1.PCS7 功能库简介

SIMATIC PCS7 即 SIMATIC Process Control System，它是西门子公司为满足市场需求，基于全集成自动化 TIA 概念，为过程自动化应用于单一的平台而提供的统一 DCS 解决方案。

基于模块化、灵活性和开放性的设计理念，针对过程控制中一些标准的控制要求，PCS7 系统预制集成了 PCS7 Library 功能库。此功能库包括标准的模板驱动功能块集（模拟量、数字量输入/输出通道）、标准过程控制功能块集（PID 控制、马达控制、阀门控制、模拟量数字量监控等）、手操控制输出功能块集、报警功能块集等。使用系统提供的标准功能块，用户可以快速地、方便地、轻松地、从下到上一体化地组态项目中的具体控制流程。使用 CFC 编程工具并结合 PCS7 Library 功能库，工程师一次性的编程即可完成如下工作：

I. AS 控制器中运行的过程控制回路控制算法；

II. 针对控制回路在上位机中相关画面下对应的监视回路图标；

III. 控制回路在上位机中对应的操作及参数设置子画面；

IV. 控制回路对应的相关报警消息、趋势归档及用户操作记录等。

因此，使用 PCS7 提供的编程工具及功能库，用户可以非常方便地组态整个项目中对应的各种不同控制算法及画面。同时，基于开放性的开发理念，PCS7 系统为用户还提供了相应的功能库开发工具，用于用户自行开发适合用户或用户当前行业需求的功能库。此功能方便了用户在基于 PCS7 所提供的功能库的基础上对整个系统的功能进行延伸扩展。使用系统提供的功能库开发工具开发出来的功能库与 PCS7 自身所提供的功能库拥有相同的功能。

此外，为了满足某些特殊行业过程控制算法的需求，基于 PCS7 系统，西门子公司开发了相应的一些行业库可以供客户选订，在 PCS7 的 Add-on 订货选型手册中可以查阅到相关订货号，如下列举了一些相关行业库供参考。

- PCS7 HVAC —— 加热、通风及空调控制功能库（A&D AS）
- PTE 400 —— 技术功能块（I&S IS KHE）
- Standard Chemistry —— 技术功能块（A&D SC MHM）
- PCS 7 PS —— 发电行业功能块（PG L6）
- 其他各种不同的驱动功能块 ——（I&S ITPS KHE 等）

在本文档中，我们将主要讲解 AS 控制功能块的创建、修改及上位机监控图标和面板的制作等。

2.自定义功能库

当用户使用 PCS7 系统定义一个功能块，例如一个功能块类型 Block Type，首先，在 AS 控制器侧，它是一个功能块（Function Block）；然而在 OS 侧，它却是此功能块对应的功能块图标 Block Icon 和一套操作面板 Faceplate。

其他的功能块类型，比如功能（Function）在此将不予讨论，因为功能不具有背景数据块 Instance Data Block，而背景数据块又是 OS 和 AS 之间进行异步处理时，OCM（**O**perator **C**ontrol & **M**onitor）属性必要的前提条件。

- ✧ AS 侧

使用系统提供的相应工具，用户定义了一个功能块类型 Block Type，它必须包含如下部分：

- ✓ 输入/输出接口；
- ✓ 通过接口与外部系统建立的连接完成控制任务的程序或算法；
- ✓ 通用属性，例如块名等；
- ✓ 作为一个 AS 功能块 FB，还需包括：
  - ♦ 输入/输出接口和功能块的系统属性；
    - 类型相关属性，这些属性将应用于所有与此功能块相关的实例中，不可修改；
    - 实例相关属性，这些属性将作为新生成实例的默认值被应用，并可以根据用户需求针对单个实例进行定制；
  - ♦ 报警相关的文本模板
    - 类型相关消息，应用于所有实例，不可修改；
    - 实例相关消息，作为新生成的实例的模板被应用，可以根据用户需求针对单个实例进行定制。

如果用户在组态过程中，调用或嵌套使用了一个 Block Type，则对应此 Block Type，系统会自动创建一个功能块实例。此功能块实例包含了使用此功能块的所有数据，但不会包含代码 Code 部分。当前功能块实例被固定分配给 Block Type，所有与此 Block Type 相关的功能块实例拥有相同的接口。每次调用功能块实例时，系统会使用该背景数据块来调用对应的 Block Type。

在 AS 编程过程中，生成某 Block Type 对应的功能块实例的典型方法是：

- ✓ 在 CFC Chart 中，嵌套调用此 Block Type。

在 CFC 中，PCS7 用户应当使用预制功能块，只有功能块/功能库创建者才可以在一个 Block 中调用多个其他 Blocks（多实例方式）。

对应每个 AS 功能块实例，用户可以单独组态的内容有：

- ✓ 实例名；

- ✓ 实例相关的系统属性；
- ✓ 实例相关的消息文本等；

✧ OS 侧

在 PCS7 OS 侧图形组态中，用户生成面板实例 Faceplate Instance 的典型方法有：

- ✓ 在过程流程图形中嵌套面板类型 Faceplate Type；
- ✓ 在其他 Faceplate Type 中嵌套面板类型；

对每个面板实例，用户能可以单独组态的内容有：

- ✓ 与 AS 功能块的连接，也可以通过动态的方式建立；
- ✓ 组显示 Group Display 中包含的内容；

一个典型的 PCS7 功能块需要包含如下部分：

- ❑ 功能及操作模式：功能的基本描述、输入/输出的详细信息、操作模式及执行的时间顺序等；
- ❑ 调用 OB：嵌套 OB 块的申明。当在 CFC 中调用此功能块时，此功能块除了将会在循环 OB（OB30－38）中被调用外，在任务列表中定义的 OB 也会调用此功能块（例如，用于热启动初始化的 OB100 等）。CFC 编辑器将会在编译过程中，自动创建这些必须的 OB 块；
- ❑ 容错处理：功能块的布尔输出端口 ENO 将会指示该功能块的执行情况。在 FB 的情况下，背景数据块的输出端 QERR（ENO 取反）同样也会被存储。用户可以利用此端口来评估功能块的运行情况，例如，在错误情况下产生相应的报警消息或采用替换值等处理机制来加强程序的容错性能；
- ❑ 启动特性：分两中情况，如下

  - 初始化启动－Initial Startup

  功能块在相应嵌套的 OB 中第一次被调用，一般情况下这些 OB 是一些过程相关的 OB（例如，循环中断 OB）。此时，功能块采用系统默认的变量或预制的一些参数；

  - 启动－Startup

  功能块在 CPU 启动时，第一次被调用。在此情况下，需要确保功能块被启动 OB 调用（PCS7 中，默认为 OB100）。此时，需要在“启动特性”中对预制的处理进行描述；

- ❑ 时间响应：带有时间响应的功能块必须在循环 OB 中调用。这样，功能块可以通过它的采样时间（前后两次连续的执行之间的中断时间）来计算它的时间常量或参数；

当编译 CFC 并激活 Update Sampling Time 选项后，系统会自动采集当前循环 OB 的执行时间并置于 SAMPLE_T 端口。（一般情况下，在 CFC 中此端口被设为不可见 Invisible）。

- 信号传输特性：带有信号传输特性的功能块可以将过程值结合报警传输到 OS 进行显示；

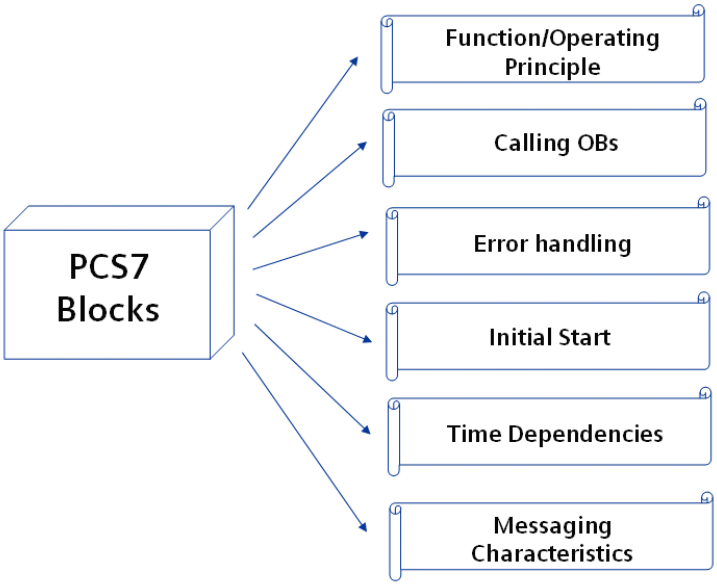


图 2.1：PCS7 功能块概念

在本文中，将以实现一个简单的两数相加求和的功能，来演示 PCS7 下如何自定义功能库的整个过程（AS 算法及上位显示画面等）。

2.1 帮助信息的获取

安装完成 PCS7 后，系统提供了相应的 PDF 文档。通过如下操作可以查看 PCS7 下的相关文档：

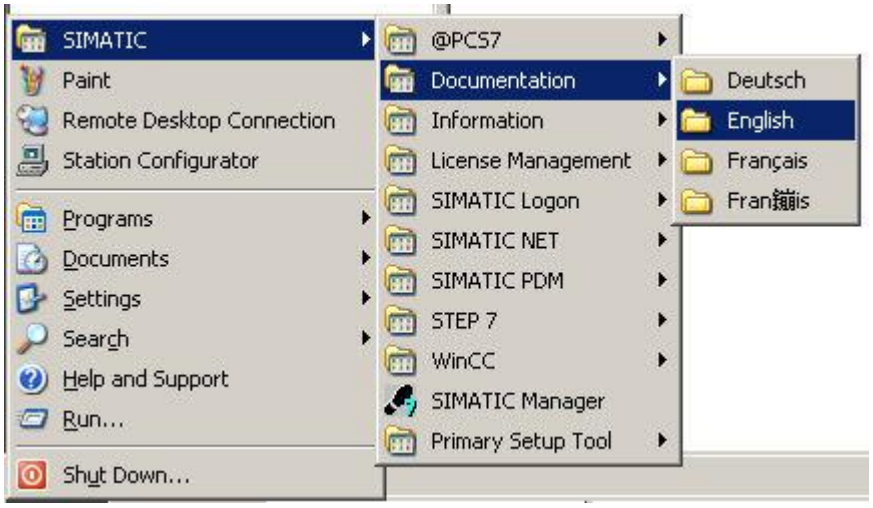


图 2.2 获取帮助文档

关于功能库的开发，请参考 PCS7－Programming Instructions for Blocks。

图 2.3 功能库参考文档

任何情况下，如果需要获得相关操作的帮助信息，请使用 F1 键或者访问西门子技术支持网站：http://www.ad.siemens.com.cn/Service/获得更多详细信息。

2.2 AS 功能块开发

PCS7 下通常使用 SCL 语言来进行功能块的开发，请参考如下文档：

图 2.4 SCL 参考文档

2.2.1 分析控制功能需求

在本例中，需要实现两个浮点数相加求和的功能，并设置一定的报警限位，当求和值大于某个设定值后，OS 系统触发相应的报警信息。

由功能分析来确定输入/输出管脚及相关属性：

- ✓ 输入管脚——IN1（加数 1），IN2（加数 2），HLIMIT（报警上限），HYS（报警阈值）；

- ✓ 输出管脚——OUT1（求和），QH_ALM（报警上限）；

2.2.2 创建测试项目

打开 PCS7，创建项目并加入 SIMATIC Station 和 PC Station，并进行相应的硬件组态，最终项目结构如下所示：

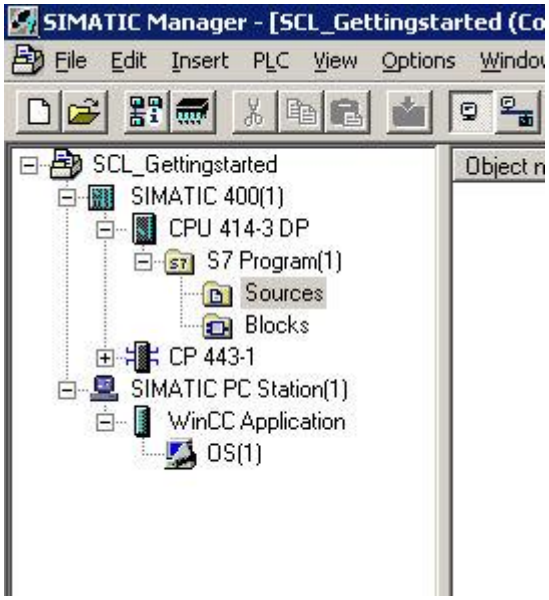


图 2.5 基本项目框架

2.2.3 编辑 SCL 源文件

2.2.3.1 插入 SCL 源文件

右键点击 Sources 文件加，插入 SCL Source 文件，或使用 External Source...导入磁盘上已有的源文件（可通过该操作将附录提供的 SCL Template 模板文件加入到项目中），同时还可以使用”Export Source…”菜单将源文件导出到磁盘上，具体操作如下：

