

Keil C51 使用详解

V1.0



©电子设计世界！版权所有，欢迎转贴，请勿修改并注明出处。保留一切权利。

第一章 Keil C51 开发系统基本知识... 6

第一节 系统概述...	6
第二节 Keil C51 单片机软件开发系统的整体结构...	6
第三节 Keil C51 工具包的安装...	7
1. C51 for Dos	7
2. C51 for Windows 的安装及注意事项: ...	7
第四节 Keil C51 工具包各部分功能及使用简介...	7
1. C51 与 A51.	7
2. L51 和 BL51.	8
3. DScope51, Tscope51 及 Monitor51.	8
4. Ishell 及 uVision.	9
第二章 Keil C51 软件使用详解...	10
第一节 Keil C51 编译器的控制指令...	10
1. 源文件控制类...	10
2. 目标文件(Object)控制类: ...	10
3. 列表文件(listing)控制类: ...	10
第二节 dScope51 的使用...	11
1. dScope51 for Dos	11
2. dScope for Windows	12
第三节 Monitor51 及其使用...	13
1. Monitor51 对硬件的要求...	13
2. Mon51 的使用...	13
3. MON51 的配置...	13
4. 串口连接图: ...	13
5. MON51 命令及使用...	14
第四节 集成开发环境(IDE)的使用...	14
1. Ishell for Dos 的使用...	14

2. uVision for windows 的使用... 15

第三章 Keil C51 vs 标准 C... 15

第一节 Keil C51 扩展关键字... 15

第二节 内存区域(Memory Areas): ... 16

1. Program Area: ... 16
2. Internal Data Memory: 16
3. External Data Memory. 16
4. Special Function Register Memory. 16

第三节 存储模式... 16

1. Small 模式... 16
2. Compact 模式... 17
3. large 模式... 17

第四节 存储类型声明... 17

第五节 变量或数据类型... 17

第六节 位变量与声明... 17

1. bit 型变量... 17
2. 可位寻址区说明 20H—2FH.. 18

第七节 Keil C51 指针... 18

1. 一般指针... 18
2. 存储器指针... 18
3. 指针转换... 18

第八节 Keil C51 函数... 19

1. 中断函数声明: ... 19
2. 通用存储工作区... 19
3. 选通用存储工作区由 using x 声明, 见上例。... 19
4. 指定存储模式... 19
5. #pragma disable. 19

6. 递归或可重入函数指定... 19

7. 指定 PL/M—51 函数... 20

第四章 Keil C51 高级编程... 20

第一节 绝对地址访问... 20

1. 绝对宏：... 20

2. `_at_`关键字... 21

3. 连接定位控制... 21

第二节 Keil C51 与汇编的接口... 21

1. 模块内接口... 21

2. 模块间接口... 21

第三节 Keil C51 软件包中的通用文件... 22

1. 动态内存分配... 22

2. C51 启动文件 STARTUP.A51. 22

3. 标准输入输出文件... 25

4. 其它文件... 25

第四节 段名协定与程序优化... 25

1. 段名协定(Segment Naming Conventions) 25

2. 程序优化... 25

第五章 Keil C51 库函数参考... 26

第一节 本征库函数(intrinsic routines)和非本征证库函数... 26

第二节 几类重要库函数... 26

1. 专用寄存器 include 文件... 26

2. 绝对地址 include 文件 `absacc.h`. 26

3. 动态内存分配函数，位于 `stdlib.h` 中... 27

4. 缓冲区处理函数位于 “`string.h`”中... 27

5. 输入输出流函数，位于 “`stdio.h`”中... 27

第三节 Keil C51 库函数原型列表... 27

1. CTYPE.H.. 27
2. INTRINS.H.. 27
3. STDIO.H.. 28
4. STDLIB.H.. 28
5. STRING.H.. 28

第六章 Keil C51 例子: Hello. c. . 29

第一节 uVision for Windows 的使用步骤... 29

第二节 Ishell for Dos 使用步骤... 30

第七章 Keil C51 的代码效率... 30

第一节 存储模式的影响... 30

第二节 程序结构的影响... 31

第八章 dScope for Windows 使用详解... 32

第一节 概述... 32

1. 主窗口 (Mainframe Window) ... 32
2. 调试窗口 (DEBUG Window) ... 32
3. 命令窗口 (Command Window) ... 32
4. 观察窗口 (Watch Window) ... 32
5. 寄存器窗口 (Registe Window) ... 32
6. 串口窗口 (Serical Windows) ... 32
7. 性能分析窗口... 32
8. 内存窗口 (Memory Window) ... 32
9. 符号浏览窗口 (Symbol Browser Window) ... 33
10. 调用线窗口 (Call—Stack Window) ... 33
11. 代码覆盖窗口... 33
12. 外围设备窗口(peripherals) 33

第二节 dScope for Windows 基本操作... 33

1. 指定初始化文件... 33
2. 观察变量... 33
3. 显示 RAM 的值... 34
4. 观察堆栈... 34
5. 中断处理程序调试... 34
6. 性能分析（Performance Analyzer: PA） ... 34

第三节 dScope for Windows 命令文件的编制... 34

1. 地址空间及地址空间类型... 34
2. 常量... 35
3. 变量... 36
4. 运算符... 38
5. 表达式... 38
6. 数组... 38
7. 结构和联合... 38
8. 指针： ... 38
9. dScope 命令语句... 38
10. 函数... 43

第一章 Keil C51 开发系统基本知识

第一节 系统概述

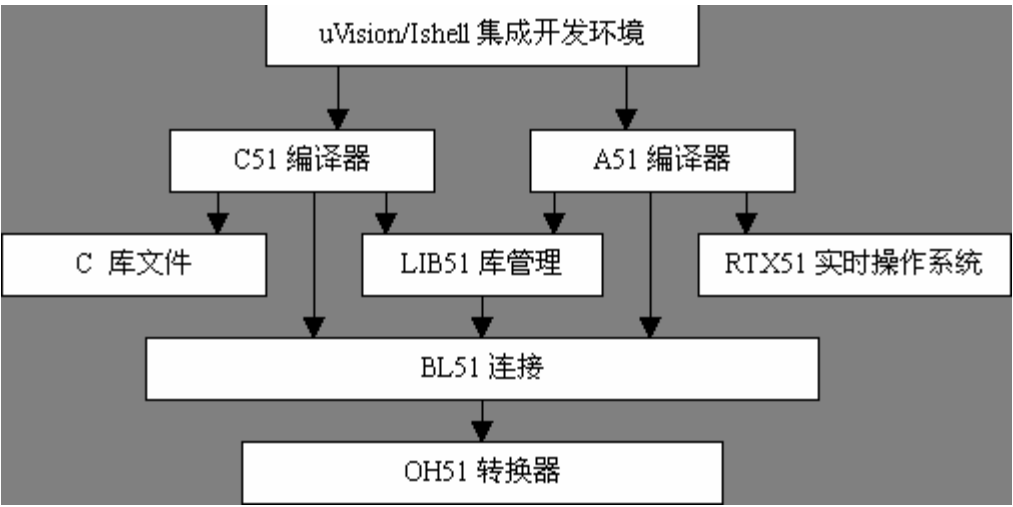
Keil C51 是美国 Keil Software 公司出品的 51 系列兼容单片机 C 语言软件开发系统，与汇编相比，C 语言在功能上、结构性、可读性、可维护性上有明显的优势，因而易学易用。用过汇编语言后再使用 C 来开发，体会更加深刻。

Keil C51 软件提供丰富的库函数和功能强大的集成开发调试工具，全 Windows 界面。另外重要的一点，只要看一下编译后生成的汇编代码，就能体会到 Keil C51 生成的目标代码效率非常之高，多数语句生成的汇编代码很紧凑，容易理解。在开发大型软件时更能体现高级语言的优势。

下面详细介绍 Keil C51 开发系统各部分功能和使用。

第二节 Keil C51 单片机软件开发系统的整体结构

C51 工具包的整体结构，如图(1)所示，其中 uVision 与 Ishell 分别是 C51 for Windows 和 for Dos 的集成开发环境(IDE)，可以完成编辑、编译、连接、调试、仿真等整个开发流程。开发人员可用 IDE 本身或其它编辑器编辑 C 或汇编源文件。然后分别由 C51 及 A51 编译器编译生成目标文件(.OBJ)。目标文件可由 LIB51 创建生成库文件，也可以与库文件一起经 L51 连接定位生成绝对目标文件(.ABS)。ABS 文件由 OH51 转换成标准的 Hex 文件，以供调试器 dScope51 或 tScope51 使用进行源代码级调试，也可由仿真器使用直接对目标板进行调试，也可以直接写入程序存贮器如 EPROM 中。



图(1) C51 工具包整体结构图

第三节 Keil C51 工具包的安装

1. C51 for Dos

在 Windows 下直接运行软件包中 DOS\C51DOS.exe 然后选择安装目录即可。完毕后欲使系统正常工作须进行以下操作(设 C:\C51 为安装目录):

