SIEMENS

S7-SCL 编程

Programming with S7-SCL

Getting Started	Edition (2008 年 7 月)

摘　要

本文档主要用于讨论与 S7-SCL 编程相关的以下问题：

- ✧ 编程软件的基本信息
- ✧ 基本概念讲解
- ✧ 基本使用讲解
- ✧ 用于示例工程的简单应用例子
- ✧ 实际使用中常见问题及讲解

关键词

编程工具；Step7；语句表；S7-SCL；

Key Words

Programming tools；　Step7；STL；S7-SCL；

目　　录

重要提示：本文为技术交流文档，不能作为订货、选型等重要事宜的唯一依据，建议您参考 Siemens 的标准产品样本和技术手册进行产品的选型和订货。

1. 前言

本文可以作为 S7-SCL 编程语言的使用参考资料，希望读者通过对本章的阅读，能够更快更好地学习 S7-SCL 编程语言。西门子提供了 S7-SCL 编程语言的详尽手册，在安装 S7-SCL 软件包后，通过点击 Windows 菜单 开始->Simatic->Documentation->English 可以阅读到名称为“S7-SCL for S7-300 and S7-400”的 PDF 手册。此手册共分为 16 个章节，其详细地讲解了 S7-SCL 编程语言。一切关于 S7-SCL 使用的问题请以此手册为准。

相对于西门子 PLC 的其它类型编程语言，S7-SCL 与计算机高级编程语言有着非常相近的特性，只要使用者接触过 PASCAL 或者 VB 编程语言，实现 S7-SCL 的快速入门是非常容易的。所以本文将不会对 S7-SCL 进行类似手册一样的详细讲述，而是通过列举一个简单例子，使读者实现 S7-SCL 快速入门。在讲解例子内容之后，再列举出一些用户在实际使用当中经常会遇到的问题。这部分内容大多来自实际用户，具有较强的针对性，希望能够对用户有所帮助。

相关手册地址连接：

S7-300 和 S7-400 的语句表 (STL) 编程

http://support.automation.siemens.com/CN/view/zh/18653496

使用 STEP 7 V5.3 编程

http://support.automation.siemens.com/CN/view/zh/18652056

S7-SCL V5.3 for S7-300/400

http://support.automation.siemens.com/CN/view/zh/5581793

2. 软件的基本信息

2.1. S7-SCL 简介

S7-SCL（Structured Control Language 结构化控制语言）具有以下特点：

- ✧ 是一种类似于PASCAL的高级编程语言，
- ✧ 符合国际标准IEC 61131-3
- ✧ PLCopen基础级认证
- ✧ 适用于 SIMATIC S7-300（推荐用于CPU314以上CPU），S7-400，C7 and WinAC

S7-SCL 为 PLC 做了优化处理，它不仅仅具有 PLC 典型的元素（例如 输入 / 输出，定时器，

计数器，符号表），而且具有高级语言的特性，例如：

- ✧ 循环
- ✧ 选择
- ✧ 分支
- ✧ 数组
- ✧ 高级函数

S7-SCL 其非常适合于如下任务：

- ✧ 复杂运算功能
- ✧ 复杂数学函数
- ✧ 数据管理
- ✧ 过程优化

2.2. S7-SCL 与 STL：

S7-SCL 可以编译成 STL，虽然其代码量相对于 STL 编程有所增加，但我们更关心的是程序结构和程序的总体效率。类似于计算机行业的发展，汇编语言已经被舍弃，取而代之的是 C/C++等高级语言。S7-SCL 对工程设计人员要求较高，需要其具有一定的计算机高级语言的知识和编程技巧。

2.3. S7-SCL 的安装与使用：

STEP7 标准版并不包括 S7-SCL 软件包及授权，需单独购买，STEP7 Professional 版包括了 S7-SCL 的软件包及授权，安装即可。在 S7 程序中，S7-SCL 块可以与其它 STEP7 编程语言生成的块互相调用。S7-SCL 生成的块也可以作为库文件被其它语言引用。由于 S7-SCL 程序由 ASCII 文本构成，所以它非常容易被导入或导出。

2.4. S7-SCL 软件兼容性

不同 S7-SCL 软件版本与 STEP7 及操作系统之间的兼容性：图中的 X 表示兼容，- 表示不兼容

		STEP 7 V5.3			STEP 7 V5.4				
Version	Order Number	Win 2000 SP4	Win XP SP1	Win XP SP2	Win 2000 SP4	Win XP SP1	Win XP SP2	Win 2003	Win 2003 SP1
V5.3	6ES7 811-1CC05-0YA5	X	X	X	X	X	X	X 3)	X 3)
V5.1	6ES7 811-1CC04-0YX0	X 1)	X 2)	X 2)	X 1)	X 2)	X 2)	-	-

1) 仅仅 S7-SCL V5.1+SP1 或以后版本支持

2) 仅仅 S7-SCL V5.1+SP1 或以后版本支持

3) 仅仅 S7-SCL V5.1+SP1 或以后版本支持

3. S7-SCL 应用于虚拟工程

3.1. 虚拟工程工艺要求

下面将以一个虚拟工程中的应用来举例说明 S7-SCL 的使用。

虚拟工程工艺参数环节要求：

- ✧ 采集某个过程量，进行工程量转换，对其进行软件滤波，计算10个采样值去除最大值及最小值之后的平均值。（见下面举例）
- ✧ 将过程参数存储，并进行分析，优化控制策略（限于篇幅，不做介绍）

3.2. S7-SCL 简单示例

在下面的例子中，编写一个完成软件滤波程序的 FB1，程序每调用一次 FB1，其采集一个新的过程变量，存储在 FIFO 堆栈中，共 10 个周期的采样值，超过 10 个周期的采样值将被舍弃。程序将此 10 个采样值中的最大最小值找出，并计算出除去最大值及最小值之后的平均值。

