EDAM-5000 Series
USB, Ethernet, RS232/485 Remote Module
User's manual

Web Site: www.inlog.com.tw

Trademark:

The names used in this manual for indentification only maybe registered trademarks of their respective companies

rev 1.4 January 11, 2012

Table of Contents

Chapter 1 Major Features

1.1 Multi-Interface DA&C I/O Modules

EDAM-5000 is based on the popular Ethernet/USB/RS485/RS232 networking standards used today in most business environments.

EDAM-5000 series provides:

1. 10/100 Mbps Ethernet interface and supports Modus/TCP protocol over TCP/IP for data connection.
2. USB 2.0 (high speed) interface and supports Modbus RTU /ASCII protocol for data connection.
3. ES485/232C interface and supports Modbus RTU /ASCII protocol for data connection.

With built-in Real Time OS (RTOS), The EDAM-5000 modules can connect to all communication interface simutimiouslly

1.2 Intelligent I/O Modules

Enhancing from traditional I/O modules, EDAM-5000 I/O modules have pre-built intelligent mathematic functions to empower the system capacity. The Digital Input modules provide Counter, Totalizer functions; the Digital Output modules provide pulse output, delay output functions; the Analog Input modules provide the Max./Min./Average data calculation; the Analog Output modules provide the PID loop control function.

1.3 Mixed I/O in One Module to fit all applications

EDAM-5000 mixed I/O module design concept provides the most cost-effective I/O usage for application system. The most common used I/O type for single function unit are collected in ONE module. This design concept not only save I/O usage and spare modules cost but also speed up I/O relative operations. For small DA&C system or standalone control unit in a middle or large scale, EDAM-5000 mixed I/O design can easily fit application needs by one or two modules only. With additional embedded control modules, EDAM-5000 can easily create a localized, less complex, and more distributed I/O architecture.

1.4 Modbus/TCP and RTU protocol supported for open connectivity

EDAM-5000 modules support the popular industrial standard, Modbus/TCP and RTU protocol, to connect with Ethernet Controller or HMI/SCADA software built with Modbus/TCP or RTU driver.

1.5 Software Support

Based on the Modbus/TCP and RTU standard, the EDAM-5000 firmware is a built-in Modbus/TCP and RTU server. Therefore, Inlog provides the necessary DLL drivers and Windows Utility for users for client data for the EDAM-5000. Users can configure this DA&C system via Windows Utility; integrate with HMI software package via Modbus/TCP driver or Modbus/TCP OPC Server. Even more, you can use the DLL driver and ActiveX to develop your own applications.

1.6 Common technical specification of EDAM-5000

- **Ethernet:** 10 BASE-T IEEE 802.3 100 BASE-TX IEEE 802.3u

  Wiring: UTP, category 5 or greater

  Bus Connection: RJ45 modular jack

  Comm. Protocol: Modbus/TCP on TCP/IP and RTU on UDP/IP or RS485, ASCII commands

  Data Transfer Rate: Up to 100 Mbps

- **USB:** USB 2.0

  Wiring: USB cable

  Bus Connection: USB type B connector

  Comm. Protocol: RTU, ASCII commands

  Data Transfer Rate: high speed

- **RS485/232C:**

  Wiring: Twist pair for RS485 or Three wires cable for RS232C

  Bus Connection: 2/3 pin terminals

  Comm. Protocol: RTU, ASCII commands

  Data Transfer Rate: 2400,4800,9600,19200,38400,57600,115200

- **Power :**

  USB powered (if USB connection)

  External power with unregulated 10 to 30VDC

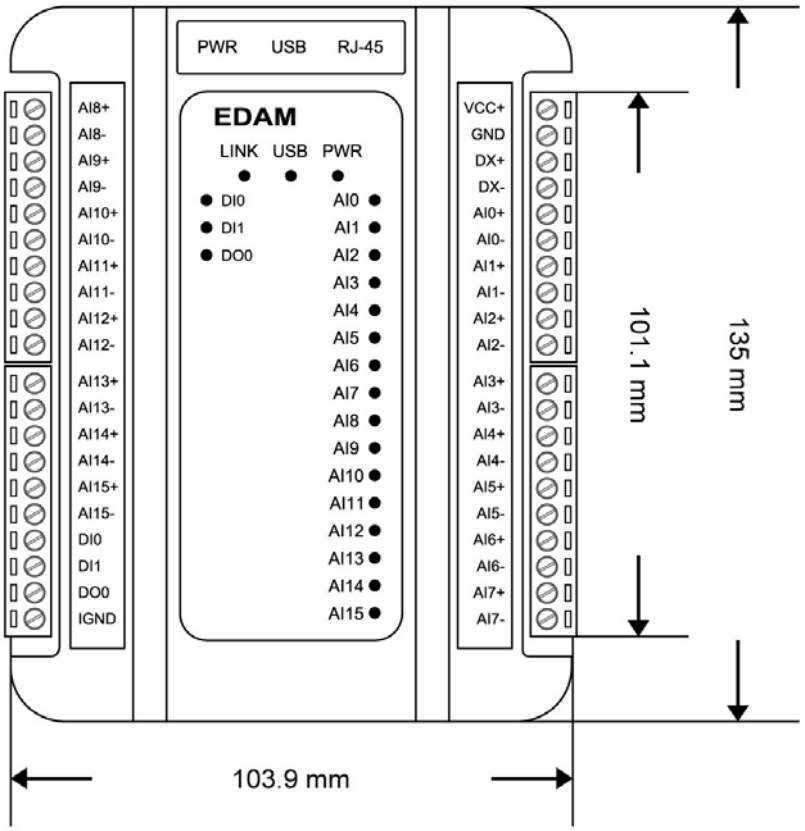
  Over-voltage protection and power reversal

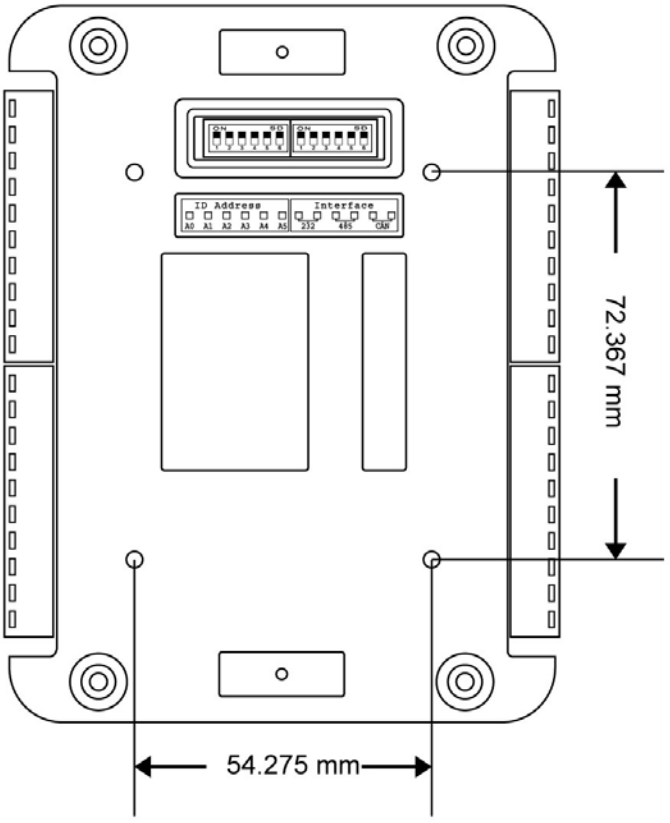
- **I/O Module input isolation:** 3000 V DC
- **Status Indicator:** Power, Communication (Ethernet,USB,RS485/232)
- **Case:** ABS with captive mounting hardware
- **Plug-in Screw Terminal Block:** Accepts 0.5 mm 2 to 2.5 mm 2 , 1 - #12 or 2 - #14 to #22 AWG
- **Operating Temperature:** - 10 to 70º C (14 to 158º F)
- **Storage Temperature:** - 25 to 85º C (-13 to 185º F)
- **Humidity:** 5 to 95%, non-condensing
- **Atmosphere:** No corrosive gases

NOTE: Equipment will operate below 30% humidity. However, static electricity problems occur much more frequently at lower humidity levels. Make sure you take adequate precautions when you touch the equipment. Consider using ground straps, anti-static floor coverings, etc. if you use the equipment in low humidity environments.

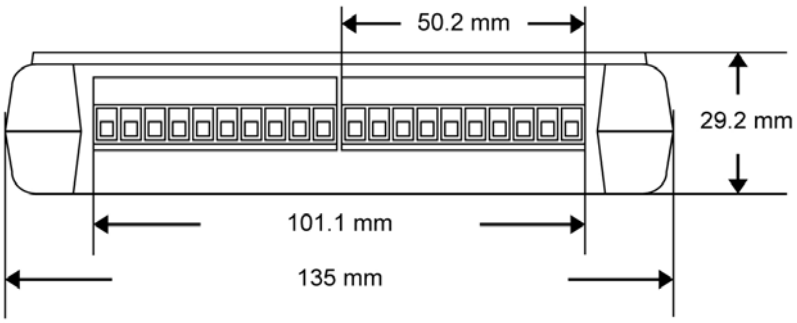
1.7 Dimensions

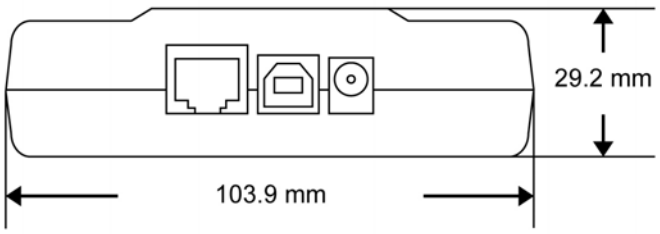
The following diagrams show the dimensions of the EDAM-5000 I/O module in millimeters.



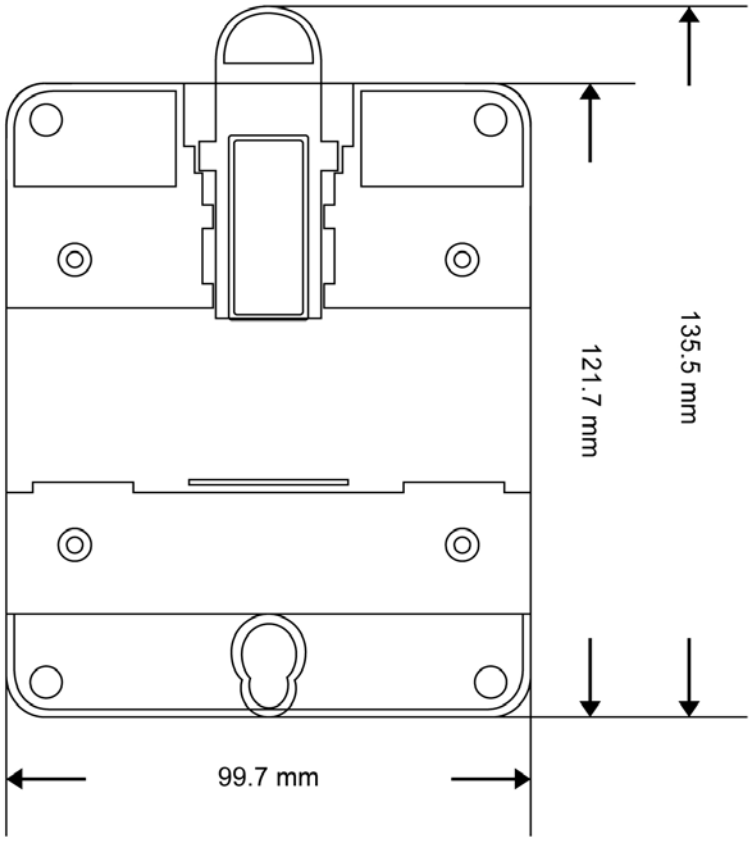


.


50.2 mm
29.2 mm
101.1 mm
135 mm


29.2 mm
103.9 mm

.


135.5 mm
121.7 mm
99.7 mm

1.8 System Requirements

- IBM PC compatible computer with 486 CPU (Pentium is recommended)
- Microsoft 95/98/2000/NT 4.0 (SP3 or SP4)/XP or higher versions
- At least 32 MB RAM
- 20 MB of hard disk space available
- VGA color monitor
- 2x or higher speed CD-ROM
- Mouse or other pointing devices
- 10 or 100 Mbps Ethernet Card
- 10 or 100 Mbps Ethernet Hub (at least 2 ports)
- USB 2.0 hub with output current at last 400mA(if powered by USB hub)
- Two Ethernet Cable with RJ-45 connector
- Power supply for EDAM-5000 (+10 to +30 V unregulated), if no USB connection
-

1.9 I/O modules wiring

The system uses a plug-in screw terminal block for the interface between I/O modules and field devices. The following information must be considered when connecting electrical devices to I/O modules.

- The terminal block accepts wires from 0.5 mm to 2.5 mm.
- Always use a continuous length of wire. Do not combine wires to make them longer.
- Use the shortest possible wire length.
- Use wire trays for routing where possible.
- Avoid running wires near high-energy wiring.
- Avoid running input wiring in close proximity to output wiring where possible.
- Avoid creating sharp bends in the wires.
-

Chapter 2 Specifications

2.1 EDAM-5015 specifications

The EDAM-5015 is a 16-bit, 12-channel RTD input module that provides programmable input ranges on all channels. It accepts Various RTD inputs (Type PT100, PT1000, Balco 500, NI604, NI1000) and provides data to the host computer.

Analog Input:

- Effective resolution: 16-bit
- Channels: 12
- Input type: PT100, PT1000, Balco 500, NI RTD
- Input range:

  PT100 type: -50 ~ 150°C/0 ~ 100°C, 0 ~ 200°C, 0 ~ 400°C, -200 ~ 200°C

  Pt1000 type: -40 ~ 160°C

  Balco 500 type: -30 ~ 120°C

  Ni604 type: -80 ~ 100°C

  Ni1000 type: -0 ~ 100°C
- Sampling rate: 10 samples/sec
- Input impedance: 10 MΩ
- Accuracy: ±0.15% or better
- Zero drift: ±20 μV/ °C
- Span drift: 25 ppm/ °C

Built-in Watchdog Timer

Power requirements: USB powered (400mA max.) or external unregulated +10 ~ +30 VDC

Power consumption: 1.5 W/Typical, 2W/max

2.2 EDAM-5017 specifications

The EDAM-5017 is a 16-bit, 16-channel Analog input module that provides programmable input ranges on all channels.

Analog Input:

- Effective resolution: 16-bit
- Channels: 16
- Input type: Voltage, Current
- Input range: ±10V, ±5V, ±2.5V, ±1V, ±500mV, ±15mV, 0~20mA, 4~20mA
- Sampling rate: 10 samples/sec.
- Input impedance: 10 MΩ
- Accuracy: ±0.15% or better
- Zero drift: ±20 μV/ °C
- Span drift: 25 ppm/ °C

Digital Input:

- Input Channel: 2 channels
- Input Type:, Voltage (logic 0 for 0<Vin < 3Vdc , logic 1 for 5V<Vin < 24Vdc) or Switch On/Off
- Isolation voltage: 2000 V

Digital Output:

- Output Channel: 1 channel
- Output Type: Open Collect to 30Vdc/3A(max)
- Isolation voltage: 2000 V

Built-in Watchdog Timer

Power requirements: USB powered (400mA max.) or external unregulated +10 ~ +30 VDC

Power consumption: 1.5 W/Typical, 2W/max

2.3 EDAM-5019 specifications

The EDAM-5019 is a 16-bit, 16-channel Thermocouple input module that provides programmable input ranges on all channels. It accepts Various Thermocouple inputs (Type J, K, T, E, R, S, B) and provides data to the host computer in engineering units (ºC). In order to satisfy various temperature requirements in one module, each analog channel is allowed to configure an individual range for several applications.

Analog Input:

- Effective resolution: 16-bit
- Channels: 16
- Input type: J, K, T, E, R, S, B
- Input range:

  J type: 0 ~ 760 o C

  K type: 0 ~ 1370 o C

  T type:-100 ~ 400 o C

  E type: 0 ~ 1000 o C

  R type:500 ~ 1750 o C

  S type:500 ~ 1750 o C

  B type:500 ~ 1800 o C
- Sampling rate: 10 samples/sec, 20 samples/sec, 50 samples/sec.
- Input impedance: 10 MΩ
- Accuracy: ±0.15% or better
- Zero drift: ±6 μV/° C
- Span drift: ±25 ppm/° C

Digital Input:

- Input Channel: 2 channels
- Input Type:, Voltage (logic 0 for 0<Vin < 3Vdc , logic 1 for 5V<Vin < 24Vdc) or Switch On/Off
- Isolation voltage: 2000 VDC

Digital Output:

- Output Channel: 1 channels
- Output Type: Open Collect to 30Vdc/3A(max)
- Isolation voltage: 2000 VDC

Built-in Watchdog Timer

Power requirements: USB powered (400mA max.) or external unregulated +10 ~ +30 VDC

Power consumption: 1.5 W/Typical, 2W/max

2.4 EDAM-5028 specifications

The EDAM-5028 is a 8-channels MOSEFT output and 24-channels input module that provides programmable I/O ranges on all channels. It accepts Various Digital inputs/MOSFET outputs and provides data to the host computer.

Digital Input:

- Channels: 24 channels
- Input type: Voltage (logic 0 for 3Vdc maximum, logic 1 for 5Vdc minimums) or Switch On/Off
- Isolation voltage: 2000 V

Digital Output:

- Output Channel: 8 channels
- Output Type: Source Output up to 30Vdc/3A(max)
- Isolation voltage: 2000 V

Built-in Watchdog Timer

Power requirements: USB powered (400mA max.) or external unregulated +10 ~ +30 VDC

Power consumption: 1.5 W/Typical, 2W/max

2.5 EDAM-5029 specifications

The EDAM-5029 is a 16-channels MOSEFT output and 16-channels digital input module that provides programmable I/O ranges on all channels. It accepts Various Digital inputs/MOSFET outputs and provides data to the host computer.

Digital Input:

- Channels: 16 channels
- Input type: Voltage (logic 0 for 3Vdc maximum, logic 1 for 5Vdc minimums) or Switch On/Off
- Isolation voltage: 2000 V

Digital Output:

- Output Channel: 16 channels
- Output Type: Source Output up to 30Vdc/3A(max)
- Isolation voltage: 2000 V

Built-in Watchdog Timer

Power requirements: USB powered (400mA max.) or external unregulated +10 ~ +30 VDC

Power consumption: 1.5 W/Typical, 2W/max

2.6 EDAM-5060 specifications

The EDAM-5060 is a 10-channels Relay and 12-channels digital input module that provides programmable I/O ranges on all channels. It accepts Various Digital inputs/Relay outputs and provides data to the host computer.

Digital Input:

- Channels: 12 channels
- Input type: Voltage (logic 0 for 3Vdc maximum, logic 1 for 5Vdc minimums) or Switch On/Off
- Isolation voltage: 2000 V

Relay Output:

- Relay Channel: 10 Relay output
- Relay Type: Form-A (DPDT)
- Contact rating: AC 3A/125V, DC 3A/30V,. 3A/110V
- Breakdown voltage: OPEN contacts: 1000VAC
  Contacts and coil: 1000VAC
- FCC Surge Voltage: Contacts and coil:1500V
- Insulation resistance: 100M ohm (at 500VDC)
- Operate time: 6ms
- Release time: 4ms
- Min. operations: 500000 times(At 1A/30VDC)

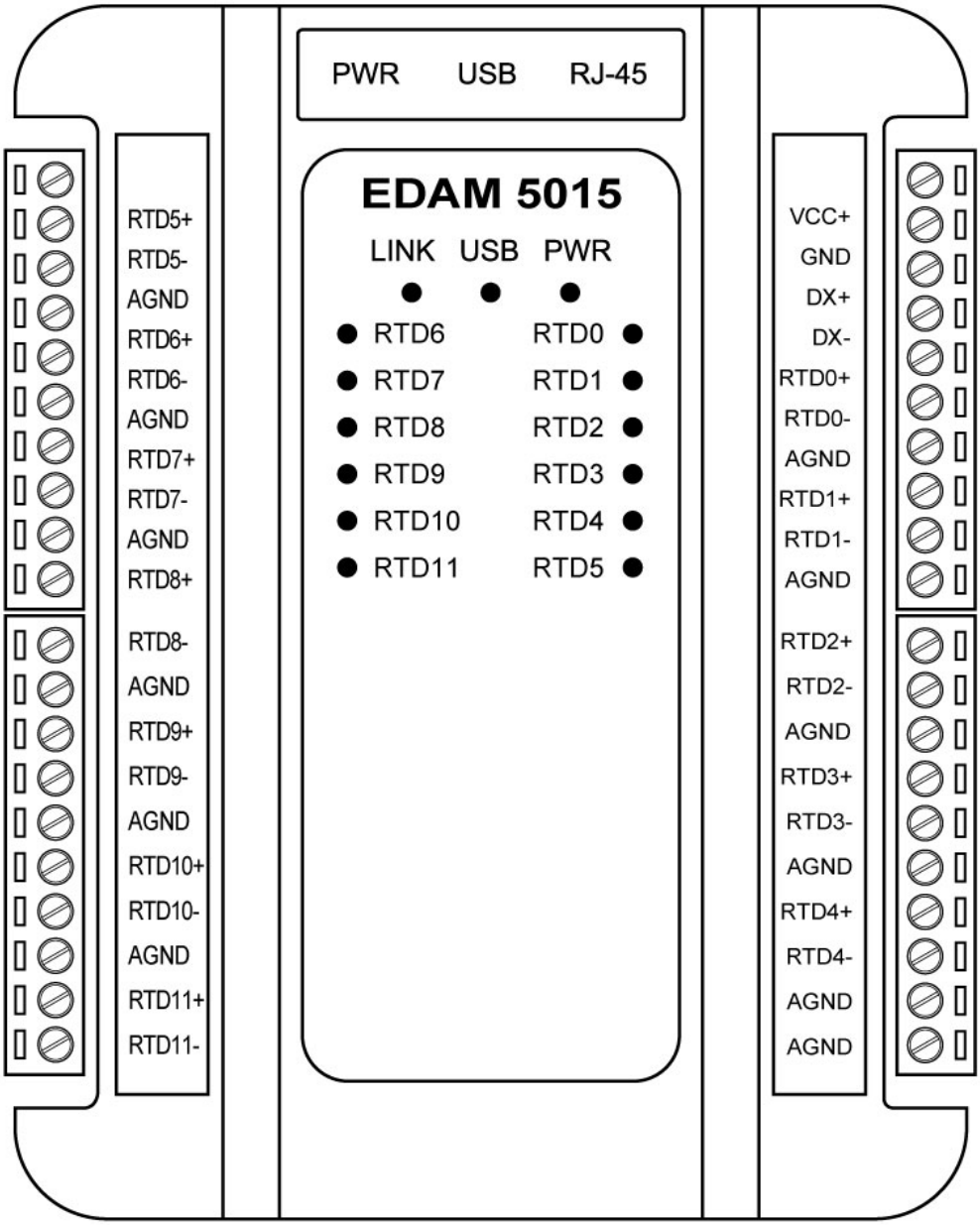
Built-in Watchdog Timer

Power requirements: USB powered (400mA max.) or external unregulated +10 ~ +30 VDC

Power consumption: 1.5 W/Typical, 2W/max

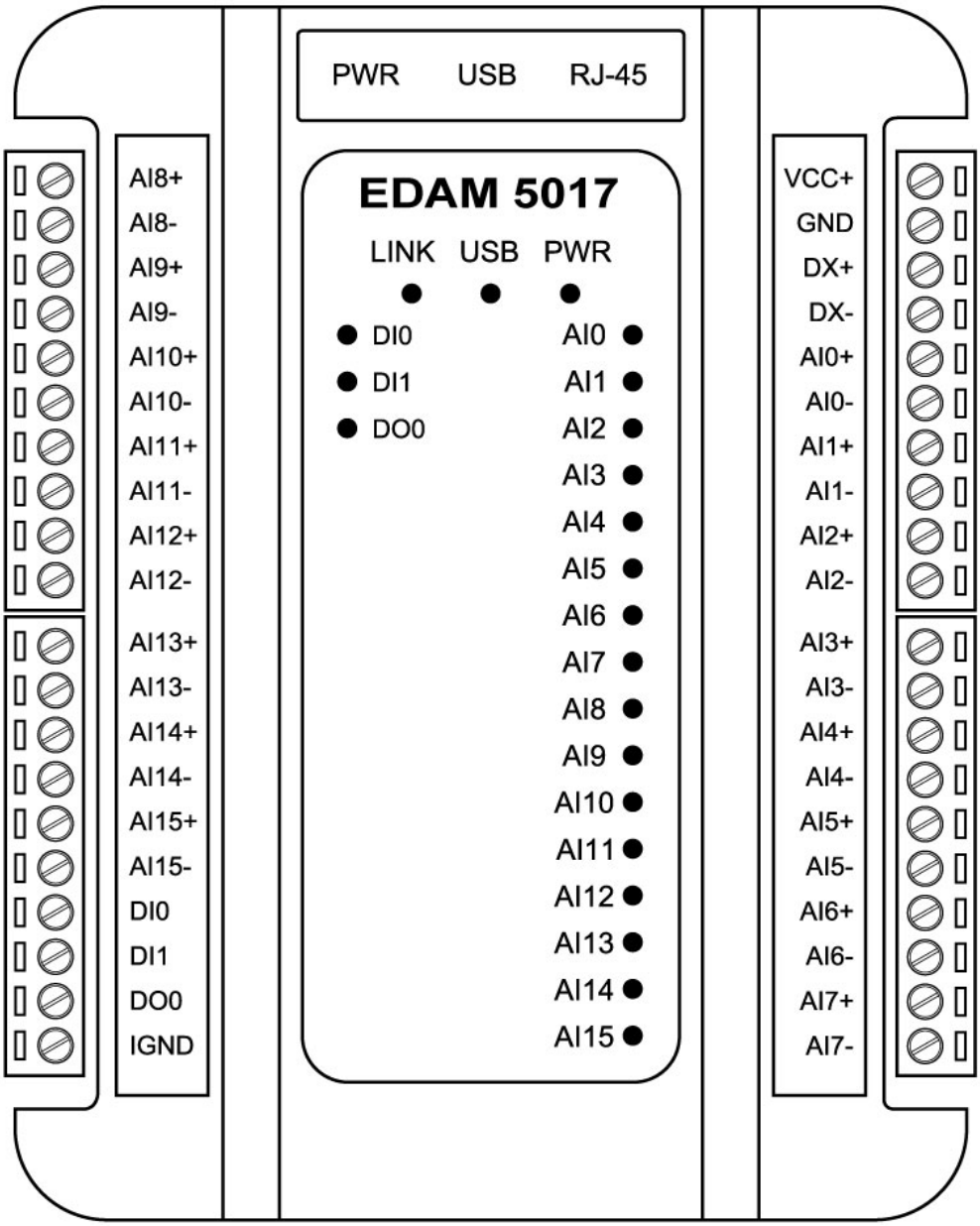
Chapter 3 Connector/pin assignment

3.1 EDAM-5015 Front side connectors



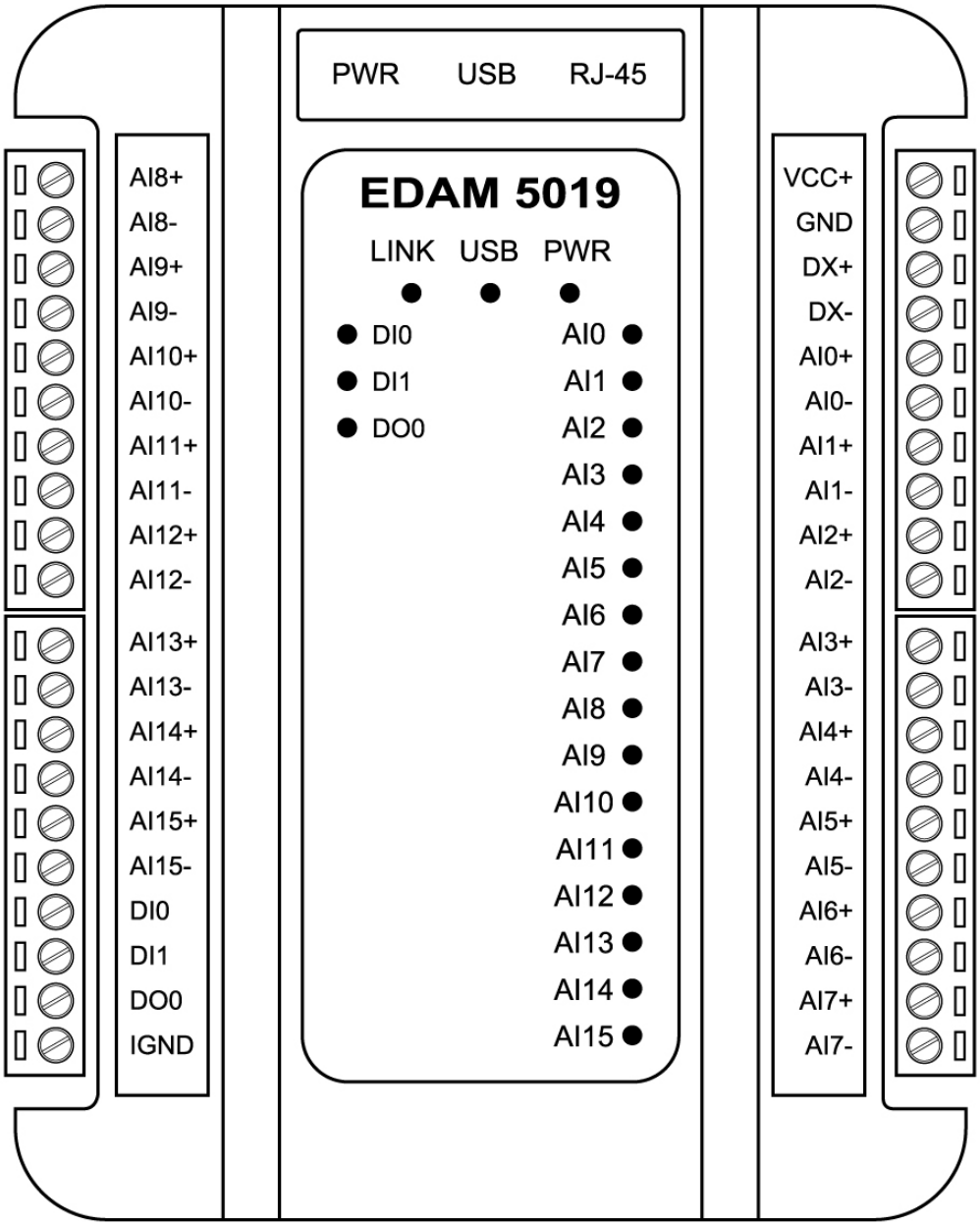
Connector	Description
RJ-45	Ethernet Connector
PWR	External power connector
VCC+	External power 10<Vdc<30
GND	Power Ground
DX+	Data+ (for RS-485) , TX (for RS-232C)
DX-	Data- (for RS-485) , RX (for RS-232C)
RTD0+, RTD0-	RTD input channel 0
AGND	RTD common 0
RTD1+, RTD1-	RTD input channel 1
AGND	RTD common 1
RTD2+, RTD2-	RTD input channel 2
AGND	RTD common 2
RTD3+, RTD3-	RTD input channel 3
AGND	RTD common 3
RTD4+, RTD4-	RTD input channel 4
AGND	RTD common 4
RTD5+, RTD5-	RTD input channel 5
AGND	RTD common 5
RTD6+, RTD6-	RTD input channel 6
AGND	RTD common 6
RTD7+, RTD7-	RTD input channel 7
AGND	RTD common 7
RTD8+, RTD8-	RTD input channel 8
AGND	RTD common 8
RTD9+, RTD9-	RTD input channel 9
AGND	RTD common 9
RTD10+, RTD10-	RTD input channel 10
AGND	RTD common 10/11
RTD11+, RTD11-	RTD common 11

3.2 EDAM-5017 Front side connectors



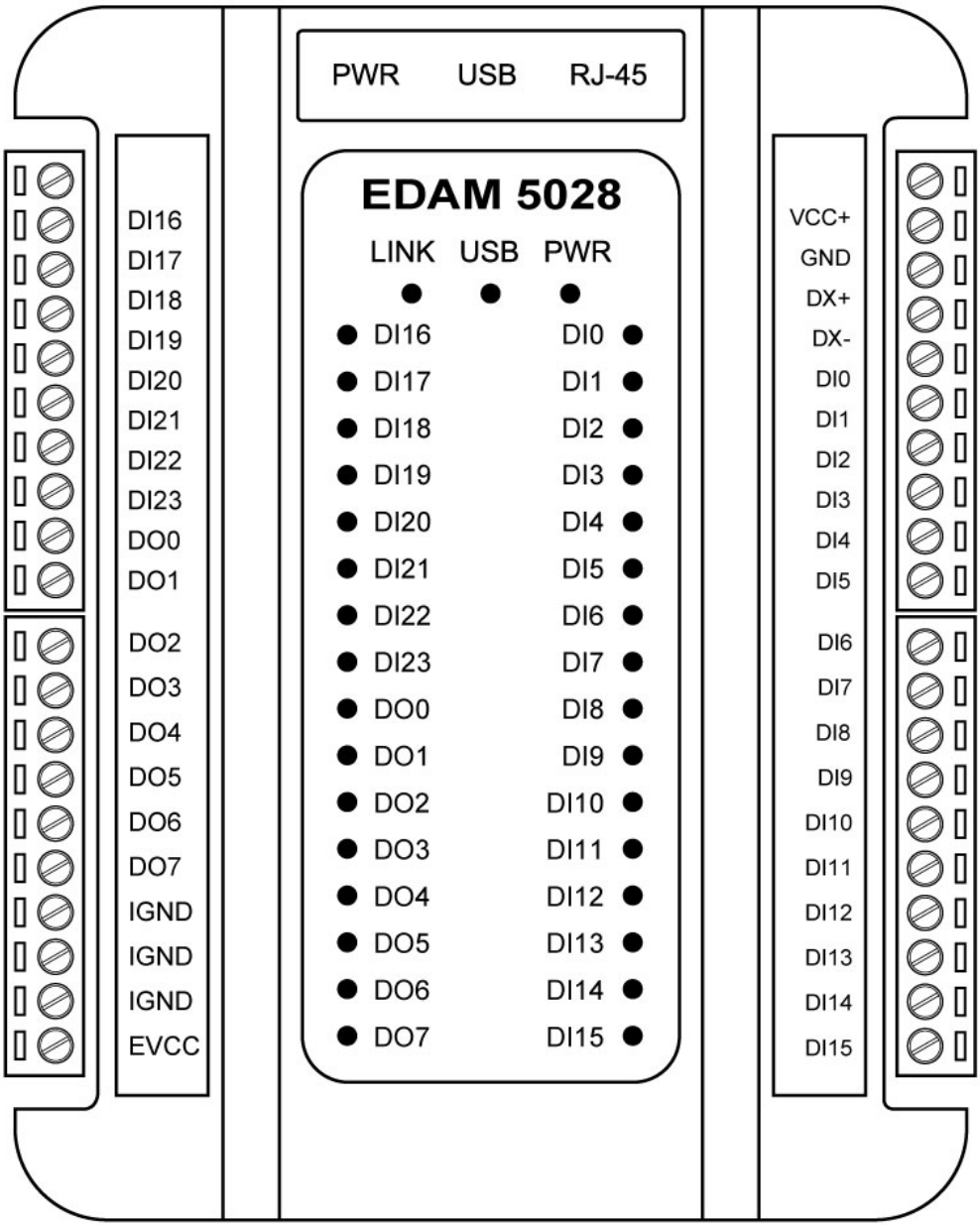
Connector	Description
RJ-45	Ethernet Connector
PWR	External power connector
VCC+	External power 10<Vdc<30
GND	Power Ground
DX+	Data+ (for RS-485) , TX (for RS-232C)
DX-	Data- (for RS-485) , RX (for RS-232C)
AI0+,AI0-	Analog Input channel 0
AI1+,AI1-	Analog Input channel 10
AI2+,AI2-	Analog Input channel 2
AI3+,AI3-	Analog Input channel 3
AI4+,AI4-	Analog Input channel 4
AI5+,AI5-	Analog Input channel 5
AI6+,AI6-	Analog Input channel 6
AI7+,AI7-	Analog Input channel 7
AI8+,AI8-	Analog Input channel 8
AI9+,AI9-	Analog Input channel 9
AI10+,AI10-	Analog Input channel 10
AI11+,AI11-	Analog Input channel 11
AI12+,AI12-	Analog Input channel 12
AI13+,AI13-	Analog Input channel 13
AI14+,AI14-	Analog Input channel 14
AI15+,AI15-	Analog Input channel 15
DI0	Digital Input channel 0
DI1	Digital Input channel 1
DO0	Digital Output channel 0
IGND	Isolated Digital GND

3.3 EDAM-5019 Front side connectors



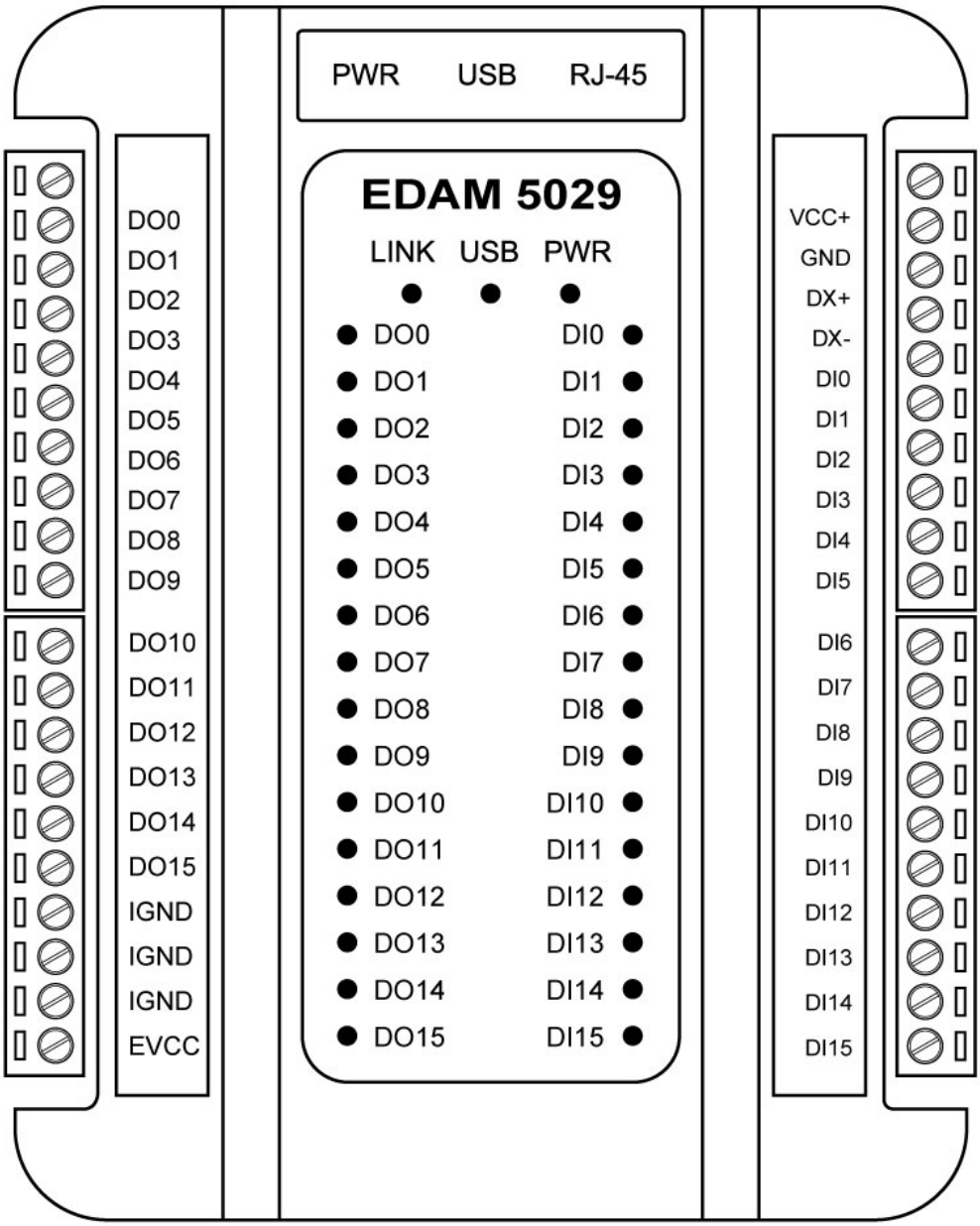
Connector	Description
RJ-45	Ethernet Connector
PWR	External power connector
VCC+	External power 10<Vdc<30
GND	Power Ground
DX+	Data+ (for RS-485) , TX (for RS-232C)
DX-	Data- (for RS-485) , RX (for RS-232C)
AI0+,AI0-	Analog Input channel 0
AI1+,AI1-	Analog Input channel 10
AI2+,AI2-	Analog Input channel 2
AI3+,AI3-	Analog Input channel 3
AI4+,AI4-	Analog Input channel 4
AI5+,AI5-	Analog Input channel 5
AI6+,AI6-	Analog Input channel 6
AI7+,AI7-	Analog Input channel 7
AI8+,AI8-	Analog Input channel 8
AI9+,AI9-	Analog Input channel 9
AI10+,AI10-	Analog Input channel 10
AI11+,AI11-	Analog Input channel 11
AI12+,AI12-	Analog Input channel 12
AI13+,AI13-	Analog Input channel 13
AI14+,AI14-	Analog Input channel 14
AI15+,AI15-	Analog Input channel 15
DI0	Digital Input channel 0
DI1	Digital Input channel 1
DO0	Digital Output channel 0
IGND	Isolated Digital GND

3.4 EDAM-5028 Front side connectors



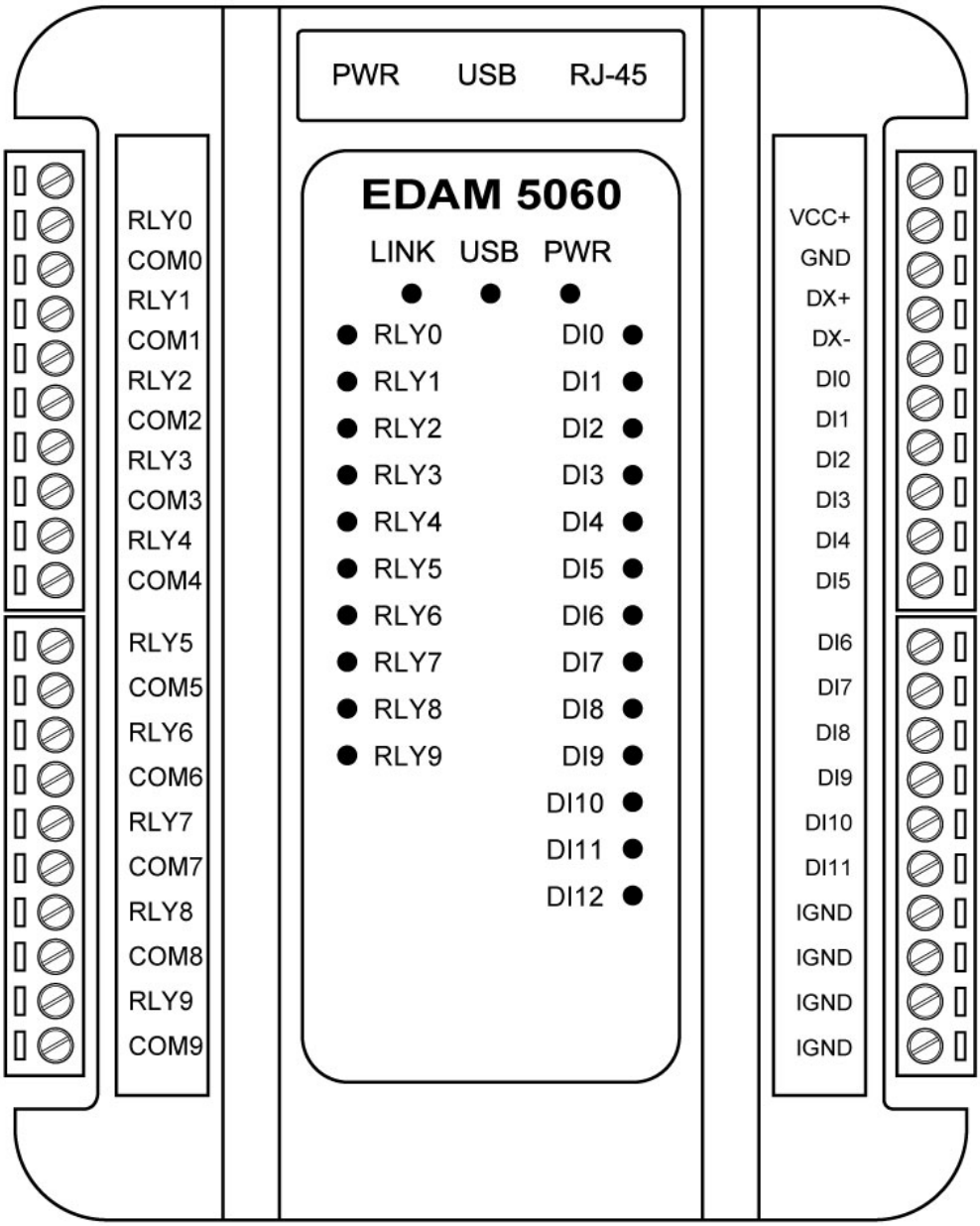
Connector	Description
RJ-45	Ethernet Connector
PWR	External power connector
VCC+	External power 10<Vdc<30
GND	Power Ground
DX+	Data+ (for RS-485) , TX (for RS-232C)
DX-	Data- (for RS-485) , RX (for RS-232C)
DI0	Digital Input channel 0
DI1	Digital Input channel 1
DI2	Digital Input channel 2
DI3	Digital Input channel 3
DI4	Digital Input channel 4
DI5	Digital Input channel 5
DI6	Digital Input channel 6
DI7	Digital Input channel 7
DI8	Digital Input channel 8
DI9	Digital Input channel 9
DI10	Digital Input channel 10
DI11	Digital Input channel 11
DI12	Digital Input channel 12
DI13	Digital Input channel 13
DI14	Digital Input channel 14
DI15	Digital Input channel 15
DI16	Digital Input channel 16
DI17	Digital Input channel 17
DI18	Digital Input channel 18
DI19	Digital Input channel 19
DI20	Digital Input channel 20
DI21	Digital Input channel 21
DI22	Digital Input channel 22
DI23	Digital Input channel 23
DO0	Digital Output channel 0
DO1	Digital Output channel 1
DO2	Digital Output channel 2
DO3	Digital Output channel 3
DO4	Digital Output channel 4
DO5	Digital Output channel 5
DO6	Digital Output channel 6
DO7	Digital Output channel 7
IGND	Isolated Digital GND
IGND	Isolated Digital GND
IGND	Isolated Digital GND
EVCC	External DO voltage input (see 4.4.3)

3.5 EDAM-5029 Front side connectors



Connector	Description
RJ-45	Ethernet Connector
PWR	External power connector
VCC+	External power 10<Vdc<30
GND	Power Ground
DX+	Data+ (for RS-485) , TX (for RS-232C)
DX-	Data- (for RS-485) , RX (for RS-232C)
DI0	Digital Input channel 0
DI1	Digital Input channel 1
DI2	Digital Input channel 2
DI3	Digital Input channel 3
DI4	Digital Input channel 4
DI5	Digital Input channel 5
DI6	Digital Input channel 6
DI7	Digital Input channel 7
DI8	Digital Input channel 8
DI9	Digital Input channel 9
DI10	Digital Input channel 10
DI11	Digital Input channel 11
DI12	Digital Input channel 12
DI13	Digital Input channel 13
DI14	Digital Input channel 14
DI15	Digital Input channel 15
DO0	Digital Output channel 0
DO1	Digital Output channel 1
DO2	Digital Output channel 2
DO3	Digital Output channel 3
DO4	Digital Output channel 4
DO5	Digital Output channel 5
DO6	Digital Output channel 6
DO7	Digital Output channel 7
DO8	Digital Output channel 8
DO9	Digital Output channel 9
DO10	Digital Output channel 10
DO11	Digital Output channel 11
DO12	Digital Output channel 12
DO13	Digital Output channel 13
DO14	Digital Output channel 14
DO15	Digital Output channel 15
IGND	Isolated Digital GND
IGND	Isolated Digital GND
IGND	Isolated Digital GND
EVCC	External DO voltage input (see 4.5.3)

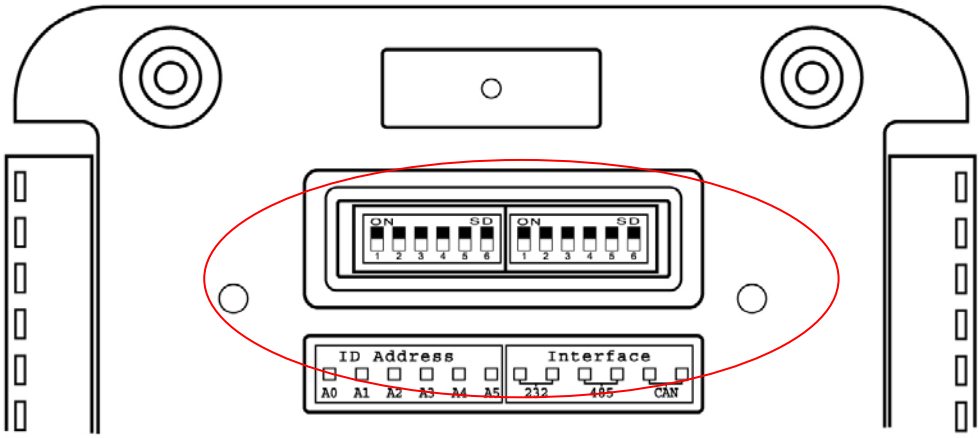
3.6 EDAM-5060 Front side connectors



Connector	Description
RJ-45	Ethernet Connector
PWR	External power connector
VCC+	External power 10<Vdc<30
GND	Power Ground
DX+	Data+ (for RS-485) , TX (for RS-232C)
DX-	Data- (for RS-485) , RX (for RS-232C)
AI0+,AI0-	Analog Input channel 0
AI1+,AI1-	Analog Input channel 10
AI2+,AI2-	Analog Input channel 2
AI3+,AI3-	Analog Input channel 3
AI4+,AI4-	Analog Input channel 4
AI5+,AI5-	Analog Input channel 5
AI6+,AI6-	Analog Input channel 6
AI7+,AI7-	Analog Input channel 7
AI8+,AI8-	Analog Input channel 8
AI9+,AI9-	Analog Input channel 9
AI10+,AI10-	Analog Input channel 10
AI11+,AI11-	Analog Input channel 11
AI12+,AI12-	Analog Input channel 12
AI13+,AI13-	Analog Input channel 13
AI14+,AI14-	Analog Input channel 14
AI15+,AI15-	Analog Input channel 15
DI0	Digital Input channel 0
DI1	Digital Input channel 1
DO0	Digital Output channel 0
IGND	Isolated Digital GND

3.7 EDAM-5000 Rear side connectors

There are two 6-pin DIP switches labeled as ***ID address*** and ***Interface***.



- **ID address switch is used for setting ID address of module**

"**ON**" =logic 1, "**OFF**"=logic 0

Where

A0=bit 0 of ID address

A1=bit 1 of ID address

A2=bit 2 of ID address

A3=bit 3 of ID address

A4=bit 4 of ID address

A5=bit 5 of ID address

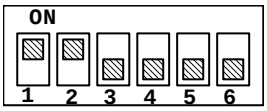
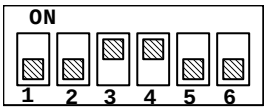
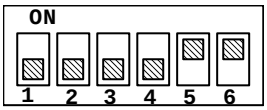
Example1: Assume A0="ON", A1,A2,A3="OFF" and A4,A5="ON"

Then the ID address= A5-A4-A3-A2-A1-A0=110001=31(hex)=49(dec)

Example2: Assume A0="OFF", A1,A2,A3="ON" and A4,A5="OFF"

Then the ID address= A5-A4-A3-A2-A1-A0=001110=0E(hex)=14(dec)

- **Interface switch is used for select serial communication interface (RS485 ,RS232 ,or CAN bus)**

Switch pin setting	DX+	DX-	GND
	TX signal of RS232	RX signal of RS232	GND of RS232
	Data+ of RS485	Data- of RS485	
	DX+ of CAN BUS	DX- of CAN BUS	

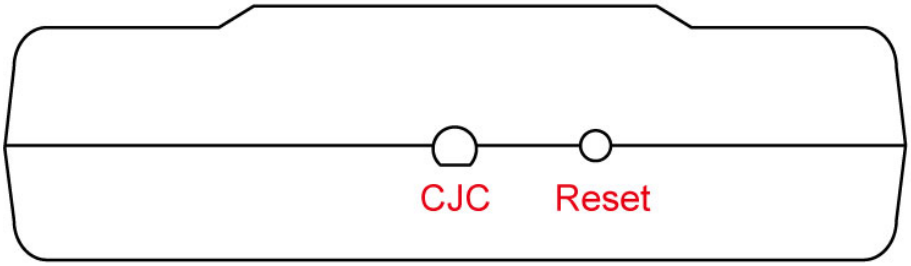
Note: CAN BUS function (option)

3.8 EDAM-5000 reset switch and CJC sensor

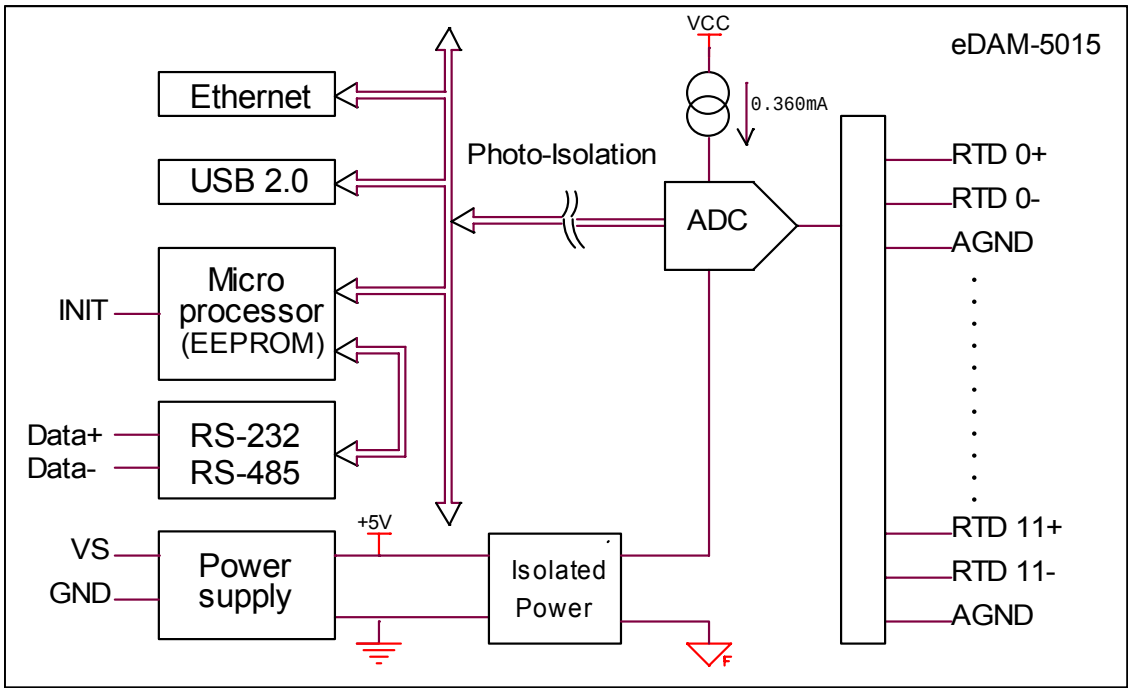
There are a reset switch (Reset) and tempearature sensor (CJC) on the botton side of the module

The reset switch is available for all module.The users could push this switch to reboot the module

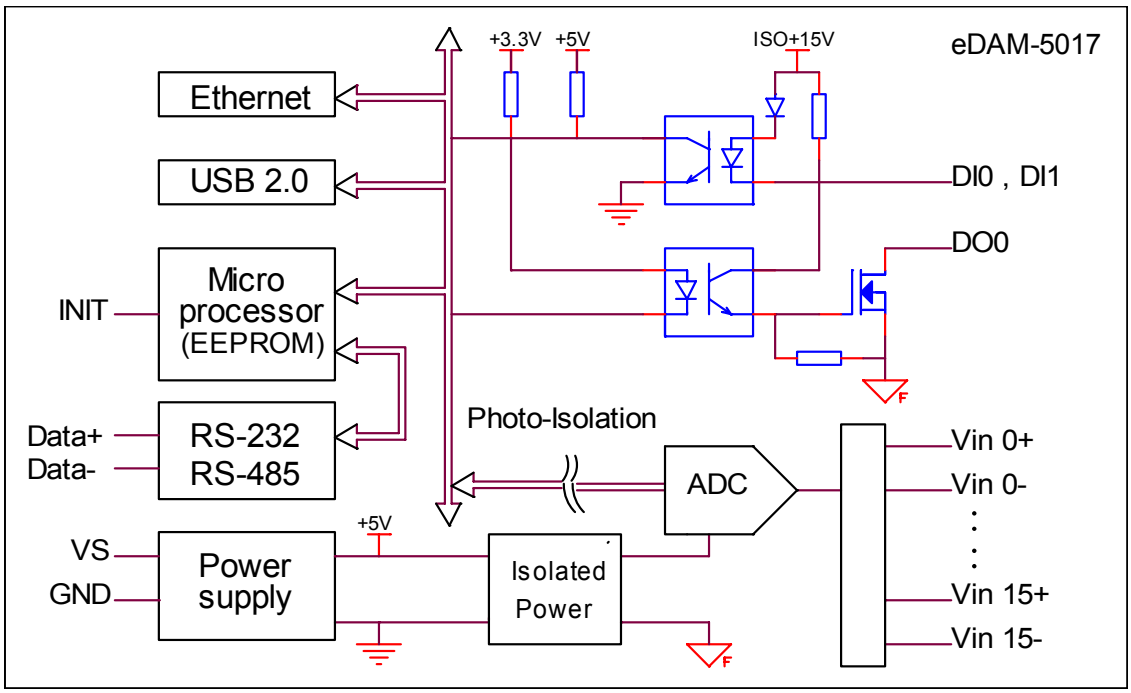
The Tempearature sensor (cold junction compensation CJC) is avalibale for EDAM5019 only



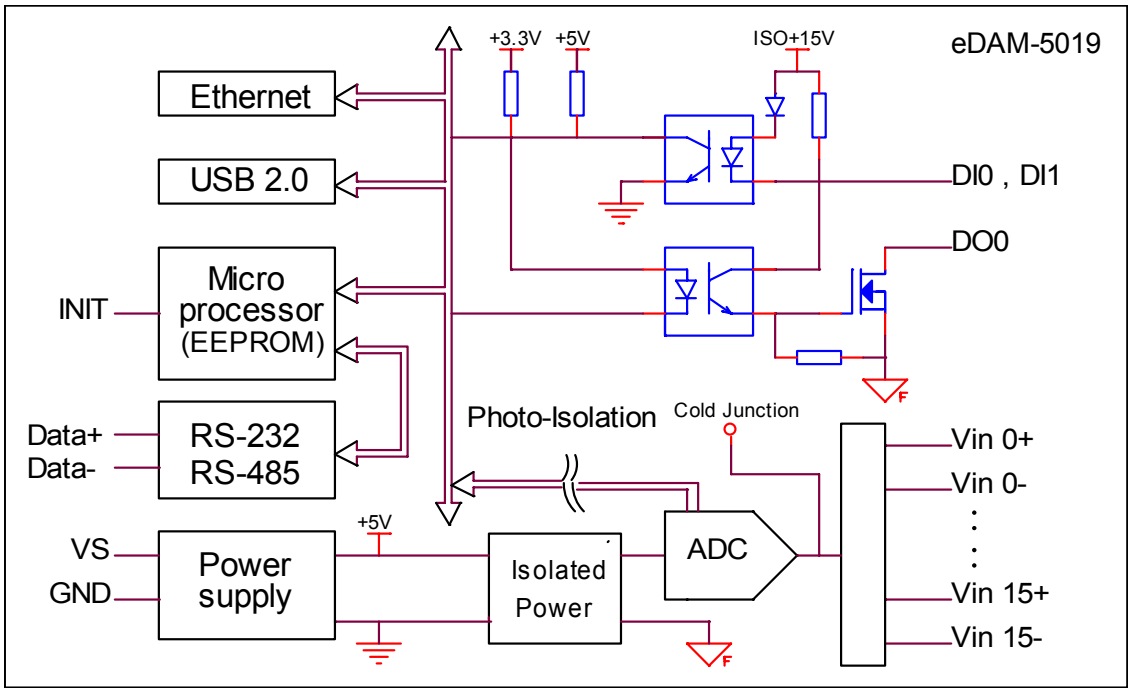
3.9 EDAM-5015 Analog/Digital I/O block diagram



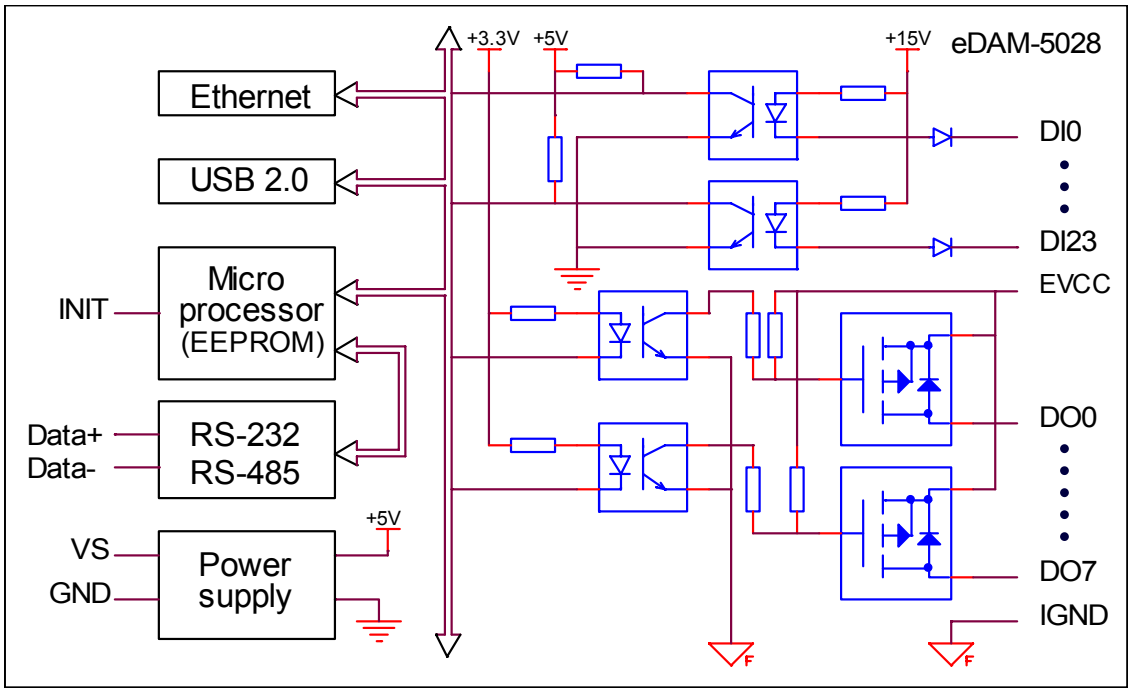
3.10 EDAM-5017 Analog/Digital I/O block diagram



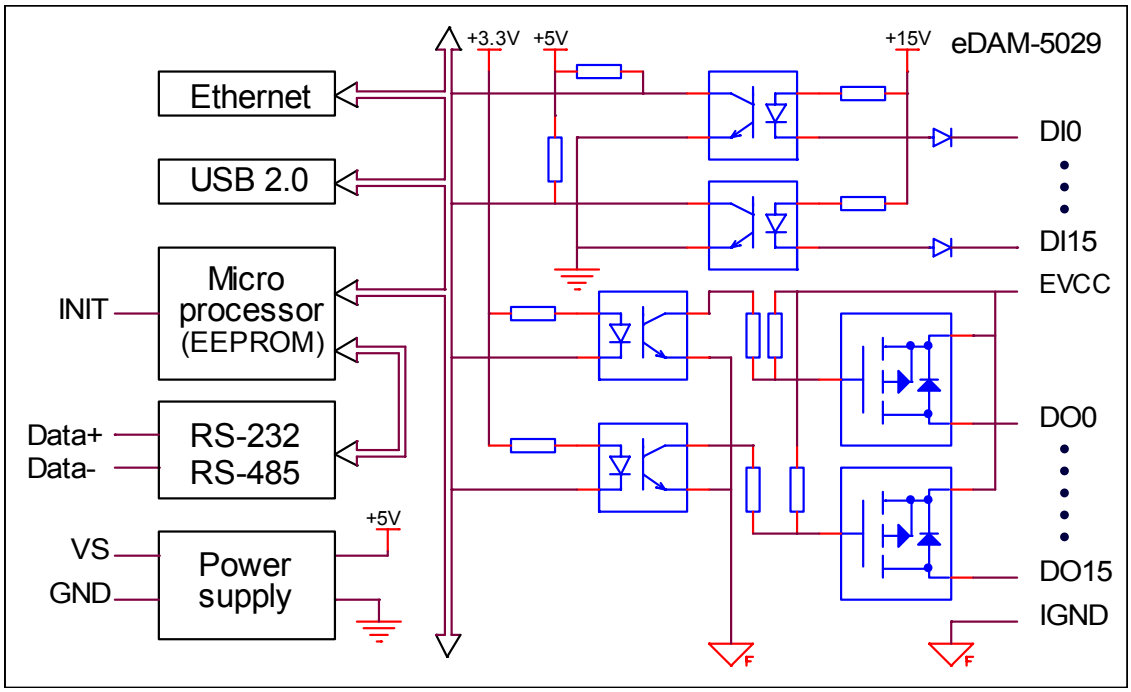
3.11 EDAM-5019 Analog/Digital I/O block diagram



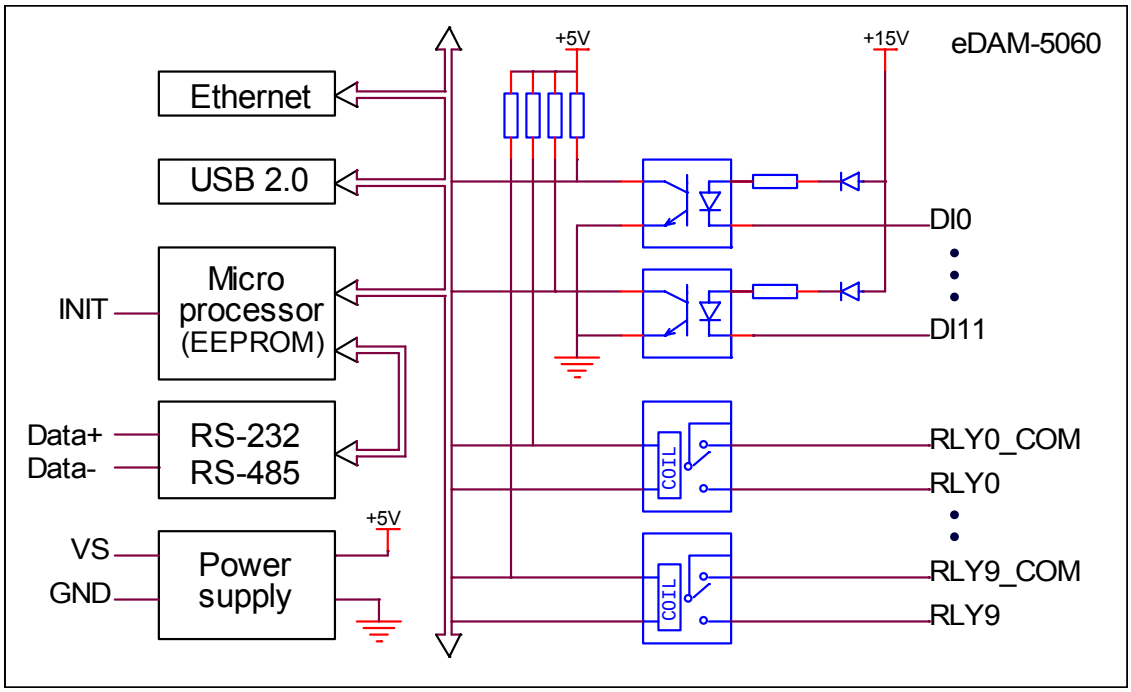
3.12 EDAM-5028 Analog/Digital I/O block diagram



3.13 EDAM-5029 Analog/Digital I/O block diagram



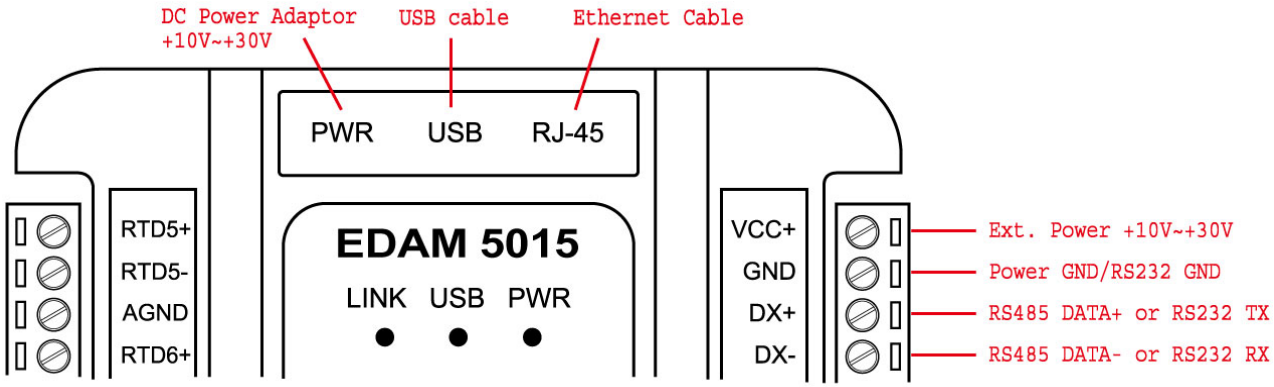
3.14 EDAM-5060 Analog/Digital I/O block diagram



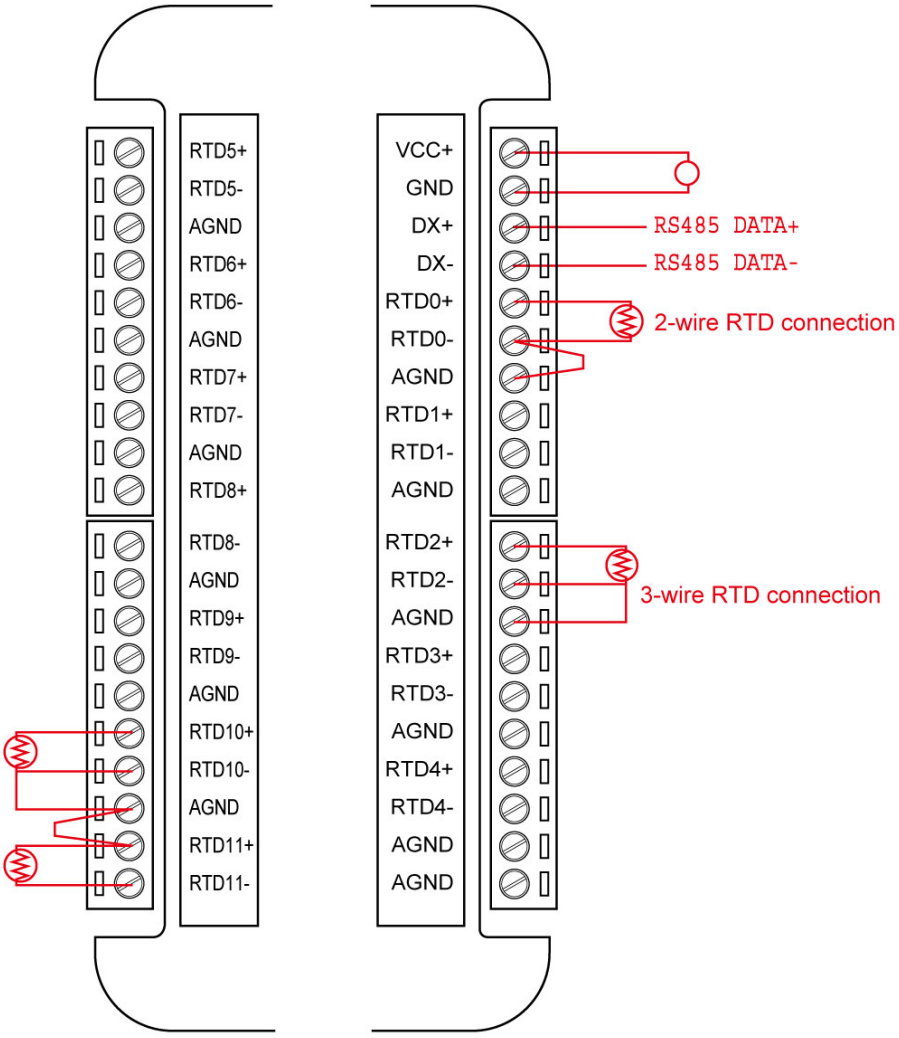
Chapter 4 Application wiring

4.1 EDAM-5015 wiring

4.1.1 Interface connection

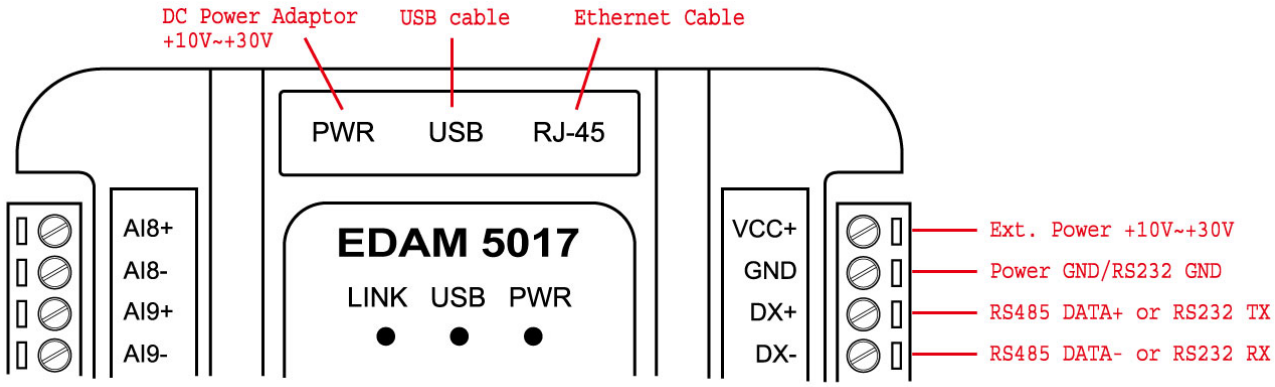


4.1.2 Analog input wiring

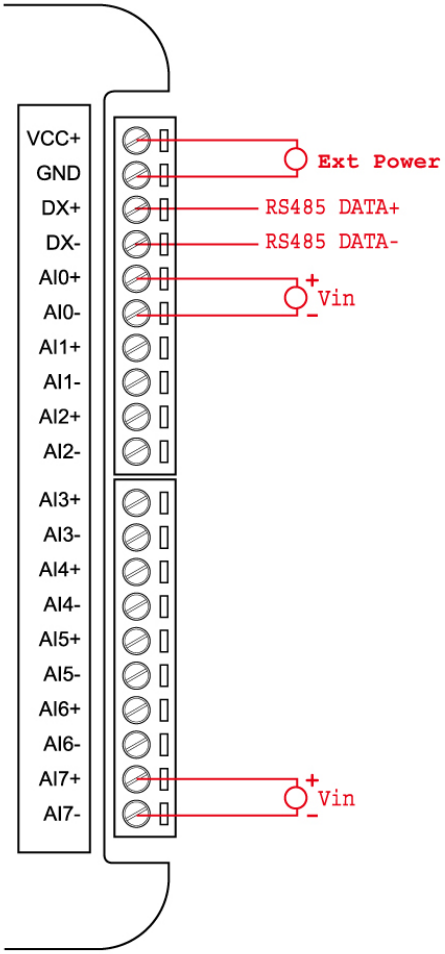


4.2 EDAM-5017 wiring

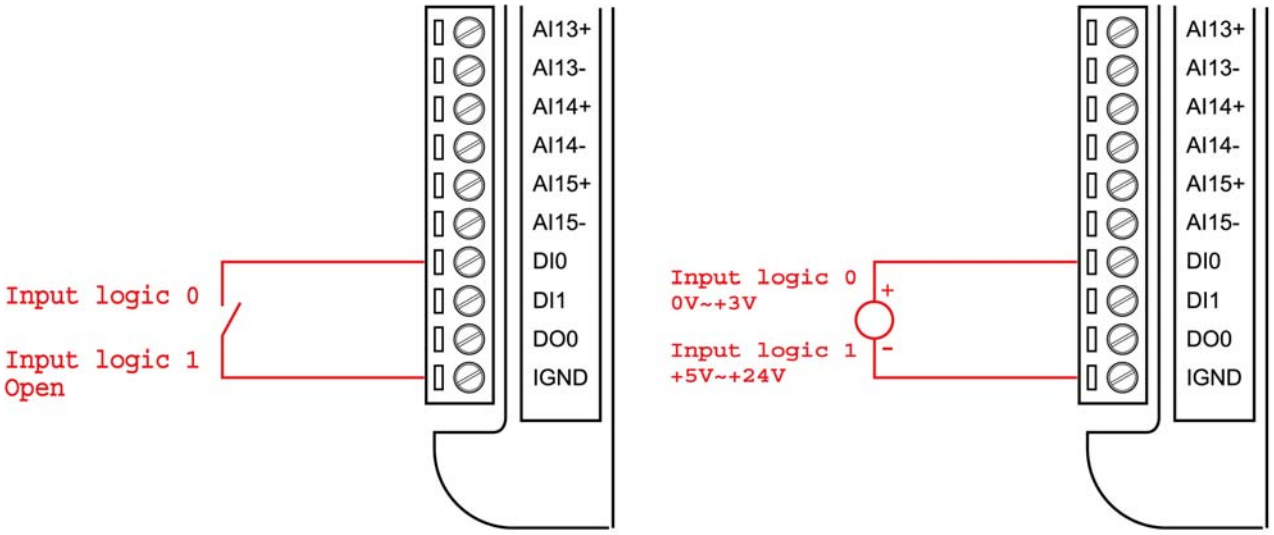
4.2.1 Interface connection



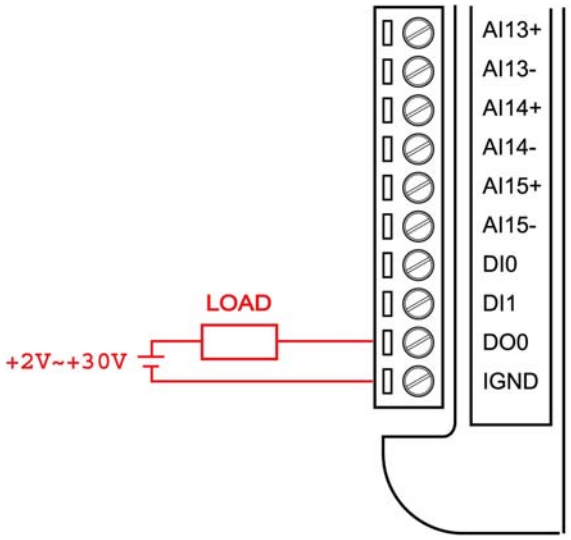
4.2.2 Analog input wiring



4.2.3 Digital input wiring

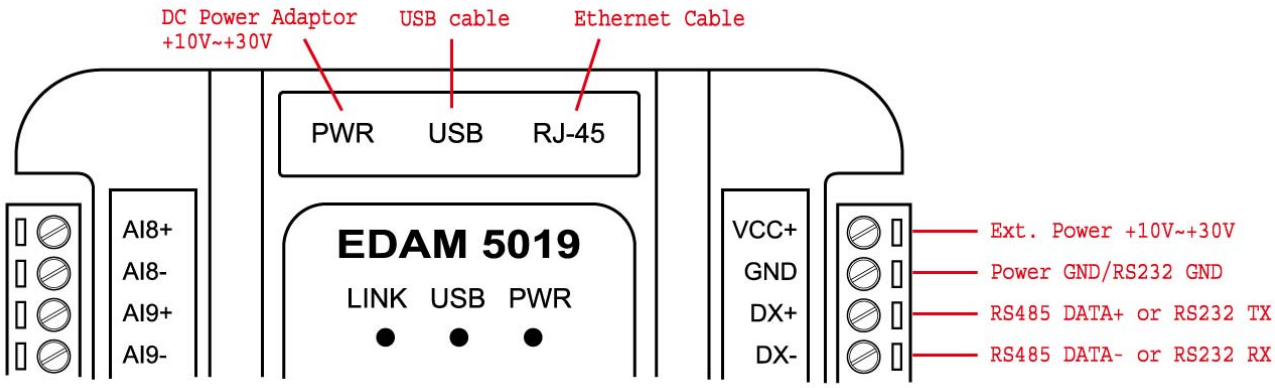


4.2.4 Digital output wiring



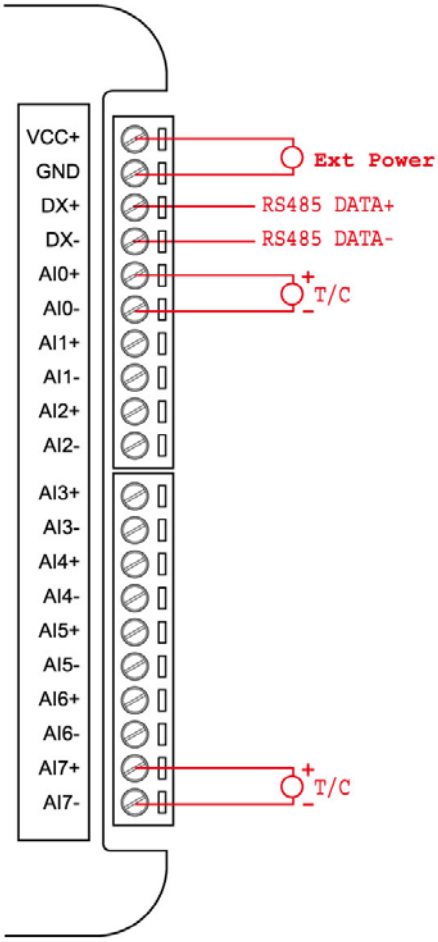
4.3 EDAM-5019 wiring

4.3.1 Interface connection



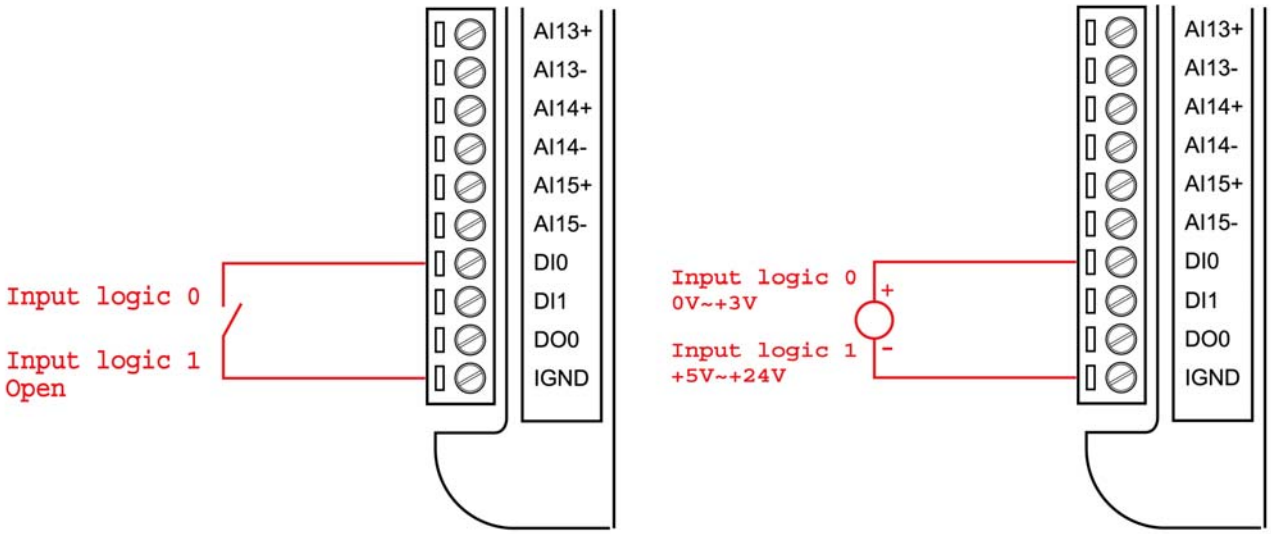
The function of DX+/DX- pins are depended on the Interface switch settings (see page 26)

4.3.2 Analog input wiring

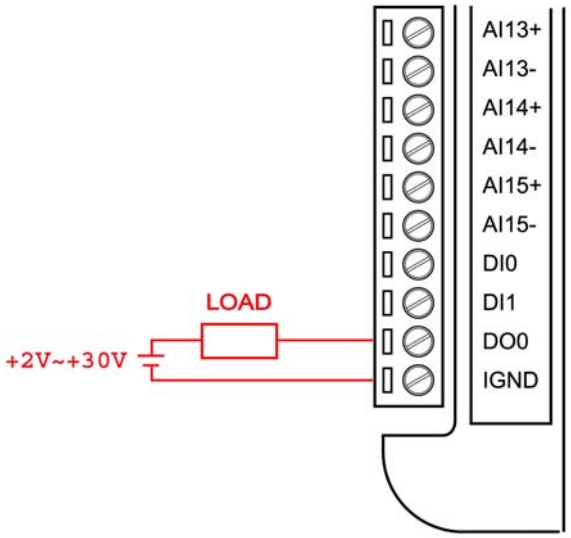


Where AIn+ and AIn- represent AI input channel n

4.3.3 Digital input wiring

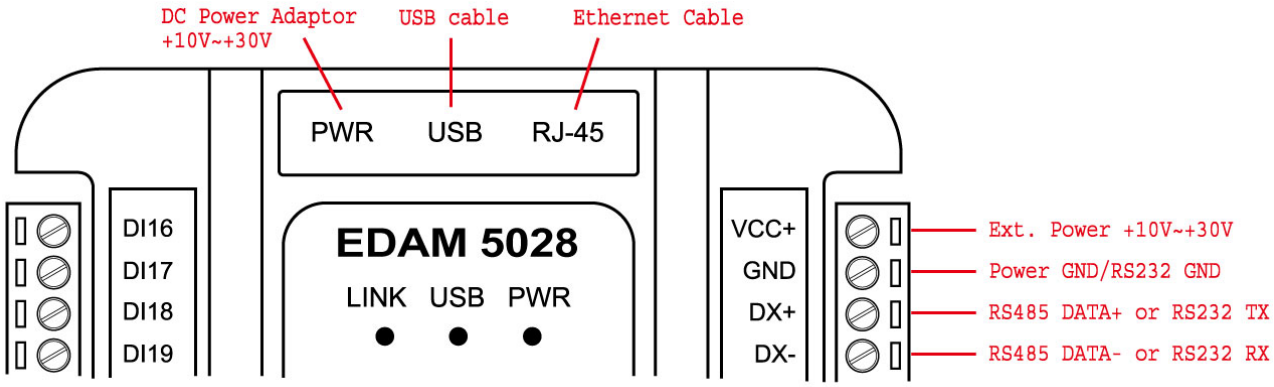


4.3.4 Digital output wiring

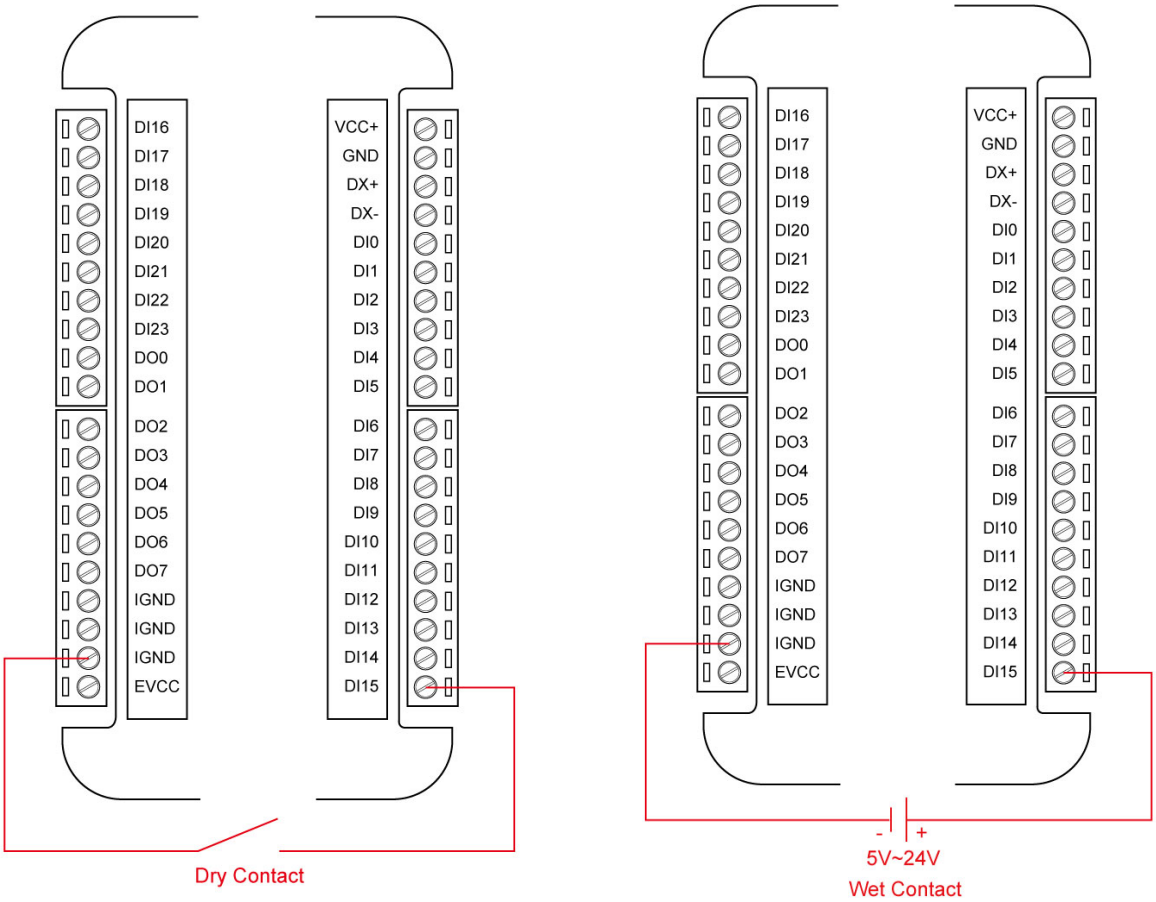


4.4 EDAM-5028 wiring

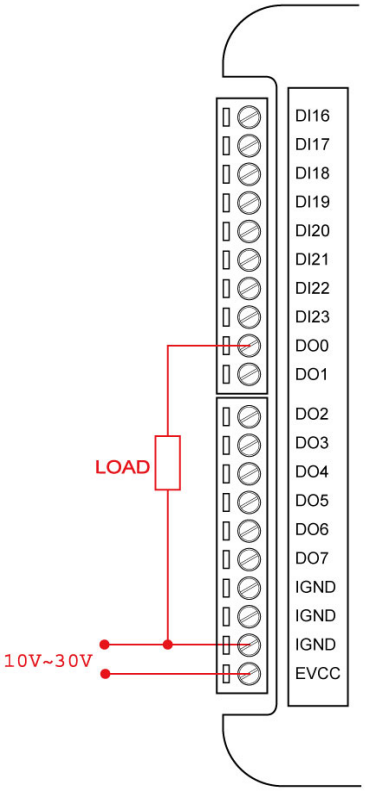
4.4.1 Interface connection



4.4.2 Digital input wiring

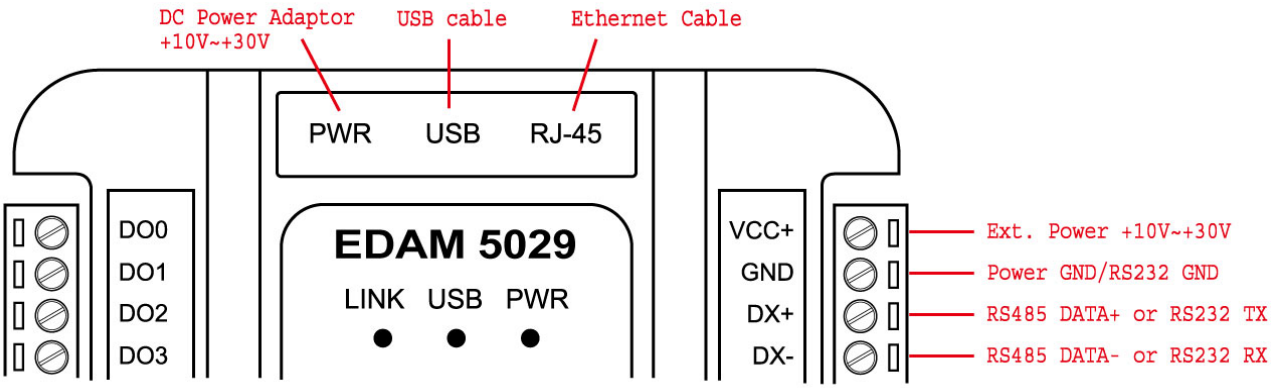


4.4.3 Digital output wiring

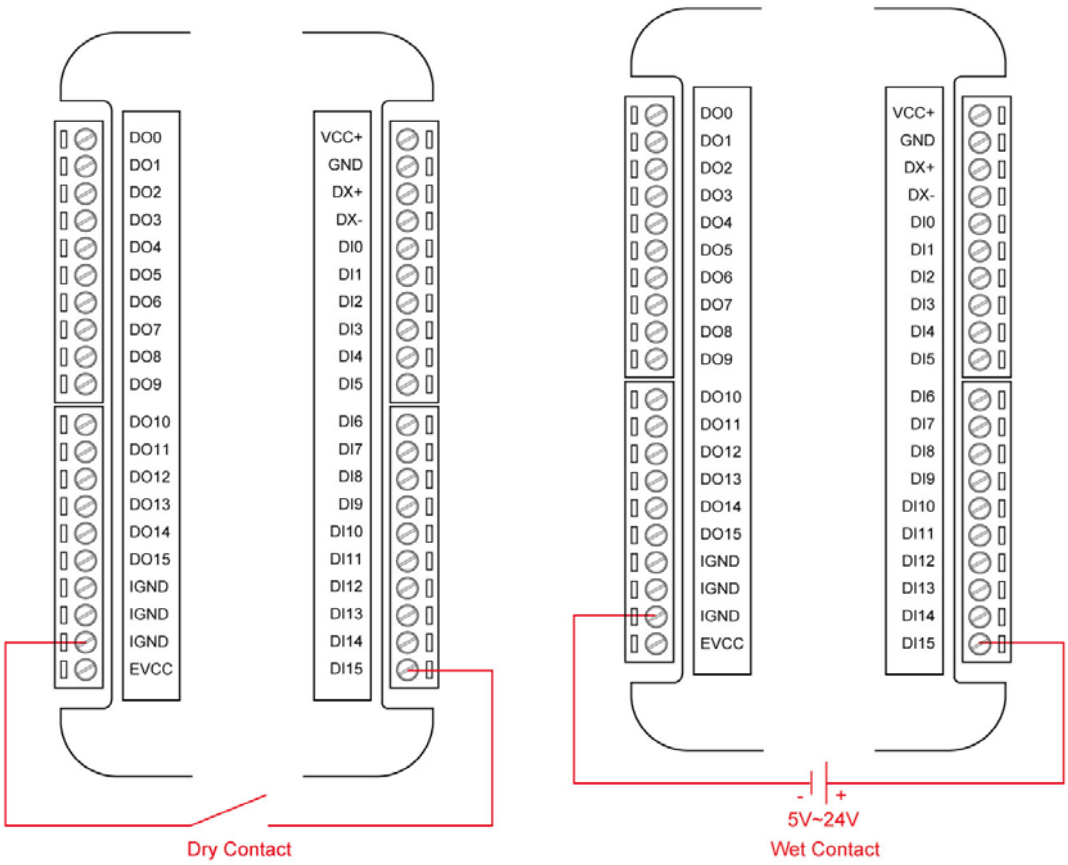


4.5 EDAM-5029 wiring

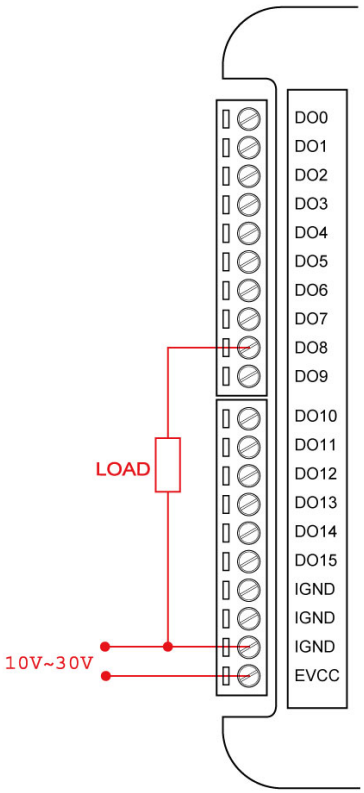
4.5.1 Interface connection



4.5.2 Digital input wiring

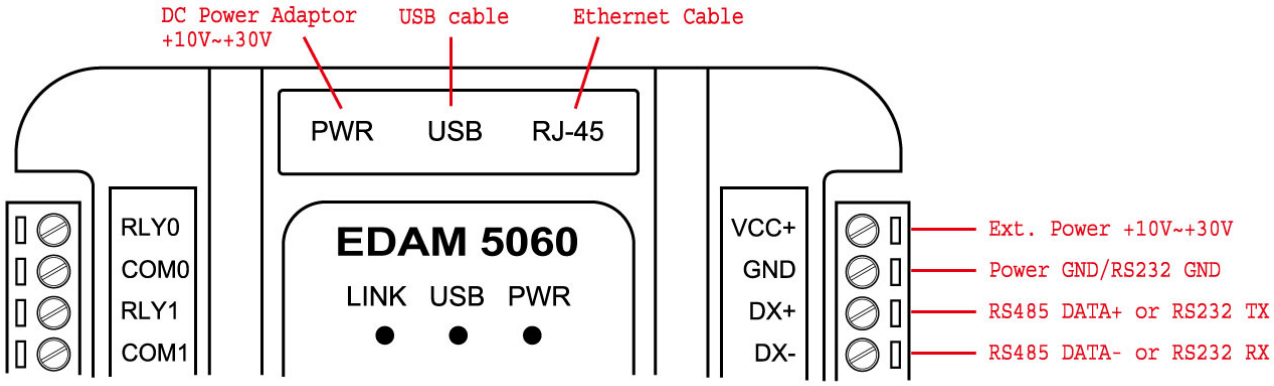


4.5.3 Digital output wiring

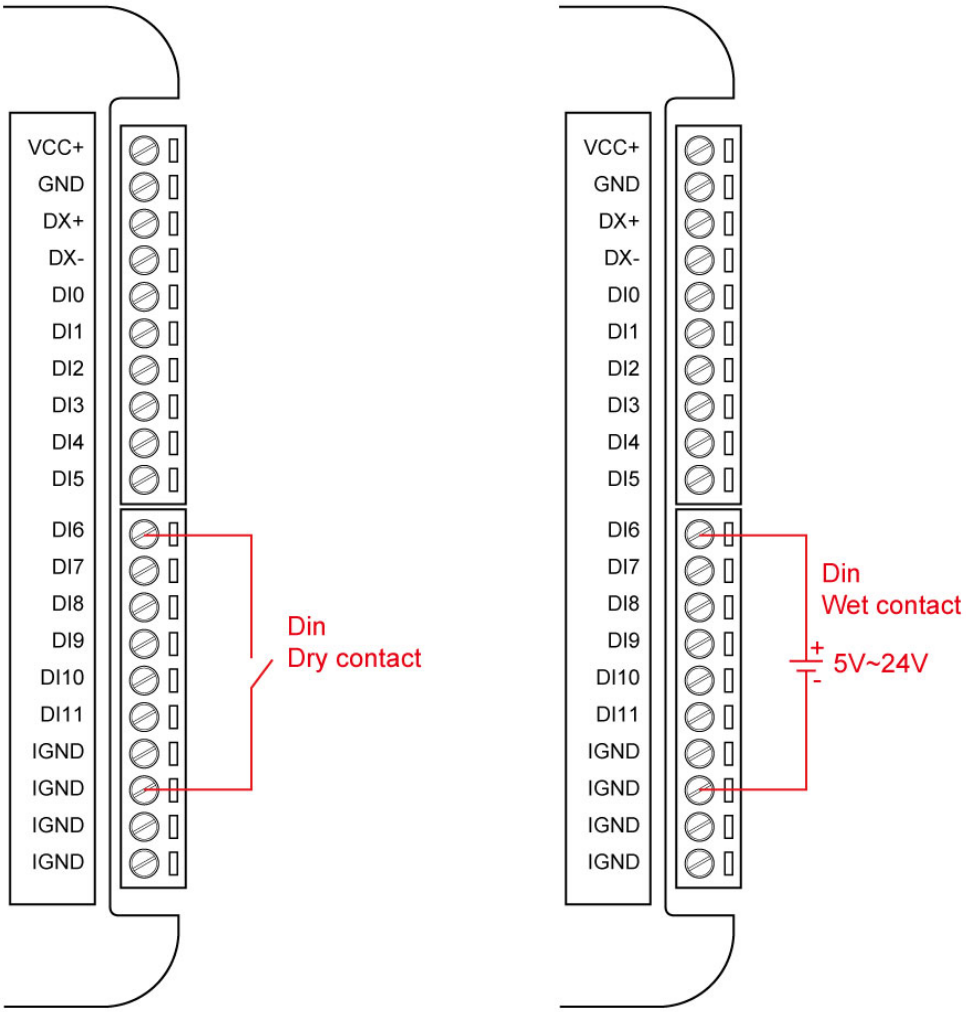


4.6 EDAM-5060 wiring

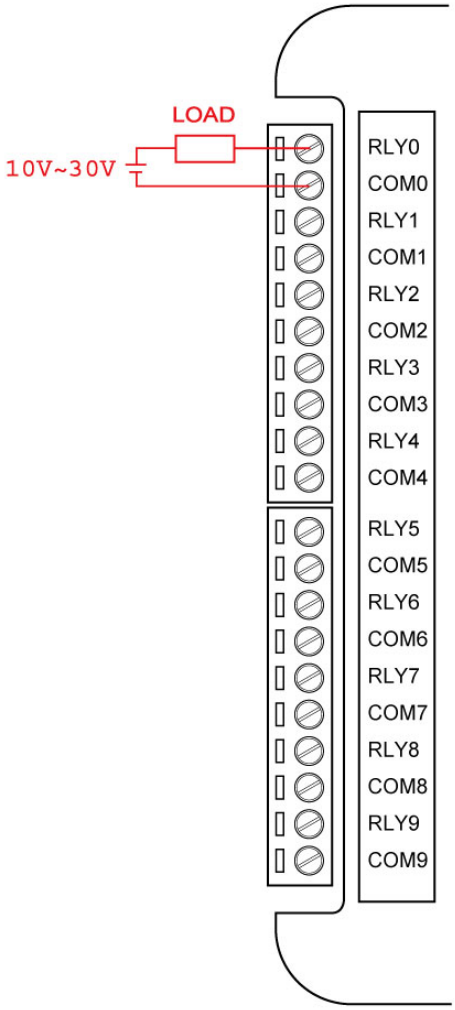
4.6.1 Interface connection



4.6.2 Digital input wiring



4.6.3 Digital output wiring



Chapter 5 Modbus Command structure

EDAM-5000 system accepts a command/response form with the host computer. When systems are not transmitting they are in listen mode. The host issues a command to a system with a specified address and waits a certain amount of time for the system to respond. If no response arrives, a time-out aborts the sequence and returns control to the host. This chapter explains the structure of the commands with Modbus/TCP protocol, and guides to use these command sets to implement user's programs.

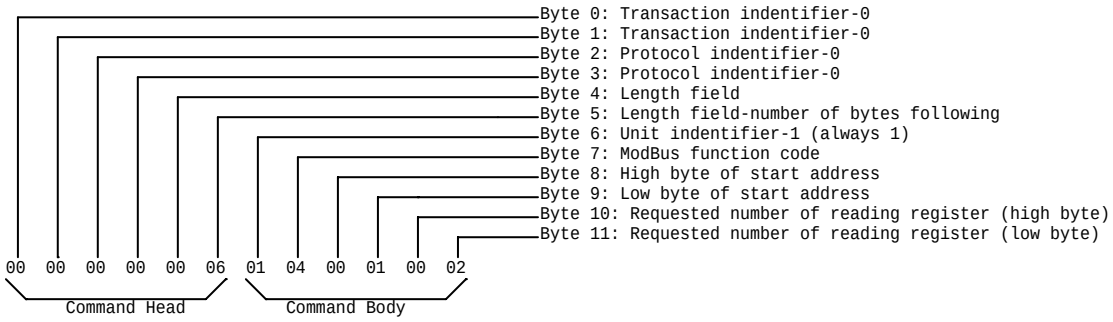
5.1 Command Structure

- **Modbus/TCP**

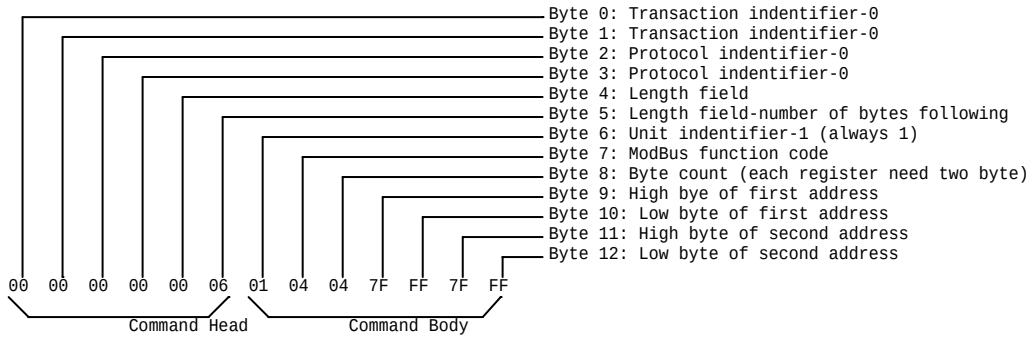
  It is important to understand the encapsulation of a Modbus request or response carried on the Modbus/TCP network. A complete command is consisted of ***command head*** and ***command body***. The command head is prefixed by six bytes and responded to pack Modbus format; the command body defines target device and requested action. Following example will help you to realize this structure quickly.

  Example:

  If you want to read the first two values of EADM-5019 (address: 40001~40002) with Modbus/TCP protocol, the request command should be:



  And the response should be:



- **Modbus/RTU**

  A Modbus request or response carried on the Modbus/RTU network. A complete command is consisted of ***command body only***. If you want to read the values of EADM-5019 with Modbus/RTU protocol, the request command is the same as Modbus/TCP, but ***without Command Head and first byte of Command body should be filled with module address***

5.2 Modbus function code introductions

Code (Hex)	Name	Usage
01	Read Coil Status	Read Discrete DI/DO Bit
02	Read Input Status	Read Discrete DI/DO Bit
03	Read Holding Registers	Read 16-bit register.
04	Read Input Registers	
05	Write Single Coil	Write data to force coil ON/OFF
06	Write Single Register	Write data in 16-bit integer format
0F	Force Multiple Coils	Write multiple data to force coil ON/OFF
10	Preset Multiple Registers	Write multiple data in 16-bit integer format

Chapter 6 Modbus Address Mapping

6.1 Modbus mapping of EDMA5015

6.1.1 Register address (unit: 16 bits)

This register address mapping support Modbus function 03(0x03), 04(0x04), 06(0x06), and 16(0x10)

Where: N=30000 for Function 04(0x04)
N=40000 for Function 03(0x03), Function 06(0x06), and Function 16(0x10)

Address(dec)	Channel	Item	Attribute
N+0290~N+0290	0~15	Analog input burnout status	
N+0291~N+0290	0~15	Analog input high alarm status	R
N+0292~N+0292	0~15	Analog input low alarm status	R
N+0294~N+0309	0~15	Analog input normal value	R (see sec. 7.2)
N+0310	Average	Analog input average value	R
N+0312~N+0327	0~15	Analog input maximum value	R (see sec. 7.2)
N+0312~N+0327	0~15	Analog input maximum value	R (see sec. 7.2)
N+0330~N+0345	0~15	Analog input minimum value	R
N+0348~N+0363	0~15	Analog Input type (0x0007~0x000E)	R/W
N+0364	Average	average type (0x0007~0x000E)	R/W

6.1.2 Bit address (unit: 1 bit)

This discrete address mapping support Modbus function 010x01), 02(0x02), 05(0x05), and 15(0x0F)

Where: N=00000 for Function 00(0x00), and 15(0x0F)
N=10000 for Function 01(0x01), Function 05(0x05)

Address(dec)	Channel	Item	Attribute
X+0256~N+0271	0~15	Enable/disable analog channel	R/W
N+0272~N+0287	0~15	Analog input high alarm status	R
N+0288~N+0303	0~15	Analog input low alarm status	R
N+0304~N+0319	0~15	Enable/disable analog channel in average	R/W
N+0320~X+0335	0~15	Reset analog input maximum value	R/W
N+0336~N+0351	0~15	Reset analog input minimum value	R/W
N+0352~N+0367	0~15	Clear analog input high alarm status	R/W
N+0368~N+0383	0~15	Clear analog input low alarm status	R/W
N+0384~N+0399	0~15	Read AD burnout status	R
N+0404		Enable/disable burnout detection (0=disable,1=enable)	R/W
N+0407		Enable/disable DHCP (0=disable,1=enable)	R/W
N+0408		Enable/disable Web Server (0=disnable,1=enable)	R/W
N+0409		Enable/disable CRC/Checksum (0=disable,1=enable)	R/W

Note:

In Modbus PDU each data is addressed is numbered from 1 to n

In the Modbus data model each element within a data block is numbered from 1 to n

6.2 Modbus mapping of EDMA5017

6.2.1 Register address (unit: 16 bits)

This register address mapping support Modbus function 03(0x03), 04(0x04), 06(0x06), and 16(0x10)

Where: N=30000 for Function 04(0x04)
N=40000 for Function 03(0x03), Function 06(0x06), and Function 16(0x10)

Address(dec)	Channel	Item	Attribute
N+0000~N+0000	0~1	Digital input data (0x0000~0x0003)	R
N+0002~N+0002	0	Digital input latch status (0x0000~0x0003)	R/W
N+0004~N+0007	0~1	Digital input counter value(2 words/channel)	R (see sec.7.1)
N+0068~N+0068	0~15	Digital output status DO0~DO15 (0x0000~0x0001)	R/W
N+0069~N+0069	16~31	Digital output status DO16~DO31 (0x0000~0x0001)	R/W
N+0080~N+0081	0~1	Digital input mode	R/W
N+0112~N+0113	0~1	Digital input debounce time interval (0~0xffff)	R/W
X+0176~N+0176	0	Digital output pulse low width(0000~0xFFFF in 0.5msec)	R/W
N+0208~N+0208	0	Digital output pulse high width(0000~0xFFFF in 0.5msec)	R/W
N+0240~N+0240	0	Digital output pulse counts	R/W
N+0272~N+0272	0~15	Digital power-on value DO0~DO15 (0x0000~0xFFFF)	R/W
N+0273~N+0273	16~31	Digital power-on value DO16~DO31 (0x0000~0xFFFF)	R/W
N+0290~N+0290	0~15	Analog input burnout status	
N+0291~N+0290	0~15	Analog input high alarm status	R
N+0292~N+0292	0~15	Analog input low alarm status	R
N+0293~N+0293	Cold junction	Cold junction temperature(in 0.1C)	R
N+0294~N+0309	0~15	Analog input normal value	R (see sec. 7.2)
N+0310	Average	Analog input average value	R
N+0312~N+0327	0~15	Analog input maximum value	R (see sec.7.2)
N+0312~N+0327	0~15	Analog input maximum value	R (see sec. 7.2)
N+0330~N+0345	0~15	Analog input minimum value	R
N+0348~N+0363	0~15	Analog Input type (0x0007~0x000E)	R/W
N+0364	Average	average type (0x0007~0x000E)	R/W

6.2.2 Bit address (unit: 1 bit)

This discrete address mapping support Modbus function 010x01), 02(0x02), 05(0x05), and 15(0x0F)

Where: N=00000 for Function 00(0x00), and 15(0x0F)
N=10000 for Function 01(0x01), Function 05(0x05)

Address(dec)	Channel	Item	Attribute
N+0000~N+0001	0~1	DI status (0X0000~0X0003)	R
N+0032~N+0033	0~1	DI latch status (0X0000~0X0003)	R
N+0064~N+0064	0	DO status	R/W
N+0096~N+0097	0~1	Clear DI latch status	R/W
N+0128~N+0129	0~1	Clear DI counter value	R/W
N+0160~N+0161	0~1	Enable/disable DI latch interrupt/Event) 0=disable, no generate interrupt or event 1=enable, generate interrupt or event (for USB/Ethernet connections only)	R/W
N+0224~N+0224	0~1	Start/Stop DO pulse output 0=disable DO pulse output 1=enable DO pulse out until Digital output pulse counts reaches zero (see * in Register address table)	R/W
X+0256~N+0271	0~15	Enable/disable analog channel	R/W
N+0272~N+0287	0~15	Analog input high alarm status	R
N+0288~N+0303	0~15	Analog input low alarm status	R
N+0304~N+0319	0~15	Enable/disable analog channel in average	R/W
N+0320~X+0335	0~15	Reset analog input maximum value	R/W
N+0336~N+0351	0~15	Reset analog input minimum value	R/W
N+0352~N+0367	0~15	Clear analog input high alarm status	R/W
N+0368~N+0383	0~15	Clear analog input low alarm status	R/W
N+0400		Save current DO as power on value	R/W
N+0405		Set DI active state (0=Open active,1=low active)	R/W
N+0406		Set DO active state (0=low active,1=open active)	R/W
N+0407		Enable/disable DHCP (0=disable,1=enable)	R/W
N+0408		Enable/disable Web Server (0=disnable,1=enable)	R/W
N+0409		Enable/disable CRC/Checksum (0=disable,1=enable)	R/W

Note:

In Modbus PDU each data is addressed is numbered from 1 to n

In the Modbus data model each element within a data block is numbered from 1 to n

6.3 Modbus mapping of EDMA5019

6.3.1 Register address (unit: 16 bits)

This register address mapping support Modbus function 03(0x03), 04(0x04), 06(0x06), and 16(0x10)

Where: N=30000 for Function 04(0x04)
N=40000 for Function 03(0x03), Function 06(0x06), and Function 16(0x10)

Address(dec)	Channel	Item	Attribute
N+0000~N+0000	0~1	Digital input data (0x0000~0x0003)	R
N+0002~N+0002	0	Digital input latch status (0x0000~0x0003)	R/W
N+0004~N+0007	0~1	Digital input counter value(2 words/channel)	R (see sec.7.1)
N+0068~N+0068	0~15	Digital output status DO0~DO15 (0x0000~0x0001)	R/W
N+0069~N+0069	16~31	Digital output status DO16~DO31 (0x0000~0x0001)	R/W
N+0080~N+0081	0~1	Digital input mode	R/W
N+0112~N+0113	0~1	Digital input debounce time interval (0~0xffff)	R/W
X+0176~N+0176	0	Digital output pulse low width(0000~0xFFFF in 0.5msec)	R/W
N+0208~N+0208	0	Digital output pulse high width(0000~0xFFFF in 0.5msec)	R/W
N+0240~N+0240	0	Digital output pulse counts	R/W
N+0272~N+0272	0~15	Digital power-on value DO0~DO15 (0x0000~0xFFFF)	R/W
N+0273~N+0273	16~31	Digital power-on value DO16~DO31 (0x0000~0xFFFF)	R/W
N+0290~N+0290	0~15	Analog input burnout status	
N+0291~N+0290	0~15	Analog input high alarm status	R
N+0292~N+0292	0~15	Analog input low alarm status	R
N+0293~N+0293	Cold junction	Cold junction temperature(in 0.1C)	R
N+0294~N+0309	0~15	Analog input normal value	R (see sec. 7.2)
N+0310	Average	Analog input average value	R
N+0312~N+0327	0~15	Analog input maximum value	R (see sec. 7.2)
N+0312~N+0327	0~15	Analog input maximum value	R (see sec. 7.2)
N+0330~N+0345	0~15	Analog input minimum value	R
N+0348~N+0363	0~15	Analog Input type (0x0007~0x000E)	R/W
N+0364	Average	average type (0x0007~0x000E)	R/W
N+366~N+381	0~15	AD channel cold junction offset(in 0.01C)	R/W

6.3.2 Bit address (unit: 1 bit)

This discrete address mapping support Modbus function 010x01), 02(0x02), 05(0x05), and 15(0x0F)

Where: N=00000 for Function 00(0x00), and 15(0x0F)
N=10000 for Function 01(0x01), Function 05(0x05)

Address(dec)	Channel	Item	Attribute
N+0000~N+0001	0~1	DI status (0X0000~0X0003)	R
N+0032~N+0033	0~1	DI latch status (0X0000~0X0003)	R
N+0064~N+0064	0	DO status	R/W
N+0096~N+0097	0~1	Clear DI latch status	R/W
N+0128~N+0129	0~1	Clear DI counter value	R/W
N+0160~N+0161	0~1	Enable/disable DI latch interrupt/Event) 0=disable, no generate interrupt or event 1=enable, generate interrupt or event (for USB/Ethernet connections only)	R/W
N+0224~N+0224	0~1	Start/Stop DO pulse output 0=disable DO pulse output 1=enable DO pulse out until Digital output pulse counts reaches zero (see * in Register address table)	R/W
X+0256~N+0271	0~15	Enable/disable analog channel	R/W
N+0272~N+0287	0~15	Analog input high alarm status	R
N+0288~N+0303	0~15	Analog input low alarm status	R
N+0304~N+0319	0~15	Enable/disable analog channel in average	R/W
N+0320~X+0335	0~15	Reset analog input maximum value	R/W
N+0336~N+0351	0~15	Reset analog input minimum value	R/W
N+0352~N+0367	0~15	Clear analog input high alarm status	R/W
N+0368~N+0383	0~15	Clear analog input low alarm status	R/W
N+0384~N+0399	0~15	Read AD burnout status	R
N+0400		Save current DO as power on value	R/W
N+0404		Enable/disable burnout detection (0=disable,1=enable)	R/W
N+0405		Set DI active state (0=Open active,1=low active)	R/W
N+0406		Set DO active state (0=low active,1=open active)	R/W
N+0407		Enable/disable DHCP (0=disable,1=enable)	R/W
N+0408		Enable/disable Web Server (0=disnable,1=enable)	R/W
N+0409		Enable/disable CRC/Checksum (0=disable,1=enable)	R/W

Note:

In Modbus PDU each data is addressed is numbered from 1 to n

In the Modbus data model each element within a data block is numbered from 1 to n

6.4 Modbus mapping of EDMA5028

6.4.1 Register address (unit: 16 bits)

This register address mapping support Modbus function 03(0x03), 04(0x04), 06(0x06), and 16(0x10)

Where: N=30000 for Function 04(0x04)
N=40000 for Function 03(0x03), Function 06(0x06), and Function 16(0x10)

Address(dec)	Channel	Item	Attribute
N+0000	0~15	Digital input data (0x0000~0xFFFF)	R
N+0001	16~23	Digital input data (0x0000~0x00FF)	R
N+0002	0~15	Digital input latch status (0x0000~0xFFFF)	R
N+0003	16~23	Digital input latch status (0x0000~0x00FF)	R
N+0004~N+0051	0~23	Digital input counter value(2 words/channel)	R (see sec.7.1)
N+0068	0~7	Digital output status DO0~DO7(0x0000~0x00FF)	R/W
N+0080~N+0103	0~23	Digital input mode	R/W
N+0112~N+0135	0~23	Digital input debounce time interval (0~0xFFFF)	R/W
X+0176~N+0183	0~7	Digital output pulse low width(0000~0xFFFF in 0.5msec)	R/W
N+0208~N+0215	0~7	Digital output pulse high width(0000~0xFFFF in 0.5msec)	R/W
N+0240~N+0247	0~7	Digital output pulse counts	R/W
N+0272	0~7	Digital power-on value DO0~DO7 (0x0000~0xFFFF)	R/W

6.4.2 Bit address (unit: 1 bit)

This discrete address mapping support Modbus function 010x01), 02(0x02), 05(0x05), and 15(0x0F)

Where: N=00000 for Function 00(0x00), and 15(0x0F)
N=10000 for Function 01(0x01), Function 05(0x05)

Address(dec)	Channel	Item	Attribute
N+0000~N+0023	0~23	DI status (0X000000~0XFFFFFF)	R
N+0032~N+0055	0~23	DI latch status (0X000000~0XFFFFFF)	R
N+0064~N+0071	0~7	DO status	R/W
N+0096~N+0119	0~23	Clear DI latch status	R/W
N+0128~N+0151	0~23	Clear DI counter value	R/W
N+0160~N+0183	0~23	Enable/disable DI latch interrupt/Event) 0=disable, no generate interrupt or event 1=enable, generate interrupt or event (for USB/Ethernet connections only)	R/W
N+0224~N+0231	0~7	Start/Stop DO pulse output 0=disable DO pulse output 1=enable DO pulse out until Digital output pulse counts reaches zero (see * in Register address table)	R/W
N+0400		Save current DO as power on value	R/W
N+0405		Set DI active state (0=Open active,1=low active)	R/W
N+0406		Set DO active state (0=low active,1=open active)	R/W
N+0407		Enable/disable DHCP (0=disable,1=enable)	R/W
N+0408		Enable/disable Web Server (0=disnable,1=enable)	R/W
N+0409		Enable/disable CRC/Checksum (0=disable,1=enable)	R/W

Note:

In Modbus PDU each data is addressed is numbered from 1 to n

In the Modbus data model each element within a data block is numbered from 1 to n

6.5 Modbus mapping of EDMA5029

6.5.1 Register address (unit: 16 bits)

This register address mapping support Modbus function 03(0x03), 04(0x04), 06(0x06), and 16(0x10)

Where: N=30000 for Function 04(0x04)
N=40000 for Function 03(0x03), Function 06(0x06), and Function 16(0x10)

Address(dec)	Channel	Item	Attribute
N+0000	0~15	Digital input data (0x0000~0xFFFF)	R
N+0002	0~15	Digital input latch status (0x0000~0xFFFF)	R
N+0004~N+0035	0~15	Digital input counter value(2 words/channel)	R (see sec.7.1)
N+0068	0~15	Digital output status DO0~DO15 (0x0000~0xFFFF)	R/W
N+0080~N+095	0~15	Digital input mode	R/W
N+0112~N+0127	0~15	Digital input debounce time interval (0~0xFFFF)	R/W
X+0176~N+0191	0~15	Digital output pulse low width(0000~0xFFFF in 0.5msec)	R/W
N+0208~N+0223	0~15	Digital output pulse high width(0000~0xFFFF in 0.5msec)	R/W
N+0240~N+0255	0~15	Digital output pulse counts	R/W
N+0272	0~7	Digital power-on value DO0~DO15 (0x0000~0xFFFF)	R/W

6.5.2 Bit address (unit: 1 bit)

This discrete address mapping support Modbus function 010x01), 02(0x02), 05(0x05), and 15(0x0F)

Where: N=00000 for Function 00(0x00), and 15(0x0F)
N=10000 for Function 01(0x01), Function 05(0x05)

Address(dec)	Channel	Item	Attribute
N+0000~N+0015	0~15	DI status (0X0000~0XFFFF)	R
N+0032~N+0047	0~15	DI latch status (0X0000~0XFFFF)	R
N+0064~N+0079	0~15	DO status	R/W
N+0096~N+0111	0~15	Clear DI latch status	R/W
N+0128~N+0143	0~15	Clear DI counter value	R/W
N+0160~N+0175	0~15	Enable/disable DI latch interrupt/Event) 0=disable, no generate interrupt or event 1=enable, generate interrupt or event (for USB/Ethernet connections only)	R/W
N+0224~N+0239	0~15	Start/Stop DO pulse output 0=disable DO pulse output 1=enable DO pulse out until Digital output pulse counts reaches zero (see * in Register address table)	R/W
N+0400		Save current DO as power on value	R/W
N+0405		Set DI active state (0=Open active,1=low active)	R/W
N+0406		Set DO active state (0=low active,1=open active)	R/W
N+0407		Enable/disable DHCP (0=disable,1=enable)	R/W
N+0408		Enable/disable Web Server (0=disnable,1=enable)	R/W
N+0409		Enable/disable CRC/Checksum (0=disable,1=enable)	R/W

Note:

In Modbus PDU each data is addressed is numbered from 1 to n

In the Modbus data model each element within a data block is numbered from 1 to n

6.6 Modbus mapping of EDMA5060

6.6.1 Register address (unit: 16 bits)

This register address mapping support Modbus function 03(0x03), 04(0x04), 06(0x06), and 16(0x10)

Where: N=30000 for Function 04(0x04)
N=40000 for Function 03(0x03), Function 06(0x06), and Function 16(0x10)

Address(dec)	Channel	Item	Attribute
N+0000	0~11	Digital input data (0x0000~0x0FFF)	R
N+0002	0~11	Digital input latch status (0x0000~0x0FFF)	R
N+0004~N+0027	0~11	Digital input counter value(2 words/channel)	R (see sec.7.1)
N+0068	0~9	Digital output status DO0~DO9(0x0000~0x03FF)	R/W
N+0080~N+091	0~11	Digital input mode	R/W
N+0112~N+0123	0~15	Digital input debounce time interval (0~0xFFFF)	R/W
X+0176~N+0187	0~15	Digital output pulse low width(0000~0xFFFF in 0.5msec)	R/W
N+0208~N+0217	0~9	Digital output pulse high width(0000~0xFFFF in 0.5msec)	R/W
N+0240~N+0249	0~9	Digital output pulse counts	R/W
N+0272	0~9	Digital power-on value DO0~DO9 (0x0000~0xFFFF)	R/W

6.6.2 Bit address (unit: 1 bit)

This discrete address mapping support Modbus function 010x01), 02(0x02), 05(0x05), and 15(0x0F)

Where: N=00000 for Function 00(0x00), and 15(0x0F)
N=10000 for Function 01(0x01), Function 05(0x05)

Address(dec)	Channel	Item	Attribute
N+0000~N+0011	0~11	DI status (0X000~0XFFF)	R
N+0032~N+0043	0~11	DI latch status (0X000~0XFFF)	R
N+0064~N+0073	0~9	DO status	R/W
N+0096~N+0107	0~11	Clear DI latch status	R/W
N+0128~N+0139	0~11	Clear DI counter value	R/W
N+0160~N+0171	0~11	Enable/disable DI latch interrupt/Event) 0=disable, no generate interrupt or event 1=enable, generate interrupt or event (for USB/Ethernet connections only)	R/W
N+0224~N+0233	0~9	Start/Stop DO pulse output 0=disable DO pulse output 1=enable DO pulse out until Digital output pulse counts reaches zero (see * in Register address table)	R/W
N+0400		Save current DO as power on value	R/W
N+0405		Set DI active state (0=Open active,1=low active)	R/W
N+0406		Set DO active state (0=low active,1=open active)	R/W
N+0407		Enable/disable DHCP (0=disable,1=enable)	R/W
N+0408		Enable/disable Web Server (0=disnable,1=enable)	R/W
N+0409		Enable/disable CRC/Checksum (0=disable,1=enable)	R/W

Note:

In Modbus PDU each data is addressed is numbered from 1 to n

In the Modbus data model each element within a data block is numbered from 1 to n

Chapter 7 Modbus data conversion

This chapter shows you how to convert Modbus register data to actual analog and digital value

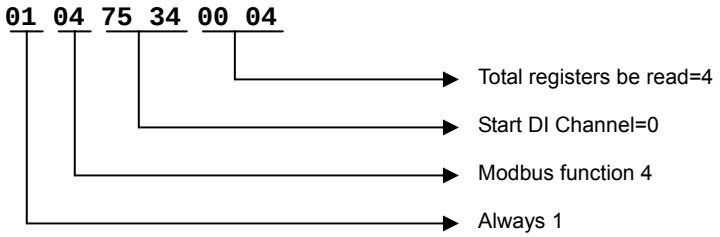
7.1 How to calculate DI counter value

Formula:

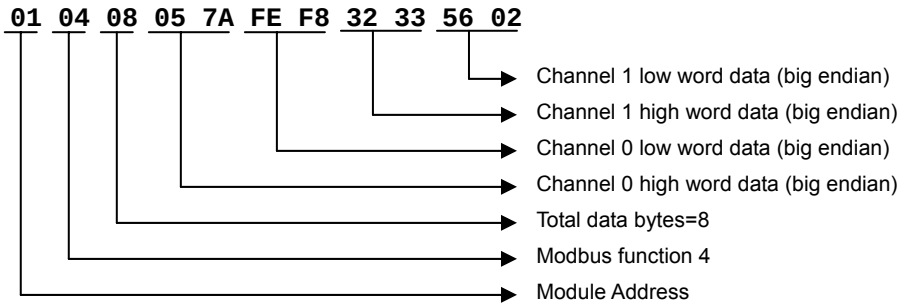
Actual DI Channel counts = (register value (high word) <<16) + register value (low word)

Example 1 written with C :

1. Assume the type of DI Channel 0 and channel 1 function as counter/frequency mode
2. Send Request command as : (Note: 0x7534=30004 start address of counter value of DI channel 0)



3. Receive Response from module as:




```
char Resp_data[];//Modbus response data received from Module
//where   Resp_data[6]=01    ; module address
//          Resp_data[7]=04    ; Modbus function 4
//          Resp_data[8]=08    ; total data bytes
//          Resp_data[9],[10]= 0x05,0x7A ;high data of channel 0
//          Resp_data[11],[12]= 0xFE,0xF8 ;low data of channel 0
//          Resp_data[13],[14]= 0x32,0x33 ;high data of channel 1
//          Resp_data[15],[16]= 0x56,0x02 ;low data of channel 1
long Chan0_Counts, Chan1_Counts;
Chan0_Counts =((long)Resp_data[9]<<24) | ((long)Resp_data[10]<<16) |
((long)Resp_data[11]<<8) | Resp_data[12];
Chan1_Counts =((long)Resp_data[13]<<24) | ((long)Resp_data[14]<<16) |
((long)Resp_data[15]<<8) | Resp_data[16];
printf ("\n\rChan 0 Counts=%d", Chan0_Counts);
printf ("\n\rChan 1 Counts=%d", Chan1_Counts);
```

4. Result :

```
Chan 0 Counts =91946744
Chan 1 Counts =842225154
```

7.2 How to convert Modbus data to AI voltage/temperature

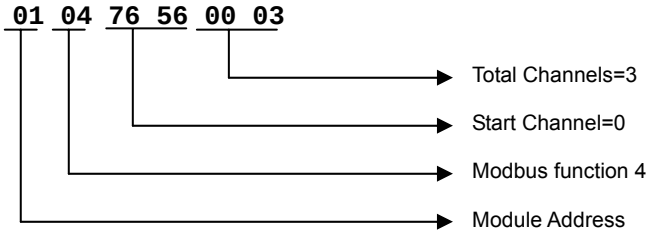
Formula:

Actual Channel voltage= (Modbus register value * Max range of channel type)/32767

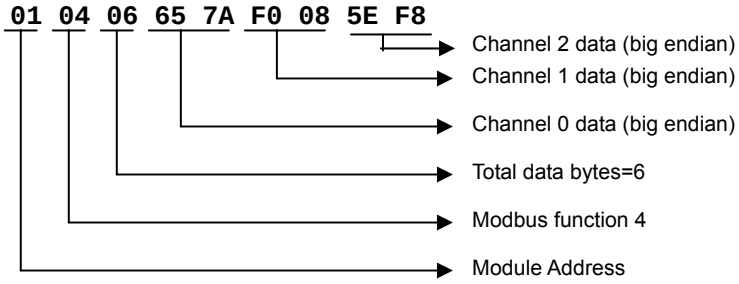
Example 1 written with C : Read analog input channel with type of voltage/current

Assume the type of Channel 0 is +/-2.5V, Channel 1 is +/-2.5V and Channel 2 is 4~20mA

Send Request command as : (Note: 0x7656=30294 start address of AI normal channel 0)



1. Receive Response from module as:

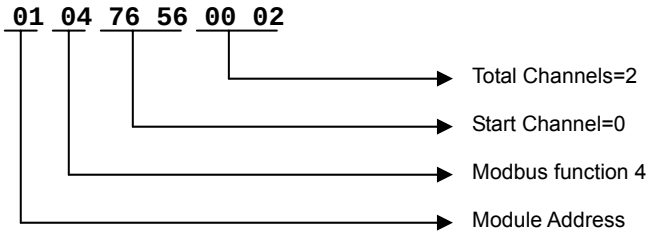



```
double Chan0_voltage,Chan1_voltage,Chan2_current;
char Resp_data[];    //Modbus response data received from Module
//where    Resp_data[6]=01    ; module address
//             Resp_data[7]=04    ; Modbus function 4
//             Resp_data[8]=06    ; total data bytes
//             Resp_data[9],[10]= 0x65,0x7A    ;data of channel 0
//             Resp_data[11],[12]= 0xF0,0x08    ;data of channel 1
//             Resp_data[13],[14]= 0x5E,0xF8    ;data of channel 2

short Chan0_data=(short)Resp_data[9]<<8 | Resp_data[10];      //0x657A=25978
short Chan1_data=(short)Resp_data[11]<<8 | Resp_data[12];    //0xF008=-4088
short Chan2_data=(short)Resp_data[13]<<8 | Resp_data[14];    //0x5EF8=24312
Chan0_voltage= ((double) Chan0_data * 2.5)/32767;    // Max.range=2.5V
Chan1_voltage= ((double) Chan1_data * 2.5)/32767;    // Max.range=2.5V
Chan2_current= ((double) Chan1_data * 20)/32767;      // Max.range=20mA
printf ("\n\rChan 0 voltage=%000.002f V", Chan0_voltage);
printf ("\n\rChan 1 voltage=%000.002f V", Chan1voltage);
printf ("\n\rChan 2 current=%000.002f mA", Chan2_current);

Result:
Chan 0 voltage=001.98 V
Chan 1 voltage=-000.31 V
Chan 2 current=014.84 mA
```

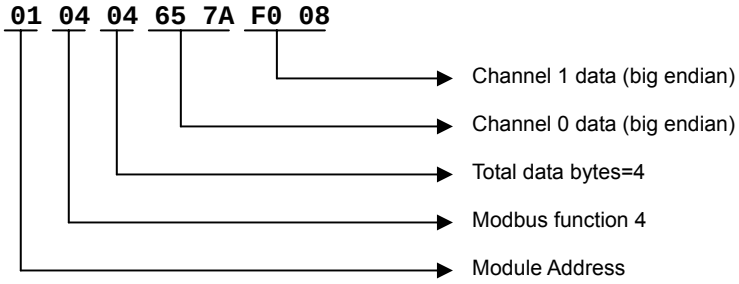
Example 2 written with C : Read analog input channel with type of temperature

(Assume Modbus data format is set to 2's complement format)

1. Assume the type of Channel 0 is T/C K type(*-100C~1370C)*, Channel 1 is T/C J type*(-100C~760C)*
2. Send Request command as : (Note: 0x7656=30294 start address of AI normal channel 0)



3. Receive Response from module as:

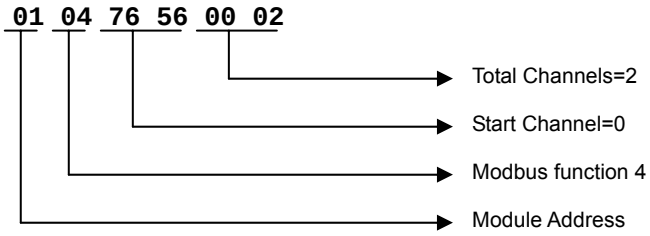



```
double Chan0_Temp,Chan1_Temp;
char Resp_data[];   //Modbus response data received from Module
//where   Resp_data[6]=01   ; module address
//          Resp_data[7]=04   ; Modbus function 4
//          Resp_data[8]=04   ; total data bytes
//          Resp_data[9],[10]= 0x65,0x7A   ;data of channel 0
//          Resp_data[11],[12]= 0xF0,0x08 ;data of channel 1

short Chan0_data=(short)Resp_data[9]<<8 | Resp_data[10];     //0x657A=25978
short Chan1_data=(short)Resp_data[11]<<8 | Resp_data[12];   //0xF008=-4088
Chan0_Temp= ((double) Chan0_data * 1370)/32767;   // K type has max.temp=1370
Chan1_Temp= ((double) Chan1_data * 760)/32767;     // J type has max.temp=760
printf ("\n\rChan 0 Temperature=%000.002f C", Chan0_Temp);
printf ("\n\rChan 1 Temperature=%000.002f C", Chan1_Temp);

Result:
Chan 0 Temperature=1086.15 C
Chan 1 Temperature=-094.82 C
```

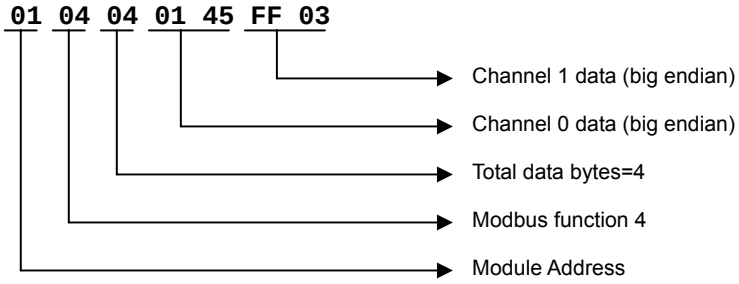
Example 3 written with C : Read analog input channel with type of temperature

(Assume Modbus data format is set to engineering format)

1. Assume the type of Channel 0 is T/C K type, Channel 1 is T/C J type
2. Send Request command as : (Note: 0x7656=30294 start address of AI normal channel 0)



3. Receive Response from module as:




```
double Chan0_Temp,Chan1_Temp;
char Resp_data[];   //Modbus response data received from Module
//where   Resp_data[6]=01   ; module address
//          Resp_data[7]=04   ; Modbus function 4
//          Resp_data[8]=04   ; total data bytes
//          Resp_data[9],[10]= 0x65,0x7A   ;data of channel 0
//          Resp_data[11],[12]= 0xF0,0x08 ;data of channel 1

short Chan0_data=(short)Resp_data[9]<<8 | Resp_data[10];     //0x0145=325
short Chan1_data=(short)Resp_data[11]<<8 | Resp_data[12];   //0Xff03=-253
Chan0_Temp= (double) Chan0_data /10;
Chan1_Temp= (double) Chan1_data/10;
printf ("\n\rChan 0 Temperature=%000.002f C", Chan0_Temp);
printf ("\n\rChan 1 Temperature=%000.002f C", Chan1_Temp);
```

Result:
Chan 0 Temperature=32.50 C
Chan 1 Temperature=-25.30 C

Chapter 8 Analog and digital /I/O channel type

8.1 DI channel types

Code	Type,	Models
00	DI transparent	5017,5019,5028,5029,5060
01	Counter	5017,5019,5028,5029,5060
02	low to high latch	5017,5019,5028,5029,5060
03	high to low latch	5017,5019,5028,5029,5060
04	Frequency	5017,5019,5028,5029,5060

8.2 AI channel types

Code	Type and Range	Models
0x07	+/-10V	5017
0x08	+/-5V	5017
0x09	+/-2.5V	5017,5019
0x0A	+/-1V	5017,5019
0x0B	+/-500mV	5017,5019
0x0C	+/-150mV	5017,5019
0x0D	0-20mA (125 ohms)	5017,5019
0x0E	4-20mA (125 ohms)	5017,5019
0x0F	T/C J type (*-100C~760C*)	5019
0x10	T/C K type (*-100C~1370C*)	5019
0x11	T/C T type (*-100C~400C*)	5019
0x12	T/C E type (*-100C~1000C*)	5019
0x13	T/C R type (*-500C~1750C*)	5019
0x14	T/C S type (*-500C~1750C*)	5019
0x15	T/C B type (*0C~1800C*)	5019
0x20	RTD IEC Pt100 (-50C ~ 150C)	5015
0x21	RTD IEC Pt100 (0C ~ 100C)	5015
0x22	RTD IEC Pt100 (0C ~ 200C)	5015
0x23	RTD IEC Pt100 (0C ~ 400C)	5015
0x24	RTD IEC Pt100 (-200C ~ 200C)	5015
0x25	RTD JIS Pt100 (-50C ~ 150C)	5015
0x26	RTD JIS Pt100 (0C ~ 100C)	5015
0x27	RTD JIS Pt100 (0C ~ 200C)	5015
0x28	RTD JIS Pt100 (0C ~ 400C)	5015
0x29	RTD JIS Pt100 (-200C ~ 200C)	5015
0x2A	RTD Pt1000 (-40C ~ 160C)	5015
0x2B	RTD Ni (-80C ~ 100C)	5015
0x2C	RTD Ni (0C ~ 100C)	5015

Chapter 9 TCP/IP port assignments

The following table shows you the TCP/IP ports used for EDAM-5000 series

Functions	**protocol**	**port**
Modbus/TCP protocol	TCP	502
ASCII command /Modbus RTU protocol	UDP	1025
Broadcast protocol	UDP	5048
Stream data	UDP	5148
Alarm Event data	UDP	5168
Httpd (web server)	TCP	80

Chapter 10 ASCII Commands

10.1 Analog commands

#AA	Reads the Analog Inputs of All	10.3
#AAn	Reads the single Analog Input	10.4
#AAMH	Read Maximum Value Of All Channels	10.5
#AAMHn	Read Maximum Value Of Specified Channel	10.6
$AAMH	Clear All Maximum Value	10.7
$AAMHn	Clear Maximum Value Of Specified Channel	10.8
#AAML	Read Minimum Value Of All Channels	10.9
#AAMLn	Read Minimum Value Of Specified Channel	10.10
$AAML	Clear All Minimum Value	10.11
$AAMLn	Clear Minimum Value Of Specified Channel	10.12
#AAAV	Read Average Value	10.13
$AAE	Read Channel Average Enable/Disable Status	10.14
$AAEnnnn	Disable/Enable Channel in Average	10.15
#AAAL	Read AD high/low Alarm Status	10.16
$AAAHnnnn	Clear A/D High Alarm	10.17
$AAALnnnn	Clear A/D Low Alarm	10.18
$AAB	Reads Channel burnout Status	10.19
%AAB	Read channel burnout enable/disable status	10.20
%AABn	enable/disable channel burnout	10.21
$AA3	Reads the CJC Temperature	10.22
~AAC	Reads the CJC Enable/disable	10.23
~AACn	Enables/Disables the CJC	10.24
$AA9Snnnn	Sets the all channel CJC Offset	10.25
$AA9c	Read single channel CJC Offset	10.26
$AA9cSnnnn	Set single channel CJC Offset	10.27
$AAR	Read A/D Filter Value	10.28
$AARf	Set A/D Filter Value	10.29
$AA6	Reads the Channel Enable/Disable Status	10.30
$AA5vvvv	Enables/Disables A/D Channel	10.31
$AA8Ci	Reads the Single A/D Channel Range	10.32
$AA7CiRrr	Sets the Single Channel Range	10.33
$AAS1	Reloads the Default Calibration	10.34

10.2 Digital commands

@AA	Reads the Digital I/O Status	10.35
@AAnn	Sets the Digital Output Channels	10.36
@AAnnnn	Sets the Digital Output Channels	10.37
@AAnnnnnn	Sets the Digital Output Channels	10.38
#AA0Ann	Sets the Digital 1's byte(DO0~DO7) Output	10.39
#AA0Bnn	Sets the Digital 2's byte(DO8~DO15) Output	10.40
#AA0Cnn	Sets the Digital 3's byte(DO16~DO23) Output	10.41
#AAnn	Reads the Digital Input Counter	10.42
$AACn	Clears the Digital Input Counter	10.43
$AACnn	Clears the Digital Input Counter	10.44
$AALS	Reads the Latched DI Status	10.45
$AAC	Clears the Latched DI Status	10.46
$AA9nn	Read Single Do Pulse High/Low Width	10.47
$AA9nnhhhhllll	Set Single Do Pulse High/Low Width	10.48
$AAAnn	Read Single Do High/Low Delay Width	10.49
$AAAnnhhhhllll	Set Single Do High/Low Delay Width	10.50
$AABnn	Read Single Do Pulse Counts	10.51
#AA2nncccc	Write Single Do Pulse Counts	10.52
#AA3nns	Start/Stop Do Pulse Counts	10.53
~AA4v	Reads the PowerOn/Safe Value	10.54
~AA5v	Sets current DO value as PowerOn/Safe Value	10.55
~AA5vnnnnnn	Sets specified value as PowerOn/Safe Value	10.56
~AAD	Read DI/O active state	10.57
~AADvv	Set DI/O active state	10.58

10.3 #AA Read the analog Inputs of all

Description The command will return the input value from a specified (AA) module in the currently configured data format.

Syntax **#AA(cr)**

is a delimiter character.

AA (range 00-FF) represents the 2-character hexadecimal address of an analog input module.

(cr) is the terminating character, carriage return (0Dh).

Response **>(data)(cr)** if the command is valid or **?AA (cr)** if the command is invalid

There is no response if the module detects a syntax error or communication error.

> is a delimiter character.

(data) is the input value in the configured data format of the module.

(cr) is the terminating character, carriage return (0Dh).

Example Command: **#21(cr)**

Response:
>+7.2111+7.2567+7.3125+7.1000+7.4712+7.2555+7.1234+7.5678+7.2111+7.2567+7.3125+7.1000+7.4712+7.2555+7.1234+7.5678 +3.5678 (cr)

The command response the analog input module at address 21h for its input values of all channels.The analog input module responds with channels from 0 to 15 with +7.2111 volts, +7.2567 volts,+7.3125 volts, +7.1000 volts, +7.4712 volts, +7.2555 volts, +7.1234 volts , +7.5678 volts,+7.2111 volts, +7.2567 volts,+7.3125 volts, +7.1000 volts, +7.4712 volts, +7.2555 volts, +7.1234 volts and +7.5678 volts. The average value is **+3.5678** volts

Example Command: **#01(cr)**

Response: **>FF5DE4323212AE332334 5663E000FF03 FF5DE4323212AE332334 5663E000FF03FE02 (cr)**

The analog input module at address01 has an input value of **FF5DE4323212AE332334 5663E000FF03FF5DE4323212AE332334 5663E000FF03FE02**. (The configured data format of the analog input module is two's complement)Where **FE02** is the average value

10.4 #AAn Read the single analog input

Description The command will return the input value from one of the all channels of a specified (AA) module in the currently configured data format.

Syntax **#AAN(cr)**

is a delimiter character.

AA (range 00-3F) represents the 2-character hexadecimal address of the analog input module.

N identifies the channel you want to read. The value can range from 0 to F

(cr) is the terminating character, carriage return (0Dh).

Response **>(data)(cr)** if the command is valid or **?AA (cr)** if the command is invalid

There is no response if the module detects a syntax error or communication error.

> is a delimiter character.

(data) is the input value of the channel number N. Data consists of a + or - sign followed by five decimal digits with a fixed decimal point.

(cr) is the terminating character, carriage return (0Dh).

Example Command: **#120(cr)**

Response: **>+1.4567(cr)**

The command requests the analog input module at address 12h to return the input value of channel 0. The analog input module responds that the input value of channel 0 is equal to +1.4567 volts.

10.5 #AAMH Read Maximum Value Of All Channels

Description The command will return the Read Maximum Value Of All Channels from a specified (AA) module in the currently configured data format.

Syntax **#AAMH(cr)**

is a delimiter character.

AA (range 00-FF) represents the 2-character hexadecimal address of an analog input module.

MH Read all channel maximum value command

(cr) is the terminating character, carriage return (0Dh).

Response **>AA(data)(cr)** if the command is valid or **?AA (cr)** if the command is invalid

There is no response if the module detects a syntax error or communication error.

AA (range 00-FF) represents the 2-character hexadecimal address of the analog input module.

> is a delimiter character.

(data) is the Maximum value of all channels in the configured data format of the module.

(cr) is the terminating character, carriage return (0Dh).

Example Command: **#21MH(cr)**

Response: **!21+7.2111+7.2567+7.3125+7.1000+7.4712+7.2555+7.1234+7.5678+7.2111+7.2567+7.3125+7.1000+7.4712+7.2555+7.1234+7.5678 (cr)**

The command response the analog input module at address 21h for its maximum values of all channels.The analog input module responds its maximum values with channels from 0 to 15 with +7.2111 volts,+7.2567 volts,+7.3125 volts, +7.1000 volts, +7.4712 volts, +7.2555 volts, +7.1234 volts ,+7.5678 volts,+7.2111 volts,+7.2567 volts,+7.3125 volts, +7.1000 volts, +7.4712 volts, +7.2555 volts, +7.1234 volts and +7.5678 volts

Example Command: **#01MH(cr)**

Response: **!01FF5DE4323212AE3323345663E000FF03 F5DE4323212AE3323345663E000FF03 (cr)**

The analog input module at address01 has maximum values of **FF5DE4323212AE3323345663E000FF03FF5DE4323212AE3323345663E000FF03**. (The configured data format of the analog input module is two's complement)

10.6 #AAMHn Read Maximum Value of Specified Channel

Description The command will return the Read Maximum value Of the specified channel from a specified (AA) module in the currently configured data format.

Syntax **#AAMHn(cr)**

is a delimiter character.

AA (range 00-FF) represents the 2-character hexadecimal address of an analog input module.

MH read single channel maximum value command

N channel number

(cr) is the terminating character, carriage return (0Dh).

Response **>AA(data)(cr)** if the command is valid or **?AA (cr)** if the command is invalid

There is no response if the module detects a syntax error or communication error.

AA (range 00-FF) represents the 2-character hexadecimal address of the analog input module.

> is a delimiter character.

(data) is the Maximum value of the specified channels.

(cr) is the terminating character, carriage return (0Dh).

Example Command: **#21MH2(cr)**

Response: **!21+7.2111(cr)**

The command response the analog input module at address 21h for its maximum values of the channels 2 with +7.2111 volts

Example Command: **#01MH2(cr)**

Response: **!01FF5D (cr)**

The command response the analog input module at address 21h for its maximum values of the channels 2 with **FF5D** (The configured data format of the analog input module is two's complement)

10.7 $AAMH Clear All Maximum Value

Description clear maximum Value Of all channels

Syntax **$AAMH(cr)**

$ is a delimiter character.

AA (range 00-FF) represents the 2-character hexadecimal address of module.

MH is the clear maximum value of specified channel command.

(cr) is the terminating character, carriage return (0Dh).

Response **!AA(cr)** if the command is valid or **?AA (cr)** if the command is invalid.

There is no response if the module detects a syntax error or communication error.

! delimiter character indicates a valid command was received.

? delimiter character indicates the command was invalid.

AA (range 00-3F) represents the 2-character hexadecimal address of module.

(cr) is the terminating character, carriage return (0Dh).

Example Command: **$01MH(cr)**

Response: **!01(cr)**

Clear Maximum Value Of all Channels

10.8 $AAMHn Clear Maximum value Of specified Channel

Description clear maximum Value Of specified channel

Syntax **$AAMHn(cr)**

$ is a delimiter character.

AA (range 00-FF) represents the 2-character hexadecimal address of module.

MH is the clear maximum value of specified channel command.

n channel 0~15

(cr) is the terminating character, carriage return (0Dh).

Response **!AA(cr)** if the command is valid or **?AA (cr)** if the command is invalid.

There is no response if the module detects a syntax error or communication error.

! delimiter character indicates a valid command was received.

? delimiter character indicates the command was invalid.

AA (range 00-3F) represents the 2-character hexadecimal address of module.

(cr) is the terminating character, carriage return (0Dh).

Example Command: **$01MHE(cr)**

Response: **!01(cr)**

Clear Maximum Value Of Channel 14 (0x0E)

10.9 #AAML Read Minimum Value Of All Channels

Description The command will return the Read Minimum Value Of All Channels from a specified (AA) module in the currently configured data format.

Syntax **#AAML(cr)**

is a delimiter character.

AA (range 00-FF) represents the 2-character hexadecimal address of an analog input module.

ML Read all channel minimum value command

(cr) is the terminating character, carriage return (0Dh).

Response **>AA(data)(cr)** if the command is valid or **?AA (cr)** if the command is invalid

There is no response if the module detects a syntax error or communication error.

AA (range 00-FF) represents the 2-character hexadecimal address of the analog input module.

> is a delimiter character.

(data) is the minimum value of all channels in the configured data format of the module.

(cr) is the terminating character, carriage return (0Dh).

Example Command: **#21ML(cr)**

Response: **!21+7.2111+7.2567+7.3125+7.1000+7.4712+7.2555+7.1234+7.5678+7.2111+7.2567+7.3125+7.1000+7.4712+7.2555+7.1234+7.5678 (cr)**

The command response the analog input module at address 21h for its minimum values of all channels.The analog input module responds its minimum values with channels from 0 to 15 with +7.2111 volts,+7.2567 volts,+7.3125 volts, +7.1000 volts, +7.4712 volts, +7.2555 volts, +7.1234 volts ,+7.5678 volts,+7.2111 volts,+7.2567 volts,+7.3125 volts, +7.1000 volts, +7.4712 volts, +7.2555 volts, +7.1234 volts and +7.5678 volts

Example Command: **#01ML(cr)**

Response: **!01FF5DE4323212AE332334 5663E000FF03 F5DE4323212AE332334 5663E000FF03(cr)**

The analog input module at address01 has minimum values of **FF5DE4323212AE332334 5663E000FF03FF5DE4323212AE332334 5663E000FF03**. (The configured data format of the analog input module is two's complement)

10.10 #AAMLn Read Minimum Value Of Specified Channel

Description The command will return the read minimum value of the specified channel from a specified (AA) module in the currently configured data format.

Syntax **#AAMLn(cr)**

is a delimiter character.

AA (range 00-FF) represents the 2-character hexadecimal address of an analog input module.

ML read single channel maximum value command

N channel number

(cr) is the terminating character, carriage return (0Dh).

Response **>AA(data)(cr)** if the command is valid or **?AA (cr)** if the command is invalid

There is no response if the module detects a syntax error or communication error.

AA (range 00-FF) represents the 2-character hexadecimal address of the analog input module.

> is a delimiter character.

(data) is the minimum value of the specified channels.

(cr) is the terminating character, carriage return (0Dh).

Example Command: **#21ML2(cr)**

Response: **!21+7.2111(cr)**

The command response the analog input module at address 21h for its minimum values of the channels 2 with +7.2111 volts

Example Command: **#01ML2(cr)**

Response: **!01FF5D (cr)**

The command response the analog input module at address 21h for its minimum values of the channels 2 with **FF5D** (The configured data format of the analog input module is two's complement)

10.11 $AAML Clear All Minimum Value

Description clear minimum Value Of all channels

Syntax **$AAML(cr)**

$ is a delimiter character.

AA (range 00-3F) represents the 2-character hexadecimal address of module.

ML is the clear minimum value of all channels command.

(cr) is the terminating character, carriage return (0Dh).

Response **!AA(cr)** if the command is valid or **?AA (cr)** if the command is invalid.

There is no response if the module detects a syntax error or communication error.

! delimiter character indicates a valid command was received.

? delimiter character indicates the command was invalid.

AA (range 00-3F) represents the 2-character hexadecimal address of module.

(cr) is the terminating character, carriage return (0Dh).

Example Command: **$01ML(cr)**

Response: **!01(cr)**

Clear Minimum Value Of all Channels

10.12 $AAMLn Clear Minimum Value Of specified Channel

Description Clear Minimum Value Of specified Channel

Syntax **$AAMLn(cr)**

$ is a delimiter character.

AA (range 00-3F) represents the 2-character hexadecimal address of module.

ML is the clear minimum value of all channels command.

n channel 0~15

(cr) is the terminating character, carriage return (0Dh).

Response **!AA(cr)** if the command is valid or **?AA (cr)** if the command is invalid.

There is no response if the module detects a syntax error or communication error.

! delimiter character indicates a valid command was received.

? delimiter character indicates the command was invalid.

AA (range 00-3F) represents the 2-character hexadecimal address of module.

(cr) is the terminating character, carriage return (0Dh).

Example Command: **$01MLE(cr)**

Response: **!01(cr)**

Clear Minimum Value Of Channel 14 (0x0E)

10.13 #AAAV Read Average Value

Description The command will return the average value from the channels which is in average mode

Syntax **#AAV(cr)**

is a delimiter character.

AA (range 00-3F) represents the 2-character hexadecimal address of the analog input module.

V identifies Read Ad average value command

(cr) is the terminating character, carriage return (0Dh).

Response **>(data)(cr)** if the command is valid or **?AA (cr)** if the command is invalid

There is no response if the module detects a syntax error or communication error.

! is a delimiter character.

The command requests the analog input module at address 12h to return the input value of channel 0. The analog input module responds that the input value of channel 0 is equal to +1.4567 volts.

Example Command: **#12V(cr)**

Response: **>+1.4567(cr)**

The command requests the analog input module at address 12h to return the average value of the module. The analog input module responds that the average value of module is equal to +1.4567 volts.

10.14 $AAE Read Channel Average Enable/Disable Status

Description read A/D channel in average status

Syntax **$AAE (cr)**

$ is a delimiter character.

AA (range 003F) represents the 2-character hexadecimal address of module.

E is read A/D channel in average status command.

(cr) is the terminating character, carriage return (ODh).

Response

!AAnnnn(cr) if the command is valid or **?AA (cr)** if the command is invalid.

There is no response if the module detects a syntax error or communication error.

! is a delimiter character indicating a valid command was received.

AA (range 00-3F) represents the 2-character hexadecimal address of module.

nnnn are four hexadecimal values. The values are interpreted by the module as four binary words (4-bit). The first word represents channel 12~15, and the second word represents channel 8~11…etc.
bit x=1 channel x is in average, bit x=0 channel isn't in average

(cr) is the terminating character, carriage return

Examples Command: **$01E**(cr)

Response: **!011020 (cr)**

Channel 5 and channel 15 are in average only

10.15 $AAEnnnn Disable/Enable Channel in Average

Description Enables/disables channels to be in average mode.

Syntax **$AAEvvvv(cr)**

$ is a delimiter character.

AA (range 00-FF) represents the 2-character hexadecimal address of module.

E is the Enable/disable channels to be in average command.

vvvv are four hexadecimal values. The values are interpreted by the module as four binary words (4-bit). The first word represents the status of channel 12~15, and the second word represents the status of channel 8~11…etc.
Value 0 means enable channel to be in average, value 1 means disable channel to be not in average.

(cr) is the terminating character, carriage return (0Dh).

Response **!AA(cr)** if the command is valid or **?AA (cr)** if the command is invalid.

There is no response if the module detects a syntax error or communication error.

! delimiter character indicates a valid command was received.

? delimiter character indicates the command was invalid.

AA (range 00-3F) represents the 2-character hexadecimal address of module.

(cr) is the terminating character, carriage return (0Dh).

Example Command: **$01E0103(cr)**

Response: **!01(cr)**

Hexadecimal **0103** equals binary 0000 0001 0000 0011, which enables channel 0, 1 and 8 to be in average mode only

10.16 #AAAL Read AD high/low Alarm Status

Description The command will return the alarm status from all channels of a specified (AA) module.

Syntax **#AAAL(cr)**

is a delimiter character.

AA (range 00-FF) represents the 2-character hexadecimal address of the analog input module.

AL identifies Read Ad high/low Alarm Status command

(cr) is the terminating character, carriage return (0Dh).

Response **!AA(hhhh)(llll)(cr)** if the command is valid or **?AA (cr)** if the command is invalid

There is no response if the module detects a syntax error or communication error.

! is a delimiter character.

Hhhh are four hexadecimal values. The values are interpreted by the module as four binary words (4-bit). The first word represents the status of channel 12~15, and the second word represents the status of channel 8~11…etc.
bit x=0 means the channel x is high alarm, bit x= 1 means the channel x isn't high alarm.

(cr) is the terminating character, carriage return (0Dh).

llll are four hexadecimal values. The values are interpreted by the module as four binary words (4-bit). The first word represents the status of channel 12~15, and the second word represents the status of channel 8~11…etc.
bit x=0 means the channel x is low alarm, bit x= 1 means the channel x isn't low alarm.

Example Command: **#01AL(cr)**

Response: **!0100010010(cr)**

The command requests the analog input module at address 12h to return thethe alarm status of all channels.

The analog input module responds that the alarm value of all channel s is equal to 00010010 (hex)

The high alarm status=0001 means the channel 0 is high alarm

The low alarm status=0010 means the channel 4 is low alarm

10.17 $AAAHnnnn Clear A/D High Alarm

Description Clear A/D High Alarm status (over range status).

Syntax **$AAAHnnnn (cr)**

$ is a delimiter character.

AA (range 003F) represents the 2-character hexadecimal address of module.

H is the clear high alarm command.

nnnn are four hexadecimal values. The values are interpreted by the module as four binary words (4-bit). The first word represents channel 12~15, and the second word represents channel 8~11…etc. bit x=1 clear high alarm status of channel x, bit x=0 no clear

(cr) is the terminating character, carriage return (ODh).

Response

!AA(cr) if the command is valid or **?AA (cr)** if the command is invalid.

There is no response if the module detects a syntax error or communication error.

! is a delimiter character indicating a valid command was received.

AA (range 00-3F) represents the 2-character hexadecimal address of module.

(cr) is the terminating character, carriage return

Examples Command: **$01H0101**(cr)

Response: **!01 (cr)**

Clear high alarm status of channel 0 and 4

10.18 $AAALnnnn Clear A/D Low Alarm

Description Clear A/D High Alarm status (under range status).

Syntax **$AAALnnnn (cr)**

$ is a delimiter character.

AA (range 003F) represents the 2-character hexadecimal address of module.

L is the clear high alarm command.

nnnn are four hexadecimal values. The values are interpreted by the module as four binary words (4-bit). The first word represents channel 12~15, and the second word represents channel 8~11…etc. bit x=1 clear low alarm status of channel x, bit x=0 no clear

(cr) is the terminating character, carriage return (ODh).

Response

!AA(cr) if the command is valid or **?AA (cr)** if the command is invalid.

There is no response if the module detects a syntax error or communication error.

! is a delimiter character indicating a valid command was received.

AA (range 00-3F) represents the 2-character hexadecimal address of module.

(cr) is the terminating character, carriage return

Examples Command: **$01L0101**(cr)

Response: **!01 (cr)**

Clear low alarm status of channel 0 and 4

10.19 $AAB Read Channel Burnout Status

Description Read channel burn out status

Syntax **$AAB(cr)**

$ is a delimiter character.

AA (range 00-3F) represents the 2-character hexadecimal address of module.

B is the Channel Diagnose command.

(cr) is the terminating character, carriage return (0Dh).

Response **!AAnnnn(cr)** if the command is valid

?AA(cr) if an invalid command was issued.

There is no response if the module detects a syntax error or communication error.

! delimiter character indicates a valid command was received.

? delimiter character indicates the command was invalid.

AA (range 00-3F) represents the 2-character hexadecimal address of the module.

Nnnn (range 0000-FFFF) is a hexadecimal number that equals the 16-bit parameter, representing the status of analog input channels. Bit value 0 means normal status; and bit value 1 means channel open wiring.

(cr) is the terminating character, carriage return (0Dh)

Examples Command: **$01B(cr)**

Response: **!010101(cr)**

Channel 0,8 are open wiring and channel 1~7 and 9~15 are all normal.

10.20 %AAB Read Channel Burnout Enable/Disable Status

Description Read channel burnout detection enables/disables status of a specified input module.

Syntax **%AAB(cr)**

% is a delimiter character.

AA (range 00-3F) represents the 2-character hexadecimal address of module.

B is the Enable/disable burnout command.

(cr) is the terminating character, carriage return **(0Dh)**.

Response **!AAbb(cr)** if the command is valid or **?AA (cr)** if the command is invalid.

There is no response if the module detects a syntax error or communication error.

! delimiter character indicates a valid command was received.

? delimiter character indicates the command was invalid.

AA (range 00-3F) represents the 2-character hexadecimal address of module.

B burnout enable/disable status, 0: disable, 1: enable

(cr) is the terminating character, carriage return **(0Dh).**

Example Command: **%01B(cr)** read burnout detection enable/disable status of specified module

Response: **!001(cr)** burnout detection is enabled

Example Command: **%01B(cr)** read burnout detection enable/disable status of specified module

Response: **!001(cr)** burnout detection is enabled

10.21 %AABn Enable/disable burnout detection

Description Enables/disables channel burnout detection of a specified input module.

Syntax **%AABn(cr)**

% is a delimiter character.

AA (range 00-3F) represents the 2-character hexadecimal address of module.

B is the Enable/disable burnout command.

n represents enable or disable burnout, value 1 means enable burnout detection, value 0 means disable burnout detection.

(cr) is the terminating character, carriage return **(0Dh)**.

Response **!AA(cr)** if the command is valid or **?AA (cr)** if the command is invalid.

There is no response if the module detects a syntax error or communication error.

! delimiter character indicates a valid command was received.

? delimiter character indicates the command was invalid.

AA (range 00-3F) represents the 2-character hexadecimal address of module.

(cr) is the terminating character, carriage return (0Dh).

Example Command: **%00B1(cr)** this command enable burnout detection of specified module

Response: **!00(cr)** this command disable burnout detection of specified module

Example Command: **%00B0(cr)**

Response: **!00(cr)**

10.22 $AA3 Read the CJC Temperature

Description Read cold junction temperature.

Syntax **$AA3(cr)**

$ is a delimiter character.

AA (range 00-3F) represents the 2-character hexadecimal address of module.

3 is the Read cold junction temperature command.

(cr) is the terminating character, carriage return (0Dh).

Response **>DATA(cr)** if the command is valid or **?AA (cr)** if the command is invalid.

There is no response if the module detects a syntax error or communication error.

> delimiter character indicates a valid command was received.

? delimiter character indicates the command was invalid.

DATA CJC temperature in degrees Celsius, consisting of of a sign byte, '+' or '-' and followed by 5 decimal digits

with a fixed decimal point in tenth of a degree

(cr) is the terminating character, carriage return (0Dh).

Example Command: **$043(cr)**

Response: **>+0030.2(cr)**

The command asks the analog input module at address 04h to send its cold junction temperature data.

The module responds with +0030.2C.

10.23 ~AAC Read the CJC Enable/disable

Description read cold junction compensation enable/disable status.

Syntax **~AAC(cr)**

~ is a delimiter character.

AA (range 00-3F) represents the 2-character hexadecimal address of module.

C read CJC enable/disable status command.

(cr) is the terminating character, carriage return (ODh).

Response **!AAn** if the command is valid or **?AA (cr)** if the command is invalid.

There is no response if the module detects a syntax error or communication error.

! is a delimiter character indicating a valid command was received.

AA (range 00-3F) represents the 2-character hexadecimal address of module.

N n=1 if CJC enabled, n=0 if CJC disabled

(cr) is the terminating character, carriage return

Examples Command: **~01C(cr)**

Response:**!011(cr)**

CJC for all channels is enabled

10.24 ~AACn Enable/Disable the CJC

Description enable/disable cold junction compensation.

Syntax **~AACn(cr)**

~ s a delimiter character.

AA range 00-3F) represents the 2-character hexadecimal address of module.

C s the enable/disable CJC command.

N =0 disable CJC, n=1 enable CJC

(cr) s the terminating character, carriage return (ODh).

Response **!AA** if the command is valid or **?AA (cr)** if the command is invalid.

There is no response if the module detects a syntax error or communication error.

! s a delimiter character indicating a valid command was received.

AA range 00-FF) represents the 2-character hexadecimal address of module.

(cr) s the terminating character, carriage return

Examples Command: **~01C1(cr)**

Response: **!01(cr)**

nable cold junction compensation

10.25 $AA9snnnn Set the all channel CJC Offset

Description set all channels to have the same cold junction offset.

Syntax **$AA9snnnn(cr)**

$ is a delimiter character.

AA (range 00-3F) represents the 2-character hexadecimal address of module.

9 is the set cold junction offset command.

S sign of cold junction offset

nnnn cold junction offset (Hex) in 0.01C unit

(cr) is the terminating character, carriage return (ODh).

Response **!AA** if the command is valid or **?AA (cr)** if the command is invalid.

There is no response if the module detects a syntax error or communication error.

! is a delimiter character indicating a valid command was received.

AA (range 00-3F) represents the 2-character hexadecimal address of module.

(cr) is the terminating character, carriage return

Examples Command: **$019+0010(cr)**

Response: **!01(cr)**

Set all channels to have the same cold junction offset to +0010(Hex)*0.01=+0.16C.

10.26 $AA9c Read single channel CJC Offset

Description read cold junction offset of specified channel

Syntax **$AA9c(cr)**

$ is a delimiter character.

AA (range 00-3F) represents the 2-character hexadecimal address of module.

9 is the set cold junction offset command.

C channel number

(cr) is the terminating character, carriage return (ODh).

Response **!AAsnnnn** if the command is valid or **?AA (cr)** if the command is invalid.

There is no response if the module detects a syntax error or communication error.

! is a delimiter character indicating a valid command was received.

AA (range 00-3F) represents the 2-character hexadecimal address of module.

S sign of cold junction offset

nnnn cold junction offset in 0.01C unit

(cr) is the terminating character, carriage return

Examples Command: **$0192(cr)**

Response: **!01+0010(cr)**

The cold junction offset of channel 2 is +0010(Hex)*0.01=+0.16C.

Response: **!AA** if the command is valid or **?AA (cr)** if the command is invalid.

There is no response if the module detects a syntax error or communication error.

! is a delimiter character indicating a valid command was received.

AA (range 00-3F) represents the 2-character hexadecimal address of module.

(cr) is the terminating character, carriage return

Examples Command: **$019+0010(cr)**

Response: **!01(cr)**

Set all channels to have cold junction offset to +0010(Hex)*0.01=+0.16C.

10.27 $AA9cSnnnn Set single channel CJC Offset

Description set channel cold junction offset individually

Syntax **$AA9csnnnn(cr)**

$ is a delimiter character.

AA (range 00-3F) represents the 2-character hexadecimal address of module.

9 is the set cold junction offset command.

c is the channel number (0~F)

s sign of cold junction offset

nnnn cold junction offset (Hex) in 0.01C unit

(cr) is the terminating character, carriage return (ODh).

Response **!AA** if the command is valid or **?AA (cr)** if the command is invalid.

There is no response if the module detects a syntax error or communication error.

! is a delimiter character indicating a valid command was received.

AA (range 00-3F) represents the 2-character hexadecimal address of module.

(cr) is the terminating character, carriage return

Examples Command: **$0193+0010(cr)**

Response: **!01(cr)**

Set cold junction offset to +0010(Hex)*0.01=+0.16C to channel 3

10.28 $AAR Read AD Filter Value

Description Read A/D cutoff frequency

Syntax **$AAR(cr)**

$ is a delimiter character.

AA (range 00-3F) represents the 2-character hexadecimal address of module.

R read A/D cutoff frequency command.

(cr) is the terminating character, carriage return (ODh).

Response **!AAn** if the command is valid or **?AA (cr)** if the command is invalid.

There is no response if the module detects a syntax error or communication error.

! is a delimiter character indicating a valid command was received.

AA (range 00-3F) represents the 2-character hexadecimal address of module.

n 0: 50Hz, 1:60Hz, 2:100Hz, 3:120Hz

(cr) is the terminating character, carriage return

Examples Command: **$01R(cr)**

Response: **!011(cr)**

A/D cutoff frequency is 60Hz

10.29 $AARf Set AD Filter Value

Description Set A/D cutoff frequency

Syntax **$AAR(cr)**

$ is a delimiter character.

AA (range 00-3F) represents the 2-character hexadecimal address of module.

R read A/D cutoff frequency command.

f 0: 50Hz, 1:60Hz, 2:100Hz, 3:120Hz

(cr) is the terminating character, carriage return (ODh).

Response **!AAn** if the command is valid or **?AA (cr)** if the command is invalid.

There is no response if the module detects a syntax error or communication error.

! is a delimiter character indicating a valid command was received.

AA (range 00-3F) represents the 2-character hexadecimal address of module.

(cr) is the terminating character, carriage return

Examples Command: **$01R0(cr)**

Response: **!011(cr)**

Set A/D cutoff frequency to 560Hz

10.30 $AA6 Read the Channel Enable/Disable Status

Description Read the status of digital input/output channels

Syntax **$AA6(cr)**

$ is a delimiter character.

AA represents the 2-character hexadecimal module address

6 is the Digital Data In command.

(cr) is the terminating character, carriage return (0Dh)

Response **!AA00(data1)(data2)(cr)** if the command is valid.

?AA(cr) if an invalid operation was entered.

! delimiter indicating a valid command was received.

? delimiter indicating the command was invalid.

AA represents the 2-character hexadecimal module address of an EDAM-5000 module.

(data1) an 8-characters hexadecimal value representing the values of the digital input channels.

(data2) an 8-characters hexadecimal value representing the values of the digital output channels.

(cr) is the terminating character, carriage return (0Dh)

Example Read the values of all DI/DO channels

command: **$016(cr)**

Response: !01**0000000F000000FD**(cr)

The 4~ 11 characters (**0000000F**) indicate DI 0~3 channels are active, and DI 04~31 channels are inactive

The 12~ 19 characters (**000000FD**) indicate DO 0,2,3,4,5,6,7 channels are active, and 1, 4, 8~31 channels are inactive

10.31 $AA5vvvv Enable/Disable A/D Channels

Description Enables/disables multiplexing simultaneously for separate channels of a specified input module.

Syntax **$AA5vvvv(cr)**

$ is a delimiter character.

AA (range 00-FF) represents the 2-character hexadecimal address of module.

5 is the Enable/disable Channels command.

vvvv are four hexadecimal values. The values are interpreted by the module as four binary words (4-bit). The first word represents the status of channel 4~7, and the second word represents the status of channel 0~3…etc. Value 0 means the channel is disabled, value 1 means the channel is enabled.

(cr) is the terminating character, carriage return (0Dh).

Response **!AA(cr)** if the command is valid or **?AA (cr)** if the command is invalid.

There is no response if the module detects a syntax error or communication error.

! delimiter character indicates a valid command was received.

? delimiter character indicates the command was invalid.

AA (range 00-FF) represents the 2-character hexadecimal address of module.

(cr) is the terminating character, carriage return (0Dh).

Example Command: **$0058100(cr)**

Response: **!00(cr)**

Hexadecimal 8100 equals binary 1000 0001 0000 0000, which enables channel 8,15 and disables channels 0,1,2,3,4, 5, 6,7, and 9,10,11,12,13, and 14..

10.32 $AA8Ci Read the Single A/D Channel Range

Description The command read individual channel type.

Syntax **$AA8Ci (cr)**

$ is a delimiter character.

AA (range 00-FF) represents the 2-character hexadecimal address of module.

8C is the read channel type command.

i channel number

(cr) is the terminating character, carriage return (ODh).

Response **!AACiRrr(cr)** if the command is valid or **?AA (cr)** if the command is invalid.

There is no response if the module detects a syntax error or communication error.

! is a delimiter character indicating a valid command was received.

AA (range 00-3F) represents the 2-character hexadecimal address of module.

i channel number(0~F)

rr type of channel **i**

(cr) is the terminating character, carriage return

Examples Command: **$018C3**(cr)

Response: **!01C3R08(cr)**

The type code of channel 3 is 08 (+/-10V).

10.33 $AA7CiRrr Set the Single Channel Range

Description The command set channel type individually.

Syntax **$AA7CiRrr(cr)**

$ is a delimiter character.

AA (range 00-FF) represents the 2-character hexadecimal address of module.

7C is the Set channel type command.

i channel number

rr channel type code

(cr) is the terminating character, carriage return (ODh).

Response **!AA** if the command is valid or **?AA (cr)** if the command is invalid.

There is no response if the module detects a syntax error or communication error.

! is a delimiter character indicating a valid command was received.

AA (range 00-FF) represents the 2-character hexadecimal address of module.

(cr) is the terminating character, carriage return

Examples Command: **$017C3R08(cr)**

Response: **!01(cr)**

Set type code 08 (+/-10V) to channel 3.

10.34 $AAS1 Reload the Default configuration

Description Reloads the Default configuration

Syntax **$AAS1(cr)**

$ is a delimiter character.

AA (range 00-FF) represents the 2-character hexadecimal address of module.

is the Reloads the Default configuration command.

(cr) is the terminating character, carriage return (0Dh).

Response **!AA(cr)** if the command is valid or **?AA (cr)** if the command is invalid.

There is no response if the module detects a syntax error or communication error.

! delimiter character indicates a valid command was received.

? delimiter character indicates the command was invalid.

AA (range 00-3F) represents the 2-character hexadecimal address of module.

(cr) is the terminating character, carriage return (0Dh).

Example Command: **$01S1(cr)**

Response: **!01(cr)**

Reloads the Default configuration

10.35 @AA Read the Digital I/O Status

Description: ead the status of its digital input and digital output channels

Syntax: **AA(cr)**

@ is a delimiter character.

AA (range 00-3F) represents the 2-character hexadecimal Modbus network address (Always 01)

(cr) is the terminating character, carriage return (0Dh)

Response: **>**(DI data)(DO data)(cr) if the command is valid.

?AA(cr) if an invalid operation was entered.

There is no response if the module detects a syntax error or communication error or if the address does not exist.

> delimiter indicating a valid command was received.

? delimiter indicating the command was invalid.

AA (range 00-3F) represents the 2-character hexadecimal Modbus network address of an EDAM-9000 module.

(DI data) 4-character hexadecimal value representing the values of the digital input module.

(DO data) 4-character hexadecimal value representing the values of the digital output module.

(cr) is the terminating character, carriage return (0Dh)

Example: Command: **@01(cr)**

Response: **>0000102000210001(cr)**

00001020=the status of digital input channels. DI channels 4,15 are active, and other channels are inactive

00210001=the status of digital output channels. DO channels 0,16,18 are active, and other channels are inactive

10.36 @AAnn Set the Digital Output Channels

Description: Sets the Digital Output Channels

Command: **@AAnn(cr)**

Syntax:

@ Command leading code

AA Module address ID (00-3F)

nn Output value to channel 0~7

(cr) is the terminating character, carriage return (0Dh)

Response:

!AA(cr) Valid command

?AA(cr) Invalid command

(cr) is the terminating character, carriage return (0Dh)

Example:

Command: **@0523(cr)**

Response: **!05(cr)**

! Valid command

05 Module ID

23 Set 23(Hex)to Digital output channels (channel 0,1,5 are active, other channels are inactive)

10.37 @AAnnnn Set the Digital Output Channels

Description: Sets the value to Digital Output Channels 0~15

Command: **@AAnnnn(cr)**

Syntax:

@ Command leading code

AA Module address ID (00-3F)

nnnn Output value to channel 0~15

(cr) is the terminating character, carriage return (0Dh)

Response:

!AA(cr) Valid command

?AA(cr) Invalid command

(cr) is the terminating character, carriage return (0Dh)

Example:

Command: **@050023(cr)**

Response: **!05(cr)**

! Valid command

05 Module ID

0023 Set 23(Hex) to Digital output channels (channel 0,1,5 are active, and channel 2,3,5,6,7.8.9.10.11.12.13.14.15 are inactive)

10.38 @AAnnnnnn Set the Digital Output Channels

Description: Sets the Digital Output Channels (0~23)

Command: **@AAnnnnnn(cr)**

Syntax:

@ Command leading code

AA Module address ID (00-3F)

nnnnnn Output value to channel 0~23

(cr) is the terminating character, carriage return (0Dh)

Response:

!AA(cr) Valid command

?AA(cr) Invalid command

(cr) is the terminating character, carriage return (0Dh)

Example:

Command: **@05010323(cr)**

Response: **!05(cr)**

! Valid command

05 Module ID

010323 Set 23(Hex)to Digital output channels (channel 0,1,5,8,9,16 are active, other channels are inactive)

10.39 #AA0Ann Set the Digital 1's byte(DO0~DO7) Output

Description: Sets the value to Digital Output Channels 0~7

Command: **#AA0Ann(cr)**

Syntax:

Command leading code

AA Module address ID (00-3F)

0A is the Sets the Digital Output lowest byte(DO0~DO7) command

nn Output value to channel 0~7

(cr) is the terminating character, carriage return (0Dh)

Response:

!AA(cr) Valid command

?AA(cr) Invalid command

(cr) is the terminating character, carriage return (0Dh)

Example:

Command: **#050A23(cr)**

Response: **!05(cr)**

! Valid command

05 Module ID

23 Set 23(Hex)to Digital output channels (channel 0,1,5 are active, and channel 2,3,4,6,7 are inactive)

10.40 #AA0Bnn Set the Digital 2's byte(DO8~DO15) Output

Description: Sets the value to Digital Output Channels 8~15

Command: **#AA0Bnn(cr)**

Syntax:

Command leading code

AA Module address ID (00-3F)

0B is the Sets the Digital Output lowest byte(DO8~DO15) command

nn Output value to channel 8~15

(cr) is the terminating character, carriage return (0Dh)

Response:

!AA(cr) Valid command

?AA(cr) Invalid command

(cr) is the terminating character, carriage return (0Dh)

Example:

Command: **#050B23(cr)**

Response: **!05(cr)**

! Valid command

05 Module ID

23 Set 23(Hex)to Digital output channels (channel 8,9,13 are active, and channel 10,11,12,14,15 are inactive)

10.41 #AA0Cnn Set the Digital 3's byte(DO16~DO23) Output

Description: Sets the value to Digital Output Channels 16~23

Command: **#AA0Cnn(cr)**

Syntax:

Command leading code

AA Module address ID (00-3F)

0C is the Sets the Digital Output lowest byte(DO16~DO23) command

nn Output value to channel 16~23

(cr) is the terminating character, carriage return (0Dh)

Response:

!AA(cr) Valid command

?AA(cr) Invalid command

(cr) is the terminating character, carriage return (0Dh)

Example:

Command: **#050C23(cr)**

Response: **!05(cr)**

! Valid command

05 Module ID

23 Set 23(Hex)to Digital output channels (channel 16,17,21 are active, and channel 18,19,21,22,23 are inactive)

10.42 #AAnn Read digital input counter

Description Read DI latch status.

Syntax **#AAnn(cr)**

is a delimiter character.

AA represents the 2-character hexadecimal Modbus address (Always 01)

nn represents DI channel number .

(cr) is the terminating character, carriage return (0Dh)

Response **!AAnnnnnnnn(cr)** if the command is valid.

?AA(cr) if an invalid operation was entered.

! delimiter indicating a valid command was received.

? delimiter indicating the command was invalid.

AA represents the 2-character hexadecimal module address of an EDAM-5000 module.

nnnnnnnn represents 4-bytes counter value

(cr) is the terminating character, carriage return (0Dh)

Example Read DI latch status

command: #01**02**(cr)

Response !01**00000003**

latch status= **00000003**, DI #2 counter value=3

10.43 $AACn Clear digital input counter

Description Clear Counter of all DI Channel

Syntax **$AACn(cr)**

$ is a delimiter character.

AA represents the 2-character hexadecimal module address

C is Clear DI counter command.

n is DI channel number

(cr) is the terminating character, carriage return (0Dh).

Response **!AA(cr)** if the command is valid.

?AA(cr) if an invalid operation was entered.

! delimiter indicating a valid command was received.

? delimiter indicating the command was invalid.

AA represents the 2-character hexadecimal address of an EDAM-5000 module.

(cr) is the terminating character, carriage return (0Dh).

Example Clear DI #2 counter value

Command: $01**C2**(cr)

Response !01(cr)

10.44 $AACnn Clear digital input counter

Description Clear Counter of all DI Channel

Syntax **$AACn(cr)**

$ is a delimiter character.

AA represents the 2-character hexadecimal module address

C is Clear DI counter command.

nn is DI channel number

(cr) is the terminating character, carriage return (0Dh).

Response **!AA(cr)** if the command is valid.

?AA(cr) if an invalid operation was entered.

! delimiter indicating a valid command was received.

? delimiter indicating the command was invalid.

AA represents the 2-character hexadecimal address of an EDAM-5000 module.

(cr) is the terminating character, carriage return (0Dh).

Example Clear DI #2 counter value

command: $01**C02**(cr)

Response !01(cr)

10.45 $AALS Read the latched DI status

Description Read DI latch status.

Syntax **$AALS(cr)**

$ is a delimiter character.

AA represents the 2-character hexadecimal Modbus address (Always 01)

LS represents read DI latch status command.

(cr) is the terminating character, carriage return (0Dh)

Response **!AAnnnnnnnn(cr)** if the command is valid.

?AA(cr) if an invalid operation was entered.

! delimiter indicating a valid command was received.

? delimiter indicating the command was invalid.

AA represents the 2-character hexadecimal module address of an EDAM-5000 module.

(cr) is the terminating character, carriage return (0Dh)

Example Read DI latch status

command: $01**LS**(cr)

Response !01**00000003**

latch status= **00000003**, DI #0 latched, DI #1 latched, and DI #2 ~ DI #11 no latched

10.46 $AAC Clear the latched DI status

Description Clear all digital input counter of specified DI channel.

Syntax **$AAC(cr)**

$ is a delimiter character.

AA represents the 2-character hexadecimal module address

CL is clear latch command.

(cr) is the terminating character, carriage return (0Dh).

Response **!AA(cr)** if the command is valid.

?AA(cr) if an invalid operation was entered.

! delimiter indicating a valid command was received.

? delimiter indicating the command was invalid.

AA represents the 2-character hexadecimal address of an EDAM-5000 module.

(cr) is the terminating character, carriage return (0Dh).

Example Clear all latch status

command: $01**C**(cr)

Response !01(cr)

10.47 $AA9nn Read Single Do Pulse High/Low Width

Description: Read Do Pulse High/Low Width of specified DO channel

Command: **$AA9nn(cr)**

Syntax:

$ Command leading code
AA Module address ID (00-3F)
9 is the Read Do Pulse High/Low Width command
nn Digital Output channel number (0~17)
(cr) is the terminating character, carriage return (0Dh)

Response:

!AAhhhhvvvv(cr) Valid command
hhhh=high pulse width in 0.5msec, vvvv=low pulse width in 0.5msec
?AA(cr) Invalid command
(cr) is the terminating character, carriage return (0Dh)

Example:

Command: **$05902(cr)**
Response: **!0502000100(cr)**
! Valid command
05 Module ID
0200 high pulse width=0200(hex)=512(dec)*0.5msec=256 msec
0100 low pulse width=0100(hex)=256(dec)*05msec=128 msec

10.48 $AA9nnhhhhllll Set Single Do Pulse High/Low Width

Description: set Do Pulse High/Low Width of specified DO channel

Command: **$AA9nnhhhhvvvv(cr)**

Syntax:

$ Command leading code
AA Module address ID (00-3F)
9 is the Read Do Pulse High/Low Width command
nn Digital Output channel number (0~17)
hhhh high pulse width in 0.5msec
vvvv low pulse width in 0.5msec
(cr) is the terminating character, carriage return (0Dh)

Response:

!AAcr) Valid command
?AA(cr) Invalid command
(cr) is the terminating character, carriage return (0Dh)

Example:

Command: **$05902000100(cr)**
Response: **!05(cr)**
! Valid command
05 Module ID
0200 high pulse width=0200(hex)=512(dec)*0.5msec=256 msec
0100 low pulse width=0100(hex)=256(dec)*05msec=128 msec

10.49 $AAAnn Read Single Do High/Low Delay Width

Description: Read Do High/Low output Delay time of specified DO channel

Command: **$AAAnn(cr)**

Syntax:

$ Command leading code
AA Module address ID (00-3F)
A is the Read Do High/Low Delay time command
nn Digital Output channel number (0~17)
(cr) is the terminating character, carriage return (0Dh)

Response:

!AAhhhhvvvv(cr) Valid command
hhhh high delay time in 0.5msec
vvvv low delay time in 0.5msec
?AA(cr) Invalid command
(cr) is the terminating character, carriage return (0Dh)

Example:

Command: **$05A02(cr)**
Response: **!0502000100(cr)**
! Valid command
05 Module ID
0200 high output delay time=0200(hex)=512(dec)*0.5msec=256 msec
0100 low low output delay time=0100(hex)=256(dec)*05msec=128 msec

10.50 $AAAnnhhhhllll Set Single Do High/Low Delay Width

Description: set Do High/Low output Delay time of specified DO channel

Command: **$AAAnnhhhhvvvv(cr)**

Syntax:

$ Command leading code
AA Module address ID (00-3F)
A is the Read Do Pulse High/Low Width command
nn Digital Output channel number (0~17)
hhhh high output delay time in 0.5msec
vvvv low output delay time in 0.5msec
(cr) is the terminating character, carriage return (0Dh)

Response:

!AAcr) Valid command
?AA(cr) Invalid command
(cr) is the terminating character, carriage return (0Dh)

Example:

Command: **$05A02000100(cr)**
Response: **!05(cr)**
! Valid command
05 Module ID
0200 high output delay time=0200(hex)=512(dec)*0.5msec=256 msec
0100 low output delay time=0100(hex)=256(dec)*05msec=128 msec

10.51 $AABnn Read Single Do Pulse Counts

Description Read Pulse Counts of single DO channel

Syntax **$AABnn(cr)**

$ is a delimiter character.

AA represents the 2-character hexadecimal module address

B is read pulse counts command.

nn represents DO channel number

(cr) is the terminating character, carriage return (0Dh).

Response **!AAcccc(cr)** if the command is valid.

?AA(cr) if an invalid operation was entered.

! delimiter indicating a valid command was received.

? delimiter indicating the command was invalid.

AA represents the 2-character hexadecimal address of an EDAM-5000 module.

cccc represents DO pulse counts

(**cr**) is the terminating character, carriage return (0Dh).

Example Read pulse counts of DO channel 03

command: $01**B03**(cr)

Response !01**0020**(cr)

The pulse counts of DO channel 03 is 0020(hex)=32(dec)

10.52 #AA2nncccc Write Single Do Pulse Counts

Description Set Pulse Counts of Single DO channel

Syntax **#AA2nncccc(cr)**

is a delimiter character.

AA represents the 2-character hexadecimal module address

2 is set DO pulse counts command.

nn is DO channel number

cccc represents DO pulse output counts

(cr) is the terminating character, carriage return (0Dh).

Response **!AA(cr)** if the command is valid.

?AA(cr) if an invalid operation was entered.

! delimiter indicating a valid command was received.

? delimiter indicating the command was invalid.

AA represents the 2-character hexadecimal address of an EDAM-5000 module.

(cr) is the terminating character, carriage return (0Dh).

Example Set DO pulse output counts=32 of DO channels 5

command: #01**2050020**(cr)

Response !01(cr)

10.53 #AA3nns Start/Stop DO Pulse Counts

Description Start/stop DO pulse output of single DO channel

Syntax **#AA3nns(cr)**

is a delimiter character.

AA represents the 2-character hexadecimal module address

3 is Start/stop DO pulse command.

nn is DO channel number

s s=0 represents stop pulse output, s=1 represents start pulse output

(cr) is the terminating character, carriage return (0Dh).

Response **!AA(cr)** if the command is valid.

?AA(cr) if an invalid operation was entered.

! delimiter indicating a valid command was received.

? delimiter indicating the command was invalid.

AA represents the 2-character hexadecimal address of an EDAM-5000 module.

(cr) is the terminating character, carriage return (0Dh).

Example Start pulse output of DO channels 5

command: #01**3051**(cr)

Response !01(cr)

10.54 ~AA4v Read the Power On/Safe Value

Description Read power-on/safe value

Syntax **~AA4v(cr)**

~ Command leading code

AA Module address ID (00 to FF)

4 Command to set DO power-on/safe value

v **v**="P" set power-on value, **v**="S" set safe value

(cr) Carriage return

Response **!AA(cr)** if the valid command

? AA(cr) Invalid command

! Delimiter for valid command

? Delimiter for invalid command

(cr) Carriage return

Example Read power-on/safe value (ID=05)

Command: ~054**P**(cr)

Response !05**00000014**(cr) //power-on value=00000014 (DO channel 2,4 on)

10.55 ~AA5v Set current Do value as power on/safe value

Description et current DO value as power-on/safe value

Syntax **~AA5v(cr)**

~ Command leading code

AA Module address ID (00 to FF)

5 Command to set DO power-on/safe value

v **v**="P" set power-on value, **v**="S" set safe value

(cr) Carriage return

Response **!AA(cr)** if the valid command

? AA(cr) Invalid command

! Delimiter for valid command

? Delimiter for invalid command

(cr) Carriage return

Example Set current DO value as power-on/safe value (ID=04)

Command: ~045**P**(cr)

Response !04(cr)

10.56 ~AA5vnnnnnn Set specified value as power on/safe value

Description **s**et DO power-on/safe value

Syntax **~AA5vnnnnnn(cr)**

~ Command leading code

AA Module address ID (00 to FF)

5 Command to set DO power-on/safe value

v **v**="P" set power-on value, **v**="S" set safe value

nnnnnn represents DO power-on/safe value

(cr) Carriage return

Response **!AA(cr)** if the valid command

? AA(cr) Invalid command

! Delimiter for valid command

? Delimiter for invalid command

(cr) Carriage return

Example Set DO power-on value to 0xFF00FF(ID=04)

Command: ~045**PFF00FE**(cr)

Response !04(cr)

Example Set DO safe value to 0xFF00FF(ID=04)

Command: ~045**SFF00FE**(cr)

Response !04(cr)

10.57 ~AAD Read DI/O active state

Description Read input/output active status.

Syntax **~AAD(cr)**

~ Command leading code

AA Module address ID (00 to FF)

D Command for reading digital input active status

(cr) Carriage return

Response **!AAmn(cr)** if the valid command

? AA(cr) if the invalid command

! Delimiter for valid command

? Delimiter for invalid command

m Input active status, m=0 input low voltage/open active, m=1 Input high voltage active (See *)

n Output active status, n=1 output short/close active, n=0 open active (See **)

(cr) Carriage return

Example Read output active status of EDAM5060 (ID=05)

Command: ~05D(cr)

Response !05**01**(cr)

01 All input channels are low active and all output channels are short/close active

Note:

(*):m is only available for the module which has digital input channels

(**):n is only available for the module which has digital output channels

10.58 ~AADvn Set DI/O active state

Description: Set input/output active status.

Command: **~AADvn[CHK](cr)**

Syntax:

~ Command leading code

AA Module address ID (00 to FF)

D Command for setting digital input active status

v Input active status, v=0 input low voltage/open active, v=1 Input high voltage active (See *)

n Output active status, n=1 output short/close active, n=0 open active (See **)

(cr) is the terminating character, carriage return (0Dh)

Response:

!AA (cr) Valid command

?AA(cr) Invalid command

(cr) is the terminating character, carriage return (0Dh)

Example:

Command: **~05D01(cr)**

Response: **!05(cr)**

05 Module ID

0 Set all inputput channels to high volt/open active

1 Set all output channels to high/open output active

Note:

(*) :v is only available for the module which has digital input channels

(**) :n is only available for the module which has digital output channels

Chapter 11 E5KDAQ.DLL API

11.1 Common functions

Function name	description	Sec.
E5K_SearchModules	Search all EDAM5000 modules	11.4
E5K_OpenModuleUSB	Open module with ID address from USB interface	11.5
E5K_OpenModuleIP	Open module with the module IP address from Ethernet	11.6
E5K_OpenModuleCOM	Open module with ID address from COM port	11.7
E5K_CloseModules	Close all modules	11.8
E5K_GetDLLVersion	Get E5KDAQ.DLL version	11.9
E5K_VerifyPassWord	Verifies Pass word (Ethernet only)	11.10
E5K_ChangePassWord	Change password (Ethernet connection only)	11.11
E5K_GetLastErrorCode	Get last DLL error code	11.12
E5K_SetRXTimeOutOption	Set receive/send Timeout	11.13
E5K_StartAlarmEventIP	Start alarm event from IP (Ethernet only)	11.14
E5K_StopAlarmEventIP	Stop alarm event from IP (Ethernet only)	11.15
E5K_StartAlarmEventUSB	Start alarm event from USB (USB only)	11.16
E5K_StopAlarmEventUSB	Stop alarm event from USB (USB only)	11.17
E5K_ReadAlarmEventData	Read alarm event data	11.18
E5K_StartStreamEvent	Start stream event from IP (Ethernet only)	11.19
E5K_StopStreamEvent	Stop stream event from IP (Ethernet only)	11.20
E5K_ReadStreamEventData	Read stream data from IP (Ethernet only)	11.21
E5K_ReadModuleConfig	Read module configuration	11.22
E5K_SetModuleConfig	Set module configuration	11.23
E5K_WriteModbusDiscrete	Write Modbus discrete(coil/input) data to the specified module	11.24
E5K_WriteModbusRegister	Write Modbus register(holding/input) data to the specified module	11.25
E5K_ReadModbusRegister	Read Modbus (holding/input)register data from the specified module	11.26
E5K_ReadModbusDiscrete	Read Modbus (coil/input)discrete data from the specified module	11.27
E5K_SendASCRequestAndWaitResponse	Send ASCII command to and wait response from the specified module	11.28
E5K_RecvASCII	Receive ASCII data from the specified module	11.29
E5K_SendASCII	Send ASCII command to specified module	11.30
E5K_SendHEXRequestAndWaitResponse	Send binary data to and wait response from the specified module	11.31
E5K_SendHEX	Send HEX command to specified module	11.32
E5K_RecvHEX	Receive Hex data from the specified module	11.33
E5K_CalculateCRC16	Calculate CRC16	11.34
E5K_SetLEDControl	Set LED control mode	11.35
E5K_WriteDataToLED	Write data to LED board	11.36
E5K_FlashLED	Force on-board LED to flash	11.37
E5K_IsValidIPAddress	Check IP address is valid or not	11.38
E5K_GetLocalIP	Get Host IP address	11.39
E5K_TCPConnect	Build a TCP connection	10.43
E5K_TCPSendData	Send data to TCP connection	11.41
E5K_TCPRecvData	Receive data from TCP connection	11.42
E5K_TCPPing	Ping the specified IP address	11.43
E5K_TCPDisconnect	Disconnect TCP connection	11.44

11.2 Analog functions

Function name	Description	Sec.
E5K_ReadAIChannelType	Read type of the specified AI channel	11.45
E5K_SetAIChannelType	Set type of the specified AI channel	11.46
E5K_SetSingleChannelColdJunctionOffset	Set CJC offset of the specified AI channel	11.47
E5K_ReadSingleChannelColdJunctionOffset	Read CJC offset of a specified AI channel	11.48
E5K_ReadMultiChannelColdJunctionOffset	Read CJC offset of the multiple AI channels	11.49
E5K_SetMultiChannelColdJunctionOffset	Set CJC offset of multiple AI channels	11.50
E5K_ReadColdJunctionTemperature	Read cold junction temperature	11.51
E5K_ReadColdJunctionStatus	Read cold junction enable/disable status	11.52
E5K_SetColdJunction	Enable/disable CJC	11.53
E5K_ReadAIChannelConfig	Read configuration of the specified AI channel	11.54
E5K_SetAIChannelConfig	Set configuration of the specified AI channel	11.55
E5K_ReadAIBurnOutStatus	Read burnout status of analog channels	11.56
E5K_ReadAIAlarmStatus	Read AI alarm event status	11.57
E5K_SetAIBurnOut	Enable/disable burnout detection	11.58
E5K_ReadAIBurnOut	Read AI burnout detection enable/disable status	11.59
E5K_SetAIModuleFilter	Set AI filter frequency	11.60
E5K_ReadAIModuleFilter	Read AI filter frequency	11.61
E5K_SetAIChannelEnable	Enable/disable AI channel	11.62
E5K_ReadAIChannelEnable	Read enable/disable status of AI channels	11.63
E5K_ReadAINormalMultiChannel	Read AI normal value of multiple AI channels	11.64
E5K_ReadAIMaximunMultiChannel	Read AI maximum value of multiple AI channels	11.65
E5K_ReadAIMinimunMultiChannel	Read AI minimum value of multiple AI channels	11.66
E5K_ResetAIMaximun	Reset Maximum value of the specified AI channels	11.67
E5K_ResetAIMinimun	Reset Minimum value of the specified AI channels	11.68
E5K_ResetAIHighAlarm	Reset high alarm flag of the specified AI channels	11.69
E5K_ResetAILowAlarm	Reset low alarm flag of the specified AI channels	11.70
E5K_ReadAIChannelAverage	Read in average status of AI channels	11.71
E5K_SetAIChannelAverage	Enable AI channels in average	11.72

11.3 DIO functions

Function name	Description	Sec.
E5K_SetDIChannelConfig	Set configuration of the specified DI channel	11.73
E5K_ReadDIChannelConfig	Read configuration of the specified DI channel	11.74
E5K_ReadDIStatus	Read DI status	11.75
E5K_ReadDILatch	Read DI latch status	11.76
E5K_ClearAllDILatch	Clear all DI latch status	11.77
E5K_ClearSingleDICounter	Clear counter value of the specified DI channel	11.78
E5K_ReadMultiDICounter	Read multiple DI counter value	11.79
E5K_WriteDO	Write DO channels	11.80
E5K_ReadDOStatus	Read DO status	11.81
E5K_SetDOSingleChannel	Set/reset single DO channel	11.82
E5K_SetDOPulseWidth	Set DO pulse high/low width	11.83
E5K_ReadDOPulseWidth	Read DO pulse high/low width	11.84
E5K_StartDOPulse	Start DO pulse output	11.85
E5K_StopDOPulse	Stop DO pulse output	11.86
E5K_ReadDOPulseCount	Read back DO pulse count value	11.87
E5K_SetDOPowerOnValue	Set DO power on value	11.88
E5K_ReadDOPowerOnValue	Read DO power on value	11.89
E5K_ReadDIOActiveLevel	Read DI/DO active flag setting	11.90
E5K_SetDIOActiveLevel	Set DI/O channel active flag	11.91

11.4 E5K_SearchModules

Description:

Search all connected EDAM5000 modules

Syntax:

Visual Basic/VB.Net: (see E5KDAQ.bas/E5KDAQ.vb)

```
Declare/public Function E5K_SearchModules Lib "E5KDAQ.dll" (pd as E5K_DEVICE_ID_INFO,
                                interface_type as integer) as integer
```

VC++/BC++Builder: (see E5KDAQ.h)

```
unsigned short      E5K_SearchModules (E5K_DEVICE_ID_INFO *pd,unsigned int interface_type );
```

Parameters:

pd	points to a structure E5K_DEVICE_ID_INFO
interface_type	indicate what connection be used for searching (see E5KDAQ.h)

Return Code:

Return how many modules be found, It no module existed, if return with 0

11.5 E5K_OpenModuleUSB

Description:

Open module by its ID address from USB interface

Syntax

Visual Basic/VB.Net: (see E5KDAQ.bas/E5KDAQ.vb)

```
Declare/public Function E5K_OpenModuleUSB Lib "E5KDAQ.dll" (ByVal id As Integer) As Integer
```

VC++/BC++Builder: (see E5KDAQ.h)

```
unsigned short      E5K_OpenModuleUSB (unsigned short id);
```

Parameters:

Id	module ID address

Return Code:

Return the same ID number as parameter id, If open success
Return -1 open error

11.6 E5K_OpenModuleIP

Description:

Open module by IP address for Ethernet connection only

Syntax

Visual Basic/VB.Net: (see E5KDAQ.bas/E5KDAQ.vb)

```
Declare/public Function E5K_OpenModuleIP Lib "E5KDAQ.dll" _
                        ( ByVal IP As String,Byval
                          Byval ConnecttimeOut as long,
                          Byval RxTotalTimeOut as long,
                          Byval RxTimeoutInterval as long) As Integer
```

VC++/BC++: (see E5KDAQ.h)

```
unsigned short    E5K_OpenModuleIP (
                  char IP[],
                  unsigned long ConnectTimeOut,
                  unsigned long RxTotalTimeOut,
                  unsigned long RxTimeoutInterval);
```

Parameters

IP	points to a IP address array (such as "192.168.0.12")
ConnectTimeOut	Connection timeout interval (msec)
RxTotalTimeOut	Receive frame timeout interval (msec)
RxTimeOutInterval	receive character timeout interval (msec)

Return Code

Return the ID address of module, if open success

11.7 E5K_OpenModuleCOM

Description:

Open module by its ID address from COM port

Syntax

Visual Basic/VB.Net: (see E5KDAQ.bas/E5KDAQ.vb)

```
Declare/public Function E5K_OpenModuleCOM Lib "E5KDAQ.dll" _
                        ( Byval devid As Integer,
                          Byval comport As Integer,
                          ByVal RxTotalTimeOut As Long,
                          ByVal RxTimeoutInterval As Long,
                          ByVal BaudRate As Long,
                          ByVal ChksumCRC As Byte) As Integer
```

VC++/BC++Builder: (see E5KDAQ.h)

```
int         E5K_OpenModuleCOM(
            unsigned int devid,
            unsigned int comport,
            unsigned long RxTotalTimeOut,
            unsigned long RxTimeoutInterval,
            unsigned long Baudrate,
            unsigned char ChksumCRC);
```

Parameters:

devId	module ID address
comport	COM port number
RxTotalTimeOut	Receive time out between characters(msec)
RxTimeoutInterval	Receive total timeout (msec)
Baudrate	Baud rate
ChksumCRC	Enable/disable Check sum/ CRC (1=Enabled,0=disable)

Return Code:

Return the same ID number as parameter id, If open success
Return -1 open error

11.8 E5K_CloseModules

Description:

Close all modules

Syntax

Visual Basic/VB.Net: (see E5KDAQ.bas/E5KDAQ.vb)

```
Declare/public Function E5K_CloseModules Lib "E5KDAQ.dll" () As Integer
```

VC++: (see E5KDAQ.h)

```
unsigned short      E5K_CloseModules (void);
```

Parameters:

none	no parameters

Return Code:

refer to the *Error code*.

11.9 E5K_GetDLLVersion

Description:

Get version of E5KDAQ.DLL

Syntax:

Visual Basic/VB.Net: (see E5KDAQ.bas/E5KDAQ.vb)

```
Declare/public Function E5K_GetDLLVersion Lib "E5KDAQ.dll" _
                        (ByRef Major As Integer, ByRef Minor As Integer) As Integer
```

VC++: (see E5KDAQ.h)

```
unsigned short      E5K_GetDLLVersion (unsigned int *Major, unsigned int *Minor);
```

Parameters:

Major	points version major buffer
Minor	points version minor buffer

Return Code:

refer to the *Error code*.

11.10 E5K_VerifyPassWord

Description:

Verify password of the Ethernet connected module. The function should be called after calling *E5K_OpendModuleIP()* function (for Ethernet Connection only)

Syntax

Visual Basic/VB.Net: (see E5KDAQ.bas/E5KDAQ.vb)

```
Declare/public Function E5K_VerifyPassWord Lib "E5KDAQ.dll" _
                        (ByVal id As Integer,
                         ByVal PassWord As String,
                         ByVal length As Integer) As Integer
```

VC++: (see E5KDAQ.h)

```
unsigned short      E5K_VerifyPassWord (int id , UNSIGNED CHAR PassWord[], unsigned int length);
```

Parameters:

id	module ID address
PassWord	points to password string buffer
length	password length

Return Code:

refer to the *Error code*.

11.11 E5K_ChangePassWord

Description:

Change password of the Ethernet connected module. The function is available after calling E5K_VerifyPassWord function (for Ethernet Connection only)

Syntax

Visual Basic/VB.Net: (see E5KDAQ.bas/E5KDAQ.vb)

```
Declare/public Function E5K_ChangePassWord Lib "E5KDAQ.dll" _
                        (ByVal id As Integer, _
                         ByVal OldPassword As String, _
                         ByVal Oldlength As Integer, _
                         ByVal NewPassword As String, _
                         ByVal Newlength As Integer) As Integer
```

VC++: (see E5KDAQ.h)

```
unsigned short    E5K_ChangePassWord(
                  int id,
                  unsigned char oldPassWord[],
                  unsigned int oldlength,
                  unsigned char newPassWord[],
                  unsigned int newlength);
```

Parameters:

id	module ID address
oldPassWord[]	points to password string buffer
oldlength	old password length
newPassWord[]	points to new password,
newlength	new password length

Return Code:

11.12 E5K_GetLastErrorCode

Description:

Get E5KDAQ.dll last error code

Syntax

Visual Basic/VB.Net: (see E5KDAQ.bas/E5KDAQ.vb)

```
Declare/public Function E5K_GetLastErrorCode Lib "E5KDAQ.dll" () As Integer
```

VC++: (see E5KDAQ.h)

```
unsigned short    E5K_GetLastErrorCode (void);
```

Parameters:

none	no parameters

Return Code:

11.13 E5K_SetRXTimeOutOption

Description:

Set receive total timeout and interval timeout of COM port and TCP/IP

Syntax

Visual Basic/VB.Net: (see E5KDAQ.bas/E5KDAQ.vb)

```
Declare/public Function E5K_SetRXTimeOutOption Lib "E5KDAQ.dll" _
                        (ByVal RxTotalTimeout As Long, ByVal RxChrTimeOutInterval As Long) As Integer
```

VC++: (see E5KDAQ.h)

```
unsigned short      E5K_SetRXTimeOutOption (long RxTotalTimeout,long RxChrTimeOutInterval);
```

Parameters:

RxTotalTimeout receive total timeout (msec)

length receive character interval timeout(msec)

Return Code:

refer to the *Error code*.

11.14 E5K_StartAlarmEventIP

Description:

Start alarm event from IP

Syntax:

Visual Basic/VB.Net: (see E5KDAQ.bas/E5KDAQ.vb)

```
Declare/public Function E5K_StartAlarmEventIP Lib "E5KDAQ.dll" (ByVal IPaddress As string) As Integer
```

VC++: (see E5KDAQ.h)

```
Int       E5K_StartAlarmEventIP (char * IPaddress);
```

Parameters:

IPaddress Alarm event source IP address

Return Code:

refer to the *Error code*.

11.15 E5K_StopAlarmEventIP

Description:

Stop alarm event from IP

Syntax:

Visual Basic/VB.Net: (see E5KDAQ.bas/E5KDAQ.vb)

```
Declare/public Function E5K_StopAlarmEventIP Lib "E5KDAQ.dll" (ByVal IPaddress As string) As Integer
```

VC++: (see E5KDAQ.h)

```
Int     E5K_StopAlarmEventIP (char * IPaddress);
```

Parameters:

IPaddress	Alarm event source IP address

Return Code:

refer to the *Error code*.

11.16 E5K_StartAlarmEventUSB

Description:

Start alarm event from USB connection

Syntax:

Visual Basic/VB.Net: (see E5KDAQ.bas/E5KDAQ.vb)

```
Declare/public Function E5K_StartAlarmEventUSB Lib "E5KDAQ.dll" (ByVal Id As integer) As Integer
```

VC++: (see E5KDAQ.h)

```
Int     E5K_StartAlarmEventUSB (int Id);
```

Parameters:

Id	Alarm event source Id

Return Code:

refer to the *Error code*.

11.17 E5K_StopAlarmEventUSB

Description:

Stop alarm event from USB connection

Syntax:

Visual Basic/VB.Net: (see E5KDAQ.bas/E5KDAQ.vb)

```
Declare/public Function E5K_StopAlarmEventUSB Lib "E5KDAQ.dll" (ByVal Id As integer) As Integer
```

VC++: (see E5KDAQ.h)

```
Int     E5K_StopAlarmEventUSB (int Id);
```

Parameters:

Id	Alarm event source Id

Return Code:

refer to the *Error code*.

11.18 E5K_ReadAlarmEventData

Description:

Read alarm event information from device

Syntax:

Visual Basic/VB.Net: (see E5KDAQ.bas/E5KDAQ.vb)

```
Declare/public Function E5K_ReadAlarmEventData Lib "E5KDAQ.dll" (Intinfo As ALARM_EVENT_INF)
                        As Integer
```

VC++: (see E5KDAQ.h)

```
Int     E5K_ReadAlarmEventData (struct ALARM_EVENT_INF    *Intinfo);
```

Parameters:

Intinfo	Points to an Alarm event structure(ALARM_EVENT_INF)

Return Code:

Refer to the *Error code*.

11.19 E5K_StartStreamEvent

Description:

Start to receive stream data from IP

Syntax:

Visual Basic/VB.Net: (see E5KDAQ.bas/E5KDAQ.vb)

```
Declare/public Function E5K_StartStreamEvent Lib "E5KDAQ.dll" (ByVal IPaddress As string) As Integer
```

VC++: (see E5KDAQ.h)

```
Int     E5K_StartStreamEventIP (char * IPaddress);
```

Parameters:

IPaddress	Stream data event source IP address

Return Code:

refer to the *Error code*.

11.20 E5K_StopStreamEvent

Description:

Stop to receive stream data from IP

Syntax:

Visual Basic/VB.Net: (see E5KDAQ.bas/E5KDAQ.vb)

```
Declare/public Function E5K_StopStreamEvent Lib "E5KDAQ.dll" (ByVal IPaddress As string) As Integer
```

VC++: (see E5KDAQ.h)

```
Int     E5K_StoptStreamEventIP (char * IPaddress);
```

Parameters:

IPaddress	Stream data event source IP address

Return Code:

refer to the *Error code*.

11.21 E5K_ReadStreamEventData

Description:

Read stream data from IP

Syntax:

Visual Basic/VB.Net: (see E5KDAQ.bas/E5KDAQ.vb)

```
Declare/public Function E5K_ReadStreamEventData Lib "E5KDAQ.dll" (
                        StreamIntInfo As STREAM_EVENT_INFO) As Integer
```

VC++: (see E5KDAQ.h)

```
Int     E5K_ReadStreamEventDa (STREAM_EVENT_INFO StreamIntInfo[]);
```

Parameters:

StreamIntInfo	Points to a stream data buffer (structure STREAM_EVENT_INFO) (see E5KDAQ.H)

Return Code:

refer to the *Error code*.

11.22 E5K_ReadModuleConfig

Description:

Read module configuration

Syntax

Visual Basic/VB.Net: (see E5KDAQ.bas/E5KDAQ.vb)

```
Declare/public Function E5K_ReadModuleConfig Lib "E5KDAQ.dll" _
                        (ByVal id As Integer, mp as MODULE_CONFIG) As integer
```

VC++: (see E5KDAQ.h)

```
unsigned short      E5K_ReadModuleConfig (unsigned int id ,MODULE_CONFIG    *mp);
```

Parameters:

id	module ID address
mp	points to a structure MODULE_CONFIG (see E5KDAQ.H)

Return Code:

refer to the *Error code*.

11.23 E5K_SetModuleConfig

Description:

Set module configuration

Syntax

Visual Basic/VB.Net: (see E5KDAQ.bas/E5KDAQ.vb)

```
Declare/public Function E5K_SetModuleConfig Lib "E5KDAQ.dll" _
                        (ByVal id As Integer , mp as MODULE_CONFIG) As integer
```

VC++: (see E5KDAQ.h)

```
unsigned short      E5K_SetModuleConfig (unsigned int id ,MODULE_CONFIG    *mp);
```

Parameters:

id	module ID address
mp	points to a structure MODULE_CONFIG (see E5KDAQ.H)

Return Code:

refer to the *Error code*.

11.24 E5K_WriteModbusDiscrete

Description:

Write data to Modbus discrete (Modbus coil)

Syntax:

Visual Basic/VB.Net: (see E5KDAQ.bas/E5KDAQ.vb)

```
Declare/public Function E5K_WriteModbusDiscrete Lib "E5KDAQ.dll" _
                        (ByVal id As Integer, ByVal startaddr As Integer,  _
                         ByVal counts As Integer, Discrete As Byte) As Integer
```

VC++: (see E5KDAQ.h)

```
unsigned short    E5K_WriteModbusDiscrete (
                  int id,
                  unsigned int startaddr,
                  unsigned int counts,
                  unsigned char Discrete[]);
```

Parameters:

id	module ID address
startaddr	start address in Modbus coil (0000~FFFF)
counts	how many bit be written
Discrete[]	data buffer be written
	Discrete[n]=0 or 1 for Modbus coil address startaddr+n (0<=n<counts)

Return Code:

refer to the *Error code*.

11.25 E5K_WriteModbusRegister

Description:

Write data to Modbus Holding registers

Syntax:

Visual Basic/VB.Net: (see E5KDAQ.bas/E5KDAQ.vb)

```
Declare/public Function E5K_WriteModbusRegister Lib "E5KDAQ.dll" _
                        (ByVal id As Integer, ByVal startaddr As Integer,  _
                         ByVal counts As Integer, regs As Integer) As Integer
```

VC++: (see E5KDAQ.h)

```
unsigned short    E5K_WriteModbusRegister (
                  int id,
                  unsigned int startaddr,
                  unsigned int counts, Sunsigned int regs[]);
```

Parameters:

id	module ID address
startaddr	start address in Modbus Holding register(0000~FFFF)
counts	how many register be written
regs[]	data buffer be written regs[n]=value for Modbus holding register address startaddr+n (0<=n<counts)

Return Code:

refer to the *Error code*.

11.26 E5K_ReadModbusRegister

Description:

Read Modbus Holding or Input registers

Syntax:

Visual Basic/VB.Net: (see E5KDAQ.bas/E5KDAQ.vb)

```
Declare/public Function E5K_ReadModbusRegister Lib "E5KDAQ.dll" _
                        (ByVal id As Integer, ByVal startaddr As Integer,  _
                         ByVal counts As Integer, regs As Integer) As Integer
```

VC++: (see E5KDAQ.h)

```
unsigned short    E5K_ReadModbusRegister (
                  int id,
                  unsigned int startaddr,
                  unsigned int counts,
                  int regs[]);
```

Parameters:

id	module ID address
startaddr	start address in Modbus Holding or Input register(0000~FFFF)
counts	how many register be read
regs[]	points data buffer

Return Code:

refer to the *Error code*.

11.27 E5K_ReadModbusDiscrete

Description:

Read Modbus coil or discrete input

Syntax:

Visual Basic/VB.Net: (see E5KDAQ.bas/E5KDAQ.vb)

```
Declare/public Function E5K_ReadModbusDiscrete Lib "E5KDAQ.dll" _
                        (       ByVal id As Integer,    _
                                ByVal startaddr As Integer, _
                                ByVal counts As Integer,    _
                                Discrete As Byte _
                        ) As Integer
```

VC++: (see E5KDAQ.h)

```
unsigned short      E5K_ReadModbusDiscrete(
                    int id,
                    unsigned int startaddr,
                    unsigned int counts,
                    unsigned char Discrete[]);
```

Parameters:

id	module ID address
startaddr	start address in Modbus Holding or Input register(0000~FFFF)
counts	how many bit be read
Discrete[]	points data buffer Discrete[n]=0 or 1 of Modbus coil or discrete input address startaddr+n (0<=n<counts)

Return Code:

11.28 E5K_SendASCRequestAndWaitResponse

Description:

Send an ASCII string to and wait for response from module

Syntax:

Visual Basic/VB.Net: (see E5KDAQ.bas/E5KDAQ.vb)

```
Declare/public Function E5K_SendASCRequestAndWaitResponse Lib "E5KDAQ.dll" _
                       (ByVal id As Integer, ByVal asccmd As String,   _
                        ByVal response As String,
                        ByVal rxbuffersize As Integer) As Integer
```

VC++: (see E5KDAQ.h)

```
unsigned short    E5K_SendASCRequestAndWaitResponse(
                  int id,
                  unsigned char asccmd[],
                  unsigned char response[],
                  unsigned int rxbuffersize);
```

Parameters:

id	module ID address
asccmd	points to ASCII string buffer
response	points response buffer
rxbuffersize	size of response buffer

Return Code:

refer to the *Error code*.

11.29 E5K_RecvASCII

Description:

Receive an ASCII string from the module

Syntax:

Visual Basic/VB.Net: (see E5KDAQ.bas/E5KDAQ.vb)

```
Declare/public Function E5K_RecvASCII     Lib "E5KDAQ.dll" _
                       (ByVal id As Integer, _
                        ByVal Rxbuffer As String, _
                        ByVal bufferSize As integer ) as integer
```

VC++: (see E5KDAQ.h)

```
unsigned short    E5K_RecvASCII (int id ,char Rxbuffer[], unsigned int BufferSize);
```

Parameters:

id	module ID address
asccmd	points to ASCII string buffer
Rxbuffer	points receive buffer
Buffersize	size of revive buffer

Return Code:

refer to the *Error code*.

11.30 E5K_SendASCII

Description:

Send an ASCII string to the module

Syntax:

Visual Basic/VB.Net: (see E5KDAQ.bas/E5KDAQ.vb)

```
Declare/public Function E5K_SendASCII     Lib "E5KDAQ.dll" _
                        (ByVal id As Integer, ByVal Txstring As String ) as integer
```

VC++: (see E5KDAQ.h)

```
unsigned short     E5K_SendASCII                (int id ,char Txbuffer[]);
```

Parameters:

id	module ID address
Txstring	points ASCII string buffer

Return Code:

refer to the *Error code*.

11.31 E5K_SendHEXRequestAndWaitResponse

Description:

Send binary data to and wait for response from module

Syntax:

Visual Basic/VB.Net: (see E5KDAQ.bas/E5KDAQ.vb)

```
Declare/public Function E5K_SendHEXRequestAndWaitResponse Lib "E5KDAQ.dll" _
                        (       ByVal id As Integer, _
                                ByRef Txdata As Byte,   _
                                ByVal Rxlen As Integer,   _
                                ByRef Rxdata As Byte, _
                                ByRef Rxlen As Integer, _
                                ByVal RxBufsize As Integer _
                        ) As Integer
```

VC++: (see E5KDAQ.h)

```
unsigned short          E5K_SendHEXRequestAndWaitResponse(
                        int id ,
                        unsigned char cTxData[],
                        unsigned int wTxlen,
                        unsigned char cRxdata[],
                        unsigned int *wRxlen,
                        unsigned int buffersize);
```

Parameters:

id	module ID address
cTxData[]	points to binary data buffer
wTxlen	how many bytes be sent
cRxdata[]	points response buffer
*wRxlen	point to buffer to store the number of byte received
buffersize	size of response buffer

Return Code:

refer to the *Error code*.

11.32 E5K_SendHEX

Description:

Send an Hex data to the module

Syntax:

Visual Basic/VB.Net: (see E5KDAQ.bas/E5KDAQ.vb)

```
Declare/public Function E5K_SendHex     Lib "E5KDAQ.dll" _
                        (ByVal id As Integer, Byref Hexdata As Byte, ByVal Datalen As Integer ) as integer
```

VC++: (see E5KDAQ.h)

```
unsigned short     E5K_SendHex (int id ,char Hexdata[],unsigned int Datalen);
```

Parameters:

id	module ID address
Hexdata	points Hex data buffer
Datalen	size of the data buffer

Return Code:

refer to the *Error code*.

11.33 E5K_RecvHEX

Description:

Receive an Hex data from the module

Syntax:

Visual Basic/VB.Net: (see E5KDAQ.bas/E5KDAQ.vb)

```
Declare/public Function E5K_RecvHex     Lib "E5KDAQ.dll" _
                        (ByVal id As Integer, _
                         Byref Rxbuffer As String, _
                         ByVal bufferSize As integer ) as integer
```

VC++: (see E5KDAQ.h)

```
unsigned short     E5K_RecvHex (int id ,char Rxbuffer[], unsigned int BufferSize);
```

Parameters:

id	module ID address
asccmd	points to ASCII string buffer
Rxbuffer	points receive buffer
Buffersize	size of revive buffer

Return Code:

refer to the *Error code*.

11.34 E5K_CalculateCRC16

Description:

Calculate CRC16

Syntax:

Visual Basic/VB.Net: (see E5KDAQ.bas/E5KDAQ.vb)

```
Declare/public Function E5K_CalculateCRC16 Lib "E5KDAQ.dll" _
                        (bData as byte , byval wLen as integer ,byref wCRC as integer) As Integer
```

VC++: (see E5KDAQ.h)

```
unsigned short      E5K_CalculateCRC16 (
                    unsigned char bData[],
                    unsigned int wLen,
                    unsigned int *wCRC);
```

Parameters:

bData[]	points to binary data buffer
wLen	size of data
wCRC	points to buffer to store CRC16 value

Return Code:

refer to the *Error code*.

11.35 E5K_SetLEDControl

Description:

Set on-board control mode (controlled by Module or by user AP)

Syntax:

Visual Basic/VB.Net: (see E5KDAQ.bas/E5KDAQ.vb)

```
Declare/public Function E5K_SetLEDControl Lib "E5KDAQ.dll" _
                        (ByVal id As Integer, ByVal ControlOption As Integer) As Integer
```

VC++: (see E5KDAQ.h)

```
unsigned short      E5K_SetLEDControl ( int id, char ControlOption);
```

Parameters:

id	module ID address
ControlOption	on-board LED control mode. 0: controlled by module, 1: controlled by user AP

Return Code:

refer to the *Error code*.

11.36 E5K_WriteDataToLED

Description:

Write data to on-board LED

Syntax:

Visual Basic/VB.Net: (see E5KDAQ.bas/E5KDAQ.vb)

```
Declare/public Function E5K_WriteDataToLED Lib "E5KDAQ.dll" _
                        (ByVal id As Integer, ByVal LedData) As Integer
```

VC++: (see E5KDAQ.h)

```
unsigned short      E5K_WriteDataToLED (int id, unsigned long LedData);
```

Parameters:

id	module ID address
LedData	Data to be written to on-board LED. bit #n=0: LED #n off, bit n=1: LED #n on

Return Code:

refer to the *Error code*.

11.37 E5K_FlashLED

Description:

Force on-board LED to flash

Syntax:

Visual Basic/VB.Net: (see E5KDAQ.bas/E5KDAQ.vb)

```
Declare/public Function E5K_FlashLED Lib "E5KDAQ.dll" _
                        (ByVal id As Integer, ByVal LedMask As Integer, FlashCount As Long) As Integer
```

VC++: (see E5KDAQ.h)

```
unsigned short      E5K_FlashLED (int id, unsigned long LedMask,unsigned int FlashCounts);
```

Parameters:

id	module ID address
LedMask	LED channel mask. bit #n=0: No Flash LED #n, bit n=1: Flash LED #n
FlashCounts	Flash counts

Return Code:

refer to the *Error code*.

11.38 E5K_IsValidIPAddress

Description:

Check the validity of the specified IP address

Syntax:

Visual Basic/VB.Net: (see E5KDAQ.bas/E5KDAQ.vb)

```
Declare/public Function        E5K_IsValidIPAddress Lib "E5KDAQ.dll" _
                               (ByVal zIP As String) As Integer
```

VC++: (see E5KDAQ.h)

```
unsigned short          E5K_IsValidIPAddress (char *zIP);
```

Parameters:

id	module ID address
zIP	IP address string (such as 192.168.0.21)

Return Code:

refer to the *Error code*.

11.39 E5K_GetLocalIP

Description:

Get host IP address

Syntax

Visual Basic/VB.Net: (see E5KDAQ.bas/E5KDAQ.vb)

```
Declare/public Function E5K_GetLocalIP Lib "E5KDAQ.dll" _
                                (       ip0 as byte, _
                                        ip1 as byte, _
                                        ip2 as byte, _
                                        ip3 as byte _
                                ) as integer
```

VC++/BC++Builder: (see E5KDAQ.h)

```
unsigned short          E5K_GetLocalIP(
                        char *ip0,
                        char *ip1,
                        char *ip2,
                        char *ip3);
```

Parameters:

Ip0	first IP address byte for an EDAM-5000 that to be connected
Ip1	second IP address byte for an EDAM-5000 that to be connected
Ip2	third IP address byte for an EDAM-5000 that to be connected
Ip3	forth IP address byte for an EDAM-5000 that to be connected

Return Code:

refer to the *Error code*.

11.40 E5K_TCPConnect

Description:

Establish a TCP connection

Syntax

Visual Basic/VB.Net: (see E5KDAQ.bas/E5KDAQ.vb)

```
Declare/public Function E5K_TCPConnect Lib "E5KDAQ.dll" _
                    (ByVal zIP As String, ByVal port As Integer, ByVal iConnectionTimeout As Integer, ByVal
iSendTimeout As Integer, ByVal iReceiveTimeout As Integer) As Integer
```

VC++/BC++Builder: (see E5KDAQ.h)

```
unsigned short          E5K_TCPConnect    (
                        char szIP[], u_short port,int iConnectionTimeout,
                        int iSendTimeout,int iReceiveTimeout);
```

Parameters:

szIP	Target IP address
port	connection port
iConnectionTimeout	connection timeout value(msec)
iSendTimeout	send timeout value(msec)
iReceiveTimeout	receive timeout value(msec)

Return Code:

refer to the *Error code*.

11.41 E5K_TCPSendData

Description:

Send data to TCP connection

Syntax

Visual Basic/VB.Net: (see E5KDAQ.bas/E5KDAQ.vb)

```
Declare/public Function E5K_TCPSendData Lib "E5KDAQ.dll" _
                    (ByVal sock As Long, ByRef pdata As Byte, ByVal datalen As Integer) As Integer
```

VC++/BC++Builder: (see E5KDAQ.h)

```
unsigned short      E5K_TCPSendData (SOCKET sock,SBYTE *pdata,u_short datalen);
```

Parameters:

sock	TCP socket handle
pdata	Points to data buffer
datalen	bytes of data

Return Code:

refer to the *Error code*.

11.42 E5K_TCPRecvData

Description:

Receive data from TCP connection

Syntax

Visual Basic/VB.Net: (see E5KDAQ.bas/E5KDAQ.vb)

```
Declare/public Function E5K_TCPRecvData Lib "E5KDAQ.dll" _
                        (ByVal sock As Long, ByRef pdata As Byte, ByVal psize As Integer) As Integer
```

VC++/BC++Builder: (see E5KDAQ.h)

```
unsigned short      E5K_TCPRecvData (SOCKET sock, char *pdata, unsigned short psize);
```

Parameters:

sock	TCP socket handle
pdata	Points to data buffer
psize	size of data buffer

Return Code:

Bytes of data received

11.43 E5K_TCPPing

Description:

Ping Specified IP address

Syntax

Visual Basic/VB.Net: (see E5KDAQ.bas/E5KDAQ.vb)

```
Declare/public Function E5K_TCPPing Lib "E5KDAQ.dll" _
                        (ByVal zIP As String, ByVal timeout As Integer) As Integer
```

VC++/BC++Builder: (see E5KDAQ.h)

```
unsigned short      E5K_TCPPing (char zIP[],int timeout);
```

Parameters:

zIP	IP address string (such as 192.168.0.123)
timeout	ping timeout (msec)

Return Code:

refer to the *Error code*.

11.44 E5K_TCPDisconnect

Description:

Release a TCP connection

Syntax

Visual Basic/VB.Net: (see E5KDAQ.bas/E5KDAQ.vb)

```
Declare/public Function E5K_TCPDisconnect Lib "E5KDAQ.dll" _
                        (ByVal sock As Long) As Integer
```

VC++/BC++Builder: (see E5KDAQ.h)

```
unsigned short      E5K_TCPDisconnect (SOCKET sock);
```

Parameters:

sock	TCP connection handle

Return Code:

refer to the *Error code*.

11.45 E5K_ReadAIChannelType

Description:

Read analog channel type

Syntax

Visual Basic/VB.Net: (see E5KDAQ.bas/E5KDAQ.vb)

```
Declare/public Function E5K_ReadAIChannelType Lib "E5KDAQ.dll" _
                        (ByVal id As Integer, ByVal AIChannel As Integer, ByRef AIType As Integer) As Integer
```

VC++/BC++Builder: (see E5KDAQ.h)

```
unsigned short      E5K_ReadAIChannelType (int    id,unsigned int AIChannel,unsigned int *AIType);
```

Parameters:

Id	target module ID
AIChannel	channel number
AIType	buffer pointer to store the Channel Type(See sec.8.2)

Return Code:

refer to the *Error code*.

11.46 E5K_SetAIChannelType

Description:

Set analog channel type

Syntax

Visual Basic/VB.Net: (see E5KDAQ.bas/E5KDAQ.vb)

```
Declare/public Function E5K_SetAIChannelType Lib "E5KDAQ.dll" _
                    (ByVal id As Integer, ByVal AIChannel As Integer, ByVal AIType As Integer) As Integer
```

VC++/BC++Builder: (see E5KDAQ.h)

```
unsigned short    E5K_SetAIChannelType (int   id,unsigned int AIChannel,unsigned int AItype);
```

Parameters:

Id	target module ID
AIChannel	channel number
AIType	buffer pointer to store the Channel Type (See sec.8.2)

Return Code:

refer to the *Error code*.

11.47 E5K_SetSingleChannelColdJunctionOffset

Description:

Set cold junction offset of single analog channel

Syntax

Visual Basic/VB.Net: (see E5KDAQ.bas/E5KDAQ.vb)

```
Declare/public Function E5K_SetSingleChannelColdJunctionOffset Lib "E5KDAQ.dll" _
                    (ByVal id As Integer, ByVal chno As Integer, ByVal CjOffset As Double) As Integer
```

VC++/BC++Builder: (see E5KDAQ.h)

```
unsigned short    E5K_SetSingleChannelColdJunctionOffset (int id ,unsigned int chno,double CjOffset);
```

Parameters:

Id	the target module id
Chno	channel number
CjOffset	channel CJC offset value (such as 0.231)

Return Code:

refer to the *Error code*.

11.48 E5K_ReadSingleChannelColdJunctionOffset

Description:

Read cold junction offset of single analog channel

Syntax

Visual Basic/VB.Net: (see E5KDAQ.bas/E5KDAQ.vb)

```
Declare/public Function E5K_ReadSingleChannelColdJunctionOffset Lib "E5KDAQ.dll" _
                    (ByVal id As Integer, ByVal chno As Integer, ByRef CjOffset As Double) As Integer
```

VC++/BC++Builder: (see E5KDAQ.h)

```
unsigned short      E5K_ReadSingleChannelColdJunctionOffset(
                    int id,
                    unsigned int chno,
                    double *CJoffset);
```

Parameters:

Id	the target module id
Chno	channel number
CjOffset	buffer pointer to store channel CJC offset value

Return Code:

refer to the *Error code*.

11.49 E5K_ReadMultiChannelColdJunctionOffset

Description:

Read cold junction offset of multiple analog channels

Syntax

Visual Basic/VB.Net: (see E5KDAQ.bas/E5KDAQ.vb)

```
Declare/public Function E5K_ReadMultiChannelColdJunctionOffset Lib "E5KDAQ.dll" _
                    (ByVal id As Integer, ByVal startch As Integer, Byval counts as integer,Byref CjOffset As
Double) As Integer
```

VC++/BC++Builder: (see E5KDAQ.h)

```
unsigned short      E5K_ReadMultiChannelColdJunctionOffset(
                    unsigned int id ,
                    unsigned int startch,
                    unsigned int counts,
                    double * CjOffset);
```

Parameters:

Id	the target module id
startch	start channel number
counts	channels to be read
CjOffset	points to an array to store CJC offset value

Return Code:

refer to the *Error code*.

11.50 E5K_SetMultiChannelColdJunctionOffset

Description:

Set cold junction offset of multiple analog channels

Syntax

Visual Basic/VB.Net: (see E5KDAQ.bas/E5KDAQ.vb)

```
Declare/public Function E5K_SetMultiChannelColdJunctionOffset Lib "E5KDAQ.dll" _
                        ( ByVal id As Integer, _
                          ByVal startch As Integer, _
                          Byval counts as integer, _
                          Byref CjOffset As Double _
                          ) As Integer
```

VC++/BC++Builder: (see E5KDAQ.h)

```
unsigned short      E5K_SetMultiChannelColdJunctionOffset(
                    unsigned int id ,
                    unsigned int startch,
                    unsigned int counts,
                    double * CjOffset);
```

Parameters:

Id	the target module id
startch	start channel number
counts	channels to be set
CjOffset	points to an array where store channel CJC offset values

Return Code:

refer to the *Error code*.

11.51 E5K_ReadColdJunctionTemperature

Description:

Read cold junction temperature

Syntax

Visual Basic/VB.Net: (see E5KDAQ.bas/E5KDAQ.vb)

```
Declare/public Function E5K_ReadColdJunctionTemperature Lib "E5KDAQ.dll" _
                        (ByVal id As Integer, ByRef CjTemp As Double) As Integer
```

VC++/BC++Builder: (see E5KDAQ.h)

```
unsigned short E5K_ReadColdJunctionTemperature(int id,double *CJtemp);
```

Parameters:

Id	the target module id
CJtemp	CJC temperature (such as 23.67)

Return Code:

refer to the *Error code*.

11.52 E5K_ReadColdJunctionStatus

Description:

Read CJC enable/disable status

Syntax

Visual Basic/VB.Net: (see E5KDAQ.bas/E5KDAQ.vb)

```
Declare/public Function E5K_ReadColdJunctionStatus Lib "E5KDAQ.dll" _
                        (ByVal id As Integer, ByRef Cjs As Byte) As Integer
```

VC++/BC++Builder: (see E5KDAQ.h)

```
unsigned short E5K_ReadColdJunctionStatus (int id ,SBYTE *Cjs);
```

Parameters:

Id	the target module id
Cjs	CJC enable/disable status (0: disabled, 1: enabled)

Return Code:

refer to the *Error code*.

11.53 E5K_SetColdJunction

Description:

Enable/disable CJC

Syntax

Visual Basic/VB.Net: (see E5KDAQ.bas/E5KDAQ.vb)

```
Declare/public Function
E5K_SetColdJunction Lib "E5KDAQ.dll" _
                        (ByVal id As Integer, ByVal Cjs As Byte) As Integer
```

VC++/BC++Builder: (see E5KDAQ.h)

```
unsigned short E5K_SetColdJunction (int id,SBYTE Cjs);
```

Parameters:

Id	the target module id
Cjs	CJC enable/disable option (0: disable, 1: enable)

Return Code:

refer to the *Error code*.

11.54 E5K_ReadAIChannelConfig

Description:

Read analog channel configuration

Syntax

Visual Basic/VB.Net: (see E5KDAQ.bas/E5KDAQ.vb)

```
Declare/public Function E5K_ReadAIChannelConfig Lib "E5KDAQ.dll" _
                    (ByVal id As Integer, ByVal chno As Integer, ByRef mConfig As AI_CHANNEL_CONFIG)
As Integer
```

VC++/BC++Builder: (see E5KDAQ.h)

```
unsigned short E5K_ReadAIChannelConfig (int id ,unsigned int chno,AI_CHANNEL_CONFIG * mConfig);
```

Parameters:

Id	the target module id
chno	channel number
mConfig	points to a structure to store the channel configuration (see E5KDAQ.H about structure CHANNEL_CONFIG)

Return Code:

refer to the *Error code*.

11.55 E5K_SetAIChannelConfig

Description:

Set analog channel configuration

Syntax

Visual Basic/VB.Net: (see E5KDAQ.bas/E5KDAQ.vb)

```
Declare/public Function E5K_SetAIChannelConfig Lib "E5KDAQ.dll" _
                    (ByVal id As Integer, ByVal chno As Integer, mConfig As AI_CHANNEL_CONFIG) As
Integer
```

VC++/BC++Builder: (see E5KDAQ.h)

```
unsigned short E5K_SetAIChannelConfig (int id ,unsigned int ch,AI_CHANNEL_CONFIG * mConfig);
```

Parameters:

Id	the target module id
chno	channel number
mConfig	points to a structure where stores the channel configuration (see E5KDAQ.H about structure CHANNEL_CONFIG)

Return Code:

refer to the *Error code*.

11.56 E5K_ReadAIBurnOutStatus

Description:

Read analog burnout status

Syntax

Visual Basic/VB.Net: (see E5KDAQ.bas/E5KDAQ.vb)

```
Declare/public Function E5K_ReadAIBurnOutStatus Lib "E5KDAQ.dll" _
                        (ByVal id As Integer, ByRef status As Long) As Integer
```

VC++/BC++Builder: (see E5KDAQ.h)

```
unsigned short E5K_ReadAIBurnOutStatus (int id ,unsigned int *status);
```

Parameters:

Id	the target module id
status	points to a buffer to store the channel burnout status (bit #n=0: channel #n is normal, bit #n=1 channel #n is burnout)

Return Code:

refer to the *Error code*.

11.57 E5K_ReadAIAlarmStatus

Description:

Read high/low alarm status of analog channels

Syntax

Visual Basic/VB.Net: (see E5KDAQ.bas/E5KDAQ.vb)

```
Declare/public Function E5K_ReadAIAlarmStatus Lib "E5KDAQ.dll" _
                        (ByVal id As Integer, ByRef hialarm As Long, ByRef loalarm As Long) As Integer
```

VC++/BC++Builder: (see E5KDAQ.h)

```
unsigned short E5K_ReadAIAlarmStatus (int    id,unsigned int *hialarm,unsigned int *loalarm);
```

Parameters:

Id	the target module id
hialarm	points to a buffer to store the channel high alarm status (bit #n=0: channel #n is normal, bit #n=1 channel #n is high alarm)
loalarm	points to a buffer to store the channel low alarm status (bit #n=0: channel #n is normal, bit #n=1 channel #n is low alarm)

Return Code:

refer to the *Error code*.

11.58 E5K_SetAIBurnOut

Description:

Enable/disable AI burnout detection

Syntax

Visual Basic/VB.Net: (see E5KDAQ.bas/E5KDAQ.vb)

```
Declare/public Function E5K_SetAIBurnOut Lib "E5KDAQ.dll" _
                        (ByVal id As Integer, ByVal option As Byte) As Integer
```

VC++/BC++Builder: (see E5KDAQ.h)

```
unsigned short      E5K_SetAIBurnOut (int id ,SBYTE option);
```

Parameters:

Id	the target module id
option	=0: disable burnout detection, =1: enable burnout detection

Return Code:

refer to the *Error code*.

11.59 E5K_ReadAIBurnOut

Description:

Read burnout detection enable/disable status

Syntax

Visual Basic/VB.Net: (see E5KDAQ.bas/E5KDAQ.vb)

```
Declare/public Function E5K_ReadAIBurnOut Lib "E5KDAQ.dll" _
                        (ByVal id As Integer, ByRef option As Byte) As Integer
```

VC++/BC++Builder: (see E5KDAQ.h)

```
unsigned short      E5K_ReadAIBurnOut (int id ,SBYTE *option);
```

Parameters:

Id	the target module id
option	points to a buffer to store the burnout detection enable/disable status =0: burnout detection disabled, =1: burnout detection enabled

Return Code:

refer to the *Error code*.

11.60 E5K_SetAIModuleFilter

Description:

Set A/D filter frequency

Syntax

Visual Basic/VB.Net: (see E5KDAQ.bas/E5KDAQ.vb)

```
Declare/public Function E5K_SetAIModuleFilter Lib "E5KDAQ.dll" _
                        (ByVal id As Integer, ByVal Hz As Integer) As Integer
```

VC++/BC++Builder: (see E5KDAQ.h)

```
unsigned short      E5K_SetAIModuleFilter (int id ,unsigned int Hz);
```

Parameters:

Id	the target module id
Hz	=50: 50Hz, =60: 60Hz, =100 100Hz, =120 120Hz

Return Code:

refer to the *Error code*.

11.61 E5K_ReadAIModuleFilter

Description:

Read A/D filter frequency

Syntax

Visual Basic/VB.Net: (see E5KDAQ.bas/E5KDAQ.vb)

```
Declare/public Function E5K_ReadAIModuleFilter Lib "E5KDAQ.dll" _
                        (ByVal id As Integer, ByRef Hz As Integer) As Integer
```

VC++/BC++Builder: (see E5KDAQ.h)

```
unsigned short E5K_ReadAIModuleFilter (int id ,unsigned int *Hz);
```

Parameters:

Id	the target module id
Hz	points to a buffer to store filter frequency =50: 50Hz, =60: 60Hz, =100 100Hz, =120 120Hz

Return Code:

refer to the *Error code*.

11.62 E5K_SetAIChannelEnable

Description:

Enable or disable analog channels

Syntax

Visual Basic/VB.Net: (see E5KDAQ.bas/E5KDAQ.vb)

```
Declare/public Function E5K_SetAIChannelEnable Lib "E5KDAQ.dll" _
                        (ByVal id As Integer, ByVal AIEnable As Long) As Integer
```

VC++/BC++Builder: (see E5KDAQ.h)

```
unsigned short      E5K_SetAIChannelEnable (int id ,unsigned int AIEnable);
```

Parameters:

Id	the target module id
AIEnable	Enable/disable settings bit #n=0: disable channel #n, bit #n=1: enable channel #n

Return Code:

refer to the *Error code*.

11.63 E5K_ReadAIChannelEnable

Description:

Read enable/disable status of analog channels

Syntax

Visual Basic/VB.Net: (see E5KDAQ.bas/E5KDAQ.vb)

```
Declare/public Function E5K_ReadAIChannelEnable Lib "E5KDAQ.dll" _
                        (ByVal id As Integer, ByRef AIEnable As Long) As Integer
```

VC++/BC++Builder: (see E5KDAQ.h)

```
unsigned short      E5K_ReadAIChannelEnable (int id ,unsigned int * AIEnable);
```

Parameters:

Id	the target module id
AIEnable	points to a buffer to store channel Enable/disable settings bit #n=0: channel #n is disabled, bit #n=1: channel #n is enabled

Return Code:

refer to the *Error code*.

11.64 E5K_ReadAINormalMultiChannel

Description:

Read normal value of the multiple analog channels

Syntax

Visual Basic/VB.Net: (see E5KDAQ.bas/E5KDAQ.vb)

```
Declare/public Function E5K_ReadAINormalMultiChannel Lib "E5KDAQ.dll" _
                    (ByVal id As Integer, ByVal startch As Integer, ByVal counts As Integer, ByRef AItemp As
Double) As Integer
```

VC++/BC++Builder: (see E5KDAQ.h)

```
unsigned short      E5K_ReadAINormalMultiChannel (
                    int id ,
                    unsigned int startch,
                    unsigned int counts,
                    double *AItemp);
```

Parameters:

Id	the target module id
Startch	start channel number
counts	channels

AItemp	points to a array to store AI normal values
	AItemp[0]=normal value of channel #startch
	AItemp[1]=normal value of channel #startch+1
	AItemp[2]=normal value of channel #startch+2
	…..etc

Return Code:

refer to the *Error code*.

11.65 E5K_ReadAIMaximumMultiChannel

Description:

Read maximum value of the multiple analog channels

Syntax

Visual Basic/VB.Net: (see E5KDAQ.bas/E5KDAQ.vb)

```
Declare/public Function E5K_ReadAIMaximumMultiChannel Lib "E5KDAQ.dll" _
                    (ByVal id As Integer, ByVal startch As Integer, ByVal counts As Integer, ByRef AItemp As
Double) As Integer
```

VC++/BC++Builder: (see E5KDAQ.h)

```
unsigned short      E5K_ReadAIMaximumMultiChannel (
                    int id ,
                    unsigned int startch,
                    unsigned int count,
                    double *AItemp);
```

Parameters:

Id	the target module id
Startch	start channel number
counts	channels

AItemp	points to a array to store AI maximum values
	AItemp[0]= maximum value of channel #startch
	AItemp[1]= maximum value of channel #startch+1
	AItemp[2]= maximum value of channel #startch+2
	…..etc

Return Code:

refer to the *Error code*.

11.66 E5K_ReadAIMinumumMultiChannel

Description:

Read minimum value of the multiple analog channels

Syntax

Visual Basic/VB.Net: (see E5KDAQ.bas/E5KDAQ.vb)

```
Declare/public Function E5K_ReadAIMinimunMultiChannel Lib "E5KDAQ.dll" _
                    (ByVal id As Integer, ByVal startch As Integer, ByVal counts As Integer, ByRef AItemp As
Double) As Integer
```

VC++/BC++Builder: (see E5KDAQ.h)

```
unsigned short      E5K_ReadAIMinimunMultiChannel (
                    int id ,
                    unsigned int startch,
                    unsigned int count,
                    double *AItemp);
```

Parameters:

Id	the target module id
Startch	start channel number
counts	channels

AItemp	points to a array to store AI minimum values
	AItemp[0]= minimum value of channel #startch
	AItemp[1]= minimum value of channel #startch+1
	AItemp[2]= minimum value of channel #startch+2
	…..etc

Return Code:

refer to the *Error code*.

11.67 E5K_ResetAIMaximum

Description:

Reset analog channel maximum value

Syntax

Visual Basic/VB.Net: (see E5KDAQ.bas/E5KDAQ.vb)

```
Declare/public Function E5K_ResetAIMaximum Lib "E5KDAQ.dll" _
                    (ByVal id As Integer, ByVal restopt As Long) As Integer
```

VC++/BC++Builder: (see E5KDAQ.h)

```
unsigned short E5K_ResetAIMaximum (int id,unsigned int resetopt);
```

Parameters:

Id	the target module ID
Restopt	rest mask option bit #n=0: no reset maximum value of channel #n bit #n=1: reset maximum value of channel #n

Return Code:

refer to the *Error code*.

11.68 E5K_ResetAIMinimum

Description:

Reset analog channel minimum value

Syntax

Visual Basic/VB.Net: (see E5KDAQ.bas/E5KDAQ.vb)

```
Declare/public Function E5K_ResetAIMinimum Lib "E5KDAQ.dll" _
                        (ByVal id As Integer, ByVal resetopts As Long) As Integer
```

VC++/BC++Builder: (see E5KDAQ.h)

```
unsigned short E5K_ResetAIMinimum (int id ,unsigned int Restopt);
```

Parameters:

Id	the target module ID
Restopt	rest mask option bit #n=0: no reset minimum value of channel #n bit #n=1: reset minimum value of channel #n

Return Code:

refer to the *Error code*.

11.69 E5K_ResetAIHighAlarm

Description:

Reset analog high alarm flag

Syntax

Visual Basic/VB.Net: (see E5KDAQ.bas/E5KDAQ.vb)

```
Declare/public Function E5K_ResetAIHighAlarm Lib "E5KDAQ.dll" _
                        (ByVal id As Integer, ByVal restopt As Long) As Integer
```

VC++/BC++Builder: (see E5KDAQ.h)

```
unsigned short E5K_ResetAIHighAlarm (int id ,unsigned int restopt);
```

Parameters:

Id	the target module ID
restopt	rest mask option bit #n=0: no reset high alarm flag of channel #n bit #n=1: reset high alarm flag of channel #n

Return Code:

refer to the *Error code*.

11.70 E5K_ResetAILowAlarm

Description:

Reset analog low alarm flag

Syntax

Visual Basic/VB.Net: (see E5KDAQ.bas/E5KDAQ.vb)

```
Declare/public Function E5K_ResetAILowAlarm Lib "E5KDAQ.dll" _
                        (ByVal id As Integer, ByVal restopt As Long) As Integer
```

VC++/BC++Builder: (see E5KDAQ.h)

```
unsigned short E5K_ResetAILowAlarm (int    id,unsigned int restopt);
```

Parameters:

Id	the target module ID
restopt	rest mask option bit #n=0: no reset low alarm flag of channel #n bit #n=1: reset low alarm flag of channel #n

Return Code:

refer to the *Error code*.

11.71 E5K_ReadAIChannelAverage

Description:

Read analog channel in average status

Syntax

Visual Basic/VB.Net: (see E5KDAQ.bas/E5KDAQ.vb)

```
Declare/public Function E5K_ReadAIChannelAverage Lib "E5KDAQ.dll" _
                        (ByVal id As Integer, ByRef inavg As Long) As Integer
```

VC++/BC++Builder: (see E5KDAQ.h)

```
unsigned short E5K_ReadAIChannelAverage (int    id,unsigned int * inavg);
```

Parameters:

Id	the target module id
Inavg	points to a buffer to store the in average status of channels bit #n=0: channel #n is not in average bit #n=1: channel #n is in average

Return Code:

refer to the *Error code*.

11.72 E5K_SetAIChannelAverage

Description:

Set analog channel in average

Syntax

Visual Basic/VB.Net: (see E5KDAQ.bas/E5KDAQ.vb)

```
Declare/public Function E5K_SetAIChannelAverage Lib "E5KDAQ.dll" _
                        (ByVal id As Integer, ByVal inavg As Long) As Integer
```

VC++/BC++Builder: (see E5KDAQ.h)

```
unsigned short E5K_SetAIChannelAverage    (int    id,unsigned int inavg );
```

Parameters:

Id the target module id

inavg in average status of channels

bit #n=0: set channel #n to be not in average
bit #n=1: set channel #n to be in average

Return Code:

refer to the *Error code*.

11.73 E5K_SetDIChannelConfig

Description:

Set DI channel configuration

Syntax

Visual Basic/VB.Net: (see E5KDAQ.bas/E5KDAQ.vb)

```
Declare/public Function E5K_SetDIChannelConfig Lib "E5KDAQ.dll" _
(ByVal id As Integer, ByVal chn As Integer, config As DI_CHANNEL_CONFIG) As Integer
```

VC++/BC++Builder: (see E5KDAQ.h)

```
unsigned short E5K_SetDIChannelConfig (int id ,unsigned int chn,DI_CHANNEL_CONFIG * config);
```

Parameters:

Id	the target module id
Chn	DI channel number
Config	points to a structure buffer where stores the DI configuration parameters(see E5KDAQ.H)

Return Code:

refer to the *Error code*.

11.74 E5K_ReadDIChannelConfig

Description:

Read DI channel configuration

Syntax

Visual Basic/VB.Net: (see E5KDAQ.bas/E5KDAQ.vb)

```
Declare/public Function E5K_ReadDIChannelConfig Lib "E5KDAQ.dll" _
(ByVal id As Integer, ByVal chn As Integer, config e As DI_CHANNEL_CONFIG) As Integer
```

VC++/BC++Builder: (see E5KDAQ.h)

```
unsigned short      E5K_ReadDIChannelConfig (
                    int id ,
                    unsigned int chn,
                    DI_CHANNEL_CONFIG * confige);
```

Parameters:

Id	the target module id
Chn	DI channel number
config	points to a structure buffer to store the DI configuration parameters(see E5KDAQ.H)

Return Code:

refer to the *Error code*.

11.75 E5K_ReadDIStatus

Description:

Read digital input status

Syntax:

Visual Basic/VB.Net: (see E5KDAQ.bas/E5KDAQ.vb)

```
Declare/public Function E5K_ReadDIStatus Lib "E5KDAQ.dll" _
                        (ByVal id As Integer, ByRef Didata As Long) As Integer
```

VC++: (see E5KDAQ.h)

```
unsigned short      E5K_ReadDIStatus (int id ,unsigned long *Didata);
```

Parameters:

id	module ID address
Didata	points to a 32-bit buffer to store DI status

Return Code:

refer to the *Error code*.

11.76 E5K_ReadDILatch

Description:

Read digital input latch status

Syntax:

Visual Basic/VB.Net: (see E5KDAQ.bas/E5KDAQ.vb)

```
Declare/public Function E5K_ReadDILatch Lib "E5KDAQ.dll" _
                        (ByVal id As Integer, ByRef Dilatch As Long) As Integer
```

VC++: (see E5KDAQ.h)

```
unsigned short      E5K_ReadDILatch (int id ,unsigned long *Dilatch);
```

Parameters:

id	module ID address
Dilatch	points to a 32-bit buffer to store DI latch status

Return Code:

refer to the *Error code*.

11.77 E5K_ClearAllDILatch

Description:

Clear all digital input latch status

Syntax:

Visual Basic/VB.Net: (see E5KDAQ.bas/E5KDAQ.vb)

```
Declare/public Function E5K_ClearAllDILatch Lib "E5KDAQ.dll" _
                            (ByVal id As Integer) As Integer
```

VC++: (see E5KDAQ.h)

```
unsigned short      E5K_ClearAllDILatch (int id);
```

Parameters:

id	module ID address

Return Code:

refer to the *Error code*.

11.78 E5K_ClearSingleDICounter

Description:

Clear counter of single digital input channel

Syntax:

Visual Basic/VB.Net: (see E5KDAQ.bas/E5KDAQ.vb)

```
Declare/public Function E5K_ClearSingleDICounter Lib "E5KDAQ.dll" _
                        (ByVal id As Integer, ByVal chan As Integer) As Integer
```

VC++: (see E5KDAQ.h)

```
unsigned short    E5K_ClearSingleDICounter (int id ,unsigned int chan);
```

Parameters:

id	module ID address
chan	channel no.

Return Code:

refer to the *Error code*.

11.79 E5K_ReadMultiDICounter

Description:

Clear counter of single digital input channel

Syntax:

Visual Basic/VB.Net: (see E5KDAQ.bas/E5KDAQ.vb)

```
Declare/public Function E5K_ReadMultiDICounter Lib "E5KDAQ.dll" _
                        (ByVal id As Integer, ByVal startchn As Integer,
                         ByVal counts As Integer, counterval As Long) As Integer
```

VC++: (see E5KDAQ.h)

```
unsigned short    E5K_ReadMultiDICounter (
                  int id,
                  unsigned int startchn,
                  unsigned int counts,
                  unsigned long counterval[]);
```

Parameters:

id	module ID address
startchn	channel no.
counts	how many chanels
counterval[]	points to buffer to store counter value

Return Code:

refer to the *Error code*.

11.80 E5K_WriteDO

Description:

Write data to DO channels

Syntax:

Visual Basic/VB.Net: (see E5KDAQ.bas/E5KDAQ.vb)

```
Declare/public Function E5K_WriteDO Lib "E5KDAQ.dll"
                        (ByVal id As Integer, ByVal dodata As Long) As Integer
```

VC++: (see E5KDAQ.h)

```
unsigned short    E5K_WriteDO (int id, unsigned long dodata);
```

Parameters:

id	module ID address
dodata	32-bit DO data, bit-n of dodata represents DO channel n bit-n=0 inactive, bit-n=1 active

Return Code:

refer to the *Error code*.

11.81 E5K_ReadDOStatus

Description:

Read DO status

Syntax:

Visual Basic/VB.Net: (see E5KDAQ.bas/E5KDAQ.vb)

```
Declare/public Function E5K_ReadDOStatus Lib "E5KDAQ.dll" _
                        (ByVal id As Integer, ByRef doval As Long) As Integer
```

VC++: (see E5KDAQ.h)

```
unsigned short    E5K_ReadDOStatus(int id, unsigned long *doval);
```

Parameters:

id	module ID address
doval	points to a 32-bit data buffer to store DO status, bit-n of doval represents DO channel n bit-n=0 inactive, bit-n=1 active

Return Code:

refer to the *Error code*.

11.82 E5K_SetDOSingleChannel

Description:

Set single DO channel

Syntax:

Visual Basic/VB.Net: (see E5KDAQ.bas/E5KDAQ.vb)

```
Declare/public Function E5K_SetDOSingleChannel Lib "E5KDAQ.dll" _
                        (ByVal id As Integer, ByVal chno As Integer, ByVal status As Byte) As Integer
```

VC++: (see E5KDAQ.h)

```
unsigned short E5K_SetDOSingleChannel (int id, unsigned int chano, unsigned char status);
```

Parameters:

id	module ID address
chano	DO channel number (0~31)
status	status=0 deactivate DO channel, status=1 activate DO channel

Return Code:

refer to the *Error code*.

11.83 E5K_SetDOPulseWidth

Description:

Set pulse high/low width of specified DO channel

Syntax:

Visual Basic/VB.Net: (see E5KDAQ.bas/E5KDAQ.vb)

```
Declare/public Function E5K_SetDOPulseWidth Lib "E5KDAQ.dll" _
                        (ByVal id As Integer, ByVal dochn As Integer, ByVal highwidth As Integer,  _
                        ByVal lowwidth As Integer) As Integer
```

VC++: (see E5KDAQ.h)

```
unsigned short      E5K_SetDOPulseWidth (   int id,  unsigned int dochn,
                                            unsigned int highInterval,
                                            unsigned int lowInterval);
```

Parameters:

id	module ID address
dochn	DO channel number (0~31)
highwidth	DO pulse high level width in 0.5msec unit
lowwidth	DO pulse low level width in 0.5msec unit

Return Code:

refer to the *Error code*.

11.84 E5K_ReadDOPulseWidth

Description:

Read pulse high/low width of specified DO channel

Syntax:

Visual Basic/VB.Net: (see E5KDAQ.bas/E5KDAQ.vb)

```
Declare/public Function E5K_ReadDOPulseWidth Lib "E5KDAQ.dll" _
                        (ByVal id As Integer, ByVal dochn As Integer, ByRef highwidth As Long,
                         ByRef Lowwidth As Long) As Integer
```

VC++: (see E5KDAQ.h)

```
unsigned short    E5K_ReadDOPulseWidth(
                  int id,
                  unsigned int dochn,
                  unsigned int *highwidth,
                  unsigned int *Lowwidth);
```

Parameters:

id	module ID address
dochn	DO channel number (0~31)
highwidth	points to 16-bit buffer to store pulse high width in 0.5msec unit
lowwidth	points to 16-bit buffer to store pulse low width in 0.5msec unit

Return Code:

refer to the *Error code*.

11.85 E5K_StartDOPulse

Description:

Start DO pulse output

Syntax:

Visual Basic/VB.Net: (see E5KDAQ.bas/E5KDAQ.vb)

```
Declare/public Function E5K_StartDOPulse Lib "E5KDAQ.dll" _
                        (ByVal id As Integer, ByVal dochn As Integer, ByVal pulses As Long) As Integer
```

VC++: (see E5KDAQ.h)

```
unsigned short    E5K_StartDOPulse (int id,unsigned int Dochn,unsigned int pulses);
```

Parameters:

id	module ID address
dochn	DO channel number (0~31)
pulses	how many pulses

Return Code:

refer to the *Error code*.

11.86 E5K_StopDOPulse

Description:

Stop DO pulse output

Syntax:

Visual Basic/VB.Net: (see E5KDAQ.bas/E5KDAQ.vb)

```
Declare/public Function   E5K_StopDOPulse Lib "E5KDAQ.dll" _
                          (ByVal id As Integer, ByVal dochn As Integer) As Integer
```

VC++: (see E5KDAQ.h)

```
unsigned short            E5K_StopDOPulse (int id,unsigned int dochn);
```

Parameters:

id	module ID address
dochn	DO channel number (0~31)

Return Code:

refer to the *Error code*.

11.87 E5K_ReadDOPulseCount

Description:

Read pulse count value. The pulse count value will start decreasing after calling E5K_StartDOPulse()

Syntax:

Visual Basic/VB.Net: (see E5KDAQ.bas/E5KDAQ.vb)

```
Declare/public Function   E5K_ReadDOPulseCount Lib "E5KDAQ.dll" _
                          (ByVal id As Integer, ByVal dochn As Integer, counts As Long) As Integer
```

VC++: (see E5KDAQ.h)

```
unsigned short            E5K_ReadDOPulseCount (int id,unsigned int dochn,unsigned int *counts);
```

Parameters:

id	module ID address
dochn	DO channel number (0~31)
counts	point to 16-bit buffer to store pulse counter value

Return Code:

refer to the *Error code*.

11.88 E5K_SetDOPowerOnValue

Description:

Set DO power-on value

Syntax:

Visual Basic/VB.Net: (see E5KDAQ.bas/E5KDAQ.vb)

```
Declare/public Function E5K_SetDOPowerOnValue Lib "E5KDAQ.dll" _
                        (ByVal id As Integer, ByVal poweronvalue As Long) As Integer
```

VC++: (see E5KDAQ.h)

```
unsigned short          E5K_SetDOPowerOnValue (int id, unsigned long poweronvalue);
```

Parameters:

id	module ID address
poweronvalue	32-bit DO power-on value

Return Code:

refer to the *Error code*.

11.89 E5K_ReadDOPowerOnValue

Description:

Read DO power-on value

Syntax:

Visual Basic/VB.Net: (see E5KDAQ.bas/E5KDAQ.vb)

```
Declare/public Function E5K_ReadDOPowerOnValue Lib "E5KDAQ.dll" _
                        (ByVal id As Integer, ByRef Dopoweron As Long) As Integer
```

VC++: (see E5KDAQ.h)

```
unsigned short          E5K_ReadDOPowerOnValue (int id,unsigned long *PowerOnValue);
```

Parameters:

id	module ID address
poweronvalue	points to a 32-bit buffer to store DO power-on value

Return Code:

refer to the *Error code*.

11.90 E5K_ReadDIOActiveLevel

Description:

Read DI/DO active level options

Syntax:

Visual Basic/VB.Net: (see E5KDAQ.bas/E5KDAQ.vb)

```
Declare/public Function E5K_ReadDIOActiveLevel Lib "E5KDAQ.dll" _
                        (ByVal id As Integer, ByRef DIActiveoption As Byte, _
                         ByRef DOActiveoption As Byte) As Integer
```

VC++: (see E5KDAQ.h)

```
unsigned short    E5K_ReadDIOActiveLevel (
                  int id,
                  unsigned char *DIActiveoption,
                  unsigned char *DOActiveoption);
```

Parameters:

id	module ID address
DIActiveoption	points to 8-it buffer to store DI active status option 0 or 1 (see Table 11.91-1)
DOActiveoption	points to 8-it buffer to store DO active status option 0 or 1 (see Table 11.91-2)

Return Code:

refer to the *Error code*.

11.91 E5K_SetDIOActiveLevel

Description:

Set DI/DO active level options

Syntax:

Visual Basic/VB.Net: (see E5KDAQ.bas/E5KDAQ.vb)

```
Declare/public Function E5K_SetDIOActiveLevel Lib "E5KDAQ.dll" _
                        (ByVal id As Integer, ByVal DIActiveoption As Byte,  _
                         ByVal DOActiveoption As Byte) As Integer
```

VC++: (see E5KDAQ.h)

```
unsigned short    E5K_SetDIOActiveLevel(
                  int id,
                  unsigned char   DIActiveoption,
                  unsigned char DOActiveoption);
```

Parameters:

id	module ID address
id	module ID address
DIActiveoption	DI active state option 0 or 1 (see Table 11.91-1)
DOActiveoption	DO active state option 0 or 1 (see Table 11.91-2)

Return Code:

refer to the *Error code*.

Module name	DI Active option	Description
EDAM8060	0	Active state ,if input open or high voltage Inactive state ,if input short or low voltage
	1	Active state ,if input short or low voltage Inactive state ,if input open or high voltage

Table 11.91-1

Module name	DO Active option	Description
EDAM8060	0	Active state ,if relay close Inactive state ,if relay open
	1	Active state ,if relay open Inactive state ,if relay close

Table 11.91-2

Chapter 12 E5KDAQ.DLL Error code

Error code	Description:
00	No Error
01	Device not supported
02	Device is not existed
03	Device driver not activated
04	Device driver open fail
05	Device time out
06	Device response Error
07	Invalid driver version
08	Invalid ID Number
09	Device ID overlapped
10	Invalid interface type
11	Invalid Pass Word or password not be verified
12	Invalid ASCII Command
13	Interrupt Already enabled
14	No Interrupt Data
15	Arguments Out Of Range
16	Invalid Port Number
17	Invalid DO Data
18	Invalid Digital Channel Number
19	Invalid Timer Value
20	Invalid Timer Mode
21	Invalid Counter Number
22	Invalid Counter Value
23	Invalid Counter Mode
24	Invalid A/D Filter Type
25	Invalid A/D Mode
26	Invalid A/D channel number
27	Invalid A/D Gain
28	Invalid A/D Range
29	Invalid A/D count Value
30	Invalid A/D Scan Rate
31	A/D FIFO Half Not Ready
32	Invalid D/A channel number
33	Invalid D/A Value
34	Invalid Debounce Mode
35	Invalid Debounce Time
36	Invalid Modbus Function
37	Invalid Modbus Start Address
38	Modbus Address Out Of Range
39	Modbus Range over 32 Channel
40	WINSCK Not Opened
41	Windows winsock2 start up error
42	Invalid IP address
43	Can Not Create TCP Socket
44	Can Not Create UDP Socket
45	Can Not Set TCP/IP Timeout
46	Can Not Send Package To Destination
47	No Package Received Until Timeout
48	Unable To Read Stream Data
49	No Connection To Remote IP Address
50	Alarm Event Buffer Empty
51	Stream Event Buffer Empty
52	Unable To Allocate Memory

Error code	Description:
53	Can Not Ping Remote IP Address
54	Check Sum /CRC error
55	IP not in then subnet
56	COMM port already open
57	No enough buffer size to receive data
58	Error Code Out of Range

Chapter 13 Event/Stream Interrupt structure

13.1 Event interrupt structure

```
typedef struct EVENT_INTERRUPT_INFO
{
    unsigned int    szID;           //the ID address which cause the alarm interrupt
    unsigned int    wIntType;       //0= DI interrupt,1= AD_INT_TYPE
    unsigned int    wChno;          //Event channel number
    unsigned int    wStatus;        //=0 for low to high interrupt for DI or high alarm for AI channel
                                    //=1 for high to low interrupt for DI or low alarm for AI channel
    double fAddata;                 //AD data if AD alarm occured
} DEVICE_INTERRUPT_INFO;
```

When event occurred, E5KDAQ.DLL will transfer argument with structure **EVENT_INTERRUPT_INFO** to callback function

13.2 Stream interrupt structure

```
typedef struct STREAM_INTERRUPT_INFO
{
    unsigned int    wszID;          //the ID address which cause the alarm change
    Unsigned long   dwDi;           //digital input status
    Unsigned long   dwDiLatch;      //digital input latch status
    Unsigned long   dwDiCount[32];  //digital input counter value
    Unsigned long   dwDo;           //digital output status
    double          fAiNorValue[17];    //analog input normal value
    double          fAiMaxValue[16];    //analog input maximum value
    double          fAiMinValue[16];    //analog input minimum value
    unsigned int    wAiHighAlarmstatus; //analog input high alarm status
    unsigned int    wAiLowAlarmstatus;  //analog input low alarm status
    unsigned int    wAiBurnOut ;        //analog input burn-out status(5019,EDAM5015 only)
    double          fCJCTemperature;    //cold junction temperature in 0.1C unit (5019 only)
    double          fAoValue[16];       //analog output value
} STREAM_INTERRUPT_INFO;
```

When received active-stream data, E5KDAQ.DLL will transfer argument with structure **STREAM_INTERRUPT_INFO** to callback function

Chapter 14 E5KDAQ ActiveX control

14.1 Properties of E5KDSAQ ActiveX control

Name	Type	Description	Model(s)
AIChannelIndex	short	Specifies the analog input channel to perform other AI properties read/write operation.	5015,,5017,5019
AINormalValue	double	Normal voltage of specifies the analog channel	5015,5017,5019
AIMaximumValue	double	Maximal voltage of specifies the analog channel	5015,5017,5019
AIMinimumValue	double	Minimal voltage of specifies the analog channel	5015,5017,59019
AILowAlarmStatus	long	Return the low alarm status of specifies the analog channel (1=Alarm occurred/ 0=No alarm)	5015,5017,5019
AIHighAlarmStatus	long	Return the high alarm status of specifies the analog channel (1=Alarm occurred/ 0=No alarm)	5015,5017,5019
AIBurnOutStatus	long	Return the Burnout status of specifies the analog channel (1=open/ 0=normal)	5015,5019
AIChannelEnable	long	Enable/disable AI channels	5015,5017,5019
AIChannels	long	Return the total AI channels of model	
AOChannelIndex	short	Specifies the analog output channel to perform other properties read/write operation.	Reserved
AOChannels	long	Return the total AO channels of model	
AOValue	double	Set the analog output voltage	All models
AIColdJunction	double	Return the cold junction temperature	reserved
AlarmEventADValue	double	Return the alarm AD value	5015,50175019
AlarmEventChannel	short	Return the alarm channel number	5015,5017,5019
AlarmEventID	short	Return the ID address of alarm model	5015,5017,5019
AlarmEventIP	string	Return the IP address of alarm model	All models
AlarmEventStatus	short	Return 1 if AD low alarm or DI high to low return 0 if AD high alarm or DI low to high	All models
AlarmEventType	short	Return 1 if AD type alarm event occurred return 0 if DI type alarm event occurred	All models
CheckSumCRC	Boolean	Enable/disable CheckSum/CRC	All models
DIChannelIndex	short	Specifies the digital input channel to perform other DI properties read/write operation.	5017,5019,5028, 5029,5060
DIChannels		Return the total DI channels of model	5017,5019,5028, 5029,5060
DIounterValue	long	Return the counting value for the specified DI channel which functions in "Count/Frequency mode"	5017,5019,5028, 5029,5060
DILatchStatus	long	Return the latch status for the specified DI channel which functions in "Lo-Hi/Hi-Lo latch mode" (1=Latched/ 0=No latched)	5017,5019,5028, 5029,5060
DIStartCount	boolean	Start/stop counting for the specified DI channel which functions in "Count/Frequency mode" (True=Start/ 0=Stop)	5017,5019,5028, 5029,5060
DIStatus	long	Return the status for the specified DI channel which functions in "DI mode" (1=Active/ 0=Inactive)	5017,5019,5028, 5029,5060
DOChannelIndex	short	Specifies the digital output channel to perform other DO properties read/write operation.	5017,5019,5028, 5029,5060
DOChannels	short	Return the total DO channels of model	5017,5019,5028, 5029,5060
DOOulseCounts	long	Set the output count value for the specified DO channel which	5017,5019,5028,

		functions in "Pulse output mode"	5029,5060
DOStatus	long	Return/set the status for the specified DO channel which functions in "D/O mode" (1=Active/ 0=Inactive)	5017,5019,5028, 5029,5060
COMBaudRate	long	Return/set COM port baud rate	All models
CommunicationType	short	Return/set communication interface	All models
StreamEventID	short	Return ID address of module which generate stream data	All models
StreamIP	string	Return IP address of module which generate stream data	All models
Version	string	Return the version of ActiveX control (E5KDAQ.OCX)	All models
LastError	short	Return the Error code of operation	All models
LastErrorDescription	string	Return the error description	All models
MoudleID	short	Return the module ID number	All models
ModuleIP	string	Set the remote module IP address	All models
ModuleName	string	Return the module name	All models
ConnectionTimeOut	long	Return or set the TCP/IP Timeout (ms)	All models
ReceiveTimeOut	long	Return or set the TCP/IP or COM receive Timeout (ms)	All models
SendTimeOut	long	Return or set the TCP/IP Send Timeout (ms)	All models
UpdatePeriod	long	Return/set data update time period(ms)	All models

14.2 Methods of E5KDAQ ActiveX control

Name	Arguments	Return	Description
Open	None	None	Open E5kDAQ.OCX to start operation (Must be called before accessing properties at run time)
Close	None	None	Close E5KDAQ.OCX(Must be called before terminating the APP)
ReadAlarmEventData	None	boolean	Return the status of alarm data TRUE=alarm data ready in queue, FLASE=no alarm data
ReadStreamData	None	boolean	Return the status of stream data TRUE=stream data ready in queue, FLASE=no stream data
SendASCII	string		Send ASCII command
RecvASCII	None		Receive ASCII command
SendHEX	short Buffer[] short length	None	Send Hex data in buffer[]
RecvHEX	short Buffer[] short buffersize	integer	Receive hex data and store into buffer[] return the data length
StartAlarmEvent	None	long	Start alarm interrupt return 0 if error occurred, or handle of interrupt
StartStreamEvent	None	long	Start stream interrupt return 0 if error occurred, or handle of interrupt

14.3 Events of E5KDAQ ActiveX control

Name	Arguments	Return	Description
OnError	short ErrCode(out) string Errmsg(out)	None	be called when error occurred

Chapter 15 Firmware Update

The EDAM-5000 utility provides on-board firmware update tool that can help you to update firmware through USB interface quickly.

The following steps show you how to update firmware

1. Set Module ID address to 3FH (A0,A1,A2, A3,A4,A5 to "ON" position)

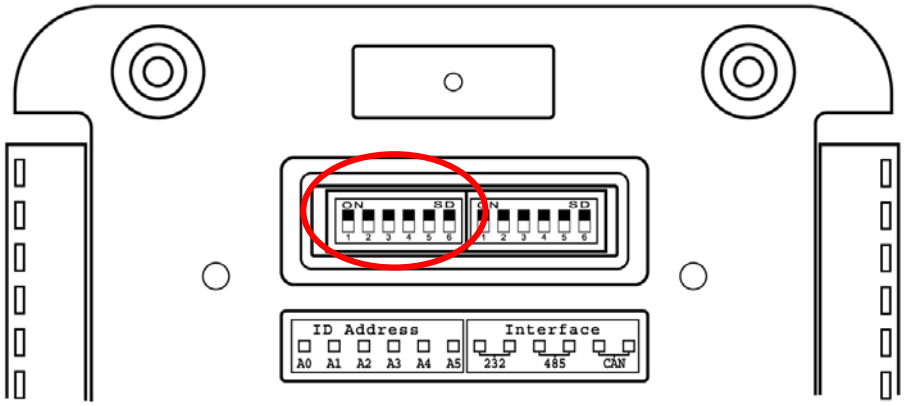


Figure 1

Where

A0=bit 0 of ID address

A1=bit 1 of ID address

A2=bit 2 of ID address

A3=bit 3 of ID address

A4=bit 4 of ID address

A5=bit 5 of ID address

2. Connect EDAM module to USB hub

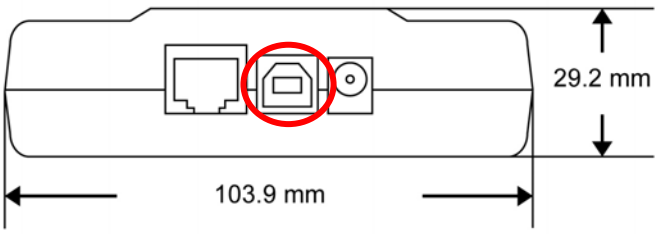


Figure 2

3. Executes provided EDAM5000 utility called "E5KUtility.exe" (see Figure 3)

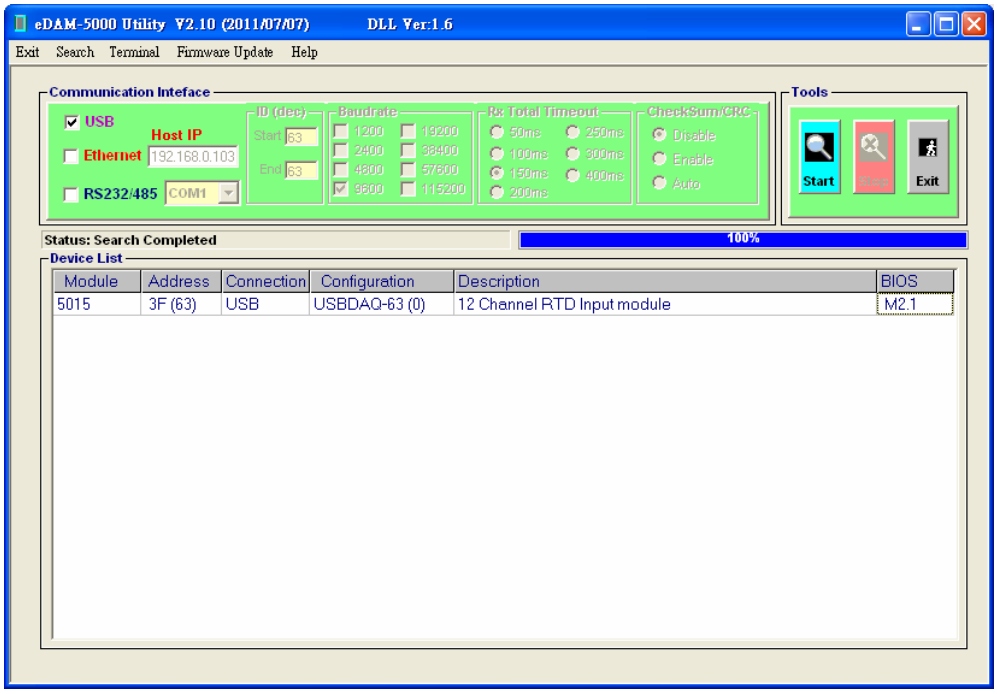


Figure 3

4. Click "Firmware Update" in the menu bar (see Figure 4)

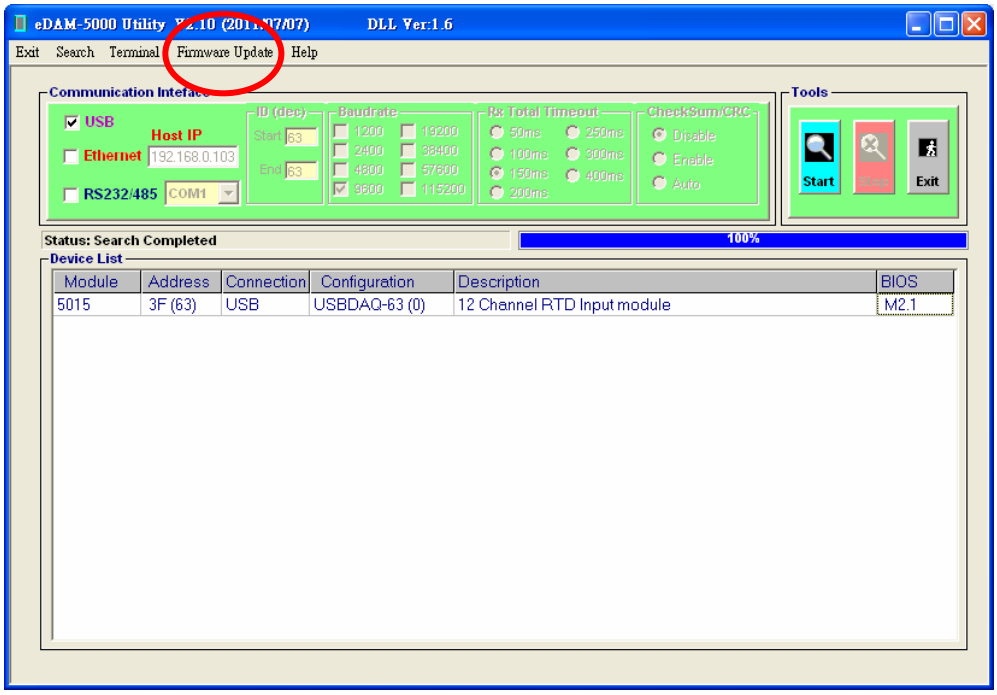


Figure 4

5. Click "Load File" button to load firmware file and select the firmware file your are going to update (see Figure 5 and Figure 6)

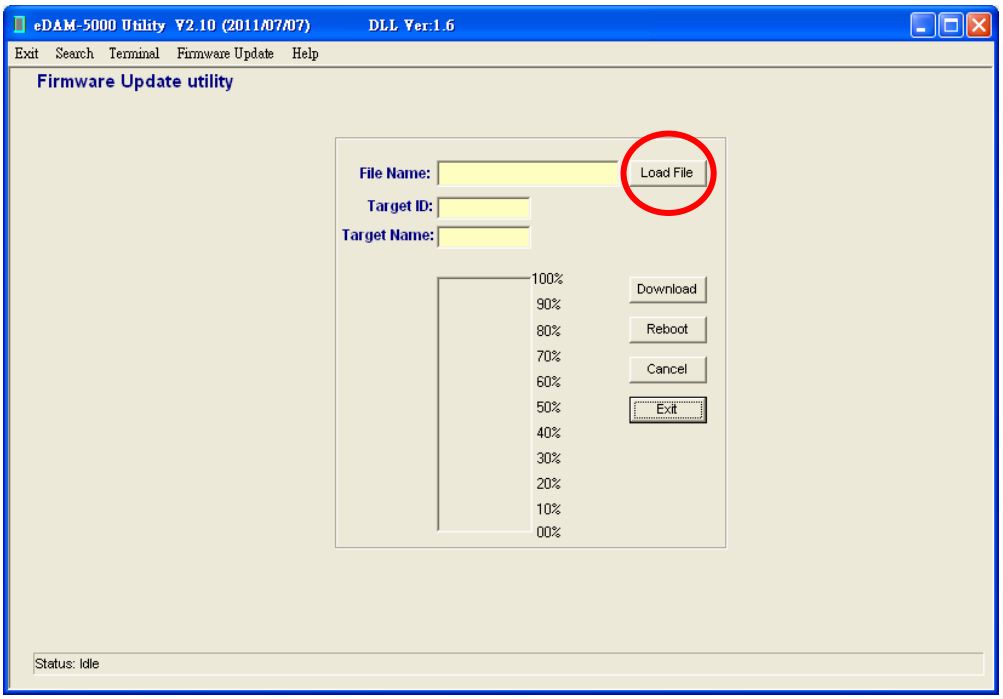


Figure 5

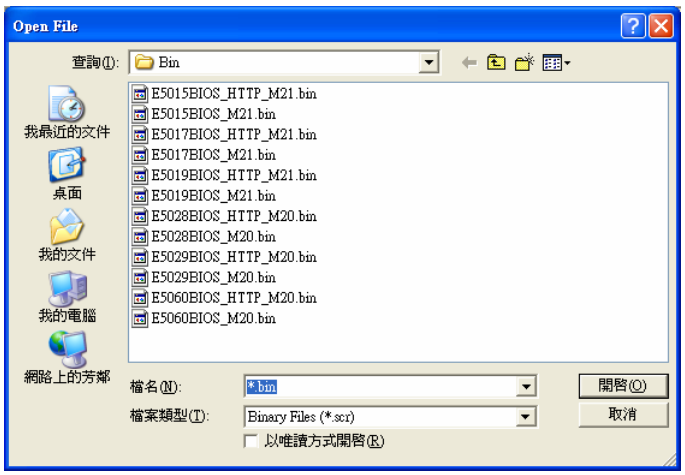
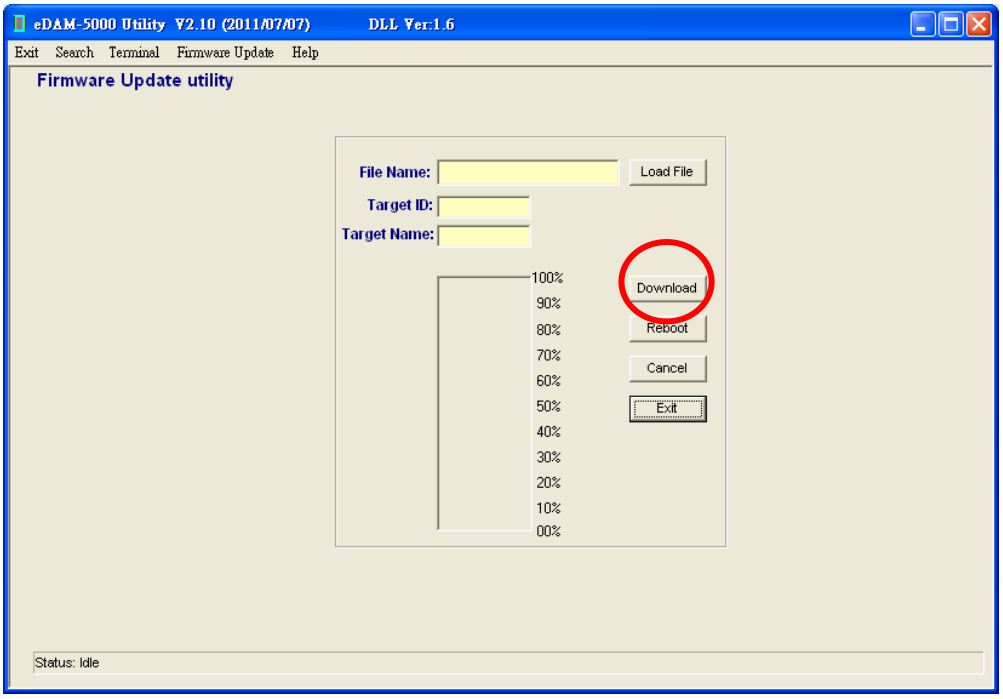


Figure 6

6. Click "Download" button to start to update firmware (see Figure 7)



♦ Figure 7

♦

7. The Utility is searching module. Power-off /on the module or press "Reset button" at the down side of the module to reboot the module. (see Figure 8)

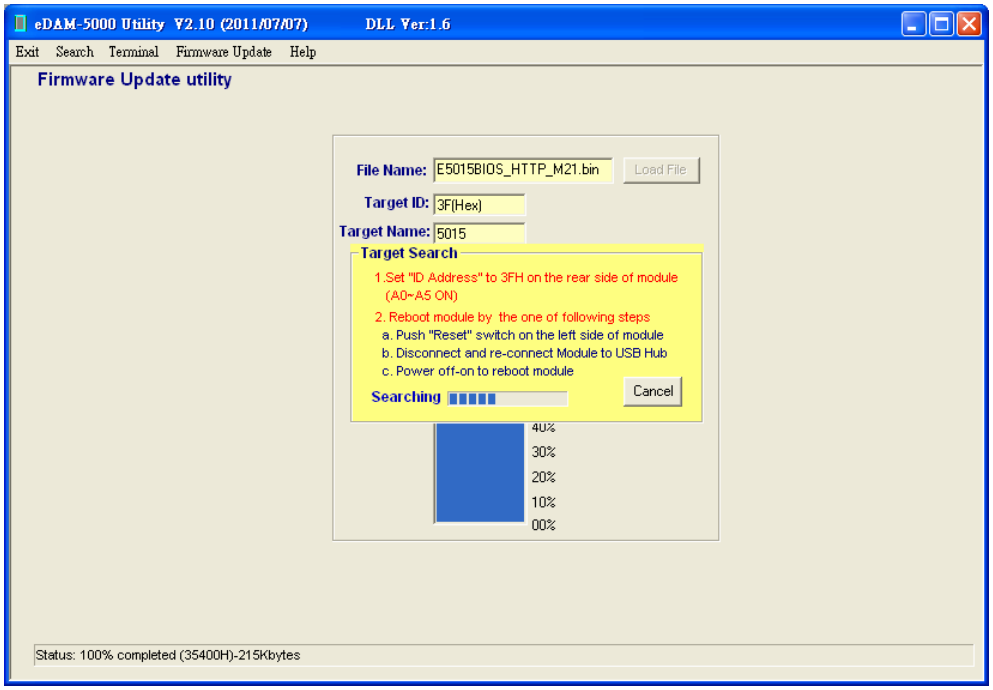


Figure 8

8. If module found. The "Target ready" window pop up (see Figure 9) .Click "OK" button to start

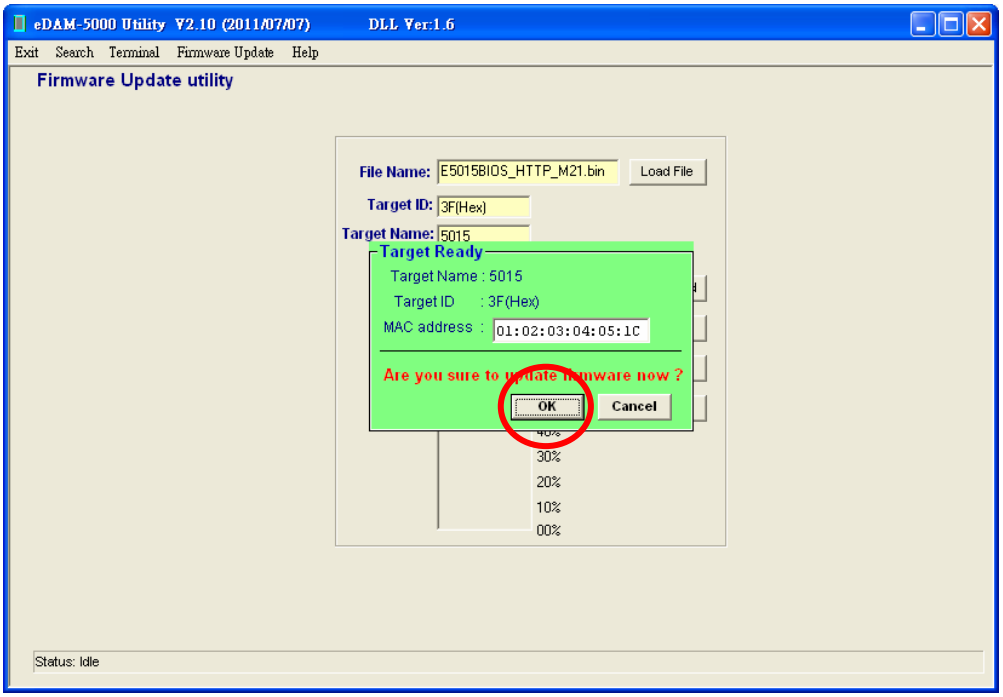


Figure 9

9. The progress bar shows the progress of updating firmware(see Figure 10)

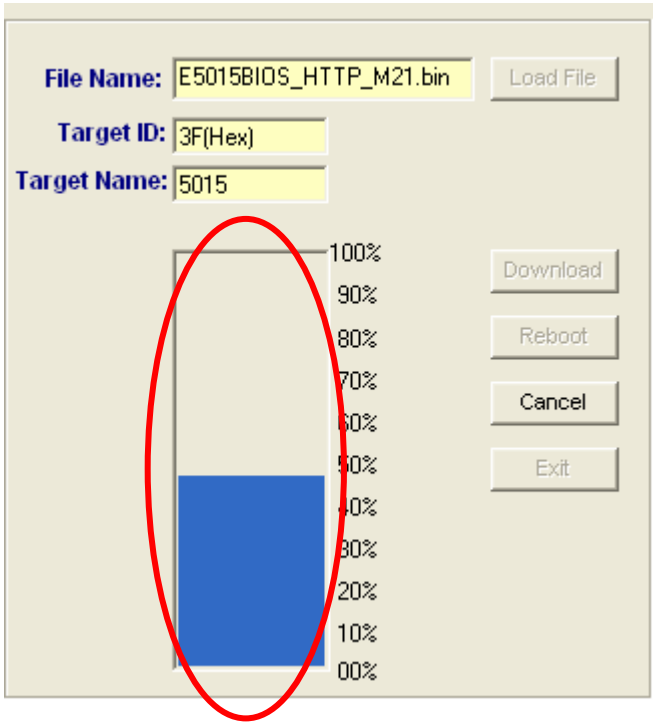


Figure 10

10. Click "Reboot" button to reboot the module