修改 Autoexec.bat, 加入

```
path=C:\C51\Bin
```

```
Set C51LIB=C:\C51\LIB
```

```
Set C51INC=C:\C51\INC
```

然后运行 Autoexec.bat

2. C51 for Windows 的安装及注意事项:

在 Windows 下运行软件包中 WIN\Setup.exe, 最好选择安装目录与 C51 for Dos 相同, 这样设置最简单(设安装于 C:\C51 目录下)。然后将软件包中 crack 目录中的文件拷入 C:\C51\Bin 目录下。

第四节 Keil C51 工具包各部分功能及使用简介

1. C51 与 A51

(1) C51

C51 是 C 语言编译器, 其使用方法为:

```
C51 sourcefile[编译控制指令]
```

或者 C51 @commandfile

其中 sourcefile 为 C 源文件(.C)。大量的编译控制指令完成 C51 编译器的全部功能。包控 C51 输出文件 C.LST, .OBJ, .I 和.SRC 文件的控制。源文件(.C)的控制等, 详见第五部分的具体介绍。

而 Commandfile 为一个连接控制文件其内容包括: .C 源文件及各编译控制指令, 它没有固定的名字, 开发人员可根据自己的习惯指定, 它适于用控制指令较多的场合。

(2) A51

A51 是汇编语言编译器, 使用方法为:

A51 sourcefile[编译控制指令]

或 A51 @ commandfile

其中 sourcefile 为汇编源文件(.asm 或.a51),而编译控制指令的使用与其它汇编如 ASM 语言类似,可参考其他汇编语言材料。

Commandfile 同 C51 中的 Commandfile 类似,它使 A51 使用和修改方便。

2. L51 和 BL51

(1) L51

L51 是 Keil C51 软件包提供的连接/定位器,其功能是将编译生成的 OBJ 文件与库文件连接定位生成绝对目标文件(.ABS),其使用方法为:

L51 目标文件列表[库文件列表] [to outputfile] [连接控制指令]

或 L51 @Commandfile

源程序的多个模块分别经 C51 与 A51 编译后生成多个 OBJ 文件,连接时,这些文件全列于目标文件列表中,作为输入文件,如果还需与库文件(.Lib)相连接,则库文件也必须列在其后。outputfile 为输文件名,缺少时为第一模块名,后缀为.ABS。连接控制指令提供了连接定位时的所有控制功能。Commandfile 为连接控制文件,其具体内容是包括了目标文件列表,库文件列表及输出文件、连接控制命令,以取代第一种繁琐的格式,由于目标模块库文件大多不止 1 个,因而第 2 种方法较多见,这个文件名字也可由使用者随意指定。

(2) BL51

BL51 也是 C51 软件包的连接/定位器,其具有 L51 的所有功能,此外它还具有以下 3 点特别之处:

- a. 可以连接定位大于 64kBytes 的程序。
- b. 具有代码域及域切换功能(CodeBanking & Bank Switching)
- c. 可用于 RTX51 操作系统

RTX51 是一个实时多任务操作系统,它改变了传统的编程模式,甚至不必用 main()函数,单片机系统软件向 RTOS 发展是一种趋势,这种趋势对于 186 和 386

及 68K 系列 CPU 更为明显和必须，对 8051 因 CPU 较为简单，程序结构等都不太复杂，RTX51 作用显得不太突出，其专业版软件 PK51 软件包甚至不包括 RTX51Full，而只有一个 RTX51TINY 版本的 RTOS。RTX51 TINY 适用于无外部 RAM 的单片机系统，因而可用面很窄，在本文中不作介绍。Bank switching 技术因使用很少也不作介绍。

3. DScope51, Tscope51 及 Monitor51

(1) dScope51

dScope51 是一个源级调试器和模拟器，它可以调试由 C51 编译器、A51 汇编器、PL/M-51 编译器及 ASM-51 汇编器产生的程序。它不需目标板（for windows 也可通过 mon51 接目标板），只能进行软件模拟，但其功能强大，可模拟 CPU 及其外围器件，如内部串口，外部 I/O 及定时器等，能对嵌入式软件功能进行有效测试。

其使用方法为：

DS51[debugfile][INIT(initfile)]

其中 debugfile 是一个 Hex 格式的 8051 文件，即待调试的文件其为可选的，可在进入 dScope51 后用 load 命令装入。

Initfile 为一个初使化文件，它在启动 dScope51 后，在 debugfile 装入前装入，装有一些 dScope 的初使化参数及常用调试函数等。下面是一个 dScope.ini 文件(for dos)的内容：

Load ..\..\ds51\8051.iof

Map 0,0xffff

dScope51 for Windows 则直接用鼠标进入，然后用 load 装入待调文件。

(2) tScope51

与 dScope51 不同的是 Scope51 必须带目标板，目前它可以通过两种方式访问目标板。(1) 通过 EMul51 在线仿真器，tScope51 为该仿真器准备了一个动态连接文件 EMUL51.IOT，但该方法必须配合该仿真器。(2) 通过 Monitov51 监控程序，这种方法是可行的，tScope51 为访问 Monitor51 专门带有 MON51.IOT 连接程序，使用时可

通过串口及监控程序来调试目标板。

其使用方法为：

```
TS51[INIT(file_name.ini)]
```

其中 file_name.ini 为一个初使化文件。

进入 TS51 后，必须装入 IOT 文件，可用的有 MON51.IOT 及 EMUL51.IOT 两种，如装入 MON51.IOT：

```
Load.C:\C51\TS51\MON51.IOT CPUYPE(80517)
```

可惜的是 tScope51 只有 for Dos 的版本。

(3) Monitor 51

Monitor51 是一个监控程序通过 PC 机的串口与目标板进行通信，Monitor 操作需要 MON51 或 dScope51 for Windows,后面部分将对 Monitor51 做较为详细的介绍。

4. Ishell 及 uVision

(1) Ishell for Dos

这是一个 for Dos 的 IDE，直接在命令行键入 Ishell，则进入该环境，它使用简单方便。其命令行与 DOS 命令行具有同样的功能，对单模块的 Project 直接由菜单进行编译连接，对多模块的 project。则通过批处理，BAT 文件进行编译连接，然后通过菜单控制由 dScope51 或 tScope51 对程序进行调试，因为是 for dos 的，不做太详细介绍。

(2) uVision for Windows

uVision for Windows 是一个标准的 Windows 应用程序，它是 C51 的一个集成软件开发平台，具有源代码编辑、project 管理、集成的 make 等功能，它的人机界面友好，操作方便，是开发者的首选，具体配置及使用见第五部分。

第二章 Keil C51 软件使用详解

第一节 Keil C51 编译器的控制指令

C51 编译器的控制指令分为三类：源文件控制类，目标文件控制类及列表控制类。

1. 源文件控制类

NOEXTEND: C51 源文件不允许使用 ANSI C 扩展功能。

DEFINE(DF): 定义预处理(在 C51 命令行)。

2. 目标文件(Object)控制类:

COMPACT LARGE SMALL 选编译模式

DEBUG(DB) 包含调试信息, 以供仿真器或 dScope51 使用。

NOAMAKE(NOAM) 禁止 AutoMake 信息记录

NOREGPARMS 禁止用寄存器传递参数

OBJECTEXTEND(OE) Object 文件包含附加变量类型信息

OPTIMIZE(OT) 指定优化级别

REGFILE(RF) 指定一个寄存器使用的文件以供整体优化用

REGISTERBANK(RB) 指定一个供绝对寄存器访问的寄存器区名

SRC 不生成目标文件只生成汇编源文件

其它控件不常用。

3. 列表文件(listing)控制类:

CODE(CD): 向列表文件加入汇编列表

LISTINCLUDE(LC): 显示 indude 文件

SYMBOLS(SB): 列表文件包括模块内所有符号的列表

WARNINGLEVEL(WL): 选择“警告”级别

第二节 dScope51 的使用

1. dScope51 for Dos

总的来说 dScope51 具有以下特性:

- 高级语言显示模式
- 集成硬件环境模拟
- 单步或“GO”执行模式
- 存储器、寄存器及变量访问
- Watch 表达式之值

●函数与信号功能

下面，具体说明在进入 dScope51 for Dos 之后，如何实现上述功能，dScope51 采用下拉菜单格式和窗口显示控制，共有 language、serial、exe、register 四个窗口，其中 exe 为命令行窗口，language 为程序窗口，serial 为串口窗，register 为寄存器窗。

(1) 高级语言显示模式

单击主菜单中的“View”，第一栏中的三条命令“Highlevel”、“Mixed”、“Assembly”分别对所装入的程序按照“高级”、“混合级”及“汇编级”三种方式显示，以方便调试使用。