平均值（滤波输出）=（10 个采样值之和-最大值-最小值）/8

平均值（非滤波输出）=转换后的实时采样值

1） 新建项目，插入 SCL Source

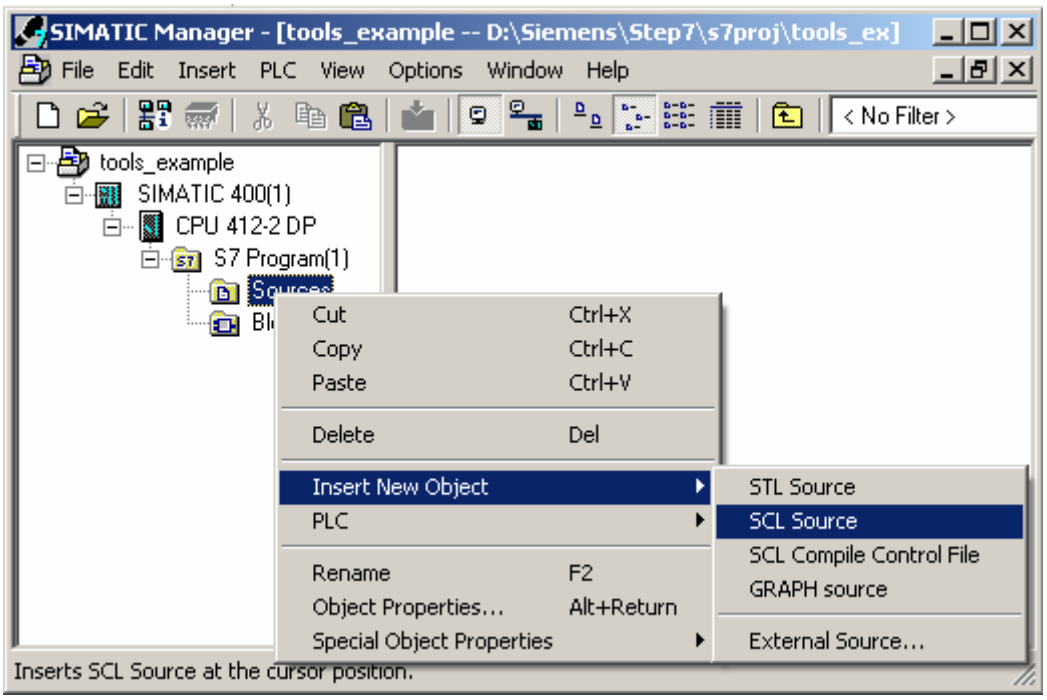


图 3-1：新建 SCL Source

2） 双击，SCL Source 打开 SCL 环境，并使用 FB 模板

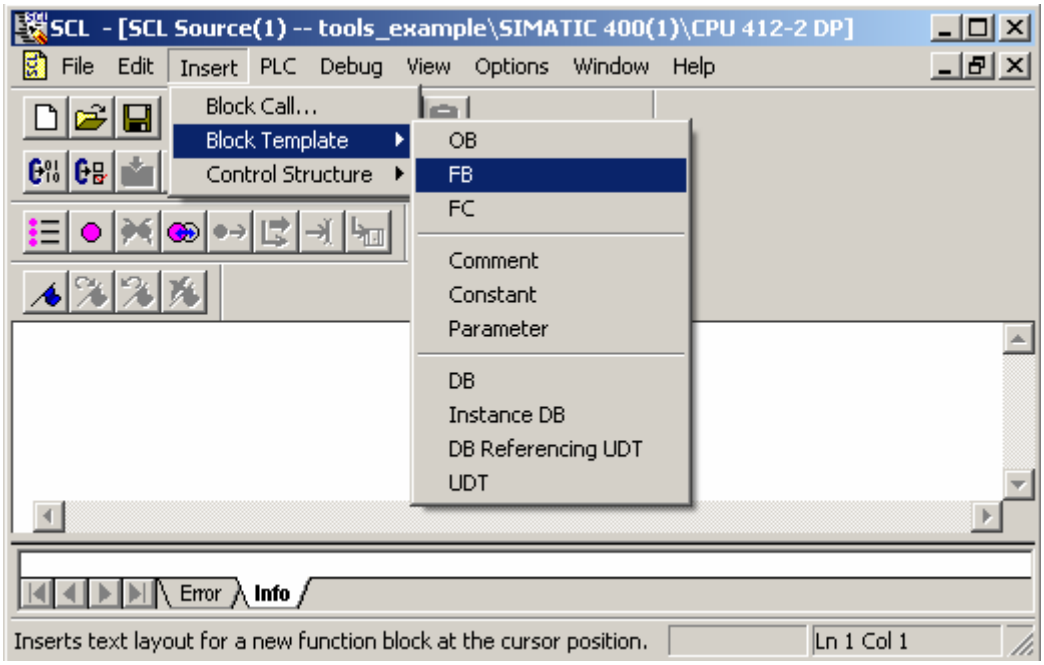


图 3-2：使用 FB 模板

3) 更改 FB 编号

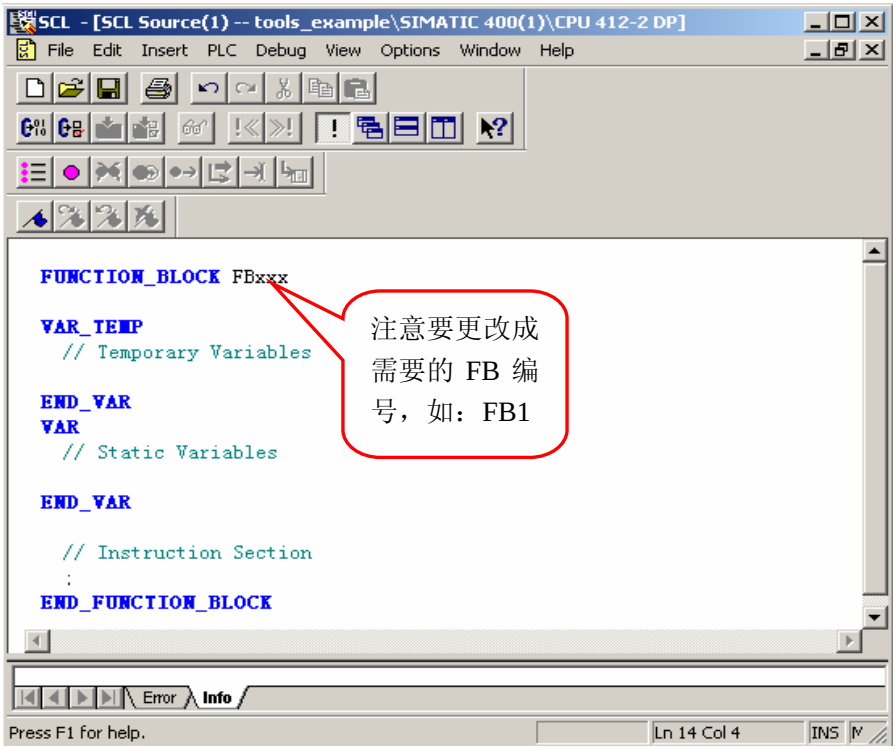


图 3-3：更改 FB 编号

4) 使用参数模板

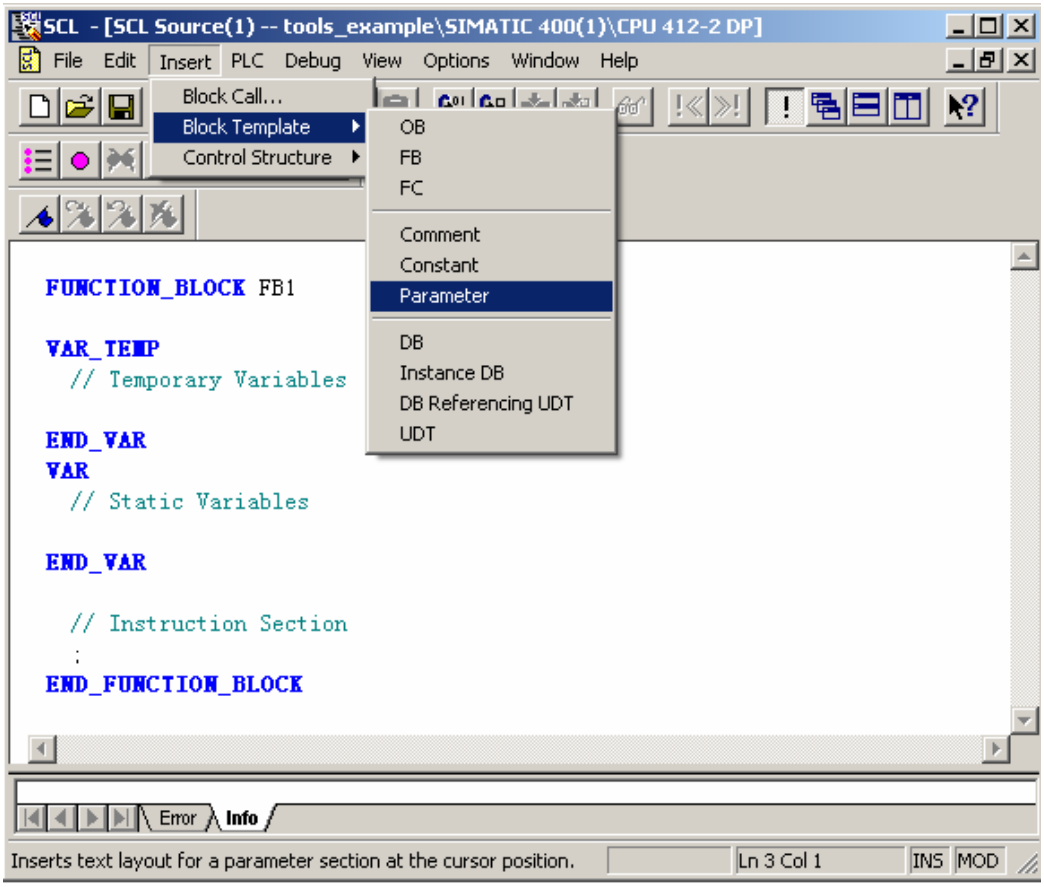


图 3-4：添加 FB 输入输出参数

5） 编辑 FB 参数

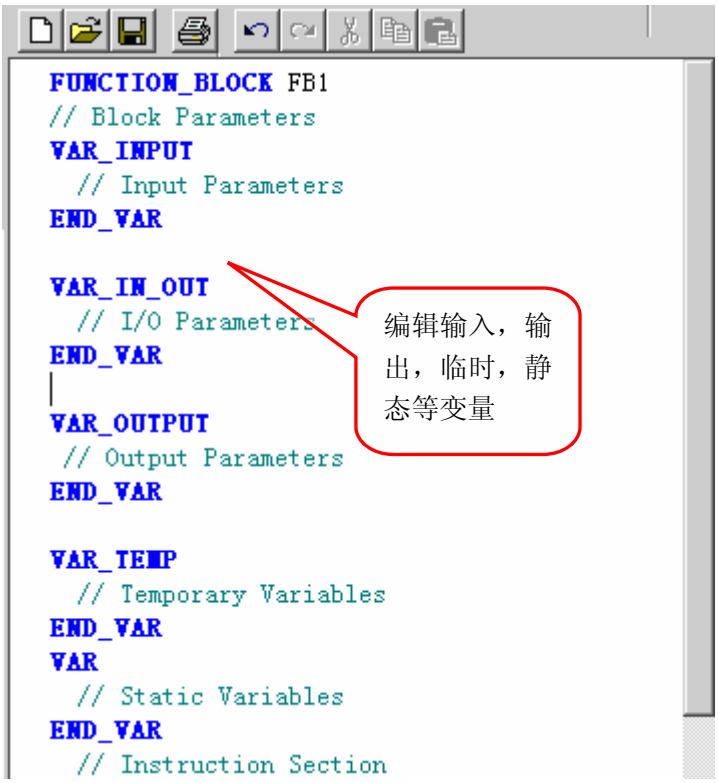


图 3-5：添加 FB 输入输出参数

6) 使用调用功能块向导

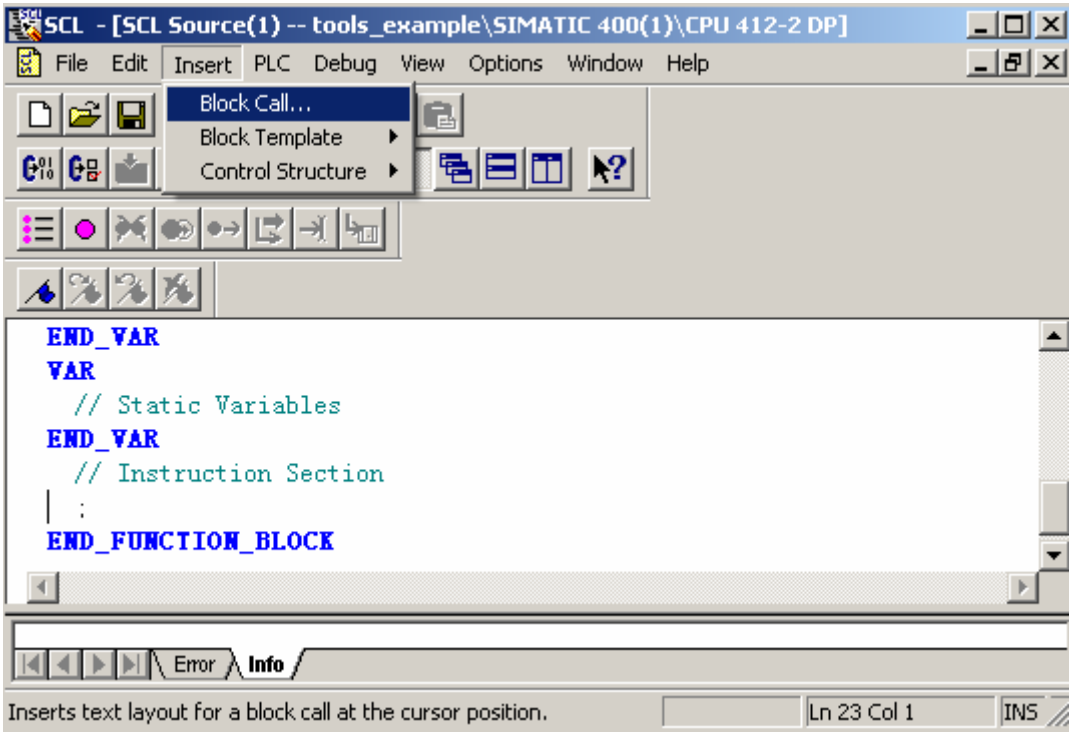


图 3-6：使用调用功能块向导

7) 调用 FC105

