SDK API User Guide

Version 1.11

Vimu Electronic Technology

2025-05-10

http://www.vimu.top/

Update Log

V1.0 (2023.3.31)

1, New design

V1.1 (2023.5.22)

1, Add Logic Channel Trigger Source support

V1.2 (2023.6.26)

1, Add Watchdog Ctrl

V1.3 (2023.8.20)

1, Added MSO21 device support

2, Add DDS ARB and gating API

V1.4 (2023.11.06)

1, Linux system to increase stability

2, DLLTest demo supports re-insertion, automatic connection and acquisition

V1.5 (2023.12.14)

1, Add MSO10 and MSO21 V2 device support

2, Linux support read more than 4MB

V1.6 (2024.03.17)

1, DDS asdd brust APIs

V1.7 (2024.05.15)

1, Add trigger point read APIs

V1.8 (2024.07.15)

1, MSO41 support

V1.9 (2024.09.29)

1, MSO21 V3 和 MSO10 V2 support

V1.10 (2024.11.21)

1, Fixed not all channel acquisition and reading data misalignments

2,Fixed USB 3.0 linux bugs

V1.11 (2025.03.09)

1, Increased system stability

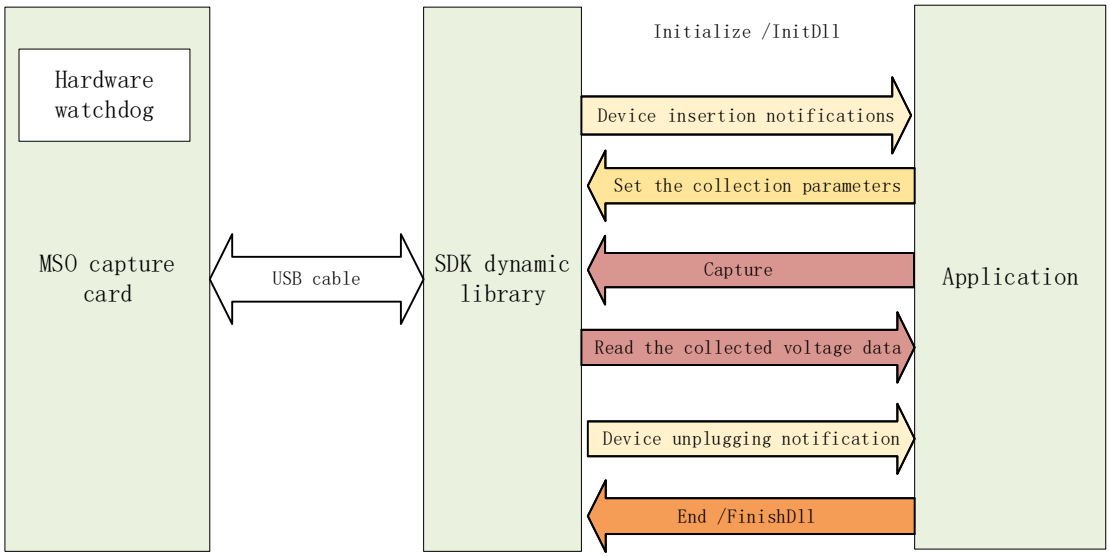
Catalog

1. Introduction

As a standard DLL interface for MOS mixed-signal oscilloscopes, mixed-signal oscilloscopes can be controlled directly.

The interface supports widows systems (X86, X64 and arm64) and linux systems (X64, arm-linux-gnueabi, arm-linux-gnueabihf, and aarch64-linux).

2. System working block diagram



MSO capture card: The hardware capture card part communicates with the host computer via USB

SDK dynamic library: Responsible for communicating with hardware; The parameters of the application are transmitted to the hardware card, and the acquisition data of the hardware card is returned to the application

Application： User-designed programs

Illustrate:

1. The parameter **watchdog enable** of InitDll can start the watchdog of the capture card, and remember to start the watchdog of the last delivered program. When debugging, the watchdog can be turned off as needed;
2. The SDK dynamic library thread is a separate thread. When using the callback mode, complex tasks cannot be handled in the callback function. If the callback function takes up too long and exceeds the threshold time of the watchdog, the hardware capture card will be restarted.

   The working mode of the capture card can be roughly divided into "cycle detection" and "callback function" mode. When developing applications, you can choose according to actual needs, or you can mix them according to actual needs.

3. Cyclic detection mode

In the cyclic detection mode, the status of the capture card is queried through API polling, and then processed accordingly.

