网址：http://www.microvt.com
Email: hightechyoung@yahoo.com
作者：HighTech Young

6.3. 调用动态链接库(DLL)

相对于 CIN 来讲，NI 更推荐用户使用 DLL 来共享基于文本编程语言开发的代码。除了共享或重复利用代码，开发人员还能利用 DLL 封装软件的功能模块，以便这些模块能被不同开发工具利用。在 LabVIEW 中使用 DLL 一般有以下几种途径：

1. 使用自己开发 DLL 中的函数。
2. 调用操作系统或硬件驱动供应商提供的 API。

对于前一种方法来说，又可以通过以下几步来实现：

a) 在 LabVIEW 中定义 DLL 原型；
b) 生成.C 或.C++文件，完成实现函数功能的代码并为函数添加 DLL 导出声明；
c) 通过外部 IDE（如 VC++）创建 DLL 项目并编译生成.dll 文件。
d) 在 LabVIEW 项目中使用 DLL 中的函数。

以下章节将通过实例对这两种情况详细进行叙述。

6.3.1. 配置 Call Library Function Node(CFN)

无论在 LabVIEW 中使用自己开发的 DLL，硬件驱动供应商或者操作系统提供的 API，都可以通过配置 Call Library Function Node (CFN，图 6-12)来完成。

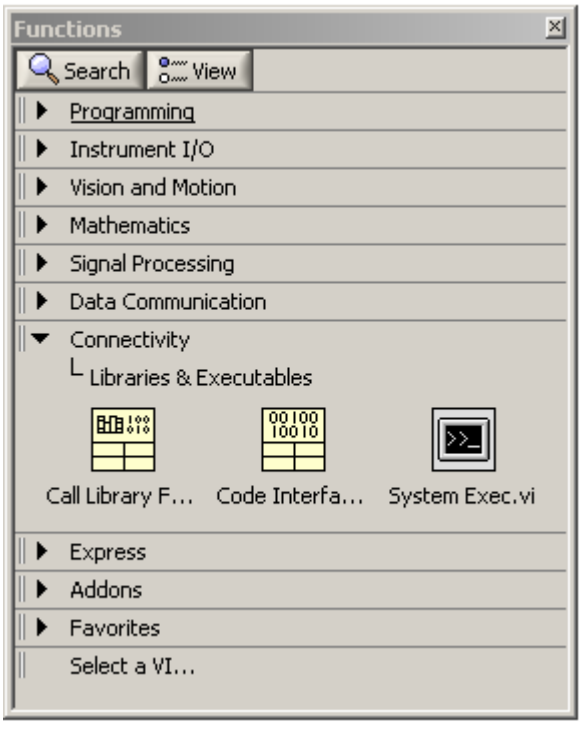


图 6-12 LabVIEW 的 Call Library Function Node

在 CFN 图标的右键菜单上选择“**Configure...**”打开 Call Library Function 配置对话框(图 6-13)。通过该对话框，可以指定动态库存放路径、调用的函数名以及传递给函数的参数类型和函数返回值的类型。在配置完后，CLF 节点会根据用户的配置自动更新其显示。

通过 **Browse** 按钮或者直接在“**Library Name or Path**”输入框中指定调用函数多在.dll 文件的路径。

通过 **Browse** 按钮下的控件用户可以指定多个线程同时调用 DLL。默认情况下，LabVIEW 以“**Run**

网址：http://www.microvt.com
Email: hightechyoung@yahoo.com
作者：HighTech Young

in UI Thread”方式调用 DLL，调用的函数将直接在用户线程中运行。另外一种方式为递归方式“**Reentrant**”，在这种情况下可以允许多个线程同时调用 DLL 中的函数。但要确保正常调用，必须使 DLL 中的代码线程安全。以下是一些最基本的线程安全特性：

a. 代码不含有未受保护的全局数据（如全局变量，文件）；
b. 代码不访问硬件（即不含有寄存器一级的代码）；
c. 代码不调用非线程安全的函数、DLL 或者驱动；
d. 代码使用信号量或者互斥量来保护全局量；
e. 代码被一个非递归的 VI 调用时为线程安全。

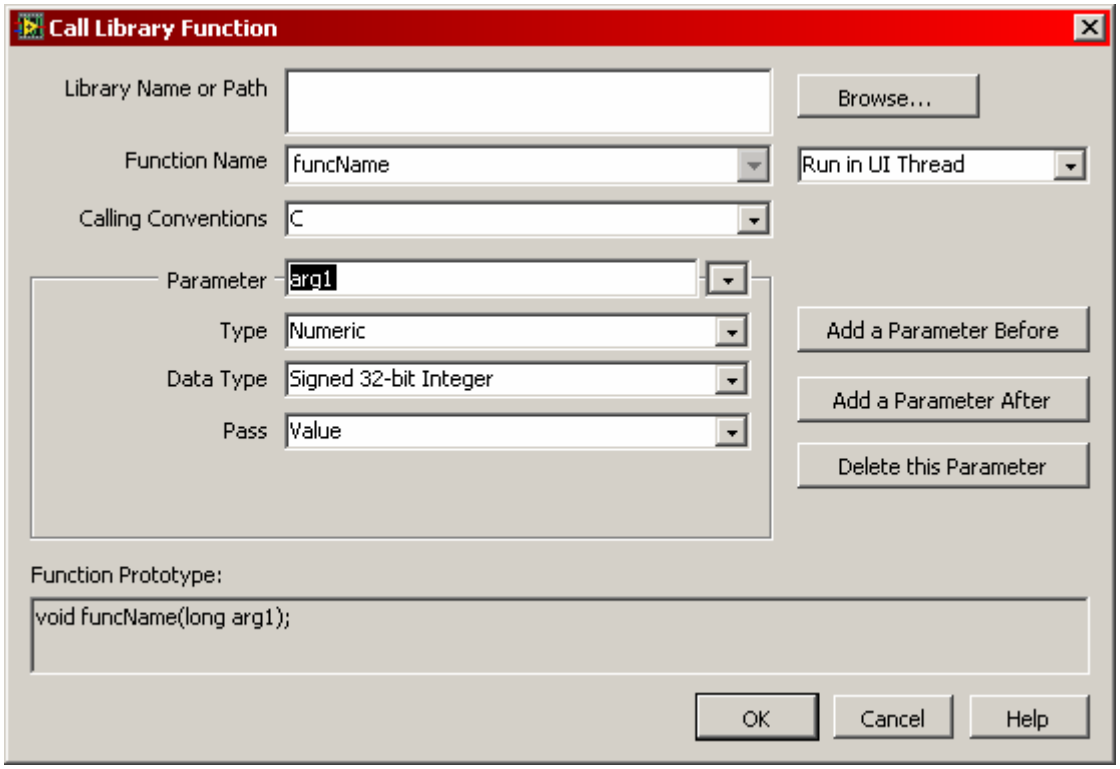


图 6-13 CLF 配置对话框

在“**Function Name**”输入框中指定要调用函数的函数名。

通过“**Call Conventions**”下拉列表框指定调用 DLL 中函数的方式。可以指定调用方式为“C”（默认方式）或 Windows 标准调用方式“**stdcall**”。一般来说用“**C**”方式调用开发人员自己写的 DLL 函数，而“**stdcall**”一般做为标准调用方式来调用 windows 的 API

通过 **Parameter** 域可以指定所调用函数的返回值类型。默认情况下 CFN 节点没有输入参数而且只有一个 **void** 类型的返回参数。该参数由 CFN 节点第一对连接点的右端返回，代表 CFN 执行结果。如果返回参数的类型是 void 类型，则 CFN 连接点为未启用状态（保持为灰色）。CFN 的每一对连接点代表一个输入或输出参数，若要传递参数给 CFN 则将参数连接至相应连接点的左端，若要读取返回值，则将相应连接点的右端连接到 Indicator。

CFN 返回参数的类型可以是 **Void**，**Numeric** 或 **String**。只能为返回参数指定 **Void** 类型，输入参数不能指定为 **Void** 类型。调用的函数没有返回值时，指定 CFN 的返回参数类型为 void 类型。即使参数有确定类型的返回值，也可以指定 CFN 的返回类型为 **Void**，但是此时，函数的返回值将被忽略。

有些时候，调用的函数返回值不是以上三种类型，可以使用与以上三种类中有相同大小的一个来代替。例如如果调用的函数返回一个 Char 类型数据，则可以用一个 **8-bit unsigned integer** 的 **Numeric** 类型来代替。此外，由于 LabVIEW 中没有指针，因此调用 DLL 中的返回指针的函数似乎不可能。但是可以设定返回值类型为一个与指针有相同大小的 Integer 类型，LabVIEW 将把地址以整型值来看待，并且用户可以在以后的调用中直接使用它。

网址：http://www.microvt.com
Email: hightechyoung@yahoo.com
作者：HighTech Young

通过 **Parameter** 域和其右边的**“Add a Parameter Before”，“Add a Parameter After”** 和**“Delete this Parameter”**三个按钮可以增加、删除以及修改 CFN 的输入参数和类型。当用户选择某参数的类型后，其详细的数据类型列表和参数传递方式列表将显示出来，以方便进行详细设定。表 6-1 列出了可以设定的输入参数类型及其详细数据类型信息。

参数类型	说明
Numeric	**Numeric** 型参数有以下几种数据类型: ● 8、16、32 和 64 位符号和无符号整型（**Signed or Unsigned Integer**） ● 4 字节单精度类型（**4-byte single**） ● 8 字节双精度类型（**8-byte Double**） 利用“**Pass**”下拉列表指定传递参数指针或是传递参数的指针。
Array	Array 型参数的数据类型同 **Numeric** 型参数的数据类型; 通过 Dimensions 指定数组的维数; Array Format 有以下几种选项: ● **Array Data Pointer** 传递一维指针到数组值 ● **Array handle** 传递一个指向数组每一维的 4 个字节指针的指针，其后为数组的值 ● **array handle Pointer** 为数组的句柄传递一个指针
String	用 String Format 下拉列表框选择字符串类型 ● C String Pointer—以 null 字符结束的字符串 ● Pascal String Pointer—附加长度字节的字符串 ● string handle—一个指向 4 个字节长度信息指针，其后为字符串的值。 ● string handle Pointer
Waveform	Waveform 类型的参数默认类型为 **8-byte double**，因此一般没有必要为其指定数据类型。但是必须为其指定维数，如果参数为单独的 Waveform，则指定 Dimensions 为 0，如果参数为 waveform 数组，则指定 Dimensions 为 1。 注意：LabVIEW 不支持超过 1D 的 waveform 数组！
Digital Waveform	如果参数为 digital waveform 数组，则指定 Dimensions 为 1,否则为 0。 注意：LabVIEW 不支持超过 1D 的 waveform 数组！
Digital Data	如果参数为 digital data 数组，则指定 Dimensions 为 1,否则为 0。 注意：LabVIEW 不支持超过 1D 的 digital Data 数组！
ActiveX	Data Type 下拉列表框有以下选择: ● ActiveX Variant Pointer 传递一个指向 ActiveX 数据的指针 ● IDispatch* Pointer 传递一个指向 ActiveX 自动化服务器 IDispatch 接口的指针 ● IUnknown Pointer 传递一个指向 ActiveX 自动化服务器 IUnknown 接口的指针
Adapt to Type	用来传递 LabVIEW 独有的数据类型给 DLL。

表 6-2 CFN 输入参数的类型

有时可能在 CFN 配置对话框中并不能找到要传递给它的参数类型，在这种情况下可以通过下面方法来解决。如果参数不含指针，则可以通过 **Flatten to String** 函数（  ）将参数转换为字符串，并将此字符串指针传递给函数。还有其它一些技巧请参见 NI 手册。

设定后的最终结果显示在“**Function Prototype**”文本框中，在确认前，可以在此检查设定是否正确。如果不正确可以在此修改设定。

网址：http://www.microvt.com
Email: hightechyoung@yahoo.com
作者：HighTech Young

6.3.2. 调用自己开发 DLL 中的函数

开发人员可以在 LabVIEW 中指定 DLL 函数的原型，然后在外部 IDE 中完成代码并编译生成.dll 文件以供项目使用。

下面就以一个简单的求数组求和的项目为例来说明这种开发过程。

1. 在 LabVIEW 中创建 DLL 函数原型。
   a) 在 LabVIEW 的 diagram 面板上添加一个 CFN 并通过其右键菜单打开 CFN 的配置对话框；
   b) 使“**Library Name or Path**”输入框为空；
   c) 指定函数名“**Function Name**”和调用方式“**Calling Conventions**”分别为 **add_num** 和 **C**；
   d) 重命名返回参数的名称为“**error**”，并指定其类型为 **Numeric** 的 **Signed 32-bit Integer**；
   e) 用**“Add a Parameter After”**按钮添加第一个参数 p，指定其类型为 **Array** 的 **4-byte Single** 并设定 **Array Format** 为 **Array Data Pointer**；
   f) 用**“Add a Parameter After”**按钮添加第二个参数 size，指定其类型为为 **Numeric** 的 **Signed 32-bit Integer** 并设置参数传递方式为 **Value**；
   g) 用**“Add a Parameter After”**按钮添加第三个参数 sum，指定其类型为为 **Numeric** 的 **4-byte Single** 并设置参数传递方式为 **Pointer to Value**；
   h) 至此，函数的原型应如下所示（图 6-14）：
   long add_num(float *p, long size, float *sum);

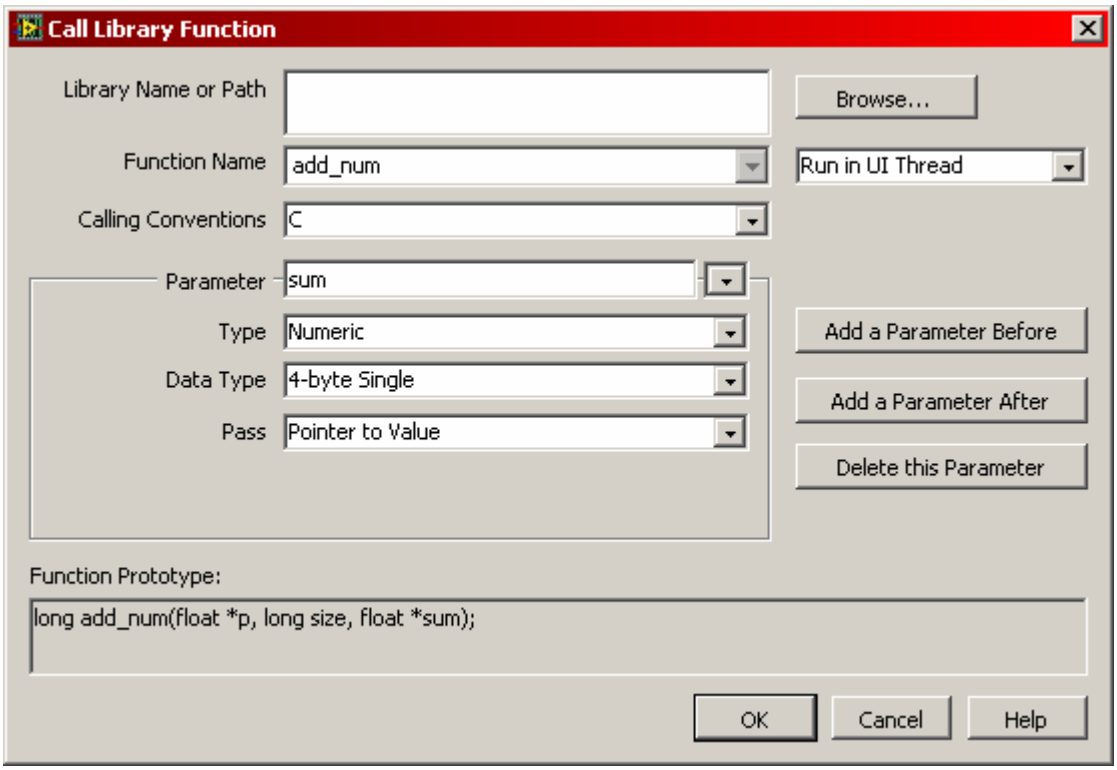
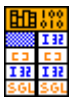


图 6-14 CFN 的配置结果

   i) 确定后会发现 CFN 根据配置自动进行了更新更新后的情况如右图示。 

2. 生成.C 或.C++文件，完成实现函数功能的代码并为函数添加 DLL 导出声明；
   在 CFN 节点上通过右键菜单选择“**Create .C File...**”生成 mydll.c 文件，其内容如下：

```
/* Call Library source file */
```

网址：http://www.microvt.com
Email: hightechyoung@yahoo.com
作者：HighTech Young

```
#include "extcode.h"

long add_num(float p[], long size, float *sum);
long add_num(float p[], long size, float *sum)
    {
    /* Insert code here */
    }
```

将以下代码插入到/* Insert code here */句之后实现函数的功能。

```
int i;
float tmpSum = 0;
if(p != NULL)
    {
        for(i=0; i < size; i++)
        tmpSum = tmpSum + p[i];
    }
else
    return (1);
* sum = tmpSum;
return (0);
```

在完成实现函数功能的代码后，还必须为函数添加导出声明以便能在 LabVIEW 中使用这些函数。C/C++声名导出函数的关键字是**_declspec (dllexport)**，使用该关键字可以代替模块定义文件。对于此处的例子来说，只要在函数声明和定义部分添加关键字即可。最终代码如下：

```
/* Call Library source file */
#include "extcode.h"

_declspec (dllexport) long add_num(float p[], long size, float *sum);
_declspec (dllexport) long add_num(float p[], long size, float *sum)
{
    /* Insert code here */
    int i;
    float tmpSum = 0;
    if(p != NULL)
        {
            for(i=0; i < size; i++)
            tmpSum = tmpSum + p[i];
        }
    else
        return (1);
    * sum = tmpSum;
    return (0);
}
```

3. 在外部 IDE（以 VC++为例）中创建 DLL 项目并编译生成.dll 文件。

用 VC++ 6.0 进行编译生成.dll 文件的步骤如下：

a) 在 VC++中创建一个 DLL 项目，如果在 DLL 中没有使用 MFC 就选择创建“**Win32**

网址：http://www.microvt.com
Email: hightechyoung@yahoo.com
作者：HighTech Young

Dynamic-Link Library”，否则选择“**MFC AppWizard(dll)**”，对此例子来说选择前者。选定后进入下一步选择创建一个空的 DLL 项目。

b) 通过 **Project»Add to Project»Files** 添加 mydll.c 到创建的 mydll 项目之中

c) 通过 **Project»Settings** 打开项目配置对话框，选择 **C/C++**选项卡。

d) 配置项目的 **All Configurations**。选择 **Settings For** 下拉列表框中的 **All Configurations**，选择 **Category** 下拉列表框中的 **Code Generation**，最后设置 **Struct member alignment** 为 **1 Byte**。配置结果如图 6-15。

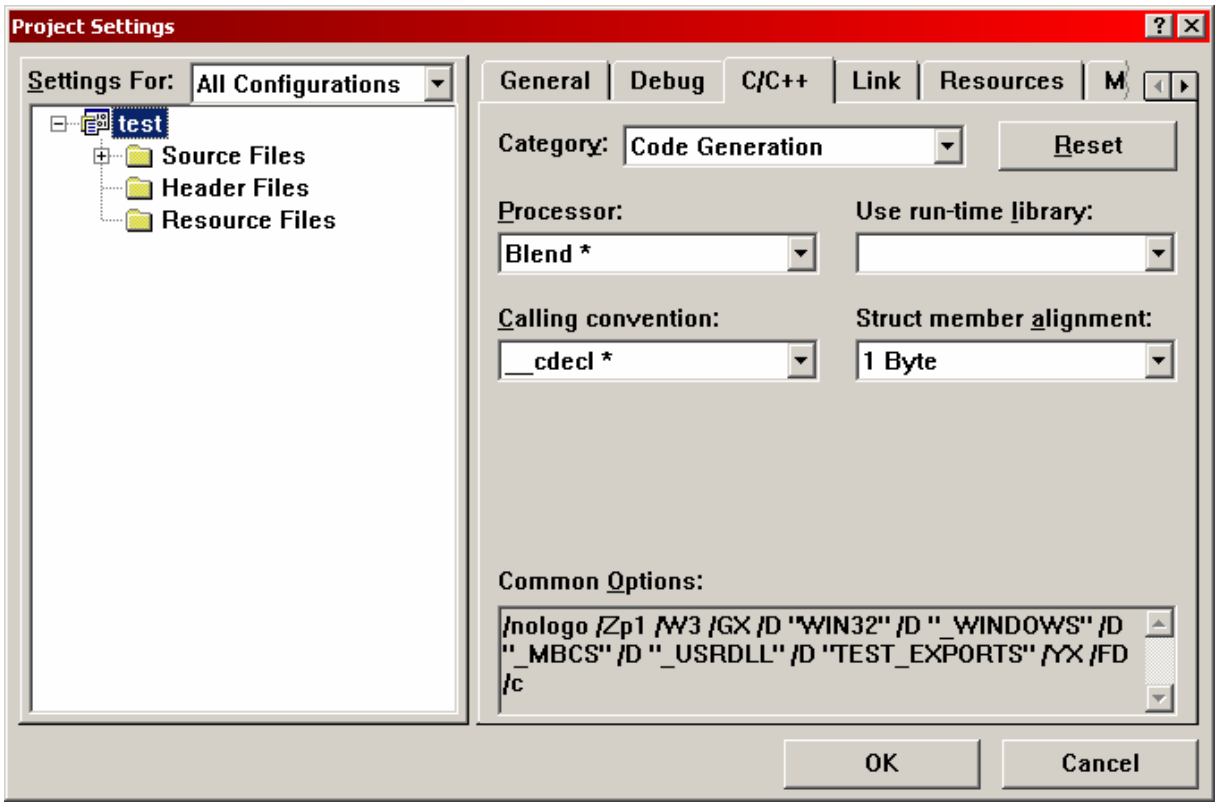


图 6-15 配置项目的 **All Configurations**

e) 配置项目的 **Release** 版本。选择 **Settings For** 下拉列表框中的 **Win32 Release**，选择 **Category** 下拉列表框中的 **Code Generation**，最后从 **Use run-time library** 下拉列表框中选择 **Multithreaded DLL**。配置结果如图 6-16。

f) 配置项目的 **Debug** 版本。选择 **Settings For** 下拉列表框中的 **Win32 Debug**，选择 **Category** 下拉列表框中的 **Code Generation**，最后从 **Use run-time library** 下拉列表框中选择 **Debug Multithreaded DLL**。配置结果如图 6-17。

网址：http://www.microvt.com
Email: hightechyoung@yahoo.com
作者：HighTech Young

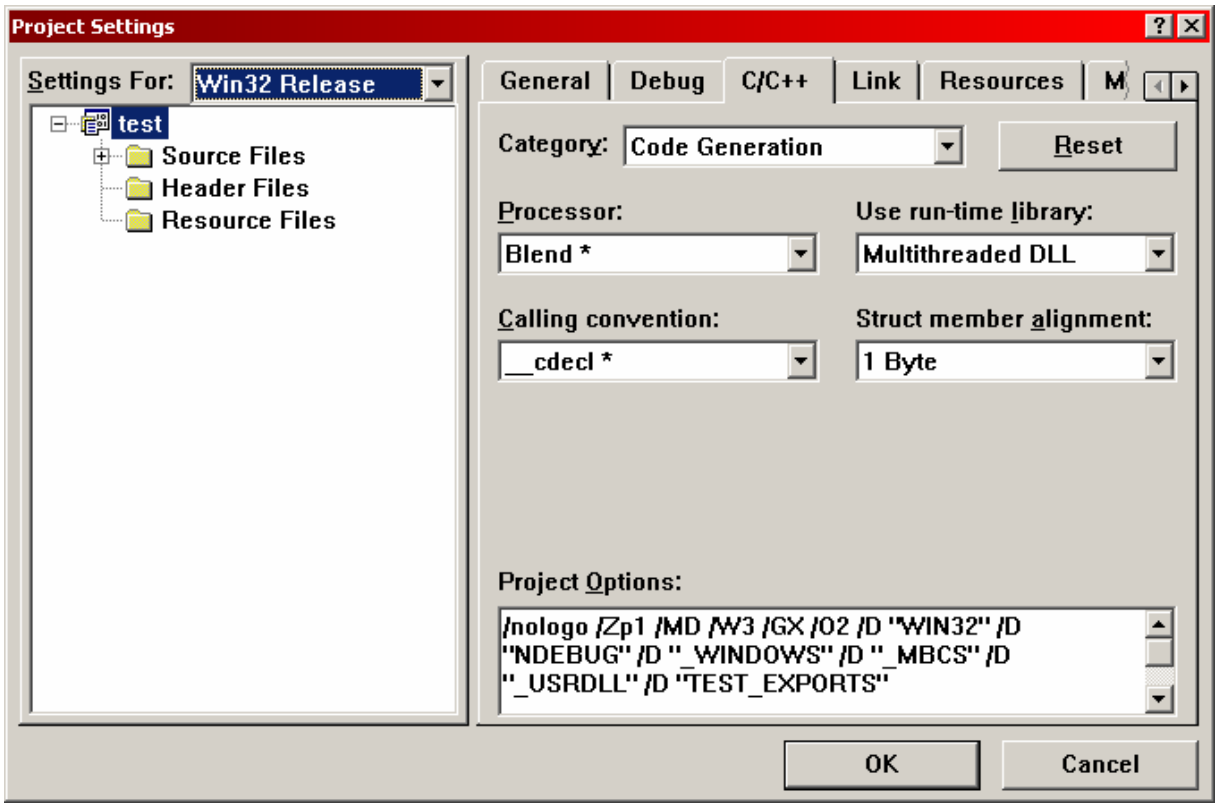


图 6-16 配置项目的 **Release** 版本

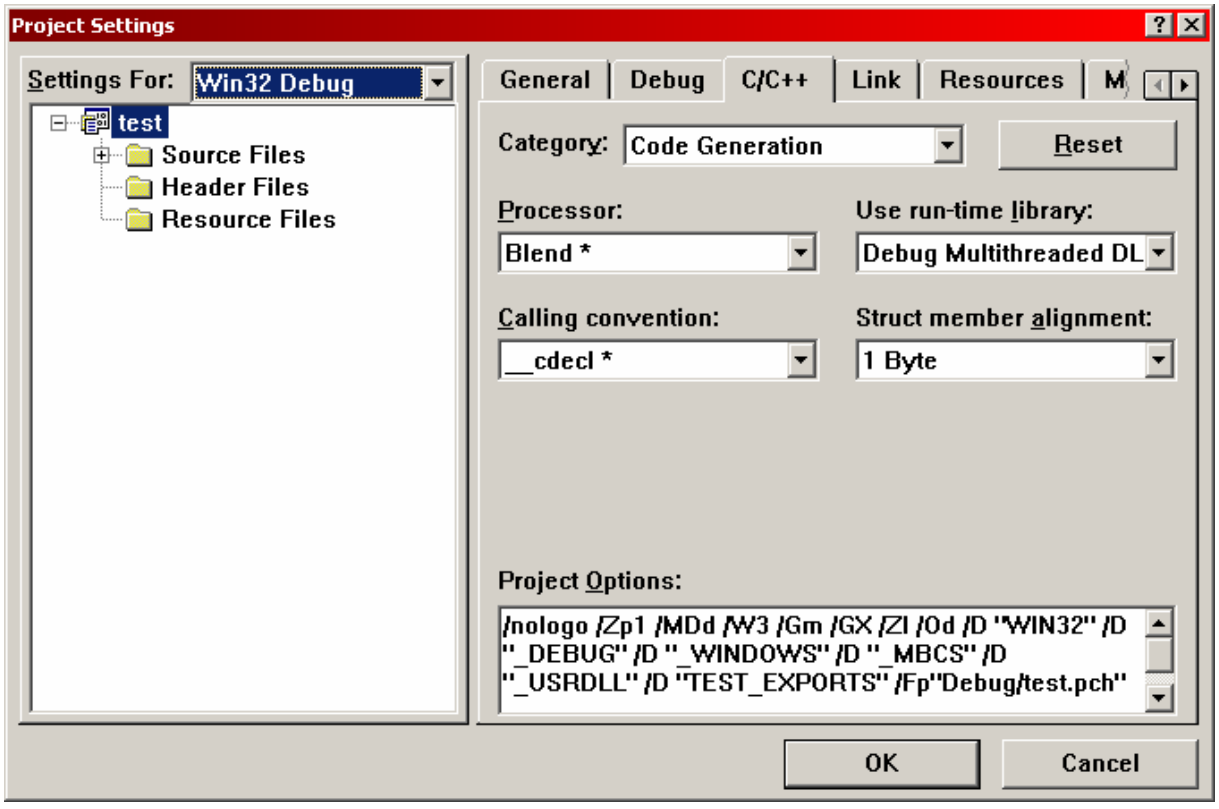


图 6-17 配置项目的 **Debug** 版本

网址：http://www.microvt.com
Email: hightechyoung@yahoo.com
作者：HighTech Young

4. 在 LabVIEW 项目中调用.dll 中的函数。

创建如图 6-18 所示的 VI，其中 Array 为 **Representation»Single Precision** 类型的数组，Sum 为 **Representation»Single Precision** 类型的 **Indicator**，error 为 **Representation»Long** 类型的 **Indicator**。运行后可以看到对数组求和的结果。

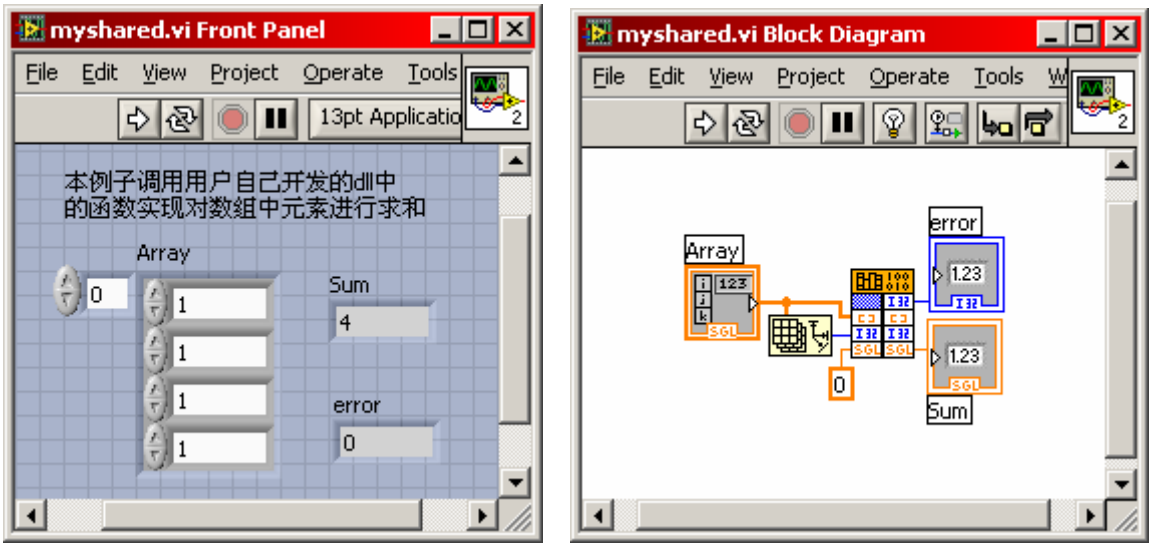


图 6-18 调用 DLL 中函数的 VI

6.3.3. 调用 Win32 API

在 LabVIEW 可以通过 CFN 调用 Win32 API 或者是硬件驱动供应商提供的驱动程序。通过调用这些函数用户能利用与操作系统进行交互或者利用第三方供应商提供的函数来扩展自己项目的功能。以下章节以调用 Win32 API 中 **MessageBox** 函数来实现一个同时带 **Abort**, **Retry**, **Ignore** 按钮和一个警告图标的消息对话框为例，叙述在 LabVIEW 中利用 CFN 调用 Win32 API 的方法。

要在 LabVIEW 项目中调用 Win32 API，首先要从 Microsoft 提供的相关开发文档中找到所要调用 API 函数的原型和其所在.dll 文件的名字。要查找这些信息，可以参考 Microsoft Software Development Kit (SDK)或 MSDN 的相关文档。从文档中找到 **MessageBox** 函数的原型和相关的一些信息如下。

```
int MessageBox(HWND hWnd,              // 父窗口的句柄
               LPCTSTR lpText,         // 消息窗中显示的文本
               LPCTSTR lpCaption,      // 消息窗的标题
               UINT uType);            // 消息窗的类型

----Requirements:----
Windows: Requires Windows 98 or later.
Windows CE: Requires version 1.0 or later.
Header: Declared in winuser.h.
Import Library: Use user32.lib.
Unicode: Implemented as Unicode and ANSI versions on Windows.
```

找到这些信息后，就可以进行调用API函数的的第二步：将参数类型映射为LabVIEW支持的数据类型。Win32 API函数用了多种非标准C的数据类型，这给在LabVIEW中调用API带来了一点麻烦。但是幸运的是，API函数所用的这些非标准数据类型只仅仅是标准C数据类型的别名。例如上面例子中**MessageBox**函数使用的参数类型与标准C的参数类型有以下映射关系：

HWND = **int * ***

LPCTSTR = **const char ***

网址：http://www.microvt.com
Email: hightechyoung@yahoo.com
作者：HighTech Young

UINT = unsigned int

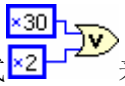
可直接将LPCTSTR和UINT映射为LabVIEW支持的类型。LPCTSTR与C语言字符串等价，UINT等价于LabVIEW的U32数据类型。但是映射**HWND**稍微有点麻烦。**HWND**是指向整型数指针的指针。观察**MessageBox**函数中**HWND**的作用，可以看出此参数只仅仅用来标识消息窗父窗口的句柄，因此，我们没有必要知道句柄本身的值，而只要关注**HWND**参数本身，由于**HWND**是指向整型数指针的指针，因此可以将它看作32位无符号整型量（U32）。在Windows SDK中有很多以H开头的量（句柄），一般说来，在开发过程中，都可以将它看作32位无符号整型量（U32）而不必关心它本身的值。如果不能确定API函数参数数据类型所对应的标准C语言数据类型，可以在头文件**windef.h**中查找相应的**#define**或**typedef**定义。

有了这些等价转换，还要关注API函数中常量与LabVIEW中数据类型或表示方法的映射。在Visual Studio中，编程人员一般不直接使用常量的值，而是使用与预定义的常量名，但是在LabVIEW中却必须使用实际的值。例如对于**MessageBox**函数来说，代表消息窗类型的参数**uType**可以使用下表中几个常量。

常量	说明
MB_ABORTRETRYIGNORE	带有Abort, Retry, Ignore按钮的消息窗
MB_CANCELTRYCONTINUE	Win2000上可替代MB_ABORTRETRYIGNORE量，具有Cancel, Try Again, Continue按钮的消息窗
MB_HELP	Win2K/XP上为消息窗添加**Help**按钮并在用户按下此按钮时发送WM_HELP消息给父窗口的的消息窗
MB_OK	默认值只有一个OK按钮的消息窗
MB_ICONWARNING	带有警告图标的消息窗

通过在SDK文档中查找到的信息（**Header:** Declared in winuser.h.）知道这些常量在头文件**winuser.h**中有定义。通过该头文件可以找到相关定义如下。

```
...
#ifdef UNICODE
  #define MessageBox MessageBoxW
#else
  #define MessageBox MessageBoxA
#endif
...
#define    MB_OK                    0x00000000L
#define    MB_ABORTRETRYIGNORE      0x00000002L
#define    MB_ICONWARNING           MB_ICONEXCLAMATION
#define    MB_ICONEXCLAMATION       0x00000030L
...
```

在SDK文档中，常量通常取位域中的某一位作为其值。位域是通常是指一个每一位控制某种属性的单个整型量。通过或运算（|）可以将多个常量组合起来达到多种需要的效果。从逻辑运算角度来讲，进行或运算等价于选择了位域中的多个控制位。例如可以通将**MB_ABORTRETRYIGNORE**和**MB_ICONEXCLAMATION**进行或运算后的值传给**uType**可以创建一个同时带**Abort**, **Retry**, **Ignore**和一个警告图标的消息窗。**MB_ABORTRETRYIGNORE**的值为**0x02**，**MB_ICONEXCLAMATION**的值为**0x30**, 经过或运算后的值为**0x32**。在Visual Studio开发中可以直接将“**MB_ABORTRETRYIGNORE | MB_ICONEXCLAMATION**”常量或运算表达式作为参数**uType**传递给**MessageBox**函数来实现一个同时带**Abort**, **Retry**, **Ignore**按钮和一个警告图标的消息对话框，但是LabVIEW却不支持这种形式的参数传递。因此必须将常量表达式转换为LabVIEW支持的形式。这可以通过表达式 

来实现。注意其中数字均为十六进制数。

最后还要根据开发项目是ANSI项目还是Unicode项目，并参照SDK文档选择合适的函数。一般来说大多数LabVIEW项目均为ANSI项目，因为大多数LabVIEW的字符串均为ANSI标准字符串。对于此处的例子

网址：http://www.microvt.com
Email: hightechyoung@yahoo.com
作者：HighTech Young

来说，通过参看从SDK文档中获取的信息可知**MessageBox**函数有ANSI和Unicode两种实现（**Unicode:** Implemented as Unicode and ANSI versions on Windows.）。查看头文件**winuser.h**中内容（参见上页表）可知对于ANSI类型的项目应选择函数名**MessageBoxA**。实际上在库文件user32.dll（在windows中相对于静态库文件一般都存在一个同名的动态库文件）中，只存在函数**MessageBoxA**和**MessageBoxW**并没有**MessageBox**函数的实体。

综上所述，要调用API实现一个同时带**Abort**, **Retry**, **Ignore**按钮和一个警告图标的消息对话框可对CFN节点完成如图6-19所示的配置。

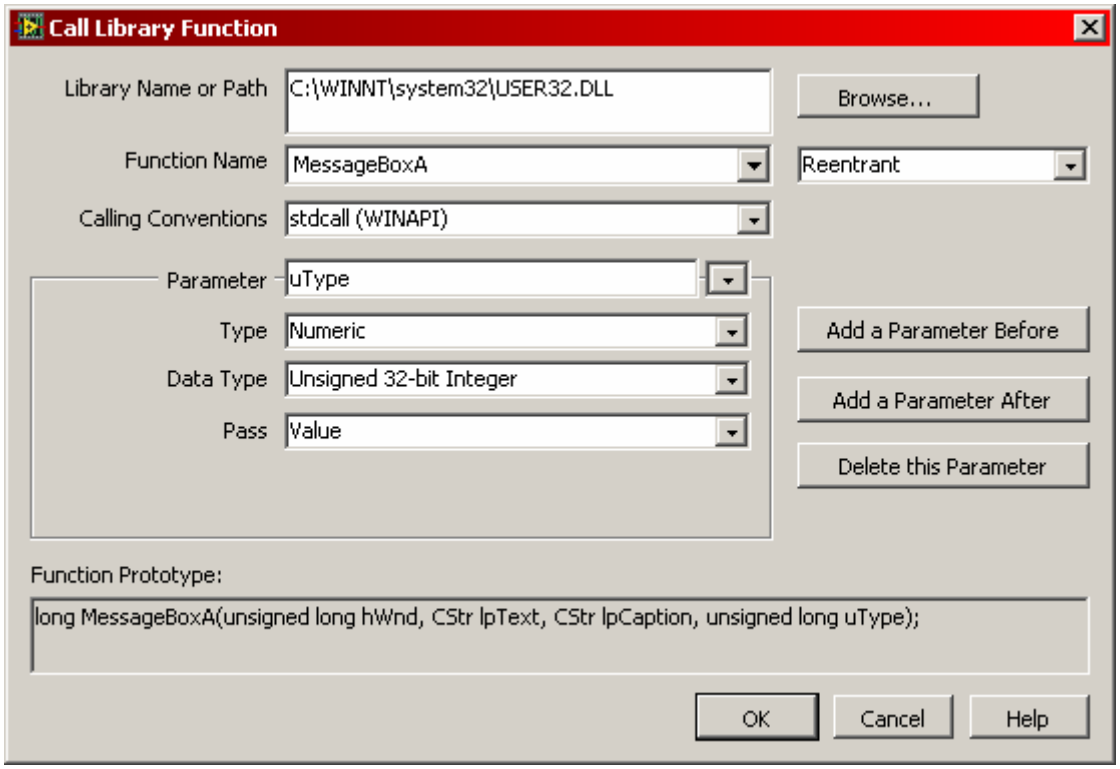


图 6-19 CFN 节点的配置

完成后VI的后面板和运行结果可能如图6-20所示。

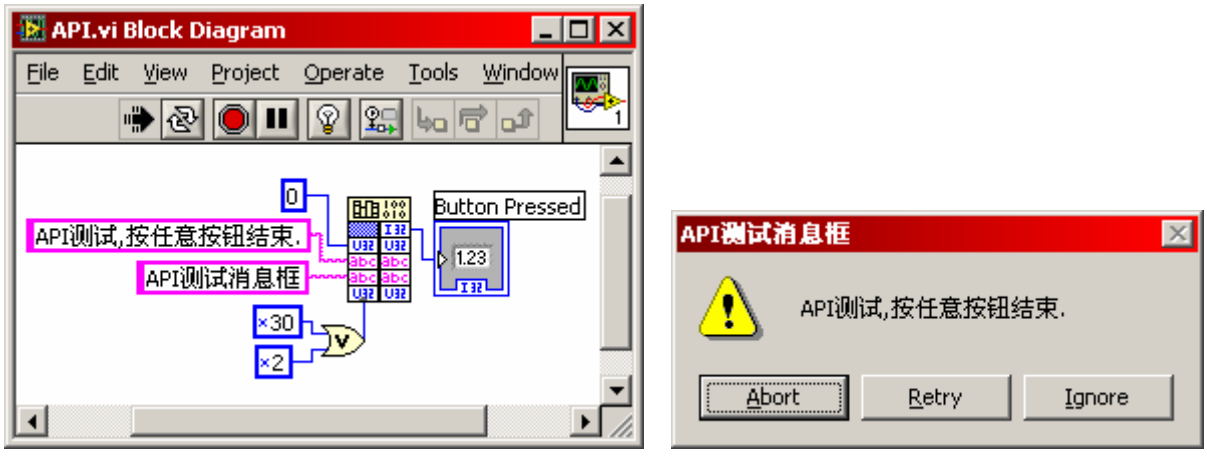


图 6-20 VI 的后面板和运行结果