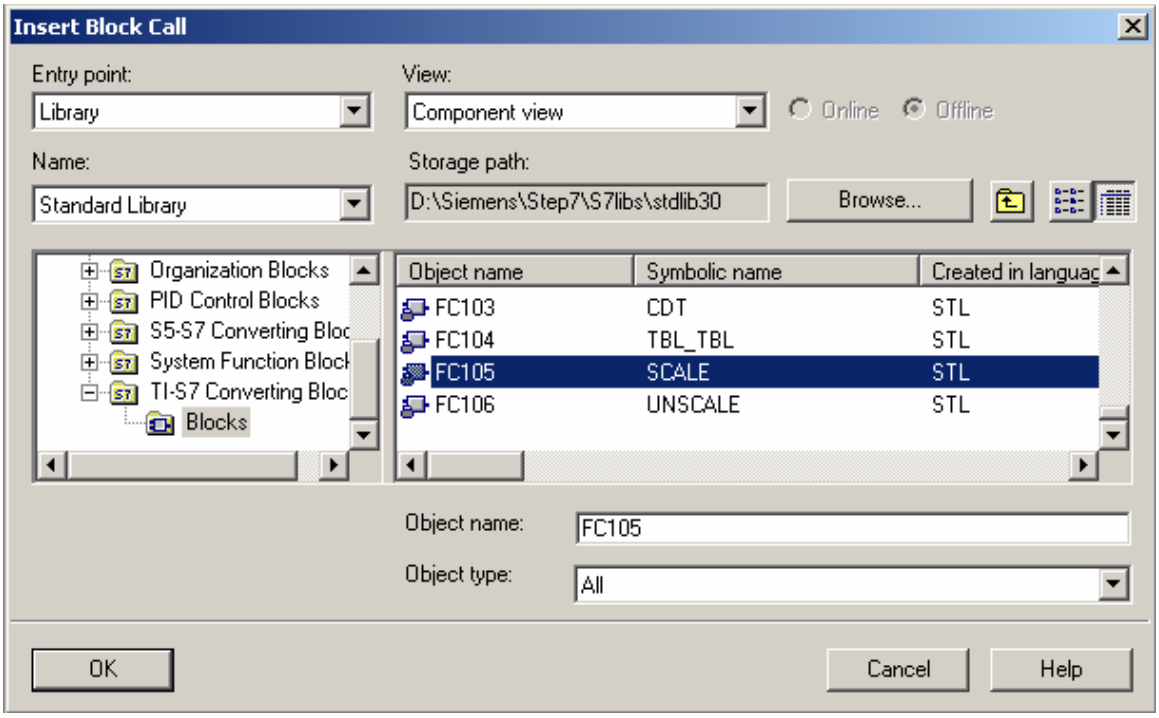


图 3-7：调用 FC105

8) 由于在 SCL 中调用了 FC105，还需在 SIMATIC Manager 将 FC105 添加到项目中

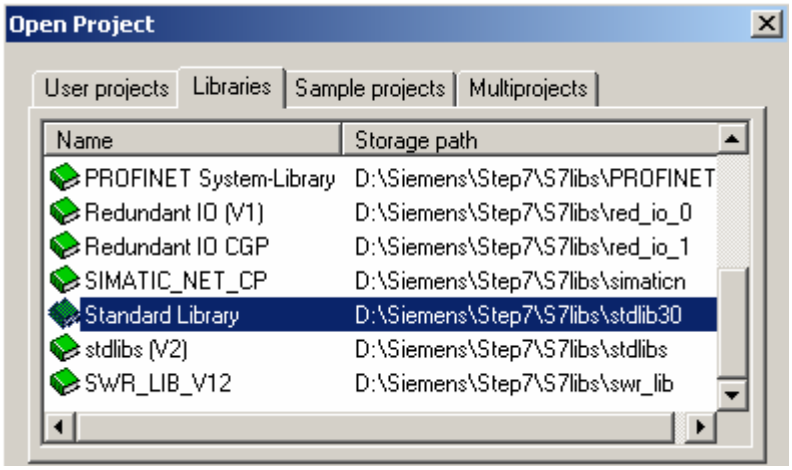


图 3-8：SIMATIC Manager 中打开库文件

9) FC105 复制到当前项目中

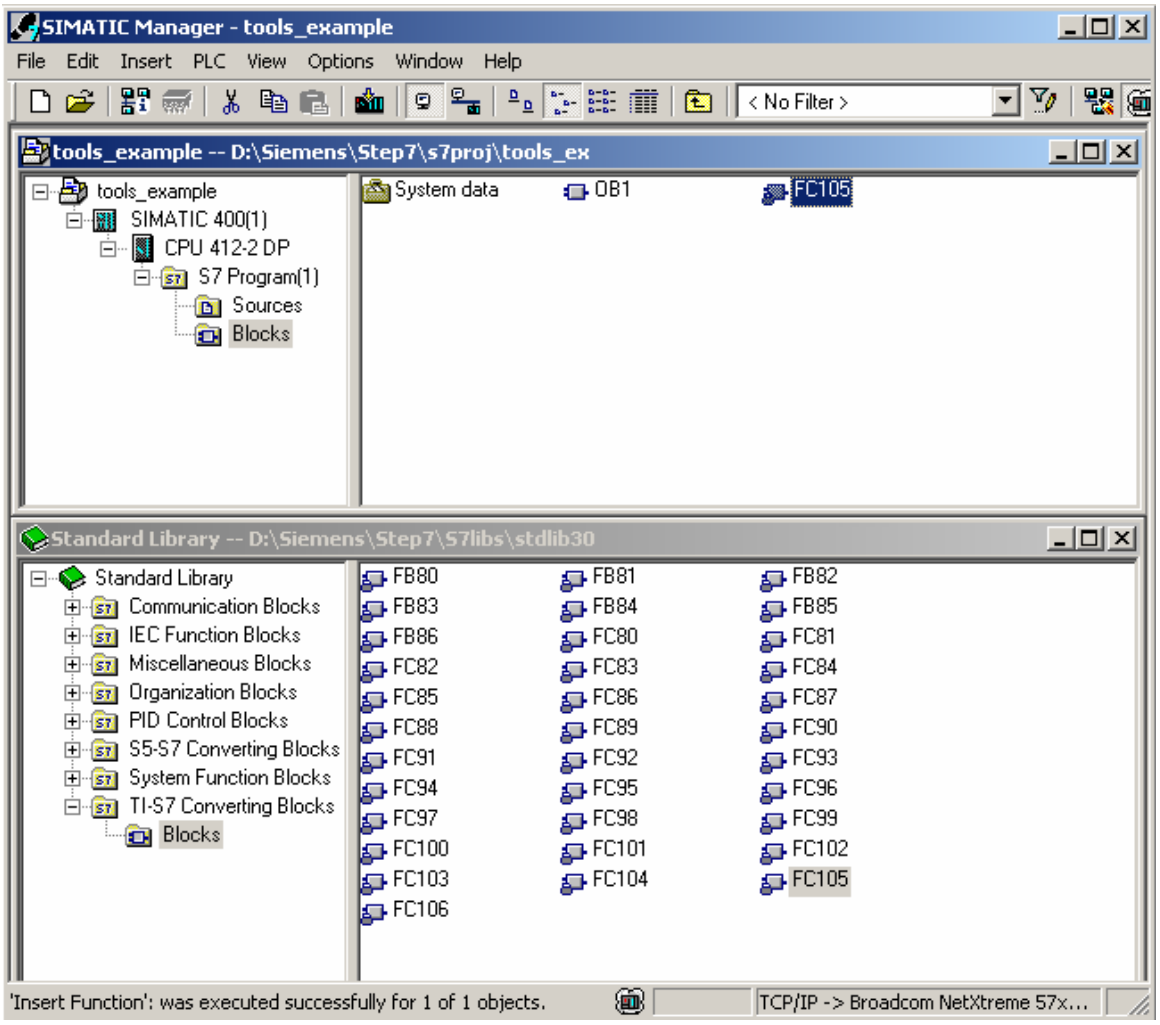


图 3-9：将 FC105 复制到当前项目中

10) FB1 参数定义

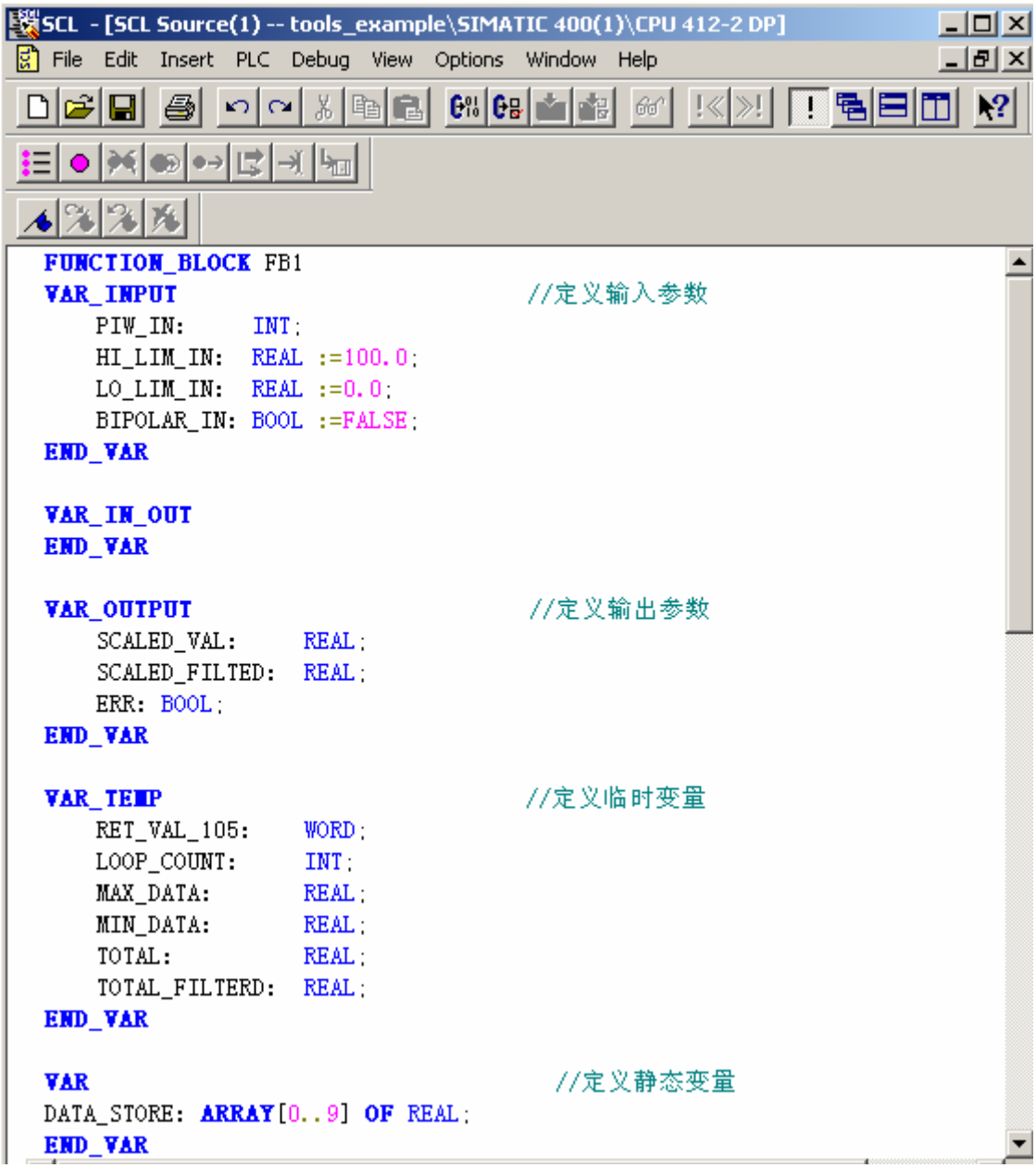


图 3-10：FB1 参数定义

11) FC105 调用

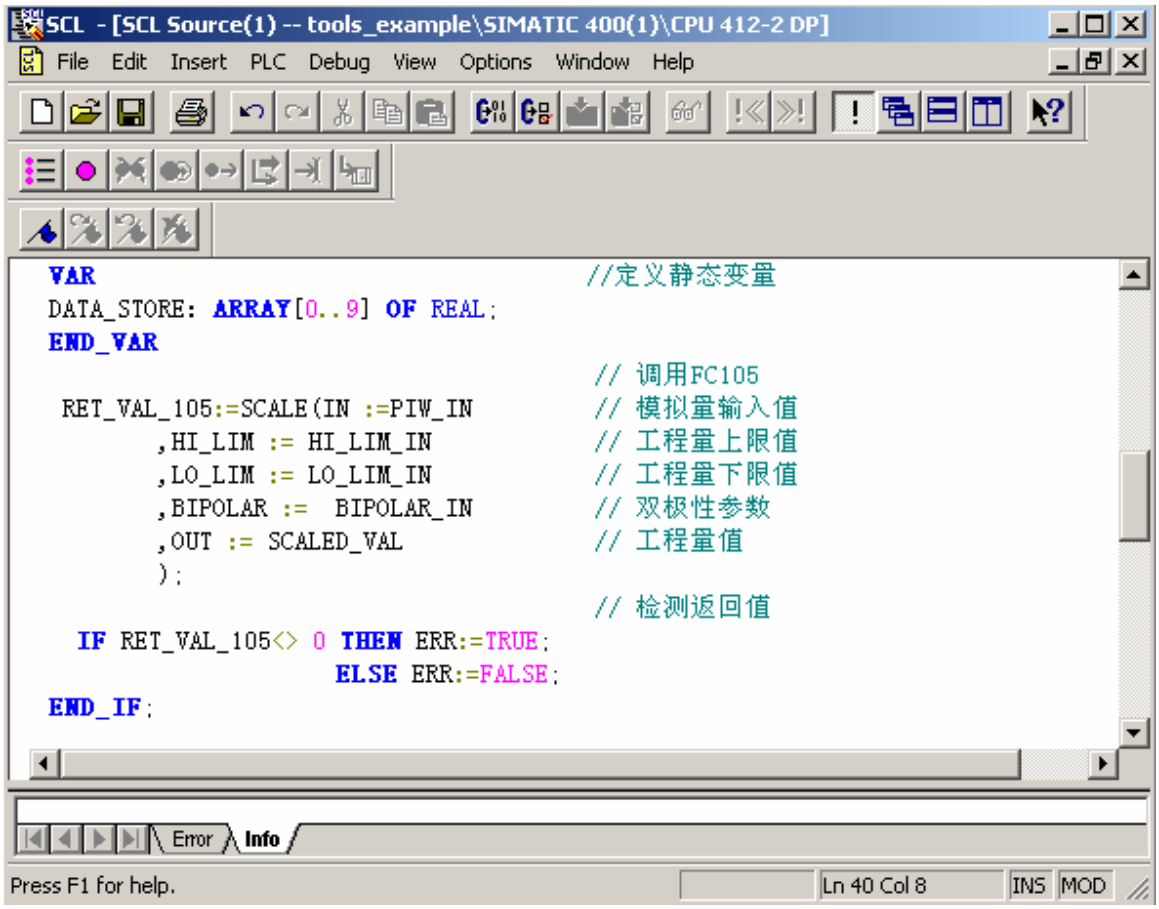


图 3-11：FC105 调用

12) 下面的程序中使用了判断条件，在此使用 IF 模板

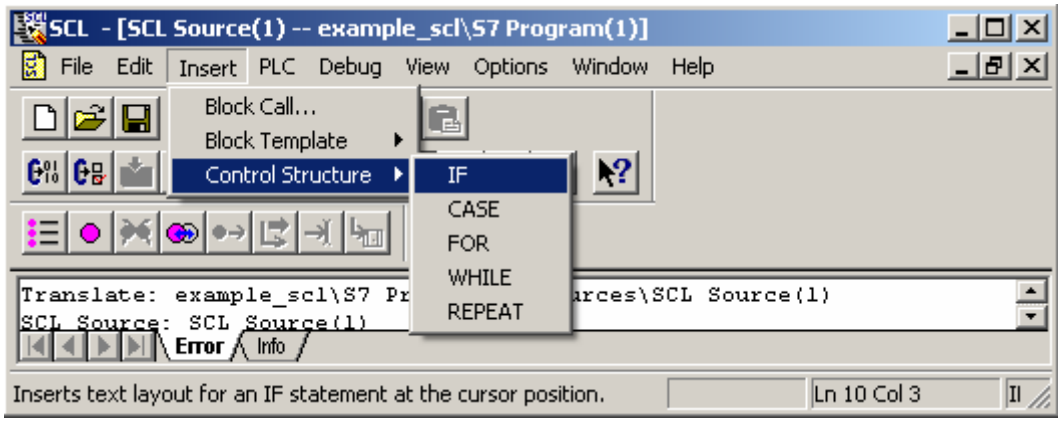
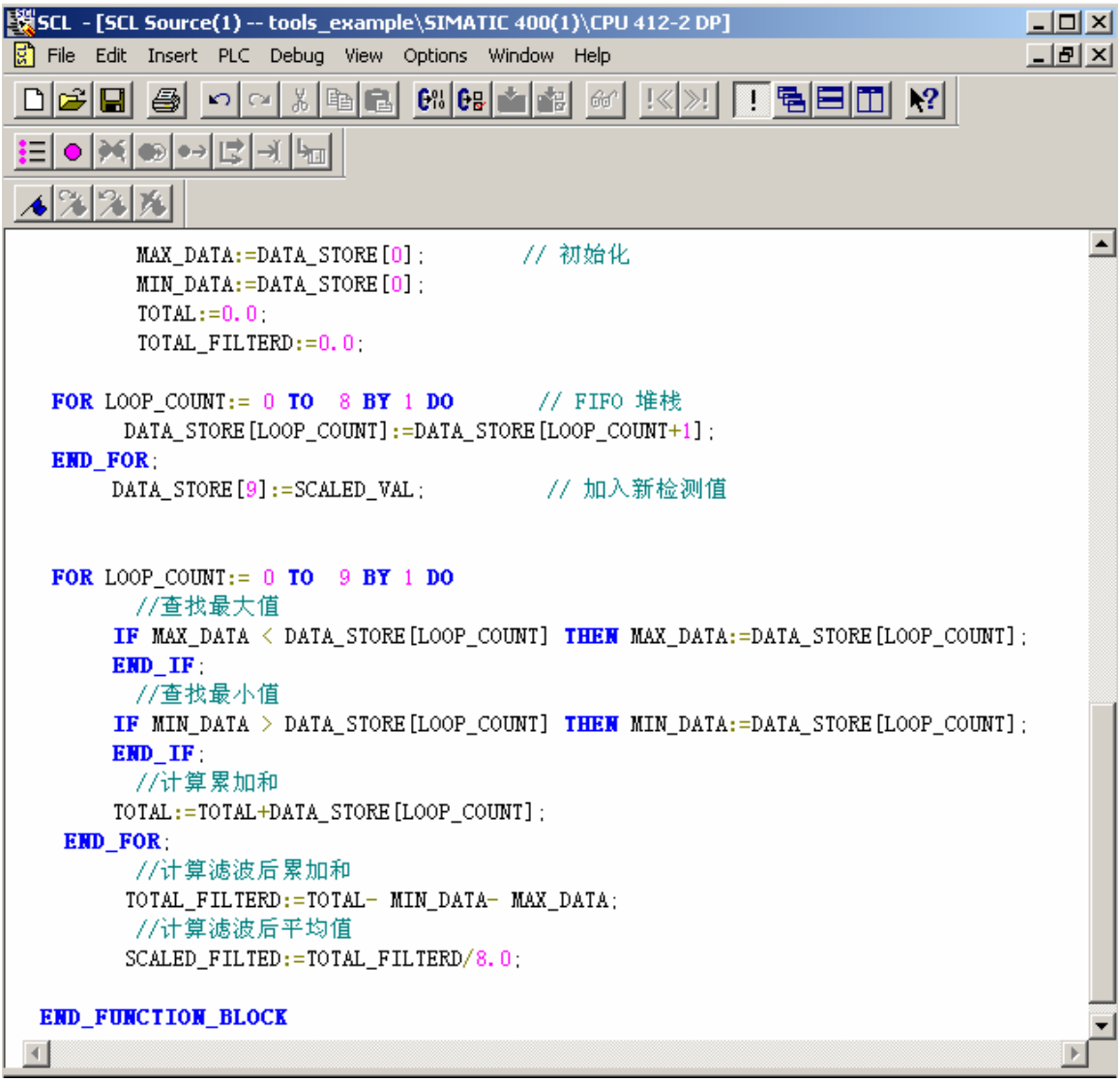


图 3-12：使用 IF 模板

13) 使用 IF，FOR-NEXT 等高级语言的方式，很容易就可编写出滤波程序




```
        MAX_DATA:=DATA_STORE[0];          // 初始化
        MIN_DATA:=DATA_STORE[0];
        TOTAL:=0.0;
        TOTAL_FILTERD:=0.0;

FOR LOOP_COUNT:= 0 TO  8 BY 1 DO        // FIFO 堆栈
     DATA_STORE[LOOP_COUNT]:=DATA_STORE[LOOP_COUNT+1];
END_FOR;
     DATA_STORE[9]:=SCALED_VAL;          // 加入新检测值


FOR LOOP_COUNT:= 0 TO  9 BY 1 DO
      //查找最大值
     IF MAX_DATA < DATA_STORE[LOOP_COUNT] THEN MAX_DATA:=DATA_STORE[LOOP_COUNT];
     END_IF;
      //查找最小值
     IF MIN_DATA > DATA_STORE[LOOP_COUNT] THEN MIN_DATA:=DATA_STORE[LOOP_COUNT];
     END_IF;
      //计算累加和
     TOTAL:=TOTAL+DATA_STORE[LOOP_COUNT];
 END_FOR;
      //计算滤波后累加和
      TOTAL_FILTERD:=TOTAL- MIN_DATA- MAX_DATA;
      //计算滤波后平均值
      SCALED_FILTED:=TOTAL_FILTERD/8.0;

END_FUNCTION_BLOCK
```

图 3-13：编写滤波程序

14) 在菜单 Option-Customize 中选择生成调试信息

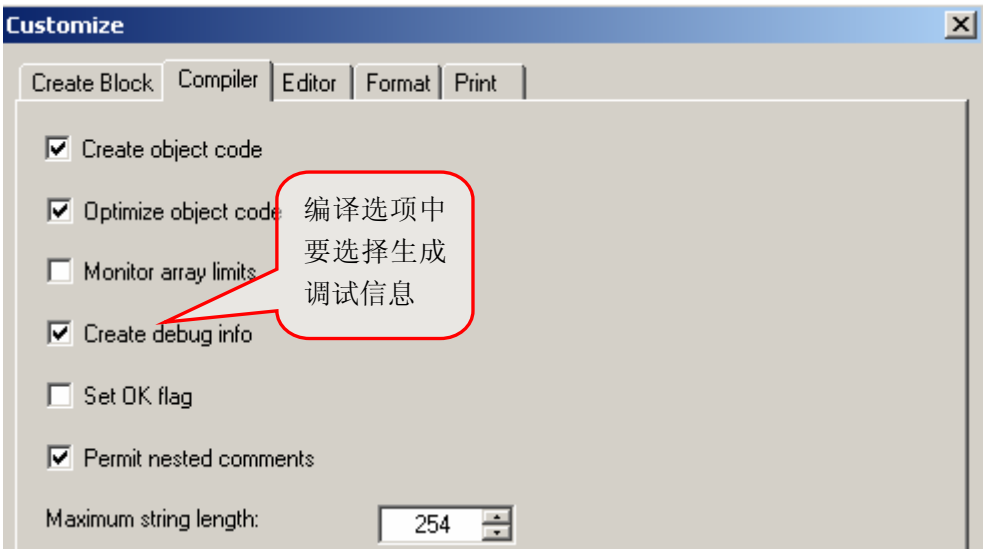


图 3-14：生成调试信息

15) 在菜单的 File-Compile 编译没有错误后，就可以生成 FB1 了，在 OB35 中可以调用 FB1,下载整个项目后监控 OB35：

- ✧ 输出参数 SCALED_VAL 为实时的采样值
- ✧ 输出参数 SCALED_FILTED 为滤波后的采样值
- ✧ 假设OB35的执行周期为T，如果某时刻采样值发生变化，那么此数值将不同于以前的多个采样值，此数值可能被认定为最大值或最小值，而最大值或最小值将被忽略，这样就实现了滤波的效果。当采样值发生变化，并维持了大于等于2T的时间，此时的输入值的两次采样值会被滤掉一个采样值，而另一个采样值将参与平均值的运算。

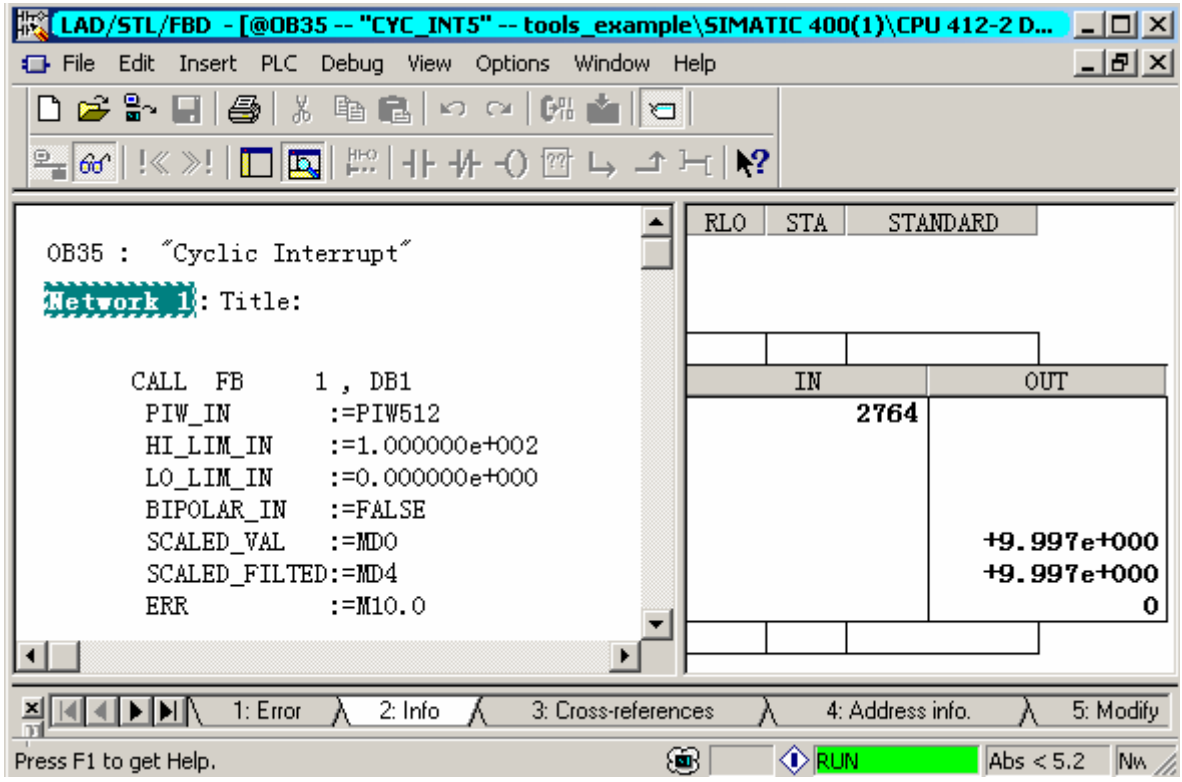


图 3-15：监控 OB35

16) 可以在高级语言界面下监控 FB1

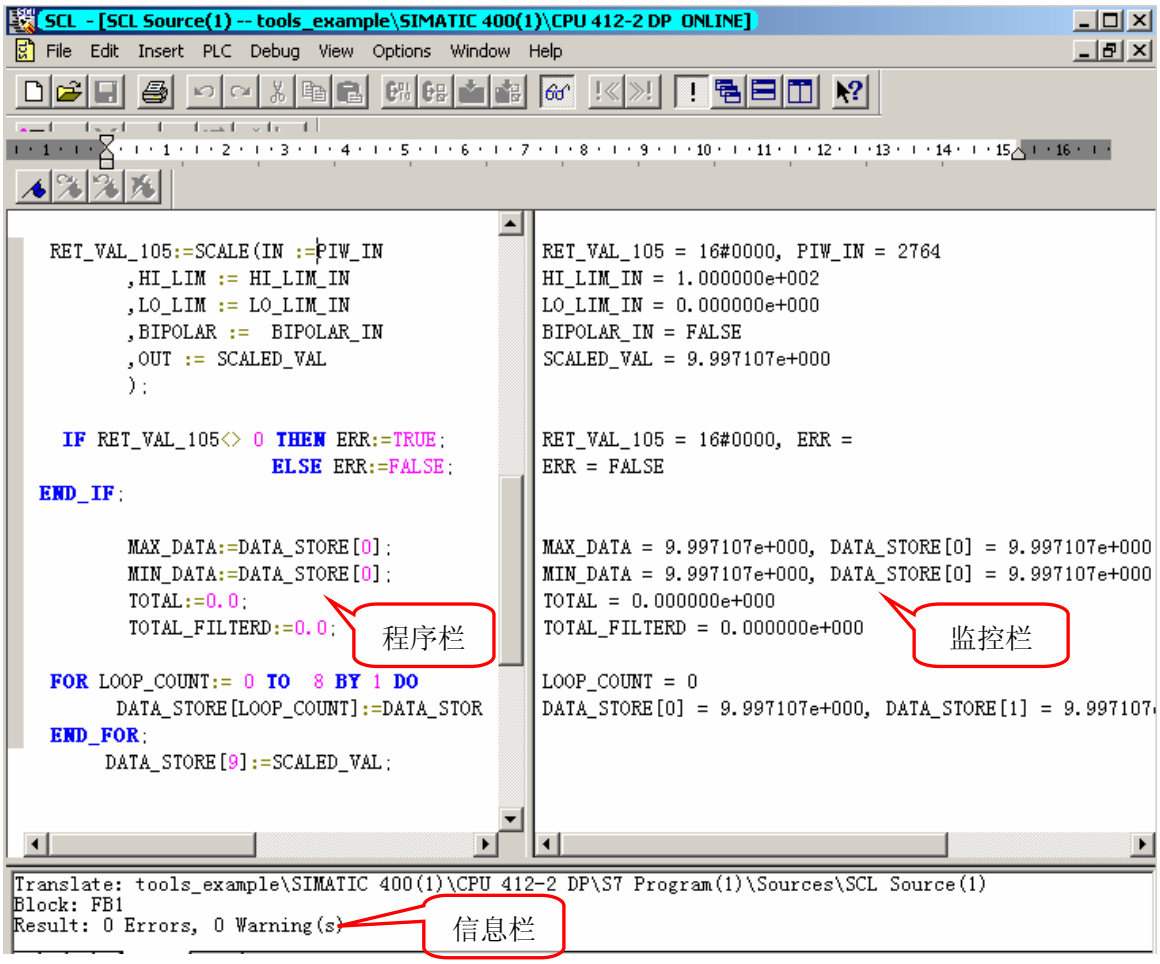


图 3-16: 监控 FB1

至此，一个简单的 S7-SCL 程序示例就结束了，本文中仅是对其非常简单的作了介绍。任何编程语言都有其复杂性，并非一朝一夕就可掌握，关于 S7-SCL 的具体使用，请按照本文提供的的地址连接下载 S7-SCL 手册。

3.2.1. 重要提示：

- ✧ 本文的虚拟工程与真实工程实例有重大差别，示例中并未遵循规范的工程设计流程进行编程，请读者切勿将其与工程实例相混淆。
- ✧ 由于此例子是免费的，任何用户可以免费复制或传播此程序例子。程序的作者对此程序不承担任何功能性或兼容性的责任，使用者风险自负。
- ✧ 西门子不提供此程序例子的错误更改或者热线支持。

4. S7-SCL 常见问题

4.1. 程序编写相关问题

4.1.1. 问题：S7-SCL 支持哪些 STEP7 块，与其它语言有什么关系？

问题：S7-SCL 支持哪些 STEP7 块，与其它语言有什么关系？

解答：S7-SCL 支持如下 STEP7 的块： OB,FC,FB,DB,UDT。在 S7 程序中，S7-SCL 块可以与其它 STEP7 编程语言生成的块互相调用。S7-SCL 可以被编译成 STL， S7-SCL 生成的块也可以作为库文件被其它语言引用。由于 S7-SCL 程序由 ASCII 文本构成，所以它非常容易被导入或导出。

4.1.2. 问题：如何规范地的开发一个 S7-SCL 程序？

问题：如何规范地的开发一个 S7-SCL 程序？

解答：开发一个 S7-SCL 程序，应当遵循如下流程：

- 规划需要的块类型，即程序的整体结构。如：是否仅仅一个 FC 就可满足要求，还是需要生成其它 OB, FB 等
- 规划子任务（FB,FC 等等）
- 定义各个子任务的之间输入/输出接口
- 定义各个子任务在原文件中的顺序及调用关系
- 定义符号表
- 编译检查
- 下载调试

4.1.3. 问题：如何加密我的 S7-SCL 程序？

问题：如何加密我的 S7-SCL 程序？

解答：S7-SCL 程序加密与 STL 程序加密方法几乎一样，用户可以在程序中加入关键字：KNOW_HOW_PROTECT，再进行编译即可，下图中的 FC1 在编译后，即出现了加锁保护的图标 。

注意：提供给最终用户的项目中应当删除 S7-SCL 源代码，否则用户在双击加密的块后（例如下图中的 FC1），依然可以关联打开 S7-SCL 源代码。

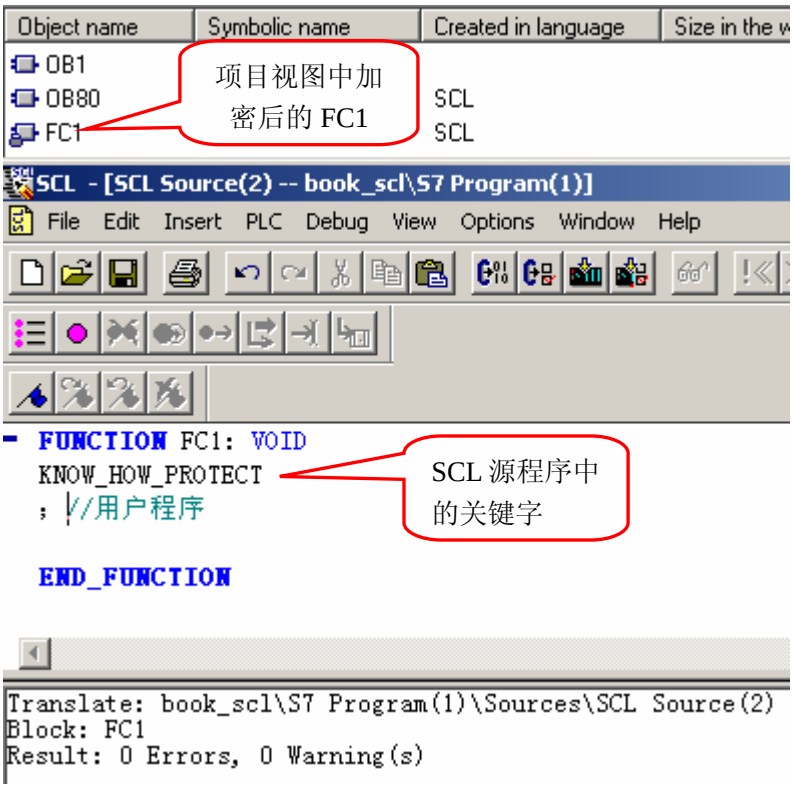


图 4-1：S7-SCL 源程序加密

4.1.4. 问题：为什么双击打开 S7-SCL 类型的程序后，显示的格式却是 STL 格式？

问题：为什么双击打开 S7-SCL 类型的程序后，显示的格式却是 STL 格式？

解答：S7-SCL 程序编译后生成的执行代码实际上为 STL 格式，如下图中的 FC1，虽然其标示为 SCL 格式，但如果用户将 Source 目录中的 FC1 的源文件删除后，再双击打开 FC1 后，显示的将为 STL 格式。

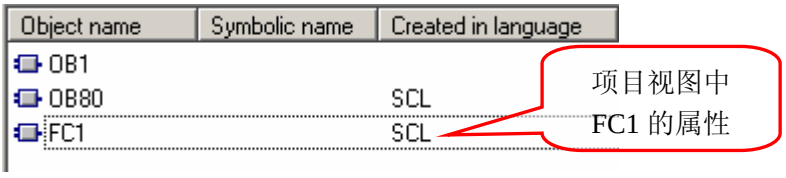


图 4-2：块的 SCL 属性

注意：提供给最终用户的项目中如果删除 S7-SCL 源代码，用户将只能看到 STL 格式的程序，稍微复杂些的 S7-SCL 程序在 STL 格式下几乎是无法阅读的。此方法类似于加密的效果。

4.1.5. 问题：什么是 OK Flag？

问题：什么是 **OK** Flag？

解答：在 S7-SCL 程序中，**OK** 是一个布尔类型的特殊本地变量。它用来显示块执行是否正确。**OK** 标志的使用与 STL 编程中对于 FC/FB 的 ENO 处理（STL 是用状态字的 BR 位来保存状态）

非常相似。在程序的开始，可以将 **OK** 标志赋值为 **TURE**,如果程序执行有错误（例如 0 作为除数），则当程序退出时，用户通过程序将 **OK** 标志改写为 **FALSE**，此数值将被存储在输出参数 ENO 当中。**OK** 标志是一个系统变量，不需要定义，如果用户希望在程序中使用此标志，可以在 S7-SCL 编程菜单 **Options>Customize>Compiler** 中，应当选择 **Set OK flag.**项。否则 ENO 将无法被评估。

OK 标志使用的例子：

```
OK:= TRUE;
Division:= 1 / IN;
IF OK THEN      // 除法操作正常
    ;// 此处可以添加在除法操作正常情况下，其它的用户程序
ELSE            //除法操作不正常
    ;// :
END_IF;
```

4.1.6. 问题：S7-SCL 中读取一个组织块的信息

问题：在 STL 中可以读取一个组织块的信息 (例如 OB1 中的循环时间)，在 S7-SCL 中是否存在同样的可能？

解答：STEP7 中的任意 OB 块在生成之后，其内部都自动生成一些特有的临时变量，例如 OB1 中的变量 OB1_PREV_CYCLE 即为 CPU 上个扫描周期所用的时间。在 S7-SCL 编辑器中，通过“Insert > Block Template > OB”创建一个 OB 模板。在缺省情况下，此模板的临时变量区域将创建一个 ARRAY OF BYTE 类型的临时变量。此区域与 OB 块的临时变量区域是相对应的，但由于没有变量名称标识，并且数据类型也不相同，所以用户使用起来非常不方便。如果用户想要使用 S7-SCL 格式的 OB 块内的特有变量，请按下列步骤进行：

- 在路径 STEP7 的安装路径 \Step7\S7DATA\S7wiz 下，包含所有 OB 块的 STL 格式模板，可以使用写字板打开这些模板 (例如 OB31.awl)， 并复制相应变量定义。
- 创建一个新的 S7-SCL 源文件并插入刚才复制的变量定义。
- 调整 S7-SCL 源文件中的语法结构
- 在这个程序的开始部分， 可以设置 "Title" 和"Version"，例如 Title=' ' 和 Version= '0.1'
- 在 BEGIN 语句后面插入一个分号，至此，即完成格式修改。

另外，也可以打开本文中的 S7-SCL 例子目录中的 OB_Templ 项目，其中包括了 44 个 S7-SCL 源文件格式的组织块。

4.1.7. 问题：如何理解 FC/SFC 的输出参数 RET_VAL（返回值）？

问题：如何理解 FC/SFC 的输出参数 RET_VAL（返回值）？

解答：对于此问题，如果读者对 PASCAL 语言或者高级编程语言非常熟悉，可能此问题就非常容易解释。许多 SFC (系统功能) 都有输出参数 RET_VAL (返回值) ，它提供一个可供评估的错误代码。STEP 7 在线帮助中提供有更多关于系统功能和输出参数 RET_VAL 的信息。通过在 SIMATIC 管理器中选中 SFC 然后按“F1”键可以获取相关的在线帮助信息。

同样，一个 FC 也可以返回一个结果，下面给出了一些例子，说明了如何在 S7-SCL 中调用带有返回值 (RET_VAL) 的功能 (FC 或 SFC)。

定义 FC1, 函数类型为 INT （返回值）

```
FUNCTION FC1: INT

VAR_INPUT
    a: INT;
    b: INT;
END_VAR

IF a>b THEN FC1:= 1;
END_IF;

IF a<b THEN FC1:= 10;
END_IF;

END_FUNCTION
```

图 4-3：带有返回值 FC 的定义

调用 FC1,得到返回值：

```
FUNCTION FC2: VOID

VAR
    c: INT;
END_VAR

BEGIN
c:= FC1 (a:=1, b:= 2);
END_FUNCTION
```

用户可以根据 FC 的返回值判断 FC 的运行结果

图 4-4：调用带有返回值的 FC

调用一个系统功能 (SFC24)：当调用系统功能 SFC 24“TEST_DB”(测试数据块) 时，得到关于 CPU 主内存中一个数据块的信息。SFC24 确定指定 DB 的数据字节数，并检查该 DB 是否受到写保护。

选定 DB 所包含的数据字节数通过参数“DB_LENGTH”输出，参数“WRITE_PROT”包含关于

选定 DB 的写保护 ID 的信息 (FALSE 表示没有写保护)。

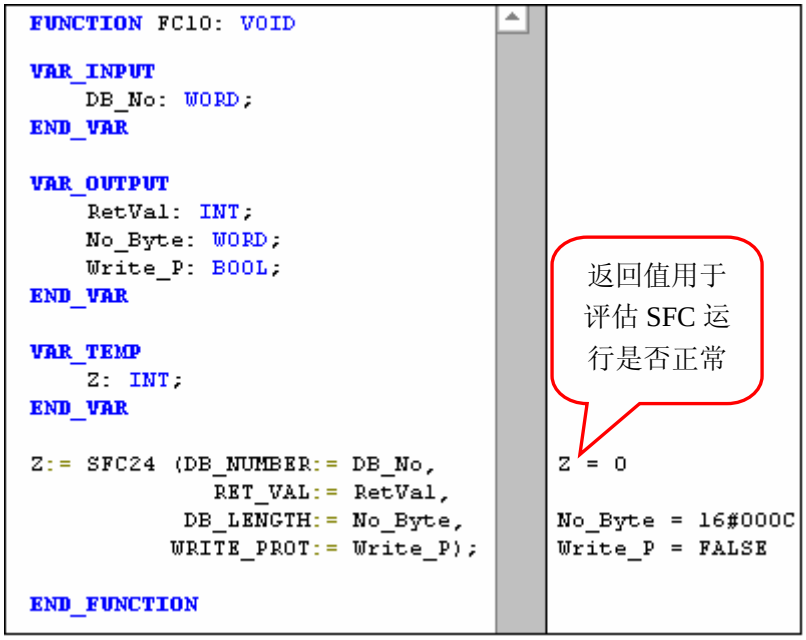


图 4-5：调用带有返回值的 SFC

4.1.8. 问题：在 S7-SCL 程序中，PLC 中的地址标识使用的注意事项

问题：在 S7-SCL 程序中，PLC 中的地址标识与 STL/LAD 中的地址标识有何区别，使用中有何注意事项？

解答：在 S7-SCL 程序中，PLC 中的地址标识与 STL/LAD 中的地址标识基本相同，方便了编程人员快速掌握 S7-SCL 语言。下图为 PLC 中的 S7-SCL 程序地址标识格式：

图中一些格式有些特殊，实际上，下面的格式在 S7-SCL 中都是正确的。

```
DB1.DO.0:=DB1.DBX0.1;
DB1.DB1:=DB1.DBB1;
DB1.DW2:=DB1.DBW2;
DB1.DD4:=DB1.DBD4;
QX0.0:=Q0.0;
IX0.0:=I0.0;
MX0.0:=M0.0;
```

所以说，在一般情况下，编程人员依旧可以 STL 的标识方式用于 S7-SCL。

注意：如果数据块编号及地址都使用了间接寻址方式，则必须使用 S7-SCL 特定的标识方式，例如

如下语句：

STATUS_1:=WORD_TO_BLOCK (INDEX).**DW** [COUNTER]; (此格式正确)

STATUS_1:=WORD_TO_BLOCK (INDEX).**DBW** [COUNTER]; (此格式错误)

Mnemonic (internat.)	addresses	Data Type
Qx,y	Ouptut (via the process image)	Bit
QBx	Output (via process image)	Byte
QDx	Output (via process image)	Double word
QWx	Output (via process image)	Word
QXx.y	Output (via process image)	Bit
Dx.y	Data block	Bit
DBx	Data block	Byte
DDx	Data block	Double word
DWx	Data block	Word
DXx.y	Data block	Bit
Ix.y	Input (via the process image)	Bit
IBx	Input (via process image)	Byte
IDx	Input (via process image)	Double word
IWx	Input (via process image)	Word
IXx.y	Input (via process image)	Bit
Mx.y	Memory bit	Bit
MBx.y	Bit memory	Byte
MDx	Bit memory	Double word
MWx	Bit memory	Word
MXx	Bit memory	Bit
PQBx	Output (Direct to peripherals)	Byte
PQDx	Output (Direct to peripherals)	Double word
PQWx	Output (Direct to peripherals)	Word
PIBx	Input (Direct from peripherals)	Byte
PIDx	Input (Direct from peripherals)	Double word
PIWx	Input (Direct from peripherals)	Word

图 4-6：S7-SCL 中的 PLC 地址格式

x 可以为 0 至 65535 之间的数字（绝对地址）

y 可以为 0 至 7 之间的数字（位地址）

对于其它 FB,OB,DB, T,C 等等的标识格式，S7-SCL 格式基本与 STL 格式相同，此处不做说明。

4.1.9. 问题：为什么用户人工输入的程序在编译时经常提示语法错误？

问题：为什么用户人工输入的程序在编译时经常提示语法错误？

解答：S7-SCL 程序和所有的高级语言一样，有着自己特定的语法，而很多编程人员容易在人工输入程序时忽视语法格式，这样就造成编译错误。因此强烈建议编程人员使用 S7-SCL 的模板向导。

在菜单 **Insert** 中，提供了 **Block Call, Block Template, Control Structure,** 三个向导，用户可以通过使用此向导，提高输入效率。例如在下图中指出了两个常见错误，用户如果使用向导，就不容易出错误：

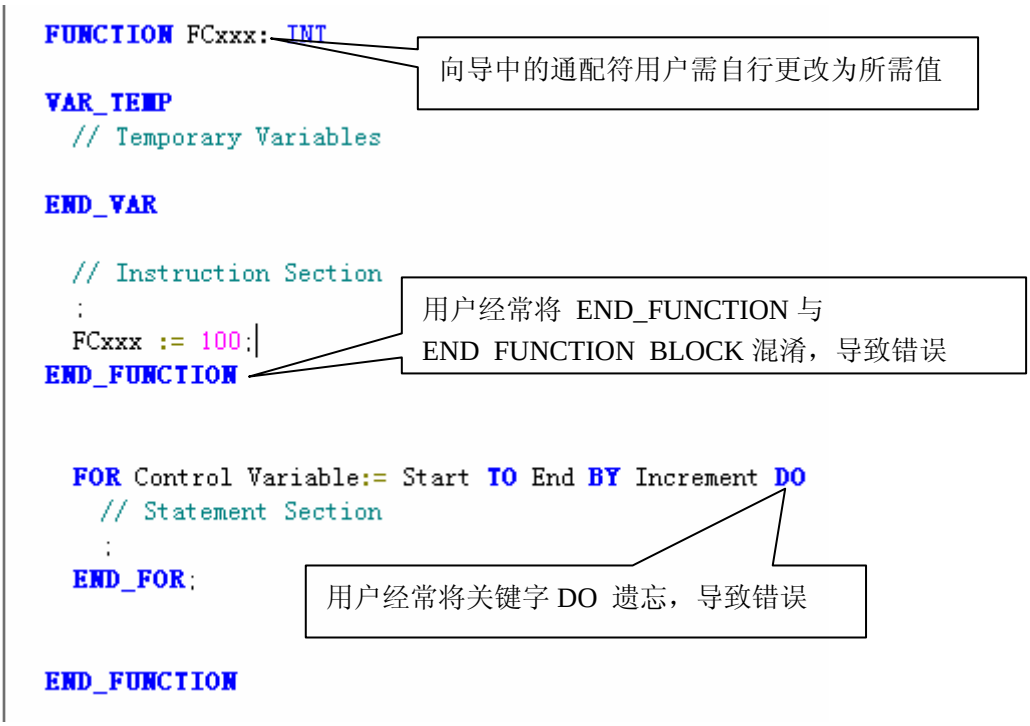


图 4-7：S7-SCL 常见的录入格式错误

4.1.10. 问题：如何对 S7-SCL 程序中的数据块及静态变量初始化？

问题：如何对 S7-SCL 程序中的数据块及静态变量初始化？

解答：S7-SCL 程序中对数据块及静态变量初始化和 STL 语言格式类似，下面的程序列出了正确格式：

```
DAT1 : REAL := 100.5;
A1 : INT := 10 ;
A2 : STRING[6] := 'FACTOR';
A3 : ARRAY[1..12] OF REAL := 0.0, 10(100.0), 1.0;
```

