

富士通电子设备
用户手册

F²MC-16LX Starter kit



用户手册

版本信息

日期	内容
2005.06.27	1.0版；首版
2005.09.12	1.1 版；更改 图 5-4、图 7-5、图 7-7、表 7-3
2005.11.25	1.2 版；图 3-5、5-3、5-4、7-4、7-5、8-3，表 1-1、1-2、1-3 修改及部分程序图修改

注意事项

- 本资料有关内容如有变更恕不另行通知。
- 本资料内所记载的设备运行情况及电路实例均是以半导体设备的标准规格及正确的使用方法为前提的，我们并不保证实际使用时所有机械的正常运作。因此，在使用此设备时，顾客将完全承担相应的使用责任。如有因使用此设备而造成的损害，本公司将不承担任何责任。
- 本资料内所记载的设备运行情况以及电路图内所含有的技术资料并不代表可以任意使用本公司以及第三责任方的专利权以及著作权。禁止通过本资料对第三责任方的知识产权以及相关的权利进行侵犯。本公司对于此类相关行为以及所产生的后果将不负任何责任。
- 本资料内若含有属于《外国汇率以及外国贸易法》范畴内的商品，或者含有相关范畴内的技术，则在出口本商品时必须得到相关法律的认可。

Copyright© 2005 FUJITSU LIMITED ALL right reserved

目录

前言	6
1 Starter-kit 的安装方法	7
1.1 PC 机上的软件安装	13
1.1.1 USB 驱动的安装	14
1.1.2 综合开发环境 SOFTUNE（限定版）的安装	15
1.1.3 ACCEMIC MDE demo version(trial 版)的安装	20
1.1.4 评测板的设定以及与 PC 机的连接	24
1.1.5 SOFTUNE 的设定与启动	27
1.1.6 ACCEMIC MDE 的设定与启动	30
1.1.7 ACCEMIC MDE 退出	44
1.1.8 SOFTUNE 的退出	44
1.1.9 关闭 Accemic 的情况下启动单片机	45
2 编写使 LED 闪烁的程序	46
2.1 关于 LED 的介绍	46
2.2 LED 为何会发光	47
2.3 利用单片机使 LED 发光的方法	48
2.4 LED 发光程序的制作及运行	51
2.4.1 程序概要	51
2.4.2 程序的制作与运行	52
2.5 LED 闪烁程序的制作与运行	54
2.5.1 程序概要	54
2.5.2 程序的制作与运行	55
3 用开关 SW 控制 LED 的亮灭	57
3.1 单片机如何检测 SW 的状态	57
3.2 通过 SW 控制 LED 程序的制作与运行	58
3.2.1 程序概要	59
3.2.2 程序的制作与运行	59
4 如何使用蜂鸣器	61
4.1 蜂鸣器内所用的材料	61
4.1.1 压电性的特点	61
4.1.2 压电材料的应用	62
4.2 单片机与压电蜂鸣器	62

4.2.1	自励式与他励式	63
4.2.2	由单片机发出的脉冲波	63
4.3	如何使用 PPG 使蜂鸣器发出声音	63
4.3.1	L 幅宽与 H 幅宽的设定	64
4.3.2	PPG count clock.....	64
4.4	蜂鸣器程序的制作与运行	65
4.4.1	程序概要	65
4.4.2	程序的制作与运行.....	67
4.4.3	改变蜂鸣器的音色.....	68
5	利用中断来控制 LED.....	69
5.1	“中断”的概念	69
5.2	利用“中断”来检测 SW 的状态的方法	70
5.3	通过 SW 控制 LED 的程序（中断法）	71
5.3.1	程序概要	71
5.3.2	程序的制作与执行.....	72
6	利用 timer（定时器）来使 LED 闪烁.....	75
6.1	什么叫 timer（定时器）	75
6.2	通过 Timer 中断控制 LED 闪烁的程序	76
6.2.1	程序概要	76
6.2.2	程序的制作与运行.....	78
7	如何使用 A/D（模/数）转换器	81
7.1	模拟信号与数字信号	81
7.1.1	A/D 转换器的概要	82
7.1.2	滑动变阻器	83
7.2	制作一个表示电压数值的程序.....	83
7.2.1	程序概要	83
7.2.2	程序的制作与执行.....	87
8	如何使用温度传感器	89
8.1	关于温度传感器	89
8.2	温度传感器的使用方法.....	90
8.3	制作一个表示温度的程序	91
8.3.1	程序概要	91
8.3.2	程序的制作与执行.....	94
A	附录 A（程序制作流程）	96
B	附录 B（寄存器的写入 / 读出方法）	104
C	附录 C（头文件包含路径的设定方法）	105

前言

首先，非常感谢您购买本公司的 **Starter-kit** 产品。

本套工具是专对单片机初学者设计的。什么是单片机？怎样才能运行它？单片机用在什么领域？等等一系列的问题都将在此为您做一一解答。我想，即使没有接触过单片机的人，通过此套设备也会对单片机有所了解从而产生进一步的学习兴趣——这便是我们设计此套工具的初衷！

本套工具内包含了一款内嵌 **Flash** 存储器的单片机（可供用户随意改写源程序），以及相应的开发软件工具，如果您已经有简单的 **C** 语言基础，则可以马上自行改写源程序并实际在单片机上运行它！此外，对于没有任何软件语言基础的用户，我们也为您做了非常易懂的介绍资料，例如如何用单片机控制 **LED** 闪烁，如何用单片机控制蜂鸣器……通过这些介绍资料再加上一些简单的 **C** 语言参考书，您就可以借由单片机来学习软件设计了。

在编写此用户手册时我们询问了许多本公司内的新进员工，在他们的建议下，争取将此手册编写得通俗易懂。因此，不管您有没有单片机的相关经验，相信你都能很快地进入单片机的世界并领略其中的风采！

本套工具适合作为大专院校单片机及其嵌入式操作系统教学、实验和开发的参考教学资料，也可以作为单片机开发工程师的教育学习工具。

1 Starter-kit 的安装方法

在安装本套工具前，请先按照表格 1-1 确认包装内的商品是否完整。

在实际进行连接前请先安装必要的软件。

本商品有两个版本，一种是包含 CD 的，另一种是不含 CD 的。如果您购买的是不含 CD 的版本，请下载必要的软件包进行安装。

表格 1-1 产品内容

	品名	数量	样式	备注
1	主板	1	搭载富士通公司单片机 F ² MC-16LX 系列 MB90F387S 的评测主板	图 1-1 主板外观
2	USB 接线	1	USB	附属品
3	CD-ROM	(1)	SOFTUNE 综合开发工具, ACCEMIC MDE USB 驱动 CD-ROM(相关的使用说明均包含在内) 用户手册: jouet_bleu_start_kit_manual_C.pdf USB 驱动安装向导: USB_driver_installation_manual_C.pdf MB90385 系列 Hardware Manual: MB90385_HM_E.pdf (英语版) MB90385 系列 Datasheet: MB90385_DS_E.pdf (英语版) SOFTUNE 相关使用说明: SOFTNE\Manual\文件夹内 (英语版)	只针对 CD-ROM 版用户 无 CD 版用户可从网上下载 (内容完全一致)
4	PC	1	安装有 WindowsXP/Me/2000/98 的 PC 机。 支持 USB2.0 的通信接口。 剩余硬盘空间约需 200Mbyte。	用户需自行准备。

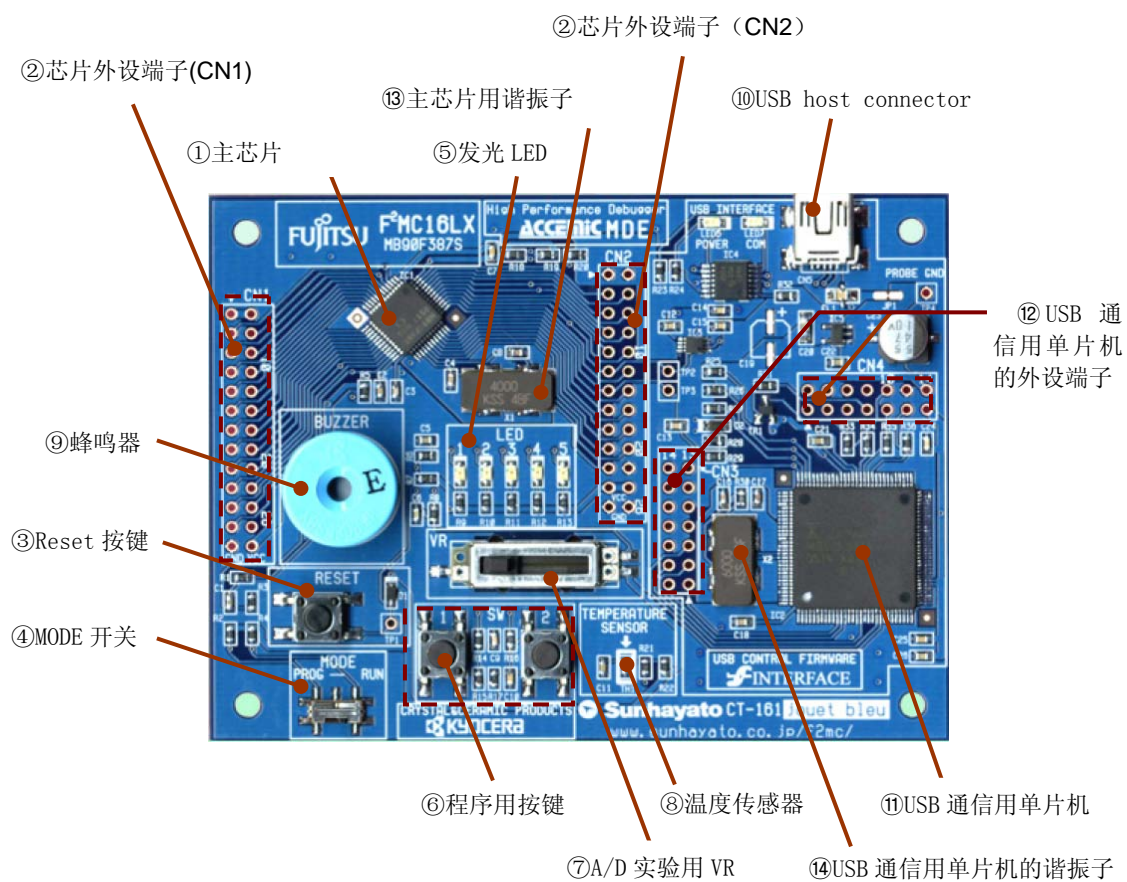


图 1-1 主板外观

图 1-2 主板电路图显示了此主板的电路连接。

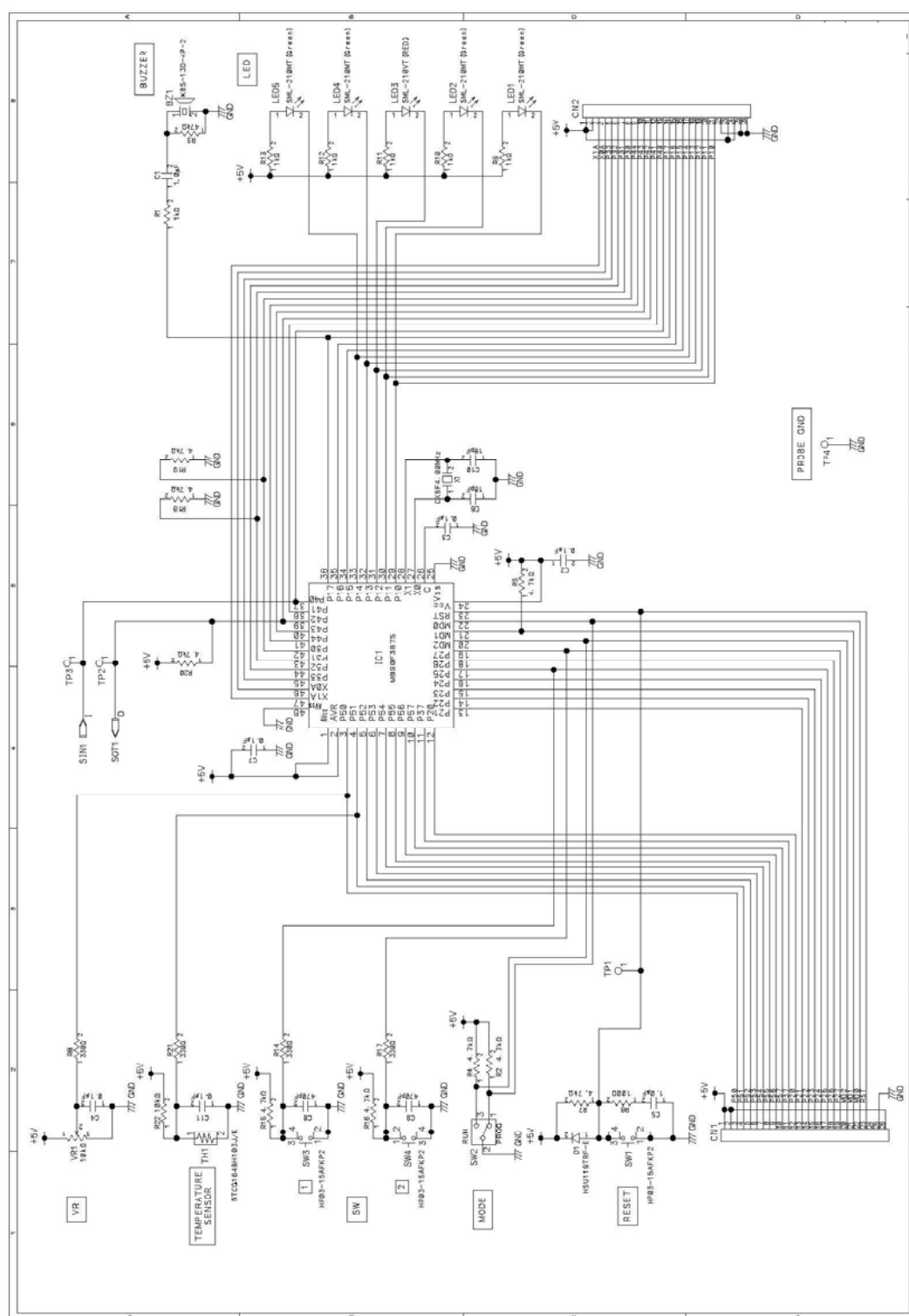


图 1-2 主板电路图

此评测主板所含部件一览表如下：

NO.	名称	规格	功能
①	主芯片	MB90F387S	为本评测板的主芯片（MB90F387S）。
②	芯片外设端子	26PIN ×2	单片机的外设端子
③	Reset 按键	按压式	按下时会使单片机 Reset（重启）
④	MODE 开关	拨动式	用来切换单片机（MB90F387S）的运行模式。
⑤	发光 LED	LED ×5（红 1、绿 4）	连接在通用 I/O 端口上，用来显示运行状态
⑥	程序用按键	按压式×2	连接在通用 I/O 端口上，用来配合程序的运行
⑦	A/D 实验用 VR	VR（滑动变阻器）	与 A/D 转换器的输入端相连，用来改变输入电压。
⑧	温度传感器	thermistor	与 A/D 转换器的输入端相连，随温度变化改变输入电压。
⑨	蜂鸣器	他励式	京瓷公司的他励式压电蜂鸣器（KBS-13DB-4P-2），与 PPG timer 的输出端相连，用来发声。
⑩	USB host connector	MIN-B	用来连接主板与 PC 机通信的 USB 接口。
⑪	USB 通信用单片机	MB90F334A	用来控制主芯片（MB90F387S）与 PC 机通信的 USB 通信控制单片机。 搭载有 Interface 公司开发的 USB To RS232C 的转换平台。
⑫	USB 通信用单片机的 外设端子	14PIN ×2	USB 通信用单片机的外设端子。
⑬	主芯片用谐振子	CX-5FD(4MHz)	京瓷公司的水晶谐振子（CX-5FD）。
⑭	USB 通信用单片机的 谐振子	CX-5FD(6MHz)	京瓷公司的水晶谐振子（CX-5FD）。

表格 1-2 主板所含部件一览

系统构成图见图 1-3

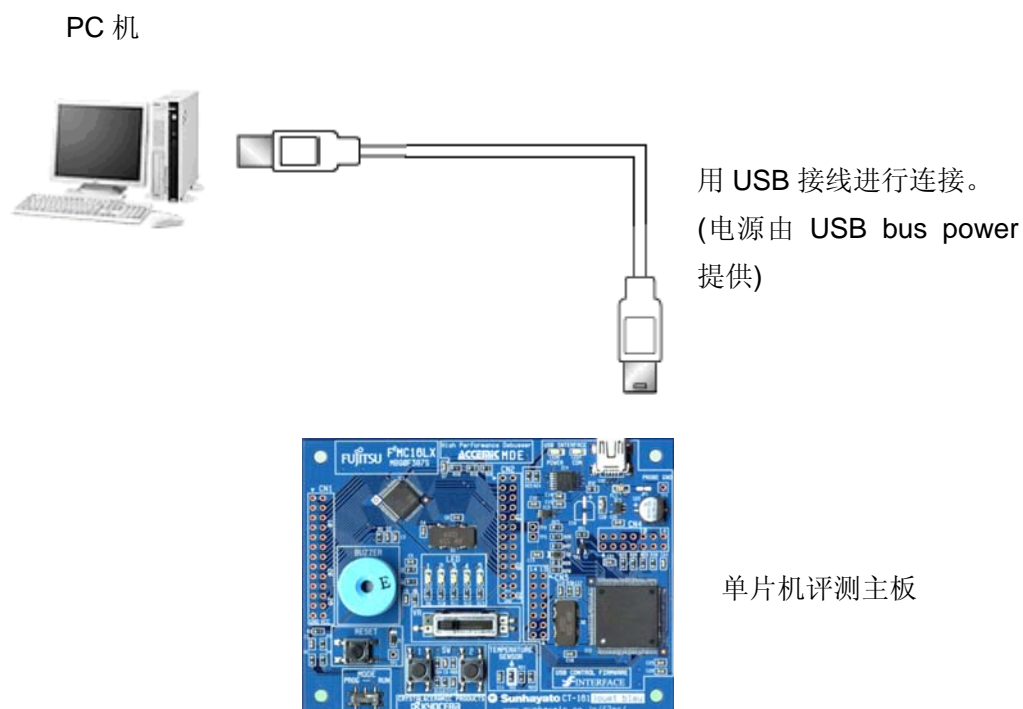


图 1-3 系统构成

在连接 PC 机与评测主板前，请先安装相应的软件（以便识别设备）

用 USB 接线将 PC 机与单片机评测板相连接。

单片机评测板的电源由 USB bus power 提供（用户无需另接电源）。

我们强烈建议您将单片机评测主板与 PC 机的 USB 接口直接连接（请不要使用 USB HUB 等设备）

评测主板上所使用的单片机 F2MC-16LX 系列 MB90F387S 的端口连接一览表如下

表格 1-3 F²MC-16LX 系列 MB90F387S 单片机的端口连接一览表

端口编号	端口符号	连接到的外部设备	所用原理	备注
3	P50/AN0	VR	模拟信号输入	电源电压分压比 0-100%
4	P51/AN1	TEMP.SENSOR	模拟信号输入	1/2VCC@25℃
17	P25/INT5	SW1	低电平有效	SW 按下时=L
19	P27/INT7	SW2	低电平有效	SW 按下时=L
20	MD2	MODE	—	—
21	MD1	PULL-UP	—	—
22	MD0	MODE	—	—
23	RST	RESET	低电平有效	端口输出 L 时=发光
29	P10/IN0	LED1	低电平有效	端口输出 L 时=发光
30	P11/IN1	LED2	低电平有效	端口输出 L 时=发光
31	P12/IN2	LED3	低电平有效	端口输出 L 时=发光
32	P13/IN3	LED4	低电平有效	端口输出 L 时=发光
33	P14/IN4	LED5	低电平有效	端口输出 L 时=发光
36	P17/PPG3	BUZZER	矩形波	初始电平 L, C couple, bias R
37	P40/SIN1	RS232C	—	—
39	P42/SOT1	RS232C	—	—
42	P30/SOT0	PULL-DOWN(50K Ω)	—	—
43	P31/SCK0	PULL-DOWN(50K Ω)	—	—

1.1 PC 机上的软件安装

要使本套工具正常工作必须先安装相应的软件。(请先安装软件后在连接主板与 PC 机)

安装步骤:

- 1、安装 USB 驱动（请参照另一个「USB 驱动安装向导」）
- 2、安装综合开发环境 SOFTUNE（限定版）(参照 1.1.2)
- 3、安装 ACCEMIC MDE demo version(参照 1.1.3)
- 4、连接 PC 机与评测版(参照 1.1.4)
- 5、启动并设定 SOFTUNE 相关选项(参照 1.1.5)
- 6、启动并设定 ACCEMIC MDE 的相关选项(参照 1.1.6)
- 7、关闭 ACCEMIC MDE(参照 1.1.7)
- 8、关闭 SOFTUNE(参照 1.1.8)

1.1.1 USB 驱动的安装

关于 USB 驱动的安装请参照另一份《USB 安装驱动向导》文件。

提示：先将此设备连接到电脑上，系统会自动提示找到新硬件并要求安装相应的驱动程序，然后您可以按照向导上的指示一步一步安装。

1.1.2 综合开发环境 SOFTUNE（限定版）的安装

关于 SOFTUNE（限定版）的说明。

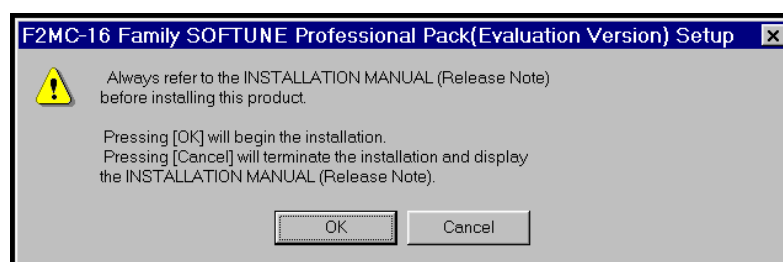
功能限定版 SOFTUNE 能够调试的程序大小上限为 32KB。

将 CD 放入 CD 驱动器，然后执行安装文件。如果购买的是无 CD 的版本，则请您先下载 F²MC-16LX Starter kit(jouet bleu)用的软件包，解压以后再执行安装文件。安装文件的路径为：

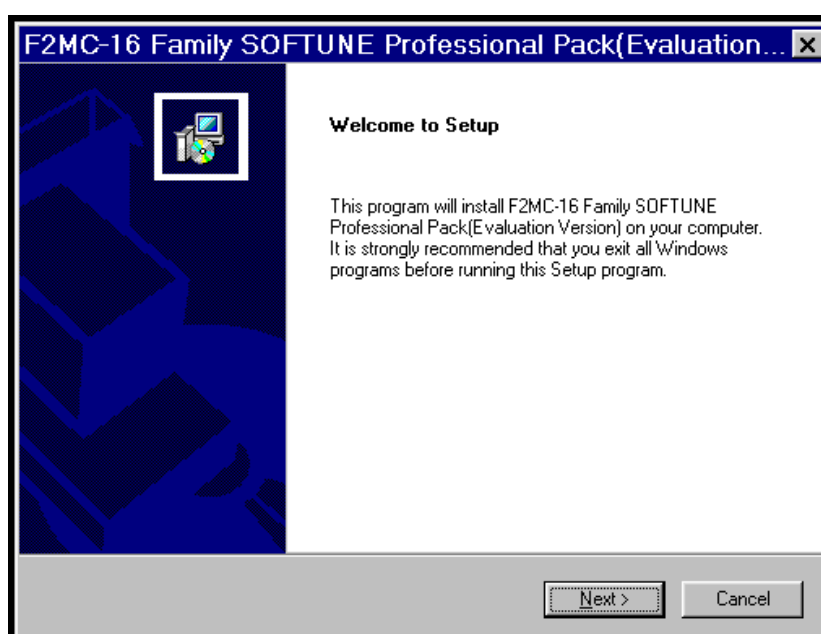
\\jouet_bleu\SOFTUNE 文件夹下名为「setup.exe」的文件，开始安装。



