




 LOTO_OpenSource_sdk.pdf -> 二次开发 SDK 函数说明

 OSC482 系列二次开发 SDK(包)


1. OpenSource_LOTO_482 -> OSC482(C# winform) Demo 例子;
2. OpenSource_LOTO_482Signal -> OSC482 信号发生器(C# winform) Demo 例子;
其他型号如: (OSCA02/OSC2002/OSCH02)均可参考;
3. Python OSC482 -> OSC482(Python 32 位) Demo 例子;
4. OpenSource_LOTO_482_python_QT -> OSC482(Python 32 位 + QT 界面)Demo 例子;
5. LabVIEW_子 VI_LOTO_482 -> OSC482(LabVIEW 2016) Demo 例子;
6. MFC_MeasuredVal -> MFC(VC++)实时获取示波器软件测量值 Demo 例子;

 OSC2002 系列二次开发 SDK(包)


1. OpenSource_LOTO_2002 -> OSC2002(C# winform) Demo 例子;
2. Python OSC2002 -> OSC2002(Python 32 位) Demo 例子;

 OSCA02 系列二次开发 SDK(包)


1. OpenSource_LOTO_A02 -> OSCA02(C# winform)Demo 例子;
2. Python OSCA02 -> OSCA02(Python 32 位)Demo 例子;
3. OpenSource_LOTO_A02_Buffer -> OSCA02(流模式 大缓冲区 4M 8M) Demo 例子;
4. OpenSource_LOTO_A02_FFT_FIR -> OSCA02(FFT 频域和 FIR 滤波) Demo 例子;
5. OpenSource_LOTO_A02_Phase -> OSCA02 计算两个通道波形(相位差) Demo 例子;
6. OpenSource_LOTO_A02Signal -> OSCA02 信号发生器(C# winform) Demo 例子;
其他型号如: (OSCA02/OSC2002/OSCH02)均可参考;
7. OpenSource_LOTO_A02Charts -> OSCA02 多通道板卡(C# winform) Demo 例子;
8. OpenSource_LOTO_A02_FreqDuty -> OSCA02 计算(频率和占空比)(C#) Demo 例子;

 OSCH02 系列二次开发 SDK(包)

1. OpenSource_LOTO_H02 -> OSCH02(C# winform)Demo 例子;
2. Python OSCH02 -> OSCH02(Python 32 位)Demo 例子;

 OSC980(双通道)系列二次开发 SDK(包)

1. OpenSource_LOTO_980 -> OSC980(C# winform)Demo 例子;
2. Python OSC980 -> OSC980(Python 32 位)Demo 例子;
3. OpenSource_LOTO_980_Buffer -> OSC980(流模式 大缓冲区 4M 8M)Demo 例子;

 OSC984(四通道)系列二次开发 SDK(包)

1. OpenSource_LOTO_984 -> OSC984(C# winform)Demo 例子;

DLL 动态链接库名称: USBInterFace.dll, 不同型号的该文件可能不同, 所以需要注意不要混用。

函数名: SpecifyDevIdx

函数定义 void SpecifyDevIdx(int index);

参数说明: int index, 32 位整数, 代表设备编号。

设备型号	index 值
OSC802	0
OSC482	6
OSCA02	6
OSC2002	2
OSCH02	7

返回值: 无

说明: 此函数在动态库 USBInterFace.dll 中, 用来设置产品编号, 不同型号产品编号可能不同, 需要在软件打开设备前调用该函数设置正确的编号。如果设置了错误的设备编号, 软件将无法正确打开设备并操作。

c#举例:

```
[DllImport("USBInterFace.dll", CharSet = CharSet.Ansi, CallingConvention =  
CallingConvention.StdCall, EntryPoint = "SpecifyDevIdx")]  
public static extern void SpecifyDevIdx(Int32 index);
```

函数名: DeviceOpen

函数定义 unsigned long DeviceOpen(void);

参数说明: 无。

返回值: unsigned long, 返回数值反映该函数调用的结果, 返回值为-1 说明函数失败, 返回值为 0 说明函数成功。

说明: 此函数在动态库 USBInterFace.dll 中, 用来打开示波器设备, 如果成功打开, DLL 中会保存该设备的句柄, 并为后续操作准备资源, 我们才能调用其他函数对示波器做设置和操作。该函数在程序运行初期打开一次即可。常见的失败原因可能是前面提到的 SpecifyDevIdx 函数设置错误, 或者示波器的驱动程序未安装, 或者示波器设备未插入。

c#举例:

```
[DllImport("USBInterFace.dll", CharSet = CharSet.Auto, CallingConvention =  
CallingConvention.StdCall)]  
public static extern Int32 DeviceOpen();
```