(2) 集成硬件环境模拟显示

主菜单中“Peripheral”各条能显示模拟硬件环境的状态，其中：

i/o Port: 显示各 I/O 口之值，对 8031 而言 SFR 中的 P1、P2、P3、P0 与引脚之值分别列出：

Interrupt: 显示 5 个中断源的入口模式是否允许，优先级等中断状态。

Timer: 显示各定时/计数器的模式，初始值状态等。

int Message: 中断信息允许，如为允许(">>"出现)，则当中断申请时，显示中断源信息。比如当中断发生时显示：

“interrupt Timer 0 occurred”等

A/D converter:

显示 A/D 转换器状态无时，则提示“无”。

Serial: 串口信息显示，包括串口模式、波特率等

Other: 其它器件，如为 8031 则显示“无”

(3) 单步或“Go”执行

“F8”单步执行，“F5”全速执行到断点。或选主菜单中 Trace 单步执行 CPU 中的 Go 全速执行。

(4) 存储器寄存器及变量访问

外部存储器管理 MAP 菜单：设置(set)、取消(reset)、显示(Display)处理可用存储

空间。

修改 Code 代码：ASM 命令

存储器显示命令：D 类别为(X、D、I、B、C)

修改存储器命令：E 有以下几种命令 EB、EC、EI、EL、EF、EP

复杂数据类型显示：Object 命令；用以显示结构或数组的内容。欲使此命令有效，C51 编译器必须有 DB 及 OBJECTEXTEND 两条。

反汇编命令：U

(5) “Watch”表达式之值

在 View 菜单的 “Watch”一栏中有四项：其中包括定义 Watch Point(Define)、删除 Watch Point(remove,kill all)，及自动更新选项。

也可用 WS、WK 等命令代替，下面具体看 “表达式” 类型：

dScope51 一次最多可设 16 个 WtchPoint 表达式，显示于 Watch Window 之中，表达式可以是简单变量，也可是复杂数据类型如结构、数组和指向结构的指针等，例如：

>WS *ptime

>WS ptime→hour

>WS some_record[o], analog 等等

(6) 关于.IOF 文件

启动 DS51 后必须装入.IOF 文件才能使 CPU 及 Peripheral 各项起作用，这个函数的使用是依据 8051 系列 CPU 的不同特点，装入 8051 各 CPU 硬件设备模拟驱动文件，比如 8031CPU 就必须 load DS51 目录下的 8051.IOF。

2. dScope for Windows

dScope for windows 具有 dScope for dos 的全部功能，此外，它还具有以下明显的优点：

- (1) 标准的 Windows 界面，操作更容易更简单；
- (2) 常用操作多用对话框，而非 Dos 的行命令方式；
- (3) 窗口资源更加丰富：存储器窗口、覆盖率分析、运行状态分析窗口，加强了调

试功能；

因为 dScope for Windows 功能强大，具体操作在第八章详细介绍。

第三节 Monitor51 及其使用

1. Monitor51 对硬件的要求

- (1) 硬件系统为 51 系列 CPU；
- (2) 带 5K 外部程序存储器(从 0 地址开始)，存放 Monitor51 程序；
- (3) 256Bytes 的外部数据存储器以及 5K 的跟踪缓冲区，此外，外部数据存储器必须足够容纳所有应用程序代码及数据，且所有外部数据存储器必须为冯·诺伊曼存储器，即能一致访问 XDATA 与 Code 空间。
- (4) 一个定时器作为波特率发生器供串口使用；
- (5) 6 Bytes 的空余堆栈。

2. Mon51 的使用

Mon51 的使用途径有三种方式：

(1) Dos 行命令方式

即先用 install 对 MON51 进行配置，然后用 MON51 进入 Monitor 状态，启用各种命令对 Monitor51 进行调试。

(2) tScope51 方式

启动 tScope51 装入 TS51 目录下的 MON51.IOT 驱动文件，与目标板通信。

(3) dScope51 for Windows 方式

在选 CPU 驱动文件时，选“MON51.dll”，则检查目标板并进入 MON51 状态。

3. MON51 的配置

(1) MON51 for Dos 的配置

运行 install 文件(在 MON51 目录下)，不同的参数可以配置不同的硬件环境。
INSTALL Serialtype [xdstart[codestart[bank][PROMCHECK]]]，具体说明见 MON51 帮助文件或使用手册。

(2) MON51 for Windows 的配置

在启用 MON51.dll 时，会使得系统自动检查目标板连接，如配置不对，则弹出“Configuration”对话框，设置 PC 串口，波特率等，完毕单击“apply”有效。

4. 串口连接图：

收发交叉互连，RTS、CTS 直连，DSR、DTR 直连，具体引脚排列参考串口资料。

5. MON51 命令及使用

详细的 MON51 命令可参阅帮助。

第四节 集成开发环境(IDE)的使用

1. Ishell for Dos 的使用

进入 Ishell 之后看到两个窗口：一个是文件窗口，一个是 Dos 命令行窗口，窗口上方是下拉式的命令菜单，其中的 Files 控制文件窗口的显隐。

使用 Ishell，第一步就是配置系统，即要学习两个文件的修改与创建：

(1) Ishell.CFG 文件

每一个 project 都有一个 Ishell.CFG，其中存放有“Option 菜单和 Setup 菜单下的部分信息；Bell enabled、Monochrome enabled、Editor Selected、CRT Lines、target enviroment、name of user edit、Automatic load for configuration enabled、file window enabled、file specification for file window、translate command line controls、project name 等。

对每个 project 都必须设置以上信息，然后存盘“setup”的“save”，这样才可正式开始下面工作。

(2) IShell.col 文件

对 IDE 颜色设置，如不改动，可以缺省为主。

(3) CDF 文件

该文件位于 BIN 目录下，每一文件定义一组外部函数工具包，即定义外部环境如 8051.CDF，USER.CDF 等，开发者可修改 CDF 文件，供自己使用，至于 CDF 文件内容可查看一下 8051.CDF 即可知道。注意.CDF 文件是 Ishell 系统的核心所在，不同的 CDF 文件可使本 IDE 适用于不同的编译、连接系统，即本 IDE 并不仅适于 C51。

下面谈一谈 Automake 工具：

C51 的 Automake 是一个 project 管理器，在 8051 工具包中以 OBJECT 文件形式保留了一个 project 的信息，AutoMake 用这些信息来进行 project 管理，一旦手工建立一个 project，Automake 可生成一个新的 OBJECT，AutoMake 利用此文件来编译那些修改过的文件。

Automake 支持 C51、A51、L51/BL51、C166、A166、L166 等编译连接器。点中主菜单中的 Automake 即运行本工具。

Ishell for Dos 使用比较繁琐，推荐使用 uVision for windows。

2. uVision for windows 的使用

uVision 是一个标准的 windows 应用程序，其编译功能、文件处理功能、project 处理功能、窗口功能以及工具引用功能(如 A51、C51、PL/M41、BL51 dScope 等)等都较 Ishell for Dos 要强得多。

uVision 采用 BL51 作连接器，因为 BL51 兼容 L51，所以一切能在 Dos 下工作的 project 都可以到 uVision 中进行连接调试。

uVision 采用 dScope for windows 作调试器，该调试器支持 MON51 及系统模拟两种方式，功能较 for DOS 要强大好用，调试功能强大。

注意：

(1) Option 菜单下的各项要会使用，其中 A51、C51、PL/M51、BL51 定义各文件所使用的编译、连接控制指令，dScope 定义一个 dScope 初始化文件。Make 则是定义一个 make 文件。

(2) 进入调试是在 RUN 菜单下运行 dScope。

(3) project 中包括新建、打开、修改、更新、编译、连接等 project 处理，具体使用可参考后面的例子。

第三章 Keil C51 vs 标准 C

深入理解并应用 C51 对标准 ANSI C 的扩展是学习 C51 的关键之一。因为大多数扩展功能都是直接针对 8051 系列 CPU 硬件的。大致有以下 8 类：

- 8051 存储类型及存储区域
- 存储模式
- 存储器类型声明
- 变量类型声明
- 位变量与位寻址
- 特殊功能寄存器(SFR)
- C51 指针
- 函数属性

具体说明如下(8031 为缺省 CPU)。

第一节 Keil C51 扩展关键字

C51 V4.0 版本有以下扩展关键字(共 19 个):

<code>_at_</code>	<code>idata</code>	<code>sfr16</code>	<code>alien</code>	<code>interrupt</code>	<code>small</code>
<code>bdata</code>	<code>large</code>	<code>_task_</code>	<code>Code</code>	<code>bit</code>	<code>pdata</code>
<code>using</code>	<code>reentrant</code>	<code>xdata</code>	<code>compact</code>	<code>sbit</code>	<code>data</code> <code>sfr</code>

第二节 内存区域(Memory Areas):

1. Program Area:

由 Code 说明可有多达 64kBytes 的程序存储器

2. Internal Data Memory:

内部数据存储器可用以下关键字说明:

data:	直接寻址区, 为内部 RAM 的低 128 字节	00H~7FH
idata:	间接寻址区, 包括整个内部 RAM 区	00H~FFH
bdata:	可位寻址区,	20H~2FH

