第 5 章 动态链接库封装和调用

CHAPTER 5

LabVIEW 项目开发时，经常会遇到仪器设备控制问题，对于 NI 提供的仪器设备，可以通过安装相关的开发工具包来解决；对于非 NI 提供的设备，可以通过调用动态链接库(Dynamic Link Library，DLL)的方式来解决。本章将以 RTL-SDR 的 LabVIEW 接口函数为例，介绍 LabVIEW 中动态链接库的封装和调用方法。

5.1 RTL-SDR 接口函数的封装

LabVIEW 通过调用 RTL-SDR 的接口函数实现对 RTL-SDR 的控制。常用的接口函数有 open 函数、set sample rate 函数、set center freq 函数、read sync 函数和 close 函数。3.2.2 节详细介绍了 RTL-SDR 常用接口函数。3.2.3 节通过数据采集实例介绍了 RTL-SDR 接口函数的使用方法。

打开 RTL-SDR 接口函数的程序框图，可以看到，LabVIEW 是通过调用库函数(Call Library Function，CLF)模块调用动态链接库 rtlsdr.dll 的方式实现设备控制。下面将介绍动态链接库和 LabVIEW 接口函数的封装方法。

5.1.1 动态链接库简介

动态链接库(DLL)是 Windows 操作系统实现共享函数库的一种方式。在 Windows 操作系统中使用 DLL，能够使应用程序代码变得更简洁，计算机内存资源的使用效率变得更高。

RTL-SDR 的应用程序接口(Application Programming Interface，API)函数是采用 C 语言编写的。通过调用 RTL-SDR 的动态链接库 rtlsdr.dll，上层 SDRsharp、LabVIEW、MATLAB/Simulink 和 GNURadio 等应用软件就可以控制 RTL-SDR 设备，如图 5-1 所示。

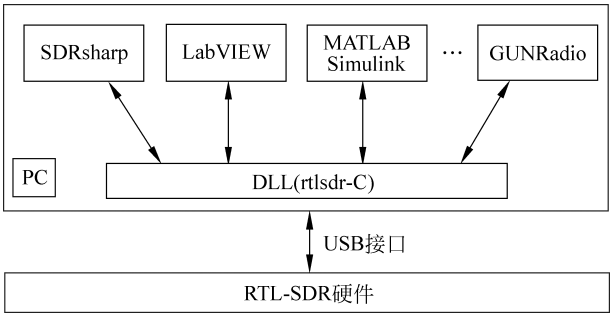


图 5-1 RTL-SDR 接口函数

动态链接库 rtlsdr.dll 实现了 RTL-SDR 的所有接口函数。RTL-SDR 源程序中的头文件 rtl-sdr.h 对所有的接口函数作了定义。

```
#ifndef __RTL_SDR_H
#define __RTL_SDR_H
#ifdef __cplusplus
extern "C" {
#endif
#include <stdint.h>
#include <rtl-sdr_export.h>
typedef struct rtlsdr_dev rtlsdr_dev_t;
RTLSDR_API uint32_t rtlsdr_get_device_count(void);
RTLSDR_API const char* rtlsdr_get_device_name (uint32_t index);
RTLSDR_API int rtlsdr_get_device_usb_strings (uint32_t index, char * manufact, char *
product, char * serial);
RTLSDR_API int rtlsdr_get_index_by_serial (const char * serial);
RTLSDR_API int rtlsdr_open (rtlsdr_dev_t ** dev, uint32_t index);
RTLSDR_API int rtlsdr_close (rtlsdr_dev_t * dev);
RTLSDR_API int rtlsdr_set_xtal_freq (rtlsdr_dev_t * dev, uint32_t rtl_freq, uint32_t tuner_
freq);
RTLSDR_API int rtlsdr_get_xtal_freq (rtlsdr_dev_t * dev, uint32_t * rtl_freq, uint32_t *
tuner_freq);
RTLSDR_API int rtlsdr_get_usb_strings (rtlsdr_dev_t * dev, char * manufact, char * product,
char * serial);
RTLSDR_API int rtlsdr_write_eeprom (rtlsdr_dev_t * dev, uint8_t * data, uint8_t offset,
uint16_t len);
RTLSDR_API int rtlsdr_read_eeprom (rtlsdr_dev_t * dev, uint8_t * data, uint8_t offset, uint16
_t len);
RTLSDR_API int rtlsdr_set_center_freq (rtlsdr_dev_t * dev, uint32_t freq);
RTLSDR_API uint32_t rtlsdr_get_center_freq (rtlsdr_dev_t * dev);
RTLSDR_API int rtlsdr_set_freq_correction (rtlsdr_dev_t * dev, int ppm);
RTLSDR_API int rtlsdr_get_freq_correction (rtlsdr_dev_t * dev);
enum rtlsdr_tuner {
    RTLSDR_TUNER_UNKNOWN = 0,
    RTLSDR_TUNER_E4000,
    RTLSDR_TUNER_FC0012,
    RTLSDR_TUNER_FC0013,
    RTLSDR_TUNER_FC2580,
    RTLSDR_TUNER_R820T,
    RTLSDR_TUNER_R828D
};
RTLSDR_API enum rtlsdr_tuner rtlsdr_get_tuner_type (rtlsdr_dev_t * dev);
RTLSDR_API int rtlsdr_get_tuner_gains (rtlsdr_dev_t * dev, int * gains);
RTLSDR_API int rtlsdr_set_tuner_gain (rtlsdr_dev_t * dev, int gain);
RTLSDR_API int rtlsdr_set_tuner_bandwidth (rtlsdr_dev_t * dev, uint32_t bw);
RTLSDR_API int rtlsdr_get_tuner_gain (rtlsdr_dev_t * dev);
RTLSDR_API int rtlsdr_set_tuner_if_gain (rtlsdr_dev_t * dev, int stage, int gain);
RTLSDR_API int rtlsdr_set_tuner_gain_mode (rtlsdr_dev_t * dev, int manual);
RTLSDR_API int rtlsdr_set_sample_rate (rtlsdr_dev_t * dev, uint32_t rate);
```

```
RTLSDR_API uint32_t rtlsdr_get_sample_rate (rtlsdr_dev_t * dev);
RTLSDR_API int rtlsdr_set_testmode (rtlsdr_dev_t * dev, int on);
RTLSDR_API int rtlsdr_set_agc_mode (rtlsdr_dev_t * dev, int on);
RTLSDR_API int rtlsdr_set_direct_sampling (rtlsdr_dev_t * dev, int on);
RTLSDR_API int rtlsdr_get_direct_sampling (rtlsdr_dev_t * dev);
RTLSDR_API int rtlsdr_set_offset_tuning (rtlsdr_dev_t * dev, int on);
RTLSDR_API int rtlsdr_get_offset_tuning (rtlsdr_dev_t * dev);
RTLSDR_API int rtlsdr_reset_buffer (rtlsdr_dev_t * dev);
RTLSDR_API int rtlsdr_read_sync (rtlsdr_dev_t * dev, void * buf, int len, int * n_read);
typedef void( * rtlsdr_read_async_cb_t) (unsigned char * buf, uint32_t len, void * ctx);
RTLSDR_API int rtlsdr_wait_async (rtlsdr_dev_t * dev, rtlsdr_read_async_cb_t cb, void * ctx);
RTLSDR_API int rtlsdr_read_async (rtlsdr_dev_t * dev, rtlsdr_read_async_cb_t cb, void * ctx,
uint32_t buf_num, uint32_t buf_len);
RTLSDR_API int rtlsdr_cancel_async (rtlsdr_dev_t * dev);
RTLSDR_API int rtlsdr_set_bias_tee (rtlsdr_dev_t * dev, int on);
RTLSDR_API int rtlsdr_set_bias_tee_gpio (rtlsdr_dev_t * dev, int gpio, int on);
#ifdef __cplusplus
}
#endif
#endif /* __RTL_SDR_H */
```

rtlsdr.h 头文件定义了 34 个接口函数。常用的接口函数有 10 个，如 rtlsdr_open(获取设备句柄)函数、rtlsdr_set_center_freq(设置中心频率)函数、rtlsdr_set_sample_rate(设置采样率)函数、rtlsdr_read_async(同步读取 I/Q 数据)函数、rtlsdr_close(关闭设备句柄)函数等。从头文件的函数声明中，可以查到函数输入参数和输出参数的名称和类型。