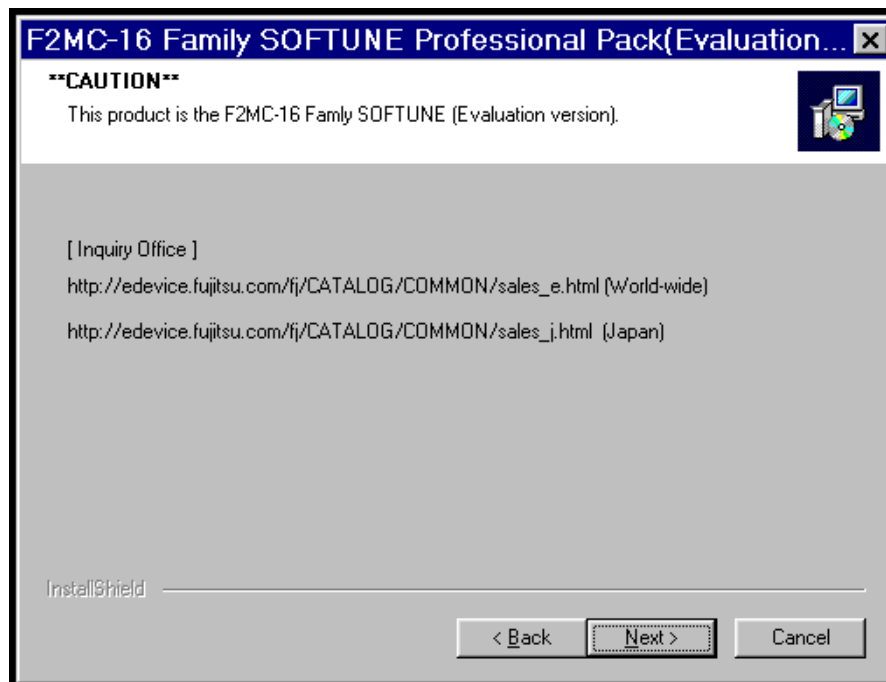
根据画面上的提示进行安装。（建议您不要更换默认安装路径）



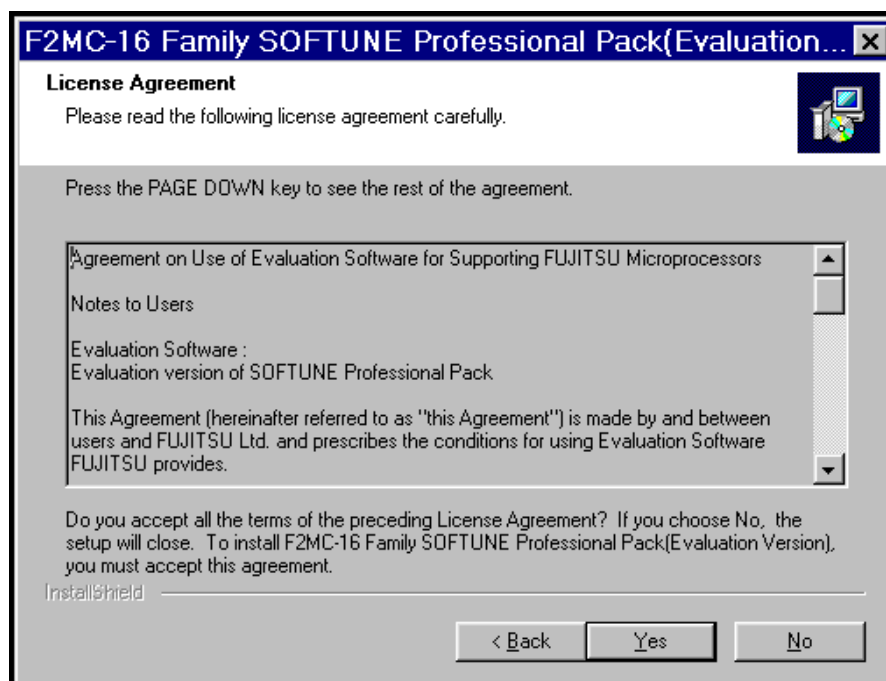
点击「OK」



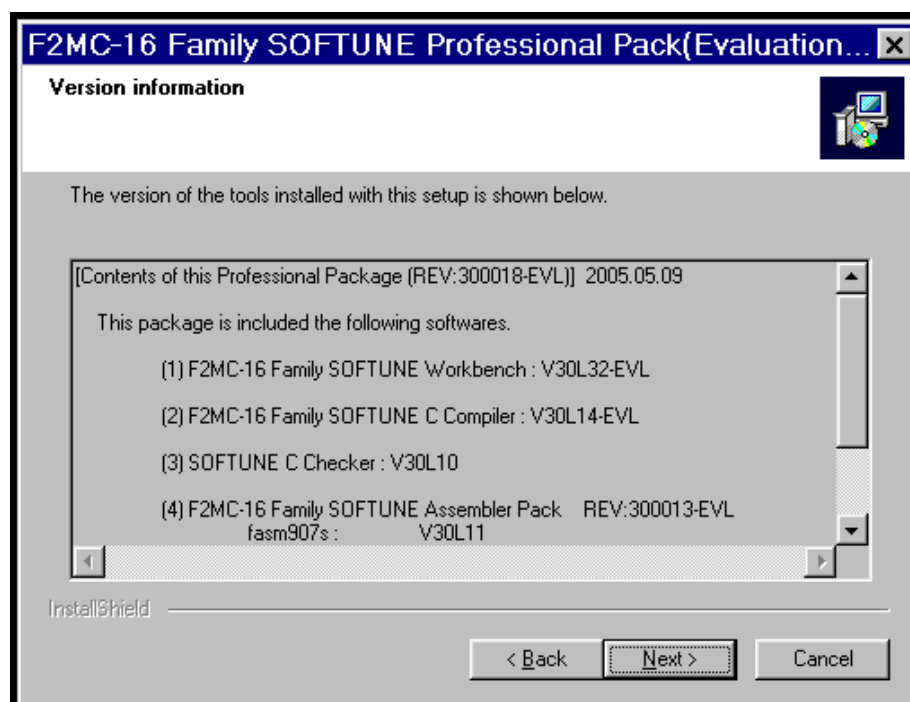
点击「Next (N)>」



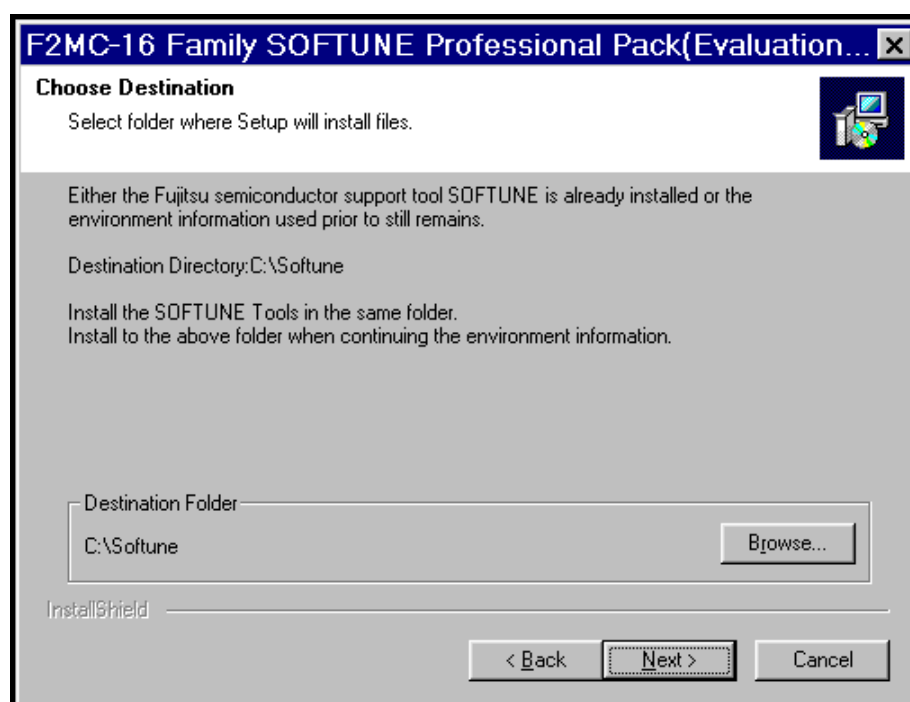
点击「Next (N)>」



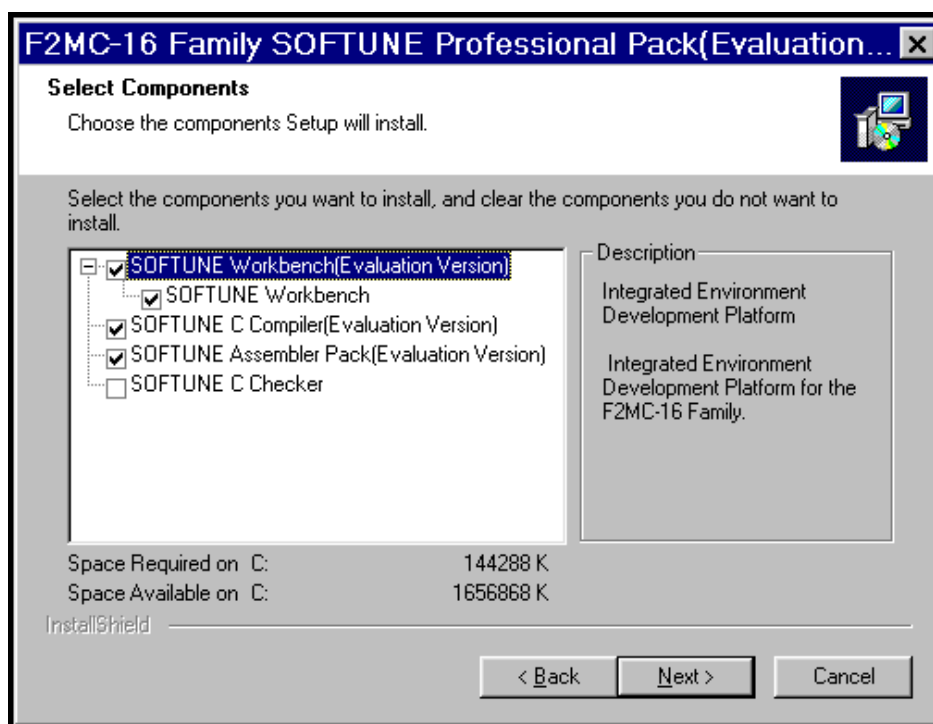
点击「Yes(Y)」



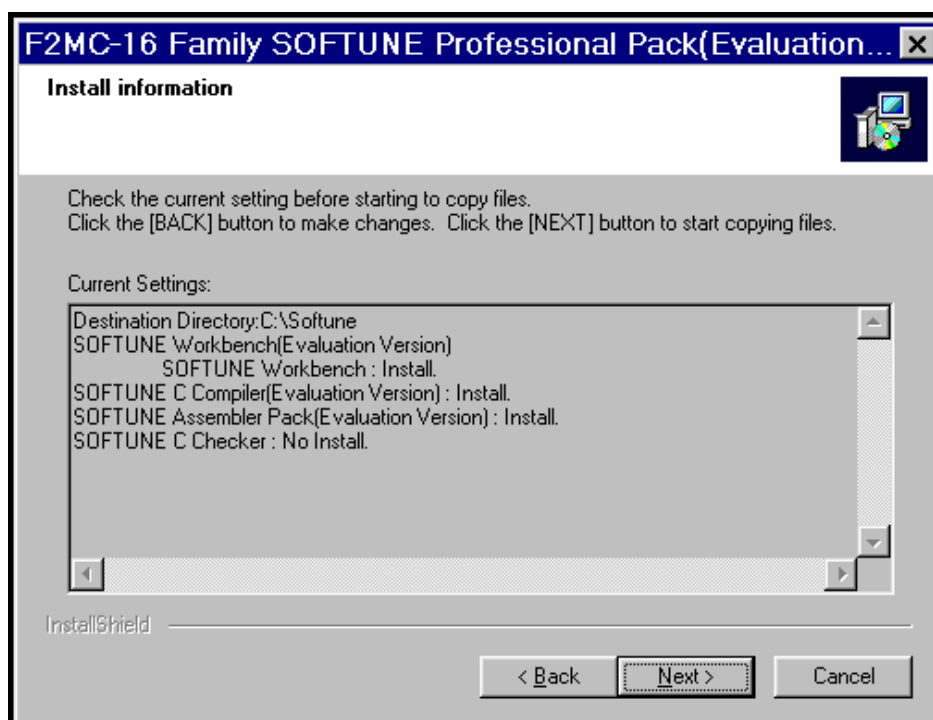
点击「Next (N)>」



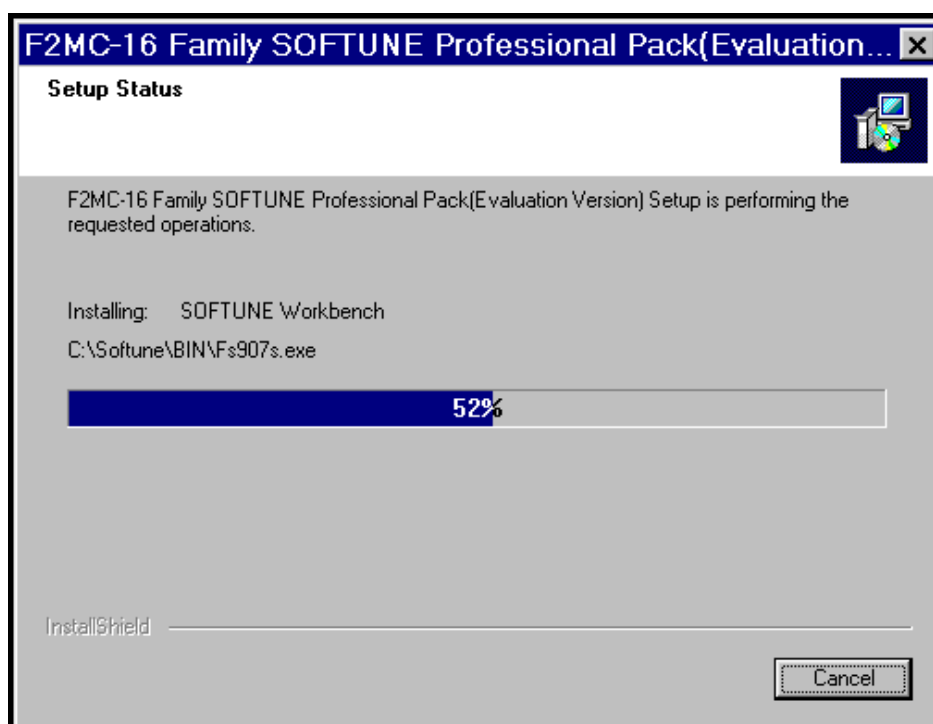
点击「Next (N)>」
(在此可更换安装路径)



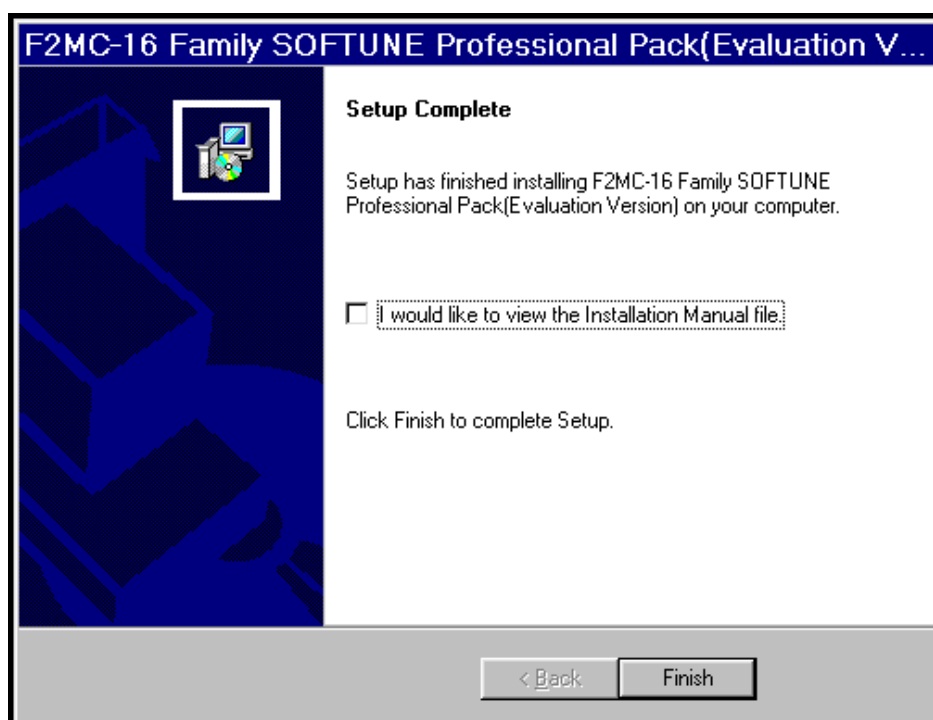
点击「Next (N)>」



点击「Next (N)>」



安装正在执行



点击「Finish」

至此，SOFTUNE（限定版）已安装完成。

接下去，请安装 ACCEMIC MDE demo version。

1.1.3 ACCEMIC MDE demo version(trial 版)的安装

关于 ACCEMIC MDE demo version (trial 版) 的说明

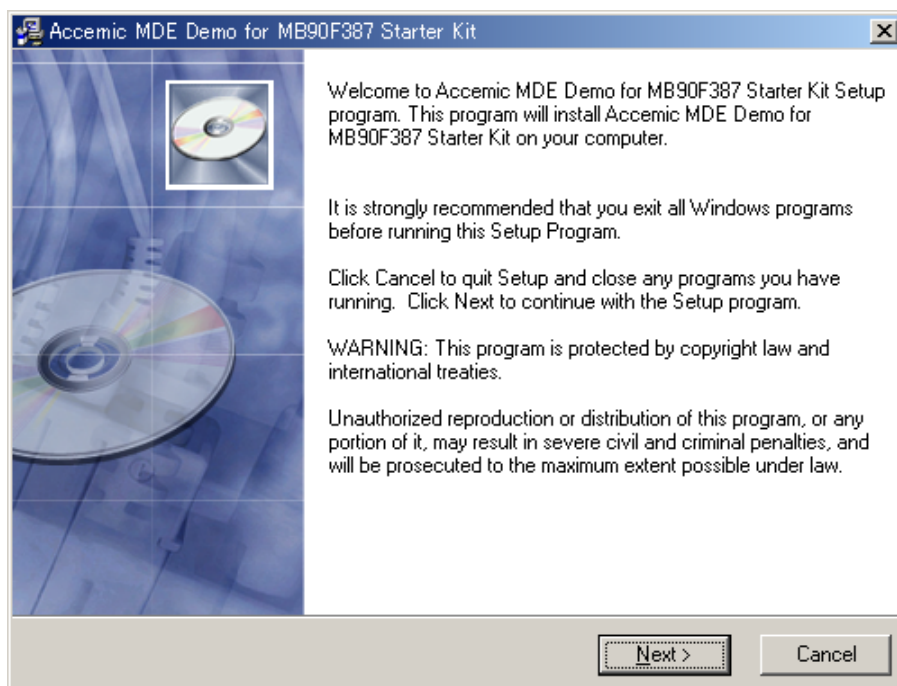
trial 版 ACCEMIC MDE 可以调试的程序大小上限为 12KB。12KB 以上的程序无法在 trial 版上运行。而且，可调试的最大程序数为 12 个。

将 CD 放入 CD 驱动器，然后执行安装文件。如果购买的是无 CD 的版本，则请您先下载 F²MC-16LX Starter kit (jouet bleu)用的软件包，解压以后再执行安装文件。安装文件的路径为：

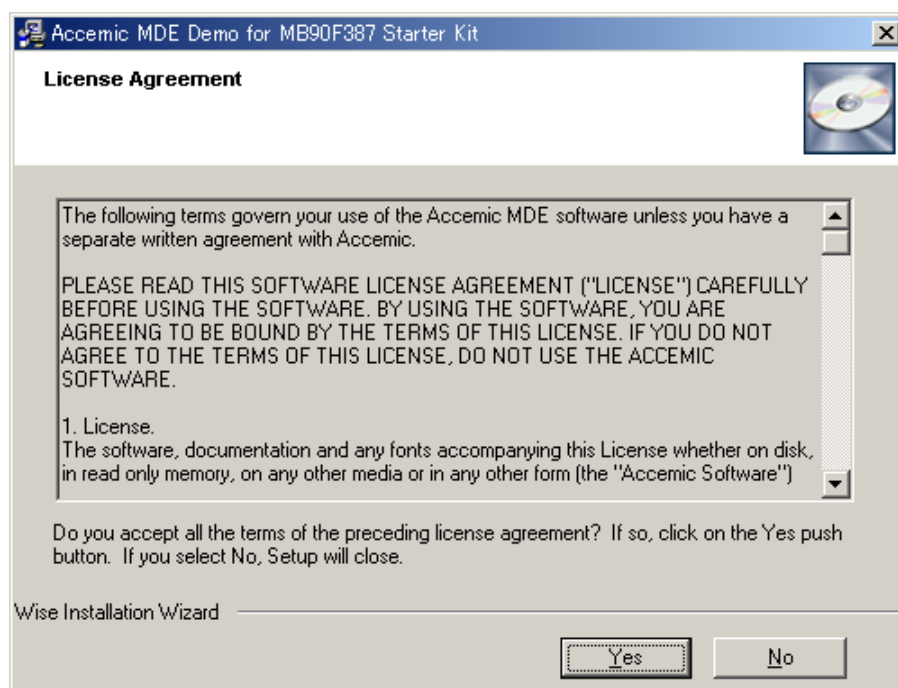
CD 或解压路径\jouet_bleu\MDE_16LX_DEMO_V22ST_STARTERKIT 下文件名为「MDE_16LX_DEMO_V22ST_STARTERKIT.exe」的文件，双击它开始安装。



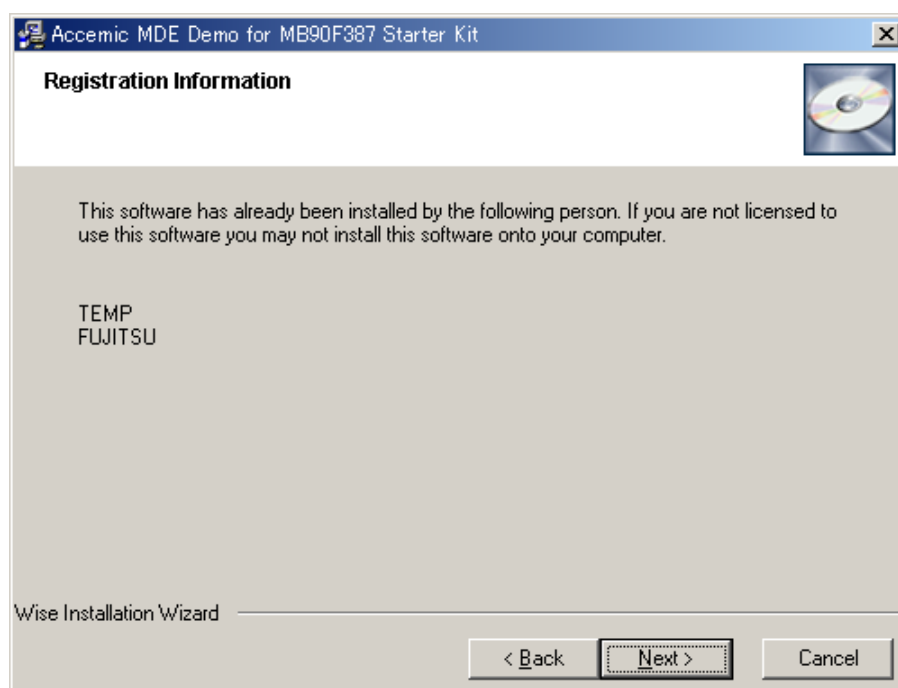
根据画面上的提示进行安装。（建议您不要更换默认安装路径）



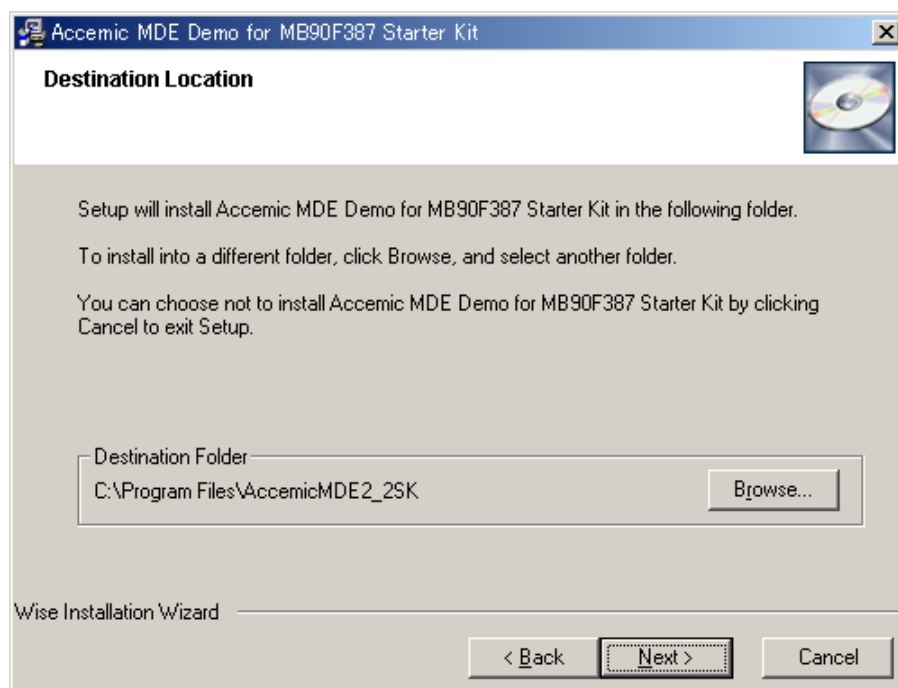
点击「Next (N)>」



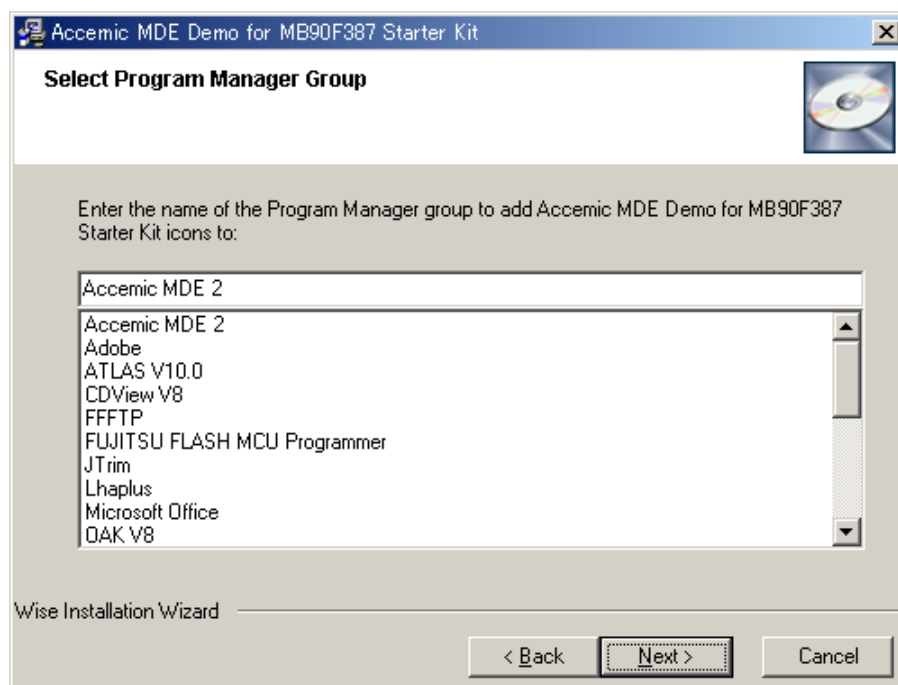
点击「Yes(Y)」



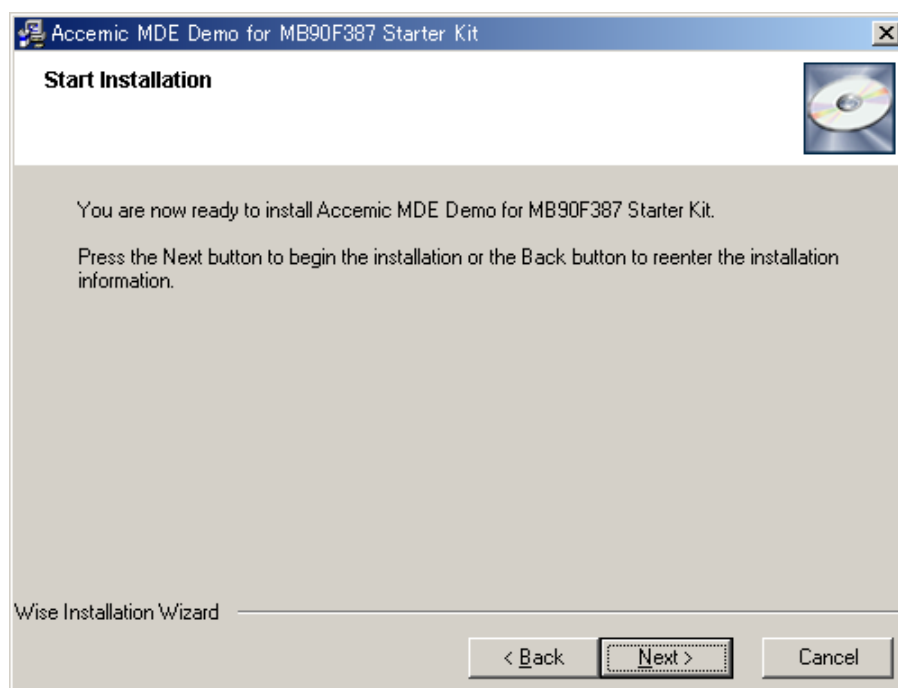
点击「Next >」



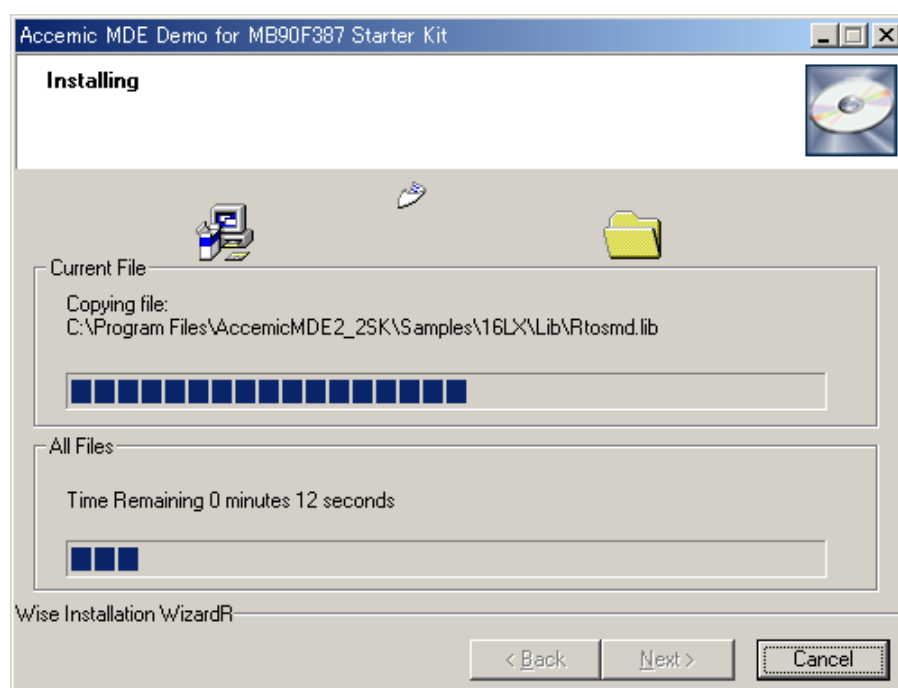
点击「Next >」
(在此可更改安装路径)



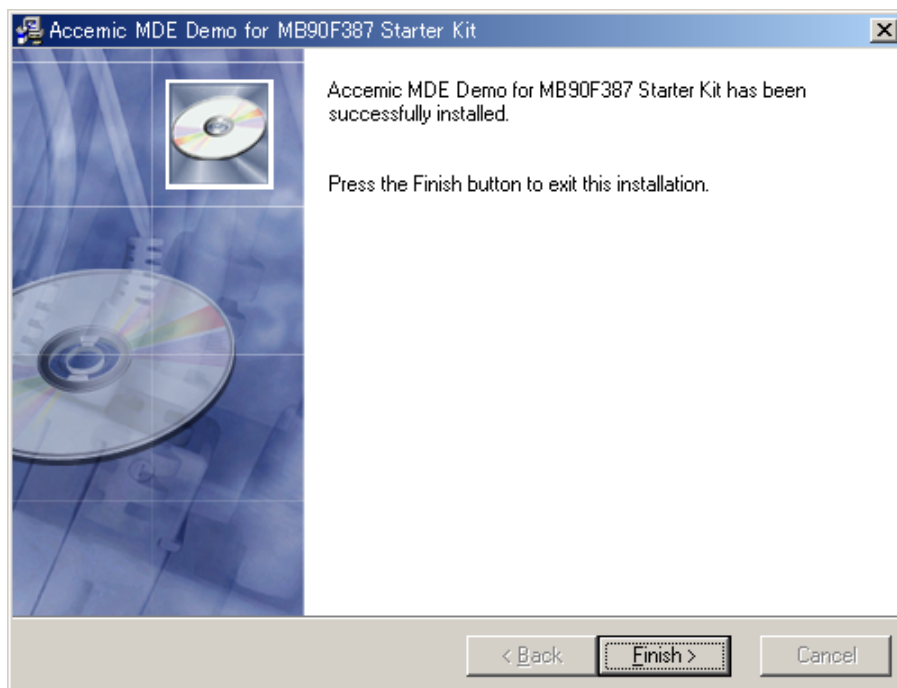
点击「Next >」



点击「Next >」



安装正在执行



安装完成后，点击「Finish >」

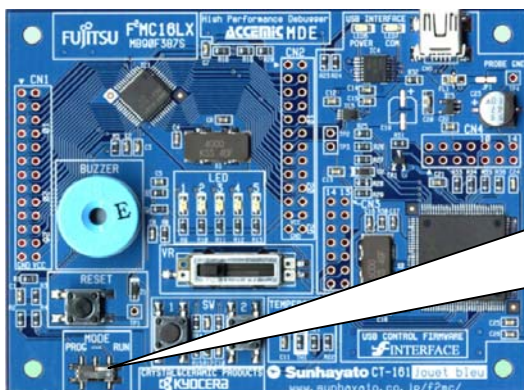
至此，ACCEMIC MDE demo version 已安装完成

接下去，将进行评测版的设定以及与 PC 机的连接。

1.1.4 评测板的设定以及与 PC 机的连接

将 SOFTUNE 与 ACCEMIC MDE 安装完成以后先要设定评测板上的 MODE 键值，然后再进行与 PC 机的连接。

请先将评测板上的「MODE」键设定到「PROG」一侧。



请先将 MODE 键设定到「**PROG**」一侧！

说明：

MODE 键	运行模式
PROG	FLASH memory 串行写入模式 →向单片机内部写入程序时所用的模式。
RUN	单一芯片运行模式 →写入程序后实际运行时的模式。

请先确认 MODE 被设定在「PROG」一侧，以备程序写入。

接着进行连接。

请用商品包装内附带的 USB 连接线将评测板的 USB 口与 PC 机的 USB 口直接连接。

注意：请不要使用任何 USB HUB 设备，直接将评测板与 PC 机进行连接。

连接至 PC 机的 USB 端口。

PC 机上的 USB 端口位置请参照 PC 机的说明书。

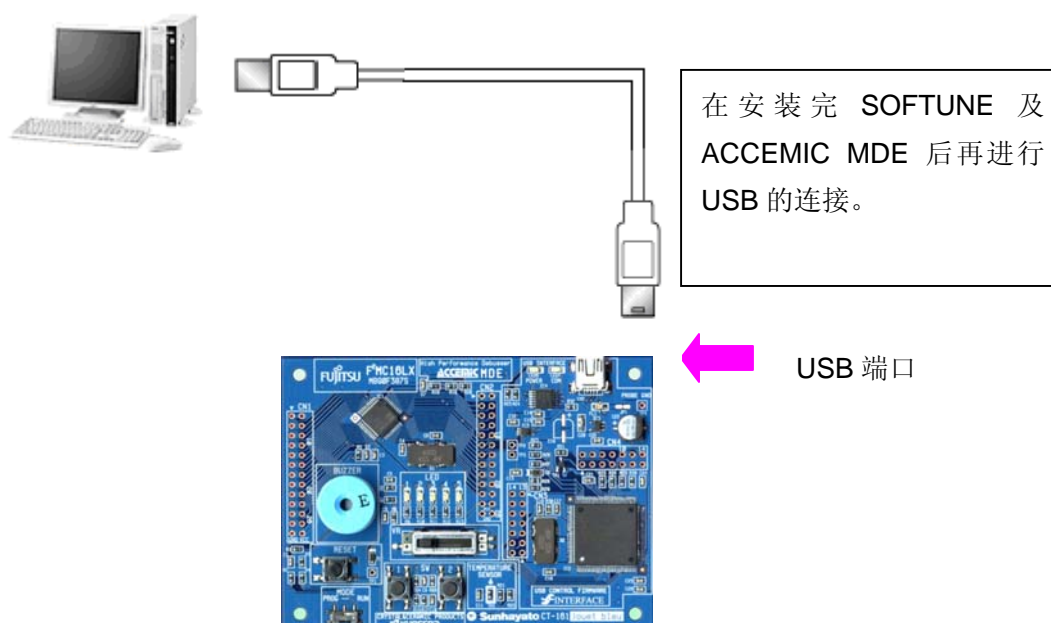
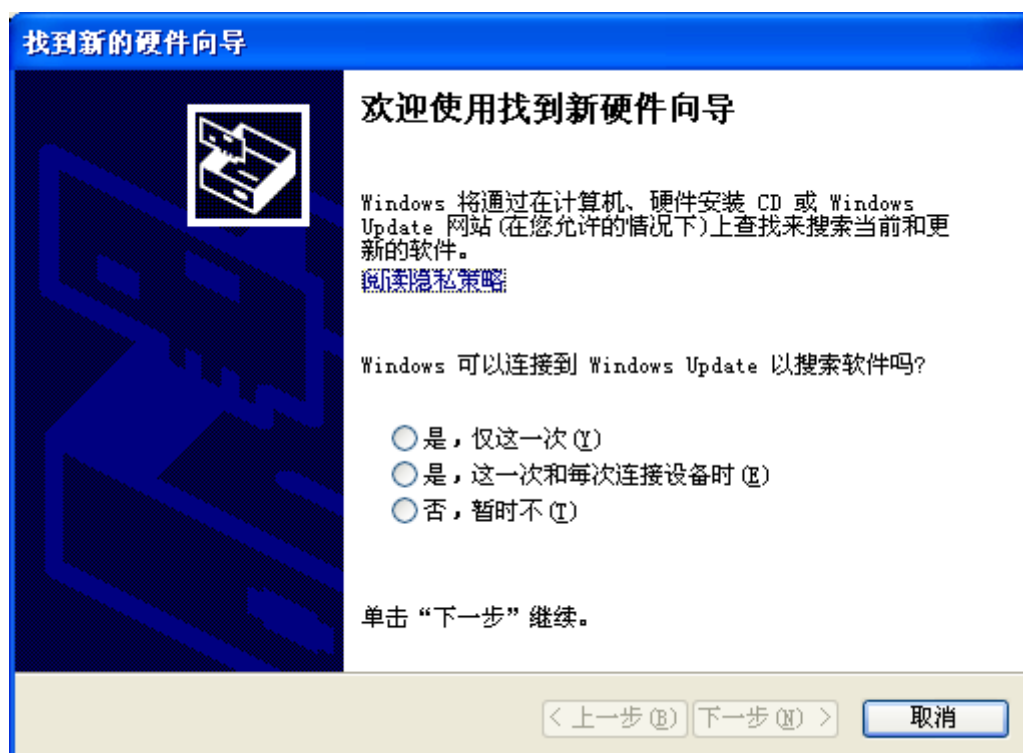


图 1-4 评测板与 PC 机的连接图

评测板的电源也是由 USB 提供的，所以无需外接电源。(USB bus power)

连接好以后会弹出发现新硬件的提示



此时请安装 USB 驱动，安装好以后再进行下一步操作，具体安装方法请参照《USB 驱动安装向导》

1.1.5 SOFTUNE 的设定与启动

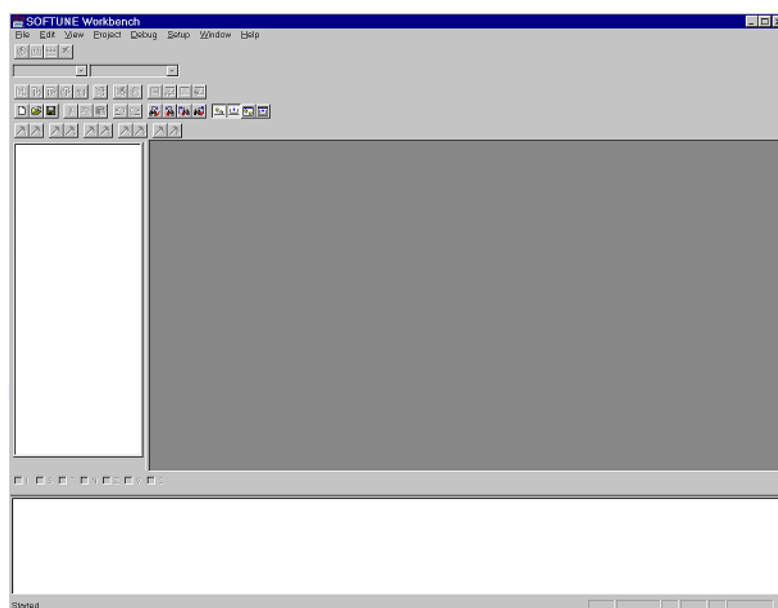
请先将样例程序文件解压并复制到硬盘上。

在 CD-ROM 或者下载的文件中您可以找到以「Fujitsu_Starter_kit.zip」为文件名的文件。

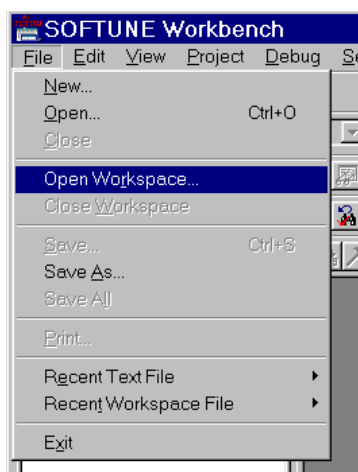
将「Fujitsu_Starter_kit.zip」文件解压到任意一个您喜欢的目录下，解压后的文件由若干个文件夹及一个 Start_kit.wsp 文件组成。

- 启动 SOFTUNE（限定版）。

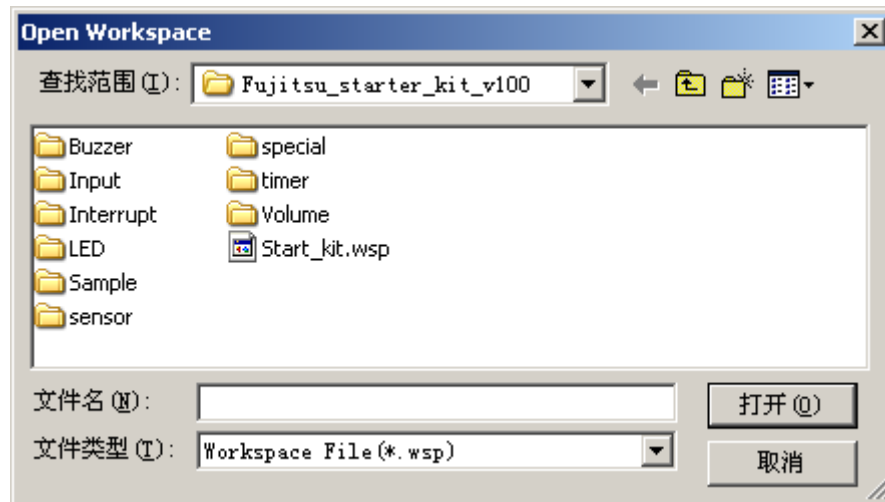
点击 Windows 的「开始」－「程序(P)」－「Softune V3」－「FFMC-16 Family Softune Workbench(trial version)」 便可启动 SOFTUNE。



- 接下去，先打开样例程序中的 work space 文件。（*.wsp 文件）
具体打开方法如下：



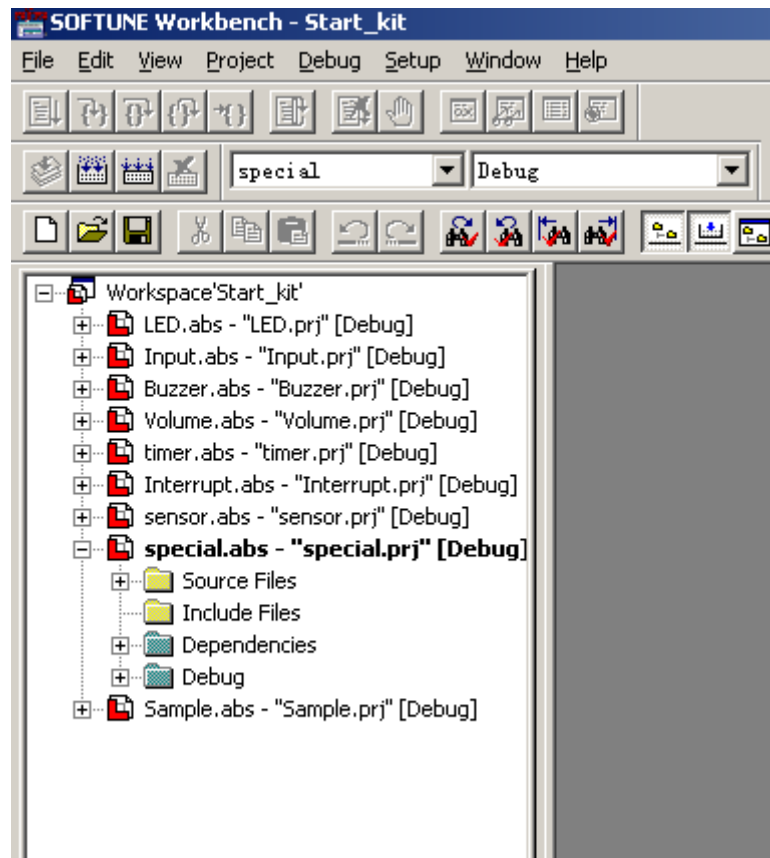
点击「File(F)」菜单的「Open Workspace(R)...」



在弹出的对话框中选择刚才样例程序解压的目录，然后打开 Workspace 文件 (Start_kit.wsp) 注意：打开 Workspace 文件 Start_kit.wsp 后还不能直接执行样例程序。

打开\Fujitsu_Starter_kit 文件夹中的「Start_kit.wsp」文件。

在 SOFTUNE 左侧的 Project window 里可以看到被打开的 workspace 文件。



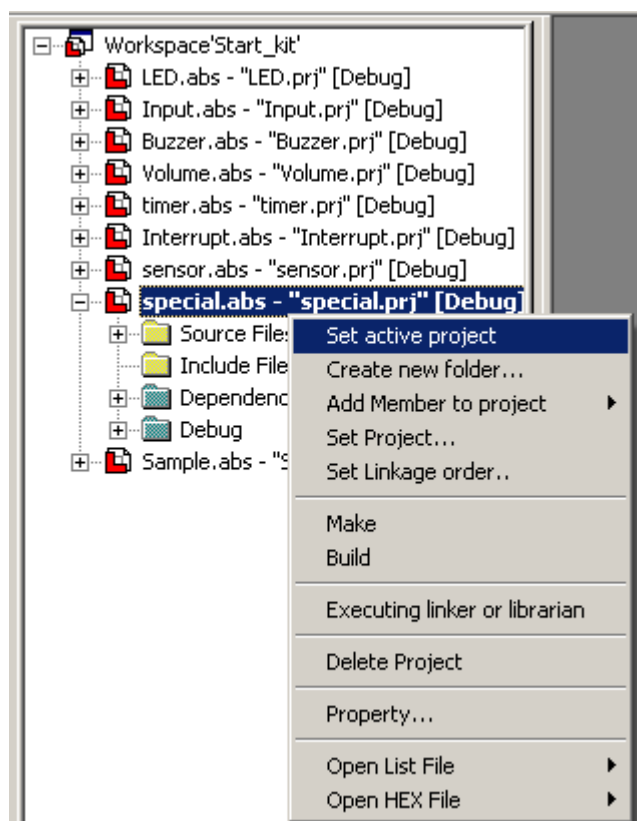
- Active project 的设定（设定当前 project）

在一个 workspace 文件里可以建立多个 project，如上图。

在样例程序中我们已经事先建立了「LED.prj」，「Input.prj」，「Buzzer.prj」等 project，以供用户直接调用。

而程序的 Debug（调试）是以 Project 为单位的，故用户必须自行设定当前的「Active Project」。

具体设定方法如下：



右击需要 debug（调试）的 project。

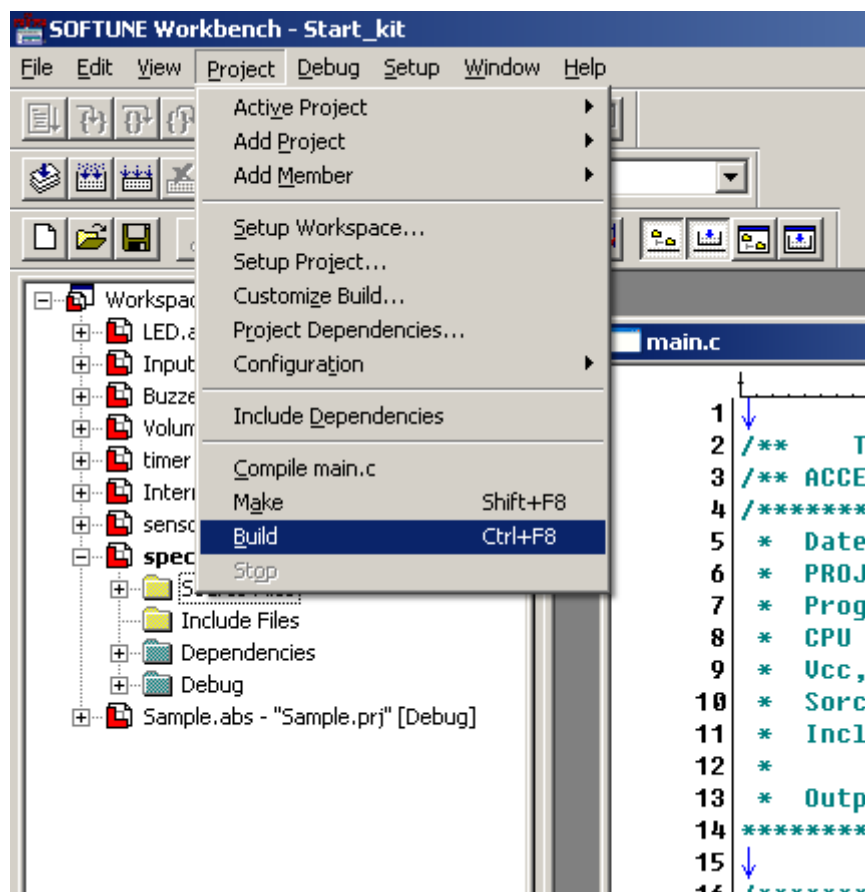
在弹出的子菜单上选择「Set active project」。您可以看到 Project 的名字变成了粗体显示，这样便可以对其进行 debug（调试）了。

注意：由于 Softune 不支持 Flash 芯片的写入，所以这里我们不能直接用 Softune 进行 Debug（调试），而需要用一个叫 AccemicMDE 的软件对 Flash 芯片进行 Debug（调试）。

1.1.6 ACCEMIC MDE 的设定与启动

您可以利用刚才在 SOFTUNE 里打开的 Workspace 文件「Start_kit.wsp」来启动 ACCEMIC MDE。

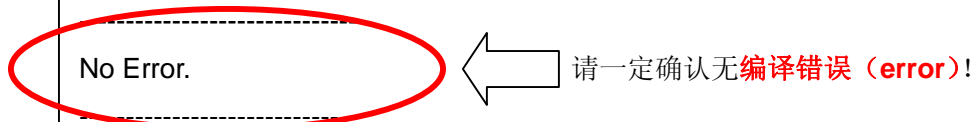
让我们先对「special.prj」进行 debug（调试），具体操作方法如下：请先在 SOFTUNE 里确认当前的 active project 为「special.prj」。然后在工具菜单里选择「Project(P)」—「Build(B)」，对当前 Project 进行编译。（主要是为了产生 ABS 文件，也就是机器码文件以供写入 Flash 芯片）



当您执行了 **Build** 以后，在 **SOFTUNE** 的输出窗口里（程序界面的下半部分）会输出如下的提示信息：

```

Now building...
-----Configuration: LED.prj - Debug-----
start905s.asm
_FFMC16.C
main.c
.....
monitor16LX.asm
Now linking...
*** I0312L: Total warning message of S.C.F check : 0
D:\jouet_bleu\Fujitsu_starter_kit_v100\special\Debug\ABS\special.abs
-----
No Error.
  
```



请一定确认无**编译错误 (error)**!

如果没有错误，**Accemic MDE** 将自动启动。

说明：在编译时如果发生以下错误，则请您从 **1.1.2** 节开始重新做起，注意请不要更改软件的默认安装路径！

Now building...

-----Configuration: LED.prj - Debug-----

start905s.asm

_FFMC16.C

*** c:\start_kit\start_kit_accemic\start_kit_accemic\led\ffmc16.c(9) E4038P: #include:
cannot find file "_ffmc16.h"

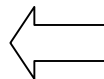
main.c。

.....

*** D:\jouet_bleu\fujitsu_starter_kit_v100\special\main.c(21) E4038P: #include: cannot find
file "monitor.h"

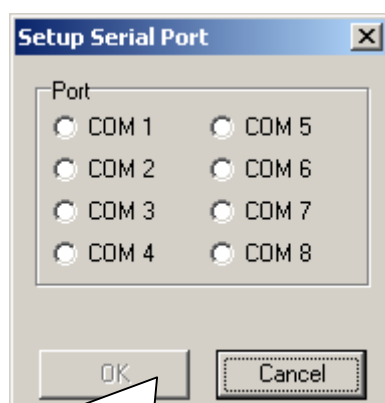
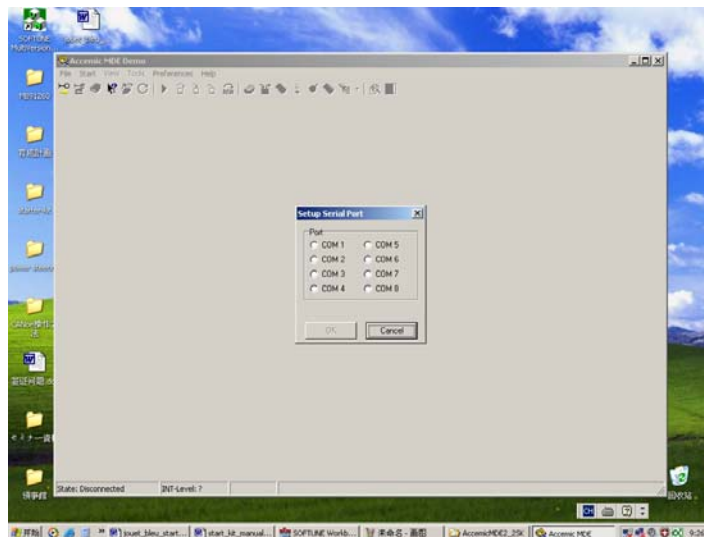
monitor16LX.asm

Error detected.

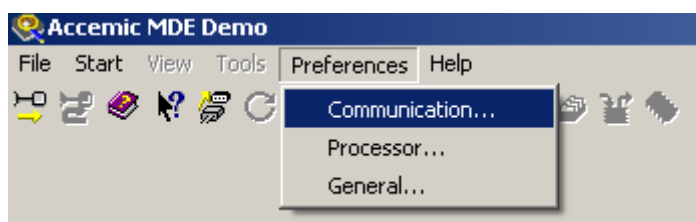


发生错误!

如果编译正常完成，则 ACCEMIC MDE 将自动启动。首先弹出 COM 端口选择画面：



如果没有显示这个画面，或者想手动更改 COM 端口号，请自行选择「Preferences」- 「Communication...」。



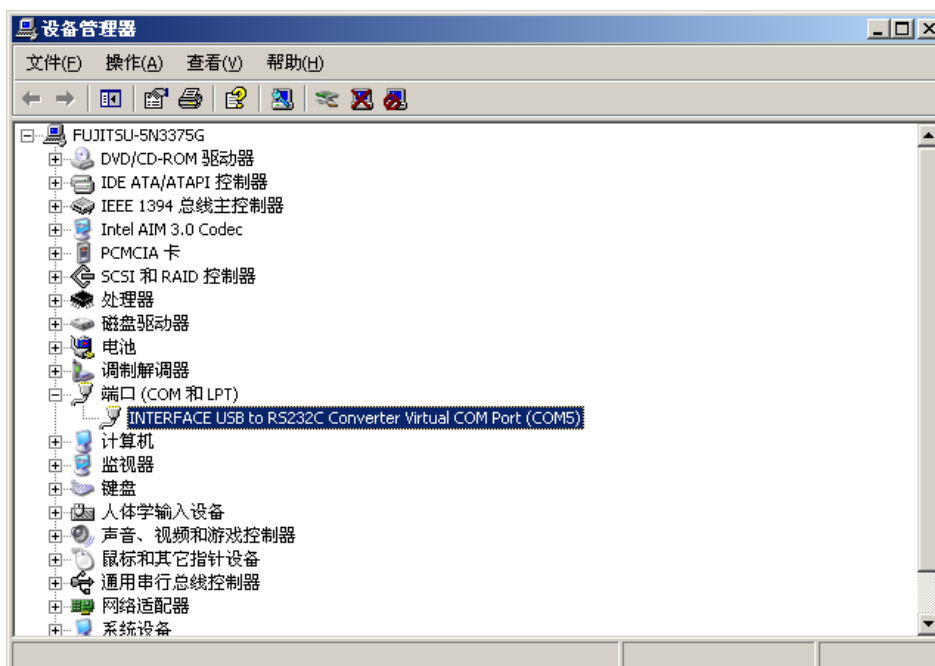
在这里选择你 PC 机分配给评测板的 COM 端口编号（提示：要知道你 PC 机分配给评测板的 COM 编号，请先安装好 USB 驱动，然后将评测板与 PC 机连接，接着用以下方法进行确认）

【COM 编号的确认】

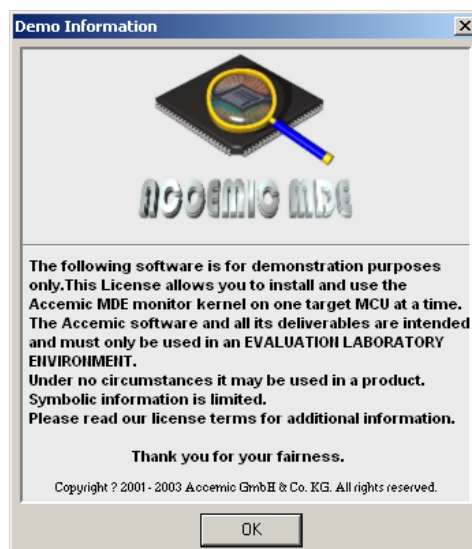
右击「我的电脑」，然后选择「属性」→「硬件」→「设备管理器」



接着，在「端口(COM / LPT)」(这个选项只有在插入 USB 线以后才会出现，在拔除设备的情况下将看不到这个选项)里选择「INTERFACE USB To RS232C Converter Virtual COM Port (COM5)」。注意：这里的(COMx)就是你的 PC 分配给评测板的端口号，根据不同的 PC 环境，所分配的编号可能不同。



在完成了端口号的设置后，将弹出一个欢迎画面（由于是 trial 版，所以要等 10s）。



点击「OK」(trial 版在点击「OK」前需要等待 10 秒。)

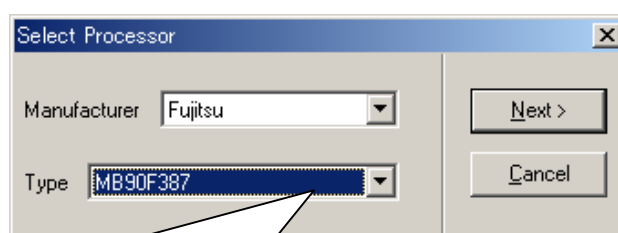
在点击「OK」以后会弹出一个 Select Processor 的对话框，它是用来设定芯片型号的。

在首次使用 ACCEMIC MDE 进行 debug（调试）时，必须先传送一个核心程序到您的单片机里，以后只要您不更换所用的单片机，则不用每次都传送此核心程序。具体操作方法如下：

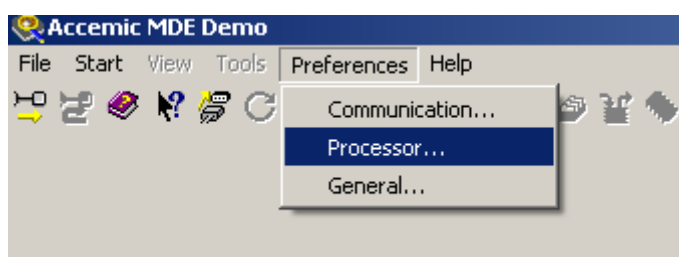
在自动弹出的型号设置对话框中设定用户使用的单片机型号：

Manufacturer: Fujitsu

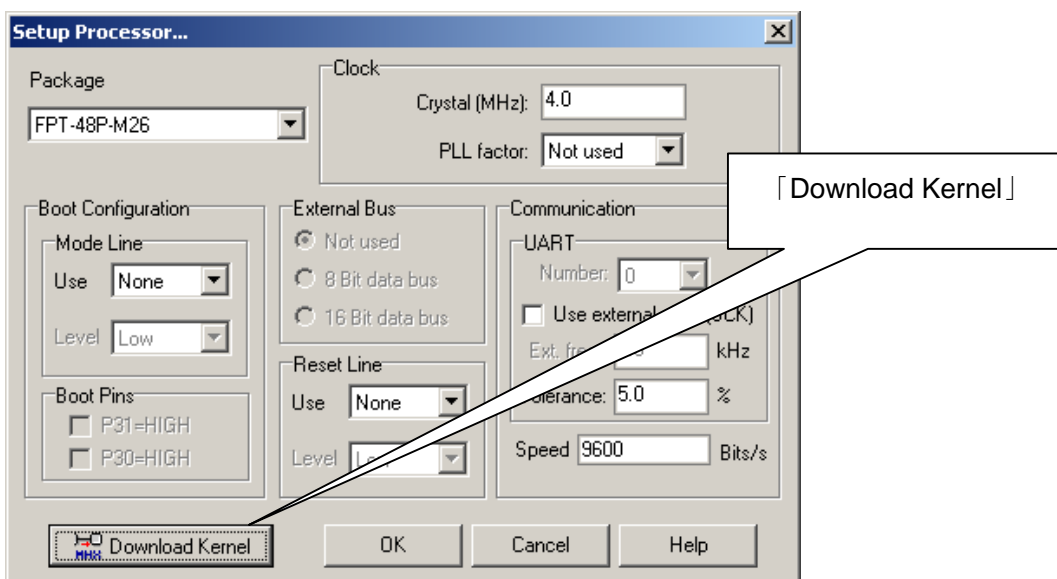
Type: MB90F387 然后点击「Next>」。



如果没有弹出这个对话框，或者想手动调节芯片型号，请选择「Preferences」-「Processor...」。



然后弹出以下设置对话框：



针对此款评测板上所用的单片机 MB90F387，请选则：

Package: **FPT-48P-M26** PLL: **Not Used**

其余选项均可保持默认值不做变动。

设定完成后，要进行通信核心程序的传送——点击「Download Kernel」

关键：在点击「**Download Kernel**」之后会弹出提示框，注意先不要点击上面的 **OK**，请按照以下步骤操作：**1**，请先将评测板上的 **MODE** 键设置为 **PROG**；**2**，然后按 **RESET** 键；**3**，之后再点击「**OK**」。(参照以下具体步骤)

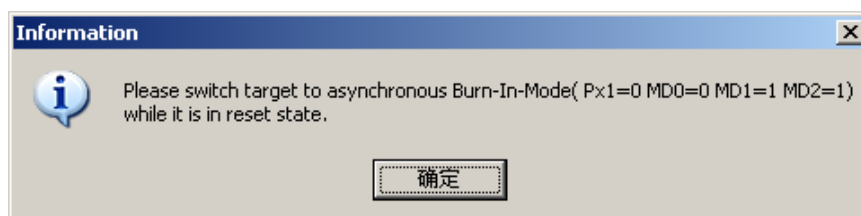
【注意事项】 PLL 的设定请不要使用“2”或者“4”，否则将不保证正常运行。

注意：按钮的操作顺序如果发生错误，评测板将无法顺利启动，此时请从 1.1.6 节开始重新操作。

具体步骤:

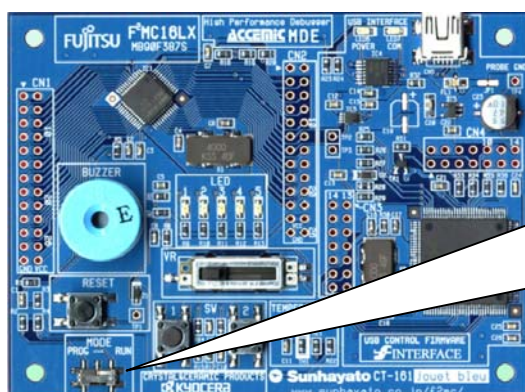
1、完成芯片型号的设置后点击「Download Kernel」

点击「Download Kernel」后会弹出提示框，提示您调整模式，在此请不要先点击直接点击确定！必须先调整运行模式



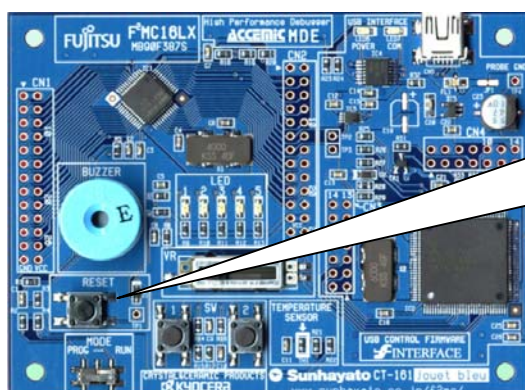
这里请先不要点击「OK」，您应该先按照刚才的方法更改运行模式，并按 **Reset**，然后再点击

2、将 MODE 键设定为「PROG」



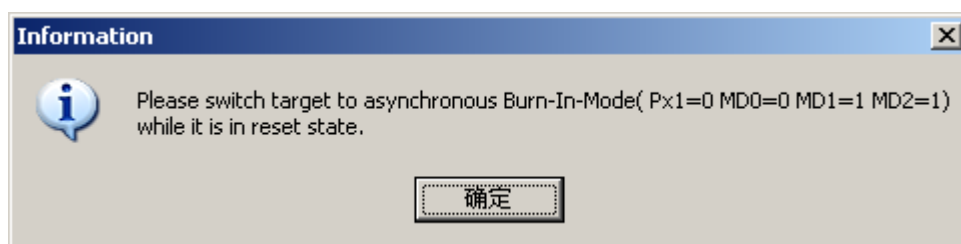
请先将 MODE 键设定到「**PROG**」一侧！

3、按下 RESET 键



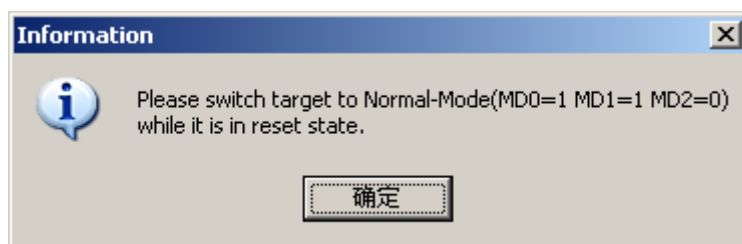
按 RESET 按钮。

4、最后点击「确定」



返回普通运行模式：

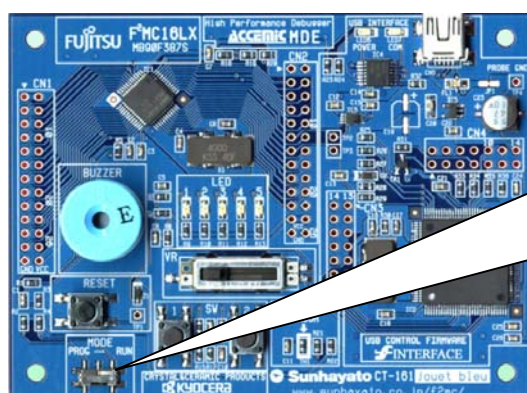
在完成了核心程序的传送后，会弹出如下信息框，要求您把运行模式恢复到 RUN 模式。



这里请先不要点击「OK」，您必须先更改运行模式到 RUN，并按 Reset，然后再点击

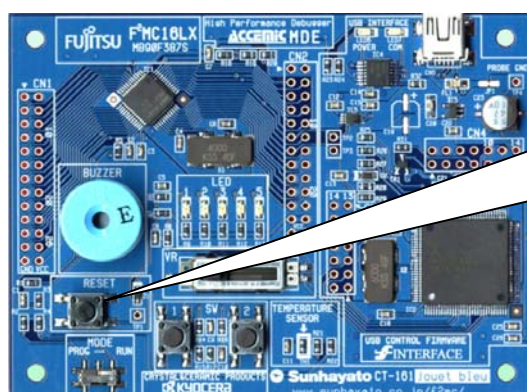
具体做法与刚才类似，只不过设定值不同：

1、先将 MODE 键设置为「RUN」



将 MODE 选择键设置为「RUN」模式

2、然后，按下 RESET 键使单片机 RESET



接着按下 RESET 键

3、以上两步完成后，再点击刚才的「确定」键

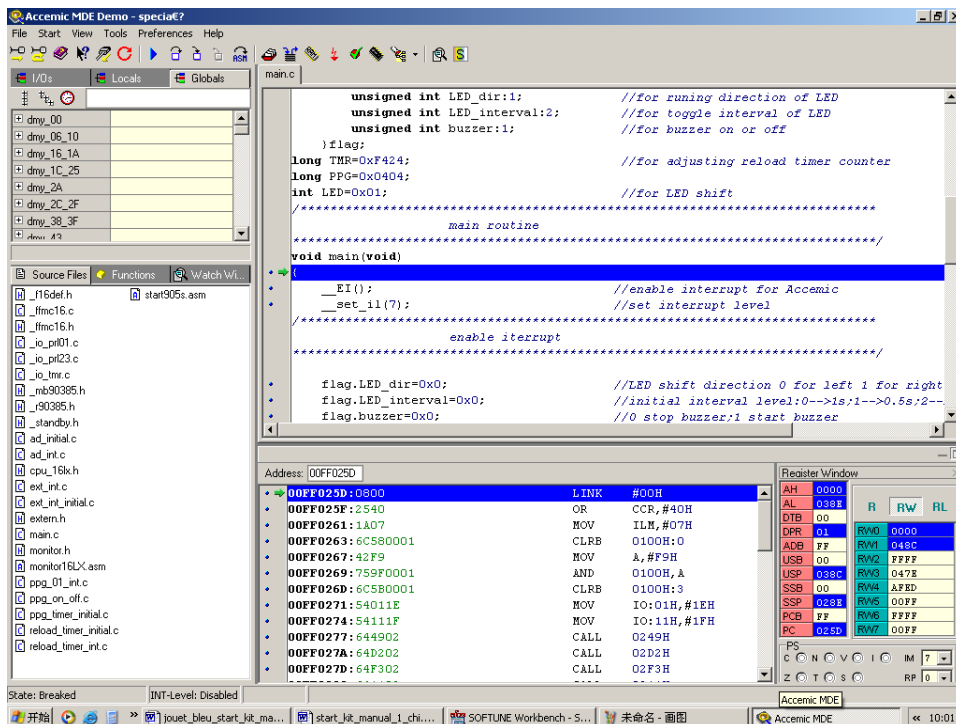


现在请点击「OK」

现在您可以放心地点击「OK」键了。

Accemic 启动好以后，我们便可以读入某个用户程序了，这里我们还是以 special.Prj 例程序为例。

说明：我们已经在 Softun 里把 Accemic 设定为自动读入 ABS 文件，所以当您点击上面的确定以后，Accemic 将自动读入文件，您可以看到以下画面（源文件已经被显示）：



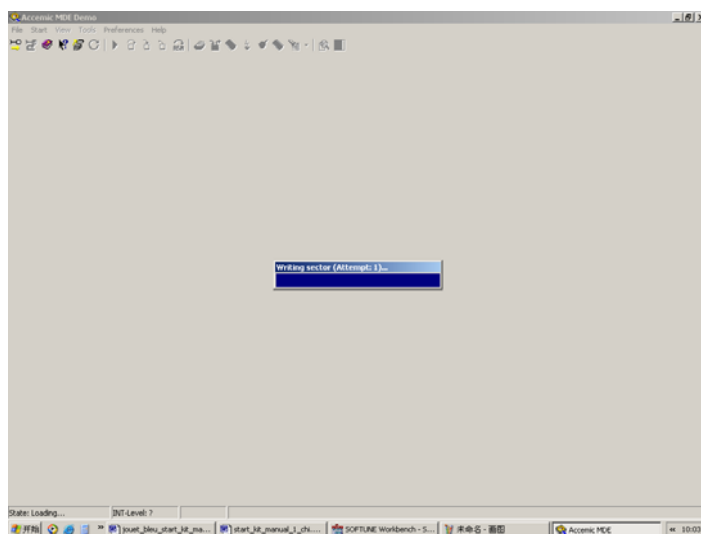
以下介绍的方法用户可作为一个参考，在想手动读入某 ABS 文件时，可以采用这个方法。

关于 ABS 文件的解释：在每个 project 文件夹里，您都可以找到一个 ABS 文件夹，我们要读取的文件就在这个文件夹里（通常必须通过 SOFTUNE 编译才会产生这个文件，并且每次编译都会重新更新这个文件，这个文件里存放的就是机器代码）

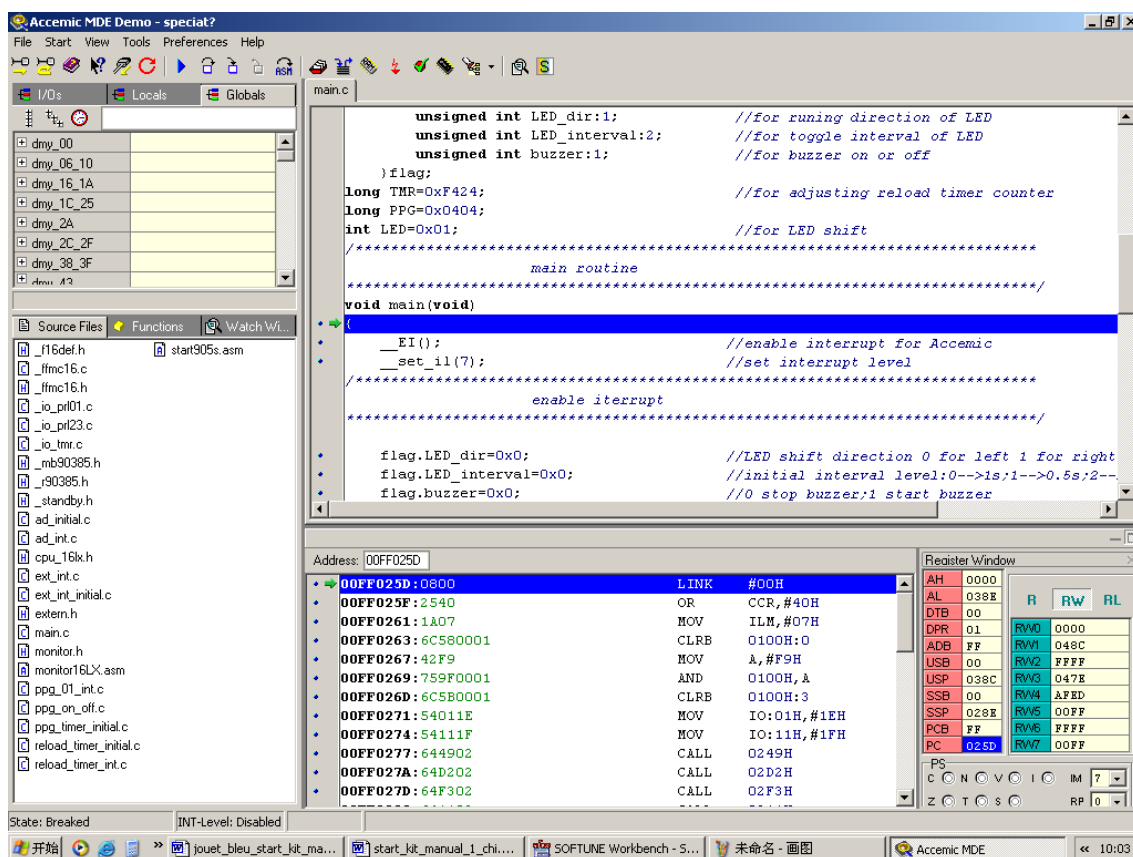
您可以从刚才「Fujitsu_Starter_kit.zip」文件的解压路径下找到 ABS 文件夹

例如：样例程序解压路径\special\Debug\ABS\special.abs

接着，在 Accemic 上通过「File」→「Load File...」→读取 ABS 文件



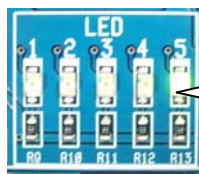
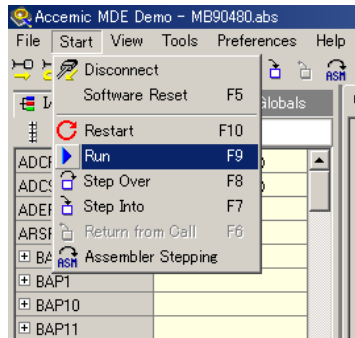
ABS 文件成功读取后可以看到如下的 Accemic 界面



接着让我们执行样例程序「special.prj」。

请点击「Start」—「Run」。

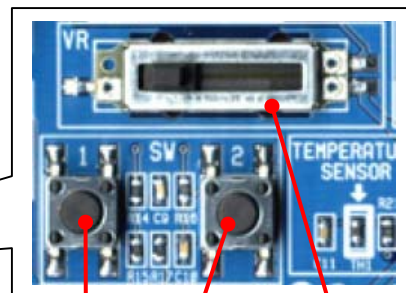
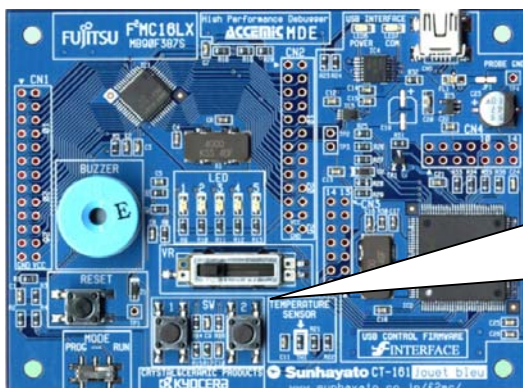
于是便可以观察到评测板上 LED1~LED5 在闪烁。



当执行了样例程序以后评测板上的 LED1~LED5 便会开始移位。

关于 special.Prj 程序的说明：

按键控制	评测板动作
按下 SW1	LED 的移动方向将改变 移动方向：左移→右移→左移→右移……
按下 SW2	LED 的移动速度将改变 移动速度：1 秒→0.5 秒→0.25 秒→0.125 秒→1 秒……
长时间按下 SW 1	开启或关闭蜂鸣器
调整滑动变阻器	再蜂鸣器被打开的状态下，调整滑动变阻器可以改变蜂鸣器的音调

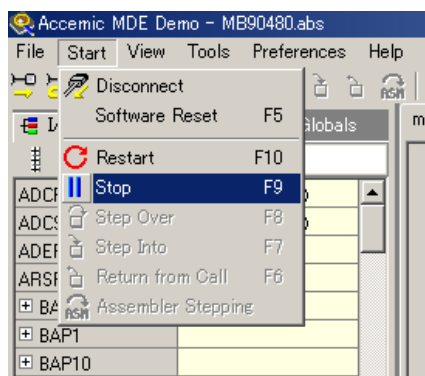


SW1 SW2 滑动变阻器

接着，让我们停止程序的运行。

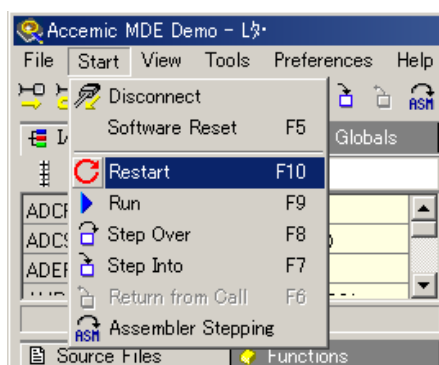
请点击「Start」－「Stop」。程序的运行将会被停止。

程序一旦停止运行，LED1~LED5 的闪烁将会立即停止。



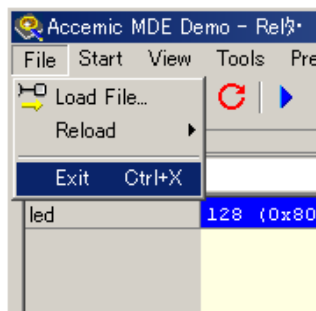
对程序进行 RESET。

点击「Start」－「Restart」。程序将会被 RESET。



1.1.7 ACCEMIC MDE 退出

要退出 ACCEMIC MDE，请点击「File」－「Exit」。

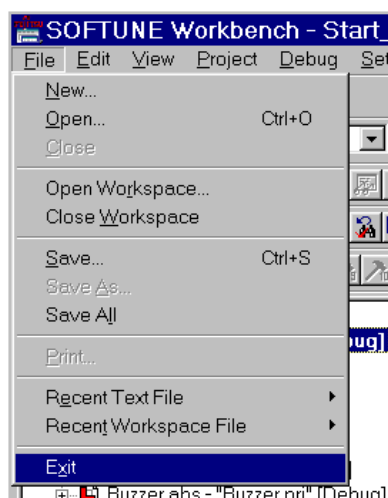


ACCEMIC MDE 的界面将被关闭。

1.1.8 SOFTUNE 的退出

退出 ACCEMIC MDE 以后接着退出 SOFTUNE。

请点击「File(F)」－「Exit(X)」，退出 SOFTUNE。

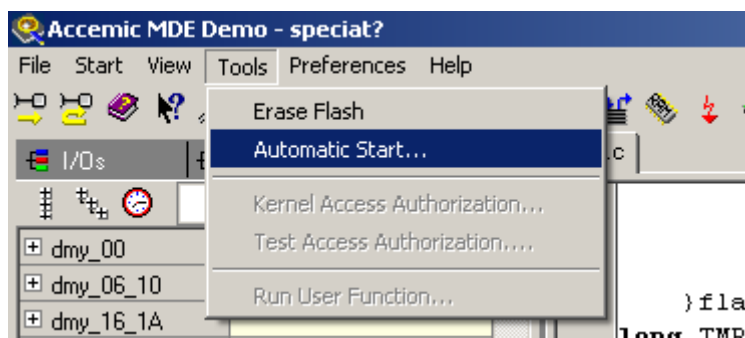


SOFTUNE 的界面将被关闭。

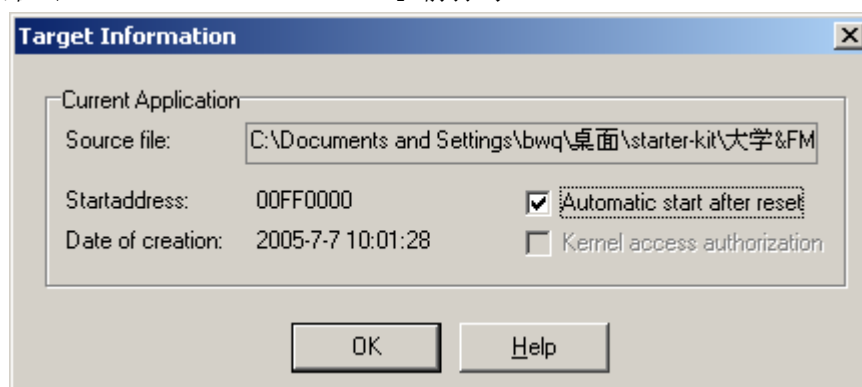
1.1.9 关闭 Accemic 的情况下启动单片机

我们还可以将单片机设定为单机启动模式，即在关闭 Accemic MDE 的情况下启动单片机，具体设置方法如下：

先利用 Accemic 在单片机上确认目标程序是否能够准确运行，确认方法如同 1.1.6 节所示，如果确认动作无误，您可以选择「Tools」－「Automatic Start」将单片机设定为单机启动模式



然后在「Automatic Start after reset」前打勾。



点击 OK 以后 Accemic 将要对单片机进行重写动作，重写完成以后单片机便进入了单机启动模式。

现在你只要按下单片机上的 Reset 键，单片机就可以像刚才一样动作。（即使将 Accemic 关闭）

以上便是对样例程序进行 Debug（调试）的基本操作方法。

接着让我来为你介绍如何自己编译程序来使 LED 闪烁！

2 编写使 LED 闪烁的程序

其实在我们的实际生活中随处可见 LED 的踪影。例如，笔记本电脑、数码摄像机、洗衣机、空调、电饭煲等，还有在地铁站里表示地铁出发时刻表的荧光板里有些用的也是 LED 器件。

本章将为您介绍如何使用单片机的输出端口来控制 LED 的闪烁。

2.1 关于 LED 的介绍

所谓 LED 是 Light Emitting Diode(发光二极管)的简称，顾名思义，它是一种通过电流来发光的半导体。LED 一般用图 2-1 的电路符号来表示。LED 共有阳极、阴极两个端子，若阳极接电源正端，阴极接电源负端则电流将通过 LED，从而使其发光。若电路图如下图所示，则闭合 SW(开关)时 LED 将发光，断开 SW 时 LED 将熄灭。一般调整 LED 的辉度可通过调整电路中电阻的阻值（即改变电流）来实现。电流越大，LED 则越亮。

根据 LED 所用材料不同，不同种类的 LED 可以发出红、橙、黄、绿、蓝等不同颜色的光，通过不同的组合还可以再现组成白光的三元色(红、绿、蓝)等。

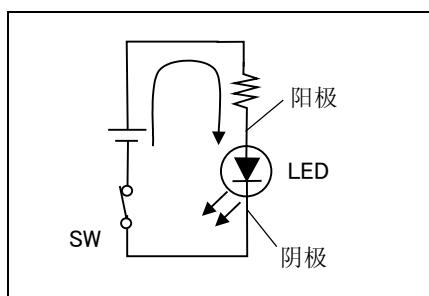


图 2-1 LED

LED 与普通的白炽照明体相比具有功耗低、使用寿命长、反应速度快等特点，故多被用在照明机械、液晶背光、信号灯还有 LED 显示器上。本产品上也采用了 5 枚 LED 如图 2-2 所示。(LED1, 2, 4, 5: 绿色; LED3: 红色)。



图 2-2 评测板上的 LED

2.2 LED 为何会发光

LED 的内部结构其实是半导体设备中最为基本的 PN 结结构，它由多电子的 N 型半导体和多空穴的 P 型半导体结合而成，如图 2-3 所示

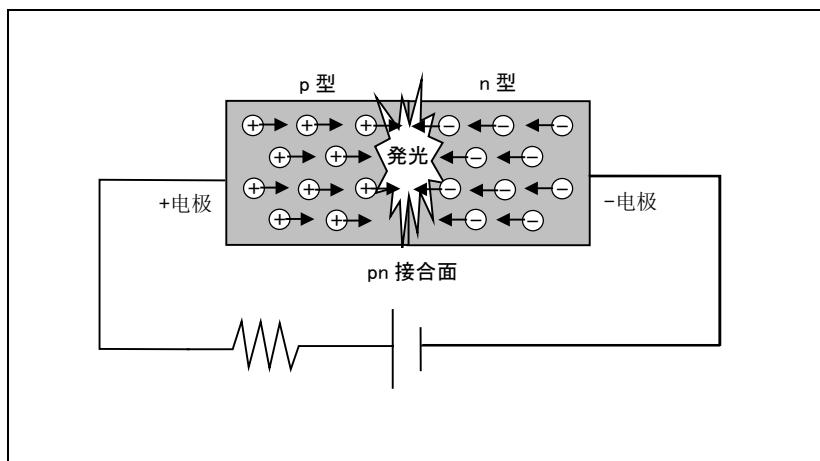


图 2-3 PN 结

当我们将 P 型半导体端接电源正极，N 型半导体端接电源负极后，则电子将从 N→P，而空穴将从 P→N 进行迁移，这样便产生电荷的定向运动从而形成了电流。但是在此过程中，空穴与电子还会产生复合，复合后的总能量将小于空穴与电子单独存在时的总能量，换句话说总能量产发生了减小，其减小的能量便会以光的形式辐射到周围空间中去。而具体发出的光的颜色则取决于 LED 的制作材料，具体可参考表格 2-1。

表格 2-1 半导体材料与发光颜色

半导体材料	发光颜色
ZnCdSe	蓝
ZnTeSe	绿
AlGaAs	红

2.3 利用单片机使 LED 发光的方法

在图 2-1 中是通过 SW 开关来控制 LED 的发光的，现在我们将用单片机来实现图 2-1 中 SW 开关的功能，即用单片机来控制 LED。

评测板与 LED 的连接如图 2-4 所示，为了便于理解，将其简化为如图 2-5 所示。

图 2-5 (a)中，由于单片机 P10 端口输出高电平，故 LED 中无电流通过，呈现熄灭的状态；图 2-5 (b)中，由于单片机 P10 端口输出低电平，故 LED 中有电流流过，从而呈现发光状态。至于单片机这个“开关”（其实是端口的输出电平），则可以通过用户的程序来进行设置。

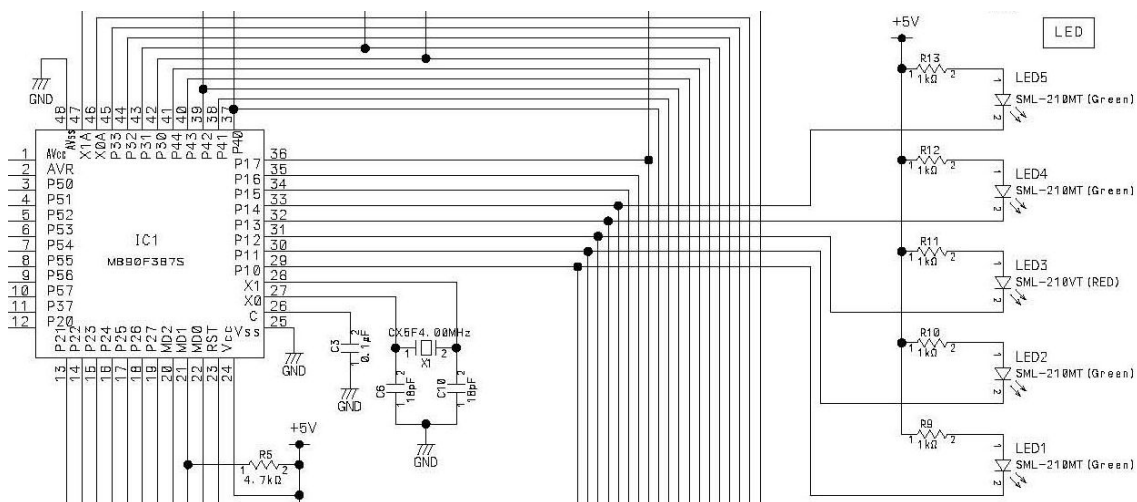


图 2-4 实际的电路图

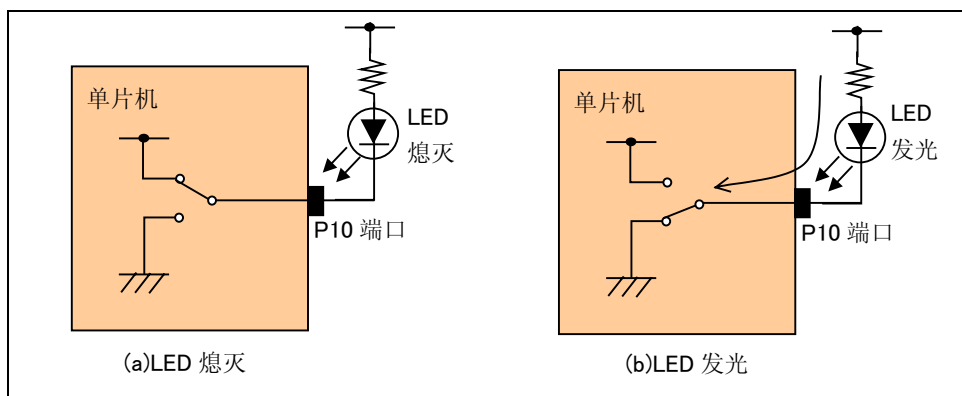


图 2-5 控制 LED 开关的电路图（单片机内部已简化）

端口 P10~P14 的输出（或输入）是通过如图 2-6 所示的 PDR1 寄存器和 DDR1 寄存器共同设定的。

简单的说 PDR1 寄存器（对应 P10～P17）用来设置端口输出（或输入）的电平(0: Low; 1: High), DDR1 寄存器（对应 P10～P17）用来设定端口的传输方向（0: 由外部输入; 1: 由内部输出）。

PDR: Port Data Register

DDR: Data Direction Register

所谓寄存器（register）是用来存放单片机设置信息的存储器。通过写入或读出有关寄存器的值可以控制和查询当前单片机外围设备（peripheral）的运行情况。所谓外围设备是指单片机内诸如 I/O 端口、定时器（timer）、A/D 转换器等的功能电路。

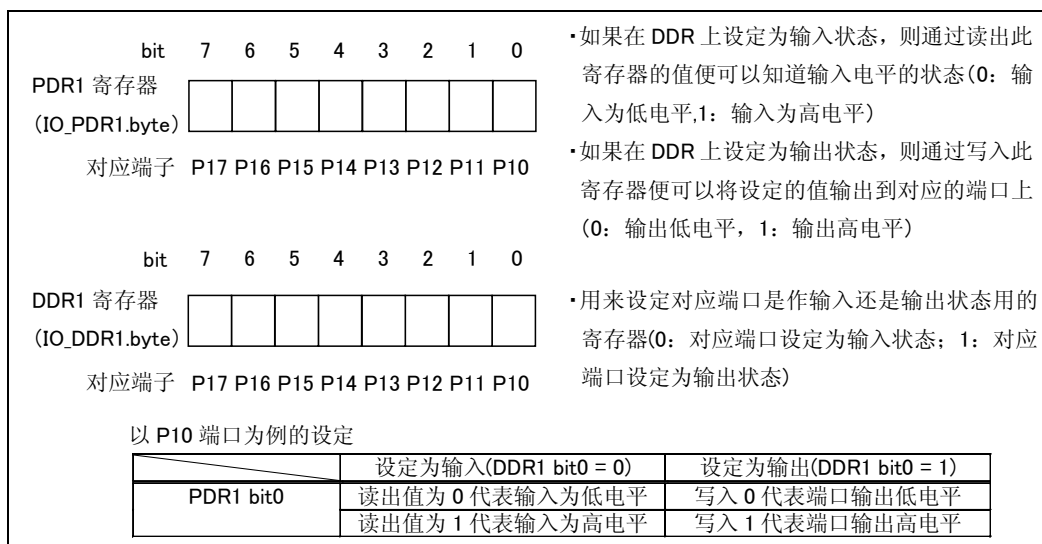


图 2-6 P10～P14 端口输出寄存器的说明

其实，将图 2-4 具体化，就成了图 2-7 所示的电路。它由一个 P 沟道晶体管和一个 N 沟道晶体管所组成。当 N 沟道晶体管为 ON 状态时输出为低电平；当 P 沟道晶体管为 ON 状态时输出为高电平。通过设定 PDR1 的值，可以使端口 P10～P17 输出不同的值。在图 2-7 中 PDR1 的 bit0 位被设定为“0”，所以 P10 输出为低电平；而 PDR1 的 bit1 位被设定为“1”，所以 P11 输出为高电平。不难想象，此时 LED0 发光，而 LED1 则熄灭。

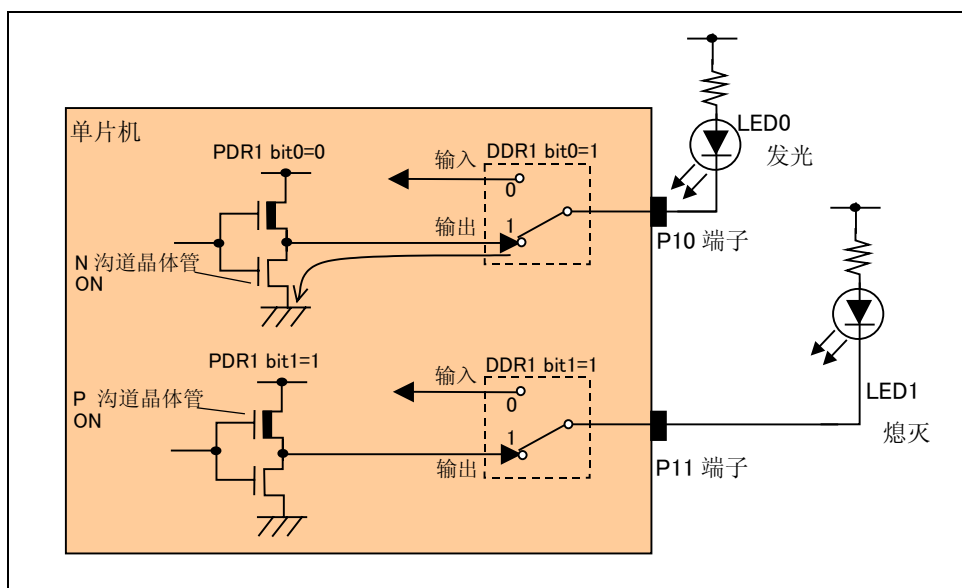


图 2-7 由 PDR 控制的端口状态（单片机内部已简化）

由于实际的评测板上只有 P10~P14 端子上连接有 LED, 所以使用时只需对 PDR1 的低 5 位进行设定便可控制这 5 个 LED 的状态了。另外, 值得注意的是: 由于启动后 (或者 reset 后) DDR 寄存器的初始值为 0 (即输入状态), 所以, 如果要使对应端子状态改为输出状态来控制 LED, 则必须将 DDR 寄存器的相应位设定为 1 (即输出状态)。

2.4 LED 发光程序的制作及运行

这一节, 我们将实际编写一个使 LED 发光的 C 源程序。

2.4.1 程序概要

由于电路设计上是低电平有效, 所以我们将单片机的 P14 端子设定为低电平输出, 来驱动 LED5 发光。程序的流程图如图 2-8 所示。注意: 在流程图中是先设定 PDR1 的值, 然后再设定 DDR1 的值的。在使端子作输出功能时必须以这个先后顺序作设定, 即先设定 PDR1 的值 (端子的输出电平), 然后在设定 DDR1 的值 (端子的数据方向)。这是因为启动以后 PDR1 的初始值是不定的, 而我们并不希望单片机输出一个不定的值。

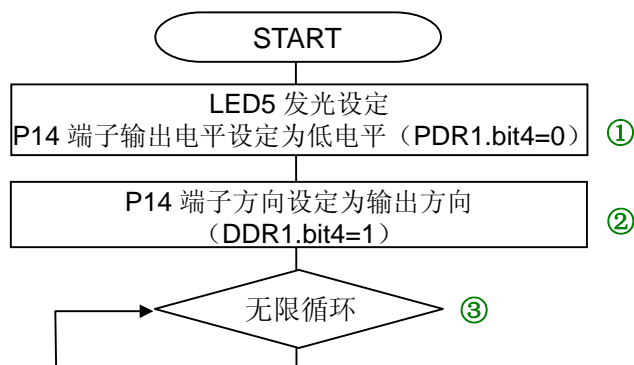


图 2-8 LED 发光程序流程图

2.4.2 程序的制作与运行

首先请按照「附录 A.1」内所示顺序，打开“sample.prj”内的源文件“main.c”，然后将

图 2-9 内所示的内容添加到 main.c 源程序里。注意：除了 main.c 以外，Project 内的其他文件请不要做任何修改。当添加完毕后，再按照「附录 A.2」所示的顺序对源程序进行“build”。如果出现编译错误，请核对

图 2-9 检查添加的内容里是否有文字错误。“build”顺利完成后“ACCEMIC MDE”将自动启动。

接着，转向“ACCEMIC MDE”界面，对编译好的程序进行 debug（调试），并实际测试运行情况。程序的执行方法请参照「附录 A.3」。当程序开始运行后，评测板上的 LED5 将会发光。

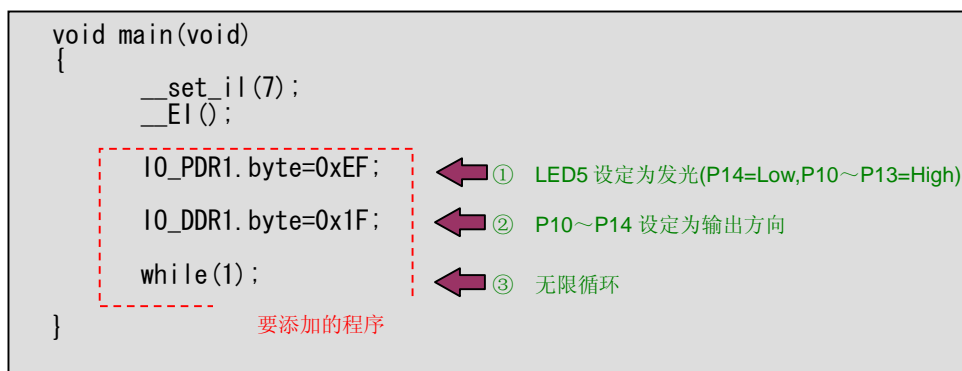


图 2-9 LED 发光程序

说明：在上述源码中有诸如“IO_XXX.byte”或者“IO_YYY.bit”之类形式的变量，其实这些都为了程序员方便标记而在头文件里被事先预定义好的形式。

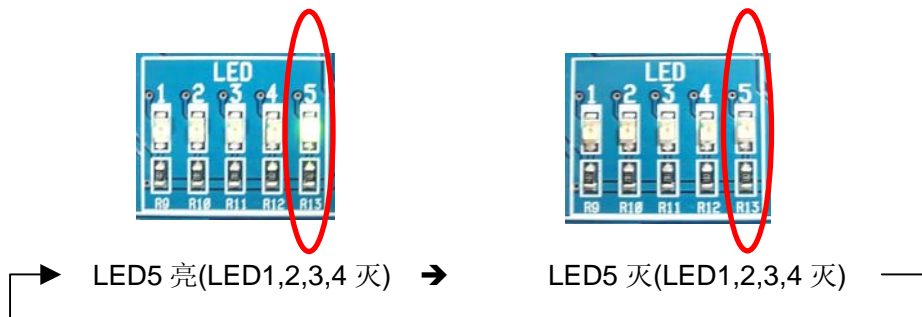
比如：IO_PDR1.byte → 就是 PDR1 的值
具体请参照、「附录 B」。

2.5 LED 闪烁程序的制作与运行

这一节，我们将实际编写一个使 LED 闪烁的 C 源程序。

2.5.1 程序概要

将单片机 P14 端子的输出电平以一定时间间隔进行高低切换，这样便可使 LED5 发生闪烁，大致过程如下图所示。



程序的流程图如图 2-10 所示。不难发现，此程序只是在图 2-8 的基础上添加了 LED 亮灭切换所需的时间间隔的处理部分。在此程序中 LED 的亮灭之间间隔处理用了 for 循环（参考 C 语言设计），为了能让肉眼明显分辨出 LED 的亮灭切换，我们暂时把循环值设定为 30000，通过更改这个值我们可以延长或缩短切换的时间。

• 亮灭切换的时间间隔可由 for 循环语句实现 `for(i=0;i<30000;i++);`

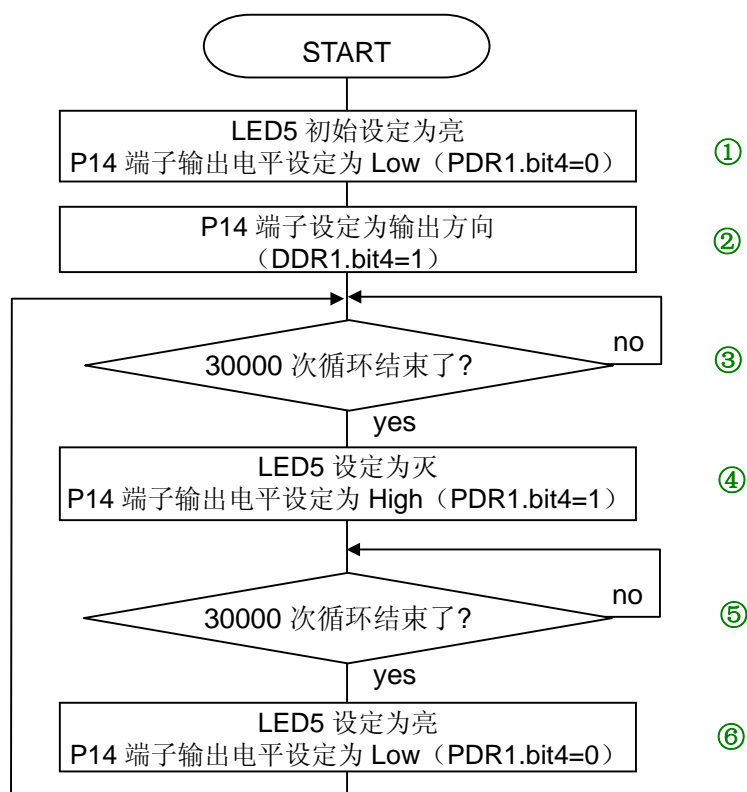


图 2-10 LED5 闪烁程序流程图

2.5.2 程序的制作与运行

接着让我们实际来制作源程序！首先，请按照「付録 A.1」所示，打开“sample.prj”内的源程序“main.c”，在 2.4 节里做好的源程序的基础上再添加

图 2-11 中虚线内的部分(注意原来 `while(1)` 后面的 “;” 要删掉，添加 “{ }”)。注意：除了 main.c 以外，Project 内的其他文件请不要做任何修改。当添加完毕后，再按照「付録 A.2」所示的顺序对源程序进行 “build”。如果出现编译错误，请核对

图 2-11 检查添加的内容里是否有文字错误。“build” 顺利完成后 “ACCEMIC MDE” 将自动启动。

接着，转向 “ACCEMIC MDE” 界面，对编译好的程序进行 debug（调试），并实际测试运行情况。程序的执行方法请参照「附录 A.3」。当程序开始运行后，请确认评测板上的 LED5 是否在闪烁。

```

void main(void)
{
    __set_irq(7);
    __EI();

    IO_PDR1.byte=0xEF;
    IO_DDR1.byte=0x1F;

    while(1)
    {
        int i;
        for (i=0; i<30000; i++);
        IO_PDR1.byte=0xFF;
        for (i=0; i<30000; i++);
        IO_PDR1.byte=0xEF;
    }
}

```

① LED5 设定为亮(P14=Low,P10~P13=High)

② P10~P14 设定为输出方向

③ LED5 发光时间

④ LED5 设定为灭(P10~P14=High)

⑤ LED5 熄灭时间

⑥ LED5 设定为亮(P14=Low,P10~P13=High)

要添加的程序

图 2-11 LED 闪烁程序

说明：在上述源码中有诸如“IO_XXX.byte”或者“IO_YYY.bit”之类形式的变量，其实这些都为了程序员方便标记而在头文件里被事先预定义好的形式。

比如：IO_PDR1.byte → 就是 PDR1 的值

具体请参照、「附录 B」。

3 用开关 SW 控制 LED 的亮灭

本章将说明如何通过开关 SW（按压式）来控制 LED 的亮灭。并且将知道您制作相应的控制程序。

3.1 单片机如何检测 SW 的状态

次评测板上共有 2 个 SW 开关，如图 3-1 所示，分别接在单片机的 P25 端口及 P27 端口上。关于单片机如何检测 SW 开关状态，下面将给予说明。

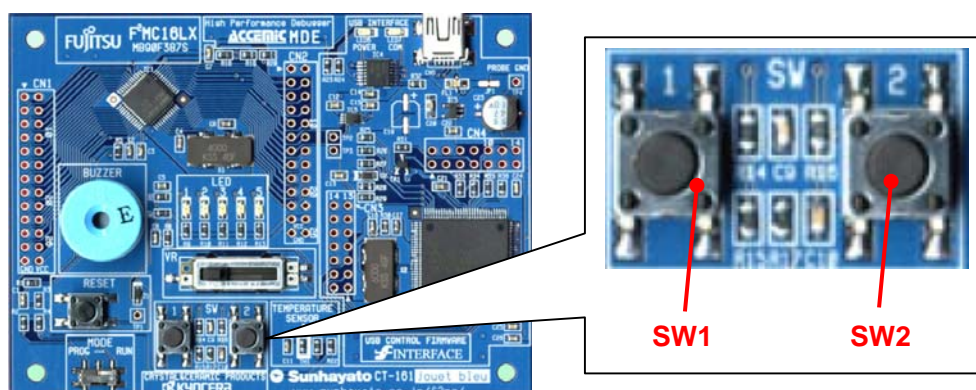


图 3-1 评测板上的开关 SW

图 3-2 显示了开关 SW1 与单片机端口的连接。如图所示，开关 SW1 连接在单片机的通用输入输出端子 P25 上。不难发现，当 SW1 断开时（OFF）P25 与电源相连，输入电压为 5V（Vcc）高电平；相反，当 SW1 闭合时（ON）P25 与地相连，输入电压为 0V（GND）低电平。通过这样的简单操作，我们便可以改变 P25 端子上的输入电平。SW2 的情况与 SW1 相同，只不过与 SW2 相连的端口为端口 P27。

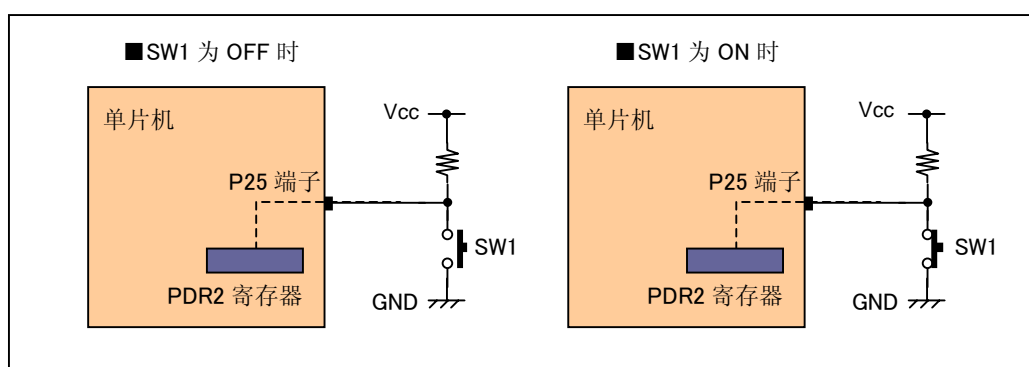


图 3-2 SW 于单片机端口的连接

关于 PDR 的说明：上述端口输入电平的变化其实可以通过单片机内的程序来进行检测。在单片机内部，您可以借由 I/O 寄存器 PDR2（类似刚才的 PDR1）来检测端口 P25、27 端子上的输入电平。换句话说，在程序中通过读出寄存器 PDR2 的值，便可以知道端口上

输入电平的状态。在此，我们再对 PDR2 寄存器的特点做点说明。PDR2 使用来表示端口 2（P20~27）状态的寄存器，长度为 8bit。它与端口 2 的对应关系如图 3-3 所示。当它用来表示端口状态时，“1”表示输入高电平；“0”表示输入低电平。因此，当读出 PDR2 的第 5 位值时便可以知道 P25 端口的状态，由此也就可以知道开关 SW1 的操作状态了。同样读出 PDR2 第 7 位的值便可以知道 SW2 的操作状态。

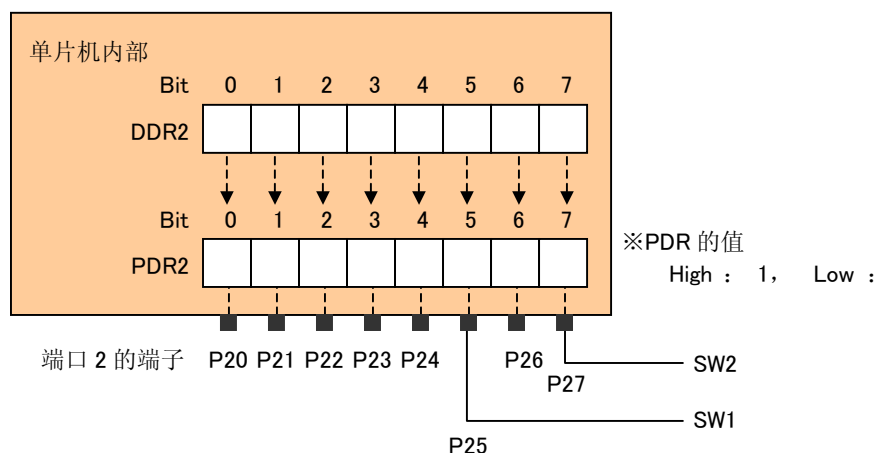


图 3-3 PDR 及 DDR 寄存器的概略图

关于 DDR 的说明：在使用端口 2 之前，你必须先指定端口上各端子的传输方向。DDR 正是用来设定端口 2 传输方向的（输入或者输出）。其长度也是 8bit。DDR2 的各 bit 位与端口 2 的各端子的对应关系也如图 3-3 所示。当对应端子作输出端子使用时，DDR 的相应位须设定为“1”；当对应端子作输入端子使用时，DDR 的相应位须设定为“0”。现在，为了读取开关 SW1 及 SW2 的输入信号，故 P25 及 P27 须作“输入”端子，相应的 DDR2 的第 5 位及第 7 位应设定为“0”。

综上所述，要在程序中读取开关 SW 的操作状态的话，你可以按照以下 2 个步骤操作。

将 DDR2 的第 5 及第 7 位设定为“0”，将 P25，P27 作输入端子使用。

读取 PDR2 的第 5 及第 7 位的值：

- 读出值为“0”时，说明 SW 为 ON 状态
- 读出值为“1”时，说明 SW 为 OFF 状态

3.2 通过 SW 控制 LED 程序的制作与运行

接下来我们来实际制作一个检测 SW 状态的程序。此外，为了能明确地了解 SW 的变化状态，我们将用 SW 来控制 LED 的亮灭。

3.2.1 程序概要

这段程序主要想实现的内容如下所示。流程图如图 3-4 所示。

当 SW1 按下时 LED1 变亮。

SW1 为 ON 时，LED1 将持续点亮。

SW1 为 OFF 时，LED1 熄灭。

SW2 与 SW1 一样，用来控制 LED2 的亮灭。

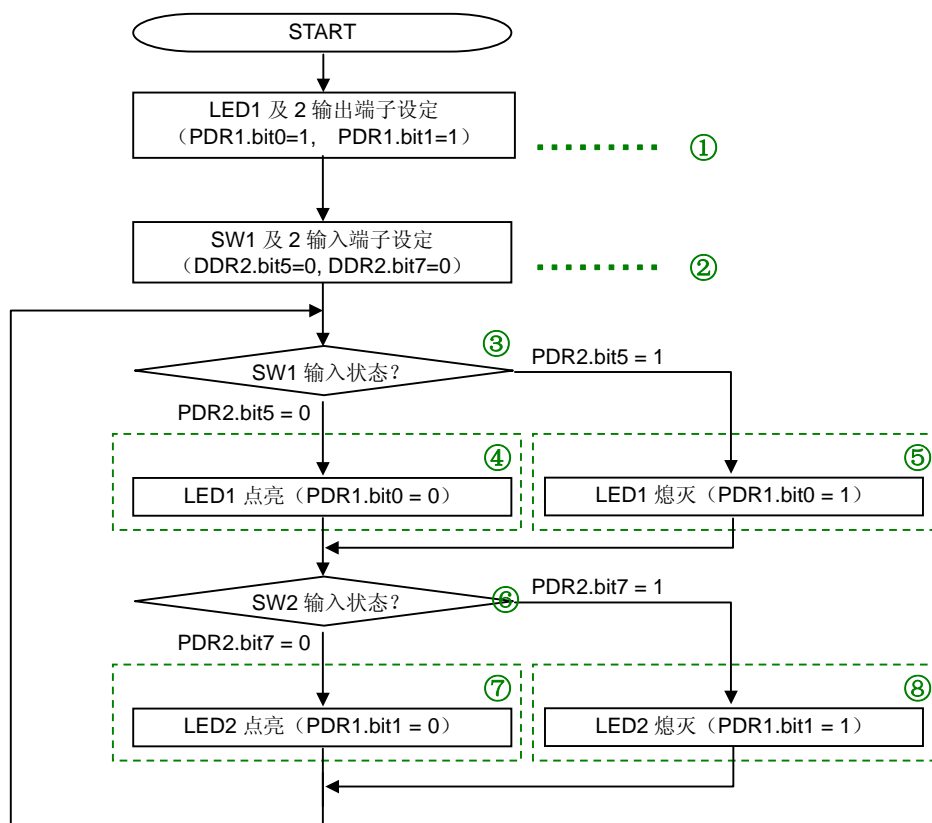


图 3-4 程序流程图

从次流程图中不难看出，程序首先对控制 LED 的端子做了初始设定。然后又对与 SW 相连的输入端子做了方向设定。这些设定正如刚才所说明的，是不能省略的。接着，为了检测 SW1 的状态将 PDR2 的第 5 位值读出。根据读出值的状态来使 LED1 点亮或熄灭。LED1 点亮或熄灭是通过 PDR1 的第 0 位的设定值来控制的。至于 SW2 的原理也和 SW1 相似，只不过操作对象换为 LED2 罢了。

3.2.2 程序的制作与运行

首先，请参照「附录 A.1」打开“sample.prj”内的源文件“main.c”，然后将图 5-3 虚线部分内的内容输入到主程序内。注意：除了 main.c 以外，Project 内的其他文件请不要做任何修

改。当添加完毕后，再按照「付録 A.2」所示的顺序对源程序进行“build”。如果出现编译错误，请核对图 5-3 检查添加的内容里是否有文字错误。“build”顺利完成后“ACCEMIC MDE”将自动启动。

接着，转向“ACCEMIC MDE”界面，对编译好的程序进行 debug（调试），并实际测试运行情况。程序的执行方法请参照「附录 A.3」。程序开始运行后，你可以按一下 SW1 和 SW2，然后观察实际情况的变化。

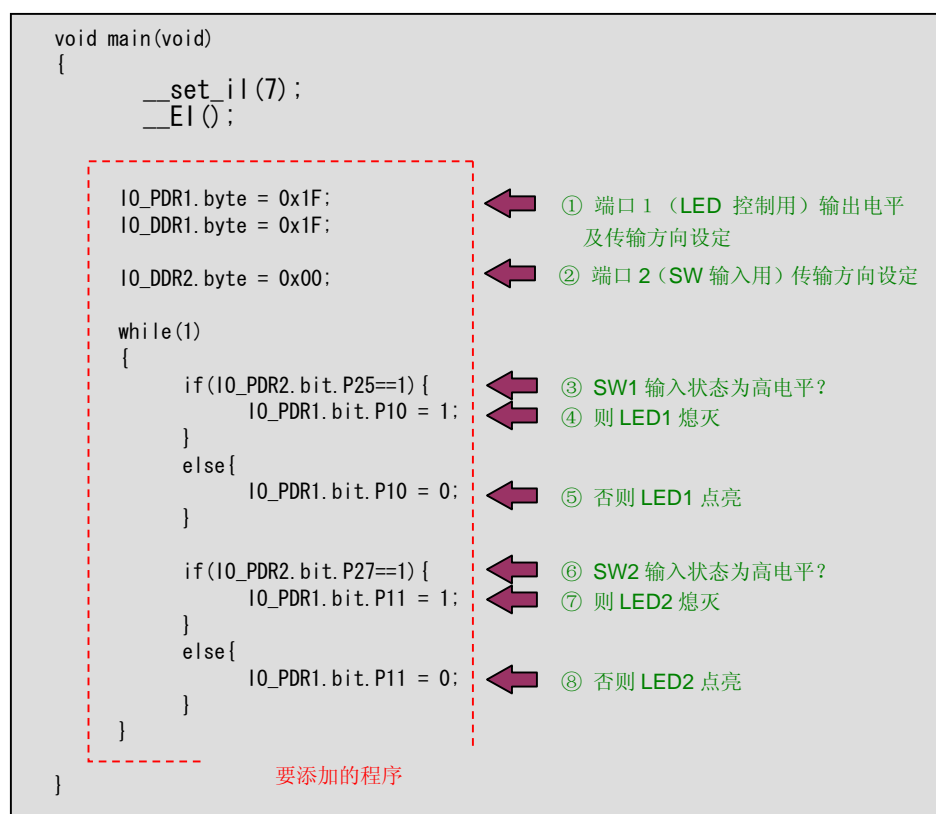


图 3-5 主程序例

说明：在上述源码中有诸如“IO_XXX.byte”或者“IO_YYY.bit”之类形式的变量，其实这些都为了程序员方便标记而在头文件里被事先预定义好的形式。

比如：IO_PDR1.byte → 就是 PDR1 的值

具体请参照、「附录 B」。

4 如何使用蜂鸣器

在我们的生活中其实到处可以听到蜂鸣器的声音，比如闹钟，定时器，电子警报器等等。

这些人为制造出来的声音其实有很多就是利用单片机来发出的。声音的响度，音色均可以通过单片机来设定并且控制。接下来就让我们来学学如何用单片机来控制蜂鸣器发出声音吧。

4.1 蜂鸣器内所用的材料

在具体解释蜂鸣器发音原理前，先让我来解释一下蜂鸣器内所使用材料的特性。

压电材料——压电材料的最大特点就是可以因机械变形产生电场，也可以因电场作用产生机械变形。

蜂鸣器内所用的材料就是压电材料的一种。此外，用到压电材料的产品还有：水晶振子、压电喇叭、晶振耳机、震动传感器、麦克风等等。

压电材料的压电特性

压电材料内的分子多为极性分子。

在极性分子组成的晶体内，电场具有一定的方向。具有极性特点的晶体内部结构如图 4-1 所示。

极性分子组成的晶体内的电场由正电荷“+”指向负电荷“-”。

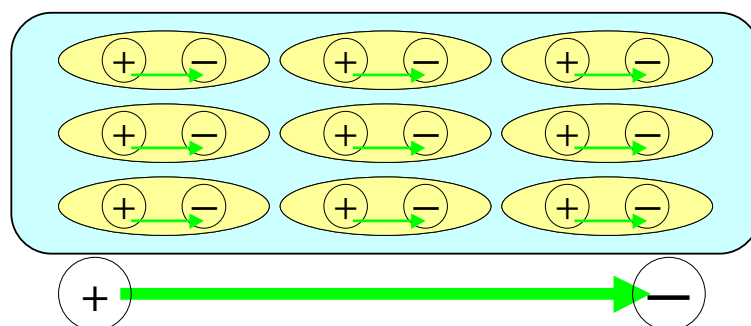


图 4-1 分子的极性

4.1.1 压电性的特点

如图 4-2 所示，当外加电场方向与压电材料内部电场方向一致时，材料将会伸长；相反材料则会缩短。

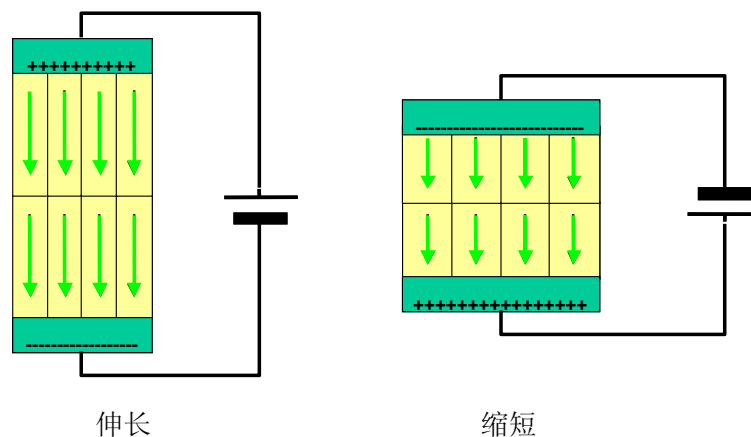


图 4-2 具有压电性的晶体

4.1.2 压电材料的应用

如图 4-3 所示，当在压电材料两端施加一个交流电源时，压电材料的伸长与缩短将会以一定的周期重复地交替发生，当改变输入电流的频率时，压电材料震动的频率也将随之变化，这样便会产生震动。并且振动的频率可以由外界输入频率来控制。如果此压电材料震动的能量大到足够引起空气共振的话，声音便产生了。

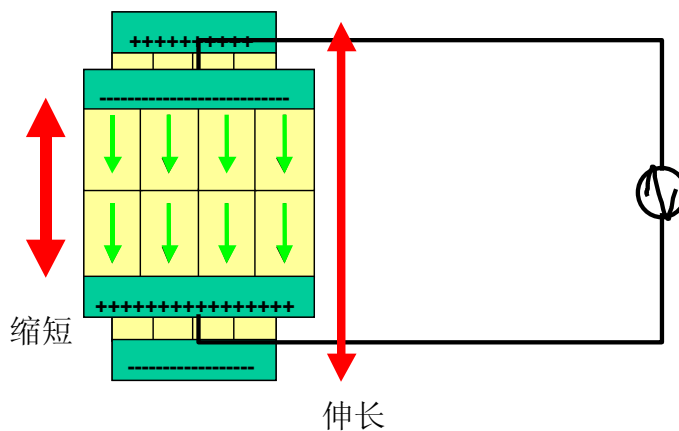


图 4-3 压电材料的应用

4.2 单片机与压电蜂鸣器

正如刚才所说明的那样，要使压电式蜂鸣器发出声音的话，必须给他外加一个交流电压或者脉冲波电压之类的具有一定变化频率的电压。这里我们就使用由单片机发出的脉冲波（电压）来驱动压电式蜂鸣器发出声音。

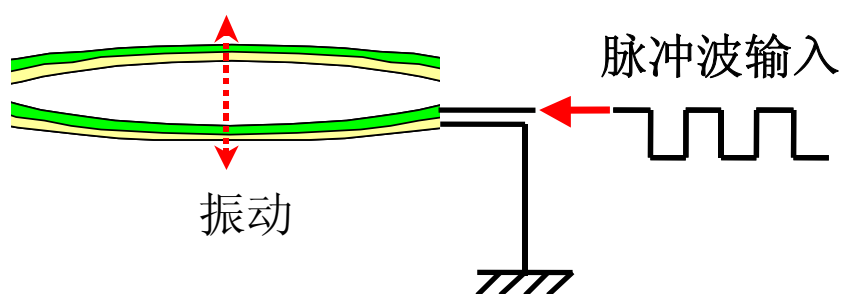
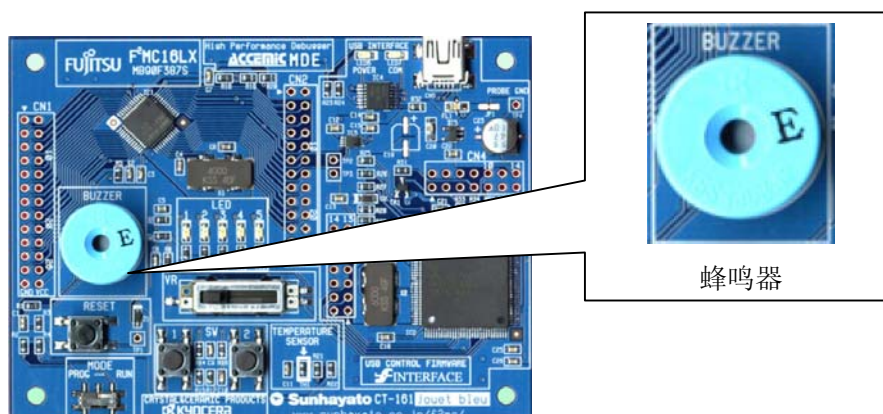


图 4-4 使用压电材料的蜂鸣器

4.2.1 自励式与他励式

蜂鸣器有自励式与他励式两种形式，前者由于材料的特性只能发出单一频率的声波，而后者则可以根据外加电压的频率而改变输出声波的频率。

本产品上搭载的蜂鸣器是 KYOCERA 公司的 KBS-13BB-4P-2 他励式蜂鸣器，可以根据输入电压的频率来改变输出声波的频率。



4.2.2 由单片机发出的脉冲波

由单片机发出脉冲波其实有很多种方法，但是其基本原理都是利用单片机内部的 timer(定时器)设定好计数时间，然后以一定的时间间隔使端口上的输出电平产生反转（“H”与“L”互换）从而产生有一定频率的方波。当然，由于目前单片机技术的发展，一个单片机内已经可以内藏有多个 timer，根据用途的不同可以选择不同的 timer。这里我们只需简单的输出一定的波形就可以了，所以我们选择了相对简单的 PPG timer 来产生蜂鸣器用的方波。

（PPG=Programmable Pulse Generator=可编程脉冲波发生器）

4.3 如何使用 PPG 使蜂鸣器发出声音

PPG 是 Programmable Pulse Generator 的简称，顾名思义它就是单片机内用来产生各

种脉冲波的功能电路。基本功能有：

L 幅宽、H 幅宽的设定

PPG 计数器（定时器）的设定

利用上述 2 个功能就可以生成各种频率各种占空比的方波。下面将介绍 PPG 的使用方法。

4.3.1 L 幅宽与 H 幅宽的设定

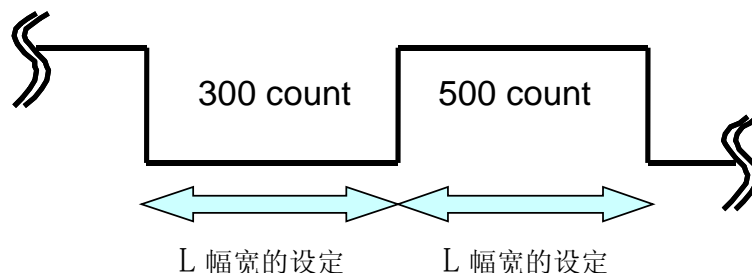


图 4-5 脉冲波周期与 L、H 幅宽

由 PPG timer 输出的脉冲波的波形其实是通过分别对 L 幅宽与 H 幅宽的设定来实现的。这里，我们需要用到一个 count（计数）的概念。例如：我们事先通过 PPG 的相关寄存器将 L 幅宽设定为 300 count, H 幅宽设定为 500 count, 设定完以后启动 PPG timer, 则 PPG 先在 0~300 count 内保持端口输出为 L, 然后在接下去的 500 count 范围内 300~800 (300+500) 内保持端口输出为 H, 就这样交替地进行输出。不难看出这样设定后的占空比就为 3/5, 周期为 800 count。

关于 count 的说明：count 其实是单片机内部的动作，它就好比一个内部的计数器，每经过一次机器周期就计数一次，单片机内部默认的机器周期为(1/2MHz)s, 这也就是 count 的周期。

4.3.2 PPG count clock

利用 PPG timer 输出脉冲波除了设定 L、H 幅宽以外，还需设定脉冲波的周期。其实，脉冲波的 count 周期是以单片机内部的机器周期为基准进行 count 的，所谓 count clock 就是指 PPG count 所用的最小基准时间。通过相关寄存器的设定，我们可以选择以 1 倍速、1/2 倍速、1/4 倍速、1/8 倍速、1/16 倍速来进行 count。

例如：我们以 1 倍速进行 count, L、H 分别为 300 count 与 500 count, 则周期为：

$(300+500) \times 1 \text{ 倍速} \times 1/2\text{MHz}$ (机器周期默认为 1/2MHz)

$$= 800 \times 1 \times (1/(2 \times 10^6))$$

$$= 0.0004\text{s}$$

占空比为 3/5。

4.4 蜂鸣器程序的制作与运行

接下去让我们实际制作一个程序来控制蜂鸣器发出声音吧。

4.4.1 程序概要

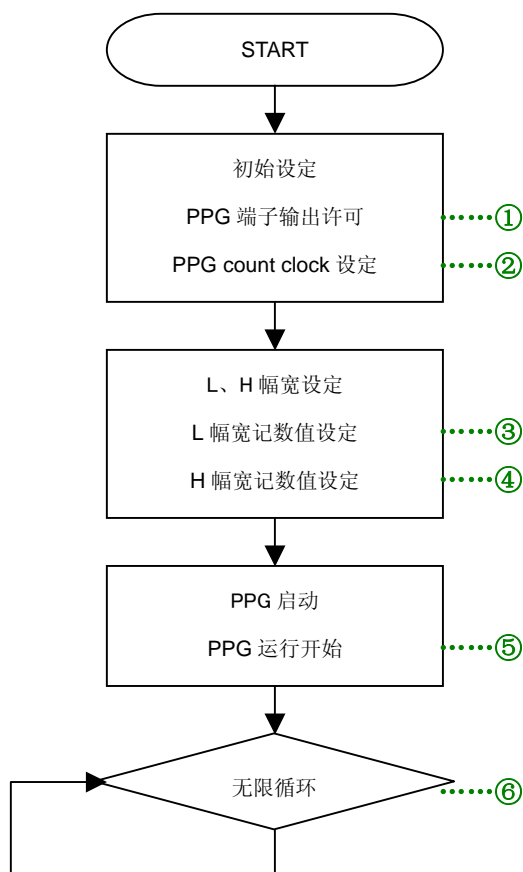


图 4-6 程序流程图

下面将对 PPG 的设定方法做具体介绍。

先来对 L、H 幅宽做设定。这些设定都是事先通过专用的寄存器来进行的。

这里作为例子，我们把 PPG 的输出脉冲波的频率设定为 4kHz。为了方便计算，把占空比设定为 50%，PPG 的 count clock 设定为 1 倍速（即与机器周期相同）。CPU 默认的机器周期为 2MHz。于是，L、H 幅宽的记数值可由下式来求得：

$$T_{\text{输出波周期}} = (L_{\text{count值}} + H_{\text{count值}}) \times 1\text{倍速} \times T_{\text{机器周期}}$$

$$(L_{\text{count值}} + H_{\text{count值}}) = \frac{T_{\text{输出波周期}}}{T_{\text{机器周期}}} = \frac{1/4000}{1/2000000} = 500$$

由于占空比为 50%，故 L、H 两边 count 值各设定为 250 即可。

明确了 reload count 值以后，接着可以设定 count clock 周期，即选择用几倍速来进行 count（计数）。然后允许对应的端子作为 PPG timer 的输出端口，否则 PPG 波形将无法从端口输出。最后启动 PPG timer 来输出波形。

具体设定如下表：

表格 4-1 寄存器设定

设定项目	设定寄存器 (bit)	设定值
	寄存器名称	
L 幅宽	PRL (bit 0~7)	0xFA
	PPG reload register (L)	(250 count)
H 幅宽	PRLH (bit 0~7)	0xFA
	PPG reload register (R)	(250 count)
PPG count clock	PPG (bit 0~7)	0x00
	PPG count clock 设定寄存器	(选择 1 倍速)
PPG 端子输出许可	PPGC (bit 13,)	0 (禁止)
	PPG 运行模式控制寄存器	1 (许可)
PPG 启动许可	PPGC (bit 15)	0 (禁止)
	PPG 运行模式控制寄存器	1 (许可)

以上这些设定都是通过相应的寄存器设定来完成的。

4.4.2 程序的制作与运行

首先，请参照「附录 A.1」打开“sample.prj”内的源文件“main.c”，然后将

图 4-7 虚线部分内的内容输入到主程序内。注意：除了 main.c 以外，Project 内的其他文件请不要做任何修改。当添加完毕后，再按照「附录 A.2」所示的顺序对源程序进行“build”。

如果出现编译错误，请核对

图 4-7 检查添加的内容里是否有文字错误。“build”顺利完成后“ACCEMIC MDE”将自动启动。

接着，转向“ACCEMIC MDE”界面，对编译好的程序进行 debug（调试），并实际测试运行情况。程序的执行方法请参照「附录 A.3」。

PPG count clock 的计数频率设定为 2MHz。“L”与“H”两边的记数值都为 0xFA，等于十进制 250。即一个周期内计数 500 次，所以输出波形的频率应为：

$$2\text{MHz} \div 500 \text{ count} = 4\text{kHz}$$

表格 4-2 count 值与脉冲波频率的关系（机器频率为 2MHz 的时候）

1 周期内的 count 值 (L+H)	脉冲波的频率
125 count	16 kHz
250 count	8 kHz
500 count	4 kHz
1000 count	2 kHz

另有，当用 Accemic 启动程序并按下运行键后，蜂鸣器便开始发出声音，但当你按下 STOP 键后，蜂鸣器仍会继续发出声音，这是因为即使按下 STOP 键以后，虽然 CPU 停止运行，但是 PPG timer 仍处在工作状态，要停止蜂鸣器请按 reset 键，使单片机 reset 就可以了。

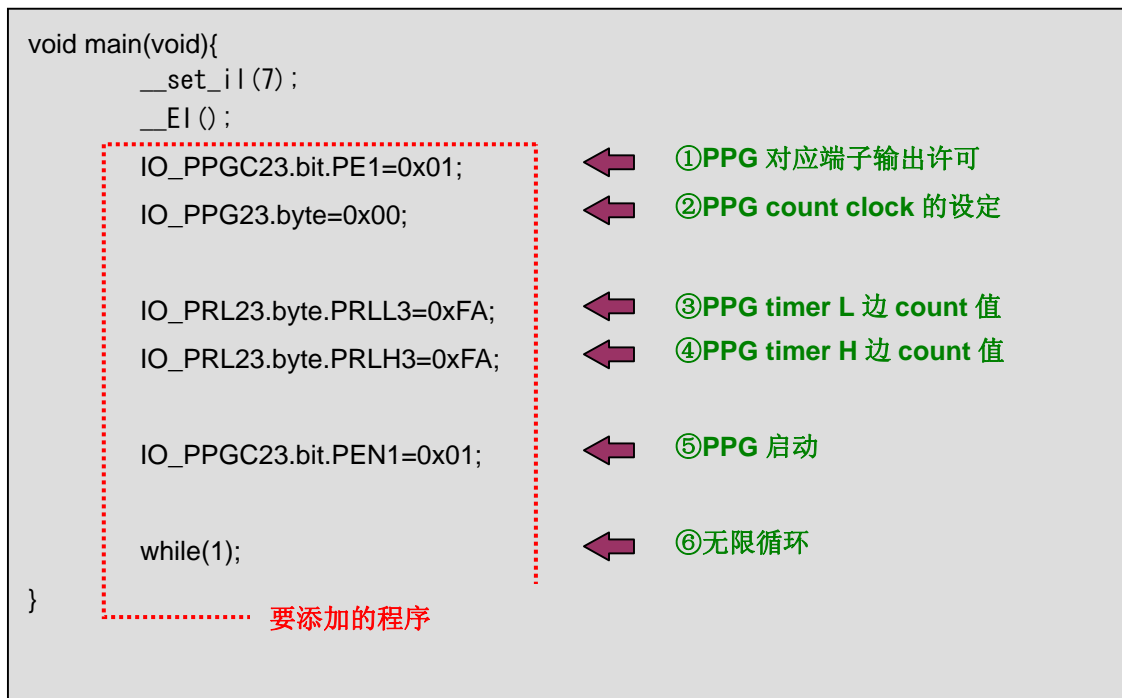


图 4-7 蜂鸣器程序

说明：在上述源码中有诸如“IO_XXX.byte”或者“IO_YYY.bit.YY”之类形式的变量，其实这些都为了程序员方便标记而在头文件里被事先预定义好的形式。

比如：IO_PPG23.byte → 就是 PPG 的值

IO_PPGC23.bit.PE1 → 就是 PPGC 中某个 bit 位的值

具体请参照、「附录 B」。

4.4.3 改变蜂鸣器的音色

综上所述，PPG timer 输出的脉冲波的频率就是蜂鸣器振动的频率，因此你可以随意调整蜂鸣器震动的频率来发出各式各样的声音。还记得中学课本里说人耳能分辨的声音范围位 20~20KHz，不如你用 PPG timer 来试试吧。另外，如果改变 count clock 的值，蜂鸣器的音色也会改变，试着分析一下原因吧。

5 利用中断来控制 LED

在第三章里我们介绍了通过判断端口上输入的电平状态来控制 LED 的方法，其实这种方法在程序的效率上并不可取。为了介绍单片机内一个至关重要的概念——中断，这章我们将通过“中断法”来对 LED 进行控制。

5.1 “中断”的概念

“中断”是单片机世界里一个非常重要的概念，几乎每本单片机的参考书里都要介绍这个概念。我们先用一个比较简单的例子作为开始。比如，让我们假设这样一个场面：你正在书桌前看书，这时候电话响了……此时，“你”就好比 CPU，“看书”是你本来正在进行的任务，电话铃成了“中断请求”，当然，你可以选择继续看书而无视电话铃，这就叫“屏蔽中断请求”，原任务继续执行；或者你起身去接电话，这就叫“中断请求的响应”，接电话的动作就相当于“中断任务”，原任务（看书）将被暂时挂起，等电话结束后原任务再从刚才停止的地方继续开始。在单片机的世界里当然没有“看书”或者“电话铃”这回事，但是“正在进行的任务”和“中断请求”是存在的。比如，单片机上 SW 的 ON/OFF 转换、通信数据的接收、计数器（定时器）溢出等，都可能造成“中断请求”。

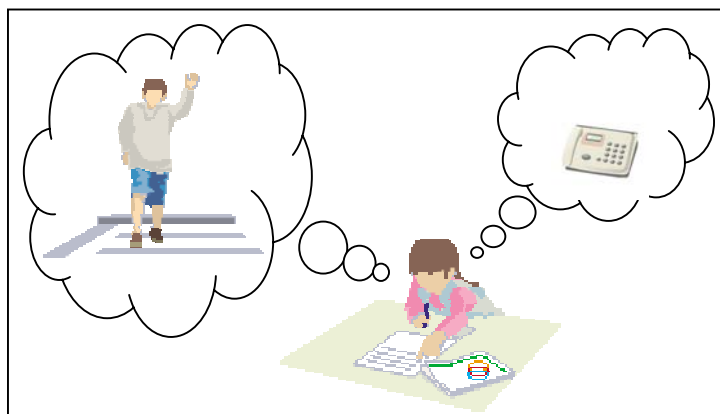


图 5-1 中断的发生

在嵌入式操作系统中，采用“中断法”要比采用“查询法”要有效率的的多。就拿开关 SW 来说，如果采用查询法，则 CPU 必须一直检测端口的状态，而没时间去执行其他任务；如果采用中断法，则只有当开关上的状态发生改变 CPU 才会响应中断服务程序（在中断服务程序里我们可以完成点亮 LED 的任务等），其余的时间 CPU 就可以进行其他任务的处理。

在第 3 章里我们介绍了用查询法来检测端口上输入电平状态的方法，这样做往往效率太低。若改用使用中断法的话，我们只需做一些中断控制寄存器的初始设定，然后就不必一直检测端口的状态，一旦端口上的状态达到我们初始设定的条件（比如端口上发生上跳延翻转等）程序将会主动跳转到中断服务程序内执行相关程序。这样 CPU 将会有更多空余时间来执行其他的任务。

说明：根据中断发生的地点，单片机内的中断大致有两种：内部中断和外部中断，内部中断主要是由单片机内部功能模块上产生的中断，比如定时器溢出中断，数模转换完成中断等，而外部中断是由外部条件引起的中断，比如端口上电平状态的变化或者端口上检测出预先设定的电平状态等，这里我们用的是外部中断！输入端口为 P25 和 P27。

注意：并不是所有端口都对应中断功能的，只有一些特殊的端口带有中断检测功能（具体请参考单片机的 hardware manual）

5.2 利用“中断”来检测 SW 的状态的方法

正如第 3 章所介绍的，评测板上共有 2 各开关 SW，分别接在 P25 和 P27 上。如果用查询法的话，我们只需将 P25 和 P27 设定为输入端口，然后读出 PDR 寄存器的值，便可知道端口上电平的状态。其实，除此之外 P25 和 P27 同时也是外部中断的输入端口，只要同样将其设定为输方向（DDR=0）再配合外部中断寄存器的设定，则外部条件一旦满足事先的设定值便会马上触发中断请求（INT5 及 INT7）。下面，将具体解释外部中断（INT5 及 INT7）的使用方法。

图 5-2 显示了评测板上 SW 的连接示意图。可以看到 SW1 与单片机上外部中断输入端口 INT5 相连。SW1 断开时（OFF）INT5 端子上输入高电平 Vcc（5V），闭合时（ON）INT5 端子上输入低电平 GND（0V）。因此，当按下 SW1 的瞬间 INT5 上将发生由 High \Rightarrow Low 电平状态的变化（即下跳延）。相反，放开 SW1 的瞬间 INT5 上将发生由 Low \Rightarrow High 电平状态的变化（即上跳延）。所以，如果利用单片机上外部中断 INT5 的功能，经过事先设定，则通过 P25 端子上的状态变化便可引起外部中断的请求。换句话说，我们可以用中断来得知 P25 端口上的状态及其变化，而不用一直去查询它。此外，SW2 与 SW1 也是同样的道理，只不过所示用的外部中断为 INT7，所连接的端口为 P27。因此，SW2 所产生的外部中断为 INT7。

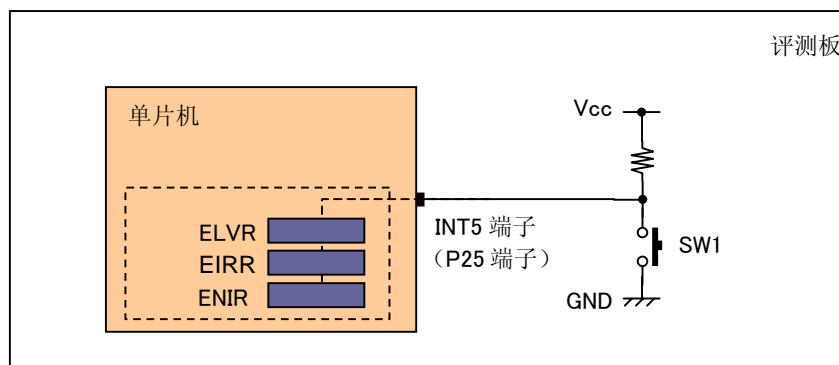


图 5-2 SW 与单片机端子的连接图

下面，将说明单片机内外部中断 INT5 的使用方法。要使用 INT5 外部中断必须先将对应端口的传输方向设定为“输入”，由于 INT5 的对应端口为 P25，所以需将 DDR2 设定为“输

入”方向。即将 DDR2 的第 5 位设置为“0”。DDR2 的设定方法请参照第 3 章的说明。当然，要使用外部中断 INT5 还必须事先对外部中断的寄存器做一些设定，比如 EIRR 寄存器、ENIR 及 ELVR 寄存器等。EIRR 是用来表示外部中断请求标志的 8 位寄存器，如果 INT5 上发生中断请求，则 EIRR 的第 5 位就会自动置 1；ENIR 是用来禁止或允许某个外部中断功能的 8 位寄存器，如果将其第 5 位置为 0，则 INT5 外部中断功能将被禁止；另外，ELVR 是用来设定外部中断检测条件的 16 位寄存器，可供选择的条件有：低电平检出、高电平检出、上跳延变化、下跳延变化 4 种。

这里我们把 SW1 从 OFF \Rightarrow ON 作为外部中断的引发条件，即按下 SW1 便引发中断请求，由于 SW1 按下后 INT5 端子上的电平由 High \Rightarrow Low 变化，即产生下跳延变化。要使单片机能检出下跳延变化，必须按照以下 4 步来设定相关的寄存器。这样，单片机在 SW1 按下的时候便会产生中断请求了。

将 ENIR 的第 5 位置 0，先禁止 INT5 的中断功能，以做初始设定；

ELVR 的第 10、11 位都置 1（设定外部中断 INT5 检出条件为下跳延变化）；

将 EIRR 第 5 位置 0，清除原有的中断请求；

将 ENIR 的第 5 位置 1，允许 INT5 的中断功能。

5.3 通过 SW 控制 LED 的程序（中断法）

下面我们将制作一个利用中断法的程序，其功能是通过 SW1 控制 LED 的亮与灭。

5.3.1 程序概要

为了更好地解释中断这个概念，现在来实际制作一个利用中断法的程序。程序的内容与第 2 章的程序很相似，但是这里对于 SW 状态的处理用的是中断法，并且增加了相应的中断处理程序，程序主要实现的目的如下所示，程序的流程图如图 5-3 所示。

程序启动后，LED1 的初始状态为熄灭。

SW1 按下后 LED3 变亮。

此后每按一次 SW1，LED3 将以亮、灭状态交替变换。

程序的主要流程是这样的：首先，对 LED 输出用的端子以及 SW 输入用的端子进行初始设定，设定方法与第 3 章相同。然后对外部中断的相关寄存器做设定，并且允许外部中断功能运行。最后，进入无限循环。在程序运行过程中每当 P25 端口上发生下跳延变化，则会引起 INT5 的外部中断请求，由于 INT5 的中断优先度高于主程序 main（）（关于中断优先度可以通过 ICR 寄存器来设定），所以程序将跳转到中断服务程序__interrupt void ext_int(void)中执行，在中断服务程序中首先必须清除中断请求位 EIRR，然后将 LED1 的状态进行反转。不难发现，与第 2 章的程序相比，LED1 状态反转的处理程序被收缩到了中断

服务程序中，并且只有在中断发生时才被执行，这样 CPU 便可以利用多余的时间进行其他的操作了。

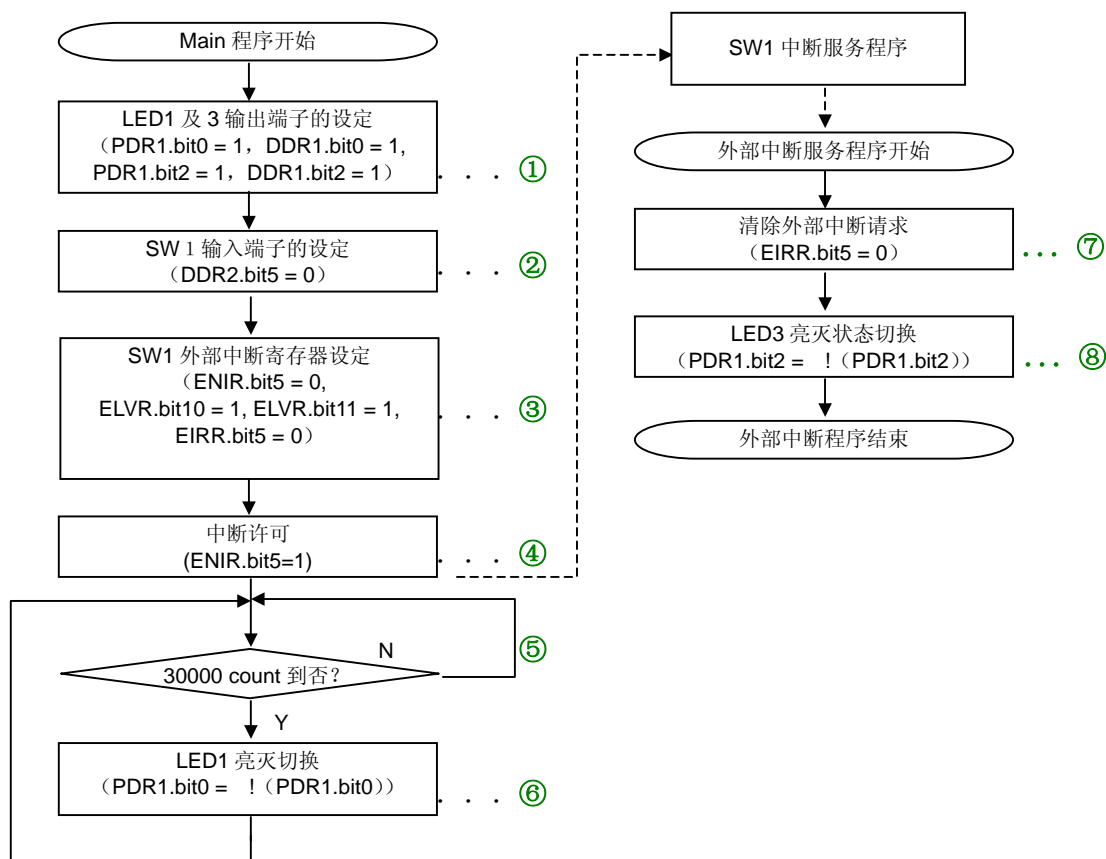


图 5-3 程序流程图

关于程序的一点补充说明：

1,在中断服务程序中请一定要清除中断请求寄存器 **EIRR**，否则程序将无法自动退出中断服务程序，而始终停留在中断服务程序里面。

2,程序中有一段对于 **ICR06** 的设定：**IO_ICR06.byte=0x00**，这其实是对中断的优先度进行的设定（**ICRxx** 是专门用来设定中断优先度的），而外部中断 **INT5** 对应的优先度寄存器正为 **ICR06**，我们将其设定为 **00**（即最高优先度）。此外，主程序 **main（）** 中用一句预定义的语句 **__set_il(7)** 声明了主程序的优先度为 **7**（即最低优先度）。一般中断服务程序按照用户的需要进行优先度的排列，然后通过所对应的 **ICRxx** 进行设定,当高优先度的中断服务正在执行时，低优先度的中断服务程序将被屏蔽。

5.3.2 程序的制作与执行

首先，请参照「附录 A.1」打开“sample.prj”内的源文件“main.c”，然后将图 5-4 及图 5-5 虚线部分内的内容输入到主程序内。注意：除了 main.c 以外，Project 内的其他文件请不要

做任何修改。当添加完毕后，再按照「付録 A.2」所示的顺序对源程序进行“build”。如果出现编译错误，请核对图 5-4 及图 5-5 检查添加的内容里是否有文字错误。“build”顺利完成后“ACCEMIC MDE”将自动启动。

接着，转向“ACCEMIC MDE”界面，对编译好的程序进行 debug（调试），并实际测试运行情况。程序的执行方法请参照「附录 A.3」。如果程序顺利启动，则 LED1 应该以一定的时间间隔闪烁，然后每次按下 SW1，LED3（红色）将会做亮灭切换，如果这两个部分能顺利实现，则说明程序的内容准确无误。

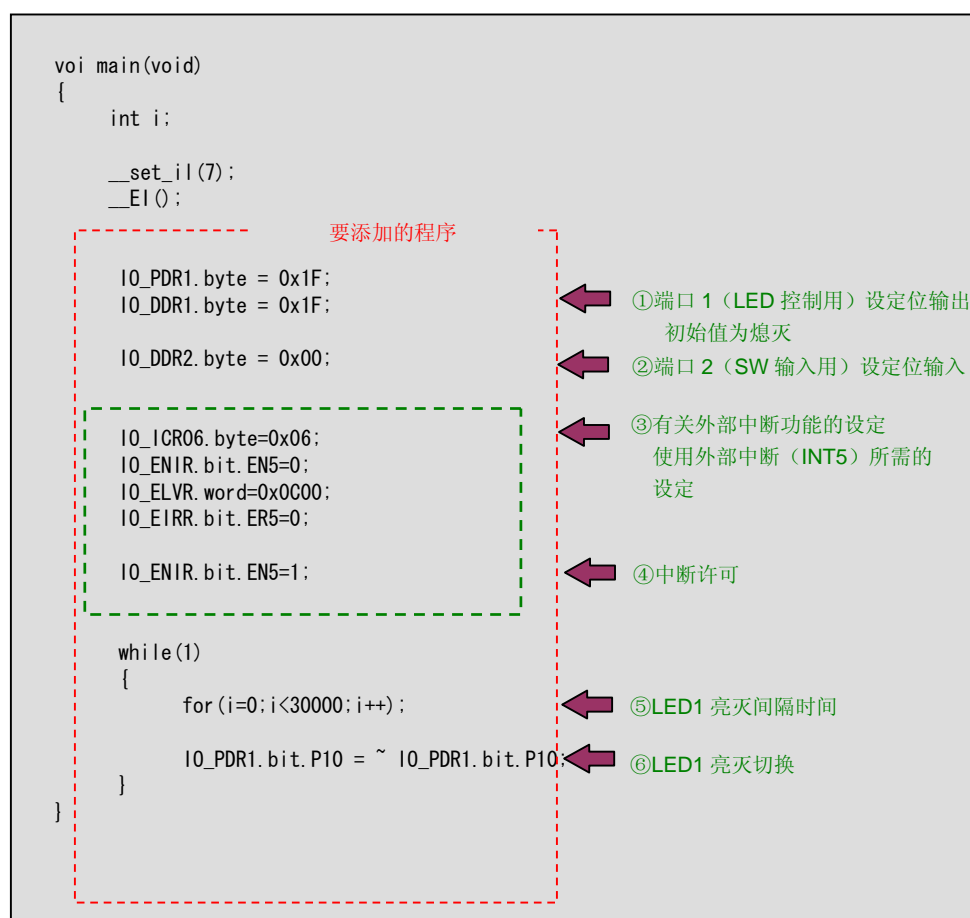


图 5-4 主程序源码

```

__interrupt void ext_int(void)
{
    IO_EIRR.bit.ER5=0x0;
    IO_PDR1.bit.P12=~IO_PDR1.bit.P12;
}

```

要添加的程序

⑦清除中断请求
 ⑧LED 亮灭切换处理

图 5-5 中断服务程序源码

另外，`ext_int` 函数是中断函数，所以要加上中断类型修饰 `__interrupt`，它只有在外部的中断请求 `INT5` 发生时才被调用。正如前面所说的，单片机内部有很多中断机能，对应不同的中断应有不同的中断服务函数，为了明确告诉程序某个中断发生时应该调用哪个中断服务程序，通常在主程序中需声明一个 **vector** 表（向量表），用来定义中断函数所对应的向量号。而本程序中已经为用户预定义好了 `INT5` 中断程序的向量号，因此不必再做定义。（注意：向量号的定义要用到中断程序名，故请不要更改原中断程序的程序名 `ext_int (void)`）

说明：在上述源码中有诸如“`IO_XXX.byte`”或者“`IO_YYY.bit.YY`”之类形式的变量，其实这些都为了程序员方便标记而在头文件里被事先预定义好的形式。

比如：`IO_EIRR.bit.ER5` → 就是 `EIRR` 的第 5 位。

具体请参照、「附录 B」。

6 利用 timer（定时器）来使 LED 闪烁

关于 LED 的控制方法我们在第 2 章的 2.5 节已经做了介绍,但是除了这种方法以外其实还有许多种控制 LED 的方法,比如为了达到更精确的时间控制(每隔 1 秒闪烁一次等等),我们就可以用 timer(定时器)来控制 LED 的闪烁。本章就将介绍利用 16 位 Reload timer(16 位定时器)来控制 LED 闪烁的方法。

6.1 什么叫 timer(定时器)

提到 timer,翻译成中文就是定时器(以下用作 timer)。在我们的身边其实就有很多定时器,比如 VCD 机里的定时器,手机里的定时器等等。单片机里的 timer 的起的就是计时的作用。我们再拿闹钟作个例子,每天早上闹钟一响我们便起床,如果没有闹钟,便无法提供一个标准的周期,我们的生活便会失去规律。单片机内部也是一样,在需要精确时间控制的场合我们必定会用到 timer。

如果把单片机的 timer 理解为闹钟,那么“闹钟闹铃”就是 timer 的“中断”。下面我们就将利用“timer 中断”来控制 LED 的闪烁(关于“中断”的概念请参见第 5 章)。Timer 在这里的作用就是每隔一定的时间便发出一个“中断请求”,这个时间间隔取决于你事先设定好的一个记数值,而在中断服务程序中我们可以反转 LED 的状态。也许有人会问用定时器中断的好处在哪,请看下面的解释:在 2.5 节我们也介绍了一个使 LED 闪烁的程序,但是在整个程序中 CPU 一直在不断地计数 LED 点亮状态的时间与熄灭状态的时间而无法完成其他的任务,这无疑降低了 CPU 的工作效率。现在,我们把“计数”这个工作交给 timer 来做,这样 CPU 就不必亲自去计数,当 timer 达到一定的值以后便会向 CPU 发出“中断申请”,这时 CPU 便可以放下“手头”的工作去相应这个“中断申请”。这个方法无疑大大提高了 CPU 的工作效率。图 6-1 是未使用中断时的 CPU 工作状态。

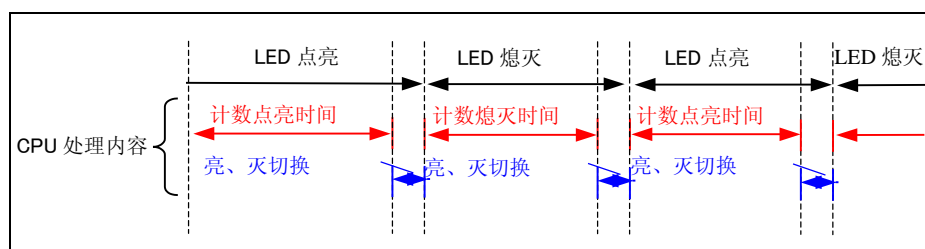


图 6-1 未使用 timer 中断的 CPU 工作状态

如果利用了 timer,点亮时间与熄灭时间的计数就可以交给 timer, CPU 的工作状态如图 6-2 所示。

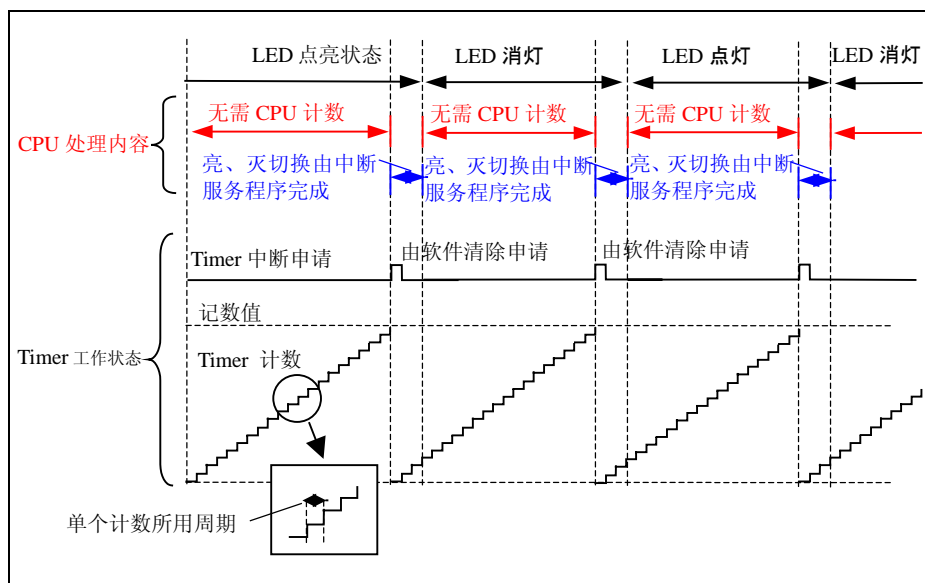


图 6-2 使用 timer 中断的 CPU 工作状态

为了让 Timer 精确完成 LED 亮灭时间的计数，我们需对其做一些初始设定。LED 亮灭的时间间隔（即 Timer 的计数时间）可以由下式求得：

$$\text{LED 亮灭时间间隔} = \text{单个计数时钟周期} \times \text{记数值}$$

这样，Timer 便可以提供精确的时间间隔。

6.2 通过 Timer 中断控制 LED 闪烁的程序

这里，我们将实际制作一个通过 Timer 中断控制 LED 闪烁的程序。我们所用的是 16 位的 Reload Timer。

6.2.1 程序概要

在第 2 章里介绍了控制 LED 闪烁的程序，其 LED 的延迟工作其实是由一个 for 循环完成的，而这里我们把这个延时工作交给 Timer 来完成，但就 LED 的动作来讲并没有很大的差别，但是从程序设计架构来讲有着本质的不同！程序流程图如图 6-3 所示



图 6-3 程序流程图

在上面这个程序中，我们先做了一些初始设定，然后主程序便进入无限循环（12 部分），但是由于使用了中断法，在实际应用中 CPU 可以在这个无限循环中做许多其他的处理。这里使用的 16 位 Reload Timer 的相关寄存器如表格 6-1 所示。这里我们设定时间间隔为 1 秒。因此 Timer 的记数值可由下式求得：

$$\begin{aligned} \text{LED 亮灭间隔时间} &= \text{单个计数时钟}(16\mu\text{s}) \times \text{记数值}(62500) \\ &= 1\text{s} \end{aligned}$$

表格 6-1 16 位 Reload Timer 寄存器设定

设定项目（位数）	寄存器名	设定值（设定的含义）
	设定位	
记数值（16 位）	TMRLR 寄存器	0xF424 （62500 次）
	D0~D15 (bit0~15)	
单个计数时钟（2 位）	TMCSR0 寄存器	10(单个计数时钟：16 μ s)※
	CSL0, CSL1(bit10,11)	
Timer 重复模式（1 位）	TMCSR0 寄存器	0(代表只循环一次，中断完成便结束动作) 1(无数此循环，中断完成后重新读入记数值进行计数)
	RELD (bit4)	
Timer 中断输出许可（1 位）	TMCSR0 寄存器	0(允许 Timer 的中断功能)
	INTE (bit3)	1(禁止 Timer 的中断功能)
Timer 中断申请位（1 位）	TMCSR0 寄存器	0(Timer 无中断申请)
	UF (bit2)	1(Timer 有中断申请)
Timer 动作许可（1 位）	TMCSR0 寄存器	0（禁止 Timer 动作）
	CNTE (bit1)	1（允许 Timer 动作）
Timer 启动位（1 位）	TMCSR0 寄存器	0（无任何影响）
	TRG (bit0)	1（启动 Timer 开始计数）

※ 单个计数时钟周期(16 μ s) = 32 / 内部机器周期（2MHz）

6.2.2 程序的制作与运行

接着让我们实际来写这个程序吧。首先，请参照「附录 A.1」打开“sample.prj”内的源文件“main.c”，然后将图 6-4 及图 6-5 虚线部分内的内容输入到主程序内。注意：除了 main.c 以外，Project 内的其他文件请不要做任何修改。当添加完毕后，再按照「附录 A.2」所示的顺序对源程序进行“build”。如果出现编译错误，请核对图 6-4 及图 6-5 检查添加的内容里是否有文字错误。“build”顺利完成后“ACCEMIC MDE”将自动启动。

接着，转向“ACCEMIC MDE”界面，对编译好的程序进行 debug（调试），并实际测试运行情况。程序的执行方法请参照「附录 A.3」。如果程序顺利启动，则 LED5 应该以 1 秒的时间间隔闪烁。



图 6-4 程序例 (main 函数)

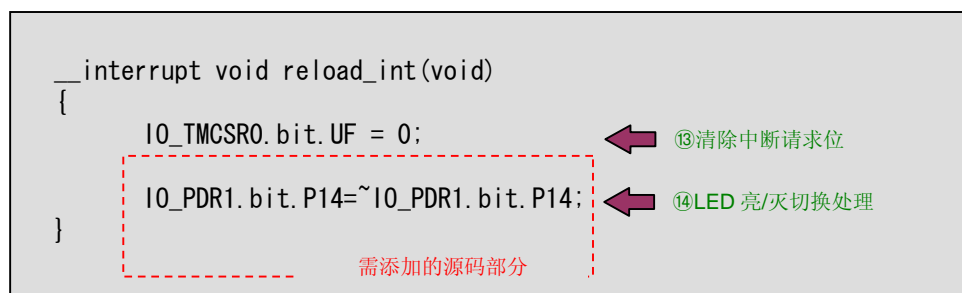


图 6-5 程序例 (中断服务程序)

另外，图 6-5 的 reload_int 函数是 16 位 Reload Timer 的中断服务程序，所以要加上中断类型修饰 __interrupt，它只有在 Reload Timer 发生中断（记数发生溢出）时才被自动调用，正如前面所说的，单片机内部有很多中断机能，对应不同的中断应有不同的中断服务函数，为了明确告诉程序某个中断发生时应该调用哪个中断服务程序，通常在主程序中需声明一个 vector 表（向量表），用来定义中断函数所对应的向量号。而本程序中已经为用户预定义好了 INT5 中断程序的向量号，因此不必再做定义。（注意：向量号的定义要用到中断程序名，故请不要更改原中断程序的程序名 ext_int (void)。

说明：在上述源码中有诸如“IO_XXX.byte”或者“IO_YYY.bit.YY”之类形式的变量，其实这些都为了程序员方便标记而在头文件里被事先预定义好的形式。

比如：IO_EIRR.bit.ER5 → 就是 EIRR 的第 5 位。

具体请参照、「附录 B」。

7 如何使用 A/D（模/数）转换器

这里，我们将介绍使用单片机将外部的模拟信号通过 A/D 转换器（逐次比较型）转换成数字信号的方法。

此评测板上带有一个滑动变阻器，通过这个滑动变阻器可以改变滑动端的输出电压，而这个变化的变压值便是 A/D 转换器的模拟输入信号。接下去我们就将介绍如何将这个模拟信号转变成数字信号。

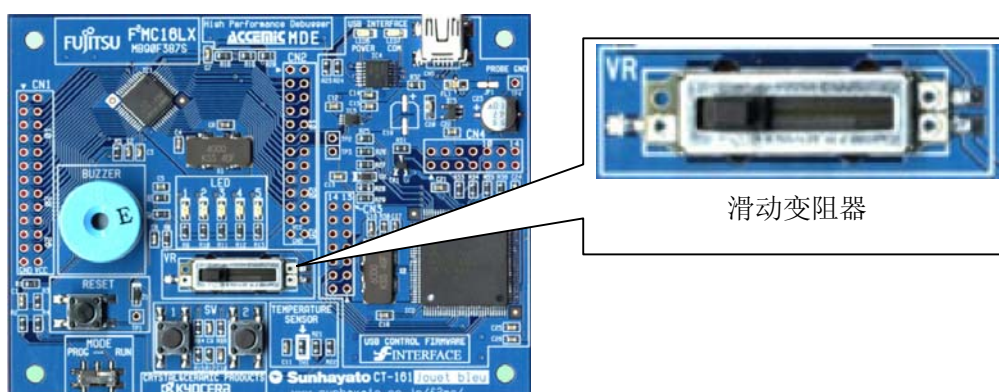


图 7-1 滑动变阻器

7.1 模拟信号与数字信号

你知道模拟信号和数字信号的区别吗？按照字典上的解释是这样的：

表格 7-1 模拟与数字信号的字典解释

	字典解释
模拟信号	是一种连续可变的信号
数字信号	是一种离散的信号

光看字典上的解释可能还很难从直观上理解这两者的分别，简单地说，在这里可变的电压值就是模拟信号，而经过 A/D 转换后就变成了数字信号。原来的信号在时间和幅度上都是连续变化的，而经过变换以后则成了离散的数字信号。

图 7-2 显示了一个比较简单的数字信号与模拟信号的关系。蓝色的部分是模拟信号，而黑色的部分就是相应的数字信号。

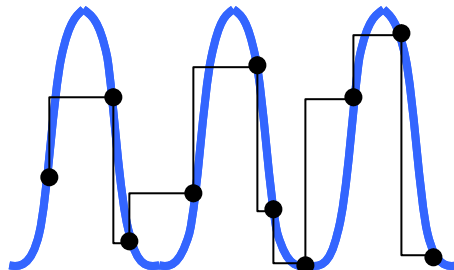


图 7-2 模拟与数字信号

7.1.1 A/D 转换器的概要

将模拟信号按照一定的变换法则进行模/数转换就可以得到相应的数字信号。而带有这个转换功能的就是单片机内部的 A/D 转换器。在这里有一个很重要的概念就是分辨率。所谓分辨率就是指转换器所能分辨最小量化信号的能力。简单地说 A/D 转换器的分辨率就像照片的解析率，解析率越高照片就越清晰。同样，分辨率越高就越能够转换较精确的模拟信号。例如：对于同一个 10V 的模拟信号，一个具有 8 位分辨率的转换器最小可达到的精度为 $\frac{10}{2^8}V$ ，而一个具有 10 位分辨率的转换器最小则可以达到 $\frac{10}{2^{10}}V$ 的精度，相比前者，后者具有更高的精度。

此评测板上搭载有一个具有 10 位分辨率的 A/D 转换器（也可设定为 8 位分辨率）。10 位分辨率模式时的精度为 $2^{10}=1024$ ；8 位分辨率模式时的精度为 $2^8=256$ 。下面我们以 5V 为例，具体说明分辨率的概念：

模拟信号(5V)

10 位分辨率时精度为： $5V/1024 \approx 0.00488V$

8 位分辨率时精度为： $5V/256 \approx 0.01953V$

7.1.2 滑动变阻器

图 7-3 为滑动变阻器的电路符号。它以图 7-4 的连接方式接在此评测板上。其可变端上的变化电压将成为 A/D 转换器的模拟输入。



图 7-3 滑动变阻器

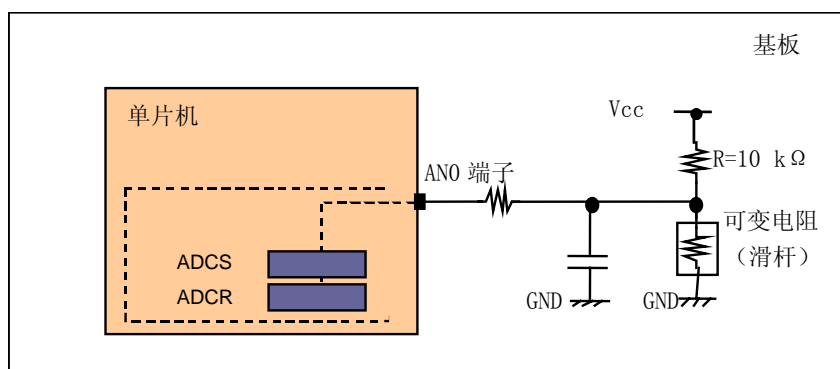


图 7-4 滑动变阻器相关电路

7.2 制作一个表示电压数值的程序

接着，让我们来实际写一个可以表示输入电压大小的程序。利用 A/D 转换器将输入的模拟信号变化为数字信号，然后来驱动 LED 发光。

7.2.1 程序概要

这个程序主要的任务是把外部的模拟信号变化为数字信号，并随之产生大小变化，然后利用这个变化的数字信号控制发光 LED 的个数。程序的主要步骤归纳如下，程序流程图如图

图 7-5 所示。

- 1、初始化设定（AD 转换器的初始化设定）。
- 2、AD 采样开始（AD 转换器先采样输入电压大小）。
- 3、AD 转换完成（产生 AD 转换值）。
- 4、根据转换后的 AD 值来控制发光 LED 的个数。
- 5、之后重复（2）～（4）的处理过程。

程序中，先要设定 LED 用的输出端子以及 AD 转换器所用的输入端子。然后，需对 AD 变换所用的采样时间、动作模式、AD 输入信道（频道）等 AD 转换器相关的部分进行设定。这里我们采用的采样时间为 $128/\phi$ ，比较时间为 $176/\phi$ ，分辨率为 8bit，AD 信道（频道）用的是 Ch0，即 0 频道。重复模式为“连续变换模式”。 ϕ 指的是单片机内部的总线频率。另外，关于以上这些设定值所要用的寄存器如下表所示：

表格 7-2 AD 转换器的相关寄存器

名称	寄存器 (bit)	设定值
AD 转换开始	ADCS:H(bit9)	1 (AD 变换开始)
AD 中断请求允许	ADCS:H(bit13)	1 (允许), 0 (禁止)
AD 中断请求位	ADCS:H(bit14)	0 (清除中断请求位)
AD 转换停止	ADCS:H(bit15)	0 (AD 强制结束)
AD 转换结束频道	ADCS:L(bit0-2)	0 (Ch0 选择 0 频道)
AD 转换开始频道	ADCS:L(bit3-5)	0 (Ch0 选择 0 频道)
AD 变换重复模式	ADCS:L(bit6-7)	2 (连续变换模式)
AD 比较时间设定	ADCR:H(bit11-12)	3 ($176 / \phi$)
AD 采样时间设定	ADCR:H(bit13-14)	3 ($128 / \phi$)
AD 分辨率设定	ADCR:H(bit15)	1 (8bit 分辨率)
模拟信号输入许可	ADER(bit0-7)	01h (只有 Ch0 允许输入)

说明：ADCS 及 ADCR 等寄存器都是 16 位的寄存器，所以他们分为高位部分和低位部分，上表中分别以“H”和“L”所示。“Bit”代表这个寄存器中的第几位。

采样时间是指 AD 转换器采样外部模拟信号所用的时间。

比较时间是指 AD 转换器将采样到的电压与 AD 基准电压比较所需的时间。

本产品搭载的单片机共有 8 个频道的 AD 转换器，分别对应 8 个输入端子，“转换开始频道”是指当开始 AD 转换动作的时候首先从哪个频道（端子）上采样数据，而“转换结束频道”则是指一轮变换中最后被采样的频道（端子）。一轮转换就是指从转换开始频道到转换结束频道的一个周期。

重复模式是指，一轮转换结束后是不断继续重复上一次的采样动作，还是就此停止。当采用连续模式时，AD 转换将从开始频道至结束频道周而复始的采样并比较，知道人为的。






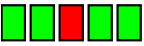
当一次采样结束后，AD 转换器会将比较后得到的数字数据存入一个叫 IO_ADCRLH.DATA8 的寄存器，并向 CPU 输出一个中断请求，CPU 响此中断请求，并在中断服务程序中将其 AD 值读出。

这样就基本上完成了 AD 的设定，接着就可以开始 AD 转换了。参照表格 7-2 的设定，不难发现此时 AD 的动作模式如下：

AD 启动（AD 变换开始） ⇒ ch0 转换结束 ⇒ ch0 转换开始 …

如此周而复始。这里需要提醒的时，在 AD 中断服务程序里一定要先清除 AD 中断请求位，否则程序将一直停留在中断服务程序中无法跳出。清除了 AD 中断请求位后便可以读出 AD 变换值，根据此值的大小来调整发光 LED 的个数。变换值与发光 LED 个数的关系可见下表：

表格 7-3 模拟电压值与发光 LED 个数的关系

模拟电压	AD 变换值（十进）	LED 个数
0~0.83 V	0 ~ 42	 全灭
0.83~1.67 V	43~ 84	 LED1 亮
1.67~2.50 V	85 ~ 127	 LED1~2 亮
2.50~3.33 V	128 ~ 170	 LED1~3 亮
3.33~4.17 V	171 ~ 212	 LED1~4 亮
4.17~5.00 V	213 ~ 255	 全亮

说明：评测板上 AD 参考电压接的是 5V，所以当输入电压为 5V 时，AD 变化值达最大（255），输入电压为 0V 时，变换值也为 0，他们之间的关系是线性变化的。如下式：

$$\frac{\text{输入电压}}{\text{参考电压}5V} = \frac{\text{AD变换值}}{255}$$

其中，参考电压 5V 是固定值，255 也是固定值。

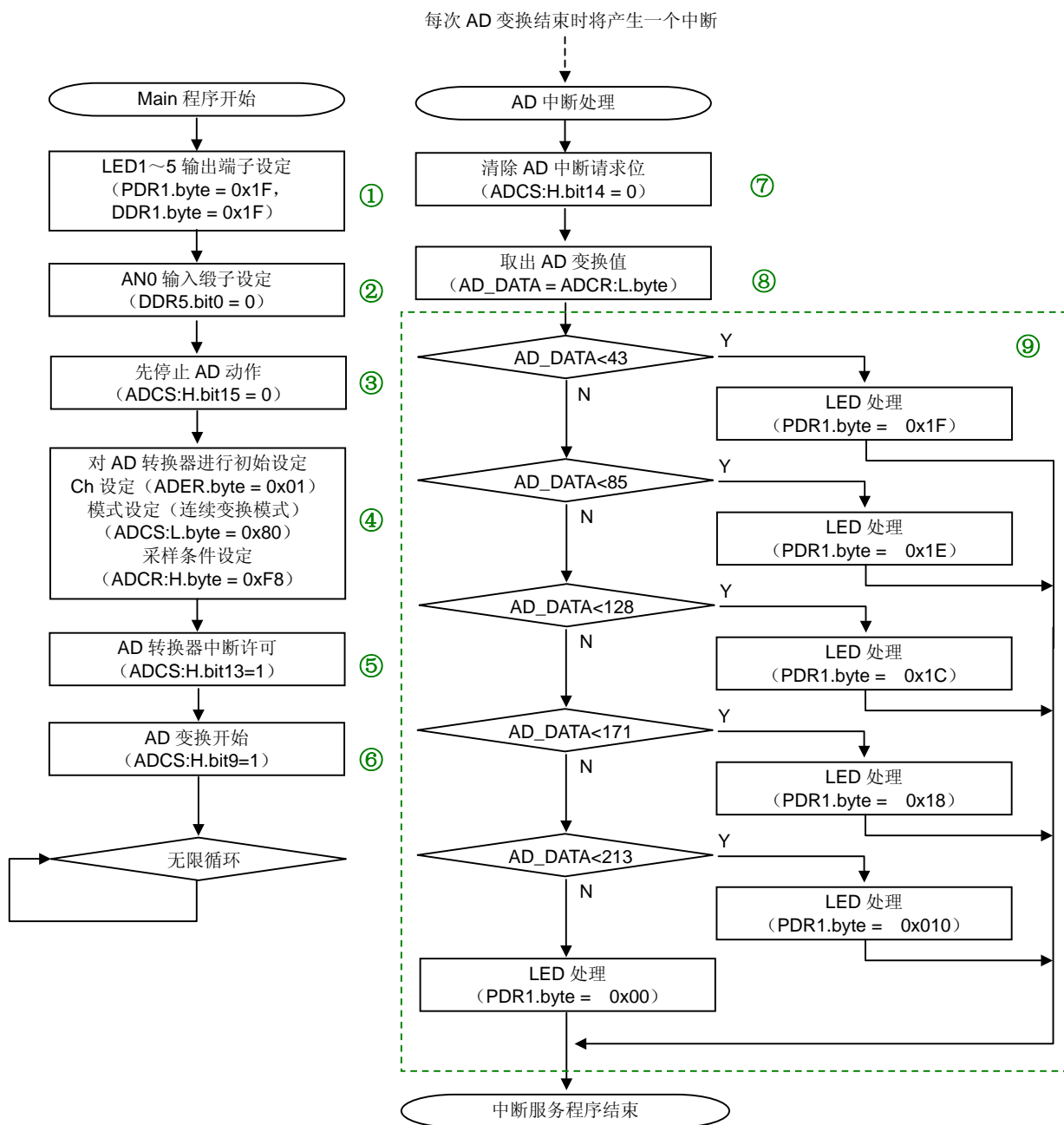


图 7-5 程序流程图

7.2.2 程序的制作与执行

接着让我们实际来写这个程序吧。首先，请参照「附录 A.1」打开“sample.prj”内的源文件“main.c”，然后将图 7-6 及图 7-7 虚线部分内的内容输入到主程序内。注意：除了 main.c 以外，Project 内的其他文件请不要做任何修改。当添加完毕后，再按照「付録 A.2」所示的顺序对源程序进行“build”。如果出现编译错误，请核对图 7-6 及图 7-7 检查添加的内容里是否有文字错误。“build”顺利完成后“ACCEMIC MDE”将自动启动。

接着，转向“ACCEMIC MDE”界面，对编译好的程序进行 debug（调试），并实际测试运行情况。程序的执行方法请参照「附录 A.3」。如果程序顺利启动，您可以调节滑动变阻器来测试 LED 的发光情况。

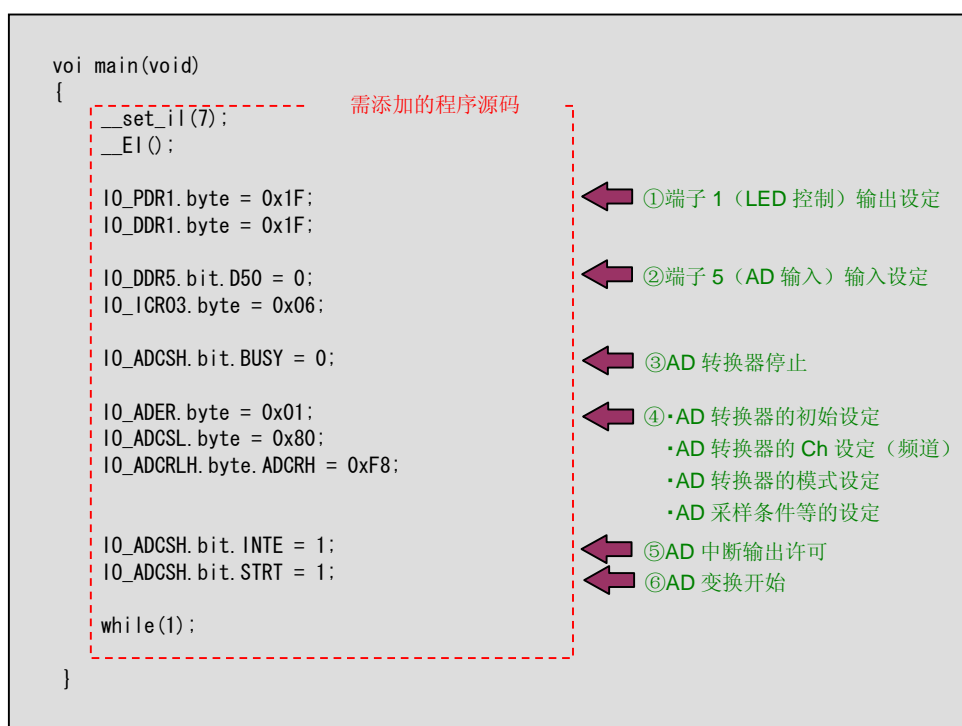


图 7-6 程序例 (main 程序)

```

__interrupt void ADC_int(void)
{
    // 需添加的程序源码
    unsigned char AD_DATA;

    IO_ADCSH.bit.INT = 0;
    AD_DATA = IO_ADCRLH.DATA8;

    if(AD_DATA < 43) {
        IO_PDR1.byte = 0x1F;
    }
    else if(AD_DATA < 85) {
        IO_PDR1.byte = 0x1E;
    }
    else if(AD_DATA < 128) {
        IO_PDR1.byte = 0x1C;
    }
    else if(AD_DATA < 171) {
        IO_PDR1.byte = 0x18;
    }
    else if(AD_DATA < 213) {
        IO_PDR1.byte = 0x10;
    }
    else {
        IO_PDR1.byte = 0x00;
    }
}

```




 ⑦清除中断请求位
 ⑧取出 AD 变换值
 ⑨根据 AD 变换值控制 LED

图 7-7 程序例（中断服务程序）

另外，图 7-7 中的 ADC_int(void)函数是 AD 转换器的中断服务程序，所以要加上中断类型修饰__interrupt，它只有在 AD 发生中断（一次 AD 转换完成）时才被自动调用，正如前面所说的，单片机内部有很多中断机能，对应不同的中断应有不同的中断服务函数，为了明确告诉程序某个中断发生时应该调用哪个中断服务程序，通常在主程序中需声明一个 vector 表（向量表），用来定义中断函数所对应的向量号。而本程序中已经为用户预定义好了 AD 中断程序的向量号，因此不必再做定义。（注意：向量号的定义要用到中断程序名，故请不要更改原中断程序的程序名 ADC_int(void)。

说明：在上述源码中有诸如“IO_XXX.byte”或者“IO_YYY.bit.YY”之类形式的变量，其实这些都为了程序员方便标记而在头文件里被事先预定义好的形式。

比如：IO_ADCSH.bit.INTE → 就是 ADCS 的高位中的 INTE 位。

具体请参照、「附录 B」。

8 如何使用温度传感器

在很多应用单片机的系统中往往搭载有很多传感器，他们的作用就是采样外部的数据，比如温度、速度、压力等等，然后传给单片机以供处理。传感器种类繁多，其中有一种就是温度传感器，温度传感器顾名思义就是感知温度的传感器，他的应用范围也相当广泛，比如冰箱等家电产品。

此评测板上也搭载有一个简易的温度传感器，本章我们就将介绍这个温度传感器的使用方法。

8.1 关于温度传感器

温度传感器即使用来检知温度变化的传感器。用来测定温度的方法有很多种，比如有用水银的，也有用非接触的红外线的。表格 8-1 中归纳集中常见的温度传感器。

表格 8-1 温度测量的方法

类别	方法・特点
水银温度计	用水银或酒精的热膨胀原理来测量温度或者体温
热电偶	将两种不同材料的导体或者半导体焊接成一个回路，当焊接点处产生温差时就会产生一个电动势，形成一定的电流，称为热点效应，热电偶就是利用这个热电效应工作的
温度传感器	利用半导体的温度特性制作的温度传感器，是使用最广泛的一种
红外线温度计	利用红外线测定温度，最大的特点是非接触式测量

此评测板上所采用的温度传感器又叫 **thermistor**，他的电阻可随温度变化而变化，也称热敏电阻。型号是 TDK 的 **NTCthermistor**(**NTCG164BH103**)。它具有负的热敏系数(**Negative Temperature Coefficient**)，温度升高电阻将会下降。接着，我们介绍它的具体使用方法。

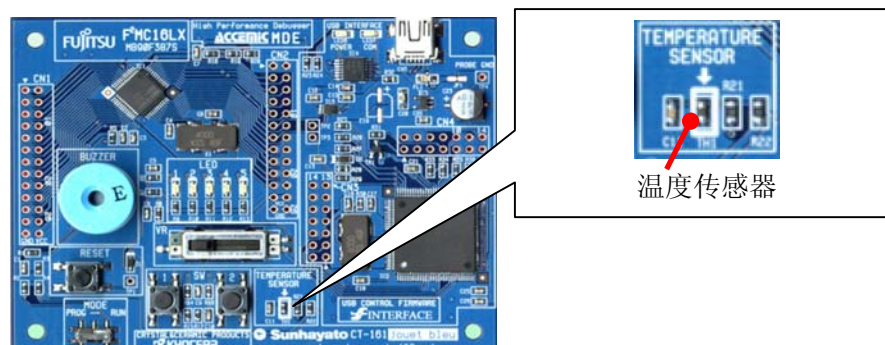


图 8-1 热敏电阻 (NTCG164BH103)

8.2 温度传感器的使用方法

要使用温度传感器采样温度数据，必须要了解所使用的温度传感器的特性，一般情况下，当我们知道了产品的型号后（NTCG164BH103）可以直接到生产厂家的网站上下载产品相关的数据手册，现在，我们将所需的资料整理在下表内，它表明了温度与电阻值之间的关系
温度范围为 5~50℃：

表格 8-2 NTCG164BH103 温度与阻值的关系

温度[℃]	阻值[kΩ]	温度[℃]	阻值[kΩ]
5	26.250	30	7.997
10	20.390	35	6.437
15	15.960	40	5.213
20	12.590	45	4.248
25	10.000	50	3.481

图 8-2 是温度传感器的连接示意图。当温度变化引起电阻变化时，热敏电阻两端的电压就将发生变化，然后通过 AD 转换器我们就可以采样这个模拟电压信号，这样就和上一章的滑动变阻器时一个原理了。图后列出了热敏电阻的阻值 R_{TH} 与其两端电压 V_{AN1} 的关系：

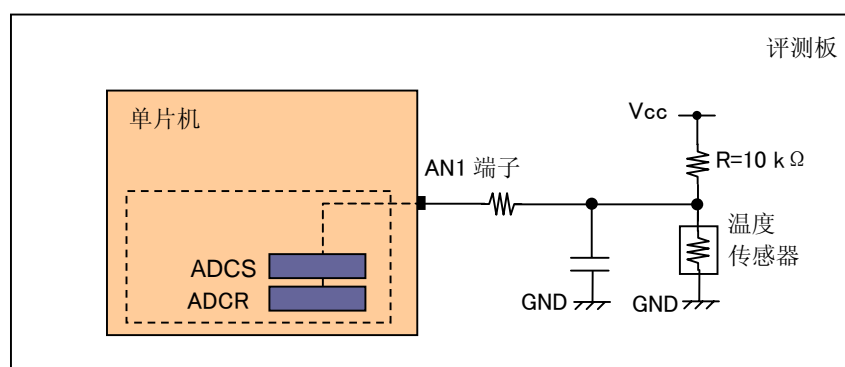


图 8-2 温度传感器的连接图

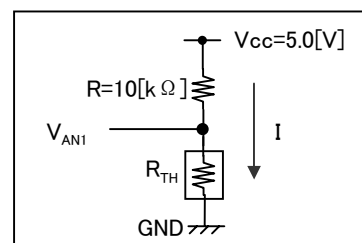
【AD 的输入电压】

总电流：

$$I = \frac{V_{CC}}{R + R_{TH}}$$

AD 输入电压：

$$V_{AN1} = R_{TH} \times I = \frac{R_{TH} \times V_{CC}}{R + R_{TH}}$$



这里， $R=10[k\Omega]$ 、 $V_{CC}=5.0[V]$ ，故：

代入：

$$V_{AN1} = \frac{5.0 \times R_{TH}}{10000 + R_{TH}}$$

通过这个式子，我们又可以列出 $5\sim 50^{\circ}\text{C}$ 范围内温度与电压的关系，如表格 8-3 所示。由于温度与电压是一一对应的关系，所以只要测定电压就可以知道当前的温度值了。

表格 8-3 温度与 AD 输入电压的关系

温度[°C]	AD 输入电压： VAN1[V]	温度[°C]	AD 输入电压： VAN1[V]
5	3.62	30	2.22
10	3.35	35	1.96
15	3.07	40	1.71
20	2.79	45	1.49
25	2.50	50	1.29

8.3 制作一个表示温度的程序

接着，我们来利用温度传感器制作一个可以检测当前温度的程序。有关 AD 转换器的使用方法在前一章已经给予说明。为了让温度检测便于观察，我们借用 LED 来表示温度的变化。

8.3.1 程序概要

此程序的主要内容为：利用 AD 转换器采样电压信号（即温度信号），然后根据这个信

号来控制发光 LED 的个数。关于 AD 转换器的设定部分，我么仍将以上一章的设定为基础程序的主要步骤归纳如下，程序流程图如图 8-3 所示。







- 1、程序启动后先要做初始化设定（AD 转换器的初始设定）。
- 2、AD 变换开始（AD 转换器采样热敏电阻两端的电压）。
- 3、AD 转换结束（取出 AD 变换值）。
- 4、根据取出的 AD 变换值来控制 LED。
- 5、之后重复（2）～（4）的处理过程。

程序的主要流程是这样的：一开始的初始设定与上一章基本相同，唯一不同的是 AD 初始设定时用到的时频道 1，这是因为温度传感器的输出端连接在模拟输入端口 1 上面（上一章种用到的是频道 0）。重复模式也与上一章相同是连续变换模式，所以 AD 转换器的启动次序为：

AD 启动（AD 变换开始） ⇒ AD 变换结束 ⇒ AD 变换开始 …

每一次 AD 转换结束时将会产生一个中断请求，CPU 相应这个中断请求后先对中断请求位清零，然后取出 AD 变换值，接着根据这个变换值来控制 LED 的状态。另外 LED 的状态如下表所示：

表格 8-4 LED 状态

取出的 AD 值	LED 状态	温度
0 ~ 65	 LED 全亮	50℃ 以上
66 ~ 86	 LED1~4 亮	40℃ ~ 50℃
87 ~ 112	 LED1~3 亮	30℃ ~ 40℃
113 ~ 141	 LED1~2 亮	20℃ ~ 30℃
142 ~ 170	 LED1 亮	10℃ ~ 20℃
171 ~ 255	 LED 全灭	10℃ 不到

每次 AD 变换结束时将产生一个中断

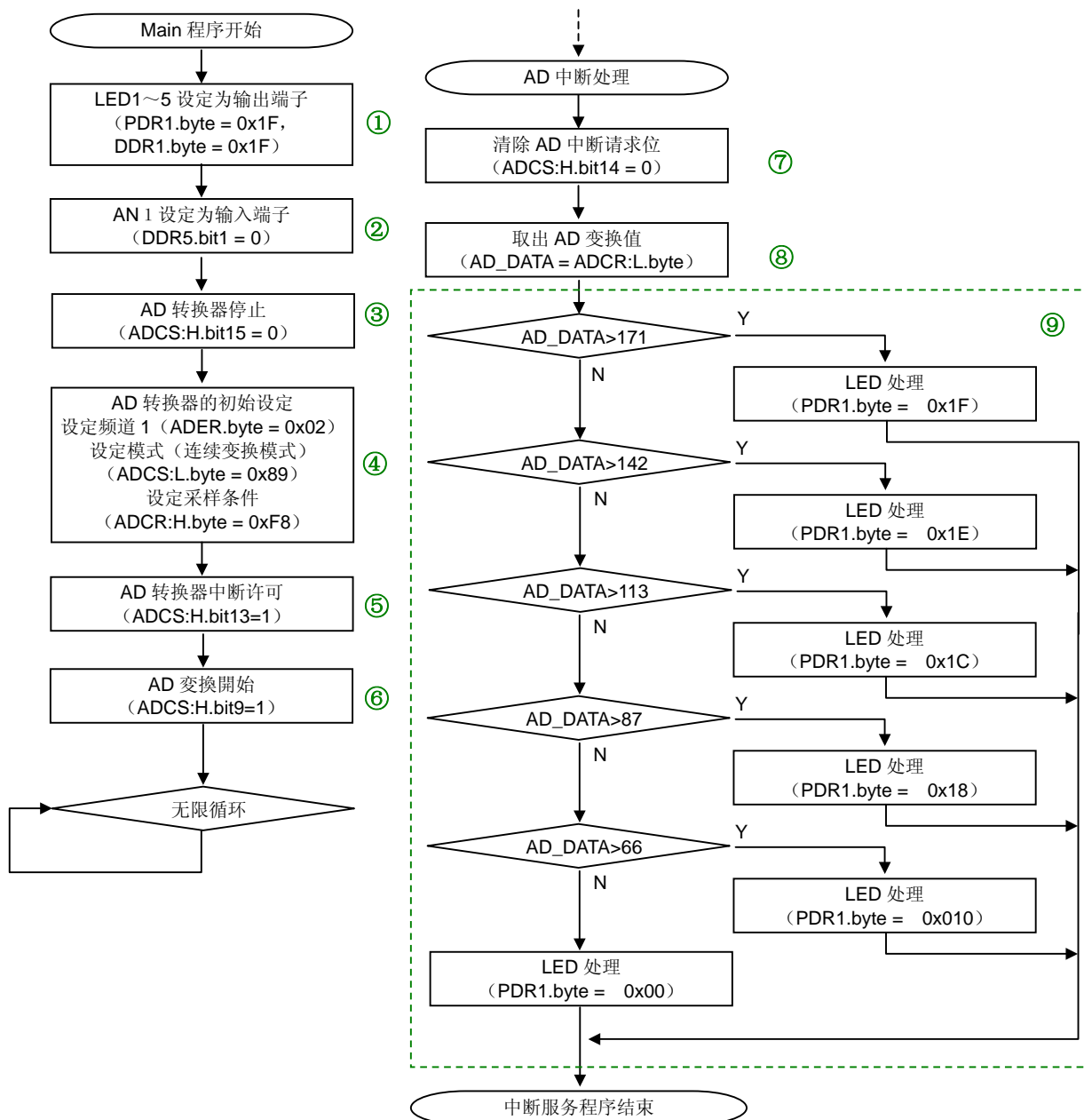


图 8-3 程序流程图

8.3.2 程序的制作与执行

接着让我们实际来写这个程序吧。首先，请参照「附录 A.1」打开“sample.prj”内的源文件“main.c”，然后将图 8-4 及图 8-5 虚线部分内的内容输入到主程序内。注意：除了 main.c 以外，Project 内的其他文件请不要做任何修改。当添加完毕后，再按照「付録 A.2」所示的顺序对源程序进行“build”。如果出现编译错误，请核对图 8-4 及图 8-5 检查添加的内容里是否有文字错误。“build”顺利完成后“ACCEMIC MDE”将自动启动。

接着，转向“ACCEMIC MDE”界面，对编译好的程序进行 debug（调试），并实际测试运行情况。程序的执行方法请参照「附录 A.3」。如果程序顺利启动，LED 将根据当前的温度来发光，比如当前的室温为 10~20℃的话，LED1 和 LED2 将会发光，然后您可以用电吹风对其稍微加热，你会发现 LED3~LED5 会逐渐点亮。值得注意的是：过高的温度将会到时 PCB 板的损坏，请用户酌情处理。

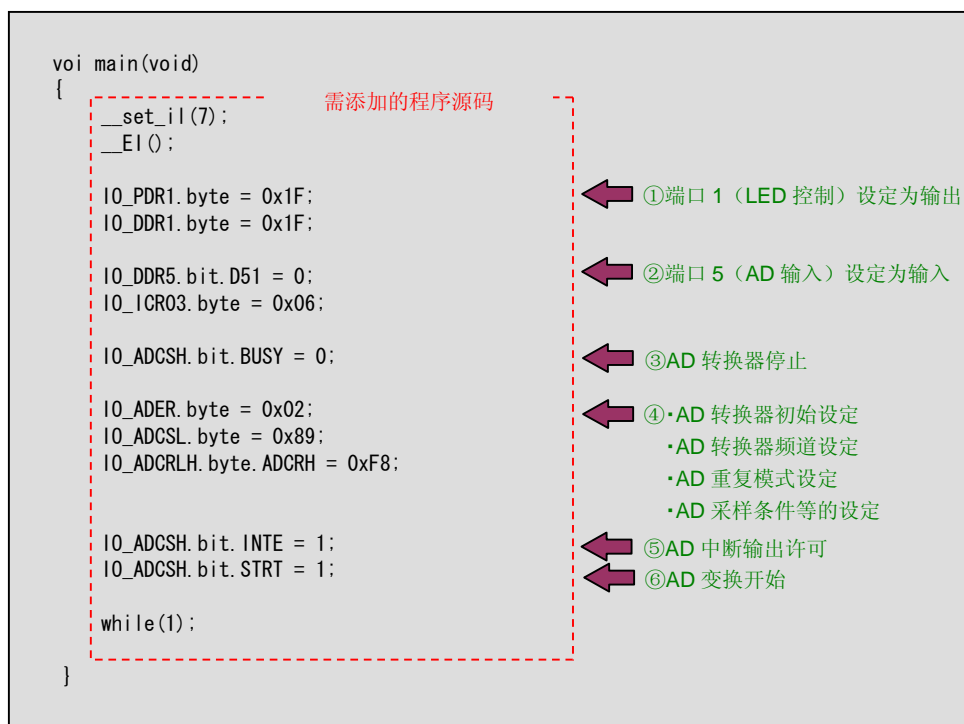


图 8-4 程序例（main 程序）

```

__interrupt void ADC_int(void)
{
    unsigned char AD_DATA;

    IO_ADCSH.bit.INT = 0;

    AD_DATA = IO_ADCRLH.DATA8;

    if(AD_DATA > 171) {
        IO_PDR1.byte = 0x1F;
    }
    else if(AD_DATA > 142) {
        IO_PDR1.byte = 0x1E;
    }
    else if(AD_DATA > 113) {
        IO_PDR1.byte = 0x1C;
    }
    else if(AD_DATA > 87) {
        IO_PDR1.byte = 0x18;
    }
    else if(AD_DATA > 66) {
        IO_PDR1.byte = 0x10;
    }
    else {
        IO_PDR1.byte = 0x00;
    }
}

```

需添加的程序源码

← ⑦清除中断请求位

← ⑧取出 AD 变换值

← ⑨根据 AD 变换值控制 LED

图 8-5 程序例（中断服务程序）

另外，图 8-5 的 ADC_int(void)函数是 AD 转换器的中断服务程序，所以要加上中断类型修饰__interrupt，它只有在 AD 发生中断（一次 AD 转换完成）时才被自动调用，正如前面所说的，单片机内部有很多中断机能，对应不同的中断应有不同的中断服务函数，为了明确告诉程序某个中断发生时应该调用哪个中断服务程序，通常在主程序中需声明一个 vector 表（向量表），用来定义中断函数所对应的向量号。而本程序中已经为用户预定义好了 AD 中断程序的向量号，因此不必再做定义。（注意：向量号的定义要用到中断程序名，故请不要更改原中断程序的程序名 ADC_int(void)。

说明：在上述源码中有诸如“IO_XXX.byte”或者“IO_YYY.bit.YY”之类形式的变量，其实这些都为了程序员方便标记而在头文件里被事先预定义好的形式。

比如：IO_ADCSH.bit.INTE → 就是 ADCS 的高位中的 INTE 位。

具体请参照、「附录 B」。

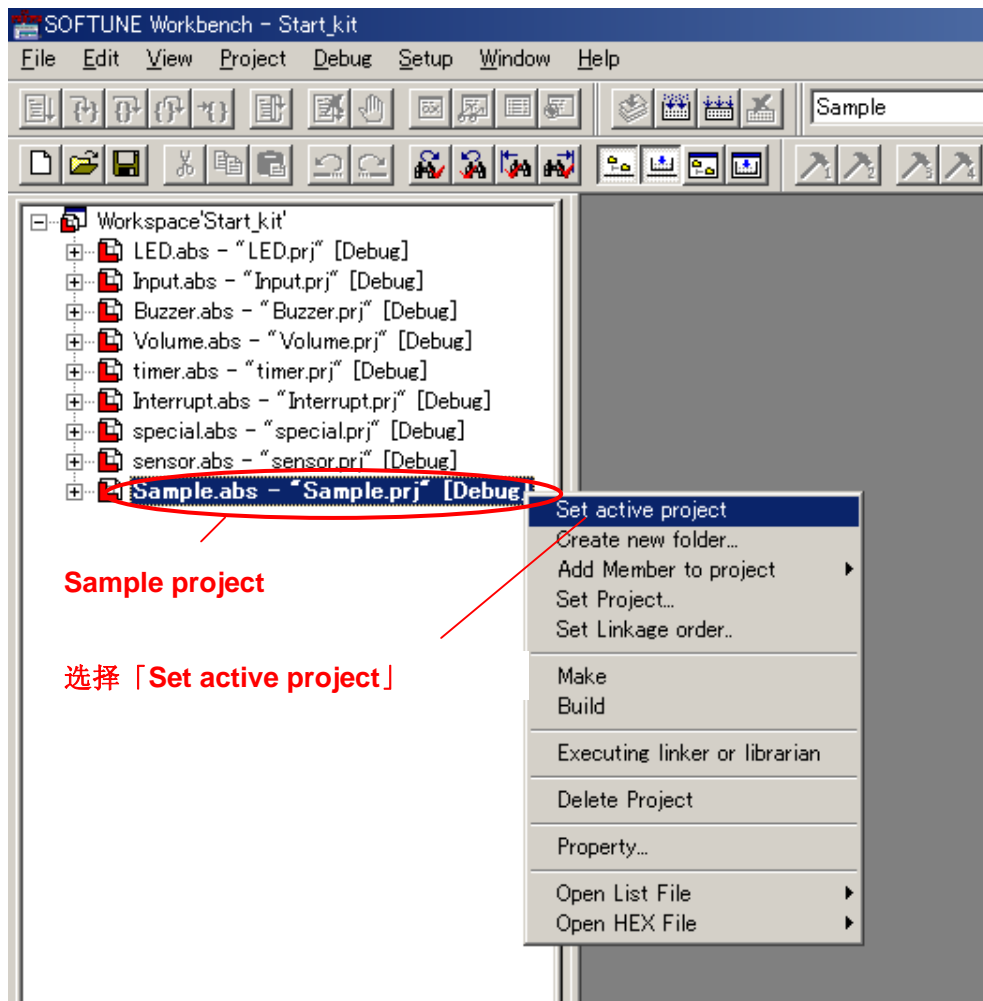
A 附录 A（程序制作流程）

A.1 程序的制作方法

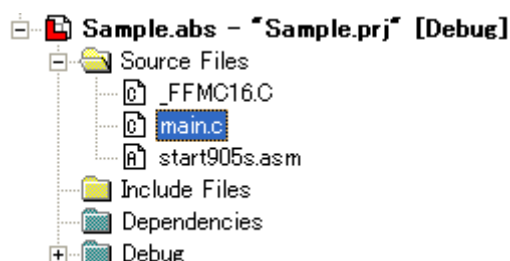
这里，我们将对程序的具体制作方法做详细说明。

首先，请参照「1.1.5 SOFTUNE 的设定与启动」启动 SOFTUNE，然后打开「Start_kit.wsp」文件。为了便于用户编写程序，我们已经准备好了一个叫 Sample 的空白 project，你可以直接利用这个 project 来编写你自己的程序。

要使用这个空白的 project，请先选择 Sample.prj，然后右击它，在弹出的子菜单中选择「Set active project」。你可以发现 Sample.prj 字样变成了组体显示表明当前 project 有效，接着你就可以输入源码然后进行编译啦。



打开 **Sample.prj** 你可以看到以下几个文件夹，在 **Source Files** 文件夹里有 **main.c**，请双击打开它。



打开 **main.c** 后，你可以看到其中的源码。

```
#include "_ffmc16.h"
#include "extern.h"
// #include "monitor.h"

void main(void)
{
    
}

/* Vector Table */
#pragma section INTVECT, locate=0xffffc00
#pragma intvect _start          0x8  0x0    // Reset Vector
```

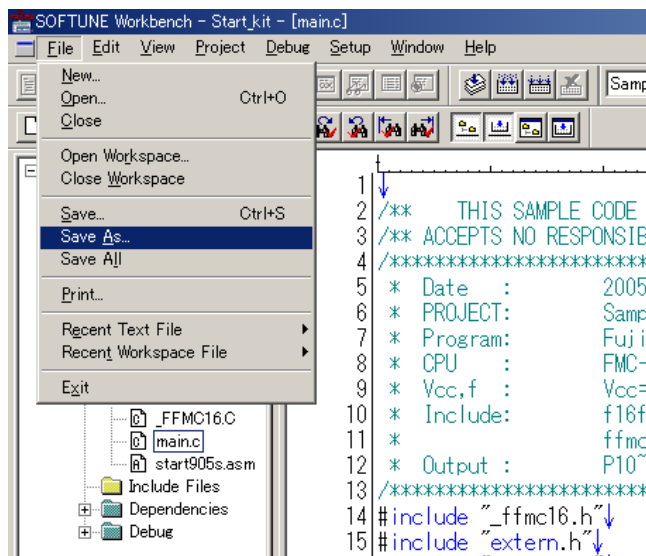
← 在这里输入源码。

← 中断向量定义。

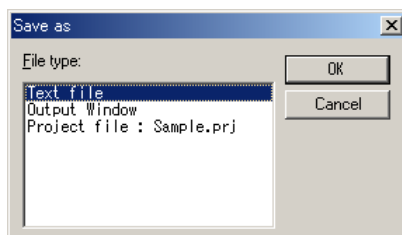
对于 **main.c** 中其他部分的源码请不要随意做改动，否则会影响正常编译，如果上图虚线部分内已经有源码存在，你可以将其删除，然后在理面添加新的源码。但是，如果你想保存之前编好的程序，请按照以下方法进行保存。

<程序（源码）的保存方法>

当源码在被编辑的状态下，点击「File」菜单 → 「Save As」。

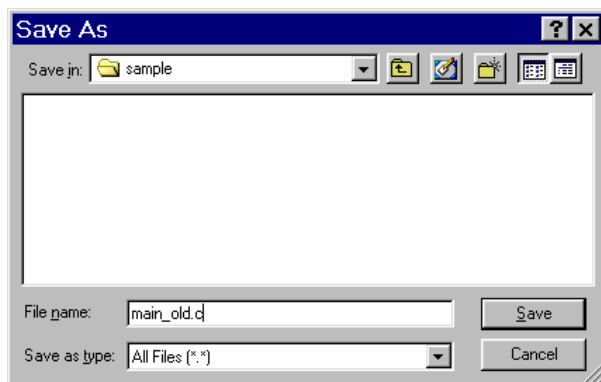


接着，你可以看到以下窗口弹出，选择「Text file」后点击「OK」。

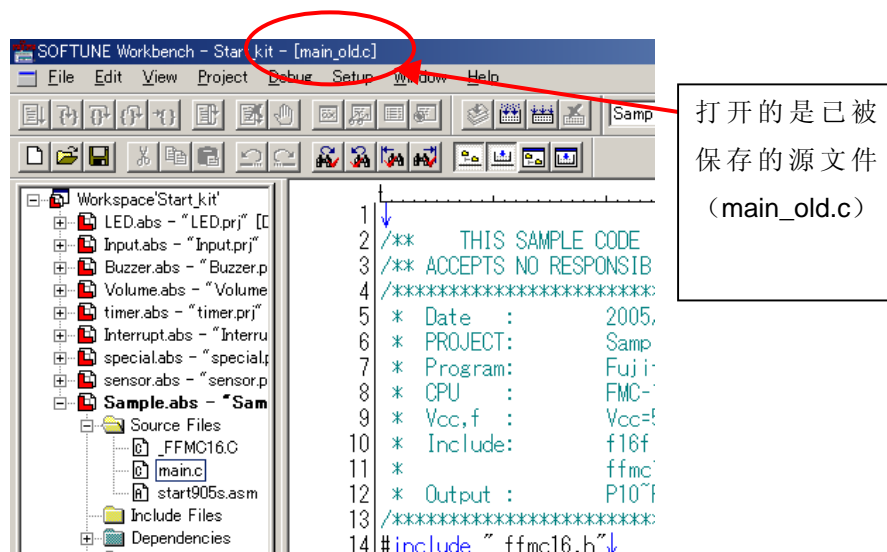


然后选择保存路径及设定文件名，最后点击「Save」。

这里假设新文件名为 `main_old.c`(注意后缀名)



注意：保存后，Softune 里打开的当前文件是已被保存的文件，请关闭它。



然后，再从 **Sample.prj** 的 **Source Files** 文件夹里打开保存前的文件 (**main.c**) 进行编辑，你可以从 **Softune** 界面右上角的文件名来判断当前所打开的文件。

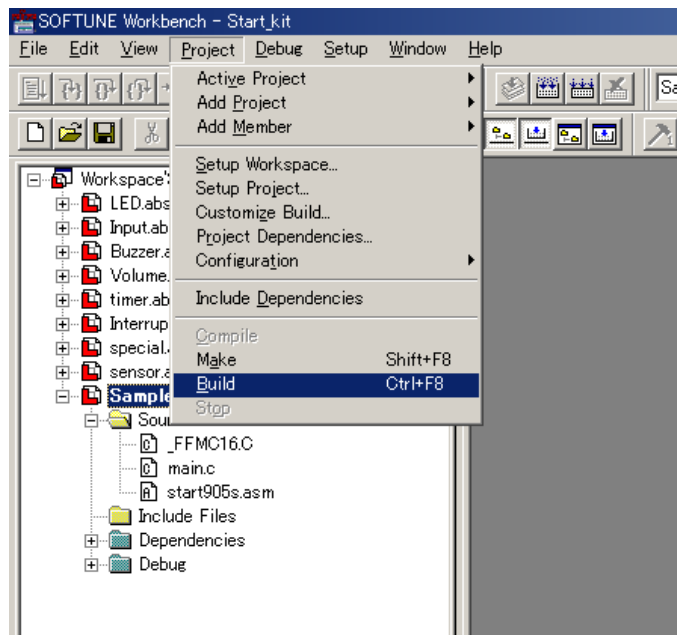
A.2 build 程序的方法

在编辑好源程序后，就要进行 build。

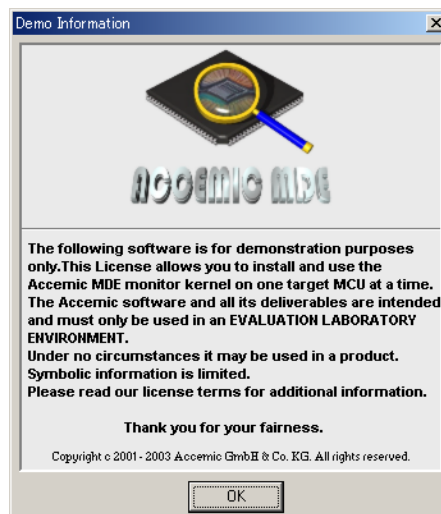
所谓 build 就是指对源程序进行编译并 link（连接）。

你可以选择「Project」→「Build」对当前的 project 进行 build。

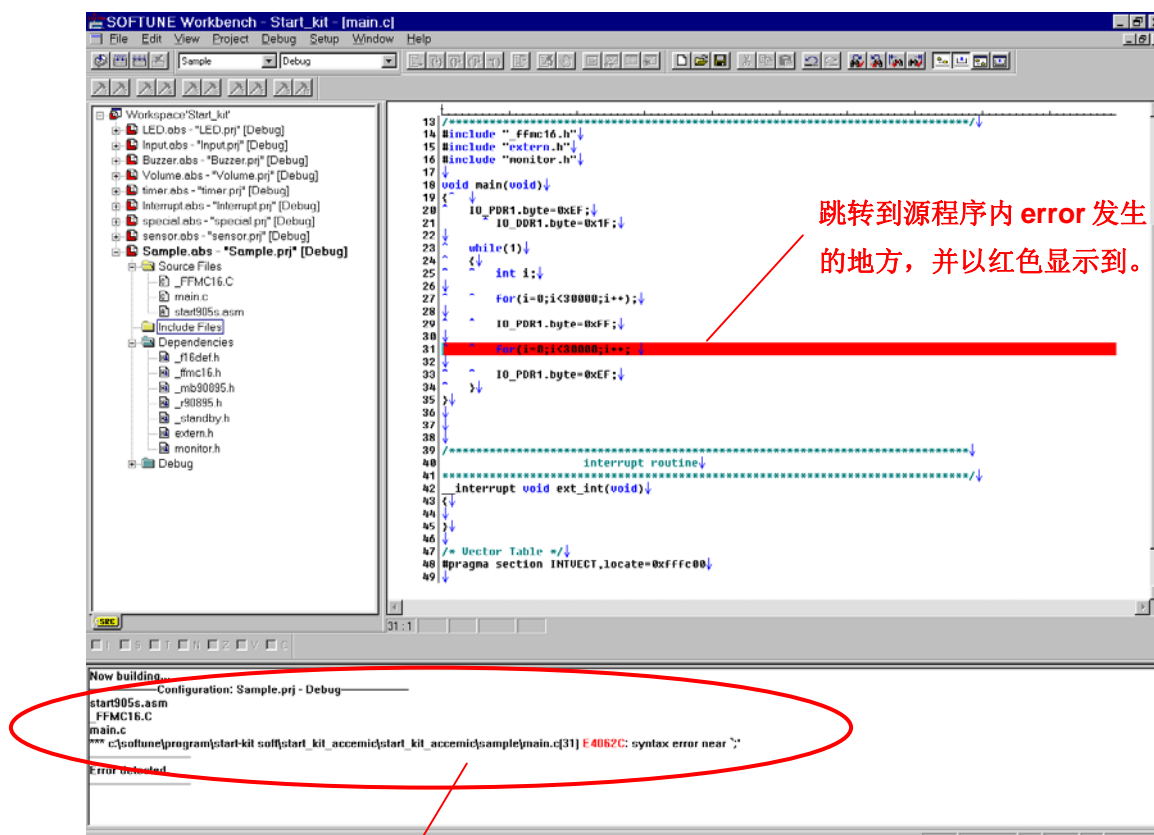
注意：只有 project 名是粗体显示的 active project 才能被 build。其他 project 还是保持原样的。



如果编译时未发生错误的话，ACCemic MDE 将会自动启动，由于是限制版，所以必须先等 10 秒，然后点击 OK，便可以进入正式的界面。



如果作成的程序内有错误的话，将会显示以下内容。请修改错误的语句并再度 build。



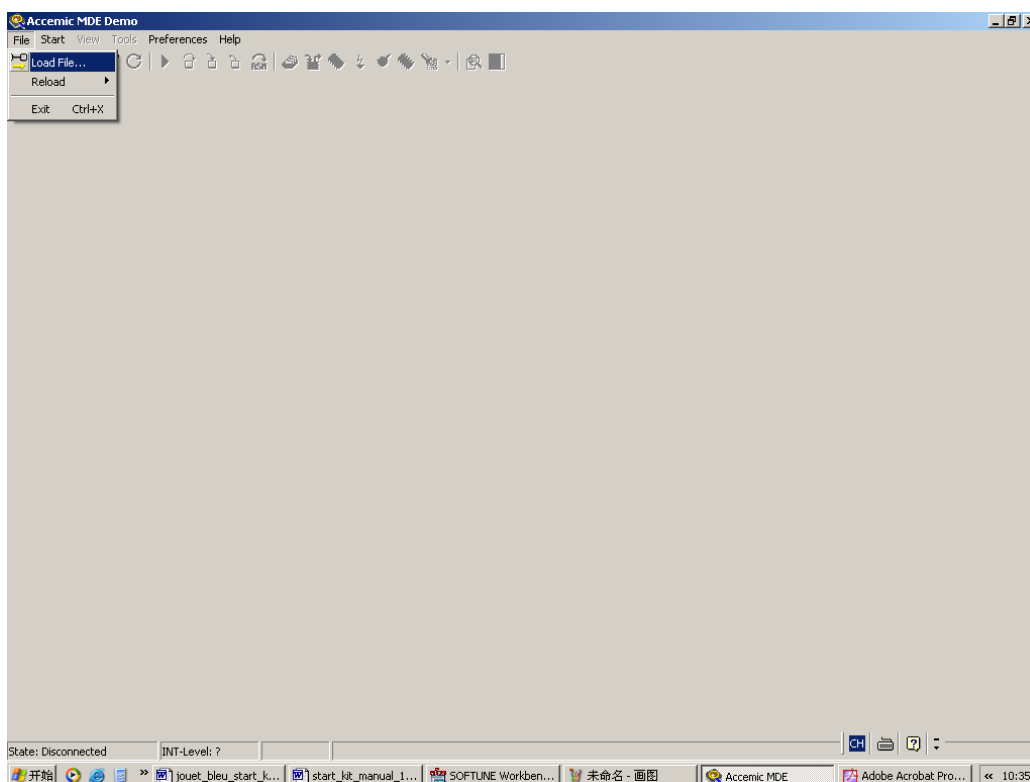
显示 error 的窗口，双击 error 语句便可以跳转至源程序 error 发生的地方

A.3 程序的运行方法

现在让我们来实际运行编写好的程序。你可以通过 **ACCEMIC MDE** 来检查源程序的运行情况是否和你预期的一样。

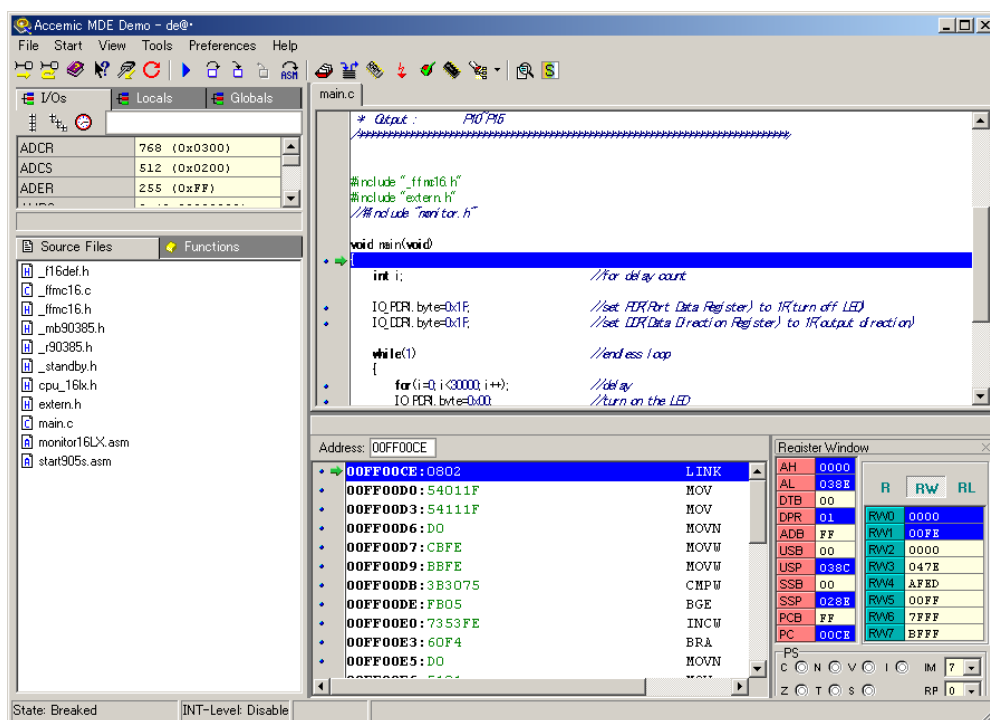
在读入目标程序前请一定确认：1、你的评测板与 PC 机已经通过 **USB** 连接。2、核心程序已经被读取。3、评测板上 **MODE** 设定为 **RUN**(具体请参照 1.1.6)

首先点击「File」→「Load File」






然后，选择刚才在 **Softune** 里被编译过的 **project** 的文件夹，并找到该文件夹下的 **ABS** 子目录，在这个子目录里有一个 ***.ABS** 的文件，*一般就是此 **project** 的名字。双击并读取它。
说明：一般对 **project** 进行 **build** 以后都会在该 **project** 下的 **ABS** 子目录里生成一个 **ABS** 文件，这便是我们要读入的文件（即实际写入 **FLASH** 芯片的机器码）

成功读入 **ABS** 文件后，**Accemic** 的界面将如下图所示：



通过以下这些按钮可以对程序的运行进行控制：

功能	按钮	按下时的操作
运行		程序开始运行
停止		将运行中的程序停止
reset		单片机 reset

B 附录 B（寄存器的写入 / 读出方法）

B.1 单片机内寄存器的写入 / 读出方法

在本书中的样例程序里常常用到类似“IO_XXX.byte”或者“IO_YYY.bit”之类的变量形式，初次接触本公司产品的用户必定会感到陌生，其实，这些变量都对应着单片机内部的寄存器，至于为什么在前面加上“IO_”或者“.byte”、“.bit”之类，这只是为了便于分辨及记忆，而在 IO 头文件中给予事先定义的，一般“.byte”的变量就是对整个 8 位寄存器进行字节操作，而“.bit”则是对一个寄存器中的某位变量进行位操作。只要用户正确设置了 Softune Workbench 头文件的包含路径，便可以正常地使用这些变量来对寄存器做设定了。

另外，在 Softune 的安装目录下可以找到本公司各种类型单片机内所使用的寄存器命名形式，本评测板所用单片机型号为 MB90F387S，属于 MB90385 系列，所以，你可以在 Softune 的安装目录下找到该系列单片机的 I/O 头文件：

\Softune\lib\907\include\sample\MB90385\ioreg.txt

或者通过

「开始」→「所有程序」→「Softune3」→「Sample IO Register Files」→「FFMC-16 Family Sample IO Register Files USERS MANUAL」→「MB90385 series Sample IO Register Files USERS MANUAL」

来打开用户手册，然后用查询功能来查找你所需要的寄存器命名形式。

其实，在 C 语言中你可以利用命令对单片机内的寄存器做修改，然后单片机就会根据这些寄存器的设定值开始运行，比如，我们欲将端口 1 的数据值设定为 0xFF（PDR1 寄存器），如果你知道 PDR 寄存器的地址为 0x000001，那你完全可以这样写：

```
*((volatile char *)0x000001) = 0xFF;
```

但是，如果一个程序中都这样写的话可能到最后连作者都不知道程序干了些什么事情，于是，我们将 PDR1 预定义为 IO_PDR1.byte 这个形式，这样既方便记忆又便于辨认：

```
IO_PDR1.byte = 0xff;
```

当然，这样写与刚才用地址直接寻址的写法在最终结果上是完全一致的。

C 附录 C（头文件包含路径的设定方法）

C.1 include path（头文件包含路径）的设定方法

上面我们解释了预定义变量名，如果你要使用这些预定义的变量名的话，必须正确设定头文件的包含路径，即告诉程序包含这些预定义的头文件在那个目录下。

如果你改变了 **Softune** 或者 **Accemic** 的默认安装路径的话，就需要重新设定包含路径。

在此，我们需要设定两个头文件的包含路径：

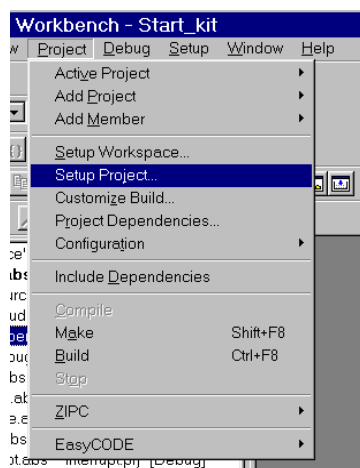
1、MB90F385 系列单片机的预定义文件路径

Softune 安装路径\Softune\lib\907\include\sample\MB90385\

2、Accemic 程序用的预定义文件路径

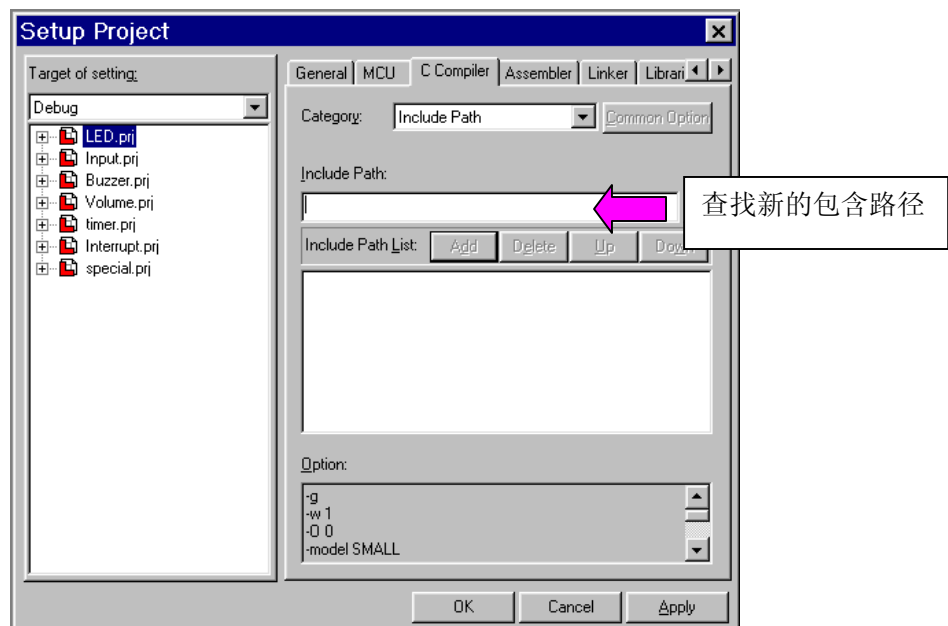
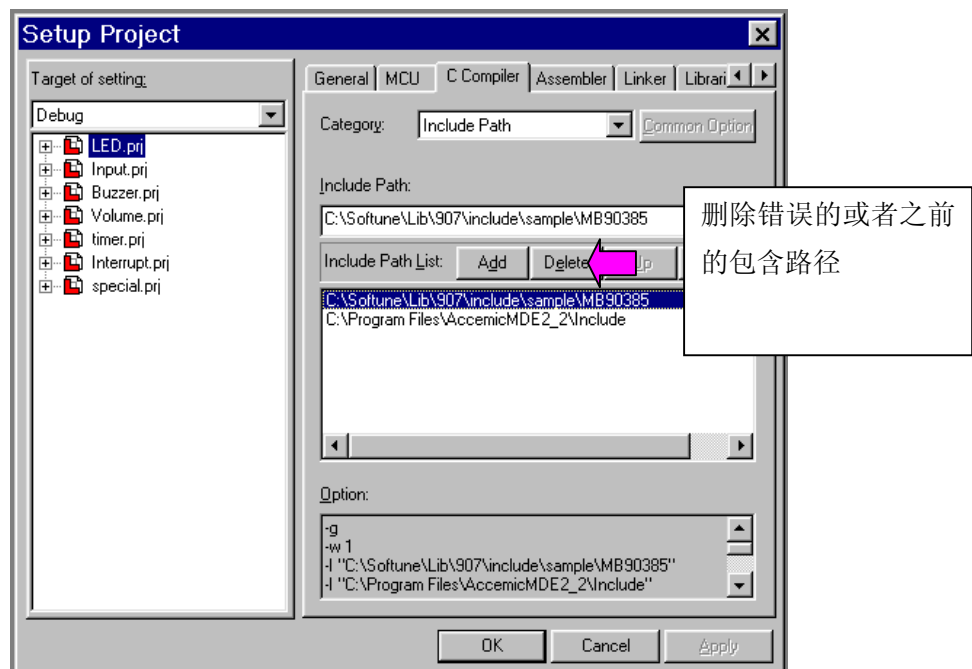
Accemic 安装路径\AccemicMDE2_2_Demo\Samples\16LX\CommonInclude

具体设定方法如下：



选择「Project...(P)」－「Setup Project...(J)」

然后选择「C compiler」标签卡，在 **category** 里选择「Include Path」这一项目，于是你可以看到如下画面的显示：

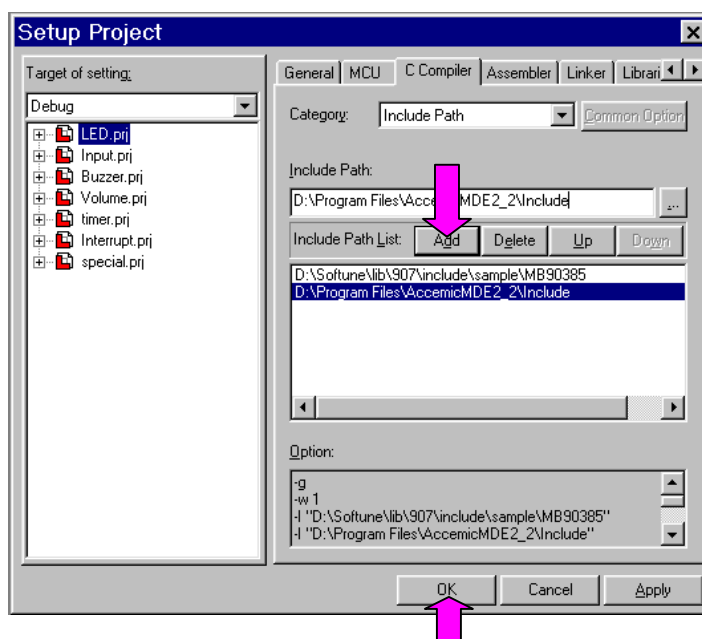


具体设定内容为:

C Compiler → Include Path(2 个路径):

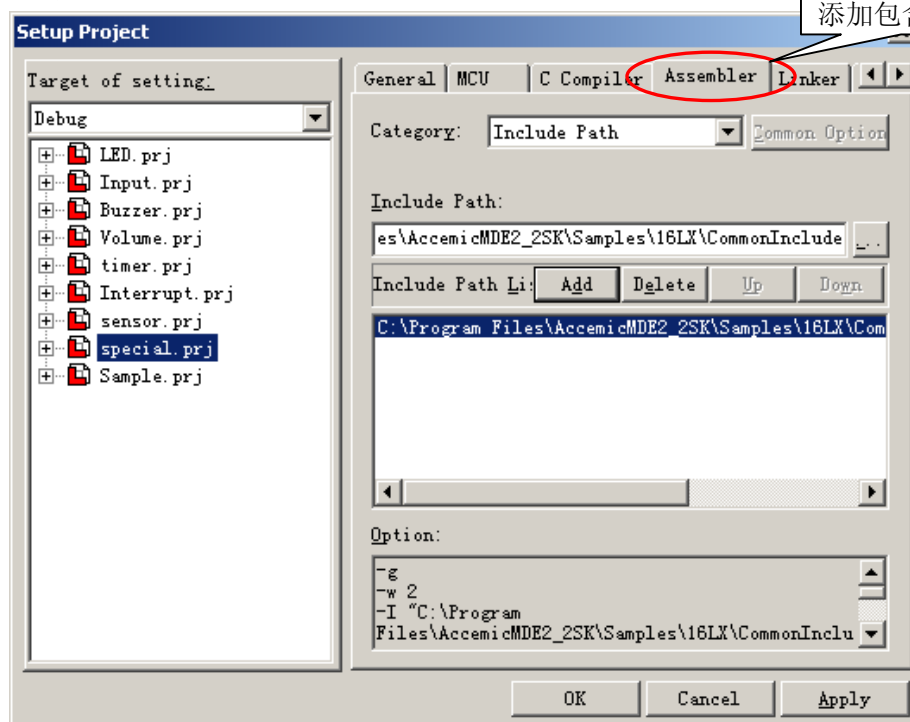
- 1、Softune 安装路径\lib\907\include\sample\MB90385
- 2、AccemicMDE2_2SK 安装路径\Samples\16LX\CommonInclude

然后点击「Add」→「OK」



点击 Add 按钮，添加新的包含路径
设定完毕后点击「OK」

同样，接着 Assembler 标签卡里的 Include Path 也要设置包含路径：



在 Assembler 里
添加包含路径

具体设置内容为：

Assembler → Include Path（1 个路径）：

1、AccemicMDE2_2SK 安装路径\Samples\16LX\CommonInclude

这样便完成了包含路径的设置。