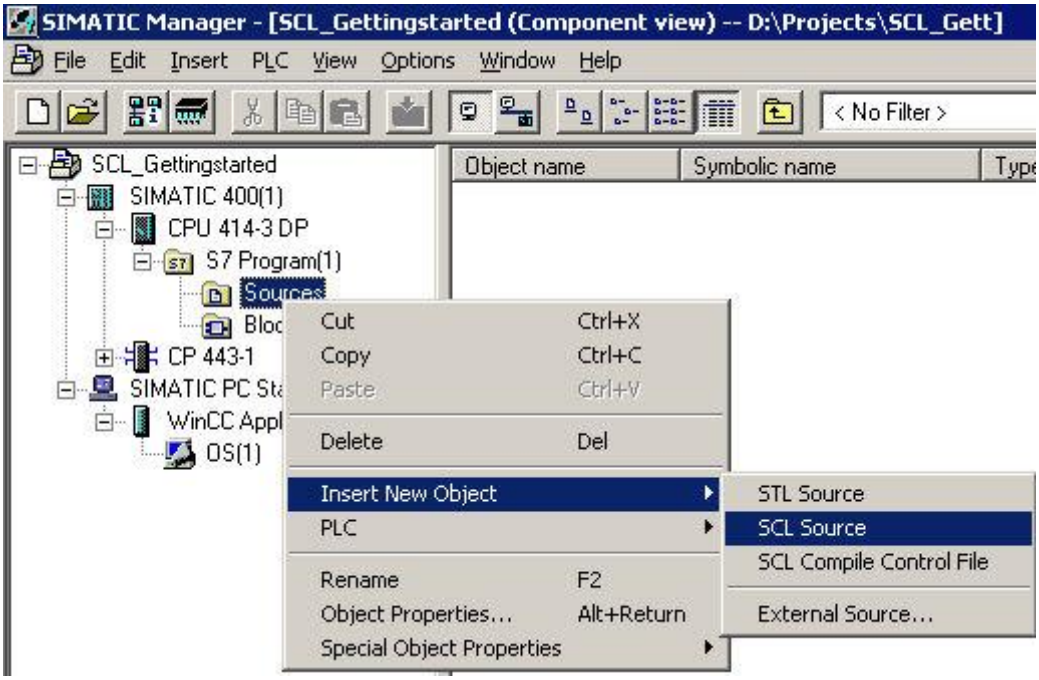


图 2.6 插入新的源文件

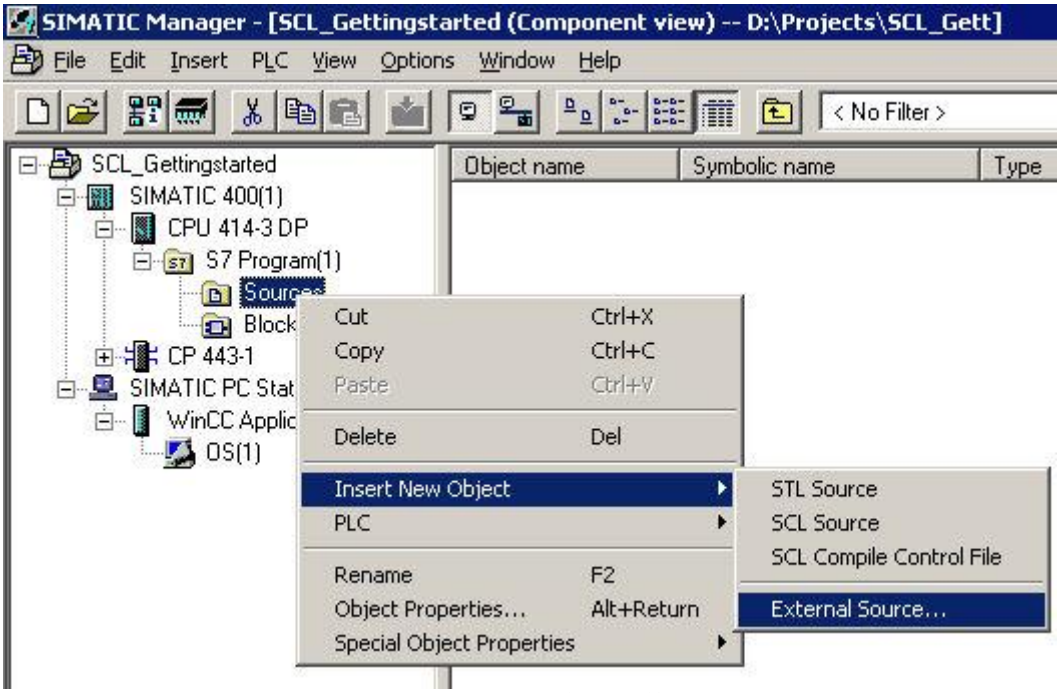


图 2.7 导入已有源文件

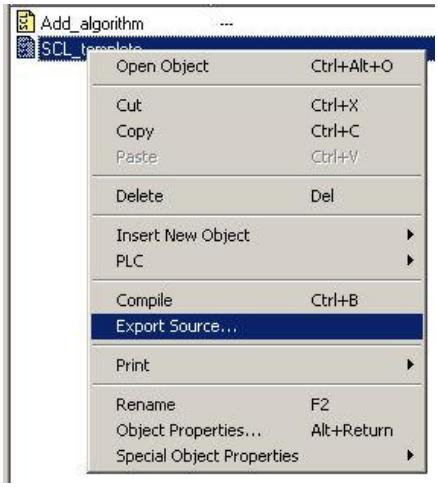


图 2.8 导出源文件

2.2.3.2 设置 SCL 编辑器并创建符号表

双击打开 SCL 源文件，点击 Options → Customize…进行 SCL 编辑器设置，激活 Create Debug Info 选项，具体操作如下：

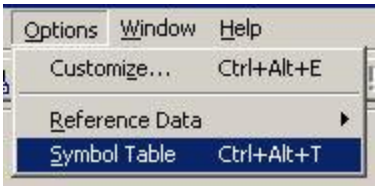


图 2.9 Options 菜单

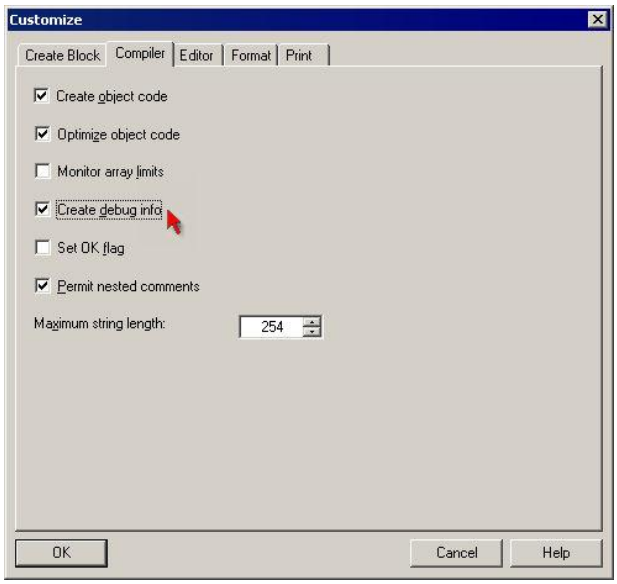


图 2.10 激活 Create Debug info 选项

注：激活该选项，主要用于 SCL 编辑器下程序的调试，会占用额外的存储空间；功能块开发完成通过测试后，最好将其关闭，并重新生成功能块。

通过菜单 Options→ Symbol table 为将要创建的功能块分配符号名，如下所示：

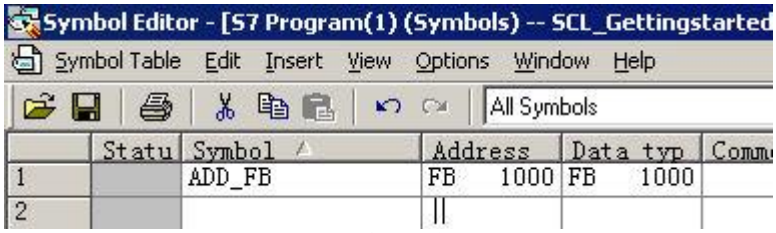


图 2.11 加入符号名

注：PCS7 下创建的功能块均为 FB 块，且为了避免和系统提供的 FB 块号冲突，通常定义 500 以后的块号，一般不定义 FC 块（无背景 DB）；

2.2.3.3 定义功能块头

定义功能块头如下所示，其中 FUNCTION_BLOCK、TITLE 和 NAME 需要和上述符号名相同。关于功能块头各项参数定义的具体意义，请参考上述图 2.3 所示文档中相关内容；

```
//*********************************************************************************************
FUNCTION_BLOCK "ADD_FB"
TITLE =   'ADD_FB'
{                        // List of system attributes
S7_tasklist:= 'OB100'; // Block is called if there is in a warm restart
S7_m_c:=      'true';       // Block can be controlled and monitored
S7_alarm_ui:= '1'           // Setting “1” for PCS 7 message dialog ('0' for standard message dialog)
}
AUTHOR:   ABC
NAME:       ADD_FB
VERSION:      '0.01'
FAMILY:   XYZ
KNOW_HOW_PROTECT
//*********************************************************************************************
```

2.2.3.4 定义输入/输出等管脚

根据上述功能分析中所需的管脚，定义输入输出等管脚，具体定义如下：

✓ 输入管脚

```
VAR_INPUT

 EV_ID   {
        S7_visible:='false';
        S7_link:='false';
        S7_param :='false';              // Parameter cannot be set in CFC
        S7_server:='alarm_archiv';        // Message no. assigned by server
        S7_a_type:='alarm_8p'             // Block signals with ALARM_8P
        }:DWORD := 0;          // Message ID

 RUNUPCYC{
        S7_visible:='false';
        S7_link:='false'
        } :INT := 3;          // Number of run up cycles

 MSG_LOCK {
```

```
        S7_visible:='false';
        S7_dynamic:='true';
        S7_m_c:='true'
        } : BOOL := 0; // Enable 1=Messages locked

  IN1 {  //加数 1
     S7_shortcut:='addend1';
     S7_unit:='';
     S7_m_c:='true';
     S7_dynamic:='true';
     S7_archive:='shortterm'
     } :REAL :=0.0; //input value 1

  IN2 {  //加数 2
     S7_shortcut:='addend2';
     S7_unit:='';
     S7_m_c:='true';
     S7_dynamic:='true';
     S7_archive:='shortterm'
     } :REAL :=0.0; //input value 2

  HLIMIT {  //限幅值
     S7_shortcut:='limit value';
     S7_unit:='';
     S7_m_c:='true';
     S7_dynamic:='true';
     S7_archive:='shortterm'
     } :REAL :=100.0; //limit value for output

  HYS {  //报警阈值
     S7_shortcut:='HYS';
     S7_unit:='';
     S7_m_c:='true'
     } :REAL :=1.0; //Hysteresis settings

END_VAR
```

注：

- EV_ID 主要用于报警触发，所有需要定义报警的功能块，必须定义该类型管脚，注意它的 S7_server 和 S7_a_type 属性；
- RUNUPCYC 管脚用于功能块启动时前几个周期内的报警抑制，防止 CPU 启动时通讯负荷过大；
- MSG_LOCK 用于报警锁定；
- 管脚相关属性在 SCL 的编程中非常重要，把握了这些属性并能灵活的引用是整个 PCS7 下自定义编程的关键，其具体意义请参考系统的在线帮助信息；

✓ 输出管脚

```
VAR_OUTPUT

  OUT1 {  //求和值
     S7_shortcut:='sum';
     S7_unit:='';
     S7_m_c:='true';
     S7_dynamic:='true';
```

```
    S7_archive:='shortterm'
    } :REAL :=0.0; //output value 1

  QH_ALM {  //报警输出
            S7_dynamic:='true'} : BOOL := 0; // 1=H-Alarm active

  QMSG_ERR  {
          S7_visible:='false';
          S7_dynamic:='true'} :  BOOL := 0;        // ALARM_8P: Error output

  QMSG_SUP {
         S7_visible:='false';
         S7_dynamic:='true';
         S7_m_c:='true'} : BOOL := 0; // 1=Message Suppression Active

  MSG_STAT  {
          S7_visible:='false';
          S7_dynamic:='true'} :  WORD := 0;        // Message: STATUS output

  MSG_ACK   {
          S7_visible:='false';
          S7_dynamic:='true'} :  WORD := 0;        // Message: ACK_STATE output

END_VAR
```

注：

- QMSG_ERR 用于输出报警错误；
- QMSG_SUP 用于**指示**报警抑制；
- MSG_STAT 用于指示报警状态；
- MSG_ACK 用于指示报警确认状态；

