Mult MSO C SDK API User Guide

Version 1.20

Vimu Electronic Technology

2025-08-12

http://www.vimu.top/

Update Log

V1.20 (2025.8.1)

Initial version (following the MSO C SDK version)

Catalog

1. introduction

The Mult MOS C SDK is a standard dynamic library that adds multi-device support APIs on top of the MSO C SDK.
The interface supports widows systems (X86, X64, and ARM64) and linux systems (X64, ARM32, and ARM64).

2. Differences from MSO C SDK

The functional APIs for device control, oscilloscope, DDS and IO are named and used in the same way as the MSO C SDK, the only difference is that the device ID parameter is ‘unsigned int dev_id’.
Compared with the MSO C SDK, the Mult MOS C SDK adds the Stream Mode function, which mainly describes the API and usage of the Stream Mode.

3. Streaming mode is used

Streaming modes require faster data processing to prevent data from accumulating into memory, and all streaming modes only support callback function mode.

3.1. main function

```
 int main()
{
      //init dll
      InitDll(1, 1);
      //Set up device and data callback functions
      SetDevNoticeCallBack(NULL, mDevNoticeAddCallBack,
mDevNoticeRemoveCallBack);
      SetStreamDataReadyCallBack(NULL, mStreamDataReadyCallBack);
     //Scan for devices that have been plugged in
      ScanDevice();
     //Main cycle
      while (true)
      {
            std::this_thread::sleep_for(std::chrono::milliseconds(100));
             //Exit when the collection is complete
            if(capture_ok)
                  break;
      };
      //Free up resources
      FinishDll();
      return 0;
}
```

3.2. Device plug in callback function

```
void CALLBACK mDevNoticeAddCallBack(void* ppara, unsigned int dev_id)
{
    //DDS init
    DDSInit(dev_id, 0, DDS_OUT_MODE_CONTINUOUS);

    //IO init
    IOInit(dev_id);

    //Select the working mode
    SetOscCaptureMode(dev_id, 1);
    //set input range
    SetStreamChannelRangemV(dev_id, 0, -10000, 10000);
    SetStreamChannelRangemV(dev_id, 1, -10000, 10000);

    //sample rate
    int sample_num = GetStreamSupportSampleRateNum(dev_id);
    if (sample != NULL)
    {
        delete[]sample;
        sample = NULL;
    }
    sample = new unsigned int[sample_num];
    //read support sample rate
    if (GetStreamSupportSampleRates(dev_id, sample, sample_num))
    {
        for (int i = 0; i < sample_num; i++)
            std::cout << std::dec << sample[i] << '\n';
        std::cout << std::endl;
    }
    // Record the device and initialize the acquisition status
    m_dev_id = dev_id;
    capture_ok = false;
    capture_ok_mask = 0;

    // Using a 1M sampling rate, 10M data is collected
    uint64_t length = 1024*1024*10;    //10M
    StreamCapture(m_dev_id, length/1024, capture_channel_mask, 1000000);
}
```

3.3. Data callback function

```
void CALLBACK mStreamDataReadyCallBack(void* ppara, unsigned int dev_id, unsigned char channel_index, double* buffer, unsigned int buffer_length, unsigned int failed, unsigned int success, unsigned long long int need_total_sample, unsigned long long int total_sample, unsigned long long int menoryuse)
{
    //Note：The callback function should not handle complex tasks, and if it takes too long,
    //it will cause the watchdog to reset and the USB to reconnect

    /*illustrate
    buffer The buffer will be destroyed after the callback is completed, so you need to copy
the data of the buffer yourself
    buffer_length The length of the current buffer
    failed Whether the acquisition failed
    success Whether the collection is complete
    need_total_sample A total of data needs to be collected
    total_sample Data that has been collected
    menoryuse How many buffers are currently used by the dynamic library (the more it is,
the more data it means the dynamic library has piled up)*/

    //Process data in the buffer according to your needs.......

    //The last channel callback is complete and ready to exit
    if(success)
    {
        capture_ok_mask |= (0x01<<channel_index);
        if(capture_ok_mask==capture_channel_mask)
            capture_ok = true; //flag complete, and the main function exits
        }
    }
}
```

4. Stream Mode API Description

4.1. Working mode

int SetOscCaptureMode(unsigned int dev_id, int is_stream_mode);

Description This routines set the osc capture mode.

Input: dev_id the dev id

is_stream_mode enable the stream mode or not

Output Return value 1 Success

0 Failed

4.2. Collection range setting

The preamplifier of the device is equipped with a programmable gain amplifier, when the collected signal is smaller than the AD range, the gain amplifier can amplify the signal, and make more use of the number of AD bits to improve the quality of the acquired signal. The DLL will automatically adjust the gain amplifier of the preamp according to the set acquisition range.

int GetStreamRangeMinmV(unsigned int dev_id);

int GetStreamRangeMaxmV(unsigned int dev_id);

Description This routines set the range of input signal.

Output Return value

minmv the minimum voltage of the input signal (mV) maxmv the maximum voltage of the input signal (mV)

int SetStreamChannelRangemV(unsigned int dev_id, 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: When the maximum acquisition range is probe X1, the maximum voltage that the oscilloscope can collect. For example, MSO20 is [-12000mV, 12000mV].

Note: In order to achieve a better waveform effect, you must set the acquisition range according to the amplitude of the waveform being measured. If necessary, the collection range can be dynamically changed.

4.3. Sampling rate

int GetStreamSupportSampleRateNum(unsigned int dev_id);

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

Input: -

Output Return value the sample number

int GetStreamSupportSampleRates(unsigned int dev_id, 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

4.4. Capture

int StreamCapture(unsigned int dev_id, unsigned long long int length_kb, unsigned short capture_channel, unsigned int sample_rate);

Description This routines set the capture length and start capture.

Input: length capture length(KB)

capture_channel: //ch1=0x0001 ch2=0x0020 ch3=0x0040 ch4=0x0080 logic=0x0100

ch1+ch2 0x03

ch1+ch2+ch3 0x07

ch1+ch2+ch3+ch4 0x0F

Output Return 1 success

-1 sample_rate error

-2 device id error

void StreamStopCapture(unsigned int dev_id);

Description This routines stop the capture

Input:

Output Return

4.5. Capture completion notification

When data acquisition is complete, the rollback function will call back the notification and provide the corresponding acquisition buffer and acquisition status.

Void SetStreamDataReadyCallBack(void* ppara, StreamDataReadyCallBack 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 **StreamDataReadyCallBack** prototype:

Output -

void CALLBACK StreamDataReadyCallBack(void* ppara, unsigned int dev_id, unsigned char channel_index, double* buffer, unsigned int buffer_length, unsigned int failed, unsigned int success, unsigned long long int need_total_sample, unsigned long long int total_sample, unsigned long long int menoryuse);

illustrate

buffer The buffer will be destroyed after the callback is completed, so you need to copy the data of the buffer yourself

buffer_length The length of the current buffer

failed Whether the acquisition failed

success Whether the collection is complete

need_total_sample A total of data needs to be collected

total_sample Data that has been collected

menoryuse How many buffers are currently used by the dynamic library (the more it is, the more data it means the dynamic library has piled up)