Acute Digital Storage Oscilloscope Software development kit (SDK) Programming guide

Version: 1.6
Publish: 2013/08/01

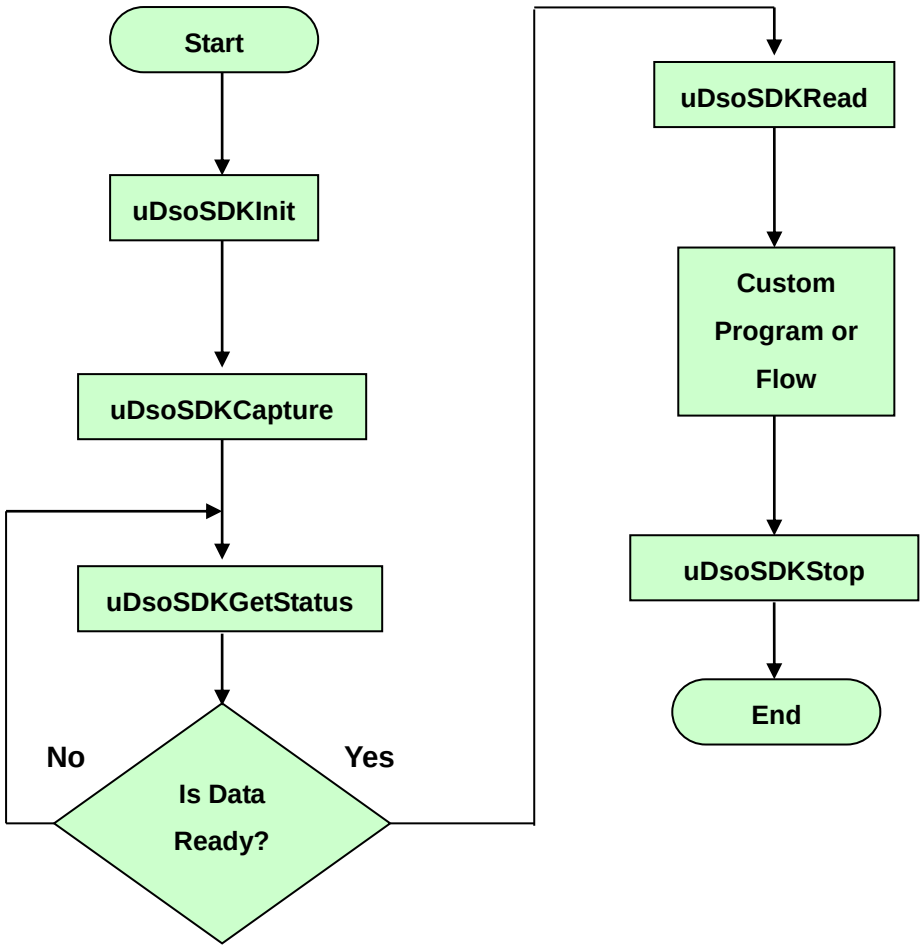
Table of Content

DSOSDK Architechture

DSOSDK.DLL provides Acute DSO control APIs based on Win32 architechture. When using these SDK functions, the DSO device must be connected to the PC. The DSOSDK.dll is inherited from the DSORun.dll, and provided more functions such as “Measurement” and “Function Generator control” to help the user to develop their test machine.

For the DS-1000 series user, you need to use the DSO.exe to calibrate the DSO before using the DSOSDK.dll.

DSOSDK.DLL Flow Chart



DSOSDK.DLL Functions

DSO initializing fucntions

int uDsoSDKInit()

Initial the DSO devices by the order of USB port number. Stack cable is required in stack mode.

Return value

Returning the numbers of DSO was found, returned zero when no DSO was found.

int uDsoSDKInitStack(LPSTR lpszSN)

Initial the DSO devices by the order of assigned serial number. Stack cable is required in stack mode.

Parameter

lpszSN[in]:

Type:LPSTR

Input the null-terminate string of DSO serial number for stack sequence, each serial number must be seperated by commons.

Return value

Returning the numbers of DSO was found, returned zero when no matched DSO was found.

