



 LOTO_OpenSource_sdk.pdf -> Secondary development SDK function description

 OSC482 series secondary development SDK (package)


1. OpenSource_LOTO_482 -> OSC482(C# winform) Demo example;
2. OpenSource_LOTO_482Signal -> OSC482 signal generator (C# winform) Demo example; Other models such as :
(OSCA02/OSC2002/OSCH02) can be referred to;
3. Python OSC482 -> OSC482(Python 32-bit) Demo example;
4. OpenSource_LOTO_482_python_QT -> OSC482(Python 32-bit + QT interface) Demo example;
5. LabVIEW_SubVI_LOTO_482 -> OSC482(LabVIEW 2016) Demo example;
6. MFC_MeasuredVal -> MFC (VC++) real-time
acquisition of oscilloscope software measurement value Demo example;

 OSC2002 series secondary development SDK (package)

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

 OSCA02 series secondary development SDK (package)


1. OpenSource_LOTO_A02 -> OSCA02(C# winform) Demo example;
2. Python OSCA02 -> OSCA02(Python 32-bit) Demo example;
3. OpenSource_LOTO_A02_Buffer -> OSCA02(stream mode large buffer 4M 8M) Demo example;
4. OpenSource_LOTO_A02-FFT_FIR -> OSCA02 (FFT frequency domain and FIR filtering) Demo example;
5. OpenSource_LOTO_A02_Phase -> OSCA02 calculate two channel waveform (phase difference) Demo example;
6. OpenSource_LOTO_A02Signal -> OSCA02 signal generator (C# winform) Demo example; other models such as :
(OSCA02/OSC2002/OSCH02) can be referred to;
7. OpenSource_LOTO_A02Charts -> OSCA02 multi-channel board (C# winform) Demo example;
8. OpenSource_LOTO_A02_FreqDuty -> OSCA02 calculation (frequency and duty cycle) (C#) Demo example;

 OSCH02 series secondary development SDK (package)

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

 OSC980 (dual channel) series secondary development SDK (package)

1. OpenSource_LOTO_980 -> OSC980(C# winform) Demo example;
2. Python OSC980 -> OSC980(Python 32-bit) Demo example;
3. OpenSource_LOTO_980_Buffer -> OSC980 (stream mode large buffer 4M 8M) Demo example;

 OSC984 (four channels) series secondary development SDK (package)

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

DLL dynamic link library name: USBInterFace.dll, this file may be different for different models, so be careful not to mix them.

Function name: SpecifyDevIdx

Function definition `void SpecifyDevIdx(int index);`

Parameter description: `int index`, a 32-bit integer, represents the device number.

Device model	index value
OSC802	0
OSC482	6
OSCA02	6
OSC2002	2
OSCH02	7

Return value: None

Note: This function is used to set the product number in the dynamic library USBInterFace.dll. The product number of different models may be different. It is necessary to call this function to set the correct number before the software opens the device. If the wrong device number is set, the software will not be able to open and operate the device correctly.

c# example:

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

Function name: DeviceOpen

Function definition `unsigned long DeviceOpen(void);`

Parameter description: None.

Return value: `unsigned long`, the return value reflects the result of the function call, the return value is -1, indicating that the function failed, and the return value is 0
Ming function succeeds.

Description: This function is used in the dynamic library USBInterFace.dll to open the oscilloscope device. If it is successfully opened, the handle of the device will be saved in the DLL and resources will be prepared for subsequent operations. Only then can we call other functions to set and operate the oscilloscope. This function can be opened only once in the early stage of program operation. Common failure reasons may be that the aforementioned SpecifyDevIdx function is set incorrectly, or the driver of the oscilloscope is not installed, or the oscilloscope device is not plugged in.

c# example:

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

Function name: DeviceClose

Function definition unsigned long DeviceClose(void);

Parameter description: None.

Return value: unsigned long, the return value reflects the result of the function call, the return value is -1, indicating that the function failed, and the return value is 0 function succeeds.

Description: This function is used in the dynamic library USBInterFace.dll to close the oscilloscope device. If it is successfully closed, the DLL will release the handle of the device, and clean up and release the occupied software resources. After we call other functions to set up and operate the oscilloscope, it will fail. Usually we only need to call it once at the end of your program.

c# example:

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

public static extern Int32 DeviceClose();
```

Function name: USBCtrlTransSimple

Function definition unsigned long USBCtrlTransSimple (unsigned long Request);

Parameter description: unsigned long Request: command code, different command codes represent different instructions. A table will be attached at the end of this function description to list the open command codes.

Return value: unsigned long, if it is a command that needs to obtain data from the lower computer, this return value will return the value sent by the lower computer.

Description: This function is in the dynamic library USBInterFace.dll, and uses the USB control transmission method to send commands to the oscilloscope of the lower computer. The premise of using it is that the oscilloscope device has been successfully opened by DeviceOpen. It can be a one-way command or a two-way command to obtain data.

c# example:

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

public static extern Int32 USBCtrlTransSimple(Int32 Request);
```

return value	Request	illustrate
/	0x33	Start collecting ADC data
A return value of 33 indicates that the collection and storage are complete Bi can read	0X50	Query whether the AD acquisition and storage is completed (not applicable to OSC482 series)

Function name: USBCtrlTrans

Function definition `unsigned char` USBCtrlTrans (`unsigned char` Request, `unsigned short` usValue, `unsigned long` outBufSize);

Parameter description: `unsigned char` Request: command code, different command codes represent different instructions. Later in this function description will

Attached is a table listing the open command codes.

`unsigned short` usValue: The parameter of the command code, which is passed to the DLL as the auxiliary information of the command code.

`unsigned long` outBufSize: Not used, just pass 1 as fixed.

Return value: `unsigned char`, if it is a command that needs to obtain data from the lower computer, this return value will return the value sent by the lower computer.

Description: This function is in the dynamic library USBInterFace.dll, and uses the USB control transmission method to send commands to the oscilloscope of the lower computer. The premise of using it is that the oscilloscope device has been successfully opened by DeviceOpen. It can be a one-way command or a two-way command to obtain data.

c# example:

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

```
public static extern byte USBCtrlTrans(byte Request, UInt16 usValue, uint outBufSize);
```

Instructions of OSC482 series:

return value	Request	usValue	illustrate
/	0x94	0x10	set 50MHz sampling rate
/		0x06	Set 4MHz sampling rate
/		0x01	set 2.4MHz sampling rate
/		0x07	set 500kHz sampling rate
/		0x11	Set 240kHz sampling rate
/	0x22	0x02	chA input range setting: -5V ~ +5V
/		0x02	chA input range setting: -1V ~ +1V
/		0x06	chA input range setting: -2.5V ~ +2.5V
/		0x04	chA input range setting: -500mV ~ +500mV
/		0x08	chA input range setting: -250mV ~ +250mV
/	0x23	0x02	chB input range setting: -5V ~ +5V
/		0x12	chB input range setting: -2.5V ~ +2.5V
/		0x00	chB input range setting: -1V ~ +1V
/		0x10	chB input range setting: -500mV ~ +500mV
/		0x20	chB input range setting: -250mV ~ +250mV
/	0x24	0x08	Set chA as DC coupling
/		0x00	Set chA as AC coupling
/	0x25	0x01	Set chB as DC coupling

/		0x00	set chB as AC coupling
/	0x84	0x00	Two commands set chB to be used as oscilloscope channel
/	0x81	0x00	
/	0x84	0x00	Two commands set chB to be used as a logic analyzer, each BIT represents a logical channel
/	0x82	0x00	

Instructions of OSCA02, OSC2002, OSCH02 series:

8-bit control byte g_CtrlByte0:

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
			chA AC/DC	Fre3	Fri2	Fre1	Fre0

8-bit control byte g_CtrlByte1:

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

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

synthetic capture		
Order	OSCA02E, OSCH02 chA	
g_CtrlByte1 = 0x80; USBCtrlTrans(0x24, g_CtrlByte1, 1)	turn on	channel = sampling rate * 2 times
g_CtrlByte1 &= 0x7f; USBCtrlTrans(0x24, g_CtrlByte1, 1)	closure	condition: chB channel needs to be turned on

