Copyright Notice

Acknowledgment

ADAM is a trademark of Advantech Co., Ltd.

IBM and PC are trademarks of International Business Machines Corporation.

CE Nonification

The ADAM-5510 series developed by Advantech Co., Ltd. has passed the CE test for environmental specifications. Test conditions for passing included the equipment being operated within an industrial enclosure, using shielded twisted-pair RS-485 cables and having SFC-6 sleeve core clamps added to the power cable (see Figure 0-1 in Chapter 0). In order to protect the ADAM-5510 system from being damaged by ESD (Electrostatic Discharge) and EMI leakage, we strongly recommend the use of CE-compliant industrial enclosure products, shielded twisted-pair RS-485 cables, and core clamps.

6th Edition
March 2005

A Message to the Customer...

Advantech Customer Services

Each and every Advantech product is built to the most exacting specifications to ensure reliable performance in the unusual and demanding conditions typical of industrial environments. Whether your new Advantech equipment is destined for the laboratory or the factory floor, you can be assured that your product will provide the reliability and ease of operation for which the name Advantech has come to be known.

Your satisfaction is our number one concern. Here is a guide to Advantech's customer services. To ensure you get the full benefit of our services, please follow the instructions below carefully.

Technical Support

We want you to get the maximum performance from your products. So if you run into technical difficulties, we are here to help. For most frequently asked questions you can easily find answers in your product documentation. These answers are normally a lot more detailed than the ones we can give over the phone.

So please consult this manual first. If you still cannot find the answer, gather all the information or questions that apply to your problem and, with the product close at hand, call your dealer. Our dealers are well trained and ready to give you the support you need to get the most from your Advantech products. In fact, most problems reported are minor and are able to be easily solved over the phone.

In addition, free technical support is available from Advantech engineers every business day. We are always ready to give advice on application requirements or specific information on the installation and operation of any of our products.

Website information:

You can access the most current support on our website:

http://www.advantech.com/support or

http://www.advantech.com/support/kbase-dir.htm

Product Warranty

Advantech warrants to you, the original purchaser, that each of its products will be free from defects in materials and workmanship for one year from the date of purchase.

This warranty does not apply to any products which have been repaired or altered by other than repair personnel authorized by Advantech, or which have been subject to misuse, abuse, accident or improper installation. Advantech assumes no liability as a consequence of such events under the terms of this Warranty.

Because of Advantech's high quality-control standards and rigorous testing, most of our customers never need to use our repair service. If an Advantech product ever does prove defective, it will be repaired or replaced at no charge during the warranty period. For out-of-warranty repairs, you will be billed according to the cost of replacement materials, service time and freight. Please consult your dealer for more details.

If you think you have a defective product, follow these steps:

1. Collect all the information about the problem encountered (e.g. type of PC, CPU speed, Advantech products used, other hardware and software used etc.). Note anything abnormal and list any on-screen messages you get when the problem occurs.
2. Call your dealer and describe the problem. Please have your manual, product, and any helpful information readily available.
3. If your product is diagnosed as defective, obtain an RMA (return material authorization) number from your dealer. This allows us to process your return more quickly.
4. Carefully pack the defective product, a completely filled-out Repair and Replacement Order Card and a photocopy of dated proof of purchase (such as your sales receipt) in a shippable container. A product returned without dated proof of purchase is not eligible for warranty service.
5. Write the RMA number visibly on the outside of the package and ship it prepaid to your dealer.

Table of Contents

Table of Figures

Tables

0

Quick Start

Quick Start

Welcome & Preview

Welcome to the Advantech ADAM-5510 user's guide. This chapter was written to provide users with a fast and easy installation guide and with the basic operating skills necessary to use the ADAM-5510's core capabilities.

This Quick Start chapter contains the following sections:

- System requirements
- Installation guide
- Set up and configuration instructions
- A training example

0.1 System Requirements

0.1.1 Host computer

1. IBM PC compatible computer with 286 or above CPU, HDD, FDD, and at least 640 KB of conventional memory plus 1 MB of extended memory.
2. At least one standard RS-232 port (e.g. COM1, COM2).
3. One RS-232 DB-9 straight-through cable for downloading/programming.
4. One RS-232 DB-9 cross-over cable for communication/debugging.
5. DOS version 3.31 or higher.
6. Borland Turbo C++ 3.0 for DOS.

0.1.2 ADAM-5510

1. One ADAM-5510 base unit with two blank slot covers.
2. One ADAM-5510 user's manual.
3. One core clamp for use in power supply connection.
4. Two ADAM-5510 utility diskettes.

5. Power supply for the ADAM-5510 (+10 to +30 V_{DC}) with power cable.

0.1.3 I/O modules

At least one I/O module is needed to use the system. A variety of I/O modules are available to meet different application requirements. Table 0-1 gives a current listing of these modules as of this printing:

Module	Name	Specification	Reference
1. Analog I/O	ADAM-5013	3-ch. RTD input	Isolated
	ADAM-5017	8-ch. AI	Isolated
	ADAM-5017H	8-ch. high speed AI	Isolated
	ADAM-5018	7-ch. thermocouple input	Isolated
	ADAM-5024	4-ch. AO	Isolated
2. Digital I/O	ADAM-5050	16-ch. universal DI/O	Non-isolated
	ADAM-5051	16-ch. DI	Non-isolated
	ADAM-5052	8-ch. DI	Isolated
	ADAM-5056	16-ch. DO	Non-isolated
3. Relay output	ADAM-5060	6-ch. relay output	Isolated
	ADAM-5068	8-ch. relay output	Isolated
4.Counter/Frequency	ADAM-5080	4-ch. counter/frequency input	Selectable Isolation
5.Serial I/O	ADAM-5090	4-port RS-232 Module	Non-isolated

Table 0-1: *ADAM-5000 I/O modules*

0.2 Installation

1. Open the ADAM-5510 package and make sure that the following items are present:
   1- One ADAM-5510 base unit with two blank slot covers.
   2- One ADAM-5510 user's manual.
   3- One core clamp for use in power supply connections.
   4- Two ADAM-5510 utility diskettes.

2. Make sure that everything described in Section 0.1 is ready.
3. Connect the ADAM-5510 power cable between the power supply and the ADAM-5510 screw terminals (+Vs and GND). Make sure that the power source is between +10 to +30 V_{DC}.
4. Connect the download cable between the host computer and the ADAM-5510. A standard DB-9 pin cable (straight-through type) can be used to connect the ADAM-5510 (programming port) and the host PC's RS-232 port.

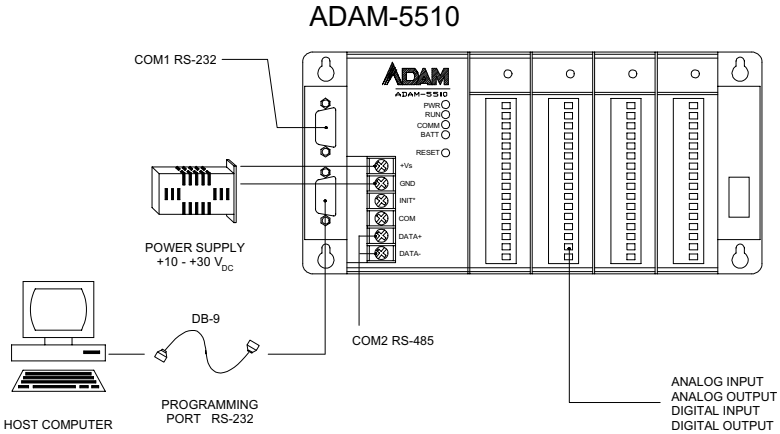


Figure 0-1: *Installation wiring of the ADAM-5510 and host PC*

5. The default software package supported by ADAM-5510 is Borland Turbo C++ 3.0 for DOS. We suggest installing C programming tools on the host PC to aid in modifying your ADAM-5510 application programs.

***** Note:*** *If you have only one COM port, make sure you use an RS-232 DB-9 straight-through cable to connect the host PC and the ADAM-5510 during downloading/programming. Reconnect the host PC and ADAM-5510 using an RS-232 DB-9 crossover cable to proceed with communication/debugging. Otherwise a failure in installation or communication will result.*

0.3 Utility application program: ADAM5510.EXE

0.3.1 Configuration

1. Insert ADAM-5510 utility diskette No. 1 into the floppy disk drive (e.g. A:) in the host PC. Change the host computer default drive from C: to A:
2. Key in **install <Enter>**. The install program on the diskette will automatically complete the installation. When installation is completed, the default working directory will be automatically changed to c:\5510.
3. Key in **adam5510 <Enter>**.
4. This starts the utility program for ADAM-5510. The highlighted cursor will be at its default location at “COMport” on the menu bar.
5. Press **<Enter>**; a pop-up window will appear. Use the arrow keys to move the highlighted cursor to select a COM port on the host PC, then press **<Enter>** to confirm the selection. Press **<ESC>** to return to the menu bar.

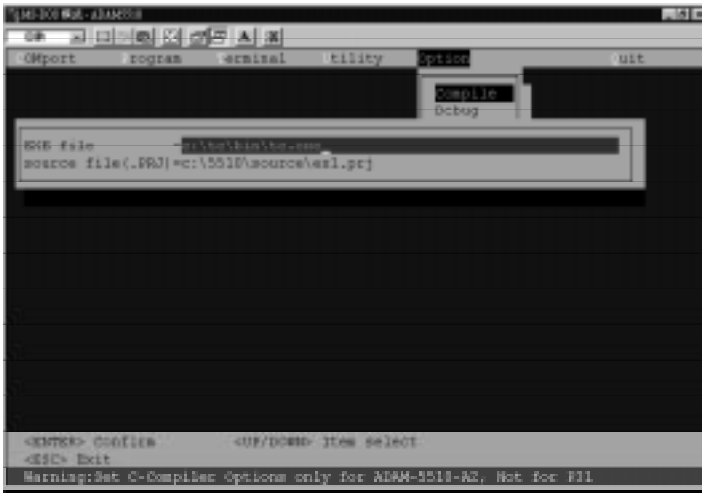


Figure 0-2: Menu for setting up compiler working paths

6. Move the cursor to **"Option"** and press **<Enter>** to activate the pull-down menu. Move the cursor to **"Compiler"** on the pull-down menu and press **<Enter>**. The screen now looks like Figure 0-2 (see previous page).
7. Use the arrow keys and <**Enter**> to set the individual access paths for each item. We suggest using the .PRJ file to manage your application program.
8. The access path for your own application program should be filled in on the **"Project file(.PRJ)"** line.
9. Other items in the **"Option"** pull-down menu can be set using procedures similar to steps 5 through 8. Table 0-2 provides a reference for this configuration procedure.

Menu item	Sub-menu option	Default value	Reference
Compiler	EXE file	C:\tc\bin\tc.exe	Modify settings according to actual paths at the user's host PC
	Project file (.PRJ)	C:\5510\source\ex1.prj	
Debugger*	EXE file	C:\td30\td.exe	Turbo debug program
	PC comm port	COM1	Host PC COM port
	PC comm baud rate	9600	Host PC COM port baud rate
	Debug file (.exe)	Ex1.exe	
	Programming 5510	Yes	
	Program path	C:\5510\debugdl\	
Editor	EXE file	C:\dos\edit.com	DOS text file editor
	Edit file	C:\5510\source\ex1.c	
Configure	EXE file	C:\5510\adam.exe	Setup ADAM-5000 I/O module utility
	Programming 5510	Yes	
	Program path	C:\5510\configdl\	

Table 0-2: *Reference table for setting up working paths*

**Note:* *In addition to Borland Turbo C, you have to purchase Borland Turbo Debugger if you need to debug your programs.*

0.3.2 I/O modules configuration

1. Move the cursor to **"Utility"**, select **"Configure"** and press **<Enter>.**
2. After displaying some processing messages, the screen shown in Figure 0-3 will appear. This screen can be used for configuring and calibrating ADAM-5000 I/O modules.

Figure 0-3: *"Utility/Configure" menu screen*

3. This menu screen consists of a menu bar at the top of the screen and a status field that displays information about the connected modules. When you first start the program, it automatically scans for any attached modules and displays their data. Module characteristics, module configuration parameters and input or output values will be displayed in the status field.

Menu Item	Description	Reference
COMport	Select host PC COM port (eg. COM1) and configure other parameters.	
Search	Search for ADAM-5000 series plug-in modules.	
Setup	Specify module types & I/O range values, including three options: ✓ System setting ✓ Module setting ✓ Output data	
Run	Get current I/O values & module status.	
Calibrate	Calibrate modules. Parameters are stored in each module's onboard EEPROM.	
Terminal	Change to ADAM-5510 terminal mode allowing key-in of commands, then return to main menu.	
File	Save module setup values in the host PC.	
Quit	Exit main menu. Return to last operating status.	

Table 0-3: *Module configuration utility menu selections*

0.3.3 Compiling and debugging user defined applications

ADAM-5510.EXE can also be used to compile your application programs. Call Turbo C and the Turbo Debugger from this utility.

1. Make sure the relevant access paths were all properly specified. (Refer to Section 0.3.1)
2. Move the cursor to **"Utility"**, select **"Compile"** and press <**Enter**>. The Turbo C Integrated Developing Environment will be invoked and will load the .PRJ file just specified. Set the compiler options to ***x87 Emulation*** and ***8088/86 or 80186 CPU Only***.

3. After compiling your source code to an executable file, you can trace the execution process using the Turbo Debugger, if necessary. Move the cursor to **"Utility"**, select **"Debug"** and press <**Enter**>. Note that if you have only one COM port on your host PC, you must replace the cable connecting the host PC and ADAM-5510 before debugging. Replace the straight-through cable with cross-over cable.
4. To modify defective source code, move the cursor to **"Utility"**, select **"Editor"** and press <**Enter**>. The text editor specified in the **"Option/Editor"** working path will be invoked.

***** Note:*** *Crossover DB-9 cable must be used with the Turbo Debugger.*

0.3.4 Run program and download

1. Move cursor to **"Program"** and press <**Enter**>.
2. The program downloading process will start and will take about two minutes. A counter will be displayed on the screen indicating the percentage completed. When the downloading process finishes, the screen shown in Figure 0-4 will be displayed.

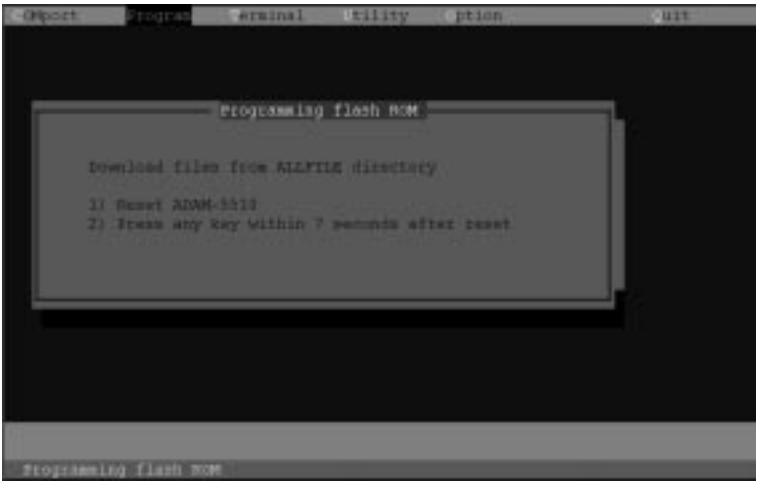


Figure 0-4: *Downloading to ADAM-5510's Flash ROM*

3. Press the **"RESET"** button on the ADAM-5510 front panel.
4. Press any key within 7 seconds after pressing **"RESET"**. Wait for ADAM-5510 to boot from ROM-DOS.
5. After ADAM-5510 resets and reboots, the welcome screen shown in Figure 0-5 will appear. Now you are operating the ADAM-5510 remotely through your host PC, and may access the ROM disk.

Figure 0-5: *Opening screen; operating ADAM-5510 remotely*

6. The ROM-DOS system can now be operated in MS-DOS/PC-DOS prompt mode. That is, you can input commands to operate the ROM disk and access files on the ADAM-5510.

***** Note:*** *The ADAM-5510's operating system 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++. However, there are limitations when running application programs in the ADAM-5510. For example, the ROM-DOS command set does not include commands that operate through the console to control the screen, e.g.* ***"CLS"*** *(clear screen), etc. In order to build successful applications, you should keep these limitations and concerns in mind. For more detailed information, please refer to Chapter 6.*

0.4 Example

Let's rehearse the above instructions with a simple example: EX1.PRJ

1. Connect the host PC and the ADAM-5510 as described in section 0.2. Note that the COM port of the host PC and ADAM-5510 must be connected by a straight-through DB-9 cable.
2. Install 3 I/O modules in the ADAM-5510: ADAM-5017, 5018 and 5013 in slots 1, 2, and 3 respectively.
3. Insert the Utility Disk into the floppy disk drive in the host PC. Run **Install.exe**. This executable program will copy necessary header files and libraries to the respective Turbo C header file and library directories.
4. The installation proceeds automatically. Press any key after successful installation.
5. Now the default working directory should be **c:\5510**. Run **Adam5510.exe**.
6. Select **"COMport"**. Choose that COM port through which you want to connect the host PC to the ADAM-5510. For example: COM1; the baud rate was previously set. Press **Esc** to return to the menu bar.
7. Select **"Utility/Config"**. Wait as instructed by the screen. File transfer and rebooting will proceed automatically and a scan will identify all ADAM I/O modules. Finally, a screen similar to that shown in Figure 0-3 should appear.
8. Configure the system and I/O modules, if necessary, then select **"Quit"** and return to the main utility.
9. Select **"Option/Compile"**, then fill in the path for the Turbo C IDE and **"c:\5510\source\ex1.prj"** for the source file path, where indicated.
10. Return to the menu bar, select **"Utility/Compile"**. Turbo C IDE will be invoked to load our example file—**Ex1.prj**. Compile this example file into an executable file.

11. Press **Alt-T** to transfer the file Ex1.exe from the host PC to the ADAM-5510. Specify **Source filename: c:\5510\source\ex1.exe** and save to **ADAM-5510 Disk: d**: (SRAM virtual disk) or **ADAM-5510 Disk: e**: (Flash disk). Note that if you save the executable file to another directory, that directory must be specified.
12. Now you can run **ex1.exe** on ADAM-5510. This example reads the data configuration of the low speed AI modules in the ADAM-5510, then shows the data on the screen.
13. The above are typical steps for initializing the host PC and ADAM-5510. After steps 1 through 8 are complete, the ADAM-5510 is ready for standalone operation, provided that a proper application executable has been transferred to it.

0.5 Conclusion

By carefully walking through all the installation and operating instructions described above, you should have experienced most of the usual procedures for using ADAM-5510. For further sophisticated utilization or programming skills, please refer to following chapters.

This page intentionally left blank

1

Introduction

Introduction

1.1 Standalone Data Acquisition and Control System

The task of monitoring and controlling a process in a laboratory or an industry can be extremely complex. As the number of data collection points and process control actuators in the network increases, this task becomes more challenging. PC-based DA&C systems provide a high value alternative to older, more expensive technology in the control industry, but most systems require a desktop PC or equivalent at their core. Now Advantech's ADAM-5510 can operate a process control network independent of a desktop PC.

1.2 Features

1.2.1 Control flexibility with C programming

The ADAM-5510 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 a way similar to how one uses an x86 PC in the office. The ADAM-5510 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 very modest training expense requirements. See **Chapter 3 ADAM-5510 System** for detailed technical specifications.

1.2.2 RS-232/485 communication ability

The ADAM-5510 has three serial communication ports, giving it excellent communication abilities. This facilitates its ability to control networked 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-5510 to satisfy diverse communication demands. COM3 is a 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-5510 uses a convenient backplane system common to the ADAM-5000 series. Advantech's complete line of ADAM-5000 modules integrates with the ADAM-5510 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-5510's function library to reduce programming effort. Users can easily call these subroutines to execute the ADAM-5510'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 microcontroller 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-5510 ideal for use in applications which require a high level of system stability.

1.3 System Configuration

The following diagram shows a possible ADAM-5510 system configuration:

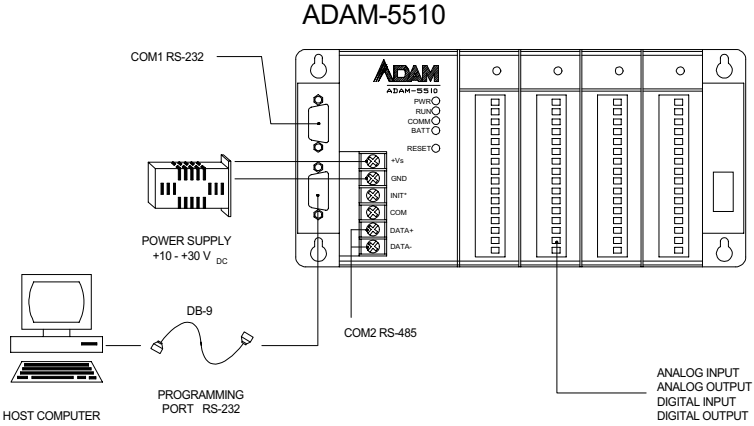


Figure 1-1: *ADAM-5510 system configuration*

This page intentionally left blank

2

Installation Guidelines

Installation Guidelines

This chapter explains how to install an ADAM-5510 microcontroller. A quick hookup scheme is provided that lets you easily configure your system before implementing it into your application.

2.1 Starting up ADAM-5510

Step 1: Review your requirements

Before you start installing the ADAM-5510, make sure the system requirements are met:

Host computer

1. IBM PC compatible computer with 286 or above CPU, HDD, FDD, and at least 640 KB of conventional memory plus 1 MB of extended memory.
2. At least one standard RS-232 port (e.g. COM1, COM2).
3. One RS-232 DB-9 straight through cable for downloading/ programming.
4. One RS-232 DB-9 crossover cable for communication/debugging.
5. DOS version 3.31 or higher.
6. Borland Turbo C++ 3.0 for DOS.
7. Borland Turbo Debugger (optional). This software is required only if you want to debug application programs on the ADAM-5510.

ADAM-5510

1. One ADAM-5510 base unit wiht two blank slot covers.
2. One ADAM-5510 user's manual.
3. One core clamp for power supply connection.
4. Two ADAM5510 utility diskettes
5. Power supply for the ADAM-5510 (+10 to +30 V_{DC}) with power cable.

I/O modules

1. At least one ADAM-5000 series I/O module.

Step 2: Wiring the power cable and download cable

Connect the power cable between the power supply and the ADAM-5510. Make sure the power source is between +10 to +30 V_{DC}. Screw terminals +Vs and GND are for power supply wiring.

Connect the download cable between the host computer and the ADAM-5510. Standard DB-9 pin cables (straight-through type) can be used to connect the ADAM-5510 (programming port) and host-PC (RS-232 port).

The following figure shows how to connect the cables:

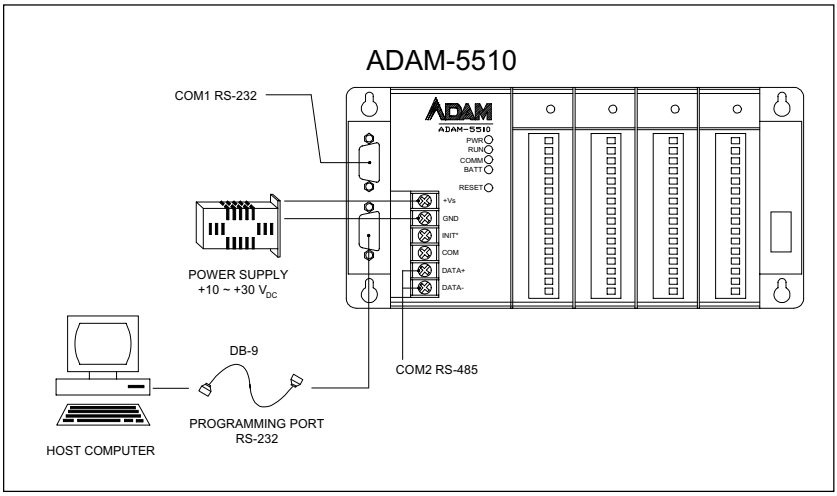


Figure 2-1: *ADAM-5510 wiring and connections*

Step 3: Run utility software in host computer

You will find a utility disk containing an ADAM5510.EXE file in the same box as your ADAM-5510 unit. This file is a menu-driven software utility provided for downloading user's programs. When the file is executed, the main screen appears, as shown in Figure 2-2.

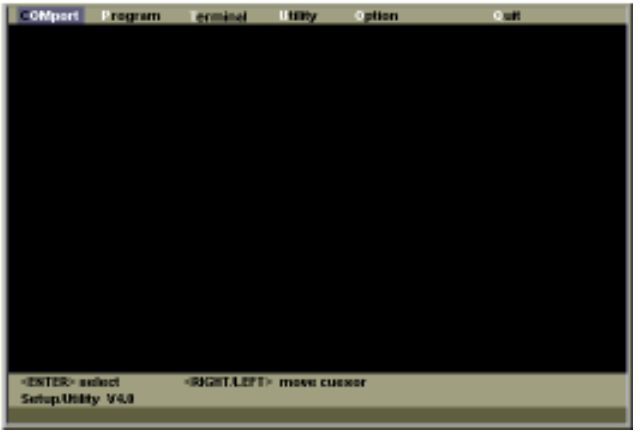


Figure 2-2: *ADAM-5510 software utility*

Select COM port

First, highlight the "**COMport**" option on the top bar and press <**Enter**>. The status field (shown below) will appear. Second, highlight the COM port of PC you used to connect the ADAM-5510, then press <**Enter**>. The baud rate is set to a default value of 57600 bps and cannot be changed. The screen is as shown in Figure 2-3:

Figure 2-3: *Communication port selection*

Step 4: Power on the ADAM-5510 unit

Highlight the "**Terminal**" option, then press <**Enter**>. Power on the ADAM-5510. After about 5 seconds, the screen shown in Figure 2-4 will appear, showing that the system has successfully started.

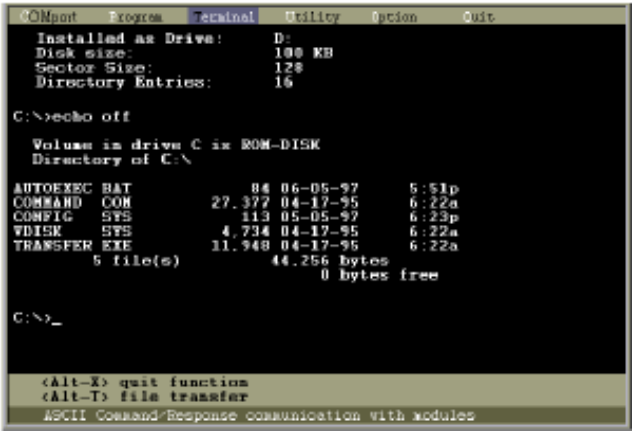


Figure 2-4: *Emulating screen of ADAM-5510*

2.2 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.

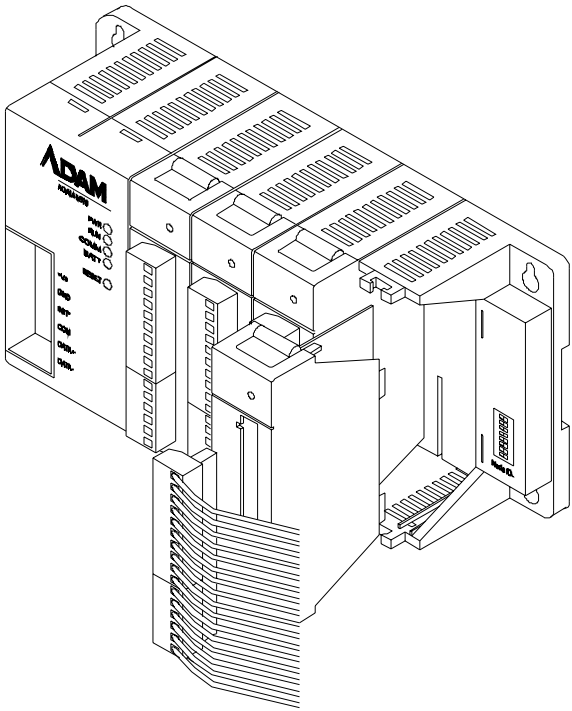


Figure 2-5: *Module alignment and installation*

2.3 I/O Slots and I/O Channel Numbering

The ADAM-5510 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 thru 7.

2.4 Mounting

The ADAM-5510 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.

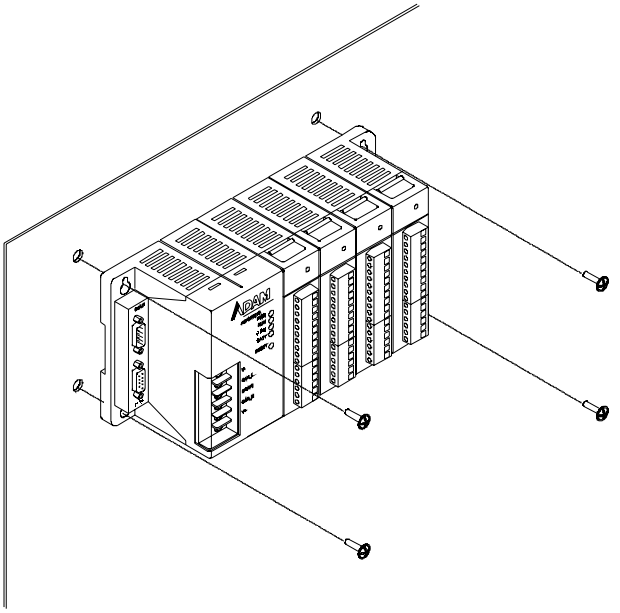


Figure 2-6: *ADAM-5510 panel mounting screw placement*

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 on to 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.

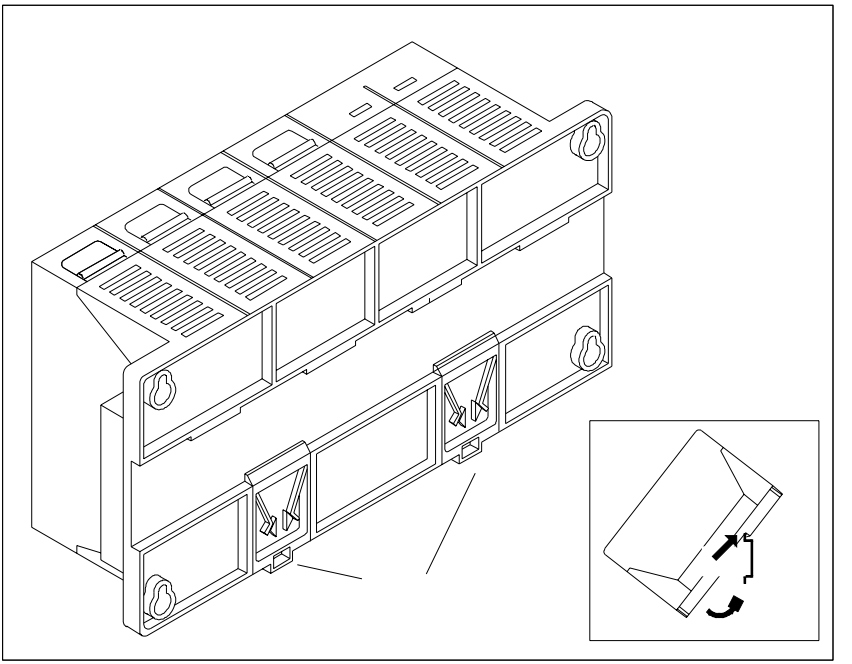
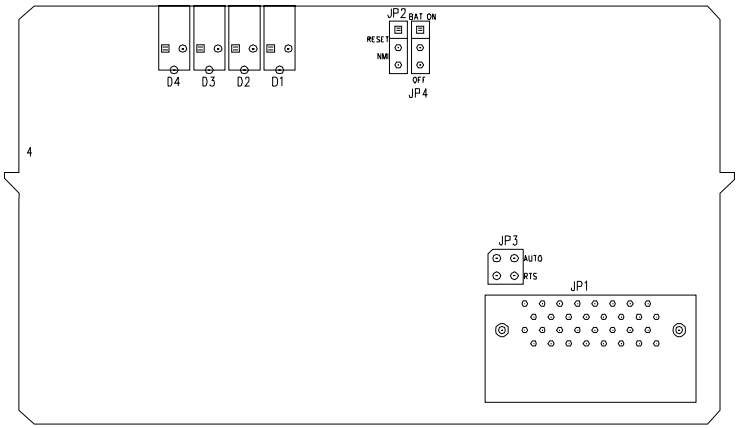


Figure 2-7: *ADAM-5510 rail mounting*

2.5 Jumper Settings and DIP Switch Settings

This section tells you how to set the jumpers and DIP switch to configure your ADAM-5510 system. It gives the system default configuration and your options for each jumper and dip switch. There are three jumpers on the CPU card, and one 8-pin DIP switch on backplane board.

The following figure shows the location of the jumpers:



* JP4 is for battery power ON/OFF

Figure 2-8: *Jumper locations on the CPU card*

COM2 port RS-485 control mode setting

The COM2 port is dedicated as an RS-485 interface. Jumper JP3 on the CPU card lets you configure the control mode for automatic control or RTS control. Jumper settings are shown below:

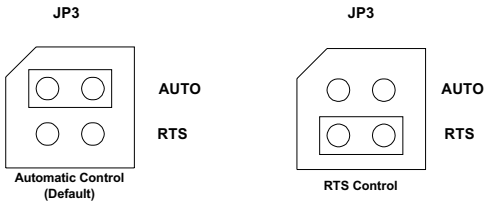


Figure 2-9: *COM2 port RS-485 control mode setting (JP3)*

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:

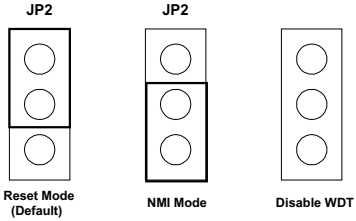


Figure 2-10: *Watchdog timer setting*

Network address setting

Set the network address using the 8-pin DIP switch. Valid settings range from 0 to 255 (00h to FFh) where ON in any of the 8 DIP switch positions equates to a binary 1, and OFF equates to a binary 0.

For example, if the Node ID is 03h, the DIP switch settings for switches 1 and 2 (representing bits 1 and 2) would both be ON while the rest of the switches would be OFF. The default Node ID is 01h.

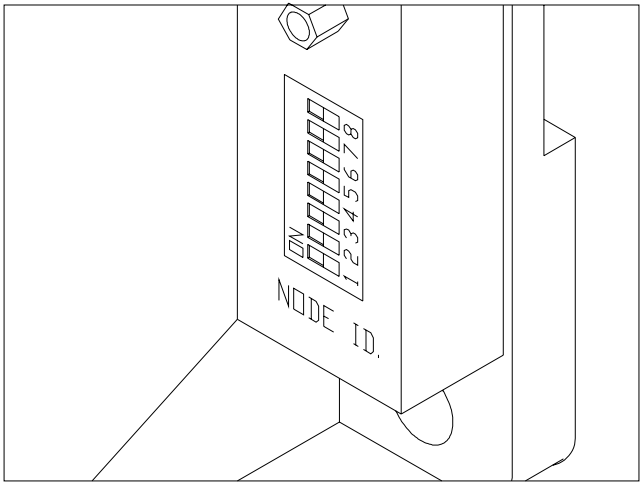


Figure 2-11: *ADAM-5510 network address DIP switch*

2.6 Wiring and Connections

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

Power supply wiring

Although the ADAM-5510 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 mm².

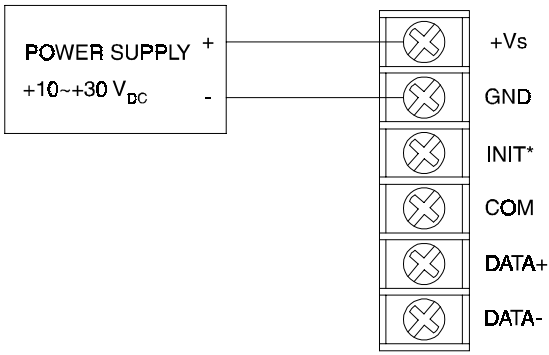


Figure 2-12: *ADAM-5510 power wiring*

I/O modules wiring

The system uses a plug-in screw terminal block for the interface between an I/O module 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 mm² to 2.5 mm².
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.

Programming port connection

The ADAM-5510 has a programming port with a DB-9 connection. This port allows you to program, configure, and troubleshoot the ADAM-5510 from your host computer. The programming port has an RS-232 interface and only uses TX, RX, and GND signals. The pin assignment of the port is as follows:

Pin No.	Description
Pin 1	Not Used
Pin 2	Tx
Pin 3	Rx
Pin 4	Not Used
Pin 5	GND
Pin 6	Not Used
Pin 7	Not Used
Pin 8	Not Used
Pin 9	Not Used

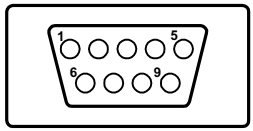


Table 2-1: *DB-9 programming port pin assignments*

RS-232 port connection

The COM1(3F8) port is dedicated as an RS-232 interface and has a DB-9 connector. Since the connection for an RS-232 interface is not standardized, different devices implement the RS-232 connection in different ways. If you are having problems with a serial device, be

sure to check the pin assignments for the connector. The following table shows the pin assignments for the COM1 port.

Pin No.	Description
Pin 1	DCD
Pin 2	Rx
Pin 3	Tx
Pin 4	DTR
Pin 5	GND
Pin 6	DSR
Pin 7	RTS
Pin 8	CTS
Pin 9	RI

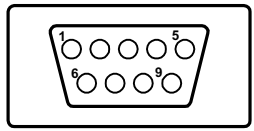


Table 2-2: *RS-232 port pin assignments*

RS-485 port connection

The COM2 (2F8) port is dedicated as an RS-485 interface. Screw terminals DATA- and DATA+ are used for making the COM2 RS-485 connections.

2.7 LED Status of the ADAM-5510-A2 Unit

The ADAM-5510-A2 unit front panel cover has four LEDs which display the following four status:

a. Red LED(Power): base unit power status; light is ON when power is ON in base unit.

b. Green LED(RUN): base unit booting status; light is on during boot-up.

c. Orange LED(COMM): base unit comm status; light is on during base unit communication.

d. Yellow LED(BATT): base unit battery status; light is on when battery is low in base unit.

This page intentionally left blank

3

ADAM-5510 System

ADAM-5510 System

3.1 Overview

The ADAM-5510 is a PC-based programmable microcontroller for standalone data acquisition and control which can control, monitor and acquire data through multichannel 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 4 I/O modules (up to 64 I/O points). The system also provides serial communication ports (RS-232/485), allowing the system to communicate with other devices for versatile applications.

3.2 Major Features

The ADAM-5510 system consists of two major components: the main unit and I/O modules. The main unit includes a CPU card, a power regulator, a 4-slot base, two serial communication ports and a programming port. It has the following major features:

Built-in 80188 CPU and ROM-DOS operating system

ADAM-5510'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 RAM disk for programming

The ADAM-5510 has built-in flash ROM and SRAM for file downloading, system operation and data storage. The system provides 170 KB free ROM to allow users to download programs. There is also 192 KB free RAM to provide the memory needed for efficient application operation and file transfer.

Built-in RS-232/485 communication ports

The ADAM-5510 has two serial communication 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-5510 system provides isolation for I/O modules (3000 V_{DC}), communication connections (2500 V_{DC}), and 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 provides a watchdog timer to monitor the microprocessor. It will automatically reset the microprocessor in the ADAM-5510 system if the system fails.

3.3 Technical Specifications of ADAM-5510 System

System

- CPU: 80188-40, 16-bit microprocessor
- Flash ROM: 256 KB (170 KB free memory for users)
- Operating system: Boot ROM-DOS
- Flash memory: 256 KB (400 KB free space for users)
- SRAM: 64 KB battery backup (192 KB 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
- I/O capacity: 4 modules (limitation: one ADAM-5024 allowed)
- CPU power consumption: 1.0 W
- Status display: Power, CPU, Communication, Battery

RS-232 interface (COM1)

- Signals: TxD, RxD, RTS, CTS, DTR, DSR, DCD, RI, 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)

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)

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)

Isolation

- Power: 3000 V_{DC}
- Input/Output: 3000 V_{DC}
- Communication: 2500 V_{DC} (COM2 only)

Power

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

Mechanical

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

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

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.

Dimensions

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

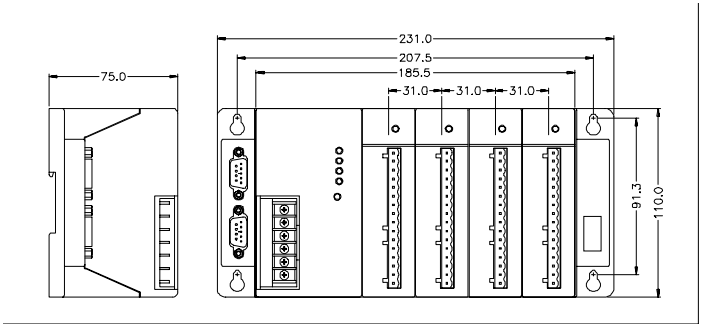


Figure 3-1: *ADAM-5510 system & I/O module dimensions*

3.4 Basic Function Block Diagram

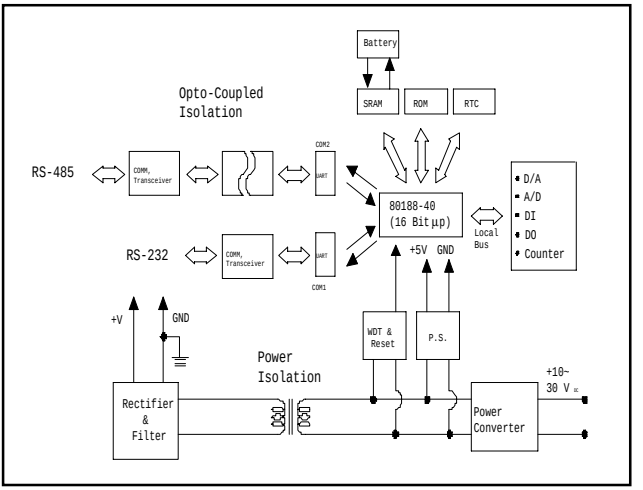


Figure 3-2: *Function block diagram*

4

I/O modules

This manual introduces the detail specifications functions and application wiring of each ADAM-5000 I/O modules. To organize an ADAM-5000 series and ADAM-5510 Series Controller, you need to select I/O modules to interface the main unit with field devices or processes that you have previously determined. Advantech provides 20 types of ADAM-5000 I/O modules for various applications so far. Following table is the I/ O modules support list we provided for user's choice. **More detailed specification and user's guides, please refer the user's manual of ADAM-5000 IO Module**. It had integrated and collected this information.

Module	Name	Specification	Reference
Analog I/O	ADAM-5013	3-ch. RTD input	Isolated
	ADAM-5017	8-ch. AI	Isolated
	ADAM-5017H	8-ch. High speed AI	Isolated
	ADAM-5018	7-ch. Thermocouple input	Isolated
	ADAM-5024	4-ch. AO	Isolated
Digital I/O	ADAM-5050	7-ch. D I/O	Non-isolated
	ADAM-5051	16-ch. DI	Non-isolated
	ADAM-5051D	16-ch. DI w/LED	Non-isolated
	ADAM-5051S	16-ch. Isolated DI w/LED	Isolated
	ADAM-5052	8-ch. DI	Isolated
	ADAM-5055S	16-ch. Isolated DI/O w/LED	Isolated
	ADAM-5056	16-ch. DO	Non-isolated
	ADAM-5056D	16-ch. DO w/LED	Non-isolated
	ADAM-5056S	16-ch. Isolated DO w/LED	Isolated
	ADAM-5056SO	16-ch. Iso. DO w/LED (source)	Isolated
Relay Output	ADAM-5060	6-ch. Relay output	Isolated
	ADAM-5068	8-ch. Relay output	Isolated
	ADAM-5069	8-ch. Relay output	Isolated
Counter/Frequency	ADAM-5080	4-ch. Counter/Frequency	Isolated
Serial I/O	ADAM-5090	4-port RS232	Non-isolated

Table 4-1: I/O Module Support List

5

Programming and Downloading

Programming and Downloading

This chapter explains how to program applications and download programs into the ADAM-5510 system. Additionally, it points out limitations and issues about which you should be aware.

5.1 Programming

The operating system of ADAM-5510 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-5510. In order to build successful applications, you should keep the following limitations and concerns in mind.

Mini BIOS functions

The ADAM-5510 provides only two serial communication ports for connecting peripherals, so the mini BIOS of ADAM-5510 only provides 10 function calls. Since the user's program cannot use other BIOS function calls, the ADAM-5510 may not work as intended. Additionally, certain language compilers such as QBASIC directly call BIOS functions that are not executable in ADAM-5510. The ADAM-5510 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-5510 mini BIOS function calls*

Converting program codes

The ADAM-5510 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*

Other limitations

1. The ADAM-5510 does not support the standard PC function “8253”. Therefore, the C language, function call “delay ( )” cannot be used in ADAM-5510 applications.

2. Certain critical files are always kept in flash ROM, such as operating system, BIOS, and monitoring files. The ADAM-5510 provides an additional 170 KB of free ROM space for downloading user applications. An additional free 192 KB of RAM is provided for operation of applications.

Programming the watchdog timer

The ADAM-5510 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 an 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.

Interrupt types

Three types of interrupts may occur in the ADAM-5510. The following table shows the types of interrupts.

Interrupt Name	Interrupt Type
Non-maskable interrupt (NMI)	02h
COM1 interrupt	0Ch
COM2 interrupt	0Eh

Table 5-2: *ADAM-5510 interrupt types*

Memory mapping

The following table shows the memory mapping of the ADAM-5510 system:

Memory Type	Function Description	Map Address	Memory Size
"Flash ROM (256KB)"	Monitor Program	0xF8000 ~ 0xFFFFF	32KB
	Mini BIOS	0xf6C00 ~ 0xF7FFF	5KB
	ROM-DOS	0xEB000 ~ 0xF6BFF	47KB
	User's Application	0xC0000 ~ 0xEAFFF	172KB
"Flash Memory (256KB)"	"User accesse and storage memory*"	0x80000 ~ 0xBFFFF	256KB
" SRAM (256KB)"	Monitor Program	0x3F000 ~ 0x3FFFF	4KB
	Battery Backup**	0x30000 ~ 0x3EFFF	60KB
	Free for user***	0x00400 ~ 0x2FFFF	191KB
	System Area	0x00000 ~ 0x003FF	1KB

Table 5-3: *ADAM-5510 memory mapping*

* Accessed by function library 'ProgramByte', 'ProgramSector' and 'EraseSector'.

** The size could be adjusted by function library 'Set_NVRAM_Size'.

*** Include 60KB RAM-Disk. You can disable RAM-Disk to get fullly 191KB memory space, or re-size the RAM-Disk via adjust the parameter of VDISK.SYS of file CONFIG.SYS.

5.2 File Download and Transfer

This section explains how to download application programs from a PC into the ADAM-5510 flash ROM and how to transfer files from a PC into ADAM-5510's SRAM.

Install utility software on host PC

ADAM-5510 systems come packaged with a utility disk containing the files and directories listed in Chapter 0 Quick Start.

Copy all the files and directories on the utility disk to the specified directory of the host computer's hard drive.

Creating the ALLFILE directory

Application programs are downloaded from a host-PC to the flash ROM of ADAM-5510 using the ADAM-5510 utility software. The ADAM-5510 utility software is first installed on a host-PC, and the user must create a new directory, also on the host PC, named ALLFILE. The user must then load into ALLFILE the following required files: The application program intended for installation in ADAM-5510; COMMAND.COM; AUTOEXEC.BAT; and CONFIG.SYS. The user should make certain that AUTOEXEC.BAT contains the name of the user's application program so that the application will automatically begin executing whenever the ADAM-5510 is powered on. When downloading to the ADAM-5510's flash ROM, the utility software first clears all non-permanent files from the flash ROM, then installs all the files contained in directory ALL-FILE into the flash ROM. It is therefore critical that all the required files be available in a directory ALLFILE when the utility software tries to access ALLFILE.

Downloading into Flash memory (ADAM-5510's C-drive)

With the ADAM-5510 utility software and the directory ALLFILE, loaded with its proper contents, installed on the host-PC, you can execute the utility software. The main screen, shown in Figure 5-2, will appear when execution begins.

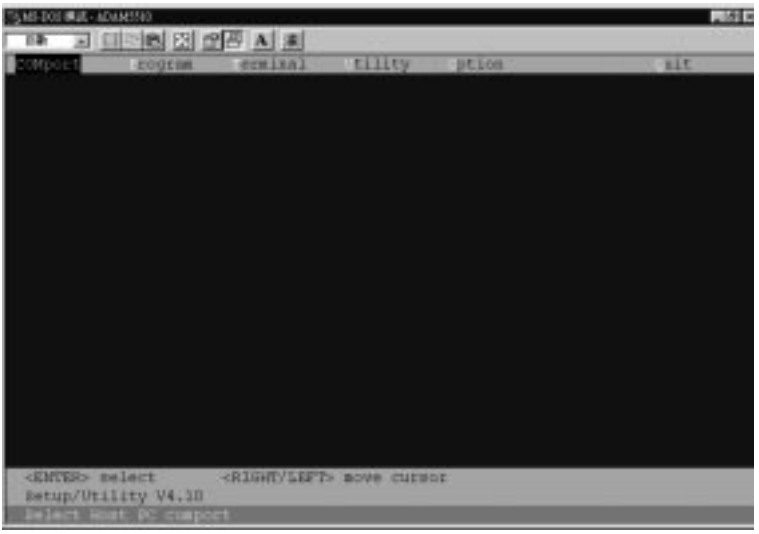


Figure 5-2: *Main screen*

After the utility software has begun executing, select the COM port of the host PC that has been connected to the ADAM-5510. Then select "**Program**" from the bar menu and press <**Enter**> to begin downloading. The screen shown in Figure 5-3 will appear.

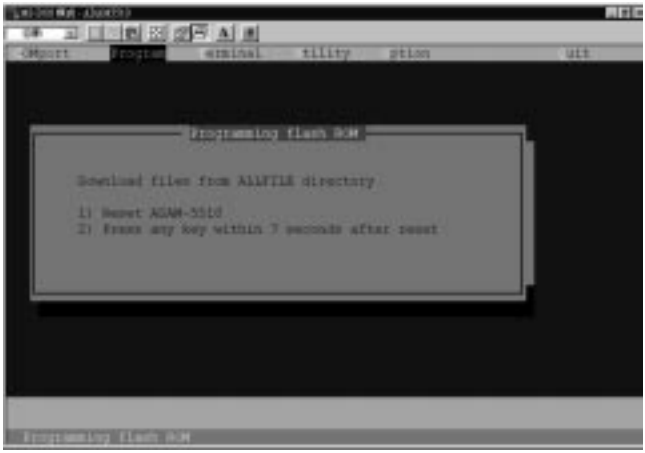


Figure 5-3: *Program downloading*

Follow the instructions shown on the screen. Power off the ADAM-5510 and then re-power on. Then press any key within 7 seconds to burn the files contained in ALLFILE into the ADAM-5510's flash ROM. After the files are successfully burned into the flash ROM, the screen shown in Figure 5-4 will appear. Power off and power on the ADAM-5510 once again. The ADAM-5510 system will automatically execute the applications.

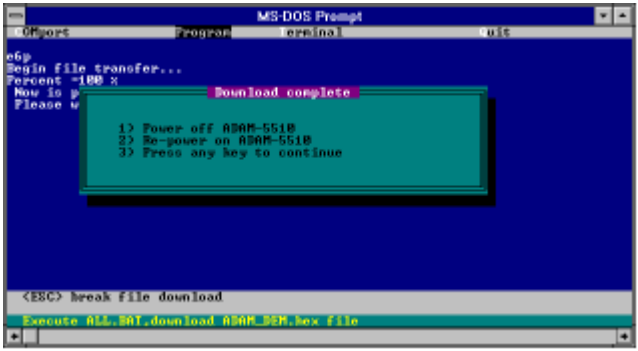


Figure 5-4: *Program downloaded successfully*

Transferring files to SRAM (ADAM-5510's D-drive)

The ADAM-5510 provides 192 KB free SRAM for use in program operation and for control logic and performance simulation before downloading the execution codes to the flash ROM. You can transfer files from a host-PC to the ADAM-5510's SRAM (D drive). Execute the utility software, select terminal mode, and press Alt-T. File transfer will begin and the screen shown in Figure 5-5 will appear.

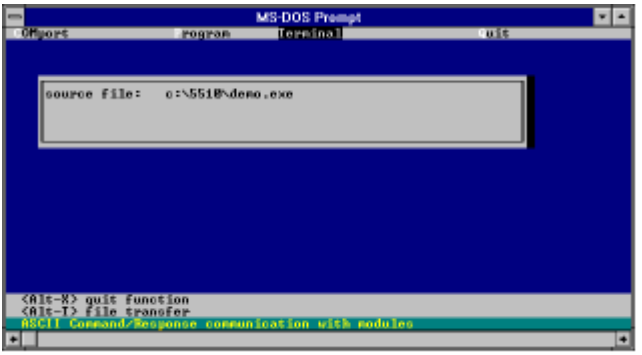


Figure 5-5: *File transfer*

Key in the specific directory and file names you want to transfer. Press <**Enter**> to complete the file transfer. You can check the files in D drive.

5.3 Setup Procedure of Remote Turbo Debugger

To help users debug applications written in Borland Turbo C, Advantech has included the executable program UPDATE.EXE on the ADAM-5510 utility disk. This executable allows users to customize the Turbo C remote debugger for ADAM-5510‘s use. Follow the steps below to customize the remote debugging environment.

1. Before you start the setup procedure, please connect a DB-9 cable between the host PC (COM1) and the ADAM-5510 (COM1) as shown in the diagram below.

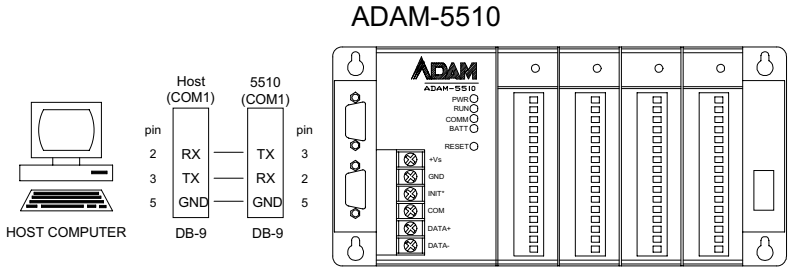


Figure 5-6: *Wiring for turbo debugging*

NOTE: A special DB-9 cable is required for this connection. The wires are transposed between pins 2 and 3 on one end, and pins 2 and 3 on the other end. Pin 2 of the connector on one end should correspond to pin 3 of the connector on the other end.

2. Copy the UPDATE.EXE from the ADAM-5510 utility disk to the directory containing the Turbo Debugger executable file in the host PC.

NOTE 1: ADAM-5510 supports Turbo C version 3.0

NOTE 2: Users must purchase and properly install the Turbo Debugger in advance.

3. Execute the UPDATE.EXE program in the Turbo Debugger directory, and follow the instructions on the screen. Input the file name TDREMOTE.EXE. The file TDADAM.EXE will be created in the directory with the Turbo Debugger after you complete this step.

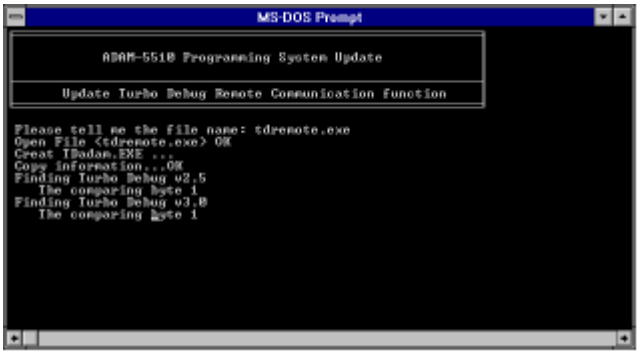


Figure 5-7: *Creating TDADAM.EXE*

4. Copy TDADAM.EXE to the DEBUGDL directory for loading into the ADAM-5510.
5. Add the following statements to the autoexec.bat file in the DEBUGDL directory:

   COM_EOI.EXE

   TDADAM.EXE -rp1 -rs4

   NOTE : Regarding related parameters for the Turbo Debugger, please refer to Borland's documentation on the Turbo Debugger

6. Download the files in the DEBUGDL directory to the ADAM-5510 FLASH ROM by selecting "**Debug**" in the "**Utility**" pull-down menu in the ADAM-5510 utility screen.
7. Reset the ADAM-5510. The ADAM-5510 is now configured for remote debugging. Now configure Turbo C on the host computer for remote debugging.

8. Open the Turbo C editor on the host computer. Select the "**Options**" item on the menu bar in the Turbo C editor. Select "**Transfer**" on the pull-down menu under the "Options" selection. See Figure 5-8:

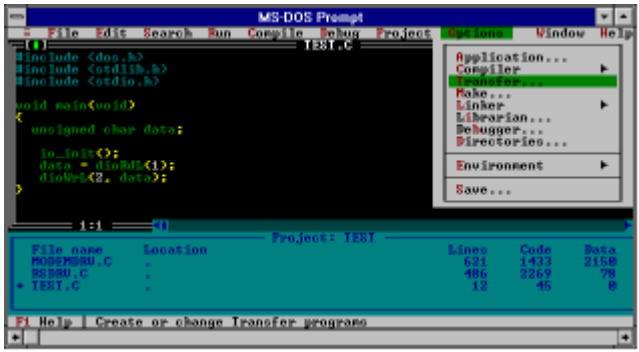


Figure 5-8: *Configuring Turbo C editor*

9. Add a new program title as shown in Figure 5-9. The Program Titles will now look something like that shown in Figure 5-10 (see next page).

Figure 5-9: *Creating new transfer item*

Figure 5-10: *New transfer item for ADAM-5510 turbo debugger*

10. Now, every time you want to use the Turbo Debugger to debug the program in ADAM-5510, you can select from the menu as shown in Figure 5-11.

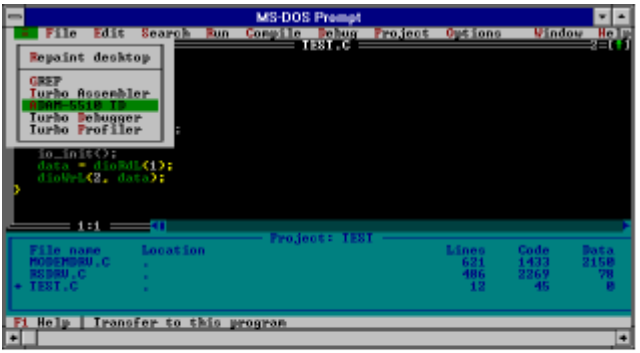


Figure 5-11: *Linking ADAM-5510 for remote debugging*

6

Function Library

Function Library

6.1 Introduction

User-designed ADAM-5510 application programs make use of ADAM-5510 library functions. To make the most efficient use of ADAM-5510's memory space, the ADAM-5510 function library has been separated into six 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 include all necessary ADAM-5510 function libraries in your project file.

6.2 Library Classification

ADAM-5510 has four function libraries, categorized according to usage:

Category A. System Functions: (UTILITY*.LIB)

Category B. Communication Functions: (COMM*.LIB)

Category C. Low Speed I/O Module Access Functions: (LIO*.LIB)

Category D. High Speed I/O Module Access Functions: (HIO*.LIB)

Category E. Counter/Frequency Module Access Functions: (LAI*.LIB)

Category F. Serial Module Access Functions: (A5090*.LIB)

6.3 Libraries Sized for Different Memory Modes

The ADAM-5510 function libraries support four memory modes: SMALL, MEDIUM, COMPACT and LARGE. You can use library files sized according to your memory mode. For example, if you use ***small*** mode you can link UTILITYS.LIB and LIOS.LIB to implement system and low speed I/O module access functions. On the other hand, if you use ***large*** mode, you can link UTILITYL.LIB and LIOL.LIB.

6.4 Index

Category A. System functions (UTILITY*.LIB)

Category B. Communication functions (COMM*.LIB)

checksum()

com_485_deinstall()

com_485_flush_rx()

com_485_flush_tx()

com_485_install()

com_485_rx()

com_485_rx_empty()

com_485_set_format()

com_485_set_speed()

com_485_tx()

com_485_tx_empty()

com_485_tx_string()

com_carrier()

com_clear_break(), com_set_break()

com_clear_local_loopback(),com_set_local_loopback()

com_deinstall()

com_disable_fifo(), com_enable_fifo()

com_flush_rx(), com_flush_tx()

com_pgm_deinstall()

com_pgm_flush_rx()

com_pgm_flush_tx()

com_pgm_install()

com_pgm_rx()

com_pgm_rx_empty()

com_pgm_set_format()

com_pgm_set_speed()

com_pgm_tx()

com_pgm_tx_empty()

com_pgm_tx_string()

com_set_format()

com_get_line_status(), com_set_line_params(), com_get_modem_status()

com_install()

com_lower_dtr(), com_raise_dtr()

com_lower_rts(), com_raise_rts()

com_read_scratch_register(), com_write_scratch_register()

com_rx()

com_rx_empty(), com_tx_empty()

com_set_parity()

com_set_speed()

com_tx()

com_tx_ready()

com_tx_string()

CRC16()

modem_autoanswer()

modem_command_state()

modem_command()

modem_dial()

modem_handup()

modem_initial()

Category C. Low speed I/O module access functions (LIO*.LIB)

AiUpdate()

Get5013()

Get501718()

Get5017H()

GetRange5013()

GetRange501718()

GetRange5017H()

Init5013()

Init501718()

Init5017H()

Category D. High speed I/O module access functions (HIO*.LIB)

Get5050()

Get5051()

Get5052()

Init5024()

Set5024()

Set5050()

Set5056()

Set5060()

Set5068()

Category E. Counter/Frequency Module Access Functions (LAI*.LIB)

Init5080()

Get5080()

Clear_Counter()

Start_Stop_Counter()

ReadOverflowFlag()

SetInitCounterVal()

Category F. Serial module access functions (A5090*.LIB)

int port_install()

int port_deinstalled()

void port_select()

int reset_slot()

void port_reset()

int which_has_been_installed()

void port_set_speed()

void port_set_format()

void port_disable_fifo()

int port_enable_fifo()

int port_carrier()

void port _clear_break()

void port _set_break()

void port _clear_local_loopback()

void port _set_local loopback()

int port_get_line_status()

int port_set_line_params()

int port_get_modem_status()

int port_get_modem_control_status()

int port_set_modem_control_params()

void port_lower_dtr()

void port_raise_dtr()

void port_raise_rts()

void port_lower_rts()

modem_initial_90()

modem_command_90()

void modem_command_state_90()

void modem_autoanswer_90()

void modem_dial_90()

void modem_handup_90()

void port_flush_rx()

void port_flush_tx()

int port_rx_error()

int port_rx_ready()

char port_rx()

int port_tx_empty()

void port_tx()

void port_tx_string()

6.5 Function Library Description

6.5.1 System Utility Library (UTILITY*.LIB)

ADAMdelay

Syntax:
void ADAMdelay(unsigned short msec)

Description:
Delays program operation by a specified number of milliseconds.

Parameter	Description
msec	From 0 to 65535.

Return value:
None.

Example:

```
void main(void)
{
   /* codes placed here by user */
   ADAMdelay(1000);          /* delay 1 sec. */
   /* codes placed here by user */
}
```

Remarks:
None.

EraseSector

Syntax:
unsigned short EraseSector(unsigned long ulBase)

Description:
Erases a 64 KB sector of data in the 256 KB Flash memory

Parameter	Description
ulBase	User-determined address range to be erased, taken from addresses in the range 0x80000L to 0xB0000L.

Return value:

TRUE	Erase successful.
FALSE	Excess address range.

Example:
```
void  main(void)
{
   EraseSector(0x80000L);
}
```

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-5510, from 0 to 3.

Return value:
The return values are:

I/O Module name	Return Value
ADAM-5017	ADAM5017_ID
ADAM-5018	ADAM5018_ID
ADAM-5017H	ADAM5017H_ID
ADAM-5013	ADAM5013_ID
ADAM-5080	ADAM5080_ID
ADAM-5052	ADAM5052_ID
ADAM-5050	ADAM5050_ID
ADAM-5051	ADAM5051_ID
ADAM-5056	ADAM5056_ID
ADAM-5060	ADAM5060_ID
ADAM-5068	ADAM5068_ID
ADAM-5024	ADAM5024_ID

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.

Get_NodeID

Syntax:
unsigned char Get_NodeID(void)

Description:
Gets the DIP switch number of the ADAM-5510 controller.

Parameter Description
None.

Return value:
The DIP switch number of the ADAM-5510 controller.

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

Remarks:
None.

Get_NVRAM_Size

Syntax:
unsigned char Get_NVRAM_Size(void)

Description:
Gets the battery backup RAM size. The unit is sectors, each sector is 4 KB in size. Maximum size is 60 KB theoretically.

Parameter	Description
None.	

Return value:

sector	Number of sectors NV RAM size is set to, from 1 to 15.

Example:
```
void main()
{
unsigned char sector;
sector = Get_NVRAM_Size();
}
```

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.

Get_SysMem

Syntax:
unsigned char Get_SysMem(unsigned char which_byte)

Description:
Reads a byte from security SRAM.

Parameter	Description
which_byte	From 0 to 112, user-determined.

Return value:
The value in a byte of security SRAM.

Example:
```
unsigned char SlotValue[4];
void main(void)
{
        int I;
   /* recover last value */
   for(I=0;I < 4;I++)
              SlotValue[I] = Get_SysMem(I);
}
```

Remarks:
None.

LED_init, LED_OFF, LED_ON

Syntax:

void LED_init(void)
void LED_OFF(int which_led)
void LED_ON(int which_led)

Description:

Turns LED lights on and off. The LED I/O port must be initialized first. It will take a little time for the light to stabilize following the signal for the turning on and turning off of the light.

Parameter	Description
which_led	PWR
	RUN
	COMM

Return value:

None.

Example:

```
void main(void)
{
   LED_init();
   /* flash COMM led */
   while(1) {
         LED_ON(COMM);
         ADAMdelay(500);
         LED_OFF(COMM);
   }
}
```

Remarks:

None.

ProgramByte

Syntax:

unsigned short ProgramByte(unsigned long ulAddress, unsigned char byte)

Description:

Programs a byte of information into the 256 KB Flash memory. This feature supports data-logging or mass information storage.

Parameter	Description
ulAddress	User–determined destination address for byte transfer, taken from the range 0x80000L to 0xBFFFFL.
byte	The value to be saved.

Return value:

TRUE	Successful transfer to Flash memory.
FALSE	Error (destination already occupied, excess address range, or program error).

Example:

```
void main(void)
{
unsigned long FlashAddr=0x80000L;
   programByte(FlashAddr, 0x55);
   programByte(FlashAddr+1, 0xAA);
}
```

Remarks:

None.

ProgramSector

Syntax:

unsigned short ProgramSector(unsigned long ulAddress, unsigned char far *SECTOR_DATA)

Description:

Programs an entire 32 KB sector of data of the global variable, SECTOR_DATA[], into 256 KB Flash memory.

Parameter	Description
ulAddress	User-determined destination address in the Flash memory, taken from addresses in the range 0x80000L to 0xB8000L.
SECTOR_DATA	Pointer at the starting address in the origin memory of the user's data array.

Return value:

TRUE	Successful transfer to Flash memory.
FALSE	Error (destination already occupied, excess address range, or program error).

Example:

```
void main(void)
{
   int i;
for(I=0;I < 32768;I++)
   SECTOR_DATA[I] = 55;
ProgramSector(0x80000L, SECTOR_DATA);
}
```

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;
   data = read_backup_ram(500);
   /* put your codes here */
}
```

Remarks:
None.

read_mem

Syntax:
unsigned char read_mem (int memory_segment , unsigned int i)

Description:
Reads far memory data, 256 KB Flash memory, from 0x80000L to 0xBFFFFL, where (the Absolute Address) = (SEG*16 + OFFSET). For example, (0x800FFL) = (0x8000*16 + 0x00FF).

Parameter	Description
memory_segment	User-determined address taken from the range 0x8000 to 0xBFFF.
i	Offset for use in location of memory taken from the range 0x0000 to 0xFFFF.

Return value:
The value in memory storage at the indicated address.

Example:
```
void main(void)
{
   unsigned char data;
   data = read_mem(0x8000, 0x0000);
   /* put your codes here */
}
```

Remarks:
None.

Release_All

Syntax:
void Release_All()

Description:
Releases all timer resources of the ADAM-5510 system.

Parameter Description
None.

Return value:
None.

Remarks:
None.

Example:
```
void main()
{
   int idx;
   /*– Initializes the timer built into the 80188
        microprocessor –*/
   Timer_Init();

   /*– Sets time interval of the timer to
        1 second. –*/
   idx=Timer_Set(1000);

   /*– Checks whether the timer has timed out –*/
   while(tmArriveCnt[idx]==0)
{
   /*– user can attend to other tasks...–*/
}
   /*– Resets the current timer to its initial
        state. –*/
   Timer_Reset(idx);
```

```
    /*— Releases all timer resources —*/
    Release_All()
}
```

Set_NVRAM_Size

Syntax:
void Set_NVRAM_Size(unsigned char sector)

Description:
Sets the battery backup RAM size. The unit is sectors, each sector is 4 Kbytes in size. Maximum size is 60 KB theoretically.

Parameter	Description
sector	NV RAM size in 4 KB sectors, from 1 to 15 sectors.

Return value:
None.

Example:
```
void main()
{
        Set_NVRAM_Size(31); /* sets NVRAM size to 15 KB*/
}
```

Remarks:
Maximum size is 60 KB theoretically. Actual size available depends on the user's programming.

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.

Set_SysMem

Syntax:
void Set_SysMem(unsigned char which_byte, unsigned char data)

Description:
Writes a byte to security SRAM. Security SRAM supports 113 bytes for user storage of important information.

Parameter	Description
which_byte	From 0 to 112, user determined.
data	Value to be saved.

Return value:
None.

Example:
```
unsigned char data[4] = {1,2,3,4};
void main(void)
{
   int I;
   /* save current value */
   for(I=10;I < 14;I++)
         Set_SysMem(I, data[I-10]);
}
```

Remarks:
None.

Timer_Init()

Syntax:
int Timer_Init()

Description:
Initializes the timer built into the 80188 microprocessor. The return value “0” means the initialization of the time was successful. The return value “1” means the timer had already been initialized.

Parameter Description
None.

Return value:
0: Initialization was successful.

1: The timer had already been initialized.

Remarks:
None.

Timer_Reset

Syntax:
void Timer_Reset(int idx)

Description:
Resets the timer identified by the integer idx to its initial state.

Parameter	Description
idx	Timer index.

Return value:
None.

Remarks:
None.

Timer_Set

Syntax:

int Timer_Set(unsigned int msec)

Description:

Requests a timer function from the microprocessor and then sets the time interval of the function. Timer intervals are set in 5 millisecond increments. The function return value is an integer representing the ID of the timer function when it is successful. A return value "-1" means the request failed. Programmers should consider whether an assigned timer has timed-out when programming for timer functions. The value of the variable tmArriveCnt[idx] can be checked to verify timer status. A value of 0 indicates that the timer is still counting. Values other than 0 mean the timer has timed-out.

Parameter	Description
msec	Time interval set, max. value is 65536.

Return value:

Integer	Function success, value represents function timer ID. Max. value of 100.
-1	Function failure.

Remarks:

Timer function calls in the ADAM-5510 are emulated as timer functions in a PLC. Applications using timer functions will run less efficiently the more timer functions are running simultaneously in a program. Please refer to Example 9 on the utility diskettes for details.

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.

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(void)
{
   int I;
   WDT_enable();
   For(I=0;I < 10;I++)
   {
   ADAMdelay(1000);
   WDT_clear();
   }
WDT_disable();
}
```

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 0 to 61439; 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.

6.5.2 Low Speed I/O Library(LIO*.LIB):

AiUpdate

Syntax:

int AiUpdate(int Board, int *channel)

Description:

Checks whether the data of a low-speed analog input module, such as ADAM-5017, ADAM-5018 and ADAM-5013, is ready to be accessed.

Parameter	Description
int Board	The slot number of an ADAM-5510, from 0 to 3.
int *channel	The return value indicates the channel for which data is ready. Valid value 0 to 7 for ADAM-5017. Valid value 0 to 6 for ADAM-5018. Valid value 0 to 2 for ADAM-5013.

Return value:

int status;	0 : Ready
	-1 : Not ready
	-2 : The hardware of the module failed

Example:

```
void main()
{
   /*— Checks whether the data of the low speed AI
module in slot 0 is ready  —*/
   if( AiUpdate(0, &channel) ==0)
   {
   /*— access data —*/
   }
}
```

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:
```
// An example for Init5013, Get5013 and GetRange5013
 main()
 {
   int *value,*range, i, j;
   / *One ADAM-5013 module on slot 0 of the
      ADAM-5510* /
   printf("Initialize ADAM-5013...\n");
   Init5013(0);
   printf("Get ADAM-5013 Value....\n");
   for (j=0;j<3;j++)
   {
      for (i=0;i<4;i++)
      {
         / *Get ADAM-5013 data and range from channel
```

```
            0 to 2 on slot 0 of ADAM-5510* /
         Get5013(0,j,value);
         if (i==3)
         {
            GetRange5013(0,j,range);
            / *See range index in Appendix C* /
            printf(" ADAM-5013#%d=%d\n",j,*value);
            printf(" (with range is 0x%x",*range);
         }
      }
   }
}
```

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:
```
/ *An example for Init501718, Get501718 and GetRange501718* /

main()
{
int *value,*range, i, j;
/  *One ADAM-5018 (ADAM-5017) module on slot 3 of the ADAM-5510* /
    printf("Init ADAM5018(or ADAM5017)...\n");
    Init501718(3);
    printf("Get ADAM5018(or ADAM5017)....\n");

    for (j=0;j<7;j++)
    {
        for (i=0;i<4;i++)
        {
            / *Get ADAM-5018 data and range from channel 0 to 6 on
```

```
            slot 3 of ADAM-5510 */
          Get501718(3,j,value);
          / *See range index in Appendix C* /
          if (i==3)
          {
              GetRange501718(3,j,range);
              printf(" ADAM-5018(or ADAM5017)#%d=%d\n",j,*value);
              printf(" (with range is 0x%x",*range);
          }
      }
   }
}
```

Remarks:

None.

GetRange5013

Syntax:
void GetRange5013(int Board, int Channel, void *pRange)

Description:
Reads the input range in an ADAM-5013 module.

Parameter	Description
Board	0 – 3 for Slot0 ...Slot3.
Channel	0 – 2 for ADAM-5013.
*pRange	The input range code returned. (See **Appendix C**.)

Return Value:
None.

Remarks:
None.

GetRange501718

Syntax:
void GetRange501718(int Board, int Channel, void *pRange)

Description:
Reads the input range in an ADAM-501718 module.

Parameter	Description
Board	0 – 3 for Slot0 ...Slot3.
Channel	0 – 7 for ADAM-5017, 0-6 for ADAM-5018.
*pRange	The input range code returned (See **Appendix C**.)

Return Value:
*pRange	The input range code returned.

Remarks:
None**.**

Init5013

Syntax:
void Init5013(int Slot)

Description:
Initializes ADAM-5013. Note that ADAM-5013 must be initialized before other commands are issued to it.

Parameter	Description
Slot	From 0 to 3.

Return Value:
None.

Example:
```
void main(void)
{
   int I;
   /* initializes 4 slots RTD modules */
   for (I=0; I < 4; I++)
   Init5013(I);
}
```

Remarks:
None.

Init501718

Syntax:
void Init501718(int Slot)

Description:
Initializes ADAM-5017 or ADAM-5018. Note that ADAM-5017 or ADAM-5018 must be initialized prior to other commands being issued to them.

Parameter	Description
Slot	From 0 to 3.

Return value:
None.

Example:
```
void main(void)
{
   int I;
   /* initializes 4 slots AI module */
   for(I=0;I < 4;I++)
   Init501718(I);
}
```

Remarks:
None.

6.5.3 High Speed I/O Library (HIO*.LIB)

Get5017H

Syntax:

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

Description:

Reads the data value in an ADAM-5017H module.

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

Note: The pValue for ADAM-5017H must be interpreted in reference to the input range that be setup in the module configuration

Return Value:

None.

Example:

```
main()
{
   int  *Value,*range;
   int  Format,Range;
   //   One ADAM-5017H module on slot 1 of the
   ADAM-5510 printf("Init  ADAM5017H...\n");
   Init5017H(1);
   printf("Get  ADAM5017H....\n");

   for  (j=0;j<8;j++)
   {
         for  (i=0;i<4;i++)
```

```
        {
        // Get ADAM-5017H data and range
         from channel 0 to 7 on
        // slot 1 of ADAM-5510

               Get5017H(1,j,value);

               if (i==3)
               {
                      GetRange5017H(1,j,range);
                      // See range index and format
in Appendix C
                      Range = range & 0xFF;
                      Format = (range & 0xFF00)>>8;
                      printf(" ADAM-
                             5017H#%d=%d\n",j,*value);
                      printf(" (with range is 0x%X
and
                      format is 0x%X)",Range,Format);
               }
        }
   }
}
```

Remarks:

None.

Get5050, Get5051, Get5052

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)

Description:

Reads the data value in an I/O module.

Parameter	Description
Board	ADAM-5510 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(void)
{
   unsigned char Bdata;
   unsigned int Wdata;

   Get5051(0, 13, ABit, &Bdata);   /* Slot0, pin13,
data=0 or 1 */
   Get5051(2, 0, AByte, &Bdata);   /* Slot2, pin0~pin7,
Bdata=Low Byte data */
   Get5051(3, 0, AWord, &Wdata); /* Slot3, pin0~pin15,
Wdata=Word data */
```

```
}
```

Remarks:

None.

GetRange5017H

Syntax:
void GetRange5017H(int Board, int Channel, void *pRange)

Description:
Reads the input range in an ADAM-5017H module.

Parameter	Description
Board	0 – 3 for Slot0 ...Slot3.
Chanel	0 – 7 for ADAM-5017H.
*pRange	The input range code returned. (See **Appendix C**.)

Return Value:
None.

Remarks:
None.

Init5017H

Syntax:
void Init5017(int Slot)

Description:
Initializes ADAM-5017H. Note that ADAM-5017H must be initialized before other commands are issued to it.

Parameter	Description
Slot	From 0 to 3.

Return Value:
None.

Example:
```
void main(void)
{
   int I ;      /* initializes 4 slots containing
ADAM-5017H modules*/
for (I=0; I < 4; I++)
Init5017H(I);
}
```

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

Set5050, Set5056, Set5060, Set5068

Syntax:

void Set5050(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-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(void)
{
unsigned char Bitdata = 1;
   Set5056( &Bitdata, 0, 13, ABit);
/* Output 1 to slot 0, pin 13 */
}
```

Remarks:

None.

6.5.4 Communication Library (COMM*.LIB)

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.

com_carrier

Syntax:
int com_carrier(void)

Description:
Detects the carrier signal of COM port.

Parameter Description
None.

Return value:

TRUE	If a carrier is present.
FALSE	No carrier.

Example:
```
void main(void)
{
   if( com_carrier() == TRUE ) {
         /* Telephone carrier signal present at COM
port, put your associate program here */
   }
}
```

Remarks:
None.

com_clear_break, com_set_break

Syntax:

void com_clear_break(unsigned baseaddr)
void com_set_break(unsigned baseaddr)

Description:

Sets COM port to clear BREAK signal.

Sets COM port to send BREAK signal.

Parameter	Description
baseaddr	The UART address, COM1=0x3F8, COM2=0x2F8.

Return value:

None.

Example:

None.

Remarks:

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

com_clear_local_loopback, com_set_local_loopback

Syntax:

void com_clear_local_loopback(unsigned baseaddr)
void com_set_local_loopback(unsigned baseaddr)

Description:

Sets COM port to disable loopback function for diagnostic.

Sets COM port to enable loopback function for diagnostic.

Parameter	Description
baseaddr	The UART address, COM1=0x3F8, COM2=0x2F8.

Return value:

None.

Example:

None.

Remarks:

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

com_deinstall

Syntax:
void com_deinstall(void)

Description:
Uninstalls the communications drivers completely, without changing the baud rate or DTR.

Parameter Description
None.

Return value:
None.

Example:
```
void main(void)
{
   /* codes placed here by user */
   com_deinstall();
}
```

Remarks:
This function MUST be called before returning to DOS, so the interrupt vector will not point to our driver anymore.

com_disable_fifo, com_enable_fifo

Syntax:

void com_disable_fifo(unsigned baseaddr)
int com_enable_fifo(unsigned baseaddr, unsigned triggerlevel)

Description:

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

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

Parameter	Description
Baseaddr	The UART address, COM1=0x3F8, COM2=0x2F8.
Triggerlevel	1, 4, 8, 14.

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_flush_rx, com_flush_tx

Syntax:

void com_flush_rx(void)

void com_flush_tx(void)

Description:

Buffer flushers. Initializes the transmit and receive queues (respectively) to their empty state.

Parameter Description

None.

Return value:

None.

Example:

```
void main(void)
{
   com_flush_tx();
   com_flush_rx();
}
```

Remarks:

None.

com_get_line_status, com_set_line_params, com_get_modem_status

Syntax:

int com_get_line_status(unsigned baseaddr)
int com_set_line_params(unsigned baseaddr, unsigned lineparams)
int com_get_modem_status(unsigned baseaddr)

Description:

Reads from COM port line control register.

Writes to COM port line control register.

Reads from COM port modem status register.

Parameter	Description
baseaddr	The UART address, COM1=0x3F8, COM2=0x2F8.
lineparams	Please refer to the UART specifications.

Return value:

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

Example:

None.

Remarks:

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

com_install

Syntax:
int com_install(int portnum);

Description:
Installs the communications drivers.

Parameter	Description
int portnum;	Desired port number, always 1 for ADAM-5510.

Return value:
int status;

0 = Successful installation.

1 = Drivers already installed.

2 = Invalid port number.

3 = No UART for specified port.

Example:
```
void main(void)
{
status = com_install(1);     /* COM1 */
   if( status == 0 ) printf("\n COM1 install OK!");
   else exit(0);
}
```

Remarks:
None.

com_lower_dtr, com_raise_dtr

Syntax:

void com_lower_dtr(void)

void com_raise_dtr(void)

Description:

Sets COM port to DTR for low signal.

Sets COM 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_lower_rts, com_raise_rts

Syntax:

void com_lower_rts(unsigned baseaddr)

void com_raise_rts(unsigned baseaddr)

Description:

Sets COM port to RTS for low signal.

Sets COM port to RTS for high signal.

Parameter	Description
baseaddr	The UART address, COM1=0x3F8, COM2=0x2F8.

Return value:

None.

Example:

```
#define COM1   0x3F8
#define COM2   0x2F8
void main(void)
{
   com_lower_rts(COM1);      /* handshaking with
external serial device */
   ADAMdelay(500);
   com_raise_rts(COM1);      /* generates a signal of
500 ms low trigger */
}
```

Remarks:

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

com_read_scratch_register, com_write_scratch_register

Syntax:

int com_read_scratch_register(unsigned baseaddr)

void com_write_scratch_register(unsigned baseaddr, int value)

Description:

Reads from COM port scratch register.

Writes to COM port scratch register.

Parameter	Description
baseaddr	The UART address, COM1=0x3F8, COM2=0x2F8.
value	Integer value one byte in length, assigned by user from the range 0 to FF.

Return value:

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

Example:

None.

Remarks:

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

com_set_format

Syntax:
void com_set_format(int data_length, int parity, int stop_bit)

Description:
Sets the parameters for data length, parity and stop bits for the COM1 port.

Parameter	Description
data_length	Valid range 5 to 8 bits for 1 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 data format of the COM1 port to 8-bit data length, no
       parity, 1 stop bit*/
    com_set_format(8, 0, 1);
}
```

Remarks:
None.

com_set_parity

Syntax:
void com_set_parity(enum par_code parity, int stop_bits);

Description:
Sets the parity and stop bits.

Parameter	Description
int code;	COM_NONE = 8 data bits, no parity COM_EVEN = 7 data bits, even parity COM_ODD = 7 data bits, odd parity COM_ZERO = 7 data bits, parity bit = zero COM_ONE = 7 data bits, parity bit = one
int stop_bits;	Must be 1 or 2.

Return value:
None.

Example:
```
void main(void)
{
   com_set_parity(COM_NONE, 1);     /* set N, 8, 1 */
}
```

Remarks:
None.

com_set_speed

Syntax:
void com_set_speed(unsigned long speed);

Description:
Sets the baud rate of the COM port.

Parameter	Description
speed	The baud rate value.

Return value:
None.

Example:
```
void main(void)
{
   com_set_speed(9600L);
   /* set baud rate = 9600 bps */
}
```

Remarks:
None.

com_rx

Syntax:
char com_rx(void)

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

Parameter	Description
None.	

Return value:
c	The returned character.

Example:
```
void main(void)
{
   unsigned char COMdata;
   COMdata = com_rx();
}
```

Remarks:
None.

com_tx

Syntax:
void com_tx(char c)

Description:
com_tx() sends a single character by waiting until the transmit buffer isn't full, then putting the character into it. The interrupt driver will then send the character once it is at the head of the transmit queue and a transmit interrupt occurs.

Parameter	Description
c	The value you would like to send.

Return value:
None.

Example:
```
void  main(void)
{
   com_tx(0x02);
   com_tx(0x03);
}
```

Remarks:
None.

com_rx_empty, com_tx_empty

Syntax:

int com_rx_empty(void)
int com_tx_empty(void)

Description:

Small routines to return status of the transmit and receive queues.

Parameter Description

None.

Return value:

Com_rx_empty(void) returns TRUE if the receive queue is empty.

Com_tx_empty(void) returns TRUE if the transmit queue is empty.

Example:

```
void  main(void)
{
   unsigned  char  data;
   if(  com_rx_empty()  ==  FALSE)  data=com_rx();
}
```

Remarks:

None.

com_tx_string

Syntax:
void com_tx_string(char *s)

Description:
com_tx_string() sends a string by repeatedly calling com_tx().

Parameter	Description
s	The string you would like to send.

Return value:
None.

Example:
```
unsigned char name[]="ADAM5510";
void main(void)
{
   com_tx_string(name);
}
```

Remarks:
None.

com_485_deinstall

Syntax:
void com_485_deinstall(void)

Description:
Releases the interrupt register of the microprocessor for use by the RS-485 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-485 port */
   com_485_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_485_flush_rx(), com_485_flush_tx()

Syntax:
void com_485_flush_rx(void)
void com_485_flush_tx(void)

Description:
COM2 (RS-485) buffer flusher. Initializes the transmitting and receiving queues to their empty states.

Parameter Description
None.

Return value:
None.

Example:
```
void main()
{
com_485_flush_rx();
com_485_flush_tx();
}
```

Remarks:
The COM2 (RS-485) transmitter uses polling-action (not interrupt-action). Its buffer is always flushed.

com_485_install

Syntax:
int com_485_install(void)

Description:
Allocates the interrupt registers of the microprocessor for use by the RS-485 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_485_install();
   if( status ==0)
   printf("\n The allocation of COM2 port (RS-485) is
         OK !");
   else
   exit(0);
}
```

Remarks
None.

com_485_rx

Syntax:
char com_485_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 C485data;
   C485data=com_485_rx();
}
```

Remarks:
None.

com_485_set_format

Syntax:
void com_485_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-485 port to 8-bit
data length, no parity, 1 stop bit*/
   com_485_set_format(8, 0, 1);
}
```

Remarks:
None.

com_485_set_speed

Syntax:
void com_485_set_speed(unsigned long speed)

Description:
Sets the baud rate of the RS-485 port.

Parameter	Description
speed	The baud rate value.

Return value:
None.

Example:
```
void main()
{
   com_485_set_speed(9600L);/*Sets the baud rate of
the RS-485 port to 9600 bps */
}
```

Remarks:
None.

com_485_rx_empty(), com_485_tx_empty()

Syntax:

int com_485_rx_empty(void)

int com_485_tx_empty(void)

Description:

Returns the status of the COM2 (RS-485) transmitting and receiving queues.

Parameter Description

None.

Return value:

Com_485_rx_empty() returns “TRUE” if the receiving queue is empty.

Com_485_tx_empty() returns “TRUE” if the transmitting queue is empty.

Example:

```
void main()
{
unsigned char data;
if( com_485_rx_empty()== FALSE)
data =com_485_rx();
}
```

Remarks:

The COM2 (RS-485) transmitter uses polling-action (not interrupt-action). Its queue is always empty.

com_485_tx

Syntax:
void com_485_tx(char c)

Description:
This function sends a single character to the Tx pin of the RS-485 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_485_tx(0x03);
   com_485_tx('$');
}
```

Remarks:
None.

com_485_tx_string

Syntax:
void com_485_tx_string(char *s)

Description:
com_485_tx_string() sends a string by calling com_485_tx() repeatedly.

Parameter	Description
s	The string you would like to send.

Return value:
None.

Example:
```
void main()
{
   com_485_tx_string("This is a string test.");
}
```

Remarks:
None.

com_pgm_deinstall

Syntax:
void com_pgm_deinstall(void)

Description:
Releases the interrupt registers of the microprocessor for use by the programming port without changing the baud rate or DTR.

Parameter Description
None.

Return value:
None.

Example:
```
void main()
{
    _
    _
   /* There are some codes before such a function call
*/
   com_pgm_deinstall();
}
```

Remarks:
The programming port is normally used for downloading control programs to the ADAM-5510 using the ADAM-5510 utility. The programming port can be used as an additional communication port if the users have such a requirement. NOTE: The user MUST reset the ADAM-5510 before he uses the port for program downloading again.

com_pgm_flush_rx(), com_pgm_flush_tx()

Syntax:

void com_pgm_flush_rx()
void com_pgm_flush_tx()

Description:

COM3 (Programming port) buffer flusher. Initializes the transmit and receive queues to their empty states.

Parameter Description

None.

Return value:

None.

Example:

```
void main()
{
com_pgm_flush_rx();
com_pgm_flush_tx();
}
```

Remarks:

The COM3 (programming port) transmitter uses polling-action (not interrupt-action). Its buffer is always flushed.

com_pgm_install

Syntax:
int com_pgm_install(void)

Description:
Allocates the interrupt registers of the microprocessor for use by the programming port (COM3) and sets the interrupt vector to the interrupt service routine.

Parameter Description
None.

Return value:
int status: 0 = Successful installation

1 = Drivers are already installed

Example:
```
void main()
{
   int status;
   status = com_pgm_install();
   if( status ==0)
   printf("\n Programming port has been installed
successfully !");
   else
   exit(0);
}
```

Remarks:
None.

com_pgm_rx

Syntax:
char com_pgm_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 CPGMdata;
   CPGMdata=com_pgm_rx();
}
```

Remarks:
None.

com_pgm_rx_empty(), com_pgm_tx_empty()

Syntax:

int com_pgm_rx_empty(void)
int com_pgm_tx_empty(void)

Description:

Returns the status of the COM3 (Programming port) transmitting and receiving queues.

Parameter Description

None.

Return value:

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

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

Example:

```
void main()
{
unsigned char data;
if( com_pgm_rx_empty()== FALSE)
data =com_pgm_rx();
}
```

Remarks:

The COM3 (programming port) transmitter uses polling-action (not interrupt-action). Its queue is always empty.

com_pgm_set_format

Syntax:
void com_pgm_set_format(int data_length, int parity, int stop_bit)

Description:
Sets the parameters data length, parity and stop bits of the programming port.

Parameter	Description
data_length	Valid ranges: 7 or 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 programming port to
8-bit data length, no parity, 1 stop bit*/
   com_pgm_set_format(8, 0, 1);
}
```

Remarks:
None.

com_pgm_set_speed

Syntax:
void com_pgm_set_speed(unsigned long speed)

Description:
Sets the baud rate of the programming port (COM3).

Parameter	Description
speed	The baud rate value.

Return value:
None.

Example:
```
void main()
{
   com_pgm_set_speed(9600L);
   /* Sets the baud rate of the programming port to
9600 bps */
}
```

Remarks:
We suggest that users set the baud rate of the programming port below 57600 bps (included) because the programming port UART chip is not a standard UART chip.

com_pgm_tx

Syntax:
void com_pgm_tx(char c)

Description:
This function sends a single character to the Tx pin of the programming 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_pgm_tx(0x03);
   com_pgm_tx('$');
}
```

Remarks:
None.

com_pgm_tx_string

Syntax:
void com_pgm_tx_string(char *s)

Description:
com_pgm_tx_string() sends a string by calling com_pgm_tx() repeatedly.

Parameter	Description
s	The string you would like to send.

Return value:
None.

Example:
```
void main()
{
   com_pgm_tx_string("This is a string test.");
}
```

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(void)
{
unsigned int code;
code = CRC16(String, strlen(String));
printf("\n The string %s CRC16 code = %d", String,
Code);
}
```

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(void)
{
   modem_autoanswer();
   /* waiting phone call */
}
```

Remarks:
None.

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

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(void)
{
   /* receiving data from modem, so modem is in trans-
fer 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 modem to dial.

Return value:
None.

Example:
```
void main(void)
{
   /* COM port and modem initial OK */
   modem_dial("886222184567");
   /* waiting to link */
}
```

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

Remarks:
None.

modem_initial

Syntax:
void modem_initial(void)

Description:
Sets modem to initial status. Due to the ADAM5510 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(void)
{
   /* you need to initialize COM1 */
   modem_initial();
   /* put your modem function… */
}
```

Remarks:
None.

6.5.5 Counter/Frequency Library (LIA*.LIB)

Name:
Initial Slot

Description:
Initial ADAM-5080 Module

Syntax:
void Init5080(int slotno)

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);
}
```

Name:
Get Value

Description:
Get Value from specific channel in ADAM-5080

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

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]));
}
```

Name:
Reset Counter

Description:
Reset the current counter value to its initial value

Syntax:
int Clear_Counter(int slotno, int channel)

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);
}
```

Name:
Start/Stop Counter

Description:
Start or stop the specific counter

Syntax:
int Stop_Start_Counter(int slotno, int channel, StartOrStop)

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");
}
```

Name:
Check Overflow

Description:
Check if counter value reach max. count limit

Syntax:
void ReadOverflowFlag(int slotno, char *pValue)

Parameter	Description
slotno	The specific slot inserted with ADAM-5080 0-3 or slot0-slot3
*pValue	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]);
 }
```

Name:
Set Initial Value

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

Syntax:
int SetInitCounterVal(int slotno, int channel, unsigned long 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 ()
{
   unsigned long int i

   i=1000

   //set 1000 to the initial counter value
   SetInitCounterVal(0,0,i)
}
```

6.5.6 Serial I/O Library (A5090*.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 6-1: *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)
  {
}
```

A

Quick Start Example

Quick Start Example

This chapter provides guidelines to what is needed to set up and install a standalone ADAM-5510 control system.

A.1 ADAM-5510 System Requirements

Before you setup an ADAM-5510 system, you can follow the steps to install I/O modules into the ADAM-5510 base.

1. Align the module with the grooves on the top and bottom of the base.
2. Push the unit straight into the base until it is firmly seated in the backplane connector.
3. Push-in the retaining clips at the top and bottom of the unit to secure the module to the base.

The following list gives an overview of what is needed to setup, install and configure an ADAM-5510 environment.

- ADAM-5510 Main Unit and I/O Modules
- An IBM PC/AT compatible computer with a standard RS-232 port.
- Power supply for the ADAM-5510 system (+10 to +30 V_{DC})
- ADAM-5510 utility diskettes
- Straight-through and crossover DB-9 cables

Host computer

Any computer that is IBM PC/AT compatible which can run and write programs in an MS-DOS environment and provides an RS-232 communication port to allow users to download programs into the ADAM-5510.

Power supply

For ease of use in industrial environments, the ADAM-5510 is designed to accept industry standard +24V_{DC} unregulated power. Operation is guaranteed when using any power supply voltage between +10 and +30V_{DC}. Power ripples must be limited to 200 mV peak to peak while the voltage in all cases must be maintained between +10 and +30 V_{DC}.

Utility software

A menu-driven utility software disk is provided with ADAM-5510 to help users to download programs to the ADAM-5510. The ADAM-5510 utility software is executed in an MS-DOS or compatible environment.

A.2 Basic Steps to Install ADAM-5510 System

The following steps explain how to build a successful ADAM-5510 application.

1. Configure the I/O modules that you want to implement in the applications.(See Chapter 4)
2. Write and simulate control logic on host PC.
3. Convert the programs into 80188 or 80186 compatible codes. The floating operation must be set to emulation mode. Then compile the application programs.
4. Transfer program code into the SRAM (D drive) of ADAM-5510. Then execute it to check if it works as intended.
5. If the program works as intended, download the program into the flash ROM (C drive) of ADAM-5510.
6. Re-power on the ADAM-5510 system to let it standalone operates.

Note: For more details on Steps 2 through 6, see Chapter 5.

This page intentionally left blank

B

COM Port Register Structure

Register Structure

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

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.
The two 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)

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

C

Data Formats and I/O Ranges

Data Formats and I/O Ranges

C.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)
```

C.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	

C.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 1000° C	Engineering Units	+1000.0	+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	

C.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.*

C.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>

C.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

C.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

D

Examples on Utility Diskettes

Examples on Utility Diskettes

Nine examples are included on the two ADAM-5510 utility diskettes. After you install the utility diskettes on your host PC, these examples will be located in the directory **C:\5510\SOURCE**. The following list describes these examples.

Example 1 (Ex1.prj)

ADAM-5510 reads data from low speed analog input modules, then shows the data on screen with the help of the ADAM-5510 utility.

Example 2 (Ex2.prj)

ADAM-5510 sets the output values of some analog output and digital output modules. Then it gets data from some analog input and digital input modules and shows the data on screen with the help of the ADAM-5510 utility.

Example 3 (Ex3.prj)

This example demonstrates the ability of the ADAM-5510 to communicate with other ADAM products, such as the ADAM-5000/485 system or ADAM-4000 series modules. In this example session, the host PC runs ADAM.EXE and acts as the master, while ADAM-5510 acts as the slave.

Example 4 (Ex4.prj)

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

Example 5 (Ex5.prj)

In this example, the user's RAM, RTC and battery-backup RAM are combined to implement a data logger. This data logger can store the time stamp of each piece of data recorded.

Example 6 (Scanio.prj)

This example scans all slots in an ADAM-5510 and then shows the status of any I/O modules located in the slots. It also demonstrates on/off control of the LEDs on the ADAM-5510, and reads RTC (real time clock) time.

Example 7 (Ex7.prj)

ADAM-5017H is a new released AI module in the ADAM-5000 series. It has a much higher sampling rate than its predecessor, ADAM-5017, as demonstrated by this example.

Example 8 (Ex8.prj)

ADAM-5510 allows users to adjust the size of the battery-backup SRAM. This example shows the way to use related functions.

Example 9 (Ex9.prj)

The ADAM-5510 utility contains three functions allowing users to access the timer in the ADAM-5510. This example demonstrates these functions.

Example 10 (Ex10.prj)

This example tells user how to read the counter value from the ADAM-5080, and how to show the data on the screen with ADAM-5510 Utility.

Example 11 (Ex11.prj)

This example provides user a testing tool to verify the ADAM-5090 module. Select any two ports on the module and connected by port-to-port cable, the program can detect whether the ports are all right.

Example 12 (Ex12.prj)

The ADAM-5090 allows a direct link to PC. This example provides internal & external loopback test function, and helps user to verify the ADAM-5090 module and the COM port of PC.

Example 13 (Ex13.prj)

Using ADAM-5090 COM port and ADAM-4520 (RS-232 to RS-422/485 converter) to scan ADAM-4000/5000 series module as remote function. User can also send command to the existing module to get response.

Example 14 (Ex14.prj)

This example tests the modem device with the basic connecting function. Users can use this program to connect to other device through a modem, and allows other modem to connect to it. It can also call a pager and leave a number message with it.

E

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.

E.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.

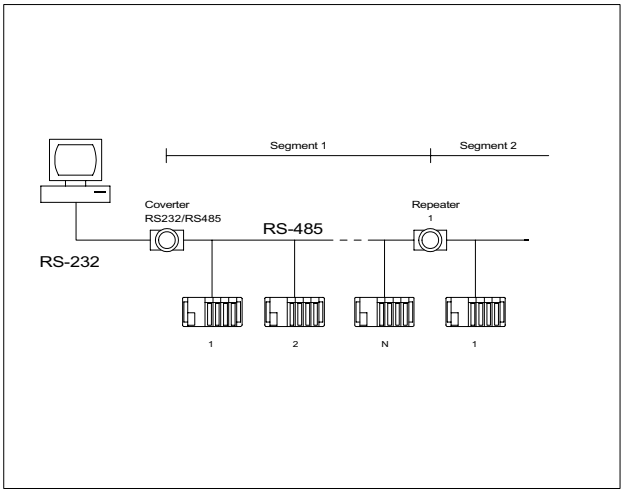


Figure E-1: *Daisychaining*

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.

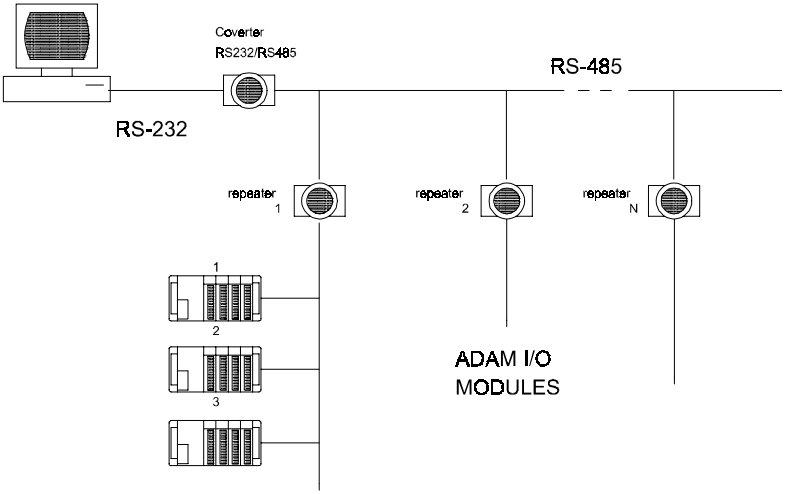


Figure E-2: *Star structure*

Random

This is a combination of daisychain and hierarchical structure.

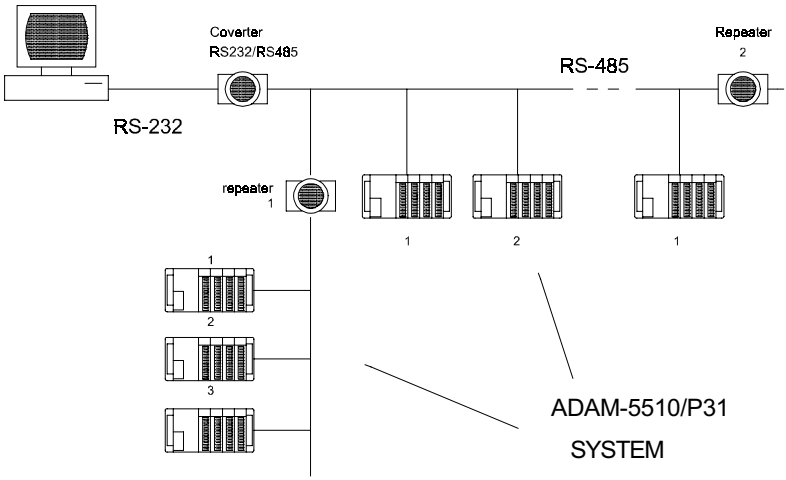


Figure E-3: *Random structure*

E.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.

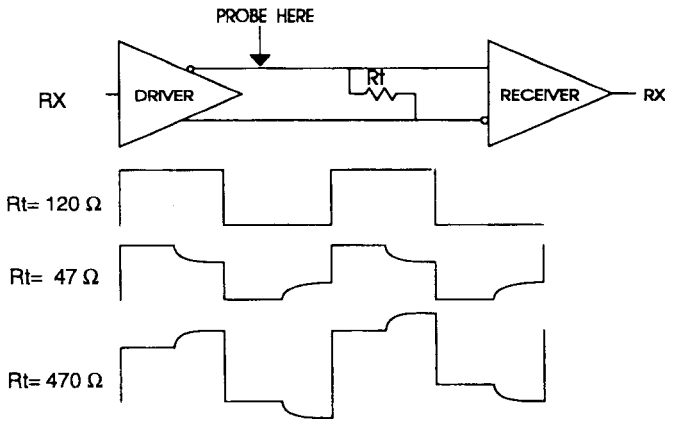


Figure E-4: *Signal distortion*

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).

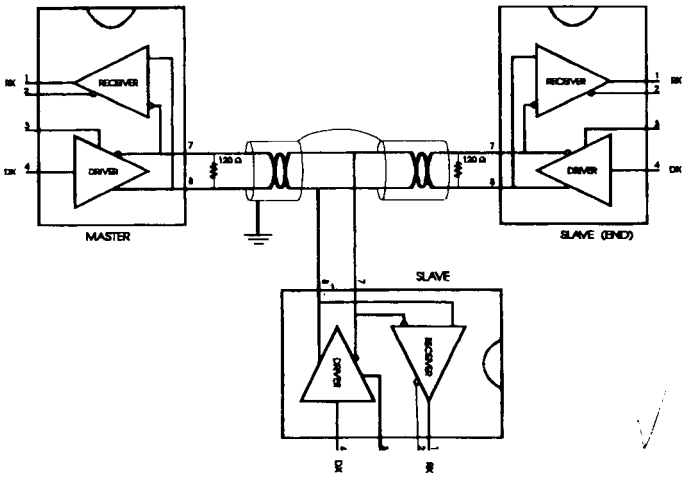


Figure E-5: *Termination resistor locations*

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.

E.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.

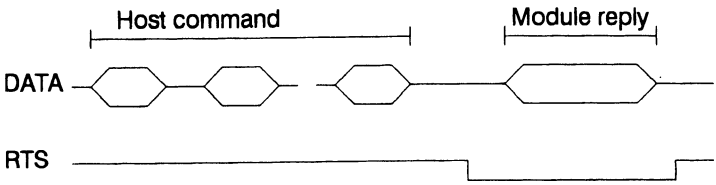


Figure E-6: *RS-485 data flow control with RTS*

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.

F

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

F.1 Grounding

1.1 The 'Earth' for reference

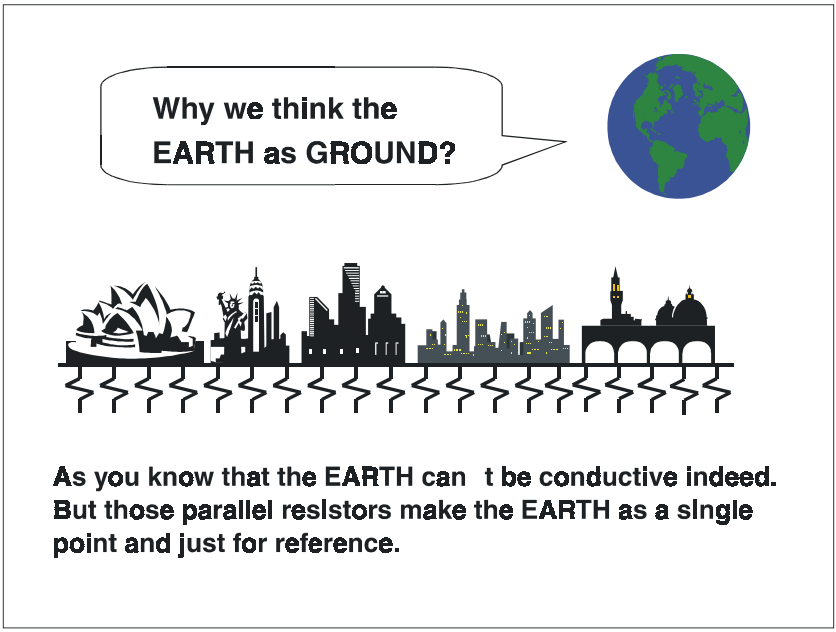
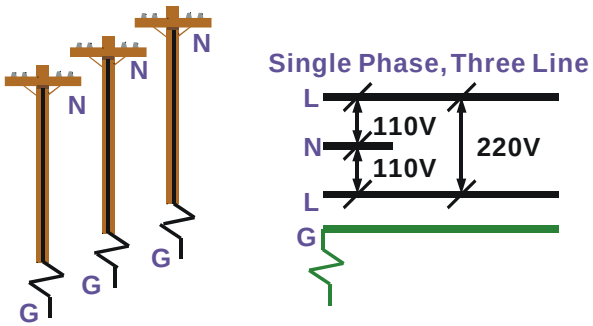


Figure F-1: Think the EARTH as GROUND.

- Why we 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 point for reference.

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 F-2: Grounding Bar.

According to previous description, 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 true ground!

1.3 Normal Mode and Common Mode

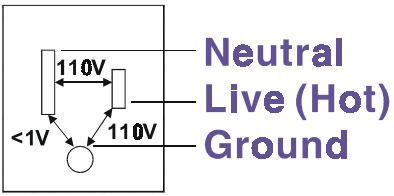


Figure F-3: Normal mode and Common mode.

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.

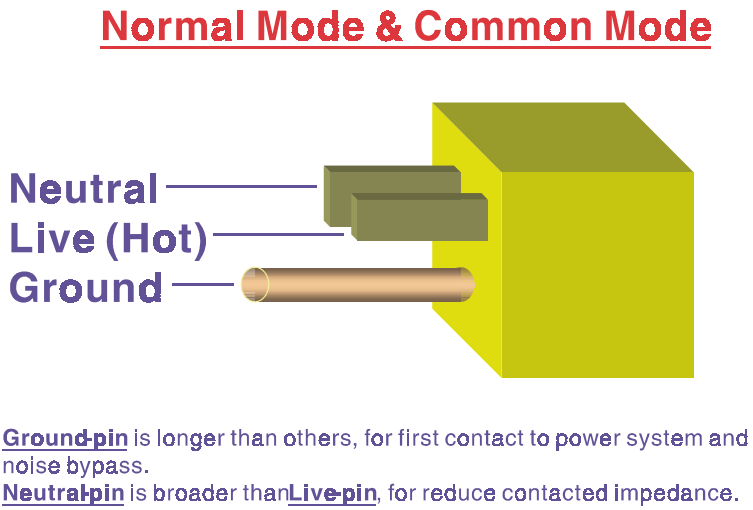


Figure F-4: Normal mode and Common mode.

- 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

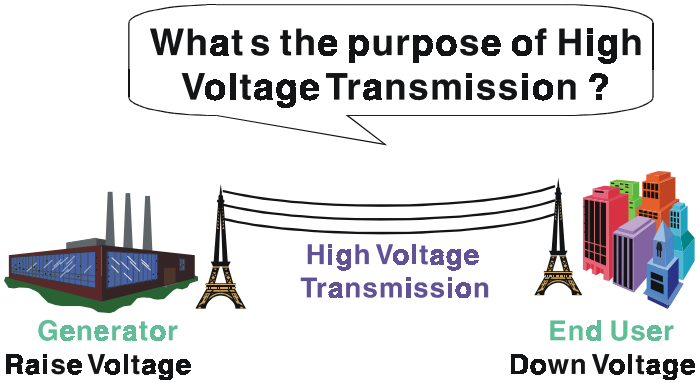


Figure F-5: The purpose of high voltage transmission

- 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.

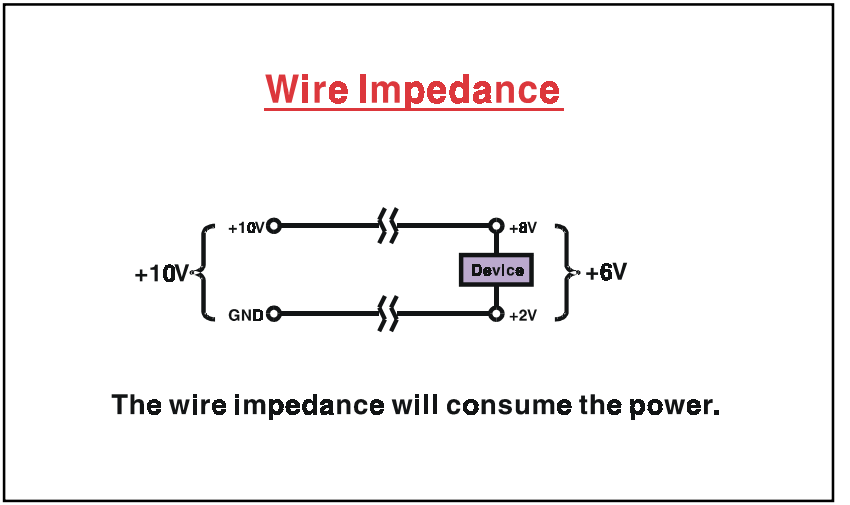


Figure F-6: wire impedance.

1.5 Single Point Grounding

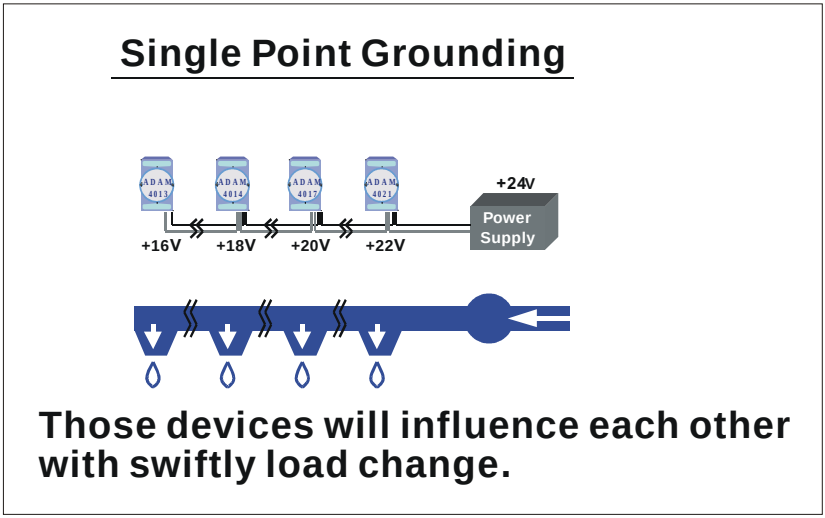


Figure F-7: Single point grounding. (1)

- 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.

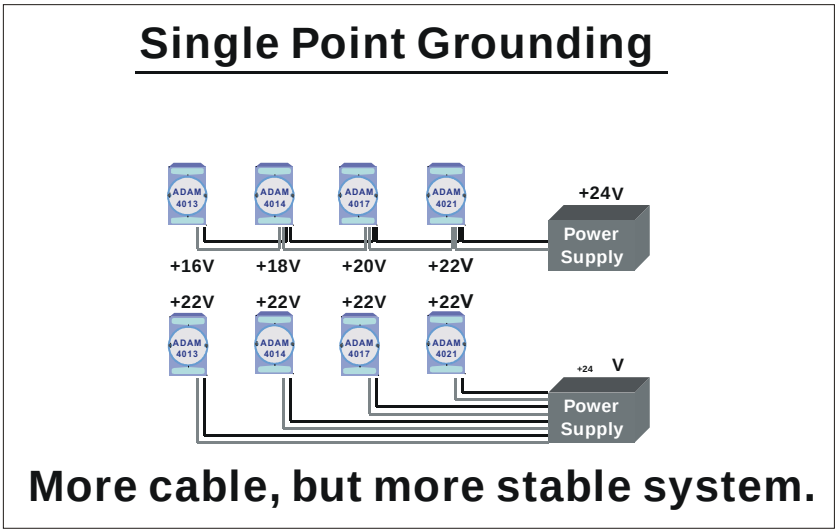


Figure F-8: Single point grounding. (2)

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.

F.2 Shielding

2.1 Cable Shield

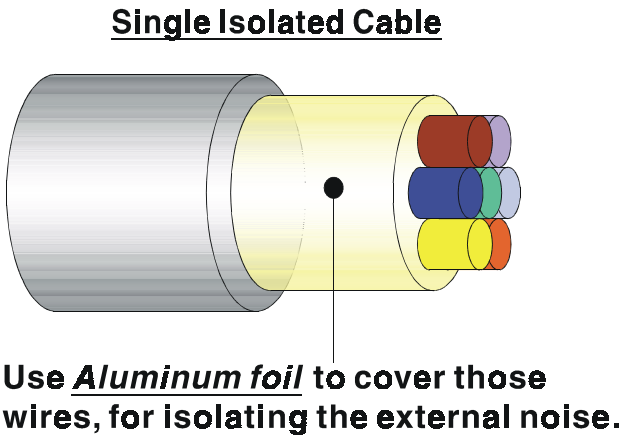


Figure F-9: Single isolated cable

- 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.

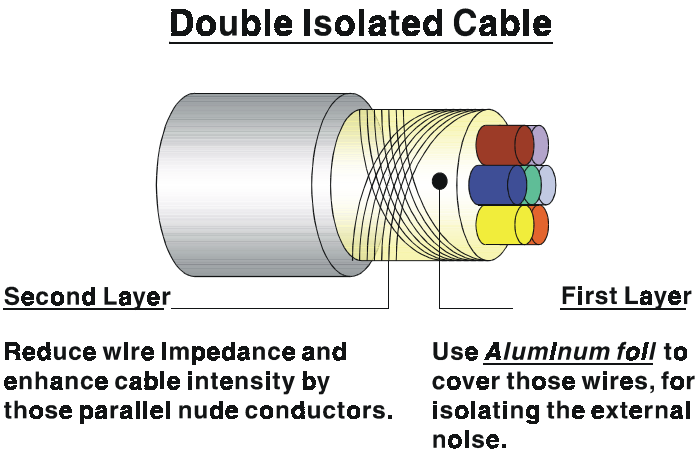


Figure F-10: Double isolated cable

- 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

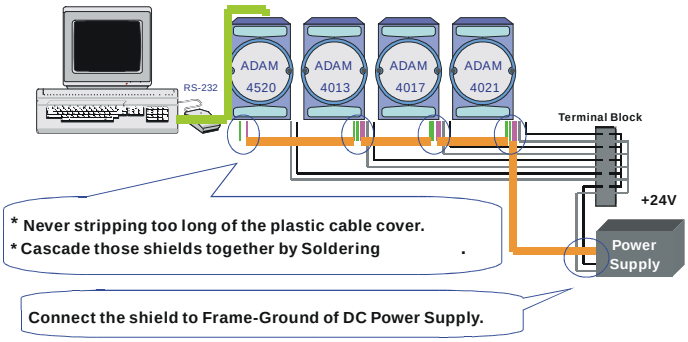


Figure F-11: System Shielding

- 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)

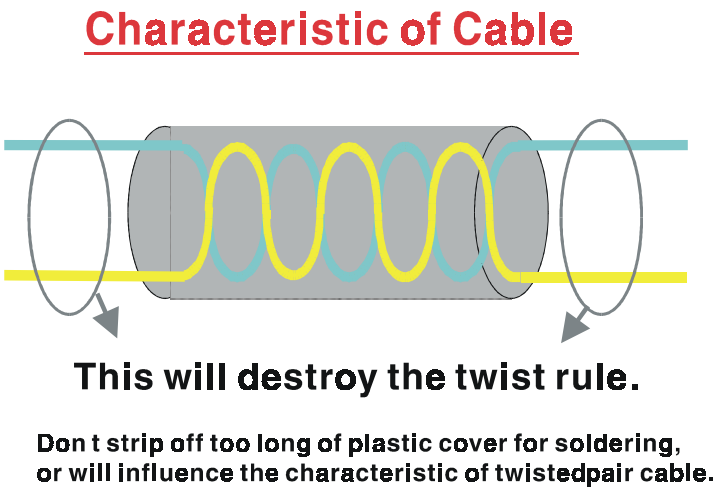
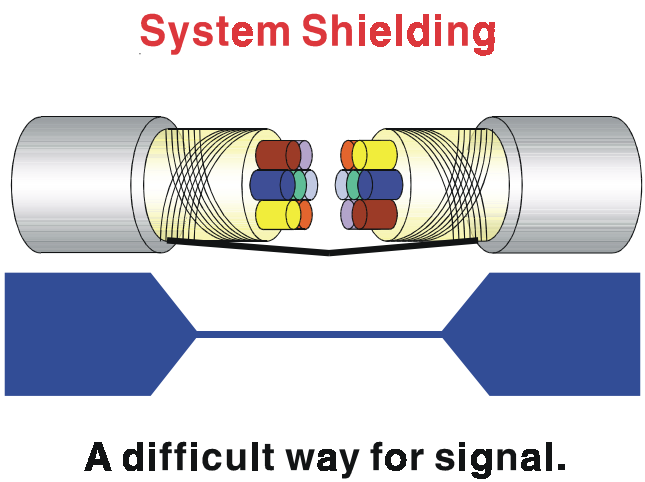


Figure F-12: The characteristic of the cable

- 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.



FigureG-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.

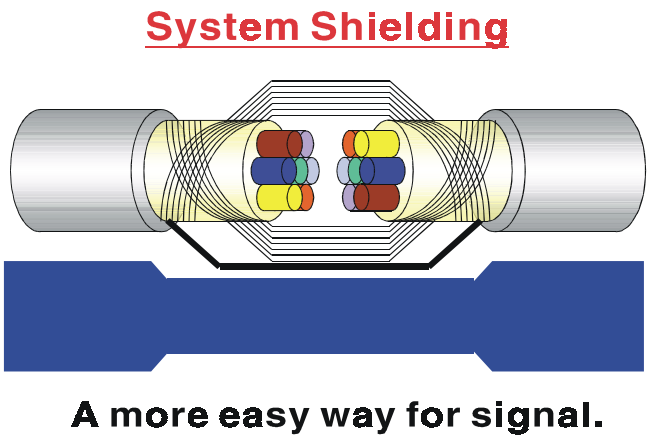


Figure F-14:System Shielding (2)

- Shield connection (2)

The previous diagram shows you that the fill soldering just makes an easier way for the signal.

F.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.

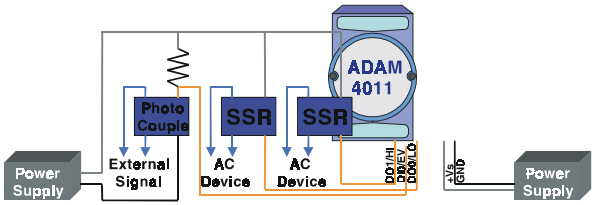


Figure F-15: Noise Reduction Techniques

F.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?