Example

Input "TSA22120001,TSA22120002,TSA22120003"

TSA22120001 – CH1, CH2

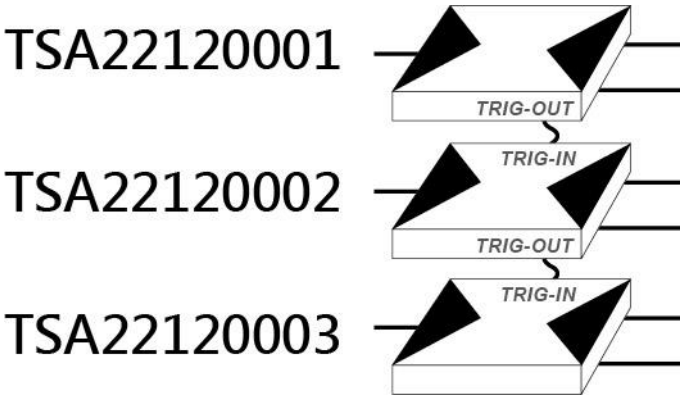
TSA22120002 – CH3, CH4

TSA22120003 – CH5, CH6

First stack cable linked from TSA22120001's Trig-Out to TSA22120002's Trig-In

Second stack cable linked from TSA22120002's Trig-Out to TSA22120003's Trig-In

(See the picture below)



BOOL uDsoSDKSelectGroup(int iGroup)

Users can control multiple DSO groups by selecting the group and pass the parameter to the group. Each group can be considered as a multiple-channel DSO, the DSOs in this group will be set to stack mode when there were more than one DSO in the group. This function should be ignored if there is only one group needed.

Parameter

iGroup[in]:

Type: int

Select target group, maximum input is 19, minimum input is 0. Group 0 will be select as default group.

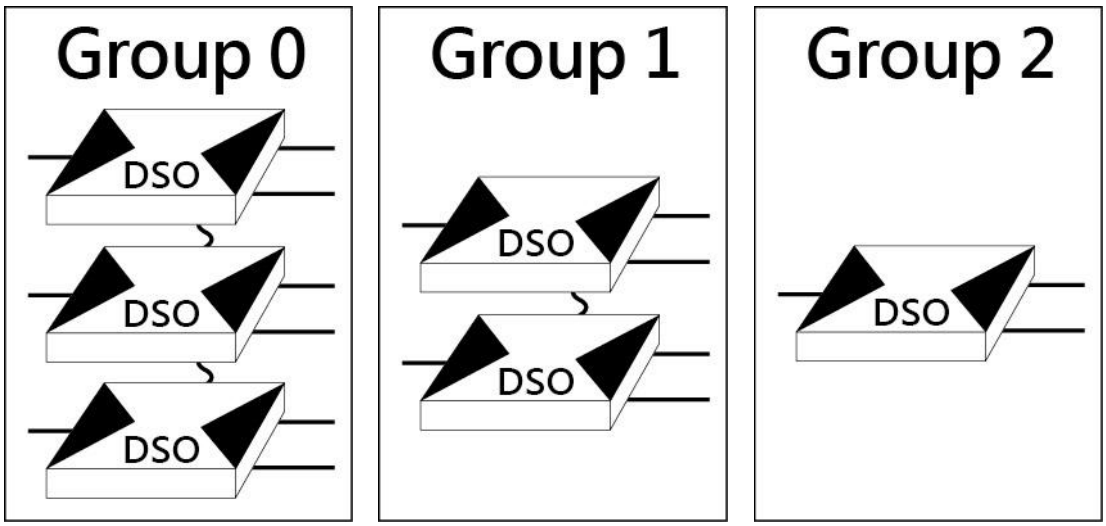
Return value

Return TRUE if success; otherwise, return FALSE.

Example

Below is the example for assigning 6 DSOs to 3 groups.

```
uDsoSDKSelectGroup(0);                          //Select Group 0
//Intial DSOs in group 0
uDsoSDKInitStack("TSA22120001,TSA22120002,TSA22120003");
uDsoSDKSelectGroup(1);                          //Select Group 1
uDsoSDKInitStack("TSA22120004,TSA22120005");    //Intial DSOs in group 1
uDsoSDKSelectGroup(2);                          //Select Group 2
uDsoSDKInitStack("TSA22120006");                //Intial DSOs in group 2
```



Retrieving DSO Hardware Parameters

BOOL uDsoSDKGetVendorName(int iDev, LPSTR lpszData)

Retrieving DSO vendor name.

Parameter

iDev[in]:

Type: int

Specifies the zero-based index of the DSO devices.

lpszData[out]:

Type: LPSTR

String buffer for the vendor name.

Return value

Return TRUE if success; otherwise, return FALSE.