4.1.11. 问题：什么是“AT”指令，如何使用？

问题：什么是“**AT**”指令，如何使用？

解答： **AT** 指令是 S7-SCL 中特有的一个指令，初学者可能会忽略此指令，或者对其理解不够深刻。**AT** 指令在 S7-SCL 中有着非常重要的地位，很多应用程序都必须使用此指令。下面将对此指令进行详细讲解。

AT 指令可以使用户能够以不同的数据类型来访问一个已经声明的变量。此指令类似于高

级编程语言中的“继承”概念。此指令有如下特点：

- 特点一：定义仅在块内有效，在接口参数中不出现
- 特点二：在块中可以使用其它数据类型的视图
- 特点三：它只是让另外一些变量继承了某个变量的特性，仅仅数据类型是新的，其它特性相同。

下面将通过具体程序对以上三个特点进行说明，首先定义 UDT100,UDT200,大小都为 12 个字节，但结构不同。UDT100 包括一个 DATE_AND_TIME 数据类型（8 个字节），还包括一个实数（4 个字节）；UDT200 包括 6 个字（12 个字节）；在 FC2 定义输入参数 Buffer, Frame1,Frame2：

特点一：定义仅在块内有效，在接口参数中不出现。块内定以的输入参数有 3 个，但仅在块内有效，在接口参数仅出现 Buffer 输入参数

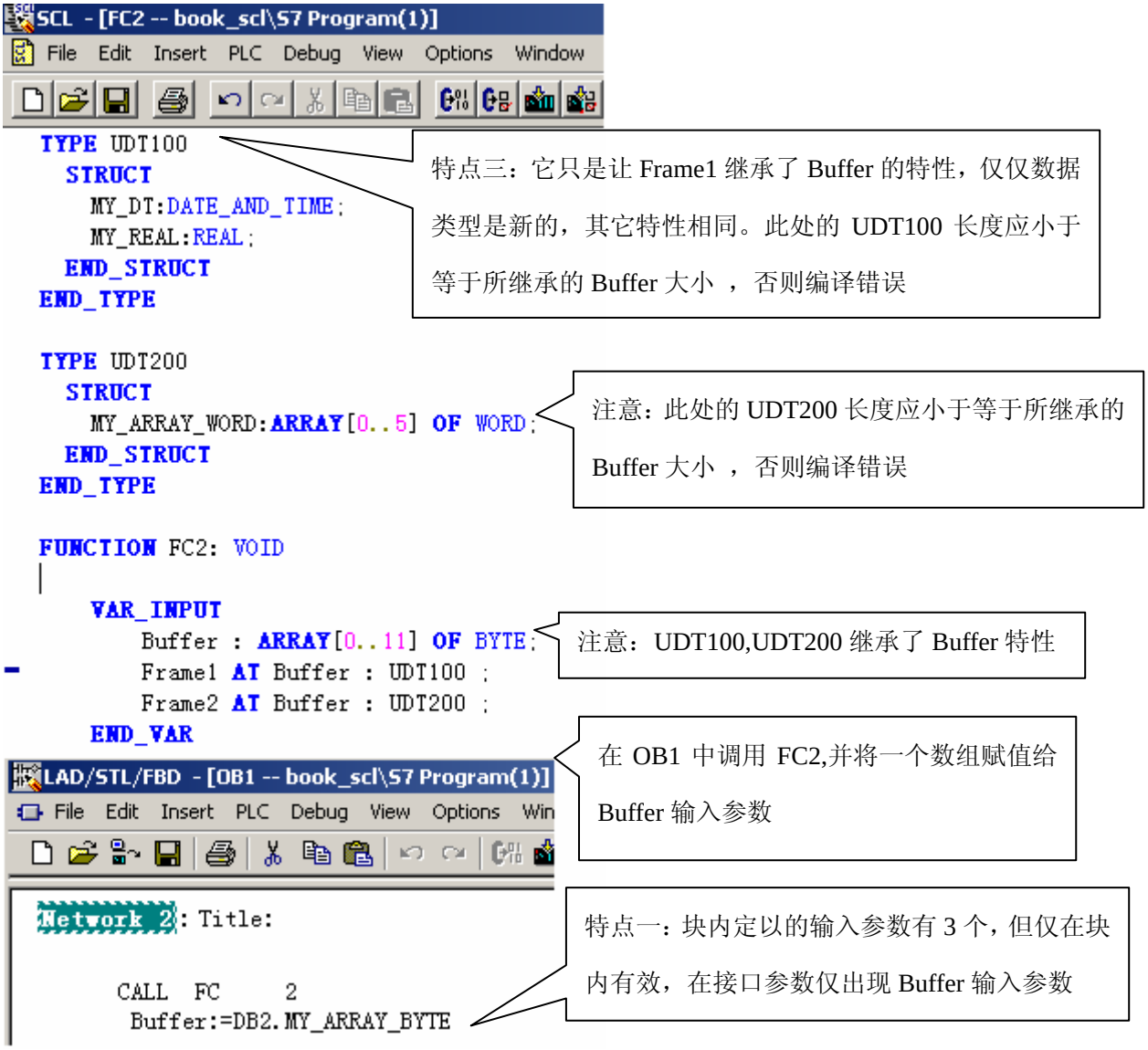


图 4-8：AT 指令特点说明(a)

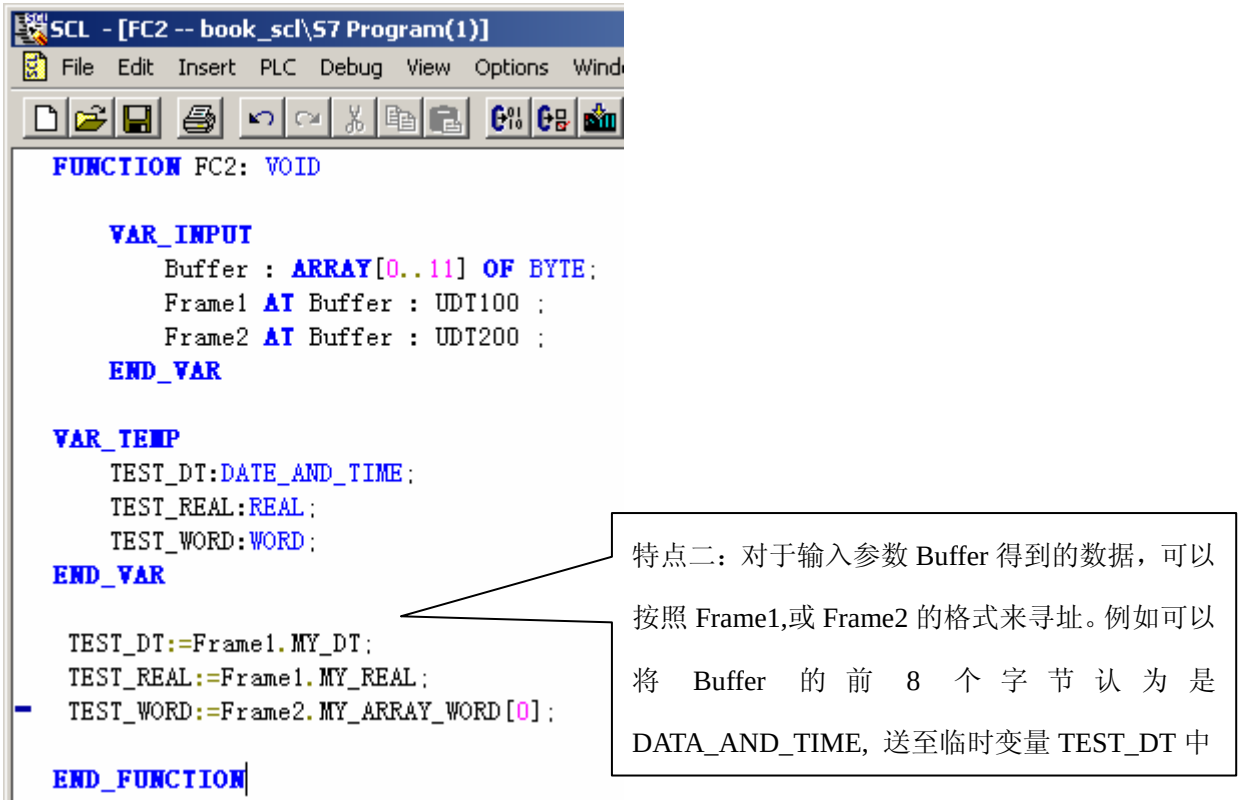


图 4-9：AT 指令特点说明(b)

AT 指令应用例子 1：在 STL 中，用户如果希望取得 MB0 的某一位的数值是非常容易的，例如第 3 位，直接寻址 M0.2 就可以了。而在 S7-SCL 中，由于其具备高级语言的特点，所以定义的变量的绝对地址一般是不显现的。因此在 STL 中简单的操作，在 S7-SCL 中却复杂起来。但是通过使用 **AT** 指令即可解决这个问题。下图中的例子中，当输入的某个字节，其输出依次为此字节的 8 个位。

```
FUNCTION_BLOCK fb123
VAR_INPUT
    MainVarByte: BYTE;
    FilterVar AT MainVarByte: ARRAY[0..7] OF BOOL;
END_VAR
VAR_OUTPUT
    X0: BOOL;
    X1: BOOL;
    X2: BOOL;
    X3: BOOL;
    X4: BOOL;
    X5: BOOL;
    X6: BOOL;
    X7: BOOL;
END_VAR

X0:= FilterVar[0];
X1:= FilterVar[1];
X2:= FilterVar[2];
X3:= FilterVar[3];
X4:= FilterVar[4];
X5:= FilterVar[5];
X6:= FilterVar[6];
X7:= FilterVar[7];
END_FUNCTION_BLOCK
```

定义输入参数，使用 AT 指令，将输入继承到以 BOOL 数据类型组成的数组中

定义输出参数

以 BOOL 方式输出

图 4-10：AT 指令应(a)

AT 指令应用例子 2：在 S7-SCL 中，ANY 数据类型的使用，经常会伴随着 **AT** 指令的使用，下面的例子演示了如何拆分改写 ANY 数据类型中的内容。程序执行结果相当于：

```
CALL    "BLKMOV"
  SRCBLK :=P#DB1.DBX 0.0 BYTE 10
  RET_VAL:=MW0
  DSTBLK :=P#DB1.DBX 0.0 BYTE 10
```

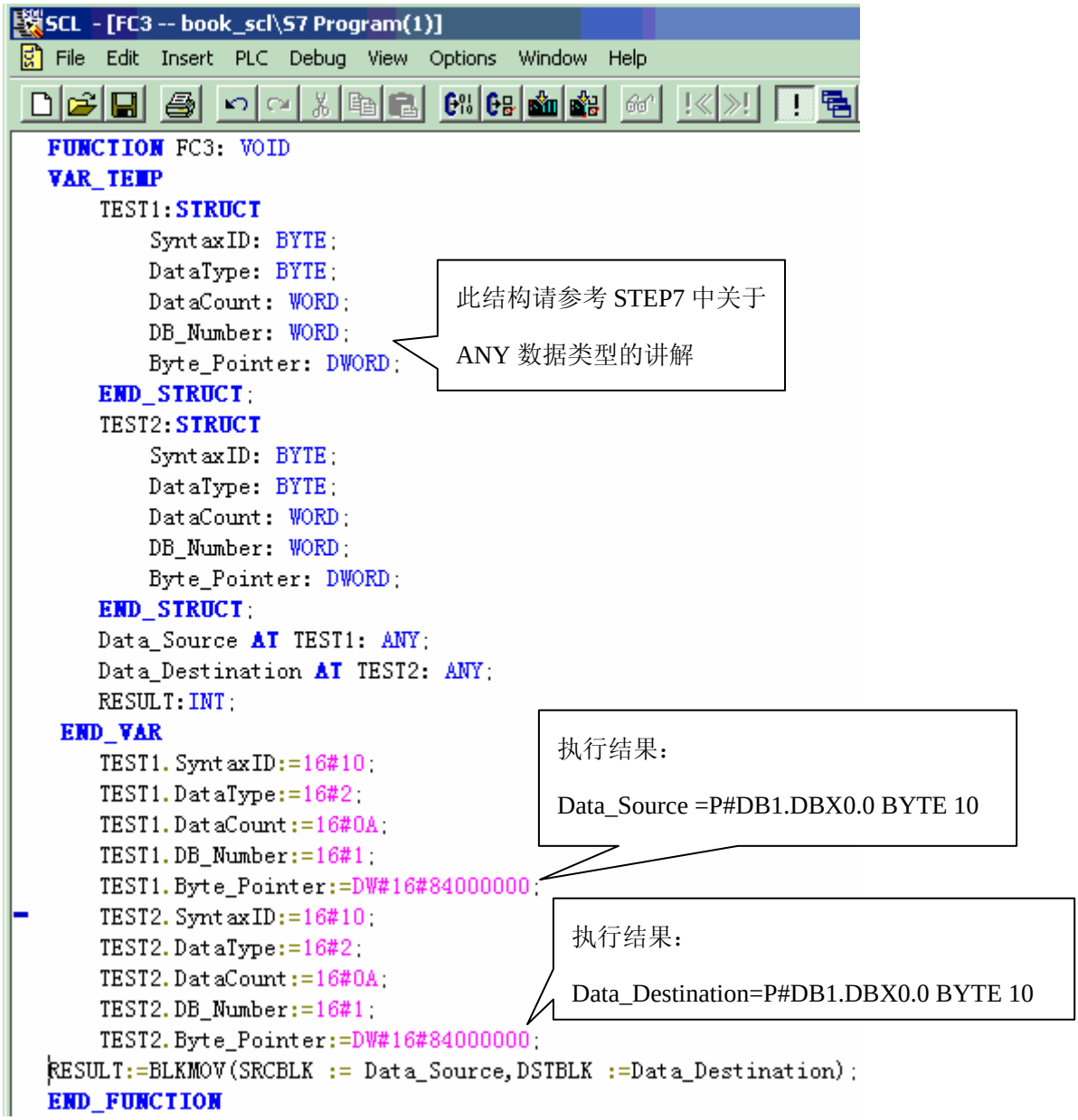


图 4-11：AT 指令应用(b)

AT 指令在使用当中，还有如下注意事项：

- AT 指令用于对某个变量进行其它数据类型的声明时，必须放在此变量的声明语句后面
- AT 指令生成的变量不可以初始化
- AT 指令用于对某个变量其它数据类型的声明时，占用内存大小要小于等于此变量的长度
- AT 指令用于对某个变量其它数据类型的声明时，如下的组合是允许的：

		视图数据类型	变量数据类型		
			Elementary	Complex	ANY/POINTER
FB	声明类型 VAR, VAR_TEMP, VAR_IN, VAR_OUT	Elementary Complex ANY/POINTER	x x	x x x(1)	x(1)
	声明类型 VAR_IN_OUT	Elementary Complex ANY/POINTER	x	x	
FC	声明类型 VAR, VAR_TEMP	Elementary Complex ANY/POINTER	x x	x x x	x
	声明类型 VAR_IN, VAR_OUT, VAR_IN_OUT	Elementary Complex ANY/POINTER	x	x	

图 4-12：AT 指令使用限制

（1）AT 指令用于 ANY 类型在 VAR_OUT 中是不允许的

Elementary (基本数据类型包括) ： BOOL, BYTE, WORD, DWORD, INT, DINT, DATE, TIME, S5TIME, CHAR

Complex (复杂数据类型包括) ： ARRAY, STRUCT, DATE_AND_TIME, STRING

4.1.12. 问题：S7-SCL 程序中如何使用多重背景？

问题：S7-SCL 程序中如何使用多重背景？

解答：S7-SCL 程序中可以使用多重背景，下面的程序列出了正确格式，声明完成后，在程序中即可调用：

```
Supply1 : FB10;                    //将 Supply1 定义为 FB10 类型
Supply2,Supply3,Supply4 : FB100;   //将 Supply2,Supply3,Supply4 定义为 FB100 类型
```

4.1.13. 问题：如何在 S7-SCL 程序中实现数据块间接寻址？

问题：如何在 S7-SCL 程序中实现数据块间接寻址？

解答：S7-SCL 程序中对数据块间接寻址提供了良好的支持，编程也很简单。下面的程序列出了正确格式：

```
STATUS_1:= DB11.DW[COUNTER];              //字节间接寻址
STATUS_2:= DB12.DX[WNO, BITNO];           //位间接寻址，用户改变 WNO,BITNO 数值即可
STATUS_1:= Database1.DW[COUNTER];         // Database1 为 DB 类型的本地变量
STATUS_2:= Database2.DX[WNO, BITNO];
STATUS_1:= WORD_TO_BLOCK_DB(INDEX).DW[COUNTER];
```

//INDEX 被定义为 **BLOCK_DB** 数据类型，COUNTER 为整数数据类型，这样可以实现/数据块编号，字节地址同时间接寻址，此功能是一种功能很强间接寻址方式。

```
//以下是数据块直接寻址方式格式，仅供参考。
STATUSBYTE :=DB101.DB10;                  //字节直接寻址格式
STATUS_3 :=DB30.D1.1;                     //位直接寻址格式
MEASVAL :=DB25.DW20;                      //字直接寻址格式
STATUSBYTE :=Status_data.DB10;
STATUS_3 :="New data".D1.1;
MEASVAL :=Measdata.DW20;
STATUS_1 :=WORD_TO_BLOCK_DB (INDEX).DW10;
```

4.1.14. 问题：在 S7-SCL 程序中调用 FC/FB 与在 STL/LAD 中有何区别？

问题：在 S7-SCL 程序中调用 FC/FB 与在 STL/LAD 中调用 FB/FC 有何区别？

解答：在 STL/LAD 程序中调用 FB 可以不把参数填写完整，但在 S7-SCL 程序中调用 FB 时，必须把 FB 参数填写完整;对于 FC 的调用 S7-SCL 与 STL/LAD 区别不大。强烈建议编程人员使用 S7-SCL 的模板向导调用 FB。下图的程序说明了两者的区别：

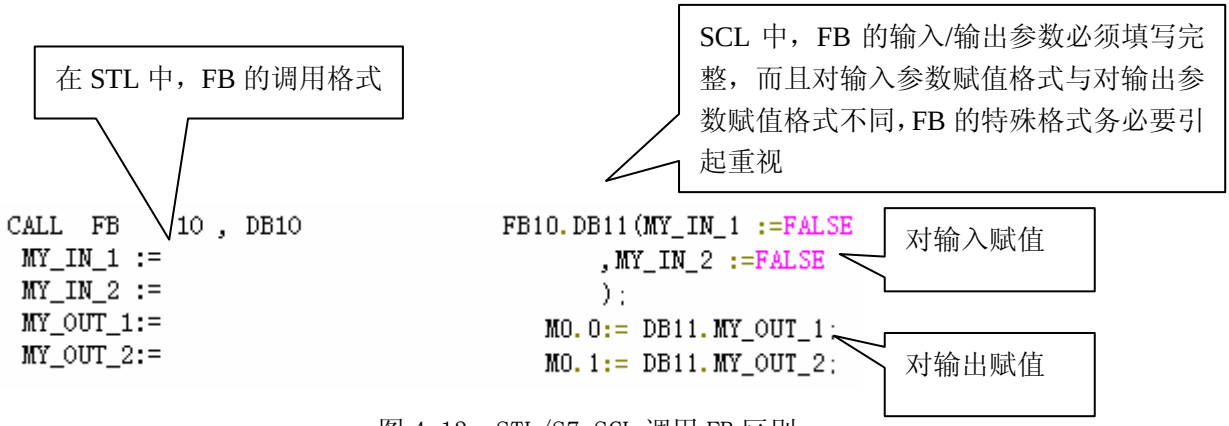


图 4-13：STL/S7-SCL 调用 FB 区别

4.1.15. 问题：转换到“REAL”数据类型需要使用哪种数据类型转换程序？

问题：为了将 S7-SCL 中的位数据类型(BYTE, WORD 或 DWORD)转换到“REAL”数据类型时，需要使用哪种数据类型转换程序？

解答：为了将 S7-SCL 中的位数据类型(BYTE, WORD 或 DWORD)转换到数字数据类型“REAL”时，需要使用以下指令：

- 从 BYTE 到 REAL：“DINT_TO_REAL(DWORD_TO_DINT(BYTE_TO_DWORD(value))”
- 从 WORD 到 REAL：
  “DINT_TO_REAL(DWORD_TO_DINT(WORD_TO_DWORD(value))”
- 从 DWORD 到 REAL：
  “DINT_TO_REAL(DWORD_TO_DINT(value))”

注意事项：用户在将 BYTE, WORD 或 DWORD 数据类型转换到 “REAL” 数据类型时，S7-SCL 中常见的错误可以参考下例：

从 WORD 到 REAL：“DWORD_TO_REAL(WORD_TO_DWORD(value))”(错误程序)。

如果执行了这一转换，会得到错误的结果。

原因：DINT_TO_REAL 函数将把源数据转换并按照 IEEE REAL 的格式存储到目的变量，而 DWORD_TO_REAL 仅仅是从源数据拷贝位串至目的变量。

因此，为了获得正确的“REAL”数，必须总是通过中间步骤“DWORD_TO_DINT”和“DINT_TO_REAL”实现。

4.1.16. 问题：在 S7-SCL 中如何区分变量名是本地变量，还是符号名？

问题：在 S7-SCL 中如何区分某个变量名是本地变量，还是符号名？

解答：S7-SCL 中本地变量与符号名的引用有如下区别：

- 如果引用中只有变量名，则此变量为本地变量，如：Motor_Status:= Motor_1
- 如果引用中变量名称上包括“”，则此变量为符号名，如：Motor_Status:= “Motor_1”

4.1.17. 问题：如何访问一个字符串中的单个字符？

问题：如何访问一个字符串中的单个字符？

解答：对于字符串中的单个字符，使用如下的访问格式，是错误的：

MB10 ：= str[5] 这样的表达式无法实现将字符串的第 5 个字符送至 MB10 中。

方法：可以使用 IEC 函数 MID 来得到字符串的某个部分，例子如下：

```
VAR
str : STRING[20];
END_VAR
MB10:=CHAR_TO_BYTE(STRING_TO_CHAR(MID(IN:=str,L:=1,P:=5)));
```

4.2. 程序优化相关问题

4.2.1. 问题：如何在访问结构时优化运行时间？

问题：如何在访问结构时优化运行时间？

解答：如果需要多次访问一个结构，可在 S7-SCL 代码中也创建一个同样类型的临时变量来优化运行时间。用户可在临时变量声明中生成此变量，并在程序中多次使用它。 假设用户希望完成如下操作：

DB4.Field.Value:= DB4.Field.Value1*DB4.Field.Value2*DB4.Field.Value3;

可见以上操作数都存在于 DB 的结构当中，在不进行优化的情况下，这段程序在运行时，将多次打开数据块，进行多次读取操作，程序运行所需时间较多。

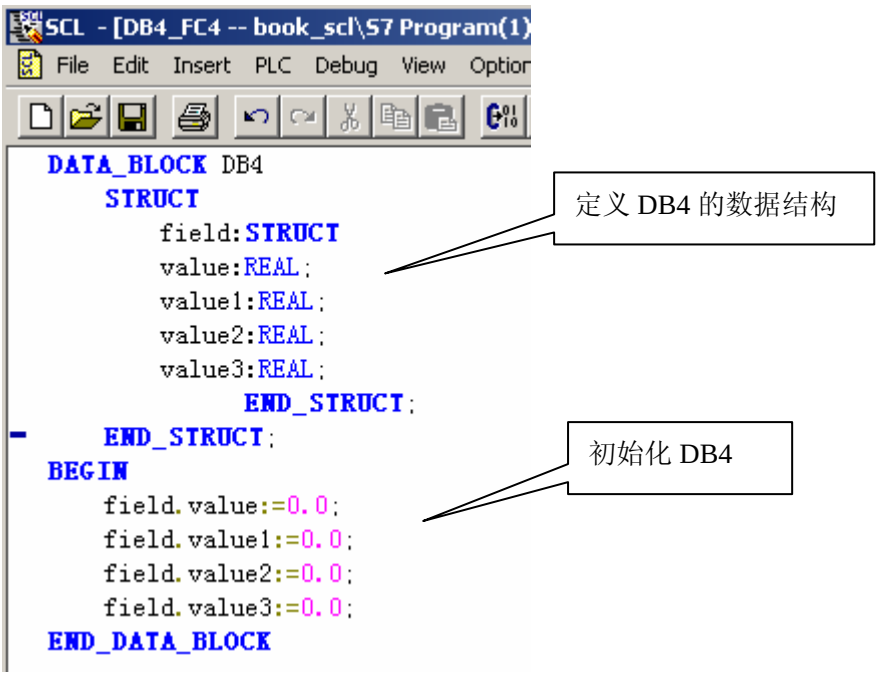


图 4-14：定义数据块

解决方法：在 FC4 中声明一个与 DB4 中“field”变量结构相同的临时变量(例如“Temp”)。然后使用如下语句从 DB4 中将数据拷贝到临时存储区内：

- Temp:= DB4.field;

对以上所描述的语句，可将 DB 变量“DB4.Field”替代为临时变量“Temp”：

- DB4.field.Value:= Temp.Value1*Temp.Value2*Temp.Value3;

此时数据块中的数值被复制到 CPU 的 L 堆栈当中，在后续程序中，可以使用这些 L 堆栈中的临时变量替代原来数据块中的变量，由于 CPU 对 L 堆栈的访问速度远快于对数据块的访问速度，这样就实现了数据访问的优化。

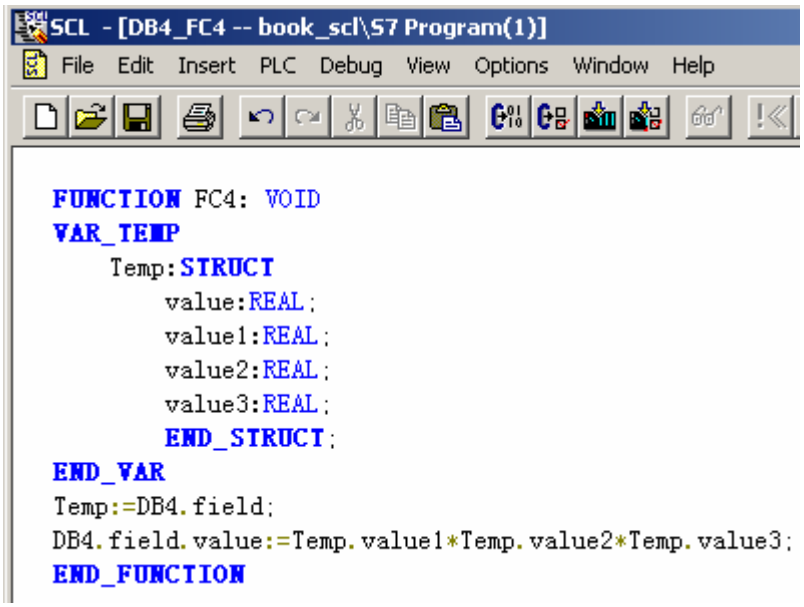


图 4-15：优化数据块访问

注意事项：变量声明“VAR_TEMP”存储在 CPU 的堆栈中。这可能会在小型 CPU 模块中导致堆栈溢出。因此使用临时变量要慎重。

4.2.2. 问题：如何用布尔型变量优化 IF 语句来缩短循环时间？

问题：如何用布尔型变量优化 IF 语句来缩短循环时间？

解答：在优化 IF 语句后，S7-SCL 代码将变得更为紧凑，这样代码序列的处理就更快。程序计算的布尔类型数据相同，优化后事件仅分配给一个变量。相对于简化格式，IF 语句完整格式编程时需要加载另一个跳转和一个常量 (TRUE 或 FALSE) 。

示例 1：

当一个模拟量值(REAL 变量)超过限定(此处是 100)时设置一个比特位。

函数“IF_Test”包含了查询限定值的 IF 语句。整个 IF 语句可用以下语句代替：

IF_Test:= (100 < value) 。

```
FUNCTION IF_Test : BOOL

VAR_INPUT
  value : REAL;
END_VAR

BEGIN
IF 100 < value THEN
  IF_Test    := true;
ELSE
  IF_Test    := false;
END_IF;

END_FUNCTION
```

IF 语句的完整格式

```
FUNCTION IF_Test : BOOL

VAR_INPUT
    value: REAL;
END_VAR

BEGIN
IF_Test:= (100 < value);
END_FUNCTION
```

IF 语句的紧凑格式

图 4-16：IF 语句不同格式(a)

示例 2：

查询在一个 WORD 变量中指定比特位 X 是否置位。

这里函数 BitX 的 IF 语句由以下语句代替：BitX:= status = 16#0004。

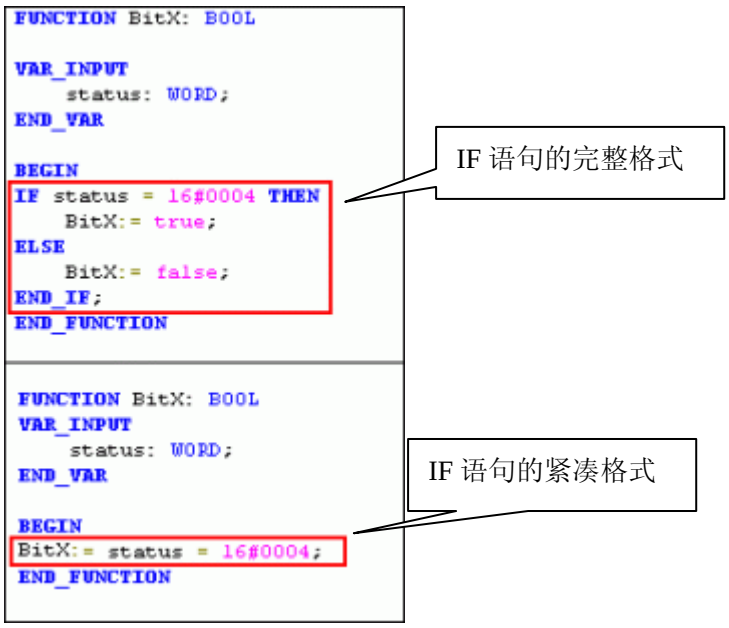


图 4-17：IF 语句不同格式(b)

注意事项：紧凑的 IF 语句加快了程序处理速度，但是优化处理可能使得程序变得不够清晰和难于理解。

4.3. 编译错误相关问题

4.3.1. 问题：S7-SCL 程序在别人的计算机上打不开或无法编译？

问题：为什么我的 S7-SCL 程序在自己的计算机上编译正常，在别人的计算机上却打不开或无法编译？

解答：应当确认其它计算机是否安装了 S7-SCL 环境，而且要检查 S7-SCL 环境版本，例如：S7-SCL V5.3 SP1 提供了新的函数（SEL,MAX,MIN,LIMIT,MUX,BYTE_TO_INT,INT_TO_BYTE 等等），旧版本无法识别这些函数。如果用户程序中使用了这些函数，在旧版本的 S7-SCL 环境下无法通过编译。

4.3.2. 问题：为什么我的程序与别人的程序完全一致，却通不过编译？

问题：为什么我的程序与别人的程序完全一致，却通不过编译？

解答：除了前面问题所需注意的事项外，符号表是初学者经常忽略的因素，这里额外单独强调一下。用户在编写 S7-SCL 程序源代码时，对符号进行良好的定义，可以使编程清晰易懂。而初学者经常仅仅复制其他人的源代码，却忘记复制符号表，以至于在 S7-SCL 程序中经常出现不清楚的名称定义，从而导致编译错误。

4.3.3. 问题：包含比较类型为 WORD/DWORD 的变量的 IF 语句不能被编译通过？

问题：为什么包含比较类型为 WORD/DWORD 的变量的 IF 语句不能被编译通过？

解答：当在 IF 指令内比较操作数时，如果变量声明为 WORD 和 DWORD 类型，S7-SCL 仅允许比较操作“=”和“<>”，而不允许比较操作“>”和“<”。作为一种补救措施，建议将变量类型声明为 DINT。 如果变量的类型不能声明为 DINT，在此比较之前则必须首先进行类型转换。

4.3.4. 问题：当给一个双字类型变量分配了一个实型数值时，出现非法数据类型错误

解答：当给一个双字类型变量分配了一个实型数值时，出现"Invalid data type"消息，是由于数据类型不匹配造成的。注意仅当数据类型为 BOOL, BYTE, WORD 和 DWORD 时，允许对 DB 块的绝对地址访问。

下图举例说明何时数据类型必须转换.

```
FUNCTION FC7: INT

VAR_TEMP
     i: int;
     h: real;
     x: dword;
END_VAR

i:= 0;
h:= 50.0;

//Conversion from Real to DWORD.
DB1.DD[0]:= real_to_dword(h);

//not valid data type, must be converted.
// DB1.DD(4):= h;

DB1.DD[4]:= x;
//this is correct, because x is a DWORD.

//x:= h;
//not valid data type, must be converted.

x:= REAL_TO_DWORD(h);
//Conversion from Real to DWORD.

FC7:= 100;
END_FUNCTION
```

图 4-18: 数据类型转换

4.3.5. 问题：在输出窗口中的错误消息与程序行数字不符

问题：在输出窗口中的错误消息与程序行数字不符

解答：当 S7-SCL 程序行数超过 65535， S7-SCL 编译器把行计数器重新从 1 开始计数。例如，在

S7-SCL 源程序中，一个编程错误出现在第 65537 行，以下行数字将被输出：

"F: L 00001 error xyz"，而不是 "F: L 65537 error xyz"

解决方法:双击 S7-SCL 编译器输出窗口中的错误消息，光标将跳至不正确的行。如果编程错误不在所指示的行，你可以使用菜单命令"Edit > Go To > Line..."然后输入" 65536 + 显示错误数字"跳至有错误语句的行。

4.3.6. 问题："The FB is not available or the instance declaration is missing"

问题："The FB is not available or the instance declaration is missing"

解答：此消息出现在当一个 FC/FB/SFB/SFC 块在程序中被调用时，该块却不包含在块文件夹下。

解决方法 1：手动复制相关的程序块到 BLOCK 文件夹中然后重新编译 S7-SCL。

解决方法 2：调用 FC/FB/SFB/SFC 时，使用 S7-SCL 菜单中的 Insert>Block Call 向导，S7-SCL 将会自动将此 FC/FB/SFB/SFC 复制到当前的 BLOCK 目录中，从而避免以上情况出现。

4.3.7. 问题："Character strings have different lengths"

问题："Character strings have different lengths"

解答：示例中包含 FC1 和 FC2 带有字符串变量"name_1" 与"name_2"，当 FC1 被编译时，得到警告"Character strings have different lengths"。在运行期间，在赋值的右侧("name_1")可能是一个比左侧所允许的 STRING 变量("name_2")具有更长长度的 STRING 字符串。

解决方法：我们建议予左侧的变量"name_2"声明一个长度为 254 字符的 STRING 字符串。如果你编译具有这种声明的功能 FC2，上述警告将不会出现。

```
FUNCTION fc1: VOID
VAR_INPUT
    name_1: STRING[10];
END_VAR
VAR_TEMP
    name_2: STRING[10];
END_VAR
BEGIN
name_2:= name_1;
END_FUNCTION
```

```
FUNCTION fc2: VOID
VAR_INPUT
    name_1: STRING[10];
END_VAR
VAR_TEMP
    name_2: STRING[254];
END_VAR
BEGIN
name_2:= name_1;
END_FUNCTION
```

图 4-19：字符串定义

注意：如果你声明长达 254 字符的 STRING 字符串，将会需要更大的本地装载存储空间。

4.3.8. 问题：CPU 消息"STOP due to unknown OP code"

问题：CPU 消息"STOP due to unknown OP code"

解答：当你在 S7-SCL 中使用转换功能"WORD_TO_BLOCK_DB(...)"，并且编辑以下语句时该消息会被发出：

```
Display: = WORD_TO_INT (BYTE_TO_WORD (WORD_TO_BLOCK_DB (DBNo).DB [DBIndex]));
```

解决方法：我们建议以以下给出的两行语句来替代上面的语句：

```
tmp := WORD_TO_BLOCK_DB (DBNo).DB[DBIndex];
Display := WORD_TO_INT (BYTE_TO_WORD (tmp));
```

4.3.9. 问题：在编译 UDT 时出现"Syntax error with UNLINKED"

问题：在编译 UDT 时出现"Syntax error with UNLINKED"

解答：如果一个 UDT 在用"UNLINKED"声明之前已标识，该 UDT 不会被编译通过。"UNLINKED"属性不可用于 UDT。编译将会以错误消息"Syntax error with UNLINKED"结束。该规则也应用于 STL 源程序声明。UDT 通常不装载到 S7 CPU 中，而总是存储于"离线"的 S7 用户程序。

相关知识：对于"UNLINKED"数据块属性，意味着 DB 块有以下特性：

- 仅仅存储于装载存储器中
- 不占据任何 RAM 空间。
- "UNLINKED"属性声明在数据块的头部
- 与程序没有关联关系

```
DATA_BLOCK DB21
version: 1.0

UNLINKED  //only for data blocks

//Data blocks with the UNLINKED property are only
//stored in the load memory. They take up no space
//in the working memory AND are NOT linked TO the PROGRAM.

STRUCT
   Test: INT;
END_STRUCT

BEGIN

END_DATA_BLOCK
```

图 4-20: 数据块的 UNLINKIF 属性

4.3.10. 问题：在编译 DB 时出现"Syntax error with 2#1100_1100"

问题：在编译 DB 时出现"Syntax error with 2#1100_1100"

解答：如果你赋一个二进制数字作为初始值给一个数据块中 BYTE 类型的变量，编译 S7-SCL 源程序时将出现错误消息"Syntax error with 2#..."

例如： Var1: BYTE:= b#2#1100_1100;

在 S7-SCL, 数据块由 AWL 编译器生成。如果你在数据块准备部分和赋值部分同时初始化变量，你必须使用 AWL 专用符号。因此在示例中的 BYTE 类型的变量，仅允许指定十六进制数字作为初始值。S7-SCL 符号（例如 B#2#1100_1100）只可用在代码区，因为 S7-SCL 编译器只在代码区被使用。

解决方法：将二进制数字改为十六进制数字作为初始值，

例如： Var1: BYTE: = b#16#cc;

4.4. 与监控调试相关问题

4.4.1. 问题：为什么我的程序编译通过，但无法运行？

问题：为什么我的程序编译通过，但无法运行？

解答：当使用 S7-SCL 语言时，建议使用的 CPU 类型 最低为 CPU314。即便如此，用户也应当注意系统资源的限制。例如：当定义临时变量的时候，不要超过所用 CPU 技术参数中的本地数据大小的限制。否则可能导致程序无法运行。

4.4.2. 问题：为什么我的程序无法被监控？

问题：为什么我的程序无法被监控？

解答：为了程序可以被监控，在编译 S7-SCL 程序时，在菜单 **Options>Customize>Compiler** 中，应当选择 **Create debug info** 项。注意：**Create object code** 项也应当被选择，否则程序编译时仅仅作语法检查，而不生成可执行程序。**Optimize object code** 项也应当被选择，此选项将优化程序代码。

4.4.3. 问题：在 S7-SCL 哪些变量在监控时无法被显示？

问题：在 S7-SCL 哪些变量在监控时无法被显示？

解答：在监控时，如下变量无法被显示：

- 复杂数据类型不被显示，但复杂数据类型中的基本数据类型可以被显示（例如字节数组中的某个字节）
- DATE_AND_TIME、STRING、BLOCK_FB、BLOCK_FC、BLOCK_DB、BLOCK_SDB、TIMER、COUNTER 不被显示
- 以符号格式访问的数据块不被显示，例如：″myDB″.DW0, ″myDB″ 为某个数据块的符号名

注意：有时候监控变量的数据超出了 CPU 能够提供的资源数量，也将出现部分无法显示的情况。

5. 附录一推荐网址

5.1. 西门子自动化与驱动产品的在线技术支持

西门子（中国）有限公司自动化与驱动集团 客户服务与支持中心

网站首页：http://www.ad.siemens.com.cn/Service/

专家推荐精品文档：http://www.ad.siemens.com.cn/Service/recommend.asp

AS常问问题：http://support.automation.siemens.com/CN/view/zh/10805055/133000

AS更新信息：http://support.automation.siemens.com/CN/view/zh/10805055/133400

“找答案”AS版区：http://www.ad.siemens.com.cn/service/answer/category.asp?cid=1027