基于 PC 的可编程控制器

ADAM－5510

北京研华客户服务部

1999.7.5

目　　录

第一章　介　　绍 3

1.1 概述 3
1.2 系统配置 4
2.1 基本步骤 5
2.2 模块安装 5
2.3 I/O 槽与 I/O 通道编号 5
2.4 安装 5
2.5 跳线设置和 DIP 开关设置 5
2.6 布线与连接 5

第三章　　ADAM5510 系统 5

3.1 概述 5
3.2 主要特点 5
3.3 ADAM5510 系统技术参数 5

第四章　　编程与下装 5

4.1 编程 5
4.2 下装与传输 5

第五章　　I/O 模块 5

5.1　　模拟量输入模块 5
5.2　　模拟量输出模块 5
5.3 数字量输出/输入模块 5
5.4 继电器输出模快 5
5.5 I/O 模块的设置 5
5.6 模拟量模块的标定 5

第六章　　ADAM-5510 与组态软件的连接 5

第七章　　ADAM-5510 A1 版函数库 5

第八章　　ADAM-5510 A2 版函数库 5

第一章　介　　绍

1.1 概述

ADAM5510 是基于 PC 的可独立完成数据采集与控制的可编程控制器。采用坚固的工业级塑封外壳可确保系统在恶劣的工业环境中可靠运行。ADAM5510 具备通用编程功能，可用于信号的程控放大、模拟 I/O、数字 I/O 及通讯参数的设定。

模拟开放式 PC 环境

ADAM5510 类似于一台紧凑式计算机，包括 80188CPU、256KB FLASH ROM、256KB SRAM、COM1、COM2 及一个编程端口，内置的 ROM-DOS 兼容 MS-DOS 操作系统，提供了除 BIOS 之外的基本 MS-DOS 功能调用，允许运行用 C 或 C++等高级语言编译的应用程序。除此之外，ADAM5510 还提供 ROM 及 RAM 空间，供程序下载及程序运行。任何程序在下载之前，应转化为 80186 或 80188 兼容性代码，下载实用程序内附通信软件。

内置 RS232/485 通信口

为便于同其它设备进行通信，ADAM5510 设置了两个串行通信口 COM1 和 COM2。COM1 固定在 RS232 方式，COM2 固定于 RS485 方式。这种独特的设计方式使 ADAM5510 可适用于多种应用场合。

内置三路隔离保护功能

各种干扰、电气躁声能通过各种途径如 I/O 通道、电源、通信端等干扰系统的正常运行。

ADAM5510 提供输入/输出（3000VDC）、通信（2500VDC）、电源（3000VDC）隔离功能，可阻绝地线电流，减少电气躁声对系统的影响，保护系统不受高压和放电所引起的浪涌电流冲击。

内置实时钟和看门狗定时器

ADAM5510 也包括实时钟和看门狗定时器功能。实时钟可实时记录事件

发生的时刻；而当系统死机时看门狗定时器可重置处理器，这不仅减少了系统维护工作，而且使 ADAM5510 可应用于系统稳定性要求较高的场合。

完整的 I/O 模块及库函数支持

为便于 I/O 模块的拆装，ADAM5510 采纳了底板结构。ADAM5510 具有完整的 I/O 模块供用户选择：所有数字模块均支持 10~30VDC 电压输出和继电器输出；所有模拟量模块均提供 16 位分辨率模入和模出，同时可编程设定输入范围。

为节省用户开发成本，ADAM5510 随机提供了完整的 C 语言库函数。用户用 C 语言在 MSC6.0 和 BC2.0 环境下可方便地调用这些子程序，来执行 ADAM5510 的所有 I/O 功能。

1.2 系统配置

图 1-1 表示了 ADAM5510 的配置。

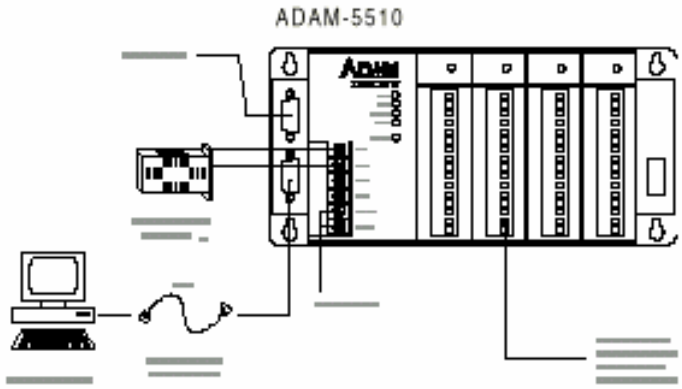


图 1-1　ADAM5510 系统配置

第二章　安装指南

本章讲述了如何安装与设置 ADAM5510，为您在使用前如何配置系统提供了一个快速参考指南。

2.1 基本步骤

第一步：系统需求

在启动 ADAM5510 之前，应准备好如下设备：

- ✧ ADAM5510 主机;
- ✧ 带有标准 RS232 接口的 IBM PC/AT 及兼容机;
- ✧ ADAM5510 供电电源;
- ✧ ADAM5510 下装应用软件.

第二步：连接电源线和下装线

- ✧ 将电源线连接至 ADAM5510，确保电源电压在+10~+30VDC 之间。ADAM5510 螺丝端子+VS 和 GND 用于电源连接;
- ✧ 用标准的 DB-9 针电缆线连接主机的 RS232 口和 ADAM5510 编程口。见图 2-1。

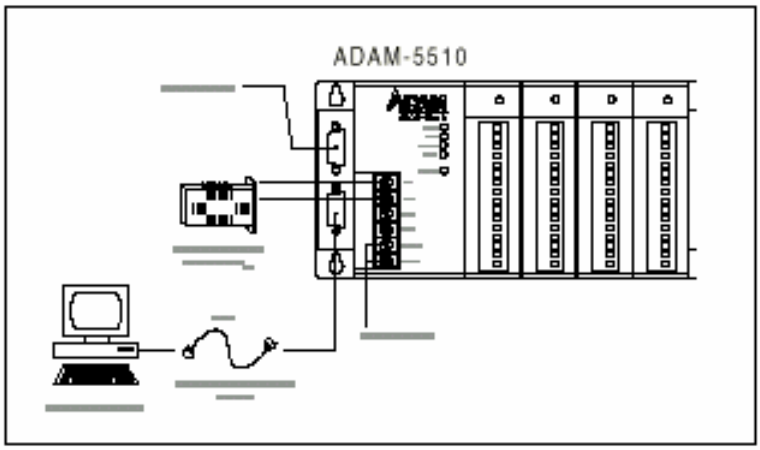


图 2-1　　ADAM5510 布线与连接

第三步：在主机上运行应用软件

用磁盘上提供了用户程序 ADAM5510.EXE，该程序采用菜单驱动方式，用于下装用户程序。该程序主菜单如图 2-2 所示。

图 2-2　ADAM5510 应用软件

设置 COM 口

- ✧ 选择 COMPORT 并回车，显示通讯口状态域;
- ✧ 选择与 ADAM5510 相连的 COM 口，然后回车。缺省的波特率设置为 7600BPS 并且不能改变。屏幕如图 2-3 所示。

图 2-3　选择通讯口

第四步：ADAM5510 上电

选择 TERMINAL 回车，然后 ADAM5510 上电，5 秒钟后，屏幕如图 2-4 所示，系统被成功启动。

图 2-4 ADAM5510 仿真屏幕

2.2 模块安装

在系统中插入模块时，将模块的 PC 板与系统的上下槽对齐，向前推模块直到其与底板连接器稳固相连，模块连接好后，按下锁定开关固定模块。

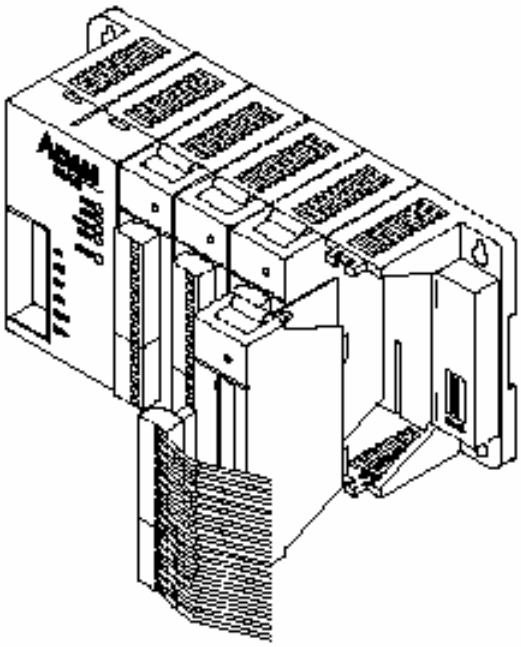


图 2-5　模块安装

2.3 I/O 槽与 I/O 通道编号

ADAM5510 系统提供了 4 个 I/O 模块安装槽位，编号为 0~3，任一槽内 I/O 模块的通道编号均从 0 开始。例如，ADAM4017 是 8 通道模拟量输入模块，其输入通道的编号为从 0 到 7。

2.4 安装

ADAM5510 安装方式有两种：面板和或导轨。

面板安装

为保证有良好的通风条件，系统须水平安装。不得垂直安装、倒装或在光滑水平面上安装。安装时应使用标准的#7（M4）螺丝。

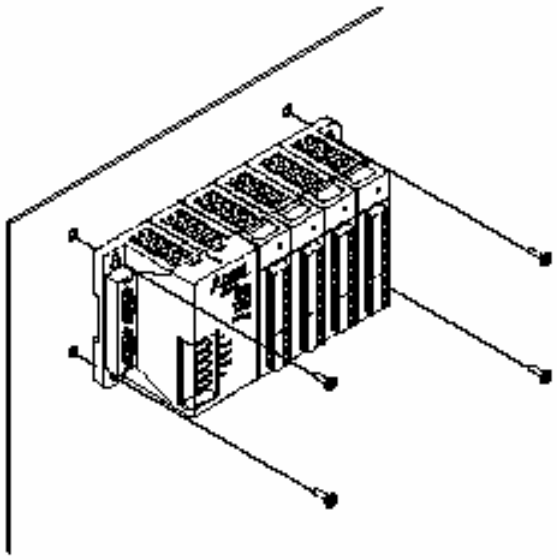


图 2-6　ADAM5510 平面安装

导轨安装

使用导轨可以把系统牢固地安装在机箱内。如果在导轨上安装系统，应考虑在导轨的两端加装堵头，以防止系统在导轨上水平滑动，这样可以防止接线松动。系统的底部备有锁定开关，当系统安装在导轨后，上推锁定开关，就会把系统锁定在轨道上。当拆卸系统时，向下推锁定开关，上提系统座，就可卸下系统。

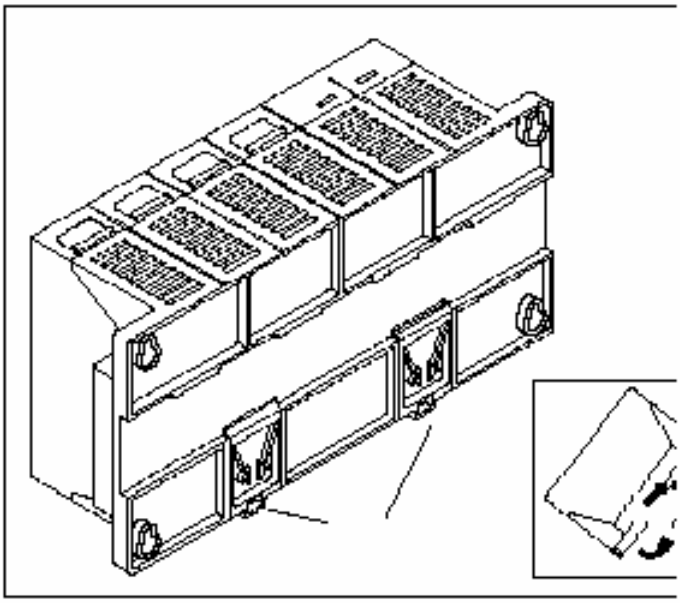


图 2-7　ADAM5510 轨道安装

2.5 跳线设置和 DIP 开关设置

CPU 卡上有二个跳线开关。图 2-8 给出了具体的位置。

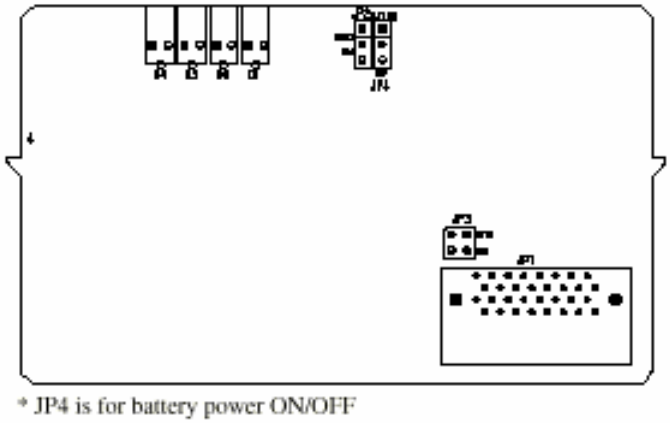
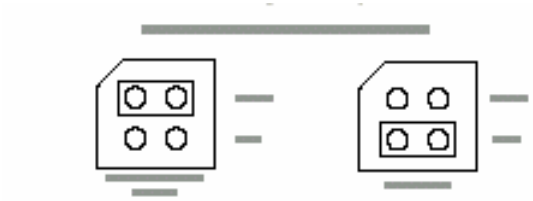


图 2-8　底板上的跳线与 DIP 开关的位置

COM2 口 RS485 控制模式设置

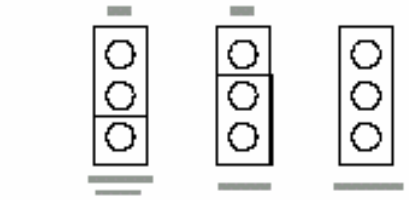
COM2 口是一个 RS485 接口，CPU 卡上的 JP3 可把其配置为自动控制方式或 RTS 控制方式，跳线设置如下：



自动控制　　RTS 控制

WATCHDOG 计时器设置

CPU 卡上的 JP2 可把 WATCHDOG 计时器设置为无效方式、复位方式或 NMI（NON-MASKABLE 中断）方式。跳线设置如下：



无效 WDT　复位方式　NMI 方式

网络地址设置

8 针 DIP 开关用于设置网络地址。有效的地址范围为 0~255，8 针开关的 ON 为二进制的 1，OFF 为 0。

例如，如果节点的 ID 为 03H，DIP 开关的 1 和 2 应为 ON，其余均为 OFF。缺省的节点 ID 为 01H。

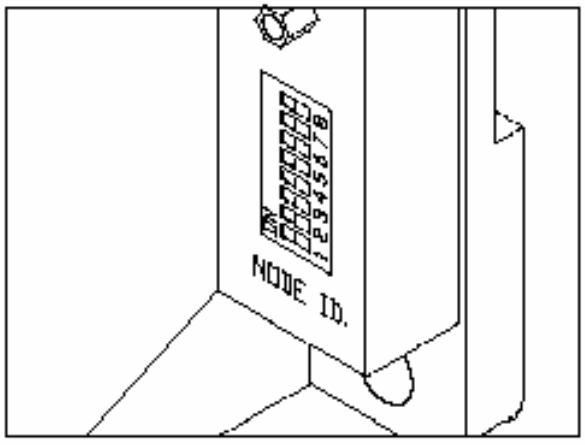


图 2-10　ADAM5510 网络地址 DIP 开关

2.6 布线与连接

本部分提供了电源线、I/O 线、通讯口和编程口的连接方法。

电源线连接：

虽然 ADAM5510 系统设计为标准工业 24VDC 供电，但其可用的电压范围为+10~+30VDC。螺丝端子+VS 和 GND 用于连接电源线。

注意：电源线直径不得低于 2.0mm

I/O 模块连接：

在 I/O 模块与现场之间通过螺丝端子模块相连。在电气设备与 I/O 模块之间连线时必须注意以下几点：

- ✧ 连线直径为 0.5 到 2.5
- ✧ 不得使用有接头的导线。
- ✧ 使用最短布线长度。
- ✧ 尽可能使用线鼻子。
- ✧ 避免靠近高压线。
- ✧ 近可能使输入线远离输出线。
- ✧ 避免导线弯折。

编程口连接

ADAM5510 编程口是 DB-9 接口，该口用于 ADAM5510 与主机相连，用于编程、配置和诊断。编程接口是 RS232 接口，仅仅使用 TX、RX 和 GND 信号。针脚分配如下：

针脚号	描述
1	NOT　USED
2	TX
3	RX
4	NOT　USED
5	GND
6	NOT　USED
7	NOT　USED
8	NOT　USED
9	NOT　USED

RS232 接口连接

COM1（3F8）配置成 RS232 接口。由于 RS232 接口是非标准的，不同的设备有不同的连接方式，如果串行设备有问题，应仔细检查针脚定义。下述表格显示了 RS232 接口的针脚定义。

针脚号	描述
1	DCD
2	RXD
3	TXD
4	DTR
5	GND
6	DSR
7	RTS
8	CTS
9	RI

COM2（2F8）指定为 RS485 接口。螺丝端子 DATA-和 DATA+用于 COM2 的 RS485 连接。

第三章　ADAM5510 系统

3.1 概述

ADAM5510 是基于 PC 的可编程微控制器，可独立进行数据采集与控制，通过多通道 I/O 模块 5510 可完成现场控制、监测和数据采集。与 IBM-PC 兼容的硬件平台允许用户运行用汇编语言或其它高级语言编写的程序。每一系统可处理任意 4 个 I/O 模块的组合（64 点），同时提供串行通讯口（RS232//485），使系统可以与其它设备进行通讯。

3.2 主要特点

ADAM5510 系统由两部分组成：主机和 I/O 模块。主机包括 CPU 卡、电源、4 槽机箱、2 个串行通讯口和一个编程口。同时还具备以下特点：

内置 80188CPU 和 ROM-DOS 操作系统

ADAM5510 的 CPU 卡包括一个 80188 微处理器。ROM-DOS 是与 MS-DOS 兼容的操作系统，提供除 BIOS 之外的 MS-DOS 的基本功能。在 ROM-DOS 环境下，用户可以运行用高级语言编写的 PC 软件或应用程序。

用于编程的内置 ROM 和 RAM 盘

ADAM5510 系统提供了 256KB FLASH ROM 和 256KB SRAM，用于用户程序下装、系统操作和数据存储。系统为用户下装程序提供了 170KB ROM 空间，为应用程序操作和文件传输提供了 234KB RAM。

内建 RS-232/422/485 通讯口

ADAM5510 具有二个串行通讯口，控制器可以很容易地与其它设备进行通讯。COM1 口指定为 RS-232 方式，COM2 口指定为 RS-485 方式。这种设计使控制器适用于任何应用。

三路隔离与看门狗计时器

电气躁声可以通过 I/O 模块、供应电源、系统接地等多种途径进入系统。ADAM5510 提供了 I/O 模块隔离（3000VDC）、通讯隔离（2500VDC）和电源隔离（3000VDC）。3 路隔离防止了对地环流，减少了电气噪音对系统的影响。系统还提供了看门狗计时器用于监视系统，当系统死机时，会自动复位

微处理器。

3.3　ADAM5510 系统技术参数

系统：

CPU：80188-40，16 位微处理器

FLASH　ROM：256KB （用户可用 170KB）

操作系统：ROM-DOS

计时器 BIOS：YES

实时时钟：YES

看门狗计时器：YES

COM1（3F8）：RS-232

COM2（2F8）：RS-485

编程口（RS232 接口，DB-9 连接器）：TX，RX，GND

I/O 能力：4

CPU 电源消耗：1.0W

状态显示：电源、CPU、通讯

电池后备（可选）：两年

RS-232 接口：

信号：TXD，RXD，RTS，CTS，DTR，DSR，DCD，RI，GND

模式 : 异步，全双工，点对点

连接器 : DB-9 针（COM1）

传输速度 : 最大 115.2KBPS

最大传输距离：50 英尺（15.2m）

RS-485 接口

信号：DATA+，DATA_

模式：半双工，多站

连接器：螺丝端子

传输速率：最大 115.2KBPS

最大传输距离：4000 英尺（1220 米）

隔离

电源：3000VDC

I/O：3000VDC

通讯：2500VDC(COM2)

电源

。+10 到+30VDC

。电源反向保护

。电源消耗 2.0W

环境

。操作温度-10---70

。储存温度-25---70

。湿度 5—95%，非凝结

。大气：非腐蚀气体

注意：

设备应在 30%湿度下进行操作，因为在低湿度下很容易发生静电反应。

如果在低湿度下使用，应采取足够的预防措施，例如使用接地环、防静电地板等等。

尺寸与重量

下述表格给出了系统单元和 I/O 单元的尺寸，单位为毫米。

表 3-1　ADAM-5510 系统与 I/O 模块的尺寸

模块	5510	5017	5018	5024	5051	5056	5060
重量	470g	79g	72g	75g	65g	68g	85g

基本结构框图

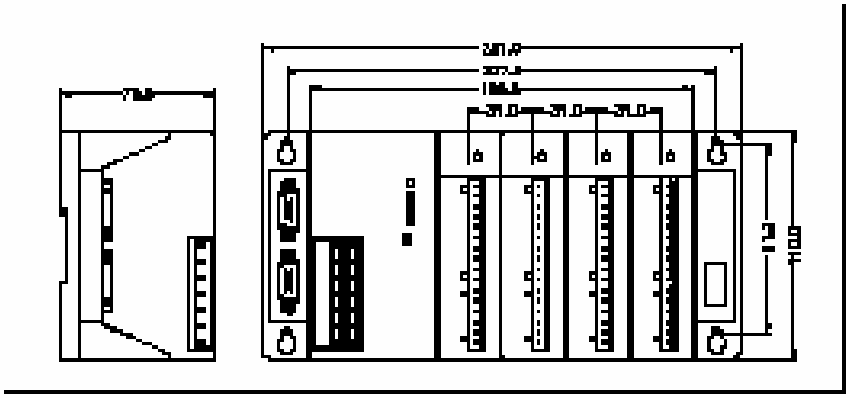


图 3-2　结构尺寸框图

第四章　编程与下装

本章讲述如何在 ADAM5510 系统上编写和下装应用程序，另外给出了一些限制和注意事项。

4.1 编程

ADAM5510 使用的 ROM-DOS 操作系统是一个与 MS-DOS 相兼容的操作系统，允许用户运行使用汇编语言或诸如 C、C++等高级语言编写的应用程序，在 ADAM5510 系统上运行应用程序还有一些额外的限制。

微型 BIOS 功能

ADAM5510 仅仅提供了二个串行口用于连接其它设备，因此微型 BIOS 仅仅提供了 10 个功能调用。由于不能使用其它功能调用，因此 ADAM5510 也许并未按您所想的方式工作。另外，有些直接调用 BIOS 功能的语言编译器，如 QBASIC，将不能在 ADAM5510 上运行。ADAM5510 微型 BIOS 功能调用列表如下：

功能	子功能	任务
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 tick0

程序代码转换

ADAM5510 的 CPU 为 80188，因此，在程序下装进 FLASH ROM 之前，必须转换为与 80186 或 80188 相兼容的代码，浮点操作必须设置为仿真方式。例如，若用 BORLAND C 语言开发应用程序，必须使用如下图所示的编译方式：

图 5-1　程序代码转换

其它限制

1．ADAM5510 不支持标准的 PC8253 功能，因此，不能使用 C 语言中的“ DELAY（）”功能调用。
2．某些特定文件，如操作系统、BIOS 和监视程序，必须保存在 FLASH ROM 中。ADAM5510 具有 170KB ROM 空间用于下装应用程序，另外具有 234KB RAM 用于运行应用程序。

看门狗计时器编程

ADAM5510 具有看门狗计时器功能，如果由于某种原因停止处理，看门狗计时器将复位 CPU 或产生一个中断。在工厂无人环境中该功能提高了系统的可靠性。

若使用看门狗计时器，必须编写功能调用激活。一旦计时器被激活，必须在应用程序中在 1.6 秒内清除。如果在规定的时间内没有清除，计时器将复位 CPU 或产生一个 NMI。可以在应用程序中使用功能调用清除计时器，在应用程序的结尾，再使用功能调用屏蔽掉计时器。

下列程序表示了如何对计时器编程。

Example

```
Main()
{
    enable_wdt();          /*enable  ADAM-5510  WDT function*/
    {                      /*user's  function block*/
        .
        .
        wdt_clear();       /*clear WDT timer*/
        .
        .
    }
disable_wdt()              /*disable ADAM-5510 WDT function*/
}
```

中断类型

在 ADAM-5510 中有三种中断，如下表所示：

中断名称	中断类型
Non-Maskable Interrupt (NMI)	02h
COM1 Interrupt	0ch
COM2 Interrupt	0eh

存储映象

下表表示了 ADAM-5510 的存储映象

0XF8000 – 0XFFFFF	Monitor program
0XF6C00 – 0XF7FFF	Mini BIOS
0XCC000 --	Start of Application ROM Disk (about 171K)
0XC0000	Start of ROM-DOS(about 48K)
0X40000 – 0XBFFFF	No Use
0X00400 – 0X3FFFF 0X00000 – 0X003FF 0X003F8 – 0X003FF 0X002F8 – 0X002FF 0X00070 – 0X00071	SRAM area System area COM1 COM2 Real Time Clock

4.2 下装与传输

这部分讲述如何从 PC 机中把应用程序下装到 ADAM5510 的 FLASH ROM 中，以及如何从 PC 机中把文件传送到 ADAM5510 的 SRAM 中。

在 PC 机中安装应用软件

在用户磁盘中包括下列文件和目录：

.SOURCE　　〈DIR〉
.ALLFILE　　〈DIR〉
.ADAMMINI.HEX
.ADAM_DEM.HEX
.ROM-DOS.HEX
.DEMO-DIS.HEX
.ADAMMINI.BAT
.ADAM5510.EXE
.HEXCAT.EXE
.ROMDISK.EXE
.TRANSFER.EXE

把用户磁盘上的所有文件拷贝到 PC 机的指定目录下。

产生 ALLFILE 目录

使用 ADAM-5510 用户软件把应用程序从 PC 机中传送到 ADAM5510 的 FLASH ROM 中。ADAM5510 用户软件首先安装在 PC 机中，用户必须创建一个新目录，名称为 ALLFILE。用户必须把下述文件装入 ALLFILE 目录中：将在 ADAM5510 中安装的应用程序，COMMAND。COM，AUTOEXEC。BAT，CONFIG。SYS，在 AUTOEXEC。BAT 中必须包括应用程序，这样一来，当 ADAM5510 上电时，应用程序将自动开始执行。在向 ADAM5510 的 FLASH　ROM 中下装时，用户软件首先从 FLASH ROM 中清除所有非固定文件，然后把 ALLFILE 目录下的所有文件安装到 FLASH ROM 中。因此特别强调当应用软件对 ALLFILE 进行存取操作时，ALLFILE 目录下的所有要求的文件必须为可用的。

下装到 FLASH ROM　（ADAM5510 的 C 盘）

在 ADAM5510 用户软件和 ALLFILE 下的文件都正确安装到 PC 机上后，可以开始执行用户软件。开始执行后屏幕显示如下：

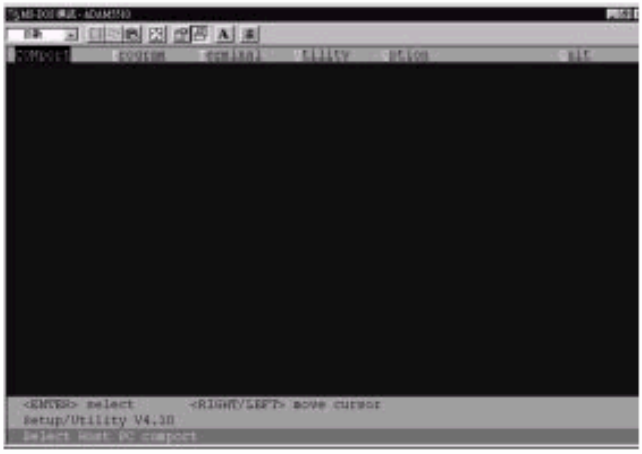


图 5—2：主屏

用户软件开始运行后，选择与 ADAM5510 相连的 PC 机的 COM 口，在菜单上选择“ PROGRAM ”并回车开始下装，屏幕如图 5-3 所示。

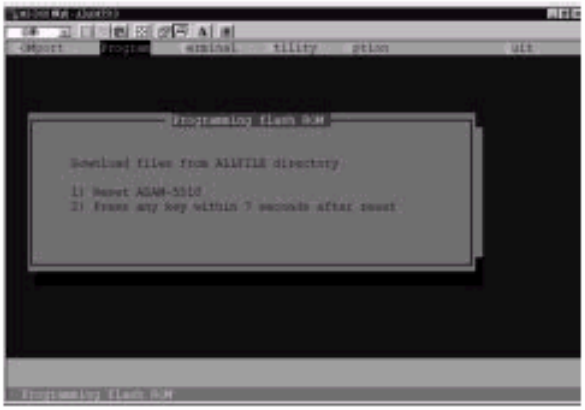


图 5-3 程序下装

按照屏幕显示的指令，关掉 ADAM5510 的电源，然后重新上电，在 7 秒之内按任意键以便把 ALLFILE 中的文件烧入 ADAM5510 的 FLASH ROM 中。在文件成功烧入后,关掉电源，再重新上电一次，ADAM5510 系统将自动执行应用程序。

向 SRAM 中传输文件（ADAM5510 的 D 盘）

ADAM5510 提供了 234KB SRAM 空间，用于程序操作、控制逻辑和形成仿真。在向 FLASH ROM 中传送执行代码之前，可以把文件从主机传送入 SRAM 中（D 盘）。执行用户软件，选择终端方式，然后按 ALT-T，文件开始传输。

输入想要传输的目录名和文件名，然后按回车开始传输。可以在 D 盘中检查所传输的文件。

第五章　　I/O 模块

5.1　模拟量输入模块

模拟量输入模块用来将来自传感器的电压、电流、热电偶（TC）、热电阻（RTD）信号转换成数字量。通过光电隔离电路，模拟量输入模块可以。

ADAM－5017　8 通道模拟量输入模块

ADAM－5017 是 16 位 8 通道差动输入模块，通道输入范围均可程控。输入量程包括：mV(±150mV，±500mV)，V（±1V，±5V，±10V）及电流输入（±20mA,需要 250Ω电阻）。5017 提供了 3000Vdc 光电隔离输入保护。

ADAM－5018 7 通道热偶输入模块

ADAM－5018 是 16 位 7 通道热偶输入模块，所有通道的输入范围均可程控。输入范围包括：mV（±15mV，±50mV,±100mV,±500mV）,V(±1V，±2.5V)，电流输入（±20mA,需要 250Ω热偶）及热电偶输入（J，K，T，E，R，S，B)。模块的接线端子接有 CJC 电路。性能指标见表。

5.2　模拟量输出模块

ADAM－5024　4 通道模拟量输出模块

ADAM－5024 是 4 通道模拟量输出模块，用来将数字量信号转换成模拟量信号。通过配置软件可定制斜率和启动电流，输出可配置成电流或电压。性能指标见表。

5.3　数字量输出/输入模块

ADAM-5050　16 通道通用数字量输出/输入模块

5050 具有 16 个数字量输入/输出通道，每个通道可通过 DIP 开关分别配置成输入或输出。数字量输出为集电极开路输出，可用来控制固态继电器（SSR)，进而控制加热器、泵及动力设备；数字量输入可用来监测限制、安全开关等信号。注意：当已经设置成输出的通道接强迫接收输入信号将会损坏通道。

ADAM－5051　16 通道数字量输入模块

5051 提供 16 路数字量输入。

ADAM－5052　8 通道隔离数字量输入

5052 提供了 8 通道全隔离型数字量输入通道。所有通道均提供 5000Vrms 的隔离，防止接地环流及电源浪涌对信号的干扰。

ADAM－5056　16 通道数字量输出模快

5056 提供了 16 路输出通道，数字量输出为集电极开路输出，用户可用来且换固态继电器（SSR)。

5.4　继电器输出模快

ADAM 5060 继电器输出模快

5060 提供了 6 路继电器输出通道，其中两个为 A 型，四个为 C 型。

5.5　I/O 模块的设置

ADAM－5510 系统随机提供了用户盘，用户盘内包括了用户程序，用户程序可帮助用户对 I/O 模块进行设置。设置步骤如下：

1. 将用户盘插入软驱（如 A：)；
2. 运行用户程序 ADAM5510.EXE 仿真 ADAM5510 操作系统，然后从菜单选择“ TERMINAL ”回车；
3. 按下“ Alt-T ”将 A：\SIMU5000.EXE 文件传至 5510 的 D 盘；
4. 从 5510 系统的 D 盘执行 SIMU5000.EXE，以便 ADAM5510 仿真 ADAM5000/485;
5. 退出 ADAM5510 用户程序；
6. 从 A 盘启动 ADAM 用户程序（A：\ADAM.EXE)，用户就可以进行 I/O 模块的设置。

5.6 模拟量模块的标定

在某些情况下，用户需要对模拟量模块进行标定，标定的结果会保存在内置的 EEPROM 中。随机提供的用户程序支持模块的标定。

输入模块的标定（5017、5018）

1. 将模拟量输入模块出入 5510 系统中，上电预热 30 分钟；
2. 利用用户程序设定输入范围；
3. 在 Vi+和 Vi-端接入精密电压源；
4. 执行零标定命令；
5. 执行范围标定命令；
6. 执行 CJC 标定命令；

模拟量输出模块标定

输出电流的标定可以采用低标定值和高标定值来标定，其范围包括：0－20mA 和 4－20mA。由于 0mA 为内部绝对参考点，仅需对 4mA 和 20mA 进行标定。基本步骤如下：

1. 插入模拟量输出模块同时上电预热 30 分钟；
2. 利用用户程序对模块进行正确设置；
3. 将 5 位电压或电流表接入接线端子，同时并入 250Ω，0.01%和 10ppm 的电阻；
4. 启用模拟量输出为 4mA 的命令；
5. 检查实际输出，如果不等于 4mA（或 1V），在”Calibrate”菜单中选择“ Trim”调节输出，使之等于 4mA 或 1V；
6. 将标定数据存入 EEPROM；
7. 启用模拟量输出为 20mA 的命令；
8. 检查实际输出，如果不等于 20mA（或 5V），在”Calibrate”菜单中选择“ Trim”调节输出，使之等于 20mA 或 5V；
9. 将标定数据存入 EEPROM。

第六章　ADAM-5510 与组态软件的连接

研华 5510 分 5510 普通版与 5510/P31 板。5510/P31 版可以利用 Advantech Paradym -31 直接和 FX 通讯，而 5510 普通板和 FX 之间的通讯则需要用户对 FX 进行二次开发。对于不熟悉 FX 开发工具的用户来说，显然难度很大。实际上有一种简易方法来实现 5510 与 FX 的通讯，即仿 5000/485。

大家知道，研华 FX 提供了 ADAM-5000/485 的驱动程序，通讯协议遵循命令/响应的方式。FX 通过发送 ASCII 命令（命令集可以在 5000 说明书中查到）来与 5000 进行数据交换。例如，FX 要获取 5000 地址为 01 的 5017（第 1 槽）的数据，FX 则发送＃01S1（CR）（注：CR 表示回车符），5000 接收到指令后则返回 1 槽中 5017 的数据（响应命令格式请参见 5000 指令集）。

如果将 5000 改成 5510，其他配置不变，FX 能否获取 5510 中 5017 的数据？答案是肯定的。当 FX 发送＃01S1 至 5510 时，5510 中用户程序（当然需要用户自己编制）首先对 FX 发送的命令进行判断，如果接收的命令符合 5000/485 指令集（如为＃01S1（CR)），则按照指令集的响应格式返回（如：>＋1.4567 +1.4852 +1.6661 +1.6321 +1.6459 +2.989 +1.7768 +1.1623 （CR)）。这时 FX 就将 5510 认为是 5000/485 了。

同样道理，对于支持 5000 的上位机组态软件均可以采用同样的方式和 5510 进行通讯，如 Advantech VISIDAQ，Intoch 等。

第七章　ADAM-5510 A1 版函数库

在 ADAM5510/A1 版系统中可以使用一些功能调用。功能库可以帮助您编写出高效的应用程序。功能库支持 TURBO C 2.0 和 MICROSOFT C 6.0 编程语言。

ADAM5510 库磁盘包括下述二个子目录：

.TC20	TURBO C 2.0 库
.MSC60	MICROSOFT C 6.0 库

TURBO C 功能库

TC20 目录包括以下子目录和文件：

ADAM5510.H	声明文件
5510L.LIB	大模式库
5510S.LIB	小模式库
\SOURCE*.*	源代码

MICROSOFT C 功能库

MSC60 目录包括以下目录和文件

ADAM5510.H	声明文件
5510L.LIB	大模式库
5510C.LIB	紧凑模式库
5510M.LIB	中模式库
5510S.LIB	小模式库
\SOURCE*.*	源代码

comm_init

语法：
Int　comm_init(int buf_size)

功能描述：
在任何功能调用使用通讯口前，初始化通讯口和中断队列。

参数
int　buf_size

描述
为每一通讯口设置用于接收数据的缓冲区的大小。单位为字节。

返回
成功返回 1，失败返回 0。

comm_exit

语法
Int　comm_exit（）

功能描述：
若用户调用 comm_init 功能，在用户程序退出之前必须调用本功能释放资源。

返回：
成功返回 1，失败返回 0。

comm_open

语法：

Int　comm_open(unsigned char port, unsigned long baud, int parity, int data, int stop, int cmd_type)

功能描述：

为收发数据打开通讯口。

参数	描述
unsigned char port	定义通讯口 1：COM1 2：COM2
unsigned long baud	波特率设定。有八种波特率可以设定--- 12000，2400，4800，9600，19200，38400，57600，115200。单位
int　parity	极性设置 0：无极性别　1：单极性　2：双极性
int　data	数据位设置。有四种数据长度类型—8、7、6、5，单位为位。
Int　stop	停止位设置 1，2 停止位=1 数据位=5，6，7 或 8 停止位=2 数据位=6，7 或 8 请参考 RS-232 芯片手册获取详细信息
int　cmd_type	接收数据格式设置 0：单字节格式，数据格式为单字节字符 用户可以使用接收单字节数据 1：命令格式，数据格式为字串。用户可以使用接收字符串数据

返回：

成功返回 1，失败返回 0。

comm_send

语法：
Int　comm_send(unsigned char c, unsigned char port)

功能描述：
向指定通讯口发送一个字符。

参数	描述
Unsigned char c	字符内容
Unsigned char port	指定通讯口（COM1 或 COM2） 1：COM1 2：COM2

返回：
成功返回报，失败返回访。

comm_get_rec_datas

语法：

Int　comm_get_rec_datas(unsigned *length, unsigned char **data, unsigned char port)

功能描述：

本功能调用接收单字节数据。通讯口接收到任何数据，调用返回成功。若通讯口为空则返回失败。“ length ”存放数据长度，“ data ”存放数据内容。

参数	描述
Unsigned *length	返回接收数据长度
Unsigned char **data	返回接收数据内容。用户必须首先分配数据缓冲区（使用 comm_int）。在程序退出之前必须释放缓冲区（使用 comm_exit）
Unsigned char port	指定通讯口（COM1 或 COM2） 1：COM1 2：COM2

返回：

成功返回 1，失败返回 0。

comm_get_rec_str

语法：

Int　comm_get_rec_str(unsigned char **data, unsigned char port)

功能描述：

本功能接收字符串数据。通讯口接收到字符串时返回成功（终止字符为 0x0d）。若通讯口为空返回失败。

参数	描述
Unsigned char **char	返回接收的字符串的内容。用户必须先分配数据缓冲区（使用 comm_int）。在退出程序之前，必须释放缓冲区。（使用 comm_exit）
Unsigned char port	定义通讯口 1：COM1 2：COM2

返回：

成功返回 1，失败返回 0。

led_init

语法：
Void　led_init()

功能描述：
若用户程序想控制 CPU 卡上的 4 个 LED，必须首先调用此功能进行初始化。

led1,led2,led3

语法

Void led1(int type)
Void led2(int type)
Void led3(int type)

功能描述：

LED　ON/OFF 控制

参数	描述
Int　type	0：LED OFF
	1：LED ON

enable_wdt

语法：
Enable_wdt()

功能描述：
本功能激活看门狗计时器。用户调用本功能后，必须调用 wdt_clear()功能刷新看门狗计时器。

wdt_clear

语法：
Wdt_clear ()

功能描述：
本功能刷新看门狗计时器，避免其复位 CPU。

disable_wdt

语法：
Disable_wdt()

功能描述：
本功能屏蔽看门狗计时器。

io_init

语法：
Int　io_init()

功能描述：
若用户要存取 ADAM5510 的 I/O 数据，必须先调用本功能对其进行处始化。

返回：
成功返回 1，失败返回 0。

io_update_data

语法：
int　io_update_data()

功能描述：
本功能调用更新 ADAM5510 的 I/O 数据。用户必须频繁调用本功能以便取得最新的数据。

返回：
成功返回 1，失败返回 0。

io_get_data_all

语法：

int　io_get_data_all(unsigned char *data)

功能描述：

本功能采集 ADAM5510 系统所有 I/O 模块的所有 I/O 数据。

参数	描述
unsigned char *data	返回所有 I/O 模块的所有的 I/O 数据，每一槽位有 32 字节的数据缓冲区，所以对 4 槽的 I/O 模块，用户必须分配 128 字节的空间用于存储所有的 I/O 数据。 对于 A/I 模块：ADAM5017，5018 ADAM5017 具有 8 个通道。 ADAM5018 具有 7 个通道。 每一通道有 4 字节存储空间用于记录模拟量输入值，数据格式为浮点方式。通道 0 为 0~3 字节，通道 1 为 4~7 字节，依此类推。 对于 A/O 模块：ADAM-5024 ADAM5024 具有 4 个通道。 每一通道有 4 字节存储空间用于记录模拟量输出值，数据格式为浮点方式。通道 0 为 0~3 字节，通道 1 为 4~7 字节，依此类推。 对于 DI/O 模块：ADAM-5050，5051，5056 ADAM5050，5051，5056 都有 16 个通道。 每一通道有 1 位用于记录输入或输出值，如下表所示：

字节	字　节　0								字　节　1							
位　号	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7
通道号	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

对于继电气模块：ADAM5060，5068

ADAM5060 有 6 个通道。

ADAM5068 有 8 个通道。

每一通道有 1 位用于记录输出继电器值，如下表所示：

字　节	字节 0							
位　号	0	1	2	3	4	5	6	7
通 道 号	0	1	2	3	4	5	6	7

返回：
成功返回 1，失败返回 0。

io_get_data_single_slot

语法：

int　io_get_data_single_slot(unsigned char slot,unsigned char *data)

参数	描述
unsigned　char slot	指定槽号。（0，1，2，3）
unsigned char *data	返回指定 I/O 模块的 I/O 数据。数据缓冲区为 32 字节。 用户必须首先分配数据缓冲区。数据格式参考 io_get_data_all。

返回：

成功返回 1，失败返回 0。

io_read_slot_type

语法：

io_read_slot_type(unsigned char slot, unsigned　char　*type)

功能描述：

unsigned char slot	指定槽号（0，1，2，3）
unsigned char *type	0xff：无 I/O 模块 0x04：ADAM-5017 0x05：ADAM-5018 0x18：ADAM-5024 0x10：ADAM-5050 0x11：ADAM-5051 0x12：ADAM-5056 0x14：ADAM-5060 0x15：ADAM5052

返回：

成功返回 1，失败返回 0。

io_set_ao

语法：

io_set_ao(unsigned char slot, unsigned char ch, float ao_data)

参数	描述
unsigned char slot	指定槽号（0，1，2，3）
unsigned char ch	指定通道号（0，1，2，3）
float ao_data	模拟量输出值。电流模式输出范围为 0.00 到 20.0， 电压模式输出范围为 0.00 到 10.0，

返回：

成功返回 1，失败返回 0。

io_set_do

语法

io_set_do(unsigned char slot, unsigned do)

功能描述：

本功能调用设置数字量数据输出。

参数	描述
unsigned char slot	指定槽号（0，1，2，3）
unsigned do	“do”是 4 字节十六进制参数，代表数字量输出值。该值表示为 16 位，0 代表数字量输出通道为 OFF，1 表示通道为 ON，如下表所示：

位　号	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
通道号	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

返回：

成功返回 1，失败返回 0。

io_get_ai_ch_mask

语法：

int io_get_ai_ch_mask(unsigned char slot, unsigned char *mask)

功能描述：

本功能获取模拟量输入通道的状态，而不管模拟量输入通道是可用或不可用。

参数	描述
unsigned char slot	指定槽号（0，1，2，3）
unsigned char *mask	返回模拟量输入通道的状态。返回值为 2 字符十六进制值，第一个 4 位表示通道 4~7 的状态，第二个 4 位表示通道 0~3 的状态。0 表示通道是不可用。1 表示通道是可用。如下表所示：

位　号	0	1	2	3	4	5	6	7
通道号	0	1	2	3	4	5	6	7

返回：

成功返回 1，失败返回 0。

第八章 ADAM-5510 A2 版函数库

7.1 介绍

为了有效利用 ADAM-5510 存储空间,ADAM-5510/2 函数库已经分成四个较小的库.因此,用户仅仅需链接那些需要的库。连接的库越小,编译后可执行文件将越小.

注 1:这些函数库仅支持 Borland Turbo C++ 3.0
注 2:请在你的工程文件中包含所有必需 ADAM-5510 函数库.

7.2 库分级

ADAM-5510 有四类函数库，依次分为:

A. 系统函数: (UTILITY*.LIB)
B. 通信函数: (COMM*.LIB)
C. 低速 I/O 模块访问函数:(LIO*.LIB)
D. 高速 I/O 模块访问函数:(HIO*.LIB)

7.3 ADAM-5510 函数库支持的存储模式

ADAM-5510 函数库支持四种存储器模式:

SMALL
MEDIUM
COMPACT
LARGE.

你应根据你的存储器方式安排应用文件大小.例如,如果你采用小模式，你需链接 UTILITYS.LIB 和 LIOS.LIB 实现系统和低速度 I/O 模块访问.另一方面,如果你应用大模式,你应链接 UTILITYL.LIB 和 LIOL.LIB.

7.4 索引分类.

系统函数 (UTILITY*.LIB)

ADAMdelay()
EraseSector()
Get_BoardID()
Get_NodeID()
Get_NVRAM_Size()
GetRTCtime()
Get_SysMem()
LED_init()
LED_OFF()
LED_ON()
ProgramByte()

ProgramSector()

通信函数 (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() c
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
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

低速 I/O 模块访问函数 (LIO*.LIB)

AiUpdate()
Get5013()
Get501718()
GetRange5013()
GetRange501718()
Init5013()
Init501718()

高速 I/O 模块访问函数 (LIO*.LIB)

Get5017H()

Init5017H()
GetRange5017H()

7.5 函数库描述

7.5.1 系统函数(UTILITY*.LIB)

ADAMdelay

语法:
void ADAMdelay(unsigned short msec)

描述: 使程序延迟特定的毫秒数。

参数	**描述**
msec：	从 0 到 65535.

返回值:
None
.

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

EraseSector

语法:

```
unsigned short EraseSector(unsigned long ulBase)
```

描述:

删除在 256 KB 闪存范围内 64 KB 数据区

参数	描述
ulBase：	用户自定义的待删除地址范围,地址范围 在 0x80000L 至 0xB0000L 之间。

返回值:

TRUE 表示删除成功； FALSE 表示超出地址范围。

例子:

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

Get_BoardID

语法:

```
unsigned char Get_BoardID(int Board)
```

描述:

获得控制器中所插模块类型

参数	**描述**
Board	ADAM-5510 中模块的编号,从 0 到 3

返回值:

I/O 模块名字	返回值
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

例子:

```
unsigned char IOModuleName;
unsigned char SlotNumber;
void main(void)
{
 /* Read IO module name in Slot 0*/
SlotNumber = 0;
IOModuleName = Get_BoardID(SlotNumber);
IOModuleName = Get_BoardID(SlotNumber);
If( IOModuleName == ADAM5051_ID)
   {
   /* IO Board is current, put your code in Here */
   }
else
   {
    printf(“ IO Board is NOT ADAM5051?”);
    printf(“Check your system set-up”);
   }
}
```

Get_NodeID

语法:
unsigned char Get_NodeID(void)

描述:
获得 ADAM-5510 控制器 DIP 开关编号.

参数描述
None.

返回值:
ADAM-5510 控制器的 DIP 开关编号.

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

Get_NVRAM_Size

语法:
unsigned char Get_NVRAM_Size(void)

描述:
获得电池备份 RAM 大小.单位是扇区，每扇区 4KB，理论上最大容量是 60 KB.

参数　　描述
none.

返回值:
返回 NV RAM 区段的编号,从 1 到 15.

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

GetRTCtime

语法:
unsigned char GetRTCtime(unsigned char Time)

描述:
读 RTC 芯片值。用户按日期要求可以激活程序.

参数描述
Time

RTC_sec	秒
RTC_min	分
RTC_hour	小时
RTC_day	天
RTC_week	星期
RTC_month	月
RTC_year	年
RTC_century	世纪

返回值:
返回用户请求值.

例子:

```
void main(void)
{
printf(“Century = %d” GetRTCtime(RTC_century) );
printf(“”Year = %d” GetRTCtime(RTC_year) );
printf(“”month = %d” GetRTCtime(RTC_month) );
printf(“”weekday = %d” GetRTCtime(RTC_week) );
printf(“”day = %d” GetRTCtime(RTC_day) );
printf(“”hour = %d” GetRTCtime(RTC_hour) );
printf(“”min = %d” GetRTCtime(RTC_min) );
printf(“”sec = %d” GetRTCtime(RTC_sec) );
}
```

Get_SysMem

语法:
unsigned char Get_SysMem(unsigned char which_byte)

描述:
从安全 SRAM 读取一个字节.

参数	**描述**
which_byte：	从 0 到 112,用户自定义.

返回值:
返回安全 SRAM 字节值.

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

LED_init, LED_OFF, LED_ON

语法:

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

描述:

控制 LED ON 或 OFF。首先 LED I/O 端口必须被初始化。控制 LED ON 或 OFF 需要一定时间的延迟

参数	**描述**
which_led	分别为 PWR、RUN 或 COMM

返回值:

None.

例子:

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

ProgramByte

语法:

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

描述:

将一字节信息写入 256 KB 闪存中。本函数支持数据记录或大量信息存储.

参数	描述
ulAddress	用户自定义的字节传输的目标地址，范围从 0x80000L 至 0xBFFFFL.
Byte	存储的值

返回值:

TRUE：　表示存储成功

FALSE：　表示错误(目标已经被占用、超出地址范围或写入错误).

例子:

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

ProgramSector

语法:

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

描述

将全部 32 KB 全程变量数据区段（SECTOR_DATA[]）写入 256 KB 闪存中.

参数	**描述**
ulAddress	用户在闪存中自定义的目标地址,地址范围从 0x80000L 至 0xB8000L.
SECTOR_DATA	原始内存的起始地址。

返回值:

TRUE 表示内存传输成功

FALSE 表示传输错误(目标已经被占用、超出地址范围或写入错误).

例子:

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

read_backup_ram

语法:
unsigned char read_backup_ram(unsigned int index)

描述:
读取在索引地址中备份 RAM 值，备份 RAM 大小为 60 KB,索引 = 0 至 61439;绝对地址从 0x30000 至 0x3EFFF.

参数	描述
index	从 0 到 61439，大小为 60 KB。

返回值:
返回索引地址中备份 RAM 值。

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

read_mem

语法:

```
unsigned char read_mem (int memory_segment , unsigned int i)
```

描述:

读取远程存储器数据(256 KB闪烁存储器),从 0x80000L到 0xBFFFFL,(绝对地址) = (SEG*16 +偏移).例如, (0x800FFL) = (0x8000*16 + 0x00FF).

参数	描述
memory_segment	用户自定义地址范围 (0x8000 0xBFFF)
i	偏移量 (范围从 0x0000 至 0xFFFF)。

返回值:

返回指定存储器存储值。

例子:

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

Release_All

语法:
void Release_All()

描述:
释放 ADAM-5510 系统所有计时器资源。

参数　　描述
none

返回值:
None.

例子:
```
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 */
 Timer_Reset(idx);
/*Releases all timer resources */
Release_All()
}
```

Set_NVRAM_Size

语法:
void Set_NVRAM_Size(unsigned char sector)

描述:
设置电池备份 RAM 大小.单位为扇区,每个区段 4K 字节大小.理论上最大尺寸是 60 KB

参数	**描述**
sector	NV RAM 大小，从 1 到 15。

返回值:
None.

例子:
```
void main()
{
 Set_NVRAM_Size(31);
 /* sets NVRAM size to 124 KB*/
}
```

SetRTCtime

语法:
void SetRTCtime(unsigned char Time, unsigned char data)

描述:
设置时时钟的日期和时间.

参数	**描述**	
Time：	RTC_sec	秒
	RTC_min	分
	RTC_hour	小时
	RTC_day	天
	RTC_week	星期
	RTC_month	月
	RTC_year	年
	RTC_century	世纪.
Data	新内容	

返回值:
None.

例子:
```
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);
} .
```

Set_SysMem

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

描述:
写一个字节到安全 SRAM。安全 SRAM 支持 113 字节，主要用于存储用户重要信息。

参数	**描述**
which_byte	从 0 到 112,用户自定义。
Data	待保存的数据。

返回值:
None.

例子:
```
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]);
}
```

Timer_Init()

语法:
int Timer_Init()

描述:
初始化 80188 微处理器的定时器

参数　　描述
None

返回值:
0:　初始化成功.
1:　定时器已初始化.

Timer_Reset

语法:
void Timer_Reset(int idx)

描述
复位由整型 idx 标识的定时器到它的初始化状态.

参数	**描述**
idx	定时器索引.

返回值:
None.

Timer_Set

语法:

int Timer_Set(unsigned int msec)

描述:

从微处理器请求定时器功能，然后设置定时器时间间隔 .定时器最小间隔为 5 毫秒.函数调用成功，返回调用定时器的 ID。程序员应考虑当对定时器进行编程时，定时器是否超时。变量 tmArriveCnt[idx]用来检查定时器状态。值 0 表示计时器仍然在计数.值不等于 0 意味着定时器已暂停.

参数	描述
msec	时间间隔,最大值是 65536.

返回值:

返回值代表定时器 ID.最大值为 100。-1 表示函数调用失败，-1 功能失败.

WDT_clear, WDT_disable, WDT_enable

语法:

```
void WDT_clear(void)
void WDT_disable(void)
void WDT_enable(void)
```

描述:

WDT_clear：　清除看门狗定时器.
WDT_disable：禁止看门狗定时器.
WDT_enable：　允许看门狗定时器.
当定时器被允许,它必须在 1.5 秒内被清除.

参数	描述
None.	

返回值:

None.

例子:

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

write_backup_RAM

语法:
void write_backup_RAM(unsigned int index, BYTE data) Description:

描述:
写字节到电池备份存储器.

参数	**描述**
index	在电池备份 RAM 中数据索引,从 0 到 61439，总共 60 KB 电池备份 SRAM.
Data	待写给电池保护 SRAM 中的一个字节数据.

返回值:
None.

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

7.5.2 低速 I/O 库函数(LIO*.LIB):

AiUpdate

语法:
int AiUpdate(int Board, int *channel)

描述:
检查低速模拟量输入模块,比如 ADAM-5017,ADAM-5018 和 ADAM-5013,是否准备好.

参数	描述
Board	ADAM-5510 的槽号，从 0 到 3
Channel	通道号 ADAM-5017 的有效值为 0--7 ADAM-5018 的有效值为 0--6. ADAM-5013 的有效值为 0--2

返回值:
0 : 就绪;-1 :不就绪 ;-2 :模块的硬件失败

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

Get5013

语法:

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

描述:

读 ADAM-5013 模块数据.

参数	描述
Board	从 0 到 3（Slot0 ...Slot3）
Channel	0-3
*pValue	返回值.

返回值:

None.

例子:

```
// An example for Init5013, Get5013 and GetRange5013
main()
{
  int *value,*range, i, j;
  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
        Get5013(0,j,value);
        if (i==3)
           {
             GetRange5013(0,j,range);
             printf(" ADAM-5013#%d=%d\n",j,*value);
             printf(" (with range is 0x%x",*range);
           }
      }
   }
}
```

Get501718

语法:

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

描述:

读 I/O 模块数据.

参数	描述
Board	0,1,2,3 表示 Slot0 ...Slot3.
Channel	0 - 6 （ ADAM-5018）
	0 – 7 （ ADAM-5017）
*pValue	返回值.

返回值:

None.

例子:

```
main()
{
  int *value,*range, i, j;
  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++)
      {
        Get501718(3,j,value);
       if (i==3)
        { GetRange501718(3,j,range);
          printf(" ADAM-5018(or ADAM5017)#%d=%d\n",j,*value);
          printf(" (with range is 0x%x",*range);
        }
      }
  }
}
```

GetRange5013

语法:

```
void GetRange5013(int Board, int Channel, void *pRange)
```

描述:

读 ADAM-5013 模块的输入范围.

参数	**描述**
Board	0 至 3 为 Slot0 ...Slot3.
Channel	0 至 2 为 ADAM-5013 通道号.
*pRange	返回输入范围代码.

返回值:

None.

GetRange501718

语法:

```
void GetRange501718(int Board, int Channel, void *pRange)
```

描述:

读 ADAM-501718 模块输入范围.

参数	**描述**
Board	0 至 3 为 Slot0 ...Slot3.
Channel	0 至 7 为 ADAM-5017 通道号. 0 至 6 为 ADAM-5018 通道号.
*pRange	返回输入范围代码

返回值:

None.

Init5013

语法:

```
void Init5013(int Slot)
```

描述:

初始化 ADAM-5013.操作 5013 之前必须执行初始化命令.

参数	**描述：**
Slot	从 0 到 3

返回值:

None.

例子:

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

Init501718

语法:
void Init501718(int Slot)

描述:
初始化 ADAM-5017 或 ADAM-5018.

参数	**描述**
Slot	从 0 到 3

返回值:
None.

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

Get5050, Get5051, Get5052

语法:

```
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)
```

描述:

读 I/O 模块数据值.

参数	描述
Board	编号,从 0 到 3；
Bit	参见Size参数
Size	ABit, AByte, AWord
	如果 Size= ABit, Bit=0..15 (pin0..pin15)
	如果 Size=AByte, Bit=0 低字节数据; Bit=8高字节数据
	如果 Size=AWord, Bit 不起作用
pValue	返回值

返回值:

None.

例子:

```
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);
}
```

Init5024

语法:

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

描述:

初始化 ADAM-5024 模块

参数	描述
Slot	5024 所在的槽号
ch0_val	初始化通道 0 输出值
ch1_val	初始化通道 1 输出值
ch2_val	初始化通道 2 输出值
ch3_val	初始化通道 3 输出值

返回值:

None.

例子:

```
void main(void)
 {
  Init5024 (0,0,0,0,0) ;
}
```

Set5024

语法:

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

描述:

设置 ADAM-5024 输出值。

参数	描述
pValue	模拟量输出值
Board	槽号（0 至 3）
Channel	通道号（0 至 3）

返回值:

None.

Set5050, Set5056, Set5060, Set5068

语法:

```
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)
```

描述

设置 ADAM-5050,ADAM-5056,ADAM-5060 和 ADAM-5068 模块输出值.

参数	**描述**
pValue	输出的数字值.
Board	0 至 3 (Slot0 .. Slot3).
Bit	参见"Size" 参数
Size	ABit, AByte, AWord 如果 Size = ABit, Bit = 0 ...15 (pin0 ...pin15) 如果 Size = AByte, Bit = 0 表示低字节, Bit = 8 表示高字节 如果 Size = AWord, 不考虑 Bit

返回值:

None.

例子:

```
void main(void)
 {
  unsigned char Bitdata = 1;
  Set5056( &Bitdata, 0, 13, ABit);
}
```

5.2.5.2 高速 I/O 函数(HIO*.LIB)

Get5017H

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

描述:
读 ADAM-5017H 模块数据值.

参数	**描述：**
Board	0 至 3 为 Slot0 ...Slot3.
Channel	0 至 7 为 ADAM-5017H 通道号.
*pRange	返回值

返回值:
None.

例子:
```
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++)
      Get5017H(1,j,value);
      if (i==3)
      {
        GetRange5017H(1,j,range);
        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);
       }
    }
  }
 }
```

GetRange5017H

语法:

```
void GetRange5017H(int Board, int Channel, void *pRange)
```

描述:

读 ADAM-5017H 模块输入范围.

参数	描述
Board	0 至 3，表示所在的槽号（Slot0 ...Slot3）
Chanel	0 至 7 ，表示 ADAM-5017H 通道号
PRange	返回输入范围代码

返回值:

None.

Init5017H

语法:

```
void Init5017(int Slot)
```

描述:

初始化 ADAM-5017H.

参数	**描述**
Slot	从 0 到 3

返回值:

None.

例子:

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

7.5.3 通信库 (COMM*.LIB)

checksum

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

描述:
计算串或缓冲区中数据的校检.

参数	描述
buffer	用户想计算校检的字符串.
Len	缓冲区中字符串的长度
seed	为安全和计算加入 checksum 中的值

返回值:
数据阵列的校检.

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

com_carrier

语法:

```
int com_carrier(void)
```

描述:

检测 COM 端口的载波信号 .

参数	描述
None	

返回值:

TRUE: 载波信号存在；
FALSE： 无载波信号

例子:

```
void main(void)
 {
  if( com_carrier() == TRUE )
  {
  }
}
```

com_clear_break, com_set_break

语法:

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

描述:

设置 COM 端口清除 BREAK 信号.
设置 COM 端口传送 BREAK 信号.

参数	描述
baseaddr	UART 地址, COM1=0x3F8, COM2=0x2F8.

返回值:

None.

com_clear_local_loopback, com_set_local_loopback

语法:

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

描述:

设置 COM 端口禁止 loopback 功能.
设置 COM 端口允许 loopback 功能.

参数	描述
baseaddr	UART 地址, COM1=0x3F8, COM2=0x2F8.

返回值:

None.

com_deinstall

语法:
void com_deinstall(void)

描述:
卸载通信驱动程序,不改变波特率或 DTR.

参数　　描述.
None

返回值:
None.

例子:
```
void main(void)
 {
  com_deinstall();
 }
```

com_disable_fifo, com_enable_fifo

语法:

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

描述:

设置 COM 端口禁止 fifo
设置 COM 端口允许 fifo

参数	描述
Baseaddr	UART 地址, COM1=0x3F8, COM2=0x2F8.
Triggerlevel	1, 4, 8, 14.

返回值:

0：成功.
-1: Fifo 不存在.
-10：失败.

com_flush_rx, com_flush_tx

语法:

```
void com_flush_rx(void) void com_flush_tx(void)
```

描述:

初始化发送和接收队列为空状态.

参数	描述
None	

返回值:

None.

例子:

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

com_get_line_status, com_set_line_params, com_get_modem_status

语法:

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

描述:

从 COM 端口控制寄存器读取状态字.
向 COM 端口控制寄存器写状态字.
从 COM 端口调制解调器状态寄存器读取状态字.

参数	描述
baseaddr	UART 地址, COM1=0x3F8, COM2=0x2F8.
Lineparams	请参考 UART 说明.

返回值:

请参考 16C550 UART 注册文档

例子:

None.

com_install

语法:

```
int com_install(int portnum);
```

描述:

安装通信驱动程序.

参数	描述
portnum	端口号,总是 1

返回值:

0：成功的安装.
1：驱动程序已经安装.
2：无效的端口编号.
3：指定端口没有 UART.

例子:

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

com_lower_dtr, com_raise_dtr

语法:

```
void com_lower_dtr(void)
void com_raise_dtr(void)
```

描述:

设置 COM 端口的 DTR 为低电平.
设置 COM 端口的 DTR 为高电平.

参数	描述
None.	

返回值:

None.

com_lower_rts, com_raise_rts

语法:

```
void com_lower_rts(unsigned baseaddr)
void com_raise_rts(unsigned baseaddr)
```

描述:

设置 COM 端口的 RTS 为低信号.
设置 COM 端口的 RTS 为高信号.

参数	描述
baseaddr	UART 地址, COM1=0x3F8, COM2=0x2F8.

返回值:

None.

例子:

```
#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 */
}
```

com_read_scratch_register, com_write_scratch_register

语法:

```
int com_read_scratch_register(unsigned baseaddr)
void com_write_scratch_register(unsigned baseaddr, int value)
```

描述:

从 COM 端口读取 scratch 寄存器值.
向 COM 端口 scratch 寄存器写.

参数	描述
baseaddr	UART 地址，COM1=0x3F8，COM2=0x2F8.
Value	一字节长度的整型值，范围 从 0 到 FF.

返回值:

请参考 16C550 UART 注册文档

com_set_format

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

描述:
设置 COM1 端口参数，数据长度,奇偶性和停止位

参数	**描述**
data_length	有效范围 从 5 至 8.
parity	0: 无奇偶，1: 奇校验 2: 偶校验
stop_bit	1: 1 个停止位， 2: 两个停止位

返回值:
None.

例子：
```
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);
}
```

com_set_parity

语法:

```
void com_set_parity(enum par_code parity, int stop_bits);
```

描述:

设置奇偶性和停止位.

参数	描述
parity	COM_NONE：8 数据位,无奇偶性 COM_EVEN：7 数据位,偶校验 COM_ODD：7 数据位,奇校验 COM_ZERO：7 数据比特,校验位 =零 COM_ONE：7 数据比特,校验位 =1
stop_bits	2 或 1

返回值:

None.

例子:

```
void main(void)
{
 com_set_parity(COM_NONE, 1);
}
```

com_set_speed

语法:
void com_set_speed(unsigned long speed);

描述:
设置 COM 端口的波特率 .

参数	**描述**
speed	波特率值.

返回值:
None.

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

com_rx

语法:
char com_rx(void)

描述:
返回接收缓冲区的值,如果为空返回 NULL。

参数　　描述.
None

返回值:
返回一个字符

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

com_tx

语法:
void com_tx(char c)

描述:
com_tx()传送单个字符并等待发送缓冲不为满时,然后将字符送入缓冲区.

参数	**描述**
c	发送的值.

返回值:
None.

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

com_rx_empty, com_tx_empty

语法:
int com_rx_empty(void) int com_tx_empty(void)

描述:
返回发送和接收队列状态.

参数　　描述
None

返回值:
Com_rx_empty(Void)：如果接收队列为空则返回 TRUE.
Com_tx_empty(Void)：如果发送队列为空则返回 TRUE.

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

com_tx_string

语法:
void com_tx_string(char *s)

描述:
com_tx_string() sends a string by repeatedly calling com_tx().
com_tx_string()通过重复调用 com_tx()发送字符串.

参数	**描述**
s	发送的字符串.

返回值:
None.

例子:
```
unsigned char name[]=“ADAM5510”
void main(void)
{
com_tx_string(name);
}
```

com_485_deinstall

语法:

```
void com_485_deinstall(void)
```

描述:

释放 RS-485 端口的中断服务程序，不改变波特率或 DTR.

参数　　描述毫不.

None

返回值:

None.

例子:

```
void main()
 {
/* Releases the interrupt register for use by the RS-485 port */
 com_485_deinstall();
}
```

com_485_flush_rx(), com_485_flush_tx()

语法:
void com_485_flush_rx(void) void com_485_flush_tx(void)

描述:
初始化发送和接收队列为空状态.

参数　　描述
None

返回值:
None.

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

com_485_install

语法:

```
int com_485_install(void)
```

描述:

设置 RS－485 口中断服务程序

参数	描述
None	

返回值:

0:成功安装，1:驱动程序已经安装

例子:

```
void main() {

int status;
status = com_485_install();
if( status ==0)
 printf("The allocation of COM2 port (RS-485) is OK !?);
else exit(0);
}
```

com_485_rx

语法:
char com_485_rx(void)

描述:
返回缓冲区中的值,如果缓冲区为空 NULL

参数　　描述
None

返回值:
返回一个字符

例子:
```
void main()
 {
 char C485data;
 C485data=com_485_rx();
 } Remarks:
```

com_485_set_format

语法:

```
void com_485_set_format(int data_length, int parity, int stop_bit)
```

描述:

设置 RS－485 口参数数据长度,奇偶性和停止位.

参数	描述
data_length	有效范围 为 5 至 8 位
parity	COM_NONE：8 数据位,无奇偶性 COM_EVEN：7 数据位,偶校验 COM_ODD：7 数据位,奇校验 COM_ZERO：7 数据比特,校验位 =零 COM_ONE：7 数据比特,校验位 =1
stop_bit	2 或 1

返回值:

None.

例子:

```
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);
}
```

com_485_set_speed

语法:
void com_485_set_speed(unsigned long speed)

描述:
设置 RS-485 的波特率

参数	**描述**
speed	波特率值.

返回值:
None.

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

com_485_rx_empty(), com_485_tx_empty()

语法:

```
int com_485_rx_empty(void) int com_485_tx_empty(void)
```

描述:

返回 COM2(RS-485)发送和接收排队的队列.

参数	描述
None	

返回值:

Com_485_rx_empty()：如果接收队列为空返回 TRUE.
Com_485_tx_empty()如果发送队列为空返回 TRUE.

例子:

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

com_485_tx

语法:
void com_485_tx(char c)

描述:
本函数发送单个字符 至 RS-485 端口的 Tx，直到上最后一位被传送至远程终端,然后设置 RTS 为 OFF.

参数	**描述**
c	发送的字符

返回值:
None.

例子:
```
void main()
 {
 com_485_tx(0x03);
 }
```

com_485_tx_string

语法:

```
void com_485_tx_string(char *s)
```

描述:

com_485_tx_string()通过反复调用 com_485_tx()发送字符串.

参数	描述
s	发送的字符串.

返回值:

None.

例子:

```
void main()
{
com_485_tx_string(“This is a string test.?”);
}
```

com_pgm_deinstall

语法:
void com_pgm_deinstall(void)

描述:
释放编程口中断服务程序，不改变波特率或 DTR.

参数　　描述
None

返回值:
None.

例子:
```
void main()
{
com_pgm_deinstall();
}
```

com_pgm_flush_rx(), com_pgm_flush_tx()

语法:

```
void com_pgm_flush_rx()
void com_pgm_flush_tx()
```

描述:

初始化编程口的接送和发收队列为空.

参数　　描述.

None

返回值:

None.

例子:

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

注： COM3 (编程口)发送数据使用查询方式，而非中断操作.

com_pgm_install

语法:

```
int com_pgm_install(void)
```

描述:

分配 COM3 中断服务程序.

参数	描述
None	

返回值:

0：成功安装， 1：驱动程序被已经安装

例子:

```
void main()
 {
 int status;
 status = com_pgm_install();
 if( status ==0) printf( " Programming port has been installed success-fully ! " );
 else exit(0);
}
```

com_pgm_rx

语法:
char com_pgm_rx(void)

描述:
从缓冲区返回下一个字符，如果缓冲区为空则返回 NULL

参数　　描述
None

返回值:
返回字符.

例子:
```
void main()
 {
 char CPGMdata;
 CPGMdata=com_pgm_rx();
}
```

com_pgm_rx_empty(), com_pgm_tx_empty()

语法:

int com_pgm_rx_empty(void) int com_pgm_tx_empty(void)

描述:

返回 COM3 发送和接收队列的状态.

参数	描述
None	

返回值:

Com_pgm_rx_empty()如果接收队列为空返回 TRUE.
Com_pgm_tx_empty()如果发送队列为空返回 TRUE.

例子:

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

com_pgm_set_format

语法:

```
void com_pgm_set_format(int data_length, int parity, int stop_bit)
```

描述:

设置 COM3（编程口）数据长度,奇偶性和停止位.

参数	描述
data_length	有效范围 为 5 至 8 位
parity	COM_NONE：8 数据位,无奇偶性 COM_EVEN：7 数据位,偶校验 COM_ODD：7 数据位,奇校验 COM_ZERO：7 数据比特,校验位 =零 COM_ONE：7 数据比特,校验位 =1
stop_bit	2 或 1

返回值:

None.

例子:

```
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);
}
```

com_pgm_set_speed

语法:
void com_pgm_set_speed(unsigned long speed)

描述:
设置 COM3(编程口的波特率。

参数	**描述**
speed	波特率值.

返回值:
None.

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

注：建议编程端口的波特率不超过 57600bps (包含 57600)，因为编程口的 UART 芯片不是标准 UART 芯片.

com_pgm_tx

语法:
void com_pgm_tx(char c)

描述:
本函数传送单个字符至编程口的 Tx，直到最后一个字符发送到远程终端，同时置 RTS 为 OFF

参数	**描述**
c	发送的字符.

返回值:
None.

例子:
```
void main()
{ com_pgm_tx(0x03);
com_pgm_tx(‘’d’);
}
```

com_pgm_tx_string

语法:

```
void com_pgm_tx_string(char *s)
```

描述:

com_pgm_tx_string()通过重复执行 com_pgm_tx()发送字符串.

参数	描述
s	发送的字符串

返回值:

None.

例子:

```
void main()
{
com_pgm_tx_string(“This is a string test”);
}
```

CRC16

语法:

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

描述:

Calculates the CRC 16-bit value of the string *data_p. Parameter Description *data_p The string which you want to calculate CRC code.
计算 *data_p 值的 CRC

参数	描述
data_p	待计算 CRC 代码的字符串.
length	字符串*data_p 的长度

返回值:

CRC16 代码.

例子:

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

modem_autoanswer

语法:
void modem_autoanswer(void)

描述:
设置调制解调器自动应答

参数　　描述
None.

返回值:
None.

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

modem_command

语法:
void modem_command(char *cmdstr)

描述:
发送 AT 命令串至调制解调器.

参数	**描述**
cmdstr	命令串;

返回值:
None.

例子:
```
void main(void)
 {
  modem_command(“atz”);/* initialize modem */
}
```

modem_command_state

语法:
void modem_command_state(void)

描述:
设置调制解调器到命令方式.

参数　　描述
None

返回值:
None.

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

modem_dial

语法:

```
void modem_dial(char *telenum)
```

描述:

连接调制解调器到指定电话号码.

参数	描述
telenum	通过调制解调器拨的电话号码

返回值:

None.

例子:

```
void main(void)
 {
/* COM port and modem initial OK */
modem_dial(“286“);
/* waiting to link */
}
```

modem_handup

语法:
void modem_handup(void)

描述:
设置调制解调器挂断电话.

参数　　描述
None

返回值:
None.

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

modem_initial

语法:

```
void modem_initial(void)
```

描述:

设置调制解调器到初始状态.由于 ADAM5510 系统结构,调制解调器仅仅能连接到 COM1.

参数　　描述

None

返回值:

None.

例子:

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