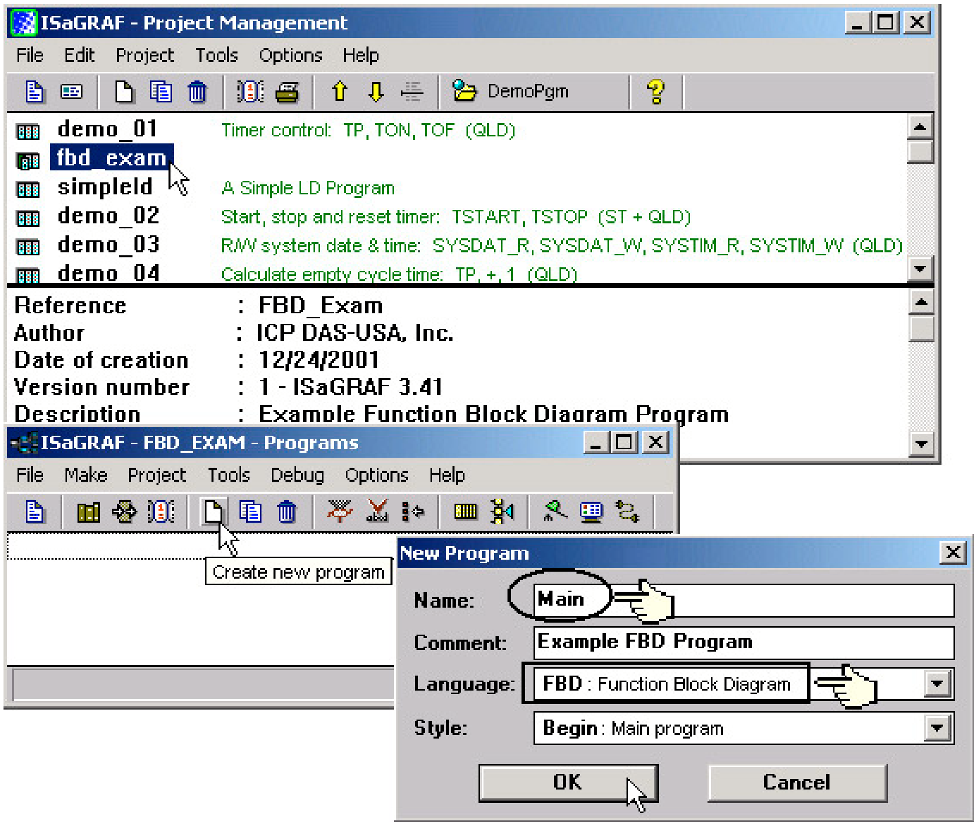
Creating a New FBD Project
From the "ISaGRAF Project Management" window click on the "Create New Project" icon and enter the name "FBD_Exam".



After you have created the new FBD project, double click on the "FBD_Exam" name in the "ISaGRAF Project Management" window to open the new FBD project. Click on the "Create New Program" icon in the "ISaGRAF Programs" window, which will open the "New Programs" window.

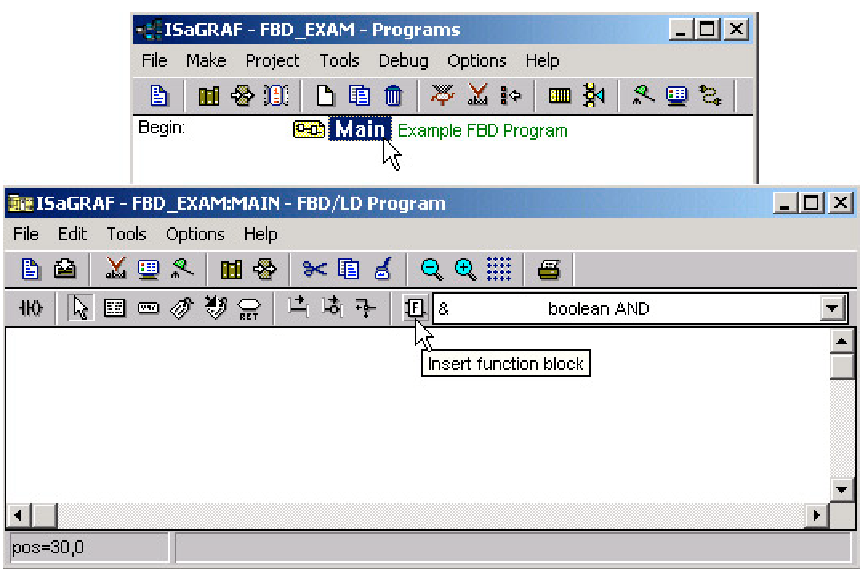
In the "New Programs" window enter in the name field "Main", and for "Language" make sure the "FBD – Function Block Diagram" is selected. You can add a comment about your program also while in the "New Program" window, but it is not mandatory.

Once you have entered in all the information in the "New Programs" window click on the "OK" button.

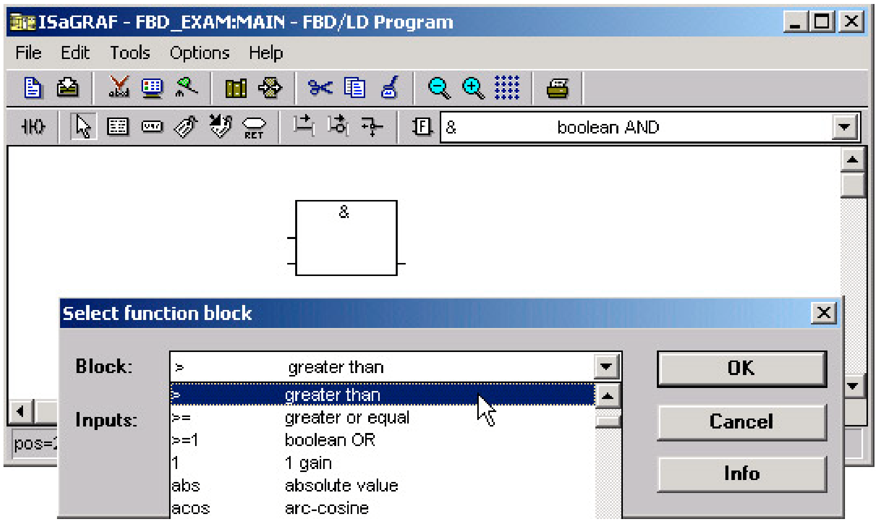


Declaring The Variables
For our example FBD program we are going to declare three variables. The variables to be used are "OUT1", "OUT2", and an integer variable called "A1". Declaring variables for the FBD program is like declaring variables for the LD program. Refer to Section 2.1.1.3 – "Declaring The Variables" to review the variable declaration process.

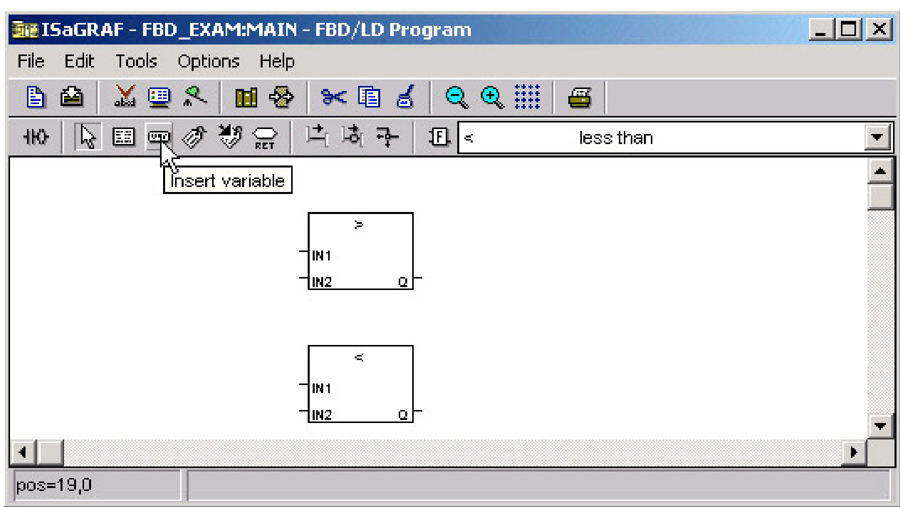
Editing The FBD Program
To create and edit the example FBD program, double click on word "MAIN" in the "ISaGRAF Programs" window, and then click on the "Insert Function Block" icon as shown below.



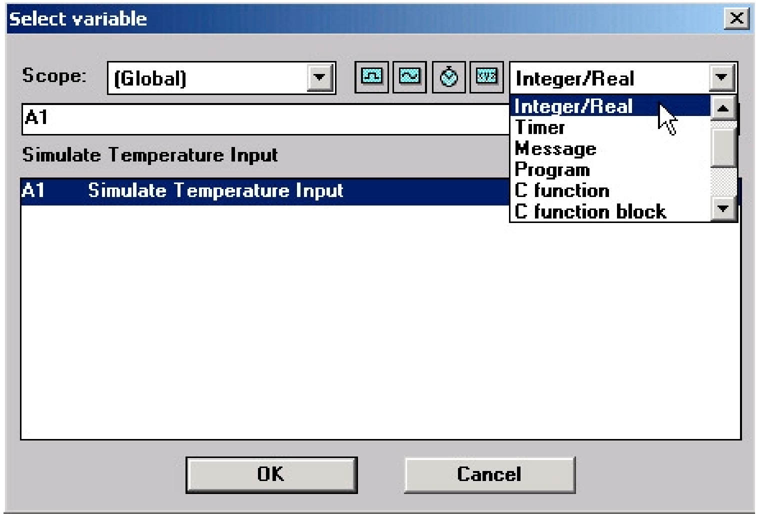
Move the cursor to approximately the middle of the "ISaGRAF FBD/LD Program" window and click the mouse one time to add the first function block. Next, double click on the block to select "> Greater Than". For more information regarding any of the function blocks available in the ISaGRAF program just click on "Info" button.



Using the same procedure as described above, add a "< Less Than" function block below the "Greater Than" function block.



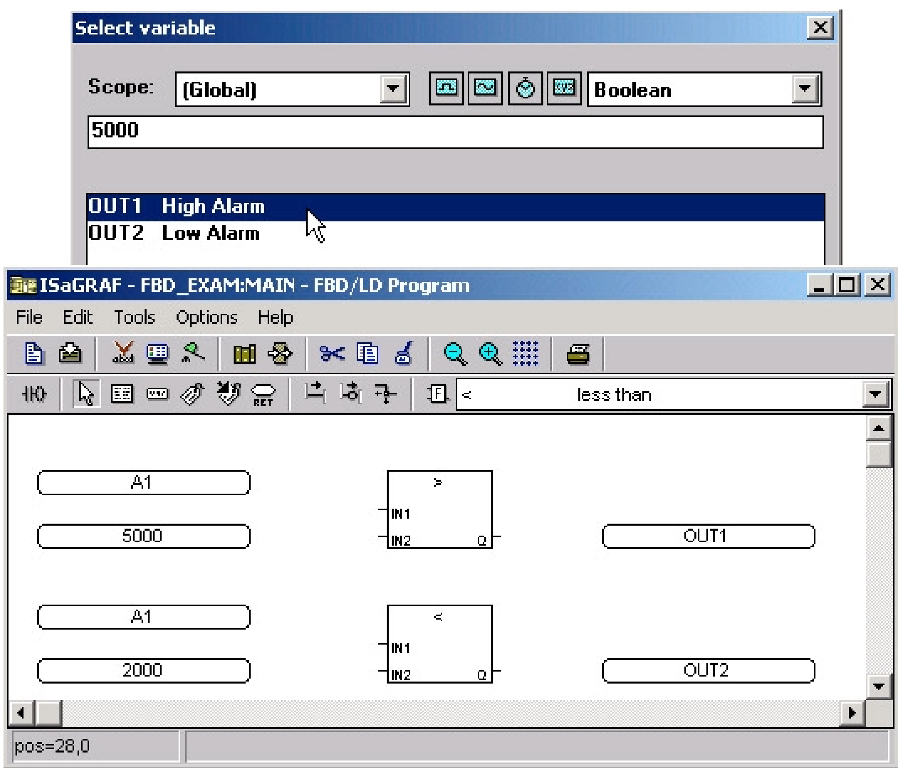
Now it is time to add the program variables to the FBD example program. Click on the "Insert Variable" icon as shown above, and then click on "Integer/Real" from the "ISaGRAF Select Variable" window. This will cause the variable "A1" to appear in the "ISaGRAF Select Variable" to appear.



Double click on the highlighted "A1 Simulate Temperature Input" which will then place the variable "A1" inside of the "ISaGRAF FBD/LD Program" window. Repeat the same process to add a second "A1" variable.

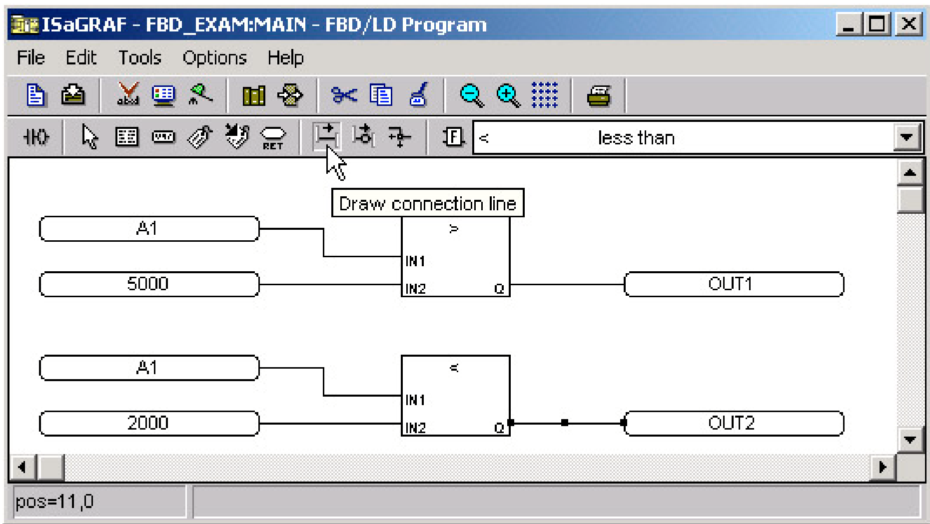
Click on the "Insert Variable" icon to add the "OUT1" and "OUT2" variables to the right of the function blocks as shown below.

Lastly, add two additional variables, the first is a constant of "5000" and place it below the first "A1" variable, then create a second constant of "2000", and place it below the second "A1" variable.



Your "ISaGRAF FBD/LD Program" window should now look like the above example. Remember, we have added a total of six variables to the program. We have added the "A1" variable twice, the "OUT1" variable, the "OUT2" variable, one constant called "5000" and another constant called "2000" to the FBD example program.

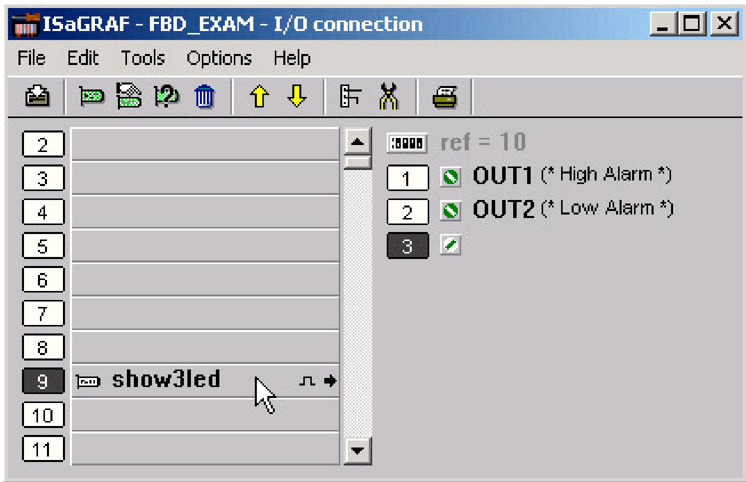
The last task to accomplish is making the connection between each of the variables (and constants) and the function blocks. Click on the "Draw Connection Line" icon and draw a line between each of the variables and function blocks as shown below.



The top "A1" variable should connect to the "IN1" of the "> Greater Than" function block, the "5000" constant to the "IN2" of the "> Greater Than" function block, the bottom "A1" variable to the "IN1" of the "< Less Than" function block, and the "2000" constant to the "IN2" of the "< Less Than" function block.

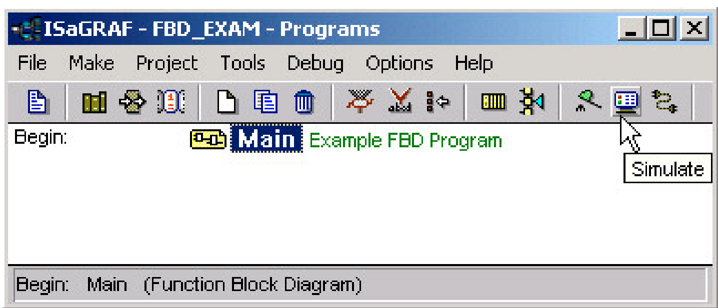
Lastly, connect the "Q" of the "> Greater Than" function block to the "OUT1" variable, and the "Q" of the "< Less Than" function block to the "OUT2" variable.

Connecting The I/O & Compiling The Project
Follow the same procedure as outlined in Section 2.1.2 and 2.1.3 for connecting the I/O and compiling the FBD example program. The "ISaGRAF I/O Connection" window should look like the example below.



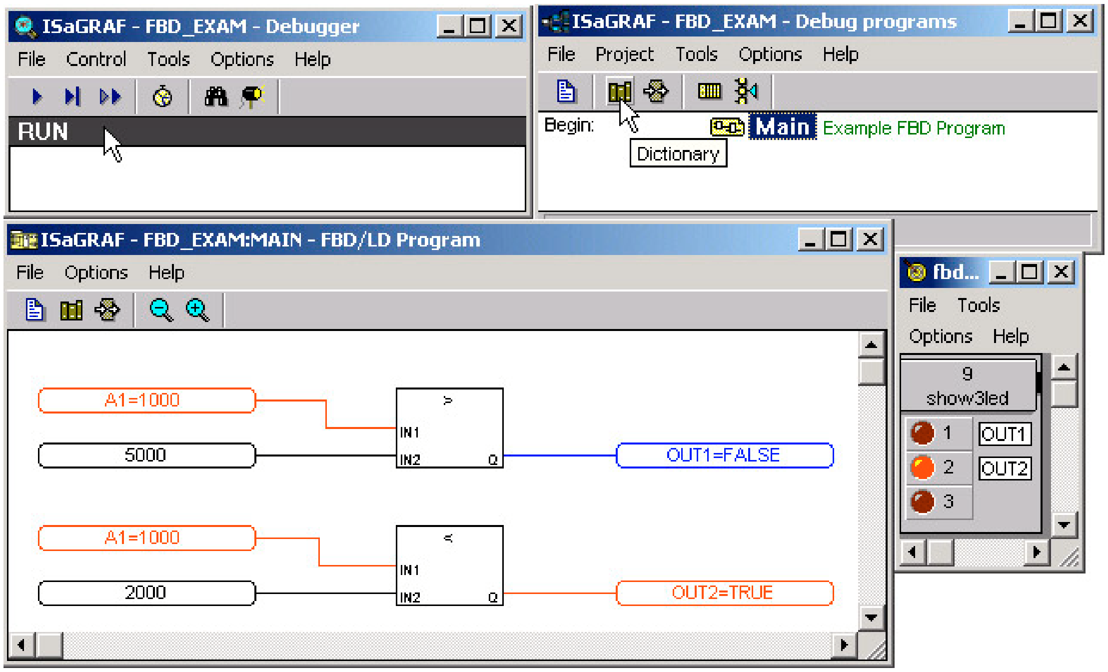
2.3.2: Simulating The FBD Program

You can now run the "Simulate" on the example FBD program by clicking on the "Simulate" icon in the "ISaGRAF Programs" window.

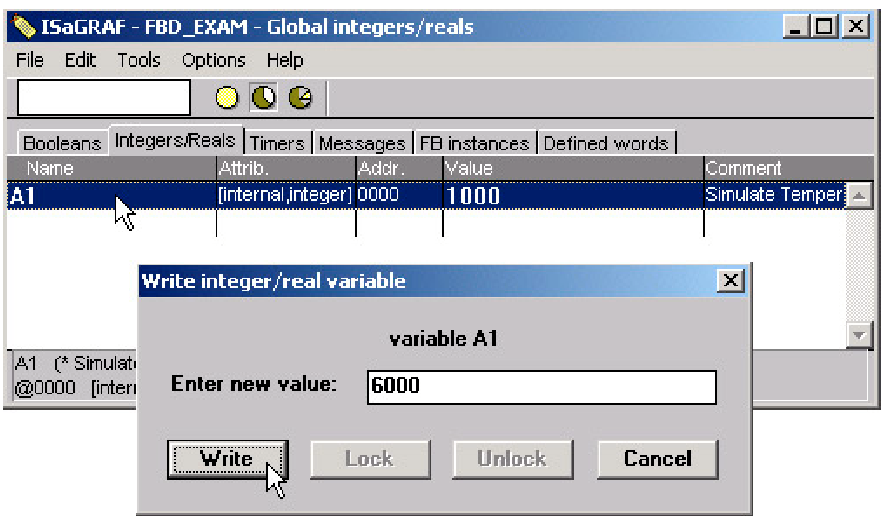


When you click on the "Simulate" icon the "ISaGRAF Debugger" window, the "ISaGRAF Debug Programs", and the "I/O Simulator" window will now open. If you double click on "MAIN" in the "ISaGRAF Debug Programs" window the "ISaGRAF FBD/LD Program" window will open showing the state of the program.

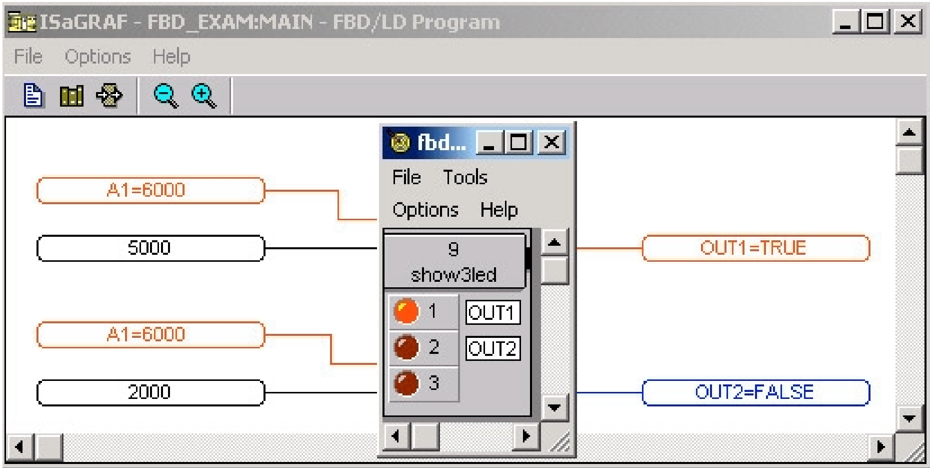
Notice that because the "A1" variable is less than 2000 (currently set to 1000 in the example below) that the "OUT2" output is currently true and the "OUT1" output is false.



To further test the example FBD program, click on the "Dictionary" icon in the "ISaGRAF Debug Programs" window to open the "Global Dictionary" window, and click on the "Integer/Real" tab. Click on the highlighted "A1" and the "Write Integer/Real Variable" will open.



Type in "6000" in the "Enter New Value" field and click on the "Write" button. Now the following changes will be observed.



You can now download the example FBD program to the I-8xx7 controller system. Follow the same procedure as outlined in Section 2.1.5 for downloading the program to the I-8xx7 controller system.

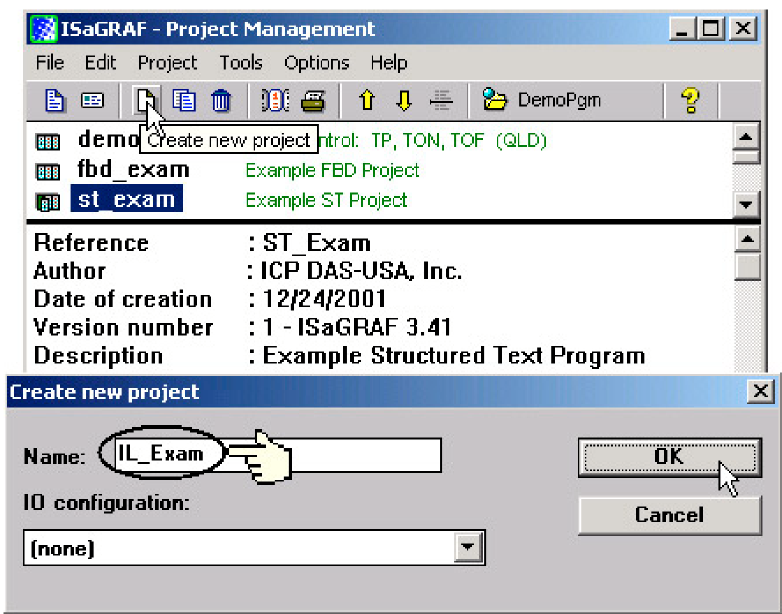
2.4: A Simple Instruction List (IL) Program

Instruction List (IL) programming is a low level programming language consisting of a list of instructions. Each instruction always relates to the **current result** (or **IL register**) and must begin on a new line and must contain an **operator.** The operator indicates the operation that must be made between the current value and the **operand**. The result of the operation is stored again in the result.

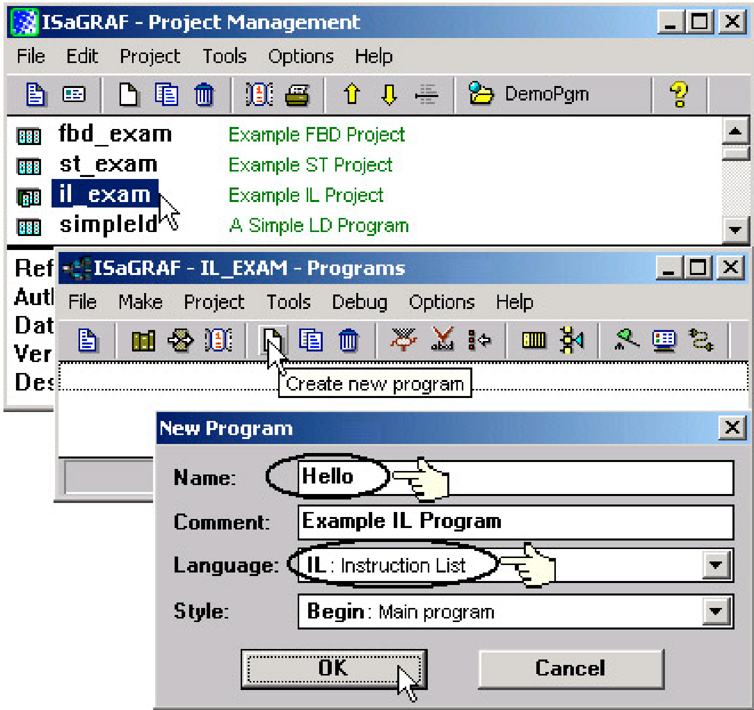
Instruction List (IL) programming requires adherence to a strict programming format that must be followed. Each instruction must begin on a new line, it must contain an **operator**, completed with optional modifiers and if necessary, for the specific operation, one or more operands, separated with commas (","). A **label** followed by a colon (":") may precede the instruction. If a comment is attached to an instruction, it must be the last component of the line. Comments must always begin with **(*** and end with ***).** The following is an example of a comment in IL; **(* place comment here *)**.

This section describes how to program an Instruction List (henceforth referred to as IL) program. This IL program has the same program specification as the FBD program as outlined in Section 2.3.

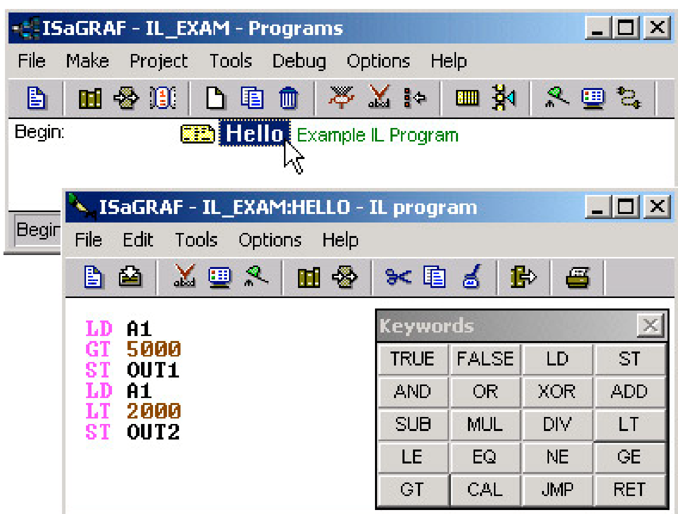
The first step to creating an IL program is to create an IL project. This is accomplished in the same manner as creating any other ISaGRAF project.



For the purpose of this example IL program I have created a new IL project name of "IL_Exam". Click on the "OK" button and the "ISaGRAF Project Management" window will appear with the new project name. Double click on the "IL_Exam" name and the "ISaGRAF Programs" window will appear. Click on the "Create New Program" icon and the "New Program" window will appear. Enter "Hello" in the name field (and you can add a program comment if desired) and make sure to select "IL: Instruction List" from the language field, click on the "OK" button when you are done.



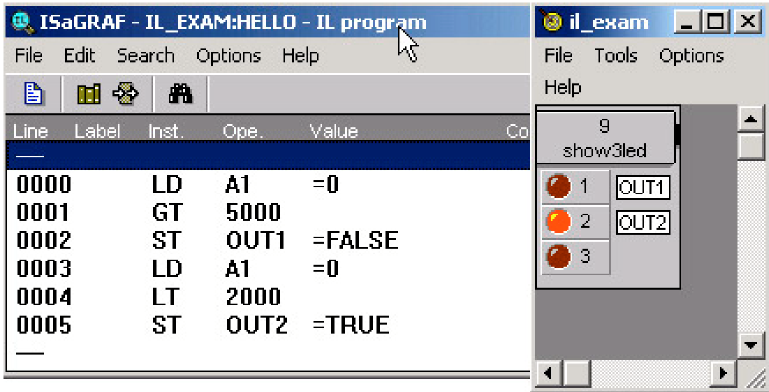
When you click on the "OK" button the "ISaGRAF Programs" window will open. Double click on "Hello" and the "ISaGRAF IL Program" window will open.



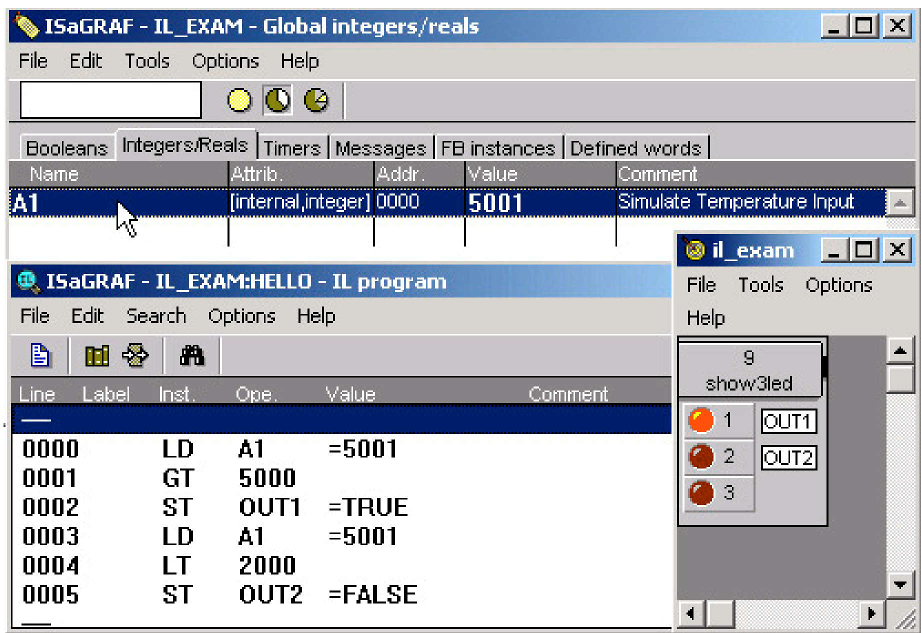
Declaring The Example IL Variables

This example IL program uses the same variables as the example FBD program, "OUT1", "OUT2" and the integer variable "A1". Refer to Section 2.1.1.3 "Declaring The Variables" for assistance. Use the same procedure for the "Connecting I/O" and "Compiling" the program as detailed in Section 2.1.2 and 2.1.3, and use the same procedure to "Simulate" the program as detailed in Section 2.3.2.

When you have connected the I/O and compiled the example IL program, click on the "Simulate" icon and the following window will appear.



Because the variable "A1" value is 0, "OUT1" is set to false and "OUT2" is set to true. Change the value of "A1" to a value greater than 5001 and you will see that "OUT1" is set to true and "OUT2" is set to false.



2.5: A Simple Sequential Function Chart (SFC) Program

A Sequential Function Chart (SFC) program is a graphical programming language used to describe **sequential operations**. The process is represented as a set of defined **steps**, linked by **transitions**. A **Boolean condition** is attached to each transition, and **actions** with the steps are detailed by using other languages such as ST, IL, LD and FDB.

An SFE program is a graphical set of **steps** and **transitions**, linked together by **oriented links**. Multiple connection links are used to represent divergences and convergences. Some parts of the complete program may be separated and represented in the main chart by a single symbol, call **macro steps**. The basic graphic rules for an SFC program are:
1. A Step CANNOT Be Followed By Another Step
2. A Transition CANNOT Be Followed By Another Transition

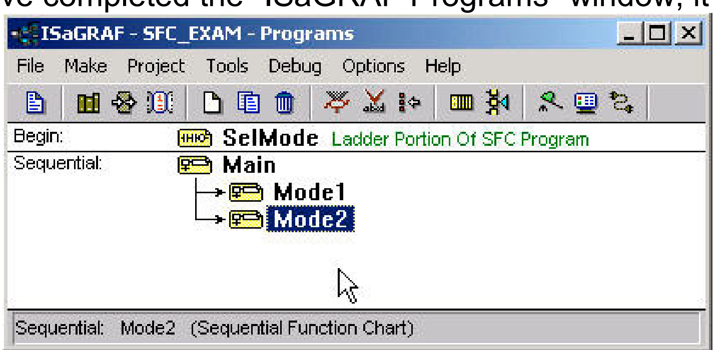
The basic components (graphical symbols) of the SFC programming language are: steps and initial steps, transitions, oriented links, and jumps to a step.

This section details how to build a Sequential Function Chart (henceforth referred to as SFC) program.

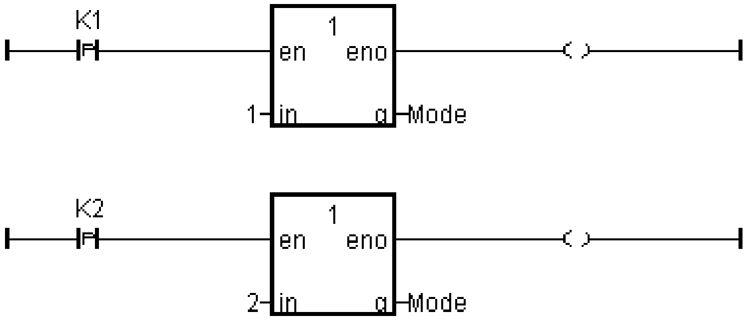
Example SFC Control Specification:
The following details the variables that will be used in our example SFC program.

Name	Type	Attribute	Description
OUT1	Boolean	Output	Output 1
OUT2	Boolean	Output	Output 2
K1	Boolean	Input	Mode 1 button input
K2	Boolean	Input	Mode 2 button input
TMR1	Timer	Internal	Switch time of output, initial value is "T#1s"
Mode	Integer	Internal	1 means mode1 , 2 means mode2, **initial value is 1**

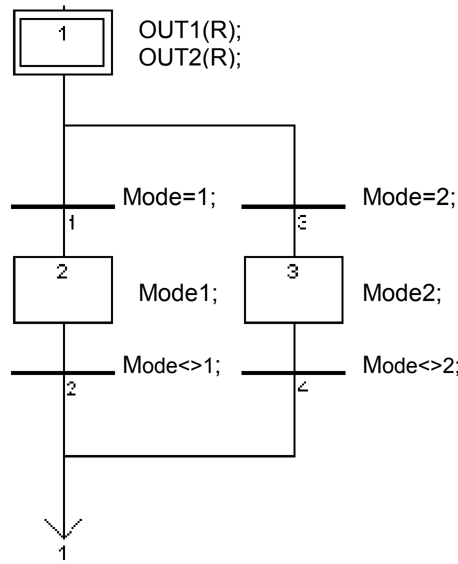
The SFC Program Outline:
When you have completed the "ISaGRAF Programs" window, it should look like the following:



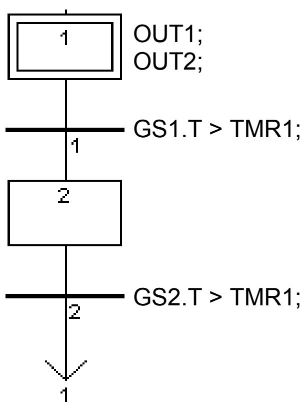
LD Program "SelMode"



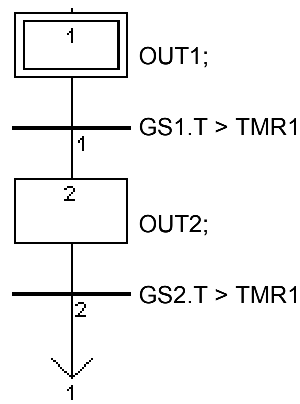
SFC Program "Main"



SFC Child Program "Mode1"



SFC Child Program "Mode2"

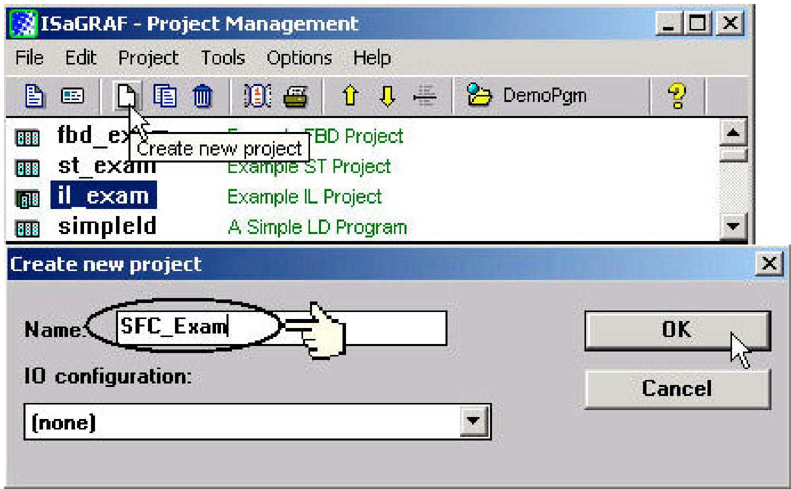


SFC Program Action:

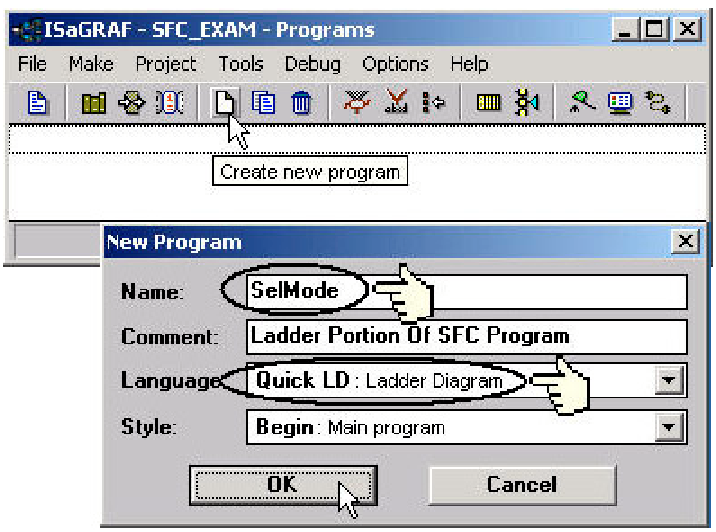
1. When "K1" is pressed, run the "Mode1" program.
2. When "K2" is pressed, run the "Mode2" program.

2.5.1: Programming The Example SFC Program

The procedure for creating the example SFC program is the same as outlined in Section 2.1. You must remember to declare the variables "K1", "K2", "OUT1", "OUT2", "TMR1" and "MODE". The following illustrates creating the new SFC project.



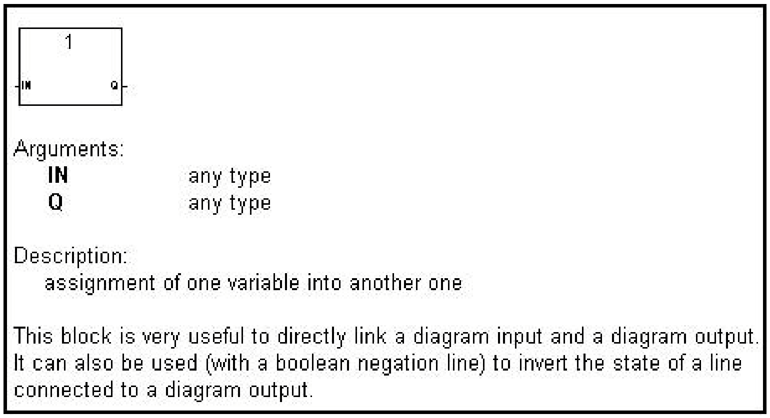
After creating the new SFC project, the next step is to create an LD program named "SelMode" as illustrated below.



When you click on the "OK" button the "ISaGRAF Quick LD Program" window will open. Add the instructions as shown in the example below.

IMPORTANT NOTE:

The example SFC program uses a function block that has not been used throughout the manual. We will be adding the "**1 Gain**" function block to our LD program.

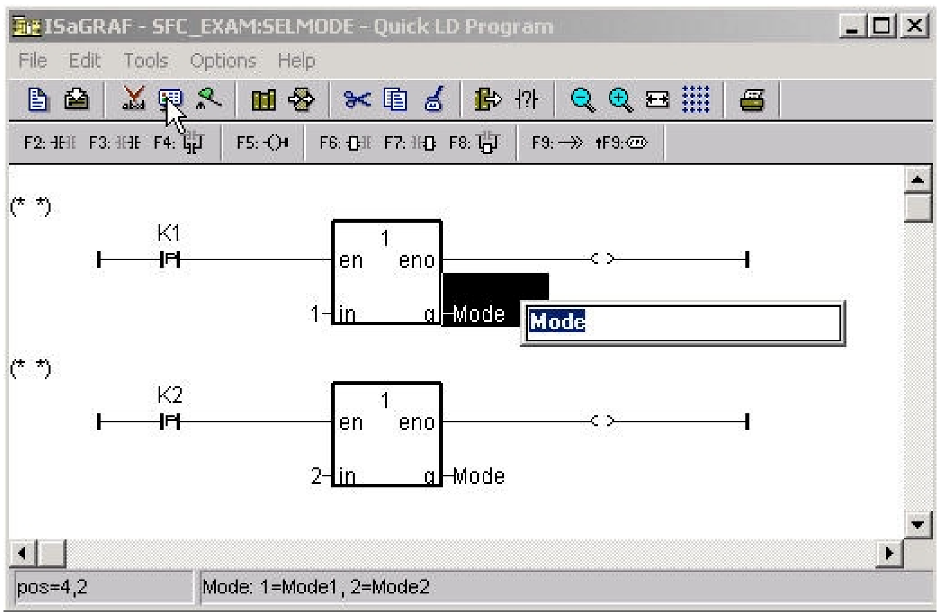


Arguments:
IN any type
Q any type

Description:
assignment of one variable into another one

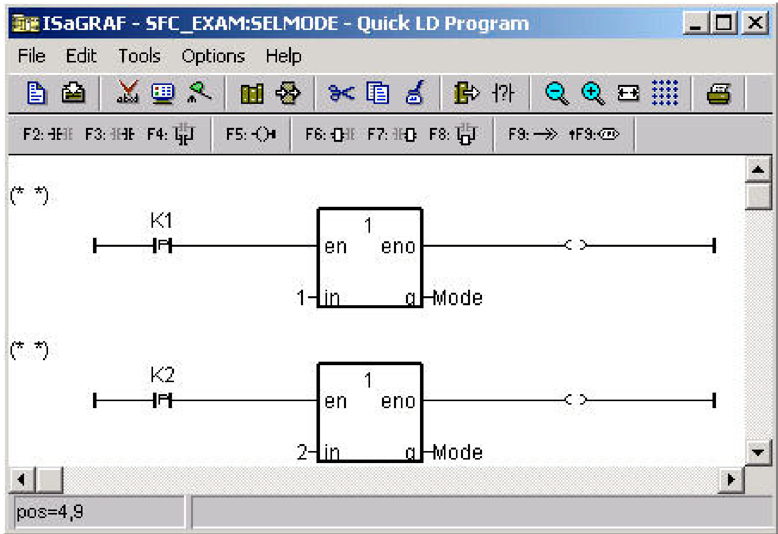
This block is very useful to directly link a diagram input and a diagram output. It can also be used (with a boolean negation line) to invert the state of a line connected to a diagram output.

Even though the "EN" (input) and "ENO" (output) arguments are not shown in the above example, they will be added when you place the "1 Gain" function block in the program.

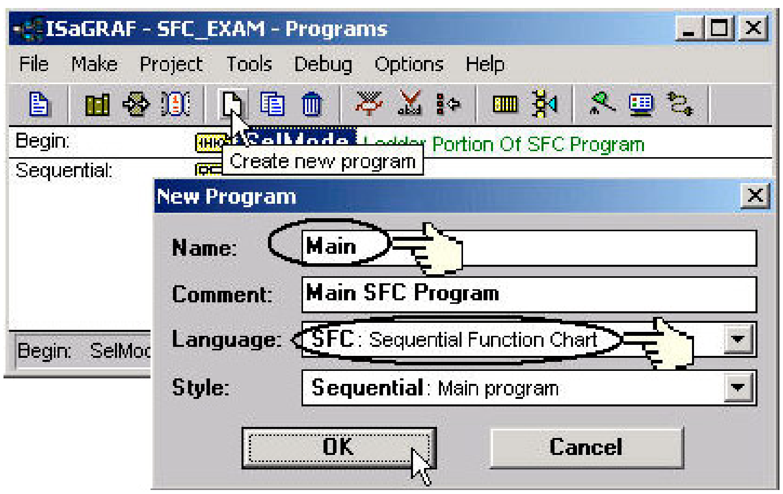


You will need to change the "K1" and "K2" contacts type to "P". The "P" contact (Positive) enables a Boolean operation between a connection line state and the rising edge of a Boolean variable. Place the cursor to the right of the "Q" and click once, then type in "Mode" for both lines of logic. Place the cursor to the left of the "IN" on the top "1 Gain" function block, click once and enter a "1". Do the same for the second LD line and enter a value of "2", then click once on the "Q" and enter in "Mode".

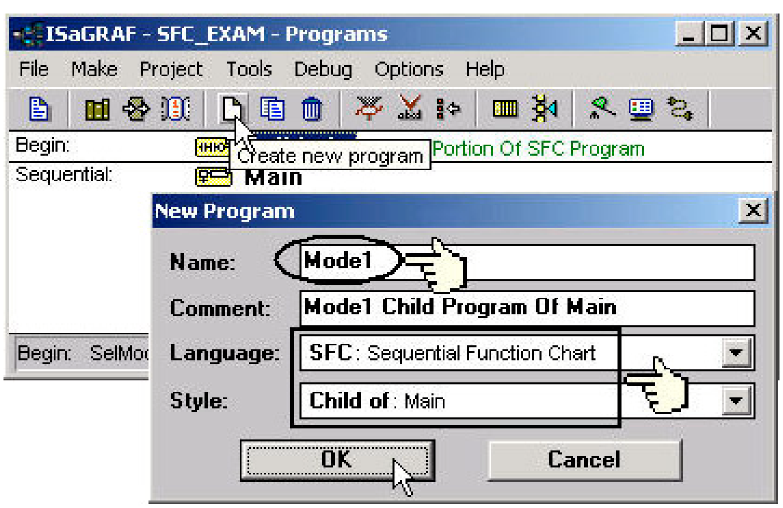
When you are finished editing the "ISaGRAF Quick LD Program" window it should look like the below example.



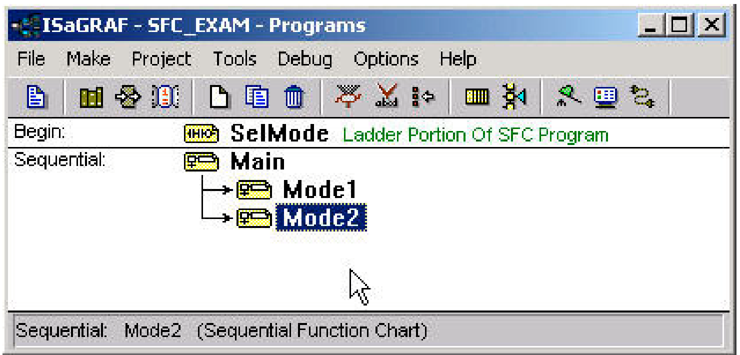
The next step is to create a new SFC program called "Main".



The next step is to create a "CHILD" program called "Mode1".

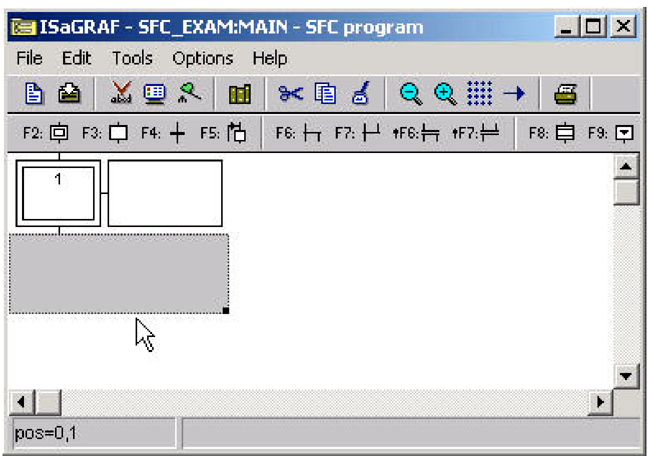


Follow the same procedure to create a second "CHILD" program named "Mode2". When you are completed the "ISaGRAF Programs" window should look as follows.

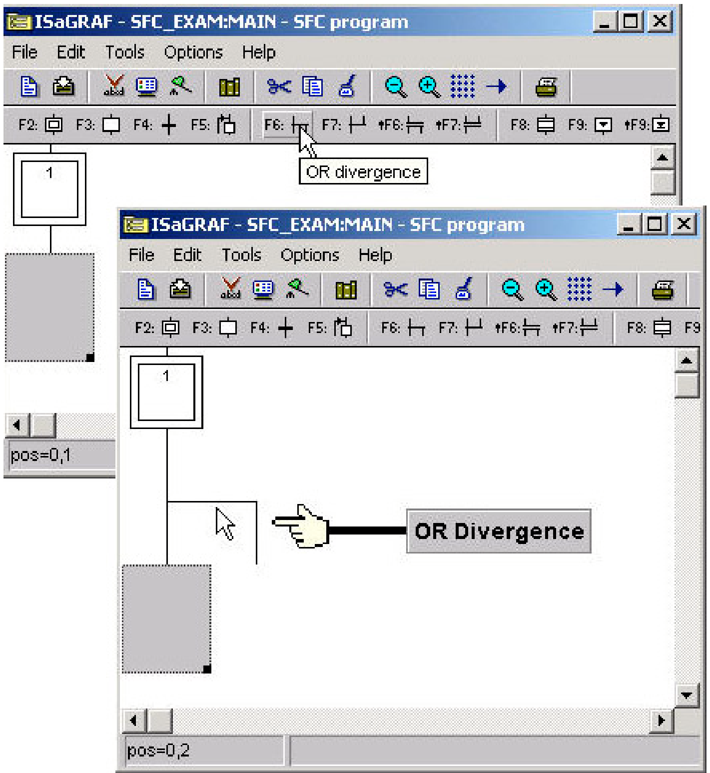


2.5.2: Editing The SFC Program

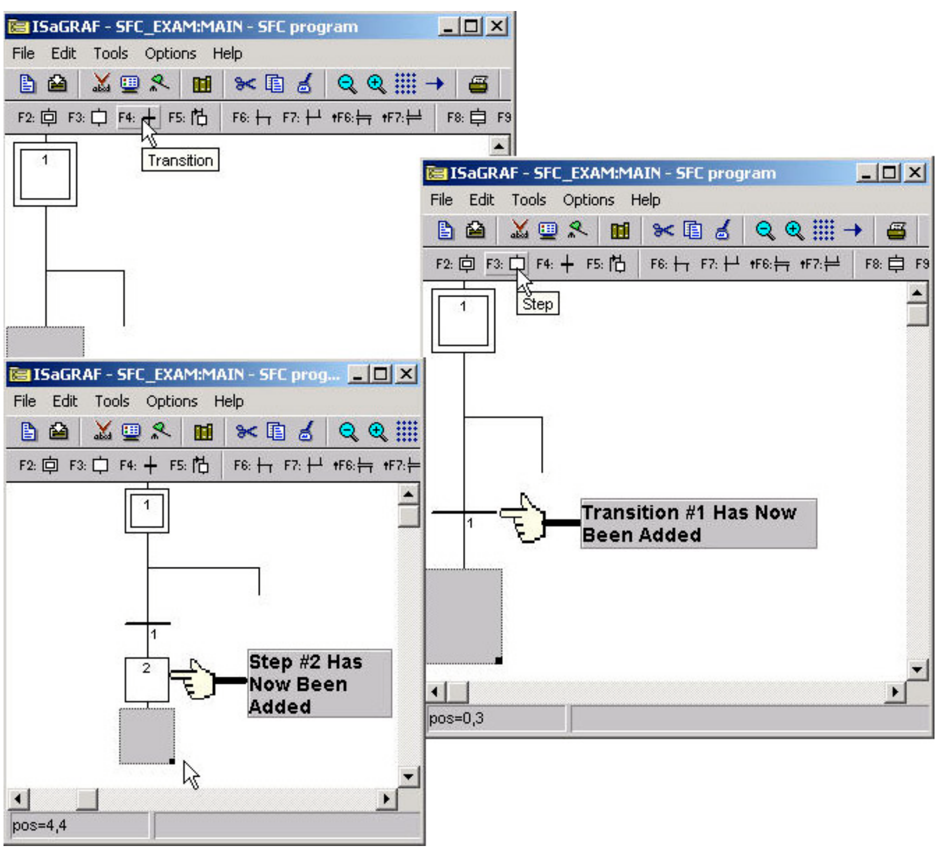
To begin editing the example SFC program double click on "Main" in the sequential portion of the "ISaGRAF Programs" window and the "ISaGRAF SFC Program" window will appear.



You will note an additional box to the right of the initial step box. This box will contain the code for each of the steps and transitions in the example SFC program. The "code box" is not required during the initial programming so you can to get rid of it temporarily by clicking on the black dot in the gray box area below the initial step and resize the window to approximately the size of the initial step box.

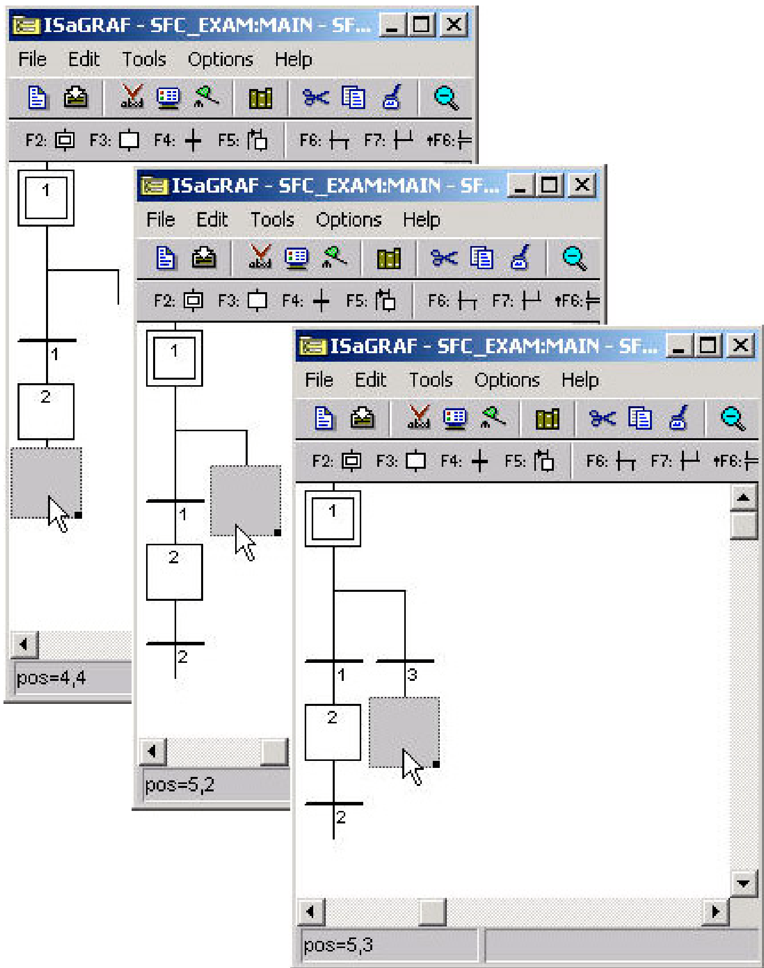


The gray box will move down automatically when you click on the "OR Divergence" icon. The next step is to click on the "Transition" icon to create "Transition 1" and then the "Step" icon to create "Step 2 as shown below.

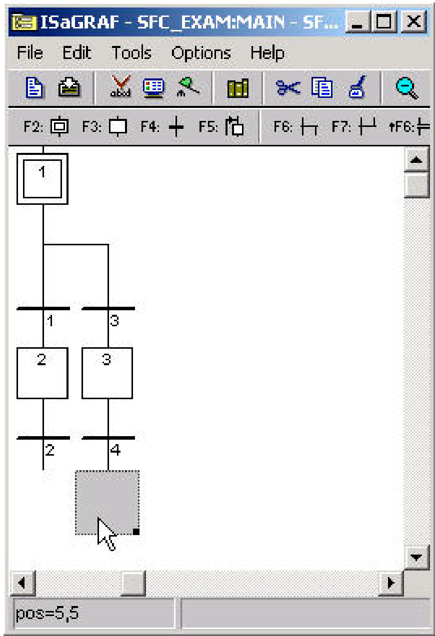


With the gray box below "Step 2" click on the transition button to add a second transition (transition #2) to the example SFC program. After adding the second transition below "Step 2", click directly below the "OR Divergence" so that the gray box is now placed there. Click on the transition icon again with the gray box below the "OR Divergence" to add a third transition (transition #3).

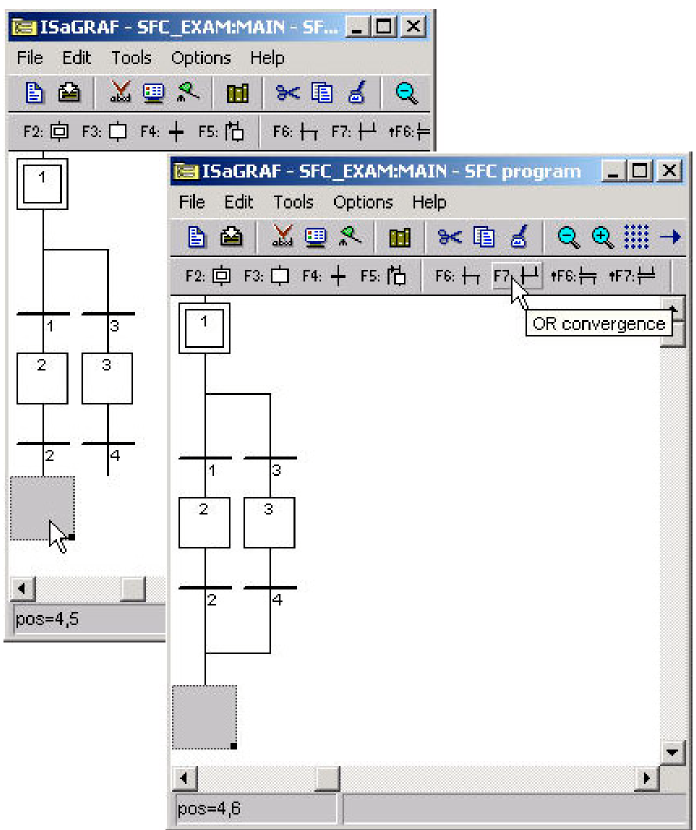
When you have completed these tasks your SFC program should now look like the third SFC picture below.



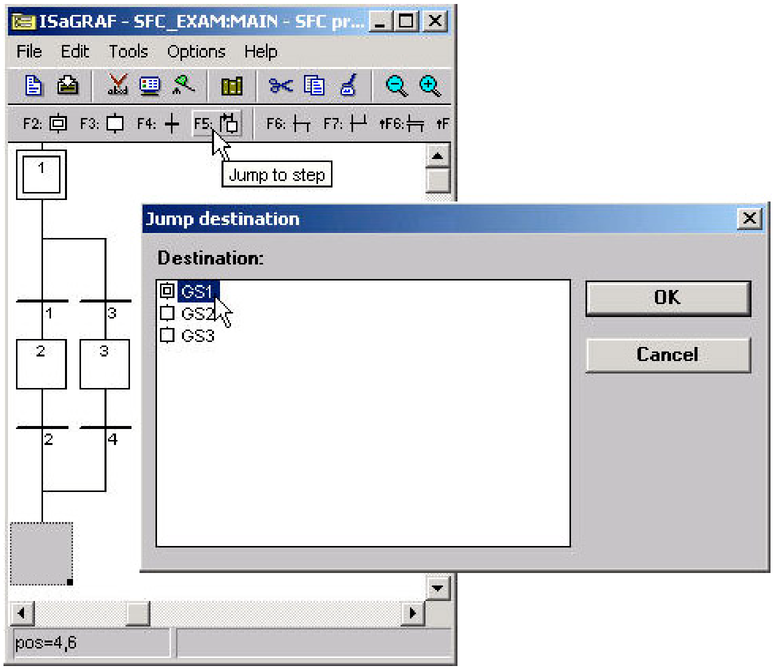
From where the gray box is currently click on the "Step" icon to add Step #3, and then with the gray box below the newly created step #3 click on the transition icon to add a fourth transition (transition #4) to the example SFC program. Your SFC program should now look like the below example.



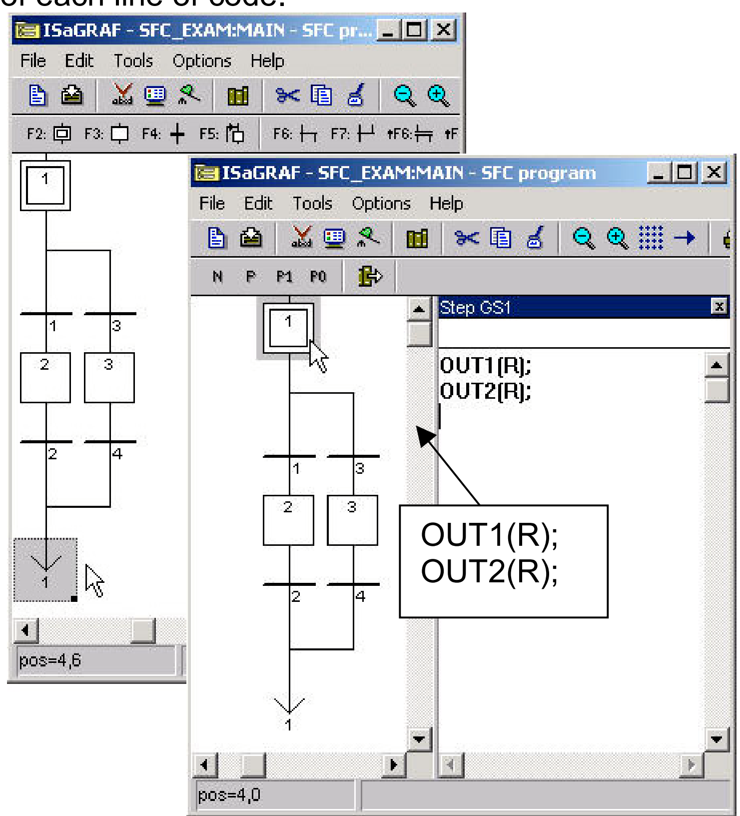
Now click the gray box below transition #2 and click on the "OR Convergence" (F7) icon.



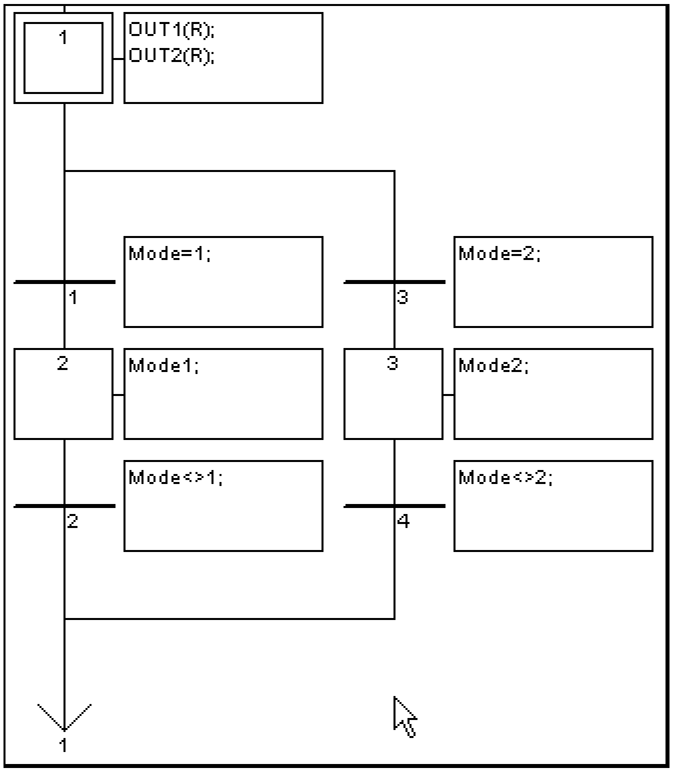
Now click on the "Jump To Step" (F5) icon, this will open the "Jump Destination" window. Double click on the "GS1" label in the "Jump Destination" window.



We have now finished programming the "Main" portion of the example SFC program. The next detail is to add the code for each of the steps and transitions. Double click on step #1 (initial step) and the "ISaGRAF SFC Program" window will open. Type the displayed text into the area shown below. This will associate the typed in code with the step #1. **REMEMBER** to type a semi-colon (":") at the end of each line of code.



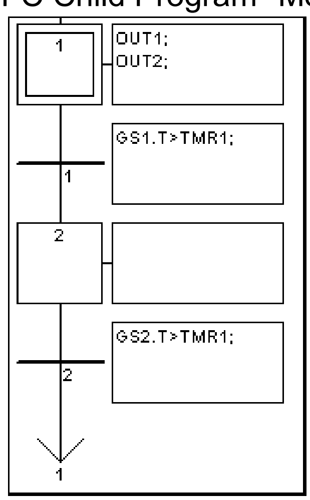
Using the same method as described above, double click on each transition and step and add the code for each item as shown below.

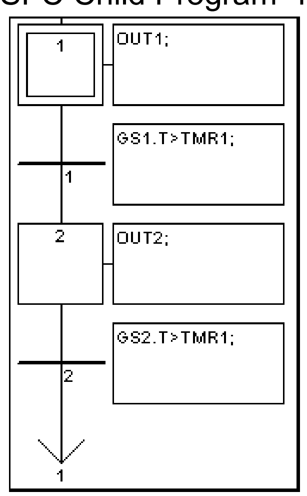


CONGRATULATIONS! You have now successfully programmed the "Main" section of the example SFC program (and the most time consuming).

The last portion of creating the example SFC program requires the creation and editing of the two "CHILD" programs. You program the "CHILD" programs using the exact same method as required for creating the "MAIN" program. When you are finished creating and editing the "CHILD" programs your two windows should look like the examples below.

SFC Child Program "Mode1" SFC Child Program "Mode2"



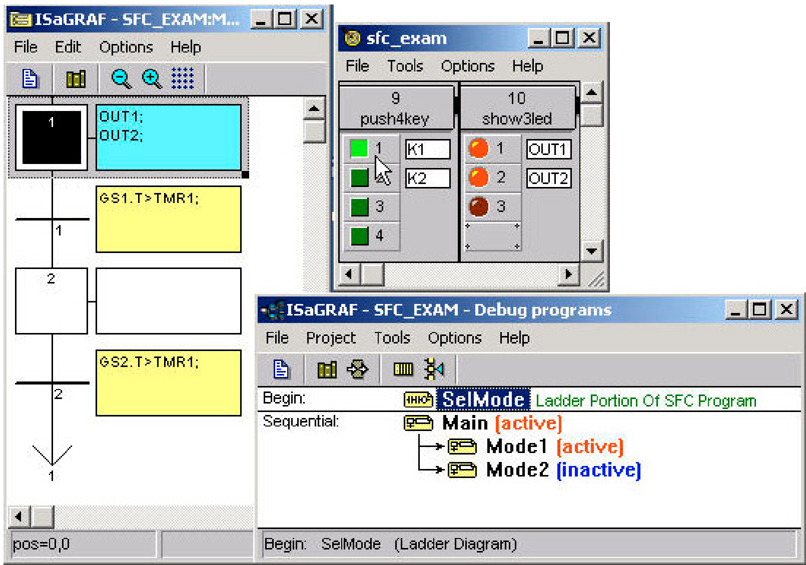


Final Details

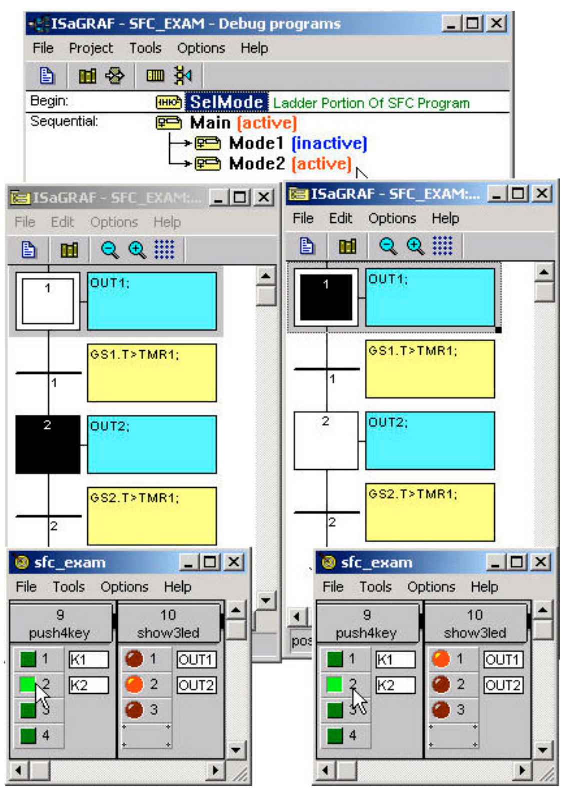
Remember that you must follow the same procedure for "Connecting I/O’s" and "Compiling The Project" as detailed in Section 2.1.2 and Section 2.1.3.

2.5.3: Simulating The SFC Program

After you have successfully compiled the SFC program, you can now run the example SFC program in "Simulate" mode to observe how the two "CHILD" programs work within the "MAIN" SFC program. When "K1" is on, "Mode1" is true and both "OUT1" and "OUT2 turn on and off together, and "Mode2" is false.



When "K2" is on "Mode2" is true "OUT1" will turn on while "OUT2" is off and then they will alternate where "OUT2" will turn on and "OUT1" will be off, and "Mode1" is false.



Chapter 3: Establishing I/O Connections

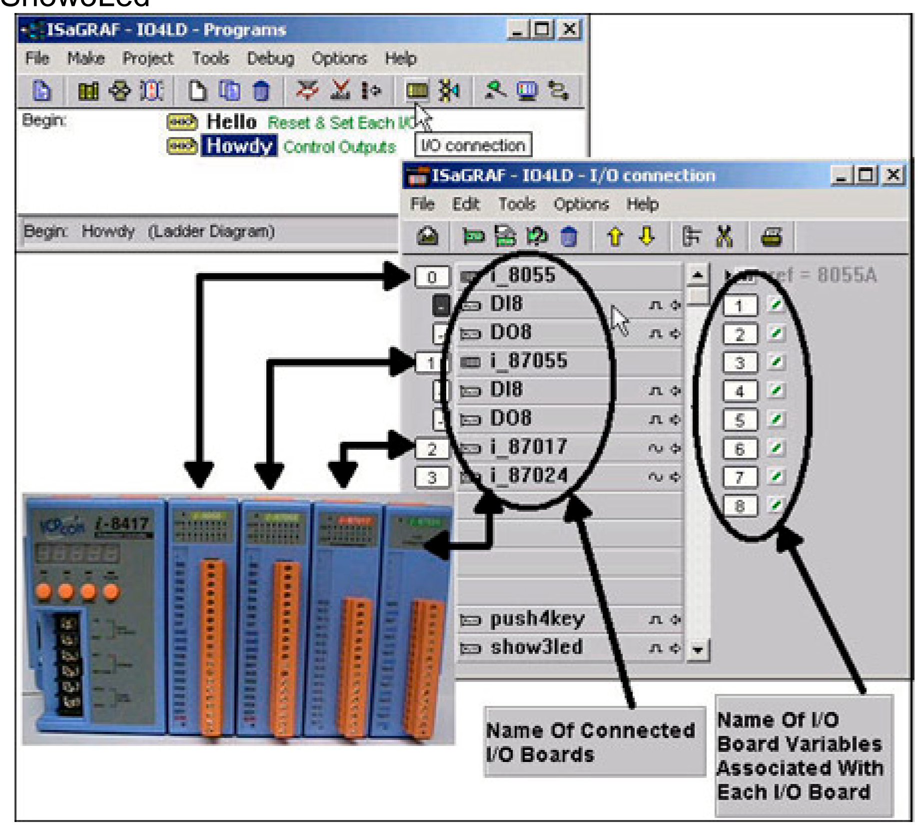
Before you can operate an ISaGRAF program with the I-8xx7, I-7188EG/XG & Wincon-8xx7 controller, you must make sure that the I/O Library has been installed. If you haven't done so already, install it as outlined in Section 1.2 "Installing The ICP DAS Utilities For ISaGRAF".

3.1: Linking I/O Boards To An ISaGRAF Project

To begin connecting I/O boards to an ISaGRAF project you must first link the I/O boards to the ISaGRAF program. The numbers on the left of the "I/O Connections" window indicate the slot number. Slots 0 through 7 are used ONLY for **real** I-8000 series I/O boards(Slot 1 through 7 for W-8xx7). Slots 8 and above can be used for "virtual" I/O boards such as the "Push4Key" and "Show3Led" functions fot I-8xx7. For I-7188EG/XG, slot 0 is for Xxxx serial I/O boards, slot 1 & above are for others.

In this example I/O connection we are using the I-8417 controller system that has the following boards installed:

Slot 0: I-8055 Board (8 digital inputs & 8 digital outputs)
Slot 1: I-87055 Board (8 serial inputs & 8 serial outputs)
Slot 2: I-87017 Board (8 channel analog input)
Slot 3: I-87024 Board (4 channel analog output)
Slot 8: "Push4Key"
Slot 9: "Show3Led"

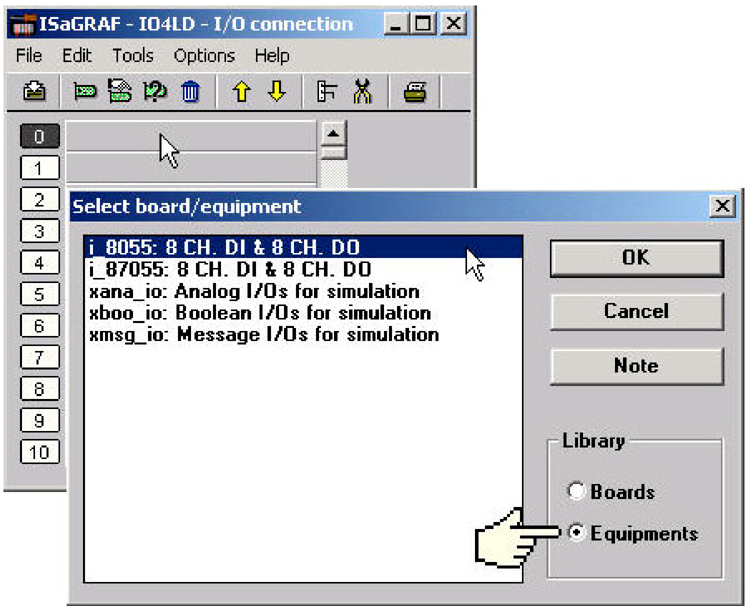


A powerful feature of the I-8xx7 controller system is that you can intersperse "real" I/O boards with "virtual" I/O boards.

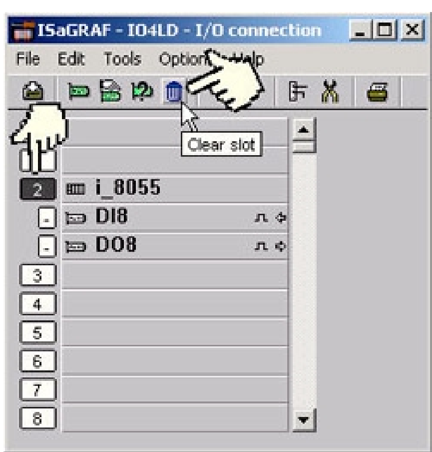
3.1.1: Linking I/O Boards

With the "I/O Connection" window open double click on the slot that you want to connect an I/O board to. The "Select Board/Equipment" window will open, scroll to the name of the I/O board that you want to associate with the particular slot.

The ISaGRAF controller library defines two basic types of real I/O boards, "Boards" and "Equipments". The "Boards" selection is for I/O boards that are "single type", meaning that all of the channels on that board are of a single type and attribute. The "Equipments" selection is for I/O boards that are "multi-type", which means boards that have multiple types (such as the I-8055 digital I/O board that has 8 digital inputs and 8 digital outputs all on the same board). To begin the linking I/O board process, double click on the slot that you want to associate an I/O board to.



If you link an I/O board to an incorrect slot, first click on the slot number you wish to correct, then just click on the "Clear Slot" icon to delete the connection. The connection is now cleared, and now you can make a connection to the desired slot location.

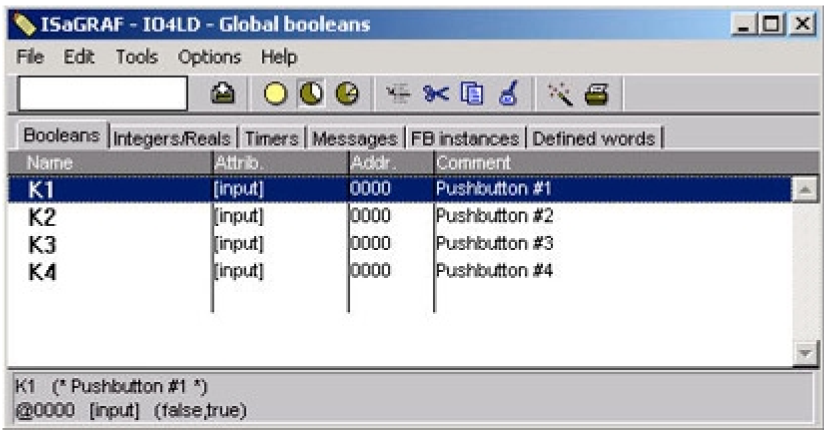


3.1.2: Linking Input & Output Board Variables

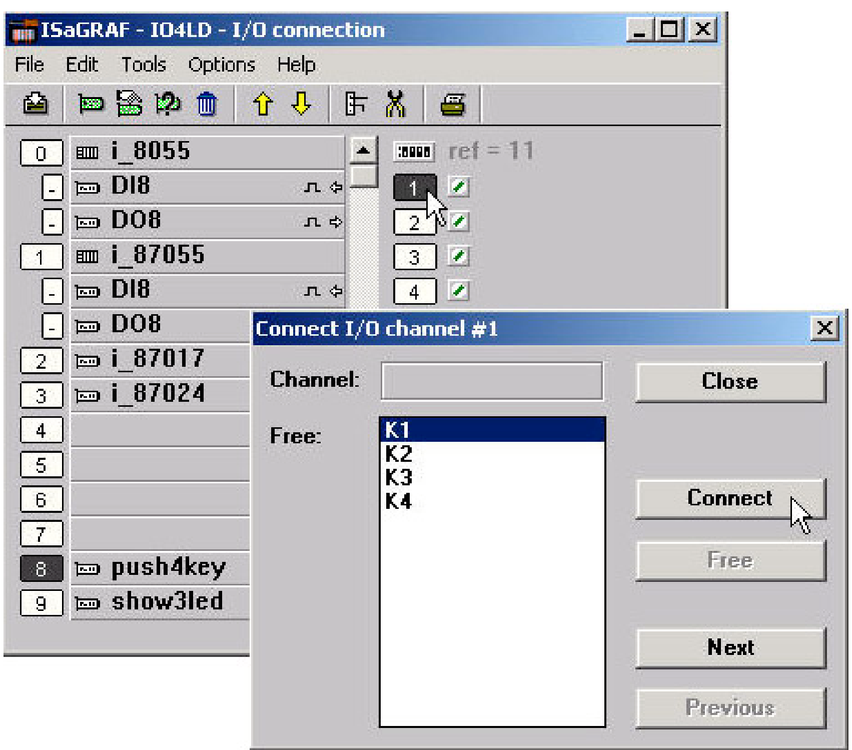
All of the input and output board "variables (or names)" must be linked (connected) in the "I/O Connection" window. Click on the slot you wish to link the attribute to, then double click on the channel (or I/O point name) number on the right hand portion of the "I/O Connection" window. Lastly, choose the variable name you wish to link to and then click on the "Connect" button.

IMPORTANT NOTE

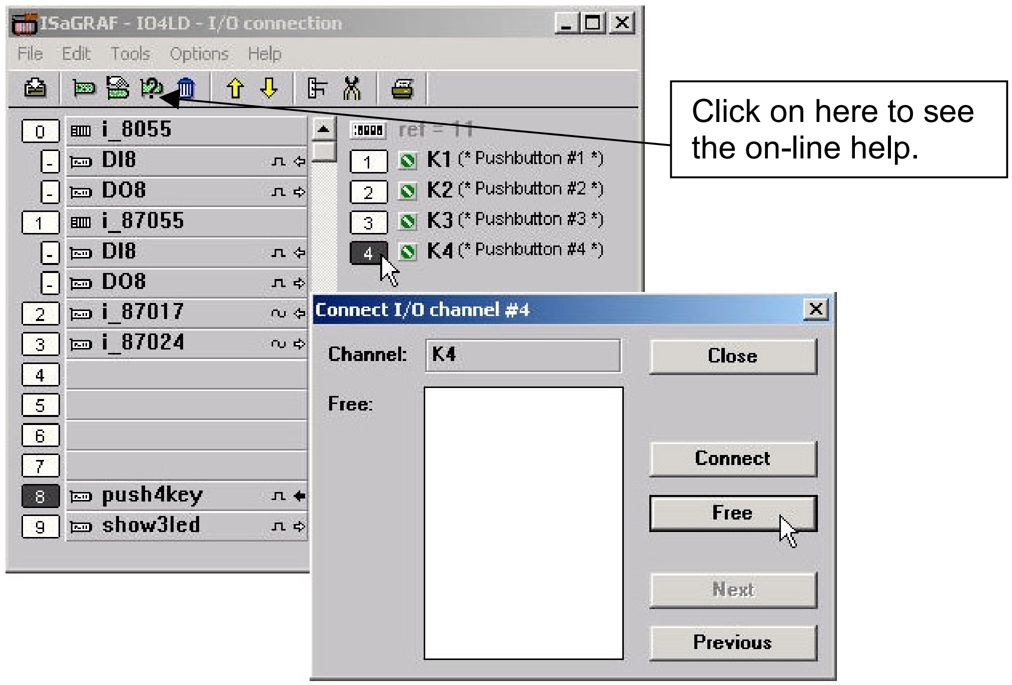
Remember that before you can assign any input or output, you must FIRST declare the variable in the "ISaGRAF Global Variables" window as shown below.



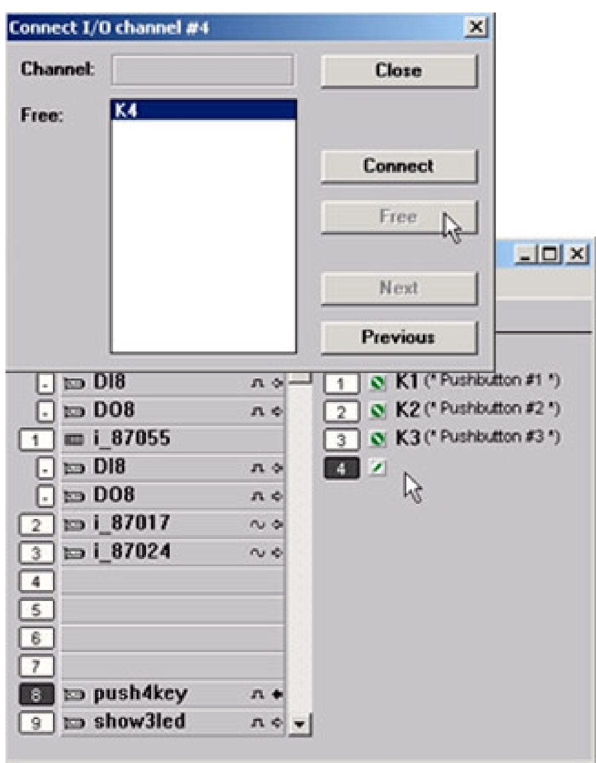
Click once on slot 8, then double click on "1" on the right hand side of the "ISaGRAF I/O Connection" window. With the "Connect I/O Channel #1" window now open, click on the "Connect" button to create the link between the variable "K1" and channel number 1 of the "Push4Key" input.



If you connect an input or an output variable to the wrong (or undesired) I/O location, double click on the I/O point you wish to remove. The "Connect I/O Channel #x" will open then click on the "Free" button to remove that variable from the I/O point.

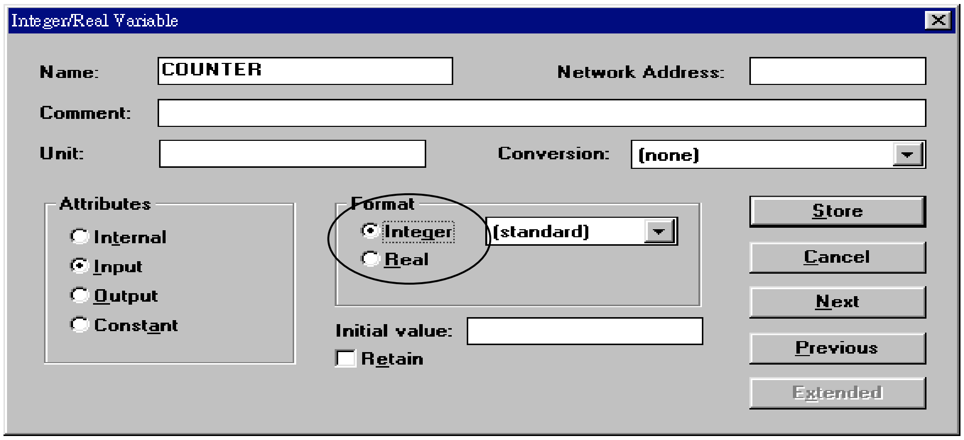


When you click on the "Free" button you will see that the variable is removed from the I/O point in the "ISaGRAF I/O Connection" window and the variable is placed in the "Free" portion of the "Connect I/O Channel #x" window.

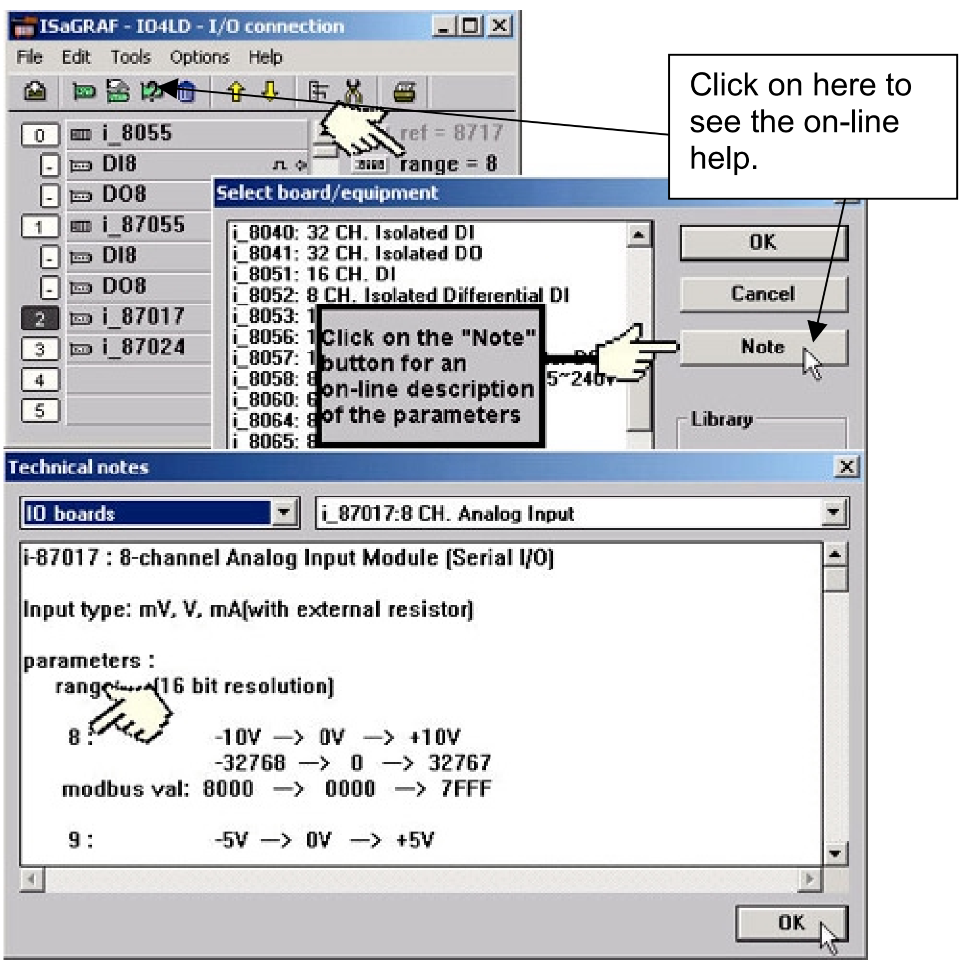


3.2: Linking Analog Type I/O Boards

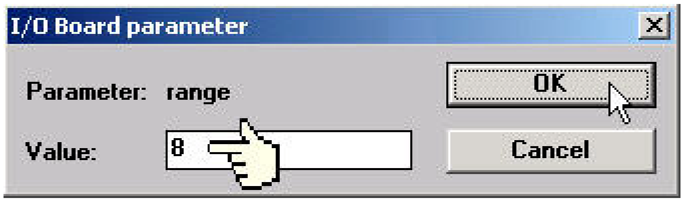
The method to connect analog type I/O boards to the controller system is very similar to that of connecting digital I/O boards. First, variables which are connected to analog type I/O boards should be declared as “Interger” format.



The ONE main difference is that you MUST define one parameter that defines the range for the analog board so it will operate as expected.



To modify the analog board "Range" parameter, click on the word "Range" in the "ISaGRAF I/O Connection" window and the "I/O Board Parameter" window will open. Enter in the correct "Range" parameter for your particular analog board application.



The below table provides information on several of the possible options for the "Range" parameter. Note that the default value is set to "8", which means you can interface to a –10v to +10v signal with a range value of –32768 to 32767. Changing the value of "Range" parameter to "9" means you can interface to a –5v to +5v signal with a range value of –32768 to 32767.

Note that if you set the "Range" parameter to "A" you will be interface to a –1v to +1v signal with a range value of –32768 to 32767. This range value can be very helpful in analog applications that require a great deal of resolution over a very small range (typically temperature) control.

```
range:    (16 bit resolution)

 8 :            -10V  --> 0V  --> +10V
                -32768 --> 0  --> 32767
modbus val:  8000  -->  0000  --> 7FFF

 9 :            -5V --> 0V  --> +5V
                -32768 --> 0  --> 32767
modbus val:  8000  -->  0000  --> 7FFF

 A :            -1V --> 0V  --> +1V
                -32768 --> 0  --> 32767
```

Please refer to **Appendix D - "Table of The Analog IO Value"** for more information for several different types of analog boards and their respective ranges.

3.3: Linking "Push4Key" & "Show3Led"

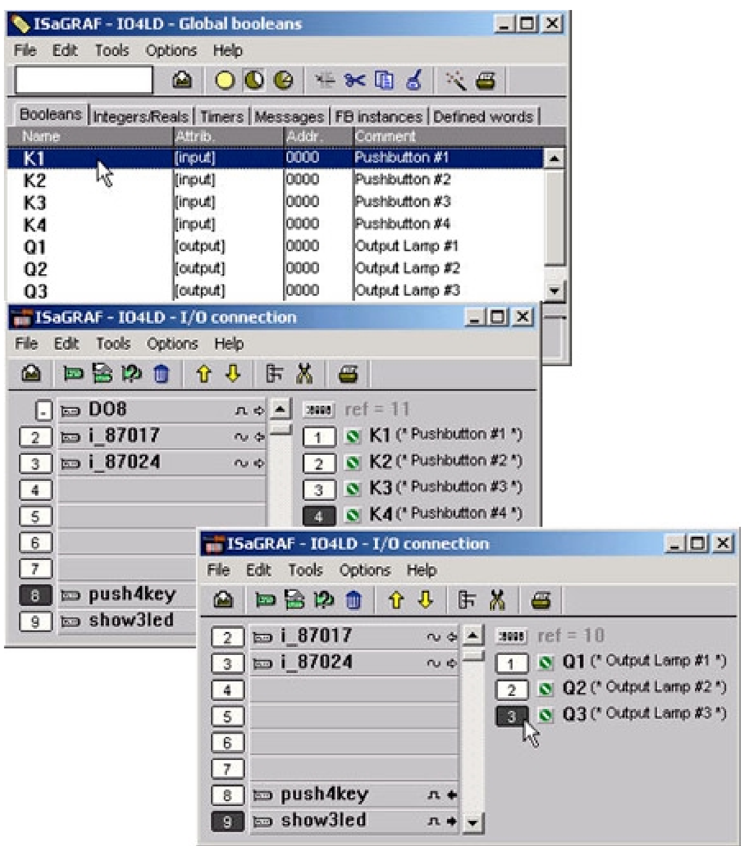
The I-8xx7 controllers have an additional feature that is useful for program testing and debugging. These features are the "Push4Key" and "Show3Led" on the front panel on the I-8xx7 controller system.

Note:
I-7188EG/XG & Wincon-8037/8337/8737 doesn’t support “Pusg4Key” & “Show3Led”

The "Push4Key" are the four pushbuttons on the I-8xx7 control front panel and they are handled as digital inputs. The "Show3Led" are three of the four LED’s on the I-8xx7 control front panel (the first three from left to right, the fourth LED is strictly to show if the power is turned on the I-8xx7 controller system) and they are handled as digital outputs.

Both of these can be linked to an ISaGRAF program through the "I/O Connection" window and can be used to interface with Man Machine Interface (MMI) programs or for program debugging. It is recommended that you assign these functions to slot 8 or higher (remember, slots 0 through 7 are reserved for real I/O boards.

IMPORTANT NOTE:
As with any real digital input or real digital output, you MUST declare a variable name for each of the "Push4Button" inputs and "Show3Led" outputs in the "ISaGRAF Global Variables" window BEFORE they can be assigned to an ISaGRAF program.



3.4: Directly Represented Variables

If you have an ISaGRAF-256 or ISaGRAF-L workbench (Version 3.4x or 3.5x) with a dongle, you don't need to use the skill described in this section.

A very useful feature of the ISaGRAF Workbench program is the ability to create "directly represented (or internal)" variables. Internal variables are program variables that can be used in an ISaGRAF program, but they are not physically connected to any of the input or output variables. There are four versions of the ISaGRAF Workbench program available with the I-8xx7 controller system: ISaGRAF-32, ISaGRAF-80, ISaGRAF-256, and ISaGRAF-L. The number after "ISaGRAF" represents the number of I/O variables that are allowed with that particular ISaGRAF Workbench program.

The ISaGRAF Workbench program comes with a hardware protection device (dongle) that plugs directly into your development computers parallel port. Every time you compile a program in ISaGRAF the hardware protection device is read to make sure that you are not trying to connect to more program variables than are allowed with your particular copy of the ISaGRAF Workbench program that you purchased with your I-8xx7 controller system.

These "directly represented (henceforth called "internal") variables can be used in lieu of your real world inputs and outputs so you can create additional program variables that do not count against the amount of ISaGRAF program variables. The only "caveat emptor" to these internal variables is that you must follow a strict programming scheme to program and access these internal variables, and they are more complicated to create than the regular input and output variables. **For a professional programmer, recommend to purchase an ISaGRAF-256 workbench rather than an ISaGRAF-80 or ISaGRAF-32 workbench for programming on I-8xx7, I-7188EG/XG & Wincon-8xx7 controllers.**

Single Type Internal Variable Programming Scheme:

For single-typed board: "**s**" is the slot No, "**c**" is the channel No.

%IX**s.c**	free channel of a **boolean input** board,	ex. %IX2.3
%QX**s.c**	free channel of a **boolean output** board,	ex. %QX0.2
%ID**s.c**	free channel of an **integer input** board,	ex. %ID3.1
%QD**s.c**	free channel of an **integer output** board,	ex. %QD2.4
%IS**s.c**	free channel of a **message input** board,	ex. %IS3.1
%QS**s.c**	free channel of a **message output** board,	ex. %QS2.4

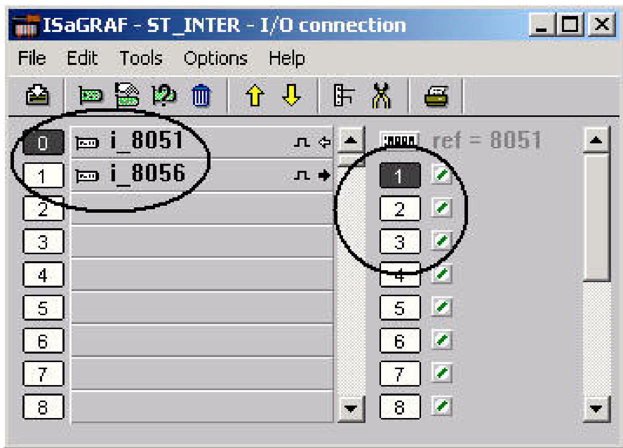
Complex Type Internal Variable Programming Scheme:

For complex board: "**s**" is the slot No, "**b**" is the index of the single board within the complex equipment. "**c**" is the channel No.

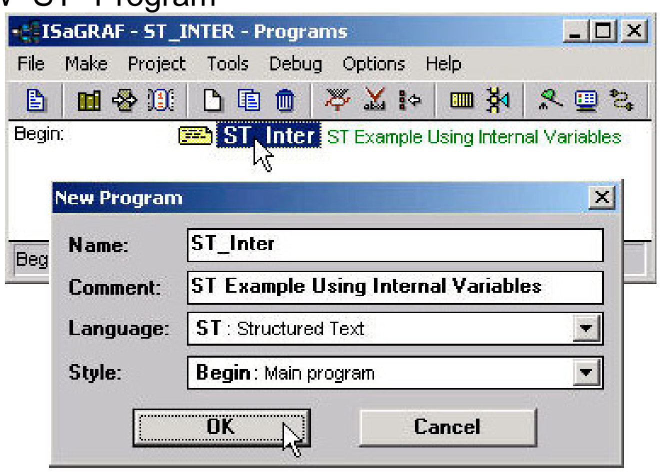
%IX**s.b.c**	free channel of a **boolean input** board,	ex. %IX2.3.2
%QX**s.b.c**	free channel of a **boolean output** board,	ex. %QX0.2.1
%ID**s.b.c**	free channel of an **integer input** board,	ex. %ID3.1.3
%QD**s.b.c**	free channel of an **integer output** board,	ex. %QD2.4.3
%IS**s.b.c**	free channel of a **message input** board,	ex. %IS3.3.1
%QS**s.b.c**	free channel of a **message output** board,	ex. %QS2.1.4

An Internal Variable Program Example

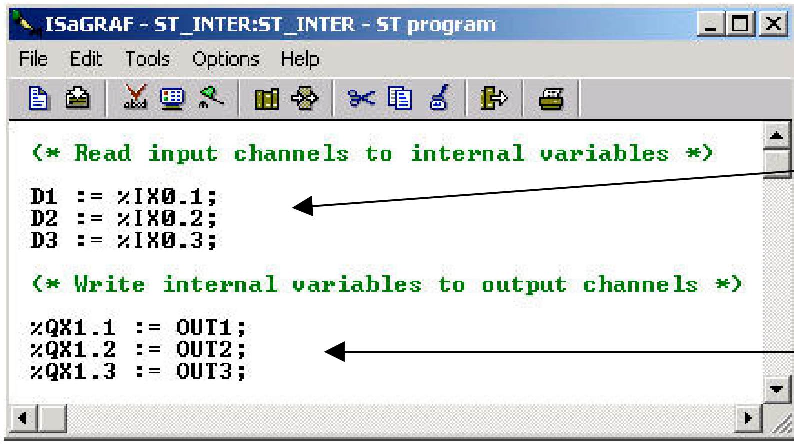
Create a new project for an ISaGRAF ST program, and then create a link to the I/O boards that are specified in the window below. Declare three input variables called "D1", "D2", & "D3" for the I-8051 board located at slot 0, and then create three output variables called "OUT1", "OUT2", & OUT3" for an I-8056 board located at slot 1. This time set each of their respective attributes to "internal" instead of input or output (this means they are not connected to any real physical I/O).



Create A New "ST" Program



Double click on the "ST_Inter" that is highlighted and the "ISaGRAF ST Program" window will open. Type in the program code displayed in the window below EXACTLY as shown. Remember, each line MUST end with a semi-colon (";").

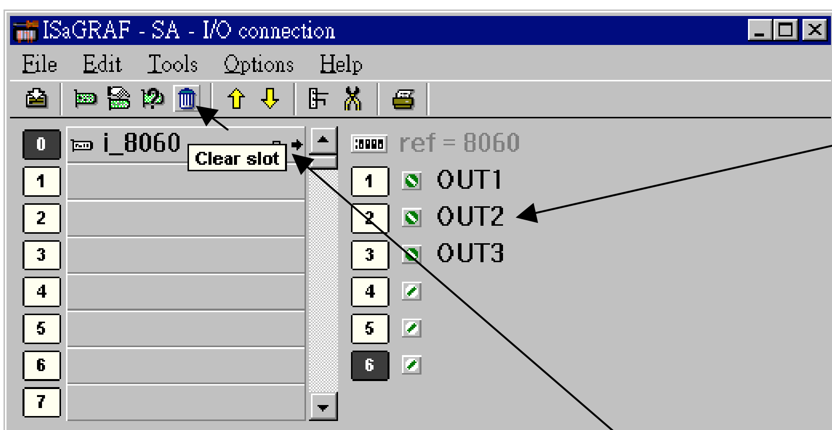


D1 := %IX0.1 ;
D2 := %IX0.2 ;
D3 := %IX0.3 ;

%QX1.1 := OUT1 ;
%QX1.2 := OUT2 ;
%QX1.3 := OUT3 ;

Now we can use the internal variables D1 through D3 and OUT1 through OUT3 that have been created in other programs in the same project. The newly created internal variables will generate input and output actions to the associated channels in this ST program.

IMPORTANT NOTE:
If once the input or output attributed variables have been connected to an connected IO board or complex equipment, and if they would like to be replaced by Directly represented variables, these input or output attributed variables have to be re-attributed to “internal” and the board or equipment **must be re-connected to the slot**.



If you wish to replaced these variables by directly represented variables, re-attributed them to “internal” attribution in the “dictionary” window.

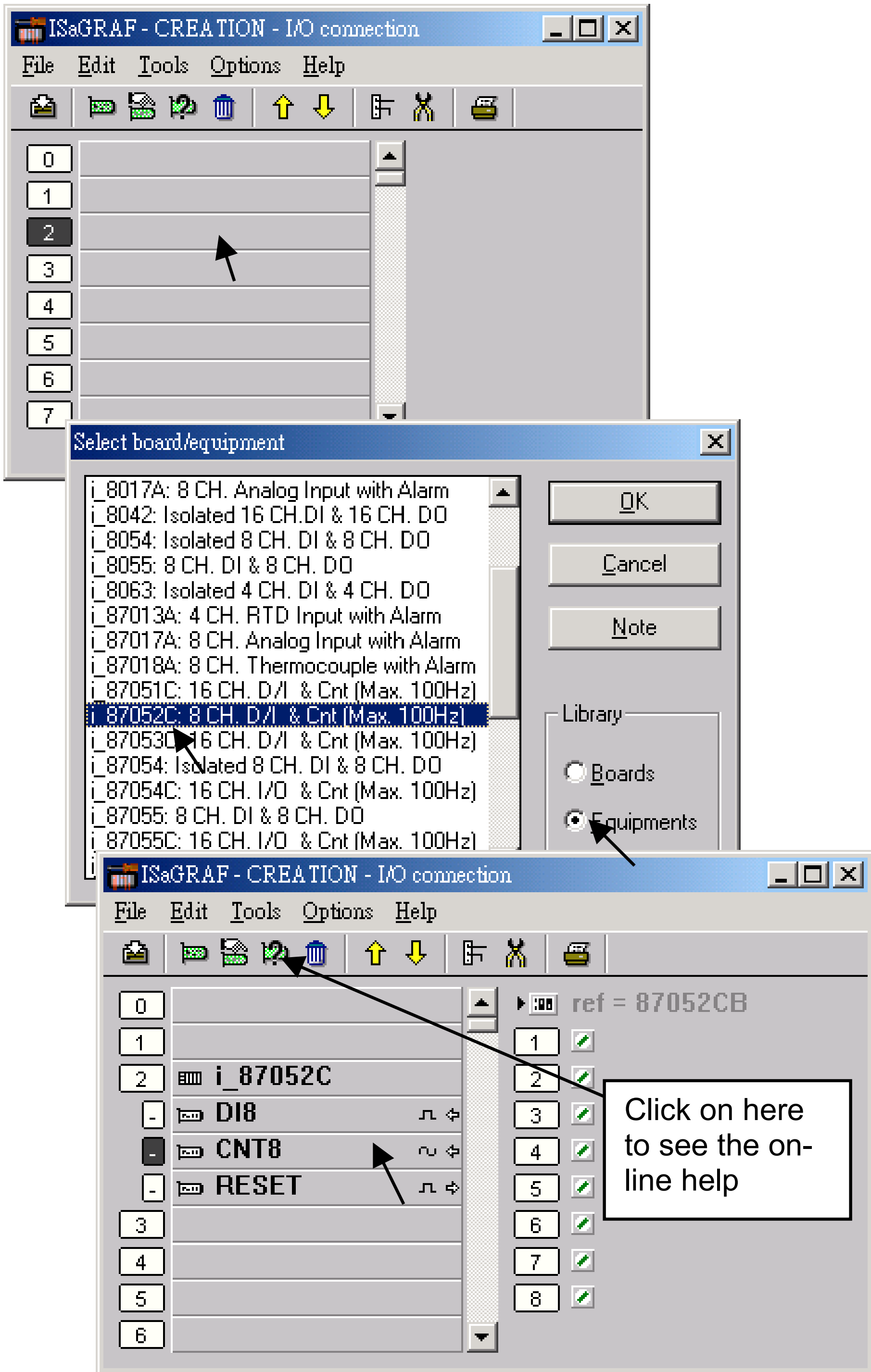
Clear slot and re-connect again.

IMPORTANT NOTE
If you enable the compiler option of upload, option “**Comments for not connected I/O channels**” must be choosed if “Directly represented variables” is used in this project (refer to section 9.2).

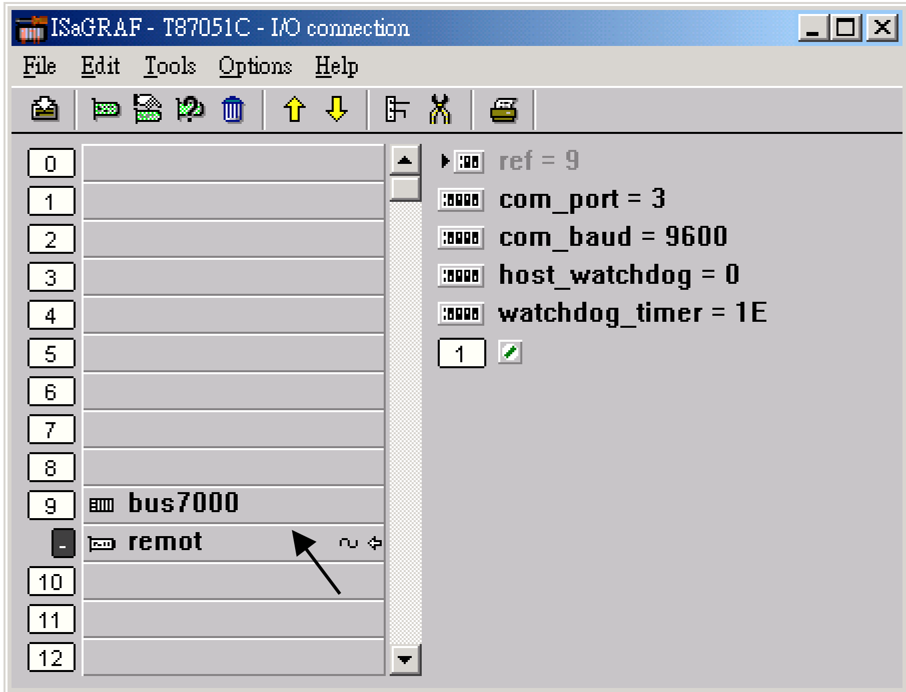
3.5: D/I Counters Built in The I-87xxx D/I Modules

87051, 87052, 87053, 87054, 87055, 87058 & 87063 have built-in low speed D/I counters associated with each D/I channel. The max counter speed of these modules is 100Hz. The counter value is ranging from 0 to 65535 and can be reset to 0.

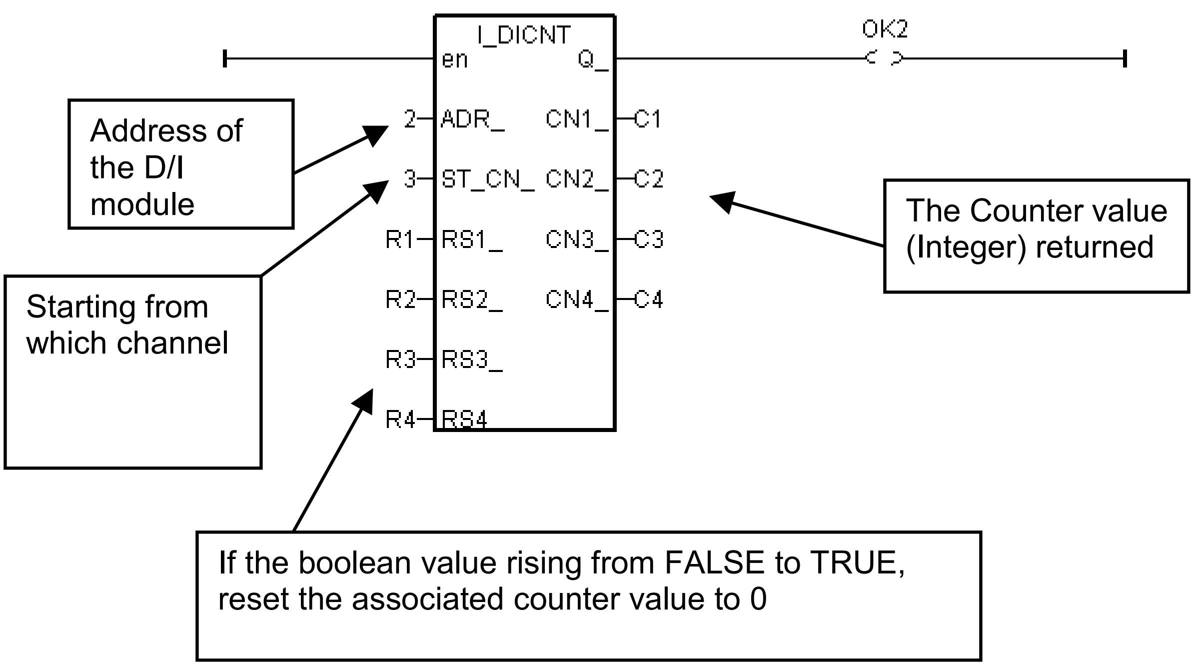
To use these D/I counters, connect these I/O modules with a last character – "C". For ex. "i_87052C" .



If the I-87xxx D/I Module is plugged in the 87K4, 87K5, 87K8 & 87K9 extension base module, or the I-7000 D/I module is used, Please refer to Chapter 6 to use “7000 utility” to set the appropriate address, baud rate , then connect “Bus7000” on the ”I/O connection” window.



Then using “I_DiCnt” block to get the “D/I Counter” value. Each “I_DiCnt” can get 4 counters.

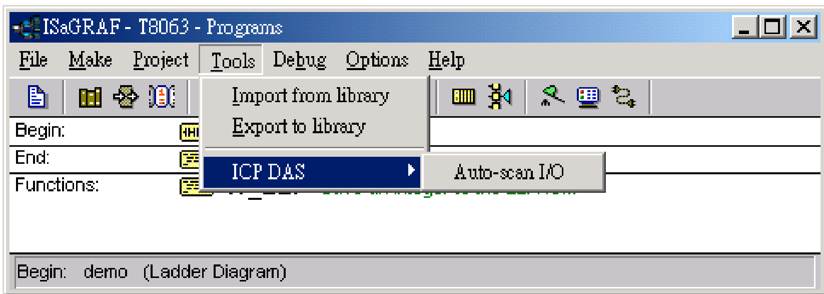


3.6: Auto-Scan I/O

Before you can use Auto-scan I/O utility, make sure the "ICP DAS Utilities For ISaGRAF" has been installed. (please refer to section 1.2)

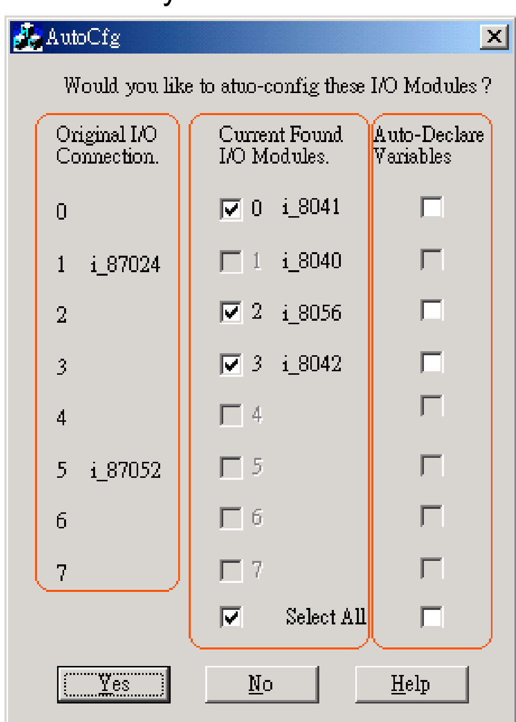
What is Auto-scan I/O :
It's a tool for ISaGRAF to easily configure your I/O connection and automatically declare variables for each I/O channel.

How to use ?
A. Open your ISaGRAF program.
B. Click on "Tools/ICP DAS/Auto-scan I/O" to run Auto-scan.



C. The Auto-scan I/O is divided into three area.

Original I/O Connection shows the modules that already exist in your I/O connection at the first eight slots of your ISaGRAF project.
Current Found I/O Modules shows the I/O modules that detected in your controller (By RS232 or TCP/IP).
Auto-Declare Variables shows what modules that you want Auto-scan to automatically declare variables for you also.



D. In the “Current Found I/O Modules.” area:

The check box will be enable only when an I/O module is detected in the controller and the slot is **not used** by original I/O connection.

E. In the “Auto-Declare Variables”:
The check box can be enable only when one I/O module **is checked** in the current found area.

F. You can check the “Select All” to check all available boxes in the respective area.

What is necessary for Auto-scan I/O ?

A. Make sure the “Link setup” parameter is correct.

B. Plug in I/O boards first before your ISaGRAF can detect them.

Naming rules of automatically declared variables

Name format : Type_Slot_Channel

Type:
Digital Input : DI
Digital Output : DO
Analog Input : AI
Analog Output : AO

Slot : one digital slot number.
Channel : two digital channel number.

For ex. :
DI_0_02 , Digital Input channel at channel No.2 of slot 0.
AI_5_06 , Analog Input channel at channel No.6 of slot 5.
DO_2_12, Digital Output channel at channel No.12 of slot 2.
AO_1_03, Analog Output channel No. 3 of slot 1.

Note:
I-8xx7 & Wincon-8xx7 supports “Auto-Scan”, however I-7188EG/XG doesn’t supoort it.

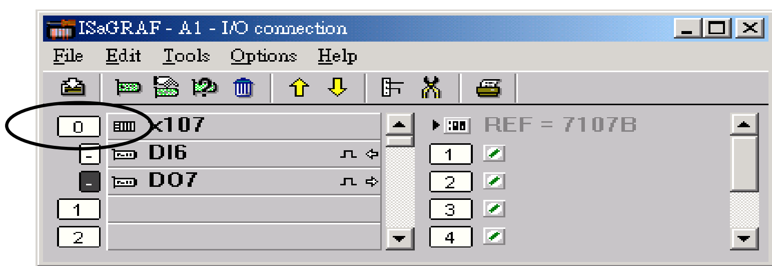
3.7: PWM Output

The scan time of the ISaGRAF controller depends on the ISaGRAF program and the hardware driver. For normal usage, the scan time is about 5 to 40 ms. It may go up to 100 ms sometime when the user's ISaGRAF program is very complicated. It is not easy to generate a precise periodic pulse output because the scan time of ISaGRAF is always varying, for example, a square curve of 2 ms OFF & then 1 ms ON. To achieve this kind of application, ISaGRAF provide PWM output functions.

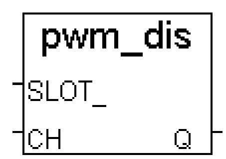
To use PWM output (Pulse Width Modulation) in **I-8417/8817/8437/8837**, please update the driver to version of **2.43** or higher. Only parallel Output boards are supported, not for serial boards. The following output boards are available with the PWM function.
I-8037, 8041, 8042, 8054, 8055, 8056, 8057, 8060, 8063, 8064, 8065, 8066, 8068, 8069

To support PWM function in **I-7188EG**, please update the driver to version of **1.35** or higher, while **1.32** or higer for **I-7188XG**
Only the Xxxx boards with digital output channels are available with PWM function.

Note:
1. Max 8 digital outputs can call PWM_en, PWM_en2, pwm_ON & pwm_OFF at the same time.
2. I-7188EG/XG must connect the Xxxx board at slot 0, or the PWM function will not work.



The below functions are for PWM output.



PWM_dis Disable PWM output

Parameters:

SLOT_	integer	Which slot ? 0 ~ 7 for I-8xx7, only 0 for I-7188EG/XG.
CH_	integer	Which channel ? 1 ~ 32.

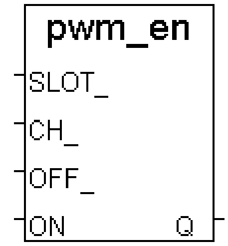
Return:

Q_	boolean	TRUE: Ok . FALSE: wrong input parameters, too many PWM outputs been enable, or the associate output channel is not found.

Note:
1. After calling PWM_dis, the associate output will then be controlled by the ISaGRAF cycle engine.
2.Max 8 output channels can call PWM_en, PWM_en2, pwm_ON, pwm_OFF at one controller.

Example: demo_50

PWM_en Enable PWM to output until PWM_dis is called



Parameters:

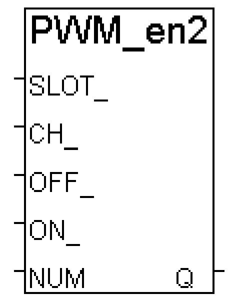
SLOT_	integer	Which slot ? 0 ~ 7 for I-8xx7, only 0 for I-7188EG/XG.
CH_	integer	Which channel ? 1 ~ 32.
OFF_	integer	Off time, 0 ~ 32,767, unit is ms. If set as 0, it means OFF_ time is 1 ms.
ON_	integer	On time, 0 ~ 32,767, unit is ms. If set as 0, it means ON_ time is 1 ms.

Return:

Q_	boolean	TRUE: Ok . FALSE: wrong input parameters, too many PWM outputs been enable, or the associate output channel is not found.

Example: demo_50

PWM_en2 Enable PWM to output a given number of pulse



Parameters:

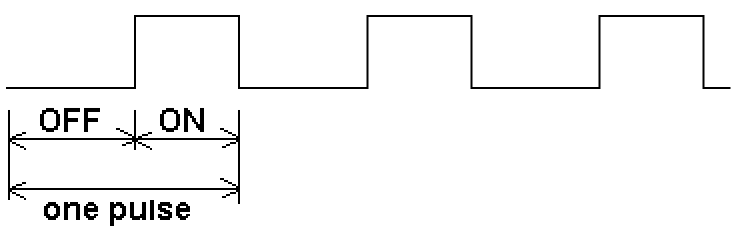
SLOT_	integer	Which slot ? 0 ~ 7 for I-8xx7, only 0 for I-7188EG/XG.
CH_	integer	Which channel ? 1 ~ 32.
OFF_	integer	Off time, 0 ~ 32,767, unit is ms. If set as 0, it means OFF_ time is 1 ms.
ON_	integer	On time, 0 ~ 32,767, unit is ms. If set as 0, it means ON_ time is 1 ms.
NUM_	integer	number of pulse to output, 1 - 2,147,483,647

Return:

Q_	boolean	TRUE: Ok . FALSE: wrong input parameters, too many PWM outputs been enable, or the associate output channel is not found.

Example: demo_55

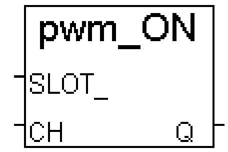
PWM output curve:



Note:

1. Every time the PWM_en or PWM_en2 is called, it will reset its internal tick to 0, and re-start ticking to OFF, ON, OFF, ON, ...
2. If the given number of pulse of pwm_en2 is reached, it will stop & disable PWM auomatically (Calling PWM_dis for pwm_en2 is not necessary).
3. PWM_sts can be used to test if pwm_en2 reaches its given number of pulse or not.
4. Max 8 output channels can call PWM_en, PWM_en2, pwm_ON, pwm_OFF at one controller.
5. Do not enable the channel that is already enable. Please disable it first.

pwm_ON Set parallel D/O to TRUE immediately



Parameters:

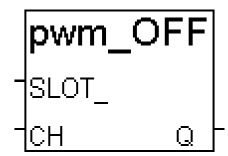
SLOT_	integer	Which slot ? 0 ~ 7 for I-8xx7, only 0 for I-7188EG/XG.
CH_	integer	Which channel ? 1 ~ 32.

Return:

Q_	boolean	TRUE: Ok . FALSE: wrong input parameters, too many PWM outputs been enable, or the associate output channel is not found.

Example: demo_55

pwm_OFF Set parallel D/O to FALSE immediately



Parameters:

SLOT_	integer	Which slot ? 0 ~ 7 for I-8xx7, only 0 for I-7188EG/XG.
CH_	integer	Which channel ? 1 ~ 32.

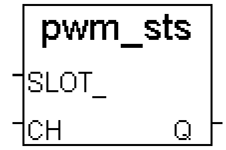
Return:

Q_	boolean	TRUE: Ok . FALSE: wrong input parameters, too many PWM outputs been enable, or the associate output channel is not found.

Example: demo_55

Note:
1. Max 8 output channels can call PWM_en, PWM_en2, pwm_ON, pwm_OFF at one controller.
2. pwm_ON will set the associate parallel D/O to TRUE immediately.
3. pwm_OFF will set the associate parallel D/O to FALSE immediately.
4. If users wish to enable one D/O as PWM output by PWM_en or PWM_en2 after pwm_ON & pwm_OFF has been called, please disable it first by PWM_dis, then call PWM_en or PWM_en2.

PWM_sts Get PWM status



Parameters:

SLOT_	integer	Which slot ? 0 ~ 7 for I-8xx7, only 0 for I-7188EG/XG.
CH_	integer	Which channel ? 1 ~ 32.

Return:

Q_	boolean	TRUE: this channel has been enable FALSE: disable (for pwm_en2 been called, it means the given pulse number is reached).

Note:
1. Max 8 output channels can call PWM_en, PWM_en2, pwm_ON, pwm_OFF at one controller.
2. This function can be used to test if "PWM_en2" reachs its given pulse number or not.

Example: demo_55

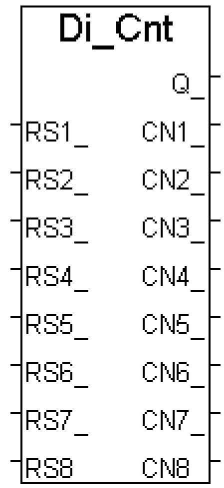
3.8: Counters Built in Parallel D/I Boards

I-8417/8817/8437/8837 supports D/I counters since its driver version of 2.43. Only parallel input boards plug **at slot 0** are supported, not for serial boards. The following input boards are available with D/I counters.

I-8040, 8042, 8051, 8052, 8053, 8054, 8055, 8058, 8063, 8077

I-7188EG supports D/I counters since its driver version of 1.35 while I-7188XG since 1.32. Only the X??? boards with digital input channels are available with D/I counters.

The max channel of parallel D/I counter available in one controller is up to 8. And the max frequency of counter input is up to 500 Hz with minimum pulse width of 1 ms.



The below c function block is for getting/reset D/I counters at slot 0.

Parameters:

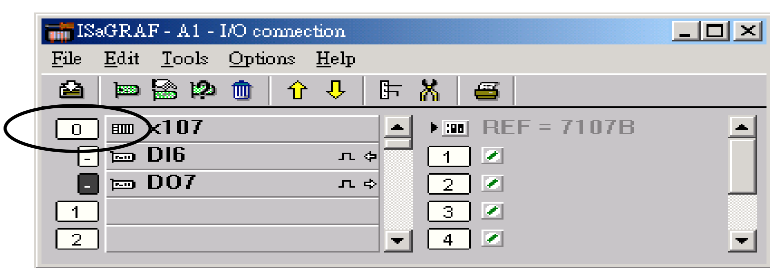
RS1_ ~ RS8_	boolean	Reset the associated D/I counter when rising from False to True

Return:

Q_	boolean	work ok. : TRUE. If Q_ is FALSE , it means "No parallel D/I module found at slot 0 "
CN1_ ~ CN8_	integer	DI Counter value of channel No. 1 to 8. Valid value is ranging from 0 to 2,147,483,647. If value is over 2,147,483,647, it restarts at 0.

Note:

Only Parallel D/I board plug in slot 0 support "Di_Cnt", not for other slots.
Only the first 8 D/I channel support "Di_Cnt".
I-7188EG/XG must connect the X??? board at slot 0, or the "Di_Cnt" will not work.



Demo: Please refer to I-8417/8817/8437/8837's demo_52 & demo_53.

3.10: Stepping Output Built in Parallel D/O Boards

I-8417/8817/8437/8837 supports D/O Stepping output since its driver version of 2.37. Only below parallel output boards are supported, not for serial boards.

I-8037, 8041, 8042, 8054, 8055, 8056, 8057, 8060, 8063, 8064, 8065, 8066, 8068, 8069

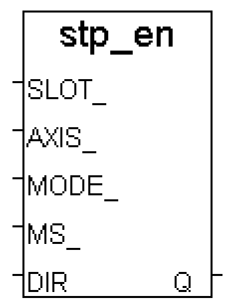
The max axis number of stepping output is 2 for one controller.
Each axis is drived by 4 digital output channels. Please connect them as below.

Axis 1:	**A --- Ch.1**	**B --- Ch.2**	**A_ --- Ch.3**	**B_ --- Ch.4**
Axis 2:	**A --- Ch.5**	**B --- Ch.6**	**A_ --- Ch.7**	**B_ --- Ch.8**

Note:
Do not use stepping output & PWM output at the same output channel.
The I-7188EG/XG & W-8xx7 doesn't support stepping output.

Available functions:

STP_en Enable stepping output to output



Parameters:

SLOT_	integer	Which slot ? 0 - 7
AXIS_	Integer	Which axis ? 1 - 2 AXIS 1: (Ch.1 - Ch.4), AXIS 2: (Ch.5 - Ch.8)
MODE_	Integer	Which mode ? 1 – 3, (A, B, A_, B_) = Mode 1: (1, 0, 0, 0)--> (0, 1, 0, 0)--> (0, 0, 1, 0)--> (0, 0, 0, 1) Mode 2: (1, 1, 0, 0)--> (0, 1, 1, 0)--> (0, 0, 1, 1)--> (1, 0, 0, 1) Mode 3: (1, 0, 0, 0)--> (1, 1, 0, 0)--> (0, 1, 0, 0)--> (0, 1, 1, 0) --> (0, 0, 1, 0) --> (0, 0, 1, 1) --> (0, 0, 0, 1) --> (1, 0, 0, 1)
MS_	Integer	Step interval time, 1 - 1000, unit is ms. For ex. set as 5 means running 200 steps/sec.
DIR_	Boolean	True: positive direction, False: opposite direction

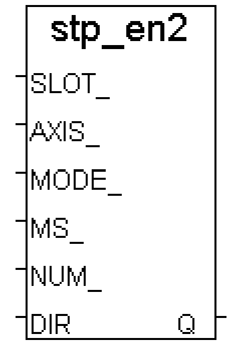
return:

Q_	Boolean	TRUE: Ok , FALSE: wrong input parameters, or the associate output channel is not found.

Example: Please refer to I-8417/8817/8437/8837's demo_58 & demo_59.

Note:
1. The way to stop "STP_en" is - call "STP_dis" function
2. If "STP_en", "STP_en2", "STP_sts" & "STP_dis" is not found, please download "**ICP DAS Utilities For ISaGRAF.zip**" from http://www.icpdas.com/products/8000/isagraf.htm and click on setup to re-install them to your ISaGRAF.

STP_en2 Enable stepping output to output some given steps



Parameters:

SLOT_ integer Which slot ? 0 - 7
AXIS_ Integer Which axis ? 1 - 2
AXIS 1: (Ch.1 - Ch.4), AXIS 2: (Ch.5 - Ch.8)
MODE_ Integer Which mode ? 1 – 3, (A, B, A_, B_) =
Mode 1: (1, 0, 0, 0)--> (0, 1, 0, 0)--> (0, 0, 1, 0)--> (0, 0, 0, 1)
Mode 2: (1, 1, 0, 0)--> (0, 1, 1, 0)--> (0, 0, 1, 1)--> (1, 0, 0, 1)
Mode 3: (1, 0, 0, 0)--> (1, 1, 0, 0)--> (0, 1, 0, 0)--> (0, 1, 1, 0)--> (0, 0, 1, 0) --> (0, 0, 1, 1) --> (0, 0, 0, 1) --> (1, 0, 0, 1)
MS_ Integer Step interval time, 1 - 1000, unit is ms.
For ex. set as 5 means running 200 steps/sec.
NUM_ Integer How many steps ? 0 - 2,147,483,647
DIR_ Boolean True: positive direction, False: opposite direction

return:

Q_ Boolean TRUE: Ok , FALSE: wrong input parameters,
or the associate output channel is not found.

Note:
User may use the "STP_sts" function to test “STP_en2” is finished or not.
2. The ways to stop "STP_en2" are
- call "STP_dis" function
- wait until it is finished

STP_sts Get stepping output status

Parameters:

AXIS_ Integer Which axis ? 1 - 2
AXIS 1: (Ch.1 - Ch.4), AXIS 2: (Ch.5 - Ch.8)

return:

Q_ Boolean TRUE: still enable , FALSE: disable (for stp_en2 been called, it means the given step number is reached).

STP_dis Disable stepping output

Parameters:

AXIS_ Integer Which axis ? 1 - 2
AXIS 1: (Ch.1 - Ch.4), AXIS 2: (Ch.5 - Ch.8)

return:

Q_ Boolean TRUE: Ok , FALSE: wrong input parameters,
or the associate output channel is not found.

Example: Please refer to I-8417/8817/8437/8837’s demo_58 & demo_59.

Chapter 4: Linking Controllers To An HMI Program

This chapter details how to make data from the I-8xx7, I-7188EG/XG & W-8xx7 controller system available to Human Machine Interface (HMI) programs. This is a powerful feature that allows customers to create their own custom HMI programs and link them to the controller system.

After you realize the material described in section 4.1, if you would like to use the I-8xx7, I-7188EG/XG controller as a **Modbus or Modbus TCP/IP I/O**, you may refer to section 4.3. Additionally there are "touch screen" monitors provided by ICP DAS that support the "Modbus" protocol, and these touch screen monitors can also access data from an I-8xx7 controller . Section 4.4 illustrates how to link a "Touch 510" monitor to an I-8xx7 controller system.

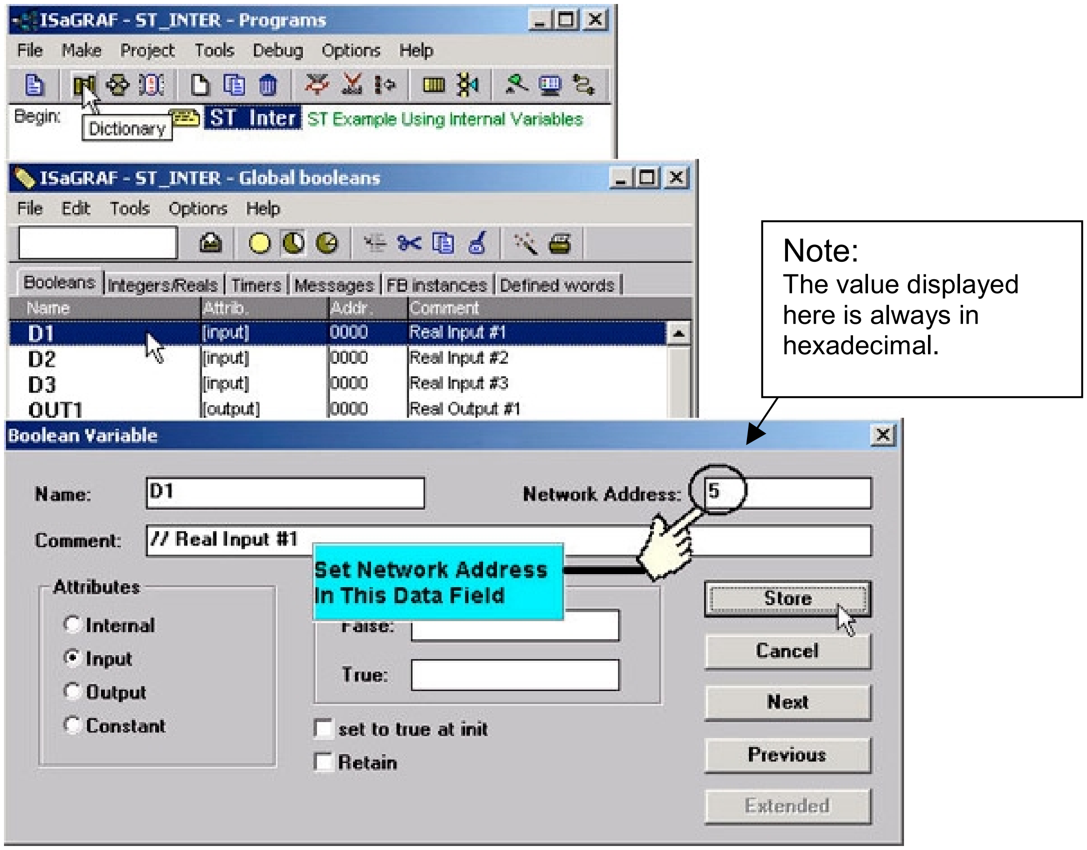
4.1: Declaring Variable Addresses For Network Access

To make data from an I-8xx7, I-7188EG/XG & W-8xx7 controller system available to other software programs or HMI devices, you must first declare the variable with a "Network Address". The variable must be declared with a network address number that is in the "Modbus" format. Other software programs or HMI devices will access the controller information through these network addresses.

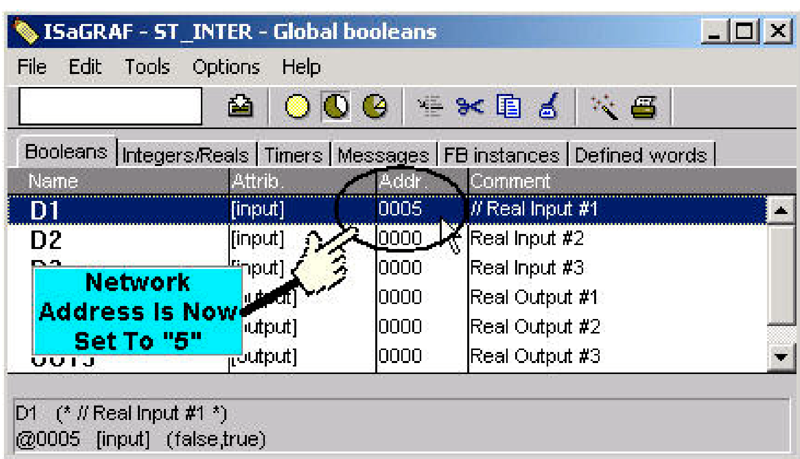
There are two methods available to declare a variable for network address access. The first method is described below. Open an "ISaGRAF Programs" windows and click on the "Dictionary" icon, then double click on the variable to assign a network address number.

Note:
1. The valid network addresses for an **I-8417/8817/8437/8837 & I-7188EG/XG** controller system is from 1 to FFF in hexadecimal (**1 ~ 4095**). Network address **5001 to 8072** is for word and integer arrays, please refer to Section 4.5.

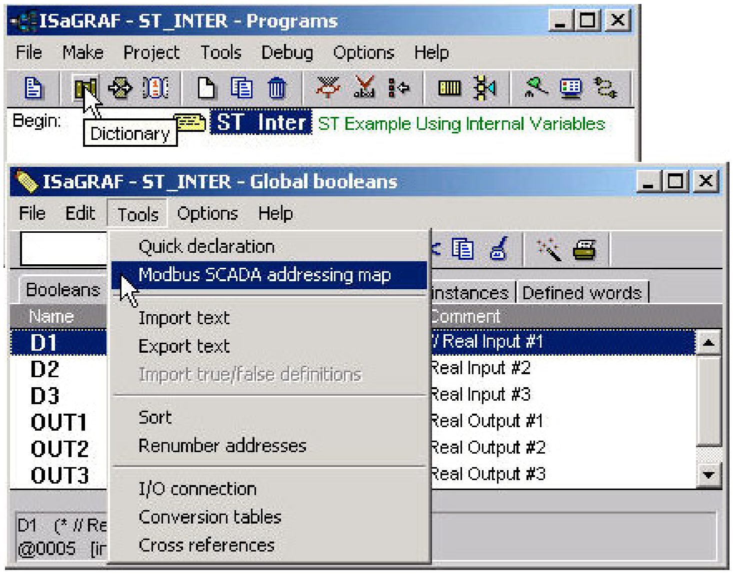
2. The valid network addresses for an **W-8037/8337/8737** controller system is from 1 to 1FFF in hexadecimal (**1 ~ 8191**). Network address **10,001 to 19,216** is for word and integer arrays, please refer to Section 4.5.



When you click on the "Store" button you will see that "ISaGRAF Global Variables" window will now be updated with the new network address for the variable.

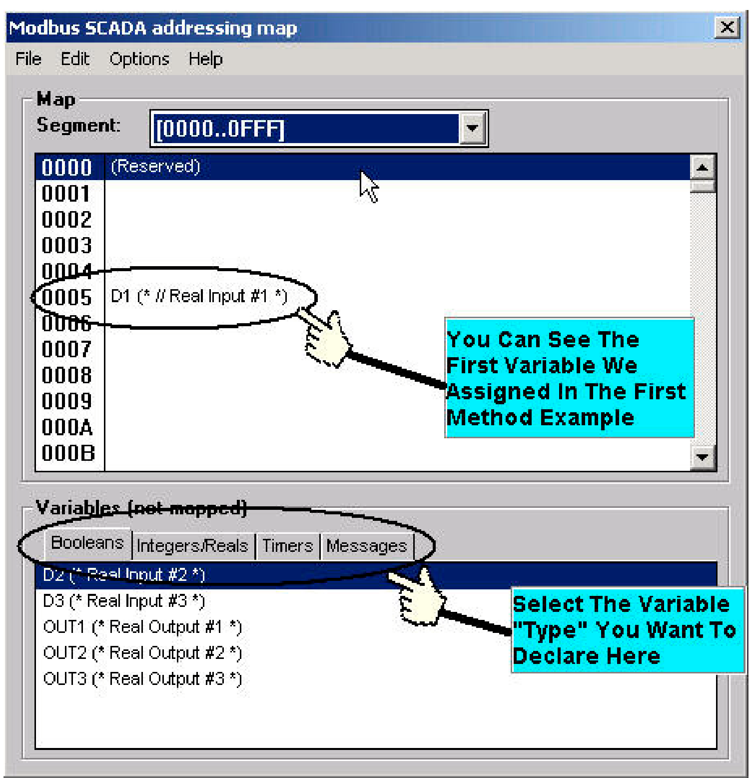


The second method for assigning network addresses to variables requires that you declare the variables BEFORE you assign them. This method allows you to assign numerous network address variables before you link them to an ISaGRAF program.

