Andriod SDK User Guide

Version 1.3

Vimu Electronic Technology

2025-3-26

http://www.vimu.top/

Update Log

V1.0 (2023.9.20)

Init Versin

V1.1 (2023.11.22)

Add DDS

Add MSO10 Support

V1.4 (2024.7.18)

Add MSO41 Support

V1.3 (2025.3.26)

Add ReScan USB device function

Catalog

1. Introduction

The MSO Mixed Signal Oscilloscope is equipped with an Android aar interface, through which the mixed signal oscilloscope can be directly controlled.

This interface can be used on Android systems that support USB Host.

2. Permission Request

2.1. USB Permission

Add the following information to the file AndroidManifest.xml.

```
<uses-feature
        android:name="android.hardware.usb.host"
        android:required="true" />

    <uses-permission android:name="android.hardware.usb.host"/>
    <uses-permission android:name="android.permission.HARDWARE_TEST"/>
    <uses-permission android:name="android.permission.SYSTEM_ALERT_WINDOW"/>

<intent-filter>
    <action android:name="android.hardware.usb.action.USB_DEVICE_ATTACHED" />
</intent-filter>

<meta-data
    android:name="android.hardware.usb.action.USB_DEVICE_ATTACHED"
    android:resource="@xml/device_filter" />
```

Copy the device_filter.xml to the res/xml directory.

2.2. Large Heap Permissions

Because the capture card supports a maximum storage depth of 32MB, in order to make the app apply for more memory, the following content is added.

android:largeHeap="true"

3. UsbDevMng

UsbDevMng is used to manage device insertion and unplugging detection, and is notified through the UsbDevMng.UsbDevDetectLister interface.

3.1. Creation and initialization

```
usbManger = new UsbDevMng(UsbDevDetectLister UsbDevDetectLister);
usbManger.intiDetect(Context context);
```

3.2. Device status change notification processing

```
public void UsbDevDetectCallback(UsbDevMng.DEVICE_DETECT_STATE state, boolean success, BasicUsbDev dev) {
    if (state == UsbDevMng.DEVICE_DETECT_STATE.DEVICE_ADD) {
        //Device Add
    }
    else if (state == UsbDevMng.DEVICE_DETECT_STATE.DEVICE_REMOVE) {
```

```
            // Device Remove
        }
        else if (state == UsbDevMng.DEVICE_DETECT_STATE.NEED_PERMISSION) {
            //no Permissions
        }
            else if (state == UsbDevMng.DEVICE_DETECT_STATE.NEED_RSCAN) {
        }
    }
```

3.3. Re-scan the device that has been plugged in

In some systems, after authorization, the broadcast information acquisition will fail, and the device needs to be manually rescanned and connected.

boolean scanDevice(Context context, boolean requestDialog);

Description: Re-scan the USB device plugged into the system

Input: **context**

requestDialog If there is no permission, whether to reapply

Output: **Return value** success or failed

4. OscDdsFactory

OscDdsFactory is used to create controls for oscilloscopes, DDS, or other corresponding functions based on BasicUsbDev devices

CreateSbqCardWave creates the control class for the oscilloscope

BasicSbqUsbCardVer12 CreateSbqCardWave(BasicSbqUsbCardVer12.WaveReceiveLister callback, BasicUsbDev dev)

Description: Create an oscilloscope's control class.

Input: **BasicSbqUsbCardVer12.WaveReceiveLister** Waveform update notification

BasicUsbDev MSO USB device class

Output: **Return value** oscilloscope's control class

CreateDDSWave creates a control class for DDS signal sources

BasicHsfUsbWaveV12 CreateDDSWave(BasicUsbDev dev)

Description: Create an dds control class.

Input: **BasicUsbDev** MSO USB device class

Output: **Return value** DDS control class

5.Oscillograph

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 SetRange(int channel, double minv, double maxv);

Description: Set the range of input signal.

Input: **channel** the set channel

0 channel 1

1 channel 2

minv the minimum voltage of the input signal (V)

maxv the maximum voltage of the input signal (V)

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.

5.2. Sample

int GetSamplesNum ();

Description: Get the number of samples that the equipment support.

Input: -

Output **Return value** the support sample number

int GetSamples(int[] sample, int maxnum);

Description: 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 SetSample(int sample);

Description: Set the sample.

Input: **sample** the set sample

Output **Return value** 0 Failed

other value new sample

int GetSample();

Description Get the sample.

Input: -

Output **Return value** sample

5.3. Trigger(hardware trigger)

Trigger Mode

```
enum TRIGGER_MODE {
        AUTO(0),
        LIANXU(1)
};
```

Trigger Style

```
enum TRIGGER_STYLE {
        NONE(0),          //not trigger
        RISE_EDGE(1),     //Rising edge
        FALL_EDGE(2), //Falling edge
```

```
        EDGE(4), //Edge
        PULSE_P_MORE(8), //Positive Pulse width(>)
        PULSE_P_LESS(16), //Positive Pulse width(<)
        PULSE_P(32), //Positive Pulse width(<>)
        PULSE_N_MORE(64), //Negative Pulse width(>)
        PULSE_N_LESS(128), //Negative Pulse width(<)
        PULSE_N(256); //Negative Pulse width(<>)
};
```

TRIGGER_MODE GetTriggerMode();

Description: Get the trigger mode.

Input: -

Output **Return value** TRIGGER_MODE

void SetTriggerMode(TRIGGER_MODE mode);

Description: Set the trigger mode.

Input: **mode** TRIGGER_MODE

Output -

TRIGGER_STYLE GetTriggerStyle();

Description: Get the trigger style.

Input: -

Output **Return value** TRIGGER_STYLE

void SetTriggerStyle(TRIGGER_STYLE style);

Description: Set the trigger style.

Input: **style** TRIGGER_STYLE

Output -

int GetTriggerPulseWidthNsMin();

Description: Get the min time of pulse width.

Input: -

Output Return min time value of pulse width(ns)

int GetTriggerPulseWidthNsMax();

Description: Get the max time of pulse width.

Input: -

Output Return max time value of pulse width(ns)

int GetTriggerPulseWidthDownNs();

Description: Get the down time of pulse width.

Input: -

Output Return down time value of pulse width(ns)

int GetTriggerPulseWidthUpNs();

Description: 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: Set the up time of pulse width.

Input: **down_ns**

up_ns up time value of pulse width(ns)

Output _

TRIGGER_SOURCE GetTriggerSource();

Description: Get the trigger source.

Input: **-**

Output **Return value**

TRIGGER_SOURCE.CH1	0x0000000000000001L	//CH1
TRIGGER_SOURCE.CH2	0x0000000000000002L	//CH2
TRIGGER_SOURCE.D0	0x0000000000010000L	//Logic 0
TRIGGER_SOURCE.D1	0x0000000000020000L	//Logic 1
TRIGGER_SOURCE.D2	0x0000000000040000L	//Logic 2
TRIGGER_SOURCE.D3	0x0000000000080000L	//Logic 3
TRIGGER_SOURCE.D4	0x0000000000100000L	//Logic 4
TRIGGER_SOURCE.D5	0x0000000000200000L	//Logic 5
TRIGGER_SOURCE.D6	0x0000000000400000L	//Logic 6
TRIGGER_SOURCE.D7	0x0000000000800000L	//Logic 7

void SetTriggerSource(TRIGGER_SOURCE source);

Description: Set the trigger source.

Input: **source**

TRIGGER_SOURCE.CH1	0x0000000000000001L	//CH1
TRIGGER_SOURCE.CH2	0x0000000000000002L	//CH2
TRIGGER_SOURCE.D0	0x0000000000010000L	//Logic 0
TRIGGER_SOURCE.D1	0x0000000000020000L	//Logic 1
TRIGGER_SOURCE.D2	0x0000000000040000L	//Logic 2
TRIGGER_SOURCE.D3	0x0000000000080000L	//Logic 3
TRIGGER_SOURCE.D4	0x0000000000100000L	//Logic 4
TRIGGER_SOURCE.D5	0x0000000000200000L	//Logic 5
TRIGGER_SOURCE.D6	0x0000000000400000L	//Logic 6
TRIGGER_SOURCE.D7	0x0000000000800000L	//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: Get the trigger level.
Input: -
Output **Return value** level (V)

void SetTriggerLevel(int level);
Description: Set the trigger level.
Input: level (V)
Output -

int GetTriggerSenseDiv();
Description: Get the trigger sense.
Input: -
Output **Return value** Sense (0-1 div)

void SetTriggerSenseDiv(int sense, double y_interval_v);
Description: Set the trigger sense.
Input: Sense (0-1 div)
Interval(V)
Output -

Note: The sensitivity of sense triggers ranges from 0.1 div to 1.0 div.

y_interval_v The oscilloscope software uses a vertical sensitivity setting, which is the voltage value for each slot.

The SDK can be set by dividing the acquisition range by 10, that is, (m_osc_range_maxv - m_osc_range_minv)/10.0, and the last sensitivity voltage is (m_osc_range_maxv - m_osc_range_minv)/10.0*sense.

int GetTriggerFrontPercent ();
Description: Get the Pre-trigger Percent.
Input: -
Output Return value Percent (5-95)

void SetPreTriggerPercent(int front);
Description: Set the Pre-trigger Percent.
Input: Percent (5-95)
Output -

int IsSupportTriggerForce();
Description: Get the equipment support trigger force or not.
Input: -
Return value 1 support
0 not support

void TriggerForce();
Description: Force capture once.

Input: -

Output: -

5.4. AC/DC

int IsSupportAcDc(int channel);

Description: 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(int channel, int ac);

Description: 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(int channel);

Description: Get the device AC coupling.

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

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

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, short capture_channel, **byte force_length);**

Description: 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: Set the abort capture

Input:

Output Return value 1:success 0:failed

int GetHardMemoryDepth();

Description: Get memory depth of equipment (KB).

Input: -

Output memory depth of equipment(KB)

5.6.Capture Completion Notice

When the data acquisition is complete, the main program is notified via the BasicSbqUsbCardVer12.WaveReceiveLister drop.

```
public boolean WaveReceiveCallBack(boolean success, int length){
    if(success) {
            //Update UI
            runOnUiThread(new Runnable() {
                public void run() {
                    WaveReceive(length); //UI datas process
                }
            });
     }
     return true;
}
```

Note: The notification callback function cannot access the Android UI data, so you need to use runOnUiThread to run the corresponding data processing function of the UI.

5.7.Data Read

int ReadVoltageDatas(byte channel, double[] buffer, int length);

Description: 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(byte channel);

Description: 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(byte channel);

Description: 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

int ReadLogicDatas(byte[] buffer, int length);

Description: Read the logic data of mso.

Input:

buffer the buffer to store logic datas

length the buffer length

Output Return value the read length

6. DDS

int GetDepth();

Description: Get DDS depth

Input:

Output: **Return value** depth

void SetOutMode(int channel_index, DDS_OUT_MODE out_mode);

Description: 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

DDS_OUT_MODE GetOutMode(int channel_index);

Description: 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

void SetBoxing(int channel_index, BOXING_STYLE boxing);

Description: Set wave style

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

boxing **W_SINE = 0x0001,**
W_SQUARE = 0x0002,
W_RAMP = 0x0004,
W_PULSE = 0x0008,
W_NOISE = 0x0010,
W_DC = 0x0020,
W_ARB = 0x0040

Output: -

void UpdateArbBuffer(int channel_index, short[] arb_buffer, int arb_buffer_length);

Description: 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 SetFreq (int channel_index, int freq);

Description: Set frequence

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

freq frequence

Output: -

void SetDutyCycle(int channel_index, int cycle);

Description: Set duty cycle

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

cycle duty cycle

Output: -

int GetCurBoxingAmplitudeMv(BOXING_STYLE boxing);

Description: Get DDS amplitdude of wave

Input: **boxing** BX_SINE~BX_ARB

Output: Return the amplitdude(mV) of wave

void SetAmplitudeMv(int channel_index, int amplitdude);

Description: Set DDS amplitdude(mV)

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

amplitdude amplitdude(mV)

Output: -

int GetAmplitudeMv(int channel_index);

Description: Get DDS amplitdude(mV)

Input: **channel_index** 0 :channel 1

1 :channel 2

Output: return amplitdude(mV)

int GetCurBoxingBiasMvMin(BOXING_STYLE boxing);

int GetCurBoxingBiasMvMax(BOXING_STYLE boxing);

Description: Get DDS bias of wave

Input: **boxing** BX_SINE~BX_ARB

Output: Return the bias(mV) range of wave

void SetBiasMv(int channel_index, int bias);

Description: Set DDS bias(mV)

Input: **channel_index** 0 :channel 1

1 :channel 2

bias bias(mV)

Output: -

int GetBiasMv(int channel_index);

Description: Get DDS bias(mV)

Input: **channel_index** 0 :channel 1

1 :channel 2

Output: Return the bias(mV) of wave

void SetSweepStartFreq(int channel_index, double freq);

Description: Set DDS sweep start freq

Input: **channel_index** 0 :channel 1

1 :channel 2

freq

Output: -

double GetSweepStartFreq(int channel_index);

Description: Get DDS sweep start freq

Input: **channel_index** 0 :channel 1

1 :channel 2

Output: **freq**

void SetSweepStopFreq(int channel_index, double freq);

Description: Set DDS sweep stop freq

Input: **channel_index** 0 :channel 1

1 :channel 2

freq

Output: -

double GetSweepStopFreq(int channel_index);

Description: Get dds sweep stop freq

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

Output: **freq**

void SetSweepTime(int channel_index, long time_ns);

Description: Set DDS sweep time

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

time/ns

Output: -

long GetSweepTime(int channel_index);

Description: Get DDS sweep time

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

Output: **time/ns**

void SetTriggerSource(int channel_index, DDS_TRIGGER_SOURCE src);

Description: Set DDS trigger source

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

src 0: internal 2
0: INTERNAL
1: EXTERNAL
2: MANUAL

Output: –

int GetTriggerSource(int 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 SetTriggerSourceIo(int channel_index, int io);

Description: 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

int GetTriggerSourceIo(int channel_index);

Description: Get DDS trigger source io

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

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

void SetTriggerSourceEnge(int channel_index, DDS_ENGE enge);

Description: Set DDS trigger source enge

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

Output: -

int GetTriggerSourceEnge(int channel_index);

Description: Get DDS trigger enge

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

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

void SetOutputGateEnge(int channel_index, DDS_OUTPUT_ENGE enge);

Description: Set DDS output gate enge

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

Output: -

int GetOutputGateEnge(int channel_index);

Description: Get DDS output gate enge

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

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

void ManualTrigger(int channel_index);

Description: Manual trigger DDS

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

Output: -

void ChannelStart (int channel_index);

Description: Enable DDS output or not

Input: c**hannel_index** 0 : channel 1
1 : channel 2

Output: -

boolean ChannelIsStart (int channel_index);

Description: Get DDS output enable or not

Input: **-**

Output **Return value** DDS enable or not