BOOL uDsoSDKGetProductName(int iDev, LPSTR lpszData)

Retrieving DSO product name.

Parameter

iDev[in]:

Type: int

Specifies the zero-based index of the DSO devices.

lpszData[out]:

Type: LPSTR

String buffer for the product name.

Return value

Return TRUE if success; otherwise, return FALSE.

BOOL uDsoSDKGetSerialNum(int iDev, LPSTR lpszData)

Retrieving DSO serial number.

Parameter

iDev[in]:

Type: int

Specifies the zero-based index of the DSO devices.

lpszData[out]:

Type: LPSTR

String buffer for the serial number.

Return value

Return TRUE if success; otherwise, return FALSE.

BOOL uDsoSDKGetHwVer(int iDev, int * piHwVer)

Retrieving DSO hardware version.

Parameter

iDev[in]:

Type: int

Specifies the zero-based index of the DSO devices.

piHwVer[out]:

Type: int *

Integer data buffer for the hardware version.

Return value

Return TRUE if success; otherwise, return FALSE.

BOOL uDsoSDKGetFwVer(int iDev, int * piFwVer)

Retrieving DSO firmware version.

Parameter

iDev[in]:

Type: int

Specifies the zero-based index of the DSO devices.

piFwVer[out]:

Type: int *

Integer data buffer for the firmware version.

Return value

Return TRUE if success; otherwise, return FALSE.

BOOL uDsoSDKGetProductID(int iDev, int * piProductID)

Retrieving DSO product ID.

Parameter

iDev[in]:

Type: int

Specifies the zero-based index of the DSO devices.

piProductID[out]:

Type: int *

Integer data buffer for the product ID.

Return value

Return TRUE if success; otherwise, return FALSE.

BOOL uDsoSDKGetUsbDeviceHandle(int iDev, HANDLE hUsbHandle)

Retrieving DSO USB handle for USB unplug detect.

Parameter

iDev[in]:

Type: int

Specifies the zero-based index of the DSO devices.

hUsbHandle[out]:

Type: HANDLE

Data buffer for the USB handle.

Return value

Return TRUE if success; otherwise, return FALSE.

Example

```
//Register the USB handle to the window that handling the Windows Message loop
DEV_BROADCAST_HANDLE NotificationFilter;
ZeroMemory( &NotificationFilter, sizeof(NotificationFilter) );
NotificationFilter.dbch_size = sizeof(DEV_BROADCAST_HANDLE);
NotificationFilter.dbch_devicetype = DBT_DEVTYP_HANDLE;
NotificationFilter.dbch_handle = m_hDsoDev;
NotificationFilter.dbch_eventguid = MYGUID;
m_hDeviceNotify = RegisterDeviceNotification( hMainWin,
```

```
        &NotificationFilter, DEVICE_NOTIFY_WINDOW_HANDLE );

    // When Device changes, the WM_DEVICECHANGE message will be send to the
    registration window.
    LRESULT MsgDeviceChange( HWND hDlg, UINT uMessage, WPARAM wParam,
    LPARAM lParam )
    {
        if ( DBT_DEVICEARRIVAL == wParam ||
            DBT_DEVICEREMOVECOMPLETE == wParam )
        {
            PDEV_BROADCAST_HDR pHdr = (PDEV_BROADCAST_HDR)lParam;
            switch ( pHdr->dbch_devicetype )
            {
                case DBT_DEVTYP_HANDLE:
                    pDevHnd = (PDEV_BROADCAST_HANDLE)pHdr;
                    // Plug-out check here !!
                    break;
                ......
            }
        }
        return true;
    }
```

BOOL uDsoSDKGetCalibrationData(int iDev, double pdbValue[6])

Retrieving DSO voltage calibration data.

Parameter

iDev[in]:

Type: int

Specifies the zero-based index of the DSO devices.

pdbValue[out]:

Type: double *

Data array for the calibration value, the array size must be greater than 6.

Return value

Return TRUE if success; otherwise, return FALSE.