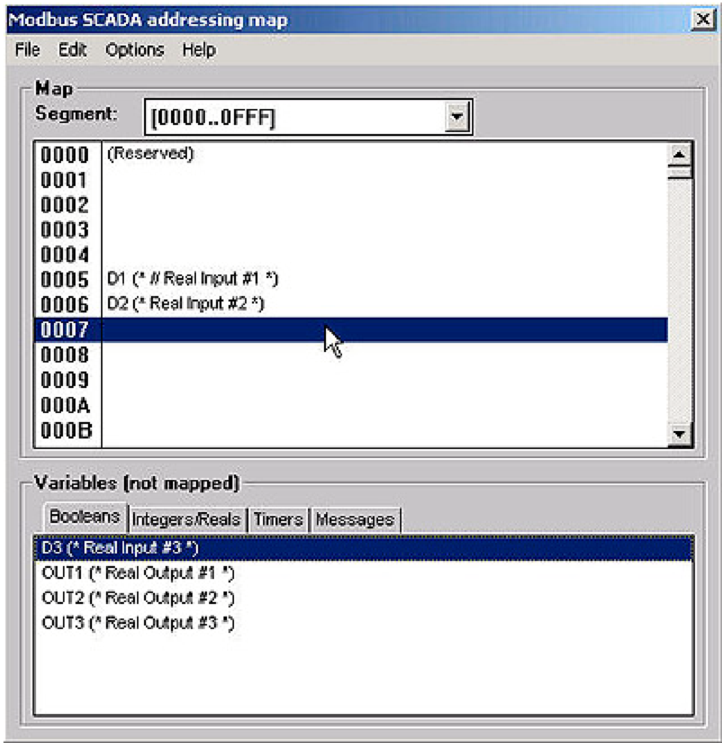


When you click on "Modbus SCADA Addressing Map" (SCADA is an industrial process control acronym that stands for "Supervisory Control And Data Acquisition") the "Modbus SCADA Addressing Map" window will open.

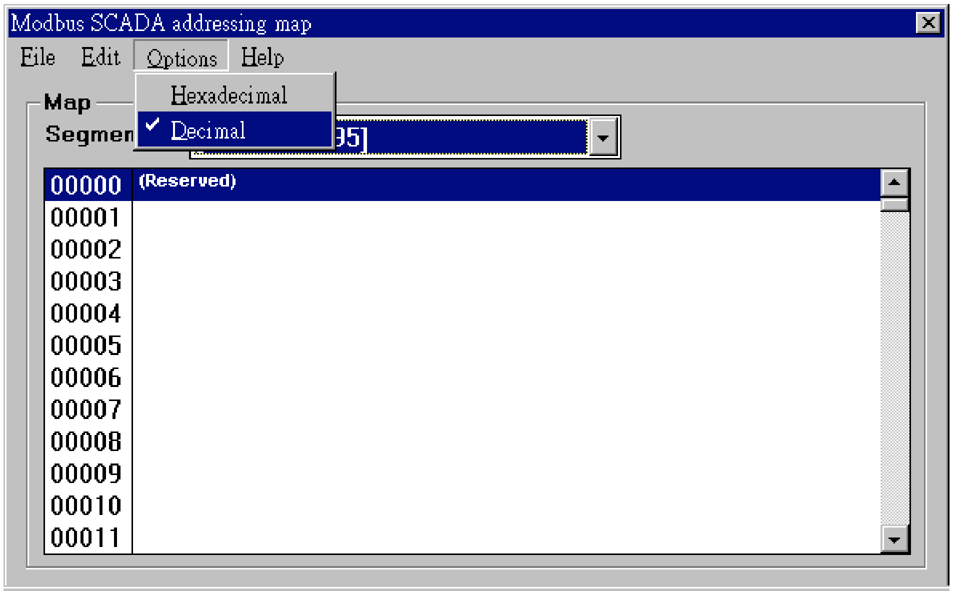
Note that one of the variables (D1) is already assigned from our previous network-addressing example. You will note that the other variables that are not yet mapped are displayed in the lower portion under the "Variables (Not Mapped)" portion of the "Modbus SCADA Addressing Map" window.



To assign the other variable address click on an unassigned "Map Segment" number, and then double click on the variable you want to assign to the address and the variable will automatically assign itself to the "Map Segment".



For human’s thinking method, network address represented in hexadecimal format is inconvenient and it increases the chance to make mistake. Therefore, it’s better to change it to be represented in decimal format. To do that is as following.



IMPORTANT NOTE REGARDING MODBUS NETWORK ADDRESSING

The Modbus network address definition scheme is sometimes different between HMI devices and other software programs. The difference is typically that the other programs may assign a network address number that is one (1) less than that of the I-8xx7, I-7188EG/X & W-8xx7 controller system.

HMI or devices such as Iconics, Citech, Wizcon, Kepware’s OPC server, Intellution’s "iFix", Wonderware’s "Intouch", National Instruments "Labview", and ICP DAS’s Touch 506L, Touch 506S and Touch 510T do have the exact same addressing scheme as the I-8xx7, I-7188EG/X & W-8xx7 controller system.

Known addressing disparities include "LabLink" and "Hitech" HMI software programs and devices. If you are assigning a network address of "B" (hexadecimal) of these products the I-8xx7 network address should be set to "C". A network address of "2" should be associated with a network address of "3" in the ISaGRAF controller system.

Another things mistaked very often is the first digit of the network address of many HMI softwares resprent the data type and Read/Write authority not one part of the network address. For example, the network address relation between “iFix” and ISaGRAF is as below.

iFix(Decimal)	I-8xx7 (Decimal)
00001 (R/W Boolean)	1
…	…
10010 (Read Boolean)	10
…	…
31000(Read Word)	1000
…	…
42101(R/W Word)	2101

ICP DAS has not been able to test every possible HMI software program or hardware device that has Modbus addressing capability. If you are trying to connect your HMI software program or hardware device with Modbus to an I-8xx7, I-7188EG/X & W-8xx7 controller system, **REMEMBER** that you **may** have to offset the Modus addressing by 1 between these products so they will properly communicate with each other.

Developers who design and write their own software interface programs using Microsoft's Visual Basic or Visual C++ programming language should refer to Chapter 5 of this manual for more information on how to interface the Modbus protocol to these programming languages.

NOTE:
While talking to the I-8xx7, I-7188EG/X & W-8xx7, **ONE** Modbus frame cannot request more than **255 bits**, and also cannot request more than **120 words**. It should be divided into 2 or more requests to achieve it.

4.2:Read/Write Word, Long Word & Float through Modbus

Modbus protocol provides function 3 for reading multiple words while function 6 and 16 to write words. Please refer to Chapter 5 for more information about the protocol.

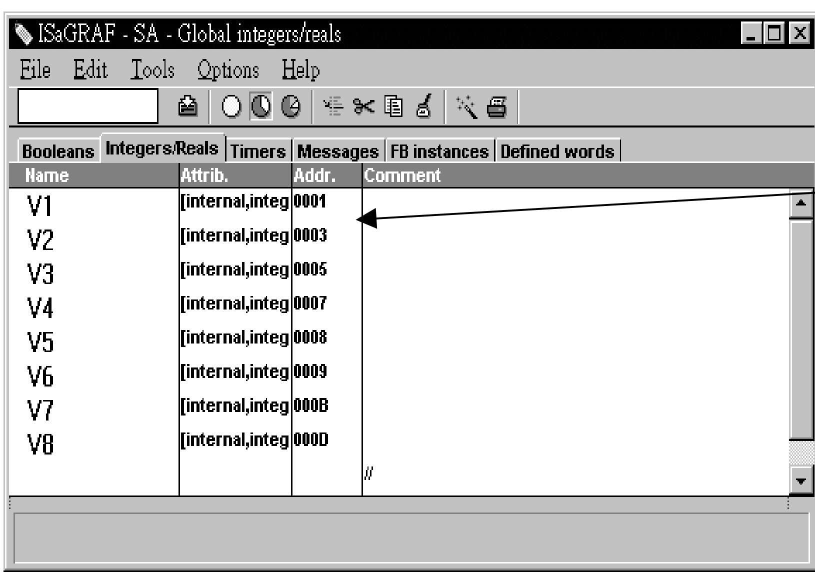
The **word** defined in the Modbus protocol of I-8xx7, I-7188EG/X & W-8xx7 controllers is like a signed short integer, which occupies 2 bytes and range from –32,768 (8000 in hexa.) to +32,767 (7FFF in hexa.). It is normally used to describe the behavior of analog I/O channels. For examples, the I-87017 I/O board (please refer to section 3.2)

I-87017 :

Range ID (hexadecimal)	Electrical Range	Values on the channel (decimal)		
		-32768	0	+32767
8 (default)	± 10V	- 10V	0V	+ 10V
9	± 5V	- 5V	0V	+ 5V
A	± 1V	- 1V	0V	+ 1V
B	± 500mV	- 500mV	0mV	+ 500mV
C	± 150mV	- 150mV	0mV	+ 150mV
D	± 20mA	- 20mA	0mA	+ 20mA

The **long word** defined in the Modbus protocol of I-8xx7, I-7188EG/X & W-8xx7 controllers is like a signed long integer, which occupies 4 bytes and range from -2,147,483,648 (8000 0000 in hexa.) to +2,147,483,647 (7FFF FFFF in hexa.). It is normally used to describe the value of internal integer variables declared on ISaGRAF workbench.

All integer variables declared on ISaGRAF are signed 32-bit format however the integer variable, which assigned with a network address will only, occupies 1 word (2 bytes) in the Mudbus transportation format. Since a long word occupies 2 words (4 bytes), to Read/Write long word through Modbus, the network address assigned to the integer variable has to be followed as below.

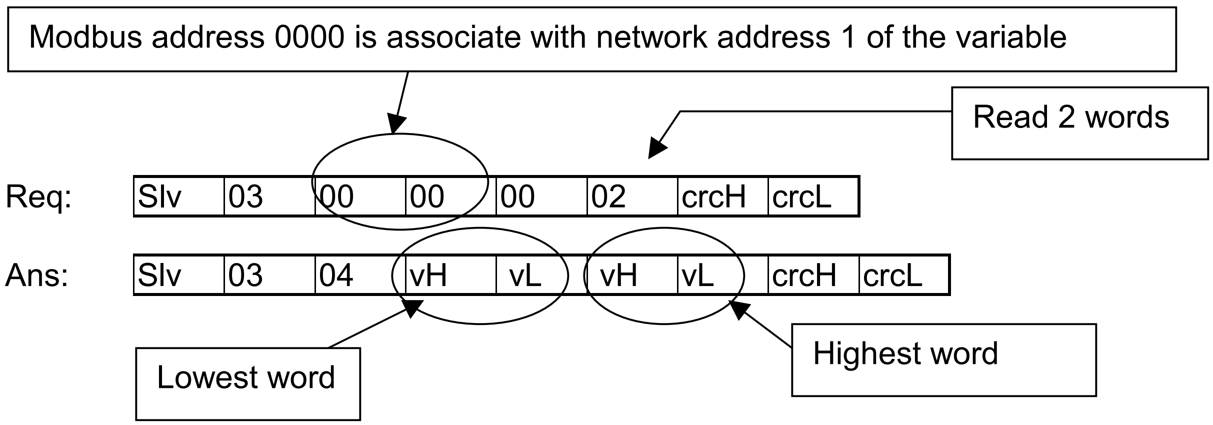


V1 is assigned to a network address "1".
If the network address "2" is not assigned to any other variable, V1 will occupy a long word (4 bytes) in the Modbus transportation formate.

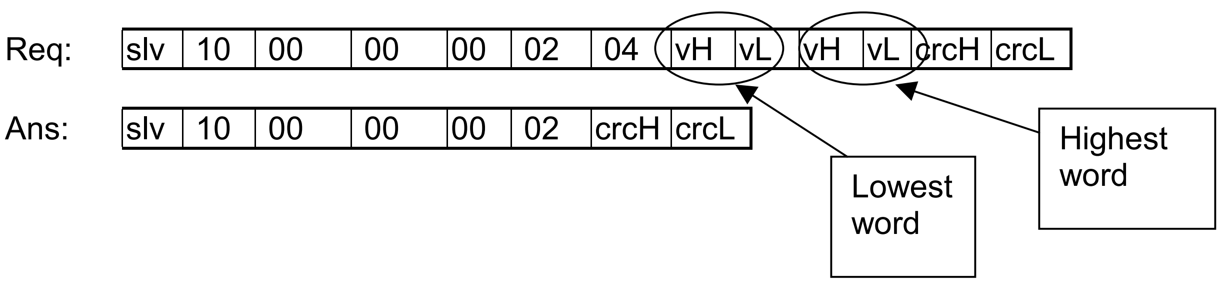
However if "2" is assigned to one another variable, V1 will only occupy one word (2 bytes) in the Modbus transportation format.

In this example, V1, V2, V3, V6, V7 and V8 will occupy 4 bytes however V4 and V5 only occupy 1 word (Lowest word) in the Modbus

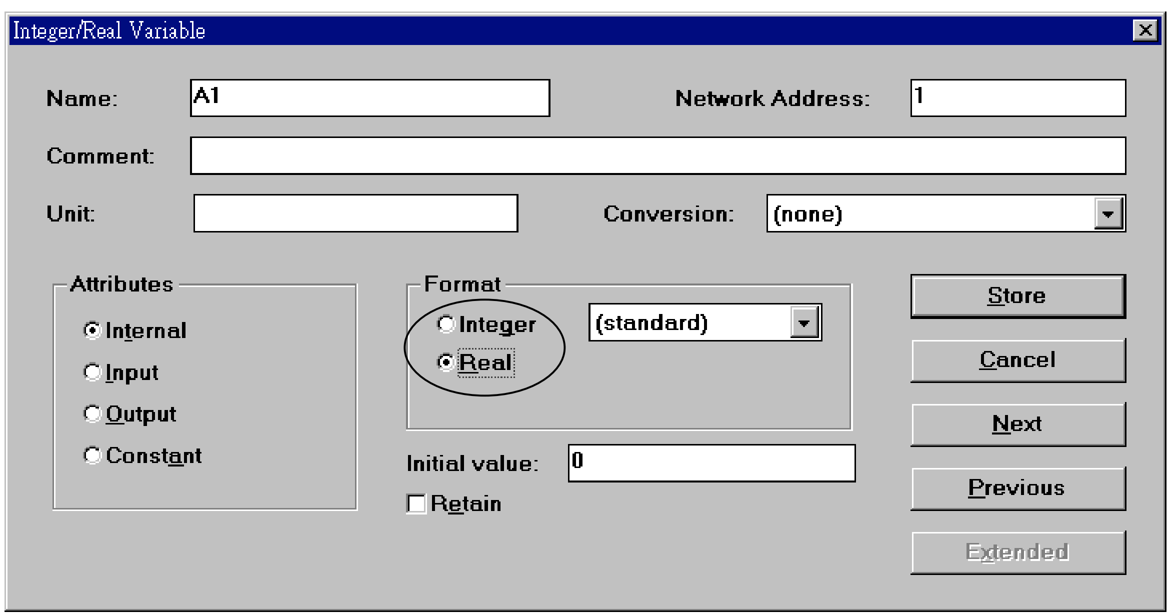
To read **long word** value of V1 is to read **2 words** by using modbus function 3 (please refer to section 5.1).



To write **long word** to V1 is to write **2 words** by using modbus function 16.



To read / write float (4 bytes) is very similar to read / write long word. The difference is the variable should be declared as “Real” type, and the next network address No. should not be assigned to any other variable.



There are much available HMI software on the market. You don’t need to care about the modbus protocol format. Just be careful to assign the correct network address on ISaGRAF.

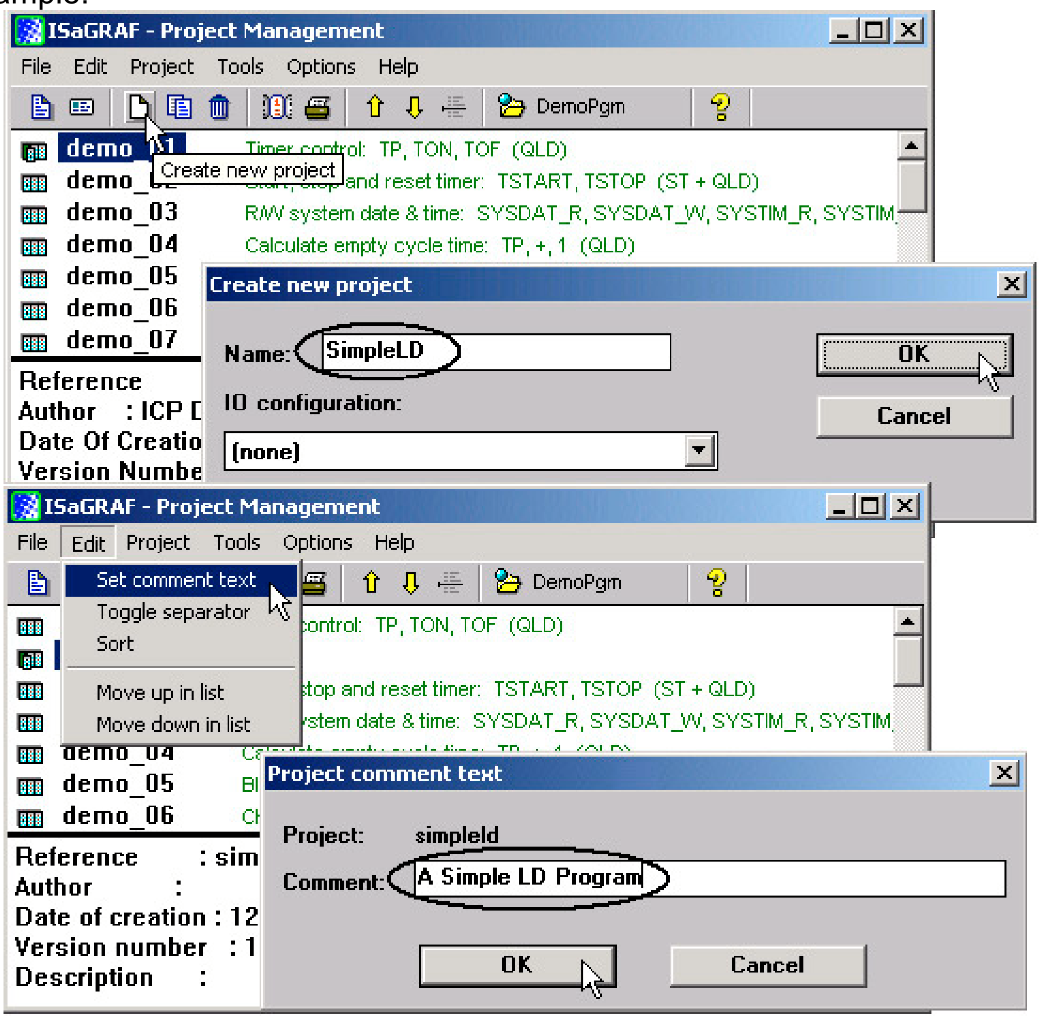
4.3: Using I-8xx7 As A Modbus I/O Or A Modbus TCP/IP I/O

There are some configurations that the HMI software gathers the I/O data from some called Modbus I/O modules. There I/O modules scan each input channels and refresh the output channels when need. Most of time there are no control logic inside these I/O modules, they are controlled by the HMI. To fit such kind of usage, the I-8417/8817/8437/8837 can be a Modbus I/O module, additionally the I-8437/8837 can be a Modbus TCP/IP I/O module. To do that, follow the following procedures (If you are not familiar with the ISaGRAF programming, recommended to review Chapter 2).

Create a new project

You may refer to section 2.1.1.2

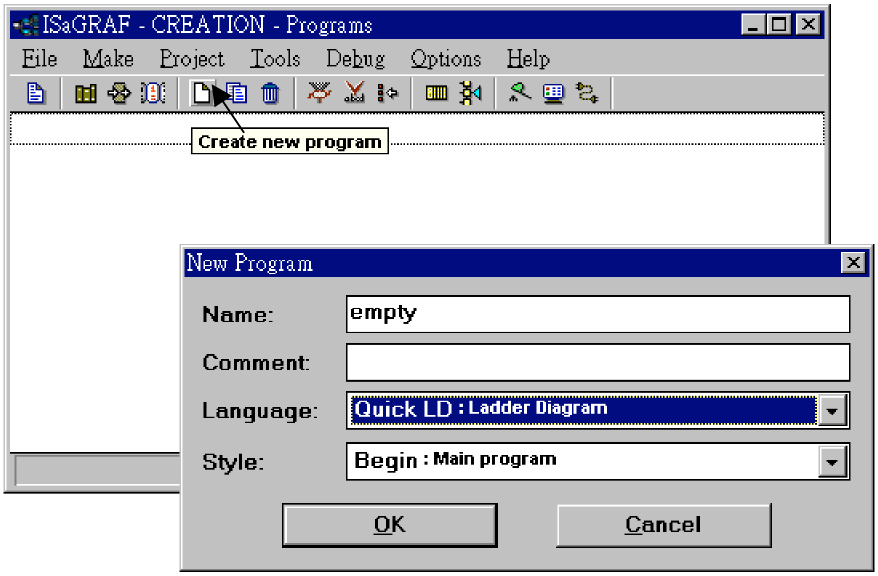
Example:



Create an empty program

No logic need.

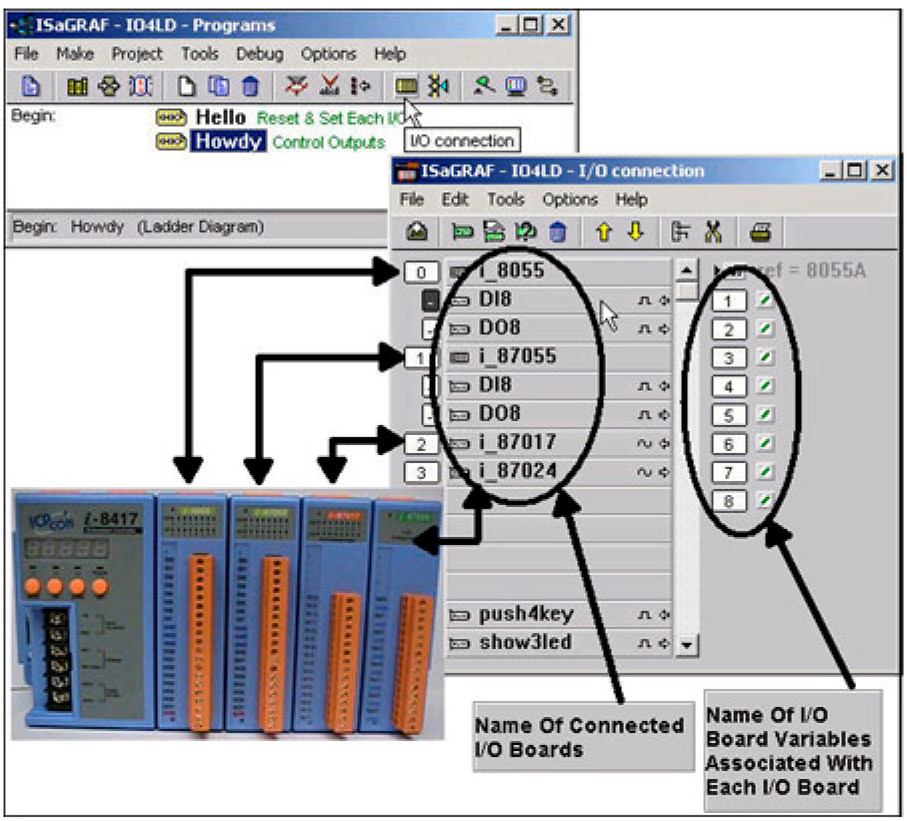
Example:



Connect I/O modules

You may refer to section 3.1

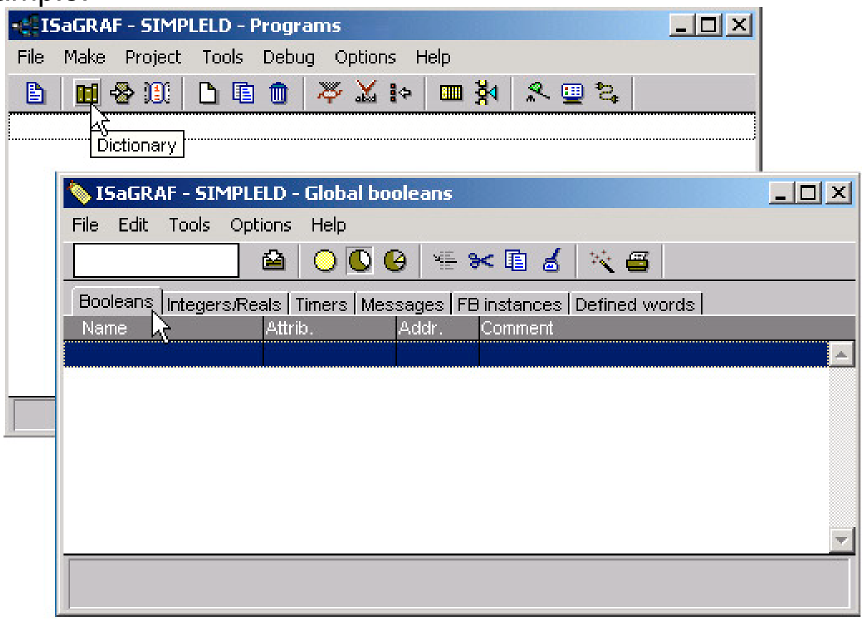
Example:

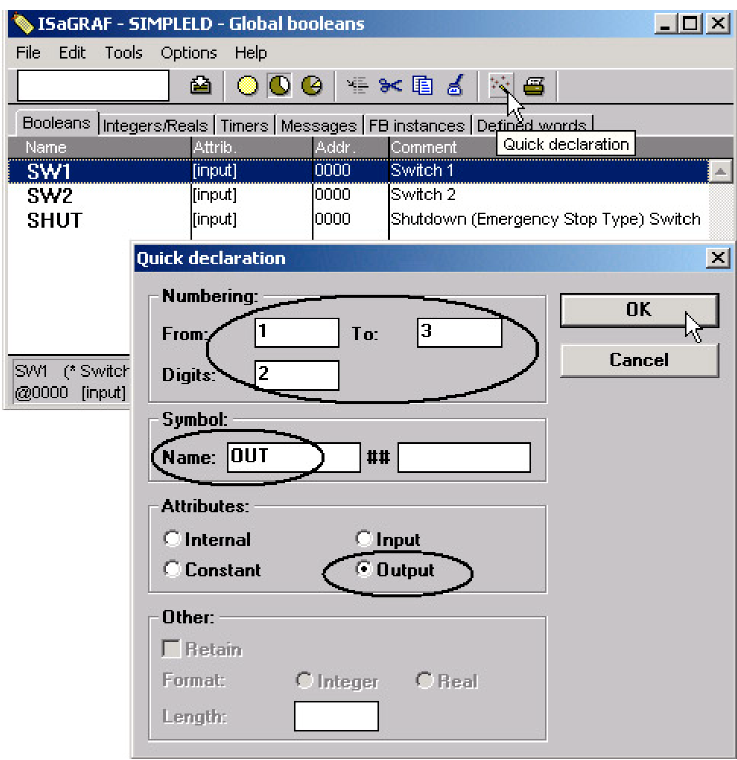


Declare Variables associated with the channels of connected I/O modules.

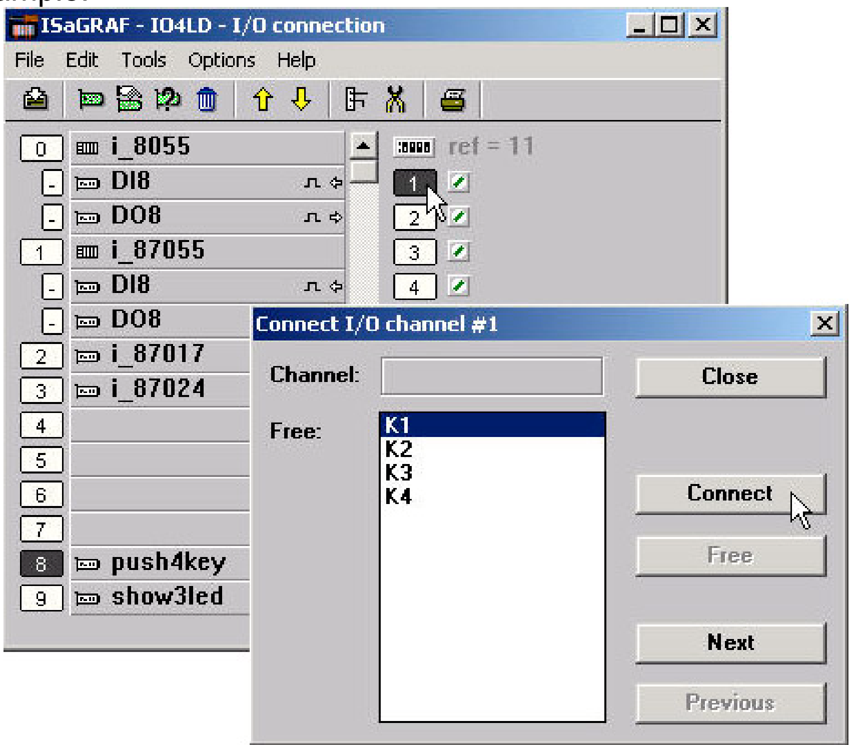
You may refer to section 2.1.1.3

Example:

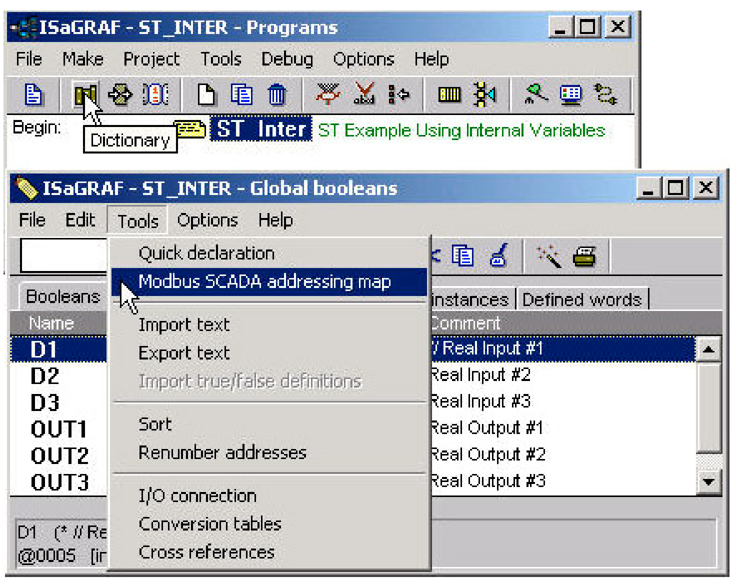




Link Variables to the associated channels of connected I/O modules.
You may refer to section 3.1.2
Example:



Assign the linked Variable a network address No.
You may refer to section 4.1
Example:



Compile & download the project

You may refer to section 2.1.3 & 2.1.5

Note:
Make sure the Net ID is set to the proper No. (section 1.3.1) For I-8437/8837, make sure the IP and Mask address is well set (appendix B).

The HMI can access to I/O channels through the associated network address now!

4.4: Linking I-8xx7, I-7188EG/XG & W-8xx7 To Touch 500

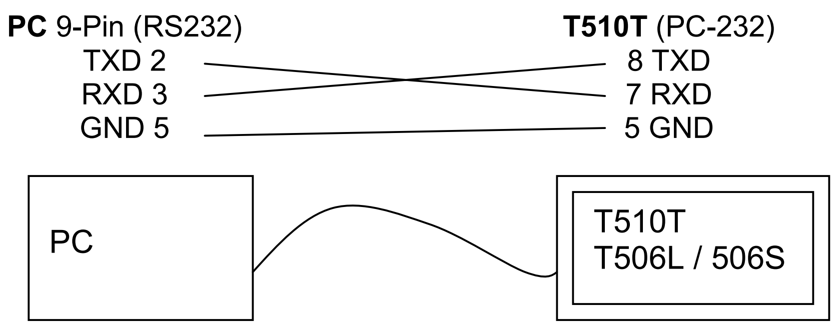
This section illustrates a demo program to link the I-8417 controller to a Touch 510T HMI.

Software Installation: EasyBuilder 500

Please download its **newly** toolkit & Manual at
http://www.icpdas.com/download/others/touch/touch.htm “**setup.zip**”
or run CD-ROM:\napdos\others\touch\500series\setup \“**setup.exe**” (V2.52 or later)

Note: Please always install it to “**c:\EB500**” (the default path)

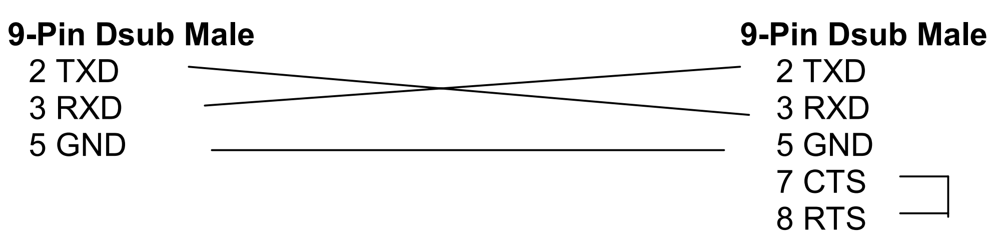
The cable to link PC to the Touch 506L/506S/510T has pin assignment as following. It can be used to download the designed MMI picture from the PC to the 506L/506S/510T.



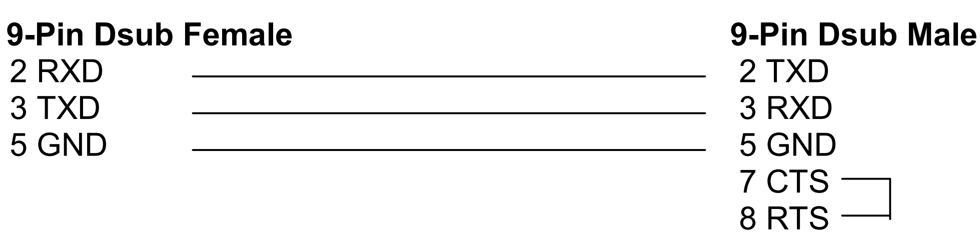
After the Touch 510T has been programmed a MMI picture, another cable should be used to link the Touch 510T to the I-8xx7, I-7188EG/XG & W-8xx7 controller.

Cable Pin Assignment:

I-8000 COM1 & I-7188 COM1 (RS232) | **Touch 506S/506L/510T (PLC 232)**



Wincon COM2 (RS232) | **Touch 506S/506L/510T (PLC 232)**



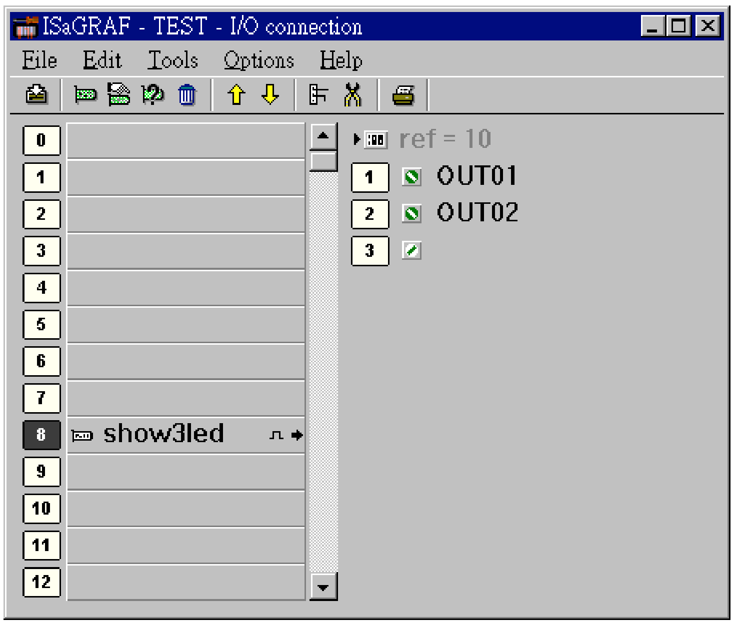
4.4.1: Program the I-8xx7, I-7188EG/XG & W-8xx7

To make data of the I-8xx7, I-7188EG/XG & W-8xx7 controller to be accessible to the Touch 510T, variables in the controller should be assigned a network address. Please refer to section 4.1, 4.2. If you are not familiar with the ISaGRAF programming, recommended to review Chapter 2.

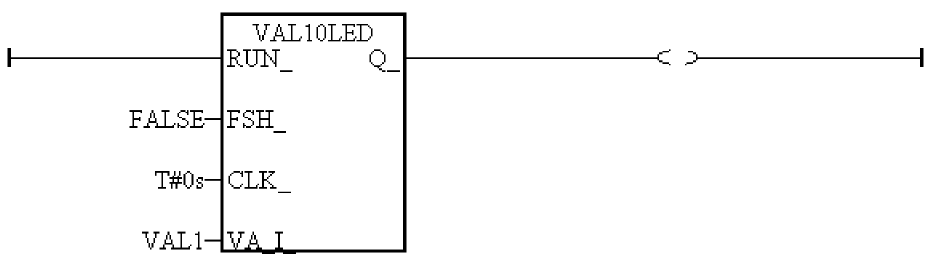
Variables used in this example.

Name	Type	Attribute	Network address	Others
OUT01	Boolean	Output	**0001**	-
OUT02	Boolean	Output	**0002**	-
VAL1	Integer	Internal	000A (10)	-

IO connection:



A simple LD program to show the "VAL1" to 7-segment LED:

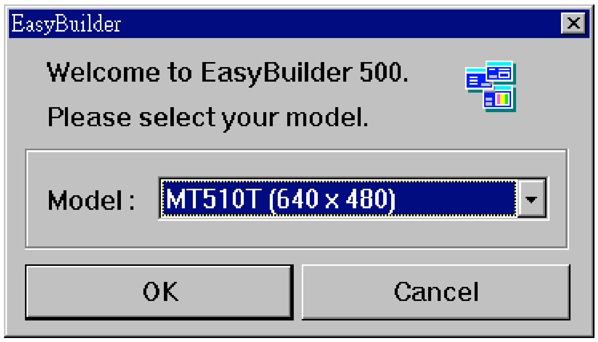


After you finish this project, compile and download it to the I-8xx7 controller.

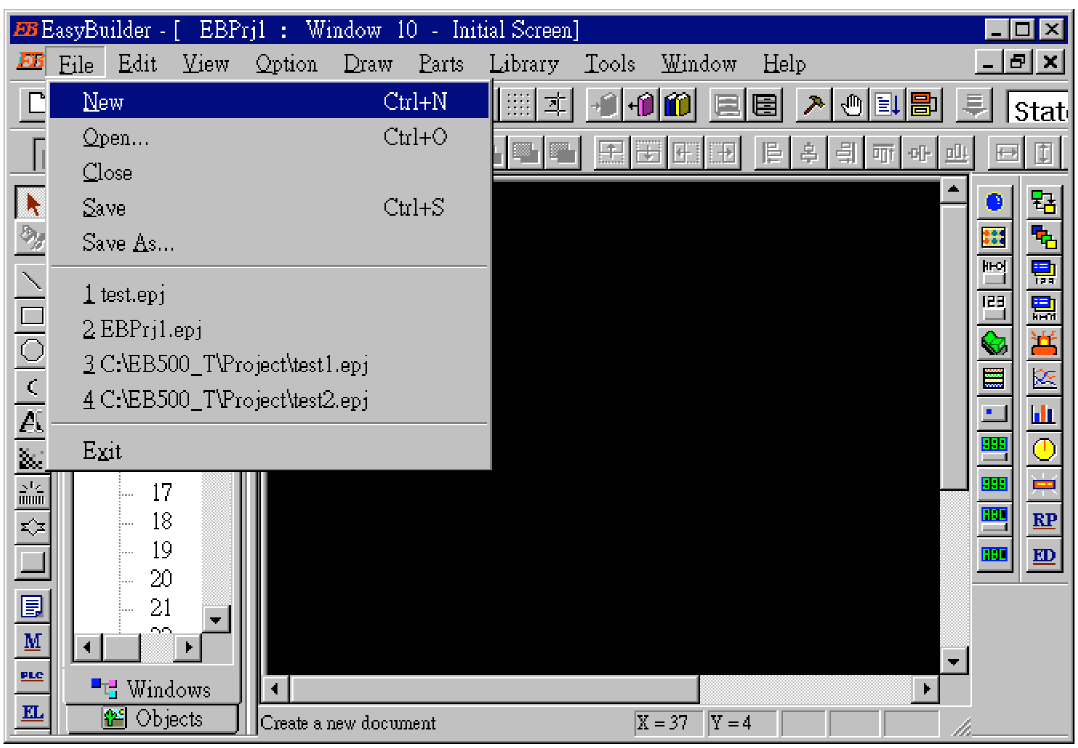
4.4.2: Program the Touch 510T

The “EasyBuilder 500” software can be used to designe many useful pictures for Touch 500 series. This section illustrates a simple example to program a Touch 510T. For more information about programming on the Touch series, please refer to the user manual which is provided with the “Touch” series hardware.

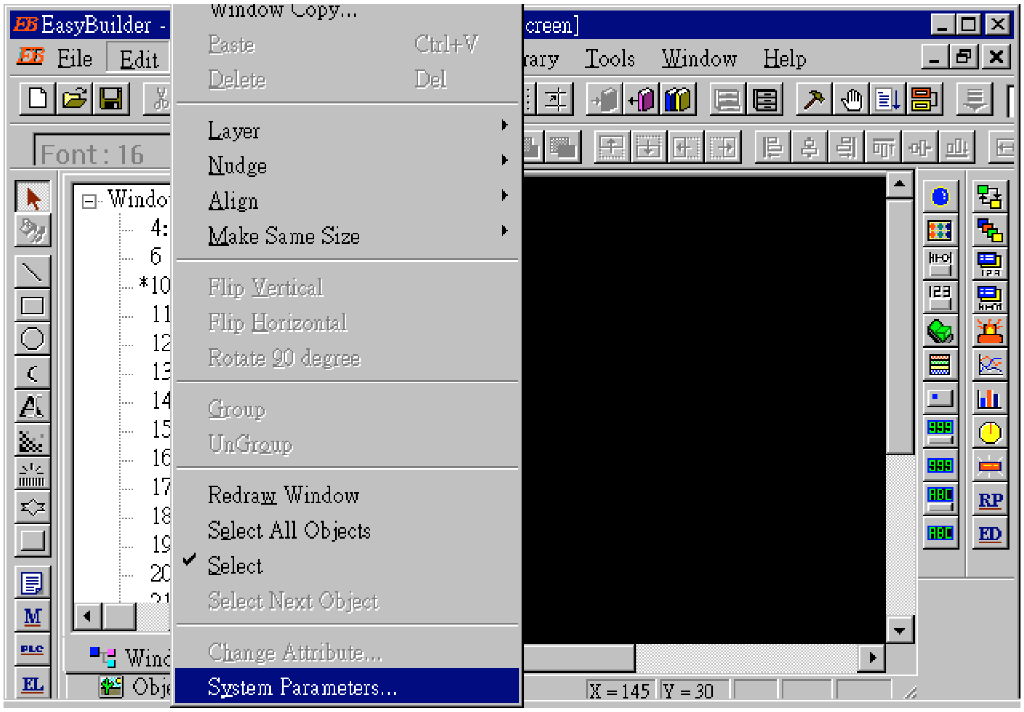
Click on the Windows "Start" button, then click on the "Program" button, then click on the "EasyBuilder" – “EasyBuilder 500” button. The following window will be displayed. Select the proper model for your application.



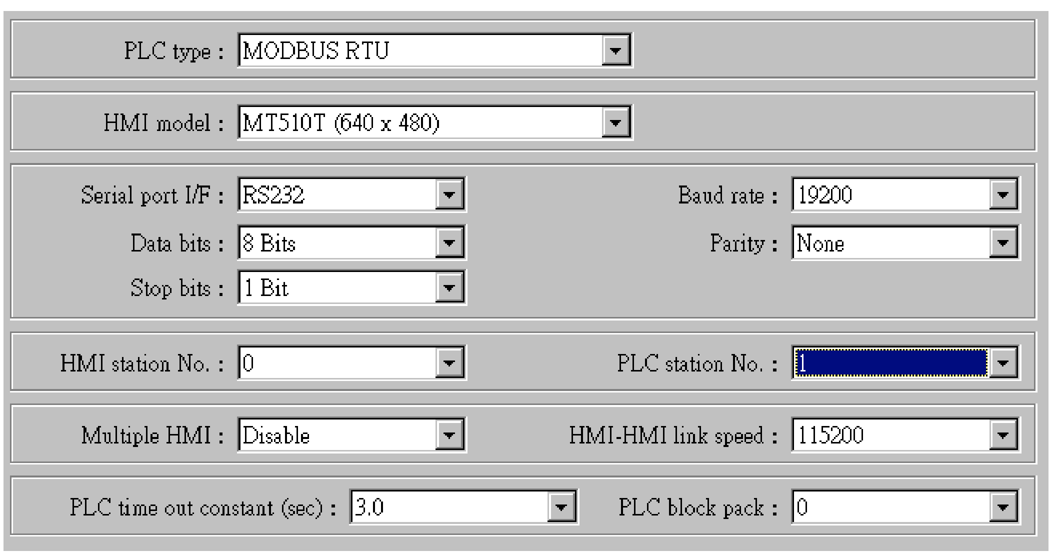
Click “File” – “New” to create a new project.



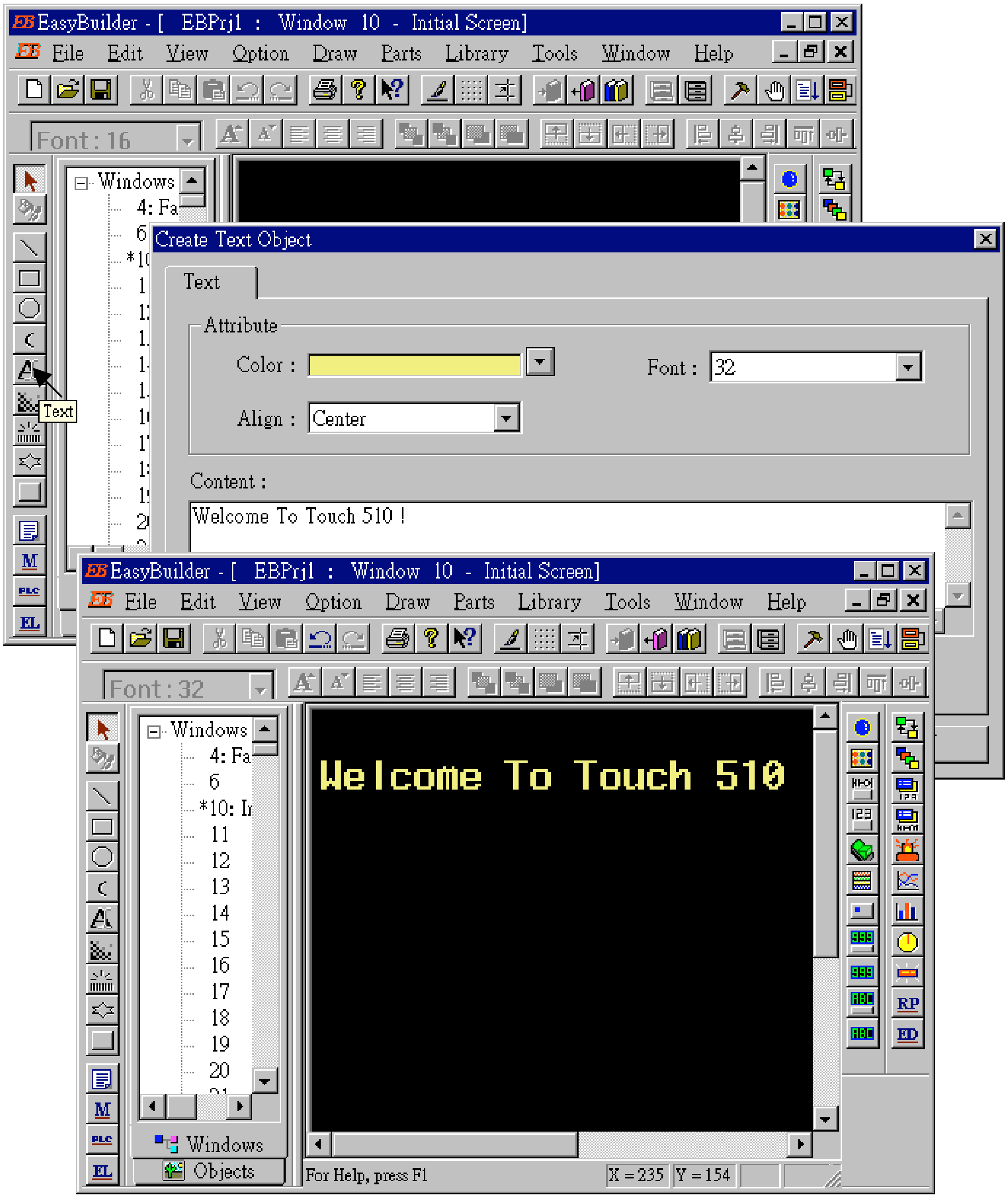
Click “Edit” – “System Parameters” to set the communication parameter between the Touch 510 and the ISaGRAF controller.



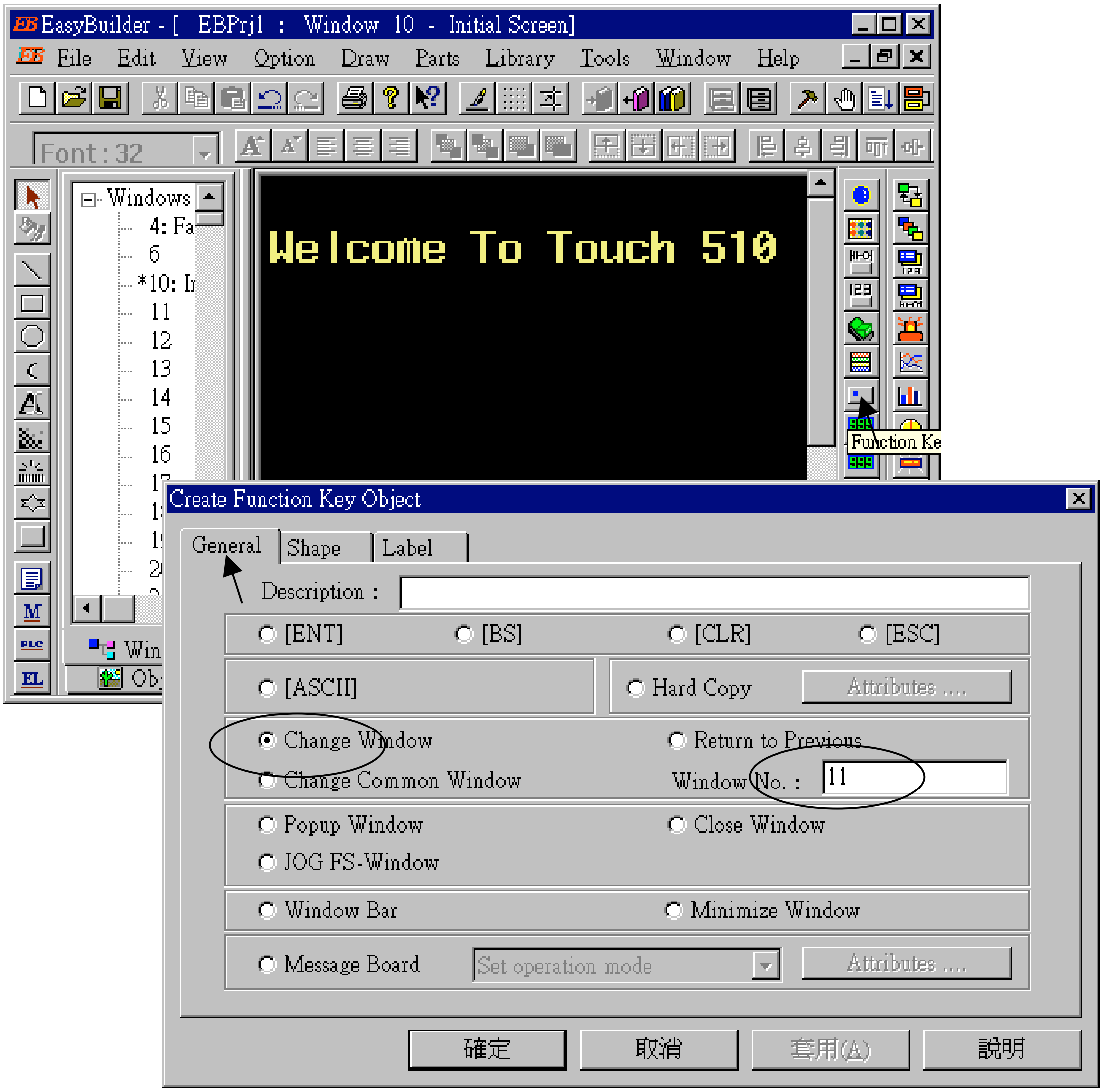
PLC type should be set to “**MODBUS RTU**”, Serial port set to “RS232”, Data bits set to “8 Bits”, Stop bits set to “1 Bit”, Baud rate set to “19200”, Parity set to “None”, PLC station No. set to be equal to the Net-ID of the I-8xx7 (set to 1 in this example).



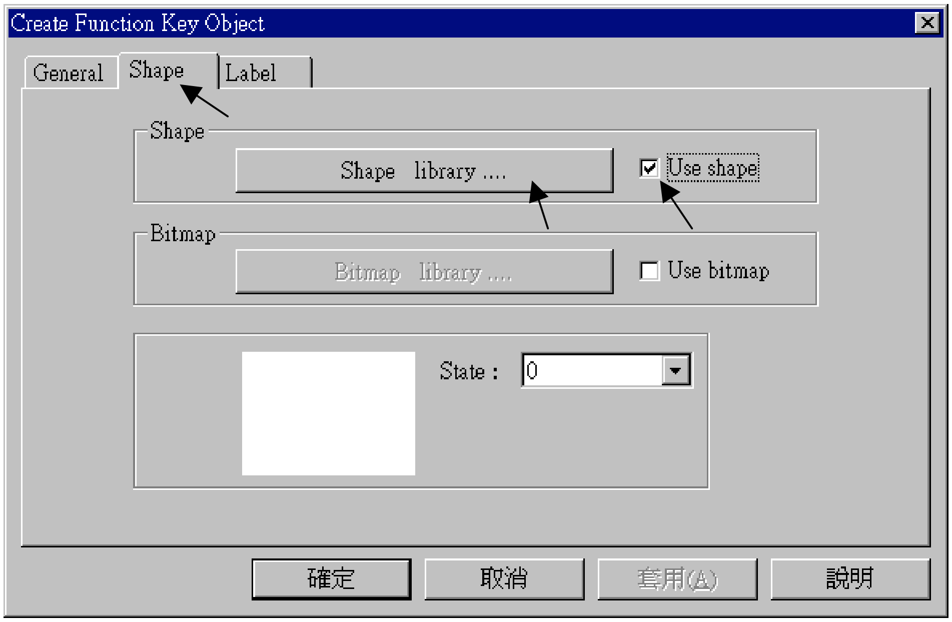
Click on “Text” to add a text. Select the prefered “Color”, “Font”, “Align” for the text and then enter the “Content”. And then place it to the proper position.



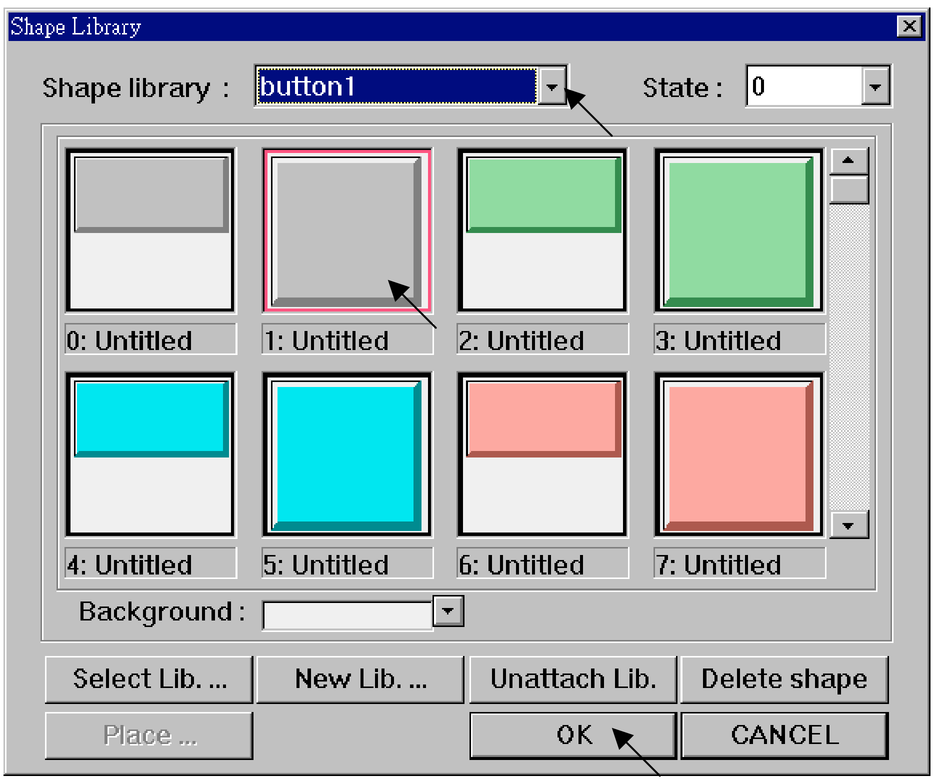
Click on “Function Key” to add a change-window button. Click on “General”, then select “Change Window” and set “Window No.” to 11.



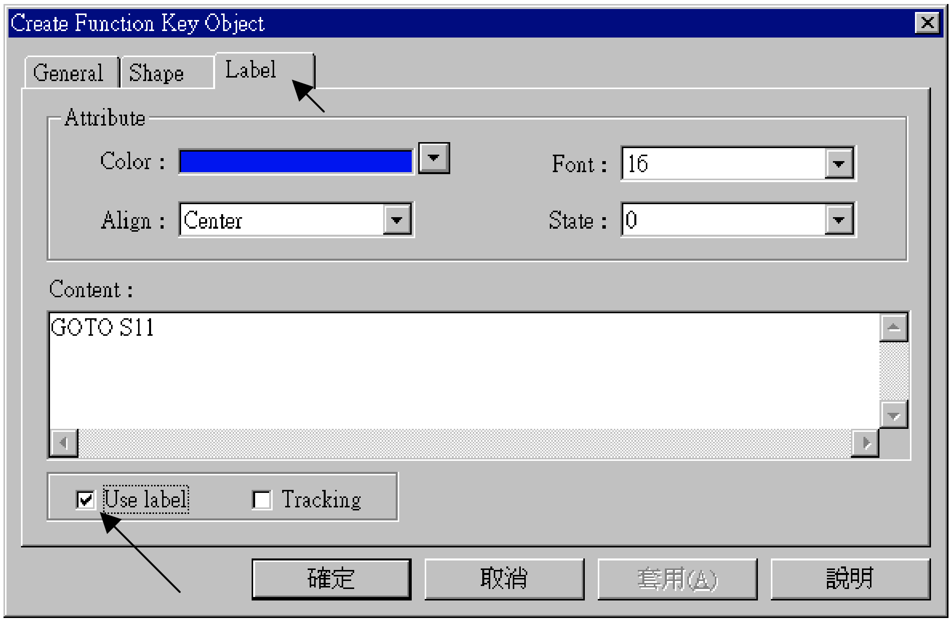
Click on “Shape”, then select “Use shape” and the click on “Shape library …”



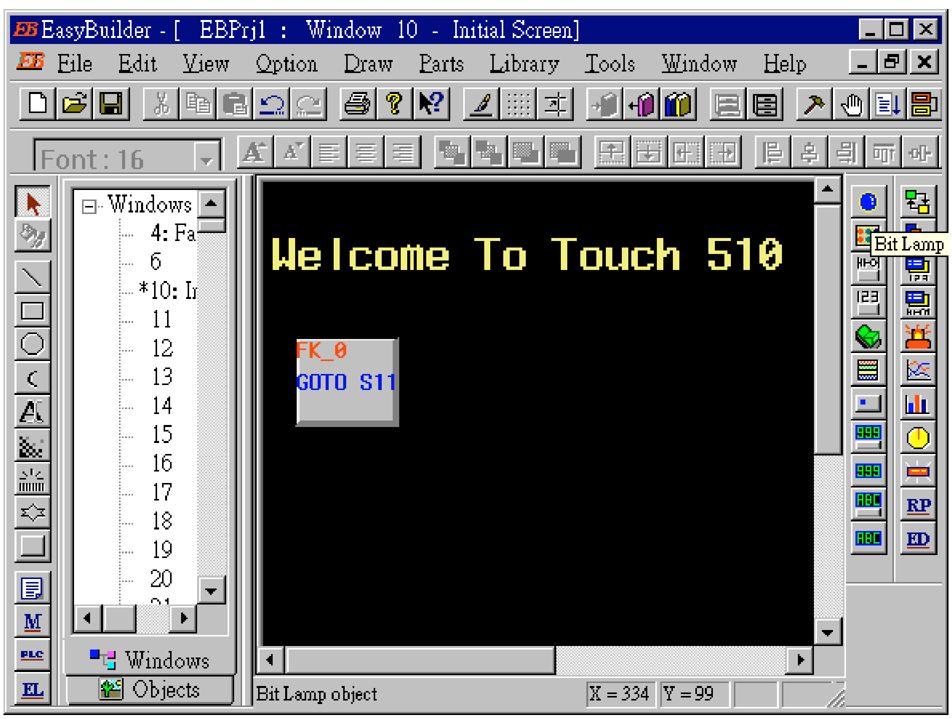
Select the prefered “Shape library” and then select one item and click on “OK”.



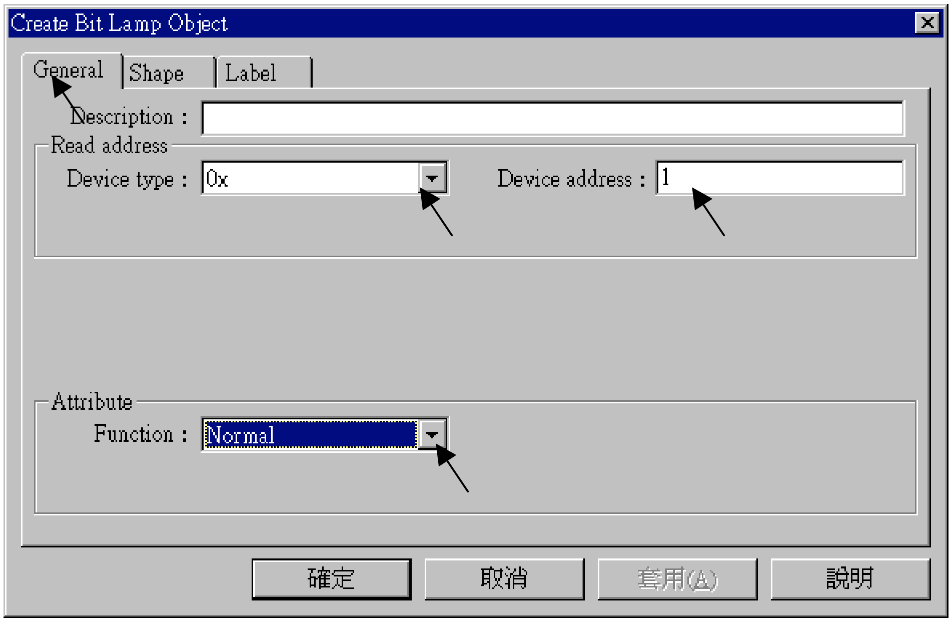
Click on “Label”, then select the prefered “Color”, “Font”, “Align” and set “Content” to “GOTO S11”, and **make sure “Use label” is selected**.



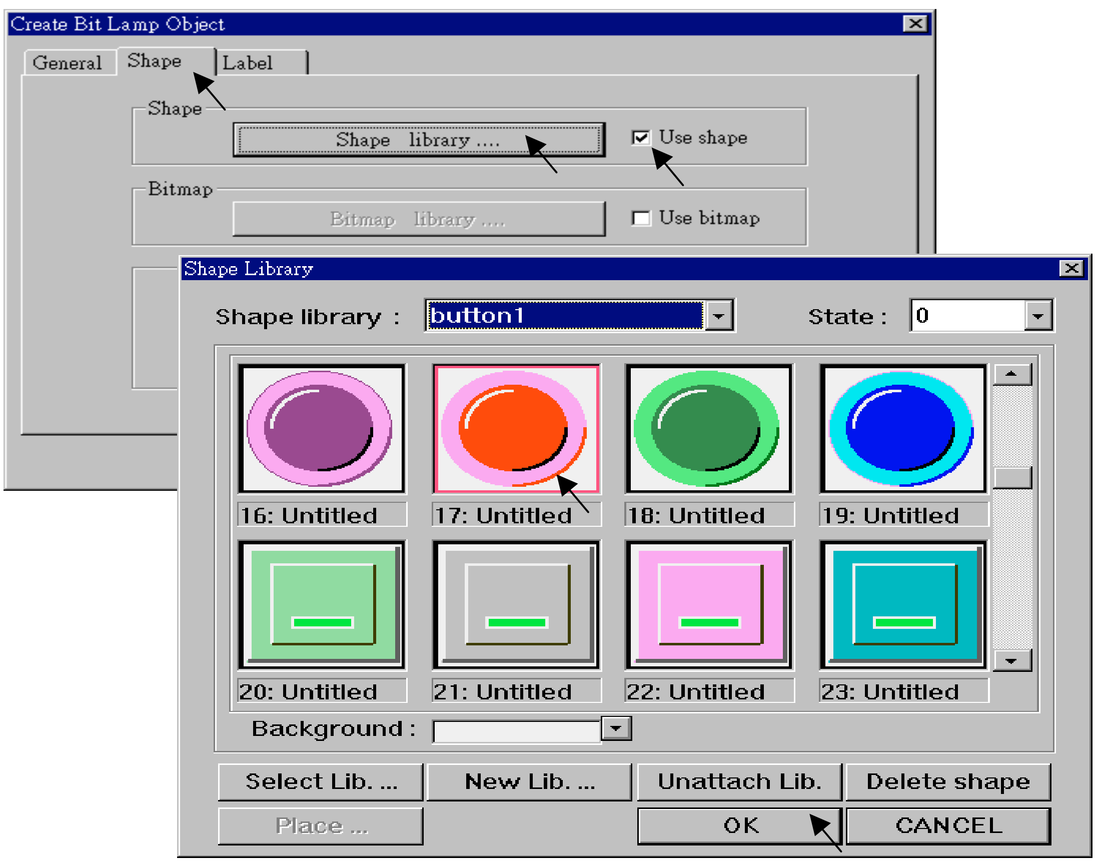
Click on “Bit Lamp”



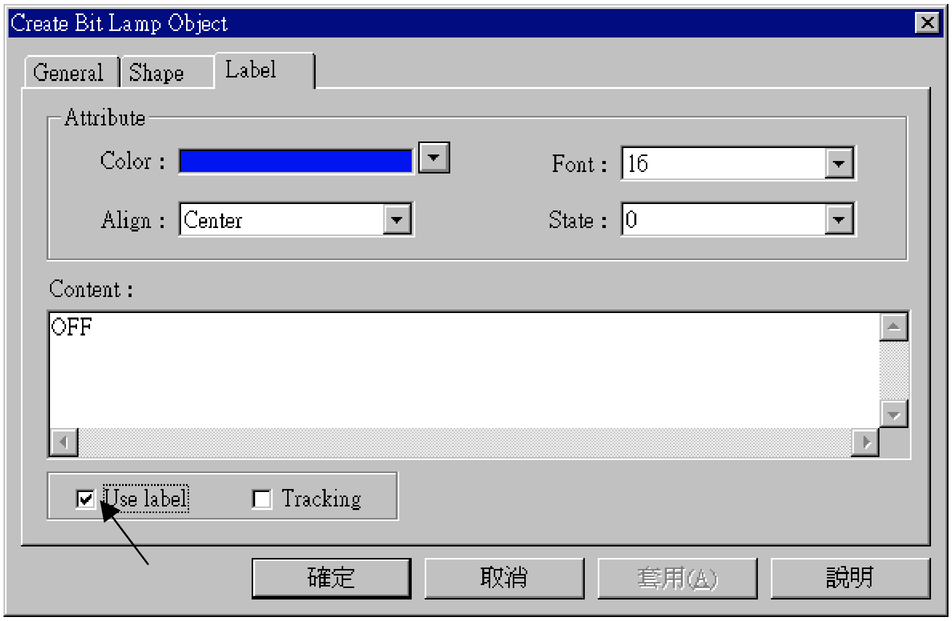
Click on “General”, then select “Device type” to “**0x**” (**0x is for boolean variables**), then set “Device address” to 1 (this value is associated with the network address value of the variable in the I-8xx7). And then set “Function” to “Normal”.



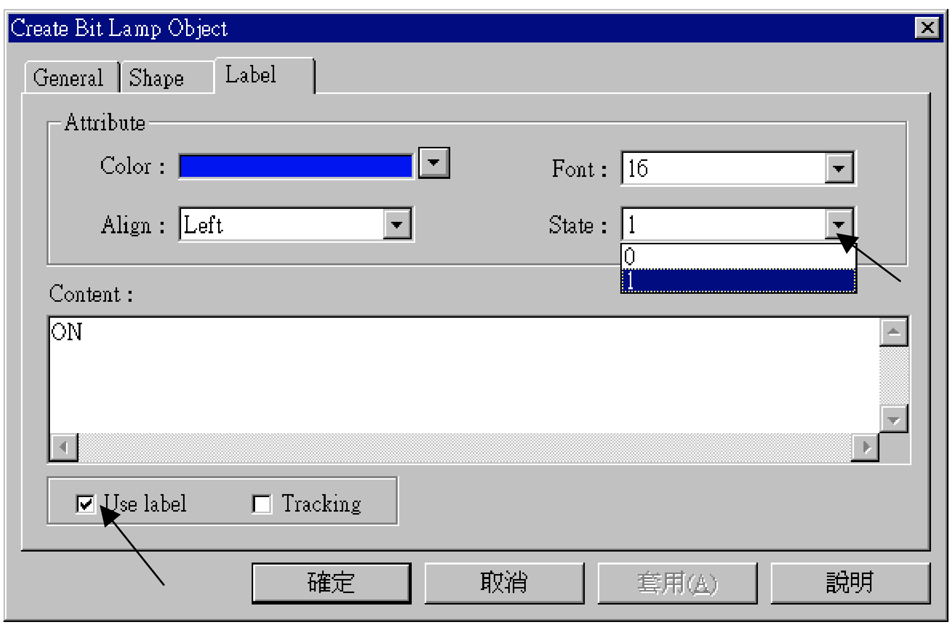
By the same way as former, select prefered “Shap library”.



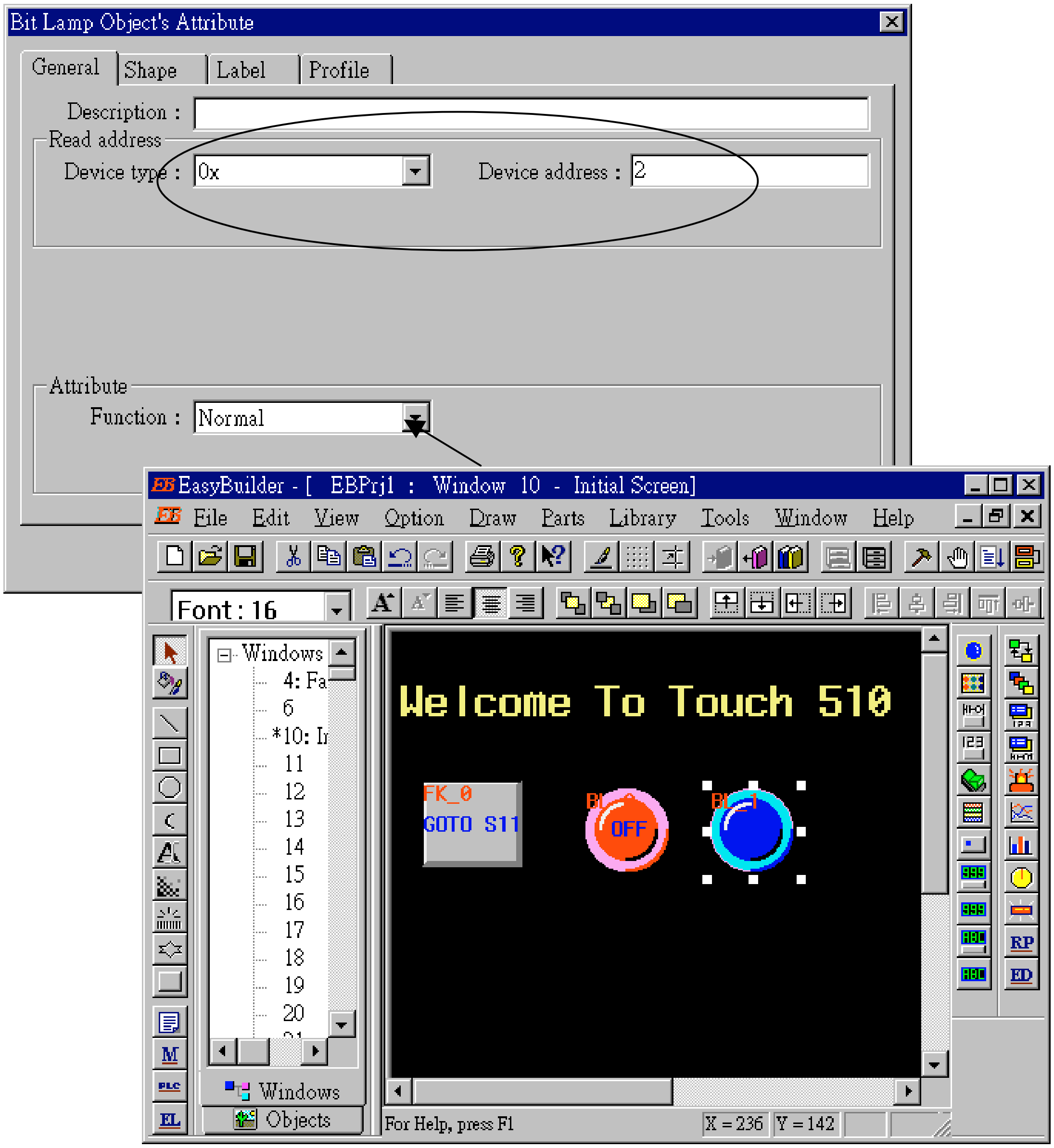
And then select “Label”, given a “OFF” to “Content” for “State : 0”. **Make sure “Use label” is choosed**.



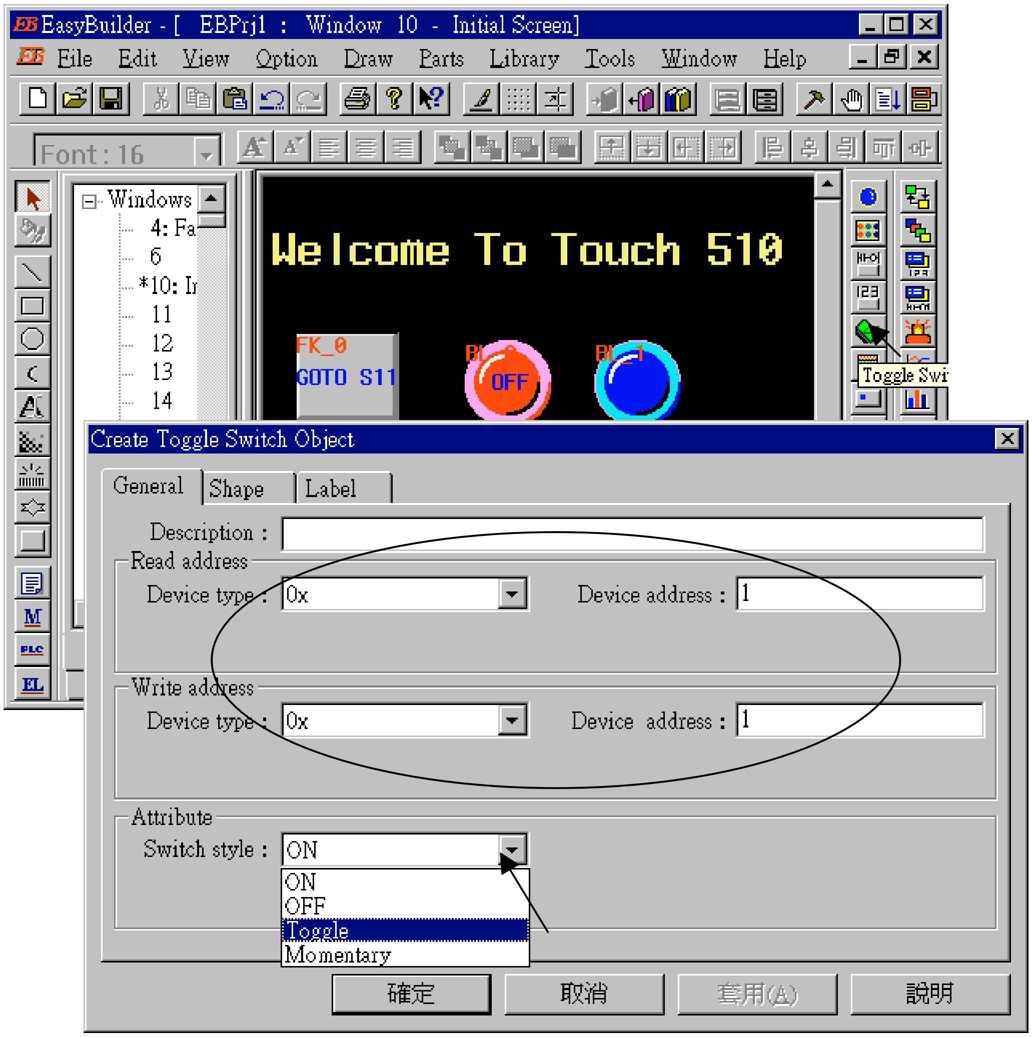
And then change “State” to 1, and given a “ON” to “Content”. **Make sure “Use label” is choosed.**



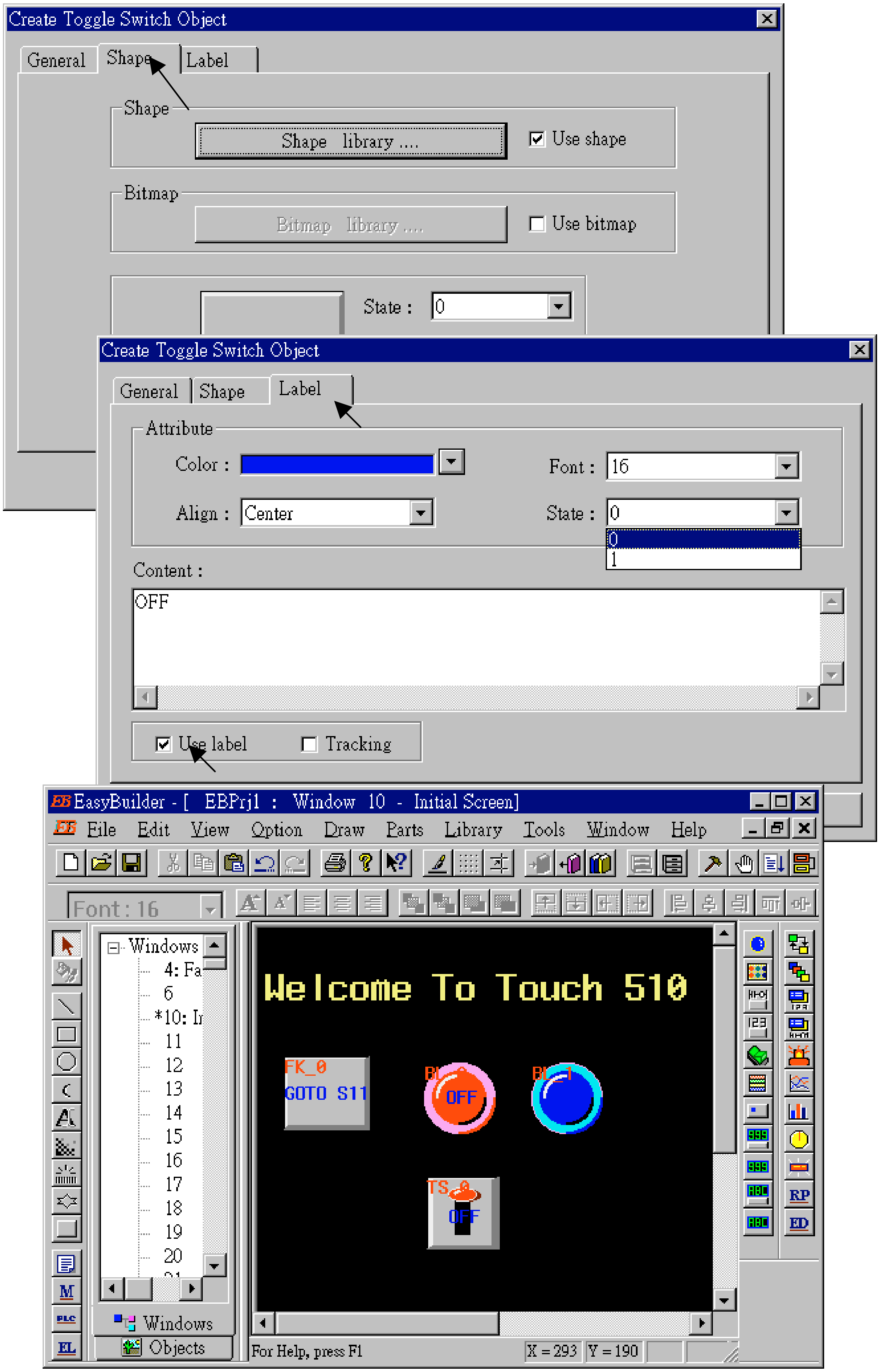
By the same way as former, create one another Bit Lamp with a “Device address” = 2.



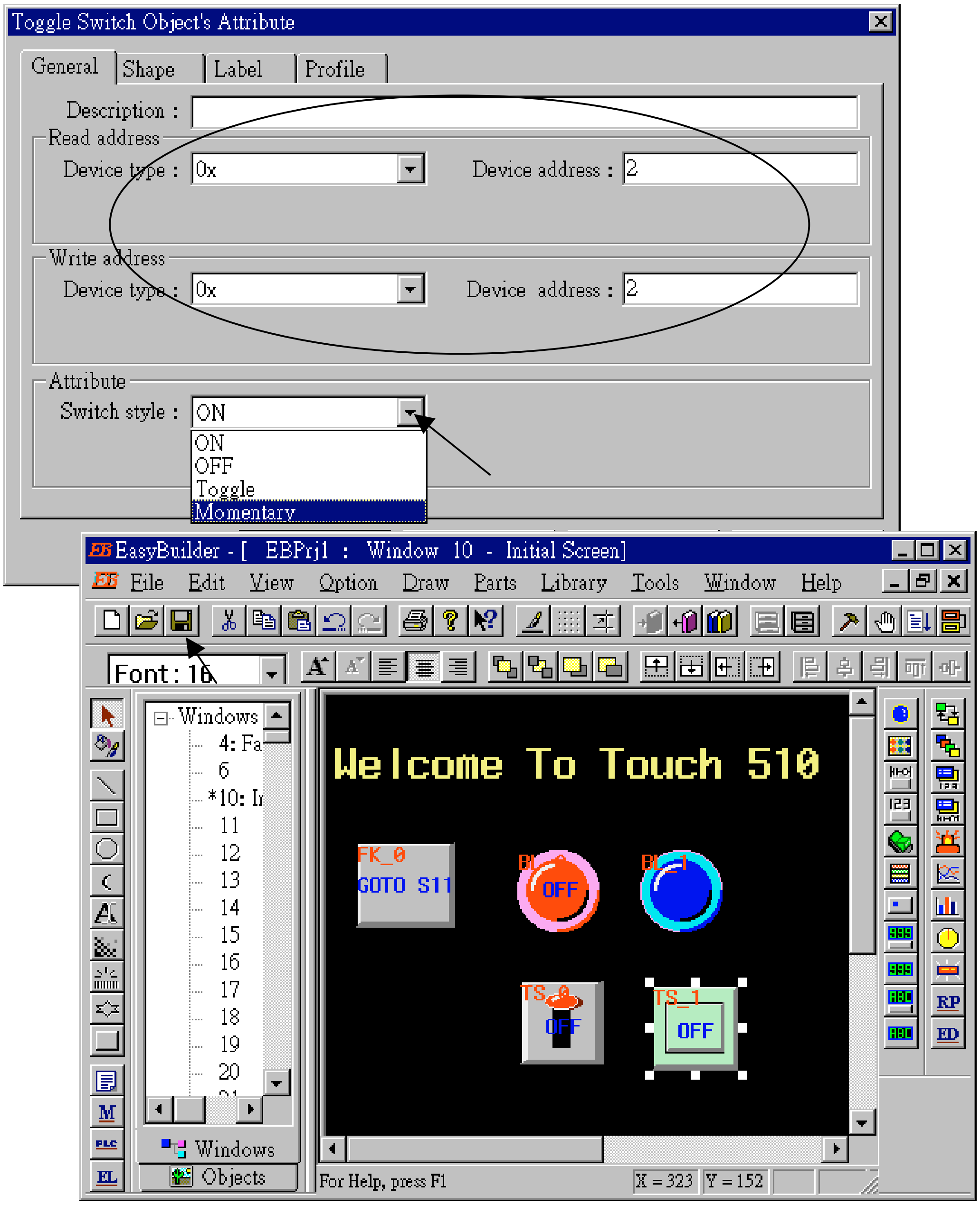
Click on “Toggle Switch”, then set all “Device Type” to “**0x**”, all “Device address” to 1 and select “Switch Type ” to “Toggle”.



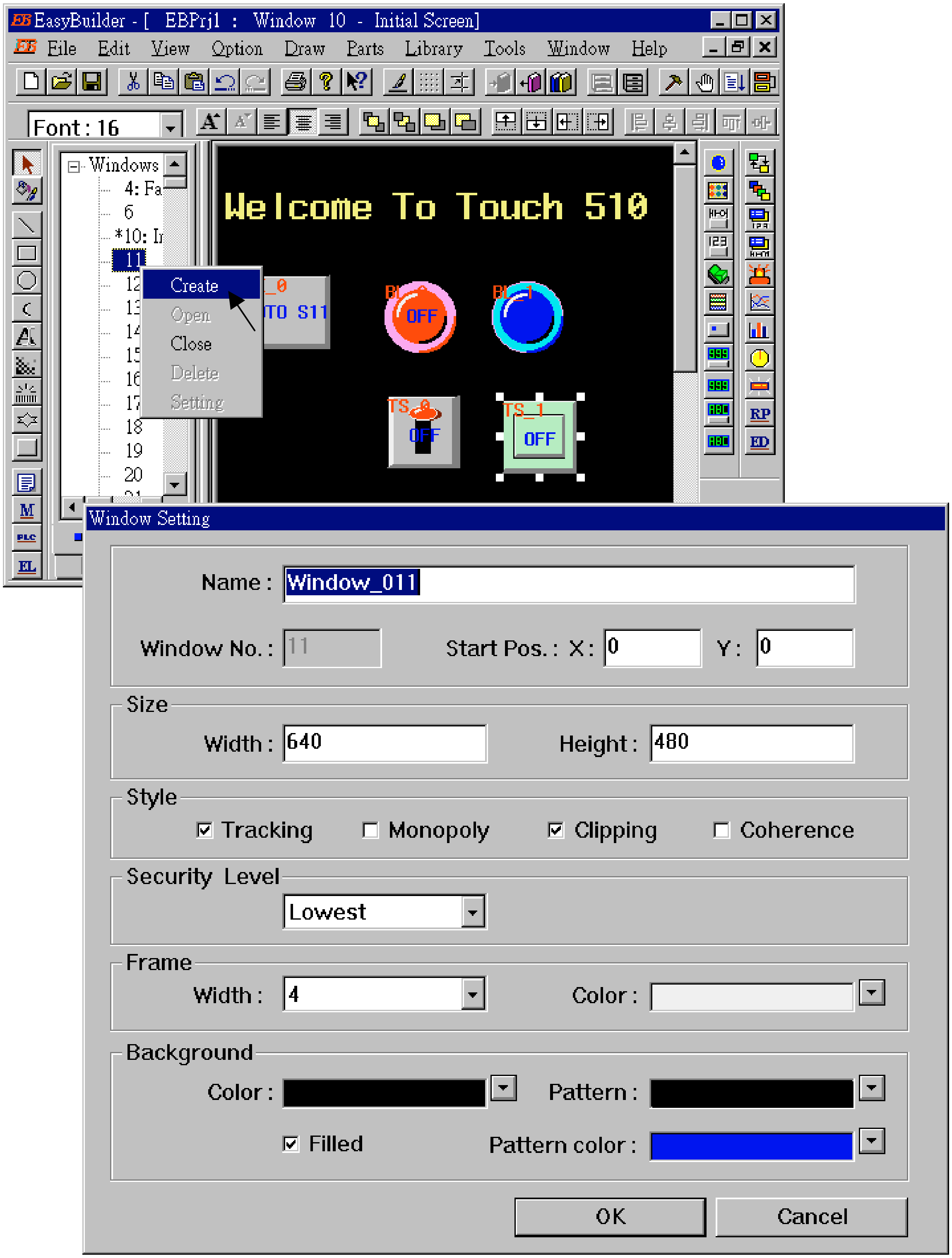
By the same way as former to choose a prefered “shape” and “label”.



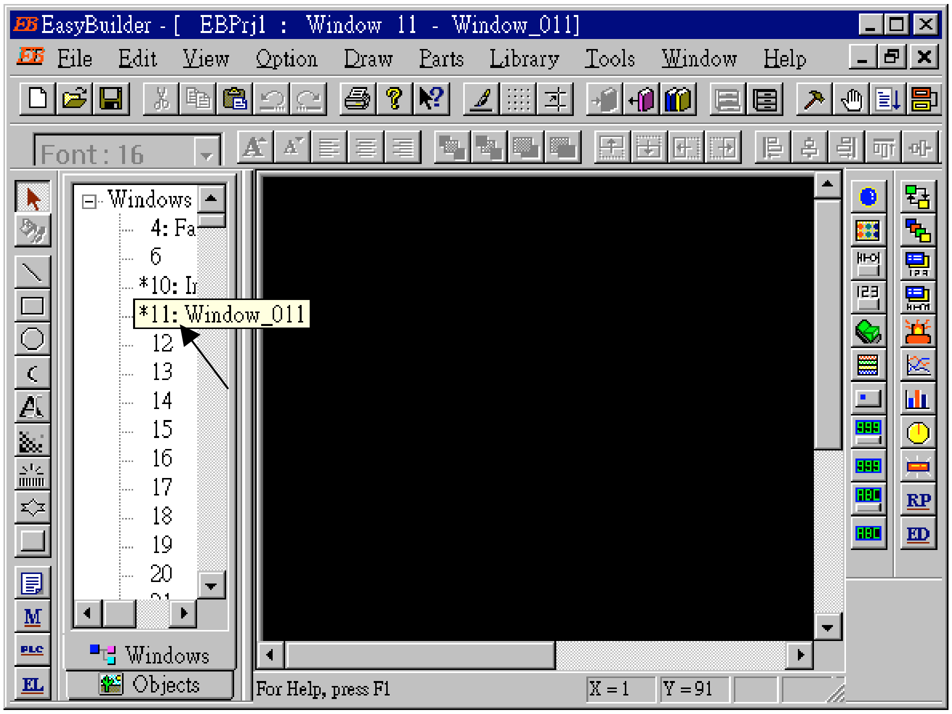
By the same way as former, create one another “Toggle Switch” however set all “Device address” to 2 and “Switch style” to “Momentary”. Click on “save” to save the project.



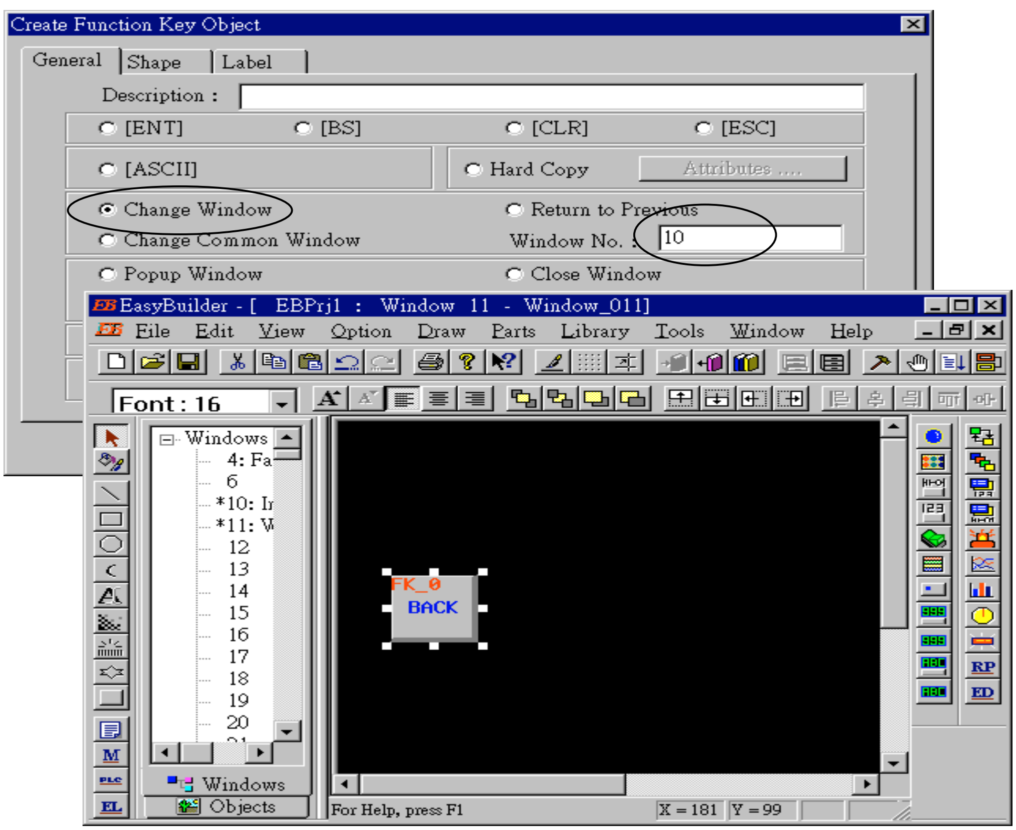
We are going to design another window. Click on “Windows” – “11”, then click and hold on the right button of the mouse and drag to “Create”.



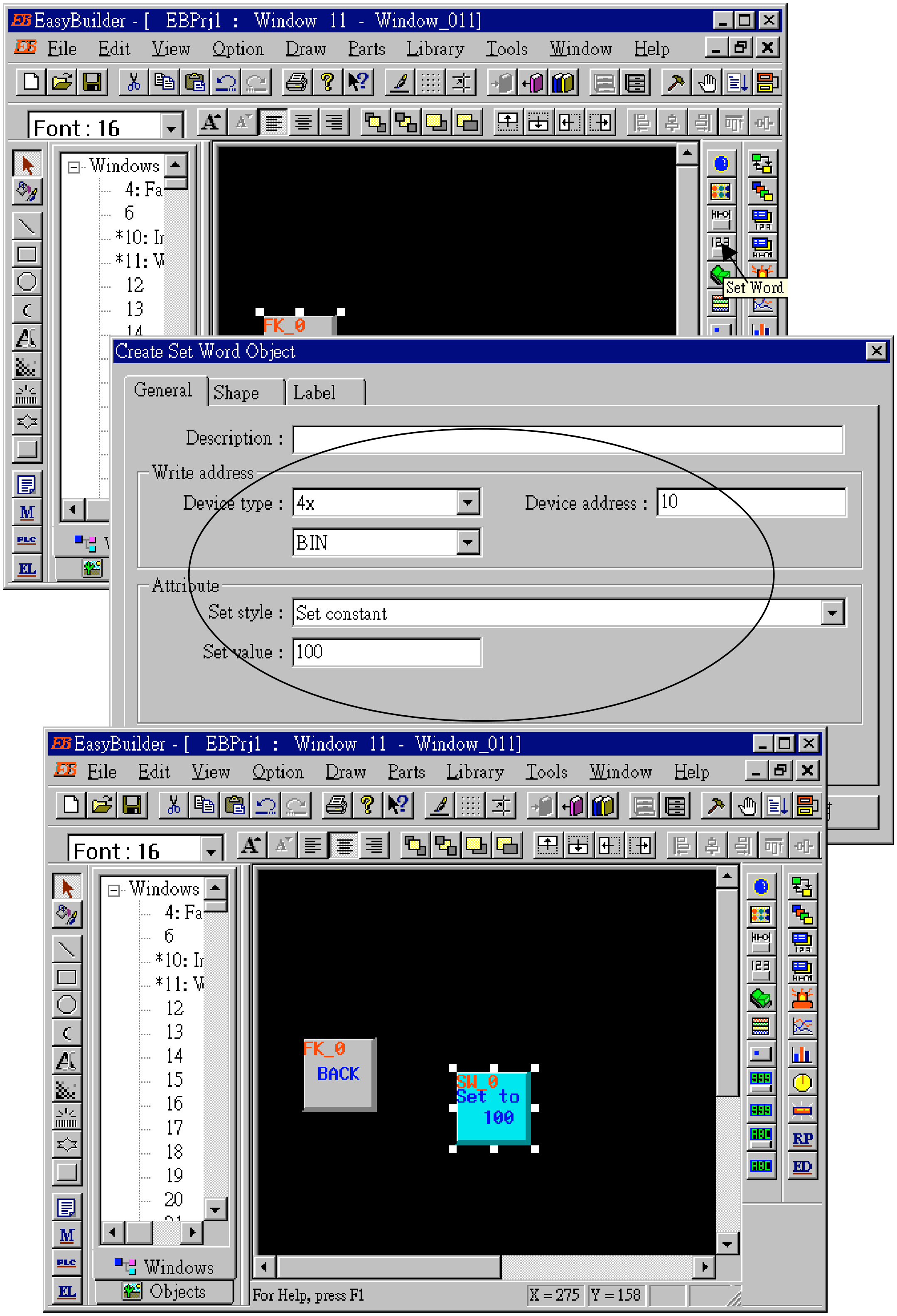
Double click on “Window_011”.



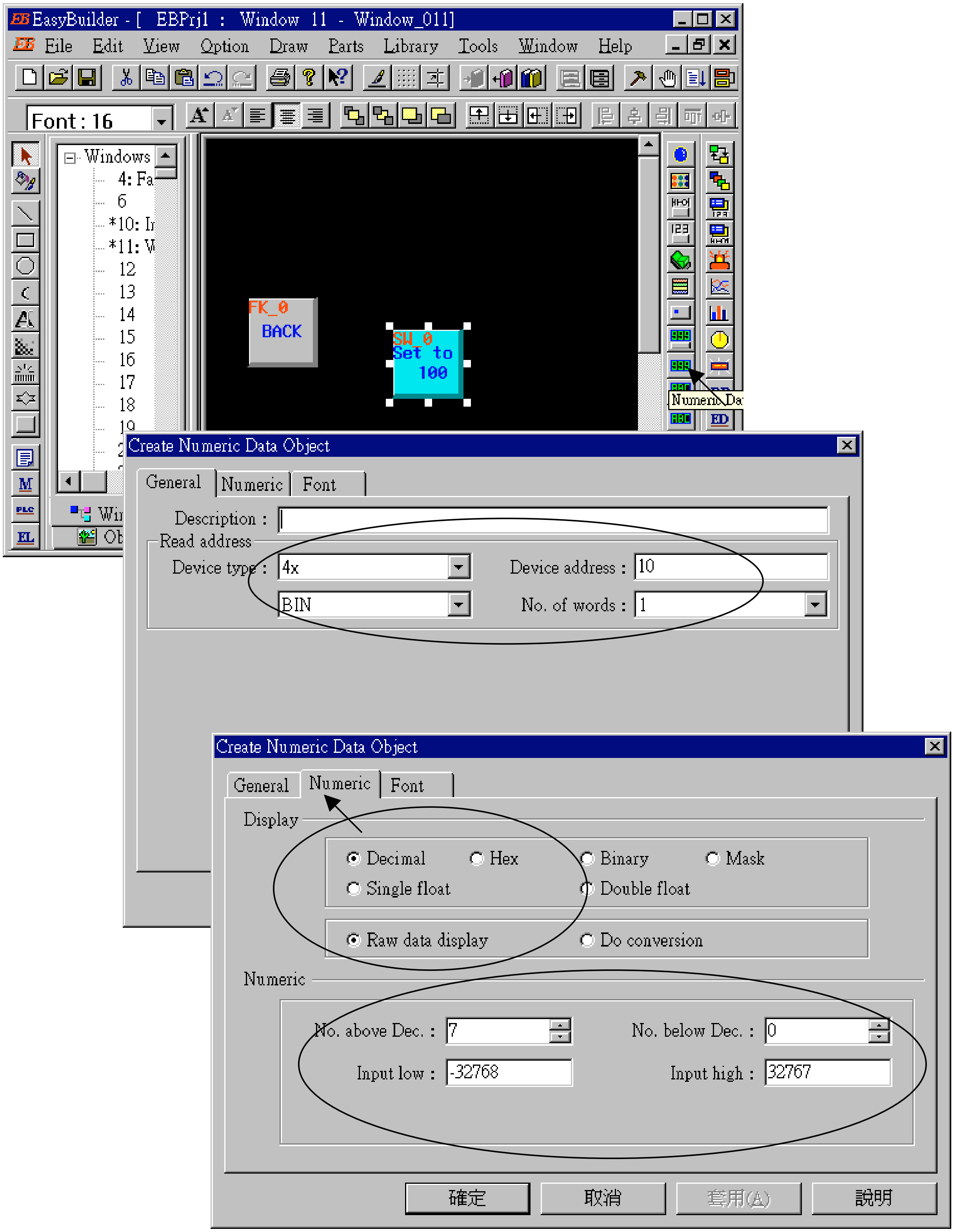
Create a change-window “Function Key” as former method to change to “Window No.” = 10, and Labeled as “BACK”.



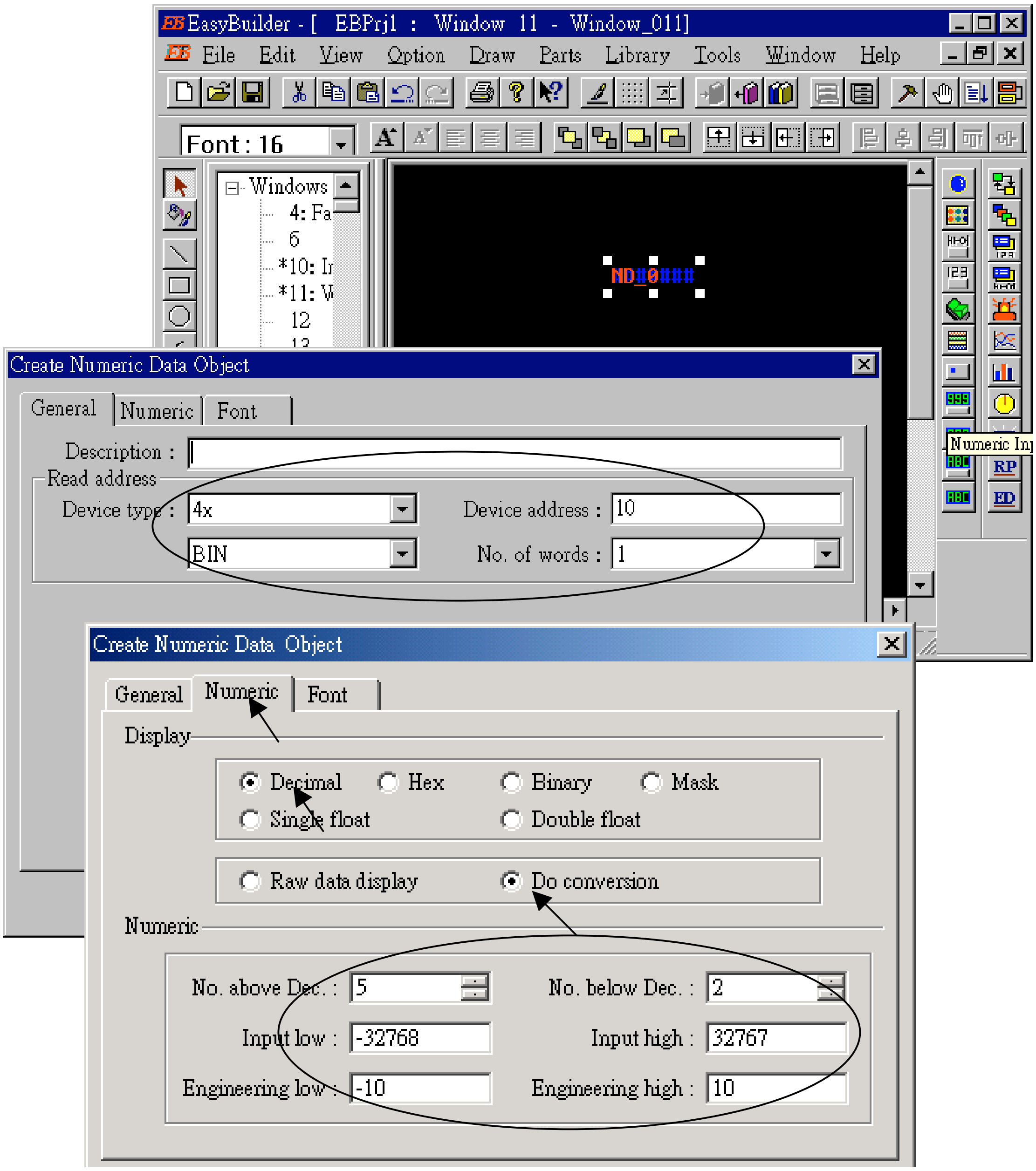
Click on “Set Word”, then set “Device Type” as “**4x**” (**4x is for short integer, 4L is for long integer**), set “Device address” to 10, “BIN”, and “Set style” to “Set Constant”, and “Set value” = 100. And then select the prefered “shape”, and set “label” to “Set to 100”.



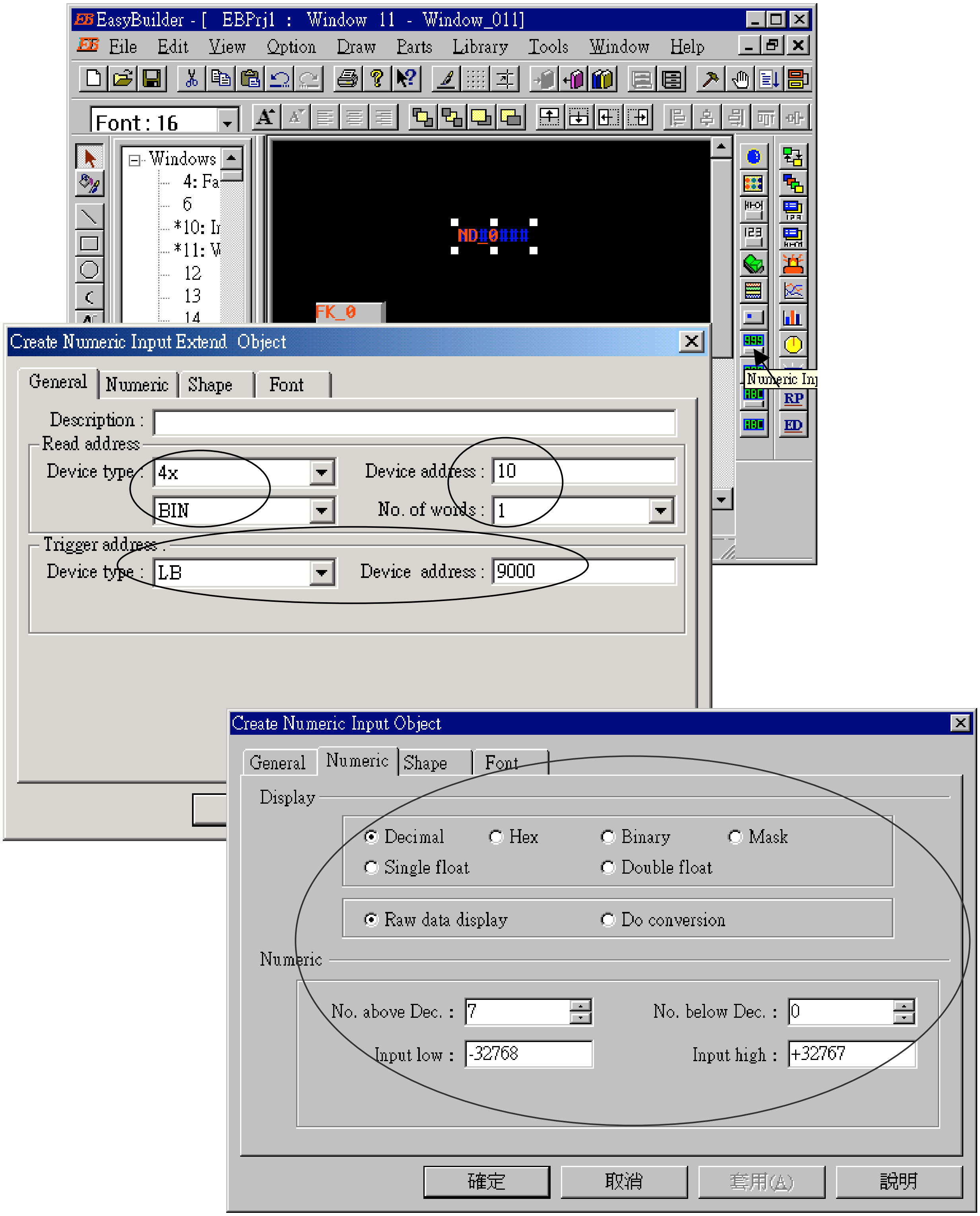
Click on “Numerical Data”, set “Device Type” to “4x” (**4x is for short integer, 4L is for long integer**), “Device address” to 10, “BIN”, “Number of words” to 1, “No. above Dec” to 7, “No. below Decimal” to 0, “Input low” to –32768, “Input high” to +32767. And then select the prefered shape.



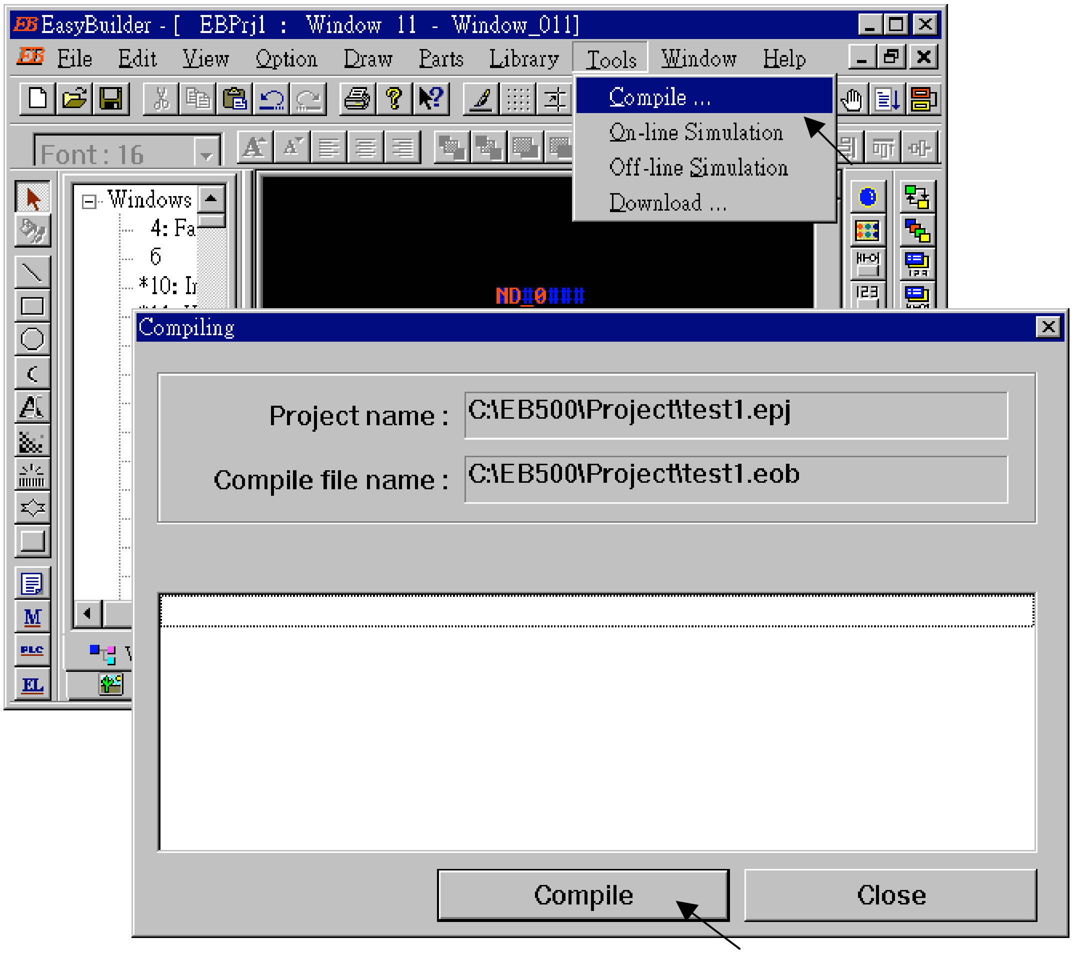
Now we are going to add one another “Numerical Data” with **conversion**.
Click on “Numerical Data”, set “Device Type” to “4x” , “Device address” to 10, “BIN”, “Number of words” to 1, “No. above Dec” to 5, “No. below Decimal” to 0, “Input low” to –32768, “Input high” to +32767, check “Do conversion”, set “engineering low” to –10, “engineering high” to +10 (**Convert [-32768,+32767] to [-10,+10]**). And then select the prefered font.



Click on “Numerical Input”, set “Device Type” to “4x”, “Device address” to 10, “BIN”, “Number of words” to 1, **“Trigger Device Type” to “LB”, “Trigger Device address” to “9000”,** “No. above Dec” to 7, “No. below Decimal” to 0, “Input low” to –32768, “Input high” to +32767. And then select the prefered shape.

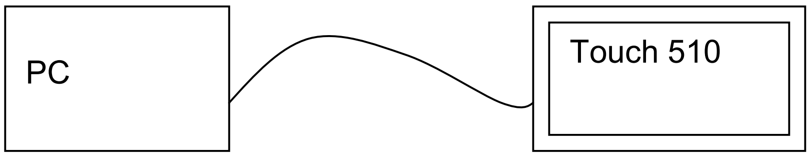


Click “Tools” – “Compile …” to compile this project.

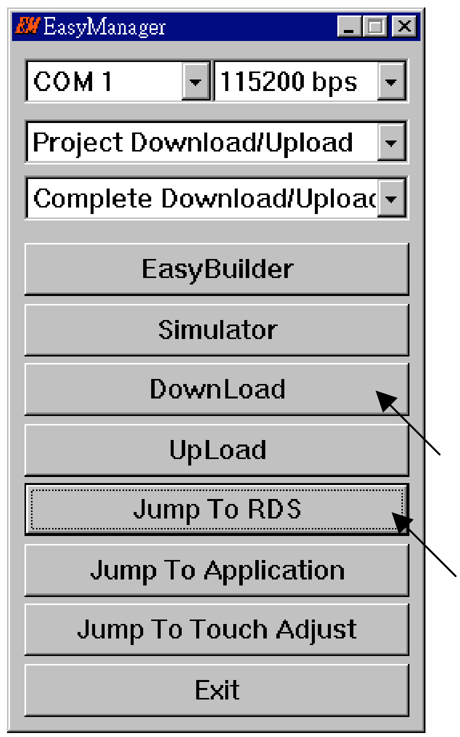


To download the project to the Touch 510, click on the Windows "Start" button, then click on the "Program" button, then click on the "EasyBuilder" – "EasyManager" button. The following window will be displayed. Choose the correct COM No. on your PC (Normally is COM1), "115200 bps".

Connect the RS232 download cable (refer to section 4.4) between PC and Touch 510.

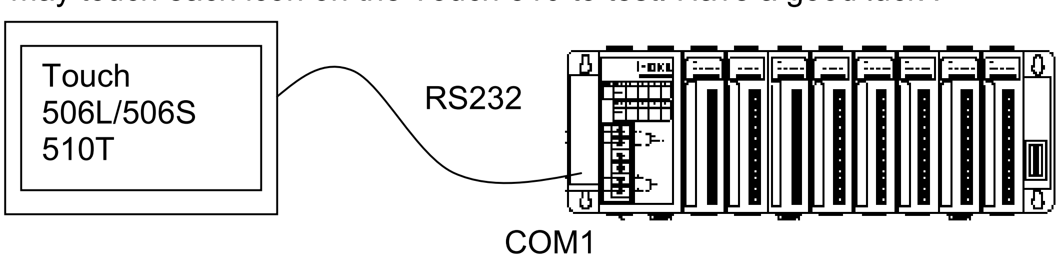


Click on "Jump To RDS" first, if OK., you can see the screen of the Touch 510 will change and wait for project download. Click on "Download" to start to download the MMI picture to the Touch 510.



If downloading is OK, You may choose to click on "Jump To Application" or reset the Touch 510T , and then connect another RS232 cable between Touch 510 and the I-8xx7 (refer to section 4.4).

Now, you may touch each icon on the Touch 510 to test. Have a good luck !



4.5: Access To Word & Integer Array Via Modbus

User can use the below functions to read/write word & integer arrays inside the ISaGRAF project. For more information about these functions, please refer to Appendix A.4.

ARY_N_R	Read one integer(4 byte, signed) from an integer array
ARY_N_W	Write one integer(4 byte, signed) to an integer array
ARY_W_R	Read one word(2 byte, signed) from an word array
ARY_W_W	Write one word(2 byte, signed) to an word array

Word and integer arrays built in the I-8xx7, I-7188EG, I-7188XG & Wincon-8xx7 controller occupy the same memory area, please use them carefully. Other softwares (HMI, OPC server, …) running on the PC can access to these word and integer arrays via **Modbus** protocol. The valid **network address** for these arrays is from **5001 to 8072 for I-8xx7, I-7188EG & I-7188XG**, while **10,001 to 19,216 for the W-8xx7** and their relation is listed in below table.

For the I-8xx7, I-7188EG, I-7188XG:

Network Address (Decimal)	Word Array	Integer Array
5001	(1,1)	(1,1)
5002	(1,2)	
5003	(1,3)	(1,2)
5004	(1,4)	
…	…	…
…	…	
8071	(12,255)	(6,256)
8072	(12,256)	

For the W-8xx7:

Network Address (Decimal)	Word Array	Integer Array
10001	(1,1)	(1,1)
10002	(1,2)	
10003	(1,3)	(1,2)
10004	(1,4)	
…	…	…
…	…	
19215	(36,255)	(18,256)
19216	(36,256)	

Note:

1. **Network address 1 to 4095 for I-8xx7 & I-7188EG/XG, while 1 to 8191 for W-8xx7,** can be defined by users, please refer to Section 4.1.
2. **Modbus address** in the physical transmission format is equal to **Network address** minus one (please refer to Chapter 5). So the valid Modbus address for word & integer arrays is from 5000 to 8071 for I-8xx7, I-7188EG/XG, and 10000 to 19215 for W-8xx7.

Chapter 5: Modbus Protocol

The Modbus protocol is a powerful and flexible communications protocol that allows numerous software programs and hardware devices to communicate with each other. Any I-8xx7, I-7188EG/XG & W-8xx7 variable that will be used to communicate through the Modbus protocol **MUST** have a unique network address before it can communicate through a Modbus link (please refer to section 4.1).

5.1: Modbus Protocol Format: RTU Serial

The Modbus "RTU Serial" format is supported by the I-8417 and I-8817 controller systems through both COM1 or COM2 communications ports, and the I-8437, I-8837, I-7188EG & I-7188XG controller systems through the COM1 communications port, and the Wincon-8037/8337/8737 controller systems through the COM2 communications port.

PC software programs and HMI hardware devices can access data from the variables in the ISaGRAF controller system **ONLY** after that variable is assigned a unique network address (please refer to Chapter 4). For more information regarding connecting a PC to an I-8xx7 controller system, please refer to Section 1.3.3 through 1.3.5 for details on how to properly connect these devices.

It is **CRITICAL** that you must program the Modbus format **EXACTLY** as described to make a proper connection between the Modbus device and the ISaGRAF controller system. The I-8xx7, I-7188EG/XG & W-8xx7 controllers support the following Modbus functions.

Modbus function	Action
1	Read N bits (booleans)
2	Read N bits (booleans)
3	Read N words (signed short integers)
4	Read N words (signed short integers)
5	Write 1 bit (boolean)
6	Write 1 word (signed short integer)
15	Write N bits (booleans)
16	Write N words (signed short integers)

To read boolean variables, both of function 1 or 3 may be used. If using function 1, values are stored in a bit field while using function 3, variable TRUE means 0xFFFF.

To write boolean variables, both of function 5, 15 could be used. If using function 5, writing bit 0 of byte-vH to 1 will set the Boolean variable to TRUE. For ex, writing vH=1 or 3, or 255 will set Boolean variable to TRUE.

To read analog variables, function 3 should be used.

To write analog variables, both of function 6, 16 could be used.

To read long words (signed long integers, float), function 3 should be used. To write long words, function 16 should be used. Please refer to section 4.2 for the definition of network address of long words.

To assist you with the naming conventions used throughout the Modbus protocol-addressing chapter, the following table describes the notations used in this chapter.

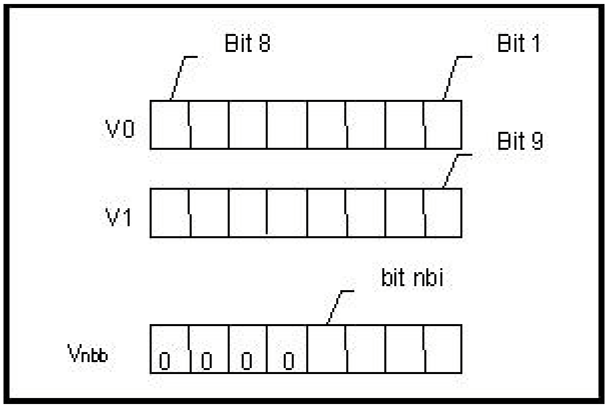
Slv	Slave number (Net ID address of the I-8xx7)
Nbw	Number of words
Nbb	Number of bytes
Nbi	Number of bits
AddH	**Modbus address**, high byte , 0 ~ 0F
AddL	**Modbus address**, low byte , 0 ~ FE
VH	Word value, high byte
VL	Word Value, low byte
V	Byte value
CrcH	Checksum, high byte , CRC-16
CrcL	Checksum, low byte , CRC-16

IMPORTANT NOTE
All of the values used in the request and answer frames are **hexadecimal** values.
Modbus address described in this chapter is equal to Network address of the variable minus one.
For ex., Modbus address 0 is associate with Network address 1. Modbus address FFE (4094) is associate with Network address FFF (4095).

Function 1: Read "N" Bits
Function 1 reads "n" number of bits (nbi) in Boolean starting from Modbus address addH/addL.

Request:	slv	01	addH	addL	00	nbi	crcH	crcL
Answer:	slv	01	nbb	V0	V1	…	crcH	crcL

V0, V1 … are the bit fields of number of bytes (nbb) using the following format.



Bit 1 corresponds to the Boolean value of the variables with the Modbus address addH/addL. Bit nbi corresponds to the Boolean value of the variable with the Modbus address addH/addL + nbi – 1. If the value of the Boolean variable is "True", then the corresponding bit will be set to a "1". If the value is "False", the corresponding bit will be set to a "0".

Function 2: Read N Bits
Function 2 has the same exact same format as function 1.

Function 3: Read N Words
Function 3 reads the number of words (nbw), in signed 16-bit integer format, starting from the Modbus address addH/addL.

Request:	slv	03	addH	addL	00	nbw	crcH	crcL
Answer:	slv	03	nbb	vH	vL	...	crcH	crcL

The number of bytes (nbb) is the total number of bytes from word value high byte (vH) to word value low byte (vL) inclusive.

IMPORTANT NOTE About Function 3
Integer values can be read by function 3. A word in the modbus protocol is a 16-bit value (signed short integer), and an integer variable is a 32-bit value, so only the lower 16 bits of the integer variable are returned. If users would like to read a 32-bit integer (signed long integer) of I-8xx7 controller, the proper network address of the variable should be set as described in section 4.2.

Function 4: Read N Words
Function 4 has the same exact format as function 3.

Function 5: Write 1 Bit
Function 5 writes one (1) bit to the Boolean variable with the Modbus address addH/addL.

Request:	slv	05	addH	addL	V	0	crcH	crcL
Answer:	slv	05	addH	addL	V	0	crcH	crcL

Writing a 0xFF value to the byte value (V) will set the Boolean variable to "True". Writing a zero to the byte value (V) is set the Boolean variable to "False".

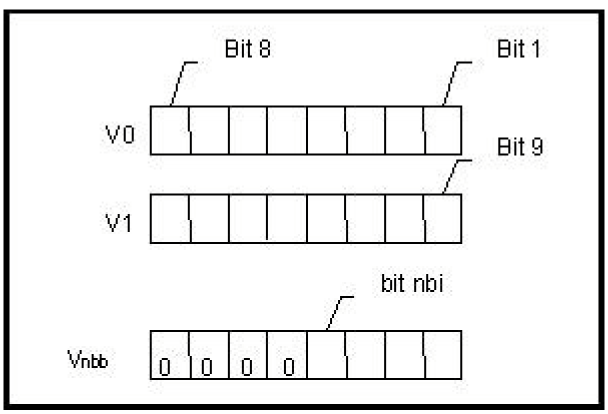
Function 6: Write 1 Word
Function 6 writes one (1) word (16 bits) to the integer variable with the Modbus address addH/addL.

Request:	slv	06	addH	addL	vH	vL	crcH	crcL
Answer:	slv	06	addH	addL	vH	vL	crcH	crcL

Function 15: Write N Bits
Function 15 writes a number of bits (nbi) to the Boolean variables starting from the Modbus address addH/addL to addH/addL + nbi – 1. The total number of bytes (nbb) is the total amount of bytes occupied by nbi bits, that means nbb = (nbi+7)/8. For ex. nbi=1~8, nbb=1; nbi=9~16, nbb=2.

Request:	slv	0F	addH	addL	00	nbi	nbb	V0	V1	…	crcH	crcL
Answer:	slv	0F	addH	addL	00	nbi	crcH	crcL				

V0, V1 … are the bit fields of number of bytes (nbb) using the following format.



Bit 1 corresponds to the Boolean value of the variables with the Modbus address addH/addL. Bit nbi corresponds to the Boolean value of the variable with the Modbus address addH/addL + nbi – 1. Writing a 1 to a bit will set the value of the corresponding Boolean variable to "True", and writing a 0 to a bit will set the corresponding Boolean variable to "False".

Function 16: Write N Words

Function 16 writes a number of words (nbw) to the integer variables starting from the Modbus address addH/AddL to addH/addL + nbw – 1. The number of bytes (nbb) is the total amount of bytes occupied by number of words (nbw), that is nbb = 2 * nbw.

Request:	slv	10	addH	addL	00	nbw	nbb	vH	vL	...	crcH	crcL
Answer:	slv	10	addH	addL	00	nbw	crcH	crcL				

Examples Of Modbus Function Formats

Function 1: Read 15 bits starting from **Modbus address 0**x1020. The NET ID address is 1.

Request:	01	01	10	20	00	0F	79	04
Answer:	01	01	02	00	12	39	F1	

In this example function 1 returns 2 bytes, the value is 0x0012. This means variables with a **network address** of 0x102A and 0x102D are "True" (**Modbus address** is 0x1029 and 0x102C), the rest of the variables are set to "False".

Function 5: Write 1 bit to the Boolean variable with the **Modbus address** 0x0006. The NET ID address is 1. The value to write to is 0xFF.

Request:	01	05	00	06	FF	00	6C	3B
Answer:	01	05	00	06	FF	00	6C	3B

In this example of function 5 the Boolean variable is set to "True".

Function 16: Write 2 words (4 bytes) to the integer variables with the **Modbus address** starting from 0x2100. The first word value to write to is 0x1234. The second word value to write to is 0x5678. The NET ID address is 1.

Request:	01	10	21	00	00	02	04	12	34	56	78	1C	CA
Answer:	01	10	21	00	00	02	4B	F4					

5.2: Modbus Protocol Format: TCP/IP

The I-8437 and I-8837 (Ethernet port) controller systems support the Modbus "TCP/IP" communications protocol.

ALL requests are sent via TCP on port number **502**.

The Modbus TCP/IP protocol adds 6 extra bytes before the Modbus RTU serial protocol, and these 6 extra bytes and the Modbus RTU serial protocol are all packed inside the TCP/IP protocol.

TCP/IP	Extra 6 Bytes	Modbus RTU serial	TCP/IP

The request and responses are prefixed by the six bytes as follows:

Byte 0: transaction identifier - copied by server
Byte 1: transaction identifier - copied by server
Byte 2: protocol identifier = 0
Byte 3: protocol identifier = 0
Byte 4: length field (upper byte) = 0 (since all messages are smaller than 256)
Byte 5: length field (lower byte) = number of following bytes

The rest of the Modbus TCP/IP protocol is the same as the Modbus RTU Serial protocol after byte No. of 6 except that the CRC-16 is not need for the Modbus TCP/IP protocol.

Example TCP/IP Transactions
The first example of a TCP/IP transaction is reading one (1) word at Modbus address 4 from slave number 9 returning a value of 8; the transaction would be as follows:

Request:	01	02	00	00	00	06	09	03	00	04	00	01
Response:	01	02	00	00	00	05	09	03	02	00	08	

The second example of a TCP/IP transaction is reading 8 bits starting from Modbus address 2 from slave number 7, returning a value of 0x49 (bit field: 01001001) would be as follows:

Request:	03	29	00	00	00	06	07	01	00	02	00	08
Response:	03	29	00	00	00	04	07	01	01	49		

5.3: Algorithm For CRC-16 Check

The following C language algorithm is for Modbus RTU Serial ONLY!! This CRC (Cyclic Redundancy Check) program provides a checksum that can be used to validate information being passed through Modbus RTU Serial protocol.
This CRC-16 check program first calls "crc_init()" one time at the beginning of the communication to initialize the checksum table. Then you can call "crc_make()" to calculate a checksum whenever you want to.

```
#define POLY_CRC16 0xA001
static BYTE TABLE1[256];
static BYTE TABLE2[256];

void crc_init(void)   /* set crc table */
{
  WORD mask,bit,crc,mem;
  for(mask=0;mask<0x100;mask++)
  {
    crc=mask;
    for(bit=0;bit<8;bit++)
    {
      mem=crc & 0x0001;
      crc/=2;
      if(mem!=0) crc ^= POLY_CRC16;
    }
    TABLE2[mask]=crc & 0xff;
    TABLE1[mask]=crc >> 8;
  }
}

void crc_make(WORD size, BYTE *buff, BYTE *hi, BYTE *lo)   /* calculate crc */
{
  BYTE car,i;
  BYTE crc[2];
  crc[0]=0xff;
  crc[1]=0xff;
  for(i=0;i<size;i++)
  {
    car = buff[i];
    car ^= crc[0];
    crc[0]=crc[1] ^ TABLE2[car];
    crc[1]=TABLE1[car];
  }
  *hi=crc[0];
  *lo=crc[1];
}
```

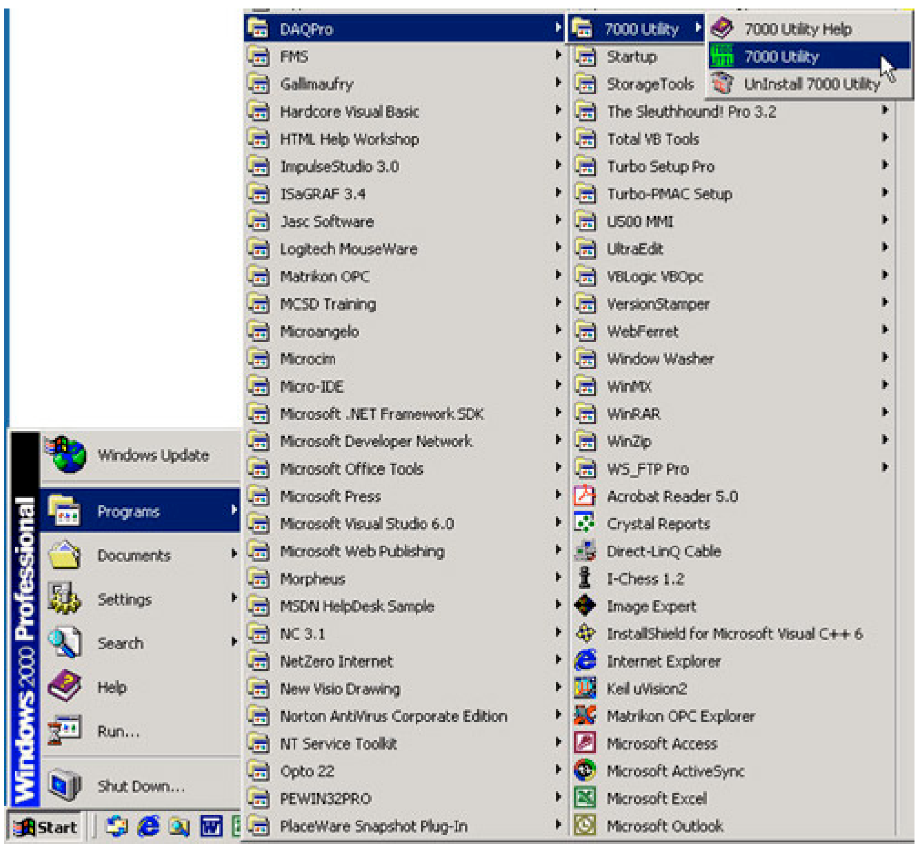
Chapter 6: Linking I-7000 & I-87xx Modules

The I-8xx7, I-7188EG/XG & W-8xx7 controller system provides the capability to integrate with ICP DAS's I-7000 and I-87xx (87K4 / 87K5 / 87K8 / 87K9) series modules. This functionality to interface with these modules expands the capability of the I-8xx7, I-7188EG/XG & W-8xx7 controller series products.

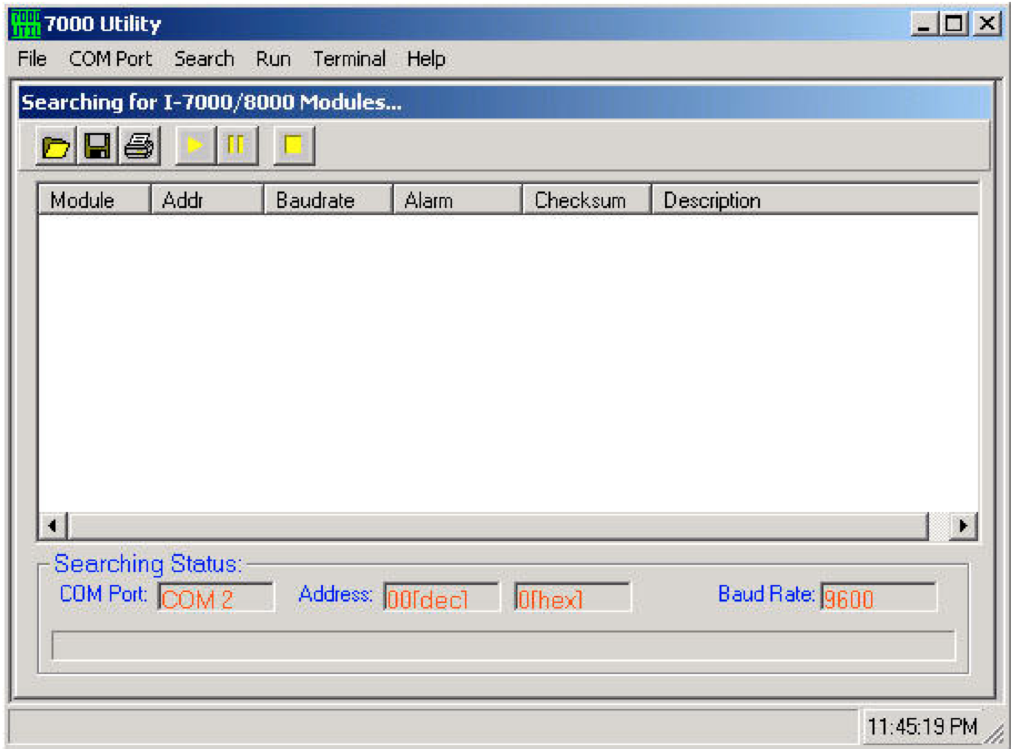
You must first make sure that the I/O libraries have been installed, please refer to Section 1.2 for Installing The "ICP DAS Utilities For ISaGRAF", and refer to Section 1.5 for connection instructions between the I-8xx7 controller system to the I-7000 and I-87xx series modules.

6.1: Configuring The I-7000 & I-87xx Modules

To begin configuration of the I-7000 and I-87xx series modules to the controller system, use the "7000 Utility" program to set up the I-7000 and I-87xx modules.



Once you have selected the "7000 Utility" program, the "7000 Utility" window will open.



The "7000 Utility" program will go out and attempt to link to any I-7000 and I-87xx modules.

IMPORTANT NOTES Regarding I-7000 & I-87xx Modules
One I-8xx7, I-7188EG/XG controller system can link up to a maximum of 64 pcs. of I-7000 and I-87xx modules(255 pcs for W-8xx7). It is recommended though that you do not link more than 40 modules to a single I-8xx7, 7188EG/XG & W-8xx7 controller system. **Each I-7000 and I-87xx module MUST have it's own unique address to properly link to an ISaGRAF controller system. Make sure to set the "Checksum" to disabled, and make sure that all of the I-7000 and I-87xx modules are set to the same baud rate as the controller system** (19200 baud by default).

When you receive any of the I-7000 series modules you will receive documentation called "Getting Started With I-7000 Series Modules" that provides instructions on how to properly configure these modules. If you need assistance on changing the baud rate or checksum, please refer to the "Change Baud Rate & Checksum" section in the "Getting Started With I-7000 Series Modules". You can find all of the documentation on the CD provided with your I-7000 series module from ICP DAS in a file titled "getstart.pdf".

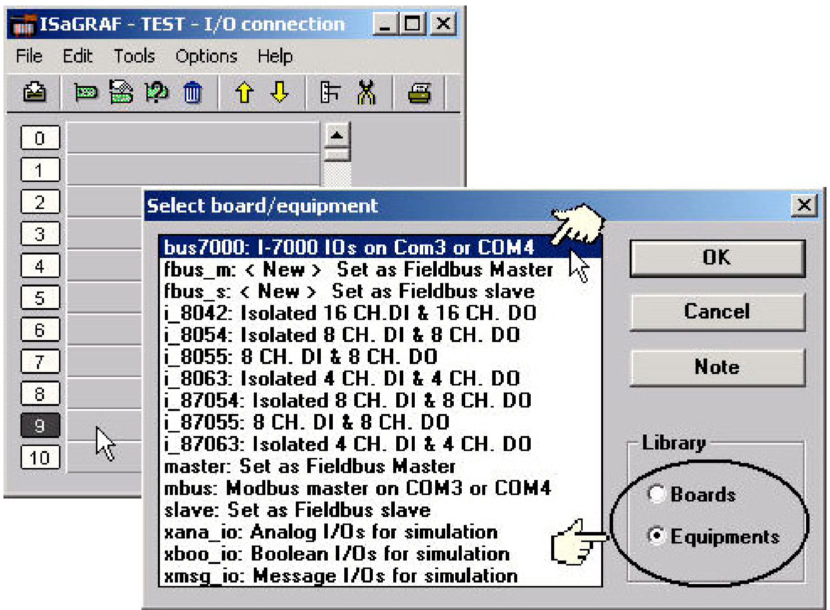
The I-7000 and I-87xx "**Analog Input**" type modules MUST have their data format set to "**2's Complement**". This includes the I-7013, I-7016, I-7017, I-7018, I-7033, I-87013, I-87017, and I-87018 analog input modules.

The I-7000 and I-87xx "**Analog Output**" type modules MUST have their data format set to "**Engineer Unit**". This includes the I-7021, I-7022, I-7024, I-87022, I-87024 and I-87026 analog output modules.

