一方梯队嵌入式 SmartPLC 实时系统

V7.0

用户手册

摘要

1 简介

本手册介绍了一方梯队 SmartPLC 嵌入式实时系统相关内容，版本为 7.0.0
本手册没有关于如何使用 OpenPCS 编程系统的内容，如果需要请查阅 OpenPCS 编程系统用户手册。

1.1 概述

第二章介绍了如何移植 OpenPCS 到控制器。
第三章的内容包含了更多关于 OpenPCS 移植的细节。
第四章介绍了一些支持一方梯队 OpenPCS 编程系统并使得移植变简单的工具。
第五章介绍了关于 OpenPCS 构架运行的大量后台信息。
最后是术语表。

1.2 其他资源

如果本书无法解答您的问题，可考虑向下列资源寻求帮助：
如果你在 IEC1131 标准或是一方梯队 OpenPCS 编程系统的基本操作有问题，见用户手册”additional sources of information”部分。
当你安装了完全嵌入式 SmartPLC 系统后，你会得到一个包含在线信息的”.doc”文件。通过浏览这些文件可对本系统有个大概的了解。
一些我们的用户已经成功地通过本手册提供的信息将嵌入式 SmartPLC 系统移植到他们的硬件中。但是对大多数用户而言，在第一阶段的移植过程中需要一方梯队工程师的技术支持。

1.3 目录

2 嵌入式 SmartPLC 移植指导

2.1 移植准备

2.1.1 构造应用环境

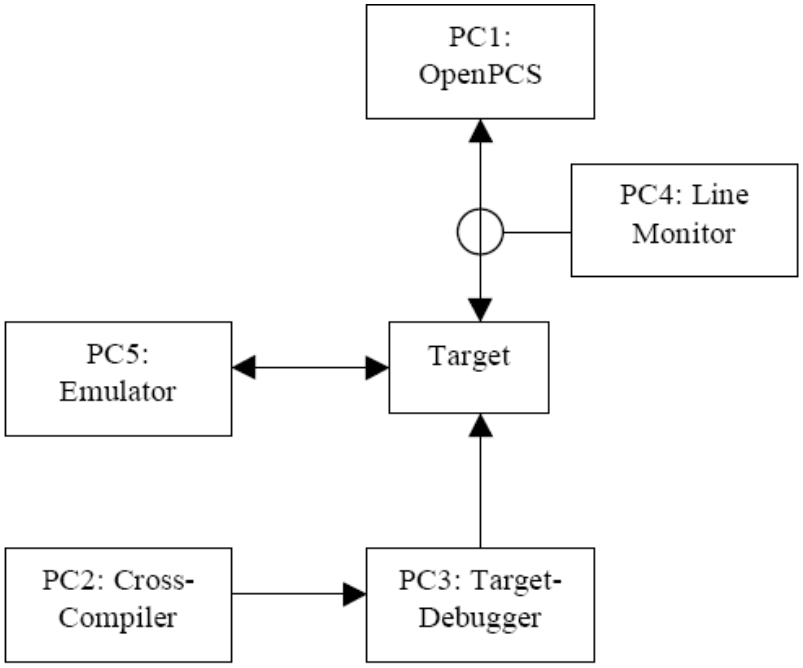


图 1 嵌入式 SmartPLC 应用环境

图 1 表示一个嵌入式 SmartPLC 应用环境。

在大多数情况下，此环境可以更简单。

Target： 表示担当控制器并管理嵌入式 SmartPLC 实时系统的计算机。不可或缺。

PC1： 用于运行 OpenPCS 编程系统的带 Windows 操作系统（2000，XP）的计算机。它通过 V24 串行总线或以太网与目标控制器连接，并需要 CD-ROM 驱动来安装 OpenPCS。

PC2： 为了移植嵌入式 SmartPLC 实时系统到目标控制器，需要一个 ANSI-C 编译器为控制器产生执行文件或下载文件。很多情况，此编译器不在控制器上运行，而是作为一个交叉编译器在其他 PC（或 UNIX 工作站）上运行，通常 PC2 可等同于 PC3。

PC3： 为了下载 SmartPLC 到控制器且监测与调试控制器的运行，需要另一个 PC，及 PC3。PC3 与控制器需要专有的连接，CAN 或其他串行总线都行。此 PC3 可等同于运行交叉编译器的 PC2 或控制器本身，但它不能等同于运行 OpenPCS 的 PC1。当程序正在目标控制器上运行时，如果程序无法调试，移植不是不可能完成，但要困难得多。

PC4： 实现控制器与 PC1 的通信与运行是一个重要的里程碑。能够实现通信是非常有意义的，因为在调试时经常会由于超时而中断通信。

PC5： 有时，仿真器适用于低级的调试应用程序。通常，需要一台电脑与仿真器接口连接，此 PC5 等同于 PC2 或 PC3。

当然，对于每个应用不一定需要所有组件，视需求情况而定。

2.1.2 准备工具

当要移植嵌入式 SmartPLC 到硬件时，确保以下项目已准备好。若在一方梯队公司完成移植，在“备注”列中标记的部分为需要准备的项目。若在自己公司完成移植，检查一方梯队公司规定了哪些项目。若在没有一方梯队公司的参与下完成移植，下表可作为参考清单。

序号	项目	备注
1	目标控制器：保证硬件稳定性	-
2	ANSI C-Compiler：用于为目标控制器创建代码	-
3	具备用 C-Compiler 编写控制器程序的能力	-
4	Serial-LINK(V24)：通过 ANSI C 语言运行子程序	-
5	OpenPCS 编程系统	×
6	完全嵌入式 SmartPLC 实时系统	×
7	PC1：搭载 OpenPCS 编程系统	×
8	PC2 搭载交叉编译器；PC3 用于调试目标控制器；PC5 用于仿真	-
9	PC4：提供 V24 串行通信	×
10	一根国际电话线——当需要专家建议的时候	×
11	一个收发邮件的邮箱——当需要专家建议和资料、工具、驱动程序等的时候	×
12	为其中一台电脑配备一台打印机，用于打印清单。例如，编译错误信息、运行结果文档等。	×

2.1.3 建立 SERTEST

见 4.1 节对 SERTEST 工具的介绍。在开始移植前，通过此工具检查是否具备正常的串口通信功能。

2.2 一方梯队嵌入式 SmartPLC 实时系统结构

在安装成功后，将得到类似下列的目录树：

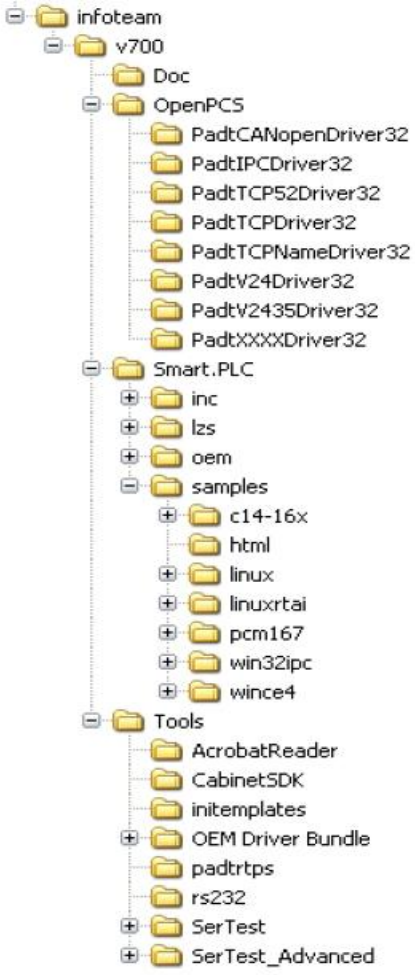


图 2：完全嵌入式 SmartPLC 实时系统结构

所有文件都被安装到 C 盘一个名为“infoteam”的目录下。每一个独立的版本都拥有一个以去除小数点的

版本号为名的目录。例如版本 7.0.0 安装在以下目录下：

C:\intoteam\V700.

子目录：

目录	子目录	内容
doc		SmartPLC 实时系统文档
OpenPCS		DLL 资源，用于 OpenPCS 与 PLC 通信
	PadtlPCDriver32	用于 SmartLIM 进程间通信的底层驱动
	PadtV2435Driver32	用于通过 V24 总线与运行 OpenPCS3 系统的控制器间通信的底层驱动
	PadtV24Driver32	用于通过 V24 总线与运行 OpenPCS4 系统的控制器间通信的底层驱动
	PadtTCPDriver32	用于通过 TCP/IP 标准接口与运行 OpenPCS4 系统的控制器间通信的底层驱动
	PadtTCPNameDriver32	用于通过 TCP/IP 标准接口与运行 OpenPCS4 系统的控制器间通信的底层驱动，控制器可由名称搜索代替地址搜索。
	PadtXXXXDriver32	生成底层驱动的模板
Smart.PLC		创建 SmartPLC 实时系统的资源
	\Inc	创建实时系统的头文件
	\Lzs	SmartPLC 实时系统不可更改的资源
	\OEM	需要你添加代码到 SmartPLC 实时系统的资源。可在工作环境中作为插入绝对代码的模板。
	\Samples	下列为 7 个准备移植实时系统的示例 C14-16x: files for a C167 cpu (Systec PLCmodule-C14) WinCE4: OS is Win CE 4.2 communication via TCP/IP Win32IPC: OS is Windows NT/95/98, communication via IPC, TCP/IP, or V24 Linux: This is for GNU-Buildenvironment and prepared for make and gcc. LinuxPPC: for PowerPC (MCP5200) architectures, with 4 byte alignment, without persistency and with native code executation. LinuxRTAI: files for a Linux (gcc) buildenvironment with RTAI (Real-Time Application Interface) Html: demonstrate embedding-features of OpenPCS online components (needs customization)
Tools		对开发 OpenPCS 有用的工具。对每一个工具的说明在 doc 文档或 readme 文件中可找到

2.3 建立串口通信

你可以用任何一种你喜欢的通信来支持与 OpenPCS 的通信，但是，我们建议你必须在所有事情都准备好并且运行正确后，才能启用 V24 串行通信和替换它。

建立串口通信是相当困难的。因此我们提供了一个测试程序 SERTEST，用于启动和运行串口通信。由于 SERTEST 与 OpenPCS 使用相同的接口，所以一旦 SERTEST 运行正常，OpenPCS 启动和运行就没多大问题了。

串口线选项为：

波特率　9600

数据位　8

无奇偶校验

停止位　1

上述参数可根据需要修改。通过接口传输的所有字节都不需要修改及译码（取值范围 0-255）。由于在无缓存模式下，接收函数不能保证能被频繁调用来接收每一个字节，所以输入与输出都需要缓存。例如可以写一个

中断程序，在每次收到数据时触发，将数据放入缓冲区。此缓冲区需要能够被实时系统的接口函数 SerReadComm 访问。所有你可对实时系统使用的函数都放在文件 Smart.PLC\OEM\V24\ser.c 中。

SERTEST 包含 2 个单独的程序。第一个作为一个 DOS 任务运行在 PC 上，第二个需要移植到目标控制器并且植入串行通信代码。它与实时系统使相同的接口函数。详见第 4.1 章。

2.4 文件管理

嵌入式 SmartPLC 包含各种各样的选项，所以编译嵌入式 SmartPLC 系统的第一步就是确定资源文件的设置。我们推荐你创建一个以公司名或 IEC1131 工程内部名称命名的目录：C:\infoteam\Vxxx\Smart.PLC(vXXX：:安装版本)例如：

C:\infoteam\v700\Smart.PLC\Your_Company

然后将所有相关文件拷贝到这个目录下，从而保证所有原始文件无法修改，且易于备份和参考。

现在拷贝下列文件到你的目录下：

Smart.PLC\LZS 下的所有文件，这些是实时内核中不需要修改的文件。

在下列文件中选择一个作为内存文件：szmbyte.c,szmbyt68.c,szmbyt86.或 szmint.c。仅能包含一个文件到工程中。详见第 3.1.13 和 3.1.15 节。

在 Smart.PLC\LZS\V24 中的 netv24.c，nlv24.c 和 v24.c，是 V24 通信子模块中的文件。为了建立不同的通信方式，如 CAN 通信，需要用相应文件替换掉上述文件。有两种模式可供选择，netv24.c 和 nlv24.c。nlv24.c 是较早的 V24 通信模块，需和 v24.c 一起应用。我们建议使用 netv24.c 文件。

在 Smart.PLC\OEM 下的 Lzsenv.c 文件包含了实时内核与工作环境间的主要接口函数。你需要检查这些函数并决定是否修改。在 3.6 节中的示例介绍了一些关于 LzsEnv.c 文件的例子。

完整的子目录 Smart.PLC\OEM\inc\OEM

在 2.3 节中提到的测试程序中的 Ser.c 文件运行在目标控制器系统中，它建立了一个 Smart.PLC\Lzs\V24 文件中串口通信子模块与在 2.3 节中移植的硬件系统间的接口。

2.5 编译

2.5.1 头文件

所有资源中的头文件的申明指令书写格式如下：

#include”subdir/hfile.h”

建议不要移动或复制头文件，但是要将编译器的寻找路径指向所有头文件的根目录。即你必须告诉编译器此子目录所在的位置。在大多数情况下，可通过设置一个环境变量 INCLUDE 实现。在 OpenPCS Development CD 标准安装中设置如下：

Set INCLUDE=C:\infoteam\v700\Smart.PLC\inc

在 Smart.PLC\OEM\inc 中的拷贝的头文件如下所示：

Set INCLUDE=C:\infoteam\v713\Smart.PLC\inc; C:\infoteam\v713\Smart.PLC\Your_Company

2.5.2 生成文件

创建一个生成文件用于存放上述所有资源。例如 linux 系统示例中的 samples\linux\makefile.

2.5.3 编译设置

OpenPCS 提供了一些可扩展的资源代码。这些扩展性由配置头文件 config.h 定义。默认设置如下：

```
#define _IP_PFLOW_
#define _IP_STRING_
#undef _IP_WSTRING_
#undef _IP_ANYDATE_
#define _IP_FLOAT_
#define _LZS_UPLOAD_
#define _LZS_DSALIGNMENT_1_
#define _LZS_8BITBYTES_
```

```
#undef _LZS_16BITBYTES_
#undef _LZS_32BITBYTES_
#define _LZS_NO_PERSISTENCE_
#undef _LZS_PERSISTENCE_DISKLIKE_
#undef _LZS_STATIC_PERSISTENCE_
#undef _LZS_PERSISTENCE_POINTER_
#undef _LZS_MIXED_PERSISTENCE_
#undef _IP_BREAKPOINTS_
#undef _LZS_NATIVCODE_
#undef _LZS_NCC_DIRECT_
#define _LZS_DWLWITHOUTSTOP_
#undef _LZS_DWL_SINGLE_SEG_
#undef _LZSDWL_RAWFILE_
#define _PASCAL_CALLING_CONVENTION_
#define _IP_ENABLE_ACCU_ANY_OPTIMIZE_
#undef _LZS_ERROR_LOG
#undef _LZS_SUPPORT_OPC_CONNECT
#undef _LZS_FORCE_VAR_RECEIPT_
```

编译所有文件并且确定没有警告与错误。

2.5.4 Visual Studio 2005 编译器

Visual Studio 2005 编译器不支持字符串功能（如 strcpy）。为了消除警告，请添加下面一行到通用头文件中

#pragma warning(disable:4996)

2.6 基础应用代码

当上述所有文件都被编译后，会在 lzsenv.c 文件中生成错误信息。所以在这个文件中，你需要加入硬件和环境指定代码。

2.6.1 LzsEnvGetTickCount

在实时系统中 LzsEnvGetTickCount 被用于所有与时间相关的函数。例如，串行通信超时就是由该函数测量的。

该函数返回一个以 ms 为单位的 LZSDWORD 类型的值，所以实时系统可通过调用两次本函数并相减来估算运行时间。

2.6.2 LzsEnvMemAlloc/LzsEnvMemFree

这些函数可充当标准 C 语言库中内存分配和释放函数的替代函数。在标准 LzsEnv.c 文件中可以完成存储管理工作。唯一需要你提供的是一个指向连续存储块的指针。这个存储空间是一个静态 LZSBYTE 队列或仅仅是一个由你提供的指向存储空间的固定指针。IECMEMORY 可作为此指针使用。在 smart.plc\oem\LzsEnv.c 中，已有如下定义

Static LZSBYTE IECMEMORY[LZS_MAXMEM];

该存储空间必须足够大以至于可存放所有 IEC 程序的信息，例如代码段，数据段，初始数据段和所有中断处理段。

你也可以用操作系统的相关函数来完成。

LzsEnvMemAlloc 申请了一个可设置容量的内存，用于实时系统存储信息。

LzsEnvMemFree 用于释放此空间，以便下次使用时再申请。

2.6.3 LzsEnvFreeNonlECMemory

当内存指针在 IECMEMORY 外面时，LzsEnvMemFree 会调用该函数（如 2.8.10 中 _LZS_PERSISTENCE_MIXED_）。OEM 在此执行内存管理的绝对代码。

2.7 运行

2.7.1 植入代码

接下来要做的事情是：在你的系统的主入口点处集成嵌入式 SmartPLC 实时系统。鉴于目标控制器上的操作系统中可能仅有一个 main()函数或一些任务控制程序，在 Smart.PLC\OEM\main.c 文件中有一个简单的主程序例子可供参考。最重要的步骤是：

用 LzsEnvInitialize()函数初始化实时系统

这两个实时系统的主循环按照无限循环或任务 LzslpMainLoop();LzsCmdMainLoop()的方式运行。

在编译通过后系统首次开始运行。

2.7.2 连接

运行后编程系统和实时系统连接将会被检测到。确保此连接在在线服务器中已被定义且可在资源属性中被选中。然后双击浏览器中的资源进行连接。

2.7.3 测试

编写一些简单测试程序，下载到 PLC 并使之运行。使用测试和试运行调试函数来监测程序。

2.8 高级平台绝对代码

2.8.1 物理 I/O

5.1 节详细介绍了 OpenPCS 进程映象布局。

当所有准备进行到这里时，你需要在 LzsEnv.c 文件中的 LzsEnvInitProcImg\LzsEnvReadProcImg \LzsEnvWriteProcImg 函数中写入使用物理 I\O 的代码。首先要确认需要多少输入、输出及标志。这意味着实时系统所使用的进程映象的布局。为了告诉编译器你的布局，你需要在硬件配置文件中配置正确的值（例如：C:\Documents and Settings\All Users\Application Data\infoteam Software\OpenPCS2008\Openpcs.520\modules\ smartsim.ini）

更多介绍见 3.2.4 节。然后你就可以编写一个测试程序，使用直接变量（%I 和%Q）并在目标控制器系统上运行。

当实现物理 I\O 时，我们建议按下列顺序处理：

数字输入

数字输出

模拟输入

模拟输出

当实现数字输出时，编写一个简单程序如下所示

```
PROGRAM count
VAR
        a : int;
        b at %QW0.0 : int;
END_VAR
LD a add 1
ST a
ST b
END_PROGRAM
```

用来测试数字输出。如果物理输出上有延迟，那么要把它设置为一个计时任务（而不是循环任务）以防止超过硬件范围。

当实现模拟 I\O 时，第一次设置一个确定的数值作为模拟输入的初始值，例如：

```
LzsEnvReadProcImg (...) {
        ...
        *((float *)(pProcImgSeg_p+..)) = 3.1415;
}
```

这将有助于判断错误是来自于不正确的字节命令还是不同的浮点数表示。

2.8.2 输出禁止

当 PLC 处于停止模式时，你可能想要将输出设置成一个明确的状态。可使用 LzsEnvDisableAllOutputs 函数。参数 wMode 指明函数调用的原因，它可能是下列情况中的一种：

```
LZS_DISABLE_OUTPUTS_HWSTOP          /*RUN/STOP switch to stop*/
LZS_DISABLE_OUTPUTS_SWSTOP:          /*TUI -> STOP*/
LZS_DISABLE_OUTPUTS_ERRORSTOP:       /*execution error STOP*/
```

2.8.3 下载

开始下载前，函数 LzsEnvBeginDownload 将被调用，用于初始化。

2.8.4 进程映象特殊内存管理

进程映象（和网络映象）不能立即通过 LzsEnvMemAlloc 直接分配，因此需要调用以段类型为参数的函数 LzsEnvSpecialMalloc。然后可以按进程映象的需求配置内存，通过调用 LzsEnvMemAlloc 函数或设置确定的进程映象的地址指针。

2.8.5 移动段

每个段在被写入段表前都将调用 LzsEnvMoveSegment 函数。并且可以将段移动到另外的地方。见函数的解释。

2.8.6 运行/停止转换的运行模式

当使用 LzsCtlRunStopSwitch 函数启动程序执行时，LzsEnvHardwareRun 函数可用于设置运行模式。

2.8.7 系统状态转换

每次内部系统转换时，LzsEnvRegisterSysState 函数都会被调用。比如你可以设置一个 LED 灯来表示 PLC 的运行状态。

2.8.8 OME 版本识别

为了使硬件通过 OpenPCS 实现可编程，并且可识别其他的 OEM 硬件，需在头文件 SmartPCL\Your_Company\Oem\Oem_Def.h 中定义 OEM_ID_NR。此 OEM-ID 在移植支持期间已告知。如有疑问请联系 infoteam。

2.8.9 增加固件函数块

OpenPCS 在没有 OEM-specific 函数块的情况下也可以运行。但是增加你自己的函数块到 OpenPCS 固件中，是根据你的需求扩展和移植 OpenPCS 最容易和强大的方式。3.5.2 节将逐步介绍如何设计固件函数块。

2.8.10 持续性

通常希望 PCL 在掉电后可自动重启，为了支持此功能，你需要如下两个性能：

自动重启程序——通常在 main()函数中定义。

Persistency（持续性）：当前运行的程序必需在 PLC 中安全地存储，以便其在掉电后能够重载或其他意义上的重构。

为了支持持续性，至少需要函数 LzsEnvSaveByte 和 LzsEnvRestoreByte。像其他 LzsEnvSave…和 LzsEnvRestore…类函数也可使用，但是仅用于特殊变量的持续性。下列持续性变量是可用的：

- _LZS_NO_PERSISTENCE_

禁用 Persistency

・_LZS_PERSISTENCE_DISKLIKE_

用一个文件存储/装载 IEC 程序

・_LZS_STATIC_PERSISTENCE_

用一个 ROM 装载 IEC 程序

・_LZS_PERSISTENCE_POINTER_

且一个 Flash 存储/装载指针，需要电池缓冲 RAM

・_LZS_PERSISTENCE_SAVERAM_

复位后部分变量保持它们的值（通过将其申明为无需初始化变量），需要指定编译器和电池缓冲 RAM

・_LZS_PERSISTENCE_MIXED_

在程序下载时，指定类型的段通过 LzsEnvMoveSegment 函数将其移到不易变的内存空间（如：电池缓冲 RAM，flash 等）。只有指向这些段的指针才被保存/清空，其他所有段将被全部保存/清空。
为了定义哪些段将被做这种处理，你需要修改函数 LzsEnvMoveSegment 和 LzsEnvIlSegTypeMoved（见例 lzsenv.c 中的代码）

2.8.11 命令检查

在文件 IP_JTAB.C 中找到函数 IpExecCommand。看完解释后转到 switch(CurrIpCmd)部分。仔细检查你正用的编译器能否以你所期望的正确方式生成机器语言。例如，当编译器编译一条”if”结构语句时，大多数都知道怎样通过一个快速跳转表来完成。

有其它能使编译完成得更好的结构，但是编译器无法执行，所以除此之外没有更好的解决办法。如果你的编译器出现了编译困难，可替换一个更好的编译器能识别的结构来执行。

2.8.12 错误记录

使能错误记录特性可为实时系统提供一个特殊的段来存储重要的错误信息。编程系统提供了一个下载错误记录段并转换为相应错误信息的功能。

错误记录功能通过定义_LZS_ERRORLOG_实现。方法 LzsErrlogWriteErrorlog1 到 LzsErrlogWriteErrorlog5 通过各自的参数增加了一个记录信息和时间标记到错误记录中。
（更多说明见 3.3.4 节）

2.8.13 与移植挂钩的其他函数

文件 LzsEnv.c 中定义的所有函数都被用于与嵌入式 SmartPLC 集成平台的特定功能挂钩。浏览本文件中的所有函数，根据提供的注释和文档决定哪些可能或不用。

2.8.14 保持

见 5.3.12 节

2.8.15 OPC 连接

当和 OpenPCS 连接（在线）的时候，嵌入式 SmartPLC 提供了与 OPC 服务器（SmartLINK）连接的能力。例如，在没有干扰 OpenPCS 在线功能性时，允许通过 OPC 运行可视化。使能此功能需要以下步骤：

通过以下定义得到 OPC 支持

Config.h _LZS_SUPPORT_OPC_CONNECT

可用的并行 OPC 连接数量（通过多样的 OPC-Server 实例）通过以下定义被设置

#define_LZS_MAX_NUMBER_OPC_CONNECTIONS 2

完成底层通信接口设置（见样本 win32ipc 中文件 nltcp.c 中的例子）

2.8.16 注销后保持变量值

在连接中断或注销 OpenPCS 后，默认变量会被自动清除。通过在处理中以如下方式定义_LZS_FORCE_VAR_RECEIPT_可改变此默认行为：嵌入式 SmartPLC 在线下通过 OpenPCS 来保持强制变量值。

根据 PCS 的重新连接，变量窗口中的当前状态被传输到 OpenPCS 并且显示在观察窗口中。注意：_LZS_FORCE_VAR_RECEIPT_的定义不与 OPC 连接兼容。
（见_LZS_SUPPORT_OPC_CONNECT 的定义）

2.9 概要

下表显示了逐步移植嵌入式 SmartPLC 实时系统的步骤，由一方梯队硬件公司在两天内实施。

No.	Task
1	解释提供的软件的结构
2	恢复通信
2.1	使用在工具目录下的测试程序建立串口通信
2.2	对测试程序使用不同的参数
3	编译实时系统
3.1	为移植收集需要的资源作为出发点
3.2	将INCLUDE路径设置到Smart.PLC\inc目录下
3.2	创建一个生成文件来编译所有使用这个模块生成文件的资源
2.3	和目标编译程序一起编译并且解决所有的警告和错误
3	填空（由#error标记）
3.1	LzsEnvMemAlloc和LzsEnvMemFree
3.2	LzsEnvGetTickCount
3.3	由host main 完善main
3.4	运行到空循环为止
4.	恢复通信（实时系统）
4.1	用在第2步写的代码替换ser.c中的空白
4.2	使逻辑命令工作
4.3	使下载命令运行
5	观察实时系统工作
5.1	下载一些简单的程序并让它们运行
5.2	使用T&G调试函数来监听程序
6	加入进程映象
6.1	检查在硬件说明文件中的设置
6.1	在LzsEnv中填入代码
6.2	下载程序来设置输出
6.3	下载程序来读取输入和设置输出
7	封装
7.1	建立移植备忘录，列出所有移植期间引起的核心修改，公开并且在之后解决。
7.2	摘要：邀请销售参与移植结果的讨论，并且讨论立即为用户解决的主题。

3 常规移植问题

3.1 常用问题

本节主要回答在一方梯队完全嵌入式 SmartPLC 实时系统移植中频繁发生的问题。

3.1.1 可零售的文件

一方梯队嵌入式 SmartPLC 实时系统仅以内部使用的方式出售。全部买下嵌入式 SmartPLC 实时系统组件将授予你以下权利：如本文所示修改 OpenPCS 编程系统并且使之与你的硬件系统相结合。其他修改的权利由一方梯队公司保留。

3.1.2 怎样用进程映象设置 I/O 逻辑接口

见文件 LzsEnv.c 中的函数 LzsEnvInitProcImg,LzsEnvReadProcImg,LzsEnvWriteProcImg。

3.1.3 怎样在没有进程映象的帮助下设置 I/O 逻辑接口

3.1.3.1 改进 LZS 函数

文件 SmartPLC\lzs\szmbyte.c 中包含了很多存取内存的函数，例如 LzsMemFarGetDword 用于从”far”内存中获取一个双字类型数，即一些非局部数据段。修改这些函数可为进程映象提供一个特殊的处理，此处理过程为：找出被寻址的段是否是进程映象段。

由于此比较可能会影响到你的系统的运行时间，所以应找出你需要的函数来修改。你需要修改”near”函数吗？它们在你工作环境中可以是只读或只写的吗？

此方法通常不适合于 OpenPCS 的本地编译器。

3.1.3.2 提供绝对函数

为了立即使用硬件，无论哪个方法执行绝对函数或函数块均可绕过进程映象。绕过进程映象不会引发运行时间的损耗。

3.1.4 怎样用 C 语言（汇编语言）实现子程序固件化

通过编写代码来执行你自己的固件函数逻辑。见文件 Smart.PLC\lzs\stdfb.c 中的例子。修改文件 Smart.PLC\lzs\stdfbtab.c 中的固件程序跳转表，使得指针指向你的新的固件程序。并为从跳转表中得到的新程序块指针作注释。

编辑硬件配置文件（例如：C:\Documents and Settings\All Users\Application Data\infoteam Software \OpenPCS2008\Openpcs.520\modules\smartsim.ini）用于说明固件块的定义。注意跳转表中指针值调整的详细说明。详见 3.2.4 节。

在你的 OpenPCS 安装目录下的 Openpcs.520 的局部子目录，修改 GLOBPROT.inc 文件，将你的固件块原型包含进去。

设计数据段和各自的初始数据段（见 3.5.1 节）。在硬件配置文件中，为每个固件块设计的初始数据段，在函数块的输入数据部分被设置。（例如 C:\Documents and Settings\All Users\Application Data\infoteam Software\OpenPCS2008\Openpcs.520\modules\smartsim.ini）

背景：编写一个固件程序，你很可能需要处理参数。这必须遵守 OpenPCS 的参数传递方法。见 3.3.3 节或文件 SmartPLC\lzs\stdfb.c 中的例子。

注：目录为隐藏目录，查看时需设置文件夹选项。

3.1.5 与嵌入式 OpenPCS/SmartPLC 适应的 IEC61131-3 标准

移动、旋转和取反（更多 OpenPCS 陆续增加的）标准函数像指令一样被 OpenPCS 虚拟机应用程序立即执行。也就是说，为了提供此性能，函数调用的开销将被保存。见文件 SmartPLC\Lzs\Ip_Cmds3.c 中这些函数的最新清单。

标准函数块在文件 SmartPLC\Lzs\stdfb.c 中被定义，可在最新的清单中查询。通常包含模块 SR,RS,R_TRIG,F_TRIG_CTU,CTD,CTUD,TP,TON,TOF。

3.1.6 如何用嵌入式 SmartPLC 实现多任务处理

嵌入式 SmartPLC 不需要特定的多任务操作系统，也不用捆绑一个实时操作系统。恰恰相反，控制器的每一部分都以资源的形式出现，使之可移植到几乎任何一个操作系统上，或者在不用操作系统的情况下运行。因此，在嵌入式 SmartPLC 资源中的多任务处理可理解为工作模式，此模式可被替换，一种处理是通过你使用的操作系统的一个定制的处理，使能有用的特性来替换它并且使之正确适应，另一种处理是假使不需要操作系统，则调整到最好的支持。

对于一些标准操作系统，一方梯队也做了这样的裁减，所以如果你使用的是标准操作系统，可向一方梯队寻求帮助。当然，一方梯队将提供移植到你的操作系统或硬件的技术支持服务。

详见 5.3.3 节

3.1.7 如何用嵌入式 SmartPLC 实现中断处理

见 3.1.6 节关于多任务处理的说明，原理相同。

3.1.8 如何处理具有相同优先级的任务

同一优先级任务的相关调度不由 OpenPCS 定义，所以你不需要依赖任何特别的调度。

具有相同优先级的任务按任务名字的字母顺序来调度，但我们将在以后版本中改变这样方式。我们建议你不使用这种处理方式。

3.1.9 什么时候及怎样更新

见 5.14 节

3.1.10 在 OpenPCS 中发现问题后的处理办法

见 5.14.1 节

3.1.11 实数的表示

对于浮点数（数据类型为 REAL 和 LREAL），嵌入式 SmartPLC 使用微软支持的 Visual C++编译器所用格式。按照微软的设定，此格式与 IEEE 数值标准一致。数据格式定义如下：

REAL:1 位符号位，8 位指数位，23 位尾数位

LREAL: 1 位符号位，11 位指数位，52 位尾数位

一些编译使用不同的表式格式，例如 Tasking`s C167 编译器，在硬件配置文件中通过打开 FloatSequence 来提供支持。

3.1.12 怎样写通信驱动程序

找到你的目标系统资源目录中 PADXXXXDriver32 的位置，例如 C：\infoteam。按照下列步骤修改 Microsoft Visual Studio 工程使之执行你的驱动程序。更多细节见 5.10 节，也可参考资源代码中的其他驱动程序。

• 拷贝 PadtXXXXDriver32 目录到另一个目录下（相同路径），并重命名，用你自己的驱动程序名字代替 XXXX（任何 4 个字母组合）

• 通过文档编辑器或者 grep 工具（例如像 grepwin 一样的免费工具），你可以替换工程中所有的文件中的 4 位特殊字符串。

• “XXXX”替换为驱动器名字（4 个字母）

• “YYYY”替换为工厂 ID（4 个数字）

• “ZZ”替换为驱动 ID（2 个数字）

• “FFFF”替换为公司名字（字符串）

• 打开 Microsoft Visual Studio 2003 解决方案

• 完成驱动函数代码。

- 执行“重构”并且拷贝生成的.dll 文件到 OpenPCS 安装目录下
- 使用 regsvr32.exe 注册此通信驱动器

3.1.13 对大/小端字节存储次序的处理

OpenPCS 编程系统可用大/小端字节存储次序数据格式编译程序代码。由硬件配置文件中的 Motorola 控制。

小端存储次序

Motorola=0 ;

大端存储次序

Motorola=1

注意所有的固件函数块的初始数据段，在硬件配置文件中给出定义的必需以合适的格式（见 3.5.2 节）

实时系统需要选择合适的内存存取模式。下列模式可选其一：

Szmbyte.c:小/大端字节次序，需正确的对齐（见 3.1.14 节）

Szmbyt68.c 大端字节次序，不需特殊对齐

Szmbyt86.c:小端字节次序，不需特殊对齐

3.1.14 处理机需要不同的对齐来存取数据

OpenPCS 编译器在所有数值被对齐后可产生数据段。1、2、4 字节对齐可供选用。由硬件配置文件中 Alignment 的值决定：

Alignment=1	所有的数据类型放置于下一个空地址（偶数或奇数）。数据间没有空隙
Alignment=2	数据类型大小>1 时放入偶地址。数据间可能有空隙
Alignment=4	数据类型大小>=4 时放入可被 4 整除的地址，大小为 2 时放入偶地址，大小为 1 时逐个放置

注意所有的固件函数块的初始数据段，在硬件配置文件中给出的定义必需以合适的格式（见 3.5.2 节）

在实时系统中，必需在 config.h 文件中选择一种数据段排列方式的定义

#define _LZS_DSALIGNMENT_1_

#undef _LZS_DSALIGNMENT_2_

#undef _LZS_DSALIGNMENT_4_

当使用 4 字节对齐时，还需要检查_ALIGN_STRUCT_HEADER_4 的定义。由硬件配置文件中的 AlignStructHeader 的值决定。

3.1.15 控制器数据位不仅仅是 8 位

若支持 16 位或 32 位的控制器，需做以下工作：

在硬件配置文件中将 OpenPCS 编程系统的”ByteSize”和”Alignment”按要求配置

应用于 16 位控制器

ByteSize=2 Alignment=4 ;

应用于 32 位控制器

ByteSize=4 Alignment=4

注意所有的固件函数块的初始数据段，在硬件说明文件中给出定义的必需以合适的格式（见 3.5.2 节）

实时系统需选择一种内存存取模式：

szmint.c

并且在 config.h 中有如下定义：

#define _LZS_16BITBYTES_ or #define _LZS_32BITBYTES_

#define _LZS_DSALIGNMENT_4

3.2 通信

3.2.1 概述

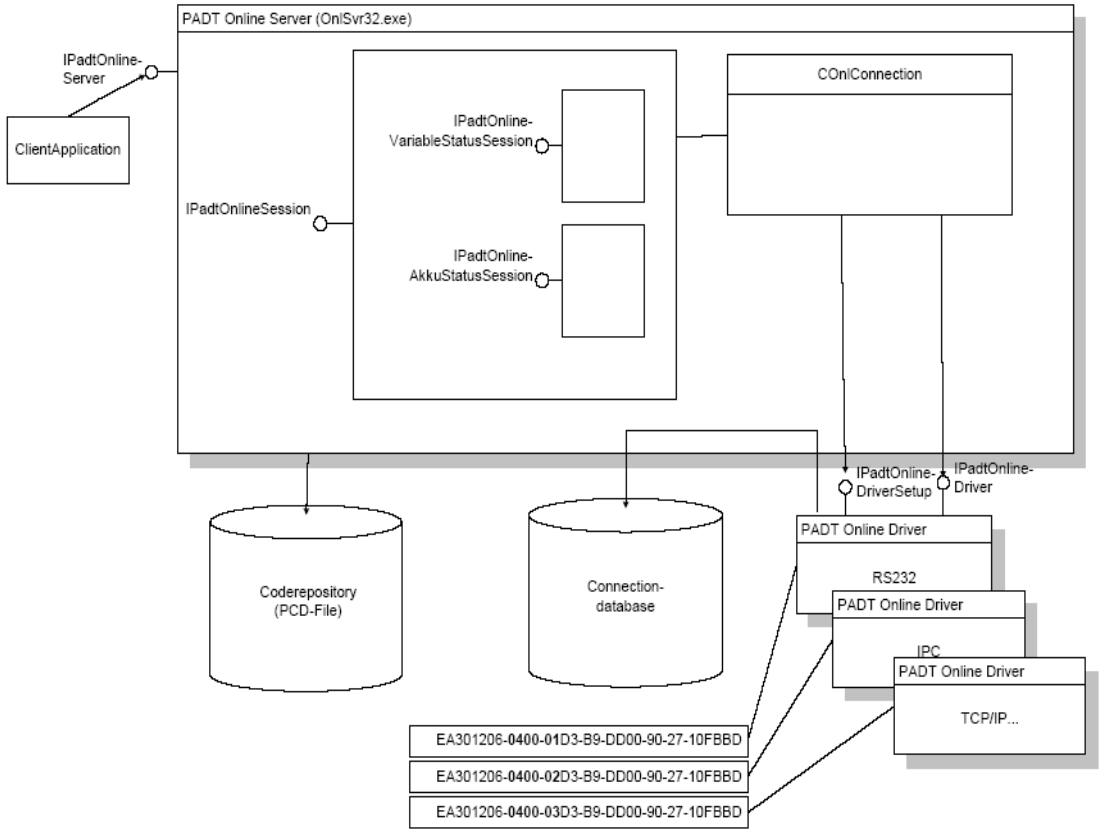


图 3 在线服务器架构

3.2.2 命令

下列命令是可用的（在 lzstypes.h 中由 tLzsCmdLst 一一列举）。命令的参数在命令值后。参数的第一字节为参数部分的长度。

Description	ENUM	VALUE	Parameter
Login	kLzsCmdLogin	0x01	return buffer size
Logout	kLzsCmdLogout	0x02	./.
Reset	kLzsCmdReset	0x03	./.
Download Resource	kLzsCmdDwlResource	0x04	sizeof Taskdefinition Seg-ment(WORD) incremental (WORD)
Download Program	kLzsCmdDwlProgram	0x05	program number (WORD)
Download Segment	kLzsCmdDwlSegment	0x06	segment size (WORD) program number (WORD) segment number (WORD)
Set State	kLzsCmdSetState	0x07	state (BYTE)
Get Resource Version	kLzsCmdGetResVer	0x08	./.
Get Firmware Version	kLzsCmdGetPlcVer	0x09	./.
Discard Watch Table	kLzsCmdDiscradWatch	0x0A	./.

Enable Watching	kLzsCmdEnableWatch	0x0B	kLzsWatchAll or kLzsWatchSingle
Add Watch Instruction	kLzsCmdAddWatchInstr	0x0C	access type (BYTE) program number (WORD) segment number (WORD) offset (WORD) size (WORD)
Set Variable	kLzsCmdSetVariable	0x0D	see AddWatch data to set
Get Error Information	kLzsCmdGetErrorInf	0x0E	./.
Get Segment Information	kLzsCmdUplGetSegInf	0x0F	segment type (BYTE) 0xFFFF program number (WORD)
Get Segment Data	kLzsCmdUplGetSegData	0x10	segment type (BYTE) segment number (WORD) program number (WORD)
Start Powerflow	kLzsCmdStartPflow	0x11	program number (WORD) instance number (WORD) start offset (WORD) end offset (WORD)
Stop Powerflow	kLzsCmdStopPflow	0x12	./-
Download a single segment	kLzsCmdDwlSingleSegment	0x13	segment size (WORD), program number (WORD), segment number (WORD), resource version (DWORD), program version (DWORD), build date (DWORD), load date (DWORD)
Force Variable	kLzsCmdForceVariable	0x14	see AddWatch data to set
Unforce Variable	kLzsCmdDisableForceVari-able	0x15	see AddWatch data to set
Delete watch instruction	kLzsCmdDelWatchInstr	0x16	WatchId(WORD)
Add Multiple Watch Instruction	kLzsCmdAddMulWatchInstr	0x17	Mutliple watch requests: access type (BYTE) program number (WORD) segment number (WORD)
Download multiple segments at once	kLzsCmdDwlMultiSegment	0x18	offset (WORD) size (WORD) program number (WORD), segment size (WORD), number segments (WORD)
Request watch data	kLzsCmdRequestData	0x20	max size of return buffer (32k)

Download data while PLC is running	kLzsCmdDwlWithoutStop	0x21	Segment size (DWORD)
Get Segment Information	kLzsCmdUplGetSegInf NoError	0x2A	segment type (BYTE), segment number (WORD), program number (WORD)
Put resource to persistent storage	kLzsCmdSaveSystem	0x2B	./.
Check resource version	kLzsCmdCheckResVer	0x2C	project name [32], resource name [32], timestamp(DWORD), build date(DWORD), load date(DWORD)
Download raw file	kLzsCmdDwlRawFile	0x30	filename, filesize
Download segment of raw file	KLzsCmdDwlRawFileSeg-ment	0x31	segsize, segnum
Download last segment of raw file	KLzsCmdDwlRawFileLast-Segment	0x32	segsize, segnum, filetimeLastChanged
Postprocessing finish download	kLzsCmdDwlCompletion	0x87	./.
Set Variable with long data	kLzsCmdSetVariableLong	0xB1	see AddWatch
Force Variable with long data	kLzsCmdForceVariableLong	0xB2	see AddWatch
Command set multiple variable	kLzsCmdMultiSetVariable	0xB3	Size of data
Get PLC capabilities	kLzsCmdGetPlcCapabilities	0xB6	./.
Discard force table	kLzsCmdDiscardForceTab	0xB7	./.

3.2.3 扩展返回代码

下列扩展返回代码存储在返回值附加数据中。

Description	ENUM	value
no change	kLzsNoChange	0x00
state of PLC updated	kLzsStateChg	0x01
error	kLzsError	0x02
new powerflow data avail-able	kLzsPflowData	0x03
watch ID	kLzsWatchID	0x05
watch data	kLzsWatchData	0x06
version of resource	kLzsResVer	0x07
version of firmware	kLzsPlcVer	0x08
error string	kLzsErrInf	0x09
segment information	kLzsSegInf	0x0A
raw file information	kLzsRawFileInf	0xA6

3.2.4 OpenPCS4.1 V24

3.2.4.1 由驱动程序向实时系统发送数据

包：

驱动程序把一个数据包划分为几个最大容量为 120 或 121 的数据块。第一个数据块的第一个字节表示包的类型，如果大于一个字节则顺延。最终，每个数据块的数据长度（除去包类型说明字节）是恒定的 120 字节且可被 2、3、4、5、6、8 整除。这使得实时系统产品的设计避免了大量的拷贝和内存分配问题。也就是说，一个 120 字节的大数据块可由一个 6 字节的存储布局来映射，且不需要处理包含在下一个数据块的剩余数据。

在数据块头中有一个标记标志这个数据块是第一个或最后一个数据块。

应答和重试：

每个数据块发送后将收到来自实时系统的一个字节的应答帧。如果数据是损坏的，实时系统可以要求重新发送。驱动器在 3 次尝试发送且都因超时而中断连接后，将拒绝本次发送。

校验和：

校验和在每个数据块中被随机计算并添加到数据块开头。实时系统将验证此校验和的值，判断是否要求重新发送。

协议结构：

• 数据块

DWORD	WORD	BYTE	DWORD	BYTE*
包大小	数据块大小	结束数据块标志	校验和（随机）	数据块数据

• 结束数据块标志

低 4 位=结束数据块标志，高 4 位=开始数据块标志

0x0F = Last

0xF0 = First

0xFF = First & Last

• 应答帧：

BYTE
应答帧

0xF0 = 数据块接收完成

0x0F = 数据块损坏，请重发

3.2.4.2 接收来自实时系统的数据

由于在线服务器具有处理大规模数据的能力，所以来自实时系统的数据包不需要被分割成块。在线服务器会检查数据并在必要的时候要求重发。

校验和：

校验和在每个数据包中被随机计算并添加到数据块开头。在线服务器会检查检验和的值并在必要的时候要求重发。

协议结构：

• 数据包：

DWORD	WORD	BYTE	BYTE*
包大小	检验和	数据类型	数据

• 应答帧：

BYTE
应答帧

0xF0 =包接收完成

0x0F = 包损坏，请重发

在示例文件\\infoteam\v400\smart.plc\samples\win32ipc\v24\netv24.c 中，检验和的创建和求值可由 V24_CHECKSUM 的定义或不定义来选择。你可以在 ser.h 中这样做。

3.2.5 OpenPCS3.5 V24

3.2.5.1 OSI 协议层

在 7 层 IOS-OSI 通信模型中，运行在 PC 上的 OpenPCS 和 PLC 间的通信建立在其中 4 层的基础上。

3.2.5.1.1 物理层（OSI 第一层）

RS-232 协议用于这一层

参数设置如下：

8 位数据位

无奇偶校验

1 位停止位

最大 19200 波特率

3.2.5.1.2 数据链路层（OSI 第二层）

为了恢复数据在物理传输过程中的错误，所有数据将被转换为下列报文形式：

报文（发送数据）

报头	类型	报文地址	长度	……	LRC16HI	LRC16LO
0x00	DATA_ID Tx REQU_ID ERROR_ID	连续数字		包		

报文的以全 0 字节开头。数据的第 0 个字节以重复的 0 标记。报文的校验通过发送两个字节的纵向冗余校验码（校验和）完成。根据 ACKN_ID 类型的报文中 POS_ACKN 或 NEG_ACKN 验证的校验和，可判断报文是否被接收。

0x00	TYPE	RECEIPT	ID	LRC
0x00	ACKN_ID Rx	R_info[0] POS_ACKN NEG_ACKN	R_info[1] Telegramid	

3.2.5.1.3 网络层（OSI 第三层）

网络层封装了数据包发送的的包信息。如果必要的话它可以将数据分为多个包。包类型用最后两个有效位来编码。有以下 4 种类型的包：

- PT_SYNC 同步 = 0x00
- PT_CMD 控制包 = 0x01
- PT_DATA 数据包= 0x02
- PT_REC 接收包= 0x03

如果发送不成功，网络层的最大重发次数为 MAX_TxREPEAT=3

包结构：

包头	CRC16	CRC16	Data……	……

包类型说明：

FIRST P.	LAST P.					Packet-	type

3.2.5.1.4 应用层（OSI 第七层）

应用层提供了与任何 PLC 的通信接口。同时具有发送命令和接收应答的能力。命令包括 1 字节的标识符，1 字节的参数数量和 0 到 n 字节的参数。

命令：

CMD	PLEN	PARAMETER	……	……

3.2.5.2 通信示例

在本章节，我们提供了一些在 V24 总线上常用的数据包传输示例。

3.2.5.2.1 00-01-01-05-C1-01-02…(登陆)

通常第一包从 PC 到控制器的下传发送如下所示：

00-01-XX-c1-01-02-10-10-00-00-bd

00：包头

01：DATA_ID——数据包

XX：数据包#XX（下传），通常 XX=01

05：数据长度，5 字节：c1-01-02-10-10

00 DB：校验和

在数据中：

C1——二进制 11000001——第一包+最后一包+命令包

在命令包中：

01：kLzsCmdLogin，表示登陆命令

02：后跟 2 字节数据

0x1010：PC 的接收缓冲区容量为 0x1010 字节

3.2.5.2.2 00-02-06…（应答）

常用数据包为：

00-02-06-AA-BB

AA 表示数据包正确的应答

00：包头

02：ACKN_ID——应答包

06：POS_ACKN——正确应答，即数据接收成功应答

AA：数据包#AA

BB：前 3 个字节的校验和

3.2.5.2.3 00-01-XX-09-C3-00-00-05(登陆应答)

另一个常用的包就是从目标控制器发送到所登陆 PC 的响应应答

00-01-XX-09-C3-00-00-05-00-00-01-03-00-00-01-00-00-00-DB

00:包头

01：DATA_ID-数据包

XX：包#XX(上传)，通常 XX=01

09：数据长度，9 字节

C3-00-00-05-00-00-01-03-00-00-01-00-00

00 DB：校验和

在数据中：

C3：二进制 11000011——第一包+最后一包+应答包

应答包

00 00：返回代码（0=ok），成功

05 00 00：后跟 2 字节的扩展返回代码

01：kLzStateChg，实时系统报告状态改变

LzStateChg 通知后的 4 个字节表示新的状态

03 00 00 01 00 00 = 0x00010003

这是一个双字类型的数据，见 lzsdefs.h 中 LZSSTAT 的定义。表示含义如下：

0x00000001:LZSSTAT_OK，控制器已就绪

0x00000002:LZSSTAT_LOGIN，调试已联机

0x00010000:LZSSTAT_ACCPT_DOWNLOAD，控制器正准备下载程序

3.2.5.2.4 00-04-04-08（命令错误）

通常，若在通信系统中出现某些错误，会发出如下数据包：

00 04 04 08

00：包头

04：ERROR_ID

04：ERROR_ID

08：校验和

3.2.5.2.5 登陆序列的跟踪示例

对如下 T&I 程序输出的跟踪解释了 PC 与 PLC 连接期间的通信。

```
                NET NetInitialize
                  COMInitialize
                  COMCreateTxBuffer
                  COMCreateRxBuffer
SKS SksSendCmd #01# login
                NET NetRecData
                NET NetSendData
                net(int) NetCalcCRC
COMGetTxBufferBalance{c1} packet type PT_CMD first and last
packet
                  COMPutMem{01} {02} {10} {10} (RX_BUFFER_EMPTY) login command
                                                                                  with parameter
                  COMSendTxBuffer
                     TX TxKernel
                     TX TxData<00> <01> <01> <05> <c1> <01> <02> <10> <10> <00> <00> <db>
                  telegram (header ; type=1; ID=1; double zero; lrc)
                              TX WaitResponse[00] telegram from PC
                       RX RxKernel[02] type ACKN
                       RX RxKernelACK[06] [01] [09] POS_ACKN for telegr 1
                       TX Brancher
                       TX TxAcknRec
                  COMClrTxBuffer
                NET NetRestartRxTimeOut[00] answer
                  COMWatchHandler Rec. OK
                    RX RxKernel[01]
                    RX RxKernelData[01] [09] [c3] [00] [00] [05] [00] [00] [01] [03] [00] [00] [01] [00] [00] [00] [00] [d8]
                TX TxAckn<00> <02> <06> <01> <09>
                NET NetGetRxStatus
                  COMGetRxMessageCount(c3)
                  COMGetRxMessageSize(00) (05) (00) (01) (03) (00) (01) (00) (EOT)
SKS SksEvaluateAppendData L änge:0005                    length 5
SKS SksEvaluateAppendData Typ:01                         type 1 = change of state
```

说明：在网络和发送/接收缓冲区初始化后，SksSendCmd()函数通过命令标识符 KLzsCmdLogin 被调用。SksSendCmd()调用一个 NetSendData()函数把命令放入一个包中。在本例中，首字节的第 8 位和第 7 位被置位，因为仅有一个数据包。第 1 位也被置位，因为这是一个命令包。NetSendData()使用 COMSendTxBuffer()来构造加入报文信息的数据包并发送到总线上（TxKernel()）。在这种情况下，首字节 0x0——包类型（0x01=命令）和包 ID(=0x01)被添加到包的开头。计算校验和并且在所有为 0 的字节（0x0）处用两个 0（0x0 0x0）替代。（包括 LRC 中的 0x0）

然后数据链路层将等待来自另一端的响应。在本例中将收到一个报文 ID 为 0x01 的正确应答（0x06）ACKN(0x02)。下一个从 PLC 收到的报文包含了命令的应答数据。此应答数据在 SksEvaluateAppendData 中被分析。在第一次指示 PLC 状态变化标志时用 5 个字节表示。

3.2.5.3 OpenPCS V24 通信时序

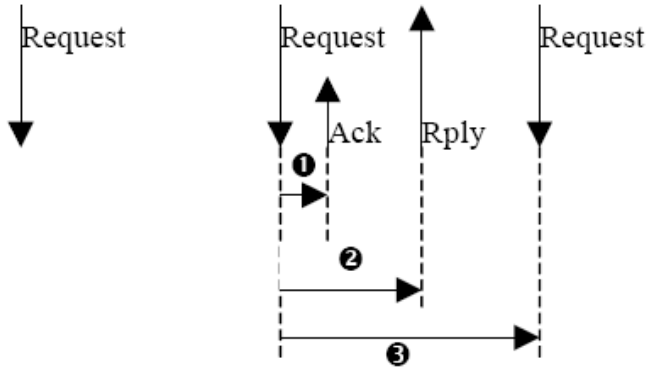


图 4 OpenPCS V24 时序

当 OpenPCS 通信时，有下列时序结构

	说明	取值	定义
①	每次请求（从 PC 发送到控制器）在 300ms 内需要第二层的应答	300ms	REC_TO
②	每次向下发送到控制器的请求都期望获得来自第七层的应答	5000ms	NET_RxTIMEOUT NET_TxTIMEOUT NET_RECTIMEOUT
③	就算没有发送数据到控制器的需求，TUI 也将向控制器发送周期请求。	500ms	REQTIME

3.2.6 通过 TCP/IP 协议通信

如果你希望通过 TCP\IP 建立 PLC 与编程系统的连接，可参考如下示例。其各自的工程放于如下目录：

infoteam\V400\smart.plc\samples\win32ipc (NetTCP 处理)

或者

infoteam\V400\smart.plc\samples\linux

对于实时系统和编程系统

\\infoteam\V400\OpenPCS\PadtTCPDriver32 (TCPDrv432.dll).

如果你注册了 TCPDrv.dll(例：regsvr32.exe),可在在线安装工具中（onlsetup.exe）看到选择驱动器对话框。为了建立一个新的 TCP\IP 连接，需要在设置对话框输入 IP 地址和目标控制器接口。对于实时系统，其接口定义在 nltcp.c 中（默认为 23042）

下列所举例子均使用 Microsoft Winsock 库（wsock32.dll）

3.2.6.1 TCP/IP 通信超时

在编程系统方面有两种情形的 TCP 连接超时。一个是在线服务已建立 5000ms 且没有变化。另一个是查找驱动程序 DLL 超时（查找嵌套字选择设置）。

在实时系统中的 TPC 超时取决于嵌套字实现。例如：在 WinCE 中嵌套字选择设置是有效的。

3.2.6.2 OpenPCS TCP/IP 通信的一般配置

3.2.6.2.1 实时系统

SmartPLC 实时系统通过一个特殊接口实现 TCP 服务端侦听。此接口在 nltcp.c 中被定义且可根据你的需要修改。在启动时（网络初始化）将建立一条线路，约束嵌套字和调用侦听。如果客户端，比如 OpenPCS 编程系统，与实时系统连接时函数 accept 将返回且线路退出。这意味着在 NetInitIalize 被重新调用前，不再有其他的客户端可与此实时系统建立连接。（单客户端服务）

实时系统查询模式中，函数 NetRecData 和 NetGetRxStatus 用于侦测客户端是否在发送数据。如果没有数据发送则函数 NetGetRxStatus 不会阻塞。可以几种方式来完成

通过嵌套字选择设置超时（见例 win32ipc）

由选择（select）工具查询嵌套字设置

通过 recv 块读取线路

注：像 recv 之类的嵌套字函数在没有读取所有预期数据时仍会返回。因此已读取的数据将被拷贝到缓冲区且函数返回值为 NET_NODATA。这导致实时系统必须再次调用此函数直到所有数据都已读取。发送也将被反复调用直到发送完成。

在连接中断或出现错误时，网络层将被重新初始化（NetShutDown 和 NetInitialize 被调用）。注意在一些系统（如 linux 样例）中，如果两个嵌套字（网络，通信）都被关闭将产生错误。在此类系统中，网络嵌套字被设置为常开状态。

3.2.6.2.2 编程系统

OpenPCS 编程系统的 TCP\IP 在线服务器需要知道目标控制器系统使用是的小端还是大端数据发送格式。因此在驱动器设置的地方有个复选框可供设置。

3.2.6.2.3 编程系统 4.3.0 和 4.3.2 的区别

4.3.2 版本的编程系统的在线服务驱动程序需要目标控制器在发送数据时，在每包数据前加 5 个字节的包头：

长度（DWORD）	包类型（BYTE）

5.0.1 以后版本的编程系统提供了 2 个 TCP 驱动程序

TCPdrv.dll	4.3.2 版本以前的与目标控制器通信的驱动程序
TCPdrv432.dll	4.3.2 版本以后的与目标控制器通信的驱动程序

3.2.6.2.4 TCP 通信示例

TCP/IP : Thread socket createdTCP/IP : Thread socket boundTCP/IP : Thread socket listenTCP/IP : EXIT Thread
NetGetRxStatus:[01][02][01][01] <- LOGIN command

TCP/IP : NetSendData Size(8) TimeOut(5000)
<0d><00><00><00><03><00><05><00><01><3b><20><04><00>-

TCP/IP : NetRecData Size(128) TimeOut(0)
NetGetRxStatus: [08][00] <- GETRESOURCEVERSION command

TCP/IP : NetSendData Size(80) TimeOut(5000)<55><00><00><00><03><00><4d><00><07><49><45><43><36><31><31>
<33><31><00><00><00><00><00><00><00><00><00><00><00><00><00><00><00><00><00><00>
<00><00><00><00><00><00><00><00><47><41><54><45><57><41><59><31><00><00>
<00><00><00><00><00><00><00><00><00><00><00><00><00><00><00><00><00><00><00><00>
<00><00><00><00><a7><20><8a><ae><91><ab><ea><40><b2><ab><ea><40>-

TCP/IP : NetRecData Size(128) TimeOut(0)
NetGetRxStatus:[09][00] <-GETPLCVERSION command

TCP/IP : NetSendData Size(76) TimeOut(5000)<51><00><00><00><03><00><49><00><08><00><02><00><03><04><2a>
<02><02><53><6d><61><72><74><50><4c><43><00><00><00><00><00><00><00><00><00>
<00><00><00><00><00><00><00><00><00><00><00><00><00><00><00><00><00><00><00><00>
<00><00><00><00><00><00><00><00><00><00><00><00><00><00><00><00><00><00><00><00>
<00><00><00><00><00><00><00><00><00><00><00><15><02>-

TCP/IP : NetRecData Size(128) TimeOut(0)
NetGetRxStatus:[0a][00] <-DISCARDWATCH command

-> LZS: LzsWatchDiscardWatchTab
TCP/IP : NetSendData Size(3) TimeOut(5000)<08><00><00><00><03><00><00><00>-

TCP/IP : NetRecData Size(128) TimeOut(0)
NetGetRxStatus: [0c][09][03][01][00][0e][00][4a][00][02][00]<- ADDWATCH command

-> LZS: LzsWatchAddInstruction... RetCode=00 [WatchID=0000]
TCP/IP : NetSendData Size(6) TimeOut(5000)<0b><00><00><00><03><00><03><00><05>
<00><00>-
TCP/IP : NetRecData Size(128) TimeOut(0)
NetGetRxStatus:[0c][09][03][02][00][0e][00][46][00][02][00]<- ADDWATCH command

-> LZS: LzsWatchAddInstruction... RetCode=00 [WatchID=0001]
TCP/IP : NetSendData Size(6) TimeOut(5000)<0b><00><00><00><03><00><03><00><05>
<01><00>-

TCP/IP : NetRecData Size(128) TimeOut(0)
NetGetRxStatus: [0b][01][01] <- ENABLEWATCH command
-> LZS: LzsWatchEnableWatching (All)

TCP/IP : NetSendData Size(3) TimeOut(5000)<08><00><00><00><03><00><00><00>-
.....
TCP/IP : NetRecData Size(128) TimeOut(0)
NetGetRxStatus:[02][00] <-LOGOUT command

TCP/IP : NetSendData Size(8) TimeOut(5000)<0d><00><00><00><03><00><05><00><01>
<01><00><00><00>-
TCP/IP : NetShutDown
TCP/IP : NetShutDown : NetSocket_g closed
TCP/IP : NetShutDown : ComSocket_g closed
TCP/IP : NetInitialize
TCP/IP : Started V1.1 HV2.2
(WinSock 2.0)(Running) MaxSock(32767)
TCP/IP : NetRecData Size(128) TimeOut(0)TCP/IP : Thread socket createdTCP/IP : T
hread socket boundTCP/IP : Thread socket listen

3.2.7 创建在线驱动程序

通过如下步骤可创建在线驱动程序：

- 拷贝一个已存在的驱动程序，并更改方案目录的名字和输出文件的名字（工程属性;连接器;所有配置）
- 把驱动程序的类 ID 由…0400-x…改为<OEM-ID>-x…
  - 所有出现在 PadtOnlineDriver32.idl 文件中的
  - 所有出现在 PadtOnlineDriver.rgs 文件中的
  - 所有出现在 PadtOnlineDriverSetup.rgs 文件中的
- 更改驱动程序名字和说明文档
  - 在 PadtOnlineDriverSetupImpl.cpp 文件中
- 更改驱动程序的”ProgID”，”CurVer”和”VersionIndependentProgID”
  - 所有出现在 PadtOnlineDriver.rgs 文件中的
  - 所有出现在 PadtOnlineDriverSetup.rgs 文件中的
- 完成 IPadtOnlineDriver 和 IPadtOnlineDriverSetup 中的所有接口函数——见 IDL 文件

通过如下步骤安装此新在线驱动：

- 将 DLL 文件拷贝到 OpenPCS 编程系统文件目录中
- 在 OpenPCS 程序文件目录下，通过命令提示符调用”regsvr32<file name>.dll”来注册此 DLL 文件
- 现在，在 OpenPCS 中可见此驱动程序了，在主菜单中选择 PLC—>Connections，点击“NEW”然后“Select”

3.2.8 调试在线驱动程序

如果你已经写好了 OpenPCS 的在线驱动程序，可通过下列选项调试此驱动程序：

- 作为可执行文件调用此驱动程序，指定 ONLSVR32.EXE 带”-Embedding”参数的 ONLSVR32.EXE 文件。
- 通过 OpenPCS 调用驱动程序。

3.2.9 暂停嵌入式实时系统通信

如果嵌入式实时系统在一个相当长的时间内没有通信，则可以发送一个锁时通知到编程系统。当 CmdMainLoop 执行一个更长的任务时这样作是很有用的，例如，当持续对一个慢 flash 驱动进行写操作的时候。

为了在规定的时间内暂停通信，可调用下列函数

LZSBOOL LzsCsvSendLockTime(LZSBYTE bConnectionId_p, LZSDWORD dwLockTime_p)

见例 Win32ipc(文件 lzsenv.c,函数 LzsEnvRegisterSysState)

3.3 OpenPCS 初始化文件配置

3.3.1 硬件配置文件

所有的硬件配置文件存放在 C:\Documents and Settings\All Users\Application Data\infoteam Software\OpenPCS2008 目录下。MODULES 是 OPENPCS INI 目录下的一个子目录。如何将你的硬件配置文件加入到当前 OpenPCS 编程系统中并安装请参考 4.6 节：添加实用驱动程序。

3.3.1.1 硬件声明

所有的硬件声明都在文件 MODULES.INI 中：

[MODULES]

<HARDWARE_ID>=<FileName of Ini-File> (e.g. IEC_PLC=PLC.INI)

...

3.3.1.2 硬件配置文件

硬件配置文件包含了所有必要的描述硬件的值和所有的固件 POU（被下载的字节流）

3.3.1.2.1 硬件配置

所有的入口都位于 MODULE 部分。

注意：一此修改会影响 OpenPCS 许可证，请查阅 3.3.1.2.2 节。

Identifier	Description	Range	Default in smartsim.ini	Unit	OEM[1]
Name	硬件全名	<String>	""	Na	Y
OemDeviceName	硬件描述字符串。这个这字符串显示在资源的硬件部分。	<String>	infoteam SmartSIM		Y
PceDeviceName	像在glob-port.inc文件中一样的驱动程序名称和在Openpcs.520目录下的其它文件	<String>_<String>	IEC_IEC_NORM	Na	Y
Processor	处理器描述（不用）	<String>	Intel 80386		Y
PcodeDll32	本地代码编译器DLL	<String>	Ncc86_32.dll		Y
FractLayout	特殊内存布局	0..1	0		N
HConfigDll32	硬件特定编译器DLL	<String>	Smartplc.dll		Y
SmartOPC	使用SmartPLC/OPC	0..1	0		N
CANOpen	使用DCF文件分析程序	0..1	0		Y
DCFParser	DCF分析程序DLL	<String>			Y
Mem	弃用		-		
PlcMemSize	PLC内存容量（见注1）	0..0xFFFFFFFF	33554432 (32MB)	Byte	Y
Pgm	弃用		-		
Seg	弃用		-		
Cyclic	支持循环任务	0: FALSE 1:TRUE	FALSE		Y
Timer	支持时钟任务	0: FALSE 1: TRUE	FALSE		Y
Periodical	支持周期任务	0: FALSE 1: TRUE	FALSE		N
Interrupt	支持中断任务	0: FALSE 1: TRUE	FALSE		Y
IpVariante	解释程序的转化（obj. code）	0..255	4		Y
IpRevision	解释程序的修订版本	0..255	0		N
IpUmfang	解释程序函数	0..255	2		N
IpOEM	OEM编号	0..65535	42		N (*)
HwVersion	硬件版本	0..255	2		Y
HwRevision	硬件修订版本	0.255	0		Y
FwVersion	实时系统版本	0.255	3		Y
FwRevision	实时系统修订版本	0..255	0		Y
InBytes	输入字节	0..65535	0	Byte	Y
OffsetIn	输入字节起始偏移	6.. 65535	6	Byte	Y
OutBytes	输出字节	0.. 65535	0	Byte	Y
OffsetOut	输出字节起始偏移	6.. 65535	6	Byte	Y
MarkerBytes	标志字节	0.. 65535	0	Byte	Y
OffsetMerker	标志字节起始偏移	6..30000	6	Byte	Y
NetOffsetIn	网络输入起始偏移	6..30000	6	Byte	Y
NetOffsetOut	网络输出起始偏移	6..30000	6	Byte	Y
NetImageSize	网络映象大小	0..30000	0	Byte	N
ByteSize	字节大小	1, 2, 4	1		Y

Identifier	Description	Range	Default in smartsim.ini	Unit	OEM[1]
Motorola	Motorola 或 Intel格式	1: Motorola 0:Intel	0		Y
RetainVariables	支持保持变量	0: not allowed 1: allowed 2: all variables have to be retain	1		N
RetainDirectVa-riables	支持保持直接变量	0: not allowed 1: allowed 2: all variables have to be retain	1		N
InitVariables	变量初始化	0: not allowed 1: allowed 2: all variables have to be retain	1		N
InitDirectVariables	直接变量初始化	0: not allowed 1: allowed 2: all variables have to be retain	1		N
Alignment	变量对齐（见注2）	1, 2 , 4 , 8 , 16	0	Byte	Y
SpecAlignment	字节的指定对齐	0: FALSE 1: TRUE	0		N
AlignStructHeader	结构体头对齐（见注3）	0:FALSE 1:TRUE	0		N
Functions	支持用户函数	0: FALSE1: TRUE	0		N
Secure	安全系统	0: FALSE 1: TRUE	0		N
BitMaskInOffset	段的带偏移的位屏蔽	0: segment 1: offset	1		N
BitAllowed	允许二进制	0: FALSE 1: TRUE	1		N
BcdAllowed	允许BCD码	0: FALSE 1: TRUE	0		N
EnumAllowed	允许枚举	0: FALSE 1: TRUE	0		N
ArrayAllowed	允许数组	0: FALSE 1: TRUE	0		N
StructAllowed	允许结构体	0: FALSE 1: TRUE	0		N
ReferenzAllowed	允许引用（VAR_EXTERNAL,VAR_INOUT）	0: FALSE 1: TRUE	0		N
ArrayOfArra-yAllowed	允许数组的数组（二维数组）	0: FALSE 1: TRUE	0		N

Identifier	Description	Range	Default in smartsim.ini	Unit	OEM[1]
ArrayOfStruc-tAllowed	允许结构体数组	0: FALSE 1: TRUE	0		N
StructOfStruc-tAllowed	允许结构体的结构体	0: FALSE 1: TRUE	0		N
StructOfArra-yAllo_wed	允许数组的结构体	0: FALSE 1: TRUE	0		N
SegTblResGlob	段表是资源或全局任务	1:resourceglobal 0: task global	0		N
UseSegTbl	使用段表（1）或都写入文件（0）		1		N
OnlineLinking	连接在PLC上完成(或已经连接的段表被下载到PLC)	0: FALSE 1:TRUE	0		Y
GenIpCode	生成解释器代码	0: FALSE 1:TRUE	1		N
ResGlobVariables	支持资源全局变量	0: FALSE 1:TRUE	0		N
OneSegTbl	在联机连接时使用一个段表	0: FALSE 1:TRUE	1		N
DebugDownload	测试和下载的调试特性	0: FALSE 1:TRUE	0		N
Incremen-talCompile	支持增量编译	0: FALSE 1:TRUE	1		N
AddFiles2CrdDataCont	所有文件被加入到CRD中的数据窗口内(大于64K将不成功)	0: FALSE 1:TRUE	0		N
OnlEditor	在TUI模式联机编辑器可被启动（功率流可用）	0: FALSE 1:TRUE	0		N
IncrementalTask-Download	任务可分别下载	0: FALSE 1:TRUE	0		N
TaskSeperatly-Startable	任务可分析启动（代替整个资源）	0: FALSE 1:TRUE	0		N
WatchEntries	有多少可用的观察记录	0..255	16		N
ComponentList	如果你使用一个特殊的P-Code Generator（P代码生成器），那么段表将被改变，所以我们选择第二个段表。	TRUE:普通编译器运行将它的段写入组件列表，P代码生成器写入段表 FALSE: 只写入段表	0		N
GenerateIoEx-change	是否可以生成命令IP_EX_EXCHANGE	TRUE: yes	FALSE		N
NoBuildDate	抑制在任务定义段中创建日期的补丁	TRUE: No build date is patched	FALSE		N
SubRangeAllowed	允许子域	TRUE: yes	FALSE		N

Identifier	Description	Range	Default in smartsim.ini	Unit	OEM[1]
FloatSequence	用于表示浮点数的格式，见3.1.11	0 = Microsoft 1 = Tasking C167 2 = TI TMS320	0		Y
ReservedSegments	通过目标实时系统定义内部使用的保留段的数量	10	10	segments	N
ConfOPC	在安装中禁止OPC	0 (disabled) 1 (enabled)	-		Y
PascalCallConvention	允许增强的字符串函数	0 (disabled) 1 (enabled)	-		Y
EnableAccuAnyOptimize	最优化增强的字符串函数	0 (disabled) 1 (enabled)	-		Y
MultiVarTab	支持多种VarTab段	0 (disabled) 1 (enabled)	-		
SmartVarTab	联机编辑的配置：在数据段中的变量在联机改变时将保持改变的值	0 (disabled) 1 (enabled)	1		
AddressAllowed	是否被编译器使用	0 (disabled) 1 (enabled)	-		
SuppressAlignmentCheck	是否允许对直接变量进行对齐检查	0(checkenabled) 1(checkdisabled)	-		
ArrayCheckRangeLimits	实行制约检查数组范围。默认为允许（尽管没有给出任何设置）	0(checkdisabled) 1(checkenabled)	1		Y

[1]Y=可被 OEM 修改，N=不可被 OEM 修改

注：

1. PlcMemSize 的用途是允许针对 PLC 的可用存储空间，检查 IEC 程序的存储需求。如果存储空间需求超标，在 OpenPCS 编译结束后会产生一个警告，这个检查可在下列选项中开启：Extras->Options->Browser->Compiler（OpenPCS 6.4.0 或更高版本）

PlcMemSize 是可选择的，如果此功能没有被选中，则不能实现上述功能。

2. 对齐指定了变量存储的界限。如果有一个 N 字节的对齐，那么每个大于或等 N 字节的变量都将以 N 字节来存储。例如，一个 2 字节的对齐意味着每个大于 1 的变量被存储于一个 2 字节界限内。1 个字节的变量存储于 1 字节界限等。他们可以存储在任何的地址。一个 4 字节的对齐意味着每个大于 3 字节的变量存储于 4 字节界限内，2 字节变量存储于 2 字节界限内，1 字节变量存储于 1 字节界限内（见 3.1.14）

3. 当 AlignStructHeader 被设置为 1 并与 4 字节对齐结合时，结构体头的大小通常为 4，包含满的 2 个字节在其中，以至于第一个结构元素开始于一个 4 字节对齐的地址。

（*）IpOEM 可通过 Infoteam 申请并且必需是唯一的。如果发生问题请检查你的许可证。

3.3.1.2.2 硬件配置文件和许可证（licences）

一个 OpenPCS 许可证绑定一个硬件初始化文件，因此如果下列项目中的一个被改变，许可证将会无效。

Name	OemDeviceName	PceDeviceName
Processor	PCodeDll32	FractLayout
HConfigDll32	SmartOPC	CANOpen
DCFParser	Motorola	Alignment
ByteSize	CDA	

在3.3.1.2.1节中有对以上项目的解释

3.3.1.2.3 hardware.ini示例

一个hardware.ini的例子就是”SmartSim.ini”，安装于OpenPCS编程系统。与硬件初始化文件的固件函数块部分模板一起，你可以调整来适应你的硬件（队列、字节顺序、大小）。固件函数块的初始化模块位于infoteam\v5xx\Tools\initemplates

3.3.1.2.4 DefinedTaskTypes

TYPE1	1. defined task type	<String>	Cyclic		N
TYPE2	2. defined task type	<String>	Timer		N
TYPE3	3. defined task type	<String>	Interrupt		N
DefaultTimerValue	default timer task interval	0..4294967295	1	ms	Y

3.3.1.2.5 DefinedInterrupts

TYPE1	1. defined interrupt name	<String>	STARTUP		N
TYPE2	2. defined interrupt name	<String>	STOP		N
TYPE3	3. defined interrupt name	<String>	ERROR		N

详见 5.4.3.4 节中断任务。

3.3.1.2.6 VARTAB

见 5.3.7.7.2 节

3.3.1.2.7 Prebuildsteps 和 Postbuildsteps

见 5.9.6 节

3.3.1.2.8 固件配置

所有的固件 POU 都在 FIRMWARE 中配置

[FIRMWARE]

CTUD=CTUD

R_TRIG=R_TRIG

每个固件 POU 都存在一个在固件表中可查寻的以输入命名的单独的部分。

Identifier	Description	Range	Default
Instantiate	这个块是否被实例化（必须为1）	0 or 1	1
CRStack	只被解释器执行固件所需要		1
RetStack	只被解释器执行固件所需要		1
Firmware	是否为固件地（必须为1）	0 or 1	1
Index	块编号	0.. 8191	0
Group	块所属的组	0 and 1 = infoteam 2 and 3 = OEM	2
DataBytes	后面的数据字节数	0..32767	0
DATA	16进制的字节数据列表。这些数据必须描述初始数据段	e.g. 00 01 a0 CD CD CD	--

3.3.1.2.9 下载文件到 PLC

从 v5.2.2 版本以后，连接所有类型的文件到工程中并且下载下 PLC 成为了可能。为了打开此功能，必须在你的硬件初始化文件中通过分项”ACTIVE=”和” EXT =”设置[ENABLE_RAWFILES]入口。”ACTIVE=”用于开启下载原始文件，”EXT=”包含了允许下载的文件扩展名，用分号隔开。

例：对于压缩文件，如下所示

[ENABLE_RAWFILES]

ACTIVE=1

EXT=zip;rar;arc;ace;arj;tar;cab

为了把文件连接到工程中，在浏览器中文件必须是可访问的。可通过AddDriver增加可访问文件的扩展名。因此，在你的CAB文件中的”hw.ini”必须包含[VISIBLE_EXTENSIONS]部分和”EXT=”分项。

例：对于压缩文件：

[VISIBLE_EXTENSIONS]

EXT= zip;rar;arc;ace;arj;tar;cab

_LZS_DWL_RAWFILE_须在config.h中定义，用于开启此功能并且在lzsenv.c中的函数原型需要被执行。见示例Win32ipc。

3.3.1.2.10 参数表达式

OpenPCS不支持在缺省调用函数块时使用参数表达式。允许参数表达式是可以的，但是不建议这样做。为了开启此功能，在你的PCL初始化文件中必须有如下入口：

[Module]

...

AllowExpressionOnParameter=1

如此就允许如下语句：

MyFBInstance(input_param := x * y);

必须注意，这样可能会造成错误。如果一个FB以如下方式被调用：

MyFBInstance(input_param := x * y + z);

Input_param不会以正确的方式计算，这将造成错误的编程。

我们强烈建议不要使用此功能。可以如下方式（或相似的）来替代：

MyFBInstance.input_param = x * y + z;

MyFBInstance();

这样input_param将被正确计算并且不会依赖任何编译器的特殊功能。

3.3.1.2.11 嵌套函数调用

从6.0版本以上，OpenPCS不支持在缺省调用函数块时使用一个对函数的调用作为其参数。可以开启此功能，但是不建议这样做。

为了开启此功能，在PLC初始化文件中加入如下入口：

[Module]

...

AllowNestedFunctionCalls=1

如此将允许如下语句：

i := func1(x, func2(func3(y), z)) ;

注意，这可能会导致错误的结果。

我们强烈建议不使用此功能，作为替代，可使用临时变量存放中间结果。

例：

temp := func3(y) ;

temp := func2(temp, z) ;

i := func1(x, temp) ;

3.3.1.2.12 改进的字符串函数

OpenPCSv5.4.0以上版本提供了有3个参数的改进的字符串函数。为了开启此功能，在文件config.h中须定义_PASCAL_CALLING_CONVENTION_和_IP_ENABLE_ACCU_ANY_OPTIMZE_，并且通过硬件初始化文件中的PascalCallingConvention和EnableAccuAnyOptimize使能。

3.3.2 Globprot.inc/Globtype.inc

这些文件描述了所有的全局原型，POU 和用户定义类型。适应于所有的 OpenPCS 工程。

3.3.2.1 Globprot.inc

在这个文件中，列举了所有的固件 POU 原型和用特殊 U 代码命令表示的标准 IEC 函数。每个设备都有一个以设备名称命名的特殊段（例：PLC_LARCE）。

每个原型都被括于 GLOBAL_PROTOTYP_BEGIN 和 GLOBAL_PROTOTYP_END。

所有的标准函数都被括于$PREDEFINED_FUNCTIONS 和$END_PREDEFINED_FUNCTIONS。

```
[PLC_LARGE]
$PREDEFINED_FUNCTIONS
        shl
        shr
        rol
        ror
$END_PREDEFINED_FUNCTIONS

GLOBAL_PROTOTYP_BEGIN
  FUNCTION_BLOCK SR
      VAR_INPUT
        SET1 :BOOL;
        RESET :BOOL;
      END_VAR
      VAR_OUTPUT
        Q1 :BOOL;
      END_VAR
  END_FUNCTION_BLOCK
GLOBAL_PROTOTYP_END
…
```

3.3.2.2 globwrap.inc

GlobWrap.inc 与 globprot.int 相似，但是可以定义所谓“封装函数块”，通常通过调用 IEC1131 函数和 USLadder 编译器内的指令来实现。Globwrap.inc 中的模块能以 USLadder 语言定义，不能用其他语言。

使用封闭函数块的原因是向用户提供适应其他语言的相似函数，但是须 EN/ENO 支持，且同样适用于文本语言。

3.3.2.3 globtype.inc

本文件表示了所有的全局类型。文件结构与 globprot.inc 相同。每个类型都被括于 GLOBAL_TYPE_BEGIN 和 GLOBAL_TYPE_END。

下列数据类型仅做示例且在不需要的时候会被移除：

DAY_OF_WEEK

ELEMENT_OF_DAY

DIAGNOSTIC_TYPE

MESSAGE_TYPE

SWITCH_POSITION

3.3.2.4 OEM 特定文件 glob*.inc

为了加入你自己的特定硬件原型和类型到 OpenPCS 编程系统中，你需要从 OpenPCS 初始文件目录下系统子目录中将原型文件 glob*.inc 拷贝到 MODULES 下的子目录中，并以你的硬件配置文件的名字命名（without.ini）。

3.3.2.5 CatalogCategoriex.xml

文件 CatalogCategories.xml 用于定义各种类型的函数块。文件的结构如下所示。下图举例说明了在一个运行中的 OpenPCS 系统的 CatalogCategoried.xml.

```
<catalog>
        <category value="Timer">
            <pou name="TON">
            </pou>
            <pou name="TOF">
            </pou>
        </category>
        <category value="Trigger">
            <pou name="RS" />
            <pou name="SR" />
        </category>
        <category value="Timer Trigger">
            <pou name="TON" />
        </category>
</catalog>
```

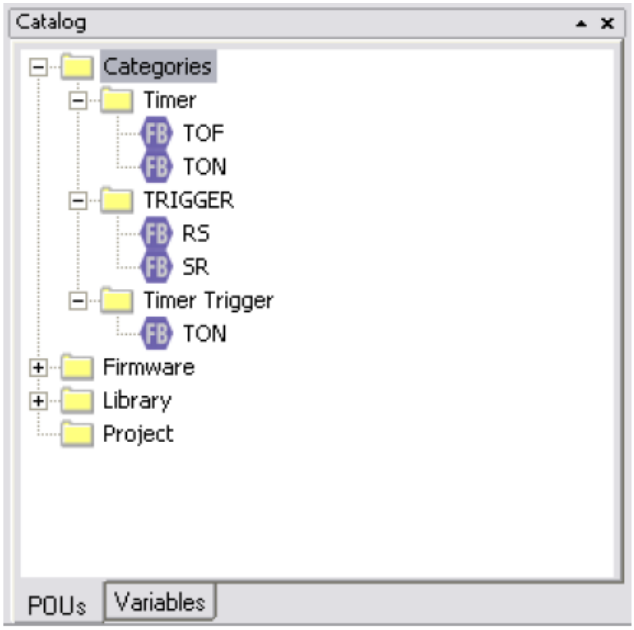


图 5 函数块分类

根元素叫做<catalog>。不同的类别通过<category value="catName">定义，catName 是将要定义类别的名字。函数块通过<pou name="TON">加入到一个类别中。别忘了关闭此标签，通过</pou>或一个在反括号前的斜线均可。

函数块可以附属在几个类别中。不用理会不存在的 POU 入口。

CatalogCategoriex.xml 必须放于 OPENPCS 初始目录下，可以是 C:\Documents and Settings\All Users\Application Data\infoteam Software\OpenPCS2008\Openpcs.520.