DllTest-Polling is an example of a circular detection. Here's the code:

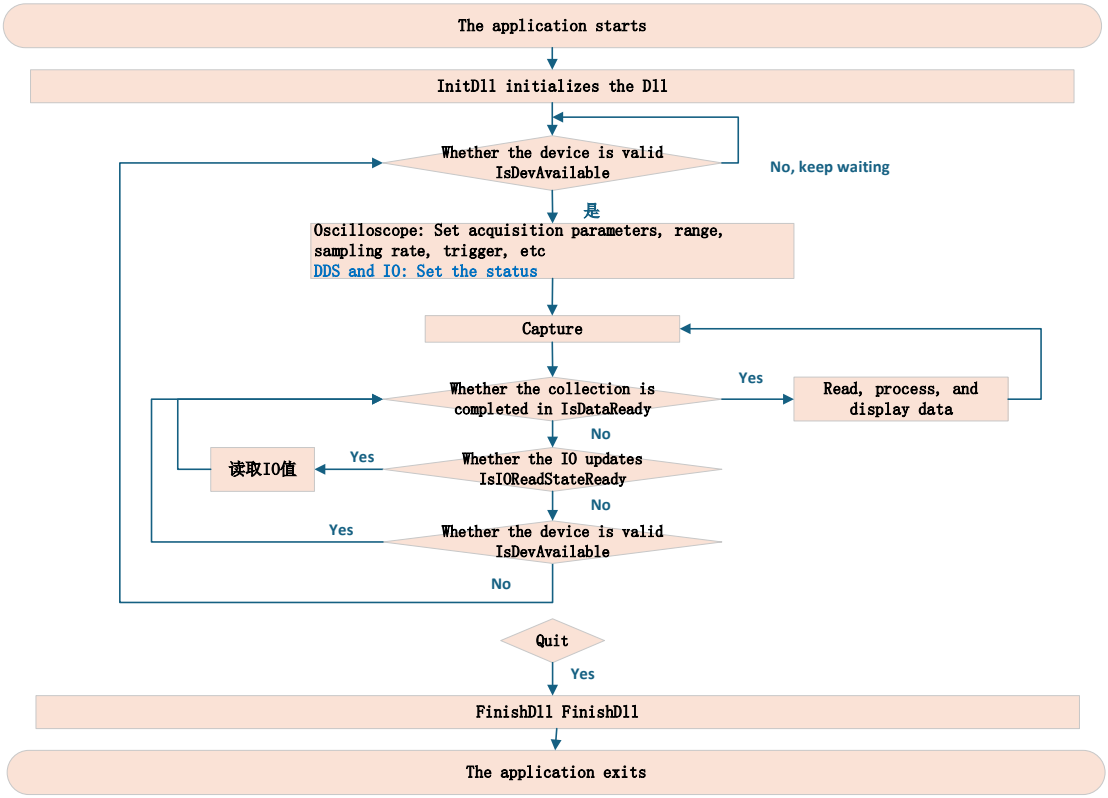
```
int main()
{
    std::cout << "Vdso Test..." << std::endl;
    InitDll(1, 1);
    std::this_thread::sleep_for(std::chrono::milliseconds(500));

    while (true)
    {
        //device connection is successful?
        if(!devAvailable)
        {
            devAvailable = IsDevAvailable();

            //init functions
            if(devAvailable)
            {
                std::cout << "devAvailable start init" << std::endl;
                initFunction();
            }
            else
                std::this_thread::sleep_for(std::chrono::milliseconds(500));
        }

        if(devAvailable)
        {
            if (IsDataReady())
            {
                ReadDatas();
                NextCapture();
            }
            else if(IsIOReadStateReady())
            {
                std::cout << "io state " << std::hex << GetIOInState() <<" \n";
            }
            else
            {
                //test device is active
                devAvailable = IsDevAvailable();
                if(!devAvailable)
                    std::cout << "devAvailable is false" << std::endl;

                std::this_thread::sleep_for(std::chrono::milliseconds(200));
            }
        }
```

```
    };
    FinishDll();
    std::cout << "...Vdso Test" << std::endl;
    return 0;
}
```

At first, check whether the device is plugged in, if not, the thread sleeps for 500ms, and then re-detects. When the device is detected, the device is initialized and Capture is invoked to start collecting data.

Then, start polling to check whether the oscilloscope has completed the acquisition and whether the IO has been updated. If it is not updated, by the way, check whether the device is still plugged in.

The oscilloscope or IO has been updated, and the corresponding processing work will be done on the reading data, and then the next acquisition will be carried out. If a device is detected to be unplugged, it re-enters the device detection poll.



4. Callback function mode

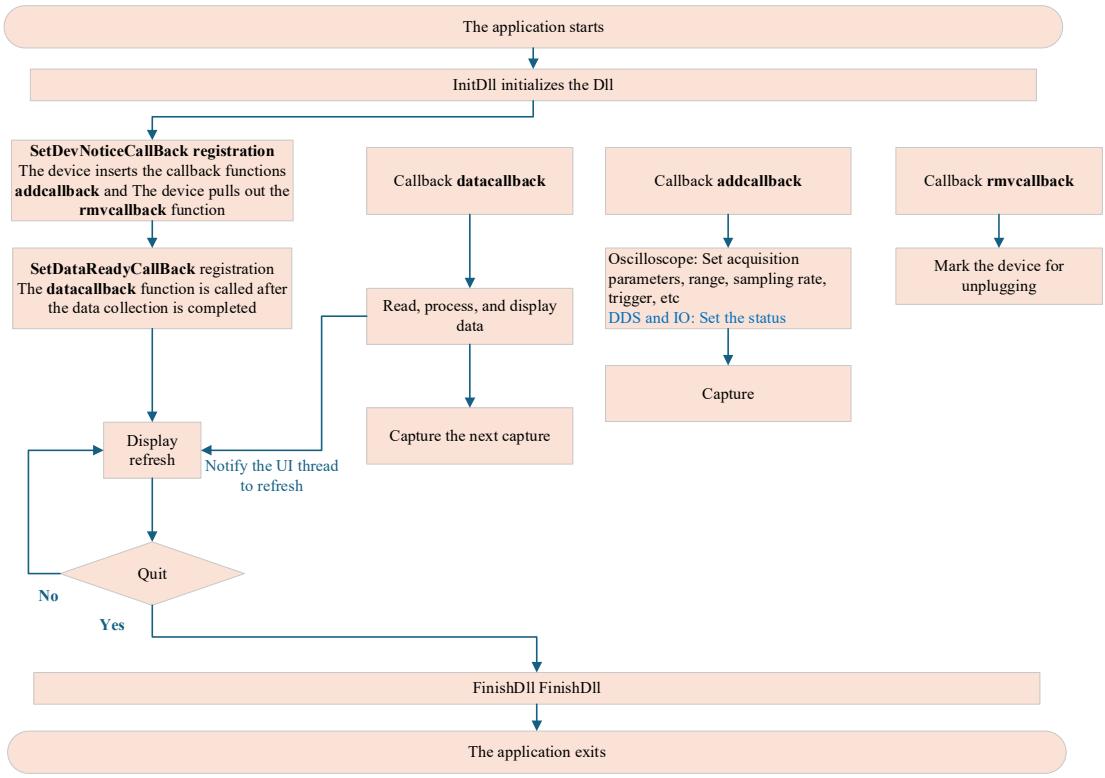
The callback function mode is to register the callback function to obtain the status of the capture card, and then process it accordingly.

The callback function mode involves the synchronization of the capture card thread and the UI thread, and the synchronization between threads needs to be handled according to the actual situation when developing the program.

DllTest-Callback is an example of a callback function. Here's the code:

```
int main()
{
```

```
    std::cout << "Vdso Test..." << std::endl;

    InitDll(1, 1);
    std::this_thread::sleep_for(std::chrono::milliseconds(500));

    //OSC
    SetDevNoticeCallBack(NULL, DevNoticeAddCallBack, DevNoticeRemoveCallBack);
    SetDataReadyCallBack(NULL, DevDataReadyCallBack);

    while (true)
    {
        std::this_thread::sleep_for(std::chrono::milliseconds(100));
    };

    FinishDll();
    std::cout << "...Vdso Test" << std::endl;
    return 0;
}
```



The callback function **addcallback**: is responsible for initializing each function after the device is inserted, and then starts the collection.

The callback function **rmcallback** is responsible for marking and handling the status of the device after it is unplugged.

The callback function **datacallback** is used to read the collected data and then perform the next collection.

5. API

5.1. Initialization and finish

Call InitDll () to complete the initialization of dynamic library, initialize memory and resources allocated for equipment monitoring and data reading.

int InitDll(unsigned int en_log , unsigned int en_hard_watchdog);

Description Dll initialization

Input: **log enable** 1 Enable Log
0 Not Enable Log
watchdog enable 1 Enable hard watchdog
0 Not Enable hard watchdog

Output: **Init Status**
Return value 1 Success
0 Failed

Call FinishDll () to complete the end of a dynamic library, freeing memory initialization and related resources in the application.

int FinishDll(void);

Description Dll finished

Input: -

Output: -**Finished Status**
Return value 1 Success
0 Failed

5.2. Equipment ID

The device ID is a 64-bit integer.

int GetOnlyId0(void);

Description This routines return device id(0-31)

Input: -

Output: - **Device ID(0-31)**

int GetOnlyId1(void);

Description This routines return device id(32-63)

Input: -

Output: - **Device ID(32-63)**

5.3. Equipment Reset

int ResetDevice(void);

Description This routines reset device

Input: -

Output: - **Return value** 1 success
0 failed

5.4. Equipment Monitor

when the device is detected, the dll have three ways to notify the main program, callback function, set Event and the main program loop detection.

5.4.1. callback function

When equipment is detect a function **"addcallback"** in the application can be called; when equipment is removed a function **"rmvcallback"** in the application can be called. The DLL has a function pointer which has to be set to these function, using

void SetDevNoticeCallBack(void* ppara, AddCallBack addcallback, RemoveCallBack rmvcallback);

Description This routines sets the callback function of equipment status changed.

Input: **ppara** the parameter of the callback function

addcallback a pointer to a function with the following prototype:
void AddCallBack(void * **ppara**)

rmvcallback a pointer to a function with the following prototype:
Void RemoveCallBack(void * **ppara**)

Output -

5.4.2. Event

When equipment is detect, an event **"addevent"** can be set by the DLL; when equipment is removed, an event **"rmvevent"** can be set by the DLL. The user must reset the event when the used them. The DLL has a function pointer which has to be set to these event using

void SetDevNoticeEvent(HANDLE addevent, HANDLE rmvevent);

Description This routines set the event handle, these will be set, when equipment status changed.

Input: **addevent** the event handle

rmvevent the event handle

Output -

5.4.3. Loop Detect

int IsDevAvailable();

Description This routines return the device is available or not.

Input: -

Output **Return value** 1 available
0 not available

Note: Only need to use one of three ways. Callback and Event functions are asynchronous, more efficient; main program loop detection over a certain time needed to detect whether the device is inserted or removed.

5.5. Oscillograph

5.5.1. Capture Range Set

Device with a programmable gain amplifier, when the signal acquisition time is less than the AD range,the signal amplification gain amplifier to use more AD digits, improving the quality of signal acquisition. Dll will adjusted the range of settings according to the pre-gain amplifier automatically.

int SetOscChannelRange(int channel, int minmv, int maxmv);

Description This routines set the range of input signal.

Input: **channel** the set channel

0 channel 1

1 channel 2

minmv the minimum voltage of the input signal (mV)

maxmv the maximum voltage of the input signal (mV)

Output **Return value** 1 Success

0 Failed

Note：The maximum range of the probe collection X1, the maximum voltage oscilloscope can capture. Like MSO20 is[-12000mV,12000mV].

Note：In order to achieve better waveform，you need to set the acquisition range, based on the magnitude of the measured waveform. When necessary, you can dynamically change the acquisition range.

5.5.2. Sample

int GetOscSupportSampleNum();

Description This routines get the number of samples that the equipment support.

Input: -

Output **Return value** the support sample number

int GetOscSupportSamples(unsigned int* sample, int maxnum);

Description This routines get support samples of equipment.

Input: **sample** the array store the support samples of the equipment

maxnum the length of the array

Output **Return value** the sample number of array stored

int SetOscSample(unsigned int sample);

Description This routines set the sample.

Input: **sample** the set sample

Output **Return value** 0 Failed

other value new sample

5.5.3. Trigger(hardware trigger)

This feature requires hardware trigger support. The hardware trigger point is the intermediate data, such as the acquisition of 128K data, trigger point is the 64K point.

Trigger Mode

```
#define TRIGGER_MODE_AUTO 0
#define TRIGGER_MODE_LIANXU 1
```

Trigger Style

```
#define TRIGGER_STYLE_NONE 0x0000          //not trigger
#define TRIGGER_STYLE_RISE_EDGE 0x0001     //Rising edge
#define TRIGGER_STYLE_FALL_EDGE 0x0002     //Falling edge
#define TRIGGER_STYLE_EDGE 0x0004          //Edge
#define TRIGGER_STYLE_P_MORE 0x0008        //Positive Pulse width(>)
#define TRIGGER_STYLE_P_LESS 0x0010        //Positive Pulse width(>)
#define TRIGGER_STYLE_P      0x0020        //Positive Pulse width(<>)
#define TRIGGER_STYLE_N_MORE 0x0040        //Negative Pulse width(>)
```

#define TRIGGER_STYLE_N_LESS 0x0080 //Negative Pulse width(>)
#define TRIGGER_STYLE_N 0x0100 //Negative Pulse width(<>)

int IsSupportHardTrigger();

Description This routines get the equipment support hardware trigger or not .

Input: **-**

Output **Return value** 1 support hardware trigger
0 not support hardware trigger

unsigned int GetTriggerMode();

Description This routines get the trigger mode.

Input: **-**

Output **Return value** TRIGGER_MODE_AUTO
TRIGGER_MODE_LIANXU

void SetTriggerMode(unsigned int mode);

Description This routines set the trigger mode.

Input: **mode** TRIGGER_MODE_AUTO
TRIGGER_MODE_LIANXU

Output -

unsigned int GetTriggerStyle();

Description This routines get the trigger style.

Input: **-**

Output **Return value** TRIGGER_STYLE_NONE
TRIGGER_STYLE_RISE_EDGE
TRIGGER_STYLE_FALL_EDGE
TRIGGER_STYLE_EDGE
TRIGGER_STYLE_P_MORE
TRIGGER_STYLE_P_LESS
TRIGGER_STYLE_P
TRIGGER_STYLE_N_MORE
TRIGGER_STYLE_N_LESS
TRIGGER_STYLE_N

void SetTriggerStyle(unsigned int style);

Description This routines set the trigger style.

Input: **style** TRIGGER_STYLE_NONE
TRIGGER_STYLE_RISE_EDGE
TRIGGER_STYLE_FALL_EDGE
TRIGGER_STYLE_EDGE
TRIGGER_STYLE_P_MORE
TRIGGER_STYLE_P_LESS
TRIGGER_STYLE_P
TRIGGER_STYLE_N_MORE

TRIGGER_STYLE_N_LESS
TRIGGER_STYLE_N

Output -

int GetTriggerPulseWidthNsMin();

Description This routines get the min time of pulse width.

Input: -

Output Return min time value of pulse width(ns)

int GetTriggerPulseWidthNsMax();

Description This routines get the max time of pulse width.

Input: -

Output Return max time value of pulse width(ns)

int GetTriggerPulseWidthDownNs();

Description This routines get the down time of pulse width.

Input: -

Output Return down time value of pulse width(ns)

int GetTriggerPulseWidthUpNs();

Description This routines set the down time of pulse width.

Input: down time value of pulse width(ns)

Output -

void SetTriggerPulseWidthNs(int down_ns, int up_ns);

Description This routines set the up time of pulse width.

Input: up time value of pulse width(ns)

Output _

unsigned int GetTriggerSource();

Description This routines get the trigger source.

Input: **-**

Output **Return value** TRIGGER_SOURCE_CH1 0 //CH1
TRIGGER_SOURCE_CH2 1 //CH2
TRIGGER_SOURCE_LOGIC0 16 //Logic 0
TRIGGER_SOURCE_LOGIC1 17 //Logic 1
TRIGGER_SOURCE_LOGIC2 18 //Logic 2
TRIGGER_SOURCE_LOGIC3 19 //Logic 3
TRIGGER_SOURCE_LOGIC4 20 //Logic 4
TRIGGER_SOURCE_LOGIC5 21 //Logic 5
TRIGGER_SOURCE_LOGIC6 22 //Logic 6
TRIGGER_SOURCE_LOGIC7 23 //Logic 7

void SetTriggerSource(unsigned int source);

Description This routines set the trigger source.

Input: **source** TRIGGER_SOURCE_CH1 0 //CH1
TRIGGER_SOURCE_CH2 1 //CH2
TRIGGER_SOURCE_LOGIC0 16 //Logic 0
TRIGGER_SOURCE_LOGIC1 17 //Logic 1
TRIGGER_SOURCE_LOGIC2 18 //Logic 2
TRIGGER_SOURCE_LOGIC3 19 //Logic 3
TRIGGER_SOURCE_LOGIC4 20 //Logic 4
TRIGGER_SOURCE_LOGIC5 21 //Logic 5
TRIGGER_SOURCE_LOGIC6 22 //Logic 6
TRIGGER_SOURCE_LOGIC7 23 //Logic 7

Output -

Note: If the logic analyzer and IO are multiplexed (for example, MSO20, MSO21), the corresponding IO needs to be turned on and set to the input state.

int GetTriggerLevel();

Description This routines get the trigger level.

Input: **-**

Output **Return value** level (mV)

void SetTriggerLevel(int level);

Description This routines set the trigger level.

Input: level (mV)

Output

int IsSupportTriggerSense();

Description This routines get the equipment support trigger sense or not.

Input: **-**

Return value 1 support
0 not support

int GetTriggerSenseDiv();

Description This routines get the trigger sense.

Input: **-**

Output **Return value** Sense (0-1 div)

void SetTriggerSenseDiv(int sense);

Description This routines set the trigger sense.

Input: Sense (0-1 div)

Output -

Note：The Range of Trigger Sense is 0.1 Div-1.0 Div. 1 Div =(Sample Range Max Value-Sample Range Min Value)/10.0. For example, Sample Range is [-1000,1000]mV, 1Div=(1000--1000)/10.0
=200mV.

bool IsSupportPreTriggerPercent();

Description This routines get the equipment support Pre-trigger Percent or not .

Input: -

Output Return value 1 support

0 not support

int GetPreTriggerPercent();

Description This routines get the Pre-trigger Percent.

Input: -

Output Return value Percent (5-95)

void SetPreTriggerPercent(int front);

Description This routines set the Pre-trigger Percent.

Input: Percent (5-95)

Output -

int IsSupportTriggerForce();

Description This routines get the equipment support trigger force or not.

Input: **-**

Return value 1 support

0 not support

void TriggerForce();

Description This routines force capture once.

Input: **-**

Output: **-**

5.5.4. AC/DC

int IsSupportAcDc(unsigned int channel);

Description This routines get the device support AC/DC switch or not.

Input: channel 0 :channel 1

1 :channel 2

Output **Return value** 0 : not support AC/DC switch

1 : support AC/DC switch

void SetAcDc(unsigned int channel, int ac);

Description This routines set the device AC coupling.

Input: channel 0 :channel 1

1 :channel 2

ac 1 : set AC coupling

0 : set DC coupling

Output **-**

int GetAcDc(unsigned int channel,);

Description This routines get the device AC coupling.

Input: channel 0 :channel 1
1 :channel 2

Output **Return value** 1 : AC coupling
0 : DC coupling

5.5.5. Capture

Call capture function to begin collecting data, **length** is the length you want to capture, using K Units, such as length = 10, is 10K 10240 points. For sample rate greater than or equal the length of the depth of the collection is stored, take the minimum **length** and depth of storage;For the sampling rate is less than the memory depth, take the minimum **length** and one second data collection length. **force_length** can be forced to cancel the limit of only 1 seconds to be collected.

int Capture(int length, char capture_channel,char force_length);

Description This routines set the capture length and start capture.

Input: **length** capture length(KB)
capture_channel
ch1=0x0001 ch2=0x0002 ch3=0x0004 ch4=0x0008 logic=0x0100
ch1+ch2 0x0003
ch1+ch2+ch3 0x0007
ch1+logic 0x0101
force_length 1: force using the length, no longer limits the max collection 1 seconds

Output **Return value** the real capture length(KB)

When using normal trigger mode (TRIGGER_MODE_LIANXU). The collection command was sent, and the data notification that the collection was complete has not been received. Now, you want to stop the software.

1. Recommended method: You change the trigger mode to TRIGGER_MODE_AUTO, wait for the data notification to be collected, and then stop the software.

2. Use **AbortCapture.**

DLL_API int WINAPI AbortCapture();

Description This routines set the abort capture

Input:

Output Return value 1:success 0:failed

unsigned int GetMemoryLength();

Description This routines get memory depth of equipment (KB).

Input: -

Output memory depth of equipment(KB)

Roll Mode: In this mode, the sampling rate is set to a minimum sample rate, and the acquisition length is fixed to a second acquisition data length. Normal call Capture, connect the each data together, is the complete waveform.

int IsSupportRollMode();

Description This routines get the equipment support roll mode or not .

Input: -

Output **Return value** 1 support roll mode

0 not support roll mode

int SetRollMode(unsigned int en);

Description This routines enable or disenable the equipment into roll mode.

Input: -

Output **Return value** 1 success

0 failed

5.5.6. Capture Completion Notice

when capture is complete, the dll have three ways to notify the main program, callback function, set Event and the main program loop detection.

5.5.6.1. callback function

when capture is complete, if the callback function **"datacallback"** is registered, it will be called。he DLL has a function pointer which has to be set to this function, using

void SetDataReadyCallBack(void* ppara, DataReadyCallBack datacallback);

Description This routines sets the callback function of capture complete.

Input: **ppara** the parameter of the callback function

datacallback a pointer to a function with the following prototype:

void **DataReadyCallBack** (void * **ppara**)

Output -

5.5.6.2. Event

when capture is complete, if the Event handle **"dataevent"** is registered, it will be set. The user must reset the event when the used it. he DLL has a function pointer which has to be set to this function, using

void SetDevDataReadyEvent(HANDLE dataevent);

Description This routines set the event handle, these will be set, when capture complete

Input: **dataevent** the event handle

Output -

5.5.6.3. loop detect

int IsDataReady();

Description This routines return the capture is complete or not.

Input: -

Output **Return value** 1 complete

0 not complete

Note: Only need to use one of three ways. Callback and Event functions are asynchronous, more efficient; main program loop detection over a certain time needed to detect whether the capture is complete or not.

5.5.7. Data Read

unsigned int ReadVoltageDatas(char channel, double* buffer,unsigned int length);

Description This routines read the voltage datas. (V)
Input: **channel** **read channel** 0 :channel 1
1 :channel 2
buffer the buffer to store voltage datas
length the buffer length
Output **Return value** the read length

unsigned int ReadVoltageDatasTriggerPoint();
Description This routines read the trigger location where the data collected
Input:
Output **Return value** the trigger points

int IsVoltageDatasOutRange(char channel);
Description This routines return the voltage datas is out range or not.
Input: **channel** **read channel** 0 :channel 1
1 :channel 2
Output **Return value** 0 :not out range
1 :out range

double GetVoltageResolution(char channel);
Description This routines return the current voltage resolution value
One ADC resolution for the voltage value:
Full scale is 1000mv
the ADC is 8 bits
voltage resolution value = 1000mV/256
Input: **channel** **read channel** 0:channel 1
1:channel 2
Output **Return value** voltage resolution value

unsigned int ReadLogicDatas(unsigned char* buffer, unsigned int length);
Description This routines read the logic data of mso.
Input:
buffer the buffer to store logic datas
length the buffer length
Output Return value the read length

unsigned int ReadLogicDatasTriggerPoint();
Description This routines read the trigger location where the data collected
Input:
Output **Return value** the trigger points

5.6. DDS

int IsSupportDDSDevice();
Description This routines get support dds or not

Input: **-**

Output **Return value** support dds or not

int GetDDSDepth();

Description This routines set dds depth

Input:

Output: **Return value** depth

void SetDDSOutMode(unsigned char channel_index, unsigned int out_mode);

Description This routines set dds out mode

Input: **channel_index** 0 :channel 1

1 :channel 2

out_mode DDS_OUT_MODE_CONTINUOUS 0x00

DDS_OUT_MODE_SWEEP 0x01

DDS_OUT_MODE_BURST 0x02

Output

unsigned int GetDDSOutMode(unsigned char channel_index);

Description This routines get dds out mode

Input: channel_index 0 :channel 1

1 :channel 2

Output **mode** DDS_OUT_MODE_CONTINUOUS 0x00

DDS_OUT_MODE_SWEEP 0x01

DDS_OUT_MODE_BURST 0x02

int GetDDSSupportBoxingStyle(int* style);

Description This routines get support wave styles

Input: **style** array to store support wave styles

Output **Return value** if style==NULL return number of support wave styles

else store the styles to array, and return number of wave styles

void SetDDSBoxingStyle(unsigned char channel_index, unsigned int boxing);

Description This routines set wave style

Input: channel_index 0 :channel 1

1 :channel 2

Input: **boxing** **W_SINE = 0x0001,**

W_SQUARE = 0x0002,

W_RAMP = 0x0004,

W_PULSE = 0x0008,

W_NOISE = 0x0010,

W_DC = 0x0020,

W_ARB = 0x0040

Output: -

void UpdateDDSArbBuffer(unsigned char channel_index, unsigned short* arb_buffer, uint32_t arb_buffer_length);

Description This routines update arb buffer

Input: **channel_index** 0 :channel 1

1 :channel 2

arb_buffer the dac buffer

arb_buffer_length the dac buffer length need equal to the dds depth

Output: -

void SetDDSPinlv(unsigned char channel_index, unsigned int pinlv);

Description This routines set frequence

Input: **channel_index** 0 :channel 1

1 :channel 2

Input: **pinlv** frequence

Output: -

void SetDDSDutyCycle(unsigned char channel_index, int cycle);

Description This routines set duty cycle

Input: **channel_index** 0 :channel 1

1 :channel 2

Input: **cycle** duty cycle

Output: -

int GetDDSCurBoxingAmplitudeMv(unsigned int boxing);

Description This routines get dds amplitdude of wave

Input: **boxing** BX_SINE~BX_ARB

Output: Return the amplitdude(mV) of wave

void SetDDSAmplitudeMv(unsigned char channel_index, int amplitdude);

Description This routines set dds amplitdude(mV)

Input: **channel_index** 0 :channel 1

1 :channel 2

amplitdude amplitdude(mV)

Output: -

int GetDDSAmplitudeMv(unsigned char channel_index);

Description This routines get dds amplitdude(mV)

Input: **channel_index** 0 :channel 1

1 :channel 2

Output: return amplitdude(mV)

int GetDDSCurBoxingBiasMvMin(unsigned int boxing);

int GetDDSCurBoxingBiasMvMax(unsigned int boxing);

Description This routines get dds bias of wave
Input: **boxing** BX_SINE~BX_ARB
Output: Return the bias(mV) range of wave

void SetDDSBiasMv(unsigned char channel_index, int bias);
Description This routines set dds bias(mV)
Input: **channel_index** 0 :channel 1
1 :channel 2
bias bias(mV)
Output: -

int GetDDSBiasMv(unsigned char channel_index);
Description This routines get dds bias(mV)
Input: **channel_index** 0 :channel 1
1 :channel 2
Output: Return the bias(mV) of wave

void SetDDSSweepStartFreq(unsigned char channel_index, double freq);
Description This routines set dds sweep start freq
Input: **channel_index** 0 :channel 1
1 :channel 2
freq
Output: -

double GetDDSSweepStartFreq(unsigned char channel_index);
Description This routines get dds sweep start freq
Input: **channel_index** 0 :channel 1
1 :channel 2
Output: **freq**

void SetDDSSweepStopFreq(unsigned char channel_index, double freq);
Description This routines set dds sweep stop freq
Input: **channel_index** 0 :channel 1
1 :channel 2
freq
Output: -

double GetDDSSweepStopFreq(unsigned char channel_index);
Description This routines get dds sweep stop freq
Input: **channel_index** 0 :channel 1
1 :channel 2
Output: **freq**

void SetDDSSweepTime(unsigned char channel_index, unsigned long long int time_ns);

Description This routines set dds sweep time

Input: **channel_index** 0 :channel 1

1 :channel 2

time/ns

Output: -

unsigned long long int GetDDSSweepTime(unsigned char channel_index);

Description This routines get dds sweep time

Input: **channel_index** 0 :channel 1

1 :channel 2

Output: **time/ns**

void SetDDSBurstStyle(unsigned char channel_index, int style);

Description This routines set dds burst style

Input: **channel_index** 0 :channel 1

1 :channel 2

style 0--n loops

1--gate

Output: -

int GetDDSBurstStyle(unsigned char channel_index);

Description This routines get dds burst style

Input: **s** 0 :channel 1

1 :channel 2

Output: **style** 0--n loops

1--gate

void SetDDSLoopsNum(unsigned char channel_index, unsigned long long int num);

Description This routines set dds loops num

Input: **channel_index** 0 :channel 1

1 :channel 2

num

Output: -

unsigned long long int GetDDSLoopsNum(unsigned char channel_index);

Description This routines get dds loops num

Input: **channel_index** 0 :channel 1

1 :channel 2

Output: **num**

void SetDDSLoopsNumInfinity(unsigned char channel_index, int en);

Description This routines set dds loops num infinity

Input: **channel_index** 0 :channel 1
1 :channel 2
en
Output: -

int GetDDSLoopsNumInfinity(unsigned char channel_index);
Description This routines get dds loops num infinity
Input: **channel_index** 0 :channel 1
1 :channel 2
Output: **loops num infinity**

void SetDDSBurstPeriodNs(unsigned char channel_index, unsigned long long int ns);
Description This routines set dds burst period(ns)
Input: **channel_index** 0 :channel 1
1 :channel 2
ns
Output: -

unsigned long long int GetDDSBurstPeriodNs(unsigned char channel_index);
Description This routines get dds burst period(ns)
Input: **channel_index** 0 :channel 1
1 :channel 2
Output: **ns**

void SetDDSBurstDelayNs(unsigned char channel_index, unsigned long long int ns);
Description This routines set dds burst delay time(ns)
Input: **channel_index** 0 :channel 1
1 :channel 2
ns
Output: -

unsigned long long int GetDDSBurstDelayNs(unsigned char channel_index);
Description This routines get dds burst delay time(ns)
Input: **channel_index** 0 :channel 1
1 :channel 2
Output: **ns**

void SetDDSTriggerSource(unsigned char channel_index, unsigned int src);
Description This routines set dds trigger source
Input: **channel_index** 0 : channel 1
1 : channel 2
src 0 : internal

1 : external
2 : manual

Output: -

unsigned int GetDDSTriggerSource(unsigned char channel_index);

Description This routines get dds trigger source

Input: **channel_index** 0 : channel 1
1 : channel 2

Output: **trigger source** 0 : internal
1 : external
2 : manual

void SetDDSTriggerSourceIo(unsigned char channel_index, uint32_t io);

Description This routines set dds trigger source io

Input: **channel_index** 0 : channel 1
1 : channel 2
io 0 : DIO0
.....
7 : DIO7

Output: -

Note: You need to use the DIO API to set the corresponding DIO to the input/output state

uint32_t GetDDSTriggerSourceIo(unsigned char channel_index);

Description This routines get dds trigger source io

Input: **channel_index** 0 : channel 1
1 : channel 2

Output: **trigger source io** 0 : DIO0
.....
7 : DIO7

void SetDDSTriggerSourceEnge(unsigned char channel_index, unsigned int enge);

Description This routines set dds trigger source enge

Input: **channel_index** 0 : channel 1
1 : channel 2
enge 0 : rising
1 : falling

Output: -

unsigned int GetDDSTriggerSourceEnge(unsigned char channel_index);

Description This routines get dds trigger enge

Input: **channel_index** 0 : channel 1
1 : channel 2

Output: **enge** 0 : rising
1 : falling

void SetDDSOutputGateEnge(unsigned char channel_index, unsigned int enge);

Description This routines set dds output gate enge

Input: **channel_index** 0 : channel 1
1 : channel 2

enge 0 : close
1 : rising
2 : falling

Output: -

unsigned int GetDDSOutputGateEnge(unsigned char channel_index);

Description This routines get dds output gate enge

Input: **channel_index** 0 : channel 1
1 : channel 2

Output: **enge** 0 : close
1 : rising
2 : falling

void DDSManualTrigger(unsigned char channel_index);

Description This routines manual trigger dds

Input: **channel_index** 0 : channel 1
1 : channel 2

Output: -

void DDSOutputEnable(unsigned char channel_index , int enable);

Description This routines enable dds output or not

Input: **channel_index** 0 : channel 1
1 : channel 2

enable 1 enable
0 not enable

Output: -

int IsDDSOutputEnable(unsigned char channel_index);

Description This routines get dds output enable or not

Input: **channel_index** 0 : channel 1
1 : channel 2

Output **Return value** dds enable or not

5.7. IO

int IsSupportIODevice();

Description This routines get support IO ctrl or not

Input: -

Output Return value support io ctrl or not

int GetSupportIoNumber();

Description This routines get support io nums of equipment.

Output Return value the sample number of io nums

When IO is set as input, there are three ways to read the IO status, drop the function, trigger the Event, and the main program loop detection.

Callback functions

The SDK will read the IO status periodically, and if the main program registers the fallback function "datacallback", it will be called. DLLs have a function dedicated to setting this fallback function.

void SetIOReadStateCallBack(void* ppara, IOReadStateCallBack callback);

Description This routines sets the callback function of read io status.

Input: **ppara** the parameter of the callback function

callback a pointer to a function with the following prototype

Event

The SDK will read the IO status regularly, and if the main program registers the Event handle "dataevent", it will be set. It should be noted that after the main program detects the Event, it needs to reset the Event. The DLL has a function specifically for setting this Event handle

void SetIOReadStateReadyEvent(HANDLE dataevent);

Description This routines set the event handle, these will be set, when capture complete

Input: **dataevent** the event handle

Output -

Loop Detect

int IsIOReadStateReady();

Description This routines return read io is complete or not.

Input: -

Output **Return value** 1 complete

0 not complete

Note: Only need to use one of three ways. Callback and Event functions are asynchronous, more efficient; main program loop detection over a certain time needed to detect whether the capture is complete or not.

void IOEnable(unsigned char channel, unsigned char enable);

Description This routines set io enable or not

Input: **channel** dio0 0

dio1 1

dio2 2

.....

enable not enable 0

enable 1

Output: -

unsigned char IsIOEnable(unsigned char channel);

Description This routines get io enable or not

Input: **channel** dio0 0
dio1 1
dio2 2
.....

Output: **return** not enable 0 or enable 1

void SetIOInOut(unsigned char channel, unsigned char inout);

Description This routines set io in or out

Input: **channel** dio0 0
dio1 1
dio2 2
.....

inout in 0
out 1

Output: -

unsigned char GetIOInOut(unsigned char channel);

Description This routines get io in or out

Input: **channel** dio0 0
dio1 1
dio2 2
.....

Output: **return** in 0
out 1

void SetIOOutState(unsigned char channel, unsigned char state);

Description This routines set io state

Input: **channel** dio0 0
dio1 1
dio2 2
.....

State 0--0
1--1
2--z
3--pulse
4--dds gate

Output: -

unsigned int GetIOInState();

Description This routines get io state

If the SetIOReadStateCallBack setting callback function is used, IOReadStateCallBack will directly notify the IO input status; If use SetIOReadStateReadyEvent and IsIOReadStateReady to read the query, you need to call GetIOState to get the IO input status

Output: return io state