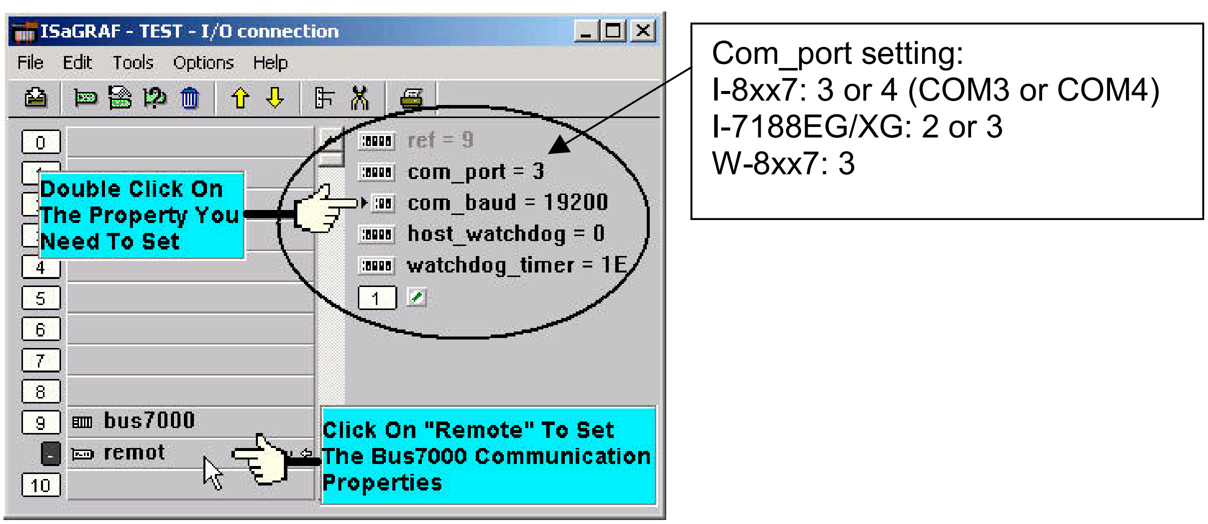
6.2: Opening The "Bus7000" Function

To create a link between the I-8xx7, I-7188EG/XG & W-8xx7 controller system and an I-7000 and I-87xx module, you need to connect the "Bus7000" function through the "ISaGRAF I/O Connection" window. The "Bus7000" function is considered a "virtual board", and must be selected from the "Equipments" section of the "Select Board/Equipment" window.

The "Bus7000" MUST be connected to slot number 8 or higher on the "ISaGRAF I/O Connection" window (since slot 0 through 7 are used to connect to real I-8000 boards). **Only one "Bus7000" can be linked to one I-8xx7, I-7188EG/XG & W-8xx7 controller system!** If you attempt to connect more than one "Bus7000" to an ISaGRAF controller, it will not work.



In the example provided, set the slot below number 9 to "Bus7000: Remote".



The "com_port" parameter can have a value of 3 (for COM3) or 4 (for COM4) for the I-8xx7 controller, while 2 (COM2) or 3(COM3) for the I-7188EG/XG, and 3 (COM3) for the W-8xx7.

This parameter defines which COM port ID the controller system will communicate with the I-7000 / I-87xx module. The default value for the "com_port" parameter is 3.

The "com_baud" parameter defines the baud rate that the I-8xx7, I-7188EG/XG & W-8xx7 will communicate with the I-7000 / I-87xx module. The possible values are 2400, 4800, 9600, 19200, 38400, 57600, and 115200. You must make sure that the controller system and the I-7000 / I-87xx modules are all set to the same "com_baud" value.

The "host_watchdog" parameter enables or disables the watchdog function for the I-7000 and I-87xx module. Setting the "host_watchdog" parameter to a non-zero value will enable the "host_watchdog" feature.

The "watchdog_timer" parameter defines the amount of time before a "host_watchdog" will occur. The value for the "watchdog_timer" is defined in a **hexadecimal** value with the units defined in 0.1-second increments. For example, if the "watchdog_timer" is set to a value of 1E, the "watchdog_timer" is set for 3 seconds. If the "watchdog_timer" value is set to 2A, the "watchdog_timer" is set for 4.2 seconds.

If the host watchdog feature is active and the watchdog timer is exceeded on the controller system (it means the connection is break between the controller and I-7000 / I-87xx modules), the I-7000 / I-87xx modules will go to a "safe" predetermined value.

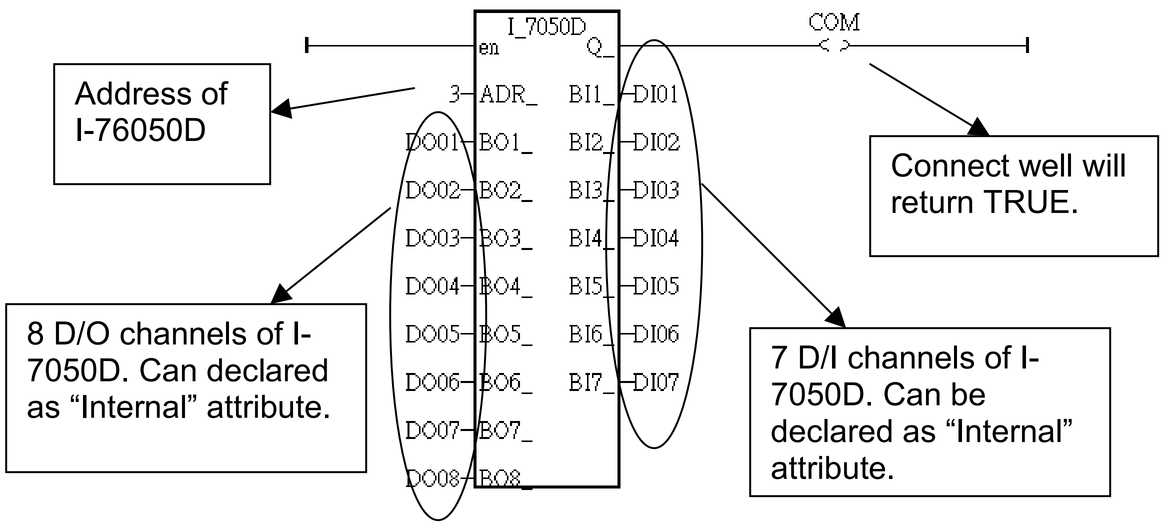
There is an analog input channel available on the "Bus7000: Remote" virtual board. This analog input channel will return a value equal to the currently set baud rate.

6.3: Programming an I-7000 Module

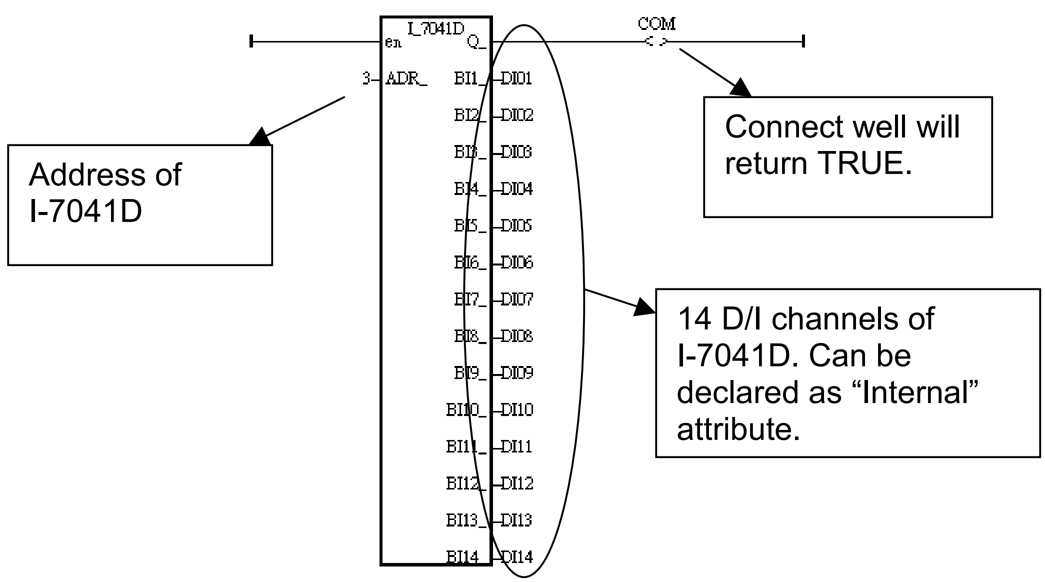
To link any I-7000 and I-87xx module to the I-8xx7, I-7188EG/XG & W-8xx7 controller system, the "Bus7000" module MUST be opened first. Once the "Bus7000" is opened, the "I_7xxx" / “I-87xx” function block can now be programmed and you can access all of the I/O channels available from that function block, and that data can now be used in a LD program.

NOTE:
You can declare all variables which connect to the I-7xxx / I-87xx function block as “Internal“ attribution.

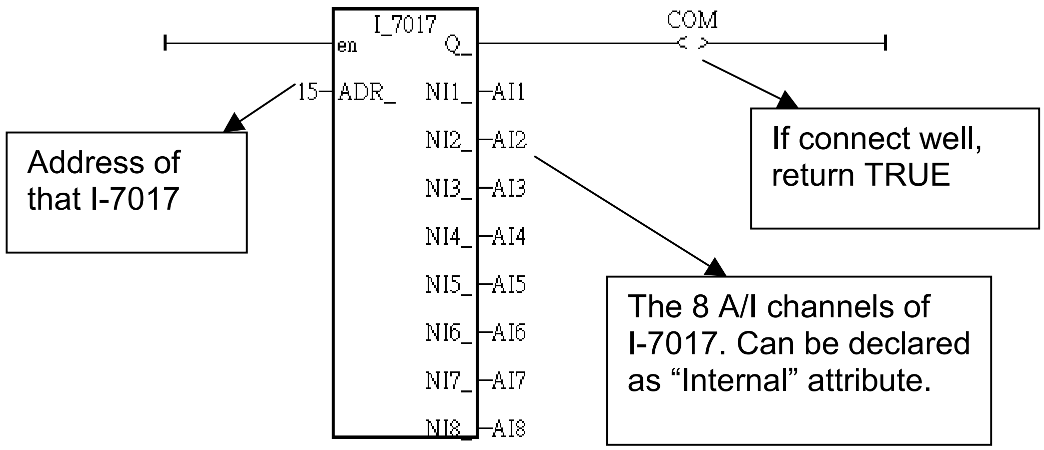
Example 1: Programming An I-7050D Module



Example 2: Programming An I-7041D Module



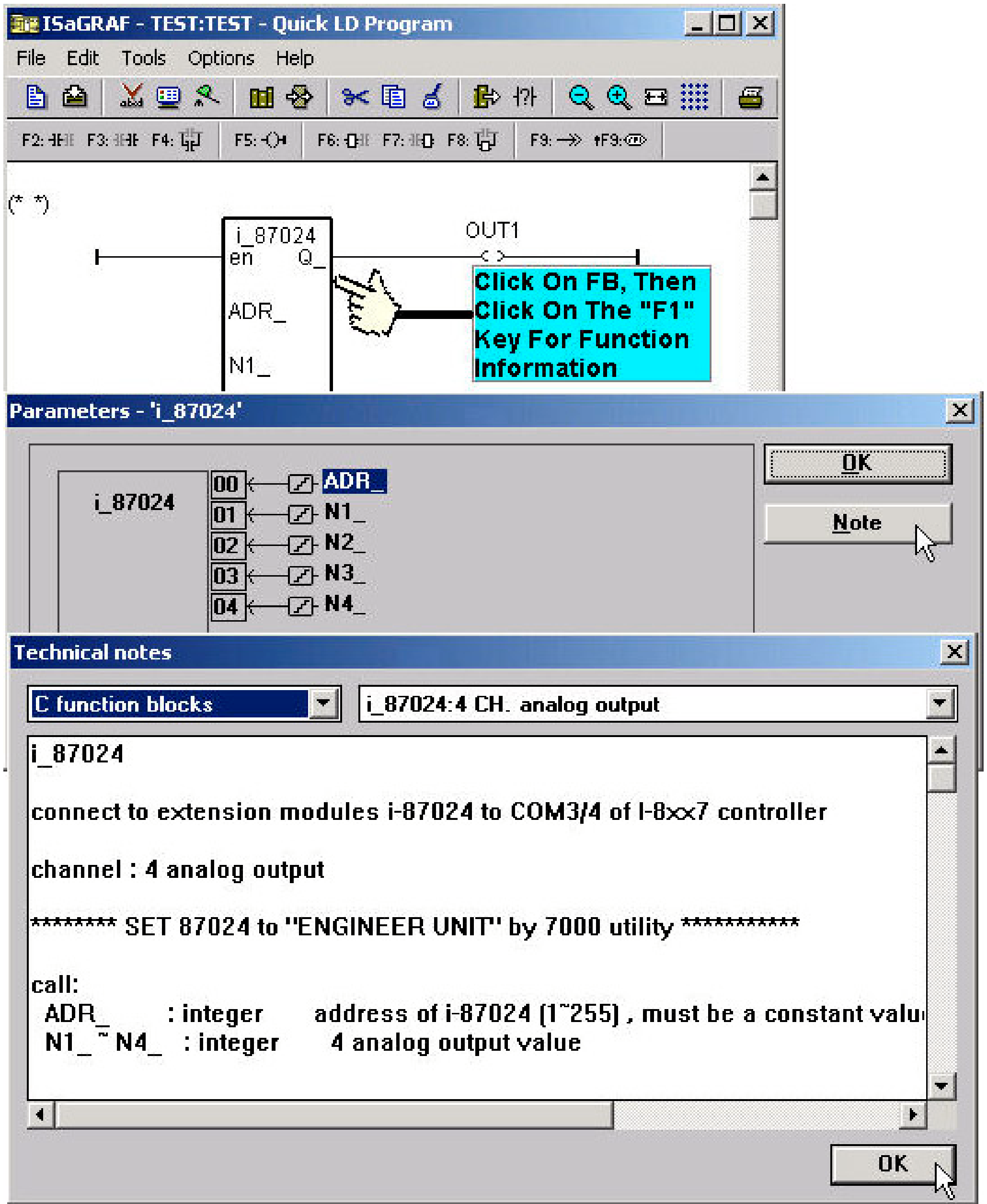
Example 3: Programming An I-7017 Module
The Data Format Used Is: 2's Complement



The following table describes the scaling factor from an analog signal to an integer value.

Range ID (set by using 7000 Utility)	Electrical range	Value in I-7017 block (decimal)		
		-32768	0	+32767
8	± 10V	- 10V	0V	+ 10V
9	± 5V	- 5V	0V	+ 5V
A	± 1V	- 1V	0V	+ 1V
B	± 500mV	- 500mV	0mV	+ 500mV
C	± 150mV	- 150mV	0mV	+ 150mV
D	± 20mA	- 20mA	0mA	+ 20mA

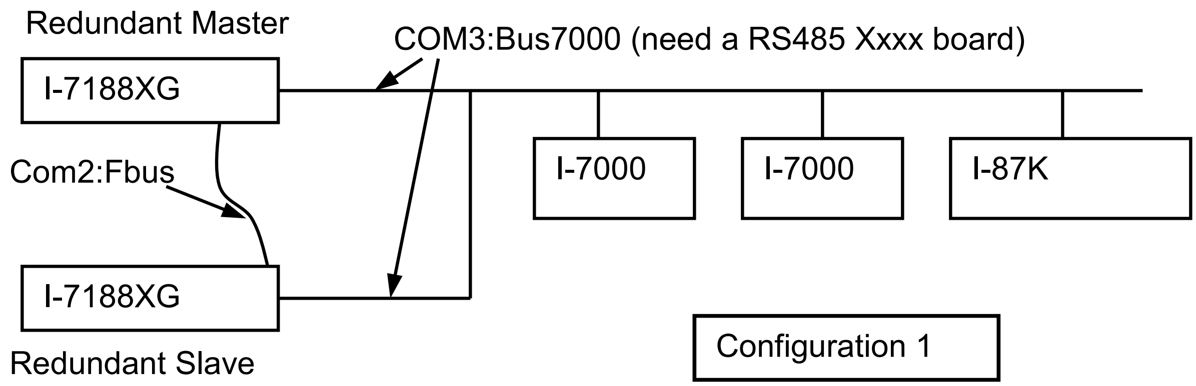
For additional information regarding any I-7000 and I-87xx module, click on the function block and press the "F1" key for an on-line description with "Technical Notes" for the selected function block.



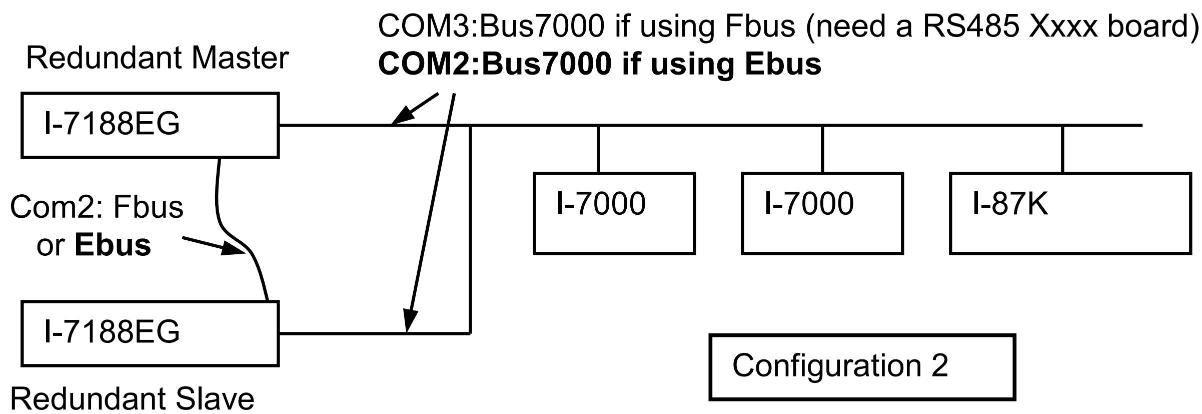
6.4: Redundant Bus7000

7188EG(Rev.1.19 or above), 7188XG(Rev.1.17 or above) & I-8417/8817/8437/8837(Rev.2.27 or above) support Redundant Bus7000. These configurations are listed as the following. The Fbus/Ebus are for exchanging data between the “Redundant Master” & “Redundant Slave”, and the **Fbus/Ebus cable** must be always working(break is not allowed).

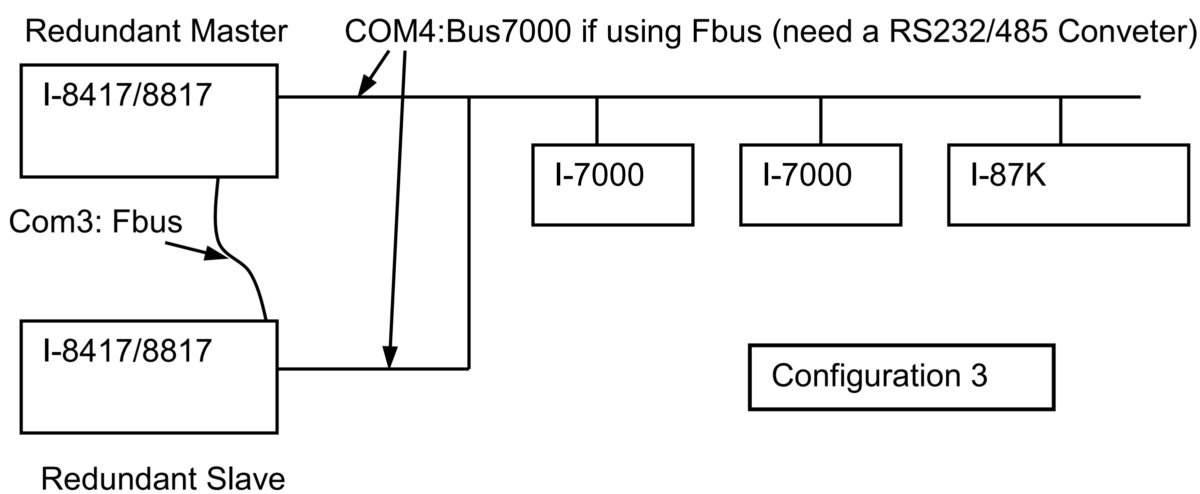
I-7188XG:



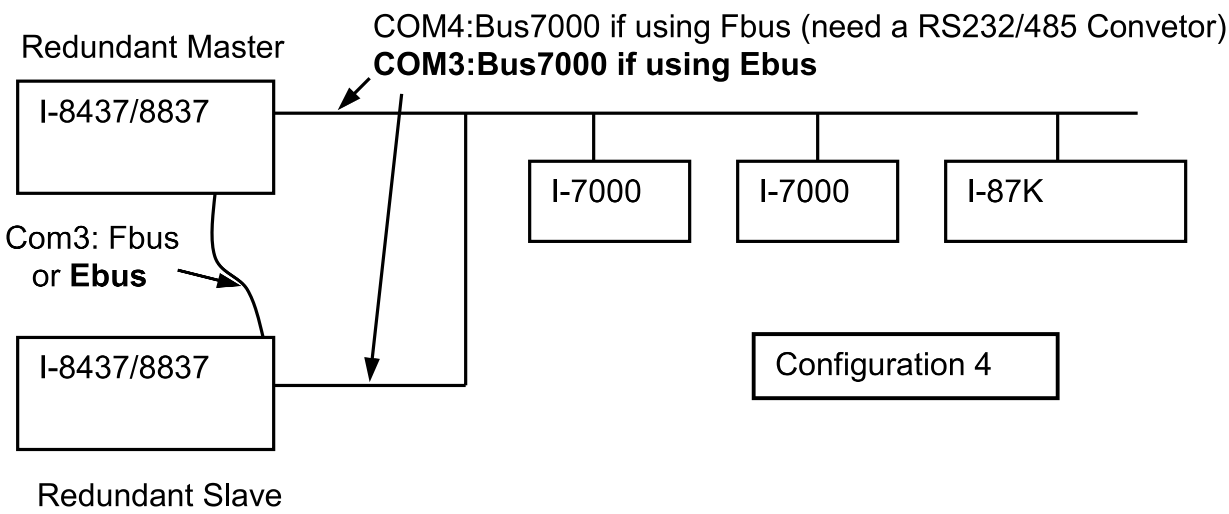
I-7188EG:



I-8417/8817:

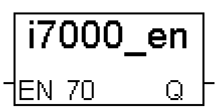


I-8437/8837:



Operations Principle:
When the system is powered up, the control right of Bus7000 belong to "Redundant Master".
If "Redundant Master" is dead(Power off), "Redundant Slave" takes over the control right of Bus7000.
If "Redundant Master" is alive from dead (power up again), it takes over the control of Bus7000 again.
User's control data is exchanging via Fbus or Ebus.

The "i7000_en" can be used to Enable/Disable the control right of Bus7000. The system's default status is Enable.



Parameter:

EN_7000_	integer	True: Enable, False: Disable

Return:

Q_	boolean	Always return True.

Demo example for I-7188XG:
The demo project uses "Configuration 1" and located at **demo_48a** & **demo_48b**.
It can be download at ICP DAS's ftp site.
ftp://ftp.icpdas.com/pub/cd/8000cd/napdos/isagraf/7188xg/demo/

Demo example for I-7188EG:
The demo project uses "Configuration 2" with Ebus and located at **demo_51a** & **demo_51b**.
It can be download at ICP DAS's ftp site.
ftp://ftp.icpdas.com/pub/cd/8000cd/napdos/isagraf/7188eg/demo/

Demo example for I-8437/8837:
The demo project uses "Configuration 4" with Ebus and located at **demo_49a** & **demo_49b**.
It can be download at ICP DAS's ftp site.
ftp://ftp.icpdas.com/pub/cd/8000cd/napdos/isagraf/8000/demo/

Chapter 7: Controller To Controller Data Exchange

The I-8xx7, I-7188EG/XG & W-8xx7 controller system provides the capability of exchanging data with other I-8xx7, I-7188EG/XG & W-8xx7 controller systems. For this functionality to work properly you must make sure that the I/O libraries MUST be installed. If you haven't installed it already, please refer to Section 1.2 for Installing The "ICP DAS Utilities For ISaGRAF" and Section 1.4 for creating a Fbus connection between I-8xx7 controller systems.

Important Note:
The max. boolean & integer package No. of Fbus & Ebus reduce from 256 to 128 since driver version of I-8417/8817/8437/8837:2.42 , I-7188EG:1.32 & I-7188XG:1.29

7.1: Basic Fbus Rules

Any I-8xx7 & I-7188EG/XG controller system can access data from another I-8xx7 & I-7188EG/XG through the Fbus data exchange system. **While Wincon-8xx7 doesn't support Fbus, it supports Ebus only. Please refer to section 7.5 for programming Ebus on I-8437/8837, I-7188EG & W-8037/8337/8737**. There are 2 types of data that can be exchanged through the Fbus protocol; they are "Boolean" and "integer".

The Fbus driver first creates a packet of eight Boolean values to form a "Boolean package", and then creates a packet of eight 32-bit integers to form an "integer package". Both of the "Boolean packages" and "integer packages" can be distributed on the Fbus to allow the data to be exchanged from one I-8xx7 & I-7188EG/XG controller system to another I-8xx7 & I-7188EG/XG controller system.

The Following Fbus Rules MUST Be Observed:

RULE #1: Each "Boolean package" must have an attached identification number ranging from 1 to 128. This means that there is a maximum of 128 "Boolean packages" that can be exchanged across an Fbus connection.

Each "Boolean package" contains 8 Boolean values, and these Boolean values can only have the value of either "True" or "False". The Boolean values in the "Boolean package" can be assigned and exchanged with either "Internal", "Input", or "Output" Boolean variables or Boolean constants.

RULE #2: Each "integer package" must have an attached identification number ranging from 1 to 128. This means that there is a maximum of 128 "integer packages" that can be exchanged across an Fbus connection.

Each "integer package" contains eight 32-bit integer values. The integer values can range from –2147483648 to 2147483647. The integer values in the "integer package" can be assigned and exchanged with either "Internal", "Input", or "Output" integer variables or integer constants.

Rule #3: Each number assigned to a "Boolean package" or an "integer package" can only be written to by one I-8xx7 & I-7188EG/XG controller system across the Fbus.

Each I-8xx7 & I-7188EG/XG controller system CANNOT **write** the same identification number for either a "Boolean package" or an "integer package" across the Fbus. WRITTING A PACKAGE IS NOT SHARED with the other I-8xx7 & I-7188EG/XG controller systems across the Fbus network.

In this example, there are five I-8xx7 controller systems communicating through an Fbus network, and the controller systems are named S1, S2, S3, S4, and S5 respectively. If the S1 controller system attempts to write a "Boolean package" with an ID of "1" and an "integer package" with an ID of "1" across the Fbus, the other four controllers CANNOT write either a "Boolean package" or an "integer package" with the same number. However, the other controller systems could write a "Boolean package" with an ID of "3" and an "integer package with an ID of "2".

There is no limitation on how many I-8xx7 & I-7188EG/XG controllers can read the same number package across the Fbus network. Any of the S2, S3, S4, S5 controller systems can read the "Boolean package" with an ID of "1" and the "integer package" with an ID of "1" if desired.

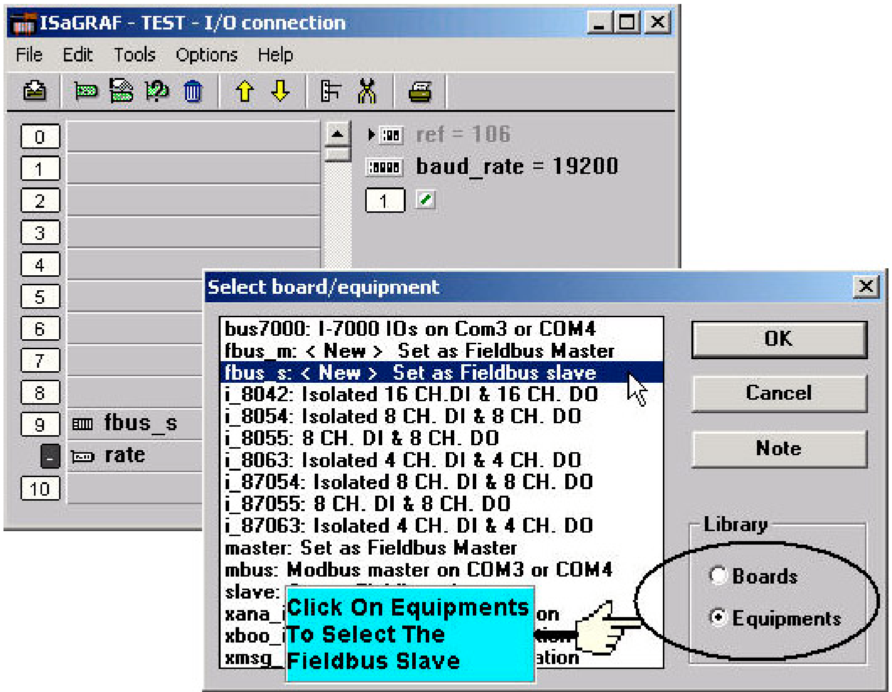
Rule #4: ONLY ONE I-8xx7 or I-7188EG/XG controller system can be configured as a Fbus "Master", all the others I-8xx7 & I-7188EG/XG controller systems MUST be configured as a Fbus "Slave".

The "master" controller sends commands for how data is to be exchanged across the Fbus network. If you configure more than one I-8xx7 or I-7188EG/XG controller system as a "master", or configure none of the I-8xx7 & I-7188EG/XG controller systems as a "master" on the Fbus, NO DATA CAN BE EXCHANGED across the Fbus network.

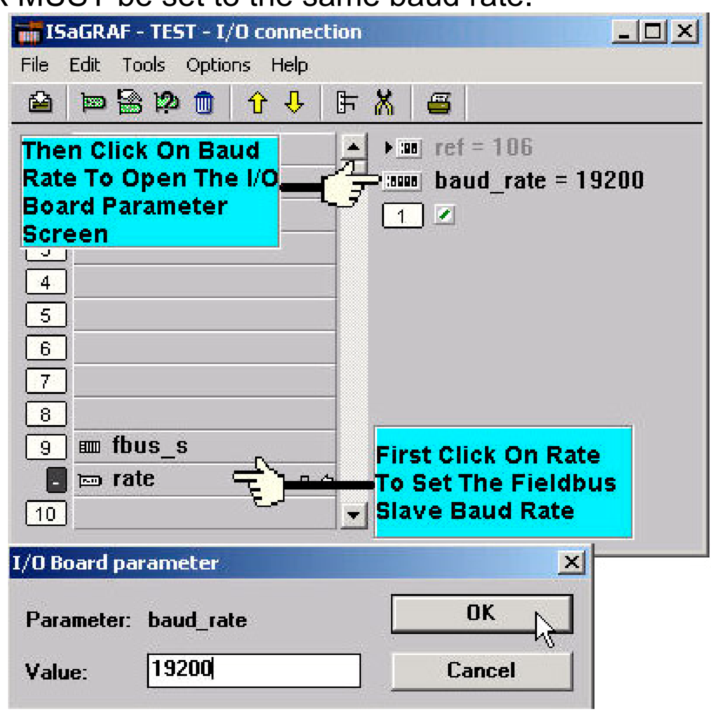
Important Note:
The max. boolean & integer package No. of Fbus & Ebus reduce from 256 to 128 since driver version of I-8417/8817/8437/8837:2.42 , I-7188EG:1.32 & I-7188XG:1.29

7.2: Configuring An I-8xx7 To Be A Fbus "Master" Or "Slave"

To begin configuring an I-8xx7 & I-7188EG/XG controller system as either a Fbus master or slave, first open up the "ISaGRAF I/O Connections" window and double click on a slot number higher than 7. The "Select Board/Equipments" window will now open, click on "Equipments", and then double click on the "fbus_s" selection to configure an Fbus slave, or double click on "fbus_m" to configure an Fbus master. Remember, **ONLY ONE** controller can be the Fbus master, and you **CANNOT** configure an I-8xx7 & I-7188EG/XG controller system to be both a Fbus master and a Fbus slave.

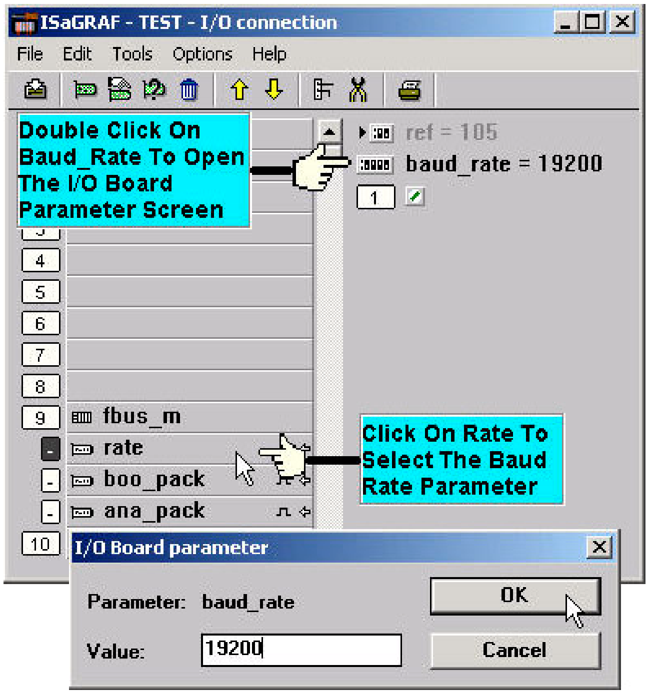


If you configure an I-8xx7 & I-1788EG/XG controller system as an Fbus slave, only one parameter needs to be set, and that is the "baud_rate" parameter. The baud rate parameter can be set to 2400, 4800, 9600, 19200, 38400, 57600 or 115200 baud rate. The default baud rate value is 19200 for the I-8xx7 & I-7188EG/XG controller system. All controllers on the same Fbus network MUST be set to the same baud rate.



There is a digital input channel associated with the "fbus_s: rate" equipment. This function will return the status when opening up an Fbus connection. If the Fbus connection has been established, the digital input channel will return a "TRUE" value. If the Fbus connection failed to establish, the digital input channel will return a "FALSE" value.

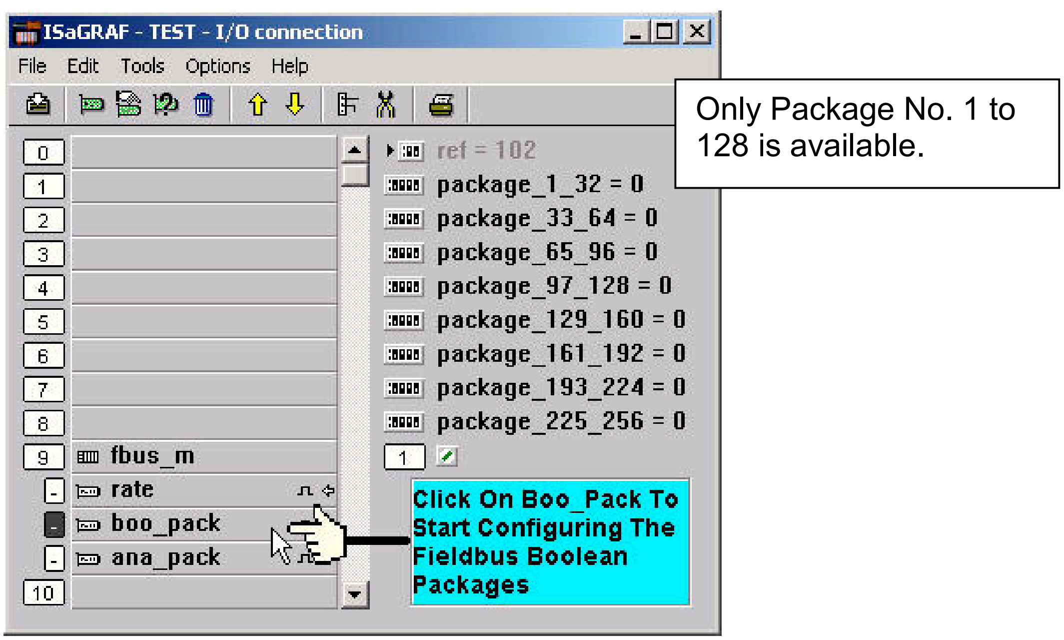
If you configure an controller as Fbus master, the parameter "baud_rate" and "fbus_m: rate" can be set to 2400, 4800, 9600, 19200, 38400, 57600 or 115200. The default value is 19200 for the controller. All controllers on the same Fbus MUST be set to the same baud rate.



There is a digital input channel associated with the "fbus_m: rate" equipment. This function will return the status when opening up an Fbus connection. If the Fbus connection has been established, the digital input channel will return "TRUE" value, if the Fbus connection failed to establish, the digital input channel would return a value of "FALSE".

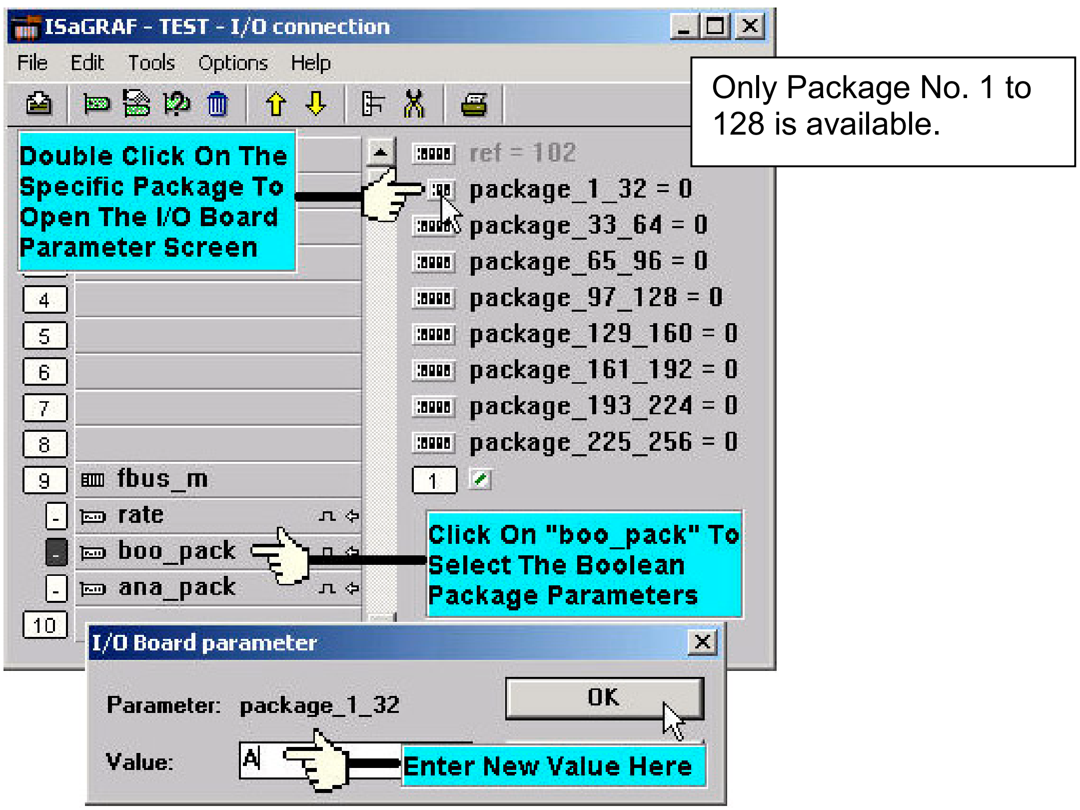
7.2.1: Configuring The Fbus Master Boolean Packages

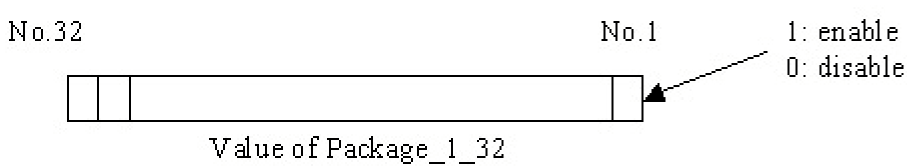
To begin configuring the Fbus Master Boolean Packages, click on the "boo_pack" selection from the "fbus_m" I/O connection.



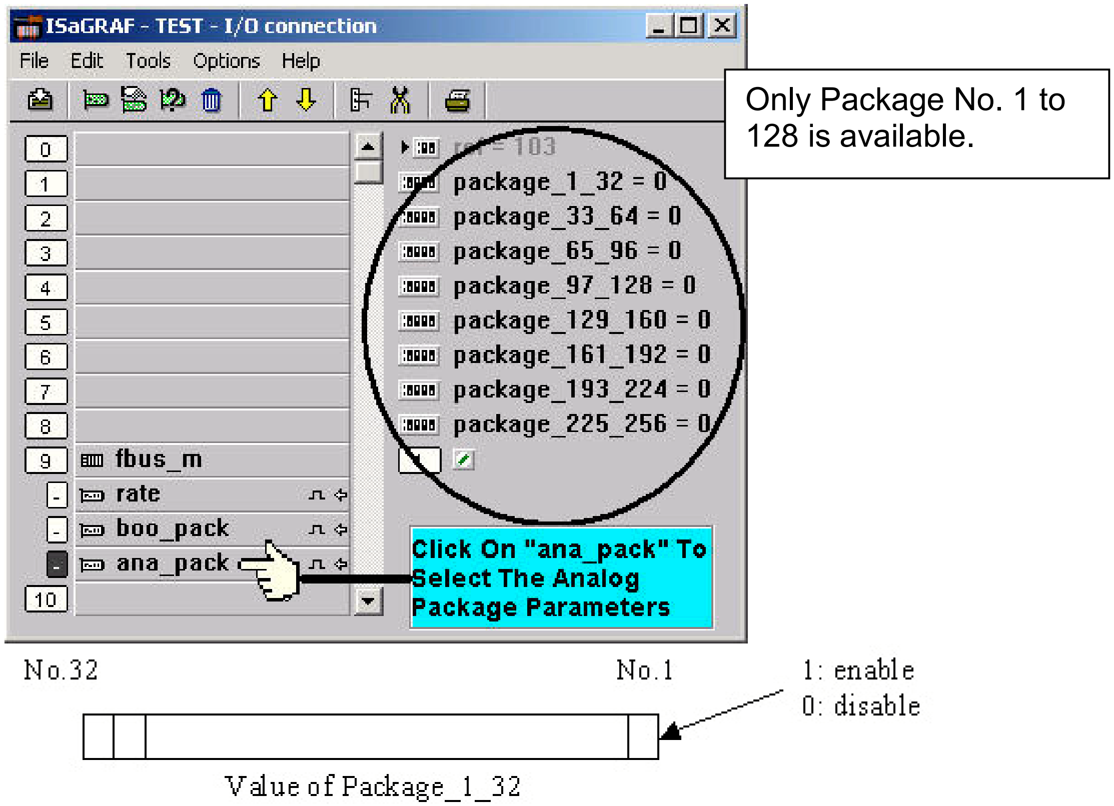
The parameter "package_xxx_xxx" at "fbus_m: boo_pack" indicates the "Boolean package" number which is allowed to be written to or read from across the Fbus network. The parameter value is given as a 32-bit integer in **hexadecimal**.

As an example, if the "package_1_32" is set to "FFFFFFFF" this will enable all the packages from number 1 to number 32 to be written to or read from across the Fbus network. If the "package _1_32" is set to a value of "A", this will only enable the number 2 and number 4 Boolean packages to be written to or read from across the Fbus network. The more packages that are enabled on a Fbus network the slower the communication efficiency will be. **With this in mind, always remember to enable only the required number of packages that you need for your application so you will have greater communication efficiency across the Fbus network**.





The parameter "package_xxx_xxx" at "fbus_m: ana_pack" indicates the "integer package" number which will be written to and read from on the Fbus network. The "fbus_m: ana_pack" is used to read and write 32-bit integer values across the Fbus network. Each of the parameter values is expressed as 32-bit integer values in **hexadecimal**, and the same configuration rules apply as those for the "Boolean package".



7.3: Programming Fbus Packages

Before you can exchange any data across a Fbus network, you must make sure that each I-8xx7 & I-7188EG/XG is either configured as either a Fbus master "fbus_m" (and remember, only ONE controller can be the master) or Fbus slave "fbus_s". Refer to Section 7.2 for details on how to implement these configurations.

The following Fbus function blocks can be used in a LD program to exchange data across an Fbus network.

Fbus_b_r	read one boolean package.
Fbus_b_w	write one boolean package.
Fbus_n_r	read one integer package.
Fbus_n_w	write one integer package.

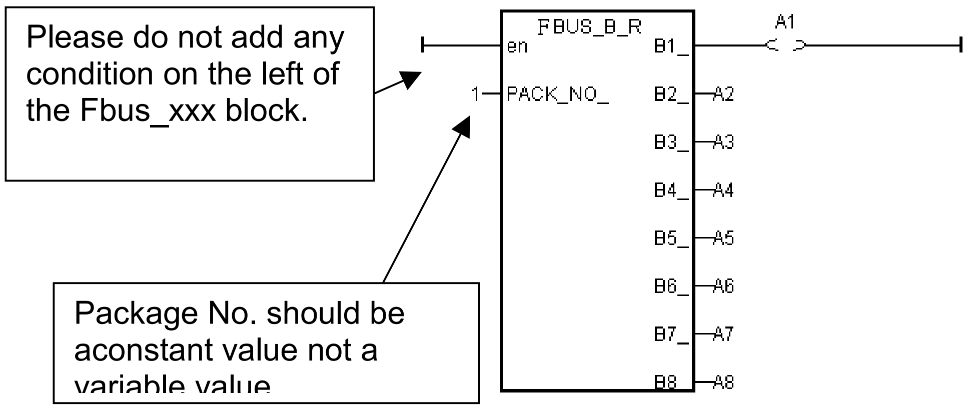
The below two blocks can be used to exchange “real” value via Fbus.
Block “Real_Int” can be used to Map a “real” value to a 32-bit integer. So that you can deliver this integer to the Fbus, and then on the receiver controller, use “Int_Real” to map this integer back to the original “real” value.

Int_Real	Map a long integer to a Real value.
Real_Int	Map a Real value to a long integer.

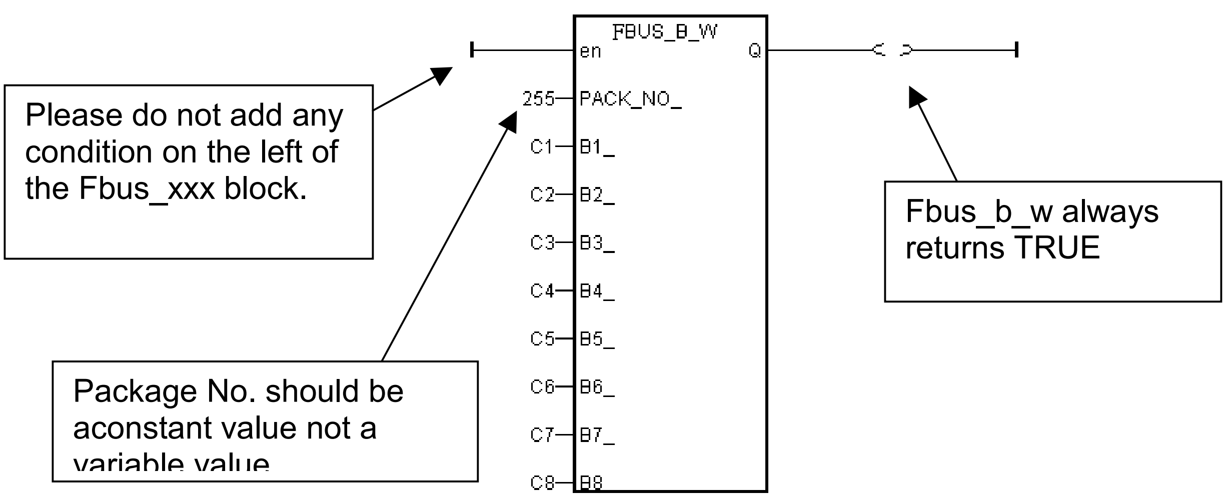
The below block is to get the communication ststus of each Boolean & Integer Package.

Fbus_sts	Get ststus of each Package.

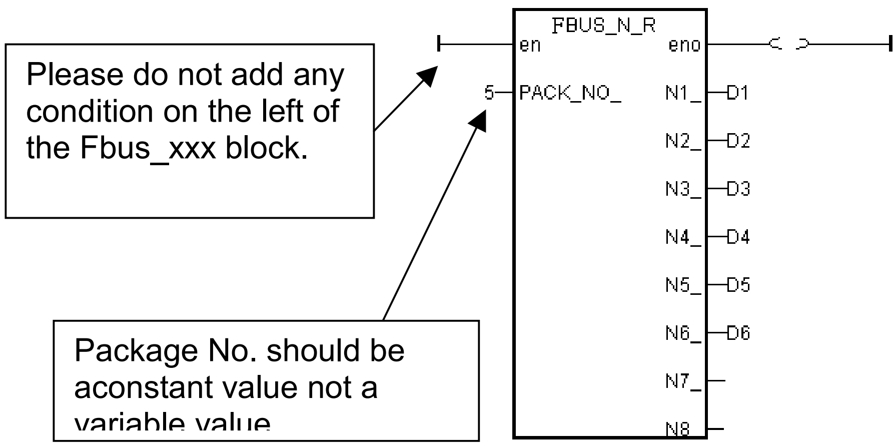
Fbus Function #1: "Fbus_b_r"
The "Fbus_b_r" function reads one Boolean package from the Fbus network. In the example below the "Fbus_b_r" function has a Boolean package ID address of "1". The "A1" output contains the value of the first Boolean of the package No. of 1, the "A2" output contains the value of the second Boolean of the package No. of 1, and the "A3" output contains the value of the third Boolean of the package No. of 1. The other outputs follow the same format to where the "A8" output contains the value of the eighth Boolean of the package No. of 1.



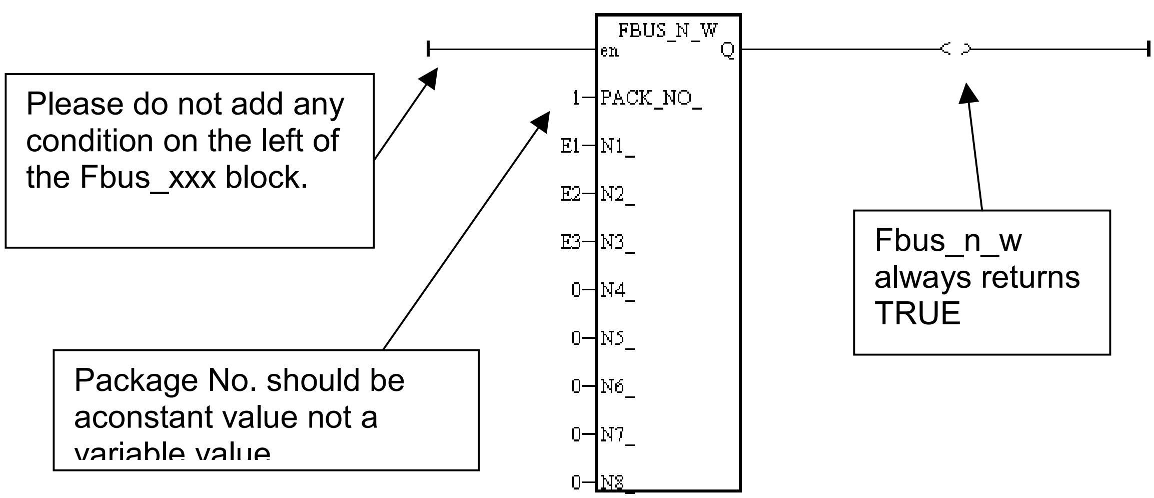
Fbus Function #2: "Fbus_b_w"
The "Fbus_b_w" function writes one Boolean package on the Fbus network. In the example below the "Fbus_b_w" function has a Boolean package ID address of "255", the "C1" input writes a value to the first Boolean of the package No. of 255, the "C2" input writes a value of the second Boolean of the package No. of 255, and the "C3" input writes a value of the third Boolean of the package No. of 255. The other inputs follow the same format to where the "C8" input writes a value of the eighth Boolean of the package No. of 255.



Fbus Function #3: "Fbus_n_r"
The "Fbus_n_r" function reads one integer package from the Fbus network. In the example below the "Fbus_n_r" function has an Integer package ID address of "5". The "D1" output contains the value of the first integer of the package No. of 5, the "D2" output contains the value of the second integer of the package No. of 5, and the "D3" output contains the value of the third integer of the package No. of 5. The other outputs follow the same format to where the "D6" output contains the value of the sixth integer of the package No. of 5.



Fbus Function #4: "Fbus_n_w"
The "Fbus_n_w" function writes one integer package to the Fbus network. In the example below the "Fbus_n_w" function write variables “E1” to the first integer of the package of No. 1. “E2” to the second integer of the package of No. 1. “E3” to the third integer of the package of No. 1.



7.4: An Fbus Data Exchange Example

Example Description:
In this Fbus data exchange example there are three I-8xx7 controller systems linked together in an Fbus network. The I-8xx7 controller systems are named "SA (master I-8xx7 controller system #1)", "SB (slave I-8xx7 controller system #2), and "SC (slave I-8xx7 controller system #3).

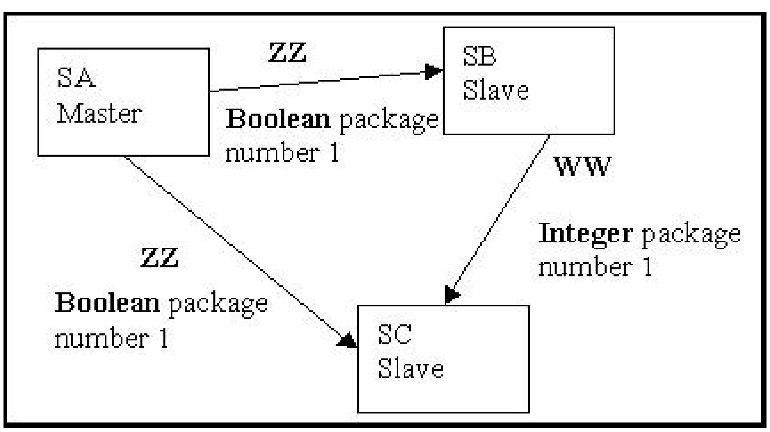
One of the digital input values from the SA controller (master I-8xx7 system) needs to be shared with the SB and SC (the slave I-8xx7 systems) controllers across the Fbus network, and the name for this digital input value will be called "ZZ".

The first task of this example is to create an **Input** variable named ZZ on the SA controller system. Use the "ISaGRAF Project" window to declare ZZ as an "input" variable, and then link the ZZ input variable using the "ISaGRAF I/O Connections" window for the SA controller system.

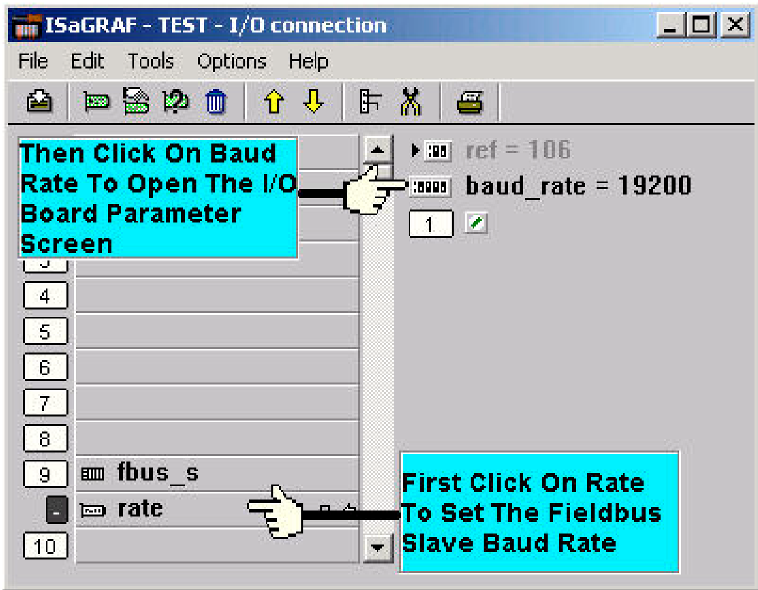
Next, you will need to declare a Boolean **Internal** variable named ZZ for both the SB and SC controllers (so they can exchange the ZZ value with the SA controller system). You must declare the ZZ variable as an internal variable for the SB and SC controllers because there is only one real input variable (from the SA controller) that is being exchanged, and neither the SB or SC has a real input variable named ZZ.

An additional requirement for this example is that an internal integer value named "WW" that comes from the SB controller system needs to be shared with the SC controller system. To accomplish this declare an **Internal** integer variable named WW on both the SB and SC controller systems.

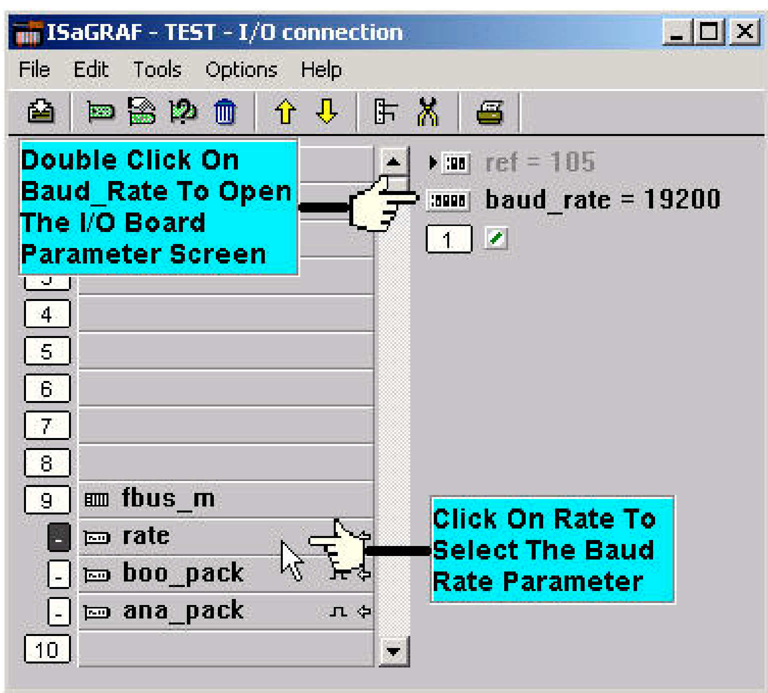
Example Prerequisites:
The SA controller system is the Fbus master controller and the SB and SC controllers are Fbus slave controllers. Each of the controllers has their baud rates set to 19200.



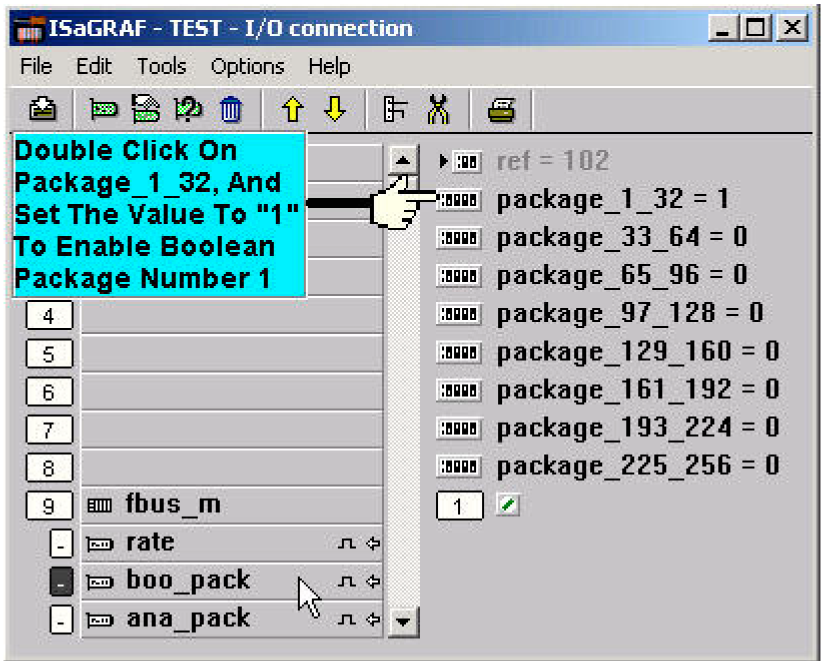
Setting The SB and SC Controllers As Fbus Slaves:
You should use the "ISaGRAF I/O Connections" window to declare the SB and SC controller systems as Fbus slaves.



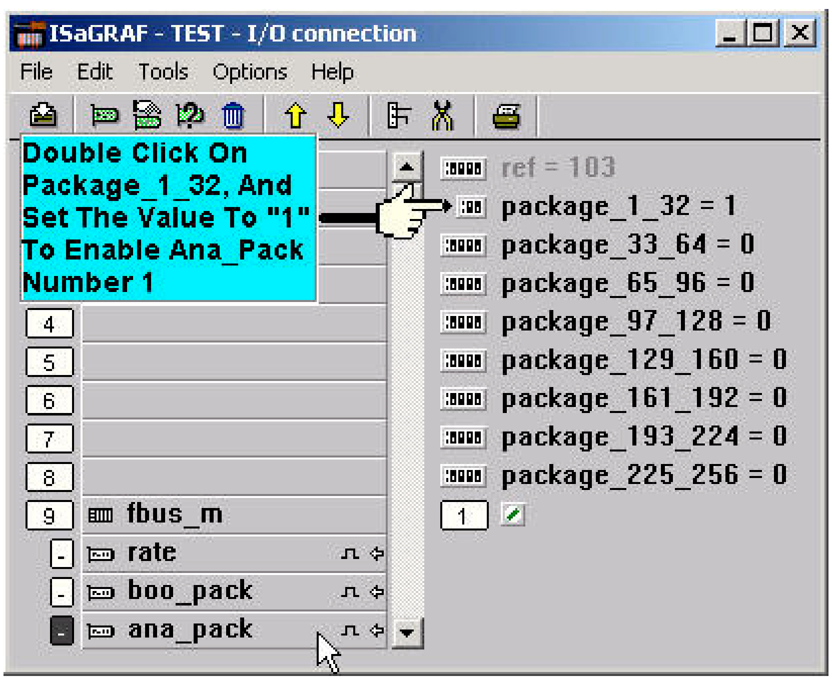
Use the "ISaGRAF I/O Connections window to declare the SA controller system as the Fbus master controller.



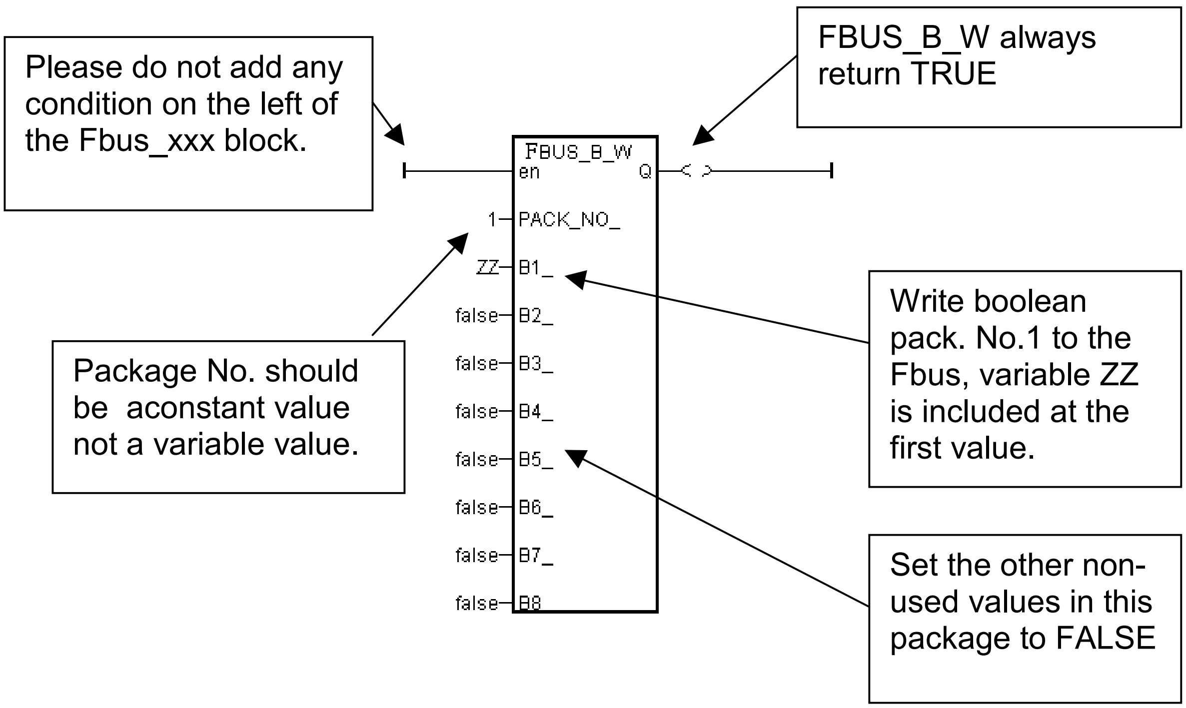
Additionally, enable the Boolean package for the SA controller:



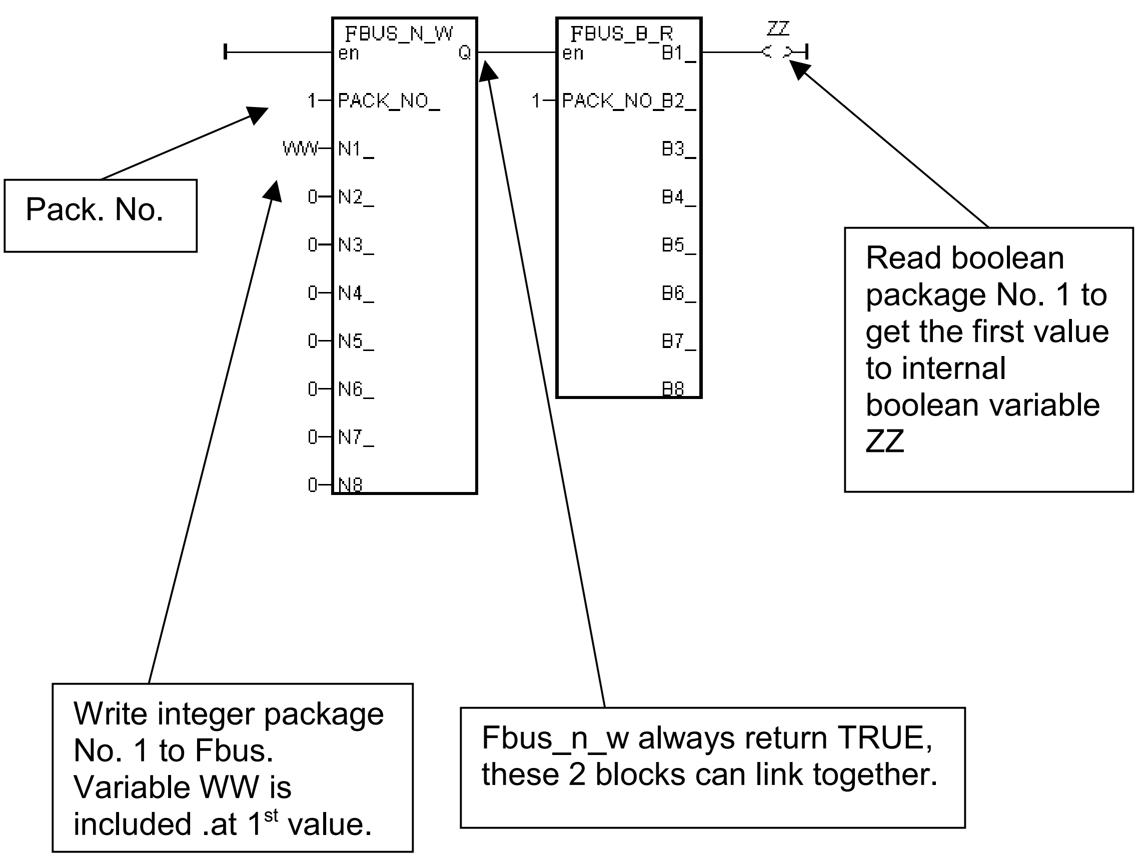
Also enable the integer package for the SA controller system:



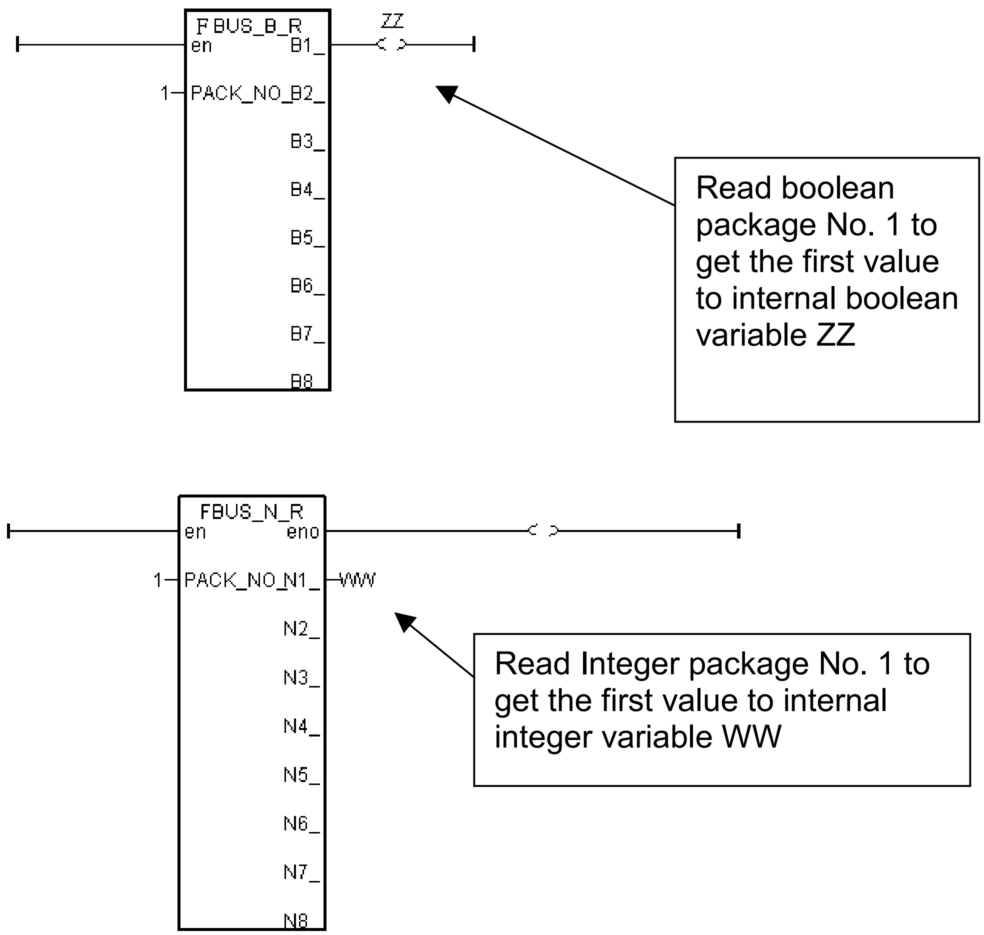
The ISaGRAF LD Project For The SA Controller:



The ISaGRAF LD Project For The SB Controller:



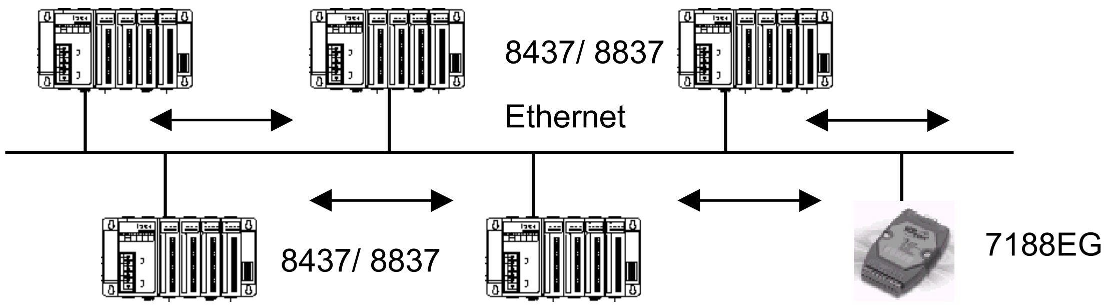
The ISaGRAF LD Project For The SC Controller:



7.5: Programming The Ebus

Ebus is a software mechanism which allows I-8437/8837, I-7188EG & W-8037/8337/8737 controllers to access data to each other through the ethernet port. Ebus is only working on the local area. That means exchanging data through a gateway is no possible.

Important Note:
The max. boolean & integer package No. of Fbus & Ebus reduce from 256 to 128 since driver version of I-8417/8817/8437/8837:2.42 , I-7188EG:1.32 & I-7188XG:1.29



The I-8437, I-8837 controllers support Ebus since its driver version of 2.15 and the I-7188EG support Ebus since its driver version of 1.08. And W-8037/8337/8737 support Ebus since its driver version of 3.10. Please refer to Appendix C to make sure your I-8xx7's controller driver version is the same or higher. You can obtain the new released driver from:

http://www.icpdas.com/products/8000/isagraf.htm

7.5.1: Basic Ebus Rules

The I-8437/ 8837, I-7188EG & W-8037/8337/8737 Ebus driver first creates a packet of eight Boolean values to form a "Boolean package", and then creates a packet of eight 32-bit integers to form an "integer package". Both of the "Boolean packages" and "integer packages" can be distributed on the Ebus to allow the data to be exchanged from one controller to another controller.

The basic Ebus rules are similiar as Fbus (refer to 7.1) as below.

RULE #1: Each Ebus network is identified with a "Group_No" ranging from 1 to 10. Data is only exchangable with controllers that are assigned with the same "Group No".

For example, there are 5 controllers located at the same local ethernet area, named A1, A2, A3, A4, A5 respectively. A1, A2 & A3 are assigned with Ebus: Group_No = 1 while A4 & A5 are assigned with Ebus: Group_No = 2. Therefore, A1 can access data from A2 & A3 however can not access data from A4 & A5.

RULE #2: Each "Boolean package" in the same Ebus:Group_No must have an attached identification number ranging from 1 to 128. This means that there is a maximum of 128 "Boolean packages" that can be exchanged across an Ebus:Group_No connection.

Each "Boolean package" contains 8 Boolean values, and these Boolean values can only have the value of either "True" or "False". The Boolean values in the "Boolean package" can be assigned and exchanged with either "Internal", "Input", or "Output" Boolean variables or Boolean constants.

RULE #3: Each "integer package" in the same Ebus:Group_No must have an attached identification number ranging from 1 to 128. This means that there is a maximum of 128 "integer packages" that can be exchanged across an Ebus:Group_No connection.

Each "integer package" contains eight 32-bit integer values. The integer values can range from –2147483648 to 2147483647. The integer values in the "integer package" can be assigned and exchanged with either "Internal", "Input", or "Output" integer variables or integer constants.

Rule #4: Each number assigned to a "Boolean package" or an "integer package" can only be written to by one I-8437/ 8837 (or I-7188EG or W-8037/8337/8737) controller system across the same Ebus:Group_No network.

Each I-8437/ 8837, I-7188EG or W-8037/8337/8737 controller CANNOT **write** the same identification number for either a "Boolean package" or an "integer package" across the same Ebus:Group_No. WRITTING A PACKAGE IS NOT SHARED with the other controller across the same Ebus:Group_No network.

In this example, there are five controllers communicating through an Ebus:Group_No network, and the controllers are named S1, S2, S3, S4, and S5 respectively. If the S1 controller attempts to write a "Boolean package" with an ID of "1" and an "integer package" with an ID of "1" across the Ebus:Group_No, the other four controllers CANNOT write either a "Boolean package" or an "integer package" with the same number. However, the other controllers could write a "Boolean package" with an ID of "3" and an "integer package with an ID of "2".

There is no limitation on how many controllers can read the same number package across the same Ebus:Group_No network. Any of the S2, S3, S4, S5 controllers can read the "Boolean package" with an ID of "1" and the "integer package" with an ID of "1" if desired.

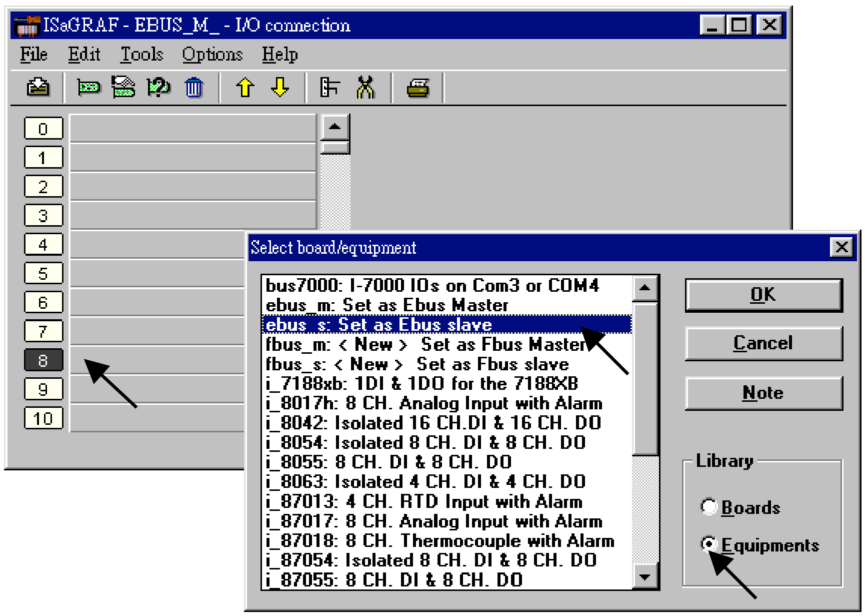
Rule #5: ONLY ONE I-8437/ 8837, I-7188EG or W-8037/8337/8737 controller in the same Group_No can be configured as a Ebus "Master", all the others controller in the same Group_No MUST be configured as a Ebus "Slave".

The "master" controller sends commands for how data is to be exchanged across the same Ebus:Group_No network. If you configure more than one controller as a "master", or configure none of the controllers as a "master", NO DATA CAN BE EXCHANGED across the Ebus:Group_No network.

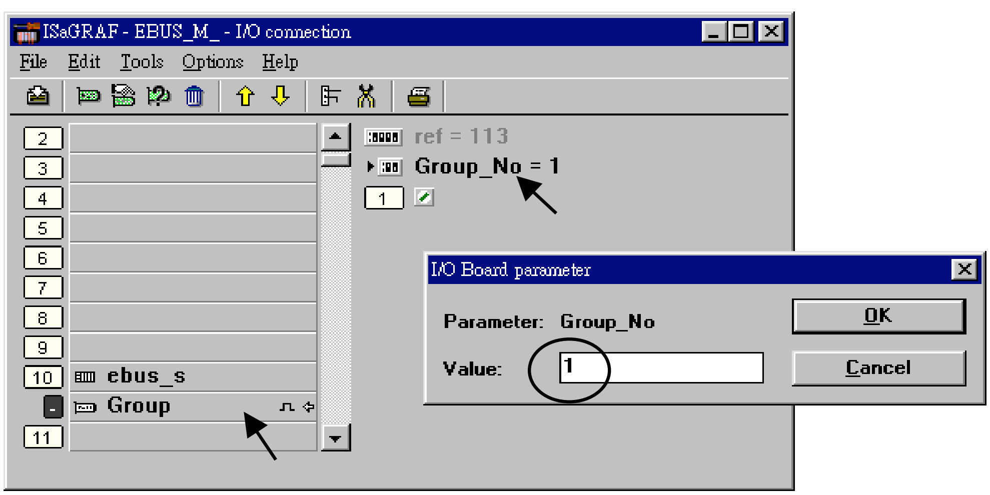
Important Note:
The max. boolean & integer package No. of Fbus & Ebus reduce from 256 to 128 since driver version of I-8417/8817/8437/8837:2.42 , I-7188EG:1.31 & I-7188XG:1.28

7.5.2: Configuring the Controller To Be A Ebus "Master" Or "Slave"

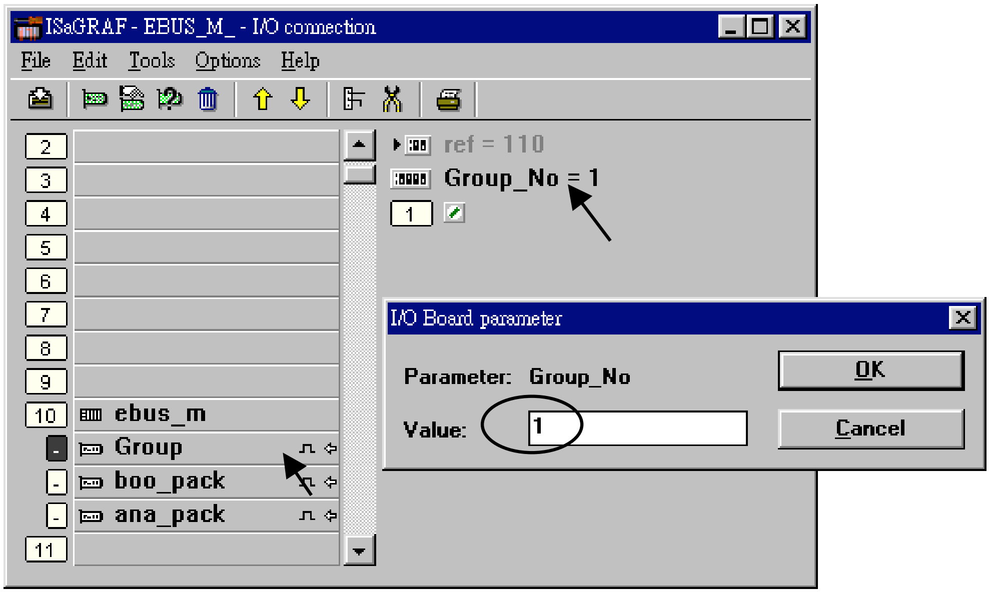
To begin configuring an I-8437/ 8837, I-7188EG or W-8037/8337/8737 controller system as either a Ebus master or a slave, first open up the "ISaGRAF I/O Connections" window and double click on a slot number higher than 7. The "Select Board/Equipments" window will now open, click on "Equipments", and then double click on the "Ebus_s" selection to configure an Ebus slave, or double click on "Ebus_m" to configure an Ebus master. Remember, **ONLY ONE** I-8437/ 8837, I-7188EG or W-8037/8337/8737 controller system can be the Ebus master, and you **CANNOT** configure an controller to be both a master and a slave.



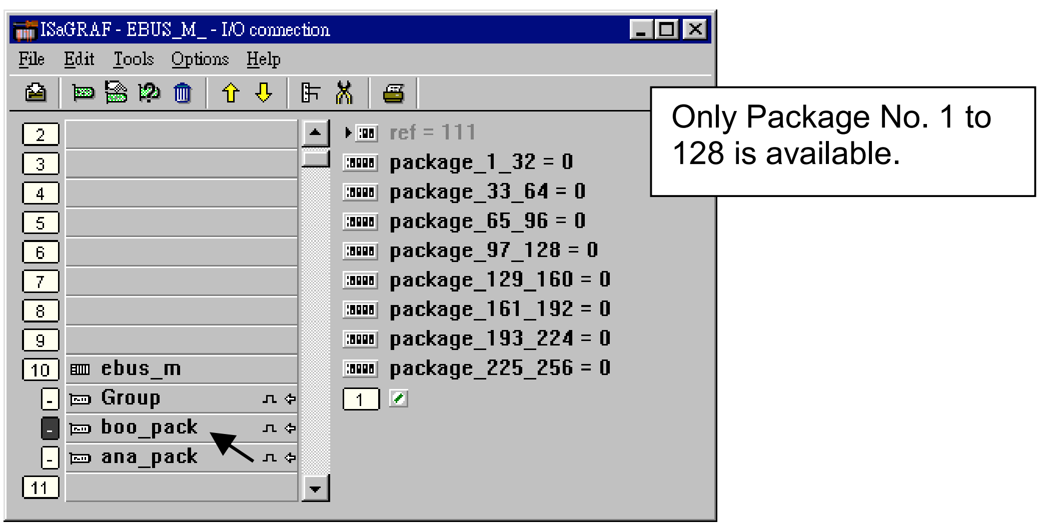
If you config a controller as an Ebus slave, only one parameter needs to be set, the "Group_No". The valid value is ranging from 1 to 10. Set to other value will become a default value , 1.



If you config a controller as an Ebus master, the parameter “Group_No” should be set to the same as the salve. The valid value is ranging from 1 to 10. Set to other value will become a default value , 1.



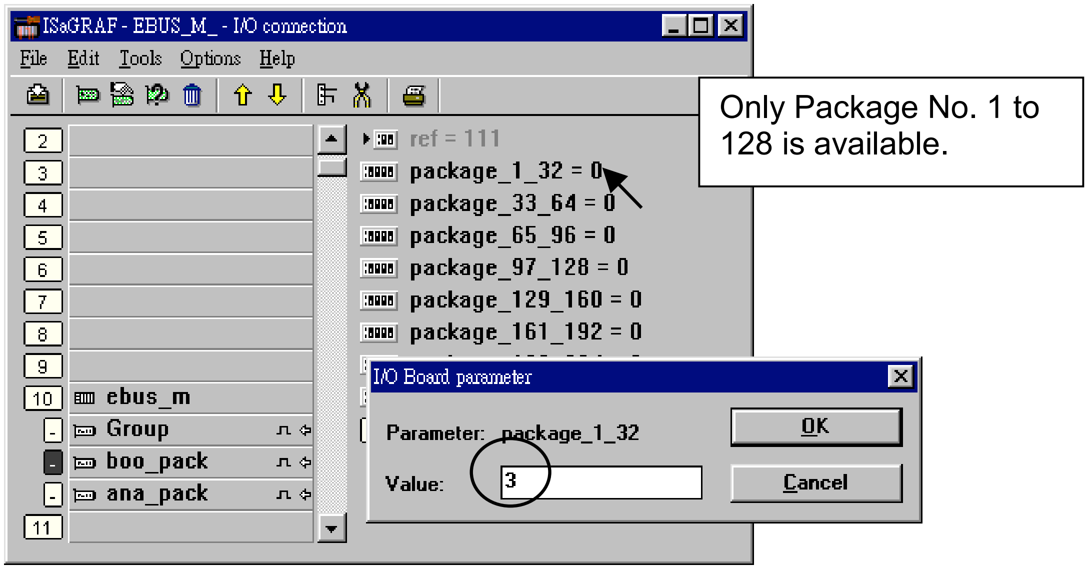
Configuring The Ebus Master Boolean Packages:
To begin configuring the Ebus Master Boolean Packages, click on the "boo_pack" selection from the "Ebus_m" I/O connection.

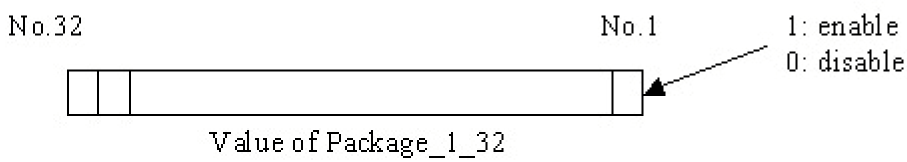


The parameter "package_xxx_xxx" at "Ebus_m: boo_pack" indicates the "Boolean package" number which is allowed to be written to or read from across the Ebus network. The parameter value is given as a 32-bit integer in **hexadecimal**.

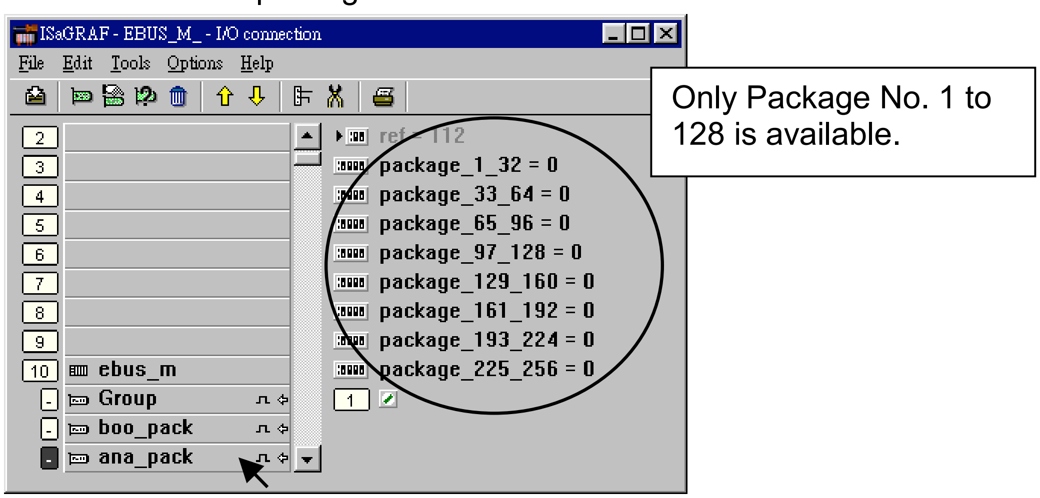
As an example, if the "package_1_32" is set to "FFFFFFFF" this will enable all the packages from number 1 to number 32 to be written to or read from across the Ebus network. If the "package _1_32" is set to a value of "A", this will only enable the number 2 and number 4 Boolean packages to be written to or read from across the Ebus network. The more packages that are enabled on a Ebus network the slower the communication efficiency will be. **With this in mind, always remember to enable only the required number of packages that you**

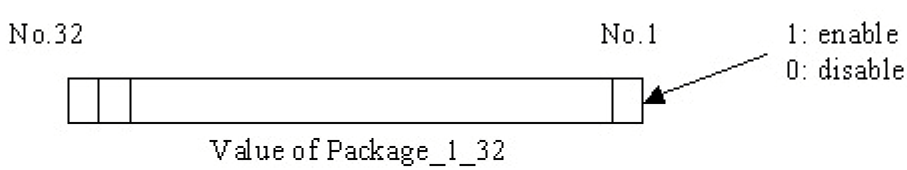
need for your application so you will have greater communication efficiency across the Ebus network.





The parameter "package_xxx_xxx" at "Ebus_m: ana_pack" indicates the "integer package" number which will be written to and read from on the Ebus network. The "Ebus_m: ana_pack" is used to read and write 32-bit integer values across the Ebus network. Each of the parameter values is expressed as 32-bit integer values in **hexadecimal**, and the same configuration rules apply as those for the "Boolean package".





7.5.3: Programming Ebus Packages

Before you can exchange any data across a Ebus network, you must make sure that each I-8437/ 8837, I-7188EG & W-8037/8337/8737 is configured as either a Ebus master "ebus_m" (and remember, only ONE controller can be the master in the same Ebus “Group_No”) or Ebus slave "ebus_s". Refer to Section 7.5.2 for details on how to implement these configurations.

The following Ebus function blocks can be used in a LD program to exchange data across an Ebus network.

Ebus_b_r	read one boolean package.
Ebus_b_w	write one boolean package.
Ebus_n_r	read one integer package.
Ebus_n_w	write one integer package.

The below two blocks can be used to exchange “real” value via Ebus.
Block “Real_Int” can be used to Map a “real” value to a 32-bit integer. So that you can deliver this integer to the Ebus, and then on the receiver controller, use “Int_Real” to map this integer back to the original “real” value.

Int_Real	Map a long integer to a Real value.
Real_Int	Map a Real value to a long integer.

The below block is to get the communication ststus of each Boolean & Integer Package.

Ebus_sts	Get ststus of each Package.

To program the Ebus_x_x blocks is similar to the Fbus, please refer to section 7.3 & 7.4 for detail.

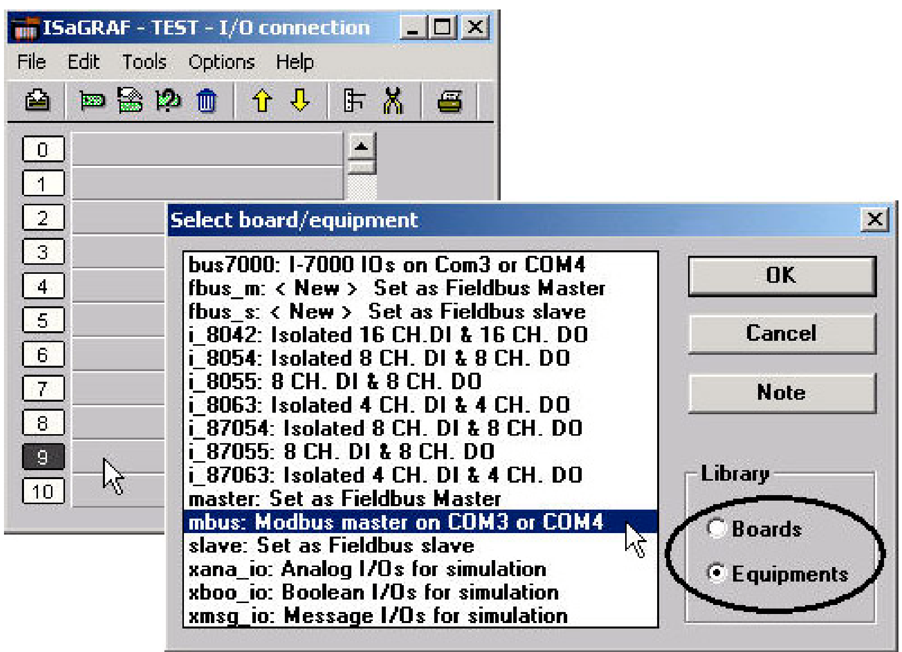
Chapter 8: Linking The Controller To Modbus RTU & Other Devices

The I-8xx7, I-7188EG/XG & W-8xx7 can interface with the Modbus RTU Serial or other Modbus devices. Please refer to Section 1.6 for the connection interface between the I-8xx7 controller system to Modbus RTU and other Modbus devices.

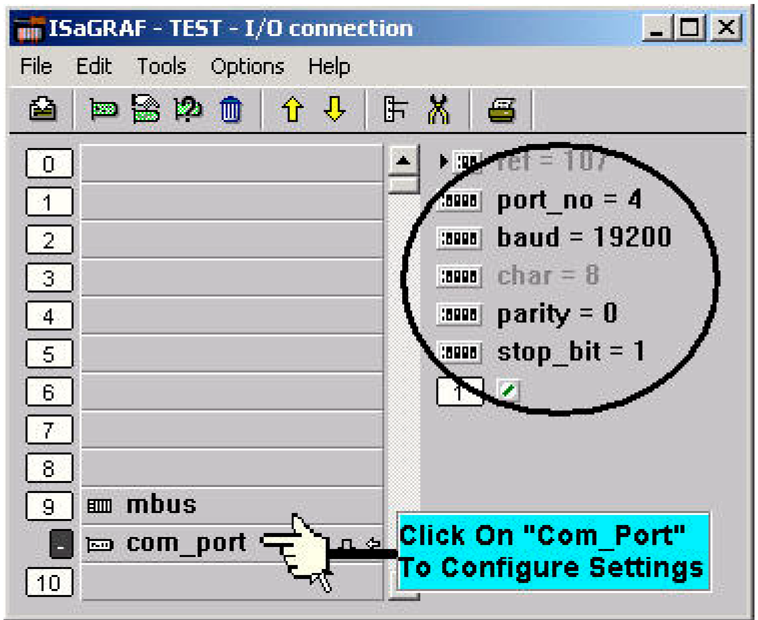
8.1: Configuring The Controller For A Modbus Device

To begin configuring an I-8xx7, I-7188EG/XG & W-8xx7 controller system to interface with a Modbus device, you must first configure the ISaGRAF program by linking the "Mbus" function to the ISaGRAF project. Open the "ISaGRAF I/O Connections" window and double click on a slot number higher than 7 and the "Select Board/Equipments" window will open. From the "Library", click on the "Equipments" choice, and then click on the "Mbus: Modbus Master On …" selection, and then click on the "OK" to complete the installation.

IMPORTANT NOTE:
Only **ONE** "Mbus" complex equipment function can be linked to **ONE** I-8xx7, I-7188EG/XG & W-8xx7 controller system.



"Mbus: com_port" Parameter
The "Mbus: com_port" parameter sets the same baud rate that the I-8xx7, I-7188EG/XG & W-8xx7 controller system and all Modbus devices will communicate at. ALL devices MUST be set to the same baud rate setting. The default baud rate setting for the "Mbus: com_port" parameter is 19200.



"Mbus: port_no" Parameter
The "Mbus: port_no" parameter defines which COM port the Modbus devices will communicate with the controller. The "Mbus: port_no" parameter can be set to either a value of "1" (COM1), "3" (COM3), "4" (COM4) or "5" (COM5 on the I-8112/8114/8142/8144 board) for the I-8417/8817/8437/8837, while "1" , "2" , "3" for the I-7188EG, and "2" , "3" for the I-7188XG & the W-8037/8337/8737. The default setting for the "Mbus: port_no" parameter is "4".

Note:
When setting COM1 of the I-8417/8817/8437/8837 & the I-7188EG to be a Modbus master port, please refer to Appendix C.1 – "Setting COM1 As None-Modbus Port" to disable COM1:Modbus RTU port.
W-8xx7's COM2 is Modbus RTU port by default, please disable it if using it as a Modbus master port. Please refer to W-8xx7's "Getting Started" Manual.

"Mbus: baud" Parameter
The "Mbus: baud" parameter defines what the communications baud rate setting will be. The "Mbus: baud" can be set to 2400, 4800, 9600, 19200, 38400, 57600 or 115200 baud rate. The default baud rate value is 19200 for the I-8xx7, I-7188EG/XG & W-8xx7 controller system. All controllers on the same Modbus MUST be set to the same baud rate.

"Mbus: parity" Parameter
The "Mbus: parity" parameter defines what the communications parity setting will be. Setting the "Mbus: parity" parameter to a value of "0" sets the parity to "none", a value of "1" sets the parity to even, and a value of "2" sets the parity to odd.

"Mbus: stop_bit" Parameter
The "Mbus: stop_bit" parameter defines the number of stop bits will be used in the Modbus communications. If the "Mbus: stop_bit" parameter is set to "1", this equals 1 stop bit, and a value of "2" equals 2 stop bits.

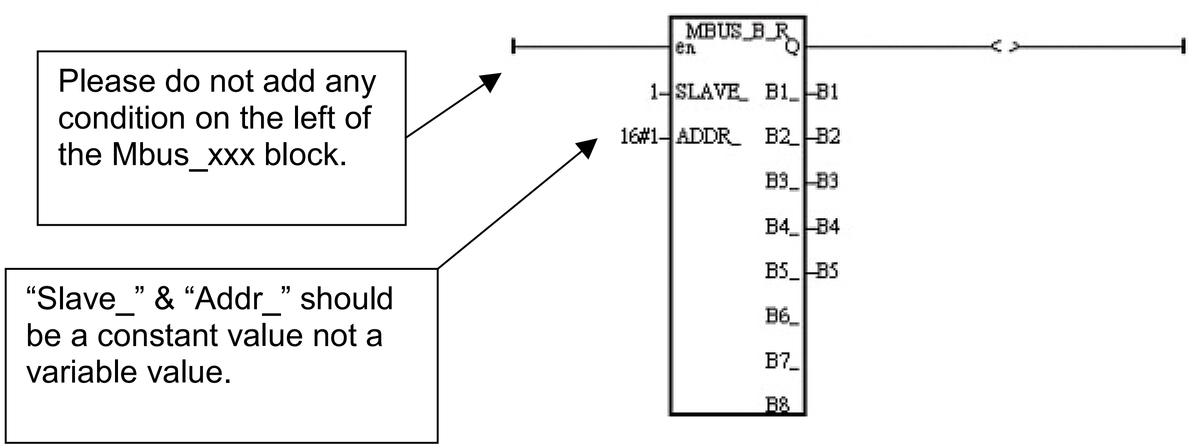
8.2: Programming A Modbus Device

The following function blocks can be used to pass data through the Modbus protocol in an LD program.

Mbus_b_r	Reads 8 bits (booleans) from modbus devices.
Mbus_br1	Reads 8 bits (booleans) with period time from modbus devices.
Mbus_b_w	Writes 1 to 4 bits to modbus devices.
Mbus_n_r	Reads 8 words (short integers) from modbus devices.
Mbus_nr1	Reads 8 words (short integers) with period time from modbus devices.
Mbus_n_w	Writes 1 to 4 words to modbus devices.
Mbus_r	Read Modbus code 1 to 4 from modbus devices
Mbus_r1	Read Modbus code 1 to 4 with period time from modbus devices
Mbus_wb	Using Modbus code 15 to write 1 to 16 bits.

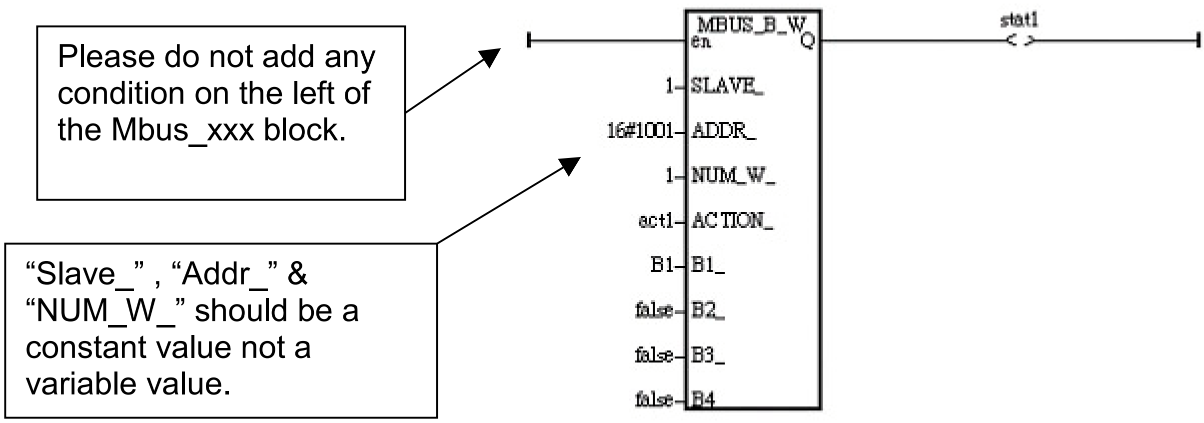
NOTE:
The maximum number of each "Mbus_x_x" function block that can be used with one I-8xx7 & I-7188EG/XG controller system is 64, while 256 for W-8037/8337/8737.

Modbus Example Function #1: "Mbus_b_r"
The following example the "Mbus_b_r" function block is reading five (5) bits from a slave Modbus device with a NET ID address of 1, with the Modbus address starting from 1. In this example the results of "B1" contains the value of the Modbus address 1, "B2" equals the value of Modbus address 2, etc. "B5" equals the value of the Modbus address 5.

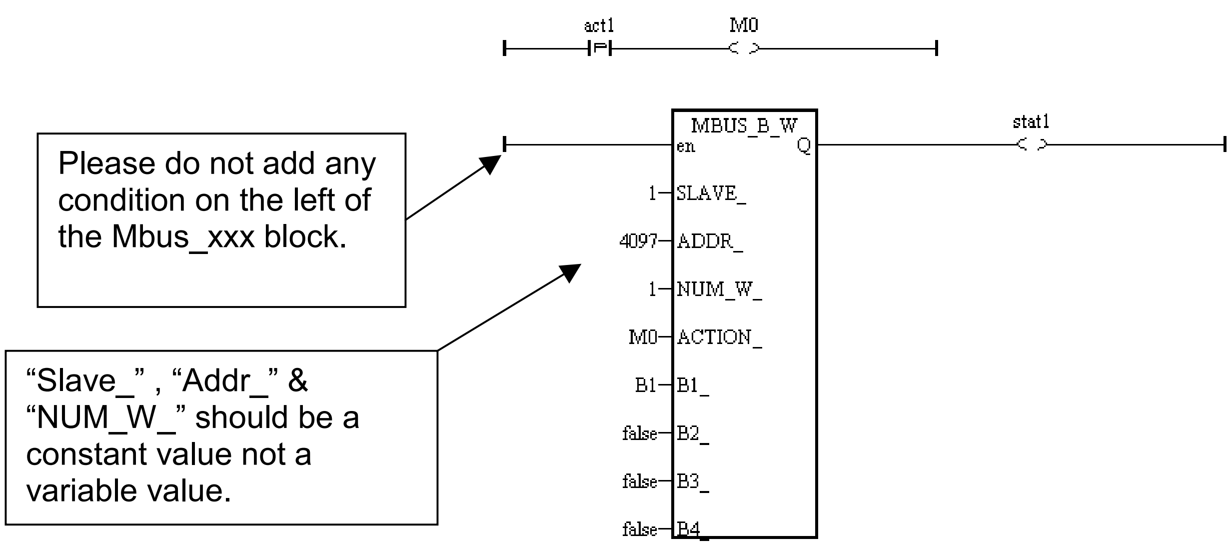


Modbus Example Function #2: "Mbus_b_w"
The following example of the "Mbus_b_w" function block is writing one (1) bit to a slave Modbus device with a NET ID address of 1. The "Mbus_b_w" function will only write this one bit when the "ACTION_" line is true. In the example below the resulting value of "B1" is written to the Modbus address 16#1001 (or 4097) of that Modbus device when the "ACTION_" line is true.

The value of "Stat1" is connected to the output coil and if the operation is successful "Stat1" will be true, otherwise the value of "Stat1" will be false.

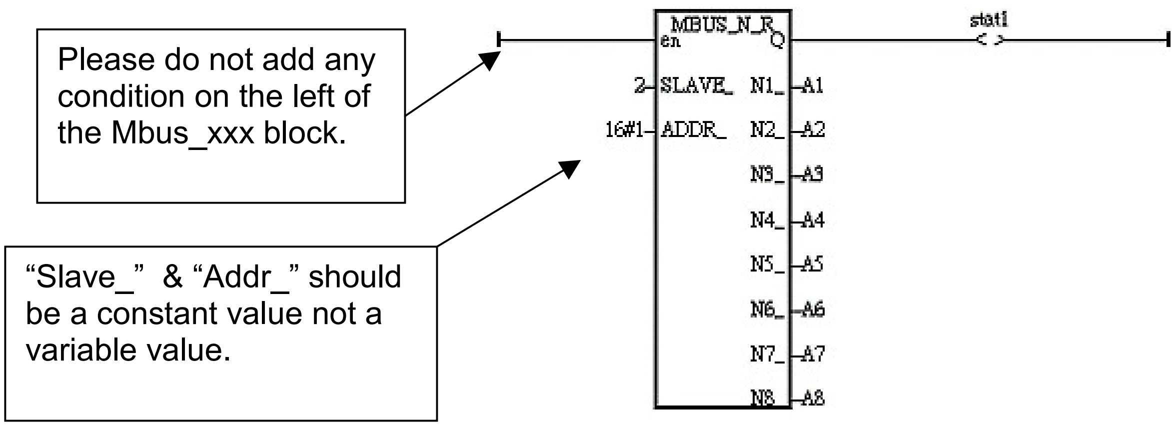


If the "ACTION_" input keeps at the status of TRUE, it will continue to write this "B1" many times to that Modbus device until it is reset to FALSE. If you just want to write one time, you can write a LD program similar as the following. The M0 is declared as an internal Boolean variable.



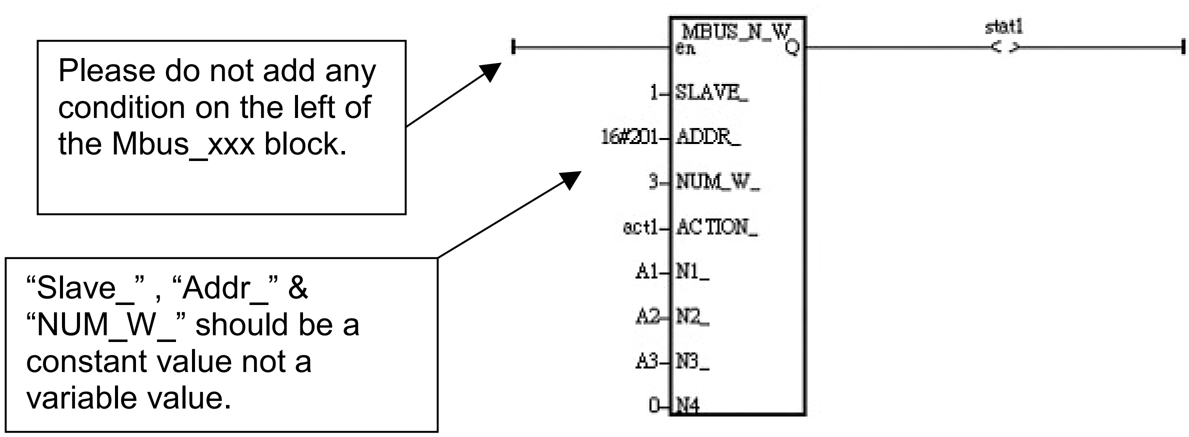
Modbus Example Function #3: "Mbus_n_r"
The following example the "Mbus_n_r" function block is reading eight (8) words from a slave Modbus device with a NET ID address of 2 (the Modbus address starts from 1). In this example the results of "A1" contains the value of the Modbus address 1, "A2" equals the value of Modbus address 2, etc., through "A8" which equals the value of the Modbus address 8.

The value of "Stat1" is connected to the output coil and if the operation is successful "Stat1" will be true, otherwise the value of "Stat1" will be false.

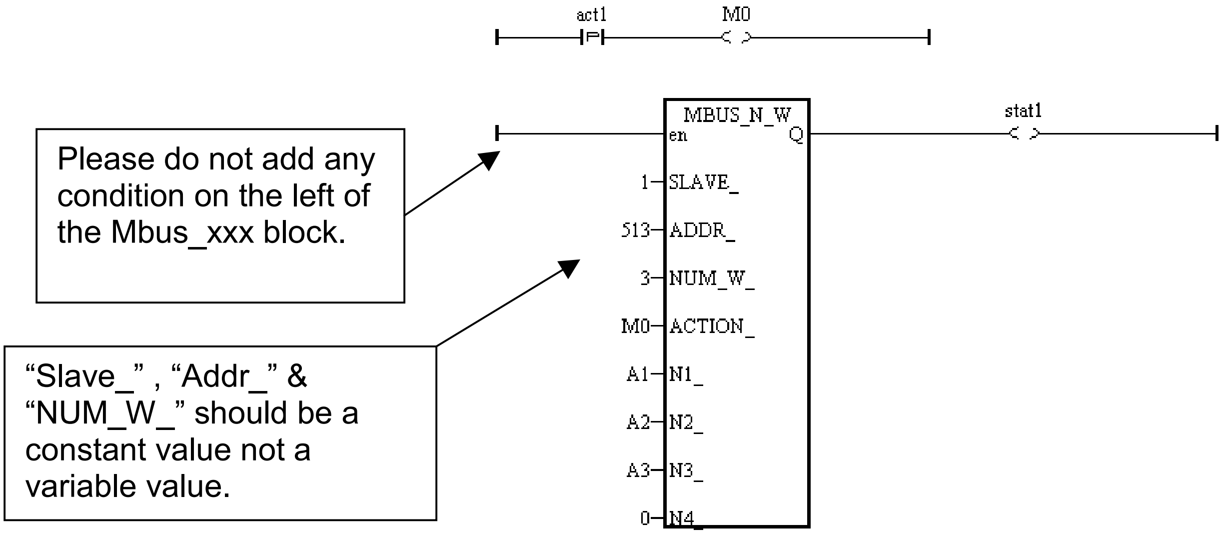


Modbus Example Function #4: "Mbus_n_w"
The following example of the "Mbus_n_w" function block is writing three (3) words to a slave Modbus device with a NET ID address of 1, and the Modbus address is starting from 16#201. The "Mbus_n_w" function will only write when the "ACTION_" line is true. In this example when the "ACT1" line is True, the value of A1 will be written to the value of Modbus address 16#201 of that Modbus device, the value of A2 will be written to the value of Modbus address 16#202, and A3 will be written to the value of Modbus address 16#203.

The value of "Stat1" is connected to the output coil and if the operation is successful "Stat1" will be true, otherwise the value of "Stat1" will be false.



If the “ACTION_” input keeps at the status of TRUE, it will continue to write these "A1" through "A3" many times to that Modbus device until it is reset to FALSE. If you just want to write one time, you can write a LD program similar as the following. The M0 is declared as an internal Boolean variable.



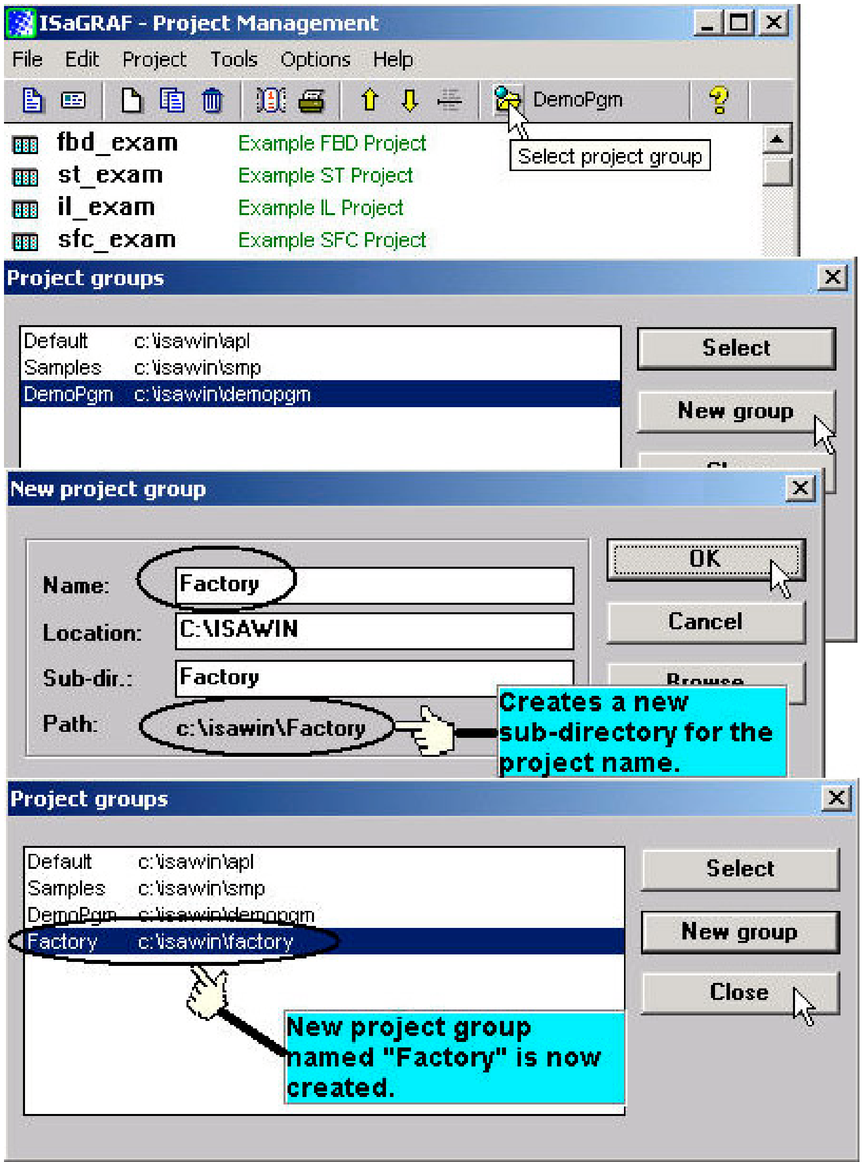
Chapter 9: Commonly Used ISaGRAF Utilities

The following chapter describes many useful features and utilities of the ISaGRAF Workbench programming environment. These features and utilities make programming an ISaGRAF project quick and easy.

This chapter in no way contains all of the features and utilities available with the ISaGRAF Workbench program. For more details and information about all the features the ISaGRAF Workbench program has to offer consult the "ISaGRAF USER’s GUIDE" manual which can be found from the CD ROM of the ISaGRAF workbench. Its file name is either “ISaGRAF.pdf” or “ISaGRAF.doc”.

9.1: Creating An ISaGRAF Project Groups

A very useful feature of the ISaGRAF program is the ability to organize numerous programs into "projects". The "Creating Projects" feature assists an ISaGRAF programmer who must create and maintain many different ISaGRAF programs for different application projects.

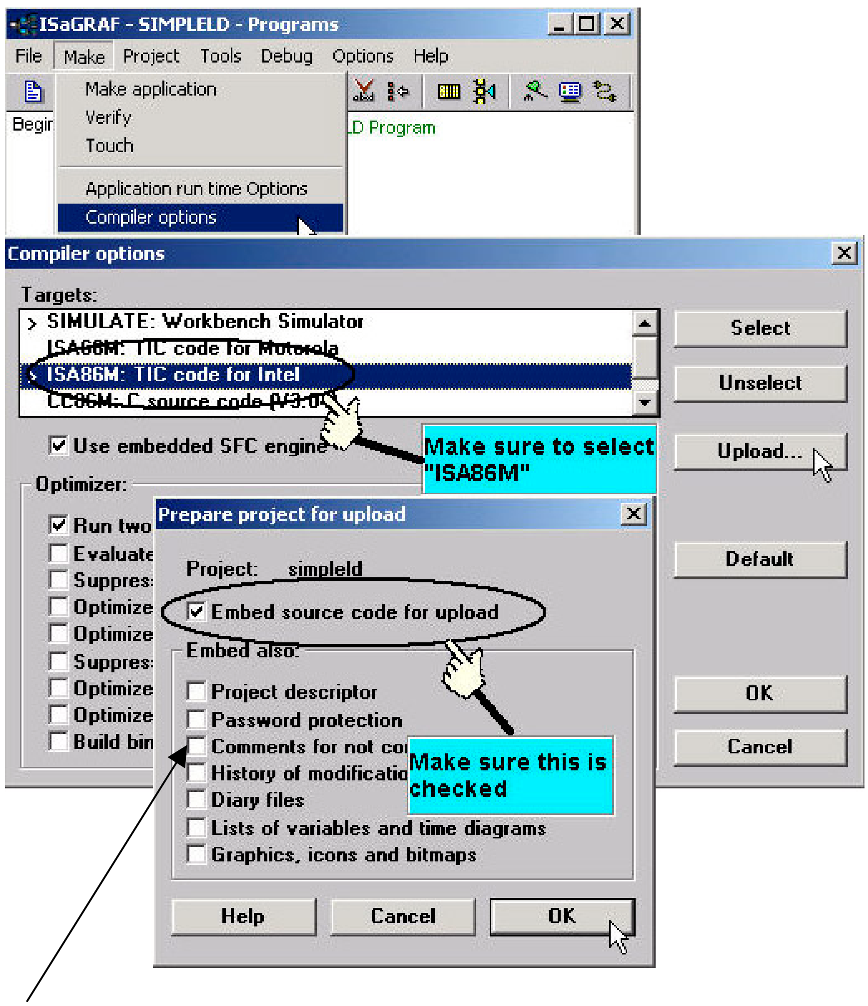


If you want to delete an existing project group, simply use the Windows Explorer to locate the ISaGRAF sub-directory you want to delete. An example of this is that if you wanted to delete the project just created, use the Windows Explorer and go to the C:\isawin\factory directory, and then just delete the "factory" sub-directory.

9.2: Uploading An ISaGRAF Project

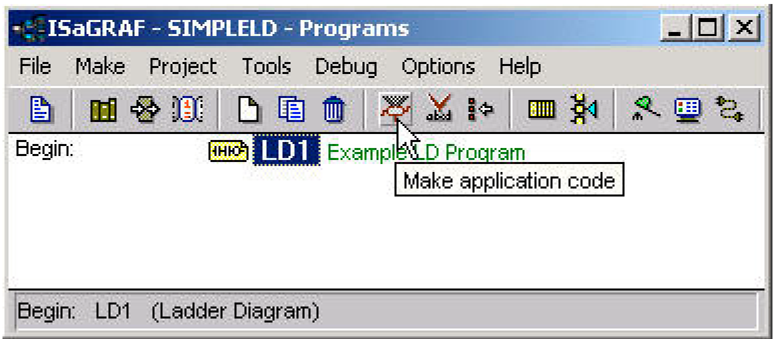
There may be occasions when you will want to "Upload" an ISaGRAF project from an I-8xx7, I-7188EG/XG & W-8xx7 controller system to your development PC. This is easily accomplished **IF** the "Upload" function from the "Compiler Option" is turned on.

To turn the upload function on from the "Compiler Option", open the "ISaGRAF Programs" window, select "Make" from the menu bar, and then click on "Compiler Options". The "Compiler Options" window will open, make sure the "ISA86M: TIC Code For Intel" is selected, and then click on the "Upload" button. The "Prepare Project For Upload" window will open, click on the "Embed Source Code For Upload" checkbox and then click on the "OK" button.



VERY IMPORTANT NOTE:
Option "**Comments for not connected I/O channels**" must be choosed if "Directly represented variables" is used in this project (refer to section 3.4).

After you have checked the "Embed Source Code For Upload" checkbox and clicked on the "OK" button, you will need to recompile the project and download the project to the I-8xx7, I-7188EG/XG & W-8xx7 controller system.

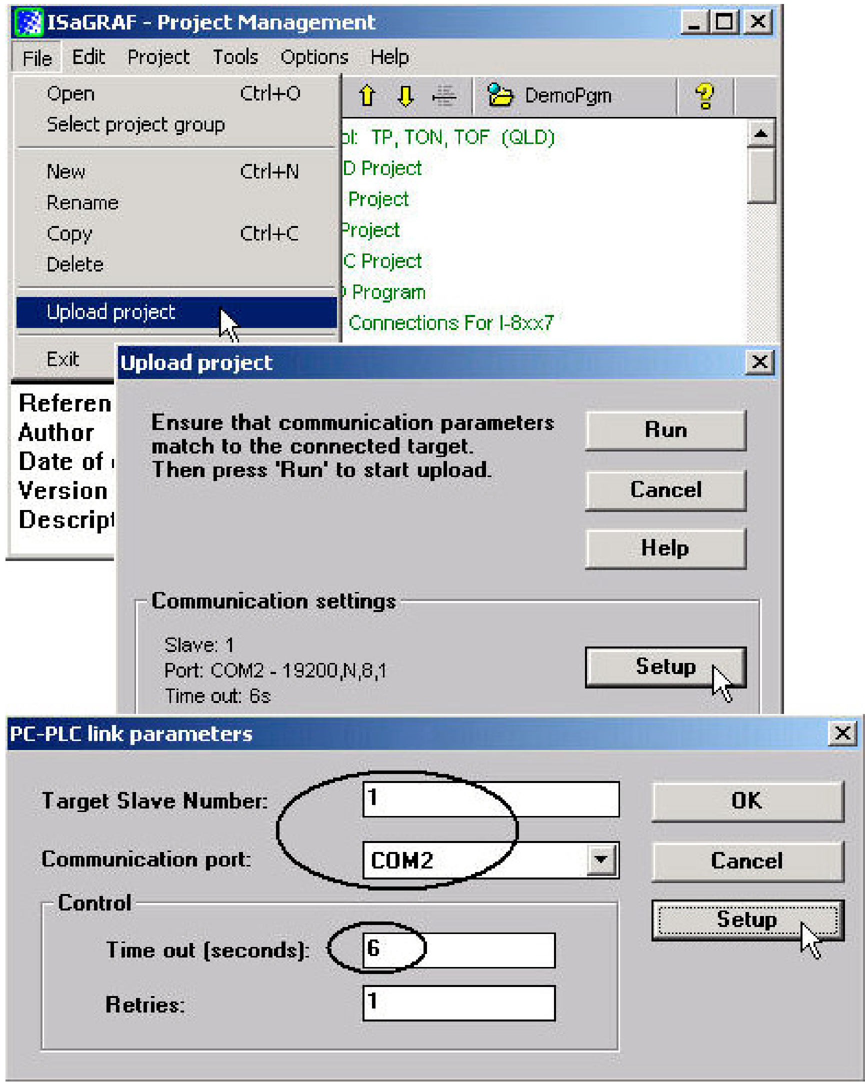


IMPORTANT NOTE:
Once you have enabled the "Upload" option, the code generated by the compiler will increase the size of the original program from **ONE & A HALF TO THREE TIMES** the original program size. If the uploaded code size is larger than 64K bytes, you will not be able to download the program to the I-8xx7 & I-7188EG/XG controller system. The code size limitation is 512K bytes for W-8xx7 controller system.

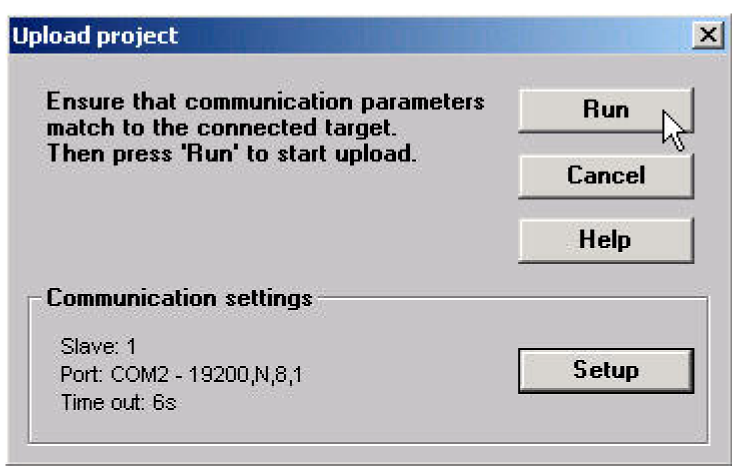
Before trying to download the program it is advisable that you check the size of the uploaded program. To check the uploaded program size, use the Windows Explorer program and go to the appropriate sub-directory that the application program resides in. As an example, the "SIMPLELD" program that was create resides in the C:\ISAWIN\DEMOPGM\SIMPLELD program sub-directory.

Remember, the "DEMOPGM" sub-directory is the **Project** group that the SIMPLELD program resides in, and the "SIMPLELD" sub-directory is where the actual application code files reside in. Look for the file named "**APPLI.X8M**" and check the size of this file. The "APPLIC.X8M" file is the file that contains the actual code that will be uploaded or downloaded to the I-8xx7 controller system. Make sure the sizes of this file DOES NOT exceed 64K for I-8xx7 & I-7188EG/XG. And Do not exceed 512K for W-8xx7.

UPLOADING AN ISaGRAF PROJECT
To upload an ISaGRAF project from an I-8xx7, I-7188EG/XG & W-8xx7 controller system open the "ISaGRAF Project Management" window, select "File", and then click on "Upload Project". The "Upload Project" window will now open, and check that the communication settings between your development PC and the I-8xx7, I-7188EG/XG & W-8xx7 controller system match each other. If the communication settings DO NOT match between the development PC and the controller, click on the "Setup" button to configure the proper communication settings.



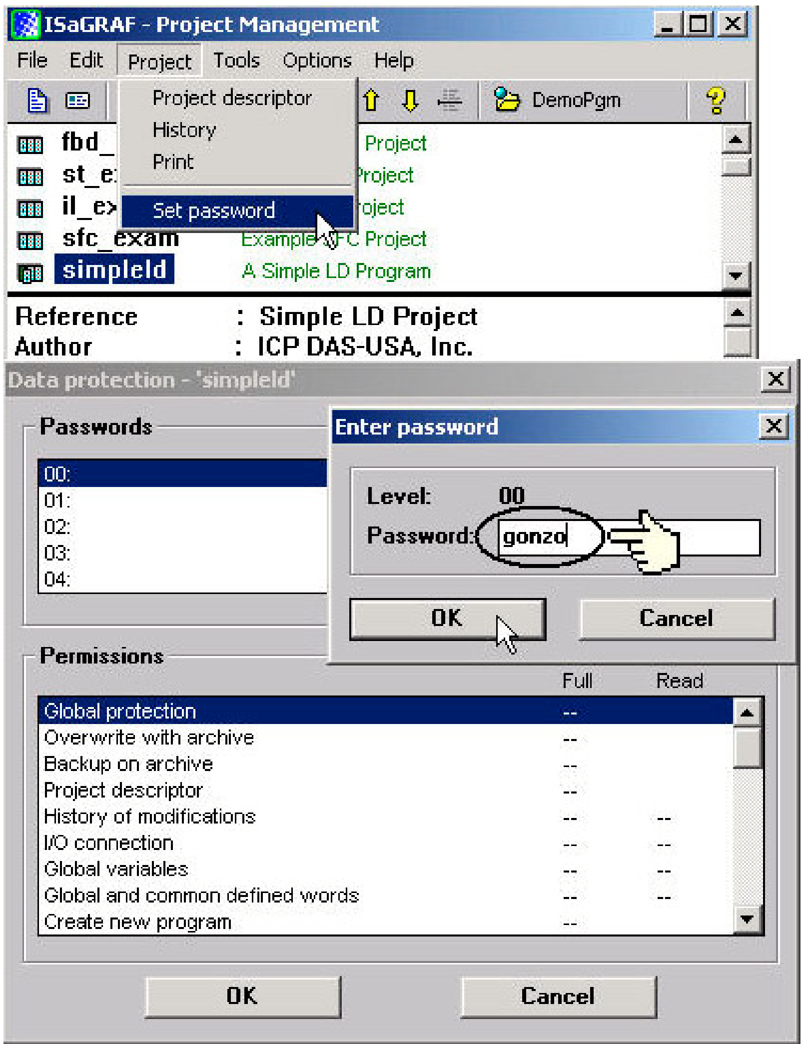
Once you have made sure that the communication settings are properly configured, click on the "RUN" button in the "Upload Project" windows.



9.3: Setting An ISaGRAF Password

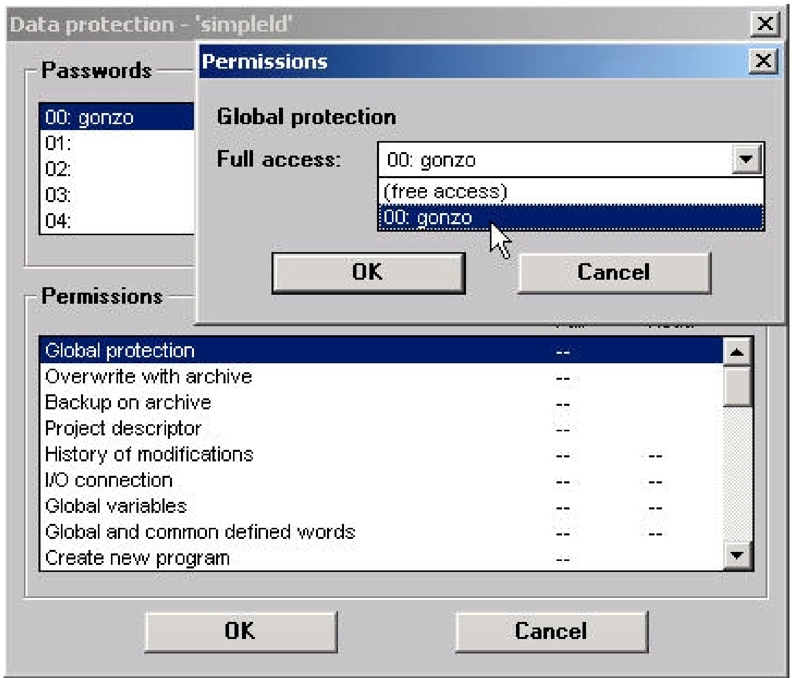
An ISaGRAF Workbench project can be password protected by configuring a user-defined password. To configure an ISaGRAF password, open the "ISaGRAF Project Window", select "Project" from the menu bar, and then click on "Set Password". The "Data Protection" window will open and then select on of the passwords from "00 to 15" to configure a password (this means that up to 16 passwords can be assigned with the ISaGRAF Workbench program).

You will also need to select the type of data protection you are creating for your ISaGRAF project. In the example below we are defining the "Global Protection" for this ISaGRAF project.

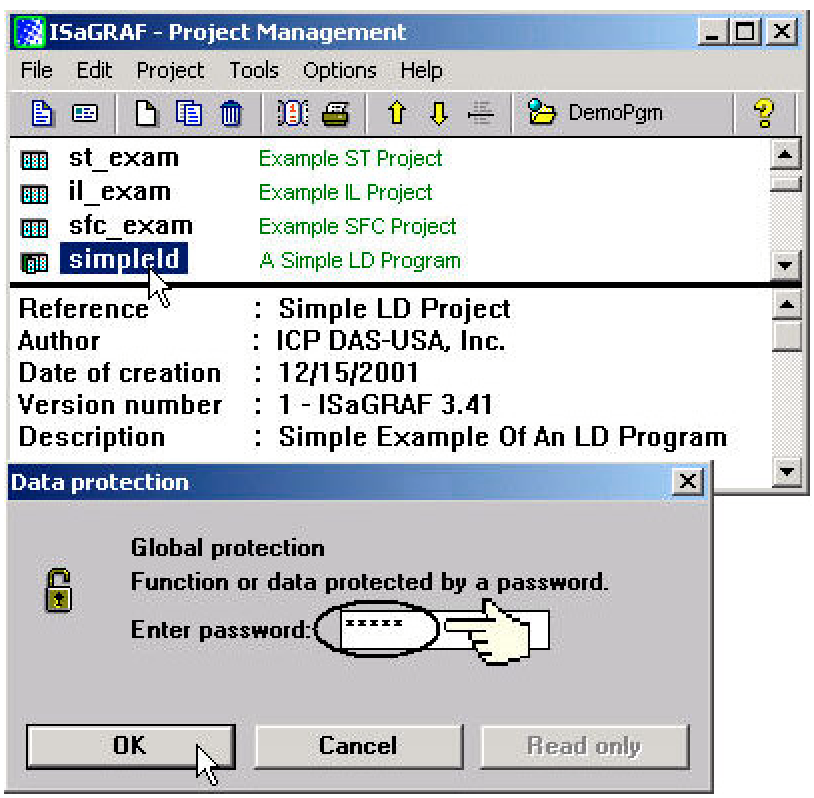


When you click on the "OK" button from the "Enter Password" window your new password will now be associated with the ISaGRAF project.

The next item you need to define is the type of data protection "Permissions" that will define for your ISaGRAF project. Double click on new password you have created and the "Data Protection Permissions" window will open. To allow full access WITH password protection, click on the "Full Access" scroll bar and click on the new password name you have created.

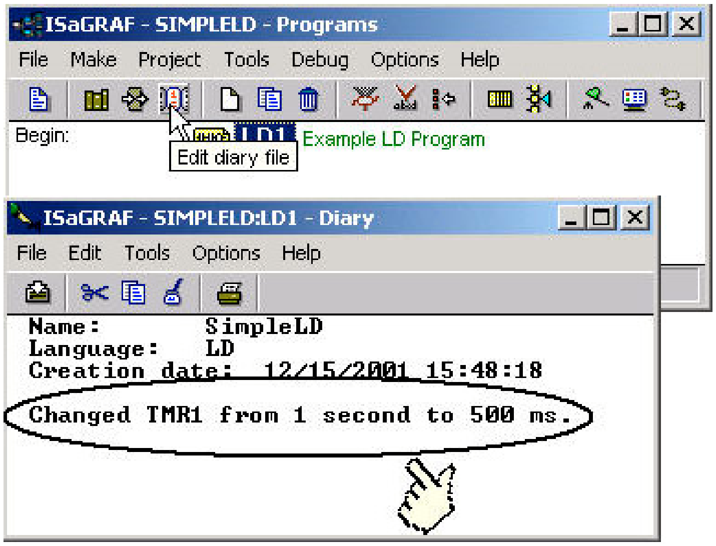


To verify that your password protection is now set for your ISaGRAF program, close all of ISaGRAF windows and then open the "ISaGRAF Project Management" window. Double click on the ISaGRAF program that you have created the password protection for. A "Data Protection" window will now open requiring you to enter the password for the ISaGRAF program you are attempting to open.



9.4: Creating An ISaGRAF Program Diary

When you modify an ISaGRAF program you can keep track of these revisions by entering a comment into the "Edit Diary" window. This affords the programmer the opportunity to add comments about program modifications and then save a record of these changes using the "Edit Diary" facility for enhanced program management capability.



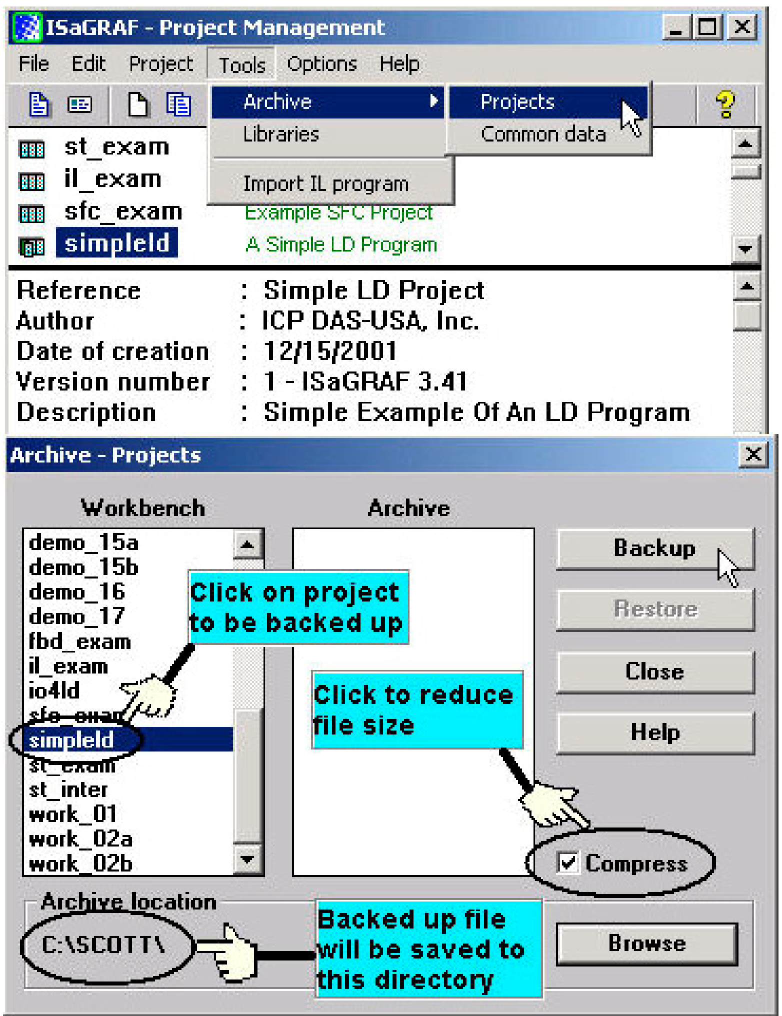
When you have completed entering information in the "ISaGRAF Diary" file, just click on the "Save" icon for your revision notes to be saved.

9.5: Backing Up & Restoring An ISaGRAF Project

For archiving purposes you can "Back Up" and "Restore" an ISaGRAF project. For example, you may want someone to test your program or email to service@icpdas.com for ICP DAS's ISaGRAF technical service.

Backing Up An ISaGRAF Project

Open the "ISaGRAF Project Management" window, select "Tools" from the menu bar, click on "Archive", and then click on "Projects". An "Archive Projects" window will open which allows you to designate where you want to save the ISaGRAF project to. Click on the name of the ISaGRAF project you want to backup, and then click on the "Backup" button. You can compress the size of the file you have backed up by clicking on the "Compress" checkbox BEFORE you click on the "Backup" button.

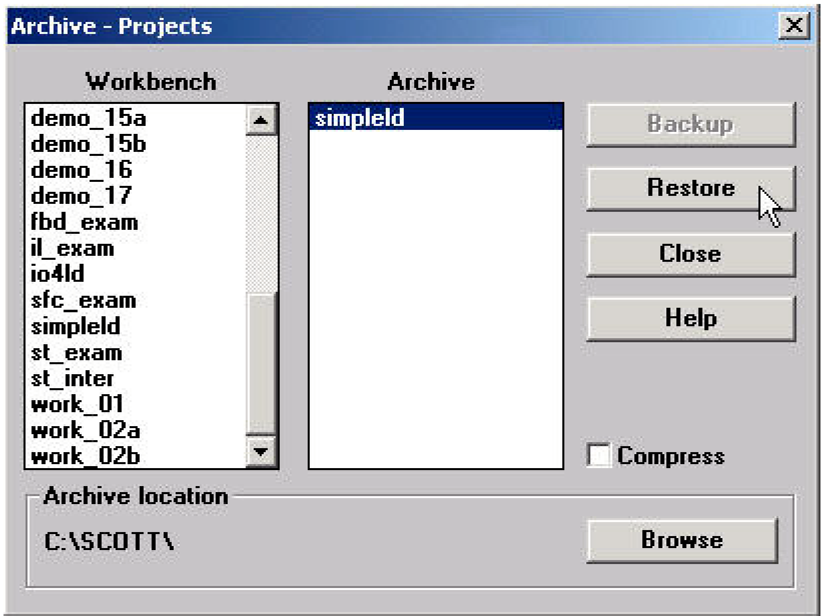


You will now find the backed up ISaGRAF project file in the "Archive" location you have designated. In the example above, the name of the backed up file is "simpleld.pia".

Restoring An ISaGRAF Project

To restore an ISaGRAF project from a backed up file, use the same method as above to access the "Archive Projects" window, click on the name of the project you want to restore from the

"Workbench" window, then click on the name of the backed up file from the "Archive" window, then click on the "Restore" button. The ISaGRAF project will now be restored to the sub-directory you designated.