Order	OSCA02, OSC2002, OSCH02
g_CtrlByte1 &= 0xF7; g_CtrlByte1 = 0x08; USBCtrlTrans(0x22, 0x00, 1); USBCtrlTrans(0x24, g_CtrlByte1, 1);	chA input range setting: -8V ~ +8V
g_CtrlByte1 &= 0xF7; g_CtrlByte1 = 0x08; USBCtrlTrans(0x22, 0x02, 1); USBCtrlTrans(0x24, g_CtrlByte1, 1);	chA input range setting: -5V ~ +5V

g_CtrlByte1 &= 0xF7; g_CtrlByte1 = 0x08 USBCtrlTrans(0x22, 0x04, 1); USBCtrlTrans(0x24, g_CtrlByte1, 1);	chA input range setting: -2.5V ~ +2.5V
g_CtrlByte1 &= 0xF7; g_CtrlByte1 = 0x08; USBCtrlTrans(0x22, 0x06, 1); USBCtrlTrans(0x24, g_CtrlByte1, 1);	chA input range setting: -1V ~ +1V
g_CtrlByte1 &= 0xF7; USBCtrlTrans(0x22, 0x02, 1); USBCtrlTrans(0x24, g_CtrlByte1, 1);	chA input range setting: -500mV ~ +500mV
g_CtrlByte1 &= 0xF7; USBCtrlTrans(0x22, 0x04, 1); USBCtrlTrans(0x24, g_CtrlByte1, 1);	chA input range setting: -250mV ~ +250mV
g_CtrlByte1 &= 0xF7; USBCtrlTrans(0x22, 0x06, 1); USBCtrlTrans(0x24, g_CtrlByte1, 1);	chA input range setting: -100mV ~ +100mV
g_CtrlByte1 &= 0xF9; USBCtrlTrans(0x23, 0x00, 1); USBCtrlTrans(0x24, g_CtrlByte1, 1);	chB input range setting: -8V ~ +8V
g_CtrlByte1 &= 0xF9; g_CtrlByte1 = 0x02; USBCtrlTrans(0x23, 0x00, 1); USBCtrlTrans(0x24, g_CtrlByte1, 1);	chB input range setting: -5V ~ +5V
g_CtrlByte1 &= 0xF9; g_CtrlByte1 = 0x04; USBCtrlTrans(0x23, 0x00, 1); USBCtrlTrans(0x24, g_CtrlByte1, 1);	chB input range setting: -2.5V ~ +2.5V
g_CtrlByte1 &= 0xF9; g_CtrlByte1 = 0x06; USBCtrlTrans(0x23, 0x00, 1); USBCtrlTrans(0x24, g_CtrlByte1, 1);	chB input range setting: -1V ~ +1V
g_CtrlByte1 &= 0xF9; g_CtrlByte1 = 0x02; USBCtrlTrans(0x23, 0x40, 1); USBCtrlTrans(0x24, g_CtrlByte1, 1);	chB input range setting: -500mV ~ +500mV
g_CtrlByte1 &= 0xF9; g_CtrlByte1 = 0x04; USBCtrlTrans(0x23, 0x40, 1); USBCtrlTrans(0x24, g_CtrlByte1, 1);	chB input range setting: -250mV ~ +250mV
g_CtrlByte1 &= 0xF9; g_CtrlByte1 = 0x06; USBCtrlTrans(0x23, 0x40, 1); USBCtrlTrans(0x24, g_CtrlByte1, 1);	chB input range setting: -100mV ~ +100mV
g_CtrlByte0 &= 0xef; g_CtrlByte0 = 0x10; USBCtrlTrans(0x94, g_CtrlByte0, 1);	Set chA to DC coupling
g_CtrlByte0 &= 0xef; USBCtrlTrans(0x94, g_CtrlByte0, 1);	Set chA to AC coupling
g_CtrlByte1 &= 0xef; g_CtrlByte1 = 0x10; USBCtrlTrans(0x24, g_CtrlByte1, 1);	Set chB to DC coupling
g_CtrlByte1 &= 0xef; g_CtrlByte1 = 0x00; USBCtrlTrans(0x24, g_CtrlByte1, 1);	Set chB to AC coupling
USBCtrlTrans(0xE7, 0x00, 1);	Disable trigger and disable external trigger
USBCtrlTrans(0xE7, 0x01, 1);	Enable trigger and enable external trigger
USBCtrlTrans(0xC5, 0x00, 1);	Rising edge trigger
USBCtrlTrans(0xC5, 0x01, 1);	Falling edge trigger
USBCtrlTrans(0x16, 128, 1);	Set trigger level to 128, non-external trigger is valid