函数名： DeviceClose

函数定义 unsigned long DeviceClose(void);

参数说明： 无。

返回值： unsigned long，返回数值反映该函数调用的结果，返回值为-1 说明函数失败，返回值为 0 说明函数成功。

说明： 此函数在动态库 USBInterFace.dll 中，用来关闭示波器设备，如果成功关闭，DLL 中会释放掉该设备的句柄，并清理和释放所占据的软件资源。之后我们调用其他函数对示波器做设置和操作将会失败。通常我们只需要在你的程序结束的时候调用一次即可。

c#举例：

```
[DllImport("USBInterFace.dll", CharSet = CharSet.Auto, CallingConvention = CallingConvention.StdCall)]  
  
public static extern Int32 DeviceClose();
```

函数名： USBCtrlTransSimple

函数定义 unsigned long USBCtrlTransSimple (unsigned long Request);

参数说明： unsigned long Request：命令码，不同的命令码代表不同的指令。在本函数说明的后面会附上表格列出开放的命令码。

返回值： unsigned long，如果是需要从下位机获取数据的命令，此返回值会返回下位机传来的数值。

说明： 此函数在动态库 USBInterFace.dll 中，用 USB 控制传输的方式给下位机示波器发送命令。使用的前提是示波器设备已经被 DeviceOpen 成功打开。可以是单向的命令，也可以是双向的获取数据的命令。

c#举例：

```
[DllImport("USBInterFace.dll", CharSet = CharSet.Auto, CallingConvention = CallingConvention.StdCall)]  
  
public static extern Int32 USBCtrlTransSimple(Int32 Request);
```

返回值	Request	说明
/	0x33	开始采集 ADC 数据
返回值 33 说明采集和存储完毕可以读取了	0X50	查询 AD 采集和存储是否完毕 (对 OSC482 系列不适用)

函数名： USBCtrlTrans

函数定义 `unsigned char USBCtrlTrans (unsigned char Request, unsigned short usValue, unsigned long outBufSize);`

参数说明： `unsigned char Request`: 命令码，不同的命令码代表不同的指令。在本函数说明的后面会附上表格列出开放的命令码。

`unsigned short usValue`: 命令码的参数，作为命令码的附属信息传递给 DLL。

`unsigned long outBufSize`: 未使用，固定传 1 即可。

返回值： `unsigned char`，如果是需要从下位机获取数据的命令，此返回值会返回下位机传来的数值。

说明： 此函数在动态库 USBInterFace.dll 中，用 USB 控制传输的方式给下位机示波器发送命令。使用的前提是示波器设备已经被 DeviceOpen 成功打开。可以是单向的命令，也可以是双向的获取数据的命令。

c#举例：

```
[DllImport("USBInterFace.dll", CharSet = CharSet.Auto, CallingConvention = CallingConvention.StdCall)]
public static extern byte USBCtrlTrans(byte Request, UInt16 usValue, uint outBufSize);
```

OSC482 系列的指令：

返回值	Request	usValue	说明
/	0x94	0x10	设置 50MHz 采样率
/		0x06	设置 4MHz 采样率
/		0x01	设置 2.4MHz 采样率
/		0x07	设置 500kHz 采样率
/		0x11	设置 240kHz 采样率
/	0x22	0x02	chA 输入量程设置为: -5V ~ +5V
/		0x02	chA 输入量程设置为: -1V ~ +1V
/		0x06	chA 输入量程设置为: -2.5V ~ +2.5V
/		0x04	chA 输入量程设置为: -500mV ~ +500mV
/		0x08	chA 输入量程设置为: -250mV ~ +250mV
/	0x23	0x02	chB 输入量程设置为: -5V ~ +5V
/		0x12	chB 输入量程设置为: -2.5V ~ +2.5V
/		0x00	chB 输入量程设置为: -1V ~ +1V
/		0x10	chB 输入量程设置为: -500mV ~ +500mV
/		0x20	chB 输入量程设置为: -250mV ~ +250mV
/	0x24	0x08	设置 chA 为 DC 耦合
/		0x00	设置 chA 为 AC 耦合
/	0x25	0x01	设置 chB 为 DC 耦合

/		0x00	设置 chB 为 AC 耦合
/	0x84	0x00	两条命令设置 chB 用作示波器通道
/	0x81	0x00	
/	0x84	0x00	两条命令设置 chB 用作逻辑分析仪使用，每个 BIT 代表一个逻辑通道
/	0x82	0x00	

OSCA02, OSC2002, OSCH02 系列的指令:

8 位控制字节 g_CtrlByte0:

位 7	位 6	位 5	位 4	位 3	位 2	位 1	位 0
			chA AC/DC	Fre3	Fre2	Fre1	Fre0

8 位控制字节 g_CtrlByte1:

位 7	位 6	位 5	位 4	位 3	位 2	位 1	位 0
			chB AC/DC	Vol2	Vol1	Vol0	

命令	OSCA02, OSC2002	OSCH02
g_CtrlByte0 &= 0xf0; g_CtrlByte0 = 0x00; USBCtrlTrans(0x94, g_CtrlByte0, 1);	100M Hz 采样率	125M Hz 采样率
g_CtrlByte0 &= 0xf0; g_CtrlByte0 = 0x08; USBCtrlTrans(0x94, g_CtrlByte0, 1);	12.5M Hz 采样率	15M Hz 采样率
g_CtrlByte0 &= 0xf0; g_CtrlByte0 = 0x0c; USBCtrlTrans(0x94, g_CtrlByte0, 1);	781K Hz 采样率	设置 976K Hz 采样率
g_CtrlByte0 &= 0xf0; g_CtrlByte0 = 0x0e; USBCtrlTrans(0x94, g_CtrlByte0, 1);	49K Hz 采样率	60K Hz 采样率
g_CtrlByte0 &= 0xf0; g_CtrlByte0 = 0x04; USBCtrlTrans(0x94, g_CtrlByte0, 1);	96K Hz 采样率	96K Hz 采样率

合成采集		
命令	OSCA02E, OSCH02	
g_CtrlByte1 = 0x80; USBCtrlTrans(0x24, g_CtrlByte1, 1)	开启	chA 通道 = 采样率 * 2 倍 条件: chB 通道需要开启
g_CtrlByte1 &= 0x7f; USBCtrlTrans(0x24, g_CtrlByte1, 1)	关闭	

命令	OSCA02, OSC2002, OSCH02
g_CtrlByte1 &= 0xf7; g_CtrlByte1 = 0x08; USBCtrlTrans(0x22, 0x00, 1); USBCtrlTrans(0x24, g_CtrlByte1, 1);	chA 输入量程设置为: -8V ~ +8V
g_CtrlByte1 &= 0xf7; g_CtrlByte1 = 0x08; USBCtrlTrans(0x22, 0x02, 1); USBCtrlTrans(0x24, g_CtrlByte1, 1);	chA 输入量程设置为: -5V ~ +5V