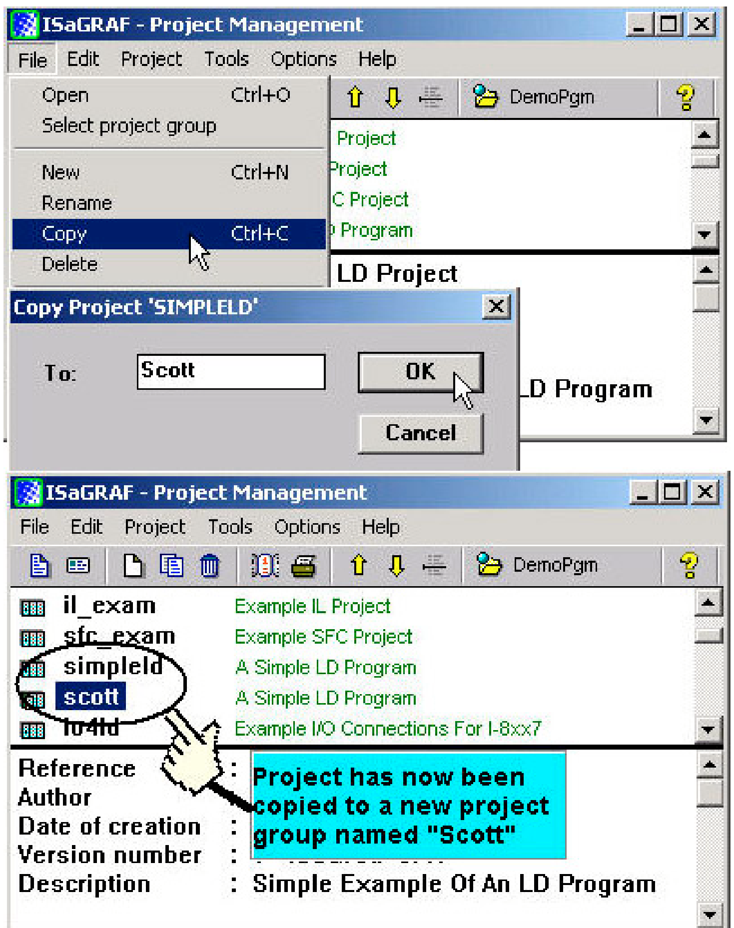
You can now open, edit and download the restored ISaGRAF project file.

9.6: Copying & Renaming An ISaGRAF Project

The ISaGRAF Workbench program has the capability of copying and renaming an ISaGRAF project or program. This is useful if you want to maintain a copy of an ISaGRAF project or program in a secondary directory.

Copying An ISaGRAF Program

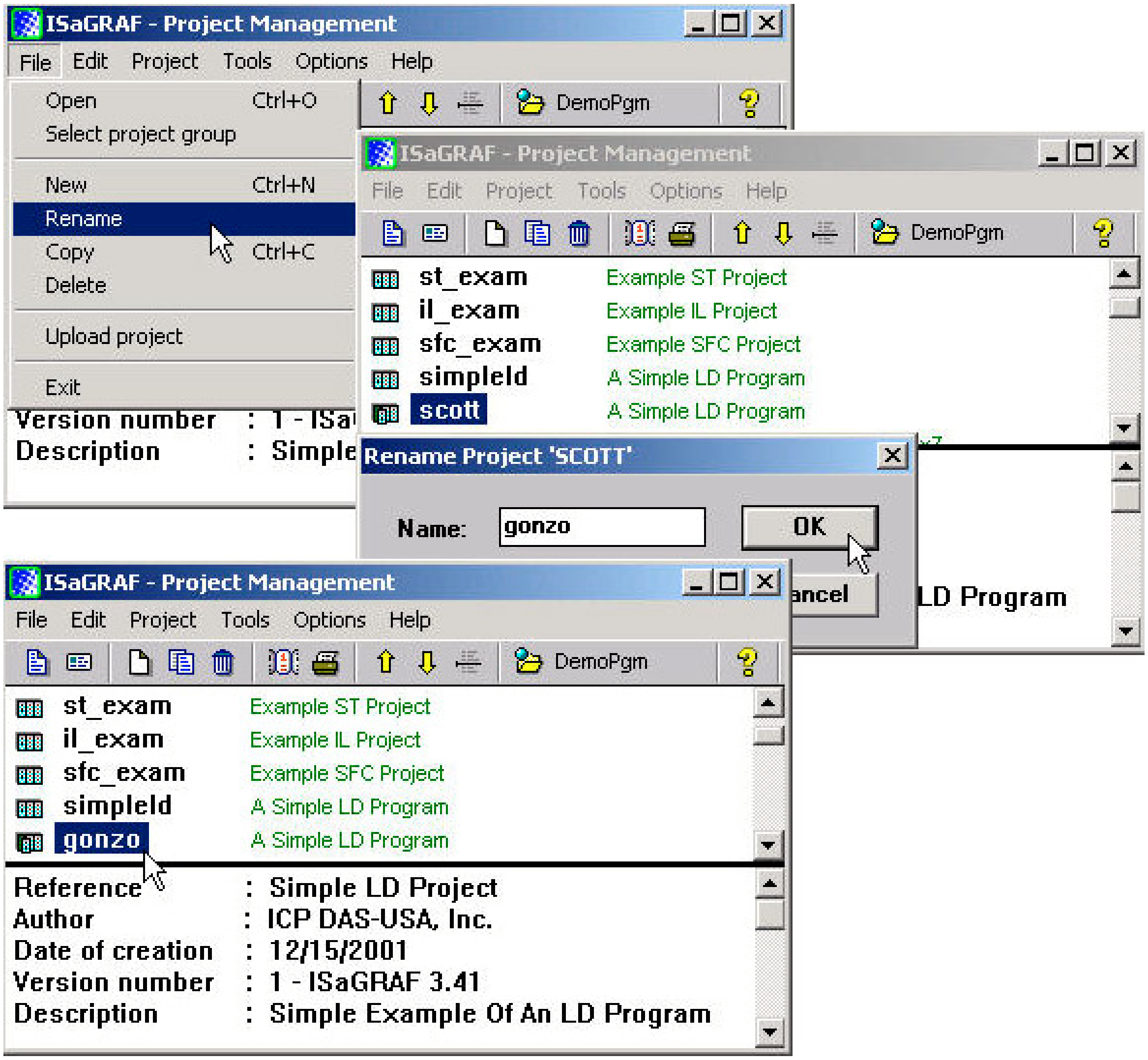
To copy an ISaGRAF program open the "ISaGRAF Project Management" window, first click on the name of the ISaGRAF program you want to copy, then select "File" from the menu bar, and then click on "Copy". When you click on "Copy" the "Copy Project" window will open, and now you can enter the name of the program you have selected to where you want to copy the program. If the new program name does not already exist, ISaGRAF will create the project name for you.



Note in the bottom screen that ISaGRAF has created a new program named "Scott" and placed a copy of all the files from "simpleld" into the "Scott" program group.

Renaming An ISaGRAF Program

To rename an ISaGRAF program open the "ISaGRAF Project Management" window, click on the name of the ISaGRAF program you want to rename, then select "File" from the menu bar, and then click on "Rename". When you click on "Rename" the "Rename Project" window will open, and now you can enter the new name for the ISaGRAF program.

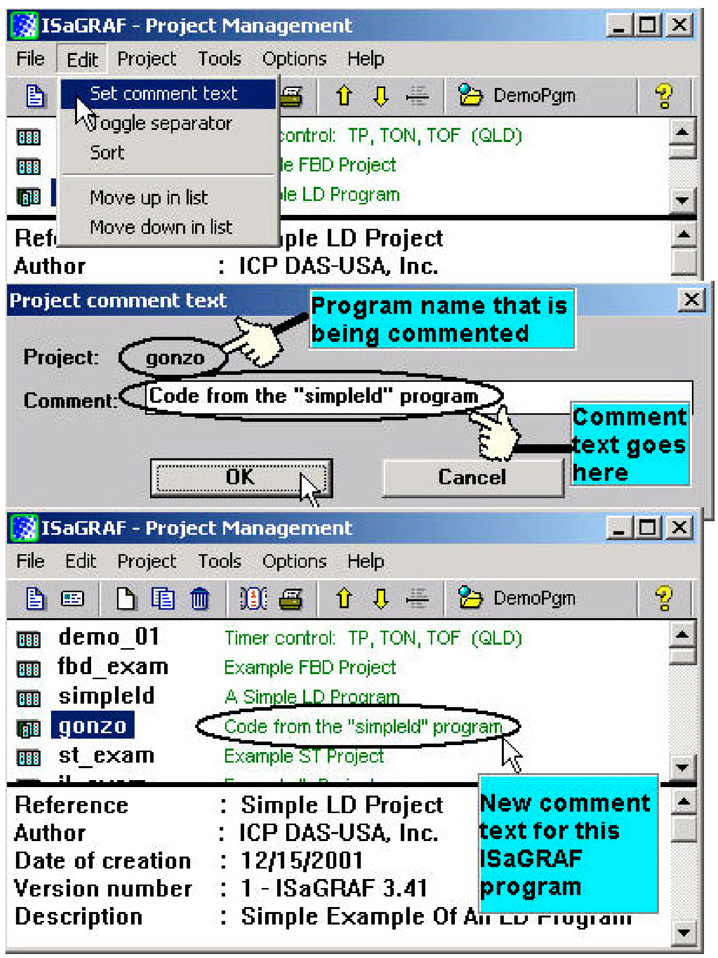


The former program named "scott" has now been changed to "gonzo", but it still has all the files from the "simpleld" program.

9.7: Setting Comment Text For An ISaGRAF Project

A useful feature of the ISaGRAF Workbench program is the ability to create "Comment Text" that will be placed next to an ISaGRAF program name in the "ISaGRAF Project Management" window. This way you can provide additional information about the purpose and any other additional comments regarding a particular ISaGRAF program.

To create "Comment Text" for an ISaGRAF program first open the "ISaGRAF Project Management" window, click on the name of the ISaGRAF program you want to create the comment text for, then select "Edit" from the menu bar, and then click on "Set Comment Text". When you click on "Set Comment Text" the "Project Comment Text" window will open, and now you can enter any comments and information you desire for the ISaGRAF program you have selected.

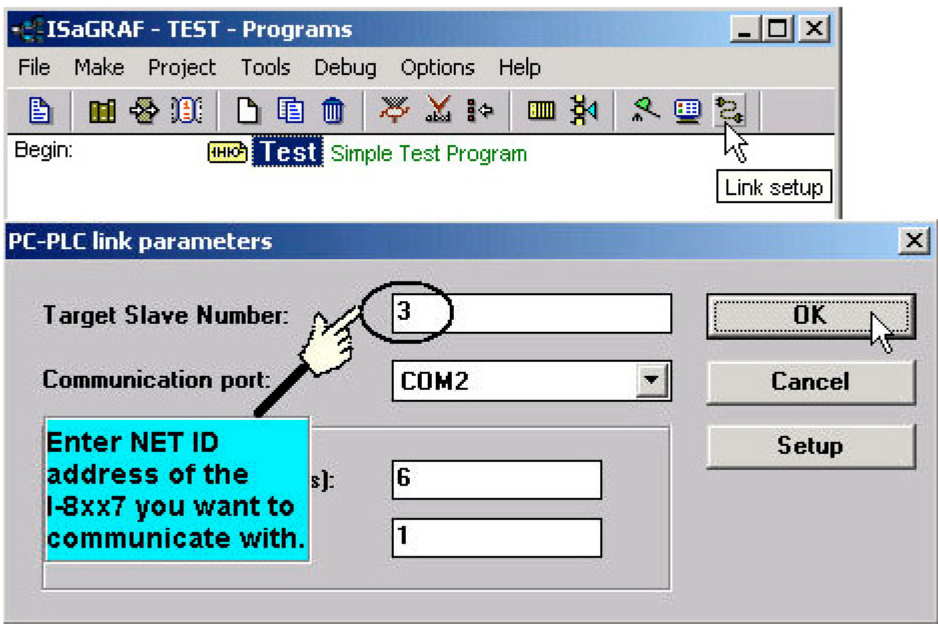


9.8: Setting The Slave ID For An ISaGRAF Controller

Each I-8xx7, I-7188EG/XG & W-8xx7 controller system has a "NET ID" address that must be set to identify the controller to the ISaGRAF Workbench program. By default the NET ID address is "1" when it is shipped out.

If you need to communicate with multiple I-8xx7, I-7188EG/XG & W-8xx7 controller systems via RS485 network, you must set the NET ID address in the ISaGRAF program for the specific I-8xx7, I-7188EG/XG & W-8xx7 controller system you want to communicate with. To communicate with different controller systems from one development PC open the "ISaGRAF Programs" window and click on the "Link Setup" icon.

When you click on the "Link Setup" icon, the "PC-PLC Link Parameters" window will open. Enter the "Target Slave Number" of the I-8xx7, I-7188EG/XG & W-8xx7 controller system you want to communicate with.

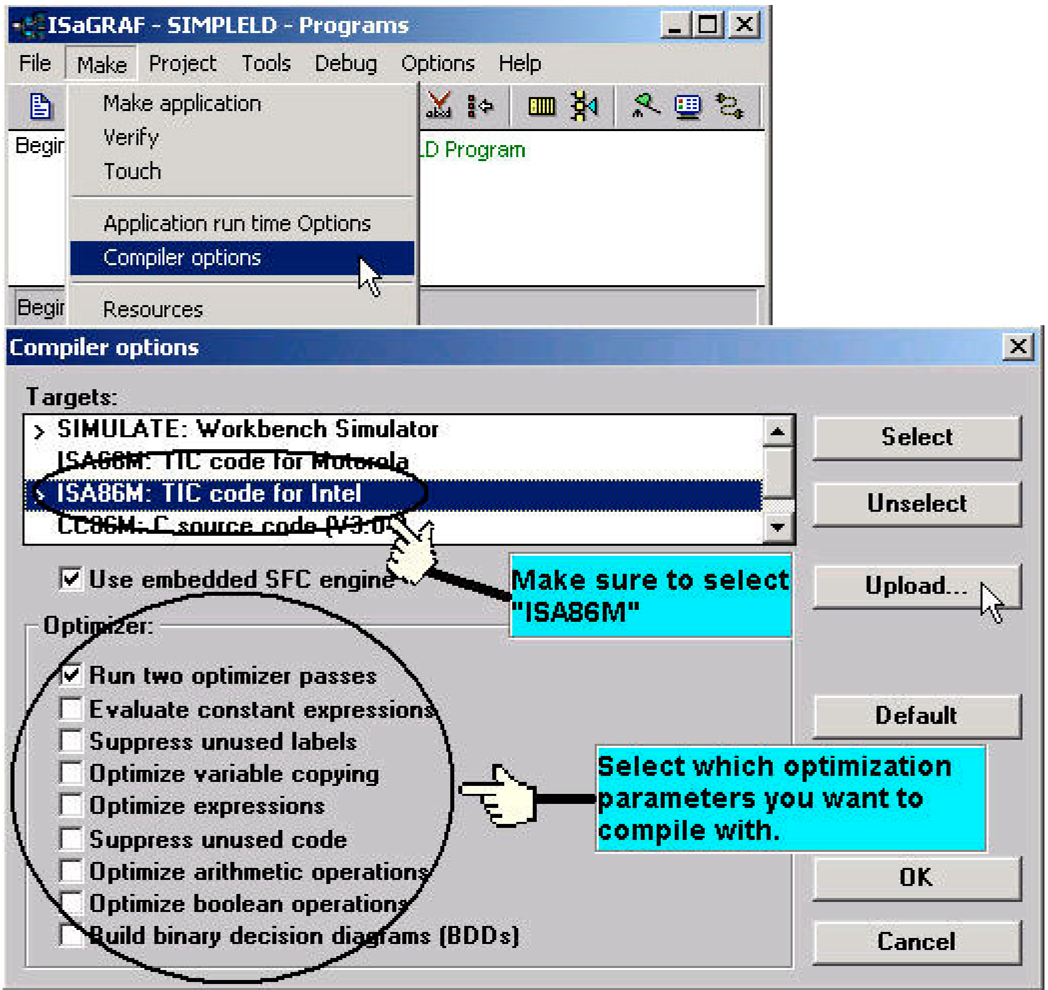


IMPORTANT NOTE

Remember that the NET ID address of the I-8xx7 controller system is determined by the DIP switch settings on the bottom right hand side of the controller. Refer to Section 1.3.1 for the DIP switch settings to determine the NET ID address for the I-8xx7 controller system you want to communicate with. To set Net-ID for the I-7188EG/XG & Wincon-8xx7, please refer to their own “Getting Started Manual” delivered with the product.

9.9: Optimizing The ISaGRAF Code Compiler

The ISaGRAF Workbench program allows you to modify the settings for the "Compiler Options" to optimize the ISaGRAF program when you compile your project. To access the "Compiler Options" open the "ISaGRAF Programs" window and select "Make" on the menu bar, and then click on "Compiler Options". The "Compiler Options" window will open, and now you can select which optimization parameters you want for when you compile your ISaGRAF program.



Selecting the "Run Two Optimizer Passes" will insure that the code is compiled into the smallest possible program code.

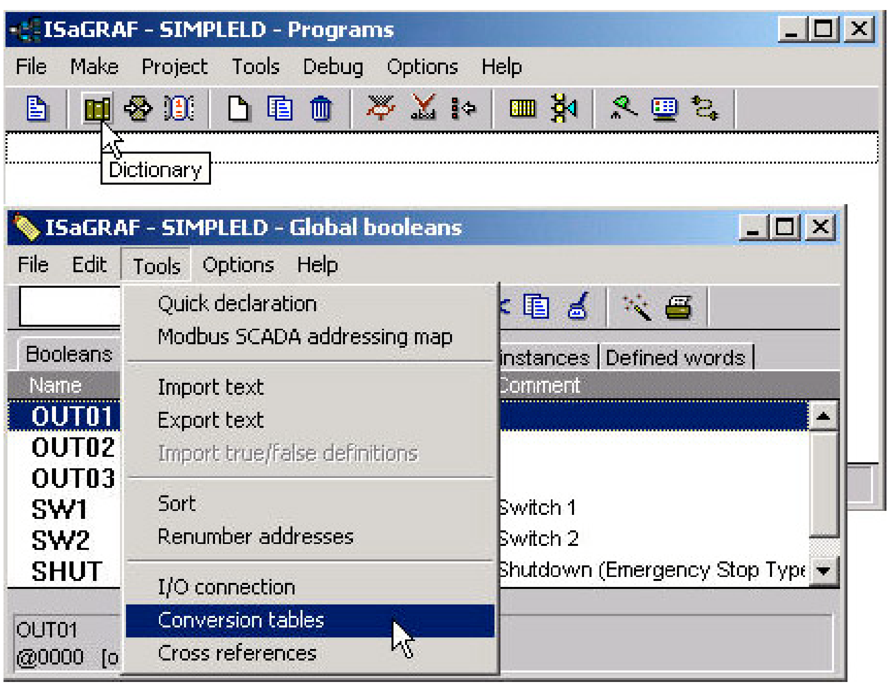
9.10: Using The ISaGRAF Conversion Table

Conversion Table Example
In this "Conversion Table" example the value from an I-87017 (an eight channel analog input module) board needs to be converted. The I-87017 is configured to receive a –10v to +10v signal, where –10v equals a value of "-32768", and a +10v signal equals a value of "+32767". You may refer to Appendix D to see the translation table of each analog board.

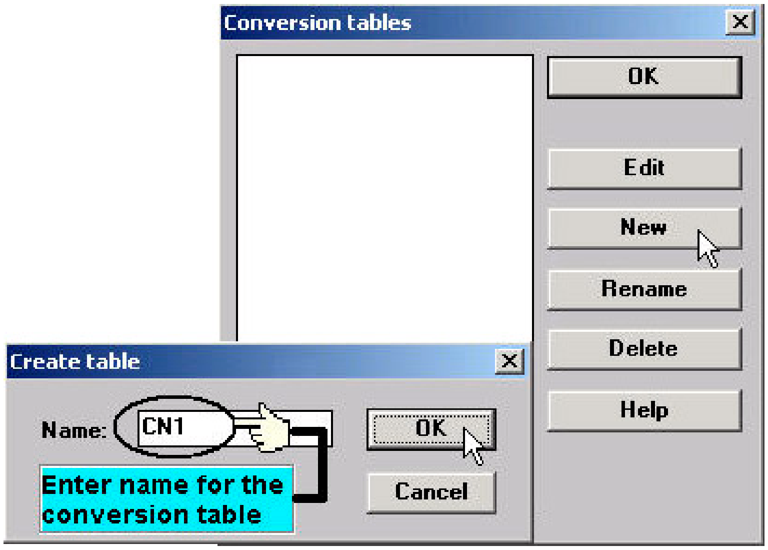
In this example we will use the "Conversion Table" to reconfigure the I-87017 so that a –10v signal will equal a value of "-10000" and a +10v signal will equal a value of "10000". In this example a value of +2.573v signal will equal a value of "2573".

Note:
The I-8xx7, I-7188EG/XG & W-8xx7 controller only supports the value before conversion within –32768 to +32767, and the value after conversion within **–10000 to +10000**. Setting conversion table out of these range may cause errors.

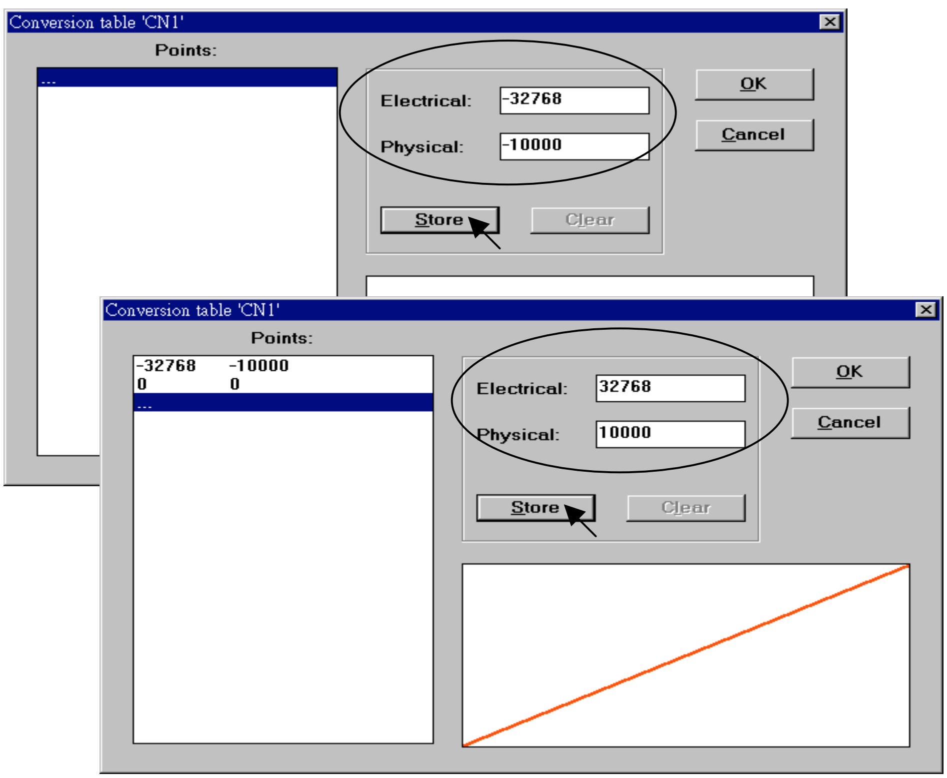
To configure a "Conversion Table" open the "ISaGRAF Programs" window and click on the "Dictionary" icon. This will open the "ISaGRAF Global Variables" window, select "Tools" from the menu bar, and then click on "Conversion Tables".



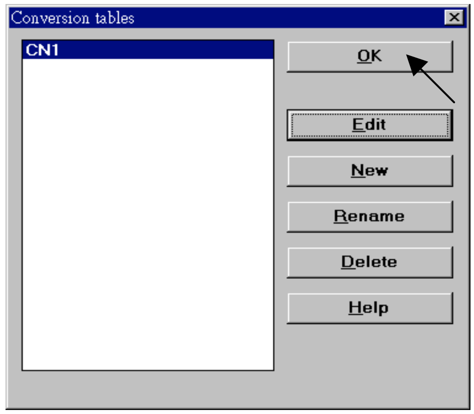
When you click on the "Conversion Tables" selection the "Conversion Tables" window will open. Next, click on the "New" button and then the "Create Table" window will now open. In the "Create Table" window enter the name for the conversion table you are creating.



To properly create our example "Conversion Table" at least two values must be defined. The "Electrical" field means the original value BEFORE conversion and the "Physical" field is for the value AFTER conversion. The two points defined in this example are (-32768, -10000 "lower limit") and (+32767, 10000 "upper limit"). Click on the "STORE" button to save each entry.

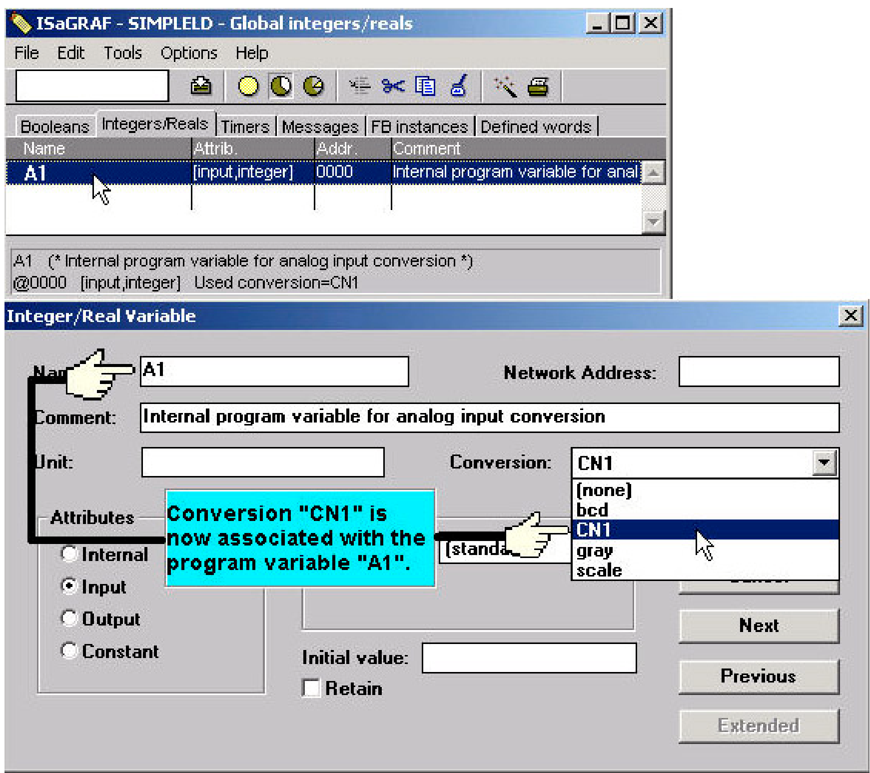


When you have completed entering in the two value points, click on the "OK" button to save the entered values.



The last step is to assign the conversion table "CN1" to a program variable that will be used in an ISaGRAF program.

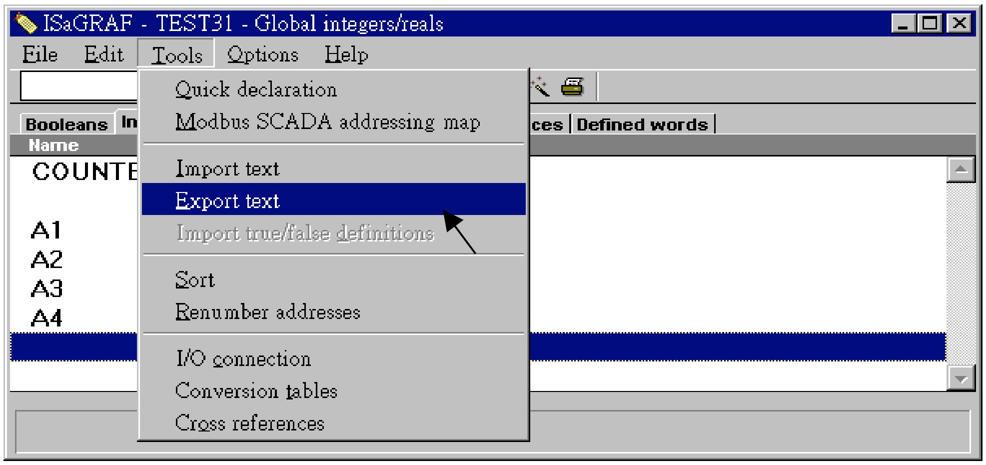
Note:
Only integer variable declared as input or output attribution can be assigned a conversion table.



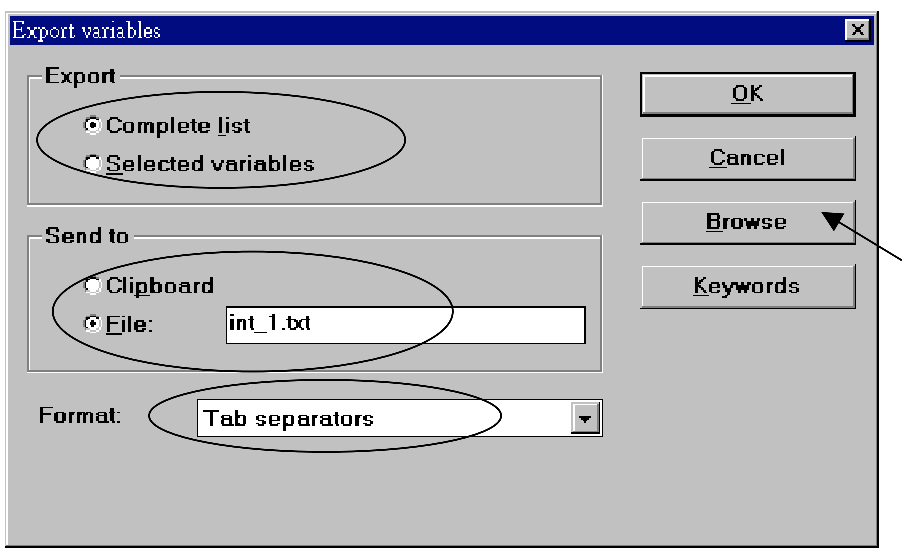
9.11: Export / Import Variable Declarations Via Microsoft Excel

Variables can be defined in Microsoft Excel and then be imported to ISaGRAF workbench. And also they can be exported from ISaGRAF to Excel.

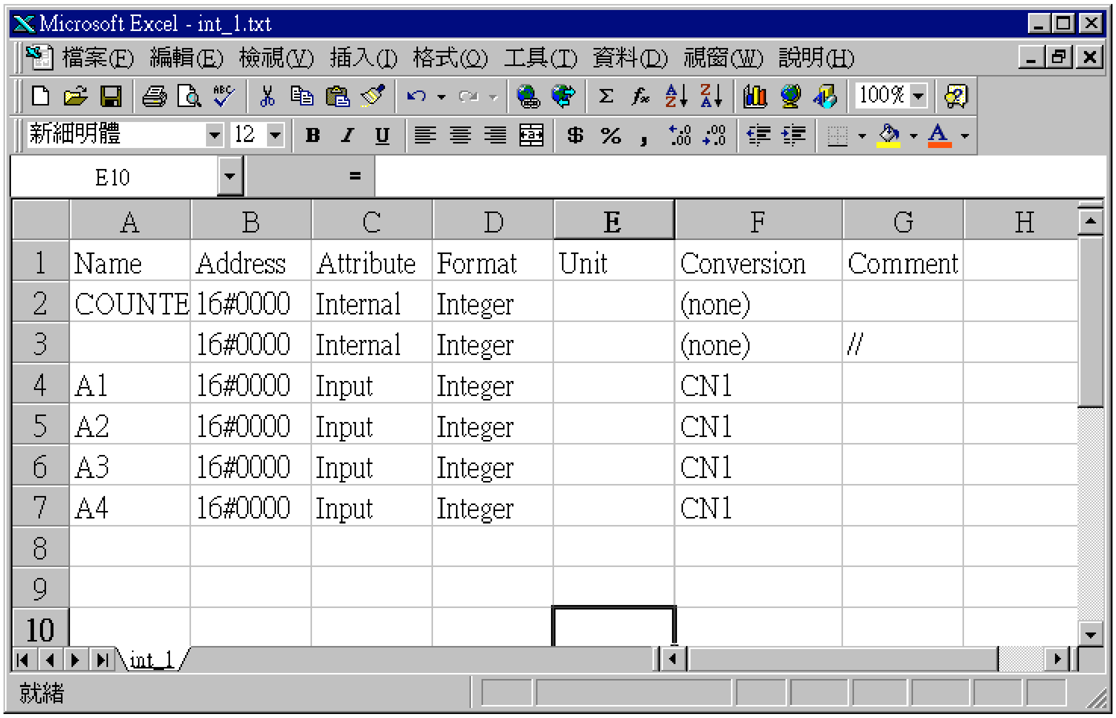
To export to a text file, with an extension name “**.txt**”, run “Tools” - “Export text” from the “dictionary” window.



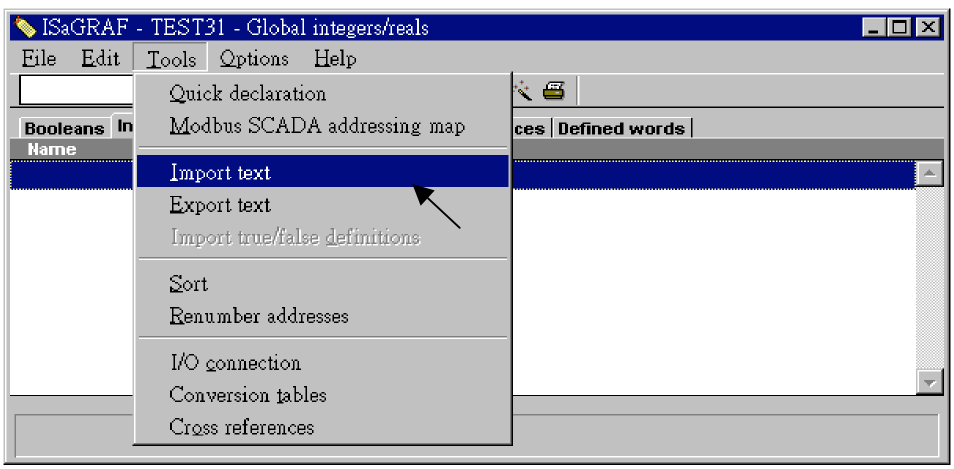
Select “File” and given a name to it, “int_1.txt” in this sample. Then click on “Browse” to select the directory where this txt file will be saved.



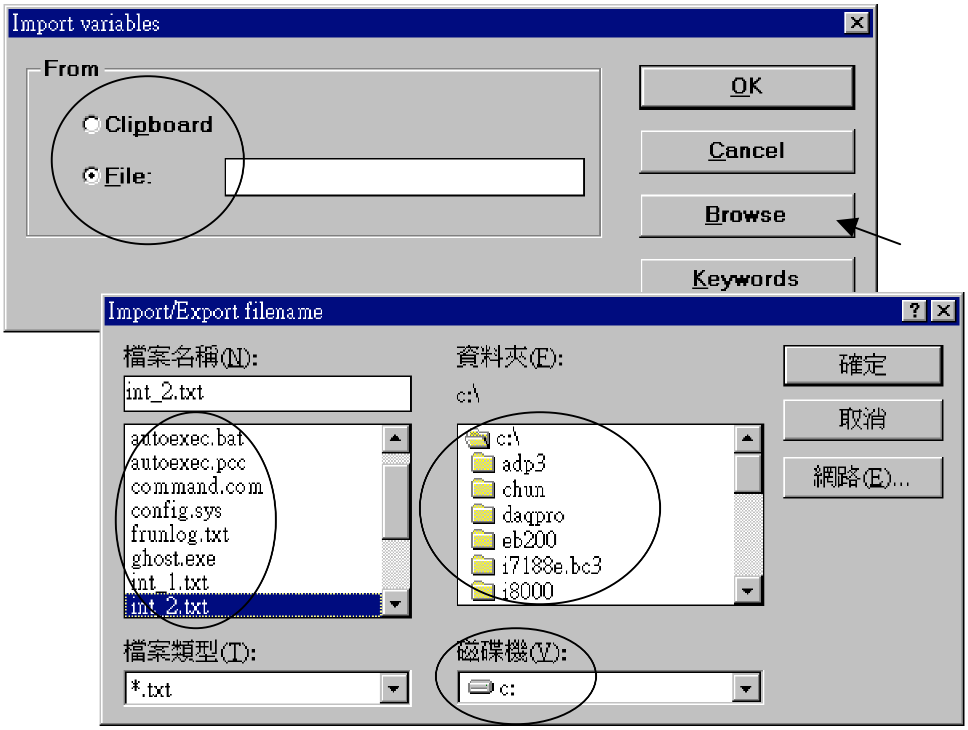
You may open and edit the file from the Excel. Please make sure to save this file with an extension ".txt".



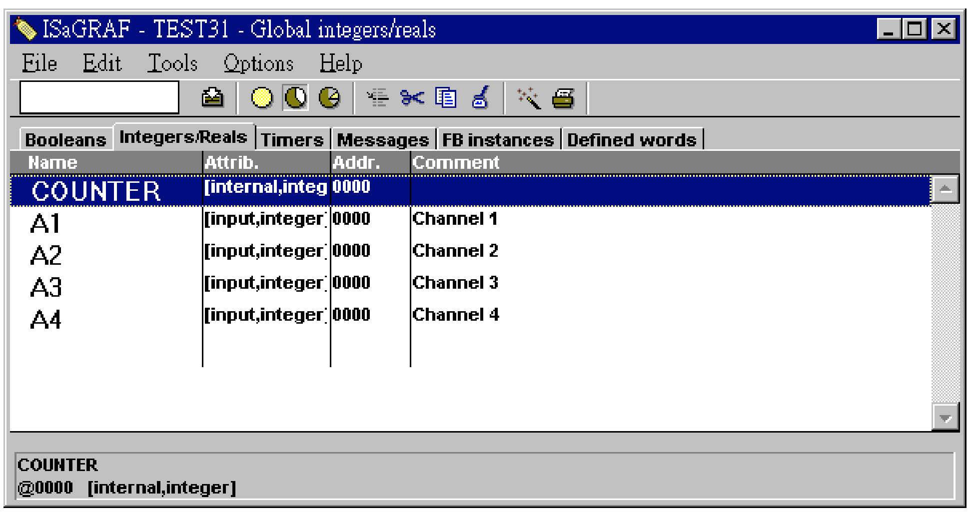
To **import** a text file to ISaGRAF, with an extension name ".txt", run "Tools" - "Import text" from the dictionary window.



Then click on "Browse" to select the associated text file.



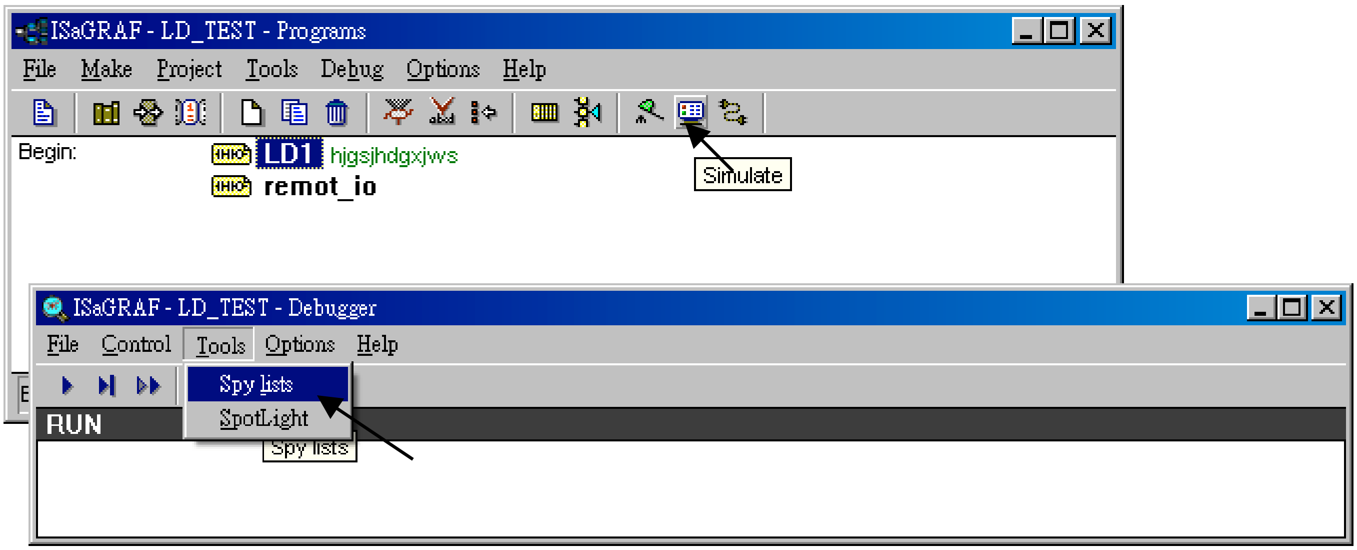
And then it is done as below.



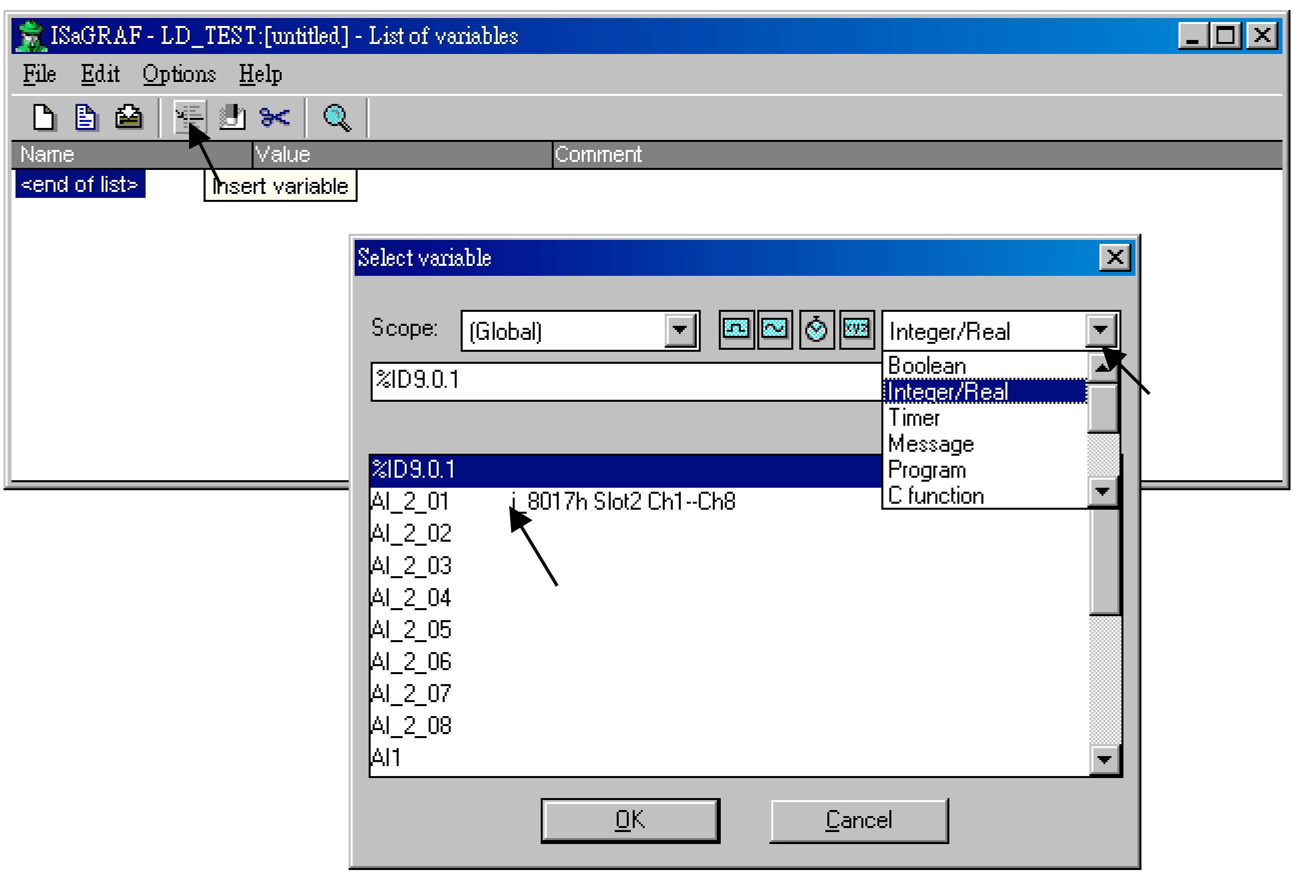
9.12: Spy list

ISaGRAF supports "Spy list" to spy some specific variables when linking to the controller. Please follow below steps to create a "spy list".

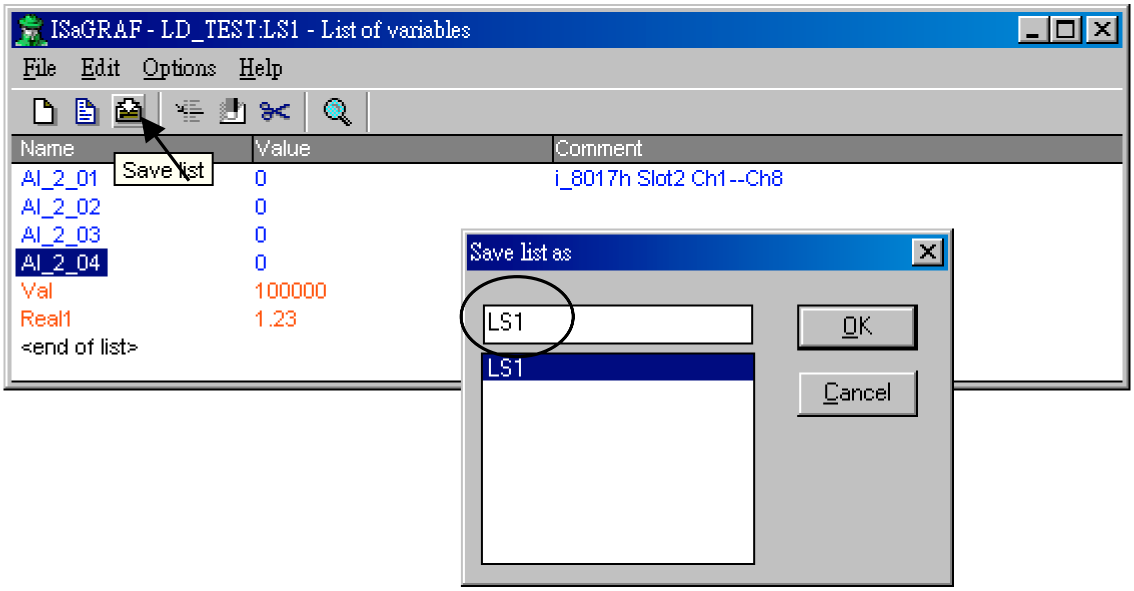
First click on "Simulate", then click on "Tools – Spy list".



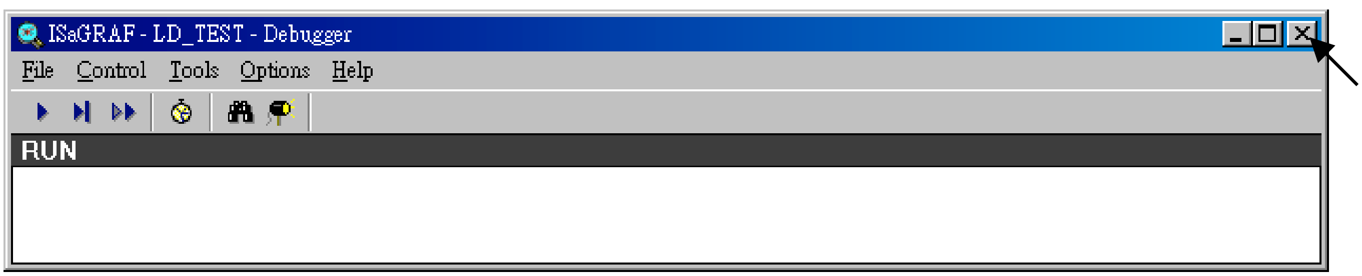
Next click on "Insert variable" to insert the variable to be spied.



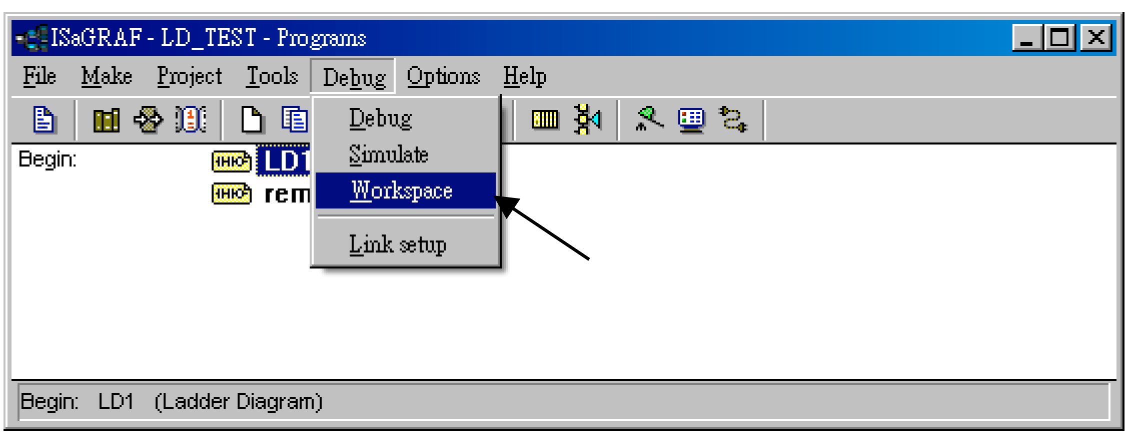
When all spied variables are inserted, remember to click on “Save list”.



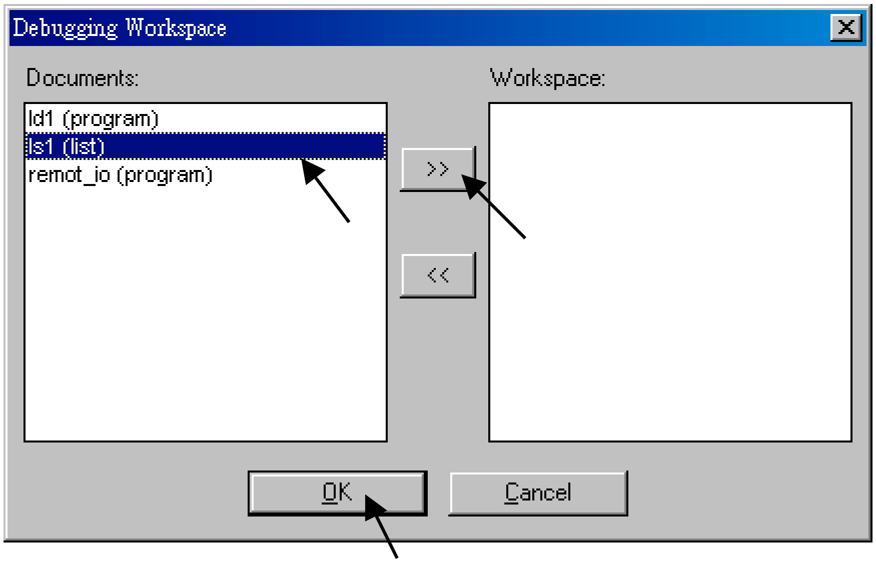
Then close the ”Debugger” window.



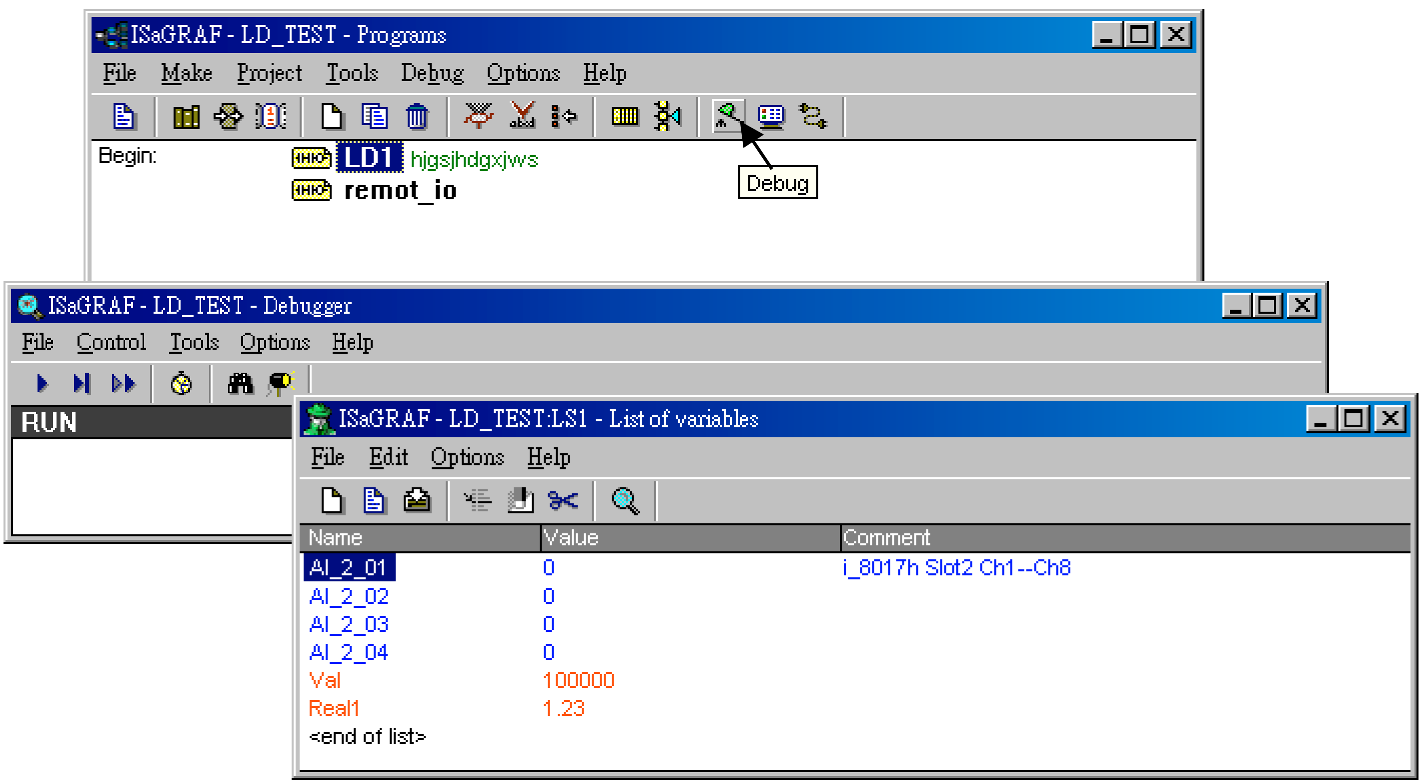
Click on “Debug – Workspace”



Move all “List” to the right hand side.



Then, you will see the “spy list” will automatically display when ISaGRAF linking to the controller.



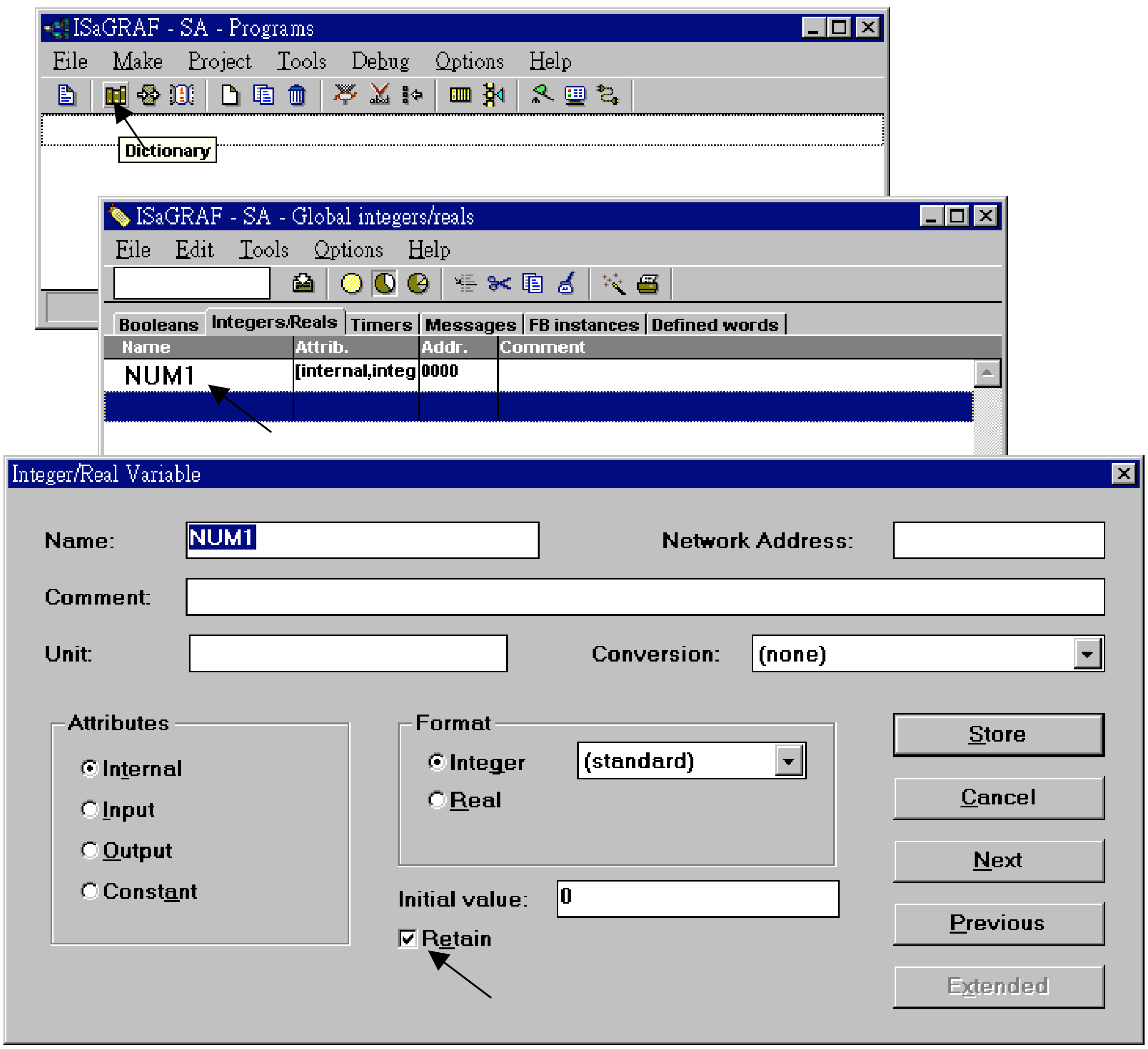
Chapter 10: The Retained Variable And Data Backup

10.1: The Retained Variable

For some real applications, data has to be retained when the power is dead, and these data should be restored to its last value when the power is coming up again. I-8xx7 & I-7188EG/XG controllers (**W-8xx7 doesn't support retained variable, however it supports file operation. Please refer to 10.5 – "Reading & Writting File"**) provide battery backup memories to fit such kind of applications. The battery used can provide the energy to keep the retained variables alive last for some years. It also can provide the energy for the Real-Time-Clock.

A maxinum of **six integers/reals** (signed 32-bit) and **sixteen Booleans** can be retained. If the amount of retained data is more, please refer to Section 10.3 – "Battery Backup SRAM" . If battery backup SRAM is found in the controller (8xx7: S256/S512, 7188EG/XG: X607/X608), the maxinum number of retained variables can be extend to 256 Boolean, 32 Timer and 256 Integer/Real.

To enable the retained function, click on "Retain" for each associated variable.



10.2: Data Backup To The EEPROM

Data can be stored into the EEPROM. The value will be always hold even the power is dead unless the value is updated. The EEPROM of I-8xx7, I-7188EG/XG & W-8xx7 controller can be read freely however can be written only about to 100,000 times.To read a value from the EEPROM, the following functions can be used.

EEP_B_R	Reads one boolean
EEP_BY_R	Reads one byte
EEP_WD_R	Reads one word (2 bytes, signed)
EEP_N_R	Reads one integer (4 bytes, signed)

To write a value to the EEPROM, should remove the protection of the EEPROM first and then write operation is possible. The following functions can be used.

EEP_EN	Removes the protection of EEPROM
EEP_PR	Set the protection of EEPROM
EEP_B_W	Writes a boolean, up to 256 booleans can be stored.
EEP_BY_W	Writes one byte, up to 1,512 bytes can be stored.
EEP_WD_W	Writes one word (2 bytes, signed), up to 756 words can be stored.
EEP_N_W	Writes one integer (4 bytes, signed), up to 378 integers can be stored.

The below two blocks can be used to Read/Save “real” value . To save a Real value to the EEPROM, use Real_Int to map the real value to an integer, and then use EEP_N_W to save this mapped integer. To read a Real value from EEPROM, use EEP_N_R to read it, and then use Int_Real to map this integer to an real value.

Int_Real	Map a long integer to a Real value.
Real_Int	Map a Real value to a long integer.

Bytes, words and integers will be stored to the same memory area in the EEPROM. Be careful to arrange their address before using the above write functions. There are total 1,512 bytes in the EEPROM memory area of the I-8xx7 & I-7188EG/XG, while much more in the W-8xx7.

For I-8xx7 & I-7188EG, the addressing No. of bytes is range from 1 to 1,512, while words is 1 to 756, and integers is 1 to 378. The following No. will use the same memory address in the EEPROM.

Byte	4n-3, 4n-2, 4n-1, 4n	(* n = 1, 2, …378 *)
Word	2n-1, 2n	
Integer	n	

For W-8xx7, the addressing No. of bytes is range from 1 to 14272, while words is 1 to 7136, and integers is 1 to 3568. The following No. will use the same memory address in the EEPROM.

Byte	4n-3, 4n-2, 4n-1, 4n	(* n = 1, 2, …3568 *)
Word	2n-1, 2n	
Integer	n	

When using the write functions, the EEPROM will be damaged if the write operation is more than 100,000 times. For example, the following program is dangerous since the EEPROM will be written once per cycle (normally, the cycle is about 2 to 60 ms depends on the application) .

```
(* ST program, Val is declared as an integer, TEMP is declared as a boolean *)
TEMP := eep_n_w(1, Val);    (* dangerous *)
```

However the following program is safe if Val is not changed frequently.

```
(* ST program, Val, Old_Val declared as integers, TEMP declared as a boolean *)
IF  Val <> Old_Val  THEN
      TEMP := eep_n_w(1, Val);
     Old_Val := Val;
END_IF;
```

Each read / write operation in the EEPROM will consume a lot of CPU time of I-8xx7, I-7188EG/XG & W-8xx7 controller system. The following approximate time is for each function being called.

EEP_EN	~ 0.08 ms	**EEP_PR**	~ 0.08 ms
EEP_B_R	~ 0.8 ms	**EEP_B_W**	~ 6 ms
EEP_BY_R	~ 0.8 ms	**EEP_BY_W**	~ 6 ms
EEP_WD_R	~ 1.5 ms	**EEP_WD_W**	~ 12 ms
EEP_N_R	~ 2.9 ms	**EEP_N_W**	~ 23 ms

Recommend to read values from the EEPROM at one time when the I-8xx7, I-7188EG/XG & W-8xx7 is powered up, and then updated the associated address in the EEPROM when the value is changed. Please refer to a sample program in Chapter 11 – “**demo_17**”. For those data which are frequently changed are not suitable to be stored in the EEPROM.

10.3: Battery Backup SRAM

The I-8417/8817/8437/8837 can integrate with a S256 or S512 battery backup SRAM to store data, alarm, and information, while X607 & X608 for the I-7188EG/XG controller. The data been stored in these SRAM is always retained unless their battery running out of energy. Their memory size is as below, however the upper 12K is reserved by ISaGRAF controllers.

I-8417/8817/8437/8837			
	S256:	244K bytes	(256-12=244)
	S512:	500K bytes	(512-12=500)
I-7188EG/XG			
	X607:	116K bytes	(128-12=116)
	X608:	500K bytes	(512-12=500)

If battery backup SRAM is found in the controller, the maxinum number of retained variables can be extend to as below.

Boolean :	256
Integer + Real :	256
Timer :	32

ICP DAS provides an utility “**ICPDAS UDloader**” that can be installed on the PC to upload and download data from/to the ISaGRAF controller. Please copy “**UDloader.exe**” from the ICP DAS’s CD-ROM:\napdos\isagraf\some_utility\ to your windows.

The I-8417/8817/8437/8837 supports S256/S512 since its driver version of 2.25, while I-7188EG supports X607/608 since its driver version of 1.18, and version 1.16 for I-7188XG. If your driver is older one, please upgrade the hardware driver to the associate version or a higher version. The driver can be found from the below ICP DAS’s web site:

http://www.icpdas.com/products/8000/isagraf.htm

The I/O library should be re-installed if yours is older one. Please refer to section 1.2.
Or you can refer to Appendix A.2 to simply install “C functions” with the below items.

S_B_R,	S_B_W,	S_BY_R ,	S_BY_W,	S_M_R,	S_M_W
S_WD_R,	S_WD_W,	S_N_R ,	S_N_W,	S_R_R,	S_R_W
S_DL_EN,	S_DL_EN,	S_DL_RST,	S_DL_STS		
S_FL_INI,	S_FL_AVL,	S_FL_RST,	S_FL_STS,	S_MV	

and “I/O complex equipment” : S256_S512.

10.3.1: Access to the SRAM

The SRAM can store boolean, byte, word, integer, real & message. Their format is as below.

Boolean:	True=1, False=0	1 byte
Byte:	0 ~ 255	1 byte
Word:	-32768 ~ 32767	2 bytes
Integer:	signed 32-bit	4 bytes
Real:	float	4 bytes
Message:	string (len<=255)	len bytes

To access to the SRAM, the below functions can be used (Please refer to Appendix A).

S_B_R, S_B_W, S_BY_R , S_BY_W, S_M_R, S_M_W
S_WD_R, S_WD_W, S_N_R , S_N_W, S_R_R, S_R_W
S_MV

10.3.2: Upload data stored in the SRAM

For PC to upload data stored in the volatile SRAM of the ISaGRAF controllers, the SRAM should be divided into 1 or up to 8 files. Each file has a ID No. of 1 to 8 and a name of up to 12 characters. The below functions are for handling file format inside the SRAM.

S_FL_INI, S_FL_AVL, S_FL_RST, S_FL_STS

Please use functions of S_FL_INI & S_FL_AVL to arrange the file resident location & current available location (Please refer to Appendix A & demo_40, 41 or 42).

The volatile SRAM is consisted of bytes. The total number of bytes available depends on which module is used as below. The upper 12K is reserved.

Module name	Byte No.
I-8xx7: S256	1 ~ 249,856 (244K), (256-244=12K is reserved)
I-8xx7: S512	1 ~ 512,000 (500K), (512-500=12K is reserved)
I-7188XG/EG: X607	1 ~ 118,784 (116K), (128-116=12K is reserved)
I-7188XG/EG: X608	1 ~ 512,000 (500K), (512-500=12K is reserved)

A file can be located at any place inside these bytes. Each file's location can be described as (**Begin**, **End**). Begin is the lower limit byte No. of the associated file, while End is the upper limit byte No., and Begin is always less than End.

A file inside the SRAM has a current available area (**Head**, **Tail**). Head is the starting position of the file, Tail is the ending position. Head can be larger, less than or equal to Tail.

For ex, a file resides at (Begin, End) = (1, 20000)

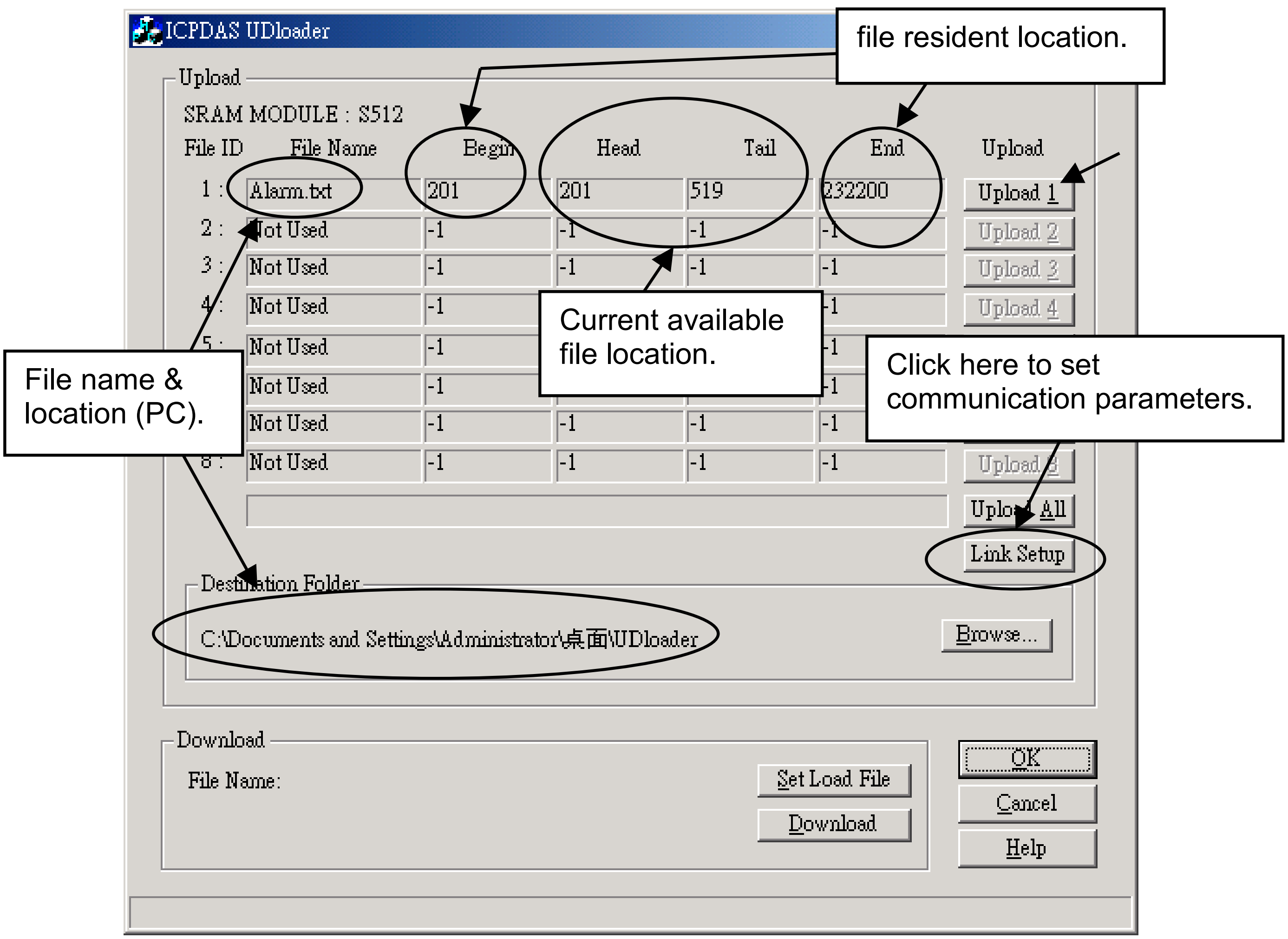
1. If (Head, Tail) = (1001,5100), it means the available data of the file is starting from byte No. of 1001 to 5100. The available file contains 4100 bytes.
2. If (Head, Tail) = (10001,5000), it means the available data of the file is starting from byte No. of 10001 to 20000 and then continued with 1 to 5000. The available file contains 15000 bytes.
3. If (Head, Tail) = (5001,5000), it means the available data of the file is starting from byte No. of 5001 to 20000 and then continued with 1 to 5000. The available file contains 20000 bytes.
4. If (Head, Tail) = (5000,5000), it means the available data of the file is empty, 0 byte.

5. If (Head, Tail) = (-1,-1), it means the available data of the file is empty, 0 byte.

To upload the data stored in the SRAM, please make sure you have installed the “ICPDAS UDloader” on your PC.

To upload data stored in the SRAM of the ISaGRAF controller to PC, please run “UDloader.exe”, then click on “Link Setup” to set proper communication parameters, then click on “Upload 1” to upload it.

Example:
Please download demo_41 to one I-8417/8817/8437/8837. Then push button 1 or 2 or 3 or 4 several times. Then upload the file stored in the SRAM.



10.3.3: Download data to the SRAM

For PC to download data to the volatile SRAM of the ISaGRAF controllers. The below functions can be used. Please refer to Appendix A & demo_44.

S_DL_EN, S_DL_DIS, S_DL_RST, S_DL_STS

Please call S_DL_EN to enable it.

The Controller accepts only the binary format for String, Byte, Word, Int & Real.

Byte:	0 ~ 255	1 byte
Word:	-32768 ~ +32767	2 byte [low bye] [high byte]
Int:	32-bit, signed integer	4 byte [lowest] [2nd] [3rd] [highest]
Real:	32-bit float	4 byte [lowest] [2nd] [3rd] [highest]
String:	up to 255 bytes	

If using the “UDloader.exe” to download data to the volatile SRAM, the data to be downloaded should be edited as a text file. Its format should follow the below rules.

The first line should be a No. indicate that to download to which starting Byte No. of the SRAM.
Valid starting byte No is as below.

S256: 1 ~ 249,856 S512: 1 ~ 512000
X607: 1 ~ 118,784 X608: 1 ~ 512000

The other line is the data.

A. String
String should start and end with the character of ‘ , for ex. ‘Abcd123’ (7 byte). The $NN (NN in hexidecimal and should not equal to 0), could be used to indicate the ASCII character. For ex, ‘ABC$0D’ contains 4 bytes, the 4th byte is <CR>.

B. Byte
Byte should start with (and end with) , for ex. (0) , (123), (255). Valid byte range is from (0) to (255).

C. Word
Word should be start with [and end with] , for ex. [-100] , [20000], [32767]. Valid word range is from [-32768] to [32767].

D. Integer
Integer should be start with { and end with } , for ex. {-1234567} , {200000}. Valid integer range is from {-2147483648} to {2147483647}.

E. Real
Real value should be start with < and end with > , for ex. <123> , <1.56E-2>, <-123.456>.

3. The character between each Byte, Word, Integer, Real, String at the same line should be at least one space character <SP> or , <Comma> or, <Tab>

For ex.

```
201   ← to download to the SRAM which staring from byte No. 201
‘Hello’  (10)  (20)  (30)  (40)  [-10000]  {70000}  ‘End’     ← data (total 18 bytes)
```

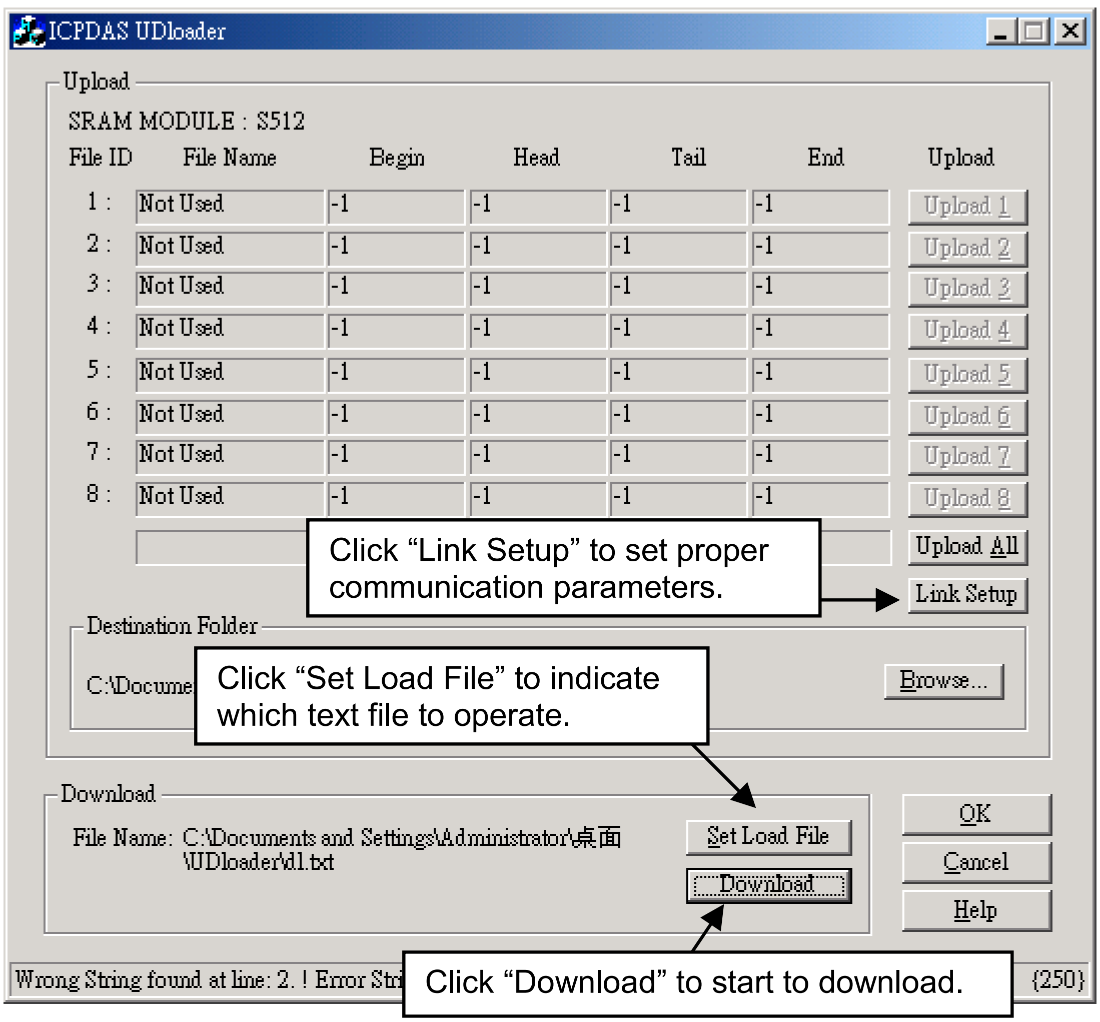
```
1        ← to download to the SRAM which staring from byte No. 1
(23)   ← data (total 57 bytes)
{-1},{2},{-3},{4},{-5},{6} {-7} {8} {-9}    {10}    ← comma, <SP> & <Tab> are all acceptable
<0.123>  <456.789>  <100> , <2.3E3>
```

Example:
Please download demo_44 to one I-8417/8817/8437/8837. Then edit a text file as below.

```
1
{1000}   {250}   {100}    'sTART'
```

The {1000} means the blinking period of L1 is 1000 ms.
The {250} means the blinking period of L2 is 250 ms.
The {100} means the blinking period of L3 is 100 ms. .

Then run "UDloader.exe". You will see something change on the led of the controller.



10.3.4: Operation Functions for the battery backup SRAM

The below functions are for the ISaGRAF controller to access to the volatile SRAM.

S_FL_INI	Init one file’s name & location for the volatile SRAM
S_FL_AVL	Set one file's current available byte No. for the volatile SRAM
S_FL_STS	Get file's Status, end byte No. that has been load by PC for the volatile SRAM
S_FL_RST	Reset file's Status to "Not been load by PC yet" for the volatile SRAM

S_B_R:	Read one Boolean (TRUE, FALSE)
S_BY_R:	Read one Byte (0 ~ 255)
S_WD_R:	Read one Word (-32768 ~ +32767)
S_N_R:	Read one Integer (32 bit, signed)
S_R_R:	Read one Real (32 bit, float)
S_M_R:	Read one String

S_B_W:	Write one Boolean (TRUE, FALSE)
S_BY_W:	Write one Byte (0 ~ 255)
S_WD_W:	Write one Word (-32768 ~ +32767)
S_N_W:	Write one Integer (32 bit, signed)
S_R_W:	Write one Real value (32 bit, float)
S_M_W:	Write one String

S_DL_EN	Enable the download permission for PC to download data to the volatile SRAM
S_DL_DIS	Disable the download permission for PC to download data to the volatile SRAM
S_DL_STS	Get PC’s Download Status for the volatile SRAM
S_DL_RST	Reset the Download Status to “-1:No action” for the volatile SRAM

10.4: Using I-8073 - MultiMediaCard to store data

The I-8073 is not support by I-8xx7, I-7188EG/XG & W-8xx7.

10.5: Reading & Writing File

The W-8037/8337/8737 controller system support file operation however I-8xx7 & I-7188EG/XG doesn't. W-8037/8337/8737 has a Compact Flah Disk with normal size of 128Mbytes (the size depends on the Compact Flash Disk been installed).

The following **ISaGRAF standard** functions are support by W-8xx7.

F_ROPEN	Open an existing binary file in READ mode .
F_WOPEN	Open an existing binary file in READ & WRITE mode .
F_CLOSE	Close an open file
F_EOF	Test if end-of-file has been reached
FA_READ	Read one integer (4 bytes, signed) from a file.
FA_WRITE	Write one integer (4 bytes, signed) to a file open with Write mode.
FM_READ	Read one message (String) from a file.
FM_WRITE	Write one message (String) to a file open with Write mode.

The following functions are support by W-8xx7.

F_CREAT	Creat an empty file for reading & writing .
F_SEEK	Move file position to …
F_READ_B	Read one byte (0 - 255) from a file .
F_WRIT_B	Write one byte (0 - 255) to a file open with Write mode.
F_READ_W	Read one Word (-32768 to +32767) from a file .
F_WRIT_W	Write one byte (-32768 to +32767) to a file open with Write mode.
F_READ_F	Read one float value(For ex. 123.45, -2.15E-03, …) from a file .
F_WRIT_F	Write one float value to a file open with Write mode.

The example programs for file operation reside at the Wincon CD-ROM:

\napdos\isagraf\wincon\demo\ "wdemo_01.pia" & "wdemo_02.pia" or

ftp://ftp.icpdas.com./pub/cd/winconcd/napdos/isagraf/wincon/demo/

Chapter 11: ISaGRAF Programming Examples

When you receive the your I-8xx7, I-7188EG/XG & W-8xx7 controller system, ICP DAS has created a number of ISaGRAF programming examples for them. These example programs are useful for understanding how to program the controller system with the ISaGRAF Workbench software program.

Users may refer to section 11.3 for the description of some demo examples.

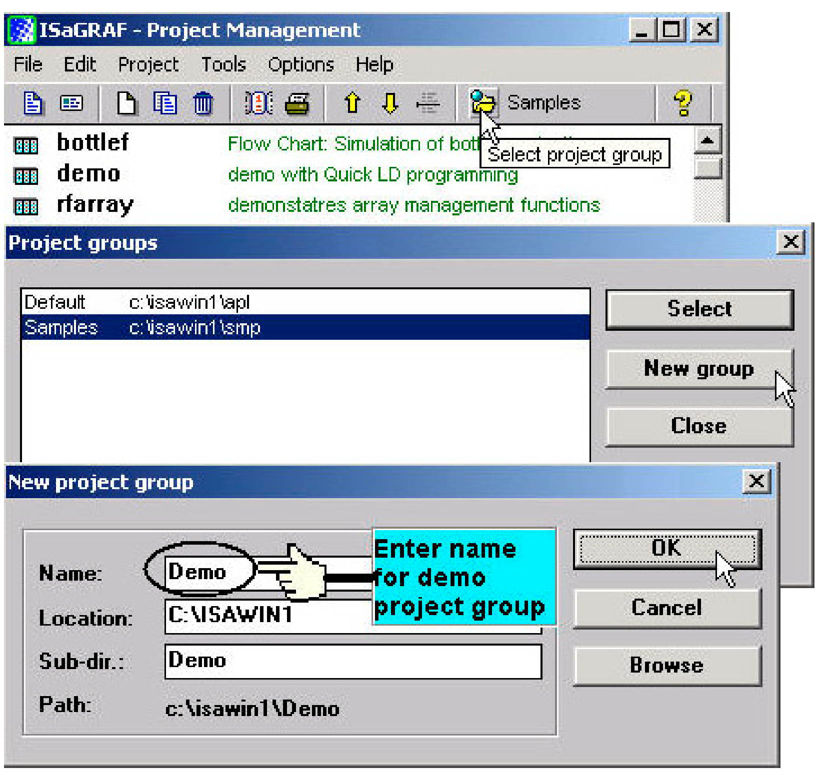
11.1: Installing The ISaGRAF Programming Examples

The ISaGRAF programming examples are installed on the same CD-ROM which the "ICP DAS Utilities For ISaGRAF" resides. The CD-ROM is delivered with the product. You will find the programming example files in the below sub-directory in the CD-ROM.

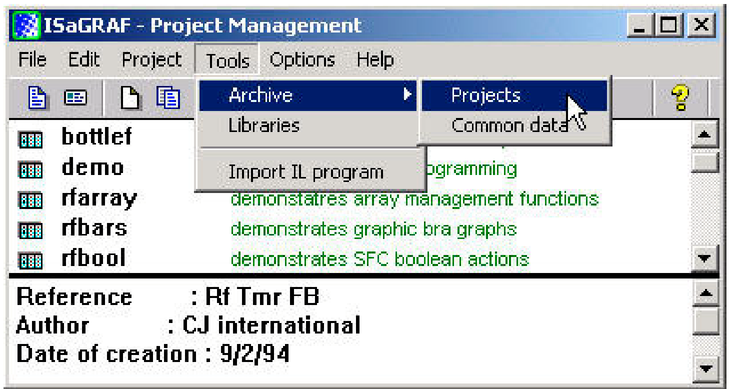
I-8xx7: I-8000 CD-ROM: \napdos\isagraf\8000\demo\
I-7188EG: I-8000 CD-ROM: \napdos\isagraf\7188eg\demo\
I-7188XG: I-8000 CD-ROM: \napdos\isagraf\7188xg\demo\
W-8xx7: Wincon CD-ROM: \napdos\isagraf\wincon\demo\

Or you may download them from below web site:
I-8xx7 & I-7188EG: ftp://ftp.icpdas.com./pub/cd/8000cd/napdos/isagraf/
W-8xx7: ftp://ftp.icpdas.com./pub/cd/winconcd/napdos/isagraf/

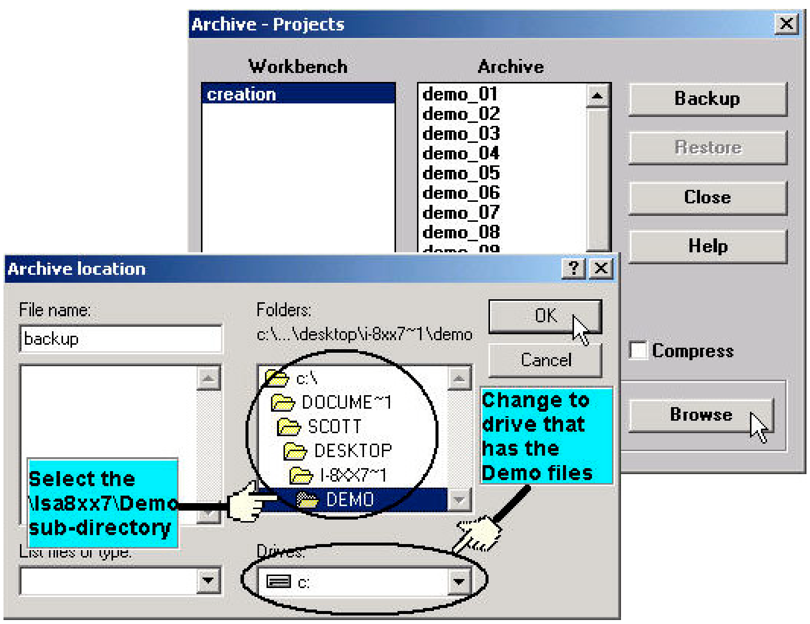
When you install the ISaGRAF example for the controller system it is recommended that you create an "ISaGRAF Project Group" to install the demo program files into.



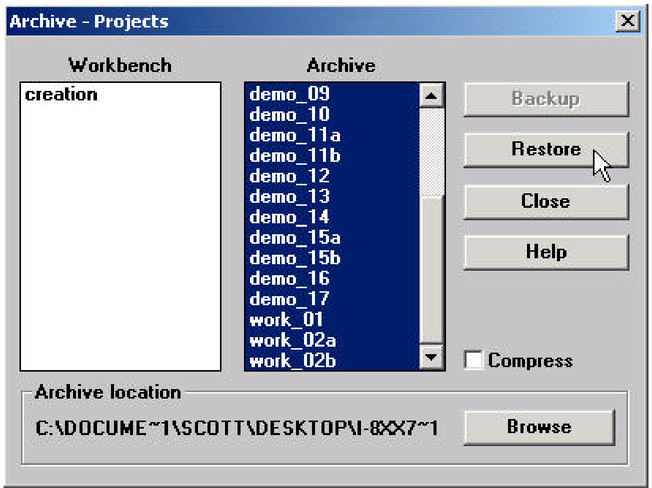
To install the demo programs into the project you have created, open the "ISaGRAF Project Management" window to select "Tools" from the menu bar, then select the "Archive" option and then click on "Projects".



When you click on the "Projects" selection the "Archive Projects" window will open. Click on the "Browse" button to select the drive and the sub-directory where the demo files are located (**For example: Napdos\ISaGRAF\8000\Demo\ on the CD-ROM**) .



To install all of the Demo files, click on the "demo_01" file, then press and hold down the "Shift" key, continue to hold down the "Shift" key and use your mouse to scroll down to last file in the "Archive" window. Click on the last file name from the demo file location and that will select the entire group of demo files. Lastly, click on the "Restore" button in the "Archive Projects" window and all of the demo files will be installed into the sub-directory you have created.


Archive - Projects
Workbench
Archive
creation
demo_09
demo_10
demo_11a
demo_11b
demo_12
demo_13
demo_14
demo_15a
demo_15b
demo_16
demo_17
work_01
work_02a
work_02b
Backup
Restore
Close
Help
Compress
Archive location
C:\DOCUME~1\SCOTT\DESKTOP\I-8XX7~1
Browse

11.2: ISaGRAF Demo Example Files

The following details the contents of the "ISaGRAF Demo" example files for the I-8xx7 & W-8xx7. For example of I-7188EG & I-7188XG, please refer to below folder.

I-7188EG: I-8000 CD-ROM: \napdos\isagraf\7188eg\demo\
I-7188XG: I-8000 CD-ROM: \napdos\isagraf\7188xg\demo\

For the I-8417/8817/8437/8837: I-8000 CD-ROM: \napdos\isagraf\8000\demo

Project Name	Description	I/O Boards Or Complex Equipment Used
Demo_01	Timer Control	Push4Key, Show3Led
Demo_01a	To do something at some sec later when an event happens	Push4Key, Show3Led
Demo_02	Start, Stop, & Reset Timer	Push4Key, Show3Led
Demo_03	R/W System Date & Time To output at a scheduled time interval, For ex. Moday, 09:00 ~ 18:00, Sunday, 10:00 ~ …	NONE
Demo_04	Calculate Empty Cycle Time	NONE
Demo_05	Blinking Output	Push4Key, Show3Led
Demo_06	Change Output Mode	Push4Key, Show3Led
Demo_07	Show A Value To S-MMI	Push4Key, Show3Led
Demo_08	Input A Value To S-MMI	Push4Key, Show3Led
Demo_09	Integer Calculation	NONE
Demo_10	Display Analog Input Value To S-MMI	I-87017, I-87024, Push4Key
Demo_11a	Fbus Master, NET_ID = 1	Fbus_m, Push4Key, Show3Led
Demo_11b	Fbus Slave, NET_ID = 2	Fbus_s, Push4Key
Demo_12	Use COM3 To Receive User-Defined Command From PC	Show3Led
Demo_13	Send User-Defined Data To PC Via COM3 Every 3 Seconds	I-87017
Demo_14	Convert I-7000 & I-87xx Protocol To Modbus Protocol	Bus7000
Demo_15a	Link To Other Modbus Devices	Mbus
Demo_15b	Simulate I-8417 As A Modbus Device For Demo_15a To Link To This Project	None

Project Name	Description	I/O Boards Or Complex Equipment Used
Demo_16	Periodic Pulse Generation, And Send Modbus Commands To Another Controller	Push4Key, Mbus
Demo_17	Read/Write EEPROM	None
Demo_18	PID control	None
Demo_19	Use retained variable to retain Integer	Show3Led
Demo_20	Use retained variable to retain Timer	Show3Led
Demo_21	Write one string to Com5 & Com6	Push4Key, Show3Led
Demo_22	Receive message and echo back to Com5 or Com6	Show3Led
Demo_23	Receive a user defined protocol from PC	Show3Led
Demo_27	Motion x, slot 0: i-8091, Slot 1:i-8090, Napdos\ISaGRAF\8000\Driver\motion.pdf	8091 I-8090 Show3Led
Demo_28	Motion x-y, slot0: i-8091, slot1: i-8090, Napdos\ISaGRAF\8000\Driver\motion.pdf	8091 I-8090 Show3Led
Demo_29	Store 1200 short-int values every 75 sec. and then send to PC via Com3	I-87017
Demo_30	Store 2880 short-int values every 18 sec. and then send to PC via Com3	I-8017h
Demo_31	Press push button 1 to send an email from Com4 of I-8xx7 controller	Push4Key
Demo_32	Press Push button 1 or 2 or 3 to send emails to two users with multi-buffers	Push4Key
Demo_33	R/W user defined protocol via Com3	Show3Led
Demo_35a	Time Synchronization : SA Update Date & Time at this controller will sychronize date & time at SB	Fbus_m
Demo_35b	Time Synchronization : SB	Fbus_s
Demo_37	Spotlight demo	Push4Key Show3Led
Demo_38	I-8xx7 talks to the MMICON : Demo 1	MMICON
Demo_39	8xx7 talks to the MMICON : Demo 2	MMICON
Demo_40	store 8 A/I (binary) to S256 per min, then PC can load it by "ICPDAS UDloader"	I-8017h S256_512 Show3Led
Demo_41	Record Alarm (text) to S256/512 & PC can load it by "ICPDAS UDloader"	S256_512 Show3Led
Demo_42	store 8 A/I (text) to S256 per min, then PC can load it by "ICPDAS UDloader"	I-8017h S256_512 Show3Led
Demo_43	SMS demo, Please declare your own phone No. in the dictionay, message type	SMS Show3Led Push4key

Project Name	Description	I/O Boards Or Complex Equipment Used
Demo_44	Demo of PC to download data to the S256/512	Show3Led
Demo_46	Motion control: Pulse move at a specified speed	I-8091 I-8090 Push4Key
Demo_49a	Redundant: 8437/8837 redundant Master	Bus7000 Ebus_m
Demo_49b	Redundant: 8437/8837 redundant slave	Bus7000 Ebus_s
Demo_50	PWM I/O demo. (Pulse Width Modulation)	I-8055
Demo_52	Parallel D/I counter demo 1 at slot 0 (Counter Value is retained in this demo)	I-8051 Push4Key
Demo_53	Parallel D/I counter demo 2 at slot 0 (high speed near 1K) (Not retained)	I-8051 I-8056 Push4key
Demo_55	PWM I/O demo 2. (Pulse Width Modulation)	I-8055
Demo_61	DI counters using DI_CNT, 8xx7 + 8051 Do somethig when DI signal happens	I-8051

NOTE:
Demo_18 uses PID_AL which is provided by CJ International for evaluation. Please refer to "CD\Napdos\isagraf\8000\english_manu\ PID_AL.ComplexPIDalgorithm implementation.htm".

Visual Basic Demo program:

I-8000 CD-ROM:\napdos\isagraf\vb_demo\ or
ftp://ftp.icpdas.com/pub/cd/8000cd/napdos/isagraf/vb_demo

Project Name	Description	I/O Boards Or Complex Equipment Used
Demo_1	PC access to I-8437/8837 by Modbus TCP/IP protocols	I-8437/8837 I-8054
Demo_2	PC access to the remote I-8417/8817/8437/8837 via a Modem with a phone line (Please refer to Chapter 13)	I-84x7/88x7 I-87064 Modem Phone line

For the W-8037/8337/8737: **Wincon CD-ROM: \napdos\isagraf\wincon\demo**

Project Name	Description	I/O Boards Or Complex Equipment Used
wdemo_01	R/W float value from file	
wdemo_02	R/W long integer value from file	
wdemo_03	To output something at a scheduled time interval: For ex. Moday, 09:00 ~ 18:00, Sunday, 10:00 ~ …	
wdemo_04	User defined Modbus protocol (No using "Mbus")	
wdemo_05	To do something at some sec later when an event happens	i-8055
wdemo_26	To move some pulse at x-axis of i-8091 of slot 1 in W-8337/8737	i-8091
wdemo_27	Motion x, slot 1: i-8091, slot 2: i-8090, Napdos\ISaGRAF\8000\Driver\motion.pdf	i-8091 i-8090
wdemo_28	Motion x-y, slot1: i-8091, slot2: i-8090, Napdos\ISaGRAF\8000\Driver\motion.pdf	i-8091 i-8090
wdemo_29	Moving to he Abs. position when CMD is given, slot 1 : i-8091, slot 2: i-8090	i-8091 i-8090

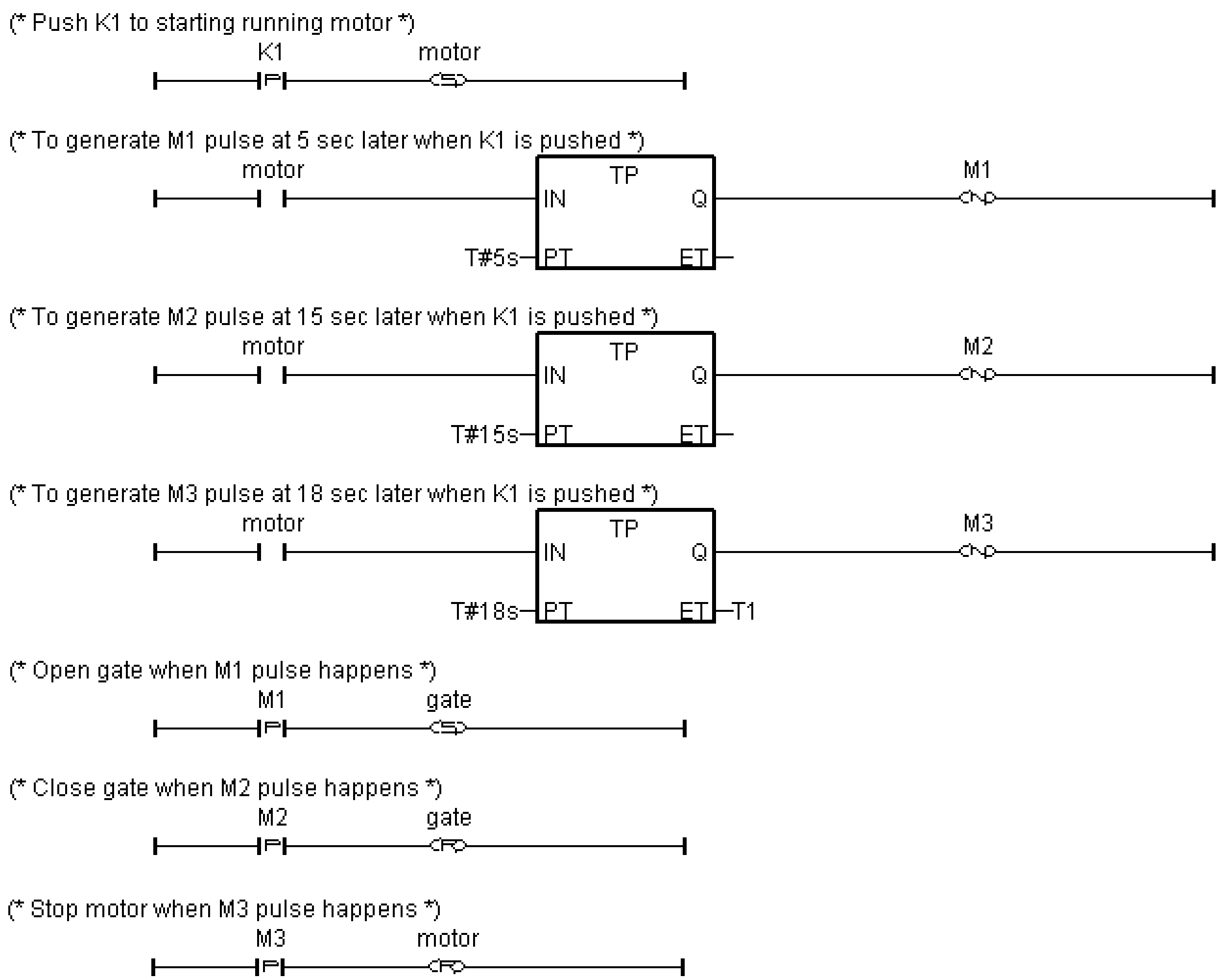
11.3: Description Of Some Demo Examples

11.3.0 Demo_01A & Demo_03: Do something at specific time

Demo_01A: Do something at some seconds later when an event happens.

Location: I-8000 CD-ROM: \napdos\isagraf\8000\demo\ “demo_01a.pia”
Variables :

Name	Type	Attribute	Description
K1	Boolean	**Input**	push K1 to start running motor (pushbutton 1 on the I-8xx7)
Motor	Boolean	**Output**	True means to run motor, False means to stop motor
Gate	Boolean	**Output**	True means to open gate, False means to close gate
M1	Boolean	**Internal**	event generated at 5 sec later when K1 is pushed
M2	Boolean	**Internal**	event generated at 15 sec later when K1 is pushed
M3	Boolean	**Internal**	event generated at 18 sec later when K1 is pushed
T1	Timer	**Internal**	Time past



Demo_03: Do something at specific weekday & some time interval

Location: I-8000 CD-ROM: \napdos\isagraf\8000\demo\ “demo_03.pia”

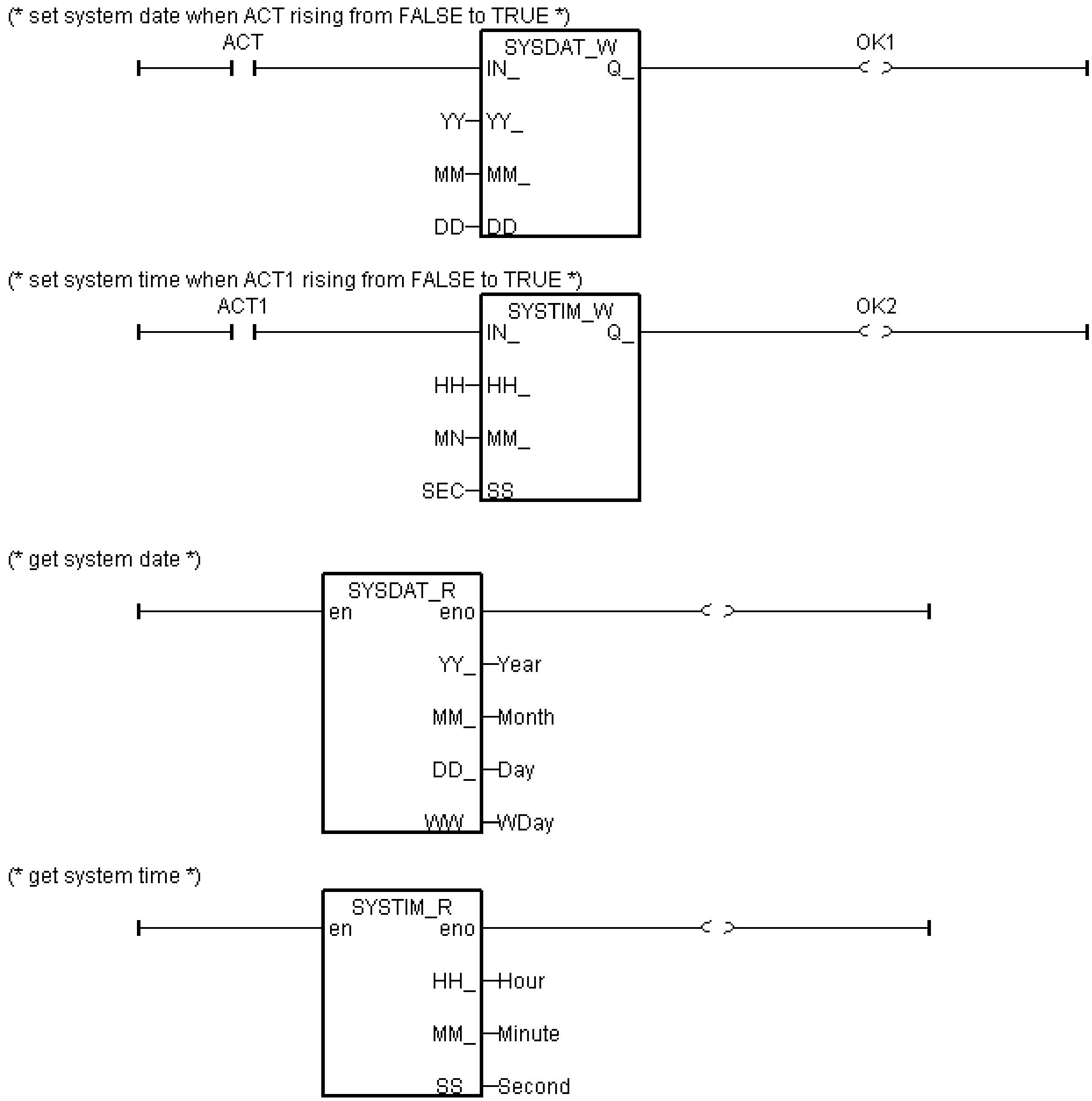
Variables :

Name	Type	Attribute	Description
Year	Integer	**Internal**	System year, 2001 ~
Month	Integer	**Internal**	System Month, 1 ~ 12
Day	Integer	**Internal**	System date, 1 ~ 31
Wday	Integer	**Internal**	System Wday, 1:Monday ~ 6:Saturday, 7:Sunday
Hour	Integer	**Internal**	System hour, 0 ~ 23
Minute	Integer	**Internal**	System minute, 0 ~ 59
Second	Integer	**Internal**	System second, 0 ~ 59
YY	Integer	**Internal**	New system year to set
MM	Integer	**Internal**	New system month to set
DD	Integer	**Internal**	New system date to set
HH	Integer	**Internal**	New system hour to set
Mn	Integer	**Internal**	New system minute to set
Sec	Integer	**Internal**	New system second to set
Act	Boolean	**Internal**	Trigger to set new date
Act1	Boolean	**Internal**	Trigger to set new time
OK1	Boolean	**Internal**	Read back of “SYSDAT_W”
OK2	Boolean	**Internal**	Read back of “SYSTIM_W”
L1 ~ L3	Boolean	**Internal**	Simulate Boolean Output 1 to 3
Time_val	Integer	**Internal**	unit is sec, = 3600 x hour + 60 x minute + sec, every day = 0~86399

Operation action:

1. Monday ~ Saturday, L1 ~ L3, 09:00:00 ~ 18:00:00 ON
2. Sunday, L1, 13:00:00 ~ 20:00:00 ON
3. Other time, L1 ~ L3 are all OFF

Ladder program : get_time



ST program : control

```
time_val := 3600*hour + 60*minute + second;  (* calculate time in sec. *)

(* set as False at the beginning of this ST program*)
L1 := False;
L2 := False;
L3 := False;

(* Monday ~ Saturday, L1 ~ L3, 09:00:00 ~ 18:00:00 ON *)
IF (Wday >= 1) AND (Wday <= 6)THEN
   IF (time_val >= 32400) AND (time_val <= 64800) THEN
      L1 := True;
      L2 := True;
      L3 := True;
   END_IF;
END_IF;

(* Sunday, L1, 13:00:00 ~ 20:00:00 ON *)
IF (Wday = 7) THEN
   IF (time_val >= 46800) AND (time_val <= 72000) THEN
      L1 := True;
   END_IF;
END_IF;
```

11.3.1 Demo_02 : Start, Stop And Reset Timer

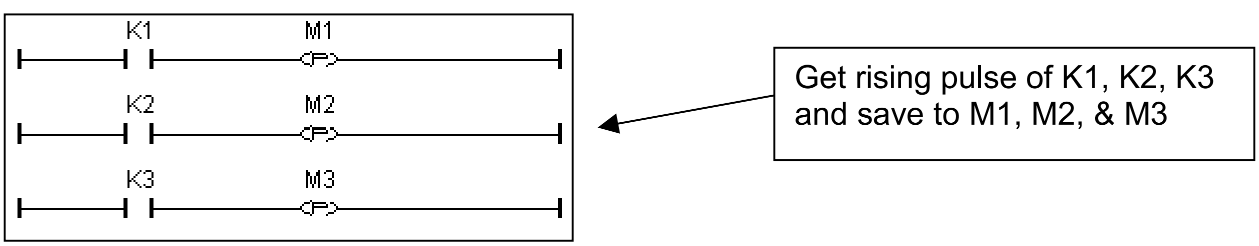
Location: I-8000 CD-ROM: \napdos\isagraf\8000\demo\ “demo_02.pia”

Project architecture:

Variables :

Name	Type	Attribute	Description
M1	Boolean	**Internal**	Indicate a rising pulse of K1
M2	Boolean	**Internal**	Indicate a rising pulse of K2
M3	Boolean	**Internal**	Indicate a rising pulse of K3
K1	Boolean	**Input**	Pushbutton 1
K2	Boolean	**Input**	Pushbutton 2
K3	Boolean	**Input**	Pushbutton 3
L1	Boolean	**Output**	Output 1
L2	Boolean	**Output**	Output 2
L3	Boolean	**Output**	Output 3
T1	Timer	**Internal**	Operation timer, **initial value is set at "T#0s"**

LD program “prg1” :



ST program “demo” :

```
(* Start timer *)
IF M1 THEN
  TSTART(T1);
END_IF;

(* Stop timer *)
IF M2 THEN
  TSTOP(T1);
END_IF;

(* Reset timer *)
IF M3 THEN
  T1 := T#0s;
END_IF;

(* Output L1 ~ L3 *)
L1 := ( T1 > T#2s ) AND ( T1 < T#15s );
L2 := L1;
L3 := L1;
```

“TSTART” will start ticking the “T1” timer

“TSOP” will stop ticking “T1” timer

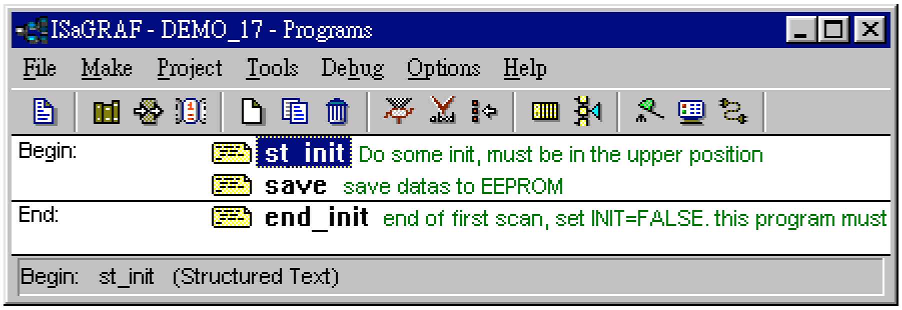
Reset “T1” timer to 0 sec.

“L1” will be TRUE between 2 and 15 sec of the value of “T1”

11.3.2 Demo_17 : R/W Integer Value From/To The EEPROM

Location: I-8000 CD-ROM: \napdos\isagraf\8000\demo\ “demo_17.pia”

Project architecture:



Variables:

Name	**Type**	**Attribute**	**Description**
V1	Integer	**Internal**	Change value of V1 to save new value to EEPROM
V2	Integer	**Internal**	
V3	Integer	**Internal**	
V4	Integer	**Internal**	
V5	Integer	**Internal**	
V6	Integer	**Internal**	
V7	Integer	**Internal**	
V8	Integer	**Internal**	
Old_V1	Integer	**Internal**	Old value of V1
Old_V2	Integer	**Internal**	
Old_V3	Integer	**Internal**	
Old_V4	Integer	**Internal**	
Old_V5	Integer	**Internal**	
Old_V6	Integer	**Internal**	
Old_V7	Integer	**Internal**	
Old_V8	Integer	**Internal**	
TEMP	Boolean	**Internal**	for temporal use
INIT	Boolean	**Internal**	If controller is just powered up, **initial value is TRUE**

ST program “st_init” :

```
if INIT=TRUE then  (* First scan cycle *)
   (* Read 8 integers from EEPROM *)
   (* save them to Old_V1 ~ 8 , V1 ~ V8 *)
   Old_V1 := eep_n_r(1);
   Old_V2 := eep_n_r(2);
   Old_V3 := eep_n_r(3);
   Old_V4 := eep_n_r(4);
   Old_V5 := eep_n_r(5);
   Old_V6 := eep_n_r(6);
   Old_V7 := eep_n_r(7);
   Old_V8 := eep_n_r(8);
   V1 := Old_V1;
   V2 := Old_V2;
   V3 := Old_V3;
   V4 := Old_V4;
   V5 := Old_V5;
   V6 := Old_V6;
   V7 := Old_V7;
   V8 := Old_V8;

   (* remove protection of EEPROM *)
   TEMP := eep_en();

end  if;
```

Read long integers stored at the position from 1 to 8 of the eeprom at the first scan cycle.

Init V1 to V8

Remove the protection of EEPROM, so that it can be written later.

ST program “save” :

```
(* save V1 ~ V8 to EEPROM *)
(* You will find write to EEPROM take lots of time, about 23ms for each eep_n_w *)
  IF V1 <> Old_V1 THEN TEMP := eep_n_w(1,V1); Old_V1 := V1; END_IF;
IF V2 <> Old_V2 THEN TEMP := eep_n_w(2,V2); Old_V2 := V2; END_IF;
IF V3 <> Old_V3 THEN TEMP := eep_n_w(3,V3); Old_V3 := V3; END_IF;
IF V4 <> Old_V4 THEN TEMP := eep_n_w(4,V4); Old_V4 := V4; END_IF;
IF V5 <> Old_V5 THEN TEMP := eep_n_w(5,V5); Old_V5 := V5; END_IF;
IF V6 <> Old_V6 THEN TEMP := eep_n_w(6,V6); Old_V6 := V6; END_IF;
IF V7 <> Old_V7 THEN TEMP := eep_n_w(7,V7); Old_V7 := V7; END_IF;
IF V8 <> Old_V8 THEN TEMP := eep_n_w(8,V8); Old_V8 := V8; END_IF;
```

The value will be saved to eeprom only when the current value is changed.

Then update Old value to the new value.

ST program “end_init” :

```
if INIT=TRUE then
  INIT := FALSE;    (* end of first PLC scan *)
end_if;
```

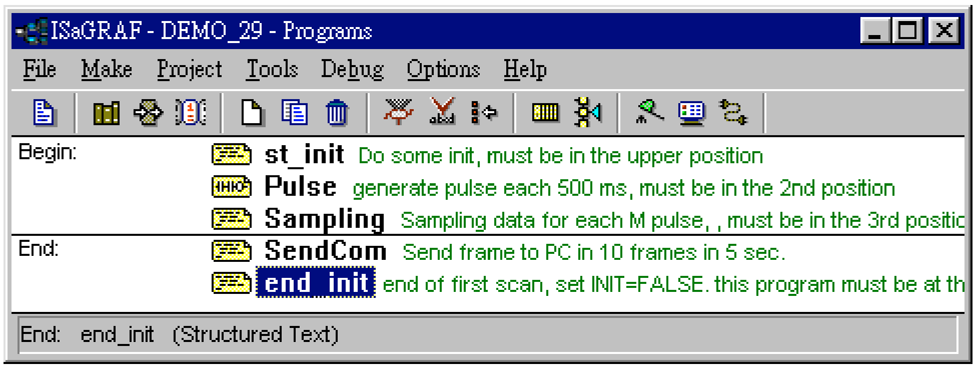
Set “INIT” to False, so that “INIT” is only TRUE at the first scan cycle since it is declared with the initial value - TRUE.

11.3.3 Demo_29: Store 1200 Short Int Every 75 sec & Send To PC Via Com3

This demo program is to save the 8 analog input value (8 samples) of the I-87017 to the short-integer array every 500ms. Then when the number of samples reach 1200, these samples will be divided in 10 frames, each frame contain 120 samples, and sent to one PC via COM3 (RS232/RS485).

Location: I-8000 CD-ROM: \napdos\isagraf\8000\demo\ “demo_29.pia”

Project architecture:



Variables :

Name	Type	Attribute	Description
M	Boolean	**Internal**	pulse to store a sample
M1	Boolean	**Internal**	pulse to send frame
M2	Boolean	**Internal**	To generate M1 pulse
INIT	Boolean	**Internal**	If controller is just powered up, **initial value is TRUE**
TMP	Boolean	**Internal**	For temporal use
A1	Integer	**Input**	Connect to Ch. 1 of I-87017
A2	Integer	**Input**	Connect to Ch. 2 of I-87017
A3	Integer	**Input**	Connect to Ch. 3 of I-87017
A4	Integer	**Input**	Connect to Ch. 4 of I-87017
A5	Integer	**Input**	Connect to Ch. 5 of I-87017
A6	Integer	**Input**	Connect to Ch. 6 of I-87017
A7	Integer	**Input**	Connect to Ch. 7 of I-87017
A8	Integer	**Input**	Connect to Ch. 8 of I-87017
count	Integer	**Internal**	No. of sample(1~1200) that is processing, **init value=1**
position	Integer	**Internal**	position in current short integer array, 1 ~ 256
No	Integer	**Internal**	current short integer array No. which is processing
Frame_No	Integer	**Internal**	only = 0 ~ 10
TMP_VAL	Integer	**Internal**	For temporal use

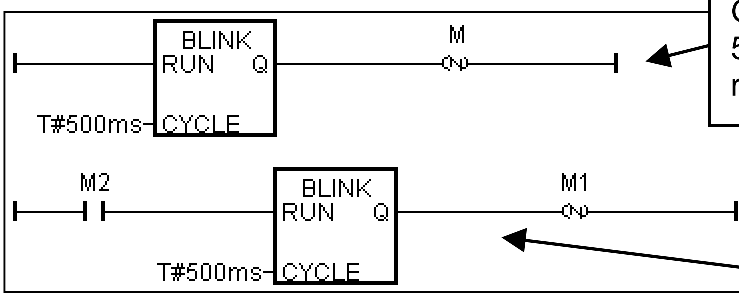
ST program “st_init” :

```
if INIT=TRUE then  (* First scan cycle *)

  COUNT:=1; (* init count to 1 *)
  FRAME_NO:=0; (* init to 0 *)
  M2 := False; (* init M2 to false *)
  TMP:=comopen(3,9600,8,0,1);

end_if;
```

Do some init at first scan cycle

Open Com3 as baud=9600, char. size=8, no parity & stop bit=1

LD program “Pulse” :



Generate M pulse every 500 ms, M pulse is used to record the A/I sample value

Generate M1 pulse every 500 ms when “M2” is set to TRUE, M1 pulse is used to send one frame to PC

ST program “Sampling” :

```
(* no M pulse, return *)
if M=False then
  return;
end_if;

No:=((COUNT-1)/240)+1;  (* which array No *)
position:=COUNT-240*(No-1);  (* which start position of the array *)

(* Store I-87017 : 8 A/I value to short integer array *)
(* Please keep in mind, the max No of short int arry can be used is 12 *)
TMP:=ary_w_w(No,position,A01);
TMP:=ary_w_w(No,position+1,A02);
TMP:=ary_w_w(No,position+2,A03);
TMP:=ary_w_w(No,position+3,A04);
TMP:=ary_w_w(No,position+4,A05);
TMP:=ary_w_w(No,position+5,A06);
TMP:=ary_w_w(No,position+6,A07);
TMP:=ary_w_w(No,position+7,A08);

if (COUNT+7 >= 1200) then
  FRAME_NO := 1;     (* set FRAME_NO=1 *)
  M1 := True;
  M2 := True;
  COUNT := 1;  (* reset COUNT to 1 *)
else
  COUNT := COUNT+8; (* next sampling *)
end_if;
```

This will make the following statement only be processed when M pulse is generated

Get current array No. and the start pos. of the array which the samples will save to. Each array can store up to 240 samples

Store I-87017 : 8 A/I value to short integer array

If number of stored samples reach 1200, send to PC via Com3 in 10 frames

set M1=True to send first frame at next ST program - "SendCom"

set M2=True to generate M1 pulse at previouse LD program - "Pulse"

If the number of stored samples hasn’t reached 1200 yet, pulse “COUNT” by 8 for next sampling.

ST program “SendCom” :

```
If M1=False then
  return;
end_if;
```

This will make the following statement only be processed when M1 pulse is generated

User defined frame format : Each contains 120 short integers

	STX	DLE	FRAME_NO	DATA	ETX
number of bytes	1	1	1	120x2	1
value	0x2	0x10	1~10	?	0x03

When “FRAME_NO” is between 1 and 10

```
if( (FRAME_NO>=1) and (FRAME_NO<=10) ) then

  No:=(FRAME_NO-1)/2+1;
```

Get the short integer array No to process. Keep in mind, each array strore up to 240 samples. (in other word -- 2 frames)

```
  case (FRAME_NO-2*(No-1)) of
    1:  position := 1;
    2:  position := 121;
  end_case;
```

Get starting position inside the array

Send one frame via Com3

```
  TMP := comwrite(3,16#2);     (* write one byte = STX to Com3 *)
  TMP := comwrite(3,16#10);     (* write one byte = DLE to Com3 *)
  TMP := comwrite(3,FRAME_NO);     (* write frame No = 1 ~ 10 to Com3 *)

  (* write 120 short integers inside the array to Com3 *)
  TMP := comay_ww(3,No,120,position);

  TMP := comwrite(3,16#3);     (* write one byte = ETX to Com3 *)

  M1 := False;
```

500 ms later, send next frame. “M1” will be turned ON after 500 ms later at LD program - "Pulse"

```
  if (FRAME_NO=10) then
    FRAME_NO := 0;
    M2 := False;
  else
    FRAME_NO := FRAME_NO+1;   (* for next cycle *)
  end_if;

end_if;
```

If all frames are sent, reset “FRAME_NO” to 0 , and set “M2” to FALSE to stop to generate “M1”

If some frames have not been sent yet, plus “FRAME_NO” by 1 to send next frame when next “M1” is generated

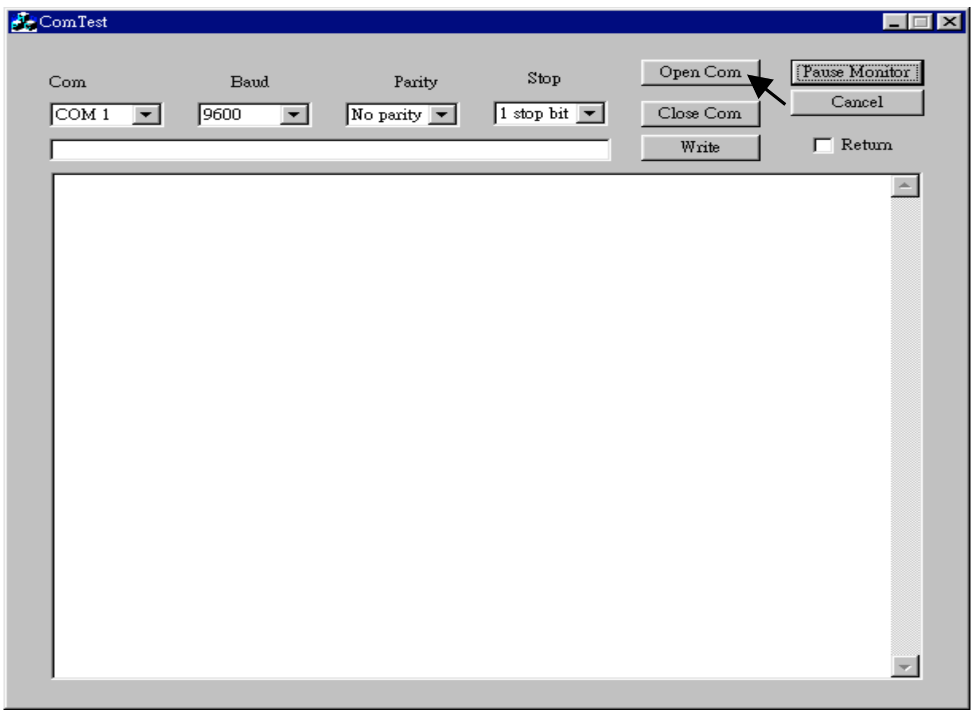
ST program “end_init” :

```
if INIT=TRUE then
  INIT := FALSE;    (* end of first PLC scan *)
end_if;
```

Set “INIT” to False, so that “INIT” is only TRUE at the first scan cycle since it is declared with the initial value - TRUE.

How to test ?

Plug one I-87017 in the slot 0 of the I-8xx7 controller.
Download Demo_29 to the controller.
Prepare a RS232 cable to connect Com3 of the controller to Com1 of your PC.
There is one ultilty named “ComTest.exe” located in the ICP DAS’s CD-ROM. Copy it to your PC. “\Napdos\ISaGRAF\some_utility\Comtest.exe” or you may obtain it from below site. ftp://ftp.icpdas.com/pub/cd/8000cd/napdos/isagraf/some_utility/
Execute “ComTest” and select the parameter to “COM1” , “9600” , “No parity” , “1 stop bit” and then click on “Open Com”.



You will receive 10 frames coming from the target controller every 75 seconds.

11.3.4 Demo_33 : R/W User Defined protocol Via Com3:RS232/RS485

This demo program can let I8417/ 8817/ 8437/ 8837 accept commands coming from PC via a RS232 cable. The command protocol format can be defined by the user. We use the below protocol format in this example.

Command is case insensitive, that means M1 & m1 are same

Protocol Format:

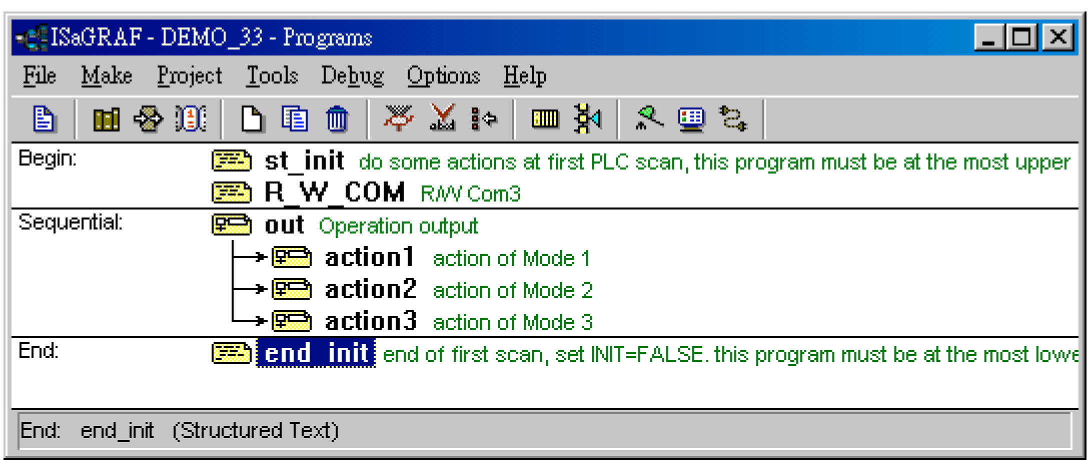
PC req.
M1<CR> : Change to Mode 1
M2<CR> : Change to Mode 2
M3<CR> : Change to Mode 3
Txxxx<CR> : Change Period time to xxxx ms
for ex. T250<CR> will change period time to 250ms
Controller Ans.
OK<CR>

PC req.
M?<CR> : Request the current Mode
Controller Ans.
Mx<CR> : for ex. M1 means Mode 1

PC req.
T?<CR> : Request the current Period time
Controller Ans.
Txxxx<CR> : for ex. T1500 means Period time is 1500ms

Timeout:
a valid command should be completely sent in 5 sec.

Project architecture:



Variables :

Name	Type	Attribute	Description
L1	Boolean	**Output**	Output 1
L2	Boolean	**Output**	Output 2
L3	Boolean	**Output**	Output 3
INIT	Boolean	**Internal**	If controller is just powered up, **initial value is TRUE**
TMP	Boolean	**Internal**	For temporal use
Mode	Integer	**Internal**	Operation Mode, range from 1 to 3
Step	Integer	**Internal**	Processing step
NUM	Integer	**Internal**	Received valid byte number
Num_com3	Integer	**Internal**	return value of Comary_R
byt	Integer	**Internal**	Current operating byte
index	Integer	**Internal**	Index of byte array
CMD	Integer	**Internal**	command type, M, m, T, or t
TMP_val	Integer	**Internal**	for temporal use
ii	Integer	**Internal**	for temporal use
T1	Timer	**Internal**	Period time, valid range is 50 ~ 9999 ms
tout	Timer	**Internal**	timer to measure timeout, tick when first valid byte recved

ST program “st_init” :

```
if INIT=TRUE then

  (* Init *)
  Mode := 1 ;
  STEP := 0 ;
  T1 := T#500ms ;
  NUM := 0 ;
  tout := T#0s ;


  (* Open Com3 as baud=9600, char. size=8, no parity & stop bit=1 *)
  TMP:=comopen(3,9600,8,0,1);

end_if;
```

Do some init at the first scan cycle

ST program “R_W_COM” :

```
(* STEP:                                              *)
(*      0: no valid data coming                        *)
(*      1: first valid byte received                   *)
(*      2: receive '?' command                         *)
(*     10: check if other byte is '0' ~ '9'            *)
(*     21: receive complete command                    *)

num_com3 := 0 ; (* reset to 0 *)
if ComReady(3) then          <-- test if data coming from Com3
  num_com3 := Comary_R(3, 1) ;   <-- read all coming bytes to byte array 1
end_if;

(* if data coming, process it *)
index := 1 ;
while num_com3 > 0 do        <-- Process all coming bytes

  num_com3 := num_com3-1;
  byt := array_r(1,index) ;  <-- get current operating byte from array 1
  index := index + 1 ;

  case  STEP  of             STEP 0 : check if 1st byte valid or not
   0:
     case byt of   (* check 1st byte *)
       16#4D, 16#6D, 16#54, 16#74 :    (* 1st byte is valid,  M, m, T, or t *)
          STEP := 1 ;     (* for next STEP *)
          NUM := 1 ;      (* plus valid received byte number by 1 *)
          TSTART(tout);   (* start ticking tout *)
          CMD := byt ;   (* record command type *)
          TMP := Array_w(2, NUM, byt);   (* save 1st valid byte to byte array 2*)
     end_case;
   1:                        STEP 1 : check if 2nd byte valid or not
     case byt of   (* check 2nd byte *)
       16#3F :    (* 2nd byte is '?' *)
          STEP := 2 ;     (* for next STEP *)
          NUM := 2 ;      (* plus valid received byte number by 1 *)
          TMP := Array_w(2, NUM, byt);   (* save 2nd valid byte to byte array 2*)
     else
       if (CMD=16#4D or CMD=16#6D)  (* cmd is M, m *)
         and (byt >= 16#31) and (byt <= 16#33) then  (* '1' ~ '3' *)
          STEP := 10 ;    (* for next step *)
          NUM := 2 ;      (* plus valid received byte number by 1 *)
          TMP := Array_w(2, NUM, byt);   (* save 2nd valid byte to byte array 2*)
       elsif (CMD=16#54 or CMD=16#74)  (* cmd is T, t *)
         and (byt >= 16#30) and (byt <= 16#39) then  (* '0' ~ '9' *)
          STEP := 10 ;    (* for next step *)
          NUM := 2 ;      (* plus valid received byte number by 1 *)
```

```
      TMP := Array_w(2, NUM, byt);   (* save 2nd valid byte to byte array 2*)
    else
      STEP := 0 ;  (* not valid data, reset STEP to 0 *)
      TSTOP(tout) ;   (* stop ticking "tout" *)
      tout := T#0s ;  (* reset "tout" *)
      NUM := 0 ;      (* reset NUM *)
    end_if;
  end_case;
```

STEP 2 : after receive 2nd byte = '?' , check if 3rd byte is <CR>

```
 2:
  if byt=16#0D then  (* check 3rd byte is <CR> or not *)
    STEP := 21 ;  (* complete command is received *)
    (* send answer to Com3 *)
    case CMD of
      16#4D, 16#6D :    (* M or m *)
```

Receive valid "M?" command, reply "Mx", x = '1' ~ '3'

```
        TMP := ComWrite(3, 16#4D);           (* M *)
        TMP := ComWrite(3, Mode+16#30 );     (* Mode *)
        TMP := ComWrite(3, 16#0D );          (* <CR> *)
      16#54, 16#74 :    (* T or t *)
```

Receive valid "T?" command, reply "Txxxx", x = '0' ~ '9'

```
        TMP := ComWrite(3, 16#54);           (* T *)
        TMP := ComStr_w(3, MSG(ANA(T1))) ;    (* Timer value *)
        TMP := ComWrite(3, 16#0D );          (* <CR> *)
    end_case ;
  else
    STEP := 0 ;  (* not valid data, reset STEP to 0 *)
    TSTOP(tout) ;   (* stop ticking "tout" *)
    tout := T#0s ;  (* reset "tout" *)
    NUM := 0 ;      (* reset NUM *)
  end_if;
```

STEP 10 : check 3rd and other byte when command is "Mx" or "Txxxx"

```
 10:
  if (byt=16#0D) then      (* received <CR> *)
    STEP := 21 ;  (* complete command is received *)
    case CMD of
      16#4D, 16#6D :    (* M or m *)
        Mode := Array_r(2,2)-16#30;  (* Change Mode *)
```

Receive valid "Mx" command, chang Mode value and reply "OK" to PC

```
        (* send answer to Com3 *)
        TMP := ComStr_w(3, 'OK');
        TMP := ComWrite(3, 16#0D );          (* <CR> *)
      16#54, 16#74 :    (* T or t *)
```

Receive valid "Txxx" command, change T1 value and reply "OK" to PC

```
        (* get Period *)
        TMP_val := 0 ;
        for ii := 1 to NUM-1 do
          TMP_val :=  10*TMP_val + (Array_r(2,1+ii)-16#30) ;
        end_for ;
        if (TMP_val >= 50) and (TMP_val < 10000) then (* T1 must be in 50 ~ 9999 ms *)
          T1 := TMR(TMP_val) ;   (* Change T1 *)
          (* send answer to Com3 *)
```

```
            TMP := ComStr_w(3, 'OK');
            TMP := ComWrite(3, 16#0D );         (* <CR> *)
          end_if;
        end_case ;

      elsif (byt >= 16#30) and (byt <= 16#39) then  (* '0' ~ '9' *)

        STEP := 10 ;        (* for next step *)
        NUM := NUM+1 ;      (* plus valid received byte number by 1 *)
        TMP := Array_w(2, NUM, byt);   (* save other valid byte to byte array 2*)

        if NUM>5 then      (* command is too long, drop it *)
          STEP := 0 ;     (* reset STEP *)
          TSTOP(tout) ;   (* stop ticking "tout" *)
          tout := T#0s ;  (* reset "tout" *)
          NUM := 0 ;      (* reset NUM *)
          EXIT;           (* exit while loop *)
        end_if;

      else
        STEP := 0 ;  (* not valid data, reset STEP to 0 *)
        TSTOP(tout) ;   (* stop ticking "tout" *)
        tout := T#0s ;  (* reset "tout" *)
        NUM := 0 ;      (* reset NUM *)

      end_if;

  end_case ;

end_while;

(* Check timeout *)
if tout > T#5s then  (* if timeout *)
  STEP := 0 ;     (* reset STEP *)
  TSTOP(tout) ;   (* stop ticking "tout" *)
  tout := T#0s ;  (* reset "tout" *)
  NUM := 0 ;      (* reset NUM *)
end_if;

(* reset STEP to 0 *)
if STEP=21 then
  TSTOP(tout) ;   (* stop ticking "tout" *)
  tout := T#0s ;  (* reset "tout" *)
  NUM := 0 ;      (* reset NUM *)
  STEP := 0 ;
end_if;
```

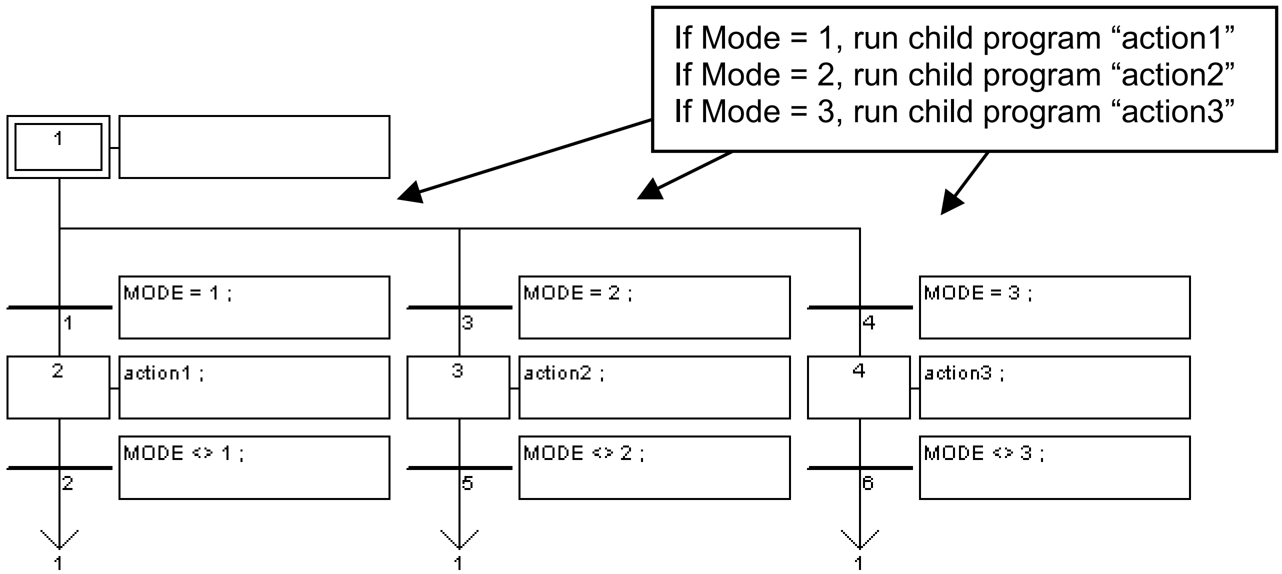
Receive '0' ~ '9', command is not completely received yet, process next byte

Check timeout, a valid complete command should be received in 5 seconds

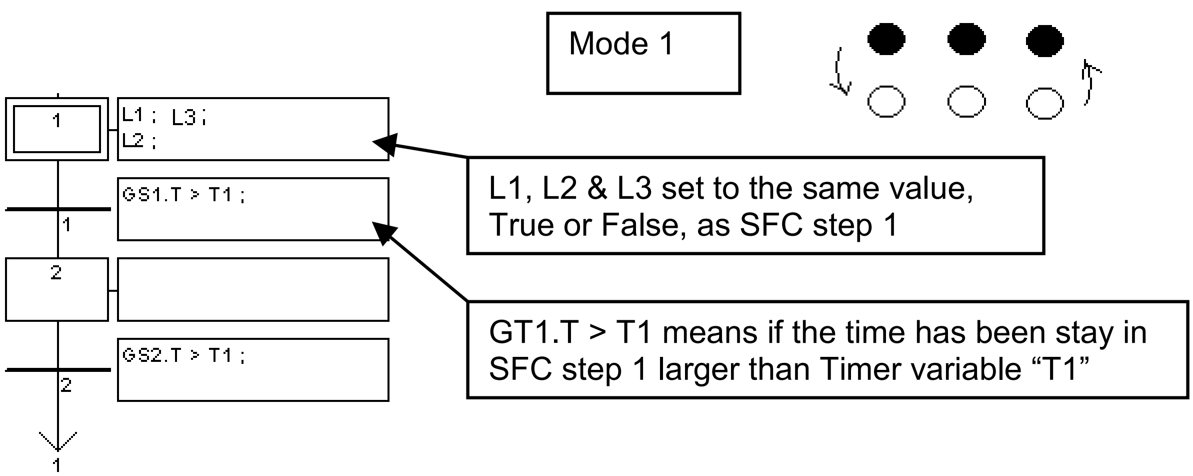
Valid command has been processed, reset to STEP 0

SFC program “Out” :

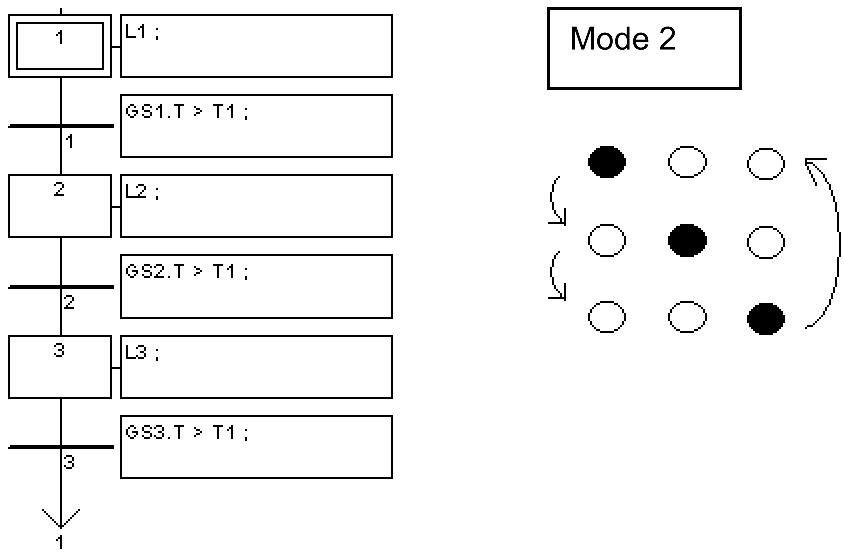
Each statement should end with a colon “;”



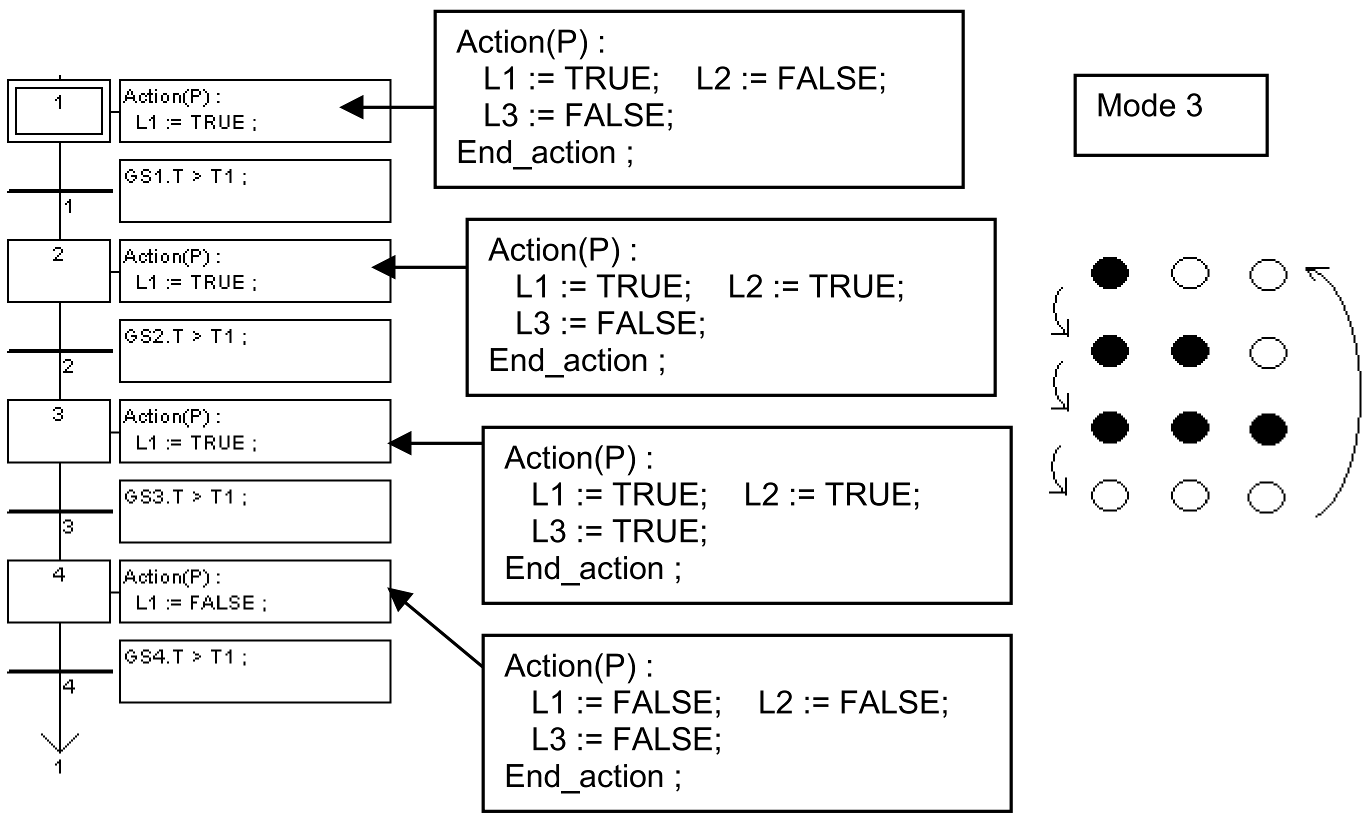
SFC child program “action1” :



SFC child program “action2” :



SFC child program “action3” :



ST program “end_init” :

```
if INIT=TRUE then
  INIT := FALSE;    (* end of first PLC scan *)
end_if;
```

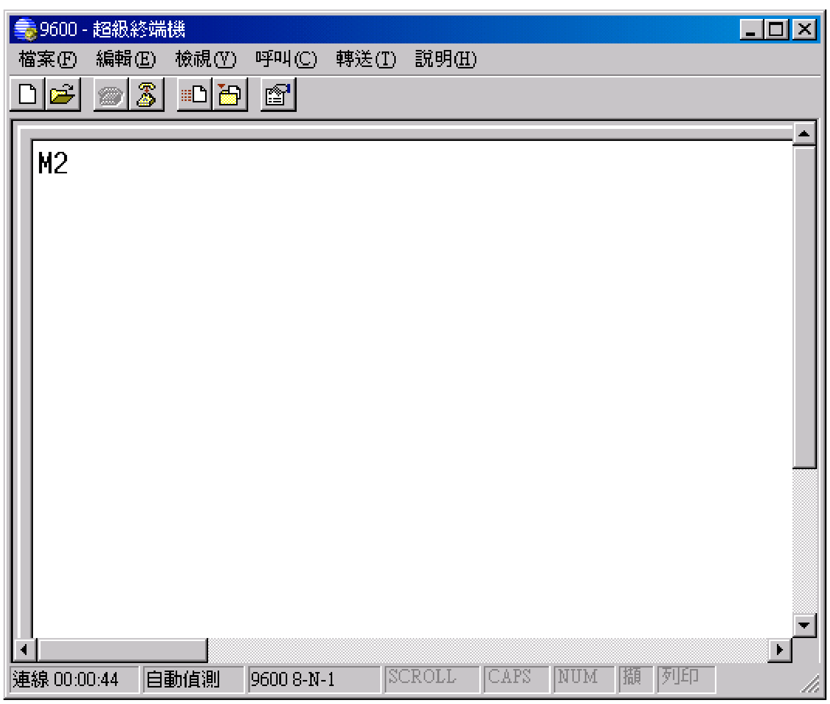
Set “INIT” to False, so that “INIT” is only TRUE at the first scan cycle since it is declared with the initial value - TRUE.

How to test ?

1 . Download Demo_33 to the controller.
2. Prepare a RS232 cable to connect Com3 of the controller to Com1 of your PC.
3. There is one ultilty named “ComTest.exe” located in the ICP DAS’s CD-ROM. Copy it to your PC. “\Napdos\ISaGRAF\some_utility\Comtest.exe” or you may obtain it from below site.
   ftp://ftp.icpdas.com/pub/cd/8000cd/napdos/isagraf/some_utility/
4. You may open a “Hyper Terminal” with Com1, 9600, N, 8, 1 and “No flow control” to type the following command to test

M2<CR> : change to mode 2
T?<CR> : request current period time
T200<CR> : change to 200ms
T1500<CR> : change to 1500ms
M?<CR> : request current mode

<CR> is the return char.



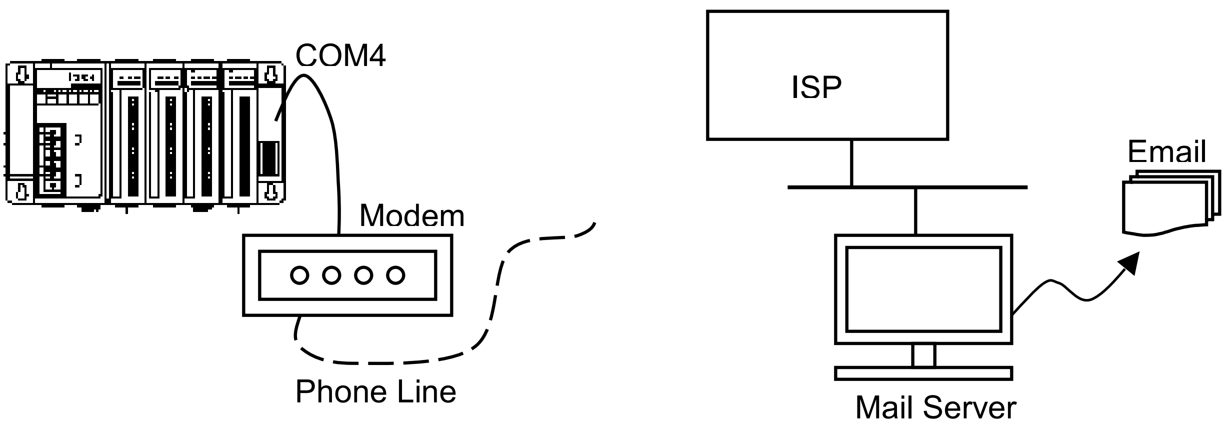
Chapter 12: Sending Emails

12.1: Introduction

COM4 of The I-8417/8817/8437/8837 supports full modem signals. It has embedded an email protocol only with the driver version of "**email_2.42**". It is a special driver version not the default released one. You have to refer to Appendix C to change your controller driver version if Email function is need. You can obtain the new released driver from:

http://www.icpdas.com/products/8000/isagraf.htm

To Send email from the controller, Com4 has to link to a modem. Com4 has exactly the same pin assignments as the Com1 (9-pin Dsub) of the PC. The operation figure is as below.

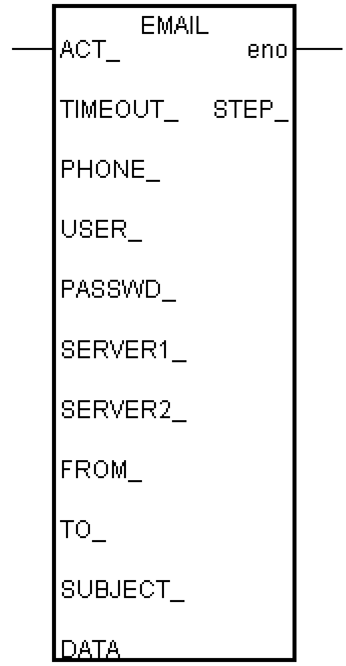


You have to register a User-name/Password from the local ISP(Internet Service Provider). And you have to get the ISP's phone No. and at least one mail-server's address near the local ISP. For example.

User Name :	David
Password :	A1234
ISP's Phone No. :	29020001
Mail server 1 :	mail.seed.net.tw
Mail server 2 :	mail.icpdas.com (not necessary)

12.2: Programming The “Email”

The “EMAIL” block is for sending email. This section provides an demo example to detail how to send an email to one receiver.



<u>Parameter description:</u>

(**Name**) (**Type**) : (**Description**)

ACT_ <boolean> :
if rising from false to true, start to send an email, and the return value - STEP_ will be changed. If no sending request occurs, the return value STEP_ will be 0 (0 means sleep)

TIMEOUT_ <integer> :
unit : seconds. The max time allowed to send an email after linking to mail server. Value should be between 50 ~ 120 (sec).

PHONE_ <message> :
ISP's phone No. For ex. '4123000' or '0,4123000' the ',' char. will delay 1 sec and then dial the rest No.

USER_ <message> :
Registerd user name from ISP. ex. 'Chun'

PASSWD_ <message> : Password of USER_ ex. 'abcd127'

SERVER1_ <message> : Mail server 1. ex. 'ms9.hinet.net'

SERVER2_ <message> :
Mail server 2. ex. 'mail.icpdas.com' . If only one mail server found, please set SERVER2 as same as SERVER1

FROM_ <message> : email address of sender. ex. 'baby@icpdas.com'

TO_ <message> : email address of receiver. ex. 'father@icpdas.com'

SUBJECT_ <message> : subject of email. ex. 'Hi !'

DATA_ <message> : email message. ex. 'Dear Chun, Hello !'

<u>return:</u>

STEP_ <Integer> :

0 : sleep

21 : mail successfully !

less than 0 , error happens

-1 : Com4 not ready
-2 : modem not ready
-3 : ISP doesn't pick up the phone
-4 : ISP request to terminate
-5 : Timeout happen
-6 : Mail server refuse to send mail
-7 : Can not link to mail server 1 & 2
-8 : Can't get IP address of mail server 1 & 2

others : reserved

Note:

1. After an email is successfully sent, if no more sending request occurs in 8 seconds, the controller will disconnect the connection from the connected ISP and then hang off the phone .

2. If sending request occurs in 8 second After an email is successfully sent, and then again, the max number of emails can be sent in one phone connection is 10. The other more emails should be sent in another phone connection (In other words, re-dial).

3. If dial fail, for ex. the target phone No. is busy. The controller will dial again about one minute later. The max re-dial number is 3 for each sending request.

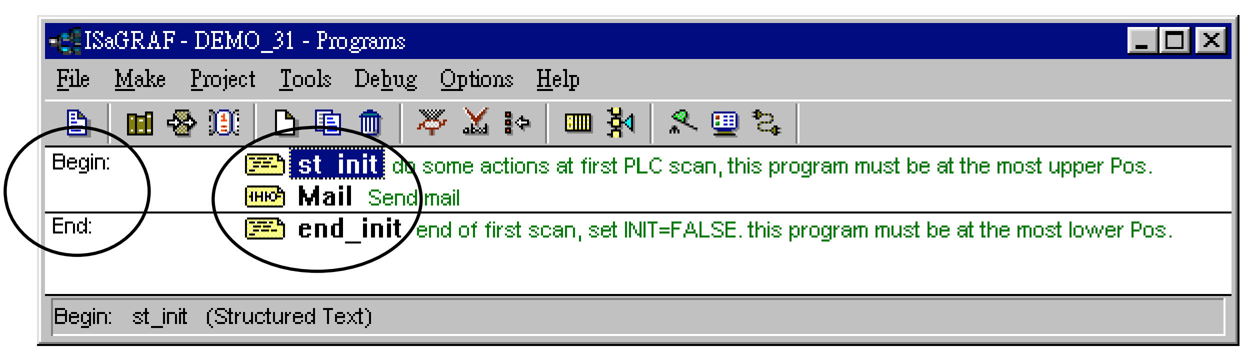
An Email sample: Please refer to section 9.5 to install the demo project into your ISaGRAF. The project file “demo31.pia” & “demo32.pia” can be found at CD-ROM: \napdos\isagraf\8000\demo\ or ftp.icpdas.com/pub/cd/8000cd/napdos/isagraf/8000/demo

Variables declared in the sample:

Name	**Type**	**Attribute**	**Description**
K1	Boolean	**Input**	Pushbutton 1, **Push it to triger the “Email" block**
INIT	Boolean	**Internal**	**initial value at “TRUE”**. TRUE means 1^{st} scan cycle
STEP	Integer	**Internal**	Return value of the “Email” block
PHONE	Message	**Internal**	Phone No. of ISP
USER	Message	**Internal**	Registered User Name from ISP
PASSWD	Message	**Internal**	Registered Password from ISP
SERVER1	Message	**Internal**	Address of mail server 1
SERVER2	Message	**Internal**	Address of mail server 2
MAIL_FROM	Message	**Internal**	Mail address of sender
MAIL_TO	Message	**Internal**	Mail address of receiver
SUBJECT	Message	**Internal**	Subject of the email
MAIL_DATA	Message	**Internal**	Content of the email

Project architecture:

st_init : a ST program to do some initial actions when the project is just beginning
Mail : a LD program to send email
End_init : a ST program to indicate the first scan cycle

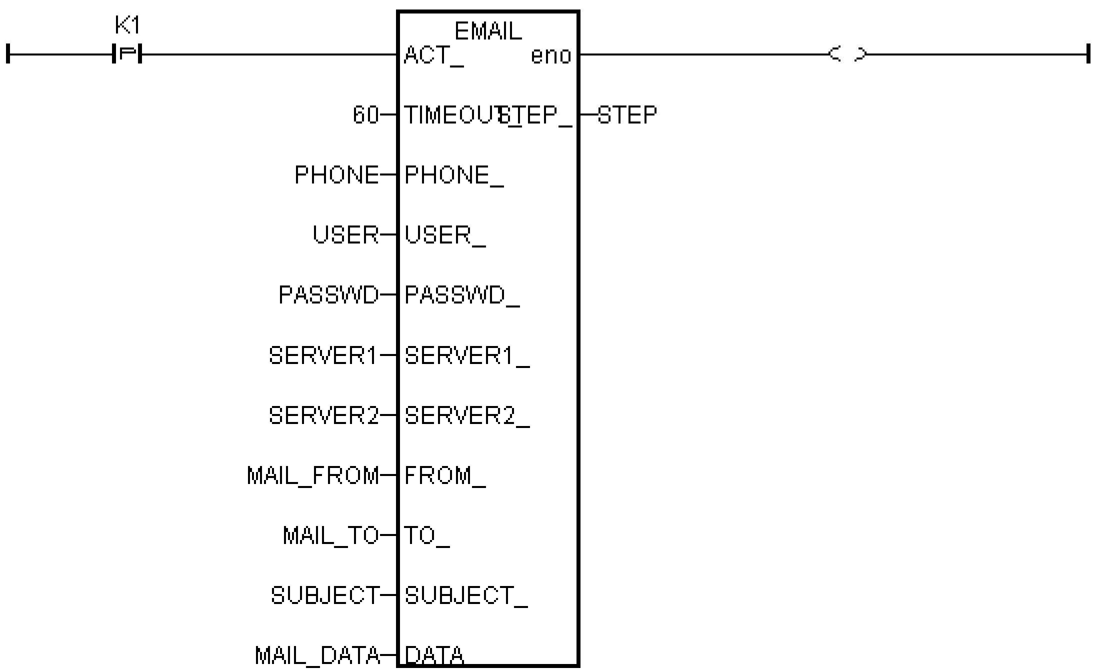


ST program - “st_init” :

Please give your correct data

```
(* first PLC scan, init the message variable *)

if INIT=TRUE then
      PHONE      := '12345678' ;        (* ISP's phone No. Please given your No. *)
      USER       := 'David' ; (* Registerd user name from ISP. given yours *)
      PASSWD     := 'abcdef' ;          (* Password. Please given yours *)
      SERVER1    := 'seed.net.tw' ;     (* Mail server 1. Please given yours *)
      SERVER2    := 'mail.seed.net.tw' ;    (* Mail server 2. Please given yours *)
      MAIL_FROM  :=   'baby@icpdas.com' ;       (* Sender. Please given yours *)
      MAIL_TO    := 'father@icpdas.com' ; (* Receiver. Please given yours *)
      SUBJECT    := 'Hello !' ;         (* Email subject *)
      MAIL_DATA  :=   'Dad, I am out !' ;        (* Email data *)
end_if;
```

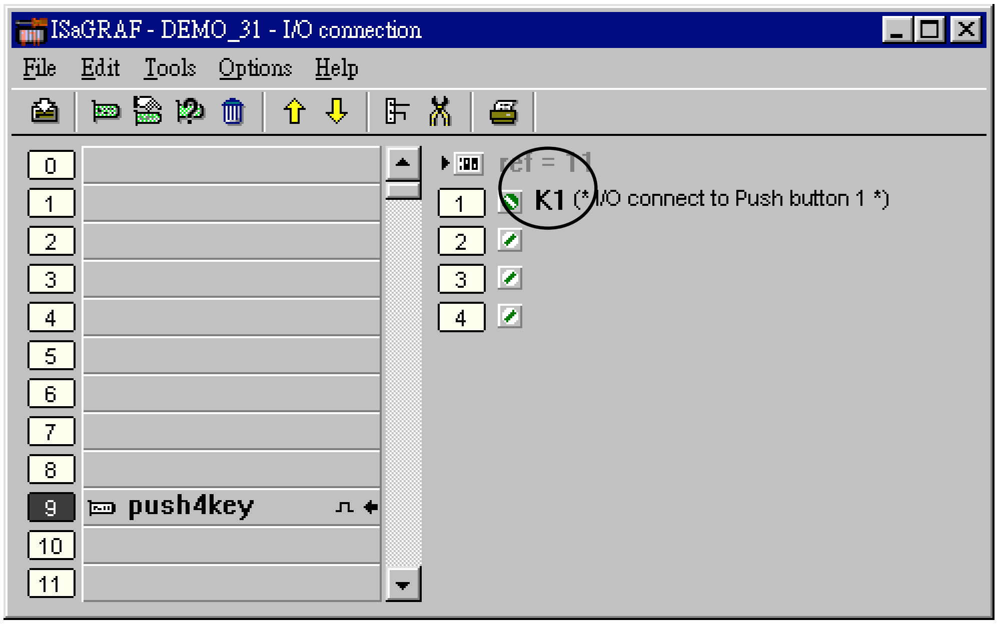
LD program – “mail” :



ST program – “end_init” :

```
(* NOTE: INIT should be declared with a initial value = TRUE in the "dictionary" window *)

if INIT=TRUE then
   INIT := FALSE ;    (* end of first PLC scan *)
end_if;
```

I/O connection:



Projection Operation Actions:

After compiling the project and download it to one I-8417/ 8817/ 8437/ 8837 controller, push the first pushbutton of the front panel. You will see the modem dialling and if everything is Ok, the email will be sent. See the return value of the “Email” block. (0 means no triggering, 21 means Ok. Less than 0 means something wrong).

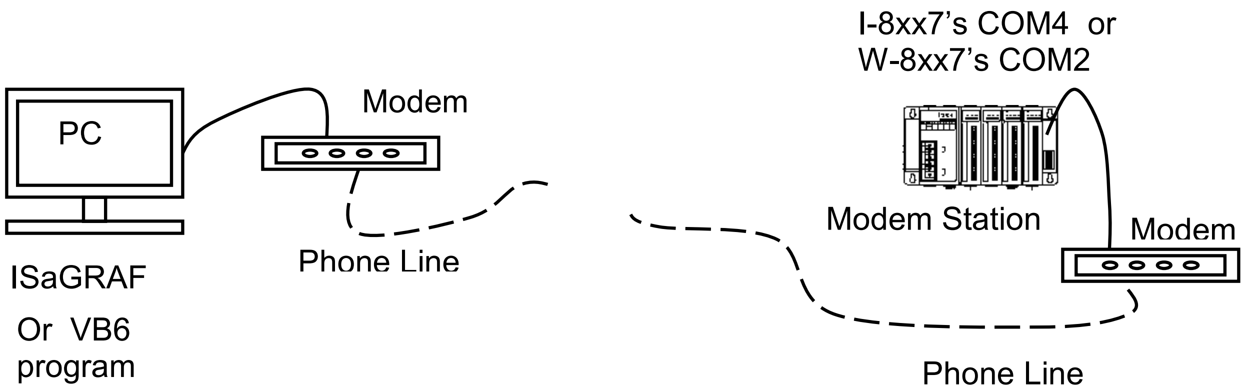
Chapter 13: Remotely Download Via Modem_Link

13.1: Introduction

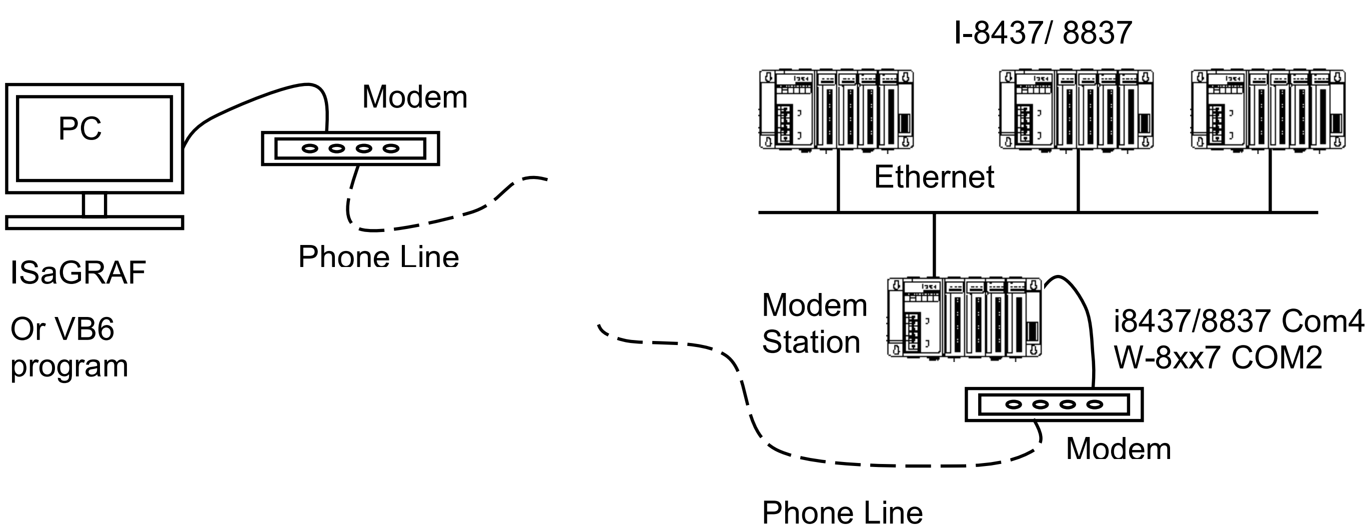
COM4 of The I-8417/8817/8437/8837 & COM2 of the W-8037/8337/8737 supports full modem signals. It has embedded the Modem_Link protocol for remotely download and monitoring since the I-8xx7 driver version of 2.14 & W-8xx7 driver version of 3.10. Please refer to Appendix C to make sure your I-8xx7 controller driver version is the same or higher. You can obtain the new released driver from:

http://www.icpdas.com/products/8000/isagraf.htm

To Remotely download and monitor program via the Modem_Link, I-8xx7's Com4 & W-8xx7's Com2 has to link to a modem. They have exactly the same pin assignments as the Com1 (9-pin Dsub) of the PC.



We name the controller as "**Modem Station**" since it will pick up the phone call coming from the remote PC running ISaGRAF. If the controller is either I-8437 or I-8837 (Ethernet controller), The configuration can be extened to link many controllers together. Therefore, the PC running ISaGRAF can remotely download to anyone of them through the modem and the Modem station.



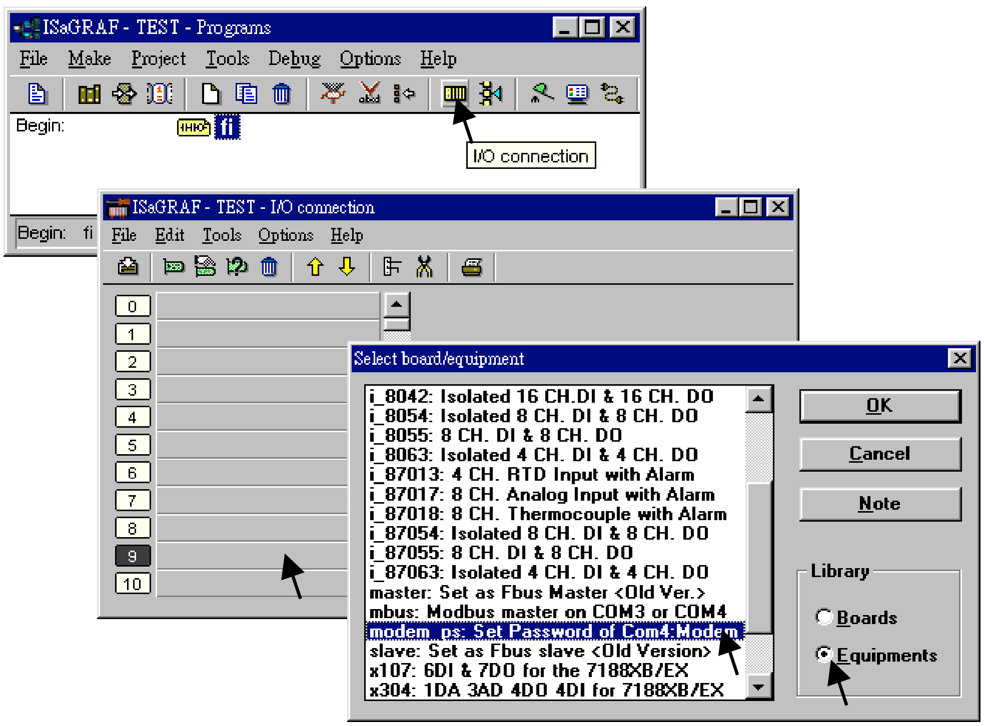
Note: W-8xx7's COM2 is Modbus RTU port by default, please disable it if using as "modem_link" port. Please refer to W-8xx7's "Getting Started" Manual.

13.2: Download Program Via Modem_Link

Warnning:
Do not download a project which uses I-8xx7's Com4 & W-8xx7's COM2 to do other things to the "Modem station" controller. For ex, do not connect "Bus7000" & "Mbus" with port_no = 4 (for I-8xx7) & port_no=2(for W-8xx7). And do not use "Comopen" to open Com4(for I-8xx7) & Com2(for W-8xx7). It will disable "Modem_Link" if you use them for other purpose. That means, you can not remotely connect to it.

Note: W-8xx7's COM2 is Modbus RTU port by default, please disable it if using as "modem_link" port. Please refer to W-8xx7's "Getting Started" Manual.

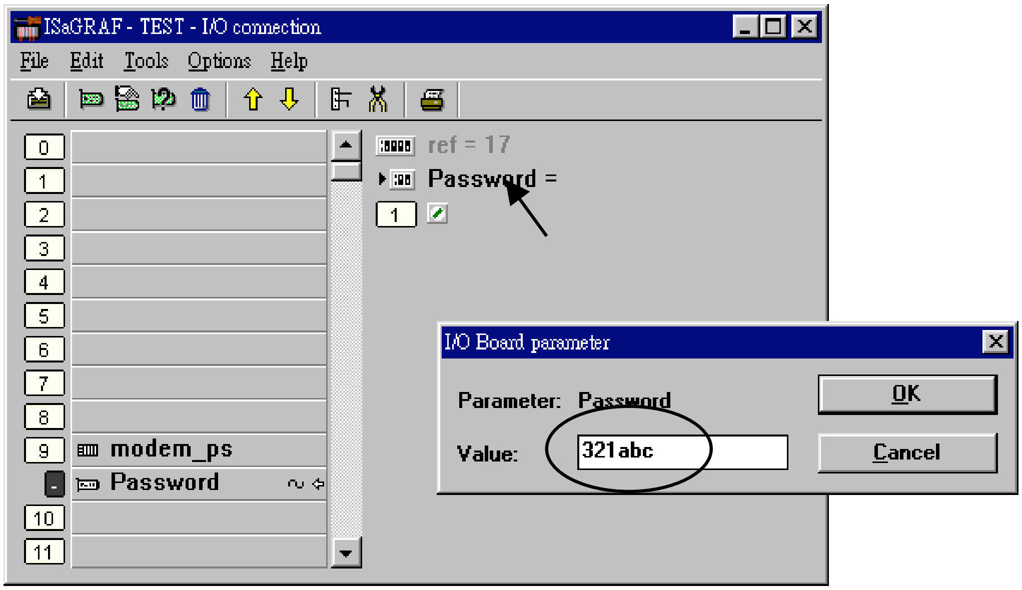
The first thing is to add a "modem password" to your ISaGRAF program of the "Modem station" controller for security. To do it, click on one empty slot No. from the I/O connection window. Then connect "Modem_PS" on the slot.



Then you got the window similar as below. Type in your prefered password for the "Modem station" controller. The password can contain up to 12 characters & can't use character " and '. Then re-compile it and download it to the "Modem station" controller.

Note:
User can write Visual Basic program to acess to the I-8417/8817/8437/8837 & W-8xx7 via Modem. Please download VB6 demo source code at

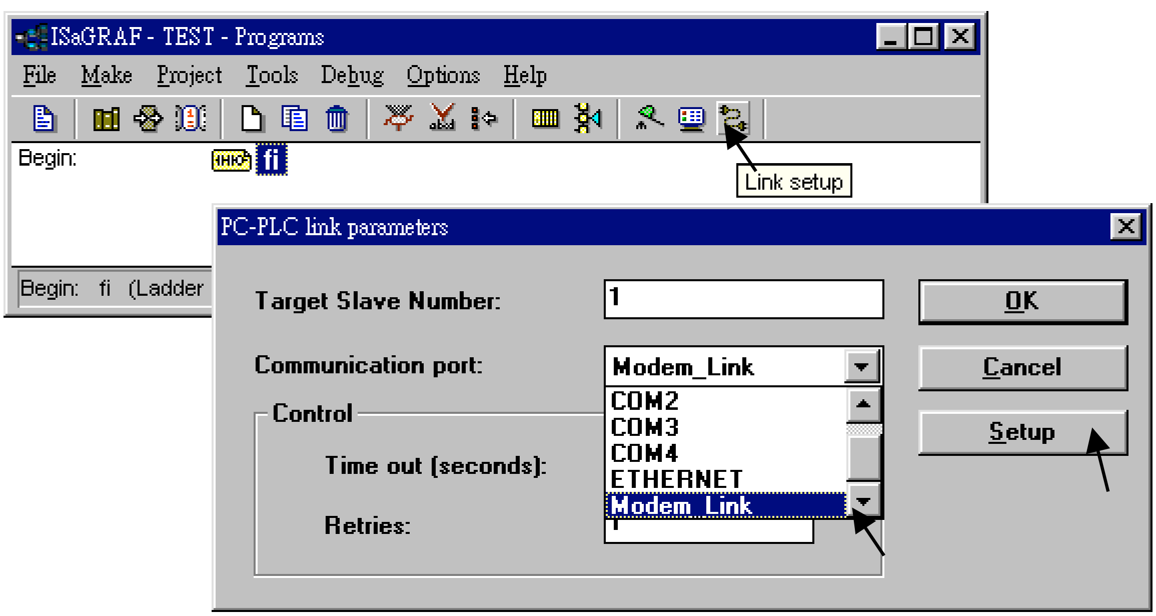
http://www.icpdas.com/products/8000/i-8417.htm or
ftp://ftp.icpdas.com/pub/cd/8000cd/napdos/isagraf/vb_demo/ or
CD-ROM:\napdos\isagraf\vb_demo\



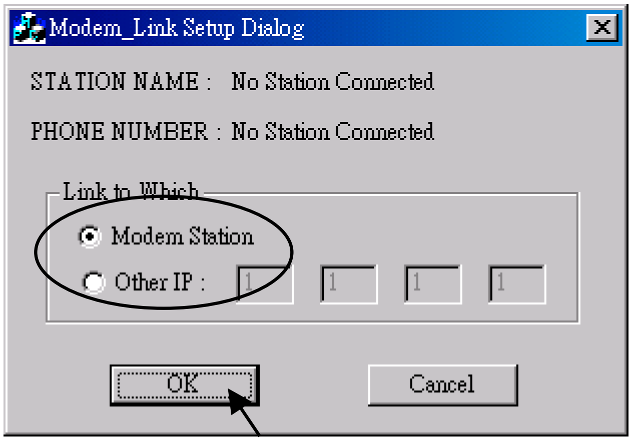
Very Important:
If you don't assign the Modem password to the "Modem station" controller, anyone who has the phone No. of your "Modem station" controller can link to it to do anything. Be very careful.

Now we are going to download and monitor the program of faraway controllers.

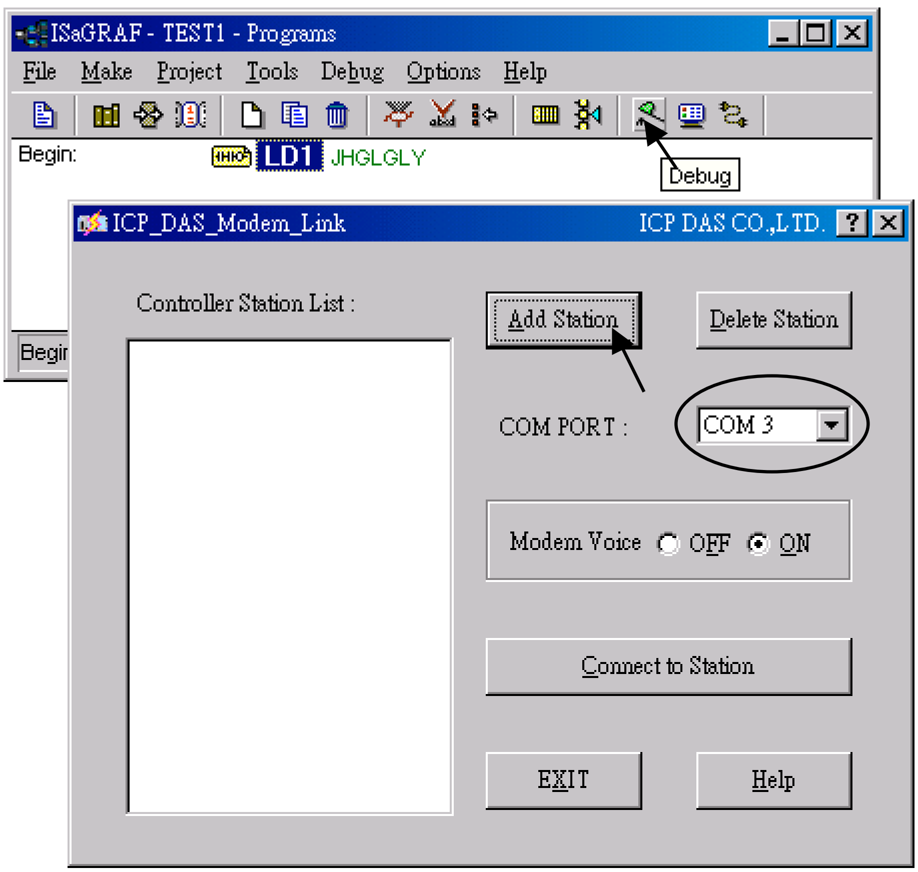
Click on "Link setup", select "Modem_Link", and then click on "Setup"



For windows NT, 2000 & XP users:
If you are going to connect the “Modem station” controller, check “Modem station”, otherwise check “Other IP”. “Other IP” means the target controller is not connect to a modem however connect to the “Modem station” controller via an ethernet cable, the IP address has to be assigned.

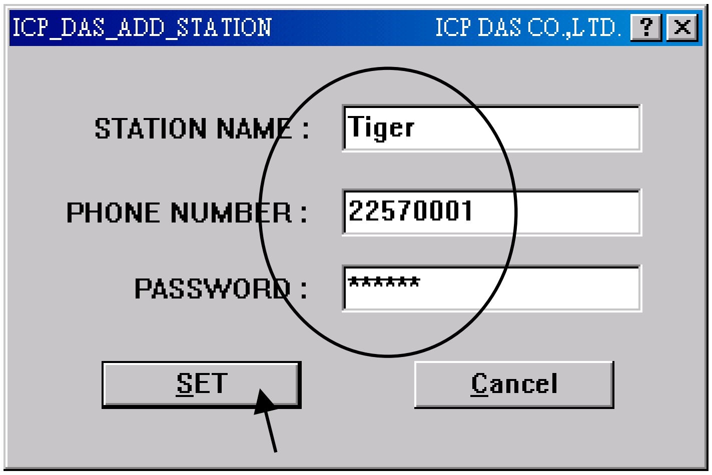


Then click on “debug”. Select the correct Com port of your PC which will dial the modem. And then click on “Add Station” to add a station if you have none.

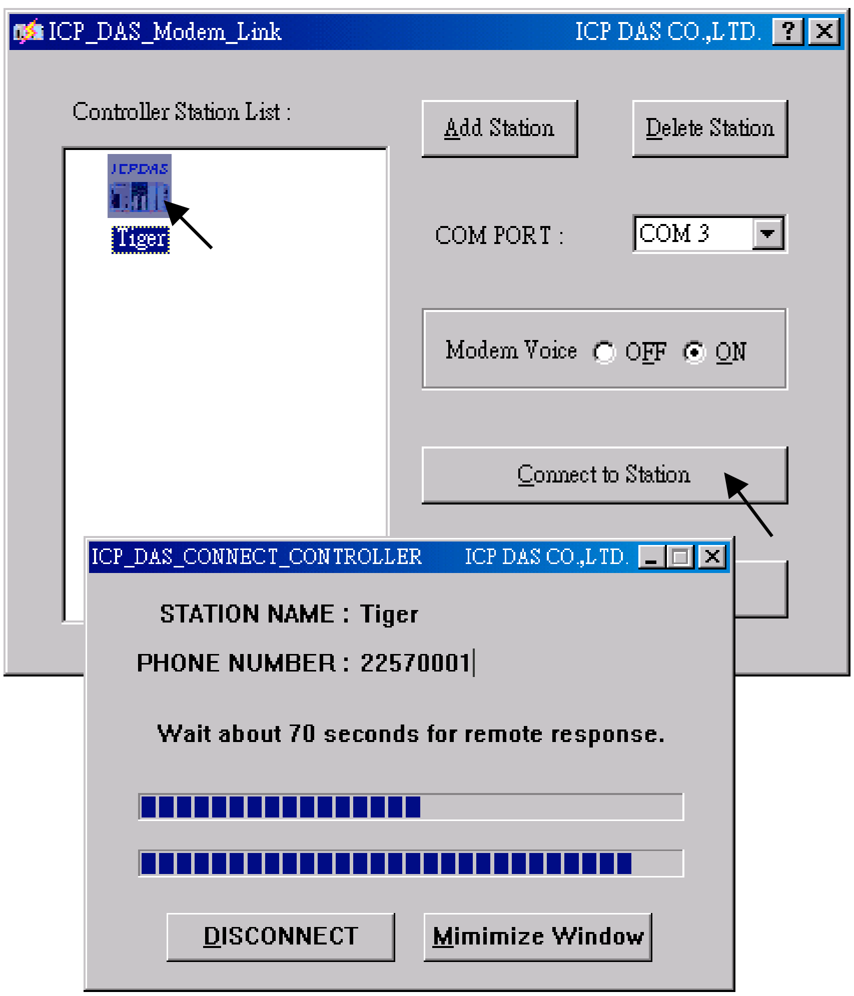


Then you will see the below window. Given a name for this new station and the target phone No. If you add a “,” character inside the phone No. It will wait one second and then dial the rest No.

For ex. Given No. as “9,,22570001” will dial “9” first, then wait 2 seconds and then dial “22570001”. The password must set to the same password of the “modem station” controller.

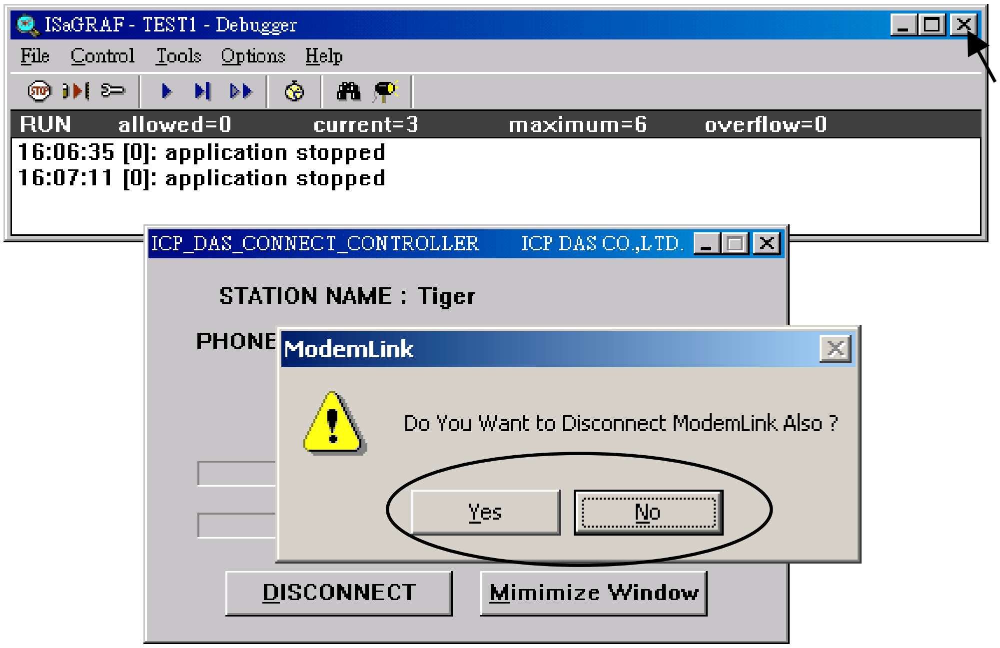


Click on the station you would like to connect first and then click on “Connect to Station” to command the modem dialing to the faraway controller.

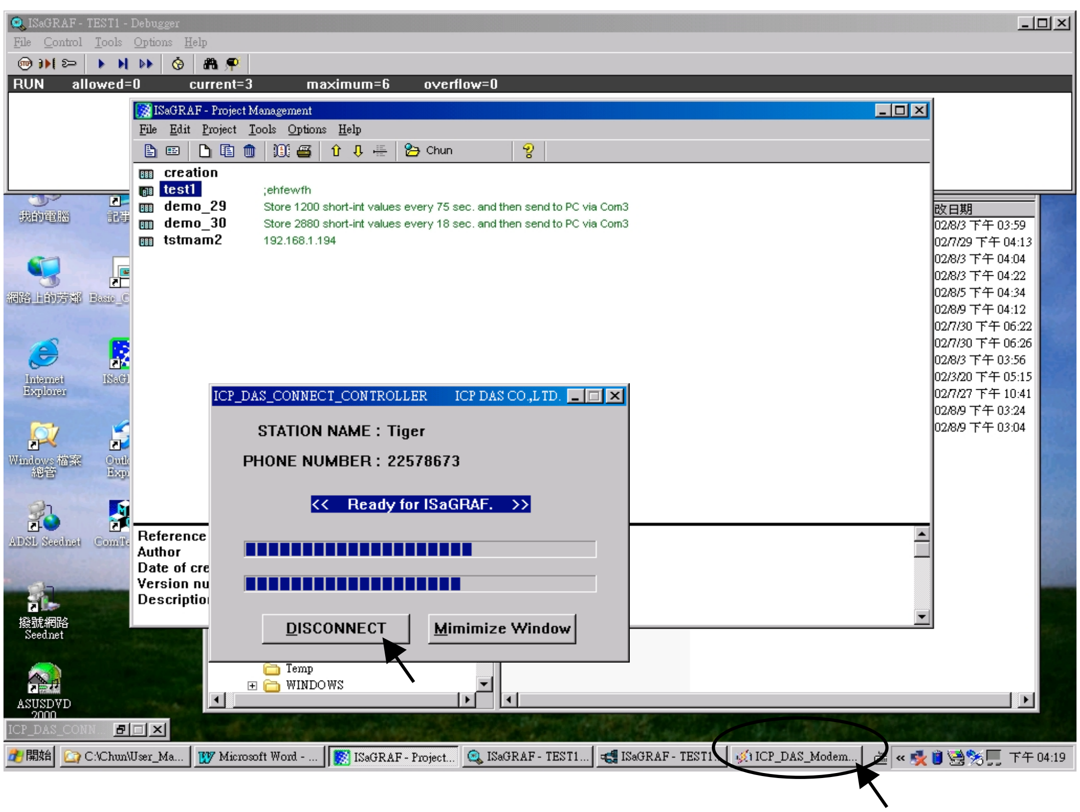


After the connection is Ok. You can download, monitor and change the variable value just like you did when the controller is near beside you.

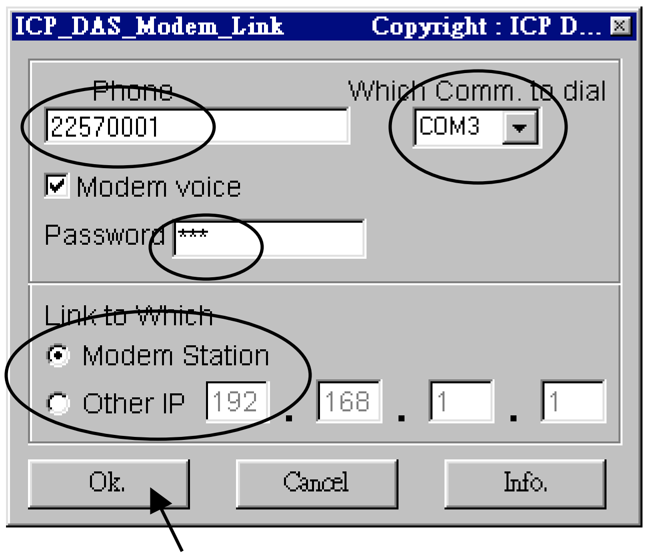
To disconnect from the target controller, close the " … Debugger" window. Then you can choose "No" to keep the phone connected, or "Yes " to hang off phone.
If you choose to keep the phone connected, you can open another ISaGRAF project to directly connect to another faraway target. The modem won't dial again.



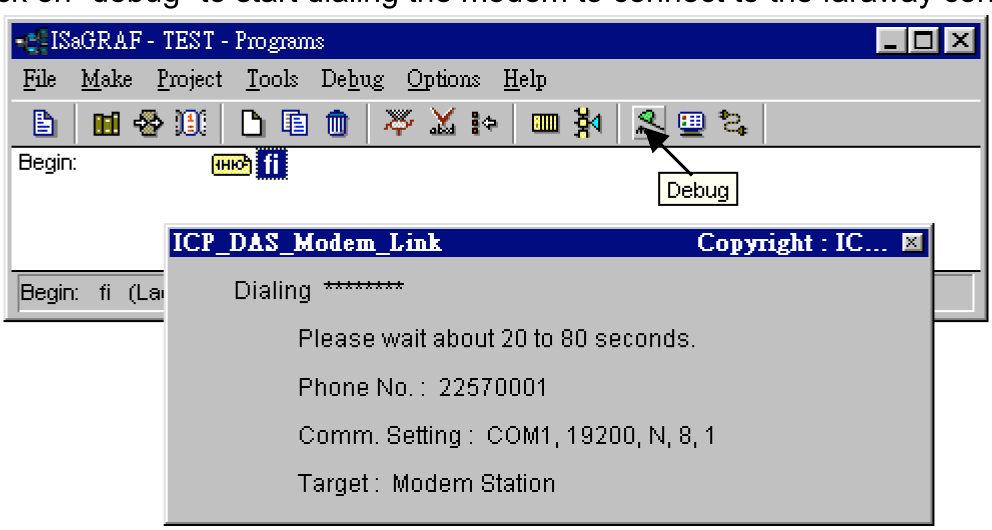
However, keep in mind, remember to disconnect the modem_link when you finish your work, don't waste the money to the telecom company.



For windows 95 & 98 users:
Given the correct target phone No. and the correct Com port of your PC which will dial the modem. If you add a "," character indise the phone No. It will wait one second and then dial the rest No. For ex. Given No. as "9,,22570001" will dial "9" first, then wait 2 seconds and then dial "22570001". The password must set to the same password of the "modem station" controller. If you are going to connect the "Modem station" controller, check "Modem station", otherwise check "Other IP". "Other IP" means the target controller is not connect to a modem however connect to the "Modem station" controller via an ethernet cable, the IP address has to assign.



Then click on "debug" to start dialing the modem to connect to the faraway controller.



After the connection is Ok., you can download a new program, monitor the variable status just like you did when the controller is near beside you.

When you close the " … Debugger" window, the PC will command the modem to hang off the phone and disconnect with the faraway controller.

Note:
The Modem_Link software installed on windows 95 & 98 doesn’t support “keep the phone connected” function. That means each time you close the “ … Debugger” window, the phone will be hanged off too. So next time when click on “debug”, you will see the modem dialing again to conect to the faraway controller.
For Windows NT, 2000 and XP users, the modem will not dial if you keep the phone connected.

Chapter 14: Spotlight : Simple HMI

Spotlight is a simple HMI coming with ISaGRAF which allows user to build **Boolean Icon, Bar Graph, Trend Curve, Value Text, Bitmap Picture** to make application more friendly.

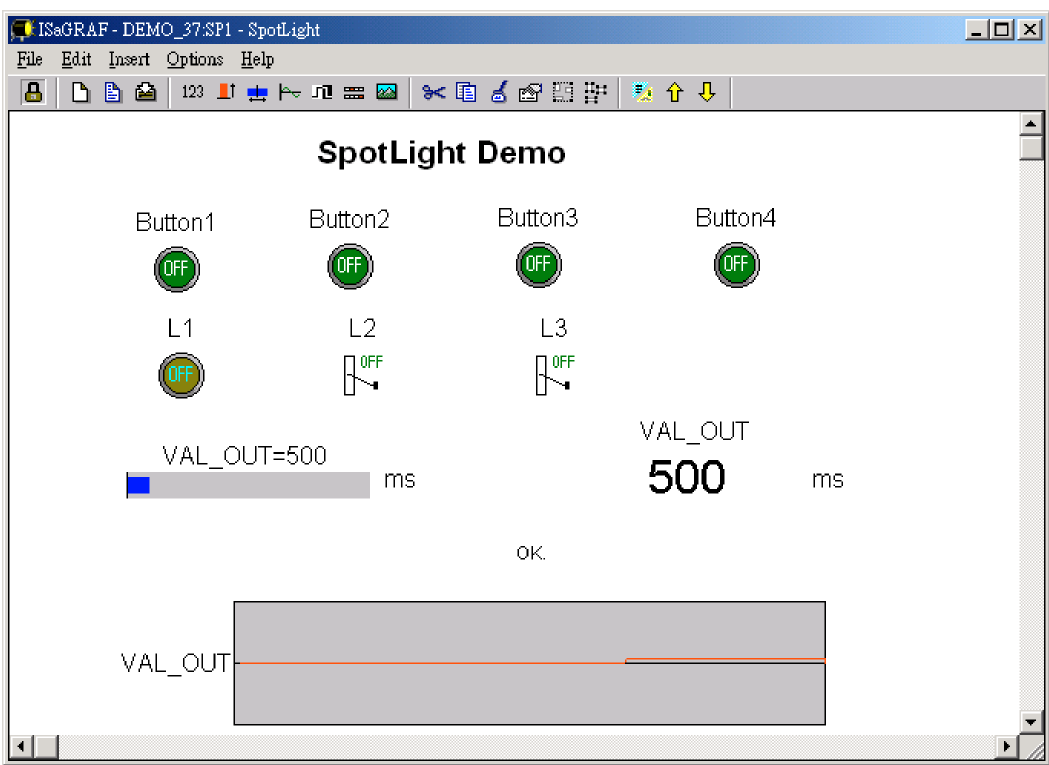
14.1 A Spotlight Example:

This Demo example can be restored from the ICP DAS's I-8000 CD-ROM - "demo_37" (For I-8xx7). Please refer to Chapter 11 to restore it.

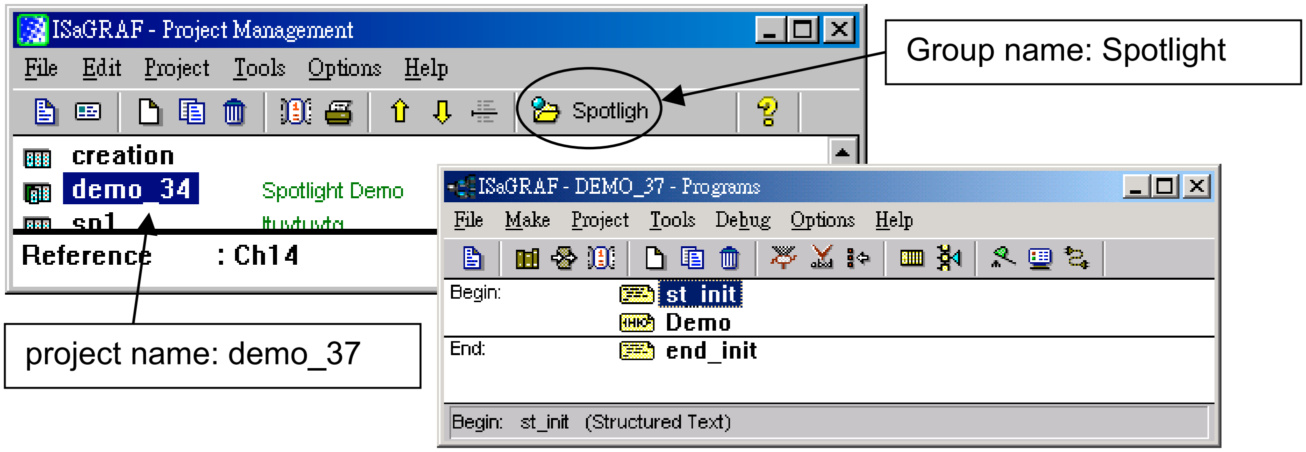
Variables used In the example:

Name	**Type**	**Attribute**	**Description**
INIT	Boolean	**Internal**	Only = TRUE at the 1st scan cycle, **INIT value is TRUE**
L1	Boolean	**Output**	Output 1, connect to Ch1 of "show3led"
L2	Boolean	**Output**	Output 2, connect to Ch2 of "show3led"
L3	Boolean	**Output**	Output 3, connect to Ch3 of "show3led"
Button1	Boolean	**Inpput**	Input 1, connect to Ch1 of "push4key"
Button2	Boolean	**Inpput**	Input 2, connect to Ch2 of "push4key"
Button3	Boolean	**Inpput**	Input 3, connect to Ch3 of "push4key"
Button4	Boolean	**Inpput**	Input 4, connect to Ch4 of "push4key"
VAL_OUT	Integer	**Internal**	to set blinking period, **initial value is set at 500 (unit:ms)**
T1	Timer	**Internal**	Time Period of blinking
MSG1	Message	**Internal**	Status report, please set its Maxinum Length to 48

HMI screen outline:



Project architecture:



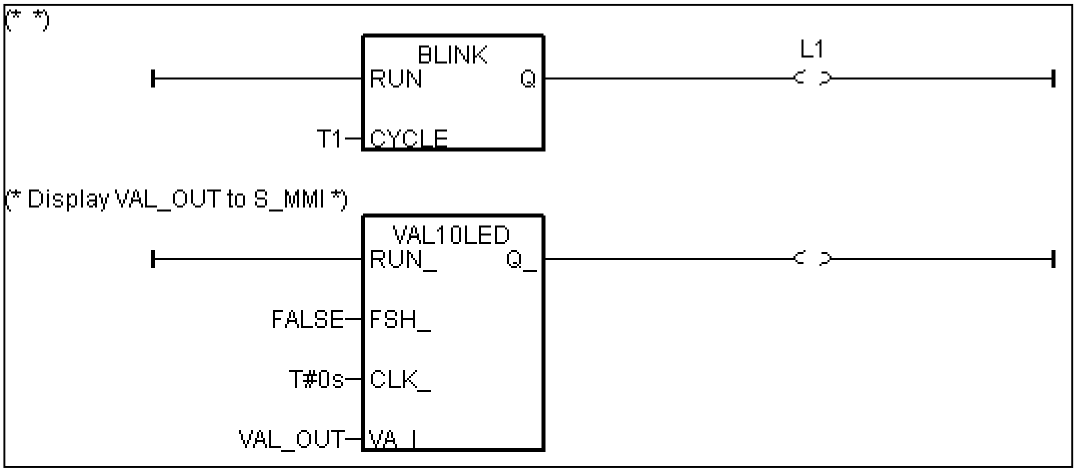
ST Program "st_init" in the "Begin" area :

```
(* Do some init action *)
if  INIT=TRUE  then
  T1 := TMR(VAL_OUT);   (* Convert integer:VAL_OUT to Timer:T1 in ms *)
  MSG1:='OK.';
  OLD_VAL_OUT := VAL_OUT;  (* init OLD value *)
end_if;

(* if set a new value to VAL_OUT *)
if  VAL_OUT <> OLD_VAL_OUT  then

  (* VAL_OUT is acceptable *)
  if  (VAL_OUT>=200) & (VAL_OUT<=5000)  then
    T1 := TMR(VAL_OUT);   (* Convert integer:VAL_OUT to Timer:T1 in ms *)
    MSG1:='OK.';
  else   (* VAL_OUT out of range *)
    MSG1:='VAL_OUT should be between 200 and 5000 .';
  end_if;

  OLD_VAL_OUT := VAL_OUT;  (* update OLD value *)

end_if;
```

ST Program "end_init" in the "End" area :

```
INIT := FALSE ;
```

LD Program “Demo” in the “Begin” area:



Operations :

The status of four push buttons will be displayed on the HMI screen
The first output will be blinking with the period defined by “VAL_OUT” in ms
Value of “VAL_OUT” can be modified from the HMI screen
The second and third output “L2” & “L3” can be controlled by the HMI screen.
The Value of “VAL_OUT” will also be displayed on the front panel of the controller.

Steps to build a Spotlight: HMI screen:

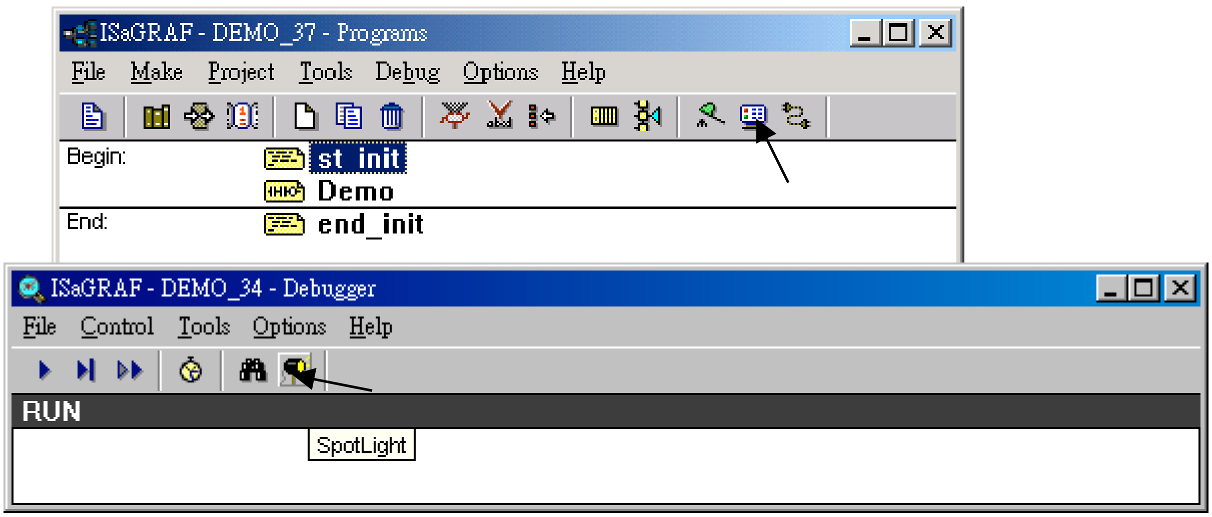
Complete this Demo project as described above.
After you finish it. Compile it to make sure there is no error.

Copy all files inside “ICO” folder to the associate directory of your project.
The “ICO” folder contains some boolean icon files already bulit by ICP DAS. They can be found from the ICP DAS’s CD-ROM : \napdos\isagraf\ICO\

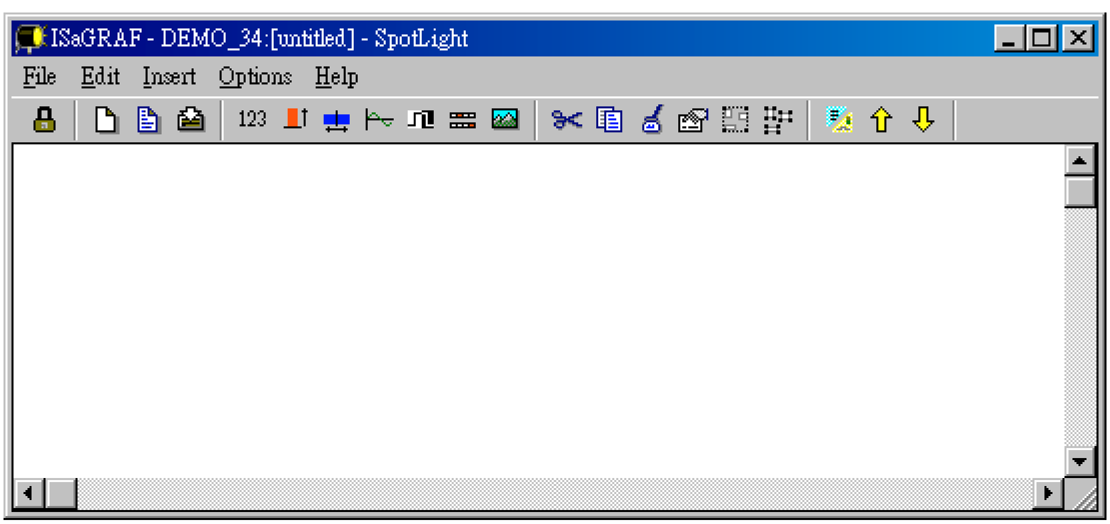
For example, this demo project is inside group “spotligh” and the project name is “demo_37”, then copy CD-ROM: \napdos\isagraf\ICO*.* to c:\isawin\spotligh\demo_37\

If the “ICO” folder is not found in your CD-ROM. Please download it from the below site.
ftp://ftp.icpdas.com/pub/cd/8000cd/napdos/isagraf/

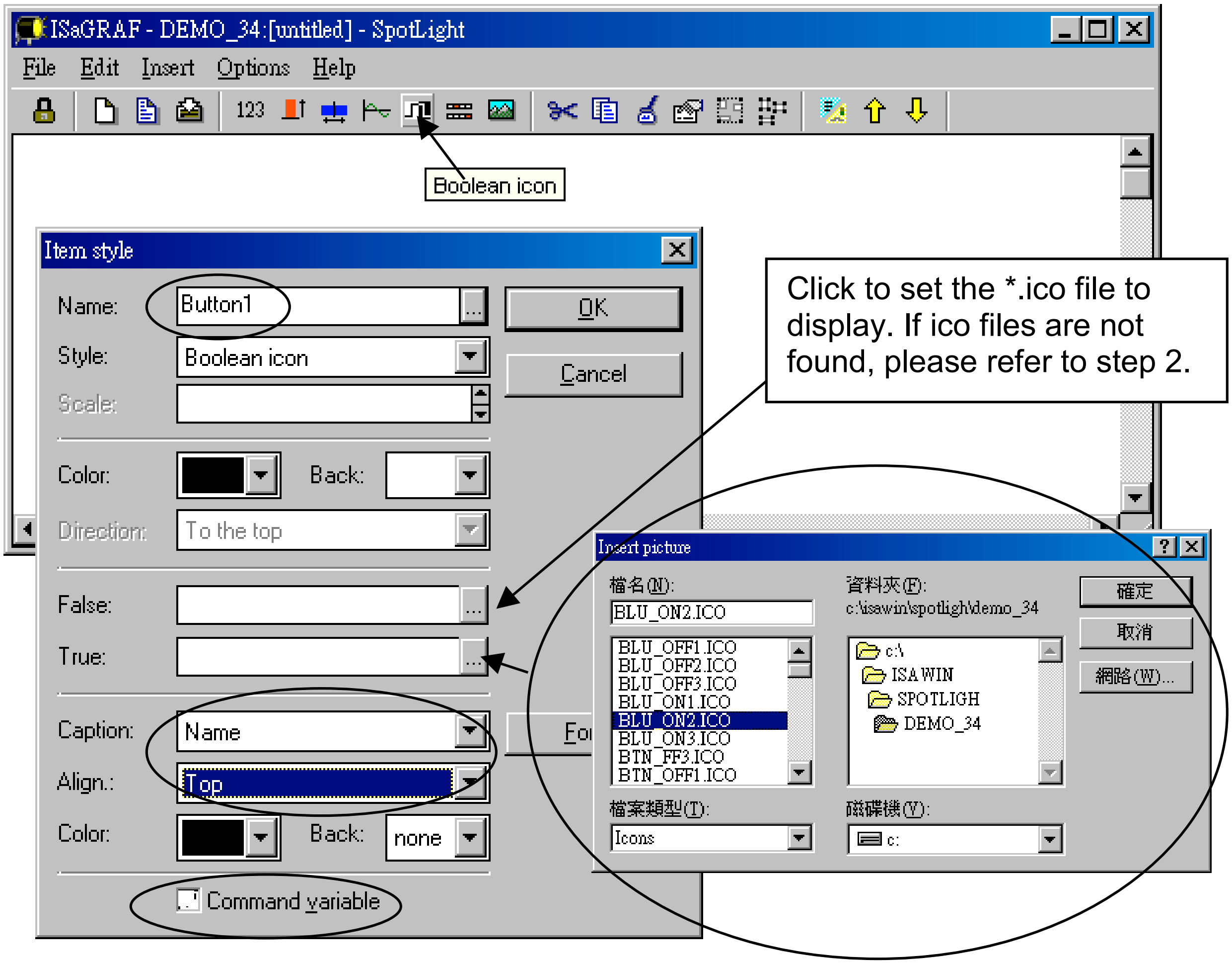
Get into the Spotlight editor.
Click on “Simulate”, then click on “Spotlight” to open spotlight editor.



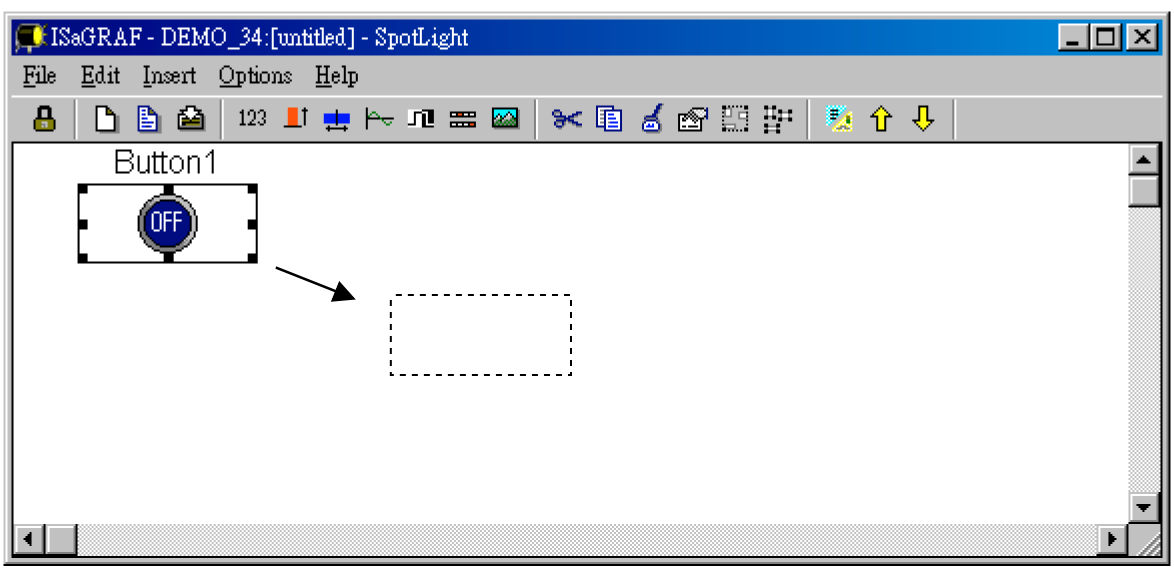
A “SpotLight” window will appear as below.



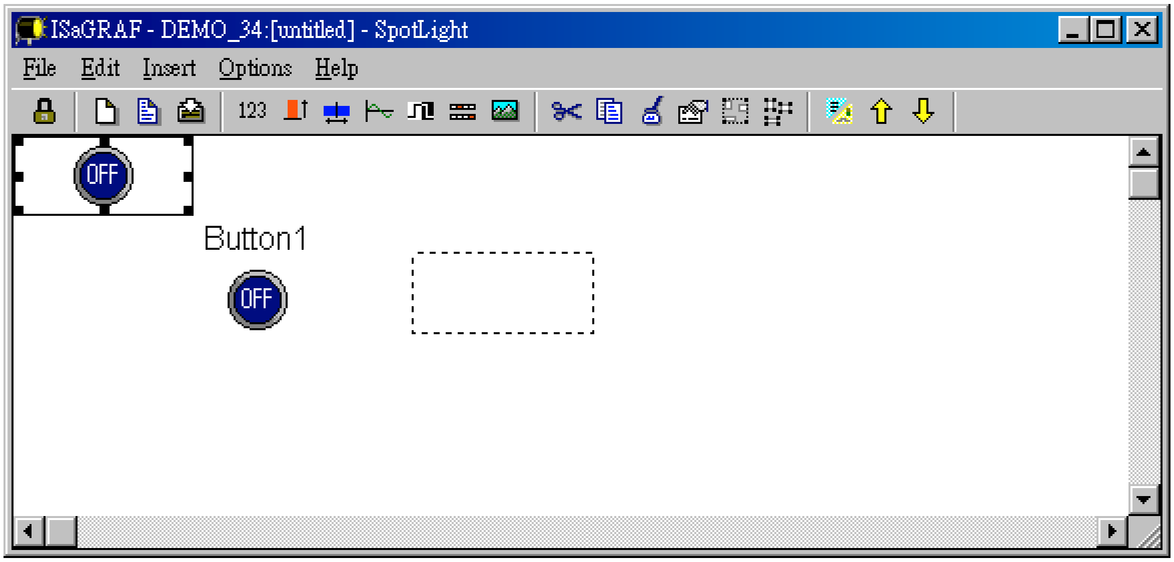
Add “boolean Icons”
Click on “Boolean icon”, then set the associated Name as “Button1”, Caption as “Name”, Align as “Top” and then set the prefered *.ico file to display with “FALSE” and “TRUE”, and un-check “Command variable”.



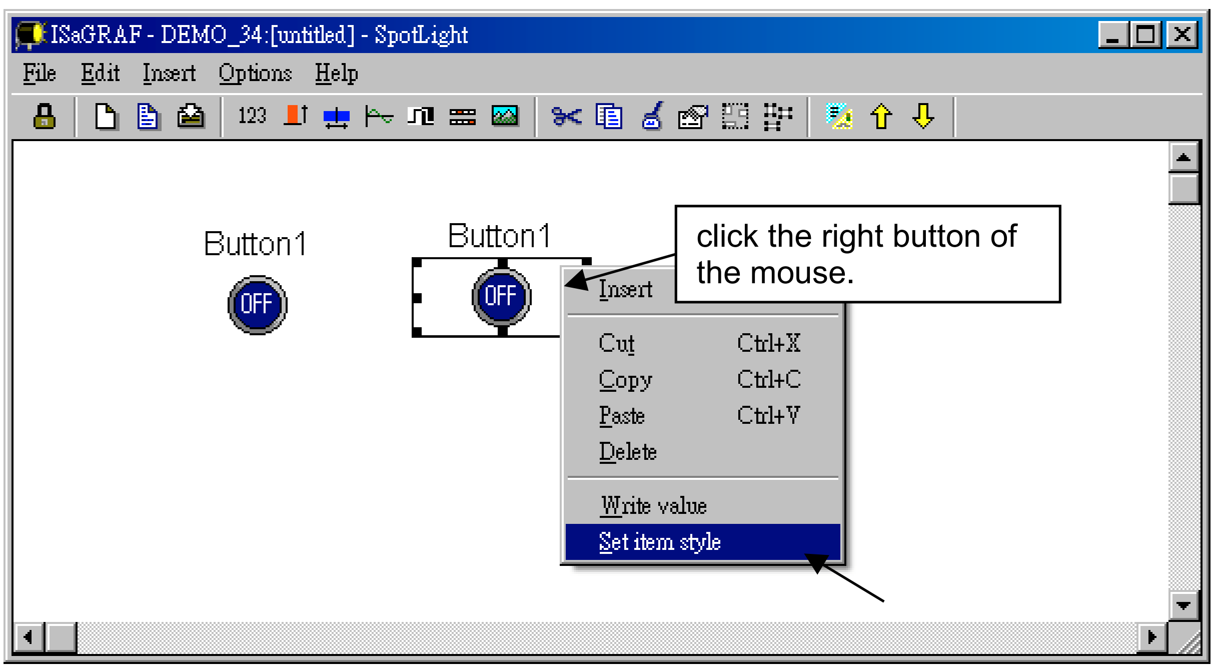
Then drag the boolean icon to appropriate place.



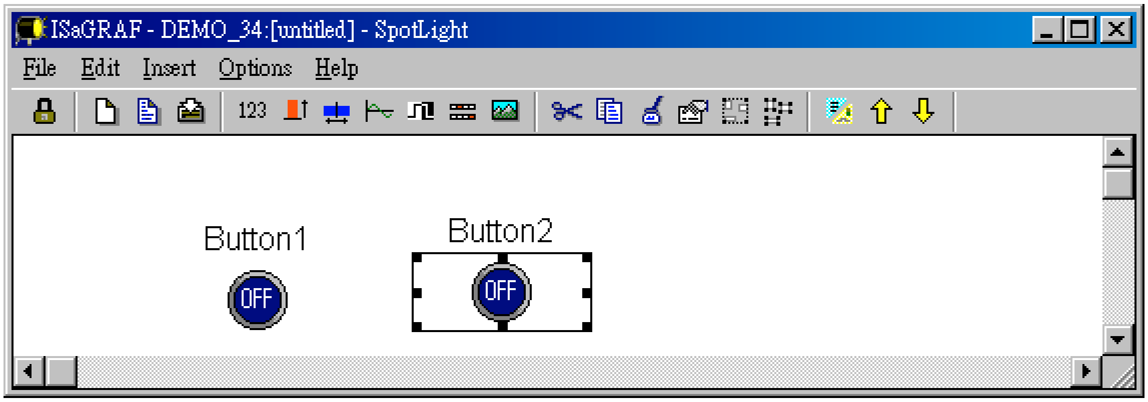
Check on the new created boolean icon, copy it(Ctrl+c) and then paste it (Ctrl+v) to reproduce one another boolean icon. Then drag it to the prefered place.



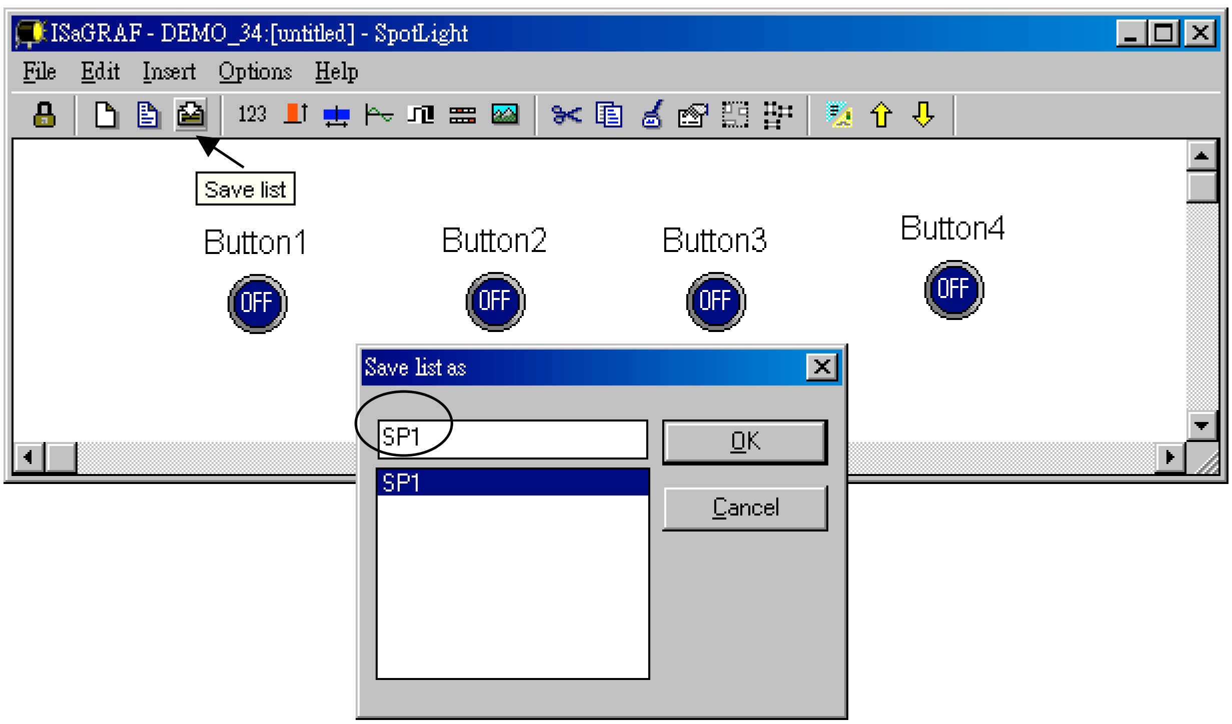
Check on the new created boolean icon, then click the right button of the mouse, select "Set item style" to modify the name to "Button2".



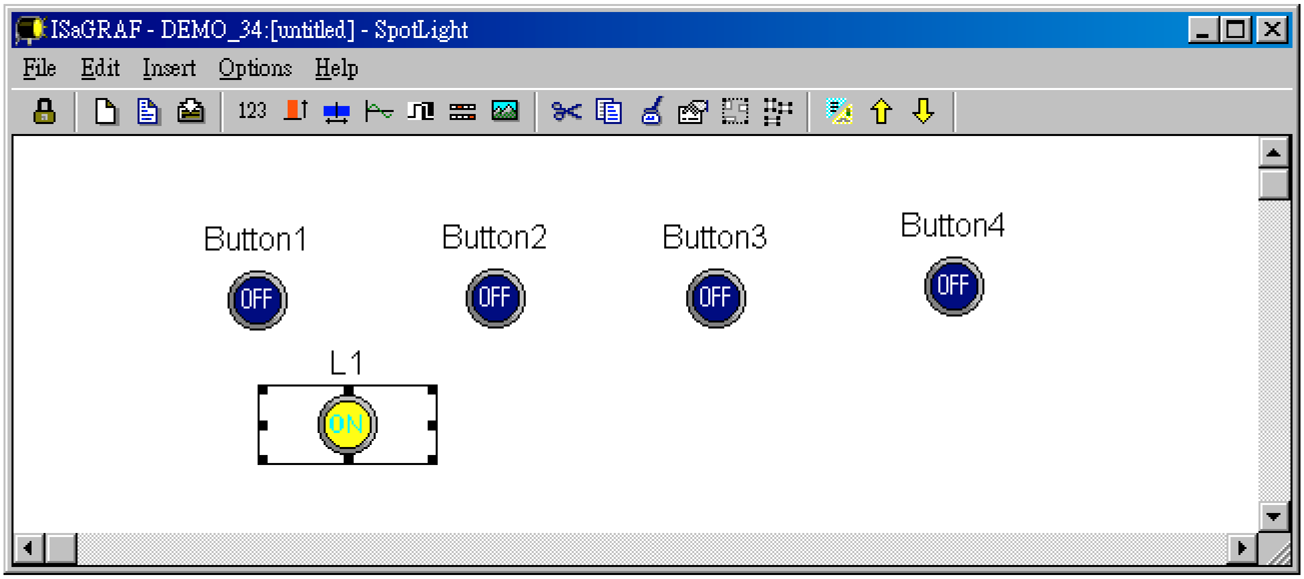
Then we have …



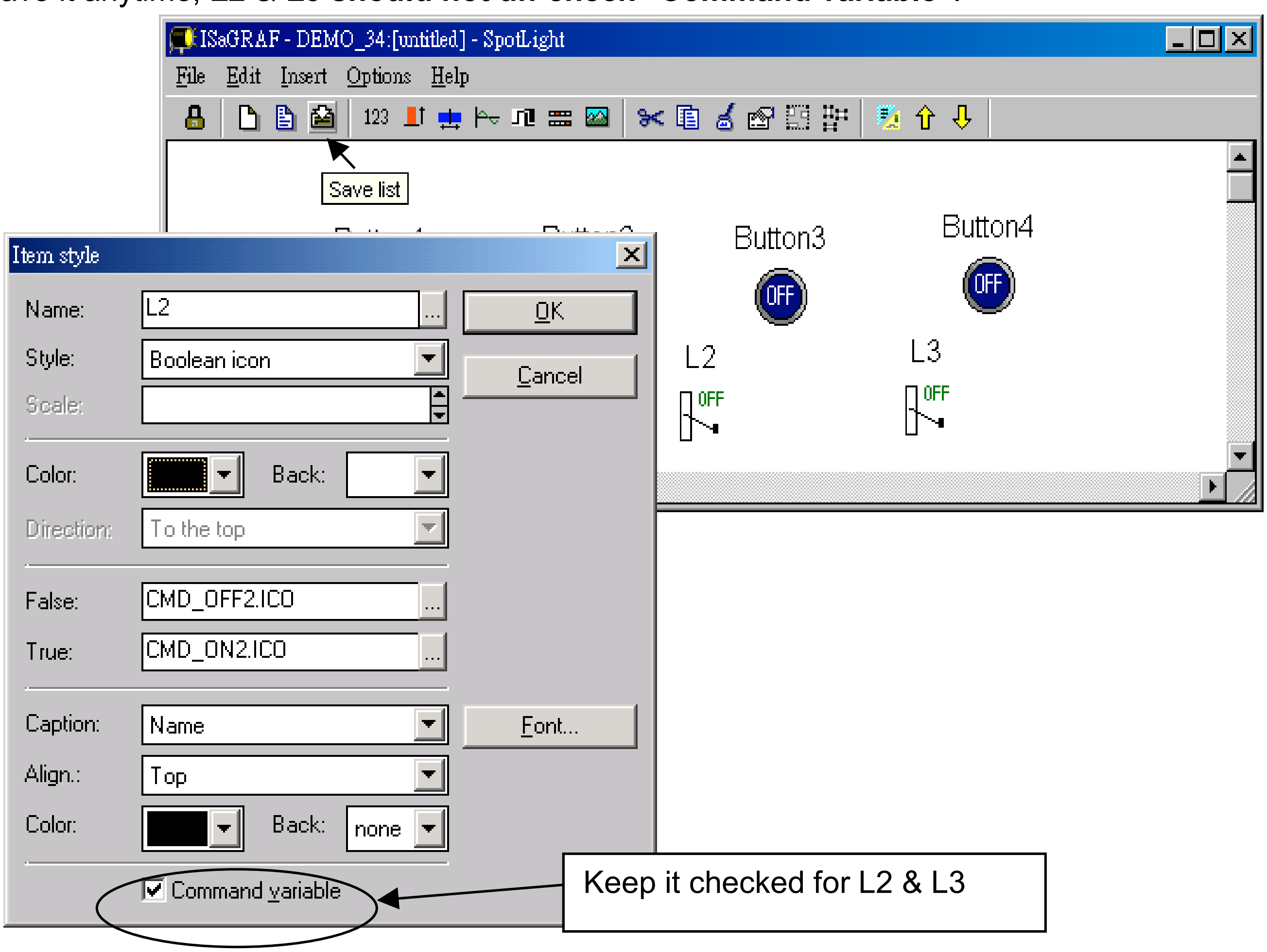
Follow the same method to create 4 boolean icons as below. Recommand to save it anytime for safety. Given a name to this screen.



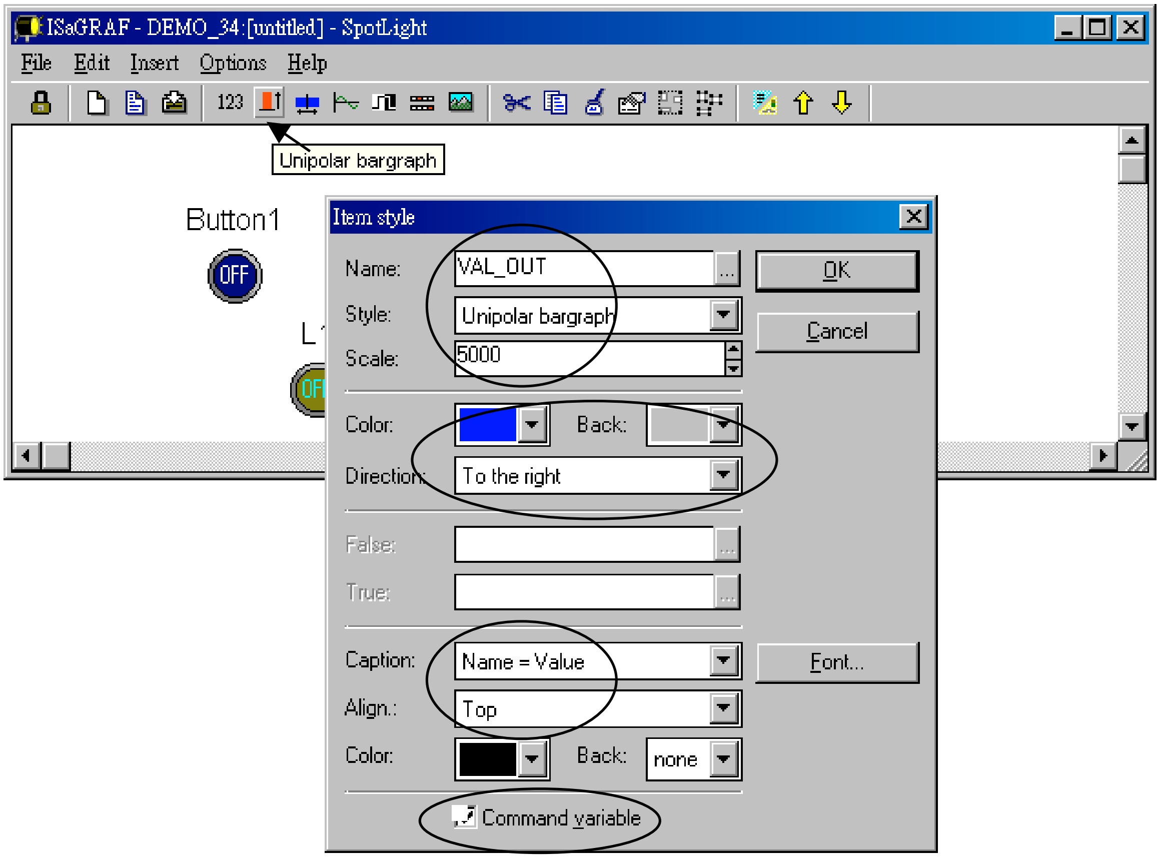
We need one another Boolean icon to display the status of “L1”. Create it with a different color (TRUE : “YEL_ON2.ico” , FALSE : “YEL_OFF2.ico”).



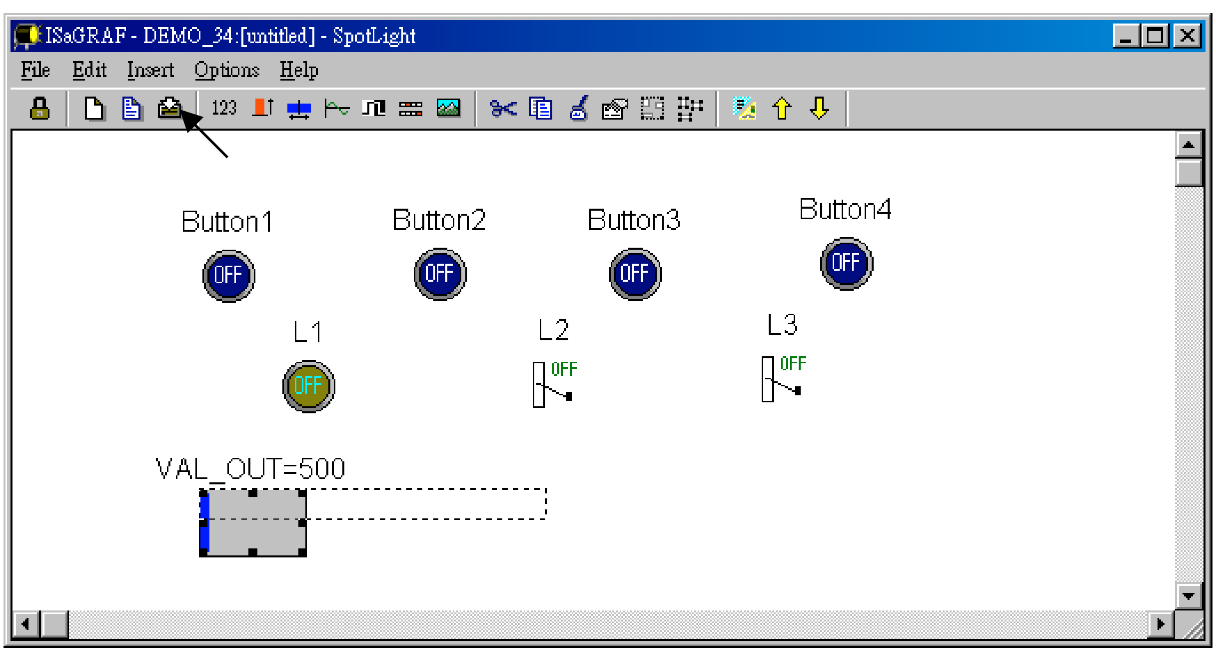
And then create L2 & L3 with TRUE:”CMD_ON2.ico” and FLASE: “CMD_OFF2.ico” as below. Save it anytime, **L2 & L3 should not un-check “Command variable”**.



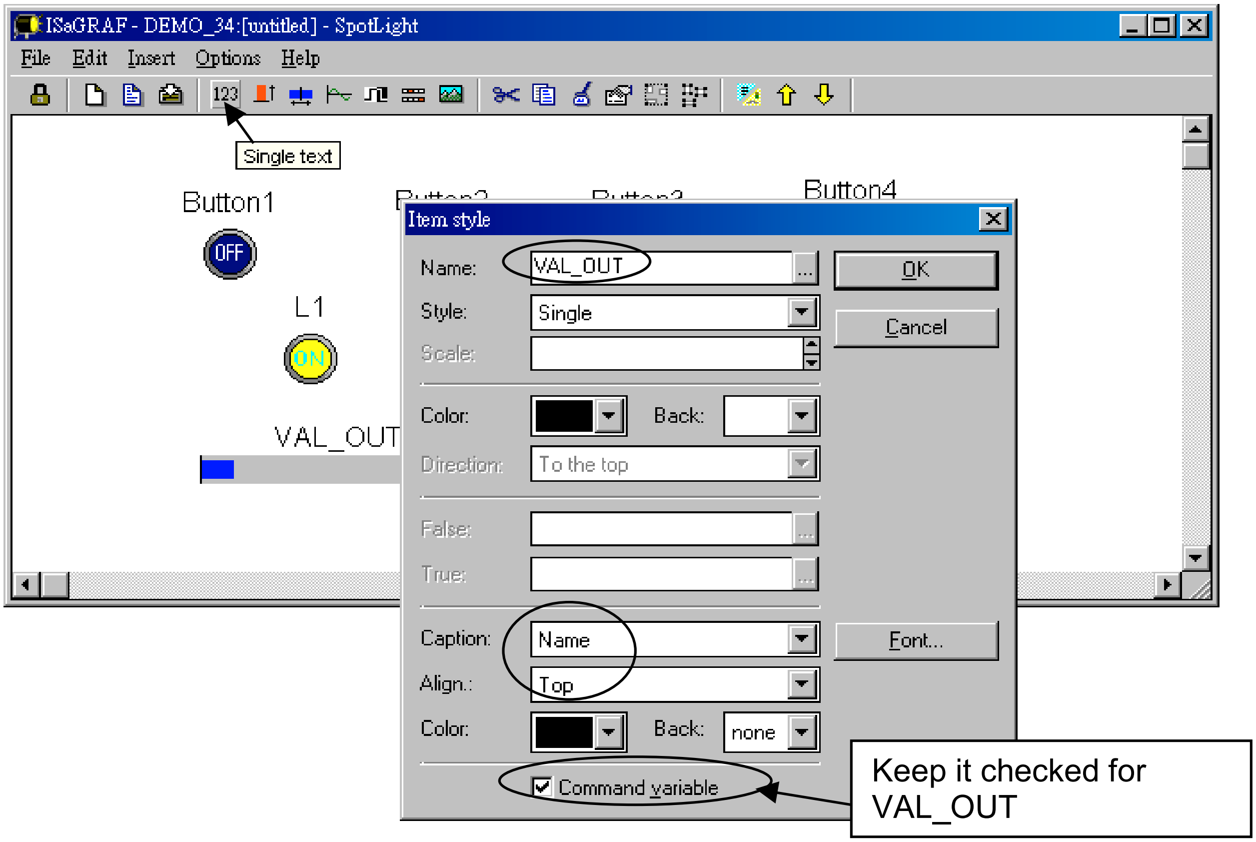
Add “Unipolar bargraph”
Click on “Unipolar bargraph”, set the associated Name as “VAL_OUT”, Scale as “5000”, Color as blue, Back as gray, Direction as “To the right”, Caption as “Name=Value”, Align as “Top”, and un-check “Command variable”



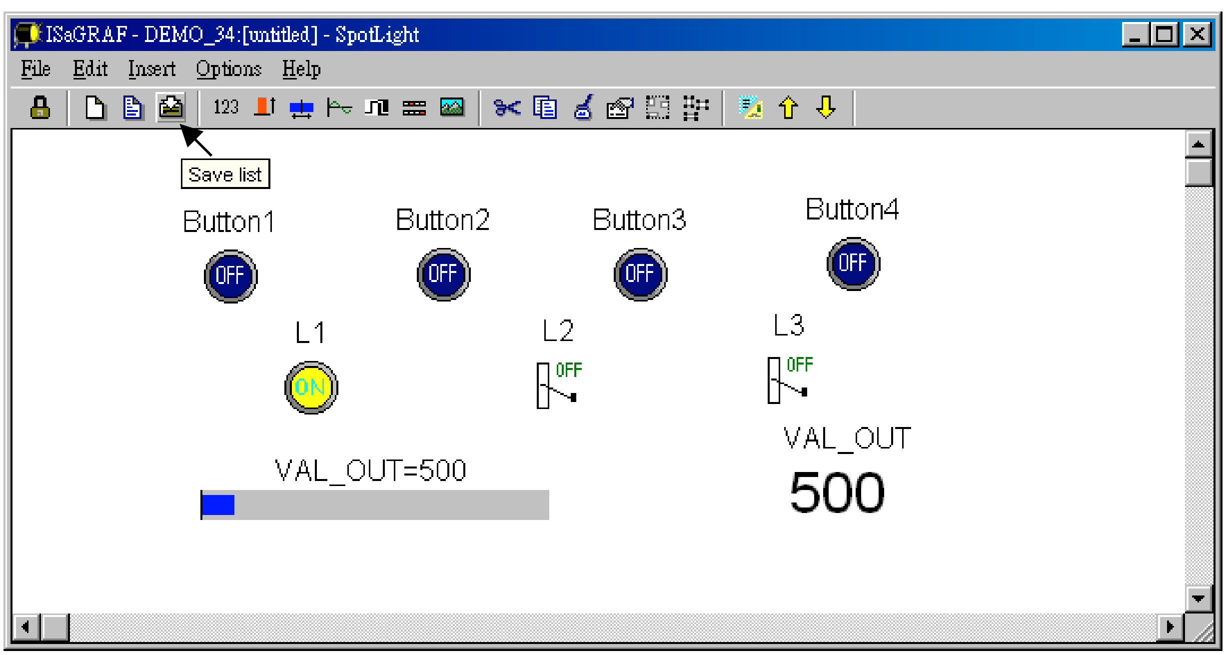
Click and hold on the left button of the mouse to change to the prefered shape as below. Save it anytime.



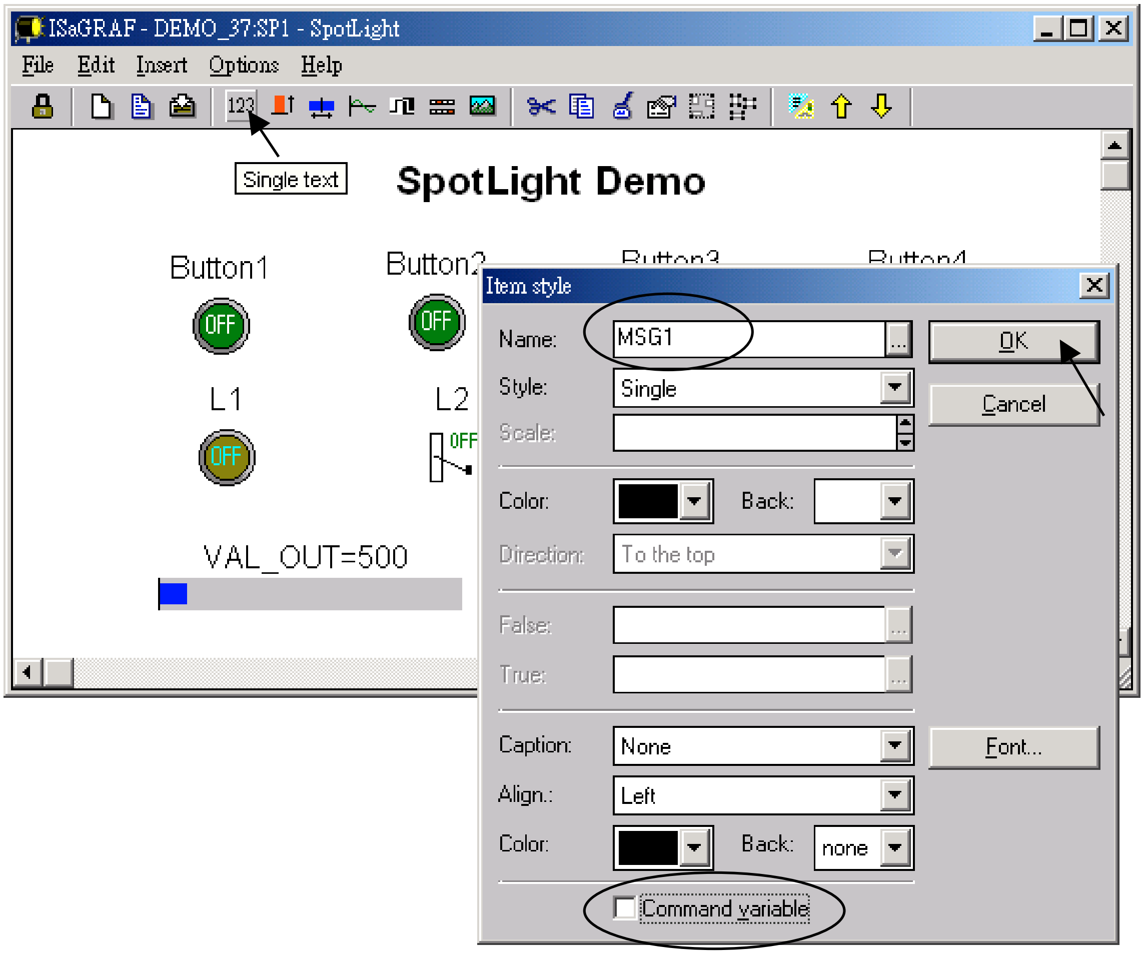
Add “Single text”
Click on “Single text”, set the associated Name as “VAL_OUT”, Caption as “Name”, Align as “Top”



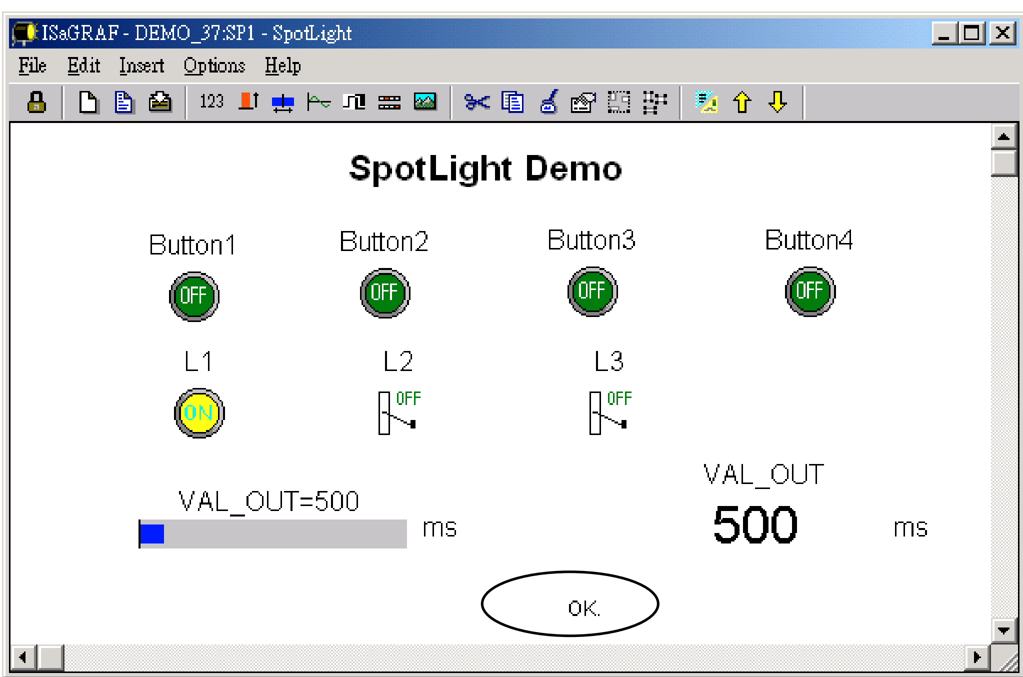
Move it to the prefered place and save it.



Click on “Single text” again, set the associated Name as “MSG1”, Caption as “None”, Align as “Left” and un-check “Command variable”.

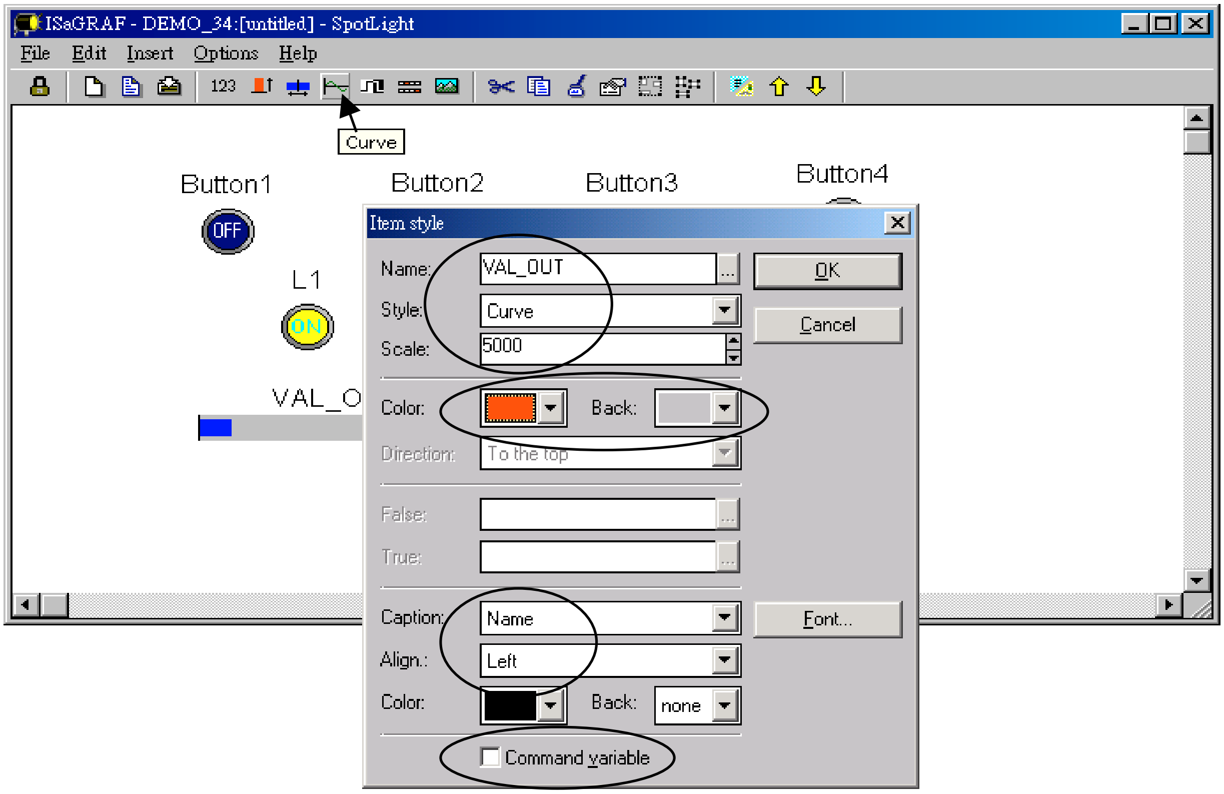


Move it to the prefered place and save it.

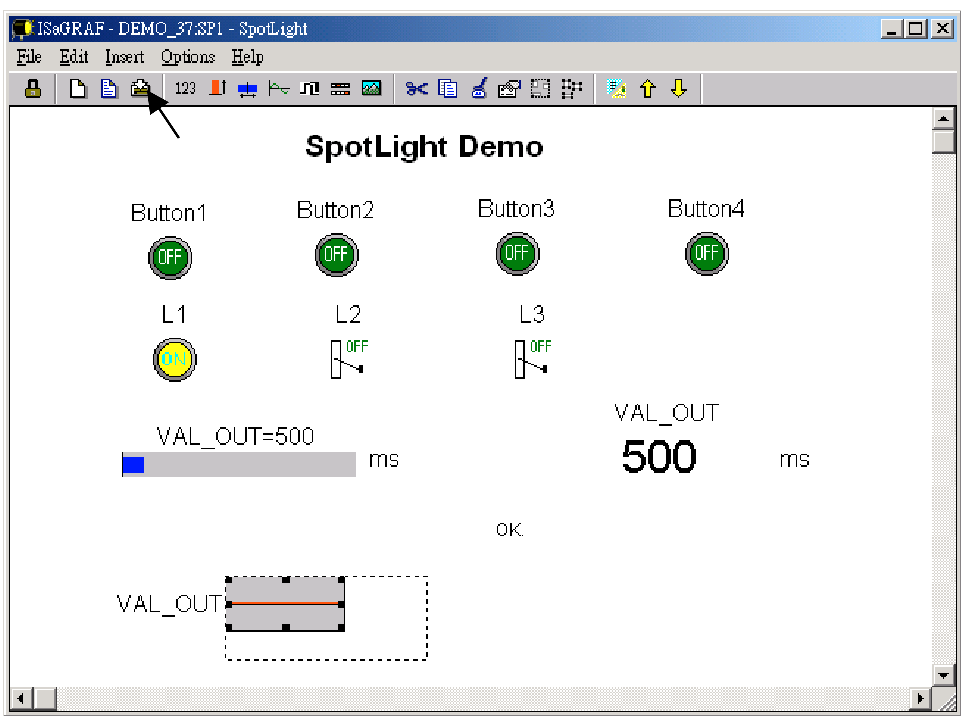


Add “Curve”

Click on “Curve”, set the associated Name as “VAL_OUT”, Scale as “5000”, Color as red, Back as gray, Caption as “Name”, Align as “Top”, and un-check “Command variable”

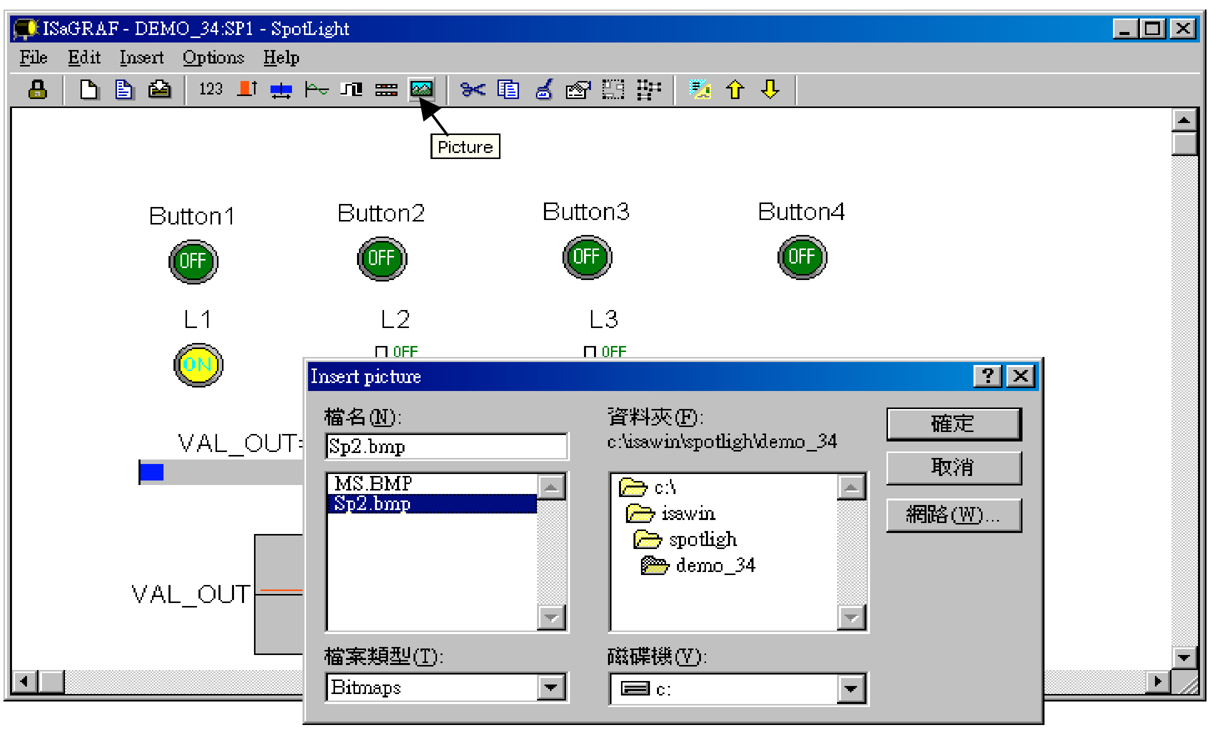


Click and hold on the left button of the mouse to change to the prefered shape as below. Save it anytime

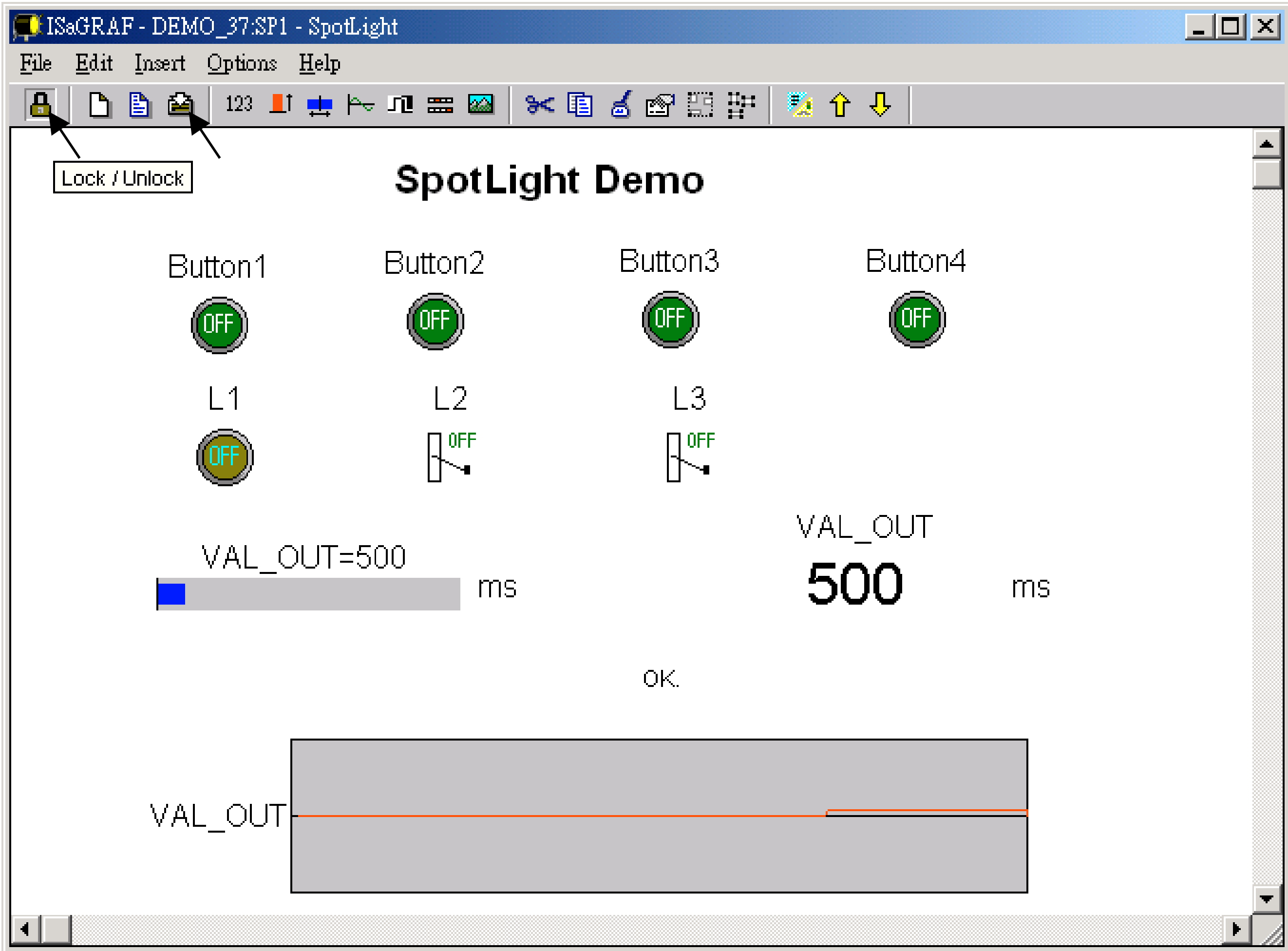


Add “picture”
Please build 2 bitmap pictures by MS painter as below. Then save them respectively with file names of “sp2.bmp” & “ms.bmp” to the associate project directory. (For this example “c:\isawin\spotligh\demo_37\”)

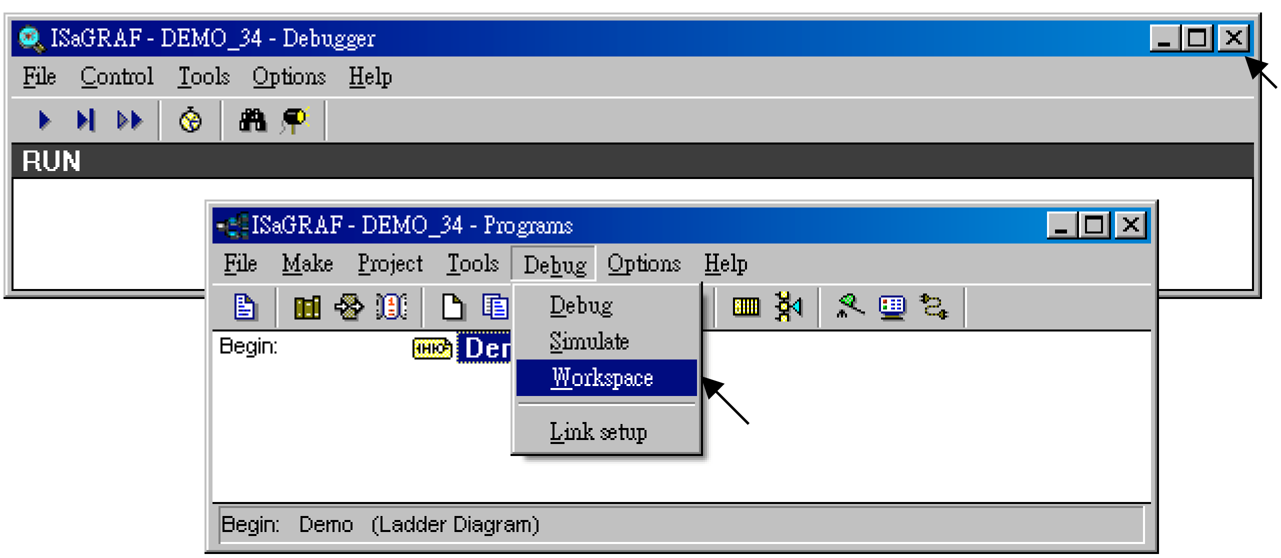
Click on “Picture”, Select the associate bmp file name.



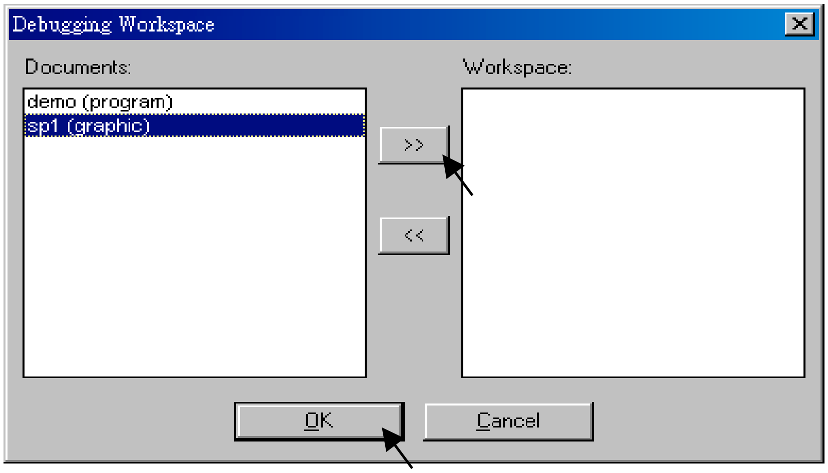
Add 2 pictures “sp2.bmp” and “ms.bmp” to the prefered place, then we got the below window. Click on “Lock” to protect it (No modification allowed). Save it anytime.



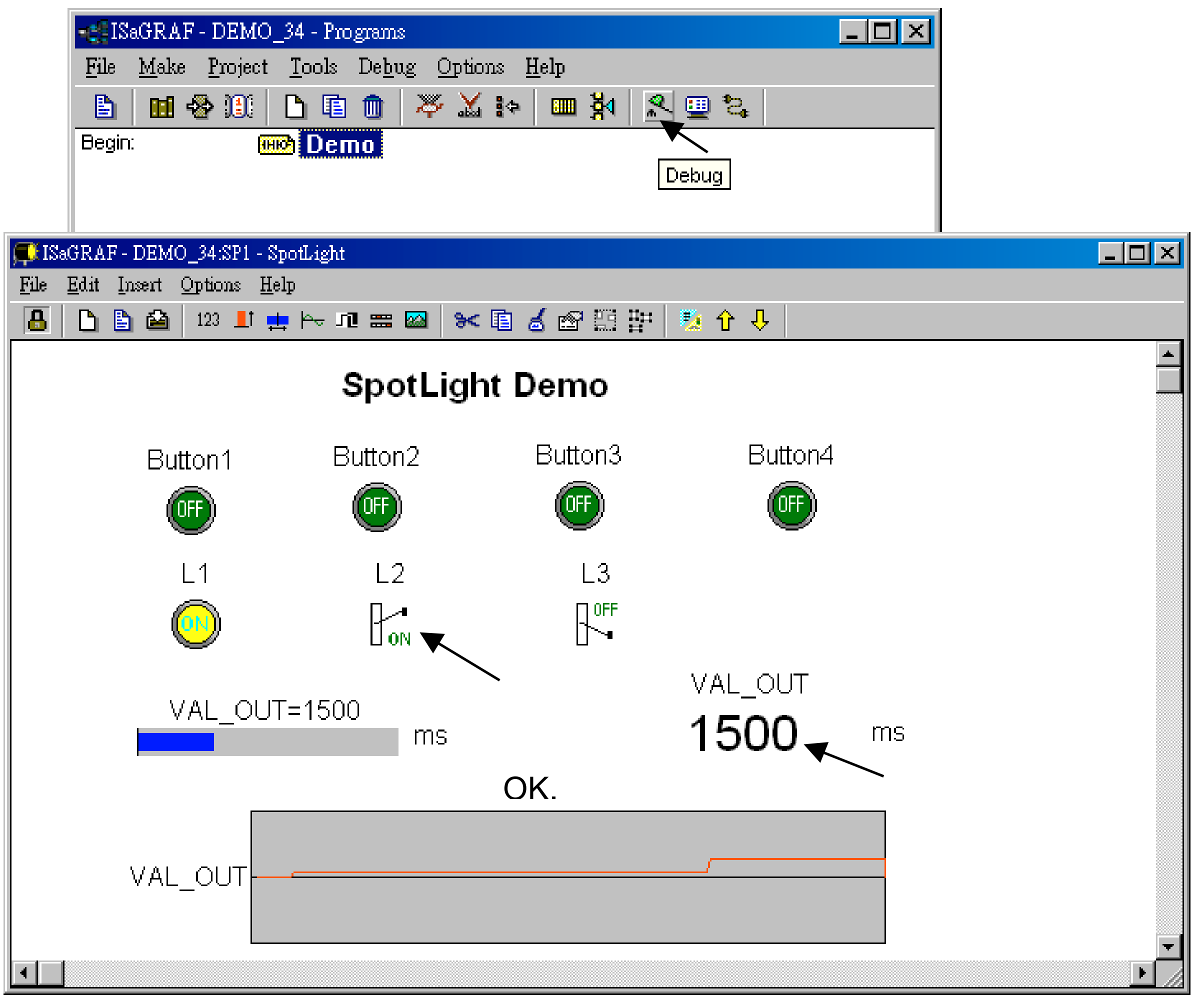
Add the HMI screen to the “Workspace”
Quit “simulation”, then run “Debug”-“Workspace”.



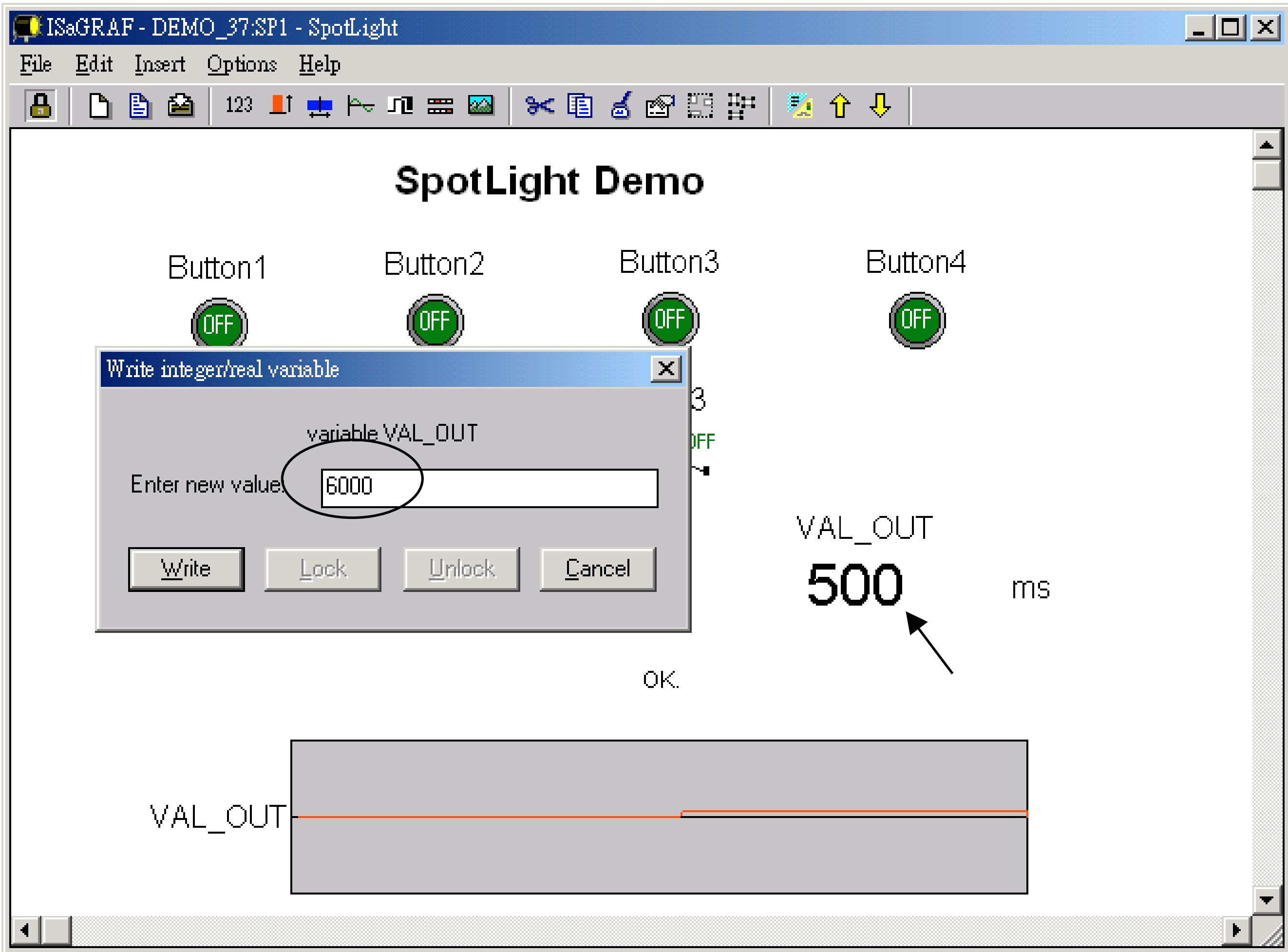
Move the HMI screen to the right (Workspace).



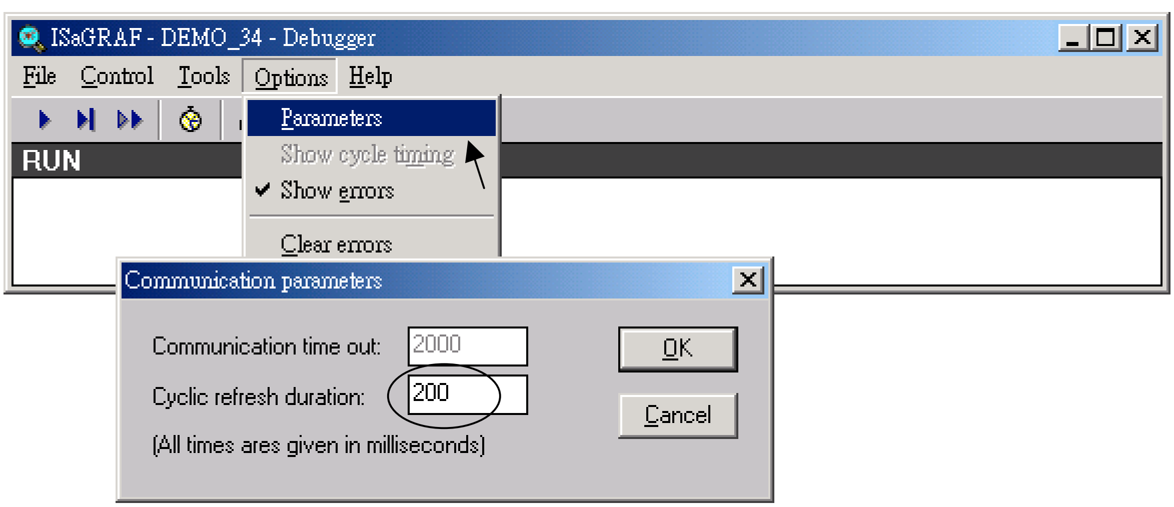
J. Time to download to the controller and test
Click on “Debug” to download the project to the controller and test it. You may double click on “L2”, “L3” or “VAL_OUT” to modify the value and see what it happens on the controller. And also you can press the 4 pushbuttons on the controller.



You may double click on “VAL_OUT” and give a value large than 5000 to see what it happens.



Note: For quick response, user may click on “Options” – “Parameters”, and then set the “Cyclic refresh duration to a smaller value. (Recommand not to set below 200 ms)



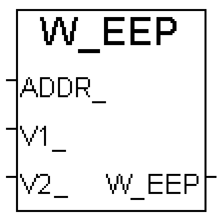
Chapter 15: Creating User-Defined Functions

ISaGRAF supoorts functions written in ST, FBD, IL and QLD languages. User-defined functions are normally for some algorithm which been used again and again.

A function always has an return value (output parameter) and its name should be the same name as the function, and may have up to 31 input parameters. The code written inside functions can not call any **function block**, however can call other ISaGRAF standard **functions** and **c functions** provided by ICP DAS.

We are going to creating a function to save an integer value to the EEPROM. Its format is as the below.

Function name : W_EEP
Description: Save an integer to the EEPROM when its value changed
Input parameters:
    ADDR_ (integer) : the address of the EEPROM to write
    V1_ (integer) : New value
    V2_ (integer) : Old value
Return parameter:
    W_EEP (integer): return the new value

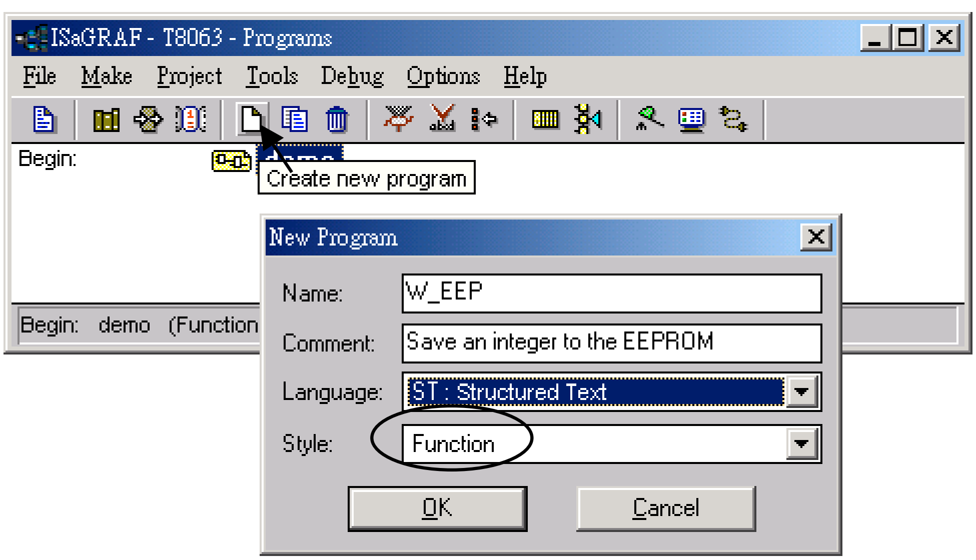


Note: The parameter names been used will become reserved names. That's why we use ADDR_ , V1_ , V2_ rather than ADDR , V1 & V2.

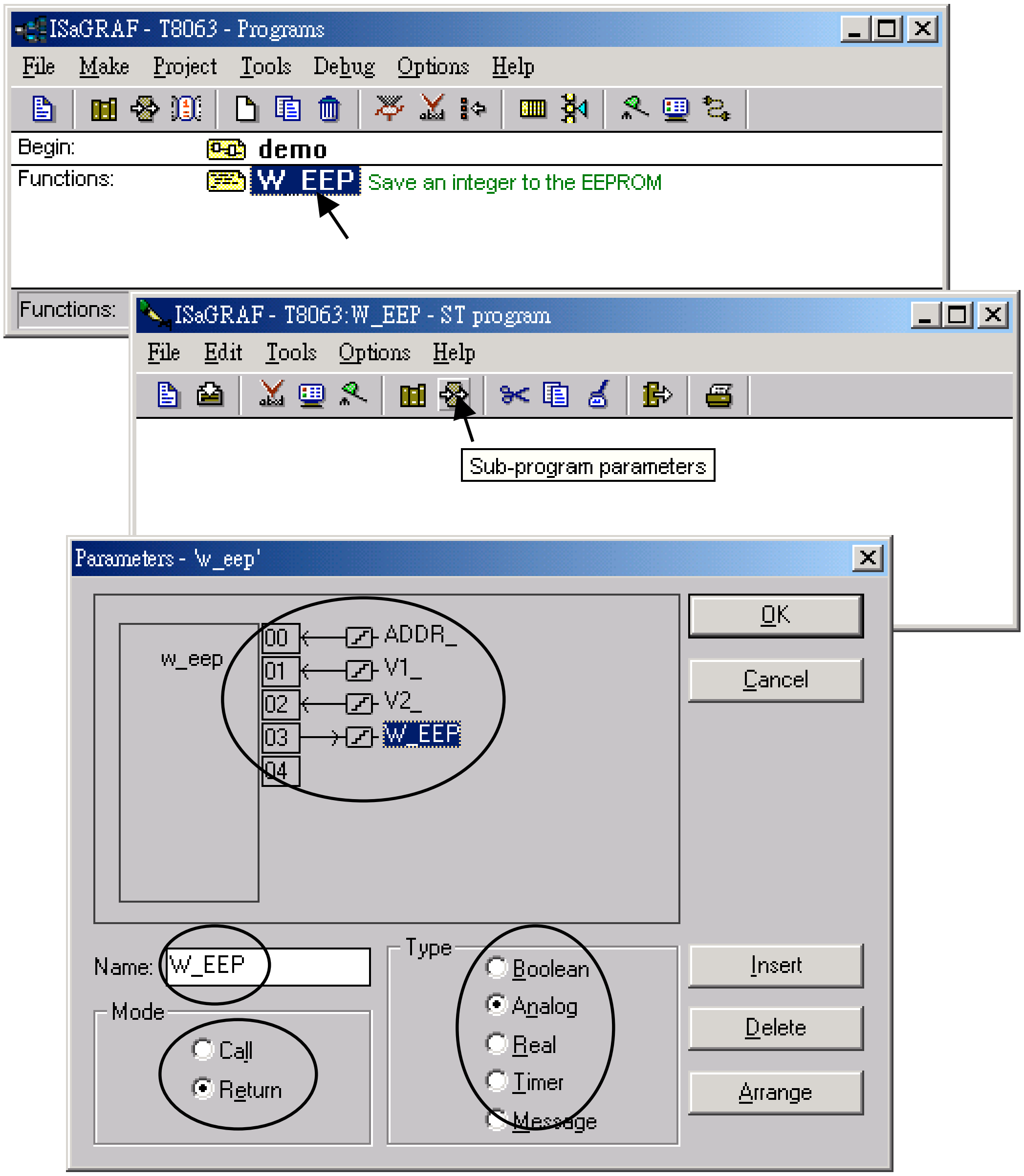
15.1: Creating functions inside one project

Functions created inside one project can be only called by other programs written in the same project.

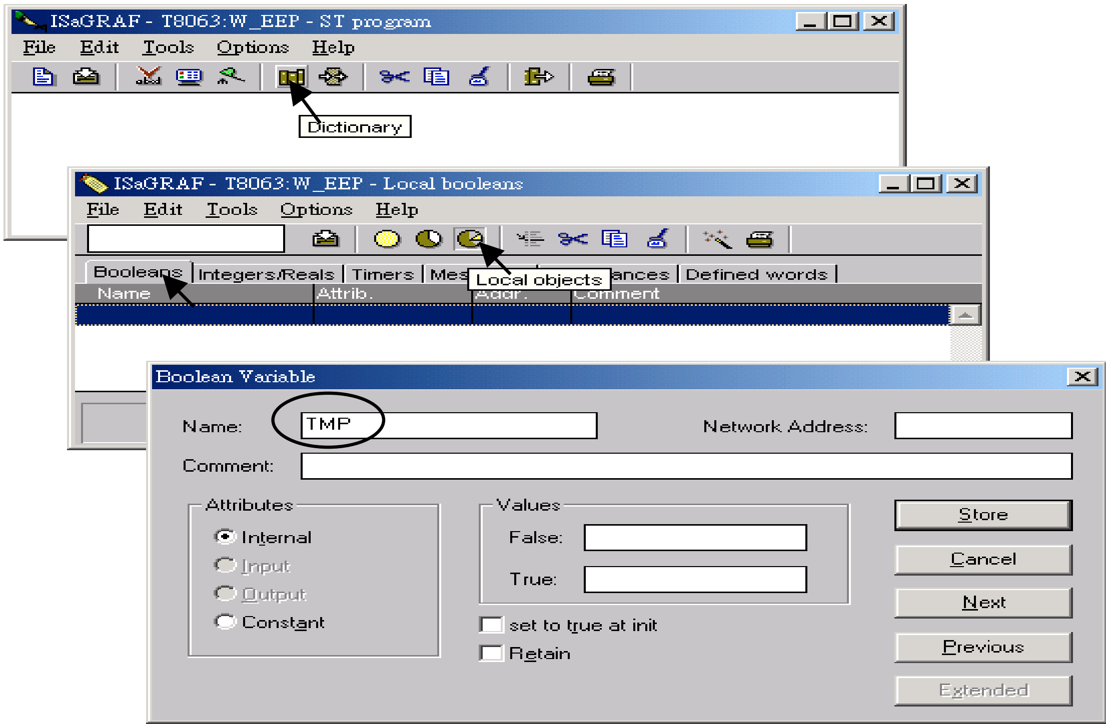
A. Click on "Create new program" inside the project. Given Name as "W_EEP", Language as "ST:…", Style as "Function".



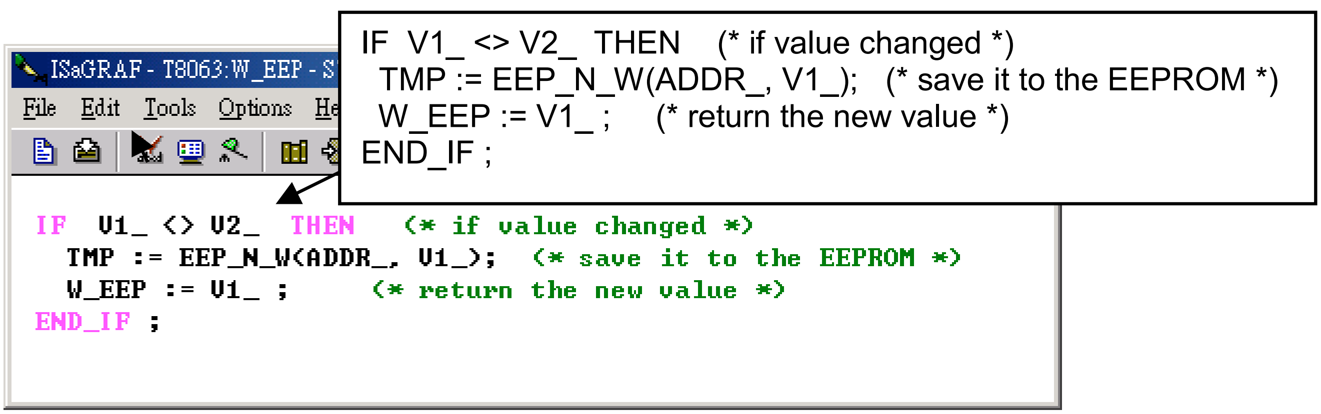
B. Double click on the function to get into it. Then click on “Sub-program parameters” to define input and output parameters.



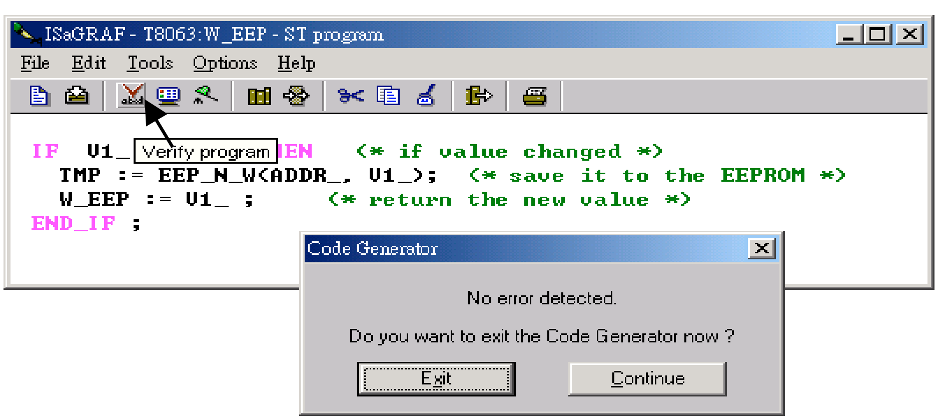
C. Declare local variables. We need a local **boolean internal** variable “**TMP**” in this example.



D. Enter function codes.



E. Verify the function.

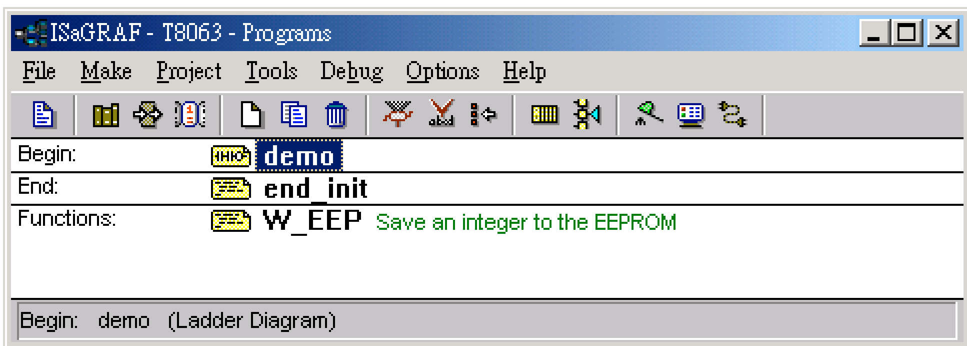


F. Call it in other programs in the same project.

Global variables used in the project:

Name	Type	Attribute	Description
INIT	Boolean	**Internal**	**initial value at “TRUE”**. TRUE means 1st scan cycle
K1	Boolean	**Input**	Connect to 1st ch. Of “push4key”, press it to get “Val”
New_Val	Integer	**Internal**	New value wish to save to the EEPROM
Old_Val	Integer	**Internal**	Old value
Val	Integer	**Internal**	Read back value of the EEPROM

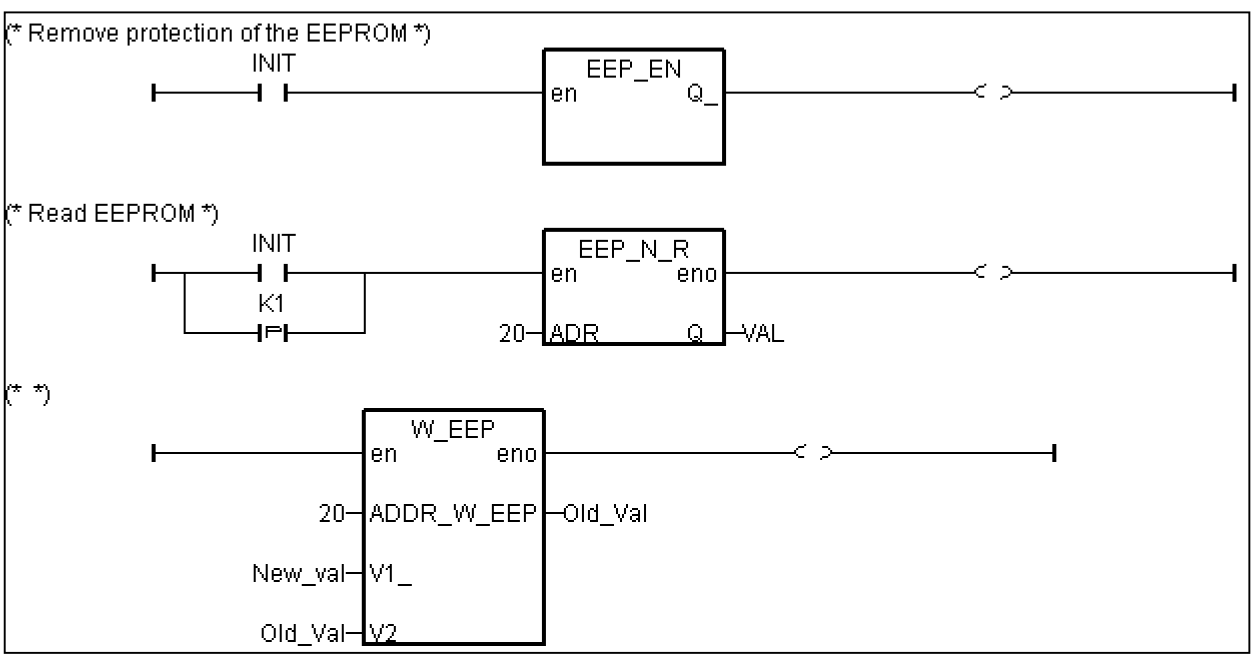
Project architecture:



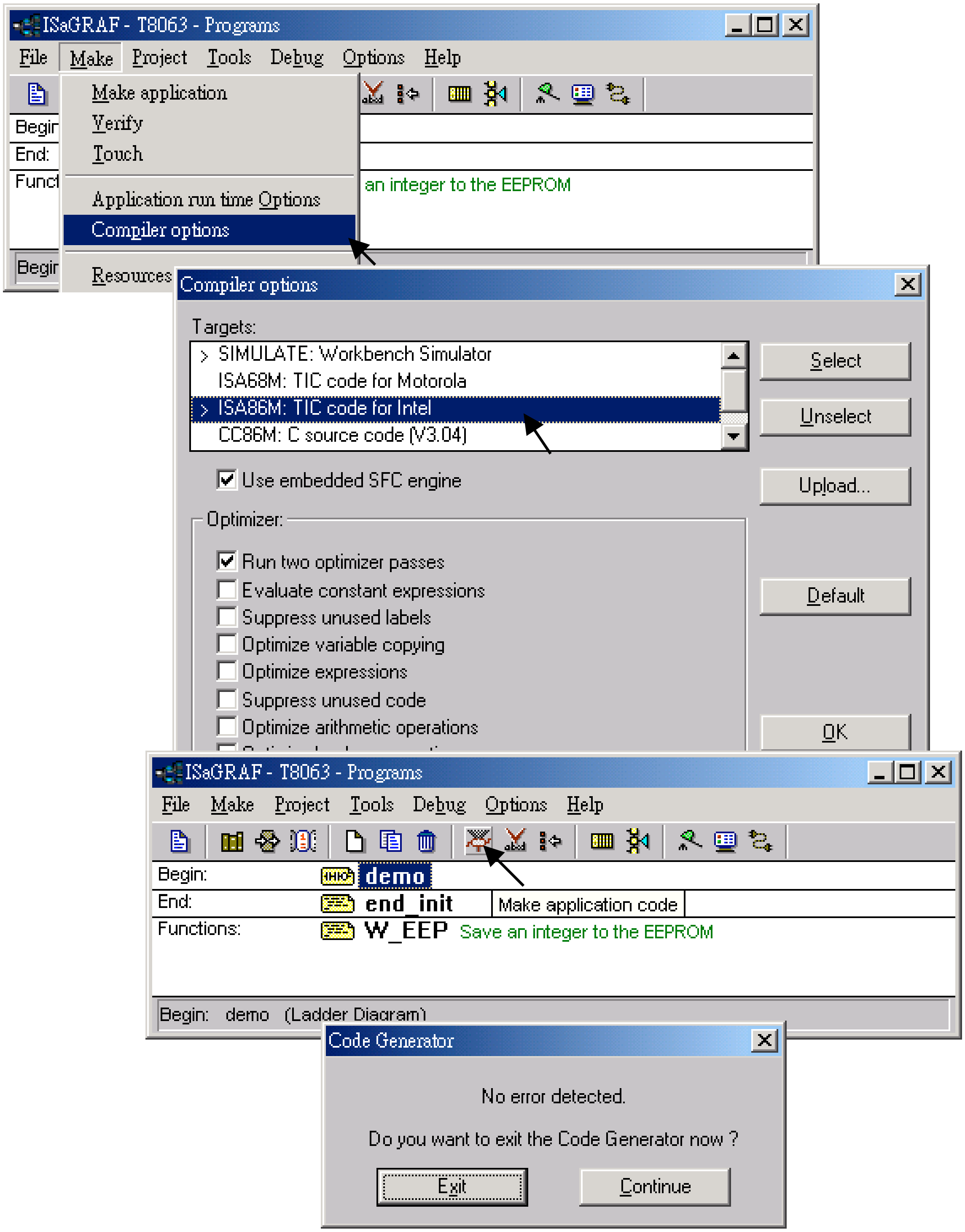
ST program – “end_init” in the “End” area :

```
IF  INIT=TRUE  THEN
     INIT := FALSE ;
END_IF ;
```

LD program – “demo” :



G. Set Compiler Options and compile the project.



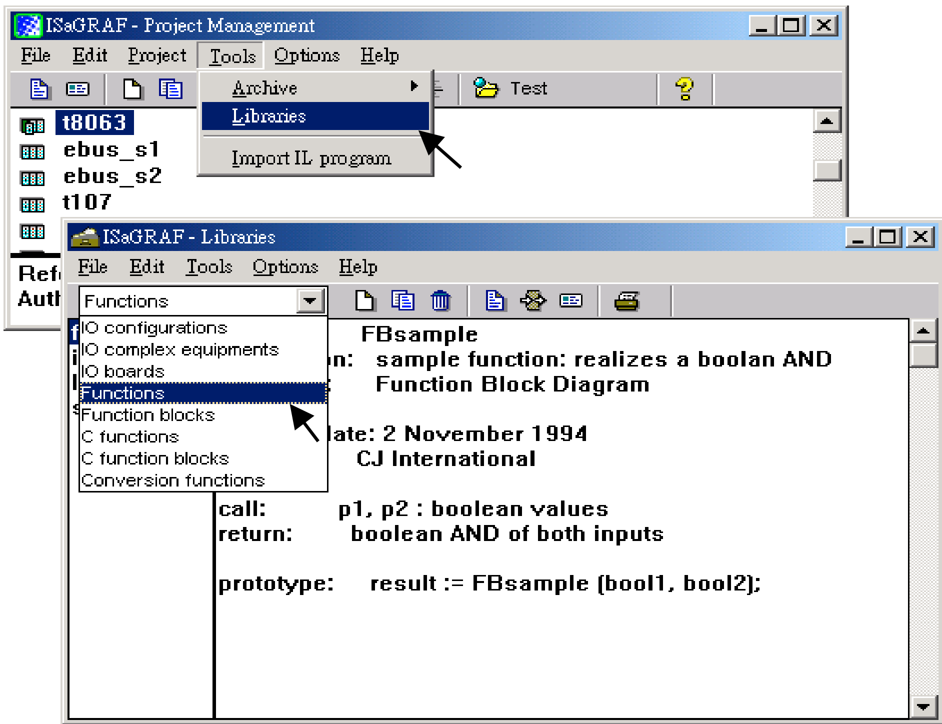
After download to the controller, you may change the “New_Val”, and then press “K1” to see what it happens.

15.2: Creating functions in the library

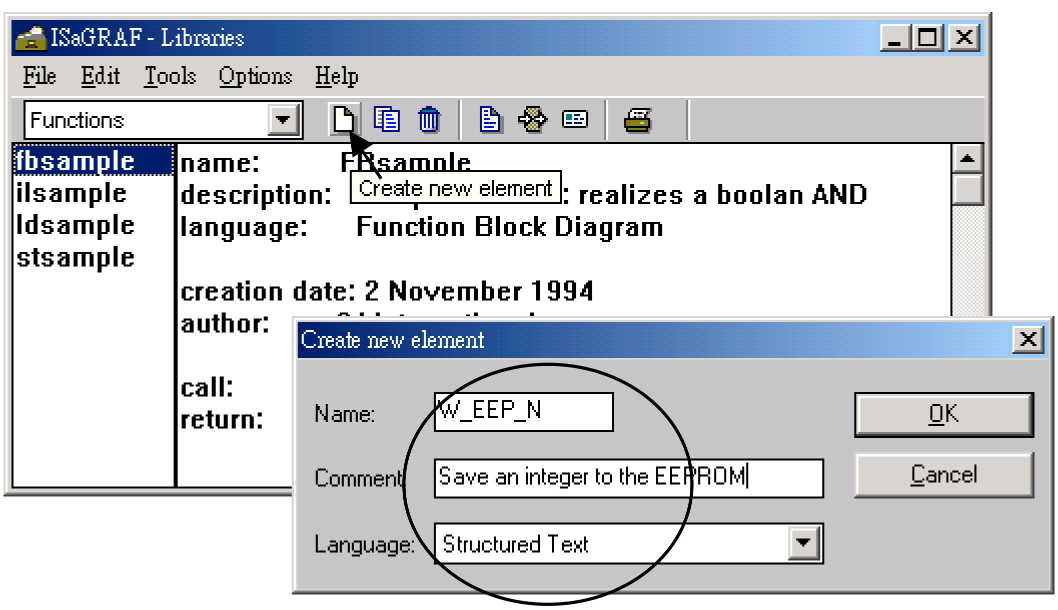
Functions created in the library can be called by programs in any project.

The steps is similar to the former section 15.1. Please refer to it in advance.

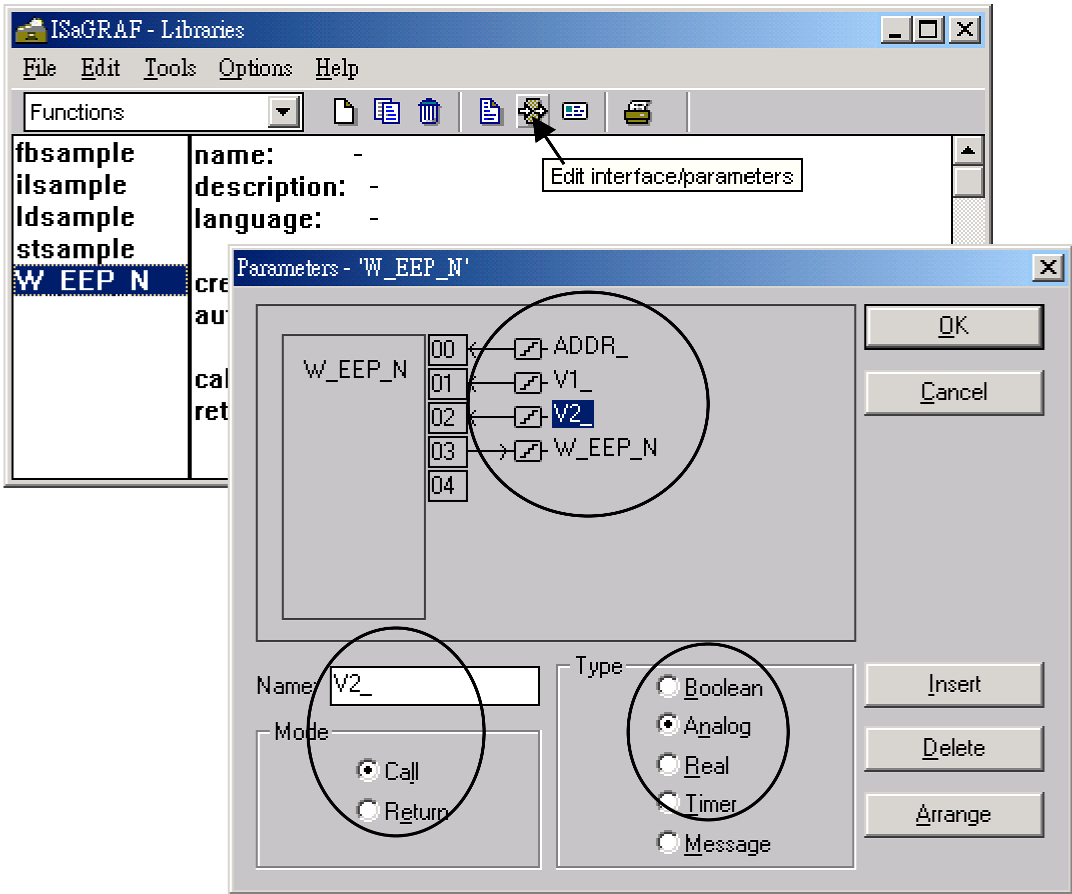
A. Get into the library. Then click on “Functions”



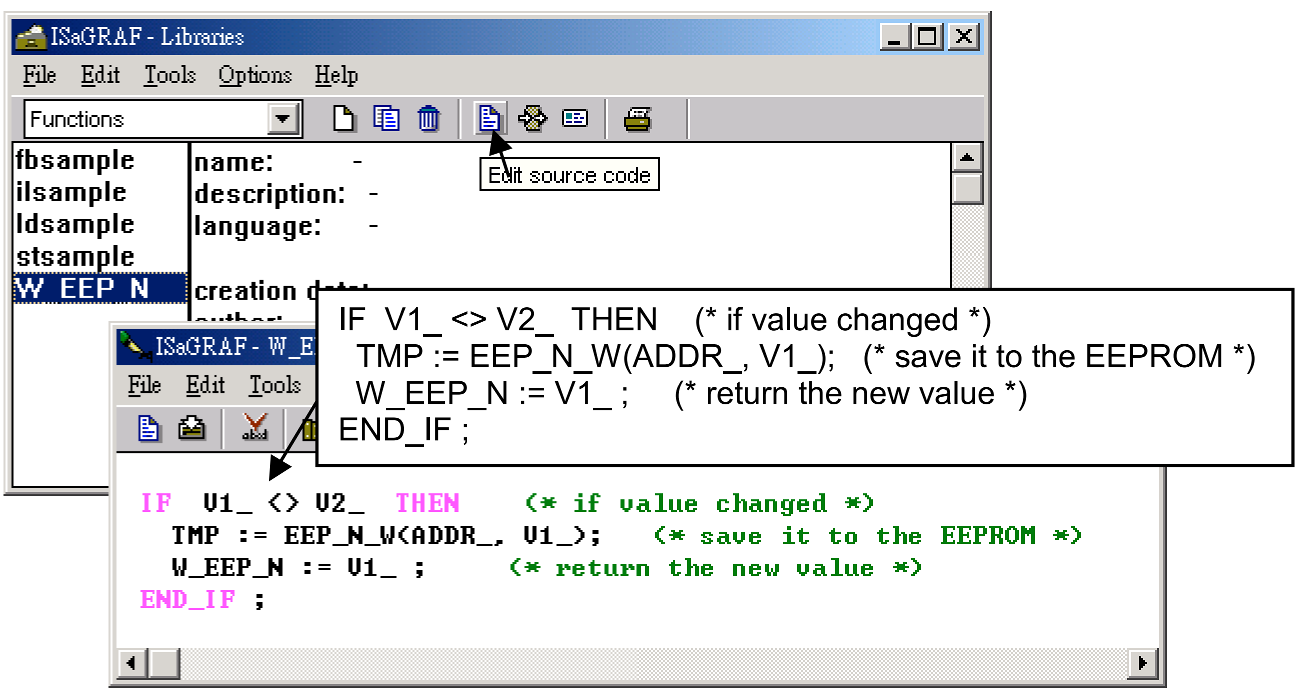
B. Create an new function and given Name as “W_EEP_N” , Language as “Structured Text”.



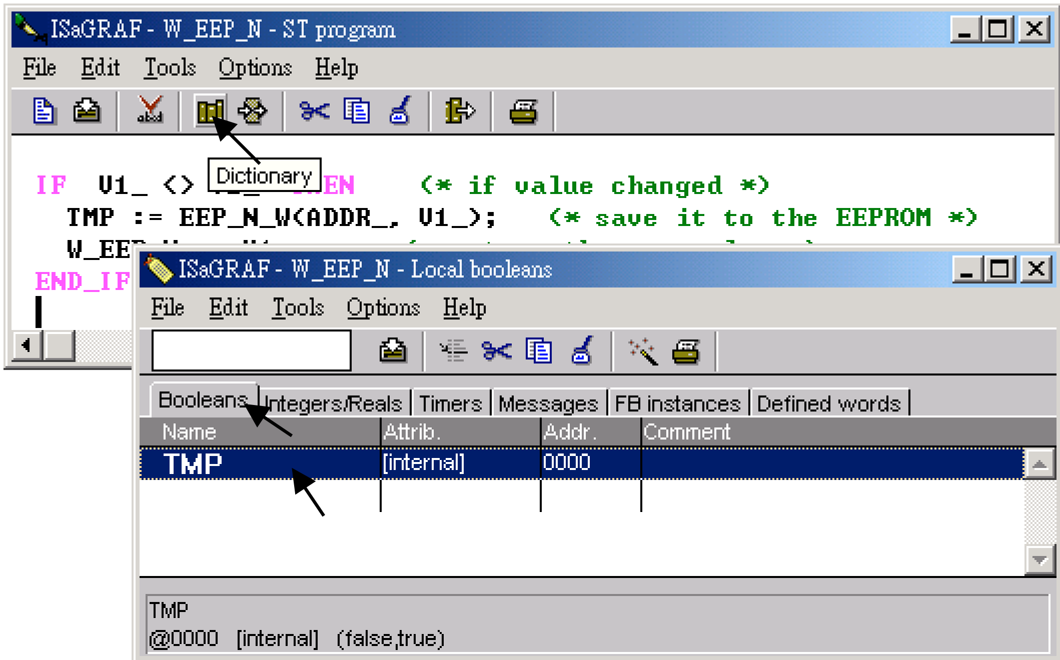
C. Define input and return parameters



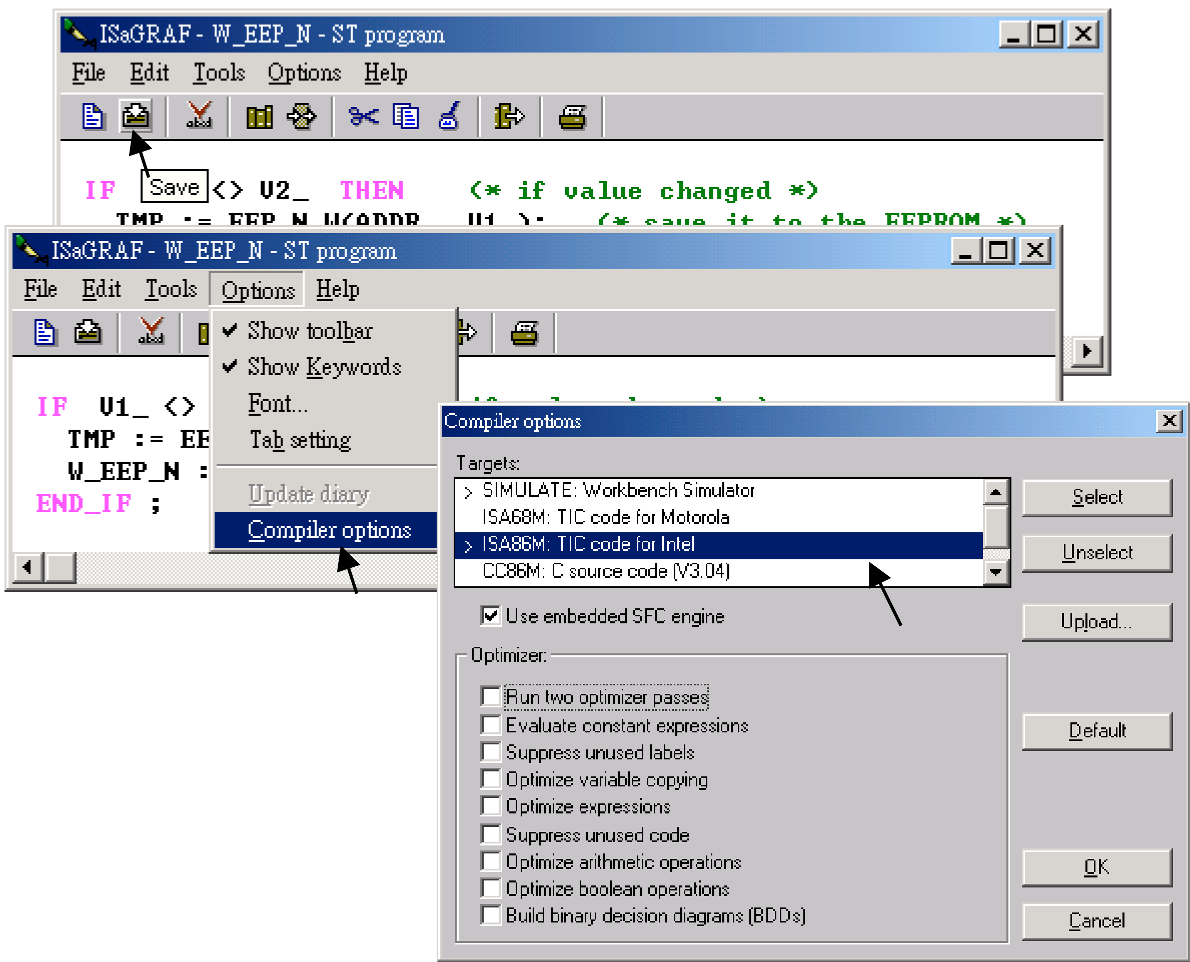
D. Add codes.



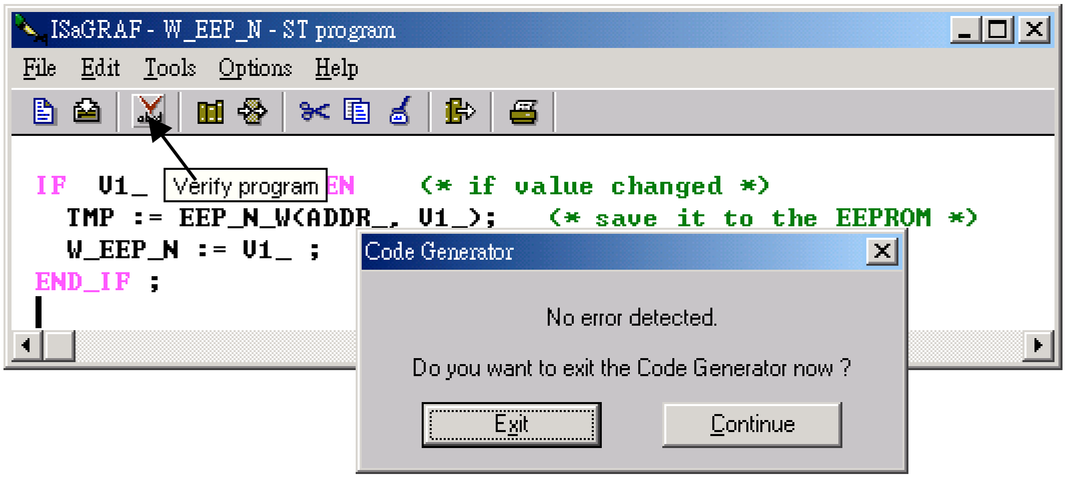
E. Declare local variables. We need a boolean internal variable – "TMP"



E. Save the function and set compiler options.



E. Verify the function.



Then you can call it in any project.

Chapter 16: Linking MMICON

The I-8417/8817/8437/8837, I-7188EG, I-7188XG & W-8xx7 controller can integrate the ICP DAS's MMICON to become their Man Machine Interface. The MMICON is featured with a 240 x 64 dot LCD and a 4 x 4 Keyboard. User can use it to display picture, string, integer, float, and input a character, string, integer and float. All control logic is written in ISaGRAF program.

16.1: Hardware Installation

Please refer to the "MMICON Hardware Manual" which is delivered with the hardware for more hardware details.

1. The MMICON has a COM port. Please set as a RS232 port. (Please look at the jumper "J7" & "J8" setting on the hardware).

Pin assignment :

I-8417/8817/8437/8837: COM3 & COM4 can be used. **W-8xx7**: COM2 can be used

I-8xx7 (COM4) W-8xx7 (COM2)		MMICON (CN3) RS232
2 TXD	———	2 RXD
3 RXD	———	3 TXD
5 GND	———	5 GND

I-8xx7 (COM3) RS232		MMICON (CN3) RS232
3 TXD	———	2 RXD
2 RXD	———	3 TXD
5 GND	———	5 GND

I-7188EG/XG: COM3 can be used. (COM3 is added on X503 ~ X51x board)

I-7188EG/XG RS232		MMICON (CN3) RS232
TXD	———	2 RXD
RXD	———	3 TXD
GND	———	5 GND

2. Please set Jumper "J2" of MMICON to position "INIT". I-8417/8817/8437/8837, I-7188EG/XG & W-8xx7 only support COM parameter "9600, 8, N, 1" and "address = 0" to talk to the MMICON.

16.2: Create Background Picture Of the MMICON

Please refer to the "MMIDOS Software User Manual" which is delivered with the hardware for more software details.

The number of the background pictures depends on the ROM memory on the MMICON. It can up to 256 pages for EPROM like "27040", and 128 pages for "27020", and 64 pages for "27010".

Note: ROM/ EPROM/ EEPROM/ FLASH are all validate.

Please Install the "MMICON" folder from the CD-ROM: \Napdos\others\mmicon\ to your hard disk. Or download folder "MMICON "at ftp://ftp.icpdas.com/pub/cd/8000cd/napdos/others/mmicon/

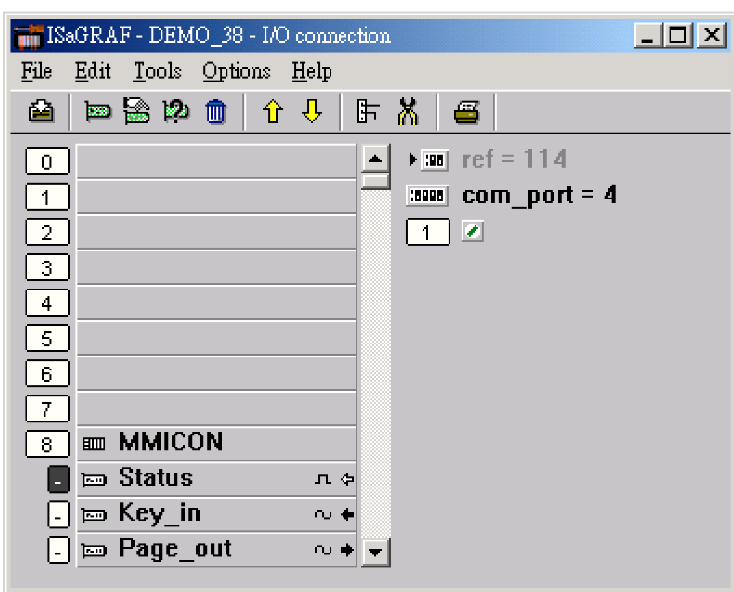
Note: Please change all these file's attribute : removing "Read-only"

Create all the background pages by Microsoft painter (Please refer to "P0.bmp").
Edit your "Autox.dat" file (Please refer to "Auto1.dat"). This file must remove its "Read-only" attribute.
Run "MMIDOS.exe" to build the "romx.bin", For ex. "rom1.bin"
Using your ROM programmer to burn this "romx.bin" image to the ROM memory. Then plug it into the socket on the MMICON.

Please refer to the "MMIDOS Software User Manual" which is delivered with the hardware for more software details.

16.3: Writing Control program

The I/O complex equipment "mmicon" should be connected to the I/O connection window first. You can find 3 boards under "MMICON".



Status:

Parameter "com_port" defines the COM No. to link to the MMOCON. 3 or 4 for I-8xx7, while 2 or 3 for I-7188EG/XG & 2 for W-8xx7

1 channel of Digital Input: True means communication between the controller and the MMICON is Ok. FALSE means fail.

Key_in:

1 channel of Integer Input: The value is the key been pressed. And the value will last only for one scan cycle, then go back to 0.

Key	Key code value	Key	Key code value
0	16#30	**Enter**	16#0D
1	16#31	**.**	16#2E
2	16#32	**Left**	16#1B
3	16#33	**Right**	16#1A
4	16#34	**Up**	16#18
5	16#35	**Down**	16#19
6	16#36	**Back space**	16#08
7	16#37	**F1**	16#F1
8	16#38	**F2**	16#F2
9	16#39	**F3**	16#F3
A	16#41	**F4**	16#F4
B	16#42		
C	16#43		
D	16#44		
E	16#45		
F	16#46		

Page_out:

1 channel of Integer Output: The value output define the page No. to display.

The I-8417/8817/8437/8837, I-7188EG, I-7188XG & W-8xx7 controller provide below functions to control the action of the MMICON.

MI_BOO	Display a boolean value as "ON" or "OFF"
MI_INT	Display an integer value
MI_REAL	Display a real value
MI_STR	Display a string
MI_INP_N	To enter an integer
MI_INP_S	To enter a string
REAL_STR	Convert a real value to a string
STR_REAL	Convert a string to a real value

Please refer to I-8xx7's demo_38, dem_39 and Appendix A.4

Chapter 17: SMS: Short Message Service

The I-8417/8817/8437/8837, I-7188EG, I-7188XG & Wincon-8xx7 controller can integrate with a GSM Modem to support SMS: Short Message Service. This allows user to request information or control something from his own cellular phone to the ISaGRAF controller. Beside, the controller can also send information and alarms to user's cellular phone.

17.1: Hardware Installation

The I-8417/8817/8437/8837 supports SMS since its driver version of 2.24, while version 1.14 for I-7188EG, and version 1.12 for I-7188XG, and version of 3.10 for W-8xx7. If your driver is older one, please upgrade the hardware driver to the associate version or a higher version. The driver can be found from the below ICP DAS's web site:

http://www.icpdas.com/products/8000/isagraf.htm

The I/O library should be re-installed if yours is older one. Please refer to section 1.2.
Or you can refer to Appendix A.2 to simply install "C functions" with the below items.
SMS_test, SMS_get, SMS_gets, SMS_send, SMS_sts
and "I/O complex equipment" : SMS.

The GSM Modem **GM29** (900/1800) is recommanded for the ISaGRAF controller since its driver version of I-8xx7:2.47, I-7188EG:1.38, I-7188XG:1.35 & Wincon-8xx7:3.10. You may purchase them from ICP DAS or from your local agent. ICP DAS is not sure for other GSM modems working or not.

Note: Please REMOVE the password setting in SIM card , then plug it into GSM modem.

I-8xx7(COM4/5) W-8xx7(COM2)		GSM cable of GM29
2 RXD	—	2 TXD
3 TXD	—	3 RXD
5 GND	—	5 GND
4 DTR	- - -	4 DSR
7 RTS	- - -	7 CTS

7188EG/XG:COM3/4 RS232		GSM cable of GM29
RXD	—	2 TXD
TXD	—	3 RXD
GND	—	5 GND
DTR (or RTS)	- - -	4 DSR
DTR (or RTS)	- - -	7 CTS

17.2: A SMS demo example

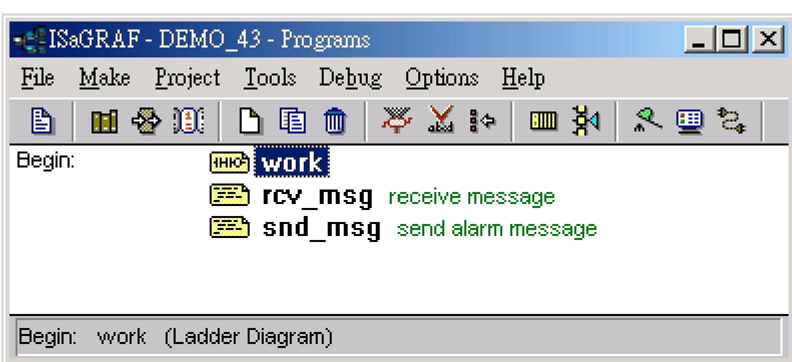
The demo project is located at I-8xx7's demo_43, please refer to section 11.1 to install it to your ISaGRAF workbench. Or It can be download at ICP DAS's ftp site.

ftp://ftp.icpdas.com/pub/cd/8000cd/napdos/isagraf/8000/demo/

Variables :

Name	Type	Attribute	Description
M1	Boolean	**Internal**	Trigger to send an alarm message when K1 is pushed
M2	Boolean	**Internal**	Trigger to send a report message when a message is coming
K1	Boolean	**Input**	Pushbutton 1, connect to push4key
L1	Boolean	**Output**	Output 1, connect to show3led
L2	Boolean	**Output**	Output 2, connect to show3led
L3	Boolean	**Output**	Output 3, connect to show3led
Q1	Boolean	**Internal**	Test if message is coming
TMP	Boolean	**Internal**	Temportary usage
SMS_available	Boolean	**Input**	is SMS available ? connect to SMS - status
T1	Timer	**Internal**	Blinking time of L1 to L3, init at T#500ms
data	Message	**Internal**	The coming Message
phone	Message	**Internal**	phone No. of sender
Date_time	Message	**Internal**	Message coming date & time in string format
To_who	Message	**Internal**	phone No of receiver, **please use your own No.**
Msg_to_send	Message	**Internal**	Message to send out
Year1	Integer	**Internal**	Message coming year
Mon1	Integer	**Internal**	Message coming month
Day1	Integer	**Internal**	Message coming date
Wday1	Integer	**Internal**	Message coming week date
Hour1	Integer	**Internal**	Message coming hour
Min1	Integer	**Internal**	Message coming minute
Sec1	Integer	**Internal**	Message coming second
Q1_cnt	Integer	**Internal**	Message coming count, declared as retained variable
Msg_status	Integer	**Internal**	Message sending status
TMP_v	Integer	**Internal**	temportary usage

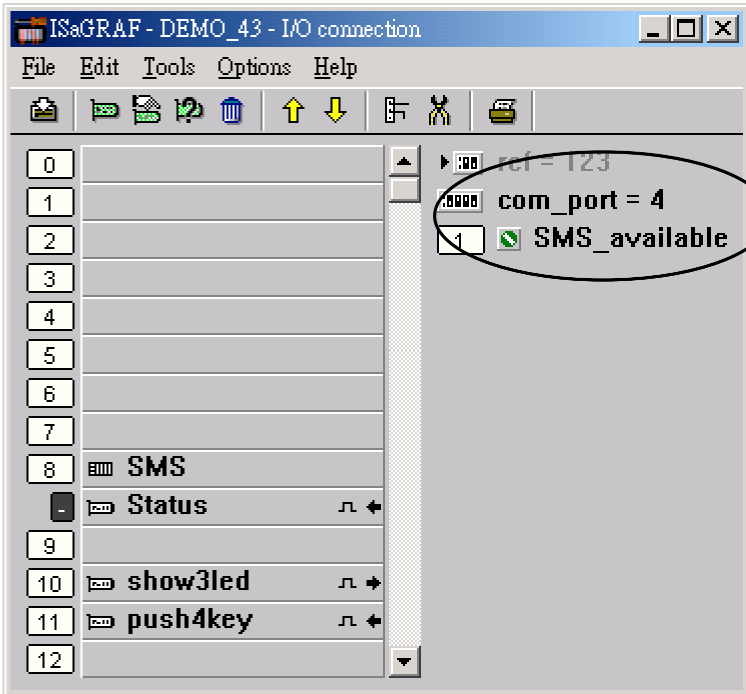
Project architecture :



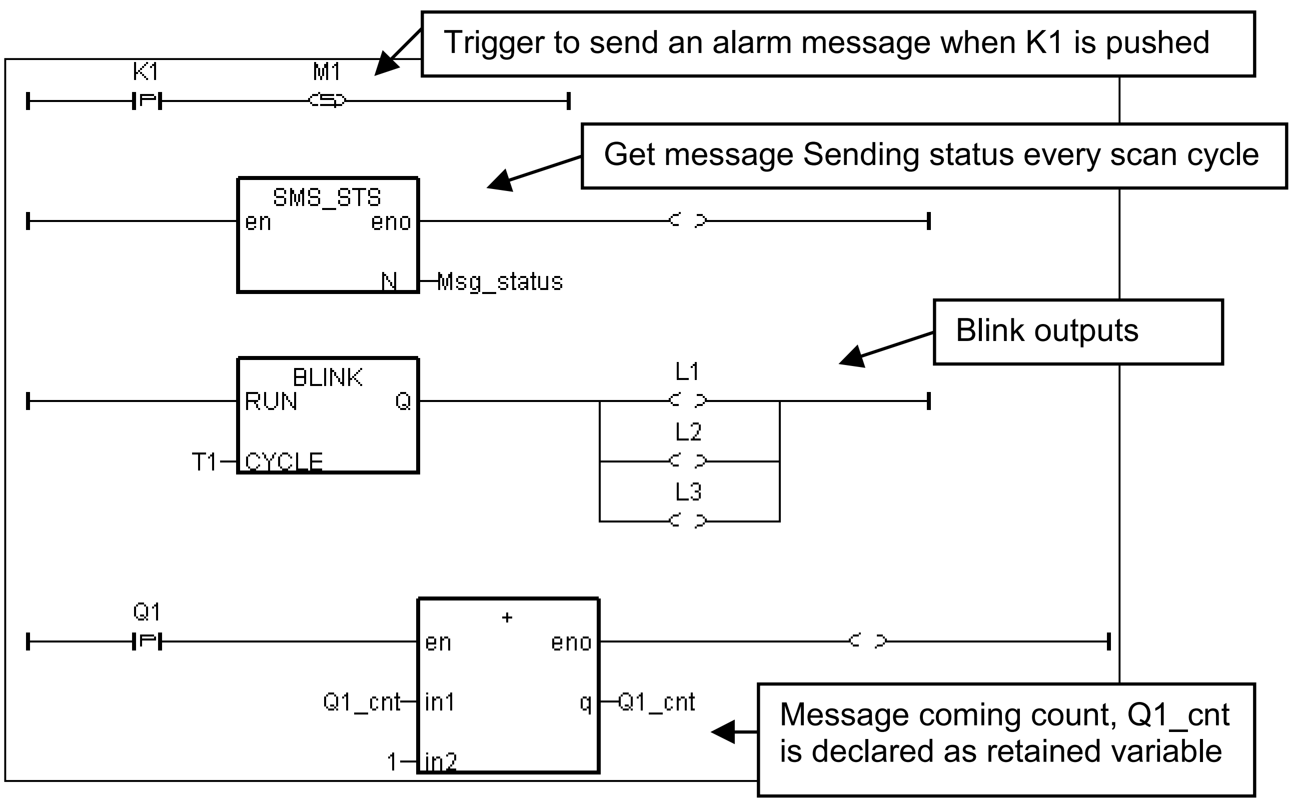
Operation actions:

1. If K1 is pushed, an Alarm message will be sent.
2. If the user send a message in format, for ex. T0200 or T1500 to the controller, the blinking period will change to 200ms and 1500ms. And then the controller will response a report message back to the user.

I/O connection:



LD program : work



ST program : rcv_msg

```
Q1 := SMS_test();
```

Test if a message is coming or not

```
if Q1 then
```

if a message coming

```
  Year1   := SMS_get(1);
  Mon1    := SMS_get(2);
  Day1    := SMS_get(3);
  Wday1   := SMS_get(4);
  Hour1   := SMS_get(5);
  Min1    := SMS_get(6);
  Sec1    := SMS_get(7);
```

call SMS_get to get message coming date & Time in integer format

get phone No. of sender

```
  phone     := SMS_gets(2);
  date_time := SMS_gets(3);
```

get message coming date & time in string format

```
  data      := SMS_gets(1);
```

get message data, SMS_gets(1) should be called in the last one, because it will reset SMS_test status to FALSE:No message coming

Check the coming message. For ex. T1500 will result T1=1500 ms, while T0300 result T1=300ms, however TAB10 will result T1=0 ms (not valid)

```
  if mid(data,1,1) = 'T' then
```

check 1st char is T or not

```
   TMP_v := ANA(mid(data,4,2));
```

extract 4 bytes starting from string position 2, and then convert to an integer

```
  (* valid format *)
   if TMP_v>=50 and TMP_v<=9999 then
    T1 := TMR(TMP_v);    (* convert to timer *)

    Msg_to_send := 'Current T1 change to ' + Msg(TMP_v) + ' ms.';
    M2 := TRUE;
```

Trigger to send a report message to sender

```
   else   (* invalid format*)

    Msg_to_send := '!!! Wrong command, Val should be between T0050 to T9999. Current T1
remains at ' + Msg(Ana(T1)) + ' ms.';
    M2 := TRUE;
```

Trigger to send a report message to sender

```
   end_if;

  end_if;    (* if mid(data,1,1) = 'T' then *)

end_if;   (* if Q1 then *)
```

ST program : snd_msg

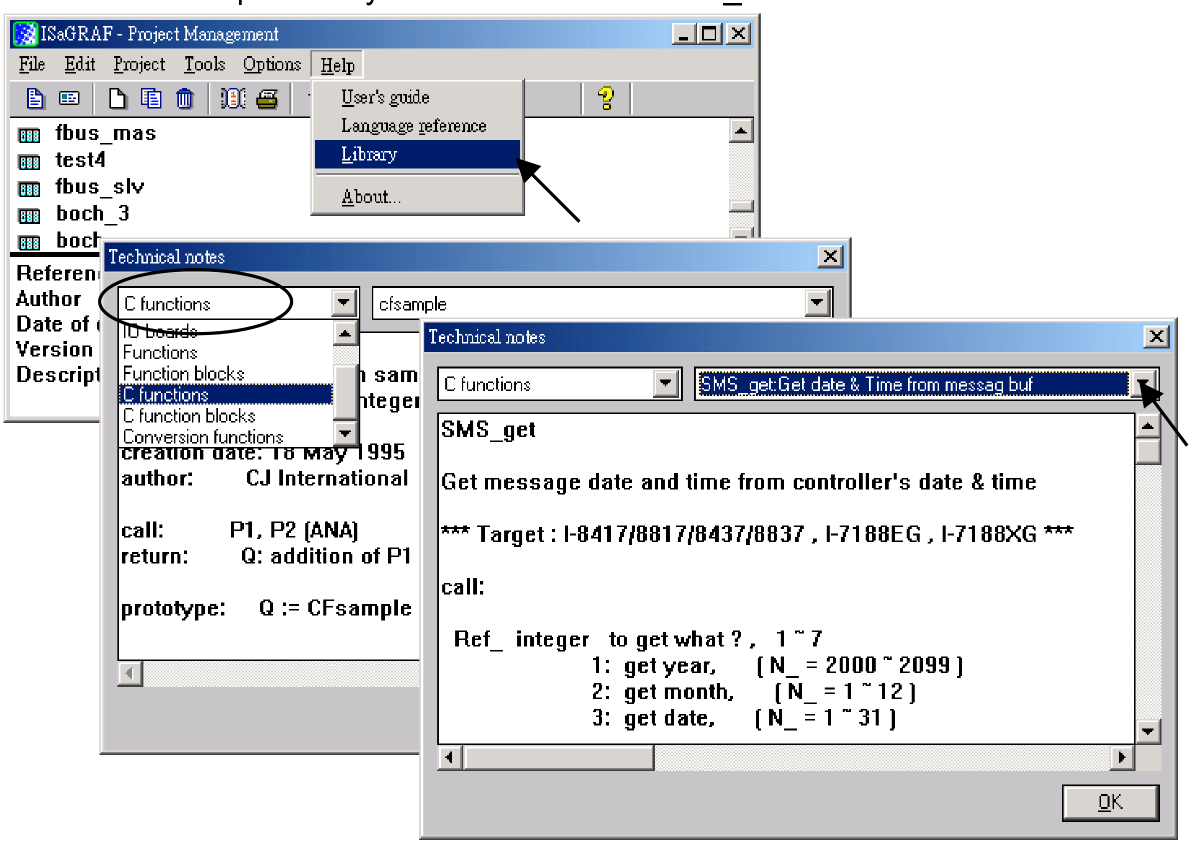
```
if (Msg_status <> 1) and SMS_available then

  if M1 then   (* alarm triggering *)

    TMP := SMS_send(to_who,'K1 is pushed!');
    M1 := FALSE;

  elsif M2 then  (* Report triggering *)

    TMP := SMS_send(phone,Msg_to_send);  (* report message back *)
    M2 := FALSE;

  end_if;

end_if;
```

if message sending status is not 1:busy

Message sending status:
0: waiting for a new sending request
1: busy. (message is processing now)
21: The message is sent successfullly
-1: SMS system is not available
-2: Timeout, No response.

Must disable it (set to FALSE) after SMS_send is called

More description of SMS_sts, SMS_send, SMS_test, SMS_get & SMS_gets, Please refer to ISaGRAF's On-line Help. "Library" – "C functions" – "SMS_xxxx"



Chapter 18 : Motion

18.1: Install motion driver

Limitation:
1. I-8437/8837 **CAN NOT** do ethernet communication when using I-8091 to do motion control, while W-8337/8737 doesn't have this limitation.
2. Only one I-8091 board in I-8417/8817/8437/8837 & W-8337/8737 can do X-Y dependent motion, other I-8091s should be moving independent. Or all I-8091s are moving independent.

The I-8417/8817/8437/8837 & Wincon-8337/8737 can integrate with the I-8091 to do Motion control. The default ISaGRAF driver burned in the Flash memory of the I-8417/8817/8437/8837 controller is for general usage not for motion control. Please update it to the motion driver by yourself. While user don't need to upgrade the driver of Wincon-8337/8737 if its driver version is 3.08 or higher.

The motion driver of I-8417/8817/8437/8837 can be found in the ICP DAS CD-ROM.
napdos\isagraf\8000\driver\motion?.??\
or can be downloaded from
ftp.icpdas.com/pub/cd/8000cd/napdos/isagraf/8000/driver/ motion?.??

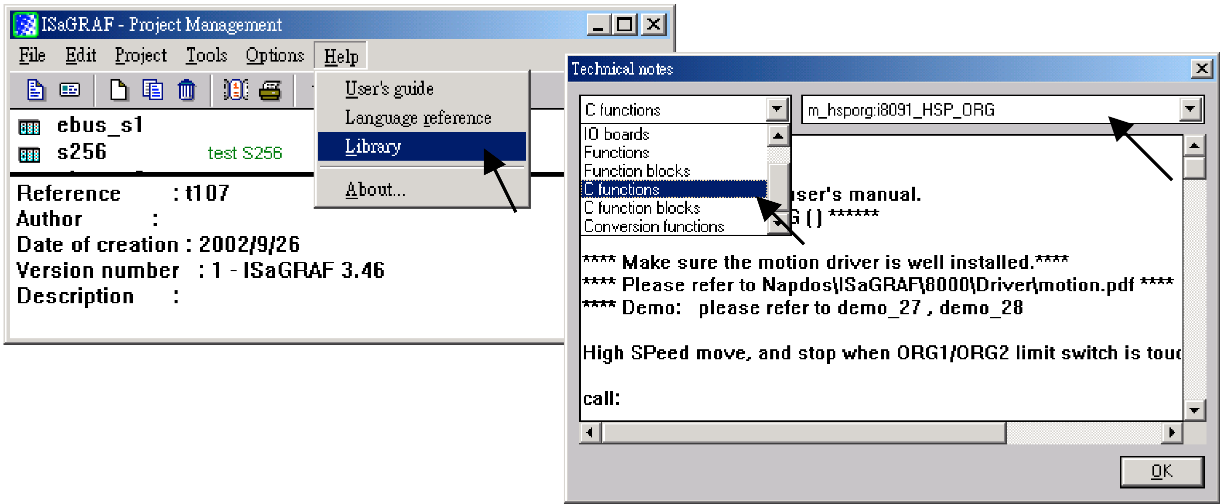
Please refer to the "ReadMe.txt" in the folder of "motion?.??" (for ex. "Motion2.45")

Restriction of the motion driver of I-8417/8817/8437/8837:
The motion driver for I-8417/8817/8437/8837 doesn't support the Ethernet communication, however W-8337/8737 desen't have this limitation.

The ISaGRAF demo projects of motion for I-8417/8817/8437/8837 are "demo_27" , "demo_28", & "demo_46". They are located in the 8000 CD-ROM: napdos\isagraf\8000\demo\" , or from
ftp.icpdas.com/pub/cd/8000cd/napdos/isagraf/8000/demo/

The ISaGRAF demo projects of motion for W-8337/8737 are "wdemo_26" , "wdemo_27", "wdemo_28" & "wdemo_29". They are located in the Wincon CD-ROM:
napdos\isagraf\wincon\demo\" , or from
ftp://ftp.icpdas.com./pub/cd/winconcd/napdos/isagraf/wincon/demo/

All functions that trigger I-8091 & I-8090 are named as ”M_???” , Please refer to the On-line help from the ISaGRAF “Help” – “Library” - “C functions” for names starting with “M_???”.



Beside, please refer to “I-8091 & I-8090 User’s Manual” .It can be found in the package box of the i-8091, or
CD-ROM: napdos\8000\motion\i8091\manual\
ftp site: ftp://ftp.icpdas.com/pub/cd/8000cd/napdos/8000/motion/i8091/manual/

18.2: Introduction

18.2.1: System Block Diagram

The I-8091 stepping motor control card is a micro-computer controlled, 2-axis pulse generation card. It includes a 2Kbytes-FIFO to receive motion command from host, a micro-computer for profile generation and protection, 2-axis DDA chip to execute DDA function when interpolation command is used, 2500Vrms optical isolation inserted for industrial application.

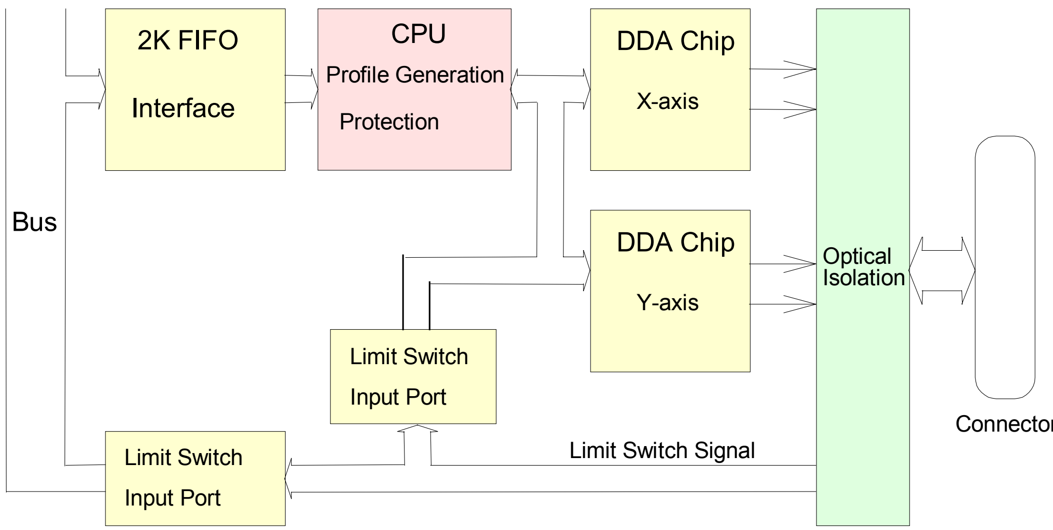


Fig.(1) block diagram of I-8091 card

18.2.2: DDA Technology

The DDA chip is the heart of I-8091 card, it will generate equal-space pulse train corresponding to specific pulse number during a DDA period. This mechanism is very useful to execute pulse generation and interpolation function. The DDA period can be determined by DDA cycle. Table(1) shows the relation among DDA cycle, DDA period and output pulse rate. When DDA cycle set to 1, the DDA period is equal to (1+1)x1.024ms = 2.048ms. The output pulse number can be set to 0~2047, therefore the maximum output pulse rate will be 1Mpps. The minimum output pulse rate is 3.83pps when set DDA cycle=254 (DDA period = (254+1)x1.024ms = 261.12ms).

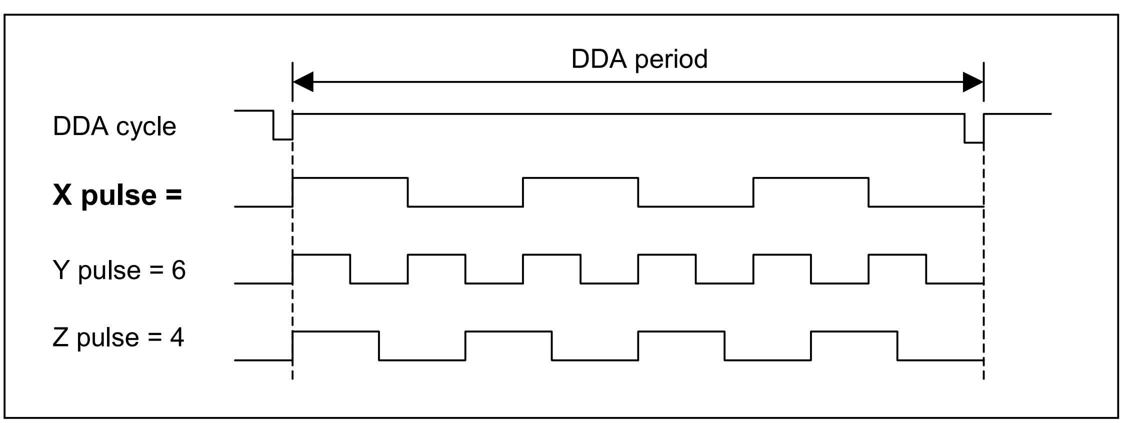


Fig.(2) DDA mechanism

Table(1) The Relation among DDA cycle, DDA period and output pulse rate.

DDA cycle	DDA period	Max. pulse rate(n=2047)	Min. pulse rate (n=1)
1	2.048ms	999511pps	488pps
2	3.072ms	666341pps	325pps
3	4.096ms	.	.
.	.	.	.
N	(N+1)*1.024ms	2047/(DDA period)	1/(DDA period)
.	.	.	.
254	261.12ms	7839pps	3.83pps

The DDA cycle can be set by i8091_SET_VAR() command which decribed in charpter 3. The selection criterion of DDA cycle was described as following.

1. The required max. output pulse rate.

PRmax = Vmax*N/60

PRmax = $\frac{2047}{(DDAcycle+1)*1.024ms}$

PRmax : max. output pulse rate.

Vmax : max. speed (rpm).

N : the pulse number of stepping motor per revolution (pulse/rev).

2. The required speed resolution.
The maximum output pulse number is Np(0~2047), therefore the speed resolution is Vmax(max. speed)/Np. The DDA cycle can be obtained by following equation.

PRmax = $\frac{Np}{(DDAcycle+1)*1.024ms}$

3. When choose large DDA cycle (DDA period), it will occur vibration between different pulse input which generally can be observed during acceleration or deceleration. So, the small DDA cycle , the smooth acceleration/deceleration curve as long as the speed resolution is acceptable.

Example: Stepping Motor
The spec. of stepping motor is 500 pulse/rev, max. speed 500 rpm, speed resolution 2 rpm.

The required max. pulse rate
PRmax = 500 rpm*500/60 = 4166.67 pps

The maximum output pulse
Np = 500rpm/2rpm =250 pulse number

The DDA cycle can be calculated by follow equation

PRmax = $\frac{Np}{(DDAcycle+1)*1.024ms}$

4166.67 = $\frac{250}{(DDAcycle+1)*1.024ms}$

DDA cycle = 58

High Speed = 247 pulse (4166.67*58*0.001024)

The above results means that maximum speed is 500rpm when send command i8091_SET_VAR(0, 58, 2, 2, 247) to I-8091 card.

Example: Pulse type input Servo Motor

The spec. of servo motor is 8000 pulse/rev, max. speed 3000 rpm, speed resolution 2 rpm.

The required max. pulse rate

PRmax = 3000 rpm*8000/60 = 400,000 pps

The maximum output pulse

Np = 3000rpm/2rpm =1500 pulse number

The DDA cycle can be calculated by follow equation

PRmax = $\frac{Np}{(DDAcycle+1)*1.024ms}$

400,000 = $\frac{1500}{(DDAcycle+1)*1.024ms}$

DDA cycle = 3

High Speed = 1638 pulse (400,000*4*0.001024)

The above results means that maximum speed is 3000rpm when send command i8091_SET_VAR(0, 3, 2, 2, 1638) to I-8091 card.

18.3: Hardware

18.3.1: I-8000 hardware address

The hardware address of I-8000 main system is fixed as following table. There are 4 slots I-8000 and 8 slots I-8000.

	Slot 0	Slot 1	Slot 2	Slot 3	Slot 4	Slot 5	Slot 6	Slot 7
I-8000, 4 slot address	0x080	0x0A0	0x0C0	0x0E0	---	---	---	---
I-8000, 8 slot address	0x080	0x0A0	0x0C0	0x0E0	0x140	0x160	0x180	0x1A0

Fig.(3) I-8000 hardware address

18.3.2: LED Indicator

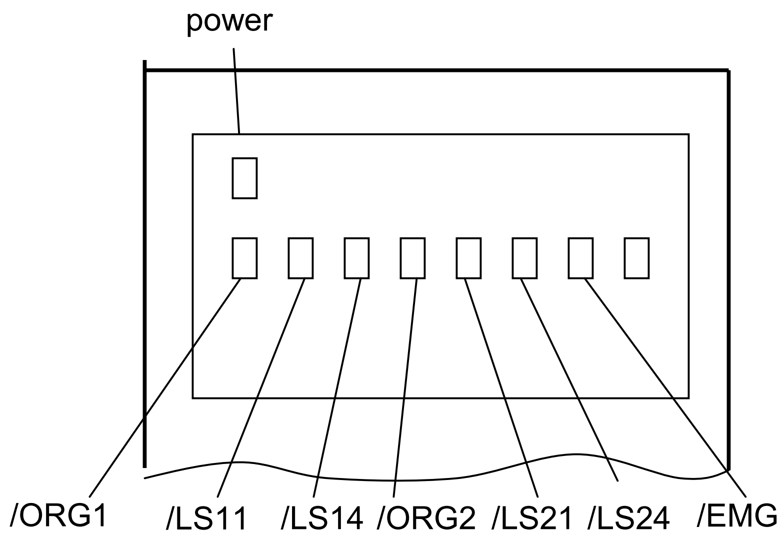


Fig.(4) I-8091 LED indicator

/ORG1: X-axis's original limit switch for machine home position.
/LS11, /LS14 : X-axis's negative and positive limit switches.
/ORG2: Y-axis's original limit switch for machine home position.
/LS21, /LS24 : Y-axis's negative and positive limit switches.
/EMG : system's emergency signal input.

18.3.3: Hardware Configuration

Limit switch configuration

Because the profile generation and protection is executed by the CPU on I-8091 card, the limit switches must configure as following diagram. The motion command just can work properly.

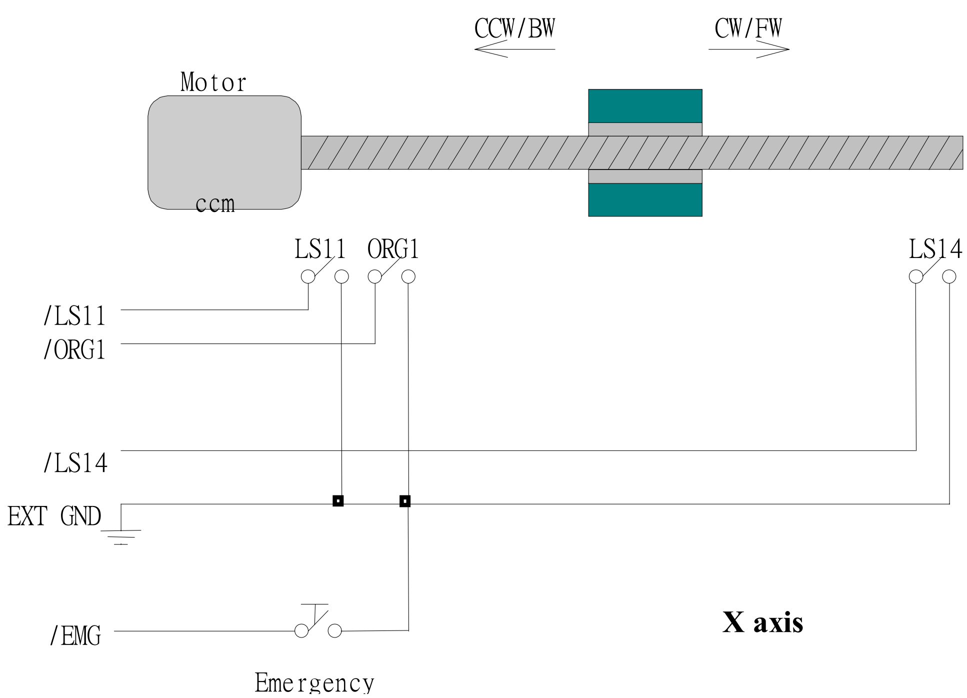


Fig.(5) Limit switch configuration of X axis

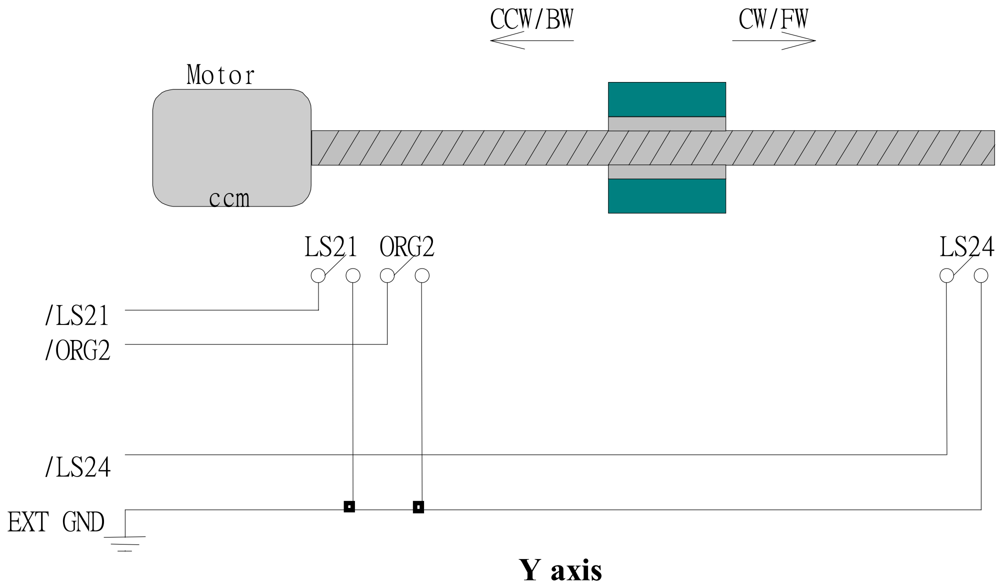


Fig.(6) Limit switch configuration of Y axis

Output pulse mode configuration

I-8091 card provide two kind output method.

(a) CW/CCW mode
(b) Pulse/Direction mode

The command **M_s_mode(card_NO_, modeX_, modeY_)** provide parameters 0: CW_CCW and 1: PULSE_DIR to define output pulse mode.

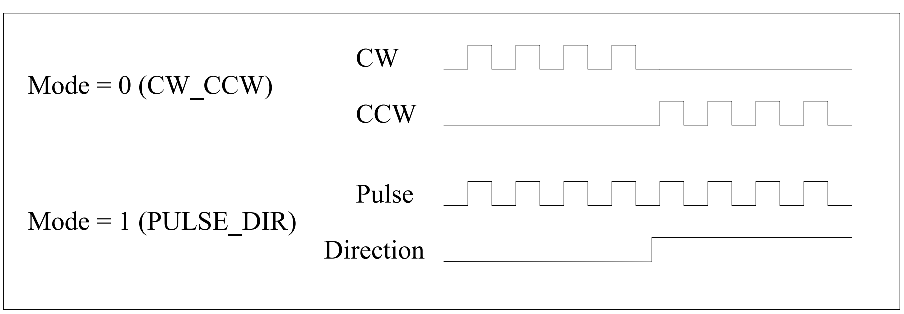


Fig.(7) Output pulse mode

Direction configuration

Sometimes, the output direction of X-axis, Y-axis is not in the desired direction due to the motor's connection or gear train. It is recommended to unify the output direction as shown in Figure(5)(6). The CW/FW direction is defined as toward outside from motor and the CCW/BW direction is defined as toward inside to motor. The **M_s_dir(card_NO_, defdirX_, defdirY_)** command provides parameters 0: NORMAL_DIR and 1:REVERSE_DIR to define the rotating direction of motor.

Turn Servo ON/OFF (Hold ON/OFF)

To turn servo motor into servo ON(OFF) state, or turn stepping motor into hold ON(OFF) state, the command **M_s_serv(card_NO_, sonX_, sonY_)** provide parameters 1:ON and 0:OFF to turn ON or OFF.

Automatic protection

The I-8091 card has a automatic protected system.

(a) If X-aixs command is executing and moving toward CW/FW direction, X-axis will immediately stop when LS14 is touched. To release this protection as long as X-axis move toward CCW/BW direction.

(b) If X-aixs command is executing and moving toward CCW/BW direction, X-axis will immediately stop when LS11 is touched. To release this protection as long as X-axis move toward CW/FW direction.

(c) If Y-aixs command is executing and moving toward CW/FW direction, Y-axis will immediately stop when LS24 is touched. To release this protection as long as Y-axis move toward CCW/BW direction.

(d) If Y-aixs command is executing and moving toward CCW/BW direction, Y-axis will immediately stop when LS21 is touched. To release this protection, as long as Y-axis move toward CW/FW direction.

(e) If the signal of the emergency limit switch /EMG was found in CPU firmware, all motion will be terminated and stop.

Set limit switch as normal close condition

The limit switches /EMG, /LS11, /LS14, /LS21, /LS24, /ORG1, /ORG2 is initially normal open condition, that is, these signal is active when connect it to ground. In industrial application, it might be recommended normal close condition, that is, these signal is active when open from ground.
The **M_s_nc(card_NO_, sw_)** command can be set sw=0 (default), for normal open condition. When set sw=1, for normal close condition.

18.3.4: Pin assignment of connector CN2

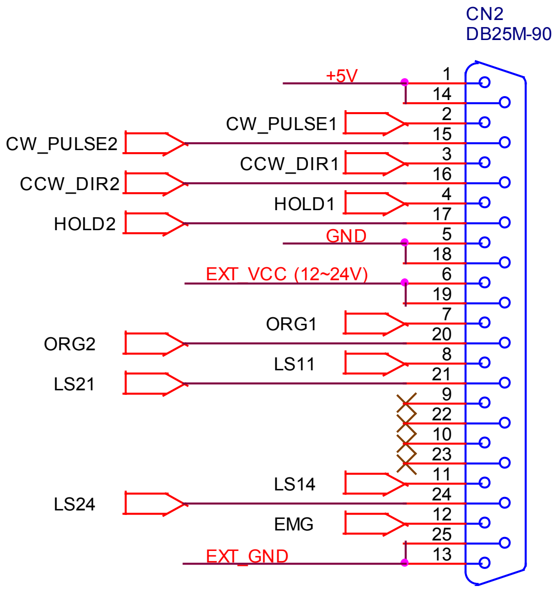


Fig.(8) CN2 connector of I-8091

Table of CN2 connector's pin assignment

pin name	pin number	Description
+5V	1	Internal +5V power, Max. output current: 50mA
CW_PULSE1	2	X-axis CW (Pulse) output pin
CCW_DIR1	3	X-axis CCW (Direction) output pin
HOLD1	4	X-axis HOLD (servo on) output pin
GND	5	Signal ground of pin 2,3,4
EXT_VCC	6	External power(12~24V) for limit switches
/ORG1	7	X-axis original (home) limit switch
/LS11	8	X-axis limit switch
	9,10	No used
/LS14	11	X-axis limit switch
/EMG	12	Emergency input
EXT_GND	13	External ground for limit switch
+5V	14	Internal +5V power, Max. output current: 50mA
CW_PULSE2	15	Y-axis CW (Pulse) output pin
CCW_DIR2	16	Y-axis CCW (Direction) output pin
HOLD2	17	Y-axis HOLD (servo on) output pin
GND	18	Signal ground of pin 15,16,17
EXT_VCC	19	External power(12~24V) for limit switches
/ORG2	20	Y-axis original (home) limit switch
/LS21	21	Y-axis limit switch
	22,23	No used
/LS24	24	Y-axis limit switch
EXT_GND	25	External ground for limit switch

The internal circuit of CW_PULSE, CCW_DIR, HOLD

When output these signal as 1, it can source 15mA(max.).
When output these signal as 0, it can sink 50mA(max.)

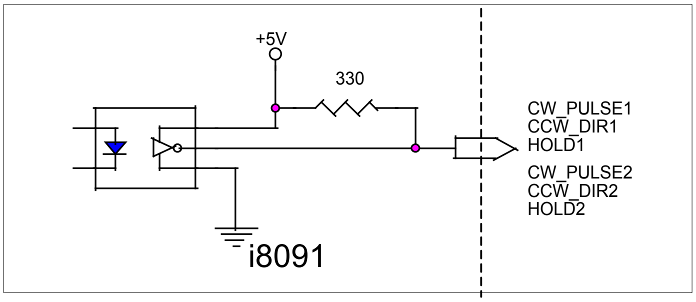


Fig.(9) internal circuit of pulse output pin

The internal circuit of limit switch input

Initially, the limit switch inputs of I-8091 board are normal open (N.O.), the I-8091 board will automatic protect when limit switch pin connect to EXT_GND. The user can use the command **M_s_nc(card_NO_, 1)** to let those limit switch input as normal close condition at the beginning of the user's program.

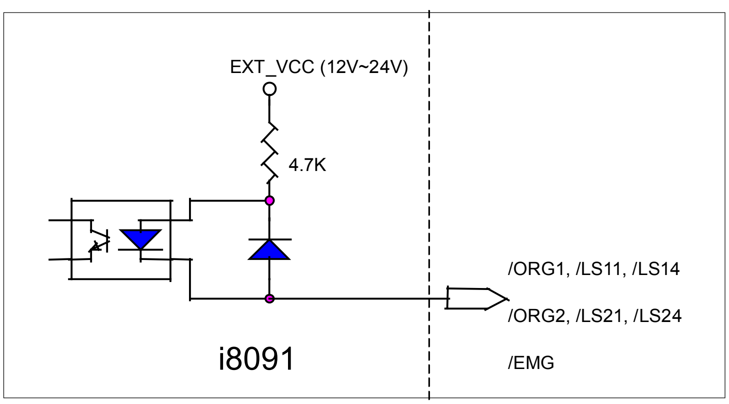


Fig.(10) internal circuit of limit switch input pin

Example of connection

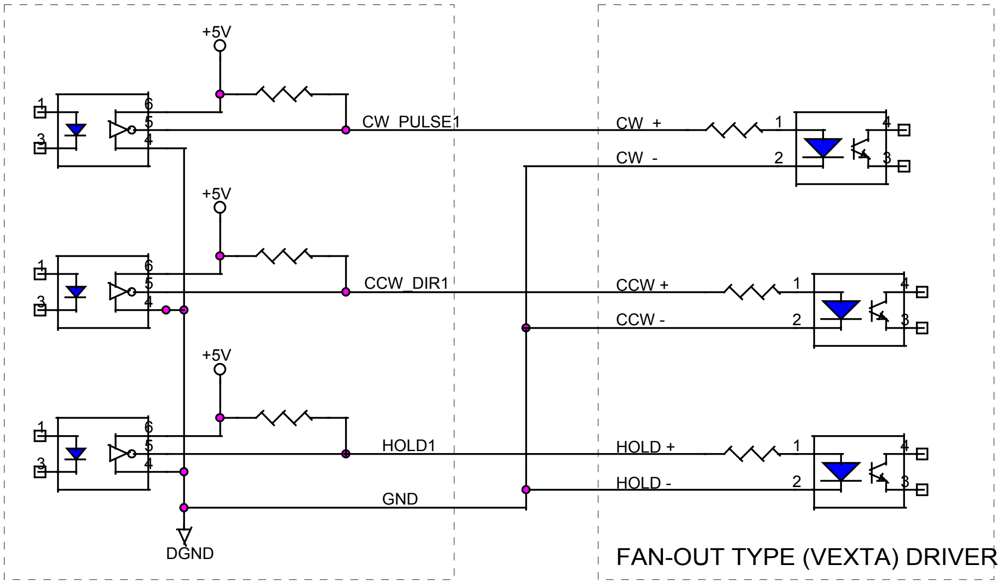


Fig.(11) fan-out type driver (VEXTA's motor driver)

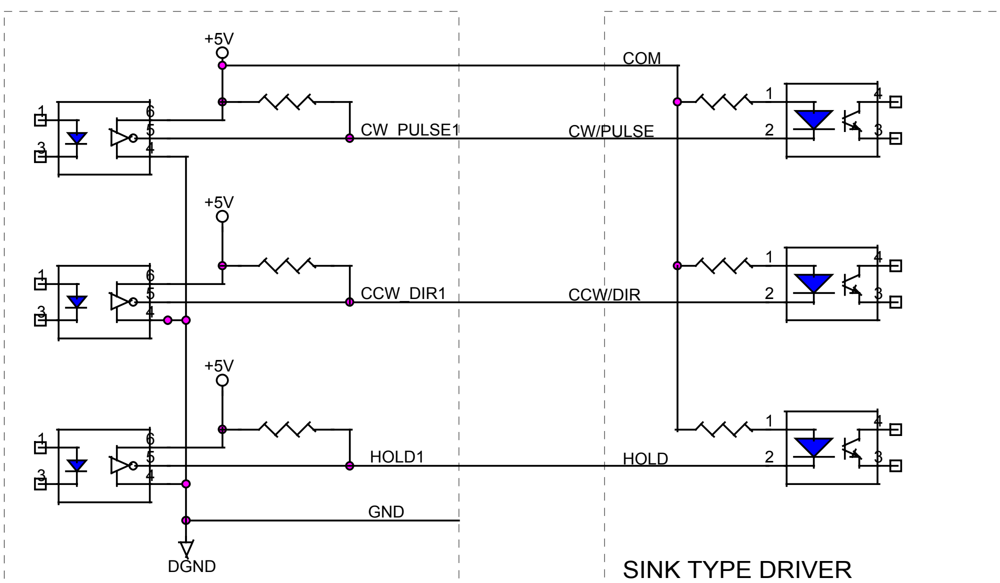


Fig.(12) Sink type driver

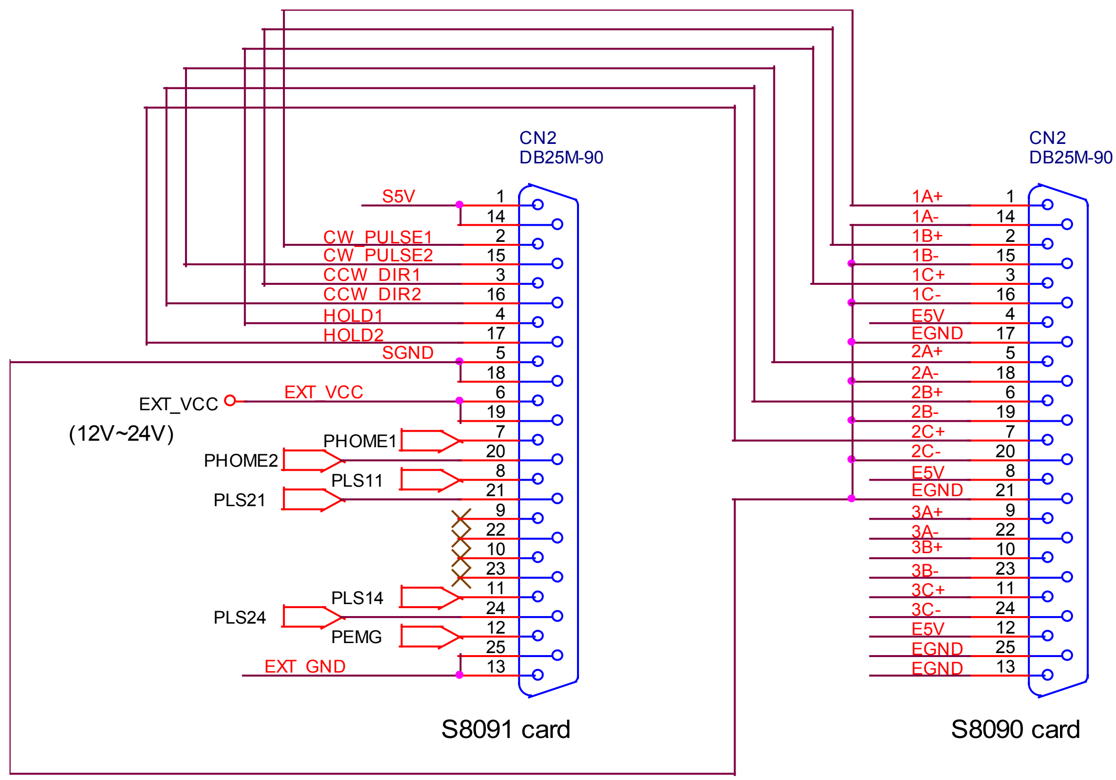
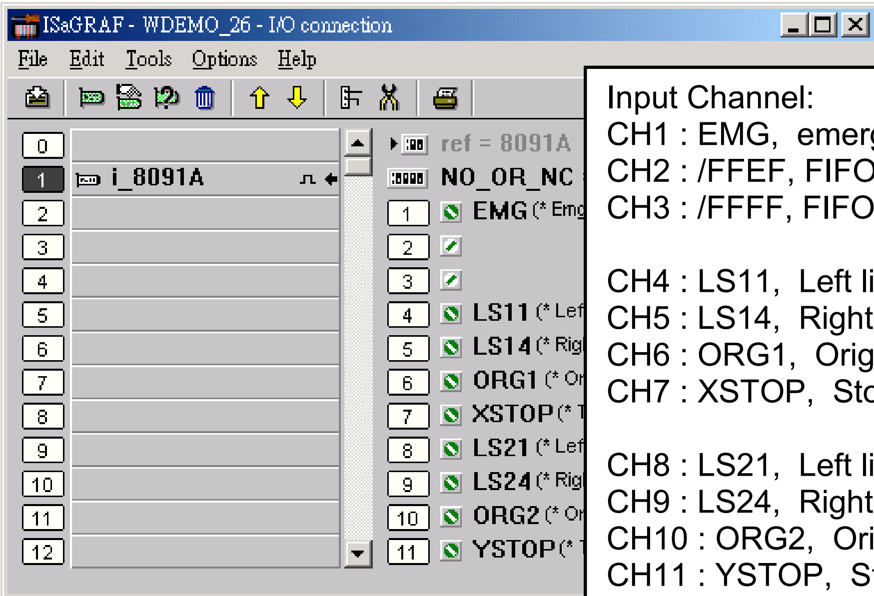


Fig.(13) The connection between I-8090 and I-8091 for function testing or pulse feedback by I-8090 encoder card.

18.4: Software

I/O connection:

The “**I-8091A**” connectted on the I/O connection window contains 11 digital input channels.

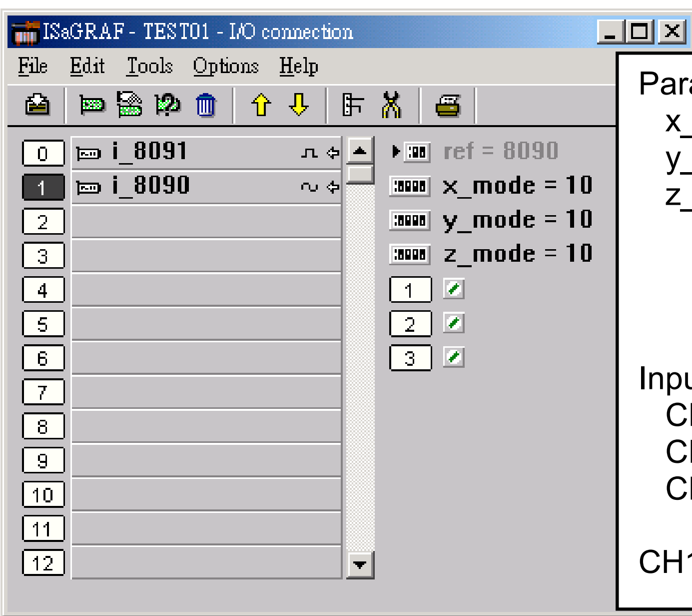


Input Channel:
CH1 : EMG, emergency stop
CH2 : /FFEF, FIFO is empty or not, TRUE: empty
CH3 : /FFFF, FIFO is full or not, TRUE: full

CH4 : LS11, Left limit swtch of X-axis
CH5 : LS14, Right limit swtch of X-axis
CH6 : ORG1, Original position swtch of X-axis
CH7 : XSTOP, Stop or not of X-axis, TRUE: stop

CH8 : LS21, Left limit swtch of Y-axis
CH9 : LS24, Right limit swtch of Y-axis
CH10 : ORG2, Original position swtch of Y-axis
CH11 : YSTOP, Stop or not of Y-axis, TRUE: stop

I-8090 contains 3 analog input channels.



Parameter:
x_mode : integer counting mode of X-axis
y_mode : integer counting mode of Y-axis
z_mode : integer counting mode of Z-axis
00: quadrant counting mode
10: CW/CCW counting mode
20: pulse/direction counting mode

Input Channel:
CH1 : encorder value of X-axis
CH2 : encorder value of Y-axis
CH3 : encorder value of Z-axis

CH1 to CH3 are signed 32-bit integer format

Setting commands:

M_regist Register one I-8091

In order to distinguish more than one I-8091 card in I-8417/8817/8437/8837 platform, the I-8091 cards should be registrated before using it. This command will assign a card number = “card_NO_” to I-8091 card at that “address_” . If there is no I-8091 at the given address, this command will return FALSE.

Note: If using “I_8091A” rather than “I_8091” on the I/O connection window, user don’t need to call “m_regist” & “m_s_nc”, they are ignored. The card_NO of “I-8091A” is equal to its slot No. I-8xx7: 0 ~ 7. W-8xx7: 1 ~ 7.

Parameters:

card_NO_	integer	valid is 0 ~ 19.
address_	integer	the plugged slot address of the i8091 card
		slot 0: 16#80
		slot 1: 16#A0
		slot 2: 16#C0
		slot 3: 16#E0
		slot 4: 16#140
		slot 5: 16#160
		slot 6: 16#180
		slot 7: 16#1A0

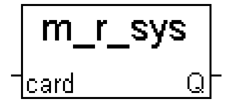
Return:

Q_	boolean	TRUE: Ok , FALSE: Fail

Example: I-8417/8817/8437/8837: demo_46, demo_27, demo_28
W-8337/8737: wdemo_26, wdemo_27, wdemo_28, wdemo_29

```
 (* declaration:   INIT as boolean <internal> and has initial value of TRUE       *)
(*  TMP as boolean <internal>                                                  *)
 (* cardNO as integer <internal> and has intial value of 1  *)
(* Do some init setting at 1st scan cycle *)
if INIT then
        INIT := FALSE;
        TMP := M_regist(cardNO,16#80);       (* plug i8091 in slot 0 *)
        TMP := M_r_sys(cardNO);                    (* reset i8091's setting *)
        TMP := M_s_var(cardNO,4,2,5,100);
        TMP := M_s_dir(cardNO,0,0);                (* Normal direction *)
        TMP := M_s_mode(cardNO,1,1);         (* pulse_dir mode *)
        TMP := M_s_serv(cardNO,1,1);               (* X & Y server ON *)
        TMP := M_s_nc(cardNO,0);             (* Normal open *)
end_if;
```

M_r_sys Reset all setting

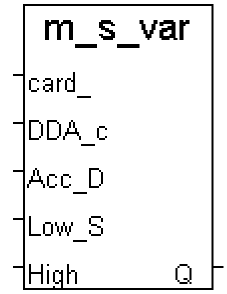


To reset I-8091 card, this command will terminate the running command in I-8091 card. User can use this command as software emergency stop. This command also will clear all of setting, so, all I-8091 card's parameter should be set again.

Parameters:
card_NO_ integer the card No. has been set by **M_regist**, valid is 0 ~ 19
Return:
Q_ boolean always return TRUE.

Example: I-8417/8817/8437/8837: demo_46, demo_27, demo_28
W-8337/8737: wdemo_26, wdemo_27, wdemo_28, wdemo_29

M_s_var Set motion system parameters



To set DDA cycle, accelerating/decelerating speed, low speed and high speed value.

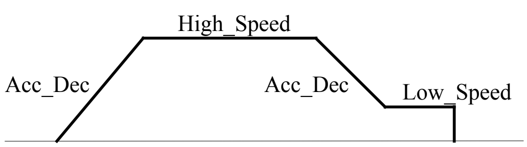
Parameters:

card_NO_	integer	the card No. has been set by **M_regist**, valid is 0 ~ 19
DDA_cycle_	integer	DDA cycle , valid is 1 ~ 254
Acc_Dec_	integer	Acc/Dec speed , valid is 1 ~ 200
Low_Speed_	integer	low speed , valid is 1 ~ 200 , Low_Speed_ >= Acc_Dec_
High_Speed_	integer	high speed , Low_Speed_ <= High_Speed <= 2047

Return:

Q_	boolean	always return TRUE.

Note:
The lower "DDA_cycle_" is given, the smaller delay time between /ORG1 ON and /X_STOP ON (or /ORG2 ON and /Y_STOP ON) when using M_hsporg & M_lsporg command. For ex, DDA_cycle_ set to 4, the delay time is about 5 to 13 ms.



Restriction:

$$1 \le DDA\_cycle \le 254$$
$$1 \le Acc\_Dec \le 200$$
$$1 \le Low\_Speed \le 200$$
$$Low\_Speed \le High\_Speed \le 2047$$

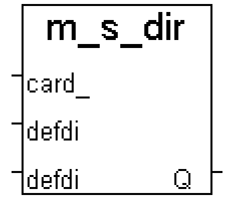
Low_Speed >= Acc_Dec

Default value
- DDA_cycle = 10
- Acc_Dec = 1
- Low_Speed = 10
- High_Speed = 100

Example: I-8417/8817/8437/8837: demo_46, demo_27, demo_28
W-8337/8737: wdemo_26, wdemo_27, wdemo_28, wdemo_29

```
TMP := M_s_var(1, 5, 2, 10, 150);
(* DDA_cycle = 5        --> DDA period = (5+1)*1.024ms = 6.144ms
   Acc_Dec = 2          --> Acc/Dec speed = 2/(6.144ms)^2 = 52981 p/s^2
  Low_Speed = 10        --> low speed = 10/6.144ms = 1628pps
  High_Speed = 150      --> high speed = 150/6.144ms = 24414pps *)
```

M_s_dir Define output direction of axes



Sometimes, the output direction of X-axis, Y-axis is undesired direction due to the motor's connection or gear train. In order to unify the output direction as shown in Fig.(5) and Fig.(6). Where CW/FW direction is defined as toward outside from motor, CCW/BW direction is defined as toward inside from motor. This command provide parameters to define the rotating direction of motor.

Parameters:

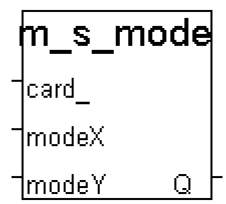
card_NO_	integer	the card No. has been set by **M_regist**, valid is 0 ~ 19
defdirX_	integer	X axis direction definition , valid is 0 ~ 1
defdirY_	integer	Y axis direction definition , valid is 0 ~ 1
		0: normal direction, 1: reverse direction

Return:

Q_	boolean	always return TRUE.

Example: I-8417/8817/8437/8837: demo_46, demo_27, demo_28
W-8337/8737: wdemo_26, wdemo_27, wdemo_28, wdemo_29

M_s_mode Set output mode

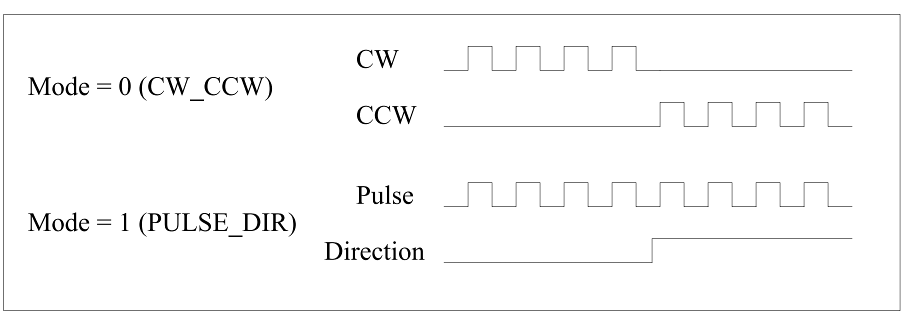


Parameters:

card_NO_	integer	the card No. has been set by **M_regist**, valid is 0 ~ 19
modeX_	integer	X axis mode, valid is 0 ~ 1
modeY_	integer	Y axis mode, valid is 0 ~ 1
		0: CW_CCW, 1: PULSE_DIR

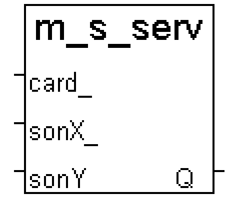
Return:

Q_	boolean	always return TRUE.



Example: I-8417/8817/8437/8837: demo_46, demo_27, demo_28
W-8337/8737: wdemo_26, wdemo_27, wdemo_28, wdemo_29

M_s_serv Set servo ON/OFF



Parameters:

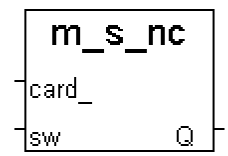
card_NO_	integer	the card No. has been set by **M_regist**, valid is 0 ~ 19
sonX_	integer	X axis servo/hold on switch , valid is 0 ~ 1
sonY_	integer	Y axis servo/hold on switch , valid is 0 ~ 1 0: OFF, 1: ON

Return:

Q_	boolean	always return TRUE.

Example: I-8417/8817/8437/8837: demo_46, demo_27, demo_28
W-8337/8737: wdemo_26, wdemo_27, wdemo_28, wdemo_29

M_s_nc Set N.O. / N.C.



To set all of the following limit switches as N.C.(normal close) or N.O.(normall open). If set as N.O., those limit switches are active low. If set as N.C., those limit switches are active high. The auto-protection will automatically change the judgement whatever it is N.O. or N.C..

Limit switches: ORG1, LS11, LS14, ORG2, LS21, LS24, EMG.

Note: If using "I_8091A" rather than "I_8091" on the I/O connection window, user don't need to call "m_regist" & "m_s_nc", they are ignored. The card_NO of "I-8091A" is equal to its slot No. I-8xx7: 0 ~ 7. W-8xx7: 1 ~ 7.

Parameters:

card_NO_	integer	the card No. has been set by **M_regist**, valid is 0 ~ 19
sw_	integer	0: N.O. (default) , 1: N.C.

Return:

Q_	boolean	always return TRUE.

Example: I-8417/8817/8437/8837: demo_46, demo_27, demo_28
W-8337/8737: wdemo_26, wdemo_27, wdemo_28, wdemo_29

Stop commands:

M_stpx Stop X axis

Parameters:
card_NO_ integer the card No. has been set by **M_regist**, valid is 0 ~ 19

Return:
Q_ boolean always return TRUE.

Example: I-8417/8817/8437/8837: demo_46, demo_27, demo_28
W-8337/8737: wdemo_26, wdemo_27, wdemo_28, wdemo_29

M_stpy Stop Y axis

Parameters:
card_NO_ integer the card No. has been set by **M_regist**, valid is 0 ~ 19

Return:
Q_ boolean always return TRUE.

Example: I-8417/8817/8437/8837: demo_46, demo_27, demo_28
W-8337/8737: wdemo_26, wdemo_27, wdemo_28, wdemo_29

M_stpall Stop X & Y axes

This command will stop X & Y axes and clear all of commands pending in the FIFO.

Parameters:
card_NO_ integer the card No. has been set by **M_regist**, valid is 0 ~ 19

Return:
Q_ boolean always return TRUE.

Example: I-8417/8817/8437/8837: demo_46, demo_27, demo_28
W-8337/8737: wdemo_26, wdemo_27, wdemo_28, wdemo_29

Simple motion commands:

M_lsporg   Low speed move to ORG

Low speed move , and stop when **ORG1/ORG2** limit switch is touched.

Parameters:

card_NO_	integer	the card No. has been set by **M_regist**, valid is 0 ~ 19
DIR_	integer	0: CW , 1: CCW
AXIS_	integer	1: X axis , 2: Y axis

Return:

Q_	boolean	always return TRUE.

M_hsporg   High speed move to ORG

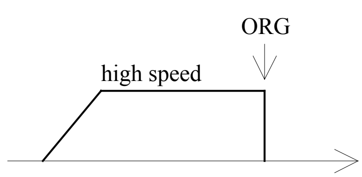
High speed move , and stop when **ORG1/ORG2** limit switch is touched.

Parameters:

card_NO_	integer	the card No. has been set by **M_regist**, valid is 0 ~ 19
DIR_	integer	0: CW , 1: CCW
AXIS_	integer	1: X axis , 2: Y axis

Return:

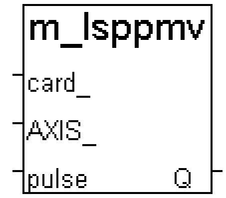
Q_	boolean	always return TRUE.



Example: I-8417/8817/8437/8837: demo_46, demo_27, demo_28
W-8337/8737: wdemo_26, wdemo_27, wdemo_28, wdemo_29

Note:
The lower “DDA_cycle_” is given, the smaller delay time between /ORG1 ON and /X_STOP ON (or /ORG2 ON and /Y_STOP ON) when using M_hsporg & M_lsporg command. For ex, DDA_cycle_ set to 4, the delay time is about 5 to 13 ms.

M_lsppmv Low speed pulse move



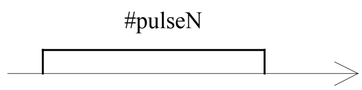
Low speed move a specified “pulse”

Parameters:

card_NO_	integer	the card No. has been set by **M_regist**, valid is 0 ~ 19
AXIS_	integer	1: X axis , 2: Y axis
Pulse_	integer	number of pulse to move. if > 0, move toward CW/FW dir. if < 0, move toward CCW/BW dir.

Return:

Q_	boolean	always return TRUE.



Example: I-8417/8817/8437/8837: demo_46, demo_27, demo_28
W-8337/8737: wdemo_26, wdemo_27, wdemo_28, wdemo_29

M_hsppmv High speed pulse move

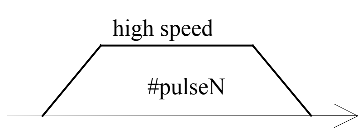
High speed move a specified “pulse”

Parameters:

card_NO_	integer	the card No. has been set by **M_regist**, valid is 0 ~ 19
AXIS_	integer	1: X axis , 2: Y axis
Pulse_	integer	number of pulse to move. if > 0, move toward CW/FW dir. if < 0, move toward CCW/BW dir.

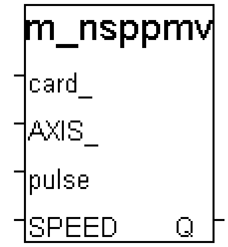
Return:

Q_	boolean	always return TRUE.



Example: I-8417/8817/8437/8837: demo_46, demo_27, demo_28
W-8337/8737: wdemo_26, wdemo_27, wdemo_28, wdemo_29

M_nsppmv Normal speed pulse move



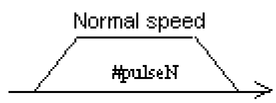
Normal speed move a specified “pulse”

Parameters:

card_NO_	integer	the card No. has been set by **M_regist**, valid is 0 ~ 19
AXIS_	integer	1: X axis , 2: Y axis
Pulse_	integer	number of pulse to move. if > 0, move toward CW/FW dir. if < 0, move toward CCW/BW dir.
SPEED_	integer	Speed, low speed <= SPEED_ <= high speed

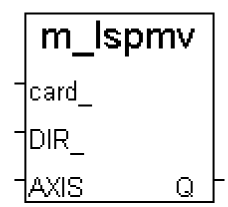
Return:

Q_	boolean	always return TRUE.



Example: I-8417/8817/8437/8837: demo_46, demo_27, demo_28
W-8337/8737: wdemo_26, wdemo_27, wdemo_28, wdemo_29

M_lspmv Low speed move



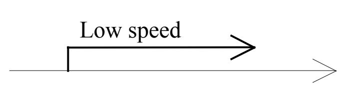
Low speed move toward the direction specified. It can be stop by **M_stpx** or **M_stpy** or **M_stpall** command

Parameters:

card_NO_	integer	the card No. has been set by **M_regist**, valid is 0 ~ 19
DIR_	integer	direction. 0: CW , 1: CCW
AXIS_	integer	1: X axis , 2: Y axis

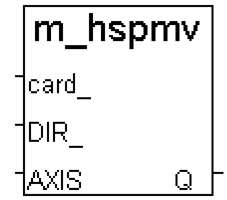
Return:

Q_	boolean	always return TRUE.



Example: I-8417/8817/8437/8837: demo_46, demo_27, demo_28
W-8337/8737: wdemo_26, wdemo_27, wdemo_28, wdemo_29

M_hspmv High speed move



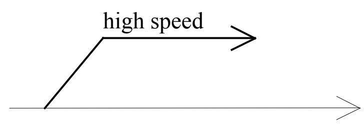
High speed move toward the direction specified. It can be stop by **M_stpx** or **M_stpy** or **M_stpall** command

Parameters:

card_NO_	integer	the card No. has been set by **M_regist**, valid is 0 ~ 19
DIR_	integer	direction. 0: CW , 1: CCW
AXIS_	integer	1: X axis , 2: Y axis

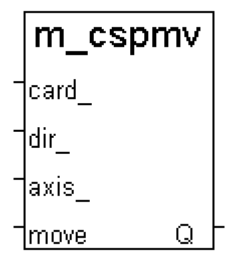
Return:

Q_	boolean	always return TRUE.



Example: I-8417/8817/8437/8837: demo_46, demo_27, demo_28
W-8337/8737: wdemo_26, wdemo_27, wdemo_28, wdemo_29

M_cspmv Change speed move



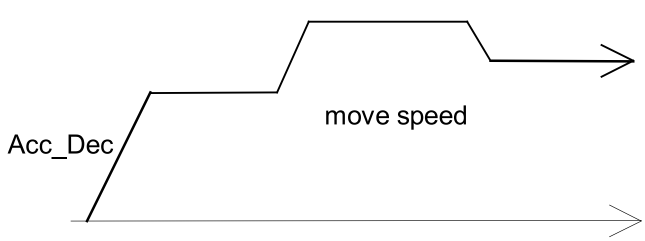
This command will accelerate/decelerate the selected axis's motor to the "move_speed". This command can be continuously send to I-8091 to dynamicly change speed. The rotating motor can be stop by the command **M_stpx**, **M_stpy**, **M_stpall**, or **M_slwstp**

Parameters:

card_NO_	integer	the card No. has been set by **M_regist**, valid is 0 ~ 19
dir_	integer	direction. 0: CW , 1: CCW
axis_	integer	1: X axis , 2: Y axis
move_speed_	integer	0 < move_speed_ <= 2040

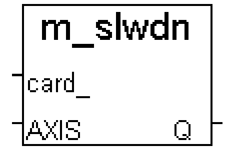
Return:

Q_	boolean	always return TRUE.



Example: I-8417/8817/8437/8837: demo_46, demo_27, demo_28
W-8337/8737: wdemo_26, wdemo_27, wdemo_28, wdemo_29

M_slwdn Slow down to low speed



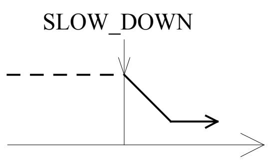
To decelerate to slow speed until **M_stpx** or **M_stpy** or **M_stpall** is executed.

Parameters:

card_NO_	integer	the card No. has been set by **M_regist**, valid is 0 ~ 19
AXIS_	integer	1: X axis , 2: Y axis

Return:

Q_	boolean	always return TRUE.



Example: I-8417/8817/8437/8837: demo_46, demo_27, demo_28
W-8337/8737: wdemo_26, wdemo_27, wdemo_28, wdemo_29

M_slwstp Slow down to stop

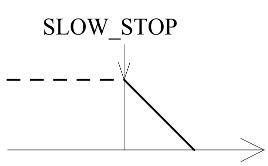
To decelerate to stop.

Parameters:

card_NO_	integer	the card No. has been set by **M_regist**, valid is 0 ~ 19
AXIS_	integer	1: X axis , 2: Y axis

Return:

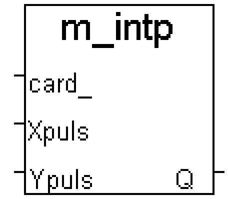
Q_	boolean	always return TRUE.



Example: I-8417/8817/8437/8837: demo_46, demo_27, demo_28
W-8337/8737: wdemo_26, wdemo_27, wdemo_28, wdemo_29

Interpolation commands:

M_intp Move a short distance on X-Y plane



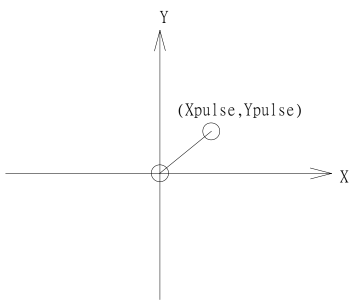
This command will move a short distance (interpolation short line) on X-Y plane. This command provided a method for user to generate an arbitrary curve on X-Y plane.

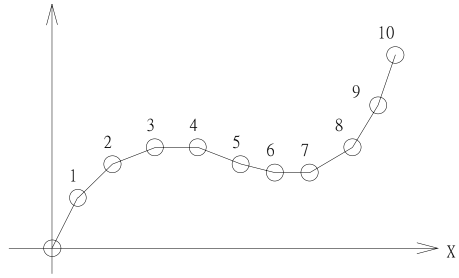
Parameters:

card_NO_	integer	the card No. has been set by **M_regist**, valid is 0 ~ 19
Xpulse_	integer	-2047 <= Xpulse_ <= 2047
Ypulse_	integer	-2047 <= Ypulse_ <= 2047

Return:

Q_	boolean	always return TRUE.





Example: I-8417/8817/8437/8837: demo_46, demo_27, demo_28
W-8337/8737: wdemo_26, wdemo_27, wdemo_28, wdemo_29

NOTE:
For a lot of **M_intp** call set at the same time, please check if the FIFO is not full. Call it if FIFO is not full. FIFO indicator is a Digital Input resides at CH3 of i-8091.

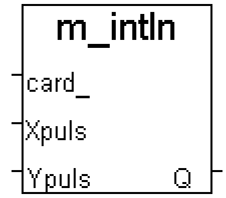
i-8091 D/I channel on ISaGRAF I/O connection window:

CH1 : EMG, emergency stop
CH2 : /FFEF, FIFO is empty or not, TRUE: empty
CH3 : /FFFF, FIFO is full or not, TRUE: full

CH4 : LS11, Left limit swtch of X-axis
CH5 : LS14, Right limit swtch of X-axis
CH6 : ORG1, Original position swtch of X-axis
CH7 : XSTOP, Stop or not of X-axis, TRUE: stop

CH8 : LS21, Left limit swtch of Y-axis
CH9 : LS24, Right limit swtch of Y-axis
CH10 : ORG2, Original position swtch of Y-axis
CH11 : YSTOP, Stop or not of Y-axis, TRUE: stop

M_intln Move a long distance on X-Y plane



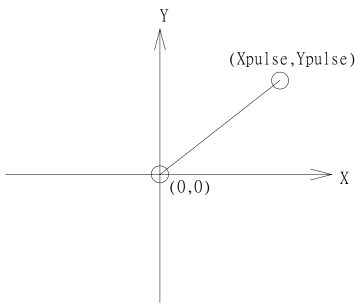
This command will move a long distance (interpolation line) on X-Y plane. The CPU on I-8091 card will generate a trapezoidal speed profile of X-axis and Y-axis, and execute interpolation by way of DDA chip.

Parameters:

card_NO_	integer	the card No. has been set by **M_regist**, valid is 0 ~ 19
Xpulse_	integer	-524287 <= Xpulse_ <= 524287
Ypulse_	integer	-524287 <= Xpulse_ <= 524287

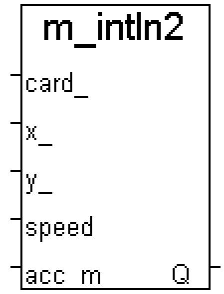
Return:

Q_	boolean	always return TRUE.



Example: I-8417/8817/8437/8837: demo_46, demo_27, demo_28
W-8337/8737: wdemo_26, wdemo_27, wdemo_28, wdemo_29

M_intln2 Move a long distance on X-Y plane



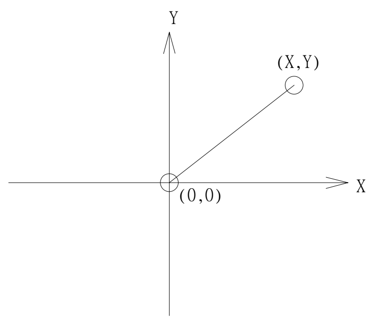
This command will move a long interpolation line on X-Y plane. It will automatically generate a trapezoidal speed profile of X-axis and Y-axis by state-machine-type calculation method.

Parameters:

card_NO_	integer	the card No. has been set by **M_regist**, valid is 0 ~ 19
x_, y_	integer	end point relate to present position
speed_	integer	0 ~ 2040
acc_mode_	integer	0: enable acceleration/deceleration profile 1: disable acceleration/deceleration profile

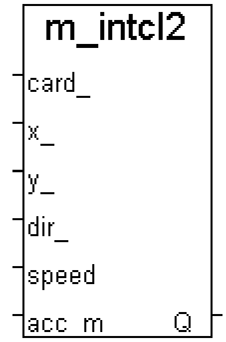
Return:

Q_	boolean	always return TRUE.



NOTE:
1. Only one of **M_intln2**, **M_intcl2** & **M_intar2** command can be called at one time, the other motion moving commands related to the same I-8091 card should not be called unless it is completed. (Please use **M_intstp** to test command of **M_intln2**, **M_intcl2** & **M_intar2** completed or not).
2. One controller can only drive one I-8091 to move by **M_intln2** , **M_intcL2** , **M_intar2** command. Two or more I-8091 cards in the same controller to use **M_intln2** , **M_intcL2** , **M_intar2** at the same time is not possible.

M_intcl2 Move a circle on X-Y plane



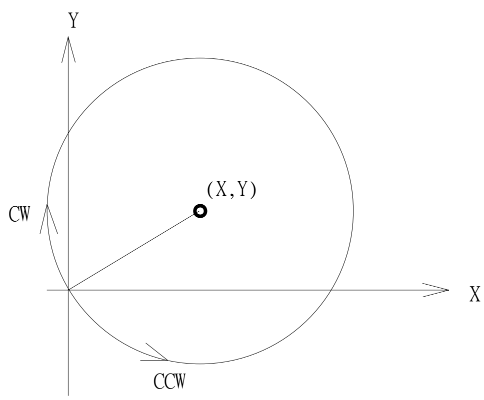
This command will generate an interpolation circle on X-Y plane. It will automatically generate a trapezoidal speed profile of X-axis and Y-axis by state-machine-type calculation method.

Parameters:

card_NO_	integer	the card No. has been set by **M_regist**, valid is 0 ~ 19
x_, y_	integer	center point of circle relate to present position
dir_	integer	moving direction. 0: CW , 1: CCW
speed_	integer	0 ~ 2040
acc_mode_	integer	0: enable acceleration/deceleration profile 1: disable acceleration/deceleration profile

Return:

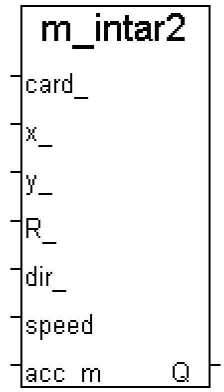
Q_	boolean	always return TRUE.



where radius = sqrt(X^2 + Y^2)

NOTE:
1. Only one of **M_intln2**, **M_intcl2** & **M_intar2** command can be called at one time, the other motion moving commands related to the same I-8091 card should not be called unless it is completed. (Please use **M_intstp** to test command of **M_intln2**, **M_intcl2** & **M_intar2** completed or not).
2. One controller can only drive one I-8091 to move by **M_intln2** , **M_intcL2** , **M_intar2** command. Two or more I-8091 cards in the same controller to use **M_intln2** , **M_intcL2** , **M_intar2** at the same time is not possible.

M_intar2 Move a arc on X-Y plane



This command will generate an interpolation arc on X-Y plane. It will automatically generate a trapezoidal speed profile of X-axis and Y-axis by state-machine-type calculation method.

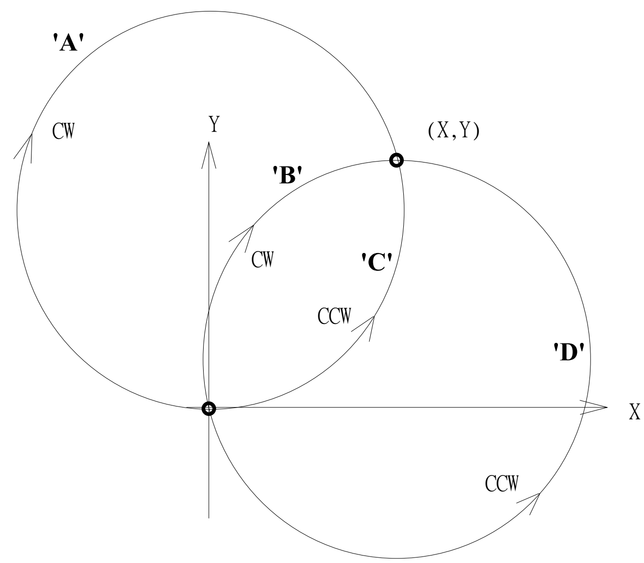
Parameters:

card_NO_	integer	the card No. has been set by **M_regist**, valid is 0 ~ 19
x_, y_	integer	end point of arc relate to present position
R_	integer	radius of arc, if > 0, the arc < 180 degree, if < 0, the arc > 180 degree R_ must > (square root of (X_*X_+Y_*Y_)) / 2
dir_	integer	moving direction. 0: CW , 1: CCW
speed_	integer	0 ~ 2040
acc_mode_	integer	0: enable acceleration/deceleration profile 1: disable acceleration/deceleration profile

Return:

Q_	boolean	always return TRUE.

R	dir	path of curve
R>0	CW	'B'
R>0	CCW	'C'
R<0	CW	'A'
R<0	CCW	'D'



NOTE:

1. Only one of **M_intln2**, **M_intcl2** & **M_intar2** command can be called at one time, the other motion moving commands related to the same I-8091 card should not be called unless it is completed. (Please use **M_intstp** to test command of **M_intln2**, **M_intcl2** & **M_intar2** completed or not).
2. One controller can only drive one I-8091 to move by **M_intln2** , **M_intcL2** , **M_intar2** command. Two or more I-8091 cards in the same controller to use **M_intln2** , **M_intcL2** , **M_intar2** at the same time is not possible.

M_intstp Test X-Y plane moving command

To test the below 3 commands completed or not.

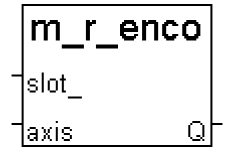
M_intln2 , **M_intcL2** , **M_intar2**

It will return FALSE for interpolation command completed while return TRUE for busy - not completed yet.

Return:
Q_ boolean TRUE: busy , FALSE: completed

NOTE:
1. Only one of **M_intln2**, **M_intcl2** & **M_intar2** command can be called at one time, the other motion moving commands related to the same I-8091 card should not be called unless it is completed. (Please use **M_intstp** to test command of **M_intln2**, **M_intcl2** & **M_intar2** completed or not).
2. One controller can only drive one I-8091 to move by **M_intln2** , **M_intcL2** , **M_intar2** command. Two or more I-8091 cards in the same controller to use **M_intln2** , **M_intcL2** , **M_intar2** at the same time is not possible.

I-8090 encorder commands:



M_r_enco   Reset I-8090's encorder value to 0

Parameters:

slot_	integer	the slot No. where the i8090 is plugged, 0 ~ 7
axis_	integer	1: x-axis, 2: y-axis, 3: z-axis

Return:

Q_	boolean	always return TRUE.

Example: demo_27, demo_28, demo_46

Chapter 19: Ethernet Communication and Security

The major t topics of this chapter are:

1.W-8037/8337/8737 communicate to Expansion Modbus TCP/IP I/O, and W-8037/8337/8737 communicate to remote PCs and workstations via TCP & UDP. Will be available at:
ftp://ftp.icpdas.com./pub/cd/winconcd/napdos/isagraf/wincon/english_manu/ "eth_io.pdf"

Note: I-8xx7 & I-7188EG/XG doesn't support this function.

2. Modbus TCP/IP security for I-8437/8837, 7188EG & W- W-8037/8337/8737. Will be available at ICP DAS's Ftp site.
ftp.icpdas.com/pub/cd/8000cd/napdos/isagraf/8000/english_manu/ "eth_security.pdf" or
ftp.icpdas.com./pub/cd/winconcd/napdos/isagraf/wincon/english_manu/ "eth_security.pdf"

Chapter 20: C Interface

This chapter details how to link user's own c routines with the ISaGRAF driver of Wincon-8037/8337/8737.

Note: I-8xx7 & I-7188EG/XG doesn't support this function.

This chapter will be available from the ICP DAS's Ftp site.
ftp://ftp.icpdas.com./pub/cd/winconcd/napdos/isagraf/wincon/english_manu/   "c_interface.pdf"

Chapter 21: Web Server For The Wincon-8xx7

This chapter details how to use Wincon-8037/8337/8737 as a web server. This will enable other PCs to access to the W-8xx7 via IE browser (Internet Explorer).

Note: I-8xx7 & I-7188EG/XG doesn't support this function.

This chapter will be available from the ICP DAS's Ftp site.
ftp://ftp.icpdas.com./pub/cd/winconcd/napdos/isagraf/wincon/english_manu/ "web_server.pdf"

Chapter 22: VB.net V.S. The Wincon-8xx7

This Chapter lists how to program VB.net application running in W-8xx7 to exchange data with the ISaGRAF application running in the same W-8xx7.

Please refer to Wincon CD-ROM.
Wincon CD-ROM: \napdos\isagraf\wincon\english_manu\ “VB.net_link_w8337.pdf”

Or ICP DAS’s Ftp site.
ftp.icpdas.com./pub/cd/winconcd/napdos/isagraf/wincon/english_manu/
“VB.net_link_w8337.pdf”

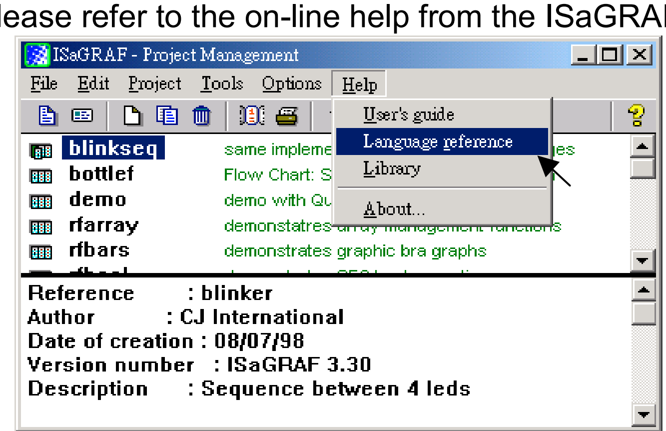
Appendix A: ISaGRAF Functions & Function Blocks For The I-8xx7, I-7188EG/XG & W-8xx7 Controller

Appendix A.1: Standard ISaGRAF Function Blocks

The following details the standard ISaGRAF function blocks that that can be programmed with the I-8xx7, I-7188EG/XG & W-8xx7 controller however labeled with "*" & "#" is not supported by I-8xx7 & I-7188EG/XG, while W-8xx7 doesn't support items with "#" label only.

-	#ARWRITE	*F_ROPEN	MSG	SHR
& (AND)	ASCII	F_TRIG	MUX4	SIG_GEN
*	ASIN	*F_WOPEN	MUX8	SIN
/	ATAN	*FA_READ	Neg	SQRT
+	AVERAGE	*FA_WRITE	NOT_MASK	SR
<	BLINK	FIND	ODD	STACKINT
<=	BOO	*FM_READ	#OPERATE	#SYSTEM
<>	CAT	*FM_WRITE	OR_MASK	TAN
=	CHAR	HYSTER	POW	TMR
=1 (XOR)	CMP	INSERT	R_TRIG	TOF
>	COS	INTEGRAL	RAND	TON
>=	CTD	LEFT	REAL	TP
>=1 (OR)	CTU	LIM_ALRM	REPLACE	TRUNC
1 gain	CTUD	LIMIT	RIGHT	XOR_MASK
ABS	#DAY_TIME	LOG	ROL	
ACOS	DELETE	MAX	ROR	
ANA	DERIVATE	MID	RS	
AND_MASK	EXPT	MIN	SEL	
#ARCREATE	*F_CLOSE	MLEN	SEMA	
#ARREAD	*F_EOF	MOD	SHL	

Please refer to the on-line help from the ISaGRAF workbench.



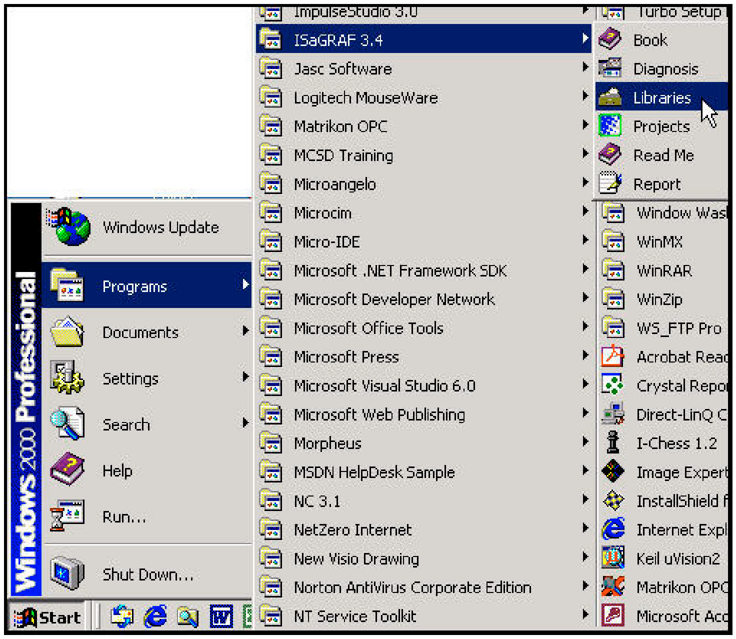
The function blocks listed in section A.4 are created by ICP DAS exclusively for the I-8xx7, I-7188EG/XG & W-8xx7 controller system. After installing the "ICP DAS Utilities For ISaGRAF" (please refer to section 1.2), these blocks in section A.4 can be found in the ISaGRAF Workbench program. Please refer to section A.4 for the "List Of Blocks" created for the controller system.

ICP DAS continually strives to improve the functionality of the I-8xx7, I-7188EG/XG & W-8xx7 controller system and the ISaGRAF Workbench program. Please visit the ICP DAS web site at http://www.icpdas.com/products/8000/isagraf.htm for updates and additions of new function blocks and functions created for the controller system.

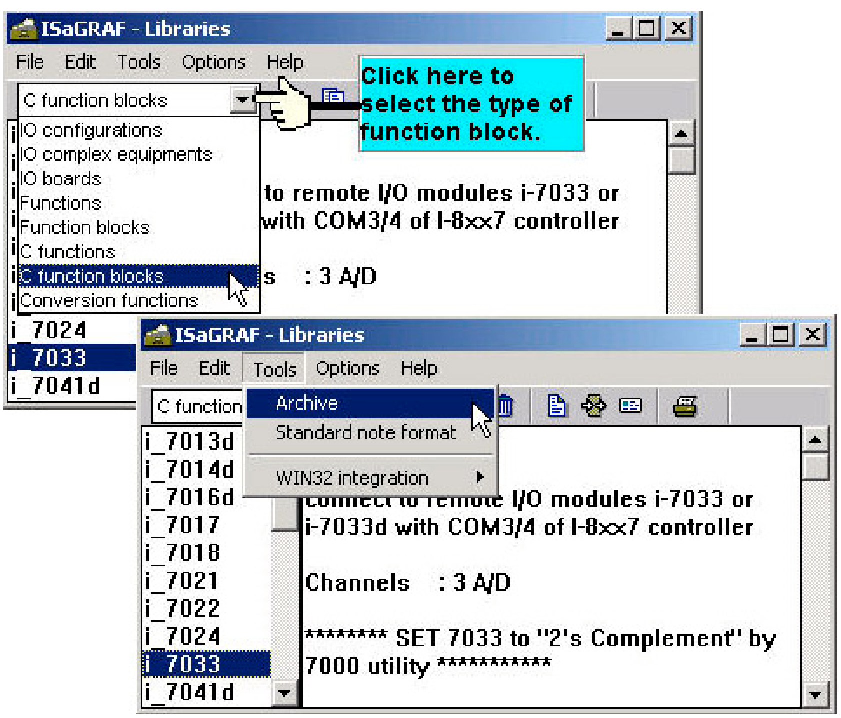
Please refer to section A.2 for more information on how to "Add New Blocks one by one To The ISaGRAF Workbench" program. (Section 1.2 is to install all of them at once)

Appendix A.2: Adding New Function Blocks To ISaGRAF

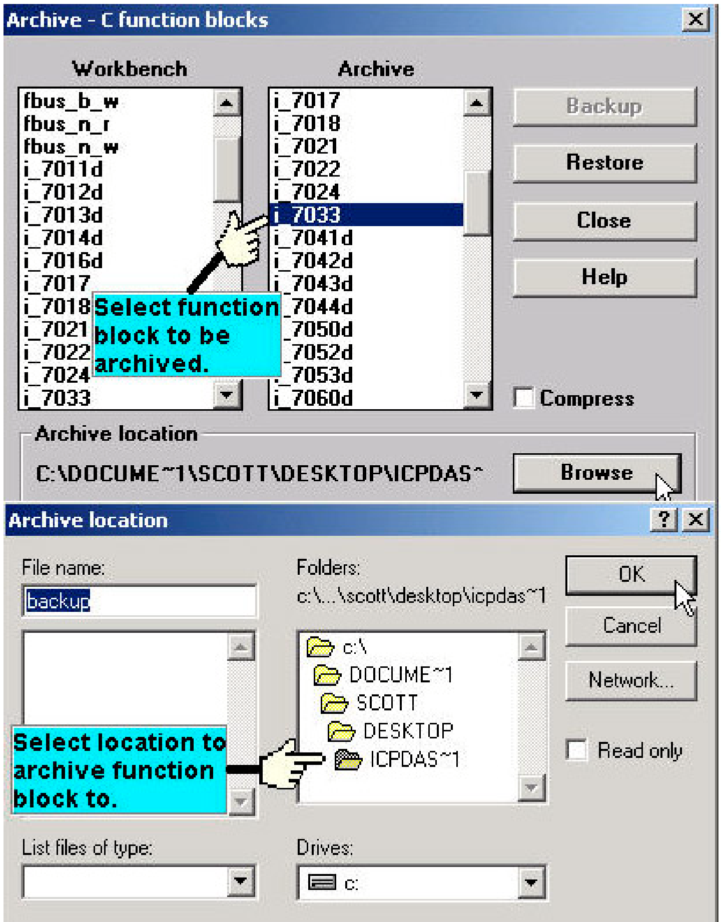
To add or update functions or function blocks one by one for the ISaGRAF Workbench program, click on the Windows "Start" menu, select "Programs", select "ISaGRAF 3.4", then click on "Libraries" to begin installing or updating ISaGRAF functions or function blocks.



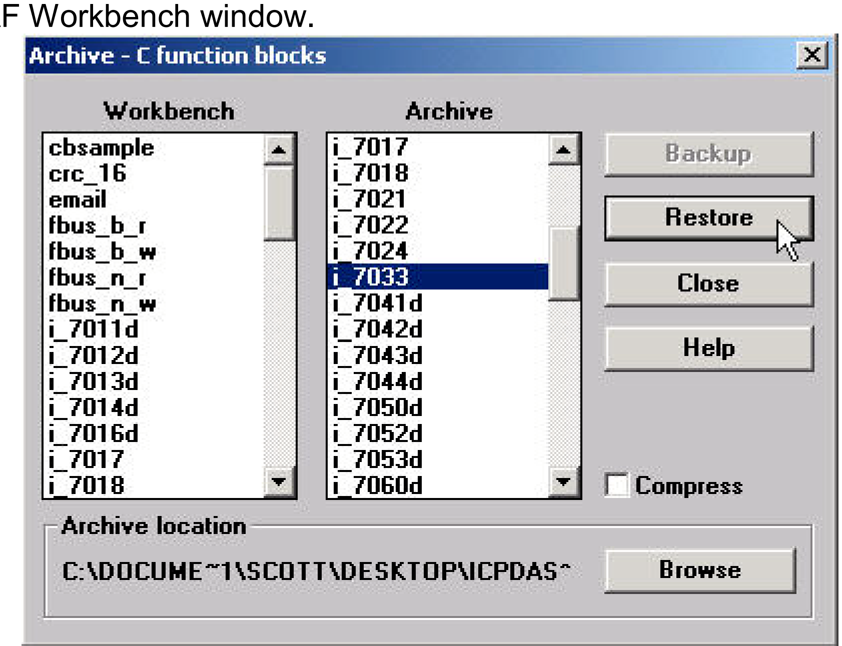
When you click on "Libraries" the "ISaGRAF Libraries" window will open. To add a new function block or function select "Tools" from the menu bar and then click on "Archive".



Click on the file name you want to "Archive" and then click "Browse" button to select the sub-directory to where (CD_ROM\Napdos\ISaGRAF\ARK\) you want to archive the function block library to.

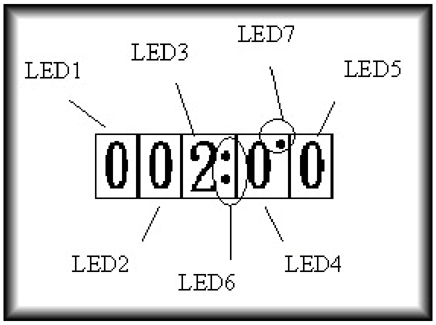


Select the new function block in the "Archive" window that you want to add, and then click on the "Restore" button. When you click on the "Restore" button the function block will be added to the ISaGRAF Workbench window.



Appendix A.3: I-8xx7 & I-7188EG/XG's 7-Segment LED Reference Table

The following table provides the reference definitions for programming the 7 LED indicators on the I-8xx7 & I-7188EG/XG controller system.



LED 6: Set to TRUE to display ":" (colon):
LED 7: Set to TRUE to display "." (period above LED 4)

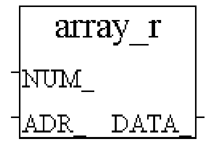
Display Table: LED 1 Through LED 5

Displayed Char.	Given Value	Displayed Char.	Given Value	Displayed Char.	Given Value
0	0	4.	20	r	40
1	1	5.	21	L	41
2	2	6.	22	n	42
3	3	7.	23	y	43
4	4	8.	24	U	44
5	5	9.	25	P	45
6	6	A.	26	o	46
7	7	b.	27	r.	47
8	8	C.	28	n.	48
9	9	d.	29	y.	49
A	10	E.	30	h.	50
b	11	F.	31	L.	51
C	12		32	U.	52
d	13	~	33	P.	53
E	14	–	34	o.	54
F	15	_	35	–.	55
0.	16	H	36	¯.	56
1.	17	h	37	_.	57
2.	18	H.	38	r	Others
3.	19	.	39		

Appendix A.4: Function Blocks For The Controller

The following function blocks have been developed specifically for the I-8xx7, I-7188EG/XG & W-8xx7 controller system.

ARRAY_R



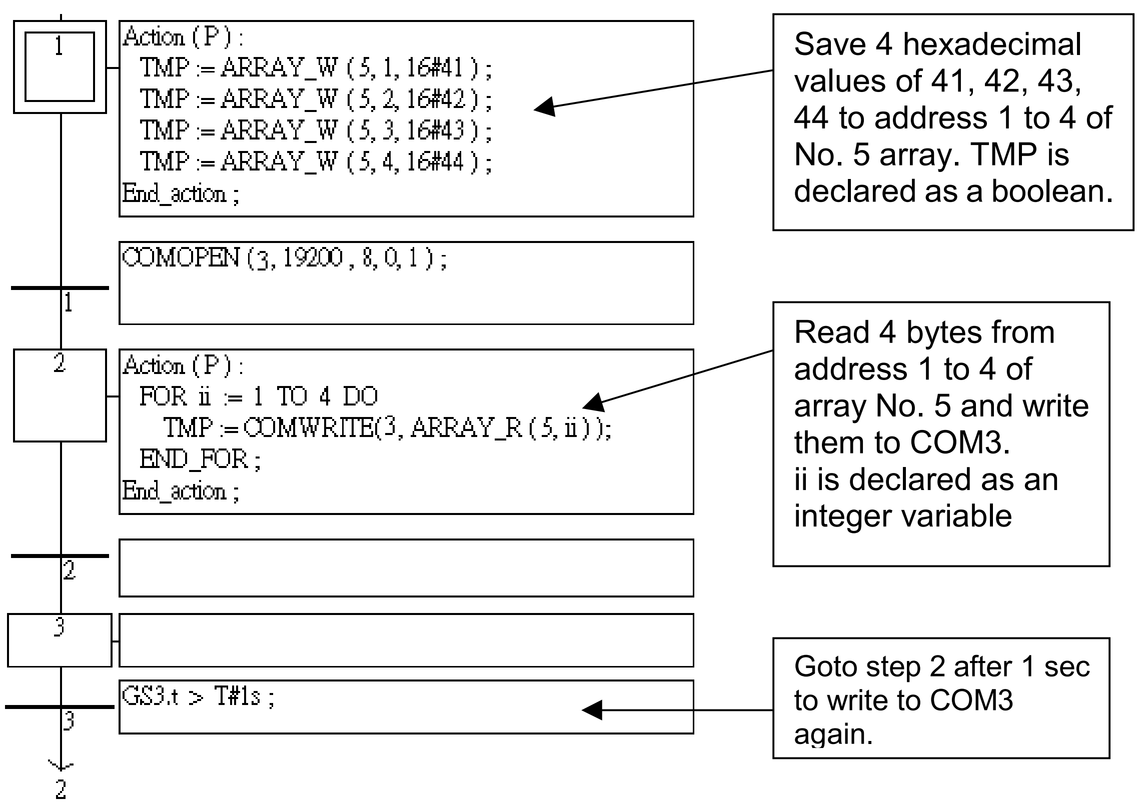
■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8037/8337/8737

Description:
Function **Read one byte from a byte array**

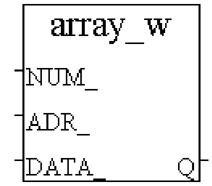
Arguments:

NUM_	integer	array ID to be operated, valid range values for the I-8xx7 & 7188EG/XG is from 1 to 24. For W-8xx7 is 1 to 48.
ADR_	integer	address in the array where the byte is to be stored, for the I-8xx7 & 7188EG/XG is from 1 to 256. For W-8xx7 is 1 to 512.
DATA_	integer	the byte value returned

Example:



ARRAY_W



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8037/8337/8737

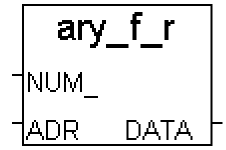
Description:
Function **Save one byte to a byte array**

Arguments:

NUM_	integer	array ID to be operated, valid range values for the I-8xx7 & 7188EG/XG is from 1 to 24. For W-8xx7 is 1 to 48.
ADR_	integer	address in the array where the byte is to be stored, for the I-8xx7 & 7188EG/XG is from 1 to 256. For W-8xx7 is 1 to 512.
DATA_	integer	the byte value to be saved to, valid range values from 0 to 255.
Q_	boolean	if OK. return TRUE, else return FALSE

Example: Refer to the "ARRAY_R" example.

ARY_F_R



□ I-8417/8817 □ I-8437/8837 □ I-7188EG □ I-7188XG ■ W-8037/8337/8737

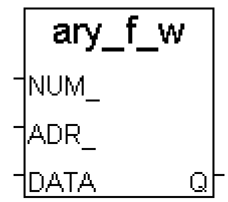
Description:

Function **Read one float value (32-bit format) from an float array**

Arguments:

NUM_	integer	array ID to be operated, valid range values is from 1 to 18.
ADR_	integer	address in the array where the integer is to be stored, valid range values from 1 to 256
DATA_	real	the float value returned

ARY_F_W



□ I-8417/8817 □ I-8437/8837 □ I-7188EG □ I-7188XG ■ W-8037/8337/8737

Description:

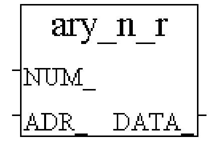
Function **Save one float value (32-bit format) to an float array**

Arguments:

NUM_	integer	array ID to be operated, valid range values is from 1 to 18
ADR_	integer	address in the array where the integer is to be stored, valid range values from 1 to 256
DATA_	real	the float value to be saved to.
Q_	boolean	if OK. return TRUE, else return FALSE

Note: The datas stored in array are cleared after power off

ARY_N_R



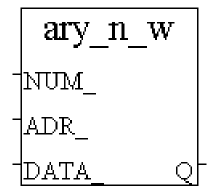
■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8037/8337/8737

Description:
Function **Read one integer (signed 32-bit) from an integer array**

Arguments:

NUM_	integer	array ID to be operated, valid range values for the I-8xx7 & I-7188EG/XG is from 1 to 6. For W-8xx7 is 1 to 18.
ADR_	integer	address in the array where the integer is to be stored, valid range values from 1 to 256
DATA_	integer	the integer value returned

ARY_N_W



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8037/8337/8737

Description:
Function **Save one integer to an integer array**

Arguments:

NUM_	integer	array ID to be operated, valid range values for the I-8xx7 & I-7188EG/XG is from 1 to 6. For W-8xx7 is 1 to 18
ADR_	integer	address in the array where the integer is to be stored, valid range values from 1 to 256
DATA_	integer	the integer value to be saved to.
Q_	boolean	if OK. return TRUE, else return FALSE

Note:

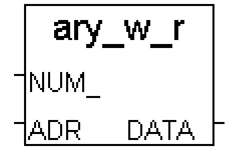
1. The long integer array use the same memory as short integer array. Be careful if using both of them at the same time.

Word array (ID, ADR)	Integer array (ID, ADR)
(1,1)	(1,1)
(1,2)	
(1,3)	(1,2)
(1,4)	
…	…
…	
(12,255)	(6,256)
(12,256)	
…	…
…	

2. Data stored in array is cleared after power off

Example: Refer to the “ARRAY_R” example.

ARY_W_R



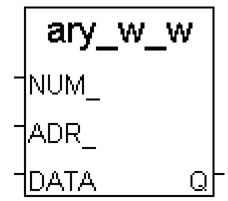
■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8037/8337/8737

Description:
Function read short integer (signed 16-bit) from array

Arguments:

NUM_	integer	array ID to be operated, for the I-8xx7 & I-7188EG/XG is from 1 to 12. For W-8xx7 is 1 to 36
ADR_	integer	address in the array where the integer is to be stored, valid range values from 1 to 256
DATA_	integer	the integer value returned, ranging from –32768 ~ +32767

ARY_W_W



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8037/8337/8737

Description:
Function write 1 short integer (signed 16-bit) to array of I-8xx7 controller

Arguments:

NUM_	integer	array ID to be operated, for the I-8xx7 & I-7188EG/XG is from 1 to 12. For W-8xx7 is 1 to 36
ADR_	integer	address in the array where the integer is to be stored, valid range values from 1 to 256
DATA_	integer	the integer value to be saved to. (-32768~+32767)
Q_	boolean	if OK. return TRUE, else return FALSE

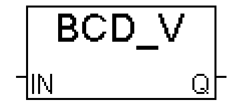
Note:
1. The long integer array use the same memory as short integer array. Be careful if use both of them at the same time.

Word array (ID, ADR)	Integer array (ID, ADR)
(1,1)	(1,1)
(1,2)	
(1,3)	(1,2)
(1,4)	
…	…
…	
(12,255)	(6,256)
(12,256)	
…	…
…	

2. The datas stored in array are cleared after power off

Example: Refer to the “ARRAY_R” example.

BCD_V



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8037/8337/8737

Description:
Function **Convert BCD value to decimal value**

Arguments:

IN_	integer	the BCD value to be converted
Q_	integer	the returned value, For ex.

16#12345 → 12345
16#3490 → 3490
18 → 12

BIN2ENG

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8037/8337/8737

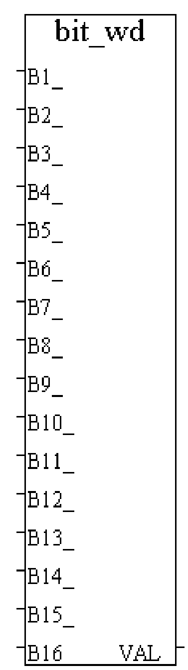
Description:
Function **Transfer 2's complement value to Engineering format value**

Arguments:

IN_	integer	2’s complement value to be converted
HI_2s_	integer	upper limit of 2’s complement, -32768 to +32767
LO_2s_	integer	lower limit of 2’s complement, -32768 to +32767
HI_EN_	integer	upper limit of engineering format, -32768 to +32767
LO_EN_	integer	lower limit of engineering format, -32768 to +32767
OUT_	integer	the returned engineering format value, for ex.

