**插入元件**

插入元件包括插入网络、触点、线圈、块元件、跳转和返回。插入元件可以通过三种操作完成，在框架菜单中添加元件、单击工具条中的图标添加元件、右键添加元件

插入网络节点：包含向前插入与向后插入

1. 向前插入网络节点 ID为ID\_INSERT\_LD\_BEFORE\_NETWORK
   1. View类的消息相应函数处理流程 CLDView：：OnAddLDBeforeNetwork()



* 1. POU中加入网络前节 CLDPOU：：AddBeforeNetwork（）



* 1. 增加与粘贴元件的关键函数是 CLDPOU::AddTotal(chType)插入前节的类型为NORMAL\_PASTE



1. 向后插入网络节点

与向前插入网络节点类似，区别在于（1）：在View类调用POU的CLDPOU：：AddAfterNetwork；（2）调用CLDPOU：：AddTotal（），传递的参数为BELOW\_PASTE

1. 插入触点

插入触点与插入网络实现流程类似，区别在于（1）在View类调用POU的CLDPOU：：AddLDContact（），在该函数中创建一个新的触点（2）调用CLDPOU：：AddTotal（），传递的参数为NORMAL\_PASTE,在实现的过程中在函数IsAbleEN（）中将触点作为Assign的孩子节点建立父子关系，如果插入的触点不是第一个触点，会和前一个触点构造一个新的元素（AND\OR）,用其id取代pList。

插入并联触点与插入触点的区别在于：（1）调用POU的CLDPOU：：AddParalletContact（）；（2）在该函数中创建新的触点，调用AddTotal（），传递的参数为BELOW\_PASTE

1. 插入线圈

与插入触点的实现流程类似，主要区别在于AddTotal（）中传入的参数为RIGHT\_PASTE

1. 插入跳转与插入返回的实现流程类似。
2. 插入块元件的实现流程与前面几种大体类似，但是在插入之前进行了很多处理

**删除元件**

删除元件操作：在删除元素的入口函数中CLDPOU：：DeleteElement（）函数中从m\_chosenIDList中设置成员的删除标志SetUnReTag（）；并调用RemoveElement（）删掉元素，父节点删掉孩子节点，删除元件流程图如下所示：



其中POU的删除操作主要流程为：



在线情况下元件不能被选中。选中输入引脚线，其m\_uchSelType为INPIN\_TEXT\_SEL

**移动元件**

鼠标拖动选中的元件可实现元件的移动，其主要涉及两个操作：（1）选中元件的删除；（2）目标位置元件的添加操作，实现函数为CLDView：：DealWithMoveElementOnLButtonUp中，流程图如下所示：



拖动元素注意：

1. 只能拖动位于连接线上的单个元素
2. 触点和算法块不能移动到网络的输出区，线圈不能移动到网络的编辑区，跳转和返回不能移动。

**选中元件**

选中元件分两种情况，单选和多选。涉及到的变量有m\_pChooseElement，m\_choenIDList，m\_pShiftElement，具体来说：单选时：pChooseElement为选中元素的指针；多选时：m\_choenIDList记录了多选选中元素的ID号；m\_pShiftElement是当选中元素时同时按下了shift键。

1. **单选**

鼠标左键不与键盘相配合，鼠标左键单击元件区域来实现选择单个元件的，实现流程图如下所示：

其中调用基本元件的MouseClick函数，注意的一点是如果鼠标所在区域为TEXT区域，则自动生成CLabelEdit控件，可以实现编辑操作。

**2）多选**

多选分两种方式：一为按住Shift键同时用鼠标左键单击要选中元件；二为按住鼠标左键拖动出一个矩形区域进行圈选。不能多选引脚、输出类元素和触点、功能块不能多选等。进行多选的元素必须满足以下条件：

1. 元素必须在一个网络中
2. 元素必须是触点、算法块或网络。

按住Shift方式下多选流程如下图所示：



其中按住Shift的同时选定LD元件，OnLButtonDown中nFlags的值为5

圈选方式的实现流程图如下所示：OnMouseMOVE



圈选的实现调用ShiftLMouseClick(),涉及到的变量主要是pLDPOU->m\_circleRect.

计算鼠标所在的元素，是LD编辑器所有鼠标操作的基础；首先需要得到鼠标所在的屏幕坐标，然后得到在LD编辑器中的视图坐标；然后再判断鼠标在POU注释上，或者在某个网络节点上，还是在编辑器中空白区域；如果在网络节点上，将由网络节点进一步判断在哪个元素上。

**增加和删除输入端**

**1）增加输入端**

对一部分库管理器下的功能块指令，可以对输入端进行增加操作。增加的输入端追加到块元件的最后一个输入端位置。功能块元件增加输入端的实现流程如下图所示：



其中增加输入端可能选中的是Box本身，也可能是Inpin,会在具体插入位置的时候进行判断。

**2）删除输入端**

调用CLDView：：OnDeleteElement（）实现的流程参见前面元件的删除操作，具体即选定的输入端从Box端删除，清除选定元件标志，对视图进行显示更新。

**块元件使能端的显示与隐藏**

所有的块元件都具有使能输入端和输出端，在显示上可以使能端显示或者隐藏。

1. **显示块元件使能端**



1. **隐藏块元件使能端**



具体在实现过程中，显示隐藏使能端调用接口一样，入口都为CLDView：：OnSwitchEN（）

**剪切、复制和粘贴**

剪切、复制和粘贴都是和剪切板相关的操作

剪切板内置在windows中，并且使用系统的内部资源RAM，或虚拟内存来临时保存剪切和复制的信息，可以存放的信息种类是多种多样的。剪切或复制时保存在剪切板上的信息，只有再剪切或复制另外的信息，或停电，或退出Windows，或有意地清除时，才可能更新或清除其内容，即剪切或复制一次，就可以进行多次粘贴。

剪切板支持几种标准的格式，本文要放到剪切板的内容不是标准格式，因此需要定义私有的剪贴板格式。

在这些操作中涉及到的主要类是CFixedSharedFile sf内存文件;清空剪贴板EmptyClipboard（）；

1. **复制**

复制完成的功能为：将选中的元件信息存入到剪贴板上，其实现流程图如下所示：



其中判断选中情况分四种情况：（1）SEL\_MUL\_IN\_OR选中OR元件中的多个元素，生成OR元件（2）SEL\_MUL\_IN\_AND 选中AND元件中的多个元素，生成AND元件（3）SEL\_MUL\_NETWORK选中多个网络，生成CLDElement对象 （4）其他

在拷贝的过程中首先存储元件类型，元件名称，再存储元件信息。元件信息对应每一种元件的成员变量

1. **粘贴**

粘贴执行的操作是：将剪贴板上的信息还原成LD元件，并将元件加入到LDPOU中，m\_pasteIDList记录了所粘贴元件ID的信息。如果LDPOU已经达到包含元件的最大数量，则不能再粘贴元件。粘贴的实现流程图如下：



1. **剪切**

剪切完成的功能是：将选中的元件信息放入剪切板中，然后删除选中元件。执行过程类似于复制，但是在将元件的所有信息放入剪贴板之后，要删除所选元件，并重新绘制POU视图。

注册剪切板：

UINT LD\_Clipboard = RegisterClipboardFormat（\_T（”LD\_CLipboard”））；

UINT LD\_Clipboard = RegisterClipboardFormat(\_T(“LD\_CLipboard”));

VC++/MFC中设计剪贴板操作的内容有（1）文本内容的操作；（2）WMF数据的操作；（3）位图的操作；（4）设置使用自定义格式（5）感知剪贴板内容的改变（6）自动将数据粘贴到另一应用程序窗口

其中设置并使用自定义格式：使用RegisterClipboardFormat（）函数，可以复制和粘贴任何你需要的数据类型。

**撤销和恢复**

撤销和恢复行为模式是围绕操作命令为核心设计的。

在每次编辑操作中，将记录选中元素在逻辑存储树结构的父节点元素，并将其放入m\_UndoList中，进行撤销操作时，从m\_UnDoList中取出最末端元素添加到LDPOU中并刷新视图；每进行一次撤销操作，将从m\_UnDoList中取出的元素加到m\_ReDoList中，从而进行恢复操作时，从m\_ReDoList中取出末端元素添加到POU中，并刷新视图。

**修改元件文本**

判断选中元素的类型是否是POU注释、网络节点注释、元素变量名、元素别名、跳转标签，如果是这些类型中的某一种，则在所选矩形区域创建文本编辑控件，就可以修改元件文本编辑的操作。

修改元件变量：修改之后的文本如果为未定义的合法文本，则弹出变量自动声明对话框。调用流程为CLDView：：OnEnter（）---》CLDView：：Enter（）---》CLDView::IsUpdateForSyncWithVar()-🡪CLDView::AutoVarDeclare();

弹出自动声明对话框调用语句为：CGlobalFunc：：AutoVarDeclare（）；

**置反**

判断选中元素是否是线圈、返回、触点、跳转、算法块引脚中的一种，如果是则可以进行置反操作，否则不能。这些元件都有一个成员变量为bool m\_bNegate

**线圈置位/复位**

当选中的元素为线圈时，可以进行置位/复位操作，其他元件不可以

**查找替换**

查找替换操作时涉及到的查找替换结构体定义如下：

typedef struct tagSearchAndReplace

{

Int iElementID; //当前查找到元素的ID

char chTextType;//待查找Text内容域

Int iFindTextIndex;//当前查找到的字符串的索引

char chPinType;//当前查找的字符串位于输入引脚，输出引脚还是元素上

int iPinIndex;//引脚索引值

CString strFind;//查找框的串

CString strInstead;//替换框的串

BOOL bMatchWholeWord;//全字匹配

BOOL bMatchCase;//大小写敏感

CString strNewName;//替换之后字符串值

Bool bIsReplaceAll;

Bool bIsReplaceInit;

}SearchAndReplace;

查找操作中搜索选项分为：全字匹配与区分大小写；范围分为当前页面和整个工程；调用CLDPOU：：OnSearch（）----》CLDPOU：：OnGlobalSearch（）；

**查找：**

全局查找的过程是从根结点开始，依次查找的过程。总的实现流程如下所示：



CLDNetwork：：GlobalSearch(CString strFind,BOOL bMatchCase,BOOL bMatchWholeWord,CLDPOU \*pPOU)；实现的是从网络节点中，查找Label和注释

全局查找函数结束后，将信息添加到CFind类中，输出查找结果。

查找顺序分为三类：

1. 元素间顺序：总体方向为：从上到下，从左到右的顺序。从左到右指一个网络内部的查找顺序，其中在一个网络内部，有并联关系的元素之间按照从上到下的顺序查找。网络之间的查找按照从上到下的顺序。
2. 元素内部顺序：块元件：类型、变量名、别名；其他：变量名、别名。
3. 字符串内部顺序：按照字符从左到右顺序，不回退查找。

**替换：**

调用CLDPOU：：OnReplace（）------》CLDPOU：：OnGlobalReplace();打开替换对话框有三种方式用于选择：查找下一个；替换；全部替换。

全部替换的大体实现流程如下所示：



**右键操作**

在LD编辑器中，当用户右击鼠标，弹出式菜单出现在光标所在位置。弹出式菜单是通过利用CMenu类和其成员函数，在程序运行过程中动态建立的。一般而言，弹出式菜单是利用现用的菜单项来进行创建，但也可以为弹出式菜单专门建立一个菜单资源，然后通过调用函数CMenu：：LoadMenu（）装入所创建的菜单资源。本文就是采用的这种方法。

CMenu menu；

VERIFY（menu.LoadMenu(IDR\_LD\_POPUP)）；

根据选中元素的不同，调用menu的不同子菜单项。例如CMenu \*pPopup = NULL;

如果选中的触点，则pPopup = menu.GetSubMenu(0);如果选中的是网络则pPopup = munu.GetSubMenu(1);

最后调用pPopup->TrackPopupMenu(TPM\_LEFTALIGN|TPM\_RIGHTBUTTON，pt.x, pt.y, this);

鼠标操作的时候都需要进行设备坐标到逻辑坐标的转换，相应的代码如下：

CPoint pt = point;

This->ScreenToClient(&point);

CClientDC dc(this);

OnPrepareDC(&dc, NULL);

dc.DPtoLP(&point);

在VC中鼠标坐标的坐标位置用设备坐标表示，但所有GDI绘图都用逻辑坐标表示，所以用鼠标绘图时，那么必须将设备坐标转换为逻辑坐标，可以使用CDC函数DptoLP()将设备坐标转化为逻辑坐标，同样可以用LPtoDP()将逻辑坐标转化为设备坐标。

**Tip提示功能：**

Tip提示使用控件CToolTipCtrl类m\_ToolTipCtrl，在用户鼠标位置置于不同元件以及不同的区域时，进行tip提示，在View类进行Tip提示相关的代码函数为：

m\_ToolTipCtrl.Activate(TRUE);

m\_ToolTipCtrl.UpdateTipText(strShowText, this);