3. External Data Memory

外部 RAM 视使用情况可由以下关键字标识:

xdata: 可指定多达 64KB 的外部直接寻址区, 地址范围 0000H~0FFFFH

pdata: 能访问 1 页(256Bytes)的外部 RAM, 主要用于紧凑模式(Compact Model)。

4. Special Function Register Memory

8051 提供 128Bytes 的 SFR 寻址区，这区域可位寻址、字节寻址或字寻址，用以控制定时器、计数器、串口、I/O 及其它部件，可由以下几种关键字说明：

sfr: 字节寻址 比如 sfr P0=0x80;为 P0 口地址为 80H, “=” 后 H~FFH 之间的常数。

sfr16: 字寻址, 如 sfr16 T2=0xcc;指定 Timer2 口地址 T2L=0xcc T2H=0xCD

sbit: 位寻址, 如 sbit EA=0xAF;指定第 0xAF 位为 EA, 即中断允许

还可以有如下定义方法:

sbit 0V=PSW^2; (定义 0V 为 PSW 的第 2 位)

sbit 0V=0XD0^2; (同上)

或 bit 0V=0xD2(同上)。

第三节 存储模式

存储模式决定了没有明确指定存储类型的变量, 函数参数等的缺省存储区域, 共三种:

1. Small 模式

所有缺省变量参数均装入内部 RAM, 优点是访问速度快, 缺点是空间有限, 只适用于小程序。

2. Compact 模式

所有缺省变量均位于外部 RAM 区的一页(256Bytes), 具体哪一页可由 P2 口指定, 在 STARTUP.A51 文件中说明, 也可用 pdata 指定, 优点是空间较 Small 为宽裕速度较 Small 慢, 较 large 要快, 是一种中间状态。

3. large 模式

所有缺省变量可放在多达 64KB 的外部 RAM 区, 优点是空间大, 可存变量多, 缺点是速度较慢。

提示: 存储模式在 C51 编译器选项中选择。

第四节 存储类型声明

变量或参数的存储类型可由存储模式指定缺省类型, 也可由关键字直接声明指

定。各类型分别用：code,data,idata,xdata,pdata 说明，例：

```
data uar1
```

```
char code array[ ]= “hello!”;
```

```
unsigned char xdata arr[10][4][4];
```

第五节 变量或数据类型

C51 提供以下几种扩展数据类型：

bit 位变量值为 0 或 1

sbit 从字节中定义的位变量 0 或 1

sfr sfr 字节地址 0~255

sfr16 sfr 字地址 0~65535

其余数据类型如：char,enum,short,int,long,float 等与 ANSI C 相同。

第六节 位变量与声明

1. bit 型变量

bit 型变量可用变量类型，函数声明、函数返回值等，存贮于内部 RAM20H~2FH。

注意：

(1) 用 #pragma disable 说明函数和用 “usign”指定的函数，不能返回 bit 值。

(2) 一个 bit 变量不能声明为指针，如 bit *ptr; 是错误的

(3) 不能有 bit 数组如：bit arr[5]; 错误。

2. 可位寻址区说明 20H—2FH

可作如下定义：

```
int bdata i;
```

```
char bdata arr[3],
```

然后：

```
sbit bit0=in0; sbit bit15=I^15;
```

```
sbit arr07=arr[0]^7; sbit arr15=arr[i]^7;
```

第七节 Keil C51 指针

C51 支持一般指针(Generic Pointer)和存储器指针(Memory_Specific Pointer).

1. 一般指针

一般指针的声明和使用均与标准 C 相同, 不过同时还可以说明指针的存储类型, 例如:

`long * state;`为一个指向 `long` 型整数的指针, 而 `state` 本身则依存储模式存放。

`char * xdata ptr;` `ptr` 为一个指向 `char` 数据的指针, 而 `ptr` 本身放于外部 RAM 区, 以上的 `long, char` 等指针指向的数据可存放于任何存储器中。

一般指针本身用 3 个字节存放, 分别为存储器类型, 高位偏移, 低位偏移量。

2. 存储器指针

基于存储器的指针说明时即指定了存储类型, 例如:

`char data * str;` `str` 指向 `data` 区中 `char` 型数据

`int xdata * pow;` `pow` 指向外部 RAM 的 `int` 型整数。

这种指针存放时, 只需一个字节或 2 个字节就够了, 因为只需存放偏移量。

3. 指针转换

即指针在上两种类型之间转化:

- 当基于存储器的指针作为一个实参传递给需要一般指针的函数时, 指针自动转化。

- 如果不说明外部函数原形, 基于存储器的指针自动转化为一般指针, 导致错误, 因而请用 “`#include`”说明所有函数原形。

- 可以强行改变指针类型。

第八节 Keil C51 函数

C51 函数声明对 ANSI C 作了扩展, 具体包括:

1. 中断函数声明:

中断声明方法如下:

```
void serial_ISR () interrupt 4 [using 1]
{
/* ISR */
```

```
}
```

为提高代码的容错能力，在没用到的中断入口处生成 `iret` 语句，定义没用到的中断。

```
/* define not used interrupt, so generate "IRET" in their entrance */
```

```
void extern0_ISR() interrupt 0 {}    /* not used */
```

```
void timer0_ISR () interrupt 1 {}    /* not used */
```

```
void extern1_ISR() interrupt 2 {}    /* not used */
```

```
void timer1_ISR () interrupt 3 {}    /* not used */
```

```
void serial_ISR () interrupt 4 {}    /* not used */
```

2. 通用存储工作区

3. 选通用存储工作区由 `using x` 声明，见上例。

4. 指定存储模式

由 `small compact` 及 `large` 说明，例如：

```
void fun1(void) small { }
```

提示：`small` 说明的函数内部变量全部使用内部 `RAM`。关键的经常性的耗时的地方可以这样声明，以提高运行速度。

5. #pragma disable

在函数前声明，只对一个函数有效。该函数调用过程中将不可被中断。

6. 递归或可重入函数指定

在主程序和中断中都可调用的函数，容易产生问题。因为 `51` 和 `PC` 不同，`PC` 使用堆栈传递参数，且静态变量以外的内部变量都在堆栈中；而 `51` 一般使用寄存器传递参数，内部变量一般在 `RAM` 中，函数重入时会破坏上次调用的数据。可以用以下两种方法解决函数重入：

a、在相应的函数前使用前述 “`#pragma disable`” 声明，即只允许主程序或中断之一调用该函数；

b、将该函数说明为可重入的。如下：

```
void func(param...) reentrant;
```

KeilC51 编译后将生成一个可重入变量堆栈，然后就可以模拟通过堆栈传递变量的方法。

由于一般可重入函数由主程序和中断调用，所以通常中断使用与主程序不同的 R 寄存器组。

另外，对可重入函数，在相应的函数前面加上开关 “#pragma noaregs”，以禁止编译器使用绝对寄存器寻址，可生成不依赖于寄存器组的代码。

7. 指定 PL/M—51 函数

由 alien 指定。

第四章 Keil C51 高级编程

本章讨论以下内容：

- 绝对地址访问
- C 与汇编的接口
- C51 软件包中的通用文件
- 段名转换与程序优化

第一节 绝对地址访问

C51 提供了三种访问绝对地址的方法：

1. 绝对宏：

在程序中，用 “#include<absacc.h>”即可使用其中定义的宏来访问绝对地址，包括：

CBYTE、XBYTE、PWORD、DBYTE CWORD 、XWORD、PBYTE、DWORD

具体使用可看一看 absacc.h 便知

例如：

rval=CBYTE[0x0002];指向程序存储器的 0002h 地址

rval=XWORD [0x0002];指向外 RAM 的 0004h 地址

2. _at_ 关键字

直接在数据定义后加上 _at_ const 即可，但是注意：

(1)绝对变量不能被初使化；

(2)bit 型函数及变量不能用_at_指定。

例如：

idata struct link list _at_ 0x40;指定 list 结构从 40h 开始。

xdata char text[25b] _at_ 0xE000; 指定 text 数组从 0E000H 开始

提示：如果外部绝对变量是 I/O 端口等可自行变化数据，需要使用 volatile 关键字进行描述，请参考 absacc.h。

3. 连接定位控制

此法是利用连接控制指令 code xdata pdata \data bdata 对“段”地址进行，如要指定某具体变量地址，则很有局限性，不作详细讨论。

第二节 Keil C51 与汇编的接口

1. 模块内接口

方法是用 #pragma 语句具体结构是：

#pragma asm

汇编行

#pragma endasm

这种方法实质是通过 asm 与 ndasm 告诉 C51 编译器中间行不用编译为汇编行，因而在编译控制指令中有 SRC 以控制将这些不用编译的行存入其中。

2. 模块间接口

C 模块与汇编模块的接口较简单，分别用 C51 与 A51 对源文件进行编译，然后用 L51 将 obj 文件连接即可，关键在于 C 函数与汇编函数之间的参数传递问题，C51 中有两种参数传递方法。