HI_2s_ = 32767 , LO_2s_ = -32768, HI_EN_ = 1000, LO_EN_ = -1000
IN_ = 16383 → OUT_= 500
IN_ = -12345 → OUT_= -377

BIT_WD



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8037/8337/8737

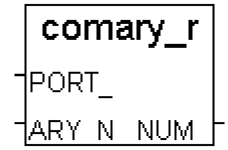
Description:
Function **Convert 16 boolean values to a word value**

Arguments:

B1_ ~ B16_	boolean	the 16 boolean values to be converted
VAL_	integer	the word value after the conversion For ex. If B1_ and B2_ are TRUE and others are all FALSE, VAL_ will be 3. If only B4_ is TRUE and others are all FALSE, VAL_ will be 8

COMARY_R

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8037/8337/8737



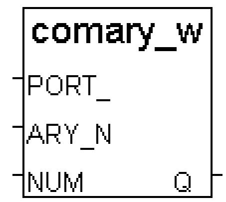
Description:
Function **Read all of the ready data of a COM PORT to a byte array**

Argument:

PORT_	integer	I-8xx7:1, 3 ~ 20, I-7188EG:1~8, I-7188XG:2~8, W-8xx7:2,3, or …
ARY_NO_	integer	Byte array ID (1-24 for I-8xx7 & I-7188EG/XG), (1-48 for W-8xx7), which is used to store the read bytes
NUM_	integer	return the number of bytes been read

COMARY_W

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8037/8337/8737



Description:
Function **Write a byte array to a COM PORT**

Argument:

PORT_	integer	I-8xx7:1, 3 ~ 20, I-7188EG:1~8, I-7188XG:2~8, W-8xx7:2,3, or …
ARY_NO_	integer	Byte array ID (1-24 for I-8xx7 & I-7188EG/XG), (1-48 for W-8xx7), which is used to store the read bytes
NUM_	integer	the number of bytes starting from the first address in the byte array to write
Q_	boolean	OK. return TRUE

Note:

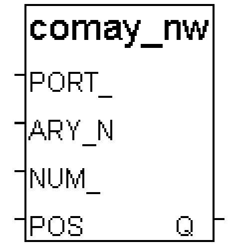
* If using I-8xx7 & I-7188EG's COM1, please set COM1 as non-Modbus-RTU port in advance before it can work. (refer to Appendix C.1)
* If Target is W-8xx7, please make sure its COM2 & COM3 is not Modbus RTU port before using them. (Please refer to W-8xx7's "Getting Started" Manual)
* For I-8xx7:
  ComPort No. on slot 0: Com5 ~ Com8
  ComPort No. on slot 1: Com9 ~ Com12
  ComPort No. on slot 2: Com13 ~ Com16
  ComPort No. on slot 3: Com17 ~ Com20
  ComPort No. on slot 4 ~ 7 is not available

Example:

Refer to Chapter 11 - Demo_21, 22 & 23.
Refer to function "ARRAY_R" & "ARRAY_W"

COMAY_NW

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8037/8337/8737



Description:
Function **Write one long integer array to COM PORT**

Each long integer is composed of 4 bytes. And the format is a signed long.
Each integer written is composed of 4 bytes in the below INTEL formate.
[lowest byte] [ ] [ ] [highest byte]
For ex., if there is 3 integers to write, the first one is 16#04030201 (67,305,985), the second one is 16#08070605 (134,678,021) and the last one is 16#FFFFFFFE (-2).
The 12 bytes been written will be [01] [02] [03] [04] [05] [06] [07] [08] [FE] [FF] [FF] [FF]

Argument:

PORT_	integer	I-8xx7:1, 3 ~ 20, I-7188EG:1~8, I-7188XG:2~8, W-8xx7:2,3, or …
ARY_NO_	integer	array ID (1-6 for I-8xx7 & I-7188EG/XG), (1-18 for W-8xx7), which is to write
NUM_	integer	the number of long integers starting from the POS_ address in the array to write
POS_	Integer	start position inside the array to write (1-256) if POS_ + NUM_ > 257, only (257-POS_) integer will be written for ex. if POS_=255, NUM_=3, only 2 integers written. They are Pos. 255 & Pos. 256.
Q_	boolean	OK. return TRUE

Note:

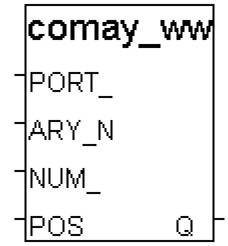
* If using I-8xx7 & I-7188EG's COM1, please set COM1 as non-Modbus-RTU port in advance before it can work. (refer to Appendix C.1)

* If Target is W-8xx7, please make sure its COM2 & COM3 is not Modbus RTU port before using them. (Please refer to W-8xx7's "Getting Started" Manual)

* For I-8xx7:
ComPort No. on slot 0: Com5 ~ Com8
ComPort No. on slot 1: Com9 ~ Com12
ComPort No. on slot 2: Com13 ~ Com16
ComPort No. on slot 3: Com17 ~ Com20
ComPort No. on slot 4 ~ 7 is not available

The long int array use the same memory as short interger array. Be careful if use both of them at the same time (please refer to Ary_n_r, Ary_n_w, Ary_w_r, Ary_w_w)

COMAY_WW



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8037/8337/8737

Description:
Function **Write one short Integer (Word) array to COM PORT**

Each short integer is composed of 2 bytes. And the format is a signed short int (-32768 ~ +32767).
Each short integer written is composed of 2 bytes in the below INTEL formate.
[low byte] [high byte]
For ex., if there is 3 short integers to write, the first one is 16#0403 (1,027), the second one is 16#0807 (2,055) and the last one is 16#FFFE (-2).
The 6 bytes been written will be [03] [04] [07] [08] [FE] [FF]

Argument:

PORT_	integer	I-8xx7:1, 3 ~ 20, I-7188EG:1~8, I-7188XG:2~8, W-8xx7:2,3, or …
ARY_NO_	integer	array ID (1-12 for I-8xx7 & I-7188EG/XG), (1-36 for W-8xx7), which is to write
NUM_	integer	the number of short integers starting from the POS_ address in the array to write
POS_	Integer	start position inside the array to write (1-256) if POS_ + NUM_ > 257, only (257-POS_) integer will be written for ex. if POS_=255, NUM_=3, only 2 integers written. They are Pos. 255 & Pos. 256.
Q_	boolean	OK. return TRUE

Note:

* If using I-8xx7 & I-7188EG's COM1, please set COM1 as non-Modbus-RTU port in advance before it can work. (refer to Appendix C.1)
* If Target is W-8xx7, please make sure its COM2 & COM3 is not Modbus RTU port before using them. (Please refer to W-8xx7's "Getting Started" Manual)
* For I-8xx7:
  ComPort No. on slot 0: Com5 ~ Com8
  ComPort No. on slot 1: Com9 ~ Com12
  ComPort No. on slot 2: Com13 ~ Com16
  ComPort No. on slot 3: Com17 ~ Com20
  ComPort No. on slot 4 ~ 7 is not available

  The long int array use the same memory as short interger array. Be careful if use both of them at the same time.

COMCLEAR

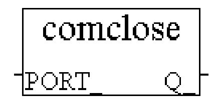
■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8037/8337/8737

Description:
Function **Clear receiving buffer of a COM PORT**

Argument:

PORT_	integer	I-8xx7:1, 3 ~ 20, I-7188EG:1~8, I-7188XG:2~8, W-8xx7:2,3, or …
Q_	boolean	OK. return TRUE

COMCLOSE



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8037/8337/8737

Description:
Function **Close COM PORT**

Argument:

PORT_	integer	I-8xx7:1, 3 ~ 20, I-7188EG:1~8, I-7188XG:2~8, W-8xx7:2,3, or …
Q_	boolean	OK. return TRUE

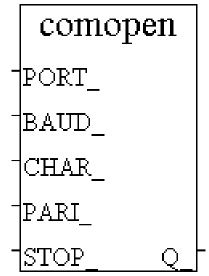
Note:

* If using I-8xx7 & I-7188EG's COM1, please set COM1 as non-Modbus-RTU port in advance before it can work. (refer to Appendix C.1)
* If Target is W-8xx7, please make sure its COM2 & COM3 is not Modbus RTU port before using them. (Please refer to W-8xx7's "Getting Started" Manual)
* For I-8xx7:
  ComPort No. on slot 0: Com5 ~ Com8
  ComPort No. on slot 1: Com9 ~ Com12
  ComPort No. on slot 2: Com13 ~ Com16
  ComPort No. on slot 3: Com17 ~ Com20
  ComPort No. on slot 4 ~ 7 is not available

Example:
Refer to the "COMOPEN" example.

COMOPEN

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8037/8337/8737



Description:
Function **Open COM port**

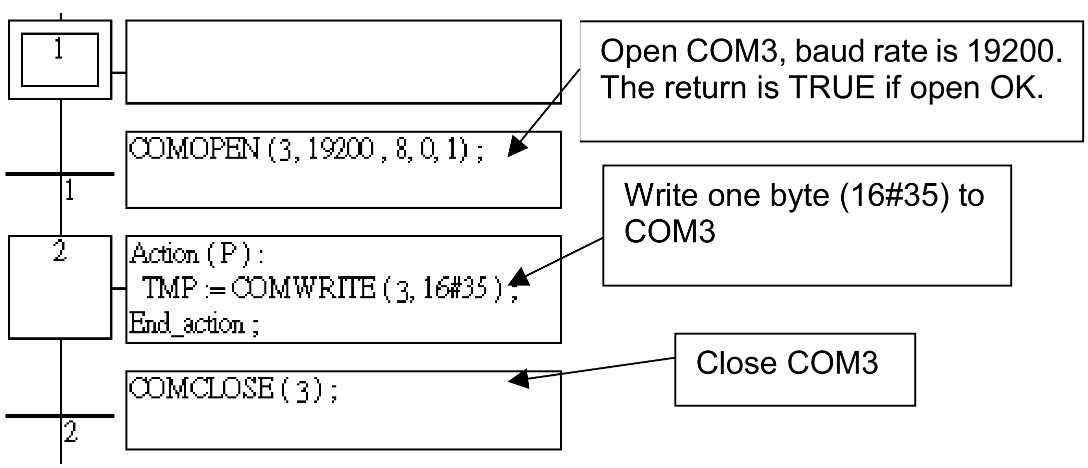
Argument:

PORT_	integer	I-8xx7:1, 3 ~ 20, I-7188EG:1~8, I-7188XG:2~8, W-8xx7:2,3, or …
BAUD_	integer	baud rate, can be 2400,4800, 9600, 19200, 38400, 57600, 115200
CHAR_	integer	character size, can be 7 or 8
PARI_	integer	parity, can be 0: none, 1: even, 2: odd
STOP_	integer	stop bit, can be 1 or 2
Q_	boolean	OK. return TRUE

Note:

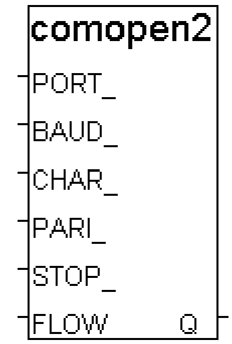
* If using I-8xx7 & I-7188EG's COM1, please set COM1 as non-Modbus-RTU port in advance before it can work. (refer to Appendix C.1)
* If Target is W-8xx7, please make sure its COM2 & COM3 is not Modbus RTU port before using them. (Please refer to W-8xx7's "Getting Started" Manual)
* For I-8xx7:
  ComPort No. on slot 0: Com5 ~ Com8
  ComPort No. on slot 1: Com9 ~ Com12
  ComPort No. on slot 2: Com13 ~ Com16
  ComPort No. on slot 3: Com17 ~ Com20
  ComPort No. on slot 4 ~ 7 is not available

Example:
Refer to Chapter 11 - Demo_21, 22 & 23.



COMOPEN2

□ I-8417/8817 □ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8037/8337/8737



Description:

Function **Open COM port with flow control, for RS232 port only**

Argument:

PORT_	integer	I-7188EG/XG:3~8, W-8xx7:2, or …
BAUD_	integer	baud rate, can be 2400,4800, 9600, 19200, 38400, 57600, 115200
CHAR_	integer	character size, can be 7 or 8
PARI_	integer	parity, can be 0: none, 1: even, 2: odd
STOP_	integer	stop bit, can be 1 or 2
FLOW_	boolean	True: flow control by hardware(CTS / RTS) (7188EG/XG 3 ~ 5), False: by software (XON / XOF) (7188EG/XG 3 ~ 8)
Q_	boolean	OK. return TRUE

* If Target is W-8xx7, please make sure its COM2 is not Modbus RTU port before using them. (Please refer to W-8xx7's "Getting Started" Manual)

COMREAD

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8037/8337/8737

Description:

Function	**Read one byte from a COM port**

Argument:

PORT_	integer	I-8xx7:1, 3 ~ 20, I-7188EG:1~8, I-7188XG:2~8, W-8xx7:2,3, or …
Q_	integer	the data returned

Note:

* If using I-8xx7 & I-7188EG's COM1, please set COM1 as non-Modbus-RTU port in advance before it can work. (refer to Appendix C.1)
* If Target is W-8xx7, please make sure its COM2 & COM3 is not Modbus RTU port before using them. (Please refer to W-8xx7's "Getting Started" Manual)
* For I-8xx7:
  ComPort No. on slot 0: Com5 ~ Com8
  ComPort No. on slot 1: Com9 ~ Com12
  ComPort No. on slot 2: Com13 ~ Com16
  ComPort No. on slot 3: Com17 ~ Com20
  ComPort No. on slot 4 ~ 7 is not available

*** Call COMREADY to test data coming or not . If there is data, COMREAD & COMARY_R can be used to read the data. If no data comimg, do not call COMREAD & COMARY_R, or COM port will block.**

Example:

Refer to "COMREADY" example.

COMREADY

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8037/8337/8737

Description:
Function **Test a COM port for data**

Argument:

PORT_	integer	I-8xx7:1, 3 ~ 20, I-7188EG:1~8, I-7188XG:2~8, W-8xx7:2,3, or …
Q_	boolean	If there is data coming, return TRUE. Else, return FALSE.

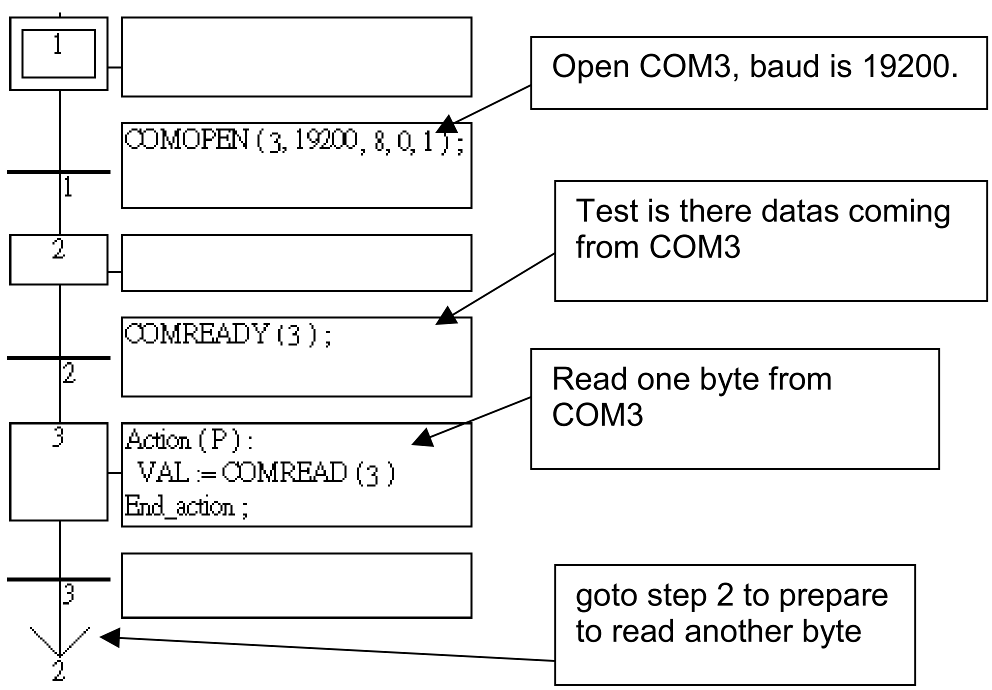
Note:

* If using I-8xx7 & I-7188EG's COM1, please set COM1 as non-Modbus-RTU port in advance before it can work. (refer to Appendix C.1)
* If Target is W-8xx7, please make sure its COM2 & COM3 is not Modbus RTU port before using them. (Please refer to W-8xx7's "Getting Started" Manual)
* For I-8xx7:
  ComPort No. on slot 0: Com5 ~ Com8
  ComPort No. on slot 1: Com9 ~ Com12
  ComPort No. on slot 2: Com13 ~ Com16
  ComPort No. on slot 3: Com17 ~ Com20
  ComPort No. on slot 4 ~ 7 is not available

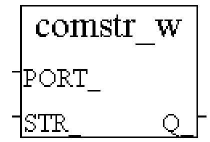
*** Call COMREADY to test data coming or not . If there is data, COMREAD & COMARY_R can be used to read the data. If no data comimg, do not call COMREAD & COMARY_R, or COM port will block.**

Example:

Refer to Chapter 11 - Demo_21, 22 & 23.



COMSTR_W



Argument:

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8037/8337/8737

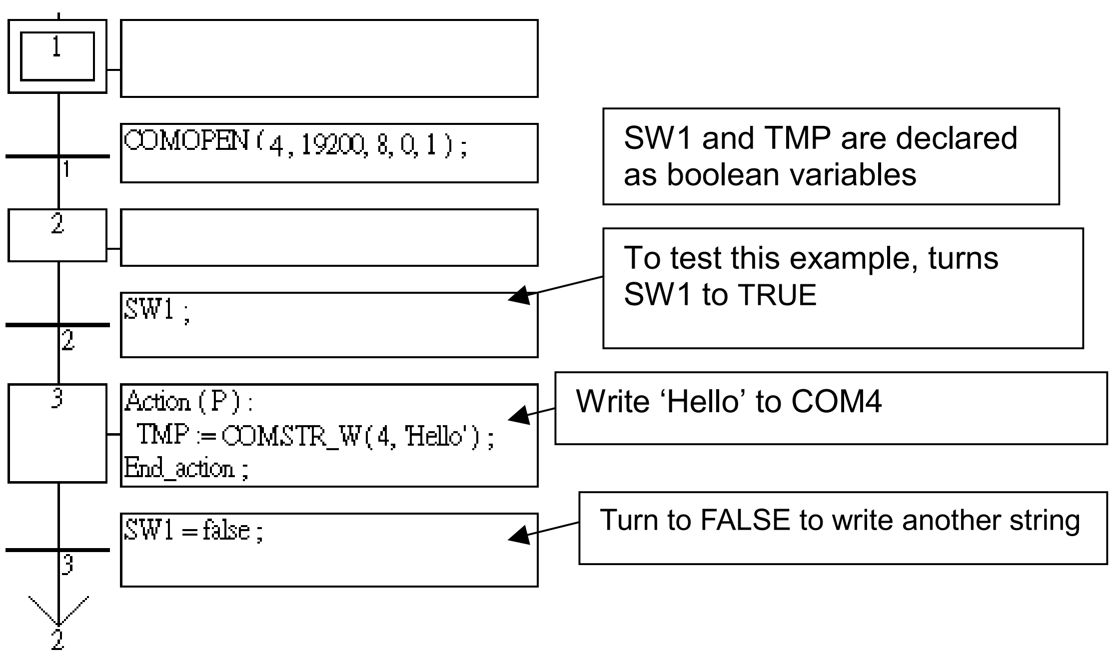
Description:

Function		**Write one string to a COM port**
PORT_	integer	I-8xx7:1, 3 ~ 20, I-7188EG:1~8, I-7188XG:2~8, W-8xx7:2,3, or …
STR_	Message	the string to be written (max length is 255).
Q_	boolean	Ok. return TRUE, else return FALSE.

Note:

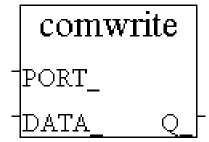
* If using I-8xx7 & I-7188EG's COM1, please set COM1 as non-Modbus-RTU port in advance before it can work. (refer to Appendix C.1)
* If Target is W-8xx7, please make sure its COM2 & COM3 is not Modbus RTU port before using them. (Please refer to W-8xx7's "Getting Started" Manual)
* For I-8xx7:
  ComPort No. on slot 0: Com5 ~ Com8
  ComPort No. on slot 1: Com9 ~ Com12
  ComPort No. on slot 2: Com13 ~ Com16
  ComPort No. on slot 3: Com17 ~ Com20
  ComPort No. on slot 4 ~ 7 is not available

Example:



COMWRITE

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8037/8337/8737



Description:

Function **Write one byte to a COM port**

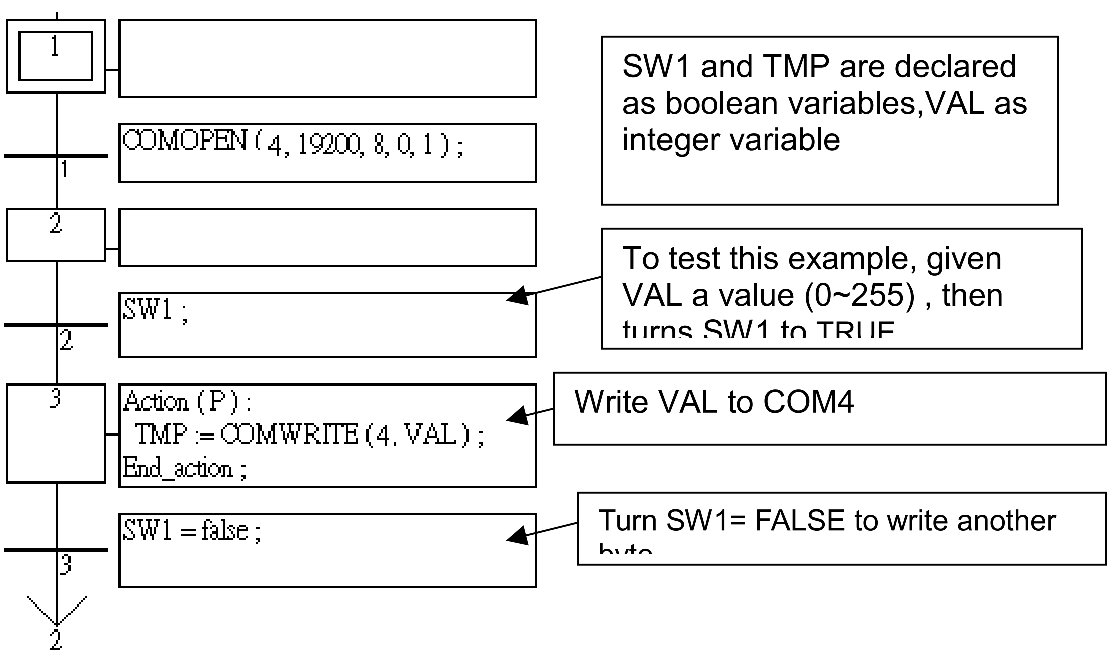
Argument:

PORT_	integer	I-8xx7:1, 3 ~ 20, I-7188EG:1~8, I-7188XG:2~8, W-8xx7:2,3, or …
DATA_	integer	the byte to be written, valid range values from 0 ~ 255.
Q_	boolean	Ok. return TRUE, else return FALSE.

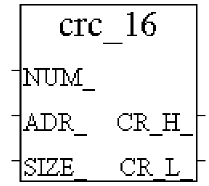
Note:

* If using I-8xx7 & I-7188EG's COM1, please set COM1 as non-Modbus-RTU port in advance before it can work. (refer to Appendix C.1)
* If Target is W-8xx7, please make sure its COM2 & COM3 is not Modbus RTU port before using them. (Please refer to W-8xx7's "Getting Started" Manual)
* For I-8xx7:
  ComPort No. on slot 0: Com5 ~ Com8
  ComPort No. on slot 1: Com9 ~ Com12
  ComPort No. on slot 2: Com13 ~ Com16
  ComPort No. on slot 3: Com17 ~ Com20
  ComPort No. on slot 4 ~ 7 is not available

Example:



CRC_16



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8037/8337/8737

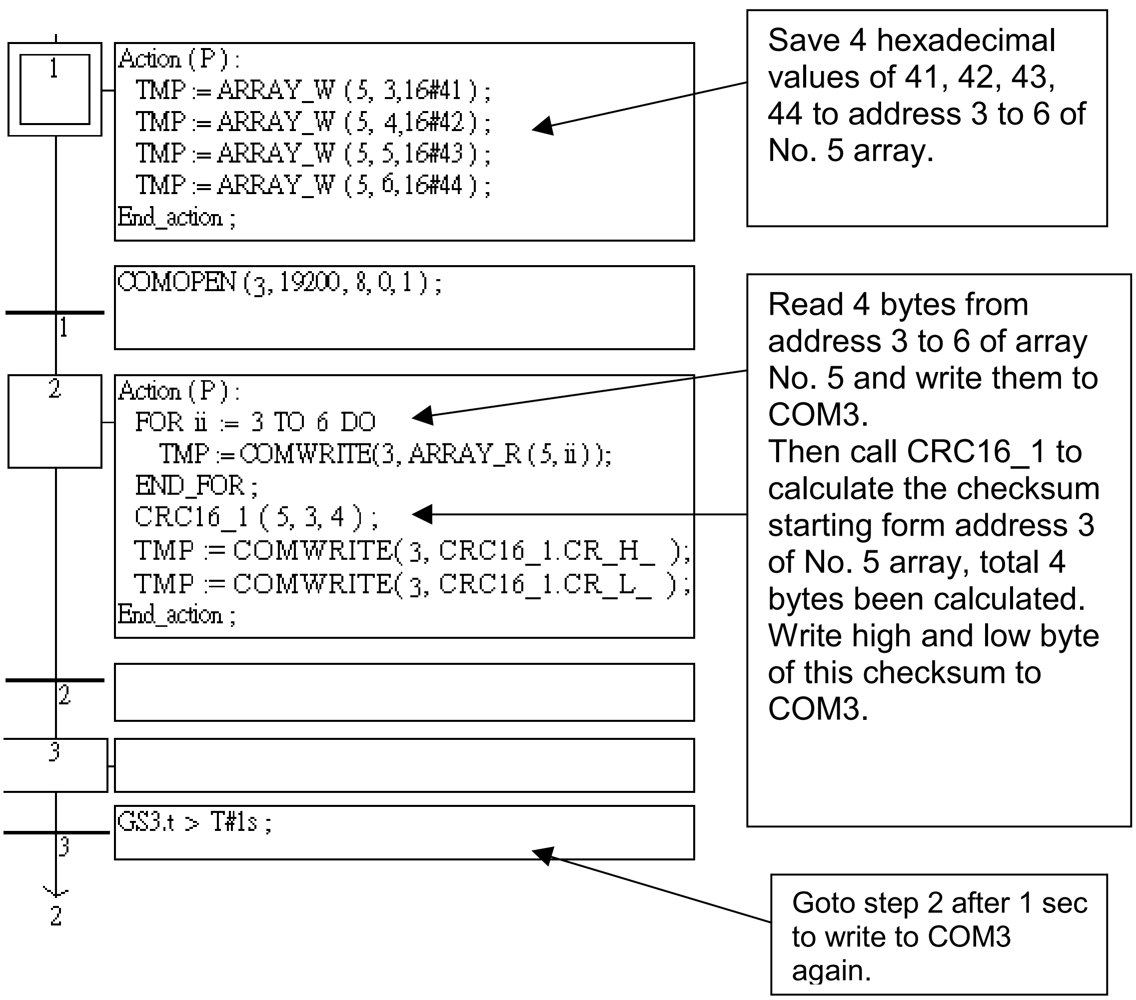
Description:
Function Block **Calculate checksum - CRC-16**

Argument:

NUM_	integer	byte array ID to be operated, Valid range for I-8xx7 & I-7188EG/XG is 1 to 24, for W-8xx7 is 1 to 48
ADR_	integer	starting address in the array which is to be calculated
SIZE_	integer	the number of bytes to be calculated
CR_H_	integer	the returned high byte of the CRC-16 after calculation.
CR_L_	integer	the returned low byte of the CRC-16 after calculation.

Example:

TMP is declared as a boolean. ii, CR_H_ and CR_L_ as integers, CRC16_1 is declared as FB instance of type – CRC_16.



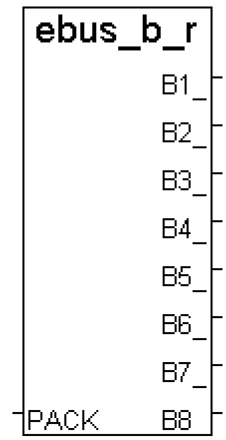
DI_CNT

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG □ W-8037/8337/8737

Description:
Function **Get parallel D/I counter at slot 0**

Please refer to Section 3.8

EBUS_B_R



□ I-8417/8817 ■ I-8437/8837 ■ I-7188EG □ I-7188XG ■ W-8037/8337/8737

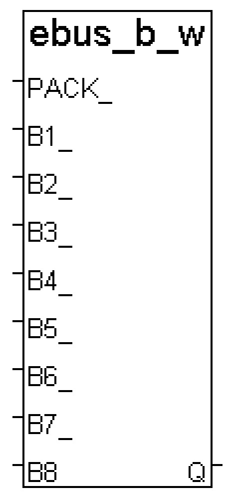
Description:
Function block **Read a boolean package from the Ebus device**

Arguments:

PACK_	integer	which package No. to read (1 - 128)
B1_ ~ B8_	boolean	the 8 boolean values contained in the package

Example:
Refer to Section 7.5

EBUS_B_W



□ I-8417/8817 ■ I-8437/8837 ■ I-7188EG □ I-7188XG ■ W-8037/8337/8737

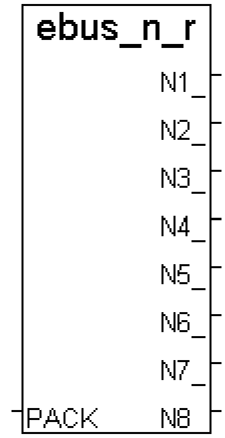
Description:
Function block **Write a boolean package to the Ebus device**

Arguments:

PACK_	integer	write to which package No. (1-128)
B1_ ~ B8_	boolean	the 8 boolean values contained in the package
Q	boolean	always return TRUE.

Example:
Refer to Section 7.5

EBUS_N_R



□ I-8417/8817 ■ I-8437/8837 ■ I-7188EG □ I-7188XG ■ W-8037/8337/8737

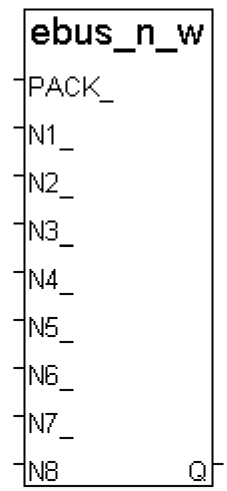
Description:
Function block **Read a integer package from the Ebus device**

Arguments:

PACK_	integer	which package No. to read. (1-128)
N1_ ~ N8_	integer	the 8 integer values contained in the package

Example:
Refer to Section 7.5

EBUS_N_W



□ I-8417/8817 ■ I-8437/8837 ■ I-7188EG □ I-7188XG ■ W-8037/8337/8737

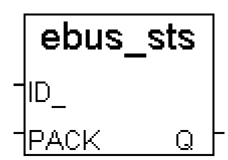
Description:
Function block **Write a integer package to the Ebus device**

Arguments:

PACK_	integer	write to which package No. (1-128)
N1_ ~ N8_	boolean	the 8 integer values contained in the package
Q	boolean	always return TRUE.

Example:
Refer to Section 7.5

EBUS_STS



□ I-8417/8817 ■ I-8437/8837 ■ I-7188EG □ I-7188XG ■ W-8037/8337/8737

Description:
Function **Get Package Status of Ebus**
Arguments:

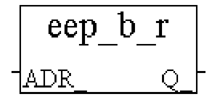
ID_	**Integer**	to get what ? 0: Boolean package , 1: Integer package
PACK_	**Integer**	get which package No. **(1-128)**

return:

Q_	**boolean**	TRUE: package is alive, FALSE: dead (communication break)

Example: Please refer to demo_49a & demo_49b

EEP_B_R



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8037/8337/8737

Description:
Function **read a boolean value from the EEPROM**

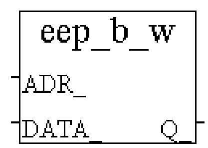
Argument:

ADR_	integer	address in the EEPROM where the boolean value is stored, I-8xx7, 7188EG/XG: 1 ~ 256 , W-8xx7: 1 ~ 1024
Q_	boolean	the boolean value returned

* Read operation of the EEPROM can be used freely without to remove the protection.
* Be careful to use EEP_B_W, EEP_BY_W, EEP_WD_W and EEP_N_W, the EEPROM can only to be written up to 100,000 times.

Example: refer to demo_17

EEP_B_W



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8037/8337/8737

Description:
Function **write a boolean value to the EEPROM**

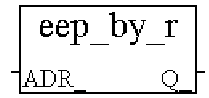
Arguments:

ADRES_	integer	address in the EEPROM where the boolean value is to be written to. I-8xx7, 7188EG/XG: 1 ~ 256 , W-8xx7: 1 ~ 1024
DATA_	Boolean	the boolean value to be written to
Q_	Boolean	Ok. return TRUE.

* To write to the EEPROM, the protection must be removed in advance
* Be careful to use EEP_B_W, EEP_BY_W, EEP_WD_W and EEP_N_W, EEPROM can only to be written up to 100,000 times.

Example: refer to demo_17

EEP_BY_R



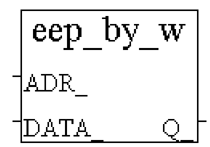
■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8037/8337/8737

Description:
Function **read a byte (8-bit integer) value from the EEPROM**

Argument:

ADR_	integer	address in the EEPROM where the byte value is stored. I-8xx7, 7188EG/XG:1 ~ 1512 , W-8xx7: 1 ~ 14272
Q_	integer	the byte value returned (0~255)

EEP_BY_W



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8037/8337/8737

Description:
Function **write a byte (8-bit integer) value to the EEPROM**

Arguments:

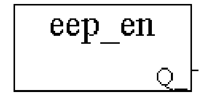
ADR_	integer	address in the EEPROM where the byte value is to be written to. I-8xx7, 7188EG/XG:1 ~ 1512 , W-8xx7: 1 ~ 14272
DATA_	integer	the byte value to be written to, valid range values from 0 to 255.
Q_	Boolean	Ok. return TRUE.

Note:
* If you are using this function with the EEP_WD_R, EEP_WD_W, EEP_N_R, and EEP_N_W functions simultaneously, you must be careful to arrange the ADR_ because they all occupy the same memory area. For example, ADR_2 of EEP_N_R occupies 4 bytes, and it uses the same memory area as ADR_3 and ADR_4 of EEP_WD_R and the same address of ADR_5, 6, 7, and 8 of EEP_BY_R.
* Read operation of the EEPROM will work without removing the EEPROM protection.
* The EEP_B_W, EEP_BY_W, EEP_WD_W and EEP_N_W functions should not be used to write to the EEPROM more than 100,000 times.

Example: refer to demo_17

EEP_EN

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8037/8337/8737



Description:
Function **Remove the EEPROM write protection**

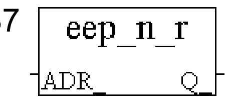
Argument:

Q_	Boolean	Ok: return TRUE, Fail: return FALSE

* BEFORE writing to the EEPROM, the EEPROM write protection must be turned off.
* The EEP_B_W, EEP_BY_W, EEP_WD_W and EEP_N_W functions should not be used to write to the EEPROM more than 100,000 times.

EEP_N_R

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8037/8337/8737



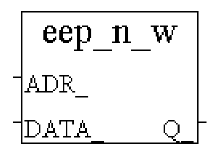
Description:
Function **read an signed 32-bit integer value from the EEPROM**

Argument:

ADR_	integer	address in the EEPROM where the 32-bit integer value is stored. I-8xx7, 7188EG/XG: 1 ~ 378 , W-8xx7: 1 ~ 3568
Q_	integer	the signed 32-bit integer value returned

EEP_N_W

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8037/8337/8737



Description:
Function **write a signed 32-bit integer value to the EEPROM**

Arguments:

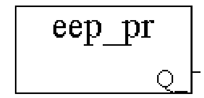
ADR_	integer	address in the EEPROM where the 32-bit integer value is to be written to , I-8xx7, 7188EG/XG: 1 ~ 378 , W-8xx7: 1 ~ 3568
DATA_	integer	the 32-bit integer value to be written to
Q_	Boolean	Ok. return TRUE.

Note:
* If you are using this function with the EEP_WD_R, EEP_WD_W, EEP_BY_R, and EEP_BY_W functions simultaneously, you must be careful to arrange the ADR_ because they all occupy the same memory area. For example, ADR_2 of EEP_N_R occupies 4 bytes, and it uses the same memory area as ADR_3 and ADR_4 of EEP_WD_R and the same address of ADR_5, 6, 7, and 8 of EEP_BY_R.
* Read operation of the EEPROM will work without removing the EEPROM protection.
* The EEP_B_W, EEP_BY_W, EEP_WD_W and EEP_N_W functions should not be used to write to the EEPROM more than 100,000 times.

Example: refer to demo_17

EEP_PR

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8037/8337/8737



Description:
Function **Set the EEPROM write protection**

Argument:

Q_	Boolean	Ok: return TRUE, Fail: return FALSE

* After writing to an EEPROM, it is better to turned off the write protection.
* The EEP_B_W, EEP_BY_W, EEP_WD_W and EEP_N_W functions should not be used to write to the EEPROM more than 100,000 times.

EEP_WD_R

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8037/8337/8737

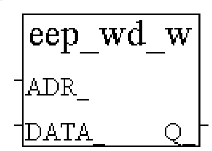
Description:
Function **read a word (signed 16-bit integer) value from the EEPROM**

Argument:

ADR_	integer	address in the EEPROM where the word value is stored. I-8xx7,7188EG/XG: 1 ~ 756 , W-8xx7: 1 ~ 7136
Q_	integer	the word value returned (-32768 ~ +32767)

EEP_WD_W

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8037/8337/8737



Description:
Function **write a word (signed 16-bit integer) value to the EEPROM**

Arguments:

ADR_	integer	address in the EEPROM where the word value is to be written to. I-8xx7,7188EG/XG: 1 ~ 756 , W-8xx7: 1 ~ 7136
DATA_	integer	the word value to be written to, range from -32768 to +32767.
Q_	Boolean	Ok. return TRUE.

Note:
* If you are using this function with the EEP_N_R, EEP_N_W, EEP_BY_R, and EEP_BY_W functions simultaneously, you must be careful to arrange the ADR_ because they all occupy the same memory area. For example, ADR_2 of EEP_N_R occupies 4 bytes, and it uses the same memory area as ADR_3 and ADR_4 of EEP_WD_R and the same address of ADR_5, 6, 7, and 8 of EEP_BY_R.
* Read operation of the EEPROM will work without removing the EEPROM protection.
* The EEP_B_W, EEP_BY_W, EEP_WD_W and EEP_N_W functions should not be used to write to the EEPROM more than 100,000 times.

Example: refer to demo_17

EMAIL

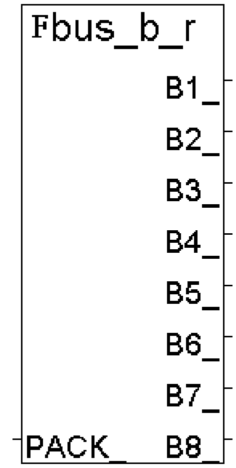
■ I-8417/8817 ■ I-8437/8837 □ I-7188EG □ I-7188XG □ W-8037/8337/8737

Description:
Function Block **Send an email**

Please refer to Chapter 12 – “Sending Emails” .

FBUS_B_R

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG □ W-8037/8337/8737



Description:
Function block **Read a boolean package from the Fbus device**

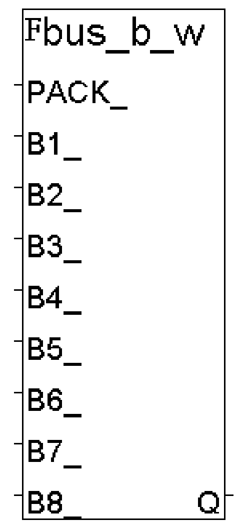
Arguments:

PACK_	integer	which package No. to read. (1-128)
B1_ ~ B8_	boolean	the 8 boolean values contained in the package

Example:
Refer to Chapter 7 or demo_11a.

FBUS_B_W

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG □ W-8037/8337/8737



Description:
Function block **Write a boolean package to the Fbus device**

Arguments:

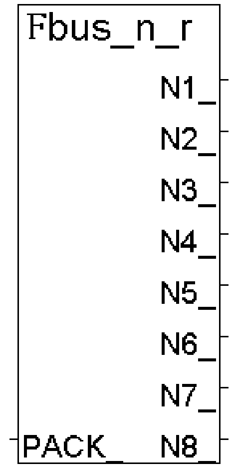
PACK_	integer	write to which package No. (1-128)
B1_ ~ B8_	boolean	the 8 boolean values contained in the package
Q	boolean	always TRUE.

Example:

Refer to Chapter 7 or demo_11b.

FBUS_N_R

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG □ W-8037/8337/8737



Description:
Function block **Read an integer package (signed 32-bit) from the Fbus device**

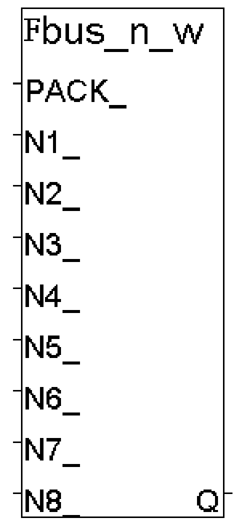
Arguments:

PACK_	integer	which package No. to read. (1-128)
N1_ ~ N8_	integer	the 8 integer values contained in the package

Example:
Refer to Chapter 7 or demo_11b.

FBUS_N_W

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG □ W-8037/8337/8737



Description:
Function block **Write an integer package (signed 32-bit) to the Fbus device**

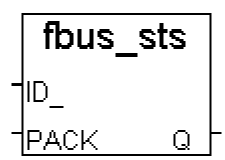
Arguments:

PACK_	integer	write to which package No. (1-128)
N1_ ~ N8_	boolean	the 8 integer values contained in the package
Q	boolean	always TRUE.

Example:
Refer to Chapter 7 or demo_11a.

FBUS_STS

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG □ W-8037/8337/8737



Description:
Function **Get Package Status of Fbus**

Arguments:

ID_	**Integer**	to get what ? 0: Boolean package , 1: Integer package
PACK_	**Integer**	get which package No. **(1-128)**

return:

Q_	**boolean**	TRUE: package is alive, FALSE: dead (communication break)

Example: Please refer to demo_49a & demo_49b

F_CREAT

□ I-8417/8817 □ I-8437/8837 □ I-7188EG □ I-7188XG ■ W-8037/8337/8737

Description:
Function **Creat an empty file for Reaing & Writing.**

Arguments:

Path_	Message	File path & name. For ex **'\Compact Flash\data.txt'**
ID_	integer	file ID returned, if error happens, it returns 0

Note:
1. If the file already exist, the data inside it will be destroyed when calling this function.
2. For reading existing file, please call ISaGRAF Standard Function – “F_ROPEN( )”
3. For writing & reading existing file, please call ISaGRAF Standard Function – “F_WOPEN( )”

F_READ_B

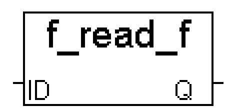
□ I-8417/8817 □ I-8437/8837 □ I-7188EG □ I-7188XG ■ W-8037/8337/8737

Description:
Function **Read one byte from current position of an open file.**

Arguments:

ID_	integer	File ID No. returned by F_ROPEN , F_WOPEN or F_CREAT
Q_	integer	the returned byte (0 - 255)

F_READ_F



□ I-8417/8817 □ I-8437/8837 □ I-7188EG □ I-7188XG ■ W-8037/8337/8737

Description:
Function **Read one float value (32-bit format) from current position of an open file.**

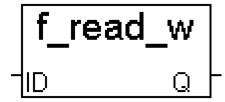
Arguments:

ID_	integer	File ID No. returned by F_ROPEN , F_WOPEN or F_CREAT
Q_	real	the returned float value (32-bit format)

Note:
1. Using ISaGRAF Standard Function –“FA_READ” & “FA_WRITE” to R/W **long** integer
2. Using ISaGRAF Standard Function –“FM_READ” & “FM_WRITE” to R/W message (string)

Example:
Refer to Wincon CD:\napdos\isagraf\wincon\demo\ “wdemo_01 & wdemo_02”

F_READ_W



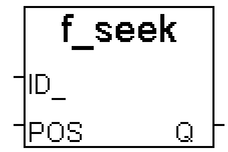
□ I-8417/8817 □ I-8437/8837 □ I-7188EG □ I-7188XG ■ W-8037/8337/8737

Description:
Function **Read one word (signed 16-bit integer) from current position of an open file.**

Arguments:

ID_	integer	File ID No. returned by F_ROPEN , F_WOPEN or F_CREAT
Q_	integer	the returned word (-32768 ~ +32767)

F_SEEK



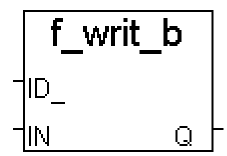
□ I-8417/8817 □ I-8437/8837 □ I-7188EG □ I-7188XG ■ W-8037/8337/8737

Description:
Function **Move file position to ...**

Arguments:

ID_	integer	File ID No. returned by F_ROPEN , F_WOPEN or F_CREAT
POS_	integer	position, unit is **byte** (1 to ...)
Q_	boolean	True: Ok , False: fail

F_WRIT_B



□ I-8417/8817 □ I-8437/8837 □ I-7188EG □ I-7188XG ■ W-8037/8337/8737

Description:
Function **Write one byte to current position of an open file**

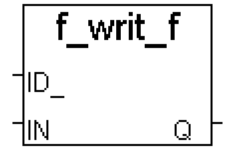
Arguments:

ID_	integer	File ID No. returned by F_ROPEN , F_WOPEN or F_CREAT
IN_	integer	The byte value to write, 0 ~ 255. if value > 255 or <0, the lowest byte is written
Q_	boolean	True: Ok , False: fail

Note:
1. Using ISaGRAF Standard Function –“FA_READ” & “FA_WRITE” to R/W **long** integer
2. Using ISaGRAF Standard Function –“FM_READ” & “FM_WRITE” to R/W message (string)

Example:
Refer to Wincon CD:\napdos\isagraf\wincon\demo\ “wdemo_01 & wdemo_02”

F_WRIT_F



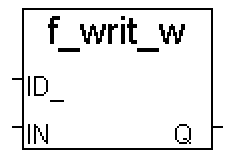
□ I-8417/8817 □ I-8437/8837 □ I-7188EG □ I-7188XG ■ W-8037/8337/8737

Description:
Function **Write one float value (32-bit format) to current position of an open file**

Arguments:

ID_	integer	File ID No. returned by F_ROPEN , F_WOPEN or F_CREAT
IN_	real	The float value to write, (32-bit format)
Q_	boolean	True: Ok , False: fail

F_WRIT_W



□ I-8417/8817 □ I-8437/8837 □ I-7188EG □ I-7188XG ■ W-8037/8337/8737

Description:
Function **Write one word value (signed 16-bit integer) to current position of an open file**

Arguments:

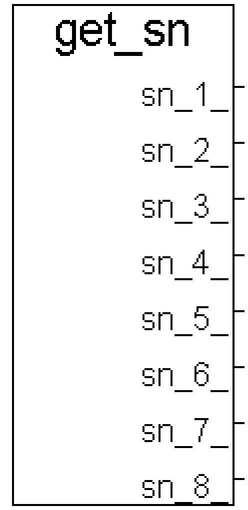
ID_	integer	File ID No. returned by F_ROPEN , F_WOPEN or F_CREAT
IN_	integer	The word value to write, (-32768 ~ +32767)
Q_	boolean	True: Ok , False: fail

Note:
1. Using ISaGRAF Standard Function –"FA_READ" & "FA_WRITE" to R/W **long** integer
2. Using ISaGRAF Standard Function –"FM_READ" & "FM_WRITE" to R/W message (string)

Example:
Refer to Wincon CD:\napdos\isagraf\wincon\demo\ "wdemo_01 & wdemo_02"

GET_SN

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8037/8337/8737



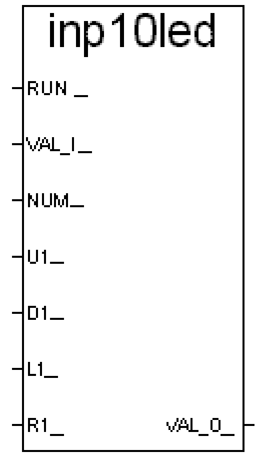
Description:
Function block **get hardware unique serial No.**

Arguments:

Sn_1_ ~ 8 Integer the returned serial No. 8 bytes.

INP10LED

■ I-8417/8817 ■ I-8437/8837 □ I-7188EG □ I-7188XG □ W-8037/8337/8737

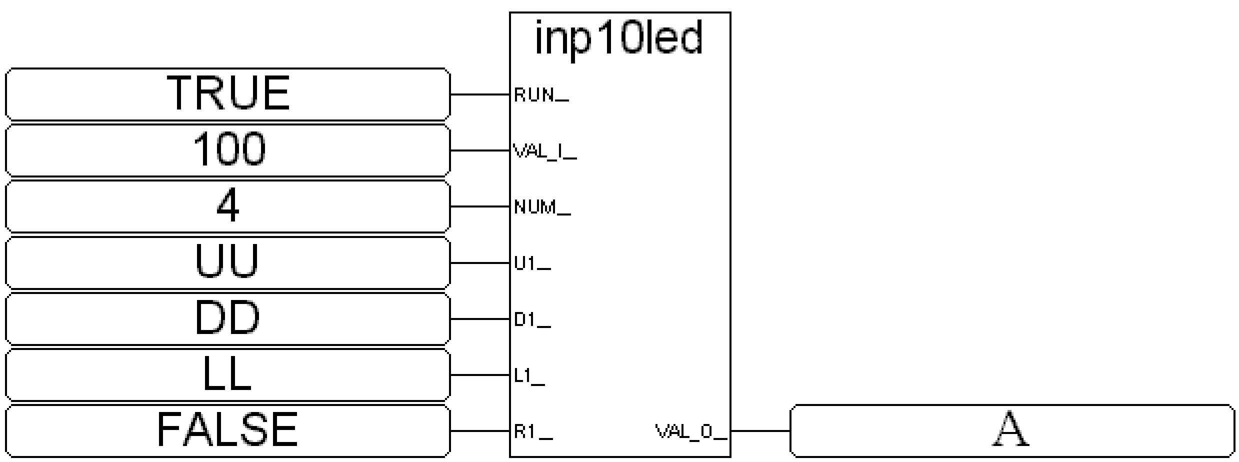


Description:
Function **input an decimal integer from the S_MMI**

Arguments:

RUN_	Boolean	When "TRUE", Process & Display Value To SMMI
VAL_I_	Integer	Initial Value Displayed On S-MMI, Minimum Value Is "0", maximum is 99999
NUM_	Integer	Number Of Digits To Display, Valid Range From 1 To 5
U1_	Boolean	When Rising From "FALSE" To "TRUE", Add 1 To The Currently Displayed Digit
D1_	Boolean	When Rising From "FALSE" To "TRUE", Subtract 1 From The Currently Displayed Digit
L1_	Boolean	When Rising From "FALSE" To "TRUE", Shift Left 1 Position From Currently Displayed Digit
R1_	Boolean	When Rising From "FALSE" To "TRUE", Shift Right 1 Position From Currently Displayed Digit
VAL_O_	integer	The Displayed Integer Value After Operation

Example: refer to demo_08, demo_11a.

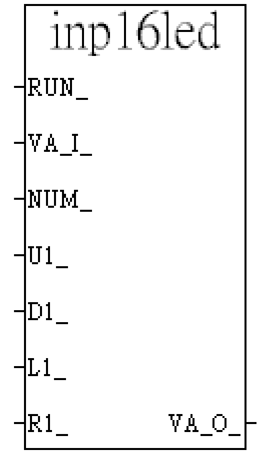


ST equivalence:

```
A := INP10LED(TRUE,100,4,UU,DD,LL,FALSE);
(* A is declared as an integer variable *)
(* UU,DD,LL are declared as boolean variables, can be linked to “push4key” board *)
```

INP16LED

■ I-8417/8817 ■ I-8437/8837 □ I-7188EG □ I-7188XG □ W-8037/8337/8737

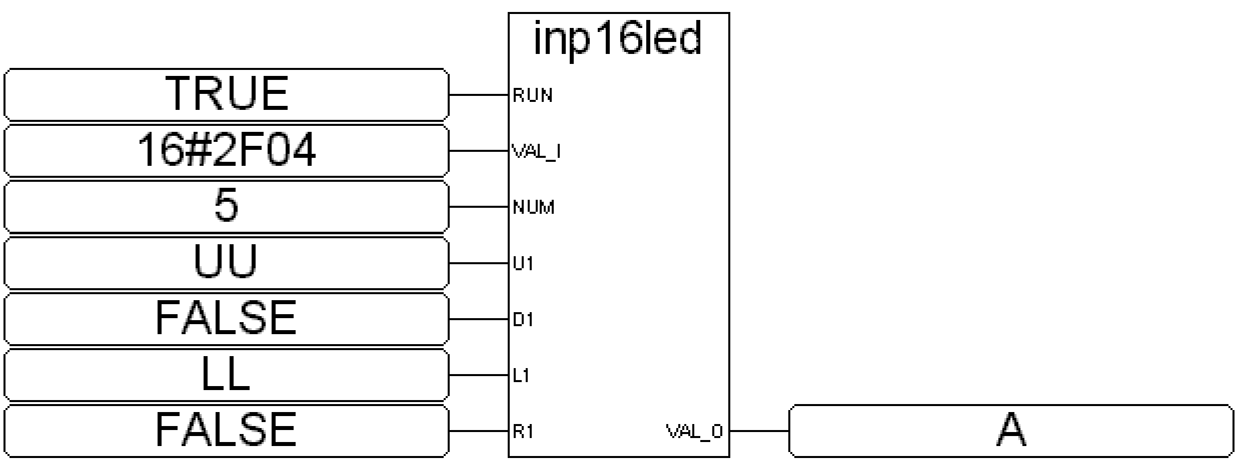


Description:
Function **input an hexadecimal integer from the S_MMI**

Arguments:

RUN_	Boolean	When "TRUE", Process & Display Value To S-MMI
VAL_I_	Integer	Initial Value Displayed On S-MMI, Minimum Value Is "0", maximum is 16#FFFF
NUM_	Integer	Number Of Digits To Display, Valid Range From 1 To 5
U1_	Boolean	When Rising From "FALSE" To "TRUE", Add 1 To The Currently Displayed Digit
D1_	Boolean	When Rising From "FALSE" To "TRUE", Subtract 1 From The Currently Displayed Digit
L1_	Boolean	When Rising From "FALSE" To "TRUE", Shift Left 1 Position From Currently Displayed Digit
R1_	Boolean	When Rising From "FALSE" To "TRUE", Shift Right 1 Position From Currently Displayed Digit
VAL_O_	integer	The Displayed Integer Value After Operation

Example:



ST equivalence:

```
A := INP16LED(TRUE,16#2F04,4,UU,FALSE,LL,FALSE);
(* A is declared as an integer variable *)
(* UU,LL are declared as boolean variables,can be linked to “push4key” board *)
```

INT_REAL

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8037/8337/8737

Description:
Function **Map a long integer to a Real value.**

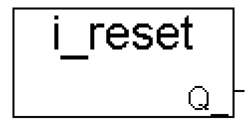
The algorithm in C language is Real_ = *((float *)&Long_);

Arguments:

Long_	integer	the 32-bit integer
Real_	real	the real value after mapping

Note: "Real_Int" can be used to map a Real value to a long integer.

I_RESET



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8037/8337/8737

Description:
Function **Reset the controller**

return:

Q_	**boolean**	The return value has no meaning since the controller will reset

Note:
Please use this function very careful. If the controller is always reset, please refer to section 1.3.7 to delete the project inside the controller.

Example:
```
(* OK1 is declared as boolean input, TMP as boolean internal *)
if  OK1=TRUE  then
  TMP := i_reset();
end_if;
```

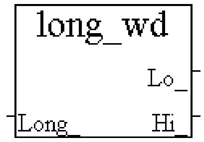
I7000_EN

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8037/8337/8737

Description:
Function **Enable/Disable Bus7000 communication**

Please refer to Section 6.4

LONG_WD

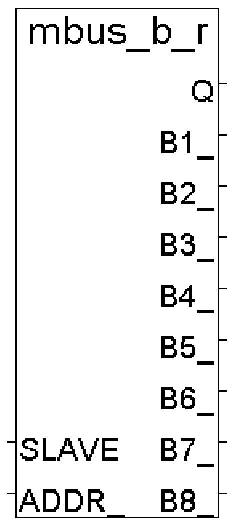


■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8037/8337/8737

Description:
Function block **Convert one integer to two words**
Arguments:

LONG_	integer	the 32-bit integer to be converted
LO_	integer	the low word value after the conversion, from -32768 to +32767
HI_	integer	the high word value after conversion, from -32768 to +32767

MBUS_B_R



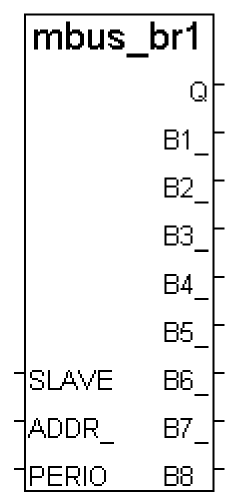
■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8037/8337/8737

Description:
Function block **Read 8 bits (booleans) from the Mdobus device**
Use Modbus function code ---- 1
Arguments:

SLAVE_	integer	slave No. of the Modbus device, valid range from 0 to 255, should be constant value not variable.
ADDR_	integer	the starting Modbus address to read, 0-65535. , should be constant value not variable.
Q_	boolean	Ok. return TRUE, else return FALSE
B1_ ~ B8_	boolean	the 8 boolean values that have been read

Note: The total number of "MBUS_B_R" that can be used in one ISaGRAF project is up to 64.
Example: Refer to Chapter 8.

MBUS_BR1



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8037/8337/8737

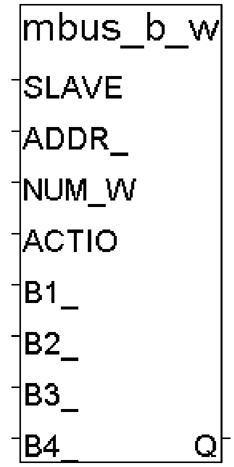
Description:
Function block **Read 8 bits (booleans) from the Mdobus device with period time**
Use Modbus function code ---- 1
Arguments:

SLAVE_	integer	slave No. of the Modbus device, valid range from 0 to 255. , should be constant value not variable.
ADDR_	integer	the starting Modbus address to read, 0-65535. , should be constant value not variable.
PERIOD_	integer	read data depends on period time, default is 1 sec. The value should be 1 ~ 600 (sec)
Q_	boolean	Ok. return TRUE, else return FALSE
B1_ ~ B8_	boolean	the 8 boolean values that have been read

Note: The total number of "MBUS_BR1" + "MBUS_B_R" that can be used in one ISaGRAF project is up to 64.

MBUS_B_W

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8037/8337/8737



Description:

Function block **write 1 to 4 bits (booleans) to the Mdobus device**

Use Modbus function code 5 when NUM_W = 1
Use Modbus function code 15 when NUM_W = 2 to 4

Arguments:

SLAVE_	integer	slave No. of the Modbus device, valid range from 0 to 255. , should be constant value not variable.
ADDR_	integer	the starting Modbus address to write, 0-65535. , should be constant value not variable.
NUM_W_	integer	the number of bits to write, valid range from 1 to 4. , should be constant value not variable.
ACTION_	boolean	Set true to write, set FALSE to do nothing
B1_ ~ B4_	boolean	bits to write
Q_	boolean	Ok. return TRUE, else return FALSE

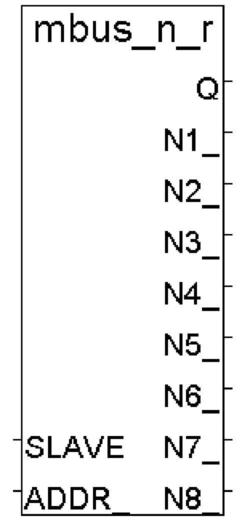
Note: The total number of “MBUS_B_W” + “MBUS_WB” blocks that can be used in one ISaGRAF project is up to 64.

Example:

Refer to Chapter 8 or demo_16.

MBUS_N_R

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8037/8337/8737



Description:
Function block **Read 8 words (16-bit integer) from the Mdobus device**
Use Modbus function code ---- 3

Arguments:

SLAVE_	integer	slave No. of the Modbus device, valid range from 0 to 255. , should be constant value not variable.
ADDR_	integer	the starting Modbus address to read, 0-65535. , should be constant value not variable.
Q_	boolean	Ok. return TRUE, else return FALSE
N1_ ~ N8_	integer	the 8 word values that have been read, valid range values from -32768 to +32767

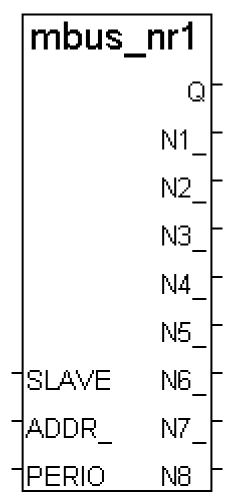
Note: The total number of "MBUS_N_R" + "MBUS_R" blocks that can be used in one ISaGRAF project is up to 64.

Example:
Refer to Chapter 8 or demo_15a.

MBUS_NR1

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8037/8337/8737

Description:
Function block **Read 8 words (16-bit integer) from the Mdobus device with period time**
Use Modbus function code ---- 3

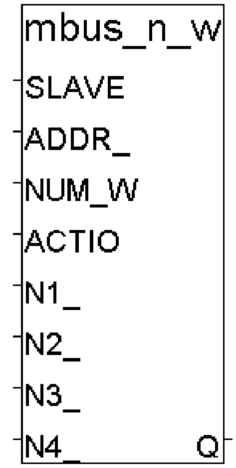


Arguments:

SLAVE_	integer	slave No. of the Modbus device, valid range from 0 to 255. , should be constant value not variable.
ADDR_	integer	the starting Modbus address to read, 0-65535. , should be constant value not variable.
PERIOD_	integer	read data depends on period time, default is 1 sec. The value should be 1 ~ 600 (sec)
Q_	boolean	Ok. return TRUE, else return FALSE
N1_ ~ N8_	integer	the 8 word values that have been read, valid range values from -32768 to +32767

Note: The total number of "MBUS_N_R" + "MBUS_R" + "MBUS_NR1" blocks that can be used in one ISaGRAF project is up to 64.

MBUS_N_W



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8037/8337/8737

Description:

Function block **write 1 to 4 words (booleans) to the Mdobus device**

Use Modbus function code 6 when NUM_W = 1
Use Modbus function code 16 when NUM_W = 2 to 4

Arguments:

SLAVE_	integer	slave No. of the Modbus device, valid range from 0 to 255. , should be constant value not variable.
ADDR_	integer	the starting Modbus address to write, 0-65535. , should be constant value not variable.
NUM_W_	integer	the number of words to write, valid range values from 1 to 4. , should be constant value not variable.
ACTION_	boolean	Set true to write, set FALSE to do nothing
N1_ ~ N4_	integer	words to write (-32768 ~ 32767)
Q_	boolean	Ok. return TRUE, else return FALSE

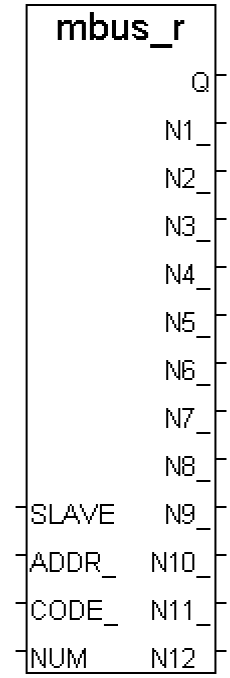
Note: The total number of “MBUS_N_W” blocks that can be used in one ISaGRAF project is up to 64.

Example:

Refer to Chapter 8.

MBUS_ R

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8037/8337/8737



Description:
Function block Read Modbus code 1-4 from the Modbus device

* ISaGRAF controller is the Master, remote equipment is Slave
* adapt Modbus function code 1 or 2 or 3 or 4
* please make sure the remote device support the associated Modbus function code

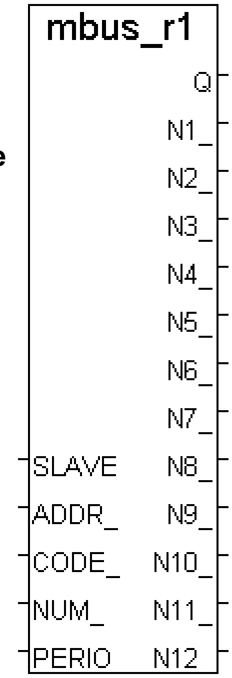
Arguments:

SLAVE_	integer	slave No. of the Modbus device, valid range from 0 to 255. , should be constant value not variable.
ADDR_	integer	the starting Modbus address to read, 0-65535. , should be constant value not variable.
CODE_	integer	Request which Modbus function codes, 1-4. , should be constant value not variable.
NUM_	integer	Request how many bits? 1-192 for code 1 & 2 or How many words? 1-12 for code 3 & 4. , should be constant value not variable.
Q_	boolean	Ok. return TRUE, else return FALSE
N1_ ~ N12_	integer	The bits or words received. If CODE_ is 1 & 2, N1_ returns bit 1 to 16, N2_ returns bit 17 to 32, ... N12_ returns bit 177 to 192. If CODE_ is 3 & 4, N1_ to N12_ returns the associated words (-32768 to 32767). N1_ to N12_ is absolutly correct Only when Q return TRUE (comm. ok)

Note: The total number of “MBUS_N_R” + “MBUS_R” + “MBUS_R1” blocks that can be used in one ISaGRAF project is up to 64.

MBUS_ R1

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8037/8337/8737



Description:
Function block **Read Modbus code 1-4 from the Modbus device with period time**

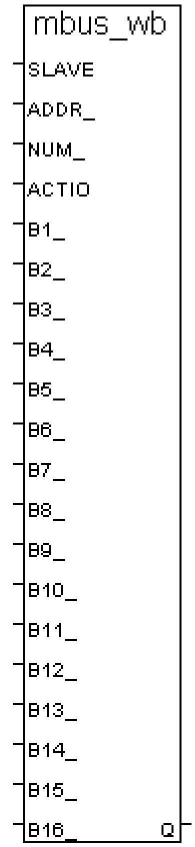
* ISaGRAF controller is the Master, remote equipment is Slave
* adapt Modbus function code 1 or 2 or 3 or 4
* please make sure the remote device support the associated Modbus function code

Arguments:

SLAVE_	integer	slave No. of the Modbus device, valid range from 0 to 255. , should be constant value not variable.
ADDR_	integer	the starting Modbus address to read, 0-65535. , should be constant value not variable.
CODE_	integer	Request which Modbus function codes, 1-4. , should be constant value not variable.
NUM_	integer	Request how many bits? 1-192 for code 1 & 2 or How many words? 1-12 for code 3 & 4 . , should be constant value not variable.
PERIOD_	integer	read data depends on period time, default is 1 sec. The value should be 1 ~ 600 (sec)
Q_	boolean	Ok. return TRUE, else return FALSE
N1_ ~ N12_	integer	The bits or words received. If CODE_ is 1 & 2, N1_ returns bit 1 to 16, N2_ returns bit 17 to 32, ... N12_ returns bit 177 to 192. If CODE_ is 3 & 4, N1_ to N12_ returns the associated words (-32768 to 32767). N1_ to N12_ is absolutly correct Only when Q return TRUE (comm. ok)

Note: The total number of “MBUS_N_R” + “MBUS_R” + “MBUS_R1” blocks that can be used in one ISaGRAF project is up to 64

MBUS_WB



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8037/8337/8737

Description:
Function block **write 1 to 16 bits (booleans) to the Mdobus device**

Use Modbus function code 15

Arguments:

SLAVE_	integer	slave No. of the Modbus device, range from 0 to 255. , should be constant value not variable.
ADDR_	integer	the starting Modbus address to write, 0-65535. , should be constant value not variable.
NUM_W_	integer	the number of bits to write, valid range from 1 to 16. , should be constant value not variable.
ACTION_	boolean	Set true to write, set FALSE to do nothing
B1_ ~ B16_	boolean	bits to write
Q_	boolean	Ok. return TRUE, else return FALSE

Note: The total number of “MBUS_B_W” + “MBUS_WB” blocks that can be used in one ISaGRAF project is up to 64.

MI_BOO

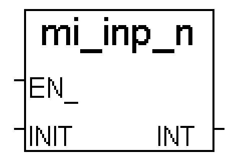
■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8037/8337/8737

Description:
Function **Display a boolean value on MMICON**

Arguments:

X_	integer	X position, 1-30
Y_	integer	Y position, 1-8
BOO_	boolean	boolean value to display. TRUE display “ON”, FALSE display “OFF”
Q_	boolean	Ok. return TRUE, else return FALSE

MI_INP_N



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8037/8337/8737

Description:
Function **Input an integer value from MMICON**

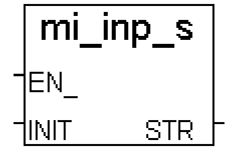
Arguments:

EN_	boolean	TRUE: enable
INIT_	integer	Initial value to input
INT_	integer	The integer value been input. If EN_ is FALSE , it returns 0

Note:
MI_INP_N & MI_INP_S Can be used only at one place in the project. Called at 2 or more places will work fail.

Demo:
Please refer to Chapter 16 & demo_38, demo_39

MI_INP_S



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8037/8337/8737

Description:
Function **Input an string from MMICON**

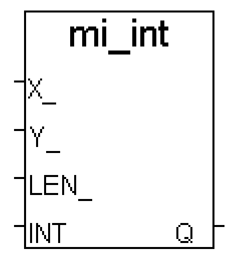
Arguments:

EN_	boolean	TRUE: enable
INIT_	message	Initial string value to input
STR_	message	The string been input. If EN_ is FALSE , it returns ‘’ (empty string)

Note:
MI_INP_N & MI_INP_S Can be used only at one place in the project. Called at 2 or more places will work fail.

To input a real value, please use MI_INP_S, STR_REAL & REAL_STR and refer to demo_39.

MI_INT



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8037/8337/8737

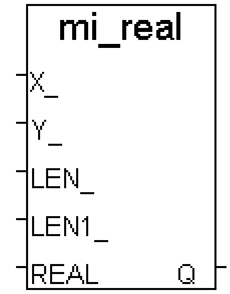
Description:
Function **Display an Integer value on MMICON**

Arguments:

X_	integer	X position, 1-30
Y_	integer	Y position, 1-8
LEN_	integer	Max number of digits to display, 1-11
INT_	integer	integer value to display.
Q_	boolean	Ok. return TRUE, else return FALSE

Demo:
Please refer to Chapter 16 & demo_38, demo_39

MI_REAL



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8037/8337/8737

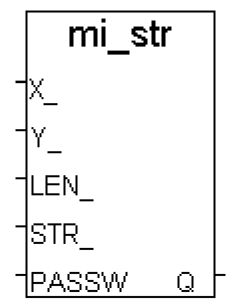
Description:
Function **Display a real value on MMICON**

Arguments:

X_	integer	X position, 1-30
Y_	integer	Y position, 1-8
LEN_	integer	Max number of digits to display, 1-13
LEN1_	integer	number of digit after '.' (0~4) and less than LEN_. For ex. if LEN_=7, LEN1_=2, "123.4567" will be displayed as " 123.45"
REAL_	real	real value to display. If the number of digits exceeds LEN_, '******' will be displayed
Q_	boolean	Ok. return TRUE, else return FALSE

Note:
If abs. of the real value >= 1,000,000 or (> 0 & < 0.0001) , please give LEN_ as 13 to display for ex. -123,456,789, please set LEN_ to 13 and it is displayed as -1.23457e+008. And 0.0000123456, please set LEN_ to 13 and it is displayed as 1.23456e-005

MI_STR



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8037/8337/8737

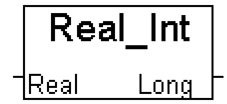
Description:
Function **Display a string on MMICON**

Arguments:

X_	integer	X position, 1-30
Y_	integer	Y position, 1-8
LEN_	integer	Max number of characters to display, 1-240
STR_	message	The string to display. If the number of characters exceeds LEN_, only the first LEN_ of char. will be displayed
PASSWD_	boolean	TRUE: display as password, all char. are replaced as '*'. FALSE: displayed as string.
Q_	boolean	Ok. return TRUE, else return FALSE

Demo:
Please refer to Chapter 16 & demo_38, demo_39

REAL_INT



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8037/8337/8737

Description:
Function **Map a Real value to a long integer.**

The algorithm in C language is Long_ = *((long *)&Real_);

Arguments:

Real_	real	the real value to map
Long_	integer	the 32-bit integer after mapping

Note:
"Int_Real" can be used to map a long integer to a Real value.

REAL_STR

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8037/8337/8737

Description:
Function **Convert a Real value to a string.**

Arguments:

REAL_	real	the real value to convert
STR_	message	the string returned (Max length is 13), For ex.

1.234 ---> '1.234'
123456789.0 ---> '1.23457E+008'
0.00001234 ---> '1.234E-005'

Note:
"STR_REAL" can be used to convert a string to a Real value.

PID_AL

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8037/8337/8737
Example:
Please refer to Chapter 11 - Demo_18, and ICP DAS CD-ROM :
\Napdos\ISaGRAF\8000\English_Manu\PID_AL.Complex PID algorithm implementation.htm

PWM_DIS

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG □ W-8037/8337/8737

Description:
Function Disable PWM output
Please refer to Section 3.7.

PWM_EN

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG □ W-8037/8337/8737
Description:
Function Enable PWM output.
Please refer to Section 3.7.

PWM_EN2

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG □ W-8037/8337/8737
Description:
Function Enable PWM output to output some pulse.
Please refer to Section 3.7.

PWM_ON

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG □ W-8037/8337/8737
Description:
Function Set parallel D/O to TRUE immediatelly
Please refer to Section 3.7.

PWM_OFF

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG □ W-8037/8337/8737
Description:
Function Set parallel D/O to FALSE immediatelly
Please refer to Section 3.7.

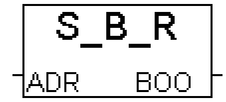
PWM_STS

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG □ W-8037/8337/8737
Description:
Function Get PWM status

Please refer to Section 3.7.

S_B_R

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG □ W-8037/8337/8737



Description:
Function   Read one boolean from the volatile SRAM

Arguments:

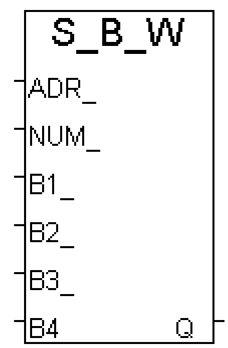
ADR_	Integer	read which address, one Boolean occupy 1 byte.
		S256: 1 ~ 249,856 (1 ~ 16#3D000) S512: 1 ~ 512,000 (1 ~ 16#7D000) X607: 1 ~ 118,784 (1 ~ 16#1D000) X608: 1 ~ 512,000 (1 ~ 16#7D000)

return:

BOO_	Boolean	The boolean value been read is 0=FALSE, not 0 = TRUE

S_B_W

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG □ W-8037/8337/8737



Description:
Function   Write up to 4 boolean to the volatile SRAM

Arguments:

ADR_	Integer	start from which address, one boolean occupy 1 byte.
		S256: 1 ~ 249,856 (1 ~ 16#3D000) S512: 1 ~ 512,000 (1 ~ 16#7D000) X607: 1 ~ 118,784 (1 ~ 16#1D000) X608: 1 ~ 512,000 (1 ~ 16#7D000)
NUM_	Integer	how many booleans to write, 0 ~ 4
B1_~B4_	Boolean	the boolean value to write

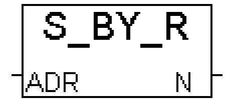
return:

Q_	Boolean	Ok: TRUE, Fail: FALSE

The boolean value will be stored is FALSE=0, TRUE=1

Please refer to section 10.3

S_BY_R



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG □ W-8037/8337/8737

Description:
Function Read one byte from the volatile SRAM

Arguments:

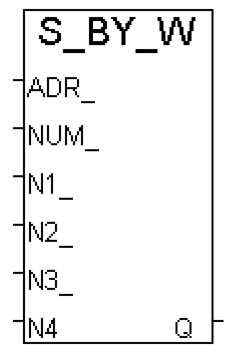
ADR_ Integer read which address, one Byte occupy 1 byte.

S256: 1 ~ 249,856 (1 ~ 16#3D000)
S512: 1 ~ 512,000 (1 ~ 16#7D000)
X607: 1 ~ 118,784 (1 ~ 16#1D000)
X608: 1 ~ 512,000 (1 ~ 16#7D000)

return:

N_ Integer The byte value been read, 0 ~ 255

S_BY_W



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG □ W-8037/8337/8737

Description:
Function Write up to 4 bytes to the volatile SRAM

Arguments:

ADR_ Integer start from which address, one byte occupy 1 byte.

S256: 1 ~ 249,856 (1 ~ 16#3D000)
S512: 1 ~ 512,000 (1 ~ 16#7D000)
X607: 1 ~ 118,784 (1 ~ 16#1D000)
X608: 1 ~ 512,000 (1 ~ 16#7D000)

NUM_ Integer how many bytes to write, 0 ~ 4
N1_~N4_ Boolean the byte value (0-255) to write

return:

Q_ Boolean Ok: TRUE, Fail: FALSE

Please refer to section 10.3

S_DL_DIS

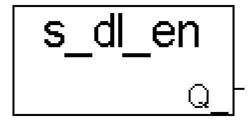
■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG □ W-8037/8337/8737

Description:
Function Disable the download permission, so that PC can not download data to the SRAM

return:

Q_	Boolean	TRUE: ok, FALSE: fail

S_DL_EN



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG □ W-8037/8337/8737

Description:
Function Enable the download permission for PC to download data to the volatile SRAM

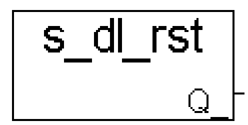
return:

Q_	Boolean	TRUE: ok, FALSE: fail

Note:
The default setting is "Disable". S_DL_EN sholud be called, then PC download data to the volatile SRAM is possible.

S_DL_RST

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG □ W-8037/8337/8737



Description:
Function Reset the Download Status to "-1:No action" for the volatile SRAM

return:

Q_	Boolean	TRUE: ok, FALSE: fail

S_DL_STS

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG □ W-8037/8337/8737

Description:
Function Get PC's Download Status for the volatile SRAM

return:

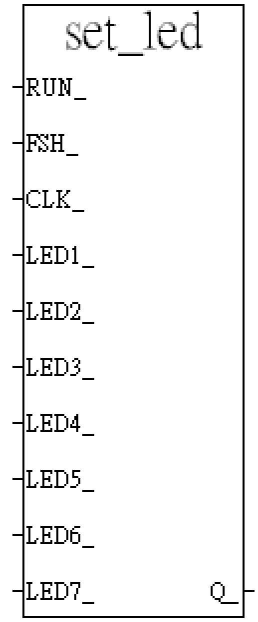
STS_	Integer	-1:	No action,
		1:	PC is Downloading data to the SRAM now
		2:	download accomplishment

Note:
S_DL_RST can be called to reset the status to -1 (reset to "No action")

Please refer to section 10.3

SET_LED

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG □ W-8037/8337/8737



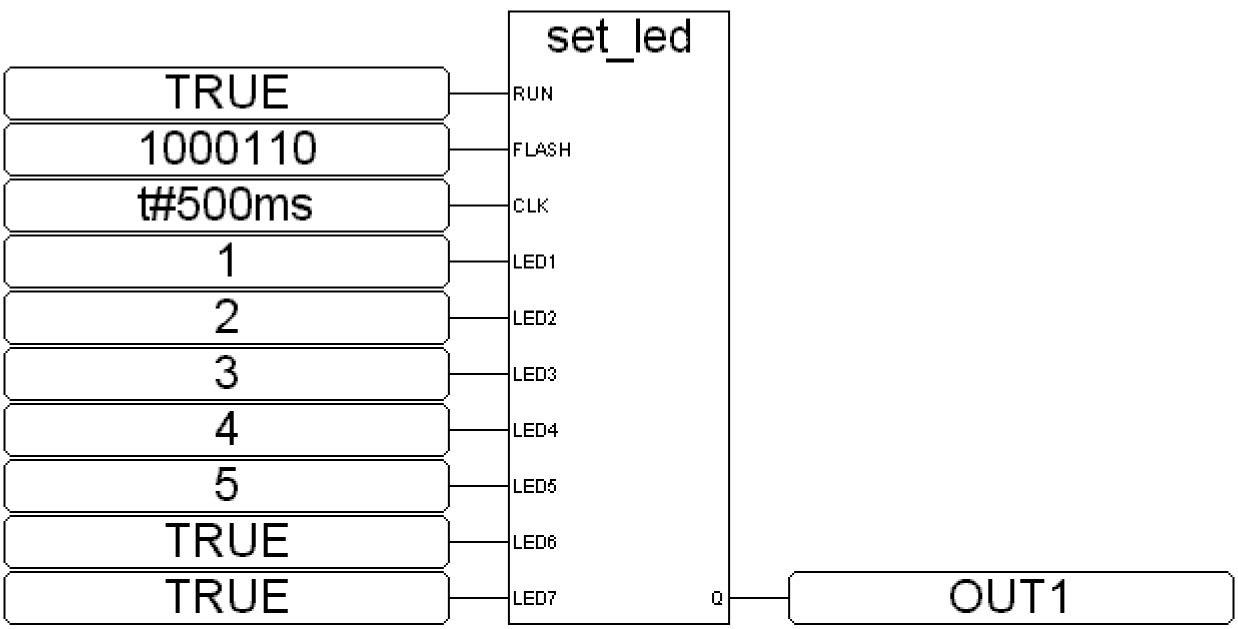
Description:
Function **Displays A Message To The S-MMI**

Arguments:

RUN_	Boolean	Set To "TRUE" To Display Message
FLASH_	Integer	Set each digit To "1" To blink each Message. Example: Set To 11 (0000011) Means The 6^{th} & 7^{th} Display Positions Will Blink. Set To 100001 (0100001) Means The 2^{nd} & 7^{th} Display Positions Will blink
CLK_	Timer	Amount Of Time For Display To blonk
LED1_	Integer	Value Of Position Display #1
LED2_	Integer	Value Of Position Display #2
LED3_	Integer	Value Of Position Display #3
LED4_	Integer	Value Of Position Display #4
LED5_	Integer	Value Of Position Display #5
LED6_	Boolean	Value Of Position Display #6
LED7_	Boolean	Value Of Position Display #7

* Refer to section A.3 to see the display char. of LED1 ~ LED5, LED6, LED7.

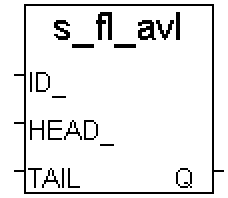
Example:



ST equivalence:

```
OUT1 := SET_LED(TRUE,1000110,t#500ms,1,2,3,4,5,TRUE,TRUE);
(* OUT1 is declared as a boolean variable *)
```

S_FL_AVL



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG □ W-8037/8337/8737

Description:
Function Set one file's current available byte No. for the volatile SRAM

Arguments:

ID_	**Integer**	File identifier No. (1 ~ 8)
HEAD_	**Integer**	The current available starting byte No.
TAIL_	**Integer**	The current available ending byte No.

(HEAD_, TAIL_) must resides inside the area of the associate file (Please refer to "S_FL_INI"), or Q_ will return FALSE

	1 or
S256:	1 ~ 249,856
S512:	1 ~ 512,000
X607:	1 ~ 118,784
X608:	1 ~ 512,000

For ex.,

A file of ID_ = 1 resides at byte No. of 1 ~ 20000 , it can store up to 20000 bytes.

1. if setting one of HEAD_ and TAIL_ to -1, no data of the file is available. It means when you load this file from PC, its size is 0 byte.

2. if setting HEAD_=1, TAIL_=1000, the current available data of the file will be at 1 ~ 1000 inside the volatile SRAM. It means when you load this file from PC, its size is 1000 bytes.

3. if setting HEAD_=10001, TAIL_=5000 : the current available data of the file will be at 10001 ~ 20000 and then continued with 1 ~ 5000 inside the volatile SRAM. It means when you load this file from PC, its size is 15000 bytes.

4. if setting HEAD_=1000, TAIL_=1000, no data of the file is available. It means when you load this file from PC, its size is 0 byte.

return:

Q_	**boolean**	TRUE: ok , FALSE: fail

Note: S_FL_INI should be called once before S_FL_AVL is called

S_FL_INI

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG □ W-8037/8337/8737

Description:
Function **Init one file's name & location for the volatile SRAM**

Arguments:

ID_	**Integer**	File identifier No. (1 ~ 8)
NAME_	**Message**	File name, up to 8 char. for the name & up to 3 char. for the extension. For ex., "data1.txt", "A1234567.bin". Valid char. are A ~ Z , a ~ z , _ , 0 ~ 9, and the 1st should be A ~ Z or a ~ z
BEGIN_	**Integer**	The begin byte No. of the file. BEGIN_ must less than END_
END_	**Integer**	The end byte No. of the file. BEGIN_ must less than END_

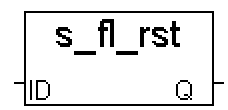
S256:	1 ~ 249,856
S512:	1 ~ 512,000
X607:	1 ~ 118,784
X608:	1 ~ 512,000

For ex.,
BEGIN_=101, END_=5000 : the file resides at 101 ~ 5000 inside the SRAM.

return:

Q_	**boolean**	TRUE: ok , FALSE: fail

S_FL_RST



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG □ W-8037/8337/8737

Description:
Function **Reset file's Status to "Not been load by PC yet" for the volatile SRAM**

Arguments:

ID_	**Integer**	File identifier No. (1 ~ 8)

return:

Q_	**Boolean**	TRUE: ok, FALSE: fail

Note:
1. S_FL_INI should be called first.
2. S_FL_STS can be called to get file's status

Please refer to section 10.3

S_FL_STS

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG □ W-8037/8337/8737

Description:
Function Get file's Status, end byte No. that has been load by PC for the volatile SRAM

Arguments:

ID_	**Integer**	File identifier No. (1 ~ 8)

return:

END_	**Integer**	The end byte No. that has been load by PC

Not been load: -1

S256:	1 ~ 249,856
S512:	1 ~ 512,000
X607:	1 ~ 118,784
X608:	1 ~ 512,000

For ex.,

A file of ID_ = 1 is located at byte No. of 1 ~ 20000 , it can store up to 20000 bytes. And its current available data is setting at 1001 ~ 10000 inside the volatile SRAM.

1. If return END_ is -1, it means PC hasn't load it yet.

2. If return END_ is 10000 (Normally the value is equal to the current available ending byte No.), it means PC has load it from 1001 ~ 10000

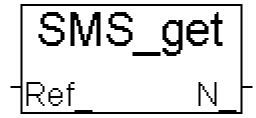
3. If return END_ is 8000, it means PC has load it from 1001 ~ 8000

Note:
1. S_FL_INI should be called first.
2. S_FL_RST can be called to reset the status to -1 (reset to "PC hasn't load it yet")

Please refer to section 10.3

SMS_GET

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8037/8337/8737



Description:
Function   Get message date and time from controller's date & time

Arguments:

REF_   **Integer**   to get what ? , 1 ~ 7

1: get year, (N_ = 2000 ~ 2099)
2: get month, (N_ = 1 ~ 12)
3: get date, (N_ = 1 ~ 31)
4: get week date,(N_ = 1 ~ 7, 7 means Sunday)
5: get hour, (N_ = 0 ~ 23)
6: get minute, (N_ = 0 ~ 59)
7: get second, (N_ = 0 ~ 59)

others: return N_=-1 : error

return:

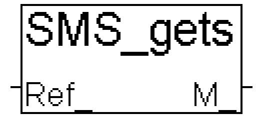
N_   **Integer**   Return associated with Ref_. If return -1, it may be "No message" or Ref_ out of range of 1 ~ 7

Note:

1. SMS_gets & SMS_get can be called to get message
2. After SMS_gets(1) is called (get message data), the message buffer will reset to "No message". So if the orther information are need, please call SMS_get(1~7) & SMS_gets(2) & SMS_gets(3) before calling SMS_gets(1)

Example: demo_43

SMS_GETS



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8037/8337/8737

Description:
Function **Get message data and other information**

Arguments:

REF_ **Integer** to get what ? , 1 ~ 3

1: get message data
2: get phone No. of sender
3: get date & time in string format

others: return M_= 'error'

return:

M_ **Message** Return associated with Ref_. If return 'error', it may be "No message" or Ref_ out of range of 1 ~ 3

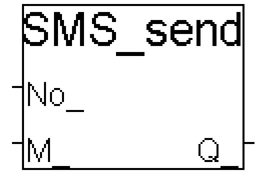
Note:

1. SMS_gets & SMS_get can be called to get message
2. After SMS_gets(1) is called (get message data), the message buffer will reset to "No message". So if the orther information are need, please call SMS_get(1~7) & SMS_gets(2) & SMS_gets(3) before calling SMS_gets(1)

Example: demo_43

SMS_SEND

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8037/8337/8737



Description:
Function **Trigger the controller to send a new message**

Arguments:

No_ **message** to which phone No. , fro ex. '+886920119135', max len is 31 digits

M_ **message** the message to send

return:

Q_ **Boolean** True: ok. ,
False: wrong phone No or "message sending status" is not 0 or 21

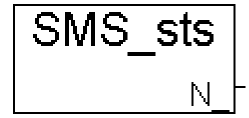
Note:

1. Please call SMS_sts to get the "Message Sending status" before calling SMS_send. SMS_send only works when status is not 1:busy

2. A successfully SMS_send request will reset the "Message sending status" to "1:busy", and after that, by the time, it will set to the associate status. For ex. 21:successfully sent

Example: demo_43

SMS_STS



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8037/8337/8737

Description:
Function **Get Message Sending status**

return:

N_ **Integer**

0: waiting for a new sending request
1: busy. (One message is processing now)
21: The message is sent successfullly

-1: SMS system is not available (Check GSM Modem & SIM card)
-2: Timeout, No response. (It May be no such a phone No.)

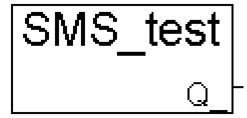
Note:

1. Please call SMS_sts to get the "Message Sending status" before calling SMS_send. SMS_send only works when status is not 1:busy

2. A successfully SMS_send request will reset the "Message sending status" to "1:busy", and after that, by the time, it will set to the associate status. For ex. 21:successfully sent

Example: demo_43

SMS_TEST



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8037/8337/8737

Description:
Function **Test if message coming or not**

return:

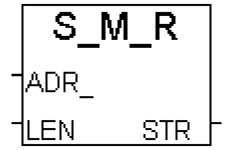
Q_	**Boolean**	TRUE: A message is coming, FALSE: No message

Note:

1. SMS_gets & SMS_get can be called to get message
2. After SMS_gets(1) is called (get message data), the message buffer will reset to "No message". So if the orther information are need, please call SMS_get(1~7) & SMS_gets(2) & SMS_gets(3) before calling SMS_gets(1)

Example: demo_43

S_M_R



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG □ W-8037/8337/8737

Description:
Function **Read one string from the volatile SRAM**
Arguments:

ADR_	**Integer**	read which address. S256: 1 ~ 249,856 (1 ~ 16#3D000) S512: 1 ~ 512,000 (1 ~ 16#7D000) X607: 1 ~ 118,784 (1 ~ 16#1D000) X608: 1 ~ 512,000 (1 ~ 16#7D000)
LEN_	**Integer**	Max length of the string to read, 0 ~ 255

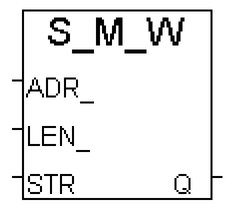
return:

STR_	**Message**	The string value been read

For ex., data in memory is 16#31, 16#32, 16#33, 16#34, 16#35, **0**, 16#37, 16#38, ...

LEN_=0 ----> STR_= '' (empty string)
LEN_=3 ----> STR_= '123'
LEN_=5 ----> STR_= '12345'
LEN_=6 ----> STR_= '12345'
LEN_=7 ----> STR_= '12345'
LEN_=100 ----> STR_= '12345'

S_M_W



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG □ W-8037/8337/8737

Description:
Function **Write one string to the volatile SRAM**

Arguments:

ADR_	**Integer**	write to which address. S256: 1 ~ 249,856 (1 ~ 16#3D000) S512: 1 ~ 512,000 (1 ~ 16#7D000) X607: 1 ~ 118,784 (1 ~ 16#1D000) X608: 1 ~ 512,000 (1 ~ 16#7D000)
LEN_	**Integer**	Max length of the string to write, 0 ~ 255.
STR_	**Message**	the string value.

For ex.

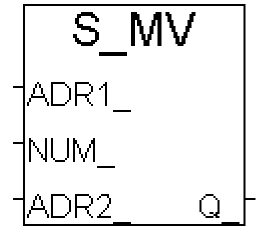
LEN_=0 , STR='12345' ----> no data written
LEN_=1 , STR='12345' ----> 16#31 (1 byte written)
LEN_=3 , STR='12345' ----> 16#31, 16#32, 16#33 (3 bytes written)
LEN_=7 , STR='12345' ----> 16#31, 16#32, 16#33, 16#34, 16#35, 0, 0 (7 bytes written)
LEN_=100 , STR='12345' --> 16#31, 16#32, 16#33, 16#34, 16#35, 0, 0, 0, … (100 bytes written)

Return:

Q_	**boolean**	Ok: TRUE, Fail: FALSE

S_MV

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG □ W-8037/8337/8737



Description:

Function **Move some bytes inside the volatile SRAM**

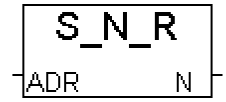
Arguments:

ADR1_	**Integer**	destination start position S256: 1 - 249856 (1 - 16#3D000) S512: 1 - 512000 (1 - 16#7D000) X607: 1 - 118784 (1 - 16#1D000) X608: 1 - 512000 (1 - 16#7D000)
NUM_	**Integer**	how many bytes to move, 0 - 512,000
ADR2_	**Integer**	Move from which starting position

return:

Q_	**boolean**	Ok: TRUE, Fail: FALSE

S_N_R



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG □ W-8037/8337/8737

Description:
Function Read one integer from the volatile SRAM

Arguments:

ADR_ **Integer** read which address, one Integer occupy 4 bytes.

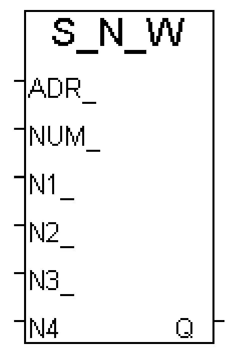
S256: 1 ~ 249,856 (1 ~ 16#3D000)
S512: 1 ~ 512,000 (1 ~ 16#7D000)
X607: 1 ~ 118,784 (1 ~ 16#1D000)
X608: 1 ~ 512,000 (1 ~ 16#7D000)

return:

N_ **Integer** The integer value been read, 32-bit, signed

The integer written in the SRAM is [Lowest byte] [2nd byte] [3rd byte] [High byte], for ex. a integer of 16#01020304, it will be saved in the SRAM as [04] [03] [02] [01]

S_N_W



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG □ W-8037/8337/8737

Description:
Function Write up to 4 integers to the volatile SRAM

Arguments:

ADR_ **Integer** start from which address,one Integer occupy 4 byte.

S256: 1 ~ 249,856 (1 ~ 16#3D000)
S512: 1 ~ 512,000 (1 ~ 16#7D000)
X607: 1 ~ 118,784 (1 ~ 16#1D000)
X608: 1 ~ 512,000 (1 ~ 16#7D000)

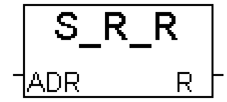
NUM_ **Integer** how many integers to write, 0 ~ 4
N1_~N4_ **Integer** the integer value (32-bit, signed) to write

The integer written in the SRAM is [Lowest byte] [2nd byte] [3rd byte] [High byte], for ex. a integer of 16#01020304, it will be saved in the SRAM as [04] [03] [02] [01]

return:

Q_ **Boolean** Ok: TRUE, Fail: FALSE

S_R_R



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG □ W-8037/8337/8737

Description:
Function Read one real value from the volatile SRAM

Arguments:

ADR_	**Integer**	read which address, one Real value occupy 4 bytes.

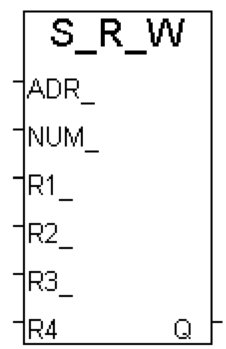
S256: 1 ~ 249,856 (1 ~ 16#3D000)
S512: 1 ~ 512,000 (1 ~ 16#7D000)
X607: 1 ~ 118,784 (1 ~ 16#1D000)
X608: 1 ~ 512,000 (1 ~ 16#7D000)

return:

R_	**Real**	The real value been read, 32-bit float

The real value written in the SRAM is [Lowest byte] [2nd byte] [3rd byte] [High byte]. For ex. Real Value of 1.23 is consists of 4 bytes --> 16#A4 , 16#70 , 16#9D , 16#3F

S_R_W



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG □ W-8037/8337/8737

Description:
Function Write up to 4 real values to the volatile SRAM

Arguments:

ADR_	**Integer**	start from which address, one Real occupy 4 byte.

S256: 1 ~ 249,856 (1 ~ 16#3D000)
S512: 1 ~ 512,000 (1 ~ 16#7D000)
X607: 1 ~ 118,784 (1 ~ 16#1D000)
X608: 1 ~ 512,000 (1 ~ 16#7D000)

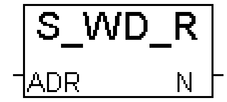
NUM_	**Integer**	how many real values to write, 0 ~ 4
R1_~R4_	**Real**	the real value (32-bit float) to write

The real value written in the SRAM is [Lowest byte] [2nd byte] [3rd byte] [High byte]. For ex. Real Value of 1.23 is consists of 4 bytes --> 16#A4 , 16#70 , 16#9D , 16#3F

return:

Q_	**Boolean**	Ok: TRUE, Fail: FALSE

S_WD_R



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG □ W-8037/8337/8737

Description:
Function Read one word from the volatile SRAM
Arguments:

ADR_ **Integer** read which address, one Word occupy 2 bytes.

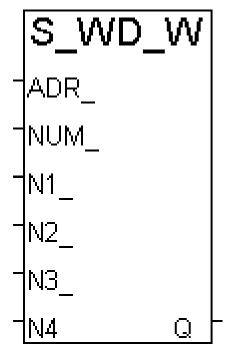
S256: 1 ~ 249,856 (1 ~ 16#3D000)
S512: 1 ~ 512,000 (1 ~ 16#7D000)
X607: 1 ~ 118,784 (1 ~ 16#1D000)
X608: 1 ~ 512,000 (1 ~ 16#7D000)

return:

N_ **Integer** The word value been read, -32768 ~ +32767

The word written in the SRAM is [Low byte] [High byte], for ex. a integer of 16#0102, it will be saved in the SRAM as [02] [01]

S_WD_W



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG □ W-8037/8337/8737

Description:
Function Write up to 4 words to the volatile SRAM

Arguments:

ADR_ **Integer** start from which address, one Word occupy 2 bytes.

S256: 1 ~ 249,856 (1 ~ 16#3D000)
S512: 1 ~ 512,000 (1 ~ 16#7D000)
X607: 1 ~ 118,784 (1 ~ 16#1D000)
X608: 1 ~ 512,000 (1 ~ 16#7D000)

NUM_ **Integer** how many words to write, 0 ~ 4
N1_~N4_ **Integer** the word value (-32768 ~ 32767) to write

The word written in the SRAM is [Low byte] [High byte], for ex. a integer of 16#0102, it will be saved in the SRAM as [02] [01]

return:

Q_ **Boolean** Ok: TRUE, Fail: FALSE

STR_REAL

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8037/8337/8737

Description:
Function **Convert a string to Real value**

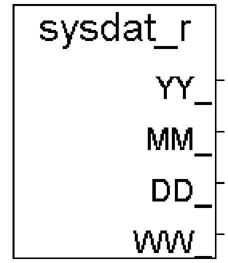
Arguments:

STR_	message	For ex, '123.456' , '-0.2345' , ' +2.13E10' , ' 15.2345E-2'
REAL_	real	The real value retured. If REAL_ is 1.23E-20 , it means STR_ is a wrong setting. For ex, if STR_=’ 123.AB’ or ‘23-45.17’ or ‘1.2.345’, REAL_ will return 1.23E-20

Note:
"REAL_STR" can be used to convert real value to a string

SYSDAT_R

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8037/8337/8737



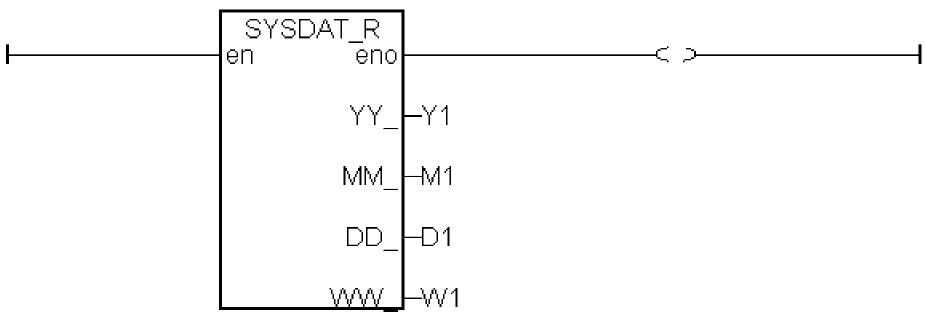
Description:
Function block Read system year, month, day and date.

Arguments:

YY_	Integer	Year Returned (Example: 2002, 2003, 2010, Etc.)
MM_	Integer	Month Returned (1 = Jan., 3 =March, 10 = October, Etc.)
DD_	Integer	Day Returned, Valid Range From 1 To 31
WW_	Integer	Date Returned (1 = Monday, 4 = Thursday, 7 = Sunday, Etc.)

Example: refer to demo_03.

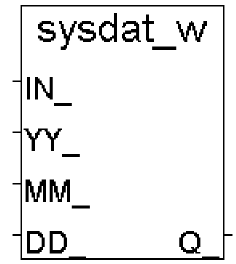
Y1, M1, D1 and W1 are declared as integer variables.



ST equivalence:

```
DAT_R1( );                  (* call DAT_R1 *)
Y1 := DAT_R1.YY_ ;          (* get year *)
M1 := DAT_R1.MM_ ;          (* get month *)
D1 := DAT_R1.DD_ ;          (* get day *)
W1 := DAT_R1.WW_ ;          (* get date *)
(* DAT_R1 is declared as FB instance with typed - SYSDAT_R *)
```

SYSDAT_W



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8037/8337/8737

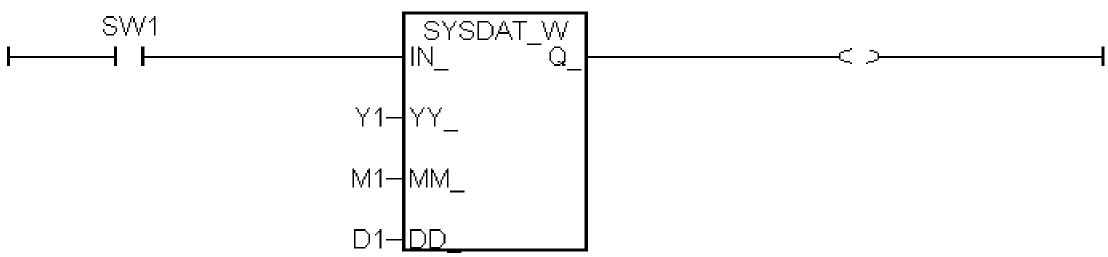
Description:
Function block **Set system year, month and day.**

Arguments:

IN_	Boolean	Set System Date When Rising From "FALSE" To "TRUE"
YY_	Integer	Year To Write (Example: 2002, 2003, 2010, Etc.)
MM_	Integer	Month To Write (1 = Jan., 3 = March, 10 =October, Etc.)
DD_	Integer	Day Returned, Valid Range From 1 To 31
Q_	Boolean	If "OK", Returns "TRUE"

Example: refer to demo_03.

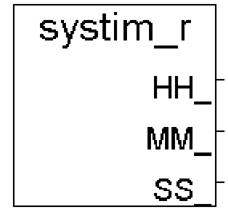
SW1 is declared as a boolean variable. Y1, M1, D1 are declared as integer variables.



St equivalence:

```
DAT_W1( SW1, Y1, M1, D1);             (* call DAT_W1 *)
OUT1 := DAT_W1.Q_ ;                   (* get return value *)
(* DAT_W1 is declared as a FB instance with type - SYSDAT_W *)
(* OUT1 as a boolean variable *)
```

SYSTIM_R



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8037/8337/8737

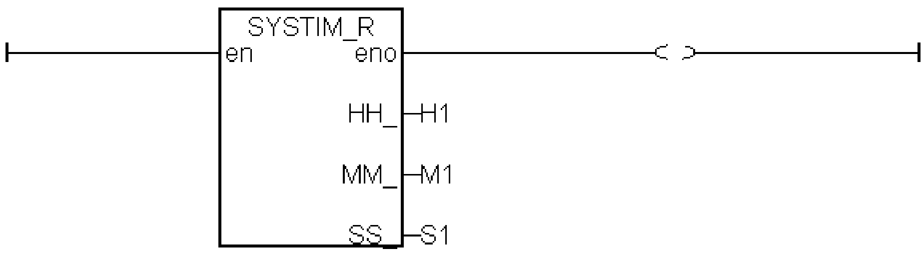
Description:
Function block Read system hour, minute and second.

Arguments:

HH_	Integer	Hour Returned (Valid Range From 0 To 23)
MM_	Integer	Minute Returned (Valid Range From 0 To 59)
SS_	Integer	Second Returned (Valid Range From 0 To 59)

Example: refer to demo_03, demo_15b.

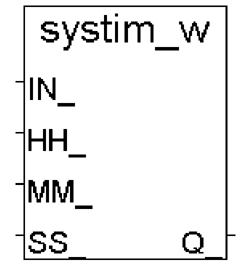
H1, M1 and S1 are declared as integer variables.



ST equivalence:

```
(* TIM_R1 is declared as FB instance with type - SYSTIM_R *)
TIM_R1( );                    (* Call TIM_R1 *)
H1 := TIM_R1.HH_ ;            (* get hour *)
M1 := TIM_R1.MM_ ;            (* get minute *)
S1 := TIM_R1.SS_ ;            (* get second *)
```

SYSTIM_W



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8037/8337/8737

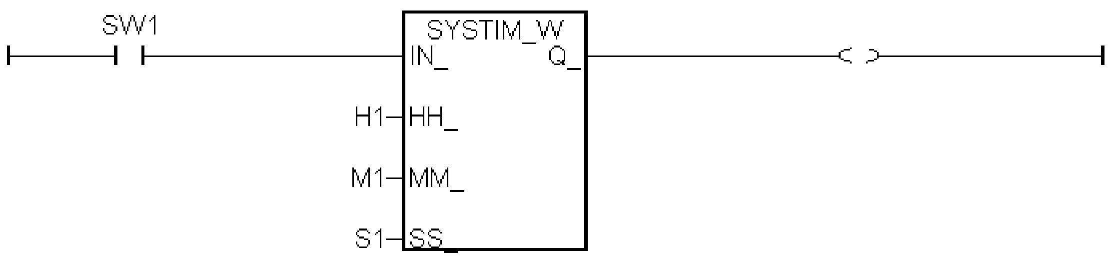
Description:
Function block Set system hour, minute and second.

Arguments:

IN_	Boolean	Set System Date When Rising From "FALSE" To "TRUE"
HH_	Integer	Hour To Write, 0 - 23
MM_	Integer	Minute to Write, 0 - 59
SS_	Integer	Second to write, 0 - 59
Q_	Boolean	If "OK", Returns "TRUE"

Example: refer to demo_03.

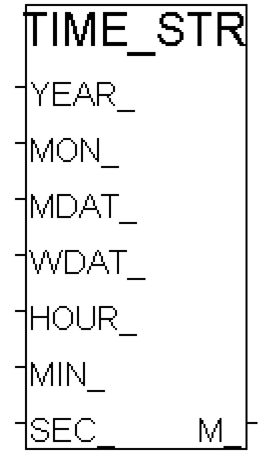
SW1 is declared as a boolean variable. H1, M1, S1 are declared as integer variables.



St equivalence:

```
TIM_W1( Sw1,H1,M1,S1);        (* call TIM_W1 *)
OUT1 := TIM_W1.Q_ ;                        (* get return value *)
(* TIM_W1 is declared as a FB instance with type - SYSTIM_W *)
(* OUT1 as a boolean variable *)
```

TIME_STR



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8037/8337/8737

Description:
Function **Convert date & time to string format**

Arguments:

YEAR_	**integer**	year, 2000 ~
MON_	**integer**	month, 1 ~ 12 (January ~ December)
MDAY_	**integer**	mday, 1 ~ 31
WDAY_	**integer**	wday, 1 ~ 7 (Monday ~ Sunday)
HOUR_	**integer**	hour, 0 ~ 23
MIN_	**integer**	minute, 0 ~ 59
SEC_	**integer**	second, 0 ~ 59

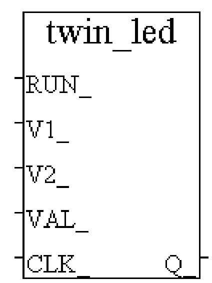
If given wrong input parameters will return M_ = '' (empty string). For. ex. give MON_=14

return:

M_ **message** length is 24 characters. For ex. 'Feb/18/2003,13:25:45,Tue'

Note: Please use sysdat_r & systim_r to get system date & time

TWIN_LED



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG □ W-8037/8337/8737

Description:
Function **show a 2 screen values to the S-MMI**

Arguments:

RUN_	boolean	to show if TRUE
V1_	integer	value displayed on the 2 digits on left of 1st screen, 0 ~ 99
V2_	integer	value displayed on the 2 digits on right of 1st screen, 0 ~ 99
VAL_	integer	value displayed on the 2nd screen, -99999 ~ 99999
CLK_	timer	the blinking period of these 2 screens
Q_	boolean	always TRUE

Example: refer to demo_10.

VAL_HEX

■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8037/8337/8737

Description:

Function **Convert an integer to a fixed-length hexa-message**

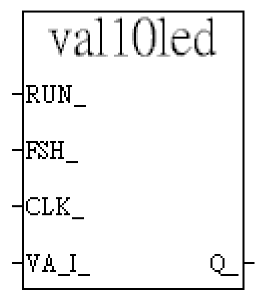
Arguments:

- **VAL_** integer the value to be converted
- **DIGIT_** integer number of digits of HEX_ , valid values are 1 ~ 8. Given others will do no conversion and force HEX_ to ' ' (empty message)
- **HEX_** message the hex-message after conversion

Example:

val_hex(100,3) ---> '064'
val_hex(192,4) ---> '00C0'
val_hex(4589,2) ---> 'ED' ('11ED', DIGIT_ is 2, force '11' trucated)
val_hex(4589,9) ---> ' ' (DIGIT_ > 8, output ' ')
val_hex(-2,8) ---> 'FFFFFFFE'

VAL10LED



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG □ W-8037/8337/8737

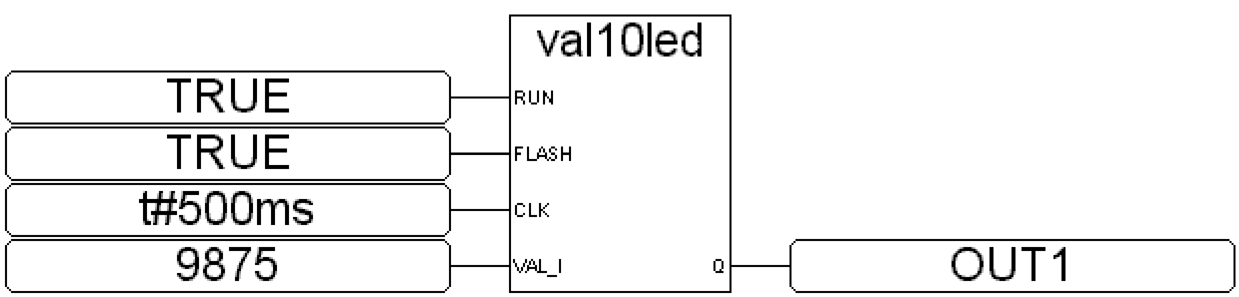
Description:

Function **disply an decimal integer on the S-MMI**

Arguments:

- **RUN_** Boolean if TRUE, display it
- **FLASH_** Boolean if TRUE, blink it
- **CLK_** Timer the blinking period
- **VAL_I_** Integer the integer to be displayed
  Range from -9999 to +99999
- **Q_** Boolean always returns TRUE。

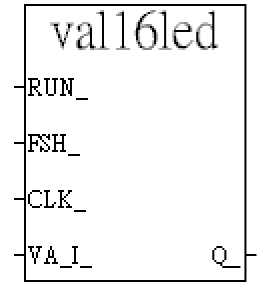
Example: refer to demo_07, demo_11b.



ST equivalence:

out1 := VAL10LED(TRUE,TRUE,t#500ms,9875);
(* out1 is declared as a boolean variable *)

VAL16LED



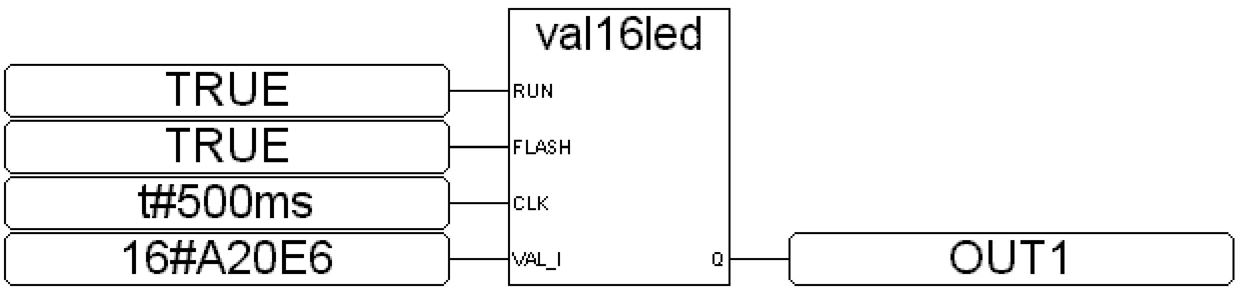
■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG □ W-8037/8337/8737

Description:
Function **display an hexadecimal integer on S-MMI**

Arguments:

RUN_	Boolean	if TRUE, display it
FLASH_	Boolean	if TRUE, blink it
CLK_	Timer	the blinking period
VAL_I_	integer	the value to be displayed Valid range from 16#0 to 16#FFFFF
Q_	Boolean	always return TRUE

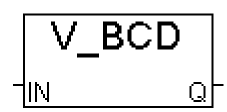
Example:



ST equivalence:

OUT1 := VAL10LED(TRUE,FALSE,t#500ms,16#A20E6);
(* OUT1 is declared as a boolean variable *)

V_BCD



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8037/8337/8737

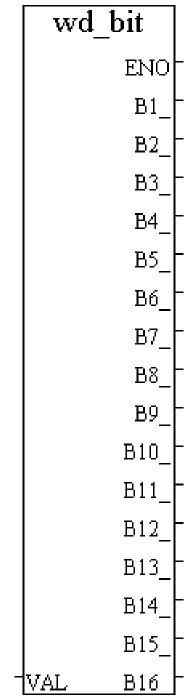
Description:
Function **Convert value to BCD value**

Arguments:

IN_	integer	the value to be converted
Q_	integer	the returned BCD value, For ex.

12345 → 16#12345
16 → 22 (16#16)

WD_BIT



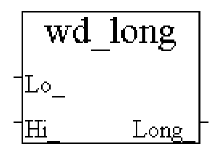
■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8037/8337/8737

Description:
Function block **Convert a word value to 16 boolean values**

Arguments:

VAL_	integer	the word to be converted.
ENO	boolean	no usage, don't care about it.
B1_ ~ B16_	boolean	the 16 boolean values after converted For ex. If VAL_ is 4, B3_ will be TRUE and others will be FALSE. If VAL_ is 3, B1_ and B2_ will be TRUE and others will be FALSE.

WD_LONG



■ I-8417/8817 ■ I-8437/8837 ■ I-7188EG ■ I-7188XG ■ W-8037/8337/8737

Description:
Function **Convert two words to one long integer**

Arguments:

Lo_	integer	Low word (only the lowest 16-bit is used)
Hi	integer	High word (only the lowest 16-bit is used)
Long_	integer	the 32-bit integer composed by Lo_ and Hi_ word

Example:

Lo_	Hi_	--->	Long_
-32768 (8000)	-1 (FFFF)	--->	-32768 (FFFF 8000)
-1 (FFFF)	-1 (FFFF)	--->	-1 (FFFF FFFF)
-32768 (8000)	0 (0000)	--->	+32768 (0000 8000)
100 (0064)	4103 (1007)	--->	+ 268 894 308 (1007 0064)

Appendix B: Setting The IP, Mask & Gateway Address of The I-8437/8837 & I-7188EG Controllers

This document describe the proper way to set the IP address, address mask
and gateway address of the I-8437/8837 & I-7188EG controllers.

****
EACH I-8437/ 8837 or I-7188EG USES TCP/IP PORT NO. 502 TO TALK TO THE HMI AND ISAGRAF WORKBENCH. A MAX. NUMBER OF 4 PCS CAN TALK TO THE I-8437/8837 or I-7188EG THROUGH MODBUS TCP/IP PROTOCOL.
****

1. Create a file folder named "8000" in your hard drive.
   For example, "c:\8000".

For Windows NT, Windows 2000 & Windows XP Users:

2. Copy \Napdos\ISaGRAF\8000\Driver\7188xw.exe, 7188xw.ini
   from the CD_ROM into your "8000" folder.
3. Run "\8000\7188xw.exe" in your hard drive. A "7188xw" screen
   will appear.

For Dos, Windows 95 & Windows 98 Users:

2. Copy \Napdos\ISaGRAF\8000\Driver\7188x.exe, 7188x.ini
   from the CD_ROM into your "8000" folder.
3. Run "\8000\7188x.exe" in your hard drive. A "7188x" screen
   will appear.

4. Link from COM1 or COM2 of your PC to COM1 of the I-8437/8837 (or I-7188EG)
   controller by a RS232 cable.

5. Power off the I-8437/8837 (or I-7188EG) controller, connect pin "INIT" to
   "INIT COM" (GND for I-7188EG), and then power it up.

6. If the connection is Ok, messages will appear on the
   7188x screen.

```
****************************************
****  7188x Ver. 1.01.0 02/23/2000  ****
***       Press F1 for help.       ***
****************************************
ICP_DAS MiniOS7 for 8000-485 Ver. 1.03 build 014,May 09 2001 14:30:36
SRAM:512K, FLASH MEMORY:512K
Serial number= 5A 5A 5A 5A 5A 5A 5A 5A
8000>
```

7. Type "ip" to see the current IP address of the I-8437/8837 (or I-7188EG).

```
8000> ip
IP=192.168.255.255
8000>
```

8. Type "setip xxx.xxx.xxx.xxx" to set to a new IP address.

```
8000> setip 192.168.1.200
Set IP=192.168.1.200
[ReadBack]IP=192.168.1.200
8000>
```

9. Type "mask" to see the current address mask of the I-8437/8837 (or I-7188EG).

```
8000> mask
MASK=255.255.0.0
8000>
```

10. Type "setmask xxx.xxx.xxx.xxx" to set to a new address mask.

```
8000> setmask 255.255.255.0
Set MASK=255.255.255.0
[ReadBack]MASK=255.255.255.0
8000>
```

11. Type "gateway" to see the current gateway address.

```
8000> gateway
Gateway=192.168.0.1
8000>
```

12. Type "setgateway xxx.xxx.xxx.xxx" to set to a new gateway address.

```
8000> setgateway 192.168.1.1
Set GATEWAY=192.168.1.1
[ReadBack]Gateway=192.168.1.1
8000>
```

13. Press ALT_X to exit "7188x" and close the DOS SHELL, or COM1/COM2 of the PC will be occupied.

14. Remove the connection between "INIT" - "INIT COM", reset the I-8437 / 8837 (or I-7188EG) controller.

Appendix C: Update The I-8417 / 8817 / 8437 / 8837 Controller to New Hardware Driver

The ISaGRAF embedded driver is firmware burned into the flash memory of the I-8417 / 8817 / 8437 / 8837 controller. It can be easily upgraded by the user.

Please refer to the respective "Getting Started" Manual for Updating driver of the I-7188EG, I-7188XG & Wincon-8xx7.

Our newly released driver can also be obtained from the following website.
    http://www.icpdas.com/products/8000/isagraf.htm

Warning:
The copyright of the firmware and the ISaGRAF embedded driver belongs to ICP DAS CO., LTD. Only the I-8417, 8817, 8437 and 8837 have registered a legal ISaGRAF Target license. To burn an ISaGRAF embedded driver into other controllers is absolutely illegal and may be punished by law.

Make sure of your current OS & driver version before you upgrade it.

1. Create a file folder named "8000" in your hard drive. For example, "c:\8000".

*** We use driver 2.50 as an example in this document.

For Windows NT, Windows 2000, Windows XP users:
2. Copy \Napdos\ISaGRAF\8000\Driver\2.50\"7188xw.exe", "7188xw.ini" , "isa.exe" , "autoexec.bat" & "8k031105.IMG" from the CD_ROM into your "8000" folder.
3. Run "\8000\7188xw.exe" in your hard drive. A "7188xw" screen will appear (Press F1 for help).

For Dos, Windows 95, 98 users:
2. Copy \Napdos\ISaGRAF\8000\Driver\2.50\"7188x.exe", "7188x.ini" , "isa.exe" , "autoexec.bat" & "8k031105.IMG" from the CD_ROM into your "8000" folder.
3. Run "\8000\7188x.exe" in your hard drive. A "7188x" screen will appear.

4. Link COM1 or COM2 of your PC to COM1 of the I-8xx7 controller through a RS232 cable.

5. Power off the I-8xx7 controller, connect pin "INIT" to "INIT COM" and then power it up.

6. If the connection is Ok, messages will appear on the 7188x screen.
    8000>

7. Type "ver" to see the current OS version.
    8000> ver

8. Type "isa *p=" to see the version No. & COMM setting of the ISaGRAF driver
    8000> isa *p=

To upgrade an ISaGRAF embedded driver, follow the following steps.

9. Power off the I-8xx7 controller, connect pin "INIT" to "INIT COM" and then power it up.

10. The OS image should upgrade first. Type "upload" to load the OS image
    8000> **upload**

    press at **ALT+E** and type in **the image name** (for version 2.50 - 8k031105.IMG)

    and then type "bios1"

    8000> **bios1**
    (WAIT ABOUT 30 SEC. ***DO NOT REMOVE THE POWER IN THESE 30 SEC.***)

11. To upgrade the ISaGRAF driver. Type "del" and reply "y" to delete the current driver.
    8000> del
    Total File number is 2, do you really want to delete(y/n)? **y**

12. Type "load", then press ALT_E and then type "autoexec.bat" .
    8000> **load**
    File will save to 8000:0000
    StartAddr-->7000:FFFF
    Press **ALT_E** to download file!
    Input filename:**autoexec.bat**

13. Type "load" again, then press ALT_E and then type "isa.exe". Wait util it finished.
    8000> **load**
    File will save to 8003:0002
    StartAddr-->8000:0031
    Press **ALT_E** to download file!
    Input filename:**isa.exe**

14. Type "dir" to make sure "autoexec.bat" and "isa.exe" are well burned.
    8000> dir

15. Press ALT_X to exit "7188x".

16. Remove the connection between "INIT" - "INIT COM", reset the I-8xx7 controller.

Appendix C.1: Setting I-8xx7 & I-7188EG's COM1 As None-Modbus-Slave port

COM1 of the I-8417/8817/8437/8837, I-7188EG supports Modbus RTU Slave protocol by default. User may change it to a None-Modbus-Slave port for other usage. For example, user may write his own defined protocol on COM1 or use COM1 as a Modbus Master port.

1. Create a file folder named "8000" in your hard drive.
   For example, "c:\8000".

For Windows NT, Windows 2000 & Windows XP Users:
2. Copy CD-ROM: \Napdos\ISaGRAF\8000\Driver\7188xw.exe, 7188xw.ini
   from the CD_ROM into your "8000" folder.
3. Run "\8000\7188xw.exe" in your hard drive. A "7188xw" screen will appear.

For Dos, Windows 95 & Windows 98 Users:
2. Copy CD-ROM: \Napdos\ISaGRAF\8000\Driver\7188x.exe, 7188x.ini
   from the CD_ROM into your "8000" folder.
3. Run "\8000\7188x.exe" in your hard drive. A "7188x" screen will appear.

4. Link from COM1 or COM2 of PC to COM1 of the I-8417/8817/8437/8837 (or I-7188EG) by a RS232 cable.

5. Power off the I-8417/8817/8437/8837 (or I-7188EG), connect pin "INIT" to "INIT COM", then power it up.

6. If the connection is Ok, messages will appear on the 7188x screen.

```
8000>
```

7. Type "**isa *f=**1" to free COM1 (set COM1 as none-Modbus-Slave port)
   (For I-7188EG, type "**isa7188e *f=1**")

```
8000> isa *f=1
```

8.Press ALT_X to exit "7188x", or COM1/COM2 of the PC will be occupied.

9. Remove the connection between "INIT" - "INIT COM", recycle the power of the controller.

Important Note:
If user wants COM1 to be back to a Modbus RTU Slave port again, follow the same step 1 to 6 & then type "isa *f=0" as below. (For I-7188EG, type "**isa7188e *f=0**"))

```
8000> isa *f=0
```

Appendix D: Table of The Analog IO Value

I-87013, I-7013, I-7033

Range Code (Hex)	RTD Type	Data Format	Max Value	Min Value
20 (Default)	Platinum 100 a = 0.00385	Input Range (Celsius)	+100.0	-100.0
		Decimal Value	+32767	-32768
		2's complement HEX	7FFF	8000
21	Platinum 100 a = 0.00385	Input Range (Celsius)	+100.0	+0.0
		Decimal Value	+32767	+0
		2's complement HEX	7FFF	0000
22	Platinum 100 a = 0.00385	Input Range (Celsius)	+200.0	+0.0
		Decimal Value	+32767	+0
		2's complement HEX	7FFF	0000
23	Platinum 100 a = 0.00385	Input Range (Celsius)	+600.0	+0.0
		Decimal Value	+32767	+0
		2's complement HEX	7FFF	0000
24	Platinum 100 a = 0.003916	Input Range (Celsius)	+100.0	-100.0
		Decimal Value	+32767	-32768
		2's complement HEX	7FFF	8000
25	Platinum 100 a = 0.003916	Input Range (Celsius)	+100.0	+0.0
		Decimal Value	+32767	+0
		2's complement HEX	7FFF	0000
26	Platinum 100 a = 0.003916	Input Range (Celsius)	+200.0	+0.0
		Decimal Value	+32767	+0
		2's complement HEX	7FFF	0000
27	Platinum 100 a = 0.003916	Input Range (Celsius)	+600.0	+0.0
		Decimal Value	+32767	+0
		2's complement HEX	7FFF	0000
28	Nickel 120	Input Range (Celsius)	+100.0	-80.0
		Decimal Value	+32767	-262140
		2's complement HEX	7FFF	999A
29	Nickel 120	Input Range (Celsius)	+100.0	+0.0
		Decimal Value	+32767	+0
		2's complement HEX	7FFF	0000
2A	Platinum 1000 a = 0.00385	Input Range (Celsius)	+600.0	-200.0
		Decimal Value	+32767	-10922
		2's complement HEX	7FFF	D556

I-8017H

* Each channel can be configured to different range ID

Range Code (Hex)	Data Format	Max value	Min value
05	Input Range	+2.5 V	-2.5 V
	Decimal Value	+32767	-32768
	2's Complement HEX	7FFF	8000
06	Input Range	+20.0 mA	-20.0 mA
	Decimal Value	+32767	-32768
	2's Complement HEX	7FFF	8000
07	Input Range	+1.25 V	-1.25 V
	Decimal Value	+32767	-32768
	2's Complement HEX	7FFF	8000
08 (Default)	Input Range	+10.0 V	-10.0 V
	Decimal Value	+32767	-32768
	2's Complement HEX	7FFF	8000
09	Input Range	+5.0 V	-5.0 V
	Decimal Value	+32767	-32768
	2's Complement HEX	7FFF	8000

I-87017, I-7017

Range Code (Hex)	Data Format	Max value	Min value
08 (Default)	Input Range	+10.0 V	-10.0 V
	Decimal Value	+32767	-32768
	2's Complement HEX	7FFF	8000
09	Input Range	+5.0 V	-5.0 V
	Decimal Value	+32767	-32768
	2's Complement HEX	7FFF	8000
0A	Input Range	+1.0 V	-1.0 V
	Decimal Value	+32767	-32768
	2's Complement HEX	7FFF	8000
0B	Input Range	+500.0 mV	-500.0 mV
	Decimal Value	+32767	-32768
	2's Complement HEX	7FFF	8000
0C	Input Range	+150.0 mV	-150.0 mV
	Decimal Value	+32767	-32768
	2's Complement HEX	7FFF	8000
0D	Input Range (with 125 ohms resistor)	+20.0 mA	-20.0 mA
	Decimal Value	+32767	-32768
	2's Complement HEX	7FFF	8000

I-87018, I-7011, I-7018

Range Code (Hex)	Data Format	Max value	Min value
00	Input Range	-15.0 mV	-15.0 mV
	Decimal Value	+32767	-32768
	2's Complement HEX	7FFF	8000
01	Input Range	+50.0 mV	-50.0 mV
	Decimal Value	+32767	-32768
	2's Complement HEX	7FFF	8000
02	Input Range	+100.0 mV	-100.0 mV
	Decimal Value	+32767	-32768
	2's Complement HEX	7FFF	8000
03	Input Range	+500.0 mV	-500.0 mV
	Decimal Value	+32767	-32768
	2's Complement HEX	7FFF	8000
04	Input Range	+1.0 V	-1.0 V
	Decimal Value	+32767	-32768
	2's Complement HEX	7FFF	8000
05 (Default)	Input Range	+2.5V	-2.5V
	Decimal Value	+100.00	-100.00
	2's Complement HEX	7FFF	8000
06	Input Range	+20.0 mA	-20.0 mA
	Decimal Value	+32767	-32768
	2's Complement HEX	7FFF	8000

Range Code (Hex)	Thermocouple Type	Data Format	Max Value	Min Value
0E	J Type	Input Range (Celsius)	+760.0	-210.0
		Decimal Value	+32767	-9054
		2's Complement HEX	7FFF	DCA2
0F	K Type	Input Range (Celsius)	+1372.0	-270.0
		Decimal Value	+32767	-6448
		2's Complement HEX	7FFF	E6D0
10	T Type	Input Range (Celsius)	+400.0	-270.0
		Decimal Value	+32767	-22118
		2's Complement HEX	7FFF	A99A

11	E Type	Input Range (Celsius)	+1000.0	-270.0
		Decimal Value	+32767	-8847
		2's Complement HEX	7FFF	DD71
12	R Type	Input Range (Celsius)	+1768.0	+0.0
		Decimal Value	+32767	+0
		2's Complement HEX	7FFF	0000
13	S Type	Input Range (Celsius)	+1768.0	+0.0
		Decimal Value	+32767	+0
		2's Complement HEX	7FFF	0000
14	B Type	Input Range (Celsius)	+1820.0	+0.0
		Decimal Value	+32767	+0
		2's Complement HEX	7FFF	0000
15	N Type	Input Range (Celsius)	+1300.0	-270.0
		Decimal Value	+32767	-6805
		2's Complement HEX	7FFF	E56B
16	C Type	Input Range (Celsius)	+2320.0	+0.0
		Decimal Value	+32767	+0
		2's Complement HEX	7FFF	0000
17	L Type	Input Range (Celsius)	+800.0	-200.0
		Decimal Value	+32767	-8192
		2's Complement HEX	7FFF	E000
18	M Type	Input Range (Celsius)	+100.0	-200.0
		Decimal Value	+16384	-32768
		2's Complement HEX	4000	8000
19	L Type DIN43710	Input Range (Celsius)	+900.0	-200.0
		Decimal Value	+32767	-7281
		2's Complement HEX	7FFF	E38F

I-7021

Range Code (Hex)	Data Format	Max Value	Min Value
30	Output Range	+20.0 mA	+0.0 mA
	Decimal Value	+32767	+0
	2's complement HEX	7FFF	0000
31	Output Range	+20.0 mA	+4.0 mA
	Decimal Value	+32767	+0
	2's complement HEX	7FFF	0000
32 (Default)	Output Range	+10.0 V	+0.0 V
	Decimal Value	+32767	+0
	2's complement HEX	7FFF	0000

I-7022

Range Type (Hex)	Data Format	Max Value	Min Value
0	Output Range	+20.0 mA	+0.0 mA
	Decimal Value	+32767	+0
	2's complement HEX	7FFF	0000
1	Output Range	+20.0 mA	+4.0 mA
	Decimal Value	+32767	+0
	2's complement HEX	7FFF	0000
2 (Default)	Output Range	+10.0 V	+0.0 V
	Decimal Value	+32767	+0
	2's complement HEX	7FFF	0000

I-8024

* Each channel can be configured to different range ID

Range Code (Hex)	Data Format	Max Value	Min Value
30	Output Range	+20.0 mA	+0.0 mA
	Decimal Value	+32767	+0
33	Output Range	+10.0 V	-10.0 V
	Decimal Value	+32767	-32768

I-87024, I-7024

Range Code (Hex)	Data Format	Max Value	Min Value
30	Output Range	+20.0 mA	+0.0 mA
	Decimal Value	+32767	+0
31	Output Range	+20.0 mA	+4.0 mA
	Decimal Value	+32767	+0
32	Output Range	+10.0 V	+0.0 V
	Decimal Value	+32767	+0
33 (Default)	Output Range	+10.0 V	-10.0 V
	Decimal Value	+32767	-32768
34	Output Range	+5.0 V	+0.0 V
	Decimal Value	+32767	+0
35	Output Range	+5.0 V	-5.0 V
	Decimal Value	+32767	-32768

Appendix E: LANGUAGE REFERENCE

ISaGRAF

Version 3.46

LANGUAGE REFERENCE

AlterSys Inc.

Published in Canada by **AlterSys Inc**.

E.1 Project architecture

An ISaGRAF project is divided into several programming units called **programs**. The programs of the project are linked together in a tree-like architecture. Programs can be described using any of **SFC, FC (Flow Chart)**, **FBD**, **LD, ST** or **IL** graphic or literal languages.

E.1.1 Programs

A **program** is a logical programming unit, which describes operations between **variables** of the process. Programs describe either **sequential** or **cyclic** operations. Cyclic programs are executed at each target system cycle. The execution of sequential programs follows the dynamic rules of either the **SFC** language or the **FC** language.

Programs are linked together in a hierarchy tree. Programs placed on the top of the hierarchy are activated by the system. Sub-programs (lower level of the hierarchy) are activated by their father. A program can be described with any of the available graphic or literal following languages:

Sequential Function Chart (SFC) for high level programming
Flow Chart (FC) for high level programming
Function Block Diagram (FBD) for cyclic complex operations
Ladder Diagram (LD) for boolean operations only
Structured Text (ST) for any cyclic operations
Instruction List (IL) for low level operations

The same program cannot mix several languages, except LD and FBD can be combined in one diagram.

E.1.2 Cyclic and sequential operations

The hierarchy of programs is divided into four main **sections** or groups:

Begin	programs executed at the beginning of each target cycle
Sequential	programs following SFC or FC dynamic rules
End	programs executed at the end of each target cycle
Functions	set of non-dedicated sub-programs

Programs of the **'Begin'** or **'End'** sections describe cyclic operations, and are not time dependent. Programs of the '**Sequential**' section describe sequential operations, where the time variable explicitly synchronises basic operations. Main programs of the **'Begin'** section are systematically executed at the beginning of each run time cycle. Main programs of the **'End'** section are systematically executed at the end of each run time cycle. Main programs of the **'Sequential'** section are executed according to either the **SFC** or the **FC** dynamic rules.

Programs of the "**Functions**" section are sub-programs that can be called by any other program in the project. A program of the "**Function**" section can call another program of this section.

Main and child programs of the sequential section must be described with **SFC** or **FC** language. Programs of cyclic sections (**begin** and **end**) cannot be described with **SFC** or **FC** language. Any program of any section may own one or more sub-programs. Any program of the sequential section may own one or more **SFC** or **FC** child programs (according to its own programming language). Sub-programs cannot be described with **SFC** or **FC** language.

Programs of the **Begin** section are typically used to describe preliminary operations on input devices to build high level filtered variables. Such variables are frequently used by the programs of the **Sequential** section. Programs of the **End** section are typically used to describe security operations on the variables operated on by the **Sequential** section, before sending values to output devices.

E.1.3 Child SFC and FC programs

Any **SFC** program of the sequential section may control other **SFC** programs. Such low-level programs are called **child SFC programs**. A **child SFC program** is a parallel program that can be started, killed, frozen or restarted by its parent program. The parent program and child program must both be described with the **SFC** language. A child SFC program may have local variables and defined words.

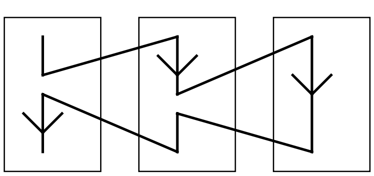
When a parent program starts a child **SFC** program, it puts an SFC **token** (activates) into each initial step of the child program. This command is described with the **GSTART** statement. When a parent program kills a child **SFC** program, it clears all the tokens existing in the **steps** of the child. Such a command is described with the **GKILL** statement.

When a parent program freezes a child **SFC** program, it suspends its execution. The suspended program can then be restarted using the **GRST** statement.

Any **FC** program of the sequential section may control other **FC** sub-programs. An **FC** father program is blocked (waits) during execution of an FC sub-program. It is not possible that simultaneous operations are done in father FC program and one of its FC sub-programs.

E.1.4 Functions and sub-programs

A sub-program or a function execution is driven by its parent program. The execution of the parent program is suspended until the sub-program or the function ends:



mainsub-programs

Any program of any section may have one or more sub-programs. A sub-program is owned by only one father program. A sub-program may have local variables and defines. Any language but **SFC** or **FC** can be used to describe a sub-program. Programs of the "**Functions**" section are sub-programs that can be called by any other program in the project. Unlike other sub-programs, they are not dedicated to one father program. A program of the "**Function**" section can call another program of this section. A function can be located in the Library.

Warning: The ISaGRAF system does not support **recursive function calls**. A run time error will occur if a program of the "**Functions**" section is called by itself or by one of its called sub-program.
Warning: A function or sub-program does not "store" the local value of its local variables. A function or sub-program is not instantiated and so can not call function blocks.

The interface of a sub-program must be explicitly defined, with a **type** and a **unique name** for each of its calling or return parameter. In order to support the **ST** language convention, the return parameter must have the same name as the sub-program.

The following table shows how to set the value of the return parameter in the body of a sub-program, in the different languages:

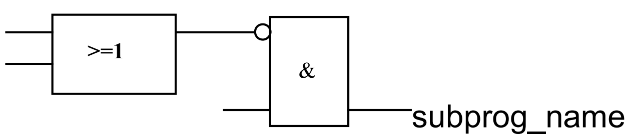
ST: assign the return parameter using its name
(the same name as the sub-program):

```
subprog_name := <expression>;
```

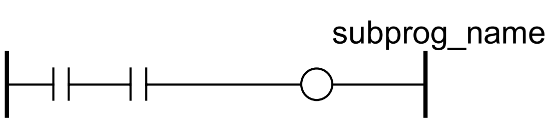
IL: the value of the current result (IL register)
at the end of the sequence is stored in the return parameter:

```
LD     10
ADD    20   (* return parameter value = 30 *)
```

FBD: set the return parameter using its name:

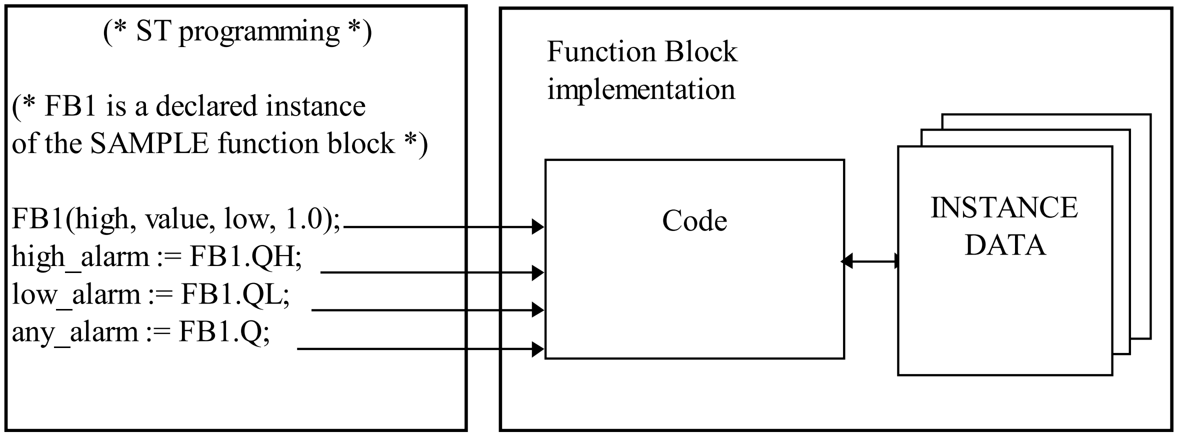


LD: use a coil symbol with the name of the return parameter:



E.1.5 Function blocks

Function blocks can use the languages: LD, FBD, ST or IL. Function blocks are instantiated. It means local variables of a function block are copied for each instance. When calling a block in a program, you actually call the instance of the block: the same code is called but the data used are the one which have been allocated for the instance. Values of the variables of the instance are stored from one cycle to the other.



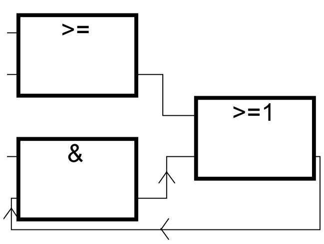
Warnings:
- A function block written with one of the IEC languages can not call other function blocks: the instantiation mechanism only manages the local variables of the block itself. Here is the list of standard function blocks that you cannot use inside an IEC function block:
SR, RS, R_Trig, F_Trig, SEMA, CTU, CTD, CTUD, TON, TOF, TP, CMP, StackInt, AVERAGE, HYSTER, LIM_ALRM, INTEGRAL, DERIVATE, BLINK, SIG_GEN

- For the same reason, you can not use Positive or Negative contact or coils, or Set and Reset coils.

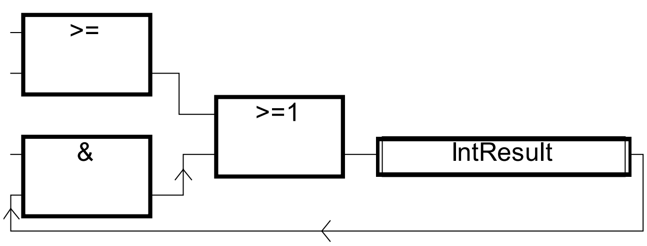
- TSTART and TSTOP functions to start and stop timers cannot be used in a function block for 3.0x targets. It works since the 3.20 target.

- When you need loop in your function block, you must use local variable before doing the loop. See the example below:

This will not work:



This is OK:



E.1.6 Description language

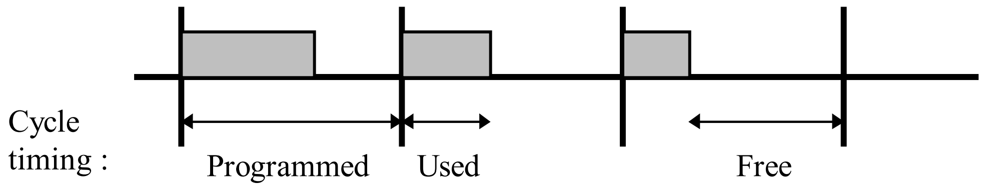
A program can be described with any of the following graphic or literal languages:

Sequential Function Chart (SFC) for high level operations
Flow Chart (FC) for high level operations
Function Block Diagram (FBD) for cyclic complex operations
Ladder Diagram (LD) for boolean operations only
Structured Text (ST) for any cyclic operations
Instruction List (IL) for low level operations

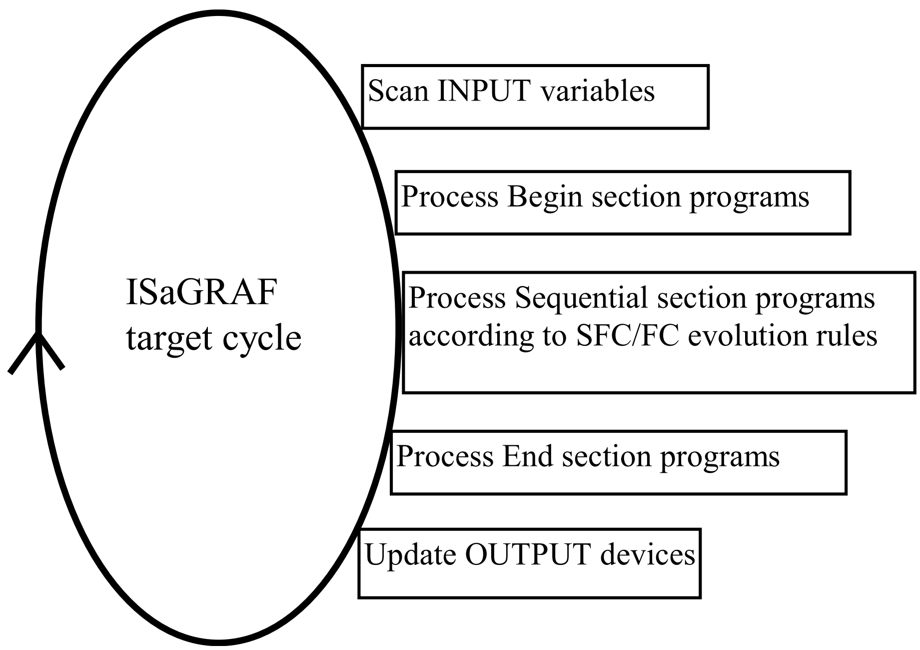
The same program cannot mix several languages. The language used to describe a program is chosen when the program is created, and cannot be changed later on. The exception is that it is possible to combine FBD and LD in a single program.

E.1.7 Execution rules

ISaGRAF is a **synchronous** system. All the operations are triggered by a clock. The basic duration of the clock is called the cycle timing:



Basic operations processed during a target cycle are:



This system makes it possible to:

- guarantee that an input variable keeps the same value within a cycle,
- guarantee that an output device is not updated more than once in a cycle,
- work safely on the same global variable from different programs,
- estimate and control the response time of the complete application.

E.2 Common objects

These are main features and common **objects** of the ISaGRAF programming database. Such objects can be used in any program written with any of the **SFC**, **FC**, **FBD**, **LD**, **ST** or **IL** languages.

E.2.1 Basic types

Any constant, expression or variable used in a program (written in any language) must be characterised by a type. Type coherence must be followed in graphic operations and literal statements. These are the available basic types for programming objects:

BOOLEAN:	logic (true or false) value
ANALOG:	integer or real (floating) continuous value
TIMER:	time value
MESSAGE:	character string

Note: Timers contain values less than one day and cannot be used to store dates.

E.2.2 Constant expressions

Constant expressions are relative to one type. The same notation cannot be used to represent constant expressions of different types.

E.2.2.1 Boolean constant expressions

There are only two boolean constant expressions:

TRUE	is equivalent to the integer value 1
FALSE	is equivalent to the integer value 0

"True" and "False" keywords are case insensitive.

E.2.2.2 Integer analog constant expressions

Integer constant expressions represent signed long integer (32 bit) values: from **-2147483647** to **+2147483647**. Integer analog constants may be expressed with one of the following **bases**. Integer constants must begin with a **prefix** that identifies the bases used:

Base	Prefix	Example
DECIMAL	**(none)**	**-908**
HEXADECIMAL	**"16#"**	**16#1A2B3C4D**
OCTAL	**"8#"**	**8#1756402**
BINARY	**"2#"**	**2#1101_0001_0101_1101**

The underscore character ('_') may be used to separate groups of digits. It has no particular significance, and is used to increase constant expression readability.

E.2.2.3 Real analog constant expressions

Real analog constant expressions can be written with either **decimal** or **scientific** representation. The **decimal point** ('**.**') separates the integer and decimal parts. The decimal point must be used to differentiate a real constant expression from an integer one. The scientific representation uses the '**E**' or '**F**' letter to separate the **mantissa** part and the **exponent**. Exponent part of a real scientific expression must be a signed integer value from **-37** to **+37**. Below are examples of real analog constant expressions:

3.14159	**-1.0E+12**
+1.0	**1.0F-15**
-789.56	**+1.0E-37**

The expression "**123**" does not represent a real constant expression. Its correct real representation is "**123.0**".

E.2.2.4 Timer constant expressions

Timer constant expressions represent time values from **0 second** to **23h59m59s999ms**. The lowest allowed unit is a millisecond. Standard time units used in constant expressions are:

Hour	The "**h**" letter must follow the number of hours
Minute	The "**m**" letter must follow the number of minutes
Second	The "**s**" letter must follow the number of seconds
Millisecond	The "**ms**" letters must follow the number of milliseconds

The time constant expression must begin with "**T#**" or "**TIME#**" prefix. Prefixes and unit letters are case insensitive. Some units may not appear. These are examples of timer constant expressions:

T#1H450MS	1 hour, 450 milliseconds
time#1H3M	1 hour, 3 minutes

The expression "0" does not represent a time value, but an analog constant.

E.2.2.5 Message string constant expressions

String or message constant expressions represent character strings. Characters must be preceded by a quote and followed by an apostrophe. For example:

'THIS IS A MESSAGE'

Warning: The apostrophe '**'**' character cannot be used within a string constant expression. A string constant expression must be expressed on one line of the program source code. Its length cannot exceed 255 characters, including spaces.

Empty string constant expression is represented by two apostrophes, with no space or tab character between them:

''   (* this is an empty string *)

The special character dollar ('**$**'), followed by other special characters, can be used in a string constant expression to represent a non-printable character:

Sequence	Meaning	ASCII (hexa)	Example
$$	'$' character	16#24	'I paid $$5 for this'
$'	apostrophe	16#27	'Enter $'Y$' for YES'
$L	line feed	16#0a	'next $L line'
$R	carriage return	16#0d	' llo $R He'
$N	new line	16#0d0a	'This is a line$N'
$P	new page	16#0c	'lastline $P first line'
$T	tabulation	16#09	'name$Tsize$Tdate'
$hh (*)	any character	16#hh	'ABCD = $41$42$43$44'

(*) "**hh**" is the hexadecimal value of the ASCII code for the expressed character.

E.2.3 Variables

Variables can be **LOCAL** to one program, or **GLOBAL**. Local variables can be used by one program only. Global variables can be used in any program of the project. Variable names must conform to the following rules:

name cannot exceed **16** characters
first character must be a **letter**
following characters can be **letters**, **digits** or the underscore character

E.2.3.1 Reserved keywords

A list of the reserved keywords is shown below. Such identifiers cannot be used to name a program, a variable or a "C" function or function block:

A   ANA, ABS, ACOS, ADD, ANA, AND, AND_MASK, ANDN, ARRAY, ASIN, AT, ATAN,
B   BCD_TO_BOOL, BCD_TO_INT, BCD_TO_REAL, BCD_TO_STRING, BCD_TO_TIME, BOO, BOOL, BOOL_TO_BCD, BOOL_TO_INT, BOOL_TO_REAL, BOOL_TO_STRING, BOOL_TO_TIME, BY, BYTE,

C CAL, CALC, CALCN, CALN, CALNC, CASE, CONCAT, CONSTANT, COS,
D DATE, DATE_AND_TIME, DELETE, DINT, DIV, DO, DT, DWORD,
E ELSE, ELSIF, EN, END_CASE, END_FOR, END_FUNCTION, END_IF, END_PROGRAM, END_REPEAT, END_RESSOURCE, END_STRUCT, END_TYPE, END_VAR, END_WHILE, ENO, EQ, EXIT, EXP, EXPT,
F FALSE, FEDGE, FIND, FOR, FUNCTION,
G GE, GFREEZE, GKILL, GRST, GSTART, GSTATUS, GT,
I IF, INSERT, INT, INT_TO_BCD, INT_TO_BOOL, INT_TO_REAL, INT_TO_STRING, INT_TO_TIME,
J JMP, JMPC, JMPCN, JMPN, JMPNC,
L LD, LDN, LE, LEFT, LEN, LIMIT, LINT, LN, LOG, LREAL, LT, LWORD,
M MAX, MID, MIN, MOD, MOVE, MSG, MUL, MUX,
N NE, NOT,
O OF, ON, OPERATE, OR, OR_MASK, ORN,
P PROGRAM
R R, REDGE, READ_ONLY, READ_WRITE, REAL, REAL_TO_BCD, REAL_TO_BOOL, REAL_TO_INT, REAL_TO_STRING, REAL_TO_TIME, REDGE, REPEAT, REPLACE, RESSOURCE, RET, RETAIN, RETC, RETCN, RETN, RETNC, RETURN, RIGHT, ROL, ROR,
S S, SEL, SHL, SHR, SIN, SINT, SQRT, ST, STN, STRING, STRING_TO_BCD, STRING_TO_BOOL, STRING_TO_INT, STRING_TO_REAL, STRING_TO_TIME, STRUCT, SUB, SYS_ERR_READ, SYS_ERR_TEST, SYS_INITALL, SYS_INITANA, SYS_INITBOO, SYS_INITTMR, SYS_RESTALL, SYS_RESTANA, SYS_RESTBOO, SYS_RESTTMR, SYS_SAVALL, SYS_SAVANA, SYS_SAVBOO, SYS_SAVTMR, SYS_TALLOWED, SYS_TCURRENT, SYS_TMAXIMUM, SYS_TOVERFLOW, SYS_TRESET, SYS_TWRITE, SYSTEM,
T TAN, TASK, THEN, TIME, TIME_OF_DAY, TIME_TO_BCD, TIME_TO_BOOL, TIME_TO_INT, TIME_TO_REAL, TIME_TO_STRING, TMR, TO, TOD, TRUE, TSTART, TSTOP, TYPE,
U UDINT, UINT, ULINT, UNTIL, USINT,
V VAR, VAR_ACCESS, VAR_EXTERNAL, VAR_GLOBAL, VAR_IN_OUT, VAR_INPUT, ,VAR_OUTPUT,
W WHILE, WITH, WORD,
X XOR, XOR_MASK, XORN

All keywords beginning with an underscore ('_') character are internal keywords and must not be used in textual instructions.

E.2.3.2 Directly represented variables

ISaGRAF enables the use of **directly represented variables** in the source of the programs to represent a free channel. Free channels are the ones which are not linked to a declared I/O variable. The identifier of a directly represented variable always begins with "**%**" character.

Below are the naming conventions of a directly represented variable for a channel of a single board. "***s***" is the slot number of the board. "***c***" is the number of the channel.

%IX*s.c*	free channel of a boolean input board
%ID*s.c*	free channel of an integer input board
%IS*s.c*	free channel of a message input board
%QX*s.c*	free channel of a boolean output board
%QD*s.c*	free channel of an integer output board

`%QS`*`s.c`* free channel of a message output board

Below are the naming conventions of a directly represented variable for a channel of a complex equipment. "***s***" is the slot number of the equipment. "***b***" is the index of the single board within the complex equipment. "***c***" is the number of the channel.

`%IX`*`s.b.c`*	free channel of a boolean input board
`%ID`*`s.b.c`*	free channel of an integer input board
`%IS`*`s.b.c`*	free channel of a message input board
`%QX`*`s.b.c`*	free channel of a boolean output board
`%QD`*`s.b.c`*	free channel of an integer output board
`%QS`*`s.b.c`*	free channel of a message output board

Below are examples:

`%QX1.6`	6th channel of the board #1 (boolean output)
`%ID2.1.7`	7th channel of the board #1 in the equipment #2 (integer input)

A directly represented variable cannot have the "**real**" data type.

E.2.3.3 Boolean variables

Boolean means **logic**. Such variables can take one of the boolean values: **TRUE** or **FALSE**. Boolean variables are typically used in boolean expressions. Boolean variables can have one of the following **attributes**:

Internal: memory variable updated by the program
Constant: read-only memory variable with an initial value
Input: variable connected to an input device (refreshed by the system)
Output: variable connected to an output device

Warning: When declaring a boolean variable, strings can be defined to replace 'true' and 'false' values during debug. Those strings cannot be used in the programs unless entered as **'defined words'** for the language.

E.2.3.4 Analog variables

Analog means **continuous**. Such variables have signed integer or real (floating) values. Available formats for an analog variable are:

Integer 32 bit signed integer: from **-2147483647** to **+2147483647**
Real standard IEEE 32 bit floating value (single precision)
1 sign bit + 23 mantissa bits + 8 exponent bits

REAL analog exponent value cannot be less than **-37** or greater than **+37**. Analog variables can have one of the following **attributes**:

Internal memory variable updated by the program

Constant: read-only memory variable with an initial value
Input variable connected to an input device (refreshed by the system)
Output variable connected to an output device

Note: When a real variable is connected to an I/O device, the corresponding I/O driver operates the equivalent integer value.

Warning: Integer and real analog variables or constant expressions cannot be mixed in the same analog expression.

E.2.3.5 Timer variables

Timer means **clock** or **counter**. Such variables have time values and are typically used in time expressions. A timer value cannot exceed **23h59m59s999ms** and cannot be negative. Timer variables are stored in 32 bit words. The internal representation is a positive number of milliseconds.
Timer variables can have one of the following **attributes**:

Internal memory variable managed by the program, refreshed by ISaGRAF system
Constant: read-only memory variable with an initial value

Warning: Timer variables cannot have the INPUT or OUTPUT attributes.

Timer variables can be automatically refreshed by the ISaGRAF system. When a timer is **active**, its value is automatically increased according to the target system real time clock. The following statements of the **ST** language can be used to control a timer:

TSTART starts automatic refresh of a timer
TSTOP stops automatic refresh of a timer

E.2.3.6 Message string variables

Message or string variables contain character strings. The length of the string can change during process operations. The length of a message variable cannot exceed the capacity (maximum length) specified when the variable is declared. Message capacity is limited to 255 characters. Message variables can have one of the following **attributes**:

Internal memory variable updated by the program
Constant: read-only memory variable with an initial value
Input variable connected to an input device (refreshed by the system)
Output variable connected to an output device

String variables can contain any character of the standard ASCII table (ASCII code from **0** to **255**). The null character can exist in a character string. Some "C" functions of the standard ISaGRAF library will not correctly operate messages which contain null (**0**) characters.

E.2.4 Comments

Comments may be freely inserted in literal languages such as **ST** and **IL**. A comment must begin with the special characters "**(***" and terminate with the characters "***)**". Comments can be inserted anywhere in a **ST** program, and can be written on more than one line.

These are examples of comments:

counter := ivalue;   **(* assigns the main counter *)**
(* this is a comment expressed
on two lines *)
c := counter **(* you can put comments anywhere *)** + base_value + 1;

Interleave comments cannot be used. This means that the "**(***" characters cannot be used within a comment.

Warning: The IL language only accepts comments as the last component of an instruction line.

E.2.5 Defined words

The ISaGRAF system allows the re-definition of constant expressions, true and false boolean expressions, keywords or complex **ST** expressions. To achieve this, an **identifier** name has to be given to the corresponding expression. For example:

YES is **TRUE**
PI is **3.14159**
OK is **(auto_mode AND NOT (alarm))**

When such equivalence is defined, its **identifier** can be used anywhere in an **ST** program to replace the attached expression. This is an example of **ST** programming using defines:

```
If OK Then
  angle := PI / 2.0;
  isdone := YES;
End_if;
```

Defined words can be **LOCAL** to one program, **GLOBAL**, or **COMMON**.
Local defined words can be used by only one program.
Global defined words can be used in any program of the project.
Common defined words can be used in any program of any project.
Note that common defined can be stored separately with the Archive manager.

Warning: When the same identifier is defined twice with different **ST** equivalencies, the last defined expression is used. For example:

Define: **OPEN** is **FALSE**
OPEN is **TRUE**

means: **OPEN** is **TRUE**

Naming defined words must conform to following rules:
- name cannot exceed **16** characters
- first character must be a **letter**
- following characters can be **letters**, **digits** or underscore ('_') character

Warning: A defined word can not use a defined word in its definition, for example, you can not have:

PI is **3.14159**
~~**PI2** is **PI*2**~~

write the complete equivalence using constants or variables and operations:

PI2 is **6.28318**

E.3 SFC language

Sequential Function Chart (SFC) is a **graphic** language used to describe **sequential operations**. The process is represented as a set of well-defined **steps**, linked by **transitions**. A **boolean condition** is attached to each transition. **Actions** within the steps are detailed by using other languages (**ST**, **IL**, **LD** and **FDB**).

E.3.1 SFC chart main format

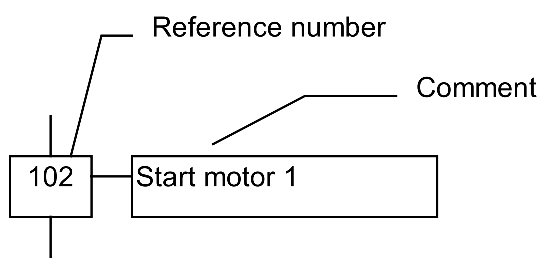
An SFC program is a graphic set of **steps** and **transitions**, linked together by **oriented links**. Multiple connection links are used to represent divergences and convergences. Some parts of the complete program may be separated and represented in the main chart by a single symbol, called **macro steps**. The basic **graphic rules** of the SFC are:
- A step cannot be followed by another step
A transition cannot be followed by another transition

E.3.2 SFC basic components

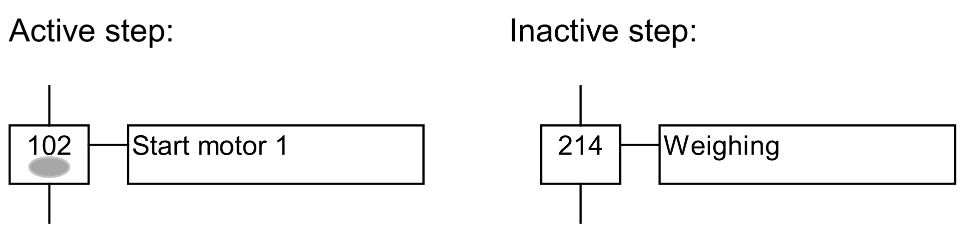
The basic components (graphic symbols) of the SFC language are: steps and initial steps, transitions, oriented links, and jumps to a step.

E.3.2.1 Steps and initial steps

A step is represented by a single **square**. Each step is **referenced** by a number, written in the step square symbol. A main description of the step is written in a rectangle linked to the step symbol. This description is a **free comment** (not part of the programming language). The above information is called the **Level 1** of the step:

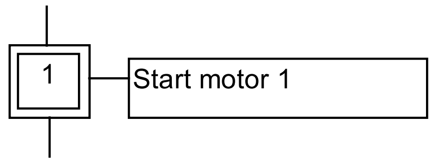


At run time, a **token** indicates that the step is **active**:



The **initial situation** of an SFC program is expressed with **initial steps**. An initial step has a **double-bordered** graphic symbol. A token is automatically placed in each initial step when the program is started.

Initial step:

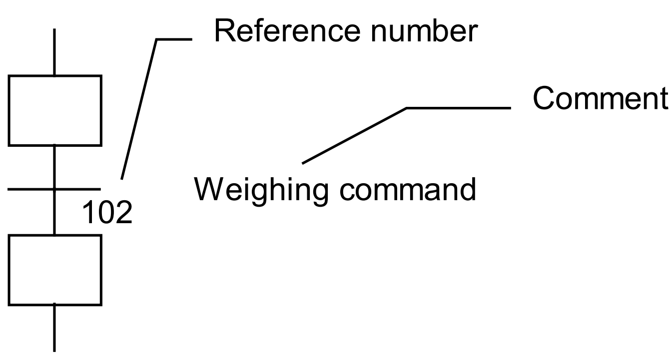


An SFC program must contain **at least one** initial step.

These are the attributes of a step. Such fields may be used in any of the other languages:
GSnnn.x..........activity of the step (boolean value)
GSnnn.t...........activation duration of the step (time value)
(where **nnn** is the reference number of the step)

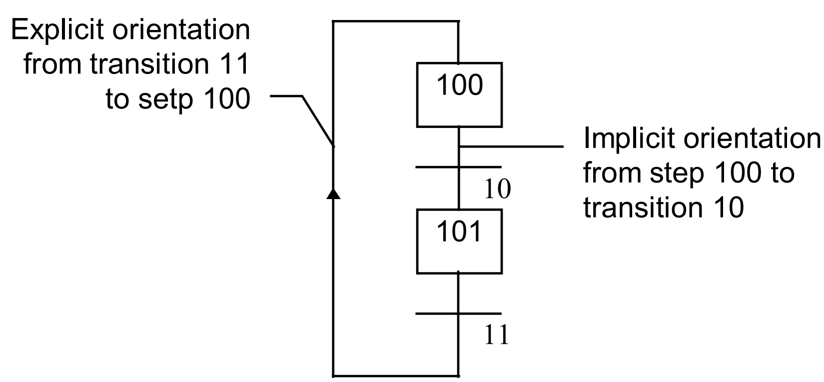
E.3.2.2 Transitions

Transitions are represented by a small horizontal bar that crosses the connection link. Each transition is **referenced** by a number, written next to the transition symbol. A main description of the transition is written on the right side of the transition symbol. This description is a **free comment** (not part of the programming language). The above information is called the **Level 1** of the transition:



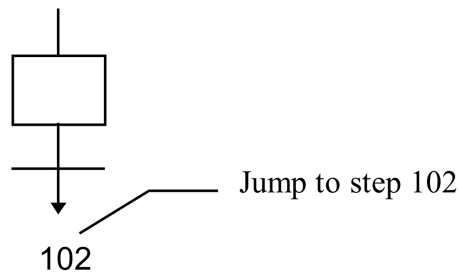
E.3.2.3 Oriented links

Single lines are used to link steps and transitions. These are oriented links. When the orientation is not explicitly given, the link is oriented from the top to the bottom.

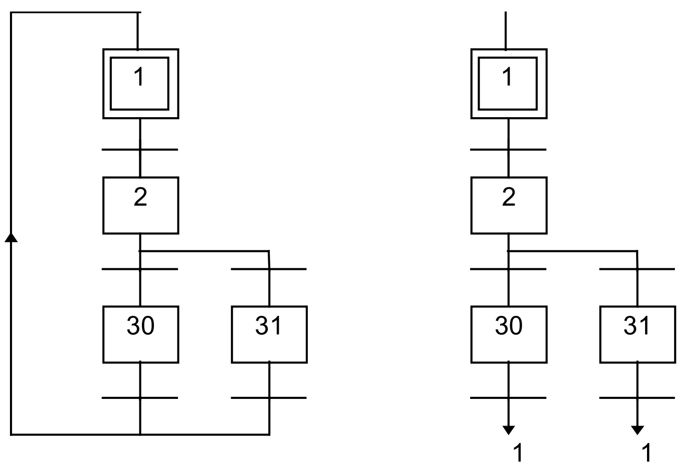


E.3.2.4 Jump to a step

Jump symbols may be used to indicate a connection link from a transition to a step, without having to draw the connection line. The jump symbol must be referenced with the number of the destination step:



A jump symbol cannot be used to represent a link from a step to a transition. Example of jumps - the following charts are equivalent:

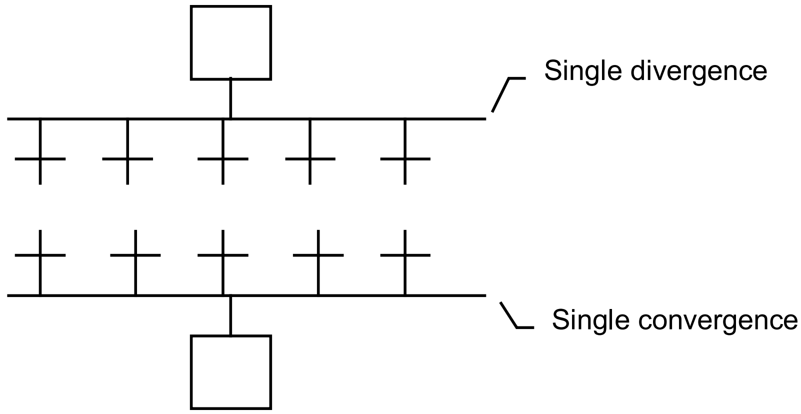


E.3.3 Divergences and convergences

Divergences are **multiple connection links** from one SFC symbol (step or transition) to many other SFC symbols. Convergences are multiple connection links from more than one SFC symbols to one other symbol. Divergences and convergences can be single or double.

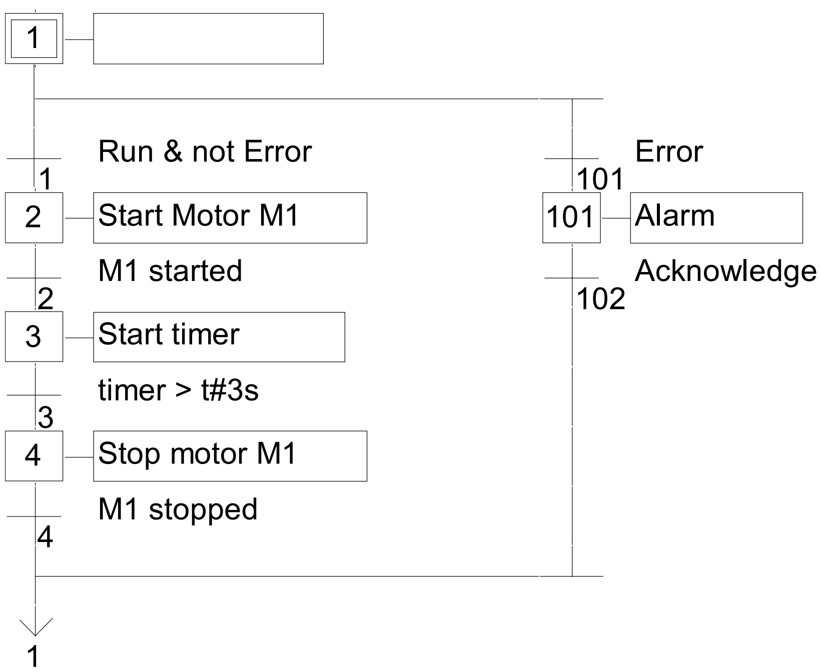
E.3.3.1 single divergences

A single divergence is a multiple link from one step to many transitions. It allows the active token to pass into one of a number of branches. A single convergence is a multiple link from many transitions to the same step. A single convergence is generally used to group the SFC branches which were started on a single divergence. Single divergences and convergences are represented by single horizontal lines.



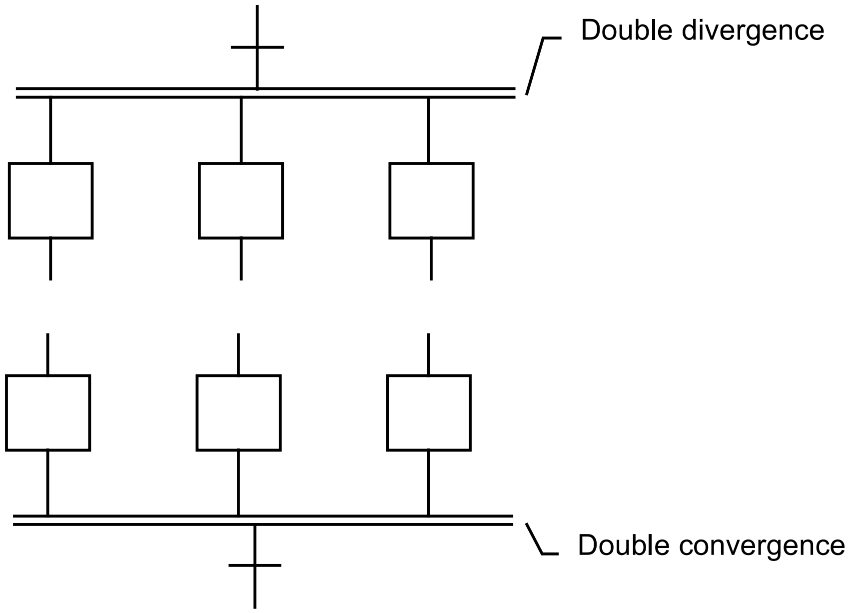
Warning: The conditions attached to the different transitions at the beginning of a single divergence are **not implicitly exclusive**. The exclusivity has to be explicitly detailed in the conditions of the transitions to ensure that only one token progresses in one branch of the divergence at run time. Below is an example of single divergence and convergence:

(* SFC program with single divergence and convergence *)



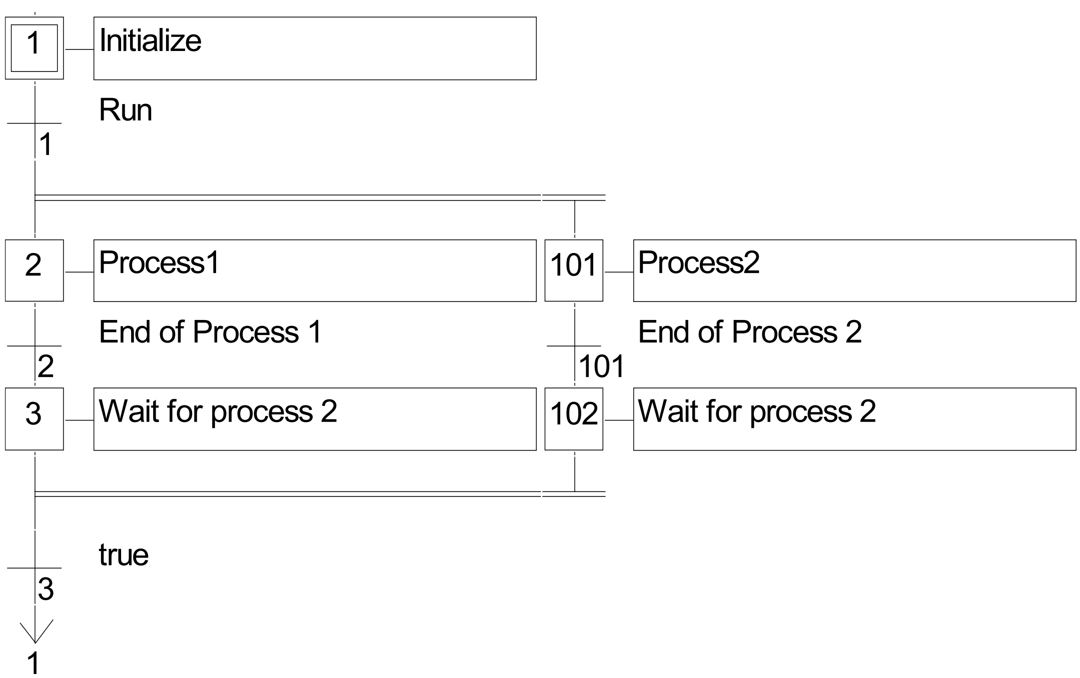
E.3.3.2 Double divergences

A double divergence is a multiple link from one transition to many steps. It corresponds to parallel operations of the process. A double convergence is a multiple link from many steps to the same transition. A double convergence is generally used to group the SFC branches started on a double divergence. Double divergences and convergences are represented by double horizontal lines.



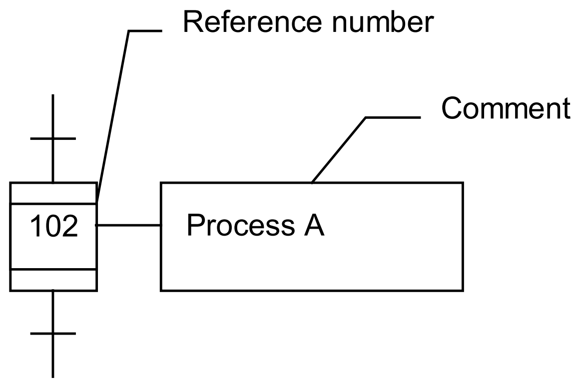
Example of double divergence and convergence:

(* SFC program with double divergence and convergence *)



E.3.4 Macro steps

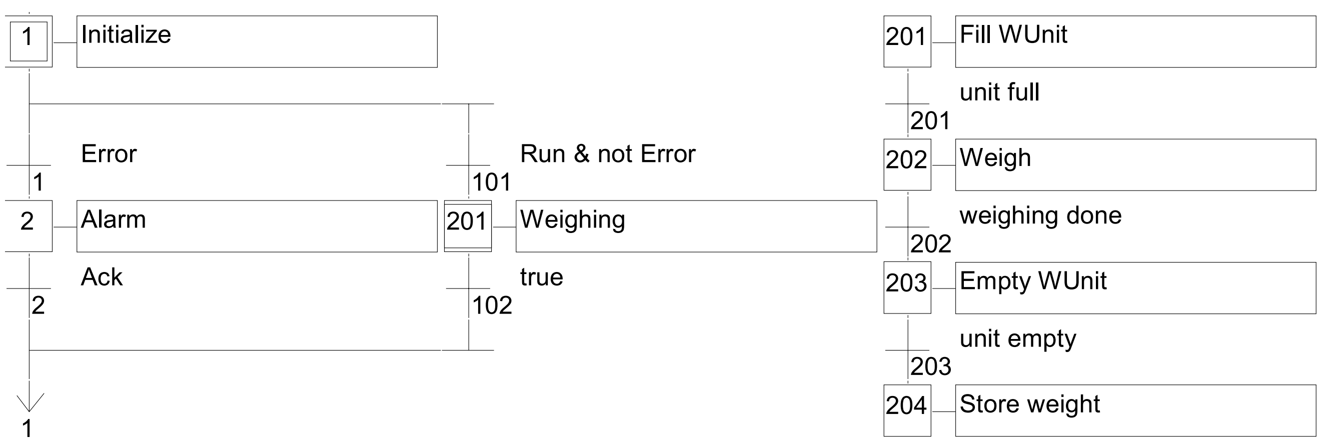
A macro step is a unique representation of a unique group of steps and transitions. The body of the macro step is described separately, elsewhere in the same SFC program. It appears as a single symbol in the main SFC chart. This is the symbol used for a macro step:



The reference number written in the macro step symbol is the reference number of the first step in the body of the macro step. The macro step body must begin with a **beginning step** and terminate with an **ending step**. The chart must be self-contained. A beginning step has no upper link (no backward transition). An ending step has no lower link (no forward transition). A macro step symbol may be put in the body of another macro step.

Warning: Because macro step is a **unique** set of steps and transitions, the same macro step cannot be used more than once in an SFC program.

Example of macro step:
(* SFC program with macro step *)
(* Main chart *)(* Body of the macro step *)



E.3.5 Actions within the steps

The **level 2** of an SFC step is the detailed description of the **actions** executed **during the step activity**. This description is made by using **SFC literal features**, and other languages such as Structured Text (**ST**). The basic types of actions are:
- Boolean actions
- Pulse actions programmed in ST
- Non-stored actions programmed in ST
- SFC actions

Several actions (with same or different types) can be described in the same step. The special features that enable the use of any of the other languages are:
- Calling sub-programs
- Instruction List (IL) language convention

E.3.5.1 Boolean actions

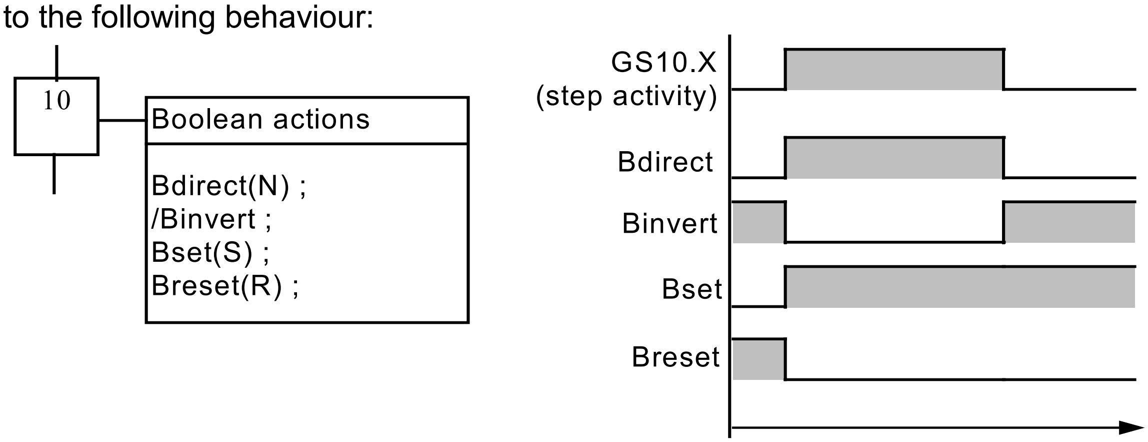
Boolean actions assign a boolean variable with the activity of the step. The boolean variable can be an output or an internal. It is assigned each time the step activity starts or stops. This is the syntax of the basic boolean actions:

<boolean_variable> (N) ;	assigns the step activity signal to the variable
<boolean_variable> ;	same effect (N attribute is optional)
/ <boolean_variable> ;	assigns the negation of the step activity signal to the variable

Other features are available to set or reset a boolean variable, when the step becomes active. This is the syntax of set and reset boolean actions:

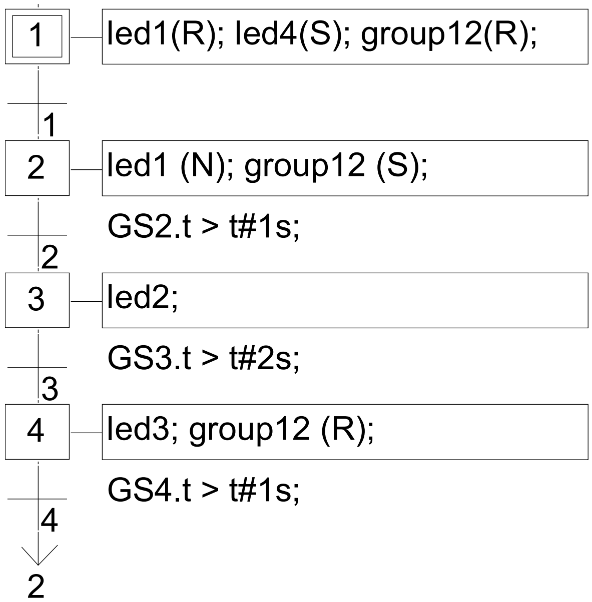
<boolean_variable> (S) ;	sets the variable to TRUE when the step activity signal becomes TRUE
<boolean_variable> (R) ;	resets the variable to FALSE when the step activity signal becomes TRUE

The boolean variable must be an OUTPUT or an INTERNAL. The following SFC programming leads to the following behaviour:



Example of boolean actions:

(* SFC program using BOOLEAN actions *)

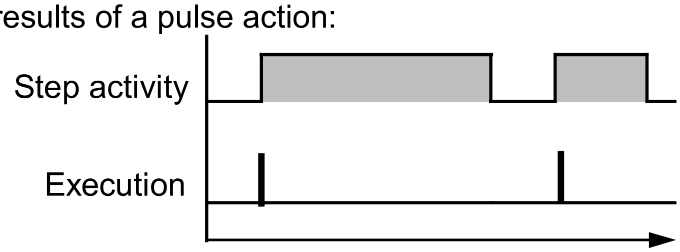


E.3.5.2 Pulse actions

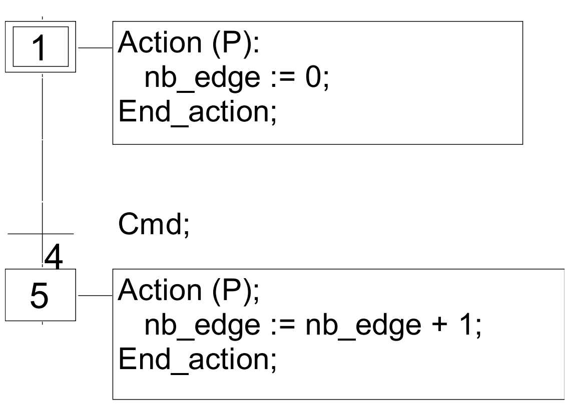
A pulse action is a list of ST or IL instructions, which are executed only **once** at the **activation** of the step. Instructions are written according to the following SFC syntax:

```
ACTION (P) :
  (* ST statements *)
END_ACTION ;
```

The following shows the results of a pulse action:



Example of pulse action:

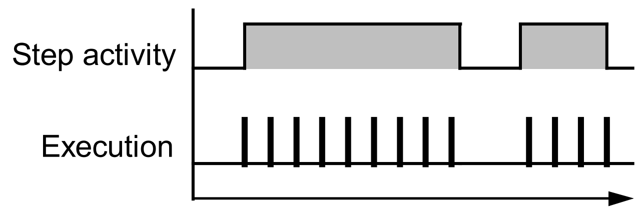


E.3.5.3 Non-stored actions

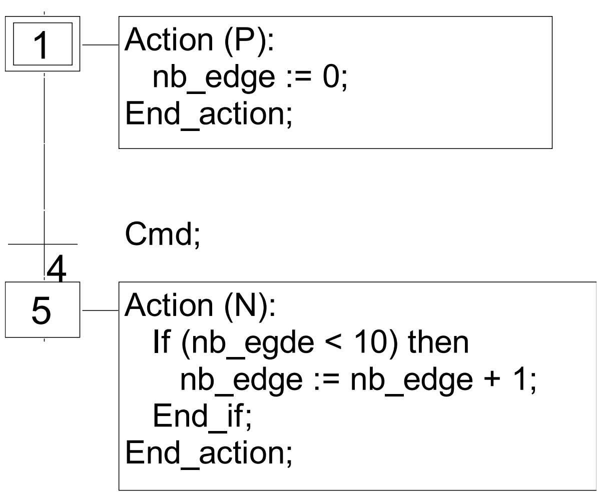
A non-stored (normal) action is a list of ST or IL instructions which are executed **at each cycle** during the whole **active** period of the step. Instructions are written according to the following SFC syntax:

```
ACTION (N) :
  (* ST statements *)
END_ACTION ;
```

The following is the results of a non-stored action:



Example of non-stored action:



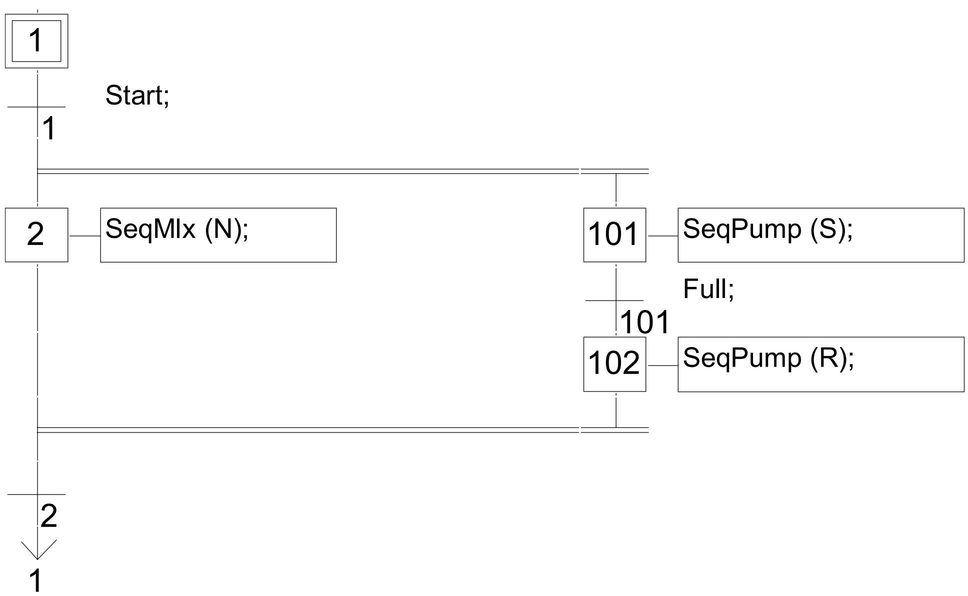
E.3.5.4 SFC actions

An SFC action is a child SFC sequence, started or killed according to the change of the step activity signal. An SFC action can have the **N** (Non stored), **S** (Set), or **R** (Reset) qualifier. This is the syntax of the basic SFC actions:

<child_prog> (N); starts the child sequence when the step becomes active, and kills the child sequence when the step becomes inactive
<child_prog> ; same effect (N attribute is optional)
<child_prog> (S); starts the child sequence when the step becomes active. Nothing is done when the step becomes inactive
<child_prog> (R); kills the child sequence when the step becomes active. Nothing is done when the step becomes inactive

The SFC sequence specified as an action must be a **child SFC program** of the program currently being edited. Note that using the **S** (Set) or **R** (Reset) qualifiers for an SFC action has exactly the same effect as the **GSTART** and **GKILL** statements, programmed in an **ST** pulse action.
Below is an example of an SFC action. The main SFC program is named **Father**. It has two SFC children, called **SeqMlx** and **SeqPump**. The SFC programming of the father SFC program is:

(* SFC program using SFC actions *)



E.3.5.5 Calling function and function blocks from an action

Sub-programs, functions or function blocks (written in ST, IL, LD or FBD language) or "C" functions and "C" function blocks, can be directly called from an SFC action block, based on the following syntax:

For sub-programs, functions and "C" functions:

```
ACTION (P) :
  result := sub_program ( ) ;
END_ACTION;
```

or

```
ACTION (N) :
  result := sub_program ( ) ;
END_ACTION;
```

For function blocks in "C" or in ST, IL, LD, FBD:

```
ACTION (P) :
  Fbinst(in1, in2);
  result1 := Fbinst.out1;
```

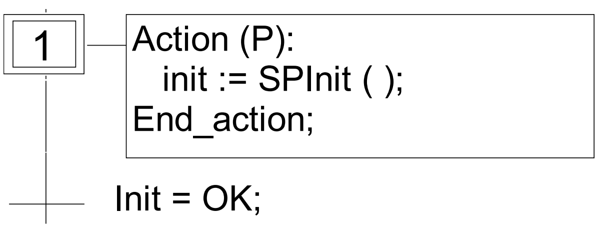
```
  result2 := Fbinst.out2;
END_ACTION;
```

or

```
ACTION (N) :
  Fbinst(in1, in2);
  result1 := Fbinst.out1;
  result2 := Fbinst.out2;
END_ACTION;
```

Detailed syntax can be found in the ST language section.
Example of a sub-program call in action blocks:

(* SFC program with a sub-program call in an action block *)



E.3.5.6 IL convention

Instruction List (IL) programming may be directly entered in an SFC action block, based on the following syntax:

```
ACTION (P) :     (* or N *)
#info=IL
    <instruction>
    <instruction>
    ....
#endinfo
END_ACTION;
```

The special "#info=IL" and "#endinfo" keywords must be entered exactly this way, and **are case sensitive**. Space or tab characters cannot be inserted into, after or before the keywords.
Below is an example of an IL program in an action block:

(* SFC program with an IL sequence in an action block *)

1

```
Action (P):
#info=IL
  LD   False
  ST   Led1
  ST   Led2
#endinfo
End_action;
```

E.3.6 Conditions attached to transitions

At each transition, a **boolean expression** is attached that conditions the clearing of the transition. The condition is usually expressed with ST language or using the LD language (Quick LD editor). This is the **Level 2** of the transition. Other structures may, however, be used:

- ST language convention
- LD language convention
- IL language convention
- Calling function from a transition

Warning: When no expression is attached to the transition, the default condition is **TRUE**.

E.3.6.1 ST convention

The **Structured Text** (ST) language can be used to describe the **condition** attached to a transition. The complete expression must have **boolean** type and must be terminated by a **semicolon**, according to the following syntax:

< boolean_expression > ;

The expression may be a TRUE or FALSE constant expression, a single input or an internal boolean variable, or a combination of variables that leads to a boolean value. Below is an example of ST programming for transitions:

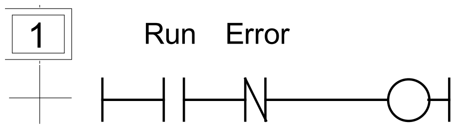
(* SFC program with ST programming for transitions *)

1

Run & not Error;

E.3.6.2 LD convention

The **Ladder Diagram** (LD) language can be used to describe the **condition** attached to a transition. The diagram is composed of only one rung with one coil. The coil value represents the transition value. Below is an example of LD programming for transitions:



E.3.6.3 IL convention

Instruction List (IL) programming may be directly used to describe an SFC transition, according to the following syntax:

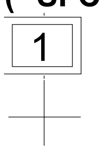
#info=IL
<instruction>
<instruction>
....
#endinfo

The value contained by the **current result** (IL register) at the end of the IL sequence causes the resulting of the condition to be attached to the transition:

current result = 0 ➔ condition is **FALSE**
current result <> 0 ➔ condition is **TRUE**

The special "#info=IL" and "#endinfo" keywords must be entered exactly this way, and **are case sensitive**. Space or tab characters cannot be inserted into, after or before the keywords. Below is an example of IL programming for transitions:

(* SFC program with an IL program for transitions *)




```
#info=IL
  LD   Run
  &N  Error
#endinfo
```

E.3.6.4 Calling functions from a transition

Any sub-program or a function (written in FBD, LD, ST or IL language), or a "C" function can be called to evaluate the condition attached to a transition, according to the following syntax:

< sub_program > ( ) ;

The value returned by the sub-program or the function must be boolean and yields the resulting condition:

return value = FALSE ➔ condition is **FALSE**
return value = TRUE ➔ condition is **TRUE**

Example of a sub-program called in a transition:

(* SFC program with sub-program call for transitions *)

E.3.7 SFC dynamic rules

The **five** dynamic rules of the SFC language are:

Initial situation

The initial situation is characterised by the **initial steps** which are, by definition, in the active state at the beginning of the operation. **At least one** initial step must be present in each SFC program.

Clearing of a transition

A transition is either **enabled** or **disabled**. It is said to be enabled when all immediately preceding steps linked to its corresponding transition symbol are **active**, otherwise it is disabled. A transition cannot be **cleared** unless:
- it is enabled, and
- the associated transition condition is true.

Changing of state of active steps

The clearing of a transition simultaneously leads to the active state of the immediately following steps and to the inactive state of the immediately preceding steps.

Simultaneous clearing of transitions

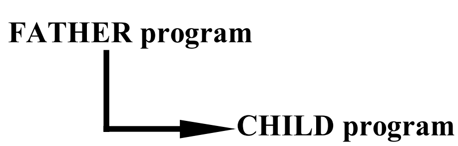
Double lines may be used to indicate transitions which have to be cleared simultaneously. If such transitions are shown separately, the activity state of preceding steps (GSnnn.x) can be used to express their conditions.

Simultaneous activation and deactivation of a step

If, during operation, a step is simultaneously activated and deactivated, priority is given to the activation.

E.3.8 SFC program hierarchy

The ISaGRAF system enables the description of the vertical structure of SFC programs. SFC programs are organised in a **hierarchy tree**. Each SFC program can control (start, kill...) other SFC programs. Such programs are called **children** of the SFC program which controls them. SFC programs are linked together into a main **hierarchy tree**, using a "**father - child**" relation:



The basic rules implied by the hierarchy structure are:

- SFC programs which have no father are called "**main**" SFC programs
- Main SFC programs are activated by the system when the application starts
- A program can have several child programs
- A child of a program cannot have more than one father
- A child program can only be controlled by its father
- A program cannot control the children of one of its own children

The basic actions that a father SFC program can take to control its child program are:

Start	(**GSTART**) Starts the child program: activates each of its initial steps. Children of this child program are not automatically started.
Kill	(**GKILL**) Kills the child program by deactivating each of its active steps. All the children of the child program are also killed.
Freeze	(**GFREEZE**) Suspends the execution of the program (deactivates actions of each of the active steps and suspend transition calculation), and memorises the status of the program steps so the program can be restarted. All the children of the child program are also frozen.
Restart	(**GRST**) Restarts a frozen SFC program by reactivating all the suspended steps. Children of the program are not automatically restarted.
Get status	(**GSTATUS**) Gets the current status (active, inactive or frozen) of a child program.

E.4 Flow Chart language

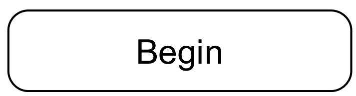
Flow Chart (**FC**) is a graphic language used to describe **sequential operations**. A Flow Chart diagram is composed of **Actions** and **Tests**. Between Actions and test are **oriented links** representing data flow. Multiple connection links are used to represents divergences and convergences. Actions and Tests can be described with ST, LD or IL languages. Functions and Function blocks of any language (except SFC) can be called from actions and tests. A Flow Chart program can call another Flow Chart program. The called FC program is a **sub-program** of the calling FC program.

E.4.1 FC components

Below are graphic components of the Flow Chart language:

- ***Beginning of FC chart***

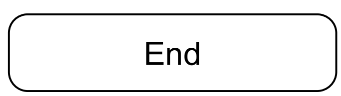
A "**begin**" symbol must appear at the beginning of a Flow Chart program. It is unique and cannot be omitted. It represents the initial state of the chart when it is activated. Below is the drawing of a "begin" symbol:



The "Begin" symbol always has a connection (on the bottom) to the other objects of the chart. A flow chart is not valid if no connection is drawn from "Begin" to another object.

- ***Ending of FC chart***

An "**end**" symbol must appear at the end of a Flow Chart program. It is unique and cannot be omitted. It is possible that no connection is drawn to the "End" symbol (always looping chart), but "End" symbol is still drawn anyway at the bottom of the chart. It represents the final state of the chart, when its execution has been completed. Below is the drawing of an "end" symbol:



The "End" symbol generally has a connection (on the top) to the other objects of the chart. A flow chart may have no connection to the "End" object (always looping chart). The "End" object is still visible at the bottom of the chart in this case.

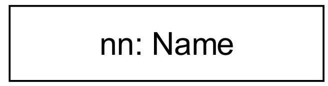
- ***FC flow links***

A flow **link** is a line that represents a flow between two points of the diagram. A link is always terminated by an arrow. Below is the drawing of a flow link:

Two links cannot start from the same source connection point.

- ***FC actions***

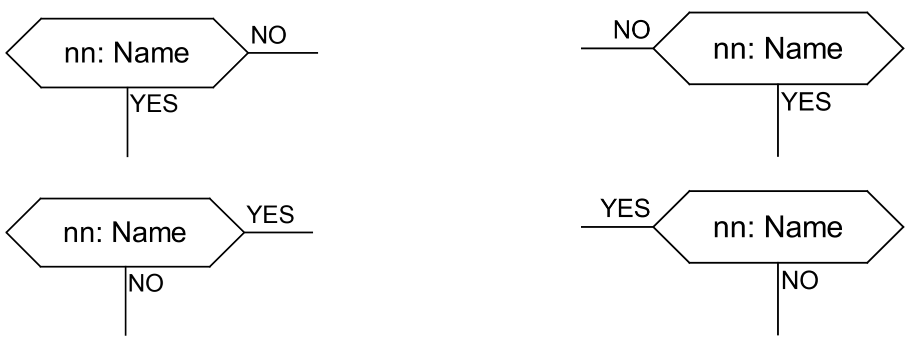
An **action** symbol represents actions to be performed. An action is identified by a number and a name. Below is the drawing of an "action" symbol:



Two different objects of the same chart cannot have the same name or logical number. Programming language for an action can be ST, LD or IL. An action is always connected with links, one arriving to it, one starting from it.

FC conditions

A **condition** represents a boolean **test**. A condition is identified by a number and a name. According to the evaluation of attached ST, LD or IL expression, the flow is directed to "YES" or "NO" path. Below are the possible drawings for a condition symbol:



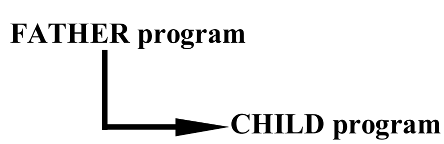
Two different objects of the same chart cannot have the same name or logical number. The programming of a test is either
- an expression in ST, or
- a single rung in LD, with no symbol attached to the unique coil, or
- several instructions in IL. The IL register (or current result) is used to evaluate the condition.

When programmed in ST text, the expression may optionally be followed by a semicolon. When programmed in LD, the unique coil represents the condition value. A condition equal to:
- 0 or FALSE directs the flow to NO
- 1 or TRUE directs the flow to YES

A test is always connected with an arriving link, and both forward connections must be defined.

FC sub-program

The system enables the description of the vertical structure of FC programs. FC programs are organised in a **hierarchy tree**. Each FC program can call other FC programs. Such a program is called a **child program** of the FC program which calls them. FC programs which call FC sub-programs are called **father program**. FC programs are linked together into a main hierarchy tree, using a "father - child" relation:



A **sub-program** symbol in a Flow Chart represents a call to a Flow Chart sub-program. Execution of the calling FC program is suspended till the sub-program execution is complete. A Flow Chart sub-program is identified by a number and a name, as other programs, functions or function blocks. Below is the drawing of a "sub-program call" symbol:

Two different objects of the same chart cannot have the same logical number. The basic rules implied by the FC hierarchy structure are:
- FC programs which have no father are called main FC programs.
- Main FC programs are activated by the system when the application starts
- A program can have several child programs
- A child of a program cannot have more than one father
- A child program can be called only by its father
- A program cannot call the children of one of its own children

The same sub-program may appear several times in the father chart. A Flow Chart sub-program call represents the complete execution of the sub chart. The father chart execution is suspended during the child chart is performed. The sub-program calling blocks must follow the same connection rules as the ones defined for action.

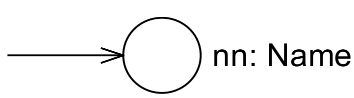
FC I/O specific action

An **I/O specific action** symbol represents actions to be performed. As other actions, an I/O specific action is identified by a number and a name. The same semantic is used on standard actions and I/O specific actions. The aim of I/O specific actions is only to make the chart more readable and to give focus on non-portable parts of the chart. Using I/O specific actions is an optional feature. Below is the drawing of an "I/O specific action" symbol:

I/O specific blocks have exactly the same behaviour as standard actions. This covers their properties, ST, LD or IL programming, and connection rules.

FC connectors

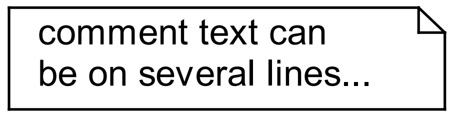
Connectors are used to represent a link between two points of the diagram without drawing it. A connector is represented as a circle and is connected to the source of the flow. The drawing of the connector is completed, on the appropriate side (depending on the direction of the data flow), by the identification of the target point (generally the name of the target symbol). Below is the standard drawing of a connector:



A connector always targets a defined Flow Chart symbol. The destination symbol is identified by its logical number.

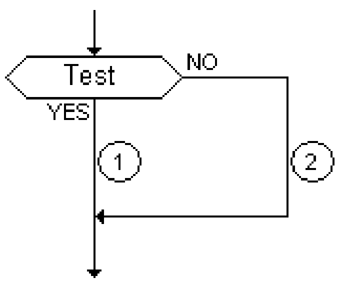
FC comments

A **comment** block contains text that has no sense for the semantic of the chart. It can be inserted anywhere on an unused space of the Flow Chart document window, and is used to document the program. Below is the drawing of a "comment" symbol:



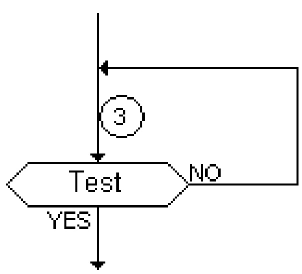
E.4.2 FC complex structures

This section shows **complex structure** examples that can be defined in a Flow Chart diagram. Such structures are combinations of basic objects linked together.



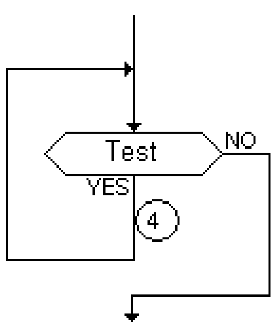
IF / THEN / ELSE

(1) place for "THEN" actions to be inserted
(2) place for "ELSE" actions to be inserted



REPEAT / UNTIL

(3) place for repeated actions to be inserted



WHILE / DO

(3) place for repeated actions to be inserted

E.4.3 FC dynamic behaviour

The **execution** of a Flow Chart diagram can be explained as follows:

- The Begin symbol takes one target cycle
- The End symbol takes one target cycle and ends the execution of the chart. After this symbol is reached, no more actions of the chart are executed.

- The flow is broken each time an item (action, decision) is encountered that has already been reached in the same cycle. In such a case the flow will continue on the next cycle.

Note: Contrary to SFC, an action is not a stable state. There is no repetition of instructions while the action symbol is highlighted.

E.4.4 FC checking

Apart of attached ST, LD or IL programming, some other **syntactic rules** apply to flow chart itself. Below is the list of main rules:
- All "connection" points of all symbols must be wired. (connection to "End" symbol may be omitted)
- All symbols must be linked together (no isolated part should appear)
- All connectors should have valid destination

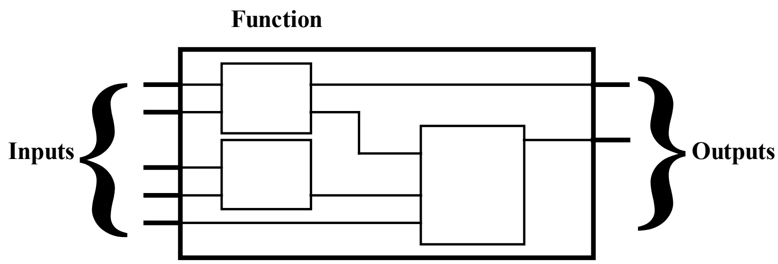
Other minor syntax errors can be reported:
- Empty actions (no programming) are considered as steps during run time scheduling
- Empty tests (no programming) are considered as "always true"

E.5 FBD language

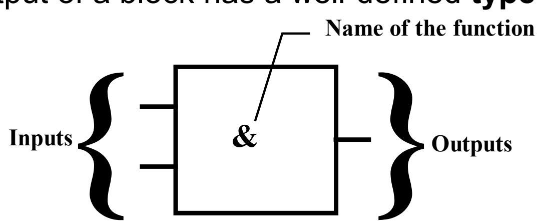
The **Functional Block Diagram** (FBD) is a graphic language. It allows the programmer to build complex procedures by taking existing **functions** from the ISaGRAF library and **wiring** them together in the graphic diagram area.

E.5.1 FBD diagram main format

FBD diagram describes a function between **input variables** and **output variables**. A function is described as a set of **elementary function blocks**. Input and output variables are connected to blocks by **connection lines**. An output of a function block may also be connected to an input of another block.



An entire function operated by an FBD program is built with standard **elementary** function blocks from the ISaGRAF library. Each function block has a fixed number of **input connection points** and a fixed number of **output connection points**. A function block is represented by a single **rectangle**. The inputs are connected on its **left** border. The outputs are connected on its **right** border. An elementary function block performs a single **function** between its inputs and its outputs. The name of the function to be performed by the block is written in its rectangle symbol. Each input or output of a block has a well-defined **type**.



Input variables of an FBD program must be connected to input connection points of function blocks. The type of each variable must be the same as the type expected for the associated input. An input for FBD diagram can be a **constant** expression, any **internal** or **input** variable, or an **output** variable.

Output variables of an FBD program must be connected to output connection points of function blocks. The type of each variable must be the same as the type expected for the associated block output. An Output for FBD diagram can be any **internal** or **output** variable, or the name of the program (for **sub-programs** only). When an output is the name of the currently edited sub-program, it represents the assignment of the return value for the sub-program (returned to the calling program).

Input and output variables, inputs and outputs of the function blocks are wired together with **connection lines**. Single lines may be used to **connect** two logical points of the diagram:
- An input variable and an input of a function block
- An output of a function block and an input of another block
- An output of a function block and an output variable

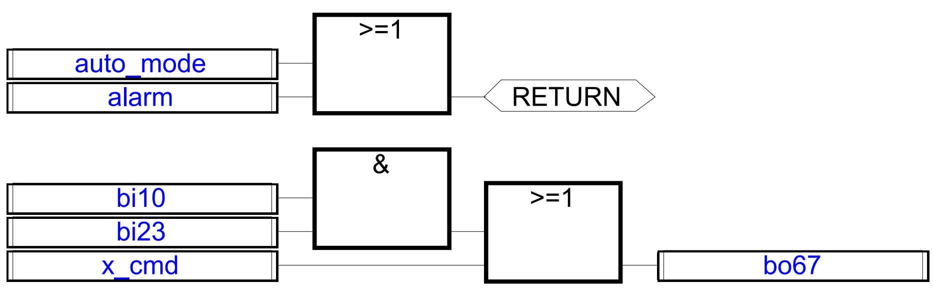
The connection is **oriented**, meaning that the line carries associated data from the left extremity to the right extremity. The left and right extremities of the connection line must be of the **same type**.

Multiple right connection can be used to broadcast an information from its left extremity to each of its right extremities. All the extremities of the connection must be of the same type.

E.5.2 RETURN statement

The "**<RETURN>**" keyword may occur as a diagram output. It must be connected to a boolean output connection point of a function block. The RETURN statement represents a **conditional end** of the program: if the output of the box connected to the statement has the boolean value **TRUE**, the end (remaining part) of the diagram is not executed.

(* Example of an FBD program using RETURN statement *)



(* ST equivalence: *)
If auto_mode OR alarm Then
      Return;
End_if;
bo67 := (bi10 AND bi23) OR x_cmd;

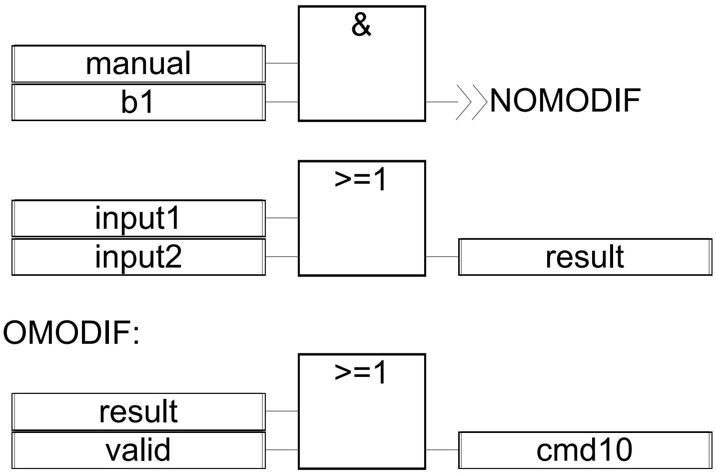
E.5.3 Jumps and labels

Labels and jumps are used to control the execution of the diagram. No other object may be connected on the right of a jump or label symbol. The following notations are used:

>>LAB ....jump to a label (label name is "LAB")
LAB: ....definition of a label (label name is "LAB")

If the connection line on the **left** of the jump symbol has the boolean state **TRUE**, the execution of the program directly jumps after the corresponding label symbol.

(* Example of an FBD program using labels and jumps *)

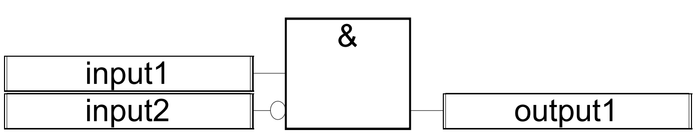



```
(* IL Equivalence: *)
     ld          manual
     and         b1
     jmpc        NOMODIF
     ld          input1
     or          input2
     st          result
NOMODIF:  ld  result
     or          valid
     st          cmd10
```

E.5.4 Boolean negation

A single connection line with its right extremity connected to an input of a function block can be terminated by a **boolean negation**. The negation is represented by a small circle. When a boolean negation is used, the left and right extremities of the connection line must have the **BOOLEAN** type.

(* Example of an FBD program using a boolean negation *)



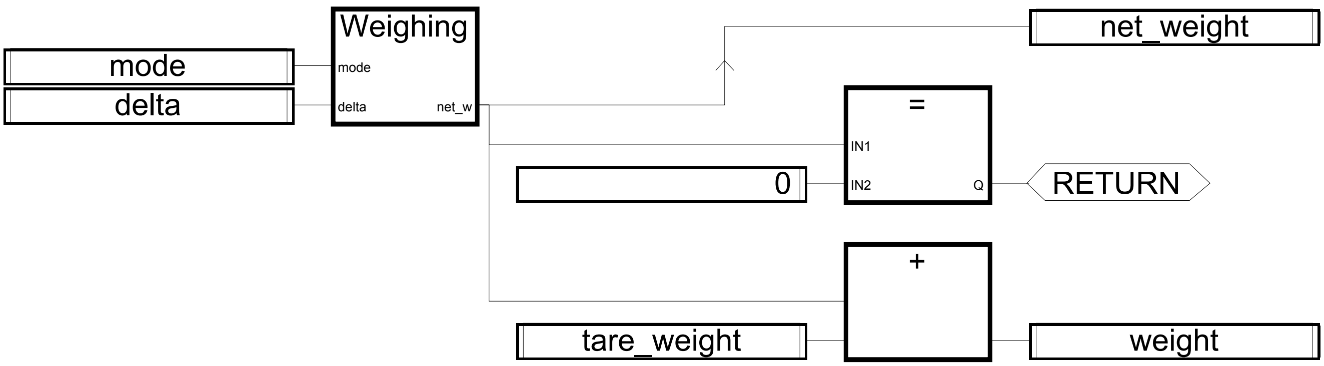
(* ST equivalence: *)
output1 := input1 AND NOT (input2);

E.5.5 Calling function or function blocks from the FBD

The FBD language enables the calling of sub-programs, functions or function blocks. A sub-program, or function or function block is represented by a function box. The name written in the box is the name of the sub-program or function or function blocks.

In case of a sub-program or a function, the return value is the only output of the function box. A function block can have more than one output.

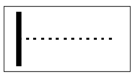
(* Example of an FBD program using SUB PROGRAM block *)

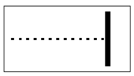


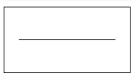

```
(* ST Equivalence *)
net_weight := Weighing (mode, delta); (* call sub-program *)
If (net_weight = 0) Then Return; End_if;
weight := net_weight + tare_weight;
```

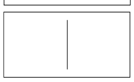
E.6 LD language

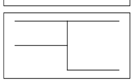
Ladder Diagram (LD) is a graphic representation of boolean equations, combining **contacts** (input arguments) with **coils** (output results). The LD language enables the description of tests and modifications of **boolean** data by placing **graphic symbols** into the program chart. LD graphic symbols are organized within the chart exactly as an electric contact diagram. LD diagrams are connected on the left side and on the right side to vertical **power rails**. These are basic graphic components of an LD diagram:

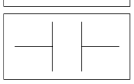

Left vertical power rail

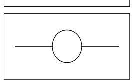

Right vertical power rail


Horizontal connection line


Vertical connection line

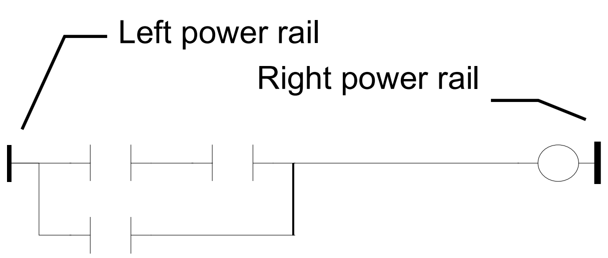

Multiple connection lines (all connected together)


Contact associated with a variable

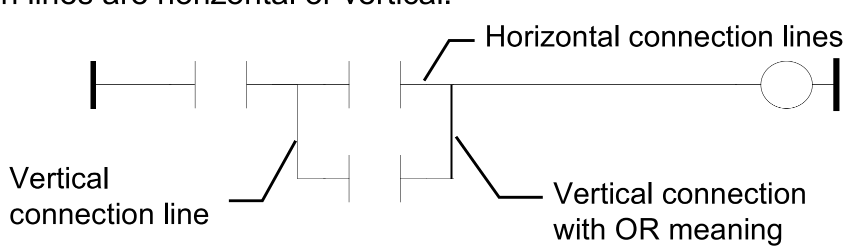

Coil associated to an output or to an internal variable

E.6.1 Power rails and connection lines

An LD diagram is limited on the left and right side by vertical lines, named **left power rail** and **right power rail** respectively.



LD diagram graphic symbols are connected to power rails or to other symbols by **connection lines**. Connection lines are horizontal or vertical.



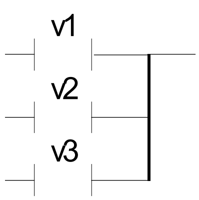
Each line segment has a boolean state **FALSE** or **TRUE**. The boolean state is the same for all the segments directly linked together. Any horizontal line connected to the left **vertical power rail** has the **TRUE** state.

E.6.2 Multiple connection

The boolean state given to a single horizontal connection line is the same on the left and on the right extremities of the line. Combining horizontal and vertical connection lines enables the building of **multiple connections**. The boolean state of the extremities of a multiple connection follows logic rules.

A **multiple connection on the left** combines **more than one** horizontal lines connected on the **left** side of a vertical line, and **one** line connected on its **right** side. The boolean state of the right extremity is the **LOGICAL OR** between all the left extremities.

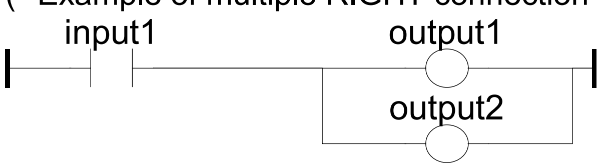
(* Example of multiple LEFT connection *)



(* right extremity state is (v1 OR v2 OR v3) *)

A **multiple connection on the right** combines **one** horizontal line connected on the **left** side of a vertical line, and **more than one** line connected on its **right** side. The boolean state of the left extremity is propagated into each of the right extremities.

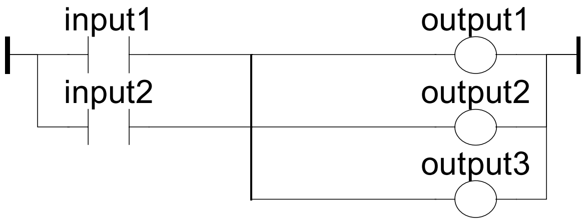
(* Example of multiple RIGHT connection *)



(* ST equivalence: *)
output1 := input1;
output2 := input1;

A **multiple connection on the left and on the right** combines **more than one** horizontal line connected on the **left** side of a vertical line, and **more than one** line connected on its **right** side. The boolean state of each of the right extremities is the **LOGICAL OR** between all the left extremities

(* Example of multiple LEFT and RIGHT connection *)



(* ST Equivalence: *)
output1 := input1 OR input2;
output2 := input1 OR input2;
output3 := input1 OR input2;

E.6.3 Basic LD contacts and coils

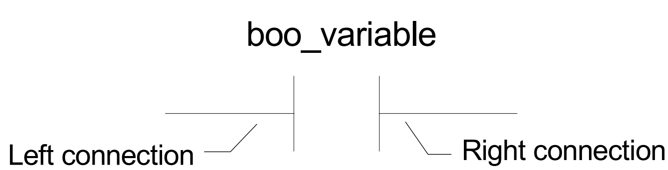
There are several symbols available for input contacts:
- Direct contact
- Inverted contact
- Contacts with edge detection

There are several symbols available for output coils:
- Direct coil
- Inverted coil
- SET coil
- RESET coil
- Coils with edge detection

The name of the variable is written above any of these graphic symbols:

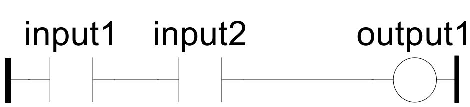
- ***Direct contact***

A direct contact enables a **boolean operation** between a **connection line** state and a boolean **variable**.



The state of the connection line on the right of the contact is the **LOGICAL AND** between the state of the left connection line and the value of the variable associated with the contact.

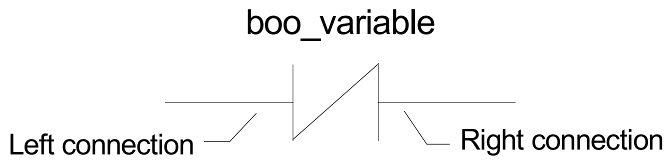
(* Example using DIRECT contacts *)



(* ST Equivalence: *)
output1 := input1 AND input2;

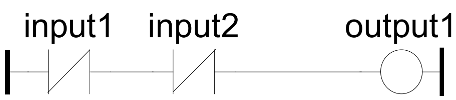
- ***Inverted contact***

An inverted contact enables a **boolean operation** between a **connection line** state and the boolean negation of a boolean **variable**.



The state of the connection line on the right of the contact is the **LOGICAL AND** between the state of the left connection line and the **boolean negation** of the value of the variable associated with the contact.

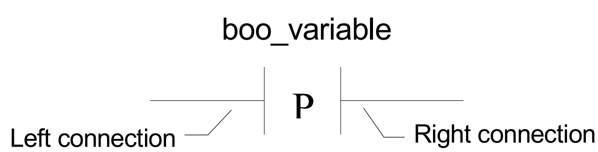
(* Example using INVERTED contacts *)



(* ST Equivalence: *)
output1 := NOT (input1) AND NOT (input2);

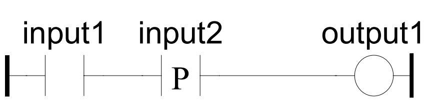
- ***Contact with rising edge detection***

This contact (positive) enables a **boolean operation** between a **connection line** state and the rising edge of a boolean **variable**.



The state of the connection line on the right of the contact is set to **TRUE** when the state of the connection line on the left is **TRUE**, and the state of the associated variable **rises** from FALSE to TRUE. It is reset to FALSE in all other cases.

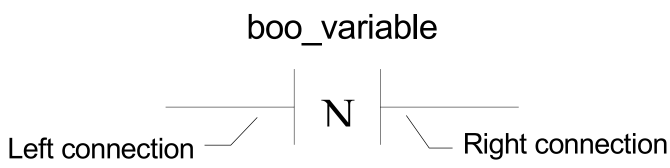
(* Example using RISING EDGE contacts *)



(* ST Equivalence: *)
output1 := input1 AND (input2 AND NOT (input2prev));
(* input2prev is the value of input2 at the previous cycle *)

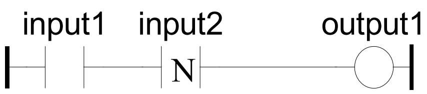
- ***Contact with falling edge detection***

This contact (negative) enables a **boolean operation** between a **connection line** state and the falling edge of a boolean **variable**.



The state of the connection line on the right of the contact is set to **TRUE** when the state of the connection line on the left is **TRUE**, and the state of the associated variable **falls** from TRUE to FALSE. It is reset to FALSE in all other cases.

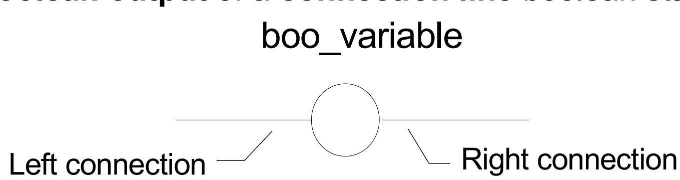
(* Example using FALLING EDGE contacts *)



(* ST Equivalence: *)
output1 := input1 AND (NOT (input2) AND input2prev);
(* input2prev is the value of input2 at the previous cycle *)

- ***Direct coil***

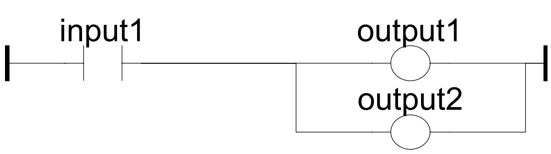
Direct coils enable a **boolean output** of a **connection line** boolean state.



The associated variable is assigned with the boolean **state of the left connection**. The state of the left connection is propagated into the right connection. The right connection may be connected to the right vertical power rail.

The associated boolean variable must be **OUTPUT** or **INTERNAL**.
The associated name can be the name of the program (for **sub-programs** only). This corresponds to the assignment of the return value of the sub-program.

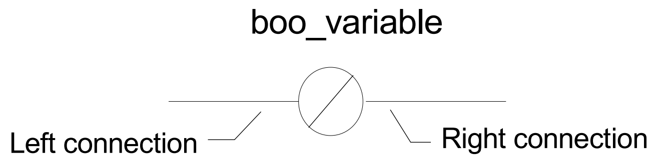
(* Example using DIRECT coils *)



(* ST Equivalence: *)
output1 := input1;
output2 := input1;

- ***Inverted coil***

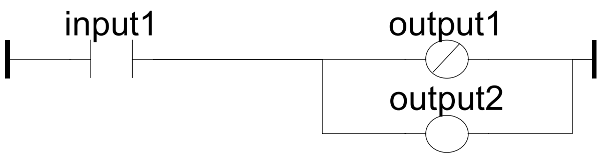
Inverted coils enable a **boolean output** according to the boolean **negation** of a **connection line** state.



The associated variable is assigned with the boolean **negation** of the **state of the left connection**. The state of the left connection is propagated into the right connection. Right connection may be connected to the right vertical power rail.

The associated boolean variable must be **OUTPUT** or **INTERNAL**.
The associated name can be the name of the program (for **sub-programs** only). This corresponds to the assignment of the return value of the sub-program.

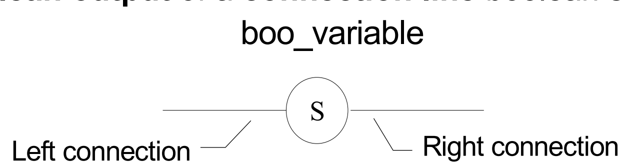
(* Example using INVERTED coils *)



(* ST Equivalence: *)
output1 := NOT (input1);
output2 := input1;

- ***SET coil***

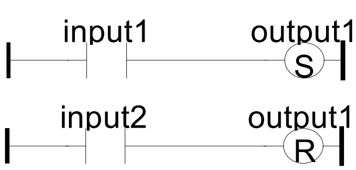
"Set" coils enable a **boolean output** of a **connection line** boolean state.



The associated variable is **SET TO TRUE** when the boolean **state of the left connection** becomes TRUE. The output variable keeps this value until an inverse order is made by a "RESET" coil. The state of the left connection is propagated into the right connection. Right connection may be connected to the right vertical power rail.

The associated boolean variable must be **OUTPUT** or **INTERNAL**.

(* Example using "SET" and "RESET" coils *)

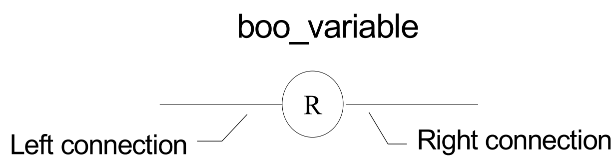


(* ST Equivalence: *)
IF input1 THEN

```
  output1 := TRUE;
END_IF;
IF input2 THEN
  output1 := FALSE;
END_IF;
```

- ***RESET coil***

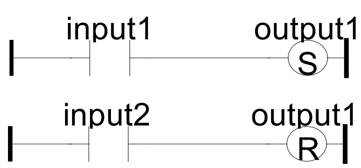
"Reset" coils enable **boolean output** of a **connection line** boolean state.



The associated variable is **RESET TO FALSE** when the boolean **state of the left connection** becomes **TRUE**. The output variable keeps this value until an inverse order is made by a "SET" coil. The state of the left connection is propagated into the right connection. Right connection may be connected to the right vertical power rail.

The associated boolean variable must be **OUTPUT** or **INTERNAL**.

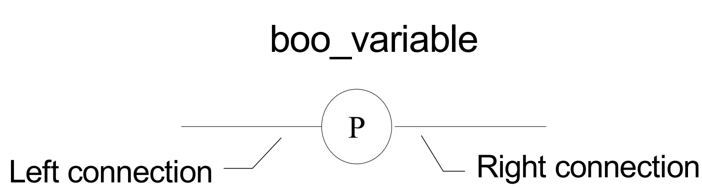
(* Example using "SET" and "RESET" coils *)




```
(* ST Equivalence: *)
IF input1 THEN
  output1 := TRUE;
END_IF;
IF input2 THEN
  output1 := FALSE;
END_IF;
```

- ***Coil with rising edge detection***

"Positive" coils enable **boolean output** of a **connection line** boolean state. This type of coils are only available using the Quick ladder editor.



The associated variable is set to **TRUE** when the boolean **state of the left connection** rises from FALSE to TRUE. The output variable resets to FALSE in all other cases. The state of the left connection is propagated into the right connection. Right connection may be connected to the right vertical power rail.

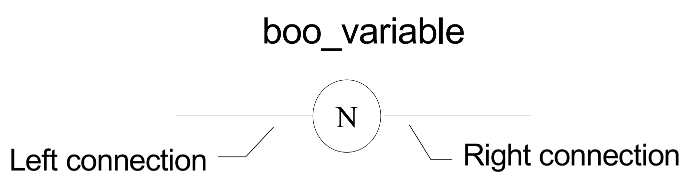
The associated boolean variable must be **OUTPUT** or **INTERNAL**.

(* Example using a "Positive" coil *)

(* ST Equivalence: *)
IF (input1 and NOT(input1prev)) THEN
  output1 := TRUE;
ELSE
  output1 := FALSE;
END_IF;
(* input1prev is the value of input1 at the previous cycle *)

- ***Coil with falling edge detection***

"Negative" coils enable **boolean output** of a **connection line** boolean state. This type of coils are only available using the Quick ladder editor.



The associated variable is set to **TRUE** when the boolean **state of the left connection** falls from TRUE to FALSE. The output variable resets to FALSE in all other cases. The state of the left connection is propagated into the right connection. Right connection may be connected to the right vertical power rail.

The associated boolean variable must be **OUTPUT** or **INTERNAL**.

(* Example using a "Positive" coil *)

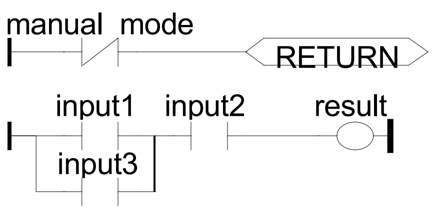
(* ST Equivalence: *)
IF (NOT(input1) and input1prev) THEN
  output1 := TRUE;
ELSE
  output1 := FALSE;
END_IF;
(* input1prev is the value of input1 at the previous cycle *)

E.6.4 RETURN statement

The **RETURN** label can be used as an output to represent a conditional end of the program. No connection can be put on the right of a RETURN symbol.

If the **left connection** line has the **TRUE** boolean state, the program ends without executing the equations entered on the following lines of the diagram.
Note: When the LD program is a sub-program, its name has to be associated with an output coil to set the return value (returned to the calling program).

(* Example using RETURN symbol *)



(* ST Equivalence: *)
If Not (manual_mode) Then RETURN; End_if;
result := (input1 OR input3) AND input2;

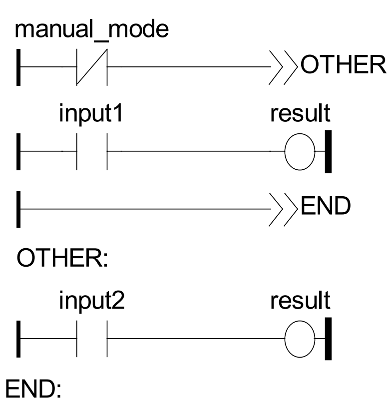
E.6.5 Jumps and labels

Labels, conditional and unconditional JUMPS symbols, can be used to control the execution of the diagram. No connection can be put on the right of the label and jump symbol. The following notations are used:

>>LAB ....jump to label named "LAB"
LAB: ....definition of the label named "LAB"

If the **connection on the left** of the jump symbol has the **TRUE** boolean state, the program execution is driven after the label symbol.

(* Example using JUMP and LABEL symbols *)



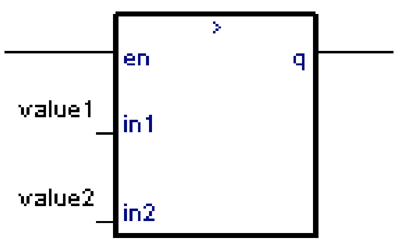
```
(* IL Equivalence: *)
      ldn     manual_mode
      jmpc    other
      ld      input1
      st      result
      jmp     END
OTHER:  ld      input2
      st      result
END:        (* end of program *)
```

E.6.6 Blocks in LD

Using the Quick LD editor, you connect function boxes to boolean lines. A function can actually be an operator, a function block or a function. As all blocks do not have always a boolean input and/or a boolean output, inserting blocks in an LD diagram leads to the addition of new parameters EN, ENO to the block interface. The EN, ENO parameters are not added if you use the FBD/LD editor as you can connect the variable with the required type.

- ***The "EN" input***

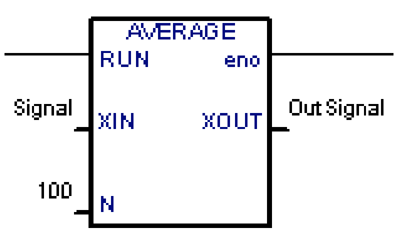
On some operators, functions or function blocks, the first input does not have boolean data type. As the first input must always be connected to the rung, another input is automatically inserted at the first position, called "**EN**". The block is executed only if the **EN** input is TRUE. Below is the example of a comparison operator, and the equivalent code expressed in ST:




```
IF rung_state THEN
   q := (value1 > value 2);
ELSE
   q := FALSE;
END_IF;
(* continue rung with q state *)
```

- ***The "ENO" output***

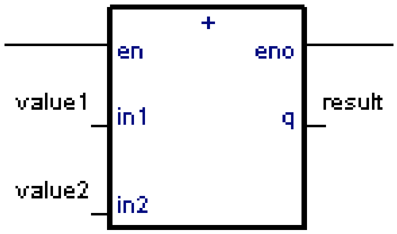
On some operators, functions or function blocks, the first output does not have boolean data type. As the first output must always be connected to the rung, another output is automatically inserted at the first position, called "**ENO**". The **ENO** output always takes the same state as the first input of the block. Below is an example with AVERAGE function block, and the equivalent code expressed in ST:




```
AVERAGE(rung_state, Signal, 100);
OutSignal := AVERAGE.XOUT;
eno := rung_state;
(* continue rung with eno state *)
```

- ***The "EN" and "ENO" parameters***

On some cases, both **EN** and **ENO** are required. Below is an example with an arithmetic operator, and the equivalent code expressed in ST:




```
IF rung_state THEN
   result := (value1 + value2);
END_IF;
eno := rung_state;
(* continue rung with eno state *)
```

E.7 ST language

ST (**Structured Text**) is a high level structured language designed for automation processes. This language is mainly used to implement complex procedures that cannot be easily expressed with graphic languages. ST is the default language for the description of the actions within the steps and conditions attached to the transitions of the **SFC** language.

E.7.1 ST main syntax

An ST program is a list of ST **statements**. Each statement ends with a semi-colon (**";"**) separator. Names used in the source code (variable identifiers, constants, language keywords...) are separated with **inactive separators** (space character, end of line or tab stops) or by **active separators**, which have a well defined significance (for example, the **">"** separator indicates a "greater than" comparison. Comments may be freely inserted into the text. A comment must begin with **"(*"** and ends with **"*)"**. Each statement terminates with a semi-colon (**";"**) separator. These are basic types of ST statements:

- **assignment** statement (variable := expression;)
- **sub-program** or **function** call
- **function block** call
- **selection** statements (IF, THEN, ELSE, CASE...)
- **iteration** statements (FOR, WHILE, REPEAT...)
- **control** statements (RETURN, EXIT...)
- special statements for links with other languages such as **SFC**

Inactive separators may be freely entered between active separators, constant expressions and identifiers. ST inactive separators are: **Space** (blank) character, **Tabs** and **End of line** character. Unlike line-formatted languages such as IL, end of lines may be entered anywhere in the program. The rules shown below should be followed when using inactive separators to increase ST program readability:

- Do not write more than one statement on one line
- Use tabs to indent complex statements
- Insert comments to increase readability of lines or paragraphs

E.7.1 Expression and parentheses

ST expressions combine ST **operators** and variable or constant **operands**. For each single expression (combining operands with one ST operator), the **type** of the operands must be the same. This single expression has the same type as its operands, and can be used in a more complex expression. For example :

```
(boo_var1 AND boo_var2)         has BOO type
not (boo_var1)      has BOO type
(sin (3.14) + 0.72) has REAL ANALOG type
(t#1s23 + 1.78)     is an invalid expression
```

Parentheses are used to isolate sub parts of the expression, and to explicitly order the priority of the operations. When no parentheses are given for a complex expression, the operation sequence is implicitly given by the default **priority** between ST operators. For example:

2 + 3 * 6	equals 2+18=20	because multiplication operator has a higher priority
(2+3) * 6	equals 5*6=30	priority is given by parenthesis

Warning: A maximum number of **8** levels of parentheses can be nested within an expression.

E.7.3 Function or function block calls

Standard ST function calls may be used for each of following objects:
- Sub-programs
- Library functions and function blocks written in IEC languages
- "C" functions and function blocks
- Type conversion functions

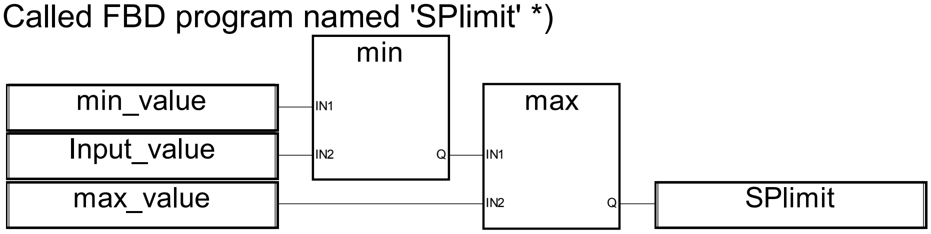
Calling sub-programs or functions

Name: name of the called sub-program or library function written in IEC language or in "C"
Meaning: calls a ST, IL, LD or FBD sub-program or function or a "C" function and gets its return value
Syntax: <**variable**> **:=** <**subprog**> **(**<**par1**>**, ...** <**parN**> **);**
Operands: The type of return value and calling parameters must follow the interface defined for the sub-program.
Return value: value returned by the sub-program

Sub-program calls may be used in any expression. They also may be used in an SFC transition.

Example1: Sub-program call

```
(* Main ST program *)
(* gets an analog value and converts it into a limited time value *)
ana_timeprog := SPlimit ( tprog_cmd );
appl_timer := tmr (ana_timeprog * 100);

(* Called FBD program named 'SPlimit' *)
```



Example2: Function call

(* functions used in complex expressions: min, max, right, mlen and left are standard "C" functions *)
limited_value := min (16, max (0, input_value));
rol_msg := right (message, mlen (message) - 1) + left (message, 1);

▭ ***Calling function blocks***

Name: name of the function block instance
Meaning: calls a function block from the ISaGRAF library or from the user's library and accesses its return parameters
Syntax: **(* call of the function block *)**
<**blockname**> **(** <**p1**>**,** <**p2**> ... **);**
(gets its return parameters *)
<**result**> **:=** <**blockname**>**.** <**ret_param1**>**;**
...
<**result**> **:=** <**blockname**>**.** <**ret_paramN**>**;**
Operands: parameters are expressions which match the type of the parameters specified for that function block
Return value: See Syntax to get the return parameters.

Consult the ISaGRAF library to find the meaning and type of each function block parameter. The function block instance (name of the copy) must be declared in the dictionary

Example :

(* ST program calling a function block *)

(* declare the instance of the block in the dictionary: *)
(* trigb1 : block R_TRIG - rising edge detection *)

(* function block activation from ST language *)
trigb1 (b1);
(* return parameters access *)
If (trigb1.Q) Then nb_edge := nb_edge + 1; End_if;

E.7.4 ST specific boolean operators

The following boolean operators are specific to the ST language:
- REDGE rising edge detection
- FEDGE falling edge detection

Other standard boolean operators such as:
- NOT boolean negation
- AND (&) logical AND
- OR logical OR
- XOR logical exclusive OR

can be used. Their description is to be found in the section 'Standard operators, function blocks and functions'.

▭ ***"REDGE" operator***

Name: **REDGE**
Meaning: evaluates the rising edge of a complete boolean expression
Syntax: **<edge> := REDGE (<boo_expression>,<memo_variable>);**
Operands: first operand is any boolean variable or complex expression
second operand is an internal boolean variable used to store the last state of the expression
Return value: TRUE when the expression changes from FALSE to TRUE
FALSE for all other cases

The rising edge of an expression cannot be detected more than once in the same execution cycle, using the REDGE operator. This operator can be used to describe the condition attached to an SFC transition.

Warning: The "memory" boolean variable used to store the last state of the expression cannot be used as a trigger for edges of different expressions.

When the expression is a boolean variable named "**xxx**", a unique internal variable named "**EDGE_xxx**" should be declared and used it in the REDGE expressions for this variable. This method ensures that the memory variable is not overwritten during other REDGE evaluations.

Example:

```
(* ST program using REDGE operator *)

(* this program counts the rising edges of a boolean input *)
(* Bi120 is an input boolean variable *)
(* Edge_Bi120 is the memory of the Bi120 variable state *)

If REDGE (Bi120, Edge_Bi120) Then
    Counter := Counter + 1;
End_if;
```

Note: this operator is not in the IEC1131-3 norm. You may prefer the use of R_TRIG standard block. It has been kept for compatibility reasons.

- ***"FEDGE" operator***

Name: **FEDGE**
Meaning: evaluates the falling edge of a boolean expression
Syntax: **<edge> := FEDGE (<boo_expression>, <memo_variable>);**
Operands: first operand is any boolean variable or complex expression
second operand is an internal boolean variable used to store
the last state of the expression
Return value: TRUE when the expression changes from TRUE to FALSE
FALSE for all other cases

The falling edge of an expression cannot be detected more than once in the same execution cycle, using the REDGE operator. The operator can be used to describe the condition attached to an SFC transition.

Warning: The "memory" boolean variable used to store the last state of the expression cannot be used as a trigger for edges of different expressions.

When the expression is a boolean variable named "**xxx**", a unique internal variable named "**EDGE_xxx**" should be declared and used it in the FEDGE expressions for this variable. This method ensures that the memory variable is not overwritten during other FEDGE evaluations.

Example:

```
(* ST program using FEDGE operator *)

(* this program counts the falling edges of a boolean input *)
(* Bi120 is an input boolean variable *)
(* Edge_Bi120 is the memory of the Bi120 variable state *)

If FEDGE (Bi120, Edge_Bi120) Then
    Counter := Counter + 1;
End_if;
```

Note: this operator is not in the IEC1131-3 norm. You may prefer the use of F_TRIG standard block. It has been kept for compatibility reasons.

E.7.5 ST basic statements

The basic statements of the ST language are:
- Assignment
- RETURN statement
- IF-THEN-ELSIF-ELSE structure
- CASE statement
- WHILE iteration statement
- REPEAT iteration statement
- FOR iteration statement
- EXIT statement

- ***Assignment***

Name: **:=**
Meaning: assigns a variable to an expression
Syntax: **<variable> := <any_expression> ;**
Operands: variable must be internal or output
variable and expression must have the same type

The expression can be a call to a sub-program or a function from the ISaGRAF library

Example:

```
(* ST program with assignments *)

(* variable <<= variable *)
bo23 := bo10;
```

```
(* variable <<= expression *)
bo56 := bx34 OR alrm100 & (level >= over_value);
result := (100 * input_value) / scale;

(* assignment with sub-program return value *)
rc := PSelect ( );

(* assignment with function call *)
limited_value := min (16, max (0, input_value) );
```

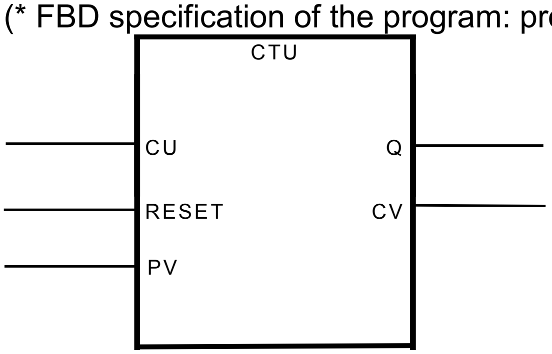
- ***RETURN statement***

Name: **RETURN**
Meaning: terminates the execution of the current program
Syntax: **RETURN ;**
Operands: (none)

In an SFC action block, the RETURN statement indicates the end of the execution of that block only.

Example:

(* FBD specification of the program: programmable counter *)



(* ST implementation of the program, using RETURN statement *)

```
If not (CU) then
    Q := false;
    CV := 0;
    RETURN;   (* terminates the program *)
end_if;

if R then
    CV := 0;
else
    if (CV < PV) then
      CV := CV + 1;
    end_if;
end_if;
Q := (CV >= PV);
```

IF-THEN-ELSIF-ELSE statement

Name: **IF ... THEN ... ELSIF ... THEN ... ELSE ... END_IF**
Meaning: executes one of two lists of ST statements
selection is made according to the value
of a boolean expression
Syntax: **IF <boolean_expression> THEN**
<statement> ;
<statement> ;
...
ELSIF <boolean_expression> THEN
<statement> ;
<statement> ;
...
ELSE
<statement> ;
<statement> ;
...
END_IF;

The ELSE and ELSIF statements are optional. If the ELSE statement is not written, no instruction is executed when the condition is FALSE.

Example:

```
(* ST program using IF statement *)

IF manual AND not (alarm) THEN
    level := manual_level;
    bx126 := bi12 OR bi45;
ELSIF over_mode THEN
    level := max_level;
ELSE
    level := (lv16 * 100) / scale;
END_IF;

(* IF structure without ELSE *)
If overflow THEN
        alarm_level := true;
END_IF;
```

CASE statement

Name: **CASE ... OF ... ELSE ... END_CASE**
Meaning: executes one of several lists of ST statements
selection is made according to an integer expression
Syntax: **CASE <integer_expression> OF**
<value> : <statements> ;
<value> , <value> : <statements> ;
...
ELSE

<statements> ;
END_CASE;

Case values must be integer constant expressions. Several values, separated by comas, can lead to the same list of statements. The ELSE statement is optional.

Example:

```
(* ST program using CASE statement *)

CASE error_code OF
    255:   err_msg := 'Division by zero';
      fatal_error := TRUE;
    1:     err_msg := 'Overflow';
    2, 3:  err_msg := 'Bad sign';
ELSE
    err_msg := 'Unknown error';
END_CASE;
```

- ***WHILE statement***

Name: **WHILE ... DO ... END_WHILE**
Meaning: iteration structure for a group of ST statements
the "continue" condition is evaluated BEFORE any iteration
Syntax: **WHILE <boolean_expression> DO**
<statement> ;
<statement> ;
...
END_WHILE ;

<u>Warning</u>: Because ISaGRAF is a **synchronous** system, input variables are not refreshed during WHILE iterations. The change of state of an input variable cannot be used to describe the condition of a WHILE statement.

Example:

```
(* ST program using WHILE statement *)

(* this program uses specific "C" functions to read characters *)
(* on a serial port *)

string := ''; (* empty string *)
nbchar := 0;

WHILE ((nbchar < 16) & ComIsReady ( )) DO
    string := string + ComGetChar ( );
    nbchar := nbchar + 1;
END_WHILE;
```

- ***REPEAT statement***

Name: **REPEAT ... UNTIL ... END_REPEAT**
Meaning: iteration structure for a group of ST statements
the "continue" condition is evaluated AFTER any iteration
Syntax: **REPEAT**
<statement> ;
<statement> ;
...
UNTIL <boolean_condition>
END_REPEAT ;

Warning: Because ISaGRAF is a **synchronous** system, input variables are not refreshed during REPEAT iterations. The change of state of an input variable cannot be used to describe the ending condition of a REPEAT statement.

Example:

```
(* ST program using REPEAT statement *)

(* this program uses specific "C" functions to read characters *)
(* on a serial port *)

string := ''; (* empty string *)
nbchar := 0;
IF ComIsReady ( ) THEN
   REPEAT
    string := string + ComGetChar ( );
    nbchar := nbchar + 1;
   UNTIL ( (nbchar >= 16) OR NOT (ComIsReady ( )) )
   END_REPEAT;
END_IF;
```

- ***FOR statement***

Name: **FOR ... TO ... BY ... DO ... END_FOR**
Meaning: executes a limited number of iterations,
using an integer analog index variable
Syntax: **FOR <index> := <mini> TO <maxi> BY <step> DO**
<statement> ;
<statement> ;
END_FOR;
Operands: **index**: internal analog variable increased at any loop
mini: initial value for index (before first loop)
maxi: maximum allowed value for index
step: index increment at each loop

The [BY step] statement is optional. If not specified, the increment step is 1

Warning: Because ISaGRAF is a **synchronous** system, input variables are not refreshed during FOR iterations.

This is the "while" equivalent of a FOR statement:

```
index := mini;
while (index <= maxi) do
  <statement> ;
  <statement> ;
  index := index + step;
end_while;
```

Example:

```
(* ST program using FOR statement *)
(* this program extracts the digit characters of a string *)

length := mlen (message);
target := ''; (* empty string *)
FOR index := 1 TO length BY 1 DO
      code := ascii (message, index);
      IF (code >= 48) & (code <= 57) THEN
        target := target + char (code);
      END_IF;
END_FOR;
```

- ***EXIT statement***

Name: **EXIT**
Meaning: exit from a FOR, WHILE or REPEAT iteration statement
Syntax: **EXIT;**
Operands: (none)

The EXIT is commonly used within an IF statement, inside a FOR, WHILE or REPEAT block.

Example:

```
(* ST program using EXIT statement *)
(* this program searches for a character in a string *)

length := mlen (message);
found := NO;
FOR index := 1 TO length BY 1 DO
      code := ascii (message, index);
      IF (code  = searched_char) THEN
        found := YES;
        EXIT;
      END_IF;
END_FOR;
```

E.7.6 ST extensions

The following functions are extensions of the ST language:

- TSTART - TSTOP: timer control

The following statements and functions are available to control the execution of the SFC child programs. They may be used inside ACTION(): ... END_ACTION; blocks in SFC steps.

- GSTART starts an SFC program
- GKILL kills an SFC program
- GFREEZE freezes an SFC program
- GRST restarts a frozen SFC program
- GSTATUS gets current status of an SFC program

Warning: These functions are not in the IEC 1131-3 norm.
Easy equivalent can be found for GSTART and GKILL using the following syntax in the SFC step:

```
child_name(S); (* equivalent to GSTART(child_name); *)
child_name(R); (* equivalent to GKILL(child_name); *)
```

The following fields can be used to access the status of an SFC step:

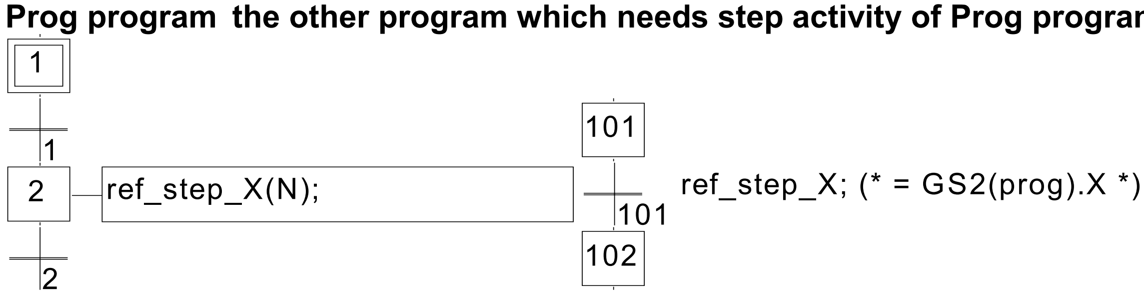
GSnnn.x boolean value that represents the activity of the step
GSnnn.t time elapsed since the last activation of the step
("**nnn**" is the reference number of the SFC step)

It is also possible to test the activity of a step declared in another SFC program, by using the following syntax:

GSnnn(progname).x

Warning: referencing a step of an other program, using this syntax is not in the IEC 1131-3 norm. An easy way to do the same respecting IEC rules, is to declare a global boolean variable in the dictionary which will represent the step activity to be tested (for example ref_step_X). Then you insert in the step, the variable with the N qualifier (ref_step_X(N);). Then in the program which wants to test the activity of the step, you use the variable.

Prog program the other program which needs step activity of Prog program



TSTART statement

Name: **TSTART**
Meaning: starts the counting of a timer variable
timer value is not modified by the TSTART command, i.e. the counting starts from the current value of the timer.
Syntax: **TSTART (<timer_variable>);**
Operands: any inactive timer variable

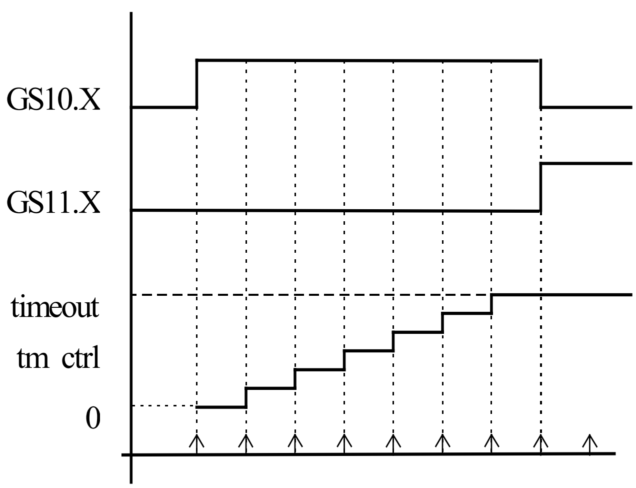
Return value: (none)

Example:

(* SFC program using TSTART and TSTOP statements *)

```
10   ACTION(P):
       bo100 := TRUE; (* boolean output *)
       tm_ctrl := t#0s;
       TSTART(tm_ctrl);
     END_ACTION;

   bi100 OR (tm_ctrl > time_out);

11   ACTION(P):
       TSTOP(tm_ctrl);
       alarm := not(bi100);
     END_ACTION;
```

Time diagram if bi100 is always FALSE:



The timer keeps the same value during one cycle.

▭ ***TSTOP statement***

Name: **TSTOP**
Meaning: stops updating a timer variable
timer value is not modified by the TSTOP command
Syntax: **TSTOP (<timer_variable>);**
Operands: any active timer variable
Return value: (none)

Example: See TSTART (the function is described above)

▭ ***GSTART statement***

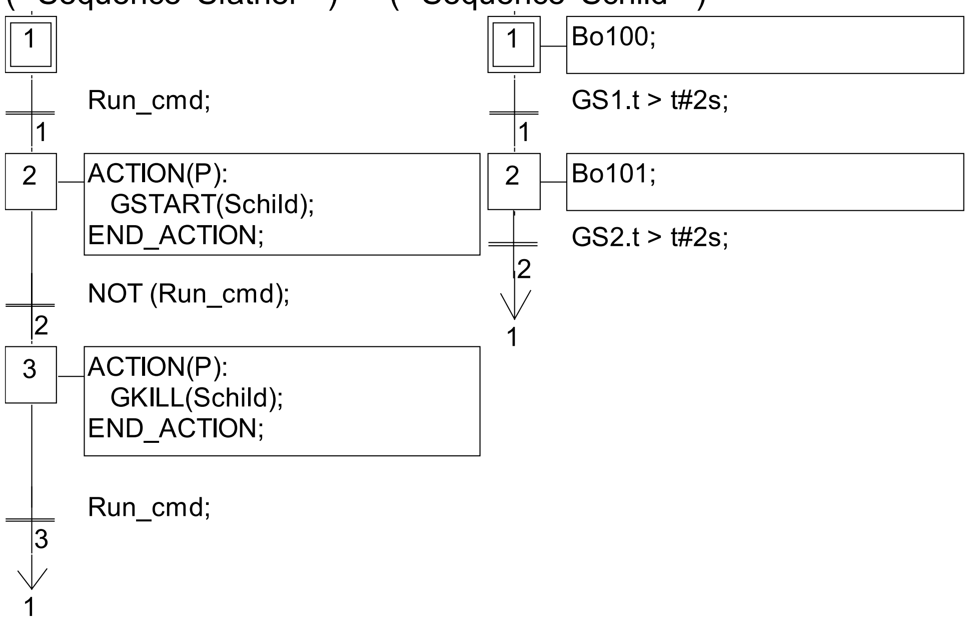
Name: **GSTART**
Meaning: starts a child SFC program by putting a token

into each of its initial steps
Syntax: **GSTART (<child_program>);**
Operands: the specified SFC program must be a child of the one in which the statement is written
Return value: (none)

Children of the child program are not automatically started by the GSTART statement.
Note: As GSTART is not in the IEC 1131-3 norm, prefer the use of the S qualifier, with the following syntax to start a child SFC:

```
Child_name(S);
```

Example: Use of GSTART and GKILL
(* Sequence 'Sfather' *) (* Sequence 'Schild' *)



- ***GKILL statement***

Name: **GKILL**
Meaning: kills a child SFC program by removing the tokens currently existing in its steps
Syntax: **GKILL (<child_program>);**
Operands: the specified SFC program must be a child of the one in which the statement is written
Return value: (none)

Children of the child program are automatically killed with the specified program.
Note: As GKILL is not in the IEC 1131-3 norm, prefer the use of the R qualifier, with the following syntax to kill a child SFC:

```
Child_name(R);
```

Example: See GSTART (function described above)

- ***GFREEZE statement***

Name: **GFREEZE**
Meaning: Suspends the execution of a child SFC program.

Frozen program can be restarted by the GRST statement.

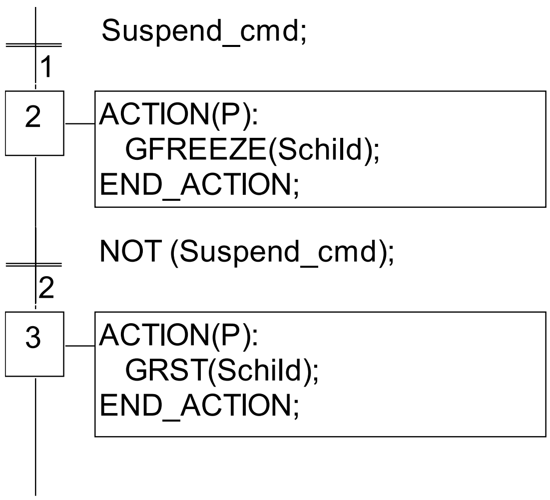
Syntax: **GFREEZE (<child_program>);**

Operands: the specified SFC program must be a child of the one in which the statement is written

Return value: (none)

Children of the child program are automatically frozen along with the specified program.
Note: GFREEZE is not in the IEC 1131-3 norm.

Example:



- ***GRST statement***

Name: **GRST**

Meaning: Restarts a child SFC program frozen by the GFREEZE statement.

Syntax: **GRST (<child_program>);**

Operands: the specified SFC program must be a child of the one in which the statement is written

Return value: (none)

Children of the child program are automatically restarted by the GRST statement
Note: GRST is not in the IEC 1131-3 norm.

Example: See GFREEZE (function described above)

- ***GSTATUS statement***

Name: **GSTATUS**

Meaning: returns the current status of an SFC program

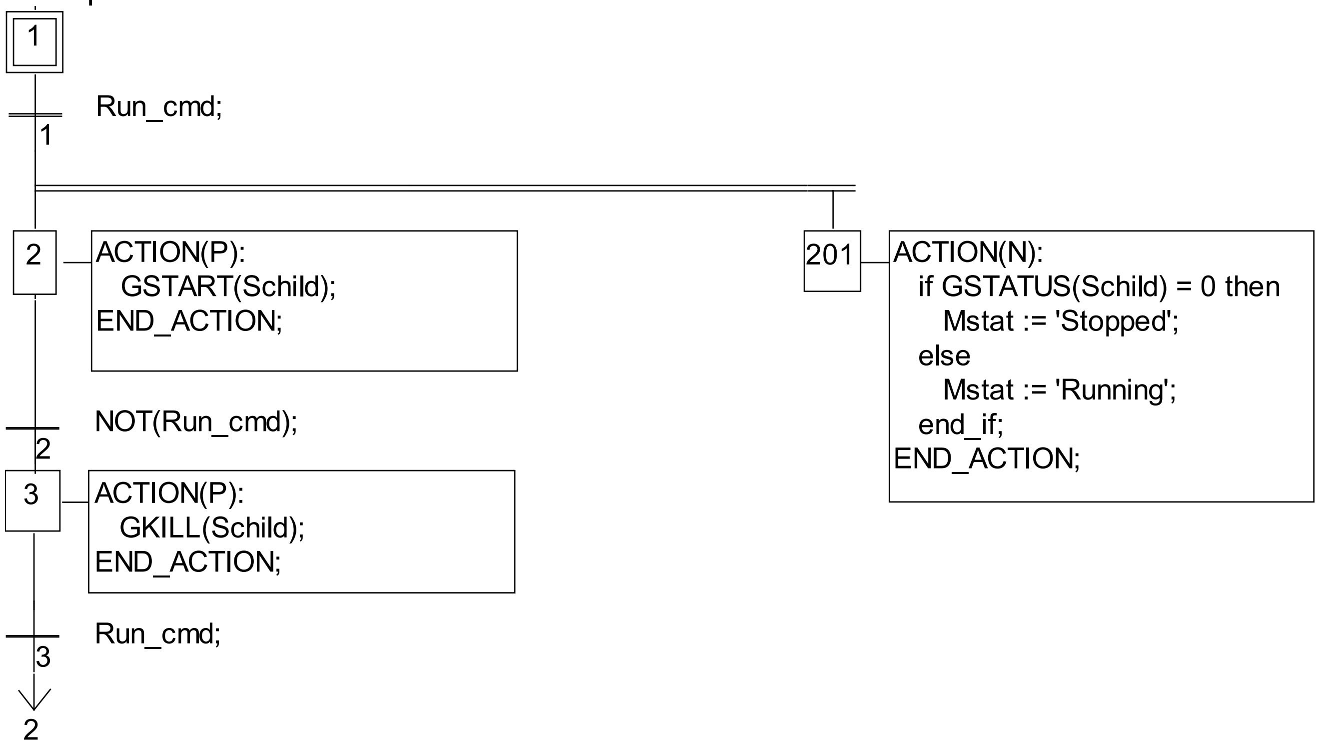
Syntax: **<ana_var> := GSTATUS (<child_program>);**

Operands: the specified SFC program must be a child of the one in which the statement is written

Return value: 0 = program is inactive (killed)
1 = program is active (started)
2 = program is frozen

Note: GSTATUS is not in the IEC 1131-3 norm.

Example:


1
Run_cmd;
1
2
ACTION(P):
GSTART(Schild);
END_ACTION;
NOT(Run_cmd);
2
3
ACTION(P):
GKILL(Schild);
END_ACTION;
Run_cmd;
3
2
201
ACTION(N):
if GSTATUS(Schild) = 0 then
Mstat := 'Stopped';
else
Mstat := 'Running';
end_if;
END_ACTION;

E.8 IL language

Instruction List, or **IL** is a low level language. Instructions always relate to the **current result** (or **IL register**). The operator indicates the operation that must be made between the current value and the operand. The result of the operation is stored again in the current result.

E.8.1 IL main syntax

An IL program is a list of **instructions**. Each instruction must begin on a new line, and must contain an **operator**, completed with optional **modifiers** and, if necessary, for the specific operation, one or more **operands**, separated with commas (**','**). A **label** followed by a colon (**':'**) may precede the instruction. If a **comment** is attached to the instruction, it must be the last component of the line. Comments always begin with **'(*'** and ends with **'*)'**. Empty lines may be entered between instructions. Comments may be put on empty lines. Below are examples of instruction lines:

Label	*Operator*	*Operand*	*Comments*
Start:	LD	IX1	(* push button *)
	ANDN	MX5	(* command is not forbidden *)
	ST	QX2	(* start motor *)

- ***Labels***

A **label** followed by a colon (**':'**) may precede the instruction. A label can be put on an empty line. Labels are used as operands for some operations such as jumps. Naming labels must conform to the following rules:
- name cannot exceed **16** characters
- first character must be a **letter**
- following characters must be **letters**, **digits** or **'_'** character

The same name cannot be used for more than one label in the same IL program. A label can have the same name as a variable.

- ***Operator modifiers***

The available operator modifiers are shown below. The modifier character must complete the name of the operator, with no blank characters between them:

N boolean negation of the operand
(delayed operation
C conditional operation

The **'N'** modifier indicates a boolean negation of the operand. For example, the instruction **ORN IX12** is interpreted as: **result := result OR NOT (IX12)**.

The parenthesis **'('** modifier indicates that the evaluation of the instruction must be delayed until the closing parenthesis **')'** operator is encountered.

The **'C'** modifier indicates that the attached instruction must be executed only if the current result has the boolean value TRUE (different than 0 for non-boolean values). The **'C'** modifier can be combined with the **'N'** modifier to indicate that the instruction must be executed only if the current result has the boolean value FALSE (or 0 for non-boolean values).

- ***Delayed operations***

Because there is only one IL register (current result), some operations may have to be delayed, so that the execution order or the instructions can be changed. Parentheses are used to indicate delayed operations:

'('	is a modifier	indicates the operation to be delayed
')'	is an operator	executes the delayed operation

The opening parenthesis **'('** modifier indicates that the evaluation of the instruction must be delayed until the closing parenthesis **')'** operator is encountered. For example, following sequence:

```
AND( IX12
OR   IX35
)
```

is interpreted as:

result := result AND (IX12 OR IX35)

E.8.2 IL operators

The following table summarizes the standard operators of the IL language:

Operator	*Modifiers*	*Operand*	*Description*
LD	N	Variable, constant	Loads operand
ST	N	Variable	Stores current result
S R		BOO variable BOO variable	Sets to TRUE Resets to FALSE
AND & OR XOR	N (N (N (N (	BOO BOO BOO BOO	boolean AND boolean AND boolean OR exclusive OR
ADD SUB MUL DIV	((((	variable, constant variable, constant variable, constant variable, constant	Addition Subtraction Multiplication Division

GT GE EQ LE LT NE	((((((	variable, constant variable, constant variable, constant variable, constant variable, constant variable, constant	Test: > Test: >= Test: = Test <= Test < Test <>
CAL JMP RET	C N C N C N	Function block instance name Label	Calls a function block Jumps to label Returns from sub-program
)			Executes delayed operation

In the next section, only operators which are specific to the IL language are described, other standard operators can be found in the section "standard operators, function blocks and functions".

LD operator

Operation loads a value in the current result
Allowed modifiers N
Operand constant expression
internal, input or output variable

Example:

```
        (* EXAMPLES OF LD OPERATIONS *)
LDex:   LD   false          (* result := FALSE boolean constant *)
    LD   true           (* result := TRUE boolean constant *)
    LD   123            (* result := integer constant *)
    LD   123.1          (* result := real constant *)
    LD   t#3ms          (* result := time constant *)
    LD   boo_var1       (* result := boolean variable *)
    LD   ana_var1       (* result := analog variable *)
    LD   tmr_var1       (* result := timer variable *)
    LDN  boo_var2       (* result := NOT ( boolean variable ) *)
```

ST operator

Operation stores the current result in a variable
the current result is not modified by this operation
Allowed modifiers N
Operand internal or output variable

Example:

```
        (* EXAMPLES OF ST OPERATIONS *)
STboo:  LD   false
    ST   boo_var1       (* boo_var1 := FALSE *)
    STN  boo_var2       (* boo_var2 := TRUE *)
STana:  LD   123
```

```
    ST      ana_var1        (* ana_var1 := 123 *)
STtmr:      LD   t#12s
    ST      tmr_var1        (* tmr_var1 := t#12s *)
```

▭ ***S operator***

Operation: stores the boolean value TRUE in a boolean variable, if the current result has the boolean value TRUE. No operation is processed if current result is FALSE. The current result is not modified by this operation

Allowed modifiers: (none)

Operand: output or internal boolean variable

Example:

```
            (* EXAMPLES OF S OPERATIONS *)
SETex:      LD   true                       (* current result := TRUE *)
    S       boo_var1        (* boo_var1 := TRUE *)
                            (* current result is not modified *)
    LD      false           (* current result := FALSE *)
    S       boo_var1        (* nothing done - boo_var1 unchanged *)
```

▭ ***R operator***

Operation stores the boolean value FALSE in a boolean variable, if the current result has the boolean value TRUE. No operation is processed if current result is FALSE. The current result is not modified by this operation

Allowed modifiers (none)

Operand output or internal boolean variable

Example:

```
            (* EXAMPLES OF R OPERATIONS *)
RESETex:LD   true                       (* current result := TRUE *)
    R       boo_var1        (* boo_var1 := FALSE *)
                            (* current result is not modified *)
    ST      boo_var2        (* boo_var2 := TRUE *)
    LD      false           (* current result := FALSE *)
    R       boo_var1        (* nothing done - boo_var1 unchanged *)
```

▭ ***JMP operator***

Operation jumps to the specified label

Allowed modifiers C N

Operand label defined in the same IL program

Example:

```
(* the following example tests the value of an analog selector (0 or 1 or 2)  *)
(* to set one from 3 output booleans. Test "is equal to 0" is made with       *)
(* the JMPC operator                 *)

JMPex:      LD   selector                   (* selector is 0 or 1 or 2 *)
    BOO                     (* conversion to boolean *)
```

```
    JMPC test1          (* if selector = 0 then *)
    LD   true
    ST   bo0            (* bo0 := true *)
    JMP  JMPend         (* end of the program *)
test1:   LD   selector
    SUB  1              (* decrease selector:  is now 0 or 1 *)
    BOO                 (* conversion to boolean *)
    JMPC test2          (* if selector = 0 then *)
    LD   true
    ST   bo1            (* bo1 := true *)
    JMP  JMPend         (* end of the program *)
test2:   LD   true                (* last possibility *)
    ST   bo2            (* bo2 := true *)
JMPend:                 (* end of the IL program *)
```

- ***RET operator***

Operation ends the current instruction list. If the IL sequence is a sub-program, the current result is returned to the calling program

Allowed modifiers C N

Operand (none)

Example:

```
(* the following example tests the value of an analog selector (0 or 1 or 2)  *)
(* to set one from 3 output booleans. Test "is equal to 0" is made with        *)
(* the JMPC operator              *)

JMPex:   LD   selector            (* selector is 0 or 1 or 2 *)
    BOO                 (* conversion to boolean *)
    JMPC test1          (* if selector = 0 then *)
    LD   true
    ST   bo0            (* bo0 := true *)
    RET                 (* end - return 0 *)
                        (* decrease selector *)
test1:   LD   selector
    SUB  1              (* selector is now 0 or 1 *)
    BOO                 (* conversion to boolean *)
    JMPC test2          (* if selector = 0 then *)
    LD   true
    ST   bo1            (* bo1 := true *)
    LD   1              (* load real selector value *)
    RET                 (* end - return 1 *)
                        (* last possibility *)
test2:   RETNC                    (* returns if the selector has *)
                        (* an invalid value *)
    LD   true
    ST   bo2            (* bo2 := true *)
    LD   2              (* load real selector value *)
                        (* end - return 2 *)
```

")" operator

Operation executes a delayed operation. The delayed operation was notified by '('
Allowed modifiers (none)
Operand (none)

Example:

```
(* The following program interleaves delayed operations: *)
(* res := a1 + (a2 * (a3 - a4) * a5) + a6; *)

Delayed:  LD    a1 (* result := a1; *)
          ADD(  a2 (* delayed ADD - result := a2; *)
          MUL(  a3 (* delayed MUL - result := a3; *)
          SUB   a4 (* result := a3 - a4; *)
          )        (* execute delayed MUL - result := a2 * (a3-a4); *)
          MUL   a5 (* result := a2 * (a3 - a4) * a5; *)
          )        (* execute delayed ADD *)
                   (* result := a1 + (a2 * (a3 - a4) * a5); *)
          ADD   a6 (* result := a1 + (a2 * (a3 - a4) * a5) + a6; *)
          ST    res (* store current result in variable res *)
```

Calling sub-programs or functions

A sub-program or a function (written in any of the IL, ST, LD, FBD or "C" language) is called from the IL language, using its name as an operator.

Operation executes a sub-program or a function - the value returned by the sub-program or function is stored into the IL current result
Allowed modifiers (none)
Operand The first calling parameter must be stored in the current result before the call. The following ones are expressed in the operand field, separated by comas.

Example:

```
(* Calling program : converts an analog value into a time value *)

Main:     LD    bi0
          SUBPRO bi1,bi2     (* call sub-program to get analog value *)
          ST    result       (* result := value returned by sub-program *)
          GT    vmax         (* test value overflow *)
          RETC               (* return if overflow *)
          LD    result
          MUL   1000         (* converts seconds in milliseconds *)
          TMR                (* converts to a timer *)
          ST    tmval        (* stores converted value in a timer *)

(* Called sub-program named 'SUBPRO' : evaluates the analog value *)
(* given as a binary value on three boolean inputs: in0, in1, in2 are the three boolean input
parameters of the sub-program *)
```

```
LD   in2
ANA               (* result = ana (in2); *)
MUL  2            (* result := 2*ana (in2); *)
ST   temporary    (* temporary := result *)
LD   in1
ANA
ADD  temporary    (* result := 2*ana (in2) + ana (in1); *)
MUL  2            (* result := 4*ana (in2) + 2*ana (in1); *)
ST   temporary    (* temporary := result *)
LD   in0
ANA
ADD  temporary    (* result := 4*ana (in2) + 2*ana (in1)+ana (in0); *)
ST   SUBPRO       (* return current result to calling program *)
```

- ***Calling function blocks: CAL operator***

Operation calls a function block

Allowed modifiers C N

Operand Name of the function block instance.

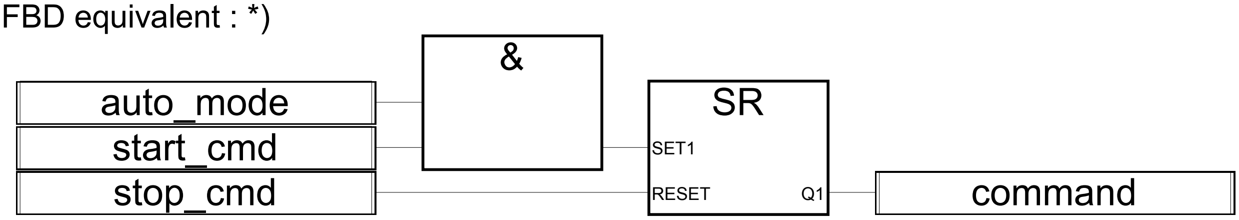
The input parameters of the blocks must be assigned before the call using LD/ST operations sequence.

Output parameters are known if used.

Example1:

```
(* Calling function block SR : SR1 is an instance of SR *)
LD  auto_mode
AND start_cmd
ST  SR1.set1
LD  stop_cmd
ST  SR1.reset
CAL SR1
LD  SR1.Q1
ST  command
```

(* FBD equivalent : *)



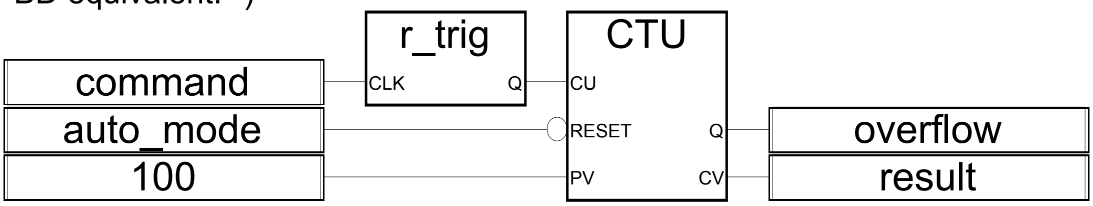
Example 2

(*We suppose R_TRIG1 is an instance of R_TRIG block and CTU1 is an instance of CTU block*)

```
LD  command
ST  R_TRIG1.clk
CAL R_TRIG1
LD  R_TRIG1.Q
ST  CTU1.cu
```

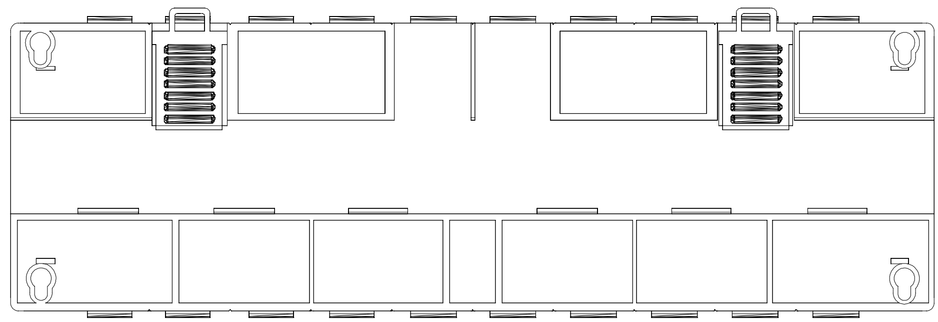
```
LDN     auto_mode
ST  CTU1.reset
LD  100
ST  CTU1.pv
CAL     CTU1
LD  CTU1.Q
ST  overflow
LD  CTU1.cv
ST  result
```

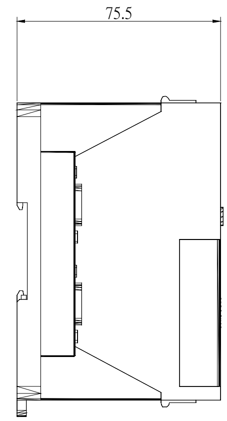
(* FBD equivalent: *)



Appendix F: Dimension

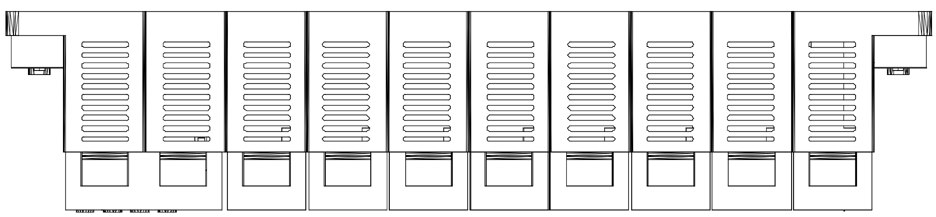
8 Slots :



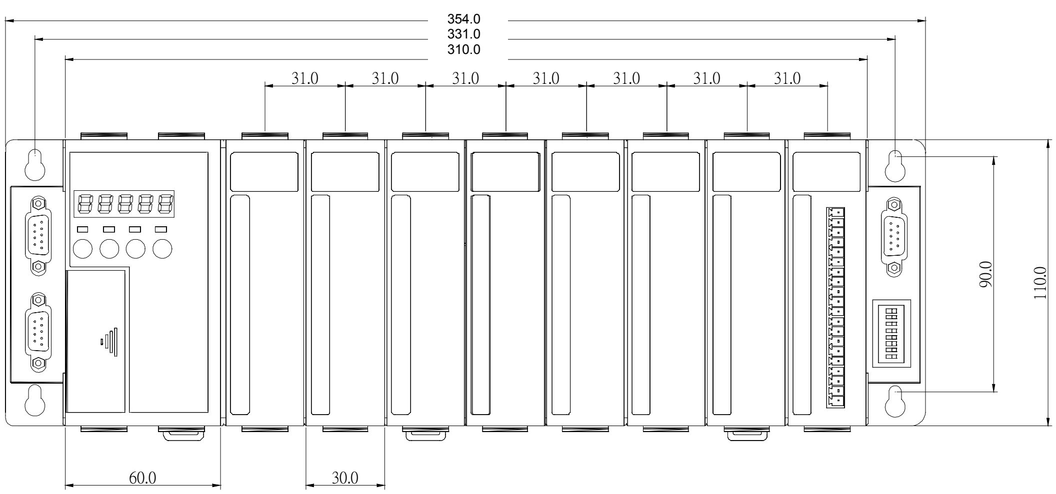


Back View

Unit : mm

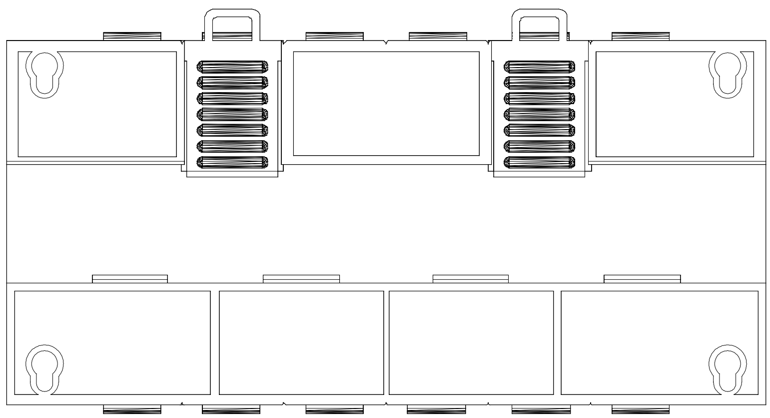


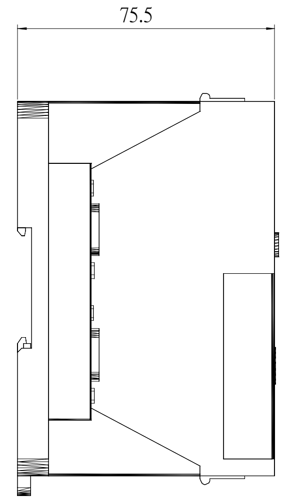
Top View



Front View

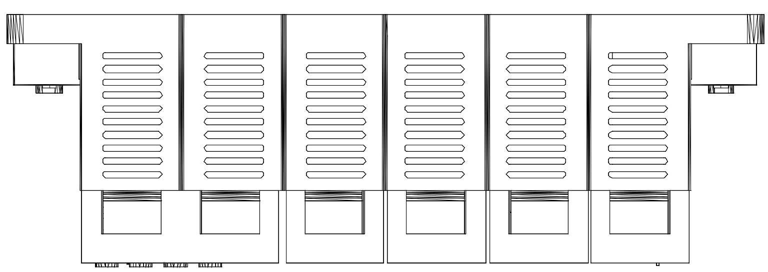
4 Slots :



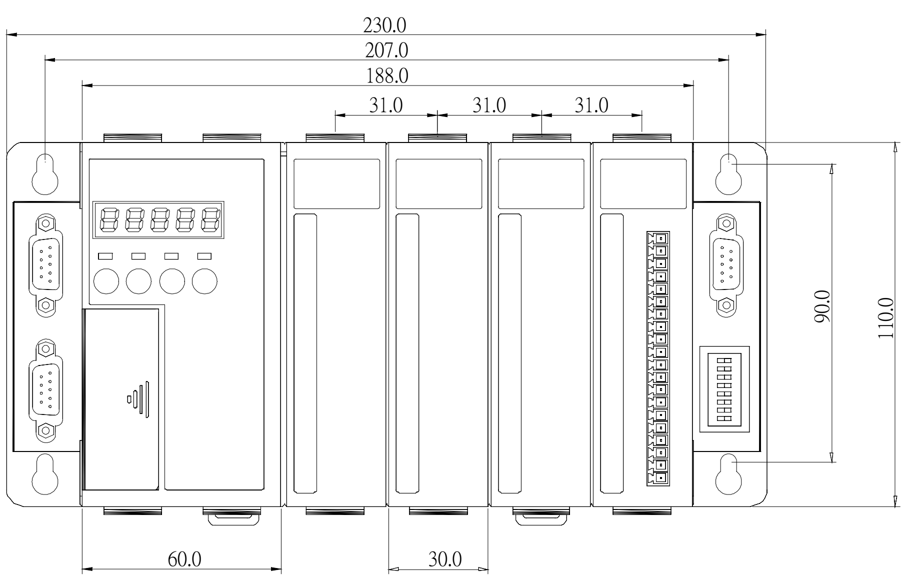


Back View

Unit : mm



Top View



Front View