3.3.3 Custtool.ini

Custtool.ini 通过浏览器中的工具菜单决定哪一个程序可以开始运行。在这个初始化文件中只有一个部分：[CUST-TOOLS]

```
[CUST-TOOLS]
File0=KISPOEED.EXE
Title0=EDITOR
Arguments0=
CmdShow0=6
InitDir0=
QueryArguments0=0
File1=...
...
```

OpenPCS 支持一些特殊宏指令术语的建立：

macro	term
$ProjPath	到当前.Var文件的路径
$ProjName	不带目录的当前.Var文件的名称
$ProjDir	到当前不带名称的.Var文件的目录
$FilePath	到浏览器中当前选择的文件的路径
$FileName	上述文件的名称
$FileDir	上述文件的目录
$ConfigDir	OpenPCS配置目录（如：C:\Documents and Settings\All Users\Application Data\infoteam Software\OpenPCS2008\Openpcs.520\）

$OemId	OEM-ID
$ActiveMakePath	包含生成文件名称的有效的资源的生成文件的实际路径
$BinDir	OpenPCS安装目录（二进制文件）
$AppDataDir	OpenPCS应用程序数据目录

如果你想要加入一个工具，例如编辑器，你需要附加此工具的入口。

例：

```
...
File6=MYEDITOR.EXE
Title6=MY_EDITOR
Arguments6=<Command line arguments>
CmdShow6=<Determines how the window is shown, see windows.h>
InitDir6=<Initial directory>
QueryArguments6=<if this value is unequal zero, the user can enter command line arguments through a dialog>
```

自定义工具最多可设置20个。

3.3.4 配置错误日志

编程系统提供了一个接口通过在线服务器从 PLC 上传错误日志段。错误日志存于以 errorlog.bin 命名的二进制文件中，此文件位于当前工程的GEN文件夹中。

在成功上传此段后，LzsErrorMap.exe 提取相应的信息并附上时间存于工程根目录下名叫 yymmdd_hhmmssErrorlog.txt 的文件中。

ErrorMap.ini 文件配置了错误日志的错误分组到 XML 定义的映射，定义了信息的产生。

例：

```
[ErrorResources]
count=2
Group0=LZSErr
LZSErr =LZSErr.xml
Group1=CanErr
CanErr =oemhardware\\CanError.xml
...
```

前 128 号错误组为 infoteam 特定错误保留，128 号以后的错误组可被 OEM 用户使用。

为了加入你自己的特定硬件错误，你需要将 ErrorMap.ini 原始文件从 OpenPCS 初始化目录下的系统子目录拷贝到以你的硬件配置文件命名的 MODULES 下的子目录中（without.ini）。XML 文件如下所示：

```
<?xml version="1.0" encoding="utf-8" ?>
<ErrorDefinitions xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="ErrorMaps.xsd">
    <ErrorGroup id="129">
        <Error id="1">
            <Message>This is an example error with two parameters: {0:x},{1:d}
            </Message>
        </Error>
        <Error id="2">
            <Message>This is another error with one parameter: {0}
            </Message>
        </Error>
        </ErrorGroup>
</ErrorDefinitions>
```

（XML 文件的布局通过”ErrorMaps.xsd”构架的定义来使之生效，此文件位于系统子目录下）

作为格式化错误报告的分类符，下列分类符均可使用（与函数字符串格式化 C#指令一致）：

Index

一个零基整数，指出格式化工程列表中的哪一个元素

Alignment

一个随机整数，指出包含格式值的最小宽度范围。如果格式值的长度小于对齐值，则剩余范围用空白填充；如果对齐值是负的，格式值将被左移；如果对齐值是正的，格式值将被右移。如果对齐值不是确定的，那么这个长度范围就是格式值的长度范围。逗号用于对齐值是确定的情况。

formatString

一个格式代码类型（见下例）的随机字符串。冒号用于格式字符串是确定的情况。

Type	Format specifier	Example (value 1000)
Decimal	{0:d}	1000
Decimal with separator	{0:n}	1.000
Hexadecimal	{0:x}	3E8

3.4 识别实时系统版本

编程系统 3.4.2 以上版本 Infoteam OpenPCS 编程系统和购买全部测试&调试工具的源目标系统将把实时系统的版本作为下载资源的版本。实时系统的版本有如下结构：

description	define in source code	entry in SMARTSIM.ini	defaultvalue
version of runtime kernel	IP_DOMAIN	IpVariante	4
revision of runtime kernel	IP_REVISION	IpRevision	1
version of hardware	HARDWARE_VERSION	HwVersion	2
revision of hardware	HARDWARE_REVISION	HwRevision	0
description of hardware	HARDWARE_DESCRIPTION		"SmartPLC"
version of firmware	FW_VERSION	FwVersion	3
revision of firmware	FW_REVISION	FwRevision	0
ID of OEM	OEM_ID_NR	IpOem	42

测试&调试工具把这3个版本号（IP_DOMAIN,HARDWARE_VERSION和FW_VERSION）和OEM ID作为硬件配置文件相应的记录并且如果在PLC和资源间存在差异将拒绝下载。不同的版本号不妨碍下载。
程序代码发生器使用硬件初始化文件中（见3.3.1）的版本号来编译资源。

3.4.1 扩展版本信息

3.4.1.1 硬件初始化文件与编译器

在硬件初始化文件中有 4 种新的记录可供使用。这些值包含了 OEM 版本号 Major.Minor.Revision.Build

OemVersMajor=<major>

OemVersMinor=<minor>

OemVersRevision=<revision>

OemVersBuild=<build>

这些值在任务定义段中通过连接器偏移量被定义

```
#define STRUCT_OFFSET_OEMVER_MAJOR 116
#define STRUCT_OFFSET_OEMVER_MINOR 118
#define STRUCT_OFFSET_OEMVER_REVISION 120
#define STRUCT_OFFSET_OEMVER_BUILD 122
```

3.4.1.2 实时系统

新的 tOemVersion 结构体将这些新的值封装进实时系统内部：

```
typedef struct {
    LZSWORD wOemVerMajor;
    LZSWORD wOemVerMinor;
    LZSWORD wOemVerRevision;
    LZSWORD wOemVerBuild;
}tOemVersion;
```

新的函数

```
LZSPUBLIC32 LZSBOOL LzsGetOemVersion(tOemVersion * pOemVersion)
```

可用于填充任务定义段的值到结构体中。结构体的存储空间由用户提供。为了编译 OEM 版本支持的实时系统，需做下列定义（例：在文件 config.h 中）：

```
#define _LZS_OEMVERSINFO_
```

3.4.1.3 抽样检查版本号

可通过 LzsEnvCheckConfiguration 来完成对版本号的检查。这个函数在下载和浏览映射文件时被调用。

```
LZSBYTE LzsEnvCheckConfiguration(tPlcMemPtr pConfigSegment_p) {
    #ifdef _LZS_OEMVERSINFO_ {
        tOemVersion OEMVersionInfo;
        LzsGetOemVersion(&OEMVersionInfo);
        if (OEMVersionInfo.wOemVerMajor != 0xAAAA) {
            return kLzsOemError01;
        }
    }
#endif
return kLzsSuccess;
}
```

返回代码 kLzsOemError01 是 OEM 可定义的错误返回代码之一。在 LzsEnvGetOemLzsError 中 OEM 可以创建错误字符串且返回给编程系统。

例：

```
tLzsErrTabEntry LZSCONST* LzsEnvGetOemLzsError (LZSBYTE bLzsErrorCode_p)
{
    tLzsErrorCode ErrCode;
    ErrCode = (tLzsErrorCode)bLzsErrorCode_p;
    switch (ErrCode)
    {
        case kLzsOemError01:
            {
                tOemVersion myOemVersion;
                LzsGetOemVersion(&myOemVersion);
                sprintf(pMsg,"Wrong OEM version : %d.%d.%d.%d",
myOemVersion.wOemVerMajor,
myOemVersion.wOemVerMinor,
myOemVersion.wOemVerRevision,
myOemVersion.wOemVerBuild);
                return &OEMErrTab_l[0];
            }
            break;
```

3.5 固件函数块

编译器生成变量的偏移量并以下列命令与接口连接。

取决于硬件对齐是 1、2 或 4 字节和字节顺序（小端或大端）

Length	Type
1	BOOL , BYTE , SINT , USINT
2	WORD , INT , UINT
3	边缘变量BOOL R_EDGE, BOOL F_EDGE(ctu, ctud, ctd的边缘变量只有1个字节). 第一个字节：新的值；第二个字节：为检测边缘的旧值；第三个字节：交换的帮助字节
4	TIME , DT , DATE, TIME , TOD, REALDWORD , UDINT, DINT
特殊	STRING, Array, Structure (见U-Code的说明)

Section	Description
Header of data segment 数据段头	6字节 (2字节长度, 1字节ID, 1字节段头长度,2 字节相应段)
EN_ENO	如果这个函数块使用了一个EN输入和/或一个ENO输出，那么为EN和ENO分配的空间将在数据段的最开头（当对齐小于4字节时，偏移为6+7，当对齐为4字节或更大时，偏移为8+9）
VAR_OUTPUT [A]	也是一个函数的返回值
VAR_INOUT [B]	对于var_inout变量需要额外的4个字节的引用。这个指向实际位置的引用被编译器在运行时补充。
VAR_INPUT [C]	
VAR_OUTPUT RETAIN [D]	
VAR_INOUT_RETAIN [E]	(目前不支持，见 var_inout)
VAR_INPUT_RETAIN [F]	
Section for handles [G]	调用固件POU和外部变量的句柄（目前不支持，大小为0字节）
VAR [H]	普通变量
VAR RETAIN [I]	普通保持变量

在初始化文件中初始数据段的字节数已被定义。如果对齐小于4并且不使用EN_ENO，它有如下结构。若是对齐为4且使用EN_ENO的情况请见下面的注释。

Offset	Size	Description
0	2	whole size of segment
2	1	id of segment -> 2 for initial data segment
3	1	length of header -> 16 bytes
4	2	corresponding segment (will be patched in by compiler)
6	2	size of section A+B+C
8	2	size of section D+E+F
10	2	size of section G
12	2	size of section H
14	2	size of section I
16	A+B+C	bytes of these sections (var_input, var_output, var_inout)
16+A+B+C	D+E+F	bytes of these retain section (var_input, var_output, var_inout)
16+A+B+C+D+E+F	H	bytes of the var-section
16+A+B+	I	bytes of the var-retain section

注释：

1）EN_ENO：使用EN_ENO，空间的分配需在数据头后。例如：在使用4对齐时分别偏移6+7和8+9.

2）4对齐：使用4对齐时，数据段开始于偏移量8，替代6（当使用EN_ENO时也可使用12）。此外，所有A到I的段都被4倍扩大。详见infoteam\vXXX\Tools\initemplates下的模板文件。

注：

在 OpenPCS3.3 到 OpenPCS3.4 的更新过程有改变。在 OpenPCS3.3 中数据段在初始文件中表示。现在可支持初始数据段保持变量的表示。

在初始数据段中没有 EN 或 ENO 的表示，因为这些变量通常在实时系统中初始化。

3.5.1 原型、数据段、初始数据段的结构

编译器用一个特殊的算法构造了一个数据段。如果使用固件 POU，接口变量（VAR_INPUT,VAR_OUTPUT, VAR_INOUT）需在相同的位置以便于 CBE 处理。因为一个 POU 可能有几个实例，所有的数据都将以初始数据段格式存储。实数数据段在实例化进程中被创建。

下表为初始数据段分布：

Summary	Offset	Description
normal Header	0	Length of whole segment
	2	Identifier for initial data segment
	3	Length of header = 16
	4	corresponding segment
Header with the description of the structure of the segment	6	length of input/output/inout section [A]
	8	length of retain input/output/inout section [B]
	10	length of instance handles and external variables (copied from Linker Table)
	12	length of other variables [c]
	14	length of other retain variables
initial bytes	16	initial bytes for input/output/inout [1]
	16+A	initial bytes for retain input/output/inout [2]
	16+A+B	initial bytes for all others non-retain variables [3]
	16+A+B+C	initial bytes for all other retain variables [4]

下列章节说明了子段[1]到[4]的格式。

3.5.1.1 子段 1、2 格式

子段 1、2 拥有相同的结构。他们定义了（固件）POU 的接口，且用于调用 POU。

原型示例：

```
FUNCTIONBLOCK pou_example

VAR_INPUT
        bool_in : BOOL ;
        byte_in : BYTE ;
        word_in : WORD ;
END_VAR

VAR_INPUT RETAIN
        time_retain : TIME ;
END_VAR

VAR_OUTPUT
        int_out : INT ;
        dint_out : INT ;
END_VAR

VAR_INOUT
        usint_inout : USINT ;
        uint_inout : UINT ;
END_VAR

VAR_INPUT :
        bool2_in : BOOL ;
END_VAR
END_FUNCTIONBLOCK
```

变量的偏移通过下列算法处理：

首先对所有的子段按 VAR_OUTPUT,VAR_INOUT,VAR_INPUT 的顺序排序。然后按原型的顺序为每个变量分配存储空间。

所以在上述示例中变量在数据段中有如下顺序：

```
int_out
dint_out
All VAR_INOUT:
usint_inout
uint_inout
All VAR_INPUT
bool_in
```

byte_in
word_int
bool2_in

变量 VAR_INPUT 和 VAR_OUTPUT 的位置是数据段本身。所以不同的内存必须为这些变量类型分配空间。变量必须有一个初值，否则可使用 IEC1131-3（表 13）规定的默认初值。

Memory	Variable Types
1 Byte	BOOL , BYTE , SINT , USINT
2 Bytes	WORD ,INT , UINT
4 Bytes	DWORD , DINT , UDINT , TIME , REAL
8 Bytes	LWORD , LINT , ULINT (not supported at the moment)
special length	structs , arrays , STRING

VAR_INOUT 变量是另一个数据段的变量。在解释程序中的引用长度是 4 字节。所以为每个 VAR_INOUT 变量分配 4 字节空间。OpenPCS 解释程序不支持对这些变量赋初值且 VAR_INOUT RETAIN 同样不可以。

对另一个子段[2]中的保持变量，用相同的算法。

3.5.1.2 函数体

函数体中只可使用 VAR_INPUT 变量和一个 VAR_OUTPUT——函数类型。所以首先申明函数的类型，然后申明所有的变量。申明格式如上所述。

3.5.1.3 局部变量（第 3、4 部分）

局部（保持）变量放在第 3、4 子段。定位哪个变量是由实现决定的。OPENPCS 编译器不用知道所有的情况。仅需要在原型部分使用相同的长度和格式即可。

3.5.1.4 外部变量

外部变量（VAR_EXTERNAL）不可以在固件 POU 中使用。

3.5.2 如何编写固件函数块

用 OpenPCS 完成一个固件函数快不是难事。首次设计时，按照下列步骤完成设计将容易得多

3.5.2.1 设计接口

首先，使用 IEC1131 标准语言设计你的固件函数块接口。包括函数块名称、输入的名称和类型、输出和输入—输出参数。

我们将用一个小例子阐明本步骤。我们设计的函数块仅可以用小时，分，秒的形式传送当前时间。使用一个输入开关量控制 12、24 小时制。

```
FUNCTION_BLOCK GetTime
VAR_INPUT
  TwelveHours: BOOL;
  END_VAR
  VAR_OUTPUT
        hours : SINT;
        minutes : SINT;
        seconds : SINT;
  END_VAR
```

图 6 固件块 GetTime

3.5.2.2 使系统知道此原型

为了方便 OpenPCS 知道你的新固件块，可将此函数块的“原型”放入 GLOBPROT.INC 文件中，此文件位于 Windows 的 Openpcs.520 目录下。此“原型”就是在上节作为接口来设计的部分。在 GLOBPROT.INC 文件中，用 GLOBAL_PROTOTYP_BEGIN 和 GLOBAL_PROTOTYP_END 将原型封装起来。（见 3.3.2.1）

与 C 语言比较，这种方式相当于添加函数的接口到头文件中，以便编译器知道怎样调用此函数。

在我们的例子中，以如下方式加入到 globprot.inc 文件中：

```
GLOBAL_PROTOTYP_BEGIN
      FUNCTION_BLOCK GetTime
      VAR_INPUT
            TwelveHours: BOOL;
      END_VAR
```

```
        VAR_OUTPUT
            hours : SINT;
            minutes : SINT;
            seconds : SINT;
        END_VAR
        END_FUNCTION_BLOCK
GLOBAL_PROTOTYP_END
```

图 7 globprot.inc 中的 GetTime

3.5.2.3 检查

打开 POU 编辑器，创建一个 POU，在构造窗口中，选择 Symbols->Manufacturer Defined function blocks.

如果在列表中没有发现你的函数块，点击”update”。如果仍然无法看见函数块，点击”troubleshooting”查看，选择你的函数块并输入一个有效的实例名，然后点击 OK。你可以在构造窗口中看见你的函数块的一个调用，然后你就可以识别出在上面定义的接口了。如果不行，可参考疑难解答。

疑难解答：

当运行 OpenPCS 的时候是不允许修改 OpenPCS 的初始化文件的。所以如果出现了错误，试着彻底关闭 OpenPCS（如果必要可重启 Windows 系统），然后再试一次。

检查在 POU 编辑器中选中的设备（设备菜单）。

检查你在 globprot.inc 适当的位置加入的原型。

检查你修改的 globprot.inc 文件部分。在你的 PC 上只能有一个文件以此命名。

仔细检查原型。用逗号代替分号，或在一条语句的结尾添加分号。

将此新程序连接到资源中并创建可执行代码。你会得一个错误报告，其内容为此函数块在当前目标上是不可用的。

3.5.2.4 连接

接下来我们需要使编译器确信这个新的并且以前不知道的函数块可以在硬件上运行。修改初始化文件以配置你的硬件。在默认情况下，OpenPCS 使用在你的 Windows 目录下的 Openpcs.520/modules 目录中的 SMARTSIM.INI 文件。如果你为你的硬件创建了一个不同的文件，则使用你创建的文件。在剩下的章节中，我们会讨论 SMARTSIM.INI 文件。

在文件 SMARTSIM.INI 中，存在一个字段[FIRMWARE]。像其他包含在清单中的语句一样，用一行语句给出你的函数块的名称。此名称必须和你在 globprot.inc 中定义的一样。

```
[Firmware]
GETTIME=GETTIME
```

图 8 将 GETTIME 添加到 SMARTSIM.INI

将一个以你的函数块命名的子段和特定固件块原型加入到连接器中。为了方便，仅拷贝和复制一个存在的字段并修改名称即可。更多细节见 3.3.1.2.8.

对于此例子，通过如下方式将字段加入到 smartsim.ini 中：

```
[GETTIME]
Instantiate=1
CRStack=1
RetStack=1
Firmware=1
Group=2
Index=111
DataBytes=??
Data=??
```

图 9 将 GETTIME 加入到 SMARTSIM.INI 中

“Index”是一个（相当）可靠的值。数据和数据类型仍然是未知的。在以后的步骤都将被添加。在 4.2 版本固件函数块将被分组。这为 OEM 处理实时系统更新提供了更合适的方式。第 0、1 组为 infoteam 所保留。OEM 可使用第 2、3 组。”index”在每个组中从 1 开始。在实时系统中这些分组被表现为下列 4 个表：

0= StandardFBTab_g

1= InfoteamFBTab_g

2= Oem1FBTab_g

3= Oem2FBTab_g

3.5.2.5 布局数据段

从你设计的接口布局此函数块的数据段实例。详见 3.3.1.2.8.

对于我们的例子，得到如下所示的数据段

Bytes	Contents
0..5	Segment Header
6	Hours
7	Minutes
8	Seconds
9	Twelvehours

图 10 GetTime 数据段布局

3.5.2.6 布局初始数据段

为了能够正确地初始化所有函数块创建的实例的数据段，编译器为所有的函数块设置了初始数据段。详见 3.3.1.2.8。

对于我们的例子，以如下方式构造初始数据段：

初始数据段的大小——未知，这里先预留 2 个字节的空间：0x?? 0x??

初始数据段 ID——必须为"2"： 0x?? 0x?? 0x02

段头的长度——必须是 16：0x?? 0x?? 0x02 0x10

相应段——也未知，但需要预留 2 个字节的空间好让编译器稍后补充。通常，0xCD 的样式用于标记一些未初始化的存储空间，所以：0x?? 0x?? 0x02 0x10 0xCD 0xCD

A+B+C 字段（例：VAR_IN,VAR_OUTPUT,VAR_IN_OUT）的大小：在输入模式，需要 4 个字节来保持所有的非保持输入和输出参数，所以得到：0x?? 0x?? 0x02 0x10 0xCD 0xCD 0x04 0x00

D+E+F,G,H 和 I 字段的大小——在我们的例子中不需要这些字段，所以得到 0x?? 0x?? 0x02 0x10 0xCD 0xCD 0x04 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00

A+B+C 字段内容：这里我们列举了在这些字段里的变量被初始化的存储目录。所有的变量赋予 0 的初值，所以仅增加另 4 个空字节即可：0x?? 0x?? 0x02 0x10 0xCD 0xCD 0x04 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00

D+E+F,H 和 I 字段内容：我们为这些字段分配了 0 存储长度，所以不用为这些字段添加数据：0x?? 0x?? 0x02 0x10 0xCD 0xCD 0x04 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00

现在初始数据段已构造好，计算所有的字节数长度并填入（在本例中为 20 字节）：0x14 0x00 0x02 0x10 0xCD 0xCD 0x04 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00

将此字节串作为 DATA 的值加入到你的 SMARTSIM.INI 文件中，但是要省去"0x"，仍然用 16 进制数。"DataBytes"用 10 进制数表示初始数据段的长度：

```
[GETTIME]
Instantiate=1
CRStack=1
RetStack=1
Firmware=1
Group=2
Index=111
DataBytes=20
DATA=14 00 02 10 CD CD 04 00 00 00 00 00 00 00 00 00 00 00 00 00
```

图 11 在 SMARTSIM.INI 中的 GetTime

注：在 Windows 中对 INI 文件的一行的长度有限制。如果你需要更长的空间，仅需要在 DATA 后面加入一行"DATA2"(DATA3,DATA4…)

3.5.2.7 检查

再一次按 3.5.2.3 节所述编译你创建的资源，这个资源包含了对你的函数块的调用。它需要被正确地编译。

下载此程序到你的控制器并运行。将出现一个错误，因为我们在 SMARTSIM.INI 中设置的"index"是错误的。

3.5.2.8 向函数块添加代码

找出 OpenPCS 实时系统的 STDFB.C 文件。在文件的最后加入你的新固件函数。

为了使函数更易读，首先定义数据段的布局，见 3.5.2.5 节。在我们的例子中，可按下列方式：

```
#define OEM_GT_TWELVE 0x09
#define OEM_GT_HOURS 0x06
#define OEM_GT_MINUTES 0x07
#define OEM_GT_SECONDS 0x08
```

图 12 GetTime 数据段布局

前缀 OEM_GE 使名称更清楚。现在按你的想法加入一个新的固件函数块。固件函数块通常是 LZSBYTE 类型，没有参数。函数的名称应该要使你想到此函数在 IEC1131 标准中的名称，但不必完全相同。在我们的例子中，可按下述来写：

```
LZSBYTE OEM_Get_Time(void)
{
    // TODO: add implementation
    return (kIecOK);
}
```

图 13 固件函数接口

固件函数块在没有错误出现时通常会返回 klecOK。这个值被返回到数据段并发到 IEC1131 系统。这个返回值通过固件函数块返回并传到 OpenPCS 的 IEC1131 内核。当 klecOK 出现一些不同的情况，应用程序将通过此内核停止。

为了使固件函数块更易读，一些#define 用于存取当前实例的数据段，见 IEC_FB.H。

在我们的例子中，如下所示：

```
LZSBYTE OEM_Get_Time(void)
{
    struct _dostime_t time;
    int TwelveHours;

    _dos_gettime( &time );                          // get system time
    TwelveHours = GETBIT(OEM_GT_TWELVE);            // get desired format
    if (TwelveHours) {                              // 12-hours-format?
        if (time.hour > 12)                         // need to adjust?
            time.hour -= 12;                        // adjust!
    }

    SETSINT(OEM_GT_HOURS, time.hour);
    SETSINT(OEM_GT_MINUTES, time.minute);
    SETSINT(OEM_GT_SECONDS, time.second);
    return (kIecOK);
}
```

图 14 GetTime 实现

3.5.2.9 添加跳转向量

为你的固件函数块加入一个指针到 STDFBTAB.C 文件中的向量 Oem1FBTab_g 中。作为注释，在这个向量中为新入口添加一个索引。

在我们的例子中，添加了 OEM_GT_GetTime,并且为每个入口编号以便能够很快的找到。本例为 77（仅为示例）：

```
OEM_GT_GetTime /* Index=77 get the current time */
};
```

图 15 GetTime 跳转向量

为了能够编译，在 IEC_FB.H 文件中加入的你固件函数的原型申明：

```
LZSBYTE OEM_GT_Get_Time(void);
```

图 16 IEC_FB.H 中的原型固件函数

3.5.2.10 添加索引

在加入指针到跳转向量时出现的索引就是 SMARTSIM.INI 文件中仍然缺失的 INDEX。所以回到 SMARTSIM.INI 将 INDEX 更新与目标索引一致：

```
[GETTIME]
Instantiate=1
CRStack=1
RetStack=1
Firmware=1
Group=2
Index=77
DataBytes=20 DATA=14 00 02 10 CD CD 04 00 00 00 00 00 00 00 00 00 00 00 00 00
```

图 17 Smartsim.INI 中的 GetTime(完成)

5.2.11 测试

如果没有出错，你可以成功测试你的新固件函数。确认再次编译实时系统，并在对 INI 文件的任何修改后重启 OpenPCS 系统。然后编写一些简单的测试程序来测试你的模块功能。

3.5.2.12 文档化

应该设计一个标签封装吗？我们推荐你在用户手册中加入各自的标准函数和模块的说明。

3.5.2.13 处理引用

如果你使用 VAR_IN_OUT 传递参数到函数块，那么函数块不会传递这个变量的值，而是它的一个“引用”或“指针”。

引用通常为 4 字节。为了存取（读和写）这个参数，这个引用必须被求值。引用的头两个字节规定了当前的数据段号，后两个字节规定了数据段的偏移量。例如函数 LzsMemFarGetByte。

3.5.2.14 显示初始值

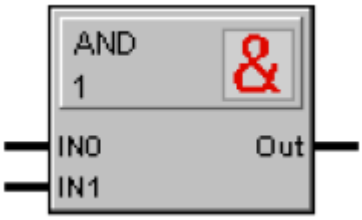
如果在一个输入或输出参数（在 GLOBPROT.INC 中）后有一个以”:=”开始的命令，这个命令在提示框中显示为这个参数的初值。像如下原型：

```
GLOBAL_PROTOTYP_BEGIN
    FUNCTION_BLOCK CTD
        VAR_INPUT
            CD :BOOL R_EDGE; (*ein normaler kommentar*)
            LOAD :BOOL; (*:= 300*)
            PV :INT; (*:= 100*)
        END_VAR
        VAR_OUTPUT
            Q :BOOL;
            CV :INT;
        END_VAR
    END_FUNCTION_BLOCK
GLOBAL_PROTOTYP_END
```

将得到如下提示框：

3.5.2.15 CFC 函数块符号

信号可以被指定给 CFC 函数模块。当一个 FB 嵌入后这些信号可被看见，如下图所示：



为了使用这个功能，创建一个和函数块拥有相同名字的.WMF 文件，并将此文件放入 Openpcs.520\Symbol 文件夹中。当一个 FB 嵌入到任何一个 CFC 文件中时，OpenPCS 会查找相符的.WMF 文件。如果发现此文件，信号将被放入函数块的右上角，尽管左边是空的。

3.5.3 固件块 4.3.1

OpenPCS 编程系统 4.3.1 版本中的固件块可被部分自动化，所以 3.5.2 节中介绍的步骤可以简化。

3.5.3.1 创建原型并发布到系统

通过 3.5.2.1 节可知原型可以自动产生。

这需要新建一个 OpenPCS 工程，并用 POU 编辑器新建一个函数块或函数。（一般地进程只通过一个.poe 文件工作）现在申明所有的变量，不同于 3.5.2.1 节所述仅仅 VAR_INPUT,VAR_OUTPUT,VAR_INOUT 变量类型需要申明，VAR(局部)和 VAR_EXTERNAL 也需要申明。

参照 3.5.2.1 节的例子并增加一些局部变量，在 GET_TIME 函数块中的申明部分如图 18 所示：

```
FUNCTION_BLOCK GET_TIME
VAR_INPUT
        TwelveHours: BOOL;
END_VAR
VAR_OUTPUT
        hours : SINT;
        minutes : SINT;
        seconds : SINT;
END_VAR
VAR
        Temp1 : BOOL;
        Temp2 : SINT;
END_VAR
```

图 18 GET_TIME 函数块申明部分

在 POU 编辑器中执行语法检查后（File->Syntaxcheck），会在工程目录下的gen目录下生成一个原型文件(.ptt)。

通过打开一个外部编辑器将入口拷贝到 OpenPCS 目录下的头文件 Globprot.inc 中（见 3.5.2.2 节）。请确定函数块的名称和你的固件块的名称是一致的。

3.5.3.2 创建初始数据段布局

现在手动地生成初始数据段布局可由 OpenPCS 编程系统编译器自动处理。

你必须用带”-Y”前缀的命令行来调用编译程序。

“itmake –Y AbsolutePath/name.var-file “

在我们的例子中，命令行如下所示：

itmake –Y D:\pattern\pattern.var

然后会在gen\工程资源目录下为每个函数块生成一个.ids 文件，如图 19 所示：

```
;PceDeviceName=IEC_IEC_NORM
;MotorolaFormat=0
;Alignment=1
;FractLayout=0
;ByteSize=1
DataBytes=22
Data=16 00 02 10 CD CD 04 00 00 00 00 00 02 00 00 00 00 00 00 00 00 00

#defines for Firmware Implementation:

#define GET_TIME_TWELVEHOURS 0x09
#define GET_TIME_HOURS 0x06
#define GET_TIME_MINUTES 0x07
#define GET_TIME_SECONDS 0x08
#define GET_TIME_TEMP1 0x0a
#define GET_TIME_TEMP2 0x0b
```

图 19 GET_TIME ids 文件

“DataBytes=…”和”Data=…”的输入可由 hardware.ini 文件中固件输入拷贝而来，见 3.5.2.6 节所示。以；添加的注释语句仅显示编译器的设置，即布局的设置。

接下来的定义是固件块相应变量的偏移量。供固件块在实时系统中使用。编译器关注硬件模型中对齐和字节顺序的分配，这些对齐和字节顺序在可执行资源中分配。在 OpenPCS 编程系统的原型资源中不同的硬件应设置不同的硬件模型。

所有 3.5.2.6 节中提到的入口定义仍需要手动实现。

3.6 操作系统

SmarPLC 可以操作几乎所有的操作系统，甚至不用操作系统。在移植期间，我们将详细介绍怎样按你的选择将 SmarPLC 移植到操作系统中。

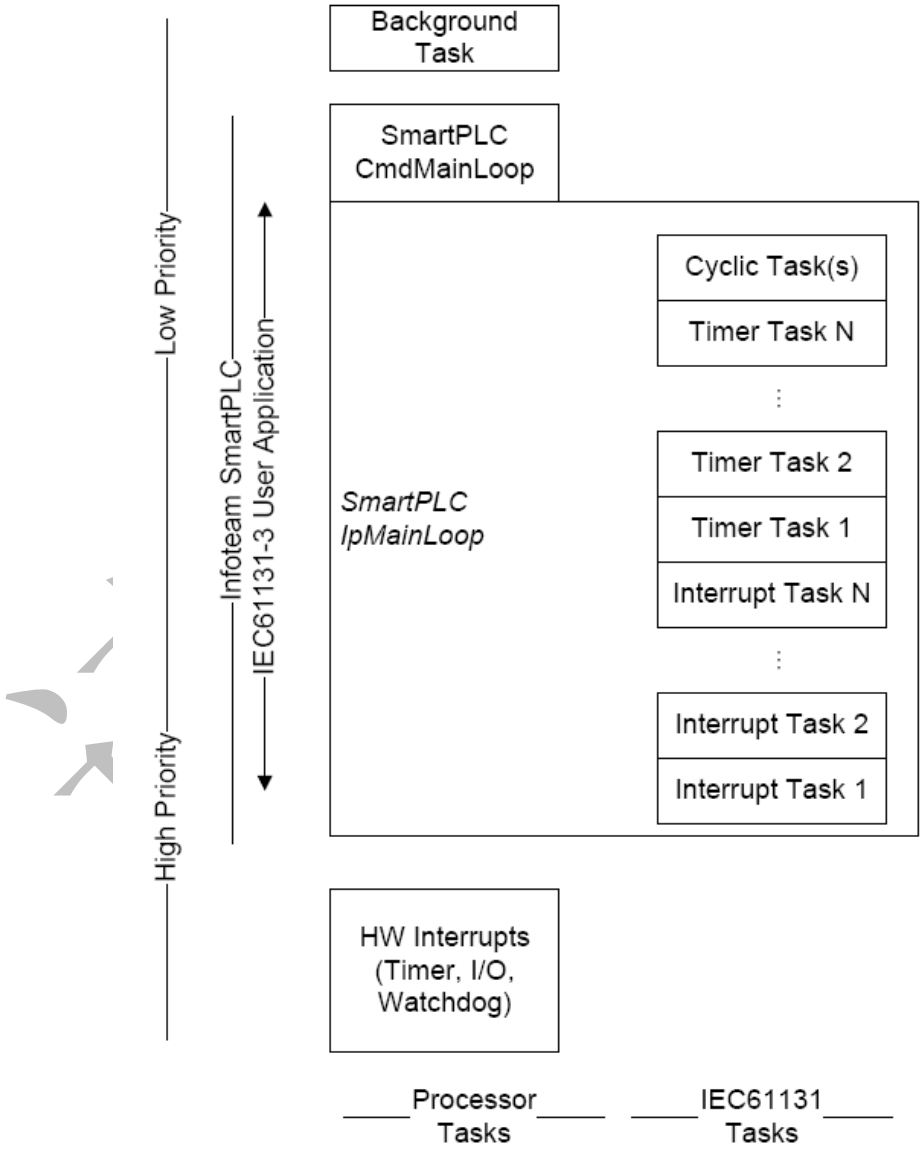
按照经验，以下设计模式在大多数情况都被成功应用。

a)无操作系统（见 3.6.1 节）

b)实时操作系统（见 3.6.2 节）

c)双操作系统（见 3.6.3 节）

3.6.1 无操作系统

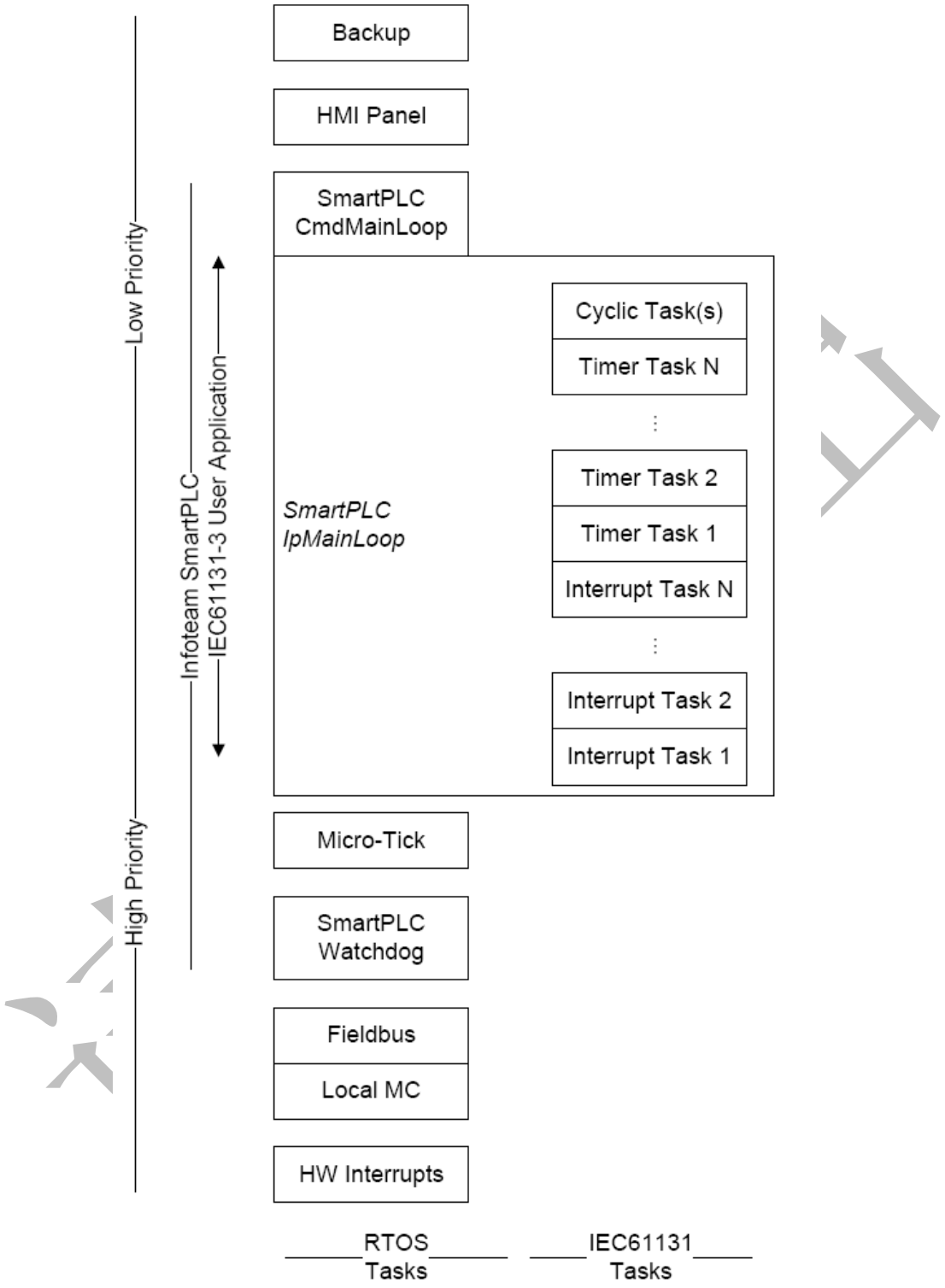


在没有实时操作系统的情况下，目标系统的调度将由 SmarPCL 调度程序完成。IEC61131-3 任务将按 IEC61131 标准来调度。中断延时和晶振时钟任务取决于最优设置和应用程序的结构。

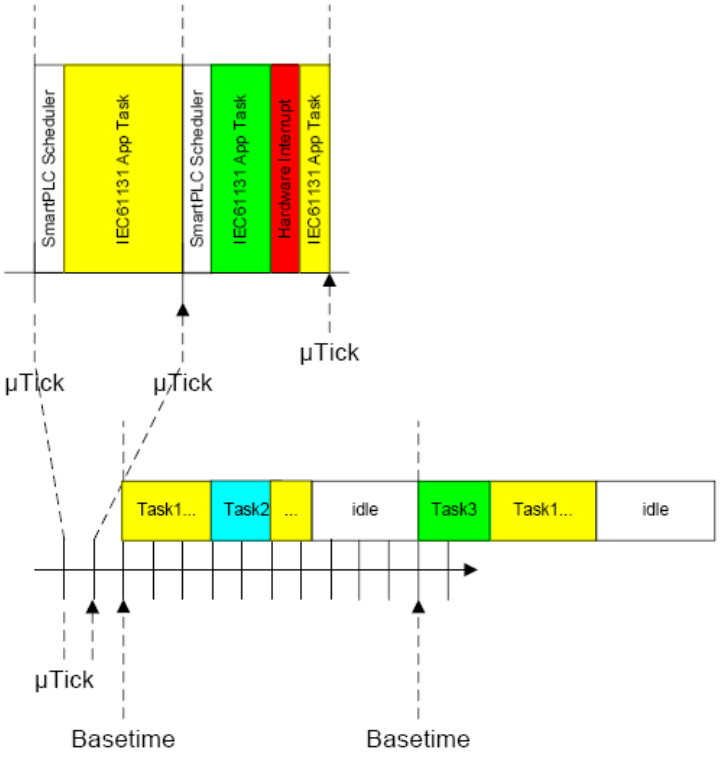
高优先级问题（如系统标记中断、硬件中断、看门狗）通常连接到 SmartPLC 外部程序中断。低优先级任

务可由一些与 SmartPLC 挂钩的外延程序完成。

3.6.2 实时操作系统

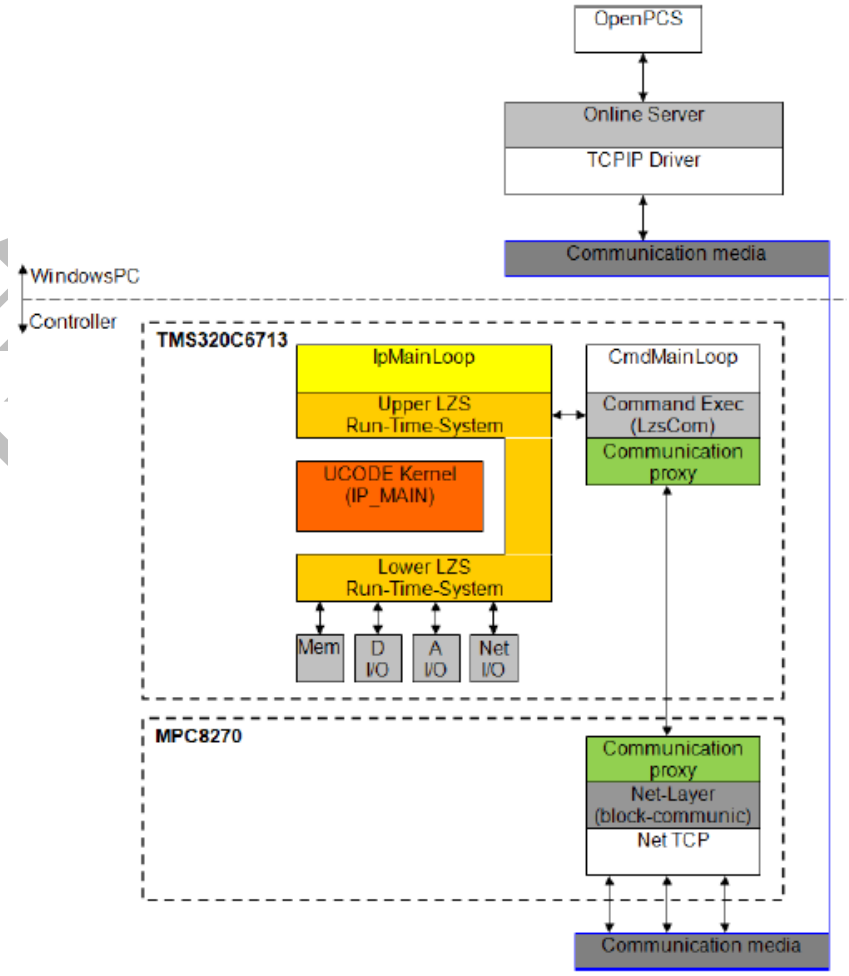


在实时操作系统中，SmartPLC 通常使用一些不同的优先级来完成不同的服务，如上图所示。高优先级和低优先级任务根据各自不同的优先级来映射。所有的 IEC61131-3 任务被在相同优先级上的操作系统处理。IEC61131-3 任务调度由 SmartPLC 调度程序完成，但是任务的调度由一个”μTick”触发，且由一个单独的优先级执行。运行效果如下图所示：



中断延迟和晶振时钟任务取决于 µ Tick。

3.6.3 双操作系统



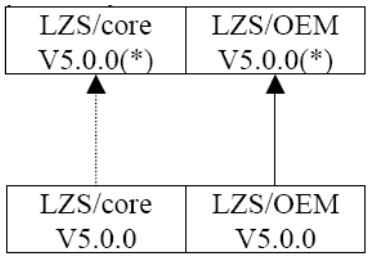
在一些架构中，多重操作系统运行在一个系统中。通常出现在一个通用系统（Windows，Linux）被一些实时系统（RTAI,RTX）扩展的时候。在这种情况下，当 IEC61131-3 应用程序需要在实时系统上运行时，一些资源（如 TCP/IP）通常会更容易通过通用操作系统处理。上图显示了本系统的常用 SmartPLC 架构。

3.7 示例

示例章节（SmartPLC\Samples 目录下）为不同的平台/操作系统提供了预构建实时系统集。
更多细节请参考相应文件夹下的 readme 文件。

3.8 更新嵌入式 SmartPLC 实时系统

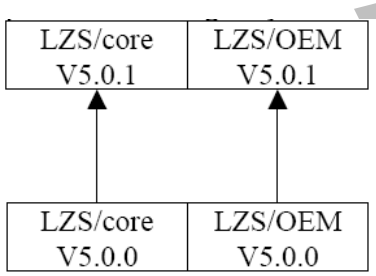
本节介绍当向 infoteam 申请后，怎样将你的目标系统更新到一个新版本的嵌入式 SmartPLC 实时系统。
在支持移植最终产品完成后，你将得一得到如下状态：



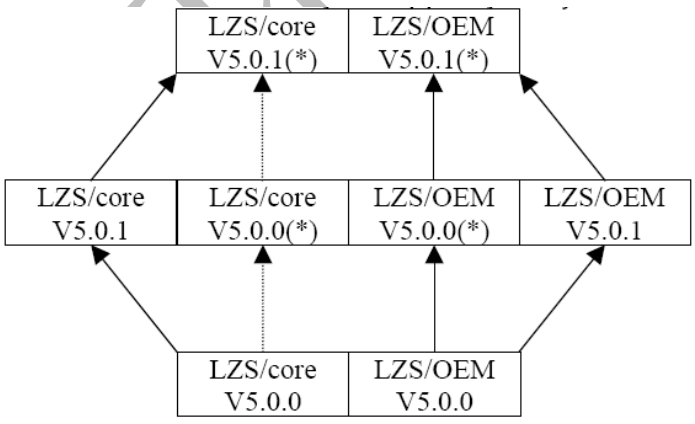
1.你必须确定目标系统通过 OEM 计划修改和移植的部分已经被修改（实线箭头），例如 LzsEnv.c,StdFb.c,config.h 等。修改后的版本与 infoteam 保留的版本是不完全相同的。

2.你需要修改与通过 OEM 修改的不完全相同的 OpenPCS 部分（虚线箭头），例如：IpCmd1.c。这是使自己适应于处理器或操作系统或其他组合的案例，且不再支持 config.h 中简单的配置选择。

当推出新版本时，infoteam 将所有的目标系统修改包含进去。例如：



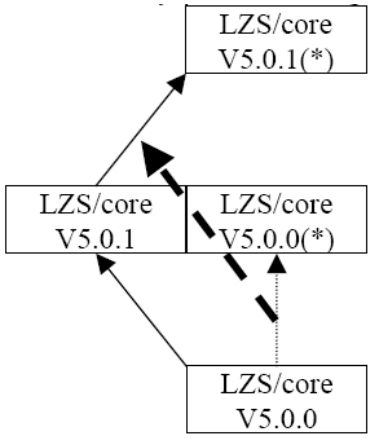
你不需要自己合并这些通过 infoteam 增加的修改，例如，你可以为新目标系统创建一个由 infoteam 提供的修改后的版本，其包含的修改同样支持你的系统：



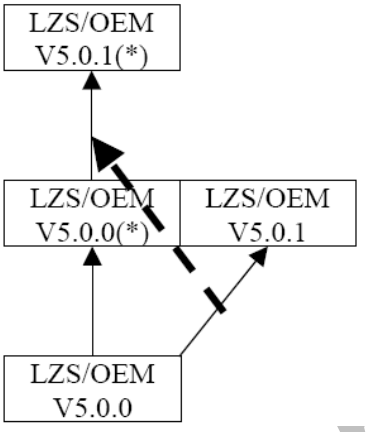
通常可以两种方式这样做：

1.仔细分析你对原目标系统所做的修改，在新提供的目标系统中找到进行修改的正确的位置，然后对新提

供的目标系统进行相似的修改，创建一个新的修改后的目标系统。



2.仔细分析 infoteam 提供的新版目标系统的改进，根据你的环境决定需要哪些适当的改进，然后适应的包含到你的目标系统中。



哪个种方式更可靠取决于个体的历史和环境，你可以根据自己的需要选择。一般你修改得较少，修改文档全，最好使用方式 1；若错过更多的版本更新，infoteam 提供了更多的改进，最好使用方式 2.。如果两都都有，则在更新中包含 infoteam.。

在任何一种情况下：

a)确保你的文件有安全的备份，且在做所以事之前做好。

b)如果你对 infoteam 交付的原始目标系统的修改不确定，重新安装到一个不同的地方且比较文件的差异会有所帮助。如果你对比较文件的差异没有经验，infoteam 可帮助你找到正确的工具——询问支持线。

c)相同的比较文件差异的工具有助于得到最终的 infoteam 对不同版本目标系统的改进的信息。

d)在 LZS.HLP 文件中的”history”内有 infoteam 所做改进的背景信息。

3.9 SmartPLC/冗余

冗余 SmartPLC 是一个基于嵌入式 SmartPLC 的 1oo2 容错系统。

相比嵌入式 SmartPLC，它有更高的可靠性和实用性。

容错就是实现冗余的意思：

两个完全相同的冗余 SmartPLC 实例控制器是并行运行的。假设一个系统出现故障，那么另一个系统可以接管（失效备援）。

接管使得两个系统紧密的同步在一起（热冗余）。

冗余功能是自动提供的。当改进 IEC 应用程序时可不用考虑。

3.9.1 一般概念

一个冗余 SmartPLC 系统包含一个主 PLC 和一个从系统。两个系统都运行扩展的嵌入式 SmartPLC 实时系

统并且通过一个快速同步通道同步。同步发生在 IEC 任务层：在 IEC 任务开始（读取输入）的时候和完成 IEC 任务（写输出）的时候。

另增加一个通信通道用于同步实时系统间的时钟源。

3.9.1.1 任务同步

同步由一个基础任务执行且具有下列属性。

1.如果任务是由主系统的调度程序开始，那么任务号和全局变量数据将发送到从系统。从系统响应一条回复信息。

2.当一个任务完成后，第二个同步开始执行。

3.在同步点主系统和从系统都需等待他们的触发信息，此触发信息来自在达到同步点后附加 maxTaskJitter 值的主系统。

4.当从系统达到同步点后，如果不能收到来自主系统的附加 TaskJitter 值的同步信息，从系统将转换到独立运行模式。

5. 当主系统达到同步点后，如果不能收到来自从系统的附加 TaskJitter 值的同步信息，主系统将转换到独立运行模式。

6.仅有周期任务和时钟任务可以同步。中断任务在两者之间没有同步。因此大多数 maxTaskJitter 在指定时间内需要考虑延时。

7.双独立系统最有可能的情况是处于一个有效的状态并且此状态需被 I/O 硬件完成。（假设发生系统转换和独立模式的通信中断）

8.当系统重启时，它将自动探测和同步另一个处于独立模式的系统。当同步完成后，系统将加入到运行中的独立主系统中；如此，一个冗余主/主系统就又建立了。

同步任务进程的重要影响：

1.主系统和从系统的系统时钟（实时系统通常使用毫秒作为时钟计数）不可能百分百同步。因此当另一个系统第一次完整的执行任务直到它探测到另一个任务需要运行时，另一个系统需要中断一个运行中的任务，有利于另一个高优先级的任务执行。但是就目前的同步问题，任务中断是不可能的。这意味着当前低优先级任务没有执行完之前，高优先级的任务不能运行。

2.在任务运行前，任务同步控制信息在主从系统间交换。信息交换需要的时间由延时完成。

总之：即使任务同步延迟需要实时操作系统的支持，PLC 系统也不支持实时任务在主/从模式执行。

3.9.1.2 IEC 程序同步

IEC 程序的同步能被充分地处理。因此，持续的数据通过一个分开的通信通道交换到另一个系统（例如：标准 TCP/IP 网络）。

3.9.1.3 数据同步

数据同步被 IEC 应用程序的资源全局变量保障。所有其他的数据，像输入、任务全局或局部变量将不被同步。

用户必须确定在无全局数据的时候不处理任何输入—输出联系。

返回变量需要特殊考虑。它们不被直接地同步。如果它们以资源全局变量的方式申明，则可通过默认数据同步进程来运行同步。

在主系统和从系统间的数据同步中，需在下列位置执行必要的工作：

1.在任务的开始，资源全局变量作为任务同步命令的一部分从主系统发送到从系统。

2.如是主系统运行在独立模式，并且重启后的从系统希望再次同步，资源全局变量将被发送到从系统。

输入和输出的同步不能自动执行。但是可以可选择地完成。

当定义最大允许任务跳跃时，必需考虑数据同步采用的延时。同步延时取决于同步通道的速度和传输数据变量的数量。

3.9.1.4 时钟同步

实时系统使用 OEM 函数 LzsEnvGetEickCount()来询问基于毫秒的时钟源。计数器返回值必须在主从系统间同步，使得两个系统上的调度程序可以选择相同的任务来运行。

同步可在任务同步的高速通道之上进行。但需要其他的脚本。

同步计数器需要每秒计数几次，吸收两个 CPU 间不可避免的时钟漂移。因此，同步也可以由主系统到从系统的心跳交换表示。如果从系统检测到计数器超时，它可以开启独立模式且不受任务同步状态的约束。

3.9.1.5 I/O

依赖 OEM 的不同的 I/O 结构的有效性需求成为可能。嵌入式 SmartPLC 和冗余 SmartPLC 都有支持任何类型的 I/O 结构的灵活的接口。

右图显示了可能的 I/O 结构：

1）单面的

- 一般实用性

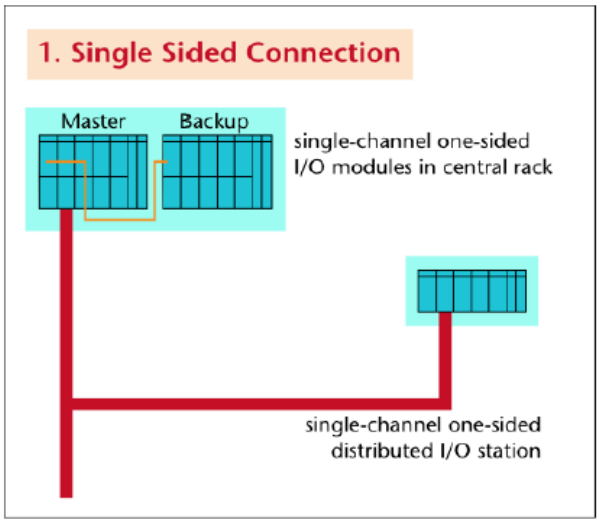
2）开关的

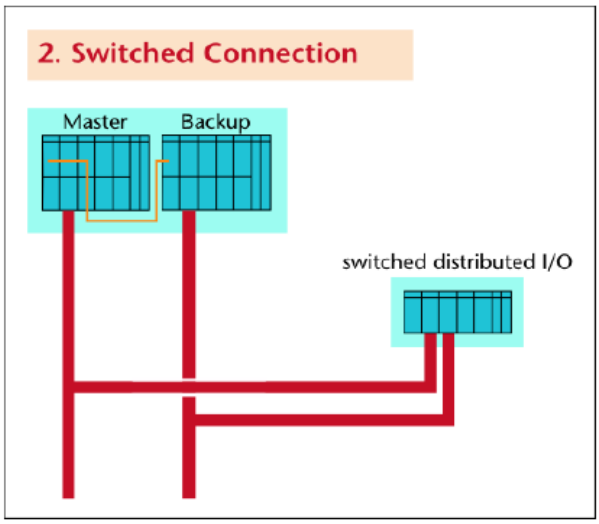
- 高实用性

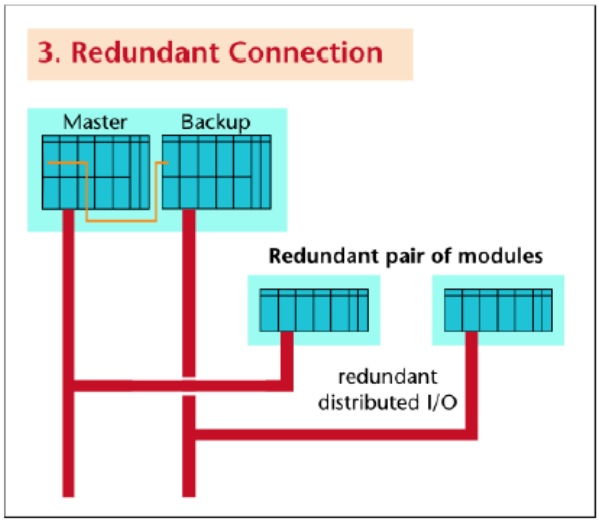
3）冗余的

主系统和从系统拥有各自的模式

- 容错







3.9.1.6 开始和结束 IEC 应用程序

在主系统和从系统开始和结束期间的同步由以下步骤完成，在那里需要区别两种脚本，硬件或软件开关。

3.9.1.6.1 通过 OpenPCS 启动/停止

与主系统一起在线的 OpenPCS 编程系统，可启动或停止整个系统。

实时系统的一个停止命令可导致主系统和从系统的停止。

对于一个已停止的系统，启动命令执行主系统和从系统的启动。

在新的程序被下载到主系统后，直到从系统与新程序同步后才可以启动系统。(通过检测超时进程来结束同步)

3.9.1.6.2 通过硬件开关启动/停止

硬件开关支持下列模式

- 运行
- 停止
- 重启：初始化所有数据段

通过硬件开关对主系统的停止命令只令主系统停止，从系统接管并成为一个独立系统。

硬件开关对从系统的停止只令从系统停止。主系统则转换到独立模式。

在单系统上的启动命令导致系统成为独立系统。

对第二个系统的启动命令将使系统进入从系统模式，并且另一个（独立）系统将转换到主系统模式。

3.9.1.7 启动和结束系统

在本节，将介绍如果一个或两个系统上电后，会发生什么。

如果系统拥有上述章节所述的硬件开关。上电与硬件开关从启动到运行是一样的。

有两种可能的状态：

1.两个系统同时上电

在上电后，两个系统将通过同步通道发送信息来相互检测。如果在一个可配置的时间内一个系统不能检测到另一个系统，那么这个系统会进入独立模式。在常规操作中，另一个系统回复并且在各自的日期和 IEC 程序的有效性的基础上，两个系统将协商他们的角色（主或从）。如果必要会建立一个从主系统到从系统的 IEC 程序同步。然后，两个系统将运行此 IEC 程序。

2.一个系统正在运行且另一个系统上电

此情况出现在一个系统掉电并重启的时候。

当前，一个系统运行在独立模式且另一个系统上电。已经在运行中的系统成为主系统。上电系统将与主系统程序同步并成为从系统。主系统将在两个任务执行间隙短暂停止，与从系统同步数据。然后，两个系统都执行 IEC 程序同步。

3.9.2 先决条件

为了在控制器上完全支持冗余 SmartPLC，需要完成一些配置。

这些需求将在嵌入式 SmartPLC 实现期间被检查/讨论。

需求：

- 硬件
  - 足够的 CPU 速度
    需要在任务同步时加入一些额外的负载到系统
  - 支持持久性磁盘
    通常为基于存储的 flash
  - 可选的：冗余 LED 状态指示和可视化主/从状态（所有运行/停止和错误状态：建议用 4 个 LED 灯）
  - 8 位 CPU（ByteSize=1）
    控制器有 8 位 CPU 工作（其他系统也可支持但是没有测试过）
- 软件：操作系统
  - 实时端对端通信堆栈
    堆栈必需支持下列功能：
    - 超时发送和接收
    - 支持双通道（同步网络和时钟同步）
    - 数据率>=100MBit/s
    - CPU 加载生成的大量的任务同步信息必须是相配的。
  - 实时操作系统
    为了支持实时通信处理
  - 支持连续的文件交换（主系统->从系统）
    目前通过 TCP/IP 网络。其它不同的方式也是可行的。
  - 网络变量：支持 TCP/IP

3.9.3 实时系统改进

冗余 SmartPLC 的资源代码可做为嵌入式 SmartPLC 的分离备份插件使用。

一些冗余 SmartPLC 的改进是标准实时系统的一部分。

安装备份插件后需要使能。为此，下列语句需要加入到 inc/oem/config.h 文件中：

#define _LZS_REDUNDANCY_

冗余备份插件的文件清单：

Filename	Purpose
lzsred.h / lzsred.c	初始化/关闭，内部状态机控制
lzsredtsync.c	任务同步
lzsrednet.c	高层同步通道网络连接
lzsredenv.c	OEM特定拦截函数

lzsredenvnet.c	OEM特定低层同步通道网络连接驱动程序
lzsredenvhb.c	OEM特定时钟源同步实现（计数器同步）
lzsip_red.c	改进实时系统高度程序（替代文件lzs/lzsip.c）

所有的 lzsredenv*.c 文件需要被 OEM 执行（比较冗余 SmartPLC 备份交付的执行示例）

3.9.3.1 OEM 指定改进

OEM 必要的改进可分为如下几类：

- 任务同步通道的实现（也用于交换控制信息）
- 计数器同步的实现
- 适应持续的数据同步

下列 OEM 函数需要被执行，使得冗余 SmartPLC 适应 OEM 指定平台。

LZSBYTE LzsEnvRedInitialize(void)
在系统启动时调用的初始化函数
LZSDWORD LzsEnvRedGetTickCount(void)
返回以ms表示 的计数值。
进程映像函数。 如果直接寻址的处理已经被自定义了，那么下列注意进程映像的函数需要被调整。如果使用标准的SmartRedundant.dll，则不必做任何改变。
LZSBYTE LzsEnvRedCheckConfiguration(tPlcMemPtr pConfigSegment_p)
在下载后或持久性被恢复后调用。收集进程映像安装信息，来自配置段的最大任务跳动和最大周期长度。
LZSBOOL LzsEnvRedIsHwSwitchInStop(void)
如果一个硬件的RUN/STOP转换是有效的，那么这个函数应该返回它的状态 (如果是STOP 位置则返回LZSTRUE).
LZSBOOL LzsEnvRedIsInErrorState(void)
只要这个函数返回LZSTRUE,系统就将进入 STOP模式(其它系统将接管).当它返回LZSFALSE,将进行普通启动。如果另一个系统正进运行，这个系统将同步并且连接上它。这个函数意图通知基本系统一个故障的冗余逻辑。例如如果I/O数据是暂时无效的。
LZSBYTE LzsEnvRedHeartbeatInitialize(void) void LzsEnvRedHeartbeatShutdown(void)
这些函数设置或关闭在主机和备份间的心跳/计数交换。 函数LzsEnvRedHeartbeatInitalize在系统启动时从LzsRedInitialize调用（实际上是从LzsEnvInitialize调用）。 从实时系统的视角出发，计数器提供者是函数LzsEnvGetTickCount()。这个函数需要被修复来为主机和备份提供一个同步的系统时钟。因此平台特定计数器（timer）可与一个计数偏移一起使用，来在一个短的间隔内获得主机与备份间的同步。
Void LzsEnvRedOnRoleChanged(tLzsRedRole RoleCurrent_p, tLzsRedRole RoleNew_p)
如果一个角色转换被发现，这个回叫将被调用。OEM特定活动可在这里完成。 可能的角色(见lzsred.h): •kLzsRedRoleNone: 既不是主机也不是备份。这种情况是系统是否通过硬件转换或通过LzsEnvRedIsInErrorState来启动或已经进入STOP模式。 •kLzsRedRoleMaster: 系统是主机且在主机/备份模式 •kLzsRedRoleBackup: 系统是备份且在主机/备份模式 •kLzsRedRoleStandAlone:系统处于孤立模式(例如：它没有与另一个系统联合)

通信函数：
tLzsRedNetRecvResult LzsRedNetRecv(tLzsRedNetConnection Connection_p, tPlcMemPtr pBuffer_p,
LZSDWORD dwBufferLen_p, LZSDWORD *pdwRecvLength_o, LZSDWORD dwMaxTimeMs_p);
两个系统间在高速连接上接收数据。 这个函数将限制dwMaxTimeMs（ms）或者如果被指定为0则一点都不限制。 如果在指定的时间间隔内没有有效的数据，那么返回值指出了一个超时(kLzsRedNetRecvResultTimeout)。 数据被写入到传递的缓冲区pBuffer_p。这个缓冲区的大小由dwBufferLen_p给出。已接收的字节数通过pdwRecvLength_o返回。 Connection_p参数是有交的，所以这个函数同样可用于其它用途。一个kConnectionInterconnect的参数值指的是系统使用的通信通道。另一个用途是执行计数交换。 返回值： 成功：kLzsRedNetRecvResultOk. 错误：kLzsRedNetRecvResultError.
tLzsRedNetSendResult LzsRedNetSend(tLzsRedNetConnection Connection_p, tPlcMemPtr pBuffer_p, LZSDWORD dwBufferLen_p, LZSDWORD dwMaxTimeMs_p)
两个系统间在高速连接上发送数据。 数据被传递给pBuffer_p。发送的字节由dwBufferLen_p给出。数据的发送必须在dwMaxTimeMs_p ms内被接受，否则将返回一个超时(kLzsRedNetSendResultTimeout)。 成功时，LzsRedNetSend返回kLzsRedNetSendResultOk。 出错时，返回kLzsRedNetSendResultError。 见LzsRedNetRecv中对参数Connection_p的更多细节。

3.9.4 OpenPCS 改进

冗余 SmartPLC 系统加入了配置选项（像 maxTaskJitter）。这些配置可通过一个 OpenPCS 自定义性能对话框来配置。并存储在工程的 Makefile 文件中。

一个自定义的硬件配置 DLL 文件提供了扩展对话框和编辑时间，它将冗余指定选项存储在一个特殊的配置段，此配置段将同 IEC 程序一起传输到控制器中。

冗余 SmartPLC 备份包括了硬件配置 DLL（SmartRedundant.dll）的资源代码。如果需要 OEM 指定扩展项的话，DLL 资源代码可作为一个起始指针使用。

3.9.5 网络变量

网络变量是创建 PLC 分布系统的基础。在 IEC 程序中被定义为网络变量的变量将在所有被配置来参与网络变量数据交换的 PLC 中明显地交换。数据通过以太网交换。

网络变量是冗余 SmartPLC 可选择的功能并且只有冗余 SmartPLC 支持。

3.9.5.1 常用特性

网络变量系统有如下限制：

- 网络图片（网络字节数）：<=64kB
- 参与 PLC 数量：<=253

- 支持的 IEC 数据类型：BOOL,BYTE,USINT,SINT,WORD,UINT,INT,DWORD,UDINT, DINT,TIME,REAL
- PCL 间仅限于完全相同的对齐、字节顺序和字节数的交换。

3.9.5.2 网络变量概念

网络变量的数据交换基于环形拓扑结构。

在 PLC 之间有一个简单的全局数据包传输。这个包包含了每个加入的 PLC 的分离的子段。每个 PLC 都可以读取所有其他子段的值，但是只能改写自己的子段。子段号由设立在编程系统方面的配置数据定义。

物理上来看每个 PLC 都与其他 PLC 完全连接，所以控制信息可用广播的形式发送并且可忽略丢失的 PLC。

3.9.5.2.1 网络变量的交换

在环形拓扑结构中运用的全局包有如下结构：

MemorySection1	MemorySection2	MemorySection3	MemorySection4

PLC1 在执行它的 IEC 任务时只能写子段 1 但是可读取其他每个子段。子段号与关联 PLC 或各自的 IP 地址间的映射由一个配置文件提供。这个配置通过硬件配置段提供给 PLC。如此 PLC 在网络中决定它的下一个合作伙伴或每个其他的合作伙伴。

3.9.5.2.2 动态反应

网络变量数据交换在网络不完备的时候也可进行（例如：部分配置点并不在线）。至少有两个网络点在线时就可进行交换。

每个新的节点入网后将自动包含进行数据交换中。

如果一个节点掉线（例如：掉电，关机，开关电源故障，电缆故障等），那么这个节点将被检测和忽略。数据在正常节点间继续交换。

完整的网络和一个或一组节点的在线状态一样可在 IEC 应用程序中可编程地设置，通过固件函数块 TestNetVarStatus 和 GetNetVarStatue 实现（见 OpenPCS 在线帮助）

3.9.5.3 配置

拓扑逻辑通过 XMS 文件配置。此文件放在ENV目录下的 OpenPCS 工程中，并且必需以 netvarconfig.xml 命名（如<project directory>/ENV/netvarconfig.xml）。

这个文件给出的网络变量节点配置将被分享到所有 OpenPCS 工程定义的资源中。

配置定义如下：

```
<Configuration>
    <Resource name="Res1" IPAddress="192.168.1.217">
        <NetVarImage>
            <StartOffset>0</StartOffset>
            <EndOffset>9</EndOffset>
        </NetVarImage>
    </Resource>
    <Resource name="Res2" IPAddress="192.168.1.219">
        <NetVarImage>
            <StartOffset>10</StartOffset>
            <EndOffset>19</EndOffset>
        </NetVarImage>
    </Resource>
</Configuration>
```

“Name”：OpenPCS 工程资源名称

“IPAddress”：节点 IP 地址

StartOffset/EndOffset：网络图中资源/PLC 的指定段

对于一个冗余系统，这两个节点中只有一个被列举到配置文件中。

配置数据通过配置段传输到实时系统。

每个列出的节点都有一个索引（0…n-1;n=节点数）作为标记，此索引基于配置中出现的数量级。索引能够识别拓扑中的节点。一个节点的索引通过比较装载工程的资源名和实时系统中的配置数据来决定。当查询状态信息时可使用节点索引，通过网络变量固件函数块 TestNetVarStatus 和 GetNetVarStatus。

3.9.5.4 实时系统安装

网络变量的实现在实时系统 config.h 文件中使能：

#define _LZS_NETVAR_

下列文件需加入到 Makefile 中：

Lzsnetvar.c（网络变量逻辑）

Netvarfb.c（网络变量指定固件函数块）

下列函数给出了接口的实现：

LzsNetVarInitialize()

初始化网络变量模式

LzsNetVarShoudown()

关闭网络变量模式

LzsNetVarProcessVarCom()

控制网络变量数据的数据交换，和控制节点出现/消失的控制数据一样。当此函数调用时，它将检查进来的信息，更新接口状态并回复。这种交叉处理的方式和 LzsCmdMainLoop()的使用很相似。

下列 OEM 函数需要修改然后加入（见示例实现 Sample/linuxrtairedundant）：

LzsEnvCheckConfiguration()

网络变量拓扑配置从配置段提取，并且本地节点安装已完成。

LzsEnvReadProcImg/LzsEnvWriteProcImg

网络图像和全局数据包的数据交换被传送到拓扑结构中。

3.9.5.5 与 OpenPCS 的协作

网络变量的定义与直接变量相似：

```
VAR
    netvar1 : int at %N<offset>
END_VAR
```

将%N 变量申明映射到一个网络图像位置（偏移量）由 SmartRedundant.dll 完成。映射可被调整。

目前此偏移量被直接指定。

OpenPCS 从未提供支持网络变量的扩展函数。网络配置文件必须用另外的编辑器分别编辑。

3.9.5.6 包含网络变量的冗余系统

网络变量在冗余系统中执行有如下含义：

1.只有主系统或独立运行的系统可能成为网络系统中一个有效的搭档。

2.网络图像（网络变量数据）将在主系统到从系统的任务同步点进行同步。

3.从系统到独立运行系统的角色转换（如：主系统丢失）将导致从前的主系统的网络变量连接中断并且从前的从系统将接管网络变量通信。

4.网络连接错误（拔出 LAN 线缆）将通过下列方式与冗余任务对接：

a.主系统出错——主系统将切换到”none”模式，这意味着停止 I/O 交换。网络变量交换同样被停止。从系统将转换到独立运行模式并且开始网络变量交换。

i.如果从前的系统通信恢复——此系统将转换到从系统模式。它不参与网络变量通信，但可通过与主系统的同步点得到网络变量。

b.从系统错误——从系统将切换到”none”模式。

i.如果从系统通信恢复——此系统将回到从系统模式。它不参与网络变量通信，但是可通过与主系统的同步点得到网络变量。

c.两个系统都出错（主系统和从系统）——主系统和从系统都切换到”none”模式，即停止 I/O 交换。网络变量交换也将停止。

i.如果两个系统都恢复通信——同时启动主系统和从系统并进行模拟，从而作为一般规则来决定哪个系统变成主系统或从系统。

在这种方式下新的主系统接管网络变量通信，从系统通过主系统得到网络变量。

4 工具

4.1 SERTEST 工具

4.1.1 概述

DOS 应用程序 comtest.exe 和 rectest.exe（仅作为示例，需先行创建）是测试 V24 通信的简单的工具，并且 comtest.exe 在 PC 上运行，rectest.exe 在 PLC 上运行。

目标是成功的通过串行线路传输一个二进制文件，再接收此文件并进行比较。

这个工具被分成两个文件夹，包括"transmitter"(PC)和"receiver"(PLC)。

\tools\SerTest\Pc 包含了comtest.exe和资源comtest.c, ser.c, ser.h, ,由标准C语言编写,还有Pcl4c.h,Pcl4c_l.lib

\tools\SerTest\Plc 只包含资源 comrec.c, ser.c 和 ser.h。

主程序仅可由下列4个命令创建：

- SerOpenComm
- SerCloseComm
- SerReadComm
- SerWriteComm

这些命令同样可由OpenPCS调用。函数需要加入代码，使得你的硬件能够使用串口连接。

PC（comtest.exe）首先传输一个字节（ONE_BYTE）到PLC，此时rectest.exe等待接收这个字节，并转化它，然后发送回PC的串口部分。在这里被转化的字节将被检查，并且成功后将有一个文件被发送、回传和比较。

这个文件被分成xx(默认：500)字节大小的块，并且被一块块地发送、回传和检查。如果比较失败，那么两个块（发送和接收）都将被保存当前目录下的文件中（OutTr.bin和OutRec.bin）。

Ser.c包含以下函数：

- int SerOpenComm(int idComDev)
- int SerClose Comm(int idComDev)
- int SerReadComm (int idComDev, char FAR * cpBuf, int cbRead)
- int SerWriteComm (int idComDev, char FAR * cpBuf, int cbWrite)

在PC方面ser.c文件中使用串口端口的函数中的代码已完成。这些命令包含在库Pcl4c_I.lib中。Pcl4c支持个人的C语言串行通信库并且受MarshallSoft Computing.的版权保护。更多细节见http://www.marshallsoft.com/。在PLC方面，你需要加入你自己的特定代码到硬件中。

ComTest.c包含：

- main()函数和提供当前时钟（ms）的TickCount()函数
- 配置下列内容的定义：
- ONE_BYTE　　发送的"one byte"的值
- BLOCK_SIZE　　发送的块的容量
- WAIT_TIL_SUCCESS　　等待接收"one byte"的时间（ms）
- WAIT_AFTER_SUCCESS　　等待接收一个块的下一个字节的时间（ms）
- READ_INTERVAL_TIMEOUT　　读取操作超时

ComRec.c包含：

- main()函数和提供当前时钟（ms）的TickCount()函数
- 下列定义：
- COM　　串口端口
- BLOCK_SIZE　　发送的块的容量（最大为127字节），确保两个文件中的BLOCK_SIZE的值是完全相同的
- WAIT_TIL_BLOCK　　等待一个块的第一个字节的时间（ms）
- READ_INTERVAL_TIMOUT　　特定的最大时间（ms），在通信线路上允许两个到达的字符

间的时间消耗

4.1.2 线路参数

见2.3节的波特率和其他线缆参数。

4.1.3 启动程序

在完成你的绝对代码后，创建一个comrec.c的".exe"可执行文件。首先在PLC上启动这个应用程序。程序直到在接收到一个字节并且回传转换的字节后停止循环。

第二步在你的PC上启动comtest.exe，以一个文件名和串口号作为控制线路参数。ONE_BYTE将被发送并且停止PLC的循环。

4.1.4 结束测试

这个程序必须流畅地运行以传输大文件。在改变主程序代码来达到这个目地上无需犹豫。这将使移植更快和更容易。按我们的经验，创建一个稳定的通信是最消耗时间的因素。

4.2 SERTEST_ADVANCED 工具

4.2.1 概述

SERTEST_ADVANCED测试新增的设置，如超时——OpenPCS和嵌入式SmartPLC实时系统间串行通信的超时。这些设置将被下列测试检查：

1st Test：通过发送一个文件检查

2th Test：检查标准超时（带负载发送）

3th Test：检查长超时（带负载发送）

4th Test：检查内部超时（SerSetTimeout）

5th Test：检查带负载的内部超时（SerSetTimeout）

6th Test：检查启动超时（SerSetTimeout）

4.2.2 线路参数

COMTEST通过一些参数使你能够分开测试特定的问题。

调用的约定有：COMTEST <FILE> <COM> <lowTO> <highTO> <SIZE> [TESTS]。

第一个参数<FILE>是包含发送数据的文件。在测试中它被发送到PCL并且回发。所以不要使用太大的文件，以致测试需要很长的时间才能完成。建议2kB~50kB内为宜。

第二个参数<COM>指定了PC的COM端口。

第三个参数<lowTO>是短的延时（ms）。在测试2和5中作为一个包中的两块间的延时。PCL通过一次性读取这个包来代替延时。所以它将比SerTeadComm函数中的内部延时短。（如果更高的延时使测试失败你也可以试试）

第四个参数<highTO>是一个长的延时。在测试3中用于一包中两块间的延时。但是在这里PLC需要检测两个分开的包，所以选择一个比SerTeadComm函数中的内部延时更长的值。

第五个参数<SIZE>是主要使用的块的容量。每个测试包括一个按块发送文件的循环，所以块容量是在执行一次循环过程中发送一个包的容量。你需要测试1字节到1kB的范围。最常用的容量是120~130字节。更高的容量是可能成功的但是不是很重要。

第六个参数[TESTS]是可选的。它规定了测试的运行。如果没有设置，所有的测试都将运行。语言是很简单的。所以由于有6个测试，参数是一个6个位长的数，每个数字表示一个测试。第一个数字表示第一个测试，第六个数字表示第六个测试。如果数字是0，测试没有运行，1则表示测试在执行。例如：010011表示手续2，5和6个测试在运行。

4.2.3 启动程序

在PC方面，每个测试通过与PLC建立一个新的连接来初始化，并且通过发送一个字节的testnumber*10 + testnumber 来传输测试号（如：测试1发送11，测试4发送44）。PLC方面，在一个循环内等待接收这个字节并且

发送取反的字节回PC（在内部测试不会取反任何数据，所以这确保测试检测器已经发送了数据）。如果一个不可解析的测试号被发送了（如-1），PLC将停止。如果PC在300ms内接收到了这个字节，测试将启动。

4.2.4 测试

每个测试都将一个文件分成若干个块传输到PLC，并且PLC将它回传。超时、大小和数据的检查通过把它们与相应的期望的值做比较来实现。

一个测试运行直到出现超时或者它自己失效。超时由最后一个进来的字节和当前时间测量。超时大概为9到11秒，可用于发送相当于10kB到15kB的包。这么大的包不需要测试，但是能够测试。

4.2.4.1 测试 1

测试1简单地检查连接。它以指定容量<SIZE>的块来发送文件到PLC并等待接收回来。期望得到相同的包容量和相同的内容。

如果容量不同测试中止。

如果内容不同，发送内容与接收内容将被存储在两个文件中，以便比较并检测错误。

这些错误在所有的测试中都以相同的方式处理。

在发送所有文件或50个块后测试成功完成。

4.2.4.2 测试 2

测试2检查PC方面带负载的SerReadComm函数。它发送数据块，此数据被平分成2个包。在两个包的传输期间PC将等待第三个参数<lowTO>指定的时间。函数SerReadComm（PLC方面）设置为默认超时。它必须足够快地接收两个包，并把它们做为一个包来处理，然后以一个包回发。PC检查接收到的包的容量和内容。

错误的包容量（测试失败）可能是由发送回两个包造成的。

如果整个文件或50个块的传输都没有错误，那么测试成功完成。

4.2.4.3 测试 3

本测试和测试2相似，除了两点不同：

延时由第四个参数<highTO>指定。

PC期望接收到两个包，而不是一个。

错误的包容量（测试失败）可能是由发送回一个包造成的。

如果整个文件或50个块的传输都没有错误，那么测试成功完成。

4.2.4.4 测试 4

测试4检查SerSetTieout的执行，特别是第一个参数——内部超时。

PC通过一个包发送被分成若干块（由块容量指定其容量）的文件（没有负载）。PLC接收并回发。随着每个包的接收，PLC改变它的内部超时。PC检查到达的包并且标记丢失的块。

很可能有一块会丢失，因为最终这最后一个超时应该是0并且对接收一个大的块来说太短。

在PLC中的超时由ComRec.c文件开头的超时数组定义。你可自由改变它们的值。

它们的初始值是：

```
LZSINT timeouts[24] =
{1000,900,800,700,600,500,450,400,350,300,250,200,150,100,90,80,70,60,50,40,30,20,10,0};
```

通过丢失块号和超时数组你可以算出最低的内部超时。

4.2.4.5 测试 5

测试 5 与测试 4 相似，不同之处在于模拟一个 PC 负载，通过第三个参数<lowTO>设置（见测试 2）

测试在标记第一个丢失块后完成（内部超时）。

4.2.4.6 测试 6

测试 6 检查函数 SerSetTimeout 的第二个参数——第一个字节的超时。

PCL 设置这个超时，当 PC 没有发送任何数据时，PLC 试图接收一些更多的字节并且（应该）在超时后回发。

PLC 将配置的超时和 SerReadComm 的时间消耗回发给 PC。

超时存储在 ComRec.c 文件开始部分的 start_timeouts 数组中。

初始值如下：

```
int start_timeouts[19] =
{0,20,40,60,80,100,150,200,250,300,350,400,450,500,600,700,800,900,1000};
```

在 9 到 11 秒（PC 方面超时）后导致失败的测试。

成功的测试将显示两个时间的值（期望，测量）。

4.2.5 结束测试

测试结束后得到所有子测试的结果，所有错误的传输数据被存储到一些新增的文件中。

4.2.6 启用 ser.c

4.2.6.1 概述

Ser.c 包含了所有低级的函数，这些函数需要移植到你的串口端口。下列函数必需被执行：

LZSINT SerOpenComm(LZSINT idComDev)

LZSINT SerCloseComm(LZSINT idComDev)

LZSINT SerReadComm (LZSINT idComDev, char FAR * cpBuf, LZSINT cbRead)

LZSINT SerWriteComm (LZSINT idComDev, char FAR * cpBuf, LZSINT cbWrite)

LZSINT SerSetTimeout(LZSINT dwIntervalTimeOut_p, LZSINT dwStartTimeOut_p)

unsigned long TickCount(void)

4.2.6.2 SerOpenComm

LZSINT SerOpenComm(LZSINT idComDev)

SerOpenComm 从 RTS 调用初始化并打开串口来支持与 OpenPCS 编程系统的通信。

4.2.6.3 SerCloseComm

LZSINT SerCloseComm(LZSINT idComDev)

如果出现网络错误将调用 SerCloseComm，重新初始化网络。它关闭连接并由 SerOpenComm 来重启。

4.2.6.4 SerReadComm

LZSINT SerReadComm (LZSINT idComDev, char FAR * cpBuf, LZSINT cbRead)

SerReadComm 试图从接口读取”cbRead”中的字节并且写入 cpBuf。因此本处理应该反映出在函数 SerSetTimeout 最后一次调用时设置的超时参数。在 RTS 中有两种情况：

只通过输入字节检查串口——无超时

期望一定数量的字节（例如：一个数据段）——适当的超时参数

4.2.6.5 SerWriteComm

LZSINT SerWriteComm (LZSINT idComDev, char FAR * cpBuf, LZSINT cbWrite)

SerWriteComm 将数据”cbWrite”由缓冲区”cpBuf”中写入接口中。

如果超过内部超时所规定的时间将不能写入所有的 cbWrite 字节，函数将返回-1。

注：这个超时不由 SerSetTimeout 设置。选择任何值都符合。例如：通过传输多字节增加的基本超时。

4.2.6.6 SerSetTimeout

LZSINT SerSetTimeout(LZSINT dwIntervalTimeOut_p, LZSINT dwStartTimeOut_p)

SerSetTimeout 设置了用于调用 SerReadComm 的内部超时。

第一个参数（dwIntervalTimeOut_p）设置了到达的两个字节间的指定超时。如果两个字节间的到达时间超过了这个超时，SerReadComm 将停止并且返回到目前为止接收到的字节数。

第二个参数（dwStartTimeOut_p）设置了第一个字节的超时。如果在此时间内没有收到任何字节，SerReadComm 将返回 0（无数据接收）。如果 dwStartTimeOut_p 被设置为 0，SerReadComm 将立即返回，仅返回在第一个参数 dwIntervalTimeOut_p 定义的期间读取的字节数。

4.2.6.7 TickCount

unsigned long TickCount(void)

本函数仅用于 SERTEST_ADVANCED 测试时间和创建一些 PLC 负载。它必需返回系统时间（ms）。

4.3 Acrobat 阅读器

为了在线浏览.PDF文件，你可以在http://www.adobe.com/下载一个免费版的Adobe's Acrobat Reader for Microsoft Windows 32bit platforms

4.4 RS232 监听器

如果你没有特殊的设备来监听串口通信线路上的通信量，可试试工具"RS232"，由 Michael Ring (ringx004@tc.umn.edu)开发。你可以从任何方要共享软件档案那里下载到 RS232105.zip。"105"是版本号，所以如果你没能找到这个文件，试试更高级的版本号。我们对运行此工具有丰富的经验，但是不能给予授权。

4.5 串口通信库

如果你的平台没有适当的串口通信驱动，见 http://www.marshallsoft.com/。我们在使用他们的 PCL4C 库上有丰富的经验，但是同样不能给予授权。

4.6 初始模板

初始模板为 hardware.ini 文件中[FIRMWARE]部分提供了不同的模板。

对齐

字节顺序

字节容量

请为你的模板选择合适的一个并拷贝到 hardware.ini 文件中。

4.7 添加通用的驱动程序

4.7.1 AddDrvr.exe 的使用

通用 AddDriver 是 OpenPCS 编程系统的安装程序的一部分。在标准安装完成后将被立即调用。

OpenPCS 的 OEM 使用它来为他们的用户提供一个简单的方法，此方法通过用户的特定硬件的需求来定做 OpenPCS 安装。它允许 OEM 捆绑所有指定的硬件配置文件、驱动程序、示例、实用程序、帮助文件、许可证、库、模板、连接等到一个简单的机柜文件（.cab）中，这个文件与 OpenPCS 安装媒体一起装载或由 OEM 网址上的下载提供。

当没有任何参数调用时，AddDrvr.exe 打开并在文件夹"OEMCAB"中搜索*.cab 文件，这个文件夹在 OpenPCS-INI 路径下（例如：C:\Documents and Settings\All Users\Application Data\infoteam Software\OpenPCS2008\Openpcs.520\OEMCAB\])。具有选择性的 AddDrvr.exe 允许用户浏览包含*.cab 文件的目录。在"可选目标驱动器"对话框中选择一个或更多*.cab 文件并且点击安装按钮后，AddDrvr.exe 安装选中的驱动器，使用 OEM 中的.cab 文件提供的配置文件中的信息捆绑到当前 OpenPCS 的安装。

作为第三种可能，AddDrvr.exe 可以被一个特定的目录路径调用。实用程序搜索目录中的.cab 文件并且安排它们分批安装，这些安装可以通过"安装"按钮被用户触发。

注：如果其中有空白你必需装入一个带引用的路径或文件名，例如"C:\Documents and Settings\All Users\Application Data\infoteam Software\OpenPCS2008\Openpcs.520\OEMCAB\" 或 " C:\Documents and Settings\All Users\Application Data\infoteam Software\OpenPCS2008\Openpcs.520 \OEMCAB\DriverBundle.cab"

4.7.2 静音模式

现在可以将 AddDrvr.exe 运行在静音模式。因此你需要在 AddDrvr.exe 的调用前附加参数。你可以找到一个可用参数的清单：

<Filename> 给出了一个有效的*.cab 文件的文件名。它涉及到 AddDrvr.exe 启动的路径。

<Directory> 给出一个*.cab 文件存储的有效的目录/路径结构。

/s 静音模式（静音模式必需的）——没有对话框出现

/f 强制重写模式——每个已经存在的特性都将被重写。如果没有"/f"，已经存在并且完全相同的特性的拷贝将被丢弃。

4.7.3 创建一个 OEM 驱动程序包

4.7.3.1 先决条件

为了能够创建一个 OEM 驱动程序的机柜文件，OEM 需要一个小工具叫作 CABARC.EXE，可由 Microsoft 免费使用（搜索"Cabinet-SDK"）。这个工具也和 OpenPCS 兼容并且可在 OpenPCS 编程目录中找到。

4.7.3.2 CAB 文件内容

机柜文件的内容仅仅是一个嵌套的目录结构，这个结构构造如下：

```
hw.ini                          // details: Chapter   4.7.3.2.1
licences.txt                    // details: Chapter   4.7.3.2.2.1
MODULES                         // details: Chapter   4.7.3.2.4
        hardware1.ini
        HARDWARE1
            globdata.inc
            globprot.inc
            globtype.inc
            globwrap.inc
            errormap.ini
            errormap.xml
            CatalogCategories.xml
        hardware2.ini
        HARDWARE2
            globdata.inc
            globprot.inc
            globtype.inc
            globwrap.inc
            …
EXE
            …
DRIVERS                         // details: Chapter   4.7.3.2.6
        driver1.dll
        driver2.dll
        …
SAMPLES                         // details: Chapter   4.7.3.2.7
        INFOTEAM
        OPC1
            […Sample Files for OPC1…]
        OPC2
        PERFORM
        …
HELP                            // details: Chapter   4.7.3.2.8
        help.ini
        file1.hlp
        file2.chm
        file3.htm
        …
PRINTFORMS                      // details: Chapter   4.7.3.2.9
        prinform1.svg
        prinform2.svg
        image.gif
LIB                             // details: Chapter   0
        library1
            […library1 project…]
        library2
        library3
        …
```

```
TEMPLATES              // details: Chapter    4.7.3.2.11
        Projects
        OEM.xml
        OEM.ico
        oem
                ProjectType1.xml
            type1.ico
            ProjectType1
                ProjectType1.INI
                ProjectType1.VAR
            ProjectType2.xml
            type2.ico
            ProjectType2
                ProjectType2.INI
                ProjectType2.VAR
            …
```

4.7.3.2.1 hw.ini 文件

这个文件说明了*.cab文件的内容并通过AddDrvr.exe解析和解释。它控制OEM的特定安装进程。

示例：

```
[OEMDriverDisk]
OEMID=425
OEMNAME=infoteam Example-OEM
DESCRIPTION=Driver Bundle for infoteam CleverPLC
VERSION=5.1
MINIMUM_VERSION=5.0
REQUIRE_RESTART=1
[DRIVERS]
tcp822f.dll = noreg
tcp821a.dll = delete

[Samples]
SampleDir = infoteam
Replace = OPC1,OPC2,perform

[CONNECTIONS]
CONNECTION1=TestTCPIP_Con
CONNECTION2=TestRS232_Con
count=2

[TestTCPIP_Con]
DRIVERNAME=TCP432
DRIVERID=0400-05
TIMEOUT=10580
COMMENT=TestConnectionTCP
VALUES=TestTCPIP_VAL

[TestTCPIP_VAL]
IpAddress=178.34.123.44
PortNumber=24099
Motorola=

[TestRS232_Con]
DRIVERNAME=RS232
DRIVERID=0400-01
TIMEOUT=15000
COMMENT=TestConnectionRS232
VALUES=TestRS232_VAL

[TestRS232_VAL]
PortNumber=2
Baudrate=5
Parity=1
Stopbits=1
Protocoll=2
CRC=1

[TIMEOUT]
TCPIP=10000
[CUST-TOOLS]
FILE1=AnyApplication.exe
```

```
Title1=AnyApplication
Arguments1=
CmdShow1=1
InitDir1=
QueryArguments1=0

[VISIBLE_EXTENSIONS]
EXT=abc;xyz

[POST_COMMANDS]
COMMAND1=notepad.exe MyText.txt
COMMAND2=setup.exe $BinDir
COMMAND3=explorer "HELP\An Introduction.html"
COMMAND4=$BinDir\test.bat

[REGISTY_SETTING]
count=1
PATH1=PowerMap\\POWERTMAPENABLED
VALUE1=1
```

OEMID是通过infoteam分配的OpenPCS OEM的数字ID。

OEMNAME是一个清晰的文本类型的OEM公司名称。

DESCREPTION是通过AddDriver.exe显示的附加的文本。

VERSION指定了为OpenPCS创建的驱动程序包的OpenPCS版本。（5.2意味着"OpenPCS 2004 Version 5.2.X"）

MINIMUM_VERSION是驱动程序包所需OpenPCS的最小版本。

REQUIRE_RESTART是一个可选的参数。如果把它设置为1，将突然出现一个消息盒子建议用户重启OpenPCS。

在"Drivers"部分你可以为DRIVERS子目录中的驱动程序DLL文件指定参数。在默认情况下，在DRIVERS目录下的所有的文件被拷贝到OpenPCS的二进制目录下并且通过调用RegSvr32注册。"noreg"指定了相应的文件仅拷贝而不用注册。"delete"让用户在来自机柜的驱动程序被拷贝前，从这个二进制目录下删除一个存在的驱动程序。这允许一个OEM通过一个不同名称的驱动程序的更新替换另一个驱动程序。注意通过另一个OEM安装的已经存在的驱动程序仅能在用户确认后被重写。

"Sample"部分说明了附加的示例文件怎样被加入。"SampleDir"是在OpenPCS\ini\Samples目录下的目录的名称。我们建议使用你的公司或产品的名称，让用户从一般的示例或者另外的也可能出现在用户的工作PC上的OEM的示例中区分出你的示例。

"Replace"参数允许你指定一个逗号来分开被替换的示例目录的清单。注意你只可以替换使用相同的OEMID且使用AddDrvr.exe预先安装的示例。

"CONNECTIONS"部分说明了你想要自动加入到OpenPCS中的连接的名称和号码。对于每个给出的连接的名称都在标准属性（所有连接都有）中有一个新的部分，这些标准属性被设置为不同的值。"DRIVERID"属性包括了7位数字（4个数字，一个连字符，再接2个数字）。所以"DRIVERID"的格式如XXXX-YY，例如：对于TCP52驱动程序：0400-0A。此外有一个叫"VALUES"的属性有一个对深层部分的名称，在这个属性中不同的驱动程序指定的值可区别于另外的驱动程序。见Values部分。

有了"TIMEOUT"部分你可以指定tcpip驱动程序的超时。

"CUST-TOOLS"部分包含了在custtool.ini文件中设置新入口的值。这对菜单Extras\Tools\...有一个影响。

最重要的是这些属性FILEx和Titlex（x是一个以1开始的索引号）。这些属性必须存在。另一个属性在必要的时候可以设置或忽略。（请注意：在hw.ini文件中的所有入口是有大小写之分的！不要用file1代替FILE1,或用title1代替Title1！请使用4.7.3.2.1节中的示例清单中所示的书写习惯）。

AddDrvr.exe获得这些入口（如果不存在）并且附加到custtool.ini文件中。如果已经存在，将询问用户来决定重写或丢弃哪个入口。在静音模式取决于"/f"——标志——如果置位，入口将被重写。

"VISIBLE_EXTENSIONS"部分包含了文件扩展名，这些文件是你想在OpenPCS文件浏览器中可见的。在入口"EXT="后你可以加入由分号隔开的扩展名。

"POST_COMMANDS"部分包含了一些命令，这些命令在驱动程序机柜安装后执行。执行的命令相当于hw.ini文件中的命令并且与COMMAND之后的号码不相符。如果一个路径包含有空白，那么它必须被引用标志围绕。目前只有两种宏指令支持：

•'$BinDir'：包含到OpenPCS安装的路径（二进制文件）。这个路径可以由/i参数改变：AddDriver /i=C:\Program Files\infoteam Software\OpenPCS2008\

• '$AppDataDir'：包含到OpenPCS应用程序数据目录的路径。

"REGISTY_SETTINT"部分允许加入注册入口。

一个注册入口包括一个路径（见根目录infoteam Software GmbH\OpenPCS\x.y.z）和一个指定给它的值。

4.7.3.2.2 连接值

4.7.3.2.2.1 IPC

名称	类型	语意
Padt_SimulationUsed	String	可执行文件名称
Padt_SimulationEnvironment	Boolean	0->单一接口 1->多个接口

SmartSim.exe使用单个接口的示例:

```
[TestIPC_Sample_Val]
Padt_SimulationUsed=SmartSim.exe
Padt_SimulationEnvironment=0
```

4.7.3.2.2.2 TCP432

名称	类型	语意
IpAddress	String	IP地址格式xxx.xxx.xxx.xxx
PortNumber	Integer	端口
Motorola	Boolean	0->小端字节顺序 1->大端字节顺序

到端口23042的小端字节顺序局部连接示例:

```
[TestTCP432_Sample_Val]
IpAddress=127.0.0.1
PortNumber=23042
Motorola=0
```

4.7.3.2.2.3 TCP52

名称	类型	语意
IpAddress	String	IP地址格式xxx.xxx.xxx.xxx
ComputerName	String	网络中PC的名称
NameResolution	Boolean	0->通过IP解析 1->通过名称解析
PortNumber	Integer	端口
Motorola	Boolean	0->小端字节顺序 1->大端字节顺序

在服务器上与23042端口大端字节顺序连接示例:

```
[TestTCP52_Sample_
ComputerName=server
PortNumber=23042
Motorola=1
NameResolution=1
```

4.7.3.2.2.4 RS232

名称	类型	语意
CRC	Boolean	0->无CRC 1->CRC

USB	Boolean	0->无USB 1->USB
PortNumber	ID-Type	COM端口
Baudrate	ID-Type	波特率
Parity	ID-Type	奇偶校验
Stopbits	ID-Type	停止位数
Protocoll	ID-Type	协议类型l

ID-Type到它的值的解析:

名称→	PortNumber	Baudrate	Parity	Stopbits	Protocoll
ID	语意				
0	COM1	75	无	1	无
1	COM2	110	奇校验	2	Xon/Xoff
2	COM3	134	偶校验		Hardware
3	COM4	150			
4	COM5	300			
5	COM6	600			
6	COM7	1200			
7	COM8	1800			
8	COM9	2400			
9	COM10	4800			
10	COM11	7200			
11	COM12	9600			
12	COM13	14400			
13	COM14	19200			
14	COM15	38400			
15	COM16	57600			
16		115200			

COM2、9600波特率、无奇偶校验、1位停止位、协议类型为“无”且需要CRC的示例：

```
[TestRS232_Sample_Val]
PortNumber=1
Baudrate=11
Parity=0
Stopbits=0
Protocoll=0
CRC=1
```

4.7.3.2.3 许可证文件

在文件licences.txt中提供了你的用户的许可证。这是一个简单的文本文件，一行表示一个许可证。这些数字通过一个简单的制表符（ASCII 0x9）与许可证钥匙分开。所有用于注册OpenPCS的许可证都在这里。

示例（下列许可证是不可用的）

425L65531 0902-ALL-E7U53R-0000-7N51BX1

425L65532 1203-86-S7W3W0-0000-8X8UT00

425L65533 0404-UCD-E5UA3R-0000-7NF18X1

4.7.3.2.4 模块文件

所有你的厂商指定的硬件的模块定义文件放在MODULES子目录下。每个模块定义文件（例如：myhardware.ini）需要有一个相同的子目录名（除开”.ini”扩展名）包含各种各样的全局申明文件（比如：

Globdata.inc,globtype.inc,errormap.ini等）。

4.7.3.2.5 本地代码编译器

为OEM指定模块提供的本地代码编译器和HwConfig-Dll放于驱动程序机柜的\EXE目录下。如果涉及到模块定义文件，他们将被拷贝到OpenPCS二进制目录下。

4.7.3.2.6 驱动程序

将新增的驱动程序（例如：连接驱动程序）放入你的目录结构下的DRIVERS子目录中。在这里的每个文件都被拷贝到OpenPCS二进制目录下并且使用RegSvr32注册除非你在hw.ini中规定不要这么做（见4.7.3.2.1）。

4.7.3.2.7 示例

SAMPLES子目录包含一个示例文件的目录结构，这些示例文件是你想要加入到OpenPCS的示例目录上的文件。你的所有的示例通过一个简单的子目录分组，分组的名称可以是你的公司名或产品名。这样就避免了与其它OEM或一般示例的名称冲突。在这个子目录中你可以放入多个包含有你的示例代码的子目录。见一些infoteam的一般示例中的示例目录。

4.7.3.2.8 附加的帮助文件

放入任何数量的.chm,.hlp或.htm(l)文件到HELP子目录下。这些文件将被合并到OpenPCS的在线帮助系统中。此外你需要提供一个名为help.ini的文件，同样放在HELP子目录下，这个文件标记了每个帮助文件在在线帮助树上的顶端节点的内容提要。它是一个简单的Windows.ini文件，包含了”Helpnodes”部分。每个帮助节点提供了到根帮助文件的连接并且与OpenPCS帮助窗口中的说明一致。

它包含了一个可选择的部分”DeleteBefore”。每个以前在这里列出的安装文件在安装完成后将从你的帮助根目录中移除。

示例：

```
[Helpnodes]
file1.hlp=OEM’s Windows Help File
file2.chm=OEM’s Compressed Windows Help File
file3.htm=Online Help in HTML format
[DeleteBefore]
File4.gif
```

为了创建一个像infoteam一样的开始页面，用htm(l)文件当作帮助文件是可以的。也可用其它的支持文件连接的帮助文件格式。

在驱动程序安装完帮助文件后将被放于下述位置

[CommonAppDataFolder]\Webhelp\[OEMID]

这就是你的帮助根目录。

4.7.3.2.8.1 加入特定帮助块

为了给POU目录或CFC（见OpenPCS用户手册）提供特定帮助块，你需要加入一个XML文件到你的帮助根目录。XML文件名称必须与你的根帮助文件一致（见4.7.3.2.8），如：根文件openpcs.htm可作为openpcs.xml文件。

你想要为每个块提供一个特定的帮助，这需要在XML文件中有一个<Block>的入口。”Name=”属性当做是块的名称并且在”RelativePath=”中给出的HEML文件用做这个块的帮助。

Xml文件结构：

```
<Help_Pattern Version="1.0" HelpNodeOpenPCS="OpenPCS Help" Company="infoteam"> <FileList>
        <File id="1" Name="abs.htm">
                <BlockReferenceList Identifier="FirmwareBlock">
                    <Block id="1" RelativePath="\referenc\keywords\abs.htm" Name="ABS" Anker="" Description=""/>
                </BlockReferenceList>
    </File>
        <File id="2" Name="acos.htm">
                <BlockReferenceList Identifier="FirmwareBlock">
                    <Block id="1" RelativePath="\referenc\keywords\acos.htm" Name="ACOS" Anker="" Description=""/>
                </BlockReferenceList>
    </File>
        <File id="3" Name="add.htm">
                <BlockReferenceList Identifier="FirmwareBlock">
                    <Block id="1" RelativePath="\referenc\keywords\add.htm" Name="ADD" Anker="" Description=""/>
                </BlockReferenceList>
    </File>
    …
    </FileList>
</Help_Pattern>
```

4.7.3.2.8.2 额外帮助文件示例

假设函数块已经加入到OpenPCS中并且他们的帮助文件位于[OpenPCS folder]\Webhelp\YourHelp\，这个目录可作为你的帮助根目录。根文件叫做”yourhelp.htm”。为了能够通过双击函数块来打开各自的帮助文件，需要在帮助根目录中创建一个”yourhelp.xml”文件。对于一个叫yourBlock的函数块，它的帮助文件位于block/yourbloc.htm，xml文件如下所示：

```
<Help_Pattern Version="1.0" HelpNodeOpenPCS="OpenPCS Help" Company="yourCompany"> <FileList>
        <File id="1" Name="yourbloc.htm">
            <BlockReferenceList Identifier="FirmwareBlock">
                <Block id="1" RelativePath="\block\yourbloc.htm" Name="yourBlock" Anker="" Description=""/>
                </BlockReferenceList>
        </File>
    </FileList>
</Help_Pattern>
```

4.7.3.2.9 打印格式

PRINTFORMS子目录包含了所有用户需要的文件的打印方式。OpenPCS只支持可伸缩向量图形（SVG）作为打印格式。在打印格式中涉及的每个图像必须在相同的文件夹或文件夹下面。在驱动程序安装完成后这些文件将被放在下述位置：

[OpenPCSINIFolder]\Printforms\[OEMID]

这就是你的打印格式根目录。

4.3.7.2.9.1 创建打印格式

为了创建一个用户打印格式，需要一个另外的编辑器。这个编辑器必须支持以SVG V1.1兼容格式的形式的存储（例如：inkscapev0.46或更高：http://www.inkscape.org）。

目前支持线条、矩形、外部图片和文本（详见下节）。要打印文件的打印区域通过一个标记指定ID的参数”printarea_”定义。

注：如果可以，文本要以简单的SVG格式存储。我们不对外部的工具承担任何责任。

4.7.3.2.9.2 SVG元素

下表中列出了支持的SVG元素和它们支持的参数。不支持任何的转化标志、不透明的或线性类型。

元素	SVG-标志	描述
线条、轨迹	PATH	可以通过封闭轨道创建多边形，但是只支持连续的线条（没有弯曲） 注：最高层的元素可用轨迹创建。 支持的参数有笔画厚度、笔画颜色、填充颜色。
矩形	RECT	只支持有角的矩形。要求一个打印的矩形必须在ID中标记”printarea_”以定义打印区域。打印区域矩形本身是不打印的。 支持的参数有笔画厚度、笔画颜色、填充颜色。
图片	IMAGE	图片仅通过连接嵌入。使用相对路径（inkscape v0.46存储图片的相对路径，如果它在相同的文件夹中或文件夹下面）。
文本	TEXT	使用的字体类型需要安装到相应的计算机上。在宏指令中加入文本是可以的（见4.7.3.2.9.3） 支持的参数有字体家族、字体大小、字体颜色、字体形式。

注：SVG使用的单位（px）被设定与90dpi(90px=1英寸)一致。这也通常是默认值。

4.7.3.2.9.3 宏指令

使用一些宏指定嵌入到文本元素中是可以的。这些指定必须在#-character和$-character之间嵌入。目前支持下列宏指令：

宏	描述
Page	当前的页码 页码属于当前打印的章。 有内容的表格被以罗马字母”i”开始的字母编号。然后从1开始的程序按照相互对照表和索引打印
Maxpage	当前打印的章的页码

Sheet	目前全部文档(*)的工作表号 工作表号对于被打印的章是独立的。是来自有内容的表格的从1开始的连续的数字。
Maxsheet	全部文档(*)的工作表号
Date	当前数据
Datels	当前打印文件最后一次存储的数据
Time	当前时间
Timels	当前打印文件最后一次存储的时间
Compound	当前打印工程的图表名称(**)
POU	当前文件的POU名称
Path	文件的绝对路径
Project	工程名称
Type	POU类型（程序、函数、函数块等）
Root	当前打印工程的路径
Logo	在SVG打印格式中使用#logo:myLogo$并定义别名myLogo(testlogo.gif).使用的文件(testlogo.gif)必须与打印格式在相同的目录下。

(*)有内容的表格仅以CFC格式打印。”sheet”用于CFC区分章节明智的编号和连续的编号。对于其它的语言，”sheet”等同于”page”。

(**)仅用于CFC

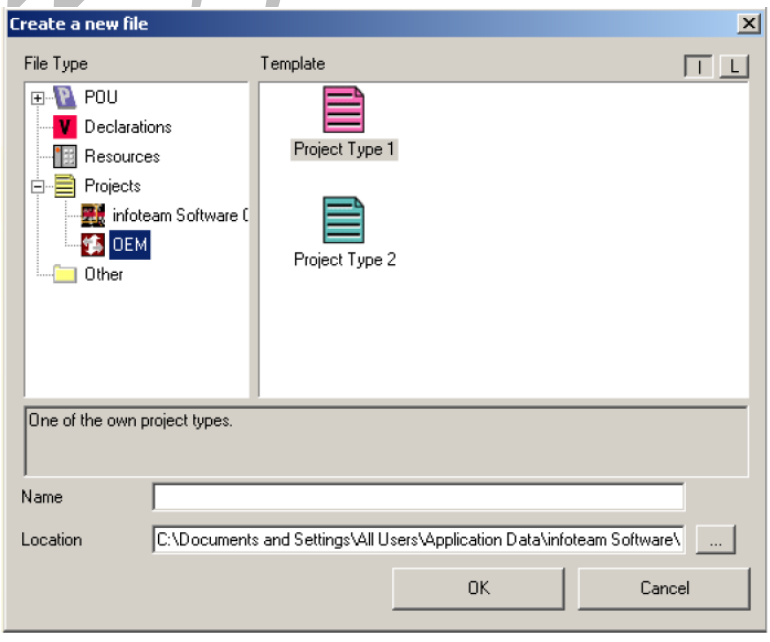
4.7.3.2.10 库

用你的cab文件自动地安装库是可行的。为了这样做需要将工程放在LIB子目录下。他们会以库的形式安装并被拷贝到OpenPCS库目录下。

为了加密你的工程/库，要将注册钥匙“…\Settings\OEMMODE”设置为”1”。现在可以通过位于文件窗口的工程树根项目的环境菜单加密一个工程。另一种方法是使用File→Project→Encrypt/Decrypt应用程序菜单项实现这个功能。在即将出现的对话框中，需要指定一个密码。在读取或更改IEC资源代码时需要输入这个密码。

4.7.3.2.11 “创建新文件”对话框中的EML说明文件模板

从OpenPCS5.2.0以后的文件都基于模板来创建。对于任何在File → New对话框中的文件，在OpenPCS文件系统结构中一个物理的表征是可以的。对话框被分为两个窗口，左边的树窗口和右边的文件窗口。在左边显示了目录。目录以集的形式说明，因为他们可以包含其它的集或文件模板，这些模板显示在右边的窗口中。



为了加入一个新的模板，创建一个空文件或工程作为想要的文件类型或工程类型的样板。同样需要创建一个XML文件来说明空模板的位置并且叙述模板的信息和一个在File → New对话框中的表征的图标。模板文件必须是

可写的。

下面是不同的XML标签和三种在括号中可能的值：

CAPTION

对话框中显示的名称

DESCRIPTION

如果模板被选中后，对话框中显示的说明

IECLANGUAGE

如果模板类型是POU的时候的IEC语言的说明

PAGEFORMAT

用不同页面格式使用的CFC平面图。

MAKEFILESECTION

如果某种模板文件的类型可以连接到生成文件，那么你可以在这里指定一个区段

PRIORITY

通过优先级，可以指定在图标中和对话框中树视图中出现的命令。

ICON

图标的路径，在创建新文件对话框是使用。

FILE

模板文件路径

RETURNTYPE

如果你申明了一个IEC6-1131函数你需要将这个属性置为1

FILETYPE(CONTAINER/PROJECT/POU/RESOURCE)

CREATIONMASK

仅在为POU子目录中组合FILETYPE CONTAINER使用。它强制用一个单选框模型的建立来替代在图标视图中显示文件。

FIXSUBDIR

为适当的类型指定一个固定子目录

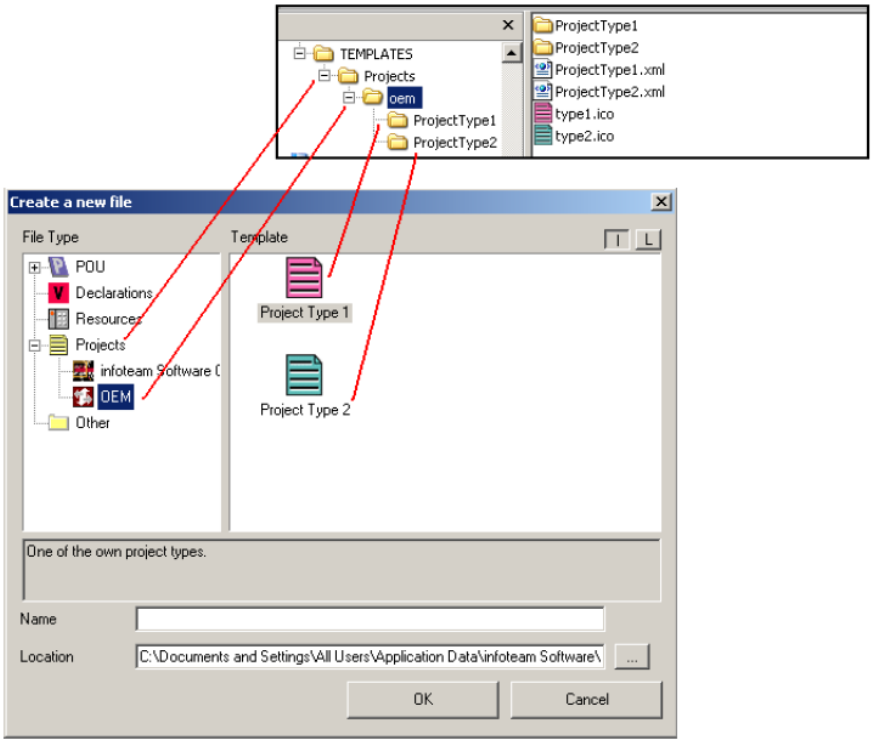
工程顶层入口（OEM.xml）的示例：

```
<TEMPLATE>
    <DESCRIPTION>Project types of the own company.</DESCRIPTION>
    <FILETYPE>CONTAINER</FILETYPE>
    <ICON>OEM.ico</ICON>
    <CAPTION>OEM Templates</CAPTION>
    <FILE>oem</FILE>
    <PRIORITY>1</PRIORITY>
</TEMPLATE>
```

模板工程（ProjectType1.xml）示例：

```
<TEMPLATE>
    <DESCRIPTION>One of the own project types.</DESCRIPTION> <FILETYPE>PROJECT</FILETYPE>
    <ICON>type1.ico</ICON>
    <CAPTION>Project Type 1</CAPTION>
    <FILE>ProjectType1</FILE>
    <PRIORITY>2</PRIORITY>
</TEMPLATE>
```

在TEMPLATES文件夹下的文件夹和文件的结构应于下图所示的File → New对话框中的目录、子目录和模板的结构一致。



4.7.3.3 打包 CAB 文件

当你按照上述说明设置好目录结构后你可以通过Microsoft的CABARC.EXE捆绑机柜文件。请确认使用PATH环境变量或者指定的目录可以找到它。然后更改你的目录结构中的根目录和类型

CABARC –r –p N mypackage.cab *.*

（”N”意味着“创建一个新的cab文件”，-r告诉内部命令在子目录中递归地加入文件，-p安排内部命令保存信息的路径，注意这是必须的！）

4.7.3.4 OEM 驱动程序机柜示例

在你的嵌入式SmartPLC实时系统安装目录的“Tools\OEM Driver Bundle”下有一些OEM驱动程序包的例子。每个包都有一个详细的readme文档。查看其中的机柜从CabinetSDK来使用EXTRACT.EXE（见4.7.3.1节）。

4.8 安装选项

4.8.1 安装选项

OpenPCS 安装程序提供了不同的可能性来运行一个用户自定义的安装——特别是在静音模式运行完整的安装时。下列设置示例(控制参数)列于模范安装文件 PS542e.exe 中。用你的安装文件名(版本)替换 PS542e.exe。

PS656e.exe /s /v/qn

安装运行在标准静音模式。除了 AddDriver 的对话框没有别的对话框出现，AddDriver 对话框出现在安装结尾。

PS656e.exe /s /v"/qn SHOW_LICENCE_DIALOG=0

SHOW_ADDDRIVER_DIALOG=0"

安装运行在静音模式。没有对话框出现。AddDriver 和 LicenceEditor 都不会启动，所以它们的对话框不会出现（在 OpenPCS 后面的版本默认情况下 LicenceEditor 都不会再出现）

PS656e.exe /s /v"/qn SHOW_LICENCE_DIALOG=0

RUN_ADDDRIVER_SILENT=1"

安装运行在静音模式。没有对话框出现。LicenceEditor 不启动所以不会出现对话框。AddDriver 功能在静音模式下启动——没有对话框并且它将安装所有在绝对路径”\Drivers”中列出来的*.cab 文件。

PS656e.exe /s /v"/qn SHOW_LICENCE_DIALOG=0 RUN_ADDDRIVER_SILENT=1

RUN_ADDDRIVER_FORCE=1"

安装运行在静音模式。没有对话框。LicenceEditor不启动所以不会出现对话框。AddDriver功能在静音模式下启动——没有对话框并且它将安装所有在绝对路径"\Drivers"中列出来的*.cab文件。此外，AddDriver将重写所有需要被安装但是已经存在的文件。

4.8.2 卸载选项

同样可用一行命令卸载 OpenPCS。安装文件的卸载部分必须与安装部分完全相同。

PS656e.exe /v"/qn REMOVE=ALL"

卸载 OpenPCS。安装程序窗口都出现。在卸载结尾，出现一个对话框询问是否删除所有安装目录中的内容。这只影响到用户在安装后创建的文件，例如：新增的硬件配置文件、库或对示例工程的更改。无论如何在安装期间创建的所有的 OpenPCS 文件都将被删除。

PS656e.exe /s /v"/qn REMOVE=ALL"

卸载运行在静音模式。安装程序窗口不出现。在卸载的结尾，出现一个对话框询问是否删除所有安装目录中的内容。

PS656e.exe /s /v"/qn REMOVE=ALL SHOW_CLEANUP_DIALOG=0"

卸载运行在静音模式。安装程序窗口不出现。询问删除安装目录的对话框不出现并且不删除安装目录。

PS656e.exe /s /v"/qn REMOVE=ALL SHOW_CLEANUP_DIALOG=0
RUN_CLEANUP_SILENT=1"

卸载运行在静音模式。安装程序窗口不出现。询问删除安装目录的对话框不出现并且自动删除安装目录。

4.9 永久性文件生成器

OpenPCS 永久性文件创建程序 padtrtps.exe 创建一个*.pcd 的永久性文件。这个永久性文件使用与标准实时系统中的序列化程序一样的格式。这个文件可以通过 OEM 功能直接地加载到标准实时系统中。实时 OpenPCS 生成器可以转换存在的*.pcd 文件。

永久性文件生成器可以通过 ini-File（见 5.9.6 节）中的 Postbuildstep 调用。

使用 6.4.0 版本以前的 OpenPCS 是不可以使用结合在线连接功能的永久性文件生成器。在 6.4.0 版本以后两个功能是兼容的。

与当前 RTS 对应的永久性文件生成器位于 tools\padtrtps 目录下。RTS 的版本信息嵌入在文件名称中

注：这个功能只能由特定 OEM 使用，这些 OEM 准备好了处理这些永久性文件格式的实时系统。如果你的实时系统没有完全准备好，请不要使用这个功能。

用法：（命令行）

padtrtps.exe <outputfile> -pcd=<pcdfile>

例：

padtrtps.exe proj\$GEN$resource\res.prs –pcd= proj\$GEN$resource\res.pcd

4.10 LicChk5

在 OpenPCS5.1 版本中介绍了一种新的许可证模式，这种模式在 OpenPCS 编程系统中执行。当 OpenPCS 启动时将为你的硬件驱动程序检查许可证，这个硬件驱动程序是你允许使用的。所以如果有一个新安装的驱动程序，许可管理系统将将用 infoteam 提供的许可证标记这个硬件。你可以在 OpenPCS 输出窗口看见一些状态信息。在编辑或在线程序启动前，这个签名将被检查并且授权此功能。一个 OEM 可以通过 AddDriver 实用程序安装一个许可证在用户的 PC 中（见 4.5 节）。

5 后台构架

5.1 构架概述

5.1.1 组件概述

OpenPCS 包含了一些虚拟处理工具，这些工具以模块化的方式提供了 PCS 编程系统的功能。这些组件是可用的

- 工程浏览器
- 编辑器（程序编辑器和变量编辑器）
- 应用文档
- 编译器和代码生成器
- 测试和试运行

5.1.2 组件功能

5.1.2.1 工程浏览器

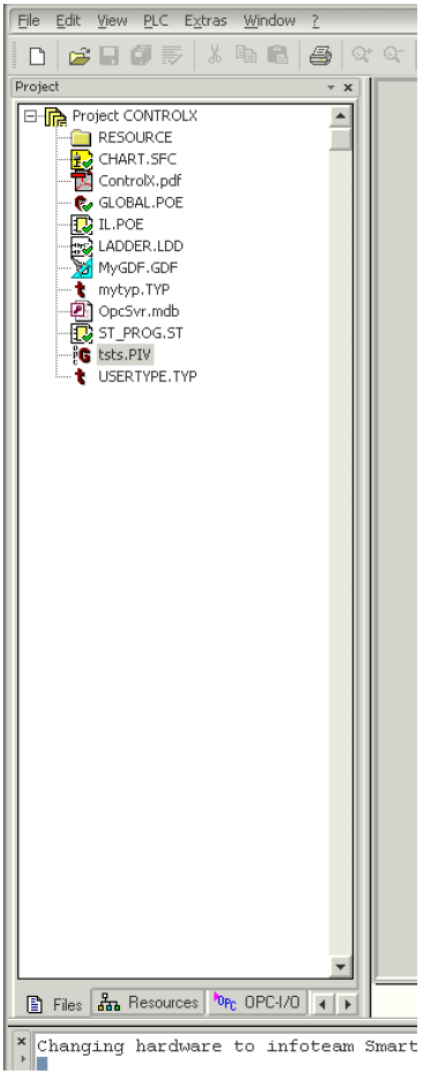


图 20 工程树

工程浏览器是一个方便的图解工具帮助你管理一个工程的程序、函数和函数块。通过工程浏览器用户可以调用 OpenPCS 的不同的编辑器。工程浏览器管理和监督这些工具的使用。

工程浏览器不但生动地显示了应用程序的结构而且显示了基于 IEC1131-3 的工程结构。

工程浏览器也创建和构造工程。有许多一般的文件管理程序可用。一个工程可在工程浏览器内部构造。

5.1.2.2 编辑器

这是一个 OpenPCS 提供可互换的编辑器的基本原理。这个灵活的架构基于一个基础编辑器，这个基础编辑器提供了系统的剩余端口的接口。

改变每个编辑器的显示方式是可以的。对于变量编辑器也是一样的。

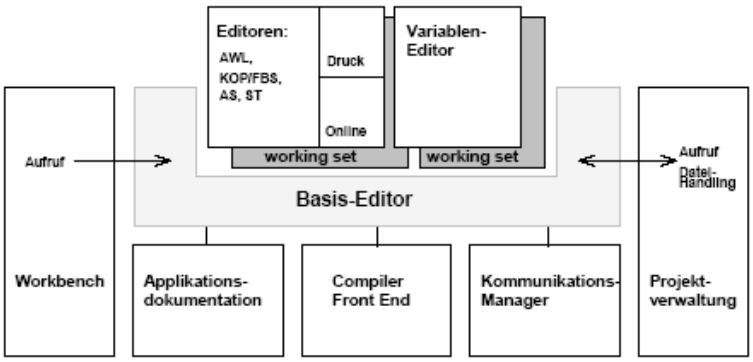


图 21 OpenPCS 编辑器

通常，程序编辑器将 POU 保存于 IEC-IL 格式。一些编辑器保存附加的信息到一个分离的文件，像 CFC 编辑器，SFC 编辑器和 LD 编辑器。对于 ST 也是一样的。因此，ST 声明通常解释成 IL 并且在资源代码中把原始声明转换为注释。所以，当测试的时候用户可以很容易的看见 IL 行与 ST 声明之间的对应。

5.1.3 编译器

代码生成分为两个阶段：前端编译（CFE）和后端编译（CBE）。这将从硬件从属的部分（CBE）分离出硬件独立的部分（CFE）。通过 CFE 生成的中间代码可能被一些用户用于指定 CBE 的变量。

CFE 执行下列步骤：

- 执行 POU 的语法和语言检查
- 生成错误和警告报告
- 生成中间代码
- 生成交叉连接清单
- 创建一个 POU 原型。然后应用环境就可识别此 POU。

CFE 是非常快的，因为前端被连接器分离并且此外还受限于一个简单的 POU。注意为了有一个快速的语法检查，在编辑器中可使用热键。

在连接程序之前 CFE 也要检查 FB 参数类型的一致性。一个广泛的系统原型清单可使之成为可能。

CFE 充分地编译 IEC1131-3（语法和主义）。用一个合规列表来证明它。

变量标识符可是任意的长度。但是，只有前 32 个字母是有效的。

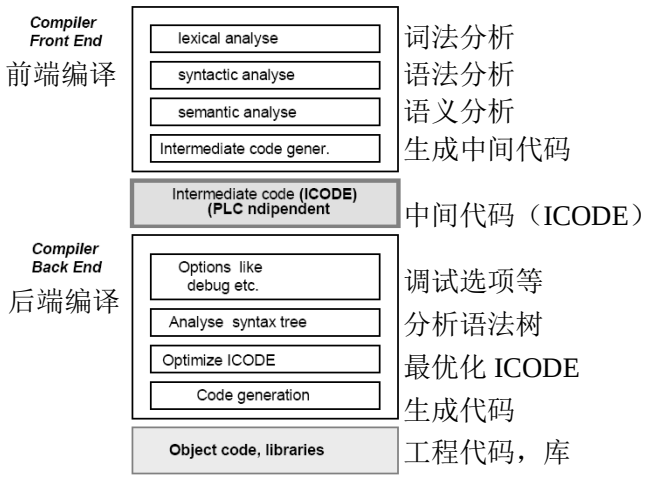


图 22 编译步骤

5.1.4 测试和试运行

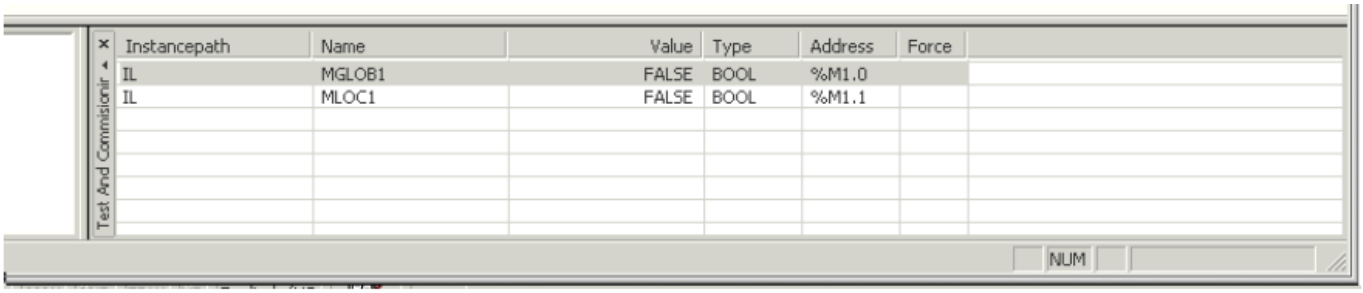


图 23 测试和试运行

测试和试运行工具支持用户查看和调试运行中的 IEC 程序。

主函数将被查看、设置和强制变量。

支持 3 种不同的启动程序。冷启动将清零所有的变量；温启动将清零所有的无保持变量；热启动将保持所有变量的值。

5.1.5 应用程序文档

为了保证所有的 OpenPCS 工具拥有统一的文档，OpenPCS 提供了创建 WMF 格式的用户定义模板的能力。打印模式关注第一行、最后一行、页数、数据等。并且由用户指定页面布局。

你可通过形式编辑器（form editor）帮助创建页面布局，这个图表的程序支持 WMF 格式。这个布局可以包含分离的线、标志或其它元素。

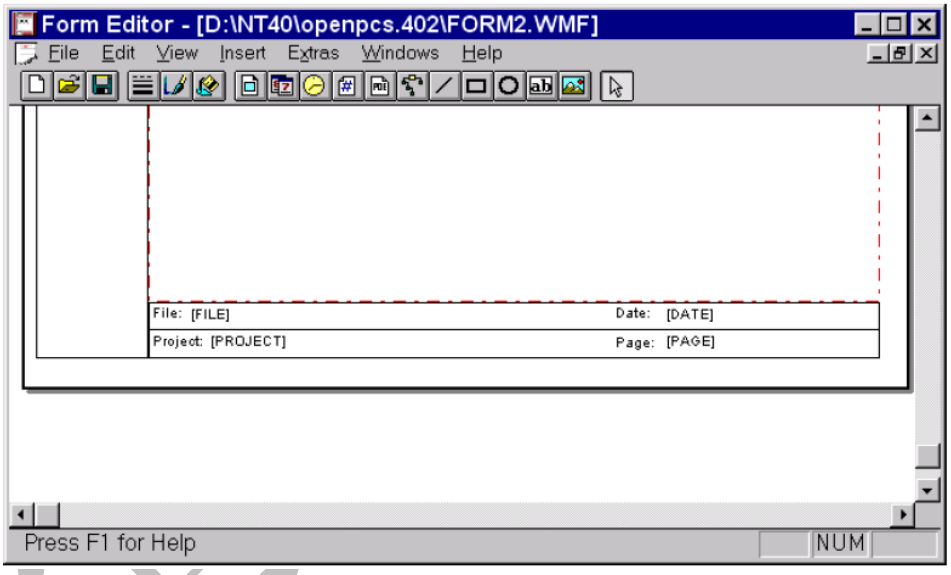


图 24 用户文档形式编辑器

你可以插入时期、页码、版本、CRC、函数块名等。当然，需要指定打印范围。

在工程浏览器中，这个有效的形式将被工程广泛地定义。因此，所有的应用程序在打印的时候都可以使用这种形式。

5.2 进程映像布局

OpenPCS 在默认情况下使用一个进程映像。这个进程映像有如下布局：

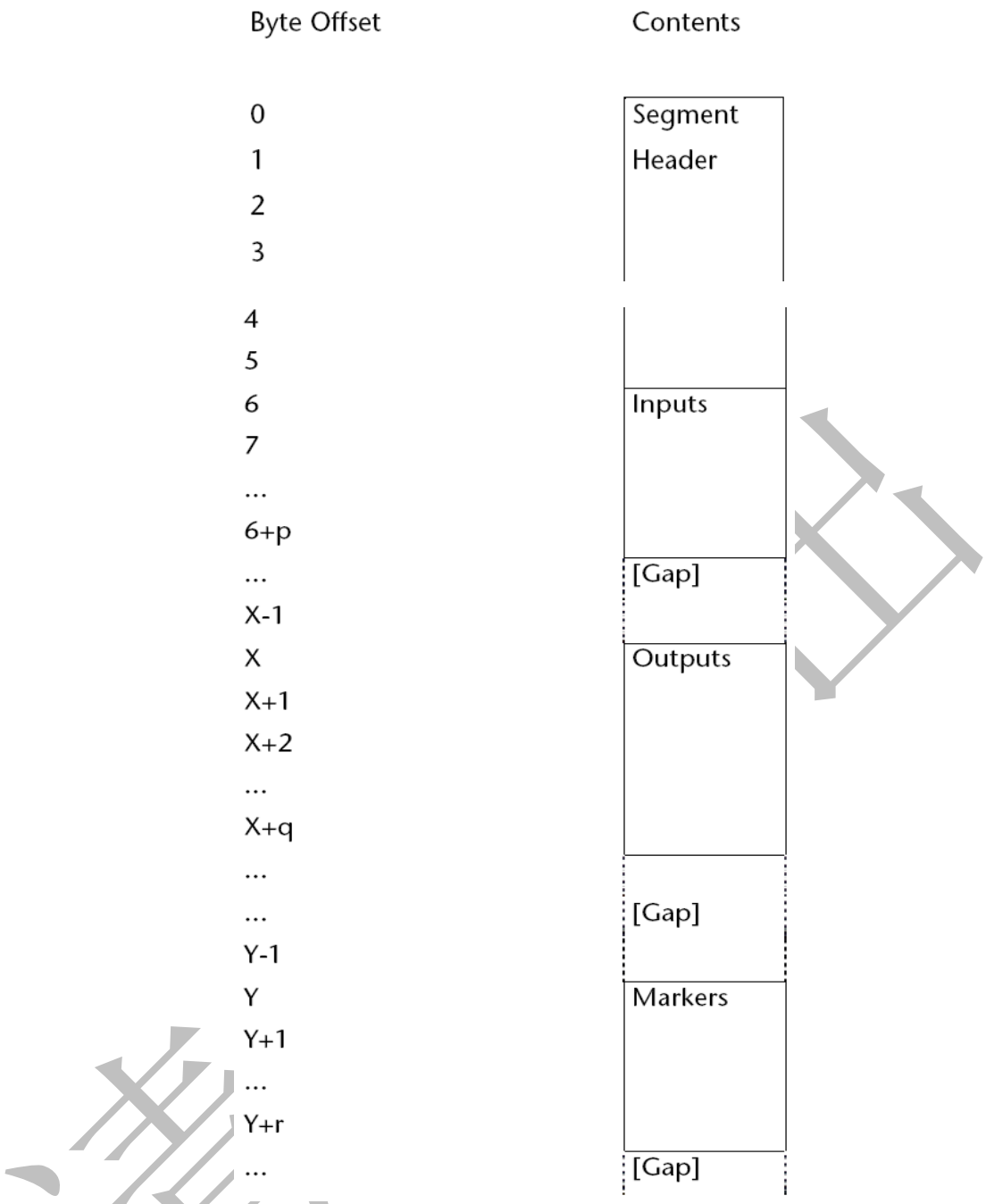


图 25 进程映像布局

这个进程映像布局，像任何一个段一样，有一个段头。在段头后是用于物理输入（%I）的存储空间，之后是物理输出(%Q)的存储空间和标记符(%M)的存储空间。在输入与输出之间和输出与标记之间，标记之后，是”gaps”（空白）的不用的存储空间。

进程映像的布局实际上是被实时系统定义的，所以请确认首先正确的执行了你的 LzsEnvReadProcImg 和 LzsEnvWriteProcImg 函数。

然后在硬件配置文件中描述进程映像布局，如：SMARTSIM.ini。下列 Smartsim.ini 的入口描述了这个布局。

a)OffsetIn 是%I 空间的偏移，从段的开头以字节为单位计算。%I 空间通常是进程映像的开头，等于段头的容量 6。

b) OffsetOut 是%Q 空间的偏移,从图 25 的段的”X”开始并以字节为单位计算。

c) OffsetMarker 是%M 空间的偏移,从图 25 的段的”Y”开始并以字节为单位计算。

d) InBits 是以位为单位的%I 区域的容量，例如：在图 25 中物理输入的总位数为”8*(p+1)”。

e) OutBits 是以位为单位的%Q 区域的容量，例如：在图 25 中物理输出的总位数为”8*(q+1)”。

f) MarkerBits 是以位为单位的%M 区域的容量，例如：在图 25 中标记符的总位数为”8*(r+1)”。

5.2.1 警告

在 SMARTSIM.INI 文件中，所有的偏移量都是从段的开头开始计算的（包括段头），函数 LzsEnvReadProcImg 和 LzsEnvWriteProcImg 传递指针到进程映像的数据区域的开头（例如：不包括段头）。所以 SMARTSIM.INI 中指定的偏移量和 LzsEnv.c 中使用的偏移量必须匹配，但不必完全相同。

5.2.2 示例

作为示例，我们将为下列结构配置一个进程映像。

16 位数字输入

8 位数字输出

128 位标记符

没有保护内存的空白区域

通过图 25 所示的助记符，得出 X=8,Y=9,p=1,q=0,r=15。

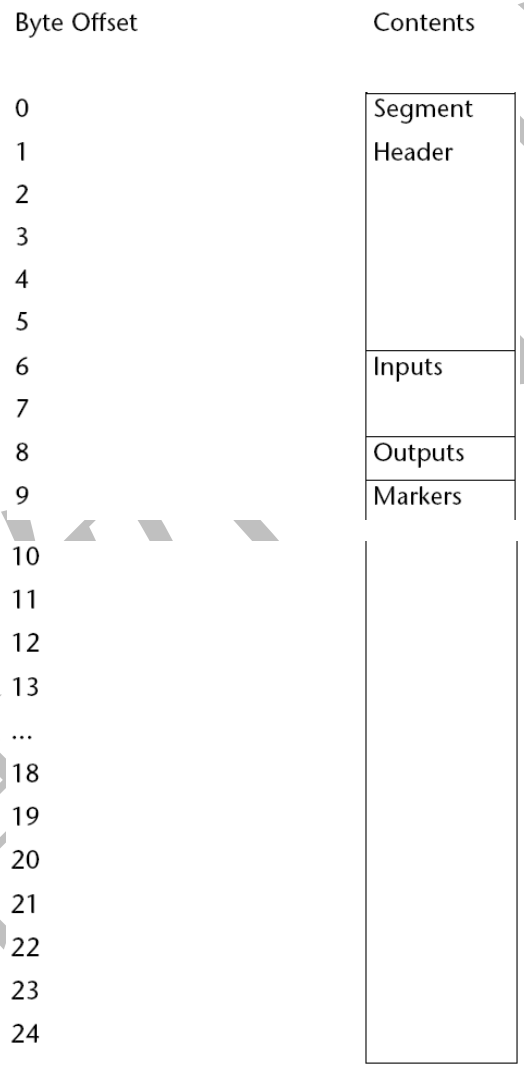


图 26 示例进程映像布局

在 SMARTSIM.INI 文件中得到如下结果：

OffsetIn=6

OffsetOut=8

OffsetMerker=9

InBits=16

OutBits=8

MarkerBits=128

图27 示例SMARTSIMINI入口

5.2.3 其它高级的性能

需要特别注意的是输入(%I)通常在进程映像的开头。实际上，这不是必须的，但是我们强烈推荐这样做。

进程映像被保存在一个段中，这个段的位置与其它实时系统的段一样。有些架构的进程映像保存在非OpecPCS提供的内存区域内，例如：在一个物理的确定的双端口RAM中。在这种情况下，OpenPCS即不用为进程映像分配空闲的储存空间，也不需要使用已存在的储存空间。在OpenPCS中这样是可行的，如果你需要请详询我们。

5.2.4 储存器直接地址映射

直接地址（如：%I4.2）在标准infoteam OpenPCS编程系统中的映射是通过smartplc.dll完成的。储存器地址由如下公式计算：

%[I|Q|M]BYTE.BIT

[OffsetIn|OffsetOut|OffsetMerker] + BYTE[BIT]

偏移量可在硬件配置文件中读取（标准：smartsim.ini）。

5.2.4.1 用户自定义储存器映射

为了用户自定义储存器映射方案，需要用像smartplc.dll文件中的一样的接口来写你自己的硬件配置DLL文件。因此smartplc.dll中的资源代码可作为一个模板来用。

你需要修改的函数是

```
tPadtFwError CPadtFirmwareBuilder::GetIOAddress(
    tPadtLocatedVariableDescription* pInputValues_p,
    tPadtLocatedVariableAddressInfo* pOutputValues_p
)
```

输入结构包含了关于你将要映射的变量的信息

```
typedef struct {
        int m_nSize;                                /*size of the structure*/
        const char* m_pszAddressDescription;        /* address string ie %I2.1*/
        const char* m_pszName;                      /* name of the variable */
        const char* m_pszType;                      /* type information */
        DWORD m_dwVarQualifier;
        BOOL m_fUserInitialization;
        WORD m_wCountInitialBytes;
        const BYTE* m_pbInitialBytes;
} tPadtLocatedVariableDescription;
```

你需要将地址信息填入输出结构中：

```
typedef struct {
        int m_nSize;
        WORD m_wSegmentNumber;      /*number of the segment for this variable*/
        WORD m_wOffset;             /* offset in this segment*/
        BYTE m_bBitMask;            /* which bit */
        BOOL m_fWriteAccessAllowed;
        BOOL m_fReadAccessAllowed;
        char m_pszErrorMessage[MAX_ERROR_MESSAGE];
} tPadtLocatedVariableAddressInfo;
```

当你把新的DLL文件的名称写入硬件配置文件中后，此DLL就可以被编译程序使用。

[Module] HConfigDll32=MyHw.dll

5.2.4.2 特定 OEM 段

OEM段是一种特殊的段，可被一个OEM特定配件配置DLL（如:smartplc.dll）文件创建。在实时系统中OEM段可在LzsEnvMoveSegment中处理（见samples\winipc\lzsenv.c）。类型为kPlcSegOemSeg。有个在SmartPLC.dll资源代码（背景：用户支持函数）中给出的示例，该示例描述了怎样产生这样一个OEM段并且为多重网络映像运行他们。在Visual C++中有两个叫做DebugMultiNet和ReleaseMultiNet的额外的配置来编译这个DLL文件。这些配置使用附加文件PadtOemSegments.cpp来执行OEM段和地址分配方案。在FwBuilder.cpp文件中查找_MULTIOEMSEGMENTS_定义的变化。在这个示例中OEM段包含了一些信息，这些信息被实时系统用于分配附加的网络映像。

SmartPLC.dll-Sample是一个网络程序映射的示例并且需要在硬件配置文件中有下列配置：

[OEMSEGMENTS]
SIZE=
SEGNR=
TYPE=
ADRESS=
PREFIX=
因此它要在硬件配置文件（如：smartsim.ini）中查找记录并且用下列信息创建OEM段：

Entry	Range	Comment
Size	0..64k	在目标系统中可创建的段的容量
Segnr	10..20	段号：段0到段9为infoteam预留
Type	0xD0..0xEF	段类型（由OEM定义）
Address	0..0xFFFFFFFF	段的32位地址，如果地址没有动态分配的情况
Prefix	A-Z	段中IEC-61131声明的前缘

在这些段下载期间LzsEnvMoveSegment函数将被调用，这个函数将查找分配真实网络映像段和将段写入段表的信息（见samples\winipc\lzsenv.c）。

这意味着这仅仅作为一个示例——怎样在目标系统中使用硬件配置DLL文件来修改地址分配方案和使用OEM段来创建新的段。

5.2.4.3 直接变量的对齐检查

自从OpenPCS5.5.0版本以后，SmartPLC.dll文件在默认情况下执行一个对直接变量的对齐检查。这意味着如果你有一个2或4的对齐，那么所有的直接变量必须正确的放置。例如：对于2的对齐，所有的WORD-和DWORD-的直接变量必须放于偶地址；对于4的对齐，所有的WORD-直接变量必须放于偶地址并且所有的DWORD-直接变量必须放于能被4整除的地址。

做这个检查的原因是一些设备会对未分配的直接变量进行错误的储存器存取。你可以通过在你的硬件配置文件（详见“硬件配置文件”节）中设置一个选项来取消这个检查。

5.3 自定义支撑函数

为了定做内存的映射、连接、任务原型和资源管理，需要像smartplc.dll一样用同样的接口写你自己的硬件配置DLL文件。因此smartplc.dll中的程序代码可在infoteam中作为一个模板使用。

硬件特定编译程序DLL实现一个对自定义支撑函数的接口，由编程系统调用。默认为SmartPLC.dll，在硬件初始化文件中的[MODULES]部分由HConfigDll32设置。

Infoteam OpenPCS使用了三个关键接口：固件管理接口、数据表示接口和资源管理接口。

5.3.1 输入/输出结构体

struct tPadtLocatedVariableDescription		
int	m_nSize	结构体大小
const char*	m_pszAddressDescription	寻址字符串，如：%I2.1
const char*	m_pszName	变量名
const char*	m_pszType	类型信息
DWORD	m_dwVarQualifier	变量修饰符，如：const 或retain
BOOL	m_fUserInitialization	为真时，用户不用初始化变量
WORD	m_wCountInitialBytes	初始值大小
const BYTE*	m_pbInitialBytes	初始值指针
char[128]	szTaskName	任务名

struct tPadtLocatedVariableAddressInfo		
int	m_nSize	结构体大小
WORD	m_wSegmentNumber	变量的段号
WORD	m_wOffset	变量偏移
BYTE	m_bBitMask	位偏移
BOOL	m_fWriteAccessAllowed	为真时变量是可写的
BOOL	m_fReadAccessAllowed	为真时变量是可读的
char[]	m_pszErrorMessage	如果出现错误，则提示错误消息

struct tPadtNetImageInfo		
int	m_nSize	结构体大小
WORD	m_wImageSize	网络映像段大小
WORD	m_wOffsetIn	输入网络偏移
WORD	m_wOffsetOut	输出网络偏移

5.3.2 IPadtFirmwareBuilder

IPadtFirmwareBuilder是对内存映射、段处理等的抽象接口，由连接程序调用。

5.3.2.1 GetIOAddress

为了定做直接全局变量的内存映射计划，你需要修改tPadtFwError GetIOAddress
(tPadtLocatedVariableDescription* pInputValues_p, tPadtLocatedVariableAddressInfo* pOutputValues_p)
输出结构体tPadtLocatedVariableDescription包含了你要映射的变量的信息。
你需要在输出结构体中填入相应的地址信息。

5.3.2.2 GetOemVarAddress

为了解析来自OEM段的变量，你需要修改tPadtFwError GetOemVarAddress
(tPadtLocatedVariableDescription* pInputValues_p, tPadtLocatedVariableAddressInfo* pOutputValues_p)
输出结构体tPadtLocatedVariableDescription包含了你要映射的变量的信息。
你需要在输出结构体中填入相应的地址信息。

5.3.2.3 GetFirmwarePou

为了定做固件块的映射，你需要修改tPadtFwError GetFirmwarePou
(const char * pszFirmwareName_p, IPadtFwCodeInfo** ppCodeInfo_p, IPadtFwDataInfo** ppDataInfo_p)
输入包含固件块的名称。
输出结构体需要填入相应信息。ppCodeInfo_p可以为空，但是ppDataInfo_p不可以。

5.3.2.4 GetReservedSegment

为了定做保留段你需要修改tPadtFwError GetReservedSegment
(unsigned long nSegmentNr_p, BOOL fResourceGlobal_p, IPadtBytePtr** ppSegBuffer_p)
依赖nSegmentNr_p和fResourceGlobal_p，你需要填入ppSegBuffer_p。

5.3.2.5 GetFirstFreeSegmentNumber

为了定做第一个没有被保留的段的段号，需要修改DWORD GetFirstFreeSegmentNumber()

5.3.2.6 tPadtFwError GetNetImageInfo

为了定做网络映像信息你需要修改tPadtFwError GetNetImageInfo(tPadtNetImageInfo* pNetImageInfo_o)
你需要在输出结构体中填入相应的网络映像信息。

5.3.2.7 GetPIInfo

为了定做pi映像信息，你需要修改tPadtFwError GetPIInfo(tPadtNetImageInfo* pPImageInfo_o)

你需要在输出结构体中填入相应的pi映像信息。

5.3.2.8 AssembleFwInstAddress

tPadtFwError AssembleFwInstAddress
(unsigned long nSegmentNr_i, unsigned long nExtention_i, IPadtBytePtr** ppAddressBytes_o);

5.3.2.9 GetOemSegment

为了创建一个OEM段，你需要修改tPadtFwError GetOemSegment(unsigned long nSegmentNr_i, unsigned int nSegmentKind_i, BOOL fResourceGlobal_i, IPadtBytePtr** ppSegBuffer_o, BOOL* pfLastOemSegment_o)

依赖nSegmentNr_p和fResourceGlobal_p，你需要在ppSegBuffer_p中填入初始数据。
如果没有多的OEM段，pfLastOemSegment_o应该被置为真。

5.3.3 IPadtRessourceBuilder

IPadtRessourceBuilder是一个为了编程系统配置资源和任务的虚拟接口。
关键功能是资源原型对话框和任务原型对话框。

5.4 实时系统

OpenPCS SoftPLC是基于OpenPCS的解释程序模式（见各自的技术论文）。解释程序中的每个实现都有SoftPLC的功能，这些实现相当于这个解释模式并且被封装进实时系统内部（LZS）。

实时系统包括：
底层：允许解释程序存取程序。
上层：允许对解释程序环境的存取。

从外部对解释程序的直接存取是不可以的。OpenPCS SoftPLC的功能与OpenPCS在线服务器很相似。因此，相同的接口函数不是偶然而是必然。

5.4.1 限制

LZS可以运行少数程序和一些函数，但是只能在工程层，不能在程序层执行（启动/停止，下载）。

5.4.2 命名规则

LZS的命名遵循OpenPCS的命名规则，例如：我们使用分层的组合的名称：所有的名称都以一个前缀”Lzs”开头，后面进一步跟一个首字母缩略词来表明其所属的函数组（Prj,Pgm,Ip,Mem…）。少数没有分入指定组的函数就没有前缀。下列为使用的前缀和后缀：

Prefix	Meaning
C	Class
t	Typedef
k	Enum constant
f	BOOL variable
p	Pointer
c	Character
m_	Member
ul	Unsigned Long (32bit)
us	Unsigned Short (16bit)

图28 命名前缀

Suffix	Meaning
_p	Parameters
_g	Global variable

图29 命名后缀

5.4.3 多任务

5.4.3.1 任务类型

目前有三种任务类型可供使用：

周期

计时器

中断

5.4.3.2 周期任务

当没有其它任务执行时实时系统将执行周期任务。可在工程浏览器的程序定义中为每个任务分配优先级。它告诉实时系统任务在哪个周期内执行，例如：优先级为1意味着每个周期，优先级为10意味着每10个周期。

5.4.3.3 计时器任务

计时器和中断任务的执行取决于你的实现。所以有两个函数需要加入到LzsIp.c文件中：

LZSPUBLIC32 LZSBYTE LZSPUBLIC LzsIpOnSystemTick(LZSDWORD dSystemTick_p)

和

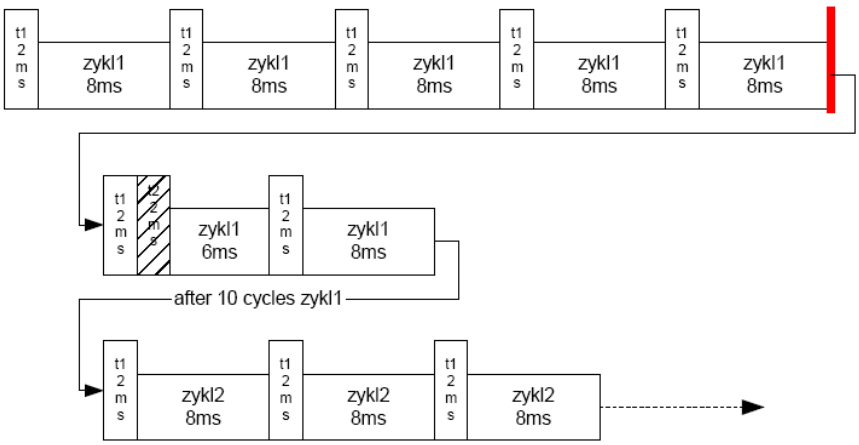
LZSPUBLIC32 LZSBYTE LZSPUBLIC LzsIpTriggerInterruptString(LZSCONST LZSCHAR * pszInterruptName_p)

在LzsIpOnSystemTick中由系统决定一个计时任务是否被执行。如果IpSetTimerTaskNow被调用来设置Status_g中的标志并且当前任务在当前U-Code后将被中断。LzsIpRunCycle将寻找等待中的计时器任务。如果有超过一个计时器任务被执行，它们将按优先级顺序开始。1是最高的优先级。在所有的等待中的计时器任务执行完后，被中断的周期任务将继续。

函数LzsIpOnSystemTick通过环境调用并存在时间间隔。参数是以ms为单位的当前系统时钟。如果这个函数找到了一个可以被执行的计时器任务并且这个任务是当前的任务，函数将返回1来标志错误，其外返回0。

作为一个示例资源有下表所示4个任务：

Task	time	priority
zykl1	needs 40 ms	1
zykl2	needs 20 ms	10
timer1	every 10ms; needs 2ms	1
timer2	every 50ms; needs 2ms	2



进程映像的改变可通过设置函数LzsIpRunTaskOnce的参数fExchangeProcImage来取消。

5.4.3.4 中断任务

如果你想要使用中断任务，你需要从你的中断服务程序中调用LzsIpTriggerInterruptString。你需要告诉这个函数需要被执行的任务的中断标志符。这个标志符必须与任务说明书中为这个在域”Nr.:”中的任务指定的标志符相同。

5.4.4 当前程序

作为LZS上层部分的函数，作为一个参数的程序号通常是被主张始终如一的，这个程序号指定了函数调用的一个引用。由于执行的原因，通常有一个“当前“的程序，引用所有在LZS的底层部分的调用。由于LZS的异步执行，需要注意同步部分（工作解释程序的部分）和异步部分（工作到编程系统的接口的部分）与当前程序号不在一边。

5.4.5 一般数据类型

C类型用于编址，取决于可以使用哪种范围的内存。为了在两种方向（存储容量，运行速度）间调节方便我们使用自己的类型来定义它。

```
typedef short int tLzsByteCount;                    // Memory Area Size, max. 64kB
```

图30 tLzsByteCount

对于版本号我们也使用自己的类型。为了检测在不同的计算机的相同的程序的后来的变化，我们把一个时间标记加入到增加的版本号中，这个版本号用来表示下载发生的次数：

```
typedef struct {
    short int m_Version;                    // incremented on each download
    short int m_Stamp;                      // Time-Tick of download
} tLzsVersion;
```

图31 tLzsVersion

5.4.6 故障处理

由于性能的原因LZS底层以另一种方式处理故障。在寄存器中装填”0”并且在每个内存中检查返回值，例如：在每个解释程序命令上，是不可忽略的开销。作为替代，我们为这种故障的情况设置了一个全局错误变量。故障处理仅需要在LZS的顶层。实际上在一个周期的开始初始化这个错误变量并且在这个周期结尾检查它是有必要的。

```
typedef enum {
        kLzsNoMem,                  // 下载期间内存溢出
        kLzsBadCode,                // 解释器报告坏的代码
        kLzsIPFail,                 // 解释器报告内部错误
        kLzsCrStack,                // CR栈溢出
        kLzsRetStack,               // 返回栈溢出
        kLzsCycleTimeOut,           // 溢出的最大周期时间
} tLzsErrorCode;
```

图32 TLzsErrorCode

5.4.7 LZS 段的架构

在OpenPCS中有一个对存储中的段的视图。这使得对代码的执行（连接、系统生成器）和T&C（程序下载、更新、在线更改）的实现更容易。本节给出了对使用的段和他们的一致性的一个简短概要。

段表：

LZS拥有一个表来控制，一个段号被分配给两个内存地址。也就是程序指定的段表（如：程序1的段号为0的段必须与程序2的段号为0的段区分开）。此外，没有涉及到特别的程序的段（如：任务定义段）被存储于程序号为0的段表中。

分配段：

段表中对于每一个段有两个地址；第一个是保存段的物理地址；第二个是叫做分配段的地址。在每个段的段头，列出了另一个段的段号。这就建立了一个基于段类型的段间的本地联系。见下表为哪个段分配哪个段类型。

段类型：

每个段都有一个确定的类型；这个类型可从段的Magic byte决定，并且存储在一个指定的位置。段类型被定义为枚举类型tLzsSegType并且有如下取值。

Type (Magic byte)	Purpose	assigned segment	DL[3]	Amount[4]
kLzsSegCode	一个块的代码段	kLzsSegInit	J	B
kLzsSegData	（可写）一个块实例的数据段	kLzsSe-gCode	J	I
kLzsSegInit	（只读）一个块的初始数据段		J	B
kLzsSegCr	当前结果栈段		N	1/P
kLzsSegRet	返回栈段		N	1/P
kLzsSegSource	原始段	none	O	Max. 1/P
kLzsSegSource2	带程序源码的结果段		O	n/P
kLzsSegTD	任务定义段	none	J	1
kLzsSegPI	进程映像		N	1
kLzsSegNI	网络变量存取段		N	1
kLzsSegNVMap	网络变量映射表		J	1
kPlcSegInstanceTable	创建的实例列表	none	O	1/P
kPlcSegDownloadTable	所有下载的POU的校验和	none	O	1/P
kPlcSegNativeCode	本地代码	kLzsSeg-Code	J	B

图33 段类型

[3]确定segment的内容是否会传到控制器中；J：segment的内容会传到控制器中；0：内容会选择性传递；n：只有segment的大小会传递。

[4]多少该类型的段会传到控制器中；B：每个块类型一个段；I：一个块的每个实例一个段；1/P：每个程序一个段

段号：

在解释程序中段的存取通过段号来完成，连接程序判定实例树的结构（这就是为什么段表也要被程序指定）。因此，段号的判定必须在连接程序和LZS中以一完全相同的方式被执行。

此外，为了最优化，在相同的段号下定义确切的段是有意义的。一个为0的分配号在8051上是特别有效且可行的。这个最常经历一个特别处理的段，更合适被分配为0号。因此我们得出自己的约定，网络映像（kLzsSegNI）通常被分配为0号。下表给出了所有的保留段号：

number	Type	description
0	KPlcSegNI	网络映像
1	kPlcSegPI	进行映像
3	kPlcSegNcHelp	本地代码的帮助段
7	kPlcSegBinDcf	CANOpen栈的配置信息
8	kPlcSegVarTable	变量表
9	kPlcSegHardwareConfig	硬件配置段

图34 保留段

段头：

所有的段都有一个6或者8字节的段头，字节长度取决于对齐。段头的头2个字节包含了段的长度（包括段头），第3个字节包含了magic代码。此外，在第4和第5个字节表示已分配的段的段号，表明段类型。LZS为段表的第二列的结构解释这条记录。

Num-ber byte	Contents
2	包括段头在内的段长度
1	段类型（逻辑字节）
1	段头长度
2	分配段的段号

图35 段头

全局段表：

如果控制器不支持多任务处理，那么它可以用一个全局段（任务定义段）来管理在仅有的程序中的表中的段。

这是可以的。但是，注意需要保留段号和下载段的命令。

5.4.7.1 示例：组成部分的调用

这里有个段结构的解释过程的示例。在这个示例中我们假设有类型为A和B的两个POU已经装入PLC中。两个POU都只有一个实例（A1和B1）并且POUA调用POUB。生成的段表如下所示：

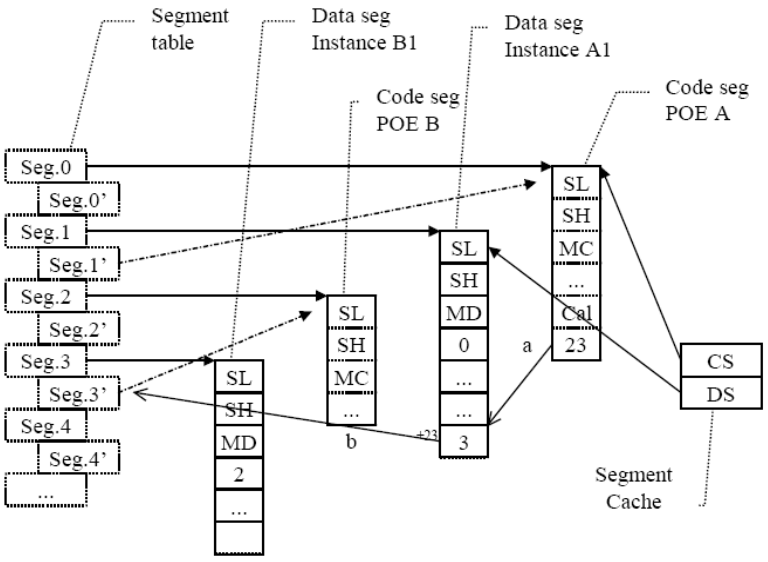


图36 段表示例

当这个被“分配”的段是一个数据段的情况下，随之而来的代码段的直接地址将是明确的。这个块的调用以如下方式处理：解释程序在运行代码段的时候执行调用（CAL）命令。调用命令在执行访问数据段时有一个偏移量（在图中：偏移量=23）。

解释程序装载在给出的偏移量上的当前数据段，这个块实例的实例处理将被调用（在图中：3）。这个实例处理与被调用的数据段的段号是完全相同的。

解释程序由段表决定数据段和代码段的地址。

现在解释程序拥有被调用的代码段的地址和属于此代码段的当前实例的数据段。它从代码段和数据段的实际的值移动到返回堆栈。解释程序装载这个绝对值并且继续运行POU（它将路过代码段的段头）。

伪代码如下所示：

```
Case CAL: //
        Offset = *ip ++;                    // 数据段本身的偏移
        OtherDS = DSAdr [offset];           // 提取数据段号
        OtherDSadr = Seg [OtherDS] [0];     // 提取数据段地址
        OtherCSadr = Seg [OtherDS] [1];     // 提取CS地址
            Push Ip, DS;                    // 保存老的寄存器
        DS: OtherDS =                       // 放入新DS
        IP: OtherCSadr+ = 4                 // 放入新IP
                                            // 为委托调用增加代码
                Break;
```

图37 解释程序调用（CAL）伪代码

5.4.7.2 任务定义段、

任务定义段由结构体tLzsTaskDefTable和tLzsTaskDefEntry定义。对结构体成员的说明见smart.plc\inc\smartplc\odk_plc.h中的结构体定义。

5.4.7.3 引用数据段

初始数据段（IDS）被明确地传输到控制器并且不会暗中产生。一个IDS的“分配”的段是代码段，例如：IDS被每个块下载并存储一次而不是每个实例。

在同一个块的不同实例间只有一个区域，这个区域被个别地用于块的实例处理的调用。自从有了这个区域，在重新启动时将不再需要复位。这个区域不能写只能读。

此外，它与重新启动有区别，不论保持变量是被保持还是复位。因此在IDS中需要保持一些信息。

一个简单的对IDS数据段的拷贝是不可以的。另一方面，一个对内存的更有效率的使用和对RETAIN变量的

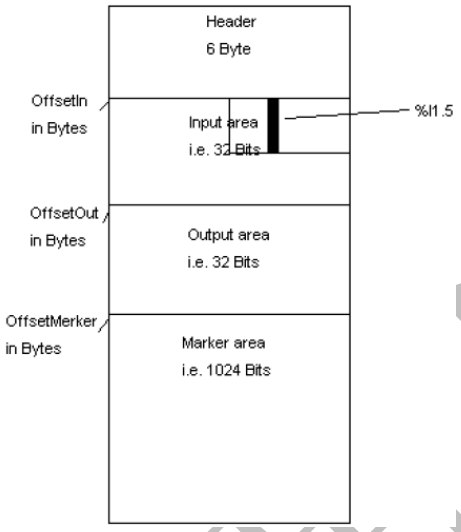
长远的发展成为可能。后端编译（CBE）必须注意实体控制代码将被存储在数据段的开始。

在3.6.1中解释了初始数据段的结构：

5.4.7.4 进程映像

5.4.7.4.1 IEC直接地地址映射

在OepnPCS中直接表示变量在进程映像中被映射。为了计算一个变量在进程映像中的位置，编译程序需要使用硬件配置文件中的记录（如:smartsim.ini）。因此将地址（右起第二个）字节与硬件配置文件中给出的偏移相加。



例如：如果你有%I1.5的直接地址并且硬件配置文件中的记录OffsetIn=6，则地址指针指向进程映像段的第8个字节的第6位。

注：前六字节为段头；地址从%I0.0到%I0.7为第七个字节；从%I1.0开始为第8个字节，则确定%I1.5位置为第6位。

5.4.7.5 网络映像

像进程映像一样，网络映像是一个资源全局段并存储被%NI或%NO编址的网络变量。这个变量被用于通过一个网络连接的不同PLC间的数据的交换。地址方案和传输机制取决于基本的网络架构，因而必须被OEM执行。这些必须被OEM执行的接口函数定义如下：

LZSBOOL LzsNetVarRTS(void)

如果网络映像正准备发送，这个函数将返回LZSTRUE。

LZSBYTE LzsNetVarStart(LZSBYTE bMode_p)

在启动初始化网络通信的时候会调用这个函数

LZSBYTE LzsNetVarStop()

LzsNetVarStop用于关闭网络通信

LZSBYTE LzsNetVarIsMaster()

这个函数用于决定在网络中的哪个PLC作为主PLC

LZSBYTE LzsNetVarReceive()

LzsNetVarReceive用于从另一个PLC接收网络映像。在网络的基础上，由分开的线路从网络接接收数据是必要的。然后LzsNetVarReceive将从接收缓冲区拷贝这些数据。

LZSBYTE LzsNetVarSend()

这个函数用于将网络映像发送到另一个PCL。需要根据网络拓扑结构计算目的地址。

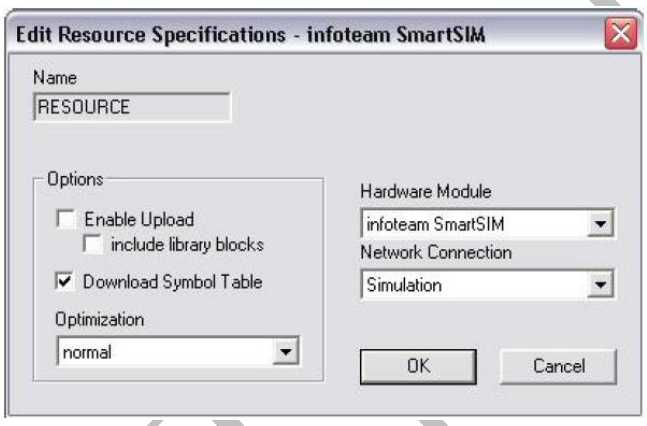
5.4.7.6 配置信息

在硬件初始化文件中定义的布局进程映像的信息将随程序一起下载到硬件中。在下载的最后，LzsCheckConfiguration将被调用来核查数据。这个函数决定段的地址且作为一个给LzsEnvCheckConfiguration的参数。见文件samples\winipc\lzsenv.c中关于这个函数的实现的示例。这个段的默认布局如下所示：

Header 6 Bytes
number of input bits 2 Byte
number of output bits 2 Byte
number of marker bits 2 Byte
offset of input in bytes 2 Byte
offset of output in bytes 2 Byte
offset of marker in bytes 2 Byte

5.4.7.7 变量表段

在V4.1或更高版本，设置变量的信息可以下载到PLC。为了生成这个变量信息表，你需要在资源说明中选择“Download Symbol Table“选项。在标准OpenPCS版本中所有的全局变量都在这个变量表中被选中。这部分可通过OEM在程序包中的“VarTab Extension”修改。为了产生这个段，tuioem32.dll需要文件pcevars.txt，这个文件在选择“Download Symbol Table“选项并编译后生成。这个段对于下载将得到号码8。这个段以Intel格式字节命令创建。



5.4.7.7.1 映射文件格式

映射文件在目录project_root\gen\resource下。这里我们说明了他们的格式。

5.4.7.7.1.1 pceVars.txt

Section	Field	Length in Byte	Description
Header	LEN	2	段长度
	TYPE	1	段类型 kPlcSegVarTable = 0x26
	HEADERLEN	1	段头长度
	CORRSEG	2	相联段 这里为0xCDCD
	NOOFREC	2	本表中的记录号
Record: NOOFREC Times	NAME	NAMELEN	变量名
	PATH	PATHLEN	变量的实例路径
	TYPE	2	变量类型
	SIZE	2	变量大小（字节）
	PROG	2	变量的程序号
	SEG	2	变量的段号
	OFFSET	2	变量在段中的偏移
	BIT	2	变量的位号
	SCOPE	2	变量的范围

5.4.7.7.1.2 pceSegs.txt

在pceSegs.txt文件中的每一行都有如下格式：

程序号/段号	段类型	相联段	大小	实例名（可选）

例如：（对于pceSegs.txt）

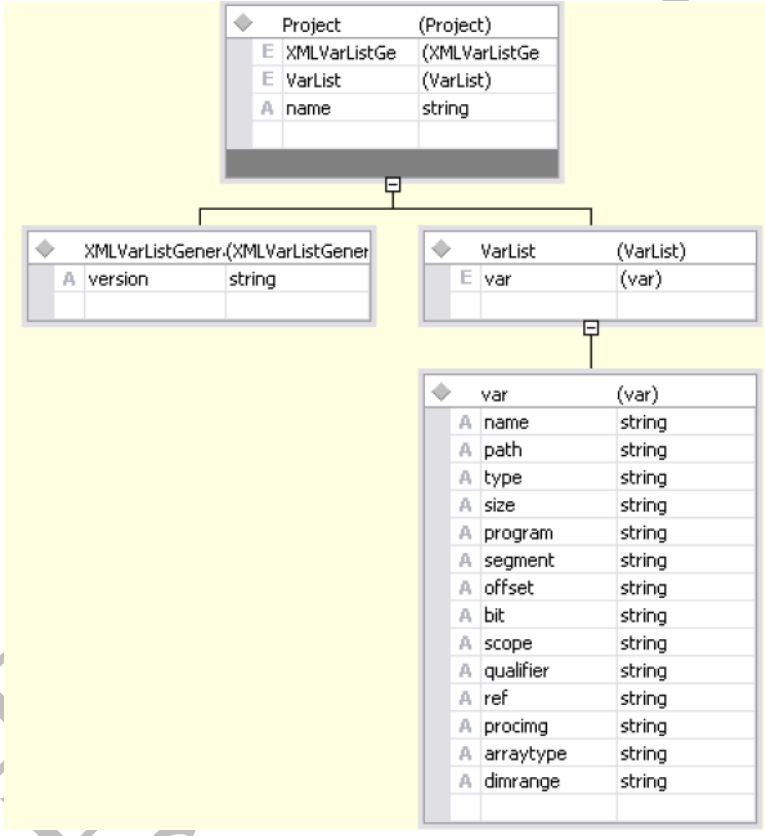
4/14	Datasegment	4/15	16	ST_PROG
4/15	Codesegment	4/16	38	
4/16	Initialsegment	4/17	22	
4/17	NativeCodeSegment	4/15	176	
3/14	Datasegment	3/15	1628	CHART
3/15	Codesegment	3/16	527	
3/16	Initialsegment	3/23	1614	
3/17	Datasegment	3/18	35	CHART._SFC_FBINS_DETERMINETRANS

5.4.7.7.1.3 pceData.txt

pceData.txt文件包含一个16进制的所有由IL编译程序生成的段的转储。此外对所有的代码段都有一个反编译UCODE的转储。

5.4.7.7.1.4 pceVars.txt.XML

在OpenPCS V6.2.0或更高版本会生成一个额外的XML文件，该文件包含了与pceVars.txt中的一样的变量。这个XML文件的设计图如下：



<Project>的属性

Attribute name	Type	Description
name	string	Path to the PCD file that was the source of this dump

<XMLVarListGenertor>属性

Attribute name	Type	Description
version	string	Version of the dumpdll that has generated this dump

<Var>属性

Attribute name	Type	Description
name	string	Name of the variable
path	string	Variable instance path
type	string	Variable type as enum (see 5.4.7.7.2.3)
size	string	Size of variable
program	string	Program number

segment	string	Segment number
offset	string	Offset in segment
bit	string	Bit position
scope	string	Scope of the variable (see 5.4.7.7.2.4)
qualifier	string	Variable qualifier CONSTANT=0x01 RETAIN=0x02 OPC=0x04 or a combination
ref	string	If 1 the variable is a referenc
procimg	string	Direct address string if variable is in the process image or “noPI” if not
arraytype	string	Optional: if variable is an array this indicates the type of members
dimrange	string	Optional: if variable is an arraythis shows ranges for all dimensions

5.4.7.7.2 V5.1或更高版本选择变量

在OpenPCS V5.1或更高版本，为变量表段选择变量的机制从硬件初始化文件中移除（如：smartsim.ini）了，所以创建一个特殊的vartab32.dll文件是必要的。

下面的VARTAB部分和它的条目定义了这个过滤器，用于选择变量。

```
[VARTAB]
type_or=65535
prog_or=65535
segment_or=65535
scope_or=8,6
name=var_local
namecomplen=9
path=main.ifbext
pathcomplen=4
type=10,9,6,7
prog=65535
segment=65535
scope=65535
```

条目”name”和”path”是字符串变量，定义了要选择变量的名称和实例路径。两个都相当于他们的”comparelength”条目”namecomplen”和”pathcomplen”，这两个条目为整数数值。已选择的变量名称或实例路径只与”complen”条目定义的数字内的字符进行比较。默认值是空字符串并且complen值为0。在两个情况（空串或complen=0）下的变量对于名称或路径都将不进行分析。

其它的条目（type,prog,segment,scope）都是整数数值，默认为65535，意味着对于这个过滤器变量将不进行分析。下面的参数介绍了这些可能的被定义的值的更多细节。

过滤器规则：

Xxx_or项（type_or,prog_or,segment_or,scope_or）以OR（或）的方式工作；如果只有一个条目匹配，这个变量定义都会被加入到变量表段中。

所有其它的项是AND（与）工作方式，所以只有所有的值都匹配，变量定义才能被加入到变量表中。

为了给一个过滤器项目选择多于1个取值，取值需要通过一个命令（对name,namecomplen,patch, patchcomplen是无效的）分离；这种多种选择像一个OR过滤器一样工作。

为了确定可以被构造的名称，名称过滤器以一个”$”标志开始。这也导致vartab32.dll文件位于路径的最后部分。

在限定词OPC中的变量定义通常是变量表的一部分，无论在初始化文件中定义的是哪个过滤器。

5.4.7.7.2.1 变量表段结构

下表为V6.4.0以后的变量表段的结构。如果你的版本低于6.4.0，请看5.4.7.7.2.2节。

Section	Field	Length in Byte	Description
HEADER	LEN	2	Length of the segment, when len > 64K set to 0xCDCD and field EXTLEN used instead
	TYPE	1	Type of segment kPlcSegVarTableExt

	HEADERLEN	1	Length of header
	CORRSEG	2	Corresponding segment here 0xCDCD
EXT_HEADER	EXTHEADERLEN	2	Length of the extended header
	EXT_LEN	4	Total length of all segments (sum of multiple VarTab segments)
	VERSION	4	Version of VarTab structure
	CHECKSUM	4	Checksum of the VarTab data: can useful to decide if VarTab changed, i.e. reload of OPC address space
	NUMBER_PATH_REC	2	Number of PATH records in this segment
	NUMBER_VAR_REC		Number of VAR records in this segment
	PRJNAME_LEN	2	String length of project name
	PRJNAME	STRING	Name of the project, i.e. COFFEE
	RESNAME_LEN	2	String length of resource name
	RESNAME	STRING	Name of the resource
PATH	RECLEN	2	Length of PATH record
	NODETYPE	1	Node type: 0 (inner node)
	RECLEN_TOTAL	2	Length of PATH record including all corresponding VAR records
	PATHLEN	2	Length of Path
	PATH	STRING	Path of this variable
	NUMBER_VARS	2	Number of VAR records related to this path
VAR (1..NUMBER_ VARS variables)	RECLEN	2	Length of VAR record
	NODETYPE	1	Node type: 1 (leaf node)
	NAMELEN	2	Length of variable name
	NAME	STRING	Name of the variable
	TYPE	2	Type of variable
	SIZE	2	Size of variable in bytes
	PROG	2	Program number of variable
	SEG	2	Segment number of variable
	OFFSET	2	Offset of variable in segment
	BIT	2	Bit number of variable
	SCOPE	2	Scope of variable
	QUALIFIER	1	Qualifier of variable
	ACCESSTYPE	1	Access type of variable
	PROCIMG	1	0: noPI, 1: Direct address
	ARRAY	1	0: noArray, 1: Array
	ARRAYTYPE	2	Type of array membervariable (exists only in case of array)
	NUMBERDIM	1	Number of dimensions (for each dimension a lower and upper range is specified)(exists only in case of array)
	LOWERRANGE	2	Lower range of array(exists only in case of array)
	UPPERRANGE	2	Upper range of array(exists only in case of array)

5.4.7.7.2.2 变量表段结构（OpenPCS 6.4.0之前版本）

Section	Field	Length in Byte	Description
Header	LEN	2	length of the segment
	TYPE	1	type of segment kPlcSegVarTable = 0x26

	HEADERLEN	1	length of header
	CORRSEG	2	corresponding segment here 0xCDCD
	NOOFREC	2	number of records in this table
Record: NOOFREC times	RECLEN	2	length of record including length
	NAMELEN	2	length of name string following
	NAME	NAMELEN	name of the variable
	PATHLEN	2	length of the path string following
	PATH	PATHLEN	instance path of variable
	TYPE	2	type of variable
	SIZE	2	size of variable in bytes
	PROG	2	program number of variable
	SEG	2	segment number of variable
	OFFSET	2	offset of variable in segment
	BIT	2	bit number of variable
	SCOPE	2	scope of variable

5.4.7.7.2.3 类型

在域TYPE中给出的变量类型有如下意义：

0	Undef
1	Bool
2	Byte
3	Word
4	Dword
5	Usint
6	Uint
7	Udint
8	Sint
9	Int
10	Dint
11	Real
12	Time
13	Date
14	Tod
15	Dt
16	Lword
17	Ulint
18	Lint
19	Lreal
20	String
21	Enum
22	Array
23	Struct
24	Bit
25	BcdReal
26	Referenz

27	User
28	ArrayOfArray
29	ArrayOfStruct
30	StructOfStruct
31	StructOfArray
32	Initvalues
33	FbInstance
34	Int24
35	Real48
40	Count