(1) 通过寄存器传递函数参数

最多只能有 3 个参数通过寄存器传递，规律如下表：

参数数目	Char	Int	long,float	一般指针
1	R7	R6 & R7	R4~R7	R1~R3
2	R5	R4 & R5	R4~R7	R1~R3
3	R3	R2 & R3		R1~R3

(2) 通过固定存储区传递(fixed memory)

这种方法将 bit 型参数传给一个存储段中：

? function_name?BIT

将其它类型参数均传给下面的段：? function_name?BYTE,且按照预选顺序存放。

至于这个固定存储区本身在何处，则由存储模式默认。

(3) 函数的返回值

函数返回值一律放于寄存器中，有如下规律：

Return type	Register	说明
Bit	标志位	由具体标志位返回
char/unsigned char 1_byte 指针	R7	单字节由 R7 返回
int/unsigned int 2_byte 指针	R6 & R7	双字节由 R6 和 R7 返回,MSB 在 R6
long&unsigned long	R4~R7	MSB 在 R4, LSB 在 R7
Float	R4~R7	32Bit IEEE 格式
一般指针	R1~R3	存储类型在 R3 高位 R2 低 R1

(4) SRC 控制

该控制指令将 C 文件编译生成汇编文件(.SRC),该汇编文件可改名后,生成汇编.ASM 文件，再用 A51 进行编译。

第三节 Keil C51 软件包中的通用文件

在 C51\LiB 目录下有几个 C 源文件，这几个 C 源文件有非常重要的作用，对它们稍事修改，就可以用在自己的专用系统中。

1. 动态内存分配

init_mem.C: 此文件是初始化动态内存区的程序源代码。它可以指定动态内存的位置及大小，只有使用了 init_mem()才可以调回其它函数，诸如 malloc calloc,realloc 等。

calloc.c: 此文件是给数组分配内存的源代码，它可以指定单位数据类型及该单元数目。

malloc.c: 此文件是 malloc 的源代码，分配一段固定大小的内存。

realloc.c: 此文件是 realloc.c 源代码，其功能是调整当前分配动态内存的大小。

2. C51 启动文件 STARTUP.A51

启动文件 `STARTUP.A51` 中包含目标板启动代码，可在每个 `project` 中加入这个文件，只要复位，则该文件立即执行，其功能包括：

- 定义内部 RAM 大小、外部 RAM 大小、可重入堆栈位置
- 清除内部、外部或者以此页为单元的外部存储器
- 按存储模式初使化重入堆栈及堆栈指针
- 初始化 8051 硬件堆栈指针
- 向 `main()`函数交权

开发人员可修改以下数据从而对系统初始化

常数名	意义
IDATALEN	待清内部 RAM 长度
XDATA START	指定待清外部 RAM 起始地址
XDATALEN	待清外部 RAM 长度
IBPSTACK	是否小模式重入堆栈指针需初始化标志，1 为需要。缺省为 0
IBPSTACKTOP	指定小模式重入堆栈顶部地址
XBPSTACK	是否大模式重入堆栈指针需初始化标志，缺省为 0
XBPSTACKTOP	指定大模式重入堆栈顶部地址
PBPSTACK	是否 Compact 重入堆栈指针，需初始化标志，缺省为 0
PBPSTACKTOP	指定 Compact 模式重入堆栈顶部地址
PPAGEENABLE	P2 初始化允许开关
PPAGE	指定 P2 值
PDATASTART	待清外部 RAM 页首址
PDATALEN	待清外部 RAM 页长度

提示：如果要初始化 P2 作为紧凑模式高端地址，必须：`PPAGEENAGLE=1`，`PPAGE` 为 P2 值，例如指定某页 `1000H—10FFH`，则 `PPAGE=10H`，而且连接时必须如下：
`L51<input modules> PDATA(1080H)`，其中 `1080H` 是 `1000H—10FFH` 中的任一个值。
以下是 `STARTUP.A51` 代码片断，红色是经常可能需要修改的地方：

;

```

; This file is part of the C51 Compiler package
; Copyright KEIL ELEKTRONIK GmbH 1990
;-----
;  STARTUP.A51:  This code is executed after processor reset.
;
;  To translate this file use A51 with the following invocation:
;
;      A51 STARTUP.A51
;
;  To link the modified STARTUP.OBJ file to your application use the following
;  L51 invocation:
;
;      L51 <your object file list>, STARTUP.OBJ <controls>
;
;-----
;
;  User-defined Power-On Initialization of Memory
;
;  With the following EQU statements the initialization of memory
;  at processor reset can be defined:
;
;                               ; the absolute start-address of IDATA memory is always 0
IDATALEN EQU                80H    ; the length of IDATA memory in bytes.
;
XDATASTART                  EQU    0H    ; the absolute start-address of XDATA
memory
XDATALEN EQU                0H    ; the length of XDATA memory in bytes.

```

```

;
PDATASTART          EQU      0H      ; the absolute start-address of PDATA
memory
PDATALEN EQU          0H          ; the length of PDATA memory in bytes.
;
; Notes:  The IDATA space overlaps physically the DATA and BIT areas of the
;          8051 CPU. At minimum the memory space occupied from the C51
;          run-time routines must be set to zero.
;-----
;
; Reentrant Stack Initilization
;
; The following EQU statements define the stack pointer for reentrant
; functions and initialized it:
;
; Stack Space for reentrant functions in the SMALL model.
IBPSTACK EQU          0 ; set to 1 if small reentrant is used.
IBPSTACKTOP          EQU      0FFH+1  ; set top of stack to highest location+1.
;
; Stack Space for reentrant functions in the LARGE model.
XBPSTACK EQU          0 ; set to 1 if large reentrant is used.
XBPSTACKTOP          EQU      0FFFFH+1; set top of stack to highest location+1.
;
; Stack Space for reentrant functions in the COMPACT model.
PBPSTACK EQU          0 ; set to 1 if compact reentrant is used.
PBPSTACKTOP          EQU      0FFFFH+1; set top of stack to highest location+1.
;

```

```

;-----
;
; Page Definition for Using the Compact Model with 64 KByte xdata RAM
;
; The following EQU statements define the xdata page used for pdata
; variables. The EQU PPAGE must conform with the PPAGE control used
; in the linker invocation.
;
PPAGEENABLE          EQU    0    ; set to 1 if pdata object are used.
PPAGE                 EQU    0    ; define PPAGE number.
;
;-----

```

3. 标准输入输出文件

putchar.c

putchar.c 是一个低级字符输出子程，开发人员可修改后应用到自己的硬件系统上，例如向 CLD 或 LEN 输出字符。

缺省：putchar.c 是向串口输出一个字符 XON|XOFF 是流控标志，换行符 “*n”自动转化为回车/换行 “\r\n”。

getkey.c

getkey 函数是一个低级字符输入子程，该程序可用到自己硬件系统，如矩阵键盘输入中，缺省时通过串口输入字符。

4. 其它文件

还包括对 Watch-Dog 有独特功能的 INIT.A51 函数以及对 8×C751 适用的函数，可参考源代码。

第四节 段名协定与程序优化

1. 段名协定(Segment Naming Conventions)

C51 编译器生成的目标文件存放于许多段中，这些段是代码空间或数据空间的一些单元，一个段可以是可重定位的，也可以是绝对段，每一个可重定位的段都有一个类型和名字，C51 段名有以下规定：

每个段名包括前缀与模块名两部分，前缀表示存储类型，模块名则是被编译的模块的名字，例如：

? CO? main1 : 表示 main1 模块中的代码段中的常数部分

? PR? function1? module 表 module 模块中函数 function1 的可执行段，具体规定参阅手册。

2. 程序优化

C51 编译器是一个具有优化功能的编译器，它共提供六级优化功能。确保生成目标代码的最高效率(代码最少，运行速度最快)。具体六级优化的内容可参考帮助。

在 C51 中提供以下编译控制指令控制代码优化：

OPTIMIZE(SJXE)：尽量采用子程序，使程序代码减少。

NOAREGS：不使用绝对寄存器访问，程序代码与寄存器段独立。

NOREGPARMS：参数传递总是在局部数据段实现，程序代码与低版本 C51 兼容。

OPTIMIZE(SIZE)AK OPTIMIZE(speed) 提供 6 级优化功能，缺省为：OPTIMIZE(6,SPEED)。

第五章 Keil C51 库函数参考

C51 强大功能及其高效率的重要体现之一在于其丰富的可直接调用的库函数，多使用库函数使程序代码简单，结构清晰，易于调试和维护，下面介绍 C51 的库函数系统。

第一节 本征库函数(intrinsic routines)和非本征证库函数

C51 提供的本征函数是指编译时直接将固定的代码插入当前行，而不是用 ACALL 和 LCALL 语句来实现，这样就大大提供了函数访问的效率，而非本征函数则必须由 ACALL 及 LCALL 调用。

C51 的本征库函数只有 9 个，数目虽少，但都非常有用，列如下：

`_crol_`, `_cror_`: 将 `char` 型变量循环向左(右)移动指定位数后返回

`_iror_`, `_irol_`: 将 `int` 型变量循环向左(右)移动指定位数后返回

`_lrol_`, `_lror_`: 将 `long` 型变量循环向左(右)移动指定位数后返回

`_nop_`: 相当于插入 NOP

`_testbit_`: 相当于 JBC bitvar 测试该位变量并跳转同时清除。

`_chkfloat_`: 测试并返回源点数状态。

使用时，必须包含 `#include <intrins.h>` 一行。

如不说明，下面谈到的库函数均指非本征库函数。

第二节 几类重要库函数

1. 专用寄存器 include 文件

例如 8031、8051 均为 `REG51.h` 其中包括了所有 8051 的 SFR 及其位定义，一般系统都必须包括本文件。

2. 绝对地址 include 文件 `absacc.h`

该文件中实际只定义了几个宏，以确定各存储空间的绝对地址。

3. 动态内存分配函数，位于 `stdlib.h` 中

4. 缓冲区处理函数位于 “`string.h`” 中

其中包括拷贝比较移动等函数如：

`memccpy memchr memcmp memcpy memmove memset`

这样很方便地对缓冲区进行处理。

5. 输入输出流函数，位于 “`stdio.h`” 中

流函数通 8051 的串口或用户定义的 I/O 口读写数据，缺省为 8051 串口，如要修改，比如改为 LCD 显示，可修改 `lib` 目录中的 `getkey.c` 及 `putchar.c` 源文件，然后在库中替换它们即可。

第三节 Keil C51 库函数原型列表

1. `CTYPE.H`

`bit isalnum(char c);`

```

bit  isalpha(char c);
bit  iscntrl(char c);
bit  isdigit(char c);
bit  isgraph(char c);
bit  islower(char c);
bit  isprint(char c);
bit  ispunct(char c);
bit  isspace(char c);
bit  isupper(char c);
bit  isxdigit(char c);
bit  toascii(char c);
bit  toint(char c);

char  tolower(char c);
char  __tolower(char c);
char  toupper(char c);
char  __toupper(char c);

```

2. INTRINS.H

```

unsigned char _crol_(unsigned char c,unsigned char b);
unsigned char _cror_(unsigned char c,unsigned char b);
unsigned char _chkfloat_(float ual);
unsigned int _irol_(unsigned int i,unsigned char b);
unsigned int _iror_(unsigned int i,unsigned char b);
unsigned long _irol_(unsigned long l,unsigned char b);
unsigned long _iror_(unsigned long L,unsigned char b);
void _nop_(void);
bit _testbit_(bit b);

```

3. STDIO.H


```

char getchar(void);
char _getkey(void);
char *gets(char * string,int len);
int printf(const char * fmtstr[,argument]...);
char putchar(char c);
int puts (const char * string);
int scanf(const char * fmtstr[,argument]...);
int sprintf(char * buffer,const char *fmtstr[,argument]);
int sscanf(char *buffer,const char * fmtstr[,argument]);
char ungetchar(char c);
void vprintf (const char *fmtstr,char * argptr);
void vsprintf(char *buffer,const char * fmtstr,char * argptr);

```

4. STDLIB.H

```

float atof(void * string);
int atoi(void * string);
long atol(void * string);
void * calloc(unsigned int num,unsigned int len);
void free(void xdata *p);
void init_mempool(void *data *p,unsigned int size);
void *malloc (unsigned int size);
int rand(void);
void *realloc (void xdata *p,unsigned int size);
void srand (int seed);

```

5. STRING.H

```

void *memccpy (void *dest,void *src,char c,int len);
void *memchr (void *buf,char c,int len);
char memcmp(void *buf1,void *buf2,int len);

```

```

void *memcpy (void *dest,void *SRC,int len);
void *memmove (void *dest,void *src,int len);
void *memset (void *buf,char c,int len);
char *strcat (char *dest,char *src);
char *strchr (const char *string,char c);
char strcmp (char *string1,char *string2);
char *strcpy (char *dest,char *src);
int strcspn(char *src,char * set);
int strlen (char *src);
char *strncat (char *dest,char *src,int len);
char strncmp(char *string1,char *string2,int len);
char strncpy (char *dest,char *src,int len);
char *strpbrk (char *string,char *set);
int strpos (const char *string,char c);
char *strrchr (const char *string,char c);
char *strrpbkr (char *string,char *set);
int strrpos (const char *string,char c);
int strspn(char *string,char *set);

```

第六章 Keil C51 例子: Hello. c

Hello 位于\C51\excmpls\Hello\目录，其功能是向串口输出 “Hello,world”整个程序如下：

```

#pragma DB OE CD
#include <reg51.h>
#include<stdio.h>
void main(void)
{
    SCON=0x50;

```

```

TMOD=0x20

TH1=0xf3;

Tri=1;

TI=1;

printf("Hello,world \n");

while(1) {   }

}

```

第一节 uVision for Windows 的使用步骤

(1) file_new 新建一个 hello.c 文件，输入如上内容或直接用目录下源文件。

(2) file_save 或工具栏将文件存盘。

(3) project_new project 创建一个 project 名为 hello 并在其中加入 hello.c。

这时该 project 已是打开状态，或用 open project 打开已存在的 project。

(4) option_C51 compiler 中选出至少包括两项 DB OE。

(5) option_dscope Debugger 选中 hello\DS51.INI

查看 DS51.INI 看其是否为：

```

“load...\BIN\8051.DLL

map 0, 0xffff”

```

否则修改。

(6) 在 option_make 选 make 文件顺序。

(7) project 选 Build project，看是否有语法错误，若无则生成 HEX 文件，若有则修改源文件后重复以上部分步骤。

(8) run_dScope debugger 进入 dScope51 后装入 hello 则可用 go 直接运行看 serial 窗口有无输出，正常每系统运行一次，serial 窗口均出现一个“Hello,world”表明运行无误。

第二节 Ishell for Dos 使用步骤

(1) 进入 Ishell 用 Setup editier 选择编辑器。

然后单击 Edit 或用 Edit 命令编辑 hello.c 源文件，存盘，也可以在 files 窗口中直接选

中 hello.c。

(2) 用 cd 改换 project 目录至 hello 目录。

(3) 在 setup_target 一项目选 8051

(4) 在 setup_C51 中输出 DB OE。

(5) 在 setup_project 输入 project 名 hello。

(6) 在 setup_save 保存 Ishell.CFG 文件。

(7) 编辑一个 Link 文件 hello.lin 中有 “hell.obj”一行。

(8) 由光标落在 files 菜单中的 Hello.c 上，单击“translate”，如无语法错，再击“link”，则 Hex 文件生成。

(9) 单击 Simulate 如在 8051.CDF 中选 Simulate 为 dScope 则进入 dScope 调试直接“Go”，看 serial 窗口输出为 “Hello.world”。

(10) 如程序有误修改源代码后不必再 translate 或 link 了，只要一步 Amake 即可。

若 project 中包括不止一个文件，在 DOS 的 Ishell 中不能用 Translate 编译，而应建立 bat 文件，直接在命令窗编译，然后 link 连接。

如还需用 Translate 则只能多个文件分别编译，然后连接。

第七章 Keil C51 的代码效率

C51 程序编译生成汇编代码的效率，是由许多因素共同决定的，对于 Keil C51，主要受以下两种因素影响：

第一节 存储模式的影响

存储模式决定了缺省变量的存储空间，而访问各空间变量的汇编代码的繁简程度决定了代码率的高低。

例如：一个整形变量 i，如放于内存 18H、19H 空间，则++i 的操作编译成四条语句：

```
INC 0x19
```

```
MOV A,0x19
```

```
JNZ 0x272D
```

```
INC 0x18
```

0x272D

而如果放于外存空间 0000H、0001H 则++i 的操作编译成九条语句：

MOV DPTR, 0001

MOVX A, @ DPTR

INC A

MOVX @ DPTR,A

JNz #5

MOV OPTR,#0000

MOVX A,@DPTR

INC A

MOVX @ DPTR,A

就汇编之后的语句而言，对外部存储器的操作较内部存储器操作代码率要低得多，生成的语句为内存的两倍以上，而程序中有大量的这种操作，可见存储模式对代码率的响了。

因此程序设计的原则是

- 1、存储模式从 small-Compact-large 依次选择，实在是变量太多，才选 large 模式。
- 2、即使选择了 large 模式，对一些常用的局部的或者可放于内存中的变量，最好放于内存中，以尽量提高程序的代码率。

第二节 程序结构的影响

程序的结构单元包括模块、函数等等。同样的功能，如果结构越复杂，其所涉及的操作、变量、功能模块函数等就越多，较之结构性好，代码简单的程序其代码率自然就低得多。

此外程序的运行控制语句，也是影响代码率的关键因素，例如：switch -case 语句，许多编译器都把它们译得非常复杂，Keil C51 也不例外，相对较为简易的 Switch-case 语句，编译成跳转指令形式，代码率较高，但对较为复杂的 Switch-Case，则要调用一个系统库函数?C?ICASE 进行处理，非常复杂。

再如 if(),while(), 等语句也是代码相对较低的语句, 但编译以后比 switch-case 要高得多。

因此建议设计者尽量少用 switch-case 之类语句来控制程序结构, 以提高代码率。

除以上两点外, 其它因素也会对代码率产生影响, 例如:

是否用寄存器传递参数 即 NOAREGS 选项是否有

是否包括调试信息: 即 DEBUG 选项

是否包括扩展的调试信息: 即 BJECTEXTEND

第八章 dScope for Windows 使用详解

第一节 概述

1. 主窗口 (Mainframe Window)

可设置其它各种调试窗口, 设置断点、观察点, 修改地址空间, 加载文件等等;

2. 调试窗口 (DEBUG Window)

支持用户程序的各种显示方式, 可连续运行, 单步运行用户程序, 并可在线 汇编;

3. 命令窗口 (Command Window)

支持命令行的输入;

4. 观察窗口 (Watch Window)

可设置所要观察的变量、表达式等;

5. 寄存器窗口 (Registe Window)

显示内部寄存器的内容, 程序运行次数等;

6. 串口窗口 (Serical Windows)

显示串口接收和发送的数据;

7. 性能分析窗口

显示所要观察的各程序段占用 CPU 的空间;

8. 内存窗口 (Memory Window)

显示所选择的内存中的数据;

9. 符号浏览窗口 (Symbol Browser Window)

显示各种符号名称，包括专有符号，用户自定义符号（函数名、变量、标号）等；

10. 调用线窗口（Call-Stack Window）

动态显示当前执行的程序段的函数调用关系；

11. 代码覆盖窗口

提供当前模块内各程序段中被执行代码的比率；。

12. 外围设备窗口(peripherals)

可显示 I/O 口，定时器，中断，串口等外围设备状态；

第二节 dScope for Windows 基本操作

1. 指定初始化文件

在 uVision 的 Option 菜单 dScope Debugger 中指定 dScope 的初始化文件，用 uVision 的 RUN 启动 dScope 将自动加载此初始化文件，自动执行其中命令；

下面是一个例子，可以看出调入一个调试代码的过程。Ds51.ini:

```
load 8051.dll
```

```
load test
```

```
slog>>test.log
```

```
xtal=11.0592
```

```
define button "go to main","g,main"
```

```
ws RevCounter
```

```
ws rm.r
```

```
g,main
```

```
PA RESET
```

```
PA serial
```

```
PA timer0
```

2. 观察变量

方法 1：命令行

```
WS expression [, numberbase ] [ LINE ]
```

其中 **numberbase** 为显示数制，10 对应 10 进制，16 对应 16 进制，缺省为 16 进制。

LINE 为单行显示，缺省为多行显示。

方法 2: **setup > Watchpoints**，在对话框中输入变量

3. 显示 RAM 的值

d i(x,d):起始地址,终止地址

d 变量名

4. 观察堆栈

View->Call-stack->Show invocation，可以跟踪调用过程；

5. 中断处理程序调试

在装入 8051.dll 后，在 **dScope** 的主菜单中将增加 **Peripheral**，其有 4 个字菜单：

I/O port: **Pi** 端口状态

Interrupt: 中断设置

Timer: 定时器中断状态

Serial 串口中断状态

设置相应的中断请求标志位即可产生中断。

6. 性能分析 (**Performance Analyzer: PA**)

PA 用来分析一段代码执行占用 **CPU** 的百分比。定义：

命令行 **PA func_name**

第三节 **dScope for Windows** 命令文件的编制

dScope 除了用命令行的方式进行调试以外，还可将各种调试命令汇集于一个调试文件中，然后调用该文件，就可达到自动测试用户源代码的目的。**dScope** 的命令文件支持 **C/PL/M** 的格式，因而编制调试命令文件与编制 **C** 语言程序有些类似。

1. 地址空间及地址空间类型

(1) 地址空间分段

dScope 提供的最大可用空间为 16M，实际上我们只用以下三段：

① 内部数据空间段（0X00 段或 D 段）

0X00:0X0000~0X00:0XFFFF(对 MCS51 而言为 0X00:0X00FF)

② 外部数据空间段（0X01 段式或 X 段）

0X01 0X0000~ 0X01~0XFFFF

③ 程序空间段（0XFF 段或 C 段）

0XFF: 0X0000~ 0XFF: 0XFFFF

(2) 地址空间类型

C: 代码空间

D: 内部直接寻址空间

I: 内部间接寻址空间

X: 外部数据空间

B: 位寻址空间

P: I/O 口

EB: 扩展的位寻址空间（MCS251 专有）

ED: 扩展的数据空间（MCS251 专有）

CO: 常数空间（MCS251 专有）

HC: 正常数空间（MCS251 专有）

2. 常量

dScope 支持十六进制、八进制、十进制、二进制常数，其后缀分别为 H、Q(O)、T（或无）、Y；

dScope 不区分常量的大、小写。

(1) 整型常量

分为整型(int)，无符号整型(uint,00rd)，长整型(long)，无符号长整型(Wlong、Word)。

(2) 浮点型常量

与 ANSI C 相同。

(3) 字符串常量

与 ANSI C 相同

(4) 字符常量

分为字符型 (Char) 和无符号字符型 (Uchar) 一种。

(5) 行号常数

指用户程序中的行号，实际上是个地址

(6) 位常量 (Bit):

0 和 1

(7) 地址常数

地址常数的种类很多，地址常数不同于行号常数，行号常数就是一个地址，而地址数被引用时，实际上是取该地址中的数据。

C: 代码地址常数，如 C: 0X0012 或 0XFF: 0X0012

D: 内部直接寻址地址常数，如 D: 0X0068 或 0X00 0X0068

I: 内部间接寻址地址常数，如 I: 0X0010 或 0X00 0X0010

X: 外部数据空间地址常数，如 X: 0X0028 或 0X01 0X0028

B: 位地址常数，如 B: 0X20 或 B: 0X24.0

EB: 扩展的位地址常数 (MCS251 专有)，

ED: 扩展的数据空间地址常数 (MCS251 专有)

CO: 常数空间地址常数 (MCS251 专有)

HC: 正常数空间地址常数 (MCS251 专有)

(8) 标识符常量

即用户源程序中的标号、函数名等，实际上代表某一地址。

(9) 用户源程序中定义的常数

3. 变量

dScope 所支持的变量名或标识符最多可由 31 个字符组成，第一个字母为 A~Z, a~z, 下划线或问号，后续字符可为字母、数字、下划线和问号。除 CPU 变量和系

统变量外，dScope 不支持全局变量，但可视 “define”命令定义的变量为全局变量。

Dscope 所支持的变量分为以下几种（变量名称不区分大、小写），支持类型转换：

(1) 整型变量

分为整型变量（int）、无符号整型变量（uint/word），长整型（Long） 、无符号长整型（Ulong/dword）。

(2) 浮点型变量(float)

与 ANSI C 相同。

(3) 字符型变量 L

分为字符型（char）变量和无符号字符型（Uchar）

(4) 位变量（Bit）

(5) 系统变量

dScope 自己定义了一系列内部变量，用户可对这些变量进行读或读/写操作，可被用户自定义数所引用。

a. Cycles (Read Only)

32 位变量（Ulong），指示当前程序执行已花费的指令周期（cycle）。

b. Ramsize(R/W)

16 位变量（Uint），指示内部可直接寻址的数据空间大小。

c. Radix(R/N)

8 位变量（Uchar），决定输出的数制

Radix=0X0A (10 进制)，Radix=0X10 (16 进制)

d. —IIP—（R/W）

8 位变量（Uchar），指示当前的中断嵌套数目。

e. \$（R/W）

32 位变量（Ulong），指出 PC 值，通过对其进行写操作，可改变程序执行的流程。

f. Itrace (R/W)

8 位变量 (Uchar)，决定是否对程序运行情况进行记录

Itrace=1，使能记录操作

Itrace=0，根本上记录操作

g. `__Break__`(R/W)

8 位变量 (Uchar) `__Break__=1`，中止程序的运行

h. `__Mode__` 和 `__Frame size__` 是 MCS 251 专有的变量。

(6) CPU 变量

即 R0~R7、A、C (位变量)、B、DPTR 及特殊功能寄存器变量，对这些变量均可进行读、写操作。

(7) 用户源程序中定义的变量、数组、结构等

4. 运算符

dScope 支持 ANSI C 的运算符，包括算术运算符，逻辑运算符，关系运算符。

5. 表达式

以运算符将 dScope 所支持的常量、变量、函数等连接在一起，就构成了 dScope 的表达式。

6. 数组

dScope 不支持在命令文件中定义数组，但可引用用户程序中的数组，引用方式如同 C。

7. 结构和联合

dScope 不支持在命令文件中定义结构和联合，但可引用用户程序中的结构和联合，引用方式如同 C，但如要输出整个结构或联合的结果，就要用命令 “OBJ”。

8. 指针：

不可自定义指针，但支持用户源程序中的指针变量。

9. dScope 命令语句

dScope 提供了一系列调试命令。在命令文件中，dScope 只支持这些语句及前述定义的表达式，C 语言的语句均不被支持，但在命令文件所包含的用户自定义函数(非

用户源程序中的函数) 中支持 C 语句, 但用户自定义函数中同样不支持数组、结构、联合和指针。

(1) ASM

在线汇编命令, 格式如下:

ASM C: 0Xnnnn (或标号); 设定插入汇编指令的地址

ASM 汇编指令

ASM 汇编指令

插入完毕后, 在 debug 窗口内选择 “Assemble->Assemble” 完成编译。

(2) Assign

串行口分配指令, 格式如下:

Assign channel<unreg>outreg

对 MCS51 为: Assign Win<SOIN> Soot

但目前的 dScope 版本并未提供完整串口窗口功能。

(3) Define

用户自定义变量指令, 格式如下:

Define <类型> <变量名>

类型一为如前所述的变量类型, Define 指令定义的变量可能为全局变量, 可为用户自定义函数所引用。

(4) Display

内存显示命令, 格式如下二:

D 起始地址, 结束地址

地址如前所述的地址常数, 标识符常量。

(5) Enter

内存修改指令, 格式如下:

E 类型地址=表达式 [表达式 2], [.....]

类型如前所述，地址如前所述的地址常数。表达式如前所述，但如果是函数名称（含标号、指针变量），则关键字 $E \rightarrow EP$

(6) Map/Reset map

Map 为内存段修改指令，Reset map 将内存段复位或缺省值。

(7) Object

用以引用用户源程序中的结构（联合）、数组、格式如下：

Obj 表达式 $[n,], [Line]$

表达式为用户源程序中的数组，结构（联合）名称。当 Line 缺省时，数目、结构（联合）的内容按 n 行输出；如有 Line，则单行输出。

(8) U

反汇编命令，格式如下：

U [地址]

地址包括地址常数及标识符常量，指明反汇编的起始地址。

(9) WK

观察点删除命令，格式如下：

WK $n1[n2], [.....]$ ；删除指定的观察点， n 为字符型，整型常数

WK * ；删除所有的观察点

(10) WS

观察点设置命令，格式如下：

WS 表达式 $[,n][LINE]$

关键字 LINE 存在时，观察点表达式单行输出

LINE 缺省时，观察点表达式 n 行输出。

(11) G

连续运行命令，格式如下：

G [起始地址], [终止地址]

地址为标识符常量或地址常数，地址缺省时，为连续运行。

(12) T/P

单步运行指令，格式如下：

T/P n ; n 指至单行运行的步数，P 指给用户当调用某函数时，把它作为一步处理，并不进入该函数运行。

(13) PA

性能分析操作指令，其分以下几种：

PA

显示当前所设置的性能分析程度段

PA Kill *

删除当前所设置的所有性能分析程序段

PA Kill n1 [,n2], [.....]

删除指定的性能分析程序段

PA 地址范围

设置性能分析程序段，地址范围可以起始地址和结束地址的方式给出，也可给出函数名，行号范围。

PA Reset

复位性能分析窗口（PA Windows），清除所有的记录。

(14) BD

断点失效命令，格式如下：

BD n1 [,n2], [.....] ; disable 指定的断点

DB * ; disable 所有的断点

(15) BE

断点使能命令，格式如下：

BE M [,n2], [.....] ; 使能指定的断点

BE * ; 使能所有的断点

(16) BK

断点删除指令，格式如下：

BK M[n2], [,.....] ; 删除指定的断点

BK * ; 删除所有的断点

(17) BL

断点显示指令，显示所有被定义的断点。

(18) BS

断点定义指令，dScope 支持多达 40 个断点，具体格式如下：

a.BS 表达式[,count] [,"cmd"]

count 经过该断点的次数 [选项]

“cmd”：断点到达后附带执行的 dScope 命令（连项）

表达式一个条件表达式，此时该断点称为条件断点（运算符为&.&&,<=>,>=,=,!=）

BS READ 表达式 [,count] [,"cmd"]

BS WRITE 表达式 [,count] [,"cmd"]

BS READWRITE 表达式 [,count] [,"cmd"]

以上三种断点称访问式（Access 断点），当某一址或变量被访问（R/W）或某些值被读写时，程序被中断。

(19) Define button

图标定义指令，用于当窗口（Toolbox）

(20) !

DOS 窗口 Open 命令，以 “EXIT”命令退出 DOS 窗口。

(21) Include

文件包含命令，格式如下：

Include [路径] 文件名

dScope 支持以文件包含的方式调入并执行调试命令文件，用户自定义函数文件，调试命令文件可以有后缀，也可无后缀。

(22) Load

加载命令，格式如下：

Load [路径] 文件名

Load 指令能够加载的文件必须具有以下格式之一。

Intel Hex/Hex 386 格式

Intel Object (OMF_51) 格式

Intel Object (OMF-251) 格式

dScope 的 CPU 驱动文件（.DLL）

(23) LOG

Command Window 存盘指令，用于将 Command Windows 中的内容输出到指定的文件中，格式如下：

LOG > [路径]文件名 ；创建一个新文件

LOG >> [路径]文件名 ；将 Command Windows 的内容输出到某个已存在的文件中。

LOG OFF 完成输出操作并开闭该文件

LOG 指令只将 LOG>或 LOG>>与 LOG OFF 指令之间的操作命令存入该指定文件。

(24) Reset

复位指令，具体格式如下：

Reset ；执行 dScope 的复位

Reset Map ；复位外部数据空间

Reset Var ；复位 SET 指令定义的变量

(25) Save

该指令将一段内存映象以 19EX386/HEX 的格式存盘，具体格式如下：

Save 路径 文件名：地址 1、地址 2

地址 1、地址 2 指所要保存的空间范围，既可是标识符，也可是址常数。

(26) SET

该指令用来定义 dScope 目标代码预定义变量的含义，这些预定义变量包括以下二种：

SRC ; 指出所在的路径

F1~F12; 对应于键盘上的 12 个功能键, 定义这些功能键的含义。

SET 指令的格式为:

SET 变量 = “字符串”

SET 变量

10. 函数

dScope 支持三种函数, 即 **dScope** 预定义函数, 用户自定义函数和信号函数, 分别详述如下:

(1) **dScope** 预定义函数

dScope 号提供 8 个预定义函数 (可视为 **dScope** 的库函数)

① **Void Printf(“String”, 输出表列)**

屏幕打印函数, 与 ANSI C 的 **Printf()** 函数相同

② **Void exec(“Command__String”)**

Command__String 为一有效的命令字符串, 此函数用于在运行用户自定义函数的过程中执行 **dScope** 命令, 这个函数提供了一个很重要的编制测试命令文件的方法。

③ **int getint(“Prompt__String”);** 从键盘输入一个整数

int getlong (“Prompt__String”); 从键盘输入一个长整数

float getfloat (“Prompt__String”); 从键盘输入一个浮点数

以上这三个函数被执行时, **dScope** 会弹出一个 **dialog box** 等待用户输入数据, 其标题栏上是 “**Prompt__String**”, 利用这个函数, 不仅可以为变量赋值, 也可使用户得以看清前一阶段的测试结果。

④ **int rand (int seed)**

该函数会输出一个随机数 (-32768~32768)

⑤ **Void memset (ulong start ,ulong end ,uchar val)**

该函数用于给地址范围 (**Start__end**) 内的内存赋值 (**Val**)

⑥ **Void twatch (Long cycles)**

定时函数，时间由（Long cycles）决定，它是以指令周期计数的，它也用于产生一个信号波形，该函数必须用于信号函数中。

(2) 用户自定义函数

这类函数不同于用户源程序中的函数，其定义格式为

Func 返回类型 函数名（参数序列）

```
{  
    语句  
}
```

返回类型如前所述的变量类型

用户自定义函数中的语句与 ANSI C 相似，只是不支持数组结构、联合、指针，可引用 dScope 系统变量，define 语句定义的变量和用户源程序变量，不支持 dScope 命令，如想在函数中执行 dScope 命令，要借助于 exec（“Command__String”）函数，可引用 dScope 预定义的函数（除了 twatch（）函数），不支持 ANSI C 的库函数。

(3) 信号函数

用于产生具有某一波形的信号，定义格式为：

Signal 返回类型函数名（参数长列）{
 语句
}

信号函数主要是利用 twatch（）函数，目前 dScope 版本在提供这一功能上面还有一定问题。

(4) dScope 函数与 ANSI 函数的区别

- ① 不支持条件汇编
- ② 不支持头文件
- ③ 无变量的初始化
- ④ 不支持数组、结构、指针

⑤ 调用方式不同，自定义函数和信号函数首先要包含一个函数文件之中，然而在测试命令文件中以 **Inclule** 指令调用该函数文件，最后才能以函数名调用之。

⑥ 函数调用只支持传值方式。

©电子设计世界! James

20002-13