SDK User Guide

Version 2.8

InstruStar Electronic Technology

2017-2-10

http://www.instrustar.com/

Update

V2.1 (2014.5.13)

1, New design

V2.2 (2014.7.10)

1, Modified instructions of software trigger description

V2.3 (2014.11.28)

1, Modified "ReadVoltageDatas" Time is too long bug

V2.4 (2014.12.5)

1, Repair the console class program, unable to detect USB plug problem

V2.5 (2015.7.27)

1, Support ISDS2062

V2.6 (2015.8.15)

1, Support Roll Mode(Need hardware support)

V2.7 (2016.5.5)

1, Support Trigger Sense Div setting (Need hardware support)

2, Support Force (Need hardware support)

3, Support the setting time of Pulse Width(Need hardware support)

4, Support the setting Pre-trigger Percent (Need hardware support)

V2.8 (2017.2.10)

1, Support Read voltage datas is out range or not

2, Fix 210 Pre-trigger Percent bug

3,Add Equipment Id API

contents

1. Introduction

SDK as a virtual oscilloscope equipped with a Windows standard DLL interface, through this interface can directly control a virtual oscilloscope and obtain the data acquired by the oscilloscope. The SDK supports MDSO, MDSO-LA, HDSO, DDSO, ISDS205, ISDS210, ISDS220 and ISDS2062 equipment.

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(void);

Description Dll initialization

Input: -

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 Info

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)**

int ResetDevice(void);

Description This routines reset device

Input: -

Output: **- Return value** 1 success

0 failed

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.

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	-	

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	-	

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. 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 ISDS220 is[-16000mV,16000mV].

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.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

7.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: -

8.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

9.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.

int Capture(int length);

Description This routines set the capture length and start capture.

Input: **length** capture length(KB)

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

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

10.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.

10. 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 -

10. 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 -

10. 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.

11.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

12.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** BX_SINE 0x00 //Sine

BX_SQUARE 0x01 //Square

BX_TRIANGULAR 0x02 //Triangular

BX_UP_SAWTOOTH 0x03 //Up Sawtooth

BX_DOWN_SAWTOOTH 0x04 //Down Sawtooth

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:	-

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