I-7188XB 系列中文用户手册

保固说明

泓格科技股份有限公司 (ICP DAS) 对于所生产的产品有瑕疵之材料，均保证原始购买者于交货日起保有为期一年的保固。

免责声明

泓格科技股份有限公司对于应用本产品所造成的损害并不负任何法律上的责任。本公司保留有任何时间未经通知即可变更与修改本文件内容之权利。本文所含信息如有变更，恕不予另行通知。本公司尽可能地提供正确与可靠的信息，但不保证此信息的使用或其他团体在违反专利或权利下使用。此处包含的技术或编辑错误、遗漏，概不负其法律责任。

版权所有

商标识别

本文件提到的所有公司商标、商标名称及产品名称分别属于该商标或名称的拥有者所有。

技术服务

如有任何问题，请与本公司客服联络，我们将尽速为您服务。

Email 信箱 ： service@icpdas.com

目 录

1. 简介

I-7188XB(D)可扩展嵌入式控制器系列，针对工业应用设计,可在恶劣环境下替代 PC 或 PLC 设备工作。I-7188XB(D)同样支持一组 I/O 扩展总线，可根据应用需求扩展不同功能的 I/O，例如：D/I，D/O，A/D，D/A，UART，Flash Memory，具有后备电池的 SRAM，AsicKey 及其它功能 I/O 等。大多数类型的 I/O 需求都可以通过这条总线来扩展实现。ICP DAS 为 I-7188XB(D)提供了大约 20 多个不同类型的 I/O 扩展板。搭配 I-7188XB(D)内部是可编程的，可使 I-7188XB(D)适用于各种不同应用功能的需求。

产品清单

除该产品用户手册外，产品包装内还包含如下项目：

- 一个 I-7188XB(D) 模块
- 一根下载线 (CA0910)
- 一张配送光盘，光盘内容有软件驱动和电子版用户手册
- 一张版本说明

注: 若以上项目有任何损害或遗失，请联系购买产品的经销商。推荐保存好所有配件及硬纸盒，以备未来可能的运输使用。

1.1 硬件特点

- 嵌入式 80188 CPU, 40M 或兼容
- 内置 RTC, NVRAM 和 EEPROM
- 内置 2 个 COM 口：COM1 和 COM2
- 64 位唯一硬件序列号
- 串口设备支持中断和 1K 队列输入/输出缓冲。
- 支持 I/O 扩展总线接口(仅能加载一个扩展板)
- 一个数字量输入通道
- 一个 OC 门输出通道
- RS-485 口内置自适应 ASIC 控制器
- 可选五位七段 LED 显示
- 内置 ICP DAS 提供的 MiniOS7 操作系统
- 程序下载口： COM1

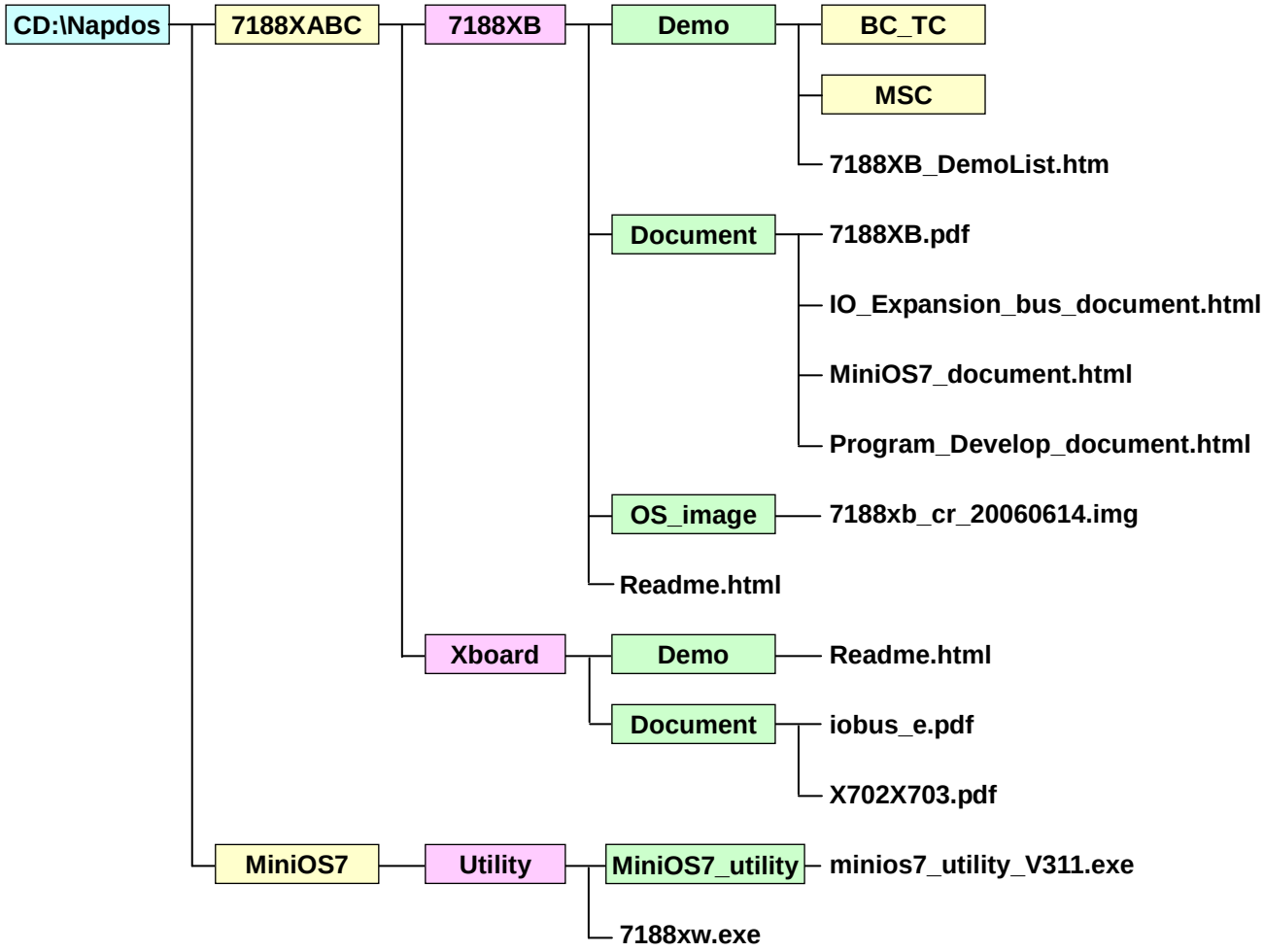
1.2 硬件规格

CPU 模块	
CPU	80188 CPU, 40MHz 或兼容
SRAM	I-7188XB 有 256K 字节 I-7188XB/512 有 512K bytes 字节
Flash	512K 字节
EEPROM	2K 字节
NVRAM	31 字节
RTC (实时时钟)	支持
硬件序列号	支持
内置看门狗时钟	支持
通讯接口	
COM 1	RS-232/RS-485 (默认 RS-232)
COM 2	RS-485 (3000V 隔离)
COM 3	无
COM 4	无
以太网口	无
数字量输入	
输入通道	1
Contact	Dry 支持干接点
低电平	接地
高电平	悬空
数字量输出	
数字量输出通道	1
输出类型	集电极开路
最大负载电流	100mA
负载电压	+30V/DC Max.
LED 显示器	
1 LED 用于 电源/通讯 指示器	
5 个 7 段 LED (只有 I-7188XBD 有)	
尺寸	
123mm x 72mm x 33mm	
运行环境	
运行温度	-25°C to +75°C
存储温度	-30°C to +80°C

相对湿度	10 to 90% RH (非凝露)
电源	
电源需求	10 to 30V/DC (非固定)
功率消耗	2.0W for I-7188XB 3.0W for I-7188XBD

1.3 软件和文档资料

在下列树状结构中显示的是 I-7188XB(D)配送光盘中文档和软件的位置。通过这个树状结构可以很快的找到相应的文件。



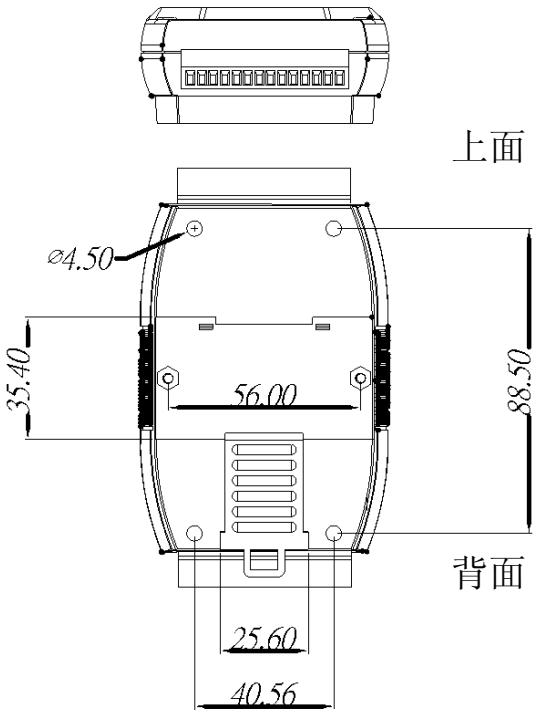
上面列出的文档和软件同样可以通过 ICP DAS 站点：http://ftp.icpdas.com/pub/cd/8000cd/napdos 找到。光盘和站点中的所有文档和软件的文件夹位置一样。

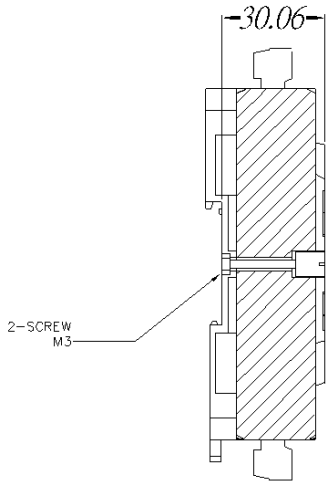
用户可以参考 CD:\Napdos\7188XABC\Xboard\Document\iobus_e.pdf 了解 I-7188XB(D)I/O 扩展总线相关的详细资料。

在继续之前，建议你阅读 CD:\Napdos\7188XABC\7188XB\Readme.html 最新的信息将包含在这个文件中。

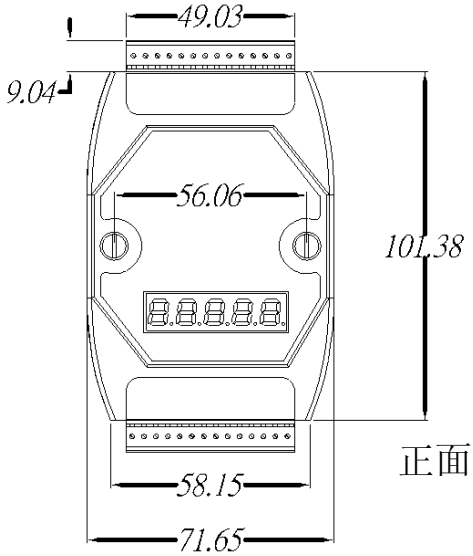
1.4 硬件资料

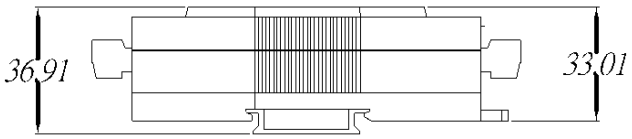
1.4.1 I-7188XB(D)的图表和尺寸



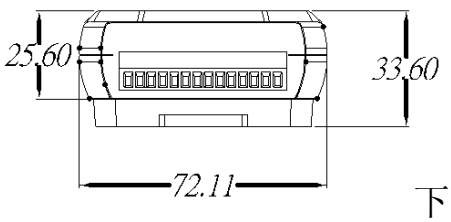


单位: mm

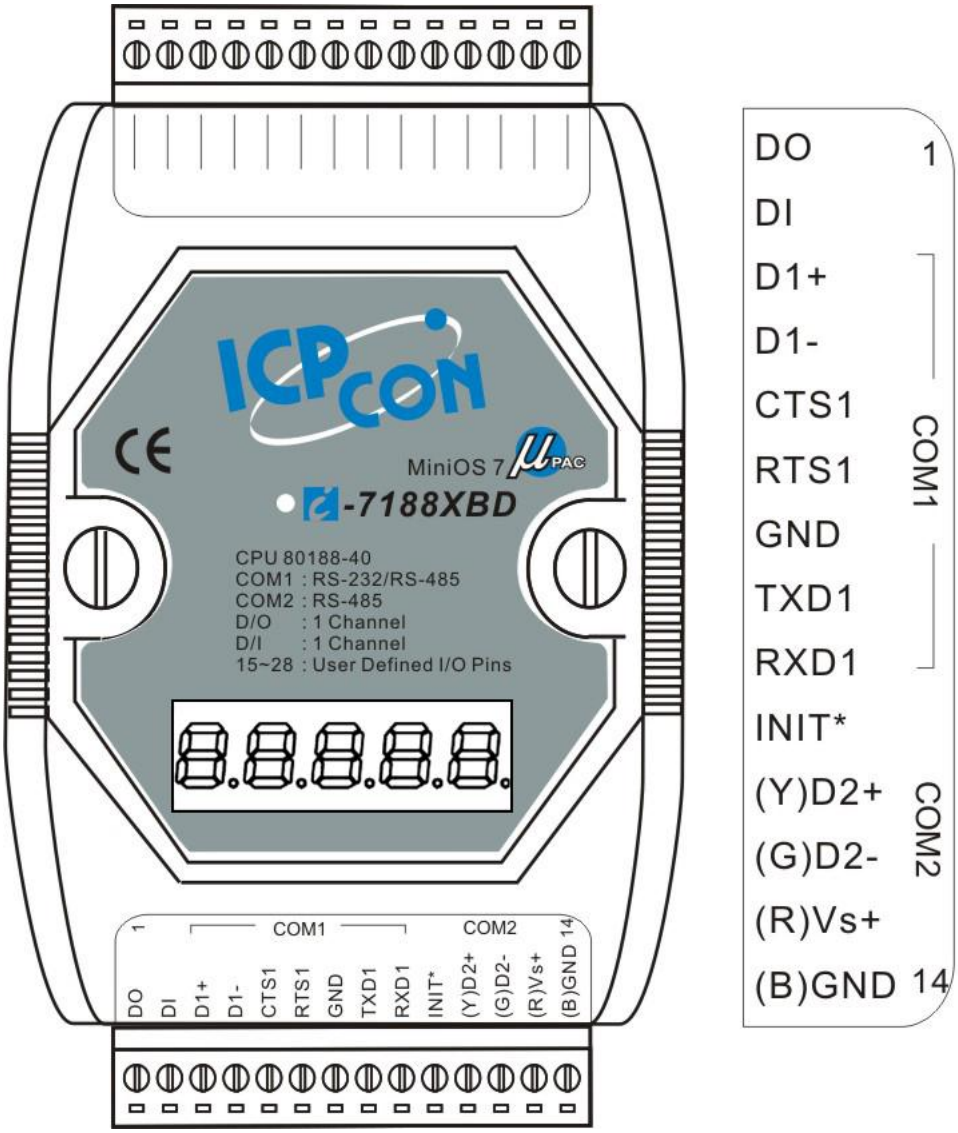




DIN-RAIL 装备支架



1.4.2 引脚定义



14 针端子板引脚分配如下：

引脚	名字	描述
1	DO	数字量输出, 100mA, 30V Max.
2	DI	数字量输入, 3.5V ~ 30V
3	D1+	COM1 (RS-485)的 DATA+
4	D1-	COM1 (RS-485)的 DATA-
5	CTS1	COM1 (RS-232)的 CTS
6	RTS1	COM1 (RS-232)的 RTS
7	GND	COM1 (RS-232)的 GND

8	TXD1	COM1 (RS-232)的 TXD
9	RXD1	COM1 (RS-232)的 RXD
10	INIT*	初始化引脚
11	D2+	COM2 (RS-485)的 DATA+
12	D2-	COM2 (RS-485)的 DATA-
13	+VS	电源 V+ (+10 to +30V/DC)
14	GND	电源 GND

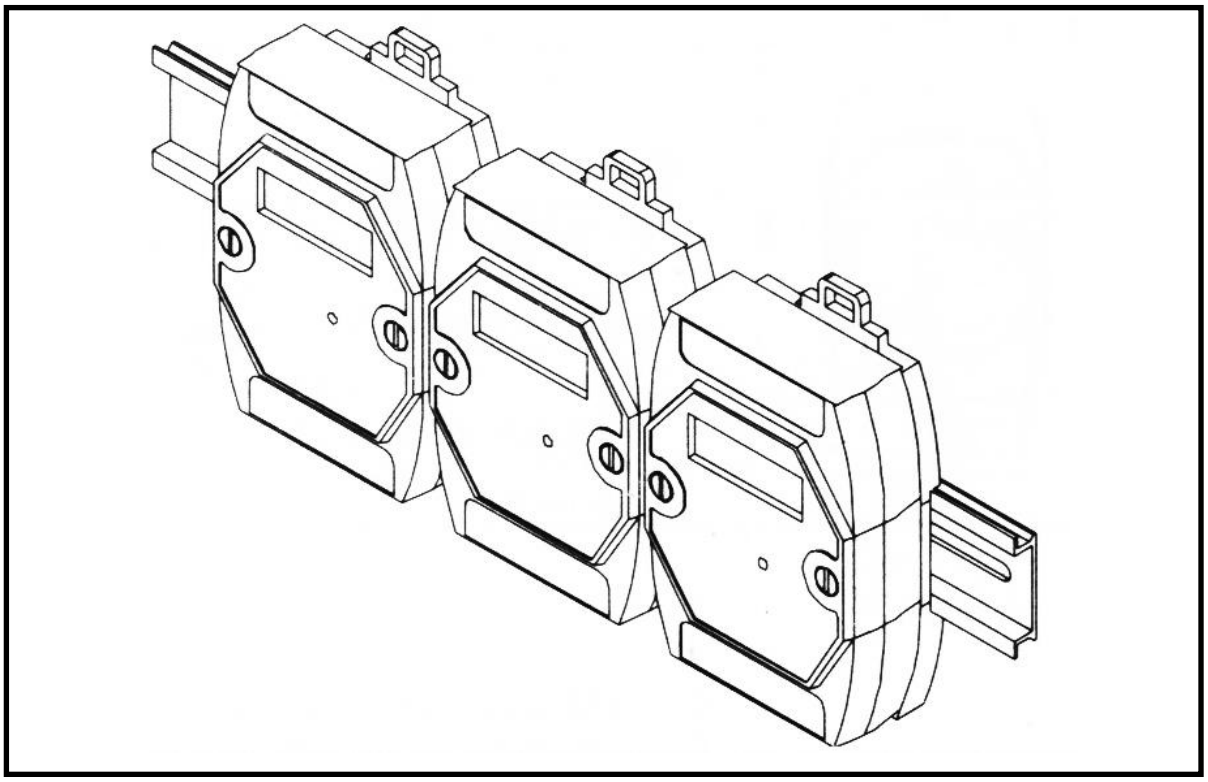
注: COM1 可以当作 RS-232 或 RS-485，但是不推荐同时使用 RS-232 和 RS-485。

14 针端子板引脚分配如下：

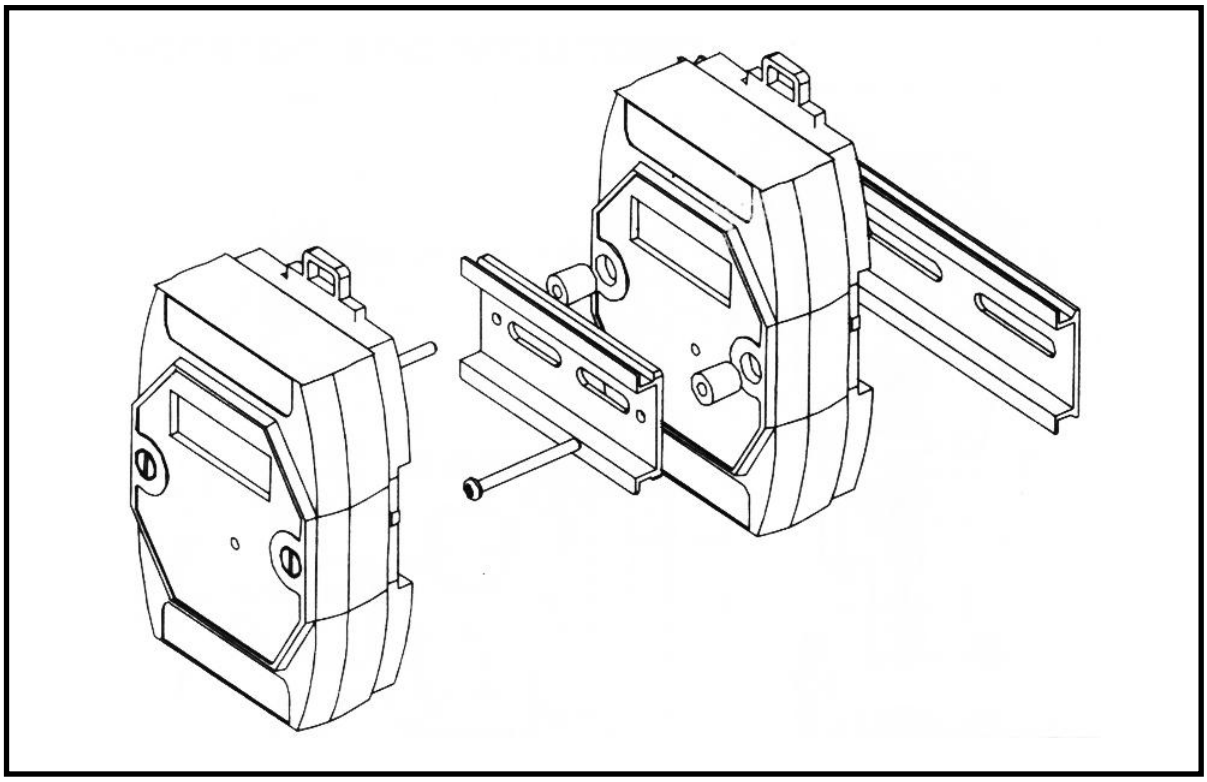
引脚	名字	描述
15	Pin 15	用户定义针 15
16	Pin 16	用户定义针 16
17	Pin 17	用户定义针 17
18	Pin 18	用户定义针 18
19	Pin 19	用户定义针 19
20	Pin 20	用户定义针 20
21	Pin 21	用户定义针 21
22	Pin 22	用户定义针 22
23	Pin 23	用户定义针 23
24	Pin 24	用户定义针 24
25	Pin 25	用户定义针 25
26	Pin 26	用户定义针 26
27	Pin 27	用户定义针 27
28	Pin 28	用户定义针 28

1.4.3 I-7188XB(D)安装

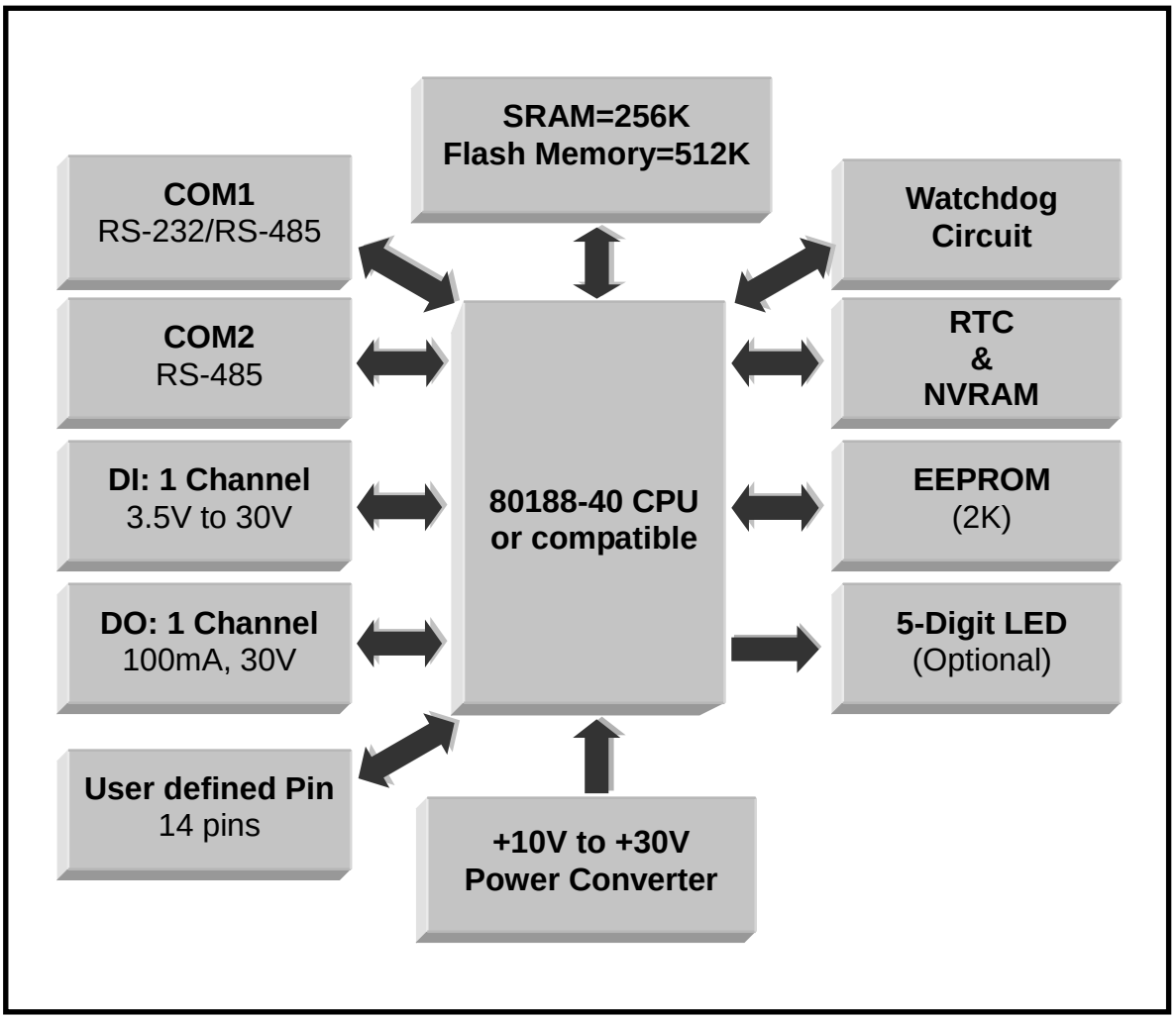
1. 导轨安装



2. 叠加安装

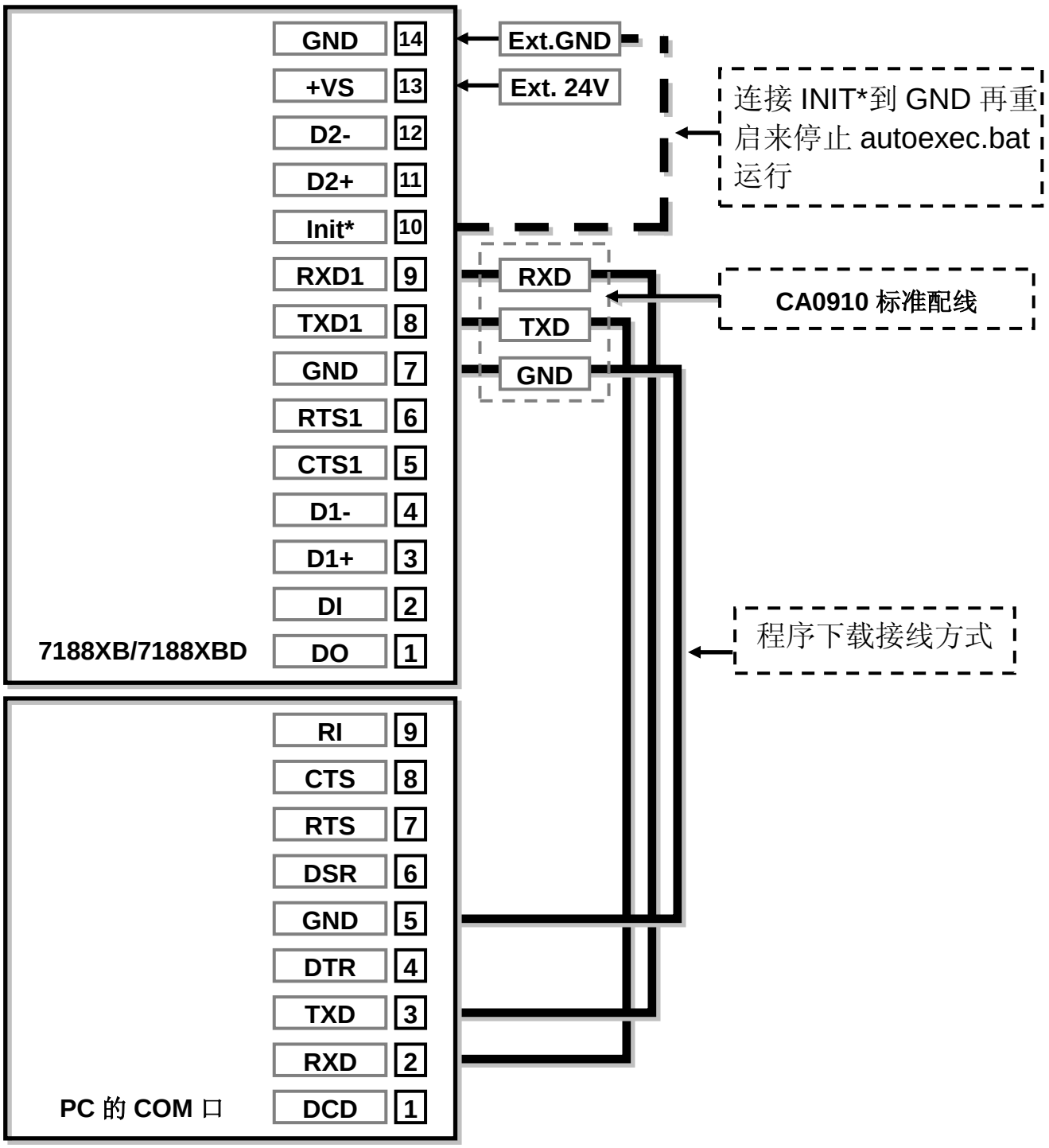


1.4.4 硬件结构图


SRAM=256K
Flash Memory=512K
COM1
RS-232/RS-485
COM2
RS-485
DI: 1 Channel
3.5V to 30V
DO: 1 Channel
100mA, 30V
User defined Pin
14 pins
80188-40 CPU
or compatible
+10V to +30V
Power Converter
Watchdog
Circuit
RTC
&
NVRAM
EEPROM
(2K)
5-Digit LED
(Optional)

1.4.5 端口接线图

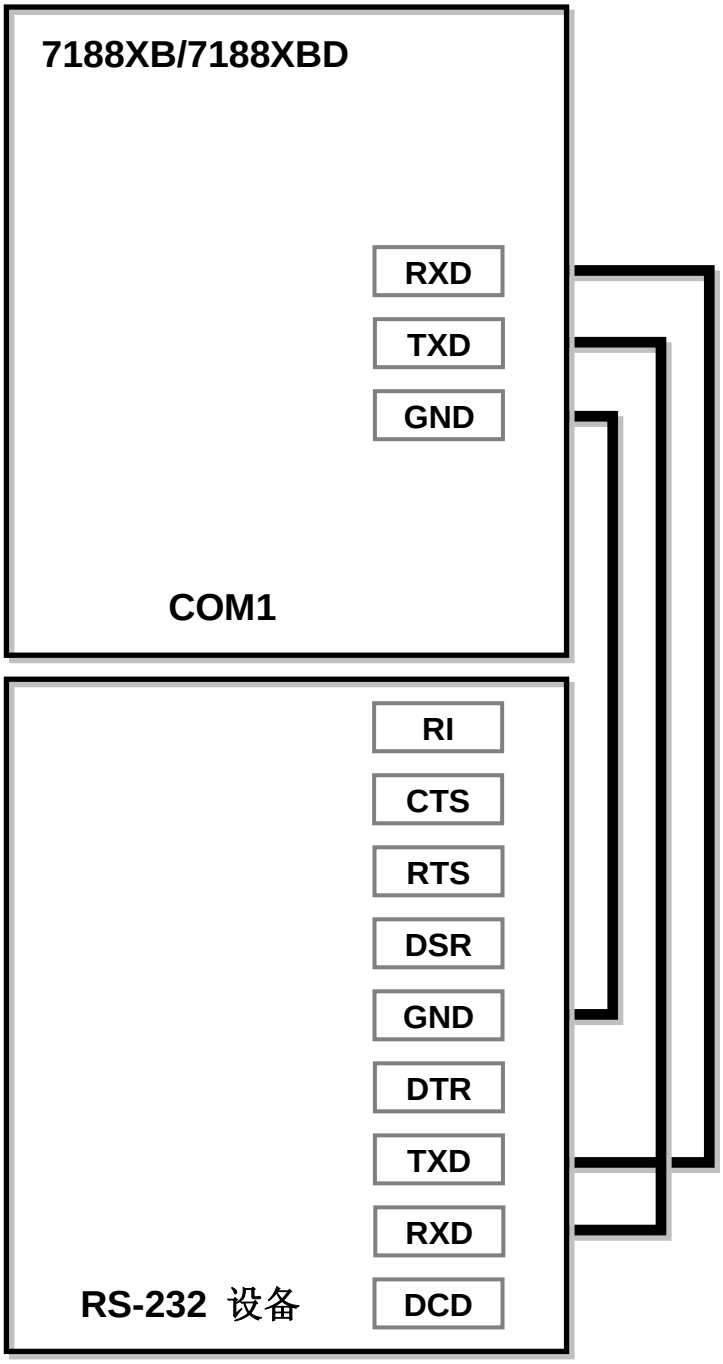
程序下载



注：使用 3 线制连接下载数据线：

- 连接线-1，标有 RX 的端子连接 I-7188XB(D)的 pin-9
- 连接线-2，标有 TX,的端子连接 I-7188XB(D)的 pin-8
- 连接线-3，标有 GND 的端子连接 I-7188XB(D)的 pin-7
- 连接下载线 DB-9 到 PC 的 COM 口

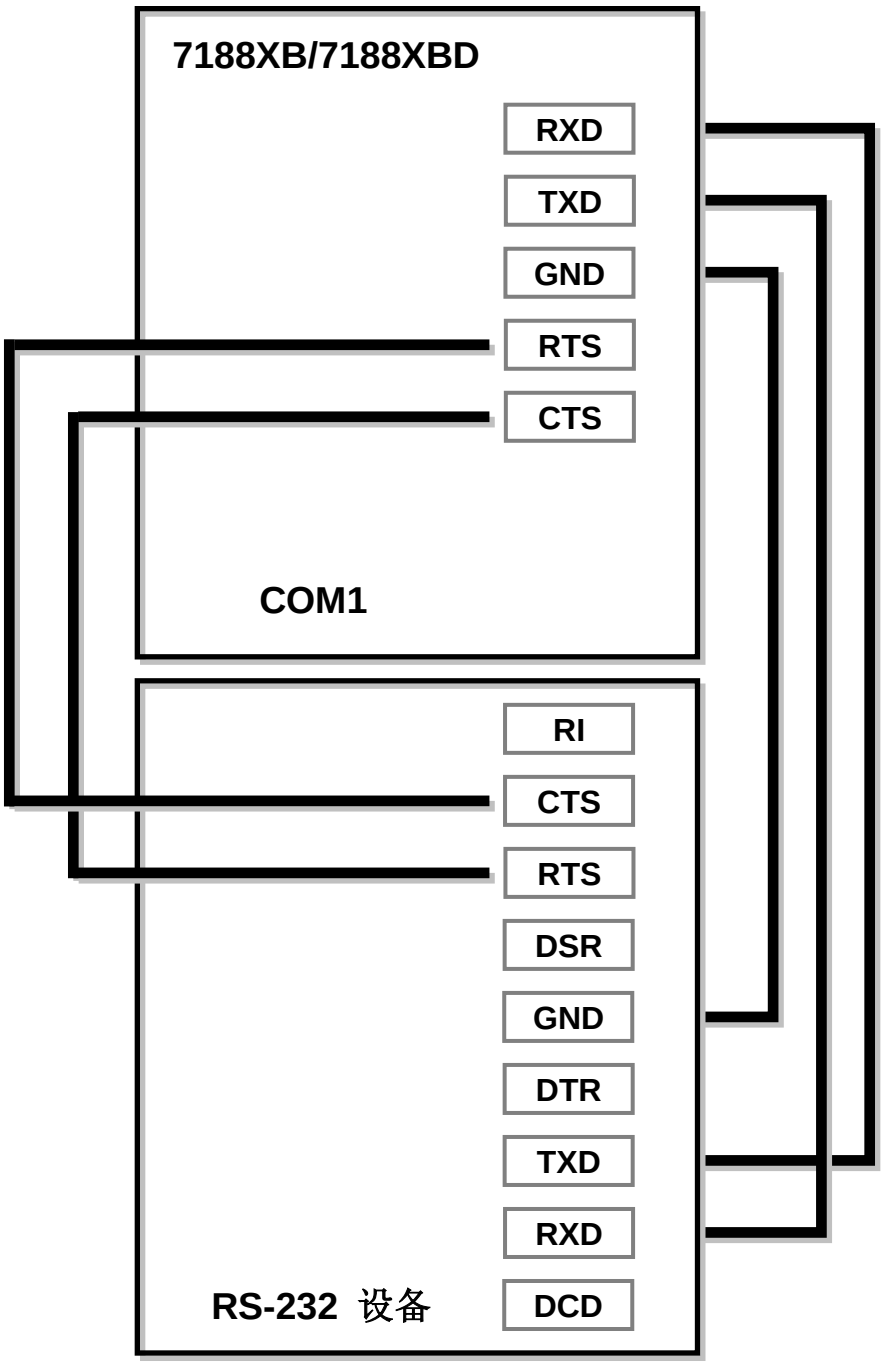
使用 3 线制 RS-232



注: 3 线制接法如下：

- 连接 RXD 到 RS-232 设备的 TXD
- 连接 TXD 到 RS-232 设备的 RXD
- 连接 GND 到 RS-232 设备的 GND

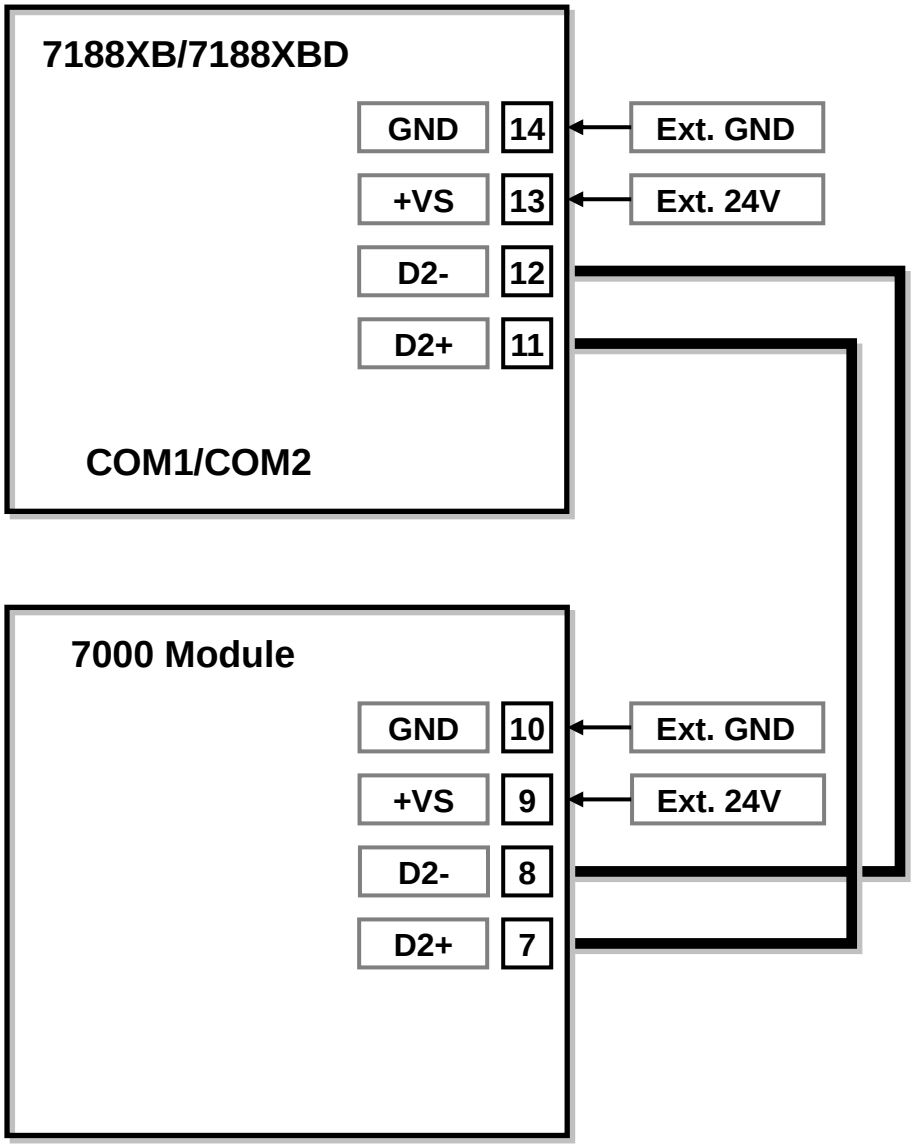
使用 5 线制 RS-232



注: 5 线制接法如下：

- 连接 RXD 到 RS-232 设备的 TXD
- 连接 TXD 到 RS-232 设备的 RXD
- 连接 RTS 到 RS-232 设备的 CTS
- 连接 CTS 到 RS-232 设备的 RTS
- 连接 GND 到 RS-232 设备的 GND

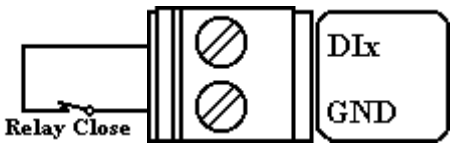
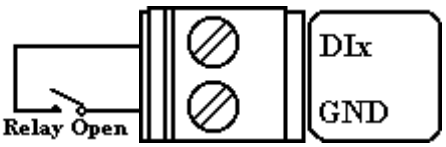
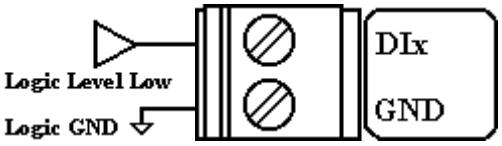
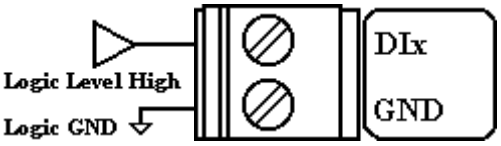
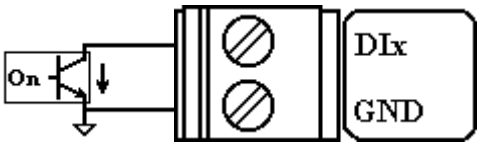
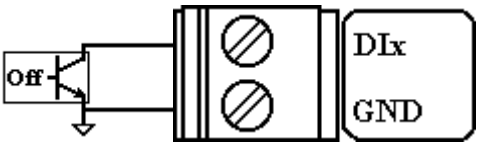
使用 RS-485 口



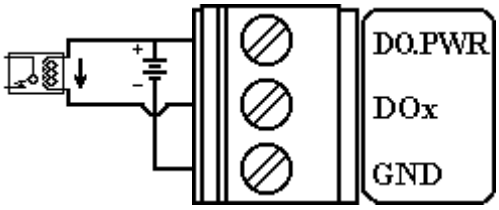
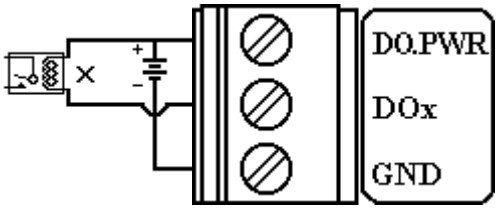
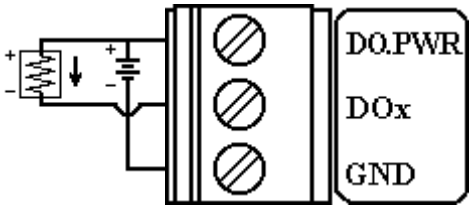
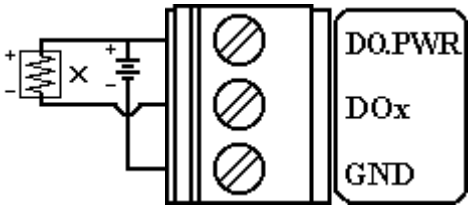
注: RS-485 接口可在不需要中继的情况下连接 256 个 I-7000 系列模块块。

1.4.6 DI/DO 接线方式

数字量输入接线方式

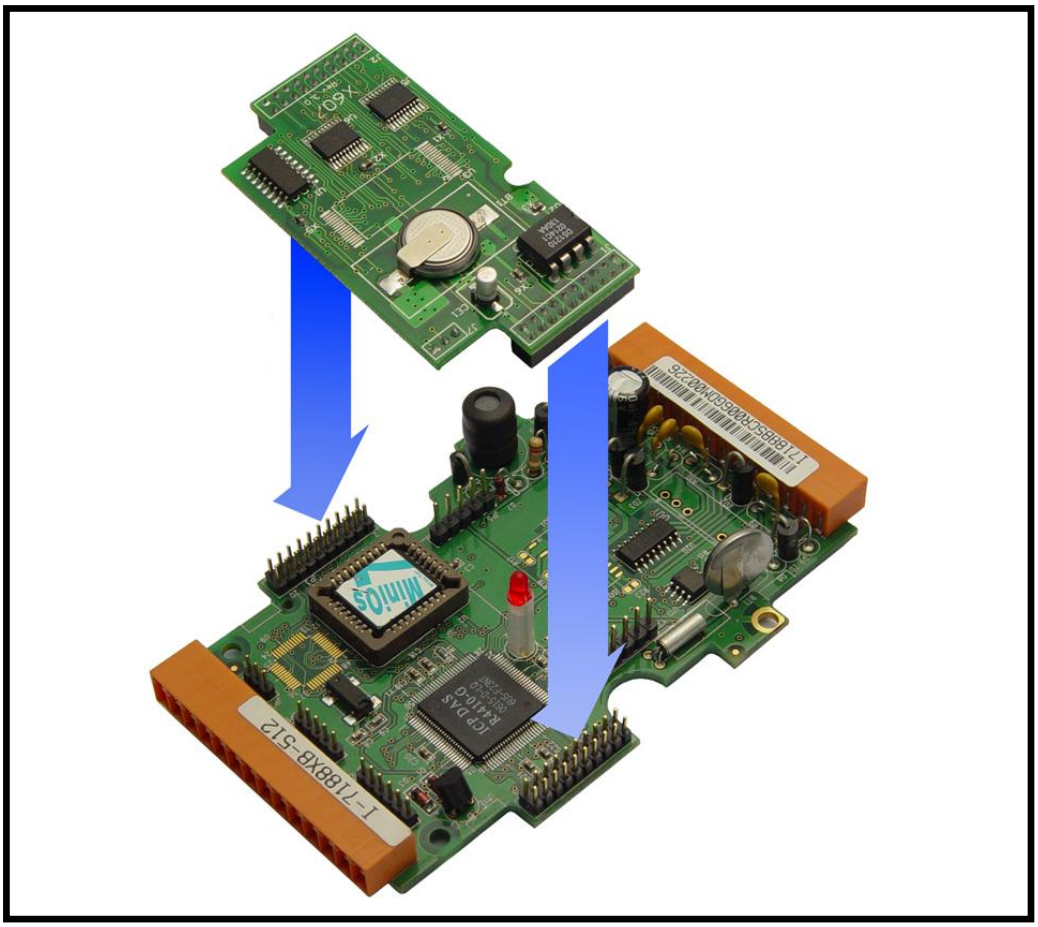
输入类型	ON State DI value as 0	OFF State DI value as 1
Relay Contact		
TTL/CMOS Logic		
Open Collector		

数字量输出接线方式

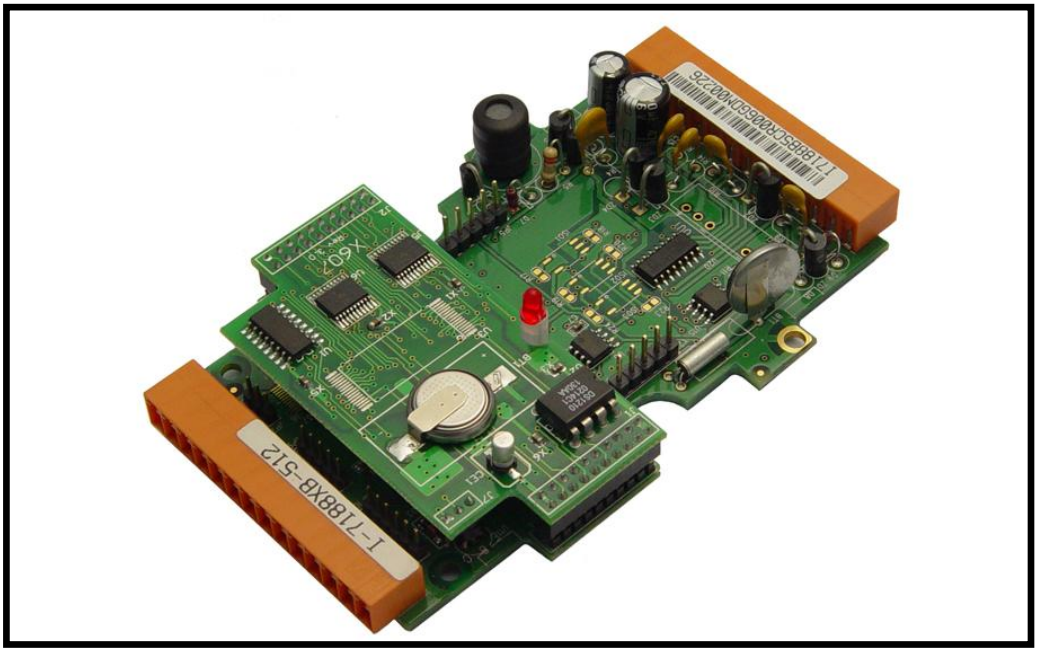
Input Type	ON State DO value as 1	OFF State DO value as 0
Drive Relay		
Resistance Load		

1.4.7 安装 I/O 扩展背板

安装前：

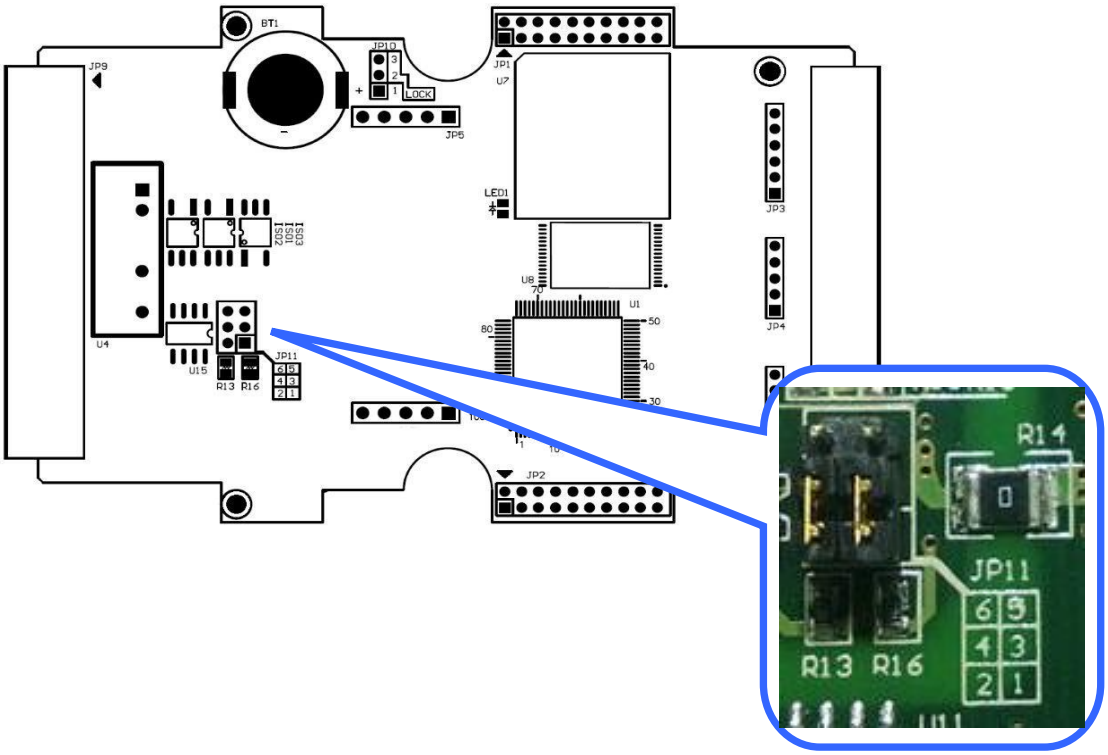


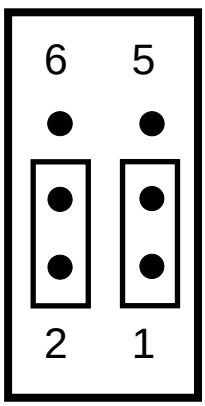
安装后：



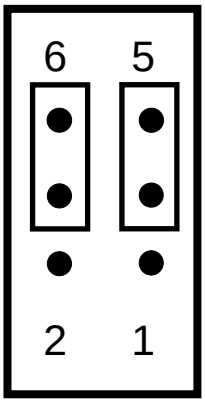
1.4.8 使用跳线设置 RS-485 上拉/下拉电阻

使用跳线 (Jumper)设定是否使用上拉/下拉电阻. 跳线的设定如下：





启用上拉/下拉电阻



取消上拉/下拉电阻

注意:

1. 跳线的设定只有针对 **COM2.**
2. 硬件版本 **3.5** 以上才有支持此跳线的功能. **(**硬件版本信息写在 **PCB** 的某处。**)**

2. 快速开始

2.1 软件安装

步骤 1：插入配送光盘到 CD 驱动器

步骤 2：复制 **CD:\Napdos\7188XABC**下面的 7188XB 的文件夹到 PC 的硬盘上。

步骤 3: 安装 MiniOS7 Utility.
找到并执行 CD:\NAPDOS\MINIOS7\UTILITY\MiniOS7_utility\或 http://ftp.icpdas.com/pub/cd/8000cd/napdos/minios7/utility/minios7_utility/文件夹中的 **minios7_utility_v311.exe**

步骤 4：复制 CD:\Napdos\MiniOS7\utility\文件夹中的 **7188xw.exe** 文件到目录中例如：**C:\Windows**

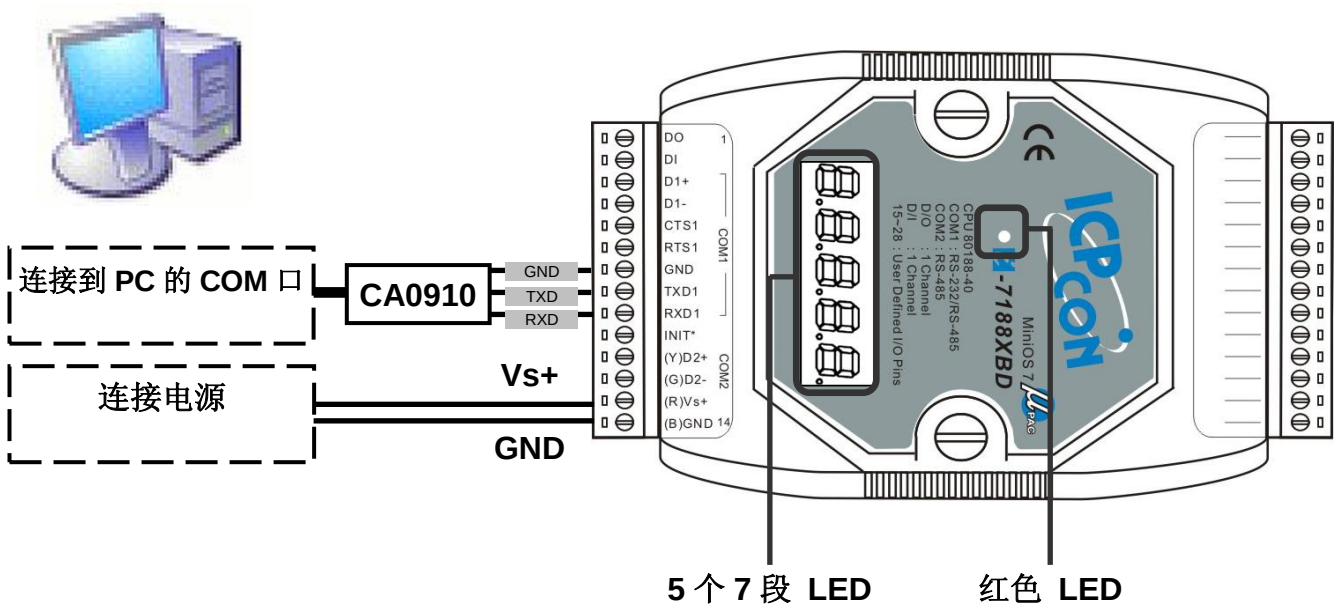
在所有软件复制到 PC 后，7188XB 文件夹内容如下：

注: 7188xw.exe 文件是 I-7188XB(D)和 PC 之间的桥梁。因此，7188xw.exe 文件需要拷贝到 “C:\Windows\”文件夹或其它任意可以执行的位置。

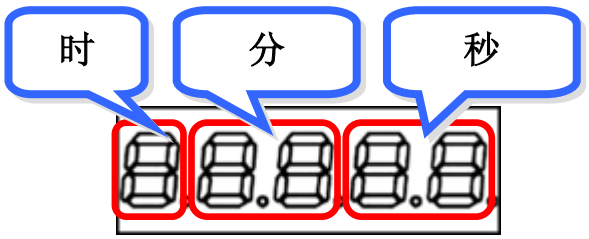
2.2 连接下载线到 PC

步骤 1: CA0910 下载线连接到 I-7188XB(D)的 COM1 和 PC 的 COM 口，下图显示接线方式。

步骤 2: 连接电源(Vs+, GND) 到 I-7188XB(D). Vs+ 可以接受的范围是 +10V 到 +30V DC.

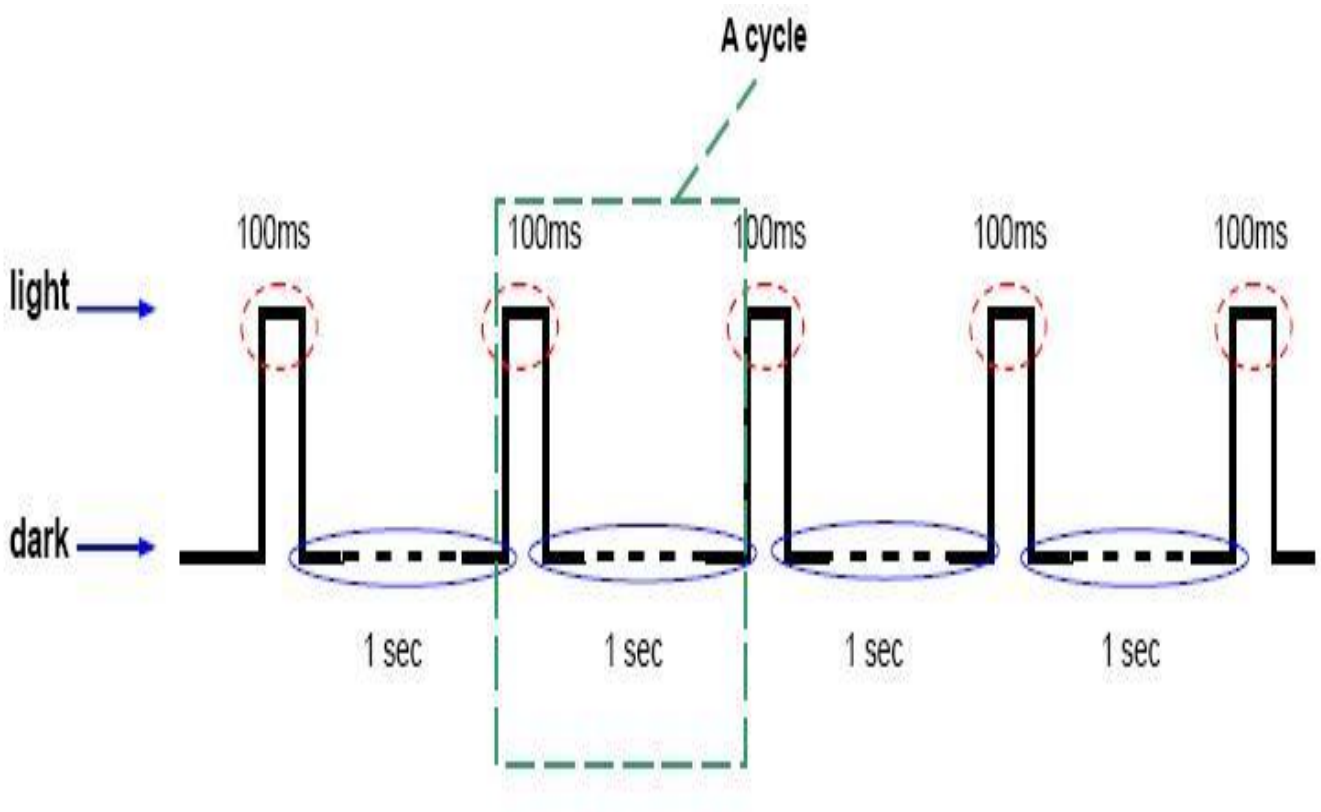


步骤 3:连接电源后 7 段 LED 将不断的显示下列信息



如果没有显示模块块的版本，请继续下个步骤

步骤 4: 检查红色 LED 不断的一秒钟闪烁 1 次，图表显示如下：



注意： I-7188XBD 带有 5 位 7 段 LED 显示。

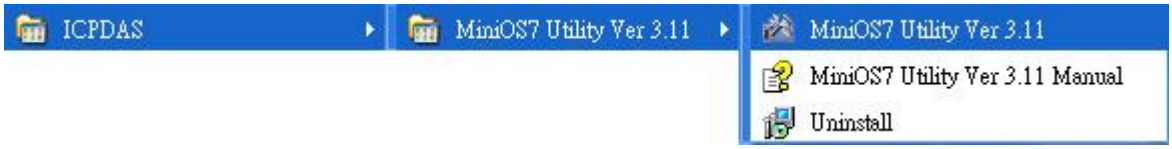
2.3 下载程序到 I-7188XB(D)

在使用 MiniOS7 Utility 之前，确定下载线于 PC 和 I-7188XB(D)已连接，且 I-7188XB(D)中没有程序正在运行。如何连接 PC 和 I-7188XB(D)COM1 的详细资料请参考章节 1.4.5 中的 程序下载接线图。

注: 7188xw.exe 文件同样也可以像 MiniOS7 Utility 一样用来下载程序。参考附录 B:MiniOS7 Utility 和 7188xw 有详细介绍。

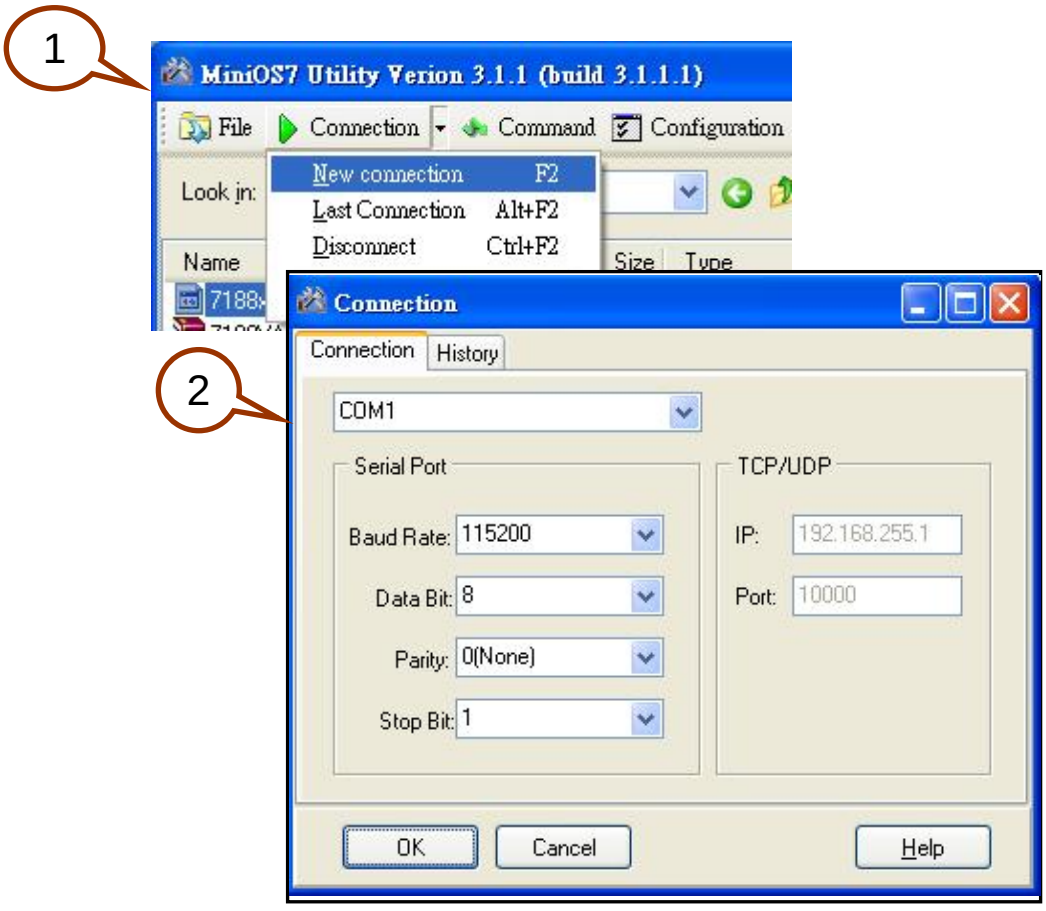
程序下载顺序如下 (以章节 2.1 中安装 MiniOS7 Utility 版本 3.11 为例):

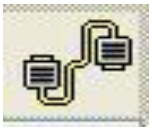
步骤 1: 通过 Windows 开始菜单定位到 Programs/ICPDAS/MiniOS7 Utility Ver 3.11 , 按下 **MiniOS7 Utility Ver3.11** 执行.



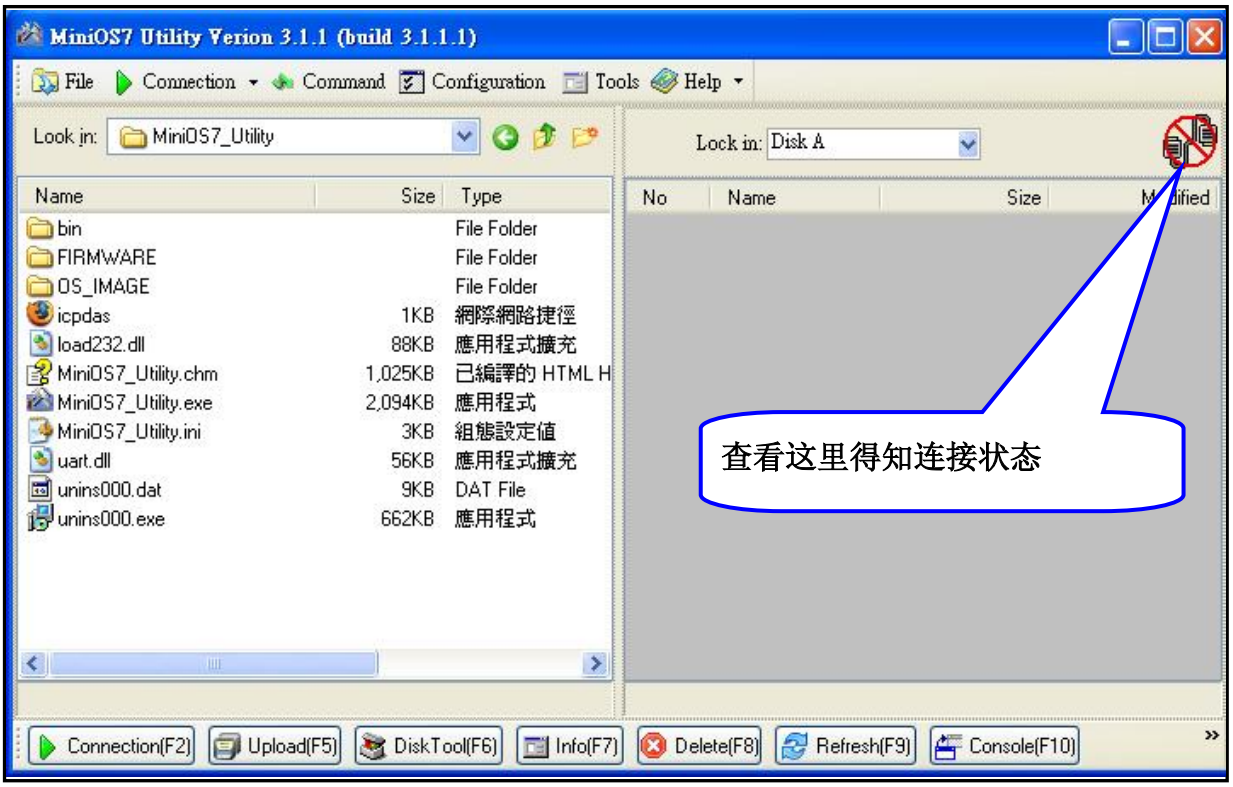
步骤 2: 按下 

并选择 “New connection”。选择正确的 COM 口并设定其它参数。 单击 **OK** 按钮自动搜索模块。

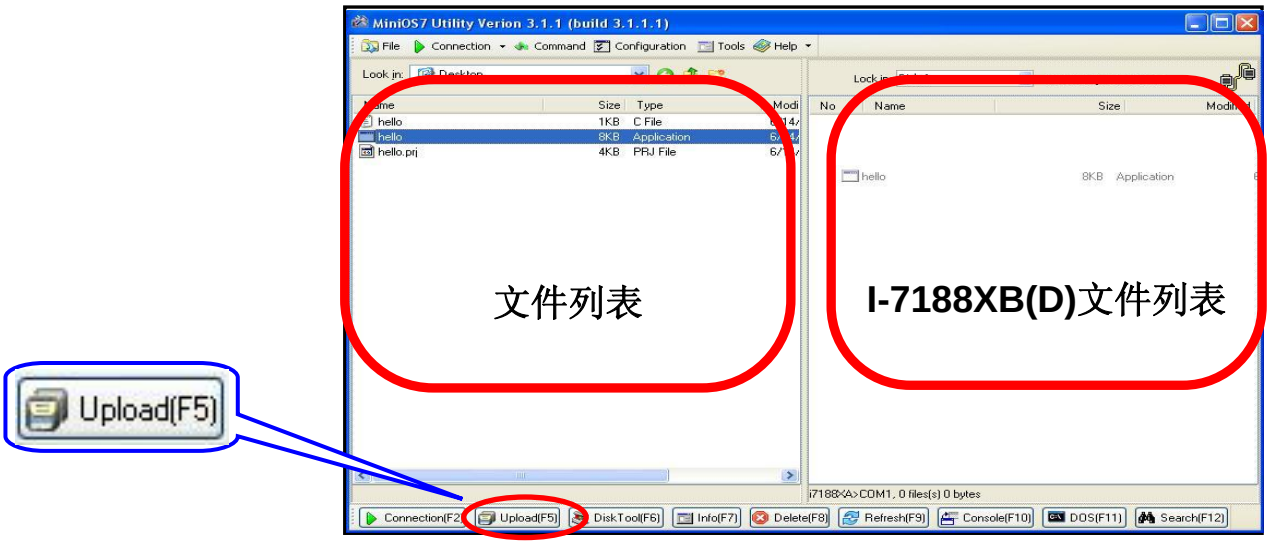


步骤 3: 观察如果 MiniOS7 Utility 连接到 I-7188XB. 连接图标是 

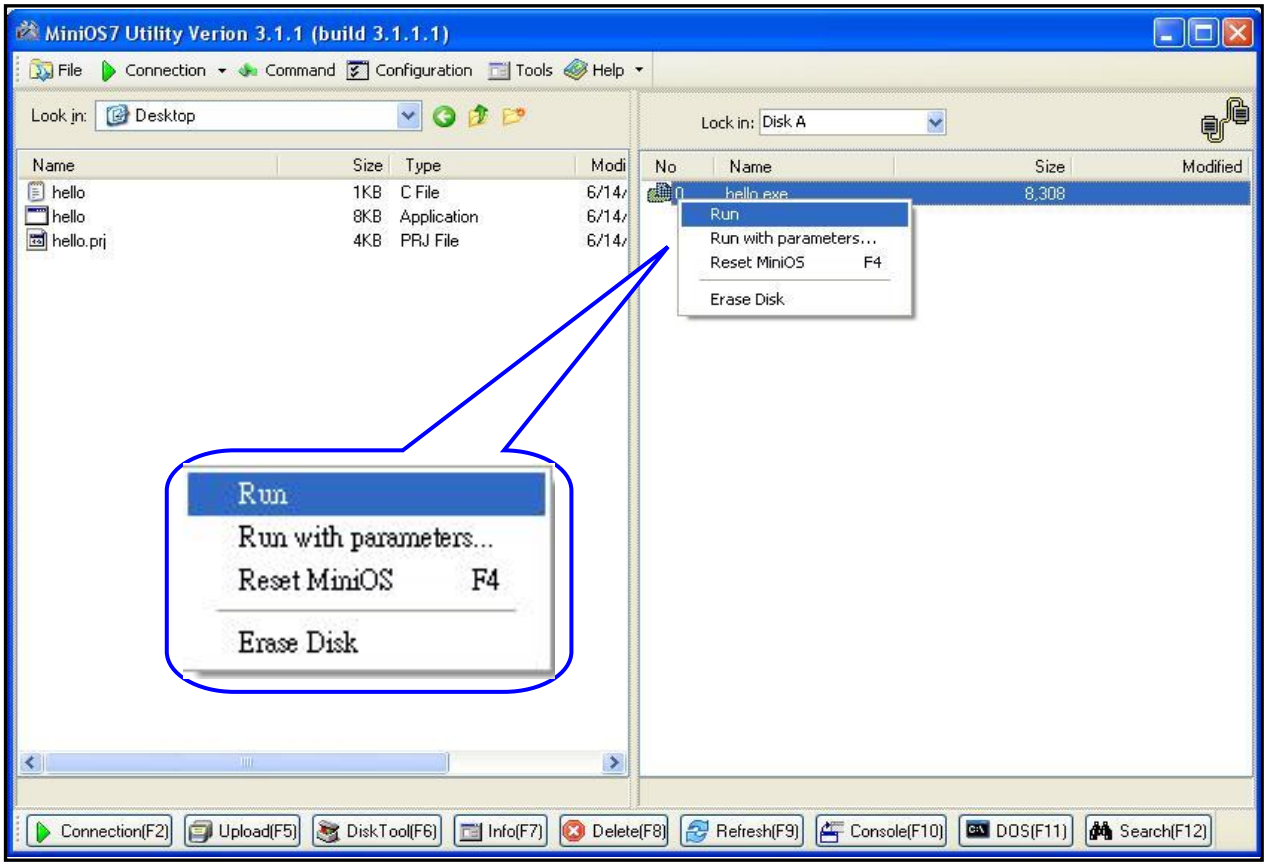
如果连接有问题连接图标是 



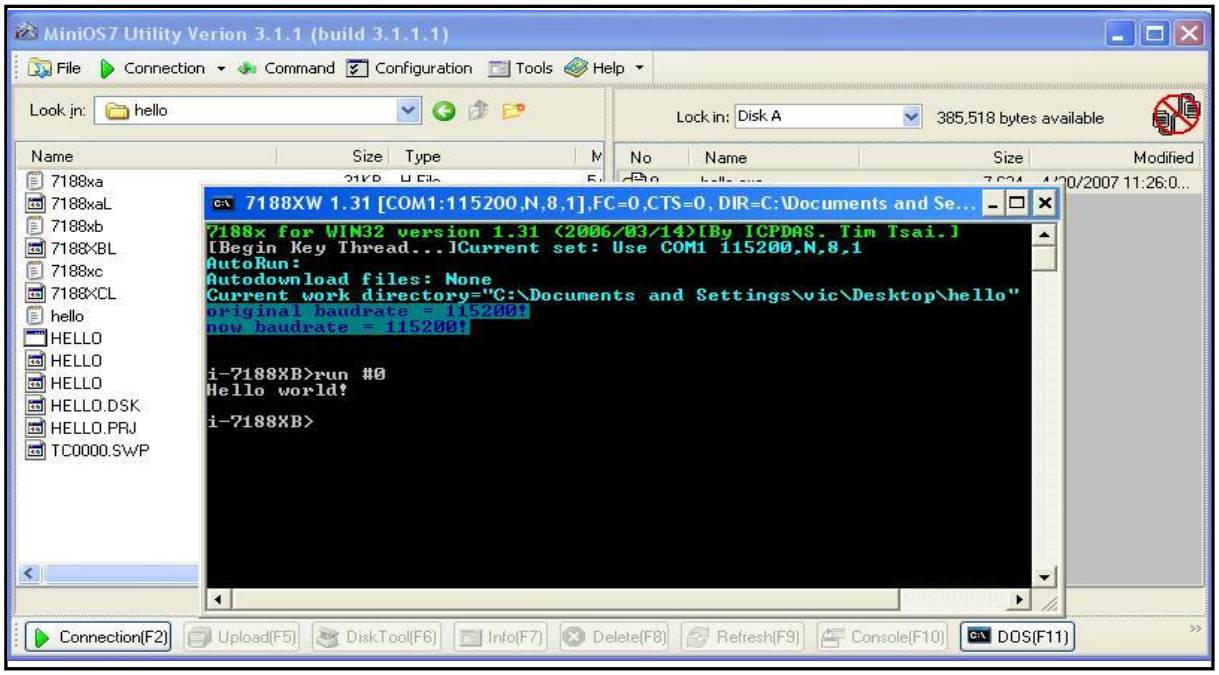
步骤 4: 选中左边文件列表中需要下载的文件，单击按钮“Upload(F5)”下载文件到模块或拖拽文件到右边。



步骤 5: 选定文件并点击鼠标右键选择 Run 执行程序



步骤 6: 程序结果将被显示在 7188xw 窗口中。



注意：当再次执行下载操作时，7188xw 窗口必须关闭。

Hello.c 文件内容如下：

```
#include “7188xb.h”      /* 包含头文件去使用 7188xbl.lib 函数 */
void main(void)
{
   InitLib();                          /* 初始化 7188xb 库 */

   Print("Hello world!\r\n");       /*在控制台中打印信息*/
}
```

2.4 MiniOS7 升级

ICP DAS 将不断的增加 MiniOS7 的新功能。因此推荐你定期检查 ICP DAS 站点获得最新的 MiniOS7 版本。

注:更多的 MiniOS7 详细信息，请参考附录 A：什么是 MiniOS7.

MiniOS7 Utility 提供非常简单的方式去更新 MiniOS7。更新过程如下：

步骤 1: 得到最新版本的 MiniOS7 镜像文件。

镜像文件名格式: **TTYYMMDD.img**

TT: 产品类型

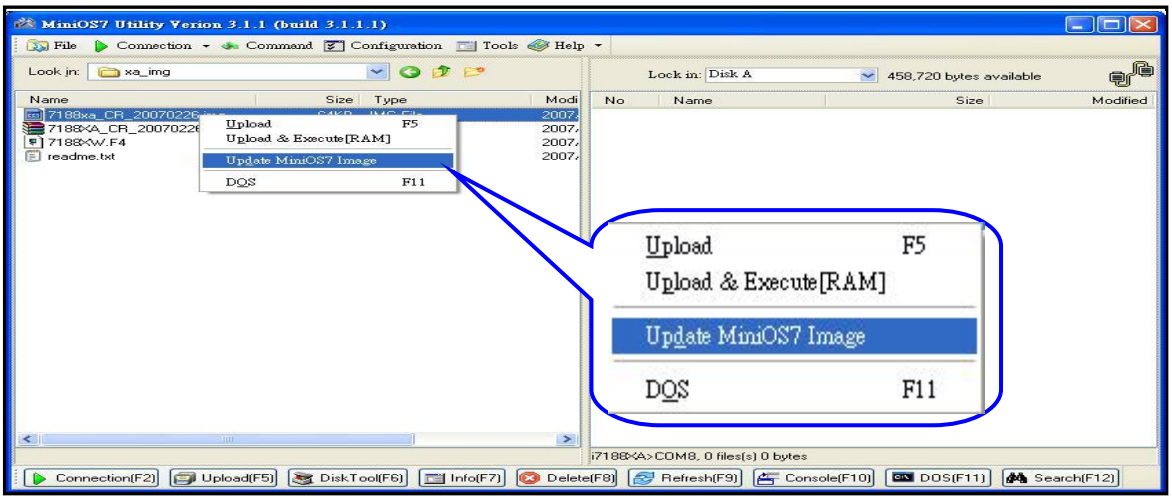
YY: 镜像文件发布年份

MM: 镜像文件发布月份

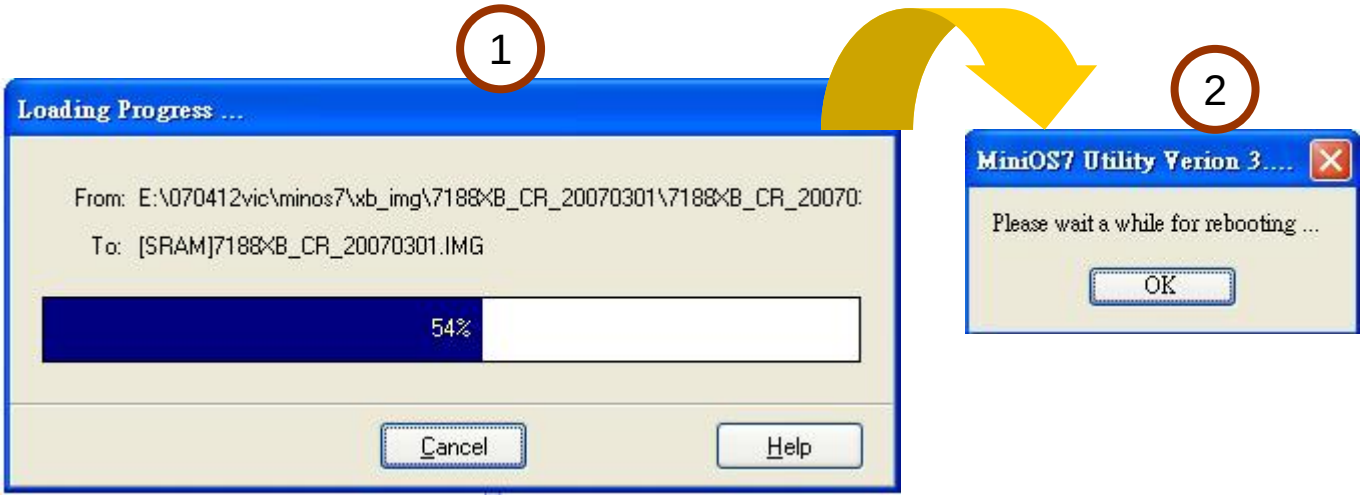
DD: 镜像文件发日期

注: MiniOS7 镜像文件在配送光盘中 CD:\NAPDOS\MiniOS7\目录中。最新版本的 MiniOS7 在 ICP DAS 站点：
http://ftp.icpdas.com/pub/cd/8000cd/napdos/7188xabc/7188xb/os_image/

步骤 2: 执行 MiniOS7 Utility. 参考章节 2.3 中的步骤 2 去连接模块。在左边选择要更新的 MiniOS7 镜像文件。单击鼠标右键选择“Update MiniOS7 Image”。

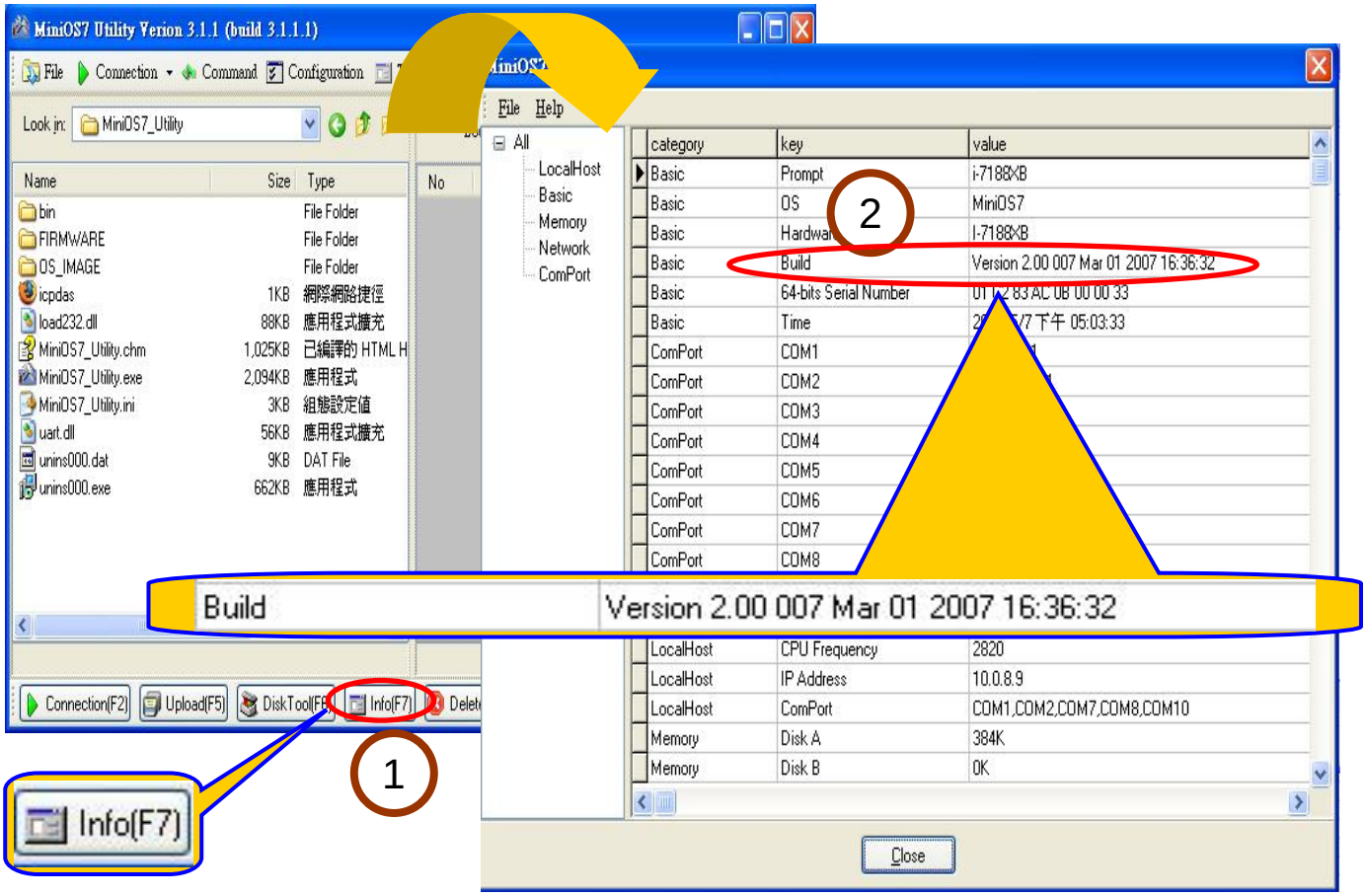


步骤 3: 更新过程将消耗 10 秒钟。如果 MiniOS7 更新成功，将弹出确认对话框。



步骤 4: 点击 

按钮查看 “Build”项目检查 MiniOS7 版本号。如下图所示：



注: 此外使用 7188xw.exe 同样可以像 MiniOS7 Utility 去更新 MiniOS7。参考附录 B：MiniOS7 Utility 和 7188xw 下载程序。

3.编写第一个程序

3.1 库文件

如下提供两个 I-7188XB(D)模块功能库：

- **7188xbs.lib** 适用于小内存模式
- **7188xbl.lib** 适用于大内存模式

两个库文件均可适用于 TC, BC++, MSC 和 MSVC++编译器。所有函数声明都在头文件 **7188XB.h** 中。

最新库文件位置：
http://ftp.icpdas.com/pub/cd/8000cd/napdos/minios7/minios7_2.0/i-7188xb/lib/ 或者 CD: \Napdos\MiniOS7\MiniOS7_2.0\i-7188xb\lib

在 7188xbs.lib/7188xbl.lib 库文件中支持数百种函数如下：

功能描述	例子
COM port	InstallCOM0, InstallCOM1, InstallCOM2 …… IsCOM0, IsCOM1, IsCOM2 …… ToCOM0, ToCOM1, ToCOM2 …… ReadCom0, ReadCom1, ReadCom2 ……
EEPROM	WriteEEP, ReadEEP, EnableEEP, ProtectEEP
LED and 5-digit LED	LedOn, LedOff, Init5DigitLed, Show5DigitLedWithDot
Flash Memory	FlashReadId, FlashErase, FlashRead, FlashWrite ……
Timer and Watchdog Timer	TimerOpen, TimeClose, TimerResetVlaue, TimerReadValue, StopWatchReset, StopWatchRead, StopWatchStop
File	GetFileNo, GetFileName, GetFilePositionByNo, GetFilePositionByName
Connect to 7000 series modules	SendCmdTo7000, ReceiveResponseFrom7000
Programmable I/O	SetDio4Dir, SetDio4High, SetDio4Low, GetDio4
Others	Kbhit, Getch, Putch, LineInput, Scanf ……

注: 函数的更多详细信息，请参考附录 D:库函数列表。

3.2 编译和连接

任何应用程序开发需要 C 语言编译器支持。有效的编译器包括：

- BC++ 3.1~5.02
- TC++ 1.01
- TC 2.01
- MSC
- MSVC++ (支持 1.52 之前版本)

建议客户使用 Borland C++ 3.1 编译器，注意在编译前请按如下步骤选择项目：

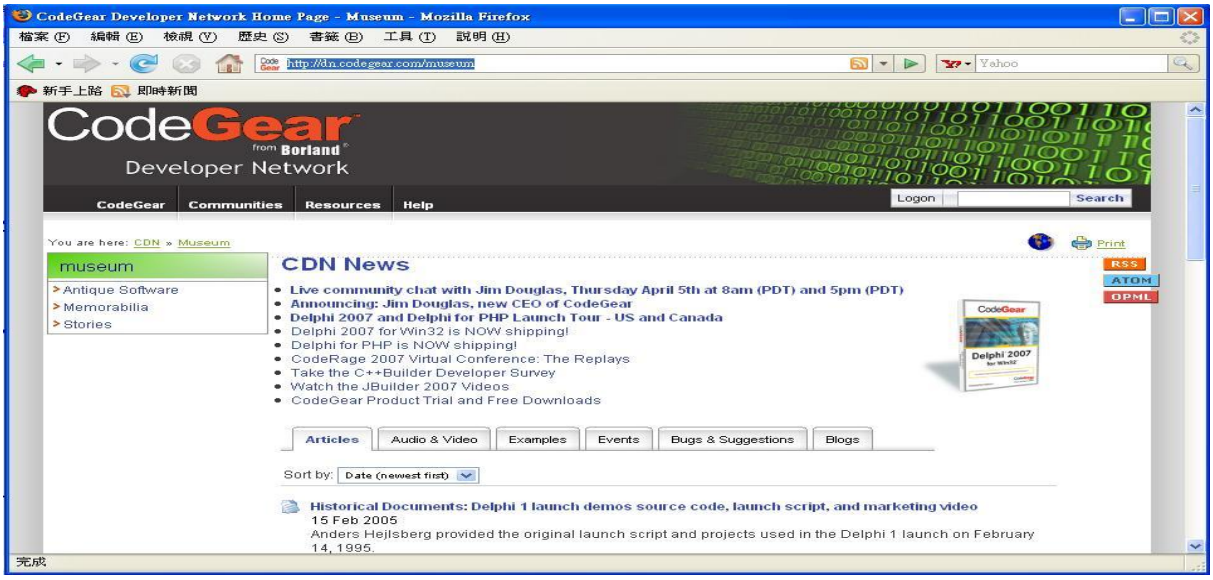
- 生成标准的 DOS 可执行程序。
- 设定 CPU 到 80188/80186
- 若有浮点运算需求，设置浮点类型为 EMULATION(注意不要设定为 8087)
- 取消调试信息功能用来帮助减少程序的大小(MiniOS7 支持该功能)

3.3 程序编写的详细步骤

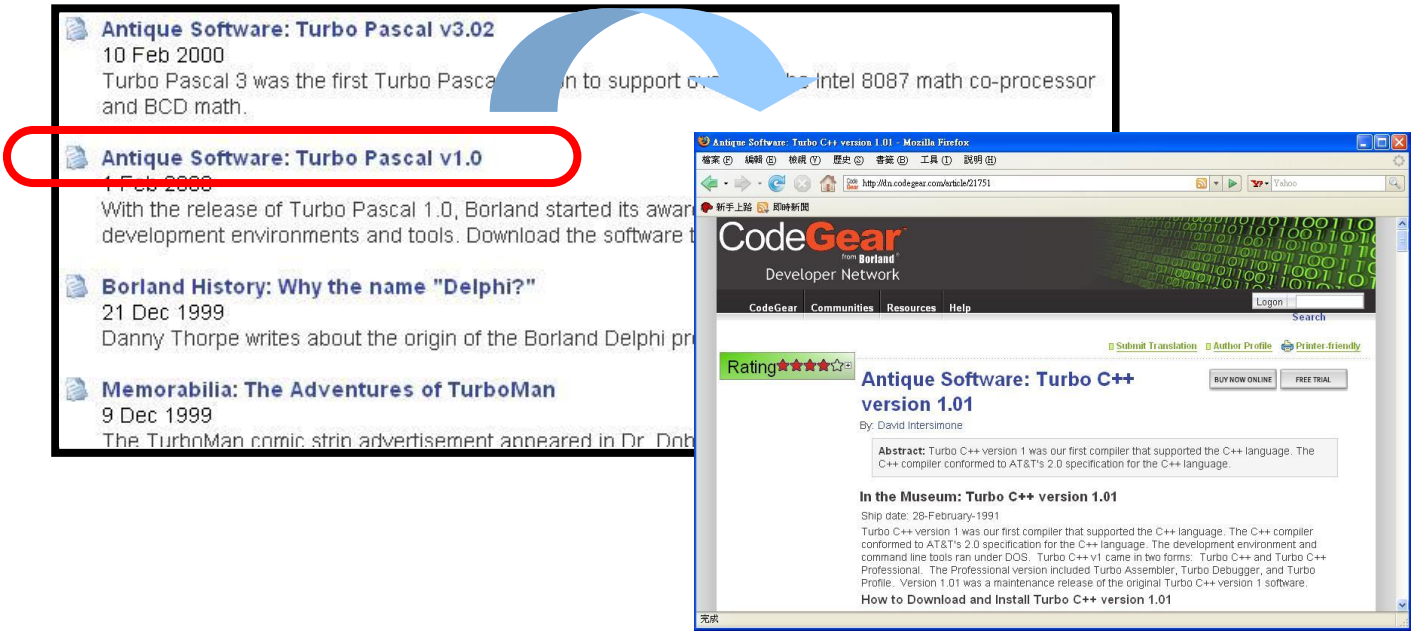
3.3.1 下载 Turbo C++ 1.01

Turbo C 2.01 和 Turbo C++ 1.01 编译器能够在 Borland 站点下载。参考如下步骤在 Windows 操作系统中安装 Turbo C++版本 1.01 编译器。

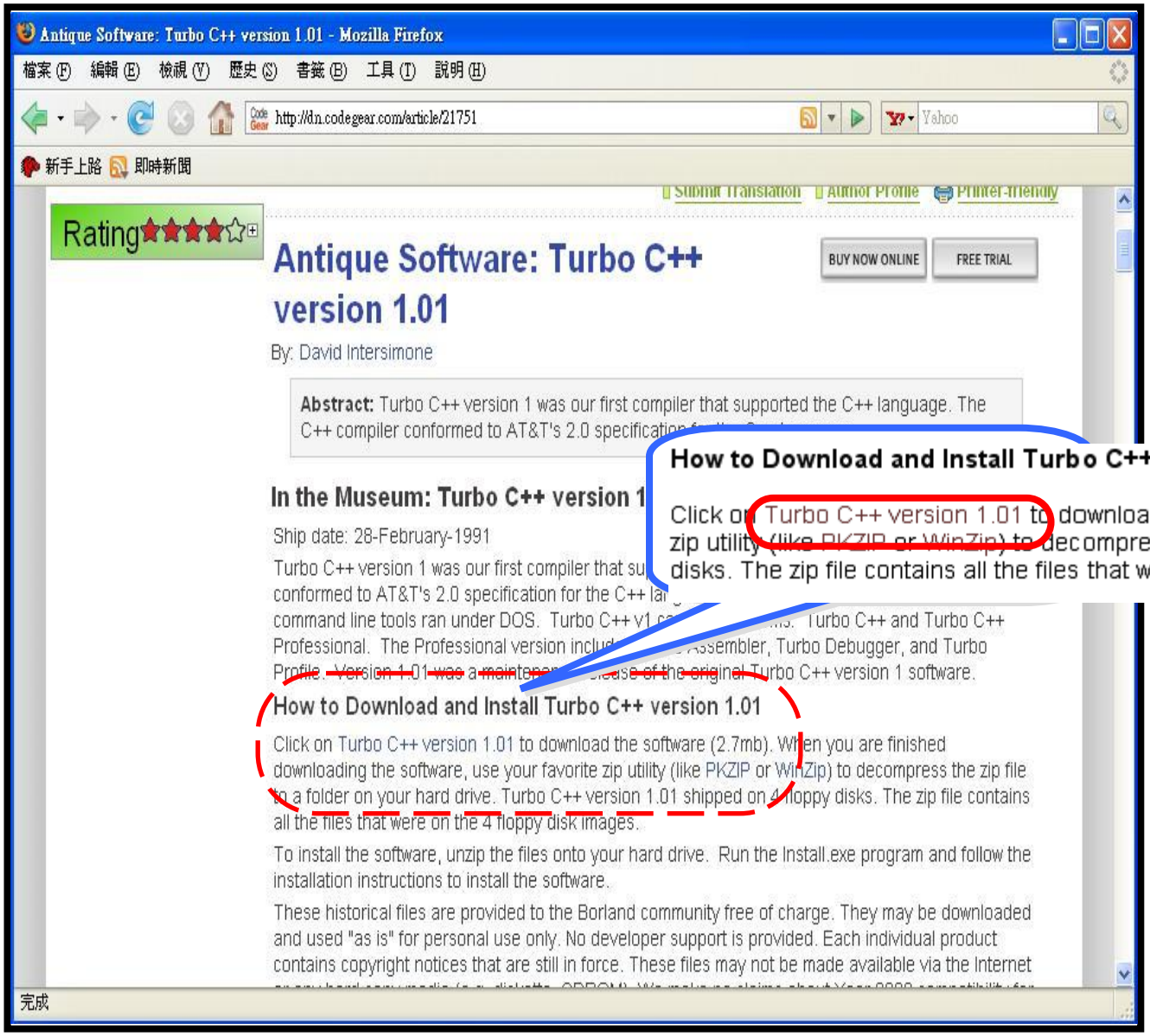
步骤 1: 定位到 Web 站点 CodeGear (http://dn.codegear.com/museum).



步骤 2: 滚动条向下滚动单击链接 **Antique Software: Turbo C++ version 1.01** 进入下载页面。



步骤 3: 单击链接 **Turbo C++ version 1.01**，如下，下载 **tcpp101.zip** 文件. 保存文件到本地硬盘。

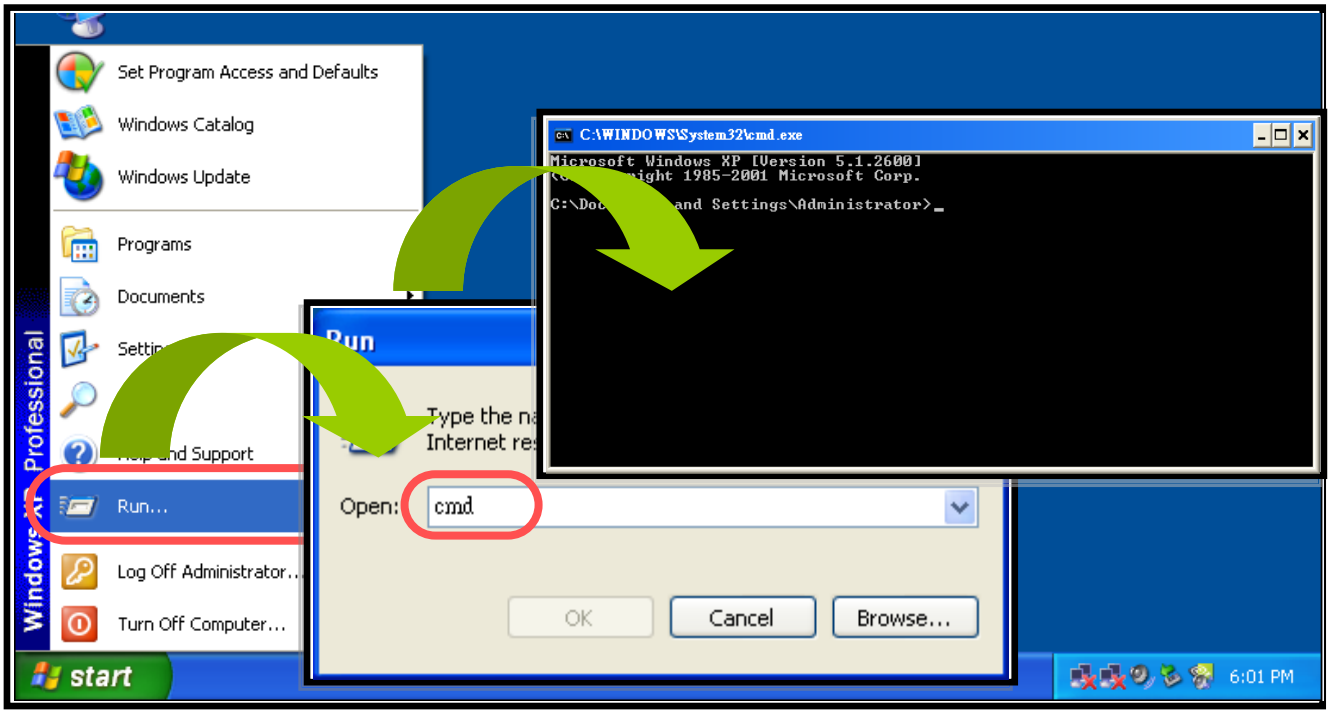


3.3.2 安装 Turbo C++ 1.01

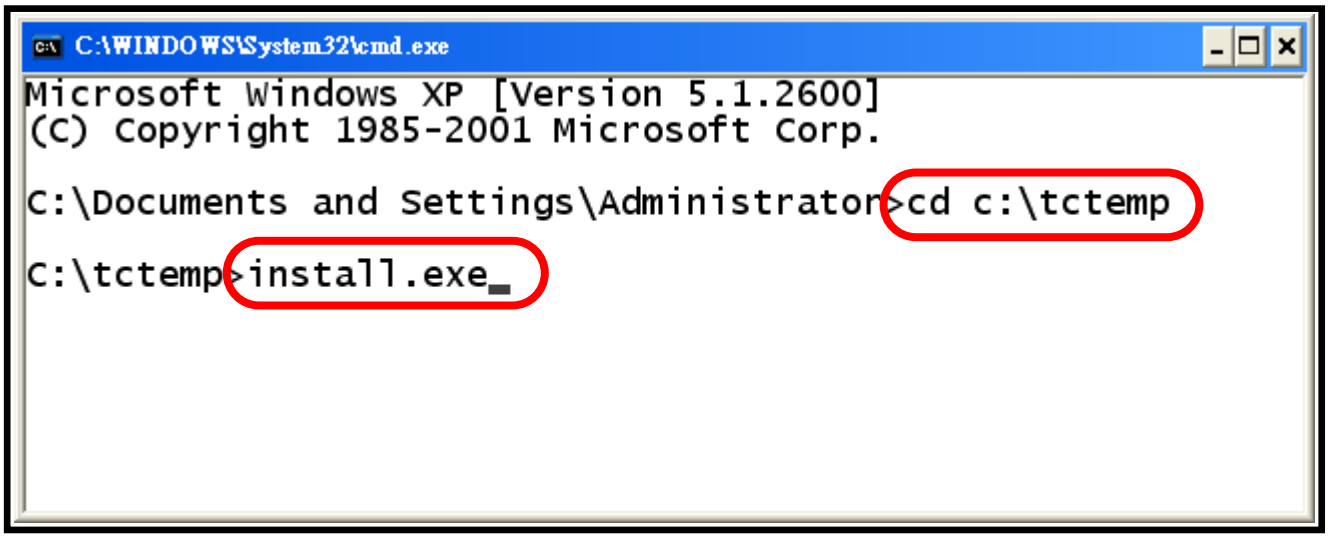
步骤 1: 找到下载的文件，双击打开自解压文件(tcpp101.zip)将其释放出来。默认解压文件路径是 C:\tctemp 目录，你也可以指定到其它位置。

步骤 2: 解压后离开自解压程序。

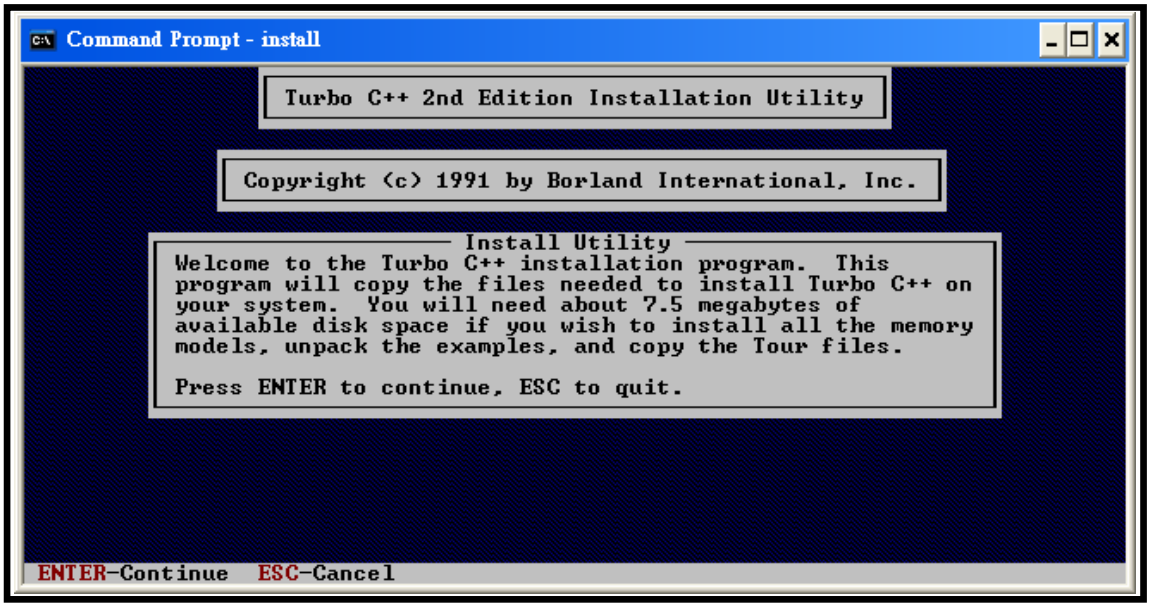
步骤 3: 打开 MS-DOS 命令提示窗口。



步骤 4: 进入目录 **c:\tctemp**,并执行 **INSTALL.EXE** 文件。



步骤 5: 依照安装向导安装软件。



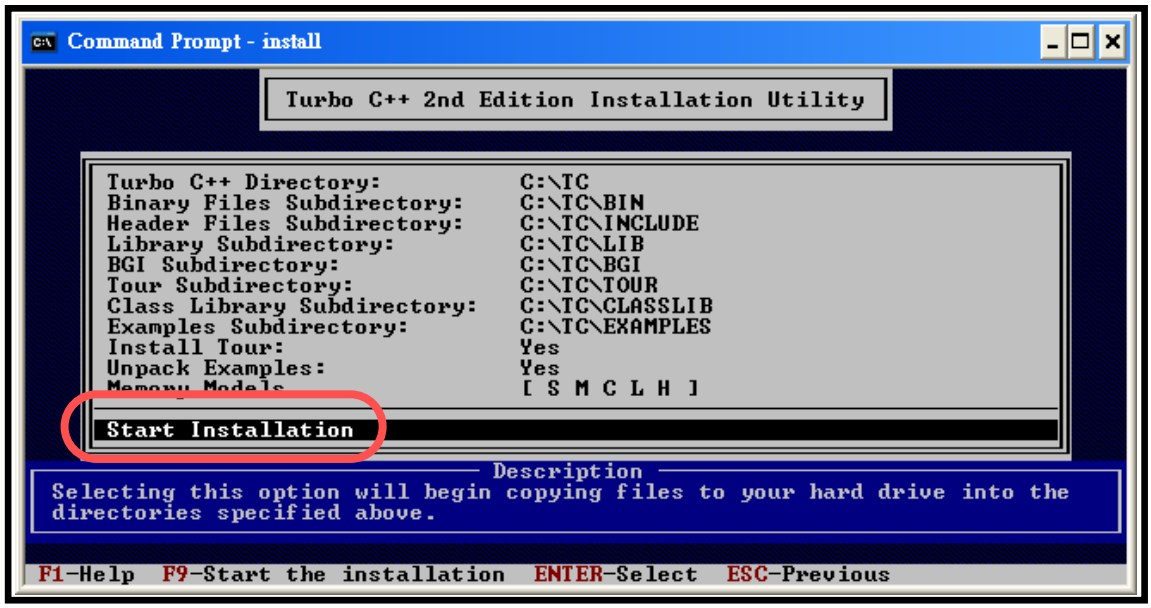
步骤 5.1: 按 **<ENTER>** 开始安装

步骤 5.2: 选择解压文件后的**驱动器**。默认为 “A”，因此输入 **“C”**,然后点击**<ENTER>**

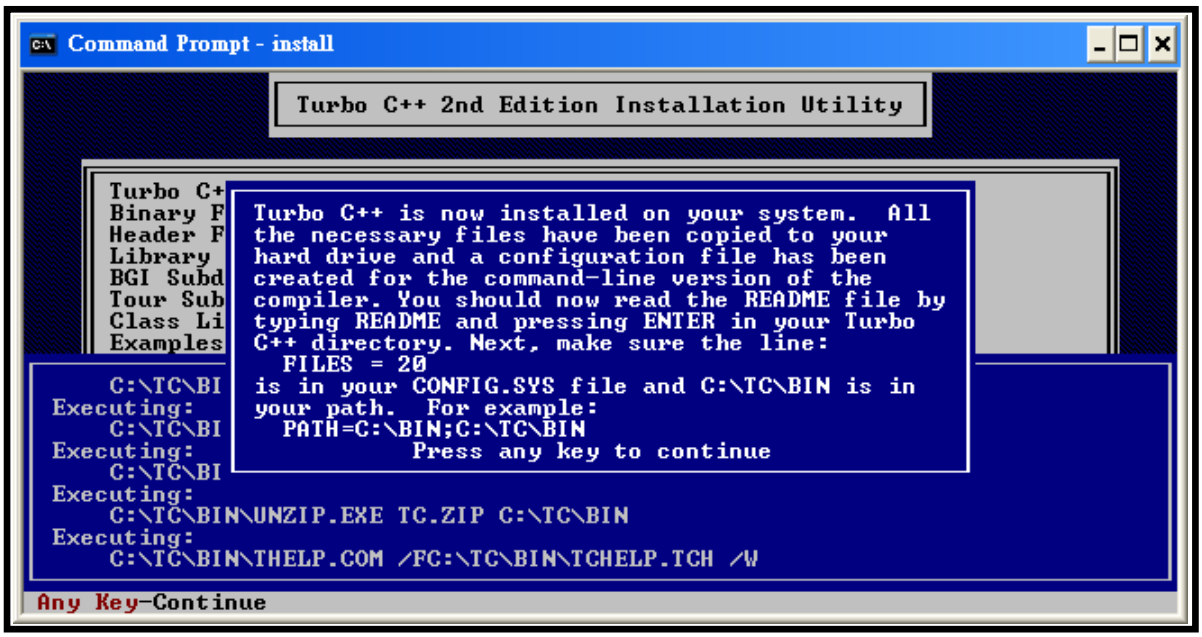
步骤 5.3: 再次点击**<ENTER>**，将从文件夹\tctemp 中安装软件

步骤 5.4: 再次点击**<ENTER>**，将 Turbo C 安装于硬盘上

步骤 5.5: 使用方向键 Up/Down 把选择 **Start Installation**，并再次点击 **<ENTER>**



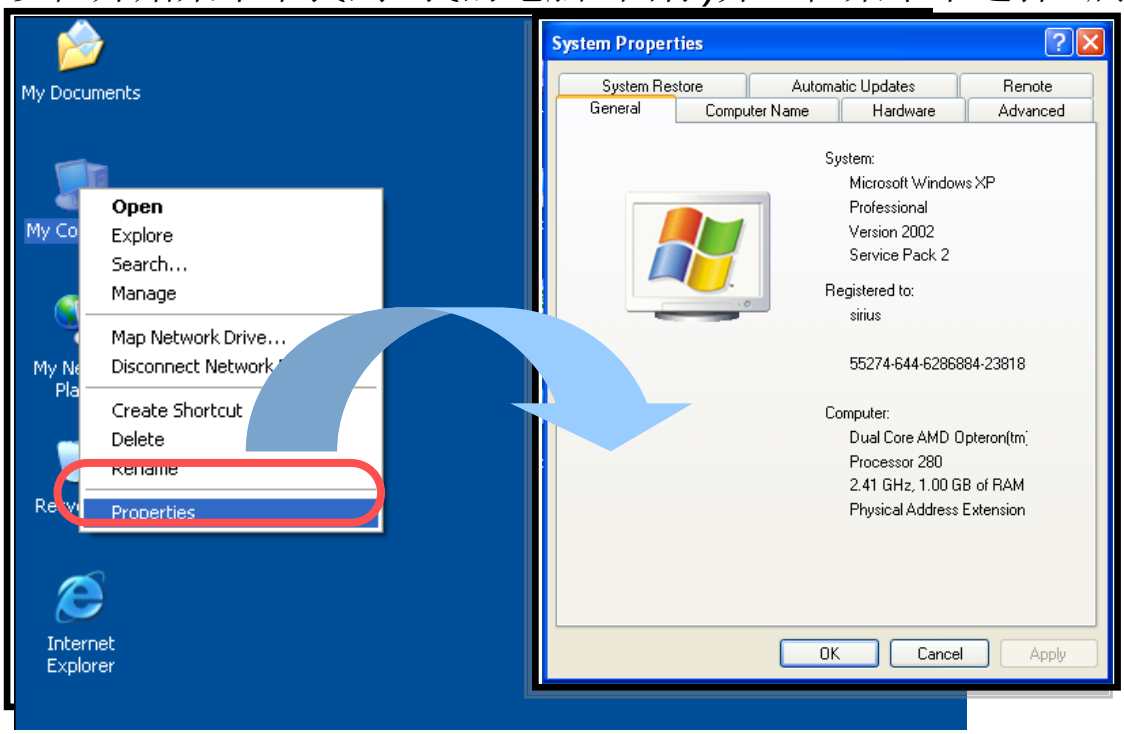
步骤 5.6: Turbo C++版本 1.01 编译器安装在 C:\TC，在这里找到 tcc.exe 执行



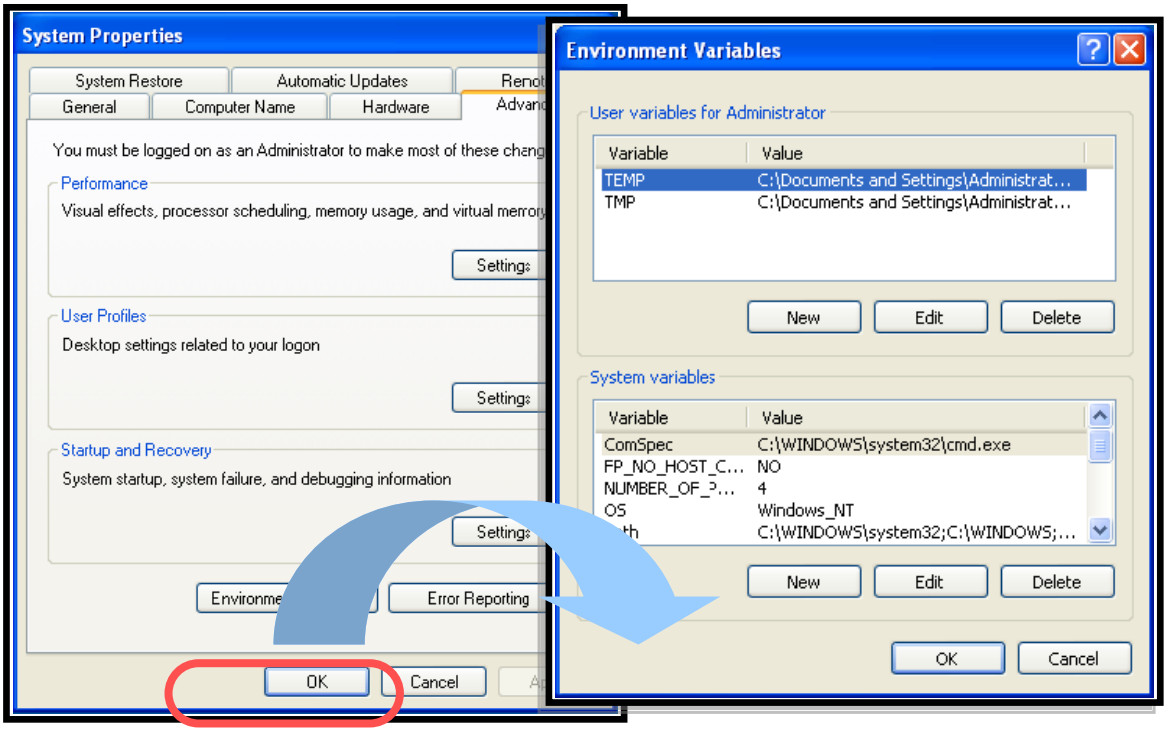
3.3.3 设定操作系统环境变量

在安装完后，你需要按照如下步骤去增加一个环境变量：

步骤 1: 在桌面鼠标右键单击“我的电脑”图标。(使用 Windows XP，可以在开始菜单中找到“我的电脑”图标)并且在菜单中选择“属性”。



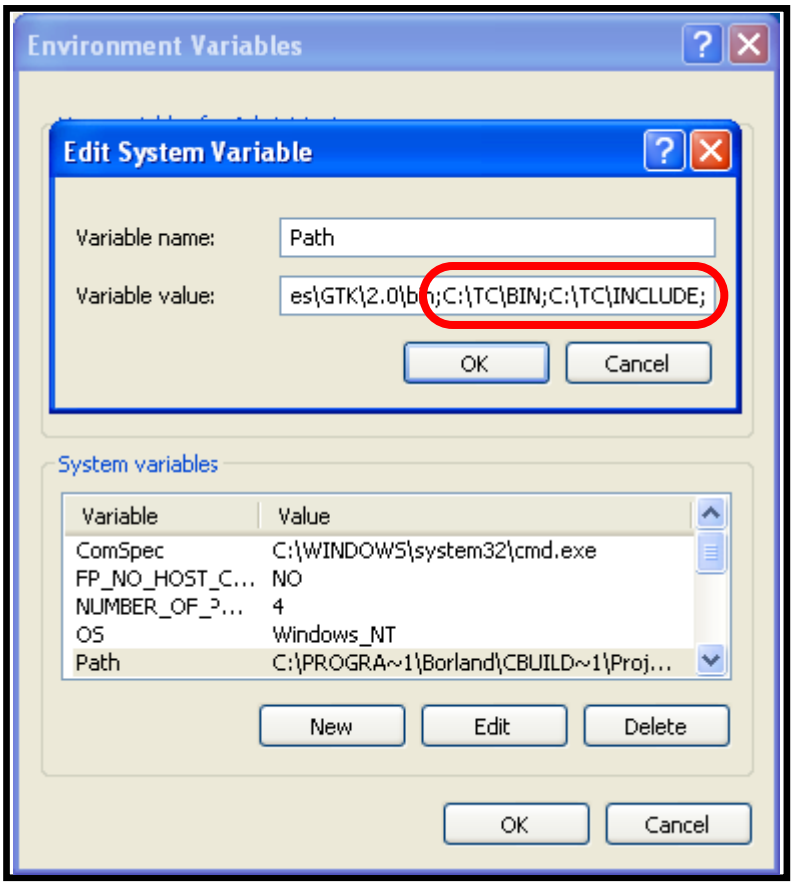
步骤 2: 单击选择标签“高级”，再单击“环境变量”按钮。



步骤 3: 在环境变量中找到“Path”单击“编辑”按钮。

步骤 4: 添加目录到最后，变量值使用分号分隔。例如：

”**C:\TC\BIN;C:\TC\INCLUDE;**”



步骤 5: 单击“确定”按钮，并重新启动你的电脑。

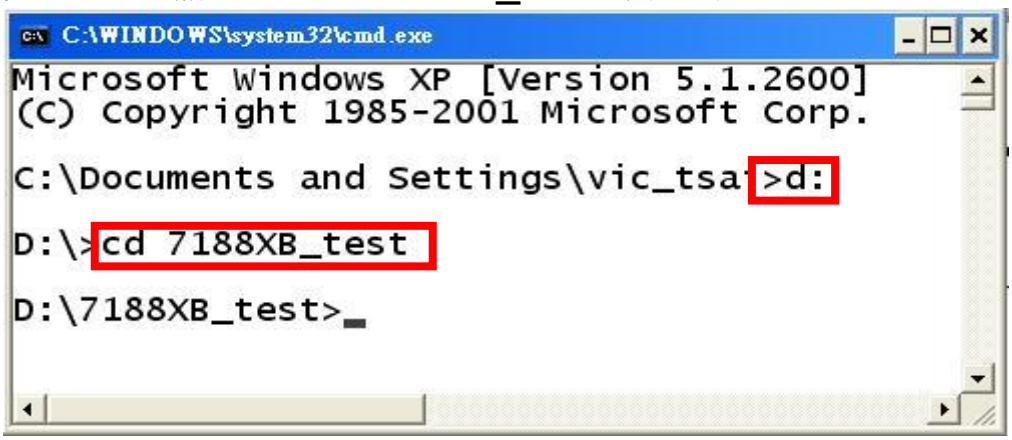
3.3.4 编译并执行程序

步骤 1: 打开 MS-DOS 命令提示窗口。

注: 在你打开 MS-DOS 命令提示窗口前，你必须关闭以前打开的 MS-DOS 命令提示窗口。

步骤 2: 输入 “d:” 并且按 **<Enter>** 进入 D 磁盘驱动器.

步骤 3: 输入 “cd 7188XB_test” 并且按 **<Enter>**.

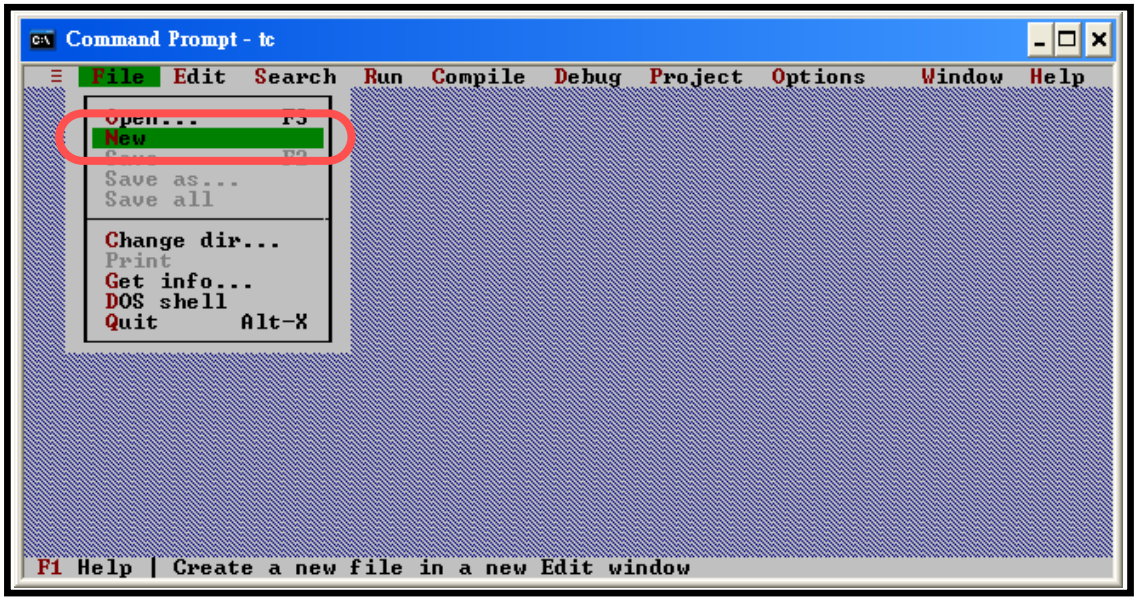


注: 假定存在文件夹 7188XB_test 建立在 d:\. 并在文件夹 7188XB_test 包含有 7188xb.h 和 7188xbl.lib。

步骤 4: 输入 **tc** 后单击 **<ENTER>**运行 TC++ 1.01 编译环境。该命令可从任务位置执行。

步驟 5: 创建一个源文件 (*.c).

步驟 5.1:选择菜单“File”-> “New”。



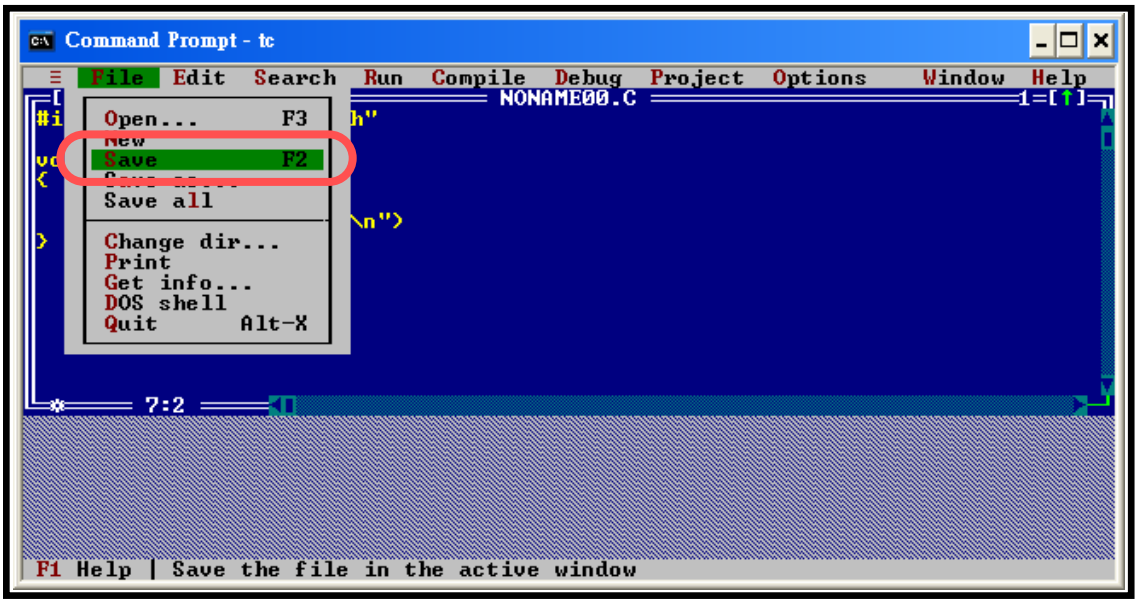
步驟 5.2: 输入如下代码，注：代码是区分大小写。

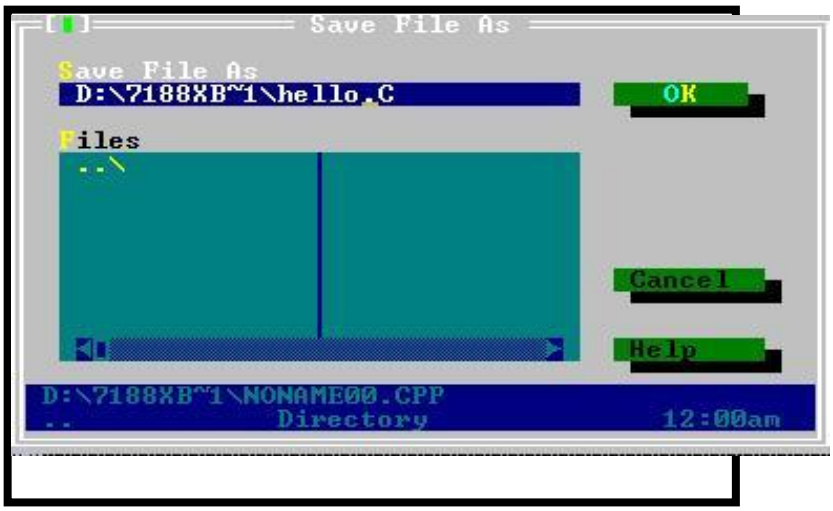
```
#include “7188xb.h”

void main(void)
{
   InitLib();

   Print("Hello world!\r\n");
}
```

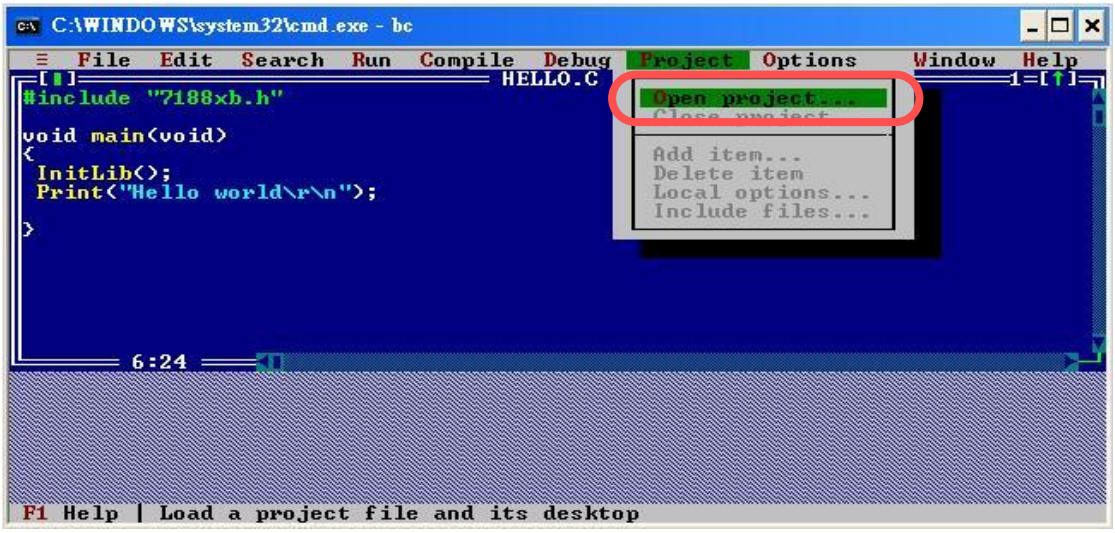
步骤 5.3: 点击菜单 ”File”->“Save”，输入文件名 Hello.C



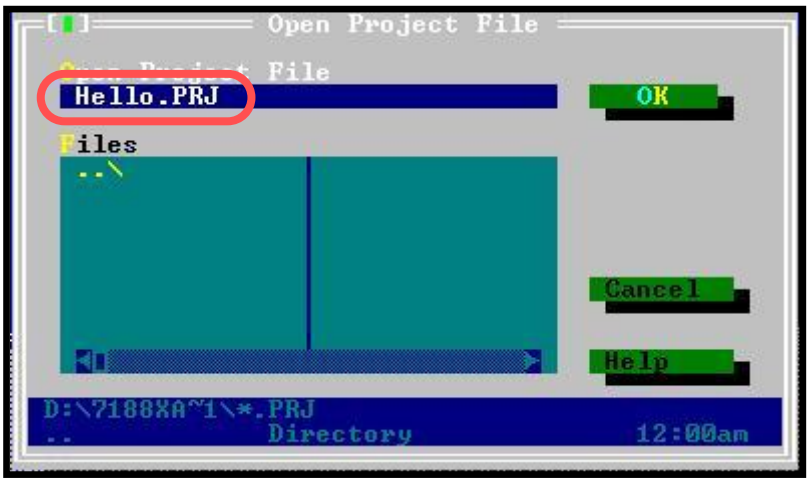


注: 如果你有熟悉的文本编辑器，你也许使用它来编辑上面的代码。但要注意代码不可用诸如 WORD 这样的文字处理软件写，使用文字编辑器必须是保存为纯文本类型，如 Windows 自带的记事本等。C 语言程序文件后缀必须为 “.C”。

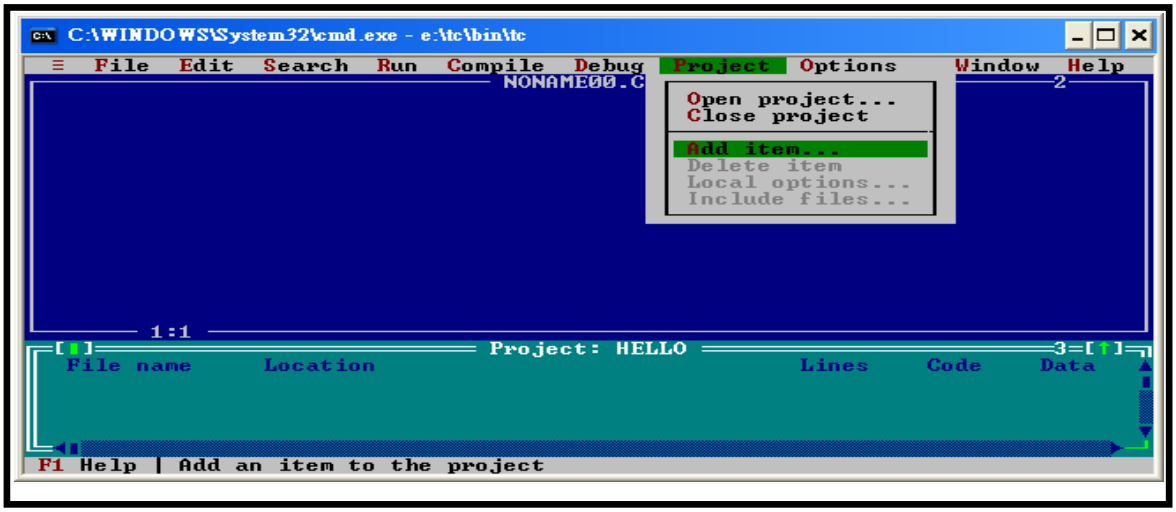
步骤 6: 创建一个新工程文件(*.prj)



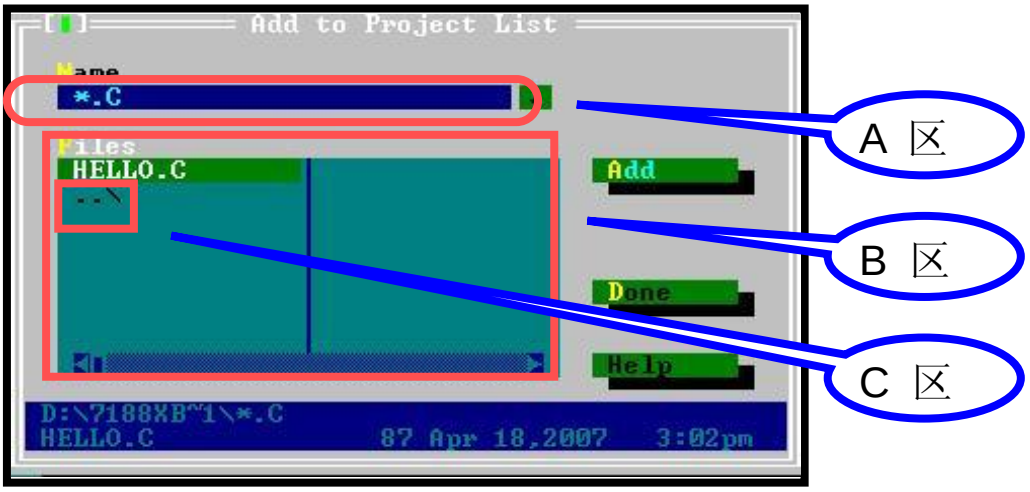
步骤 6.1: 输入工程文件名并单击 “OK”按钮。



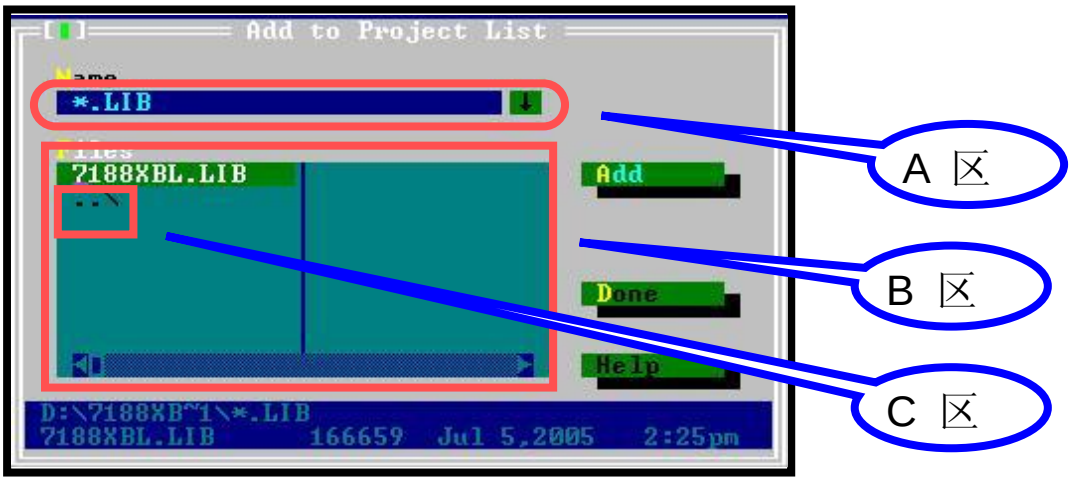
步骤 7:添加所有必须的文件到工程。



步骤 7.1: 选择程序源文件，输入 “*.c” 在 A 区点击 **Enter**。若所需文件在 B 区中，移动绿色滑条选定文件，单击按钮 **Add**；若没有,则移动 C 区中绿色滑条，重新选择文件。

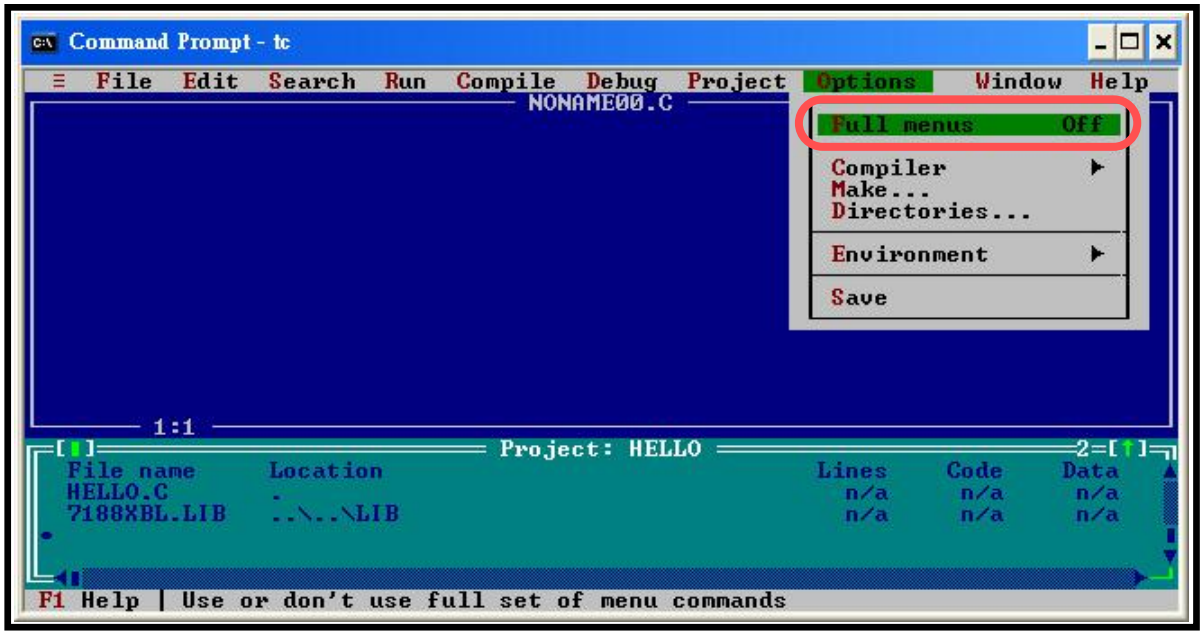


步骤 7.2: 选择函数库。输入“*.lib” 在 A 区点击 **Enter**。若所需文件在 B 区中，移动绿色滑条选定文件，单击按钮 **Add**；若没有,则移动 C 区中绿色滑条，重新选择文件。

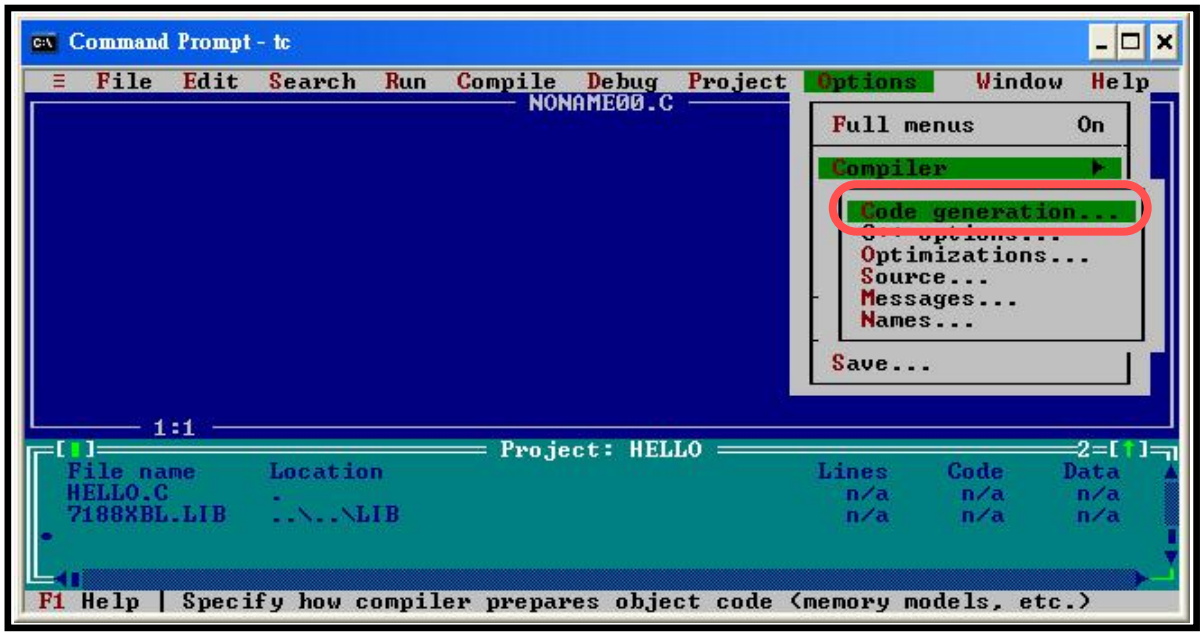


步骤 8: 单击 “**Done**” 离开。

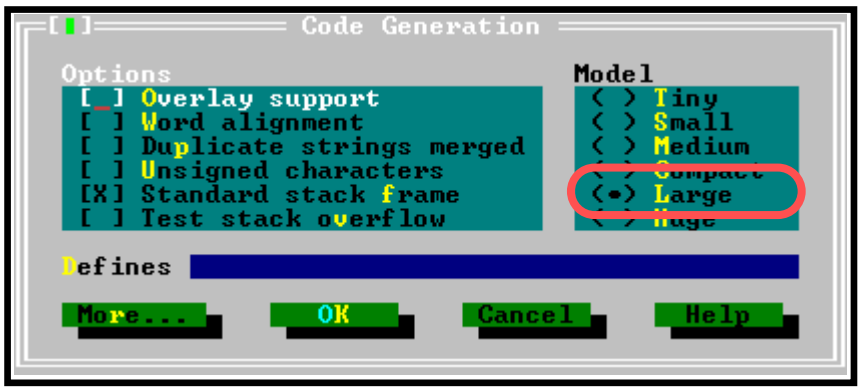
步骤 9: 单击菜单 “**Options**” 选择 “Full menus”。



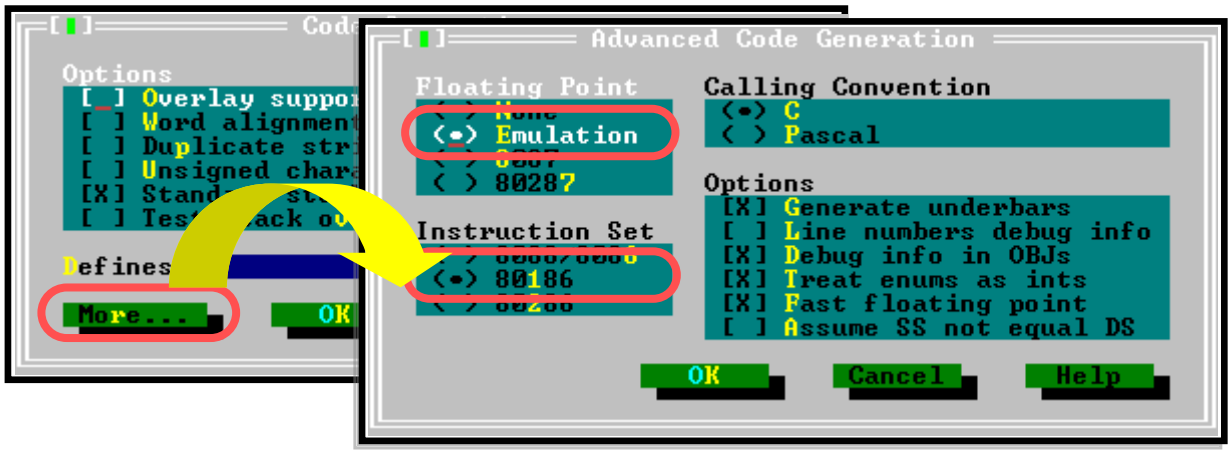
步骤 10: 单击菜单 “Options”选择 “Compiler”，进入 “Code generation options”设定代码产生选项。



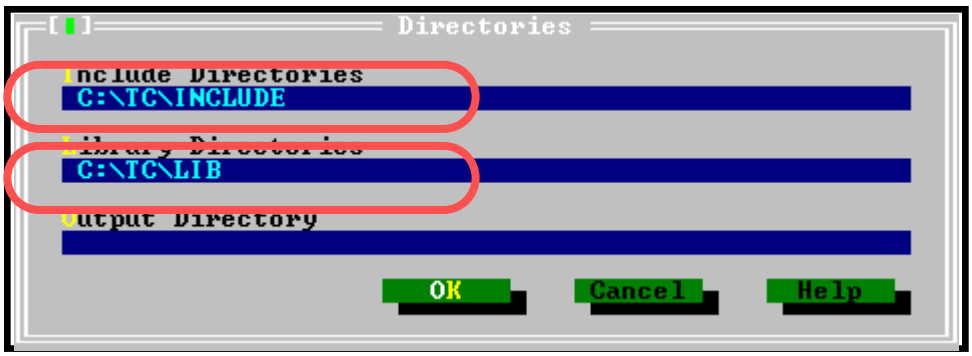
步骤 10.1: 改变内存模式 (Small 适用于 7188xbs.lib, large 适用于 7188xbl.lib)



步骤 10.2: 单击 “**More…**”,设置 Floating Point 为 **Emulation**，**Instruction Set** 为 **80186**.

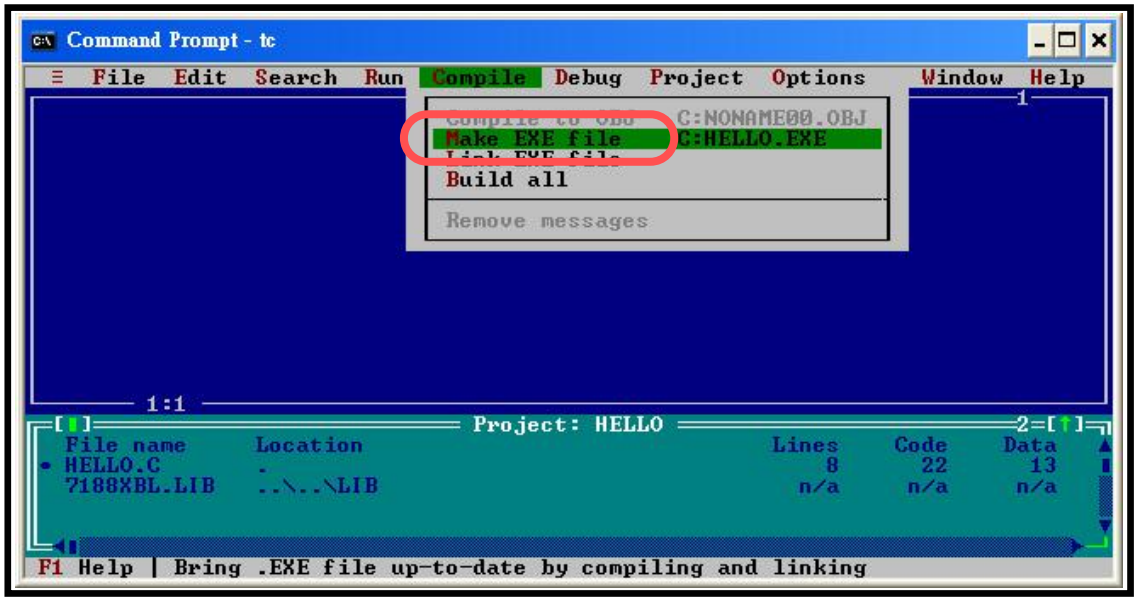


步骤 11: 单击“**Options**” 选择“**Directories…**” 进入 TC++ 1.01 include 及 library 目录路径。默认设置为 TC++ 1.01 安装文件夹。

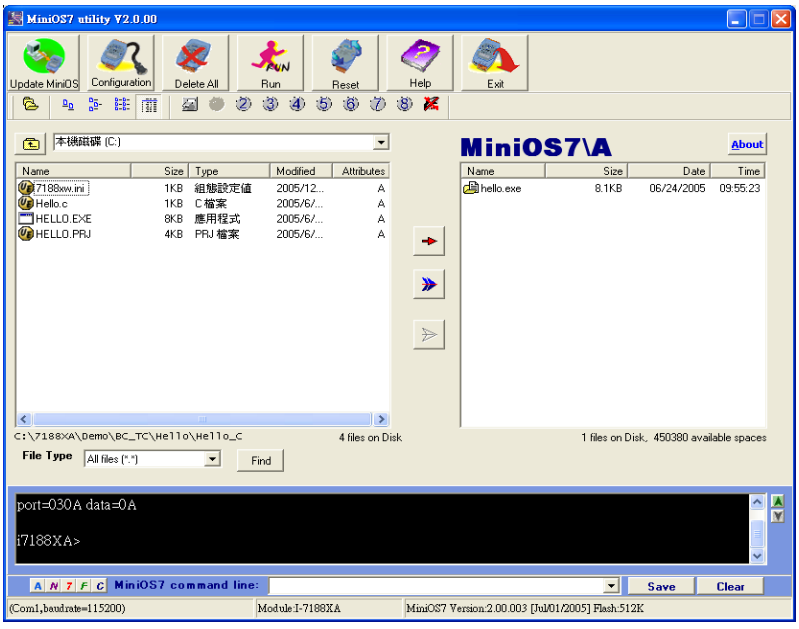


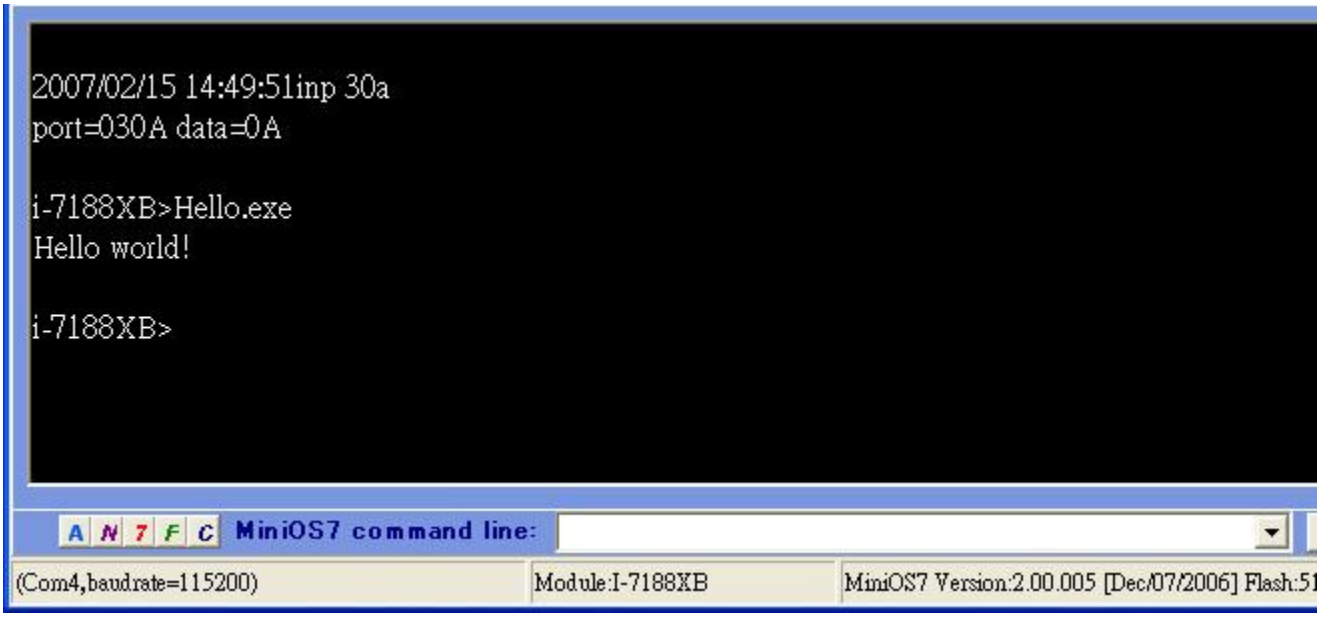
注： Include Directories 特指包含有标准 include 文件目录；Library Directories 特指包含 TC++ 1.01 启动目标文件及 run-time 库文件。

步骤 12: 单击 “**Compile**” 并选择 “**Make EXE file**” 生成 EXE 文件。



有关下载与执行应用程序，详情请参考章节.2.3.





有关各种不同 C 语言编译器(TC/BC/MSC/MSVC)详情编译和链接说明请参考**附录 E: 编译和链接**。

3.4 在 64 位平台建立工程(Project)

假如要在 Windows 7 或 Windows 8 这种有支持 64 位的 OS 平台上使用 16 位的编译器(像是 BC3.1 或是 TC++3.0)来创建 MiniOS7 项目时，将会告知有 64 位平台兼容性问题信息出现。

请参阅**附录 E: 编译和链接** 的子章节“在 64 位平台编译”.

4. 操作原理

4.1 系统映射

装置	地址映射
Flash ROM	512K: 从 8000:0000 到 F000:FFFF
SRAM	256K: 从 0000:0000 到 3000:FFFF 512K: 从 0000:0000 到 7000:FFFF
COM1 BASE	0XFF80 到 0XFF88
COM2 BASE	0XFF10 到 0XFF18

中断号	中断映射
0	Divided by zero
1	Trace
2	NMI
3	Break point
4	Detected overflow exception
5	Array bounds exception
6	Unused opcode exception
7	ESC opcode exception
8	Timer 0
9	Reserved
0A	DMA-0
0B	DMA-1
0C	\INT0 of the I/O expansion bus
0D	\INT1 of the I/O expansion bus
10	\INT4 of the I/O expansion bus
11	COM2
12	Timer 1
13	Timer 2
14	COM1

4.2 使用 COM1 调试用户程序

I-7188XB(D) COM1 口 (下载口) 有三个主要功能。

- 从 PC 端下载程序。
- 连接到 PC 用来调试程序。
- 作为通用 COM 口。

当 I-7188XB(D)电源开启，它将初始化 COM1 口在控制台模式下配置信息如下：

- **Start Bit=1, Data Bit=8, Stop Bit=1, no parity**
- **Baud Rate=115200 bps**

I-7188XB(D) 将自行检测引脚 INIT*状态。若**引脚 INIT*与 GND 短接**, 则 I-7188XB(D) 将发送启动信息，使 COM1 进入控制台模式来允许用户下载/调试程序, 通过根据以下步骤可显示启动信息：

- 关闭 PC 和 I-7188XB(D)
- 连接下载线到 I-7188XB(D)和 PC 的 COM 口(更多详细资料参考章节 2.2)
- 打开 PC 执行 7188xw.exe
- I-7188XB(D)电源开启
- 所有初始化信息将被显示在 PC 的显示器上。

如果 INIT*处于打开状态, I-7188XB(D) 将搜索 **autoexec.bat** 文件。 如果 autoexec.bat 文件存在,I-7188XB(D) 将执行这个文件。如果 autoexec.bat 文件不存在， I-7188XB(D) 将返回到控制台模式以便用户下载/调试程序。

完成初始化设置阶段后, I-7188XB(D) 可将 COM1 作为标准的输入/输出端口使用。I-7188XB(D)的标准输出可显示在主机 PC 的显示器上。当点击主机 PC 键盘时，这个字符可作为标准输入回传到 I-7188XB(D)。按如下步骤， 主机 PC 的键盘和显示器可作为 I-7188XB(D)的标准输入和输出设备：

- 使用 7188xw.exe 或 MiniOS7 Utility 作为 I-7188XB(D)和主机 PC 之间的桥梁。

- 在主机 PC 上执行 7188xw.exe 或 MiniOS7 Utility 进行该沟通方式的设置。
- 主机 PC 的键盘 → I-7188XB(D)的标准输入装置。
- 主机 PC 的显示器→ I-7188XB(D)的标准输出装置。

采用这种方式，I-7188XB(D)可读取通过键盘读取数据并显示在显示器上。因此，调试程序将变得简单和高效。

注意：7188xw.exe 和 MiniOS7 Utility 在随机光盘中提供。请参考章节 2.2 节 了解详细的配线信息和章节 2.3 节了解下载程序。

4.3 下载端口作为通用串口

I-7188XB(D)的下载端口 (COM1)通过如下方式可被用做普通的 RS-232 端口：

步骤 1: 首先下载用户程序和 autoexec.bat 到 I-7188XB(D)。

步骤 2: 关闭 I-7188XB(D)电源并从主机 PC 移除下载电缆。

步骤 3: 断开 INT*引脚和 GND 引脚的连接。

步骤 4: 接通 I-7188XB(D)的电源 (无标准输入、无标准输出、无调试信息)。

步骤 5: 使用下载电缆连接新的 RS-232 设备到 I-7188XB(D) 的 COM1 端口。

步骤 6: 重新初始化设置该 COM1 口 。

步骤 7: 该 COM1 口可变成普通的 RS-232 端口。

4.4 功能和 Demo 程序列表

光盘中提供一些为 I-7188XB(D)设计的 demo 程序。关于这些程序的详细信息请参考后面章节中的内容。Demo 程序的功能请见下表：

文件夹	Demo 程序	解释	章节
Hello	Hello_C	检测操作系统是否为 MiniOS7。 **Note:** MSC 不支持 C++ 语言。Hello_C++文件仅支持 BC。	3.3.4
	Hello_C++		
COM_Port	C_Style_IO	1. 示范如何写程序进行数据输入。 2. 示范如何得到一个字符串。 3. 示范如何使用 C 函数： sscanf 或 Scanf()。	4.6
	Receive	从 COM 端口接收数据。	
	Slv_COM	PC 发送命令到 I-7188XB(D)，并接收从 I-7188XB(D)返回的数据。也可示范如何使用另外的 COM 端口 或 LED 显示相关信息来协助调试程序。	
	ToCom_In_Out	通过 COM 端口读/写数据。	
DateTime		从 RTC 读取数据和时间标签并显示在显示器上（用户也可设置数据和时间）。	
IO_PIN		读/写 I-7188XB(D)的 DI 和 DO 值。	4.11
LED	Led	示范如何使用 DelayMs 函数来控制 LED 开或关。	4.7
	Seg7led	控制红色 LED 和 5 位 7 段数码管。	
File	Config_1_Basic	在一些应用中，程序需要从 text 文件中读取文件记录的特别信息。Fseek 可用来从文件中读取这些特别的信息。	
	Config_2_Advanced	拓展 config_1_Basic 以及增加 GetProFileInt 、 GetProFileFloat 和 GetProFileStr 这三个函数可用来定义文件的类型。	
Memory	Demo5	示范如何访问 NVRAM。	4.8
	EEPROM	在 EEPROM 中写入一个值，并将该值显示在显示器上。	
	EEPROM-r	读取 EEPROM 上已存在的数据。	
	EEPROM-w	在 EEPROM 的区块 1 中输入并存储一个值 (值将自动以 1 递增)。	
	Flash	对 Flash 进行读、写和擦操作。	
	Flash-r	读取已经写入在 Flash 中的值。	

	Flash-w	在 Flash memory 写入一个值 (值将自动以 1 递增).	
	NVRAM-r	读取已经写入在 NVRAM 中的值。	
	NVRAM-w	在 NVRAM 中写入一个值 (值将自动以 1 递增)。	
	Top-Mem	示范分配 TopMemory 功能。	
Misc	Reset	复位	
	Runprog	使用 Ungetch 函数来运行另外的程序。	
	SerialNumber	取回 I-7188XB(D)的序列号。	
	Watchdog	使用 Watchdog 或者跳过使用 Watchdog。	4.9
7K87K_Module	7K87K_demo_for_com	示范如何通过 COM2 端口连接并控制 7K 或 8K 串行模块。.	4.6.3
	7K87K_AI_for_Com		
	7K87K_DI_for_Com		
	7K87K_DIO_for_Com		
	7k87K_DO_for_Com		
	AO_024_for_Com		
	AO_22_26_for_Com		
Timer	Demo90	用来示范如何使用 Timer 功能的示范程序。	4.10
	Demo91	示范如何使用 CountDownTimer 函数在 channel 0 来控制 LED 开或关。	
	Demo92	示范如何使用 StopWatch 函数在 channel 0 控制 LED 开或关。	
	Demo96	示范使用 InstallUserTimer 函数控制 5 位 7 段数码管。	
	Demo97	示范使用 DelayMs 函数控制 LED 开或关 。	
	Demo98	示范使用 I-7188XB(D)的 timer 功能发送/接收数据自 7000 系列串行模块。	
XBoard		应用在 I-7188XB(D)上的 I/O 扩展板的 DEMO 程序。	4.12

4.5 COM 端口对照

I-7188XB(D) 的 COM 端口如下表:

COM Port	Hardware
COM1	80188 CUP 的内部 UART-0, 5 线 RS-232 (默认)或者 2 线 RS-485
COM2	80188 CUP 的内部 UART-1, 2 线 RS-485

I-7188XB(D)的 COM1 端口可设置成 RS-232 或 RS-485 类型如下:

COM 端口类型	引脚名称
2 线 RS-485	Data+, Data-
3 线 RS-232	TXD, RXD, GND
5 线 RS-232	TXD, RXD, GND, RTS, CTS

16C550 需要的程序与 80188 UART 所需要的有很大不同。 80188 上的中断处理与 PC 上的 8259 也有很大不同。因此，PC 环境下的 RS-232 应用程序无法在 I-7188XB(D)上执行。

I-7188XB(D)上的 COM 软件驱动是一个中断驱动库文件，并且为每个 COM 端口提供 1K 的队列缓存。因此，COM 端口的应用软件非常容易设计和使用。

软件驱动为所有两个 COM 端口提供相同的接口，因此每个 COM 端口可用同样方式进行操作，并不会因串口不同而增加开发难度。

4.6 使用 COM 端口

I-7188XB(D) 具有两个通信端口。

- COM1 可作为 RS-232 (默认) 或 RS-485 端口：
  RS-232: TXD, RXD, RTS, CTS 和 GND
  RS-485: D1+, D1- (内置 Self-tuner 芯片)
- COM2 is an RS-485 port (D2+, D2-, 内置 Self-tuner ASIC)

在使用 COM 端口之前，必须使用 ***InstallCom()*** (或 ***InstallCom1/2***) 函数装入 COM 端口的驱动。
在退出程序前，需使用 ***RestoreCom()*** (或 ***RestoreCom1/2***)函数释放端口。

在使用 ***InstallCom()***函数后即可从 COM 端口进行数据的读取、写入、打印或其它的操作。

在从 COM 端口读取数据前，需使用 I***sCom()***函数检查是否收到数据。若为真，则使用 ***ReadCom()***函数从 COM 端口的缓存中读取数据。
在向 COM 端口发送数据前，需使用 **ClearCom()函数确认 COM 端口的输出缓存是否清空，然后使用** ***ToCom()*** 函数向 COM 端口发送数据。

例如，从 COM1（RS-232）端口读取/发送数据的代码如下：

```
int port=1;   /*to use COM1*/
int quit=0, data;

InitLib();    /* Initiate the 7188xb library */
InstallCom(port, 115200L, 8, 0, 1);   /*install the COM driver*/
while(!quit){
   if(IsCom(port)){ /*check if any data is in the COM Port input
                      buffer*/
      data=ReadCom(port);   /*read data from the COM Port*/
      ToCom(port, data);   /*send data via the COM Port*/
      if(data=='q') quit=1;   /*if 'q' is received, exit the program*/
   }
}
RestoreCom(port);   /*uninstall the COM driver*/
```

使用“port” 变量进行 COM1 和 COM2 的切换，简单从 **port=1** 切换到 **port=2**.

如果程序设置成使用 COM1，则代码修改后如下:

```
int quit=0, data;

InitLib();    /* Initiate the 7188xb library */
InstallCom1(115200L, 8, 0, 1);    /*install the COM1 driver*/
while(!quit){
   if(IsCom1()){    /*check if any data is in the COM1 input buffer*/
      data=ReadCom1();    /*read data from COM1*/
      ToCom1(data);    /*send data via COM1*/
      if(data=='q') quit=1;    /*if 'q' is received, exit the program*/
   }
}
RestoreCom1();    /*uninstall the COM driver*/
```

4.6.1 从 COM 端口打印

I-7188XB(D) 库文件支持例如 ***printf()*** 等标准的 C 语言库文件进行文件的输出。

printCom() 函数可用于所有的 COM 端口,并且 printCom1/2 函数可向每个单独的 COM 端行请求，但是首先必须使用 ***InstallCom()***函数。代码如下:

```
int port=2;    /*to use COM2*/
int i;

InitLib();    /* Initiate the 7188xb library */
InstallCom(port, 115200L, 8, 0, 1);    /*install the COM2 driver*/
for(i=0; i<10; i++){
   printCom(port, "Test %d\r\n", i);    /*print data from COM2*/
}
RestoreCom(port);    /*uninstall the COM driver*/
```

4.6.2 使用 COM1/COM2 进行 RS-485 应用

COM1/COM2 是 2 线的 RS-485 端口, 包含如下 2 个引脚：

- D+：连接 RS-485 网络的 Data+
- D-：连接 RS-485 网络的 Data-

COM1/COM2 是一个半双工的 2 线 RS-485 网络，无法用于全双工的 4 线应用。它用于直接驱动 I-7000 系列模块。

发送/接收的方向控制在 RS-485 网络中非常重要，因此 I-7188XB(D)为每个 RS-485 端口配置了一片 Self-Tuner ASIC 控制芯片，可自动侦测和控制 RS-485 网络的发送/接收方向。通过这种方式，应用程序开发人员无需担心 RS-485 网络发送/接收的方向控制。

4.6.3 向 I-7000 模块发送命令

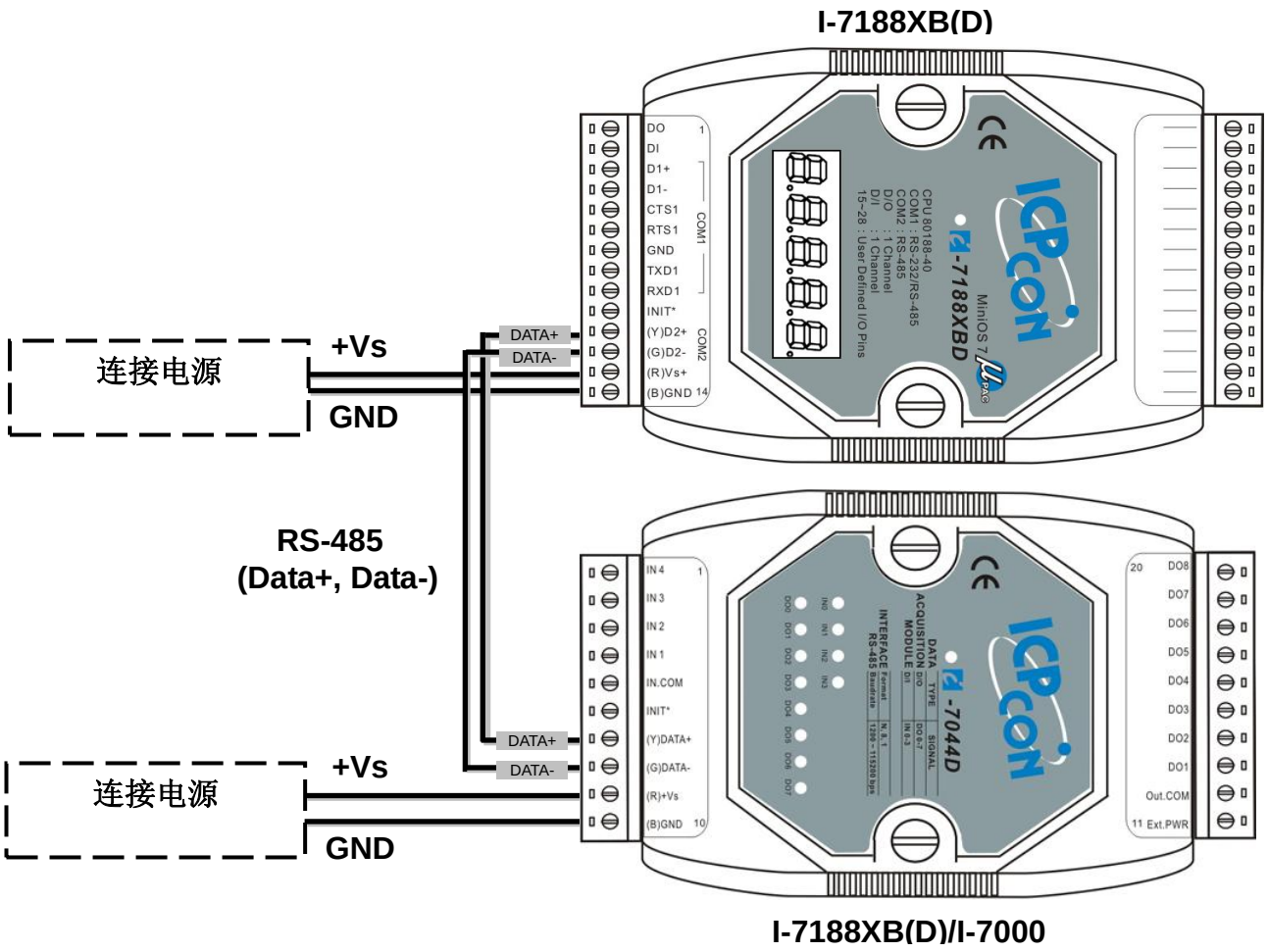
I-7000 系列模块使用的命令与 I-7188XB(D)使用的是不同的, 但是可以使用 ***ToCom()*** 函数通过 I-7188XB(D) 的 COM 端口向 I-7000 模块发送命令。

使用 COM1/COM2 连接和控制 I-7000 模块

I-7000 相关的应用步骤如下：

步骤 1: I-7188XB(D) 向 I-7000 系列模块发送字符串命令。
步骤 2: I-7000 模块执行相关命令。
步骤 3: I-7000 模块延迟 1 个字节后回应。
步骤 4: I-7000 模块返回数据给 I-7188XB(D)。

注意: 步骤 3 中的延迟时间仅为 1 个字节。



向 COM2 (RS-485)发送命令的例程代码如下：

```
int port=2;    /*to use COM2*/
int i;
char data[ ]="$01M\r";    /*command to read a module's name*/

InitLib();    /* Initiate the 7188xb library */
InstallCom(port);    /*install the COM2 driver*/
for(i=0; i<5; i++)
   ToCom(port, data[i]);    /*send a command to the I-7000 module*/
……    /*program code*/
RestoreCom(port);    /*uninstall the COM driver*/
```

除使用 ***ToCom()*** 函数之外, ***SendCmdTo7000()*** 也可以用于向 I-7000 模块发送命令。***ReceiveResponseFrom7000()***函数可用于接收 I-7000 模块的返回数据。

用于连接 I-7000 模块的函数:

- ***SendCmdTo7000(int iPort, unsigned char *cCmd, int iChksum);***
  该函数可以向 I-7000 发送命令。如果进行数据校验，该函数将在命令的最后加 2 个字节的校验码。
- ***ReceiveResponseFrom7000_ms(int iPort, unsigned char *cCmd, long lTimeout, int iChksum);***
  在使用 ***SendCmdTo7000()*** 函数后 ***ReceiveResponseFrom7000_ms()*** 可用来接收来自 I-7000 模块的返回数据。

在如下地址参考详细的 DEMO 程序：
CD:\Napdos\7188XABC\7188XB\Demo\BC_TC\7K87K_Module

注意: 更详细的 I-7000 模块命令请参考 **7000 DIO 用户手册**。

4.7 使用红色 LED 和 7 段数码管显示

I-7188XBD 带有 5 位 7-段 LED，并且每个位数的右下角皆有一个小数点，其可利用软件设定是否显示。为使各 LED 容易区分，每位 LED 从左到右使用数字 1 到 5 加以编号，且每个 LED 可独立编程。这样将在实际应用中非常实用，一定程度上可取代显示屏或触摸屏。

在使用 LED 之前，需先使用 ***Init5DigitLed()*** 函数，然后使用 ***Show5DigitLed()*** 函数来显示数据。用于在 5 位 7 段 LED 显示 “7188d” 的代码如下：

```
InitLib();    /* Initiate the 7188xb library */
Init5DigitLed();
Show5DigitLed(1, 7);
Show5DigitLed(2, 1);
Show5DigitLed(3, 8);
Show5DigitLed(4, 8);
Show5DigitLed(5, 13);    /* The ASCII code for ‘d’ is 13 */
```

DEMO 程序请参考
CD:\Napdos\7188XABC\7188XB\Demo\BC_TC\LED

4.8 访问 I-7188XB(D)的存储器

4.8.1 使用 Flash 储存器

I-7188XB(D)模块包含 512K 字节的 Flash 储存器，其中包括 MiniOS7 操作系统占用的存储空间。MiniOS7 占用 0xF000 段. 因此用户可用的其它段的空间总共为 448K 字节。

Flash 存储器的每个位只能从 1 到 0 写入，不能从 0 到 1 写入。从将数据从 0 到 1 进行改变则只能使用 ***EraseFlash()*** 函数擦除 1 个区块的 Flash 存储器(64K 字节)。用户需要决定是写入区块还是擦除该区块

在 Flash 存储器的 0xD000 段到 0x1234 段写入整数的代码如下：

```
int data=0xAA55, data2;
char *dataptr;
int *dataptr2;

InitLib();    /* Initiate the 7188xb library */
dataptr=(char *)&data;
FlashWrite(0xd000, 0x1234, *dataptr++);
FlashWrite(0xd000, 0x1235, *dataptr);

/* read data from the Flash Memory method 1 */
dataptr=(char *)&data2;
*dataptr=FlashRead(0xd000, 0x1234);
*(dataptr+1)=FlashRead(0xd000, 0x1235);

/* read data from the Flash memory method 2 */
dataptr2=(int far *)_MK_FP(0xd000, 0x1234);
data=*data
```

从 Flash 存储器读取数据有点像在 SRAM 中读取数据。用户需要先分配一个指针指向存储位置，然后通过这个指针连接存储器。在向 Flash 写入数据前，用户需要先调用 ***FlashWrite()*** 函数，并且检验数据是否可被正确写入。之后，可用 ***EraseFlash()*** 函数清除该存储段。

请在如下地址参考 DEMO 程序
CD:\Napdos\7188XABC\7188XB\Demo\BC_TC\Memory

4.8.2 使用 RTC 和 NVRAM

I-7188XB(D) 模块包含 1 个 RTC 和 NVRAM，这两个组件在一个芯片内，由板载的锂电池提供备份至少 10 年的供电。RTC 的特性如下：

- BIOS 支持 RTC 时间和数据
- MiniOS7 支持 RTC 时间和数据
- 月的秒、分钟、小时、日
- 月、星期、年(最大到 2079)
- NVSRAM: 31 字节

NVRAM 可无限次读写，其特性如下：

- 数据有效期: 10 年
- 读写次数: 不限
- 共 31 字节

ReadNVRAM() 函数可用于读取 NVRAM 的数据，***WriteNVRAM()*** 可在 NVRAM 中写入数据。在 NVRAM 的地址 0 写入数据的代码如下：

```
int data=0x55, data2;

InitLib();    /* Initiate the 7188xb library */
WriteNVRAM(0, data);
data2= ReadNVRAM(0);       /* now data2=data=0x55 */
```

在 NVRAM 中写入 1 个整数数据的代码如下：

```
int data=0xAA55, data2;
char *daraptr=(char *)&data;

InitLib();    /* Initiate the 7188xb library */
WriteNVRAM(0, *dataptr);          /* write the low byte */
WriteNVRAM(1, *dataptr+1);        /* write the high byte */
dadaptr=(char *)&data2;
*dataptr=ReadNVRAM(0);            /* read the low byte */
(*dataptr+1)=ReadNVRAM(1);        /* read the high byte */
/* now data2=data=0xAA55 */
```

详细代码信息请参考
CD:\Napdos\7188XABC\7188XB\Demo\BC_TC\Memory

4.8.3 使用 EEPROM

EEPROM 用来存储不需经常变动的数据，比如：

- 模块 ID,配置设定
- COM 端口配置设定
- 小型数据库

EEPROM 的擦/写周期是 1,000,000 次,因此在测试中应该尽量少使用 EEPROM。EEPROM 最小可擦/写一个字节的数据，因此非常适合实际的应用场合。

I-7188XB(D) 提供 2K 的 EEPROM 存储器，包含 8 个区块并且每个区块有 256 个字节，这样 EEPROM 具有 2048 个字节的存储空间。通常，EEPROM 处于保护模式，意味着数据无法写入 EEPROM。在写入数据前需调用 ***EE_WriteEnable()*** 函数来改变 EEPROM 的保护状态。

例如：在 EEPROM 的区块 1, 地址 10 写入数据时，首先调用 ***EE_WriteEnable()*** 函数。代码示范如下。

```
int data=0x55, data2;

InitLib();   /* Initiate the 7188xb library */
EE_WriteEnable();
EE_MultiWrite(1, 10, 1, &data);
EE_WriteProtect();

EE_MultiRead(1, 10, 1, &data2);      /* now data2=data=0x55 */
```

注意: 在 EEPROM 中写入整数时,需要调用两次 ***EE_WriteEnable()*** 函数，在 NVRAM 中写入整数时采用同样的方式。

请在如下地址参考 DEMO 程序
CD:\Napdos\7188XABC\7188XB\Demo\BC_TC\Memory

4.9 使用看门狗

I-7188XB(D)的看门狗计时器在 MiniOS7 2.0 下固定为 0.8 秒。 当 I-7188XB(D)初次启动时，看门狗计时器也将启动。当看门狗计时器在 0.8 秒内没有刷新，它将重新启动 I-7188XB(D)。

I-7188XB(D)的 MiniOS7 操作系统在系统启动后将自动刷新看门狗计时器。用户程序可利用软件驱动停止 MiniOS7 刷新看门狗计时器，但是程序必须去刷新看门狗计时器。如果程序没有在每 0.8 秒去刷新看门狗计时器，将导致 I-7188XB(D)重新启动。

程序必须告知 MiniOS7 操作系统重置看门狗计时器，然后返回 MiniOS7 命令模式。

使用 ***EnableWDT()*** 函数启动看门狗功能，或者使用 ***DisableWDT()***函数停止使用。当看门狗启动后，程序需使用 ***RefreshWDT()*** 函数在 0.8 秒内刷新看门狗计时器，否则看门狗将从新启动 I-7188XB(D)模块。例程代码如下：

```
InitLib();    /* Initiate the 7188xb library */
EnableWDT();
while(!quit)
{
   RefreshWDT();
   User_function();
}
DisableWDT();
```

IsResetByWatchDogTimer() 函数可用来确认 I-7188XB(D)模块是否被看门狗重启。这个函数必须插入在程序的开始部分，代码如下：

```
main()
{
   InitLib();    /* Initiate the 7188xb library */

   if(IsResetByWatchDogTimer())
   {
      /* do something here to check the system */
   }
   quit=0;
   EnableWDT();
   while(!quit)
   {
      RefreshWDT();
      User_function();
   }
```

详细的 DEMO 例程请参考如下地址
CD:\Napdos\7188XABC\7188XB\Demo\BC_TC\Misc

4.10 使用计时器功能

I-7188XB(D) 能支持一个主时间计时器，8 个 StopWatch 计时器和 8 个 CountDown 计时器。I-7188XB(D)使用一个 16 位计时器实现这些计时器函数(有效单位 1ms)。函数 ***InstallUserTimer()*** 用于安装一个用户计时器函数且该函数在每隔 1ms 时间间隔内调用。MiniOS7 系统计时器将在每隔 1ms 调用 INT 9 ，每隔 55ms 调用 INT 0x1C。库文件的计时器函数与 INT9 相联系，并调用用户计时器函数。

TimerOpen() 函数用来启用 I-7188XB(D) 的计时器，并且这个函数可插入程序的开始部分。***TimerClose()*** **函数用来停止计时器功能。**例程代码如下：

```
unsigned long time iTime;

InitLib();    /* Initiate the 7188xb library */
TimerOpen();     /* Begin using the 7188XB timer function */
while(!quit)
{
   if(Kbhit())
      TimerResetValue();      /* Reset the main time ticks to 0 */

   iTime=TimerReadValue();    /* Read main time ticks */
}
TimerClose();    /* Stop using the 7188XB timer function */
```

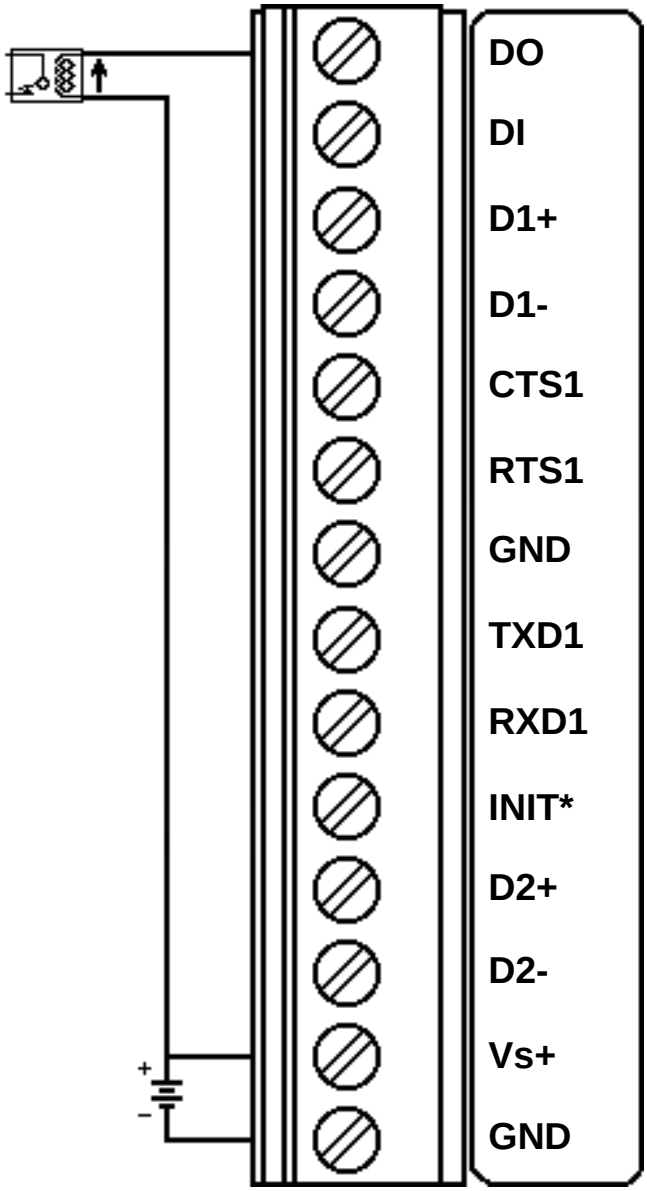
参考详细的例程信息请访问如下地址
CD:\Napdos\7188XABC\7188XB\Demo\BC_TC\Timer

4.11 使用开关量输入/输出功能

I-7188XB(D) 提供 1 路 DI 和 1 路 DO。***SetDo1High()*** 与 ***SetDo1Low()*** 函数可用来控制 D0 通道，***GetDi1()*** 读取 DI 通道的状态。

关于 DI 和 DO 的配线信息请参考 **1.4.6 DI 和 DO 配线连接。**

DO 应用的配线连接请参考下图：



得到和设定 DI 和 DO 状态的代码如下：

```
int Do1;

InitLib();    /* Initiate the 7188xb library */
Print("DI=%s\n\r", GetDi1()?"High":"Low"); /* Read the state of DI */
Do1=GetDo1();    /* Read the state of DO */
Print("DO=%s\n\r", Do1?"High":"Low");
if(!Do1)
   SetDo1High();      /* Set the DO1 to ON */
else
   SetDo1Low();       /* Set the DO1 to OFF */
```

更多 DEMO 程序信息请参考
CD:\Napdos\7188XABC\7188XB\Demo\BC_TC\IO_Pin

4.12 使用 I/O 拓展总线

随着目前越来越多的串口设备的出现，I-7188XB(D)的 I/O 拓展总线也包含有串行和并行的通信接口。并行通信接口非常类似于 ISA 总线，因此旧的 ISA 总线设计经过很小的修改就可以移植到 I/O 拓展总线中。串行总线的 I/O 引脚可编程并且类似于 D/I 和 D/O。

这些串行设备的特点如下：

- 相比并行设备有更小的体积
- 相比并行设备功耗更低
- 更容易独立应用的设计

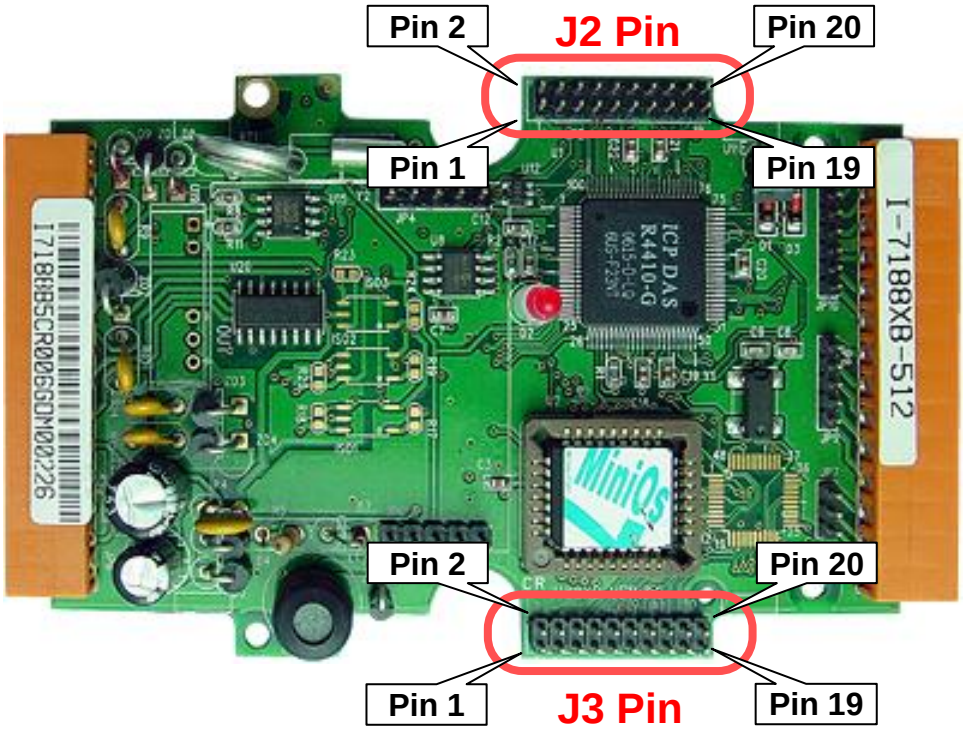
I/O 拓展总线的串行接口非常容易连接到这些串口设备。

4.12.1 I/O 拓展总线的定义

I-7188XB(D)模块的 I/O 拓展总线可被分成如下 3 组：

1. 电源供应和重启信号: VCC, GND, RESET 和/RESET
2. 并行总线:
   - 系统时钟: CLOCKA
   - 异步准备控制: ARDY
   - 地址总线: A0 ~ A7
   - 数据总线: D0 ~ D7
   - 接口控制 INT0, INT1 and INT4
   - 芯片选择和读/写滤波：/CS, /WR and /RD
3. 串行总线: TO_0, TO_1, TI_0, TI_1, SCLK, DIO9, DIO4 和 DIO14

I/O 拓展总线的定义如下:



J2 引脚的定义和描述:

No	名称	描述
1	GND	PCB 地
2	GND	PCB 地
3	CLOCKA	CPU 同步时钟输出
4	ARDY	异步准备输入 (电平感应, OPEN=ready)
5	INT0	通道 0 中断请求输入(异步, 高有效)
6	INT1	通道 1 中断请求输入(异步, 高有效)
7	VCC	PCB 电源供应
8	RESET	电源重启脉冲 (高有效)
9	GND	PCB 地
10	/RESET	电源重启脉冲 (低有效)
11	TO_0	CPU 时钟输出 0(可用于可编程的 D/I/O)
12	TO_1	CPU 时钟输出 1(可用于可编程的 D/I/O)
13	TI_0	CPU 时钟输入 0(可用于可编程的 D/I/O)
14	TI_1	CPU 时钟输入 1(可用于可编程的 D/I/O)
15	SCLK	为 7188 系列模块提供的共享串行时钟输出
16	DIO9	可编程的 D/I/O 位
17	DIO4	可编程的 D/I/O 位

18	DIO14	可编程的 D/I/O 位
19	VCC	电源供应
20	VCC	电源供应

- **CLOCKA:** 40M
- **ARDY:** 此引脚预留 OPEN 给不需要等待状态的应用。
- **INT0 和 INT1:** 这 2 个引脚预留 OPEN 给不需要中断的应用。
- **TO_0 and TO_1:** 这 2 个引脚可用于 CPU 的时钟输出或可编程的 DI/O。
- **TI_0 and TI_1:** 这 2 个引脚可用于 CPU 的时钟输入或可编程的 DI/O。
- **DIO4, DIO9 and DIO14:** 可编程的 DIO 位。
- **SCLK:** The I-7188XB(D)利用此信号作为时钟信号源去驱动所有的板载串行设备，因此它的编程方式类似于 DO。改变该信号的配置将导致严重的错误。这个信号可用来驱动外部的串口设备而不需要其它的辅助信号。

J3 引脚定义和描述：

No	名称	描述
1	A0	地址总线
2	D0	数据总线
3	A1	地址总线
4	D1	数据总线
5	A2	地址总线
6	D2	数据总线
7	A3	地址总线
8	D3	数据总线
9	A4	地址总线
10	D4	数据总线
11	A5	地址总线
12	D5	数据总线
13	A6	地址总线
14	D6	数据总线
15	A7	地址总线
16	D7	数据总线
17	INT4	通道 4 中断需要的输入(异步，高有效)

18	/WR	写滤波输出（同步，低有效）
19	/CS	芯片选择输出（同步，低有效）
20	/RD	读滤波输出（同步，低有效）

- **地址总线(输出):** A0 ~ A7
- **数据总线 (三态, 双向):** D0 ~ D7
- **INT4:** 这个引脚预留 OPEN 给不需要中断的应用。
- **/CS, /RD 和 WR:** 这三个引脚于 CLOCKA (Pin3 of JP2)同步和 ARDY (Pin4 of JP2)异步。
- 在程序需要通过 I/O 地址 0 到 0xff 进行数据输入/输出时/CS 将被激活。

注意:更详细信息请参考 “**iobus_e.pdf**”。

4.12.2 I/O 扩展板

面包板和测试板：

型号	描述
X002	面包板
X004	测试板
X005	面包板 (小尺寸)
X006	面包板 (大尺寸)

DI 和 DO 扩展板：

型号	描述
X107	6 个 DI 、 7 个 DO
X109	7 个继电器输出
X110	14 个 DI
X111	13 个 DO
X116	4 个 DI 、6 个继电器输出 (即将推出)
X119	7 个 DI 、7 个 DO (即将推出)

A/D, D/A 和多功能板：

型号	描述
X202	7 路 A/D (0~20mA)
X203	2 个 DI 、 6 个 DO 、2 路 A/D (0~20mA)

X303	4 个 DI、 6 个 DO 、 1 路 A/D (+/-5V)、 1 路 D/A (+/-5V)
X304	4 个 DI 、4 个 DO、3 路 A/D (+/-5V) 、 1 路 D/A (+/-5V)
X305	2 个 DI、 2 个 DO、7 路 A/D (+/-5V) 、 1 路 D/A (+/-5V)
X308	6 个 DO c、 4 路 A/D (0~10V)
X310	3 个 D、 3 个 DO 、 2 路 A/D (0~20mA/0~10V) 、2 路 D/A (0~10V)

RS-232/422/485 通信扩展版：

型号	描述
X503	1 个 5 线 RS-232
X504	1 个 5 线 RS-232、 1 个 9 线 RS-232 通道
X505	3 个 5 线 RS-232 通道
X506	6 个 3 线 RS-232 通道
X507	4 个 D、 4 个 DO 、 1 个 RS-422 通道 (TxD+、TxD-、RxD+ 、RxD-)
X508	4 个 DI 、4 个 DO、1 个 5 线 RS-232 通道
X509	4 个 DI 、4 个 DO 、2 个 3 线 RS-232 通道
X510	5 个 DI 、5 个 DO 、1 个 3 线 RS-232 通道、 2 个 128K bytes EEPROM
X510-128	5 个 DI 、5 个 DO 、1 个 3 线 RS-232 通道、 128K bytes EEPROM
X511	3 个 RS-485 通道
X518	8 个 DO 和 1 个 5 线 RS-232 通道
X560	3 个 3 线 RS-232 通道、8M bytes NAND Flash

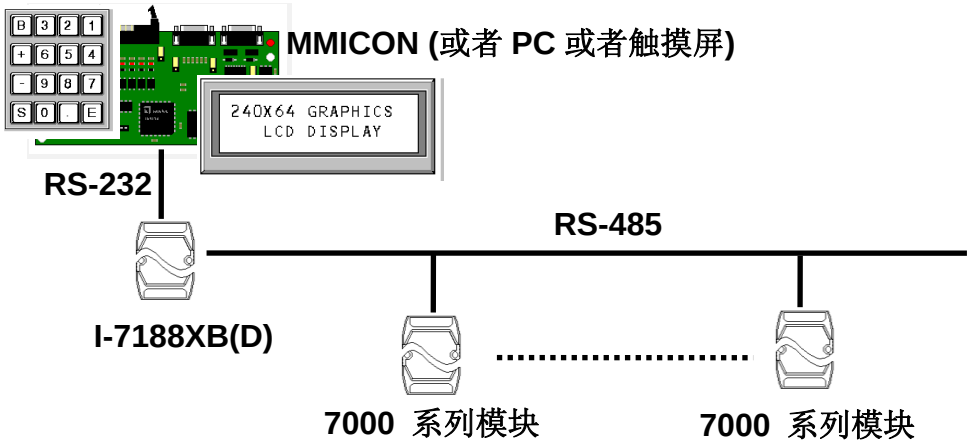
存储器扩展板：

型号	描述
X600	4M bytes NAND Flash
X601	8M bytes NAND Flash
X607	128K 电池备份 SRAM
X608	512K 电池备份 SRAM

注意: 更详细信息请参考“**iobus_e.pdf**”。

5. 应用

5.1 嵌入式控制器



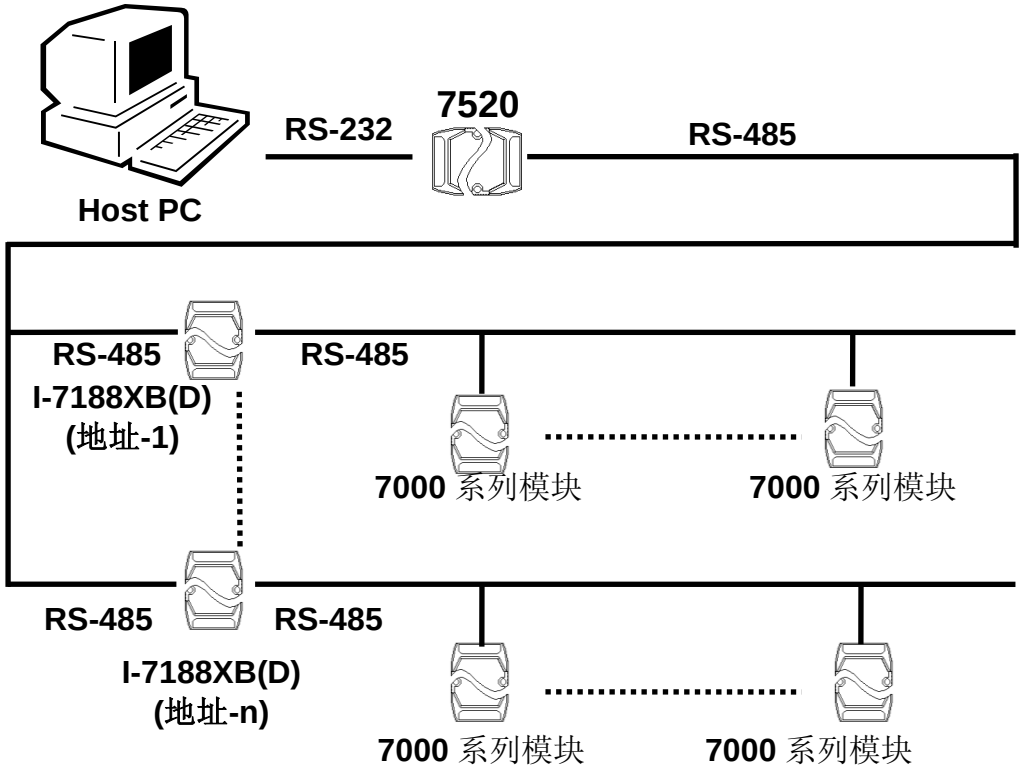
应用：

- **4500 系列产品的取代和增进（不兼容）**
- **替代 PC-based 控制器**
- **替代 PLC 控制器**
- **替代其它专用控制器**

I-7188XB(D)在一般的应用中可作为嵌入式控制器，意味着它可以用于代替一个主机 PC、PLC 或者其它的专用控制器。

编程工具	TC/BC/MSC
调试工具	通过标准的输入/输出 (PC 的键盘和显示器)
人机界面	• MMICON • PC 的键盘和显示器 • 触摸屏 (RS-232 或 RS-485 接口)
程序	存储在 Flash Memory
输入/输出	• 板载 DI 或 DO • I/O 拓展总线 • 7000 系列模块，最大可连接 256 个 7000 系列模块提供上千点 I/O

5.2 现场实时控制器 (RTC)



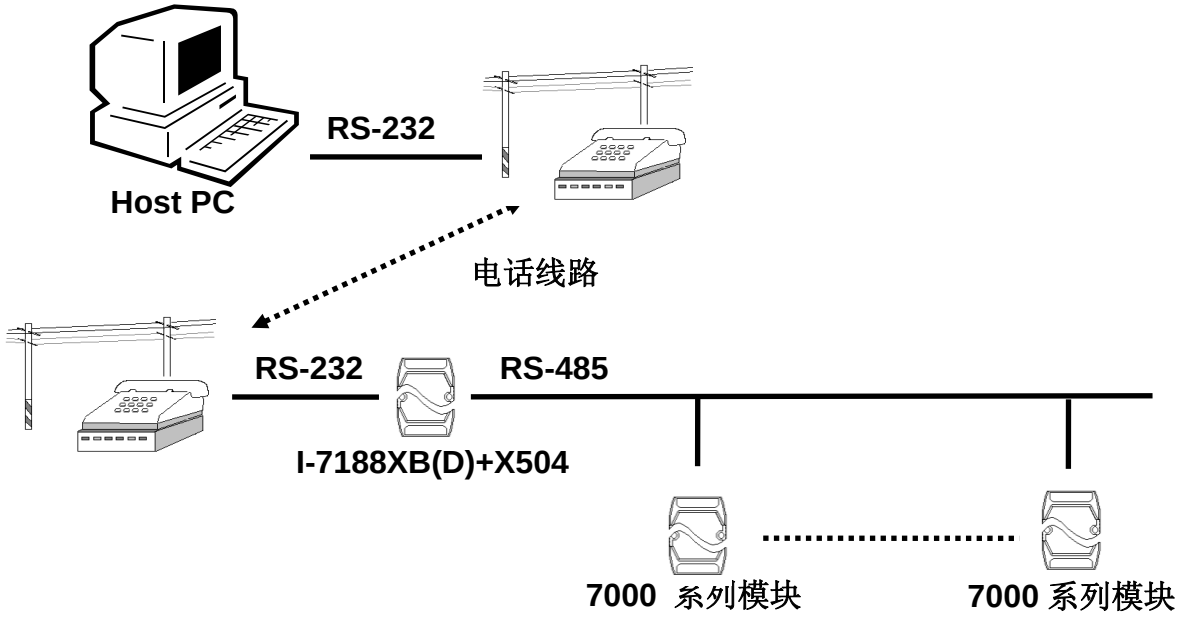
在一般应用架构中，7000 系列模块可作为从机设备，控制程序程序在主机 PC 中执行。运作流程如下：

- PC 发送命令到 7000 系列模块并且接收输入的数据。
- PC 分析这些输入的数据并且生成输出的数据。
- PC 发送命令到 7000 系列模块作为输出数据。

如果应用中有上百个 7000 系列模块，这将消耗 PC 大量的资源去分析和控制这些模块。因此，控制程序可在 I-7188XB(D)中执行，此时 PC 只需要发送控制策略到 I-7188XB(D)，由本地的 I-7188XB(D)根据这些策略来控制 7000 系列模块。通过这种方式，上千个 7000 系列模块都可以由 PC 通过 I-7188XB(D)进行控制。

有些应用对控制功能有实时要求，所以现场 I-7188XB(D)可进行实时控制，而无需由主机 PC 参与。

5.3 远程控制器



在该应用构架中，控制程序在 I-7188XB(D)中执行，I-7188XB(D)可根据控制要求直接控制现场的 7000 系列模块。

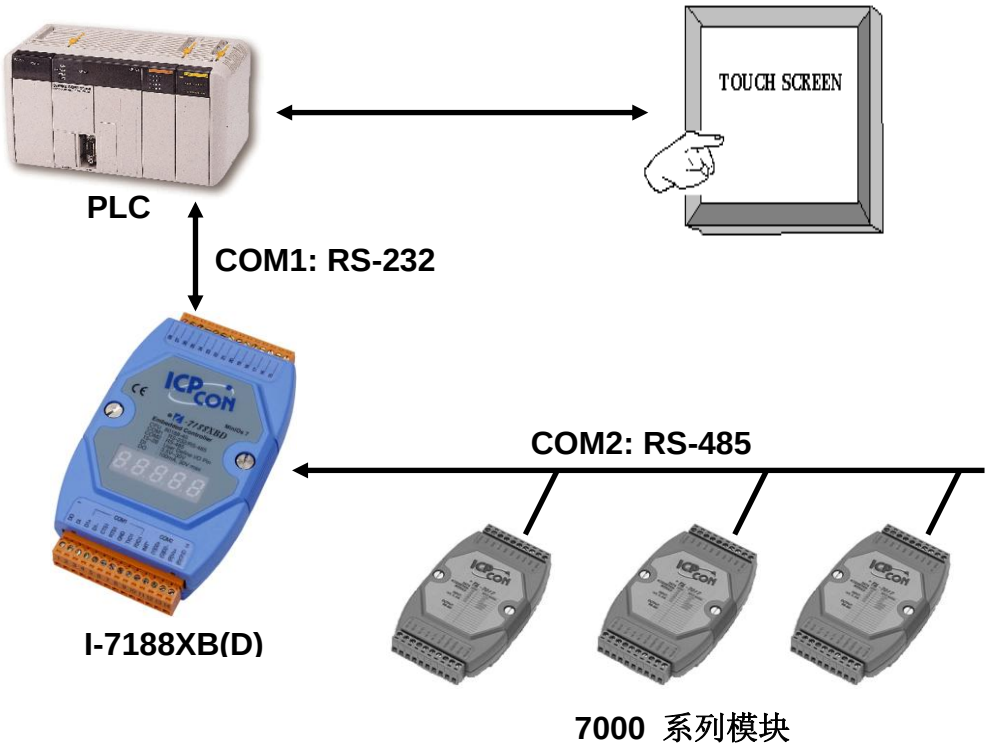
主机 PC 可访问远程的 I-7188XB(D)：

- 查询和记录远程系统的状态
- 下载控制策略到远程的 I-7188XB(D)

远程的 I-7188XB(D)与主机 PC 通信：

- 紧急时间报警和反馈
- 远程系统状态报告

5.4 PLC I/O 拓展应用



大多数 PLC 系包含一个源于人机界面应用的人机界面。 I-7188XB(D)可利用这个界面建立一个 PLC 与 7000 模块之间的通信桥梁。

I-7188XB(D)可直接读/写 PLC 的内部寄存器，意味着 PLC 可按如下方式连接 7000 系列模块的输入模块：

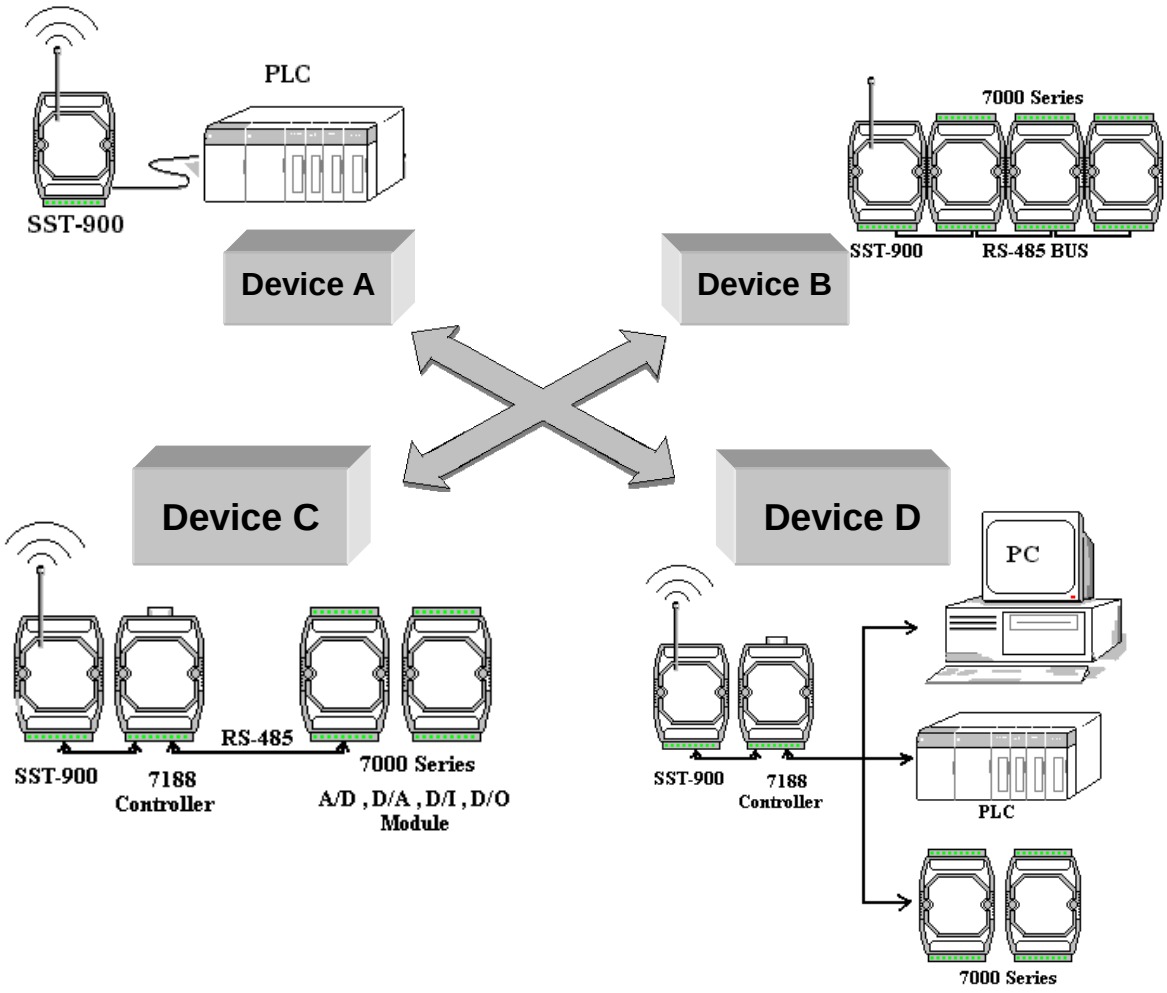
- I-7188XB(D) 发送命令到 7000 系列的输入模块
- I-7188XB(D) 将此数据写到 PLC 的内部寄存器
- PLC 通过内部寄存器得到该数据

PLC 可按如下步骤控制 7000 系列的输出模块：

- PLC 将数据写到内部寄存器
- I-7188XB(D)从 PLC 的内部寄存器读取这些数据
- I-7188XB(D) 发送相应的命令到 7000 系列输出模块

通过这种方式，7000 模块的输入数据可显示在触摸屏上，拓展的 7000 系列输出模块也可以通过触摸屏进行输出控制。

5.5 无线 Modem 应用



SST-900/SST-2400 设置: (Device A)

- RS-232
- 半双工模式
- 同步方式
- 从站状态
- 波特率：9600
- 通道=3
- 频率=915.968MHz

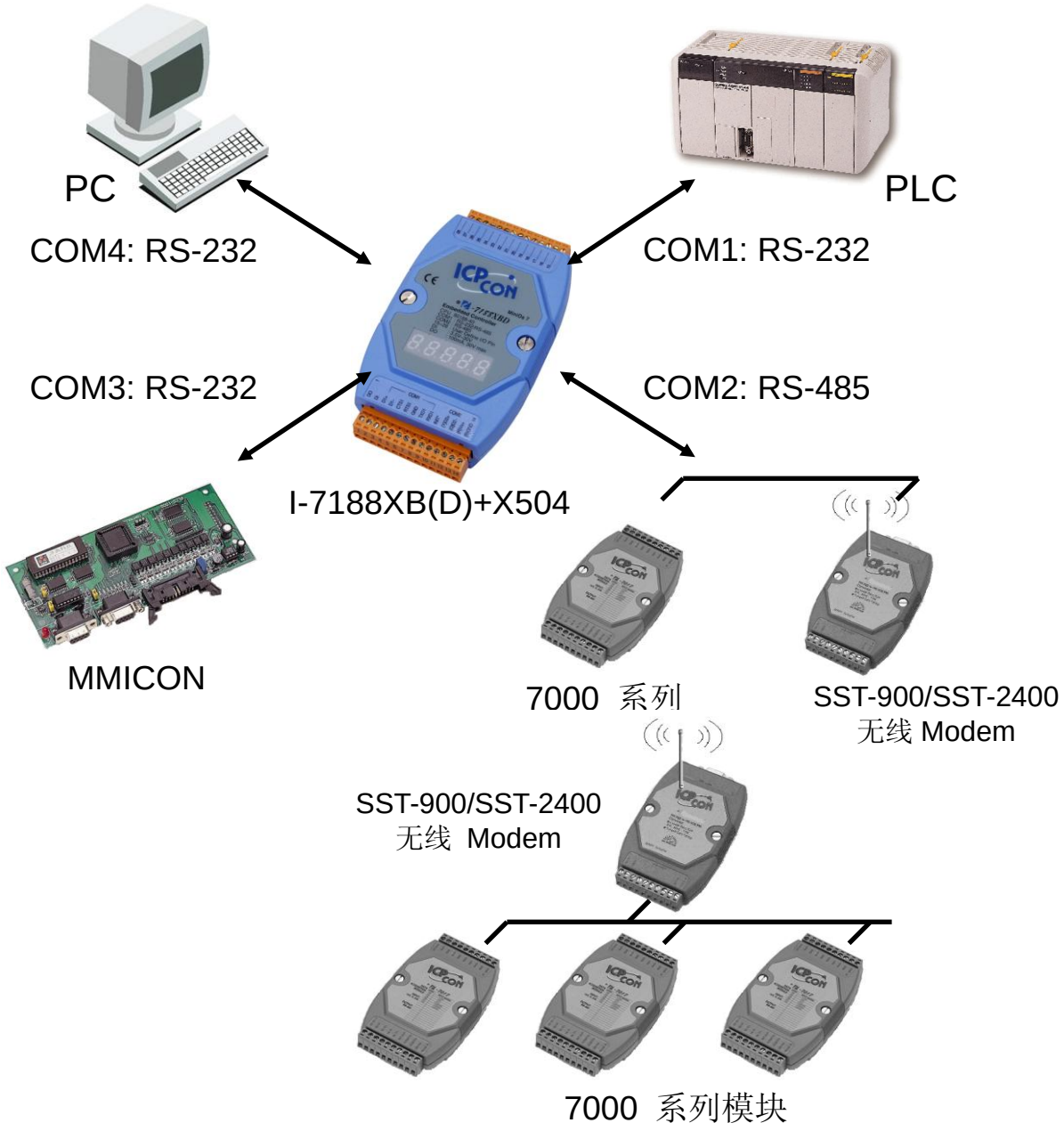
SST-900/SST-2400 设置: (Device B/C/D)

- RS-485 或 RS-232
- 半双工模式

- 同步方式
- 从站状态
- 波特率：9600
- 通道=3
- 频率=915.968MH

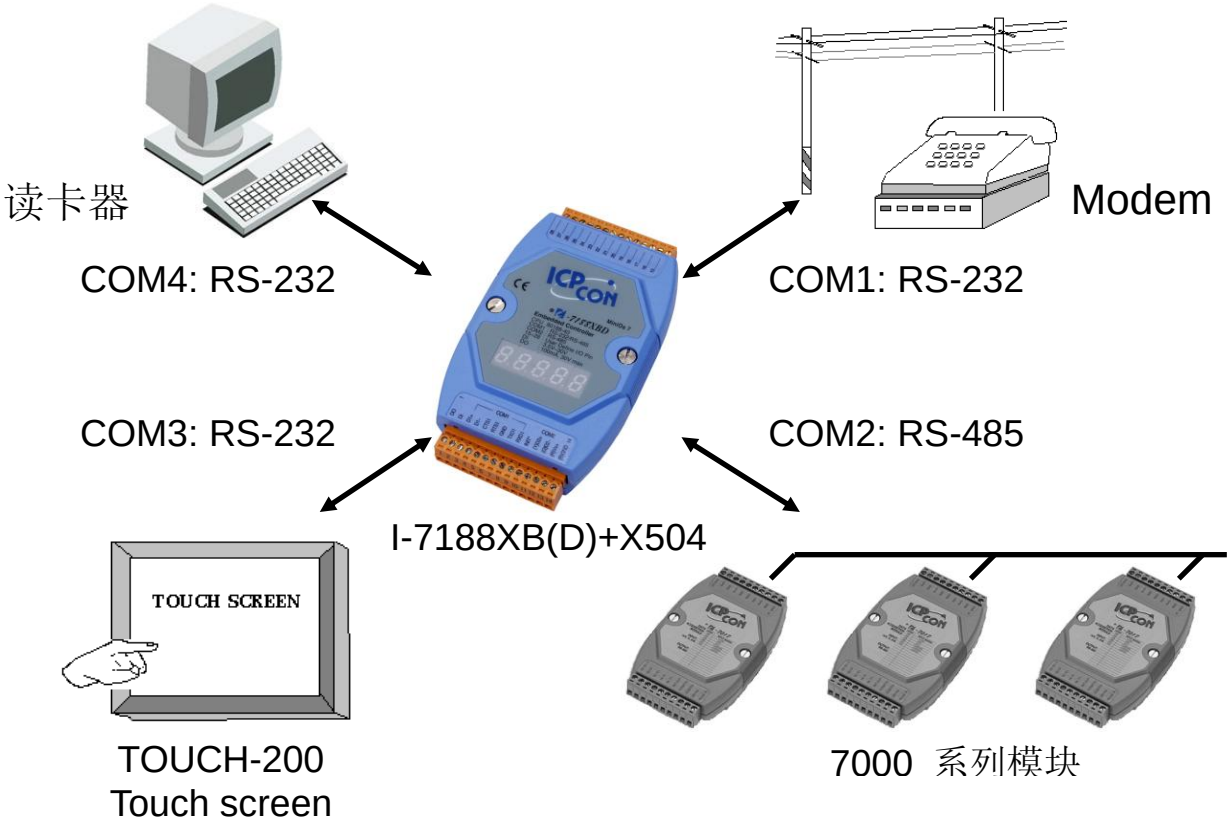
既然 I-7188XB(D) 可作为嵌入式控制器，它就可以成为 SST-900 和任何外部设备之间的桥梁，例如 PLC、控制器或其它 7000 系列模块 。

5.6 4 COM 端口应用 (1)



COM1: PLC 可用来访问 7000 系列模块的 I/O 状态
COM2: 可用来控制 7000 系列模块的输入/输出
COM3: MMICON 可作为 MMI
COM4: PC 利用这个端口监视和记录系统数据

5.7 4 COM 端口应用(2)



COM1: 连接远程主机 PC
COM2: 直接控制 7000 模块的输入/输出
COM3: TOUCH-200 可作为本地 MMI
COM4: 读卡器可作为标准输入设备

附录 A: 什么是 MiniOS7

MiniOS7 是由 泓格科技（ICP DAS）开发的嵌入式操作系统，它用来在 7188 系列模块中取代 ROM-DOS 操作系统。一些公司已经创建了很多品牌的 DOS 操作系统，所有的 DOS 不管是 PC-DOS、MS-DOS 还是 ROMDOS) 都是一个用来告诉计算机如何处理信息的指令集或代码，它可以执行程序、管理文件、控制信息处理、指定输入和输出，并执行许多其它相关的功能。

MiniOS7 提供与 ROM-DOS 相同的功能， 另外 I-7188XB(D)还可为用户提供很多特别的功能。

以下是 MiniOS7 和 ROM-DOS 的对照表

	MiniOS7	**ROM-DOS**
启动时间	0.1 sec	4 ~ 5 sec
占用存储空间	<64K bytes	64K bytes
支持 I/O 拓展总线	Yes	No
支持 ASIC Key	Yes	No
Flash ROM 管理	Yes	No
更新 O.S. (由串口线下载更新)	Yes	No
内嵌硬件诊断功能	Yes	No
直接控制 7000 系列模块	Yes	No
用户 ODM 功能	Yes	No
免费	Yes	No

注意: ICP DAS 有权改变 MiniOS7 的规格而无需声明。

MiniOS7 的典型命令套件

命令	**描述**
LED5 pos value	显示一个 HEX 的值在 5 位 LED 的指定位置
USE NVRAM	读/写 NVRAM
USE EEPROM	读/写 EEPROM
USE FLASH	读/写 Flash Memory
USE COM2 /option	从 COM2 端口发送/接收数据
DATE [mm/dd/yyyy]	设置 RTC 的时间

Time [hh:mm:ss]	设置 RTC 的时间
MCB	检测当前存储器
UPLOAD	在 I-7188XB(D)的 SRAM 中存储 MiniOS7 镜像文件 (此命令用来更新操作系统)
BIOS1	在 I-7188XB(D)的 Flash memory 中存储 MiniOS7 镜像文件 (此命令用来更新操作系统)
LOAD	下载程序到 I-7188XB(D)的 Flash Memory
DIR [/crc]	显示 I-7188XB(D)的 Flash Memory 上的文件信息
RUN fileno	通过文件编号执行文件
Filename	通过文件名称执行文件
DELETE or DEL	删除所有文件。
RESET	复位
DIAG [option]	执行硬件诊断
BAUD baudrate	设置 COM1 的波特率
TYPE filename [/b]	显示文件内容
REP [/#] command	重复执行同样的指令
RESERVE [n]	为储存程序保留 n 个 Flash Memory 扇区
LOADR	将文件下载到 SRAM
RUNR [option]	在 I-7188XB(D)的 SRAM 运行程序 (使用 LOADR 命令下载程序)
I/INP/IW/INPW port	从硬件端口读取数据
O/OUTP/OW/OUTPW port value	向硬件端口发送数据
更多 ……	

Note: 关于 MiniOS7 的更多信息请参考
CD:\Napdos\MiniOS7\document\Lib_Manual_For_7188XABC\index.htm
或者
http://ftp.icpdas.com/pub/cd/8000cd/napdos/minios7/document/lib_manual_for_7188xabc/index.htm

附录 B: MiniOS7 Utility 和 7188XW

MiniOS7 Utility 和 7188xw.exe 应用程序都可以帮助用户简单地更新 MiniOS7 到最新的版本。MiniOS7 Utility 和 7188xw.exe 应用程序执行今本的配置功能和下载应用程序到嵌入在 I-7188XB(D)控制器中。

MiniOS7 Utility

MiniOS7 Utility 程序提供以下三个主要功能：

- 更新 MiniOS7 镜像文件
- 下载程序到 Flash Memory
- 配置 COM 端口设置

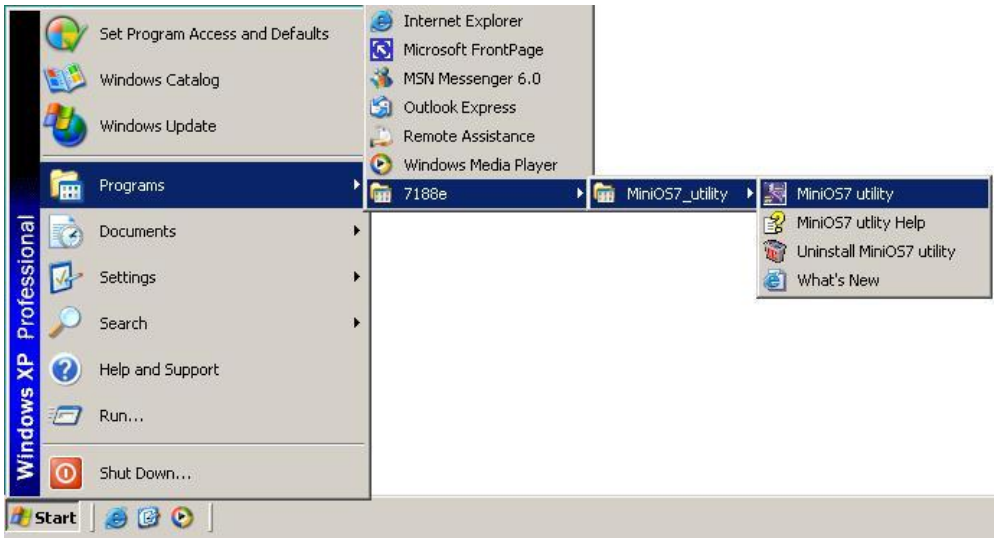
MiniOS7 utility 位置

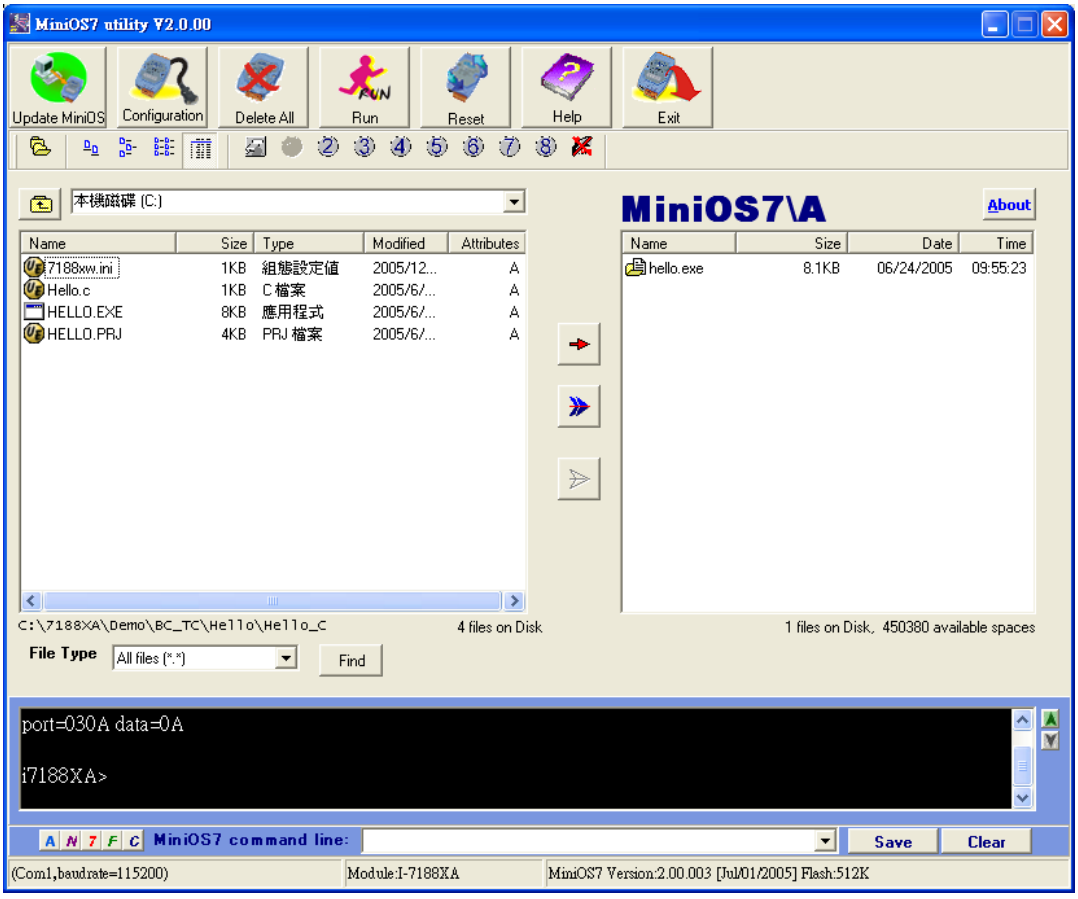
MiniOS7 utility 在如下地址可以找到
CD:\NAPDOS\MINIOS7\UTILITY\MiniOS7_utility 或
http://ftp.icpdas.com/pub/cd/8000cd/napdos/minios7/utility/minios7_utility/

安装步骤

步骤 1: 找到并执行 **minios7_utility_v2000.exe** 在
CD:\Napdos\MiniOS7\utility\MiniOS7_utility 文件夹中

步骤 2: 安装完成后一个新的文件夹**7188E**将出现在PC的开始目录的程序选项中。展开这个文件夹即可访问 MiniOS7 Utility 文件。详细信息请见如下图表：





7188XW

7188xw.exe 应用程序是 I-7188XB(D)最主要的工具，可用来执行如下功能：

- **从主机 PC 下载程序到 I-7188XB(D)模块的存储单元。**
- **下载 MiniOS7 镜像文件到 I-7188XB(D)控制器的 Flash Memory，从而来可升级 MiniOS7 操作系统。**
- **在主机 PC 的显示器上显示调试字符串。**
  三个标准的输出功能库，比如 Putch、Print 和 Puts 函数，可让主控单元发送输出字符串到主机 PC 的显示器。
- **使用主机 PC 的键盘发送数据到 I-7188XB(D) 模块**
  三个标准的输入功能库，比如 Getch、Scanf 和 LineInput 函数，可让主控单元接收主机 PC 输入的字符串。

7188xw.exe 可在如下地址找到
CD:\Napdos\MiniOS7\utility\ 或网页
http://ftp.icpdas.com/pub/cd/8000cd/napdos/minios7/utility/

7188xw.exe 的 MiniOS7 命令行操作选项：

选项	描述
/c#	使用主机 PC 的 COM# 端口
/b#	设置主机 PC 上 COM 端口的波特率(默认为 115200)
/s#	设置显示器显示的行数 (默认 25，最大 50)

7188xw.exe 热键

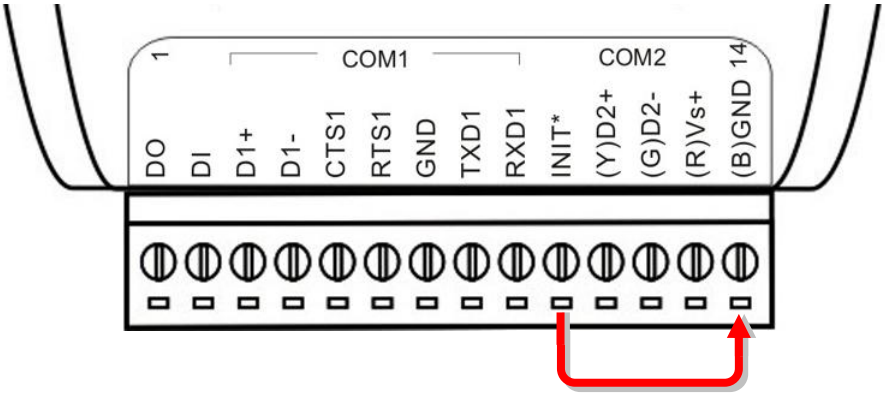
命令	描述
F1	显示 7188xw.exe 的帮助信息
Alt_F1	用中文（big5 码）进行显示 7188xw.exe 的帮助信息
Ctrl_F1	用中文（GB2312 码）进行显示 7188xw.exe 的帮助信息
Alt_1	使用主机 PC 的 COM1 端口
Alt_2	使用主机 PC 的 COM2 端口
Alt_3	使用主机 PC 的 COM3 端口
Alt_4	使用主机 PC 的 COM4 端口

Alt_5	使用主机 PC 的 COM5 端口
Alt_6	使用主机 PC 的 COM6 端口
Alt_7	使用主机 PC 的 COM7 端口
Alt_8	使用主机 PC 的 COM8 端口
Alt_9	使用主机 PC 的 COM9 端口
Alt_A	在普通模式和 ANSI-Escape-code-support 模式间切换
Alt_C	切换到命令模式，支持命令： b#:设置主机 PC 上 COM 端口的波特率 c#:使用主机 PC 的 COM#端口 n/e/o: 设置奇偶位为 none/even/odd 5/6/7/8: 设置数据位到 5/6/7/8 p#: 设置主机 PC 的工作目录 q: 推出命令模式
Alt_D	将 RTC 的日期设置成主机 PC 的日期
Alt_T	将 RTC 的时间设置成主机 PC 的时间
Alt_E	用于下载文件到存储器。屏幕上出现“Press ALT_E to download file!” 信息后点击 Alt_E
Alt_H	切换 Hex/ASCII 显示模式
Alt_L	切换 normal/line 模式。在 line 模式中，在点击 ENTER 键前所有的参数将不会发送到 COM 端口，它可用来测试 7000 系列模块
Alt_X	退出 7188xw.exe 应用程序
F2	设置下载文件名 (无需初始化下载操作)
F5	运行由 F2 指定的程序，并使用由 F6 设置的变量参数
F6	设置由 F2 指定的程序的变量(最大 10 个变量，如果设置少于 10 个需要在结尾加‘*’)
Ctrl_F6	清空屏幕
F8	F8=F9+F5
F9	将由 F2 下载的程序下载到 FLASH 存储器
Alt_F9	将所有用 ALT_F2 选定的文件下载到 FLASH 存储器
F10	将所有用 ALT_F2 选定的文件下载到 SRAM 并执行
Alt_F10	将所有用 ALT_F2 选定的文件下载到 SRAM 存储器
Ctrl_B	发送 BREAK 信号到主机 PC 表示此端口正被 7188xw.exe 占用
更多…	

更多关于 7188xw.exe 应用程序的详细信息请参考以下目录的 index.htm 文件
CD:\Napdos\MiniOS7\document\Lib_Manual_For_7188XABC\或者
http://ftp.icpdas.com.tw/pub/cd/8000cd/napdos/minios7/document/lib_manual_for_7188xabc/

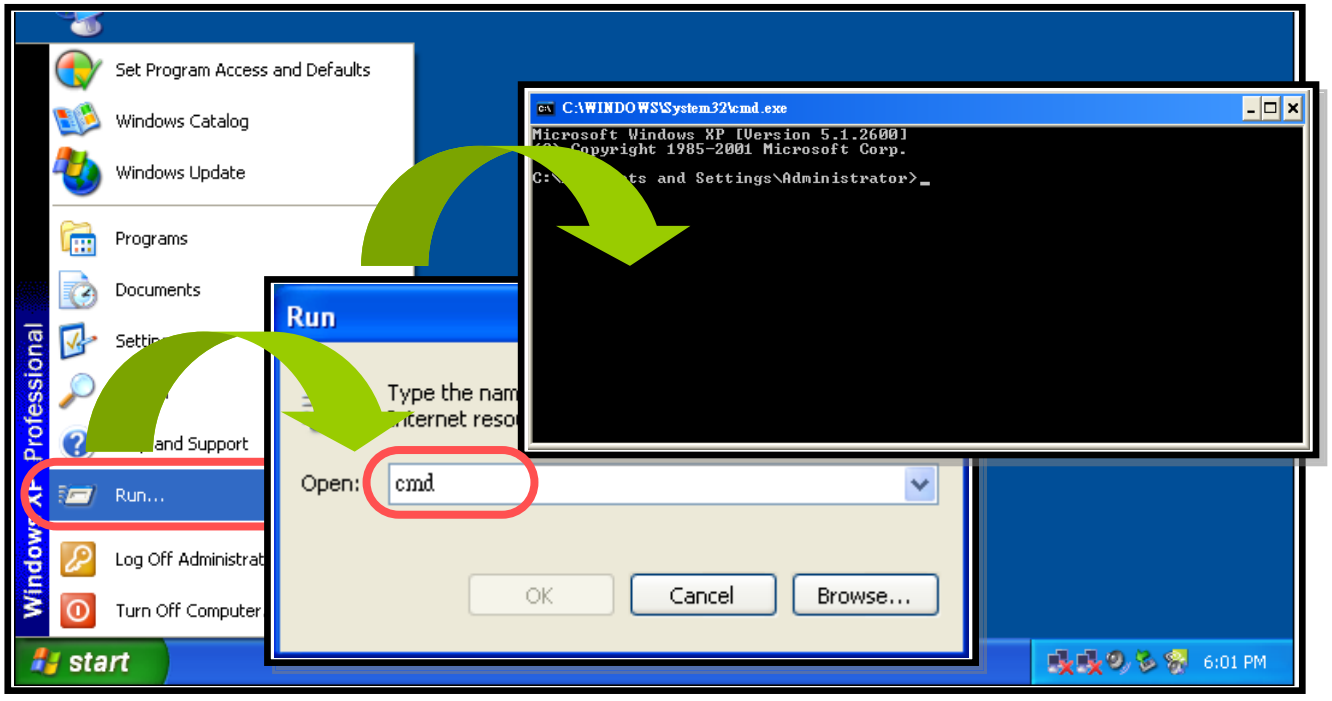
使用 7188xw.exe 程序下载文件到 I-7188XB(D)控制器

步骤 1: 关闭 I-7188XB(D)的电源，将 INIT*引脚 与 GND 引脚相连，然后接通 I-7188XB(D) 的电源。



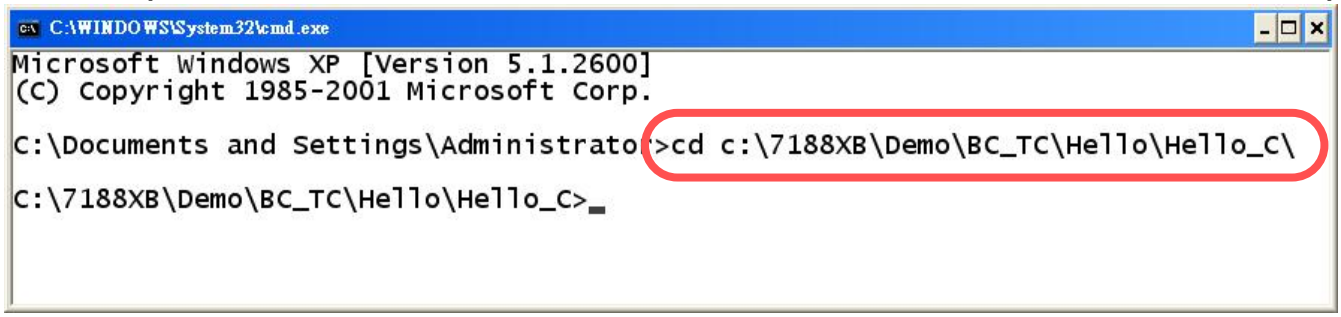
步骤 2: 当 I-7188XB(D) 启动后，断开 INIT*和 GND 引脚的连接。

步骤 3: 打开 MS-DOS 命令提示符窗口并按如下图步骤操作：

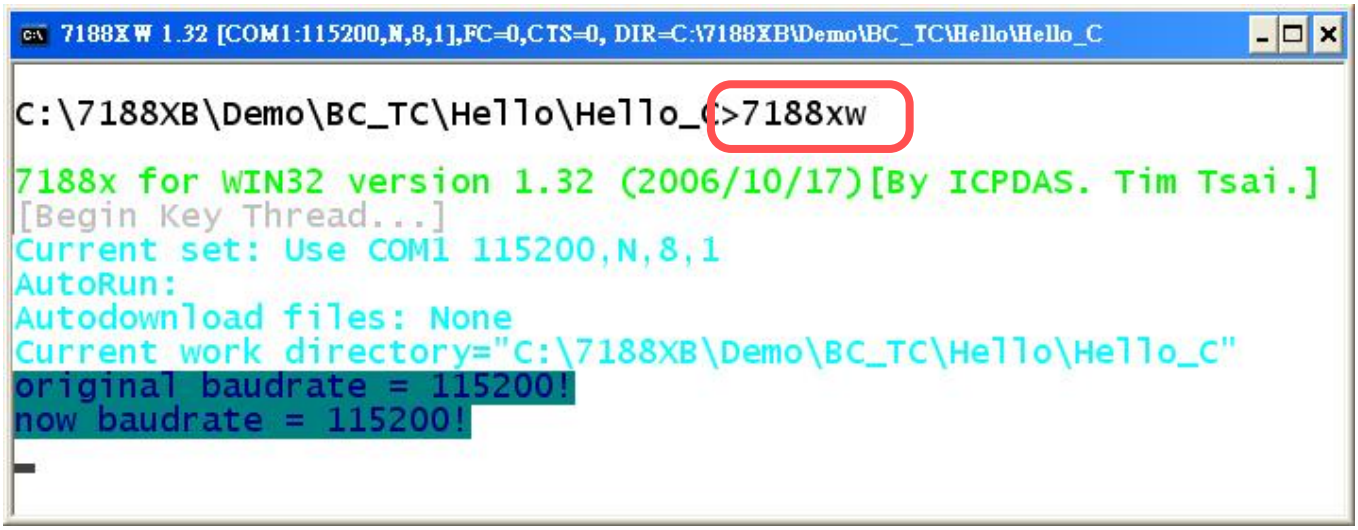


步骤 4: 输入 “cd c:\7188XB\Demo\BC_TC\Hello\Hello_C\”然后点击 **<Enter>**。

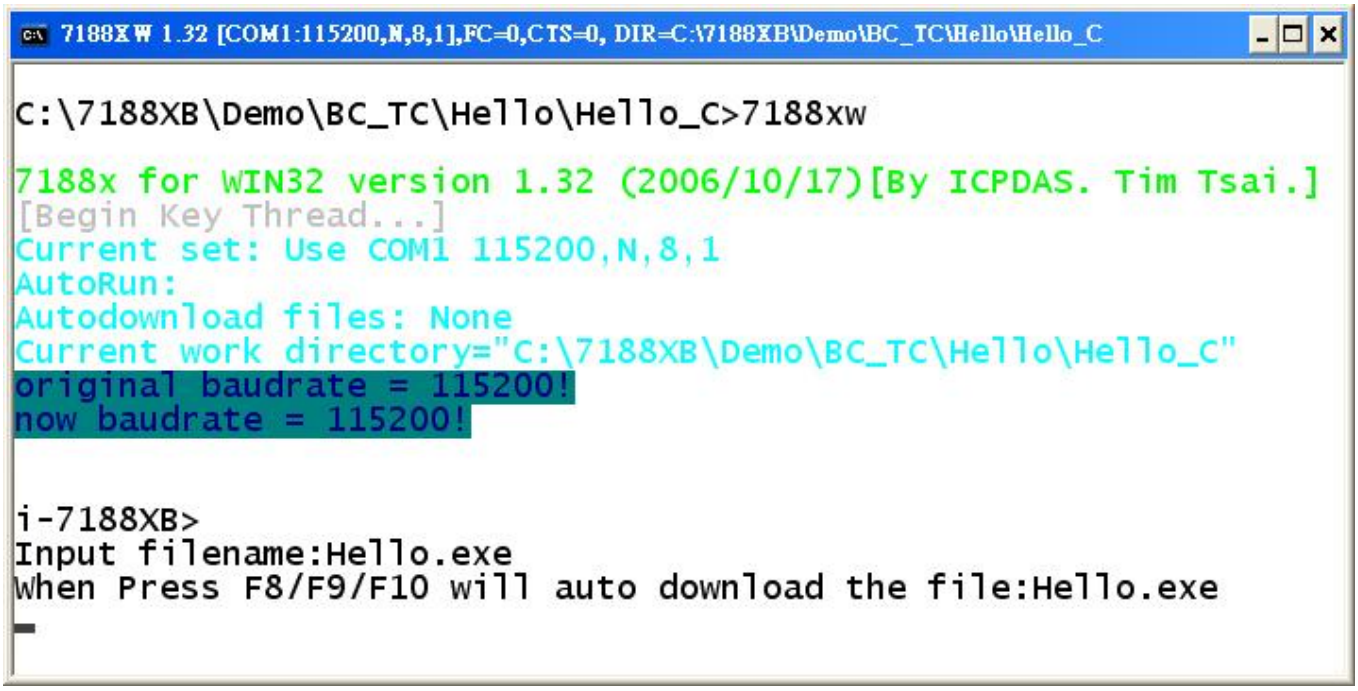
(假定用户复制 7188XB 文件夹到 C 盘，参考 Sec.2.1 的步骤 2)



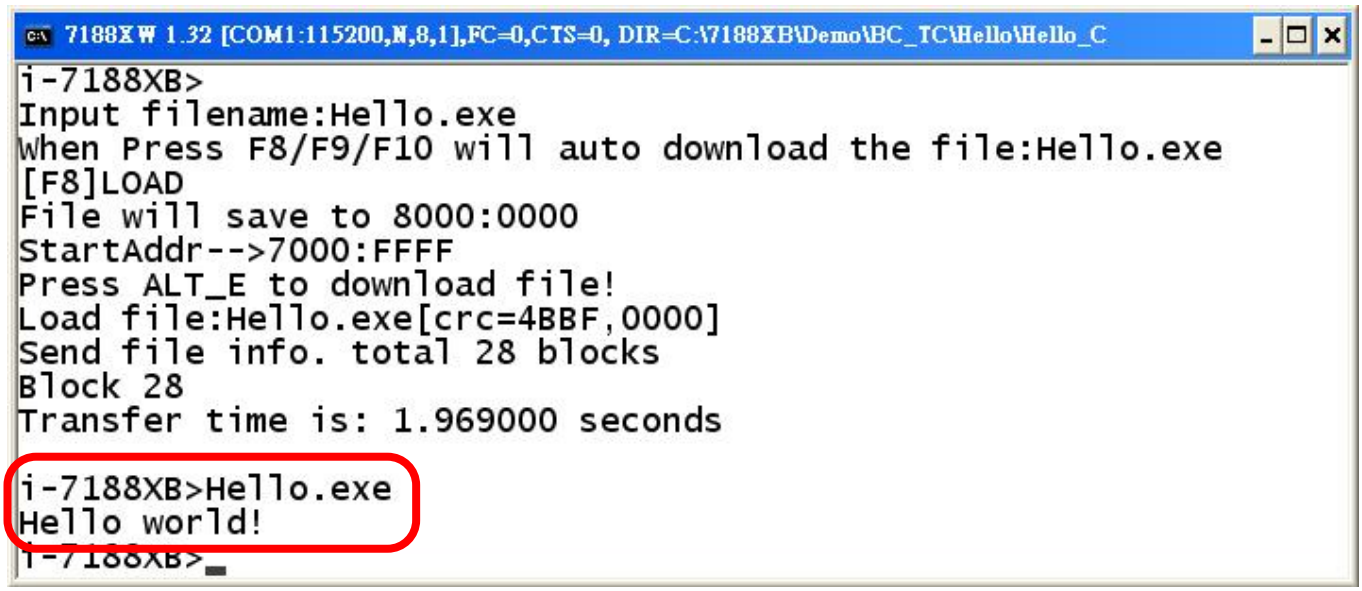
步骤 5: 执行 **7188xw.exe** 应用程序出现如下界面。



步骤 6: 点击 **<F2>**并且输入“**Hello.exe**” 并点击 **<Enter>**。



步骤 7: 点击 **<F8>** 来下载 Hello.exe 文件到 I-7188XB(D) 并执行这个程序。



注意: 热键功能的描述如下所示：

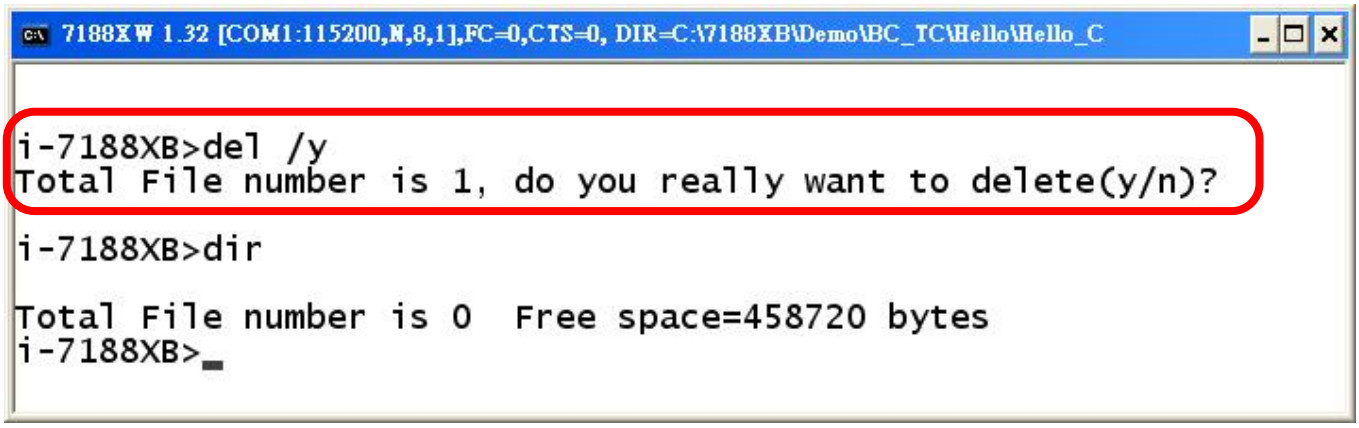
F8: 下载一个文件到 FLASH 存储器并执行该文件

F9: 下载一个文件到 FLASH 存储器

F10: 下载一个文件到 SRAM 并执行这个程序

步骤 8: 输入 “dir” 并点击 **<Enter>**确认文件是否已经保存到了 I-7188XB(D)的 Flash 存储器。

步骤 9: 输入 “**del /y**” 并点击 **<Enter>**删除 I-7188XB(D)的 Flash 存储器上已经存储的文件。



注意: MiniOS7 只支持 **delete all** 命令，单独的文件无法被选择删除。

步骤 10: 点击 **<Alt + X>** 推出 MiniOS7。

使用 7188xw.exe 程序升级 MiniOS7

步骤 1: 使用 CA0910 电缆将 I-7188XB(D)连接到主机 PC 的 COM 端口，详细信息请参考 2.2 章的内容。

步骤 2: 确定 MiniOS7 最新的镜像文件

镜像文件名的格式是： **TTYYMMDD.img**

TT: 产品类型

YY: 镜像文件发布的年份

MM: 镜像文件发布的月份

DD: 镜像文件发布的日期

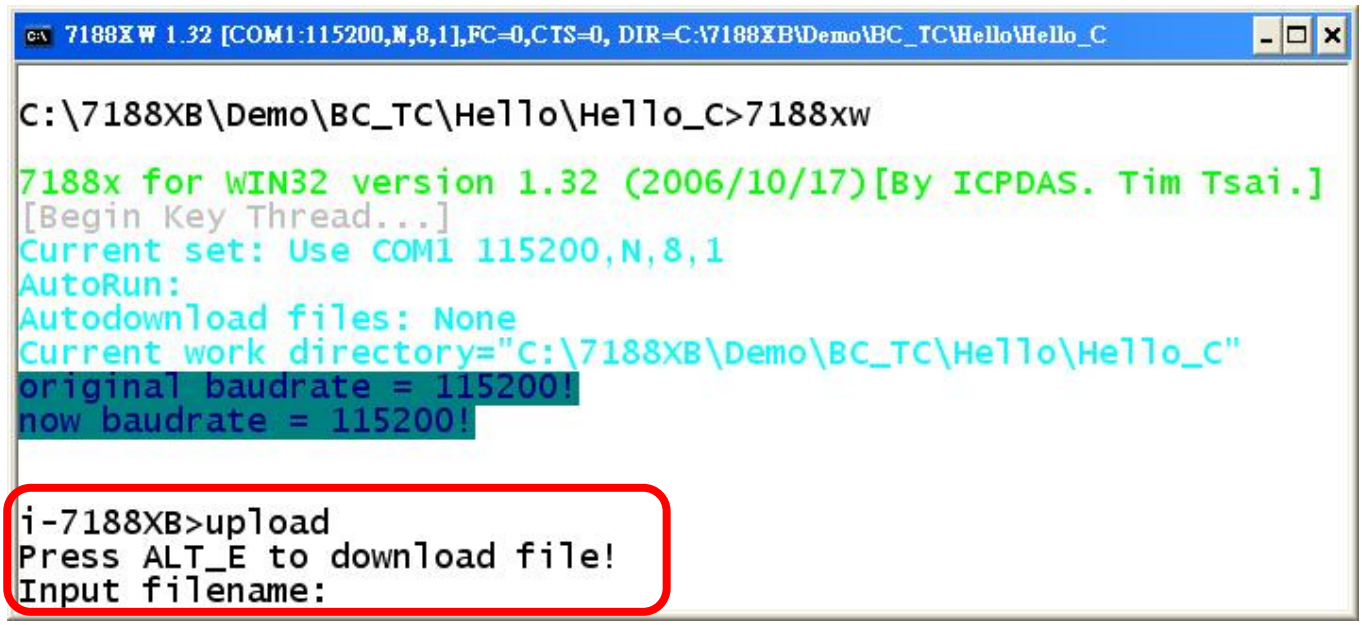
提示: MiniOS7 镜像文件可在 CD:\NAPDOS\MiniO7\找到。

最新的 MiniOS7 版本可从如下地址下载：

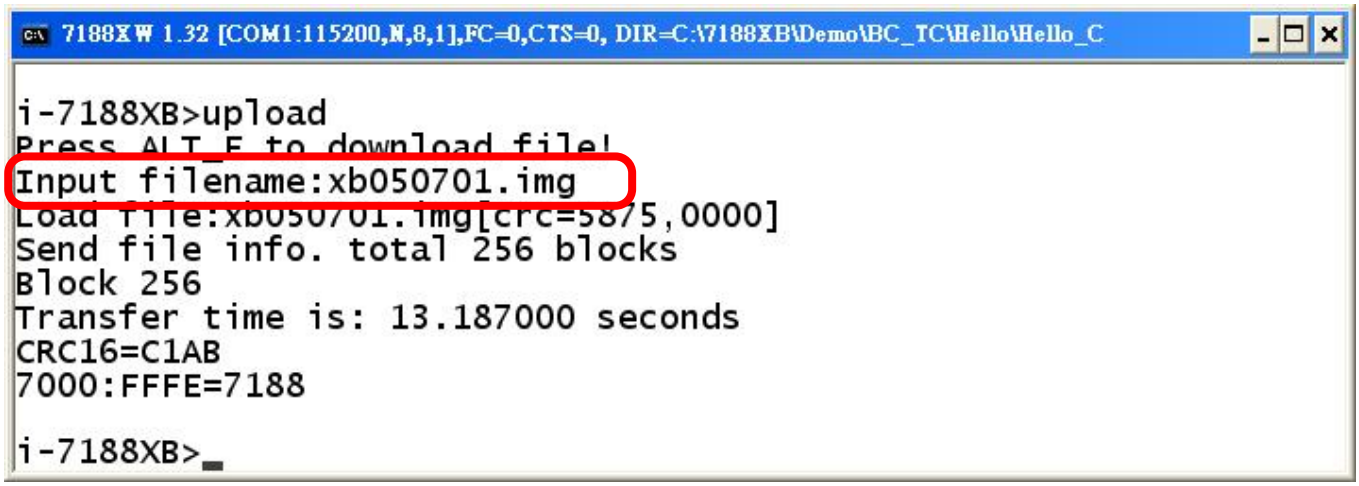
http://ftp.icpdas.com/pub/cd/8000cd/napdos/7188xabc/7188xb/os_image/

步骤 3: 从主机 PC 打开镜像文件存储的文件夹，执行该文件夹中的 7188xw.exe 程序来连接主机 PC 和 I-7188XB(D) 控制器。

步骤 4: 使用 “**UPLOAD**”命令，在屏幕中出现“Press ALT_E to download file!”信息后点击**<ALT + E>** 。

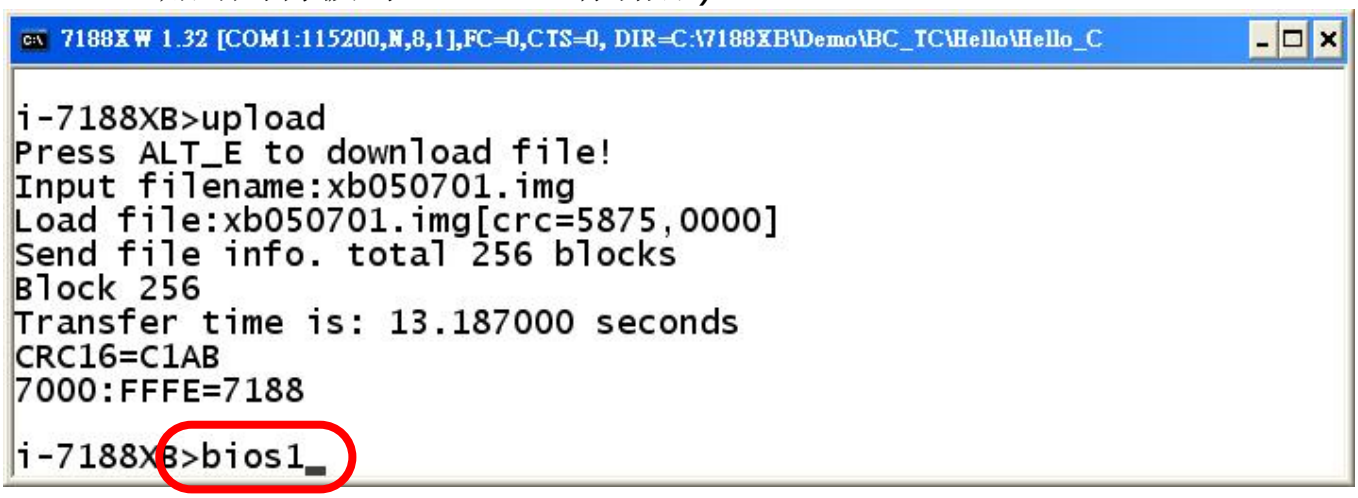


步骤 5: 输入镜像文件夹文件名 (例如: xb050701.img)并点击 **<ENTER>**。

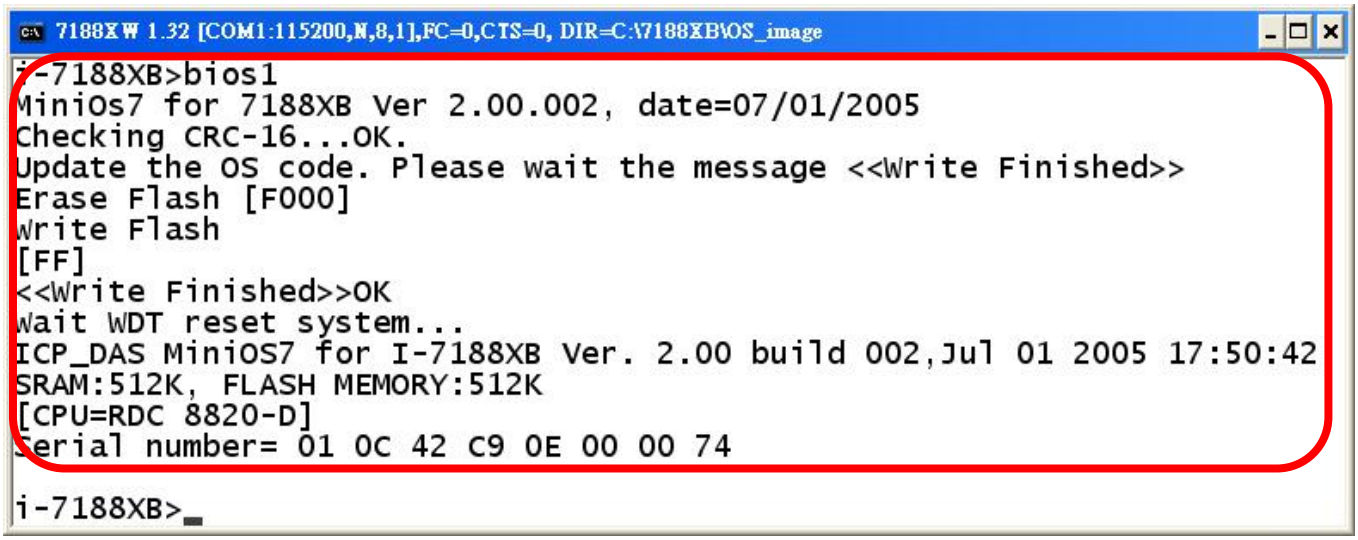


步骤 6: 等待升级完成 (镜像文件将存储在 SRAM)

步骤 7: 在 I-7188XB(D)的命令行中输入 “**bios1**”命令 (OS 将检查存储在 SRAM 中的镜像文件，并且显示版本信息，如果镜像文件是正确的则将被写入 Flash 存储器)



步骤 8: 升级 MiniOS7 将占用 10 秒时间。在升级完成后系统将自动重新启动，如果没有重新启动则需手动重启。



步骤 9: 输入“**ver**” 确认 MiniOS7 的版本

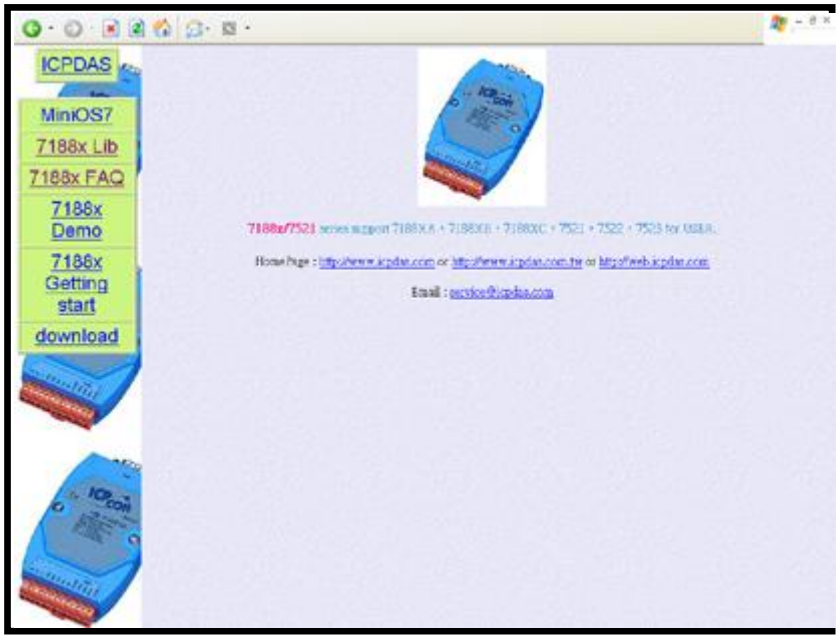
附录 C: 对照表

7188 和 7188X 系列的特征对照表:

	I-7188XA(D)	I-7188XB(D)	I-7188XC(D)	I-7188(D)
CPU 时钟	80188, 40MHz	80188, 40MHz	80188, 20MHz	80188, 40MHz
SRAM	512K	256K(I-7188XB) 512K(I-7188XB/512)	128K	256K
Flash Memory	512K	512K	256K (512K for ODM)	256K/512K
COM1	RS-232 带 modem 模式或 RS-485 含 self-tuner	RS-232 或 RS-485 含 self-tuner	RS-232 或 RS-485 含 self-tuner	RS-232 带 modem 模式或 RS-485
COM2	S-485 含 self-tuner， 3000V 隔离	S-485 含 self-tuner	S-485 含 self-tuner	RS-485
COM3	RS-232 (TxD, RxD)	无	无	RS-232 (TxD, RxD)
COM4	RS-232 (TxD, RxD)	无	无	RS-232 (TxD, RxD)
用户自定义引脚	0	14	3	0
Modem 控制	COM1	无	无	COM1
RTC	Yes	Yes	No	Yes
64 唯一硬件序列号	Yes	Yes	无	无
EEPROM	2K bytes	2K bytes	2K bytes	2K bytes
D/I (3.5V~30V)	2 通道	1 通道	2 通道	0
D/O (100mA, 30V)	2 通道	1 通道	3 通道	0
I/O 扩展总线	Yes	Yes	Yes	无
支持 ASIC Key	Yes	Yes	Yes	无
操作系统	MiniOS7	MiniOS7	MiniOS7	MiniOS7
编程语言	TC/MSC/BC	TC/MSC/BC	TC/MSC/BC	TC/MSC/BC
程序下载口	COM4	COM1	COM1	COM4

附录 D: 库函数列表

下面的表中列举了最常用的函数，更详细信息请参考
CD:\Napdos\MiniOS7\document\Lib_Manual_For_7188XABC\index_e.htm 或
http://ftp.icpdas.com/pub/cd/8000cd/napdos/minios7/document/lib_manual_for_7188xabc/index_e.htm



类型 1:标准 IO

函数	描述
Kbhit	检查是否有键盘输入的数据在 COM1 端口的缓存
Getch	等待到接收从键盘输入的 1 个单独字符
Ungetch	返回 1 个单独的字符到 COM1 端口的输入缓存
Putch	发送 1 个单独的字符到 COM1 端口
Puts	发送字符串到 COM1 端口
Scanf	类似 C 语言中的 Scanf 检索格式化的数据(无法用在 MSC/VC++，只能 TC/BC)
Print	类似 C 语言中的 printf 打印格式化的数据
ReadInitPin	读取 INIT*脚状态
LineInput	从 StdInput 输入单独的线
…More…	另外还有很多关于标准 IO 的函数，详细请参 7188xb.h 和以下位置的文件 CD:\Napdos\MiniOS7\document\Lib_Manual_For_7188XABC\index.htm

注意: Print 和 **printCom** 不可同时在一个程序中使用

- **Kbhit()**

  功能: 检测输入缓存是否有键盘输入的数据

  语法: **int Kbhit(void);**

  头文件: #include ”7188xb.h”

  描述: 检测输入缓存是否有键盘输入的数据

  返回值: 0: 无数据

  Other: 缓存中有数据，并且返回缓存中的数据。如果下个数据是 “\0”函数将返回 -1 (0xFFFF)。

  例程:

```
#include “7188xb.h”
void main()
{
   int quit=0, data;
   InitLib();
   Puts("\n\rPress any key to show ASCII ('Q' to quit):\n\r");
   while(!quit){
      if(Kbhit()){
         data=Getch();
         if(data=='Q') quit=1;
         Putch(data);
         Print(" ASCII is: %d\n\r", data);
         Puts("\n\rPress any key to show ASCII ('Q' to quit):\n\r");
      }
   }
}
```

- **Getch()**

  功能: 等待到接收从键盘输入的一个单字符

  语法: **int Getch(void);**

  头文件: #include ”7188xb.h”

  描述: 从输入缓存读取一个单独的字符，如果缓存没有输入数据，函数将一直等待从缓存中接收到数据为止

  返回值: 0 ~ 255

  例程: 请参考“Kbhit()”

- **Ungetch()**

  功能: 在输入缓存放入单字符。
  语法: **int Ungetch(int data);**
  头文件: #include ”7188xb.h”
  描述: 调用 Ungetch()时，如果没有数据在输入缓存，函数 Getch() 将在下周期继续调用，直到收到返回数据为止。
  数据： 0 ~ 255，如果数据 > 255，仅发送低字节。
  返回值: 如果成功返回 NoError，否则(例如缓存已满)返回 1。
  例程: 请参考“Kbhit()”

- **Putch()**

  功能: 在屏幕中显示单字符。
  语法: **void Putch(int data);**
  头文件: #include ”7188xb.h”
  描述: 数据: 0~255，如果数据 > 255，仅发送低字节。
  例程: 请参考“Kbhit()”

- **Puts()**

  功能: 在屏幕中显示字符串。
  语法: **void Puts(char *str);**
  头文件: #include ”7188xb.h”
  描述: Puts 将调用 Putch()来发送字符串
  str: 字符串的指针将被发送
  例程: 请参考“Kbhit()”

- **Scanf()**

  功能: 类似于 C 语言的 scanf()函数从输入区扫描一个字符 (此函数无法用于 MSC /VC++)
  语法: **int Scanf(char *fmt, ...);**
  头文件: #include ”7188xb.h”
  描述: 在完成成功的扫描、转换和存储后返回输入区的数值。返回值不包括没有存储的区域。
  返回值: 0: 没有存储文件
  EOF: 尝试读到字符串的结尾

例程: 参考 CD:\Napdos\7188XABC\7188XB\Demo\MSC\COM_Ports\C_Style_IO\

- **Print()**

功能: 在屏幕上打印一个格式符，用法类似于 C 语言的 printf() 函数。

语法: **int Print(char *fmt,...);**

头文件: #include ”7188xb.h”

描述: 该函数可用于取代函数 printf()，而函数 Print() 和 printf()两者不同之处在于，函数 Print() 并不转换字符 “\n” 为 “\n” + “\r”。字符“\n”仅仅只发送代码 0x0A，而并非 0x0A + 0x0D，因此“\n\r”必须在执行 “换行并置首” 操作时，全部完成。显示信息将从 COM4(默认参数： 115200, N, 8, 1)发送。

输入参数: 请参考 C 语言的标准函数 printf() 。

返回值: 发送的字符数

例程: 请参考 “Kbhit()”

类型 2: COM 端口

函数	描述
InstallCom	安装 COM 端口驱动。COM 端口号未分派
InstallCom1	安装 COM1 端口驱动
InstallCom2	安装 COM2 端口驱动
RestallCom	卸载 COM 端口驱动。COM 端口号未分派。
RestallCom1	卸载 COM1 端口驱动。COM 端口号分派为 1。 RestallCom2 与该函数相同
IsCom	检查 COM 端口的缓存是否有数据。COM 端口号未分派
IsCom1	检查 COM1 端口的缓存是否有数据
IsCom2…	检查 COM2 端口的缓存是否有数据
ClearCom	清除所有 COM 端口的缓存上的数据。 COM 端口号未分派
ClearCom1	清除所有 COM1 端口的缓存上的数据
ClearCom2	清除所有 COM2 端口的缓存上的数据
ReadCom	读取 COM 端口的缓存上的数据。 COM 端口号未分派
ReadCom1	读取 COM1 端口的缓存上的数据
ReadCom2	读取 COM2 端口的缓存上的数据
ToCom	向 COM 端口发送数据。COM 端口号未分派
ToCom1	向 COM1 端口发送数据
ToCom2	向 COM2 端口发送数据
printCom	打印 COM 端口的缓存上存储的数据。COM 端口号未分派
printCom1	打印 COM1 端口的缓存上存储的数据
printCom2	打印 COM2 端口的缓存上存储的数据
…More…	还提供很多为 COM 端口应用的函数。请参考 7188xb.h 文件和 CD:\Napdos\MiniOS7\document\Lib_Manual_For_7188XABC\index.htm

注意: Print 和 **printCom** 不能在程序中同时使用。

- **InstallCom()**

  功能: 安装 COM 端口驱动。

  语法: **int InstallCom(int port, unsigned long baud, int data, int parity, int stop);**

头文件: #include ”7188xb.h”

描述: 安装 COM 端口驱动。COM 端口号未分派，但可以通过“port”参数进行修改。

port: 分派端口号

baud: 波特率， I-7188XB(D) 默认是 115200

例程:

```
#include “7188xb.h”
void main()
{
  int quit=0, data, i, port=1;   /*port=1, uses COM1*/
  InitLib();
  InstallCom(port,115200,8,0,1);   /*installs the COM port driver*/
  for(i=0; i<10; i++)
    printCom(port,"Test %d\n\r",i);   /*prints data to the COM Port*/
  while(!quit) {
    if(IsCom(port)) {   /*checks if any data is in the COM Port buffer*/
      data=ReadCom(port);   /*reads data from the COM Port buffer*/
      ToCom(port,data);   /*sends data to the COM Port buffer*/
      ClearCom(port);   /*clears all the data in the COM Port buffer*/
      if(data=='Q') quit=1;   /*if 'Q' is received, exit the program*/
    }
  }
  RestoreCom(port);   /*uninstalls the driver for COM Port */
}
```

- **InstallCom1()**

功能: 安装 COM1 端口驱动。

语法: **int InstallCom1(unsigned long baud, int data, int parity, int stop);**

头文件: #include ”7188xb.h”

描述: 安装 COM 端口驱动。

波特率， I-7188XB(D) 默认是 115200

例程:

```
#include “7188xb.h”
void main()
{
  int quit=0,data;
  InitLib();
  InstallCom1(115200,8,0,1);   /*install the driver for COM1*/
```

```
    while(!quit) {
        if(IsCom1()) {    /*checks if any data is in the COM1 buffer*/
            data=ReadCom1();    /*reads data from COM1*/
            ToCom1(data);    /*sends data to COM1*/
            if(data=='q') quit=1;    /*if 'q' is received, exit the program*/
        }
    }
    RestoreCom1();    /*uninstalls the driver for COM1*/
}
```

- **RestoreCom()**

  功能: 卸载 COM 端口驱动。COM 端口号未分派。

  语法: **int RestoreCom(int port);**

  头文件: #include "7188xb.h"

  描述: 卸载 COM 端口驱动。COM 端口号未分派，但可以通过"port"参数进行修改。
  port: 分派 COM 端口号

  例程: 请参考 "InstallCom()"

- **RestoreCom1()**

  功能: 卸载 COM1 端口驱动.

  语法: **int RestoreCom1(void);**

  头文件: #include "7188xb.h"

  描述: 卸载 COM1 端口驱动

  例程: 请参考 "InstallCom()"

- **IsCom()**

  功能: 检查 COM 端口的缓存是否有数据。COM 端口号未分派

  语法: **int IsCom(int port);**

  头文件: #include "7188xb.h"

  描述: 检查 COM 端口的缓存是否有数据。COM 端口号未分派，但可以通过"port"参数进行修改。
  port: 分派端口号

  例程: 请参考 "InstallCom()"

- **IsCom1()**

  功能: 检查 COM1 端口的缓存是否有数据。
  语法: **int IsCom1(void);**
  头文件: #include ”7188xb.h”
  描述: 检查 COM1 端口的缓存是否有数据。
  例程: 请参考 “InstallCom()”

- **ReadCom()**

  功能: 从 COM 端口的换粗读取数据，COM 端口号未分派。
  语法: **int ReadCom(int port);**
  头文件: #include ”7188xb.h”
  描述: 从 COM 端口的换粗读取数据，COM 端口号未分派，但可以通过 “port”参数修改
  port: 分配的 COM 端口号。
  例程: 请参考 “InstallCom()”

- **ReadCom1()**

  功能: 从 COM1 端口的缓存读取数据。
  语法: **int ReadCom1(void);**
  头文件: #include ”7188xb.h”
  描述: 从 COM 端口的缓存读取数据
  例程: 请参考 “InstallCom()”

- **ClearCom()**

  功能: 清除所有 COM 端口的缓存上的数据。COM 端口号未分派
  语法: **int ClearCom(int port);**
  头文件: #include ”7188xb.h”
  描述: 清除所有 COM 端口的缓存上的数据。 COM 端口号未分派，但是可以通过“port” 参数修改
  port: 分配 COM 端口号。
  例程: 请参考 “InstallCom()”

- **ClearCom1()**

  功能: 清除 COM1 端口的缓存上的所有数据。
  语法: **int ClearCom1(void);**
  头文件: #include ”7188xb.h”
  描述: 清除 COM1 端口的缓存上的所有数据。
  例程: 请参考 “InstallCom()”

- **ToCom()**

  功能: 向 COM 端口发送数据。COM 端口号未分派。
  语法: **int ToCom(int port);**
  头文件: #include ”7188xb.h”
  描述: 向 COM 端口发送数据。COM 端口号未分派，但是可以通过“port” 参数修改
  port: 分配 COM 端口号。
  例程: 请参考 “InstallCom()”

- **ToCom1()**

  功能: 向 COM1 端口发送数据。
  语法: **int ToCom1(void);**
  头文件: #include ”7188xb.h”
  描述: 向 COM1 端口发送数据。
  例程: 请参考 “InstallCom()”

- **printCom()**

  功能: 打印 COM 端口的缓存上存储的数据。COM 端口号未分派
  语法: **int printCom(int port,char *fmt,...);**
  头文件: #include ”7188xb.h”
  描述: 打印 COM 端口的缓存上存储的数据。COM 端口号未分派，但是可以通过“port” 参数修改.。可生产格式化的数据输出，类似于 C 语言的 printf()函数。
  例程: 请参考 “InstallCom()”

- **printCom1()**

  功能: 打印 COM1 端口的缓存上存储的数据

  语法: **int printCom_1(char *fmt,...);**

  头文件: #include ”7188xb.h”

  描述: 打印 COM 端口的缓存上存储的数据，可生产格式化的数据输出，类似于 C 语言的 printf()函数。

  例程: 该函数类似于 printCom()。请参考 “InstallCom()”。

类型 3: EEPROM

函数	描述
EE_WriteEnable	设置 EEPROM 为可写入模式
EE_MultiWrite	向 EEPROM 写入数据
EE_WriteProtect	设置 EEPROM 为写保护模式
EE_MultiRead	读取 EEPROM 中的数据
…更多…	更多有关 EEPROM 函数请参考头文件 7188xb.h 及随机赠送 CD 中的用户手册，地址：CD:\Napdos\MiniOS7\document\Lib_Manual_For_7188XABC\index.htm

- **EE_WriteEnable ()**

  功能: 设置 EEPROM 为可写入模式。

  语法: **void EE_WriteEnable (void);**

  头文件: #include ”7188xb.h”

  描述: 设置 EEPROM 为可写入模式。EEPROM 默认为写保护模式。在向 EEPROM 中写入数据前必需调用 EE_WriteEnable()函数。

  例程:

```
#include <7188xb.h>
void main()
{
   Int data=55, data2;
   InitLib();
   EE_WriteEnable ();
   EE_MultiWrite(1,10,1,&data);
   EE_WriteProtect();
   EE_MultiRead(1,10,1,&data2);
   Print("data=%d, Data2=%d", data,data2);
}
```

- **EE_MultiWrite ()**

  功能 向 EEPROM 写入数据

  语法: **int EE_MultiWrite(int Block,unsigned Addr,int no,char *Data);**

  头文件: #include ”7188xb.h”

  描述: 向 EEPROM 写入数据。

Block: 0 到 7 (Block 共有 8 个).
Addr: 0 到 255 (每个 block 可包含 256 bytes).
no: 1 到 16
Data:存储在缓存的起始地址的数据

返回值: 操作成功，返回 NoError.
操作失败，返回-1，表示 EEPROM 忙、 Block 损坏或者地址错误。

例程: 请参考 “EE_WriteEnable()”

- **EE_WriteProtect ()**

功能: 设置 EEPROM 为写保护模式。

语法: **void EE_WriteProtect(void);**

头文件: #include ”7188xb.h”

描述: 设置 EEPROM 为写保护模式，EEPROM 默认为写保护模式。在向 EEPROM 中写入数据前必需调用 EE_WriteEnable() 函数。在写完数据后，推荐使用 EE_WriteProtect () 将 EEPROM 恢复为写保护模式。

Example: 请参考 “EE_WriteEnable()”

- **EE_MultiRead ()**

功能: 读取 EEPROM 中的数据。

语法: **int EE_MultiRead(int StartBlock,unsigned StartAddr,int no,char *databuf);**

头文件: #include ”7188xb.h”

描述: 从 EEPROM 读取多位字节的数据。
StartBlock: 0 到 7 (总共 8 个 block).
StartAddr: 0 到 255 (每个 block 可包含 256 bytes)
no: 1 到 2048
databuf: 存储数据的地址

返回值: 操作成功，返回 NoError。操作失败，返回-1，表示 EEPROM 忙、 Block 损坏或者地址错误。

例程: 请参考“EE_WriteEnable()”

类型 4: NVRAM 和 RTC

函数	描述
ReadNVRAM	从 NVRAM 读取数据
WriteNVRAM	在 NVRAM 中写入数据
GetTime	自 RTC 中取得系统时间
SetTime	为 RTC 设置系统时间
GetDate	自 RTC 取得系统日期
SetDate	为 RTC 设置系统日期
GetWeekDay	自 RTC 取得星期信息
…更多…	其它关于 NVRAM 和 RTC 的函数请参考 7188xb.h 头文件或光盘上用户手册。地址如下： CD:\Napdos\minios7\document\lib_manual_for_7188xabc\index.htm

- **ReadNVRAM()**

  功能: 从 NVRAM 读取数据。

  语法: **int ReadNVRAM(int addr);**

  头文件: #include ”7188xb.h”

  描述: 从 NVRAM 读取数据。
  addr: 0 到 30, 共 31 个字节

  返回值: 成功则返回在指定地址存储的数据 (0-255) ，
  失败则返回 AddrError (-9)。

  例程:

```
#include “7188xb.h”
void main()
{
   int data=55, data2;
   InitLib();
   WriteNVRAM(0,data);
   data2=ReadNVRAM(0);    /* now data2=data=55 */
   Print("data=%d, data2=%d",data,data2);
}
```

- **WriteNVRAM()**

  功能: 在 NVRAM 中写入数据。

  语法: **int WriteNVRAM(int addr, int data);**

  头文件: #include ”7188xb.h”

描述: 在 NVRAM 中写入一个字节的数据。
addr: 0-30.
data: 1 个字节的数据(0-255).
如果数据>255，仅有低位的数据会被写入 NVRAM。
返回值: 成功则返回 NoError，
失败则返回 AddrError (-9)。
例程: 请参考“ReadNVRAM()”

- **GetTime()**

功能: 自 RTC 中取得系统时间
语法: **void GetTime(int *hour, int *minute, int *sec);**
头文件: #include ”7188xb.h”
描述: hour: 用于存储小时数据的地址 (0-23) 。
minute: 用于存储分钟数据的地址(0-59)。
sec: 用于存储秒数据的地址 (0-59) 。
例程:

```
#include “7188xb.h”
void main()
{
  int year, month, day, hour, min, sec, wday;
  InitLib();
  SetDate(2006,1,12);   /*sets the system date for the RTC*/
  SetTime(15,35,50);   /*sets the system time for the RTC*/
  SetWeekDay(4);   /*sets the system day of the week for the RTC*/
  GetDate(&year,&month,&day);   /*reads the system date from the RTC*/
  GetTime(&hour,&min,&sec);   /*reads the system time from the RTC*/
  wday=GetWeekDay();
  Print("Date=%02d/%02d/%04d(%d) Time=%02d:%02d:%02d\n\r",
  month,day,year,wday,hour,min,sec);
}
```

- **SetTime()**

功能: 为 RTC 设置系统时间
语法: **int SetTime(int hour,int minute,int sec);**
头文件: #include ”7188xb.h”
描述: hour: 0-23.
minute: 0-59.

sec: 0-59.

返回值: 成功则返回 NoError，失败则返回 TimeError (-19)。

例程: 请参考 “GetTime()”

- **GetDate()**

功能: 自 RTC 取得系统日期

语法: **void GetDate(int *year,int *month,int *day);**

头文件: #include ”7188xb.h”

描述: year: 2000-2080
month: 1-12
day: 1-31

例程: 请参考 “GetTime()”

- **SetDate()**

功能: 为 RTC 设置系统日期

语法: **int SetDate(int year,int month,int day);**

头文件: #include ”7188xb.h”

描述: year: 2000-2080
month: 1-12
day: 1-31

返回值: 成功则返回 NoError，
失败返回 DateError (-18).

例程: 请参考 “GetTime()”

- **GetWeekDay()**

功能: 自 RTC 取得星期信息

语法: **int GetWeekDay(void);**

头文件: #include ”7188xb.h”

描述: 自 RTC 取得星期信息

返回值: 0=>周日
1-6=>周一到周六

例程: 请参考 “GetTime()”

注意: GetWeekDay() 并非验证星期几是否正确，只是从 RTC 读取数据。当使用 MiniOS7 的“date” 命令去设置日期时，MiniOS7 将计算正确的星期信息并设置 RTC 时间。当调用 SetDate()函数时，该函数也会计算正确的星期信息并设置 RTC 时间。然而调用 SetWeekDay() 函数时，该函数会自己计算正确星期数。

类型 5: Flash 存储器

函数	描述
FlashReadId	读取 Flash 存储器的信息
FlashErase	擦除 Flash 存储器的 1 个扇区
FlashWrite	在 Flash 存储器中写入数据的 1 个字节
FlashRead	自 Flash 存储器中读取数据的 1 个字节
…更多…	还有很多 Flash 存储器相关的函数，详细信息请参考 7188xb.h 头文件 和 CD 上 的用户手册，地址是 CD:\Napdos\minios7\document\lib

I-7188XB(D)系列模块具有 512K 字节的 Flash 存储器。MiniOS7 占用其中最后的 64K 字节，其它的空间可用来存储用户程序和数据。

应用开发人员可应用这些函数来向 Flash 存储器写入数据。当写入数据时，数据必须从“1” 到 “0”写入，而不能从“0” 到“1”。所以，在向 Flash 存储器写入数据前必须进行擦除动作，擦除操作将使数据转化成 0xFF，这样所有的数据为将变成“1”，只有这样数据才能被写入。FlashErase() 函数每次擦除 1 个扇区的数据 (64K bytes)。

- **FlashReadId()**

  功能: 读取 Flash 存储器的信息
  语法: **int FlashReadId(void);**
  头文件: #include ”7188xb.h”
  描述: 读取 Flash 存储器的设备代码（高位）和生产代码（低位）。
  返回值: 0xA4C2 (MXIC 29f040), 0xA401 (AMD 29f040)
  例程: 请参考 CD:\Napdos\7188XABC\7188XB\Demo\BC_TC\Memory\

- **FlashErase()**

  功能: 擦除 Flash 存储器的 1 个扇区
  语法: **int FlashErase(unsigned seg);**
  头文件: #include ”7188xb.h”
  描述: 擦除 Flash 存储器的 1 个扇区(64K bytes)。 所有这个扇区

上数据的值将变成 0xFF。
seg: 0x8000, 0x9000, 0xA000, 0xB000, 0xC000, 0xD000 or 0xE000.

返回值: On success, returns NoError (0).
On error, returns TimeOut (-5).

例程: 详细信息请参考 “\Demo\BC_TC\Memory\Flash\FLASH.C”

注意: 0xF000 区段用来存储 MiniOS7 操作系统，如果尝试擦除该区段，FlashErase() 函数将不会起作用。

- **FlashWrite()**

功能: 在 Flash 存储器中写入数据的 1 个字节

语法: **int FlashWrite(unsigned int seg, unsigned int offset, char data);**

头文件: #include ”7188xb.h”

描述: seg: 0x8000, 0x9000, 0xA000, 0xB000, 0xC000, 0xD000 或 0xE000.
offset: 0 to 65535 (0xffff).
data: 0 to 255 (8 位数据).

返回值: 成功则返回 NoError(0)，
失败返回 TimeOut(-5)或 SegmentError(-12).

例程:

```
#include “7188xb.h”
void main()
{
  int data=0xAA55, data2;
  char *dataptr;
  InitLib();
  dataptr=(char *)&data;
  FlashWrite(0xd000,0x1234, *dataptr++);      /*writes data to the Flash
                                               memory*/
  FlashWrite(0xd000,0x1235, *dataptr);
  dataptr=(char *)&data2;      /*reads data from the Flash memory*/
  *dataptr=FlashRead(0xd000, 0x1234);
  *(dataptr+1)=FlashRead(0xd000, 0x1235);
}
```

注意: 当向 Flash 存储器写入数据时，数据位只能由 1 变到 0 ，FlashWrite()函数不会检查 Flash 存储器状态，只会写入数据，如果尝试将数据由 0 变成 1，将会出现 TimeoutError 错误信息。在调用 FlashErase() 函数后数据才可被写入。

- **FlashRead()**

  功能: 自 Flash 存储器中读取数据的 1 个字节。

  语法: **int FlashRead(unsigned int seg, unsigned int offset);**

  头文件: #include ”7188xb.h”

  描述: seg: 0-65535(0xffff).
  offset: 0 to 65535(0xffff).

  返回值: FlashRead() 只会返回地址的值
  seg:区距。地址可以是 SRAM、Flash 存储器或其它地址(都会返回 0xff)。

  例程: 请参考 “FlashWrite()”

类型 6: 计时器和 Watchdog 计时器

函数	描述
TimerOpen	开启计时器功能
TimerClose	关闭计时器功能
TimerResetValue	设置计时器为 0
TimerReadValue	读取主要计时点
DelayMs	在指定的时间段插入延时，时间单位为 ms，使用系统计时点
Delay	在指定的时间段插入延时，时间单位为 ms，使用 CPU 的 Timer 1
Delay_1	在指定的时间段插入延时，时间单位为 0.1 ms，使用 CPU 的 Timer 1
Delay_2	在指定的时间段插入延时，时间单位为 0.01 ms，使用 CPU 的 Timer 1
StopWatchStart	使用 1 个 StopWatch 通道
StopWatchReset	重置 StopWatch 的值为 0
StopWatchStop	停止 StopWatch 通道
StopWatchPause	暂停 StopWatch
StopWatchContinue	重新启动 StopWatch
StopWatchReadValue	读取当前 StopWatch 值
CountDownTimerStart	开始使用 CountDownTimer
CountDownTimerReadValue	读取当前 CountDownTimer 值
InstallUserTimer	装入用户 timer 功能,该函数在 1 ms 内调用
InstallUserTimer1C	在中断 0x1c 装入用户 timer 功能，系统 timer 在 55 ms 内使用该中断点
EnableWDT	使用 Watchdog timer
DisableWDT	停止使用 Watchdog timer
RefreshWDT	刷新 Watchdog timer
…更多…	还有更多计时器和 Watchdog 计时器函数提供，更多详情请参考 7188xb.h 头文件或参考用户手册 CD:\Napdos\minios7\document\lib_manual_for_7188xabc\index.htm

- **TimerOpen()**

  功能:　　打开 function 功能

  语法:　　**int TimerOpen(void);**

  头文件:　　#include ”7188xb.h”

  描述:　　在使用任何 timer 功能前必须调用 TimerOpen()函数。

  返回值:　　成功则返回 NoError，如果 Timer 已经打开则返回 1。

  例程:

```
#include “7188xb.h”
void main()
{
   unsigned long time;
   int quit=0;
   InitLib();
   Print("\n\rPress any key to start the timer");
   Print("\n\rthen Press '0' to Reset the timer, ’1’~’4’ to insert a delay, 'q' to
            quit\n\r");
   Getch();
   TimerOpen();   /*open the timer function*/
   while(!quit){   /*sets the key function*/
      if(Kbhit()){
         switch(Getch()){
            case '0':
               TimerResetValue();   /*reset the timer*/
               break;
            case '1':
               DelayMs(1000);   /*delay unit is ms, uses system timeticks. */
               break;
            case '2':
               Delay(1000);   /*delay unit is ms, uses the Timer 1 feature of the
                                 CPU. */
               break;
            case '3':
               Delay_1(1000);   /*delay unit is 0.1 ms, uses the Timer 1 feature
                                  of the CPU.*/
               break;
            case '4':
               Delay_2(1000);   /*delay unit is 0.01 ms, uses the Timer 1
                                  feature of the CPU .*/
               break;
            case 'q':
```

```
            quit=1;
            break;
        }
      }
      time=TimerReadValue();   /*reads the timer*/
      Print("\r\nTime=%8.3f sec", 0.001*time);
    }
    TimerClose();   /*closes the timer function*/
}
```

- **TimerClose()**

  功能: 停止使用计时器功能。
  语法: **int TimerClose(void);**
  头文件: #include ”7188xb.h”
  描述: 在使用 OpenTimer() 函数前需先调用 TimerClose() 函数。
  返回值: 总返回 NoError。
  例程: 更多详情请参考“TimerOpen()”

- **TimerResetValue()**

  Function: 将计时器复位为 0。
  语法: **void TimerResetValue(void);**
  头文件: #include ”7188xb.h”
  描述: 重置计时点为 0。
  例程: 更多详情请参考“TimerOpen()”

- **TimerReadValue()**

  功能: 读取主要计时点
  语法: **unsigned long TimerReadValue(void);**
  头文件: #include ”7188xb.h”
  描述: 读取主要计时点，时间单位为 ms.当 TimerOpen,TimerReset 函数被调用，计时点将变成 0。
  例程: 更多详情请参考“TimerOpen()”

- **DelayMs()**

  功能: 在指定的时间段插入延时，时间单位为 ms，使用系统计时点

语法: **void DelayMs(unsigned t);**
头文件: #include ”7188xb.h”
描述: 延时时间单位为 ms.
t: 延时时间
例程: 更多详情请参考“TimerOpen()”

- **Delay()**

功能: 在指定的时间段插入延时，时间单位为 ms，使用 CPU 的计时器 Timer 1
语法: **void Delay(unsigned ms);**
头文件: #include ”7188xb.h”
描述: 在指定的时间段插入延时，时间单位为 ms，使用 CPU 的 Timer 1 特征
ms: 延时时间
例程: 更多详情请参考“TimerOpen()”

- **Delay_1()**

功能: 在指定的时间段插入延时，时间单位为 0.1ms，使用 CPU 的计时器 Timer 1
语法: **void Delay_1(unsigned ms);**
头文件: #include ”7188xb.h”
描述: 在指定的时间段插入延时，时间单位为 0.1ms，使用 CPU 的 Timer 1 特征
ms: 延时时间
例程: 更多详情请参考“TimerOpen()”

- **Delay_2()**

功能: 在指定的时间段插入延时，时间单位为 0.01，使用 CPU 的计时器 Timer 1
语法: **void Delay_2(unsigned ms);**
头文件: #include ”7188xb.h”
描述: 在指定的时间段插入延时，时间单位为 0.01，使用 CPU 的 Timer 1 特征
ms: 延时时间

例程: 请参考“TimerOpen()”

- **StopWatchStart()**

  功能: 使用 1 个 StopWatch 通道并重置 StopWatch 值为 0。

  语法: **int StopWatchStart(int channel);**

  头文件: #include ”7188xb.h”

  描述: 系统 timer ISR 将在每 1 ms 为 StopWatch 值增加 1。
  channel: 0-7, a total of 8 channels.

  返回值: 成功则返回 NoError，
  如果超出通道范围返回 ChannelError (-15)。

  例程:

```
#include “7188xb.h”
void main(void)
{
   unsigned long value;
   int quit=0; InitLib();
   Print("\n\rTest the StopWatch ... Press 'q' to quit\n\r ");
   TimerOpen();
   StopWatchStart(0);    /*start using the StopWatchStart function*/
   while(!quit){
      if(Kbhit()){ switch(Getch()){ case 'q': quit=1; break; } }
      StopWatchReadValue(0,&value);
      Print("SWatch=%d \r",value);
      if(value==2000){
         StopWatchPause(0);
         DelayMs(2000);
         StopWatchContinue(0); }
         if(value==4000){
            StopWatchStop(0);
            DelayMs(2000);
            StopWatchReset(0);
            StopWatchStart(0);
         }
      }
      TimerClose();
   }
```

- **StopWatchReset()**

  功能: 将 StopWatch 复位为 0
  语法: **int StopWatchReset(int channel);**
  头文件: #include ”7188xb.h”
  描述: channel: 0-7，共 8 个 channel。
  返回值: 成功则返回 NoError，
  如果超出通道范围返回 ChannelError (-15)。
  例程: 更多详情请参考“StopWatchStart()”

- **StopWatchStop()**

  Function: 停止 StopWatch 通道
  语法: **int StopWatchStop(int channel);**
  头文件: #include ”7188xb.h”
  描述: 系统 timer ISR 将停止增加 StopWatch 的值，
  channel: 0-7，共 8 个 channel。
  返回值: 成功则返回 NoError，
  如果超出通道范围返回 ChannelError (-15)。
  例程: 更多详情请参考“StopWatchStart()”

- **StopWatchPause()**

  功能: 暂停使用 StopWatch
  语法: **int StopWatchPause(int channel);**
  头文件: #include ”7188xb.h”
  描述: 在调用 StopWatchPause()用，必须调用
  StopWatchContinue() 函数恢复时间计时。
  channel:0-7，共 8 个 channel。
  返回值: 成功则返回 NoError，
  如果超出通道范围返回 ChannelError (-15)。
  例程: 更多详情请参考“StopWatchStart()”

- **StopWatchContinue()**

  功能: 重新启动 StopWatch。
  语法: **int StopWatchContinue(int channel);**

头文件: #include ”7188xb.h”
描述: channel:0-7，共 8 个 channel。
返回值: 成功则返回 NoError，
如果超出通道范围返回 ChannelError (-15)。
例程: 更多详情请参考“StopWatchStart()”

- **StopWatchReadValue()**

Function: 读取当前 StopWatch 值
语法: **int StopWatchReadValue(int channel,unsigned long *value);**
头文件: #include ”7188xb.h”
描述: 这个值表示自 StopWatchStart()或 StopWatchReset()函数被调用后占用的时间。
channel: 0-7，共 8 个 channel。
返回值: 成功则返回 NoError，
如果超出通道范围返回 ChannelError (-15)。
例程: 更多详情请参考“StopWatchStart()”

- **CountDownTimerStart()**

功能: 开始使用 CountDownTimer。
语法: **int CountDownTimerStart(int channel,unsigned long count);**
头文件: #include ”7188xb.h”
描述: channel: 0-7，共 8 个 channel。
count: 已经计数的数值
返回值: 成功则返回 NoError，
如果超出通道范围返回 ChannelError (-15)。

例程:

```
#include “7188xb.h”
void main(void)
{
  unsigned long value;
  int quit=0;
  InitLib();
  Print("\n\rTest the CountDownTimer...");
```

```
    Print("\n\rPress 'q' to quit\n\r");
    TimerOpen();
    CountDownTimerStart(0,1000);    /*use the CountDownTimer*/
    while(!quit){
      if(Kbhit()&&(Getch()=='q')) quit=1;
      CountDownTimerReadValue(0,&value);    /*reads the
                                              CountDownTimer*/
      Print("Test CountDown=%d\r",value);
      if(value==0)
        CountDownTimerStart(0,1000);    /*restarts the CountDownTimer*/
    }
    TimerClose();
}
```

- **CountDownTimerReadValue()**

  功能: 读取当前 CountDownTimer(count)值。

  语法: **int CountDownTimerReadValue(int channel,unsigned long *value);**

  头文件: #include ”7188xb.h”

  描述: 如果值为 0 代表时间已经终止。
  channel: 0-7，共 8 个 channel。
  value: 值存储位置的指针

  返回值: 成功则返回 NoError，
  如果超出通道范围返回 ChannelError (-15)。

  例程: 请参考 “CountDownTimerStart ()”

- **InstallUserTimer()**

  功能: 启动用户 timer 功能，该函数在每 1ms 内调用

  语法: **void InstallUserTimer(void (*fun)(void));**

  头文件: #include ”7188xb.h”

  描述: fun:用户函数的指针
  该函数无法使用输入变量，并且不会提供返回值

  例程:

```
#include “7188xb.h”
int Data[3]={0,0,0};
void MyTimerFun(void)   /*custom timer function*/
{
   static int count[3]={0,0,0};
   int i;
   for(i=0;i<3;i++){
      Print("count[%d]=%d\r",i,count[i]);
      count[i]++;
   }
   if(count[0]>=200){   /*LCD lamp1 blinks each 200 units*/
      count[0]=0;
      if (Data[0]==0) Data[0]=1;
      else Data[0]=0;
      lamp(1,1,Data[0]);
   }
   if(count[1]>=500){   /*LCD lamp2 blinks each 500 units*/
      count[1]=0;
      if (Data[1]==0) Data[1]=1;
      else Data[1]=0;
      lamp(2,1,Data[1]);
   }
   if(count[2]>=1000){   /*LCD lamp3 blinks each 1000 units*/
      count[2]=0;
      if (Data[2]==0) Data[2]=1;
      else Data[2]=0;
      lamp(3,1,Data[2]);
   }
}
void main(void)
{
   int quit=0;
   Print("\n\rTest the LCD lamp blink using user timer ");
   Print("\n\rPress 'q' to quit\n\r");
   InitLib(); InitLCD();   /*initial Lib & LCD*/
   ClrScrn();   /*clear the LCD screen*/
   TimerOpen();   /*open timer function*/
   InstallUserTimer(MyTimerFun); /*install and call user timer */
   while(!quit){
      if(Kbhit() && Getch()=='q') quit=1;
   }
   TimerClose();}
```

- **InstallUserTimer1C()**

  功能: 在中断 0x1c 装入用户计时器功能，系统计时器在 55 ms 内使用该中断点

  语法: **void InstallUserTimer1C(void (*fun)(void));**

  头文件: #include ”7188xb.h”

  描述: fun: 用户函数的指针。该函数无法使用输入变量，并且不会提供返回值

  例程: 请参考“InstallUserTimer()”

- **EnableWDT()**

  功能: 启用 WatchDog timer.

  语法: **void EnableWDT(void);**

  头文件: #include ”7188xb.h”

  描述: WatchDog Timer (WDT) 一直处于激活状态，并由系统 Timer ISR 不断的刷新。当用户程序调用 EnableWDT() 函数时，系统 timer ISR 将停止刷新 WDT，直到在程序中调用 RefreshWDT()函数，否则系统将被 WDT 重新启动。. MiniOS7 2.0 下 WDT 的超时时间为 0.8 秒。

  例程:

```
#include ”7188xb.h”
void main(void)
{
   int quit=0,k;
   InitLib();
   if(IsResetByWatchDogTimer())   /*test whether the system has been
                                    reset by the WDT*/
      Print("reset by WatchDog timer\n\r");
   EnableWDT();   /*after callng EnableWDT, Refresh WDT must be called
                    within 0.8s*/
   while(!quit){
      if(Kbhit()) {
         k=Getch();
         if(k=='q') {
            Print("quit the program\r\n");
            quit=1;   /*quit the program*/
         }
```

```
        else {
          Print("more than 0.8s has elapsed reset the system\r\n");
          Delay(1000);   /*There has been a delay for more than 0.8s. Reset
              the system*/
        }
      }
      RefreshWDT();   /*Refresh WDT must be called within 0.8s*/
      Print("call Refresh WDT\n\r");
    }
    DisableWDT(); /*Disable the WDT. The system will refresh the WDT*/
    Print("Call DisableWDT\n\r");
}
```

- **DisableWDT()**

  功能: 停止 WatchDog 计时器
  语法: **void DisableWDT(void);**
  头文件: #include ”7188xb.h”
  描述: 更多详情请参考 EnableSDT()的描述。
  例程: 更多详情请参考 “EnableWDT()”

- **RefreshWDT()**

  功能: 刷新 WatchDog 计时器
  语法: **void RefreshWDT(void);**
  头文件: #include ”7188xb.h”
  描述: 更多详情请参考 EnableSDT()的描述。
  例程: 更多详情请参考“EnableWDT()”

- **IsResetByWatchDogTime()**

  功能: 检查系统是否被 WatchDog 计时器重启。
  语法: **int IsResetByWatchDogTime(void);**
  头文件: #include ”7188xb.h”
  描述: “是”则返回 0
  例程: 更多详情请参考 “EnableWDT()”

类型 7:文件操作

函数	描述
GetFileNo	得到存储在 Flash 存储器上的文件数
GetFileName	使用文件索引得到文件名
GetFilePositionByNo	使用文件号得到文件位置
GetFileInfoByNo	使用文件号得到文件信息
GetFileInfoByName	使用文件名得到文件信息
…更多…	还有更多 timer 和 Watchdog Timer 函数提供，请参 7188xb.h 头文件 或参考用户手册 CD:\Napdos\minios7\document\libndex.htm

注意: MiniOS7 操作系统支持用户程序读取按文件，但并不支持以文件方式写入。

- **GetFileNo()**

  功能: 得到存储在 Flash 存储器上的文件数。
  语法: **int GetFileNo(void);**
  头文件: #include ”7188xb.h”
  描述: 返回文件数
  例程: 更多详情请参考“GetFilePositionByNo()”

- **GetFileName()**

  功能: 使用文件索引得到文件名。
  语法: **int GetFileName(int no,char *fname);**
  头文件: #include ”7188xb.h”
  描述: no: 文件索引 (第一个文件的索引为 0)
  fname: 存储文件名的缓存
  返回值: 成功则返回 NoError，并将文件名存入 fname。
  失败则返回-1，不在 fname 中存入任何数据。
  例程: 更多详情请参考“GetFilePositionByNo()”

- **GetFilePositionByNo()**

  功能: 使用文件号得到文件位置
  语法: **char far *GetFilePositionByNo(int no);**

头文件: #include ”7188xb.h”
描述: 地址可用来得到文件数据。
no: 文件索引 (第一个文件的索引为 0)
返回值: 成功则返回文件的起始地址，失败则返回 NULL。

注意: 如果文件大于 64K-16, 超出部分必须使用一个巨型指针来得到文件数据。

例程:

```
#include ”7188xb.h”
static FILE_DATA far *fdata;   /*file_data structure, please see the file.c
                                 for details*/
char far *fp_no;
void main()
{
   int fileno,i;
   char fname[13];
   InitLib();   /*Initialize the Library*/
   fileno=GetFileNo();   /*get file number*/
   Print("Total file number=%d\n\r",fileno);
   fname[12]=0;
   for(i=0;i<fileno;i++){
      fdata=GetFileInfoByNo(i);   /*get file information using the file
                                    number*/
      if(fdata) {
         GetFileName(i,fname);   /*get file name*/
         Print("[%02d]:%-12s start at %Fp "
               "%02d/%02d/%04d %02d:%02d:%02d size=%lu\n\r", i,fname,
         fdata->addr,fdata->month,fdata->day,(fdata->year)+1980,
         fdata->hour,fdata->minute,fdata->sec*2,fdata->size);
      }
   }
   for(i=0;i<fileno;i++){
      fp_no=(char far *)GetFilePositionByNo(i);   /*get file position*/
      if(fp_no){
         GetFileName(i,fname);
         Print("file %d [%-12s] position: [ %Fp ]\r\n",i,fname,fp_no);
      }
   }
}
```

- **GetFileInfoByNo()**

  功能: 使用文件数索引得到文件信息
  语法: **FILE_DATA far *GetFileInfoByNo(int no);**
  头文件: #include ”7188xb.h”
  描述: no: 文件索引(第一个文件的索引为 0)
  返回值: 成功则返回文件的起始地址，失败则返回 NULL。
  例程: 更多详情请参考“GetFilePositionByNo()”

- **GetFileInfoByName()**

  功能: 使用文件名得到文件信息
  语法: **char far *GetFileInfoByName(char *fname);**
  头文件: #include ”7188xb.h”
  描述: fname: 文件名
  返回值: 成功则返回文件的起始信息，失败则返回 NULL。
  例程: 更多详情请参考“GetFilePositionByNo()”

类型 8: 连接 I-7000/I-87K 系列模块

函数	描述
SendCmdTo7000	向 I-7000/I-87K 系列模块发送数据
ReceiveResponseFrom7000_ms	接收 I-7000/I-87K 系列模块的回应
ascii_to_hex	将 ASCII 码换算成 16 进制值
hex_to_ascii	将 16 进制值换算成 ASCII 码
…更多…	还提供很多为 COM 端口应用的函数。 更多详情请参考 7188xb.h 文件和 CD:\Napdos\minios7\document\lib

- **SendCmdTo7000()**

  功能: 向 I-7000/I-87K 系列模块发送数据

  语法: **int SendCmdTo7000(int iPort, unsigned char *cCmd, int iChksum);**

  头文件: #include ”7188xb.h”

  描述: 如果启用数据校验，函数将在命令最后加 2 个直接的校验位。

  iPort: 0/1/2/3/4 代表 COM0/1/2/3/4.

  cCmd: 将被发送的命令(如果没有则在 cCmd 最后加“\r”就像 SendCmdTo7000()函数在最后加校验位一样 。

  iChecksum: 1 代表启用数据校验, 0 代表不校验。

  返回值: 成功则返回 NoError，失败则返回错误代码。请参考 I-7000 产品用户手册。

  例程:

```
#include “7188xb.h”
void main()
{
   int port=2,quit=0,x;
   char k;
   InitLib();
   InstallCom(port,115200L,8,0,1);    /*install a COM Port for the I-7065D*/
   ClearCom(port);
   SendCmdTo7000(port, "@0100", 0);    /*send a command to DO to off*/
   if(ReceiveResponseFrom7000_ms(port,"@0100",20,0))
      Print("I-7065D is not available\r\n");
   while(!quit){    /*control Do*/
```

```
        Print("\n\r Enter 1~5 to set [Do] on...'9' to quit\n\r");
        k=Getch();
        x=ascii_to_hex(k);    /*convert ASCII code to hex*/
        ClearCom(port);
        switch(x){    /*send a command to set the I-7065D Do1~5 light to on*/
          case 1:    /*for command details, refer to the "I-7000 DIO manual"*/
            SendCmdTo7000(port, "@0101", 0);Print("[%x]=ON",x);break;
          case 2:
            SendCmdTo7000(port, "@0102", 0); Print("[%x]=ON",x);break;
          case 3:
            SendCmdTo7000(port, "@0104", 0); Print("[%x]=ON",x);break;
          case 4:
            SendCmdTo7000(port, "@0108", 0); Print("[%x]=ON",x);break;
          case 5:
            SendCmdTo7000(port, "@0110", 0); Print("[%x]=ON",x);break;
          case 9:
            quit=1; Print("*quit*");break;
        }   /*end of switch*/
      }   /*end of while loop*/
    }
```

- **ReceiveResponseFrom7000_ms()**

  功能: 接收来自 I-7000 模块的返回数据。

  语法: **int ReceiveResponseFrom7000_ms(int iPort, unsigned char *cCmd, long lTimeout, int iChksum);**

  头文件: #include "7188xb.h"

  描述: 在调用 SendCmdTo7000()函数后，即使不需要接受回应数据，也请调 ReceiveResponseFrom7000_ms() 函数。
  iPort: 1 代表 COM1、2 代表 COM2
  cCmd: 从 I-7000 发挥的回应。如果启用数据校验，函数将校验并移动校验码，CR 也将被移动。
  lTimeout: 设置超时，单位 ms。
  iChecksum: 1 代表启用数据校验，0 代表不启用。

  返回值: 成功则返回 NoError，失败则返回错误代码。请参考 I-7000 产品用户手册。

  例程: 更多详情请参考 endCmdTo7000()

- **ascii_to_hex()**

  功能:　　将 ASCII 转换成 16 进制值

  语法:　　**int ascii_to_hex(char ascii);**

  头文件:　　#include ”7188xb.h”

  描述:　　返回 1 个整数代表 16 进制值

  　　ascii: ASCII 代码字符

  例程:　　更多详情请参考“ SendCmdTo7000()”

附录 E: 编译和链接

使用 TC 编译器

使用 TC 2.01 有 2 种编译过程，参考方式如下：

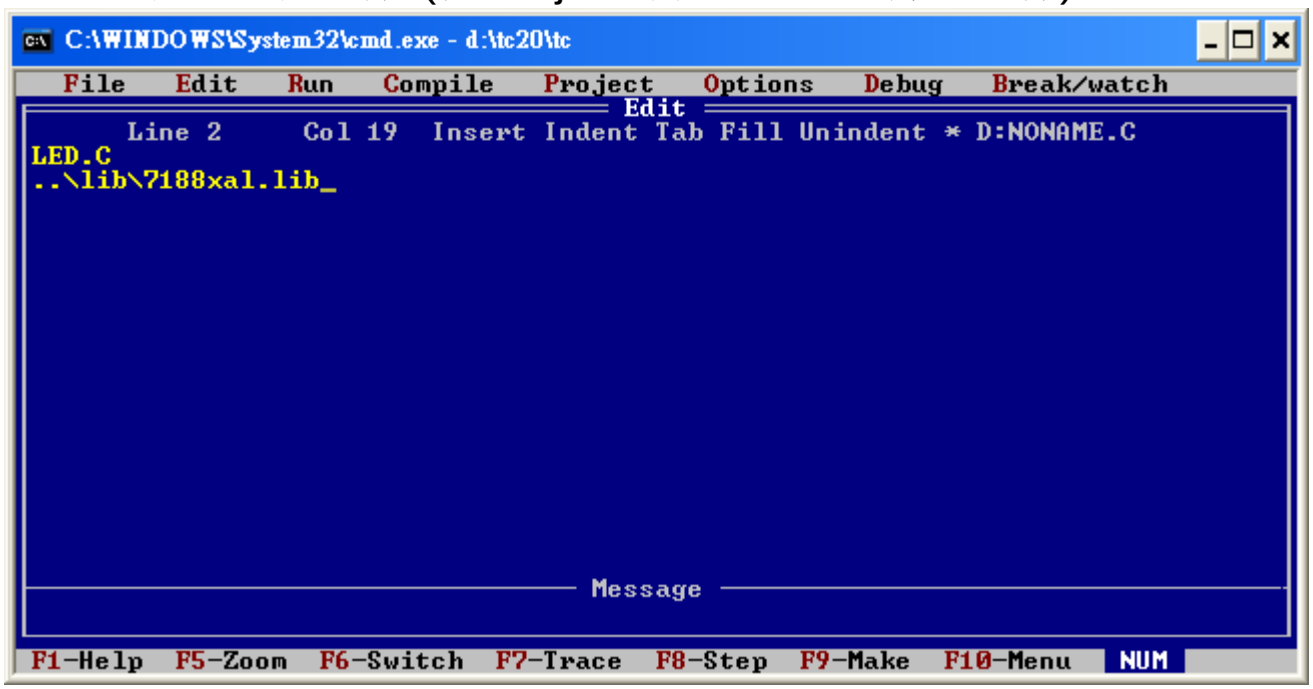
方法 1: 使用命令行 (详情请参考 **CD:\8000\NAPDOS\7188XABC\7188XB\Demo\BC_TC\Hello_C\gotc.bat**)

tcc -Ic:\tc\include -Lc:\tc\lib hello1.c ..\lib\7188xbs.lib

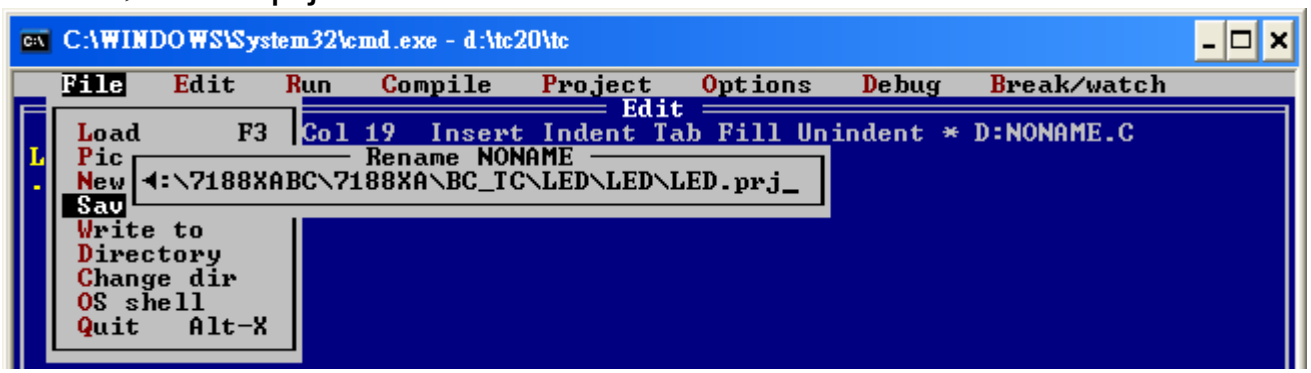
方法 2: 使用 TC 集成编译环境

步骤 1: 执行 TC.EXE 运行 TC 2.01 使用 TC 集成编译环境。

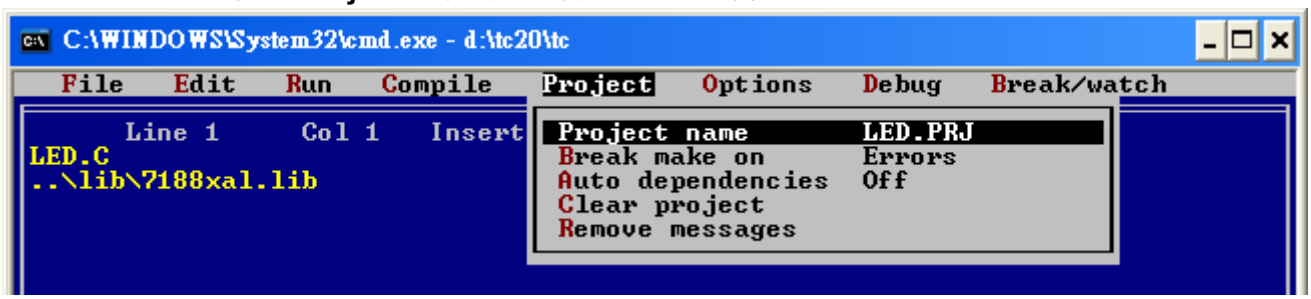
步骤 2: 编辑工程文件 (在 Project 中加入必要的库和文件)。



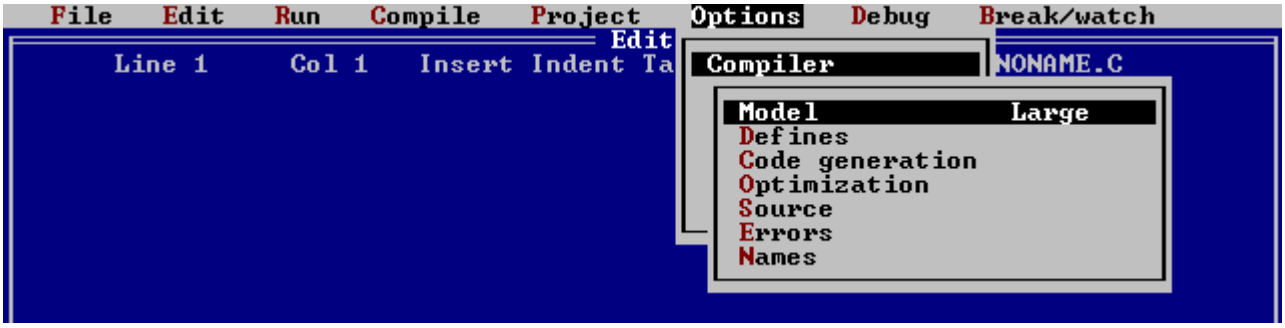
步骤 3: 在 File 目录中选择“save”将工程保存为工程文件，并输入文件名，如 LED.prj。

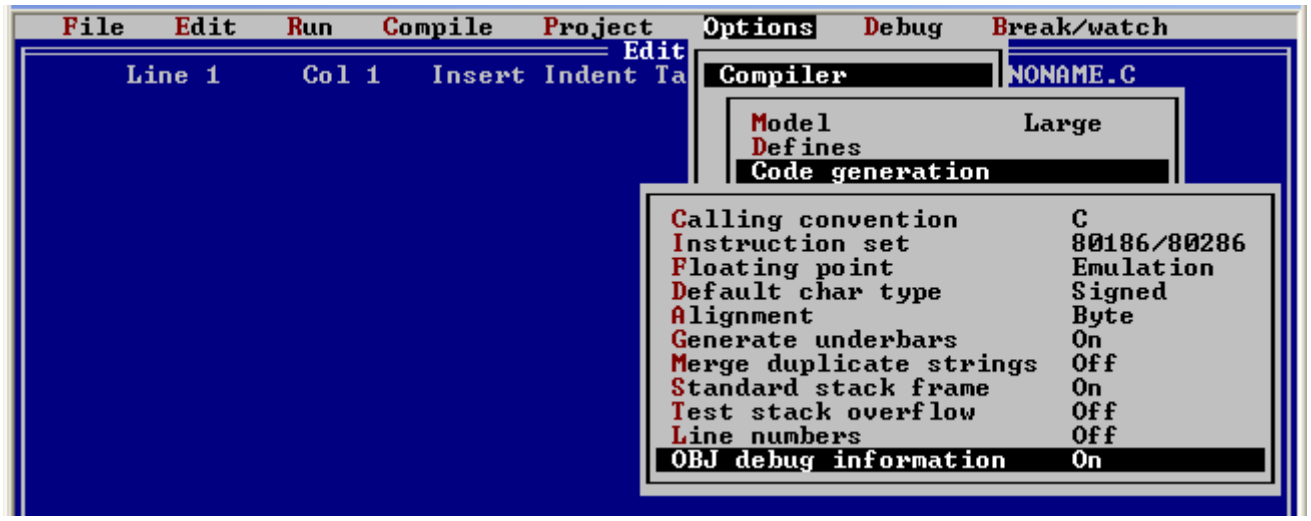


步骤 4: 通过在 Projent 菜单选择工程文件导入工程。

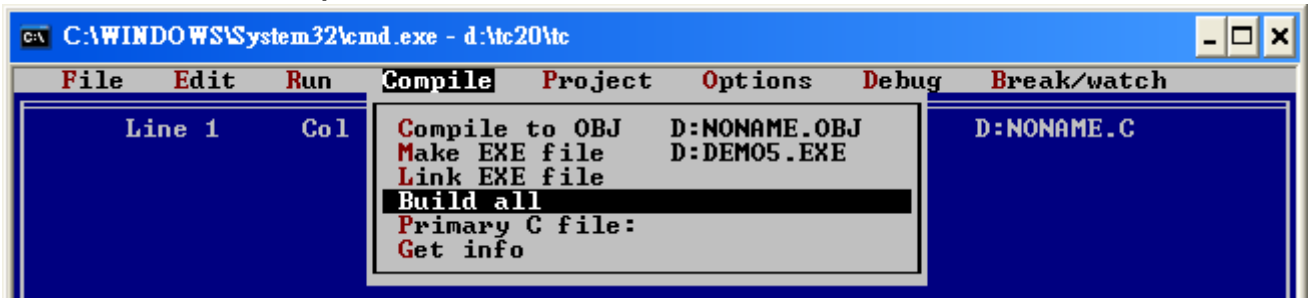


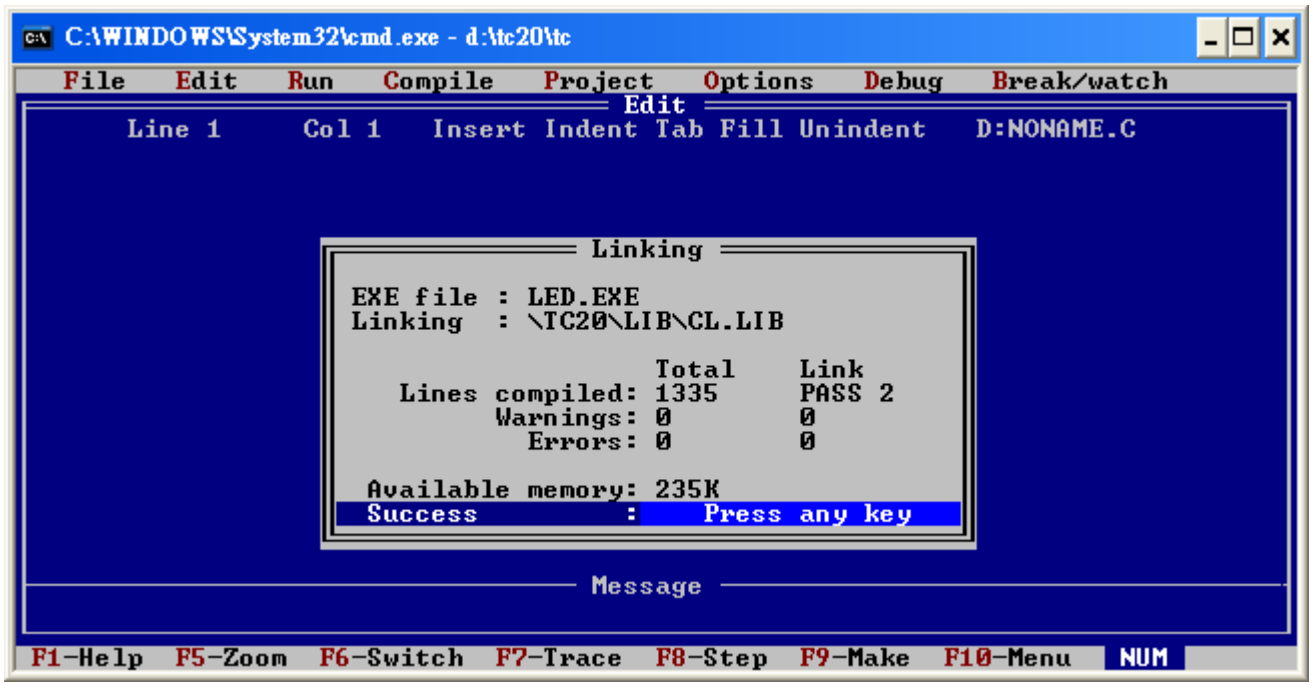
步骤 5: 在编译器的 Options 目录改变存储模式 (Small for 7188xbs.lib, large for 7188xbl.lib) 并设置代码生成为 80186/80286，请见如下对话框：





步骤 6: 选择 compile 的“Build all” 选项建立工程



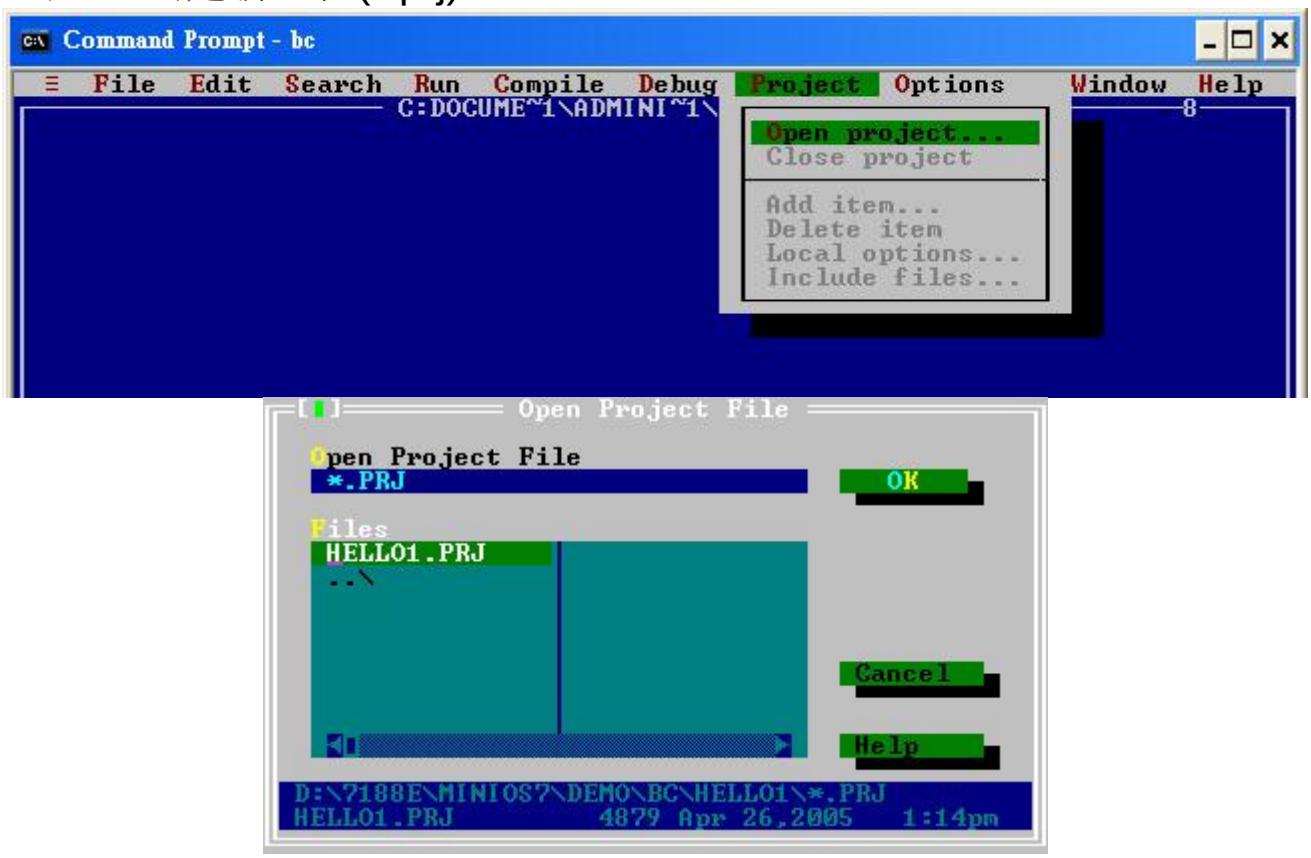

C:\WINDOWS\System32\cmd.exe - d:\tc20\tc
File Edit Run Compile Project Options Debug Break/watch
Edit
Line 1 Col 1 Insert Indent Tab Fill Unindent D:NONAME.C
Linking
EXE file : LED.EXE
Linking : \TC20\LIB\CL.LIB
Total Link
Lines compiled: 1335 PASS 2
Warnings: 0 0
Errors: 0 0
Available memory: 235K
Success : Press any key
Message
F1-Help F5-Zoom F6-Switch F7-Trace F8-Step F9-Make F10-Menu NUM

使用 BC++ 编译器

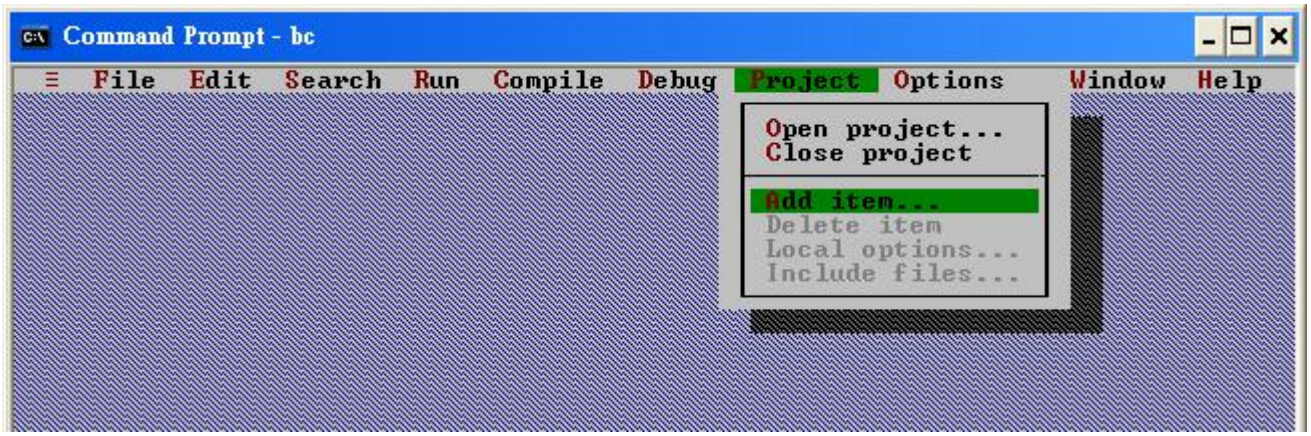
使用 BC++ 编译器的编译步骤如下：

步骤 1: 执行 Borland C++ 3.1.

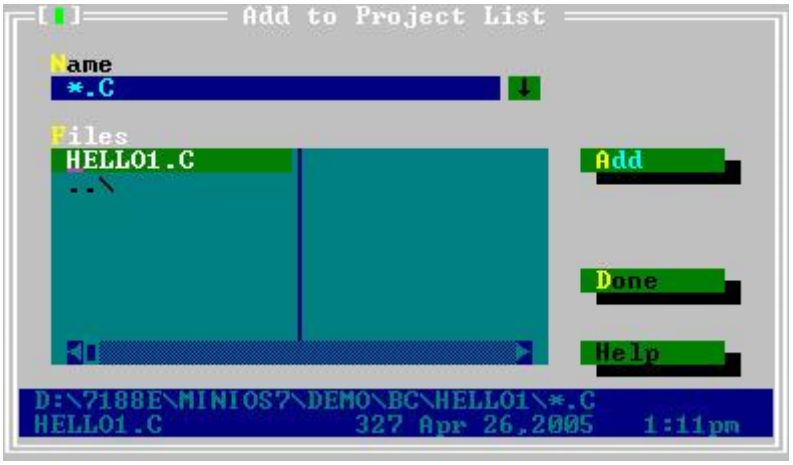
步骤 2: 创建新工程(*.prj).



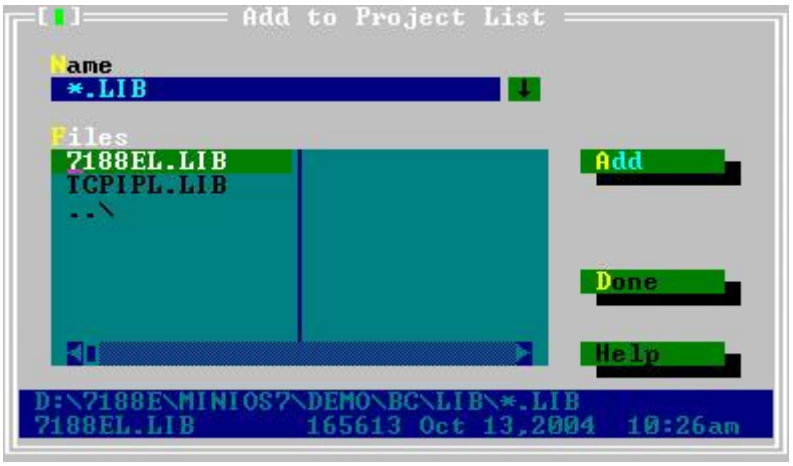
步驟 3:在工程文件中添加必要的文件。



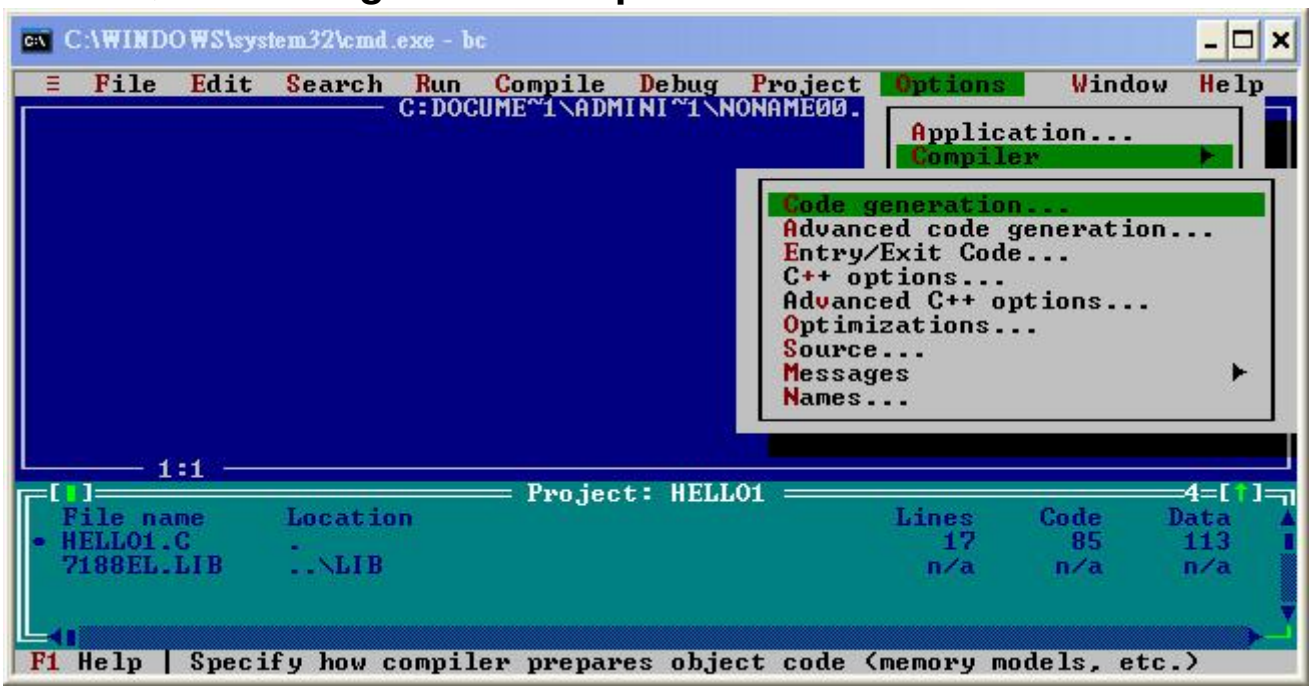
步驟 3.1: 选择源代码文件。



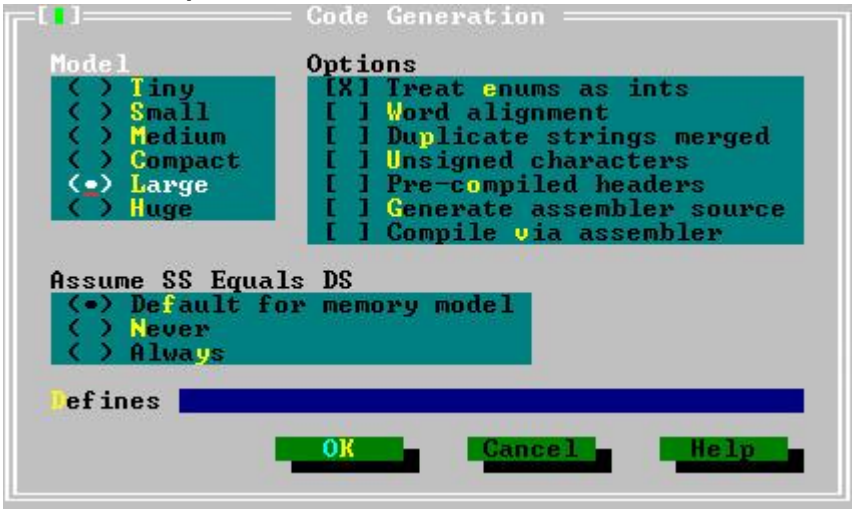
步驟 3.2: 选择函数库文件并点击按钮 **Done**。



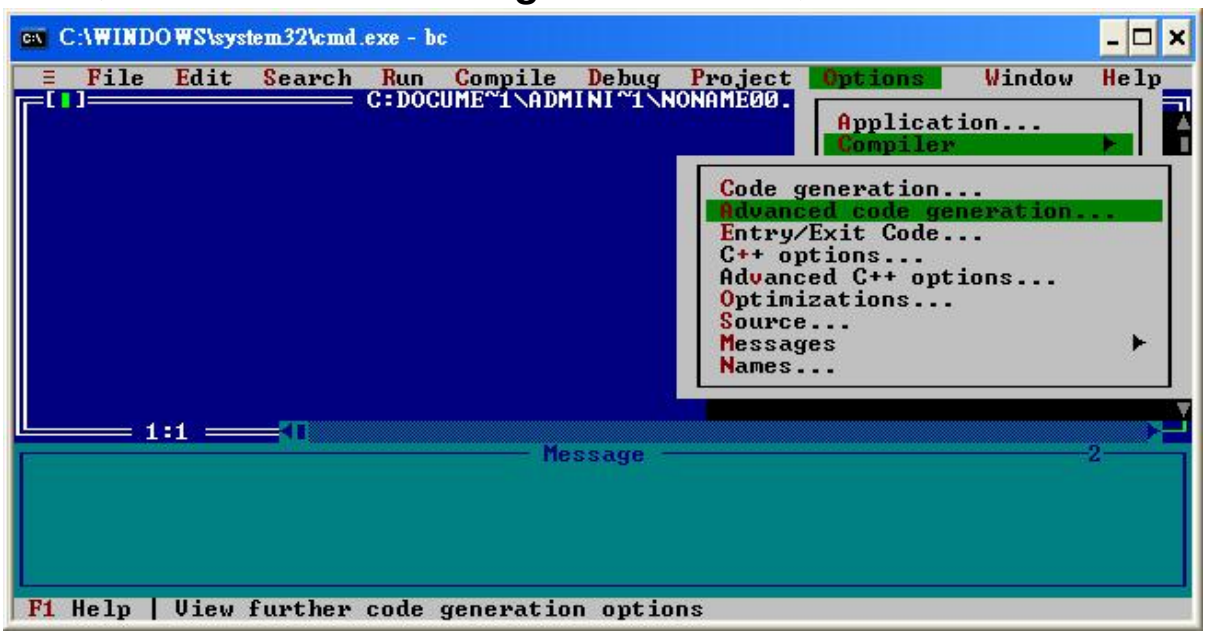
步驟 4: 设置 **Code generation option**



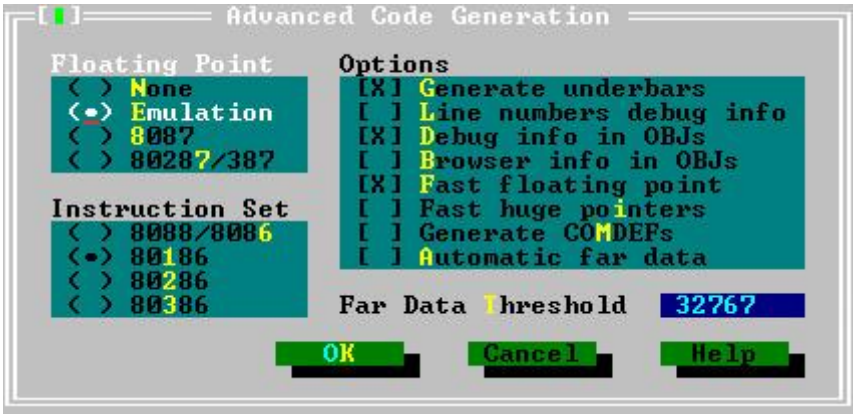
步驟 4.1: 改变存储模式(7188xbs.lib 适用于小型模式, 7188xbl.lib 适用于大型模式)。



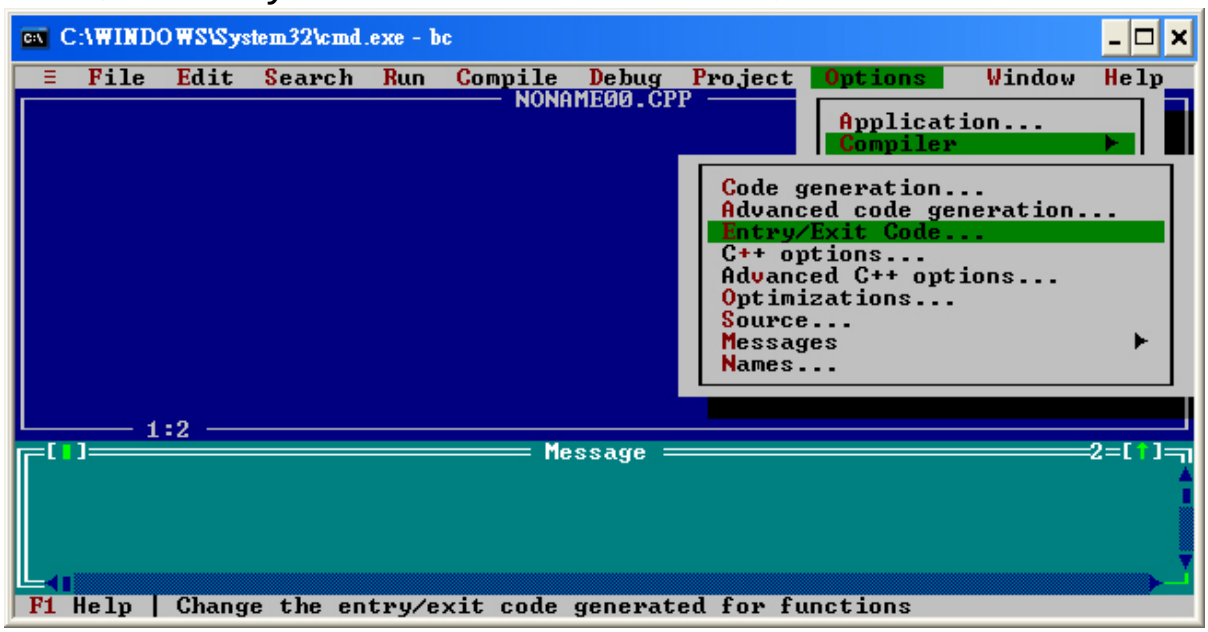
步骤 5: 设置 **Advanced code generation** 选项。



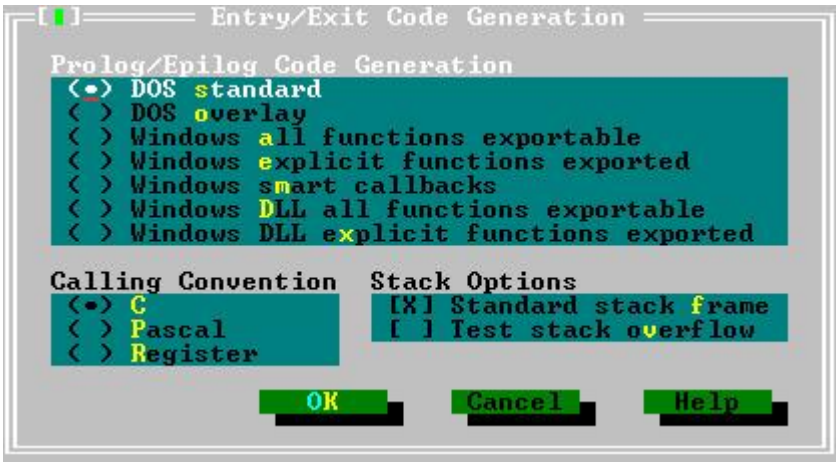
步骤 5.1: 设置 **Floating Point** 为 **Emulation**，设置 **Instruction Set** 为 80186。



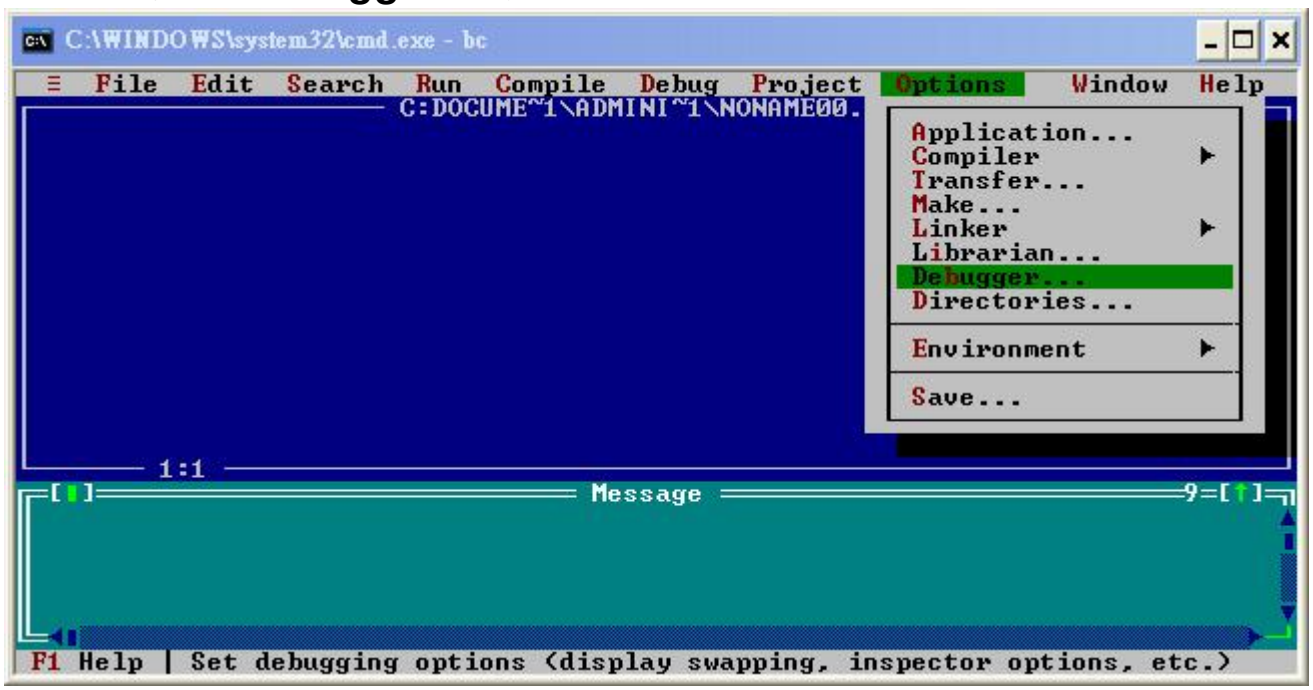
步骤 6: 设置 Entry/Exit Code Generation 选项。



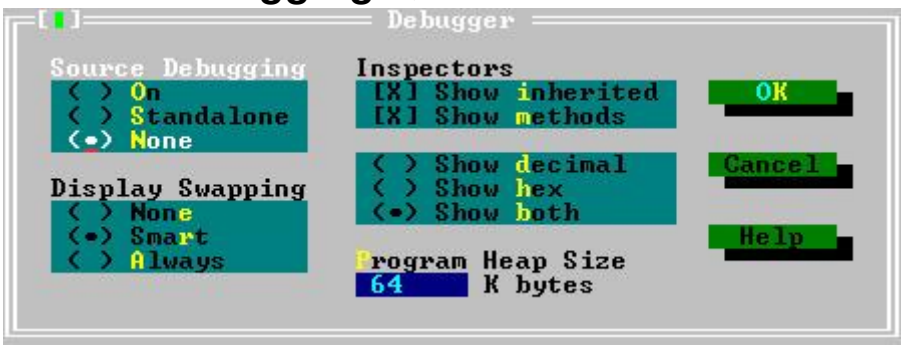
步骤 6.1: 设置为 **DOS standard**。



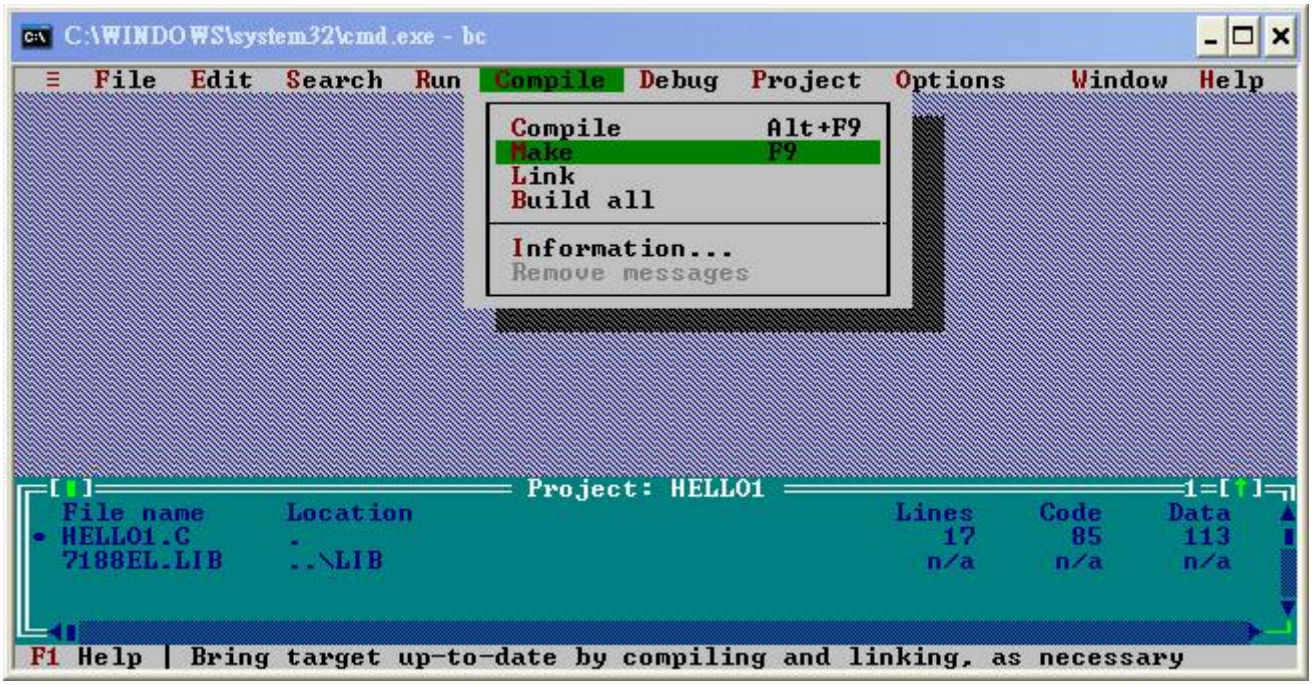
步骤 7: 设置 **Debugger** 选项。

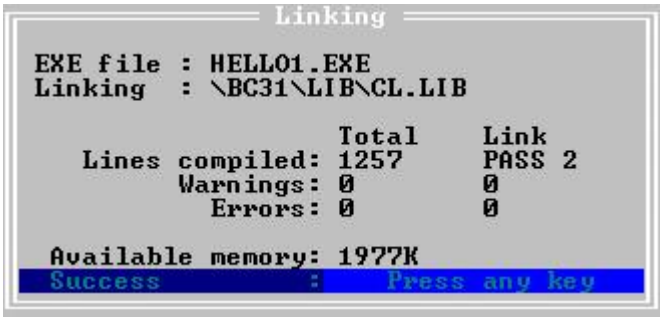


7.1 设置 **Source Debugging** 为 **None**。



步骤 8: 编译工程生成工程。

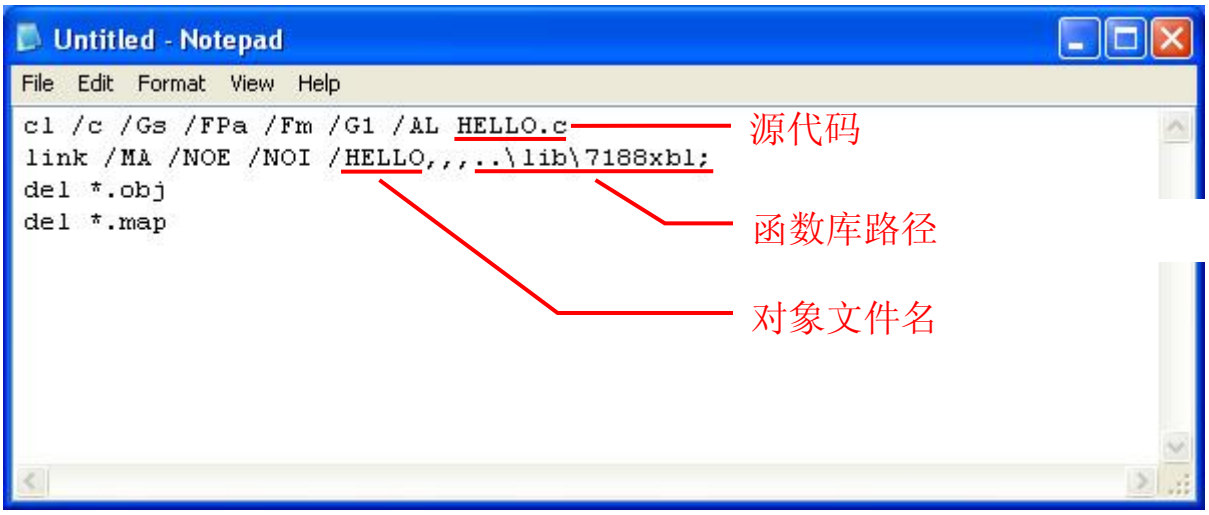




使用 MSC 编译器

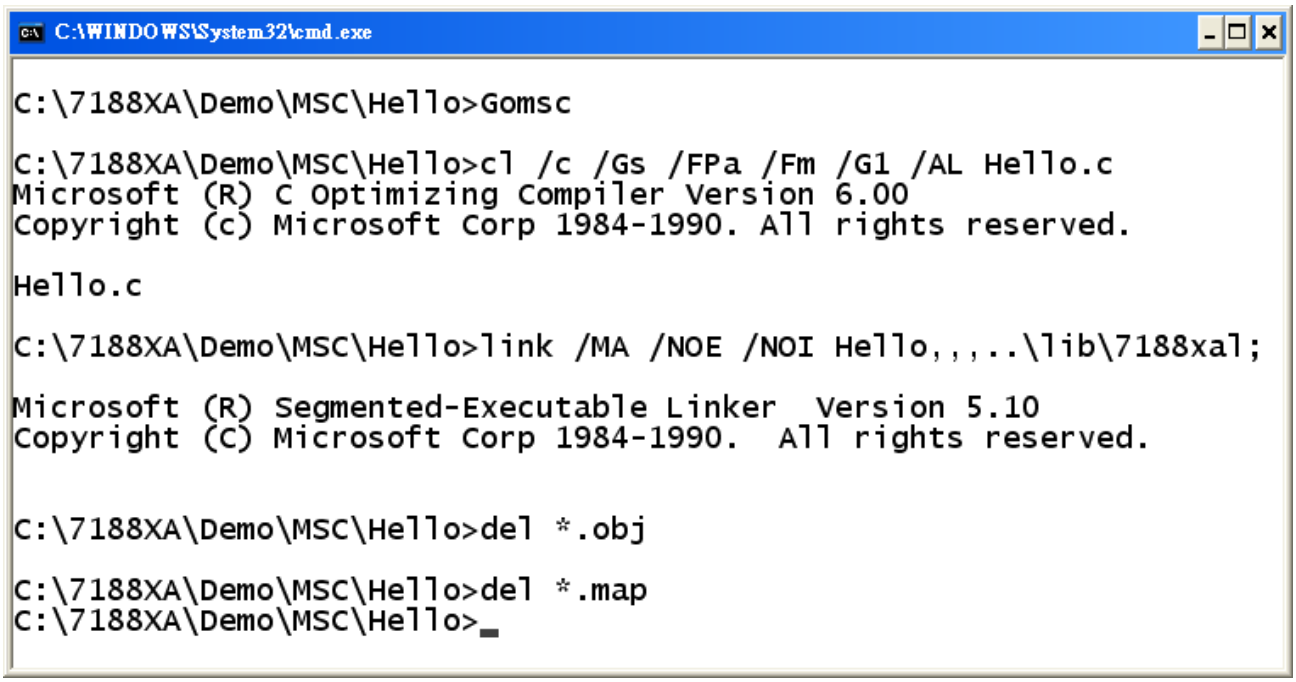
使用 BC++ 编译器的编译过程如下：

步骤 1: 在代码文件所在文件夹使用文本编译器创建一个文件名为 Gomsc.bat 的批处理文件。

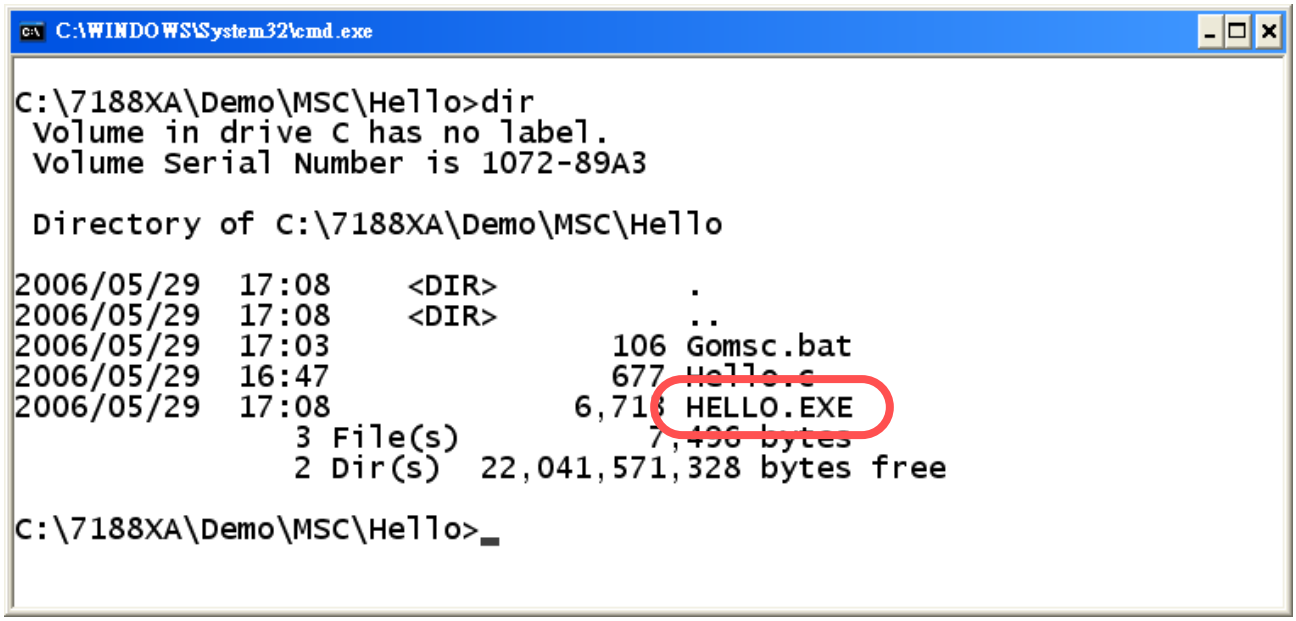


注意: **/C:** 不列出注释
/Gs: 不检查堆栈
/Fpa: 在目标文件中放入浮点运算库名称
/Fm: [map 文件]
/G1: 186 指令
/AL: 大型模式

步骤 2: 运行 Gomsc.bat。



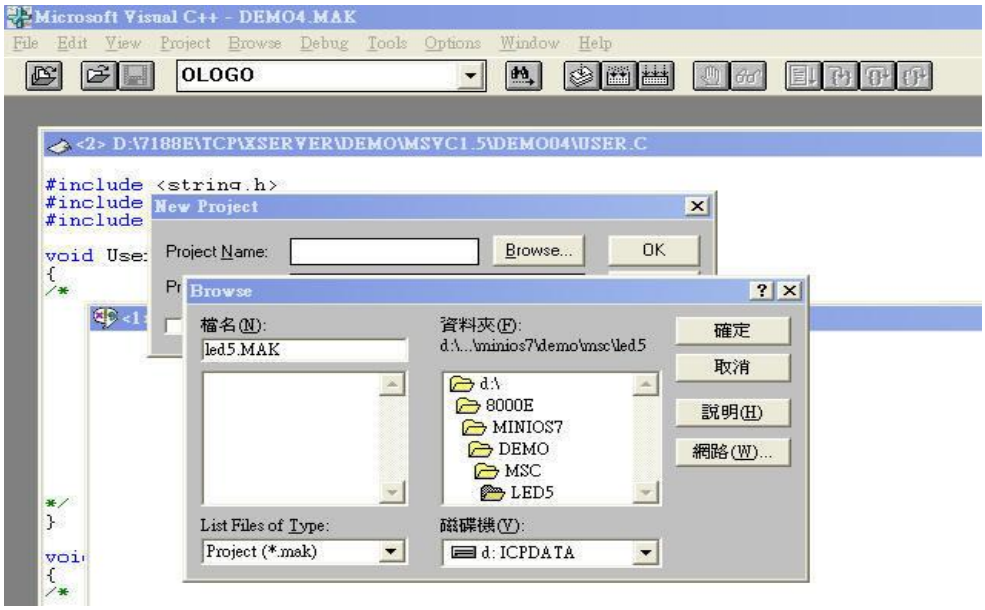
步骤 3: 如果成功编译将创建一个新的可执行文件。



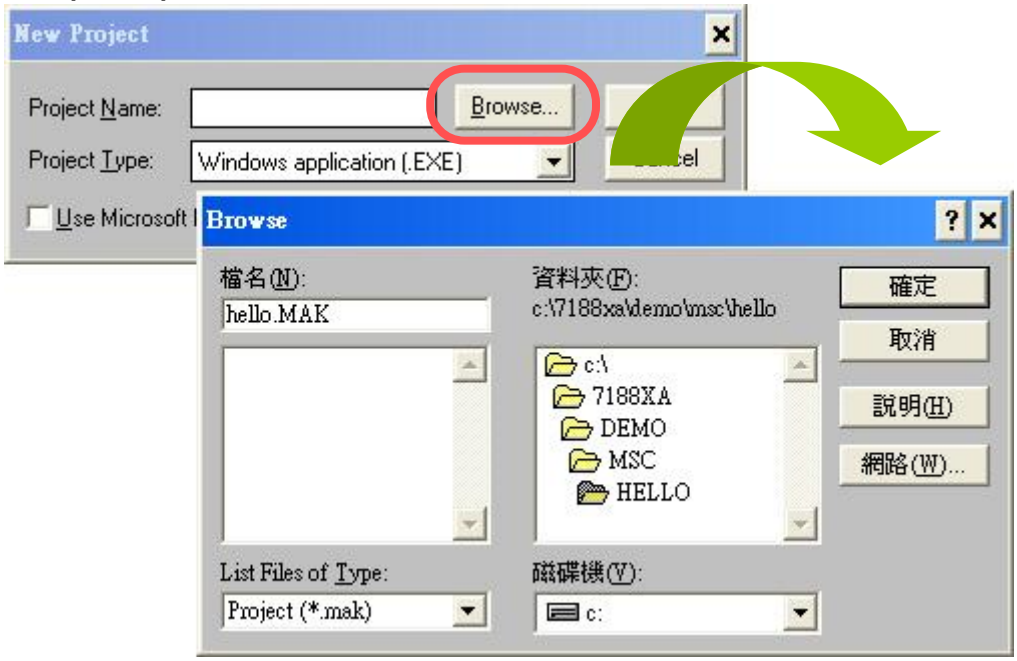
使用 MSVC++编译器

使用 MSVC 1.50 编译器的编译过程如下：

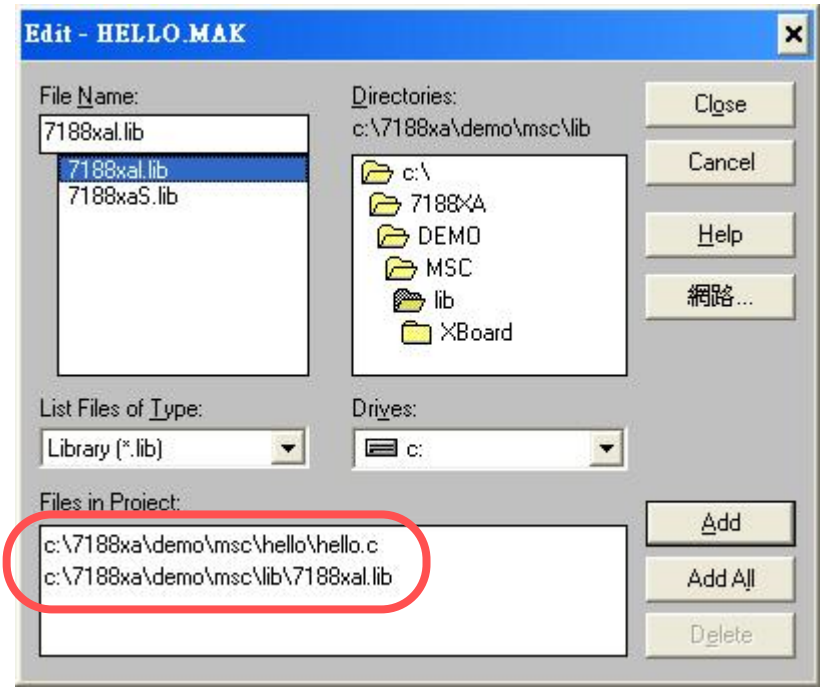
步骤 1: 运行 MSVC.exe



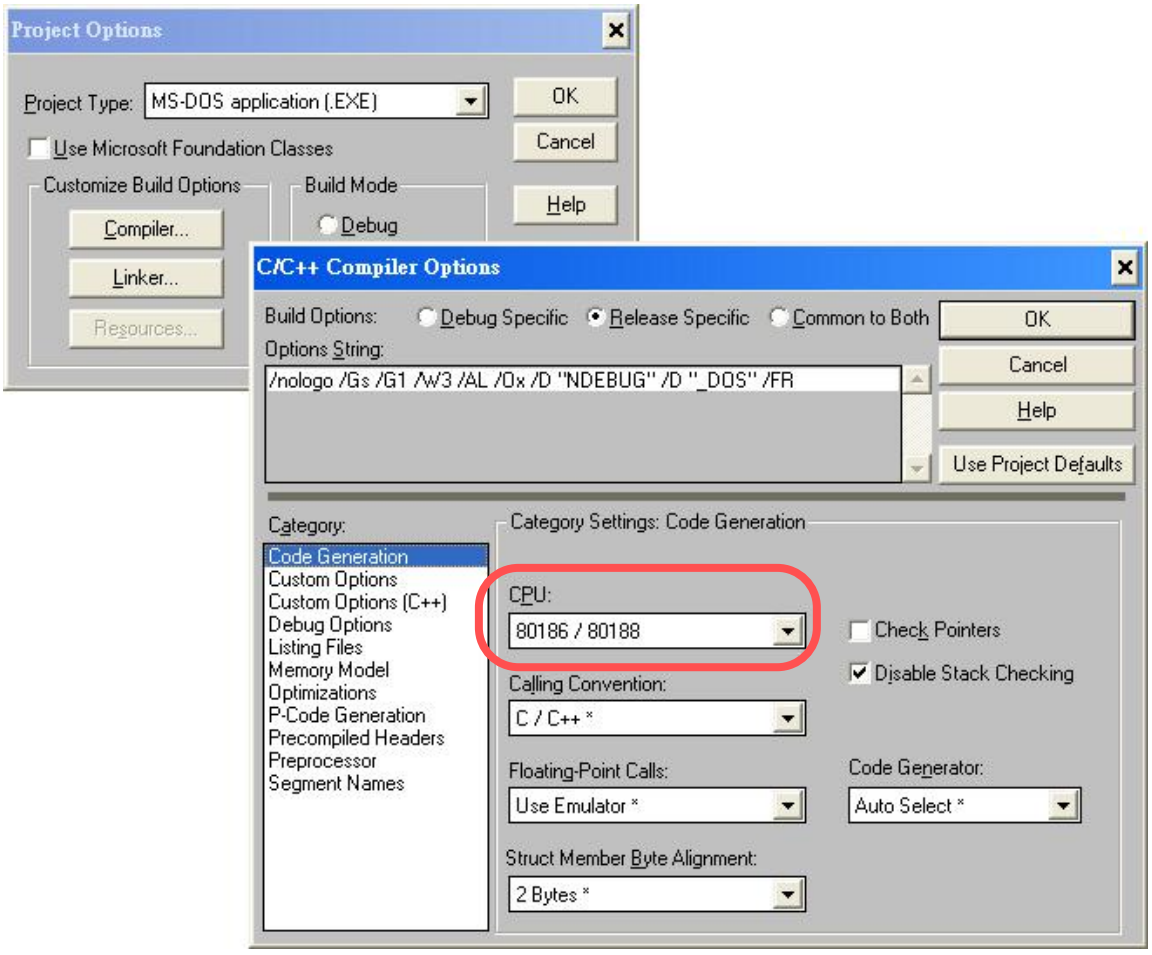
步骤 2: 在工程名区域输入工程名创建一个新的工程，并选择 MS-DOS 应用(EXE) 作为工程类型。



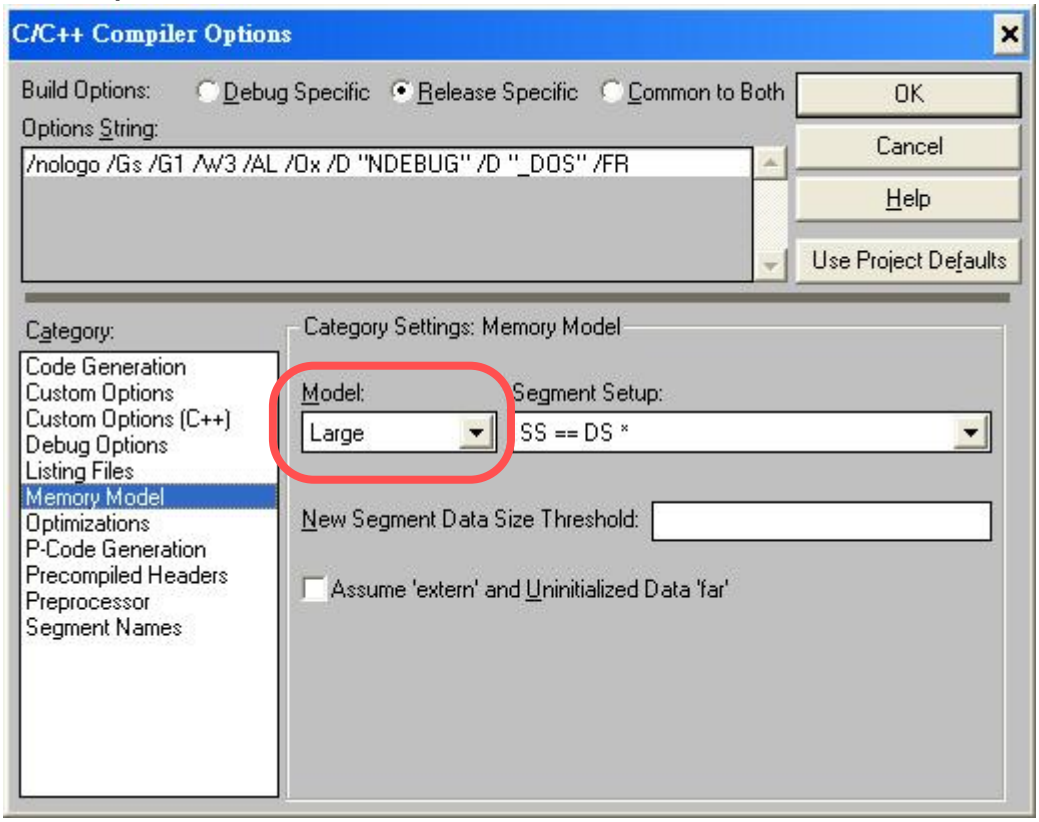
步骤 3: 在工程中加入用户程序和必要的库文件。



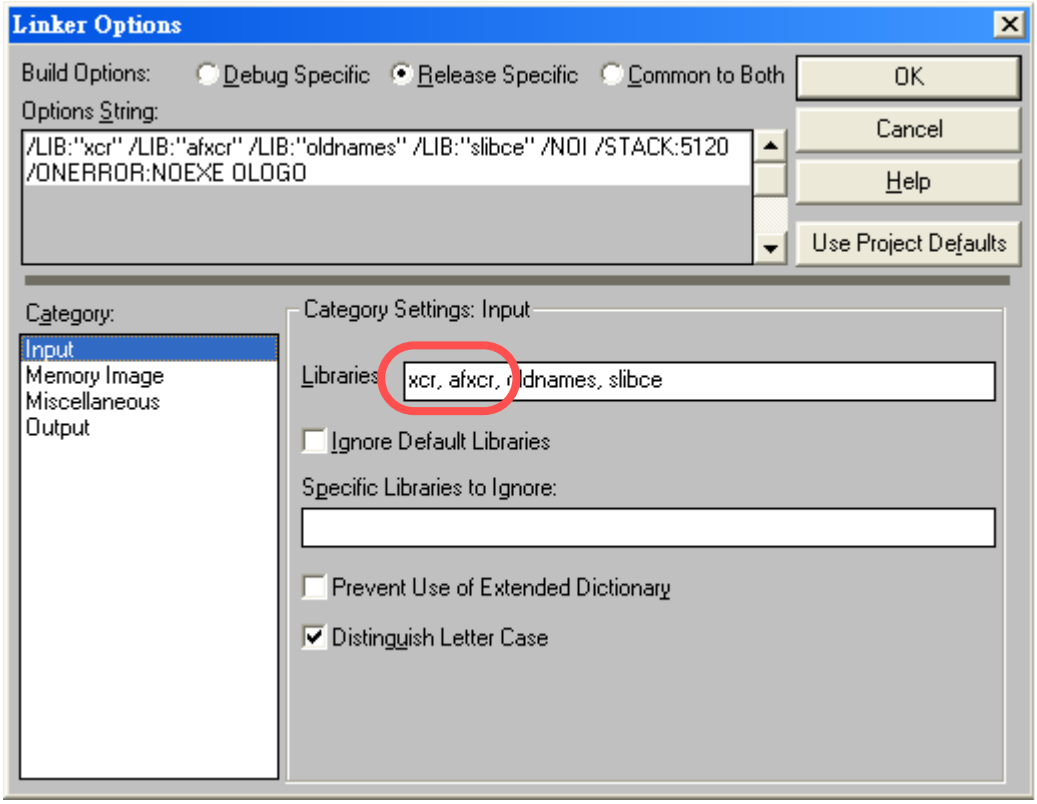
步骤 4: 在编译器中设置 **Code Generation**。

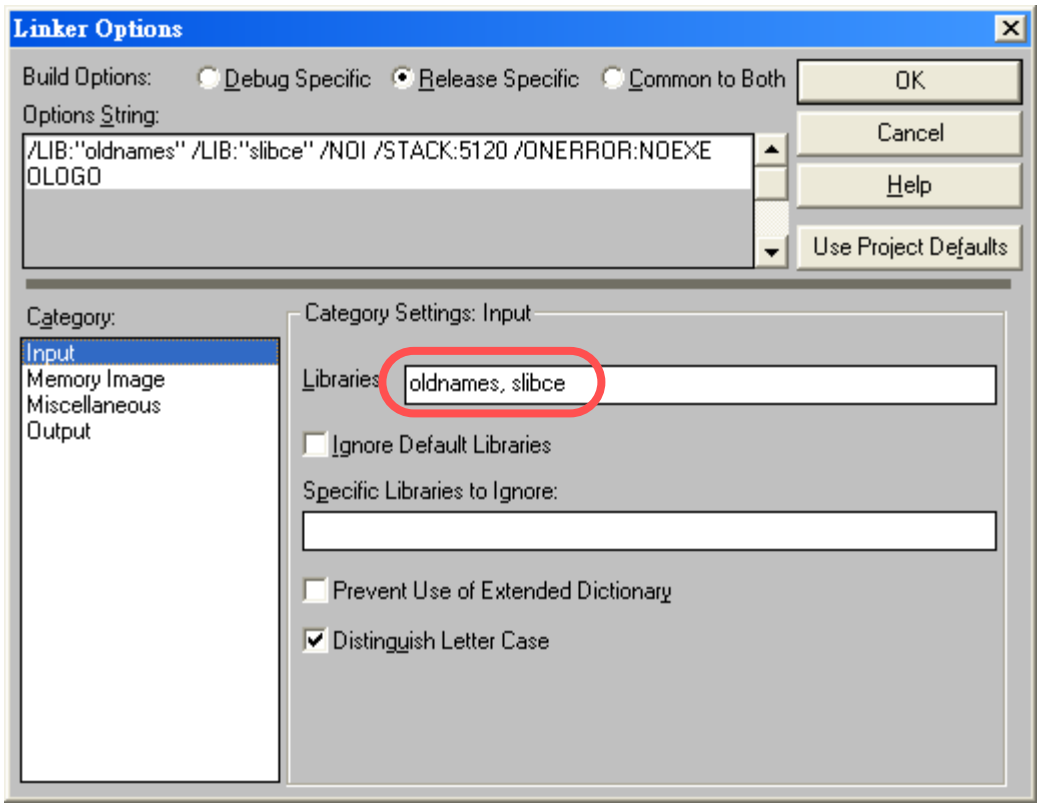


步骤 5: 变换 **Memory model** (7188xbs.lib 适用于小型,7188xbl.lib 适用于大型)。

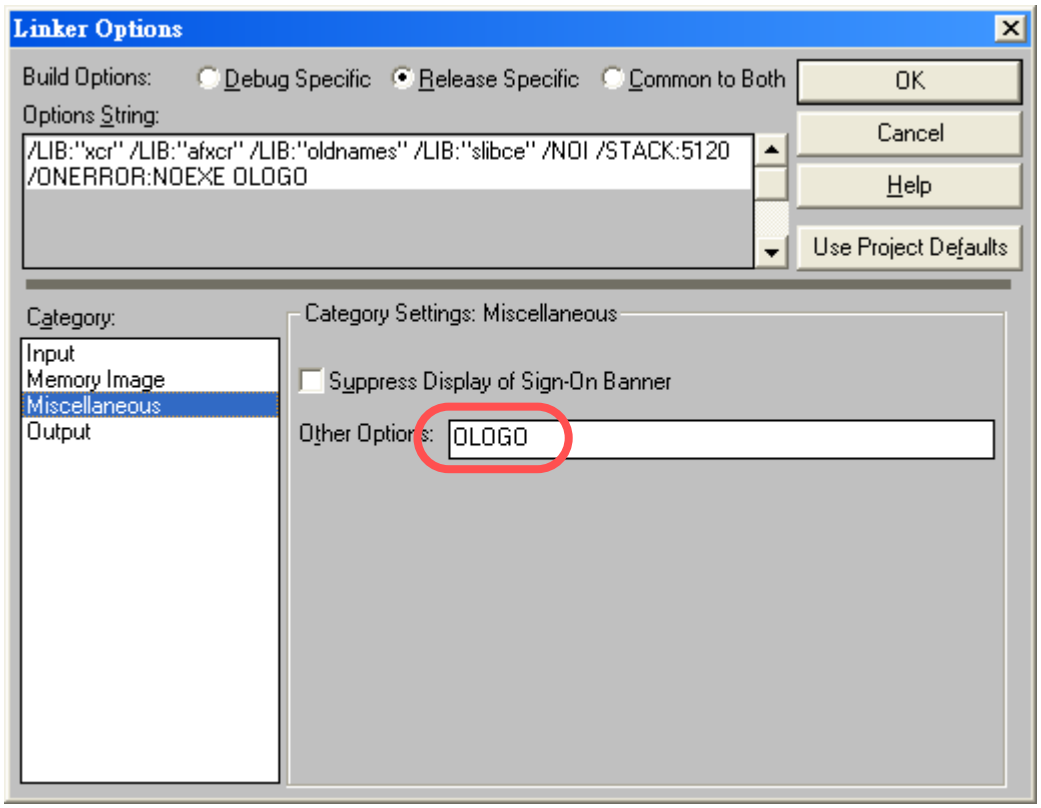


步骤 6: 在 Input 类中移除 xcr, afxcr 库。

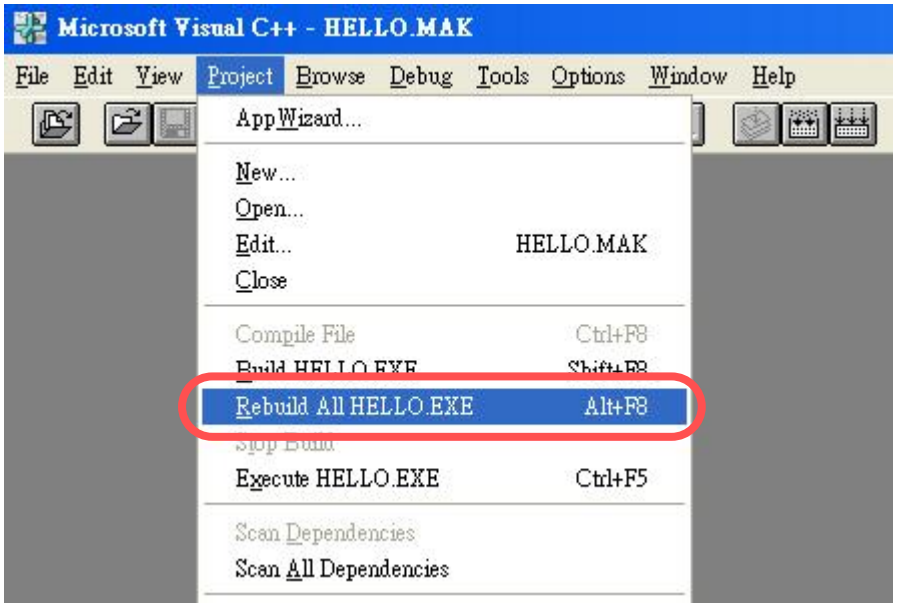


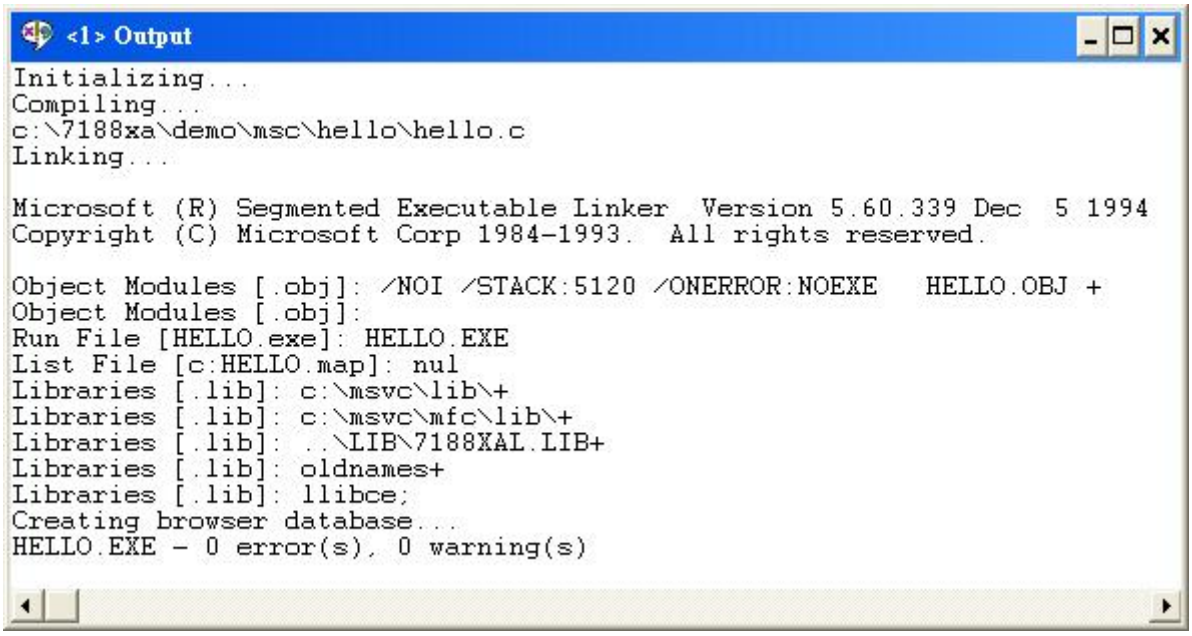


步驟 7: 在 **Miscellancous** 类中移除 OLOGO 选项。



步驟 8: Rebuild 项目。



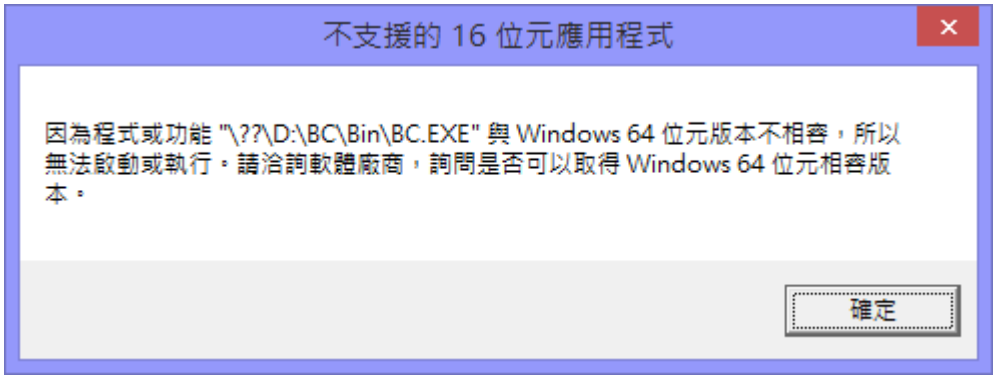


在 64 位平台编译

前言:

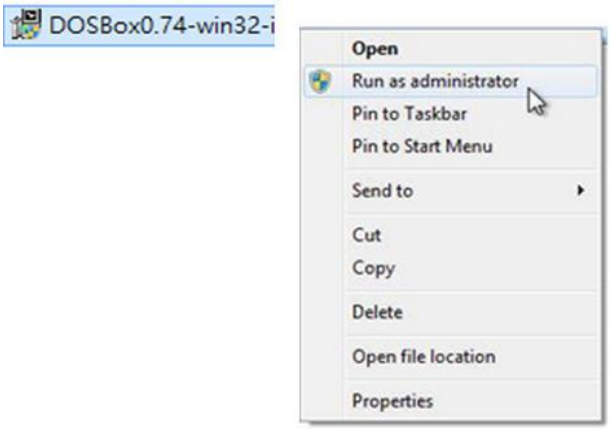
假如要在 Windows 7 或 Windows 8 这种有支持 64 位的 OS 平台上使用 16 位的编译器(像是 BC3.1 或是 TC++3.0)创建 MiniOS7 项目时，将会告知有 64 位平台兼容性问题的信息出现。

“**Application can’t run on your PC. Contact with your software publisher**” 或是下列截图信息.



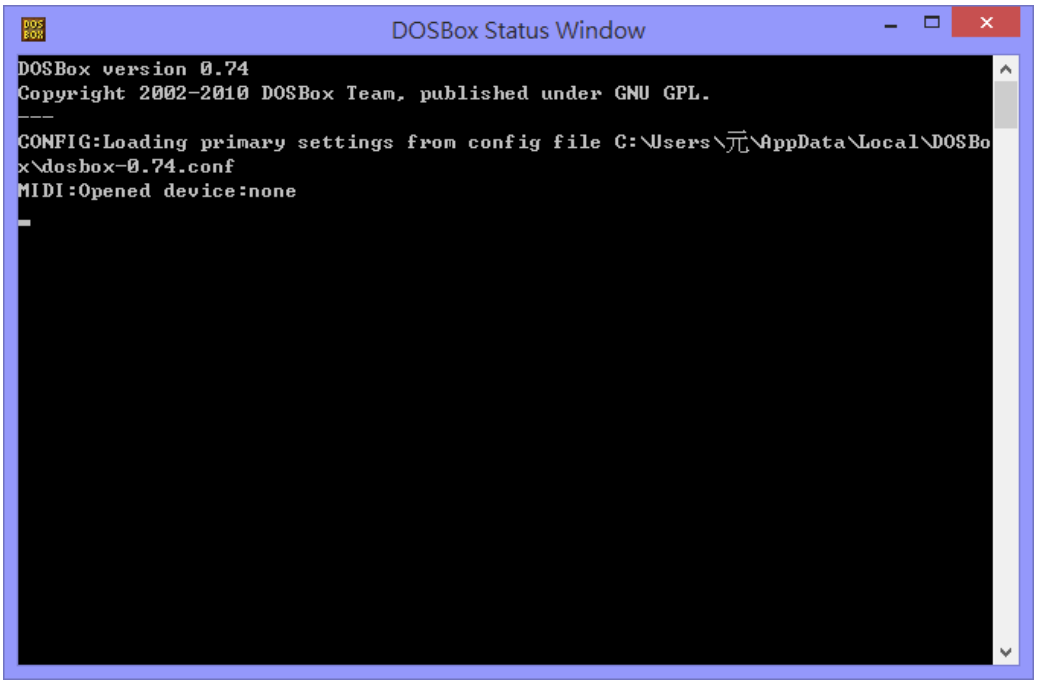
请遵循如下的步骤来解决上述的问题。

步骤 1: 下载和安装 Windows 环境的 DosBox. 你可以很容易在 64 位的 OS 上安装 32 位的版本。下载的网页如下：
http://sourceforge.net/projects/dosbox/files/dosbox/0.74/DOSBox0.74-win32-installer.exe/download
注意：强烈建议以系统管理员身份安装 DOSBox。

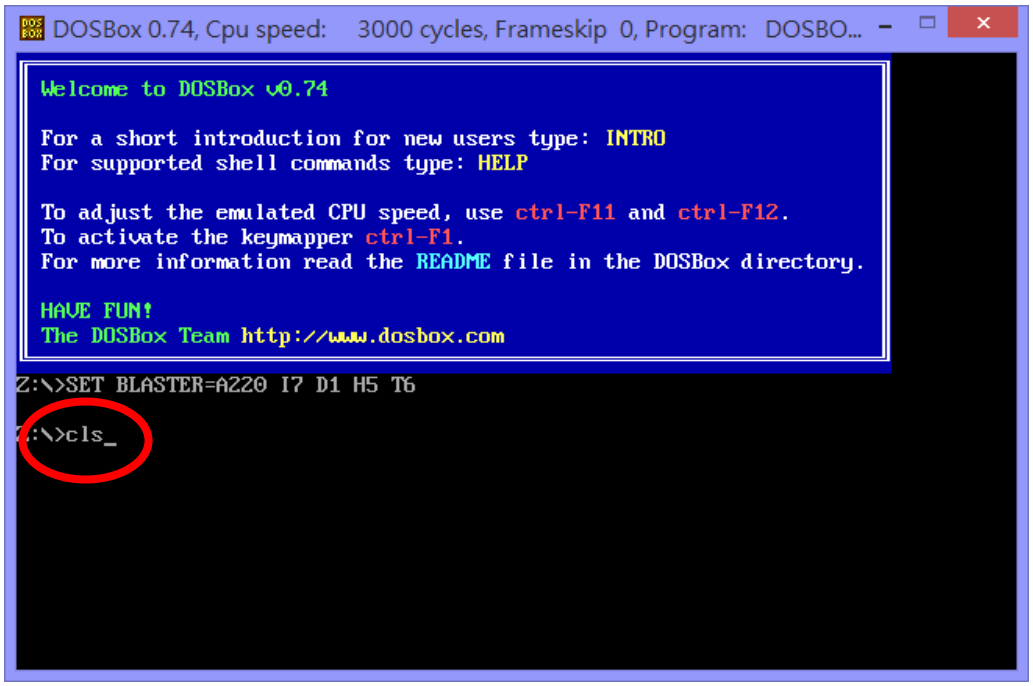


步骤 2: 启动 DOSBox. 双击 DOSBox 

，将会产生二个窗口，一个是 DOSBox Status Window, 另一个是 DOSBox Console. 请参考如下的图。

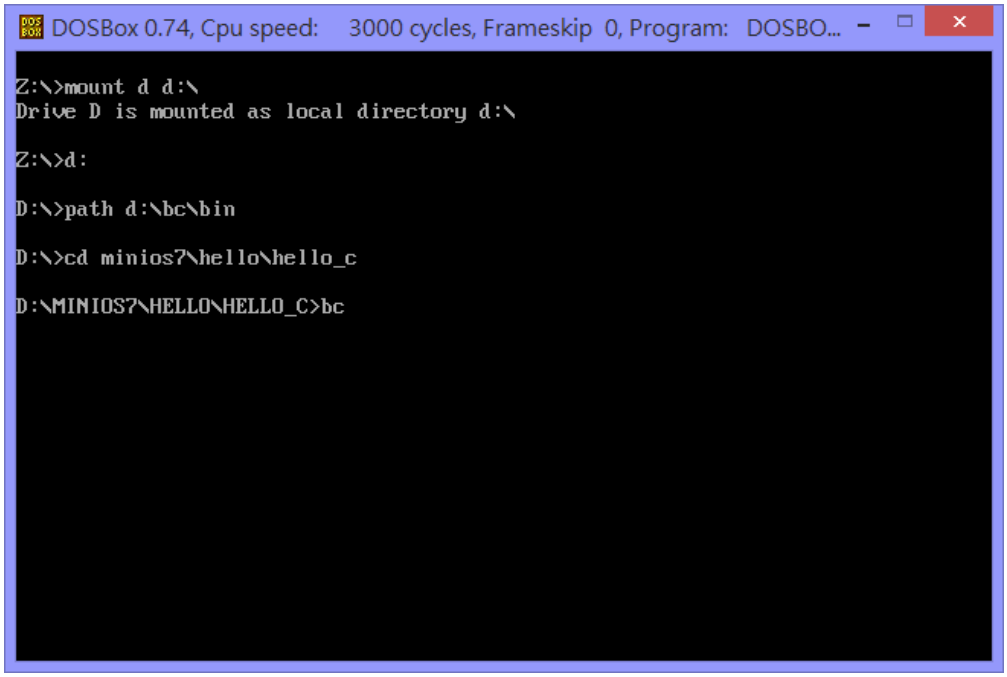


首先输入命令“CLS”或“cls”来清除屏幕上的文字。

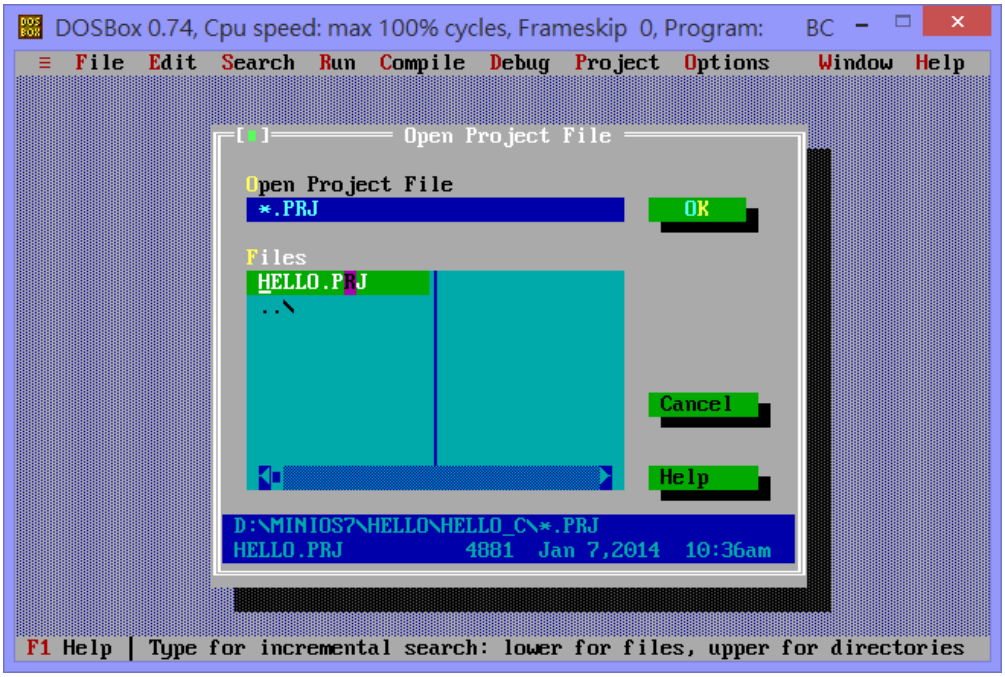


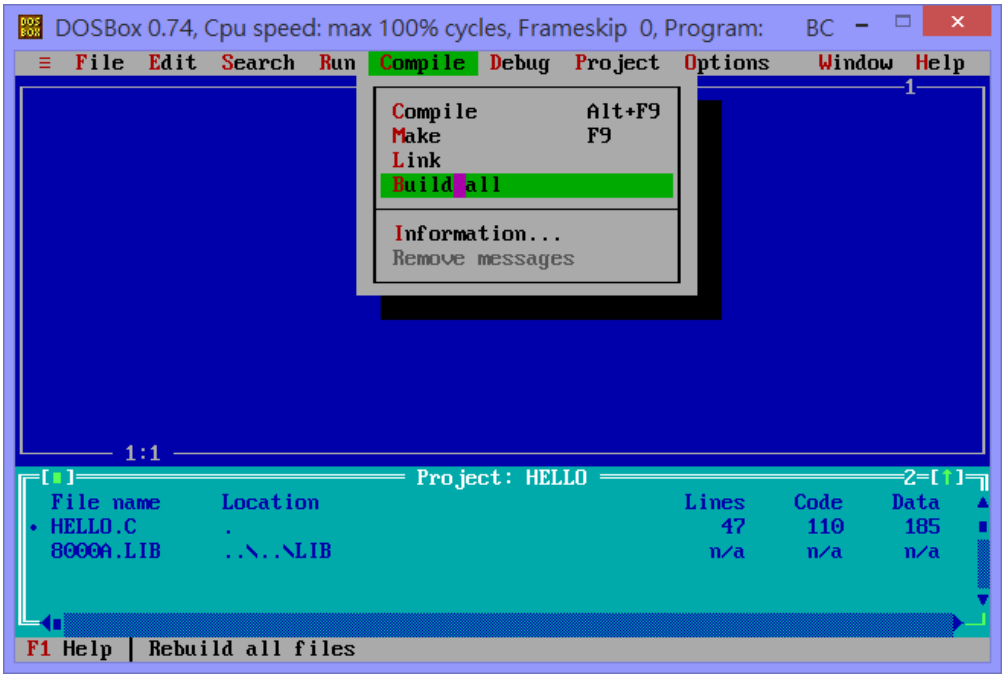
步驟 3: 设定 DOSBox 的环境设定.

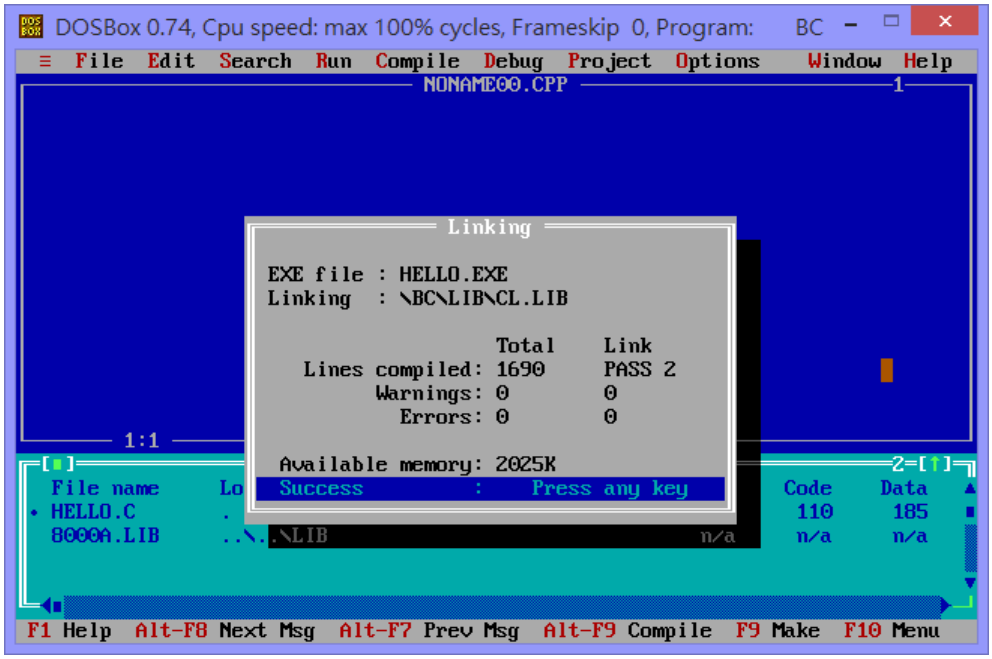
1. 确定 Demo 档案路径在哪，例如 D:\MiniOS7\hello\hello_c
2. 确定 BC\Bin 的档案路径，例如 D:\BC\Bin
3. 使用”mount”命令加载磁盘到 DOSBox，例如“mount d d:\”
4. 加载磁盘之后，输入“D:” ,可将“Z:\>”变成“D:\>”.
5. 设定编译器路径到 BC\Bin，例如 “path d:\bc\bin”
6. 变换路径到 Demo 所在的数据夹，然后输入“bc”

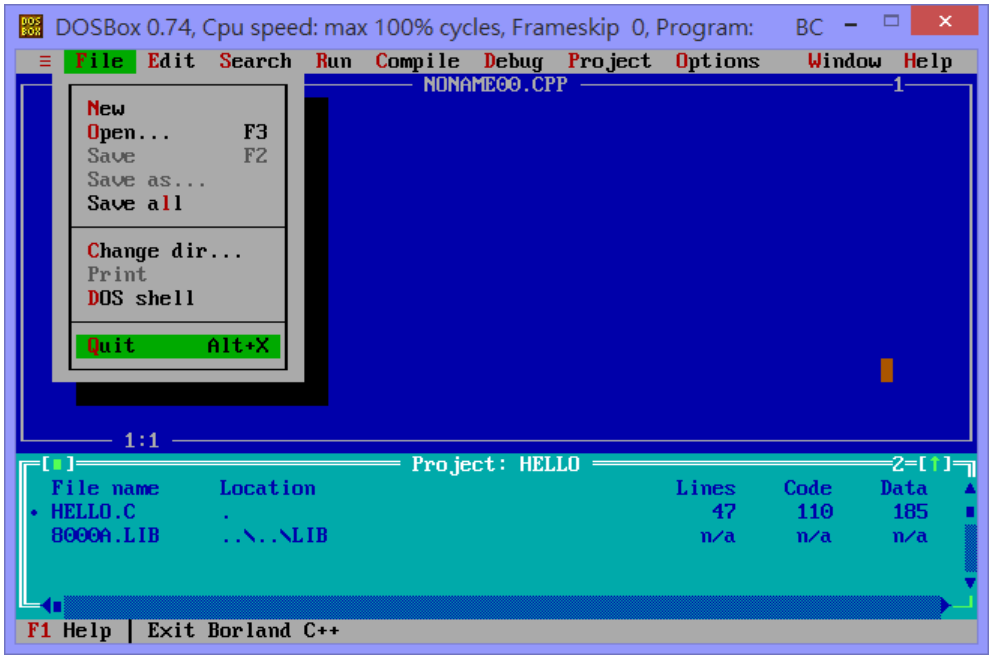


步驟 4: 点击”Project” ➔ 选择 “*. prj “ 来打开专案档 ➔ 点击”Compile” 编译项目.





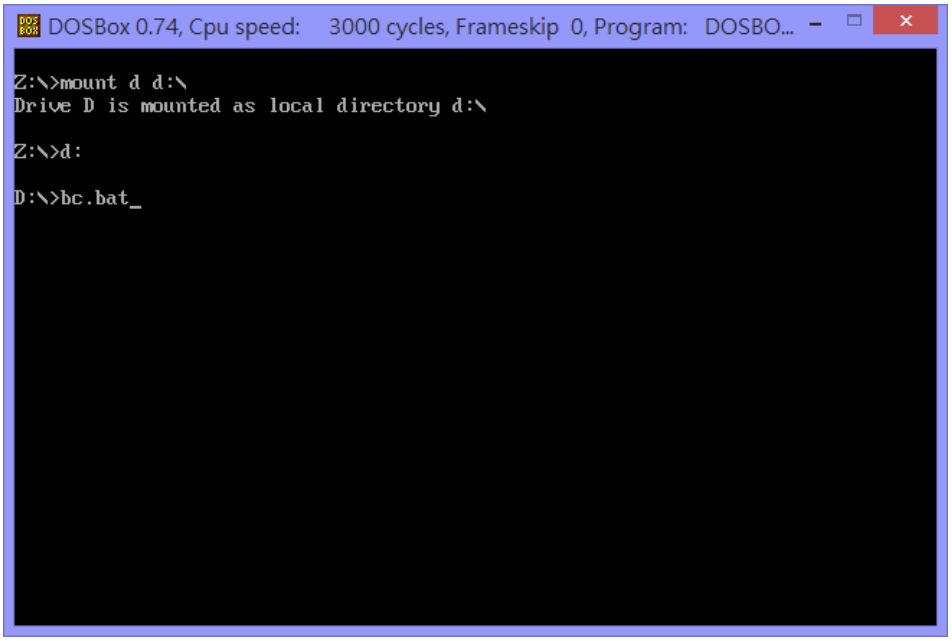

DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: BC
File Edit Search Run Compile Debug Project Options Window Help
NONAME00.CPP
Linking
EXE file : HELLO.EXE
Linking : \BC\LIB\CL.LIB
Total Link
Lines compiled: 1690 PASS 2
Warnings: 0 0
Errors: 0 0
Available memory: 2025K
Success : Press any key
1:1
File name Lo Code Data
HELLO.C . 110 185
8000A.LIB ..\..\LIB n/a n/a
F1 Help Alt-F8 Next Msg Alt-F7 Prev Msg Alt-F9 Compile F9 Make F10 Menu


DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: BC
File Edit Search Run Compile Debug Project Options Window Help
NONAME00.CPP
New
Open... F3
Save F2
Save as...
Save all
Change dir...
Print
DOS shell
Quit Alt+X
1:1
Project: HELLO
File name Location Lines Code Data
HELLO.C . 47 110 185
8000A.LIB ..\..\LIB n/a n/a n/a
F1 Help | Exit Borland C++

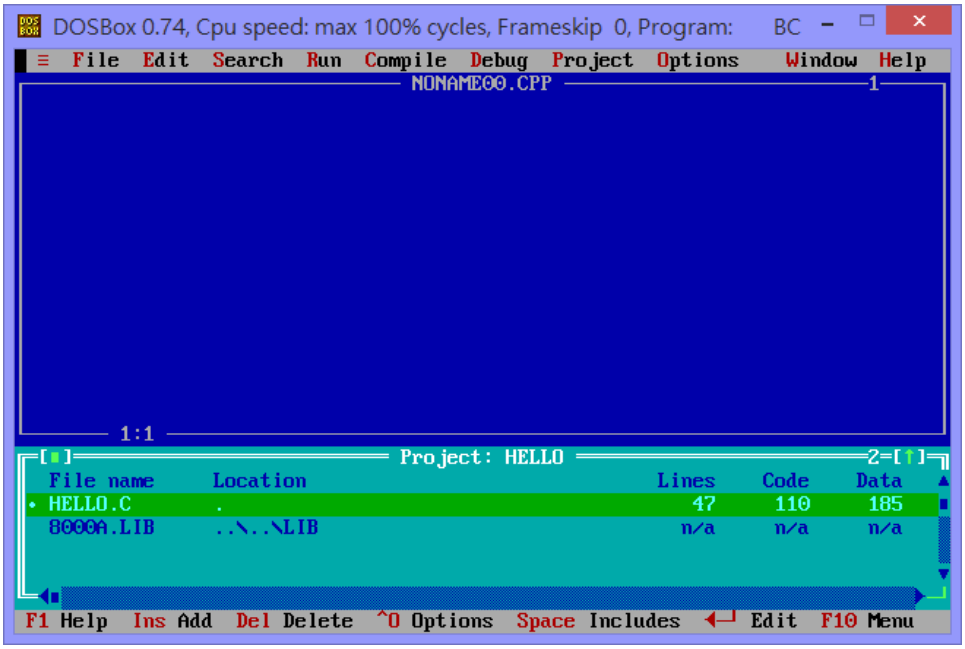
小技巧——使用批处理文件:
在 DOSBox Console 窗口中设定路径和呼叫编译器是有一点点令人感到麻烦的。我们可以制作一个**批处理文件**来省却麻烦,

```
path d:\bc\bin
cd minios7\hello\hello_c
bc
```

将此 batch 档储存在“D:\”路径下，当 D 槽被加载到 DOSBox 后，更换路径到 d:，执行**批处理文件**。

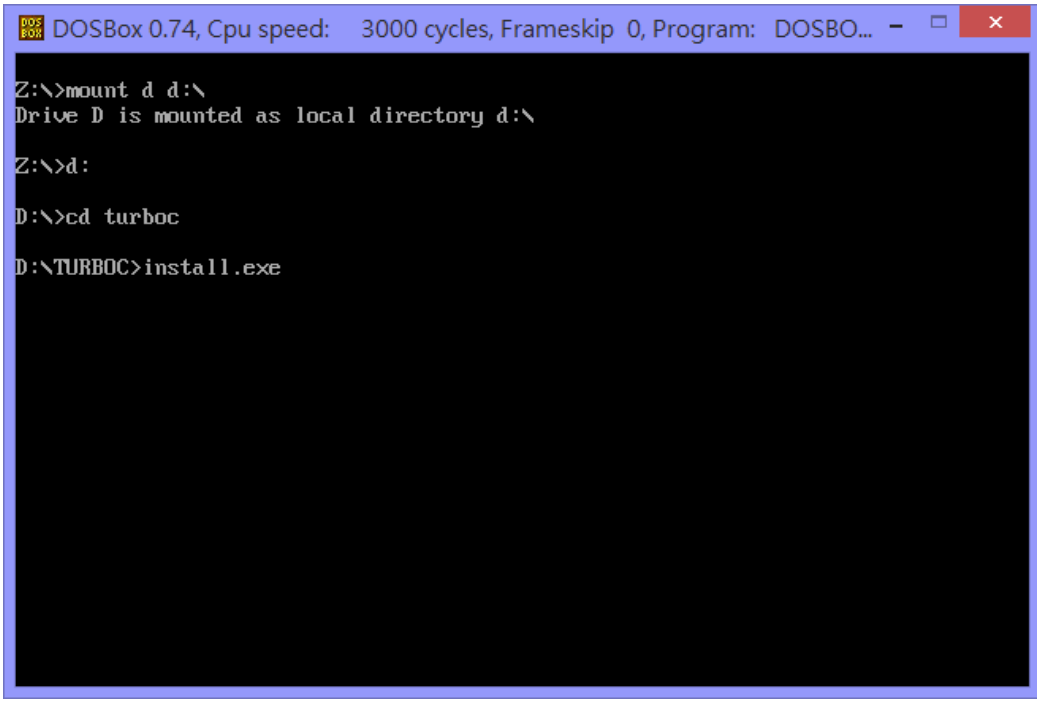


当执行 bc.bat 档后, Demo 项目会成功的被呼叫起来.

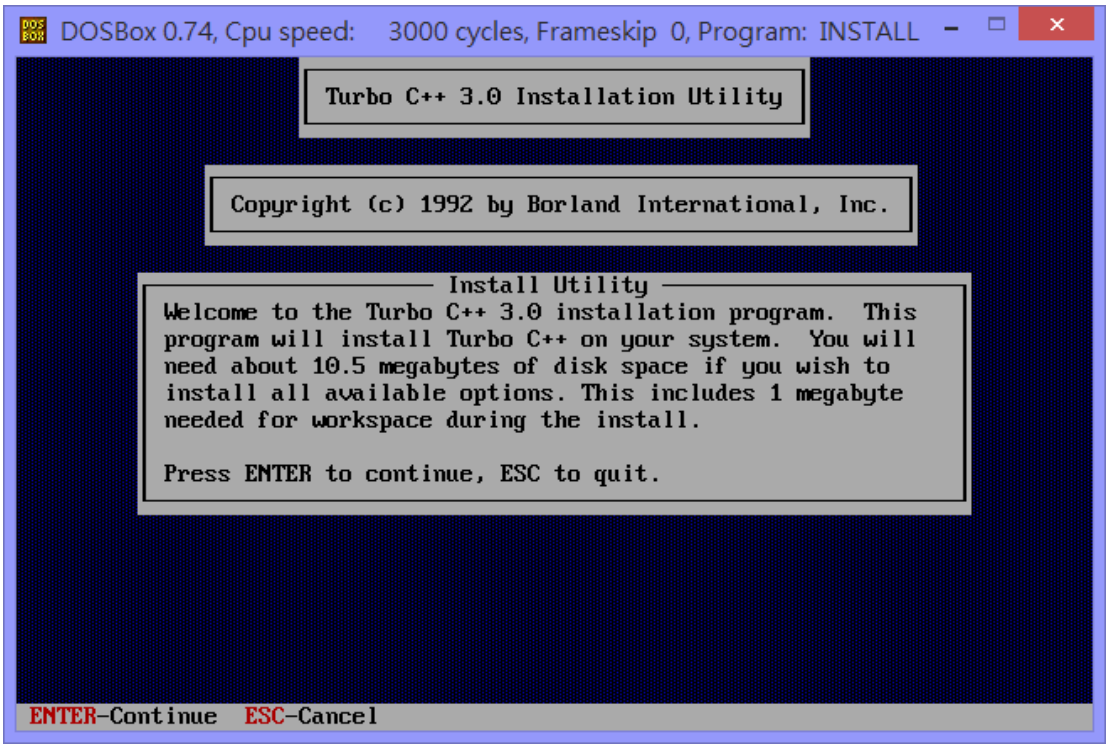


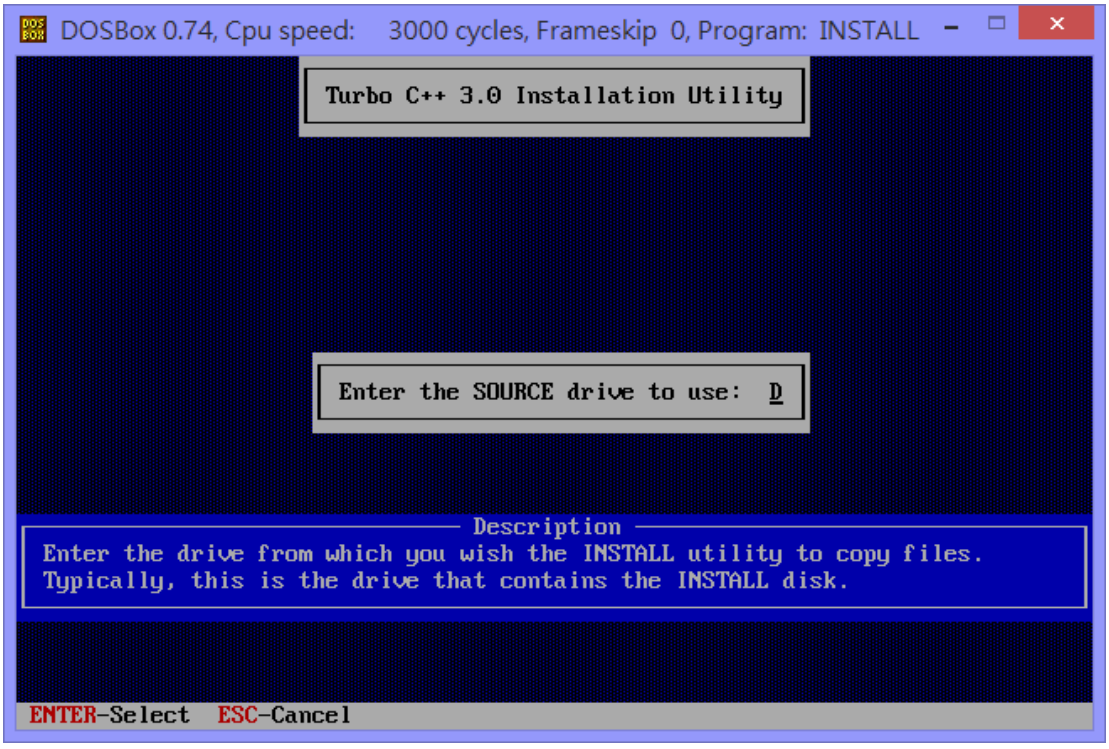
假设 BC3.1 编译器无法取得，可以从网络下载免费的 TurboC++3.0.
可以从如下的链接下载免费的 **turboc.zip**
http://www.bestfreewaredownload.com/download/t-free-turbo-c--freeware-flggsdpz.html

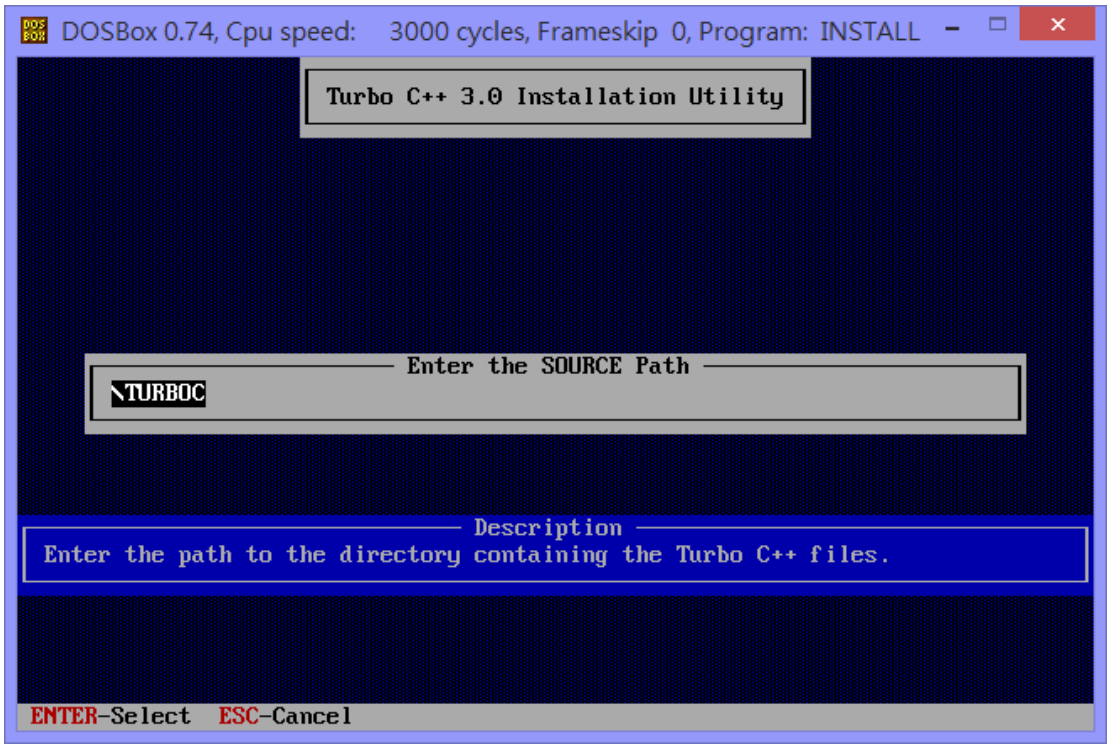
将 turboc.zip 解压缩后放到 D:\ 下，执行 DOSBox ➔ 执行 install.exe 安装 TC++3.0

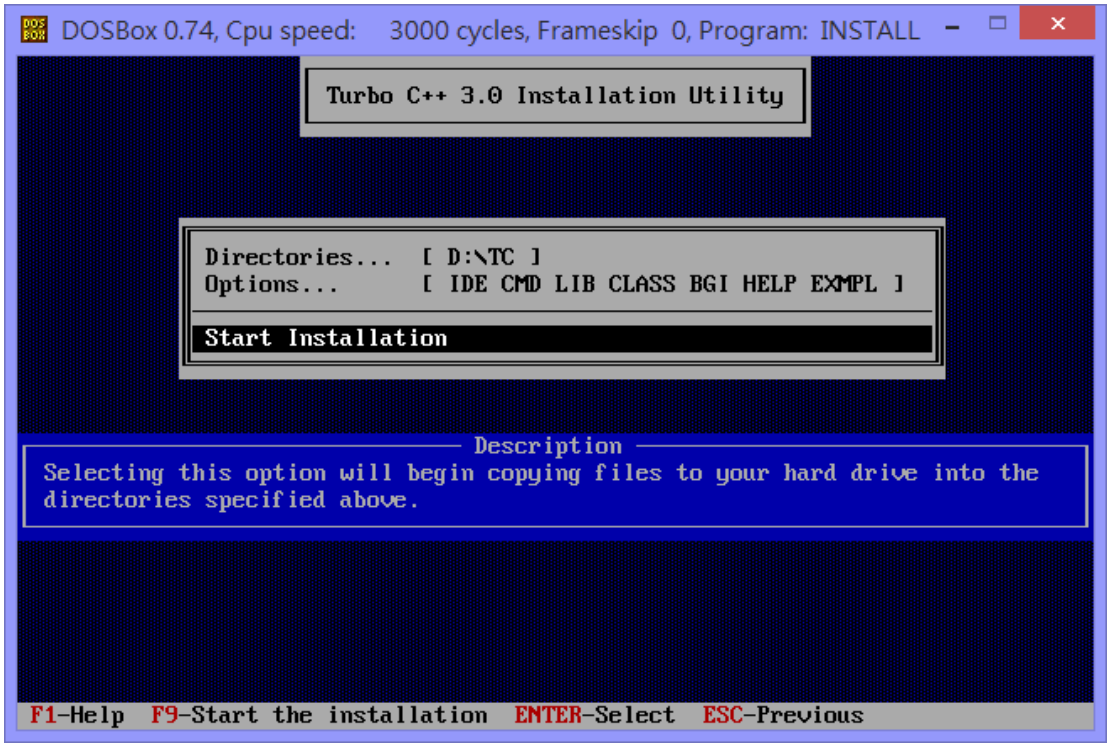


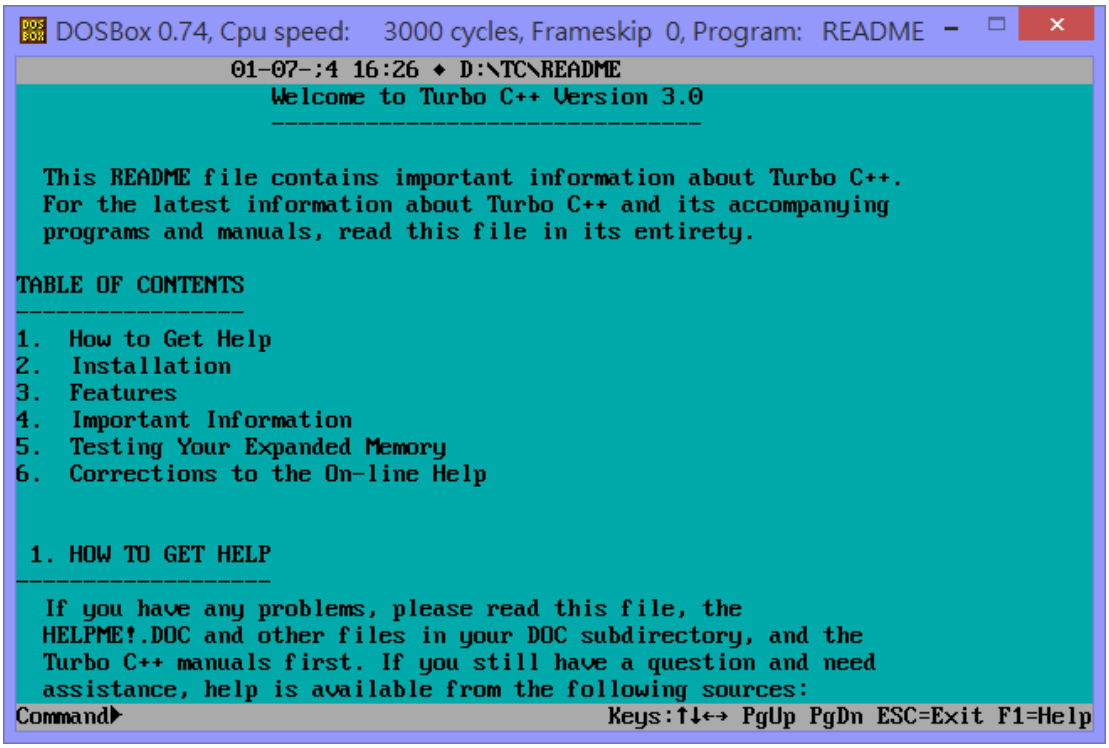
下面是 TC++3.0 安装步骤图解



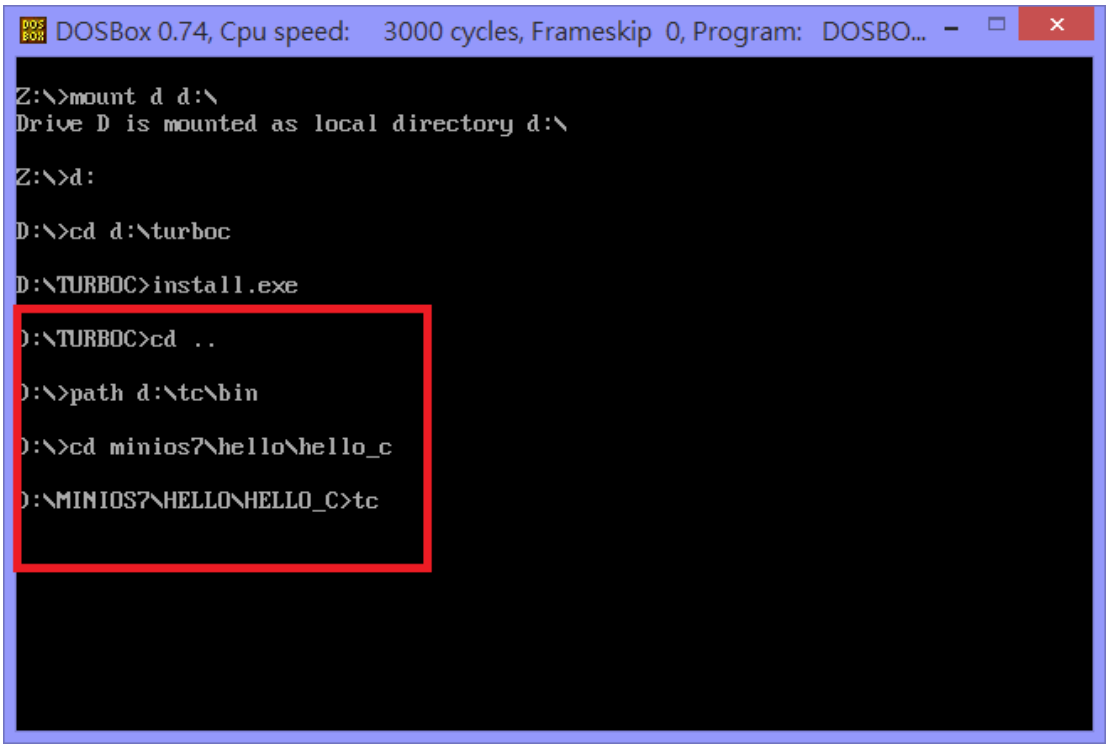


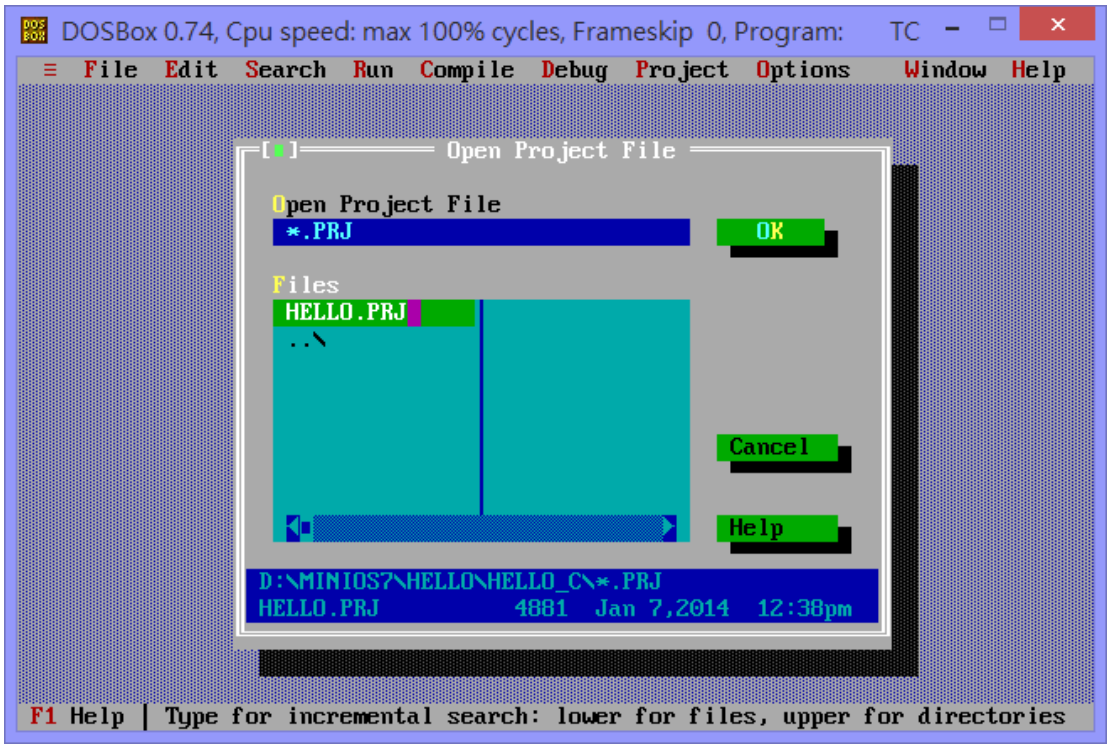

DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: INSTALL
Turbo C++ 3.0 Installation Utility
Enter the SOURCE Path
\TURBOC
Description
Enter the path to the directory containing the Turbo C++ files.
ENTER-Select ESC-Cancel


DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: INSTALL
Turbo C++ 3.0 Installation Utility
Directories... [D:\TC]
Options... [IDE CMD LIB CLASS BGI HELP EXMPL]
Start Installation
Description
Selecting this option will begin copying files to your hard drive into the directories specified above.
F1-Help F9-Start the installation ENTER-Select ESC-Previous



安装完 TC++3.0 之后，设定编译器路径和切换到 Demo 所在的数据夹下 ➔ 执行 TC 编译器.




DOSBox 0.74, Cpu speed: max 100% cycles, Frameskip 0, Program: TC
File Edit Search Run Compile Debug Project Options Window Help
Open Project File
Open Project File
*.PRJ
OK
Files
HELLO.PRJ
..\
Cancel
Help
D:\MINIOS7\HELLO\HELLO_C*.PRJ
HELLO.PRJ 4881 Jan 7,2014 12:38pm
F1 Help | Type for incremental search: lower for files, upper for directories

附录 F: 术语说明

1. 64 位唯一硬件序列号:

I-7188XB(D)具有一个 64 位唯一硬件序列号。这个数字是唯一的，并且不会与其它的 I-7188XB(D)控制器重号。应用软件可使用这个数字来验证硬件的有效性和软件的合法性。它是当前最低成本的 I-7188XB(D)保护装置。

2. AsicKey:

I/O 拓展总线支持 AsicKey。AsicKey 具有一个复杂的机制来完成有效性检查。为实现同样的目的其中还包括 128 字节的专用数据。它为软件的违法复制提供非常强的保护。每个合法用户具有一个唯一的和唯一的软件库，用户可自己检查这个 key，或者通过软件自动进行。总的来说，不可能移除 AsicKey 保护。

附录 G: 文件修订纪录

版本	出版日期	修正内容
1.0	4 月 2007	第一次出版
1.1	2 月 2012	1. 修改 DI 规范 2. 修改储存温度规格： 原：-40° C 至+80° C 新：-30° C 至+80° C 3. 修改湿度规格： 原：0～90％ 新：10～90％RH（无冷凝）
1.2	7 月 2013	增加章节 1.4.8 “使用跳线帽启动 RS-485 pull-high/pull-low 电阻”
1.3	2 月 2014	1. 新增章节 3.4 “在 64 位平台建立项目(Project)” 2. 新增章节”在 64 位平台编译”于附录 E:编译和链接 3. 新增章节”附录 G: 版本纪录”