If the external trigger function is required, contact the mfr to request before purchasing. When triggering externally, the set trigger level is invalid.	external trigger
g_CtrlByte1 &= 0xfe; g_CtrlByte1 = 0x01; USBCtrlTrans(0x24, g_CtrlByte1, 1);	chB on
g_CtrlByte1 &= 0xfe; g_CtrlByte1 = 0x00; USBCtrlTrans(0x24, g_CtrlByte1, 1);	chB off

capture card mode	
Order	OSCA02, OSC2002, OSCH02
g_CtrlByte0 &= 0xf0; g_CtrlByte0 = 0x06; USBCtrlTrans(0x94, g_CtrlByte0, 1);	Set 4M sampling rate
g_CtrlByte0 &= 0xf0; g_CtrlByte0 = 0x05; USBCtrlTrans(0x94, g_CtrlByte0, 1);	Set 2.4M sampling rate
g_CtrlByte0 &= 0xf0; g_CtrlByte0 = 0x07; USBCtrlTrans(0x94, g_CtrlByte0, 1);	Set 500K sample rate

Function name: GetBuffer4Wr

Function definition `unsigned char* GetBuffer4Wr (int index);`

Parameter description: `int` index: fixedly pass -1 as a parameter.

Return value: `unsigned char*`, returns the first pointer of a buffer, as long as it is not NULL, it can be used.

Description: This function is used in the dynamic library USBInterFace.dll to obtain the first pointer of the data buffer. The total size of the data buffer opened is 20M bytes, how much can be used is limited by the setinfo function. The buffer data format is that each byte represents an acquisition voltage, and the AB channels are staggered as: A channel, B channel, A channel, B channel, A channel, B channel...

After obtaining the data of the oscilloscope channel from the pointer, it is recommended to discard the first 100 data, usually the first 100 data is not accurate.

c# example:

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

```
public static extern IntPtr GetBuffer4Wr ( Int32 index);
```

Function name: SetInfo

Function definition: `void SetInfo(double p1, double p2, unsigned char p3, int p4, unsigned int p5, unsigned int bufferSize);`

Parameter description: `double` p1: fixedly pass 1 as a parameter.

`double` p2: always pass 0 as a parameter.

`unsigned char` p3: fixedly pass 0x11 as a parameter. `int` p4: fixedly

pass 0 as a parameter. `unsigned int` p5:

fixedly pass 0 as a parameter.

unsigned int bufferSize: Set the size of the data buffer in bytes. Because the buffer is a byte-type interleaved arrangement of the voltages collected by the AB channel, the size must be an even number and cannot exceed the total buffer size of 20M bytes.

Return value: None.

Description: This function is used to set the size of the data buffer in the dynamic library USBInterFace.dll, which is the GetBuffer4Wr function. The buffer whose first pointer is obtained.

c# example:

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

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

Function name: AiReadBulkData

Function definition: **unsigned long** AiReadBulkData(**unsigned long** SampleCount, **unsigned int** num, **unsigned long** ulTimeout, **unsigned char*** PBuffer, **unsigned char** flag, **unsigned int** p1, **unsigned int** p2);

Parameter description: **unsigned long** SampleCount: How many bytes of voltage data to collect, this value cannot exceed the buffer size set by SetInfo.

unsigned int num: 1.

unsigned long ulTimeout: The timeout setting for this collection, in milliseconds.

unsigned char* PBuffer: data buffer pointer, which can be obtained directly by using GetBuffer4Wr.

unsigned char flag : fixedly pass 0 as a parameter.

unsigned int p1 : fixedly pass 0 as a parameter.

unsigned int p2 : fixedly pass 0 as a parameter.

Return value: None.

Description: This function is in the dynamic library USBInterFace.dll, this function starts the ADC chip of the oscilloscope to collect the voltage, and converts the voltage data. The voltage data is arranged in the specified data buffer in the form of two-channel interleaved bytes.

c# example:

```
[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);
```


Oscilloscope Zero Voltage Calibration Data

Each channel of the oscilloscope uses an 8-bit ADC, so the collected data is 0~255 bytes, representing the magnitude of the voltage. Under each different voltage range gear, 0~255 represents a different voltage range. Because the voltage range is positive and negative, the zero voltage is not 0 from 0 to 255, but a value close to 128. A value greater than this value is a positive voltage, and a value smaller than this value is a negative voltage. Since it is impossible for every oscilloscope to have exactly the same parameters, we have calibrated the zero voltage value and solidified it in the device before leaving the factory. During the secondary development, you can use the following commands to get the zero voltage calibration value at different voltage levels. For example, if you take out a value of 132, it means that within the range of 0~255, the zero position is centered on 132, which is unified in this article. Call this 132 ZeroByte.

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

This command is applicable to oscilloscope models: OSC482 OSC802 OSCA02 OSC2002 OSCH02

Channel No.	Voltage range	Request	usValue
Channel A	1V/DIV	0x90	0x01
Channel B	1V/DIV		0x02
Channel A	500mV/DIV		0x0E
Channel B	500mV/DIV		0x0F
Channel A	200mV/DIV		0x14
Channel B	200mV/DIV		0x15
Channel A	100mV/DIV		0x12
Channel B	100mV/DIV		0x13
Channel A	50mV/DIV		0x10
ChanB	50mV/DIV		0x11
Channel A	20mV/DIV		0xA0
Channel B	20mV/DIV		0xA1

Oscilloscope Voltage Amplitude Calibration Data

Each channel of the oscilloscope uses an 8-bit ADC, so the collected data is 0~255 bytes, representing the magnitude of the voltage. The above has clarified the zero position calibration data under different voltage gears, and there is another calibration parameter used to convert the character data of 0~255 into the actual voltage. We call this parameter AltitudeByte in this article. We obtain the AltitudeByte corresponding to different channels and different voltage levels through the command code, as follows:

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

This command is applicable to oscilloscope models: OSC482 OSC802 OSCA02 OSC2002 OSCH02

channel number	Voltage range	Request	usValue
----------------	---------------	---------	---------

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

After we get the voltage amplitude calibration data `AltitudeByte`, we need to convert it into a floating-point system through a simple formula Number, this coefficient is used to calibrate the amplitude theory formula.

Double voltage calibration factor = (double) (`AltitudeByte`*2)/255;

For example, when we use the voltage range of 200mV/DIV, it means 200mV per division in the vertical direction of the screen. The 0 voltage of the screen is in the middle, with 5 grids above and 5 grids below. So the voltage range of this gear is -1V~1V, that is, the range of 2V. Theoretically, a span of 255 represents a voltage range of 2V. So we get the theoretical formula:

Current sampling point theoretical voltage value = (current sampling point byte `value-ZeroByte`) * (2V/255);

The voltage value of the current sampling point after calibration = the theoretical voltage value of the current sampling point * voltage calibration coefficient;

Merge acquisition command

Some products support synthetic acquisition. After the synthetic acquisition function is enabled, the channel sampling rate is increased by 2 times. Example: OSCA02E, at 100Mhz Under the premise of sampling rate, turn on the synthetic acquisition function, and the sampling rate can reach 100Mhz * 2 = 200Mhz.

1. Before enabling the synthetic acquisition function, both chA and chB must be turned on. 2.

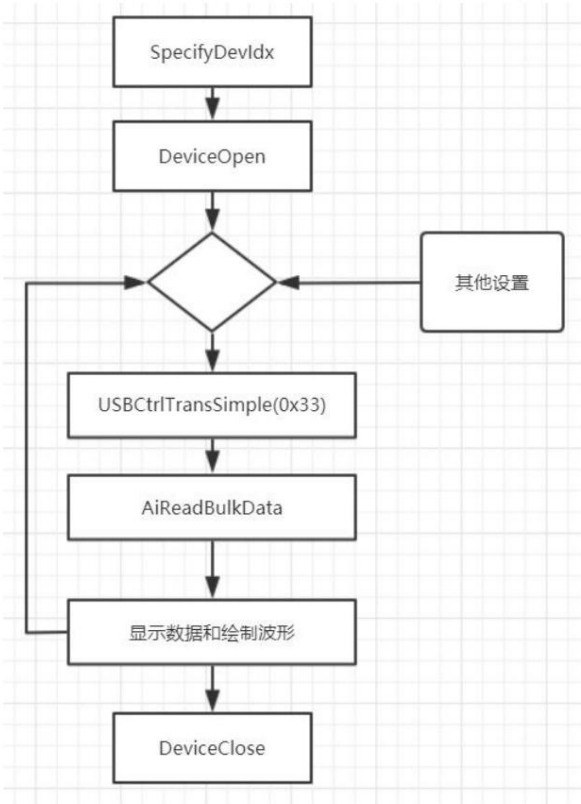
Command to enable synthetic capture:

```
g_CtrlByte1 |= 0x80;
USBCtrlTrans(0x24, g_CtrlByte1, 1)
```

3. Command to turn off synthetic acquisition:

```
g_CtrlByte1 &= 0x7f;
USBCtrlTrans(0x24, g_CtrlByte1, 1);
```

The flow chart of using OSC482:



OSCA02, OSC2002, OSCH02 usage flow chart:

