

C/C++驱动程序接口

C/C++是驱动程序设计的主要编程语言。因此，C/C++的接口是最好的匹配。所有显示硬件编程不同部分的手册的小例子都是用C完成的。由于库提供了一个标准接口，所以也很容易使用其他编程语言，如Delphi、Basic、Python或Java访问库。有关这方面的补充资料，请阅读以下各章。

头文件

在使用驱动程序之前的基本任务是将在USB-Stick上交付的头文件与板一起包含在内。头文件在目录/Driver/c_header中找到。请不要以任何方式更改它们，因为它们是用每个新的驱动程序版本更新的，以包括新的寄存器和新的功能。

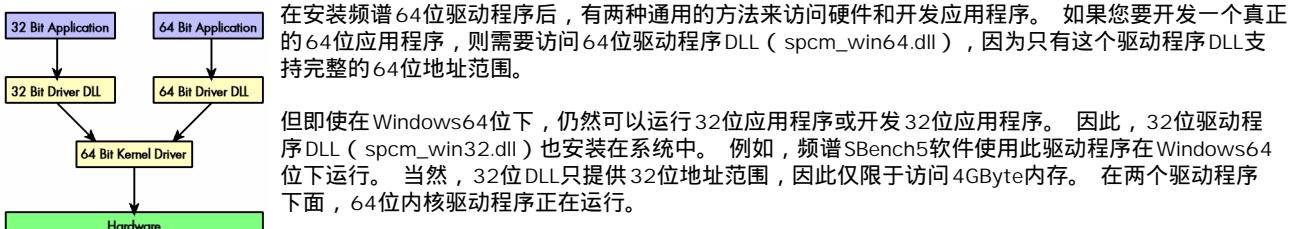
dlltyp.h	包括平台对数据类型和功能声明的特定定义.. 所有数据类型都基于这些定义。此类型定义文件的使用允许在不同平台上使用示例和程序，而不对程序源进行更改。头文件直接支持Micros of t VisualC、BorlandC++Builder和GNUC/C++。在使用其他编译器时，可能需要复制此文件并根据此编译器更改数据类型。
regs.h	定义所有寄存器和命令，这些寄存器和命令在频谱驱动程序中为不同的板使用。板使用的寄存器在文档的板特定部分中描述。此头文件对于所有卡片都是常见的。因此，该文件还包含大量用于其他卡类型的寄存器，而不是本手册中描述的寄存器。请坚持使用手册，看看哪些寄存器对您的类型卡是有效的。
spcm_drv.h	定义使用的Spcm驱动程序的功能。所有定义均取自文件dlltyp.h.. 这些职能本身说明如下。
spcerr.h	包含与频谱驱动程序一起使用的所有错误代码。所有可以由任何驱动程序函数返回的错误代码也在这里简要描述。错误代码及其含义详见本手册附录..

包含头文件的示例：

```
//----驱动程序包括-----
包括 " dlltyp.h " /1st包括#包括 " regs.h "
/2nd包括#包括 " spcerr.h " /3rd包括#包
括 " spcm_drv.h " /4th包括
```

 请始终保持包括四个频谱头文件的顺序。否则，一些或所有的功能不能正常工作，否则编译您的程序将是不可能的！

Windows 64位驱动程序的一般信息



将64位应用程序与32位DLL混合是不可能的，反之亦然。

Micros of t VisualC++6.0, 2005 and new32Bit

包括司机

驱动程序文件可以直接包含在Micros of t C++中，只需使用与驱动程序一起交付的库文件spcm_win32_msviscpp.lib。库文件可以在路径/示例/c_cpp/c_header中的USB-Stick上找到。请在VisualC++项目中包含库文件，如示例所示。下面描述的所有功能现在都可以在您的程序中使用。

例子

可以在路径/示例/c_cpp中的USB-Stick上找到示例。此目录包括许多不同的示例，可以与同一类型的任何卡一起使用（例如。A/D采购卡、D/A采购卡）。您可以使用这些示例作为自己编程的基础，并随心所欲地修改它们。示例目录包含Micros of t Visual C++6.0 (*.dsw)的运行工作区文件以及Micros of t Visual Studio2005和较新的 (*.vcproj)的项目文件，这些文件可以直接加载或导入和编译。在单独的子目录中还有一些独立的板类型示例。这些示例显示了卡片的不同方面，如编程选项或同步，并且可以与板类型特定的示例之一相结合。

由于这些例子是为卡类构建的，所以有一些检查例程和卡族之间的区别。差异化方面可以是渠道数量，数据宽度，最大速度或其他细节.. 建议更改与您的卡类型匹配的示例以获得最大的性能。请告知，这些例子是为了便于理解和简单地显示编程的一个方面。大多数示例没有针对最大吞吐量或重复率进行优化。

微软VisualC++2005和更新64位

根据Visual Studio套件的版本，可能需要在系统上安装一些额外的64位组件（SDK）。请按照MSDN上的说明获得更多信息。

包括司机

驱动程序文件可以通过简单地使用与驱动程序一起交付的库文件 spcm_win64_msvcpp.lib直接包含在Micros of tC++中。 库文件可以在路径/示例/c_cpp/c_header中的USB-Stick上找到。 下面描述的所有功能现在都可以在您的程序中使用。

C++Builder32位包括司机

通过简单地使用与驱动程序一起交付的库文件 spcm_win32_bcppb.lib，驱动程序文件可以很容易地包含在C++Builder中。 库文件可以在路径/示例/c_cpp/c_header中的USB-Stick上找到。 请在C++Builder项目中包含库文件，如示例所示。 下面描述的所有功能现在都可以在您的程序中使用。

例子

C++Builder示例与VisualC++示例共享源代码。 请参阅上述章节，以获得更详细的示例文档。 在每个示例目录中都有VisualC++以及C++Builder的项目文件。

Linux Gnu C/C++ 32/64 Bit包括司机

linux驱动程序的接口与windows接口没有区别。 请在makefile中包含spcm_linux.lib库，以便访问所有驱动程序函数。 makefile可能是这样的：

```
COMPILER = gcc
EXECUTABLE = test_prg
LIBS = -lspcm_linux

OBJECTS = test.o \
          test2.o

全部 : $(行政)

(执行部分) : (目的)
$月$月$月$月$月

%.o: %.cpp
$公司$公司$公司
```

例子

GnuC/C示例与VisualC++示例共享源代码。 请参阅上述章节，以获得更详细的示例文档。 每个示例目录包含GnuC/C示例的makefile。

.的C++ .NET

有关.的更多细节，请参阅下一章。 .NET including.

其他WindowsC/C++编译器32位包括司机

要访问驱动程序，必须从32位驱动程序DLL加载驱动程序函数。 大多数编译器提供特殊的工具来生成匹配的库（例如。 Borland提供了在windows驱动程序DLL中生成匹配库的Implib工具。 如果有这样的工具，建议使用它。 否则，驱动程序函数需要使用标准Windows函数从DLL加载。 在示例目录/示例/c_cpp/dll_loading中有一个示例显示该过程。

函数加载示例：

```
hDLL=Load Library (" spcm_win32.dll " ) ; /Load32位版本的Spcm驱动程序 pf n_spcm_h Open= ( SPCM_HOPEN* ) 获取
proc地址 ( hDLL , " _spcm_hOpen@4 " ) ; pf n_spcm_vClose= ( SPCM_VCL* ) 获取Proc地址 ( hDLL , " _spcm_vClose
4 " ) ;
```

其他Windows C/C++编译器64位包括司机

要访问驱动程序，必须从驱动程序DLL的64位加载驱动程序函数。 大多数编译器提供特殊的工具来生成匹配的库（例如。 Borland提供了在windows驱动程序DLL中生成匹配库的Implib工具。 如果有这样的工具，建议使用它。 否则，驱动程序函数需要使用标准Windows函数从DLL加载。 示例目录/示例/c_cpp/dll_loading中有一个示例，显示32位环境下的进程。 唯一需要修改的行是加载DLL的行：

函数加载示例：

```
h DLL=Load Library( " spcm_win64.dll " ) ; /Modified : 加载 Spcm 驱动程序的 64 位版本 , 这里为 pf n_spcm_h Open=( SPCM_HOPEN* ) 获取
Proc 地址 ( h DLL , " spcm_h Open " ) ;
pf n_spcm_v Close=( SPCM_VCLOSE* ) get Proc 地址 ( h DLL , " spcm_v Close " ) ;
```

National Instruments LabWindows/CVI

包括司机

要使用 LabWindows/CVI 下的频谱驱动程序，首先需要从驱动程序 DLL 加载函数。请使用库文件 spcm_win32_cvi.lib 访问驱动程序函数。

例子

实验室 Windows/CVI 的示例可以在目录 / 示例 / CVI 中的 USB-Stick 上找到。请将这些示例与标准 C/C 示例混合，以便访问卡的所有功能和模式。

驱动函数

驱动程序包含七个主要功能来访问硬件。

我们的司机用的是自己的型号

为了简化使用不同平台和编译器的头文件和示例，并避免任何隐式类型转换，我们决定使用我们自己的类型声明。这使我们能够使用平台独立和通用的示例和驱动程序接口。如果您不坚持这些声明，请务必使用相同的数据类型宽度。但是，强烈建议您使用我们定义的类型声明，以避免在程序中查找错误。如果您在我们的示例和驱动程序不支持的环境中使用驱动程序，请确保使用表示类似数据宽度的类型声明

第 8 号声明	类型	声明	uint8	类型
int16	8 位有符号整数 (从 -128 到 +127) 16 位有符号整数 (从 -32768 到 32767)	uint16	8 位无符号整数 (范围从 0 到 255) 16 位无符号整数 (范围从 0 到 65535)	
int32	32 位有符号整数 (范围从 -2147483648 到 2147483647)	uint32	32 位无符号整数 (范围从 0 到 4294967295)	
int64	64 位有符号整数 (全部范围)	uint64	64 位无符号整数 (全部范围)	
drv_handle	处理驱动程序，实现依赖于操作系统平台			

变量和函数表示法

在我们的头文件和示例中，我们对变量和函数使用了一种常见和可靠的表示法。每个名称还包含作为前缀的类型。这种表示法形式可以很容易地看到隐式类型转换，并尽量减少由于使用不正确的类型而导致的编程错误。你也可以在你的程序中使用这个符号形式 -

第 8 号声明	表示法	声明	uint8	表示法
int16	c 名称 (字节) n 名 l 名 (长) ll 名 (长)	uint16	c 姓名 (字) w 姓名 (字) d w 名称 (双字)	
int32		uint32	q w 名称 (四字)	
int64		uint64		
int32*	pl name (指向 long 的指针)	char	sz 名称 (带零终止的字符串)	

函数 spcm_hOpen

此功能初始化并打开支持新的 SpcM 驱动程序接口的已安装的卡，在打印时，这些卡都是 M2i/M3i/M4i/M4x/M2p 系列和相关的数字化器 NETBOX/生成器 NETBOX 设备的卡。该函数返回一个必须用于驱动程序访问的句柄。如果找不到卡，或者驱动程序的加载产生了错误，则函数返回 NULL。调用此函数时，从硬件中读出所有特定于卡的安装参数，并存储在驱动程序中。只有通过一个软件打开一个设备是可能的，因为并发的硬件访问可能是非常关键的系统稳定性。因此，当试图打开同一设备两次时，将引发错误，并且函数返回 NULL。

函数 spcm_hopen (const char*sz 设备名称) :

```
打开 ( / 尝试打开设备并返回句柄或错误代码 constchar*sz 设备名称 ) ; / 要打开的设备名称
```

在 Linux 下，函数调用中的设备名称需要是有效的设备名称。如果不使用标准 Linux 设备名称，请根据设备的位置更改字符串。驱动程序在 /dev/spcm0, /dev/spcm1 等下安装为默认.. 内核驱动程序以 0 开始对设备进行编号。

在 Windows 下使用的设备名称的唯一部分是尾号。其余的设备名称被忽略。因此，为了使示例保持简单，我们在所有示例中使用 Linux 表示法。尾号给出设备开启的指标.. Windows 内核驱动程序对从 0 开始的启动时间中发现的所有设备进行编号。

本地已安装的卡片示例

```
在 hD RV=spcm_hOpen ( “ /dev/spcm0 ” ) 错误的情况下 , 将句柄返回到打开的驱动程序或 NULL ; / 字符串返回到要打开的驱动程序
如果 ( ! h Drv )
打印 f ( “ 打开驱动程序失败 \n ” ) ;
```

数字化器 NETBOX/生成器 NETBOX 和远程安装卡的示例

```
在 hD RV=spcm_hOpen ( “ TCPIP : 192.168.169.14 : INS TO : INS TR ” ) 错误的情况下 , 将句柄返回到打开的驱动程序或 NULL ;
如果 ( ! h Drv )
打印 f ( “ 打开驱动程序失败 \n ” ) ;
```

如果函数返回 NULL , 则可以通过简单地将此 NULL 传递给错误函数来读出失败的打开函数的错误描述。 错误函数将在下一个主题中描述。

函数 spcm_v Close

此函数关闭驱动程序并释放所有分配的资源。 关闭驱动程序句柄后 , 不可能再访问此驱动程序。 确保关闭驱动程序 , 如果您不再需要它 , 以允许其他程序访问此设备。

函数 spcm_v Close :

```
void_stdcall spcm_v Close ( / 关闭设备 )
/ 已打开设备的句柄
```

例子 :

```
spcm_vClose ( hDrv );
```

函数 spcm_dw Set Param

所有的硬件设置都是基于软件寄存器 , 可以由其中一个函数 spcm_dw SetParam 设置。 这些函数将寄存器设置为定义值或执行命令。 板必须首先由 spcm_hOpen 函数初始化。 参数寄存器必须具有在 regs.h 中定义的有效软件寄存器常数。 驱动程序的可用软件寄存器列于下面文档的董事会特定部分。 如果发生错误 , 函数返回 32 位错误代码。 如果没有发生错误 , 则函数返回 ERR_OK , 什么是零。

函数 spcm_dw Set Param

```
uint32_stdcall spcm_dw Set Param_i32 ( /Return 值是一个错误代码 drv_handle h Device , / 句柄到已经打开的设备 int32l Register , / 软件寄存器要修改 int32l Value )
; / 要设置的值
```

```
uint32_stdcall spcm_dw Set Param_i64m ( /return 值是一个错误代码 drv_handle h Device , /handle 到已经打开的设备 int32l Register , / 软件寄存器要修改
```

在 32l 值高 , / 上 32 位的值。 包含符号位的 ! uint32d w 值较低) ; / 值较低 32 位。

```
uint32_stdcall spcm_dw Set Param_i64 ( /return 值是错误代码 drv_handle h Device
, / 句柄到已经打开的设备 int32l Register , / 软件寄存器要修改 int64ll Value ) ; / 要
设置的值
```

例子 :

```
如果 ( spcm_dw SetParam_i64 ( h Drv , SPC_MEMS IZE , 16384 ) !=ERR_OK ) printf
( “ 设置内存大小时的错误 ” ) ;
```

此示例将内存大小设置为 16k 样本 (16384) 。 如果发生错误 , 示例将显示一个简短的错误消息

函数 spcm_dwGet Param

所有硬件设置都基于软件寄存器 , 其中一个函数 spcm_dwGet Param 可以读取这些寄存器。 这些函数读取内部寄存器或状态信息。 板必须首先由 spcm_hOpen 函数初始化。 参数寄存器必须具有 regs.h 文件中定义的有效软件寄存器常数。 驱动程序的可用软件寄存器列于下面文档的董事会特定部分。 如果发生错误 , 函数返回 32 位错误代码。 如果没有发生错误 , 则函数返回 ERR_OK , 什么是零。

函数 spcm_dwGet Param

```
uint32_stdcall spcm_dw Get Param_i32 ( /Return值是一个错误代码drv_handle h Device , /句柄到已经打开的设备int32l Register , /软件寄存器读出int32*pl Value )
; /返回值的指针
```

```
uint32_stdcall spcm_dw Get Param_i64m ( /return值是一个错误代码drv_handle h Device , /handle到已经打开
的设备int32l Register , /软件寄存器读出
```

返回值 uint32*pd w值低的上部的指针；返回值的下部的指针

```
uint32_stdcall spcm_dw Get Param_i64 ( /Return值是一个错误代码drv_handle h Device , /句柄到已经打开的设备int32l Register , /软件寄存器读出int64*PLL Value
) ; /指针对于返回值
```

例子：

```
int32l序号 ;
spcm_dw Get Param_i32 ( h Drv , SPC_PCIS ERIA L NO , &l序号 ) ; printf ( “ 您的卡有
序列号 : %05d\n ” , l序号 ) ;
```

示例读出已安装的卡的序列号并打印出来.. 由于序列号在所有情况下都可用，因此调用此函数时没有错误检查。

spcm_dw_Set Param和spcm_dw_Get Param的不同调用类型： i32、 i64、 i64m

这三个函数只在用来调用它们的参数的类型上有所不同。由于有些寄存器可以超过32位整数范围（如内存大小或后置触发器），建议使用_i64函数访问这些寄存器。然而，由于有些程序或编译器不支持64位整数变量，所以有两个函数仅限于32位整数变量。如果您没有访问超过32位整数的寄存器，请使用_i32函数。如果您访问的寄存器超过64位值，请使用_i64m调用约定。在这里，64位值被分成一个低双字部分和一个高双字部分。请务必用有效的信息填写这两个部分。

如果访问具有32位功能的64位寄存器，则行为取决于当前位于寄存器中的实际值。请看这张表，看看不同的反应取决于登记册的大小：

内部登记册	读/写	功能类型	行为
32位寄存器	读	spcm_dwGetParam_i32	值在pl值中作为32位整数返回，在PLL值
32位寄存器	读读	spcm_dwGetParam_i64	中作为64位整数返回
32位寄存器		spcm_dwGetParam_i64m	值以64位整数返回，下部在pl值低，上部在pl值高。上面的部分可以忽略，因为它只是一个符号扩展
32位寄存器	写写	spcm_dwSetParam_i32	32位值可直接写入
32位寄存器	写	spcm_dwSetParam_i64	64位值可直接写入，请务必不要超过有效寄存器值范围
32位寄存器		spcm_dwSetParam_i64m	32位值被写入lvalue Low , value llvalue High需要包含此值的符号扩展。如果ll值Low是一个值>=0，则ll值高可以是0，如果ll值Low是一个值<0，则ll值高必须是-1。
64位寄存器	念	spcm_dwGetParam_i32	如果内部寄存器有一个值在32位整数范围内(-2G到(2G-1))，则该值正常返回。如果内部寄存器超过此大小，则返回错误代码ERR_EXCEEDSINT32。作为一个例子：只要这个内存<2GByte，读取安装的内存就会工作。如果安装的内存>=2GByte，则函数将返回错误。
64位寄存器	读读	spcm_dwGetParam_i64	值在PLL值中作为64位整数值返回，而不依赖于内部寄存器的值。
64位寄存器		spcm_dwGetParam_i64m	内部值分为低和高两部分。只要内部值在32位范围内，低部分pl值低包含32位值，上部pl值高可以忽略.. 如果内部值超过32位范围，则绝对需要考虑两个值部分。
64位寄存器	write	spcm_dwSetParam_i32	要写入的值限制在32位范围内。如果应写入高于32位范围的值，则需要使用其他函数类型之一。
64位寄存器	write	spcm_dwSetParam_i64	这个值必须分成两部分。确保用正确的符号扩展填充上部pl值高，即使您每次只编写32位值作为驱动程序解释函数调用的两个部分。
64位寄存器	write	spcm_dwSetParam_i64m	该值可以直接独立于大小写入。

函数 spcm_dwGet ContBuf

此函数以字节为单位读取内部连续内存缓冲区，以防分配了一个。如果没有分配缓冲区，则函数返回大小为零和NULL指针。您可以使用此缓冲区进行数据传输。因为缓冲区是在内存中连续分配的

数据传输将加快高达 15%-25%，这取决于您的具体类型的卡。请参阅本手册附录中的进一步细节。

```
uint32_stdcall spcm_dw Get Cont Buf_i64 ( /return值是错误代码drv_handle h Device , /handle到已经打开的设备
uint32dw Buf Type , /type of buffer to reading as under SPCM_BUF_XX XX void**ppv Data Buffer , /address of available data buffer
uint64*pq w Cont Buf Len ) ; /可用连续缓冲区长度
uint32_stdcall spcm_dw Get Cont Buf_i64m ( /return值是错误代码drv_handleh Device , /handle到已经打开的设备
uint32dw Buf Type , /type of buffer to reading as under SPCM_BUF_XX XX void**ppv Data Buffer , /address of available data buffer
uint32*pd w Cont Buf Len H , /可用连续缓冲区长度的高部分uint32*pd w Cont Buf Len L ) ; /可用连续缓冲区长度的低部分
```

 这些函数已在驱动程序版本 1.36 中添加。这些函数在旧的驱动程序版本中不可用。

 这些功能也只对本地安装的卡产生影响，对任何数字化器 NETBOX 或生成器 NETBOX 产品既没有用也不可用，因为在这样的设置中不涉及本地内核驱动程序。对于远程设备，这些函数将返回缓冲区的 NULL 指针，长度为 0 字节。

函数 spcm_dw Def Transfer

spcm_dw Def Transfer 函数定义了以下数据传输的缓冲区。此函数只定义缓冲区，在调用此函数时没有执行数据传输。相反，数据传输是从单独的寄存器命令开始的，这些命令将在后面的章节中记录下来。在这个位置还有一个功能参数的详细描述。

请确保此函数的所有参数匹配。特别需要的是，缓冲区地址是指向至少具有函数调用中定义的大小的内存缓冲区的有效地址。请告知，使用非有效参数调用此函数可能会使您的系统崩溃，因为这些值是后续 DMA 传输的基础。

此函数的使用将在后面的一章中更详细地描述。

函数 spcm_dw Def Transfer

```
uint32_stdcall spcm_dw Def Transfer_i64m ( /定义传输缓冲区 2x32位无符号整数drv_handle设备 , /句柄到已打开的设备
uint32d wBuf Type , /类型的缓冲区，以定义如上面所列的SPCM_BUF_XX XXuint32dw Direction , /上面定义的传输方向
通知大小 , /否。之后发送事件的字节 ( 0=传输结束 ) void*pv数据缓冲区 , /指向数据缓冲区的指针
uint32dw BrdOffs H , /board内存中偏移量的高部分 uint32dw BrdOffs L , /board内存中偏移量的低部分 uint32dw Transfer Len H , /传输缓冲区长度的高部分 uint32dw Transfer Len L ) ; /传输缓冲区长度的低部分

使用 64 位无符号整数值drv_handleh Device , /句柄到已经打开的设备定义传输缓冲区
uint32d wBuf Type , /类型的缓冲区，以定义如上面所列的SPCM_BUF_XX XXuint32dw Direction , /上面定义的传输方向
通知大小 , /否。之后发送事件的字节 ( 0=传输结束 ) void*pv数据缓冲区 , /指向数据缓冲区的指针
uint64qw BrdOffs , /偏移量，用于板存储器uint64qw传输Len ) ; /缓冲区长度
```

此函数有两种不同的格式，如 spcm_dwGet Param 和 spcm_dw SetParam 函数。背景是一样的。只要您使用的编译器支持 64 位整数值，请使用 _i64 函数。任何其他平台都需要使用 _i64m 函数，并在两个 32 位单词中分割偏移量和长度。

例子：

```
int16* pnBuffer = (int16*) pvAllocMemPageAligned (16384);
if (spcm_dwDefTransfer_i64 (hDrv, SPCM_BUF_DATA, SPCM_DIR_CARDTOPC, 0, (void*) pnBuffer, 0, 16384) != ERR_OK)
printf ("DefTransfer failed\n");
```

该示例定义了一个 8k 样本的数据缓冲区，该数据缓冲区为 16 位整数值 = 16k 字节 (16384 字节)，用于从卡到 PC 内存的传输。由于通知大小设置为 0，我们只希望在传输完成时得到一个事件。

函数 spcm_dw Invalidate Buf

无效缓冲区函数用于告诉驱动程序，已经使用 spcm_dw DefTransfer 调用设置的缓冲区不再有效。有必要使用相同的缓冲区类型，因为驱动程序同时处理不同的缓冲区。如果要在调用 spcm_dw DefTransfer 函数后删除缓冲区内存，请调用此函数。如果在调用 spcm_dw Def Transfer 之后已经传输了缓冲区，则不需要调用此函数。当调用 spcm_dw DefTransfer 时，任何进一步定义的缓冲区都会自动失效。

函数 spcm_dw Invalidate Buf

```
uint32_stdcall spcm_dw Invalidate Buf ( /使传输缓冲区 drv_handle h Device失效 , /使已经打开的设备 uint32dw BufType的句柄失效 ) ; /缓冲区的类型，以使其失效，如上述SPCM_BUF_XX XX下所列
```

函数 spcm_dw获取错误信息

函数返回上次发生的错误的完整错误信息。 错误处理本身将在后面的一章中更详细地解释。 在调用此函数时，请确保分配了至少具有ER RORT EXTLEN长度的文本缓冲区。 错误文本函数返回错误的完整描述，包括引发错误的寄存器/值组合和错误细节的简短描述。 此外，还可以为自己的错误处理获取生成寄存器/值的错误。 如果不需要，寄存器/值的缓冲区可以留给NULL。



请注意，超时事件（ERR_TIMEOUT）不算为内部错误，因为它不是锁定驱动程序，而是作为有效事件。因此，Get错误信息函数不会返回超时事件，即使它发生在两者之间。您只能将ERR_TIMEOUT识别为调用的等待函数的直接返回值。

函数 spcm_dw获取错误信息

```
uint32_stdcall spcm_dwGetErrorInfo_i32 ( drv_handle h Device , /handle到已经打开的设备 uint32*pd wError Reg , /address of error寄存器 (如果不感兴趣，可以为零) int32*plerror值 , /address of error值 (如果不感兴趣，可以为零) char psz Error Text Buffer[ER RORT EXTLE N] ; /text buffer for text Error
```

例子：

```
[ER RORT EXTLE N] ;
if (spcm_dwSetParam_i64 (hDrv, SPC_MEMSIZE, -1))
{
    spcm_dwGetErrorInfo_i32 (hDrv, NULL, NULL, szErrorBuf);
    打印f ("一组memsize失败，错误消息：%s\n" , szErrorBuf);
}
```