5.1.2　RTL-SDR 接口函数封装

LabVIEW 编程中，提供了一套调用动态链接库的方法。根据头文件提供的函数声明，就可以利用 CLF 模块将头文件中的接口函数封装成子 VI，接下来将以 RTLSDR_API int rtlsdr_open(rtlsdr_dev_t ** dev,uint32_t index)函数为例说明动态链接库的封装和调用过程。

(1) 准备两个文件，一个是 RTL-SDR 的库文件 rtlsdr.dll，另一个是头文件 RTL-SDR.h。需要特别注意的是，如果主机安装的 LabVIEW 是 32 位的，库文件 rtlsdr.dll 必须是 32 位的，否则 LabVIEW 将无法调用 rtlsdr.dll 中的函数。

(2) 新建一个 VI，在 Connectivity→Libraries & Executables 路径下找到 CLF 模块，如图 5-2 所示。通过 CLF 模块，就可以实现动态链接库的调用。

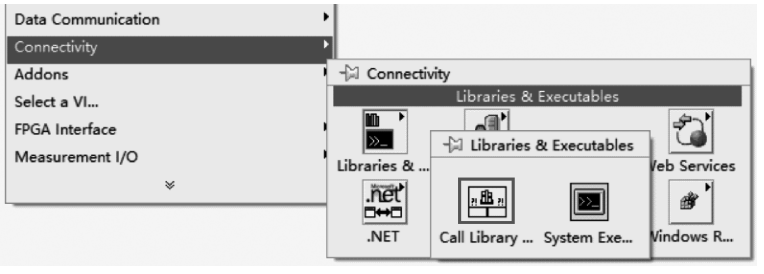


图 5-2　调用库函数(CLF)模块

(3) 在程序框图中创建一个 CLF 模块,双击该模块,就会弹出该模块的参数配置对话框,在这个对话框中,可以配置 DLL 文件路径、被调函数名、输入和输出参数等,如图 5-3 所示。

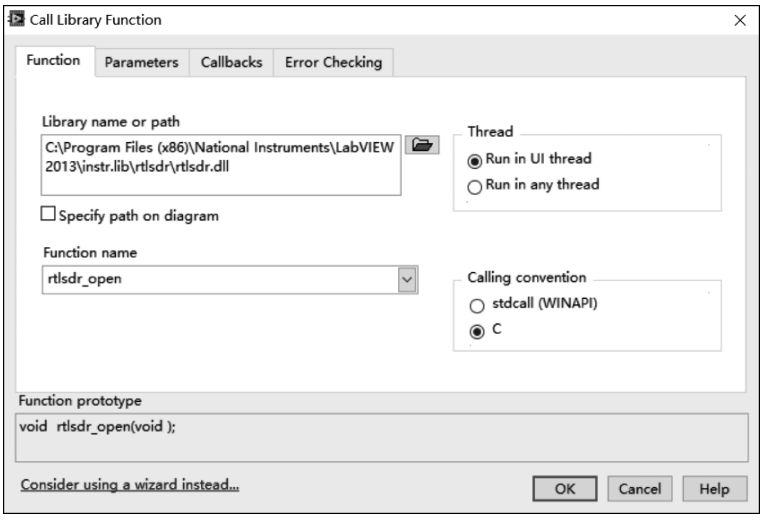


图 5-3 CLF 参数配置

在 Function 选项卡中,配置被调用函数的信息。在 Library name or path 文本框中设置 rtlsdr.dll 的路径(注意选择 rtlsdr.dll 文件的存放路径)。在 Function name 下拉列表中选择需要调用的接口函数,本例中选择 rtlsdr_open。在 Thread 选项中选择运行的线程范围,本例中默认选择 Run in UI thread。在 Calling convention 选项中选择被调用函数的调用约定,本例中默认选择 C。

在 Parameters 选项卡中,根据头文件中的函数声明对输入和输出参数进行设置。需要特别注意的是,为了正确设置这些参数,需要参考 rtl-sdr.h 头文件中的函数声明。例如,在 rtl-sdr.h 中,可以找到 rtlsdr_open 函数的输入输出参数以及它们的类型:

```
RTLSDR_API int rtlsdr_open (rtlsdr_dev_t ** dev, uint32_t index);
```

在 Parameters 选项卡中,默认创建了一个 return type 参数,如图 5-4 所示。根据头文件,rtlsdr_open 函数返回值是 int 类型,在图 5-4 的参数设置 Type 下拉列表中选择 Numeric 类型,在 Data type 下拉列表中选择 Signed 32-bit Integer 类型。

在 CLF 的 Parameters 选项卡中,还需要创建两个参数,一个是指向设备的句柄指针,另一个是设备索引。选中左侧列表中的 DevRefnum,单击页面中的 ⊕ 按钮,在 Current parameter 栏中设置参数,Name 可以自定义,如设置为 DevRefnum,Type 选择为 Numeric,Data type 选择为 Unsigned Pointer-sized Integer,Pass 选择为 Point to Value,如图 5-5 所示。

以同样的方式创建设备索引 index。注意 index 是一个输入参数。Name 设置为 index,Type 选择为 Numeric,Data type 选择为 Unsigned 32-bit Integer,Pass 选择为 Value。DevRefnum 和 index 均创建完成之后,在页面左下角的 Function prototype 区域可以看到

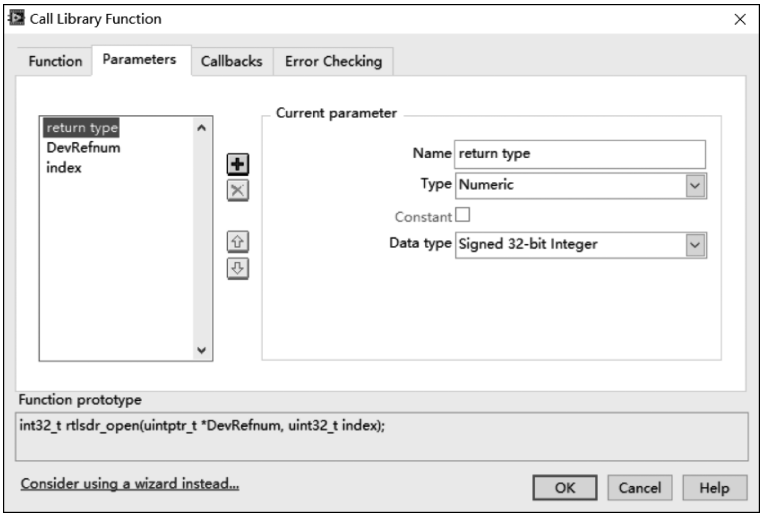


图 5-4　Parameters 选项卡

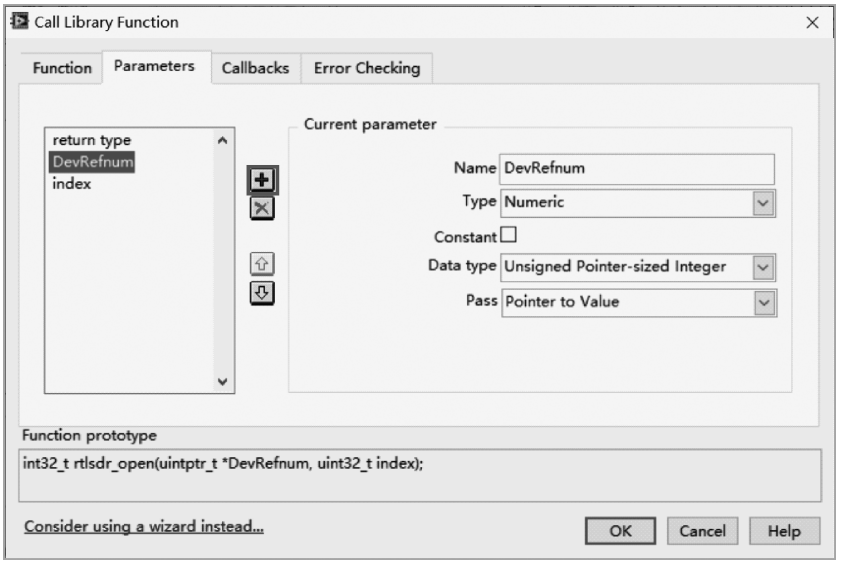


图 5-5　DevRefnum 配置

函数声明，此声明应当与头文件中的函数保持一致。

（4）CLF 模块配置完成之后，需要定义输入和输出端口，在 CLF 的输入和输出端口分别创建数值输入控件和显示控件，如图 5-6 所示。需要注意的是，在前面板中需要完成端口关联，才可以作为子 VI 被其他 VI 调用。

（5）按照同样的方法，可以对 rtl-sdr.h 头文件中的其他函数进行封装。将需要的函数都封装完成之后，可以将封装后的子 VI 打包成一个库文件，方便维护。执行 File→New 菜单命令，新建一个 Library，如图 5-7 所示。

然后在库文件名上右击，在弹出的菜单中选择 Add→File，选择已经封装好的 VI。更改库文件名，就完成了库文件的创建，如图 5-8 所示。需要注意的是，rtlsdr.dll 文件和 rtlsdr.lvlib

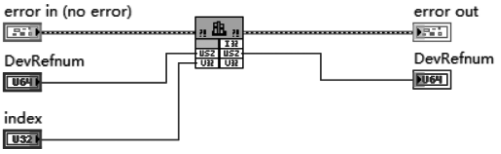


图 5-6　rtl_open 子 VI

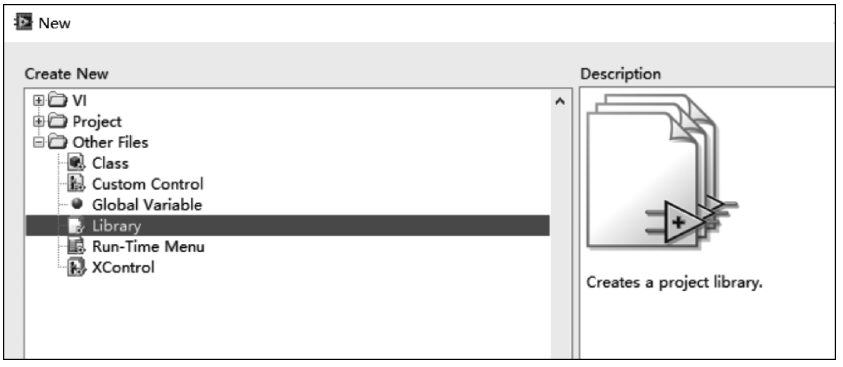


图 5-7　Library 创建

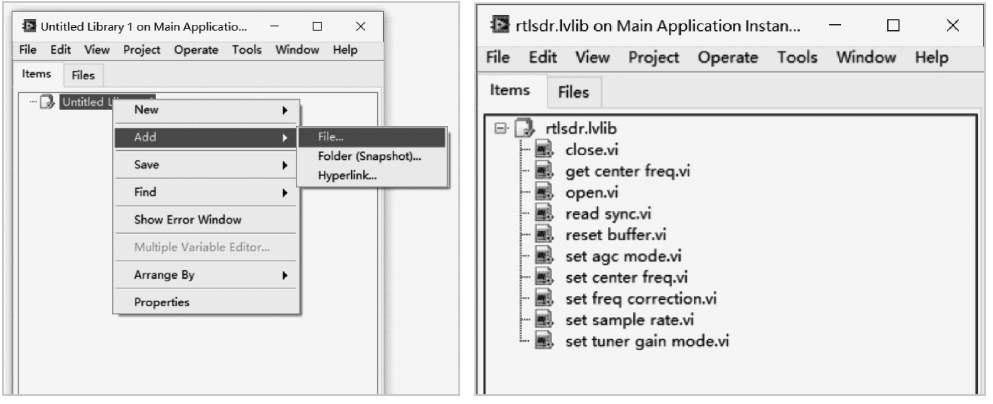


图 5-8　rtlsdr. lvlib 创建

一般放在同一个文件夹下。

最后将整个库文件夹复制到 LabVIEW 的\instr. lib 文件夹中，这样，在程序框图的函数选板中，就可以直接调用相应的子 VI，如图 5-9 所示。

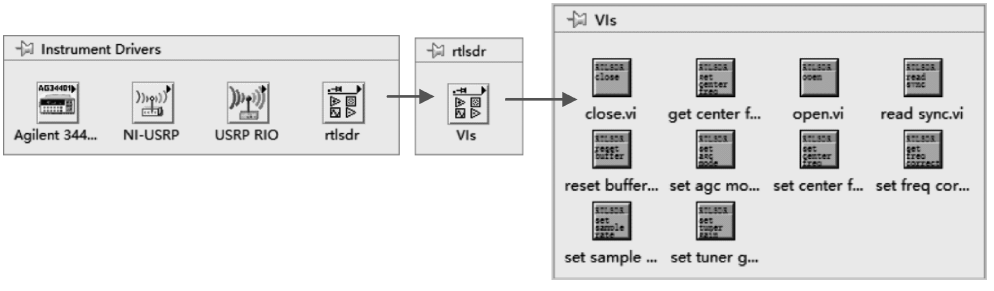


图 5-9　RTL-SDR 子 VI

5.2　导入共享库向导

手动封装动态链装库中的接口函数是比较麻烦的，尤其是在库文件中的函数比较多的时候。LabVIEW 提供了自动封装的方法——导入共享库向导。接下来，本节将以 rtlsdr. dll 为例，介绍基于导入共享库向导的封装方法。

5.2.1　导入前准备

在使用导入共享库向导之前，需要准备两个文件，一个是 RTL-SDR 的库文件 rtlsdr. dll，另一个是头文件 RTL-SDR. h。

5.2.2　导入共享库向导过程

(1) 执行 Tools(工具)→Import(导入)→Shared Library(. dll)...(共享库)菜单命令，如图 5-10 所示。

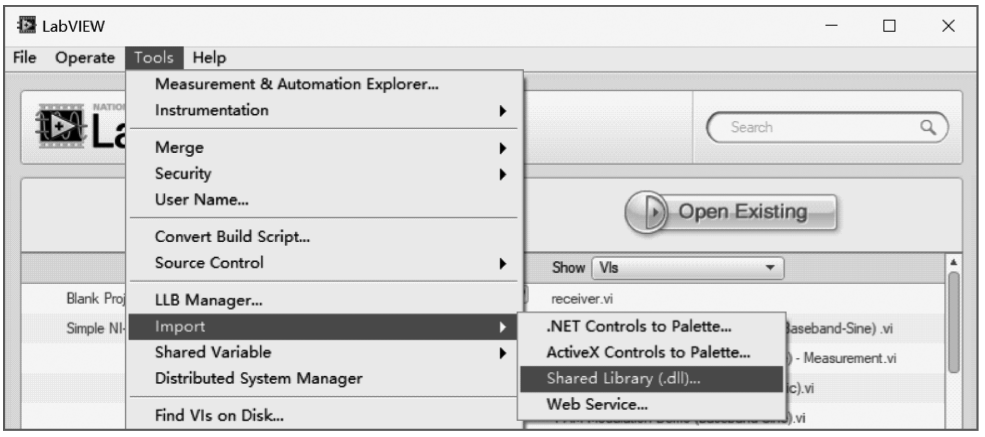


图 5-10　导入共享库向导

(2) 在弹出的对话框中选择 Create VIs for a shared library，如图 5-11 所示。如果需要对已经生成的 VI 进行更新，选择第 2 项 Update VIs for a shared library。

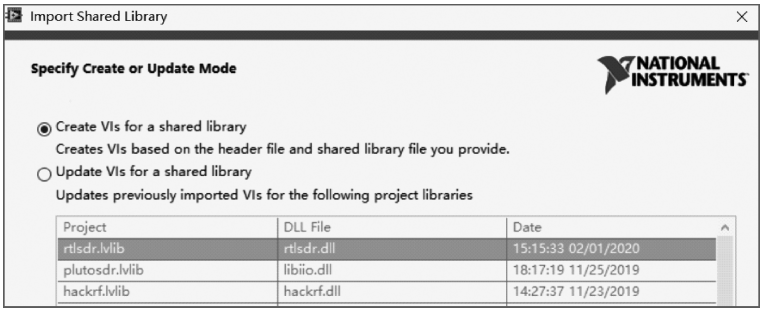


图 5-11　创建或更新模式

(3) 进入路径配置对话框,设置 rtlsdr.dll 的路径和头文件的路径。注意这里设置的头文件是 rtl-sdr.h,如图 5-12 所示。

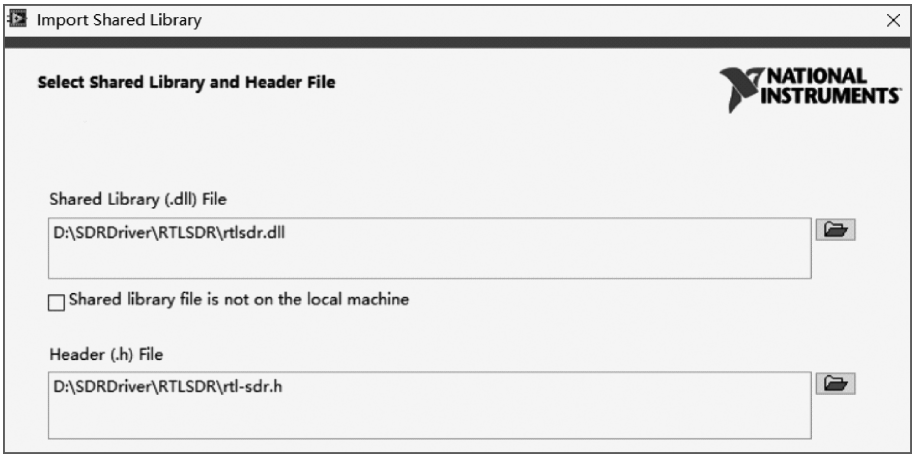


图 5-12 选择库文件和头文件

(4) 用记事本打开 rtl-sdr.h,可以看到头文件 rtl-sdr.h 还包含其他两个头文件:

```
#include <stdint.h>
#include <rtl-sdr_export.h>
```

在 Include 文件夹中,配置两个头文件 stdint.h 和 rtl-sdr_export.h(本书配套程序)的路径,如图 5-13 所示。

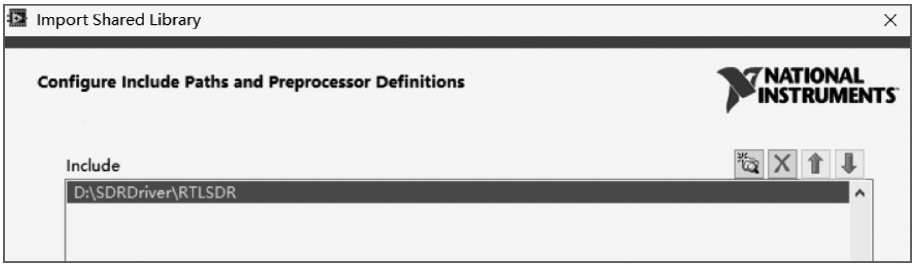


图 5-13 Include 文件夹

(5) 如果路径配置正确,共享库向导就会根据库文件和头文件生成一个函数列表。正常情况下,头文件中的 33 个接口函数能够被识别和封装,如图 5-14 所示。对于 FM 接收机,只会用到其中的少部分函数,如 rtlsdr_open()、rtlsdr_set_center_freq()、rtlsdr_set_sample_rate()函数等。

(6) 选定需要生成的函数之后,设置 LabVIEW 库函数的文件名和保存路径。本例中设置文件名为 rtlsdr,路径为 C 盘安装文件夹\user.lib 下,如图 5-15 所示。

(7) 在 Select Error Handing Mode 页面中,可以配置错误输出模式,如图 5-16 所示。有 3 种模式可以选择,本例中选择 Simple Error Handing。

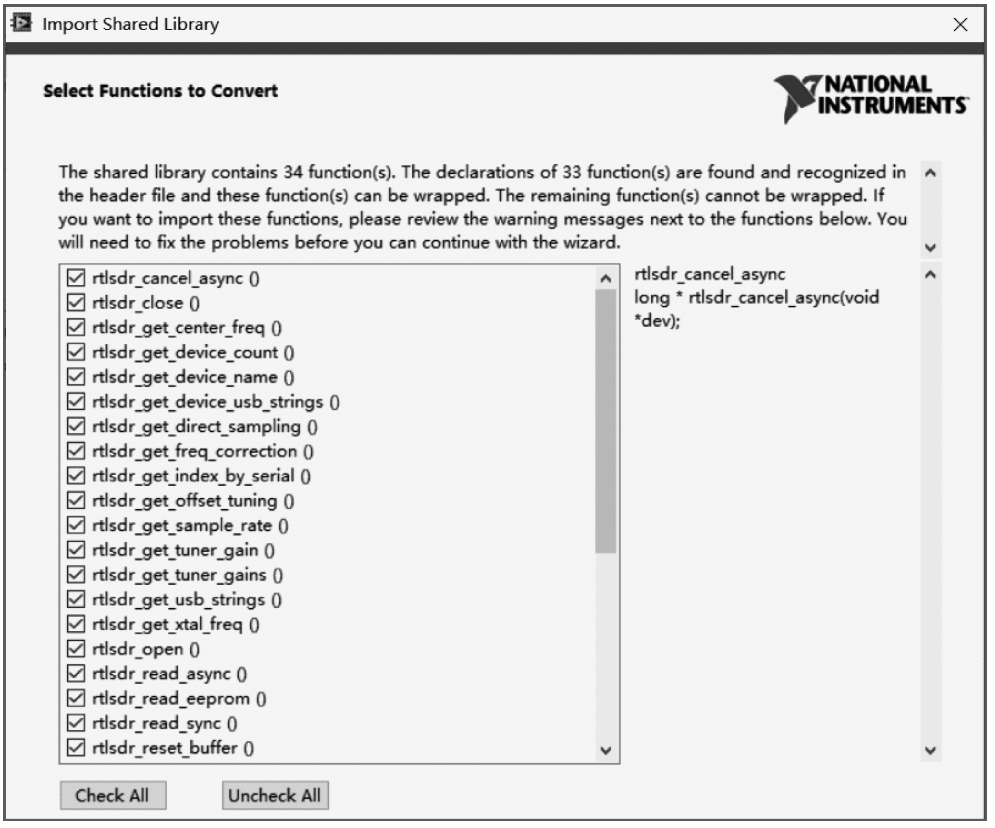


图 5-14　选择需要封装的函数

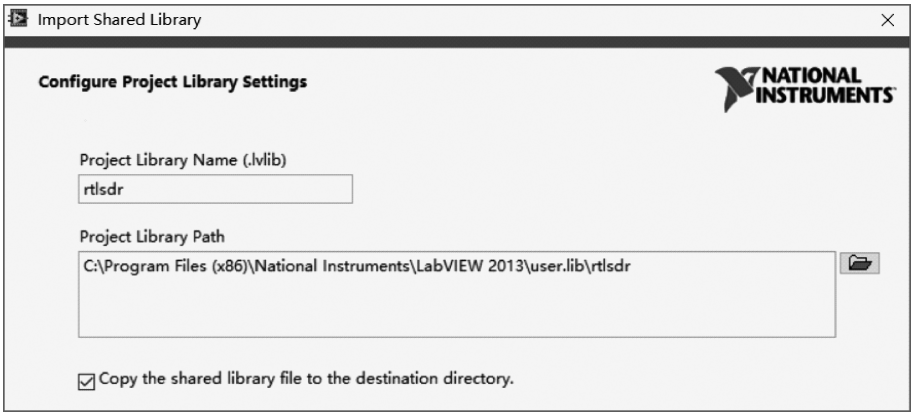


图 5-15　设置库文件名和保存路径

(8) 在 Configure VIs and Controls 页面中,可以对每个函数的输入和输出参数进行重配置。例如,选择 rtlsdr_set_center_freq()这个函数,在默认的情况下,Control Type 设置为 Numeric,Pass Type 设置为 Pass by Value,Representation 设置为 Unsigned Long,如图 5-17 所示。

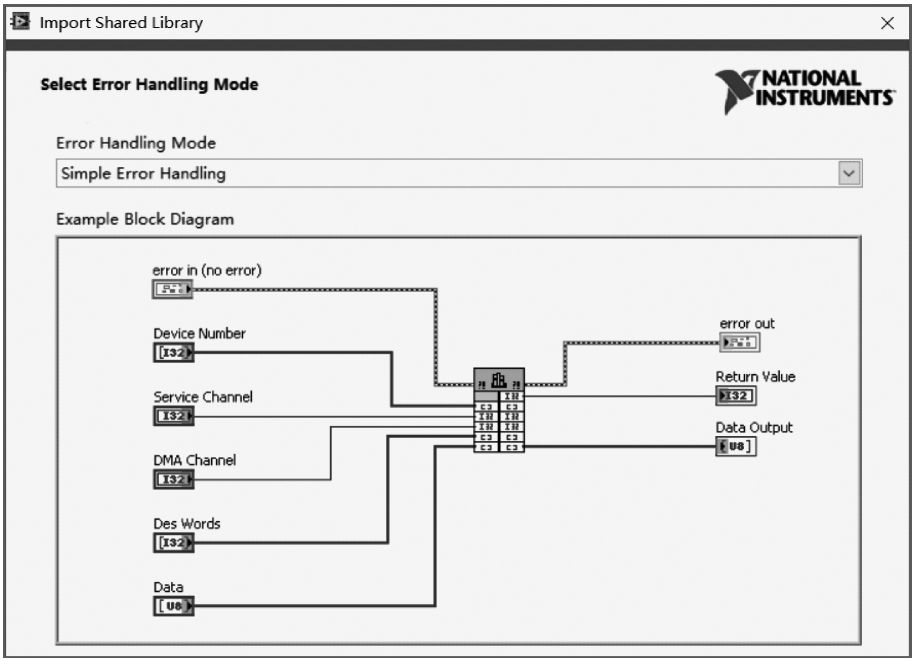


图 5-16　配置错误输出模式

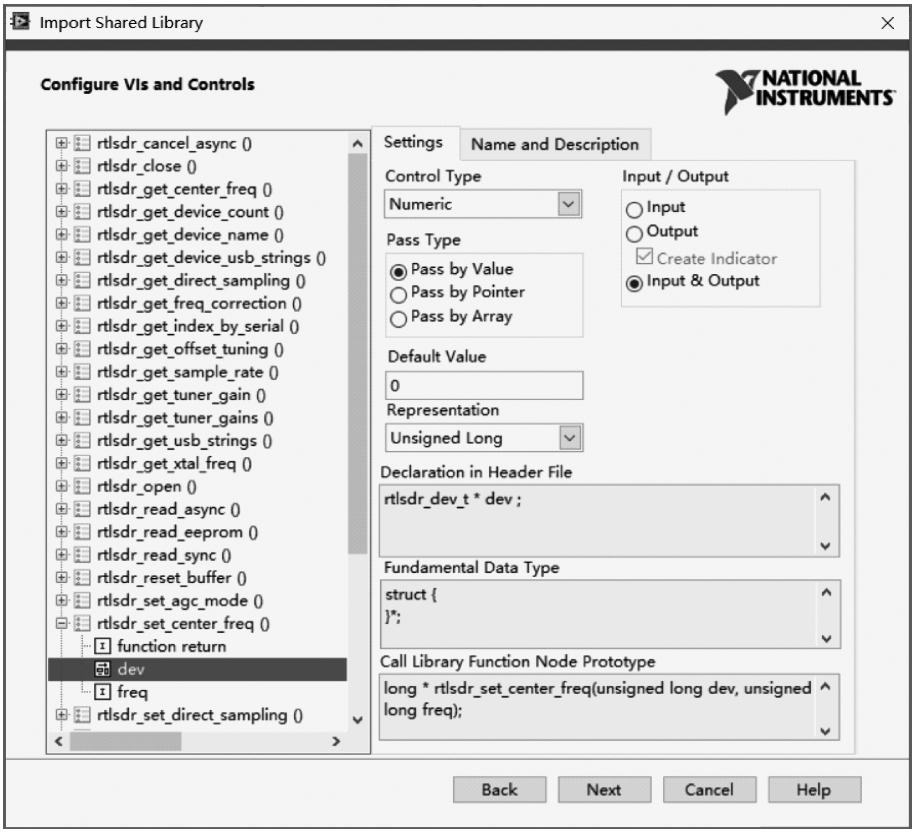


图 5-17　重新配置输入输出参数

（9）配置完成每个函数的输入和输出参数，单击 Next 按钮，就可以进行封装。等待几分钟，将返回封装后的 rtlsdr 库，如图 5-18 所示。

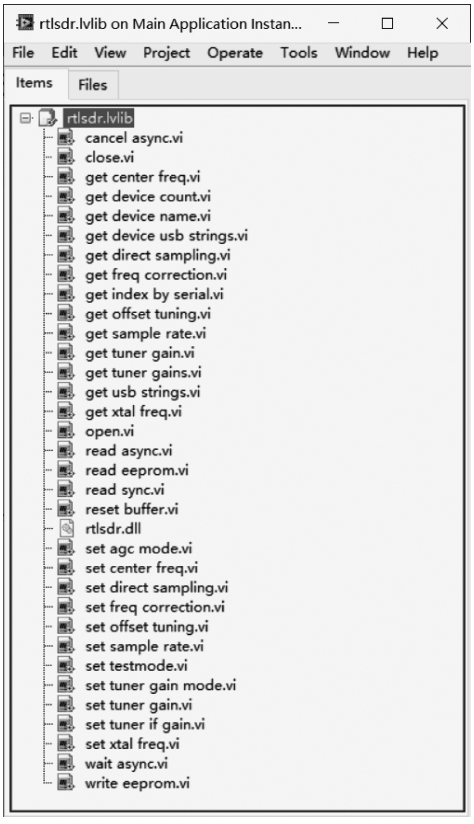


图 5-18　封装后的 rtlsdr 库

（10）检查接口函数封装的正确性。打开任意一个子 VI，如 set center freq. vi。进一步打开 CLF，可以看到其配置，如图 5-19 所示。

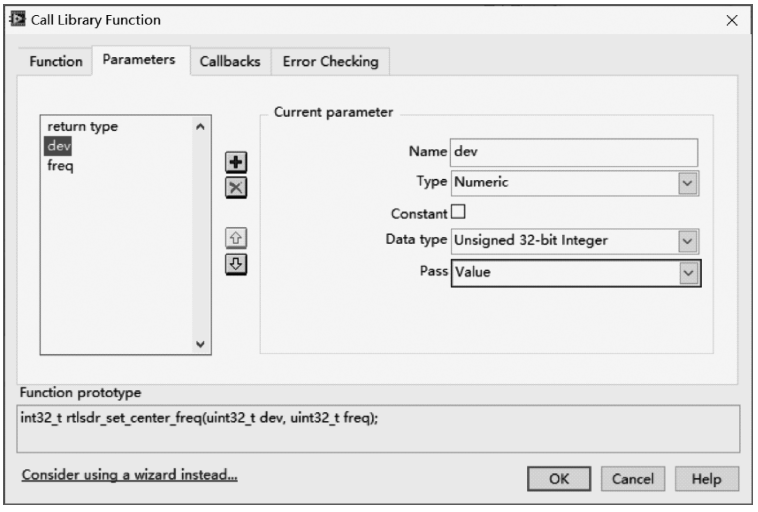


图 5-19　封装的正确性验证

5.3 动态链接库编译

在5.1.2节和5.2.2节中，不论是手动方式还是导入共享库向导的方式，都需要动态链接库文件rtlsdr.dll。接下来，本节将讨论如何从源程序中编译动态链接库文件。

5.3.1 编译前准备

首先在GitHub网站中搜索到rtlsdr的源文件，有多个版本可以选择，如可以选择osmocom/rtl-sdr，网址为https://github.com/osmocom/rtl-sdr。

在rtl-sdr-master\src文件下，可以找到rtlsdr.dll对应的源程序librtlsdr.c。预先安装pthread-win32和libusbx-1.0.18-win(本书配套程序)。预先准备编译依赖的静态库文件libusb-1.0.lib和pthreadVC2.lib(本书配套程序)。

5.3.2 编译步骤

将librtlsdr.c文件编译成动态链接库文件，需要预先安装C程序的编译软件，本例中选择VS 2013。Microsoft Visual Studio简称VS，是美国微软公司提供的软件开发工具。librtlsdr.c的编译步骤如下。

(1) 首先启动VS 2013，新建一个Win32项目，设置项目名称和保存路径，如图5-20所示。

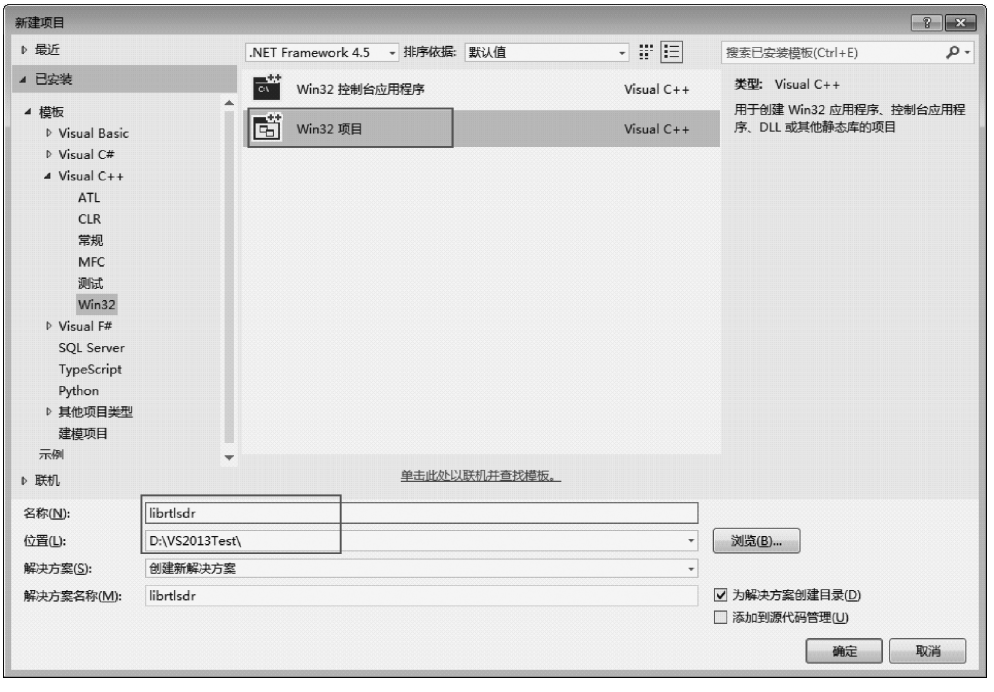


图5-20 新建一个Win32项目

(2) 在弹出的应用程序向导中选择应用程序类型，本例中选择DLL，在附加选项中选择

“空项目”。

（3）右击“源文件”文件夹，在弹出的快捷菜单中依次选择“添加”→“现有项”，将rtl-sdr-master\src文件夹下的librtlsdr.c、tuner_e4k.c、tuner_fc0012.c、tuner_fc0013.c、tuner_fc2580.c、tuner_r82xx.c 6个C文件导入，如图5-21所示。

图5-21　解决方案资源管理器

（4）将静态库文件libusb-1.0.lib和pthreadVC2.lib添加到VS的VC++目录中，如图5-22所示。右击图5-21所示的项目名称rtlsdrdll，在弹出的菜单中选择“配置属性”，在“VC++目录”页面中配置包含目录和库目录。

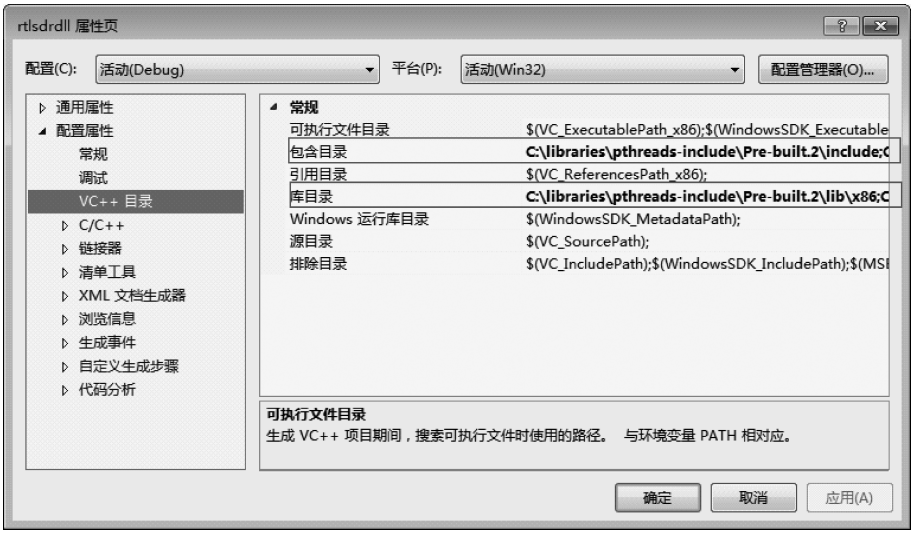


图5-22　配置包含目录和库目录

编辑包含目录，将Pre-built.2\include文件夹、libusbx-1.0.18-win文件夹下的include\libusbx-1.0和rtl-sdr-master\include文件夹添加到包含目录中，如图5-23所示。

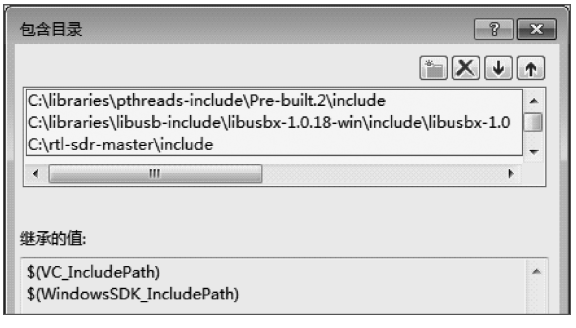


图5-23　配置包含目录

编辑库目录，将 Pre-built. 2\lib\x86 文件夹、libusbx-1. 0. 18-win\MS32\static 文件夹添加到库目录中，如图 5-24 所示。

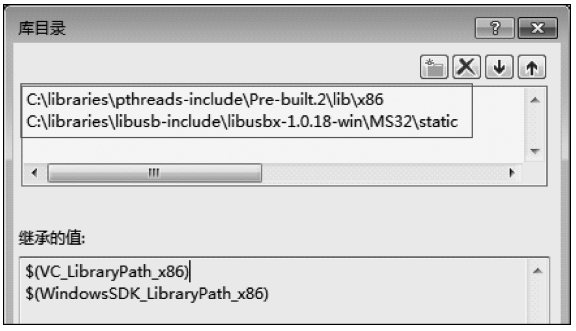


图 5-24　配置库目录

在“链接器”→“附加依赖项”中添加 libusb-1. 0. lib 和 pthreadVC2. lib 静态库文件，如图 5-25 所示。

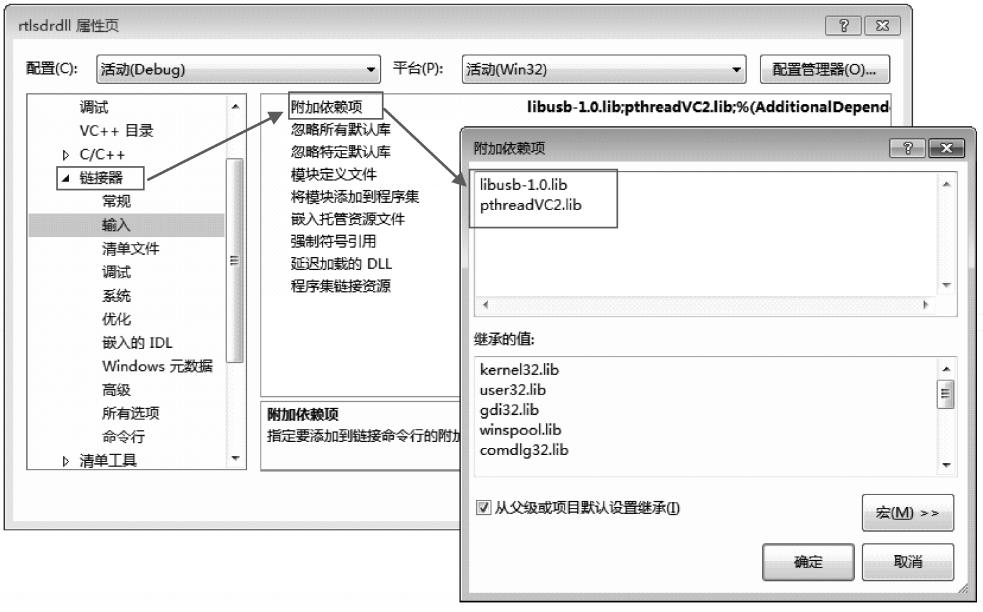


图 5-25　在链接器中添加附加依赖项

(5) 编译生成动态链接库 rtlsdrdll. dll，VS 编译输出如图 5-26 所示。需要特别注意，VS 默认生成的动态链接库是 32 位的。

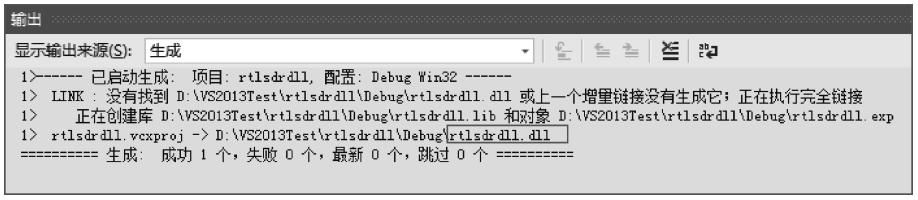


图 5-26　VS 编译输出

5.4　调用库函数的配置

在接口函数封装过程中，使用了 CLF 模块。接下来，本节将对 CLF 配置对话框中的 Function(函数)、Parameters(参数)、Callbacks(回调)、Error Checking(错误检测)4 个选项卡加以说明。

5.4.1　函数配置

在 Function(函数)选项卡中，可以设置 DLL 文件路径。在 Library name or path(库名/路径)文本框中，单击文件夹图标，选择 DLL 所在的路径，就可以完成设置。当调用包含 DLL 的 VI 被装入内存时，DLL 被自动装入内存。在 Library name or path 文本框的下方，还有一个 Specify path on diagram(在程序框图中指定路径)选项，选中该选项，DLL 将会被“动态加载”：只有程序运行到需要使用该 DLL 中的函数时，DLL 才会被装入内存。需要注意的是，选中 Specify path on diagram 之后，Library name or path 中输入的路径将无效。与此同时，CLF 模块将多出一对输入和输出端口，用于指定 DLL 的路径。

设置完 DLL 文件路径之后，在 Function name(函数名称)下拉列表中选择被调用函数的名称。单击下拉列表，将显示 DLL 中所有的可用函数。

在 Function 选项卡中可以配置 Thread(线程)。在 Thread 选项下，有 Run in UI thread 和 Run in any thread 两个单选按钮。Run in UI thread 表示仅运行在 UI 线程中；Run in any thread 表示运行在任意线程中。在本例的程序中，选择默认的 Run in UI thread。

在 Function 选项卡中还可以指明被调用函数的 Calling convention(调用约定)。CLF 支持两种调用约定：stdcall 和 C。stdcall 由被调用者负责清理堆栈，C 由调用者清理堆栈。Windows API 一般使用的都是 stdcall，标准 C 的库函数则大多使用 C。

5.4.2　端口参数配置

在 CLF 配置中，端口参数(Parameters)的配置比较复杂。DLL 调用出现的问题，大多是由于 Parameters 配置错误所引起的。接下来，本节将通过 rtlsdr 实例介绍该页面的配置。

在 Parameters 选项卡中，可以添加相应的参数并修改它们的返回值类型，直到页面底部的 Function prototype(函数原型)与 DLL 头文件中的函数定义相匹配，如图 5-27 所示。在 Type(类型)下拉列表中选择函数返回值的类型，如 Void(空)、Numeric(数值)或 String(字符串)。本例中选择 Numeric。

在 Data type 下拉列表中可以看到多种数据类型，如图 5-28 所示。本例中选择 Signed 32-bit Integer。

这里需要特别指出的一种数据类型，就是 C 语言中的指针类型，如 Signed Pointer-sized Integer。对于简单的 C 函数构成的 DLL，LabVIEW 利用 CLF 调用时就比较简单，但是对于参数或返回值是指针类型的 DLL 函数，LabVIEW 调用时就比较复杂。

指针就是变量的地址，将地址作为参数传递到 DLL 函数中，DLL 函数就可以操作这个

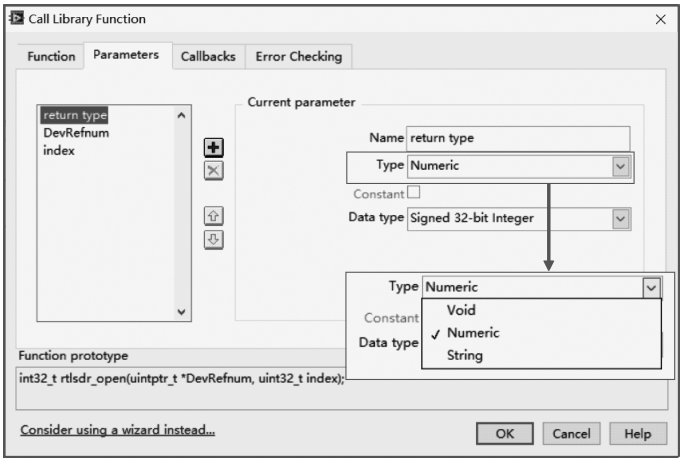


图 5-27　Parameters 选项卡

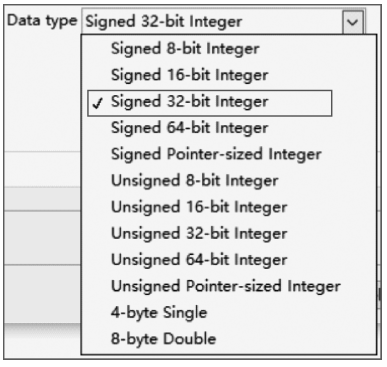
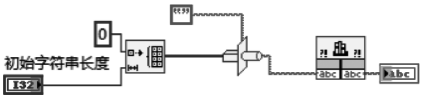
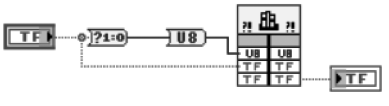


图 5-28　Data type 下拉列表

地址指向的变量。需要注意的是，在 32 位的操作系统中，可以使用 int32 数值表示指针。在 64 位的系统中，只能使用 I64 或 U64 表示指针。如果无法确知是 32 位还是 64 位的系统，则可以使用 Pointer-sized Integer 这种数据类型。

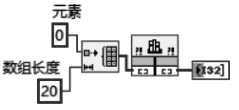
CLF 中常用指针类型配置和使用如表 5-1 所示，指向数值类型和字符类型的指针配置相对简单，找到与之相对应的选项即可。在使用布尔类型时，由于布尔类型在 DLL 函数和 LabVIEW VI 之间传递没有对应的数据类型，需要利用数值类型来传递，因此输入时先要把布尔量转换为数值，再传递给 DLL 函数，输出时再把数值转换为布尔量。需要注意的是，如果在 C 语言函数参数声明中有 const 关键字，需要选中 Constant 选项。

表 5-1　CLF 中指针配置和使用

C 语言声明	CLF 配置	CLF 使用
double * a	Type Numeric Constant ☐ Data type 8-byte Double Pass Pointer to Value	
char * a	Type String Constant ☐ String format C String Pointer Minimum size <None>	
bool * a	Type Adapt to Type Constant ☐ Data format Handles by Value	

数组在传递给 DLL 函数时，只能是指针，如表 5-2 所示。在传递数组类型时，Array format（数组格式）要选择为 Array Data Pointer（数组数据指针）。需要注意的是，LabVIEW 只支持 C 语言中的数值型数组。

表 5-2　CLF 中数组配置和使用

C 语言声明	CLF 配置	CLF 使用
int a[]	Type Array Constant ☐ Data type Signed 32-bit Integer Dimensions 1 Array format Array Data Pointer Minimum size <None>	
int * a[]	Type Array Constant ☐ Data type Signed 32-bit Integer Dimensions 1 Array format Array Data Pointer Minimum size <None>	

簇结构在 LabVIEW 中是常用的数据类型，C 语言的 struct（结构体）数据类型与之对应。在 CLF 节点的配置面板中，没有专门命名为 struct 或 cluster 参数类型，选择 Adapt to Type 就可以。需要注意的是，在 cluster 较为复杂的情况下，需要考虑字节对齐问题。

5.4.3　回调函数配置

Callbacks（回调）为 DLL 设置一些回调函数。DLL 文件中实现了各种类型的函数，当程序需要调用函数时，就要先载入 DLL，取得函数地址然后进行调用。有些情况下，需要将应用程序的某些功能提供给 DLL 使用，于是就可以使用回调函数。回调函数通常用于程序初始化、资源清理等工作。

5.4.4　错误检测配置

Error Checking（错误检测）用于设置错误处理方式，有 Maximum（最高级）、Default（默认）和 Disabled（不检测）3 个选项。最高级别的检测会对运行过程中检测到的每处错误进行反馈，但同时也会使 CLF 节点的运行速率降低。默认级别的检测会对程序执行最小程度的错误进行排查，对 CLF 节点的运行速率影响较小。不检测即不对程序中的错误进行检测，此时 CLF 节点以最快速度运行。

5.5　本章小结

本章以 RTL-SDR 为例，介绍了非 NI 设备控制方法。通过 CLF 模块，LabVIEW 可以调用 DLL 中的接口函数，从而实现对设备的控制。

首先以 RTL-SDR 的 LabVIEW 接口函数封装过程为例，介绍了 LabVIEW 中调用库函数模块的使用方法。

然后详细介绍了 LabVIEW 中导入共享库向导的使用方法。

接着介绍了动态链接库文件 rtlsdr. dll 在 VS 2013 环境下的编译过程。

最后介绍了调用库函数模块中 Fuction、Parameters、Callbacks、Error Checking 4 个选项卡的配置方法。