✓ 静态变量

```
VAR

  sbRESTART   :BOOL := TRUE;       // Initial start memory bit
  siRUNUPCNT  :INT := 0;           // Counter for RUNUPCYC execution
  sb_SIG_1: BOOL := FALSE; //Merker ALARM_8P Signal 1
  sb_SIG_2: BOOL := FALSE; //Merker ALARM_8P Signal 2
  sb_SIG_3: BOOL := FALSE; //Merker ALARM_8P Signal 3
  sb_SIG_4: BOOL := FALSE; //Merker ALARM_8P Signal 4
  sb_SIG_5: BOOL := FALSE; //Merker ALARM_8P Signal 5
  sb_SIG_6: BOOL := FALSE; //Merker ALARM_8P Signal 6
  sb_SIG_7: BOOL := FALSE; //Merker ALARM_8P Signal 7
  sb_SIG_8: BOOL := FALSE; //Merker ALARM_8P Signal 8

//*********************************************************************************************
// Declaration Section Multiple Instances(SFB/FB), remember to copy the FB/FC blocks into
your projects manually!
//*********************************************************************************************

ALARM_8P_1:  ALARM_8P;   // Generation of max. 8 messages with max. 10 auxiliary
values
```

```
END_VAR
```

注：

- 静态变量能保持上次运算的值，常用于保存数据；
- 如果需要在程序中调用其他的 FB 块，则需要为这些调用的 FB 声明静态变量；
- 实现报警功能，需要定义 ALARM_8p 的静态变量；

✓ 临时变量

```
VAR_TEMP

  pbALARM: BOOL; // Call up ALARM_8P
  pbM_SUP: BOOL; // Message suppression

pb_SIG_1,pb_SIG_2,pb_SIG_3,pb_SIG_4,pb_SIG_5,pb_SIG_6,pb_SIG_7,pb_SIG_8:BOO
L; // temp variable for alarm function
  // Start info: Structure with info for the OB that has just called the block
  TOP_SI:    STRUCT
   EV_CLASS :BYTE;
   EV_NUM   :BYTE;
   PRIORITY :BYTE;
   NUM      :BYTE;
   TYP2_3   :BYTE;
   TYP1     :BYTE;
   ZI1      :WORD;
   ZI2_3    :DWORD;
  END_STRUCT;

  // Start info: Structure with info for the last called startup OB
  START_UP_SI:  STRUCT
   EV_CLASS :BYTE;
   EV_NUM   :BYTE;
   PRIORITY :BYTE;
   NUM      :BYTE;
   TYP2_3   :BYTE;
   TYP1     :BYTE;
   ZI1      :WORD;
   ZI2_3    :DWORD;
  END_STRUCT;

  DUMMY  :INT;                  // Auxiliary variable

END_VAR
```

2.2.3.5 获取当前调用功能块的 OB 号

```
//*********************************************************************************
// Dependence on Calling OB
//*********************************************************************************

// Read out start info with SFC6 (RD_SINFO)

 DUMMY := RD_SINFO (TOP_SI := TOP_SI, START_UP_SI := START_UP_SI);
 pbM_SUP := MSG_LOCK;
```

```
IF sbRESTART THEN
 // Initial start
 TOP_SI.NUM := 100;        // Execute initial start as warm restart
 sbRESTART := FALSE;       // Reset initial start
END_IF;
```

使用 RD_SINFO 获取当前调用该功能块的 OB，通过评估 **TOP_SI.NUM** 即可判断该 OB 的 OB 号，并在下续程序中做相应处理，例如 OB100 中的初始化处理。注意需要将需要定义处理动作的 OB 加入功能块头的 **S7_tasklist** 属性中（见上述功能块头的定义）。

注：当 CPU 在运行状态下，下载该功能块，该功能块第一次在控制器中运行，则此时系统并不会运行 OB100，因此，初时化动作需要通过判断功能块中的静态变量 **sbRESTART** 为 true 来进行相应处理；

2.2.3.6 根据调用 OB 进行相应处理动作

```
// In which OB was the block called ?

 CASE WORD_TO_INT(BYTE_TO_WORD(TOP_SI.NUM)) OF

//*********************************************************************************
// Startup
//*********************************************************************************
// OB100: Warm restart
   100:
   QH_ALM := 0;
   OUT1:=0;

//default reset commands
   QMSG_ERR := 0;
   QMSG_SUP := 0;
   MSG_STAT := 0;
   MSG_ACK :=0;
   pb_SIG_1:= 0;
   pb_SIG_2:= 0;
   pb_SIG_3:= 0;
   pb_SIG_4:= 0;
   pb_SIG_5:= 0;
   pb_SIG_6:= 0;
   pb_SIG_7:= 0;
   pb_SIG_8:= 0;
   siRUNUPCNT := RUNUPCYC;  // Save RUNUPCYC value

 ELSE

//*********************************************************************************
// Technological Section
//*********************************************************************************
   OUT1:=IN1+IN2;
   IF OUT1>HLIMIT THEN
      QH_ALM:=true;
   ELSIF OUT1<HLIMIT-HYS THEN
      QH_ALM:=false;
   END_IF;
//*********************************************************************************
```

```
// Message suppression during the startup
//*********************************************************************************

  IF siRUNUPCNT = 0              // RUNUPCYC cycle already elapsed ?
  THEN
   IF MSG_LOCK THEN
      pb_SIG_1:= 0; // Report possible outgoing
   ELSE
      pb_SIG_1:= QH_ALM; // Alarm high
   END_IF;

   pbALARM:=sb_SIG_1<>pb_SIG_1;

  ELSE
   siRUNUPCNT := siRUNUPCNT - 1;
   pbALARM :=FALSE; // Initialization no ALARM
   pbM_SUP := TRUE;
  END_IF;

END_CASE;
```

使用 CASE 语句来评估 TOP_SI.NUM 号，并进行相应处理动作：

- ✓ 当 OB100 调用该功能块时，即进行相应的初始化动作；
- ✓ 当其他 OB 调用该功能块时，即进行相应的的控制任务；
- ✓ 判断 siRUNUPCNT 是否为 0，来进行启动前几个周期内的报警抑制功能；

2.2.3.7 报警处理

```
//*********************************************************************************************
// Messages with ALARM_8P
//*********************************************************************************************

// STRING variables must not be linked to ALARM8_P as auxiliary values
// so transfer in array of bytes
IF pbALARM THEN
 ALARM_8P_1
    (EN_R := TRUE,    // Update output ACK_STATE
     ID := 16#EEEE,        // Data channel for messages (always 16#EEEE)
     EV_ID:= EV_ID,        // Message number > 0
     SIG_1:= pb_SIG_1,                    // Signal 1 to be monitored
     SIG_2:= 0,//pb_SIG_2,                    // Signal 2 to be monitored
     SIG_3:= 0,//pb_SIG_3,                    // Signal 3 to be monitored
     SIG_4:= 0,//pb_SIG_4,                    // Signal 4 to be monitored
     SIG_5:= 0,//pb_SIG_5,                    // Signal 5 to be monitored
     SIG_6:= 0,//pb_SIG_6,                    // Signal 6 to be monitored
     SIG_7:= 0,//pb_SIG_7,                    // Signal 7 to be monitored
     SIG_8:= 0,//pb_SIG_8,                    // Signal 8 to be monitored
     SD_1 := AUX_PR01,      // Auxiliary value 1
     SD_2 := AUX_PR02,     // Auxiliary value 2
     SD_3 := AUX_PR03,       // Auxiliary value 3
     SD_4 := AUX_PR04,       // Auxiliary value 4
     SD_5 := AUX_PR05,      // Auxiliary value 5
     SD_6 := AUX_PR06,      // Auxiliary value 6
     SD_7 := AUX_PR07,      // Auxiliary value 7
     SD_8 := AUX_PR08,      // Auxiliary value 8
     SD_9 := AUX_PR09,      // Auxiliary value 9
     SD_10:= AUX_PR10);     // Auxiliary value 10
```

```
   QMSG_ERR := ALARM_8P_1.ERROR;      // ERROR status parameter
   MSG_STAT := ALARM_8P_1.STATUS;     // STATUS status parameter
   MSG_ACK := ALARM_8P_1.ACK_STATE;   // Current OS confirmation status
END_IF;

IF (NOT QMSG_ERR) THEN // Note historical signals.
   sb_SIG_1:= pb_SIG_1;
END_IF;

IF (MSG_STAT = 21) THEN // Block locked
   pbM_SUP := TRUE;
END_IF;
QMSG_SUP := pbM_SUP;
```

通过调用定义的 Alarm_8P 静态变量来触发报警功能。

- ✓ 首先判断触发报警的信号是否发生变化（pbALARM 是否为 true）来选择性的调用 Alarm_8p 功能块，优化 CPU 的运算和通讯负荷；
- ✓ 将定义的输入管脚 EV_ID 管脚付给 alarm_8p 的 EV_ID 管脚；
- ✓ 将触发报警的变量赋给 SIG_1、SIG_2、……；
- ✓ 将当前 pb_SIG_1 赋值给 sb_SIG_1 保存当前的报警状态；

注：PCS7 下通常使用 alarm_8p 功能块来触发报警，一个 Alarm_8p 功能块能同时实现最多 8 个变量的报警，如果功能块中需要触发报警的变量超过 8 个，则需要定义多个类似于 EV_ID 的输入管脚（注意 S7_server 和 S7_a_type 属性）及多个 Alarm_8p 的静态变量，并在程序中多次调用这些 alarm_8p 的静态变量，不同的 alarm_8p 使用不同的 EV_ID 输入变量；

2.2.4 编译 SCL 源文件

保存 SCL 源文件，并点击编译按钮编译 SCL 源文件，如果无错误信息，则 SCL 将在 Blocks 文件夹中生成该功能块；

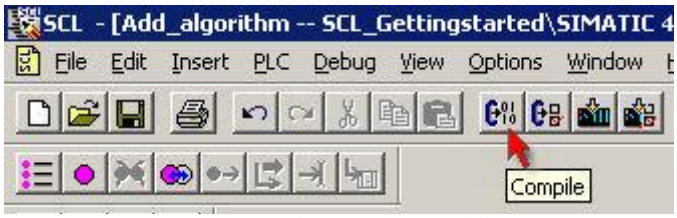


图 2.12 编译 SCL 源文件

2.2.5 定义功能块报警信息

Blocks 文件夹中找到生成的功能块，点击右键，Special object Properties → Message…为该功能块定义报警信息，具体操作如下所示。

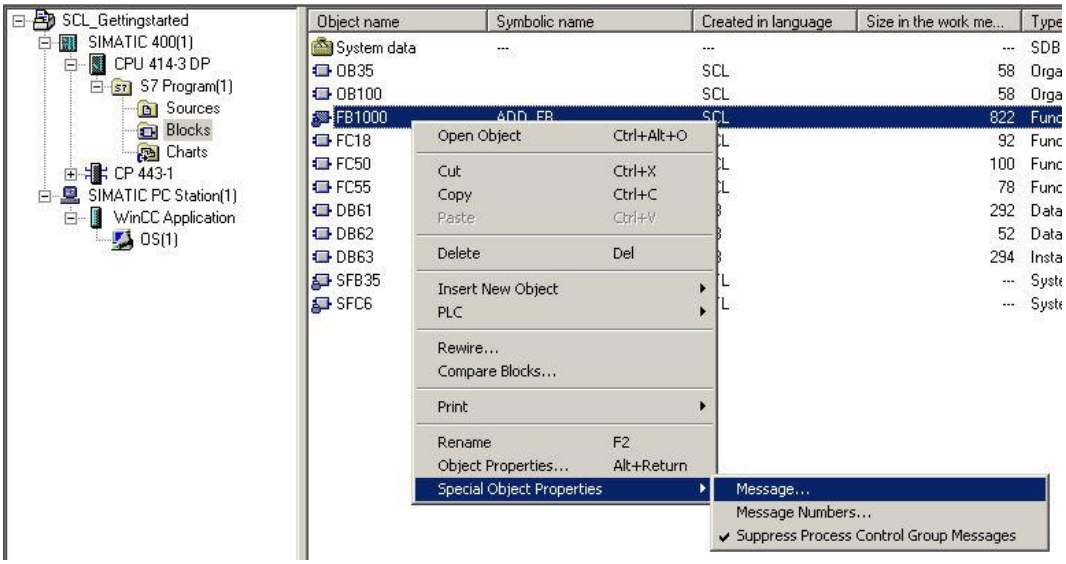


图 2.13 定义报警信息

按照下图定义功能块的报警信息：

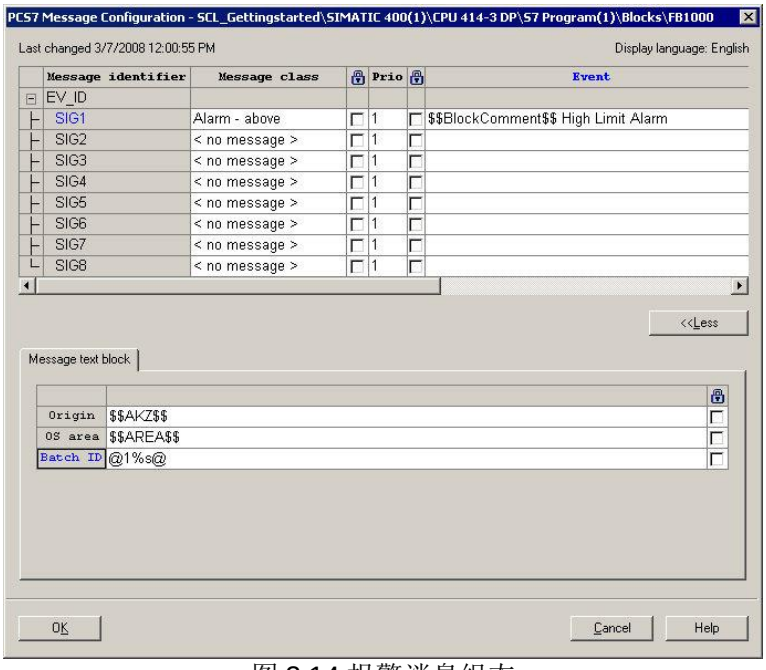


图 2.14 报警消息组态

注：定义报警消息时可以使用通配符，$$BlockComment$$代表功能块的注释信息，$$AKZ$$代表功能块的源，$$AREA$$代表功能块所处的 Area 区域。关于报警消息的组态方式详情请参考”STEP7-Programming with Step7”手册中的第 16 章节 “Configuring Messages”的内容。

2.2.6 调用功能块并编译 CFC

Plant view 下在某个 Hierarchy 中插入 CFC，打开 CFC 并从左侧 Block 下拖拽生成的功能块到 CFC 下进行调用，如下图所示，并编译 CFC；

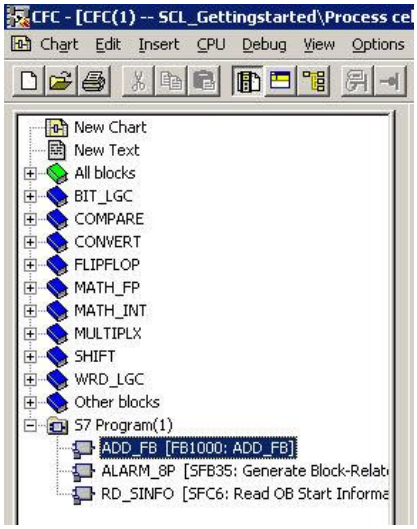


图 2.15 调用功能块

2.2.7 测试程序

将程序下载到 CPU 中，并测试其功能通过。到此为止功能块的算法部分即完成，可以进行 OS 部分的开发了。

2.3 Block Icon 开发

2.3.1 另存@@PCS7Typicals.pdl

打开 OS 项目，并使用图形编辑器打开@@PCS7Typicals.pdl 文件，将其另存为 @PCS7Typicals.pdl，如下图所示：

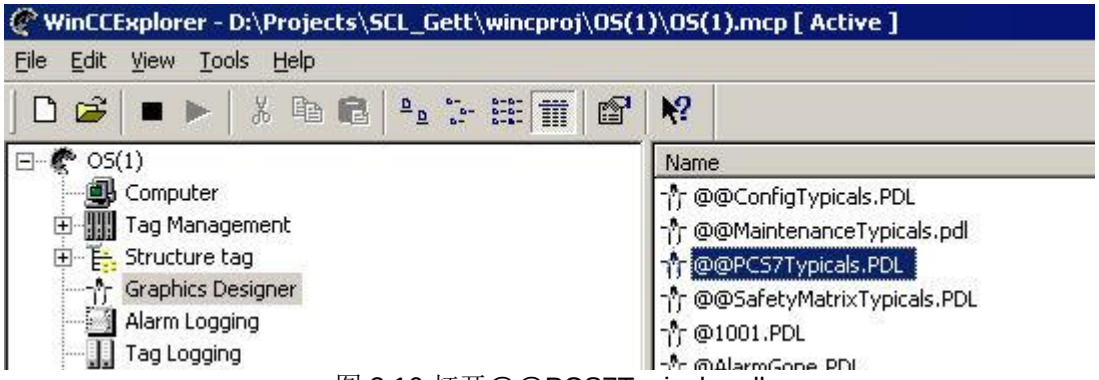


图 2.16 打开@@PCS7Typicals.pdl

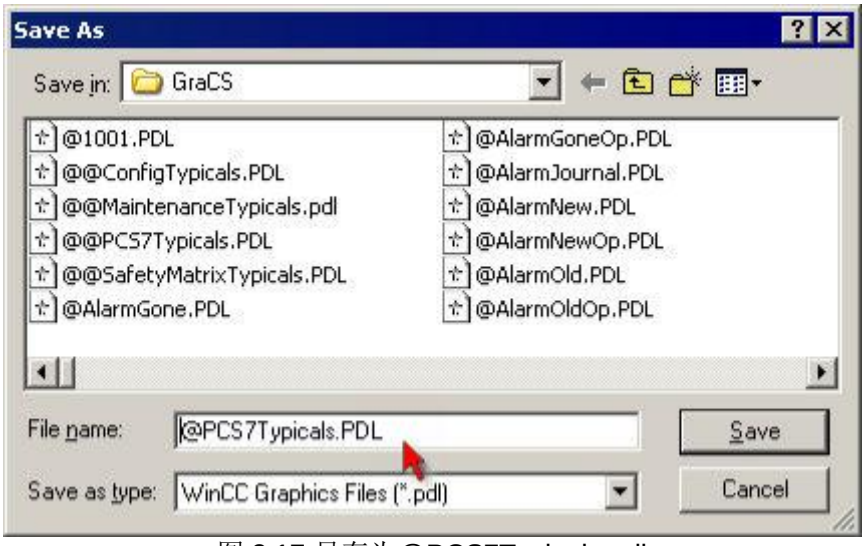


图 2.17 另存为@PCS7Typicals.pdl

注：
另存的文件名可以以@PCS7typicals 开始后面可以添加自定义的字符，例如，可以另存为@PCS7typicals _XYZ.pdl。

2.3.2 生成 Block Icon

2.3.2.1 拷贝生成 Block Icon

将图形下除 DIG_MON 以外其他对象删除，基于 DIG_MON 拷贝一个新的 Block Icon。

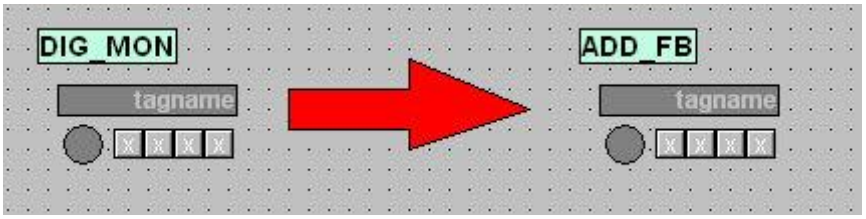


图 2.18 基于 DIG_MON 生成新的 Block Icon

2.3.2.2 编辑 Block Icon

点击拷贝的 Block Icon，右键 → Customized Object → Edit …，删除不必要的圆，加入 I/O field，并定义 I/O 域的相应属性，如下所示：

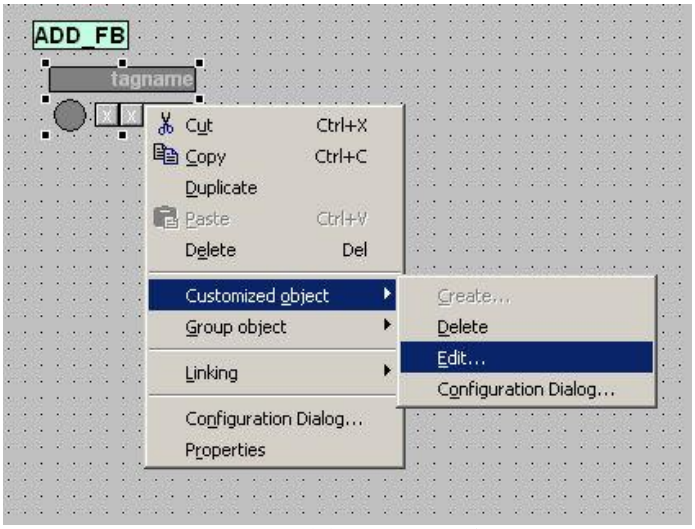


图 2.19 编辑自定义对象

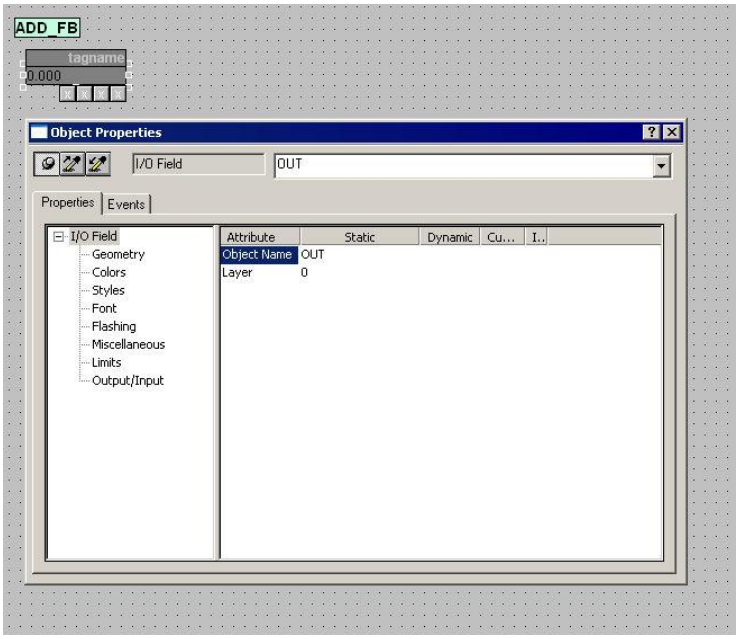


图 2.20 修改 I/O field 对象名为 OUT

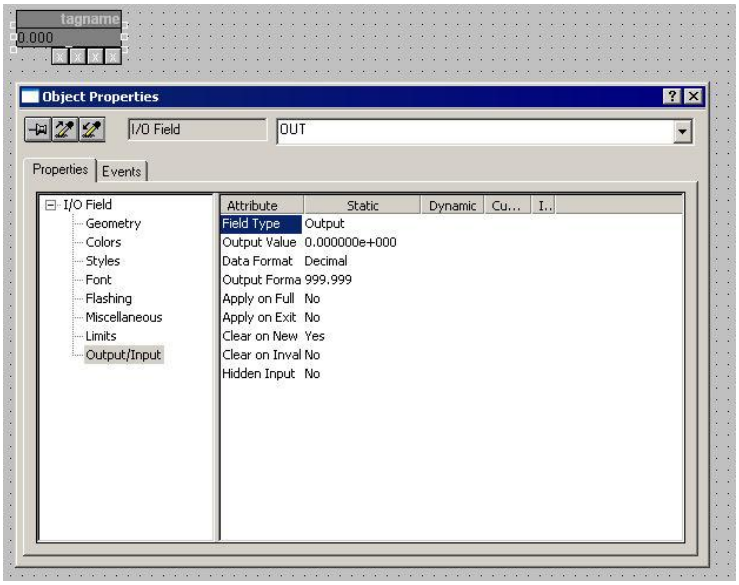


图 2.21 修改 I/O field 的类型为 Output

注：
Block Icon 中有很多属性为 PCS7 中必须的属性，因此删除不必要的对象时，一定需要注意，建议除圆型对象外，不要删除其他的对象，特别是那些隐藏在 Tagname 对象后面的对象。

全选 Block Icon 的所有对象，右键 → Customized Object → Finish Editing 完成 Block Icon 的编辑，如下图所示：

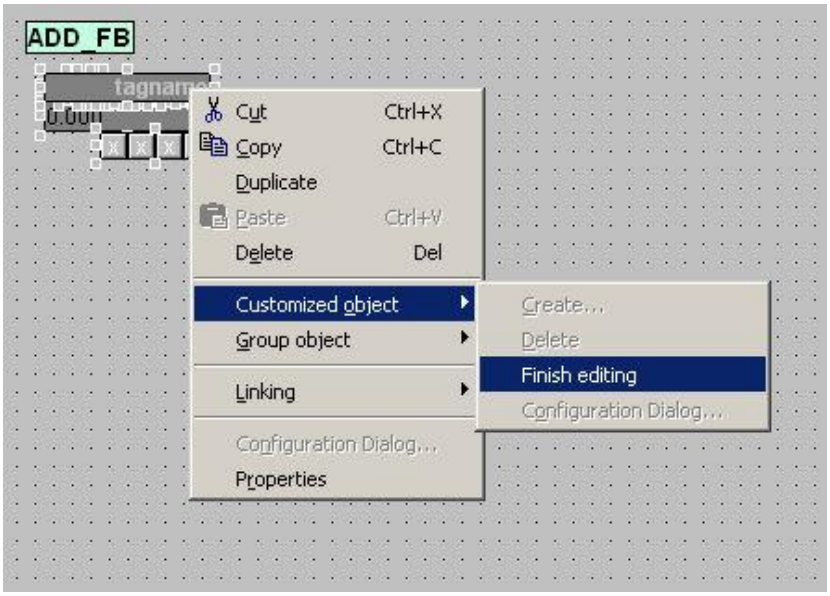


图 2.22 完成 Block Icon 的编辑

2.3.2.3 添加 Block Icon 属性

右键点击 Block Icon → Configuration Dialog…，打开属性添加对话框，将 I/O field 的 Outputvalue 属性加入到 Block Icon 中，并重新命名为 OUT1，详细组态如下所示：

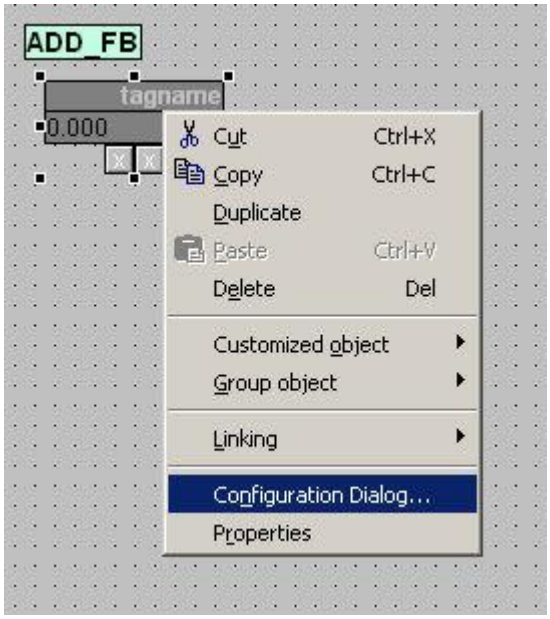


图 2.23 打开属性组态对话框

点击左侧的 Links 接点，双击右侧 Output value 属性，将其添加到左侧的 Links 下；

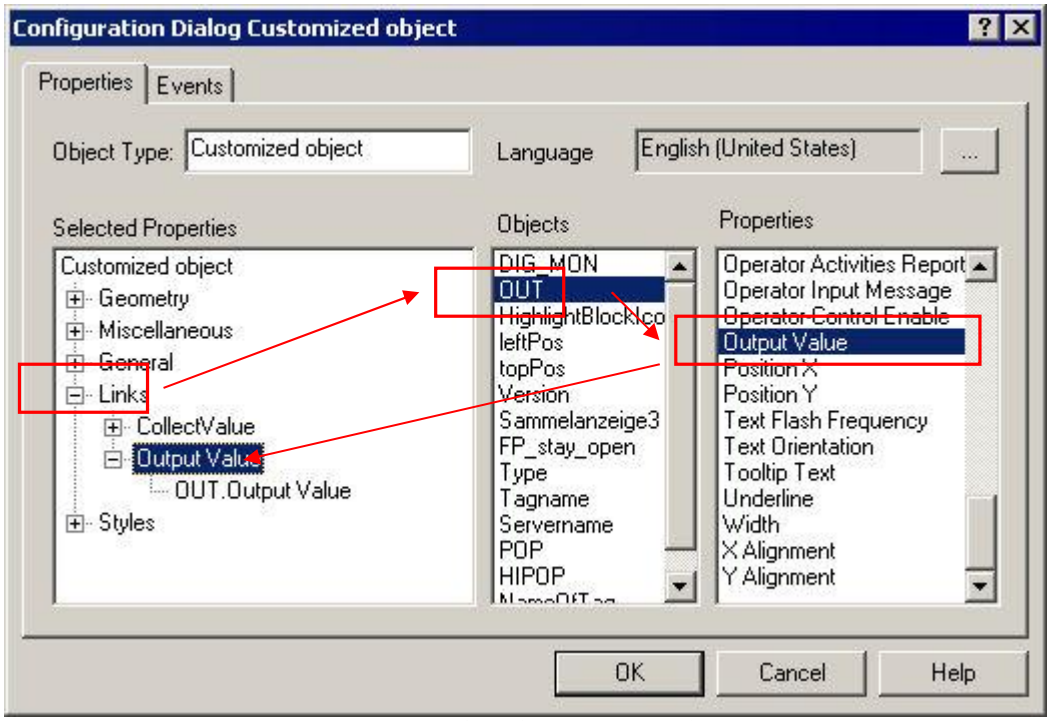


图 2.24 添加属性到左侧对话框

将属性重命名为 OUT1；

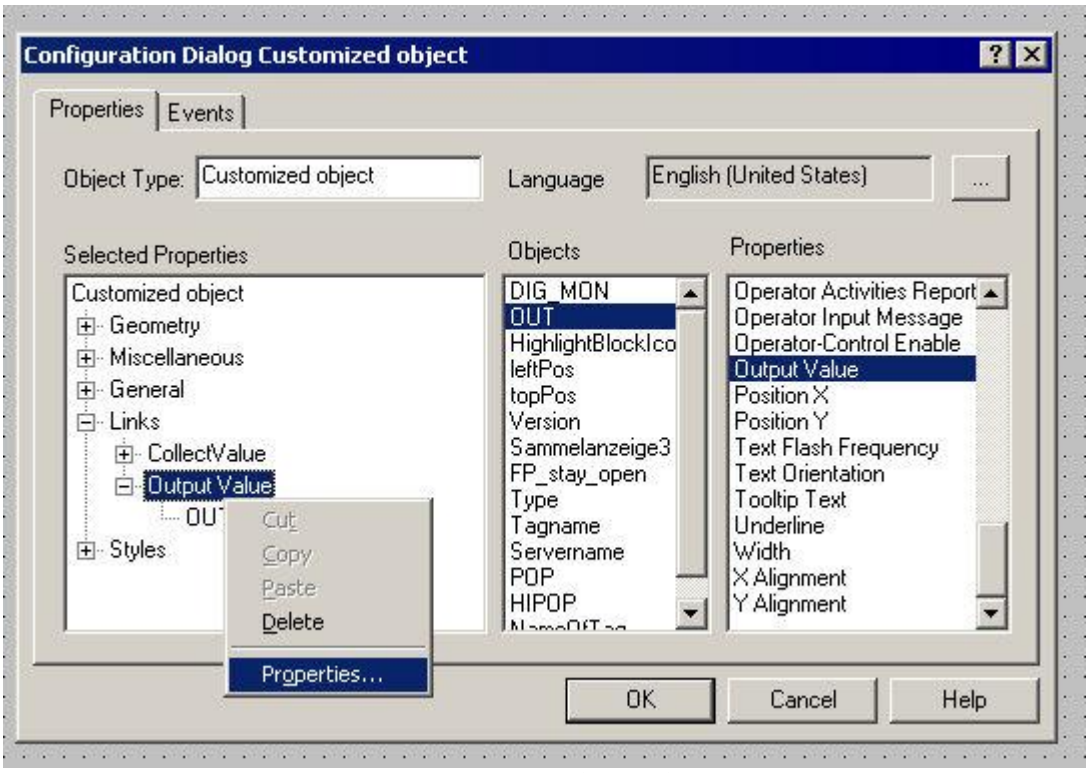


图 2.25 打开属性对话框

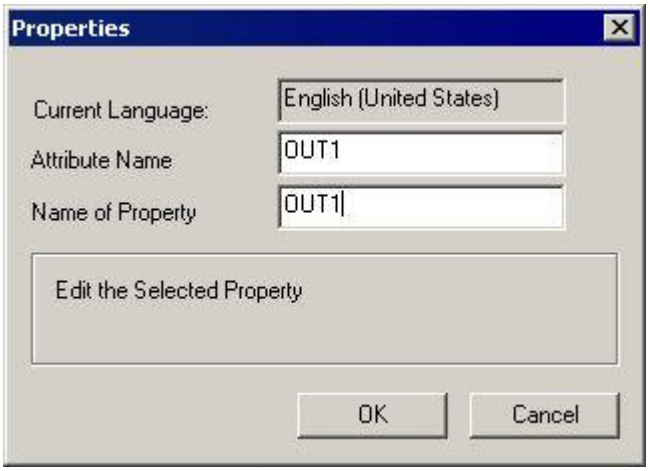


图 2.26 修改属性名为 OUT1

2.3.2.4 修改 Block Icon 属性

右键点击 Block Icon，打开属性对话框，修改 General 和 Links 下的相关属性，并保存该图形，具体操作如下图所示：

修改 General 下的 type 属性为**@ADD_FB**/1，Servername 属性为 PCS7 **ADD_FB** Control；

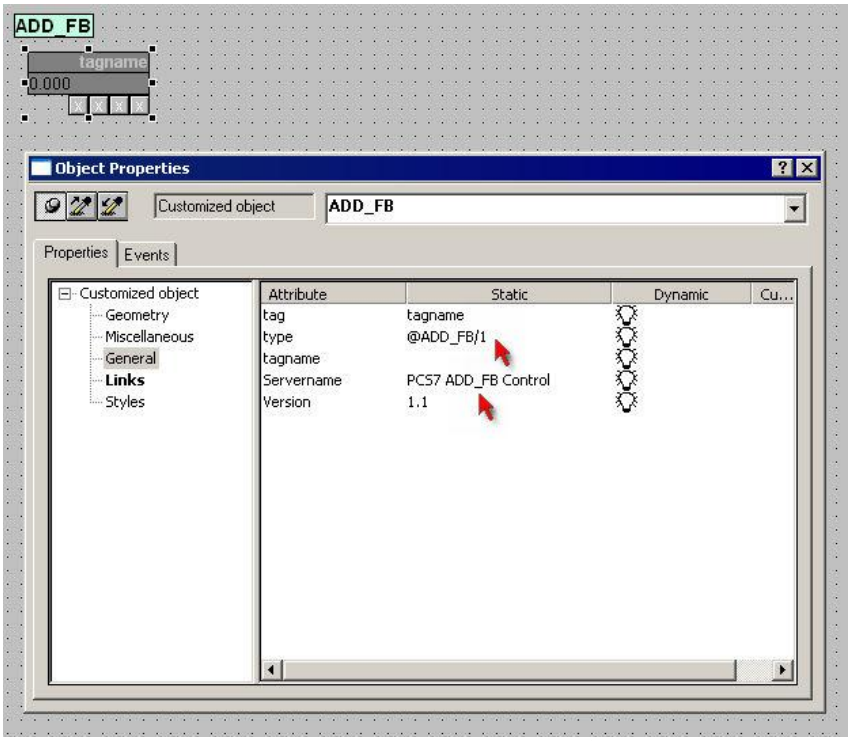


图 2.27 设置 Block Icon 的属性

注：type 和 Servername 属性非常重要，用于 Block Icon 的识别和标识打开的面板名；

修改 Links 下的 OUT1 属性 Dynamic 为**.OUT1**，更新周期为 2s；

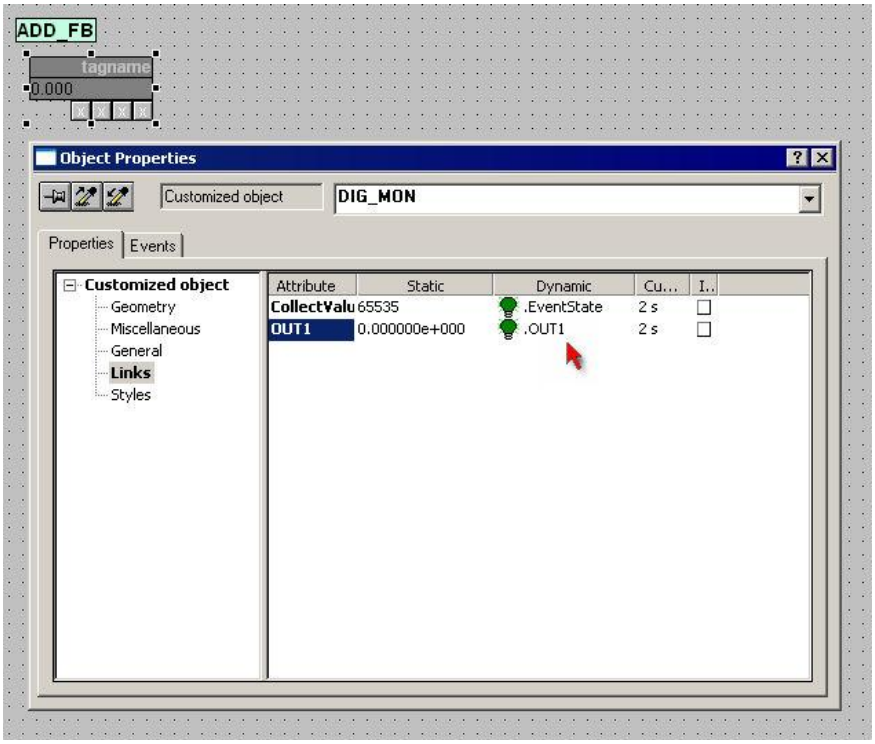


图 2.28 设置 Block Icon 的连接变量

2.4 Faceplate 开发

2.4.1 打开并创建 Faceplate 模板

双击 WinCC 下的 Faceplate Designer 打开面板设计器，如下图所示：

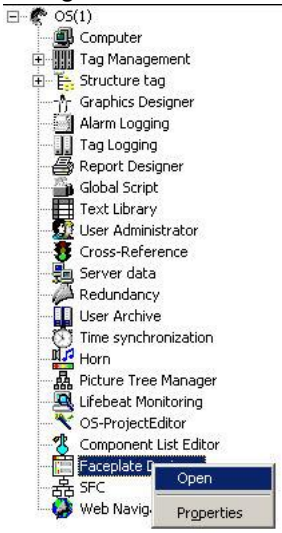


图 2.29 打开面板设计器

Type 中输入功能块名 ADD_FB，激活 **No Batch Variables** 选项，点击 **Generate** 按钮；

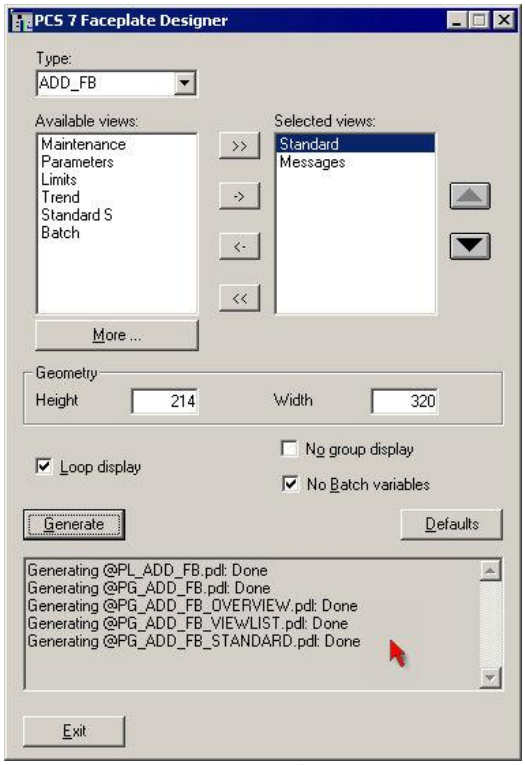


图 2.30 生成模板文件

系统将自动生成

- @PL_ADD_FB.pdl,
- @PG_ADD_FB_pdl,
- @PG_ADD_FB_OVERVIEW.pdl,
- @PG_ADD_FB_VIEWLIST.pdl,
- @PG_ADD_FB_STANDARD.pdl

图 2.31 生成的模板文件

这些文件为 ADD_FB 面板显示的模板文件，可拷贝到其他项目中使用；

2.4.2 修改 Faceplate 模板

打开上面生成的@PG_ADD_FB_STANDARD.pdl 文件，打开@PCS7elements.pdl 文件，将该文件下的用于过程值显示和输入的控件拷贝到@PG_ADD_FB_STANDARD.pdl 下。

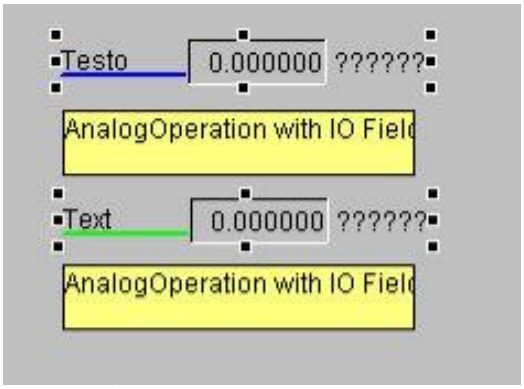


图 2.32 拷贝对象到@PG_ADD_FB_STANDARD.pdl 中

注：
@PCS7elements.pdl 文件为 PCS7 系统提供的面板开发时的对象库文件，所有用于操作的对象都必须拷贝自该文件，例如，按钮、输入框、选择框等；

加入其他对象，编辑该图形如下所示：

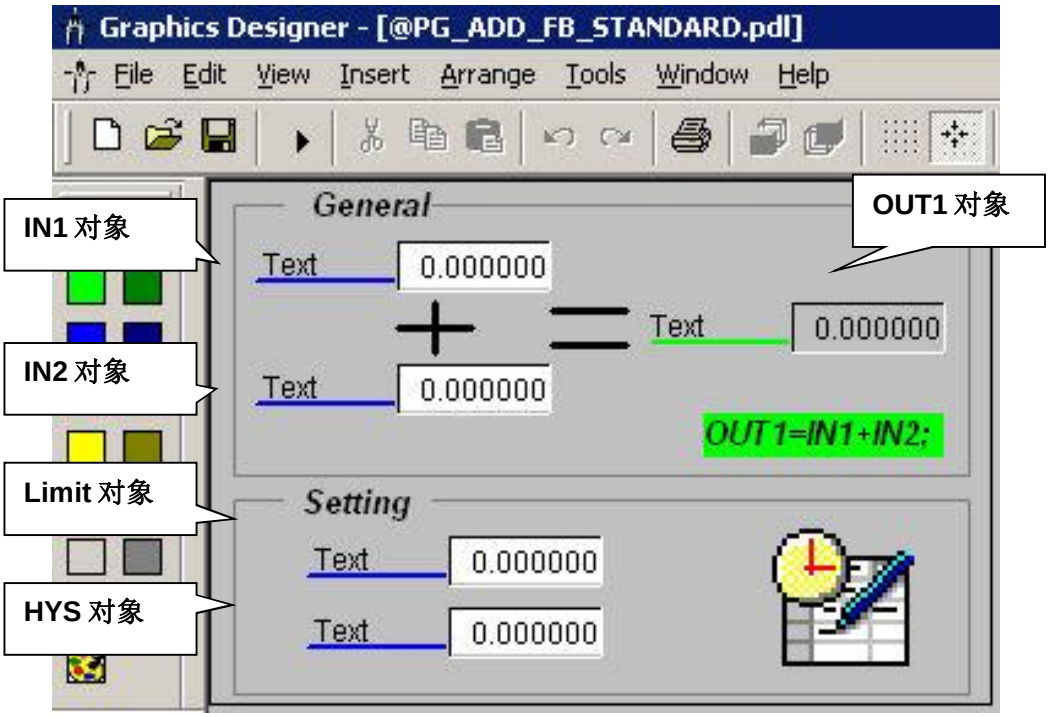


图 2.33 编辑图形对象

修改各对象的属性，如下所示，保存文档：

- 修改 IN1 的 Font 属性，Text 为**.IN1#shortcut**，Unit 为**.IN1#unit**，更新时间 1h；

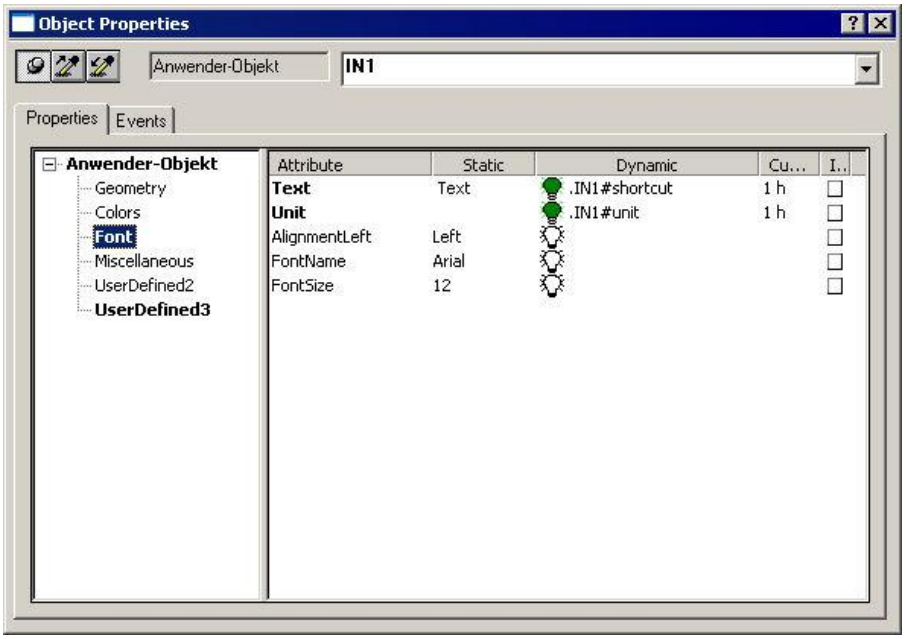


图 2.34 修改 IN1 对象的 Font 属性

- 删除 IN1 的 UserDefined2 下的属性，修改 IN1 的 UserDefined3 属性，OutputValue 为**.IN1**，VisibleValue 为**.IN1**，更新周期 2s；

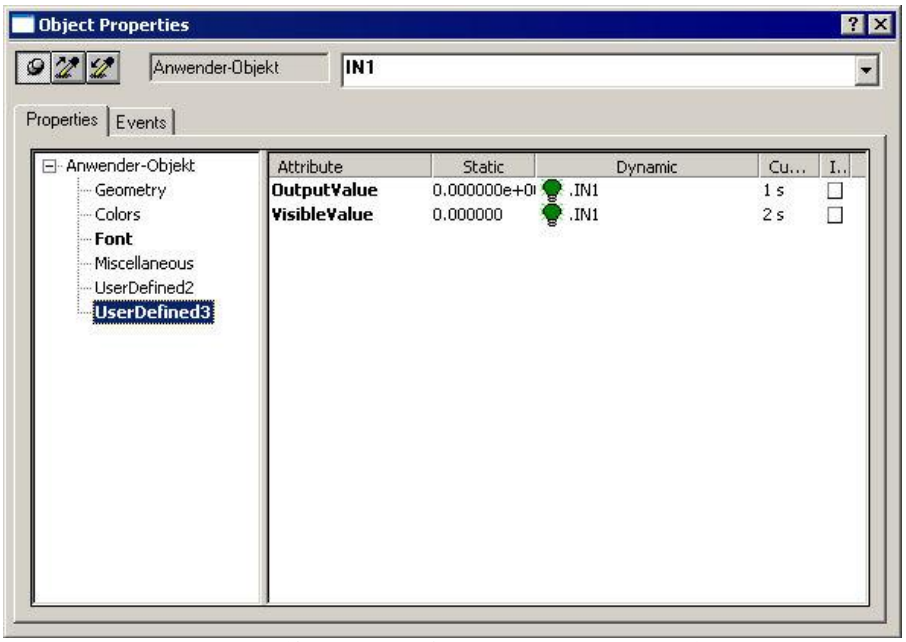


图 2.35 修改 IN1 对象的 UserDefined3 属性

- 修改 IN2 的 Font 属性，Text 为**.IN2#shortcut**，Unit 为**.IN2#unit**，更新时间 1h；

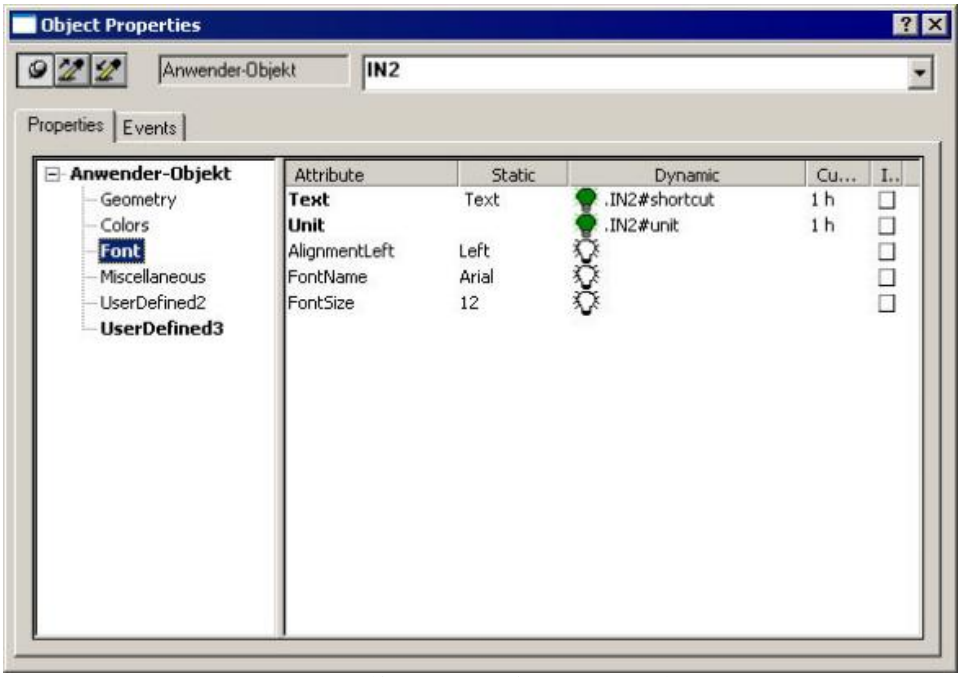


图 2.36 修改 IN2 对象的 Font 属性

- 删除 IN2 的 UserDefined2 下的属性，修改 IN2 的 UserDefined3 属性，OutputValue 为**.IN2**，VisibleValue 为**.IN2**，更新周期 2s；

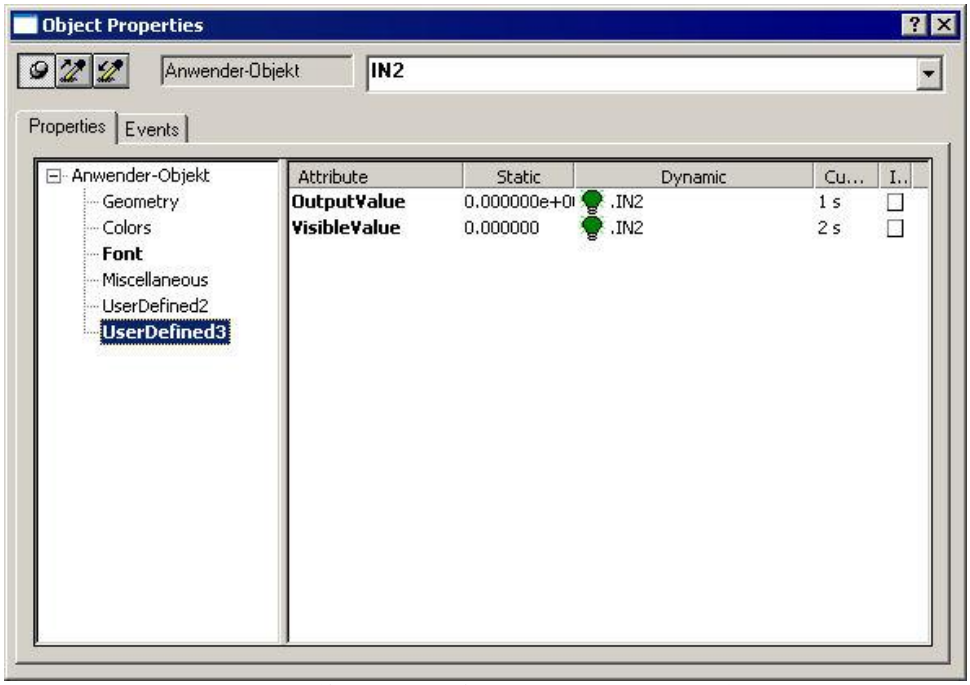


图 2.37 修改 IN2 对象的 UserDefined3 属性

- 修改 OUT1 的 Font 属性，Text 为**.OUT1#shortcut**，Unit 为**.OUT1#unit**，更新时间 1h；

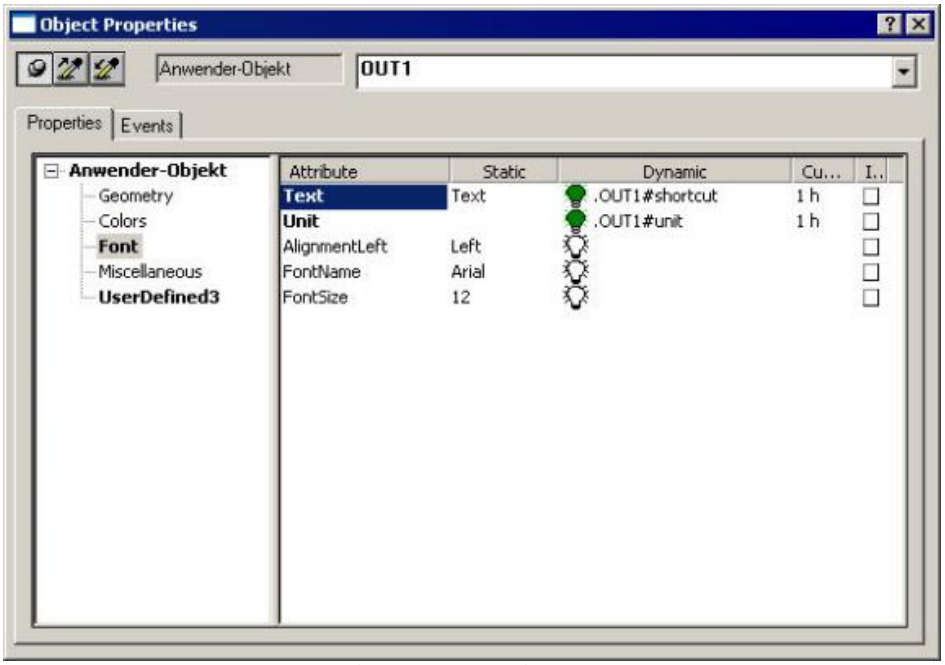


图 2.38 修改 OUT1 对象的 Font 属性

- 删除 OUT1 的 UserDefined2 下的属性，修改 OUT1 的 UserDefined3 属性，VisibleValue 为**.OUT1**，更新周期 2s；

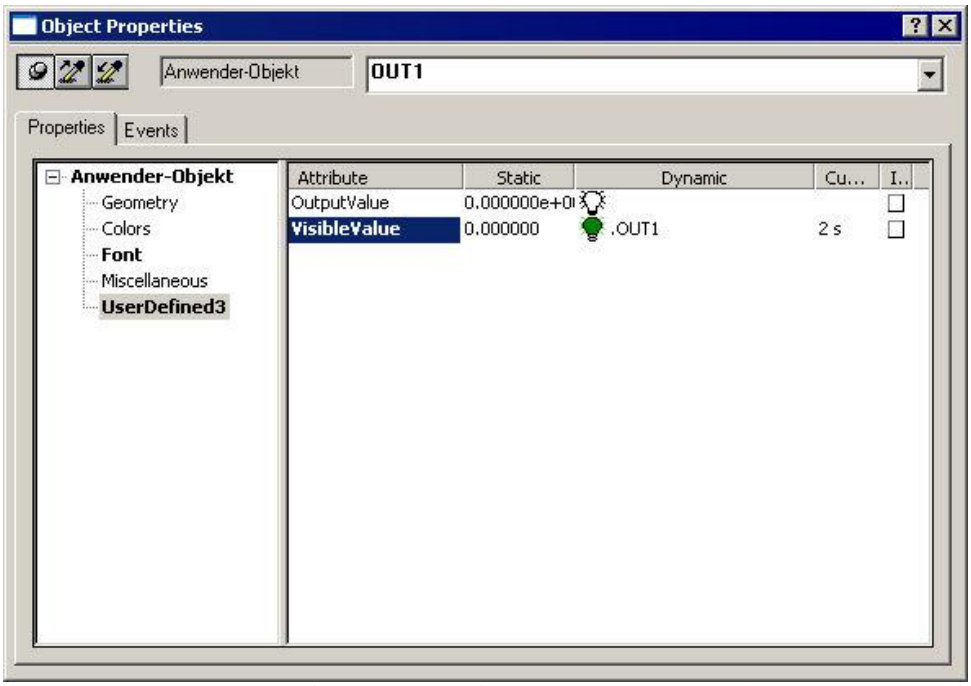


图 2.39 修改 OUT1 对象的 UserDefined3 属性

- 修改 limit 的 Font 属性，Text 为**.HLIMIT#shortcut**，Unit 为**.HLIMIT#unit**，更新时间 1h；

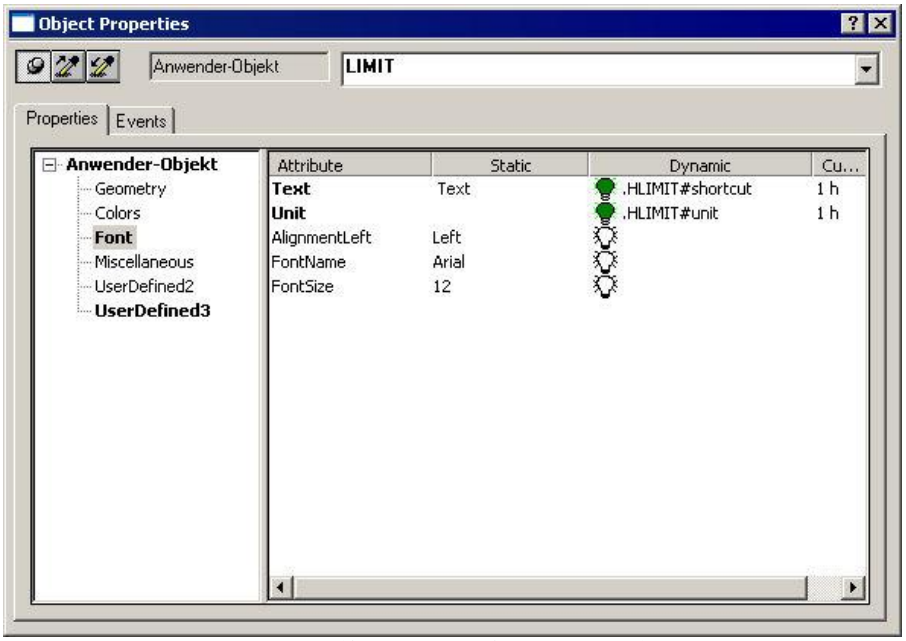


图 2.40 修改 limit 对象的 Font 属性

- 删除 limit 的 UserDefined2 下的属性，修改 limit 的 UserDefined3 属性，OutputValue 为**.HLIMIT**，VisibleValue 为**.HLIMIT**，更新周期 2s；

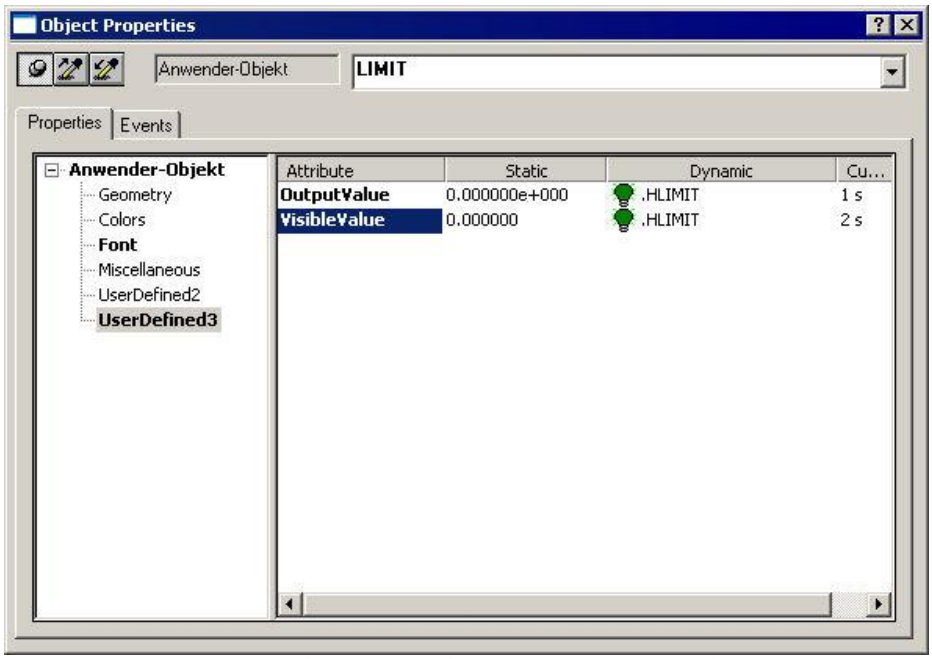


图 2.41 修改 limit 对象的 UserDefined3 属性

- 修改 HYS 的 Font 属性，Text 为**.HYS#shortcut**，Unit 为**.HYS#unit**，更新时间 1h；

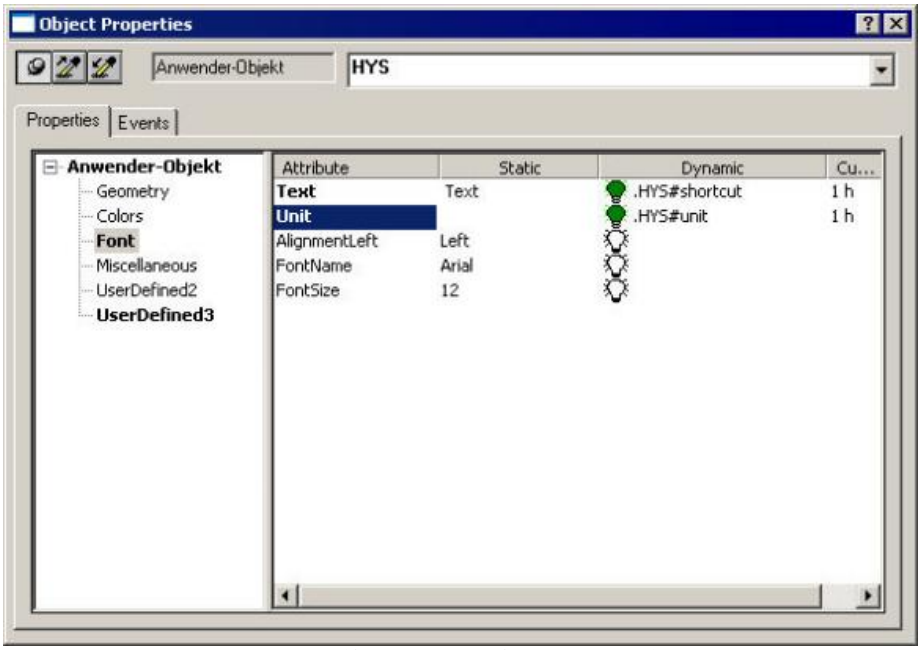


图 2.42 修改 HYS 对象的 Font 属性

- 删除 HYS 的 UserDefined2 下的属性，修改 HYS 的 UserDefined3 属性，OutputValue 为**.HYS**，VisibleValue 为**.HYS**，更新周期 2s；

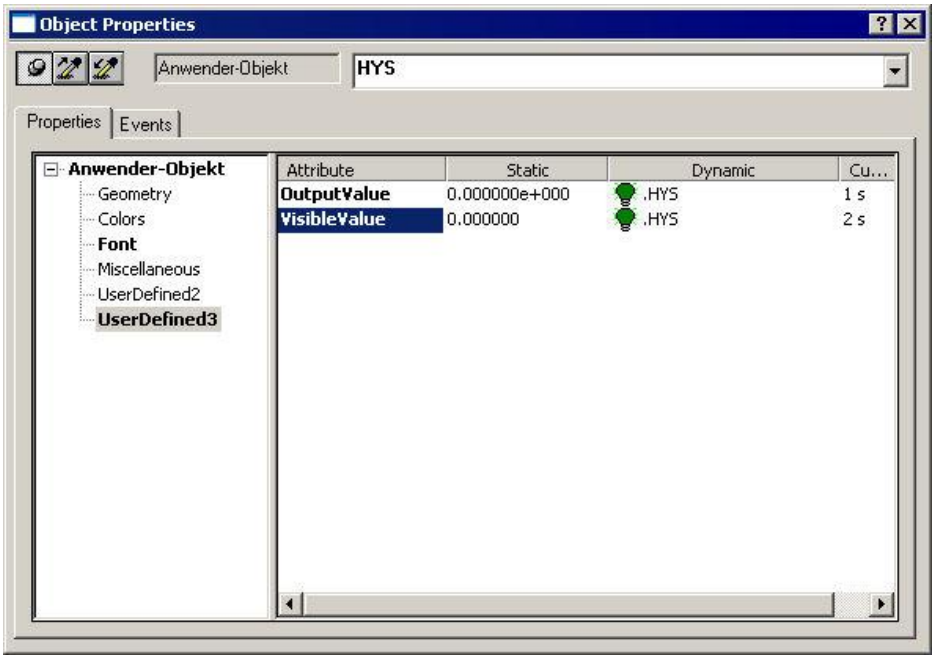


图 2.43 修改 HYS 对象的 UserDefined3 属性

2.5 编译 OS 项目

编译 OS 项目，选择 Create/Update Block Icon 选项，如下所示；

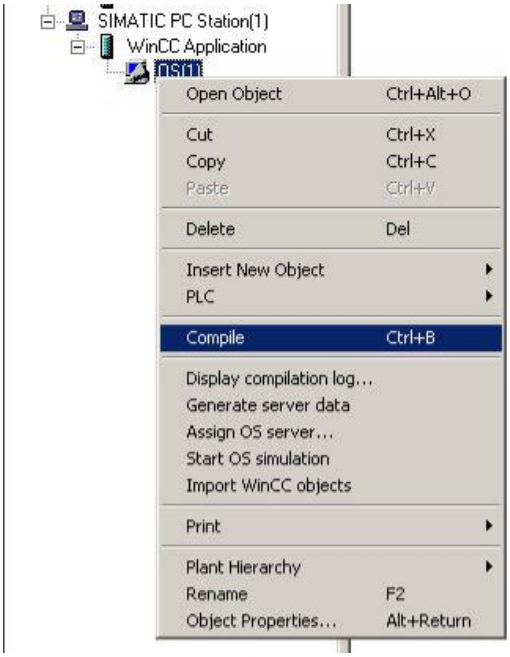
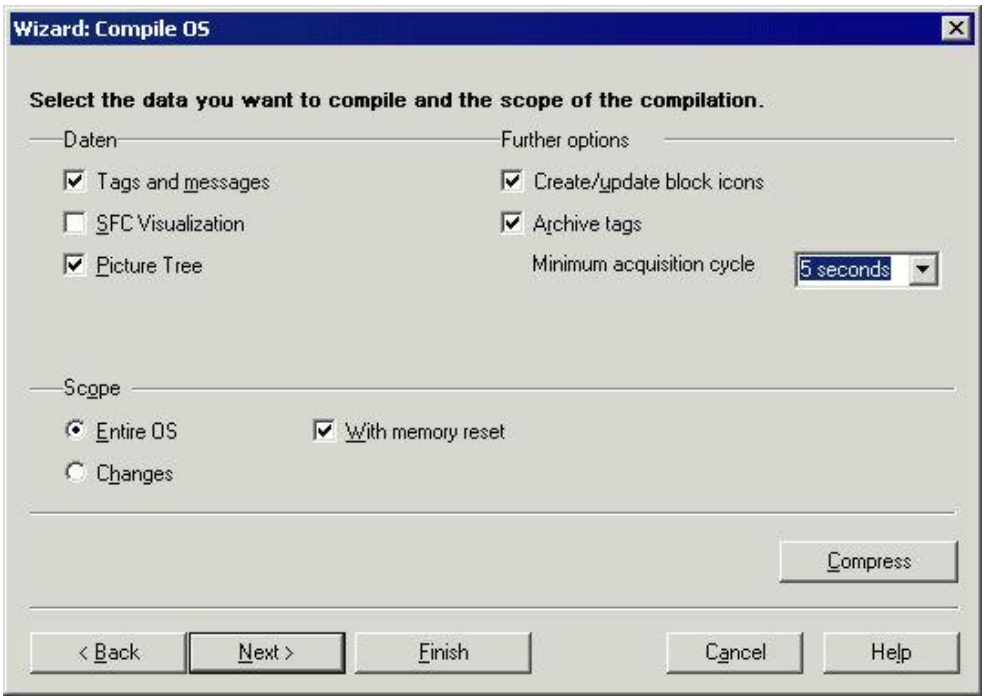


图 2.44 编译 OS 项目



2.45 Compile OS 选项

2.6 运行调试项目

下载 AS 程序到控制器中，激活 OS 项目运行调试面板，运行情况如下所示；

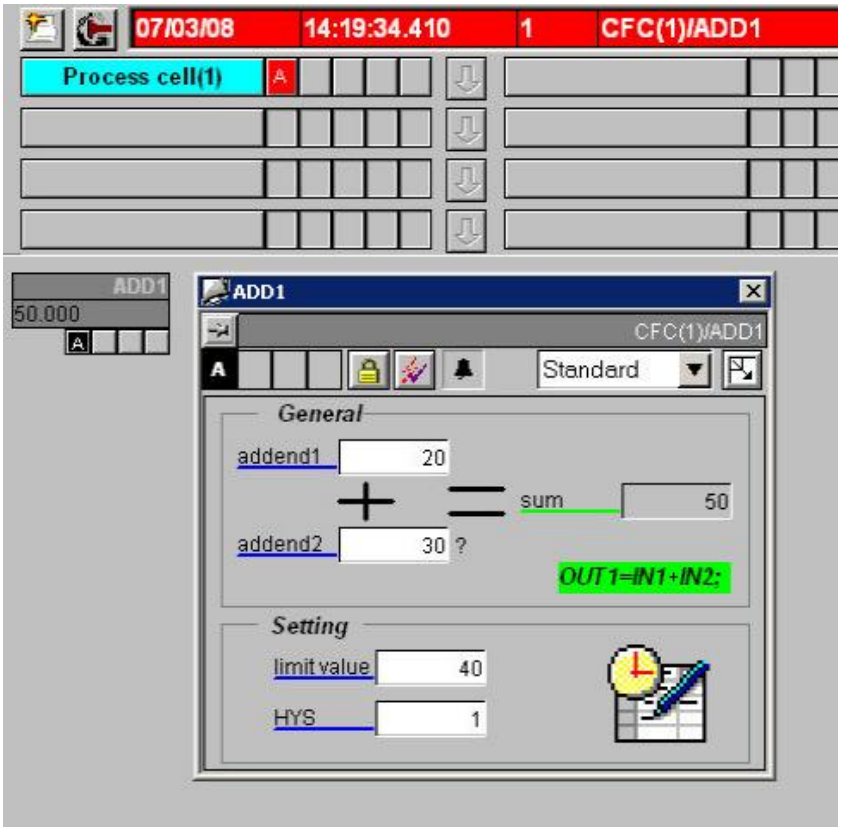


图 2.46 打开 Group Display 面板

点击右侧的 Loop Display 按钮，显示 Loop Display 面板

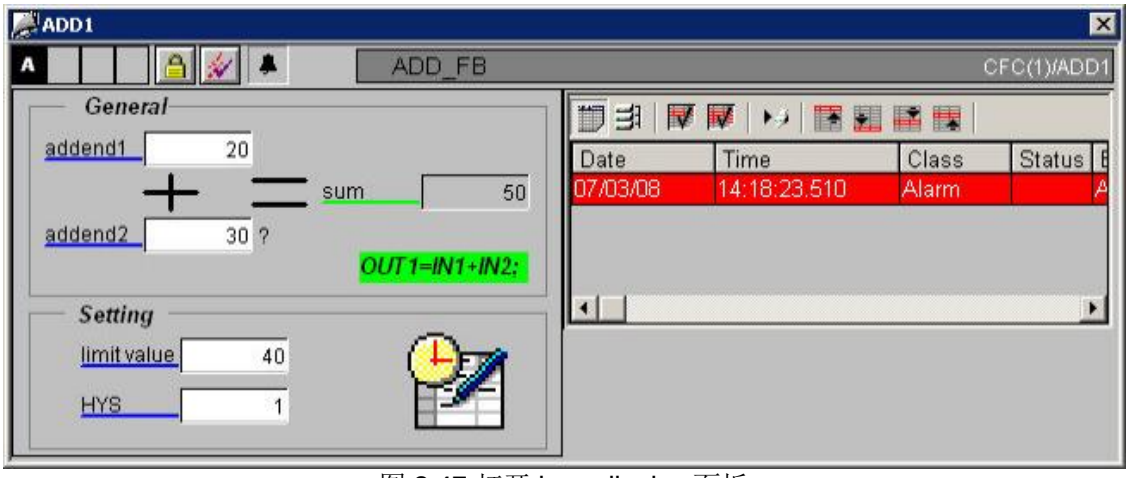


图 2.47 打开 loop display 面板

2.7 功能库的发布

功能块经过上述操作后测试通过，即可在实际项目中使用了。只需要将相应功能块拷贝到实际项目的 Blocks 文件夹下或拷贝到项目的库中，上述创建的面板图形文件，需要将其拷贝到项目下的 OS 项目下的 GraCS 目录下。更多关于项目发布的内容请参考图 2.3 中文档第四章中的内容。

3. 总结

PCS7 提供了一个开放的平台用于自定义功能库的开发，本文仅仅演示了一个简单功能库的开发过程，而且仅仅介绍了功能块的编程，block icon 和 faceplate 的简单制作过程。现总结一下其中的几个关键点：

- ✓ 功能块头的定义，注意 S7_tasklist 和 S7_alarm_ui 两个属性。前者设置该功能块在 CFC 调用中，将自动加入哪些 OB 块中，例如，本文例中的自动加入 OB100 用于初始化动作；后者用于设置组态报警的窗口类型，见图 2.14，尝试将其修改为 0 后，测试一下报警组态窗口有何不同；
- ✓ 管脚的属性定义，例如，S7_m_c 表述自动上传变量到 OS，S7_shortcut 和 S7_unit 专门为模拟量设置，用于上位显示该变量的名称和单位，S7_string_0/1 用于显示该变量为 0 或 1 时的字符串。鉴于篇幅的原因该处不一一列举，希望用户能详细参考系统的帮助信息，把握好属性是 PCS7 下自定义功能块的关键；
- ✓ 报警功能的实现，需要定义一个端口（名字随意），其属性必须包含 S7_server 和 S7_a_type，及定义一个 alarm_8p 的静态变量。程序中调用该静态变量时，将上述声明的端口分配给 EV_ID 端口；
- ✓ 此外，顺便提醒一下用户，PCS7 中需要做定时功能时，一般不使用系统提供的定时器 Tn，而是通过自己编写计数器的方式来实现定时功能，则此时需要借助名为 S7_sampletime 的属性，有兴趣的用户可以自己实验一下；

文中没有涉及太多 Block Icon 细节的东西，Faceplate 中权限传递、显示格式、归档显示的内容，也没有介绍制作功能库帮助文件及发布程序的过程（类似于 exe 安装程序）。用户如果有兴趣，请参考图 2.3 中的文档，其中全面描述了一个标准功能库开发的整个过程，并附录了几个 PCS7 库下的功能块源代码。此外，还可以访问西门子中文网站或西门子热线，获取更多详细信息。此处列举一些西门子中文网站中能获取的相关 FAQ，供大家参考：

http://support.automation.siemens.com/cn/view/zh/22109936
http://support.automation.siemens.com/CN/view/zh/22123899
http://support.automation.siemens.com/CN/view/zh/22779943

随本文档附录文中实例的开发源代码、Block Icon 及面板文件、PCS7 下 SCL 开发的一个典型模板文件供大家参考学习。