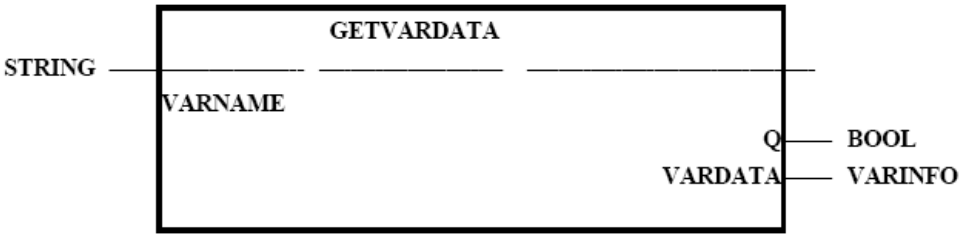
5.4.7.7.2.4 变量范围

域SCOPE中给出的范围有如下含义：

0	undefined/not allowed
1	INTERNAL
2	VAR_INPUT VAR_OUTPUT VAR_INOUT
3	VAR_OUTPUT RETAIN
4	VAR
5	VAR_GLOBAL
6	VAR RETAIN
7	VAR_GLOBAL RETAIN
8	VAR_EXTERNAL
9	%IX
10	%QX
11	%IW
12	%QW
13	%IX RETAIN
14	%QX RETAIN
15	%IW RETAIN
16	%QW RETAIN
17	%MX
18	%MW
19	%MX RETAIN
20	%MW RETAIN

5.4.7.7.3 固件FB存取变量表

下面有一个固件函数块示例，显示了在实时系统中怎样存取变量表。

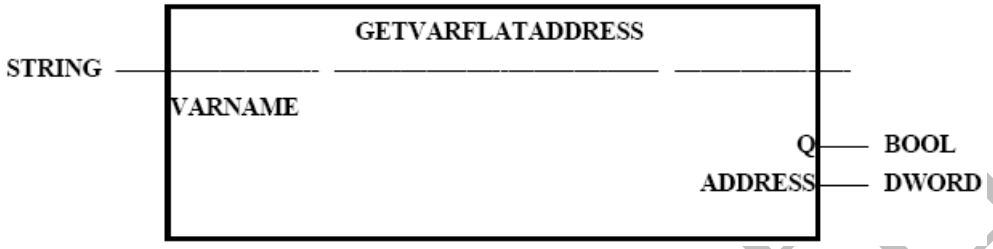


VARINFO的结构体VARDATA：

```
TYPE
    VARINFO : Struct
        TYP : UINT;
        SIZE : UINT;
        PROG : UINT;
        SEG : UINT;
        OFFSET: UINT;
        BIT: UINT;
        SCOPE: UINT;
    end_struct;
END_TYPE
```

如果Q为TRUE，VARDATA的值是有效的，否则无效。

另一个函数块显示了怎样得到一个存储在变量表中的变量的物理地址：



这个函数呈现了一个变量的物理地址可以32位数来存储。如果Q为TRUE则输出为有效的。

5.4.7.7.4 LzsVar.c

这个固件函数块使用下列lzsvar.c文件中的函数

LZSBOOL LzsVarGetVarInfo(LZSCHAR *szPath,tVarDes *sVarDes)

来从表中得到信息。在这个函数中域NAME与参数szPath比较，如果相等段中的数据将被拷贝到结构体sVarDes中，sVarDes是IEC1131结构体的一个映像。

另一个在lzsvar.c文件中的函数

tPlcMemPtr LzsVarGetVarAddress(LZSCHAR *szPath);

返回由参数给出的变量的物理地址。

注：输入-输出变量和外部变量不能被分解（见3.5.2.13处理引用节）。

5.4.8 LZS 高层

LZS高层函数提供了下列功能：

上传\下载程序；

（周期）查询功率流信息；

解释器执行；

条件控制与查询解释器

在线编辑代码段

5.4.8.1 解释器控制

和通常的启动/停止/复位函数一起，解释程序的环境可通过LZS控制。函数LzsIpCycle必须被周期地独立地调用（如：通过中断），来给予解释程序工作的时机。解释程序的环境（启动/停止）通过LzsIpCycle被监督，被周期调用的环境是可靠的。

在没有在程序中引起一个完全的周期的情况下调用LzsIpCycle是肯定可行的。例如：如果由于执行原因，有一个LZS间的对于”cooperative multitasking”的请求，这样对于IP和系统的休止的每次LzsIpCycle的调用都有最多100条指令被执行。但是，它必须在解释程序至少返回到LZS的系统检查点的时候被处理，这是为了给LZS周期执行的时机（功率流等）。

tLzsErrorCode LzsIpCycle (Void);

图38 LzsIpCycle

5.4.8.2 功率流（LzsPflow）

功率流以这样一个方式被执行：解释程序周期地记下在想得到的区域的必要的信息（不必在每个周期）。这

些信息不能在每个周期被传输。在线服务器的任务在定期的周期时间内需要这些数据并且传输这些数据。所以这个功率流可以被LZS异步地处理，LZS将轮流地使用两个缓冲区。

像上面提到的，解释程序不在每次调用LzsIpCycle时完全执行它的程序是肯定可以的。因此不确定在每次调用时有效的功率流数据都会存在。

随着LzsPflowStart的调用LZS获知这个区域，功率流是需要的。LzsPflowStop停止功率流。

```
tLzsErrorCode LzsPflowStart(
    tLzsPgmNr Program_p, // i: program id
    tLzsCodeAdr Start_p,
    tLzsCodeAdr End_p,
    tLzsSegNr Instance_p
);
```

图39: LzsPflowStart

```
tLzsErrorCode LzsPflowStop (Void);
```

图40: LzsPflowStop

LZS将把两个函数的调用（LzsPflowStart和LzsPflowStop）转到有相似名称的解释程序的接口函数。

高层实时系统用LzsPlowGetData得到当前功率流数据。

```
tLzsErrorCode LzsPflowGetData(
    BOOL * pfValid_p,          // o: new data available and valid
    BYTE ** pbData_p,          // o: pointer to data
    tLzsByteCount *pulSize_p   // o: size of powerflow data area
);
```

图41: LzsPflowGetData

在这里定义的接口，都没有控制流，这个控制流给在线服务指明新的功率流数据可用了。

当没有数据可用时，LzsPflowGetData将返回一个错误代码。系统解释器的每个组件——由高层实时系统传递给PC可以提升传输速度，因为这些组件中的每一个都可能成为最后的有效的一个。

5.4.8.3 强制（LzsForce…）

强制表通过下面的函数初始化，这意味着所有的强制变量都将清零：

```
void LzsForceInitForceTab (void);
```

图42: LzsForceInitForceTab

在一个解释周期执行前强制值与LzsForce一起应用：

```
LZSBYTE LzsForce()
```

图43: LzsForce

可通过函数LzsForceAddItem或LzsForceDeleteItem加入或移除一个强制项目，并从OpenPCS评估各自的命令：

```
LZSBYTE LzsForceAddItem (tLzsPSCmd LZSFAR* pLzsPSCmd_p)
```

图44: LzsForceAddItem

```
LZSBYTE LzsForceDeleteItem (tLzsPSCmd LZSFAR* pLzsPSCmd_p)
```

图45: LzsForceDeleteItem

使用函数LzsForceGetInfo，可以检索当前强制变量的号码：

```
void LzsForceGetInfo(LZSDWORD* p_dwNumberForcedVariables)
```

图46: LzsForceGetInfo

5.4.8.4 管理函数

接口函数LzsStart意在给LZS一个有序的初始化的时机。它期望作为一个参数，一个在CallBack函数上的指示符，来通知异步事件的环境。LzsStart仅可在开始的时候和在每次LzsStop被调用。每次LzsStart后面必须有LzsStop。所有的LZS函数都只能在LzsStart和LzsStop之间调用。

```
tLzsErrorCode LzsStart( (short int FAR PASCAL _export *pFct_p) (tLzsReason, USHORT) );
```

图47:

```
LzsStart tLzsErrorCode LzsStop (Void);
```

图48: LzsStop

5.4.9 LZS，底层

LZS底层函数提供了下列功能：

段管理

管理存储返回地址和当前结果的栈；

内存、输入、输出和网络接口的使用；

联机模式下设置变量；

联机模式下设置I/O;

5.4.9.1 段管理（LzsSeg…）

段管理没有外部接口，因此这里没有定义接口函数。下列函数只可以给出可行的实现的一些粗浅的印象。

LzsSegCreateSegTab

LzsSegCheckSegNum

LzsSegGetSegType

LzsSegGetSegNum

需要接受的是，LZS必须为我们的应用保持的变量信息（表，程序名，进程映象），为了达到一个统一的存储管理，也存储在段中。为不能为外部系统提供访问的段保留其段号是非常有意义的。

5.4.9.2 内存存取和 I/O(LzsMem…)

不同的可寻址地址区域（内存，I/O，CAN等）间的区别体现在LZS中，这是为了提供一个统一的接口给解释程序和查询编程系统。涉及到当前程序的所有调用都在LZS中设置。

为了提高性能，存取函数分为”Near”和”Far”两种形式。Near存取涉及“当前”数据段，该数据段的初始地址通过LZS在一个”Cache Register”中被分别地管理。因此这个使用权仅仅包括一个初始地址的添加和偏移的提交。函数LzsSetCurrentSeg为当前CS和DS提供一个外部的”Cache-Register”的加载。这个函数在各自的新的POU的入口上调用（解释器-启动和CAL-命令）。不同于Near存取，Far存取首先需要段表中的段绝对地址，代表一个重要的开销。

自从在Intel和Motorola底层（汇编在8051上）的最优化得以实现，存取函数”Bit”，”Byte”，”Word”和”Dword”间的区别提高了性能。对byte-函数有一个二选一的多样的调用，这是为了执行一个word-和dword-的存取（更多时间消耗）。不像其它函数，对于类型为”Any”的函数从不转换加工过的数据，所以你可以用这些函数处理像String一样的命令。

对于内存存取解释程序使用不同的函数，这些函数由LZS提供。解释程序不进行任何的直接内存存取。自从存取函数被封装后，解释程序可以通过用过的数据格式被独立地执行。这对于8051有着特殊的意义，自从通用C编译器（C51）以Motorola格式（MSB/LSB）编址变量，虽然CBE归于Intel格式（LSB/MSB）。撤销内存存取将导致数据转换封装到LZS内部。因此，解释程序通常只用一种方式获得数据，就是以通用C编译器为先决条件（Intel表示法在PC上，Motorola表示法在8051上）。

5.4.9.2.1 编址段

为了允许解释程序有一个对内存的有效的存取，我们需要区分NEAR和FAR存取。所有的NEAR存取涉及了按下列函数设置的数据段：

```
tLzsErrorCode LzsMemSetCurrentSeg(
    tLzsSegNr CodeSeg_p,            // i: current code segment number
    tLzsSegNr DataSeg_p             // i: current data segment number
);
```

图49: LzsMemSetCurrentSegment

5.4.9.2.2 当前数据段的Near存取（LzsMemNear…）

```
void LzsMemNearSetBit(
    tLzsAdrNearBit Address_p,       // i: address
    BOOL fValue_p // i: value to set
);
```

图50: LzsMemNearSetBit

```
void LzsMemNearSetByte(
    tLzsAdrNearByte Address_p,      // i: address
    BYTE Value_p // i: value to set
);
```

图51：LzsMemNearSetByte

```
void LzsMemNearSetWord(
    tLzsAdrNearByte Address_p,      // i: address
    WORD Value_p // i: value to set
);
```

图52：LzsMemNearSetWord

```
void LzsMemNearSetDword(
    tLzsAdrNearByte Address_p,      // i: address
    DWORD Value_p // i: value to set
);
```

图53：LzsMemNearSetDword

```
void LzsMemNearSetAny(
    tLzsAdrNearByte Address_p,          // i: address
    BYTE * pbValue_p,                   // i: pointer to new contents
    tLzsByteCount cBytes_p              // i: length of memory area to set
);
```

图54：LzsMemNearSetAny

```
BOOL LzsMemNearGetBit(
    tLzsAdrNearBit Address_p        // i: address to read from
);
```

图55：LzsMemNearGetBit

```
BYTE LzsMemNearGetByte(
    tLzsAdrNearBit Address_p        // i: address to read from
);
```

图56：LzsMemNearGetByte

```
WORD LzsMemNearGetWord(
    tLzsAdrNearBit Address_p        // i: address to read from
);
```

图57：LzsMemNearGetWord

```
DWORD LzsMemNearGetDWord(
    tLzsAdrNearBit Address_p        // i: address to read from
);
```

图58：LzsMemNearGetDword

```
void LzsMemNearGetAny(
    tLzsAdrNearByte Address_p,      // i: address
    BYTE * pbValue_p,               // i: pointer to receive contents
tLzsByteCount cBytes_p          // i: length of memory area to get
);
```

图59：LzsMemNearGetAny

5.4.9.2.3 对任意段的Far存取（LzsMemF...）

```
void LzsMemFarSetBit(
    tLzsAdrFarBit Address_p,        // i: address
    BOOL fValue_p // i: value to set
);
```

图60：LzsMemFarSetBit

```
void LzsMemFarSetByte(
    tLzsAdrFarByte Address_p,           // i: address
    BYTE Value_p // i: value to set
);
```

图61：LzsMemFarSetByte

```
void LzsMemFarSetWord(
    tLzsAdrFarByte Address_p,           // i: address
    WORD Value_p // i: value to set
);
```

图62：LzsMemFarSetWord

```
void LzsMemFarSetDword(
    tLzsAdrFarByte Address_p,           // i: address
    DWORD Value_p // i: value to set
);
```

图63：LzsMemFarSetDword

```
void LzsMemFarSetAny(
    tLzsAdrFarByte Address_p,           // i: address
```

```
    BYTE * pbValue_p,           // i: pointer to new contents
    tLzsByteCount cBytes_p      // i: length of memory area to set
);
```

图64：LzsMemFarSetAny

```
BOOL LzsMemFarGetBit(
    tLzsAdrFarBit Address_p     // i: address to read from
);
```

图65：LzsMemFarGetBit

```
BYTE LzsMemFarGetByte(
    tLzsAdrFarBit Address_p     // i: address to read from
);
```

图66：LzsMemFarGetByte

```
WORD LzsMemFarGetWord(
    tLzsAdrFarBit Address_p     // i: address to read from
);
```

图67：LzsMemFarGetWord

```
DWORD LzsMemFarGetDWord(
    tLzsAdrFarBit Address_p     // i: address to read from
);
```

图68：LzsMemFarGetDword

```
void LzsMemFarGetAny(
    tLzsAdrFarByte Address_p,       // i: address
    BYTE * pbValue_p,           // i: pointer to receive contents
    tLzsByteCount cBytes_p      // i: length of memory area to get
);
```

图69： LzsMemFarGetAny

5.4.9.2.4 代码段存取（LzsMemCode...）

在相似的条件下，数据段的存取也适用于代码段的存取。由于如16位的偏移（作为一个指令的操作数）存储于Intel格式，依赖于C编译器使用一个数据格式的转换也是必要的。因此，LZS提供了额外的函数来处理CS的存取。一方面这个功能的容量是简化的，因为代码段是只读的；另一方面存取只在当前的CS上进行。因此，”Far”寻址将被淘汰。把代码和数据存取分在不同的函数中实现的优势是当函数被使用时，可减少对函数的测试。

实际上作为一个规则，代码是有序的。因此，“解释指定指针（IIP）”的管理被移到LZS来完成。现在，为了决定CS的初始地址，当进入POU时仅需要调用函数LzsSetCurrentSegment。总之，通过使用函数LzsSetCodePosition，IIP为这个段（跳转到段头）的第一个指令而设置。此外，为了摆脱把每次调用的偏移都提交给LzsGetCode（然后需要额外的“段起始”与“实际偏移”相加），IIP提高了独立性并且存储在内部。

所有对代码段的存取提高了实际指令指针给予的可读数据的容量。（递增量）。

```
BYTE LzsMemCodeGetInstr (void);
```

图70： LzsMemCodeGetInstr

```
BYTE LzsMemCodeGetByte (void);
```

图71：LzsMemCodeGetByte

```
WORD LzsMemCodeGetWord (void);
```

图72：LzsMemCodeGetWord

```
DWORD LzsMemCodeGetDword (void);
```

图73: LzsMemCodeGetDWord

为了在启动的时候设置代码段里的位置，进入一个新的块（CAL,RET）和跳转，LZS提供了函数LzsMemCodeSetPos。

```
tLzsErrorCode LzsMemCodeSetPos (void);
```

图74: LzsMemCodeSetPos

5.4.9.2.5 任何数据类型累加器

一直到32位的数据都通过解释程序直接装入“累加器”。更大的数据类型被装入一个“外部累加器”，只有一个指向内存的指示符在累加器中被处理。外部累加器的最大容量将通过任务定义段被传递给LZS。仍然没有函数被指定来支持这样一个外部累加器的使用。这并不妨碍大数据类型的一般应用（如数组和结构体），但是只能作为单个个体体来使用（如：如果X和X是结构体变量，那么LD X.a;ST Y.b是可以的，但是LD X;ST Y不可以）。

5.4.9.3 功率流缓冲区管理（LzsPfbuf…）

为了执行功率流，解释程序有两个分开的内存位置可供选择（见2.8.5节）。LZS提交其中一个缓冲区给高层实时系统，这是为了将它传递给编程系统。这个缓冲区的良序的释放和阻塞就和管理缓冲区溢出一样是LZS的任务，不是解释程序的。

功率流缓冲区的结构如下所示：

Bytes	Content
2	Instance handle
2	Offset in Code segment 1
4	actual result 1
2	Offset in Code segment 2
4	actual result 2
etc.	etc.

图75 功率流缓冲区结构

所有LzsPfbufWrite的调用都应该以LzsPfbufOpen开始并且以LzsPfbufClose结束。通过这些调用的使用，LZS来决定哪个可用的缓冲区将被使用。通过LzsPfbufOpen它可以通知解释程序缓冲区是否可用。如果这些缓冲都不可以被存储，解释程序可以制止LzsPfbufWrite的调用。LzsPfbufWrite必须接受调用，即使没有可用的缓冲区或者出现缓冲区溢出。

```
BOOL LzsPfbufOpen(void); // return TRUE if buffer ok
```

图76: LzsPfbufOpen

```
void LzsPfbufClose(void);
```

图77: LzsPfbufClose

5.4.9.3.1 解释程序接口

为了运行装载的程序，LZS只使用一个解释程序的函数。不是每一个对IpCycle的调用都必须完全的运行程序，但是IpCycle必须至少在每个程序周期内响应它的调用。

```
typedef enum {
        // ...
} tIpErrorCode;
```

图78: tIpErrorCode

```
tIpErrorCode IpCycle (
        tIpRunData * pRunData_p,        // io: detailed start info
        BOOL * pfSync_p                 // o: TRUE if program completed
);
```

图79: IpCycle

当调用此函数时，在程序开始的地方，解释程序必须通过IpCycle接收信息。我们将在IpCycle上提交一个指向结构体的指针，这个结构体包含了这些想要的信息。

对于更远的应用这个结构体可以两个方向来使用。

解释程序使用这个结构体来存储运行中的程序停止的断点，当程序下次调用时方可由此断点断续运行。这对于LZS是可见的。

LZS可以使用这个结构体来存储关于程序运行的进一步的信息，如：到段表的指针（可选的）或解释程序应该多长时间和多大范围内运行程序的信息。同样还有功率流是否并且在哪个区域执行。

```
typedef struct {
        tLzsSegNr m_Seg;                // start/continue execution here
        USHORT m_Offset;                // start/continue execution here
        tLzsSegNr *m_pSegTable;         // pointer to segment table
        USHORT m_cInst;                 // do not more than this instructions
} tIpRunData;
```

图80: tIpRunData

5.4.9.3.2 解释程序中的功率流

通过LZS将提交一个功率流的请求给解释程序，用于累积请求信息。

```
tIpErrorCode IpPflowStart(
        tLzsPgmNr Program_p,            // i: program id
        tLzsCodeAdr Start_p,            // i: start address
```

```
    tLzsCodeAdr End_p,          // i: end address
    tLzsSegNr Instance_p,           // i: instance to do powerflow for
    tLzsSegNr Buffer1_p,            // i: first buffer to use
    tLzsSegNr Buffer2_p         // i: second buffer to use
);
```

fig 81: IpPflowStart

```
tIpErrorCode IpPflowStop(void);
```

Fig. 82: IpPflowStop

5.4.10 任务信息

V4.2版本的嵌入式SmartPLC实时系统提供了一个叫GetTaskInfo的固件函数块。这个函数块被用于获得任务在调用的地方的信息。在输出参数中存储的信息有如下意义：

Name	Type	Description
Count	DWORD	number of cycles this task is executed
LastCT	TIME	time needed for last cycle
AverageCT	TIME	average time needed for execution
MinCT	TIME	minimum time needed for execution
MaxCT	TIME	maximum time needed for execution
State	DWORD	not yet used

这个函数没有输入变量。为了使用这个函数块，实时系统必须完成_LZS_TASKINFO_的定义（config.h）。

5.4.11 编程模式

实时系统为维护设置了一个编程模式。当编程模式关闭后，下列行为是被禁止的：

- 下载
- 复位
- 启动/停止
- 加入和删除断点
- 从TUI设置和强制变量

默认的编程模式是开启。编程模式可通过调用函数LzsSetProgramMode（LZSBOOL fSet）来改变。如果fSet为LZSTRUE，编程模式开启；如果fSet为LZSFALSE，编程模式关闭，例如可从硬件开关的状态的程序操作来实现。

5.4.12 RETAIN(掉电保持)

保持功能的基本类函数是作为实时系统（LZS）的一部分被一般地执行。他们是为了决定所有的RETAIN的存储区域，为了提供RETAIN的值并且调用OEM指定函数。RETAIN值由字节数组的方式提供，这些数组通过使用OEM指定函数被写到连续内存中，并且在重启的时候可在这个存储空间中重新存储。

文件LzsEnv.c中包含了下列对于这个应用的函数：

LZSBOOL LzsEnvEnableRetain(void)

这个函数在每个IEC任务结束后被调用。如果返回值为LZSTRUE，所有的RETAIN将被收集并且函数LzsEnvEnableRetain()将被调用。如果返回值为LZSFALSE，没有任何动作发生。LzsEnvEnableRetain()定义了RETAIN多少时间存储一次。例如：使用一个示例处理所示的确定的间隔。

LZSBOOL LzsEnvWriteRetain(void)

这个函数以一个适当的方式存储来自实时系统的字节数组，例如：在flash内存或电池支持存储空间的一个文件中。这个字节数组的位置和大小由全局变量”pRetainData_g”和”dwRetainDataSize_g”定义。

LZSPUBLIC32 LZSBOOL LZSPUBLIC LzsEnvReadRetain(void)

这个函数从连续内存中还原RETAIN值使之以字节数组的形式为实时系统所用。从字节数组中还原RETAIN值需要调用函数LzsMemRestoreRetainData()，这个函数包含了指向这个字节数组的指针和数组的大小两个参数。

LZSCONST LZSCHAR* LzsEnvGetRetainStorageName(LZSBYTE bParam)

这个函数可以返回一个RETAIN值的存储区域的引用（如：一个文件名），在需要读取或写入这些值的时候。

5.5 OpenPCS RT-内核

5.5.1 OPENPCS-RT-内核描述

5.5.1.1 一般描述

本文档所述的RT内核是OPENPCS的一部分。它的用途是根据IEC1131标准的IL编程的执行。有了OPENPCS-RT内核，抽象地执行成为可能，在不同的目标系统上有独立的硬件中间代码（”RT-内核代码”）。通常，它是PLC的固件的一部分（与通信软件和实时系统/操作系统无关）。由于有非常广泛的不相同的目标系统被放置在一个简单的可移植内核的实现上。因此平台指定的操作（如：内存存取程序）被移入实时系统内部（德国”Lauf-Zeit-System”，因此叫做LZS）。这使得OPENPCS-RT内核可在不同的微处理器和PC中使用。它包含了任务管理的功能和编译程序生成的代码的执行。

RT-内核使用了一个段式内存模式。通过这个模式，每个POU拥有一个自己的代码段并且这个POU的每个实例都拥有一个自己的数据段。内存段通过OPENPCS代码生成器生成并且由LZS来分析和管理。

RT-内核在最初被设计为集成到一个”多任务-环境”中。它提供了特殊的函数，这些函数可以将所有的指定变量存储在一个外部存储区域。其他的函数可以还原这里的变量的值。通常，在一个完全的RT-内核构造的执行后两个任务间可以实现转换。内核构造的执行是不可以中断的。

5.5.1.2 RT-内核接口

RT-内核被封闭进实时系统内部。它为高层LZS和底层LZS提供了接口。当高层LZS控制RT-内核时（它选择程序来执行，启动RT-内核），底层LZS提供带内存存取程序的内核（函数LzsMem_xxx和LzsCode_xxx）。用于内存存取的LZS-函数被集中到一个“内存存取模块”，这是为了使对目标系统的移植更舒畅。这个内存存取模块可以汇编或者C语言执行，这取决于目标系统。对于8051，这个模块作为一个汇编程序集来执行，因为在没有丢失性能的情况下区分Intel和Motorola的数据格式（小端字节顺序/大端字节顺序）的区别是必要的。由于这个内存存取程序被集中于一个外部的模块，RT-内核可以一个独立的虚拟硬件方式来执行。内存存取模块提供给内核的函数将在5.5.3节中描述。

另一方面，这个内核输出了一些函数给LZS，如控制RT-内核的函数；程序执行的函数；版本信息的函数。这些函数将在5.2.2节中描述。

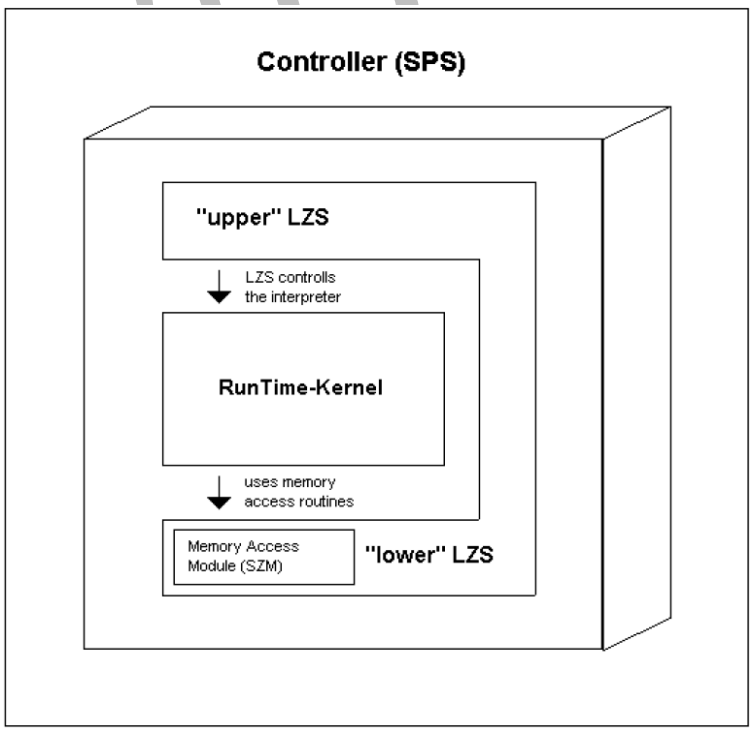


图83 RT-内核在系统中的位置

图83显示了RT-内核与LZS间的接口并且给出了RT-内核在全部的系统中的位置。

5.5.1.3 RT-内核代码执行

OPENPCS-RT内核在不同的目标系统上有独立的硬件中间代码（“RT-内核代码”），并且可以抽象地执行。中间代码由1字节长的操作码组成。取决于代码的指令，一个或多个字节的附加的操作码将跟在其后。对此指令的约定详见5.6.7节。对于操作码的执行，对每个结构都有其各自的叫做IpIP_xxx的执行函数。

在逐步执行初始化期间，RT-内核的“指令指针”必须设置为指向程序-POU（主-POU）的第一条指令的操作码。为了读取一条指令，内核将调用函数LzsMemCodeGetInstr（见5.5.3.5），这个函数由内存存取模块提供。随着操作码的读取，RT-内核将调用与此指令相应的执行函数。如果指令带有任意的操作数，它的函数必须自身读取操作数（然后指令指针将适当地增加）。然后指令将被调用的RT-内核函数执行。执行完成后指令指针指向下一条指令的操作码。当程序已经完全执行后（周期结束）或者出现一个错误后RT-内核指令的执行将停止，LZS发送一个中断信号或RT-内核指令IP_OP_BREAK被调用（见5.5.4.2）。

5.5.1.4 RT-内核数据处理

OPENPCS-RT-内核的工作就像一个自动堆栈，如：全部的数据首先通过装载指令入栈，在指令执行前使第一个和第二个堆的元素结合。这个操作的结果被出栈并且可能在稍后被处理或都回写入内存。在IL指令执行期间，结合的指令（和变量）被代码生成器分离为a)一个将操作数入栈的装载指令b)指令自己的执行处理。

IL指令通常为一个操作数结合了一个累加器（德国人称”aktuelles Ergebnis”为AE，”current result”）；这个操作的结果被回写入累加器中。这意味着累加器在相同时间是堆栈的栈顶元素，而操作数是堆栈的第二个元素。所有其它的堆栈的层只被用于分解括号运算。

由于完全的堆栈管理在8051上是非常消耗时间的，所以这两个堆栈的高层元素被“映射”为RT-内核的全局变量，这两个变量可以在微控制器的内部RAM中发现。这意味着在RT-内核指令的执行时间上的巨大的改进，因为结合的操作可以直接地将第一个和第二个堆栈元素作为变量来编址。这样为了分解括号运算只需要对“外部”堆栈编址，这个“外部”堆栈将相当地少。这两个全局变量叫做AE和AEs（阴影）。由于实现了将第一个和第二个堆栈元素映射为全局变量，由此使得用“装载”指令直接地对这两个堆栈（AEs）的元素的编址成为可能。因此我们可以在没有为栈顶元素分配一个位置的情况下执行。为此我们需要两次“装载”指令的调用，一次用于存取AE，一次用于存取AEs。

下述示例说明了AWL指令是怎样被代码生成器分离为可被RT-内核执行的“装载”和“结合”指令的。

IL资源代码	生成的代码 / RT 内核
LD Var1	AE := Var1
ADD 10	AEs := 10
	AE := AE + AEs
ST Var2	Var2 := AE

IL编程语言的一个特色就是累加器（AE）没有固定的数据类型，但是它的当前的数据类型取决于处理过的数据的类型。在RT-内核中累加器被作为一个UNION来执行，这个UNION包含每个BOOL,BYTE,WORD和DWORD类型一个的变量。

5.5.1.5 RT-内核任务管理

把RT-内核安装到一个多任务环境是可以的。无论如何LZS拥有任务管理的能力，虽然RT-内核在给定的时间只需要知道“当前”的程序。对于RT-内核在多任务环境中的应用，相应的有初始化、存储和还原当前任务状态的函数。这个用途的内存的分配由LZS来执行。下图举例说明了LZS和RT-内核间的接口。此外它显示了任务管理必要的变量，这些变量在哪个模块中可发现以及他们怎样存取。

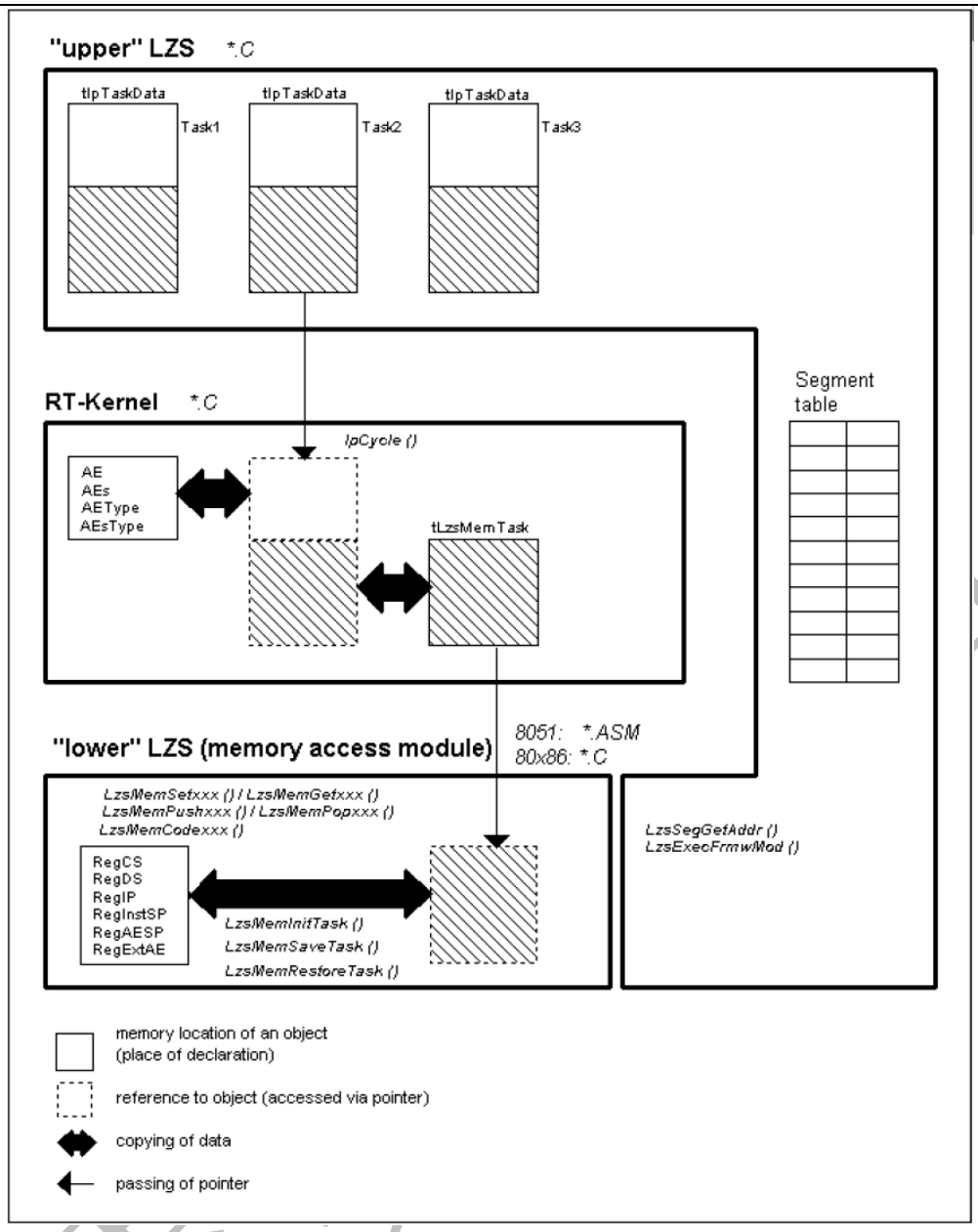


图84 RT-内核与LZS间的接口和传递参数

高层LZS与RT-内核间的接口是函数IpCycle（见5.5.2.1节）。它接受RT-内核的控制命令和一个指向结构体类型为tIpTaskData的变量的指针作为其参数，此结构体的声明如下所示：

```
typedef packed struct
{
        /* initializiation data from LZS to interpreter */
        /* ( this values may be modifed by the interpreter ) */
    tPlchInst m_MainInst;           /* handle to main instanz ( CS / DS ) */
    tPlcSegNr m_SegInstStack;       /* number of segment instance stack */
    tPlcSegNr m_SegAEStack;         /* number of segment AE stack */
    tPlcSegNr m_SegExtAE;           /* number of segment external AE (for ANY) */
    tPlcSegNr m_SegExtAEs;          /* number of segment external AEs (for ANY) */

    /* current state of interpreter */
    tPlchInst m_CurrInst;           /* handle to current instance */
    LZSDWORD m_CurrNsOffset;        /* offset of not completed native code segment -cd- 80x86 ncc */
    tIpStatus m_Status;             /* return state of interpreter */

    /* buffer for save / restore of task state of the interpreter */
    LZSDWORD m_CurrAE;              /* current value of AE */
    LZSDWORD m_CurrAEs;             /* current value of AEs */
    tIpAEType m_TypeAE;             /* current type of AE */
    tIpAEType m_TypeAEs;            /* current type of AEs */
```

```
        tPlcMemPtr m_CurrCS;            /* base address of current CS of this instance */
        tPlcMemPtr m_CurrDS;            /* base address of current DS of this instance */
        tPlcMemPtr m_CurrIP;            /* current IP of this instance */
        tPlcMemPtr m_CurrInstSP;        /* current SP InstStack of this instance */
        tPlcMemPtr m_CurrAESP;          /* current SP AEStack of the instance */
        tPlcMemPtr m_CurrExtAE;         /* Base address of external AE (for ANY) */
        tPlcMemPtr m_CurrExtAEs;        /* Base address of external AEs (for ANY) */
            tPlcPgmNr m_PgmNr;          /* Programnumber */
        LZSWORD m_UsedIStackEntrys;     /* used stack entries */
        LZSWORD m_MaxIStackEntrys;      /* max. stack entries */
        LZSWORD m_UsedAStackEntrys;     /* used stack entries for AE stack */
        LZSWORD m_MaxAStackEntrys;      /* max. stack entries AE stack */
    } tIpTaskData;
```

对于每一个任务，LZS都会为其创建任务自己的类型为tIpTaskData的结构体实例。这个变量用来一方面为LZS传递数据给内核另一方面为任务转换存储任务的内部状态。此外RT-内核举例说明了一个类型为tLzsMemTask的结构体变量，这个结构体变量拥有tIpTaskData中的数据的子集。这个结构体用于RT-内核与底层LZS间的数据交换（“内存分配模块”）。随着tIpTaskData的一部分复制到tLzsMemTask中，“高层”和“底层”RT-内核的接口将各自生效。这是特别重要的，因为对于一些目标系统以汇编语言执行底层LZS（“内存分配模块”）是合理的。如在8051上，这可以使得数据转换更快（小端字节顺序/大端字节顺序）。这是这种方法的缺点，无论如何C语言模块与汇编语言模块之间的接口不能通过共享头文件来“同步”。作为替代，所有的外部结构体必须声明两次，作为头文件和包含文件，这取决于语言使用的约定（#define vs.EQU,…）。另一方面，这分离的两个接口使得在同一时间并且没有适应内存分配模块的情况下修改结构体tIpTaskData成为可能。

```
    typedef packed struct
    {
                /* Initialization data (to be passed by IP) */
        tPlchInst m_MainInst;           /* Handle Main-Instance ( CS / DS ) */
        tPlcSegNr m_SegInstStack;       /* Segment-No. Instance-Stack */
        tPlcSegNr m_SegAEStack;         /* Segment-No. AE-Stack */
        tPlcSegNr m_SegExtAE;           /* Segment-No. external AE (for ANY) */
        tPlcSegNr m_SegExtAEs;          /* Segment-No. external AEs (for ANY) */

        /* buffer area for Save-/Restore */
        tPlcMemPtr m_CurrCS;            /* Baseaddress current CS of instance */
        tPlcMemPtr m_CurrDS;            /* Baseaddress current DS of instance */
        tPlcMemPtr m_CurrIP;            /* current IP of instance */
        tPlcMemPtr m_CurrInstSP;        /* current SP InstStack of instance */
        tPlcMemPtr m_CurrAESP;          /* current SP AEStack of instance */
        tPlcMemPtr m_CurrExtAE;         /* Baseaddress external AE (for ANY) */
        tPlcMemPtr m_CurrExtAEs;        /* Baseaddress external AEs (for ANY) */

        LZSWORD m_UsedIStackEntrys;     /* current Stackdepth Instance-Stack */
        LZSWORD m_MaxIStackEntrys;      /* max. Stackdepth Instance-Stack */
        LZSWORD m_UsedAStackEntrys;     /* current Stackdepth AE-Stack */
        LZSWORD m_MaxAStackEntrys;      /* max. Stackdepth AE-Stack */
    } tLzsMemTask;
```

一方面虽然在两个结构体之间拷贝数据是一个缺点，但是优点是分离出“高层”和“底层”RT-内核接口。因此RT-内核与内存存取模块间的接口针对RT-内核与“高层”LZS间的接口的变化来说是稳健的。如果RT-内核仅仅从LZS传递指向tIpTaskData的指针给内存存取模块，虽然在运行时间内是有优势的，但是系统将更难于维护。

5.5.1.6 通过 RT-内核暗中执行的标准函数清单

OPENPCS-RT-内核为数据类型的转换提供了RT-内核指令。因此在没有确定的*_to_**类型的LZS的外部的函数的情况下是可以实现的。在RT-内核中的直接类型转换与明确地把所有需要的函数编码并且将它们包含进LZS相比是更有效率的。

5.5.2 RT-内核输出函数

RT-内核为其控制提供了它的环境函数，用于程序的执行和生成程序版本信息。这些函数集中于资源文件IP_MAIN.C中并且以EXPORT来声明。“关键字”EXPORT的定义，在头文件ODK_PRJ.H中声明且必须移植到目标系统和开发环境中来使用（见5.5.7.1节）。

5.5.2.1 RT-内核函数<IpCycle>

函数IpCycle是RT-内核的主函数并授权LZS控制RT-内核。它的原型声明如下：

tIpErrorCode EXPORT IpCycle (tIpTaskData *pIpTaskData_p, tIpTaskCmds IpTaskCmd_p);

这个函数的第一个参数是一个指向类型为tIpTaskData的结构体变量的指针。这个类型的声明在任务管理节有解释（5.5.1.5）。第二个参数IpTaskCmd_p是一个控制代码编码初始化、存储、还原或一个任务的执行。下列为可能的控制代码：

```
// Control Codes for RT-kernel
typedef enum
{
    kIpInitTask = 0x00,          // initialize
    kIpSaveTask = 0x01,          // save task state
    kIpRestoreTask = 0x02,       // restore task state
    kIpRunTask = 0x03            // execute program
} tIpTaskCmds;
```

一些命令需要通过赋值结构体类型tTaskData中的一些元素的值并由LZS传递值给RT-内核。RT-内核也对这个结构进行写操作。下列对于内核命令的说明包含了结构体中的元素，这些元素必须由LZS来设置。此外还解释了每个控制命令的执行过程：

kIpInitTask　这个命令初始化一个任务。LZS将赋值主实例的外部累加器的操作号给这些成员：m_MainInst,m_SegInstStack,m_SegAEStack和m_SegExtAE，这个操作号和堆栈段的段号一样。RT-内核拷贝这些值到结构体tMemTask中并且传递他们给内存存取模块。有了这些值，内存存取模块将计算代码段、数据段和堆栈段的地址和外部AE的地址，并且赋值结果给内部的“高速缓存寄存器”（见5.5.3节对内存存取模块的描述）。此外，RT-内核将初始值赋值给AE寄存器、AEs和存储类型变量AEType和AEsType。

kIpSaveTask　这个命令在一个任务转换前存储当前任务的状态的时候被调用。LZS传递一个指向类型为tIpTaskData的结构体变量的指针，这个结构体变量保留了当前任务的信息，并且用于之后的对任务的当前状态的存储。RT-内核向内存分配模块传递函数调用。这个模块保存当前的结构体tMemTask中的“高速缓存寄存器”的内容。在这之后，RT-内核将这些值放入结构体tIpTaskData并且加入它自己的内部数据（AE,AEs,…）。在这些操作完成后，tIpTaskData保留一个全部任务相关的数据的拷贝。

kIpRestoreTask　这个控制命令在还原一个已经中断的任务的当前状态时被调用。LZS传递一个类型为tIpTaskData的结构体变量的指针，这个结构体变量保留了这个任务的全部将要被还原的信息的拷贝。RT-内核拷贝“高速缓存寄存器”中的内容到结构体tMemTask中并且传递这个结构体到内存存取模块。在这之后，内核将还原它自己的内部数据（AE,AEs,…）。

kIpRunTask　这个命令使RT-内核以当前代码的位置开始执行命令。在调用kIpRunTask之前，RT-内核的命令kIpInitTask必须被调用，这是为了初始化所有的内部“高速缓存寄存器”（如：RT-内核的指令指针）。在一个优先的kIpRestoreTask的调用后，任务的执行将从它中断的位置继续。

如果只有一个程序在执行，在时间上有明显优势，如果任务的状态已经通过明确地调用命令kIpSaveTask和kIpRestoreTask被存储和还原——相对于当RT-内核被停止并且还原时暗中地存储的情况下。(这至少为8051保留）此外，为了快速处理网络报告，RT-内核可能不止一次的随着一个运行中的程序的“时间片”来为支持LZS而释放处理器。在这种情况下，存储和还原内部任务状态是不必要的。另一方面，如果LZS想要在任务间转换，控制命令kIpSaveTask和kIpRestoreTask必须被适当地调用。在8051中的每个函数调用的运行时间大概是1.3ms。如果把LZS的其它管理操作也考虑进去，在任务间的转换操作的时间花费也能控制在5ms以内。

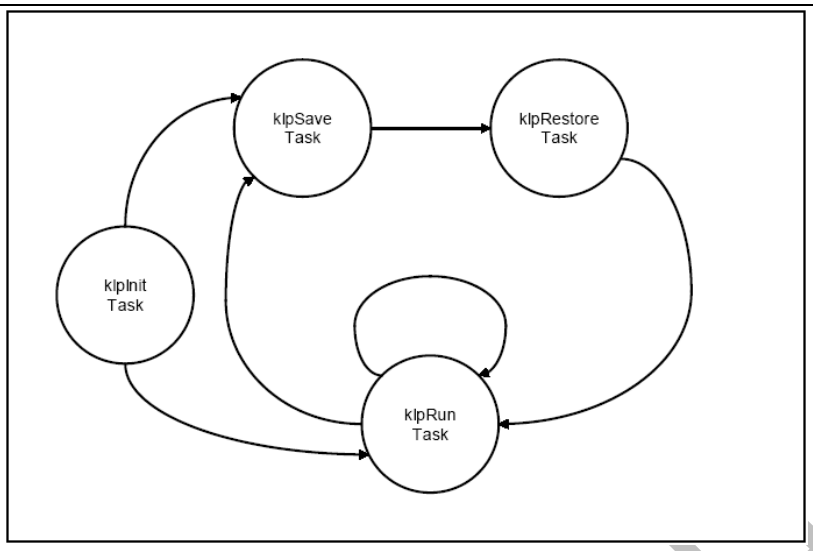


图85 RT-内核的控制命令

函数IpCycle的返回值保留了程序解释错误的信息。对于这些错误代码的含意，请查阅IP_DEF.H文件。如果出现异常的程序RT-内核的内部任务状态将被保持。因此如果问题被保存了，程序的执行可在发生错误的点被恢复。

随着控制命令kIpRunTask的调用RT-内核函数IpCycle将启动现有程序的执行。指令的执行将在下列情况下被RT-内核中断：

- POU-程序的RET-指令到达（周期结束）
- 执行一条指令期间出现错误
- 接收到一条LZS的中断报告，或者
- RT-内核指令IP_OP_BREAK执行

LZS可以通过求值函数IpCycle的返回值和结构体tIpTaskData中的元素m_Status来识别RT-内核的返回状态。

- 如果函数IpCycle的返回值不等于kIpOK(0x00)，程序的执行将由于一个错误而中止。出现错误的原因将被编码在返回值中。（见文件IP_DEF.H中的tIpErrorCode的定义）

- 如果在标志寄存器IpTaskData.m_status中的kIpCycleEnd位是打开的，则周期将结束并且LZS将更新进程映象。仅在程序执行期间没有错误的时候这个标志才会被置位。RT-内核将自动把指令指针设置为指向程序的开始，并且在下一个IpCycle的调用后将开始一个新的周期。

- 如果在标志寄存器IpTaskData.m_status中的kIpRecBrkReq位是打开的，将接收到一个LZS的断点需求。内部的RT-内核状态不会改变，所以程序可以在下个IpCycle调用的中断点继续执行。LZS可以通过调用函数IpSetBreakRequest来得到一个断点（见5.5.2.2节）。

- 如果在标志寄存器IpTaskData.m_status中的kIpExecBrkCmd位是打开的，RT-内核指令IP_OP_BREAK将被调用，这个指令被用于调试（见5.5.4.2节）。内部的RT-内核状态不会改变，所以在RT-内核指令IP_OP_BREAK之后的下一个IpCycle的调用时程序可以继续执行（透明的断点）。

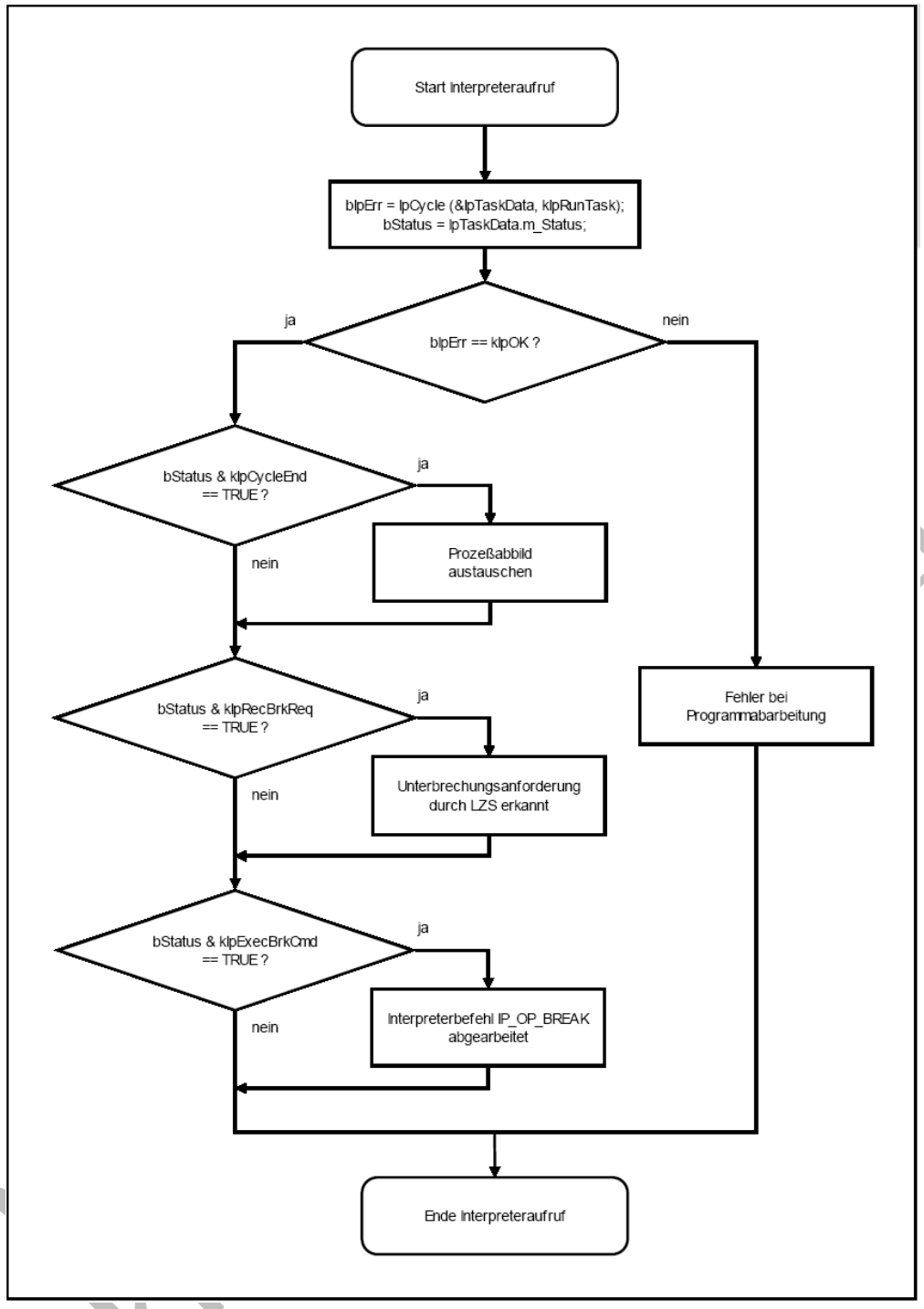


图86 RT-内核状态求值

第0部分包含了一个详细的RT-内核指令执行和返回值求值的示例。

5.5.2.2 RT-内核函数<IpSetBreakRequest>

函数IpSetBreakRequest允许LZS发送一个同步的断点请求。它的首部分声明如下：

void EXPORT IpSetBreakRequest (void);

为了在任何时候中断RT-内核的程序执行，LZS可使用这个函数。在内，这个函数将设置一个请求位，这个请求位将在每个完全的指令执行完成（就像一个处理的中断标志）后由RT-内核检查。如果请求位打开，RT-内核将返回给LZS。在这之后，LZS将通过另一个对函数IpCycle的调用来继续从断点的地方执行程序。

只要断点请求没有被RT-内核应答，函数IpSetBreakRequest将被调用不止一次。从一个中断程序中来调用这个函数也是可以的（计时器，通信子系统，…）。

5.5.2.3 RT-内核应用示例

下列代码摘录说明了对于任务管理和程序执行的RT-内核命令的调用。这里有一个主程序和一个警告-POU。假定为了一个警告，LZS通过调用函数IpSetBreakRequest来完全程序执行的中断。

```
//---------------------------------------------------------------------------
// Initialization of tasks
//---------------------------------------------------------------------------
// Initialization Program-POU
IpTaskData1.m_MainInst = hPorgMainInst;
IpTaskData1.m_SegInstStack = hProgSegInstStack;
IpTaskData1.m_SegAEStack = hProgSegAEStack;
IpTaskData1.m_SegExtAE = hProgSegExtAE;
IpCycle (&IpTaskData1, kIpInitTask);
IpCycle (&IpTaskData1, kIpSaveTask);
// Initialization of Alert-POU
IpTaskData2.m_MainInst = hAlarmInst;
IpTaskData2.m_SegInstStack = hAlarmSegInstStack;
IpTaskData2.m_SegAEStack = hAlarmSegAEStack;
IpTaskData2.m_SegExtAE = hAlarmSegExtAE;
IpCycle (&IpTaskData2, kIpInitTask);
IpCycle (&IpTaskData2, kIpSaveTask);
//---------------------------------------------------------------------------
// Cycle loop
//---------------------------------------------------------------------------
IpCycle (&IpTaskData1, kIpRestoreTask);
while (1)
{
    // read process image
    LzsReadImage();
    // call RT-kernel with program POU
    IpErr = IpCycle (&IpTaskData1, kIpRunTask);
    if (IpErr == kIpOK)
    {
        // End of cycle ?
        if (IpTaskData1.m_Status & kIpCycleEnd)
        // write process image
            LzsWriteImage();
            // break request ?
        if (IpTaskData1.m_Status & kIpRecBrkReq)
        {
            //------------------------
            // insert Alert-POU
            //------------------------
            IpCycle (&IpTaskData1, kIpSaveTask);
            IpCycle (&IpTaskData2, kIpRestoreTask);
            IpCycle (&IpTaskData2, kIpRunTask);
            IpCycle (&IpTaskData2, kIpSaveTask);
            IpCycle (&IpTaskData1, kIpRestoreTask);
        }
        // BREAK-command ?
        if (IpTaskData1.m_Status & kIpExecBrkCmd)
        // ...
    }
    else
    // error handling ...
    break;
}
```

5.5.3 内存存取模块（SZM）的输入函数

5.5.3.1 内存存取模块介绍

内存存取模块（SZM——德语”Speicher-Zugriffs-Modul”）是LZS的一部分。它隐藏在系统内存的RT-内核性能下（指针类型，内存区域DATA,XDATA,CODE,…）并且有数据格式（小端字节顺序/大端字节顺序）。这个模块由一些叫LzsMem_xx的函数组成，这些函数必须由LZS提供。同时它还可以通过保存一些全局变量类型的段地址和指针来提升内存存取的效率，这些地址和指针被指定为“快速存储寄存器”。这减少了RT-内核中的函数的运行时间和工作量。由于内存存取程序在RT-内核中是相对独立的，所以移植到其它目标系统时不需要修改RT-内核。因此RT-内核变成了独立的平台并且仅需要很小的工作量就可移植。自从内存取越来越频繁（平均每个RT-内核命令有3次内存操作），一个内存存取模块（SZM）的最优化在所有类型的系统的执行上有着积极的作用。

下列为SZM支持的函数：

Near memory access(Near内存存取)： Near内存存取用于存取POU的正在运行的实例（“当前数据段”）。实例的句柄在POU的入口被传递给SZM。相应的段地址必须通过SZM计算并且被分配到一个内部的“快速存储寄存器”中。当调用一个Near函数时，RT-内核仅仅报告地址的偏移量。段的地址通常就是“当前段”。这个方法可与在80x86处理器上的存储编址比较：在80x86上，数据段的基础地址必须在程序执行前写入到DS寄存器中（这里：POU里的入口）。在没有任何程序行动时，处理器内部地加入基础段和偏移量，这个基础段和偏移量是命令的一部分。

Far memory access(Far内存存取)： Far数据存取用于与其它POU的数据交换（参数传递）和存取进程映象。RT-内核即不报告偏移量也不保存将要存取的数据段。照例，LZS将从段句柄处计算段地址。

Code access(代码存取)： RT内核只依赖POU的代码段里的当前正在执行的代码的存取（“当前代码段”）。代码存取像Near数据存取一样实例的句柄在POU的入口被传递给SZM。相应的段地址必须通过SZM计算并且被分配到一个内部的“快速存储寄存器”中。自从代码被普遍地存取在一种完全连续的方式（连续的指令执行），SZM必须维持一个比得上硬件处理的“指令指针”。这个指针在每个单独的代码存取后将自动增加。相当不寻常的是，用一个特殊将函数这个“指令指针”移到代码段的任何位置是可以的，这是为了执行跳转指令。由于这个原因，SZM必须不仅要存储“指令指针”，而且也要存储“当前代码段”的基础地址。

Stack operations(堆栈操作)： 堆栈操作与代码存取的方式很像。SZM必须通过一个堆栈指针维持它自身的堆栈。这需要两个不同的堆栈，一个用于累加器，一个用于实例数据（“返回堆栈”）。当执行PUSH和POP指令时，SZM必须内部地修改堆栈指针。

Management functions(管理函数)： 管理函数的分类不但包含装载“高速存储寄存器”的确定的程序而且包含任务转换和调用固件块的程序。

需要指出的是，内存存取模块保持少数“高速存储寄存器”来管理当前任务的“工作指针”。这加速了内存操作和封装进硬件内部的从属指针的表示。“高速存储寄存器”由函数LzsMemInitTask初始化，这个函数在控制命令kIpInitTask执行期间由RT-内核调用（见5.5.1.5节）。对于这个应用，RT-内核从LZS将实例句柄或段号传递到内存存取模块上。LZS将通过段号计算物理地址并将结果放到“高速存储寄存器”中。主实例的代码和数据段的地址可通过它的实例句柄来决定。堆栈段的段号（实例和AE-堆栈）和外部累加器在LZS中的段号是完全相同的。对于上述函数和存取方法的执行，内存存取模块对于一个确定的目标系统来说需要下列“高速存储寄存器”：

```
LZSBYTE LZSFAR* RegCS;
LZSBYTE LZSFAR* RegDS;
LZSBYTE LZSFAR* RegIP;          /* current IP */
LZSDWORD LZSFAR* RegInstSP;     /* current SP Inst-Stack (RetStack) */
LZSDWORD LZSFAR* RegAESP;       /* current SP AE-Stack */
LZSBYTE LZSFAR* RegExtAE;       /* Baseaddr. external AE */
LZSBYTE LZSFAR* RegExtAEs;      /* Baseaddr. external AEs */
```

当在两个POU之切换时寄存器RegCS（代码段寄存器）和RegDS（数据段寄存器）会被修改。指令指针和堆栈指针必须被内存存取模块连续不断地记到最近的位置（就像在常用操作系统中一样）。只要任务存在只有指向外部累加器（RegExtAE）的指针是保持不变的。当任务切换的时候“高速存储寄存器”的值必须分别还原。这将由函数LzsMemSaveTask和LzsMemRestoreTask完成，这两个函数由RT-内核在控制命令kIpInitTask和kIpRestoreTask的执行期间调用（见5.5.1.5）。在调试版本中为了识别堆栈溢出和下溢（见头文件MEMACCSS.H），维持除“高速存储寄存器”外的附加变量是可行的。需要考虑的是在PC中作为段的部分并指向段的指针的值是不够的。作为替代，我们通常必须使用FAR指针，因为当分配段的时候（标准=段:0004）MALLOC函数一般不能处理偏移量=0的指针。这使得它在SZM中必须保持所有的指针并且当任务切换的时候保存它。在8051上，用2个字节表示一个指针是足够的。然而，当移植到特殊的硬件上的软件超过64k时需要扩大指针，这是为了给一个“存储体号”分配地址。

5.5.3.2 存取数据类型

一般地，几个内存存取函数（Near，Far，Code）必须支持每种数据类型bit,byte,word,dword和any。这使得移

植Intel代码处理器的数据格式（小端字节顺序）到一个通过目标系统支持的数据格式成为可能。直到8051，小端字节顺序（下载格式）和大端字节顺序（C编译器期望的模式）才可以转换字和双字的存取。

如果字节，字和双字变量被编址，为了计算变量的绝对地址，把RT-内核提供的OFFSET16加入到段基础地址中是很易容的。

为了编址位变量，操作数OFFSET16必须根据不同情况解释。既然那样，只有第0到12位用于编址（限制在8k内），高13到15位指定在此变量中使用位变量的位置。在访问此变量之前需要将些变量与#01FFF与操作，以去掉高13到15位。

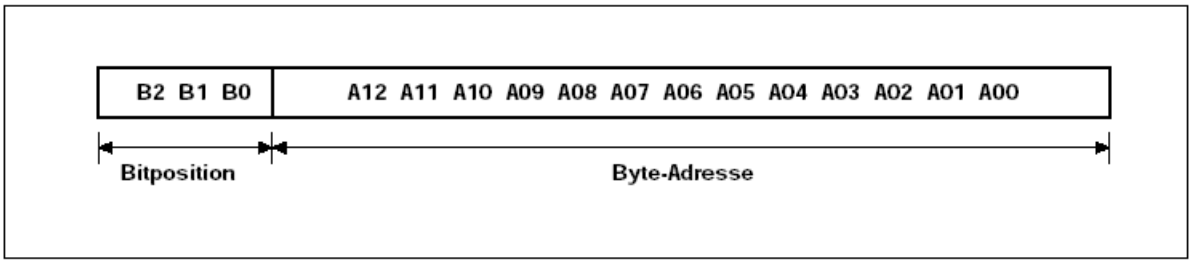


图87 位地址的结构

关于“any”变量的特殊情况是他们的内容不能被RT-内核直接处理。作为数据类型“any”的各自的目标源，SZM必须使用一个叫做“external AE”(外部AE)的数据段，这个数据段的大小在下载程序的时候提交给LZS。指向这个“external AE”的指针必须被SZM在一个内部“高速存储寄存器”中保持。与数据类型“any”有关，OFFSET16适用于用户定义数据类型。第一个信息必须是变量的长度，第二个为变量值。

```
WORD UserVarSize;              // number of bytes
BYTE UserVar[UserVarSize];     // byte sequence
```

长度值包含了变量的总字节数，包括第一个WORD的长度值自己。

例：
```
WORD Size = 6
BYTE Data1
BYTE Data2
BYTE Data3
BYTE Data4
```

5.5.3.3 堆栈段管理

由于在不同的目标系统上的多种多样的指针表示法，堆栈段的绝对大小在下载程序/工程的时候是不会提交的，但是会提交在实例中可能的输入的数量和AE栈。这样将允许LZS计算和分配必要的内存区域。与常用实现比较，被SZM使用的堆栈段应该得到更大的从低到高的地址，并带有“初始化指针”指向段的开始地址。因此，LZS的常用段管理函数同样可用于这些堆栈。堆栈溢出则可由段自身的长度字段来识别（由于性能的原因，这只在调试版本中是合理的）。下图为标准解决方案中堆栈段的布局：

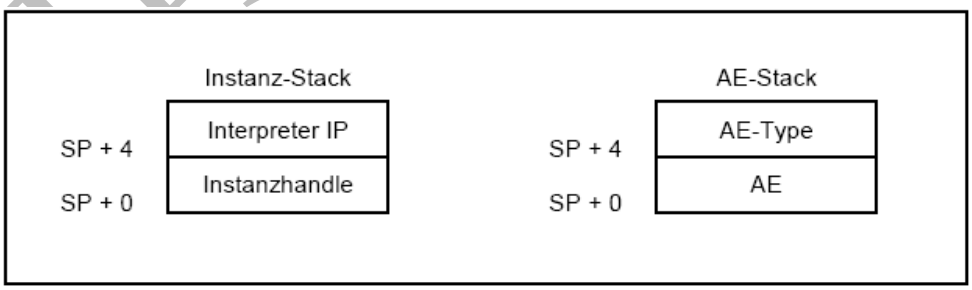


图88 堆栈输入布局

当当前实例的状态被保存后，每当函数LzsMemRetPushInst调用的时候，对于实例句柄和指令指针的堆栈输入必须生成。为了保存当前累加器和它的类型，RT-内核必须在其后调用两次函数LzsMemCrPushAE。当第一次调用函数时，累加器的当前内容被传递，第二次调用函数时传递AE的当前类型。这样分离成两次独立的函数调用使参数传递变得简单（在8051上，只能通过寄存器直接传递类型为DWORD的参数）。

此外，类型信息可以被省略并且堆栈的大小在移植到小系统时会减小。AE类型对于处理函数是不必要的但

是对监听是必要的。

5.5.3.4 段头结构

所有被LZS管理的段都有一个统一的段头，这些段头的详细描述可在LZS文档中找到。所有的段头的生成由代码生成器完成并且在下载期间被加载到PLC中，段头结构如下所示：

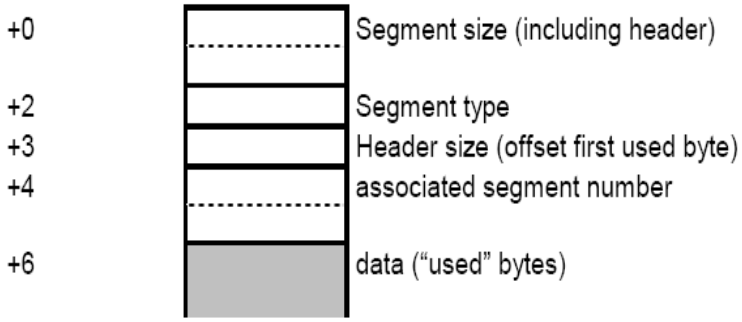


图89 段头结构

在第3个位置的段头大小对SZM特别重要。有了这个值，在段中第一个有用的字节的偏移就可以确定。这个值使得SZM对代码段中的第一条指令编址（见函数LzsMemCodeSetPos）并且决定堆栈段的第一个字节。

5.5.3.5 SZM 函数说明

LZS或SZM需要为RT-内核提供下述函数。许多函数用一个偏移量作为其参数。这通常指向段的绝对起始点，例如，包含段头的起始点。

void LzsMemSetInstSeg (tLzshInst hInst_p);

这个函数为当前代码段和数据段设置内部“高速存储寄存器”。这个函数将在实体控制代码的帮助下决定代码段和数据段的适当的基础地址。

void LzsMemNearSetBit (tLzsBitAdr Addr_p, BOOL BitVal_p);

void LzsMemNearSetByte (tLzsOffs16 Addr_p, BYTE ByteVal_p);

void LzsMemNearSetWord (tLzsOffs16 Addr_p, WORD WordVal_p);

void LzsMemNearSetDword (tLzsOffs16 Addr_p, DWORD DwordVal_p);

void LzsMemNearSetAny (tLzsOffs16 Addr_p);

这些函数将一个变量写入到当前正在运行中的POU的当前数据段中。第一个参数指定了它在当前数据段中的偏移量。段的基础地址必须由SZM内部地提供。第二个参数（除了LzsMemNearSetAny）是被写入变量的值。

BOOL LzsMemNearGetBit (tLzsBitAdr Addr_p);

BYTE LzsMemNearGetByte (tLzsOffs16 Addr_p);

WORD LzsMemNearGetWord (tLzsOffs16 Addr_p);

DWORD LzsMemNearGetDword (tLzsOffs16 Addr_p);

void LzsMemNearGetAny (tLzsOffs16 Addr_p);

这些函数从当前正在运行的POU中的当前数据段中读取一个变量。参数（除了LzsMemNearGetAny）指定了它在当前数据段中的偏移量。基础地址必须由SZM内部地提供。

void LzsMemFarSetBit (tLzshInst hInst_p, tLzsBitAdr Addr_p, BOOL BitVal_p);

void LzsMemFarSetByte (tLzshInst hInst_p, tLzsOffs16 Addr_p, BYTE ByteVal_p);

void LzsMemFarSetWord (tLzshInst hInst_p, tLzsOffs16 Addr_p, WORD WordVal_p);

void LzsMemFarSetDword (tLzshInst hInst_p, tLzsOffs16 Addr_p, DWORD DwordVal_p);

void LzsMemFarSetAny (tLzshInst hInst_p, tLzsOffs16 Addr_p);

这些函数写入一个变量到任意段中。第一个参数指定了将被寻址的数据段的句柄。第二个参数是它在段中的偏移量。段的地址将被SZM根据句柄内部地决定。第三个参数（除了LzsMemFarSetAny）是将要写入的变量的值。

BOOL LzsMemFarGetBit (tLzshInst hInst_p, tLzsBitAdr Addr_p);

BYTE LzsMemFarGetByte (tLzshInst hInst_p, tLzsOffs16 Addr_p);

WORD LzsMemFarGetWord (tLzshInst hInst_p, tLzsOffs16 Addr_p);

DWORD LzsMemFarGetDword (tLzshInst hInst_p, tLzsOffs16 Addr_p);

void LzsMemFarGetAny (tLzshInst hInst_p, tLzsOffs16 Addr_p);

这些函数从任意段中读取一个变量。第一个参数指定了将要被寻址的段的句柄。第二个参数是它在段中的偏移量。段的地址由SZM根据句柄内部地决定。

void LzsMemCodeSetPos (tLzsOffs16 IPAddr_p);

这个函数为RT-内核设置内部的“指令指针”。参数是当前代码段中的指令指针的偏移量。代码段的基础地址必须通过函数LzsMemSetInstSeg内部地存储在一个变量中。这个函数用于进入到一个POU（程序启动或调用，在更新段的时候联合函数LzsMemSetInstSem使用。）中并且用于跳转指令的实现。此外，这个函数在一个周期结束时被RT-内核调用，用于复位指令指针使之指向程序的开头。自从段结构被隐藏到RT-内核（在SZM中内存存取的封装）中后，Null就作为一个偏移量被使用。这种特殊情况（IPAddr = = 0）必须由函数配置。在这种情况下，指令指针将被自动地设置为指向段头后的第一条指令。

BYTE LzsMemCodeGetInstr (void);

BYTE LzsMemCodeGetByte (void);

WORD LzsMemCodeGetWord (void);

DWORD LzsMemCodeGetDword (void);

void LzsMemCodeGetAny (void);

这些函数从当前代码段读取数据。这些数据被SZM的内部“指令指针”寻址。这个指针必须在每次代码存取后自动增加。因此它通常指向下一个将要被读取的字节。为了初始化指令指针，会使用函数LzsMemSteInstSeg和LzsMemCodeSetPos。在现在的RT-内核实现中，函数LzsMemCodeGetInstr和LzsMemCodeGetByte完全相同的。

void LzsMemCrPushAE (DWORD AEVal_p);

DWORD LzsMemCrPopAE (void);

这些函数将累加器入栈并且分别出栈。SZM必须管理堆栈段和栈指针。如此，SZM在一个任务的初始化期间提供堆栈段的句柄（见5.5.1.5节函数LzsMemInitTast）。独立于它的实际的数据类型，累加器通常作为一个32位数值入栈。堆栈结构的注释可在5.5.3.3节中的到。

void LzsMemRetPushInst (tLzshInst hInst_p);

tLzshInst LzsMemRetPopInst (void);

这些函数将实例数据入栈并且分别出栈。这使得嵌套POU成为可能（返回栈）。SZM必须管理堆栈段和栈指针。如此，SZM在一个任务的初始化期间提供堆栈段的句柄（见5.5.1.5节函数LzsMemInitTast）。这个将被入栈的根-POU的实例句柄将被RT-内核传递给SZM。这个指令指针必须被SZM在它自己的栈内入栈。函数LzsMemRetPopInst必须执行下列操作：

-恢复指令指针

-为代码段和数据段恢复“高速存储寄存器”

-从栈中读取实例句柄并且传递给RT-内核

为了为代码段的数据段恢复“高速存储寄存器”，这两个段的地址都将被入栈。作为一个选择，LZS可以要么从栈中读取实例句柄要么根据段表决定这个地址。为了最优化运行时间或内存需求这些程序中的一个可以关闭。堆栈段结构的注释可在5.5.3.3节中找到。

BOOL LzsMemCallFrmwMod (tLzsFrmwModNr FrmwModNr_p);

这个函数允许RT-内核调用一个固件块（固件-FB或固件函数）。参数指定了固件块号。在一个PC上，头文件包含了哪个号数被分配给了哪个固件块。这个头文件必须被包括到代码生成器中。返回值TRUE标志了一个固件块的错误的释放执行。返回值FALSE表示RT-值使程序中止执行并且返回给LZS。更多固件块执行的结节见5.5.6.1节。

void LzsMemInitTask (tLzsMemTask FAR *pMemTask_p);

函数LzsMemInitTask对任务的“高速存储寄存器”进行初始化。一个指向类型为tLzsMemTask的结构体变量的指针作为参数被传递（此结构体变量的说明见5.5.1.5节）。这个结构体包含了程序-POU的句柄（为决定当前代码段和数据段），堆栈段的句柄（实例堆栈和AE-栈），和外部AE的句柄（为处理“any”数据类型）。基于这些句柄，“高速存储寄存器”可被下列地址（指针值）初始化：

-当前代码段基地址

-当前数据段基地址

-指向程序-POU的代码段内的第一亲指令的指令指针

-指向实例堆栈栈顶的指针

-指向AE-栈栈顶的指针

-指向外部AE段开始部分的指针

注意指令指针必须被设置为指向一条可执行的指令代码（在代码段头后）。RT-内核不明确地调用函数LzsMemCodeStePos。从SZM的角度来讲，在任何时刻都只有一个任务在执行。

void LzsMemSaveTask (tLzsMemTask FAR *pMemTask_p);

void LzsMemRestoreTask (tLzsMemTask FAR *pMemTask_p);

这些函数用于保存内部“高速存储寄存器”和在任务转换之前和之后恢复它们。一个指向类型为tLzsMemTask的结构体变量的指针作为参数被传递（此结构体变量的说明见5.5.1.5节）。这个结构体包含保存下列地址（指针值）的当前值的元素：

-当前代码段基地址

-当前数据段基地址

-指令指针

-实例栈栈指针

-AE-栈栈指针

-指向外部AE段的指针

从SZM的角度来讲，在任何时刻都只有一个任务在执行。

有了上述函数，RT-内核就可以执行所有代码生成器生成的指令了。它同样也可以在SZM需要的时候进行任务间的转换。另外，RT-内核可以生成它的活动的文本信息（串口或文本窗口）。RT-内核为生成这个指令执行的协议配备了一个重汇编函数（见5.5.4.1节IpGenerateDebugInfo）。这个函数是基于下面将要介绍的函数。然而，如果在RT-内核资源代码的应用期间定义了宏NDEBUG，那么就不需要提供这些函数（见5.5.7.2节发布的版本）。

WORD LzsMemGetCodePos (void);

这个函数读取指令指针的偏移量，这个指令指针被SZM内部地管理（在当前代码段内的偏移量）。这个偏移量用于显示当前代码地址（显示模式：“segmenthandle:offset”）。如果传递了一个无效的偏移量，这将导致重汇编程序的无效输出，但是不会影响RT-内核的指令的执行。

BYTE LzsMemDumpCode (WORD CodeOffs_p);

这个函数从当前代码段读取一个字节并返回。与其它所有的LzsMemCodeGet_xxx函数相比，这个函数不必改变SZM的当前的指令指针。这个函数用于显示RT-内核的一个16进制的跳转的执行。一个无效的返回值只会导致重汇编程序的无效输出，但是不会影响RT-内核的指令的执行。

5.5.3.6 决定段地址

内存存取模块（SZM）的标准实现使用了函数LzsSegGetAddr来将段句柄转换为物理段地址。这个函数由LZS提供。它的接口声明如下所示：

tLzsMemPtr LzsSegGetAddr (tLzshInst hInst_p, BYTE SegType_p);

对于代码段和数据段，参数hInst_p是适当的POU的实例的句柄。对于堆栈段和外部累加器，必须使用LZS的段号。参数SegType_p是期望的类型常量。内存存取模块使用下列段类型：

```
typedef enum {
    kLzsSegCode = 0x00,         // Code segment
    kLzsSegData = 0x01,         // Data segment of an Instance
    kLzsSegInit = 0x02,         // Initialization data segment
    kLzsSegInstStack = 0x80,    // instance stack segment (return stack)
    kLzsSegAEStack = 0x81,      // AE Stack segment
    kLzsSegExtAE = 0x82         // Segment of external AE
} tLzsSegType;
```

函数的返回值类型为tLzsMemPrt并且指定了相应段的起始地址。指针表示法取决于使用的目标系统（XDTAT-Ptr,FAR-Ptr,等）。

5.5.4 RT-内核特殊功能描述

5.5.4.1RT-内核重汇编函数

在测试版本中，OPENPCS-RT-内核为一条指令执行协议的生成提供了一个集成的重汇编程序。这个重汇编

程序被函数IpGenerateDebugInto执行。但是，这个函数只在RT-内核资源代码（见5.5.7.2节）的应用期间没有定义宏NDEBUG的情况下才可用。在发布版本，由于编译程序指令，整个IpGenerateDebugInfo函数和它的所有的函数调用都将不被编译。另一个使用重汇编程序的前提是SZM中的函数LzsMemGetCodePos和LzsMemDumpCode的执行（见5.5.3.5节）。

无论何时一条指令的操作码被读取，这个重汇编程序都会被RT-内核IpCycle的主函数调用。然后重汇编程序将显示代码的地址（显示模式：“segmenthandle:offset”）、整个指令的16进制跳转（这包括操作数）、和简单文本的标志符作为重汇编程序的操作码。此后，RT-内核的主函数调用当前指令的执行函数。在指令执行完成后，显示的信息将被当前AE和AEs的内容填充。下列为输出协议的格式：

0020:0005 58 06 00　　LDS_NEAR_1 ->　　AE.b=00,　　AE'.b=64

0020:0008 88　　ADD_1 ->　　AE.b=64,　　AE'.?=?

AE和AEs的显示格式取决于它们的当前数据类型。当前数据类型也同样会被显示。可能的数据类型缩写如下所示：

? = undefined,

f = bit (Flag),

b = byte,

w = word,

d = double word

重汇编程序为输出使用宏LZSTRACE。这个宏为RT-内核的宏控制版本而分解为一个C函数printf的调用。这意味着串口必须在RT-内核开始前就被正确地初始化。另一方面，DLL-版本为输出使用函数trace。这个函数必须作为一个头文件被包含进RT-内核工程中（TRACE.C）。函数trace根据API-函数OutputDebugString在一个特殊的输出窗口中实现输出。

5.5.4.2 RT-内核测试支持

在重汇编程序的输出函数基础上，OPENPCS-RT-内核支持指令IP_OP_BREAK，这条指令对测试来说是特别保留的。这条指令使RT-内核中断程序执行并且返回到LZS。LZS可以识别中断指令是否通过结构体tIpTaskData(见5.5.2.1节)中的元素m_StatusLZS来执行。自从IP_OP_BREAK被作为一个1字节指令来执行（不带任何操作数），它将可以在设置一个断点后用于临时地重写一个不同的操作数。当断点到达的时候，最初的指令将被恢复。

有另一种方式中断这个可用于调试用途的RT-内核。一个异步断点可以通过函数IpSetBreakRequest来请求。这使依赖外部事件的中断可以实现（物理控制器输入）。函数IpSetBreakRequest的进一步的信息见5.5.2.2节。

跟踪AE的函数还没有定义或实现。

所有进一步调试的功能，例如：强迫输入和输出，必须在LZS中实现。

5.5.5 RT-内核指令结构

对于RT-内核指令的结构，参考OpenPCS文档。对于IEC-1131-3标准的U-code-OpenPCS-通用中间代码。

5.5.6 RT-内核的修改

下列章节介绍了提供OPENPCS-RT-内核的能力的不同的可能性。然而每一次RT-内核的改进都在完成编程系统的代码生成器的移植的前提下，因为RT-内核只可以执行通过代码生成器系统生成的程序。

5.5.6.1 固件块的包含

通过包含固件块，RT-内核可以更有效地被移植为指定目标硬件的属性。通过控制器的处理器直接地执行固件块比RT-内核执行一个相似的代码片段更快。固件块可以选择作为一个函数或者一个函数块（FB）来执行。通过分析“标准”POU，例如：通过RT-内核执行的POU，每个固件块的实例都有它自己的数据段。

固件块是实时系统的一部分因此拥有完全的控制器特性。RT-内核的连接固件块调用的接口是SZM函数LzsMemCallFrmwMod。下列为这个函数的接口的声明（见5.5.3.5节）：

BOOL LzsMemCallFrmwMod (tLzsFrmwModNr FrmwModNr_p);

这个函数把固件块的序号作为参数。在PC上，有一个头文件指定了哪个固件块被指定的是哪个序号。这个头文件被包含在代码生成器内。RT-内核可以通过指令IP_OP_CALxx从扩展地址段读这个块号（指令声明见5.5.7.3

节）。

在RT-内核与固件块之间的参数的传递通过特别的实例的数据段实现。这个数据段通过分析所有其他POU的数据段并由代码生成器分配。至于固件块的实现，这意味着所有的参数都分配在函数的“外面”并且不能根据它们的名称直接地存取。此外，固件块因为实现的缘故根据指针来存取参数也是不可以的。这是因为内存接口的性能必须被重视。比如，在8051上，当固件-POU的代码由使用Motorola格式（大端字节顺序）的C-编译器生成时，对一个WORD-指针的存取将得到错误的结果，因为编程系统的代码生成器使用的是Intel格式（小端字节顺序）。

为了存取固件-POU的数据段，必须使用内存存取模块的函数，例如：RT-内核使用的相同的函数。在调用函数LzsMemCallFrmwMod前，RT-内核将选择固件-POU的数据段作为“当前”数据段。这使得这个固件块可使用“快速”函数LzsMemNear_xxx用于内存存取。这确保了RT-内核和固件使用一个独特的数据表示。作为通过这个“当前”数据段间接地寻址的结果，当调用固件块时传递参数给固件块是多余的。

RT-内核可以在一个固件块的执行期间通过RT-内核函数IpSetErrorCode来标记实时错误。这个函数“存储”任意错误代码到RT-内核的错误寄存器中。由于这个错误代码将不再被RT-内核处理，所以如果发生错误函数LzsMemCallFrmwMod将返回FALSE。只有这个返回值能令RT-内核中止指令执行并且返回到LZS。然后错误寄存器中的值将被上传到LZS。如果函数LzsMemCallFrmwMod在固件块调用后返回TRUE到RT-内核，内核将继续执行程序。在这种情况下，由IpSetErrorCode设置的错误代码将作为一个“警告”报告给LZS。

下列代码片段以标准FB“CTU”（计数器）为例说明了固件块的执行。在本例中的数据段的布局只阐明了POU的参数存取的方式。对于数据段的实际的结构，请参考CBE的文档！

```
//***************************************************************************
// Standard-IEC-Function CTU
//***************************************************************************
BOOL CTU (void)
{
/* ---------------------------------------------------------------------
IEC-Prototype: CTU(CU, R, PV, Q, CV)
                    +----------+
                    | CTU      |
                    |          |
        BOOL --->CU       Q|--- BOOL
        BOOL ---  |R         |
        INT ---   |PV    CV|--- INT
                    |          |
                    +----------+
在调用固件块之前，RT-内核设置“当前”数据段到块的DS以便调用。
在这个块中，通过使用对于NEAR-寻址的函数来存取它的“自己的”
数据。下列为数据段的结构
Header: InstDS+0    WORD    SegSize
                    +2      BYTE    SegType
                    +3      BYTE    HeaderSize
                    +4      WORD    HandleCodeSeg
Data:   InstDS+6    INT     PV
                    +8      BOOL    CU
                    +9      BOOL    R
                    +10     INT     CV
                    +12     BOOL    Q
--------------------------------------------------------------------- */
#define PVmax 0x7FFE
SINT PV;
BOOL CU;
BOOL R;
SINT CV;
BOOL Q;
        PV = LzsMemNearGetWord (6);
        CU = LzsMemNearGetBit (8);
        R = LzsMemNearGetBit (9);
        CV = LzsMemNearGetWord(10);
        if ( !R )
        {
                if ( CU & (CV < PVmax) )
                        CV++;
        }
        else
        CV = 0;
        Q = (BOOL) (CV >= PV);
```

```
    LzsMemNearSetWord(10, CV);
    LzsMemNearSetBit (12, Q);
    return (TRUE);
}
```

5.5.6.2 指令设置的改进

对OPENPCS-RT-内核的指令设置的修改只可以在编程系统的代码生成器移植的时候才可以进行。为了包含附加的指令或者减少对存储空间的实现的全部的容量，改进是必要的。RT-内核为每条指令都有一个执行函数。这些函数调用时没有任何的参数，但是它们向RT-内核报告执行的状态。如果返回值是TRUE，下一条指令将执行，如果返回值是FALSE，RT-内核将中止程序的执行并且返回到LZS。第一这样的执行函数都使用累加器AE或AES中的一个作为资源和操作的目标。这两种累加器都以类型为tIpAE（UNION）的全局变量来处理并且可通过标识符RegAE_g和RegAEs_g来存取：

```
RegAE_g.f,    RegAEs_g.f      // Bit (Flag)
RegAE_g.b,    RegAEs_g.b      // Byte
RegAE_g.w,    RegAEs_g.w      // Word
RegAE_g.d,    RegAEs_g.d      // Dword
```

若要加入一条新的RT−内核指令需要写成以下步骤：

-完成相应指令的执行函数。函数的类型必须是BOOL IpIP_OP_xxx(void)；

-将执行函数的原型加入到文件IP_CMDS.H中；

-文件IP_CMDS.H中定义一个操作符码；

-在函数IpCycle中插入一条新的case分支；

-对于测试版本，还要在函数IpGenerateDebugInfo中插入一个适当的case分支。

为了减少指令变量的数量，需要在上述位置移除声明。

5.5.7 RT-内核的编译

5.5.7.1 目标系统在文件 ODK_PRJ.H 中的依赖移植

头文件ODK_PRJ.H（OpenPCS工程定义）定义了抽象信号（用于端口）并且使得目标系统的依赖移植成为可能，例如：涉及到内存模式或编译转换（#pragma）。随着这个头文件被包含于RT-内核的所有的资源文件中，一个贯穿整个工程的信号的独一无二的定义和转换将被保证。这个头文件包含了少数对不同的开发环境的定义分支。

下列定义是相应依赖平台的抽象概念。资源文件将参考这些定义。

LZSNEAR：	用于RT-内核的常用变量。它被以DATA定义并且使得移动变量可以进8051的内部存储区域，保证效率和快速数据存取。
LZSFAR：	用于指向任何类型的段的指针。在8051上，它被定义为XDATA。
LZSCONST：	用于常数数组并且在8051上被定义为CODE。这防止当进入一个函数时数组被拷贝为XDATA。
LZSEXPORT：	这个修改被定义为_far_pascal_export并且对RT-内核的DLL版本很重要。对于微控制器，这个修改为空。
LZSEXPORT32：	这个修改对RT-内核的32位DLL版本很重要。对于微控制器，这个修改为空。
LZSHUGE：	用于指向超大数据区域的指针（如：对C167）。
TRACE：	这个宏执行一个跟踪协议。在5.5.4.1节中有说明。

所有这里没有列出来的定义涉及的开发环境都不在资源文件中使用。

5.5.7.2 RT-内核的编译选择

不仅可以将RT-内核移植到不同的目标系统上，而且还可以在一个平台上编译不同的RT-内核版本，这需要根据有条件的编译资源文件。为了这个目的，当编译资源文件时将定义宏_DEBUG或NDEBUG。注意这些宏的定义对整个工程来说必须是唯一的，如：所有的资源文件必须以相同的定义编译（生成文件中的全局定义）。基于这些宏的定义，这两个内核版本就可以被编译：

NDEBUG：为了执行RT-内核的核心函数只需要编译那些代码段。下列这些函数和代码段不用编译：

-重汇编程序（和这些函数的调用）；
-所有被TRACE执行的协议输出；
-函数IpCycle中的代码，这些代码拷贝用于显示堆栈的状态确定的结构体元素；
-当标准-SZM使用时，不会生成代码来检查当前段边界或显示堆栈的内容。重汇编程序使用的函数LzsMemGetCodePos和LzsMemDumpCode也不会被编译。
这个选项生成一个有最小代码容量和最大速度的RT-内核。

5.6 UCODE-IEC-1131-3 通用中间代码

5.6.1 指令执行

U-Code使得在不同的目标系统上执行抽象的，独立于硬件的中间代码成为可能。中间代码由长度为1字节的操作码组成。对于不同的指令，可有一个或多个操作码。对指令约定的详细信息见5.6.7节。

由于操作码内容的长度为1字节，一个简单的转换-声明就可以决定函数调用的执行，如：对每个操作码都有一个case分支。

5.6.1.1 第二个库

必须的指令的数量不可以被限制在255条。因此出现了第二个指令库。它们由2个字节来表示。第一个字节仍然是第一个库里的指令（IP_OP_EXT1）。表示下一个字节为需要解释执行的指令。这些2字节指令多为IEC标准函数。

所有以IP_OP_*开头的操作码属于第一个库（1字节指令）。

所有以IP_EX_*开头的操作码属于第二个库（2字节指令）。它们通常被加上U-code IP_OP_EXT1的前言。

5.6.2 解释程序的数据处理

U-Code的工作像一个堆栈自动机，如：全部数据首先被加载指令入栈，在指令执行前结合栈中的第一个和第二个元素。这个操作的结果被返回栈并且可能被进一步地处理或写回内存。在IL指令执行期间，逻辑指令（AND Var）被代码生成器分离为a)将操作码入栈的加载指令b)指令自身的执行。

IL指令通常把一个累加器（AE在德语中为”aktuelles Ergebnis”, “当前结果“）与一个操作数结合；这个操作的结果被返回到累加中（AE）。这意味着这个累加器同时是栈顶元素，而操作数是栈的第二个元素。所有栈的其它的层只用于分解带括号的操作。

下列示例举例说明了AWL指令怎样被代码生成器分离成“加载”和“结合”指令，这些指令可被RT-内核执行。

IL source code	Code generator / Interpreter
LD Var1	AE := Var1
ADD 10	AEs := 10
	AE := AE + AEs
ST Var2	Var2 := AE

程序语言IL的一个特征是累加器（AE）没有特定的数据类型但是它的当前数据类型取决于上一个被处理的数据的类型。

理论上，一个函数块的全部的结构、数据或仅仅一个实例都可能被存储进这个累加器。但是，存储一个函数块的实例，由于一些现实的原因将在编译后期被排除。对于所有其它的存取，只有一个地址将被存储进这个累加器中。这就是为什么只有数据类型BOOL,BYTE,WORD和DWORD可以出现在这个累加器中——或者数据类型ANY。在这种情况下，长度将被存储在第一、二个字节。

5.6.3 段头结构

所有被LZS管理的段都有统一的头。这个头由代码生成器生成并且有如下结构：

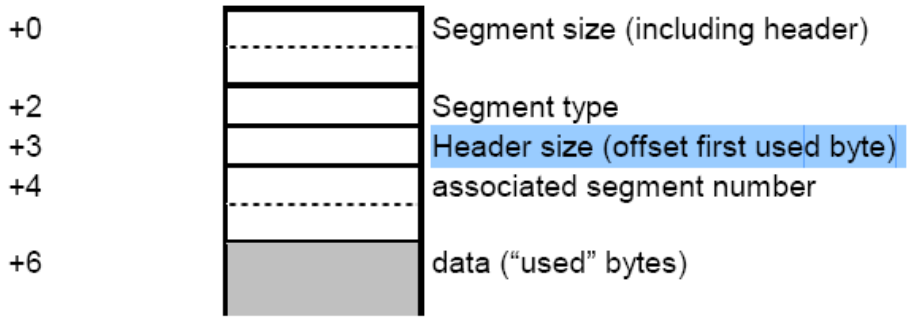


图90 段头结构

目前，有两种分配这些段的方式。

1.在编程系统内进行连接。将为每个函数块分配一个引用数据段。这些段将被正确地实例化，如：所有实例特性信息都将被写入（例如：块的调用的引用，外部引用等）。这个引用数据段产生实例数据段。所有的实例数据段和解释程序代码将被传输到控制器。实例可以通过他们的段句柄调用。因此可以决定调用块的实例数据段。在每个段区域的开头，有一个对代码开头的引用。这些代码可以与实例数据段的引用一起执行。

2.另一种方式是，连接和实例化可以根据额外的信息从程序代码生成器独立地完成。在这种情况下，这些信息必须对实例树是可理解的。实例树描述了哪个实例必须被分配和在哪种条件下可以进行实例化。然后初始数据可以被写入引用数据段。这个引用数据段是这个新分配的实例数据段的基础。一个外部的表指定了可被使用的外部变量（存取到I/O或其他全局数据）。实例数据段需要使这些数据是可理解的。

5.6.4 寻址数据类型

通常，一些内存存取函数（Near,Far,Code）必须支持每个数据类型bit,byte,word,dword和any。这使得移植Intel数据格式的代码生成器（小端字节顺序）到目标系统支持的数据格式成为可能。直到8051被关注后，小端字节顺序（下载格式）和大端字节顺序（C编译器期望的格式）可以为word和dword存取而转换。如果byte,word和dword变量被寻址，那么RT-内核提供的OFFSET16将被简单地加入到段基础地址中，这是为了计算变量的绝对地址。

5.6.4.1 直到 OpenPCS3.3 版本的位寻址

为了寻址位变量，叫做OFFSET16的操作数必须被不同地解释。在那种情况下，只有0到12位被用于寻址位变量（范围为8k），而高3位13到15标记变量中被使用的位置。因此13到15位必须在存取变量前与#01FFFh作与操作。

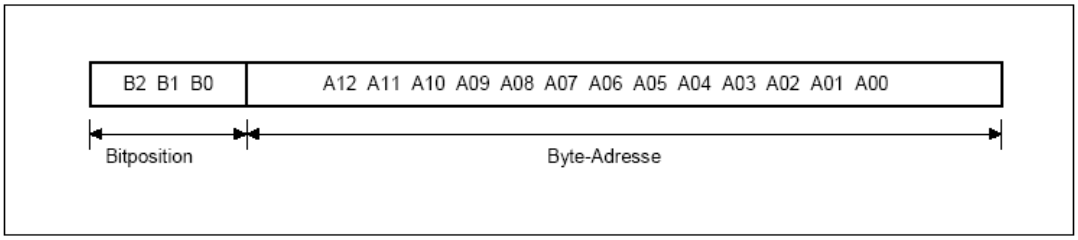


图91 位地址的结构

例：

位地址为12.6（12的偏移量的第6位）被映射为Hex：0xCC,BIN=1100 0000 0000 1100

位地址为27.3（27的偏移量的第3位）被映射为Hex：0x73,BIN=0111 0000 0000 1011

5.6.4.2 OpenPCS3.4 以上版本的位寻址

上述寻址方式的一个缺点是数据段的最大容量只有8k。另一方面64k的段都可被管理。由于8k的段是足够的，位的位置从偏移量移动到了段内。在本地DS中的位通常在位置0（位由字节映射）。这意味着位的位置通常是知道的。位的位置只在使用BYREF寻址的时候被需要，因为I/O地址也是可存取的并且位可在一个字节的任何位置。段号将像下列文档一样被标记处理。

简而言之，DS现在可有64k大，但是只有最大8k的段。

5.6.4.3 寻址 ANY 数据类型

这里我们使用ANY的概念来概述结构体、数组和字符串。IEC1131仅仅允许以纯整数来加载这些数据类型，但是实现是不同的。

与数据类型"ANY"有关，OFFSET16适用于用户定义数据类型。第一个信息是变量的长度，第二个是变量值。

```
WORD        UserVarSize;            /*number bytes following */
BYTE        UserVar[UserVarSize];   /*Byte stream*/
```

这个长度值包含了变量的总长度，包括第一个表示长度值的WORD变量。

Example: WORD Size = 6

BYTE Data1

BYTE Data2

BYTE Data3

BYTE Data4

存取的例子将在下面说明。

5.6.5 数组处理

为了计算一个数组元素的地址，将使用到指令IP_OP_SELCT_ARRAY_IND。为了计算这个地址的所有必要的信息都可以在数组头读取，这个数组头有如下结构：

```
LZSSTATIC LZSWORD    wArraySize;     /* Size of the array */
LZSSTATIC LZSWORD    wSizeOfType;    /* Size of one element */
LZSSTATIC LZSBYTE    bType;          /* Type information */
LZSSTATIC LZSBYTE    bDimension;     /* dimensions of array */
LZSSTATIC LZSSINT    sLowerRange;    /* lower limit of act. dimension */
LZSSTATIC LZSSINT    sUpperRange;    /* upper limit of act. dimension */
LZSSTATIC LZSSINT    sIndex;         /* Index of act. dimension */
```

在数组的"常规"头之后，是对每个指定的低层和高层索引范围的维的描述符号和将被寻址的元素的索引。这个索引范围值是常数，然而在通过叫做IP_OP_ST_xx的确定的指令存取前当前元素的索引将被修改。在"常规"头之后的第一个符号描述了数组的最"内部的"维（如：递增最快的维），最后一个（最高地址）描述了最"外面的"维。

有了这些信息，就可以检查每个维的索引是否在索引范围内。如果测试成功，那么这个想得到的元素的地址可以被计算出来。计算需要根据下列方案将描述符号从"内部的"移动到"外面的"维。

```
/* initialize counters for calculation*/
wAbsElmt = 0;
wDimElmt = 1;
/* process dimension from inside to the outside */
for (i=0; i<bDimensions; i++)
{
    /* Read information of actual Dimension */
    sLowerRange = (LZSSINT)LZS_MEM_FAR_GET_WORD(hArryDS_p, wArrayOffs_p);
    wArrayOffs_p += LZS_SIZEOF_WORD;
    sUpperRange = (LZSSINT)LZS_MEM_FAR_GET_WORD(hArryDS_p, wArrayOffs_p);
    wArrayOffs_p += LZS_SIZEOF_WORD;
    sIndex = (LZSSINT)LZS_MEM_FAR_GET_WORD(hArryDS_p, wArrayOffs_p);
    wArrayOffs_p += LZS_SIZEOF_WORD;
    /* Check ArrayIndex */
    if ( (sIndex < sLowerRange) || (sIndex > sUpperRange) )
    {
        Error_g = kIpArryIndexInvalid;
        fErrFlag = LZSTRUE;
        break;
    }
    /* computation of the array pointer */
    wAbsElmt += (LZSWORD) Abs((LZSINT)(sIndex - sLowerRange))*wDimElmt ;
    wDimElmt *= (LZSWORD) Abs((LZSINT)(sUpperRange - sLowerRange)) + 1;
}
```

变量wDimElmt保持了一个维的元素号，变量wAbsElmt保持了数组中想得到的元素的"线性的"号；这个号

只适用于数组元素。

根据当前对齐可以在头和数组元素间填入字节。下一个例子举例说明了在一个2维数组中填入4字节对齐的字节。

```
// ARRAY [0..3, 0..2] of BYTES
0x20, 0x00,          // wUserVarSize = 32
0x01, 0x00,          // wSizeOfType (Bytefeld)
kIpBit8,             // bType
0x02,                // bDimension
0x00, 0x00,          // sLowerRange
0x02, 0x00,          // sUpperRange
0x00, 0x00,          // sIndex
0x00, 0x00,          // sLowerRange
0x03, 0x00,          // sUpperRange
0x00, 0x00,          // sIndex
0x00, 0x00,          // fill bytes
// data
[0, 0], [0, 1], [0, 2]     // Offset 20 … 22
[1, 0], [1, 1], [1, 2]     // Offset 23 … 25
[2, 0], [2, 1], [2, 2]     // Offset 26 … 28
[3, 0], [3, 1], [3, 2]     // Offset 29 … 32
```

下一个示例将举例说明一个将被解释程序使用的3维数组的内存布局。让我们以坐标[1,5,7]来寻址这个元素。在上述算法执行之后，变量wAbsElmt的值是16。这是这个想得到的元素在数组元素中的偏移。由于寻址一个数组元素要参考数据段的起始地址，这个值必须通过数组的偏移，数组头的大小和这3个维来增加。这产生了一个46的绝对偏移。这个值将通过IP_OP_SELCT_ARRAY_IND指令与实例句柄一起存储到引用累加器。然后可通过IP_OP_LD_IND_1指令读取这个值。

```
// ARRAY [0..2, 3..5, 6..8]
// element to be addressed: [1, 5, 7]
0x33, 0x00, // wUserVarSize = 51
0x01, 0x00, // wSizeOfType (Bytefeld)
kIpBit8, // bType
0x03, // bDimension
0x06, 0x00, // sLowerRange
0x08, 0x00, // sUpperRange
0x07, 0x00, // sIndex
0x03, 0x00, // sLowerRange
0x05, 0x00, // sUpperRange
0x05, 0x00, // sIndex
0x00, 0x00, // sLowerRange
0x02, 0x00, // sUpperRange
0x01, 0x00, // sIndex
// order of array elements in memory
[0, 3, 6], [0, 3, 7], [0, 3, 8] // Offset 24..26
[0, 4, 6], [0, 4, 7], [0, 4, 8] // Offset 27..29
[0, 5, 6], [0, 5, 7], [0, 5, 8] // Offset 30..32
[1, 3, 6], [1, 3, 7], [1, 3, 8] // Offset 33..35
[1, 4, 6], [1, 4, 7], [1, 4, 8] // Offset 36..38
[1, 5, 6], [1, 5, 7], [1, 5, 8] // Offset 39..41
[2, 3, 6], [2, 3, 7], [2, 3, 8] // Offset 42..44
[2, 4, 6], [2, 4, 7], [2, 4, 8] // Offset 45..47
[2, 5, 6], [2, 5, 7], [2, 5, 8] // Offset 48..50
PROGRAM TEST_01
VAR
FELD : ARRAY [0..2, 3..5, 6..8] of BYTE;
END_VAR
    LD FELD [1,5,7]
    ST FELD [1,3,6]
END_PROGRAM
```

Data segment

```
    0000 : 4B 00 01 06 02 00 1B 00
    0008 : 01 00 00 03 00 00 02 00
    0010 : CC CC 03 00 05 00 CC CC
    0018 : 06 00 08 00 CC CC 00 00
    0020 : 00 00 00 00 00 00 00 00
    0028 : 00 00 00 00 00 00 00 00
    0030 : 00 00 00 00 00 00 00 00
    0038 : 00
```

Data segment resolved:

```
    Header: 4B 00 01 06 02 00
```

ArrayHeader: Arraysize 1B 00
TypeSize 01 00
Type Info 00
Dimension 03
LowerRange 00 00
UpperRange 02 00
Index CC CC
LowerRange 03 00
UpperRange 05 00
Index CC CC
LowerRange 06 00
UpperRange 08 00
Index CC CC
Data: 27 * 00

Code:

```
CREATE_REFERENCE 00 06
PUSH_REF
ADD_OFFSET 00 0E
LD_CONST_4 00 00 00 01
ST_IND_4
ADD_OFFSET 00 0C
LD_CONST_4 00 00 00 05
ST_IND_4
ADD_OFFSET 00 0C
LD_CONST_4 00 00 00 07
ST_IND_4
POP_REF
SELCT_ARRAY_IND
LD_IND_ANY
PUSH_AE
CREATE_REFERENCE 00 06
PUSH_REF
ADD_OFFSET 00 0E
LD_CONST_4 00 00 00 01
ST_IND_4
ADD_OFFSET 00 0C
LD_CONST_4 00 00 00 03
ST_IND_4
ADD_OFFSET 00 0C
LD_CONST_4 00 00 00 06
ST_IND_4
POP_REF
SELCT_ARRAY_IND
POP_AE
ST_IND_1
RET
```

例1 完整数组存取

5.6.6 结构体处理

结构体的依据所有元素的正确大小的长度（1WORD）如下所示。

```
TYPE
    MYSTRUCT : Struct
        FIRST : int;
        SECOND : BYTE;
        THIRD : int;
    END_STRUCT;
END_TYPE
VAR
        SampleStruct : mystruct;
        y : Byte;
        SampleS2 : mystruct;
END_VAR
```

Data segment:

```
0000 : 15 00 01 06 02 00 07 00
0008 : 00 00 00 00 00 00 07 00
0010 : 00 00 00 00 00
```

Interpretation:

Header: 15 00 01 06 02
Length Structure SampleStruct: 07 00
DataStructure: 00 00 00 00 00 (WORD, BYTE, WORD).
Y 00

Length Structure SampleS2: 07 00
DataStructure: 00 00 00 00 00 (WORD, BYTE, WORD).

Accessing structures:
LD_NEAR_1 00 0A
ST_NEAR_1 00 0D

Copying structures:
LD_NEAR_ANY 00 06
ST_NEAR_ANY 00 0E

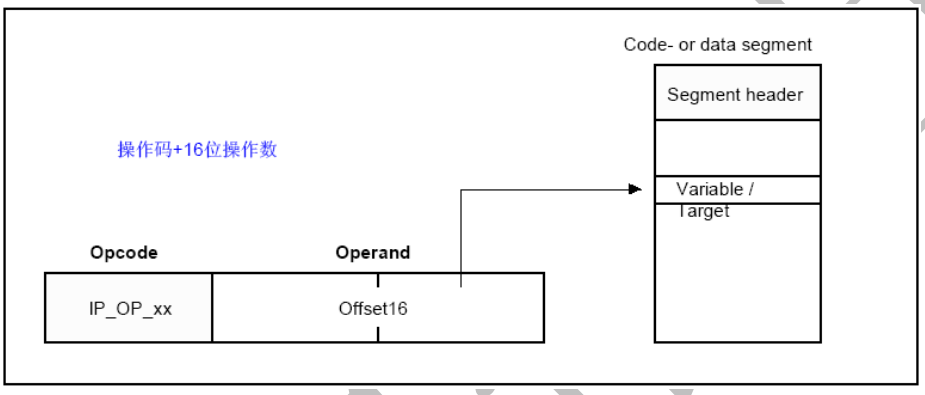
5.6.7 解释程序指令的结构

OPENPCS解释程序支持的指令由抽象的，硬件独立的中间代码组成，这些代码可以在不同的目标系统上执行。这些中间代码由1字节长的操作码组成。根据不同的指令，一个或多个操作数后可跟额外的字节。在下面的说明中，将介绍在不同的指令类型的操作数类型的结构上调用的详细信息。

5.6.7.1 操作数类型说明

解释程序指令使用的多数操作数都属于下列类别中的一种：

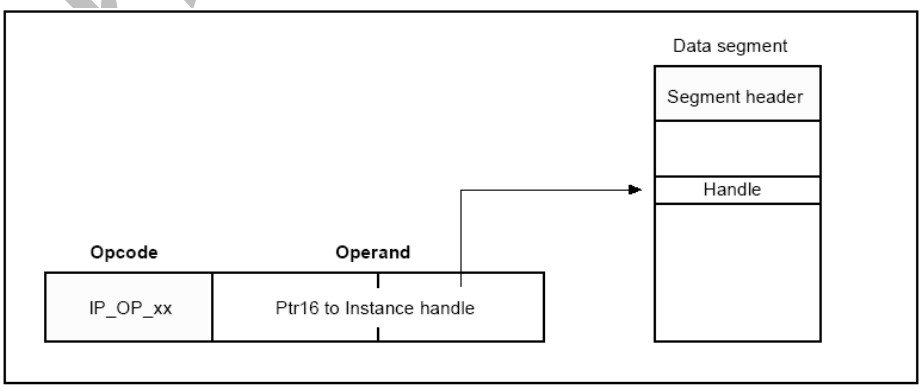
16位绝对偏移



一个操作数指定的“16位绝对偏移”以字节表明了一个变量或跳转位置与特定段之间的距离。这个值包括了段头的大小。这意味着，这个值可被直接用于寻址（如：段基地址加上16位偏移）。

Example:
PROGRAM z
VAR
x : byte ;
END_VAR
ld x
END_PROGRAM
Data segment
0000 : 08 00 01 06 02 00 00
Note: Bytes 1 - 6 Header !
11 LD_NEAR_1 00 06
例2 寻址16位绝对偏移

实例句柄**的16位指针**



在函数块的实例化期间（一个FB的多种“派生”），句柄不能被直接地存储进代码段（所有的实例使用相同的代码段）。作为替代，句柄必须存储进每个实例的数据段。因此，操作数是一个指向实例的“当前”数据段的

指针（如，16位偏移）。这里就是可以找到句柄的地方。句柄是一个16位的值并且帮助寻址一个不同POE的代码或数据段。

```
PROGRAM z
VAR
    inst_fb1 : TEST_FB1;
    x : int;
    y : int;
END_VAR
cal inst_fb1(in1:=x,in_out1:=x|y:=out1)
END_PROGRAM
```

Data segment

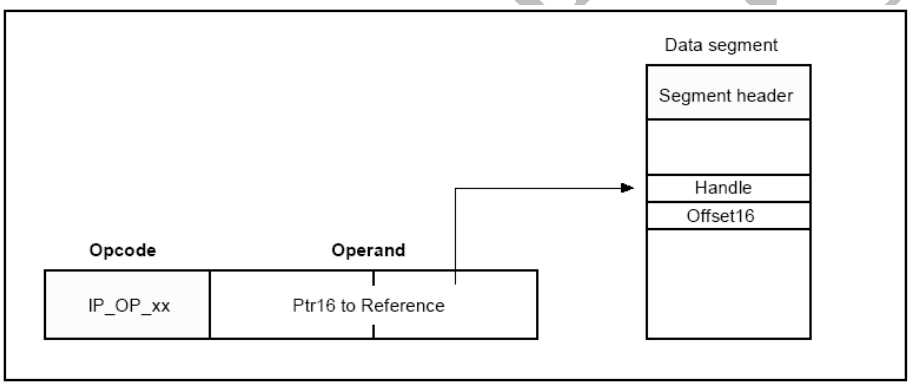
```
    0000 : 0E 00 01 06 02 00 00 00
    0008 : 00 00 00 00 00 00
```

Code:

```
    CREATE_REFERENCE 00 0A
    ST_REFERENCE 00 06 00 08
    LD_NEAR_2 00 0A
    ST_FAR_2 00 06 00 0E
    CAL 00 06
    LD_FAR_2 00 06 00 06
    ST_NEAR_2 00 0C
    RET
```

例3 实例句柄的16位指针

引用的16位指针



在函数块的实例化期间（一个FB的多种“派生”），引用不能被直接地存储进代码段（所有的实例使用相同的代码段）。作为替代，引用必须存储进每个实例的数据段。因此，操作数是一个指向实例的“当前”数据段的指针（如，16位偏移）。这里就是可以找到引用的地方。这个引用由一个为了寻址一个不同的POE的数据段的16位句柄和”far”数据段中的一个变量的16位偏移量构成。

```
FUNCTION_BLOCK test_fb1
VAR_INPUT
    in1 : int ;
END_VAR
VAR_IN_OUT
    in_out1 : int ;
END_VAR
VAR_OUTPUT
    out1 : int ;
END_VAR
ld in1
st out1
st in_out1
END_FUNCTION_BLOCK
```

Data segment

```
    0000 : 10 00 01 06 02 00 00 00
    0008 : C7 27 00 00 00 00 00 00
```

Code:

```
    LD_NEAR_2 00 0E
    ST_NEAR_2 00 06
    ST_BYREF_2 00 08
RET
```

例4 引用的16位指针

5.6.7.2 UCODE 指令列表

下表列出了所有UCODE指令的定义，在各自的本地的代码编译器中加入了实现状态（X=支持，/=不支持，O=通过回调到解释程序执行，P=取决于参数类型）。如果更多的UCODE指令的文档包含在本手册中，将有一个以逗号“see §”的参考。这个表被以UCODE指令名称存储。对于用数字表示的存储清单，见头文件ip_opc.h。

UCODE (hex)	UCODE (dec)	UCODE instruction	ncc167h	ncc68k	Ncc86	Ncc Pent Opt	nccPPC	nccArm7 ARM	nccTMS 32OVC33	ColdFire	see §
(ext)11	(ext)17	IP_EX_ABS_1_SGN	X	X	X	X	X	X	X	X	
(ext)12	(ext)18	IP_EX_ABS_2_SGN	X	X	X	X	X	X	X	X	
(ext)13	(ext)19	IP_EX_ABS_4_SGN	X	X	X	X	X	X	X	X	
(ext)98	(ext)152	IP_EX_ABS_8	/	/	/	/	/	/	/	/	
(ext)99	(ext)153	IP_EX_ABS_8_SGN	/	/	/	/	/	/	/	/	
(ext)9a	(ext)154	IP_EX_ABS_DOUBLE	/	/	O	X	/	X	/	/	
(ext)14	(ext)20	IP_EX_ABS_FLOAT	X	X	X	X	/	X	X	X	
(ext)1d	(ext)29	IP_EX_ACOS	X	X	X	O	X	X	X	X	
(ext)b6	(ext)182	IP_EX_ACOS_DOUBLE	/	/	O	O	/	X	/	/	
(ext)79	(ext)121	IP_EX_ADD_8	/	/	/	/	/	/	/	/	
(ext)7a	(ext)122	IP_EX_ADD_8_SGN	/	/	/	/	/	/	/	/	
(ext)7b	(ext)123	IP_EX_ADD_DOUBLE	/	/	O	X	/	X	/	/	
(ext)bf	(ext)191	IP_EX_ADD_DT	/	/	/	/	/	/	/	/	
(ext)76	(ext)118	IP_EX_AND_8	/	/	/	/	/	/	/	/	
(ext)1c	(ext)28	IP_EX_ASIN	X	X	O	O	X	X	X	X	
(ext)b5	(ext)181	IP_EX_ASIN_DOUBLE	/	/	O	O	/	X	/	/	
(ext)1e	(ext)30	IP_EX_ATAN	X	X	O	O	X	X	X	X	
(ext)b7	(ext)183	IP_EX_ATAN_DOUBLE	/	/	O	O	/	X	/	/	
(ext)4a	(ext)74	IP_EX_CONCAT	/	/	O	O	X	/	X	X	
(ext)1a	(ext)26	IP_EX_COS	X	X	O	O	X	X	X	X	
(ext)b3	(ext)179	IP_EX_COS_DOUBLE	/	/	O	O	/	X	/	/	
(ext)1f	(ext)31	IP_EX_CREATE_BITREF_DIR_ BYREF	/	/	/	/	/	/	/	/	5.6.7.3.32
(ext)4c	(ext)76	IP_EX_DELETE	/	/	O	O	/	/	X	X	
(ext)82	(ext)130	IP_EX_DIV_8	/	/	/	/	/	/	/	/	
(ext)83	(ext)131	IP_EX_DIV_8_SGN	/	/	/	/	/	/	/	/	
(ext)84	(ext)132	IP_EX_DIV_DOUBLE	/	/	O	X	/	X	/	/	
(ext)c4	(ext)196	IP_EX_END_PARA	/	/	/	/	/	/	/	/	
(ext)91	(ext)145	IP_EX_EQ_8	/	/	X	X	/	/	/	/	
(ext)50	(ext)80	IP_EX_EXCHANGE_IO_READ	/	/	/	/	/	/	/	/	
(ext)4f	(ext)79	IP_EX_EXCHANGE_IO_WRITE	/	/	/	/	/	/	/	/	
(ext)15	(ext)21	IP_EX_EXP	X	X	O	O	X	X	X	X	
(ext)ae	(ext)174	IP_EX_EXP_DOUBLE	/	/	O	O	/	X	/	/	
(ext)20	(ext)32	IP_EX_EXPT_1	X	X	O	O	X	X	X	X	
(ext)21	(ext)33	IP_EX_EXPT_1_SGN	X	X	O	O	X	X	X	X	
(ext)22	(ext)34	IP_EX_EXPT_2	X	X	O	O	X	X	X	X	
(ext)23	(ext)35	IP_EX_EXPT_2_SGN	X	X	O	O	X	X	X	X	

UCODE (hex)	UCODE (dec)	UCODE instruction	ncc167h	ncc68k	Ncc86	Ncc Pent Opt	nccPPC	nccArm7 ARM	nccTMS 32OVC33	ColdFire	see §
(ext)24	(ext)36	IP_EX_EXPT_4	X	X	O	O	X	X	X	X	
(ext)25	(ext)37	IP_EX_EXPT_4_SGN	X	X	O	O	X	X	X	X	
(ext)9e	(ext)158	IP_EX_EXPT_8	/	/	/	/	/	/	/	/	
(ext)9f	(ext)159	IP_EX_EXPT_8_SGN	/	/	/	/	/	/	/	/	
(ext)a0	(ext)160	IP_EX_EXPT_DOUBLE	/	/	O	O	/	X	/	/	
(ext)dd	(ext)221	IP_EX_EXPT_DOUBLE_1	/	/	O	O	/	X	/	/	
(ext)dc	(ext)220	IP_EX_EXPT_DOUBLE_1_SGN	/	/	O	O	/	X	/	/	
(ext)df	(ext)223	IP_EX_EXPT_DOUBLE_2	/	/	O	O	/	X	/	/	
(ext)de	(ext)222	IP_EX_EXPT_DOUBLE_2_SGN	/	/	O	O	/	X	/	/	
(ext)e1	(ext)225	IP_EX_EXPT_DOUBLE_3	/	/	/	/	/	/	/	/	
(ext)e0	(ext)224	IP_EX_EXPT_DOUBLE_3_SGN	/	/	/	/	/	/	/	/	
(ext)e3	(ext)227	IP_EX_EXPT_DOUBLE_4	/	/	O	O	/	X	/	/	
(ext)e2	(ext)226	IP_EX_EXPT_DOUBLE_4_SGN	/	/	O	O	/	X	/	/	
(ext)e5	(ext)229	IP_EX_EXPT_DOUBLE_8	/	/	/	/	/	/	/	/	
(ext)e4	(ext)228	IP_EX_EXPT_DOUBLE_8_SGN	/	/	/	/	/	/	/	/	
(ext)e8	(ext)232	IP_EX_EXPT_DOUBLE_DOUBLE	/	/	/	/	/	X	/	/	
(ext)e6	(ext)230	IP_EX_EXPT_DOUBLE_FLOAT	/	/	/	/	/	X	/	/	
(ext)e7	(ext)231	IP_EX_EXPT_DOUBLE_REAL48	/	/	/	/	/	/	/	/	
(ext)26	(ext)38	IP_EX_EXPT_FLOAT	X	X	O	O	X	X	X	X	
(ext)c5	(ext)197	IP_EX_EXPT_REAL48	/	/	/	/	/	/	/	/	
(ext)4e	(ext)78	IP_EX_FIND	/	/	O	O	X	/	X	X	
(ext)88	(ext)136	IP_EX_GE_8	/	/	/	/	/	/	/	/	
(ext)89	(ext)137	IP_EX_GE_8_SGN	/	/	/	/	/	/	/	/	
(ext)8a	(ext)138	IP_EX_GE_DOUBLE	/	/	O	X	/	X	/	/	
(ext)85	(ext)133	IP_EX_GT_8	/	/	/	/	/	/	/	/	
(ext)86	(ext)134	IP_EX_GT_8_SGN	/	/	/	/	/	/	/	/	
(ext)87	(ext)135	IP_EX_GT_DOUBLE	/	/	O	X	/	X	/	/	
(ext)c2	(ext)194	IP_EX_INPUT_PARA_C	/	/	/	/	/	/	/	/	
(ext)4b	(ext)75	IP_EX_INSERT	/	/	O	O	/	/	X	X	
(ext)5c	(ext)92	IP_EX_LD_BYREF_8	/	/	X	X	/	X	/	/	
(ext)5e	(ext)94	IP_EX_LD_CONST_8	/	/	X	X	/	X	/	/	
(ext)c7	(ext)199	IP_EX_LD_DIR_BYREF_1	/	/	X	X	/	/	/	/	5.6.7.3.32
(ext)c8	(ext)200	IP_EX_LD_DIR_BYREF_2	/	/	X	X	/	/	/	/	5.6.7.3.32
(ext)c9	(ext)201	IP_EX_LD_DIR_BYREF_4	/	/	X	X	/	/	/	/	5.6.7.3.32
(ext)ca	(ext)202	IP_EX_LD_DIR_BYREF_8	/	/	/	/	/	/	/	/	5.6.7.3.32
(ext)cb	(ext)203	IP_EX_LD_DIR_BYREF_ANY	/	/	X	X	/	/	/	/	5.6.7.3.32
(ext)c6	(ext)198	IP_EX_LD_DIR_BYREF_BIT	/	/	X	X	/	/	/	/	5.6.7.3.32
(ext)5b	(ext)91	IP_EX_LD_FAR_8	/	/	X	X	/	X	/	/	
(ext)5d	(ext)93	IP_EX_LD_IND_8	/	/	X	X	/	X	/	/	
(ext)5a	(ext)90	IP_EX_LD_NEAR_8	/	/	X	X	/	X	/	/	
(ext)61	(ext)97	IP_EX_LDN_BYREF_8	/	/	/	/	/	X	/	/	

UCODE (hex)	UCODE (dec)	UCODE instruction	ncc167h	ncc68k	Ncc86	Ncc Pent Opt	nccPPC	nccArm7 ARM	nccTMS 32OVC33	ColdFire	see §
(ext)cd	(ext)205	IP_EX_LDN_DIR_BYREF_1	/	/	X	X	/	/	/	/	5.6.7.3.32
(ext)ce	(ext)206	IP_EX_LDN_DIR_BYREF_2	/	/	X	X	/	/	/	/	5.6.7.3.32
(ext)cf	(ext)207	IP_EX_LDN_DIR_BYREF_4	/	/	X	X	/	/	/	/	5.6.7.3.32
(ext)d0	(ext)208	IP_EX_LDN_DIR_BYREF_8	/	/	/	/	/	/	/	/	5.6.7.3.32
(ext)cc	(ext)204	IP_EX_LDN_DIR_BYREF_BIT	/	/	X	X	/	/	/	/	5.6.7.3.32
(ext)60	(ext)96	IP_EX_LDN_FAR_8	/	/	/	/	/	X	/	/	
(ext)62	(ext)98	IP_EX_LDN_IND_8	/	/	/	/	/	X	/	/	
(ext)5f	(ext)95	IP_EX_LDN_NEAR_8	/	/	/	/	/	X	/	/	
(ext)6a	(ext)106	IP_EX_LDNS_BYREF_8	/	/	/	/	/	X	/	/	
(ext)f2	(ext)242	IP_EX_LDNS_DIR_BYREF_1	/	/	X	X	/	/	/	/	5.6.7.3.32
(ext)f3	(ext)243	IP_EX_LDNS_DIR_BYREF_2	/	/	X	X	/	/	/	/	5.6.7.3.32
(ext)f4	(ext)244	IP_EX_LDNS_DIR_BYREF_4	/	/	X	X	/	/	/	/	5.6.7.3.32
(ext)f5	(ext)245	IP_EX_LDNS_DIR_BYREF_8	/	/	X	/	/	/	/	/	5.6.7.3.32
(ext)f1	(ext)241	IP_EX_LDNS_DIR_BYREF_BIT	/	/	X	X	/	/	/	/	5.6.7.3.32
(ext)69	(ext)105	IP_EX_LDNS_FAR_8	/	/	/	/	/	X	/	/	
(ext)6b	(ext)107	IP_EX_LDNS_IND_8	/	/	/	/	/	X	/	/	
(ext)68	(ext)104	IP_EX_LDNS_NEAR_8	/	/	/	/	/	X	/	/	
(ext)65	(ext)101	IP_EX_LDS_BYREF_8	/	/	X	X	/	X	/	/	
(ext)67	(ext)103	IP_EX_LDS_CONST_8	/	/	X	X	/	X	/	/	
(ext)ec	(ext)236	IP_EX_LDS_DIR_BYREF_1	/	/	X	X	/	/	/	/	5.6.7.3.32
(ext)ed	(ext)237	IP_EX_LDS_DIR_BYREF_2	/	/	X	X	/	/	/	/	5.6.7.3.32
(ext)ee	(ext)238	IP_EX_LDS_DIR_BYREF_4	/	/	X	X	/	/	/	/	5.6.7.3.32
(ext)ef	(ext)239	IP_EX_LDS_DIR_BYREF_8	/	/	/	/	/	/	/	/	5.6.7.3.32
(ext)f0	(ext)240	IP_EX_LDS_DIR_BYREF_ANY	/	/	X	X	/	/	/	/	5.6.7.3.32
(ext)eb	(ext)235	IP_EX_LDS_DIR_BYREF_BIT	/	/	X	X	/	/	/	/	5.6.7.3.32
(ext)64	(ext)100	IP_EX_LDS_FAR_8	/	/	X	X	/	X	/	/	
(ext)66	(ext)102	IP_EX_LDS_IND_8	/	/	X	X	/	X	/	/	
(ext)63	(ext)99	IP_EX_LDS_NEAR_8	/	/	X	X	/	X	/	/	
(ext)8b	(ext)139	IP_EX_LE_8	/	/	/	/	/	/	/	/	
(ext)8c	(ext)140	IP_EX_LE_8_SGN	/	/	/	/	/	/	/	/	
(ext)8d	(ext)141	IP_EX_LE_DOUBLE	/	/	O	X	/	X	/	/	
(ext)47	(ext)71	IP_EX_LEFT	/		O	O	X	X	X	X	
(ext)46	(ext)70	IP_EX_LEN	/		O	O	X	X	X	X	
(ext)3f	(ext)63	IP_EX_LIMIT_1	/	/	/	/	/	/	/	/	
(ext)40	(ext)64	IP_EX_LIMIT_1_SGN	/	/	/	/	/	/	/	/	
(ext)41	(ext)65	IP_EX_LIMIT_2	/	/	/	/	/	/	/	/	
(ext)42	(ext)66	IP_EX_LIMIT_2_SGN	/	/	/	/	/	/	/	/	
(ext)43	(ext)67	IP_EX_LIMIT_4	/	/	/	/	/	/	/	/	
(ext)44	(ext)68	IP_EX_LIMIT_4_SGN	/	/	/	/	/	/	/	/	
(ext)a8	(ext)168	IP_EX_LIMIT_8	/	/	/	/	/	/	/	/	
(ext)a9	(ext)169	IP_EX_LIMIT_8_SGN	/	/	/	/	/	/	/	/	

UCODE (hex)	UCODE (dec)	UCODE instruction	ncc167h	ncc68k	Ncc86	Ncc Pent Opt	nccPPC	nccArm7 ARM	nccTMS 32OVC33	ColdFire	see §
(ext)aa	(ext)170	IP_EX_LIMIT_DOUBLE	/	/	/	/	/	/	/	/	
(ext)45	(ext)69	IP_EX_LIMIT_FLOAT	/	/	/	/	/	/	/	/	
(ext)17	(ext)23	IP_EX_LN	X	X	O	O	X	X	X	X	
(ext)b0	(ext)176	IP_EX_LN_DOUBLE	/	/	O	O	/	X	/	/	
(ext)18	(ext)24	IP_EX_LOG	X	X	O	O	X	X	X	X	
(ext)b1	(ext)177	IP_EX_LOG_DOUBLE	/	/	O	O	/	X	/	/	
(ext)8e	(ext)142	IP_EX_LT_8	/	/	/	/	/	/	/	/	
(ext)8f	(ext)143	IP_EX_LT_8_SGN	/	/	/	/	/	/	/	/	
(ext)90	(ext)144	IP_EX_LT_DOUBLE	/	/	O	X	/	X	/	/	
(ext)38	(ext)56	IP_EX_MAX_1	X	X	O	O	X	X	X	X	
(ext)39	(ext)57	IP_EX_MAX_1_SGN	X	X	O	O	X	X	X	X	
(ext)3a	(ext)58	IP_EX_MAX_2	X	X	O	O	X	X	X	X	
(ext)3b	(ext)59	IP_EX_MAX_2_SGN	X	X	O	O	X	X	X	X	
(ext)3c	(ext)60	IP_EX_MAX_4	X	X	O	O	X	X	X	X	
(ext)3d	(ext)61	IP_EX_MAX_4_SGN	X	X	O	O	X	X	X	X	
(ext)a5	(ext)165	IP_EX_MAX_8	/	/	/	/	/	/	/	/	
(ext)a6	(ext)166	IP_EX_MAX_8_SGN	/	/	/	/	/	/	/	/	
(ext)52	(ext)82	IP_EX_MAX_ANY	/	/	/	/	/	X	X	X	
(ext)a7	(ext)167	IP_EX_MAX_DOUBLE	/	/	O	O	/	X	/	/	
(ext)3e	(ext)62	IP_EX_MAX_FLOAT	X	X	O	O	X	X	X	X	
(ext)49	(ext)73	IP_EX_MID	/		O	O	/	/	X	X	
(ext)31	(ext)49	IP_EX_MIN_1	X	X	O	O	X	X	X	X	
(ext)32	(ext)50	IP_EX_MIN_1_SGN	X	X	O	O	X	X	X	X	
(ext)33	(ext)51	IP_EX_MIN_2	X	X	O	O	X	X	X	X	
(ext)34	(ext)52	IP_EX_MIN_2_SGN	X	X	O	O	X	X	X	X	
(ext)35	(ext)53	IP_EX_MIN_4	X	X	O	O	X	X	X	X	
(ext)36	(ext)54	IP_EX_MIN_4_SGN	X	X	O	O	X	X	X	X	
(ext)a2	(ext)162	IP_EX_MIN_8	/	/	/	/	/	/	/	/	
(ext)a3	(ext)163	IP_EX_MIN_8_SGN	/	/	/	/	/	/	/	/	
(ext)51	(ext)81	IP_EX_MIN_ANY	/	/	/	/	/	X	X	X	
(ext)a4	(ext)164	IP_EX_MIN_DOUBLE	/	/	O	O	/	X	/	/	
(ext)37	(ext)55	IP_EX_MIN_FLOAT	X	X	O	O	X	X	X	X	
(ext)27	(ext)39	IP_EX_MOD_1	/	X	O	O	X	X	X	X	
(ext)28	(ext)40	IP_EX_MOD_1_SGN	/	X	O	O	X	X	X	X	
(ext)29	(ext)41	IP_EX_MOD_2	/	X	O	O	X	X	X	X	
(ext)2a	(ext)42	IP_EX_MOD_2_SGN	/	X	O	O	X	X	X	X	
(ext)2b	(ext)43	IP_EX_MOD_4	/	X	O	O	X	X	X	X	
(ext)2c	(ext)44	IP_EX_MOD_4_SGN	/	X	O	O	X	X	X	X	
(ext)9b	(ext)155	IP_EX_MOD_8	/	/	/	/	/	/	/	/	
(ext)9c	(ext)156	IP_EX_MOD_8_SGN	/	/	/	/	/	/	/	/	
(ext)9d	(ext)157	IP_EX_MOD_DOUBLE	/	/	/	/	/	/	/	/	

UCODE (hex)	UCODE (dec)	UCODE instruction	ncc167h	ncc68k	Ncc86	Ncc Pent Opt	nccPPC	nccArm7 ARM	nccTMS 32OVC33	ColdFire	see §
(ext)2d	(ext)45	IP_EX_MOD_FLOAT	/	/	/	/	/	/	/	/	
(ext)2e	(ext)46	IP_EX_MOVE_1	/	/	/	/	/	/	/	/	
(ext)2f	(ext)47	IP_EX_MOVE_2	/	/	/	/	/	/	/	/	
(ext)30	(ext)48	IP_EX_MOVE_4	/	/	/	/	/	/	/	/	
(ext)a1	(ext)161	IP_EX_MOVE_8	/	/	/	/	/	/	/	/	
(ext)7f	(ext)127	IP_EX_MUL_8	/	/	/	/	/	/	/	/	
(ext)80	(ext)128	IP_EX_MUL_8_SGN	/	/	/	/	/	/	/	/	
(ext)81	(ext)129	IP_EX_MUL_DOUBLE	/	/	O	X	/	X	/	/	
(ext)92	(ext)146	IP_EX_NE_8	/	/	X	X	/	X	/	/	
(ext)b9	(ext)185	IP_EX_NEG_1	X	X	X	X	X	X	X	X	
(ext)ba	(ext)186	IP_EX_NEG_2	X	X	X	X	X	X	X	X	
(ext)bb	(ext)187	IP_EX_NEG_4	X	X	X	X	X	X	X	X	
(ext)bc	(ext)188	IP_EX_NEG_8	/	/	/	/	/	/	/	/	
(ext)b8	(ext)184	IP_EX_NEG_BIT	X	/	/	/	X	X	X	X	
(ext)be	(ext)190	IP_EX_NEG_DOUBLE	/	/	O	X	/	X	/	/	
(ext)bd	(ext)189	IP_EX_NEG_FLOAT	O	X	O	X	X	X	X	X	
(ext)ff	(ext)255	IP_EX_NOP	/	/	/	/	/	X	/	/	
(ext)02	(ext)2	IP_EX_NOT_1	X	X	X	X	X	X	X	X	
(ext)03	(ext)3	IP_EX_NOT_2	X	X	X	X	X	X	X	X	
(ext)04	(ext)4	IP_EX_NOT_4	X	X	X	X	X	X	X	X	
(ext)93	(ext)147	IP_EX_NOT_8	/	/	/	/	/	/	/	/	
(ext)01	(ext)1	IP_EX_NOT_BIT	X	X	X	X	X	X	X	X	
(ext)77	(ext)119	IP_EX_OR_8	/	/	/	/	/	/	/	/	
(ext)c3	(ext)195	IP_EX_OUTPUT_PARA_C	/	/	/	/	/	/	/	/	
(ext)75	(ext)117	IP_EX_POP_AE_8	/	/	X	X	/	X	/	/	
(ext)74	(ext)116	IP_EX_PUSH_AE_8	/	/	X	X	/	X	/	/	
(ext)ea	(ext)234	IP_EX_R_DIR_BYREF_BIT	/	/	X	X	/	/	/	/	5.6.7.3.32
(ext)4d	(ext)77	IP_EX_REPLACE	/	/	O	O	/	/	X	X	
(ext)48	(ext)72	IP_EX_RIGHT	/	/	O	O	X	X	X	X	
(ext)0b	(ext)11	IP_EX_ROL_1	X	X	X	X	X	X	X	X	
(ext)0c	(ext)12	IP_EX_ROL_2	X	X	X	X	X	X	X	X	
(ext)0d	(ext)13	IP_EX_ROL_4	X	X	X	X	X	X	X	X	
(ext)96	(ext)150	IP_EX_ROL_8	/	/	/	/	/	/	/	/	
(ext)0e	(ext)14	IP_EX_ROR_1	X	X	X	X	X	X	X	X	
(ext)0f	(ext)15	IP_EX_ROR_2	X	X	X	X	X	X	X	X	
(ext)10	(ext)16	IP_EX_ROR_4	X	X	X	X	X	X	X	X	
(ext)97	(ext)151	IP_EX_ROR_8	/	/	/	/	/	/	/	/	
(ext)e9	(ext)233	IP_EX_S_DIR_BYREF_BIT	/	/	X	X	/	/	/	/	5.6.7.3.32
(ext)54	(ext)84	IP_EX_SEL_1	/	/	/	/	/	/	/	/	
(ext)55	(ext)85	IP_EX_SEL_2	/	/	/	/	/	/	/	/	
(ext)56	(ext)86	IP_EX_SEL_4	/	/	/	/	/	/	/	/	

UCODE (hex)	UCODE (dec)	UCODE instruction	ncc167h	ncc68k	Ncc86	Ncc Pent Opt	nccPPC	nccArm7 ARM	nccTMS 32OVC33	ColdFire	see §
(ext)ab	(ext)171	IP_EX_SEL_8	/	/	/	/	/	/	/	/	
(ext)ac	(ext)172	IP_EX_SEL_8_SGN	/	/	/	/	/	/	/	/	
(ext)57	(ext)87	IP_EX_SEL_ANY	/	/	/	/	/	/	/	/	
(ext)53	(ext)83	IP_EX_SEL_BIT	/	/	/	/	/	/	/	/	
(ext)ad	(ext)173	IP_EX_SEL_DOUBLE	/	/	/	/	/	/	/	/	
(ext)58	(ext)88	IP_EX_SEL_FLOAT	/	/	/	/	/	/	/	/	
(ext)05	(ext)5	IP_EX_SHL_1	X	X	X	X	X	X	X	X	
(ext)06	(ext)6	IP_EX_SHL_2	X	X	X	X	X	X	X	X	
(ext)07	(ext)7	IP_EX_SHL_4	X	X	X	X	X	X	X	X	
(ext)94	(ext)148	IP_EX_SHL_8	/	/	/	/	/	/	/	/	
(ext)08	(ext)8	IP_EX_SHR_1	X	X	X	X	X	X	X	X	
(ext)09	(ext)9	IP_EX_SHR_2	X	X	X	X	X	X	X	X	
(ext)0a	(ext)10	IP_EX_SHR_4	X	X	X	X	X	X	X	X	
(ext)95	(ext)149	IP_EX_SHR_8	/	/	/	/	/	/	/	/	
(ext)19	(ext)25	IP_EX_SIN	X	X	X	X	X	X	X	X	
(ext)b2	(ext)178	IP_EX_SIN_DOUBLE	/	/	O	X	/	X	/	/	
(ext)16	(ext)22	IP_EX_SQRT	X	X	O	X	X	X	X	X	
(ext)af	(ext)175	IP_EX_SQRT_DOUBLE	/	/	O	X	/	X	/	/	
(ext)6e	(ext)110	IP_EX_ST_BYREF_8	/	/	X	X	/	X	/	/	
(ext)d2	(ext)210	IP_EX_ST_DIR_BYREF_1	/	/	X	X	/	/	/	/	5.6.7.3.32
(ext)d3	(ext)211	IP_EX_ST_DIR_BYREF_2	/	/	X	X	/	/	/	/	5.6.7.3.32
(ext)d4	(ext)212	IP_EX_ST_DIR_BYREF_4	/	/	X	X	/	/	/	/	5.6.7.3.32
(ext)d5	(ext)213	IP_EX_ST_DIR_BYREF_8	/	/	/	/	/	/	/	/	5.6.7.3.32
(ext)d6	(ext)214	IP_EX_ST_DIR_BYREF_ANY	/	/	/	/	/	/	/	/	5.6.7.3.32
(ext)d1	(ext)209	IP_EX_ST_DIR_BYREF_BIT	/	/	X	X	/	/	/	/	5.6.7.3.32
(ext)59	(ext)89	IP_EX_ST_DIR_BYREF_STRING	/	/	X	X	/	/	/	/	5.6.7.3.32
(ext)6d	(ext)109	IP_EX_ST_FAR_8	/	/	X	X	/	X	/	/	
(ext)6f	(ext)111	IP_EX_ST_IND_8	/	/	X	X	/	X	/	/	
(ext)6c	(ext)108	IP_EX_ST_NEAR_8	/	/	X	X	/	X	/	/	
(ext)72	(ext)114	IP_EX_STN_BYREF_8	/	/	/	/	/	X	/	/	
(ext)d8	(ext)216	IP_EX_STN_DIR_BYREF_1	/	/	X	X	/	/	/	/	5.6.7.3.32
(ext)d9	(ext)217	IP_EX_STN_DIR_BYREF_2	/	/	X	X	/	/	/	/	5.6.7.3.32
(ext)da	(ext)218	IP_EX_STN_DIR_BYREF_4	/	/	X	X	/	/	/	/	5.6.7.3.32
(ext)db	(ext)219	IP_EX_STN_DIR_BYREF_8	/	/	/	/	/	/	/	/	5.6.7.3.32
(ext)d7	(ext)215	IP_EX_STN_DIR_BYREF_BIT	/	/	X	X	/	/	/	/	5.6.7.3.32
(ext)71	(ext)113	IP_EX_STN_FAR_8	/	/	/	/	/	X	/	/	
(ext)73	(ext)115	IP_EX_STN_IND_8	/	/	/	/	/	X	/	/	
(ext)70	(ext)112	IP_EX_STN_NEAR_8	/	/	/	/	/	X	/	/	
(ext)7c	(ext)124	IP_EX_SUB_8	/	/	/	/	/	/	/	/	
(ext)7d	(ext)125	IP_EX_SUB_8_SGN	/	/	/	/	/	/	/	/	
(ext)7e	(ext)126	IP_EX_SUB_DOUBLE	/	/	O	X	/	X	/	/	

UCODE (hex)	UCODE (dec)	UCODE instruction	ncc167h	ncc68k	Ncc86	Ncc Pent Opt	nccPPC	nccArm7 ARM	nccTMS 32OVC33	ColdFire	see §
(ext)c1	(ext)193	IP_EX_SUB_DT_DT	/	/	/	/	/	/	/	/	
(ext)c0	(ext)192	IP_EX_SUB_DT_TIME	/	/	/	/	/	/	/	/	
(ext)1b	(ext)27	IP_EX_TAN	X	X	O	O	X	X	X	X	
(ext)b4	(ext)180	IP_EX_TAN_DOUBLE	/	/	O	O	/	X	/	/	
(ext)78	(ext)120	IP_EX_XOR_8	/	/	/	/	/	/	/	/	
(ext2)cc	(ext2)204	IP_EX2_ABS_3	/	/	/	/	/	/	/	/	
(ext2)cd	(ext2)205	IP_EX2_ABS_3_SGN	/	/	/	/	/	/	/	/	
(ext2)48	(ext2)72	IP_EX2_ABS_6	/	/	/	/	/	/	/	/	
(ext2)49	(ext2)73	IP_EX2_ABS_6_SGN	/	/	/	/	/	/	/	/	
(ext2)4a	(ext2)74	IP_EX2_ABS_REAL48	/	/	/	/	/	/	/	/	
(ext2)b5	(ext2)181	IP_EX2_ADD_3	/	/	/	/	/	/	/	/	
(ext2)b6	(ext2)182	IP_EX2_ADD_3_SGN	/	/	/	/	/	/	/	/	
(ext2)29	(ext2)41	IP_EX2_ADD_6	/	/	/	/	/	/	/	/	
(ext2)2a	(ext2)42	IP_EX2_ADD_6_SGN	/	/	/	/	/	/	/	/	
(ext2)2b	(ext2)43	IP_EX2_ADD_REAL48	/	/	/	/	/	/	/	/	
(ext2)b2	(ext2)178	IP_EX2_AND_3	/	/	/	/	/	/	/	/	
(ext2)26	(ext2)38	IP_EX2_AND_6	/	/	/	/	/	/	/	/	
(ext2)bb	(ext2)187	IP_EX2_DIV_3	/	/	/	/	/	/	/	/	
(ext2)bc	(ext2)188	IP_EX2_DIV_3_SGN	/	/	/	/	/	/	/	/	
(ext2)32	(ext2)50	IP_EX2_DIV_6	/	/	/	/	/	/	/	/	
(ext2)33	(ext2)51	IP_EX2_DIV_6_SGN	/	/	/	/	/	/	/	/	
(ext2)34	(ext2)52	IP_EX2_DIV_REAL48	/	/	/	/	/	/	/	/	
(ext2)c5	(ext2)197	IP_EX2_EQ_3	/	/	/	/	/	/	/	/	
(ext2)41	(ext2)65	IP_EX2_EQ_6	/	/	/	/	/	/	/	/	
(ext2)50	(ext2)80	IP_EX2_EXPT_3	/	/	/	/	/	/	/	/	
(ext2)51	(ext2)81	IP_EX2_EXPT_3_SGN	/	/	/	/	/	/	/	/	
(ext2)52	(ext2)82	IP_EX2_EXPT_6	/	/	/	/	/	/	/	/	
(ext2)53	(ext2)83	IP_EX2_EXPT_6_SGN	/	/	/	/	/	/	/	/	
(ext2)55	(ext2)85	IP_EX2_EXPT_REAL48_1	/	/	/	/	/	/	/	/	
(ext2)54	(ext2)84	IP_EX2_EXPT_REAL48_1_SGN	/	/	/	/	/	/	/	/	
(ext2)57	(ext2)87	IP_EX2_EXPT_REAL48_2	/	/	/	/	/	/	/	/	
(ext2)56	(ext2)86	IP_EX2_EXPT_REAL48_2_SGN	/	/	/	/	/	/	/	/	
(ext2)59	(ext2)89	IP_EX2_EXPT_REAL48_3	/	/	/	/	/	/	/	/	
(ext2)58	(ext2)88	IP_EX2_EXPT_REAL48_3_SGN	/	/	/	/	/	/	/	/	
(ext2)5b	(ext2)91	IP_EX2_EXPT_REAL48_4	/	/	/	/	/	/	/	/	
(ext2)5a	(ext2)90	IP_EX2_EXPT_REAL48_4_SGN	/	/	/	/	/	/	/	/	
(ext2)5d	(ext2)93	IP_EX2_EXPT_REAL48_8	/	/	/	/	/	/	/	/	
(ext2)5c	(ext2)92	IP_EX2_EXPT_REAL48_8_SGN	/	/	/	/	/	/	/	/	
(ext2)60	(ext2)96	IP_EX2_EXPT_REAL48_DOUBLE	/	/	/	/	/	/	/	/	
(ext2)5e	(ext2)94	IP_EX2_EXPT_REAL48_FLOAT	/	/	/	/	/	/	/	/	
(ext2)5f	(ext2)95	IP_EX2_EXPT_REAL48_REAL48	/	/	/	/	/	/	/	/	

UCODE (hex)	UCODE (dec)	UCODE instruction	ncc167h	ncc68k	Ncc86	Ncc Pent Opt	nccPPC	nccArm7 ARM	nccTMS 32OVC33	ColdFire	see §
(ext2)bf	(ext2)191	IP_EX2_GE_3	/	/	/	/	/	/	/	/	
(ext2)c0	(ext2)192	IP_EX2_GE_3_SGN	/	/	/	/	/	/	/	/	
(ext2)38	(ext2)56	IP_EX2_GE_6	/	/	/	/	/	/	/	/	
(ext2)39	(ext2)57	IP_EX2_GE_6_SGN	/	/	/	/	/	/	/	/	
(ext2)3a	(ext2)58	IP_EX2_GE_REAL48	/	/	/	/	/	/	/	/	
(ext2)bd	(ext2)189	IP_EX2_GT_3	/	/	/	/	/	/	/	/	
(ext2)be	(ext2)190	IP_EX2_GT_3_SGN	/	/	/	/	/	/	/	/	
(ext2)35	(ext2)53	IP_EX2_GT_6	/	/	/	/	/	/	/	/	
(ext2)36	(ext2)54	IP_EX2_GT_6_SGN	/	/	/	/	/	/	/	/	
(ext2)37	(ext2)55	IP_EX2_GT_REAL48	/	/	/	/	/	/	/	/	
(ext2)98	(ext2)152	IP_EX2_LD_BYREF_3	/	/	/	/	/	/	/	/	
(ext2)c	(ext2)12	IP_EX2_LD_BYREF_6	/	/	/	/	/	/	/	/	
(ext2)9a	(ext2)154	IP_EX2_LD_CONST_3	/	/	/	/	/	/	/	/	
(ext2)e	(ext2)14	IP_EX2_LD_CONST_6	/	/	/	/	/	/	/	/	
(ext2)97	(ext2)151	IP_EX2_LD_FAR_3	/	/	/	/	/	/	/	/	
(ext2)b	(ext2)11	IP_EX2_LD_FAR_6	/	/	/	/	/	/	/	/	
(ext2)99	(ext2)153	IP_EX2_LD_IND_3	/	/	/	/	/	/	/	/	
(ext2)d	(ext2)13	IP_EX2_LD_IND_6	/	/	/	/	/	/	/	/	
(ext2)96	(ext2)150	IP_EX2_LD_NEAR_3	/	/	/	/	/	/	/	/	
(ext2)a	(ext2)10	IP_EX2_LD_NEAR_6	/	/	/	/	/	/	/	/	
(ext2)9d	(ext2)157	IP_EX2_LDN_BYREF_3	/	/	/	/	/	/	/	/	
(ext2)11	(ext2)17	IP_EX2_LDN_BYREF_6	/	/	/	/	/	/	/	/	
(ext2)9c	(ext2)156	IP_EX2_LDN_FAR_3	/	/	/	/	/	/	/	/	
(ext2)10	(ext2)16	IP_EX2_LDN_FAR_6	/	/	/	/	/	/	/	/	
(ext2)9e	(ext2)158	IP_EX2_LDN_IND_3	/	/	/	/	/	/	/	/	
(ext2)12	(ext2)18	IP_EX2_LDN_IND_6	/	/	/	/	/	/	/	/	
(ext2)9b	(ext2)155	IP_EX2_LDN_NEAR_3	/	/	/	/	/	/	/	/	
(ext2)f	(ext2)15	IP_EX2_LDN_NEAR_6	/	/	/	/	/	/	/	/	
(ext2)a6	(ext2)166	IP_EX2_LDNS_BYREF_3	/	/	/	/	/	/	/	/	
(ext2)1a	(ext2)26	IP_EX2_LDNS_BYREF_6	/	/	/	/	/	/	/	/	
(ext2)a5	(ext2)165	IP_EX2_LDNS_FAR_3	/	/	/	/	/	/	/	/	
(ext2)19	(ext2)25	IP_EX2_LDNS_FAR_6	/	/	/	/	/	/	/	/	
(ext2)a7	(ext2)167	IP_EX2_LDNS_IND_3	/	/	/	/	/	/	/	/	
(ext2)1b	(ext2)27	IP_EX2_LDNS_IND_6	/	/	/	/	/	/	/	/	
(ext2)a4	(ext2)164	IP_EX2_LDNS_NEAR_3	/	/	/	/	/	/	/	/	
(ext2)18	(ext2)24	IP_EX2_LDNS_NEAR_6	/	/	/	/	/	/	/	/	
(ext2)a1	(ext2)161	IP_EX2_LDS_BYREF_3	/	/	/	/	/	/	/	/	
(ext2)15	(ext2)21	IP_EX2_LDS_BYREF_6	/	/	/	/	/	/	/	/	
(ext2)a3	(ext2)163	IP_EX2_LDS_CONST_3	/	/	/	/	/	/	/	/	
(ext2)17	(ext2)23	IP_EX2_LDS_CONST_6	/	/	/	/	/	/	/	/	
(ext2)a0	(ext2)160	IP_EX2_LDS_FAR_3	/	/	/	/	/	/	/	/	

UCODE (hex)	UCODE (dec)	UCODE instruction	ncc167h	ncc68k	Ncc86	Ncc Pent Opt	nccPPC	nccArm7 ARM	nccTMS 32OVC33	ColdFire	see §
(ext2)14	(ext2)20	IP_EX2_LDS_FAR_6	/	/	/	/	/	/	/	/	
(ext2)a2	(ext2)162	IP_EX2_LDS_IND_3	/	/	/	/	/	/	/	/	
(ext2)16	(ext2)22	IP_EX2_LDS_IND_6	/	/	/	/	/	/	/	/	
(ext2)9f	(ext2)159	IP_EX2_LDS_NEAR_3	/	/	/	/	/	/	/	/	
(ext2)13	(ext2)19	IP_EX2_LDS_NEAR_6	/	/	/	/	/	/	/	/	
(ext2)c1	(ext2)193	IP_EX2_LE_3	/	/	/	/	/	/	/	/	
(ext2)c2	(ext2)194	IP_EX2_LE_3_SGN	/	/	/	/	/	/	/	/	
(ext2)3b	(ext2)59	IP_EX2_LE_6	/	/	/	/	/	/	/	/	
(ext2)3c	(ext2)60	IP_EX2_LE_6_SGN	/	/	/	/	/	/	/	/	
(ext2)3d	(ext2)61	IP_EX2_LE_REAL48	/	/	/	/	/	/	/	/	
(ext2)6a	(ext2)106	IP_EX2_LIMIT_6	/	/	/	/	/	/	/	/	
(ext2)6b	(ext2)107	IP_EX2_LIMIT_6_SGN	/	/	/	/	/	/	/	/	
(ext2)6c	(ext2)108	IP_EX2_LIMIT_REAL48	/	/	/	/	/	/	/	/	
(ext2)c3	(ext2)195	IP_EX2_LT_3	/	/	/	/	/	/	/	/	
(ext2)c4	(ext2)196	IP_EX2_LT_3_SGN	/	/	/	/	/	/	/	/	
(ext2)3e	(ext2)62	IP_EX2_LT_6	/	/	/	/	/	/	/	/	
(ext2)3f	(ext2)63	IP_EX2_LT_6_SGN	/	/	/	/	/	/	/	/	
(ext2)40	(ext2)64	IP_EX2_LT_REAL48	/	/	/	/	/	/	/	/	
(ext2)d6	(ext2)214	IP_EX2_MAX_3	/	/	/	/	/	/	/	/	
(ext2)d7	(ext2)215	IP_EX2_MAX_3_SGN	/	/	/	/	/	/	/	/	
(ext2)67	(ext2)103	IP_EX2_MAX_6	/	/	/	/	/	/	/	/	
(ext2)68	(ext2)104	IP_EX2_MAX_6_SGN	/	/	/	/	/	/	/	/	
(ext2)69	(ext2)105	IP_EX2_MAX_REAL48	/	/	/	/	/	/	/	/	
(ext2)d4	(ext2)212	IP_EX2_MIN_3	/	/	/	/	/	/	/	/	
(ext2)d5	(ext2)213	IP_EX2_MIN_3_SGN	/	/	/	/	/	/	/	/	
(ext2)64	(ext2)100	IP_EX2_MIN_6	/	/	/	/	/	/	/	/	
(ext2)65	(ext2)101	IP_EX2_MIN_6_SGN	/	/	/	/	/	/	/	/	
(ext2)66	(ext2)102	IP_EX2_MIN_REAL48	/	/	/	/	/	/	/	/	
(ext2)ce	(ext2)206	IP_EX2_MOD_3	/	/	/	/	/	/	/	/	
(ext2)cf	(ext2)207	IP_EX2_MOD_3_SGN	/	/	/	/	/	/	/	/	
(ext2)4b	(ext2)75	IP_EX2_MOD_6	/	/	/	/	/	/	/	/	
(ext2)4c	(ext2)76	IP_EX2_MOD_6_SGN	/	/	/	/	/	/	/	/	
(ext2)4d	(ext2)77	IP_EX2_MOD_REAL48	/	/	/	/	/	/	/	/	
(ext2)b9	(ext2)185	IP_EX2_MUL_3	/	/	/	/	/	/	/	/	
(ext2)ba	(ext2)186	IP_EX2_MUL_3_SGN	/	/	/	/	/	/	/	/	
(ext2)2f	(ext2)47	IP_EX2_MUL_6	/	/	/	/	/	/	/	/	
(ext2)30	(ext2)48	IP_EX2_MUL_6_SGN	/	/	/	/	/	/	/	/	
(ext2)31	(ext2)49	IP_EX2_MUL_REAL48	/	/	/	/	/	/	/	/	
(ext2)c6	(ext2)198	IP_EX2_NE_3	/	/	/	/	/	/	/	/	
(ext2)42	(ext2)66	IP_EX2_NE_6	/	/	/	/	/	/	/	/	
(ext2)de	(ext2)222	IP_EX2_NEG_3	/	/	/	/	/	/	/	/	

UCODE (hex)	UCODE (dec)	UCODE instruction	ncc167h	ncc68k	Ncc86	Ncc Pent Opt	nccPPC	nccArm7 ARM	nccTMS 32OVC33	ColdFire	see §
(ext2)7b	(ext2)123	IP_EX2_NEG_6	/	/	/	/	/	/	/	/	
(ext2)7a	(ext2)122	IP_EX2_NEG_REAL48	/	/	/	/	/	/	/	/	
(ext2)0	(ext2)0	IP_EX2_NOP	/	/	/	/	/	/	/	/	
(ext2)c7	(ext2)199	IP_EX2_NOT_3	/	/	/	/	/	/	/	/	
(ext2)43	(ext2)67	IP_EX2_NOT_6	/	/	/	/	/	/	/	/	
(ext2)b3	(ext2)179	IP_EX2_OR_3	/	/	/	/	/	/	/	/	
(ext2)27	(ext2)39	IP_EX2_OR_6	/	/	/	/	/	/	/	/	
(ext2)b1	(ext2)177	IP_EX2_POP_AE_3	/	/	/	/	/	/	/	/	
(ext2)25	(ext2)37	IP_EX2_POP_AE_6	/	/	/	/	/	/	/	/	
(ext2)b0	(ext2)176	IP_EX2_PUSH_AE_3	/	/	/	/	/	/	/	/	
(ext2)24	(ext2)36	IP_EX2_PUSH_AE_6	/	/	/	/	/	/	/	/	
(ext2)ca	(ext2)202	IP_EX2_ROL_3	/	/	/	/	/	/	/	/	
(ext2)46	(ext2)70	IP_EX2_ROL_6	/	/	/	/	/	/	/	/	
(ext2)cb	(ext2)203	IP_EX2_ROR_3	/	/	/	/	/	/	/	/	
(ext2)47	(ext2)71	IP_EX2_ROR_6	/	/	/	/	/	/	/	/	
(ext2)6d	(ext2)109	IP_EX2_SEL_6	/	/	/	/	/	/	/	/	
(ext2)6e	(ext2)110	IP_EX2_SEL_6_SGN	/	/	/	/	/	/	/	/	
(ext2)6f	(ext2)111	IP_EX2_SEL_REAL48	/	/	/	/	/	/	/	/	
(ext2)c8	(ext2)200	IP_EX2_SHL_3	/	/	/	/	/	/	/	/	
(ext2)44	(ext2)68	IP_EX2_SHL_6	/	/	/	/	/	/	/	/	
(ext2)c9	(ext2)201	IP_EX2_SHR_3	/	/	/	/	/	/	/	/	
(ext2)45	(ext2)69	IP_EX2_SHR_6	/	/	/	/	/	/	/	/	
(ext2)aa	(ext2)170	IP_EX2_ST_BYREF_3	/	/	/	/	/	/	/	/	
(ext2)1e	(ext2)30	IP_EX2_ST_BYREF_6	/	/	/	/	/	/	/	/	
(ext2)a9	(ext2)169	IP_EX2_ST_FAR_3	/	/	/	/	/	/	/	/	
(ext2)1d	(ext2)29	IP_EX2_ST_FAR_6	/	/	/	/	/	/	/	/	
(ext2)ab	(ext2)171	IP_EX2_ST_IND_3	/	/	/	/	/	/	/	/	
(ext2)1f	(ext2)31	IP_EX2_ST_IND_6	/	/	/	/	/	/	/	/	
(ext2)a8	(ext2)168	IP_EX2_ST_NEAR_3	/	/	/	/	/	/	/	/	
(ext2)1c	(ext2)28	IP_EX2_ST_NEAR_6	/	/	/	/	/	/	/	/	
(ext2)ae	(ext2)174	IP_EX2_STN_BYREF_3	/	/	/	/	/	/	/	/	
(ext2)22	(ext2)34	IP_EX2_STN_BYREF_6	/	/	/	/	/	/	/	/	
(ext2)ad	(ext2)173	IP_EX2_STN_FAR_3	/	/	/	/	/	/	/	/	
(ext2)21	(ext2)33	IP_EX2_STN_FAR_6	/	/	/	/	/	/	/	/	
(ext2)af	(ext2)175	IP_EX2_STN_IND_3	/	/	/	/	/	/	/	/	
(ext2)23	(ext2)35	IP_EX2_STN_IND_6	/	/	/	/	/	/	/	/	
(ext2)ac	(ext2)172	IP_EX2_STN_NEAR_3	/	/	/	/	/	/	/	/	
(ext2)20	(ext2)32	IP_EX2_STN_NEAR_6	/	/	/	/	/	/	/	/	
(ext2)b7	(ext2)183	IP_EX2_SUB_3	/	/	/	/	/	/	/	/	
(ext2)b8	(ext2)184	IP_EX2_SUB_3_SGN	/	/	/	/	/	/	/	/	
(ext2)2c	(ext2)44	IP_EX2_SUB_6	/	/	/	/	/	/	/	/	

UCODE (hex)	UCODE (dec)	UCODE instruction	ncc167h	ncc68k	Ncc86	Ncc Pent Opt	nccPPC	nccArm7 ARM	nccTMS 32OVC33	ColdFire	see §
(ext2)2d	(ext2)45	IP_EX2_SUB_6_SGN	/	/	/	/	/	/	/	/	
(ext2)2e	(ext2)46	IP_EX2_SUB_REAL48	/	/	/	/	/	/	/	/	
(ext2)b4	(ext2)180	IP_EX2_XOR_3	/	/	/	/	/	/	/	/	
(ext2)28	(ext2)40	IP_EX2_XOR_6	/	/	/	/	/	/	/	/	
a3	163	IP_OP_ADD_1	X	X	X	X	X	X	X	X	5.6.7.3.29
a4	164	IP_OP_ADD_1_SGN	X	X	X	X	X	X	X	X	5.6.7.3.29
a5	165	IP_OP_ADD_2	X	X	X	X	X	X	X	X	5.6.7.3.29
a6	166	IP_OP_ADD_2_SGN	X	X	X	X	X	X	X	X	5.6.7.3.29
a7	167	IP_OP_ADD_4	X	X	X	X	X	X	X	X	5.6.7.3.29
a8	168	IP_OP_ADD_4_SGN	X	X	X	X	X	X	X	X	5.6.7.3.29
a9	169	IP_OP_ADD_FLOAT	X	X	O	X	X	X	X	X	5.6.7.3.29
63	99	IP_OP_ADD_OFFSET	X	X	X	X	X	X	X	X	5.6.7.3.21
98	152	IP_OP_AND_1	X	X	X	X	X	X	X	X	5.6.7.3.29
99	153	IP_OP_AND_2	X	X	X	X	X	X	X	X	5.6.7.3.29
9a	154	IP_OP_AND_4	X	X	X	X	X	X	X	X	5.6.7.3.29
97	151	IP_OP_AND_BIT	X	X	X	X	X	X	X	X	5.6.7.3.29
fe	254	IP_OP_BREAK	/	/	/	/	/	/	/	/	
04	4	IP_OP_CAL	X	X	X	X	X	X	X	X	5.6.7.3.3
05	5	IP_OP_CALC	X	/	/	/	X	X	X	X	5.6.7.3.3
f9	249	IP_OP_CALCEN	/	/	/	/	/	/	/	/	
06	6	IP_OP_CALCN	X	/	/	/	X	X	X	/	5.6.7.3.3
f2	242	IP_OP_CREATE_BITREF_BYREF	/	X	X	X	X	X	X	X	
f1	241	IP_OP_CREATE_BITREF_FAR	/	X	/	/	/	X	/	/	
f3	243	IP_OP_CREATE_BITREF_IND	/	/	X	X	/	X	X	X	
f0	240	IP_OP_CREATE_BITREF_NEAR	/	X	X	X	/	X	X	X	
60	96	IP_OP_CREATE_REFERENCE	X	X	X	X	X	X	X	X	5.6.7.3.18
f8	248	IP_OP_CREATE_REFERENCE_FAR	X	X	X	X	X	X	X	X	5.6.7.3.19
b8	184	IP_OP_DIV_1	X	X	X	X	X	X	O	X	5.6.7.3.29
b9	185	IP_OP_DIV_1_SGN	X	X	X	X	X	X	X	X	5.6.7.3.29
ba	186	IP_OP_DIV_2	X	X	X	X	X	X	O	X	5.6.7.3.29
bb	187	IP_OP_DIV_2_SGN	X	X	X	X	X	X	X	X	5.6.7.3.29
bc	188	IP_OP_DIV_4	X	X	X	X	X	X	O	X	5.6.7.3.29
bd	189	IP_OP_DIV_4_SGN	X	X	X	X	X	X	X	X	5.6.7.3.29
be	190	IP_OP_DIV_FLOAT	X	X	O	X	X	X	O	X	
fd	253	IP_OP_END_OF_IL	/	/	/	/	/	/	/	/	
e0	224	IP_OP_EQ_1	X	X	X	X	X	X	X	X	5.6.7.3.29
e1	225	IP_OP_EQ_2	X	X	X	X	X	X	X	X	5.6.7.3.29
e2	226	IP_OP_EQ_4	X	X	X	X	X	X	X	X	5.6.7.3.29
e8	232	IP_OP_EQ_ANY	/	/	O	O	X	X	O	X	5.6.7.3.29
df	223	IP_OP_EQ_BIT	X	X	X	X	X	X	X	X	5.6.7.3.29
fc	252	IP_OP_EXT1	X	X	X	X	X	X	X	X	

UCODE (hex)	UCODE (dec)	UCODE instruction	ncc167h	ncc68k	Ncc86	Ncc Pent Opt	nccPPC	nccArm7 ARM	nccTMS 32OVC33	ColdFire	see §
c8	200	IP_OP_GE_1	X	X	X	X	X	X	X	X	5.6.7.3.29
c9	201	IP_OP_GE_1_SGN	X	X	X	X	X	X	X	X	5.6.7.3.29
ca	202	IP_OP_GE_2	X	X	X	X	X	X	X	X	5.6.7.3.29
cb	203	IP_OP_GE_2_SGN	X	X	X	X	X	X	X	X	5.6.7.3.29
cc	204	IP_OP_GE_4	X	X	X	X	X	X	X	X	5.6.7.3.29
cd	205	IP_OP_GE_4_SGN	X	X	X	X	X	X	X	X	5.6.7.3.29
ec	236	IP_OP_GE_ANY	/	/	O	O	X	X	O	X	5.6.7.3.29
c7	199	IP_OP_GE_BIT	X	X	X	X	X	X	X	X	5.6.7.3.29
ce	206	IP_OP_GE_FLOAT	X	X	O	X	X	X	X	X	5.6.7.3.29
c0	192	IP_OP_GT_1	X	X	X	X	X	X	X	X	5.6.7.3.29
c1	193	IP_OP_GT_1_SGN	X	X	X	X	X	X	X	X	5.6.7.3.29
c2	194	IP_OP_GT_2	X	X	X	X	X	X	X	X	5.6.7.3.29
c3	195	IP_OP_GT_2_SGN	X	X	X	X	X	X	X	X	5.6.7.3.29
c4	196	IP_OP_GT_4	X	X	X	X	X	X	X	X	5.6.7.3.29
c5	197	IP_OP_GT_4_SGN	X	X	X	X	X	X	X	X	5.6.7.3.29
ea	234	IP_OP_GT_ANY	/	/	O	O	X	X	O	X	5.6.7.3.29
bf	191	IP_OP_GT_BIT	X	X	X	X	X	X	X	X	5.6.7.3.29
c6	198	IP_OP_GT_FLOAT	X	X	O	X	X	X	X	X	5.6.7.3.29
01	1	IP_OP_JMP	X	X	X	X	X	X	X	X	5.6.7.3.2
02	2	IP_OP_JMPC	X	X	X	X	X	X	X	X	5.6.7.3.2
03	3	IP_OP_JMPCN	X	X	X	X	X	X	X	X	5.6.7.3.2
15	21	IP_OP_LD_BYREF_1	X	X	X	X	X	X	X	X	5.6.7.3.7
16	22	IP_OP_LD_BYREF_2	X	X	X	X	X	X	X	X	5.6.7.3.7
17	23	IP_OP_LD_BYREF_4	X	X	X	X	X	X	X	X	5.6.7.3.7
18	24	IP_OP_LD_BYREF_ANY	X	/	X	X	X	X	X	X	5.6.7.3.7
14	20	IP_OP_LD_BYREF_BIT	X	X	X	X	X	X	X	X	5.6.7.3.7
20	32	IP_OP_LD_CONST_1	X	X	X	X	X	X	X	X	5.6.7.3.10
21	33	IP_OP_LD_CONST_2	X	X	X	X	X	X	X	X	5.6.7.3.10
22	34	IP_OP_LD_CONST_4	X	X	X	X	X	X	X	X	5.6.7.3.10
23	35	IP_OP_LD_CONST_ANY	/	/	X	X	X	X	X	X	5.6.7.3.10
1f	31	IP_OP_LD_CONST_FALSE	X	X	X	X	X	X	X	X	5.6.7.3.9
1e	30	IP_OP_LD_CONST_TRUE	X	X	X	X	X	X	X	X	5.6.7.3.9
10	16	IP_OP_LD_FAR_1	X	X	X	X	X	X	X	X	5.6.7.3.6
11	17	IP_OP_LD_FAR_2	X	X	X	X	X	X	X	X	5.6.7.3.6
12	18	IP_OP_LD_FAR_4	X	X	X	X	X	X	X	X	5.6.7.3.6
13	19	IP_OP_LD_FAR_ANY	X	/	X	X	X	X	X	X	5.6.7.3.6
0f	15	IP_OP_LD_FAR_BIT	X	X	X	X	X	X	X	X	5.6.7.3.6
1a	26	IP_OP_LD_IND_1	X	X	X	X	X	X	X	X	5.6.7.3.8
1b	27	IP_OP_LD_IND_2	X	X	X	X	X	X	X	X	5.6.7.3.8
1c	28	IP_OP_LD_IND_4	X	X	X	X	X	X	X	X	5.6.7.3.8
1d	29	IP_OP_LD_IND_ANY	X	/	X	X	X	X	X	X	5.6.7.3.8

UCODE (hex)	UCODE (dec)	UCODE instruction	ncc167h	ncc68k	Ncc86	Ncc Pent Opt	nccPPC	nccArm7 ARM	nccTMS 32OVC33	ColdFire	see §
19	25	IP_OP_LD_IND_BIT	X	X	X	X	X	X	X	X	5.6.7.3.8
0b	11	IP_OP_LD_NEAR_1	X	X	X	X	X	X	X	X	5.6.7.3.5
0c	12	IP_OP_LD_NEAR_2	X	X	X	X	X	X	X	X	5.6.7.3.5
0d	13	IP_OP_LD_NEAR_4	X	X	X	X	X	X	X	X	5.6.7.3.5
0e	14	IP_OP_LD_NEAR_ANY	X	/	X	X	X	X	X	X	5.6.7.3.5
0a	10	IP_OP_LD_NEAR_BIT	X	X	X	X	X	X	X	X	5.6.7.3.5
61	97	IP_OP_LD_REFERENCE	X	X	X	X	X	X	X	X	5.6.7.3.20
2d	45	IP_OP_LDN_BYREF_1	X	X	X	X	X	X	X	X	5.6.7.3.7
2e	46	IP_OP_LDN_BYREF_2	X	X	X	X	X	X	X	X	5.6.7.3.7
2f	47	IP_OP_LDN_BYREF_4	X	X	X	X	X	X	X	X	5.6.7.3.7
2c	44	IP_OP_LDN_BYREF_BIT	X	X	X	X	X	X	X	X	5.6.7.3.7
29	41	IP_OP_LDN_FAR_1	X	X	X	X	X	X	X	X	5.6.7.3.6
2a	42	IP_OP_LDN_FAR_2	X	X	X	X	X	X	X	X	5.6.7.3.6
2b	43	IP_OP_LDN_FAR_4	X	X	X	X	X	X	X	X	5.6.7.3.6
28	40	IP_OP_LDN_FAR_BIT	X	X	X	X	X	X	X	X	5.6.7.3.6
31	49	IP_OP_LDN_IND_1	X	X	X	X	X	X	X	X	5.6.7.3.8
32	50	IP_OP_LDN_IND_2	X	X	X	X	X	X	X	X	5.6.7.3.8
33	51	IP_OP_LDN_IND_4	X	X	X	X	X	X	X	X	5.6.7.3.8
30	48	IP_OP_LDN_IND_BIT	X	X	X	X	X	X	X	X	5.6.7.3.8
25	37	IP_OP_LDN_NEAR_1	X	X	X	X	X	X	X	X	5.6.7.3.5
26	38	IP_OP_LDN_NEAR_2	X	X	X	X	X	X	X	X	5.6.7.3.5
27	39	IP_OP_LDN_NEAR_4	X	X	X	X	X	X	X	X	5.6.7.3.5
24	36	IP_OP_LDN_NEAR_BIT	X	X	X	X	X	X	X	X	5.6.7.3.5
89	137	IP_OP_LDNS_BYREF_1	X	X	X	X	X	X	X	X	5.6.7.3.7
8a	138	IP_OP_LDNS_BYREF_2	X	X	X	X	X	X	X	X	5.6.7.3.7
8b	139	IP_OP_LDNS_BYREF_4	X	X	X	X	X	X	X	X	5.6.7.3.7
88	136	IP_OP_LDNS_BYREF_BIT	X	X	X	X	X	X	X	X	5.6.7.3.7
85	133	IP_OP_LDNS_FAR_1	X	X	X	X	X	X	X	X	5.6.7.3.6
86	134	IP_OP_LDNS_FAR_2	X	X	X	X	X	X	X	X	5.6.7.3.6
87	135	IP_OP_LDNS_FAR_4	X	X	X	X	X	X	X	X	5.6.7.3.6
84	132	IP_OP_LDNS_FAR_BIT	X	X	X	X	X	X	X	X	5.6.7.3.6
8d	141	IP_OP_LDNS_IND_1	X	X	X	X	X	X	X	X	5.6.7.3.8
8e	142	IP_OP_LDNS_IND_2	X	X	X	X	X	X	X	X	5.6.7.3.8
8f	143	IP_OP_LDNS_IND_4	X	X	X	X	X	X	X	X	5.6.7.3.8
8c	140	IP_OP_LDNS_IND_BIT	X	X	X	X	X	X	X	X	5.6.7.3.8
81	129	IP_OP_LDNS_NEAR_1	X	X	X	X	X	X	X	X	5.6.7.3.5
82	130	IP_OP_LDNS_NEAR_2	X	X	X	X	X	X	X	X	5.6.7.3.5
83	131	IP_OP_LDNS_NEAR_4	X	X	X	X	X	X	X	X	5.6.7.3.5
80	128	IP_OP_LDNS_NEAR_BIT	X	X	X	X	X	X	X	X	5.6.7.3.5
71	113	IP_OP_LDS_BYREF_1	X	X	X	X	X	X	X	X	5.6.7.3.7
72	114	IP_OP_LDS_BYREF_2	X	X	X	X	X	X	X	X	5.6.7.3.7

UCODE (hex)	UCODE (dec)	UCODE instruction	ncc167h	ncc68k	Ncc86	Ncc Pent Opt	nccPPC	nccArm7 ARM	nccTMS 32OVC33	ColdFire	see §
73	115	IP_OP_LDS_BYREF_4	X	X	X	X	X	X	X	X	5.6.7.3.7
74	116	IP_OP_LDS_BYREF_ANY	X	/	X	X	X	X	X	X	5.6.7.3.7
70	112	IP_OP_LDS_BYREF_BIT	X	X	X	X	X	X	X	X	5.6.7.3.7
7c	124	IP_OP_LDS_CONST_1	X	X	X	X	X	X	X	X	5.6.7.3.10
7d	125	IP_OP_LDS_CONST_2	X	X	X	X	X	X	X	X	5.6.7.3.10
7e	126	IP_OP_LDS_CONST_4	X	X	X	X	X	X	X	X	5.6.7.3.10
7f	127	IP_OP_LDS_CONST_ANY	/	/	X	X	X	X	X	X	5.6.7.3.10
7b	123	IP_OP_LDS_CONST_FALSE	X	X	X	X	X	X	X	X	5.6.7.3.9
7a	122	IP_OP_LDS_CONST_TRUE	X	X	X	X	X	X	X	X	5.6.7.3.9
6c	108	IP_OP_LDS_FAR_1	X	X	X	X	X	X	X	X	5.6.7.3.6
6d	109	IP_OP_LDS_FAR_2	X	X	X	X	X	X	X	X	5.6.7.3.6
6e	110	IP_OP_LDS_FAR_4	X	X	X	X	X	X	X	X	5.6.7.3.6
6f	111	IP_OP_LDS_FAR_ANY	X	/	X	X	X	X	X	X	5.6.7.3.6
6b	107	IP_OP_LDS_FAR_BIT	X	X	X	X	X	X	X	X	5.6.7.3.6
76	118	IP_OP_LDS_IND_1	X	X	X	X	X	X	X	X	5.6.7.3.8
77	119	IP_OP_LDS_IND_2	X	X	X	X	X	X	X	X	5.6.7.3.8
78	120	IP_OP_LDS_IND_4	X	X	X	X	X	X	X	X	5.6.7.3.8
79	121	IP_OP_LDS_IND_ANY	X	/	X	X	X	X	X	X	5.6.7.3.8
75	117	IP_OP_LDS_IND_BIT	X	X	X	X	X	X	X	X	5.6.7.3.8
67	103	IP_OP_LDS_NEAR_1	X	X	X	X	X	X	X	X	5.6.7.3.5
68	104	IP_OP_LDS_NEAR_2	X	X	X	X	X	X	X	X	5.6.7.3.5
69	105	IP_OP_LDS_NEAR_4	X	X	X	X	X	X	X	X	5.6.7.3.5
6a	106	IP_OP_LDS_NEAR_ANY	X	/	X	X	X	X	X	X	5.6.7.3.5
66	102	IP_OP_LDS_NEAR_BIT	X	X	X	X	X	X	X	X	5.6.7.3.5
d0	208	IP_OP_LE_1	X	X	X	X	X	X	X	X	5.6.7.3.29
d1	209	IP_OP_LE_1_SGN	X	X	X	X	X	X	X	X	5.6.7.3.29
d2	210	IP_OP_LE_2	X	X	X	X	X	X	X	X	5.6.7.3.29
d3	211	IP_OP_LE_2_SGN	X	X	X	X	X	X	X	X	5.6.7.3.29
d4	212	IP_OP_LE_4	X	X	X	X	X	X	X	X	5.6.7.3.29
d5	213	IP_OP_LE_4_SGN	X	X	X	X	X	X	X	X	5.6.7.3.29
ee	238	IP_OP_LE_ANY	/	/	O	O	X	X	O	X	5.6.7.3.29
cf	207	IP_OP_LE_BIT	X	X	X	X	X	X	X	X	5.6.7.3.29
d6	214	IP_OP_LE_FLOAT	X	X	O	X	X	X	X	X	5.6.7.3.29
d8	216	IP_OP_LT_1	X	X	X	X	X	X	X	X	5.6.7.3.29
d9	217	IP_OP_LT_1_SGN	X	X	X	X	X	X	X	X	5.6.7.3.29
da	218	IP_OP_LT_2	X	X	X	X	X	X	X	X	5.6.7.3.29
db	219	IP_OP_LT_2_SGN	X	X	X	X	X	X	X	X	5.6.7.3.29
dd	221	IP_OP_LT_4_SGN	X	X	X	X	X	X	X	X	5.6.7.3.29
ed	237	IP_OP_LT_ANY	/	/	O	O	X	X	O	X	5.6.7.3.29
d7	215	IP_OP_LT_BIT	X	X	X	X	X	X	X	X	5.6.7.3.29
de	222	IP_OP_LT_FLOAT	X	X	O	X	X	X	X	X	5.6.7.3.29

UCODE (hex)	UCODE (dec)	UCODE instruction	ncc167h	ncc68k	Ncc86	Ncc Pent Opt	nccPPC	nccArm7 ARM	nccTMS 32OVC33	ColdFire	see §
b1	177	IP_OP_MUL_1	X	X	X	X	X	X	X	X	5.6.7.3.29
b2	178	IP_OP_MUL_1_SGN	X	X	X	X	X	X	X	X	5.6.7.3.29
b3	179	IP_OP_MUL_2	X	X	X	X	X	X	X	X	5.6.7.3.29
b4	180	IP_OP_MUL_2_SGN	X	X	X	X	X	X	X	X	5.6.7.3.29
b5	181	IP_OP_MUL_4	X	X	X	X	X	X	X	X	5.6.7.3.29
b6	182	IP_OP_MUL_4_SGN	X	X	X	X	X	X	X	X	5.6.7.3.29
b7	183	IP_OP_MUL_FLOAT	X	X	O	X	X	X	X	X	5.6.7.3.29
e4	228	IP_OP_NE_1	X	X	X	X	X	X	X	X	5.6.7.3.29
e5	229	IP_OP_NE_2	X	X	X	X	X	X	X	X	5.6.7.3.29
e6	230	IP_OP_NE_4	X	X	X	X	X	X	X	X	5.6.7.3.29
e9	233	IP_OP_NE_ANY	/	/	O	O	X	X	O	X	5.6.7.3.29
e3	227	IP_OP_NE_BIT	X	X	X	X	X	X	X	X	5.6.7.3.29
00	0	IP_OP_NOP	X	X	X	X	X	X	X	X	5.6.7.3.1
9c	156	IP_OP_OR_1	X	X	X	X	X	X	X	X	5.6.7.3.29
9d	157	IP_OP_OR_2	X	X	X	X	X	X	X	X	5.6.7.3.29
9e	158	IP_OP_OR_4	X	X	X	X	X	X	X	X	5.6.7.3.29
9b	155	IP_OP_OR_BIT	X	X	X	X	X	X	X	X	5.6.7.3.29
91	145	IP_OP_POP_AE	X	X	X	X	X	X	X	X	5.6.7.3.23
95	149	IP_OP_POP_AE_ANY	X	/	X	X	X	X	X	X	5.6.7.3.25
93	147	IP_OP_POP_REF	X	X	X	X	X	X	X	X	5.6.7.3.27
90	144	IP_OP_PUSH_AE	X	X	X	X	X	X	X	X	5.6.7.3.22
94	148	IP_OP_PUSH_AE_ANY	X	/	X	X	X	X	X	X	5.6.7.3.24
92	146	IP_OP_PUSH_REF	X	X	X	X	X	X	X	X	5.6.7.3.26
5e	94	IP_OP_R_BYREF_BIT	X	X	X	X	X	X	X	X	5.6.7.3.17
5d	93	IP_OP_R_FAR_BIT	X	X	X	X	X	X	X	X	5.6.7.3.16
5f	95	IP_OP_R_IND_BIT	X	X	X	X	X	X	X	X	
5c	92	IP_OP_R_NEAR_BIT	X	X	X	X	X	X	X	X	5.6.7.3.15
96	150	IP_OP_RESTORE_DATA	X	X	X	X	X	X	X	X	5.6.7.3.28
07	7	IP_OP_RET	X	X	X	X	X	X	X	X	5.6.7.3.4
08	8	IP_OP_RETC	X	X	X	X	X	X	X	X	5.6.7.3.4
09	9	IP_OP_RETCN	X	X	X	X	X	X	X	X	5.6.7.3.4
5a	90	IP_OP_S_BYREF_BIT	X	X	X	X	X	X	X	X	5.6.7.3.17
59	89	IP_OP_S_FAR_BIT	X	X	X	X	X	X	X	X	5.6.7.3.16
5b	91	IP_OP_S_IND_BIT	X	X	X	X	X	X	X	X	
58	88	IP_OP_S_NEAR_BIT	X	X	X	X	X	X	X	X	5.6.7.3.15
eb	235	IP_OP_SELCT_ARRAY_IND	X	X	X	X	X	X	X	X	5.6.7.3.31
3f	63	IP_OP_ST_BYREF_1	X	X	X	X	X	X	X	X	5.6.7.3.13
40	64	IP_OP_ST_BYREF_2	X	X	X	X	X	X	X	X	5.6.7.3.13
41	65	IP_OP_ST_BYREF_4	X	X	X	X	X	X	X	X	5.6.7.3.13
42	66	IP_OP_ST_BYREF_ANY	X	/	X	X	X	X	X	X	5.6.7.3.13
3e	62	IP_OP_ST_BYREF_BIT	X	X	X	X	X	X	X	X	5.6.7.3.13

UCODE (hex)	UCODE (dec)	UCODE instruction	ncc167h	ncc68k	Ncc86	Ncc Pent Opt	nccPPC	nccArm7 ARM	nccTMS 32OVC33	ColdFire	see §
f6	246	IP_OP_ST_BYREF_STRING	X	/	X	X	X	X	X	X	5.6.7.3.13
3a	58	IP_OP_ST_FAR_1	X	X	X	X	X	X	X	X	5.6.7.3.12
3b	59	IP_OP_ST_FAR_2	X	X	X	X	X	X	X	X	5.6.7.3.12
3c	60	IP_OP_ST_FAR_4	X	X	X	X	X	X	X	X	5.6.7.3.12
3d	61	IP_OP_ST_FAR_ANY	X	/	X	X	X	X	X	X	5.6.7.3.12
39	57	IP_OP_ST_FAR_BIT	X	X	X	X	X	X	X	X	5.6.7.3.12
f5	245	IP_OP_ST_FAR_STRING	X	/	X	X	X	X	X	X	5.6.7.3.12
44	68	IP_OP_ST_IND_1	X	X	X	X	X	X	X	X	5.6.7.3.14
45	69	IP_OP_ST_IND_2	X	X	X	X	X	X	X	X	5.6.7.3.14
46	70	IP_OP_ST_IND_4	X	X	X	X	X	X	X	X	5.6.7.3.14
47	71	IP_OP_ST_IND_ANY	X	/	X	X	X	X	X	X	5.6.7.3.14
43	67	IP_OP_ST_IND_BIT	X	X	X	X	X	X	X	X	5.6.7.3.14
f7	247	IP_OP_ST_IND_STRING	X	/	X	X	X	X	X	X	5.6.7.3.14
35	53	IP_OP_ST_NEAR_1	X	X	X	X	X	X	X	X	5.6.7.3.11
36	54	IP_OP_ST_NEAR_2	X	X	X	X	X	X	X	X	5.6.7.3.11
37	55	IP_OP_ST_NEAR_4	X	X	X	X	X	X	X	X	5.6.7.3.11
38	56	IP_OP_ST_NEAR_ANY	X	/	X	X	X	X	X	X	5.6.7.3.11
34	52	IP_OP_ST_NEAR_BIT	X	X	X	X	X	X	X	X	5.6.7.3.11
f4	244	IP_OP_ST_NEAR_STRING	X	/	X	X	X	X	X	X	5.6.7.3.11
62	98	IP_OP_ST_REFERENCE	X	X	X	X	X	X	X	X	
51	81	IP_OP_STN_BYREF_1	X	X	X	X	X	X	X	X	5.6.7.3.13
52	82	IP_OP_STN_BYREF_2	X	X	X	X	X	X	X	X	5.6.7.3.13
53	83	IP_OP_STN_BYREF_4	X	X	X	X	X	X	X	X	5.6.7.3.13
50	80	IP_OP_STN_BYREF_BIT	X	X	X	X	X	X	X	X	5.6.7.3.13
4d	77	IP_OP_STN_FAR_1	X	X	X	X	X	X	X	X	5.6.7.3.12
4e	78	IP_OP_STN_FAR_2	X	X	X	X	X	X	X	X	5.6.7.3.12
4f	79	IP_OP_STN_FAR_4	X	X	X	X	X	X	X	X	5.6.7.3.12
4c	76	IP_OP_STN_FAR_BIT	X	X	X	X	X	X	X	X	5.6.7.3.12
55	85	IP_OP_STN_IND_1	X	X	X	X	X	X	X	X	5.6.7.3.14
56	86	IP_OP_STN_IND_2	X	X	X	X	X	X	X	X	5.6.7.3.14
57	87	IP_OP_STN_IND_4	X	X	X	X	X	X	X	X	5.6.7.3.14
54	84	IP_OP_STN_IND_BIT	X	X	X	X	X	X	X	X	5.6.7.3.14
49	73	IP_OP_STN_NEAR_1	X	X	X	X	X	X	X	X	5.6.7.3.11
4a	74	IP_OP_STN_NEAR_2	X	X	X	X	X	X	X	X	5.6.7.3.11
4b	75	IP_OP_STN_NEAR_4	X	X	X	X	X	X	X	X	5.6.7.3.11
48	72	IP_OP_STN_NEAR_BIT	X	X	X	X	X	X	X	X	5.6.7.3.11
aa	170	IP_OP_SUB_1	X	X	X	X	X	X	X	X	5.6.7.3.29
ab	171	IP_OP_SUB_1_SGN	X	X	X	X	X	X	X	X	5.6.7.3.29
ac	172	IP_OP_SUB_2	X	X	X	X	X	X	X	X	5.6.7.3.29
ad	173	IP_OP_SUB_2_SGN	X	X	X	X	X	X	X	X	5.6.7.3.29
ae	174	IP_OP_SUB_4	X	X	X	X	X	X	X	X	5.6.7.3.29

UCODE (hex)	UCODE (dec)	UCODE instruction	ncc167h	ncc68k	Ncc86	Ncc Pent Opt	nccPPC	nccArm7 ARM	nccTMS 32OVC33	ColdFire	see §
af	175	IP_OP_SUB_4_SGN	X	X	X	X	X	X	X	X	5.6.7.3.29
b0	176	IP_OP_SUB_FLOAT	X	X	O	X	X	X	X	X	5.6.7.3.29
ef	239	IP_OP_TRUNC	/	X	O	O	X	/	X	X	5.6.7.3.30
e7	231	IP_OP_TYPECAST	P	P	O	O	X	P	O	O	5.6.7.3.30
a0	160	IP_OP_XOR_1	X	X	X	X	X	X	X	X	5.6.7.3.29
a1	161	IP_OP_XOR_2	X	X	X	X	X	X	X	X	5.6.7.3.29
a2	162	IP_OP_XOR_4	X	X	X	X	X	X	X	X	5.6.7.3.29
9f	159	IP_OP_XOR_BIT	X	X	X	X	X	X	X	X	5.6.7.3.29

5.6.7.3 中间代码指令类描述

5.6.7.3.1 NOP(IP_OP_NOP)

Opcode
IP_OP_NOP

NOP指令不带操作数。

5.6.7.3.2 跳转指令（IP_OP_JMPxx）

Opcode	Operand	Operand
IP_OP_JMPxx	absolute jump target (offset16)	

这个WORD操作数指定跳转目的地从段头（包括段头的大小）的绝对偏移量。

```
PROGRAM TEST_01
VAR
    x : int;
    y : int;
END_VAR
ld x
st y
jmp ende
ld y
st x
ende:
END_PROGRAM
```

Data segment

```
0000 : 0A 00 01 06 02 00 00 00
0008 : 00 00
```

Code:

```
6: LD_NEAR_2 00 06
9: ST_NEAR_2 00 08
c: JMP 00 15
f: LD_NEAR_2 00 08
12: ST_NEAR_2 00 06
15: RET
```

5.6.7.3.3 调用指令（IP_OP_CALxx）

OpCode	Operand	Operand
IP_OP_CALxx	Ptr16 to Instance Handle:Extension	

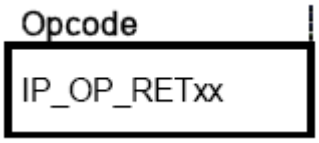
这个WORD操作数是一个指向调用的实例句柄的16位指针，并存储在“当前”数据段中。这个句柄是一WORD类型数据，然后是扩展域：

Handle_CallInstanz = *(WORD *) Operand+0;

Extension_Field = *(WORD *) Operand+2;

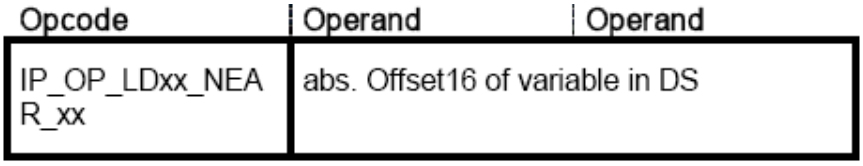
如果扩展域包含NULL值，那么调用的POE将被解释器处理。此外，扩展域还表示指定调用的固件块的数量。

5.6.7.3.4 RET指令（IP_OP_RETxx）



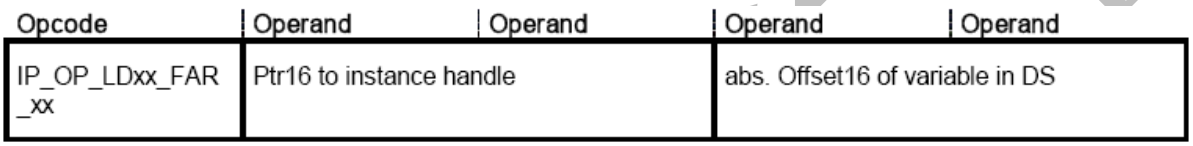
返回指令不带操作数。解释程序自动地区别从“嵌套”POE的返回和退出“主程序”。当退出主程序时，指令指针将被重新设置为指向周期的开始。随着下一个解释程序的调用，一个新的周期将被执行。

5.6.7.3.5 NEAR装载指令（IP_OP_LDxx_NEAR_xx）



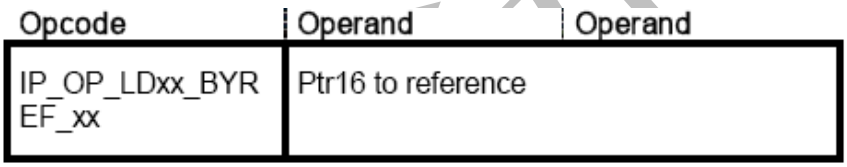
这个WORD操作数指定了变量在“当前”数据段中从段头（包括段头的长度）开始的绝对偏移量。

5.6.7.3.6 FAR装载指令（IP_OP_LDxx_FAR_xx）



FAR装载指令带有两个WORD操作数。第一个操作数是一个16位的指向将要被寻址的实例句柄的指针。这个句柄被存储在“当前”数据段中。第二个WORD操作数指定了变量在另一个不同的实例的数据段中从段头（包括段头的长度）开始的绝对偏移量。

5.6.7.3.7 BYREF装载指令（IP_OP_LDxx_BYREF_xx）



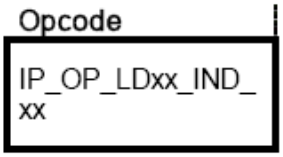
这个WORD操作数是一个16位的指向期望的变量的引用的指针。这个指针被存储在“当前”数据段中（引用的从段头开始的绝对偏移量包含段头的长度）。这是这个引用的结构：

Ptr16_To_Reference + 0: WORD Instance handle

Ptr16_To_Reference + 2: WORD VarOffset16

这个存储在引用中的实例句柄指定了将要被寻址的数据段。在寻址实例的数据段中，VarOffset16指定了变量的从段头开始的绝对偏移量（包括段头长度）。

5.6.7.3.8 Ind装载指令（IP_OP_LDxx_IND_xx）



Ind指令与Byref指令很相似。但是，对于ind指令，这个引用不通过指针来存取。作为替代，引用可以通过引用累加器存取，并且可由Load_Reference或者Create_Reference存取。这个累加器动态地存取引用。

5.6.7.3.9 Bit常量的装载指令（IP_OP_LDxx_CONST_TRUE/IP_OP_LDxx_CONST_FALSE）

Opcode
IP_OP_LDxx_CON ST_TRUE

Opcode
IP_OP_LDxx_CON ST_FALSE

装载Bit常量的指令不带任何操作数。将要被装载的Bit值在指令代码中被暗中地编码。

LD_CONST_TRUE

5.6.7.3.10 一般常量的装载指令（IP_OP_LDxx_CONST_xx）

Opcode	Operand
IP_OP_LDxx_CON ST_1	BYTE constant

Opcode	Operand	Operand
IP_OP_LDxx_CON ST_2	WORD constant	

Opcode	Operand	Operand	Operand	Operand
IP_OP_LDxx_CON ST_4 REFERENCE	DWORD constant			

将要被装载的常量作为装载指令的一个操作数被直接地存储在代码段中。

```
PROGRAM TEST_01
VAR
      x : Bit;
      y : byte;
      z : int;
      w : dword;
END_VAR
ld 1
St x
ld 1
st y
ld 1
st z
ld 1
st w
END_PROGRAM
Code:

LD_CONST_TRUE
ST_NEAR_BIT 00 06
LD_CONST_1 01
ST_NEAR_1 00 07
LD_CONST_2 00 01
ST_NEAR_2 00 08
LD_CONST_4 00 00 00 01
ST_NEAR_4 00 0A
RET
```

例5：常量

5.6.7.3.11 NEAR设置指令（IP_OP_STxx_NEAR_xx）

OpCode	Operand	Operand
IP_OP_STxx_NEAR_xx	absolute Offset16 of variable in DS	

这个WORD的操作数指定了变量在“当前”数据段中的从段头开始的绝对偏移量（包括段头长度）。

5.6.7.3.12 FAR设置指令（IP_OP_STxx_FAR_xx）

Opcode	Operand	Operand	Operand	Operand
IP_OP_STxx_FAR_xx	Ptr16 to instance handle		absolute Offset16 of variable in DS	

FAR装载指令带有两个操作数。第一个操作数是一个16位的指向将要被寻址的实例句柄的指针。这个句柄被存储在“当前”数据段内。第二个WORD操作数指定了变量在另一个不同的实例的数据段中从段头（包括段头的长度）开始的绝对偏移量。

5.6.7.3.13 BYREF设置指令（IP_OP_STxx_BYREF_xx）

Opcode	Operand	Operand
IP_OP_STxx_BYREF_xx	Ptr16 to reference	

这个WORD操作数是一个16位的指向期望的变量的引用的指针。这个指针被存储在“当前”数据段中（引用的从段头开始的绝对偏移量包含段头的长度）。这是这个引用的结构：

Ptr16_To_Reference + 0: WORD Instance handle

Ptr16_To_Reference + 2: WORD VarOffset16

这个存储在引用中的实例句柄指定了将要被寻址的数据段。在寻址实例的数据段中，VarOffset16指定了变量的从段头开始的绝对偏移量（包括段头长度）。

5.6.7.3.14 IND设置指令（IP_OP_STxx_IND_xx）

对于IND设置指令，与IND装载指令是一样的。变量的引用被存储在当前引用寄存器中。

5.6.7.3.15 NEAR 位操作指令（IP_OP_S_NEAR_BIT/IP_OP_R_NEAR_BIT）

Opcode	Operand	Operand
IP_OP_S/R_NEAR_BIT	absolute Byte-Offset16 of variable in DS bit is always at position 0	

这个WORD操作数指定了在“当前”数据段中的位的地址。

```
PROGRAM TEST_01
VAR_GLOBAL
      glob_x : int;
END_VAR
VAR
      x : Bit;
END_VAR
ld x
s x
END_PROGRAM
Data segment
      0000 : 09 00 01 06 02 00 00 00
      0008 : 00
Code:
      LD_NEAR_BIT 00 06
      S_NEAR_BIT 00 06
RET
```

例6：位设置

5.6.7.3.16 FAR位操作指令（IP_OP_S_FAR_BIT/IP_OP_R_FAR_BIT）

Opcode	Operand	Operand	Operand	Operand
IP_OP_S/R_FAR_BIR	Ptr16 to instance handle		absolute bit address 13:3	

FAR位操作指令带有两个WORD操作数。第一个操作数是一个16位的指向将要被寻址的实例句柄的指针。这个句柄被存储在“当前”数据段内。第二个WORD操作数指定了在另一个不有实例的数据段中的这个位数据的地址。

5.6.7.3.17 BYREF操作指令（IP_OP_S_BYREF_BIT/IP_OP_R_BYREF_BIT）

Opcode	Operand	Operand
IP_OP_S/R_BYREF_BIT	Ptr16 to reference	

这个WORD操作数是一个16位的指向期望的变量的引用的指针。这个指针被存储在“当前”数据段中（引用的从段头开始的绝对偏移量包含段头的长度）。这是这个引用的结构：

Ptr16_To_Reference + 0: WORD Instance handle

Ptr16_To_Reference + 2: WORD VarOffset16

这个存储在引用中的实例句柄指定了将要被寻址的数据段。在寻址实例的数据段中，VarOffset16指定了变量的从段头开始的绝对偏移量（包括段头长度）。

5.6.7.3.18 创建一个引用（IP_OP_CREATE_REFERENCE）

Opcode	Operand	Operand
IP_OP_CREATE_REFERENCE	absolute Offset16 of variable in DS	

这个WORD操作数指定了变量在“当前”数据段中从段头开始（包括段头的长度）绝对偏移量。变量的引用将被存储在引用寄存器中并且包含了“当前”实例的句柄和期望的变量的偏移量，这个偏移量将作为一个操作数被传递。

实际上，这个指令的名称不是地十分合适。

一个更好的名称是IP_OP_CREATE_REFERENCE_NEAR..

引用的结构如下所示：

Ptr16_To_Reference + 0: WORD Instance handle

Ptr16_To_Reference + 2: WORD VarOffset16

存储在引用中的实例句柄指定了将被寻址的数据段。在寻址实例的数据段中，VarOffset16指定了变量的从段头开始的绝对偏移量（包括段头长度）。这个引用将被写入引用累加器。这里，通过IP_OP_ADD_OFFSET是可进入的并且可通过IP_OP_ST_REFERENCE写到别一个内存位置。变量可通过IND指令来改变。

5.6.7.3.19 从不同的DS创建引用（IP_OP_CREATE_REFERENCE_FAR）

Opcode	Operand	Operand	Operand	Operand
IP_OP_CREATE_REFERENCE_FAR	Ptr16 to Instance Handle		Offset16 of variable in other DS	

这个指令创建一个在不同DS中的变量的引用（这个变量是FB的一个参数）。

FAR位操作指令带有两个WORD操作数。第一个操作数是一个16位的指向将要被寻址的实例句柄的指针。这个句柄被存储在“当前”数据段内。第二个WORD操作数指定了变量在“当前”的数据段中从段头（包括段头的长度）开始的绝对偏移量。变量的引用将被存储于AE并且包含了“当前”实例的句柄和期望变量的偏移量，这个引用将作为一个操作数传递。

5.6.7.3.20 装载引用（IP_OP_LD_REFERENCE）

Opcode	Operand	Operand
IP_OP_LD_REFERENCE	Ptr16 to reference	

这个WORD操作数是一个16位的指向期望的变量的引用的指针。这个指针被存储在“当前”数据段中（引用的从段头开始绝对偏移量包含段头的长度）。

5.6.7.3.21 改变一个引用的偏移量（IP_OP_ADD_OFFSET）

Opcode	Operand	Operand
IP_OP_ADD_OFFSET	relative offset16	

这个WORD操作数指定了将要被加成到偏移量上的值（一个结构体成员的偏移量），这个值存储在引用累加器中（解释程序的内部寄存器）。在调用这个指令前，将要被操作的引用必须已经被创建或者通过下列指令中的一个装载：IP_OP_CREATE_REFERENCE,IP_OP_REFERENCE或IP_OP_SELCT_ARRAY_IND。

5.6.7.3.22 将AE入栈（IP_OP_PUSH_AE）

Opcode
IP_OP_PUSH_AE

这条将AE入栈的指令不带任何参数。栈指针将由内存存取模块内部地管理。这条指令将DWORD累加器推入AE栈。

5.6.7.3.23 将AE出栈（IP_OP_POP_AE）

Opcode
IP_OP_POP_AE

这条将AE出栈的指令不带任何参数。栈指针将由内存存取模块内部地管理。这条指令通过把DWORD累加器的状态出栈来恢复它。

5.6.7.3.24 将外部AE入栈（IP_OP_PUSH_AE_ANY）

Opcode
IP_OP_PUSH_AE_ANY

这条将外部AE入栈的指令不带任何参数。栈指针将由内存存取模块内部地管理。这条指令将外部AE推入AE栈。ANY变量决定了将被存储的字节数（通过它的头两个字节）。

5.6.7.3.25 将外部AE出栈（IP_OP_POP_AE_ANY）

Opcode
IP_OP_POP_AE_ANY

5.6.7.3.26 将引用累加器入栈（IP_OP_PUSH_REF）

Opcode
IP_OP_PUSH_REF

5.6.7.3.27 将引用累加器出栈（IP_OP_POP_REF）

Opcode
IP_OP_POP_REF

5.6.7.3.28 恢复数据段（IP_OP_RESTORE_DATA）

Opcode	Operand	Operand
IP_OP_RESTORE_DATA	Ptr16 to Instance handle	

在OpenPCS-U-Code模式中函数仅像函数块一样对待。唯一不同的地方是它们只用实例化一次。此外，在每次调用前，DS已经被恢复（冷启动）。这就是指令IP_OP_RESTORE_DATA的任务。

这个WORD的操作数是一个16位的指向将要被恢复的实例句柄的指针，这个指针被存储在“当前”数据段中。

5.6.7.3.29 复合操作的指令

Opcode
IP_OP_xxx

这条复合操作的指令不带任何操作数，因为它们通过处理AE和AE`的内容并写回AE。下列为属于这个组的指令：

IP_OP_AND_xx
IP_OP_OR_xx
IP_OP_XOR_xx
IP_OP_ADD_xx
IP_OP_SUB_xx
IP_OP_MUL_xx
IP_OP_DIV_xx
IP_OP_GT_xx
IP_OP_GE_xx
IP_OP_LE_xx
IP_OP_LT_xx
IP_OP_EQ_xx
IP_OP_NE_xx

```
PROGRAM TEST_01
VAR_GLOBAL
	glob_x : int;
END_VAR
VAR
	x : Bit;
	y : Bit;
END_VAR
ld x
and y
s x
END_PROGRAM
Data segment
	0000 : 0A 00 01 06 02 00 00 00
	0008 : 00 00
Code:
	LD_NEAR_BIT 00 06
	LDS_NEAR_BIT 00 07
	AND_BIT
	S_NEAR_BIT 00 06
RET
```

例7：与操作

这些很少使用的指令EXPT,MOD,NOT和MOVE（还没实现）在第二个库。下列为它们的指令（注意指令IP_OP_EXT1必须在它们之前出现）。

IP_EX_EXPT_xx

IP_EX_MOD_xx

IP_EX_NOT_xx

IP_EX_MOVE_xx

IP_OP_MOVE只带有一个参数，它仅使用AE。也不用区别变量是有符号的还是无符号的。这个简单的长度是足够的（xx->1,2或4）。

5.6.7.3.30 类型转换（IP_OP_TYPECAST,IP_OP_TRUNC）

Opcode	Operand	Operand
IP_OP_TYPECAST	source type ("FROM")	destination type ("TO")

通常，类型转移通过IEC-1131的标准函数实现。但是，自从在解释器中的转换比标准函数的调用更快并且更有效率后，OPENPCS的代码生成器使用解释器指令IP_OP_TYPECAST。

类型转换随着AE的当前值进行。两个分开的操作数对资源类型和目标类型进行编码。下列为可能的常量值：

kIpBit1 -> BIT

kIpBit8 -> BYTE, USINT, SINT

kIpBit16 -> WORD, UINT, INT

kIpBit32 -> DWORD, UDINT, DINT

这条指令的第一个操作数指定了将要被转换的值的资源类型。第二个操作数指定了目标类型。操作的靠近0..6的位编码了数据类型的位号。如果靠近的7位打开，这意味着数据类型是有符号的。

对于其他的数据类型，可使用下列计数器。

Data type	Enumerator	Value
real	**IP_TYPE_REAL**	**4**
string	**IP_TYPE_STRING**	**8**
time	**IP_TYPE_TIME**	**12**
tod	**IP_TYPE:TOD**	**16**
dt	**IP_TYPE_DT**	**20**
date	**IP_TYPE_DATE**	**24**
bcd	**IP_TYPE_BCD**	**28**

不是所有可能的组合都可以使用，但是可使用那些IEC声明了的。

一个解释器类型转换操作由两个阶段组成：首先，来自AE的值将被扩展到32位。这个值的符号将通过操作数的值被配置或忽略。然后，这个32位的值将被强制转换为目的数据类型的大小。在这个阶段中，自从这个结果值不必通过解释器计算后，符号就可以被忽略。靠近的7位的约定是否编码一个将要被配置的符号位，这只对错误信息的处理是必要的。例如，可能出现一个类型转换的随机的检查结果。但是解释器的当前的实现不这样做。

指令trunc与指令_TO_的处理是一样的。输入参数通常是实数并且操作码是IP_OP_TRUNC。

```
PROGRAM TEST_01
VAR_GLOBAL
    glob_x : int;
END_VAR
VAR
    x : Bit;
    y : Byte;
END_VAR
ld x
```

BOOL_TO_BYTE

st y

END_PROGRAM

Data segment

0000 : 0A 00 01 06 02 00 00 00

0008 : 00 00

Code Segment

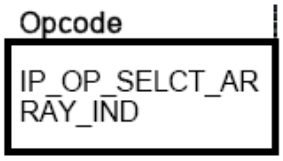
6: 10 LD_NEAR_BIT 00 06

9: 231 TYPECAST 01 00

c: 53 ST_NEAR_1 00 07

f: 7 RET

5.6.7.3.31 直接数组寻址（IP_OP_SELCT_ARRAY_IND）

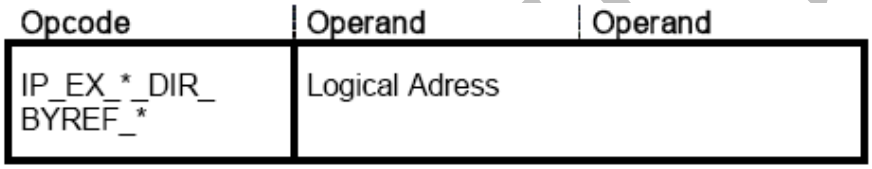


这条INDIRECT选择指令适用于对数据开头的FAR寻址（“段句柄：偏移量”），这个数据存储于引用累加器（解释器的内部寄存器）中。这个地址通过指令IP_OP_CREATE_REFERENCE，IP_OP_LD_REFERENCE存储在引用累加器中。被选择的元素的地址将被写回引用累加器。这个选择指令不带任何操作数。数组存取在5.6.5节中有描述。

将这条指令作为NEAR,FAR或,BYREF通过一个期望的规范执行是没有必要的。

5.6.7.3.32 BYREF-最优化（*_DIR_BYREF_*）

BYREF-最优化加速了全局变量的存取。取代在每次需要存取变量是计算偏移量，它的逻辑地址被计算一次并且在程序存储并下载后被临时插入。



这个DWORD操作数是一个32位的期望的变量的地址。为了激活BYREF最优化，保留段的数量必须在初始文件中增加并且ByrefAccessOpt需要被激活：

[MODULE]

ByrefAccessOpt=1

ReservedSegments=11

此外_LZS_DIRECT_BYREF_必须在config.h文件中定义。

5.6.7.4 字符串处理

OpenPCS3.4以上的版本支持字符串。

下列新的指令必须被添加：

5.6.7.4.1 字符串结构

当存储字符串时需要考虑分配的长度与当前长度一样。分配的长度被存储在首两个字节内。它通过加入的字节的大小，这个大小被写入到变量声明中，表示长度的两个字节和结尾一个额外的’\0’来计算。这允许我们使用ANY装载指令。当前长度通过C约定来编码，例如：字符串通过第一个出现的’\0’结束。

例：

VAR

hallo : string(8) := ‘Str’ ;

END_VAR

分配的长度=8+2+1=11

allocated 0x0B	length 0x00	S	t	r	\0	not valid	not valid	not valid	not valid	\0

5.6.7.4.2 字符串比较

字符串比较必须考虑它们当前长度。它们的分配长度可以不同。这使得有特别的指令的必要。

IP_OP_EQ_ANY

IP_OP_NE_ANY

IP_OP_GT_ANY

IP_OP_GE_ANY

IP_OP_LT_ANY

IP_OP_LE_ANY

将被比较的字符被存储在AE和AEs内。

5.6.7.4.3 字符串存储

如果写入的字节数超过了分配的字节数，多出来的字节将丢失。用户有责任进行适当的字符串分配。

下列为可用的字符串命令。这些参数相当于一般的存储命令。

IP_OP_ST_NEAR_STRING

IP_OP_ST_FAR_STRING

IP_OP_ST_BYREF_STRING

IP_OP_ST_IND_STRING

5.6.7.4.4 字符串处理函数

IEC1131-3提出了一些字符串处理函数。所有的这些指令都在U-code指令的第二个库。他们中没有一条带有参数。这些指令在调用前必须被加载到栈或其中一个累加器中。

这些字符串函数不带类型为ANY_INT的参数，但可带UINT类型的参数。这在IEC1131-3的最低版本中就被提出了。注意有一个合并多个字符串函数的两个参数版本。

IEC-Function	Opcode	AE	AES	Stack	second Stack Elem
Len	IP_OP_LEN	String	-	-	-
Left	IP_OP_LEFT	IN	UINT L	-	-
Right	IP_OP_RIGHT	IN	UINT L	-	-
Mid	IP_OP_MID	IN	UINT L	UINT P	-
Concat	IP_OP_CONCAT	IN1	IN2	-	-
Insert	IP_OP_INSERT	IN1	IN2	UINT P	-
Delete	IP_OP_DELETE	IN	UINT L	UINT P	-
Replace	IP_OP_REPLACE	IN1	IN2	UINT L	UINT P
Find	IP_OP_FIND	IN1	IN2	-	-

5.6.7.5 表 27 中的标准函数

目前，只支持MAX和MIN（只有两个参数）。这些指令是：

IP_EX_MAX_xx和IP_EX_MIN_xx。Xx由数据类型替代（1，1_SGN,..,FLOAT）。

两个参数都被AE和AEs需要，结果被写入到AE。

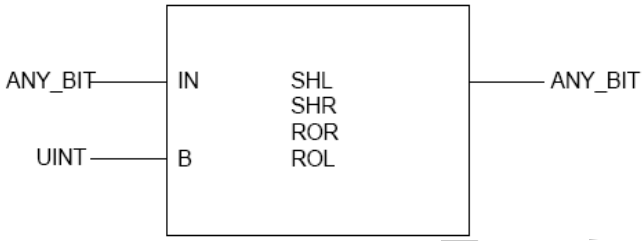
5.6.7.6 数学的函数

所有的数字的函数只带有一个参数，这个参数从AE中读取。结果也被写入到AE中。AEs不使用。

Function	Possible Types	U-Code	Note
ABS	ANY_NUM	IP_EX_ABS_1_SGN, IP_EX_ABS_2_SGN, IP_EX_ABS_4_SGN, IP_EX_ABS_FLOAT	There is no instruction for unsigned types.
SQRT	REAL	IP_EX_SQRT	

LN	REAL	IP_EX_LN
LOG	REAL	IP_EX_LOG
EXP	REAL	IP_EX_EXP
SIN	REAL	IP_EX_SIN
COS	REAL	IP_EX_COS
TAN	REAL	IP_EX_TAN
ASIN	REAL	IP_EX_ASIN
ACOS	REAL	IP_EX_ACOS
ATAN	REAL	IP_EX_ATAN

5.6.7.7 表 25 中的标准函数（标准位移函数）



支持所有的位移函数。指定了位移位数的参数不能是ANY_INT类型但是可以是UINT类型（有IEC中已经有一个更新）。

输入在AE（参数IN）和AEs（参数B）中。结果被写回AE。

5.6.8 段头的布局

布局标准段头

Number of bytes	Content
2	Segment length (=cD)
1	Magic code (tPlcSegxxx)
1	Header length (=cH) = 6
2	corresponding segment

布局代码段

Number of bytes	Content
2	Segment length (=cD)
1	Magic code (tPlcSegCode)
1	Header length (=cH) = 6
2	corresponding initial data segment
N	Interpreter-Code

布局数据段

Number of Bytes	CONTENT
2	Segment length (=cD)
1	Magic code (tPlcSegData)
1	Header length (=cH) = 6
2	corresponding code segment
cIO	Input/Output area (VAR_INPUT, VAR_OUTPUT, ...)
cIOR	Input/Output Retain area
cIH	Instance handle area
cV	normal Variables area
cR	Retain-Variables area

（也可见3.5节固件函数块的表格）

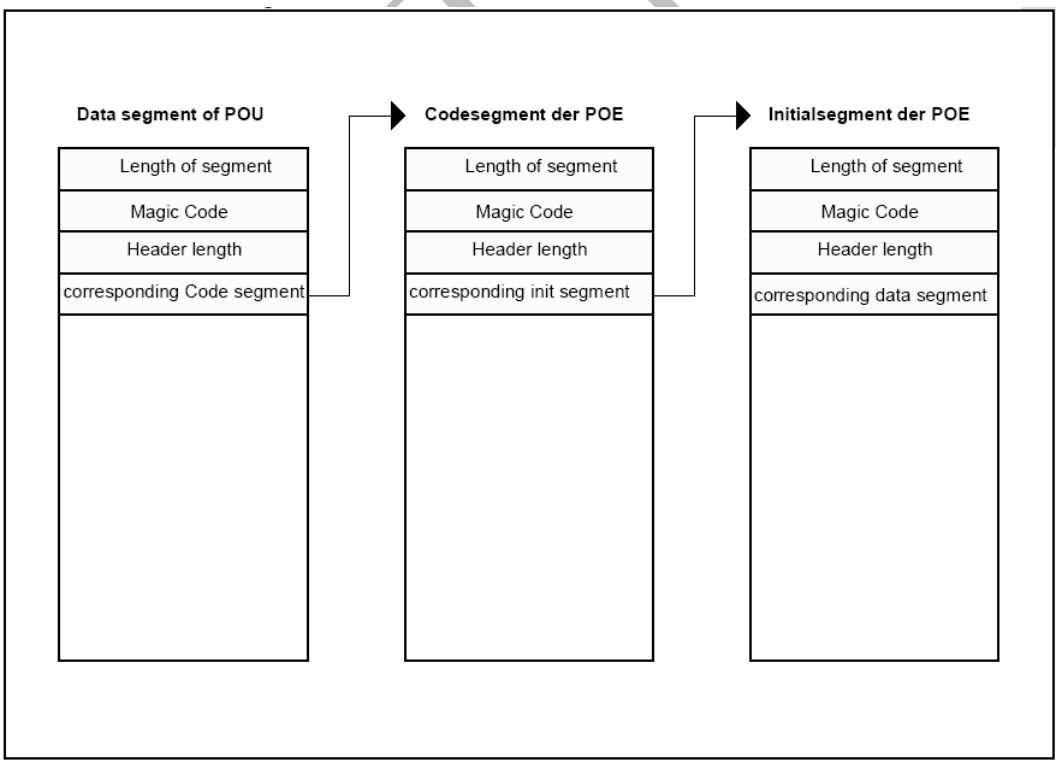
布局初始数据段

Number of Bytes	CONTENT
2	Segment length (=cD)
1	MAGIC CODE (KPLCSEGINIT)
1	Header length (=cH) = 16
2	corresponding code segment
2	Number of Bytes for Input/Output-Variables (=cIO)
2	Number of Bytes for Input/Output- Retain-Variables (=cIOR)
2	Number of Bytes for Instance handles (=cIH)
2	NUMBER OF BYTES FOR NORMAL VARIABLES (=CV)
2	Number of Bytes for Retain-Variables (=cR)
cIO	Input/Output area (VAR_INPUT, VAR_OUTPUT, ...)
cIOR	Input/Output Retain area
cV	normal Variables area
cR	Retain-Variables area

（也可见3.5节固件函数块的表格）

5.6.8 数据段的恢复

恢复数据段不仅在冷启动后是必要的，在一个函数—POU的执行后也是必要的，这是为了将内部变量恢复到它们的配置值。在这种情况下，解释器调用LZS函数LzsMemRestoreDataSeg。它的参数是将要被恢复的数据段的句柄。这个数据段的头包含了POU的代码段的号码（关联的段）。这个代码段的头同样包含了初始数据段的号码。下图举例说明了段是怎样连接的：



在工作数据段中，“I/O变量”和“其它变量”区域必须被重写为初始数据段的值。

5.7 实时系统内存监控

5.7.1 一般说明

内存监控（现在叫做Memmon）是SmartPLC实时系统的一个附加组件，可以被开启或关闭。它只用于调试用途并且提供少数函数来监控实时系统的动态分配的内存的存取（IECMEMORY）。忽略对外部存储硬件推出的分配。它帮助查出内存错误，这些错误很难通过常规调试被找到。在特别的Memmon应该帮助找出下列故障：

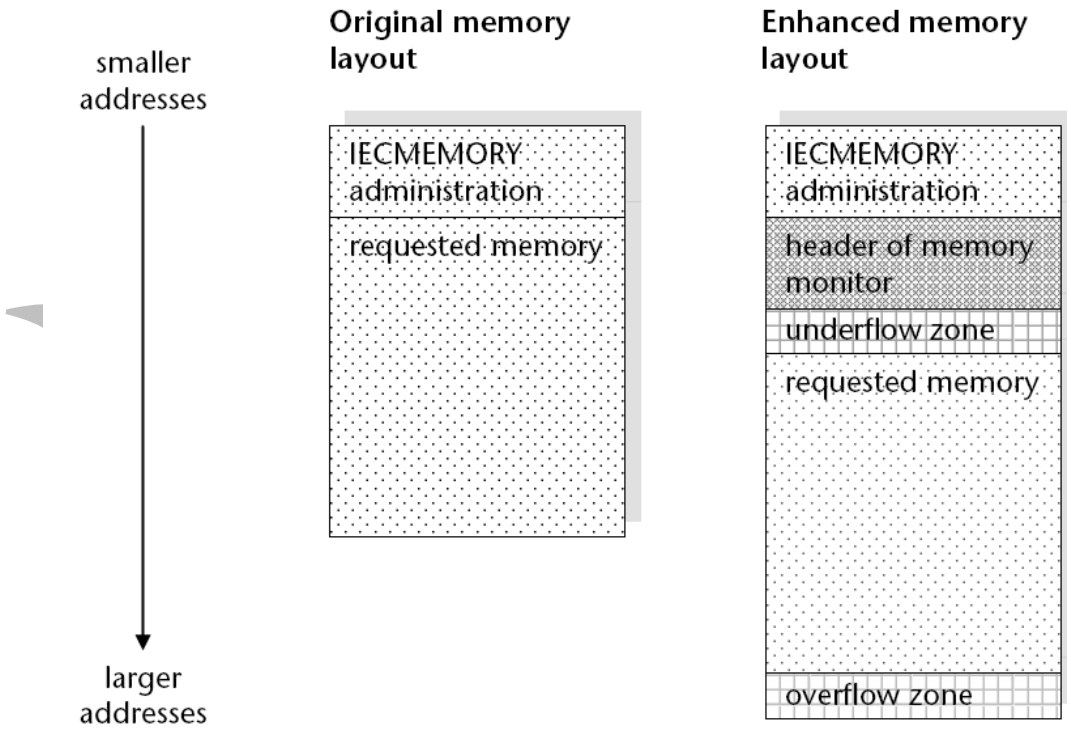
-已释放内存的再利用
-属于一个分配的内存块的边界的存取
-没有正确地分配的内存的使用
-释放以前没有分配的内存
-释放一个内存块两次
-释放一个NULL指针
-在分配的内存使用后丢失其释放

当Memmon被激活后，它将随着自己的实现增加函数LZSMALLOC()和LZSFREE()。所以每次通过LZSMALLOC()的分配执行都将通过一个管理数据头，一个上溢和一个下溢范围来扩展。

管理数据头的用途是注意确定的统计数值的动向。如：一个分配的段的大小。另外它通过形成一个与所有其它的管理数据头的双向连接表来注意所有的有效的分配的动向。这个上溢范围在分配请求之前被直接地定位于可用的内存区域。它由一个独特的特性模式组成因此在管理数据头和可用的内存之间形成很好的缓冲区。这个缓冲区的大小可以是多变的以提高命中某一内存故障的机率。这个下溢范围，与上溢范围形成对照，被定位在请求的内存区域之后。

这个内存布局允许实现一些内存检查，如：在每一次LZSMALLOC()或LZSFREE()Memmon可以测试这个内存监控的数据头和上/下溢范围是否是无错的。

下图显示了增加的内存布局：



5.7.2 使用

本节介绍了Memmon的使用。

5.7.2.1 激活

内存监控可以通过在文件config.h中定义_LZS_USE_MEMORY_MONITOR并重新编译LZS来使能。下次程序

启动后Memmon将修改内存并且通过它的设置进行自动检查。

Memmon通过利用实时系统的追踪器来显示所有的信息，这是可以实现的。

请注意Memmon的激活禁止联机的函数调用（_LZS_SZM_INLINE_）

Memmon的进一步的配置在文件“lzsmemmonitorsettings.h”中。

5.7.2.2 基本步骤

基本步骤如下所示：

-通过改变Memmon的配置来改变IEC内存布局和/或Memmon的特性

-如果Memmon可以通过使能指定的内存检查来探测内存故障，那么进行检查

首先需要考虑的是内存布局怎样在文件“lzsmemmonitorsettings.h”中配置。

作为默认上溢和下溢部分的长度被设置为确定的值：

MEMMON_OVERFLOW_ZONE_SIZE	上溢部分的固定长度
MEMMON_UNDERFLOW_ZONE_SIZE	下溢部分的固定长度

通过逐步地扩大上溢和下溢部分那么如非法的内存重写被发现的机率将可提升。

作为额外选项下溢部分可以通过下列定义被设置为可变的大小：

MEMMON_USE_VARIABLE_UNDERFLOW_SIZE	下溢部分使用可变长度
MEMMON_MAX_UNDERFLOW_SIZE	当使用变长时的下溢部分的最大长度

内存检查由下列定义配置：

MEMMON_CHECK_LIST_CONSISTENCY_IP	在每个解释程序周期后执行一次内存检查
MEMMOM_CHECK_HALT_ON_ERROR	当内存检查失败后停止IEC程序的执行
MEMMON_CHECK_LIST_CONSISTENCY_MALLOC	在每次内存分配期间执行一次内存检查
MEMMON_CHECK_LIST_CONSISTENCY_FREE	在每次内存释放期间执行一次内存检查
MEMMON_CHECK_LOWLVL_MEM_ACCESS	当内存被存取时检查内存块
MEMMON_LOWLVL_MEM_CHECK_DEPTH_DEFAULT	低层内存检查的精度

(*)IEC程序执行的内存存取只检查小容量模式。

注：使能内存检查将降低性能。因此当运行一个较慢的硬件时内存检查应该被个别地使能。

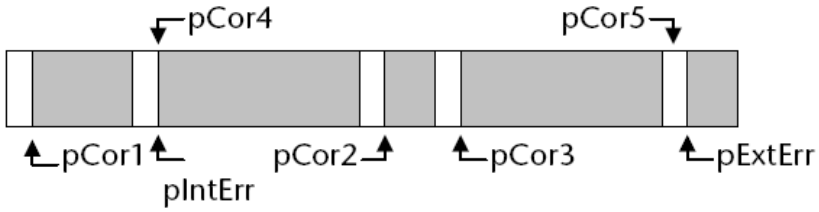
5.7.2.3 示例

下述示例举例说明了错误指针被找到的方式。在这个例子中有5个内存块。pCor1到pCor5是指向它们的指定分配的指针。pIntErr是一个无效的指针且没有被分配，但是仍然指向内存块并被pCor4使用。pExtErr是一个错误的外部指针，如：一些外部模块试图存取定位在pExtErr的内存段。这个外部模块没有检查pExtErr是否是正确的。在没有使能Memmon的时候这个错误是很难被发现的。在Memmon的激活并插入附加的管理信息后，将是一个好机会来找出这个错误的指针。

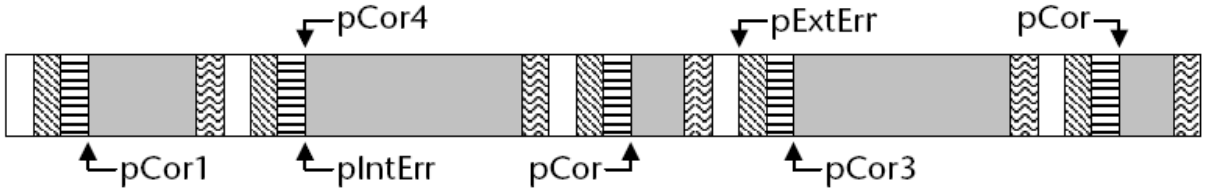
可以在“增强的内存布局1”中看见pExtErr现在指向这个块的管理数据头的右边，这通过Memmon由一个校验和来保护。如果pExtErr修改这个内存，Memmon将报告这个坏头并且生成一条错误信息。但是，pIntErr不能被这个内存布局找出，因为pIntErr仍然指向一个存在的内存块。在这种情况下这个无效的指针pIntErr将在使能上溢部分的大小的变量后被找出。就像“增强的内存布局2”所示，pIntErr现在指向上溢范围的右边并且被Memmon再次找出。

知道不同的设置导致不同的内存布局并且是没有保证的，这是很重要的，一个内存故障会被一个特定设置找出。

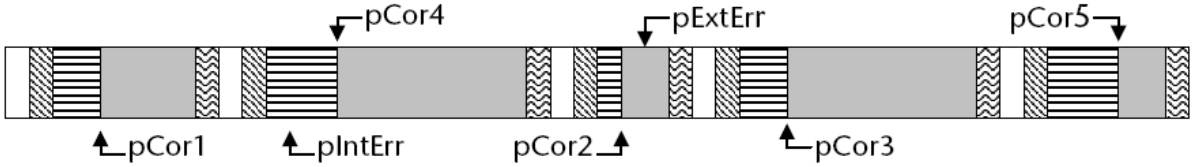
原始内存布局

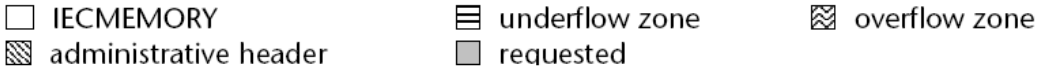


增强的内存布局（固定的上溢部分的大小）



增强的内存布局（变化的上溢部分的大小）





5.7.3 内存监控的接口

5.7.3.1 LzsMemMonitorIntialize

void LzsMemMonitorInitialize(LZSBYTE *pStartOfMem,

LZSDWORD dwSize, LZSBYTE *pExternalMemStart[], LZSDWORD dwExternalMemSize[])

void LzsMemMonitorInitialize(LZSBYTE *pStartOfMem, LZSDWORD dwSize)

这个函数初始化内存监控并且必须在实时系统启动的时候调用（由函数LzsEnvMemInitialize）。作为选项，外部内存可通过使能下列定义指定：

#define MEMMON_USE_EXTERNAL_MEM

指向这个外部内存的指针将通过内存监控的检查来忽略。

5.7.3.2 LzsMemMonitorShowQueue

void LzsMemMonitorShowQueue()

这个函数LzsMemMonitorShowQueue通过内存块表循环并且由LZSTRACE()打印出每个元素的内容。它仅仅是一个输出并且不检查清单一致性，如：下列输出将被完成：

-基地址（指向管理数据头的指针）

-内存块的大小

-被LZSMALLOC调用的文件

-被LZSMALLOC调用的行号

-生成的校验和

5.7.3.3 LzsMemMonitorShowQueueCatalogued

void LzsMemMonitorShowQueueCatalogued(LZSWORD wSort)

这个函数LzsMemMonitorShowQueueCatalogued打印一个所有的分配大小和每个分配大小被配置的频率的清单。这个分类选项由参数wSort指定，如，清单可以通过大小或分配号来存储（见枚举tMemoryListSortMode）。

5.7.3.4 LzsMemMonitorShowTotals

void LzsMemMonitorShowTotals()

这个函数LzsMemMonitorShowTotals打印一些内存使用的统计数值，如下列信息：

-当前分配内存的总字节数

-激活的分配的数量

-分配的总数量（包括空闲的）

-内存使用的最高点

-最大的单个分配

-最小的单个分配

5.7.3.5 LzsMemMonitorShowMemory

void LzsMemMonitorShowMemory(void *pPtr, LZSDWORD dwLen, LZSWORD format)

这个函数LzsMemMonitorShowMemory打印开始于pPrt并且结束于pPtr+dwLen的内存的内容。参数wFormat决定了输出格式，见tMemMonOutputFormat。

5.7.3.6 LzsMemMonitorShowBoundaries

void LzsMemMonitorShowBoundaries()

这个函数打印IEC内存的内存边界：

-IEC内存的起始地址

-IEC内存的大小

-IEC内存的结束地址

5.7.3.7 LzsMemMonitorIsInsideBoundaries

LZSBYTE LzsMemMonitorIsInsideBoundaries(void *pPtr)

这个函数检查指针pPrt是否指向IECMEMORY段内。

返回值：

-1：在边界内

-0：不在边界内

5.7.3.8 LzsMemMonitorCheckList

LZSDWORD LzsMemMonitorCheckList()

这个函数通过管理的清单循环并且检查所有的入口是否是有效的。如果没有问题函数将返回0。如果有地方出错误将返回一条错误代码（见枚举tErrorCodesCheckList）。

5.7.3.9 LzsMemMonitorDistribute

LZSBYTE LzsMemMonitorDistribute(LZSWORD wFunctionId,LZSWORD wParameter, void *pPtr, const LZSCHAR *pFile, LZSDWORD dwLine)

这是一个调用其它内存监控函数的封装式函数。这个函数由参数wFunctionId（见枚举tLzsmemMonitorFunctionId）指定。

5.7.3.10 LzsMemMonitorMalloc

void* LzsMemMonitorMalloc (LZSDWORD dMemSize,const LZSCHAR* pFile, LZSDWORD dwLine)

当内存监控使能的时候，宏LZSFREE将被映射到函数LzsMemMonitorFree替代函数LzsEnvMemFree。为了调用这个函数通常要使用宏LZSFREE。

5.7.4 高级配置

5.7.4.1 LZSMALLOC 的配置

通过使用下列定义为内存的分配指定约束是可以的：

ENABLE_MALLOC_RESTRICTIONS	Enable restrictions of LZSMALLOC
MAX_NUM_OF_ALLOCS	Maximum number of allocations that are allowed to be active at one time
MAX_MEMORY_USAGE	Defines how much memory is allowed to be allocated
MAX_NUM_TOTAL_ALLOCS	Determines how many allocations are allowed in total
MAX_SINGLE_ALLOC	Defines how much memory is allowed to be requested for one single allocation
MIN_SINGLE_ALLOC	Defines the minimum size forone single allocation

如果上述约束中的一个被应用，那么LZSMALLOC将打印一条错误并且宏操作将失败。

5.7.4.2 LZSFREE 的配置

当使能定义MEMMON_CHECK_ELEMENT_VALIDITY时LZSFREF将在指针指向的地址关于下列错误检查内存内容：

-释放一个已经释放的地址（“双重释放”）

-释放一个从未分配的地址

-释放一个分配了但从未初始化的地址

特别对于“双重释放”的检查，释放的地址的最大数量将被记忆，并且可由定义MEMMON_FREE_LIST_ELEMENTS指定。

5.8 实时系统外部控制接口

实时系统可以随意地和一个外部控制接口一起被创建。这个特征打算从目标系统内部来控制实时系统（如：从一个在目标系统上的分离的程序运行）。为了从一个远程（Windows）PC来使用实时系统，在线服务器和它的大量的接口将是推荐的方式（见5.11节在线服务）。

外部控制（extctrl）接口支持：

变量值的获得和设置

系统状态的获得和设置

程序执行的控制（如：启动和停止）

5.8.1 系统概述

接口使用通过一个简单的运行在一个从属平台上的基础文本协议来实现，是一个双向的进程通信（IPC）机。

基本上通过IPC连接一个到实时系统的extctrl组件的客户端。它将发送控制信息。这个extctrl组件接收这些请求。然后在程序的周期结尾实现并且最终通知客户端这些结果。

这个通信被限制在一个客户端上。

5.8.2 协议

这个协议是基础文本。信息主体/数据可以通过一个检验和来保证。

有三种信息类型（READ,WRITE和COMMAND）。描述如下：

基本信息结构：

READ/WRITE message	READ/WRITE response
BEGIN-<message name>	BEGIN-<message name>-RESPONSE
	$timestamp=<integer>
<message body>	<response body>
$checksum=<integer>	$checksum=<integer>
END-<message name>	END-<message name>-RESPONSE

行之间通过一个新行符号（\n）分离。

$timestamp由LzsEnvGetTickCount()给出（如：以ms的形式），是实时系统中数据检索的绝对时间。

READ/WRITE信息细节：

Message	Body	Example
READ message	<variable instance path> < … more variables >	task1.temperature task2.done task3.nonexistantVar
READ response	<variableinstance path>= <variable value> Or =*ERROR*if variable was not found	task1.temperature=22.1 task2.done=TRUE task3.nonexistantVar=ERROR
WRITE message	<variableinstance path>= <variable value>	task1.temperature=22.1
WRITE response	Similar to READ response	

一个完整的信息示例：

BEGIN-WRITE
task1.delta=0.3
task2.struct.value=23
task3.name=dummy
$checksum=0
END-WRITE

变量值约束：

BOOL值为true/false（不分大小写）

整数值为十进制的

系统状态信息可以通过这些预先定义的变量名查询：

Name	Read	Write	Type	Describtion
$system.running	Yes	No	BOOL	程序执行状态
$system.cycletime	Yes	No	DINT	周期时间（ms）
$system.refreshtime	Yes	Yes	DINT	周期更新的刷新时间（ms） 默认1000
$system.LZSversion	Yes	No	String	目标系统版本号 格式 "LzsVersion.LzsType.LzsRevision.OemId". 有下列硬件说明文件中的记录： IpVariante, IpUmfang, IpRevision, IpOem
$system.project-name	Yes	No	String	工程名称，显示在OpenPCS资料属性中
$system.resource-name	Yes	No	String	资源名称，显示在OpenPCS资料属性中
$system.Resource-Version	Yes	No	DWORD	资源版本，显示在OpenPCS资料属性中
$system.errorcode	Yes	No	BYTE	实时系统当前错误代码

READ注脚：

很多应用需要在规律的时间内获取变量值。通过适当地发送READ请求是可行的。但是这里有一个更简单的方法。Extxtrl接口可以被配置为自动为那些通过最后一条READ信息获得的变量在一个给定的刷新时间发送最新的值。为此，LZSEXTCTRL_PERIODIC_UPDATES必须在config.h中被定义。一条带有空主体的READ信息将重置这个注脚。这个更新间隔可以通过系统变量$system.refreshtime来配置。

Command信息可用于启动和停止实时系统：

Message	Response
BEGIN-COMMAND COMMAND=stop|coldstart|hotstart $checksum=0 END-COMMAND	COMMAND=<command> RESULT=ok|error $checksum=0 END-COMMAND-RESPONSE

5.8.3 必需的移植

当前实现可以在Linux上完成，但是可以是其它简单的平台的一部分。

Linux例子是一个好的开始点。文件samples\linux\extctrl\lzsenvextctrlio.c包含了LinuxIPC通信硬件的实现。这里，一个指定的管道被用于IPC。

需求：

这个平台必须支持一些IPC的形式，与线程和同步原语一样。

变量元信息需要在实时系统中被提出，这是为了使用变量的值。这是通过VarTab扩展完成的（见5.3.7.7.2节）。

在文件config.h中定义_LZS_EXTCTRL_以使能extctrl接口。

下列文件必须加入到工程中：lzsshmdata.c,lzsshmctrl.c,lzsshmdebug.c,lzsshmvartab_nonwindows.c lzsshmutil.c, lzsshmctrlclient.c, bintree.c, lzsshmipc_linux.c, lzsshmenv_linux.c,lzsextctrl.c,lzsextctrlprot.c, lzsextctrlvar.c, lzsextctrlcs.c, lzsenvextctrlio.c, lzsenvextctrlcrc.c。

这些文件放于.\Lzs\shm and .\Lzs\extctrl。

有一个通信硬件是必需的。请参见inc/smartplc/extctrl/lzsenvextctrlio.h中需要实现的硬件函数的清单。头文件也应用文件证明这些硬件函数原型。

5.9 OpenPCS 远程控制接口

这个文档解释了可以用于远程控制OpenPCS编程系统的OpenPCS COM接口。

5.9.1 OpenPCS 接口

函数：

CreateOpenPCSInterface

CreateOpenPCSInterface(LPDISPATCH *ppVal)

返回OpenPCSRemoteCtl接口的引用。

ProcessComandlineExternal Function

ProcessComandlineExternal(BSTR bstrCmdLine)

从一个外部客户端像VB一样执行ProcessComandline()函数，并且以这种方式打开文件。

5.9.2 OpenPCSRemoteCtl 接口

方法：

Close

Close()

关闭当前打开的工程。关闭意味着保存所有。

GetActionList

GetActionList(BSTR NodeID,SAFEARRAY(BSTR)* Actions)

这个方法返回活动的清单，这个清单可以与NodeID后的节点一起被调用。

活动需要以字符串标识。下列活动可被执行：

Action	Description
edit	opens POU
print	prints POU
go_online	opens resource; is only available if and only if the most recent build did not fail
build	builds resource
buildall	builds all resources
monitor	opens POU in online mode
rename	renames file without the file extension. Calling convention: rename?newname
check_syntax	performs a syntax check on the POU

GetStatus

GetStatus(DWORD* StatusFlags)

StatusFlags包含：正在进行的当前文件打开、在线的编辑，修改。下列的标志被执行：

#define OPENPCSIF_STATUS_ISCOMPILING 0x00000001

#define OPENPCSIF_STATUS_ISONLINE 0x00000002

#define OPENPCSIF_STATUS_MODFILESOPEN 0x00000004

Invoke

Invoke(BSTR NodeID, BSTR Action)

调用活动在指定的节点。活动见GetActionList（）。参数用?分离。

例：Invoke（“SOURCE/file.st”,”rename?newfilename”）；

Open

Open(BSTR bstrVarFilePath, VARIANT_BOOL bForceCreation)

这个方法打开一个存在的工程。bForceCreation指定是否创建不存在的工程。打开意味着关闭当前打开的工程。

Refresh

Refresh()

扫描磁盘和有关的文件（.VAR,代码库）并且相应地更新树。

Resize

Resize(DWORD X, DWORD Y, DWORD Width, DWORD Height, DWORD ResizeFlags)

调整OpenPCS窗口的大小。ResizeFlags指定了允许哪种大小选项（左，右，上，下，最小，最大）。见Win32 API的SetWindowpos()的标志。

SaveAll

SaveAll()

如果有修改，保存所有打开的文件到磁盘。

Scan

Scan(BSTR NodeID, IOpenPCSNodeList** NodeList)

通过一个包含完整名称（字符串）、基本名称（字符串）、一个DWORD的NodeFlags和一个NodeType的结构体识别的节点。

AppendMessage

AppendMessage(BSTR Text)

这个自动的方法附加“文本”的字符串到OpenPCS中的输出窗口里的文本的结尾。

GetCurrentProjectPath

BSTR GetCurrentProjectPath()

GetCurrentProjectPath返回当前打开的工程的路径。

GetCurrentOpenFilesAbsolutePaths

BSTR GetCurrentOpenFilesAbsolutePaths()

GetCurrentOpenFilesAbsolutePaths返回当前打开的文件的绝对路径，以换行符分隔。

CloseOpenFileByAbsolutePath

CloseOpenFileByAbsolutePath(BSTR FileToClose)

CloseOpenFileByAbsolutePath关闭一个当前打开的文件，这个文件由字符串“FileToClose”表示的绝对路径指定。

OpenFileByAbsolutePath

OpenFileByAbsolutePath(BSTR FileToOpen)

OpenFileByAbsolutePath打开由字符串“FileToOpen”表示的绝对路径指定的文件。

5.9.3 OpenPCSNodeList 接口

是一个可以被用于浏览工程树目录的迭代列表工程。两个属性都返回一个到带OpenPCSNode类型节点的接口。

属性：

First

First(LPDISPATCH *ppVal)

返回第一个OpnePCSNode的接口。

Next

Next(LPDISPATCH *ppVal);

返回在OpenPCSNodeList工程中的下一个OpenPCSNode的接口引用。

5.9.4 OpenPCSNode 接口

OpnePCSNode是一个到OpenPCS浏览树中的节点的接口。Nodes应该在一个分层的图表SECTION\PATHNAME上被识别，SECTION和PATHNAME如下所示：

SECTION	PATHNAME
SOURCE	path to source file, relative to project root, e.g. DIR1\DIR2\MYFILE.ST
CONFIG	path to instance of variable or code block CONFIG\resource\task.fb1.fb2.myvariable
LIB	Not implemented yet
TYPE	Not implemented yet

属性：

Attributes

Attributes(DWORD *pVal)

IsVisible	The node is visible in the browser tree
IsParent	The node is a parent node with child nodes

Name

Name(BSTR *pVal)

在浏览树中显示的名称。

PathName

PathName(BSTR *pVal)

这个路径相当于在NodeID后的物理文件的工程目录。

Type

Type(CXNodeType *pVal)

NodeType被大致地以一种方式定义：一个NodeType相当于在存在的浏览中的一图标。更多细节列表：

CX_UNDEF	Unknown node type
CX_CFC	base type CFC
CX_CFC_PROGRAM	CFC program node
CX_DATA	Base type for physical files
CX_DIRECTORY	Directory node
CX_EXTERNAL	Node is an external file
CX_FORM	Not supported yet
CX_GLOBAL	Global definition node
CX_GLOBAL_DIRECT	Direct global definition node
CX_IL	Base type IL
CX_IL_PROGRAMkBtfILProgram	IL program node
CX_IL_FUNCTION	IL function node
CX_IL_FUNCTIONBLOCK	IL function block node
CX_LD	Base type ladder
CX_LD_PROGRAM	Ladder program node
CX_LD_FUNTIONBLOCK	Ladder function block node
CX_LIBRARY	Library node
CX_POU	Base type POU
CX_POU_PROGRAM	POU program node
CX_POU_FUNCTION	POU function node
CX_POU_FUNCTIONBLOCK	POU function block node
CX_PROJECTFILE	OpenPCS project file node
CX_RESOURCE	Base type resource nodes
CX_RESOURCE_FB	Function block instance node
CX_RESOURCE_FUNCTION	Function instance node
CX_RESOURCE_MAKE	Resource node
CX_RESOURCE_TASK	Task instance node
CX_RESOURCE_VAR	Base type for variable nodes
CX_ROOT	Base type for parent nodes
CX_ROOT_CONFIGURATION	Head of the configuration tree
CX_ROOT_FILES	Head of the files tree
CX_ROOT_PROJECT	Head of the project
CX_SFC	SFC program node
CX_SOURCE	Base type for all sources
CX_ST	Base type for ST
CX_ST_PROGRAM	ST program node
CX_ST_FUNCTION	ST function node
CX_ST_FUNTIONBLOCK	ST function block node
CX_TYPE	Type definitions
CX_RESOURCE_GLOBAL_VAR	global variable instance node
CX_RESOURCE_DIRECT_GLOBAL_VAR	direct global variable instance node
CX_RESOURCE_ARRAY_VAR	Array instance node
CX_RESOURCE_ARRAY_TYPE	

在openpcs.idl中的枚举CXNodeType中定义。

5.10 OpenPCS32 位 UCODE 生成工具

本节包括infoteam涉及创建的工具，如：编译器，连接器和制作。你可以从这些开发环境或命令行来创建你的程序。这里包括这些代码生成工具的32位实现，又叫做PCE32。

5.10.1 概述

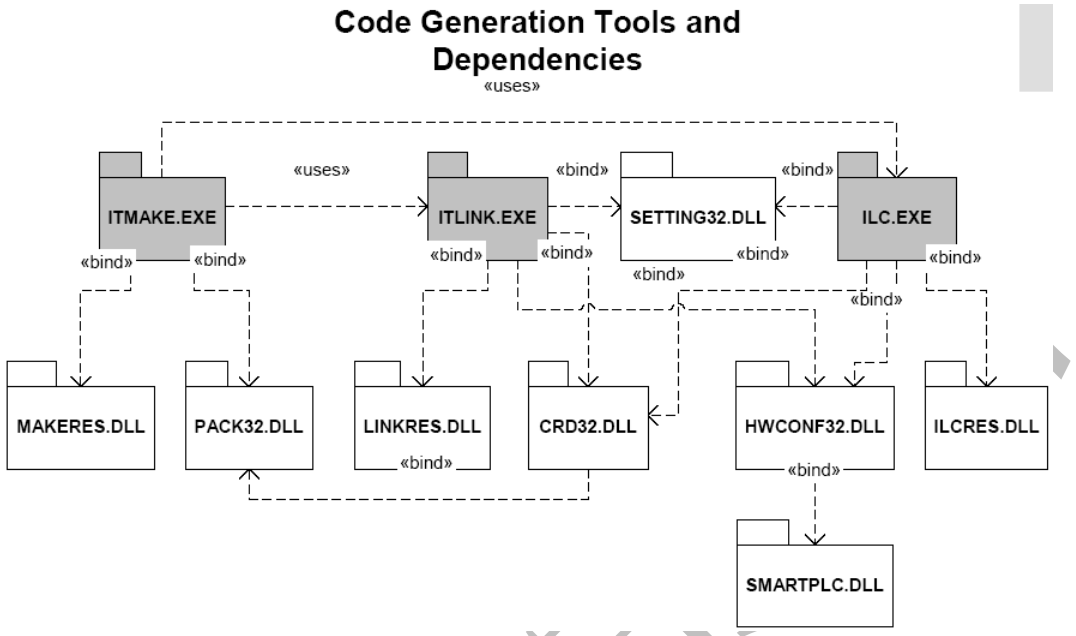


图92 代码生成工具

5.10.2 ITMake

本节包括infoteam制作工具。

5.10.2.1 用户接口

ITMake从一个工程或所有在指定工程中的资源创建一个或多个资源。

ITMake语法

ITMake [-p | (-m <makefilename> | -u <ArchiveFileName> –v) | -y]

<Varfilename> { -o <OutputDir> | <Outputfilename> } {-i } { -s } {-w<OutputLevel>} {-b} {-4 <string>} {-5 <string>}

这些命令和选项通过一个斜线(/)或一个破折号(-)优先并且不区分大小写。在一个调用中有多样的命令是不被允许的。

命令：

• -p：创建所有在指定工程中的资源。这条命令后必须跟至少一个工程文件路径（VAR-file-path）。

• -m：创建指定资源。这条命令后必须跟一个资源文件名称的列表并且说明符-v后跟工程文件路径。将要被创建的所有的资源，必须都是通过VAR-file-name指定的工程的一部分。

• -u：压缩档案。这条命令后必须跟一个包含被压缩的档案的文件名。

• -y：创建在工程中的所有POU的初始数据段并且以文本文件写出。这条命令应被用于获得固件POU为<hardware>.ini文件的初始数据段。

选项：

• -o：如果一个单独的资源被创建，随着这个选项用户可以指定一个输出目录或目标路径名称。如果-o后跟了一个目录名称，那么这个目标将被存储在这个指定的目录下。

• -e：这个选项是为进一步的实现保留的。它使得用户可以指定一个输出信息的目标路径。如果指定了一个错误文件，那么用户输出将替代stdout被打印进指定的文件。如果-e后跟了一个目录名，那么会为每次编译与连

接实体创建一个错误文件并且这个错误文件被存储在指定的目录下。

• -i：递增的创建。如：只改变文件和被这个改变影响的文件将被重新编译和连接。

• -s：使用短文件名。ITMake假定名称长度大于8个字母的POU的文件名将被剪切到8个字母。

• -w：通过使用这个标志用户可以指定输出信息的输出标准。它的后面必须跟一个表明输出标准的整数。下面的值是为“输出标准“的定义：

• 0：打印所有可用的信息。如：错误，警告和消息。

• 2：隐藏警告

• 4：隐藏消息（如：Compiling c:\test\test.poe）

• 6：隐藏警告和消息

请注意错误信息通常会被打印且不能隐藏。

• -b：使用短文件名。如果这个选项被指定并且一个POU的名称大于8个字母，那么编译器将使用这个POU的前8个字母作为其名称。

• -n：隐藏版权信息。

5.10.2.2 组件

Infoteam维护工具是一个叫做ITMake的可执行文件：

ITMAKE和依赖关系

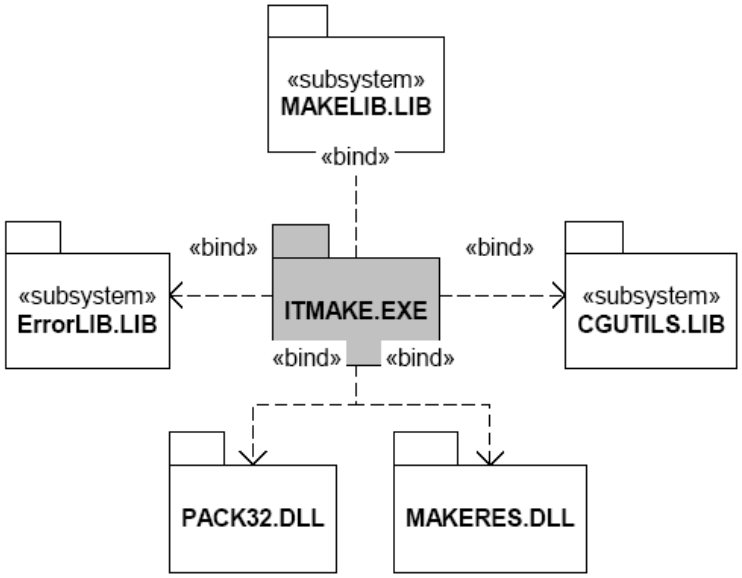


图93：ITMAKE

5.10.2.2.1 ITMake

可执行文件ITMake.exe实现infoteam制作工具的用户接口。它从命令行读取变元，解释它们并且委托给一个IITMakeInterface工程的适当的成员函数来执行。

这个IITMakeInterface在静态库MakeLib中被执行。错误输出和其它用户输出信息通过一个IErrorHandler工程处理，这个IErrorHandler在静态库ErrorLib中被执行。为了获得来自VAR-和MAK-文件的需要的信息，静态库CGUtils将被连接到这个可执行文件。这个库包含了一个对一般INI文件操作的分析程序，infoteamVAR文件、MAKE文件和硬件初始化文件操作的实现和对文件和文件名处理的一般函数。对于这个库的更多细节信息请见本文档的相关章节。

除了上述提到的静态库，动态连接库PACK32.DLL被IITMakeInterface实现使用来压缩和解压缩这些为一个资源上传和更新所需要的文件。进一步地可执行文件ILC.EXE和ITLink.exe被用于编译POU并且连接任务和将要创建的资源。

5.10.2.2.2 MakeLib

静态库MakeLib实现维护工具的需求，如：创建工程，创建资源功能和解压存档特征。它作为一个静态库来实现而不是作为可执行文件的一部分，以此减轻不同的用户接口的实现。例如：如果一个动态连接库被需要来替

代一个可执行文件，那么只有实现用户接口的这部分需要被改变。来自MakeLib的IITMakeInterface实现可以通过连接这个库到工程中来被包括进去。

对于连接的工程的接口见下列列表或头文件\P431CodeGen\MakeLib\ITMakeIf.h.。

类IITMakeInterface：

Initialize	初始化
GenerateProjects	生成工程
GenerateMakeFiles	生成生成文件
UncompressArchive	解压档案

IITMakeInterface是一个纯理论的接口定义。这个接口的实现类是MakeLib的一部分，但是不输出。作为一个结果类型为IITMakeInterface的工程不能被直接地实例化。工程创建必须委托给MakeLib，MakeLib通过调用C函数CreateITMake()来完成。如果实例化成功了这个函数返回一个指向IITMakeInterface工程的指针。否则将返回一个NULL指针。

在使用一个IITMakeInterface工程为解压一个存档创建一个或多个资源或工程前，它必须通过方法Initialize()初始化。这个函数必须提供一个指向一个IErrorHandler工程的指针，这个指针被MakeLib用于用户输出。这个指针必须不为NULL，它应该是一个指向错误处理工程的指针并且被调用的应用程序或DLL使用。

成员函数GenerateProjects(…)使调用程序创建一个或多个工程。这些工程的名称被编译后，必须放入一个CStringArray中并传递给这个函数。这个函数的第二个参数的内容为bool类型并且指定这个资源是否需要被递增地创建。如果目前的参数是TRUE，将执行递增的创建。此外所有编译器生成的文件（*.ppt,*.inc.*.lst,*.Obj,*.crd,*.pcd）都将被删除和重建。第三个参数使调用程序可以为输出（错误）消息指定一个文件。这个参数的缺省值是一个空字符串。在这种情况下输出消息将被打印在stdout上。

成员函数GenerateMakefiles(…)被用于从一个工程创建一个或多个资源。第一个参数指定了这个工程文件的路径（如：VAR文件）。第二个参数的内容是CStringArray类型的。这个数据保存了将要被创建的资源的名称。第三个参数指定是否执行一个递增或完全的创建。对于每个在上述CStringArray aMakeFiles_p中指定的资源都是无效的。参数strErrorFile_p使调用程序为输出（错误）消息指定一个文件。这个参数的缺省值是一个空字符串。在这种情况下输出消息将被打印在stdout上。

函数UncompressArchive(…)被嵌入到接口中，允许浏览器通过这个32位代码生成工具解压压缩存档。这个压缩存档从被第一个参数srtCompressedArchive_p指定的文件中读取，解压并且写到被第二个参数srtUncompressedArchive_p指定的文件中。

5.10.2.2.2.1 IITMakeInterface实现

在MakeLib.lib中ITMakeInterface通过类CITMake实现。取决于需要被执行的操作，它自己执行这些操作或委托给其他的工程。如果一个存档必须被解压，CITMake工程通过调用来自PACK32.DLL的相应的函数来执行这个操作。

如果需要创建一个或多个工程（通过方法GenerateProjects调用）那么所有属于这些工程的资源都将被决定并且方法GenerateMakefiles为每个工程所调用。这个CITResource工程是GenerateMakefiles的一个局部变量。它随着函数调用被实例化并且在函数的调用栈消失的时候被删除。

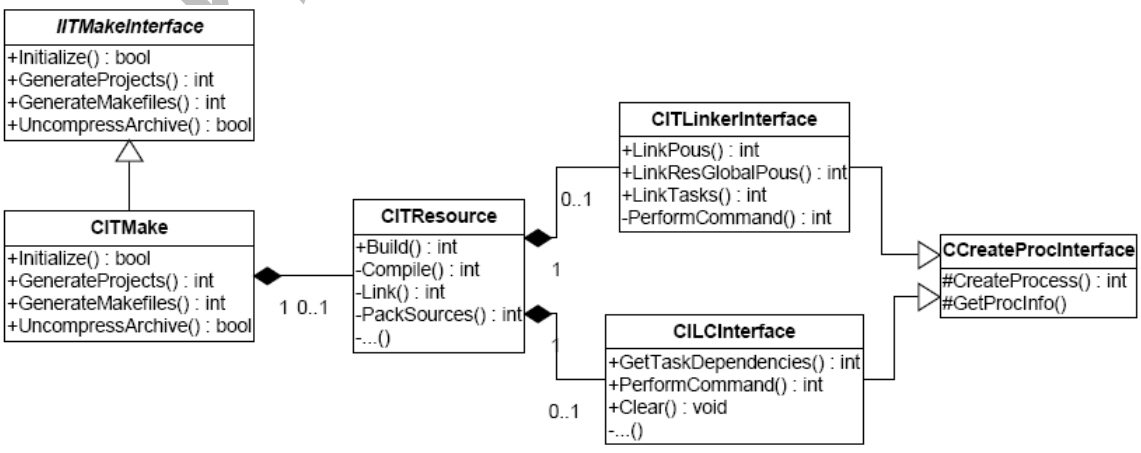


图94 MakeLib

CITResource工程的Build-方法决定了所有的依赖关系，检查哪个文件需要被（重）编译，如果需要调用编辑并且如果编辑成功了调用联接器。

由于一个任务的依赖关系不明确地存储在其中一个工程文件中，且暗中的声明在POU中，这些依赖关系的信息必须在每次一个资源被创建的时候完成。这个功能在ILC.EXE中实现因为它已经包含检查这些声明的分析程序。所以如果一个资源的依赖关系需要被更新，那么CITResource工程需要由相应的命令调用ILC.EXE。如果需要生成原型和/或头文件或者如果POU需要被编译，ILC.EXE也通过CITResource调用。为了简化在CITResource中的这些操作的调用，在封装类CILCInterface中实现ILC.EXE调用和它的结果求值的功能。

另一个在创建一个资源时使用的可执行文件是ITLink.exe。这个程序的调用和它的结果求值在封装类CITLinkerInterface中实现。

两个封装类都必须为可执行文件的调用提供代码。为了避免在两个类中实现，这个功能已经在类CCreateProcInterface中被实现了。这个类被作为一个CILCInterface和CITLinkerInterface的基类并且可以作为每一个必须创建另一个进程的其他类的基类使用。

5.10.3 ITLink

5.10.3.1 用户接口

Infoteam连接器（ITLINK.EXE）连接输入工程信息到一个包含一个任务或资源的工程信息。只有资源工程是可执行的。程序组织单元的工程必须被连接到任务工程并且一个或多个任务工程可以被连接到一个资源工程上。

ITLink语法：

ITLink [-r | -t] <source file names> -v <Varfilename> -m <makefile name> {-g <resource global object filename>} {-s <packed sources file> } { -o <OutputDir> } {-x} {-2 <float>}

这些命令和选项通过通过一个斜线(/)或一个破折号(-)优先并且不区分大小写。在一个调用中有多样的命令是不被允许的。

命令：

- -r：连接在命令后文件列表中指定的文件到一个资源工程中（*.pcd文件）。
- -t：连接在命令后文件列表中指定的文件到一个任务工程中（*.crd文件）。

在两种情况下连接器都需要这个工程文件的路径和生成文件路径。工程文件路径必须以’-v’开始，生成文件必须以’-m’开始。

选项：

- -o：如果创建了一个单独的资源，那么指定一个输出目录或目标路径名。如果-o命令后面跟有一个目录名那么目标将被存储在这个指定的目录下。
- -g：包含关于资源全局变量的工程信息的输入文件（这些文件在命令后被指定）
- -s：这条选项指定的文件包含了资源的封装源。这个文件的内容将被嵌入到资源全局段表中。这个选项只在结合命令’-r’是才有效。
- -x：跳转工程文件。
- -w：通过使用这个标志用户可以指定输出信息的输出标准。它的后面必须跟一个表明输出标准的整数。下面的值是为“输出标准“的定义：
  - 0：打印所有可用的信息。如：错误，警告和消息。
  - 2：隐藏警告
  - 4：隐藏消息（如：Compiling c:\test\test.poe）
  - 6：隐藏警告和消息

请注意错误信息通常会被打印且不能隐藏。

5.10.3.2 组件

可执行文件ITLINK.EXE实现infoteam连接器的用户接口。它的组件和依赖关系如图2-3所示。

ITLink和依赖关系

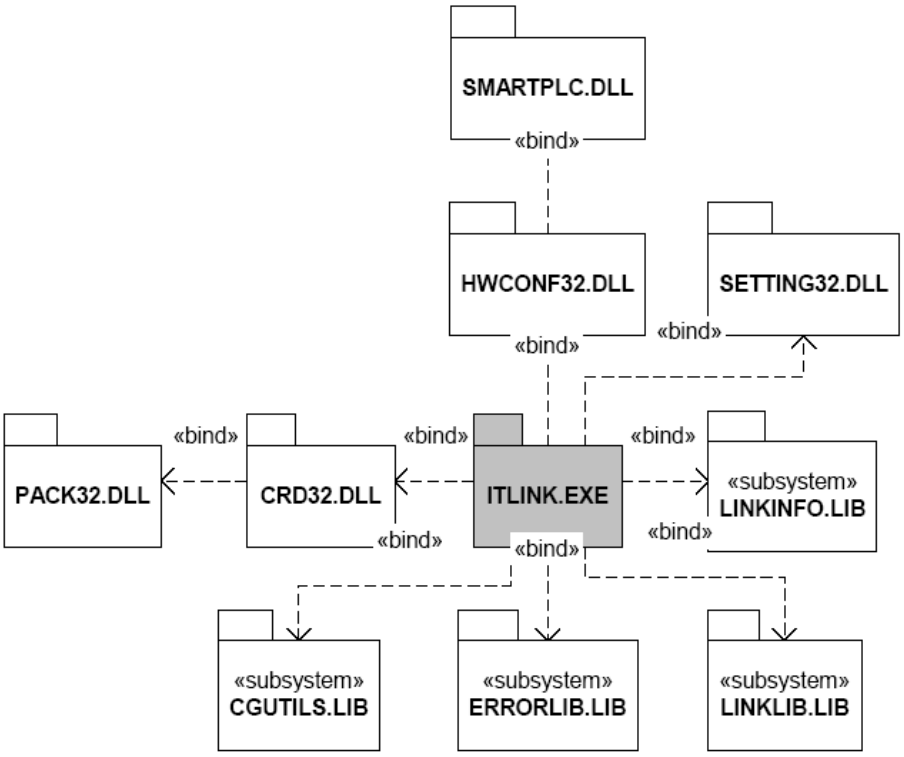


图95：ITLINK

5.10.3.3 ITLink

可执行文件ITLink.EXE从命令行中读取参数，解释它们并且将执行委托给一个IITLinkInterface工程的适当的成员函数完成。这个IITLinkInterface在静态库LinkLib中实现。如果这条被请求的命令已经成功地执行，那么连接器的输出是一个任务工程文件（*.crd）还是一个资源工程文件（*.pcd）取决于已经被连接的是一个任务还是一个资源。只有资源工程文件可以被下载到实时系统。其它种类的工程文件，通过infoteam代码生成工具生成的是POU工程文件（*.obj）。这些工程文件通过ILC.Exe生成并且在实例化和连接进程期间被ITLink使用。这此工程需要的信息通过类型IITLinkInfo的实例存储和序列化。这些工程信息类的实现在LinkInfo.Lib中实现。

错误输出和其它用户输出信息通过一个IErrorHandler工程处理，这个工程在静态库ErrorLib中实现。为了从VAR-和MAK-文件中获得需要的信息，静态库CGUtils将被连接到可这个可执行文件。这个库包含一个对普通INI文件操作的解析器，对infoteamVAR-文件，MAK-文件和硬件初始化文件操作的实现和对文件和文件名称处理的一般函数。对于这些库的更多详细信息见本文档的相关章节。

除了上述提到的静态库，INLINK.EXE还装载动态连接库CRD32.DLL，用于创建和序列化已连接的实体的代码资源库树。这个DLL依次装载PACK32.DLL来压缩被需求的段。

进一步地HWCONF32.DLL被用于分解分配的变量的固件实例的地址。这个DLL结合了一个动态硬件配置器的从属硬件实现。

5.10.3.4 LinkLib

静态库LinkLib实现infoteam连接器的功能，例如：任务和资源的实例化和连接。它已经被当作一个静态库来实现而不是当作可执行文件的一部分，这是为了减轻不同用户接口的实现。例如，假设一个动态连接库需要被一个可执行文件替代，那么仅仅实现用户接口的部分需要被改变。在LinkLib中实现的ITLinkerInterface可通过连接这个库到工程而被包含进去。

5.10.3.4.1 接口说明

连接器工程的接口见如下列表或文件\P431CodeGen\LinkLib\ITLinkIf.h。

ITLinkerInterface类：

Initialize

LinkTask

LinkGlobalTask

LinkResource

IITLinkerInterface是一个纯接口定义。这个接口的实现类是LinkLib的一部分，但是不输出。由于IITLinkerInterface类型的一个结果对象不能被直接地实例化。工程的产生必须委托给LinkLib，LinkLib通过调用C函数CreateITLinker()来完成。如果实例化成功这个函数将返回一个指向一个IITLinkerInterface工程的指针。否则返回一个NULL指针。这个对象可通过使用析构函数删除（如：通过使用这条命令删除返回指针）。

在使用一个IITLinkerInterface对象连接一个任务或资源之前，它必须通过方法Initialize()初始化。必须向这个函数提供对象文件（*.VAR文件）的路径，资源文件（*.MAK文件）的路径和一个指向IErrorHandler对象的指针，这个指针被这个对象用于用户的输出。这个指针不能是NULL，它必须是一个指向错误处理对象的指针，这个指针通过调用应用程序或DLL来使用。

LinkGlobalTask(…)被用于实例化全局资源变量并且将它们插入到全局资源段表中。第一个参数必须是一个CStringArray类型的数组并且以包含这些变量的对象文件名命名。第二个参数是输出文件的名称，第三个参数指定了输出信息的文件名。这个参数有一个空字符串作为缺省值。如果没有文件名被指定，那么输出信息将被打印到stdout。

LinkTask(…)被用于实例化用在任务中的POU实例并且为指定的任务将相应的段插入到段表中。第一个参数必须是一个CStringArray类型的数组并且保存任务所需的POU对象文件的名称。第二个参数指定了包含全局资源信息的任务对象文件的名称。这个文件必须已经由IITLinkerInterface对象中的方法LinkGlobalTask创建。最后一个参数是输出文件的名称并且这第三个参数为输出信息指定了一个文件名。这个参数有一个空字符串作为缺省值。如果没有文件名被指定，那么输出信息将被打印到stdout。

LinkResource(…)被用于实例化和连接所有的任务的全局资源变量。第一个参数必须是一个CStringArray类型的保存属于资源的任务对象文件的名称的数组。这个任务对象文件必须已经由IITLinkerInterface对象中的方法LinkTask创建。第二个参数指定了包含全局资源信息的任务对象文件的名称。这个文件必须已经由IITLinkerInterface对象中的方法LinkGlobalTask创建。最后一个参数是输出文件的名称并且这第三个参数为输出信息指定了文件名。这个参数有一个空字符串作为缺省值。如果没有文件名被指定，那么输出信息将被打印到stdout。

5.10.3.4.2 IITLinkInterface实现

在LinkLib.lib中IITLinkerInterface通过类CITLinker实现。一个CITLinker对象必须在其可以用于连接一个任务或资源之前被初始化。

为了连接一个资源，下列步骤必须由已给出的命令完成：

资源全局变量必须通过对IITLinkerInterface对象的方法LinkGlobalTask的调用来连接到资源全局任务。这些POU的对象文件（*.obj）包含了必须被作为输入文件传递的资源全局变量。

资源的每个任务必须通过调用IITLinkerInterface对象中的LinkTask来连接。资源全局任务的对象文件（*.crd）和被任务包含（直接或间接）的POU的对象文件必须作为输入参数传递。

这些资源通过调用IITLinkInterface对象中的LinkResource来连接。资源全局任务和所有其他属于这个资源的任务的对象文件（*.crd）必须作为输入参数传递。

鉴于需要被连接的对象(全局任务（*.crd）、任务（*.crd）或资源（*.pcd）)的类型，CITLinker创建一个适当的对象来执行这个操作。当调用连接一个资源全局任务时，将有一个CGlobalTaskLinker对象被作为CITLinker对象的方法LinkGlobalTask的一个本地成员创建。CGlobalTaskLinker对象是初始化的并且连接方法被调用来创建一个资源全局任务并且将资源全局变量连接到这个任务。CITLinker对象的方法LinkTask使用一个类型为CTaskLinker的本地变量并且在对象初始化后通过调用它的连接方法来完成对这个对象的请求操作。当CITLinker对象的方法LinkResource被调用时，一个本地的CResourceLinker对象将被创建、初始化并且它的成员函数Link将调用来连接这个任务到一个资源。CGlobalTaskLinker，CTaskLinker和CResourceLinker类都有一个相同的基类：CLinkBase。

如果在硬件说明文件中指定了一个原代码DLL，那么将有一个CNCCIf对象被创建并且被CTaskLinker对象调用来生成原代码。

被连接器使用和创建的对象信息通过IITLinkInfo类型的对象保存且序列化。

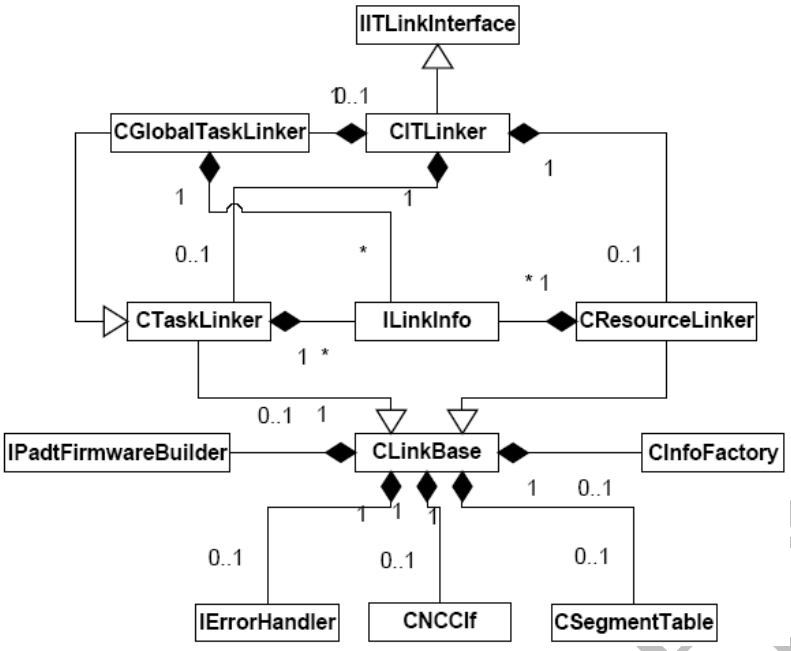


图96 ITLinkInfo

5.10.3.5 LinkInfo

5.10.3.5.1 接口定义

ILinkInfo是一个为共同的POU（*.obj）、任务（*.crd）和资源（*.pcd）对象信息的存取、操作和序列化的方法定义的接口。对于连接器对象的接口见如下列表或头文件\P431CodeGen\Include\ILnkInfIf.h。

类ILinkInfo：

WriteToFile

ReadFromFile

ReadFromArchive

GetLinkInfoKind

GetLinkInfoIdent

GetLinkInfoIdent

SetCrdNode

GetCrdNode

GetGlobalTable

InsertGlobalTable

GetMaxSegmentNr

SetTaskNr

SetSegmentNumbers

GetFileName

GetName

GetPath

Dump

DumpToFile

序列化方法WriteToFile，ReadFromFile和ReadFromArchive允许适当的对象从一个文件或一个CArchive对象写它的信息到一个文件中并且初始化它的内容。这些函数的第一个参数指定应该被初始化的对象的来源或应该被分别存储的对象的目标。如果这个对象被写入，那么这个目标指定一个文件路径到对象信息应该被存储的地方。如果对象被读取，那么这个来源可以是一个文件也可以是一个CArchive对象的引用。序列化函数的第二个参数是一个指向IPadtDevice对象的指针。如果一个任务被连接那么这个指针被用于获得必须被写入对象文件的硬件相关的信息，或用于从一个对象文件分别传递硬件相关的信息到硬件配置程序。

存储在ITLinkInfo对象中的信息的结构和内容与POU，任务和资源是不一样的。方法GetLinkInfoKind允许一个ITLinkInfo实例的用户获得关于哪种类型的对象被存储在这个实例中的信息。这些有效的类型被定义在枚举tLinkInfoKind中。方法GetLinkInfoIdent用一个字符串返回对象信息的类型。如果没有参数被传递，那么实例的对象类型将被使用。如果当调用GetLinkInfoIdent时传递了一个对象类型，将使用于传递参数。

一个对象文件的主要信息被存储在一个代码资源库树（CRD）中。ILinkInfo对象必须保持指针指向根节点。这个代码资源库通过编译程序（如是它是一个POU-CRD）或连接程序（如果它是一个任务或资源CRD）创建并且被连接程序使用。通过使用方法SetCrdNode，CRD树可以被传递给ILinkInfo对象或通过使用GetCrdNode从这个对象中获得。

另一个需要被存储在每个ILinkInfo对象中的信息是全局表。这个表包含了POU、任务和/或资源的全局变量的定义。通过调用GetGlobalTable用户可以获得指向这个全局表的指针。方法InsertGlobalTable允许调用者将一个表的内容插入到另一个表中。如果一个任务的全局表必须被加入到一个资源的全局表中时，这是很有用的。

成员函数GetMaxSegmentNr，SetSegments，GetName和GetPath是在CRD树中存取相应成员的封装函数。

GetFileName返回用于ILinkInfo对象的初始化的对象文件的名称。Dump和DumpToFile将存储在ILinkInfo对象中的信息以ASCII文本的格式在stdout或一个文件中打印出来。

ILinkInfo是一个抽象类定义。这意味着它不用实例化。ILinkInfo对象即可以通过直接地调用实现类的构造函数创建也可以使用CLinkInfoFactory对象的方法CreateLinkInfoObject或方法ReadObjectFile中的一个来创建。这个类的类定义如下表所示并且可以在文件\P431CodeGen\Include\LnkInfIf.h.中找到。

class CLinkInfoFactory

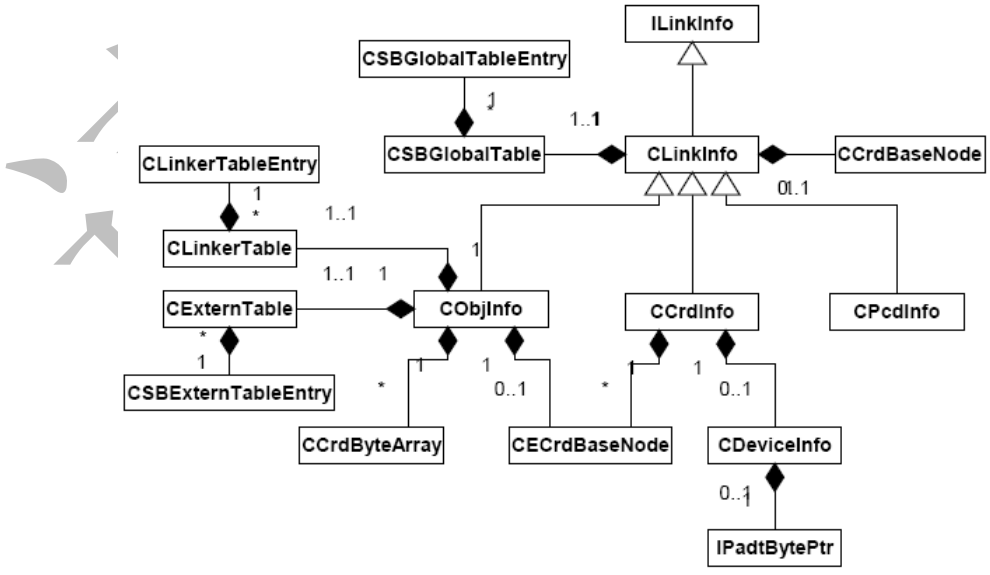
CreateLinkInfoObject

ReadObjectFile

方法CreateLinkInfoObject创建一个由tLinkInfoKind指定类型的空实例。第二个参数用于错误输出。它应该指出应和这个对象一起被分配的文件名。ReadObjectFile使用一个已经存在的对象文件来创建和初始化一个ILinkInfo对象。这个必须被创建的LinkInfo对象的类型从对象文件中读取。

5.10.3.5.2 接口实现

静态库ILinkInfo包含了ILinkInfo接口的实现。对于POU的ILinkInfo接口在类CObjInfo是实现。CCrdInfo实现任务对象信息的接口，CPcdInfo实现资源对象信息的接口。被所有三个类需要的功能在它们共同的基类CLinkInfo中实现。



当一个POU被编译的时候类CObjInfo的对象通过ILC.EXE创建，并且在一个任务被连接的时候被ETLINK.EXE使用。对于每次编译POU都有一个新的CObjInfo类型的实例被创建并且所有为连接程序所需的数据都存储在这个对象中。如果编译已经成功，那么这个对象将在文件中被序列化，这个文件同序列化函数的调用来指定。这个对象包含了一个CCrdBaseNode类型的指针，这个指针指向POU的代码资源库树（CRD树）的根节点；一个ECrdBaseNode类型的指针，这个指针指向扩展代码资源库树（ECRD树）的根节点；还包含了三个

CCrdByteArray类型的指针，分别指向POU的数据、代码和初始段。这个类型的第四个指针被保留用于本地的代码段并且它通常是NULL。CRD树包含一个对所有用在POU中的变量的说明，ECRD树包含一个对所有用在POU中的声明的说明。由于CCrdBaseNode指针同时也被CCrdInfo和CPcdInfo所需要，所以它是基类CLinkInfo的一个成员。

进一步地一个CObjInfo的一个实例包含一个类型为CSBGlobalTable的映射，一个类型为CSBExternTable的映射和一个类型为CLinkerTable的映射。CSBGlobalTable包含了一个对每个在POU中定义的全局变量的入口。这个入口是CSBGlobalTableEntry类型的并且包含了所有关于全局变量的信息，这些全局变量被ITLINK.EXE需要。CSBExternTable包含一个对每个在POU中定义的外部变量的类型为SCBExternTableEntry的入口。如此的一个入口为外部变量保持变量信息，这些变量信息为ITLINK.EXE所需要。CLinkerTable为每个被POU使用的函数块实例和函数保持信息。这些信息被存储在类型为CLinkerTableEntry的对象中。CSBGlobalTable实例也需要CCrdInfo和CPcdInfo，因此它是基类CLinkInfo的一个成员。

作为调用ITLINK.EXE来连接一个任务的结果，类CCrdInfo被创建，并且当连接这个资源全局任务与另一个任务或一个完成的资源被连接时，类CCrdInfo将会被ETLINK.EXE使用。像类CObjInfo一样，这个类的实现是从它的基类CLinkInfo继承一个指向一个CRD节点的指针，这个指针保持了一个指向任务代码资源库树的根节点，和一个CSBGlobalTable类型的成员来在任务（如：程序POU）中保持全局变量的定义。此外它还包含一个指向一个CDeviceInfo对象的指针和指向CECrdBaseNode实例的指针的映射。类CDeviceInfo实现了硬件相关数据的序列化。这些硬件相关的数据作为一个IPadtBytePtr从硬件配置程序的IPadtDevice实例中获得。CECrdBaseNode实例是指向所有被这个任务使用的POUECRD树根节点的指针。ECRD信息在每个POU中被存储一次而不是在每个POU实例中。

类CPcdInfo的对象是被实时系统使用的可执行文件。它们由ITLINK.EXE创建。它们同样从基类CLinkInfo中继承CCrdBaseNode指针和CSBGlobalTable实例。它们不包含任何额外的数据成员。这个类型为CPcdInfo的实例的CCrdBaseNode指针指向资源代码资源库树的根节点。这个树包含了所有被其它OpenPCS工具所需的信息。全局表仅在资源被连接并且没有序列化的时候被需要。

5.10.4 ILC

5.10.4.1 用户接口

ILC在指定的文件上编译或执行语法检查。

指令语言编译程序（ILC）的语法：

ILC [-s | -c | -p | -i | -l] <Poe file names> -v <Varfilename> -d <Devicename> -r <ResourceName> { -o <OutputDir> } {-g | -m } {-x} { -b } {-y} { -w<OutputLevel> } {-2 <float>}

ILC command <Poe file names> -v <Varfilename> -d <Devicename> -r <ResourceName> options

这些命令和选项由一个斜线（/）或一条破折号（-）分隔并且不区分大小写。允许一次调用多条命令。

命令：

- -s：在该命令后的文件上执行一个IL语法检查
- -c：编译所有该命令后的文件
- -p：为所有的在该命令后的文件中指定的列表中的程序组织单元创建原型
- -i：为所有的在该命令后的文件中指定的列表中的程序组织单元创建头文件
- -l：为所有的在该命令后的文件中指定的列表中的程序组织单元创建一个带所有依赖关系的文件

如果在一次ILC.EXE调用中使用了超过一条的命令那么一条简单的命令适用于所有跟在该命令后的文件。

选项：

- -o：有了这个选项如果一个简单的资源被创建，用户可以指定一个输出目录或目标路径名。如果-o后跟了一个目录名，目标将被存储在这个指定的目录下。
- -g：输入文件（*.poe文件）不分配资源全局变量
- -m：输入文件（在命令后指定的文件）分析资源全局变量
- -x：转储对象文件
- -f：隐藏最终的错误信息

• -b：使用短文件名。ILC规定POU的文件名长度大于8个字母的将被剪切至8个字母

•-w：通过使用这个标志用户可以为输出信息指定输出层次。它必须跟一个整数，用于表明输出层次。对“输出层次”的定义如下所示：

• 0：打印所有可用的信息：如错误、警告和消息

• 2：隐藏警告

• 4：隐藏消息（如：Compiling c:\test\test.poe）

• 6：隐藏警告和消息

请注意错误消息会一直被打印且不可以隐藏。

•-y：在一个文档文件中写入将要被编译的POU的初始数据段。这条命令用于为<hardware>.ini文件获得固件POU的初始数据段

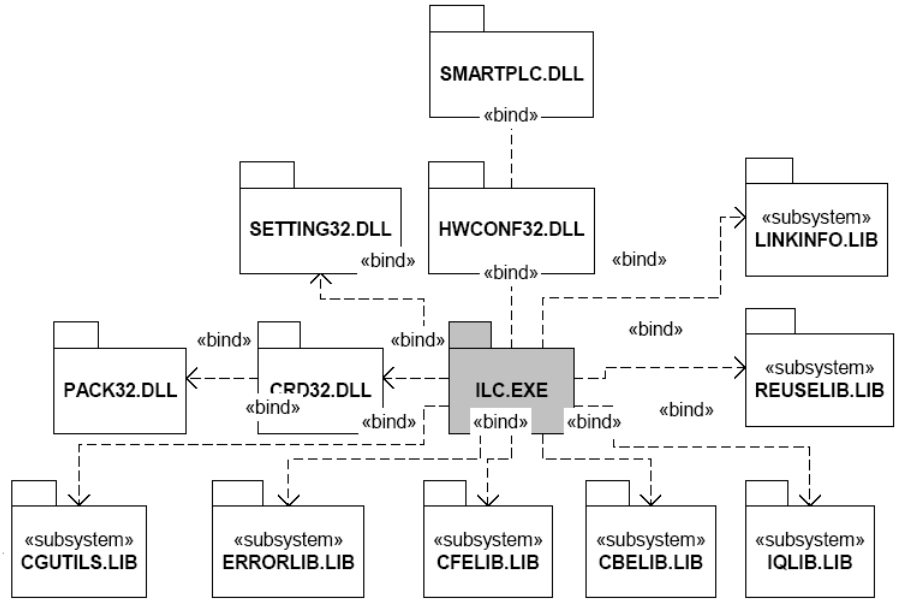
• -d：没有.ini的hardware.ini文件

• -r：资源名称

• -n：隐藏版权信息

5.10.4.2 组件

ILC和依赖关系



5.10.4.3 CfeLib

类IILCFEInerface：

CreatePrototype
CreatePrototype
CreateIncludeFile
CreateIncludeFile
CreateLinkList
CreateLinkList
SyntaxCheck
SyntaxCheck
ReleasePouData
Release

类IILPouDataContainer：

GetTree
GetReuse
GetFinalIECType
GetFinalCBEType
GetCBETypeOfFinalIECType
GetIdentString
GetIdentStringCaseSensitive
GetReuseStrLen
GetReuseString
GetNoPosition

5.10.4.4 IqLib

5.10.4.5 ReuseLib

5.10.4.6 CbeLib

类IILCBEInterface：

ILCompile

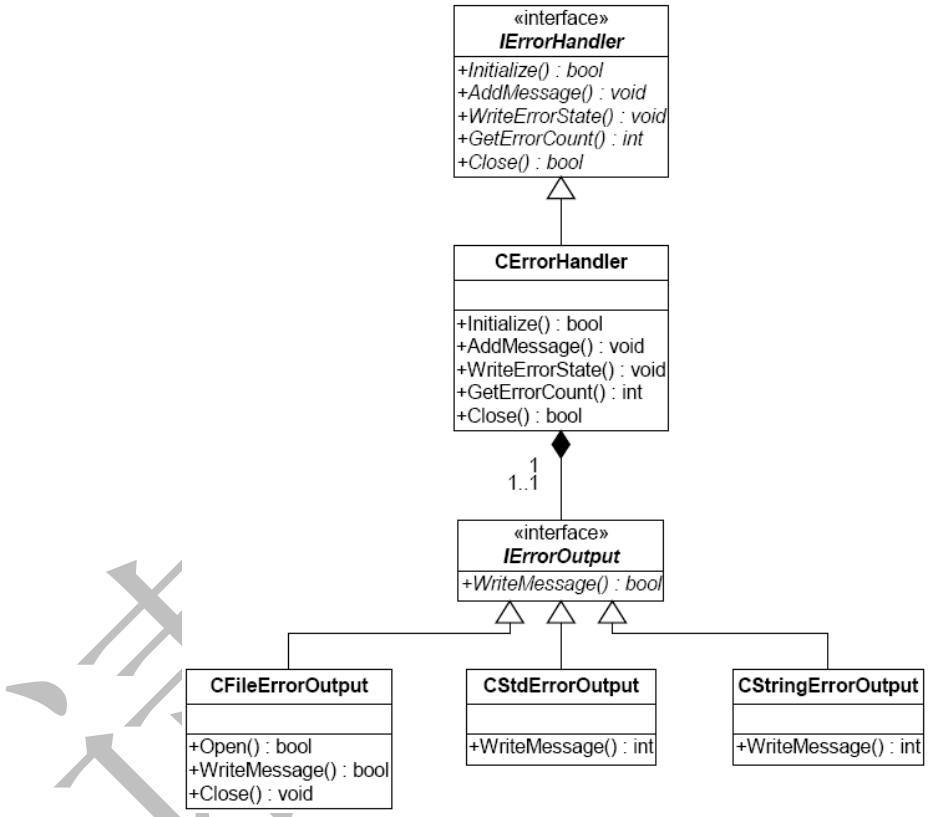
5.10.4.7 LinkInfo

5.10.5 通用库

5.10.5.1 ErrorLib

ErrorLib是一个静态库，实现类的错误处理。

类IErrorHandler：
Initialize
AddMessage
WriteErrorState
GetErrorCount
Close



5.10.5.2 CGUtils

5.10.5.2.1 PrivateProfileParser

类CprivateProfileParser：
Init
GetPrivateProfileInt
GetPrivateProfileString
GetProfileSection
SetPrivateProfileInt
SetPrivateProfileString
WritePrivateProfileSection
ClearSection
InsertNewSection
GetFilePath
SetFilePath

5.10.5.2.2 PrivateProfileSection

类CprivateProfileSection：
Init
GetPrivateProfileInt
GetPrivateProfileString
SetPrivateProfileInt
SetPrivateProfileString
WriteSection
ClearSection
SetFilePath
GetFilePath
SetSectionName
GetSectionName

5.10.5.2.3 Arguments

类CArguments：
Procede
GetCommand
GetCommandCount
GetInputFiles
GetOptions
GetInputFileCount
GetOptionCount
GetOutputDir

5.10.5.2.4 Datparam Info

类CDatparamManager：
Init
UpdateGenFile
UnregisterPrototype
UnregisterAllPrototypes:
RegisterPrototype RegisterAllPrototypes
RegisterAllPrototypesOfSection
RegisterExecutables
RegisterLibraries

5.10.5.2.5 Make Info

类CMakeFileInfo：
Init
GetDeviceClassName
GetTaskName
GetInterruptName
GetTaskType
GetPriority
GetTaskId
GetTaskEntryNum
GetOptimization
GetNetDependencies
GetTime
GetPackSourcesFlag

5.10.5.2.6 Devive Info

类CDeviceFileInfo：
Init
GetPceDeviceName
GetPCodeDll
GetMotorolaFlag
GetResGlobalSegTableFlag
GetRetainVarFlag
GetNoBuildDateEntry
GetAlignment
GetFractLayout
GetOffsetIn
GetOffsetOut
GetOffsetMarker
GetInputSize
GetOutputSize
GetMarkerSize
GetNetOffsetIn
GetNetOffsetOut
GetReservedSegments
GetDownloadTableFlag
GetDebugDownloadFlag

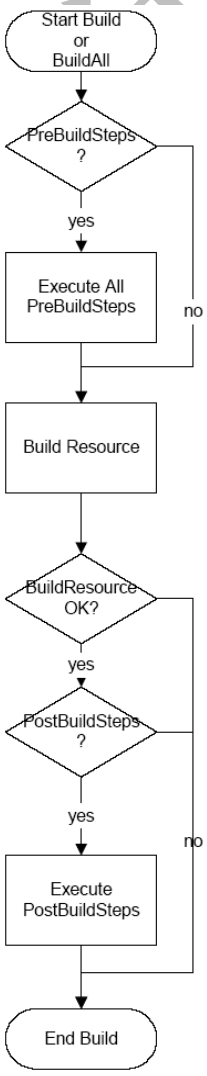
GetOneSegTblFlag
GetOnlineEditorFlag
GetIncrementalTaskDownloadFlag
GetTaskSeparatlyStartableFlag
GetMaxWatchEntries
GetByteSize
GetHwVersion
GetHwRevision
GetFwVersion
GetFwRevision
GetIpVersion
GetIpRevision
GetIpType
GetIpOEM

5.10.5.2.7 File Functions

CreateDirRecursive
GetProjectPath
GetFileName
NewExtention
GetTempFile
GetFileEntryNum
FileExists

5.10.6 构造前和构造后步骤

在版本5.2.0以上的infoteam OpenPCS编程系统，定义构造前和构造后的步骤成为可能。这些额外的步骤必须在硬件说明文件中定义（如：smartsim.ini）因此具有硬件特性。有这这些步骤一个OEM就可以在构造一个资源前做任何的准备或在构造一个资源后执行额外的命令。



构造进行的执行流程

5.10.6.1 构造前和构造后步骤部分和入口

为了定义构造前步骤和构造后步骤，你必须分别加入一个[PREBUILDSTEPS][POSTBUILDSTEPS]节到硬件说明文件中。在这个节中下列入口是有效的：

Count=

PBSn=

Count给出了在本节中列出的构造前步骤的数量。PBSn是一个构造前步骤入口，n为1~N且表示Count中给出的数字。

在PBSn=的右边你需要写入应该被执行的命令行。OpenPCS使用API函数CreateProcess来执行这条命令。因此可执行文件应该被分配到以OpenPCS安装目录开始的标准搜索路径下。

在命令行了可使用一些占位符。在执行这些命令前这些字符串将被以下相应的值替代：

$WorkDir$	Output drirectory for all generated files with ending “\” i.e.: C:\Documents and Settings\All Users\Application Data\infoteam Software\OpenPCS2008\samples\controlx\GEN\resource\
$ResName$	The resource name i.e.: RESOURCE
$IniPath$	The complete path to the corresponding hardware description file i.e.: C:\Documents and Settings\All Users\Application Data\infoteam Software\OpenPCS2008\Openpcs.520\modules\smartsim.ini
$MakePath$	The complete path to the corresponding make file i.e.: C:\Documents and Settings\All Users\Application Data\infoteam Software\OpenPCS2008\samples\controlx\ENV\resource\resource.mak
$ProjPath$	The complete path to the projects var file i.e.: C:\Documents and Settings\All Users\Application Data\infoteam Software\OpenPCS2008\samples\controlx\controlx.var
$ProjDir$	The project root directory with “\” i.e.: C:\Documents and Settings\All Users\Application Data\infoteam Software\OpenPCS2008\samples\controlx\
$BuildAll$	Replaced by 1 if the user used “Rebuild All” and 0 if the user used “Build”

注：所有字符串被替代时都没有引号。因为路径可能包含空格，在这样的命令行中需求使用引号。

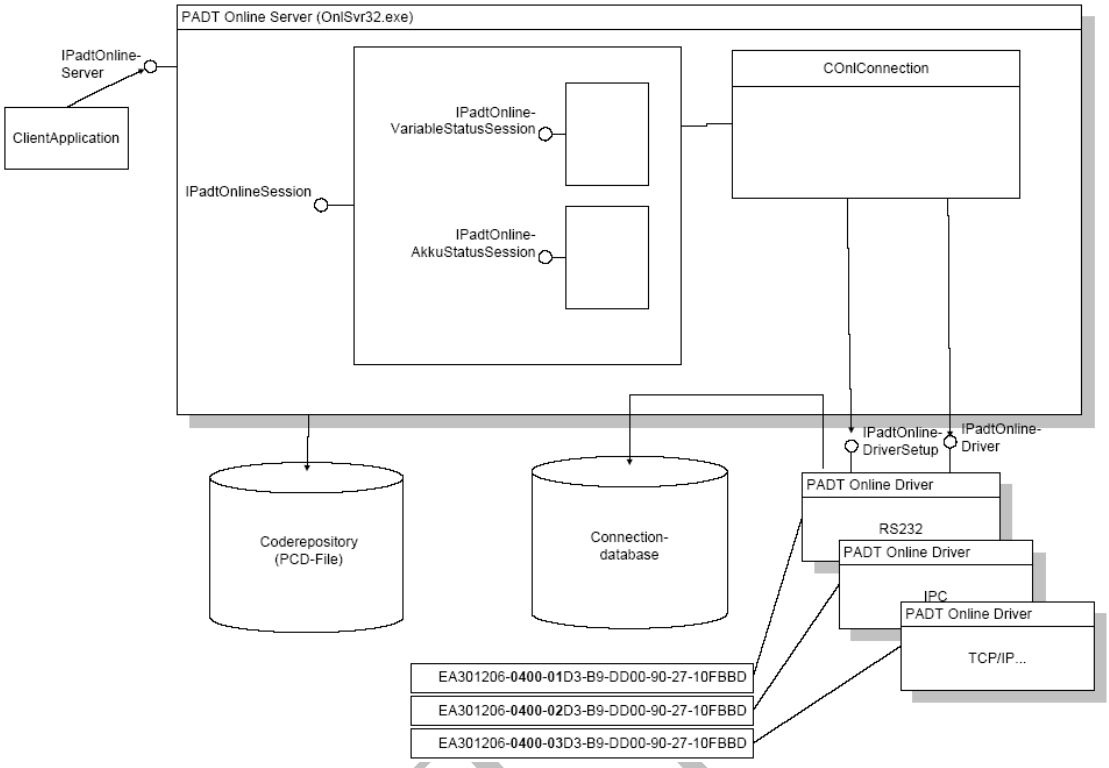
下面的示例显示一个调用工程目录作为参数的MyApp.exe的构造前步骤。第一个构造后步骤第二次调用持续创建（第一次在资源创建时）但是使用不同的输出文件。第二个构造后步骤显示一个编译程序工作/输出目录的列表目录。

```
[PREBUILDSTEPS]
Count=1
PBS1=MyApp.exe "$ProjDir$"
[POSTBUILDSTEPS]
Count=2
PBS1=PADTRTPS.EXE -s "$WorkDir$$ResName$.pr1" -pcd="$WorkDir$$ResName$.PCD"
PBS2=cmd /c dir "$WorkDir$"
```

5.11 联机服务器

5.11.1 基本要素

5.11.1.1 架构



这个联机服务器是一个32位的COM本地服务器（OnlSvr32.exe），它拥有一个输出接口IPadtOnlineServer和三个内部接口IPadtOnlineSession，IPadtOnlineVariableStatusSession及IPadtAkkuStatusSession。这个服务器可以有多个连接且每个连接可以附属多个用户。一个用户附属一个连接也叫做“online-session”，用接口IPadtOnlineSession的一个实例表示。一个连接有一个联机服务器的一个实例，这个实例实现联机协议的物理层。联机协议的软件层在联机服务器中的ConlConnection的实例中运行。有一些驱动程序用于不同的物理层，如：RS232,IPC,TCP/IP等。这些驱动程序设置的诸如波特率、端口…被存储在一个连接数据库中。在连接数据库有一个连接名称标志每个驱动程序数据的设置。这些驱动程序由它们的GUID区别，所有驱动程序的GUID都是一样的除了6个数字。4个数字用为一个用户ID（infoteam用0400），其它的两个数字作为序号识别驱动程序。所以每个用户可有255个不同的驱动。考虑到与其它OpenPCS用户的兼容性，我们强烈建议你在你的对这个用户ID进行移植支持期间使用OEM-ID（见文件oem_def.h中对OEM_ID的定义）。如果有疑虑，请询问infoteam。

为了从一个实时系统中检索联机数据联机服务器需要关于下载的资源的类型信息。这个类型信息被存储在被编译程序创建的代码资源库中（PCD文件）。你也可以在没有PCD文件设置的情况下进行联机，但是将没有可用的功率流和变量状态。

因此为了进行联机（打一个OnlineSession接口）你需要下列信息：

- 连接名称
- PCD文件路径（如果没有，Online-Server将试图使用上次下载的PCD文件路径）

5.11.2 获得联机

首先你需要一个OnlineServer的接口，这通常在你的用户应用程序中由像CreateObject或CoCreateInstance一样的函数完成。如果你使用VisualBasic，你可以通过onlsvr32.tlb类型库插入接口。如果你使用C++ Visual Studio，你可以使用下列头文件：

PadtOnlineServerNotifyID.h	MessageID’s for callback messages.
PadtOnlineServer_i.h	Declaration of the Online-Server interfaces

PadtOnlineServer_i.c Declaration of the GUIDs of the Online-Server

并将它们包含进你的工程。在你可以使用任何OnlineServer函数前，你必须用IPadtOnlineServer::Init（BSTR strConnectionDatabase）初始化这个服务器；BSTRstrConnectionDatabase必须对应一个到连接数据库的有效的路径。连接数据库拥有你可以使用的带缺省连接数据库的扩充*.con.OpenPCS片。

请使用OpenPCS连接安装工具声明和安装连接。

获得联机意味着实例化了一个IPadtOnlineSession类型的接口。这由下列函数完成：

IPadtOnlineServer::GetOnlineSession(BSTR strConnectionName_p, BSTR strPCDFilePath_p, BSTR strReserved_p,long lDialogMode_p,LPDISPATCH *ppDispatch_p);

strConnectionName_p是需要的连接的名称。你可以由枚举函数IPadtOnlineServer::GetFirstConnectionName和IPadtOnlineServer::GetNextConnectionName得到一个可用连接的列表。strPCDFilePath_p是到你想要联机的资源的代码资源库的路径。如果你使这个字符串为空，联机服务器将试图使用之前下载到这个资源上的路径。如果没有可用的路径，联机状态为不可用。strReserved_p必须为空。lDialogMode_p=ture表示联机服务器处于对话模式，如：联机服务器将显示消息；=false表示它不处于此模式。ppDispatch_p是对Online-Session（联机会话）需要的接口。你可以为相同的连接名称创建多个Online-Session。根据实例化的一个IPadtOnlineSession接口，联机服务器将自动登陆到资源。当你释放这个接口时，联机服务器将自动注销。

警告：调用一些Online-Session函数可能会直接修改连接的实时系统的性能。系统可能会减速、停止、启动或改变行为。请确认你的应用程序不会对系统或人员带来危险。

大多数联想服务器函数异步地运行。如：它们根据调用立即返回并且应用在任务队列上。这个任务队列在分离的线程中运行。如果函数运行成功，将不会告诉你这个异步Online-Session函数的返回值。因此，你永远不知道一个任务是否已经执行或者如果执行成功，是否根据任务的完成有一个不同的方式来通告用户。

大多数的Online-Session函数（异步运行的）有一个参数INotify_p。INotify_p=false禁止用户通知，true允许。当允许用户通知，联机服务器通过函数IPadtOnlineSession::SetNotifyWindow发送一条消息PADT_ONLINE_SERVER_NOTIFY_TASK_DONE给窗口寄存器。你可以捕捉这条消息。它在PadtOnlineServerNotifyID.h中被定义。wParam告诉你任务进程（在lzstypes.h tLzsCmdLst中定义）的CommandID，IParam表示结果（在lzstypes.h tLzsErrorCode中定义）。因此你会发现在大多数情况下用户通知都是废弃的。

注：用户通知目前只在窗口消息中可用。如：它不适用于通知一个VisualBasic用户或使用DCOM服务的用户。VB用户和DCOM用户需要建立它们自己的查询时间，因此它们不能依靠通知。

通过OLE连接指针的用户应用程序可能会成为以后更高版本联机服务器的主题。

对于Online-Session函数的更多细节，见接口章节。

5.11.3 变量状态

变量状态用于显示在资源上的变量的当前值。变量状态由接口IPadtOnlineVariableStatusSession的一个实例实现，这个接口可通过下列online-session函数获得：

IPadtOnlineSession ::GetVariableStatusSession(long lReserved_p, LPDISPATCH *pSession_p)

lReserved必须为零，pSession_p返回需要的接口实例。你可以为一个online-session创建多个变量状态会话。

警告：调用一些变量状态会话函数可能会直接修改连接的实时系统的性能。系统可能会减速或改变行为。请确认你的应用程序不会对系统或人员带来危险。

像IPadtOnlineSession一样，IPadtVariableStatusSession接口允许为通知设置一个用户窗口，这需要函数IPadtVariableStatusSession::SetNotifiyWindow。当新的状态值是可用的时候，联机服务器将通过SetNotifyWindow发送一条通知PADT_ONLINE_SERVER_NOTIFY_NEW_STATUS_DATA（在PadtOnlineServerNotifyID.h中定义）给窗口寄存器。这允许你的客户端同步即将到来的数据，但是你也可以用你自己的周期查询数据。

为了获得联机状态值你需要告诉联机服务器哪个变量需要被注意。IPadtVariableStatusSession::SetRange(BSTR strRange_p,BSTR * strRetVal_p)被用于这个功能。strRange_p包含了一个你想观察的以换行符分隔的变量实例路径列表。strRetVal_p以一个以换行符分隔的列表，返回请求的变量的数据类型。

请求字符串的语法如下所示：

<variableinstancepath1> \n <variableinstancepath2> ...

例:

```
PROGRAM PROG1
VAR
A : INT := 123;
B : SINT := 45;
END_VAR
...
```

请求范围:

PROG1.A\nPROG1.B

返回的数据类型:

INT\nSINT

现在你可使用下列函数查询联机数据：

IPadtOnlineVariableStatusSession::GetValues(BSTR *strValues_p)

strValues_p包含了请求变量的值，这正是它们已经用SetRange设置的相同的命令。在strValues_p中的值由换行符分隔。此外，一个值像时间标记一样可有0个、一个或多个参数。这些参数不被完全定义且内部地使用。你必须解析返回值并且去除这些参数。这些参数由PadtOnlineServerNotifyID.h中定义的一个定界符PADT_ONL_STATUS_PARAMETER_DELIMITER_*分隔。

字符串值的语法如下所示：

<Value1> [\x08 <Parameter1> \x08 <Parameter2>...] \n

<Value2> [\x08 <Parameter1> \x08 <Parameter2>...] \n

如：返回值：

123\x0800000000\n45\x0800000000

改变一个变量的值由IPadtOnlineVariableStatusSession::SetVariable或IPadtOnlineVariableStatusSession::ForceVariable完成。

IPadtOnlineVariableStatusSession::SetFormat(long lFormat_p,long *lRetVal_p)用于改变恢复变量值的格式。缺省情况下这些值是小数格式，但是你可以通过使用联机服务器的类型库中的枚举定义tPADT_StatusValueFormat来改变这个格式。

```
enum tPADT_StatusValueFormat
{
    kPADT_StatusValueFormatDec = 0,
    kPADT_StatusValueFormatHex = 1,
    kPADT_StatusValueFormatBCD = 2, // NOT SUPPORTED YET
    kPADT_StatusValueFormatBin = 3,
    kPADT_StatusValueFormatOct = 4,
};
```

对于变量状态会话函数的更多细节，见接口章节。

5.11.4 累加器状态（功率流）

与变量状态不同，累加器状态不观察单个变量，它在一个周期内观察累加器的当前状态。累加器状态范围是一个指令列表代码行的范围。

你可通过以下函数获得累加器状态接口：

IPadtOnlineSession::GetAkkuStatusSession(long lReserved_p, BSTR strNodePath_p, LPDISPATCH *pSession_p)

lReserved_p,必须为0，strNodePath_p选择你想要的累加器状态的接口。pSession_p是被请求的接口。

注：虽然你可以创建多个累加器状态会话的接口，但是在同一时间内只允许有一个累加器状态会话。这是因为实时系统没有被设计成可运行多于1个的累加器状态会话。

为了设置请求范围，可使用

IPadtOnlineAkkuStatusSession::SetRange(long lStartLine_p, long lEndLine_p,BSTR * strRetVal_p)

lStartLine_p和lEndLine_p定义了请求的指定列表行数量。strRetVal_p在目前是不用的。像变量状态会话一样，你可以定义一个通知窗口来接收新数据的消息，见变量状态章节。

查询联机数据和设置格式与变量状态的方法一样，使用函数IPadtOnlineAkkuStatusSession::GetRange和IPadtOnlineAkkuStatusSession::SetFormat。更多细节曲子变量状态章节。

警告：调用一些变量状态会话函数可能会直接修改连接的实时系统的性能。系统可能会减速（特别是当使用本地代码时），或改变行为。请确认你的应用程序不会对系统或人员带来危险。

对于累加器状态会话会话函数的更多细节，见接口章节。

5.11.5 接口

定义：

Released　函数对于OEM使用是开放的。

Sub. Ch.　函数仍然在开发，在以后的版本中其性能可能会改变，请不要使用。

Internal Use　函数仅被OpenPCS内部地使用，请不要使用。

Only with PCD　函数仅当PCD文件路径被设置在GetOnlineSession上时工作。

5.11.5.1 IPadtOnlineServer

Name	Async/ Sync	Only with PCD	Description	Released / Subject to change / Internal use
ConnectionSetupDialog	Sync		打开一个允许安装连接和驱动程序的无模式的会话	released
CreateConnectionEntry	Sync		在连接数据库中创建一个连接名（设置驱动）	Sub. Ch.
GetConnectionEntry	Sync		为一个连接名获得一个接口（设置驱动）	Sub. Ch.
GetFirstConnectionName	Sync		枚举函数用来检索连接数据库中的第一个连接。如果数据库中没有连接将返回空字符串。	released
GetNextConnectionName	Sync		枚举函数用来检索连接数据库中的下一个连接，调用前需要调用GetFirstConnectionName函数。如果没有更多其他的，返回空字符串。	released
GetOnlineSession	Sync		登录到一个PLC并且创建一个在线会话接口	released
Init	Sync		由一个到连接数据库的路径初始化这个联机服务器（必需在调用其它任何函数前调用）	released

5.11.5.2 IPadtOnlineSession

Name	Async / Sync	Only with PCD	Description	Released / Subject to change / Internal use
ColdStartPLC	Async		在已连接的PLC上执行冷启动	released
CustomCommand	Async		在已连接的PLC上执行一条用户命令。这个行为取决于实时系统的实现。	Sub. Ch.
CustomDriverCommand	Sync		在驱动器中执行一条命令。这个行为取决于驱动器的实现。	Sub. Ch.
DisplayPLCInfo	Async		从已连接的PLC中检索PLC版本信息且显示一个无模式对话框。	released
DisplayResourceInfo	Async		从已连接的PLC中检索资源版本信息且显示一个无模式对话框。	released

DownloadFile	Async		下载一个文件到已连接的实时系统上。	Sub. Ch.Internal use
DownloadResource	Async	X	下载一个资源到已连接的PCL上。这个PLC被设置为停止模式。	Released
DownloadTypeInfo	Async	X	下载类型信息到已连接的PLC上，这些信息在没有PLC可用时可以被使用。	Internal use Sub. Ch.
DownloadUserSegment	Async	X	下载一个简单的段到已连接的PLC上，用于在线转换。	Internal use Sub. Ch.
GetAkkuStatusSession	Sync	X	创建一个累加器状态会话。	released
GetPLCInfo	Sync		从已连接的PCL中检索版本信息并以字符串的形式返回。	released
GetPLCStatus	Sync		获得已连接的PCL的状态标志。可以是任何一个tPADT_PLCStatusFlags标志的组合。也可见SetRequestDataFlag。	released
GetResourceInfo	Sync		从已连接的资源中检索版本信息并以字符串的形式返回。	released
GetTaskContainerNode	Sync	X	从代码库中检索任务窗口节点。	Internal useSub. Ch.
GetVariableStatusSession	Sync	X	创建一个变量状态会话。	released
HotStartPLC	Async		在已连接的PLC上执行热启动。	released
IsOnline	Sync		检查连接是否仍是有效的。	released
SetNotifyWindow	Sync		设置一个窗口操作来检索通知。	released
SetRequestDataFlag	Sync		这个设置影响GetPLCStatus函数的行为。如果打开，那么Statusflags将一直被请求。如果关闭，函数将返回一个老的设置。将状态请求打开，会增加网络负载。因此它将在请求累加器或变量状态时被暗中地打开。	released
StopPLC	Async		停止已连接的PLC。	released
UploadFile	Async		上传一个预先由DownloadFile下载的文件	Internal use Sub. Ch.
WarmStart	Async		在已连接的PCL上执行热启动	released

5.11.5.3 IPadtOnlineVariableStatusSession

Name	Async / Sync	Only with PCD	Description	Released /Subject to change / Internal use
ForceVariable	Async	X	在已连接的PCL上设置一个变量值并且强制在每个周期入口保持它的值。	released
GetValues	Sync	X	检索已连接的PLC中的值。	released
IsOnline	Sync	X	检查连接是否仍是有效的。	released
SetFormat	Sync	X	选择检索值的格式。见tPADT_StatusValueFormat。	released
SetNotifyWindow	Sync	X	设置一个窗口操作来检索通知。	released
SetRange	Async	X	设置被观察的变量的范围。	released
SetVariable	Async	X	在已连接的PCL上设置一个变量的值。	released
RemoveAllForced	Async	X	移除所有加在变量上的强制。	released

5.11.5.4 IPadtOnlineAkkuStatusSession

Name	Async /Sync	Only with PCD	Description	Released / Subject to change / Internal use
GetValues	Sync	X	检索已连接的PLC的值。	released
IsOnline	Sync	X	检查连接是否仍然有效。	released
SetFormat	Sync	X	选择检索值的格式。	Sub. Ch.
SetNotifyWindow	Sync	X	设置一个窗口操作来检索通知。	released
SetRange	Async	X	设置被观察的指令表的行数的范围。	released

5.11.6 示例

5.11.6.1 VisualBasic

VisualBasic举例说明了联机服务。这些示例已经设计完成并在HTML页中运行，但是它们可能在任何地方被恢复运行。

```
...
<span id="AkkuStatusValues">
...
</span>
<script language="VBScript">
Dim OnlineSession
Dim OnlineServer
Dim AkkuSession
Dim VariableSession

Set OnlineSession = Nothing
Set OnlineServer = Nothing
Set AkkuSession = Nothing

Sub GoOnline
Set OnlineServer = CreateObject("infoteam.PadtOnlineServer")
OnlineServer.Init("PadtOnlineServer.con")
Set OnlineSession = OnlineServer.GetOnlineSession("SmartSIM32","d:\project1\test\test.pcd","",True)
End Sub

Sub GoOffline
Set OnlineSession = Nothing
Set OnlineServer = Nothing
End Sub

Sub StopPLC
If Not OnlineSession Is Nothing Then
OnlineSession.StopPLC(0)
End If
End Sub

Sub ColdStartPLC
If Not OnlineSession Is Nothing Then
OnlineSession. ColdStartPLC(0)
End If
End Sub

Sub DisplayResourceInfo
If Not OnlineSession Is Nothing Then
OnlineSession.DisplayResourceInfo(0)
End If
End Sub

Sub DisplayPLCInfo
If Not OnlineSession Is Nothing Then
OnlineSession.DisplayPLCInfo(0)
End If
End Sub

Sub StartVariableStatus
If Not OnlineSession Is Nothing Then
```

```
Dim VariableSession
Set VariableSession = OnlineSession.GetVariableStatusSession(0)
VariableSession.SetRange("PROG1.A" + vbLf + "PROG1.A")
End If
End Sub

Sub StopVariableStatus
Set VariableSession = Nothing
End Sub

Sub DisplayVariableStatus
If Not VariableSession Is Nothing Then
InputBox VariableSession.GetValues
End If
End Sub

Sub StartAkkuStatus
If Not OnlineSession Is Nothing Then
Set AkkuSession = OnlineSession.GetAkkuStatusSession(0,"PROG1")
AkkuSession.SetRange 1,24
window.setTimeout "UpdateAkkuStatus()",1000
End If
End Sub

Sub StopAkkuStatus
Set AkkuSession = Nothing
End Sub

Sub SetAkkuFormat (Format)
If Not AkkuSession Is Nothing Then
AkkuSession.SetFormat(Format)
End If
End Sub

Sub UpdateAkkuStatus
If Not AkkuSession Is Nothing Then
AkkuStatusValues.innerHTML = "<pre> " + AkkuSession.GetValues + "</pre>"
window.setTimeout "UpdateAkkuStatus()",300
End If
End Sub

</script>
...
```

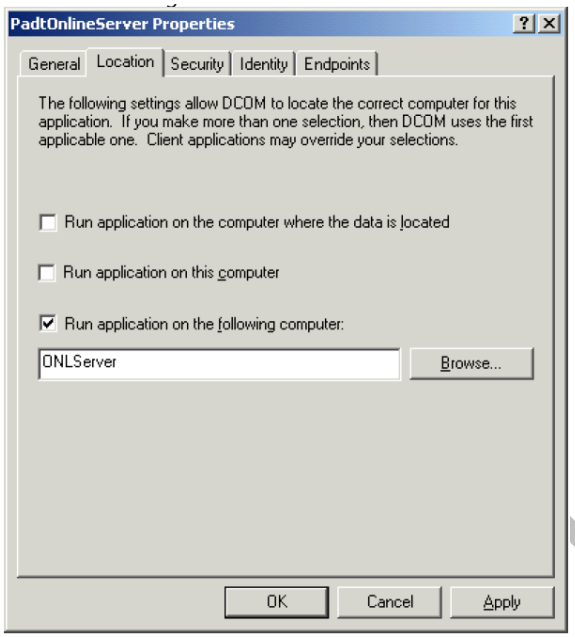
5.11.6.2 Java

Infoteam为Java在OpenPCS联机服务上的应用提供了一个接口，叫做JeditorCon.dll，它随着OpenPCS一起被安装。目录infoteam\vxxx\OpenPCS\JavaInterface下包含了一个展示这个模块使用的示例。更多信息见readme.txt的内容。

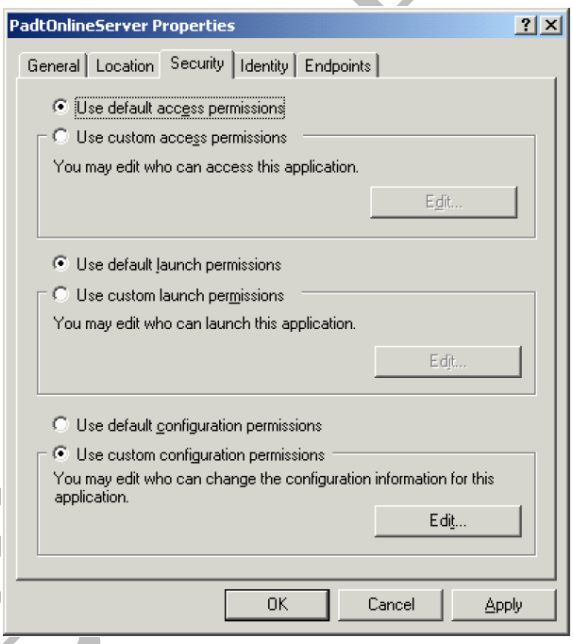
5.11.7 通过 DCOM 使用联机服务

5.11.7.1 准备工作

为了通过DCOM使用联机服务需要一些准备工作。首先联机服务器需要被注册为客户端计算机。这可以在OpenPCS环境中由启动联机服务器完成。然后DCOM需要被配置。调用dcomcnfg.exe，在窗口中安装。在叫做“Applications”的选项卡中找到PadtOnlineServer，选择它并且进入属性。在这联机服务器应该被启动来替换本地联机服务器的启动的地方，你可以选择这个服务器计算机。



安全设置需要被适当地设置。



5.11.7.2 使用联机服务器

当初始化联机服务器的时候你需要以一个参数的形式获得到连接数据库的路径。这个路径必须是在服务器计算机上的路径。联机会话需要一个到这个数据库的连接。

5.11.7.3 必要条件和约束

为了对代码资源库进行存取它必须在连接数据库中被分配连接。这通过下载程序来完成。联机服务器不可以通过DCOM通知客户端新数据。因此客户端必须通过调用GetValues查询新数据。下面的列表表明了一个在VBScript中获得联机会话的例子。

```
Set OnlineServer = CreateObject("infoteam.PadtOnlineServer")
OnlineServer.Init("C:\Documents and Settings\All Users\Application Data\infoteam
Software\OpenPCS2008\Openpcs.520\Onlsvr.con")
Set OnlineSession = OnlineServer.GetOnlineSession("Simulation","","",True)
```

5.10.8 联机服务 OEM 命令

OEM用户可以发送OEM命令给实时系统。

LZSBYTE LzsEnvOEMCommand(LZSINT iCmd_p, LZSDWORD dwValue_p)

如果联机服务器发送了一条OEM命令LzsEnvOEMCommand将被调用。参数iCmd是OEM定义的命令的序号，dwValue_p是一个4字节的命令值。

5.12 TUI-控制

5.12.1 基本要素

TUI-控制是一个ActiveX控制，用于显示和从一个PLC修改联机变量值。它由TUI32.EXE的大多数功能组成。它显示实例路径、名称、取值和请求值的数据类型。这个控制使用一个由联机服务器提供的类型为IPadtVariableStatusSession的实例。

5.12.1.1 架构

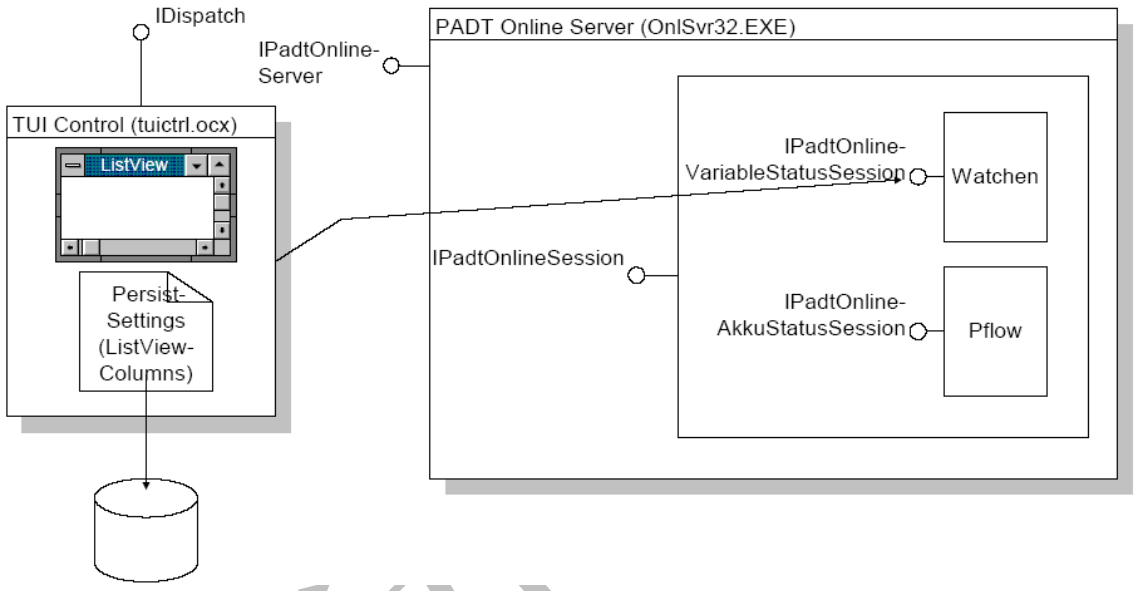


图97：TUI-控制架构

T&C-控制包括一个MS-ListView。这些设置像列，宽度和位置一样在控制IPersist接口中被调整成序列。为了控制运行，它必须由一个类型为IPadtVariableStatusSession的接口初始化。这个控制可以使用来自联机服务器的客户通知消息来同步来自PLC的新数据，或者它可以随着一个不变的查询时间周期被初始化。当一个DCOM连接溢出时这是特别有用的，这里窗口信息不能被捕捉。

注意：通过OLE Connection-Points的客户端可能会成为以后版本联机服务器的主题。

更多细节见“联机服务器”。

5.12.2 用途

为了使用T&C-控制你需要在你的客户端创建一个实例。这个实例通常由函数CreateControl完成。在VisualBasic或C++ Visual Studio中你可以插入控制类型库tuictrl.tlb。

在使用任何控制函数前你必须用一个VariableStatusSession接口对这些控制进行初始化，这个接口可从联机服务器中获得。由下列函数设置这个接口：

SetVariableStatusSession(IDispatch* pVariableSession_p,long lPollTime_p);

pVariableSession_p是一个指向IPadtOnlineVariableStatusSession的指针，lPollTime_p为控制分配了一个查询时间。如：新状态数据请求之前的时间周期。如果你将这个参数设置为0，控制将使用客户端通知来同步联机服务器。

注意：客户端通知只在相同的机器上运行，如果你使用DCOM，客户端通知将不能运行并且你必须为lPollTime_p设置一个值。

现在你可以使用函数SetWatchEntries(BSTR pszNodePathes_p)或AddWatchEntries(BSTR pszNodePathes_p)分配将要被观察的变量实例。

两个函数根据下列语法获得一个变量实例路径的列表：

<variableinst1> \n <variableinst1> ...

如：

PROG1.A\nPROG1.B

RemoveWatchEntries(BSTR pszNodePathes_p)可使用相同的语言用于从列表中移除变量实例。

对于控制函数的更多细节，见接口章节。

5.12.3 控制接口

定义：

Released　　函数对于OEM使用是开放的。

Sub. Ch.　　函数仍然在开发，在以后的版本中其性能可能会改变，请不要使用。

Internal Use　函数仅被OpenPCS内部地使用，请不要使用。

Name	Async/Sync	Description	Released /Subject to change / Internal use
AboutBox	Sync	显示一个AboutBox	released
AddWatchEntries	Sync	加入要观察的变量实例	released
EnableForceVariable	Sync	如果ForceVariable对已选择的项目是可用的时为真	released
EnableMoveDown	Sync	如果MoveDown对已选择的项目是可用的时为真	released
EnableMoveUp	Sync	如果MoveUp对已选择的项目是可用的时为真	released
EnableRemoveWatchEntry	Sync	如果RemoveWatchEntry对已选择的项目是可用的时为真	released
EnableSetVariable	Sync	如果SetVariable对已选择的项目是可用的时为真	released
ForceVariable	Async	在PLC上强制已选择的项目的值	released
GetFormat	Sync	获得这些值的格式。见tPADT_StatusValueFormat	released
GetGrid	Sync	如果网格被打开则为真，否则为假	released
GetRange	Sync	获得一个当前观察的变量实例的清单。语法与SetWatchEntries完全相同。	released
MoveDown	Sync	将选择的项目下移	released
MoveUp	Sync	将选择的项目上移	released
ReloadWatchEntries	Sync	向联机服务器提交一条新的请求来重装所有的变量实例。如果资源同时改变了，将更新T$C控制的变量类型	released
RemoveAllWatchEntries	Sync	从ListView中移除所有的变量实例	released
RemoveWatchEntries	Sync	从ListView中移除指定的变量实例	released
RemoveWatchEntry	Sync	从ListView中移除已选择的项目	released
SetFormat	Sync	设置变量值的格式。见tPADT_StatusValueFormat	released
SetGrid	Sync	打开网格时为真，关闭时为假	released
SetVariable	Async	在PLC上设置被选择的项目的值	released
SetVariableStatusSession	Sync	由一个VariableStatusSession初始化控制	released
SetWatchEntries	Sync	设置要观察的变量实例，所有预先设置的实例将被清除	released

```
enum tPADT_StatusValueFormat
{
    kPADT_StatusValueFormatDec = 0, kPADT_StatusValueFormatHex = 1, kPADT_StatusValueFormatBCD = 2, // NOT
    SUPPORTED YET
    kPADT_StatusValueFormatBin = 3,
    kPADT_StatusValueFormatOct = 4,
};
```

5.12.4 示例

VisualBasic举例说明了联机服务器和T&C-控制。这些示例已经设计完成并在HTML页中运行，但是它们可能在任何地方被恢复运行。

```
<object
classid="clsid:1B96AE65-20C6-11D4-BA12-00902710FBBD" width=500 height=300 ID="TUI_CTRL" >
</object>
<script language="VBScript">
Sub StartVariableStatus
Dim OnlineServer
Set OnlineServer = CreateObject("infoteam.PadtOnlineServer") OnlineServer.Init("PadtOnlineServer.con")
Set OnlineSession = OnlineServer.GetOnlineSession("SmartSIM32","d:\project1\test\test.pcd ","",True)
Dim VSession Set VSession = OnlineSession.GetVariableStatusSession(0) TUI_CTRL.SetVariableStatusSession
VSession,0
TUI_CTRL.SetWatchEntries("PROG1.A" + vbLf + "PROG1.B")
End Sub
</script>
```

5.13 历史数据

历史数据是另一个从实时系统中收集和检索变量值的途径。

实时系统为需要被考虑的历史数据维持一个变量表。每个表的入口有一个缓冲区，这个缓冲区被分配来存储多个变量值/时间戳。

历史数据缓冲区的安装和在这些缓冲区中的变量值的存储，和放在一起的查询结果一样，可以被从属的应用程序处理。

对历史数据的变量的列表的修改是被保持的，和通过联机服务器的COM接口完成分配的历史数据的检索一样。

5.13.1 服务器接口

为了为历史数据收集标记变量和查询这些历史数据值，一个类型为IPadtOnlineHistStatusSession的联机服务器接口必须被恢复。联机服务器接口IPadtOnlineSession有一个方法GetHistStatusSession实现那个用途。

下列内容描述了IPadtOnlineHistStatusSession的方法。这些方法可用于在实时系统中对历史数据表增加或移除变量和查询历史数据的值。

所有方法队了变量列表的格式如下所示：

<variable instance path 1>\n<variable instance path 2>\n

例如：“main.counter\nglobal.axis\n”

同样，所有的方法通过plErrorCode_p参数返回错误代码。在PadtOnlineServer.idl中的实际错误代码为enum tPADT_Hist_ResultType。

AddRange(BSTR strListOfVarPaths_p, LONG* pnErrorCode_p);
增加一个变量的列表到历史数据表中。 如果给出的变量中有一个不能在代码库中被分解，将不会添加任何变量到这个表中，并且将返回一条错误。
DeleteRange(BSTR strListOfVarPaths_p, LONG* pnErrorCode_p);
从历史数据表中删除一个变量列表。 所有在这个表中出现的变量都将被删除。如果其中有一个变量不在这个表中将返回一个错误，但这个变量表仍会被删除。

ClearRange(LONG* plErrorCode_p);
删除历史数据表中的所有变量。
GetRange([in] LONG lQueryId_p, [in] BSTR strListOfVarPaths_p, [out,retval] LONG* plErrorCode_p);
为给出的变量列表请求历史数据缓冲区。如果有一个给出的变量不是历史数据变量表的一部分，将不会返回错误。作为替代，它通过由GetData()返回的数据处理。 一个查询通过查询ID识别。在这种方式下，大多数查询都可以成功开始。 客户端通过一条窗口消息获知已完成的查询： PADT_ONLINE_SERVER_NOTIFY_HIST_QUERY_FINISHED (see PadtOnlineServerNotifyID.h) 这条消息的参数有：WPARAM: BOOL fResult, LPARAM: LONG nQueryId. 如果查询成功地执行了，客户端通过调用GetData(..)检索结果。
GetData(LONG lQueryId_p, [out] VARIANT* pData_p, [out,retval] LONG* plErrorCode_p);
GetData返回由传递的查询ID分配的一个成功的查询的结果。数据通过参数VARIANT pData_p 以一个原始字节缓冲区的形式返回。如： 一个类型为(VT_UI1 | VT_ARRAY)的SAFEARRAY。客户端删除这些数据的操作是可靠的。存储在缓冲区中的这些数据的布局依赖于应用程序（见LzsEnvHistGetRange below）. 如果没有传递的查询ID的结果是有效的，GetData将失败。 重要提示： 在客户端通过窗口消息获知一次成功的查询后，它无论如何都将检索这个结果，即使它不使用这个结果。联机服务器不知道何时删除结果数据缓冲区，所以它将保持这些数据直到它关闭。这种方式导致了很大的内存消耗。

5.13.2 实时系统

本节描述了怎样为历史数据加入支持到实时系统中。

实时系统为需要被维持的历史数据维持一个变量表。每次一个新变量被加入到这个表中，必须创建一个为了收集这个变量值的数据缓冲区。如果从这个表中删除了一个变量，这个缓冲区也必须被相应地释放。在这些缓冲区中变量的存储只在一个周期完成后执行一次。如果历史数据的一个或多个变量被请求，那么来自分配的缓冲区的数据将被收集并且发送回联机服务器。

实现分为两个部分：

系统：包括所有对插入/删除/清除/请求的通信处理和列表自己的更新（见lzshist.c）。

用户：创建/解构数据缓冲区，在周期结束收集数据，请求的结果的汇编（见下面的LzsEnvHist*函数）。

为了包含系统部分到你的实时系统，加入文件lzshist.c到你的工程中并且通过在config.h中的定义_LZS_HISTDATA_来使能它。

文件oem\LzsEnvHist.c有一个对实现历史数据支持的OEM指定用户部分的纲要。Win32IPC示例的LzsEnvHist.c实现可以作为一个开始指针使用。

历史数据表为需要收集的历史数据处理一个变量列表。它在lszhist.h中被一个叫做tHistElem的结构体中的一个固定大小的数组histElems_g[LZS_HISTMAX] (LZS_MAXHIST被定义在config.h中)定义。tHistElem可按你的需要移植。它的pVar成员保持一个直接指向变量值的指针。如果这个指针是非空，那么表示入口正在使用。所有在pVar后的成员（像pHist,pRead,pWrite）都是与应用相关的。系统部分不使用它们。例如，它们可用于为存储每个变量的历史数据安装一个环形缓冲区。所有对这些缓冲区的存取都通过修改过的用户部分的下列函数来处理：

LZSBYTE LzsEnvHistInitialize(void)
初始化函数，在实时系统启动后调用一次。
LZSBYTE LzsEnvHistInitElemBuffer(LZSWORD wID_p)
为收集表记录histElems_g[wID_p]的历史数据创建一个缓冲区。
voidLzsEnvHistFreeElemBuffer(LZSWORD wID_p)
释放由LzsEnvHistInitElemBuffer()为记录histElems_g [wID_p]创建的缓冲区。

voidLzsEnvHistRecord(LZSWORD wPgmIndex_p)
在周期结束时调用。 这个函数存储当前变量值到分配的缓冲区中。为此它将依次通过历史数据变量表(histElems_garray)。 在一个任务完成一个周期后，只有这个任务的变量值与任务全局变量值一样会被改变。 说明只有那些变量值需要被存储到立等可取数据缓冲区中。 wPgmIndex_p给出了刚完成的任务的索引。
histElems_g[i].wPgmIndex是任务的索引，变量被声明在其中。如果它为0，则变量被全局声明。 除变量值外这个函数还可以同时存储这个值的时间戳。
LZSBYTE LzsEnvHistGetRange(LZSWORD wCount, tPlcMemPtr pListOfIDs, tPlcMemPtr *pResult_p, LZSDWORD *pdwResultSize_p, void(**ppFuncFreeResult_p)(tPlcMemPtr pResult_p, LZSDWORD dwSize_p))
处理来自联机服务器的查询。 pListOfIDs是大小为wCount 的LZSWORD listIDs[](histElems_g[wID])。每个记录是一个在立等可取数据表中的变量的索引。 这个函数被期望收集所有请求的变量历史数据到一个缓冲区中并且返回这个缓冲区。系统将传递这个缓冲区回联机服务器。 pResult_p: 结果缓冲区指针 pdwResultSize_p:结果缓冲区大小 ppFuncFreeResult_p:指向一个析构函数的指针。这个函数在缓冲区被传递后调用一次。然后这个函数可以释放这个缓冲区。这是可选的。如果不需要它只用传递LZSNULL给ppFuncFreeResult_p. 如果有一个请求的变量不能被联机服务器分解，它将发送一个为0xFFFF的索引来表明。这被用来返回错误值给调用的应用程序。

由于来自IpMainLoop的LzsEnvHistRecord在周期结束时被调用并且大多数其它的历史数据函数在命令接收上都从IpCmdMainLoop调用，所以必须注意IpMainLoop和IpCmdMainLoop是否运行在不同的线程。在这种情况下，变量表histElems_g必须被一些互斥量保护起来。

加入锁定是很简单的。互斥量可以被LzsEnvHistInitialize()安装。锁定和解锁通过函数LzsEnvHistMutexLock()和LzsEnvHistMutexUnlock()完成。_LZS_HISTDATA_LOCKING_必须在config.h中被定义，以便系统函数使用互斥量锁定/解锁函数。

5.14 联机编辑

有了联机编辑，当程序正在被执行时，由用户制造的改变就可以适应于PLC。

如果用户在一个OpenPCS内的编译器上制造了一些改变并且为那个编辑器选择了一个监听，联机编辑将被执行：

一个递增的创建将运行。联机服务器异步地下载程序到PLC上。PLC不会停止。下载完成后，在PLC中存在两个程序，新的一个和当前正在执行的一个。

在下一个周期结束后，当前程序被注销并且新程序取代当前程序。

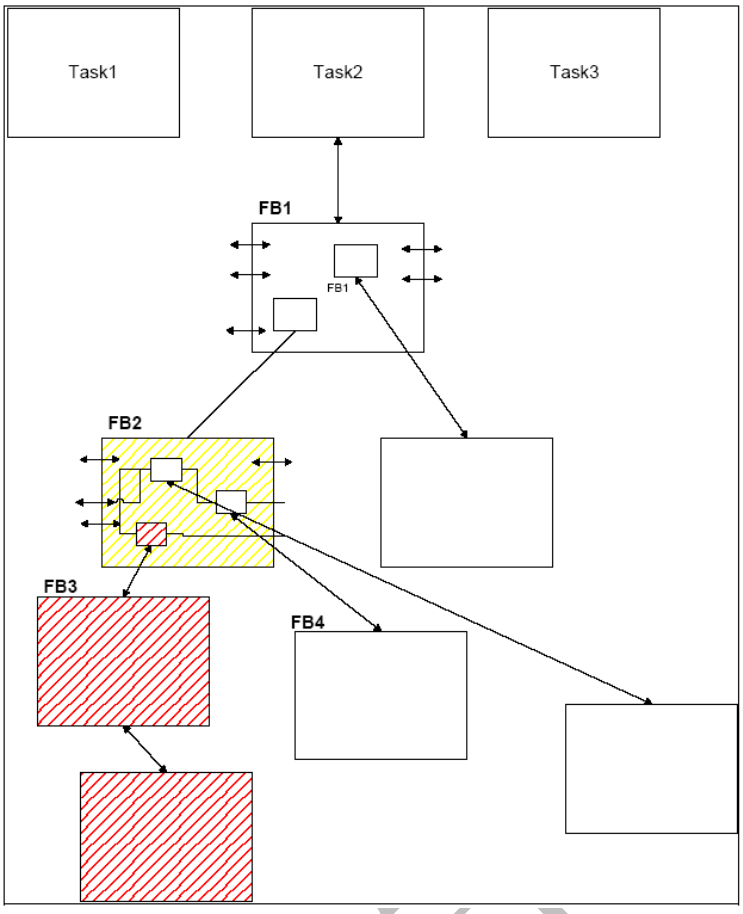
新程序继续执行。

由于下载一个完全的程序会需要一点时间，下载已经是最优化的，所以只有被改变的段会下载到PLC。没有改变的段将会从旧程序连接到新程序上。这对代码段，本地代码段，初始数据段和数据段都有效。

再利用数据段意味着，POU实例没有被改变，或更准确地说，它的初始数据段没有被改变，当执行联机编辑时将保持它们的值。

换句话说，如果只有POU_1的代码被改变，POU_1实例的变量没有一个将会被恢复到它们的默认值。如果POU_2的变量申明部分改变了（如：初始数据段），所有POU_2实例的变量值都将被恢复。在其它POU中的变量数据将不会受影响。

下图显示了一个典型的情况：



如果一个新的函数块FB3被加入到已经存在的函数块FB2中，那么FB2的变量数据将被恢复到它的初始值。FB3和它的子函数块是新创建的，因此它们也将被设置为它们的初始值。但是在其它任务中的FB1、FB4和POU不会受这个改变影响（当然它们和FB2不是一样的POU）。它们将保持它们的值。

在OpenPCS和目标系统5.2.2以上版本中在线编辑都是可用的。

在目标系统6.1.0版本以上，支持下述在线编辑的扩展：在数据段中的被改变的变量将在在线编辑时保持它们的值。为了保持它们的值，这些变量必须在变量表段中配置（见5.3.7.7.2节）并且它们应保持相同的名称、范围和类型。

对于在系统中不同部分使能支持在线编辑，见5.14.1和5.14.2。

5.14.1 硬件描述

在硬件说明文件的[Module]部分，必须提出一个OnlineLinking=1的记录。

对于“SmartVarTab”功能的使用，必须提出SmartVarTab=1的入记录，“SmartVarTab”功能的意思是在数据段中被改变的变量在在线修改时将保持它们的值。

5.14.2 实时系统

实时系统需要异步下载和新系统交换以及当前程序在线编辑的工作的支持。

为了允许这个支持，文件lzsdwlws.c需要被加入到工程中并且_LZS_DWLWITHOUTSTOP_必须在文件config.h中被定义。

请注意当异步下载发生时，需要足够的内存空间来存储两个程序。不过在实践中，如果只申请了很小的改变，那么大多数的段可被重复利用，因此内存消耗会少很多。

在其它功能上的影响：

在线编辑不能和ByRef最优化（_LZS_DIRECT_BYREF_）同时使用。

5.14.3 在线服务器

对于在线编辑支持，在线服务器的IPadtOnlineSession接口已经被下列方法扩展：

HRESULT DownloadResourceWithoutStop([in] long lNotify_p, [in] BOOL fIncremental_p, [out,retval] long * plRetVal_p)

这个方法将异步地下载程序于PLC。Flncremental_p是绝对的。通过传递TRUE。

本地代码含义

同OpenPCS一起使用本地代码编译程序在在线性能和运行性能上有一些不明显的含义。这将在本节中进行介绍。

影响这些性能的因素有：

在相应硬件说明文件（如：SMARTSIM.INI）中定义的目标性能、也就是PCodeDLL入口和在线连接

本地代码编译程序支持的功能（即“直接调用”）

在资源定义中的最优化设置（容量、速度、普通模式）

在目标中的实现

影响因素有：

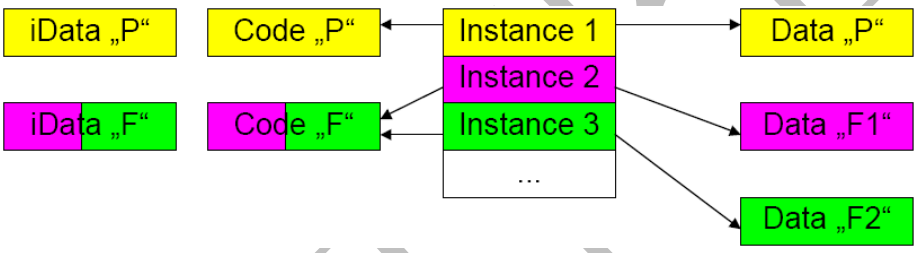
执行速度

任务转换延迟

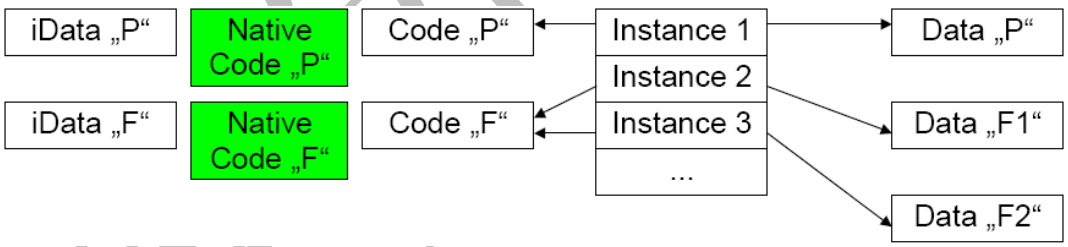
在线功能的实用性（如：程序状态显示、断点、在线修改、异步下载）

5.14.4 一般架构

对于单独的UCODE，为每个函数块（和程序）类型下载一个代码段进目标，见下图：



对于本地代码，下载“本地代码段”被随机地加入到每个UCODE代码段的旁边，见下图：



请注意：不管最优化的设置，UCODE段通常在当前约束实现期间被下载到目标，尽管它从未使用过。本地代码段通常比UCODE段大，但是像其它段一样最大限制为64k。

5.14.5 本地代码详述

5.14.5.1 混合 UCODE 和本地代码

OpenPCS支持在一个应用程序中混合UCODE和本地代码。每次从一个函数块的调用/返回都（一般地）不直接执行，但是可通过实时系统来决定是否使用UCODE或本地代码。这样的好处是当开发一个新本地代码编译程序时，它在全部指令设置可以被编译前就已经可以使用。不被支持的代码使用指令将仅仅导致函数块以UCODE的形式被执行的结果。

一些在线功能对UCODE是可用的；实时系统将自动对一个函数块使用UCODE来替代本地代码，这是为了正确地完成这些在线功能。

5.14.5.2 直接调用

通常，在本地代码中的调用和返回通过实时系统执行，见5.14.5。但是这需要时间。大多数本地代码编译程序的以后的版本都支持“直接调用”，这个“直接调用”绕开实时系统并且从一个本地代码段执行调用和返回到另一个本地代码段。UCODE支持的功能对于“直接调用”的应用程序编译是不可用的。

直接调用只在下列情况下使用：

本地代码编译程序支持它（如果是不能肯定，见本地代码编译程序文档），最优化设置为“速度”。

5.14.5.3 最优化设置

OpenPCS支持对“速度”、“容量”和“普通模式”的最优化设置。

“容量”将只生成UCODE而不生成本地代码。本地代码编译程序将不被调用。

“普通模式”将对于最适宜的调试支持随着两个代码的混合的使能来编译UCODE和本地代码。如果本地代码编译程序支持“直接调用”，这些将不被使用。

“速度”将产生一个错误消息，如果应用程序的任一部分不能与本地代码一起编译。如果本地代码编译程序支持直接调用，这些将不使用。

5.14.6 调试只对 UCODE 有效的功能

5.14.6.1 程序状态显示

以下语言显示的程序状态

指令列表

欧梯形图

函数块图

结构化文本

US梯形图

只在UCODE被执行的情况下是可用的，在本地代码上不可用。连续的函数图表语言和断续函数图表语言是不影响的。

5.14.7 实时效果

5.14.7.1 任务转换延迟

SmartPLC实时系统通常允许任务转换在每次UCODE指令调用后触发，如：甚至在小循环中的频繁地完全转换。这在本地代码中是不可能的。如果没有使用直接调用，对于任务转换的实时检查将在每次调用/返回时进行。若使用直接调用，对于任务转换的检查只在一个任务已经完全完成后进行。

5.14.7.2 执行速度

本地代码的执行通常比UCODE快，加速因素取决于处理器，实时系统的实现和本地代码编译程序，C-编译器通常用于编译实时系统、应用程序、内存架构等。一些已知的作用有：

如果没有使用“直接调用”，在约定的转换调用期间，用本地代码调用一个函数块可能会比用UCODE调用慢。因此，如果你的应用程序使用了大量的微功能块，回事会有如下期望。

对于复杂的指令（如：正弦或实数的除法），UCODE的开销将比较小，因此由本地代码获得的回事可能会很小。

5.14.7.3 错误检查

在实时系统中的UCODE虚拟机将在执行UCODE的时候进行一些检查。在本地代码中，大多数这种检查都被省略了，因为它会增加有效的实时系统开销并且使用本地代码的最终原因是速度改进。需要强调的是首次用UCODE测试和调试你的应用程序应在使用本地代码之前。如果你需要一个包含检查的本地代码编译程序，请咨询infoteam。在本地代码中检查是不可以的，包括：

数组下标的上溢/下溢

字符串长度检查

5.15 OpenPCS 的更新

OpenPCS的更新已交付了三个不同的类型：

升级影响版本号的第一个数字并且表示OpenPCS的较多的改进（Version 3->Version 4）。升级只在CD上进行完整版本的交付。在以前OpenPCS版本的许可证上的升级将导致不能工作，如：通常许可证都必须升级。

升级影响版本号的第二个数字（Version 3.4->Version 3.5），通过提供新的功能和问题收集。升级只在CD上

进行完整版本的交付。在以前OpenPCS版本的许可证上的升级将导致不能工作，如：通常许可证都必须升级。

服务包影响版本号的第三个数字（Version 3.4.2->Version 3.4.3）并且通常只提供问题收集，间或较小的新功能。服务包只通过Internet作为增量补丁交付。服务包可在基础版本的许可证上运行，如：不需要更新许可证。

OpenPCS的新版本通常在http://www.infoteam.de上能和并且服务包可在此网站上下载使用。这个网页只被OEM使用并且不对终端用户开放。

为了获得OpenPCS的新版本，询infoteam。OpenPCS的新版本被自动发送到OEM是可以的，请询infoteam获得许可。对于更新的定价，见我们的当前价格列表或询infoteam。

5.15.1 报告问题

你可能会遇到你的OpenPCS版本的问题，使用本文档最后显示的内容（有必要的话将其拷贝）并且传真它到infoteam Software GmbH,Fax.+49-9131-7800-50。使用分离的工作表是必要的。请为每个更改需求提供一张工作表。

这个服务只提供给OEM，如果你希望infoteam为你的终端用户提供支持，请询infoteam。

5.16 OpenPCS 文件

5.16.1 扩展文件

OpenPCS使用下列扩展文件：

Extension	Description
.POE	POU code (instruction list)
.ST	POU code (structured text, only in custom versions)
.P$E	Mix-POU
.AWL	POU code in instruction list, only in custom versions
.DCL	declarations
.ERR	error list
.E$R	re-translated error list
.PTT	Prototype
.CRO	Crossreference List
.INC	includes
.VAR	Project registration file
.CFC	CFC file
.SFC	SFC file
.ICA	ICodedump output
.GDF	GraphWorX draw file
.DOC	Microsoft Word document
.PIV	Process image variables assignment file
.MDB	Microsoft Access database (OpcSvr.mdb used for configuration of the OpenPCS OPC-Server)

注：不是所有的扩展都适应于所有的OpenPCS版本。

5.16.2 可执行文件

在OpenPCS可执行文件的目录下分配的文件有如下用途：

文件	用途	路径
adddrvr.exe	安装额外的OEM的工具	D:\p169util\adddrvr\adddrvr.sln
brwtree.ocx	ActiveX控件树	
canopendrv.dll	CanOpen驱动程序	
cfc.ocx	CFC的ActiveX版本	
CfgPrint.exe	工程配置打印工具	D:\p169util\ConfigPrint\ConfigPrint.sln

文件	用途	路径
Config1.io		
crd32.dll	代码库	D:\P165PCE\CRD\crd32\crd32.sln
cxedit_res.dll	SFC编辑器资源	
cxprojdata.dll	工程数据库（CX编辑器）	
dcfpars.dll	DCF分析程序，用于CANOpen	
dumpdll32.dll	编译器32位转储函数	D:\p169util\dumpdll32\dumpdll32.sln
HwConf32.dll	32位基础硬件配置DLL	
ILC.exe	32位指令表编译器	
ILCRes.dll	ILC.exe资源	
iledit.ocx	指令表编辑器	
instlog.exe	安装程序的登陆	D:\p169util\instlog\instlog.sln
ipcdrv.dll	驱动里程间通信的联机服务器	D:\infoteam\v400\OpenPCS\PadtIPCDriver32\PadtOnlineDriver32.sln
ItLink.exe	32位OpenPCS连接器	
ITMake.exe	32位Make	
itsetup.dll	OpenPCS安装的部分	D:\p169util\itsetup\itsetup.sln
JeditorCon	Java语言的联机服务	D:\P446PadtOnline\JEditorCon\JEditorCon.sln
ldedit.ocx	US梯形图编辑器	
Licchk51.dll	许可证检查（32位版本）	D:\p169lic\licchk51\licchk51.sln
licedt32.exe	许可证钥匙编辑器	D:\p169lic\licedit32\licedit32.sln
LinkRes.dll	32位连接器资源	
ListNcc.exe		D:\p169util\ListNcc\ListNcc.sln
MakeRes.dll	32位编译器资源	
migrate.exe	移植实用程序	D:\p169util\migrate\migrate.sln
Ncc167h_32.dl		D:\P165PCE\Pcode\ncc167H\ncc167h_32.sln
Ncc563_32.dll		D:\P165PCE\Pcode\ncc563\ncc563_32.sln
Ncc68k_32.dll		D:\P165PCE\Pcode\ncc68k\ncc68k_32.sln
ncc86_32.dll	本地代码编译器插件（32位版本）	D:\P165PCE\Pcode\ncc86\ncc86_32.sln
ncc86r_32.dll	实址方式NCC，32位版本	
nch8300h_32.dll		D:\P165PCE\Pcode\ncH8300H\nch8300h_32.sln
onlsvr32.exe	联机服务器	D:\P446PadtOnline\PadtOnlineServer\PadtOnlineServer.sln
onlsvr32.tlb	联机服务器类型库	
OpcBrwsr.ocx	Opc浏览器控件	
OpcMapr.ocx	分配编辑器控件	
OpcPars.dll	Piv文件分析程序	
OpcRegDB.exe		
OPCServer.exe	OPC服务器（4.2.3以上版本）	
OpcSvr.exe	OPC服务器（4.2.4上以版本）	
OpcSrv.mdb	OPC服务器配置文件	
OpenPCS.exe	主可执行程序	
OpenPCS50D.pdf	用户手册	
OT802as.dll		
pack32.dll	32位压缩函数	D:\P165PCE\PACK\PACK32\pack32.sln

文件	用途	路径
padtcrd.dll	联机服务器—crd接口	D:\P165PCE\CRD\PadtCRD32\PadtCRD32.sln
padtprct.dll	CX编辑器的一部分	
padtprop.exe	工程特定打印域配置器	D:\p169util\PadtPrintOptions\PadtPrintOptions.sln
padtrtps.exe	实时持续文件生成器	D:\p169util\padtruntimepersist\padtruntimepersist.sln
padtutils.dll	SFC编辑器辅助类	
pcddump32.exe	编译器转储实用程序（32位版本）	D:\p169util\pcddump32\pcddump32.sln
qvl.exe	交叉引用	D:\p169util\qvl\QVL.MAK
RegisterOpenPCS.bat	OpenPCS二值寄存器的批处理文件	
rs232_35.dll	联机服务器（RS232,3.5目标）	D:\infoteam\v400\OpenPCS\PadtV2435Driver32\PadtOnlineDriver32.sln
rs232drv.dll	联机服务器（RS232）	
rtxipcdrv.dll	RTX通信驱动	
RWUXThemeS.dll		
Setting32.dll	OEM设置	D:\p169set\setting32\setting32.sln
sfceditcontrol.ocx	SFC编辑器的ActiveX控件	
Sfl202as.dll		
SftTv32.dll	树形视图控件（32位版本）	
smartPlc.dll	SmartSIM特定硬件配置	
smartsim.exe	SmartSIM	D:\Smartplc\Platform\PadtSmartSIM32\PadtSmartSIM32.sln
ssm32.dll	32位版本的ssm.ddl	
Stedit.ocx	结构化文本编辑器	
syvared.ocx	声明编辑器	
tcpdrv.dll	联机驱动程序（TCP/IP）	D:\infoteam\v400\OpenPCS\PadtTCPDriver32\PadtOnlineDriver32.sln
Tuictrl.ocx	检验与调试	D:\P446PadtOnline\PadtOnlineTuiControl\PadtOnlineTuiControl.sln
tuioem32.dll	OEM-specific联机功能	D:\P446PadtOnline\TuiOem32\tuioem32.sln
Vartab32.dll	变量表过滤模块	D:\p169util\vartab32\VARTAB32.sln
winlzs32.dll	SmartSIM32实时系统	

5.16.3 注册入口

在OpenPCSVersion5以上版本使用下列注册记录。

Key	Entry	Type	Description	UsedBy	ValueOnSetup	OEM changable
Settings\SFCReadOnlyKey	@	REG_SZ	If set to "1", the chart gets unchangeable, while the code of the elements remains writeable.	sfceditcontrol.ocx	"0"	yes
[HKLM]\Software\infoteam Software GmbH\CurrentVersion	@	REG_SZ	current used version to get subkey for all other registry entries		“5.0”	no

Key	Entry	Type	Description	UsedBy	ValueOnSetup	OEM changable
IniPath	@	REG_SZ	Directory where all OpenPCS configuration files are. This replaces the hardcoded path in setting.dll	setting32.dll	„C:\Documents and Settings\All Users\Application Data\infoteam Software\OpenPCS2008\Openpcs.520“	yes: with own setup
Project\Last Project Path	@	REG_SZ	Directory of the last open project	OpenPCS.exe		
Project\Recent Project List\File0 ...Project\Recent Project List\File7	@	REG_SZ	Directories of the last 8 open projects	OpenPCS.exe		
AppWindow\XPos	@	REG_SZ	last horizontal position of main window	CMainframe		
AppWindow\YPos	@	REG_SZ	last vertical position of main window	CMainframe		
AppWindow\Width	@	REG_SZ	last width of main window	CMainframe		
AppWindow\Height	@	REG_SZ	last height of main window	CMainframe		
BRWTREECTRL\HELPFILE_COUNT	@	REG_SZ	number of help files	BrwTree.ocx	2	
BRWTREECTRL\HELPFILE_NODENAME<nr>	@	REG_SZ	name of help file <nr>, <nr> starts with 1	BrwTree.ocx	nr=1: "CX Editor Help"	
					nr=2: "Infoteam Online"	
BRWTREECTRL\HELPFILE_PATH<nr>	@	REG_SZ	path of help file <nr>, <nr> starts with 1	BrwTree.ocx	nr=1:<openpcsdir>\webhelp\cxedit.htm	
					nr=2:http://www.infoteam.de	
BRWTREECTRL\VISIBLEEXTENSIONS	@	REG_SZ	collection of extension which are visible additionally to openpcs files, separated by ;	BrwTree.ocx	""	
BRWTREECTRL\NEWSTYLEDICONS	@	REG_SZ	flag: show new or old style icons	BrwTree.ocx		
BRWTREECTRL\SHOWONLYKNOWNTYPES	@	REG_SZ	flag: show only openpcs files or not	BrwTree.ocx		
Dialogs\SaveAllBeforeCompile	@	REG_SZ	flag: show dialog to save all or not before compiling; if key left the dialog is always shown	BrwTree.ocx, OpenPCS.exe	"1"	

Key	Entry	Type	Description	UsedBy	ValueOnSetup	OEM changable
BRWTREECTRL\NETWORKPANENAME	@	REG_SZ	name of resourcetreepane, if ist empty, name comes from VS resources	BrwTree.ocx	""	
BRWTREECTRL\FILEADDDIALOGDISABLE	@	REG_SZ	Bitmask for disabling filetypes shown in AddNewFileDialog IL 0x0001 ST 0x0002 POU 0x0004 LDD 0x0008 SFC 0x0010 CFC 0x0020 Projects 0x0040 Directories 0x0080 Resource 0x0100 Type 0x0200 Global 0x0400 Direct 0x0800 Global FBD 0x1000	BrwTree.ocx	0	
BRWTREECTRL\NEWRESOURCENAME	@	REG_SZ	Default name for the new resource in a new project	BrwTree.ocx		
BRWTREECTRL\NEWRESOURCEHARDWARE	@	REG_SZ	Default hardware for a new created resource	BrwTree.ocx		
BRWTREECTRL\NEWRESOURCECONNECTION	@	REG_SZ	Default connection for a new created resource	BrwTree.ocx		
BRWTREECTRL\NEWRESOURCEOPTIMIZE	@	REG_SZ	Default optimization level for a new created resource	BrwTree.ocx		
BRWTREECTRL\NEWRESOURCEPACKSOURCE	@	REG_SZ	Default packsource setting for a new created resource.	BrwTree.ocx		
BRWTREECTRL\DONTSHOWSFCVARS	@	REG_SZ	Flag: hide variables with underscore(SFC variables)	BrwTree.ocx		
BRWTREECTRL\COMPILEROUTPUTLEVEL	@	REG_SZ	Compiler output level (modify via browser options dialog): infos,errors,warnings: 0 or “” infos,errors,no warnings: 2 errors,warnings: 4 errors only: 6	Brwtree.ocx		

Key	Entry	Type	Description	UsedBy	ValueOnSetup	OEM changable
BRWTREECTRL\COMPILERWARNSIZERESON BRWTREECTRL\COMPILERWARNSIZERESPERCENT	@	REG_SZ	Compiler shows warning if Resource size is less than COMPILERWARNSIZERESPERCENT percent below maximum. Maximum is given by PlcMemSize in hardware ini file. COMPILERWARNSIZERESON: “1” = enabled	Brwtree.ocx Browser passes these settings via command line parameters to the compiler/linker	empty	
BRWTREECTRL\COMPILERWARNSIZESEGON BRWTREECTRL\COMPILERWARNSIZESEGPERCENT	@	REG_SZ	Compiler shows warning if the size of a segment is less than COMPILERWARNSIZESEGPERCENT percent below maximum. COMPILERWARNSIZESEGON: “1” = enabled	Brwtree.ocx Browser passes these settings via commandline parameters to the compiler/linker	empty	
Settings\StartUpScreen	@	REG_SZ	Full Path to startupsceen HTML File. If this entry is empty the default startup is searched in [installdir]\splhtm\\startup.htm	cxedit.exe	empty	
Settings\HiddenItemPrefix	@	REG_SZ	(direct) global variables with this prefix are not shown in margin variables dialog of CFC editor	OpenPCS.exe cfc.ocx	empty	
Settings\OEMCompanyInfo	@	REG_SZ	shown in output window before version info. if empty, "infoteam" is shown	OpenPCS.exe	empty	Yes
Settings\OEMAppname	@	REG_SZ	Application Name sshown in titlebar etc.	openpcs.exe	empty	Yes
Settings\OEMMODE	@	REG_SZ	If set to "1", the encrypt and decrypt items are available in the project menu. Used for library encryption.	BrwTree.ocx	"0"	Yes
Settings\CfcAnyColorMarked	@	REG_SZ	special color in CFC editor	cfc.ocx	255, 255, 80	
Settings\CfcAnyColorUnmarked	@	REG_SZ	special color in CFC editor	cfc.ocx	255, 220, 0	
Settings\CfcBoolColorMarked	@	REG_SZ	special color in CFC editor	cfc.ocx	255, 72, 72	
Settings\CfcBoolColorUnmarked	@	REG_SZ	special color in CFC editor	cfc.ocx	200, 33, 33	

Key	Entry	Type	Description	UsedBy	ValueOnSetup	OEM changable
Settings\CfcColorSchemeConnections	@	REG_SZ	flag if color schemes have to be read from registry	cfc.ocx	1	
Settings\CfcDefaultColorMarked	@	REG_SZ	special color in CFC editor	cfc.ocx	77, 77, 77	
Settings\CfcDefaultColorUnmarked	@	REG_SZ	special color in CFC editor	cfc.ocx	0, 0, 0	
Settings\CfcIntColorMarked	@	REG_SZ	special color in CFC editor	cfc.ocx	51, 102, 255	
Settings\CfcIntColorUnmarked	@	REG_SZ	special color in CFC editor	cfc.ocx	0, 51, 204	
Settings\CfcRealColorMarked	@	REG_SZ	special color in CFC editor	cfc.ocx	0, 255, 0	
Settings\CfcRealColorUnmarked	@	REG_SZ	special color in CFC editor	cfc.ocx	0, 172, 0	
Settings\CFCBlockYOffset	@	REG_SZ	block offset from the upper edge of the grid	cfc.ocx	1	
Settings\ControlUnitObligatory	@	REG_SZ	??	OpenPCS.exe		
Settings\DiasableAccuRangeOnlineMode	@	REG_SZ	If “1”, variable range is used instead of accurange	OpenPCS.exe, ladder.ocx	0	yes
Settings\DisableFileNewDirectory	@	REG_SZ	flag if menu entry new dir is disabled	OpenPCS.exe	0	
Settings\DisableFileNewResource	@	REG_SZ	flag if menu entry new resource is disabled	OpenPCS.exe	0	
Settings\DisableFileProject	@	REG_SZ	flag if menu entrie for project are disabled	OpenPCS.exe	0	
Settings\DisablePLCDownload	@	REG_SZ	flag if menu entry plc download is disabled	OpenPCS.exe	0	
Settings\DisablePLCDownloadInc	@	REG_SZ	flag if menu entry plc incremental download is disabled	OpenPCS.exe	0	
Settings\DisablePLCUpload	@	REG_SZ	If set to 1, the menu entry “plc upload” is disabled. Set to 0 the menu entry “plc upload” is enabled, if the runtime system supports this feature.	OpenPCS.exe	0	
Settings\DisableRecentProject	@	REG_SZ	flag if menu entries for recent projects are disabled	OpenPCS.exe	0	
Settings\Parameters	@	REG_SZ	??	??	0	
Settings\CfcWideBlocks	@	REG_SZ	flag if cfc blocks are wide (value = "1") or narrow (value != "1")	cfc.ovx	0	

Key	Entry	Type	Description	UsedBy	ValueOnSetup	OEM changable
Settings\DlgPropFileName	@	REG_SZ	file name for "Insert Global Variables" Dialog settings	OpenPCS.exe	empty	
Online\TcpDriverTimeout_ms	@	REG_SZ	timeout value for TCP/IP driver in milli seconds	TCPdrv432.dll	30000	yes
Settings\ProtectedGlob		REG_SZ	Pasglop.poe read-only		empty	yes
Settings\CFCFBDMarginFactorInPercent		REG_SZ	Enlargin margin connectors		empty	yes
Settings\CFCFBDBlockWFactorInPercent		REG_SZ	Variable block widths		empty	yes
Settings\CFCFBDBlockWidthFactorInPercent	@	REG_SZ	Variable block widths	cfcfbdcontrol.ocx	empty	yes
Settings\CfcColorSchemeConnections		REG_SZ	Syntax coloring		empty	yes
Settings\CfcPrintComments		REG_SZ	Printing comments		empty	yes
Settings\UseNewCFC		REG_SZ	Use new CFC		0	
Settings\ConfirmPLCStartStop		REG_SZ	If this value is set to 1,a message box for confirmation of Hotstart, Warmstart and Stop is shown.		0	
Settings\DisableActiveDocumentServer	@	REG_SZ	If set to 1, active documents are not shown within OpenPCS but opened in an extra window	openpcs.exe	empty	
Settings\CfcHideOldCatalogDialog	@	REG_SZ	If set to 1, the variable table does not appear on double clicking the skirt board	cfcfbdcontrol.ocx	empty	
Settings\CfcXrefXslPath		REG_SZ	The path to the XSL file for CFC cross reference can be set (without the filename). Project path is used if the key is empty or does not exists.		empty	
Settings\CfcYOffset		REG_SZ	If this key does not exist the default offset is 1. It can be set to 0.		empty	
Settings\AutoDeclarationExternal		REG_SZ	If set to 1, the feature “automatic declaration of external variables” is enabled		empty	
Settings\EnableSaveSystem		REG_SZ	If set to 1, the menu entry “Save System”is enabled if the runtime system supports this feature.		1	

5.16.3.1 只读全局声明文件

为了保护一个已定义的全局声明文件在OpenPCS中不被修改（写保护），注册关键字
HKEY_LOCAL_MACHINE\SOFTWARE\infoteam Software GmbH\OpenPCS\6.5.6\Settings\ProtectedGlob
必须被设置为“文件名”（没有扩展输入点）。如果这个值被设置为0，则允许对这个文件的修改。

5.16.3.2 CFC 入口

5.16.3.2.1 扩大边界连接器

现在边界（连接器）的宽度是可变的并且可通过注册关键字由百分比的形式设置
HKEY_LOCAL_MACHINE\SOFTWARE\infoteam SoftwareGmbH\OpenPCS\6.5.6\Settings\CFCFBDMarginFactorInPercent.
这个值由百分比表示，所以一个100的值表示结果为原始宽度。负值或0的结果也以原始宽度表示。

5.16.3.2.2 可变块宽度

现在所有的固件和混合块的宽度都是可变的并且可通过注册关键字由百分比的形式设置
HKEY_LOCAL_MACHINE\SOFTWARE\infoteam Software GmbH\OpenPCS\6.5.6\Settings\CFCFBDBlockWidthFactorInPercent
这个值由百分比表示，所以一个100的值表示结果为原始宽度。负值或0的结果也以原始宽度表示。
注意：老的布尔注册关键字
KEY_LOCAL_MACHINE\SOFTWARE\infoteam Software GmbH\OpenPCS\6.5.6\Settings\CfcWideBlocks
不再有任何效用。

5.16.3.2.3 语法彩色标记

语法彩色标记可通过配置布尔注册关键字
HKEY_LOCAL_MACHINE\SOFTWARE\infoteam Software GmbH\OpenPCS\6.5.6\Settings\CfcColorSchemeConnections
为1/0来开关。
如果语法彩色标记是打开的那么下列（平行的）注册关键字定义了颜色：
CfcAnyColorUnmarked: 没有标记的ANY类型的连接器的颜色
CfcIntColorUnmarked: 没有标记的INT类型的连接器的颜色
CfcBoolColorUnmarked: 没有标记的BOOL类型的连接器的颜色
CfcRealColorUnmarked: 没有标记的REAL类型的连接器的颜色
CfcCommentColor: 注释的颜色（文本块和别名）
一个注册颜色入口必须有一个3倍字节的值（RGB模式）。如值为“255.0.0”表示深红色。

5.16.3.2.4 打印注释

有一个新的叫做
HKEY_LOCAL_MACHINE\SOFTWARE\infoteam Software GmbH\OpenPCS\6.5.6\Settings\CfcPrintComments.
的注册关键字。如果这个关键字被设置为1，那么CFC将在每个图示页前打印一页注释页。注释和块由一个索引连接，这个索引对整个设计是唯一的。

5.17 安装

为了在安装过程中不会有突然出现的窗口（如：没有与用户间的互动），你需要调用带有一些参数的安装文件：

/s	For silent-mode (no installation-process is shown)
/v/qn	Pass arguments to Msiexec (/v); e.g. /qn means nopop-up window.

为了完成一个安装文件和所有的标准值的“静音”安装（如：”PS5111e.exe”），你需要调用下列命令（假设

安装文件在D:\的根目录下）：

d:\PS656e.exe /s /v/qn

如果希望决定一个指定的安装目录，你需要按如下所示的方式修改命令的调用：

d:\PS656e.exe /s /v"/qn

INSTALLDIR=C:\directory\subdirectory\OpenPCS2008"

在这两种情况下的OpenPCS2004的安装过程中都不会突然出现窗口。

在安装结尾处出现的"AddDrvr.exe"和"Licedit32.exe"不受影响。这两个OpenPCS2004的窗口在任何情况下都会出现。

5.18 CANopen

SmartPLC嵌入式实时系统4.2以上版本准备有CANopen的接口。函数LzsEnvRegisterSysState实现接口协议栈的调用。它通常包括一个开关状态，对协议栈函数的适当的调用取决于系统状态。对协议的主调用入口是函数NetProcessIoData。

CANopen不被OpenPCS编程系统支持（自4.2.0版本起），但是计划在下一个版本实现。

在OpenPCS中不包括协议栈，这个栈必须被OEM自己添加进去。Infoteam当然可以也期望提供帮助。

6 OpenPCS 选项

本节介绍了OpenPCS可用的选项。本节介绍的特征和工具在标准SmartPLC嵌入式实时系统中是不可用的，但是如果需要你可以购买一个单独的产品和/或许可证。

6.1 品牌标签包

OpenPCS品牌标签包可用于不同的语言（如：英语和德语）。每个包的内容为：

1. 用户手册（以各自的语言），电子文档格式
2. 帮助文件资源（以各自的语言）
3. 替换“启动屏”和“关于”所必要的信息

更多信息见CD上的相应目录下的README文件。

6.1.1 创建用户手册和帮助文件

你可以使用ToboHelp Classic200（http://www.blue-sky.com）从一些资源中创建用户手册和HTML帮助文件（\InfoteamBLP\<language>\vXXX\Help\v5\v5e）.

6.1.2 用户 SmartSIM 组件

BLP中包含的仿真资源给出了所有你需要按你的喜好修改的仿真的外观和感觉，如：用你的公司的名称和标志替换infoteam名称和标志。

为了增加其他“组件”到SmartSIM（从16位数字输入和输出模式开始）中，请按下列步骤执行：

在文件Wmodule.cpp中，加入类CwinMyModule的实现，例如：通过拷贝可用的类的实现。

在文件Wmodule.cpp中，加入CwinMyModule的声明。

在文件Config1.io中，加入模式定义。

6.1.3 可定制字符串

6.1.3.1 应用程序名称

在openpcs.exe标题栏中显示的应用程序名称可通过使用如下注册关键字设置：

[HKEY_LOCAL_MACHINE\SOFTWARE\infoteam Software GmbH\OpenPCS\6.5.1\Settings \OEMAppname]

6.1.3.2 输出窗口中的公司信息

在启动OpenPCS后显示在输出窗口中的公司名称可通过下列注册关键字修改：

[HKEY_LOCAL_MACHINE\SOFTWARE\infoteam Software GmbH\OpenPCS\6.5.6\Settings\OEMCompanyInfo]

6.1.4 启动屏

在启动OpenPCS时显示的一个连接到示例工程的HTML网页。这个HTML网页是品牌标签包的一的部分。所有的位图和startup.htm都包含在\InfoteamBLP\<language>\vXXX\Splash\splhtm中。

6.1.4.1 每个工程的启动屏

为每个工程设置一个它自己的启动页是可行的。因此，工程的VAR文件需要被编辑：

[AUTORUN]

STARTUP_SCREEN=startup.html

6.1.5 关于

OpenPCS的关于箱使用一个HTML网页。你可以在OpenPCS的INI目录下找到这个文件（HAB.htm）。

6.1.6 图标

由Visual Studio打开组件来修改OpenPCS的图标。

打开openpcs.exe来改变安装目录中OpenPCS.exe的图标。

打开位于安装目录下的cxedit_res.dll来改变OpenPCS的主结构的图标。

6.2 本地代码编译器

6.2.1 一般信息

6.2.1.1 引言

本节是对本地代码编译程序（ncc）的架构的一个概述。

它介绍了本地代码（nc）生成和执行的方式。

假设你有资源代码，你将在最后发现关于创建本地代码编译程序的章节。

6.2.1.2 NCC 的优缺点

随着编译选项的提出，本地代码使得你的IEC程序执行得更快，但是也需要更多的内存空间为它的额外的本地代码段。

<- slower faster ->

Ucode Native Code Native Code with Native Call

(size only) (normal) (speed only)

<- less memory consumption more memory needed ->

6.2.1.3 兼容性

你可以与相同版本的编程系统和目标系统资源（DP）一起使用本地代码编译程序。

为了使能本地代码的执行，需要在文件smart.plc/inc/oem/config.h.中定义_LZS_NATIVCODE_。

6.2.1.4 如何在编程系统中使用 NCC

生成nc的代码，包含在相应的DLL文件中。为了使用它你需要确认在你的硬件初始化文件的Module部分有如下记录：

```
[Module]
;16bit version
PcodeDll=ncc86.dll
;32bit version
PcodeDll32=ncc86_32.dll
```

本地代码可由两种不同的方式生成。直接本地调用或者不直接本地调用。使用编译器选项’speed only’来生成本地调用或者选项’normal’不直接调用来生成本地代码。

如果本地调用的生成被选择了并且编译器遇到了一条不能被编译的指令，不会有本地代码生成并且在浏览器的信息区域会出现一条适当的消息。

如果你想禁止本地代码的生成，选择编译器选项’size only’。

6.2.1.5 实时系统中本地代码的执行

6.2.1.5.1 基本信息

你不能决定是执行本地代码还是执行解释器代码。实时系统为每个分离的实例检查是否有可用的本地代码段。如果有它将被使用。如果没有解释器代码将被执行。

这意味着在一个本地代码段（NS）中的子程序的调用不作为一个调用来完成，而是作为到LZS的返回来完成。LZS处理这个工作并且调用其它的本地代码，或者如果没有代码被执行，就执行解释器代码。

如果解释器代码调用了一个子程序，LZS将完成控制。

如果你在编译你的IEC程序时选择了’speed only’选项，本地代码将不返回到LZS。在这种情况下本地代码编译器产生到子程序的直接本地调用。因此，选择’speed only’的本地代码编译器比’normal’本地代码快得多。但是这也意味着，你的IEC程序必须被NCC成功地编译。如果发现了一条不能被执行的指令，NCC将不会产生任何的本地代码并且将打印一条参考消息。

如果你喜欢用状态显示来调试你的IEC应用程序，你既可使用’size only’也可使用’normal’来作为最优选项。

假设你选择了选项'size only'，POU以状态显示开始由Ucode的方式执行。

6.2.1.5.2 调用本地代码

在本地代码中有两处入口点。第一个被用于本地代码段对主实例是可用的时候。在这种情况下，函数LzsNccCallNc从函数LzsIpRunTaskOnce中调用。在本地代码中的子POE从函数IpIP_CALL中调用。

在调用本地代码之前段表的地址和相关数据段的地址被存储在处理器寄存器中。

然后IP由一个简单的调用转换成下一个本地代码声明的地址。这个地址由下列工式计算：

StartAddress = SegmentAddress + HeaderSize + Offset

HeaderSize通常为6，假设我们由一个子程序返回并且需要在这里继续执行，那么Offset非空。

本地代码到LZS的返回可以是一个真返回（实例执行结束），也可以是对其它实例的调用。有3个值会被传递给LZS：

NextInst：没有实例调用 0 如果下一个实例已完成（=真返回）

Offset：指令指针的偏移量。稍后我们可以继续执行这个实例的地址（与NextInst=0意义不同）。

Extension：下一个实例是一个固件程序。

LZS也在它自己的实例栈中处理本地代码实例。

6.2.1.6 实现细节

6.2.1.6.1 基本信息

在不使用本地代码编译器的情况下OpenPCS产生了在实时系统中被解释的Ucode。本地代码编译器生成额外地机器绝对工程代码，这个代码相比Ucode可被更快地执行。NCC可由两种不同的方式产生NC，直接或间接本地调用。

如此本地代码段解析所有的Ucode段（CS）并且为每个CS生成一个本地代码段（NS）。

这样做的前提是编译器可以为每个UCODE声明产生一个等价的本地操作码。

所有的UCODE声明不全被实现是可以的。如果你想在这种情况下显示一条消息，需将下列说明加入于ncc.ini中：

[common]

Debug=1

这条出现的消息由十进制给出了关于UCOSE没有实现的信息。所有UCOSE的列表见文件ip_opc.h。在编译处理的时候选择一个较高的值来获得更多的信息。把这个值改为0可以避免这些消息。0也是默认值。

假设一个本地代码段被成功地生成了，那么ncc将修改段表和初始段头。

没有本地码段的情况初始段指针指向这个代码段作为相应的段。如果我们有一个本地代码段，那么它将指向这个新的NS作为相应的段。然后这个本地代码段自己指向这个代码段作为相联段。由此创建了段表：初始段入口的第二元素将不再指向代码段，而是指向本地代码段。本地代码段入口的第二元素现在将指向代码段。

5.3.7节介绍了关于段和段表的更多细节。本地代码编译器适用于80386处理器或更高。这意味着ncc适用于单调的编址模式，32位寄存器和相应的指令设置。

6.2.1.6.2 本地代码段

一个本地代码以其它段一样的段头开始。这个段头有6个字节且包含下列信息：

word 1 size of segment

byte 3 type of segment (kPlcSegNativeCode or kPlcSegNativeCode2)

byte 4 size of header (=6)

word 3 no of corresponding segment

本地代码编译器产生纯粹的机器代码，不使用助记符。所以这个段的其余内容包括了操作码和它的操作数。就处理器选择适当的符号格式（小/大端字节格式）。

6.2.1.6.3 本地代码帮助段

对于每个任务都会生成一个本地代码帮助段。这个段的段号是3（0到9为保留段号。0x0A通常是第一个数据段）。这个段由在lzsdwl.c中的函数LzsCreateInternalSegments(…)创建。这个段的安装被定义在ncc资源中的lzsncc.c文件中。

有了这个段nc和LZS就可以同时使用AE栈和引用寄存器。

6.2.1.7 支持的对齐、字节顺序和字节大小

由于不同的处理器拥有不同的对齐、字节顺序和字节大小，它们的相应指令设置必须与它们的要求一致，从而避免对内存的错误的存取。

下表列举了可用的本地代码编译器支持的对齐、字节顺序和字节大小。

NCC	Alignment			Byte	Endianess	
	1	2	4	Size	Little	Big
ncc86	✓	✓	✓	1	✓	
ncc Pentium Opt	✓	✓	✓	1	✓	
ncc167h		✓		1	✓	
ncc68k			✓	1		✓
ncc563	✓			1	✓	
nch8300h		✓		1		✓
nccPPC			✓	1		✓
nccARM			✓	1	✓	✓
nccTMS320VC33	✓			4	✓	
nccColdFire			✓	1		✓

6.2.1.8 异常处理

本地代码与Ucode的执行拥有不同的异常处理。不是所有的像'无效的数据索引'一样的错误都会被发现。下表给出了可处理的错误综述：

	ncc86	ncc Pentium Opt*	ncc167h	ncc68k	ncc563	nch8300h	nccPPC	nccARM**	nccTMS320 VC33	nccColdFire
invalid array index	✓	✓				✓	✓	✓	✓	✓
division by zero	✓	✓	✓	✓		✓	✓	✓	✓	✓

*）精确识别为NCC奔腾V优化

**）精确识别为NCC ARM7 arm模式，如：在姆指模式由32位指令替代16位指令。

6.2.1.9 附加性能

已实现的功能表：

	ncc86	ncc Pentium Opt*	ncc167h	ncc68k	ncc563	nch8300h	nccPPC	nccARM**	nccTMS230 VC33	nccColdFire
ArrayOptimization	✓	✓								
SpanSegments	✓	✓		✓			✓	✓	✓	
DirectCall	✓	✓	✓	✓		✓	✓	✓	✓	✓
DirectFwCall		✓		✓						✓
FPU		✓								
ByrefAccessOpt***	✓	✓								
NccDirect		✓								

*）精确识别为NCC奔腾V优化

**）精确识别为NCC ARM7 arm模式，如：在姆指模式由32位指令替代16位指令。

***）见5.6.7.3.32节BYREF-Optimization (*_DIR_BYREF_*)

6.2.1.9.1 数组最优化

对于数组存取编译器生成的Ucode序列如下所示：

IP_OP_PUSH_AE

IP_OP_CREATE_REFERENCE

IP_OP_PUSH_REF

IP_OP_ADD_OFFSET

IP_OP_LD_NEAR_2

IP_OP_ST_IND_2

IP_OP_POP_REF

IP_OP_SELCT_ARRAY_IND

IP_OP_POP_AE

本地代码编译器结合这个序列并且生成更多紧凑的代码来代替逐条编译Ucode指令。

此优化可在ncc.ini中进行开关。

[80386]

ArrayOptimization=0

6.2.1.9.2 展开段

一个段的容量被限制最大为64k。因为一个本地代码段比它相应的Ucode代码段大得多，所有有一种可能是大的UCODE段被编译后的代码不能放入一个64k的段。因此将一个本地代码展开为多个段成为可能。每个展开的段都有kPlcSegNativeCode2类型。这些额外的段将被分别下载并且在实时系统中被连结成一个段。

6.2.1.9.3 直接固件调用

如果使用这个功能，函数的地址将在本地代码中计算并且函数被直接调用。因此全局变量RegDS被本地代码修改。在固件函数中没有其它的全局变量会被使用。

为了加注一个固件函数中的异常，需要为两个全局变量blecFBErr_l和Error_g填充适当的值，如：

```
if (fMyFirmwareError)
{
bIecFBErr_l = kIecHardwareError;
Error_g = kIpFirmwareExecError;
return (bIecFBErr_l);
}
```

6.2.1.9.4 NccDirect

当支持NccDirect并且被本地代码编译器使用时，下列代码指令的执行将被优化：

- CALLs
- FAR-access and
- BYREF-access

当使能NccDirect时，有一个额外的补丁段，包含了在下载时所需要的在数据段中的修补特殊数据的信息。这个补丁段的类型为kPlcSegDirFarByRefInfo且有一个固定的编号，因此在初始化文件中的保留段的编号必须增加（大于10的值）。

[Module]

ReservedSegments=11

为了在嵌入式SmartPLC实时系统中使能这个功能，在配置的时候需完成下列定义：

#define _LZS_NCC_DIRECT_

这个定义特别地控制实时系统中的一个累加器来防止不匹配函数的下载。

注：如果使用ByrefAccessOpt (在模块部分)，那么就不能使用NccDirect，反之亦然。

6.2.1.9.5 ncc.ini中的关键字

NCC在ncc.ini文件中查询下列开关：

[common]

SpanSegments=1

DirectFWCall=1

ArrayOptimization=1

NopSeparator=0

Debug=1

如果不能找到这些关键字，那么将使用上面规定的默认值。

6.2.1.10 UCODE 指令执行

OpenPCS本地代码编译器的架构允许本地代码只为一个所有UCODE指令的子集执行。没有执行的UCODE指令将导致在相应POU中本地代码不被执行，但是UCODE可以替代，使程序执行，但只能达到较低的性能。见6.2.1.6.1节怎样在你的代码中确定这些指令。

见5.5.7.2节关于UCODE指令执行的列表。

6.2.2 Intel x86(保护模式)

本节介绍了对Intel处理器本地代码编译器(ncc)和上述所有的细节。

6.2.2.1 在编程系统中如何使用 NCC86

为了编译本地代码，需要包含文件ncc86.dll。使用这个文件你需要确认在你的硬件初始化文件的模块部分有下列入口：

[Module]

PcodeDll=ncc86.dll

[NCC] Exist=1

NccUseExtAE=1

InvalidArrayIndex=1

如果你的RTS支持lreal数据类型，你可以通过在[NCC]部分设置这个标志来使能算术操作（如：add_double）。此外编译器将提出错误extended instruction not supported。然而，如果RTS的lzsncc.c文件不包含需要的代码，那么结果将是不可预料的。

LrealSupported=1

6.2.2.2 执行细节

6.2.2.2.1 本地代码帮助段

对于每个任务都会生成一个本地代码帮助段。这个段的段号是3（0到9为保留段号。0x0A通常是第一个数据段）。这个段的大小256字节并且由下列内容组成：

Byte	Contents
0 – 5	Segment Header
6 – 9	Address of Segment table
10 – 13	AE stack pointer
14 – 17	Address of reference register
18 – 21	Number of current instance
22 – 23	Errorcode
24 – 27	Firmware function
28 – 29	Firmware extension
30 – 31	Firmware instance
34 – 35	OP code
36 – 37	OP code ext1
38 – 41	Accumulator(AE)
42 – 45	Shadow accumulator (AEs)
46 – 49	Address of the interpreter function
50	Typecast from

51	Typecast to
52 – 55	Address of extended accumulator (ExtAE)
56 – 59	Address of extended shadow accumulator (ExtAEs)
60 – 67	Accumulator8 (AE8) (used for 8 byte data types)
68 – 73	Shadow accumulator8 (AEs8) (used for 8 byte data types)

有了这个段nc和LZS就可以同时使用AE栈和引用寄存器。

6.2.2.2.2 寄存器组

ncc不使用段寄存器组。在调试模式的开发环境中使用这些寄存器会导致系统崩溃。所以ncc只使用数据寄存器组。80386有4个这样的寄存器：EAX,EBX,ECD和EDX。

本地代码的操作需要段表和相应数据段的地址。当进入nc时这两个值被存储在ECX和EDX中。

EAX和EBX被作为累加器AE和影子累加器AEs的代表使用。

操作通常需要2个或更多寄存器，如：计算一个地址。在这些情况下寄存器的内容被临时地放入处理器的默认栈中。当希望在ECX中进行一个操作时其它实例是移动和循环的操作。Cx也通常被用于循环的默认自动计数器。

6.2.2.2.3 ncc.ini中的关键字

NCC86在ncc.ini中使用附加的关键字：

[80386]

SubFloat=0

这个关键字只为内部调试处理采用。

6.2.2.2.4 any（字符串/数组/结构体）的存取模式的最优化

当装载一个any变量时，所有的any变量将被拷贝进RegExtAE(s)。这非常消耗时间，但是为正确的执行所需。

因此，出现了不这样做的模式：

1 Load any

2 Store any / string / array

3 Load / return / call / jmp

在第一点编译器解析转发并且如果它遇到第2点和第3点时，它将只装载指向这个any的指针并且为存储设置一个标志位来获知怎样处理它。这对用户来说是快捷的并且不需要任何特别的设置。

6.2.3 NCC 奔腾 V（最优化）

本节详细介绍了Intel处理器奔腾x86的本地代码编译器(ncc)。

6.2.3.1 在编程系统中如何使用 NCC 奔腾 V（最优化）

为了编译本地代码，需要包含文件ncc86.dll。使用这个文件你需要确认在你的硬件初始化文件的模块部分有下列记录：

[Module]

PcodeDll= ncc86_PentOpt_32.dll

在你的硬件初始化文件中的NCC部分的默认配置如下所示：

[NCC]

Exist=1

InvalidArrayIndex=1

DivisionByZero=1

LrealSupported=1

FpuEnable=1

ArrayOptimization=1

NccDirect=0

更多设置的细节见6.2.3.2.3节。

6.2.3.2 执行细节

6.2.3.2.1 本地代码帮助段

对于每个任务都会生成一个本地代码帮助段。这个段的段号是3。这个段的大小256字节并且由下列内容组成：

Offset	Content
0	Segment header (6 Bytes)
6	Address of segment table
10	AE stack pointer
14	Address of reference register
18	Number of current instance
22	Errorcode (i.e. DIVISION_BY_ZERO) (1 byte)
24	firmware function
28	firmware extension
30	firmware instance
34	opcode
35	opcode ext1
38	AE
42	AEs
46	Interpreter function
50	Typecast from
51	Typecast to
52	RegExtAE
56	RegExtAEs
60	Akku8
68	SAkku8
76	Temporary data(4 Bytes)
80	Address of firmware function table
84	Address of RegDS

帮助段用于从NC到实时系统解释器的回收，如：形态转换。AE栈或引用寄存器也被存储在帮助段内部，所以他们可以被NC和LZS同时使用。

6.2.3.2.2 寄存器组

Ncc使用和保留x86的寄存器如下所示：

EAX	Accumulator (AE)
EBX	Shadow accumulator (AEs)
ECX	Work register, i.e. used as the default automatic counter for loops
EDX	Address of current data segment
ESI	Address of the segment table
EDI	Address of the help segment

6.2.3.2.3 ncc.ini中的关键字

Ncc在硬件初始化文件的ncc部分使用下列附加的关键字：

Setting	Default	Description
InvalidArrayIndex	1	检查无效的数组索引
DivisionByZero	1	检查除以0的计算
LrealSupported	0	如果你的RTS支持lreal数据类型，你可以通过在[NCC]部分设置这个标志来使能算术操作（如：add_double）。此外编译器将提出错误extended instruction not supported。

FpuEnable	1	在浮点数上使用浮点指针协助处理算术操作
ArrayOptimization	1	最优化数组存取的执行
DirectFwCall	1	最优化直接调用固件函数的执行
NccDirect	0	执行下列代码指令的最优化：CALLs, FAR-access, 和BYREF-access (更多信息见6.2.1.9.4)

6.2.3.2.4 浮点算术

下表列出了进行浮点运算的UCode指令。它显示了计算是否可以在x86处理器的浮点单元进行和/或被回调到实时系统（X=支持,/=不支持）。

UCODE instruction	FPU supported	Callback to RTS supported
IP_EX_ABS_FLOAT	X	X
IP_OP_ADD_FLOAT	X	X
IP_OP_SUB_FLOAT	X	X
IP_OP_MUL_FLOAT	X	X
IP_OP_DIV_FLOAT	X	X
IP_OP_GT_FLOAT	X	X
IP_OP_GE_FLOAT	X	X
IP_OP_LE_FLOAT	X	X
IP_OP_LT_FLOAT	X	X
IP_EX_SIN	X	X
IP_EX_COS	X	X
IP_EX_TAN	/	X
IP_EX_ASIN	/	X
IP_EX_ACOS	/	X
IP_EX_ATAN	/	X
IP_EX_SQRT	X	X
IP_EX_EXP	/	X
IP_EX_EXPT	/	X
IP_EX_MIN_FLOAT	/	X
IP_EX_MAX_FLOAT	/	X
IP_EX_LOG	/	X
IP_EX_LN	/	X
IP_EX_ABS_DOUBLE	X	X
IP_EX_ADD_DOUBLE	X	X
IP_EX_SUB_DOUBLE	X	X
IP_EX_MUL_DOUBLE	X	X
IP_EX_DIV_DOUBLE	X	X
IP_EX_GT_DOUBLE	X	X
IP_EX_GE_DOUBLE	X	X
IP_EX_LE_DOUBLE	X	X
IP_EX_LT_DOUBLE	X	X
IP_EX_SIN_DOUBLE	X	X
IP_EX_COS_DOUBLE	X	X
IP_EX_TAN_DOUBLE	/	X
IP_EX_ASIN_DOUBLE	/	X
IP_EX_ACOS_DOUBLE	/	X

IP_EX_ATAN_DOUBLE	/	X
IP_EX_SQRT_DOUBLE	X	X
IP_EX_EXPT_DOUBLE	/	X
IP_EX_MIN_DOUBLE	/	X
IP_EX_MAX_DOUBLE	/	X
IP_EX_LOG_DOUBLE	/	X
IP_EX_LN_DOUBLE	/	X

6.2.4 Infineon C16x（巨模式）

6.2.4.1 前言

本节是对16位微控制器家族SAB C167的本地代码编译器(ncc)的架构的一个概述。

示例是针对Keil-C166-Compiler/Linker而写。

6.2.4.2 本地代码的产生

6.2.4.2.1 基本信息

本地代码编译器解析所有的u-code段（CS）并且为每个CS分别生成一个本地代码段（NS）。

这只在编译器可以为每个UCODE声明产生等价的本地操作码时发生。

假设一个本地代码段已经成功生成了ncc，那么需要修改段表的初始段头。

没有本地代码段是初始段作为相连段指向代码段。如果我们拥有一个本地代码段，那么它将作为相联段指向这个新的NS。本地代码段自己作为相联段指向代码段。由此创建了段表：初始段入口的第二元素将不再指向代码段，而是指向本地代码段。本地代码段入口的第二元素现在将指向代码段。

6.2.4.2.2 产生本地代码的接口

生成nc的代码被包含在文件ncc167h.dll中。为了使用这个代码你需要确认在你的硬件初始化文件的模块部分有下列记录：

[Module]

PcodeDll=ncc167h.dll

它使用方法GeneratePCODE调用自己。

Alignment=2是另一个必须设置的选项。当然，在相同文件中的固件声明必须也要有2字节对齐的布局。

如果你想禁止本地代码的生成，移除PcodeDLL入口即可。

6.2.4.2.3 本地代码段

一个本地代码段以与其它段一样的段头开始。这个段头有6个字节且包含了下列信息：

byte 0 size of segment (lobyte)

byte 1 size of segment (hibyte)

byte 2 type of segment (kPlcSegNativeCode)

byte 3 size of header (=6)

byte 4 no of corresponding segment (lobyte)

byte 5 no of corresponding segment (hibyte)

本地代码编译器生成纯十六进制代码，不使用任何助记符。所以这个段的其余内容包括了80C167的操作码和它的操作数。

6.2.4.2.4 本地代码帮助段

针对每个任务都会生成一个本地代码帮助段。这个段的段号是3（0到9为保留段号。0x0A通常是第一个数据段）。这个段的大小32字节并且由下列内容组成：

Byte	Contents
0 – 5	Segment Header
6 – 9	Operand 1 of float operations
10 – 13	Operand 2 of float operations
14 – 17	Result of a float operation

18 – 21	Reference register
22 – 23	Errorcode
24 – 25	Number of current instance
26 – 29	AE stack pointer
30 – 33	Pointer to the current native code segment (behind header)
34 - 35	Opcode number for called functions in the RTS
36 - 37	Extended Opcode number for called functions in the RTS

38-255字节不用。

6.2.4.2.5 寄存器组

通用寄存器（GPRs）有如下用途：

R1	User stack
R2	Accumulator (AE) – low word
R3	Accumulator (AE) – high word
R4	GPR
R5	Address working register LOW
R6	Shadow accu (AEs) - low word
R7	Shadow accu (AEs) – high word
R8	Address working register HIGH
R9	Address of own (actual) data segment LOW.
R10	Address of own (actual) data segment HIGH.
R11	GPR
R12	Address of the segment table LOW
R13	Address of the segment table HIGH
R14	GPR
R15	GPR

6.2.4.3 运行本地代码

6.2.4.3.1 基本信息

实时系统将分别检查每个实例是否有可用的本地代码段。如果有将被使用。如果没有解释器代码将被执行。

这意味着一个在NS中的子程序的调用不作为一次调用来执行而是作为对RTS的返回来完成。RTS执行这个工作并且调用另一个本地代码，或者如果没有本地代码被执行，则调用解释器代码。

如果解释器代码调用了一个子程序，RTS将得到这个控制权。

6.2.4.3.2 调用本地代码

在本地代码中有两个入口指针。第一个被用于本地代码段对主实例是可用的时候。

函数LzsNccCallNc从函数LzsIpRunCycle中调用。在本地代码中的子POE从函数IpIP_CALL中调用。

在调用本地代码之前段表的地址被存储在GPR R12中，已标记的数据段的地址被存储在R9中。

本地代码到LZS的返回可以是一个真返回（实例执行结束），也可以是对其它实例的调用。有3个值会被传递给RTS：

NextInst：没有实例调用 0 如果下一个实例已完成（=真返回）

Offset：在一个函数或函数块执行后继续执行实例的偏移量。

Extension：固件程序的编号。

RTS也在它自己的实例栈中处理本地代码实例。

6.2.4.4 实时系统 RTS 中的变化

6.2.4.4.1 资源文件

你需要infoteam SmartPLC实时系统4.0版本来正确地运行本地代码4.0。从3.4.3版本到4.0实时系统的下列关于本地代码的文件被修改了：

smart.plc/inc/oem/config.h

smart.plc/lzs/lzstable.c

smart.plc/lzs/ip_cmds1.c

smart.plc/lzs/ncc167/lzsncc.c

为了使能本地代码的执行，请确认在文件smart.plc/inc/oem/config.h中有对_LZS_NATIVCODE_的定义。

6.2.4.4.2 创建RTS

为了生成实时系统你必须使用LARGE内存模块。然后你需要确认所有的本地代码段（不是其它的代码、数据或初始数据段）都位于一个64k的段中。你可以通过文件lzsenv.c中的如下两个函数来完成：

例：

```
void LzsEnvBeginDwlResource(void)
{
pNativeCode = NATIVECODE_ADDRESS;
}
LZSBYTE LzsEnvMoveSegment(...)
{
...
case kPlcSegNativeCode:
MemCopy(pNativeCode, *ppSeg_p, wSegSize_p);
            FREE(*ppSeg_p);
            *ppSeg_p = pNativeCode;
            pNativeCode += wSegSize_p;
            if (pNativeCode > 0x03EFFF) { pNativeCode = LZSNULL;
                  pNativeCode = LZSNULL;
                  bRetCode = kLzsNoMem;
            }
      ...
}
void LzsEnvMemFree (void LZSFAR* pMem)
{
            if (pMem >= NATIVECODE_ADDRESS) return;
```

如果声明NATIVECODE_ADDRESS被定义为，如：0x30000，那么你必须保留这个内存区域（在本例中从0x300000到x3EFFF）。有很多文件来保留内存，一个方法是通过使用在连接文件中的RESERVE关键字。

例：

```
            RESERVE (0x000008-0x00000B,
              ...,
            0x030000-0x03EFFF,          ; Native Code
              ...)
```

在你的连接文件你需要为一个语言函数加入一个额外的SECTIONS控制，这个语言函数将被本地代码调用（见1.6节）。

6.2.4.5 NCC.INI 文件

本地代码编译器需要一些关于在PLC上的内存布局的信息。这些信息将从ncc.ini文件中获得，你必须把这个文件放在C:\Documents and Settings\All Users\Application Data\infoteam Software\OpenPCS2008\Openpcs.520目录下。ncc.ini文件有如下形式：

```
[C167]
Codeseg=0x3
FirmwareOffset=0xFC00
FloatOp=0xEF00
```

记录”Codeseg”指定了放入RTS的64k的段，记录”FloatOp”表示在一些情况下（如，浮点操作）被本地代码调用的C语言函数的偏移地址。另一个偏移地址由记录”FirmwareOffset”指定。这个记录只在直接调用固件函数块（“直接调用”的意思是一个函数块直接地被本地代码调用）时使用。这些记录在连接文件中必须与下列SECTIONS规范一致：

例：

```
SECTIONS (?PR?NCCDISP%FCODE(0x03EF00),
            ?PR?NCCFB%FCODE(0x03FC00))
```

这些函数被自命名为“CallFloatOpreations()”和CallFirmwareFB()并且被定义在文件nccdisp.c和中nccfb.c。

注意：

如果你想要使用封装函数块（支持US-Ladder的固件FB），你必须为调用这些FB而扩展函数CallFirmwareFB()。

6.2.5 Motorola 68376

本节介绍了Motorola 68376中本地代码编译器的更多细节。

6.2.5.1 在编程系统中如何使用 NCC68k

生成nc的代码被包含在文件ncc68k_32.dll中。为了使用它你必须在你的硬件初始化文件的模块部分有如下记录：

[Module]

PcodeDll32=ncc68k_32.dll

6.2.5.2 执行细节

6.2.5.2.1 寄存器组

68000处理器有8个32位数据寄存器（D0-D7）和8个32位地址寄存器（A0-A7）。所有的寄存器都在本地代码中被使用。它们中的一些为特殊用途所保留。

A7	address of the user stack
A6	address of the AEStackPointer
A5	address of the segment table
A4	address of the current data segment
D3	accu (AE)
D4	Shadow accu (AEs)
D0	Reference register
D6	instance handle

6.2.5.2.2 本地代码调用约定

在调用本地代码前所有的68000寄存器将被入栈然后出栈。所以所有的寄存器都可以被本地代码使用。一些寄存器拥有特殊用途：

当进入本地代码时，下列寄存器将被设置：

A0	Address of the help segment
A3	Start address of the native code
A4	Address of the current data segment to work on
A5	Address of the segment table
A6	Address of the AEStackPointer
D6	Current instance

当从本地代码返回时，这些这些寄存器有如下返回值：

A2	Last offset in current instance (for the UCODE instruction CAL)

6.2.5.2.3 运行UCODE和本地代码的区别

自从处理器有2字节对齐后，意味着在奇数地址上布局字将不被允许，与解释器一起运行UCODE和直接的运行本地代码，它们的执行是不同的。UCODE解释器按字节存取WORD和DWORD，而本地代码按word或dword存取。因此可以声明一个有直接INT变量的程序并且存储在奇地址上（如：INT AT%MW3）。在相同的程序上运行本地代码可以终止PLC。

6.2.6 Hitachi H8/300

本节介绍了在Hitachi H8/300H处理器上的本地代码编译器（ncc）。

6.2.6.1 在编程系统中如何使用 NCC

生成nc的代码包含在文件nch8300h.dll中。为了使用它你必须确认在你的硬件初始化文件的模块部分有以下记录：

[Module]

PcodeDll=nch8300h.dll

本地代码可由两种方式生成。直接本地调用生成或不直接本地调用。使用编译器选项'speed only'来生成本地调用或选项'normal'非直接地调用生成本地代码。如果本地代码的生成已经选择好并且编译器遇到了一条不能被编译的指令，将不会有本地代码生成并且在浏览器的信息区域会出现适当的消息。

如果你想要禁止本地代码的生成，选择选项'size only'即可。

6.2.6.2 本地代码帮助段

针对每个任务都会生成一个本地代码帮助段。这个段的段号是3（0到9为保留段号。0x0A通常是第一个数据段）。这个段的大小256字节并且由下列内容组成：

0-5　　6字节段头（像本地代码段）

段表地址

AE栈指针

引用寄存器地址

当前实例数量

错误代码（如：DIVISION_BY_ZERO）

这个段可同时被nc和LZS用于使用AE栈和引用寄存器。

在帮助段的下一个记录被在本地代码调用一个固件函数或来自LZS的函数时使用。

24　　固件函数

28　　固件扩展

固件实例

Opcode(操作码)

Opcode_ext1

AE

AEs

C函数

形态转换-from

形态转换-to

6.2.6.3 Motorola 模式

Hitachi处理器与Motorola处理器的字节命令很像（大端字节顺序）。与默认情况下infoteam编程系统在所有段中生成的Intel样式是相反的。在<hardware>.ini文件中可对其进行开关：

[Module]

Motorola=1

如果使用了这个功能，实时系统也必须转换到Motorola模式。这个由将文件szmbyte.c转换为szmbyt68.c实现。

6.2.6.4 寄存器组

HITACHI H8/300H处理器有8个32位的数据寄存器（ER0-ER7）。

每个数据寄存器都可以作为8位、16位或32位寄存器使用。

所有的寄存器都在本地代码中使用。它们中的一些已经为特殊用途保留。

ER7	Stack pointer
ER6	Start-address of the current code segment
ER5	Start-address of the segment table
ER4	Mostly used as pointer to the current data-location
ER3	For pointer calculations
ER2	For pointer calculations
ER1	Shadow accu(AEs)
ER0	Accu(AE)

6.2.6.5 实时系统中本地代码的执行

在本地代码中有两个入口指针。第一个用于如果一个本地代码段对主实例是可用的情况。在这种情况下函数LzsNccCallNc将从函数LzsIpRunCycle中调用。在本地代码中的一个子POE从函数IpIP_CALL中被调用。

在调用本地代码之前段表的地址将被存储在ER5中，相应数据段的地址将被存储在ER6中。

然后IP转换到下一个带有一个简单调用的本地代码声明的地址上。这下一个本地代码声明的地址按下列方式计算：

StartAddress = SegmentAddress + HeaderSize + Offset

Header的大小通常为6，假设我们从一个子程序返回且需要在这个段中继续执行时Offset非空。

本地代码返回到LZS可以是一个真返回（实例已经执行结束），也可以是一个到其它实例的调用。有3个值被传递回LZS：

NextInst：没有实例被调用 如果最后一个实例已经完成则为0（真返回）

Offset：指令指针的偏移量。指向我们可以继续执行实例的地址（与NextInst=0意义不同）。

Extension：假设下一个实例是一个固件程序。

6.2.7 Motorola PowerPC

本节介绍了在Motorola PowerPC处理器上的本地代码编译器(ncc)的更多细节。

6.2.7.1 在编程系统中如何针对 Motorola PowerPC 使用本地代码编译器

生成nc代码被包含在文件nccPPC_32.dll中。为了使用它你需要确认在你的硬件初始化文件中的模式部分有如下记录：

[Module]

PcodeDll32=nccPPC_32.dll

[NCC]

Exist=1

NccUseExtAE=1

NccHelpSegAligned=1

6.2.7.2 执行细节

6.2.7.2.1 寄存器组

Motorola PowerPC处理器包含下列寄存器：

32个32位通用寄存器（GPR）。它们作为数据源或目的寄存器为整数指令服务并且为生成的地址提供数据。

32个64位的浮点寄存器（FPR），用作数据源和目的地（浮点寄存器不由NCC使用，由于浮点指令由解释器执行，所以它们也可以在没有FPU的系统上使用）。

1个32位的条件寄存器，被分为8个4位字段，CR0-CR7,反映算术操作的结果和为测试和转移提供一个途径。

1个32位的浮点状态和控制寄存器（FBSCR），包含23位状态位和8位控制位，用于使能或禁止浮点异常和控制浮点操作使用的舍入模式。

1个32位的XER寄存器，为整数操作显示溢出和进位条件。

1个32位的连接寄存器（LR），为某些转移指令（对LR有条件的执行）提供转移目标地址。

1个32位的计数寄存器（CTR），处理在正确编码的转移指令被执行期间会被递减的循环计数。它也可以为某些转移指令（对CTR有条件的执行）提供转移地址。

在一些Motorola PowerPC的实现中，GPRs,LR和CTR是64位宽的。

寄存器组：

LR	Address for branch instructions
R0	AE
R1	address of the stack-pointer
R3	address of the segment table
R4	address of the current data segment
R5	Instance handle

R6	Pointer to far segment
R7	Help register for bit-operations
R10	AEs
R20	Help register
R21	Help register 2
R22	Help register 3
R23	Help register for load/store operations with offset > 32767
R24	Reference register
R31	Frame pointer

6.2.7.2.2 本地代码调用约定

在对本地代码的调用前，所有本地代码编译器使用的寄存器必须被存储在栈中然后从栈中恢复。所以只有已经存储的寄存器可以在本地代码中使用。一些寄存器有特殊用途：

当进入本地代码时，下列寄存器必须被设置：

LR Start address of the native code

R1 Address of the stack-pointer

R3 Address of the segment table

R4 Address of the current data segment to work on

R5 instance handle (= segment number) of the current instance

当从本地代码返回时，下列寄存器将返回相应的值：

R3 Next instance to be called (for the UCODE instruction CAL)

R4 Extension (used with firmware functions)

R5 code position at which to continue if the instance returns to call another instance

6.2.7.2.3 运行UCODE和本地代码的区别

自从处理器有4字节对齐，意味着在地址空间上布局不是它的长度的整数倍的操作数将不被允许，在解释器中运行Ucode和直接运行本地代码是不同的。UCODE解释器按字节存取WORD和DWORD变量，然而本地代码按word或dword。因此当使用UCODE时，拥有直接INT变量的程序可在奇地址上声明和存储（如：INT AT%MW3）。运行本地代码时，相同的程序可使PLC中止。

6.2.7.2.4 帮助段

本地代码帮助段通过LZS初始化并且用于LZS和NC间的通信。它有下列结构：

Byte	Contents
0 – 7	Segment Header
8 – 11	Address of segment table (not used)
12 – 15	AE stack pointer (not used)
16 – 19	Address of reference register (not used)
20 – 23	Number of current instance
24	Error code (e.g. division by zero)
28 – 31	Firmware function
32 – 35	Firmware extension
36 – 39	Firmware instance
40	OP code
44	OP code ext1
48	Accumulator (AE)
52	Shadow accumulator (AEs)
56	Address of the interpreter function
60	Typecast from
64	Typecast to

68	Address of global pointer
72	Extended accumulator 1 (ExtAE1)
76	Extended accumulator 2 (ExtAE2)

6.2.7.2.5 错误管理

对于PowerPC使用错误管理是可以的。这个管理将在帮助段中设置错误代码并且返回给LZS。为了使用这个功能，在文件NCC.INI中需要有如下记录设置：

[common]

DivisionByZero=1

InvalidArrayIndex=1

6.2.8 ARM7(ARM 模式)

本节详细介绍了ARM7处理器的本地代码编译器。

6.2.8.1 在 ARM 处理器（ARM 模式）中如何使用本地代码编译器

生成nc的代码被包括在文件NCCARM7_ARM.dll中。为了使用它你需要确定在你的硬件初始化文件的模块部分有如下记录：

[Module]

PCodeDll32=NCCARM7_ARM.dll

[NCC]

Exist=1

NccUseExtAE=1

6.2.8.2 本地代码段

本地代码段以一个和其它段一样的段头开始。这个段头有8个字节且包含下列信息：

byte 0 size of segment (lobyte)

byte 1 size of segment (hibyte)

byte 2 type of segment (kPlcSegNativeCode)

byte 3 size of header (=6)

byte 4 no of corresponding segment (lobyte)

byte 5 no of corresponding segment (hibyte)

byte 6 fill-byte to align header

byte 7 fill-byte to align header

本地代码编译器生成纯16进制代码，不使用任何助记符。所以这个段的剩余部分为32位的操作码。

6.2.8.3 执行细节

6.2.8.3.1 寄存器组

ARM处理器总共有37个32位的寄存器。这些寄存器的可见性取决于当前处理器的模式。在任何时候，15个通用寄存器是可存取的（R0到R15）。它们中的一些为特殊用途所保留。当前程序状态寄存器（CPSR）在所有的处理器模式中也都是可存取的。它包含条件代码标志和其它的状态信息。

R0	Help register
R1	Address of the segment table
R2	Address of the help segment
R3	Address of the current data segment
R4	Accu (AE)
R5	Shadow accu (AEs)
R6	Offset register
R7	Base register for far access
R8	Low DWORD of 8-byte-Accu

R9	High DWORD of 8-byte-Accu
R10	Low DWORD of 8-byte-SAccu
R11	High DWORD of 8-byte-SAccu
R12	unused
R13	Stack pointer
R14	Link register
R15	Program counter
CPSR	Program status register

6.2.8.3.2 本地代码调用约定

在调用本地代码之前寄存器R4到R9将被入栈然后出栈。寄存器R1到R3不保存到栈，因为它们被用于到本地代码的参数传递。寄存器R10到R12目前没有使用。

当进入本地代码时，下列寄存器将被设置：

R1 Address of the segment table

R2 Address of the help segment

R3 Address of the current data segment to work on

当从本地代码返回时，下列寄存器将返回相应的值：

R0 Return code

R1 Extension, if the next instance is a firmware routine

R2 Last offset in current instance (for the UCODE instruction CAL)

R3 Next instance: No of instance to call

6.2.8.3.3 运行UCODE与本地代码的区别

ARM处理器有4字节的对齐。在地址空间上布局不是4字节对齐的字是不允许的。因此在解释器中运行UCODE和直接运行本地代码是不同的。UCODE解释器按字节存取WORD和DWORD变量，然而本地代码按word或dword。因此当使用UCODE时，拥有直接INT变量的程序可在奇地址上声明和存储（如：INT AT%MW3）。运行本地代码时，相同的程序可使PLC中止。

6.2.8.3.4 内存系统的字节顺序

本地代码编译器支持两种字节顺序，小端字节顺序内存系统（Intel）和大端字节顺序内存系统（Motorola）。这个配置可在<hardware>.ini文件中进行转换：

[Module]

Motorola=1

如果使用了这个功能，实时系统也应该转换为相应的字节顺序。这可通过编辑实时系统来完成。

6.2.8.3.5 数据类型为Double的指令

数据类型为double指令被转发到LZS。因此double类型的参数放在寄存器R0到R3中被传递给LZS。DWORD的8字节参数的顺序可由下述的文件ncc.ini中的记录进行配置。在默认配置中低DWORD位于相应的低寄存器。

[ARM7_ARM]

; Default DWORD-order

; R0: Low-DWORD of first param

; R1: High-DWORD of first param

; R2: Low-DWORD of second param

; R3: High-DWORD of second param

SwitchRegDW=0x00 ; Angabe hex

; Switched DWORD-order

; R0: High-DWORD of first param

; R1: Low-DWORD of first param

; R2: High-DWORD of second param

; R3: Low-DWORD of second param

; SwitchRegDW=0xFF ; Angabe hex

6.2.8.3.6 本地代码帮助段

Nc帮助段存储本地代码使用的信息。Nc帮助段中的入口被分成常量元素和其它元素，在执行一个任务之前这些元素必须被更新。

常量元素是指向LZS函数的函数指针且只被初始化一次，在下载完成之后。第56到第183个字节的记录符合对于数据类型为float的指令的函数指针，第184到第307个字节的记录符合对于数据类型为double的指令的函数指针。

Nc帮助段的动态部分在LZS的每个周期被更新一次，如：扩展的Akku入口。

6.2.8.3.7 Cirrus EP93xx FPU支持

Cirrus EP93xx FPU（单片上系统）模式的特色是一个ARM内核与一个Maverick Crunch协同处理器集成在一起。

如果在硬件初始化文件中有如下设置，那么本地代码编译器就可以为这个协同处理器生成代码：

[NCC]

FPU=4

加速的操作：add/sub/mull（只限类型为REAL的数据）

如果操作系统的硬件在默认条件下不能这样做，那么就有必要明确地使能这个协同处理器。完成这个工作的代码由文件lzs\NCCARM7_ARM\MaverickCrunch_FPU.提供。

6.2.9 ColdFire

6.2.9.1 如何为 ColdFire 使用本地代码编译器

生成nc的代码包括在文件nccColdFire.dll中。为了使用它你需要在你的硬件初始化文件的模块部分有下列记录：

[Module]

PcodeDll32=nccColdFire.dll

Alignment=4

Motorola=1

6.2.9.2 执行细节

NCC在一个ColdFire5272处理器上开发和测试。本地代码应该与任何一个低版本的ColdFire兼容（高于或等于V2版本）。

ColdFire的所有寄存器（D0-D7,A0-A6,A7是栈指针）都在本地代码中使用。在进入本地代码之前，这个寄存器必须被存储（入栈）并且下列值应被设置到相应的寄存器：

wNextInstance →D6

pAdrHS →A0

RegInstSP →A1

pSegTab_l →A5

pAdrDS →A4

当然RegAESP也应该被设置到帮助段中的相应位置。（见文件lzsncc.c中的定义）

本地代码的起始地址被装载在A3中并且调用已经完成。

本地代码返回下列值：

D5 – wNextInst

D2 – wExtension

A2 – dLastOffset

全局变量RegAESP必须在本地代码执行完成后从帮助段中读回。

在本地代码执行完成后这些寄存将被从栈中恢复。

6.3 OEM 接口工具箱

OEM接口工具箱（叫做OEMKITVxxx）不分开销售，它与其它OpenPCS产品捆绑在一起，叫做本地代码编译器。它提供创建模块所需要的报头和库来接口一系列OpenPCS编程系统提供的拦截。为了更新在一个新版本编程系统上使用这些拦截的模块，只有用适当的OEM接口工具箱的报头和库来重建这些模块。

6.4 数据类型可选项

6.4.1 数据类型 WSTRING

WSTRING数据类型描述了在IEC61131-3，2rd版本提出的由双字（16位）组成的Unicode字符串工作的一种手段。

在实时系统中支持WSTRING数据类型是一个可选的功能。

Data type	WSTRING
CBE type	kCbeTypeWString
Structure	Analogous toSTRING datatype (see section 5.6.7.4.1.)String length is specified as number of characters.

6.4.1.1 配置

为了在实时系统中支持WSTRING数据类型，加入下列定义到你的oem/config.h文件中：

#define _IP_WSTRING_

为了在OpenPCS中支持WSTRING，如入下列配置到你的GLOBDATA.INC文件中：

[DATA_TYPES_IEC_NORM]

WSTRING=WSTRING

此外WSTRING实例的最大长度可通过加入下列设置到你的硬件说明文件中：

[Module]

MaxWStringLength=<length>

6.4.1.2 UCode 指令

WSTRING UCode被包含在EX2组：

UCode	**Hex**	**Dec**
IP_EX2_ST_NEAR_WSTRING	0xEB	235
IP_EX2_ST_FAR_WSTRING	0xEC	236
IP_EX2_ST_BYREF_WSTRING	0xED	237
IP_EX2_ST_IND_WSTRING	0xEE	238
IP_EX2_ST_DIR_BYREF_WSTRING	0xEF	239

6.4.2 指针数据类型

无类型指针是一个可选的功能并且不是IEC语言的扩展。

无类型指针：

Data type	*POINTER*
CBE-Typ	kCbeTypeAddress
Grösse	8 Byte

POINTER的内部表示：

Element	Address	Bit address	CBE type	Size
Size	DWORD	BYTE	BYTE	WORD

POINTER变量通常初始化为0.在OpenPCS中声明的指针常量NULL可供使用。

6.4.2.1 在 OpenPCS 中的使用

指针声明：

```
VAR
        intp : POINTER;
END_VAR
```

为了存取一个变量的地址，需要在变量名前写一个地址操作符'&'。

例如：

```
IL: LD &i
    ST: intp := &i;
```

6.4.2.2 配置

由于实际上POINTER是一个可选的功能，下列部分必须被插入到GLOBDATA.INC以激活编译器的支持：

```
[DATA_TYPES_IEC_IEC_NORM]
POINTER = ADDRESS
```

注：OpenPCS需要重启来接受变更。

6.4.2.3 应用

POINTER变量各自的地址生成只允许在下列指令后使用：

LD,ST,EQ和NE

所有其它的指令将在语法检查时报错。

没有间接引用的运算符。也没有指针算法可以使用。

POINTER变量不能是CONSTANT或RETAIN。

在下列情况下，地址生成是允许的：

函数块的参数（INPUT,OUTPUT,INOUT）

常数变量（CONSTANT）

数组存取

函数块实例

在声明部分

POINTER变量（指向指针的指针）

6.4.2.4 从扩展固件中存取指针

为了存取指针需要提供宏GETPOINTER()。

GETPOINTER()是针对LzsMemNearGetPointer(tPlcMemPtr, tPlcPointerInfo *)的宏。这个函数写数据到已给出的结构体tPlcPointerInfo中。

```
typedef struct {
    tPlcMemPtr    pAddr;
    LZSBYTE       bBitAddr;
    tCbeType      bType;
    LZSWORD       wSize;
} tPlcPointerInfo;
```

6.4.2.5 UCode 指令

为了分别的与LD，ST一起使用地址生成，下列UCode将被使用。所有的UCode使用'IP_EXT2_'作为前缀。

UCode	Short description	Details	Parameter
LDA_NEAR	Load Address Near	Creates a pointer to a variable in the local data segment and saves it in the accu	WORD wOffset BYTE bBitAddr BYTE bCbeType WORD wSize

LDA_FAR1	Load Address Far	Creates a pointer toa variable, that is not in the local data segment, and saves it in the accu	WORD wFarSegNr WORD wFarOffset BYTE bBitAddr BYTE bCbeType WORD wSize
LDA_BYREF	Load Address ByRef	Creates a pointer to a variable that is given by a reference in the local data segment. The pointer is saved in the accu.	WORD wOffset BYTE bBitAddr BYTE bCbeType WORD wSize
LDAS_NEAR	Load Shadow Address Near	Analogous to above but with the difference that the pointer is written to the shadow accu.	
LDAS_FAR*	Load Shadow Address Far		
LDAS_BYREF	Load Shadow Address ByRef		
1) FAR instructions are not used at the moment because the adress operator is not allowed to be used with parameters.			

6.4.3 Date/Time 数据类型

Date/Time数据类型支持包括数据类型DATE,TIME_OF_DAYT和DATE_AND_TIME以及在这些类型上的算术操作。

这是一个可选的实时系统功能，对TIME数据类型的支持通常是可用的。

同样在IEC标准函数块RTC中也包括这个支持。

Data type	Size	Unit	minimum value (=default value)	Maximum value	Notes
DATE	32-bit, unsigned	days	0001-01-01	11759222-01-20-23:59:59.999	
TIME_OF_DAY	32-bit, unsigned	ms	00:00:00.000	23:59:59.999	Normalized to 24h
DATE_AND_TIME	64-bit, unsigned	This type is a combination of DATE and TIME_OF_DAY			

6.4.3.1 配置

对Date/Time数据类型和RTC函数块在实时系统中的支持通过加入下列定义到文件oem/config.h来完成：

#define _IP_8BYTEDATE_

#define _IP_ANYDATE_

在OpenPCS中由GLOBDATA.INI文件分配需要被扩展的硬件：

[DATA_TYPES_IEC_IEC_NORM]

DATE = DATETIME

TIME_OF_DAY = DATETIME

DATE_AND_TIME = DATETIME

6.4.3.2 UCode 指令

Date/Time UCode被包含在EX组：

UCode	**Function**	**Hex**	**Dec**
IP_EX_ADD_DT	DT := T + DT	0xBF	191
IP_EX_SUB_DT_TIME	DT := DT – T	0xC0	192
IP_EX_SUB_DT_DT	T := D – D	0xC1	193
IP_EX_ADD_TOD_TIME	TOD := TOD + T	0xFA	250
IP_EX_SUB_DATE_DATE	T := D – D	0xFB	251
IP_EX_SUB_TOD_TIME	TOD := TOD - T	0xFC	252
IP_EX_SUB_TOD_TOD	T := TOD - TOD	0xFD	253

6.4.3.3 LZS 指令

6.4.3.3.1 配置

RTC函数块不返回系统的时间但是需要它作为一个输入变量。因此一个叫做GetSystemTimeAndDate的进一步的函数被定义来返回系统的时间和日期。类似的函数块SetSysetemTimeAndDate允许设置这两个值。两个函数块都作为OEM函数实现，因为它们需要平台指定调节器。

文件smart.plc/oem/oemfbsystime.c为这两个OEM函数块自己的实现提供了一个构架。

示例WIN32IPC包含一个Microsoft Windows的实现示例。头文件lzsdttime.h和lzsdttime.c定义了辅助函数来转换日期（day/month/year）和时间（hours/minutes/seconds/milliseconds）为内部表示法（DATE或相应的TIME_OF_DAY）。

6.5 外部变量自动声明

自从OpenPCS5.5.0版本以来，在程序和函数块中的ST,FBD和Ladder的外部变量就可以自动声明了。它可通过注册设置激活（见“注册入口”章节）。

如果这个功能被激活，用户就不再需要在VAR_EXTERNAL声明的方式声明引用为全局变量了。全局变量可以在一个POU的指令部分被简单地使用，不用在这个POU中做任何声明。所有在这个POU中的定义都不仅局部声明，而且全局变量，都将自动地加入到生成的POE文件中的VAR_EXTERNAL声明部分中。

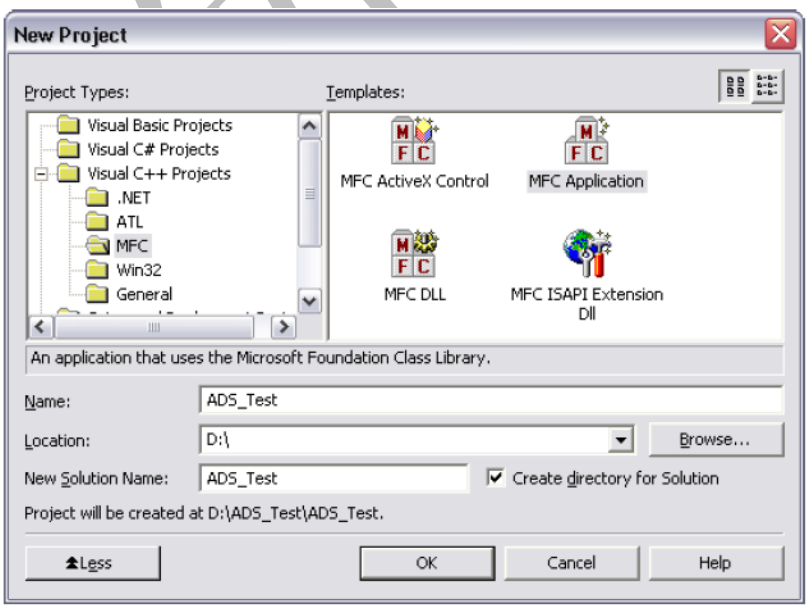
6.6 有效文档服务

OpenPCS包含一个有效文档服务接口，这意味着所有的注册有效的文档都为OpenPCS所支持。这个关联由文件扩展名完成。

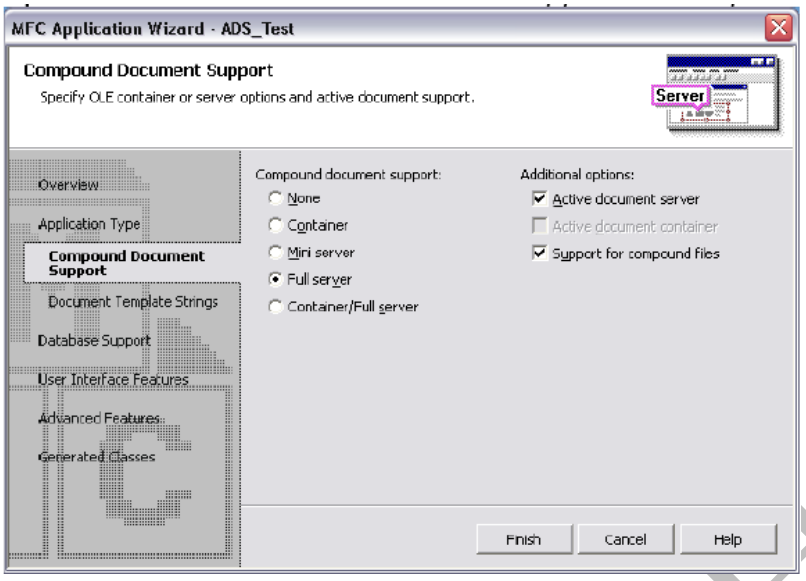
当由OpenPCS打开一个有效的文档时，这个文档不被关联的唯一的应用程序打开，而是被OpenPCS像一个编辑器文件一样打开。

6.6.1 怎样由 VS.NET 创建一个嵌入式应用程序

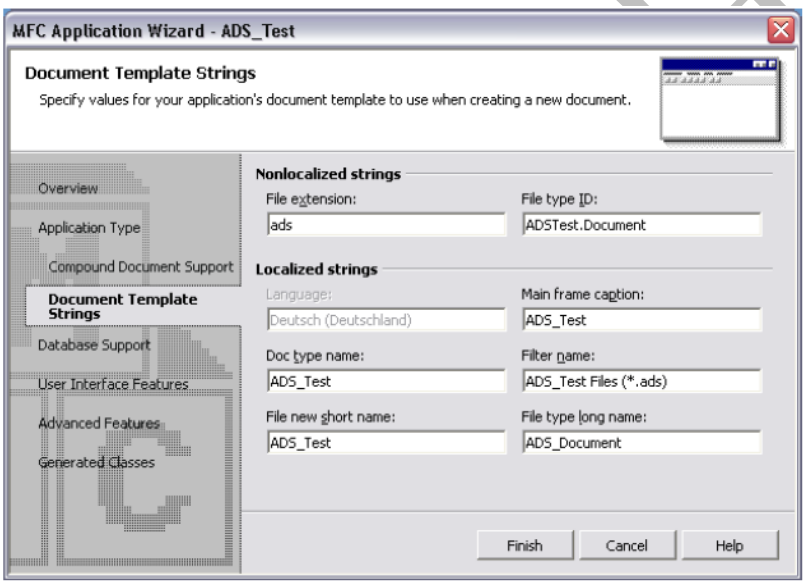
1、从VS.NET创建一个新的C++ MFC应用程序



2、将Compound Document Support声明为Full Server并且激活选项Active Document Server和Support for compound files。



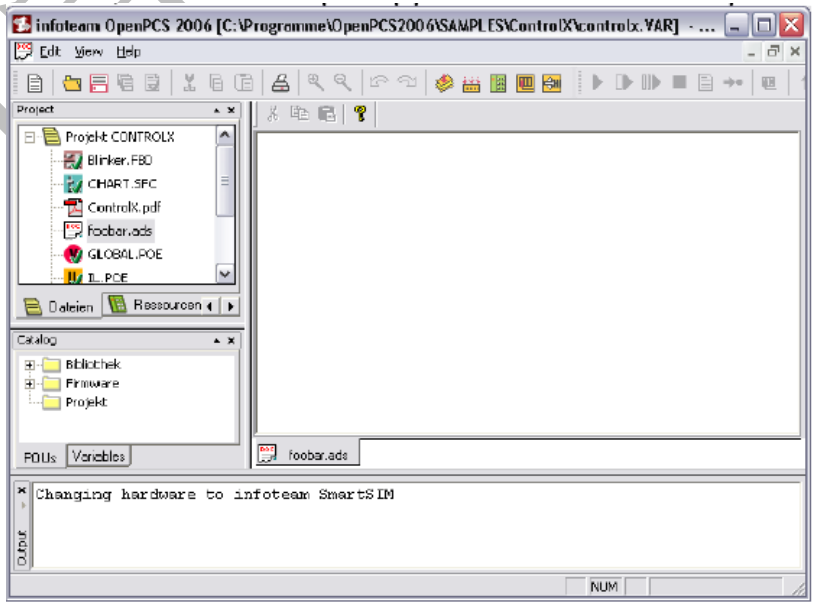
3、然后声明文件扩展名



4、为需要显示的想要的文件类型编写代码。

5、OpenPCS像一个综合编辑窗口一样打开这个文件

（下面有一个第1步到第3步的示例应用程序）



6.7 OpenPCS 命令行选项

当你从命令提示符中调用openpcs.exe时，你可以指定一些命令行选项。

/RegServer	Register server
/UnregServer	Unregister server
/DUMP_ON	Add debugging information when saving .ldd-filesPerform several diagnostic services in _DEBUG builds.
/H /? /Help	Shows a dialog with useful command line commands
/ONLDBASE	Specify database which to use to go online
/ONLCON	Specify connection which to use to go online

7 术语表

deutsch		english
Aktuelles Ergebnis	AE	german for: CR
Ablaufsprache	AS	german for: SFC
Anweisungsliste	AWL	german for: IL
Benutzerbaustein	BB	german for: UB
CAN Application Layer	CAL	CAN Application Layer (CAN)
Controller Area Network	CAN	Controller Area Network
Continuous Function Chart	CFC	Continuous Function Chart
CAN in Automation e.V.http://www.can-cia.de(CAN)	CiA	CAN in Automation e.V., http://www.can-cia.de (CAN)
CAN base message specification (CAN)	CMS	CAN base message specification (CAN)
Communication Object (CAN)	COB	Communication Object (CAN)
COB Identifier (CAN)	COB-ID	COB Identifier (CAN)
englisch für: AE	CR	current result
Code Repository Data-Structure	CRD	Code Repository Data-Structure
Code Segment	CS	Code Segment
COB-ID Distributor (CAN)	DBT	COB-ID Distributor (CAN)
Data Segment	DS	Data Segment
Funktionsblock	FB	Function Block
englisch für: FBS	FBD	Function Block Diagram
Funktionsbausteinsprache	FBS	german for: FBD
englisch für: FP	FC	functionchart
Funktionsplan	FP	german for: FC
Gerätekonfigurator	GKF	Device Configuration Utility
Short for Iconics GraphWorX	GWX	Short for Iconics GraphWorX
englisch für: AWL	IL	instruction list
Kontaktplan	KOP	german for: LD
englisch für: KOP	LD	Ladder diagram
Layer Management (CAN)	LMT	Layer Management (CAN)
Laufzeitsystem	LZS	Runtime System
Microsoft Foundation Classes	MFC	Microsoft Foundation Classes
Network Management (CAN)	NMT	Network Management (CAN)
Original Equipment Manufacturer	OEM	Original Equipment Manufacturer
OLE for Process Control	OPC	OLE for Process Control
Programming and Debugging Tools	PADT	Programming and Debugging Tools
process data object (CAN)	PDO	process data object (CAN)
englisch für: SPS	PLC	Programmable Logic Controller
service data object (CAN)	SDO	service data object (CAN)
englisch für: AS	SFC	Sequential Function Chart
Speicherprogrammierbare Steuerung	SPS	german for: PLC
Strukturierter Text	ST	Structured Text
Test und Inbetriebnahme	TUI	Test and Commissioning Tool
englisch für: BB	UB	userblock