g_CtrlByte1 &= 0xF7; g_CtrlByte1 = 0x08 USBCtrlTrans(0x22, 0x04, 1); USBCtrlTrans(0x24, g_CtrlByte1, 1);	chA 输入量程设置为: -2.5V ~ +2.5V
g_CtrlByte1 &= 0xF7; g_CtrlByte1 = 0x08; USBCtrlTrans(0x22, 0x06, 1); USBCtrlTrans(0x24, g_CtrlByte1, 1);	chA 输入量程设置为: -1V ~ +1V
g_CtrlByte1 &= 0xF7; USBCtrlTrans(0x22, 0x02, 1); USBCtrlTrans(0x24, g_CtrlByte1, 1);	chA 输入量程设置为: -500mV ~ +500mV
g_CtrlByte1 &= 0xF7; USBCtrlTrans(0x22, 0x04, 1); USBCtrlTrans(0x24, g_CtrlByte1, 1);	chA 输入量程设置为: -250mV ~ +250mV
g_CtrlByte1 &= 0xF7; USBCtrlTrans(0x22, 0x06, 1); USBCtrlTrans(0x24, g_CtrlByte1, 1);	chA 输入量程设置为: -100mV ~ +100mV
g_CtrlByte1 &= 0xF9; USBCtrlTrans(0x23, 0x00, 1); USBCtrlTrans(0x24, g_CtrlByte1, 1);	chB 输入量程设置为: -8V ~ +8V
g_CtrlByte1 &= 0xF9; g_CtrlByte1 = 0x02; USBCtrlTrans(0x23, 0x00, 1); USBCtrlTrans(0x24, g_CtrlByte1, 1);	chB 输入量程设置为: -5V ~ +5V
g_CtrlByte1 &= 0xF9; g_CtrlByte1 = 0x04; USBCtrlTrans(0x23, 0x00, 1); USBCtrlTrans(0x24, g_CtrlByte1, 1);	chB 输入量程设置为: -2.5V ~ +2.5V
g_CtrlByte1 &= 0xF9; g_CtrlByte1 = 0x06; USBCtrlTrans(0x23, 0x00, 1); USBCtrlTrans(0x24, g_CtrlByte1, 1);	chB 输入量程设置为: -1V ~ +1V
g_CtrlByte1 &= 0xF9; g_CtrlByte1 = 0x02; USBCtrlTrans(0x23, 0x40, 1); USBCtrlTrans(0x24, g_CtrlByte1, 1);	chB 输入量程设置为: -500mV ~ +500mV
g_CtrlByte1 &= 0xF9; g_CtrlByte1 = 0x04; USBCtrlTrans(0x23, 0x40, 1); USBCtrlTrans(0x24, g_CtrlByte1, 1);	chB 输入量程设置为: -250mV ~ +250mV
g_CtrlByte1 &= 0xF9; g_CtrlByte1 = 0x06; USBCtrlTrans(0x23, 0x40, 1); USBCtrlTrans(0x24, g_CtrlByte1, 1);	chB 输入量程设置为: -100mV ~ +100mV
g_CtrlByte0 &= 0xef; g_CtrlByte0 = 0x10; USBCtrlTrans(0x94, g_CtrlByte0, 1);	设置 chA 为 DC 耦合
g_CtrlByte0 &= 0xef; USBCtrlTrans(0x94, g_CtrlByte0, 1);	设置 chA 为 AC 耦合
g_CtrlByte1 &= 0xef; g_CtrlByte1 = 0x10; USBCtrlTrans(0x24, g_CtrlByte1, 1);	设置 chB 为 DC 耦合
g_CtrlByte1 &= 0xef; g_CtrlByte1 = 0x00; USBCtrlTrans(0x24, g_CtrlByte1, 1);	设置 chB 为 AC 耦合
USBCtrlTrans(0xE7, 0x00, 1);	禁止触发 和 禁止外触发
USBCtrlTrans(0xE7, 0x01, 1);	使能触发 和 开启外触发
USBCtrlTrans(0xC5, 0x00, 1);	上升沿触发
USBCtrlTrans(0xC5, 0x01, 1);	下降沿触发
USBCtrlTrans(0x16, 128, 1);	设置触发电平为 128, 非外触发有效

需要外触发功能时，购买前请联系厂商提出要求。 外触发时，设置的触发电平是无效的。	外触发
g_CtrlByte1 &= 0xfe; g_CtrlByte1 = 0x01; USBCtrlTrans(0x24, g_CtrlByte1, 1);	chB 开启
g_CtrlByte1 &= 0xfe; g_CtrlByte1 = 0x00; USBCtrlTrans(0x24, g_CtrlByte1, 1);	chB 关闭

采集卡模式	
命令	OSCA02, OSC2002, OSCH02
g_CtrlByte0 &= 0xf0; g_CtrlByte0 = 0x06; USBCtrlTrans(0x94, g_CtrlByte0, 1);	设置 4M 采样率
g_CtrlByte0 &= 0xf0; g_CtrlByte0 = 0x05; USBCtrlTrans(0x94, g_CtrlByte0, 1);	设置 2.4M 采样率
g_CtrlByte0 &= 0xf0; g_CtrlByte0 = 0x07; USBCtrlTrans(0x94, g_CtrlByte0, 1);	设置 500K 采样率

函数名： GetBuffer4Wr

函数定义 unsigned char* GetBuffer4Wr (int index);

参数说明： int index: 固定传 -1 作为参数。

返回值： unsigned char*, 返回一个缓冲区的首指针，只要不为 NULL 即可使用。

说明： 此函数在动态库 USBInterFace.dll 中，用来获取数据缓冲区首指针。数据缓冲区开辟的总大小为 20M 字节，可以使用多大受 setinfo 函数限定。缓冲区数据格式为每个字节代表一个采集电压，AB 通道交错排列为：A 通道，B 通道，A 通道，B 通道，A 通道，B 通道 ...

