ADAM-5511
User’s Manual

Support Firmware 1.01 or above

Contents

figures

Tables

Chapter 1

Introduction

Introduction

1.1 Standalone Data Acquisition and Control System

As a result of the growth of industrial automation technology, there are countless control systems designed by different venders and different technical bases. Most of them are closed systems with poor integration capabilities, and users have had difficulties finding suitable solutions to these issues. The ADAM-5511 adopts Modbus/RTU protocol, which is a very popular low-cost industrial standard for data communication, a perfect fit for each user's needs. This powerful full-featured stand-alone controller is very easy to learn and use, not only saving an engineer's time and effort, but also increasing the reliability in your versatile SCADA.

1.2 Features

1.2.1 Control flexibility with C programming

The ADAM-5511 is a compact PC in its own right and includes an 80188 CPU and a built-in ROM-DOS operating system. It can be used in the say way one uses an x86 PC at the office. The ADAM-5511 can be controlled by programs written in C language.

Given the prevalence of C language programming tools, this is a distinct advantage for many users and can result in a very short learning curve and modest training expense requirements. See Chapter 3 ADAM-5511 System for detailed technical specifications.

1.2.2 RS-232/485 communication ability

The ADAM-5511 has three serial communication ports, giving it excellent communications abilities. This enables it to control net-worked devices. Among its three ports, COM1 is a dedicated RS-232 port and COM2 is a dedicated RS-485 port. These two ports allow the ADAM-5511 to satisfy diverse communication and integration demands within Modbus protocol. COM3 is a spare programming port for downloading or transferring executable programs from a host PC. It can also be used as an RS-232 communication port.

1.2.3 Complete set of I/O modules for total solutions

The ADAM-5511 uses a convenient backplane system common to the ADAM-5000 series. Advantech's complete line of ADAM-5000 modules integrate with the ADAM-5511 to support your applications.

A full range of digital modules support 10 to 30 V DC I/O and relay outputs. A set of analog modules provide 16-bit resolution and programmable input and output (including bipolar) signal ranges. For details, refer to **Chapter 4 I/O Modules.**

A complete set of C language I/O subroutines are included in the ADAM-5511's function library to reduce programming effort. Users can easily call on these subroutines to execute the ADAM-5511's I/O functions while programming in Borland C 3.0 languages. For a detailed description, refer to **Chapter 6 Function Library.**

1.2.4 Built-in real-time clock and watchdog timer

The micro-controller also includes a real-time clock and watchdog timer. The real-time clock records events while they occur. The watchdog timer is designed to automatically reset the microprocessor if the system fails. This feature greatly reduces the level of maintenance required and makes the ADAM-5511 ideal for use in applications which require a high level of system stability.

Chapter 2

System Specifications

System Specifications

2.1 Overview

The ADAM-5511 is a PC-based programmable micro-controller for standalone data acquisition and control which can control, monitor and acquire data through multi-channel I/O modules. Its IBM-PC compatible hardware and operating system runs programs written in both assembly and high level languages. Each system can handle up to four I/O modules (up to sixty-four I/O points). The system also provides serial communication ports (RS-232/485), allowing the system to communicate with other devices for versatile applications.

2.2 Major Features

The ADAM-5511 system consists of two major components: the main unit and I/O modules. The main unit consists of a CPU card, a power regulator, a 4-slot base, two serial communications ports and a programming port. It has the following major features:

Built-in 80188 CPU and ROM-DOS operating system

ADAM-5511's CPU card includes an 80188 microprocessor. Its ROM-DOS operating system is an MS-DOS compatible system. It provides all the basic functions of MS-DOS except the BIOS. Users can run standard PC software or application programs written in high level languages under this ROM-DOS environment.

Built-in ROM and Flash disk for programming

The ADAM-5511 has a built-in flash ROM, SRAM and flash disk. The system provides 400 KB of free flash disk to allow users to download programs. There are also 60 KB of free SRAM with battery backup to provide the memory needed for temporary variable storage in user's programs.

Built-in RS-232/485 communication ports

The ADAM-5511 has two serial communications ports to enable the controller to communicate with other devices in your applications. The COM1 port is dedicated as an RS-232 interface. The COM2 port is dedicated as the RS-485 port. This unique design makes the controller suitable for use in a variety of applications.

3-way isolation and watchdog timer

Electrical noise can enter a system in many different ways. It may enter through an I/O module, a power supply connection or the communication ground connection. The ADAM-5511 system provides isolation for I/O modules (3000 V_{DC}), communication connections (2500 V_{DC}), and communication power connections (3000 V_{DC}). The 3-way isolation design prevents ground loops and reduces the effect of electrical noise on the system. It also offers better surge protection to prevent dangerous voltages or spikes from harming your system. The system also has a watchdog timer to monitor the microprocessor. The watchdog timer automatically resets the microprocessor in the ADAM-551 if the system fails.

2.3 Technical Specifications of the ADAM-5511 System

2.3.1 System

- CPU: 80188-40, 16-bit microprocessor
- Flash ROM: 256 KB (system use only)
- Operating system: ROM-DOS
- Flash disk: 512 KB (400 KB free space for users)
- SRAM: 60 KB battery backup free memory for users
- Timer BIOS: Yes
- Real-time clock: Yes
- Watchdog timer: Yes
- COM1(3F8): RS-232
- COM2(2F8): RS-485
- Programming port (RS-232 interface, DB-9 connector): Tx, Rx, GND

System Specifications

- I/O capacity: 4 modules (limitation: only one ADAM-5024 is allowed in one ADAM-5511 main unit)
- CPU power consumption: 1.0 W
- LED Status display: Power, CPU, Communication, Battery

2.3.2 RS-232 interface (COM1)

- Signals: TxD, RxD, GND
- Mode: Asynchronous full duplex, point to point
- Connector: DB-9 pin
- Transmission speed: Up to 115.2 Kbps
- Max transmission distance: 50 feet (15.2 m)

2.3.3 RS-485 interface (COM2)

- Signals: DATA+, DATA-
- Mode: Half duplex, multi-drop
- Connector: Screw terminal
- Transmission speed: Up to 115.2 Kbps
- Max transmission distance: 4000 feet (1220 m)

2.3.4 RS-232 programming port (COM3)

- Signals: Tx, Rx, GND
- Mode: Asynchronous, point to point
- Connector: DB-9 pin
- Transmission speed: Up to 115.2 Kbps
- Max transmission distance: 50 feet (15.2 m)

2.3.5 Remote I/O Modules

- Nodes: 32 (At most 16 AI/O modules allowed to be installed) Isolation
- Communication Power: 3000 V DC

- Input/Output: 3000 V DC
- Communication: 2500 V DC (COM2 only)

2.3.6 Power

- Unregulated +10 to +30 V DC
- Protected against power reversal
- Power consumption: 2.0 W

2.3.7 Mechanical

- Case: KJW with captive mounting hardware
- Plug-in screw terminal block:

  Accepts 0.5 mm 2 to 2.5 mm 2, or 1 - #12 or 2 - #14 to #22 AWG wires.

2.3.8 Environment

- Operating temperature: -10° to 70° C (14° to 158° F)
- Storage temperature: -25° to 85° C (-13° to 185° 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.*

System Specifications

Dimensions

The following diagrams show the dimensions of the system unit and an I/O unit. All dimensions are in millimeters.

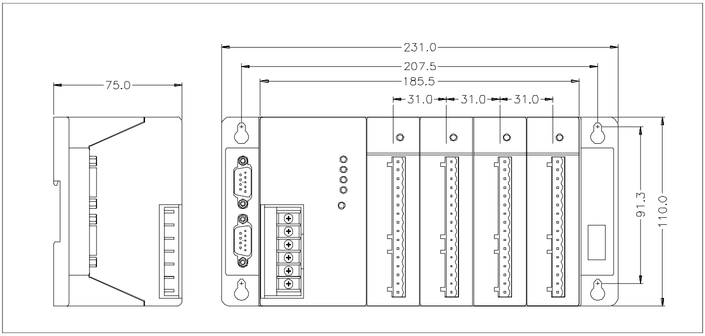


Figure 2-1: ADAM-5511 system & I/O module dimensions

2.4 Basic Function Block Diagram

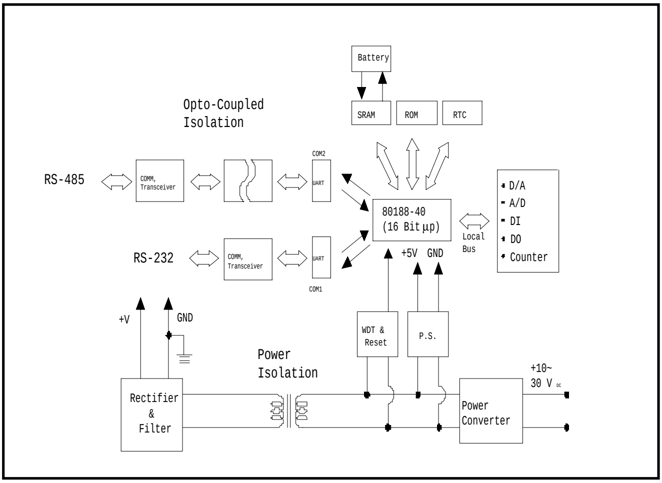


Figure 2-2: Function block diagram

Chapter 3

Installation Guidelines

Installation Guidelines

This chapter explains how to install an ADAM-5511 stand-alone controller. A quick hookup scheme is provided that lets you easily configure your system before implementing your application to it.

3.1 Module Installation

When inserting modules into the system, align the PC board of the module with the grooves on the top and bottom of the system. Push the module straight into the system until it is firmly seated in the backplane connector. Once the module is inserted into the system, push in the retaining clips (located at the top and bottom of the module) to firmly secure the module to the system.

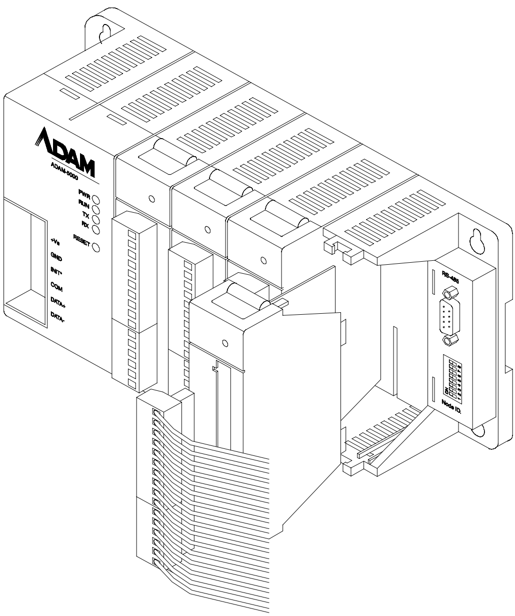


Figure 3-1: Module alignment and installation

3.2 I/O Slots and I/O Channel Numbering

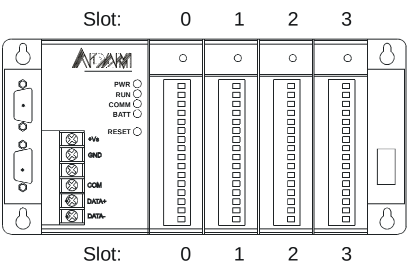
The ADAM-5511 system provides 4 slots for use with I/O modules. The I/O slots are numbered 0 through 3, and the channel numbering of any I/O module in any slot starts from 0. For example, the ADAM-5017 is an 8-channel analog input module. Its input channel numbering is 0 through 7.

3.3 Assigning Address for I/O Modules

Basing on Modbus standard, the addresses of the I/O modules you place into the ADAM-5511 system are defined by a simple rule. Please refer the Figures 3-2 to map the I/O address.

Analog Mapping Table

0	40001~40008
1	40009~40016
2	40017~40024
3	40024~40032



ADAM-5511 contains 4 I/O slots

Digital Mapping Table

0	00001~00016
1	40016~40032
2	40033~40048
3	40049~40064

Figure 3-2 I/O Module Address Mapping

For example, if there is a ADAM-5024 (4-channel AO Module) in slot 2, the address of this module should be 40017~40020.

Note: ADAM-5080 is a special 4-channel counter module. Each channel consists of two words. To read the actual value in HMI software or user's AP, you can follow this formula:

Counter Value= Low Word + High Word * 65536

For Example, when you insert an ADAM-5080 in slot 0, the value of channel 0 in the module should be:

(value of address 40001) + (value of address 40002)*65536

3.4 Mounting

The ADAM-5511 system can be installed on a panel or on a DIN rail.

Panel mounting

Mount the system on the panel horizontally to provide proper ventilation. You cannot mount the system vertically, upside down or on a flat horizontal surface. A standard #7 tating screw (4 mm diameter) should be used.

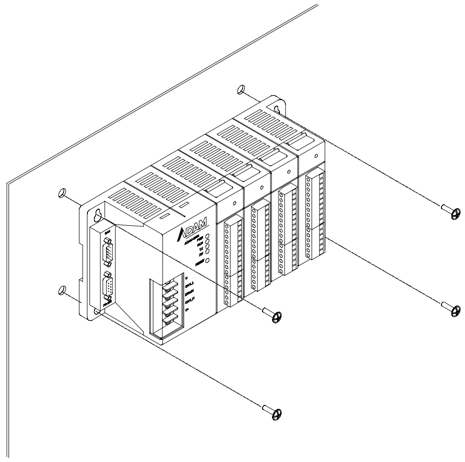


Figure 3-3: ADAM-5511 panel mounting screw placement

DIN rail mounting

The system can also be secured to the cabinet by using mounting rails. If you mount the system on a rail, you should also consider using end brackets at each end of the rail. The end brackets help keep the system from sliding horizontally along the rail. This minimizes the possibility of accidentally pulling the wiring loose. If you examine the bottom of the system, you will notice two small retaining clips. To secure the system to a DIN rail, place the system onto the rail and gently push up on the retaining clips. The clips lock the system on the rail. To remove the system, pull down on the retaining clips, lift up on the base slightly, and pull it away from the rail.

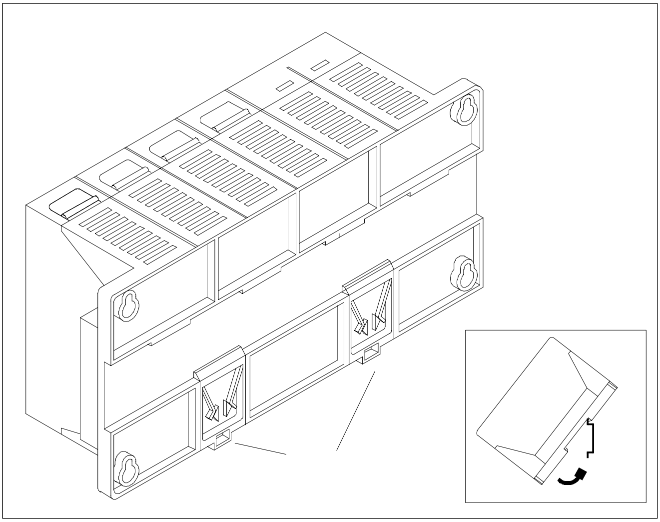


Figure 3-4: ADAM-5511 rail mounting

3.5 Jumper Settings and DIP Switch Settings

This section tells you how to set the jumpers and DIP switches to configure your ADAM-5511 system. It gives the system default configuration and your options for each jumper and dip switch. There are three jumpers (JP2~JP4) on the CPU card, and one 8-pin DIP switch on backplane board. Note that JP1 is actually a connector.

The following figure shows the location of the jumpers:

* JP4 is for battery power ON/OFF

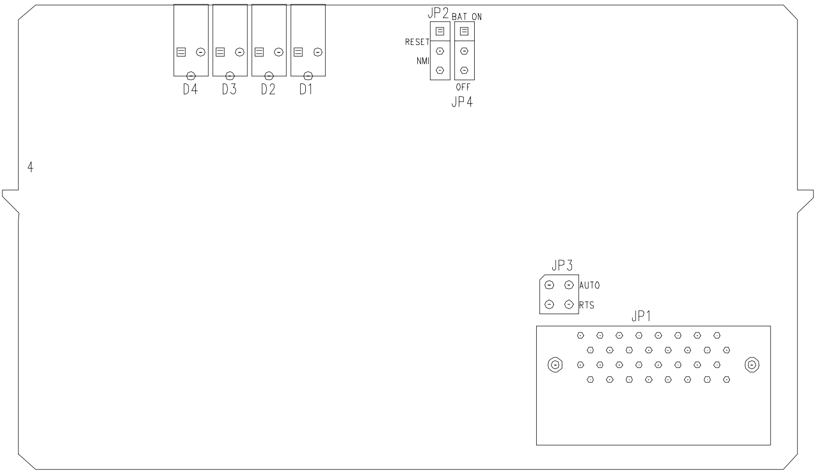


Figure 3-5: Jumper locations on the CPU card

COM2 port RS-485 control mode setting

The COM2 port is dedicated as an RS-485 interface. In an RS-485 network, handshaking signals such as RTS (Request to Send), normally control the direction of the data flow. A special I/O circuit in the ADAM-5511 senses the data flow direction and automatically switches the transmission direction, making handshaking signals unnecessary. Jumper JP3 gives users the option of configuring the COM2 port for automatic control ler RTS control. Jumper settings are shown in Figure 3-6:

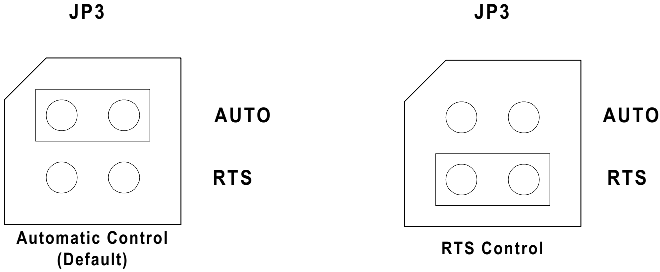


Figure 3-6: COM2 port RS-485 control mode settings (JP3)

Watchdog timer setting

Jumper JP2 on the CPU card lets you configure the watchdog timer to disable mode, reset mode or NMI (Non-maskable interrupt) mode.

Jumper settings are shown below:

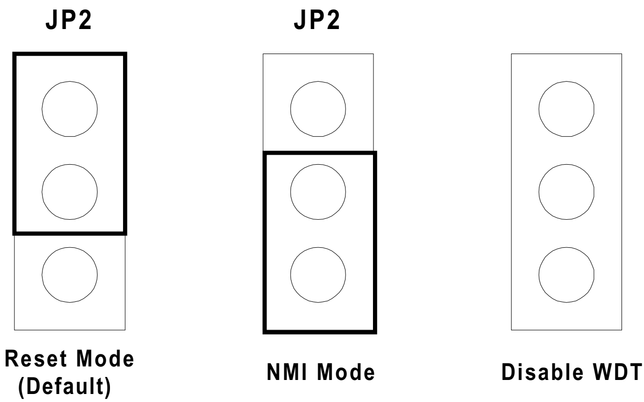


Figure 3-7: Watchdog timer setting

Network address/baud rate setting

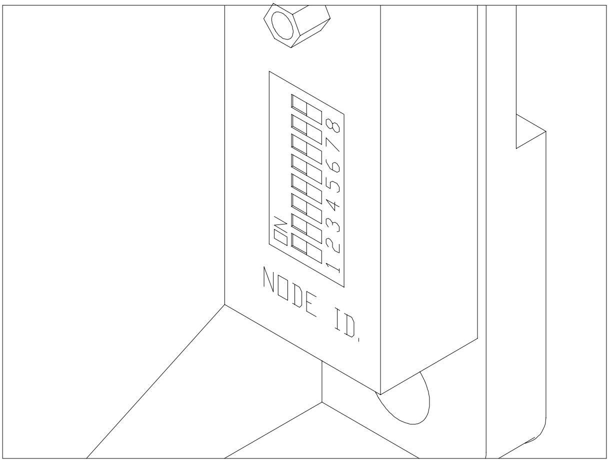


Figure 3-8: ADAM-5511 network address & baud rate DIP switch

There is a set of DIP switch terminals on the right side of the ADAM-5511 back plane.

Node ID:

Dip	1	2	3	4	5
ON	2^0	2^1	2^2	2^3	2^4
OFF	0	0	0	0	0

Set the network address using DIP switches 1~5. Valid settings range from 0 to 31.

Port Select:

Dip	6
ON	Com1 Enable
OFF	Com1 Disable

Baud Rate:

Set DIP switch 6 on when you need to use COM1.

Dip 7	Dip 8	Baud Rate
OFF	OFF	9600
ON	OFF	19200
OFF	ON	38400
ON	ON	115200

DIP switches 7 and 8 are for baud rate settings.

3.6 Wiring and Connections

This section provides basic information on wiring the power supply, I/O units, communication port connections and programming port connection.

Power supply wiring

Although the ADAM-5511 systems are designed for a standard industrial unregulated 24 V_{DC} power supply, they accept any power unit that supplies within the range of +10 to +30 V_{DC}. The power supply ripple must be limited to 200 mV peak-to-peak, and the immediate ripple voltage should be maintained between +10 and +30 V_{DC}. Screw terminals +Vs and GND are for power supply wiring.

Note: The wires used should be at least 2 mm in size.

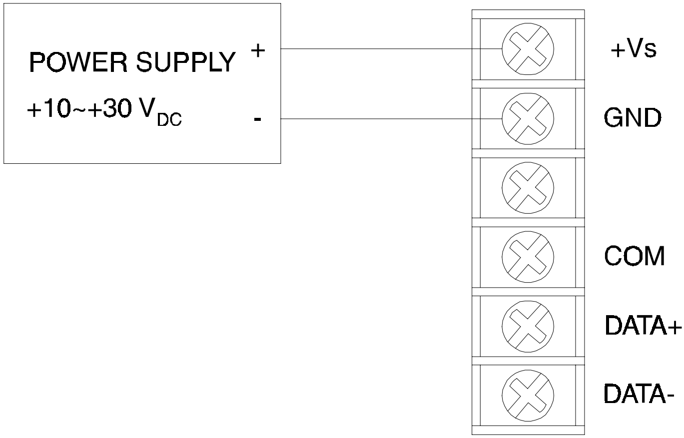


Figure 3-9: ADAM-5511 power wiring

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.

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

Port connection

The ADAM-5511 has three communications ports. These ports allow you to program, configure, monitor, and integrate ADAM-4000 modules as remote I/O.

COM1: (for system configuration and operation) This is an RS-232 interface and only uses TX, RX, and GND signals. The pin assignment of the cross over cable is as follows:

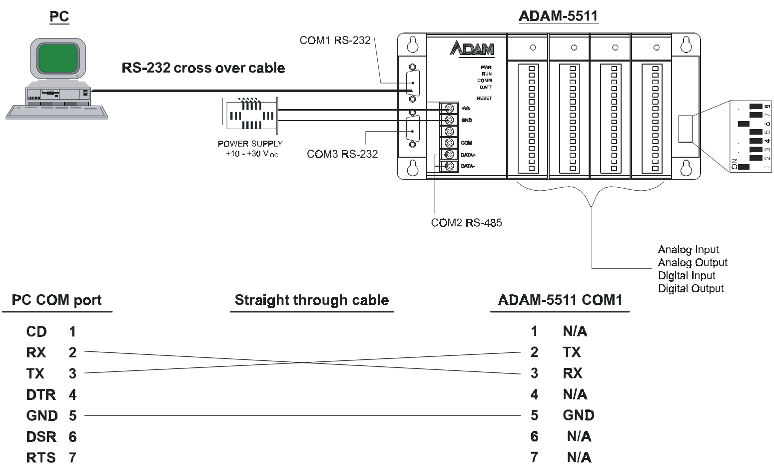


Figure 3-10: COM1 RS-232 Connection

COM2: (for system configuration and operation) The COM2 port is dedicated as an RS-485 interface. Screw terminals DATA- and DATA+ are used for making the COM2 RS-485 connections. Users have to prepare an ADAM-4520 RS232/485 converter for the linkage with PC.

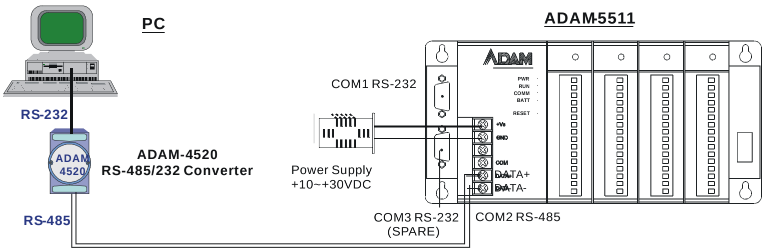
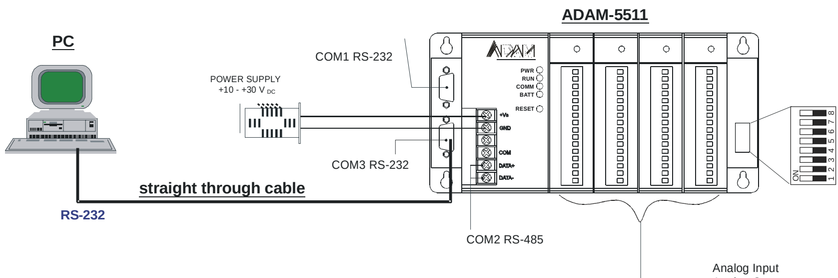


Figure 3-10: COM2 RS-485 Connection

COM3: This is a RS-232 port for Analog Module calibration and configuration. Please set all DIP switch to off before you use this port.



PC COM port		straight through cable	ADAM-5511 COM3	
CD	1		1	N/A
RX	2	———	2	TX
TX	3	———	3	RX
DTR	4		4	N/A
GND	5	———	5	GND
DSR	6		6	N/A
RTS	7		7	N/A
CTS	8		8	N/A
RI	9		9	N/A

Figure 3-11: COM3 RS-232 Connection

3.7 LED Status of the ADAM-5511 Unit

There are four LEDs on the ADAM-5511 front panel. The LED's indicate ADAM-5511's operating status, as explained below:

(1) PWR: power indicator. This LED is on when the ADAM-5511 is powered on.

(2) RUN: program execution indicator. This LED is regularly blinks whenever the ADAM-5511 is executing a program.

(3) COMM: communication indicator. This LED blinks whenever the host PC and the ADAM-5511 are communicating. Please notice: if the host COM port is connected to the ADAM-5511's RS-232 port, this LED will normally be off. On the other hand, if the host COM port is connected to the ADAM-5511's RS-485 port, this LED will normally be on.

(4) BATT: battery status indicator. This LED will be on whenever the SRAM backup battery is low.

Chapter 4

I/O Modules

I/O Modules

4.1 Analog Input Modules

Analog input modules use an A/D converter to convert sensor voltage, current, thermocouple or RTD signals into digital data. The digital data is then translated into engineering units. The analog input modules protect your equipment from ground loops and power surges by providing opto-isolation of the A/D input and transformer based isolation up to 3,000 V_{DC}.

ADAM-5013 3-channel RTD input module

The ADAM-5013 is a 16-bit, 3-channel RTD input module that features programmable input ranges on all channels. This module is an extremely cost-effective solution for industrial measurement and monitoring applications. Its opto-isolated inputs provide 3,000 V_{DC} of isolation between the analog input and the module, protecting the module and peripherals from damage due to high input line voltage.

Note: *Owing to the conversion time required by the A/D converter, the initialization time of each ADAM-5013 module is 5 seconds. Thus the total initialization time will be about 20 seconds if all 4 I/O slots in an ADAM-5000 main unit contain ADAM-5013 modules.*

ADAM-5013

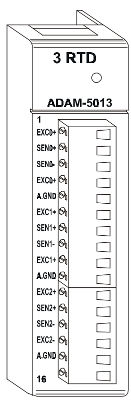
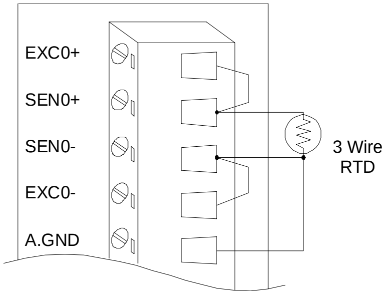


Figure 4-1: *ADAM-5013 module frontal view*

Application wiring



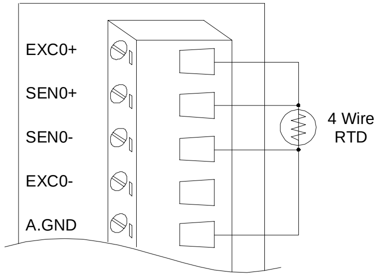


Figure 4-2: *RTD inputs*

Technical specifications of ADAM-5013

Analog input channels	three
Input type	Pt or Ni RTD
RTD type and temperature range	Pt -100 to 100° C a=0.00385 Pt 0 to 100° C a=0.00385 Pt 0 to 200° C a=0.00385 Pt 0 to 600° C a=0.00385 Pt -100 to 100° C a=0.00392 Pt 0 to 100° C a=0.00392 Pt 0 to 200° C a=0.00392 Pt 0 to 600° C a=0.00392 Ni -80 to 100° C Ni 0 to 100° C
Isolation voltage	3000 V_{DC}
Sampling rate	10 samples/sec (total)
Input impedance	2 MΩ
Bandwidth	13.1 Hz @ 50 Hz, 15.72 Hz @ 60 Hz
Input connections	2, 3 or 4 wire
Accuracy	± 0.1% or better
Zero drift	± 0.015 °C/°C
Span drift	± 0.01 °C/°C
CMR@50/60 Hz	150 dB
NMR@50/60 Hz	100 dB
Power consumption	1.2 W

Table 4-1: *Technical specifications of ADAM-5013*

4.2 ADAM-5013 RTD Input Resistance Calibration

1. Apply power to the module and let it warm up for about 30 minutes.
2. Make sure that the module is correctly installed and is properly configured for the input range you want to calibrate. You can use the ADAM utility software to help in this.
3. Connect the correct reference self resistance between the screw terminals of the ADAM-5013 as shown in the following wiring diagram. Table 4-2 below shows the correct values of the span and zero calibration resistances to be connected. Reference resistances used can be from a precision resistance decade box or from discrete resistors with the values 60, 140, 200 and 440 ohms.

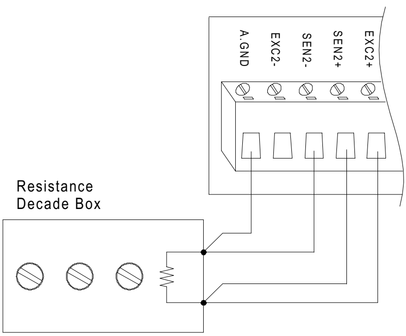


Figure 4-3: *Applying calibration resistance*

4. First, with the correct zero (offset) calibration resistance connected as shown above, issue a Zero Calibration command to the module using the Calibrate option in the ADAM utility software.
5. Second, with the correct span resistance connected as shown above, issue a Span Calibration command to the module using the Calibrate option in the ADAM utility software. Note that the module zero calibration must be completed prior to the span calibration.

Note: *If the above procedure is ineffective, the user must first issue an RTD Self Calibration command $aaSi2 to the module and then complete steps 4 and 5 after self calibration is complete.*

Calibration resistances (ADAM-5013)

Input Range Code (Hex)	Input Range	Span Calibration Resistance	Zero Calibration Resistance
20	Pt, -100 to 100° C A = 0.00385	140 Ohms	60 Ohms
21	Pt, 0 to 100° C A = 0.00385	140 Ohms	60 Ohms
22	Pt, 0 to 200° C A = 0.00385	200 Ohms	60 Ohms
23	Pt, 0 to 600° C A = 0.00385	440 Ohms	60 Ohms
24	Pt, -100 to 100° C A = 0.00392	140 Ohms	60 Ohms
25	Pt, 0 to 100° C A = 0.00392	140 Ohms	60 Ohms
26	Pt, 0 to 200° C A = 0.00392	200 Ohms	60 Ohms
27	Pt, 0 to 600° C A = 0.00392	440 Ohms	60 Ohms
28	Ni, -80 to 100° C	200 Ohms	60 Ohms
29	Ni, 0 to 100° C	200 Ohms	60 Ohms

Table 4-2: *Calibration resistances of ADAM-5013*

ADAM-5017 8-channel analog input module

The ADAM-5017 is a 16-bit, 8-channel analog differential input module that provides programmable input ranges on all channels. It accepts millivolt inputs (±150mV, ±500mV), voltage inputs (±1V, ±5V and ±10V) and current input (±20 mA, requires 125 ohms resistor). The module provides data to the host computer in engineering units (mV, V or mA). This module is an extremely cost-effective solution for industrial measurement and monitoring applications. Its opto-isolated inputs provide 3,000 V_{DC} of isolation between the analog input and the module, protecting the module and peripherals from damage due to high input line voltage. Additionally, the module uses analog multiplexers with active overvoltage protection. The active protection circuitry assures that signal fidelity is maintained even under fault conditions that would destroy other multiplexers. This module can withstand an input voltage surge of 70 Vp-p with ±15 V supplies.

ADAM-5017

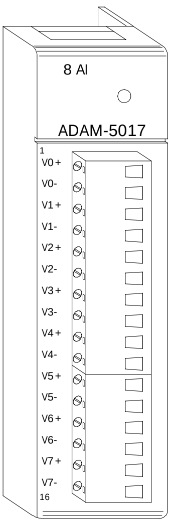


Figure 4-4: *ADAM-5017 module frontal view*

Application wiring

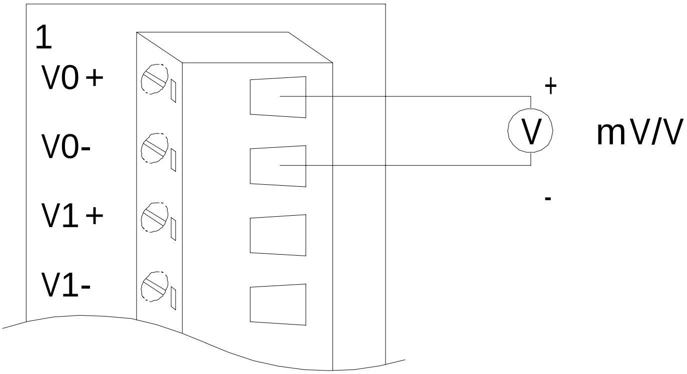


Figure 4-5: *Millivolt and volt input*

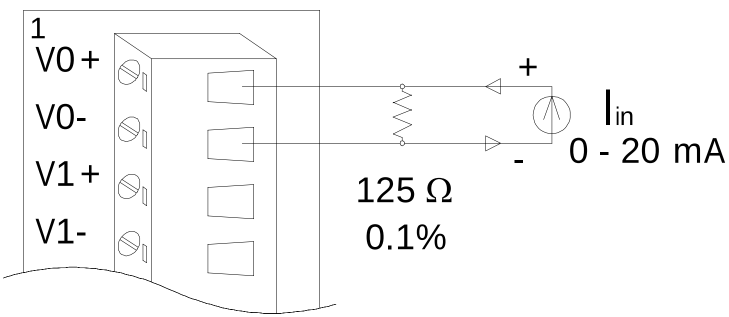


Figure 4-6: *Process current input*

Note: *To keep measurement accuracy please short the channels that are not in use.*

Technical specifications of ADAM-5017

Analog Input Channels	Eight differential
Input Type	mV, V, mA
Input Range	± 150 mV, ± 500 mV, ± 1 V, ± 5 V, ± 10 V and ± 20 mA
Isolation Voltage	3000 V_{DC}
Sampling Rate	10 samples/sec (total)
Analog Input Signal Limit	15 V max.
Max. allowable voltage difference between two connectors in a module	15 V max.
Input Impedance	2 Mohms
Bandwidth	13.1 Hz @ 50 Hz, 15.72 Hz @ 60 Hz
Accuracy	± 0.1% or better
Zero Drift	± 1.5 µV/°C
Span Drift	± 25 PPM/°C
CMR @ 50/60 Hz	92 dB min.
Power Requirements	+ 10 to + 30 V_{DC} (non-regulated)
Power Consumption	1.2 W

Table 4-3: *Technical specifications of ADAM-5017*

ADAM-5017H 8-channel high speed analog input module

The ADAM-5017H is a 12-bit plus sign bit, 8-channel analog differential input module that provides programmable input ranges on each channel. It accepts millivolt inputs (± 500 mV, 0-500 mV), voltage inputs (±1 V, 0-1 V, ±2.5 V, 0-2.5 V, ±5 V, 0-5 V, ±10 V and 0-10 V) and current inputs (0-20 mA and 4-20 mA; requires a 125 ohms resistor). The module provides data to the host microprocessor in engineering units (mV, V or mA) or two's complement format. Its sampling rate depends on the data format received: up to 100 Hz (total). Space is reserved for 125-ohm, 0.1%, 10 ppm resistors (See Figure 4-9). Each input channel has 3000 V_{DC} of optical isolation between the outside analog input line and the module, protecting the module and peripherals from high input line voltages. Additionally, the module uses analog multiplexers with active overvoltage protection. The active protection circuitry assures that signal fidelity is maintained even under fault conditions that would destroy other multiplexers. The analog inputs can withstand a constant 70 Vp-p input with ±15 V supplies.

ADAM-5017H

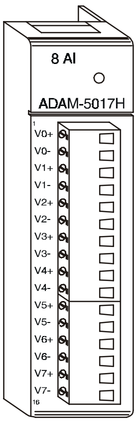


Figure 4-7: *ADAM-5017H module frontal view*

Application wiring

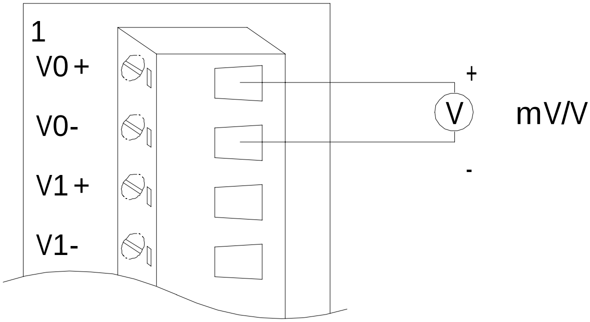


Figure 4-8: Millivolt and volt input

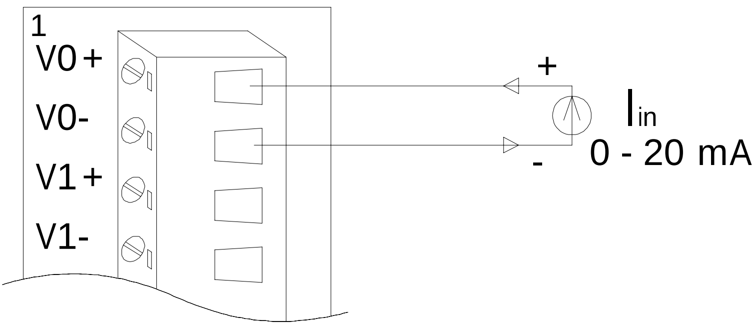


Figure 4-9: Process current input

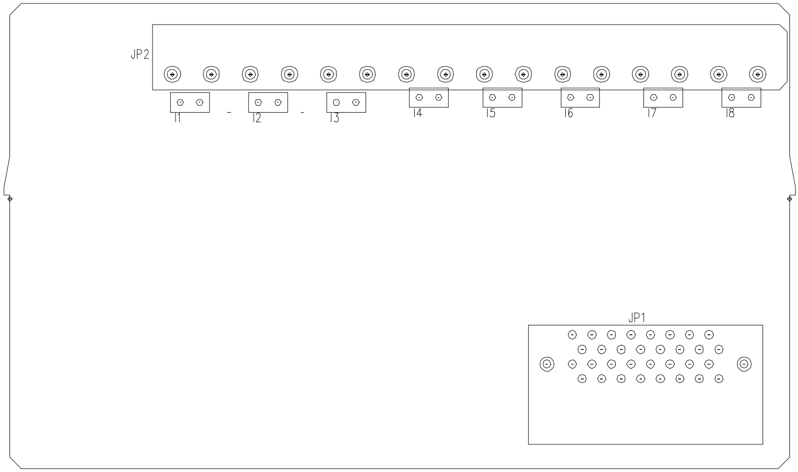


Figure 4-10: *Locations of 125-ohm resistors*

Note: *To maintain measurement accuracy please short channels not in use.*

Technical specifications of ADAM-5017H

Analog Input Channels	8 differential
ADC Resolution	12 bits, plus sign bit
Type of ADC	Successive approximation
Isolation Voltage	3000 V_{DC}
Sampling Rate	100 Hz
Input Impedance	20 Mohms (voltage inputs); 125 ohms (current inputs)
Signal Input Bandwidth	1000 Hz for both voltage inputs and current inputs
Analog Signal Range	±15 V max.
Analog Signal Range for any two measured Pins	±15 V max.
Power Requirements	+10 to +30 V_{DC} (non-regulated)
Power Consumption	1.8 W

Table 4-4: *Technical specifications of ADAM-5017H*

	Input Range	With Overranging	Offset Error @ 25° C	Offset Error @ -10 to +70° C	Gain Error @ 25° C	Gain Error @ -10 to +70° C	Offset Drift	Gain Drift	Display Resolution
Voltage Inputs	0 ~ 10 V	0 ~ 11 V	±1 LSB	±2 LSB	±1 LSB	±2 LSB	17 µV/°C	50 ppm/°C	2.7 mV
	0 ~ 5 V	0 ~ 5.5 V	±1 LSB	±2 LSB	±1.5 LSB	±2 LSB	16 µV/°C	50 ppm/°C	1.3 mV
	0 ~ 2.5 V	0 ~ 2.75 V	±1 LSB	±2 LSB	±1.5 LSB	±2 LSB	20 µV/°C	55 ppm/°C	0.67 mV
	0 ~ 1 V	0 ~ 1.375 V	±1 LSB	±2.5 LSB	±2 LSB	±2.5 LSB	20 µV/°C	60 ppm/°C	0.34 mV
	0 ~ 500 mV	0 ~ 687.5 mV	-	±5 LSB	±3 LSB	±3.5 LSB	20 µV/°C	67 ppm/°C	0.16 mV
	± 10 V	±11 V	±1 LSB	±2 LSB	±1 LSB	±2 LSB	17 µV/°C	50 ppm/°C	2.7 mV
	± 5 V	±0 ~ 5.5 V	±1 LSB	±2 LSB	±1.5 LSB	±2 LSB	17 µV/°C	50 ppm/°C	1.3 mV
	± 2.5 V	±0 ~ 2.75 V	±1 LSB	±2 LSB	±1.5 LSB	±2 LSB	20 µV/°C	55 ppm/°C	0.67 mV
	± 1 V	±0 ~ 1.375 V	±1 LSB	±2.5 LSB	±2 LSB	±2.5 LSB	20 µV/°C	60 ppm/°C	0.34 mV
	± 500 mV	±0 ~ 687.5 mV	-	±5 LSB	±3 LSB	±3.5 LSB	20 µV/°C	67 ppm/°C	0.16 mV
Current Inputs	0 ~ 20 mA	22 mA	±1 LSB	±1 LSB	±1.5 LSB	±2 LSB	nA/°C	ppm/°C	5.3 µA
	4 ~ 20 mA	22 mA	±1 LSB	±1 LSB	±1.5 LSB	±2 LSB	nA/°C	ppm/°C	5.3 µA

Table 4-5: *ADAM-5017H input signal ranges*

ADAM-5018 7-channel thermocouple input module

The ADAM-5018 is a 16-bit, 7-channel thermocouple input module that features programmable input ranges on all channels. It accepts millivolt inputs (±15 mV, ±50 mV, ±100 mV, ±500 mV), voltage inputs (±1 V, ±2.5 V), current input (±20 mA, requires 125 ohms resistor) and thermocouple input (J, K, T, R, S, E, B).

The module forwards the data to the host computer in engineering units (mV, V, mA or temperature °C). An external CJC on the plug-in terminal is designed for accurate temperature measurement.

ADAM-5018

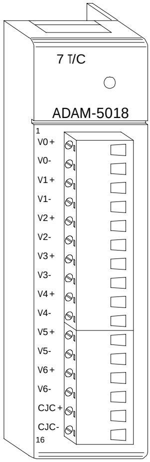


Figure 4-11: *ADAM-5018 module frontal view*

Application wiring

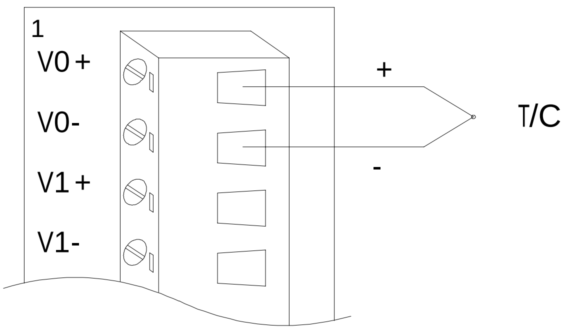


Figure 4-12: *Thermocouple input*

Technical specifications of ADAM-5018

Analog Input Channels	Seven differential
Input Type	mV, V, mA, Thermocouple
Input Range	± 15 mV, ± 50 mV, ± 100 mV, ± 500 mV, ± 1 V, ± 2.5 V and ± 20 mA
T/C Type and Temperature Range	J 0 to 760 °C K 0 to 1370 °C T -100 to 400 °C E 0 to 1400 °C R 500 to 1750 °C S 500 to 1750 °C B 500 to 1800 °C
Isolation Voltage	3000 V_{DC}
Sampling Rate	10 samples/sec (total)
Input Impedance	2 Mohms
Bandwidth	13.1 Hz @ 50 Hz, 15.72 Hz @ 60 Hz
Accuracy	± 0.1% or better
Zero Drift	± 0.3 µV/°C
Span Drift	± 25 PPM/°C
CMR @ 50/60 Hz	92 dB min.
Power Consumption	1.2 W

Table 4-6: *Technical specifications of ADAM-5018*

4.3 Analog Output Modules

ADAM-5024 4-channel analog output module

The ADAM-5024 is a 4-channel analog output module. It receives its digital input through the RS-485 interface of the ADAM-5510 system module from the host computer. The format of the data is engineering units. It then uses the D/A converter controlled by the system module to convert the digital data into output signals.

You can specify slew rates and start up currents through the configuration software. The analog output can also be configured as current or voltage through the software utility. The module protects your equipment from ground loops and power surges by providing opto-isolation of the D/A output and transformer based isolation up to 500 V_{DC}.

Slew rate

The slew rate is defined as the slope indicated the ascending or descending rate per second of the analog output from the present to the required.

ADAM-5024

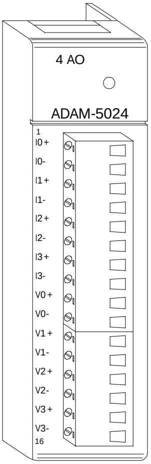


Figure 4-13: *ADAM-5024 module frontal view*

Application wiring

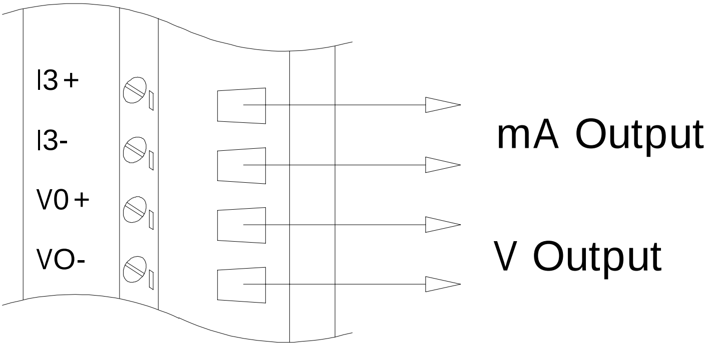


Figure 4-14: *Analog output*

Technical specifications of ADAM-5024

Analog Output Channels	Four
Output Type	V, mA
Output Range	0-20mA, 4-20mA, 0-10V
Isolation Voltage	3000 Vdc
Output Impedance	0.5 Ohms
Accuracy	±0.1% of FSR for current output ±0.2% of FSR for voltage output
Zero Drift	Voltage output: ±30 μV/ºC Current output: ±0.2 μA/ºC
Resolution	±0.015% of FSR
Span Temperature Coefficient	±25 PPM/ºC
Programmable Output Slope	0.125-128.0 mA/sec 0.0625-64.0 V/sec
Current Load Resistor	0-500 Ohms (source)
Power Consumption	2.5W (Max.)

Table 4-7: *Technical specifications of ADAM-5024*

4.4 Analog I/O Modules Calibration

Analog input/output modules are calibrated when you receive them. However, calibration is sometimes required. No screwdriver is necessary because calibration is done in software with calibration parameters stored in the ADAM-5000 analog I/O module's onboard EEPROM.

The ADAM-5000 system comes with the ADAM utility software that supports calibration of analog input and analog output. Besides the calibration that is carried out through software, the modules incorporate automatic Zero Calibration and automatic Span Calibration at bootup or reset.

Analog input module calibration

Modules: ADAM-5017, 5017H, 5018

1. Apply power to the ADAM-5000 system that the analog input module is plugged into and let it warm up for about 30 minutes
2. Assure that the module is correctly installed and is properly configured for the input range you want to calibrate. You can do this by using the ADAM utility software.
3. Use a precision voltage source to apply a span calibration voltage to the module's V0+ and V0- terminals. (See Tables 5-2 and 5-3 for reference voltages for each range.)

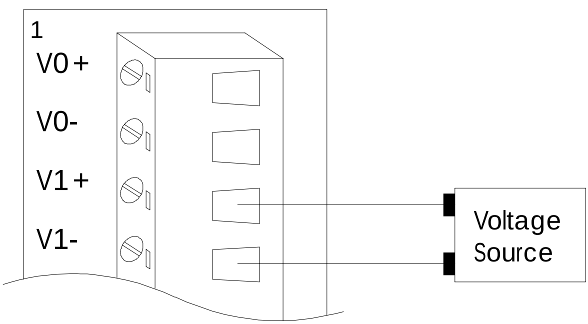


Figure 4-15: *Applying calibration voltage*

4. Execute the Zero Calibration command (also called the Offset Calibration command).

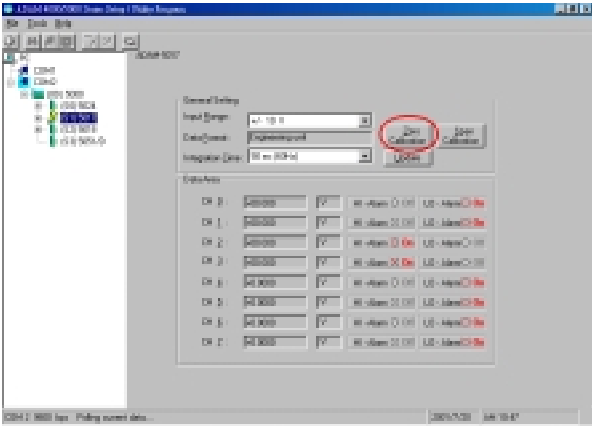


Figure 4-16: *Zero calibration*

5. Execute the Span Calibration command. This can be done with the ADAM utility software.

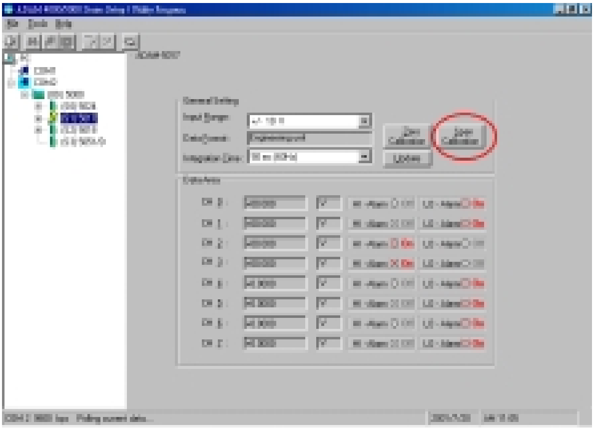


Figure 4-17: *Span calibration*

6. CJC Calibration (only for T/C input module)

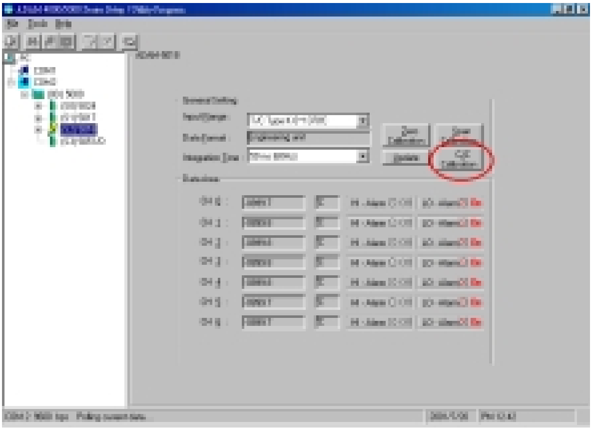


Figure 4-18: *Cold junction calibration*

** Note:* *Zero calibration and span calibration must be completed before CJC calibration. To calibrate CJC, the thermocouple attached to ADAM-5018 and a standard thermometer should be used to measure a standard known temperature, such as the freezing point of pure water. The amount of offset between the ADAM-5018 and the standard thermometer is then used in the ADAM utility to complete CJC calibration.*

Calibration voltage (ADAM-5017/5018)

Module	Input Range Code (Hex)	Input Range	Span Calibration Voltage
5018	00h	±15 mV	+15 mV
	01h	±50 mV	+50 mV
	02h	±100 mV	+100 mV
	03h	±500 mV	+500 mV
	04h	±1 mV	+1 V
	05h	±2.5 V	+2.5 V
	06h	±20 mV	+20 mA (1)
	0Eh	J thermocouple 0 to 1370° C	+50 mV
	0Fh	K thermocouple 0 to 1370° C	+50 mV
	10h	T thermocouple -100 to 400° C	+22 mV
	11h	E thermocouple 0 to 1000° C	+80 mV
	12h	R thermocouple 500 to 1750° C	+22 mV
	13h	S thermocouple 500 to 1800° C	+22 mV
	14h	B thermocouple 500 to 1800° C	+152 mV
5017	07h	Not used	
	08h	±10 V ℃	+10 V
	09h	±5 V	+5 V
	0Ah	±1 V	+1 V
	0Bh	±500 mV	+500 mV
	0Ch	±150 mV	+150 mV
	0Dh	±20 mA	+20 mV (1)

Table 4-8: *Calibration voltage of ADAM-5017/5018*

Calibration voltage (ADAM-5017H)

Module	Input Range Code (Hex)	Input Range	Span Calibration Voltage
5017H	00h	±10 V	+10 V
	01h	0 ~ 10 V	+10 V
	02h	±5 V	+5 V
	03h	0 ~ 5 V	+5 V
	04h	±2.5 V	+2.5 V
	05h	0 ~ 2.5 V	+2.5 V
	06h	±1 V	+1 V
	07h	0 ~ 1 V	+1 V
	08h	±500 mV	+500 mV
	09h	0 ~ 500 mV	+500 mV
	0ah	4 ~ 20 mA	*(1)
	0bh	0 ~ 20 mA	*(1)

Table 4-9: *Calibration voltage of ADAM-5017H*

(1) Note: *You can substitute 2.5 V for 20 mA if you remove the current conversion resistor for that channel. However, the calibration accuracy will be limited to 0.1% due to the resistor's tolerance.*

Analog output module calibration

The output current of analog output modules can be calibrated by using a low calibration value and a high calibration value. The analog output modules can be configured for one of two ranges: 0-20 mA and 4-20 mA. Since the low limit of the 0-20 mA range (0 mA) is internally an absolute reference (no power or immeasurably small power), just two levels are needed for calibration: 4 mA and 20 mA.

1. Apply power to the ADAM-5000 system including the analog output module for about 30 minutes.
2. Assure that the module is correctly installed and that its configuration is according to your specifications and that it matches the output range you want to calibrate. You can do this by using the ADAM utility software.
3. Connect either a 5-digit mA meter or voltmeter with a shunt resistor (250 ohms, .01 % and 10 ppm) to the screw terminals of the module.

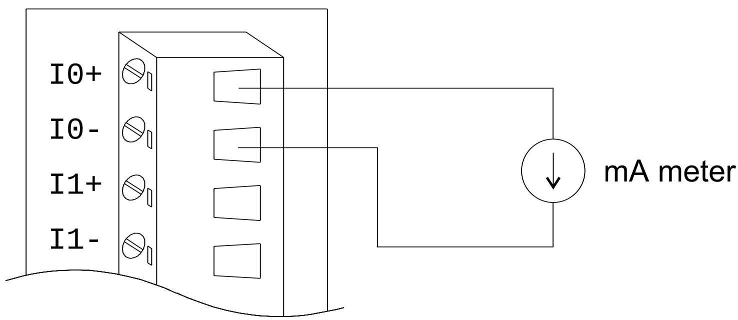


Figure 4-19: *Output module calibration*

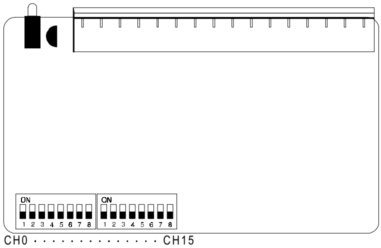
4. Issue the Analog Data Out command to the module with an output value of 4 mA.
5. Check the actual output value at the modules terminals. If this does not equal 4 mA, use the "Trim" option in the "Calibrate"sub-menu to change the actual output. Trim the module until the mA meter indicates exactly **4 mA**, or in case of a voltage meter with shunt resistor, the meter indicates exactly **1 V**. (When calibrating for **20 mA** using a voltage meter and shunt resistor, the correct voltage should be **5 V**.)
6. Issue the 4 mA Calibration command to indicate that the output is calibrated and to store the calibration parameters in the module's EEPROM.
7. Execute an Analog Data Out command with an output value of 20 mA. The module's output will be approximately 20 mA.
8. Execute the Trim Calibration command as often as necessary until the output current is equal to exactly 20 mA.
9. Execute the 20 mA Calibration command to indicate that the present output is exactly 20 mA. The analog output module will store its calibration parameters in the unit's EEPROM.

4.4 Digital Input/Output Modules

ADAM-5050 16-channel universal digital I/O module

The ADAM-5050 features sixteen digital input/output channels. Each channel can be independently configured to be an input or an output channel by the setting of its DIP switch. The digital outputs are open-collector transistor switches that can be controlled from the ADAM-5000. The switches can also be used to control solid-state relays, which in turn can control heaters, pumps and power equipment. The ADAM-5000 can use the module’s digital inputs to determine the state of limit or safety switches, or to receive remote digital signals.

Warning! *A channel may be destroyed if it is subjected to an input signal while it is configured to be an output channel.*



Dip Switch Key
ON = Digital Output
OFF = Digital Input

Figure 4-20: *Dip switch setting for digital I/O channel*

ADAM-5050

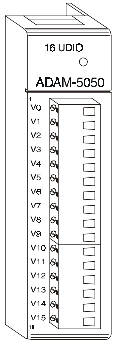


Figure 4-21: *ADAM-5050 module frontal view*

Application wiring

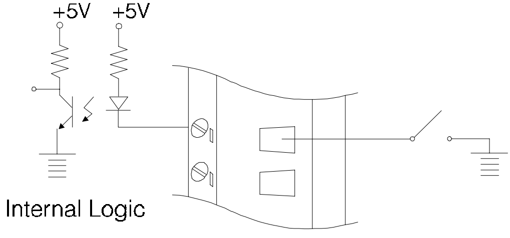


Figure 4-22: *Dry contact signal input (ADAM-5050)*

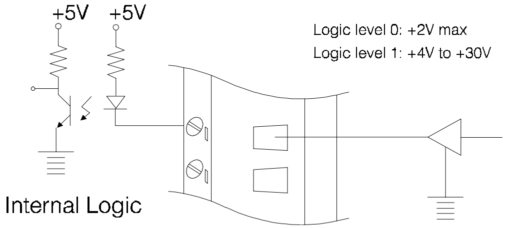


Figure 4-23: *Wet contact signal input (ADAM-5050)*

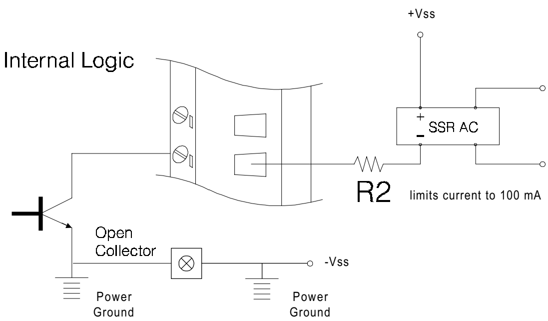


Figure 4-24: *Digital output used with SSR (ADAM-5050/5056)*

Technical specifications of ADAM-5050

Points	16
Channel Setting	Bitwise selectable by DIP switch
Digital Input	Dry Contact Logic Level 0: close to GND Logic Level 1: open Wet Contact Logic Level 0: +2 V max Logic Level 1: +4 V to 30 V
Digital Output	Open collector to 30 V, 100mA max load
Power Dissipation	450 mW
Power Consumption	0.4 W

Table 4-10: *Technical specifications of ADAM-5050*

ADAM-5051(D) 16-channel digital input module

The ADAM-5051 provides sixteen digital input channels. The ADAM-5510 can use the module's digital inputs to determine the state of limit or safety switches or to receive remote digital signals.

ADAM-5051/5051 D

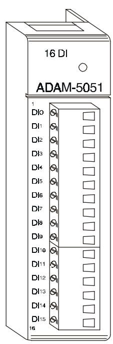


Figure 4-25: *ADAM-5051 module frontal view*

Application wiring

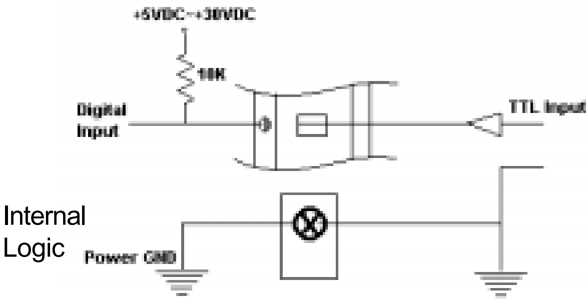


Figure 4-26: *TTL input (ADAM-5051/5051D)*

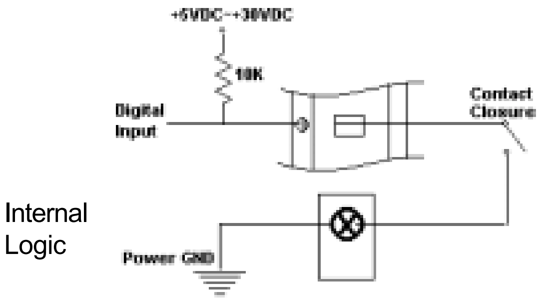


Figure 4-27: *Contact closure input (ADAM-5051/5051D)*

Technical specifications of ADAM-5051/5051D

Points	16
Digital input	Logic level 0: + 1 V max Logic level 1: + 3.5 to 30 V Pull up current: 0.5 mA 10 kΩ resistor to + 5 V
Power consumption	0.3 W
indicator	ADAM-5051 D only

Table 4-11: *Technical specifications of ADAM-5051*

ADAM-5051S 16-channel Isolated Digital Input Module with LED

The ADAM-5051S provides 16 isolated digital input channels for critical environments need individual channel isolating protection. Different from other ADAM-5000 I/O modules, ADAM-5051S designed with 21 pins plug terminal.

ADAM-5051S

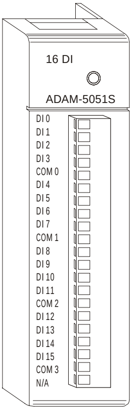


Figure 4-28: *ADAM-5051S module front view*

Application Wiring

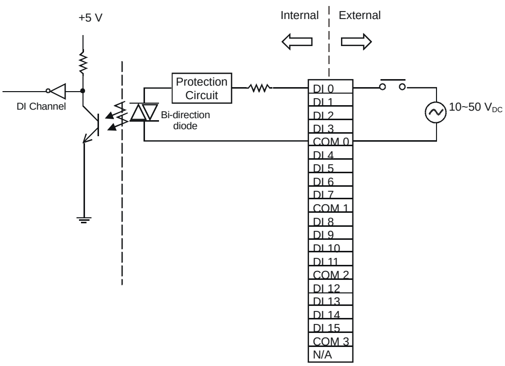


Figure 4-29: *ADAM-5051S module wiring diagram*

Technical specification of ADAM-5051S

Point	16(4-channel/group)
Digital Input	Logic Level 0: + 3 V max Logic Level 1: + 10 to 50 V
Optical Isolation	2500 V_{DC}
Opto-isolator response time	25 μs
Over-voltage Protection	70 V_{DC}
Power Consumption	0.8 W
LED Indicator	On when active
I/O Connector Type	21-pin plug-terminal

Table 4-12: Technical specification of ADAM-5051S

ADAM-5052 8-channel isolated digital input module

The ADAM-5052 provides eight fully independent isolated channels. All have 5000 V_{RMS} isolation to prevent ground loop effects and to prevent damage from power surges on the input lines.

ADAM-5052

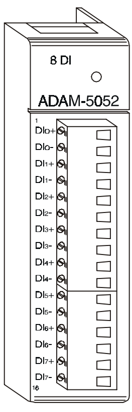


Figure 4-30: *ADAM-5052 module frontal view*

Application wiring

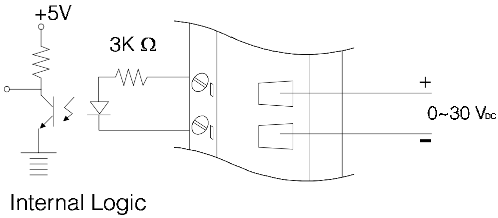


Figure 4-31: *Isolation digital input (ADAM-5052)*

Technical specifications of ADAM-5052

Points	8 Differential
Digital input	Logic level 0: + 1 V max Logic level 1: + 3.5 to 30 V Isolation voltage: 5000 V_{RMS} Resistance: 3 kΩ / 0.5 W
Power consumption	0.4 W

Table 4-13: *Technical specifications of ADAM-5052*

ADAM-5055S 16-channel Isolated Digital I/O Module with LED

The ADAM-5056S provides 8 isolated digital input and 8 isolated output channels for critical environments need individual channel isolating protection. Different from other ADAM-5000 I/O modules, ADAM-5051S designed with 21 pins plug terminal.

ADAM-5055S

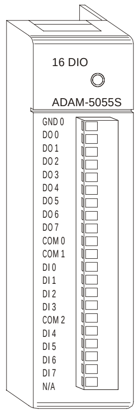


Figure 4-32: ADAM-5055S module front view

Application Wiring

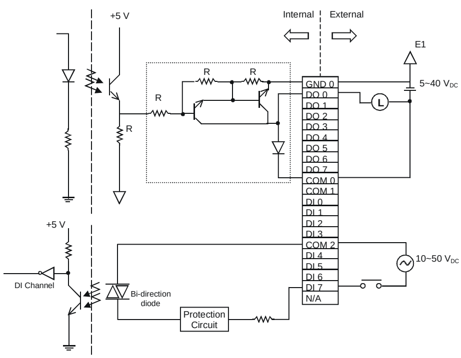


Figure 4-33: ADAM-5055S module wiring diagram

Technical specification of ADAM-5055S

Points	16
Digital Output	8 (8-channel/group)
Open collector to 40 V	200 mA max load per channel
Optical Isolation	2500 V_{DC}
Opto-isolator response time	25 µs
Supply Voltage	5 ~ 40 V_{DC}
Digital Input	8(4-channel/group) **Dry Contact** Logic Level 0: close to GNDLogic Level 1: open **Wet Contact** Logic Level 0: + 3 V maxLogic Level 1: + 10 to 50 V
Dry Contact & Wet contact	Selectable
Optical Isolation	2500 V_{DC}
Opto-isolator response time	25 µs
Over-voltage Protect	70 V_{DC}
Power Consumption	0.68 W
LED Indicator	On when active
I/O Connector Type	21-pin plug-terminal

Table 4-14: Technical specification of ADAM-5055S

ADAM-5056(D) 16-channel digital output module w/LED

The ADAM-5056 features sixteen digital output channels. The digital outputs are open-collector transistor switches that you can control from the ADAM-5000 main unit. You also can use the switches to control solid-state relays.

ADAM-5056

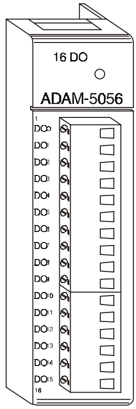


Figure 4-34: *ADAM-5056 module frontal view*

Application wiring

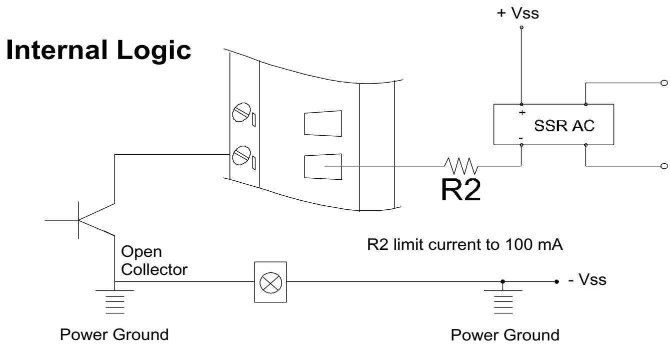


Figure 4-35: *Digital output used with SSR (ADAM-5050/5056)*

Technical specifications of ADAM-5056

There are 16-point digital input and 16-point digital output modules in the ADAM-5000 series. The addition of these solid state digital I/O devices allows these modules to control or monitor the interfaces between high power DC or AC lines and TTL logic signals. A command from the host converts these signals into logic levels suitable for the solid-state I/O devices.

Points	16
Digital output	Open collector to 30 V 100 mA max load
Power dissipation	450 mW
Power consumption	0.25 W

Table 4-15: *Technical specifications of ADAM-5056*

ADAM-5056S 16-channel Isolated Digital Output Module with LED

The ADAM-5056S provides 16 isolated digital output channels for critical environments need individual channel isolating protection. Different from other ADAM-5000 I/O modules, ADAM-5056S designed with 21 pins plug terminal.

ADAM-5056S

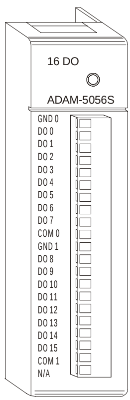


Figure 4-36: *ADAM-5056S module front view*

Application wiring

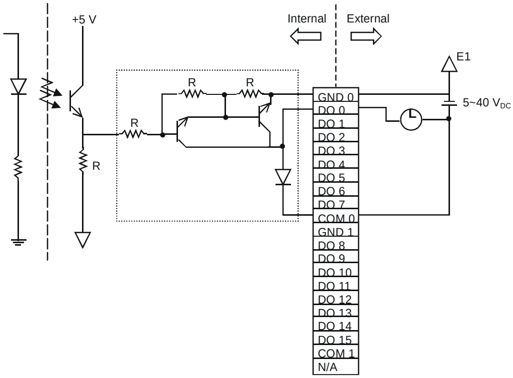


Figure 4-37: *ADAM-5056S module wiring diagram*

Technical Specification of ADAM-5056S

Points	16(8-channel/group)
Digital Output	Open collector to 40 V 200 mA max load per channel
Optical Isolation	2500 V_{DC}
Opto-isolator response time	25 µs
Supply Voltage	5 ~ 40 V_{DC}
Power consumption	0.6 W
LED Indicator	On when active
I/O Connector Type	21-pin plug-terminal

Table 4-16: *Technical specification of ADAM-5055S*

4.5 Relay Output Modules

ADAM-5060 relay output module

The ADAM-5060 relay output module is a low-cost alternative to SSR modules. It provides 6 relay channels, two of Form A and four of Form C.

ADAM-5060

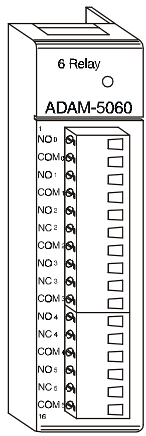


Figure 4-38: *ADAM-5060 module frontal view*

Application wiring

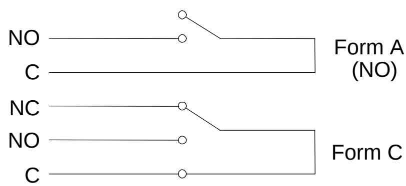


Figure 4-39: *Relay output*

Technical specifications of ADAM-5060

Points	6, two Form A and four Form C
Contact rating	AC: 125 V @ 0.6A; 250 V @ 0.3 A DC: 30 V @ 2 A; 110 V @ 0.6 A
Breakdown voltage	500 V_{AC} (50/60 Hz)
Relay on time (typical)	3 ms
Relay off time (typical)	1 ms
Total switching time	10 ms
Insulation resistance	1000 MΩ min. @ 500 V_{DC}
Power consumption	0.7 W

Table 4-17: *Technical specifications of ADAM-5060*

ADAM-5068 relay output module

The ADAM-5068 relay output module provides 8 relay channels of Form A. Switches can be used to control the solid-state relays.

ADAM-5068

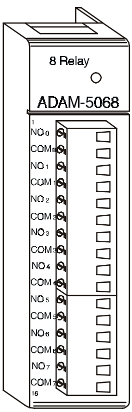


Figure 4-40: *ADAM-5068 module frontal view*

Application wiring

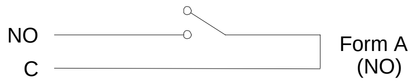


Figure 4-41: *Relay output*

Technical specifications of ADAM-5068

Points	8 Form A
Contact Rating	AC: 120 V @ 0.5 A DC: 30 V @ 1 A
Breakdown Voltage	500 V_{AC} (50/60 Hz)
Relay On Time (typical)	7 msec.
Relay Off Time (typical)	3 msec.
Total Switching Time	10 msec.
Power Consumption	2.0 W

Table 4-18: *Technical specifications of ADAM-5068*

4.6 Counter/Frequency Module

Overview

Compatible ADAM-5000 Series Main Units

ADAM-5080 is a 4-channel counter/frequency module designed to be implemented within the following Advantech ADAM-5000 series main units:

ADAM-5000/485
ADAM-5510
ADAM-5511

Please make sure that the ADAM-5080 counter/frequency module is properly inserted into the compatible main units.

ADAM-5080 4-channel Counter/Frequency Module

With ADAM-5080 4-Channel Counter/Frequency Module, users can select either counter or frequency mode for data output. ADAM-5080 offers users a variety of very flexible and versatile applications such as below:

Counter Mode or Frenquency Mode

If you want to measure the number of input signals for totalizer function, you may use counter mode to measure quantities such as movement and flow quantity. Alternatively, you can also select frequency mode to calculate the instantaneous differential of quantities such as rotating speed, frequency or flow rate, and present them in specific engineering formats.

Up/Down or Bi-direction Function

When operating in counter mode, you can choose either the Up/Down function or the Bi-direction function for different application purposes. The counter will count up or down according to your applications. This counting function helps users obtain the most accurate data.

Alarm Setting Function

While in counter mode, you can set alarm status--Disable and Latch. If you want to disable it, you can select Disable.If Latch status is selected, it means the Alarm status will be "latched" whenever the alarm being triggered. Once the alarm status being "latched," it will thereafter stay in that triggered state. Users will have to issue a "Clear Alarm Status" command to return the "latched" alarm status back to normal. Users can designate the high-limit value and low-limit value to regulate your alarm behavior through the utility program.

Digital Output Mapping

Users can either run the utility program or issue a "Set Alarm Connetion" command to designate a specific digital output module for the alarm signal to be sent through.

ADAM-5080 Module Diagram

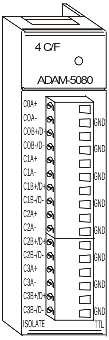


Figure 4-42: *ADAM-5080 Module*

ADAM-5080 Application Wiring

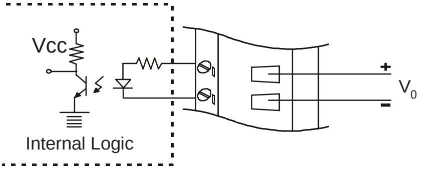


Figure 4-43: *Isolated Input Level*

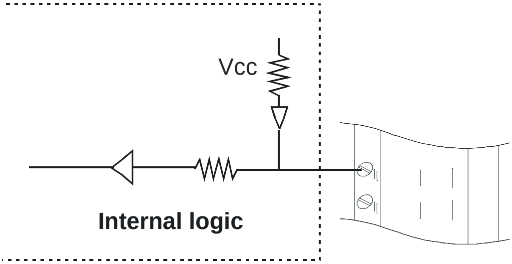


Figure 4-44: *TTL Input Level*

ADAM-5080 Counter/Frequency Mode Selection

Users can select Bi-direction, Up/Down Counter or Frequency option as shown in Figure 4-44.

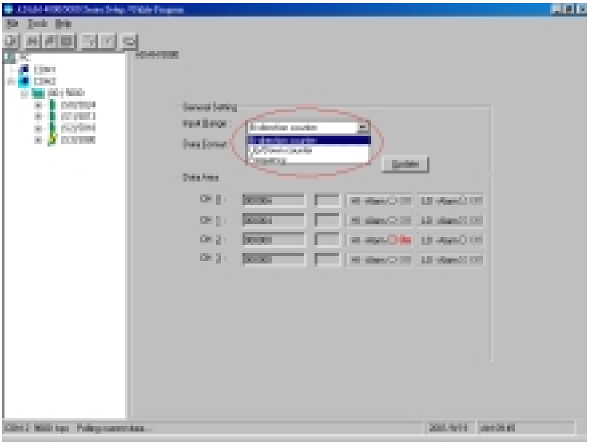


Figure 4-45: *Counter / Frequency Mode*

Note: *All four channels of ADAM-5080 will operate simultaneously in the mode you have selected. i.e. If you switch the ADAM-5080 to Counter Mode, all four channels will operate in Counter Mode.*

Features -- Counter Mode

Up/Down Counting

The Up/Down Counter Function offers two types of counting:
Up Couting (increasingly) and Down Counting (decreasingly).

Up Counting : when C0A+ and C0A- sense any input signals, the counter counts up.

Down Counting : when C0B+ and C0B- sense any input signals, the counter counts down.

On receiving Up and Down signal simultaneously, the counter will not perform each specific counting accordingly, but will remain at the previous counting value, since these simultaneous signals won't have any effect on counting values.

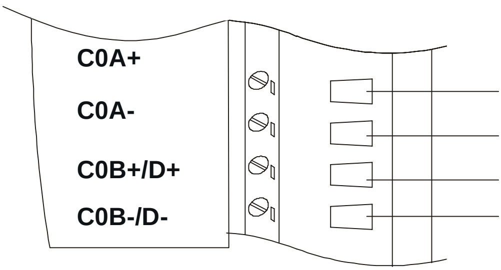


Figure 4-46: *Wiring for Up/Down Counting*

***Note**:* *If you need only one type of counting, connect C0A+ and C0A- for Up Counting only; or connect C0B+ and C0B- for Down Counting only.*

Bi-direction Counting

For implementing Bi-derection Counting, you need to connect C0B+/D+ and C0B-/D- to implement the control function for Up/Down Counting.

Up Counting : when the input signal is within logic level "1", the counter value increases.

Down Counting : when the input signal is within logic level "0", the counter value decreases.

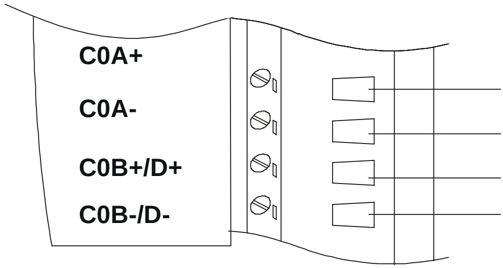


Figure 4-47: *Wiring for Bi-direction Counting*

Note: *If users select TTL mode and don't connect C0B+ C0B-, the counter value will increase. If users select Isolated mode and don't connect C0B+ C0B-, the counter value will decrease.*

Features -- Frequency Mode

If users want to select frequency mode, they can only utilize Up Counting type, and can only connect to C0A+ and C0A-.

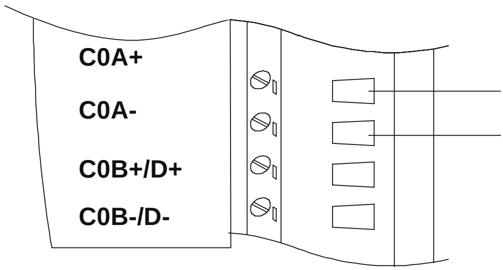


Figure 4-48: *Wiring for Frequency Mode*

Features -- Alarm Setting

According to your application purposes, you can run the utility program to set different limit values for High/Low Alarm.

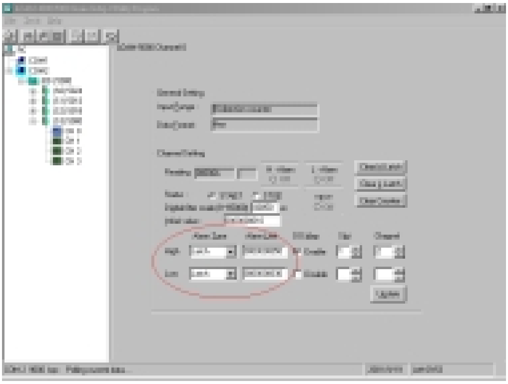


Figure 4-49: *Setting Alarm Limit*

Setting Initial Counter Value

In oder to utilize the alarm function, users have to set a high-alarm limit value and/or a low alarm limit value, and a initial value to fulfill the requirements for a basic alarm setting.

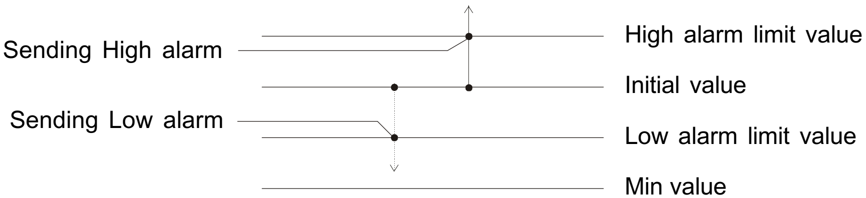


Figure 4-50: *Sending Alarm Signal (recommended settings)*

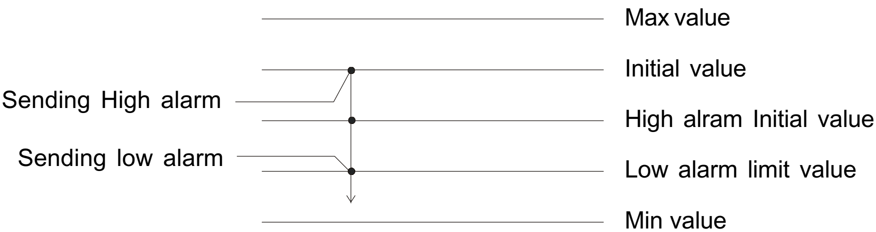


Figure 4-51: *Sending Alarm Signal (settings not recommended)*

Overflow Value

Overflow value is the number of times the counter value exceeds the Max/Min values you specified. When the counter value exceeds Maximum value, the overflow value increases; When the counter value goes under Minimum value, the overflow value decreases. Besides, when the counter value runs beyond the range of Max/Min value, it will continue counting from the initial value. Furthermore, if users want to check the counter value to see if it is higher or lower than the Max/Min value, they can use the "ReadOverflowFlag" library to gain a readout of the overflow value.

Getting the Totalizer Value

If users want to get the actual counter value, a formula such as follows can facilitate an easy calculation from the initial counter value, overflow value and current counter value:

$$V_{tol} = \{ |V_{ini} - V_{min}\,(\text{or } V_{max})| + 1 \} \text{ x } |V_{vf}| + |V_{ini} - V_{cur}|$$

V_{tol} : totalizer value

V_{ini} : initial counter value

V_{min} : min. couner value = 0 (fixed value)

V_{max} : max. counter value = 2^{32} = 4,294,967,295 (fixed value)

V_{vf} : overflow value

V_{cur} : current counter value

Example:

If the initial value =10, overflow value =4, min. value = 0, current counter value = 3, the totalizer value could be calculated as

totalizer value = {|10 - 0| + 1} x| 4 |+ |10 -3| = 51

Features--Digital Output Mapping

If users want to use Digital Output function, ADAM utility is available for setting specifically which module, channel or slot to receive the alarm signals.

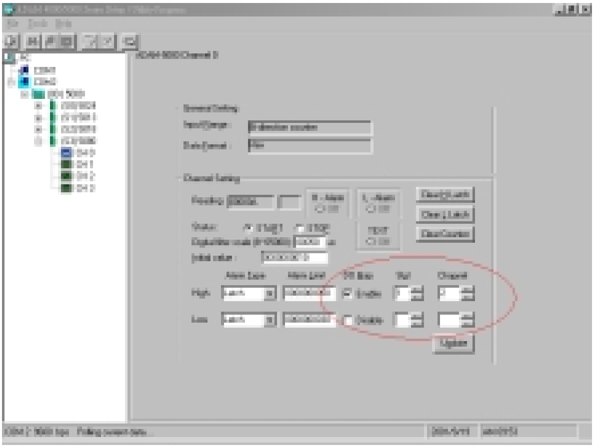


Figure 4-52: *Digital Output Mapping*

1: **High Alarm State--**Set Alarm state to "Latch" or "Disable".

2: **High Alarm Limt--**Set Alarm limit from 0 to 4,294,967,295.

3: **High Alarm Output Mode--**Enable or Disable D.O. Mapping.

4: **High Alarm Output Slot--**Users can select D.O Modules such as ADAM-5050, ADAM-5055, ADAM-5056, ADAM-5060, ADAM-5068 for the alarm signal to be sent through.

5: **High Alarm Output Channel--**Select Alarm Output Channel

6: **Clear Latch Alarm--**Users can Select "Enable" or "Disable" option. When selecting "Enable", the latch will be relieved and the alarm state will return to normal. Once the alarm state returns to normal, the **Clear Latch Alarm** will return to "Disable".

TTL/Isolated Input Level

According to your need, you can select either TTL or Isolated Input Level by setting the configuration for the jumpers. Select the proper jumper settings for either TTL or Isolated Input according to Figure 4-53. Please note that you must configure all six jumpers to the correct configuration for proper function.

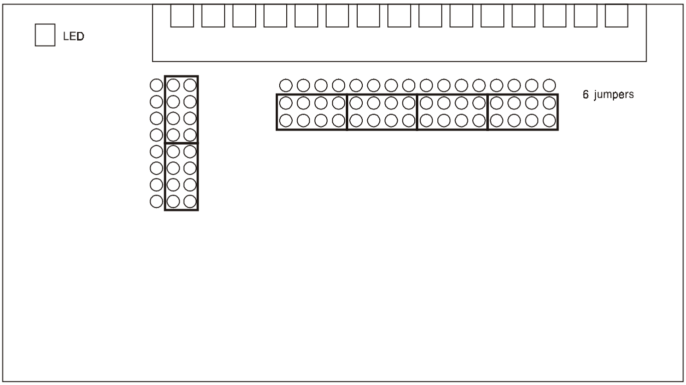
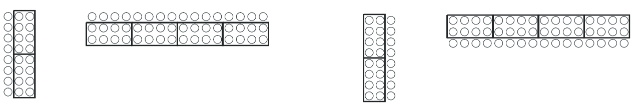


Figure 4-53: *Jumper Location on the ADAM-5080 Module*



TTL Input Level   Isolated Input Level

Figure 4-54: *TTL/Isolated Input Level Selectting*

ADAM-5080 Technical Specifications

Channel	4
Input Frequency	0.3 ~ 1000 Hz max. (Frequency mode) 5000 Hz max. (Counter mode)
Input Level	Isolated or TTL level
Minimum Pulse Width	500 μ sec. (Frequency mode) 100 μ sec. (Counter mode)
Minimum Input Current	2mA (Isolated)
Isolated Input Level	Logic Level 0 : +1 V_{MAX} Logic Level 1 : + 3.5 V to 30 V
TTL Input Level	Logic Level 0 : 0 V to 0.8 V Logic Level 1 : 2.3 to 5 V
Isolated Voltage	1000 V_{RMS}
Mode	Counter (Up/Down, Bi-direction) Frequency
Programmable Digital Noise Filter	8 ~ 65000 μ sec

Table 4-19: *ADAM-5080 technical specifications*

4.7 Serial Module

Overview

Compatible ADAM-5000 Series Main Units

The ADAM-5090 is a 4-port RS-232 communication module to be implemented with the following Advantech ADAM-5000 series main units:

ADAM-5510 (with library Version V1.10 or above)
ADAM-5511 (with library Version V1.10 or above)

ADAM-5090 4-port RS-232 Communication Module

Bi-direction Communication

The ADAM-5090 is equipped with four RS-232 ports, which makes it especially suitable for bi-direction communication. It can simultaneously read data from other third-party devices such as Bar Code and PLC as long as these devices are equipped with a RS-232 interface. Furthermore, the ADAM-5090 can issue commands to control other devices. It is fully integrated with the ADAM-5000, ADAM-5500 and ADAM-4000 series, and transmits data to each other through the RS-232 port. The whole integrated system is an intelligent stand-alone system and can connect and issue commands to control devices such as printers and PLCs in remote factory location.

The ADAM-5090 transmits and receives data by polling communication, and each port can receive up to 128 bytes in the FIFO. For continuous data longer than 128 bytes, please refer to Table 4.20 for Baud Rate setting to avoid data loss.

Baud Rate (bps)	115200	57600	38400	19200	9600	4800	2400
Polling interval (ms)	11.11	22.22	33.33	66.66	133.33	266.66	533.33

Table 4-20: *Baud Rate setting reference table*

Communication Backup Function

With the ADAM-5090 you can implement dual communication channels between your PC and the ADAM system. Even when one of the two communication channels is down, your system can still function through the alternative communication channel. This dual communication channels can be implemented by application software.

ADAM-5090 Module Diagram

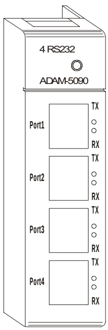


Figure 4-55: *ADAM-5090 Module*

ADAM-5090 Application Wiring

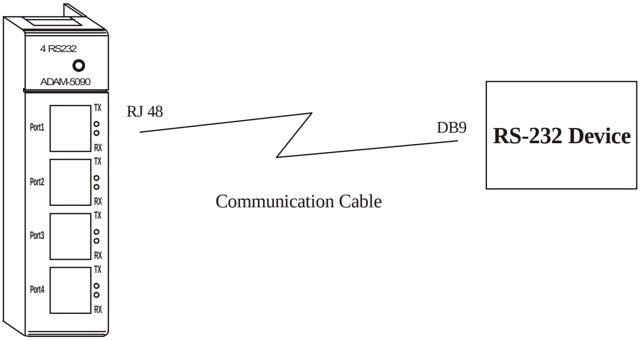


Figure 4-56: *ADAM-5090 Application Wiring*

PIN Mapping

PIN Name	RJ-48	DB9
/DCD	1	1
RX	2	2
TX	3	3
/DTR	4	4
GND	5	5
/DSR	6	6
/RTS	7	7
/CTS	8	8
RI or +5V	9	9
GND	10	X

Table 4-21: *Pin Mapping*

ADAM-5090 Technical Specification

Function	Provides communication ports for the ADAM-5510 to integrate other devices with communication function into your system
Electrical Interface	4 ports (RS-232)
Communication Rates	4800, 9600, 19200, 38400, 115200bps
FIFO	128 bytes/per UART (Tx/Rx)
Indicator	Tx (Orange), Rx (Green)
Power Required	100mA @ $5V_{DC}$ Default in RI mode (*)

Table 4-22: *ADAM-5090 technical specifications*

* User can define the communication ports with 5VDC output by switching the jumper, and the maximum current output is 400mA.

I/O Slots and I/O Ports Numbering

The ADAM-5090 module provides four RS-232 ports for communication with target devices. The ports are numbered 1 through 4. For programming, the definition of port number depends on the slot number and port number. For example, the second port on the ADAM-5090 in slot 1 is defined to port 12 (refer to table 6.1).

Jumper Settings

This section tells you how to set the jumpers to configure your ADAM-5090 module. There are four jumpers on the PC Board. User can choose RI signal or 5V output for each port by setting these jumpers (system default is RI signal).

The following figure shows the location of the jumpers:

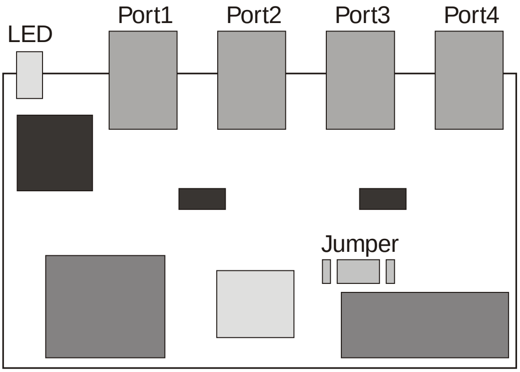


Figure 4-57: *Jumper locations on the CPU card*

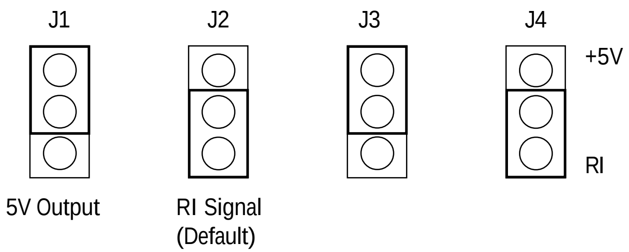


Figure 4-58: *Jumper Settings*

LED Status of the ADAM-5090 Module

There are two LEDs for each port on the front panel of the ADAM-5090 to display specific communication status:

a. Green LED (RX): Data Receiving Status; the LED indicator is on when the port is receiving data.

b. Orange LED (TX): Data Transmitting Status; the LED indicator is on when the port is transmitting data.

Configure Your ADAM-5090 Module

This section explains how to configure an ADAM-5090 module before implementing it into your application.

Quick Start

Step 1: Get your host PC ready, and run the ADAM-5510 Utility Software.

Step 2: Install the ADAM-5090 Module and power on your ADAM-5510 main unit.

Step 3: Download the executable program to the main unit

Step 4: Monitor the ADAM-5090 Module's current status from the PC through the utility software.

A basic example program for the ADAM-5090

```
main ()
{
  //Install the port you would like to use. Here we install slot 0, port 1.
  port_install(1);
  // Here we install slot 2, port 2.
  port_install(22);

  //Select working port. Here we select slot 0, port 1.
  port_select(1);
  //Set port data format.
  //Here we set the data format of port 1 as lengh:8; parity:0;stop_bit:1. (N81)
```

```
    port_set_format(1,8,0,1);

    //Set port speed. Here we set communication speed of port 1 as 115200 bps.
    //(L is necessary)
    port_set_speed(1,115200L);

    //Enable Port FIFO. Here we enable 128 byte FIFO for port1.
    port_enable_fifo(1);

    //After these above settings are enabled, you can apply any other
    function library to implement your program.
}
```

—A receive-and-transmit example program for the ADAM-5090

```
main()
{
    int err_value, char character
    port_installed(1)
    :
    :
    port_enable_fifo(1);

    //check whether error has been received or not
    err_value=port_rx_error(1);

    //if error detected, print out the message
    if(err_value)
    {
     printf("\n Rx Error, The LSR Value=%02X", Err_value)";
    }
    //check whether FIFO receives data or not; if data received, read a character
    if(port_rx_ready(1))
```

```
    {
    character=port_rx(1);
    }

    //check whether FIFO is empty or not, if empty, send a character
    if(port_tx_empty(1));
    {
    port_tx(1, character)
    }
}
```

This page inyrntionally left blank

Chapter 5

Programming and Downloading

Programming and Downloading

This chapter explains how to program applications and download programs into the ADAM-5511 system. Additionally, it points out limitations and issues about which you should be aware.

5.1 Programming

The operating system of ADAM-5511 is ROM-DOS, an MS-DOS equivalent system. It allows users to run application programs written in assembly language as well as high level languages such as C or C++. However, there are limitations when running application programs in the ADAM-5511. In order to build successful applications, you should keep the following limitations and concerns in mind.

5.1.1 Mini BIOS functions

The ADAM-5511 provides only two serial communication ports for connecting peripherals, so the mini BIOS of ADAM-5511 only provides 10 function calls. Since the user's program cannot use other BIOS function calls, the ADAM-5511 may not work as intended.

Additionally, certain language compilers such as QBASIC directly call BIOS functions that are not executable in ADAM-5511. The ADAM-5511 mini BIOS function calls are listed in the following table.

Function	Sub-function	Task
07h		186 or greater co-processor esc instruct
10h	0eh	TTY Clear output
11h		Get equipment
12h		Get memory size
15h	87h	Extended memory read
	88h	Extended memory size
	c0h	PS/2 or AT style A20 Gate table
16h	0	Read TTY char
	1	Get TTY status
	2	Get TTY flags
18h		Print “Failed to BOOT ROM-DOS” message
19h		Reboot system
1ah	0	Get tick count
	1	Set tick count
	2	Get real time clock
	3	Set real time clock
	4	Get date
	5	Set date
1ch		Timer tick

Table 5-1: ADAM-5511 mini BIOS function calls

5.1.2 Converting program codes

The ADAM-5511 has an 80188 CPU. Therefore, programs downloaded into its flash ROM must first be converted into 80186 or 80188 compatible code, and the floating point operation must be set to emulation mode. For example, if you were to develop your application program in Borland C, you would compile the program as indicated in the screen below.

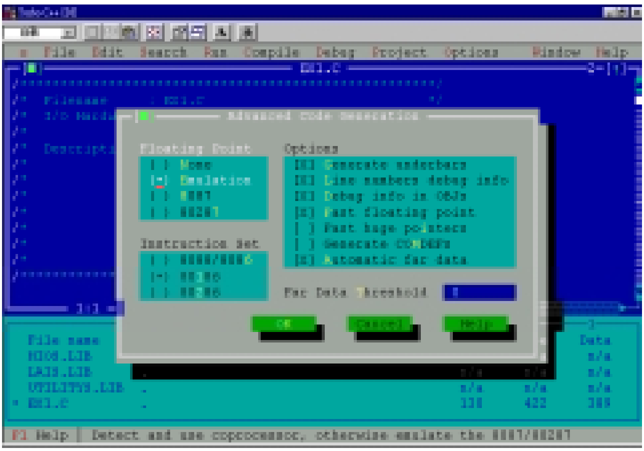


Figure 5-1: Converting program codes

5.1.3 Other limitations

1. The ADAM-5511 does not support the standard PC function "8253". Therefore, the C language, function call "delay ( )" cannot be used in ADAM-5511 applications.

2. Certain critical files are always kept in flash ROM, such as operating system, BIOS, and monitoring files. The ADAM-5511 provides an additional 400KB free space of flash disk for downloading and operation user applications.

5.1.4 Programming the watchdog timer

The ADAM-5511 is equipped with a watchdog timer function that resets the CPU or generates an interrupt if processing comes to a standstill for any reason. This feature increases system reliability in industrial standalone and unmanned environments.

If you decide to use the watchdog timer, you must write a function call to enable it. When the watchdog timer is enabled, it must be cleared by the application program at intervals of less than 1.6 seconds. If it is not cleared at the required time intervals, it will activate and reset the CPU, or generate a NMI (Non-Maskable Interrupt). You can use a function call in your application program to clear the watchdog timer. At the end of your program, you still need a function call to disable the watchdog timer.

5.2 System Configuration

This section explains how to configure I/O module before applied into ADAM-5511 system.

5.2.1 System Requirements

Host Computer Requirements

1. IBM PC compatible computer with 486 CPU (Pentium is recommended).
2. Microsoft 95/98/NT 4.0 (SP3 or SP4) or higher versions.
3. Borland Turbo C for DOS (V2.0 or above)
4. At least 32 MB RAM.
5. 20 MB of hard disk space available.
6. VGA color monitor.
7. 2x or higher speed CD-ROM.

8. Mouse or other pointing devices.

9. At least one standard RS-232 port (e.g. COM1, COM2).

10. One RS-485 card or RS-232 to RS-485 converter (e. g. ADAM-4520) for system communication.

Note: *Users can visit the website of Borland Co. to apply as a member of Borland Community.*

community.borland.com

Then you can free download the Turbo C version 2.01 from the website.

community.borland.com/museum

ADAM-5511 Requirements

1. One ADAM-5511 main unit with two blank slot covers.

2. One ADAM-5511 Quick Start Book

3. One core clamp for power supply connection.

4. One ADAM-4000/5000 Products Utilities CD.

5. Power supply for ADAM-5511 (+10 to +30 VDC unregulated)

6. One RS-232 crossover DB-9 cable

7. One RS-232 straight through DB-9 cable

5.2.2 Analog Module Configuration Guide

Before setup ADAM-5511 system, users need to configure the Analog module first, and ADAM-4000/5000 Utility Software provides a graphical user interface for configuring and calibrating Advantech ADAM-4000/5000 Modules.

Hardware Setting for ADAM-5000 Analog Module Configuration

1. Open the ADAM-5511 package and make sure that everything described in Chapter 5.2.1 is ready.

2. Set all the DIP switches off on ADAM-5511 back plant. See Figure 5-2.

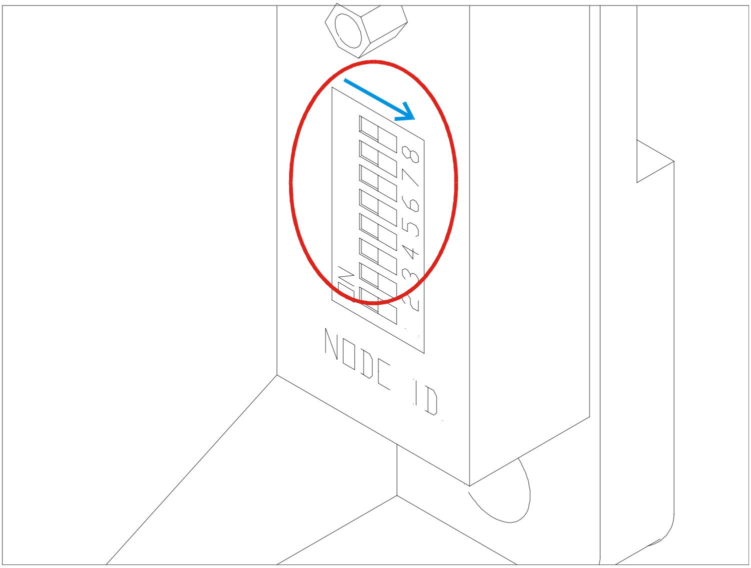


Figure 5-2: ADAM-5511 network address & baud rate DIP switch

3. Connect the ADAM-5511 power cable between the power supply and the ADAM-5511 screw terminals (+Vs and GND). Please make sure that the power source is between +10 to +30 VDC.

4. Connect the straight through cable between the host computer and the ADAM-5511 COM3 for I/O modules configuration. Refer to Figure 5-3. Then press the reset button and wait the orange indicator on.

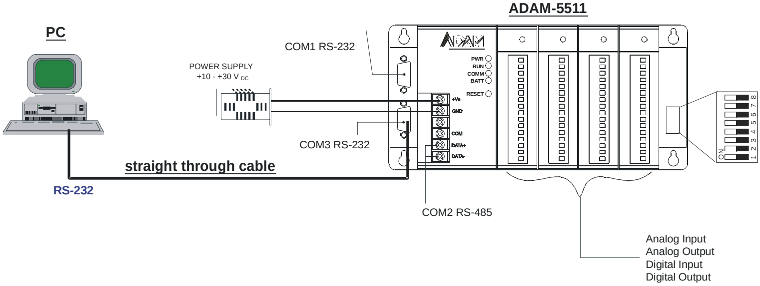


Figure 5-3: Cable connection for I/O Module Configuration

Hardware setting for ADAM-4000 module configuration

Prepare an environment as Figure 5-3 for ADAM-4000 Module configuration.

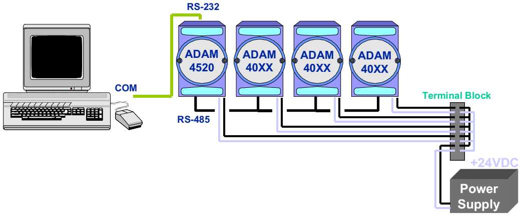


Figure 5-3: Configure ADAM-4000 Module

Install utility software on host PC

ADAM-5511 systems come packaged with a Utility CD containing ADAM-4000/5000 Windows Utility as ADAM-4000 and ADAM-5000 I/O module configuration tool. Installing procedure are as follows:

1. Insert ADAM-4000/5000 Products Utilities CD into the CD drive (e.g. D:) of the host PC. When the installation menu appears, select "Install ADAM-4K5K Utility".

2. When the installation is completed, there will be a shortcut icon of ADAM Utility on the Windows' Desktop.

3. Double click the ADAM Utility icon on your Desktop, and the Utility screen will pop up as below.

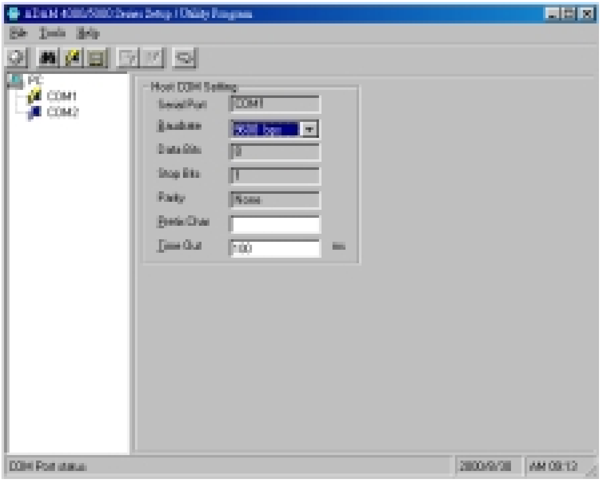


Figure 5-5: ADAM-4000/5000 Windows Utility

Attention: Please be aware of setting all the DIP switches to off position before you use ADAM-4000/5000 Utility software.

5.2.3 Analog Module Configuring by ADAM-4000/ 5000 Utility:

Sets the input/output range, data format, integration time, and high/ low alarm for each module in your ADAM-5511 system.

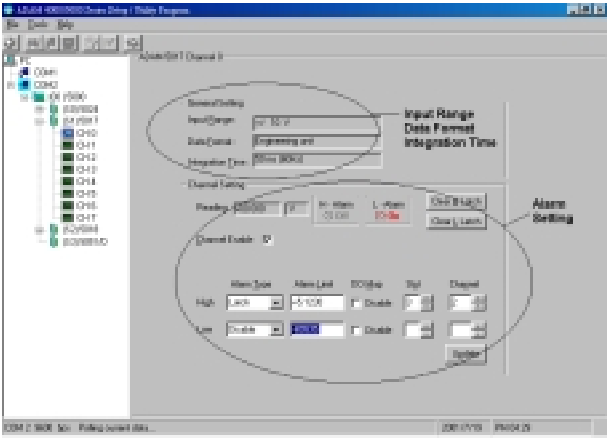


Figure 5-6: I/O Module Configuration

Input Range: represents the type of input signal. Please refer to Chapter 4 for further detail of individual ADAM-5000 modules.

Data Format: represents the data format. Please refer to Chapter 4 for further detail of individual ADAM-5000 modules.

Integration Time: the data scan rate of specific channels or modules.

Alarm Setting: The alarm setting is only effective against to ADAM-5000/485 and ADAM-5000E. Users can skip this portion when apply to ADAM-5511.

5.2.4 Analog Module Calibrating by ADAM-4000/ 5000 Utility:

Analog input/output modules are calibrated when you receive them. However, calibration is sometimes required. If you do not need to calibrate the modules right now, please skip to Chapter 5.3 for using ADAM-5511 Windows Utility.

No screwdriver is necessary because calibration is done in software with calibration parameters stored in the ADAM-5000 analog I/O module's onboard EEPROM.

Note: The calibrating function supports ADAM-5013/5017/5017H/5018/5024.

Zero Calibration:

(1). Apply power to the module and let it warm up for about 30 minutes.

(2). Make sure that the module is correctly installed and is properly configured for the input range you want to calibrate.

(3). Use a precision voltage source to apply a calibration voltage to the modules' terminals of the specific channel.

(4). Click the "Zero Calibration" button. See Figure 5-7

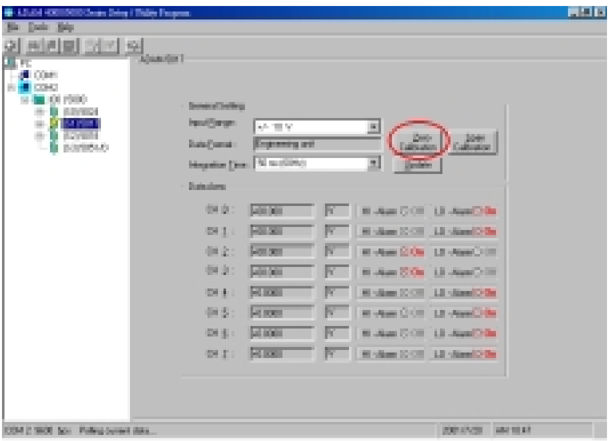


Figure 5-7: Zero Calibration

(5). Click the Execute button to begin the calibration

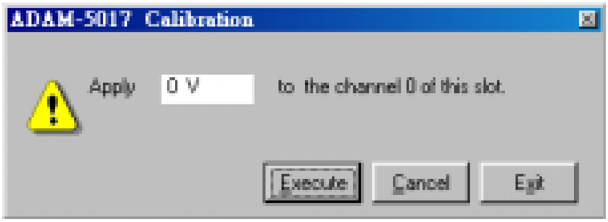


Figure 5-8: Execute Zero Calibration

Span Calibration:

(1). Use a precision voltage source to apply a calibration voltage to the modules' terminals of the specific channel.

(2). Click the "Span Calibration" button. See Figure 5-9

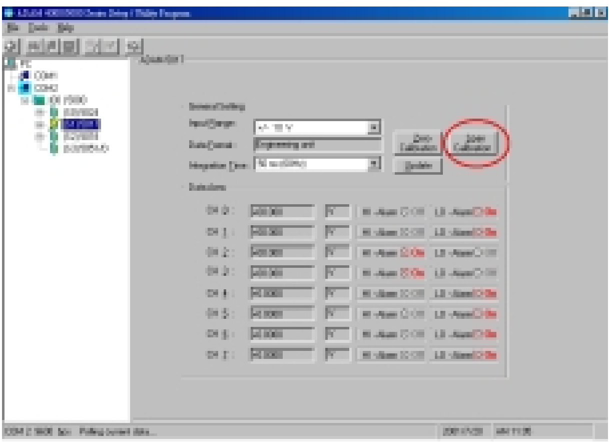


Figure 5-9: Span Calibration

(3). Click the Execute button to begin the calibration

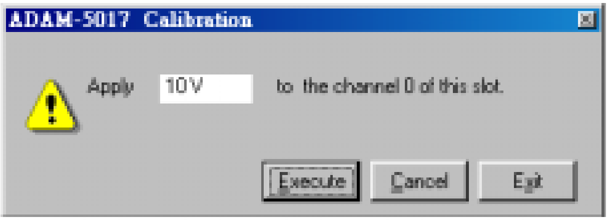


Figure 5-10: Execute Span Calibration

CJC Calibration:

CJC (cold junction sensor) calibration only applies to the ADAM-5018

(1). Prepare a voltage source which is accurate to the mV level.

(2). Run the zero calibration and span calibration function.

(3). Use a temperature emulation device (such as Micro-10) to send a temperature signal to the ADAM module and then compare this signal with the reading from the ADAM module. If the reading value is different from the signal, adjust the CJC value to improve it.

(4). Click the "CJC Calibration" button. See Figure 5-11.

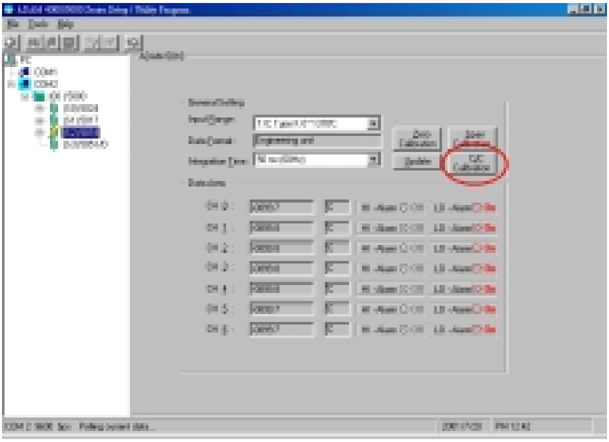


Figure 5-11: CJC Calibration

(5). Click the Execute button to begin the calibration

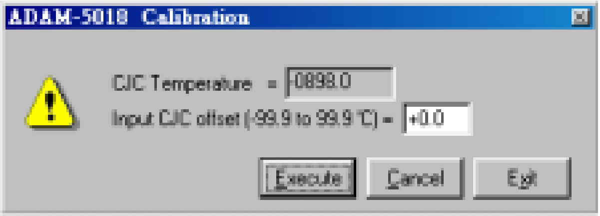


Figure 5-12: Execute CJC Calibration

Analog Input Resistance Calibration:

RTD sensor calibration only applies to the ADAM-5013

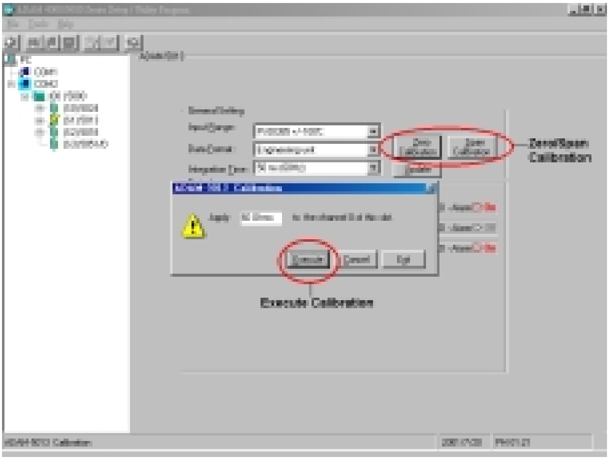


Figure 5-13: RTD Module Calibration

Analog Output Calibration:

4~20 mA: ADAM 5024

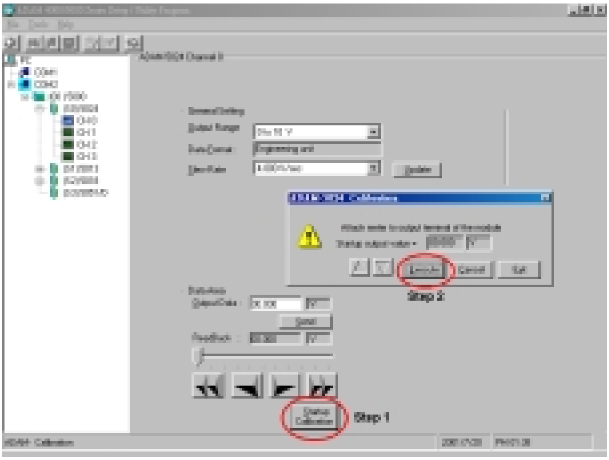


Figure 5-14: Analog Output Calibration

5.2.5 System Installation Guide

After completing the I/O modules configuration, users can setup ADAM-5511 as a dummy I/O system or a programmable stand-alone controller. This section will guide you to complete your system configuration.

Hardware setup

Connect the network between the host computer and the ADAM-5511.

Host Computer Setting (Windows Utility Setting, refer to Chapter 5.3.2):

1) Port Number: assign a specific COM port in your computer to communicate with the ADAM-5511

2) Baud Rate: set a communication rate between host PC and ADAM-5511

3) Time Out: set a time out period for network diagnostic

Network Cable Connection:

There are two kind of connecting methods for system linkage. One is connected via RS-232 DB9 cross over cable to COM1 of ADAM-5511 (see Figure 5-15), and the other one is connected via ADAM-4520 to COM2 of ADAM-5511 (see Figure 5-16).

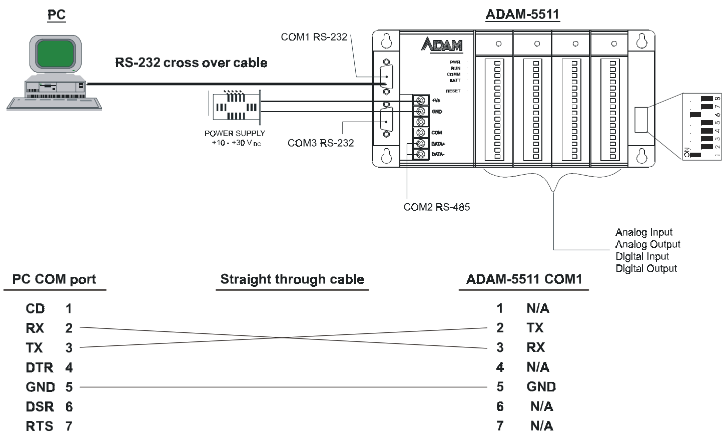


Figure 5-15: COM1 RS-232 Connection

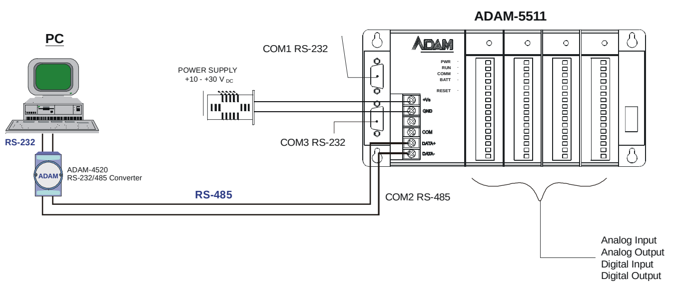


Figure 5-16: COM2 RS-485 Connection

ADAM-5511 Setting:

1) ID Address: Node ID of Modbus network
2) COM1 Enable/Disable: set enable when you use COM1
3) Baud Rate: set a communication rate the same as the host PC's setting

Refer these tables below to adjust the DIP switches in ADAM-5511.

Node ID:

Dip	1	2	3	4	5
ON	2^0	2^1	2^2	2^3	2^4
OFF	0	0	0	0	0

Port Select:

Dip	6
ON	Com1 Enable
OFF	Com1 Disable

Baud Rate:

Dip 7	Dip 8	Baud Rate (bps)
OFF	OFF	9600
ON	OFF	19200
OFF	ON	38400
ON	ON	115200

5.3 Using ADAM-5511 Windows Utility

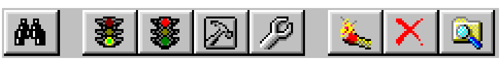
The ADAM-5511 Windows Utility offers a graphical interface that helps you configure the ADAM-5511 stand-alone controller. This Windows Utility makes it very convenient to monitor your Data Acquisition and Control system. The following guidelines will give you some brief instructions on how to use the utility.

- Overview
- COM port settings
- Search connected modules
- Module configuration & Data monitor
- Download procedure
- Remote I/O
- Integrated with HMI

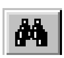
5.3.1 Overview

Main Menu

The window utility consists of a toolbar on the top and a display area that shows forth the relevant information about the connected modules. The utility's main toolbar is as shown below:

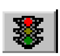


The main toolbar buttons are shortcuts to some commonly used menu items:

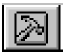


Search: Search for com port or the connected module on network.

Run: Execute the selected program remotely.



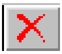
Stop: Stop the running program remotely.



Terminate: Restart ADAM-5511 and do not execute any program.

Reset: Restart ADAM-5511 and re-execute the original program.

Download: Transfer the selected program from PC to ADAM-5511.



Delete: Delete the selected program from ADAM-5511.

Refresh: Refresh the directory of ADAM-5511.

5.3.2 Com port settings

ADAM-5511 Utility will auto-detect the com port in your PC when you start it up.

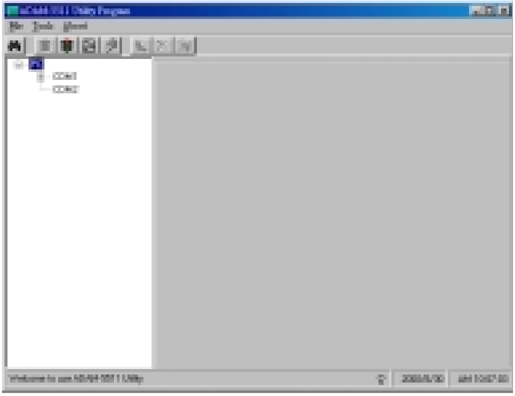


Figure 5-17: Auto-detect COM port

Programming and Downloading

Select the specific com port, then setting the baud rate and time out parameter.

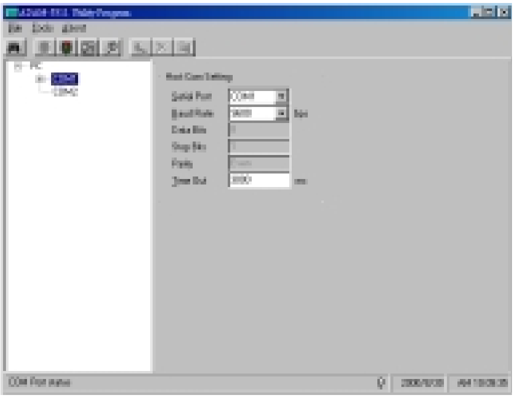


Figure 5-18: Setting the parameter of COM port

Baud rate: The communication speed (baud rate) can be configured from 9600 bps to 115.2 Kbps.

Timeout: Timeout means the time limit for waiting a response after the system has issued a command. If no response has been received when timeout has passed, we'll see the "Timeout !" message on the screen.

5.3.3 Search Connected Module

When you use the Search command, it will search for any connected modules on network and display their data. There are three ways to search for:

1. Click the Toolbar button:

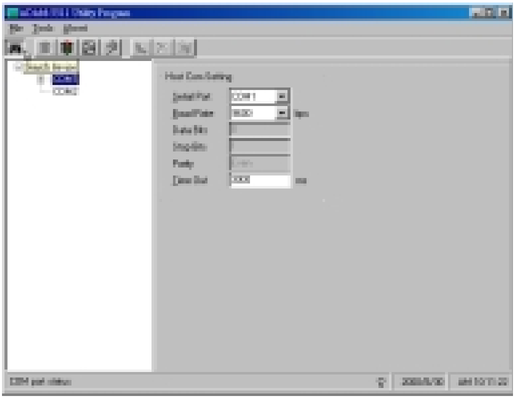


Figure 5-19: Click search button

2. Double click the left mouse button:

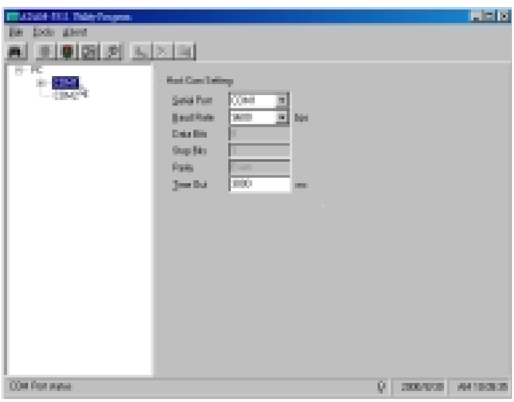


Figure 5-20: Double click left mouse button for search

3. Click the tool menu and choose the Scan device command:

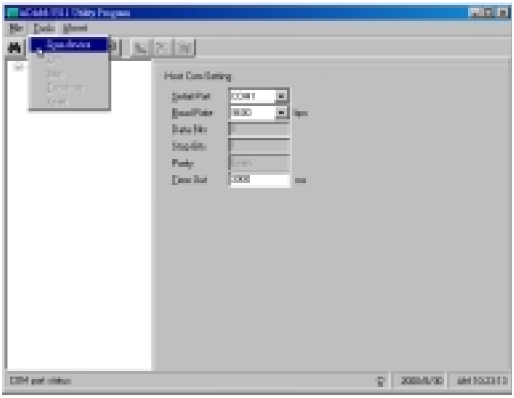


Figure 5-21: Choose Scan device command

The detected modules on network will show on the display.

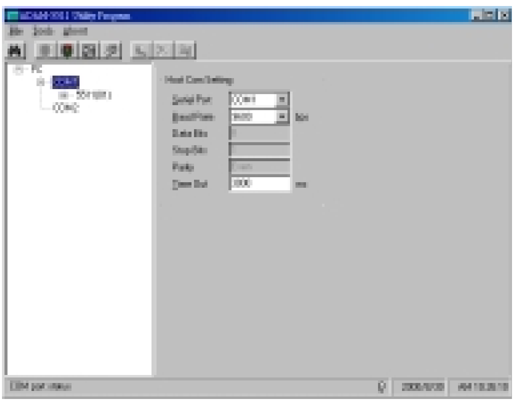


Figure 5-22: ADAM-5511 has been detected

5.3.4 Data Monitor

Windows Utility provides user a friendly environment. As you click the selected ADAM-5511, it will detect all module inserted on board.

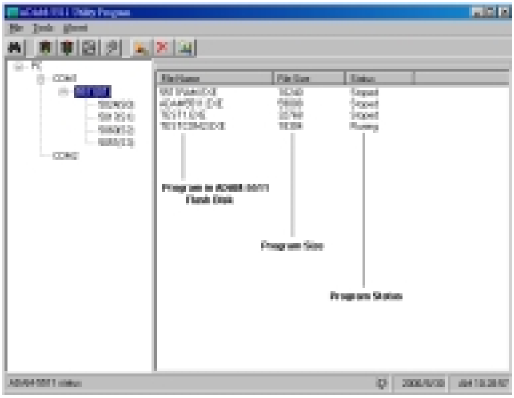


Figure 5-23: Auto-detect module on board

Select the specific module, the status will appeal without any configuration.

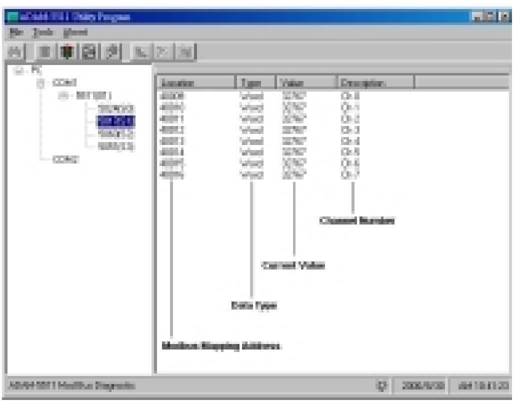


Figure 5-24: Module current status

Note: The range of AI/O module may be different by its resolution. For 12 bits resolution module, it scaled as 0~4095; for 16 bits resolution module, it scaled as 0~65535.

In addition, Windows Utility provides a powerful tool for user to read easily in engineer unit. Just double click the left mouse button on the data field, the scaling tool will pop up. Select the “Enable” item and setting the Min/Max value as the range of your engineer unit. See figure 5-25.

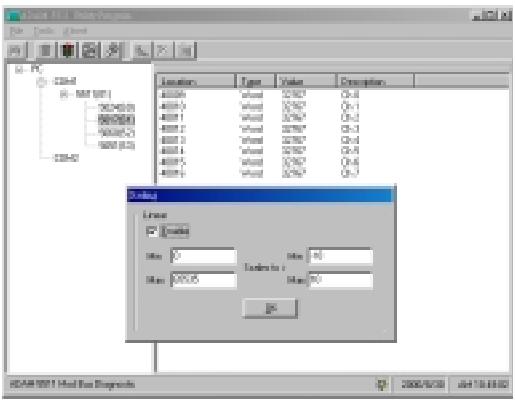


Figure 5-25: Data scaling tool

5.3.5 Data Force Output

In addition to data monitoring, this utility allows user to force both digital and analog output without any programming.

Digital output Force ON/OFF:

Select a digital output module and double click the specific point.

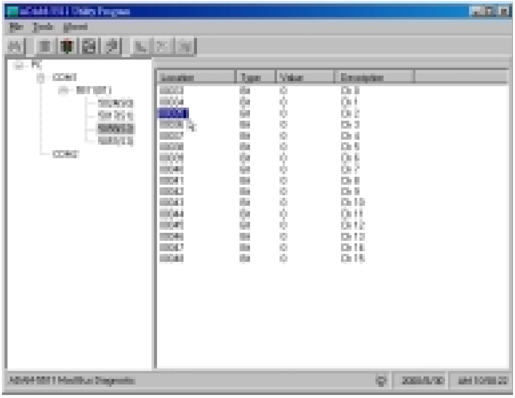


Figure 5-26: Double Click the specific point

Execute force on/off command via write coil function block

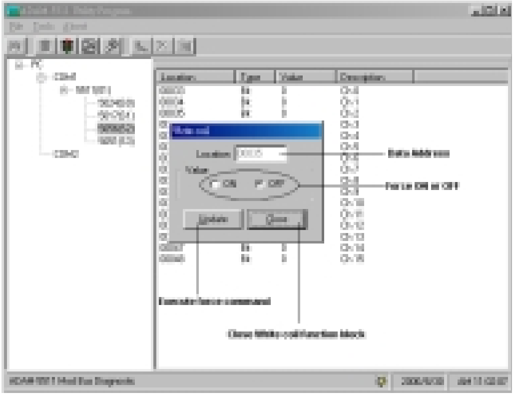


Figure 5-27: Write coil function block

Analog output Force value:

Select an analog output module and double click the specific point.

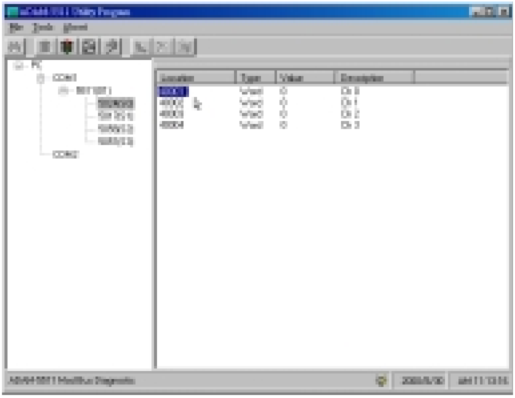


Figure 5-28: Select a specific analog point

Execute force output value command via write coil function block

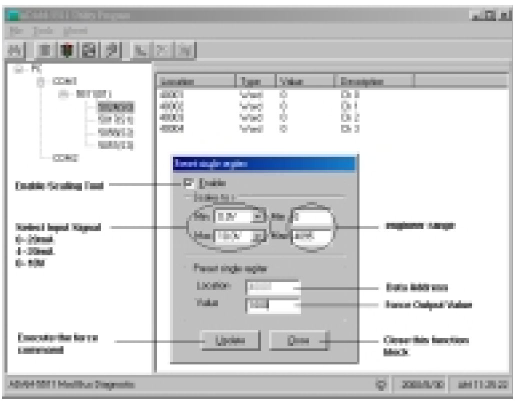


Figure 5-29: Preset single register function block

5.3.6 Download Procedure

Before you download any program to ADAM-5511, please stop each program, then follow these steps...

Step 1. Click the download icon and select the specific program in your PC.

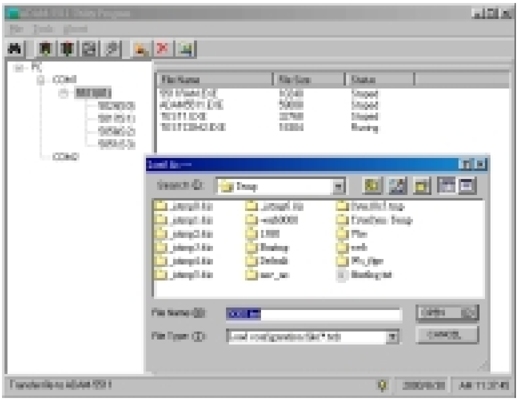


Figure 5-30: Select the specific file for download

Step 2. Wait for the file transfer from PC to ADAM-5511.

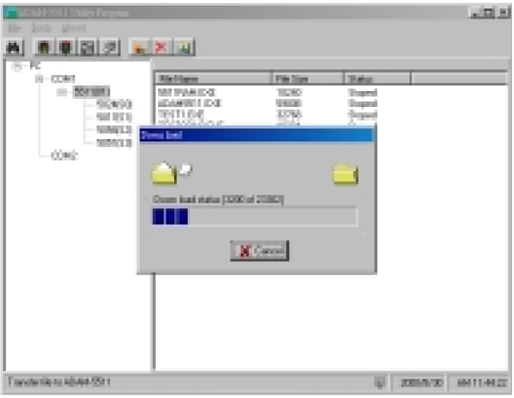


Figure 5-31: File transfer from PC to ADAM-5511

Step 3. Select the specific program, then click the "Run" icon.

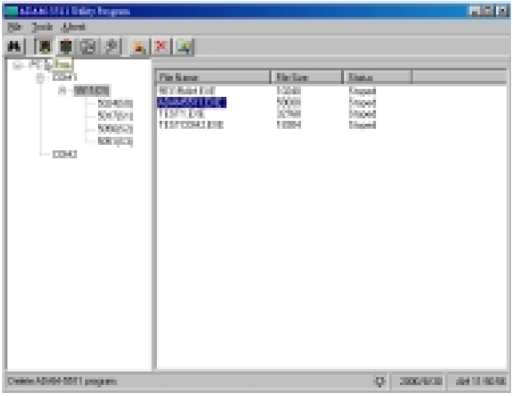


Figure 5-32: Run the program downloaded in ADAM-5511

5.3.7 Remote I/O

ADAM-5511 has expansion ability by integrating ADAM-4000 module as remote and distributed I/O. However, there are some steps you need to follow.

Step 1. Configure ADAM-4000 Module (Refer to chapter 5.2)

Step 2. Edit Program:

Please add the below section of program into your executive program.

Note: The address setting of ADAM-4000 module must be the same with your configuration in ADAM-4000/5000 Utility. For this example, we defined address 2 for ADAM-4011, address 3 for ADAM-4012, address 4 for ADAM-4013, address 5 for ADAM-4014, address 6 for ADAM-4017, address 7 for ADAM-4018, ad-

dress 8 for ADAM-4021, address 9 for ADAM-4050, address 10 for ADAM-4052, address 11 for ADAM-4053, address 12 for ADAM-4060, and address 13 for ADAM-4080.

```
#include    <5511drv.h>

void main()
{
unsigned char BitData, rddata, ex11, ex12, ex13, ex14, ex17, ex18, ex21,
ex50, ex52, ex53, ex60, ex80;
int  status;
unsigned int Data11, Data12, Data13, Data14, Data17, Data18, Data52,
Data21=1000,Data50 = 1,Data50i,Data53l,Data53h,Data60 = 1;
unsigned long lData80;
char i;

printf("Demo program of Remote ADAM-4XXX serier\n");

ADAM_BaudRate_Setup((unsigned  long)9600);

//detect whether ADAM-4011 module addressed "2" exist in the
network or not.
if( InitADAM4011(2, FALSE) ) ex11 = 1; else ex11 = 0;
//detect whether ADAM-4012 module addressed "3" exist in the
network or not.
if( InitADAM4012(3, FALSE) ) ex12 = 1; else ex12 = 0;
if( InitADAM4013(4, FALSE) ) ex13 = 1; else ex13 = 0;
```

```
if( InitADAM4014(5, FALSE) ) ex14 = 1; else ex14 = 0;
if( InitADAM4017(6, FALSE) ) ex17 = 1; else ex17 = 0;
if( InitADAM4018(7, FALSE) ) ex18 = 1; else ex18 = 0;
if( InitADAM4021(8, FALSE) ) ex21 = 1; else ex21 = 0;
if( InitADAM4050(9, FALSE) ) ex50 = 1; else ex50 = 0;
if( InitADAM4052(10, FALSE) ) ex52 = 1; else ex52 = 0;
if( InitADAM4053(11, FALSE) ) ex53 = 1; else ex53 = 0;
if( InitADAM4060(12, FALSE) ) ex60 = 1; else ex60 = 0;
if( InitADAM4080(13, FALSE) ) ex80 = 1; else ex80 = 0;

printf("I/O cards present :\n11,12,13,14,17,18,21,50,52,53,60,80\n");
printf("%2d %2d %2d %2d %2d %2d %2d %2d %2d %2d %2d
%2d",ex11,ex12,ex13,          ex14,ex17,ex18,ex21,ex53,ex60,ex80);

while(1)
  {
  //Get4017(1, 0, &Data)
  //void Set4060(int ID, void *pValue, int Bit, int Size)
  //Clear_Mem_Buffer()
  if( ex11 ) Get4011(2, &Data11);
  if( ex12 ) Get4012(3, &Data12);
  if( ex13 ) Get4013(4, &Data13);
  if( ex14 ) Get4014(5, &Data14);
  printf("\nAdam-4017 = ");
  if( ex17 )
      {
```

```
    for ( i = 0; i < 8; i++ )
      {
     Get4017(6, i, &Data17);
      printf("%d ",Data17);
       }
     }
printf("\nAdam-4018 = ");
if( ex18 )
     {
    for ( i = 0; i < 8; i++ )
      {
     Get4018(7, i, &Data18);
      printf("%d ",Data18);
       }
     }
if( ex21 ) Set4021(8, &Data21);
if( ex50 )
     {
   Get4050(9, &Data50i, 0, AByte);
   Set4050(9, &Data50, 0, AByte);
    Data50 = Data50<<1;
   if( Data50 > 256 ) Data50 = 1;
     }
if( ex52 )
   Get4052(10, &Data52, 0, AByte);
if( ex53 )
```

```
        Get4053(11, &Data53l, 0, AByte);
        Get4053(11, &Data53h, 8, AByte);
      if( ex60 )
         {
        Set4060(12, &Data60, 0, AByte);
        printf("\nAdam4060 = %d",Data60);
        Data60 = Data60<<1;
        if( Data60 > 8 ) Data60 = 1;
         }
      if( ex80 )
        for( i = 0; i < 4; i++)
          Get4080(13, i, &lData80);

      ADAMDelay(100000);

      if ( check_prog_stop() )
      exit(1);
    }
  }
```

Step 3. System Organization and Program Download

Set up the system as figure 5-32

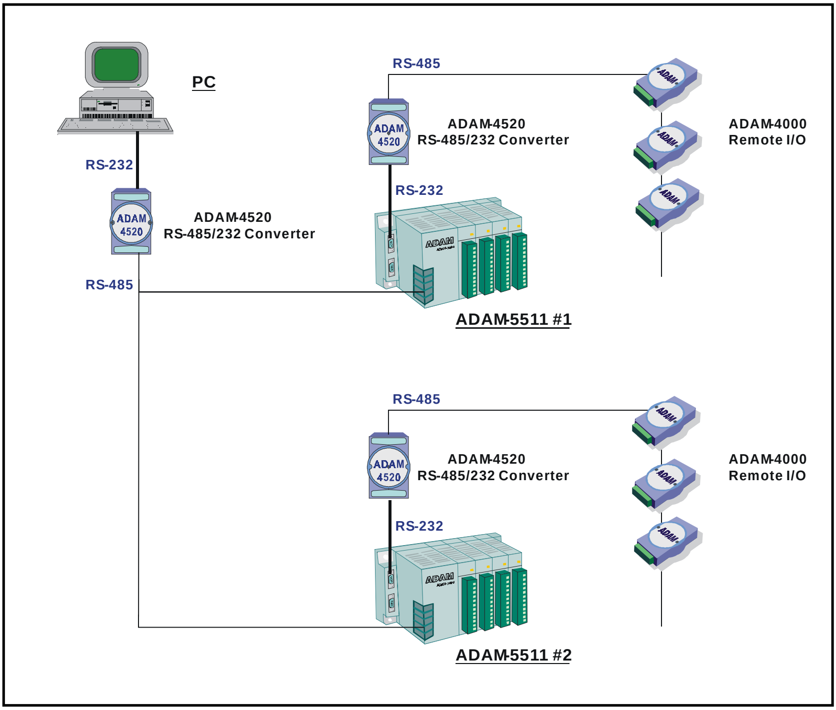


Figure 5-32: ADAM-5511 Remote I/O Organization

When remote I/O function applied in the application, there must be two ADAM-4520 in addition. One for PC monitoring and one for remote I/O network.

Programming and Downloading

Run ADAM-5511 Windows Utility and download the executive program with remote I/O function.

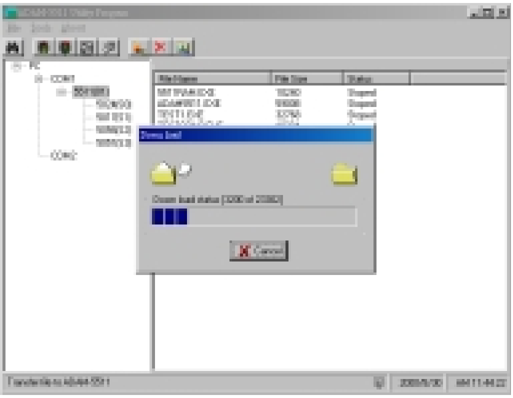


Figure 5-34: Download new executive program

Run the executive program, then click the specific 5511 again for search ADAM-4000 module.

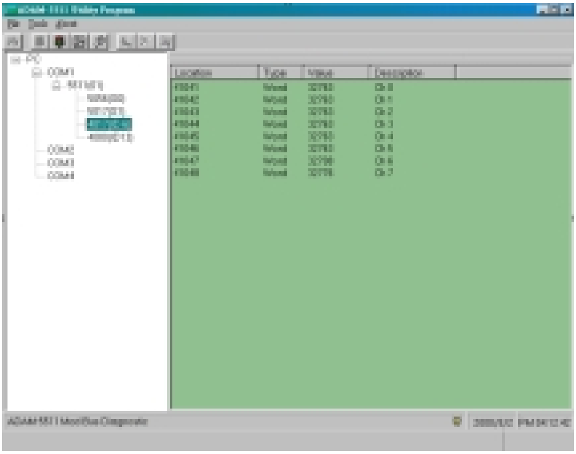


Figure 5-35: Executive the program for ADAM-4000 monitoring

Note: ADAM-5511 Utility will not detect ADAM-4000 module automatically before download the program with remote I/O and restart ADAM-5511.

5.3.8 Integrated with HMI

ADAM-5511 designed as a standard Modbus product and could be integrated with HMI via simple method. Here is an integration example for ADAM-5511 and Fix 6.0 HMI software.

Step 1. Configure the Communication Network

Define the communication setting of HMI, thus ADAM-5511 could be detected and recognized on the network.

Start System configuration and select configure/SCADA, then select MB1 and click configure. When MB1 Driver Configurator pop up, click setup.

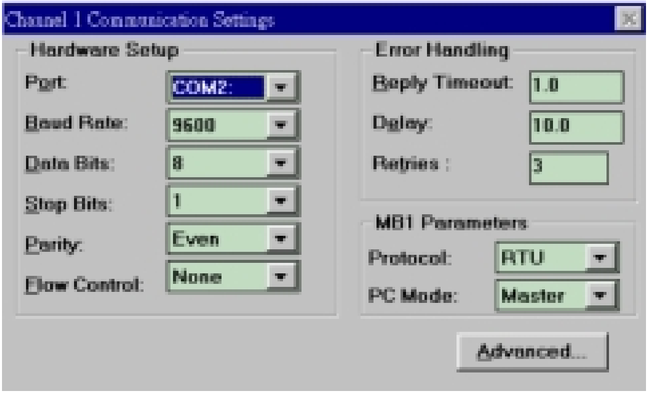


Figure 5-36: Network Setting of Fix software

Step 2. I/O Module Address Setting

Before integrating ADAM I/O Module into the database of HMI software, please refer the table as below to map ADAM-4000/5000 I/O address.

Analog type of ADAM-5000 I/O module:

Slot Position	**Address**
0	40001-40008
1	40009-40016
2	40017-40024
3	40025-40032

For example, if there is a ADAM-5024 (4-channel AO Module) in slot 2, the address of this module should be 40017~40020.

Note: *ADAM-5080 is a special 4-channel counter module. The data type is designed as “unsigned long”. When you insert an ADAM-5080 in slot 0, the address should be 40001, 40003, 40005 and 40007.*

Digital type of ADAM-5000 I/O module:

Slot Position	**Address**
0	00001-00016
1	00017-00032
2	00033-00048
3	00049-00064

Analog type of ADAM-4000 I/O module:

Node Number	Address
0	41001-41008
1	41009-41016
2	41017-41024
3	41025-41032
:	:
31	41249-41256

Digital type of ADAM-4000 I/O module:

Node Number	Address
0	01001-01016
1	01017-01032
2	01033-01048
3	01049-01064
:	:
31	01497-01512

Note: *There is a limitation when you apply ADAM-4000 as remote I/O. The maximum of any mixed ADAM-4000 module is 31 nodes.*

Step 3. Database Development

This is the last step to link ADAM-5511 and HMI software. Just fill the table to develop the database for ADAM-5511.

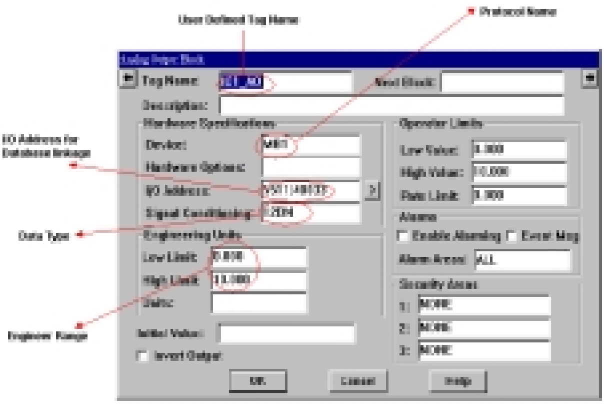


Figure 5-37: Database Setup Table of Fix software

HMI software will create a database table automatically.

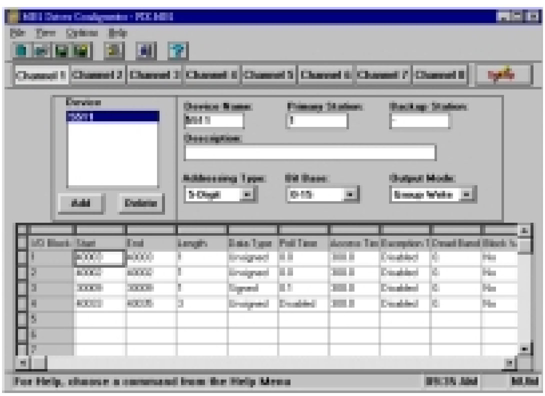


Figure 5-38: Database Table of Fix software

After completed the database development, it can be applied to various applications as your will.

Chapter 6

Function Library

Function Library

6.1 Introduction

User-designed ADAM-5511 application programs make use of ADAM-5511 library functions. To make the most efficient use of ADAM-5511's memory space, the ADAM-5511 function library has been separated into five smaller libraries. Therefore, a user can link only those libraries needed to run his application, and only those libraries will be included in the compiled executable. The smaller the linked libraries, the smaller the compiled executable will be.

Note 1: *These function libraries support Borland Turbo C++ 3.0 for DOS only.*

Note 2: *Please included all necessary ADAM-5511 function libraries in your project file.*

6.2 Library Classification

ADAM-5511 has five function libraries, categorized according to usage:

Category A. System Functions: (UTILITY.LIB)

Category B. Communication Functions: (COMM.LIB)

Category C. I/O Module Access Functions: (IO.LIB)

Category D. Remote I/O Module Access Functions: (RIO.LIB)

Category E. Serial Module Access Functions: (SIO.LIB)

6.3 Index

Category A. System Functions: (UTILITY.LIB)

Library Name	***Page***
ADAMdelay()	6-9
WDT_enable()	6-10
WDT_disable()	6-10
WDT_clear()	6-10
read_backup_ram()	6-14
write_backup_ram()	6-15
GetRTCtime()s	6-16
SetRTCtime()	6-18
Get_NodeID()	6-20
Get_BoardID()	6-22
check_prog_stop()	6-24
read_user_ram()	6-25
write_user_ram()	6-26

Table 6-1 *System Functions Library*

Category B. Communication Functions: (COMM.LIB)

Library Name	***Page***
com_232_set_format()	6-28
com_232_install()	6-30
com_232_deinstall()	6-34
com_232_set_speed()	6-35
com_232_tx()	6-36
com_232_tx_string()	6-37
com_232_rx()	6-38
com_232_tx_ready()	6-39
com_232_tx_empty()	6-40
com_232_rx_empty()	6-40

com_232_flush_tx()	6-42
com_232_flush_rx()	6-42
com_232_carrier()	6-43
com_232_lower_dtr()	6-44
com_232_raise_dtr()	6-44
com_232_raise_rts()	6-45
com_232_lower_rts()	6-45
com_232_set_break()	6-47
com_232_clear_break()	6-47
com_232_set_local_loopback()	6-48
com_232_clear_local_loopback()	6-48
com_232_enable_fifo()	6-49
com_232_disable_fifo()	6-49
com_232_read_scratch_register()	6-50
com_232_write_scratch_register()	6-50
com_232_set_line_params()	6-51
com_232_get_line_status()	6-51
com_232_get_modem_status()	6-51
RS232CallBackRoutine()	6-52
modem_command()	6-54
modem_initial()	6-55
modem_handup()	6-56
modem_autoanswer()	6-57
modem_command_state()	6-58
modem_dial()	6-59
CRC16()	6-60
checksum()	6-61

Table 6-2 *Communication Function Library*

Category C. I/O Module Access Functions: (IO.LIB)

Library Name	***Page***
Get5050()	6-62
Get5051()	6-62
Get5052()	6-62
Get 5055()	6-62
Set5050()	6-64
Set5055()	6-64
Set5056()	6-64
Set5060()	6-64
Set5068()	6-64
Get501718()	6-66
Get5017H()	6-68
Get5013()	6-70
Init5024()	6-72
Set5024()	6-73
Init5080()	6-74
Get5080()	6-75
GetRange5080()	6-77
Clear_Counter()	6-78
Start_Stop_Counter()	6-79
ReadOverflowFlag()	6-81
SetInitCounterVal()	8-82

Table 6-3 *I/O Module Access Function Library*

Category D. Remote I/O Module Access Functions: (RIO.LIB)

Table 6-4 *Remote I/O Access Function Library*

Category E. Serial Module Access Functions: (SIO.LIB)

char port_rx()	6-146
port_tx_empty()	6-147
port_tx()	6-148
port_tx_string()	6-149

Table 6-5 *Serial Module Access Function Library*

6.4 Function Library Description

6.4.1 System Utility Library (UTILITY.LIB)

ADAMdelay

Syntax:

void ADAMdelay(unsigned short msec)

Description:

Delays program operation by a specified number of milliseconds.

Parameter	Description
55 msec	From 0 to 65535.

Return value:

None.

Example:

```
void main(void)
{
// codes placed here by user
//delay 5.5 sec.
ADAMdelay(100);
//codes placed here by user
}
```

Remarks:

None.

WDT_clear, WDT_disable, WDT_enable

Syntax:

void WDT_clear(void)

void WDT_disable(void)

void WDT_enable(void)

Description:

Clear watchdog timer

Disable watchdog timer

Enable watchdog timer

Note: *When the watchdog timer is enabled, it will have to be cleared at least once every 1.5 seconds. The watchdog timer default value is "disable".*

Parameter	**Description**
None.	

Return value:

None.

Example:

```
void main()
{
int I;
WDT_enable();
For(I=0;I < 10;I++)
```

```
 {
 ADAMdelay(1000);
 WDT_clear();
 }
WDT_disable();
}
```

Remarks:

None.

Read_backup_ram

Syntax:

unsigned char read_backup_ram(unsigned int index)

Description:

Reads the value in backup RAM at index address, 60 KB total backup RAM, index = 0 - 61439; absolute addresses from 0x30000 - 0x3EFFF.

Parameter	**Description**
index	From 0 to 61439, 60 KB in total

Return value:

The single-byte value in backup RAM at address index.

Example:

```
void main(void)
{
unsigned char data;
//put your codes here
data = read_backup_ram(500);
}
```

Remarks:

None.

Write_backup_RAM

Syntax:

void write_backup_RAM(unsigned int index, BYTE data)

Description:

Writes a byte to battery backup memory.

Parameter	Description
index	An index for data in the battery backup RAM, from 42000 to 46097; 60 KB battery backup SRAM in total.
data	A byte of data that the programmer wants to write to battery-protected SRAM.

Return value:

None.

Example:

```
void main()
{
unsigned char data=0x55;
//Writes the data 0x55 into battery backup memory, index 10
write_backup_RAM(10,data);
}
```

Remarks:

None.

GetRTCtime

Syntax:

unsigned char GetRTCtime(unsigned char Time)

Description:

Reads Real-Time Clock chip timer. A user can activate a program on the date desired.

Parameter	Description
Time	RTC_sec the second RTC_min the minute RTC_hour the hour RTC_day the day RTC_week day of the week RTC_month the month RTC_year the year RTC_century the century

Return value:

The value requested by the user.

Example:

```
void main(void)
{
printf("\n Century = %d",
GetRTCtime(RTC_century) );
printf("\n Year = %d", GetRTCtime(RTC_year) );
printf("\n month = %d", GetRTCtime(RTC_month) );
printf("\n weekday = %d", GetRTCtime(RTC_week) );
```

```
printf("\n day = %d", GetRTCtime(RTC_day) );
printf("\n hour = %d", GetRTCtime(RTC_hour) );
printf("\n min = %d", GetRTCtime(RTC_min) );
printf("\n sec = %d", GetRTCtime(RTC_sec) );
}
```

Remarks:

None.

SetRTCtime

Syntax:

void SetRTCtime(unsigned char Time, unsigned char data)

Description:

Sets date and time of the real-time clock.

Parameter	Description
Time	RTC_sec the second RTC_min the minute RTC_hour the hour RTC_day the day RTC_week day of the week RTC_month the month RTC_year the year RTC_century the century
data	New contents.

Return value:

None.

Example:

```
void main()
{
unsigned char sec=0, min=0, hour=12;
//set current time 12:00:00
SetRTCtime(RTC_sec,sec);
SetRTCtime(RTC_min,min);
```

```
SetRTCtime(RTC_hour,hour);
}
```

Remarks:

None.

Get_NodeID

Syntax:

unsigned char Get_NodeID(void)

Description:

Gets the DIP switch number of the ADAM-5511 controller.

Parameter	Description
None.	

Return value:

The DIP switch number of the ADAM-5511 controller.

Example:

```
unsigned char SystemNodeNumber;
void main()
{
SystemNodeNumber = Get_NodeID();
If(SystemNodeNumber == 0x15)
 {
 //put your code in Here
 }
else
 {
printf("\nNode number Error!");
 }
```

```
}
```

Remarks:

None.

Get_BoardID

Syntax:

unsigned char Get_BoardID(int Board)

Description:

Gets the type identification of the I/O module in a controller slot.

Parameter	Description
Int Board	The slot number of an ADAM-5511, from 0 to 3.

Return value:

The return values are:

Example:

```
unsigned char IOModuleName;
unsigned char SlotNumber;
void main(void)
{
//Read IO module name in Slot 0
SlotNumber = 0;
IOModuleName = Get_BoardID(SlotNumber);
If(IOModuleName == ADAM5051_ID)
 {
 //IO Board is current, put your code in Here
 }
else
```

```
{
printf("\nThe IO Board is NOT ADAM5051");
printf("\nPlease Check your system setup");
}
}
```

Remarks:

None.

Check_Prog_Stop

Syntax:

void check_prog_stop(void)

Description:

Check program stop command and stop the executive program

Parameter	**Description**
None	

Return value:

0	detect no program stop command
1	detect program stop command and stop the executive program

Example:

```
void main()
{
//check program stop command. If the command is detected, stop this executive program
check_prog_stop();
}
```

Remarks:

The program stop command is come from the “stop program” function icon of ADAM-5511 Windows Utility. User have to add this library in the end of each program, thus the “stop program” function icon will take effect.

Read_User_RAM

Syntax:

unsigned int read_user_ram(unsigned int index)

Description:

Read value from user RAM

Parameter	Description
Index	The index of Modbus memory address 0~511 mapping to Modbus address 42001~46097

Return value:

The value in the specific address.

Example:

```
void main()
{
//Read address 42002
read_user_ram(1);
}
```

Remarks:

None.

Write_User_RAM

Syntax:

void write_user_ram(unsigned int index, unsigned int data)

Description:

Write value to user RAM

Parameter	**Description**
Index	The index of Modbus memory address 0~511 mapping to Modbus address 42001~42512
data	The value you would like to write into the address

Return value:

None

Example:

```
void main()
{
unsigned int Value;

//Read address 42004
Vaule = read_user_ram(3);

//Write "value" into address 42001
```

```
write_user_ram(0);
}
```

Remarks:

None.

6.4.2 Communication Function Library (COMM.LIB)

com_232_set_format

Syntax:

void com_232_set_format(int data_length, int parity, int stop_bit)

Description:

Sets the parameters data length, parity and stop bits of the RS-485 port.

Parameter	**Description**
data_length	Valid range 5 to 8 bits for one character.
parity	0: no parity 1: odd parity 2: even parity
stop_bit	1: 1 stop bit 2: 2 stop bits

Return value:

None.

Example:

```
void main()
{
//Sets the data format of the RS-232 port to 8-bit data length, no parity,
1 stop bit
com_232_set_format(8, 0, 1);
}
```

Remarks:

None.

com_232_install

Syntax:

int com_232_install(char IsCustomerDefine)

Description:

Allocates the interrupt registers of the microprocessor for use by the RS-232 port and sets the interrupt vector to the interrupt service routine.

Parameter	**Description**
IsCustomerDefine	Communication interrupt status

Return value:

0	Installation success
1	Installation fail

Example:

```
void main()
{
unsigned char status,IsCustomDefine=0,rddata;
// If you want to process communication interrupt, pleases set IsCustomDefine to 1, otherwise set to 0

status = com_232_install(IsCustomDefine);
if( status == 0 )
  {
    printf("\nRS232 port intall ok !");
```

```
    com_232_set_speed((unsigned long)9600);
    com_232_set_format(8,0,1);
    printf("\nBR = 9600, 8 bit, even ,stop bit = 1");
  }
  else
    printf("\nRS232 port install failed!");
}
```

//This is a very important library. As you using RS-232 port, please remember to add it in the end of your program.

```
RS232CallBackRoutine(unsigned char rddata)
{
  //your interrupt program
}
```

Remarks

None.

com_232_install

Syntax:

int com_232_install(void)

Description:

Allocates the interrupt registers of the microprocessor for use by the RS-232 port and sets the interrupt vector to the interrupt service routine.

Parameter	**Description**
None.	

Return value:

integer; Installation status.

0 = Successful installation

1 = Drivers are already installed

Example:

```
void main()
{
int status;
status = com_232_install();
if( status ==0)
 printf("\n The allocation of COM2 port (RS-232) is OK !");
 else
 exit(0);
}
```

Remarks

None.

com_232_deinstall

Syntax:

void com_232_deinstall(void)

Description:

Releases the interrupt register of the microprocessor for use by the RS-232 port without changing the baud rate or DTR.

Parameter Description

None.

Return value:

None.

Example:

```
void main()
{
//Releases the interrupt register for use by the RS-232 port
com_232_deinstall();
}
```

Remarks:

This function MUST be called before returning to DOS. The interrupt vector will not be pointed to the interrupt service routine again.

com_232_set_speed

Syntax:

void com_232_set_speed(unsigned long speed)

Description:

Sets the baud rate of the RS-232 port.

Parameter	Description
speed	The baud rate value.

Return value:

None.

Example:

```
void main()
{
//Sets the baud rate of the RS-232 port to 9600bps
com_232_set_speed(9600L);
}
```

Remarks:

None.

com_232_tx

Syntax:

void com_232_tx(char c)

Description:

This function sends a single character to the Tx pin of the RS-232 port, waits until the last bit is sent to the remote terminal, and then sets the RTS pin to OFF.

Parameter Description

c The character you would like to send.

Return value:

None.

Example:

```
void main()
{
com_232_tx(0x03);
com_232_tx('$');
}
```

Remarks:

None.

com_232_tx_string

Syntax:

void com_232_tx_string(char *s)

Description:

com_232_tx_string() sends a string by calling com_232_tx() repeatedly.

Parameter	**Description**
s	The string you would like to send.

Return value:

None.

Example:

```
void main()
{
com_232_tx_string("This is a string test.");
}
```

Remarks:

None.

com_232_rx

Syntax:

char com_232_rx(void)

Description:

Returns the next character from the receiving buffer, or a NULL character('\0') if the buffer is empty.

Parameter	**Description**
None.	

Return value:

c	The return character.

Example:

```
void main()
{
char C232data;
C232data=com_232_rx();
}
```

Remarks:

None.

com_232_tx_ready

Syntax:

int com_232_tx_ready(void)

Description:

Check data transmitted already

Parameter	Description
None.	

Return Value:

0 : data not ready

1 : data ready

Remarks:

None.

com_232_rx_empty()

com_232_tx_empty()

Syntax:

int com_232_rx_empty(void)

int com_232_tx_empty(void)

Description:

Returns the status of the COM2 (RS-232) transmitting and receiving queues.

Parameter	Description
None.	

Return value:

Com_232_rx_empty() returns "TRUE" if the receiving queue is empty.

Com_232_tx_empty() returns "TRUE" if the transmitting queue is empty.

Example:

```
void main()
{
unsigned char data;
if( com_232_rx_empty()== FALSE)
data =com_232_rx();
}
```

Remarks:

The COM2 (RS-232) transmitter uses polling-action (not interrupt-action). Its queue is always empty.

com_232_flush_rx()

com_232_flush_tx()

Syntax:

void com_232_flush_rx(void)

void com_232_flush_tx(void)

Description:

COM2 (RS-232) buffer flusher. Initializes the transmitting and receiving queues to their empty states.

Parameter	**Description**
None.	

Return value:

None.

Example:

```
void main()
{
com_232_flush_rx();
com_232_flush_tx();
}
```

Remarks:

The COM2 (RS-232) transmitter uses polling-action (not interrupt-action). Its buffer is always flushed.

com_232_carrier

Syntax:

int com_232_carrier(void)

Description:

Detects the carrier signal of 232 port.

Parameter	**Description**
None.	

Return value:

TRUE: If a carrier is present.

FALSE: No carrier.

Example:

```
void main(void)
{
if( com_232_carrier() == TRUE )
 {
 //Telephone carrier signal presented at 232 port, put your associate
program here
 }
}
```

Remarks:

None.

com_232_lower_dtr

com_232_raise_dtr

Syntax:

void com_232_lower_dtr(void)

void com_232_raise_dtr(void)

Description:

Sets 232 port to DTR for low signal.

Sets 232 port to DTR for high signal.

Parameter	Description
None.	

Return value:

None.

Example:

None.

Remarks:

Please refer to the 16C550 UART register document (Appendix B).

com_232_lower_rts

com_232_raise_rts

Syntax:

void com_232_lower_rts(void)

void com_232_raise_rts(void)

Description:

Sets 232 port to RTS for low signal.

Sets 232 port to RTS for high signal.

Parameter	**Description**
None.	

Return value:

None.

Example:

```
void main(void)
{
//handshaking with external serial device
com_232_lower_rts();

//generates a signal of 500 ms low trigger
ADAMdelay(500);
com_232_raise_rts();
```

```
}
```

Remarks:

Please refer to the 16C550 UART register document (Appendix B).

com_232_clear_break

com_232_set_break

Syntax:

void com_232_clear_break(void)

void com_232_set_break(void)

Description:

Sets 232 port to clear BREAK signal.

Sets 232 port to send BREAK signal.

Parameter	Description
None.	

Return value:

None.

Example:

None.

Remarks:

Please refer to the 16C550 UART register document (Appendix B).

com_232_clear_local_loopback

com_232_set_local_loopback

Syntax:

void com_232_clear_local_loopback(void)

void com_232_set_local_loopback(void)

Description:

Sets 232 port to disable loopback function for diagnostic.

Sets 232 port to enable loopback function for diagnostic.

Parameter	Description
None.	

Return value:

None.

Example:

None.

Remarks:

Please refer to the 16C550 UART register document (Appendix B).

com_232_disable_fifo

com_232_enable_fifo

Syntax:

void com_232_disable_fifo(void)

int com_232_enable_fifo(void)

Description:

Sets 232 port to disable fifo receiving trigger level 1, 4, 8, 14.

Sets 232 port to enable fifo receiving trigger level 1, 4, 8, 14.

Parameter	Description
None.	

Return value:

0: success.

-1: fifo not available.

-10: failure to enable.

Example:

None.

Remarks:

Please refer to the 16C550 UART register document (Appendix B).

com_232_read_scratch_register

com_232_write_scratch_register

Syntax:

int com_232_read_scratch_register(void)

void com_232_write_scratch_register(int value)

Description:

Reads from COM port scratch register.

Writes to COM port scratch register.

Parameter	**Description**
value	Integer value one byte in length, as signed by user from the range 0 to FF.

Return value:

Please refer to the 16C550 UART register document (Appendix A).

Example:

None.

Remarks:

This byte is reserved for the user. Please refer to the 16C550 UART register document (Appendix A).

com_232_get_line_status,

com_232_set_line_params,

com_232_get_modem_status

Syntax:

int com_232_get_line_status(void)

int com_232_set_line_params(unsigned lineparams)

int com_232_get_modem_status(void)

Description:

Reads from COM port line control register.

Writes to COM port line control register.

Reads from COM port modem status register.

Parameter	Description
lineparams	Please refer to the UART specifications.

Return value:

Please refer to the 16C550 UART register document (Appendix A).

Example:

None.

Remarks:

None.

RS232CallbackRoutine

Syntax:

void RS232CallBackRountine(unsigned char rddata)

Description:

Call subroutine applied for RS-232 port

Parameter	Description
rddata	default character

Return value:

None.

Example:

```
void main()
{
unsigned char status,IsCustomDefine=0,rddata;
status = com_232_install(IsCustomDefine);
if( status == 0 )
  {
     printf("\nRS232 port intall ok !");
      com_232_set_speed((unsigned long)9600);
     com_232_set_format(8,0,1);
      printf("\nBR = 9600, 8 bit, even ,stop bit = 1");
```

```
 }
 else
    printf("\nRS232 port install failed!");
}

RS232CallBackRoutine(unsigned char rddata)
{
 //your interrupt program
}
```

Remarks:

This is a very important library. As you using RS-232 port, please remember to add it in the end of your program.

modem_command

Syntax:

void modem_command(char *cmdstr)

Description:

Sends an AT command string to the modem. For details, refer to the AT command document provided by the manufacturer.

Parameter	Description
cmdstr	Specifies command string; refer to AT command string.

Return value:

None.

Example:

```
void main(void)
{
//initialize modem
modem_command("atz");
}
```

Remarks:

None.

modem_initial

Syntax:

void modem_initial(void)

Description:

Sets modem to initial status. Due to the ADAM5511 system's construction, the modem can only be connected to COM1. This resets the modem to the initial state. The command has the same effect as sending the ASCII command "atz" to the modem.

Parameter	Description
None.	

Return value:

None.

Example:

```
void main()
{
//you need to initialize COM1
modem_initial();
//put your modem function...
}
```

Remarks:

None.

modem_handup

Syntax:

```
void modem_handup(void)
```

Description:

Sets the modem to hand up the telephone. The command has the same effect as sending the ASCII command "atho" to the modem.

Parameter	Description
None.	

Return value:

None.

Example:

```
void main()
{
//close phone
modem_handup();
}
```

Remarks:

None.

modem_autoanswer

Syntax:

void modem_autoanswer(void)

Description:

Sets up modem to auto answer phone calls.

Parameter	**Description**
None.	

Return value:

None.

Example:

```
void main()
{
//set modem auto answer and waiting phone call
modem_autoanswer();
}
```

Remarks:

None.

modem_command_state

Syntax:

void modem_command_state(void)

Description:

Sets modem to command mode. In other words, this causes the modem to escape from data mode to command mode. The modem will delay at least 3 seconds before switching back to command mode. This command has the same effect as sending the ASCII command "+++" to the modem.

Parameter	Description
None.	

Return value:

None.

Example:

```
void main()
{
//receiving data from modem, so modem is in transfer data mode
modem_command_state();
//now, you can send an AT command string to modem
}
```

Remarks:

None.

modem_dial

Syntax:

void modem_dial(char *telenum)

Description:

Directs modem to connect to the specified telephone number.

Parameter	Description
telenum	The phone number you would like to dial out.

Return value:

None.

Example:

```
void main()
{
//COM port and modem initial OK
//set the dial out number as 886222184567
modem_dial("886222184567");
//waiting for link
}
```

Remarks:

None.

CRC16

Syntax:

unsigned int CRC16(char *data_p, unsigned int length)

Description:

Calculates the CRC 16-bit value of the string *data_p.

Parameter	Description
*data_p	The string which you want to calculate CRC code.
length	The length of string *data_p.

Return value:

The CRC16 code.

Example:

```
unsigned char String[]="this is a test CRC16";
void main()
{
unsigned int code;
code = CRC16(String, strlen(String));
printf("\n The string %s CRC16 code = %d", String, Code);
}
```

Remarks:

None.

Checksum

Syntax:

unsigned int checksum(void *buffer, int len, unsigned int seed)

Description:

Calculates the checksum of the string or data array in the string buffer.

Parameter	Description
buffer	The string for which a user wants to calculate the checksum.
len	The length of the data array in the buffer.
seed	A seed value added into the checksum for the purpose of calculation or security.

Return value:

The checksum of the data array buffer.

Example:

```
unsigned char String[]=”this is a test CheckSum”;
void main(void)
{
unsigned int code;
code = checksum(String, strlen(String),0);
}
```

Remarks:

None.

6.4.3 I/O Module Access Functions Library (IO.LIB)

Get5050, Get5051, Get5052, Get5055

Syntax:

void Get5050(int Board, int Bit, int Size, void *pValue)

void Get5051(int Board, int Bit, int Size, void *pValue)

void Get5052(int Board, int Bit, int Size, void *pValue)

void Get5055(int Board, int Bit, int Size, void *pValue)

Description:

Reads the data value in an I/O module.

Parameter	**Description**
Board	ADAM-5511 slot number, from 0 to 3.
Bit	See "Size" parameter below.
Size	ABit, AByte, AWord If Size= ABit, Bit=0..15 (pin0..pin15) If Size=AByte, Bit=0 for Low Byte data; Bit=8 for High Byte data If Size=AWord, Bit does not care. Always word data.
pValue	The value returned.

Return value:

None.

Example:

void main()

```
{
unsigned char Bdata;
unsigned int Wdata;
//Slot0, pin13, data=0 or 1
Get5051(0, 13, ABit, &Bdata);

//Slot2, pin0~pin7, Bdata=Low Byte data
Get5051(2, 0, AByte, &Bdata);

// Slot3, pin0~pin15, Wdata=Word data
Get5051(3, 0, AWord, &Wdata);
}
```

Remarks:

None.

Set5050, Set5055, Set5056, Set5060, Set5068

Syntax:

void Set5050(void *pValue, int Board, int Bit, int Size)

void Set5055(void *pValue, int Board, int Bit, int Size)

void Set5056(void *pValue, int Board, int Bit, int Size)

void Set5060(void *pValue, int Board, int Bit, int Size)

void Set5068(void *pValue, int Board, int Bit, int Size)

Description:

Sets the digital output for ADAM-5050, ADAM-5055, ADAM-5056, ADAM-5060 and ADAM-5068 modules to the specified values.

Parameter	**Description**
pValue	The digital value specified by the user to be output
Board	0 to 3 (Slot0 .. Slot3)
Bit	See “Size” parameter below
Size	ABit, AByte, AWord If Size = ABit, Bit = 0...15 (pin0 ... pin15) If Size = AByte, Bit = 0 is Low Byte data Bit = 8 is High Byte data If Size = AWord, Bit does not care, always word data

Return Value:

None.

Example:

```
void main()
{
unsigned char Bitdata = 1;
//Output 1 to slot 0, pin 13
Set5056( &Bitdata, 0, 13, ABit);
}
```

Remarks:

None.

Get501718

Syntax:

void Get501718(int Board, int Channel, void *pValue)

Description:

Reads the data value in an I/O module.

Parameter	Description
Board	0 - 3 for Slot0 ...Slot3
Channel	0 - 6 for ADAM-5018 0 - 7 for ADAM-5017
*pValue	The value returned

*Note: The *pValue for ADAM-5017 and ADAM-5018 must be interpreted in reference to the range input that was set during module configuration.*

Return value:

None.

Example:

```
void main()
{
int *value, j;
//One ADAM-5018 (ADAM-5017) module on slot 3 of the ADAM-
5511
printf("Get ADAM5018(or ADAM5017)....\n");
```

```
for (j=0;j<7;j++)
{
//Get ADAM-5018 data and range from channel 0 to 6 on slot 3 of
ADAM-5511
Get501718(3,j,value);
}
}
```

Remarks:

None.

Get5017H

Syntax:

void Get5017H(int Board, int Channel, void *pValue)

Description:

Reads the data value in ADAM-5017H module.

Parameter	Description
Board	0 - 3 for Slot0 ...Slot3
Channel	0 - 7 for ADAM-5017
*pValue	The value returned

Return value:

None.

Example:

```
void main()
{
int *value, j;
//One ADAM-5017H module on slot 3 of the ADAM-5511
printf("Get ADAM5017H....\n");
for (j=0;j<8;j++)
 {
 //Get ADAM-5017H data from channel 0 to 7 on slot 3 of ADAM-5511
 Get5017H(3,j,value);
 }
}
```

Remarks:

None.

Get5013

Syntax:

void Get5013(int Board, int Channel, void *pValue)

Description:

Reads the data value in an ADAM-5013 module.

Parameter	Description
Board	0 - 3 for Slot0 ...Slot3
Channel	0 - 2 for ADAM-5013
*pValue	The value returned

Note: *The *pValue for ADAM-5013 must be interpreted in reference to the input range that was set during module configuration.*

Return Value:

None.

Example:

```
void main()
{
int *value, j;
//One ADAM-5013 module on slot 0 of the ADAM-5511
printf("Get ADAM-5013 Value....\n");
for (j=0;j<3;j++)
```

```
{
//Get ADAM-5013 data and range from channel 0 to 2 on slot 0 of
ADAM-5511
Get5013(0,j,value);
}
}
```

Remarks:

None.

Init5024

Syntax:

void Init5024(int Slot, int ch0_val, int ch1_val, int ch2_val, int ch3_val)

Description:

Initializes ADAM-5024 module in the slot indicated, loading user-specified analog output values into each of the modules' four channels.

Parameter	**Description**
ch0_val	The initial value output by channel 0
ch1_val	The initial value output by channel 1
ch2_val	The initial value output by channel 2
ch3_val	The initial value output by channel 3

Return Value:

None.

Example:

```
void main()
{
// initializes outputs of all channels of the ADAM-5024 in slot 0 to
output a value of 0
Init5024 (0,0,0,0,0);
}
```

Remarks:

None.

Set5024

Syntax:

void Set5024(void *pValue, int Board, int Channel)

Description:

Specifies the output of a channel of a selected ADAM-5024.

Parameter	Description
*pValue	The value set for analog output
Board	Slot number = 0 - 3
Channel	AO channel = 0 - 3

Return Value:

None.

Remarks:

None.

Init5080

Syntax:

void Init5080(int slotno)

Description:

Initial ADAM-5080 Module

Parameter	**Description**
slotno	The specific slot inserted with ADAM-5080 0-3 or slot0-slot3

Return Value:

None

Example:

```
void main ()
{
//initializes the ADAM-5080 Module in slot 0
init5080(0);
}
```

Remarks:

None.

Get5080

Syntax:

void Get5080(int slotno, int channel, long *pValue)

Description:

Get Value from specific channel in ADAM-5080

Parameter	Description
slotno	The specific slot inserted with ADAM-5080 0-3 or slot0-slot3
channel	The specific channel in ADAM-5080 0-3
*pValue	The Value returned

Return Value:

The Value from the specific channel

Example:

```
void main ()
{
unsigned long int aiv[4];
int i;
for(i=0;i<4;i++)
//get each value from ADAM-5080 in slot 0
Get5080(0, i, & (aiv[i]));
}
```

Remarks:

None.

GetRange5080

Syntax:

void GetRange5080(int Board, void *pValue)

Description:

Reads the counter range of ADAM-5080 module.

Parameter	**Description**
Board	0 - 3 for Slot0 ...Slot3.
*pValue	The counter range code returned.

Return Value:

None.

Remarks:

None

Clear Counter

Syntax:

int Clear_Counter(int slotno, int channel)

Description:

Reset the current counter value to its initial value

Parameter	**Description**
slotno	The specific slot inserted with ADAM-5080 0-3 or slot0-slot3
channel	The specific channel in ADAM-5080 0-3

Return Value:

None

Example:

```
void main ()
{
//reset ADAM-5080 channel 0 counter value in slot 0
int Clear_Counter(0, 0);
}
```

Remarks:

None.

Start/Stop Counter

Syntax:

int Stop_Start_Counter(int slotno, int channel, StartOrStop)

Description:

Start or stop the specific counter

Parameter	**Description**
slotno	The specific slot inserted with ADAM-5080 0-3 or slot0-slot3
channel	The specific channel in ADAM-5080 0-3
Start	1
Stop	0

Return Value:

None

Example:

```
void main ()
{
int Start=1, Stop=0;
//Start counter
ids=Start_Stop_Counter(0, 0, 1);
//if the returned value is 0, print out the start fail message
if(ids==0)
```

```
printf('start failed\n");
}
```

Remarks:

None.

Read Overflow

Syntax:

void ReadOverflowFlag(int Board, int Channel)

Description:

Check if counter value reach max. count limit

Parameter	Description
int Board	The specific slot inserted with ADAM-5080 0-3 or slot0-slot3
int Channel	The value returned

Return Value:

The overflow value returned

Example:

```
void main ()
{
char overflag_value[4];
int i;
ReadOverflowFlag(0, &(overflag_value[0]));
for (i=0;i<4;i++)
printf("channel %d over_flag_value=%d\n",i,overflag_value[i]);
}
```

Remarks:

None.

Set Initial Value

Syntax:

int SetInitCounterVal(int slotno, int channel, unsigned long Value)

Description:

Set initial counter value (between 0 to 4,294,967,295)

Parameter	Description
slotno	The specific slot inserted with ADAM-5080 0-3 or slot0-slot3
channel	The specific channel in ADAM-5080 0-3

Return Value:

None

Example:

```
void main ()
{
unsigned long int i;
i=1000;
//set 1000 to the initial counter value
SetInitCounterVal(0,0,i);
}
```

Remarks:

None.

6.4.4 Remote I/O Module Access Functions Library (RIO.LIB)

Baud Rate Setup

Syntax:

void ADAM_BaudRate_Setup(unsigned long speed)

Description:

Set the baud rate of ADAM-4000 Remote I/O Network

Parameter	Description
speed	Baud Rate Setting 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200

Return Value:

None

Example:

```
void main ()
{
//set the baud rate of ADAM-4000 I/O as 9600bps
ADAM_BaudRate_Setup(9600);
}
```

Remarks:

None.

Init4011

Syntax:

char initADAM4011(int ID, char ChkSumEn)

Description:

Initializes the network ID and check Sum for ADAM-4011 module

Parameter	**Description**
ID	Node number on the network 1~32
ChkSumEn	Check Sum Enable True, False

Return Value:

None

Example:

```
void main ()
{
//initialize ADAM-4011 as ID#2, Check Sum False
initADAM4011(2, False);
}
```

Remarks:

None.

Init4011D

Syntax:

char initADAM4011D(int ID, char ChkSumEn)

Description:

Initializes the network ID and check Sum for ADAM-4011D module

Parameter	**Description**
ID	Node number on the network 1~32
ChkSumEn	Check Sum Enable True, False

Return Value:

None

Example:

```
void main ()
{
//initialize ADAM-4011D as ID#2, Check Sum False
initADAM4011D(2, False);
}
```

Remarks:

None.

Init4012

Syntax:

char initADAM4012(int ID, char ChkSumEn)

Description:

Initializes the network ID and check Sum for ADAM-4012 module

Parameter	**Description**
ID	Node number on the network 1~32
ChkSumEn	Check Sum Enable True, False

Return Value:

None

Example:

```
void main ()
{
//initialize ADAM-4012 as ID#3, Check Sum False
initADAM4012(3, False);
}
```

Remarks:

None.

Init4013

Syntax:

char initADAM4013(int ID, char ChkSumEn)

Description:

Initializes the network ID and check Sum for ADAM-4013 module

Parameter	**Description**
ID	Node number on the network 1~32
ChkSumEn	Check Sum Enable True, False

Return Value:

None

Example:

```
void main ()
{
//initialize ADAM-4013 as ID#4, Check Sum False
initADAM4013(4, False);
}
```

Remarks:

None.

Init4017

Syntax:

char initADAM4017(int ID, char ChkSumEn)

Description:

Initializes the network ID and check Sum for ADAM-4017 module

Parameter	Description
ID	Node number on the network 1~32
ChkSumEn	Check Sum Enable True, False

Return Value:

None

Example:

```
void main ()
{
//initialize ADAM-4017 as ID#6, Check Sum False
initADAM4017(6, False);
}
```

Remarks:

None.

Init4018

Syntax:

char initADAM4018(int ID, char ChkSumEn)

Description:

Initializes the network ID and check Sum for ADAM-4018 module

Parameter	**Description**
ID	Node number on the network 1~32
ChkSumEn	Check Sum Enable True, False

Return Value:

None

Example:

```
void main ()
{
//initialize ADAM-4018 as ID#7, Check Sum False
initADAM4018(7, False);
}
```

Remarks:

None.

Init4021

Syntax:

char initADAM4021(int ID, char ChkSumEn)

Description:

Initializes the network ID and check Sum for ADAM-4021 module

Parameter	**Description**
ID	Node number on the network 1~32
ChkSumEn	Check Sum Enable True, False

Return Value:

None

Example:

```
void main ()
{
//initialize ADAM-4021 as ID#8, Check Sum False
initADAM4021(8, False);
}
```

Remarks:

None.

Init4050

Syntax:

char initADAM4050(int ID, char ChkSumEn)

Description:

Initializes the network ID and check Sum for ADAM-4050 module

Parameter	**Description**
ID	Node number on the network 1~32
ChkSumEn	Check Sum Enable True, False

Return Value:

None

Example:

```
void main ()
{
//initialize ADAM-4050 as ID#9, Check Sum False
initADAM4050(9, False);
}
```

Remarks:

None.

Init4052

Syntax:

char initADAM4052(int ID, char ChkSumEn)

Description:

Initializes the network ID and check Sum for ADAM-4052 module

Parameter	Description
ID	Node number on the network 1~32
ChkSumEn	Check Sum Enable True, False

Return Value:

None

Example:

```
void main ()
{
//initialize ADAM-4052 as ID#10, Check Sum False
initADAM4052(10, False);
}
```

Remarks:

None.

Init4053

Syntax:

char initADAM4053(int ID, char ChkSumEn)

Description:

Initializes the network ID and check Sum for ADAM-4053 module

Parameter	Description
ID	Node number on the network 1~32
ChkSumEn	Check Sum Enable True, False

Return Value:

None

Example:

```
void main ()
{
//initialize ADAM-4053 as ID#11, Check Sum False
initADAM4053(11, False);
}
```

Remarks:

None.

Init4060

Syntax:

char initADAM4060(int ID, char ChkSumEn)

Description:

Initializes the network ID and check Sum for ADAM-4060 module

Parameter	Description
ID	Node number on the network 1~32
ChkSumEn	Check Sum Enable True, False

Return Value:

None

Example:

```
void main ()
{
//initialize ADAM-4060 as ID#12, Check Sum False
initADAM4060(12, False);
}
```

Remarks:

None.

Init4080

Syntax:

char initADAM4080(int ID, char ChkSumEn)

Description:

Initializes the network ID and check Sum for ADAM-4080 module

Parameter	**Description**
ID	Node number on the network 1~32
ChkSumEn	Check Sum Enable True, False

Return Value:

None

Example:

```
void main ()
{
//initialize ADAM-4080 as ID#13, Check Sum False
initADAM4080(13, False);
}
```

Remarks:

None.

Init4080D

Syntax:

char initADAM4080D(int ID, char ChkSumEn)

Description:

Initializes the network ID and check Sum for ADAM-4080D module

Parameter	**Description**
ID	Node number on the network 1~32
ChkSumEn	Check Sum Enable True, False

Return Value:

None

Example:

```
void main ()
{
//initialize ADAM-4080D as ID#13, Check Sum False
initADAM4080D(13, False);
}
```

Remarks:

None.

Get4011

Syntax:

void Get4011(int ID, unsigned int *pValue)

Description:

Get Value from ADAM-4011 Module

Parameter	**Description**
ID	Node number on the network 1~32
*pValue	The value returned

Return Value:

None

Example:

```
void main ()
{
unsigned int *data11

//initialize ADAM-4011 as ID#2, Check Sum False
initADAM4011(2, False);

//get value from ADAM-4011
Get4011(2, data11);
printf("\nAdam-4011 = %d ", *data11);
```

```
}
```

Remarks:

None.

Get4012

Syntax:

void Get4012(int ID, unsigned int *pValue)

Description:

Get Value from ADAM-4012 Module

Parameter	**Description**
ID	Node number on the network
	1~32
*pValue	The value returned

Return Value:

None

Example:

```
void main ()
{
unsigned int *data12

//initialize ADAM-4012 as ID#3, Check Sum False
initADAM4012(3, False);

//get value from ADAM-4012
Get4012(3, data12);
printf("\nAdam-4012 = %d ", *data12);
```

```
}
```

Remarks:

None.

Get4013

Syntax:

void Get4013(int ID, unsigned int *pValue)

Description:

Get Value from ADAM-4013 Module

Parameter	**Description**
ID	Node number on the network 1~32
*pValue	The value returned

Return Value:

None

Example:

```
void main ()
{
unsigned int *data13

//initialize ADAM-4013 as ID#4, Check Sum False
initADAM4013(4, False);

//get value from ADAM-4013
Get4013(4, data13);
printf("\nAdam-4013 = %d ", *data13);
```

}

Remarks:

None.

}

Get4017

Syntax:

void Get4017(int ID, int channel, unsigned int *pValue)

Description:

Get Value from ADAM-4017 Module

Parameter	Description
ID	Node number on the network 1~32
Channel	channel number of ADAM-4017 module 0~8
*pValue	The value returned

Return Value:

None

Example:

```
void main ()
{
unsigned int *data17;

//initialize ADAM-4017 as ID#6, Check Sum False
initADAM4017(6, False);

//get value from ADAM-4017, channel 3
Get4017(6, 3, data17);
```

```
printf("\nAdam-4017 channel3= %d ", *data17);
}
```

Remarks:

None.

Get4018

Syntax:

void Get4018(int ID, int channel, unsigned int *pValue)

Description:

Get Value from ADAM-4018 Module

Parameter	Description
ID	Node number on the network 1~32
Channel	channel number of ADAM-4018 module 0~7
*pValue	The value returned

Return Value:

None

Example:

```
void main ()
{
unsigned int *data18;

//initialize ADAM-4018 as ID#7, Check Sum False
initADAM4018(7, False);

//get value from ADAM-4018, channel 3
Get4018(7, 3, data18);
```

```
printf("\nAdam-4018 channel3= %d ", *data18);
}
```

Remarks:

None.

Set4021

Syntax:

void Set4021(int ID, unsigned int *pValue)

Description:

Set Value to ADAM-4021 Module

Parameter	Description
ID	Node number on the network 1~32
*pValue	The setting value

Return Value:

None

Example:

```
void main ()
{
unsigned int *data21=1000;

//initialize ADAM-4021 as ID#8, Check Sum False
initADAM4021(8, False);

//set value to ADAM-4021
Set4021(8, data21);
}
```

Remarks:

None.

Set4050, Set4060

Syntax:

void Set4050(int ID, void *pValue, int Bit, int Size)

void Set4060(int ID, void *pValue, int Bit, int Size)

Description:

Sets the digital output for ADAM-4050, ADAM-4060 modules to the specified values.

Parameter	**Description**
ID	Node number on the network 1~32
pValue	The digital value specified by the user to be output
Bit	See "Size" parameter below
Size	ABit, AByte, AWord If Size = ABit, Bit = 0...15 (pin0 ... pin15) If Size = AByte, Bit = 0 is Low Byte data Bit = 8 is High Byte data If Size = AWord, Bit does not care, always word data

Return Value:

None.

Example:

```
void main()
{
unsigned char *Bitdata = 1;
```

```
//initialize ADAM-4050 as ID#9, Check Sum False
initADAM4050(9, False);

//Output 1 to slot 0, pin 13
Set4050(9, Bitdata, 13, ABit);
}
```

Remarks:

None.

Get4050, Get4052, Get4053

Syntax:

void Get4050(int ID, void *pValue, int Bit, int Size)

void Get4052(int ID, void *pValue, int Bit, int Size)

void Get4053(int ID, void *pValue, int Bit, int Size)

Description:

Reads the data value in an I/O module.

Parameter	Description
ID	Node number on the network 1~32
Bit	See “Size” parameter below.
Size	ABit, AByte, AWord If Size= ABit, Bit=0..15 (pin0..pin15) If Size=AByte, Bit=0 for Low Byte data; Bit=8 for High Byte data If Size=AWord, Bit does not care. Always word data.
pValue	The value returned.

Return value:

None.

Example:

void main()

{

unsigned char *Bdata;

```
//initialize ADAM-4050 as ID#9, Check Sum False
initADAM4050(9, False);

//Get value from ADAM-5050, pin13, data=0 or 1
Get5050(9, Bdata, 13, ABit);
}
```

Remarks:

None.

Get4080

Syntax:

char Get4080(int ID, int channel, unsigned long *pValue)

Description:

Get Value from ADAM-4080 Module

Parameter	Description
ID	Node number on the network 1~32
Channel	channel number of ADAM-4017 module 0~8
*pValue	The value returned

Return Value:

None

Example:

```
void main ()
{
unsigned long *Idata80;

//initialize ADAM-4080 as ID#13, Check Sum False
initADAM4080(13, False);

//get value from ADAM-4080, channel 2
Get4080(13, 2, Idata80);
```

```
printf("\nAdam-4080 channel2= %d ", *Idata80);
}
```

Remarks:

None.

Clear 4080 Counter

Syntax:

char Clear_4080_Counter(int ID, int Channel)

Description:

Reset the current counter value to its initial value

Parameter	**Description**
ID	Node number on the network 1~32
channel	The specific channel in ADAM-4080 0-3

Return Value:

None

Example:

```
void main ()
{
//initialize ADAM-4080 as ID#13, Check Sum False
initADAM4080(13, False);

//reset ADAM-4080 channel 0
Clear_4080_Counter(13, 0);
}
```

Remarks:

None.

Start/Stop 4080 Counter

Syntax:

char Start_Stop_4080_Counter(int ID, int channel, StartOrStop)

Description:

Start or stop the specific counter

Parameter	Description
ID	Node number on the network 1~32
channel	The specific channel in ADAM-5080 0-3
Start	1
Stop	0

Return Value:

None

Example:

```
void main ()
{
int Start=1, Stop=0;

//initialize ADAM-4080 as ID#13, Check Sum False
initADAM4080(13, False);

//Start channel 0 counter
```

```
ids=Start_Stop_4080_Counter(13, 0, 1);

//if the returned value is 0, print out the start fail message
if(ids==0)
printf('start failed\n");
}
```

Remarks:

None.

6.4.5 Serial I/O Library (SIO.LIB)

Port \ Slot	Slot 0	Slot 1	Slot 2	Slot 3
Port 1	1	11	21	31
Port 2	2	12	22	32
Port 3	3	13	23	33
Port 4	4	14	24	34

Table 6-6: *ADAM-5090 Port No. Definition*

Name:
Install Port

Description:
Install the communication drivers

Syntax:
int port_install(int portno)

Parameter	**Description**
portno	The specified port number

Return Value:

0	first time install and install completely!
4	not first time install but install completely!
5	portno error
6	no ADAM5090 Module in this slot

Name:
Deinstalled Port

Description:
Uninstalled the communication drivers completely

Syntax:
int port_deinstalled(int portno)

Parameter	Description
portno	The specified port number

Return Value:

0	:	deinstall success
-1	:	deinstall fail

Name:
Select Working Port

Description:
Select a specified port for work

Syntax:
void port_select(int portno)

Parameter	Description
portno	The specified port number

Return Value:
None

Name:
Reset Slot

Description:
Reset specified slot

Syntax:
int reset_slot(int slotno)

Parameter	Description
slotno	The slot you would like to reset 0~3

Return Value:
None

Example:
```
void main ()
{
    //reset all port in the slot 0
    reset_slot(0);
}
```

Name:
Reset Port

Description:
Reset specified port

Syntax:
void port_reset(int portno)

Parameter	Description
portno	The specified port number

Return Value:
None

Name:
Detect Installed Port

Description:
Detects which ports have been installed

Syntax:
int which_has_been_installed(void)

Parameter	Description
portno	The specified port number

Return Value:
Port mask which has been installed

EX.

0x2353 (0010-0011-0101-0011B)
The port01,02,11,13,21,22,32 have been installed

0x0082 (0000-0000-1000-0010B)
The port02,14 have been installed

Example:
```
void main ()
{
    int Flag;
    //here we install port1, 12, 23
    port_install(1);
    port_install(12);
    port_install(23);

    //set flat as the return value
    Flag=which_has_been_install();
    //Flag must be 0000-0100-0010-0001B
}
```

Name:
Set Port Baud Rate

Description:
Set the baud rate of specified port

Syntax:
void port_set_speed(int portno, long speed)

Parameter	Description
portno	The specified port number
long speed	4800L, 9600L, 19200L, 38400L, 115200L

Return Value:
None

Example:
```
void main ()
{
    //here we install port1, 2
    port_install(1);
    port_install(2);

    //select working port1, and set the communication rate to 38400bps
    port_select(1);
    port_speed(1, 38400L)

    //select working port2, and set the communication rate to 9600bps
    port_select(2);
    port_speed(2, 9600L)
}
```

Name:
Set Port Data Format

Description:
Set the parameters for data length, parity and stop bits for specified port

Syntax:
void port_set_format(int portno, int data_length, int parity, int stop_bit)

Parameter	Description	
portno	The specified port number	
data length	5 - 8	
parity	0x00	no parity
	0x01	odd parity
	0x02	even parity
stop bit	0x01	1 stop bit
	0x02	2 stop bits

Return Value:
None

Example:

```
void main ()
{
    port_install(1);
    port_select(1);
    port_speed(1, 9600L);

    //set data format(Data Length=8; Parity=None; Stop Bit=1)
    port_set_format(1, 8, 0, 1);
}
```

Name:

Disable Port FIFO (FIFO Size=1, for Tx and Rx)

Enable Port FIFO (FIFO Size=128, for Tx and Rx)

Description:

Set specified port to disable FIFO

Set specified port to enable FIFO

Syntax:

void port_disable_fifo(int portno)

int port_enable_fifo(int portno)

Parameter	Description
portno	The specified port number

Return Value:

Disable FIFO	: None	
Enable FIFO	: 0x00	FIFO enable success
	0x01	FIFO not available
	0x04	portno error

Example:

```
void main ()
{
    port_install(1);
    :
    :
    port_set_format(1, 8, 0, 1)

    //enable port1 FIFO to 128 byte
    port_enable_fifo(1);
}
```

Name:

Detect Port Carrier

Description:

Detect the carrier signal of specified port

Syntax:

int port_carrier(int portno)

Parameter	**Description**
portno	The specified port number

Return Value:

0	:	no carrier been detected or bad command or parameter
1	:	detect carrier

Example:

```
void main ()
{
    port_install(1);
    :
    :
    port_enable_fifo(1);

    //if port1 detected carrier, print out the message
    if(port_carrier(1));
    {
     printf("\n port1 detect carrier");
    {
}
```

Name:

Clear Port Break

Set Port Break

Description:

Set specified port to clear BREAK signal

Set specified port to send BREAK signal

Syntax:

void port_clear_break(int portno)

void port_set_break(int portno)

Parameter	Description
portno	The specified port number

Return Value:

None

Example:

```
void main ()
{
    port_install(1);
    :
    :
    port_enable_fifo(1);

    //set port1 to clear break signal
    port_clear_break(1);
    //or "port_set_break(1)"
}
```

Name:
Clear Local Loopback

Set Local Loopback

Description:
Set specified port to disable loopback function for diagnostic

Set specified port to enable loopback function for diagnostic

Syntax:
void port_clear_local_loopback(int portno)

void port_set_local_loopback(int portno)

Parameter	Description
portno	The specified port number

Return Value:
None

Example:
```
void main ()
{
    port_install(1);
    :
    :
    port_enable_fifo(1);

    //set port1 to enable loopback function for diagnostic
    port_set_local_loopback(1);
    //or "port_clear_local_loopback(1)"
}
```

Name:
Read LSR
Set LCR

Description:
Read from specified port line status register (LSR)
Write to specific port line control register (LCR)

Syntax:
int port_get_line_status(int portno)
int port_set_line_params(int portno, int lineparams)

Parameter	Description
portno	The specified port number
lineparams	Line control register parameter (see UART Register Description Table)

Return Value:

port_get_line_status		:	
	0x00XX	:	LSR value
	0xFF00	:	bad command or parameter
port_set_line_params		:	
	0x00	:	write success
	0x01	:	LCR read back error
	0xFE00	:	LCR write not able
	0xFF00	:	bad command or parameter

Example:

```
void main ()
{
   int LSR_Value, LCR_Params;
   port_install(1);
   :
   :
   port_enable_fifo(1);
   //get LSR value
   LSR_Value=port_get_line_status(1);

   //set LCR value=0x03
   LCR_Params=0x03;
   port_set_line_status(1, LCR_Params);
}
```

Register Name	Description	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
LSR	Line Status Register	Data Error	Tx Empty	THR Empty	Rx Break	Framing Error	Parity Error	Overrun Error	RxRDY
LCR	Line Control Register	divisor latch access	Tx Break	Force parity	odd/even parity	Parity enable	Number of stop bit	data length bits[1:0]	

UART Register Description Table

Name:
Read Modem Status (MSR)

Description:
Read from specified port modem status register

Syntax:
int port_get_modem_status(int portno)

Parameter	Description
portno	The specified port number

Return Value:

0x00XX	:	modem status
0xFF00	:	bad command or parameter

Example:
```
void main ()
{
    int MSR_Value;
    port_install(1);
    :
    :
    port_enable_fifo(1);

    //get MSR value
    MSR_Value=port_get_modem_status(1);
}
```

Register Name	Description	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
MSR	Modem Status Register	DCD	RI	DSR	CTS	Delta DCD	Trailing RI edge	Delta DSR	Delta CTS

UART Register Description Table

Name:
Read Modem Control Register (MCR)
Set Modem Control Register (MCR)

Description:
Read from specified port modem control register
Set from specified port modem control register

Syntax:
int port_get_modem_control_status(int portno)
int port_set_modem_control_params(int portno, int MCRparams)

Parameter	**Description**
portno	The specified port number
MCRparams	Modem control register parameter (see UART Register Description Table)

Return Value:
Read MCR:

0x00XX	:	modem status
0xFF00	:	bad command or parameter

Write MCR:

0x0000	:	write MCR success
0x0001	:	read back error
0xFF00	:	bad command or parameter

Example:

```
void main ()
{
    int MCR_Value, MCR_Params;
    port_install(1);
    :
    :
    port_enable_fifo(1);

    //set MCR value=3 (RTS=1; DTR=1)
    MCR_Params=3
    port_set_modem_control_params(1, MCR_Params);

    //get MCR value
    MCR_Value=port_get_modem_control_status(1);
    // MCR value must be 3
}
```

Register Name	Description	Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
LSR	Line Status Register	Data Error	Tx Empty	THR Empty	Rx Break	Framing Error	Parity Error	Overrun Error	RxRDY
LCR	Line Control Register	divisor latch access	Tx Break	Force parity	odd/even parity	Parity enable	Number of stop bit	data length bits[1:0]	

UART Register Description Table

Name:
Set DTR Low
Set DTR High

Description:
Set specified port DTR low
Set specified port DTR high

Syntax:
void port_lower_dtr(int portno)
void port_raise_dtr(int portno)

Parameter	Description
portno	The specified port number

Return Value:
None

Example:
```
void main ()
{
    port_install(1);
    :
    :
    //set port1 DTR low
    port_lower_dtr(1);

    //set port1 DTR high
    port_raise_dtr(1);
}
```

Name:
Set RTS High
Set RTS Low

Description:
Set specified port RTS high
Set specified port RTS low

Syntax:
void port_raise_rts(int portno)
void port_lower_rts(int portno)

Parameter	Description
portno	The specified port number

Return Value:
None

Example:
```
void main ()
{
    port_install(1);
    :
    :
    //set port1 RTS low
    port_lower_rts(1);

    //set port1 RTS high
    port_raise_rts(1);
}
```

Name:
Modem Initial

Description:
Set modem to initial status

Syntax:
modem_initial_90(int portno)

parameter	Description
portno	The specified port number

Return Value:
None

Name:
Send Modem AT Command

Description:
Send AT command string to the modem

Syntax:
modem_command_90(int portno, char *cmdstr)

parameter	Description
portno	The specified port number
*cmdstr	AT command string

Return Value:
None

Name:
Set Modem Command Mode

Description:
Set modem to command mode

Syntax:
void modem_command_state_90(int portno)

parameter	Description
portno	The specified port number

Return Value:
None

Name:
Set Modem Autoanswer

Description:
Set up modem to auto answer phone calls

Syntax:
void modem_autoanswer_90(int portno)

parameter	Description
portno	The specified port number

Return Value:
None

Name:
Modem Dial Out

Description:
Direct modem to dial the specified telephone number

Syntax:
void modem_dial_90(int portno, char *telnumber)

parameter	Description
portno	The specified port number
*telnumber	The telephone number you would like to dial out

Return Value:
None

Example:
```
void main ()
{
    port_install(1);
    :
    :
    //initial modem for port1
    modem_initial_90(1);

    //set the dial out number as "1234-5678"
    modem_dial_90(1, "12345678");
}
```

Name:
Han up Modem

Description:
Set modem to hand up the telephone

Syntax:
void modem_handup_90(int portno)

parameter	Description
portno	The specified port number

Return Value:
None

Name:
Rx Flush
Tx Flush

Description:
Flush Rx or Tx FIFO

Syntax:
void port_flush_rx(int portno)
void port_flush_tx(int portno)

parameter	Description
portno	The specified port number

Return Value:
None

Name:
Receive Error Check

Description:
Check whether receive error or not

Syntax:
int port_rx_error(int portno)

Parameter	Description
portno	The specified port number

Return Value:

0 : no error

0x00XX : receive error and return LSR value

Example:

```
void main ()
{
    int Err_Value;
    port_install(1);
    :
    :
    //get error check value; if error, print out the message
    Err_Value=port_rx_error(1);
    If(Err_Value)
    {
    printf("\n Rx Error, The LSR value=%X", Err_Value);
    }
}
```

Name:
Ready Check

Description:
Check received data in port FIFO already

Syntax:
int port_rx_ready(int portno)

Parameter	Description
portno	The specified port number

Return Value:

0 :data not ready
1 :data ready

Name:
Receive Character

Description:
Receive a character from specific port

Syntax:
char port_rx(int portno)

Parameter	Description
portno	The specified port number

Return Value:
Character

Example:
```
void main ()
{
    char C;
    port_install(1);
    :
    :
    //if port1 FIFO receive data, read a character and print it out
    If(port_rx_ready(1));
    {
    C=port_rx(1);
    printf("\n %C", C);
    }
}
```

Name:
Empty Check

Description:
Return the status of the specified port transmit queues

Syntax:
int port_tx_empty(int portno)

Parameter	Description
portno	The specified port number

Return Value:

0 : not empty

1 : FIFO empty

2 : FIFO and Transmitting empty

Name:
Send Character

Description:
Send a character to the THR of the specified port

Syntax:
void port_tx(int portno, char c)

Parameter	Description
portno	The specified port number
c	The character you would like to send

Return Value:
None

```
main()
{
   char character
   port_installed(1)
   :
   :
   //check whether FIFO empty or not, if empty, send a character
   if(port_tx_empty(1);
   {
   character='a'
   port_tx(1, character)
   {
}
```

Name:
Send String

Description:
Sends a string by calling port_tx() repeatedly

Syntax:
void port_tx_string(int portno, char *s)

Parameter	Description
portno	The specified port number
*s	the string you would like to send

Return Value:
None

```
main()
  {
  char string
  port_installed(1)
  :
  :
  //check whether FIFO empty or not, if empty, send a string
  if(port_tx_empty(1);
  {
  string=”abcde”
  port_tx_string(1, string)
  {
}
```

Appendix A

COM Port Register Structure

Register Structure

This appendix gives a short description of each module's registers. For more information, please refer to the STARTECH 16C550 UART chip data book.

All registers are one byte. Bit 0 is the least significant bit, and bit 7 is the most significant bit. The address of each register is specified as an offset from the port base address (BASE), COM1 is 3F8h and COM2 is 2F8h.

DLAB is the "Divisor Latch Access Bit", bit 7 of BASE+3.

BASE+0 Receiver buffer register when DLAB=0 and the operation is a read.
BASE+0 Transmitter holding register when DLAB=0 and the operation is write.
BASE+0 Divisor latch bits 0 - 7 when DLAB=1
BASE+1 Divisor latch bits 8-15 when DLAB=1.
Bytes BASE+0 and BASE+1 together form a 16-bit number, the divisor, which determines the baud rate. Set the divisor as follows:

Baud rate	Divisor	Baud rate	Divisor
50	2304	2400	48
75	1536	3600	32
110	1047	4800	24
133.5	857	7200	16
150	768	9600	12
300	384	19200	6
600	192	38400	3
1200	96	56000	2
1800	64	115200	1
2000	58	x	x

BASE+1 Interrupt Status Register (ISR) when DLAB=0
bit 0: Enable received-data-available interrupt
bit 1: Enable transmitter-holding-register-empty interrupt
bit 2: Enable receiver-line-status interrupt
bit 3: Enable modem-status interrupt

BASE+2 FIFO Control Register (FCR)
bit 0: Enable transmit and receive FIFOs
bit 1: Clear contents of receive FIFO
bit 2: Clear contents of transmit FIFO
bits 6-7: Set trigger level for receiver FIFO interrupt

Bit 7	Bit 6	FIFO trigger level
0	0	01
0	1	04
1	0	08
1	1	14

BASE+3 Line Control Register (LCR)
bit 0: Word length select bit 0
bit 1: Word length select bit 1

Bit 1	Bit 0	Word length (bits)
0	0	5
0	1	6
1	0	7
1	1	8

bit 2: Number of stop bits
bit 3: Parity enable
bit 4: Even parity select
bit 5: Stick parity
bit 6: Set break
bit 7: Divisor Latch Access Bit (DLAB)

Register Structure

BASE+4 Modem Control Register (MCR)
bit 0: DTR
bit 1: RTS

BASE+5 Line Status Register (LSR)
bit 0: Receiver data ready
bit 1: Overrun error
bit 2: Parity error
bit 3: Framing error
bit 4: Break interrupt
bit 5: Transmitter holding register empty
bit 6: Transmitter shift register empty
bit 7: At least one parity error, framing error or break indication in the FIFO

BASE+6 Modem Status Register (MSR)
bit 0: Delta CTS
bit 1: Delta DSR
bit 2: Trailing edge ring indicator
bit 3: Delta received line signal detect
bit 4: CTS
bit 5: DSR
bit 6: RI
bit 7: Received line signal detect

BASE+7 Temporary data register

Appendix B

Data Formats and I/O Ranges

Data Formats and I/O Ranges

B.1 Analog Input Formats

The ADAM analog input modules can be configured to transmit data to the host in Engineering Units.

Engineering Units

Data can be represented in Engineering Units by setting bits 0 and 1 of the data format/checksum/integration time parameter to 0.

This format presents data in natural units, such as degrees, volts, millivolts, and milliamps. The Engineering Units format is readily parsed by the majority of computer languages because the total data string length, including sign, digits and decimal point, does not exceed seven characters.

The data format is a plus (+) or minus (-) sign, followed by five decimal digits and a decimal point. The input range which is employed determines the resolution, or the number of decimal places used, as illustrated in the following table:

Input Range	Resolution
±15 mV, ±50 mV	1 μV (three decimal places)
±100 mV, ±150 mV, ±500 mV	10 μV (two decimal places)
±1 V, ±2.5 V, ±5 V	100 μV (four decimal places)
±10 V	1 mV (three decimal places)
±20 mA	1 μA (three decimal places)
Type J and T thermocouple	0.01ºC (two decimal places)
Type K, E, R, S, and B thermocouple	0.1ºC (one decimal place)

Example 1

The input value is -2.65 V and the corresponding analog input module is configured for a range of ±5 V. The response to the Analog Data In command is:

```
-2.6500(cr)
```

Example 2

The input value is 305.5ºC. The analog input module is configured for a Type J thermocouple whose range is 0ºC to 760ºC. The response to the Analog Data In command is:

```
+305.50(cr)
```

Example 3

The input value is +5.653 V. The analog input module is configured for a range of ±5 V range. When the engineering units format is used, the ADAM Series analog input modules are configured so that they automatically provide an over range capability. The response to the Analog Data In command in this case is:

```
+5.6530(cr)
```

B.2 Analog Input Ranges - ADAM-5017

Module	Range Code	Input Range Description	Data Formats	+F.S.	Zero	-F.S.	Displayed Resolution	Actual Value
ADAM-5017	08h	±10 V	Engineering Units	+10.000	±00.000	-10.000	1 mV	Reading/ 1000
			% of FSR	+100.00	±000.00	-100.00	0.01%	
			Two's Complement	7FFF	0000	8000	1 LSB	
	09h	±5 V	Engineering Units	+5.0000	±0.0000	-5.0000	100.00 µV	Reading/ 1000
			% of FSR	+100.00	±000.00	-100.00	0.01%	
			Two's Complement	7FFF	0000	8000	1 LSB	
	0Ah	±1 V	Engineering Units	+1.0000	±0.0000	-1.0000	100.00 µV	Reading/ 10000
			% of FSR	+100.00	±000.00	-100.00	0.01%	
			Two's Complement	7FFF	0000	8000	1 LSB	
	0Bh	±500 mV	Engineering Units	+500.00	±000.00	-500.00	10 µV	Reading/ 10
			% of FSR	+100.00	±000.00	-100.00	0.01%	
			Two's Complement	7FFF	0000	8000	1 LSB	
	0Ch	±150 mV	Engineering Units	+150.00	±000.00	-150.00	10 µV	Reading/ 100
			% of FSR	+100.00	±000.00	-100.00	0.01%	
			Two's Complement	7FFF	0000	8000	1 LSB	
	0Dh	±20 mA	Engineering Units	+20.000	±00.000	-20.000	1 µV	Reading/ 1000
			% of FSR	+100.00	±000.00	-100.00	0.01%	
			Two's Complement	7FFF	0000	8000	1 LSB	

B.3 Analog Input Ranges - ADAM-5018

Module	Range Code	Input Range Description	Data Formats	+F.S.	Zero	-F.S.	Displayed Resolution	Actual Value
ADAM-5018	00h	±15 mV	Engineering Units	+15.000	±00.000	-15.000	1 µV	Reading/ 1000
			% of FSR	+100.00	±000.00	-100.00	0.01%	
			Two's Complement	7FFF	0000	8000	1 LSB	
	01h	±50 mV	Engineering Units	+50.000	±00.000	-50.000	1 µV	Reading/ 100
			% of FSR	+100.00	±000.00	-100.00	0.01%	
			Two's Complement	7FFF	0000	8000	1 LSB	
	02h	±100 mV	Engineering Units	+100.00	±000.00	-100.00	10 µV	Reading/ 100
			% of FSR	+100.00	±000.00	-100.00	0.01%	
			Two's Complement	7FFF	0000	8000	1 LSB	
	03h	±500 mV	Engineering Units	+500.00	±000.00	-500.00	10 µV	Reading/ 10
			% of FSR	+100.00	±000.00	-100.00	0.01%	
			Two's Complement	7FFF	0000	8000	1 LSB	
	04h	±1 V	Engineering Units	+1.0000	±0.0000	-1.0000	100 µV	Reading/ 10000
			% of FSR	+100.00	±000.00	-100.00	0.01%	
			Two's Complement	7FFF	0000	8000	1 LSB	
	05h	±2.5 V	Engineering Units	+2.5000	±0.0000	-2.5000	100 µV	Reading/ 10000
			% of FSR	+100.00	±000.00	-100.00	0.01%	
			Two's Complement	7FFF	0000	8000	1 LSB	
	06h	±20 mA	Engineering Units	+20.000	±00.000	-20.000	1 µA	Reading/ 1000
			% of FSR	+100.00	±000.00	-100.00	0.01%	
			Two's Complement	7FFF	0000	8000	1 LSB	
	07h	Not Used						

Module	Range Code	Input Range Description	Data Formats	Maximum Specified Signal	Minimum Specified Signal	Displayed Resolution	Actual Value
ADAM-5018	0Eh	Type J Thermocouple 0° C to 760° C	Engineering Units	+760.00	+000.00	0.1° C	Reading/ 10
			% of FSR	+100.00	+000.00	0.01%	
			Two's Complement	7FFF	0000	1 LSB	
	0Fh	Type K Thermocouple 0° C to 1370° C	Engineering Units	+1370.0	+0000.0	0.1° C	Reading/ 10
			% of FSR	+100.00	+000.00	0.01%	
			Two's Complement	7FFF	0000	1 LSB	
	10h	Type T Thermocouple -100° C to 400° C	Engineering Units	+400.00	-100.00	0.1° C	Reading/ 10
			% of FSR	+100.00	-025.00	0.01%	
			Two's Complement	7FFF	E000	1 LSB	
	11h	Type E Thermocouple 0° C to 1000° C	Engineering Units	+1000.00	+0000.0	0.1° C	Reading/ 10
			% of FSR	+100.00	±000.00	0.01%	
			Two's Complement	7FFF	0000	1 LSB	
	12h	Type R Thermocouple 500° C to 1750° C	Engineering Units	+1750.0	+0500.0	0.1° C	Reading/ 10
			% of FSR	+100.00	+028.57	0.01%	
			Two's Complement	7FFF	2492	1 LSB	
	13h	Type S Thermocouple 500° C to 1750° C	Engineering Units	+1750.0	+0500.00	0.1° C	Reading/ 10
			% of FSR	+100.00	+028.57	0.01%	
			Two's Complement	7FFF	2492	1 LSB	
	14h	Type B Thermocouple 500° C to 1800° C	Engineering Units	+1800.0	+0500.0	0.1° C	Reading/ 10
			% of FSR	+100.00	+027.77	0.01%	
			Two's Complement	7FFF	2381	1 LSB	

B.4 Analog Input Ranges - ADAM-5017H

Range Code	Input Range	Data Formats	+Full Scale	Zero	-Full Scale	Displayed Resolution
00h	±10 V	Engineering	11	0	-11	2.7 mV
		Two's Comp	0FFF	0	EFFF	1
01h	0 ~ 10 V	Engineering	11	0	Don't care	2.7 mV
		Two's Comp	0FFF	0	Don't care	1
02h	±5 V	Engineering	5.5	0	-5.5	1.3 mV
		Two's Comp	0FFF	0	EFFF	1
03h	0 ~ 5 V	Engineering	5.5	0	Don't care	1.3 mV
		Two's Comp	0FFF	0	Don't care	1
04h	±2.5 V	Engineering	2.75	0	-2.75	0.67 mV
		Two's Comp	0FFF	0	EFFF	1
05h	0 ~ 2.5 V	Engineering	2.75	0	Don't care	0.67 mV
		Two's Comp	0FFF	0	Don't care	1
06h	±1 V	Engineering	1.375	0	-1.375	0.34 mV
		Two's Comp	0FFF	0	EFFF	1
07h	0 ~ 1 V	Engineering	1.375	0	Don't care	0.34 mV
		Two's Comp	0FFF	0	Don't care	1
08h	±500 mV	Engineering	687.5	0	-687.5	0.16 mV
		Two's Comp	0FFF	0	EFFF	1
09h	0 ~ 500 mV	Engineering	687.5	0	Don't care	0.16 mV
		Two's Comp	0FFF	0	Don't care	1
0ah	4 ~ 20 mA	Engineering	22	4.0	Don't care	5.3 µA
		Two's Comp	0FFF	02E9	Don't care	1
0bh	0 ~ 20 mA	Engineering	22	0	Don't care	5.3 µA
		Two's Comp	0FFF	0	Don't care	1

Note: *The full scale values in this table are theoretical values for your reference; actual values will vary.*

B.5 Analog Output Formats

You can configure ADAM analog output modules to receive data from the host in Engineering Units.

Engineering Units

Data can be represented in engineering units by setting bits 0 and 1 of the data format/checksum/integration time parameter to 0.

This format presents data in natural units, such as milliamps. The Engineering Units format is readily parsed by the majority of computer languages as the total data string length is fixed at six characters: two decimal digits, a decimal point and three decimal digits. The resolution is 5 µA.

Example:

An analog output module on channel 1 of slot 0 in an ADAM-5000 system at address 01h is configured for a 0 to 20 mA range. If the output value is +4.762 mA, the format of the Analog Data Out command would be #01S0C14.762<cr>

B.6 Analog Output Ranges

Range Code	Output Range Description	Data Formats	Maximum Specified Signal	Minimum Specified Signal	Displayed Resolution
30	0 to 20 mA	Engineering Units	20.000	00.000	5 µA
		% of Span	+100.00	+000.00	5 µA
		Hexadecimal Binary	FFF	000	5 µA
31	4 to 20 mA	Engineering Units	20.000	04.000	5 µA
		% of Span	+100.00	+000.00	5 µA
		Hexadecimal Binary	FFF	000	5 µA
32	0 to 10 V	Engineering Units	10.000	00.000	2.442 mV
		% of Span	+100.00	+000.00	2.442 mV
		Hexadecimal Binary	FFF	000	2.442 mV

B.7 ADAM-5013 RTD Input Format and Ranges

Range Code (hex)	Input Range Description	Data Formats	Maximum Specified Signal	Minimum Specified Signal	Displayed Resolution
20	100 Ohms Platinum RTD -100 to 100° C a=0.00385	Engineering Units	+100.00	-100.00	±0.1° C
21	100 Ohms Platinum RTD 0 to 100° C a=0.00385	Engineering Units	+100.00	+000.00	±0.1° C
22	100 Ohms Platinum RTD 0 to 200° C a=0.00385	Engineering Units	+200.00	+000.00	±0.2° C
23	100 Ohms Platinum RTD 0 to 600° C a=0.00385	Engineering Units	+600.00	+000.00	±0.6° C
24	100 Ohms Platinum RTD -100 to 100° C a=0.00392	Engineering Units	+100.00	-100.00	±0.1° C
25	100 Ohms Platinum RTD 0 to 100° C a=0.00392	Engineering Units	+100.00	+000.00	±0.1° C
26	100 Ohms Platinum RTD 0 to 200° C a=0.00392	Engineering Units	+200.00	+000.00	±0.2° C

Note: *See next page for table continuation.*

Note: This table continued from previous page.

27	100 Ohms Platinum RTD 0 to 600° C a=0.00392	Engineering Units	+600.00	+000.00	±0.6° C
28	120 Ohms Nickel RTD -80 to 100° C	Engineering Units	+100.00	-80.00	±0.1° C
29	120 Ohms Nickel RTD 0 to 100° C	Engineering Units	+100.00	+000.00	±0.1° C

ADAM 5000 AI/AO Scaling

Module	Type	Range Low	Range High	Scale Low	Scale High	Data Format
5013RTD	385(IEC)	-100	100	0	65535	U16B
		0	100	0	65535	U16B
		0	200	0	65535	U16B
		0	600	0	65535	U16B
	395(JIS)	-100	100	0	65535	U16B
		0	100	0	65535	U16B
		0	200	0	65535	U16B
		0	600	0	65535	U16B
	Ni	-80	100	0	65535	U16B
		0	100	0	65535	U16B
5017AI	mV	-150	150	0	65535	U16B
	mV	-500	500	0	65535	U16B
	V	-1	1	0	65535	U16B
	V	-5	5	0	65535	U16B
	V	-10	10	0	65535	U16B
	mA	-20	20	0	65535	U16B
5017H AI	mV	-500	500	0	4095	U12B
	mV	0	500	0	4095	U12B
	V	-10	10	0	4095	U12B
	V	0	10	0	4095	U12B
	V	-5	5	0	4095	U12B
	V	0	5	0	4095	U12B
	V	-2.5	2.5	0	4095	U12B
	V	0	2.5	0	4095	U12B
	V	-1	1	0	4095	U12B
	V	0	1	0	4095	U12B
	mA	4	20	0	4095	U12B
	mA	0	20	0	4095	U12B
5018 AI	mV	-15	15	0	65535	U16B
	mV	-50	50	0	65535	U16B
	mV	-100	100	0	65535	U16B
	mV	-500	500	0	65535	U16B
	V	-1	1	0	65535	U16B
	V	-2.5	2.5	0	65535	U16B
	mA	-20	20	0	65535	U16B
	T/C(J)	0	760	0	65535	U16B
	T/C(K)	0	1370	0	65535	U16B
	T/C(T)	-100	400	0	65535	U16B
	T/C(E)	0	1000	0	65535	U16B
	T/C(R)	500	1750	0	65535	U16B
	T/C(S)	500	1750	0	65535	U16B
	T/C(B)	500	1800	0	65535	U16B
5024 AO	V	0	10	0	4095	U12B
	mA	4	20	0	4095	U12B
	mA	0	20	0	4095	U12B

This page is blank

Appendix C

Examples on CD

Examples on CD

Three examples are included on the ADAM-5511 CD. After you install the utility CD on your host PC, these examples will be located in the directory C:\ADAM5511\Example. The following list describes these examples.

Example 1 (Ex1.prj)

This example scans all slots in an ADAM-5511 and then shows the status of any I/O modules(include AI/O, DI/O, Counter, and Series Communication Module) located in the slots.

Example 2 (Ex2.prj)

This is a modem test example which includes dial, hang-up, auto-answer and set break.

Example 3 (Ex3.prj)

Using ADAM-5511 COM port and ADAM-4520 (RS-232 to RS-422/485 converter) to scan ADAM-4000 series module as remote I/O function.

Appendix D

RS-485 Network

RS-485 Network

EIA RS-485 is the industry's most widely used bidirectional, balanced transmission line standard. It is specifically developed for industrial multi-drop systems that should be able to transmit and receive data at high rates or over long distances.

The specifications of the EIA RS-485 protocol are as follows:

- Maximum line length per segment: 1200 meters (4000 feet)
- Throughput of 10 Mbaud and beyond -Differential transmission (balanced lines) with high resistance against noise
- Maximum 32 nodes per segment
- Bi-directional master-slave communication over a single set of twisted-pair cables
- Parallel connected nodes, true multi-drop

ADAM-5510/P31 systems are fully isolated and use just a single set of twisted pair wires to send and receive! Since the nodes are connected in parallel they can be freely disconnected from the host without affecting the functioning of the remaining nodes. An industry standard, shielded twisted pair is preferable due to the high noise ratio of the environment.

When nodes communicate through the network, no sending conflicts can occur since a simple command/response sequence is used. There is always one initiator (with no address) and many slaves (with addresses). In this case, the master is a personal computer that is connected with its serial, RS-232, port to an ADAM RS-232/RS-485 converter. The slaves are the ADAM-5510/P31 systems. When systems are not transmitting data, they are in listen mode. The host computer initiates a command/response sequence with one of the systems. Commands normally contain the address of the module the host wants to communicate with. The system with the matching address carries out the command and sends its response to the host.

D.1 Basic Network Layout

Multi-drop RS-485 implies that there are two main wires in a segment. The connected systems tap from these two lines with so called drop cables. Thus all connections are parallel and connecting or disconnecting of a node doesn't affect the network as a whole. Since ADAM-5510/P31 systems use the RS-485 standard, they can connect and communicate with the host PC. The basic layouts that can be used for an RS-485 network are:

Daisychain

The last module of a segment is a repeater. It is directly connected to the main-wires thereby ending the first segment and starting the next segment. Up to 32 addressable systems can be daisychained . This limitation is a physical one. When using more systems per segment the IC driver current rapidly decreases, causing communication errors. In total, the network can hold up to 64 addressable systems. The limitation on this number is the two-character hexadecimal address code that can address 64 combinations. The ADAM converter, ADAM repeaters and the host computer are non addressable units and therefore are not included in these numbers.

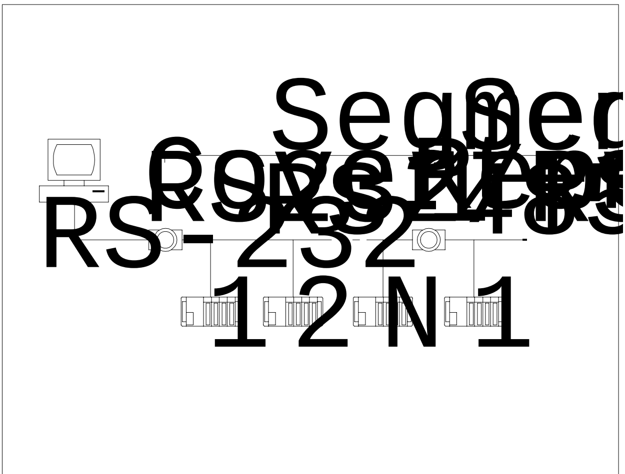


Figure D-1: *Daisychaining*

RS-485 Network

Star Layout

In this scheme the repeaters are connected to drop-down cables from the main wires of the first segment. A tree structure is the result. This scheme is not recommended when using long lines since it will cause a serious amount of signal distortion due to signal reflections in several line-endings.

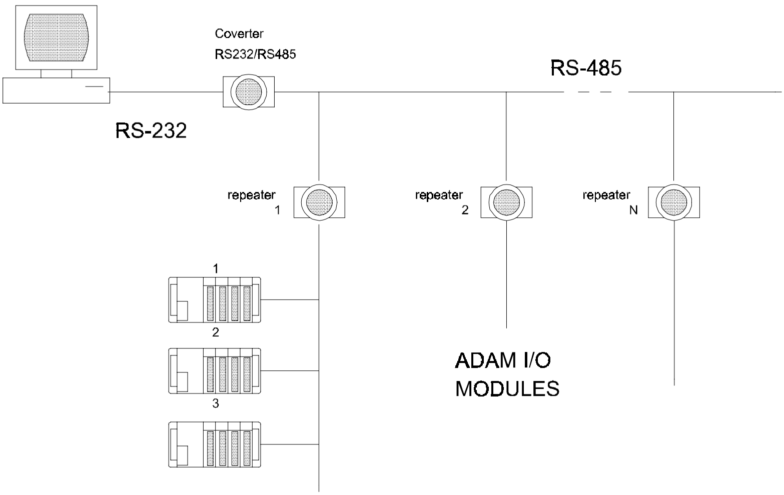


Figure D-2: *Star structure*

Random

This is a combination of daisychain and hierarchical structure.

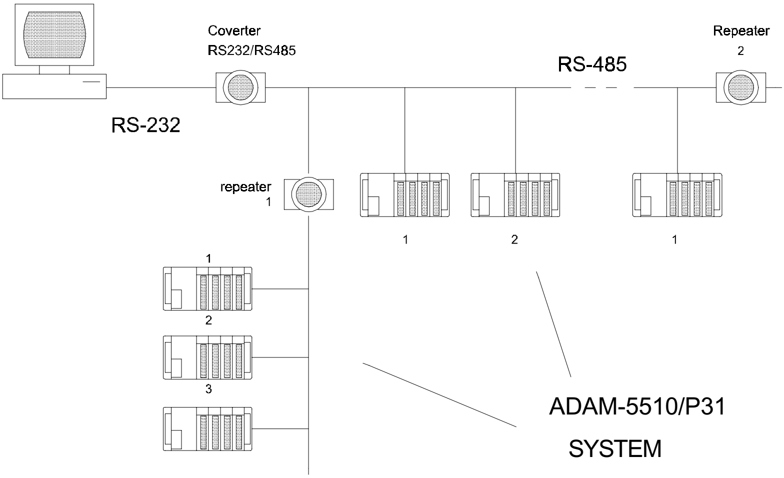


Figure D-3: *Random structure*

D.2 Line Termination

Each discontinuity in impedance causes reflections and distortion. When a impedance discontinuity occurs in the transmission line the immediate effect is signal reflection. This will lead to signal distortion. Specially at line ends this mismatch causes problems. To eliminate this discontinuity, terminate the line with a resistor.

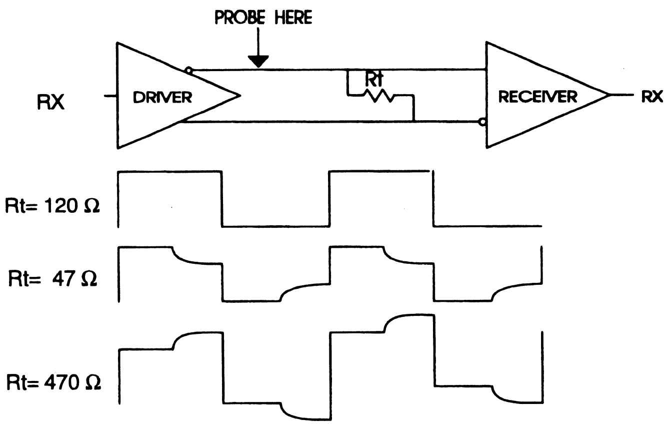


Figure D-4: *Signal distortion*

The value of the resistor should be a close as possible to the characteristic impedance of the line. Although receiver devices add some resistance to the whole of the transmission line, normally it is sufficient to the resistor impedance should equal the characteristic impedance of the line.

Example:

Each input of the receivers has a nominal input impedance of 18 kΩ feeding into a diode transistor- resistor biasing network that is equivalent to an 18 kΩ input resistor tied to a common mode voltage of 2.4 V. It is this configuration which provides the large common range of the receiver required for RS-485 systems! (See Figure E-5 below).

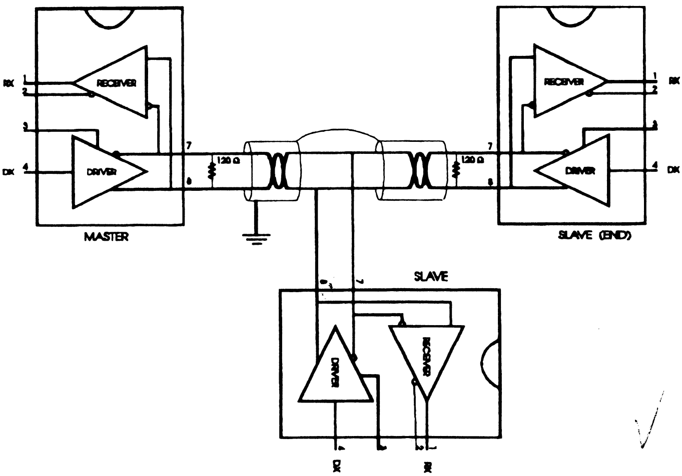


Figure D-5: *Termination resistor locations*

Because each input is biased to 2.4 V, the nominal common mode voltage of balanced RS-485 systems, the 18 kΩ on the input can be taken as being in series across the input of each individual receiver.

If thirty of these receivers are put closely together at the end of the transmission line, they will tend to react as thirty 36kΩ resistors in parallel with the termination resistor. The overall effective resistance will need to be close to the characteristics of the line. The effective parallel receiver resistance R_P will therefore be equal to:

$R_P = 36 \times 10^3/30 = 1200\,\Omega$

While the termination receptor R_T will equal:

$R_T = R_O / [1 - R_O/R_P]$

Thus for a line with a characteristic impedance of 100 Ω resistor

$R_T = 100/[1 - 100/1200] = 110\,\Omega$

Since this value lies within 10% of the line characteristic impedance.

Thus as already stated above the line termination resistor R_T will normally equal the characteristic impedance Z_o.

The star connection causes a multitude of these discontinuities since there are several transmission lines and is therefore not recommend.

Note: *The recommend method wiring method, that causes a minimum amount of reflection, is daisy chaining where all receivers tapped from one transmission line needs only to be terminated twice.*

D.3 RS-485 Data Flow Control

The RS-485 standard uses a single pair of wires to send and receive data. This line sharing requires some method to control the direction of the data flow. RTS (Request To Send) and CTS (Clear To Send) are the most commonly used methods.

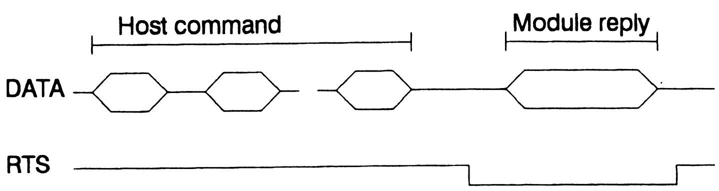


Figure D-6: *RS-485 data flow control with RTS*

Intelligent RS-485 Control

ADAM-4510 and ADAM-4520 are both equipped with an I/O circuit which can automatically sense the direction of the data flow. No handshaking with the host (like RTS, Request to Send) is necessary to receive data and forward it in the correct direction. You can use any software written for half-duplex RS-232 with an ADAM network without modification. The RS-485 control is completely transparent to the user.

Appendix E

Grounding Reference

Grounding Reference

Field Grounding and Shielding Application

Overview

Unfortunately, it's impossible to finish a system integration task at one time. We always meet some trouble in the field. A communication network or system isn't stable, induced noise or equipment is damaged or there are storms. However, the most usual issue is just simply improper wiring, ie, grounding and shielding. You know the 80/20 rule in our life: we spend 20% time for 80% work, but 80% time for the last 20% of the work. So is it with system integration: we pay 20% for Wire / Cable and 0% for Equipment. However, 80% of reliability depends on Grounding and Shielding. In other words, we need to invest more in that 20% and work on these two issues to make a highly reliable system.

This application note brings you some concepts about field grounding and shielding. These topics will be illustrated in the following pages.

1. Grounding
   - 1.1 The 'Earth' for reference
   - 1.2 The 'Frame Ground' and 'Grounding Bar'
   - 1.3 Normal Mode and Common Mode
   - 1.4 Wire impedance
   - 1.5 Single Point Grounding
2. Shielding
   - 2.1 Cable Shield
   - 2.2 System Shielding
3. Noise Reduction Techniques
4. Check Point List

Appendix E

E.1 Grounding

1.1 The ‘Earth’ for reference

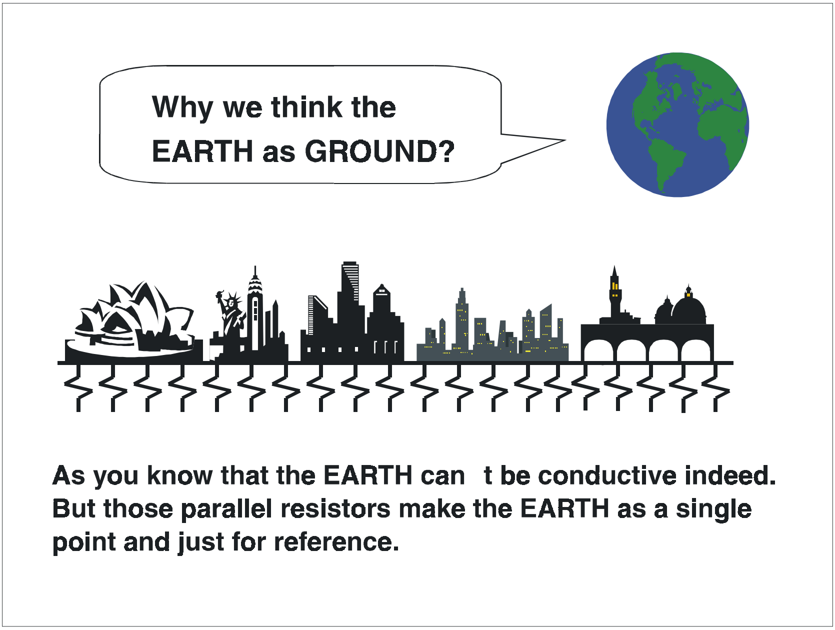
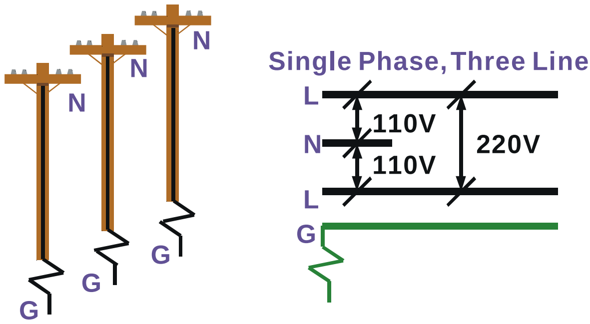


Figure E-1: Think the EARTH as GROUND.

As you know, the EARTH cannot be conductive. However, all buildings lie on, or in, the EARTH. Steel, concrete and associated cables (such as lighting arresters) and power system were connected to EARTH. Think of them as resistors. All of those infinite parallel resistors make the EARTH as a single reference point.

1.2 The 'Frame Ground' and 'Grounding Bar'



Neutral is the physical cable from Generator.
Ground is the local physical cable that connected to Ground Bar.

Figure E-2: Grounding Bar.

Grounding is one of the most important issues for our system. Just like Frame Ground of the computer, this signal offers a reference point of the electronic circuit inside the computer. If we want to communicate with this computer, both Signal Ground and Frame Ground should be connected to make a reference point of each other's electronic circuit. Generally speaking, it is necessary to install an individual grounding bar for each system, such as computer networks, power systems, telecommunication networks, etc. Those individual grounding bars not only provide the individual reference point, but also make the earth a our ground!

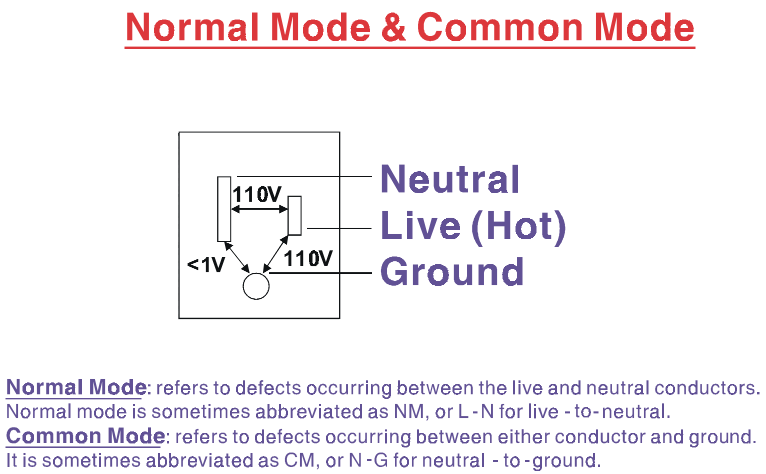


Figure E-3: Normal mode and Common mode.

1.3 Normal Mode and Common Mode

Have you ever tried to measure the voltage between a live circuit and a concrete floor? How about the voltage between neutral and a concrete floor? You will get nonsense values. ‘Hot’ and ‘Neutral’ are just relational signals: you will get 110VAC or 220VAC by measuring these signals. Normal mode and common mode just show you that the Frame Ground is the most important reference signal for all the systems and equipments.

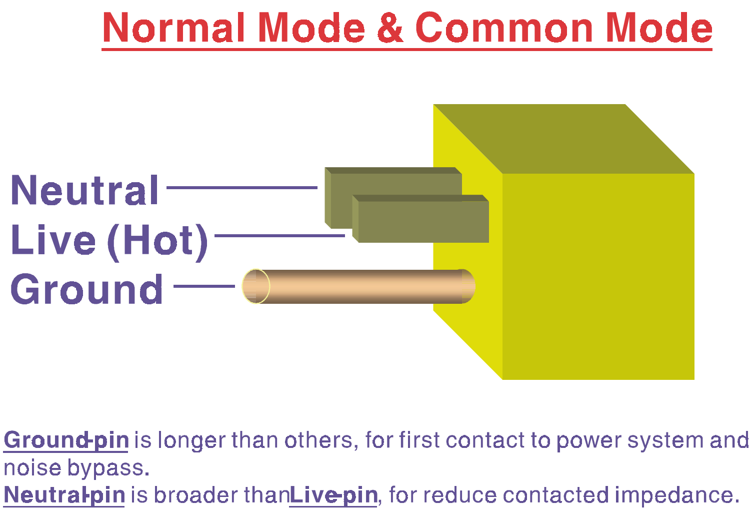


Figure E-4: Normal mode and Common mode.

- Ground-pin is longer than others, for first contact to power system and noise bypass.
- Neutral-pin is broader than Live-pin, for reducing contact impedance.

1.4 Wire impedance

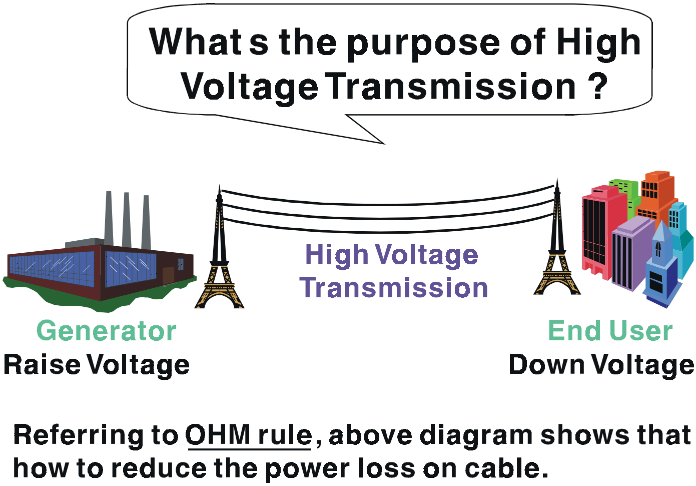


Figure E-5: The purpose of high voltage transmission

- What’s the purpose of high voltage transmission?

We have all seen high voltage transmission towers. The power plant raises the voltage while generating the power, then a local power station steps down the voltage. What is the purpose of high voltage transmission wires ? According to the energy formula, P = V * I, the current is reduced when the voltage is raised. As you know, each cable has impedance because of the metal it is made of. Referring to Ohm’s Law, (V = I * R) this decreased current means lower power losses in the wire. So, high voltage lines are for reducing the cost of moving electrical power from one place to another.

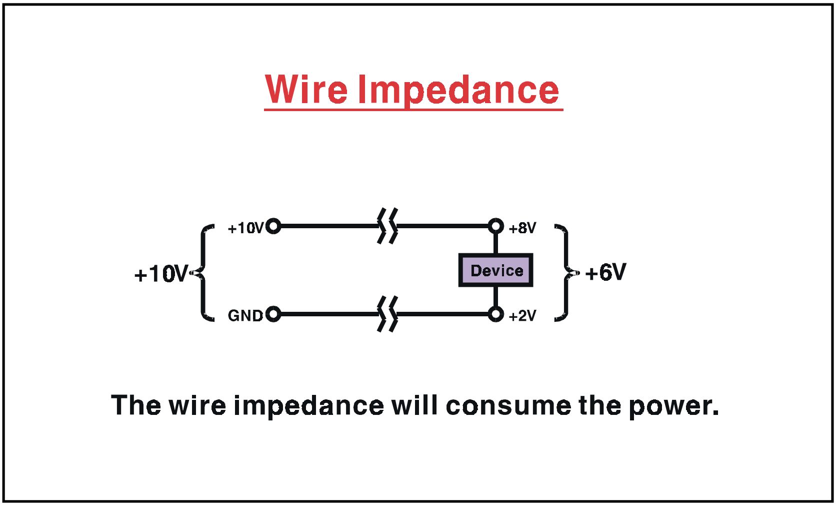


Figure E-6: wire impedance.

1.5 Single Point Grounding

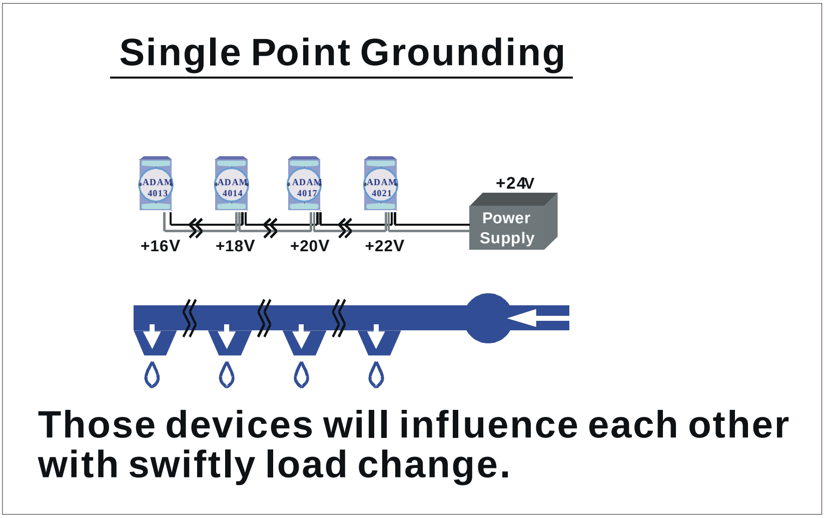


Figure E-7: Single point grounding. (1)

- What's Single Point Grounding?

Maybe you have had an unpleasant experience while taking a hot shower in Winter. Someone turns on a hot water faucet somewhere else. You will be impressed with the cold water!

The bottom diagram above shows an example of how devices will influence each other with swift load change. For example, normally we turn on all the four hydrants for testing. When you close the hydrant 3 and hydrant 4, the other two hydrants will get more flow. In other words, the hydrant cannot keep a constant flow rate.

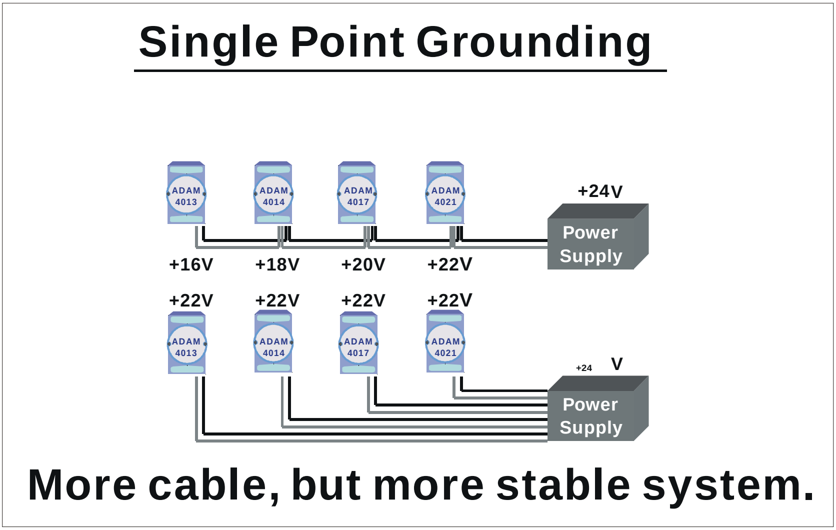


Figure E-8: Single point grounding. (2)

The above diagram shows you that a single point grounding system will be a more stable system. If you use thin cable for powering these devices, the end device will actually get lower power. The thin cable will consume the energy.

E.2 Shielding

2.1 Cable Shield

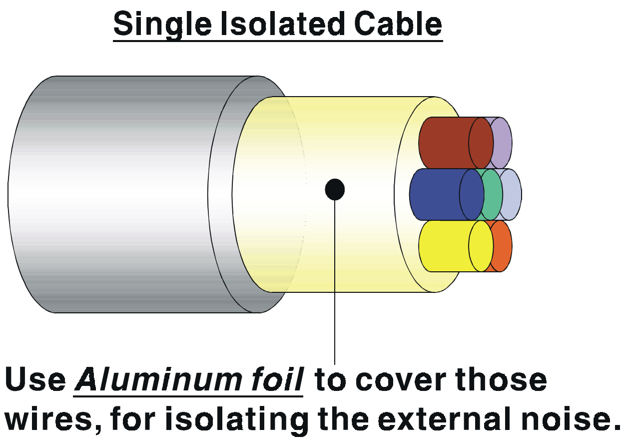


Figure E-9: Single isolated cable

- Single isolated cable

The diagram shows the structure of an isolated cable. You see the isolated layer which is spiraled Aluminum foil to cover the wires. This spiraled structure makes a layer for shielding the cables from external noise.

Grounding Reference

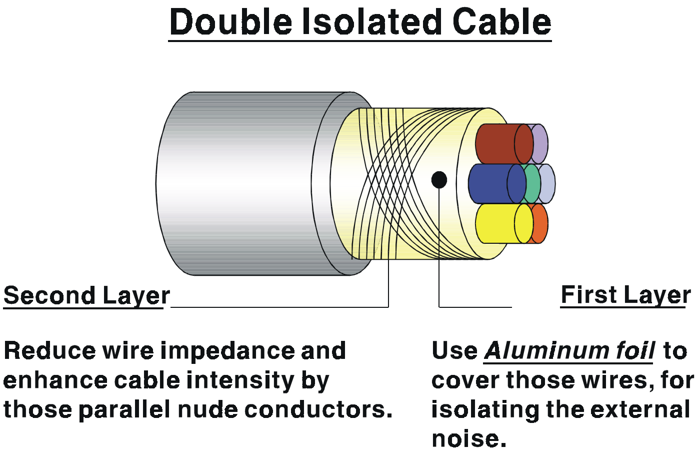


Figure E-10: Double isolated cable

- Double isolated cable

Figure 10 is an example of a double isolated cable. The first isolating layer of spiraled aluminum foil covers the conductors. The second isolation layer is several bare conductors that spiral and cross over the first shield layer. This spiraled structure makes an isolated layer for reducing external noise.

Additionally, follow these tips just for your reference.

- The shield of a cable cannot be used for signal ground. The shield is designed for carrying noise, so the environment noise will couple and interfere with your system when you use the shield as signal ground.
- The higher the density of the shield - the better, especially for communication network.
- Use double isolated cable for communication network / AI / AO.
- Both sides of shields should be connected to their frame while inside the device. (for EMI consideration)
- Don’t strip off too long of plastic cover for soldering.

2.2 System Shielding

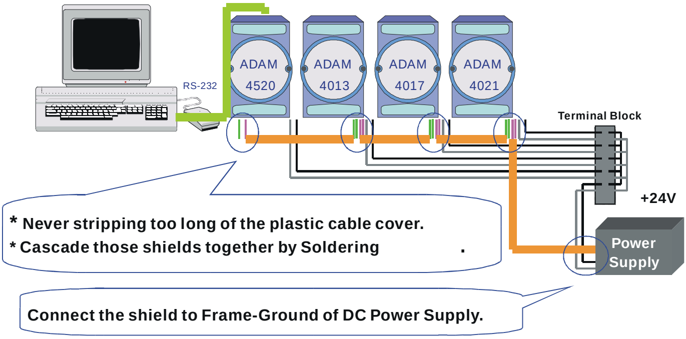


Figure E-11: System Shielding

- Never stripping too much of the plastic cable cover. This is improper and can destroy the characteristics of the Shielded-Twisted-Pair cable. Besides, the bare wire shield easily conducts the noise.
- Cascade these shields together by soldering. Please refer to following page for further detailed explanation.
- Connect the shield to Frame Ground of DC power supply to force the conducted noise to flow to the frame ground of the DC power supply. (The 'frame ground' of the DC power supply should be connected to the system ground)

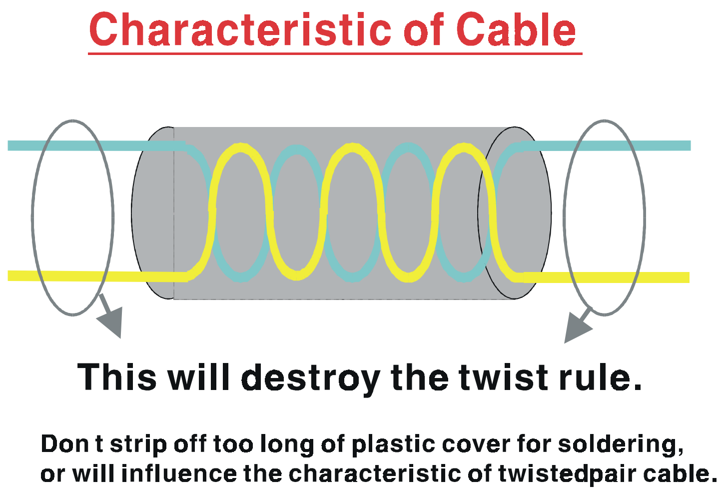


Figure E-12: The characteristic of the cable

- The characteristic of the cable

Don't strip off too much insulation for soldering. This could change the effectiveness of the Shielded-Twisted-Pair cable and open a path to introduce unwanted noise.

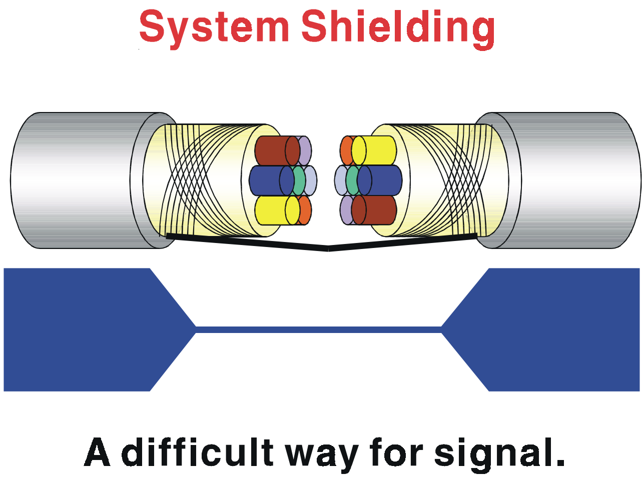


Figure E-13: System Shielding (1)

- Shield connection (1)

If you break into a cable, you might get in a hurry to achieve your goal. As in all electronic circuits, a signal will use the path of least resistance. If we make a poor connection between these two cables we will make a poor path for the signal. The noise will try to find another path for easier flow.

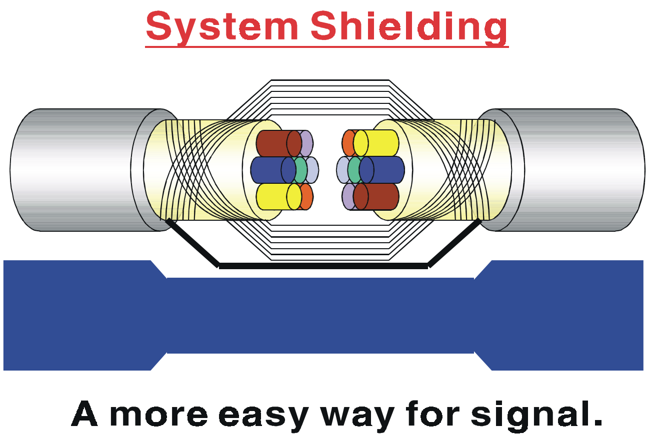


Figure E-14:System Shielding (2)

- Shield connection (2)

The previous diagram shows you that the fill soldering just makes an easier way for the signal.

E.3 Noise Reduction Techniques

- Isolate noise sources in shielded enclosures.
- Place sensitive equipment in shielded enclosure and away from computer equipment.
- Use separate grounds between noise sources and signals.
- Keep ground/signal leads as short as possible.
- Use Twisted and Shielded signal leads.
- Ground shields on one end ONLY while the reference grounds are not the same.
- Check for stability in communication lines.
- Add another Grounding Bar if necessary.
- The diameter of power cable must be over 2.0 mm^2.
- Independent grounding is needed for A/I, A/O, and communication network while using a jumper box.
- Use noise reduction filters if necessary. (TVS, etc)
- You can also refer to FIPS 94 Standard. FIPS 94 recommends that the computer system should be placed closer to its power source to eliminate load-induced common mode noise.

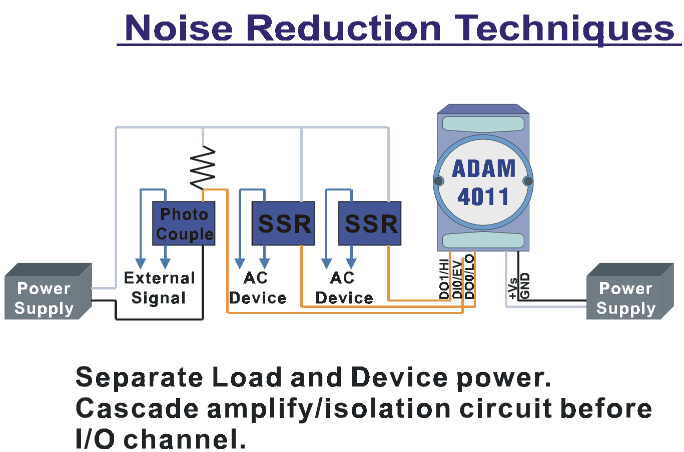


Figure E-15: Noise Reduction Techniques

E.4 Check Point List

- Follow the single point grounding rule?
- Normal mode and common mode voltage?
- Separate the DC and AC ground?
- Reject the noise factor?
- The shield is connected correctly?
- Wire size is correct?
- Soldered connections are good?
- The terminal screw are tight?