SDK API User Guide

Version 1.0

Vimu Electronic Technology

2023-3-31

http://www.vimu.top/

Update Log

V1.0 (2023.3.31)

1, New design

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. 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);

Description Dll initialization

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

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

3. 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)**

4. Equipment Reset

int ResetDevice(void);

Description This routines reset device

Input: -

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

5.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.

5.1. 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 -

5.2. 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 -

5.3. 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.

6.Oscillograph

6.1. 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.

6.2. 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

6.3. 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** 0 :channel 1
1 :channel 2

void SetTriggerSource(unsigned int source);

Description This routines set the trigger source.

Input: **source** 0 :channel 1
1 :channel 2

Output -

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: -

6.4. AC/DC

int IsSupportAcDc();

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

Input: -

Output **Return value** 0 :support AC/DC switch

1 :not 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

6.5. 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=0x0020 ch3=0x0040 ch4=0x0080 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

6.6. 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.

6.6.1. 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 -

6.6.2. 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 -

6.6.3. 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.

6.7. 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

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

7. DDS

int IsSupportDDSDevice();

Description This routines get support dds or not

Input: -

Output **Return value** support dds or not

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 int boxing);

Description This routines set wave style

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 SetDDSPinlv(unsigned int pinlv);

Description This routines set frequence

Input: **pinlv** frequence

Output: -

void SetDDSDutyCycle(int cycle);

Description This routines set duty cycle

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 WINAPI 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 DDSOutputEnable(int enable);

Description This routines enable dds output or not

Input: **enable** 1 enable
0 not enable
Output: -

int IsDDSOutputEnable();
Description This routines get dds output enable or not
Input: -
Output **Return value** dds enable or not

8. 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** channel number

enable not enable 0

enable 1

Output: -

unsigned char IsIOEnable(unsigned char channel);

Description This routines get io enable or not

Input: **channel** channel number

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** channel number

inout in 0

out 1

Output: -

unsigned char GetIOInOut(unsigned char channel);

Description This routines get io in or out

Input: **channel** channel number

Output: **return** in 0

out 1

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

Description This routines set io state

Input: **channel** channel number

state 0 or 1

Output: -

char GetIOInState(unsigned char channel);

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

Input: channel channel number

Output: return 0 state

1 state