从指针中取得示波器通道的数据后，建议丢弃前 100 个数据不用，通常前 100 个数据不准确。

c#举例：

```
[DllImport("USBInterFace.dll", CharSet = CharSet.Auto, CallingConvention =
CallingConvention.StdCall)]
public static extern IntPtr GetBuffer4Wr ( Int32 index);
```

函数名： SetInfo

函数定义： void SetInfo(double p1, double p2, unsigned char p3, int p4, unsigned int p5, unsigned int bufferSize);

参数说明： double p1: 固定传 1 作为参数。
double p2: 固定传 0 作为参数。
unsigned char p3: 固定传 0x11 作为参数。
int p4: 固定传 0 作为参数。
unsigned int p5: 固定传 0 作为参数。

unsigned int bufferSize: 设置数据缓冲区的大小, 单位是字节。因为这个缓冲区是 AB 通道采集电压的字节型交错排列, 所以这个大小必须是偶数, 并且不能超过缓冲区总大小 20M 字节。

返回值: 无。

说明: 此函数在动态库 USBInterFace.dll 中, 用来设定数据缓冲区的大小, 就是 GetBuffer4Wr 函数所获取首指针的那个缓冲区。

c#举例:

```
[DllImport("USBInterFace.dll", CharSet = CharSet.Auto, CallingConvention =
CallingConvention.StdCall)]

public static extern void SetInfo(double dataNumPerPixar, double currentSampleRate,
byte ChannelMask, Int32 m_ZrroUnlnt, uint BufferOffset, uint HWbufferSize);
```

函数名: AiReadBulkData

函数定义: **unsigned long** AiReadBulkData(**unsigned long** SampleCount,
unsigned int num, **unsigned long** ulTimeout,
unsigned char* PBuffer, **unsigned char** flag,
unsigned int p1, **unsigned int** p2);

参数说明: **unsigned long** SampleCount: 采集多少字节的电压数据, 这个数值不能超过 SetInfo 所设定的缓冲区大小。

unsigned int num: 1。

unsigned long ulTimeout: 本次采集的超时时间设置, 单位为毫秒。

unsigned char* PBuffer: 数据缓冲区指针, 可直接使用 GetBuffer4Wr 获得的指针。

unsigned char flag : 固定传 0 作为参数。

unsigned int p1 : 固定传 0 作为参数。

unsigned int p2 : 固定传 0 作为参数。

返回值: 无。

说明: 此函数在动态库 USBInterFace.dll 中, 这个函数启动示波器的 ADC 芯片采集电压, 将转换的电压数据以双通道交错的字节形式排列到指定的数据缓冲区中。

c#举例:

```
[DllImport("USBInterFace.dll", CharSet = CharSet.Auto, CallingConvention =
CallingConvention.StdCall)]

public static extern Int32 AiReadBulkData(Int32 SampleCount, uint num, Int32 ulTimeout,
IntPtr PBuffer, byte flag, uint packetNum, uint eventNumTotal);
```


示波器零电压校准数据

示波器每个通道使用的是 8 bit 的 ADC，所以采集得到的数据是 0~255 的字节，代表电压的大小。在每个不同的电压量程档位下，0~255 代表着不同的电压范围。因为电压的量程是正负的，所以零电压并不是 0~255 的 0，而是靠近 128 的一个数值，大于这个数值是正电压，小于这个数值是负电压。由于每台示波器不可能完全做到参数相同，所以我们在出厂时已经对零电压数值进行了校准并固化在设备里。二次开发时，可以通过下列命令取到不同电压档位时的零电压校准数值来使用，比如取出一个数值是 132，代表着 0~255 的数值范围内，以 132 为中心零位置，本文统一称这个 132 为 **ZeroByte**。

```
unsigned char ZeroByte = USBCtrlTrans(0x90, usValue, 1);
```

该命令适用示波器型号：OSC482 OSC802 OSCA02 OSC2002 OSCH02

通道号	电压档位	Request	usValue
通道 A	1V/DIV	0x90	0x01
通道 B	1V/DIV		0x02
通道 A	500mV/DIV		0x0E
通道 B	500mV/DIV		0x0F
通道 A	200mV/DIV		0x14
通道 B	200mV/DIV		0x15
通道 A	100mV/DIV		0x12
通道 B	100mV/DIV		0x13
通道 A	50mV/DIV		0x10
通道 B	50mV/DIV		0x11
通道 A	20mV/DIV		0xA0
通道 B	20mV/DIV		0xA1

示波器电压幅值校准数据

示波器每个通道使用的是 8 bit 的 ADC，所以采集得到的数据是 0~255 的字节，代表电压的大小。上面已经阐明了不同电压档位下的零位置校准数据，还有一个校准参数是用来将 0~255 的字符数据转换成实际电压量。这个参数我们在本文中称为 **AltitudeByte**。我们通过命令码获取不同通道不同电压档位下对应的 **AltitudeByte**，如下所示：

```
unsigned char AltitudeByte = USBCtrlTrans(0x90, usValue, 1);
```

该命令适用示波器型号：OSC482 OSC802 OSCA02 OSC2002 OSCH02

通道号	电压档位	Request	usValue
-----	------	---------	---------

通道 A	1V/DIV	0x90	0x03
通道 B	1V/DIV		0x04
通道 A	500mV/DIV		0x08
通道 B	500mV/DIV		0x0B
通道 A	200mV/DIV		0x06
通道 B	200mV/DIV		0x07
通道 A	100mV/DIV		0x09
通道 B	100mV/DIV		0x0C
通道 A	50mV/DIV		0x0A
通道 B	50mV/DIV		0x0D
通道 A	20mV/DIV		0x2A
通道 B	20mV/DIV		0x2D

我们获取到电压幅值校准数据 **AltitudeByte** 以后，需要通过一个简单的公式转换成一个浮点型的系数，这个系数是用来校准幅值理论公式的。

double 电压校准系数 = (**double**) (**AltitudeByte***2) /255;

举例说明，我们在使用 200mV/DIV 的电压档位时，代表屏幕纵向每格 200mV。屏幕 0 电压在中间，上面有 5 格，下面有 5 格。所以这个档位的电压量程是-1V~1V，也就是 2V 的范围。理论上是 255 的跨度代表了 2V 的电压范围。所以我们会得出理论公式：

当前采样点理论电压数值 = (当前采样点 byte 数值-**ZeroByte**) * (2V/255) ;

当前采样点校准后电压数值 = 当前采样点理论电压数值 * 电压校准系数；

合并采集命令

部分产品支持合成采集，合成采集功能开启后，通道采样率提升 2 倍。例如：OSCA02E, 在 100Mhz 采样率前提下，开启合成采集功能，采样率可以达到 100Mhz * 2 = 200Mhz。

1. 开启合成采集功能前，chA 和 chB，两个通道必须全部打开。

2. 开启合成采集的命令：

g_CtrlByte1 |= 0x80;

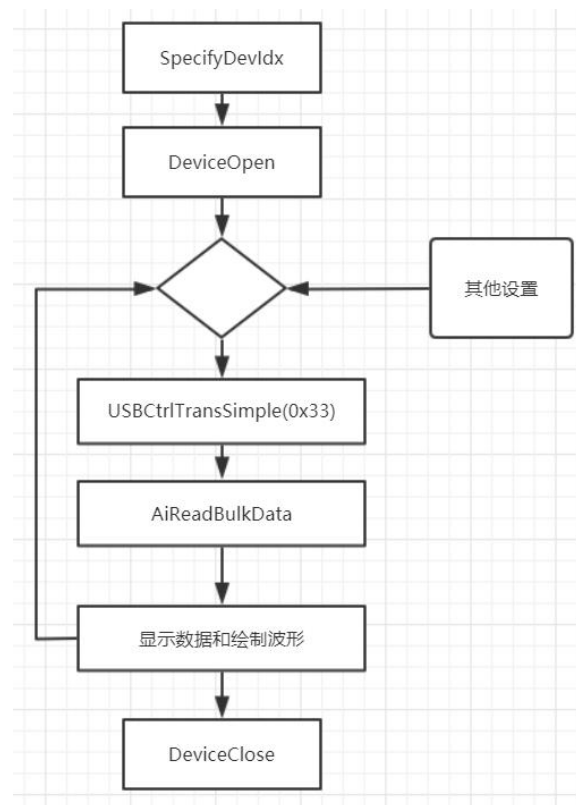
USBCtrlTrans(0x24, **g_CtrlByte1**, 1)

3. 关闭合成采集的命令：

g_CtrlByte1 &= 0x7f;

USBCtrlTrans(0x24, **g_CtrlByte1**, 1);

OSC482 的使用流程图:



OSCA02, OSC2002, OSCH02 的使用流程图:

