



HT-IDE3000 使用手册

版本 : V1.30 日期 :2022-05-09

www.holtek.com

目录

第一章 概要与安装	4
IDE 整合开发环境.....	4
微控制器仿真器 (ICE & e-ICE & e-Link).....	5
系统配置	5
安装	6
第二章 快速开始	11
第三章 选单 – 档案 / 编辑 / 检视 / 工具 / 选项	14
启动 HT-IDE3000 系统	14
档案选单	16
编辑选单	16
检视选单	17
工具选单	18
选项选单	30
书签	41
第四章 选单 – 专案 & 建制	44
建立新项目	44
开启和关闭专案	47
管理项目的原始档	47
建成项目的工作档	48
组译 / 编译	49
确认单查看器	49
备份 / 恢复工程	50
第五章 选单 – 除错	51
重置 HT-IDE3000 系统	51
应用程序的硬件仿真	53
单步执行	54
断点	55
追踪应用程序	58
第六章 选单 – 窗口	65
窗口选单命令	65
第七章 汇编语言和组译器	72
惯用符号	72
语句语法	72
组译假指令	73
汇编语言指令	82
其它	84
组译器选项	87
编译列表档案格式	88
第八章 连结器	91
连结器之功用	91
连结器的选项	91

地址对映图档案	93
连结器的工作档案与除错档案	94
第九章 函式库总管	95
函式库总管的功能	95
设定函式库档案	95
第十章 组译器使用的保留字	98
汇编语言保留字	98
指令集	99
第十一章 LCD 仿真器	102
简介	102
LCD 面板配置档案	102
LCD 面板图案文件	103
建立 LCD 面板配置档案	103
LCD 模拟	109

第一章 概要与安装

为简化应用程序的发展过程，支持工具的重要性和有效性对于微控制器来说是不可低估的。为了支持所有系列的 MCU，本公司用心的提供具有完整功能的工具，让用户在开发与使用上更加便利，例如众所周知的 IDE 整合开发环境，软件方面具有 HT-IDE3000 软件，提供友善的窗口接口以便从事程序的编辑及除错，同时硬件方面为 ICE 仿真器，提供多种实时仿真功能，包含多功能追踪、单步执行和设定断点功能。IDE 发展系统提供完整系列的适配卡与定期软件服务包装的更新，因此保证设计者可以有较佳的工具，且能以较高效率进行微控制器应用程序的设计与开发。

IDE 整合开发环境

IDE (Integrated Development Environment) 是一个具有高效能，使用于设计本公司 8 位 MCU 应用程序的整合开发环境。系统中的硬件及软件工具能够帮助客户使用本公司 8 位 MCU 芯片，快速且容易地发展应用程序。在 IDE 整合开发环境中最主要的组件为 e-ICE 和 ICE (In-Circuit Emulator)，它们提供了本公司 8 位微控制器的实时仿真功能以及强而有力的除错与追踪功能。

在软件方面，HT-IDE3000 发展系统提供友善的工作平台。将所有的软件工具，例如编辑器、组译器、连结器、函式库管理员和符号除错器，整合成为一窗口环境，使程序发展过程更为容易。

HT-IDE3000 使用手册中包含 IDE 发展系统的相关细节。为了提供 HT-IDE3000 的安装作业以及确保发展系统包含有最新的微控制器和软件更新信息，本公司也定期提供 HT-IDE3000 服务软件 (Service Pack)。这些服务软件不是用来取代 HT-IDE3000，它必须要在 HT-IDE3000 系统软件已安装后才能被安装。

由 HT-IDE3000 所具有的特色如下：

模拟

程序指令的实时仿真

硬件

- ICE
 - ◆ 使用及安装容易
 - ◆ 可使用内部或外部振荡器
 - ◆ 断点机制
 - ◆ 支持追踪功能与触发能力的追踪仿真器
 - ◆ ICE 经由打印口与计算机连接
 - ◆ 使用者的应用电路板经由 I/O 适配卡连接至 ICE
- e-ICE
 - ◆ 使用及安装容易
 - ◆ 可使用内部或外部振荡器
 - ◆ 断点机制
 - ◆ e-ICE 经由 USB 与计算机连接
 - ◆ 使用者的应用电路板经由 2.54mm 标准排针连接至 e-ICE
- e-Link
 - ◆ EV 采用 OCDS(On-Chip Debug Support) 架构，只需 2 条信号线便可进行调试
 - ◆ EV 与 IC 脚数相同或比 IC 多 1~2 根引脚，可焊接在应用板，方便调试
 - ◆ 提供多种形式的断点功能
 - ◆ 提供 RAM 实时监控功能
 - ◆ 工作电压范围更宽 1.7V~5.5V

软件

- ◆ 窗口架构的软件工具
- ◆ 原始程序层次的除错器 (符号除号器)
- ◆ 支持多个原始程序文件的工作平台 (一个应用项目可以包含一个以上的原始程序文件)
- ◆ 所有工具被使用于开发、除错、评估和产生最终的应用程序代码 (Mask ROM file)
- ◆ 可将公用程序建立成函式库并在之后连结到其它的项目中

微控制器仿真器 (ICE & e-ICE & e-Link)

对于本公司的 8 位微控制器而言, 本公司的 ICE 是全功能的仿真器, 系统中的硬件及软件工具能帮助客户快速且容易的发展应用程序。系统中最主要的是硬件仿真器, 除了能够有效地提供除错和追踪功能之外, 还能以实时的方式进行本公司 8 位 MCU 的仿真工作。在软件方面, HT-IDE3000 发展系统提供友善的工作平台, 将所有软件工具, 例如编辑器、组译器、连结器、函式库管理员和符号除错器, 合并到窗口环境。

ICE 适配卡

伴随 ICE 的适配卡 (图 1-1) 可以被许多的应用电路使用, 但是使用者也可自行设计适配卡。将必要的接口电路放在他们自己的适配卡, 使用者可以直接把他们的应用电路板连接到 ICE 的 CN1 和 CN2 连接器。

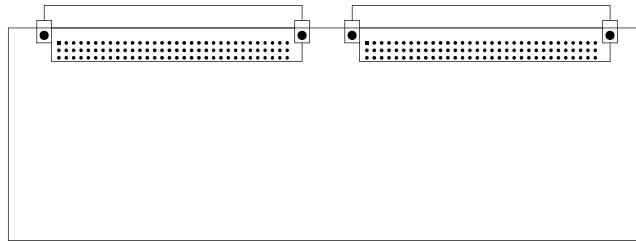


图 1-1

系统配置

IDE 整合环境硬件配置如图 1-2, 主机需为 Pentium 兼容机器, 系统为 Windows XP 或以后版本, 注意在 Windows XP 或新版系统下安装 HT-IDE3000 时, 需要在 Supervisor Privilege 模式下执行 HT-IDE3000 软件安装。

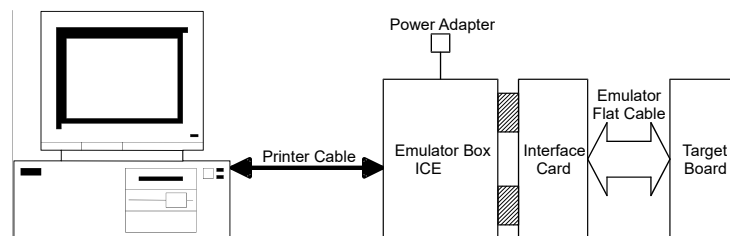


图 1-2

IDE 整合环境包含下列硬件成员：

ICE

- ◆ ICE 机台，包含印刷电路板 (PCB)，其中有一个打印机端口的连接头，可与个人计算机连接，输入 / 输出讯号接头及 LED 电源指示灯。
- ◆ 连接目标电路板与 ICE 机台的 I/O 适配卡
- ◆ 变压器 (输出 16V)
- ◆ D 型 25 根引脚的打印机缆线
- ◆ 若干排线
- ◆ HT-IDE3000 软体光盘

e-ICE

- ◆ e-ICE 机台基本上包含了一个 MEV 母板和一个可插拔替换的 DEV 子板。
- ◆ 5 根引脚的 USB Mini B 型缆线

e-Link

- ◆ e-Link 台机
- ◆ 2×6 双排排针
- ◆ Flat-Cable 支头 2×6 排针母接头 (长 25cm)
- ◆ e-FPC06A
- ◆ USB 数据线
- ◆ 重要提示卡
- ◆ 软体光盘

安装

系统需求

安装 HT-IDE3000 系统的硬件及软件需求如下：

- ◆ Microsoft Windows 操作系统 (32 位或者 64 位)
- ◆ 至少 1G 以上的 RAM
- ◆ 至少 500M 以上的硬盘空间
- ◆ 具有平行口或 USB 口，可连接 PC 和 ICE

硬件安装

本公司推出三种 ICE 供使用者选择，种类如下：

ICE

- 步骤 1
第一次使用必须另行安装驱动
- 步骤 2
将电源变压器插入 ICE 的电源插孔
- 步骤 3
藉由 I/O 适配卡或扁平缆线连接目标电路板与 ICE
- 步骤 4
使用打印机缆线连接 ICE 与主机
此时 ICE 上的 LED 应该是亮的，如果不是，则重新操作连接的程序或与代理商联系。

警告：请小心使用电源变压器，勿使用输出不是 16V 的变压器，否则可能导致 ICE 损坏，因此强烈建议使用由本公司所提供的变压器。首先将电源变压器插入 ICE 的电源插座。

e-ICE

- 步骤 1
MEV 板和 DEV 板连接
- 步骤 2
使用 USB 缆线连接 e-ICE 与主机
此时 e-ICE 上的 LED 应该是亮的，如果不是，则重新操作连接的程序或与代理商联系。

e-Link

- 步骤 1
e-Link 与 EV 板通过排线连接
- 步骤 2
使用 USB 缆线连接 e-Link 与主机
此时 e-Link 上的 LED 应该是亮的，如果不是，则重新操作连接的程序或与代理商联系。

注意：要使用 e-Link 进行调试，请确保 HT-IDE3000 版本至少为 7.6。

软件安装

- 步骤 1
首先点击 HT-IDE3000 安装程序并开始安装 HT-IDE3000
- 步骤 2
按下 <Next> 按钮 (继续安装) 或按下 <Cancel> 按钮 (中止安装)。

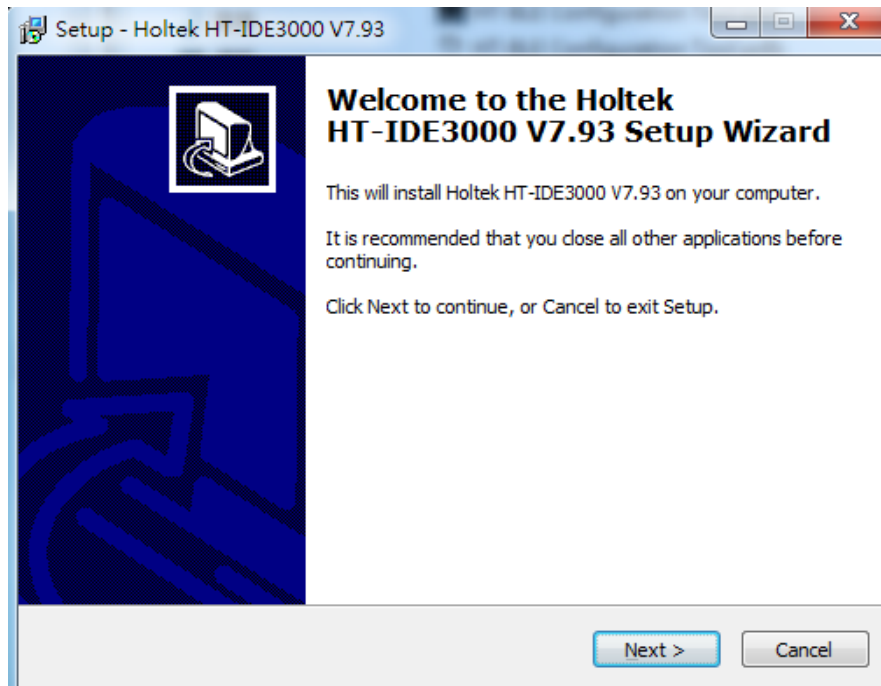


图 1-6

- 步骤 3
下面显示的对话框会要求使用者输入安装处的数据夹名称。

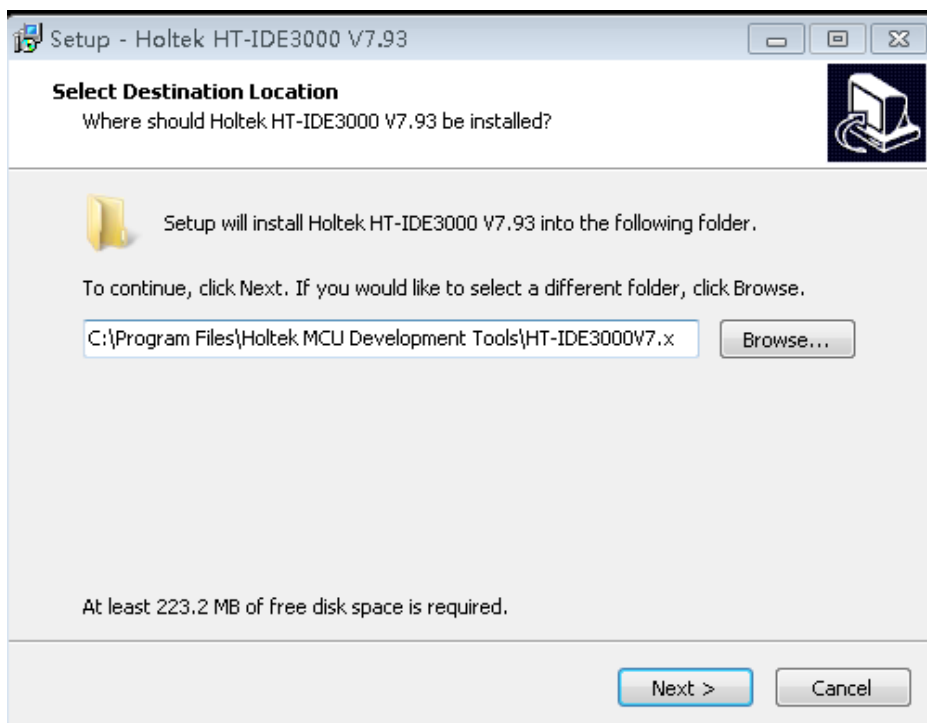


图 1-7

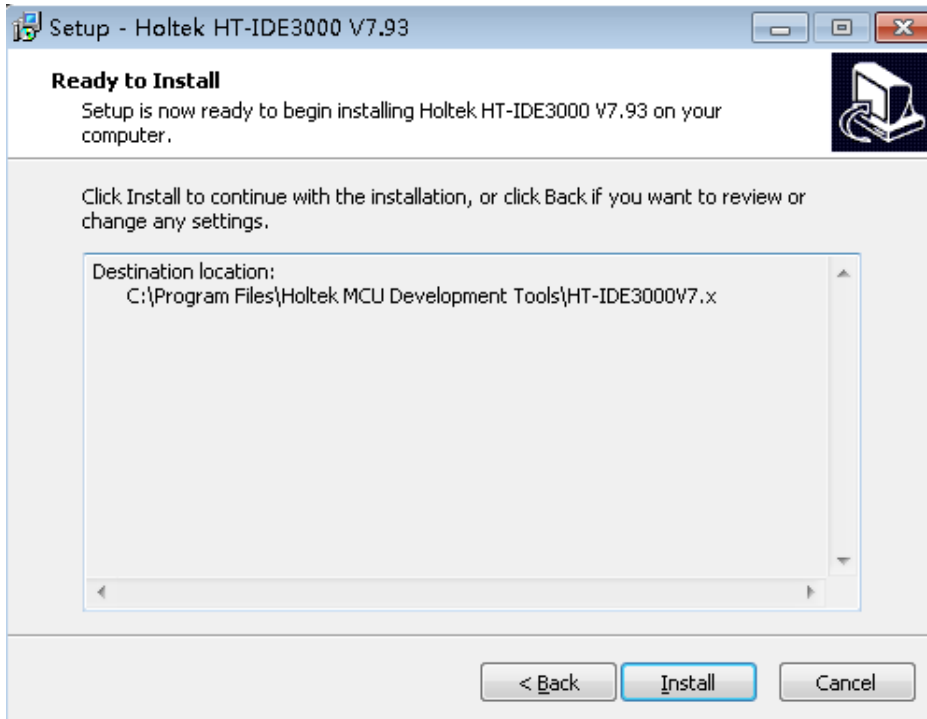


图 1-8

- 步骤 4
指定你希望安装 HT-IDE3000 的数据夹路径，然后按下 <Next> 按钮。
- 步骤 5
SETUP 会将档案复制到你指定的数据夹。

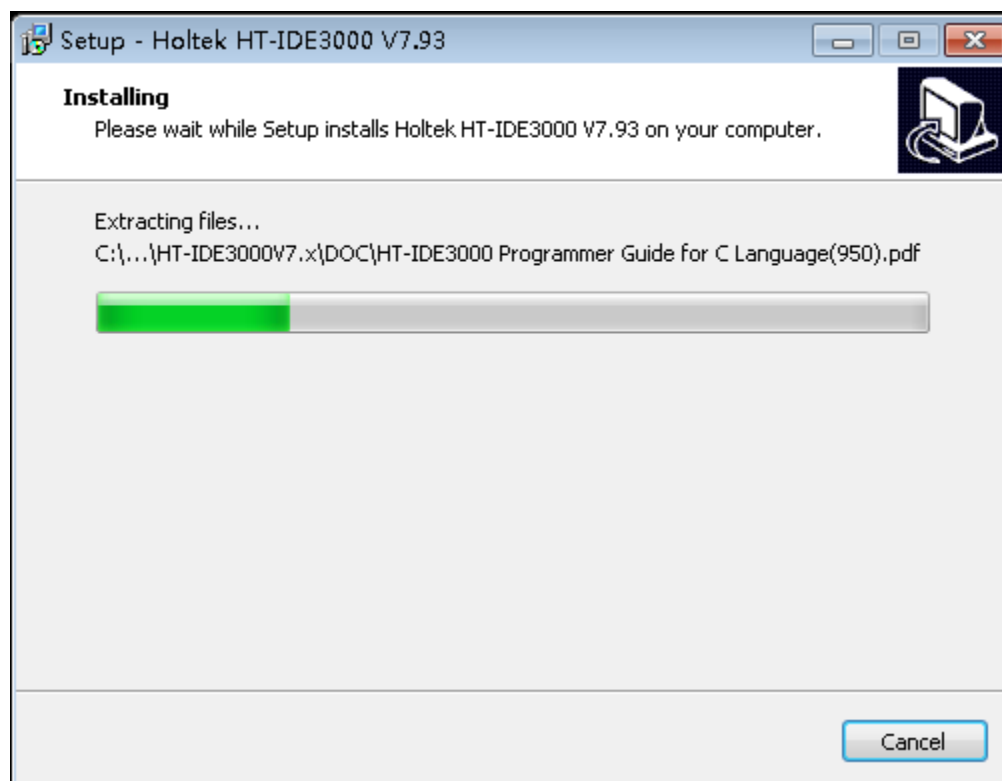


图 1-9

- 步骤 6
安装成功的话，则会出现下面的提示对话框。

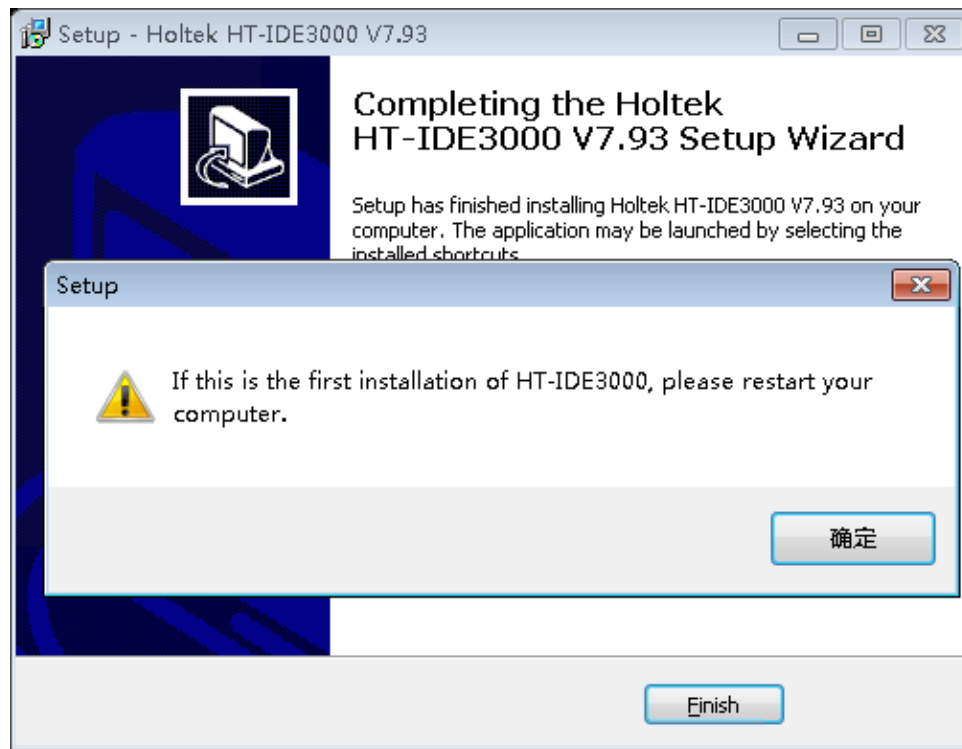


图 1-10

- 步骤 7
请按下确定，然后点击 <Finish> 按钮完成安装，如果是第一次安装 HT-IDE3000，需要重新启动计算机。

第二章 快速开始

本章简述如何快速使用 HT-IDE3000 去开发一个应用程序项目。

步骤一：使用 CodeWizard 建立一个新项目

- 按下工程选单并选择新建命令
- 输入项目名称以及从 combo 框选择此项目使用的微控制器型号
- 选择 .ASM 或 .C 为开发程序语言
- 按下 OK 按键则系统将会要求设定微控制器的配置选项
- 最后，完成项目的设定选项后按下 OK 按键

步骤二：建制专案

- 按下编译选单并且选择构建命令
- 系统将会对项目中的所有原始档执行编译动作
 - ◆ 如果程序中有错误，只要在错误讯息行连接两次，则系统将会提示错误发生的位置并且开启此错误所在的原始程序文件，便可直接修改程序与储存档案
 - ◆ 如果所有程序文件皆没有错误，则系统会产生一个执行档并且加载到 ICE 中，准备模拟及除错
- 你可以重复上述的步骤直到没有错误

步骤三：烧录微控制器

- 建立微控制器的项目并且产出烧录档案
- 请使用本公司通用型烧录器 e-WriterPro 搭配 HOPE3000 来烧录 MCU。

步骤四：传送程序与配置选单至本公司

- 按下工程选单以及选用选项表查看器命令，出现图 2-1 所示界面，点击 Print 打印配置选项确认单
- 传送 .COD/.OTP/.MTP 档案和配置选项确认单到本公司，进行生产

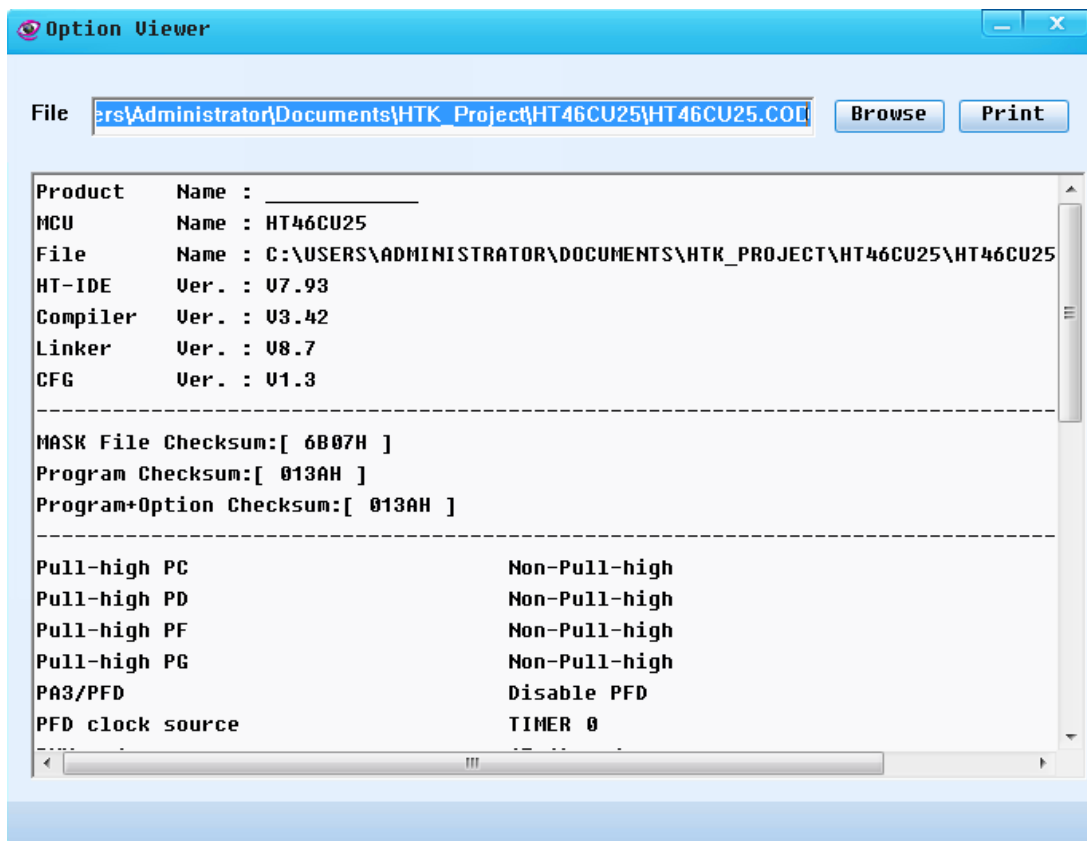


图 2-1

程序及数据流可以下图来表示：

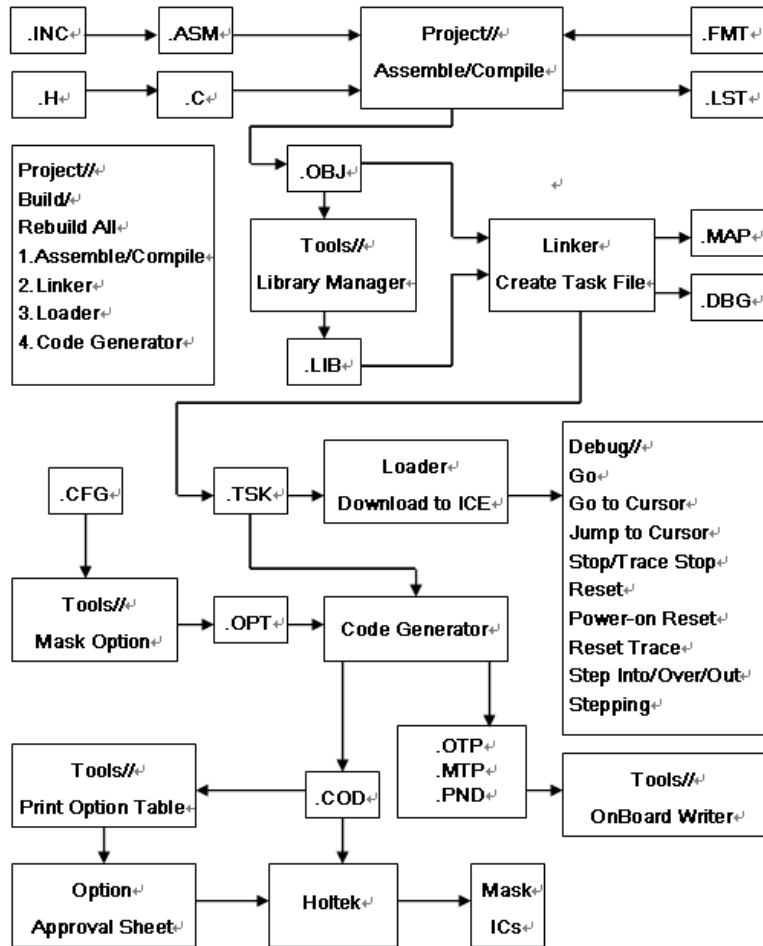


图 2-2

第三章 选单 – 档案 / 编辑 / 检视 / 工具 / 选项

本章描述 HT-IDE3000 的部分选单和命令，其它的选单则在项目、除错及窗口等章节中叙述。

启动 HT-IDE3000 系统

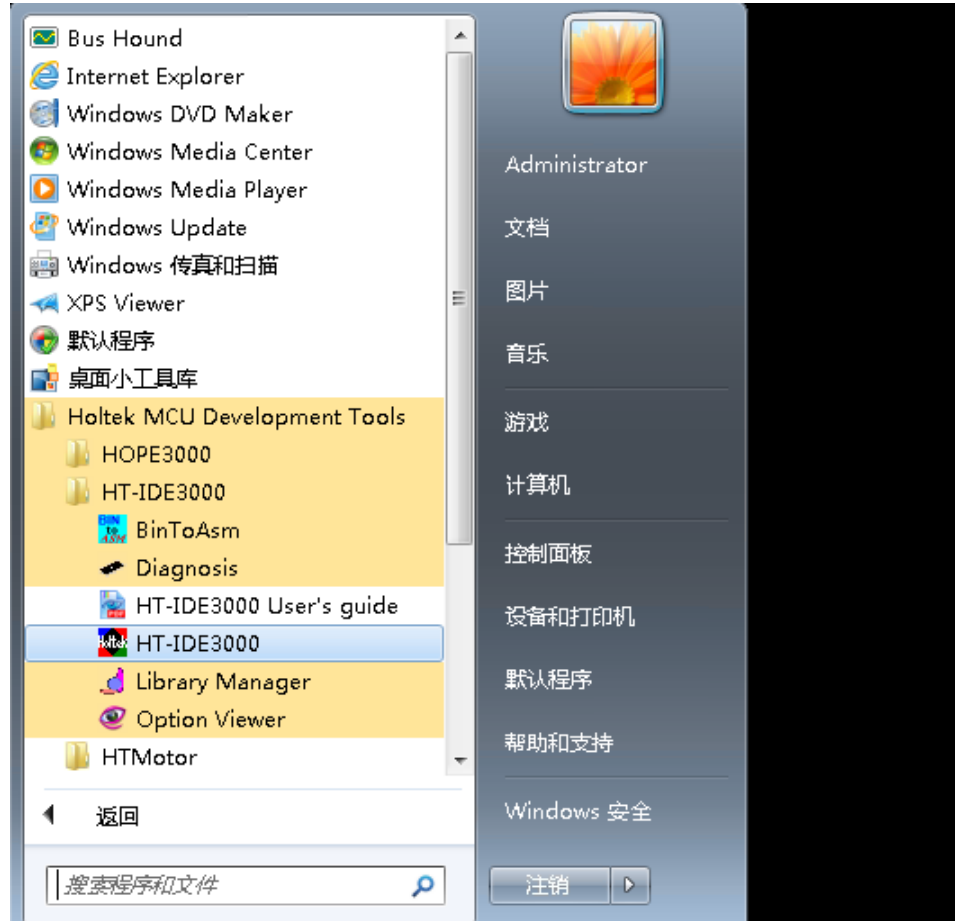


图 3-1

- 按下 [开始] 按钮，选择 [程序集] 和选择 HT-IDE3000
 - ◆ 按下 HT-IDE3000 的小图案
- 当下面的条件发生时，将会显示图 3-2 的画面。
 - ◆ 计算机与 ICE 之间没有连接或连接失败。
 - ◆ ICE 电源关闭。



图 3-2

如果选择 重试并且主计算机与 ICE 已经连接，则显示如图 3-3 的画面，HT-IDE3000 进入仿真模式，ICE 开始动作。

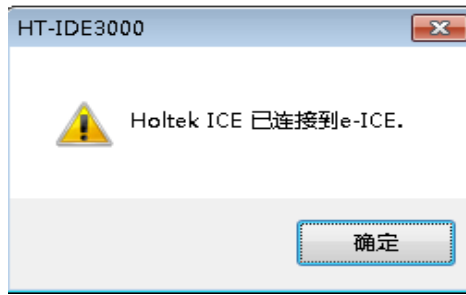


图 3-3

HT-IDE3000 提供十种选单—文件 (File)、编辑 (Edit)、视图 (View)、工程 (Project)、编译 (Build)、调试 (Debug)、工具 (Tools)、选项 (Options)、窗口 (Window) 及帮助 (Help)，下面的章节会描述每一个选单的命令及功能。

在选单列下方的是工具列 (如图 3-4 所示)，可以随处移动此工具列的位置。其中所包含的小图示，皆是经常使用的选单命令，可让使用者更方便去执行选单命令。将光标放置在工具列的任何一个小图示时，相对应的命令名称会显示在旁边，按下小图示就会执行此命令。

状态列位于底线处 (如图 3-4 所示)，显示硬件仿真的现在状态以及命令的执行结果状态。

当执行调试时 (调试选单)，状态列中的字段 (PC=0001H) 会显示程序计数器的内容。

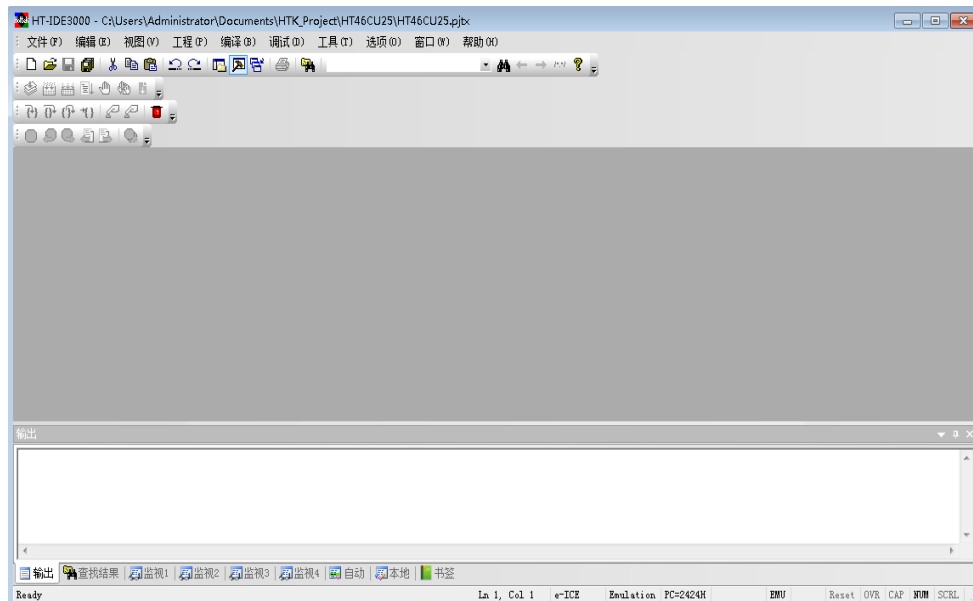


图 3-4

状态列所显示的信息可能在程序除错阶段有所帮助，例如程序计数器在程序执行期间会显示实际程序计数器的内容，而在使用程序编辑器时，列与行的指示器 (状态列最右边的两个字段) 会显示光标目前的位置。

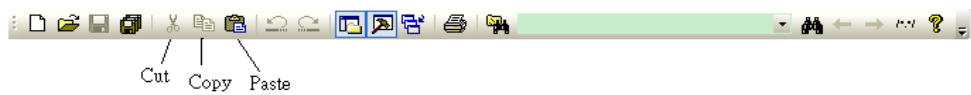
档案选单



档案选单提供档案处理命令，对应的工具列小图标显示如上，细节说明如下：

- **New**
建立新档案
- **Open**
开启旧档案
- **Close**
关闭动作窗口内的档案
- **Save**
储存动作窗口内的档案
- **SaveAs**
将动作窗口的档案另存成新档名
- **Save All**
储存所有被开启的档案
- **Print**
将动作窗口的数据印出到打印机
- **Print Setup**
设定打印机
- **PrintPreview**
预览打印效果
- **Recent Files**
列出最近所开启和关闭的四个档案
- **Exit**
从 HT-IDE3000 离开且回到 Windows 窗口

编辑选单



- **Undo**
复原前次的编辑动作
- **Redo**
取消复原的编辑动作
- **Cut**
将所选定的内容从档案中移除并且将之存放到剪贴板 (clipboard)
- **Copy**
将所选定的内容复制到剪贴板
- **Paste**
将剪贴板中的内容贴到当前的插入点

- Delete
删除所选定的内容
- Select All
选择整份文件内容
- Find
在编辑数据储存区中寻找特定字符
- Find Next
搜寻下一个出现的指定字符串
- Find Previous
搜寻前一个出现的指定字符串
- Find in Files
从复数档案搜寻指定字符串
- Replace
将编辑数据储存区中指定的字符用特定字符取代
- Go To...
移动到指定位置
- Read Only
取消档案的只读设定

检视选单

检视选单提供下列命令去控制 HT-IDE3000 的窗口 (参考图 3-5)

- Full Screen
全屏幕模式切换
- Toolbar
此命令可将工具列的信息显示在窗口上或从窗口上取消, 其中包含一些按钮群组, 它的命令功能相同于每个对应的选单项目。当鼠标光标放置在工具列按钮上时, 相对应的功能名称将显示在按钮之旁, 若按下鼠标按钮则开始执行此命令, 每一个按钮的功能请参考相对应的章节。Toggle Breakpoint 按钮可将光标位置所在的行设为断点 (明亮标示), 再次按这个按钮则是取消已经设定的断点。
- Status Bar
打勾表示在窗口上显示状态列信息。
- Application Look
应用程序的外观风格。可以选择的风格有: Office XP、Windows XP、Office 2003、Visual Studio .Net 2005、Office 2007(包含蓝、黑、银、水绿四种), 默认为 Visual Studio .Net 2005。
- Restore Defalult LayOut
还原预设布局, 此命令会自动重启程序。
- Display Line Numbers
行数显示切换
- Cycle Count
累计指令的执行周期数。按下重置按钮可清除周期计数, 使用基数按钮的十六进制及十进制改变计数的基数为十六进制或十进制, 最大计数周期为 4294967295。

注意：新旧版 ICE 最大计数周期略有不同；e-ICE 可支援到 4294967295；但 ICE 只支援到 65535。

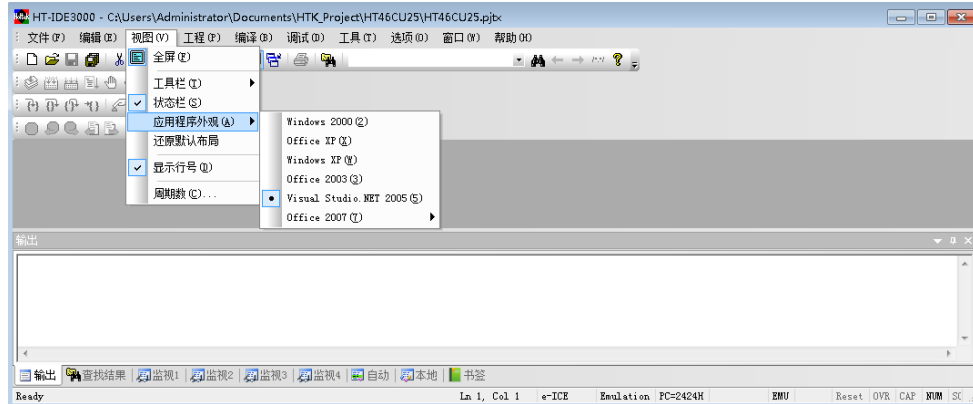


图 3-5

工具选单

工具选单提供特别的命令帮助使用者程序除错，这些命令为配置选项、导入配置选项、系统诊断、函数库管理器、程序位置设定、编辑器、LCD 软件模拟器、Voice & Flash 下载及 OCDS 切换。

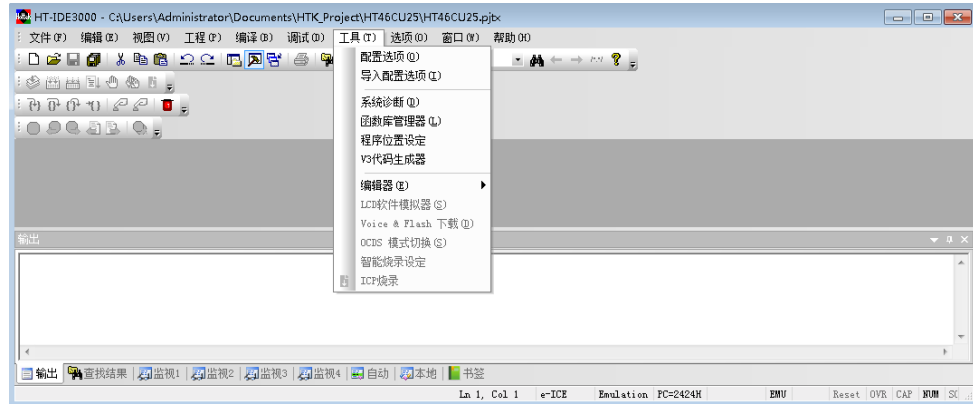


图 3-6

配置选项

在新建立一个项目时 (请参考第四章)，系统会根据使用者所选定的微控制器以及外围配置，产出一个配置选项文件。当使用编译选单的构建命令完成此项目的过程中，系统需要此配置选项文件以便产出执行档。在项目建立后，可以使用配置选项命令去修改微控制器的外围配置，之后必须重新使用构建命令完成项目，否则方才的修改无效。

选择时钟源

当建立一个新项目或是修改配置选项文件时，必须要设定 ICE 所提供的内部或外部两种时钟源。如果使用内部的时钟源，则必须指定 ICE 的系统频率。HT-IDE3000 软件系统将会估算一个 ICE 能够支持而且最近似于所指定的系统频率值。当估算频率不等于指定的频率时，将会显示警告讯息以及指定的频率值与估算的频率值。客户必须确认要使用估算的频率或指定另一个系统频率，否则

就只有使用外部的时钟源。无论选用何种时钟源，必须要指定系统频率。

注意：e-ICE 内部及外部时钟源设定，请参考 e-ICE 用户说明文件。

导入配置选项

导入配置选项文件。

系统诊断

诊断命令将验证 9 个项目，目的是帮助使用者检查 ICE 是否可以正常工作 (图 3-7)。选取检查方块并按下 Test 键可以选择验证数个项目，也可按下 Test All 键去诊断所有项目，这些项目如下所列。

- 单片机资源选项空间
诊断 ICE 的微控制器配置选项空间
- 程序代码存储器空间
诊断 ICE 的程序代码内存
- 跟踪记录存储器空间
诊断 ICE 的追踪内存
- 数据存储器空间
诊断 ICE 的程序数据存储器
- 系统空间
诊断 ICE 的系统数据存储器
- I/O 仿真器 0
诊断 ICE 在插槽 0 的 I/O 仿真器
- I/O 仿真器 1
诊断 ICE 在插槽 1 的 I/O 仿真器
- I/O 仿真器 2
诊断 ICE 在插槽 2 的 I/O 仿真器

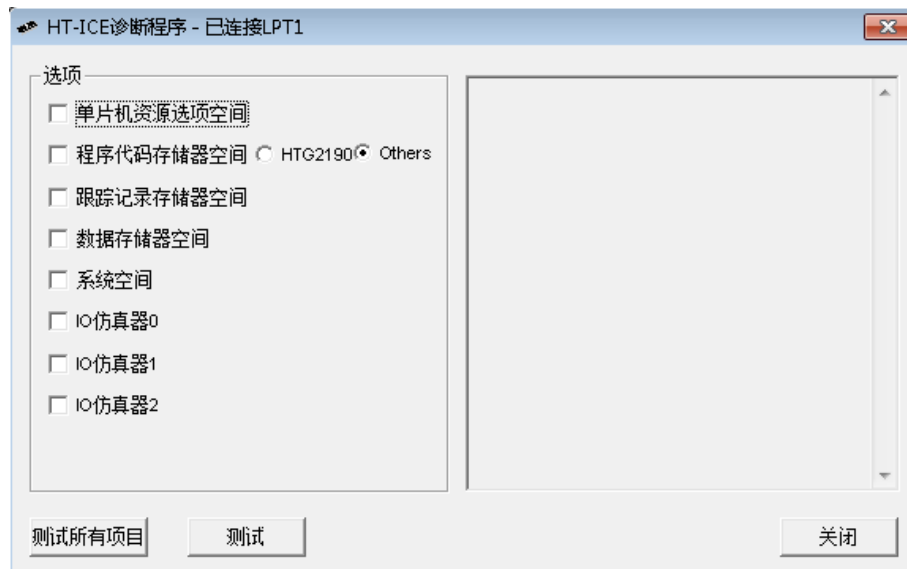


图 3-7

函数库管理器

在图 3-8 中函数库管理器命令支持库功能。经常使用的程序代码可以结合成函式档，然后经由选项选单中的工程选项命令将之引用于应用程序内 (请参考选项选单中的工程选项命令的连接选项)。函数库管理器的功能如下：

- 建立新的函式库档案或修改函式库档案。
- 将程序模块加到函式库档案或从函式库档案删除程序模块。
- 从函式库档案中取出程序模块然后建立一个目的档。

有关函数库管理器更详细的资料，会在第三部份说明。

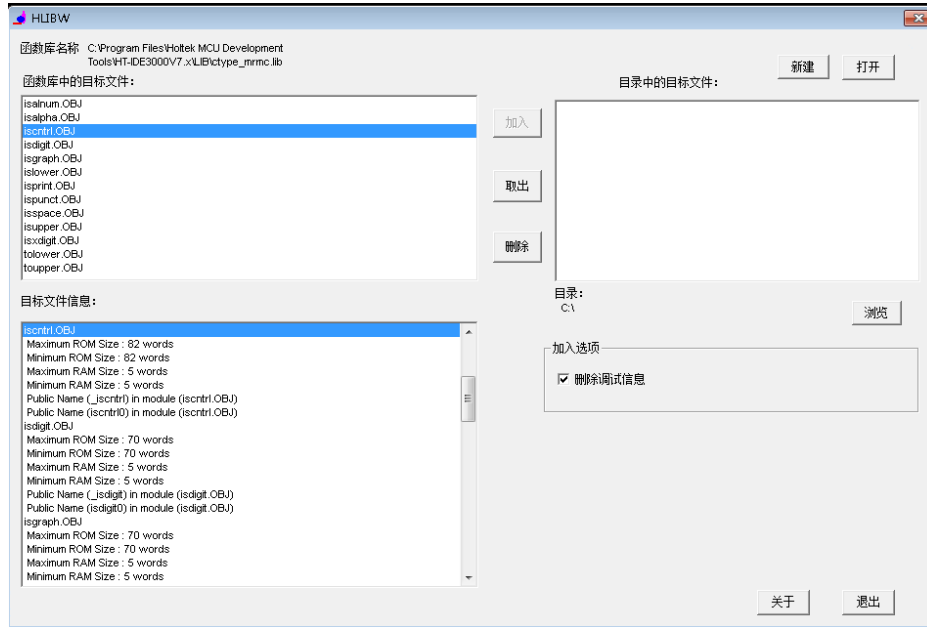


图 3-8

程式位置设定

可以设置工程的各个代码模块在 ROM 中的排列位置，如图 3-10。



图 3-10

- 类型
代码模块类型，可以对 C、ASM、OBJ、LIB 进行设置。
- 设置
选择列表框的模块 (可以用 Shift 键或者 Ctrl 键进行多选)，然后单击 Set 按钮进行地址设定，弹出如下对话框：



图 3-11

起始地址跟结束地址必须以 16 进制表示。
如果想取消某个模块的设定，只要将起始地址与结束地址删除然后单击 ok 即可。

- 重置所有
清空所有模块的设置。
除了从菜单进行设定外，还可以从工作空间 (Work Space) 进行设置。

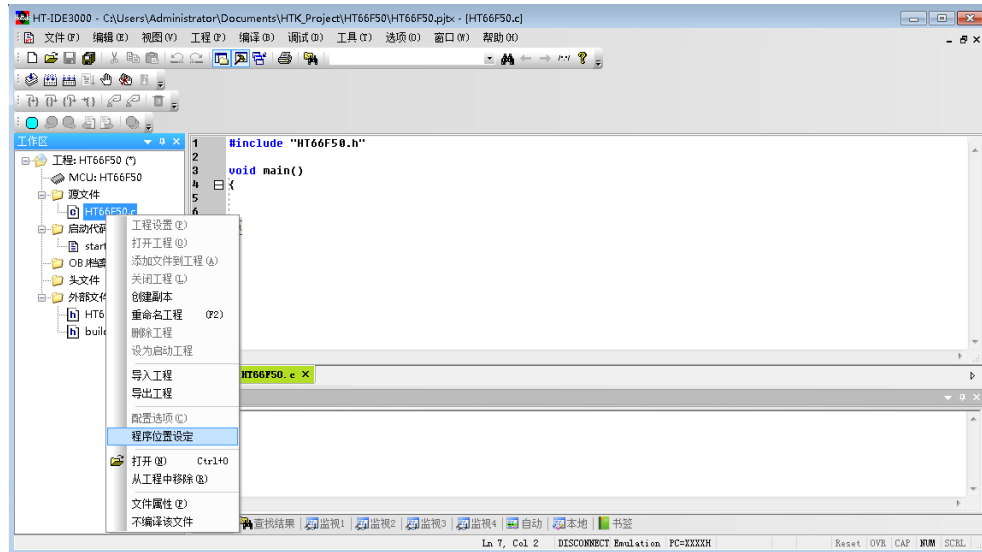
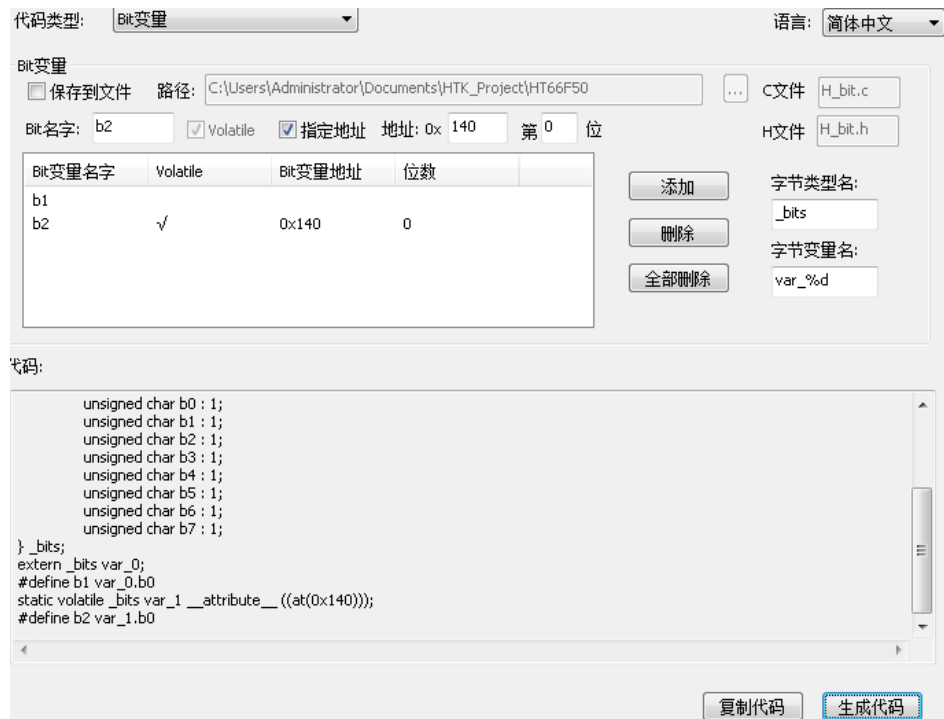


图 3-12

V3 代码生成器

V3 代码生成器可以帮助用户生成 V3 特定语法的代码，如图：

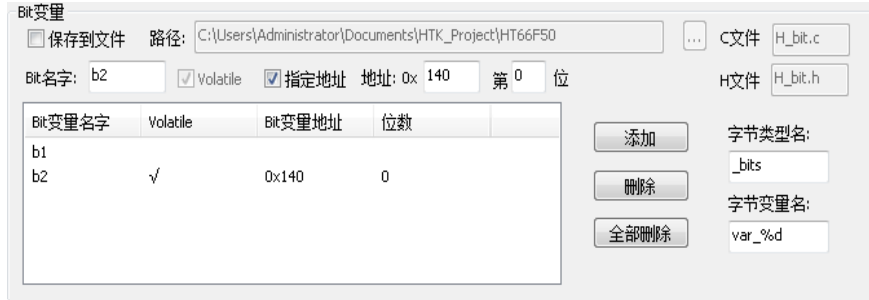


● 代码类型

生成代码的类型：

1) Bit 变量

生成 Bit 变量，如图：



其中：

保存到文件：可以将生成的代码保存到指定的文件中

路径：文件所在的路径

C 文件和 H 文件：要保存的 C 文件和 H 文件的文件名

字节类型名和字节变量名：指定 Byte 类型名和变量名的格式

Volatile：将 bit 变量指定为 volatile

指定地址：勾选可以指定 bit 变量在哪个地址的第几位

添加：将 bit 变量添加到列表中

删除：从列表中删除选择的 bit 变量

全部删除：删除列表中所有的 bit 变量

双击 bit 变量列表可以对已添加的 bit 变量进行编辑

2) ISR (中断函数)

生成中断函数，如图：



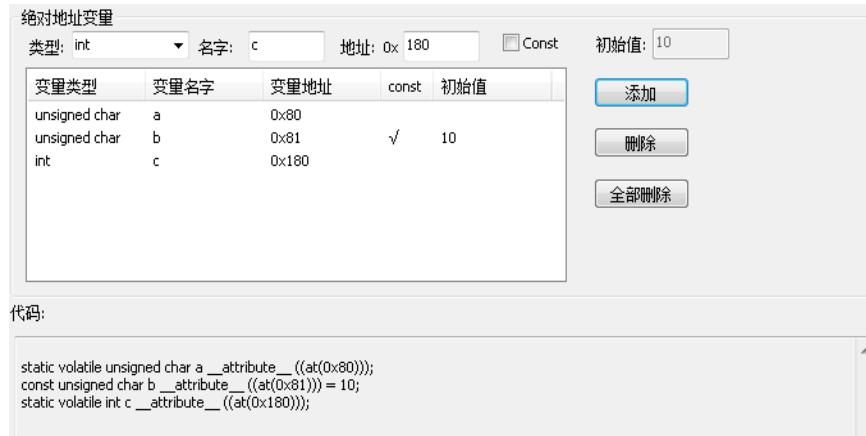
其中：

中断名字为中断函数名，中断向量为中断向量的值，中段地址为指定中断函数的地址。

双击中断函数列表可以对已添加的中断函数进行编辑。

3) 绝对地址变量

生成绝对地址变量，如图：



绝对地址变量

类型: int 名字: c 地址: 0x 180 Const 初始值: 10

变量类型	变量名字	变量地址	const	初始值
unsigned char	a	0x80		
unsigned char	b	0x81	√	10
int	c	0x180		

代码:

```
static volatile unsigned char a __attribute__((at(0x80)));
const unsigned char b __attribute__((at(0x81))) = 10;
static volatile int c __attribute__((at(0x180)));
```

其中：

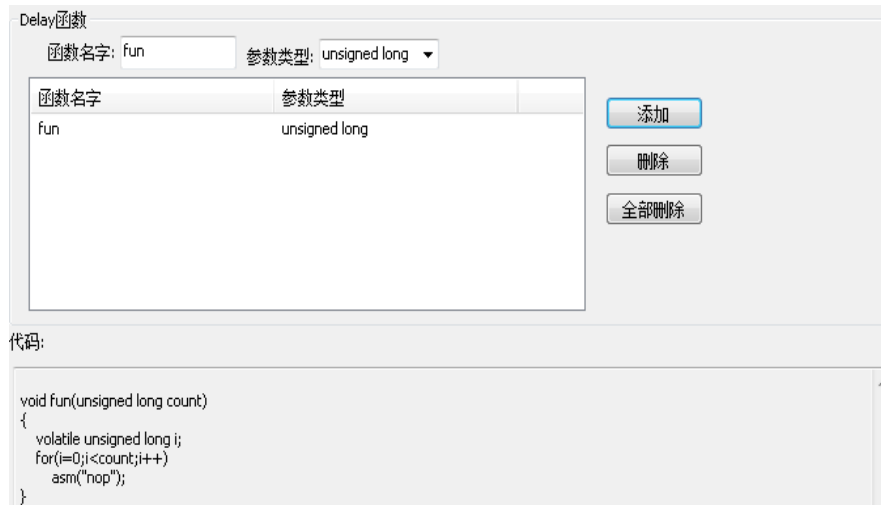
类型、名字和地址分别是变量的类型、名字和地址，变量名可以是数组。

Const 表示该变量是否是 const 变量，const 变量必须要有初始值。

双击变量列表可以对已添加的变量进行编辑。

4) Delay 函数

生成 Delay 函数，如图：



Delay函数

函数名字: fun 参数类型: unsigned long

函数名字	参数类型
fun	unsigned long

代码:

```
void fun(unsigned long count)
{
    volatile unsigned long i;
    for(i=0;i<count;i++)
        asm("nop");
}
```

其中：

函数名字为 Delay 函数的名字，参数类型为函数参数的类型。

Delay 函数内的变量应定义成 volatile，否则循环会被 V3 编译器优化。

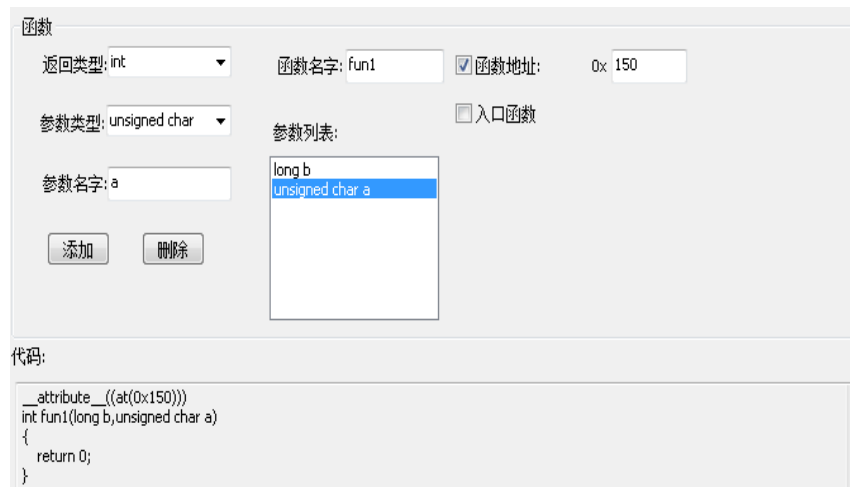
5) 内嵌汇编

生成内嵌汇编，输入汇编指令，将汇编指令转换为 V3 格式的内嵌汇编，如图：



6) 函数

生成函数，可以设定返回类型、参数、指定地址和是否是入口函数等，如图：



其中：

返回类型和函数名字：函数的返回类型和函数名

参数列表：函数的参数列表

函数地址：指定函数的地址

入口函数：勾选表示该函数是入口函数

添加：添加参数到函数的参数列表中

删除：删除参数列表中选择的参数

7) 指定范围变量

生成指定范围变量，根据变量的长度自动分配在指定的范围中，如图：

指定范围变量

指定范围: 0x 0180 ~ 0x FFFF

类型: double 类型长度: 4 字节 名字: c

变量类型	类型长度	变量名字
char*	2	pa
short	2	arr[10]
float	3	b[2]
double	4	c

代码:

```
static volatile char* pa __attribute__((at(0x180)));
static volatile short arr[10] __attribute__((at(0x182)));
static volatile float b[2] __attribute__((at(0x196)));
static volatile double c __attribute__((at(0x19C)));
```

其中：

指定范围：指定的 RAM 范围

类型和类型长度：变量的类型和类型的长度，如果类型是基本类型或指针，则类型长度不可改变。

名字是变量的名字，变量名可以是数组。

双击变量列表可以对已添加的变量进行编辑。

8) 指定 Bank 变量

生成指定 Bank 变量，根据变量的长度自动分配在指定的 Bank 中，如图：

指定Bank变量

MCU: HT66F50

类型: int 类型长度: 2 字节 名字: e Bank: 2 0x280~0x2FF

变量类型	类型长度	变量名字	Bank
char	1	a	0
short	2	b	1
float	3	c[5]	1
double*	2	d	1
int	2	e	2

代码:

```
static volatile char a __attribute__((at(0x80)));
static volatile short b __attribute__((at(0x180)));
static volatile float c[5] __attribute__((at(0x182)));
static volatile double* d __attribute__((at(0x191)));
static volatile int e __attribute__((at(0x280)));
```

其中：

MCU：MCU 的名称

类型和类型长度：变量的类型和类型的长度，如果类型是基本类型或指针，则类型长度不可改变。

名字是变量的名字，变量名可以是数组。

Bank：指定的 Bank。

双击变量列表可以对已添加的变量进行编辑。

- 复制代码
复制生成的代码
- 生成代码
生成代码
- 语言
切换界面的语言

编辑器

Voice ROM Editor

对于特别指定的微控制器 (例如 HT56R 系列), 此命令提供 VROM 编辑器, 让使用者可安排语音码。

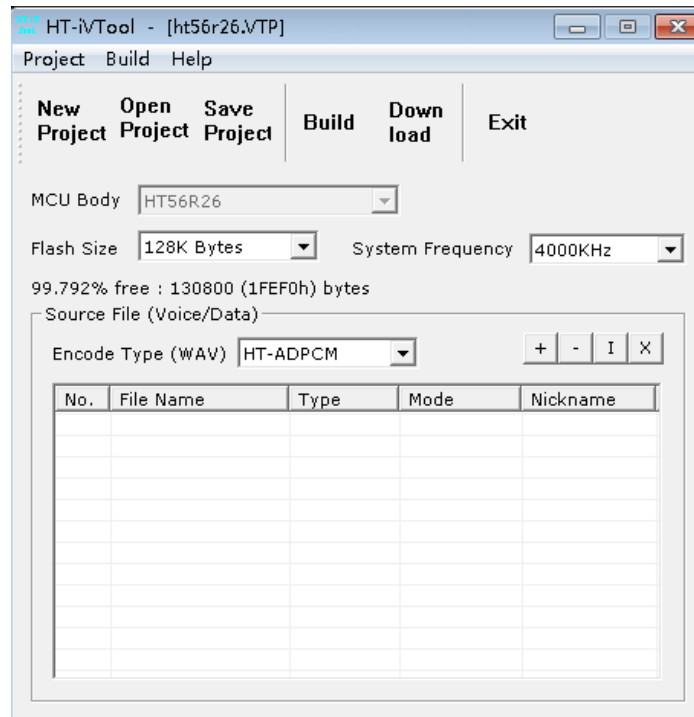


图 3-13

Data Editor

本公司的某些 MCU 如 HT48E 系列内部含有 Data EEPROM。Data EEPROM 编辑器提供一个接口可以让使用者来编辑资料，并可以从 ICE 上载或下传数据。

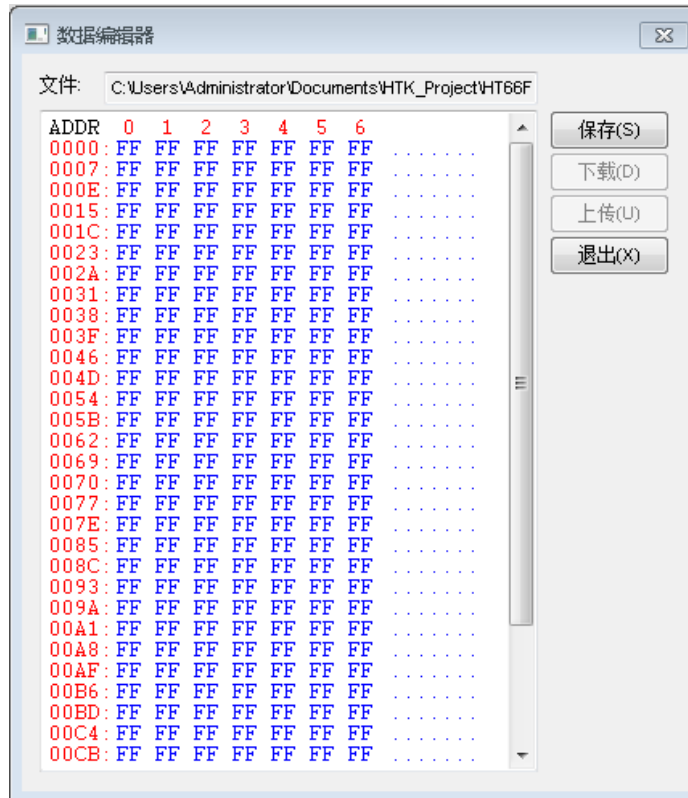


图 3-14

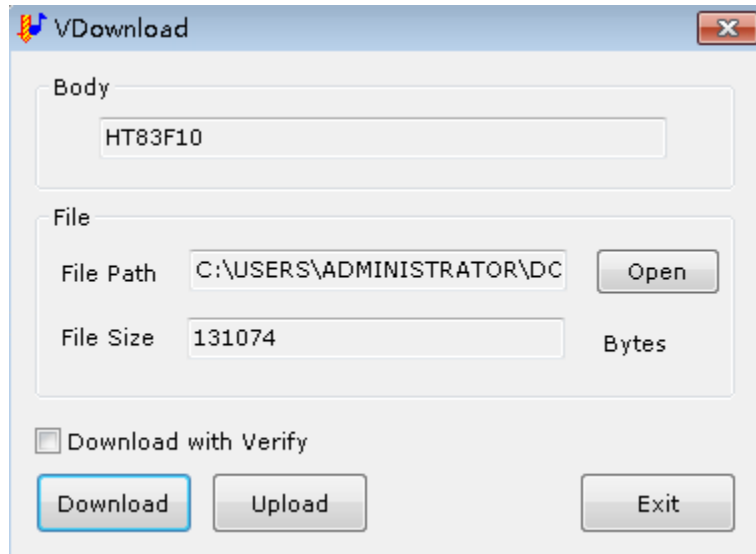


图 3-15

LCD 软体模拟器

LCD 软件仿真器，简称 LCDS，提供一种机制来仿真 LCD 驱动装置的输出。根据设计好的图案和控制程序，LCDS 能够实时将图案显示在屏幕上。LCD Simulator 仅支持 ICE 架构的部分 MCU，另外该工具已不再提供后续的更新。

Voice & Flash 下载

此工具下载扩展名为 .VOC 或 .DAT 的语音文件内容到 ICE 做硬件仿真或透过 e-Writer 烧录到 SPI Flash，也可以从 ICE VROM 或 SPI Flash 上传数据并储存成 .VOC 或 .DAT 档。图 3-15 之对话框显示文件名由 VROM 编辑器所产生的下载语音文件 .VOC，文件名下方的方块盒会显示项目中微控制器的语音数据所需要的内存大小 (单位 Bytes)。在下载语音文件之前，必须确定语音文件 .VOC 已经由 VROM 编辑器产生。

OCDS 模式切换

此命令只针对 OCDS-ICE，有 2 个模式可供选择，对应不同的封装 (图 3-16)。



图 3-16

智能烧录设定

对于类型为 OTP 或 MTP 的 MCU，可以进行智能烧录的设定。

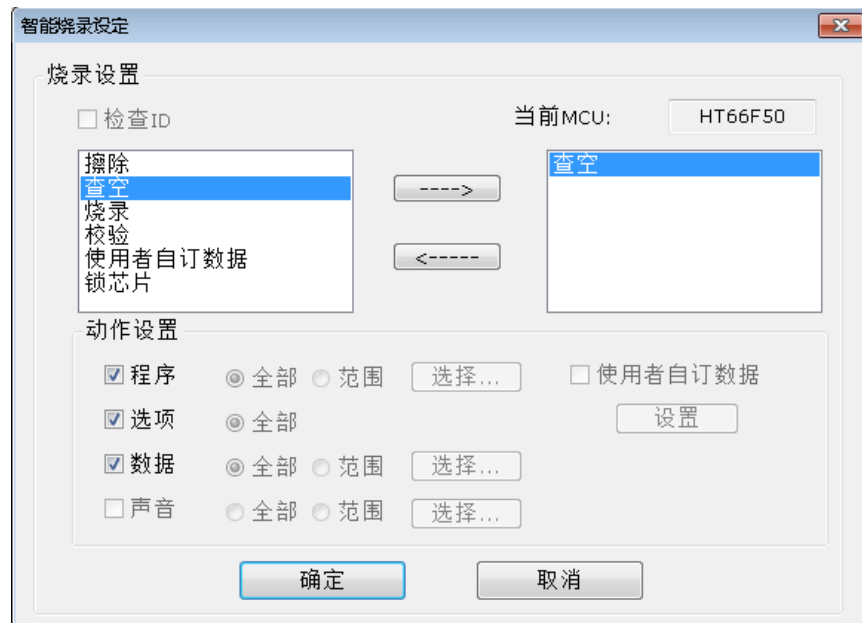
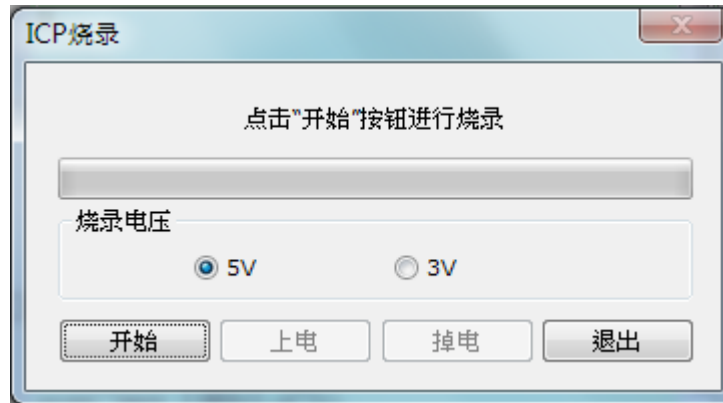


图 3-17

智能烧录详细介绍可以参考 HOPE3000 说明文档的第四章。

ICP 烧录

如果 MCU 为 Flash 类型，并且没有开启 Bootloader 功能，那么就可以使用 ICP 进行烧录。



- 烧录电压
选择烧录电压，用户可以自行选择使用 3V 或者 5V 烧录，对于那些只能以固定电压烧录的 MCU，IDE 会屏蔽电压的选择，自动选择合适的电压。
- 开始
开始烧录 .ICP 烧录成功后，IDE 会对 MCU 上锁。
- 上电
只有烧录成功后，该按钮才会使能，将根据所选的烧录电压进行上电。
- 掉电
只有烧录成功后，该按钮才会使能。
- 退出
退出 ICP 烧录。

选项选单

选项选单 (图 3-18) 提供下面命令，可对其他选单与命令设定工作参数。

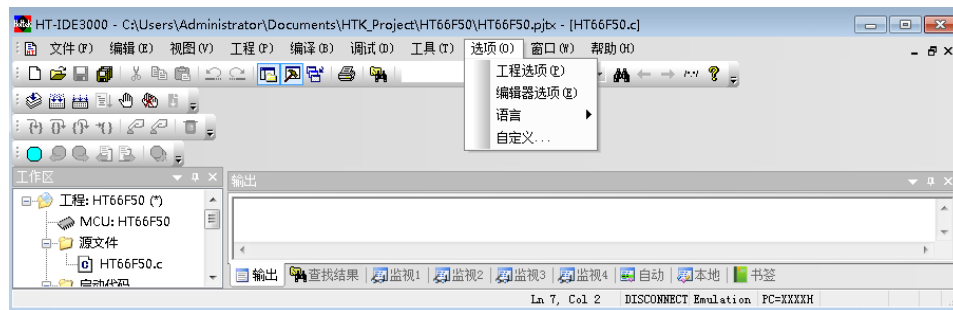


图 3-18

工程设置

工程选项

工程命令设定编译选单中构建命令的内定参数值。在程序开发期间，项目选项依据应用程序的需求而改变。HT-IDE3000 在执行编译选单中的构建命令时，将根据这些选项的设定产生对应的工作档案及内容。对话框 (图 3-19) 是用来设定工程的选项。



图 3-19





图 3-20

注意：在执行构建命令之前，要先确认项目选项的设定正确。

- 微控制器
 - 项目中的微控制器名称。使用滚动条去浏览可使用的微控制器，并选择恰当的微控制器。
- ISP Bootloader 选项
 - 该按钮将弹出配置对话框如图 3-20。
 - ◆ 该功能仅适合 USB Flash MCU 系列。
 - ◆ 使用 ISP 烧录模式需勾选使用 ISP BooLoader，再勾选上电时优先执行程序区块
 - ◆ Start at ISP mode 上电先执行 BootLoader 程式区块，等待 AP 開啟進行 USB Link，更新完成後再跳躍至使用者程式段 (User code)。
 - ◆ Start at User mode 上电先执行使用者程式 (User code) 区块，需要更新才呼叫 Library 返回 BootLoader 进行更新，更新完成後再跳躍至新的使用者程式段 (User code)。
 - ◆ ID 选择，可辨识相同 bootloader 的不同产品
 - ◆ bootloader VID/PID 选项，该功能可使用 HOLTEK 提供的 VID/PID，也可使用自定义 VID/PID
 - ◆ 使用自定义 bootloader 选项，该功能可载入自定义 bootloader 的 mtp 档案
- 语言工具
 - 有 Holtek C compiler V1、Holtek C compiler V2、Holtek C compiler V3 供选择，其中 Holtek C compiler V3 有优化程式功能，建议您使用此版 C compiler 开发程式。
- USB FIFO
 - 可设置 USB FIFO 大小
- 编译选项
 - 该按钮将弹出配置对话框，用于设置编译选项。
- 汇编 / 编译选项
 - 组译器的命令列选项。定义符号栏允许使用者对指定的符号定义数值，而所指定的符号则在编译过程中使用。语法如下：


```
symbol1 [=value1] [,symbol2 [=value2] [,...]]
```
 - 例如：


```
debugflag=1, newer=3
```
 - 可点选产生列表档 (Generate listing file) 检查盒以便产出原始程序行表档。另外，可点选生成工程列表档 (list) 产出反汇编记录档案 (.list) 在工程目录下

- 连接选项

指定连结器的选项。Libraries 文字盒用来指定被连结器所参考使用的函式库档案。例如：

```
libfile1, libfile2
```

也可以按下 Browse 键来浏览及选择函式库档案。

另外，程序段地址 (Section address) 输入盒则是用来设定程序段的 ROM/RAM 地址。例如：

```
codesec=100, datasec=40
```

最后一行的对映档检查盒 (Generate map file) 则是命令连结器产出项目的程序对映文件。

- 调试选项

此命令设定被调试选单使用的选项功能 (第五章 HT-IDE3000 选单 - 除错)。对话框 (图 3-21) 列出附带有检查盒的除错选项，当完成选项的选择和按下确定钮之后，除错过程中调试选单及命令就根据这些选项执行对应的动作。

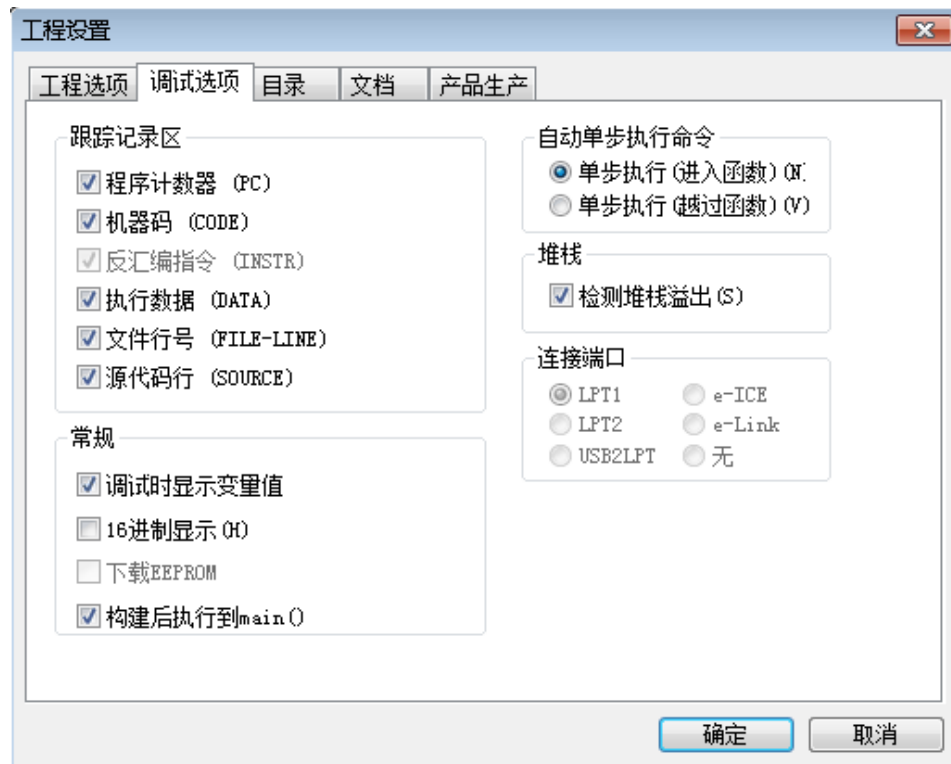


图 3-21

- 跟踪记录区

此字段中的各选项将会影响程序追踪 (trace) 时所记录的数据是否要显示在窗口内。在执行窗口选单的跟踪记录表命令时，这些选项会指示那些追踪数据需要显示出来。对于每个原始文件内的指令，信息的显示顺序将与对话框项目的排列秩序一样，从上往下。如果其中的项目没有被选取，下一个被选取的项目的数据将往前移。追踪表列的固定数据只会显示文件名称和各行的行数。程序计数器是指令的内存地址，机器码是指令的机器码，反汇编指令是从机器码直接反译回来的指令，而源代码行是原始程序文件的指令行。

如果是只读动作的指令，执行数据显示的是读取的数据，然而若是只读或读

写动作的指令，则显示被写入的数据。文件行号是要显示原始程序文件名与指令行的行数。

- 常规

此栏中可以选择在除错模式下所使用的功能，例如变量值显示、十六进制显示、下载 EEPROM 数据。只有选定的 IC 具有 EEPROM 才可以勾选下载 EEPROM，勾选后，当执行构建命令时，会自动将 EEPROM 下载到硬件仿真器里。勾选构建后执行到 main()，会在初始化后，将光标停留在 main 函数首行。
- 自动单步执行命令

此选项可以选择自动呼叫程序的执行步骤，即单步执行 (进入函数) 或单步执行 (越过函数)。只能选择其中之一。
- 堆栈

如果你不需要系统在发现堆栈溢满时显示警讯，请不要勾选此检测堆栈溢出。
- 连接窗口

选择 ICE 连接口。

目录

这个命令是设定用来找寻执行档或储存输出档案的内定路径和档案夹 (图 3-22)。

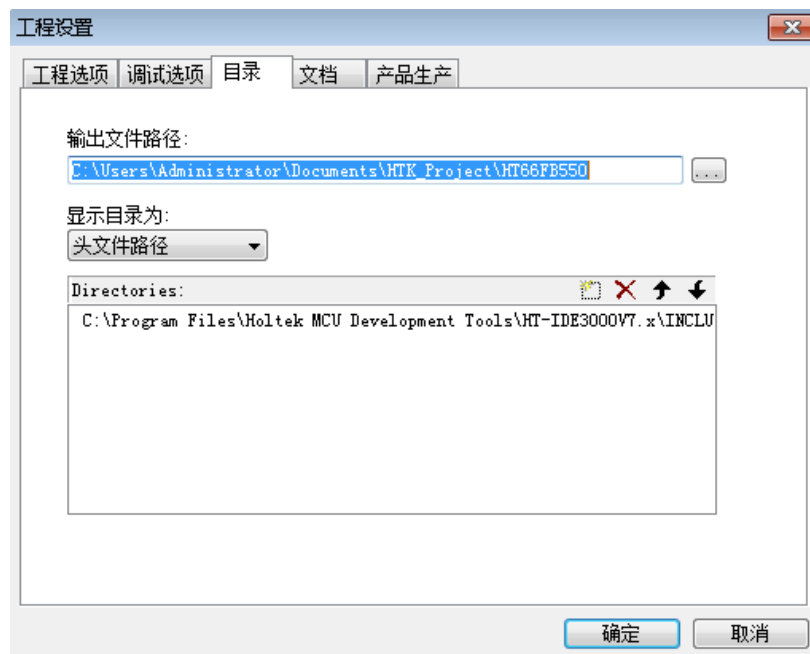


图 3-22

- 头文件路径

组译器会从此处设定的路径或档案夹寻找含入档。
- 库文件路径

连结器从此处所设定的路径或档案夹寻找函式库档案。
- 输出文件路径

储存组译器的输出档案 (.obj, .list) 和连结器的输出档案 (.tsk, .map, .dbg) 所使用的路径。

文档

这个命令是设定用来添加说明文档至工程，添加之后可通过左侧的文档窗口看到(图 3-25)。

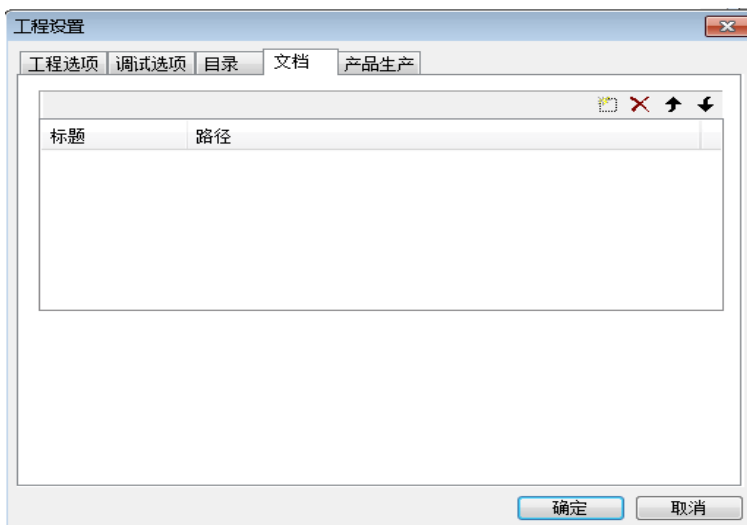


图 3-23

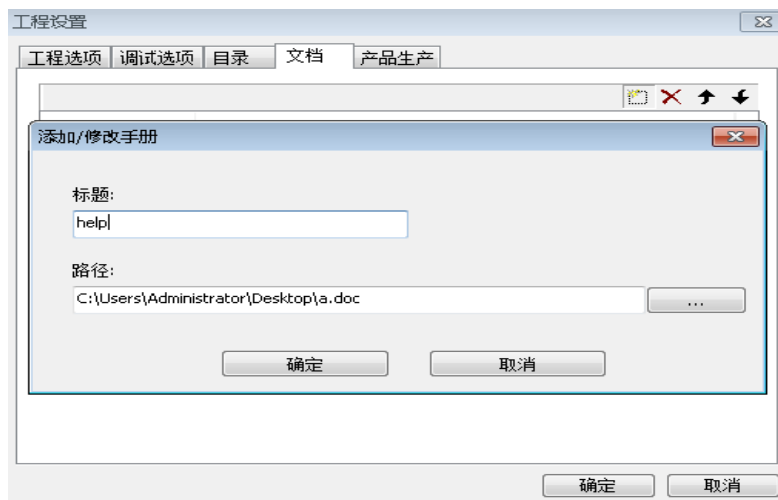


图 3-24

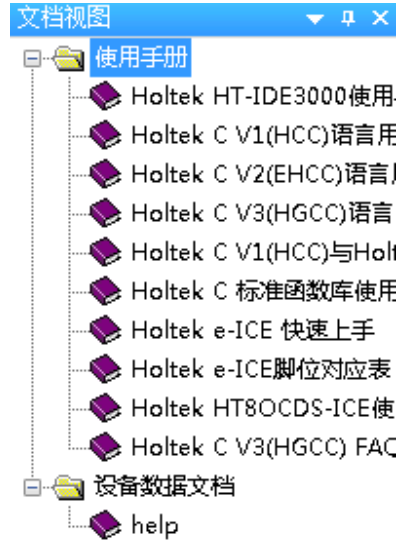


图 3-25

产品生产

为了可以让客户识别已被 lock 的 IC 是属于何种产品，HT-IDE3000 提供了一项功能：烧录识别码 (图 3-26)。

设置完成后，点击“ReBuild All(全部重建)”按钮，即可将识别码写入相应的文件 (OTP、MTP、PND)。

注意：只有选择的 MCU 为 OTP 或 flash 类型时才能使用该功能。

HT-IDE3000 v7.71 及以上版本才支援该功能

HOPE3000 v3.06 及以上版本才支援该功能

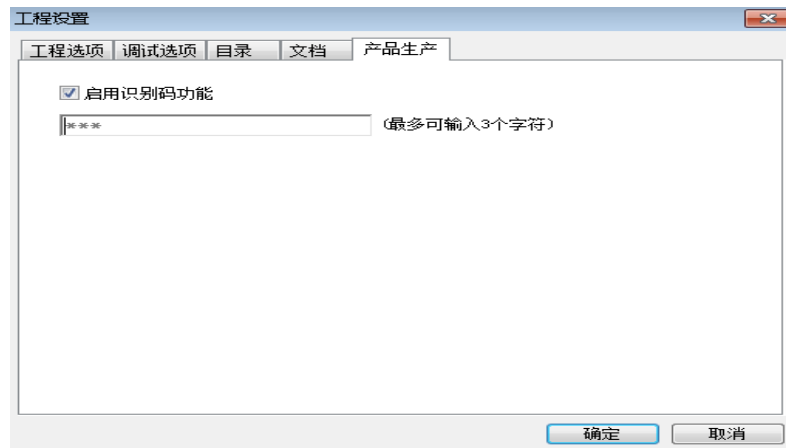


图 3-26

编辑器设置

格式

这个命令可以设定指定列的前景和背景颜色。在图 3-27 左边的选项中，选中的文本是用在编辑选单，当前行，断点行，跟踪行和堆栈行是用在调试选单，而错误行则是针对组译器的输出。

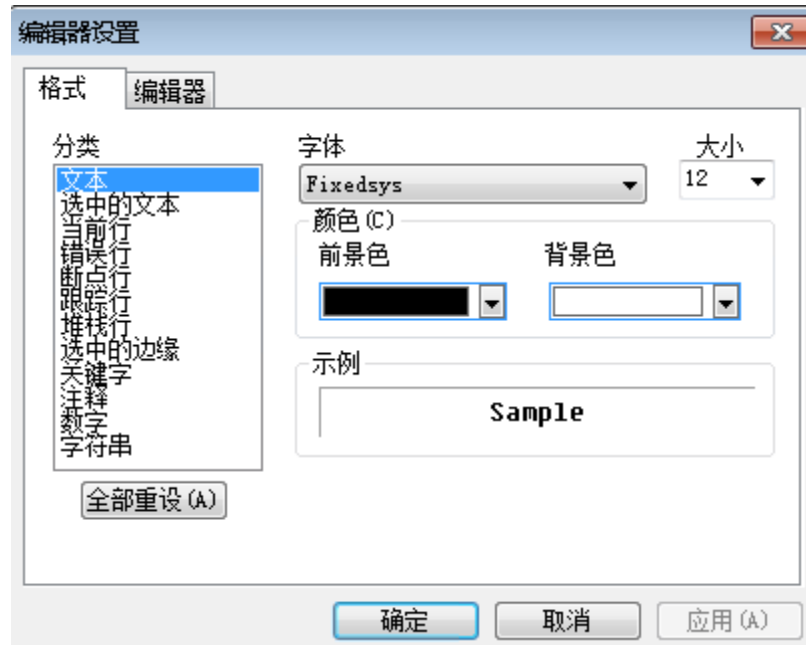


图 3-27

编辑器

这个命令会设定编辑器选项，如空格键的大小和撤销的有效次数，以及禁用某些功能。运行工具前保存选项是要在组译前预先储存档案。自动重新加载外部修改的文件会自动加载已被外部修改后的档案。不使用 Classview 在类视图窗口将不显示任何信息。不显示变量列表禁用变量提示列表，不自动提示函数参数禁用函数参数提示。



图 3-28

语言

这个命令是设定目前 HT-IDE3000 要用哪一种语言接口，包含英文、繁体中文、简体中文三种，预设是指系统语言。

自定义

这个命令是设定重新排列和修改工具栏按钮、菜单和菜单命令，用户可以根据自己的操作习惯来自定义工具栏，例如将自己常用的命令添加到自定义的工具栏上。

关于 e-Link 的激活

对于第一次使用 e-link 必须要注册。

具体为点击 Help 下的 About HT-IDE3000 菜单项，出现如下对话框：



图 3-29

点击 Register e-Link 按钮，出现如下：

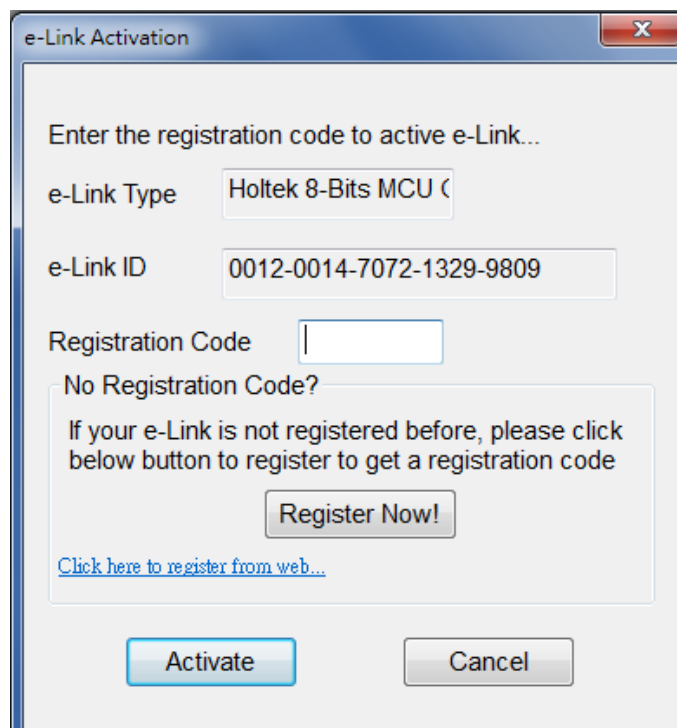


图 3-30

点击 Register Now 按钮，出现：



The image shows a 'Registration Sheet' dialog box with the following fields and values:

- e-Link 类型: Holtek 8-Bits MCU OCDS
- e-Link ID: 0100 - 1021 - 8939 - 0225 - 4164
- E-mail(*): [Empty]
- 名: [Empty]
- 姓: [Empty]
- 公司: [Empty]
- 国家: [Empty]
- 城市: [Empty]
- 购买日期: 2016/12/ 9

Buttons: 注册 (Register), 取消 (Cancel)

图 3-31

填写注册信息，其中 e-Mail 栏位为必填项，其余的为选填项。填写完毕后点击 Register，稍等片刻，您的邮箱将会收到一封主旨为 e-Writer Pro Registry Key 的邮件。填写邮件里的注册码，点击 Active，将出现如下提示，说明您已激活成功。

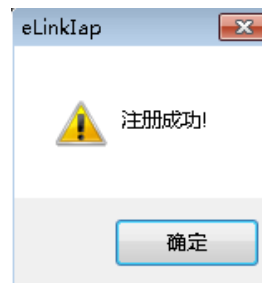
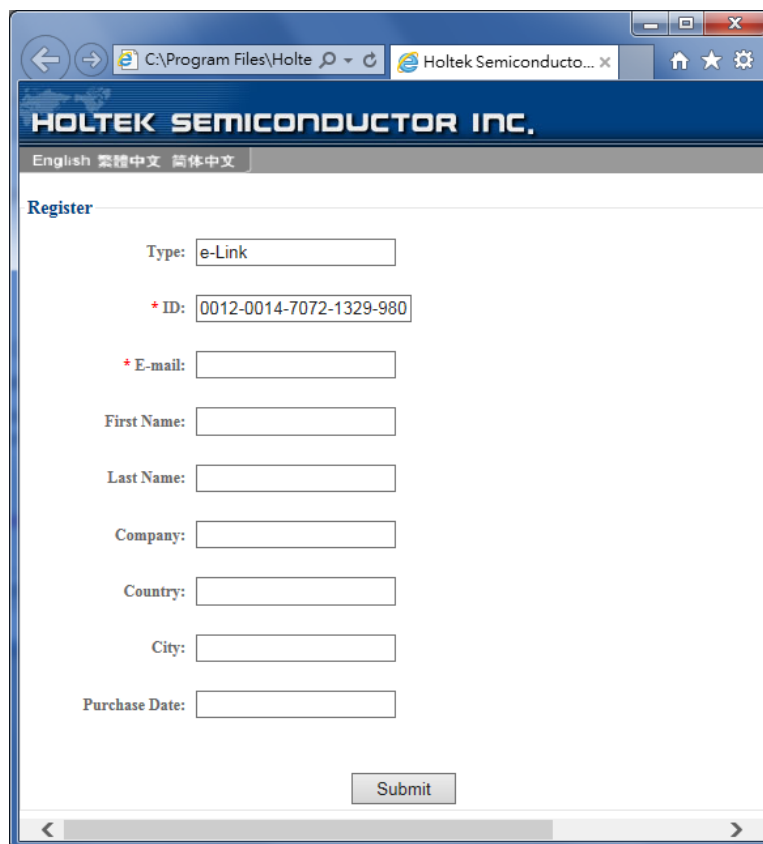


图 3-32

如果注册过程遭到杀毒软件或者防火墙的拦截，那么你可以尝试通过网页进行注册。

点击超链接“Click here to register from web...”



The screenshot shows a web browser window displaying the registration page for Holtek Semiconductor Inc. The page title is "HOLTEK SEMICONDUCTOR INC." and the language is set to "English". The registration form includes the following fields:

- Type: e-Link
- * ID: 0012-0014-7072-1329-980
- * E-mail: [Empty]
- First Name: [Empty]
- Last Name: [Empty]
- Company: [Empty]
- Country: [Empty]
- City: [Empty]
- Purchase Date: [Empty]

A "Submit" button is located at the bottom of the form.

程序会自动填入 Type 和 ID 值，其他项目由用户填写，其中 E-mail 为必填项目。

书签

[书签] 窗口是 [程序代码编辑器] 随附的一种便利工具。您可以将程序代码档案中的行标记成书签，这样，只要双击一下 [书签] 窗口中的项目，就可开启档案并直接浏览至已标记的程序代码行。

书签窗口

您可以从 [窗口] 菜单呼叫出 [书签] 窗口。

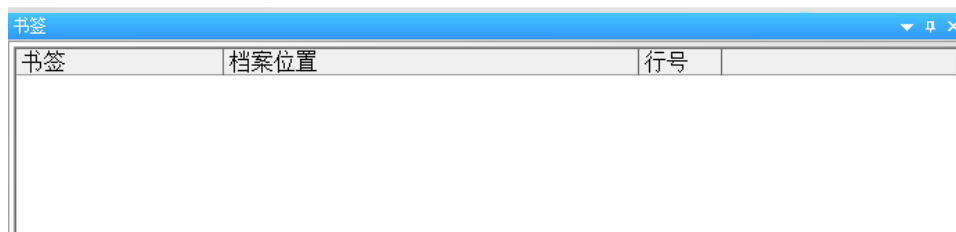


图 3-33

书签

显示书签的名称。预设所建立的名称为“未命名”。您可以双击书签的名称，以建立自订书签名称。每个书签都有自己的复选框。若要启用现有的书签，请在 [书签] 窗口中选取该书签的复选框。若要隐藏 (但不移除) 现有的书签，请在 [书签] 窗口中取消该书签的复选框。

档案位置

列出完整的档案路径。

行号

列出书签所在的程序代码行行号。

提示：双击“档案位置”或者“行号”，可以快速跳转到该位置。

书签工具列



图 3-34

工具列从左到右分别为：

- 切换目前行的书签
在编辑器中，于文件的插入号所在行上加入或移除书签。但不会修改加入书签的程序代码行。
- 将插入号 (Caret) 移到上一个书签
跳转到 [书签] 窗口中启用的上一个书签。当到达第一个书签时，直接跳到最后一个书签。依照需要，编辑器会显示所选书签的档案。将文件卷动至书签的程序行，并将插入点放在这行上。
- 将插入号移到下一个书签
跳转到 [书签] 窗口中启用的下一个书签。当到达最后一个书签时，直接跳回第一个书签。依照需要，编辑器会显示所选书签的档案。将文件卷动至书签的程序行，并将插入点放在这行上。
- 将插入号移到当前文件的上一个书签
跳转到当前文件中启用的上一个书签。当到达第一个书签时，直接跳到该文件的最后一个书签。依照需要，编辑器会显示所选书签的档案。将文件卷动至书签的程序行，并将插入点放在这行上。
- 将插入号移到当前文件的下一个书签
跳转到当前文件中启用的下一个书签。当到达最后一个书签时，直接跳回该文件的第一个书签。依照需要，编辑器会显示所选书签的档案。将文件卷动至书签的程序行，并将插入点放在这行上。
- 删除所有的书签
删除 [书签] 窗口中所有书签。但不会修改书签所标记的程序代码行。

书签菜单

除了用书签工具列进行操作，您还可以从 [编辑] 菜单中的 [书签] 子菜单来对书签进行操作。


	切换书签	Ctrl+F2
	上一个书签	Shift+F2
	下一个书签	F2
	文件中上一个书签	
	文件中下一个书签	
	清除所有书签	Ctrl+Shift+F2

图 3-35

第四章 选单 – 专案 & 建制

IDE300 提供项目范例帮助第一次使用者快速地熟悉项目开发程序，从 HT-IDE3000 系统的观点来看，一个工作单元是由包含应用程序的项目所描绘。

当首次开发一个 HT-IDE3000 的应用程序时，应遵循先前所建议的开发步骤进行。

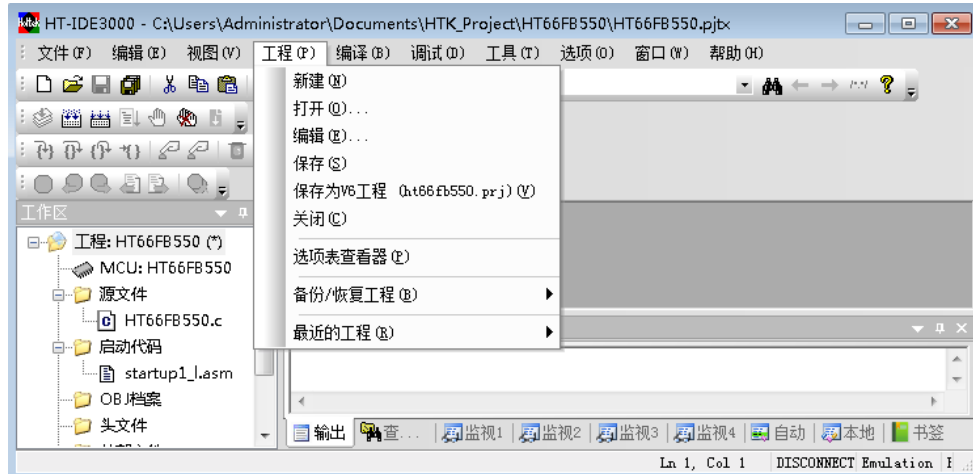


图 4-1

建立新项目



在工程选单 (图 4-1) 选择仙剑命令去建立一个新项目，此命令会呼叫 CodeWizard 来帮助使用者快速建立项目。

注意：专案名称的扩展名为 .PRJ 和 .PJTJX

CodeWizard 流程

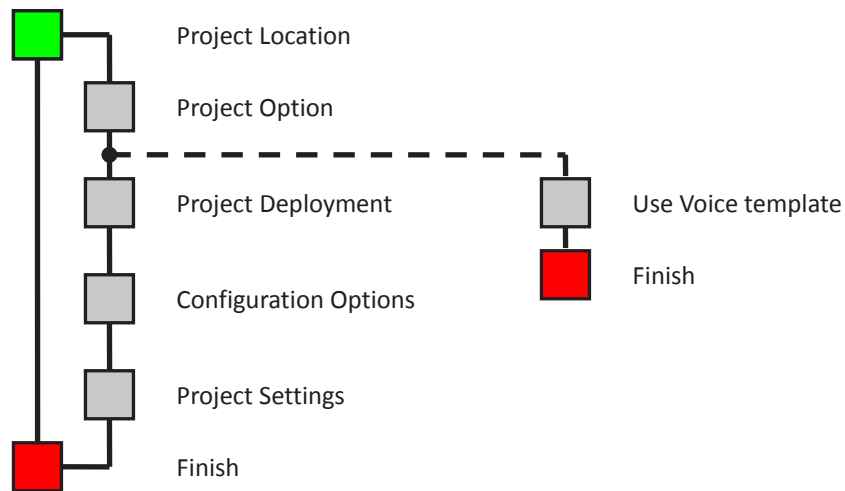


图 4-2

步骤 1: Project Location

首先使用者输入项目名称、项目路径 (如图 4-3)。使用者可以浏览所有资料与已存在的项目, 再从其中选择已经存在的项目 (覆盖旧有项目) 或是输入新的项目名称。另外再选择项目微控制器类型、以及开发语言工具, 对于某些特定的 MCU, 能够启用 with BootLoader, 该选项使 IC 有 IAP 功能 (如图 4-4)。

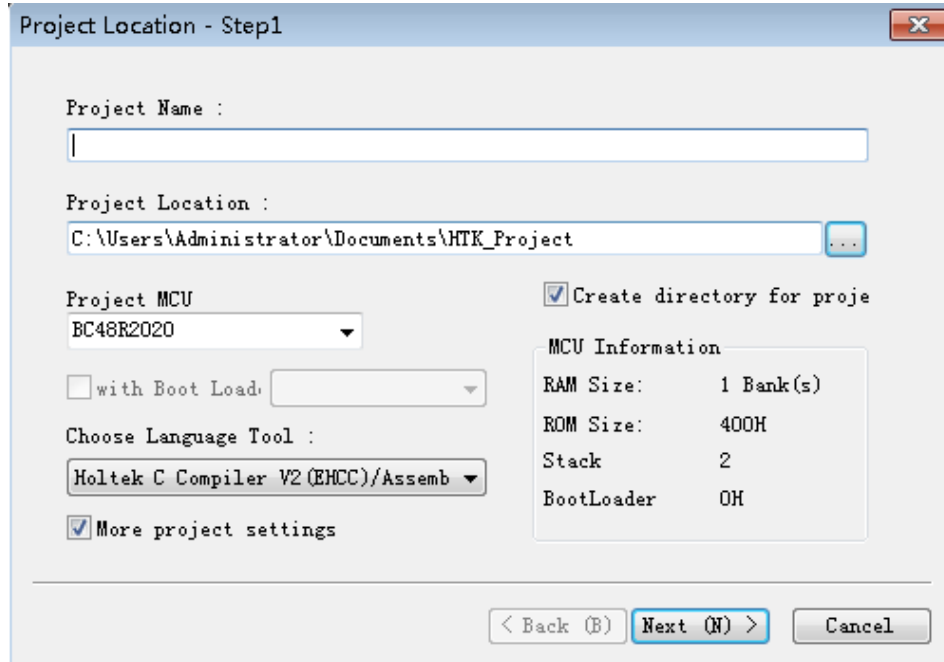


图 4-3

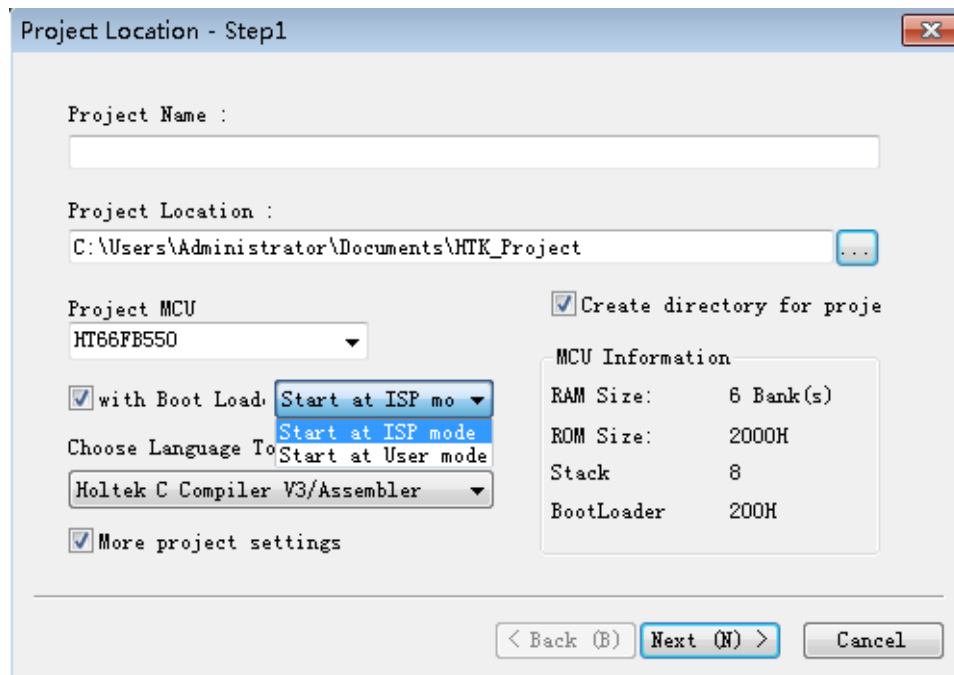


图 4-4

步骤 2: Project Option

步骤 2 则是选择用组译器或 C 编译器去产生预设的项目。

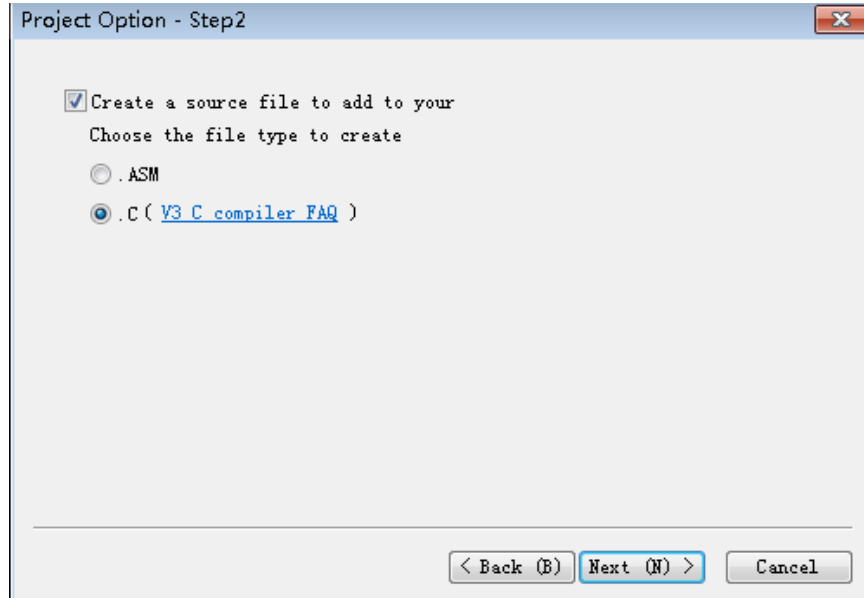


图 4-5

步骤 3: Project Deployment

步骤 3 则是变更原始程序文件名称、程序节区名称、以及数据节区名称。只有选择组译器时才会去编辑程序节区名称与数据节区名称。

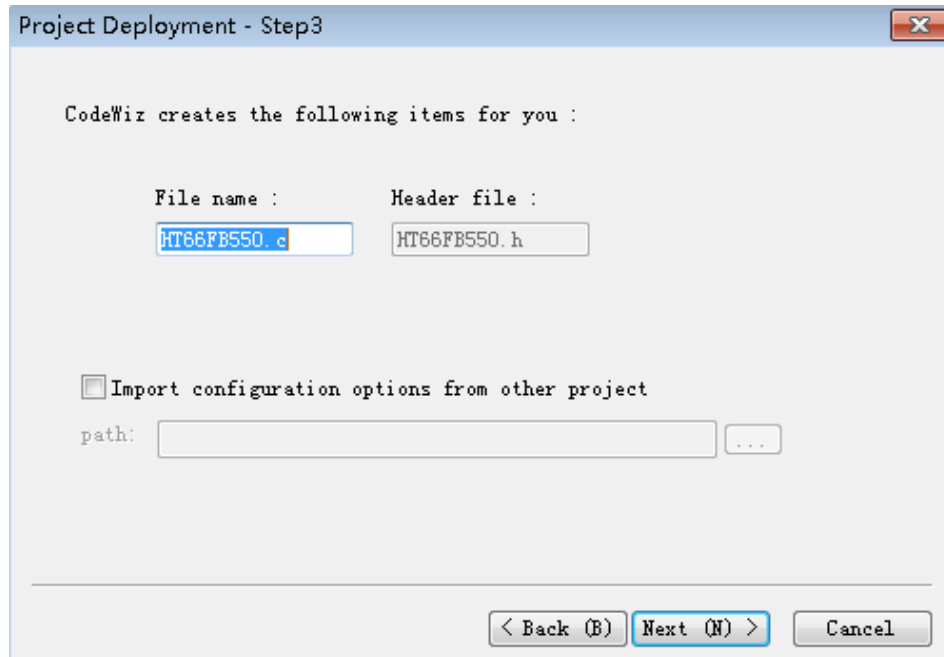


图 4-6

最后，系统会要求设定配置选项和项目设定，产生初始项目。

开启和关闭专案

任何时候 HT-IDE3000 只能执行一个项目，就是已经开启的项目。如果要使项目开始作业，则首先要使用 Project 选单的 Open 命令开启此项目 (图 4-1)，可直接输入项目名称，或浏览数据夹去选择项目名称。Close 命令将关闭目前使用的项目。

注意：当开启一个项目时，则目前已开启的项目会自动关闭。在开发过程中，亦即在编辑、设定选项及除错的阶段时，必须确定项目在开启的状态。可以从显示在 HT-IDE3000 窗口上方的开启项目名称得知。如果项目不在开启状态下，则结果是不可预测的。

当结束 HT-IDE3000 的执行回到一般窗口时，如果没有关闭已开启的项目，则系统会将此项目的数据保存起来，在下次执行 HT-IDE3000 时，系统会自动去开启此项目。

管理项目的原始档

项目中必须要有原始程序文件做为其骨干，使用工程选单的编辑命令可以将原始程序文件加到项目中，也可从项目中将原使程序文件删除。列表盒中原始程序文件的顺序由上而下是依照这些档案加入的顺序而定，也可以使用上移与下移按钮重新调整档案的排列次序。连结器会根据列表盒中档案的顺序去处理这些输入档。图 4-7 是工程选单中编辑命令的对话框。

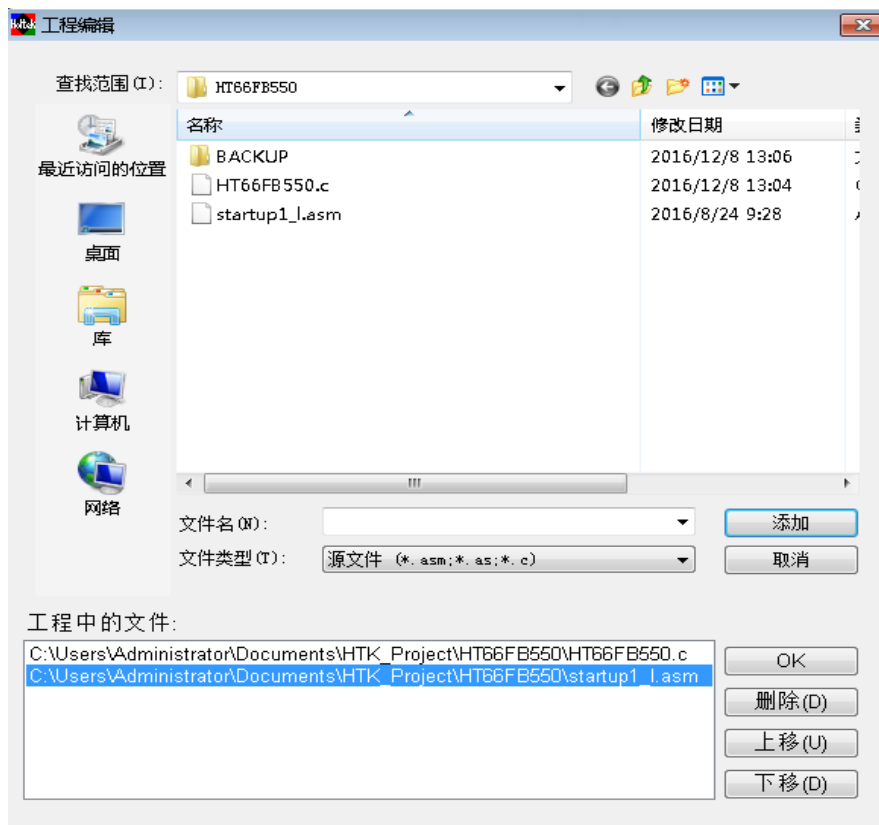


图 4-7

将原始程序文件加入项目

从档案列表盒中选取原始程序文件名称，双击所选取的文件名称或是按下添加按钮，将原始程序文件加到项目中。当所选取的原始程序文件被加入到项目后，档案的名称会显示在项目中的档案列表盒上 (Files in Project)。

从项目中删除原始程序文件

- 从工程中的文件列表盒中选择欲删除的档案。
- 按下删除按钮。

注意：从项目中删除原始程序文件并不会真正将档案删除，只是将档案数据从项目中移除。

向上或向下移动原始程序文件的位置

- 在工程中的文件列表盒中，将光标移到指定的档案处，并按下鼠标左键。
- 再按下上移按钮或下移按钮，将指定的档案往上移或往下移动位置。

建成项目的工作档

建成新的项目之前，应确定下列工作已完成：

- 项目已经开启
- 已完成项目选项的设定
- 已经将原始程序文件加到项目中
- 已选定项目所使用的微控制器 (参考工具选单章节)

两个建成项目档的命令，分别为构建命令和重建命令。

构建命令会执行下列的工作：

- 根据程序文件的扩展名为 .asm 或 .c 分别呼叫组译器或 C 编译器去编译项目中所有的原始程序文件
- 将组译器或 C 编译器产生的目的档连结，并产生工作档和除错档
- 如果 ICE 的电源已打开，并且与个人计算机相连，则下载工作档至 ICE
- 将开始执行点的原始程序显示在工作窗口上 (HT-IDE3000 硬件仿真须参考原始文件、工作档和除错档)

注意：构建命令会根据相关档案建立的日期与时间决定上述那些工作会被执行或不被执行。规则如下：

- 若原始程序文件的产出日期 / 时间晚于它的目的档的产出日期及时间，则构建命令会呼叫组译器或 C 编译器去编译原始程序文件并产生新的目的档。
- 若至少有一个目的档的产出日期 / 时间晚于工作档的产出时间，则构建命令会呼叫连结器将所有目的档连结起来并产生新的工作档。

无论上述的工作是否都被执行，构建命令会自动将工作档的内容下载到 ICE。

重建全部命令如同构建命令一样去执行相同的工作，其差异在重建全部会立刻执行工作，而不会先比较所有档案的产出日期 / 时间。

构建或重建命令执行的结果讯息会显示在输出窗口。在处理过程中如果发生错误，其后的工作会被忽略，不会产出工作档及下载工作档。

建成项目的工作档

- 按下工程选单中的打开命令以便开启项目。
- 按下工程选单中的构建命令或工具列中的构建键 (图 4-1)，开始建立一个项目。

重建项目的工作档

- 按下工程选单中的打开命令以便开启项目。
- 按下编译选单中的重建全部命令或工具列中的重建全部键 (图 4-1)，开始建立一个项目。

一旦建好项目工作，就可以开始进行应用程序的仿真和除错 (参考 HT-IDE3000 选单 - 除错章节)。

组译 / 编译

可以先行用此命令去编译原始程序与在输出窗口上显示其结果的方式，来验证应用程序的正确与否。因为一次只会编译一个原始程序文件，使用者可以不用一次面对整个项目中所有原始程序文件编译的结果，尤其是不只一个程序档案发生错误时。

编译程序

- 使用文件选单打开命令开启原始程序文件以进行组译或编译
- 选用编译选单的编译命令或按下工具列的编译键去编译程序文件

假如已开启档案的扩展名为 `.asm`，则系统会呼叫组译器执行编译工作。如果已开启档案的扩展名为 `.c`，则系统会呼叫 C 编译器去编译程序。

如果没有发生错误，则会产生扩展名为 `.OBJ` 的目的档，并且储存在 **Output Files Path** 所指定的目录内 (参考选项选单中的目录命令)。如果发生错误，则在输出窗口上显示错误讯息，可使用下列的方式将错误行所在的原始程序档案开启并将光标置放于错误行。

- 将光标移到输出窗口内的错误讯息行，双击鼠标左键或按下 `<Enter>` 键

确认单查看器

该命令用来查看工程中的 `otp,mtp,pnd,cod` 文件 (如图 4-8)。

Browse 按钮用来打开文件，**Print** 按钮用来将目前使用中之配置选项文件的内容从指定的打印机上印出，可以选用不同的打印机去打印。建议不要使用连接至 ICE 的打印机端口。

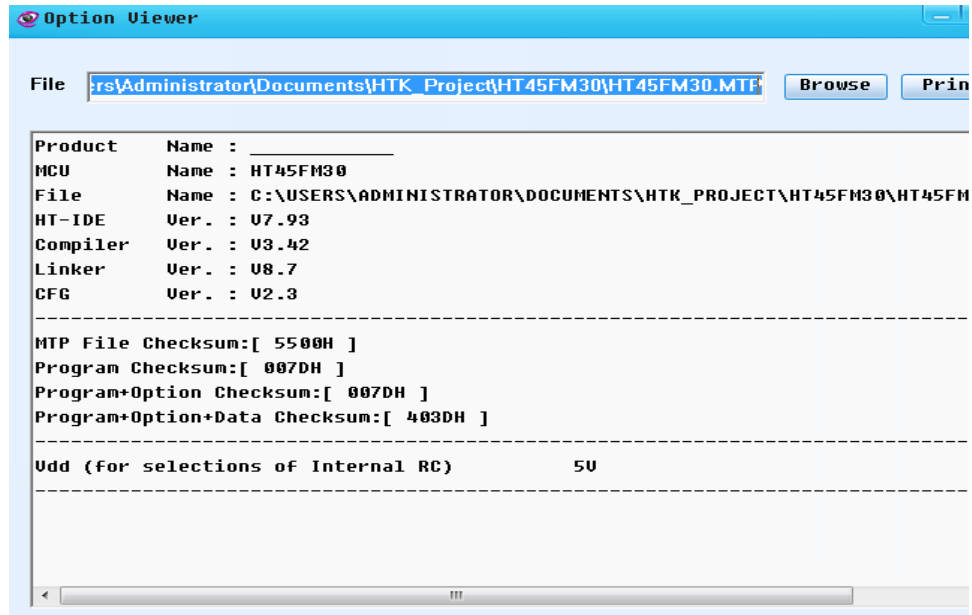


图 4-8

假如打印机和 ICE 使用相同的打印机端口，则发出此命令时会造成除错信息与相关数据的损失。在打印工作结束后，假如应用程序仍需做硬件仿真与侦错，则使用者应回到开发过程的开始处，再使用工程选单中的构建指令做下载动作。

备份 / 恢复工程

备份工程命令使用 PROJECT_DATE_VERSION 格式文件名称去压缩目前的项目。你能够在 [Description] 编辑盒中加入项目的描述。

恢复命令会以目前在备份列示盒中所选的压缩档案，将项目恢复。

第五章 选单 – 除错

在程序开发的过程中，重复的修正和测试原始程序是不可避免的。HT-IDE3000 提供许多工具，不只让除错工作更加容易，而且也减少开发时间。这些功能包含了单步执行、符号中断、自动单步执行、追踪触发条件等。

当应用程序建立成功之后(参考前一章中建成项目的工作文件)，在工作窗口(图 5-1)中的原始程序代码，被反白标示的程序行，将是第一个被执行的程序行。之后，系统准备接受及执行使用者所下的除错命令。

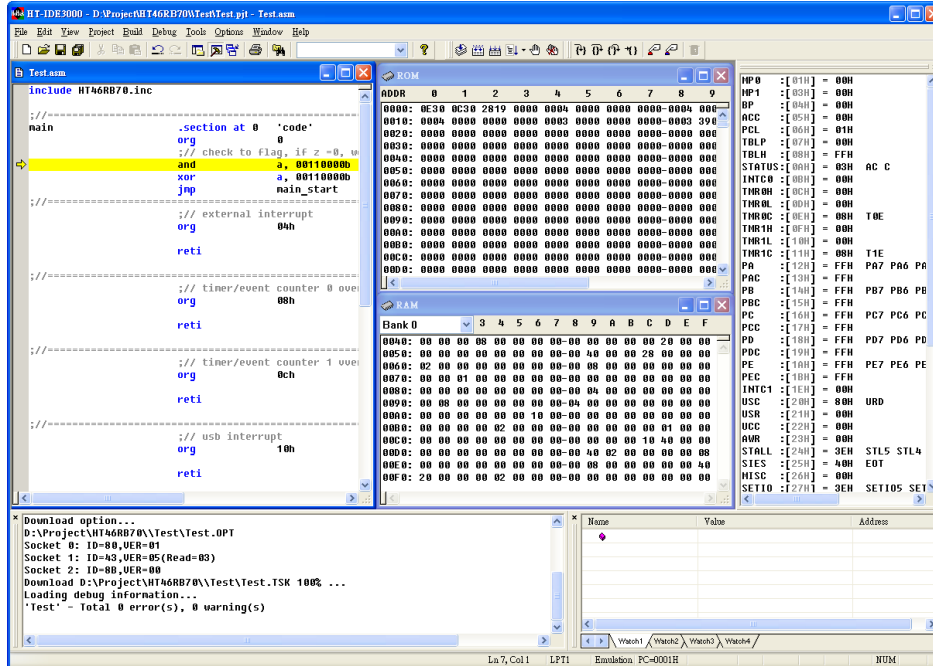


图 5-1

重置 HT-IDE3000 系统

HT-IDE3000 系统提供四种重置方法：

- 电源重置 (Power-on reset, POR) 可重新将电源接到 ICE 或按下 ICE 上的重置钮。
- 来自应用电路板的重置讯号。
- 来自 HT-IDE3000 中调试选单 (图 5-2) 的软件重置复位命令。
- 来自 HT-IDE3000 中调试选单 (图 5-2) 的软件电源重置上电复位命令。

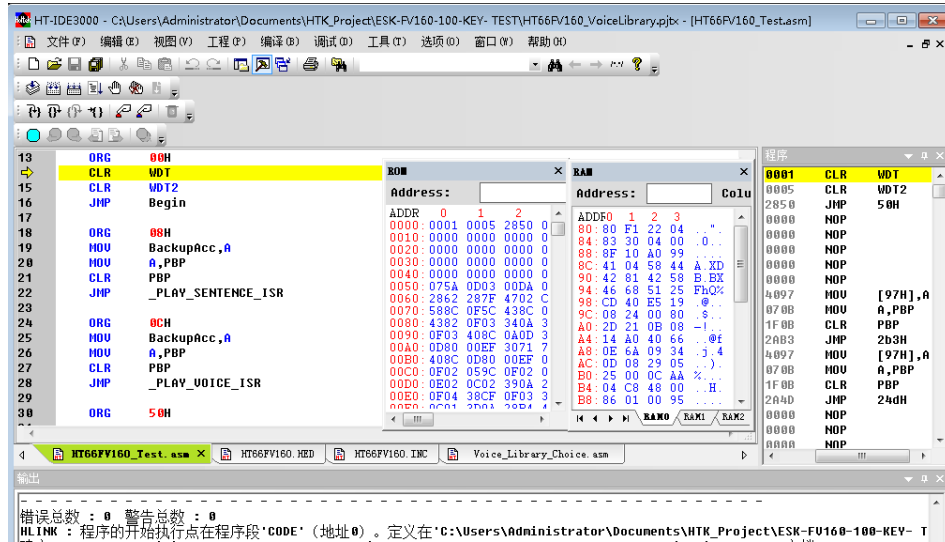


图 5-2



表 5-1 列出上述四种重置的影响。

Reset Item	Power-On Reset	Target Board Reset	Software Reset Command	Software Power-On Reset Command
Clear Registers	(*)	(*)	(*)	(*)
Clear Options	Yes	No	No	No
Clear PD, TO	Yes	No	No	Yes
PC Value	(**)	0	0	0
Emulation Stop	(**)	No(***)	Yes	Yes
Check Stand-Alone	Yes	No	No	No

表 5-1

注意：* 寄存器在不同重置下所受的影响，请参考各微控制器的 Data Book。

** PC 值为 0，同时停止硬件仿真。

*** 假如重置来自于应用电路板，则系统会在重置完成后自动开始硬件的仿真。

PC – 程序计数器 program counter

PD – 电源下降旗标 power down flag

TO – 逾时旗标 time out flag

从 HT-IDE3000 命令执行重置

- 选按调试选单的复位命令或按下工具列的复位钮去执行软件重置。
- 选按调试选单的上电复位命令或按下工具列的上电复位钮去执行重置。

应用程序的硬件仿真

在应用程序撰写完成后，需要使用构建或重建命令建成项目的工作文件以及将程序下载到硬体仿真器。如果此命令被执行完毕，没有发生错误，则在工作窗口 (图 5-1) 中会以反白字形来显示原始程序中最先被执行的程序行。此刻，可以使用 HT-IDE3000 的除错命令开始进行应用程序的仿真。

注意：在应用程序的硬件仿真期间，对应的项目必须已被开启。

硬件仿真应用程序

选用 Debug 选单的 Go 命令

- 或按下快速键 F5
- 或按下工具列的 Go 按钮

在硬件仿真期间可以启动其它的窗口。如果中断条件符合的话，HT-IDE3000 系统会自动停止仿真，否则它会继续执行到应用程序结束为止。在硬件仿真时，工具列的停止钮会呈现红色，代表此时停止钮是有效的。如按下停止钮则模拟会立刻停止。停止钮在模拟停止状态下会呈现暗色，代表此时停止钮是无效的。

停止硬件仿真应用程序

共有三种停止硬件仿真的方法，如下所示：

- 开始硬件仿真之前先设定断点。
- 选用调试选单的停止命令或按下快速键 Alt+F5。
- 按下工具列的停止钮。

执行应用程序到指定的程序行

在进行程序的除错过程中，可以命令模拟动作停在指定的程序行。下列的方法提供此种功能。除了条件跳行的指令之外，从目前停在的程序行到指定程序行之间所有的指令都会被执行。由于条件跳行或其它情况，程序执行的结果也许不会停留在指定的程序行。

- 移动光标到停止的程序行 (或反白此行)。
- 选择调试选单的运行到光标处命令
 - ◆ 或按下快速键 F7
 - ◆ 或按下工具列的运行到光标处钮

直接跳跃到应用程序中的某一行

如果从目前停在的程序行到指定程序行之间所有的指令，其执行结果不影响除错效能，则可以直接跳跃至某一程序行。除了程序计数器之外，此命令不会改变数据存储器、缓存器内容和状态，下一个要被执行的指令则是所指定的程序行。

- 移动光标至要跳跃到的程序行或将此行反白。
- 选用调试选单的跳转到光标处命令。

单步执行

上面章节所描述的除错命令是检视与侦错数个程序行执行的结果，如果想要执行一程序之后就去检视其结果，则需要使用单步执行命令。HT-IDE3000 提供两种单步执行模式，手动模式与自动模式。

手动模式下，使用者每发出一个单步执行命令，HT-IDE3000 只会执行一次单步执行命令。但是在自动模式下，HT-IDE3000 则会重复执行单步执行命令直到用调试选单的停止命令让硬件仿真停止。在自动模式下，所有设定的断点皆无效，而单步执行的速率可以被设定为 FAST、0.5、1、2、3、4、和 5 秒。总共有三个单步执行命令，分别为单步执行 (进入函数)、单步执行 (越过函数) 和单步执行 (跳出函数)。

- 单步执行 (进入函数) 命令是一次只执行一个指令就停止，但是如果所执行的指令是 CALL 程序指令时，则会进入此程序并且停在程序中的第一个指令处。
- 单步执行 (越过函数) 命令是一次只执行一个指令就停止，但是如果所执行的指令是 CALL 程序指令时，则不会进入程序中，而是停留在 CALL 指令后的下一个指令处，此程序中所有的指令会被执行，而且缓存器的内容与状态也根据执行的结果做变更。
- 单步执行 (跳出函数) 命令只能在仿真停在程序之内时被使用，它会执行目前停在的程序行和 RET 指令 (包括 RET 指令) 之间所有的指令，然后停留在 CALL 指令后的下一个指令。
- 启动单步执行 (进入函数) 命令
 - ◆ 从调试选单中选择单步执行 (进入函数) 命令
 - ◆ 或按下快速键 F8
 - ◆ 或按下工具列的 Step Into 钮
- 启动单步执行 (越过函数) 命令
 - ◆ 从调试选单中选择单步执行 (越过函数) 命令
 - ◆ 或按下快速键 F10
 - ◆ 或按下工具列的单步执行 (越过函数) 钮
- 启动单步执行 (跳出函数) 命令
 - ◆ 从调试选单中选择单步执行 (跳出函数) 命令
 - ◆ 或按下快速键 Shift + F7
 - ◆ 或按下工具列的单步执行 (跳出函数) 钮

注意：Step out 命令只能被使用于目前的停止点是在程序内的情况下，否则会发生无法预期的结果。

如果要使用自动模式，则需要先使用选项选单下的工程选项，在工程选项中的调试选项属性页去设定自动模式的单步执行命令，单步执行 (进入函数) 或是单步执行 (越过函数)。

- 启动单步执行的自动模式
 - 从调试选单中选择自动单步执行命令，然后也可选择单步执行的速度 Stepping Speed(使用选项选单的调试命令去选择单步执行命令：单步执行 (进入函数) 或是单步执行 (越过函数))
- 结束单步执行的自动模式
 - 从调试选单中选择停止命令
- 改变自动模式下的单步执行命令
 - ◆ 从选项选单中选择工程选项命令
 - ◆ 在工程选项对话框中的调试选项属性页里选择单步执行 (进入函数) 或是单步执行 (越过函数)

断点

HT-IDE3000 提供一个有力的断点机制，可接受多种形式的条件，包括程序地址、原始程序行号、及符号中断等。

断点特征

HT-IDE3000 断点机制的主要特征如下：

- 任何断点被设定之后，它会被记录在断点列表盒 (Breakpoints) 内，然而此断点或许不会立即有效，但是只要它不被删除的话 (亦即仍然在断点列表盒内)，稍后可将它再设为有效的。
- 允许使用二进制型式 (含 don't care) 的地址与数据断点。
- 当指令行被设定为有效的断点时，ICE 会停在这个指令行但是不会执行此指令，亦即这个指令将是下个被执行的指令。虽然指令行被设定为有效的断点，由于执行流程或条件跳跃的缘故，ICE 可能不会停留在这个指令行。如果有有效的断点是在数据空间 (RAM)，符合断点数据的指令将会被执行，而且 ICE 会停在下一个指令行。

注意：1. ICE 最多只能同时有 3 个断点发生效应，e-ICE 则可同时有 65536 个断点发生效应。

2. ICE 可在 Free Run 的情况下设断点，e-ICE 则不支持。

断点叙述项目之说明

下列说明断点的叙述项目及内容，如图 5-3 所示，并非所有的项目皆需要设定：

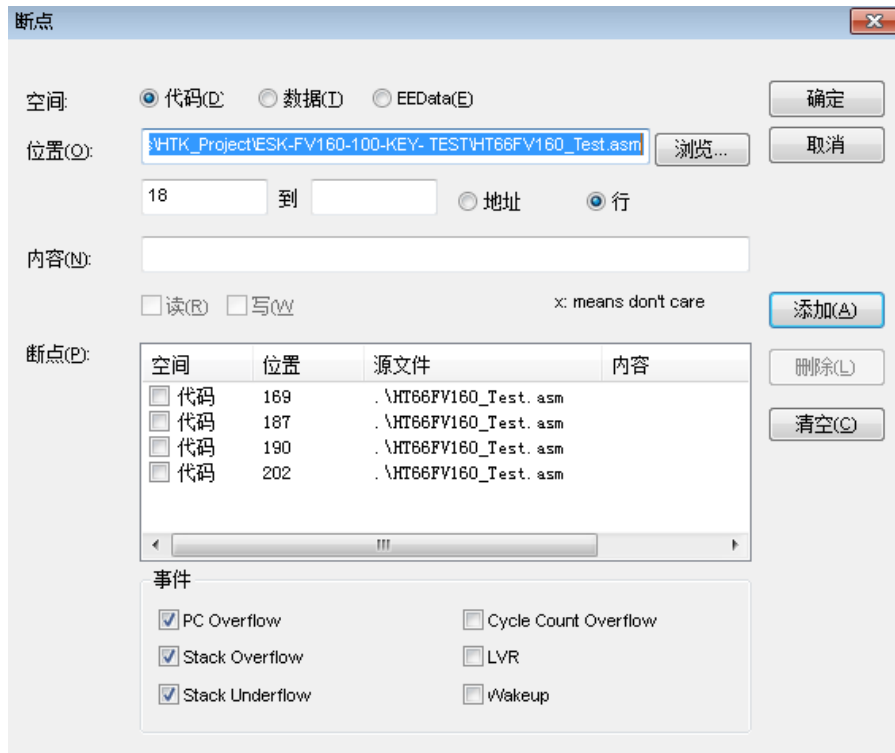


图 5-3

空间

断点的位置空间，可以设定为程序代码空间 (Code) 或资料空间 (Data) 或可编程存储器 (EEData)。

位置

断点的确实位置。可以通过 2 种不同方式来确定位置：地址 (Address) 和行号 (Line)。

行号 (Line) 方式：

- 首先输入该断点所在的文件名，该文件可以是 .C 文件或者 .ASM 文件。如果没有文名，则默认为当前文件。

- 然后输入断点所在行号：

对于 ICE 和 E-ICE，不支持一次设置多行，而是一次只能设置一行，所以只有第一个编辑框是有效的。第二个编辑框则不需要输入。

对于 E-Link，如果只要设置一行，那么第二个编辑框不需填写。如果要设置多行，那么将行号的始末填写在 2 个编辑框里。

- 除了输入行号，第一个编辑框还允许输入符号名称。

当空间为程序代码空间时，输入的符号可以是：

- ◆ Label name 标记名，如果是 C 语言，该名称为 ASM 中的名称。
- ◆ Section name 程序段的名称。
- ◆ Procedure name 函数的名称，如果是 C 语言，该名称为 ASM 中的名称。

当空间为资料空间时，输入的符号可以是：

- ◆ Dynamic data symbols defined in data section 数据段中所定义的数据变量，如果是 C 语言，该名称为 ASM 中的名称，并且必须是全局变量。

- ◆ 地址 (Address) 方式

- 该模式下，只需输入地址。
- 空间为程序代码空间时，该地址表示 ProgROM 中的地址。
- 空间为资料空间时，该地址表示 Ram 中的地址。
- 空间为 EEData 时，该地址表示 EEPROM 的地址。

内容

断点的数据内容。此项目只有当位置空间被设为数据空间时才有效，同时需要使用读和写检查盒设定数据的存取型态当做中断的执行条件。

断点

断点列表盒包含所有已加入的断点，包括有效及无效的断点。添加钮可用于增加新的断点到列表盒，删除钮则是从列表盒移除断点。

事件

只有 E-Link 支持事件，当勾选的事件发生时，E-Link 便会停下来。

注意：数据空间的断点叙述只适合 ICE，e-ICE 则不支持。

- 叙述项目的格式 – 位置

位置项目的格式如下：

共有四种型式的绝对地址 (在程序代码空间或数据空间)，名称为：十进制、十六进制 (字尾为 H 或 h，以及前缀为 0x)、二进制及全部符合的二进制位 (don't-care bits)，例如：

20, 14h, 0x14, 00010100b, 10xx0011

表示十进制的 20、十六进制的 14h/0x14、二进制的 00010100b 及都符合的位 4 和 5。

注意：Don't-care bits 必须是二进制的形式。

- 叙述项目的格式 – 内容
内容的格式是用五个数字表示，类似于位置项目的格式。这四种数字格式分别为十进制、十六进制（字尾为 H 或 h）、二进制及全部符合的二进制位（don't-care bits）。

如何设定断点

有四种方法去设定与启动断点，一种是使用调试选单的断点命令；其它种则是使用工具列的插入断点钮、双击或按 F9 在编辑窗口中的灰色条上。断点机制的规则如下：

- 如果断点没有被加到断点列表盒（图 5-3），则要先设定各叙述项目，然后再加入断点列表盒。
- 只要断点存在于列表盒中，即使先前此断点是无效的，也可以经由致能断点设为有效状态。
- 最后必须按下 OK 钮以确认设定，否则所有的更改皆无效。
- 如果要用工具列的插入断点钮去设定断点，则应先将光标移到断点所在的程序行，再按下插入断点钮，此时这个断点就被设为有效。如果要将有效的断点改变为无效的断点，则可再按下插入断点钮之改变。

增加断点

- 从调试选单中选择断点命令（或按下快速键 Ctrl+B），断点对话框会出现在窗口中（图 5-3）。
- 设定断点的各叙述项目
 - ◆ 设定空间，位置项目。
 - ◆ 如果空间是数据存储器空间，则需要设定内容项目和读 / 写检查盒。
- 按下添加钮将此断点加到断点列表盒。
- 按下 OK 钮确认设定。

注意：假如 ICE 有效断点的总数小于 3 个，则新加入的断点将会自动地生效。

删除一个断点

- 从调试选单中选择断点命令或按下快速键 Ctrl+B，断点对话框会出现在窗口中（图 5-3）。
- 从断点列表盒中选择或反白要被删除的断点。
- 按下删除钮则会将此断点从断点列表盒中删除。
- 按下 OK 钮确认设定。

删除所有断点

- 从调试选单中选择断点命令或按下快速键 Ctrl+B，断点对话框会出现在窗口中（图 5-3）。
- 从断点列表盒中选择清除所有钮以删除所有的断点。
- 按下 OK 钮确认设定。
- 也可以按下工具列的清除所有断点钮去完成这项工作。

设定一个断点为有效或无效

- 从调试选单中选择断点命令或按下快速键 Ctrl+B，断点对话框会出现在窗口中 (图 5-3)。
- 从断点列表盒选择要设定的断点。
- 按下 Enable(Disable) 钮去设定此断点为有效 (无效)。
- 按下 OK 钮确认设定。

追踪应用程序

当 HT-IDE3000 仿真应用程序时，它会应用一个有力的追踪机制，以记录执行过程与所有相关的信息。此机制并提供追踪筛选器 (qualifier) 去过滤需要被追踪的指令以及停止追踪触发器 (trigger) 以便停止记录追踪的数据。另外，它提供一种方法，可以指定在停止追踪触发点 (trigger point) 之前或之后需要储存多少笔的追踪记录。

注意：当 HT-IDE3000 开始硬件仿真时 (参考应用程序的硬件仿真章节)，追踪机制将会自动地开始记录被执行的指令与相关信息，反过来则不一定可以。

只有 ICE 支持追踪机制功能，E-ICE 与 OCDS 均不支持。

追踪机制的初步设定

使用追踪机制的基本要求，是在具有或是没有追踪筛选条件下，设定追踪模式 (Trace Mode)。追踪模式是定义程序追踪的范围，而追踪筛选条件 (qualify) 则是定义追踪记录的条件。

可使用的追踪模式有：

- Normal
此为预设模式，设定追踪的范围是全部的应用程序。
- Trace Main
除了中断服务程序之外，其它所有的应用程序皆属于追踪的范围。
- Trace INT
只有中断服务程序才属于追踪的范围。

在硬件仿真过程期间，追踪机制会根据追踪筛选条件决定那个指令和何种相关的信息应该被记录在追踪记忆器。它的原则是如果指令的相关信息与状态符合任何一个有效的追踪筛选条件，则会记录这条指令。如果需要将所有执行过的程序指令皆记录起来，只要设定为 No Qualify 即可 (也就是不要设定追踪筛选条件)，而这也是默认值。

追踪筛选条件的格式

- [source_file_name!].line_number
这里的 source_file_name 是原始档的名称，可有或没有。如果没有档名，则会以目前工作中的档案为准。如果指定了源文件名，则必须在档名之后加上惊叹号“！”，而句点“.”必须放在行数之前，格式为十进制。
例如：
C:\HIDE\USER\GE.ASM!.42
跟踪时只记录 GE.ASM 第 42 行对应代码的执行情况。
例如：
.48
跟踪时只记录当前文档第 48 行对应代码的执行情况。

- [source_file_name!].symbol_name
除了将 line_number 换为 symbol_name 外，所有的格式及内容皆与上述的行号数字位置格式相同。下列的程序符号皆可被接受：
 - ◆ Label name 标记名
 - ◆ Section name 程序段的名称
 - ◆ Procedure name 程序的名称
 - ◆ Dynamic data symbols defined in data section 数据段中所定义的数据变量

相对于 Trace Mode 与 Qualify 是设定记录的条件，Trigger Mode 和 Forward Rate 则是指定停止追踪记录的条件。

Trigger Mode 指定停止追踪触发点的种类，它也可以用来决定停止追踪点的位置。Forward Rate 则是设定追踪范围的比率，此追踪范围是落在触发点与停止追踪点之间。

可使用的 Trigger Mode 如下：

- No Trigger
此为默认值，没有设定停止追踪的条件。
- Trigger at Condition A
触发点设在条件 A。
- Trigger at Condition B
触发点设在条件 B。
- Trigger at Condition A or B
触发点设在条件 A 或条件 B 两者之一。
- Trigger at Condition B after A
触发点设在条件 B 但是要在条件 A 发生之后。
- Trigger when meeting condition A k times
触发点是当条件 A 符合 k 次时。
- Trigger at Condition B after meeting A k times
触发点是在条件 B 但是要在条件 A 符合 k 次后。

条件 A 和条件 B 设定触发条件，其格式和追踪筛选条件的格式相同。

Loop Count 用来设定条件 A 发生的次数，只有当 Trigger Mode 选用上述最后两个模式之一时才需要设定此项数字。

Forward Rate 是用来设定追踪记录段 (trace record) 占用追踪记忆区 (trace buffer) 的百分比，此段追踪记录是介于停止追踪触发点 (trigger point) 与停止追踪点 (stop trace point) 之间的记录。从另外一个角度来看，可以先行设定停止追踪触发点，再根据想要得到多少追踪资料，也就是停止追踪触发点之后的数据，去设定 Forward Rate 百分比。停止追踪触发点将追踪记忆区划分为触发点之前与之后两部分，Forward Rate 设定触发点之后的百分比，以便限制追踪记录的范围，百分比可设定在 0 至 100% 之间。

注意：追踪记录的数据笔数不一定会刚好等于 Forward Rate。例如在追踪记录尚未达到 Forward Rate 指定的百分比之前，碰到断点而停止模拟或是使用追踪停止命令 (参考停止追踪机制的章节)，这些都会停止追踪记录，因此造成追踪记录与 Forward Rate 的不一致。

Qualify 列表盒记载及显示被 Trace Mode 使用到的追踪筛选条件。列表盒中可以加入高达 20 个筛选条件并且最多有 6 个筛选条件是同时有效的。可以将筛选条件设为无效或是将之从列表盒中删除。在 Qualify 列表盒中。

追踪筛选条件的格式

```
<status> {<space and read/write>, <location>,
<data content>, <external signal>}
```

这里的 <status> 是有效状态，“+”表示有效(致能) 而“-”表示无效(除能)，<space and read/write> 是空间型式及操作模式，“C”是程序代码内存空间，“D/R”是从数据存储器空间读取，“D/W”是写入数据存储器空间，“D/RW”是读取与写入数据存储器空间。

<location>、<data content> 及 <external signal> 则与各别的输入格式相同。

停止追踪机制

有三种方法停止追踪记录机制：

- 如上述，设定停止追踪触发点 (Trigger Mode) 和 Forward Rate 百分比
- 设定断点去停止硬件仿真与追踪记录
- 从调试选单 (图 5-2) 发出 Trace Stop 命令去停止追踪记录

图 5-4 列出使用追踪机制的必要数据，此对话框是执行调试选单中跟踪命令的结果。

追踪机制的启动与停止

- 设定追踪模式
- 从调试选单选择跟踪命令
跟踪对话框显示如图 5-4。
- 从 Trace Mode 下拉式列表盒选择追踪模式
- 按下 OK 按钮确认

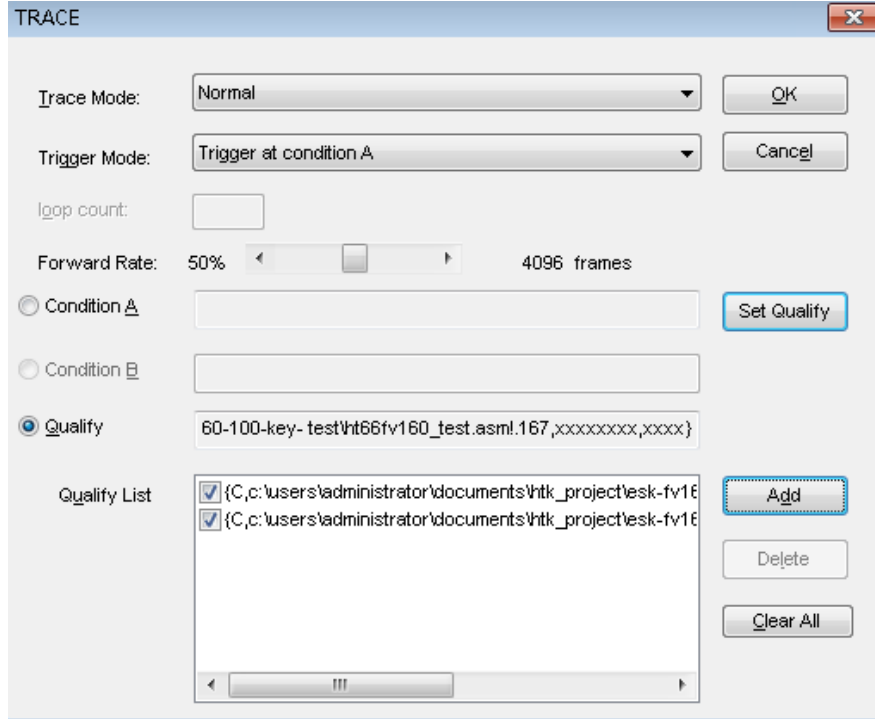


图 5-4

设定停止追踪的触发模式

- 从调试选择跟踪命令
跟踪对话框会显示如图 5-4。
- 从 Trigger Mode 下拉式列表盒选择触发模式
- 按下 OK 按钮确认

改变 Forward Rate

- 从调试选择跟踪命令
跟踪对话框显示如图 5-4。
- 使用 Forward Rate 滚动条去指定所要的百分比
- 按下 OK 按钮确认之

设定条件 A 或条件 B

- 从调试选择跟踪命令
跟踪对话框显示如图 5-4。
- 按下 Condition A 或 Condition B 传送钮
- 按下 Set Condition 按钮
Set Qualify 对话框显示如图 5-5
- 键入条件 A 或条件 B 的资料
- 按下确定按钮，关闭 Set Condition 对话框
- 按下 OK 按钮，关闭 Trace 对话框



图 5-5

加入追踪的筛选条件

- 从调试选单选择跟踪命令
跟踪对话框显示如图 5-4。
- 按下 Qualify 传送钮
- 按下 Set Qualify 按钮
Set Qualify 对话框显示如图 5-5
- 键入筛选条件的资料
- 按下确定按钮，关闭 Set Condition 对话框
- 按下 Add 按钮，将筛选条件加入 Qualify List 盒之内
- 按下 OK 按钮，关闭 Trace 对话框

删除追踪的筛选条件

- 从调试选单选择跟踪命令
跟踪对话框显示如图 5-4。
- 从 Qualify List 盒中选择要被删除的筛选条件
- 按下 Delete 钮
- 按下 OK 钮确认

删除所有的追踪筛选条件

- 从调试选单选择跟踪命令
跟踪对话框显示如图 5-4。
- 按下 Clear All 钮
- 按下 OK 钮确认

注意：如果没有设定任何追踪筛选条件，则所有指令将被内定为符合筛选条件。

设定追踪筛选条件为有效或无效

- 从调试选择跟踪命令
跟踪对话框显示如图 5-4。
- 从 Qualify List 盒中选择要被设定的筛选条件
- 按下 Enable(Disable) 钮
- 按下 OK 钮确认

注意：最多可同时设定 6 个追踪筛选条件为有效。

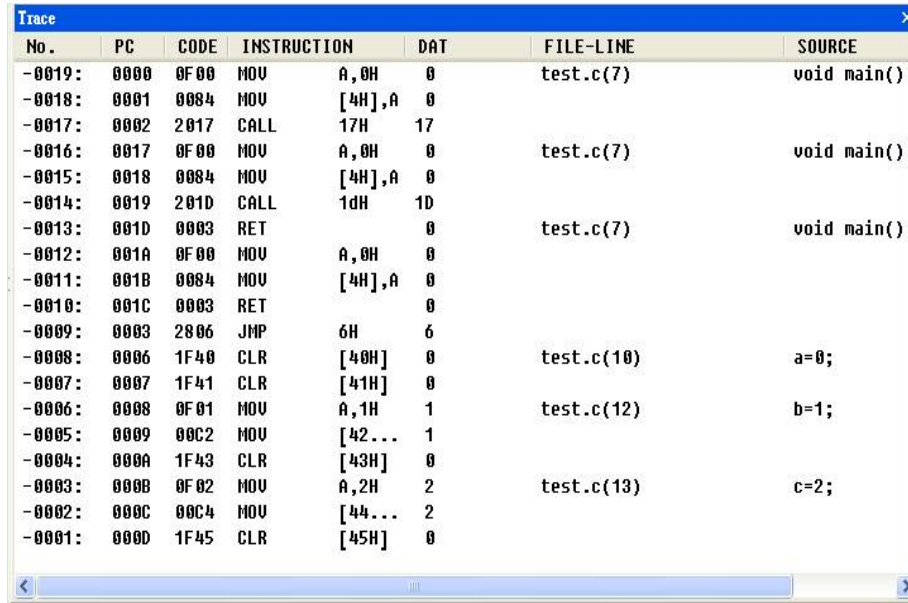
追踪记录的格式

在完成追踪筛选条件和停止追踪触发条件的设定之后，则所有符合筛选条件的指令将被记录在追踪记忆区内。使用窗口选单的跟踪记录表命令可以查验追踪记录数据，帮助程序的除错。当窗口中显示追踪记录时，除了记录编号和反汇编指令一定会显示，追踪记录内其它的数据并不一定皆会显示出来，这些字段的显示与否取决于选项选单下的工程选项中调试选项属性页里关于跟踪记录区的设定。在下面的说明标题文字中，用小括号所包含的文字会出现在窗口选单的跟踪记录表命令标题上，代表各字段，例如记录编号在跟踪列表盒的标题上是以 No. 来代表。图 5-6 和图 5-7 是在不同的调试选项下所显示之追踪列表的内容。

No.	PC	CODE	INSTRUCTION	DAT	FILE-LINE	SOURCE
-0019:	0000	MOV	A, 0H	0		
-0018:	0001	MOV	[4H], A	0		
-0017:	0002	CALL	17H	17		
-0016:	0017	MOV	A, 0H	0		
-0015:	0018	MOV	[4H], A	0		
-0014:	0019	CALL	1dH	1D		
-0013:	001D	RET		0		
-0012:	001A	MOV	A, 0H	0		
-0011:	001B	MOV	[4H], A	0		
-0010:	001C	RET		0		
-0009:	0003	JMP	6H	6		
-0008:	0006	CLR	[40H]	0		
-0007:	0007	CLR	[41H]	0		
-0006:	0008	MOV	A, 1H	1		
-0005:	0009	MOV	[42...]	1		
-0004:	000A	CLR	[43H]	0		
-0003:	000B	MOV	A, 2H	2		
-0002:	000C	MOV	[44...]	2		
-0001:	000D	CLR	[45H]	0		

图 5-6

- 记录编号 (No.)
不论何种停止追踪触发模式，触发点的记录编号均为 +0，在触发点之前的追踪记录使用负数的记录编号，在触发点之后的追踪记录则使用正数的记录编号。如果追踪记录中所有的字段 (Options 选单下的 Project Settings 中 Debug option 属性页里的 Trace Recoord Fields) 皆被圈选，则数据显示的结果如图 5-7。如果选择 No trigger mode 或是还未执行到触发点，则记录编号是从 -00001 开始往回以每件追踪记录减 1 的数字编列 (图 5-6)。
 - 程序计数器 (PC)
追踪记录中指令的程序计数器。
 - 机器码 (CODE)
指令的机器码。
 - 反汇编指令 (INSTRUCTION)
反汇编指令是执行 HT-IDE3000 的反组译程序所得的结果。
 - 执行数据 (DAT)
被执行的数据内容 (读或写)。
 - 带有行号数字的源文件名称 (FILE-LINE)
源文件名称和此指令的行号数字。
 - 源文件 (SOURCE)
原始行列陈述 (包括符号)。
- 除了记录编号和反汇编指令一定会显示之外，上述中所有的字段均非必须的。



No.	PC	CODE	INSTRUCTION	DAT	FILE-LINE	SOURCE
-0019:	0000	0F00	MOV A,0H	0	test.c(7)	void main()
-0018:	0001	0084	MOV [4H],A	0		
-0017:	0002	2017	CALL 17H	17		
-0016:	0017	0F00	MOV A,0H	0	test.c(7)	void main()
-0015:	0018	0084	MOV [4H],A	0		
-0014:	0019	201D	CALL 1DH	1D		
-0013:	001D	0003	RET	0	test.c(7)	void main()
-0012:	001A	0F00	MOV A,0H	0		
-0011:	001B	0084	MOV [4H],A	0		
-0010:	001C	0003	RET	0		
-0009:	0003	2806	JMP 6H	6		
-0008:	0006	1F40	CLR [40H]	0	test.c(10)	a=0;
-0007:	0007	1F41	CLR [41H]	0		
-0006:	0008	0F01	MOV A,1H	1	test.c(12)	b=1;
-0005:	0009	00C2	MOV [42...]	1		
-0004:	000A	1F43	CLR [43H]	0		
-0003:	000B	0F02	MOV A,2H	2	test.c(13)	c=2;
-0002:	000C	00C4	MOV [44...]	2		
-0001:	000D	1F45	CLR [45H]	0		

图 5-7

注意：使用 Options 选单下的 Project Settings 中 Debug option 属性页里的 Trace Record Fields 去选择追踪记录的字段。

使用 Window 选单中的 Trace List 命令去查看追踪记录各字段的内容。

- 清除追踪记忆区的内容

使用 Debug 选单的 Reset Trace 命令将追踪记忆区的内容清除，之后追踪信息将由追踪记忆区的开始端储存起。另外，Reset 命令和 Power-On Reset 命令也会清除追踪记忆区的内容。

第六章 选单 – 窗口

HT-IDE3000 提供许多种类的窗口，以便帮助使用者对应用程序进行硬件或软件仿真，这些窗口 (如图 6-1 所示) 包括数据存储器 (RAM)、程序内存 (ROM)、追踪列表、缓存器内容、变量检测、堆栈、还原程序、命令执行等数据窗口。

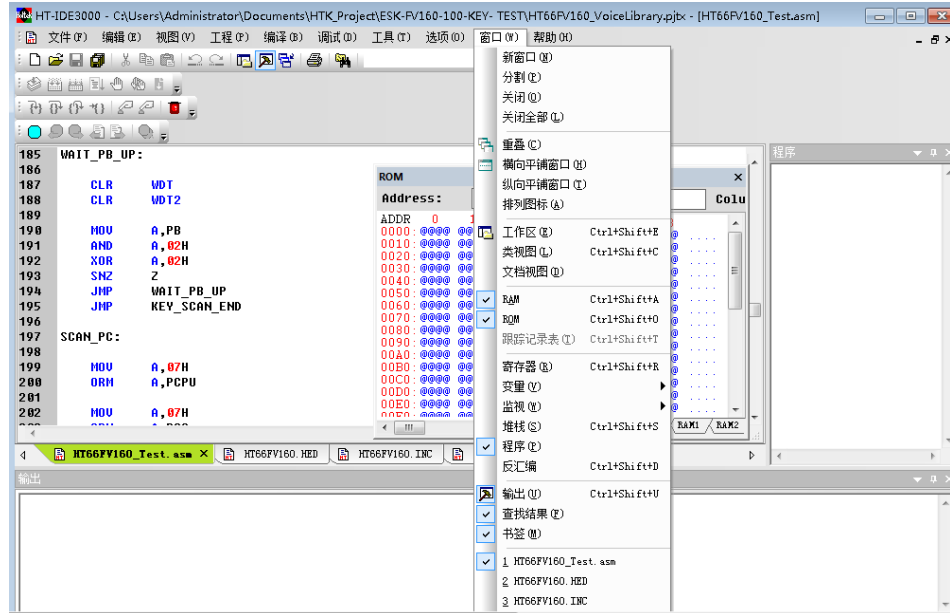


图 6-1

窗口选单命令

工作区

Workspace 窗口陈列出项目中所有程序代码档案，如 6-2；使用者在此窗口中可快速敲击开启档案，也可使用鼠标右键新增与移除档案。

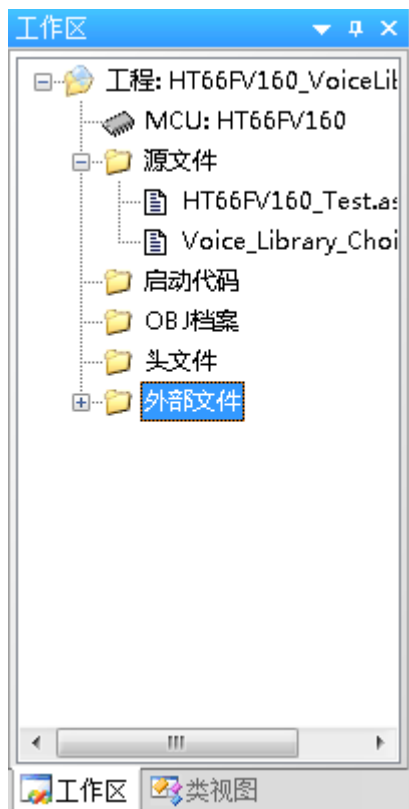


图 6-2

RAM

RAM 窗口会显示数据存储器空间的内容，如图 6-3 所示。缓存器的地址空间并不包含于 RAM 窗口中，因为它会显示在缓存器窗口中。RAM 窗口的内容可以因除错的目的而直接被修改，垂直边显示的地址是基本地址而水平方向各数字则是地址偏移量，所有的数字均以十六进制格式显示，左边 Address 栏可以输入地址进行直接定位，右边的 Column 可以选择以多少列数显示。

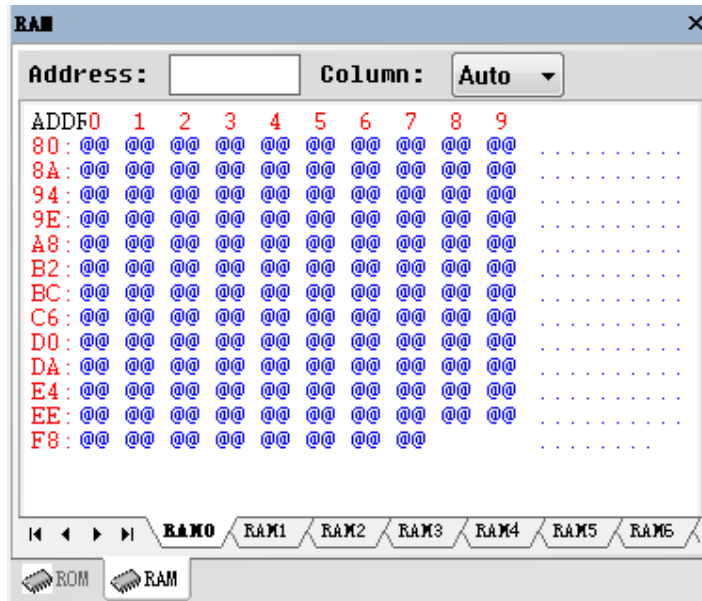


图 6-3

ROM

ROM 窗口会显示程序内存空间的内容，如图 6-4 所示。ROM 地址范围是从 0 到最大的程序地址，此最大地址取决于在项目中所选取的微控制器。水平与垂直滚动条可以用来查看任何 ROM 窗口中的地址，ROM 窗口的内容以十六进制格式显示并且不可以被修改。

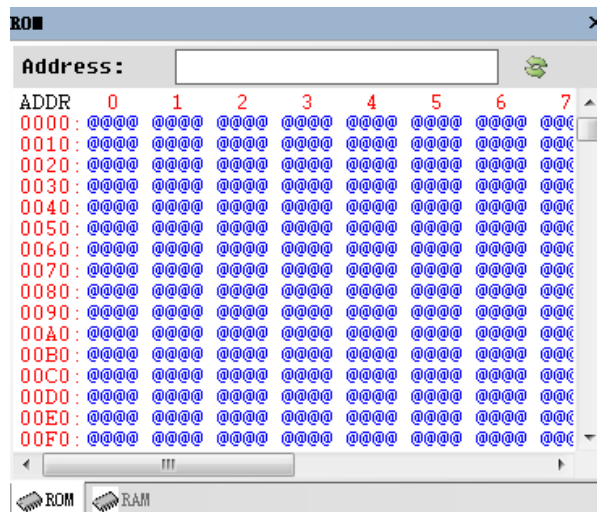


图 6-4

跟踪记录表

跟踪记录表窗口会显示追踪记录的信息，如图 6-5 所示。追踪记录的内容可以使用选项选中调试命令定义之。使用鼠标双击跟踪记录表窗口中的追踪记录会将此行所在之原始程序文件开启并且放置于工作的源文件窗口中，光标停止在对应的行列上。

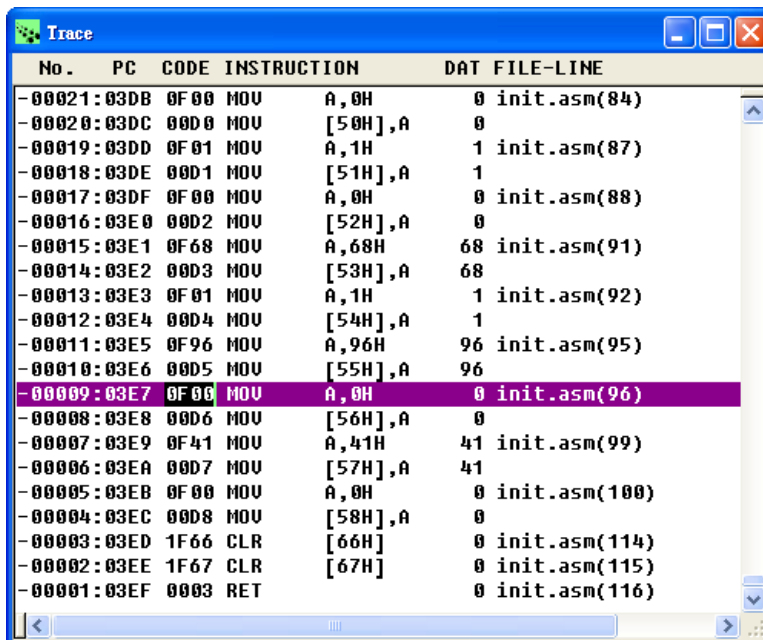


图 6-5

寄存器

寄存器窗口会显示所有被定义在项目中所选取的微控制器内之缓存器内容，图 6-6 所示为 HT48C70-1 的缓存器窗口的范例，寄存器窗口的内容可被修改，此窗口是可随处停留的，可以在主窗口内将之到处移动。

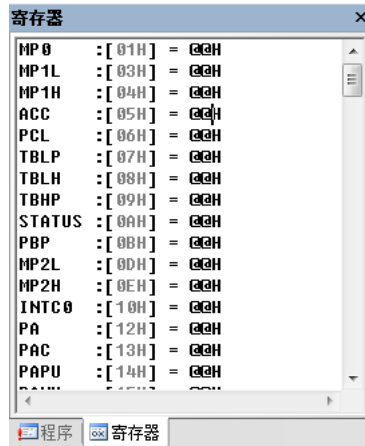


图 6-6

监视

监视窗口会显示所指定符号的内存地址和内容，此符号是定义在数据存储器，即 RAM 空间中的。

- 可以显示缓存器的内容，只要键入符号名称或缓存器名称再按下 Enter 键。指定的符号或缓存器的内存地址及内容将被显示于符号的右边。可以显示指定符号所属的空间，Ram 或 ROM。

- 地址以十六进制格式显示，数据可以显示十、十六、二进制格式，如图 6-7 所示。符号与相对应的数据会被 HT-IDE3000 所储存并且在下一次打开监视窗口时显示出来。按下 Delete 键可将符号从监视窗口中删除，监视窗口是可随处移动的。

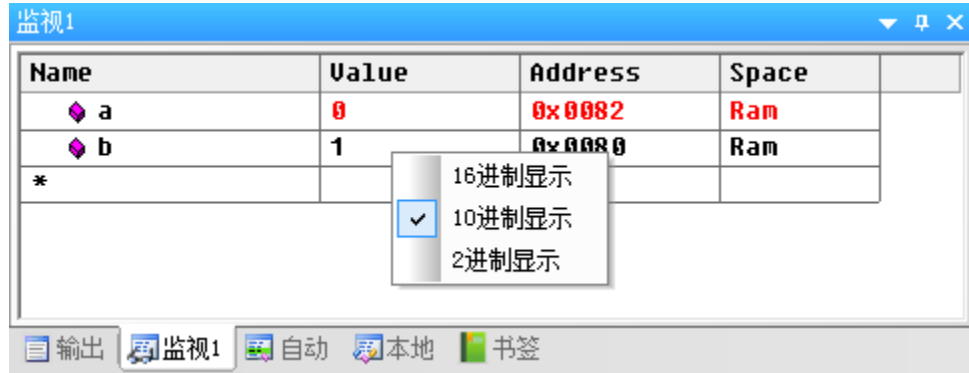


图 6-7

堆栈

堆栈窗口显示当前工程中所选之微控制器堆栈缓冲器的内容，最大的堆栈层次是依照所选的微控制器的规格而定。图 6-8 显示一个堆栈窗口的例子。堆栈是从 0 开始往上增加层次，每个推入运算会使数字加 1(例如 CALL 指令或中断事件)，而每个取出运算则使它减 1(RET 或 RETI 指令)。左边的箭头指示最上层的堆栈，并且文字以反白标示。堆栈缓冲器中没有用到的部分以“@@@@”表示。例如图 6-8 中 01 是最上层的堆栈列。当执行一个 RET 或 RETI 指令时，标示在最上层的程序行数(此例中为 26)被视为下一个要执行的指令，同样在堆栈列的最上层的再上一列(此例中为 00)将被当成新的最上层堆栈列。如果堆栈中没有任何堆栈列，则堆栈窗口中将没有任何列会被特别标示。此外，ICE STKPTR 为 ICE 内部堆栈指针指向下一层堆栈，若出现 ICE STKPTR=00 Stack Status:E/E 则代表堆栈为空或刚好全满的状况。堆栈列的格式如下：

```
Stack_level: program_counter source_file_name(line_number)
```

此处 stack_level 是堆栈的层数，program_counter 为呼叫程序的十六进制回复地址或被中断时的指令地址，source_file_name 是包含此呼叫程序或被中断的指令的源文件名称，而 line_number 则是在源文件中呼叫指令的程序行数或被中断的指令的程序行数，line_number 是十进制数。

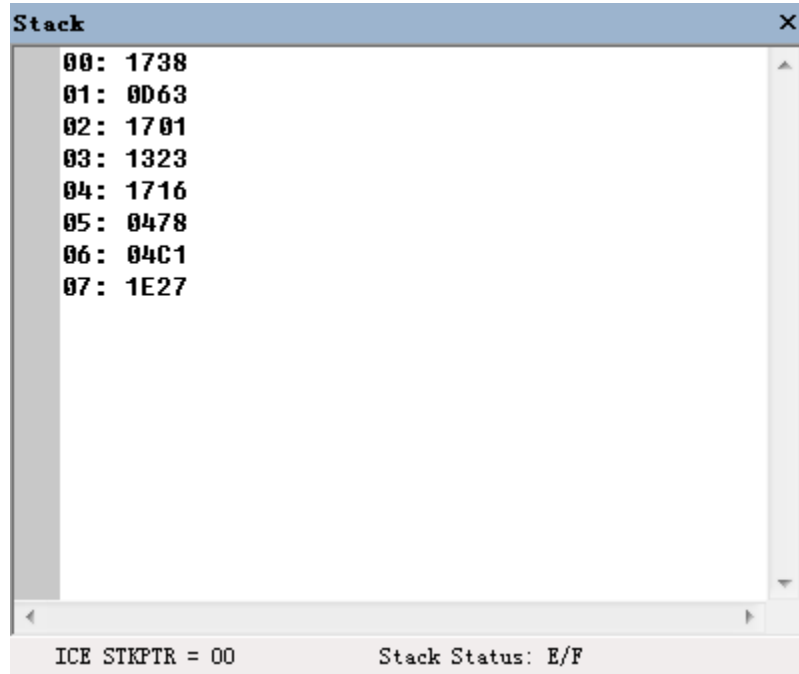


图 6-8

自动

此窗口可以检视和修改变量的数值。包含二个标签：

- Auto 卷标，观察和设置与当前函数相关的局部和全局变量。
- Local 卷标，观察和设置当前函数内的局部变量。

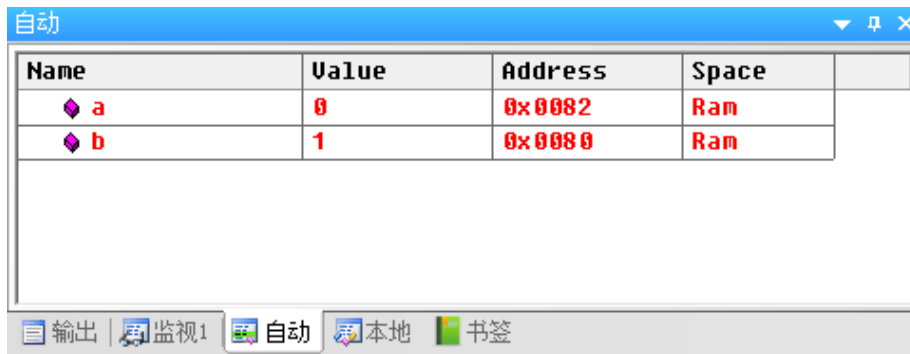


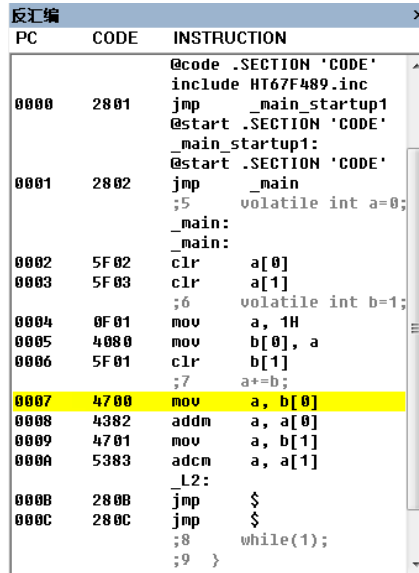
图 6-9

程序

程序窗口会以反组译的格式显示程序内存或是 ROM 的内容。地址范围是从 0 到最大的地址为止，此最大的地址将由项目中所选取的微控制器决定。

反汇编

在除错模式中，此窗口会显示原始码相对应的汇编语言。



PC	CODE	INSTRUCTION
0000	2801	@code .SECTION 'CODE' include HT67F489.inc jmp _main_startup1 @start .SECTION 'CODE' _main_startup1: @start .SECTION 'CODE'
0001	2802	jmp _main ;5 volatile int a=0; _main: _main:
0002	5F02	clr a[0]
0003	5F03	clr a[1] ;6 volatile int b=1;
0004	0F01	mov a, 1H
0005	4080	mov b[0], a
0006	5F01	clr b[1] ;7 a+=b;
0007	4700	mov a, b[0]
0008	4382	addm a, a[0]
0009	4701	mov a, b[1]
000A	5383	adcm a, a[1] _L2:
000B	280B	jmp \$
000C	280C	jmp \$;8 while(1); ;9 }

图 6-10

输出

输出窗口显示执行构建或重建所有命令后的结果讯息。在错误讯息列双击鼠标左键，会出现包含源文件的窗口以及错误所在的程序行会特别标示出来。

校验和与校验码。执行构建或重建所有命令后，会输出用户程序的校验和 (Checksum) 跟校验码 (Verify Code)，可以方便用户知道程序是否被改变。

第七章 汇编语言和组译器

原始程序文件是以汇编程序所构成的，经由组译器 (Assembler) 编译成目的档案 (Object File)，再经由连结器 (Linker) 结合并产生工作档案 (Task file)。

原始程序 (source program) 是由陈述式 (statement) 和查询表 (look up table) 所组成的，用在组译器进行编译或程序执行时给予指示之用，而陈述式是由助忆符号 (mnemonic)、操作数 (operand) 和批注 (comment) 所组成。

惯用符号

下表描述文章中所用到的惯用符号

范例	描述
[optional items]	中括号内的项目是可选择的，下列的命令列语法中： <code>HASM [options] filename [;]</code> [options] 和 [;] 都是可选用的，而 filename 则必须设定。但是在指令操作数中的中括号则是指定内存地址之用，必须要有。
{choice1 choice2}	大括号和垂直线代表两个或更多的选项，大括号圈出选项而垂直线则用来分隔选项，只能有一个选项被选出。
Repeating elements...	三个连续的点表示允许输入更多同样形式的数据，例如下列的指令格式： <code>PUBLIC name1 [,name2 [,...]]</code> name2 之后的三个连续点表示允许输入更多的名称，只要每两个名称之间以逗号隔开即可。

语句语法

陈述式的文法格式如下：

[名称] [操作元] [操作数] [批注]

上述四个成员不一定都要指定。

每两个成员之间 (除了批注) 最少要以一个空格或一个 tab 符号分隔开。

成员的字型是无大小写之分，换言之，组译器在编译之前会将小写字母改为大写字母。

名称

陈述式前可有标记以便于其它陈述式使用，如果名称当做标记使用，则必须在名称后紧接一个冒号 (:)。名称是由下列字符符号所组成：

A~Z a~z 0~9 ? _ @

在使用上有以下的限制：

- 不可使用数字 0~9 作为名称的第一个字符符号。
- ? 不能单独当做名称。
- 只有最前面的 31 个字符符号被认可。

操作元

操作元定义两种型态的陈述式，假指令与指令。假指令用来指导组译器如何在编译时产生目的码，指令则是引导微控制器执行各种运算，两者都会在编译时产生目的码，而指令被编译出的目的码会在执行时指导微控制器的运作。

操作数

操作数定义假指令与指令所使用的数据，由符号、常数、表达式 (expression) 和缓存器所组成。

批注

批注是对程序代码的一种叙述与说明，组译器不会编译它。任何在分号 (semicolon;) 之后的文字均被视为批注。

组译假指令

组译假指令用来指导组译器如何在编译时产生目的码，组译假指令可以依其行为细分如下。

条件编译假指令

条件区段的格式如下：

```
IF
statements
[ELSEIF
Statements]
[ELSE
statements]
ENDIF
```

语法

```
IF expression
IFE expression
```

说明

假指令 IF 和 IFE 对其后的 expression 进行检测。

如果 expression 的数值为真，换言之不为零，则在 IF 与 ELSE 或 IF 与 ENDIF (没有 ELSE) 之间所有的陈述式会被编译。

如果 expression 的数值为假，换言之为零，则在 IFE 与 ELSE 或 IFE 与 ENDIF (没有 ELSE) 之间所有的陈述式会被编译。

范例

```
IF debugcase
ACC1 equ 5
extern username: byte
ENDIF
```

在此范例中，如果符号 debugcase 的数值不为零，则变量 ACC1 的数值将被设定为 5 同时 username 被宣告为外部变量。

语法

```
IFDEF name
IFNDEF name
```

说明

IFDEF 和 IFNDEF 的差异在检测 name 是否被定义，只要 name 已在前面定义为标记、变量或符号，则在 IFDEF 与 ENDIF 之间的陈述式皆会被编译，相反的情况下如果 name 还未被定义，则在 IFNDEF 与 ENDIF 之间的陈述式会被编译，条件组译假指令提供最多 7 层的巢状架构。

范例

```
IFDEF buf_flag
    buffer DB 20 dup (?)
ENDIF
```

在此范例中，只要 `buf_flag` 被事先定义，即配置内存给 `buffer`。

语法

```
IF DEFINE_EXP
ELSEIF DEFINE_EXP
IFE DEFINE_EXP
```

说明

(1). `DEFINE_EXP` 支持以下语法：

```
DEFINED name [ | DEFINE_EXP]
DEFINED name [ & DEFINE_EXP]
!DEFINED name [ | DEFINE_EXP]
!DEFINED name [ & DEFINE_EXP]
```

(2). 如果 `name` 有被定义，则 `DEFINED name` 为真，否则为假

(3). 多个 `DEFINED name` 可用 `&(AND)`, `| (OR)` 运算符连接

(4). `IF !DEFINED name` 相当于 `IFE DEFINED name`.

范例

```
__V3__ EQU 3
IF (DEFINED __V1__ | DEFINED __V3__)
NOP
ENDIF
```

在此例中，`NOP` 会被执行。

档案控制假指令

语法

```
INCLUDE file-name
```

或

```
INCLUDE file-name
```

说明

此假指令会在编译时，将含入档 `file-name` 的内容，嵌入至目前的原始码档案，并被视为原始码。组译器提供最多 7 层的巢状架构。

范例

```
INCLUDE macro.def
```

在此范例中，组译器将含入档 `macro.def` 内的原始码，嵌入至目前的原始码档案。

语法

```
PAGE size
```

说明

此假指令指定程序行表文件 (`program listing file`) 中每一页的行数大小，其范围介于 10 行至 255 行之间，组译器的内定值为 60 行。

范例

```
PAGE 57
```

在此范例中，程序行表文件的每一页最多为 57 行。

语法

```
.LIST  
.NOLIST
```

说明

假指令 `.LIST` 和 `.NOLIST` 用来决定是否要将原始程序行储存到程序行表档 (program listing file)。 `.NOLIST` 会停止将其后的原始程序存写到程序行表档，而 `.LIST` 则会将其后的原始程序行存写到程序行表档。组译器的内定值为 `.LIST`。

范例

```
.NOLIST  
mov a, 1  
mov b1, a  
.LIST
```

上面的范例中，被 `.NOLIST` 和 `.LIST` 所包围的两个指令将不会存写到程序行表档。

语法

```
.LISTMACRO  
.NOLISTMACRO
```

说明

假指令 `.LISTMACRO` 会引导组译器将宏假指令中包括批注的所有原始陈述式存写到程序行表档案中。假指令 `.NOLISTMACRO` 则中止写入所有宏假指令的原始陈述式。组译器内定值为 `.NOLISTMACRO`。

语法

```
.LISTINCLUDE  
.NOLISTINCLUDE
```

说明

`.LISTINCLUDE` 会引导组译器将所有含入档 (included files) 的内容写入程序行表档案中，而假指令 `.NOLISTINCLUDE` 则会引导组译器不要将其后的含入档内容写进程序行表档案中。组译器的默认值为 `.NOLISTINCLUDE`。

语法

```
MESSAGE 'text-string'
```

说明

假指令 `MESSAGE` 指导组译器将 `text-string` 显示于屏幕上，`'text-string'` 的字符必须使用一对单引号包含之。

语法

```
ERRMESSAGE 'error-string'
```

说明

假指令 `REEMESSAGE` 指导组译器显示错误讯息，`'error-string'` 的字符必须使用一对单引号包含之。

程序假指令

语法 (批注)

```
; text
```

说明

批注是以分号 (semicolon) 开始的字符所组成，以 CR/LF 当做结束。

语法

```
name .SECTION [align] [combine] 'class'
```

说明

假指令 `.SECTION` 用来标示程序段 (program section) 或数据段 (data section) 的开始，程序段开始地址的型态与程序段连结的方式等。程序段是由指令或指令加上数据所组成，这些指令及数据的地址则是以该程序段的段名 `name` 为起始标准而定出的。

程序段的段名 `name` 可以是唯一的或者是与程序中其它区段的段名相同。若两个程序段设定有完全相同的名称 (complete name)，则被视为是同一个程序段。

完全相同的名称是表示任何两个程序段的段名 `name` 及类别名 `class` 皆相同。

选项 `align` 则定义程序段开始地址的型态，可选用下列中的一种：

BYTE 以任意字节地址 (byte) 当做开始地址 (组译器的内定地址型式)

WORD 以字符地址 (word, 两个字节, 或偶数地址) 当做开始地址

PARA 以节段地址 (paragraph, 16 的倍数) 当做开始地址

PAGE 以分页地址 (page, 256 的倍数) 当做开始地址

针对 **CODE** 类别 (class) 的程序段，是以一个指令当做一个字节地址。**BYTE** 会将程序段的开始地址安排在任何指令的地址，**WORD** 则将程序段的开始地址安排在偶数的指令地址，**PARA** 将程序段的开始地址安排在 16 倍数的指令地址，而 **PAGE** 则将程序段的开始地址安排在 256 倍数的指令地址。

对于 **DATA** 类别的程序段而言，是以一个字节 (8 位 / 字节) 当做地址的计算单位。**BYTE** 会将程序段 (此时应为数据段) 的开始地址安排在任何字节地址，**WORD** 会将数据段的开始地址安排在偶数地址，**PARA** 则将数据段的开始地址安排在 16 倍数的地址，而 **PAGE** 会将数据段的开始地址安排在 256 倍数的地址。

选项 `combine` 定义如何结合有完全相同名称的程序段的方法，可选用下列中的一种：

- **COMMON**

将具有完全相同名称的所有程序段的开始地址安排在同一地址，所使用的内存长度则是以最长的程序段的长度为准。

- **AT address**

此选项是指定程序段的开始地址为 `address`，一个固定地址。组译器及连结器不能将之安排到其它的地址，而其内的标记 (label) 和变量 (variable) 的地址可直接从 `address` 计算出。除了不可有前置引用 (forward reference) 的变量或符号外，可以使用任何合乎规定的表达式来表示 `address`，而运算结果的数值必须是合法的 ROM/RAM 内存地址，且不能超出 ROM/RAM 的大小范围。

如果没有设定 `combine` 的型式，则此程序段是可结合的，换句话说，此程序段和其它具有完全相同名称的程序段可以连结成一个单一的程序段。这些具有完全相同名称的程序段可以分别定义在不同的原始程序文件。

Class 则是定义程序段的类别。相同类别的程序段被安排在内存中的连续区域，以其输入的先后顺序一个个紧接地安排在内存中。类别名称为 **CODE** 的程序段将会放置于程序内存 (program memory; ROM)，而类别名称为 **DATA** 的程序段 (应为数据段) 则是储存在数据存储器 (data memory; RAM)。在此假指令之后，直到下一程序段假指令之前的所有指令及数据，皆属于此程序段。如果没有下一个程序段假指令，则一直到程序文件的结尾都属于此程序段。

语法

```
ROMBANK banknum section-name [,section-name,...]
```

说明

此假指令是用来宣告程序内存 (program memory) 的某一区块 (bank) 所包含的程序段。banknum 指定程序内存的区块编号, 范围从 0 到微控制器最大的程序内存区块数。section-name 则是先前已定义的程序段名称, 可以在同一个内存区块内宣告不只一个程序段, 只要这些被宣告的程序段的总和不超过 8K。如果程序中没有宣告此假指令, 则所有类别为 CODE 的程序段皆被视为属于区块 0 (bank 0), 如果某个类别为 CODE 的程序段没有被宣告属于任何程序内存的区块内, 此程序段将被视为属于区块 0。

语法

```
RAMBANK banknum section-name [,section-name,...]
```

说明

此假指令与 ROMBANK 相似, 相异的地方是宣告数据存储器 (data memory) 的区块所包含的数据段 (data section)。数据存储器区块的大小则为 256 字节。

语法

```
END
```

说明

此假指令宣告程序的结束, 因此最好不要在含入档 (included file) 中加入这个指令。避免组译器编译到此假指令后就结束程序的编译流程, 之后的指令及假指令就不会被编译。

语法

```
ORG expression
```

说明

此假指令会将 expression 的计算数值设定给组译器的地址计数器 (location counter), 其后之程序代码和数据地址将根据 expression 所计算的偏移量做相对的调整。程序代码和数据偏移量与假指令 ORG 所在的程序段的开始地址有关, 程序段的属性会决定偏移量的实际值 (是绝对地址或相对地址)。

范例

```
ORG 8
mov A, 1
```

在此范例中, 指令 mov A, 1 的地址是在程序段的第 8 个地址。

语法

```
PUBLIC name1 [,name2 [,...]]
EXTERN name1:type [,name2:type [,...]]
```

说明

假指令 PUBLIC 用来宣告可被其它程序文件中的程序所使用的变量或标记, 也就是公用变量或标记。另一方面, 假指令 EXTERN 则用来宣告程序将使用的外部变量、标记或符号。可使用下列四种形式中的一种: BYTE、WORD、BIT(这三种形式适用于数据变量) 和 NEAR(标记形式, 用于呼叫或跳越)。

范例

```
PUBLIC start, setflag
EXTERN tmpbuf:byte
CODE .SECTION `CODE`
```

```

start:
    mov    a, 55h
    call  setflag
    ...
setflag proc
    mov    tmpbuf, a
    ret
setflag endp
end
    
```

在此范例中，标记 `start` 和程序 `setflag` 都被宣告为公用变量，而其它原始程序文件中的程序可以使用这些变量。变量 `tmpbuf` 被宣告为外部变量，则需要有一个名称为 `tmpbuf` 的变量定义在其它的原始程序文件、而且被宣告为公用变量。

语法

```

name PROC
name ENDP
    
```

说明

假指令 `PROC` 和 `ENDP` 用来定义一段可被其它程序呼叫或跳入的程序代码。必须要指定一个名称 `name` 给 `PROC` 代表此程序 (procedure) 第一个指令的地址，而组译器会将标记的值设定至地址计数器中。

范例

```

toggle PROC
mov    tmpbuf, a
mov    a, 1
xorm   a, flag
mov    a, tmpbuf
ret
toggle ENDP
    
```

语法

```
[label:] DC expression1 [,expression2 [,...]]
```

说明

假指令 `DC` 会将 `expression1` 及 `expression2` 等的值储存在内存的连续位置里，此假指令只能使用于 `CODE` 类别的程序段之内。`expression1` 及 `expression2` 计算的数值将视微控制器的程序内存的宽度大小而定，组译器会将任何多余的位清除掉，`expression1` 必须为数值或标记，此假指令通常被用在程序段之内建立表格以便查询。

范例

```
table: DC 0128H, 025CH
```

在此范例中，组译器会预留两个地址的程序内存空间、并将 `0128H` 和 `025CH` 储存至这两个地址中。

数据定义假指令

汇编程序是由陈述式和批注所组成的，一行陈述式或批注则是由字符、数字和名称所构成的。汇编语言提供整数的数字，而一个整数可以二进制、八进制、十进制或十六进制来表示 (配合字尾的基底)，如果未选基底，则组译器会使用内定值 (十进制)，下表为可用的基底。

基底	型态	数字
B	二进制	01
O	八进位	01234567
D	十进制	0123456789
H	十六进制	0123456789ABCDEF

语法

```
[name] DB value1 [,value2 [,...]]
[name] DBIT
[name] DB repeated-count DUP(?)
```

说明

上述的假指令会引导组译器在数据存储器 (data memory) 内保留空间给变量 `name` (如果有指定 `name`)。内存保留的空间大小则由其后的个数及数据型态, 或由重复次数及数据型态来决定。由于微控制器的数据存储器无法事先记录数据内容, 组译器不会对数据存储器做初始值的设定, 因此 `value1` 和 `value2` 必须为 `?` 表示只是保留内存空间给程序执行时使用, 并没有设定其初始值。DBIT 只保存一个位, 组译器会将每 8 个 DBIT 整合在一起并且保留一个字节给这 8 个 DBIT 变量。

范例

```
DATA .SECTION 'DATA'
tbuf DB ?
flag1 DBIT
sbuf DB ?
cflag DBIT
```

在这个范例中, 组译器保留地址 0 给变量 `tbuf`、地址 1 的位 0 给变量 `flag1`、地址 2 给变量 `sbuf` 和地址 1 的位 1 给变量 `cflag`。

语法

```
name LABEL {BIT|BYTE|WORD}
```

说明

此假指令会将 `name` 的地址设定为其后的变量的内存地址。

范例

```
lab1 LABEL WORD
d1 DB ?
d2 DB ?
```

在这个范例中, `d1` 是 `lab1` 的低字节, 而 `d2` 则是 `lab1` 的高字节。

语法

```
name EQU expression
```

说明

藉者将 `expression` 指定给 `name`, 假指令 EQU 会制造一个新的数值符号、别名或文字符号 (`name`) 来代表 `expression`。数值符号是一个代表 16 位值的名称、别名则是另一个符号的名称、而文字符号则是代表一串字符组合的名称。`name` 必须是唯一的, 就是之前未被定义过。`expression` 可以是一个整数、字符串常数、指令助忆符号、数学表达式或地址表达式。

范例

```
accreg EQU 5
bmove EQU mov
```

在这个范例中, 变量 `accreg` 等于 5, 而 `bmove` 相当于指令 `mov`。

宏假指令

宏假指令是定义一个名称来代表一段原始陈述式(指令及假指令),而在原始程序档案中可以重复使用此名称以取代这段陈述式,也就是程序中所有用到此段陈述式的地方皆可用此名称代替之。在编译时,组译器会自动将每一个宏假指令的名称用宏假指令所定义的陈述式来取代。

在源文件的任何地方皆可定义宏假指令,只要呼叫此宏假指令的地方是在宏假指令定义之后即可。宏假指令的定义中,可以呼叫先前已经被定义的其它宏假指令,如此将形成一种巢状的架构,组译器可提供的最大巢状为7层。

语法

```
name MACRO [dummy-parameter [,...]]
statements
ENDM
```

在宏假指令中,可以使用假指令 LOCAL 来定义只能在宏假指令本体内使用的变量。

语法

```
name LOCAL dummy-name [,...]
```

说明

宏假指令 LOCAL 用来定义只能在宏假指令本体内使用的符号,使用时必须定义在 MACRO 假指令之后的第一行。dummy-name 是一个暂时使用的名称,当宏假指令被呼叫展开时,它将被一个唯一的名称所取代,组译器针对 dummy-name 产生对应的实际名称,这个实际名称的格式为 ??digit, 其中 digit 数字为十六进制且范围由 0000 至 FFFF。当 MACRO/ENDM 的定义区段中使用到标记(label)时,最好将此标记加入 LOCAL 假指令之中,否则当 MACRO 被源文件呼叫多次时,相同的标记名称会重复出现在程序中,组译器将会发布程序错误的信息。

下面的范例中, tmp1 和 tmp2 都是虚假参数,当呼叫此宏假指令时,都会被实际参数所取代。label1 和 label2 都被宣告为 LOCAL, 如果没有其它的 MACRO 被呼叫,在第一次呼叫时将分别被 ??0000 和 ??0001 所取代,如果没有宣告 LOCAL, label1 和 label2 则会类似于原始程序中的标记宣告,而在第二次呼叫此宏假指令时,就会出现重复定义的错误讯息。

```
Delay MACRO tmp1, tmp2
LOCAL label1, label2
mov a, 70h
mov tmp1, a
label1:
mov tmp2, a
label2:
clr wdt1
clr wdt2
sdz tmp2
jmp label2
sdz tmp1
jmp label1
ENDM
```


下面的原始程序将会呼叫名为 Delay 的宏指令

```

; T.ASM
; Sample program for MACRO.
.ListMacro
Delay MACRO tmp1, tmp2
    LOCAL label1, label2
    mov a, 70h
    mov tmp1, a
label1:
    mov tmp2, a
label2:
    clr wdt1
    clr wdt2
    sdz tmp2
    jmp label2
    sdz tmp1
    jmp label1
ENDM

data .section 'data'
BCnt db ?
SCnt db ?

code .section at 0 'code'
Delay BCnt, SCnt
end

```

组译器会将宏假指令 Delay 展开如下列的程序，请注意在宏假指令本体内的第 4 行到第 17 行，它们的位置差距 (offset) 皆是 0000，也就是宏假指令在定义时，本体内的指令并不占用内存空间。在程序第 24 行呼叫 Delay 宏假指令时，它就被展开成 11 行并且排放在宏假指令之后，这 11 行的指令则沿用相同的行数 (line number) 第 24 行。同时，虚假参数 tmp1 和 tmp2 分别被实际参数 BCnt 和 SCnt 所取代。

```

File: T.asm           Holtek Cross-Assembler Version 2.80           Page 1

1 0000                ; T.ASM
2 0000                ; Sample program for MACRO.
3 0000                .ListMacro
4 0000                Delay MACRO tmp1, tmp2
5 0000                  LOCAL label1, label2
6 0000                  mov a, 70h
7 0000                  mov tmp1, a
8 0000                label1:
9 0000                  mov tmp2, a
10 0000               label2:
11 0000                  clr wdt1
12 0000                  clr wdt2
13 0000                  sdz tmp2
14 0000                  jmp label2
15 0000                  sdz tmp1
16 0000                  jmp label1
17 0000                  ENDM
18 0000
19 0000                data .section 'data'
20 0000 00            BCnt db ?
21 0001 00            SCnt db ?
22 0002
23 0000                code .section at 0 'code'
24 0000                Delay BCnt, SCnt
24 0000 0F70          1      mov a, 70h
24 0001 0080          R1     mov BCnt, a
24 0002              1      ??0000:
24 0002 0080          R1     mov SCnt, a
24 0003              1      ??0001:
24 0003 0001          1      clr wdt1
24 0004 0005          1      clr wdt2
24 0005 1780          R1     sdz SCnt
24 0006 2803          1      jmp ??0001
24 0007 1780          R1     sdz BCnt
24 0008 2802          1      jmp ??0000
25 0009                end

0 Errors

```

汇编语言指令

指令的语法如下：

```
[name:] mnemonic [operand1 [,operand2] ] [; comment]
```

其中

name:	→	符号名称
mnemonic	→	指令名称 (关键词)
operand1	→	缓存器 内存地址
operand2	→	缓存器 内存地址 立即值

名称

名称是由字母、数字或特殊字符所组成的，可以当标记 (label) 使用。当标记使用时，必须在名称后面紧接一个冒号 (colon)。

助忆符号

助忆符号是原始程序中使用的指令名称。

操作数、运算符和表达式

操作数 (来源或目的) 定义被指令使用的数值，它们可以是常数、变量、缓存器、表达式或关键词。当使用指令时，必须谨慎选择正确的来源操作数及目的操作数。钱字符号 \$ 是一个特殊的操作数，它代表目前的地址。

表达式是由操作数所组成，在程序编译时用来计算出数值或内存地址。表达式是常数、符号以及任何被算术运算符分隔之常数和符号的组合。

运算符定义表达式中各操作数之间的运算动作，组译器提供了许多运算符去处理操作数，有些运算符只处理常数，有些则处理内存数值，也有两者兼具的。如果运算符只是处理常数，则在程序编译时就会直接计算出数值。以下是组译器所提供的运算符。

● 算术运算符

+ - * / %(MOD)

● SHL 和 SHR 运算符

语法

```
expression SHR count
expression SHL count
```

这些位平移运算符的值全都为常数，expression 依照 count 所指定的位数目向右移 (SHR) 或向左移 (SHL)，如果被平移的位超过有效位数时，则对应的位会以 0 填满，例如：

```
mov A, 01110111b SHR 3 ; result ACC=00001110b
mov A, 01110111b SHL 4 ; result ACC=01110000b
```

• 位格式运算符 NOT、AND、OR、XOR

语法

```

NOT expression
expression1 AND expression2
expression1 OR expression2
expression1 XOR expression2

```

NOT 各位的 1 阶补码
 AND 各位 AND 运算
 OR 各位 OR 运算
 XOR 各位 XOR 运算

• 偏移运算符

语法

```
OFFSET expression
```

OFFSET 运算符会产出 `expression` 的偏移地址，`expression` 可以是标记、变量或其它内存地址的操作数，被 OFFSET 运算符所传回的数值必须是立即值。

• LOW、MID 和 HIGH 运算符

语法

```

LOW expression
MID expression
HIGH expression

```

如果 `expression` 的结果是一个立即值的话，则 LOW/MID/HIGH 运算符会产出 `expression` 的值，而且是分别取此数值的低 / 中 / 高字节。但是如果 `expression` 的结果是标记，则 LOW/MID/HIGH 运算符将取得此标记所在之内存地址的低 / 中 / 高字节的数值。

比如：

```

cs .section at 456H 'code'
lab1:
...

```

则：

```

mov a,low 9840H → mov a,40H
mov a,mid 9840H → mov a,84H
mov a,high 9840H → mov a,98H
mov a,low lab1 → mov a,56H
mov a,mid lab1 → mov a,45H
mov a,high lab1 → mov a,4H

```

• LOW_NIBBLE、HIGH_NIBBLE 运算符

语法

```

LOW_NIBBLE expression
HIGH_NIBBLE expression

```

LOW_NIBBLE 会算出 `expression` (立即值或标记) 的低字节的低四位，HIGH_NIBBLE 会算出 `expression` (立即值或标记) 的低字节的高四位。

比如：

```

cs .section at 123h 'code'
lab2:

```

则：

```
MOV A,LOW_NIBBLE 23h → MOV A,3H
MOV A,LOW_NIBBLE lab2 → MOV A,3H
MOV A,HIGH_NIBBLE 23h → MOV A,2H
MOV A,HIGH_NIBBLE lab2 → MOV A,2H
```

●BANK 内存区块运算符

语法

```
BANK name
```

BANK 运算符会产出程序段所在的内存区块的编号，此程序段是由 **name** 所宣告的。如果 **name** 是标记型态，产出的则是 ROM 程序内存区块，如果 **name** 为数据变量则产出 RAM 数据存储区区块。内存区块的数值格式与缓存器 BP 的格式相同，请参考各微控制器的数据册，不同的微控制器可能有不同的 BP 格式。

范例 1：

```
mov A, BANK start
mov BP, A
jmp start
```

范例 2：

```
mov A, BANK var
mov BP, A
mov A, OFFSET var
mov MP1, A
mov A, R1
```

运算符之优先权

优先权	运算符
1 (Highest)	(), []
2	+, - (unary), LOW, MID, HIGH, OFFSET, BANK
3	*, /, %, SHL, SHR
4	+, - (binary)
5	>(greater than), >=(greater than or equal to), <(less than), <= (less than or equal to)
6	== (equal to), !=(not equal to)
7	! (bitwise NOT)
8	& (bitwise AND)
9 (Lowest)	(bitwise OR), ^ (bitwise XOR)

其它

前置引用

当标记、变量名称和其它符号在原始码中被宣告之前，组译器允许它们被使用（前置命名引用），但是在假指令 EQU 右边的符号是不允许前置引用的。

局部标记

局部标记有固定形式的，如 \$number。其中 **number** 可以为 0 至 127，局部标记除了是可以重复使用外，其它功用与一般标记是相同的。局部标记必须使用在任意两个连续的标记之间而且同样的局部标记名称也可以用在其它两个连续标记之间。在编译原始程序文件之前，组译器会将每一个局部标记转换成唯一的标记。任何两个连续标记之间，最多可以定义 128 个局部标记。

范例

```

Label1:                ; label
    $1:                ;; local label
        mov a, 1
        jmp $3
    $2:                ;; local label
        mov a, 2
        jmp $1
    $3:                ;; local label
        jmp $2
Label2:                ; label
        jmp $1
    $0:                ;; local label
        jmp Label1
    $1:                jmp $0
Label3:

```

汇编语言保留字

下表是组合语上使用的保留字。

●保留字 (指令、运算符)

\$	DUP	INCLUDE	NOT
*		LABEL	OFFSET
+	ELSE	.LIST	OR
-	END	.LISTINCLUDE	ORG
.	ENDIF	.LISTMACRO	PAGE
/	ENDM	LOCAL	PARA
=	ENDP	LOW	PROC
?	EQU	MACRO	PUBLIC
[]	ERRMESSAGE	MESSAGE	RAMBANK
AND	EXTERN	MID	ROMBANK
BANK	HIGH	MOD	.SECTION
BYTE	IF	NEAR	SHL
DB	IFDEF	.NOLIST	SHR
DBIT	IFE	.NOLISTINCLUDE	WORD
DC	IFNDEF	.NOLISTMACRO	XOR
ELSEIF	DEFINED	HIGH_NIBBLE	LOW_NIBBLE

●保留字 (指令助忆符号)

ADC	HALT	RLCA	SUB
ADCM	INC	RR	SUBM
ADD	INCA	RRA	SWAP
ADDM	JMP	RRC	SWAPA
AND	MOV	RRCA	SZ
ANDM	NOP	SBC	SZA
CALL	OR	SBCM	TABRDC
CLR	ORM	SDZ	TABRDL
CPL	RET	SDZA	XOR
CPLA	RETI	SET	XORM
DAA	RL	SIZ	
DEC	RLA	SIZA	
DECA	RLC	SNZ	

●保留字 (寄存器名称)

A	WDT	WDT1	WDT2
---	-----	------	------

组译器选项

组译器选项可以经由 HT-IDE3000 中的 Options 选单的 Project 命令来设定，组译器的选项位于 Project Option 对话框的中心部分，可在符号定义 (Define Symbol) 编辑框中定义符号。

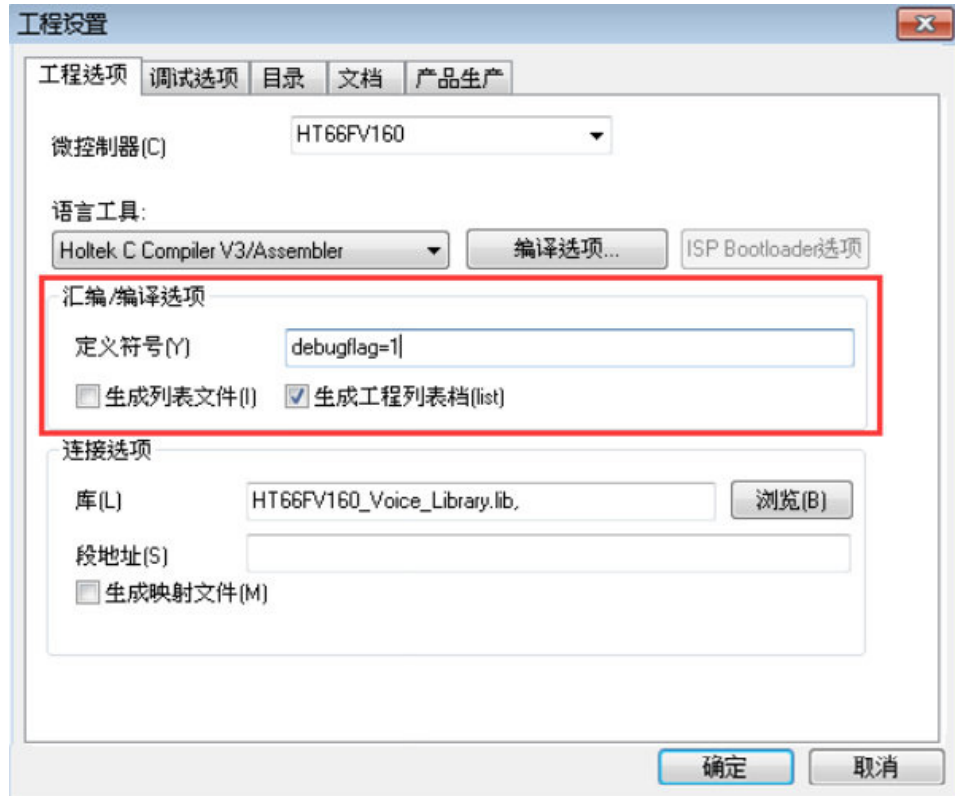


图 7-1

语法

```
symbol1 [=value1] [,symbol2 [=value2 ] [,...]]
```

范例

```
debugflag=1, newver=3
```

产生列表档案的检验框可用来指定是否要产出列表档案 (Listing file)，如果检验框被设定，则要产出列表档案，否则将不会产生列表档案。

专案编译选项里面有一个选项 `case sensitive for assembly`，若勾选此选项，则表示区分大小写，一般在写混合语言时勾选，不勾选则默认不区分大小写：

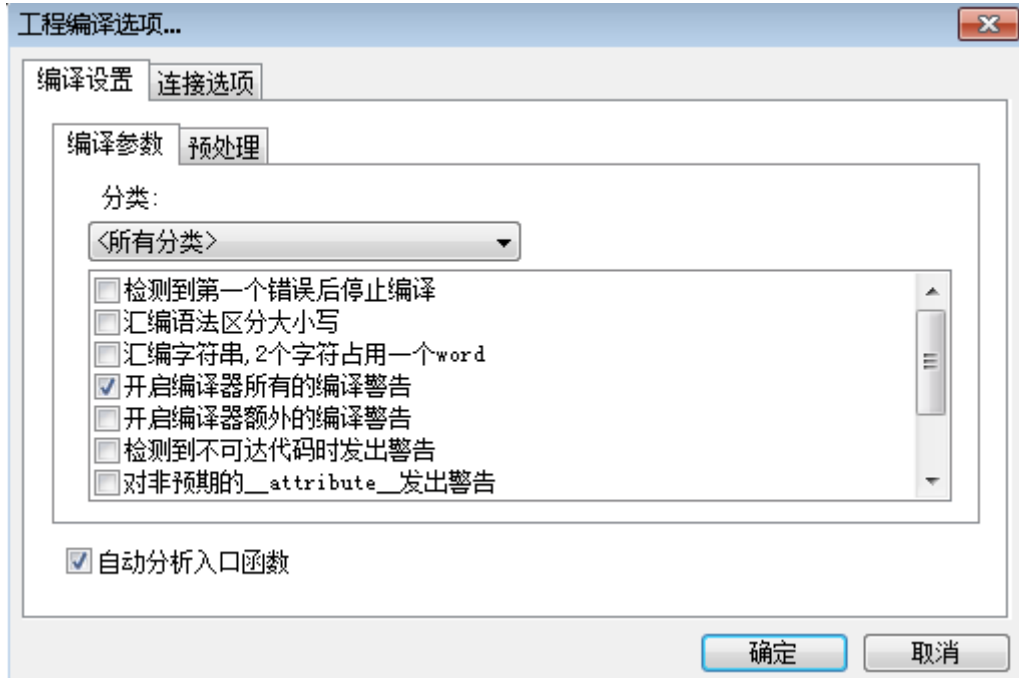


图 7-2

编译列表档案格式

编译列表档案包含原始程序的列表和概要信息，每页的第一行是标题，内容则包括公司名称、组译器版本、源文件名称、编译时的日期，时间以及页码。

原始程序行表

在原始程序中的每行陈述式都以下列的格式输出到编译列表档：

[line-number] offset [code] statement

- Line-number 是指陈述式在原始程序文件的第几行，从第一个陈述式开始计算起 (4 个十进制数)。
- offset 是从陈述式所在的程序段开头到这个陈述式的内存地址差距 (4 个十六进制数字)。
- code 只有会产生机器码 (machine code) 或数据的陈述式才会出现此项 (两个十六进制、每个占用 4 位数字的数据)。

如果数值在编译时已确定的话，会用十六进制数字表示 code 的数值，否则的话，将使用适当的旗标表明应该使用何种方式去计算此数值。下列两个旗标可能会出现于 code 项目之后。

- ◆ R→ 需要重新安置地址 (连结器解决此状况)
- ◆ E→ 需要参考外部符号 (连结器解决此状况)

下列旗标可能会出现于 code 项目之前。

- ◆ = → EQU 或等号

code 项目中可能出现下列的符号或数字。

- ◆ ---- → 代表程序段的开始地址 (连结器会解决此符号)
- ◆ nn[xx] → DUP 符号: nn DUP(?) 重复次数

- statement 与原始程序档案中相同的陈述式或是宏假指令所展开的陈述式。在陈述式之前可能会出现下列的符号。
 - ◆ n → 宏假指令展开时的巢状层次
 - ◆ C → 此陈述式是从含入档 (INCLUDE 档案) 引进的
- 总结

```

0           1           2           3           4           5           6
123456789012345678901234567890123456789012345678901234567890...
IIII  oooo hhhh hhhh EC source-program-statement
                        Rn

```

IIII → 行数字 (4 位数, 向右靠齐)
 oooo → 机器码的地址差距 (4 位数)
 hhhh → 两个 4- 位数的机器码
 E → 参考外部数据
 C → 从含入档加入的陈述式
 R → 需要重新安置地址
 n → 宏假指令展开后的巢状层次

编译总结

在编译列表档案的结尾处会统计此次编译所发生之警告及错误的总数。

其它

在编译期时如果发生错误, 则错误讯息和编号会直接出现在发生错误的陈述式下方。

● 编译列表档案的范例

File: SAMPLE.ASM Holtek Cross-Assembler Version 2.86 Page 1

```

1 0000          page 60
2 0000          message      'Sample Program 1'
3 0000
4 0000          .listinclude
5 0000          .listmacro
6 0000
7 0000          #include "sample.inc"

1 0000          C pa      equ   [12h]
2 0000          C pac     equ   [13h]
3 0000          C pb      equ   [14h]
4 0000          C pbc     equ   [15h]
5 0000          C pc      equ   [16h]
6 0000          C pcc     equ   [17h]
7 0000          C

8 0000
9 0000          extern extlab : near
10 0000         extern extb1 : byte
11 0000
12 0000         clrpb macro
13 0000         clr pb
14 0000         endm
15 0000
16 0000         clrpa macro
17 0000         mov a, 00h
18 0000         mov pa, a
19 0000         clrpb
20 0000         endm
21 0000
22 0000         data .section 'data'
23 0000 00      b1      db ?
24 0001 00      b2      db ?
25 0002 00      bit1    dbit
26 0003
27 0000         code .section 'code'
28 0000 0F55     mov a, 055h
29 0001 0080     R mov b1, a
30 0002 0080     E mov extb1, a
31 0003 0FAA     mov a, 0aah
32 0004 0093     mov pac, a
33 0005         clrpa
33 0005 0F00     1 mov a, 00h
33 0006 0092     1 mov [12h], a
33 0007         1 clrpb
33 0007 1F14     2 clr [14h]
34 0008 0700     R mov a, b1
35 0009 0F00     E mov a, bank extlab
36 000A 0F00     E mov a, offset extb1
37 000B 2800     E jmp  extlab
38 000C
39 000C 1234 5678 dw 1234h, 5678h, 0abcdh, 0ef12h
   ABCD EF12
40 0010         end

```

0 Errors

第八章 连结器

连结器之功用

连结器会将组译器或 C 编译器所产出的目的档案结合起来并产生工作档案。它会将目的档案中的程序代码和数据结合为一个整体，并且从函式库档案中搜寻公用函式或变量以解决原始程序中所使用的外部函式或外部变量。如果没有指定明显的地址，连结器也会将程序段及数据段安排在 ROM/RAM 内存的指定或预定的地址。最后，连结器会将程序代码和其它数据输出到工作文件 (task file)，此工作档会由 IDE (整合发展环境) 加载 ICE (硬件仿真器) 中执行除错。另外，连结器所引用的函式档是由函式库总管产生的。

连结器的选项

选项将会控制连结器执行所指定的工作。第三章的 Options 选单中，Project 命令内提供一个连结器选项的对话框，可以设定这些选项。包括：

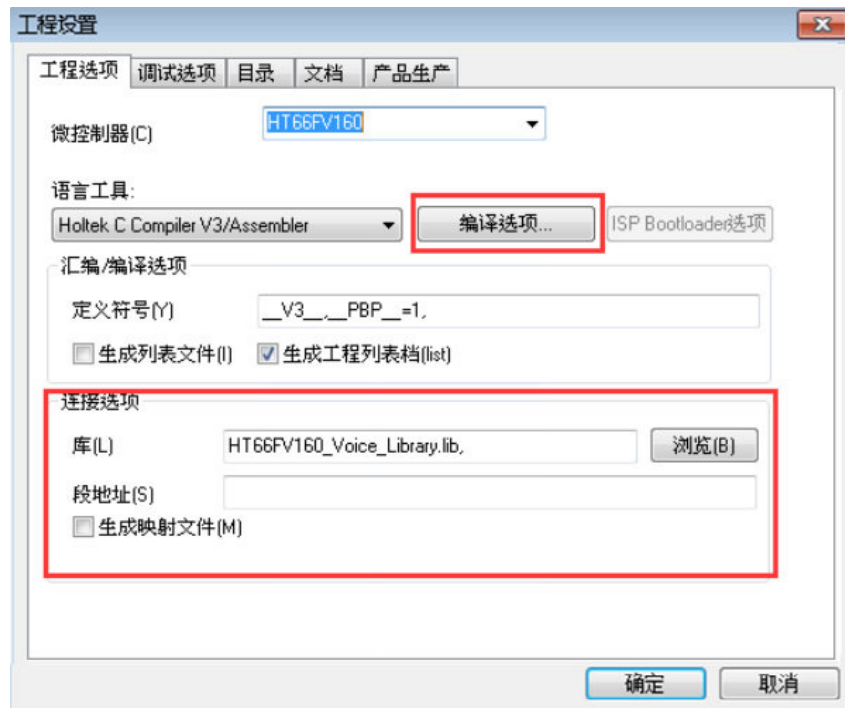


图 8-1

函式库档案

- 语法

```
libfile1[,libfile2...]
```

如果输入的目的地文件中用到其它目的档皆没有定义的程序或变量时，这个选项则是告知连结器从函式库档案中找寻。如果函式库的某个程序模块包含被呼叫的程序或变量，则只有此程序模块会被引进到输出的工作档案，而不是将整个函式库档案引进 (参考第九章函式库总管)。

程序段地址

语法

`section_name=address[,section_name=address]...`

此选项是指定程序段的内存地址；`section_name` 是被寻址的程序段名称，必须被定义在至少一个输入的档案中，否则会出现警告讯息。`address` 是被指定的程序段地址，格式为 `xxxx` 的十六进制。

产生地址对映图档案

此选项的检查盒用来指定是否要产出地址对映图档案。

连接选项

连接选项仅在编译器为 `Compiler V3.10/Assembler` 时有效。

优化 RAM 空间 (不允许巢状中断): 优化 RAM 空间，不允许在一个中断里面进入另一个中断。

删除没有被调用的函数 (针对 C): 若一个函数不被调用，将不为其分配空间，这个选项默认开启。

将未初始化的全局变量，全局 / 局部静态变量默认设为 0: 若 `global` 变数没有初始值，则默认其初始值为 0。在程序开始运行的时候，将其 `clear`。

绝对地址变量地址重迭时发出警告: 检查绝对地址变量的地址是否重迭，并在重迭时发出 `warning`。

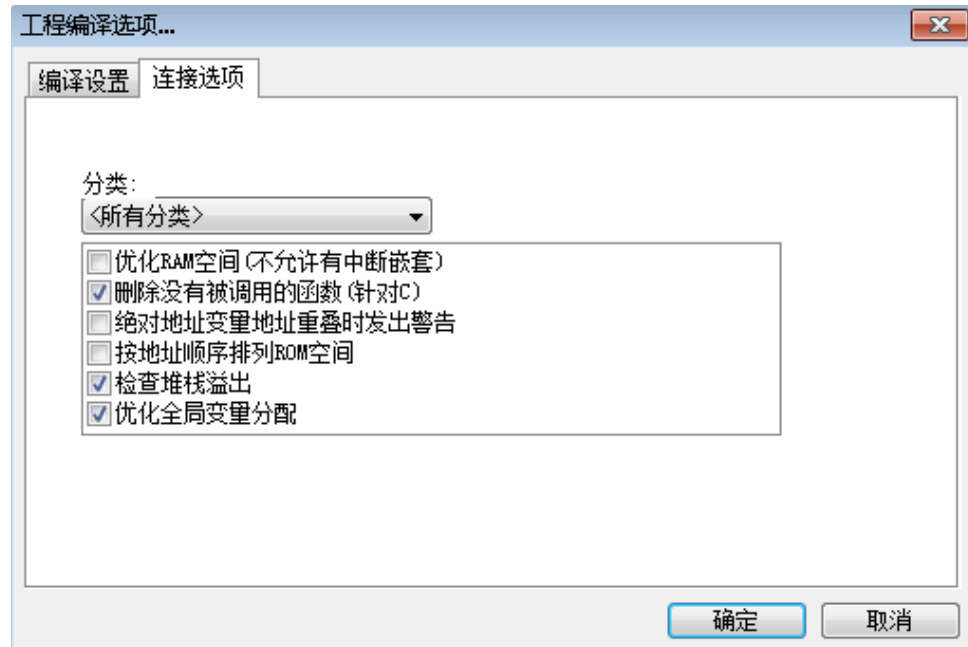


图 8-2

地址对映图档案

地址对映图档案会列出程序中所有程序段及数据段的名称、内存地址与长度，也会列出连结档案时发生状态的信息。连结器会在此档案的结尾列出程序进入点的地址。地址对映图档案还会列出所有公用变量的名称与地址。如果连结器找不到外部变量或程序所要对应的公用变量或公用程序时，它会将外部变量或程序的名称及所在的文件名称记录在地址对映图档案中。以下为档案的内容：

```
Input Object File: C:\t\T.OBJ
Input Object File: C:\t\STARTUP1_L.OBJ

Input Library File: C:\Program Files\Holtek MCU Development Tools\HT-
IDE3000V7.81\LIB\LIBHOLTEKGCC.LIB
Input Library File: C:\Program Files\Holtek MCU Development Tools\HT-
IDE3000V7.81\LIB\v3_mrsc.lib
Input Library File: C:\Program Files\Holtek MCU Development Tools\HT-
IDE3000V7.81\LIB\v3_tabrd_mrsc.lib
```

Bank	Start	End	Length	Class	Name
00h	0000h	0000h	0001h	CODE	@code (C:\t\T.OBJ)
00h	0001h	0001h	0001h	CODE	_main (C:\t\T.OBJ)
00h	0002h	0002h	0001h	CODE	@ROMDATA_BASE (C:\t\ STARTUP1_L.OBJ)
00h	0004h	0007h	0004h	CODE	@isr04_code (C:\t\T.OBJ)
00h	0008h	0028h	0021h	CODE	@start (C:\t\T.OBJ)
00h	0029h	0033h	000bh	CODE	_isr04 (C:\t\T.OBJ)
00h	0034h	0038h	0005h	CODE	_fun (C:\t\T.OBJ)
00h	0080h	0081h	0002h	DATA	@HCCINIT0 (C:\t\T.OBJ)

Local Sections

Bank	Start	End	Length	Class	Name
00h	0085h	0085h	0000h	LOCAL	@dummy4 (C:\t\T.OBJ)
00h	0082h	0084h	0003h	LOCAL	_isr04 (C:\t\T.OBJ)
00h	0085h	0085h	0000h	LOCAL	@dummy (C:\t\T.OBJ)
00h	0085h	0085h	0000h	LOCAL	_main (C:\t\T.OBJ)
00h	0085h	0085h	0000h	LOCAL	_fun (C:\t\T.OBJ)

Public Symbols Information

Address	Public by Name
0034h	_fun
0080h	_g_a
0029h	_isr04
0082h	_isr04fs
0001h	_main
0001h	startup_value_1

Address	Public by Value
0001h	_main
0001h	startup_value_1
0029h	_isr04
0034h	_fun
0080h	_g_a
0082h	_isr04fs

```
Error(L2001) : Unresolved external symbol '_g_b' in file 'C:\t\t\T.OBJ'
```

ROM Usage Statistics

Size	Used	Percentage
2000h	0038h	0%

RAM Usage Statistics

Bank	Size	Used	Percentage
00h	0080h	0005h	3%
01h	0080h	0000h	0%
02h	0080h	0000h	0%
Total	0180h	0005h	1%

Use Stack Count

```
_isr04: 0
_main: 1
```

Call Tree

```
@dummy4
  _isr04
    @dummy
      _main
        _fun
```

Total 1 error(s), Total 0 Warning(s)

连结器的工作档案与除错档案

连结器输出的档案之一是工作档，它由档案表头和二进制数据等两个部分所组成。档案表头的部分包含有连结器的版本、微控制器名称型号和程序内存的大小，二进制的部分则是程序代码。连结器输出的另一个档案是除错文件，它记录所有 IDE 除错程序所需要的数据，包括源文件的文件名、符号的名称和源文件中程序行的行数。IDE 系统会参考这些符号除错的数据。除非已完成除错程序，否则不要将此档案删除，避免 IDE 系统无法提供符号除错的功能。

第九章 函式库总管

除了先前讨论的一般用途 8 位微控制器发展工具外，对于特殊用途的语音和 LCD 微控制器，经由所提供的模拟工具及对声音合成和语调产生器执行模拟的逐步导引，本公司提供一些额外的公用程序。这个部份包含能够快速及有效的开发程序和除错所需要的信息。

函式库总管的功能

函式库总管提供处理函式库档案的功能。连结器 (Linker) 在产生输出档的过程中会使用到函式库档案。它是由一个或多个被编译过的目标模块所组成的，而且是在连结档案时使用的。此档案存有其它程序可能需要的执行模块。

可使用函式库总管去建立函式库档案，并将包含有公用程序的目的文件加到函式库档案中。在产出这些目的档之前，必须利用汇编语言的假指令 PUBLIC 将所有可共享的程序设定为公用程序 (可参考汇编语言和组译器等章节)。之后使用组译器将原始程序文件编译成目的文件 (.OBJ)，再经函式库总管将目的档加到指定的函式库档案中。在连结的过程中，如果连结器发现在程序中有未解决的名称时 (也就是程序中使用到外部变量或函式，但是在所有输入的目的档之中找不到同名的公用变量或函式)，它会从函式库档案中找寻这些未解决的名称，并且将包含这些名称的程序模块取出及复制。假如在函式库模块中找到未解决的名称，连结器会将此模块连结到输出的工作档案内。

设定函式库档案

函式库总管提供以下的功能：

- 产生新的函式库档案
- 将程序模块加到函式库档案或从函式库档案删除程序模块
- 从函式库档案撷取一个程序模块并且产出包含此模块的目的文件

如图 9-1 选用 Tools 选单的 Library Manager 命令。图 9-2 显示函式库总管的功能对话盒。

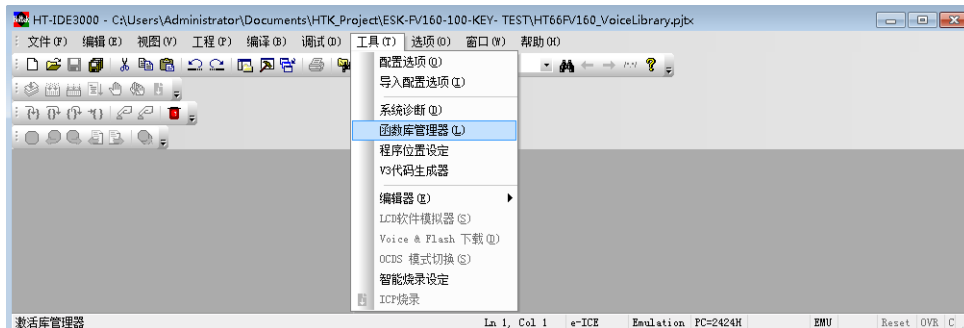


图 9-1

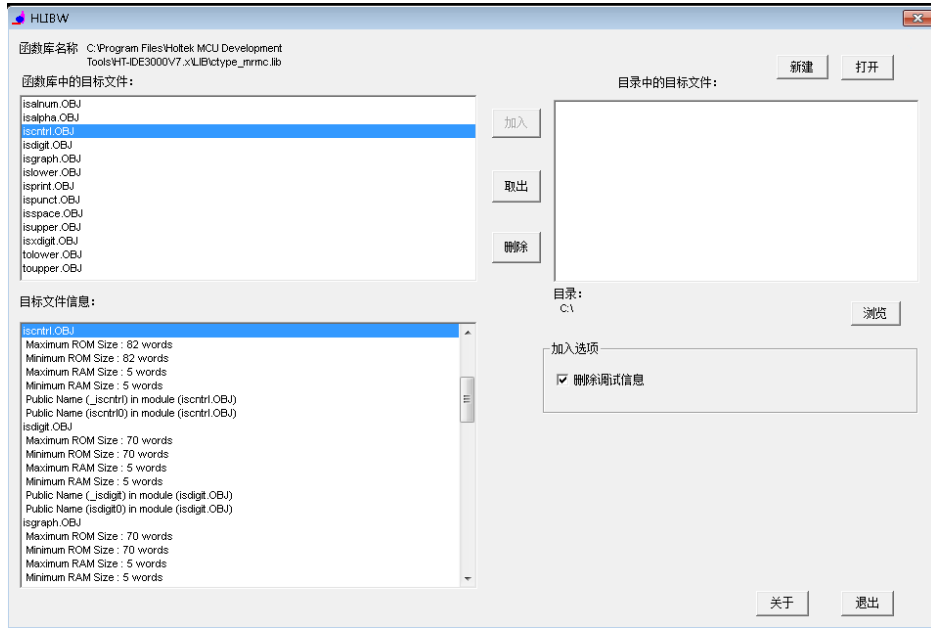


图 9-2

产生新的函式库档案

按下图 9-2 右上角的 Open 按钮，会出现图 9-3。

在 File Name[N] 处键入一个新的函式库档案的档名，并按下 OK 按钮，会出现图 9-4 的对话框以便确认。假如选择 YES 按钮，将会产生一个新的函式库档案，但是不包含任何程序模块。

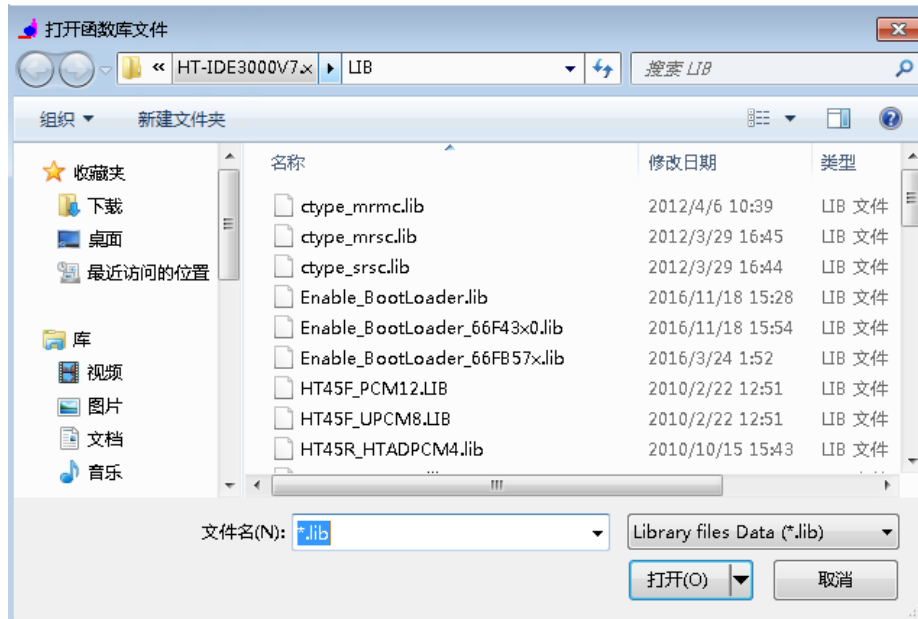


图 9-3

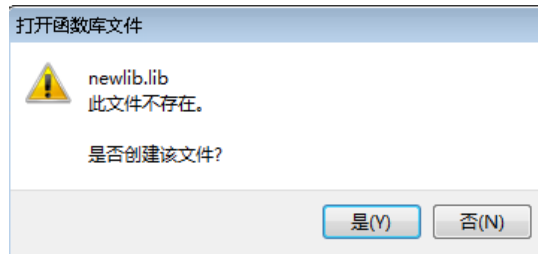


图 9-4

将程序模块加到函式库档案

从“Object in Directory”对话框中选择一个目标模块，并按下 [ADD] 钮将这个目标模块加到函式库档案中。

从函式库档案删除程序模块

从“Object in Library”盒中选择一个目标模块，并按下 [Delete] 钮从函式库档案中将目标模块删除。

从函式库档案中撷取程序模块并产出目的档案

从“Object in Library”盒中选择一个目标模块，并按下 [ExTract] 钮。之后会产生一个与目标模块同名而且有相同内容的档案，这个档案的文件名会显示在“Object in Directory”盒中。

目标模块的信息

按下 Open 钮，将出现图 10-3 所示的画面。从 File Name 对话框上方的列表盒中选择一个函式库档案，按下 OK 钮。在图 9-2 中，所选择的函式库档案中所有的目标模块会被列在“Object in Library”盒中。与每个目标模块有关的信息也会列在“Objects Information”盒中，详述如下：

- 最大的程序内存容量 (ROM)
目标模块程序代码使用的最大内存容量，会与程序段 (code section) 的位置型态有关。
- 最小的程序内存容量 (ROM)
目标模块程序代码实际使用的最小容量。
- 最大的数据存储器容量 (RAM)
目标模块程序数据所使用的最大容量，与数据段 (data section) 的位置型态有关。
- 最小的数据存储器容量 (RAM)
目标模块程序数据实际使用的最小容量。
- 公用名称
目标模块中所有公用符号的名称。

第十章 组译器使用的保留字

汇编语言保留字

下表所列的是使用在组合语上的保留字。

●保留字 (假指令、运算符)

\$	DUP	INCLUDE	NOT
*		LABEL	OFFSET
+	ELSE	.LIST	OR
-	END	.LISTINCLUDE	ORG
.	ENDIF	.LISTMACRO	PAGE
/	ENDM	LOCAL	PARA
=	ENDP	LOW	PROC
?	EQU	MACRO	PUBLIC
[]	ERRMESSAGE	MESSAGE	RAMBANK
AND	EXTERN	MID	ROMBANK
BANK	HIGH	MOD	.SECTION
BYTE	IF	NEAR	SHL
DB	IFDEF	.NOLIST	SHR
DBIT	IFE	.NOLISTINCLUDE	WORD
DC	IFNDEF	.NOLISTMACRO	XOR
ELSEIF	DEFINED	HIGH_NIBBLE	LOW_NIBBLE

●保留字 (指令助忆符号)

ADC	HALT	RLCA	SUB
ADCM	INC	RR	SUBM
ADD	INCA	RRA	SWAP
ADDM	JMP	RRC	SWAPA
AND	MOV	RRCA	SZ
ANDM	NOP	SBC	SZA
CALL	OR	SBCM	TABRDC
CLR	ORM	SDZ	TABRDL
CPL	RET	SDZA	XOR
CPLA	RETI	SET	XORM
DAA	RL	SIZ	
DEC	RLA	SIZA	
DECA	RLC	SNZ	

●保留字 (缓存器名称)

A	WDT	WDT1	WDT2
---	-----	------	------

指令集

● 算术指令

ADD A,[m]	将数据存储器内容相加到累加器中
ADDM A,[m]	将累加器内容相加到数据存储器中
ADD A,x	将数值相加到累加器中
ADC A,[m]	将数据存储器内容和进位旗标的内容相加到累加器中
ADCM A,[m]	将累加器和进位旗标的内容相加到数据存储器中
SUB A,x	从累加器中减去指定的数值
SUB A,[m]	从累加器中减去数据存储器内容
SUBM A,[m]	从累加器中减去数据存储器内容, 将结果存入数据存储器中
SBC A,[m]	从累加器中减去数据存储器内容和进位旗标
SBCM A,[m]	从累加器中减去数据存储器内容和进位旗标, 将结果存入数据存储器中
DAA [m]	根据前面两个 BCD 值相加之结果, 将累加器的内容转换成 BCD 值, 再存入数据存储器中

● 逻辑运算指令

AND A,[m]	累加器和数据存储器中的内容作 AND 运算, 将结果存到累加器中
OR A,[m]	累加器和数据存储器中的内容作 OR 运算, 将结果存到累加器中
XOR A,[m]	累加器和数据存储器中的内容作 XOR 运算, 将结果存到累加器中
ANDM A,[m]	数据存储器内容和累加器中的内容作 AND 运算, 将结果存到数据存储器中
ORM A,[m]	数据存储器内容和累加器中的内容作 OR 运算, 将结果存到数据存储器中
XORM A,[m]	数据存储器内容和累加器中的内容作 XOR 运算, 将结果存到数据存储器中
AND A,x	将累加器的内容和数值作 AND 运算, 将结果存到累加器中
OR A,x	将累加器的内容和数值作 OR 运算, 将结果存到累加器中
XOR A,x	将累加器的内容和数值作 XOR 运算, 将结果存到累加器中
CPL [m]	将数据存储器中的每一位取 1 的补码
CPLA [m]	将数据存储器中的每一位取 1 的补码并将结果存到累加器中

● 递增和递减指令

INCA [m]	将数据存储器的内容加 1 并将结果存到累加器中
INC [m]	将数据存储器的内容加 1
DECA [m]	将数据存储器的内容减 1 并将结果存到累加器中
DEC [m]	将数据存储器的内容减 1

● 旋转指令

RRA [m]	将数据存储器的内容向右移 1 个位并将结果存到累加器中
RR [m]	将数据存储器的内容向右移 1 个位
RRCA [m]	将数据存储器的内容和进位旗号向右移 1 个位并将结果存到累加器中
RRC [m]	将数据存储器的内容和进位旗号向右移 1 个位
RLA [m]	将数据存储器的内容向左移 1 个位并将结果存到累加器中
RL [m]	将数据存储器的内容向左移 1 个位
RLCA [m]	将数据存储器的内容和进位旗号向左移 1 个位并将结果存到累加器中
RLC [m]	将数据存储器的内容和进位旗号向左移 1 个位

● 数据位移指令

MOV A,[m]	将数据存储器的内容复制到累加器中
MOV [m],A	将累加器的内容复制到数据存储器中
MOV A,x	将数值载入至累加器中

● 位运算指令

CLR [m].i	将数据存储器中的位清除
SET [m].i	设定数据存储器中的位

● 分支指令

JMP addr	无条件地跳跃
SZ [m]	如果数据存储器的内容为 0 则跳过下一个指令
SZA [m]	将数据存储器的内容复制到累加器，如果值为 0 则跳过下一个指令
SZ [m].i	如果数据存储器的 i 位为 0 则跳过下一个指令
SNZ [m].i	如果数据存储器的 i 位不为 0 则跳过下一个指令
SIZ [m]	将数据存储器的内容加 1，如果结果为 0 则跳过下一个指令
SDZ [m]	将数据存储器的内容减 1，如果结果为 0 则跳过下一个指令
SIZA [m]	将数据存储器的内容加 1，将结果存到累加器中，如果结果为 0 则跳过下一个指令
SDZA [m]	将数据存储器的内容减 1，将结果存到累加器中，如果结果为 0 则跳过下一个指令
CALL addr	呼叫特定地址的子程序
RET	从子程序回到主程序
RET A,x	从子程序回到主程序并将数值存入累加器
RETI	从中断程序回到主程序

● 查表指令

TABRDC [m]	读取 ROM 码 (本页) 到数据存储器 and TBLH 中
TABRDL [m]	读取 ROM 码 (最后一页) 到数据存储器 and TBLH 中

● 其它指令

NOP	无动作
CLR [m]	清除数据存储器
SET [m]	设定数据存储器
CLR WDT	清除看门狗定时器
CLR WDT1	清除看门狗定时器
CLR WDT2	清除看门狗定时器
SWAP [m]	将数据存储器的低 4 位与高 4 位互相交换
SWAPA [m]	将数据存储器的低 4 位与高 4 位互相交换并将结果存到累加器
HALT	进入省电模式

第十一章 LCD 仿真器

简介

盛群 LCD 仿真器, HT-LCDS, 提供一个机制让使用者仿真 LCD 驱动器的输出。根据使用者设计的图型和控制程序, HT-LCDS 在屏幕上实时显示这些图型, 在尚未取得真实的 LCD 硬件面板之前, LCD 仿真器将使发展的过程更为便利。但是如果项目所用的微控制器没有支持 LCD 功能, 这些命令是无效的。

只有 HT-ICE 和 e-Link 支持该功能。

LCD 面板配置档案

在开始 LCD 的模拟之前, 必须先建立一个 LCD 面板配置档案。HT-LCDS 将从 LCD 面板配置文件取得 LCD 的设定数据并根据仿真时取得的数据将图案显示在屏幕上, 如果这个档案不存在, HT-LCDS 将无法模拟 LCD 的动作。对于具有处理 LCD 功能的微控制器, 必须要建立 LCD 面板配置文件以便执行 LCD 的模拟。工具选单 (Tools menu) 中的 LCD 仿真器命令可用来建立面板配置文件及执行 LCD 的模拟 (图 11-1)。LCD 面板配置文件包含面板的配置数据及图案信息。

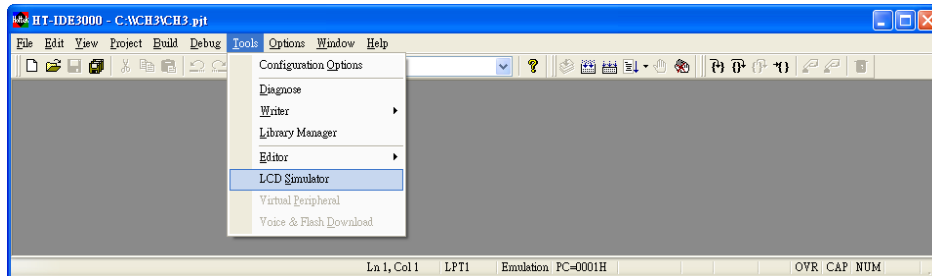


图 11-1

面板档档名和项目的关系

原则上, 除了扩展名为 .lcd 外, 面板配置文件的主文件名和现行项目 (current project) 的名称是相同的。HT-LCDS 会认为这个档案是对应于现行项目的面板配置文件。使用 HT-LCDS 的 File 选单中的 New 命令或工具列的 New 按钮可以建立新的面板配置文件。如果想要将面板配置文件改为与现行项目不同的档名时, 可以使用 File 选单的 Save 命令或工具列中的 Save 按钮。

当 HT-LCDS 开始模拟时, 它要参考现行的面板配置文件以获取仿真所需要的信息, 此时可由选择 HT-LCDS 的 File 选单的 New 或 Open 命令将面板配置文件启动为现行的档案。LCD 面板配置文件的文件名可能和现行项目的名称相同, 也可以选用不同的名称。

选择 HT-LCDS

从 Tools 选单中选用 LCD simulator 命令时, 如果现行项目对应的面板配置文件已存在, 如图 11-2 的窗口会出现。在此窗口下方的表格中, 某些指定的 COM/SEG 位置中会显示所对应之图案位图文件 (bitmap) 的档名, 同时这些图案会显示在窗口上方的面板显示框。如果在项目的目录中并未存有相关的面板配置文件, 则 LCD 仿真器在启动时并不会显示面板显示框和 COM/SEG 表格。图 11-3 是 HT-LCDS 的工具列信息。

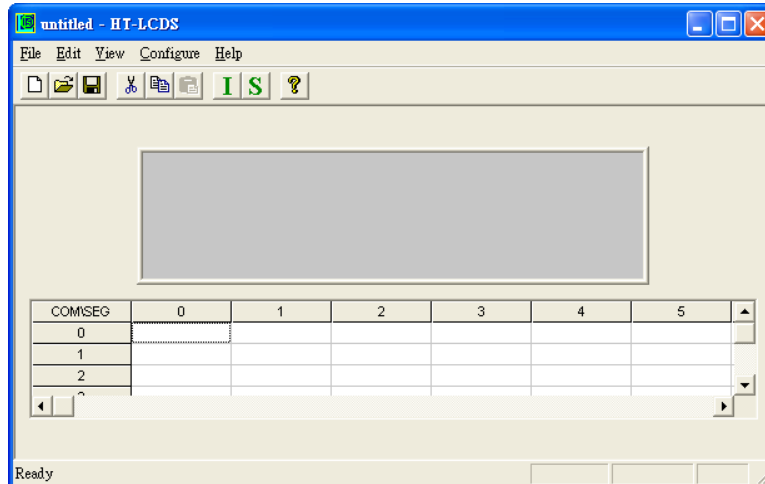


图 11-2

下图为 HT-LCDS 功能列讯息

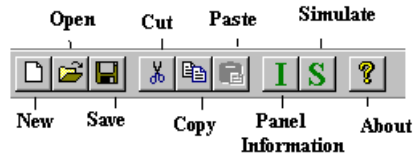


图 12-3

- New: 建立一个新的面板配置文件。
- Open: 开启一个已存在的面板配置文件。
- Save: 储存面板配置文件。
- Cut: 删除一个图案。
- Copy: 复制一个图案到剪贴板
- Paste: 加入已复制的图案到控制板上。
- I: 面板信息对话框。
- S: 进入 LCD 仿真模式。

LCD 面板图案文件

LCD 面板中的图案是一种位图档案 (.bmp)，用来表示面板上实际的图案和它们在面板上的位置，任何位图编辑器都可以用来建立此种档案。HT-LCDS 中 Edit 选单的 Panel Editor 命令也可以用来设定 LCD 面板图案的信息。位图文件可有可无，当 LCD 面板是点矩阵形式时，是不需要图案文件的。

建立 LCD 面板配置档案

底下两个步骤是用来建立一个面板配置档案。

- 设定面板的配置，包括 LCD 驱动器的 segment 和 common 的个数、面板的宽度和高度，也可以设定面板配置档案的目录以及点矩阵形式的面板。
- 选择图案和它们的位置，这将建立图案与 COM/SEG 位置间的关系。

建立面板的配置架构

使用 HT-LCDS 中 File 选单的新命令建立面板的配置信息。面板配置对话框会被显示 (图 11-4)。设定正确的 LCD 规格数据, 如 COM/SEG 的个数、宽度、高度和图案文件所在的目录, 之后按下 [OK] 钮。完成面板配置的设定后, 系统会回到图 12-2 以便选取图案。

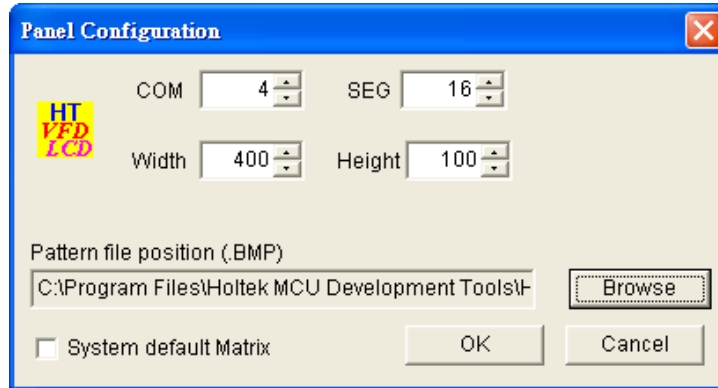


图 11-4

面板的配置包括:

- COM 和 SEG。设定 LCD 驱动器的 COMMON 个数和 SEGMENT 个数。当图 11-4 出现时, 会将微控制器中 LCD 驱动器的内定个数显示出。必须确认这些个数和微控制器中 LCD 驱动器的实际个数是一致的。
- 宽度和高度。这是面板显示在屏幕的尺寸。可以改变此项以调整面板画面的大小。
- 面板配置档案的路径。使用浏览按钮选择面板配置文件所存放的目录或设定在项目的目录。
- 点矩阵形式。仿真点矩阵型式的 LCD 面板。图 11-5 显示点矩阵屏幕。

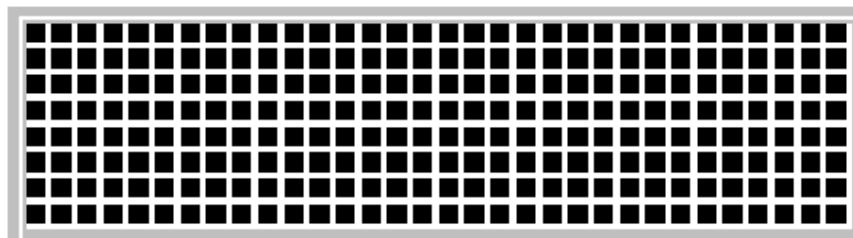


图 11-5

注意: 不要设定与实际 LCD 驱动器不相同的 COM 和 SEG 个数, 否则会发生不可预测的结果。

选择图案并设定位置

底下的步骤说明如何选择图案和位置。

- 使用 HT-LCDS 中 File 选单的新命令建立一个新的面板配置文件, 完成面板配置的设定后, 图 11-2 会显示。接着需要从 Pattern Information 对话框 (图 11-6) 中选择图案以及设定 COM/SEG 位置。加入新的图案的这一节会有详细的描述。

- 使用 HT-LCDS 中 File 选单的 Open 命令开启一个存在的面板配置档案。所有的图案皆会显示在图 11-2 上方的面板显示框内，而图案文件的文件名会出现在图 11-2 中 COM/SEG 的表格中。使用者可以加入，删除或改变图案信息、包括图案的文件名和位置。
- 使用 HT-LCDS 中 Edit 选单的 Panel Editor 命令去开启一个面板图形文件，如果这个面板图形档案已经被设定，便不需要选择图案，只需设定图案的位置。使用 Panel Editor 定义图案的这一节中对此部分有详细描述。

加入新的图案

- 移动光标到如图 11-2 中 COM/SEG 位置的格子中并点选鼠标 2 次，将会出现如图 11-6 所示的 Pattern Information 对话框。Pattern List 列表盒会将此项目目录中所有的图案文件 (.bmp) 排列显示。Size 方格中是所选之图案的位大小，Com 和 Seg 方格中是这个图案被设定的 COM/SEG 位置，这三个方格中的数据是无法修改的。
- 从 Pattern List 列表盒中选择一个图案文件 (一个位图文件) 或使用 Browse 钮从其它目录中选取一个图案文件。HT-LCDS 使用双色的位图文件当作图案的影像来源。Preview 窗口会将所选取的图案放大显示。
- 设定图案在面板显示框内的 X/Y 坐标位置。
- 按下 [OK] 钮且回到图 11-2，然后点选 File 选单的 Save 命令或点选工具列上的 Save 钮，此时已完成面板档案的建立或修改。

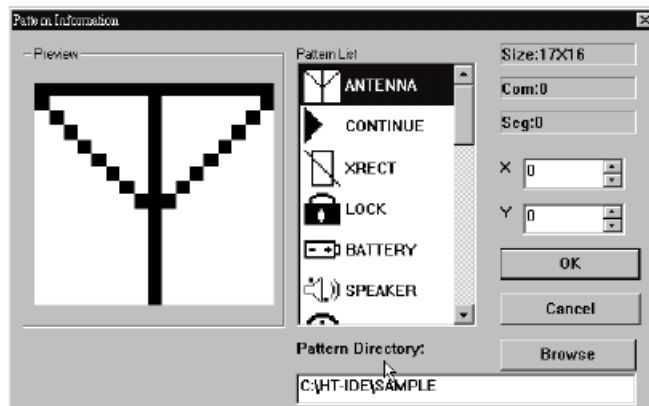


图 11-6

删除图案

- 如图 11-2 所示，选取欲删除图案的 COM/SEG 并按下 [Delete] 钮或点选工具列上的 Cut 钮。

改变图案

- 首先删除要改变的图案，然后加入一个新的图案以更换之。
- 另外一种方式，在图 11-2 中，选定图案的 COM/SEG 位置并双击鼠标，图 11-6 Pattern Information 对话框会出现，从这个对话框中的 Pattern List 列表盒内选一个图案并且按下 [OK] 钮。

改变图案位置

- 如图 11-2 所示，可使用点选并拖拉 (Select-drag-drop) 的方法直接将图案搬到面板显示框的适当位置。
- 另外一种方法，在图 11-2 中，选定图案的 COM/SEG 位置并双击鼠标，图 11-6 Pattern Information 对话框会出现，在这个对话框中的 X 及 Y 方格内设定图案的新位置并且按下 [OK] 钮。

当以上的操作皆已完成且系统回到如图 11-2 的窗口时，点选 HT-LCDS 中 File 选单的 Save 命令或点选工具列上的 Save 钮，则完成面板档案的建立或修改。

如何加入使用者定义的矩阵

当 COM/SEG 个数与面板的 ROW/COL 个数不一致时，HT-LCDS 提供一个映成策略 (File 选单的 Import user matrix 命令) 以帮助定义一个新的矩阵，例如：

假设一个使用 2 COMs 和 6 SEGs 的 LCD 面板并且具备 3 ROWs × 4 COLs 的矩阵，规划成下列的对应表：

COM0-SEG0	COM0-SEG1	COM0-SEG2	COM0-SEG3
COM1-SEG0	COM1-SEG1	COM1-SEG2	COM1-SEG3
COM0-SEG4	COM0-SEG5	COM1-SEG4	COM1-SEG5

上面的矩阵可用底下的定义档定义之：

```

;MATRIX.DEF
;Comment line
ROW=3
COLUMN=4
;mapping syntax:ROW,COL=>COM,SEG
0,0 => 0,0 ; Map Row0 col0 to COM0 SEG0
0,1 => 0,1 ; Map Row0 col1 to COM0 SEG1
0,2 => 0,2 ; Map Row0 col2 to COM0 SEG2
0,3 => 0,3 ; Map Row0 col3 to COM0 SEG3
1,0 => 1,0 ; Map Row1 col0 to COM1 SEG0
1,1 => 1,1 ; Map Row1 col1 to COM1 SEG1
1,2 => 1,2 ; Map Row1 col2 to COM1 SEG2
1,3 => 1,3 ; Map Row1 col3 to COM1 SEG3
2,0 => 0,4 ; Map Row2 col0 to COM0 SEG4
2,1 => 0,5 ; Map Row2 col1 to COM0 SEG5
2,2 => 1,4 ; Map Row2 col2 to COM1 SEG4
2,3 => 1,5 ; Map Row2 col3 to COM1 SEG5
    
```

使用 Panel Editor 定义图案

HT-LCDS 提供一个完整的面板编辑接口去定义 LCD 面板的图案，如果已经存在整个面板的图形文件，就可以不用在面板上去设定所有图案的文件名，只需要去设定这些图案在面板上的位置。

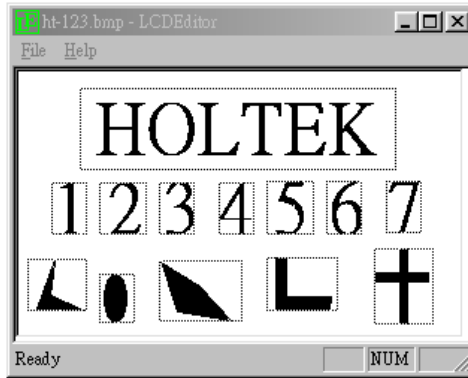


图 11-7

底下的步骤说明如何设定图案在 LCD 面板上的位置

- 在完成面板配置的设定之后，使用 Edit 选单的 Panel Editor 命令进入 Panel Editor 窗口。
- 在 Panel Editor 窗口中使用 File 选单的 Open 命令开启面板图形档案 (.bmp)。

注意：Panel Editor 只能支援双色的 .bmp 档案 (黑白)

- 面板图形档案中所有的图案会显示在如图 12-7 的 Panel Editor 窗口中。
- 双击鼠标或使用按下拖放的方法，在 COM/SEG 字段中设定显示的图案文件名。在 Save Pattern 对话框出现后就可以输入图案的数据。
- 针对面板上的每个图案重复执行上述的步骤。
- 在完成所有图案的数据设定后，回到 Panel Editor 窗口。再使用 File 选单的 Save 命令实际将这些设定储存。
- 离开 Panel Editor 窗口回到 HT-LCDS 窗口，此时面板上会显示新的设定。

使用批次文件将图案加入面板

经由 Edit 选单的 Add Item Batch 命令，HT-LCDS 提供可从批处理文件中将图案加入面板中。批次档是以扩展名为 .BTH 的文字文件，在批处理文件中每个图案项目需要定义图案文件的文件名和图案在面板中的位置。当使用 Menu 选单的 Add Item Batch 命令选取一个批处理文件后，HT-LCDS 会将此批次文件中描述的所有图案加入到面板中的指定位置。底下是一个 .BTH 档案的例子。

```
;this is a comment line.
;item syntax: BMPfile.bmp, COM, SEG, X, Y
CRYSTAL.BMP,      0, 2, 120, 30
FION.BMP,         2, 3, 200, 50
CLIN.BMP,         3, 2, 130, 90
STEVE.BMP,        4, 4, 20, 40
```

选择 LCD 面板的颜色

HT-LCDS 提供如图 11-8 所示的调色对话框，可使用 HT-LCDS 中 Configure 选单的 Set Panel Color 命令来选择面板的颜色。

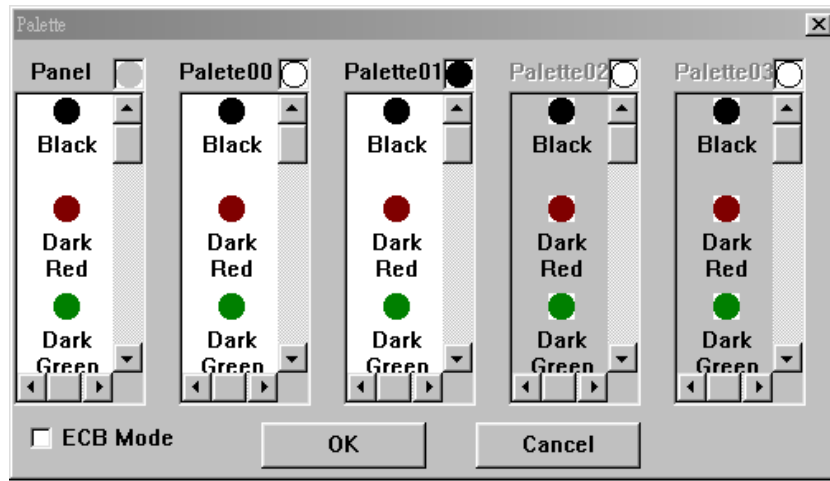


图 11-8

注意：ECB 模式只针对 HTG21×0 的彩色 LCD

为 VFD 面板设定颜色样式

HT-LCDS 提供一套接口如下图 14-9 来设定盛群 MCU(如 HT49CVX 系列)的每个颜色样式。

至选单上选 Configure 并执行 Set VFD pattern Color 来完成这个设定。

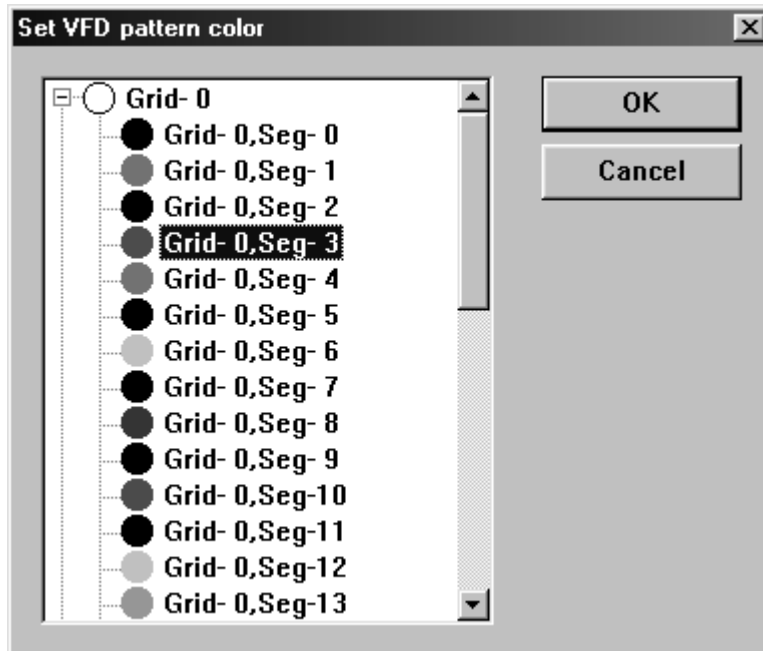


图 11-9

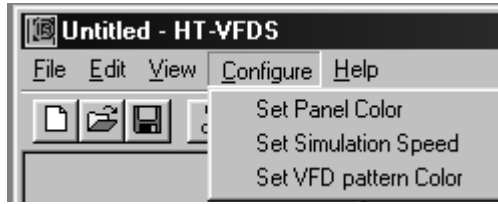


图 11-10

LCD 模拟

在开始 LCD 模拟之前，确定 HT-LCDS 参照到正确的面板配置档案。然后如图 11-1 和 11-2 所示的，使用 Tools 选单的 LCD Simulator 命令即可进入 HT-LCDS 环境。

- 点选工具列的 S 钮，HT-LCDS 会参阅对应的面板配置文件开始进行 LCD 的模拟。
- 如果开启一个并非现行项目对应的面板配置文件，并点选工具列的 S 钮，则 HT-LCDS 会参阅这个开启的面板配置文件并进行 LCD 的模拟。

当 HT-LCDS 开始仿真时将会显示如图 11-9 的窗口，此时最新的 LCD 图案会出现在面板显示框之中。

停止模拟

用鼠标双击 LCD 仿真窗口的名称栏将使 HT-LCDS 回到编辑模式。

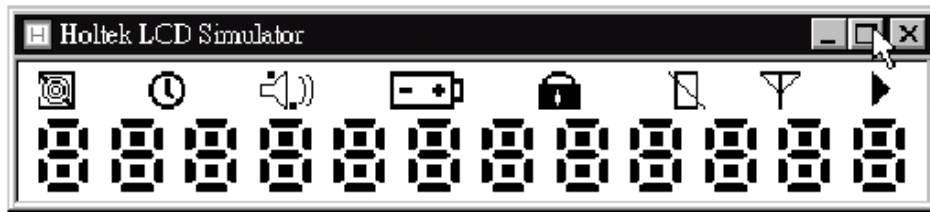


图 11-9

Copyright® 2022 by HOLTEK SEMICONDUCTOR INC.

使用指南中所出现的信息在出版当时已尽量做到合理注意，但合泰不保证信息准确无误，文中提到的应用目的仅仅是用来做为参考，合泰不保证这些说明将是适当的，也不推荐将合泰的产品使用在会由于故障或其它原因可能会对人身造成危害的地方。合泰特此声明，不授权将产品使用于救生、维生从机或系统中做为关键从机。合泰对于客户或第三方因说明书所载信息错误或遗漏、使用产品或说明书而遭受的一切损失，一概不负任何责任。合泰拥有不事先通知而修改使用指南中所记载的产品或规格的权利，如欲取得最新的信息，请与我们联系。