Remark

The calibration data is required for calculating the true voltage value from DSO RAW data. Users can either apply this formula by themselves, or use *uDsoSDKRawToDouble_mv* or *uDsoSDKRawToDouble_uv* provided by the SDK library to convert the whole waveform.

```
double dMul[2], dOfsA[2], dOfsB[2];
dMul[0] = pdbValue[0];
dOfsA[0] = pdbValue[1];
dOfsB[0] = pdbValue[2];
dMul[1] = pdbValue[3];
dOfsA[1] = pdbValue[4];
dOfsB[1] = pdbValue[5];

RealVolt = ((RawValue - dOfsA) * dMul + dOfsB – VOffset) * Probe;
Volt_CH1 = (RawValue_CH1 – dOfsA[0] ) * dMul[0] + dOfsB[0] – VOffset
RealVolt_CH1 = (Volt_CH1 – VOffset) * Probe;    // CH1 Voltage(uV)
Volt_CH2 = (RawValue_CH2 – dOfsA[1] ) * dMul[1] + dOfsB[1] – VOffset)
RealVolt_CH2 = (Volt_CH2 – VOffset) * Probe;    // CH2 Voltage(uV)
```

INT uDsoSDKGetDeviceCount( )

Retrieving the initialized device count from current selected group.

Return value

Returning the device count from current group.

Acquisition Parameter Setting

BOOL uDsoSDKSetSampleRate(__int64 i64SampleRate)

BOOL uDsoSDKGetSampleRate(__int64 & i64SampleRate)

Set/Read the DSO sampling rate, the default value is 10MSa/s

Parameter

i64SampleRate[in]:

Type: int

Input the DSO sampling rate, the avialable values were listed below. Unit: Samples per second (Sa/s).

DSO Model	Available sample rate
DS1002	2.5G S/s, 100M S/s, 50MS/s, 20MS/s, 10MS/s, 5MS/s, 2MS/s, 1MS/s, 500KS/s, 200KS/s, 100KS/s, 50KS/s, 20KS/s, 10KS/s, 5KS/s, 2KS/s, 1KS/s,500S/s, 200S/s, 100S/s.

DS1102 DS1202 DS1302	5GS/s, 200MS/s, 100MS/s, 50MS/s, 20MS/s, 10MS/s, 5MS/s, 1MS/s, 500KS/s, 200KS/s, 100KS/s, 50KS/s, 20KS/s, 10KS/s, 5KS/s, 2KS/s, 1KS/s,500S/s, 200S/s, 100S/s.
TS2202A TS2212A	25GS/s, 10GS/s, 5GS/s, 2.5GS/s, 1GS/s, 500MS/s, 250MS/s, 100MS/s, 50MS/s, 25MS/s, 10MS/s, 5MS/s, 2.5MS/s, 1MS/s, 500KS/s, 250KS/s, 100KS/s, 50KS/s, 25KS/s, 10KS/s, 5KS/s, 2.5KS/s 1KS/s, 500S/s, 200S/s, 100S/s.

Return value

Return TRUE if success; otherwise, return FALSE.

BOOL uDsoSDKSetRecordLength(int iRecordLength)
BOOL uDsoSDKSetRecordLength(int & iRecordLength)

Set/Read the DSO record length. There's no default value for this setting, record length must be assigned before any capture.

Parameter

iRecordLength[in]:

Type: int

Specified the waveform capture legnth. Please refer to the DSO user manual for the maximum record length. If the sample rate is 2.5GSa/s (DS1002) and 5GSa/s (DS1102, DS1202, DS1302), the record length must not greater than 25k points. The record length must be set to multiple of 8 for all DSO model.

Return value

Return TRUE if success; otherwise, return FALSE.

BOOL uDsoSDKSetWaitMode(int iWaitMode, __int64 i64CustomWaitTime_ps)
BOOL uDsoSDKGetWaitMode(int iWaitMode, __int64 &i64CustomWaitTime_ps)

Set/Read DSO trigger wait mode. DSO will automatically capture current waveform if there are no trigger signal matched in the wait time period. There's no default value for this setting, wait mode must be assigned before any capture.

Parameter

iWaitMode[in]:

Type: int

Parameter		TravelScope series	DS1000 series
WAIT_QUICK	0	Quick, approx. 1.28ms	Quick, approx. 6.25ms

WAIT_SLOW	1	Slow, approx. 1s	Slow, approx. 4.88s
WAIT_FOREVER	2	Forever	
WAIT_CUSTOM	3	Depend on user setting	

i64CustomWaitTime_ps[in]:

Type: int64

Custom value is only available when iWaitMode = WAIT_CUSTOM. Unit: ps.

Return value

Return TRUE if success; otherwise, return FALSE.

BOOL uDsoSDKSetDelayTime(__int64 i64DelayTime_ps)
BOOL uDsoSDKGetDelayTime(__int64 & i64DelayTime_ps)

Set/Read DSO trigger delay time, the default value is 0 ps.

Parameter

i64DelayTime_ps[in]:

Type: int64

Trigger delay time, unit: ps.

Return value

Return TRUE if success; otherwise, return FALSE.

BOOL uDsoSDKSetHoldoffTime(__int64 i64HoldOffTime_ps)
BOOL uDsoSDKGetHoldoffTime(__int64 & i64HoldOffTime_ps)

Set/Read DSO trigger holdoff time after previous trigger. The default value is 0 ps.

(Only available for TravelScope series)

Parameter

i64HoldOffTime_ps[in]:

Type: int64

Trigger holdoff time, unit: ps.

Return value

Return TRUE if success; otherwise, return FALSE.

BOOL uDsoSDKSetTrigPos(__int64 i64TrigPosition)
BOOL uDsoSDKGetTrigPos(__int64 & i64TrigPosition)

Set/Read DSO trigger position. The default value is 0 for center of waveform.

Parameter

i64TrigPosition[in]:

Type: int64

Trigger position value, unit: ps. When trigger position is 0, the trigger signal will be located at the center of waveform (50%), 0-50% if trigger position < 0, 50-100% if i64TrigPosition < 0.

Return value

Return TRUE if success; otherwise, return FALSE.

BOOL uDsoSDKSetBWL(int iCh, int iBwlFlag)
BOOL uDsoSDKGetBWL(int iCh, int & iBwlFlag)

Set/Read DSO bandwidth limited function, the default value is BWL_FULL.

Parameter

iCh[in]:

Type: int

Specifies the zero-based index of the DSO channel. Ex: 0 for channel 1.

iBwlFlag[in]:

Type: int

Parameter		Description
BWL_FULL	0	Full bandwidth
BWL_20M	1	Bandwidth limited to 20MHz
BWL_100M	2	Bandwidth limited to 100MHz ***(Only available for TravelScope series)***

Return value

Return TRUE if success; otherwise, return FALSE.

BOOL uDsoSDKSetCoupling(int iCh, int iCouplingFlag)
BOOL uDsoSDKGetCoupling(int iCh, int & iCouplingFlag)

Set/Read AC coupling function. The default value is COUPLING_DC for DC coupling.

Parameter

iCh[in]:

Type: int

Specifies the zero-based index of the DSO channel. Ex: 0 for channel 1.

iCouplingFlag[in]:

Type: int

Parameter		Description
COUPLING_DC	0	DC coupling mode
COUPLING_AC	1	AC coupling mode
COUPLING_GND	2	Ground coupling mode

Return value

Return TRUE if success; otherwise, return FALSE.

BOOL uDsoSDKSetChOnOff(int iCh, bool bChOn)
BOOL uDsoSDKGetChOnOff(int iCh)

Set/Read channel display on/off. The default display status is ON.

The maximum sample rate is 500MSa/s @ 2 Ch, 1GSa/s @ 1 Ch for TravelScope series;

100MSa/s @ 2 Ch, 200MSa/s @ 1 Ch for DS1302 / DS1202 / DS1102;

50MSa/s @ 2 Ch, 100MSa/s @ 1 Ch for DS1002.

Parameter

iCh[in]:

Type: int

Specifies the zero-based index of the DSO channel. Ex: 0 for channel 1.

bChOn[in]:

Type: bool

Specifies the channel display status, input TRUE for display ON.

Return value

Return TRUE if success; otherwise, return FALSE.

BOOL uDsoSDKSetAcquireMode(int iCh, int iAcquireMode)
BOOL uDsoSDKGetAcquireMode(int iCh, int & iAcquireMode)

Set/Read DSO acquisition mode for specified channel. The default value is

ACQ_SAMPLE for Sample mode.

Parameter

iCh[in]:

Type: int

Specifies the zero-based index of the DSO channel. Ex: 0 for channel 1.

iAcquireMode[in]:

Type: int

Parameter		Description
ACQ_SAMPLE	0	Sample mode
ACQ_PEAKDETECT	4	Peak-detect mode
ACQ_HIRES	5	High resolution mode

Return value

Return TRUE if success; otherwise, return FALSE.

BOOL uDsoSDKSetVoltDiv(int iCh, int iVoltDiv_uV)
BOOL uDsoSDKGetVoltDiv(int iCh, int & iVoltDiv_uV)

Set/Read DSO Voltage Division for specified channel. The default value is 100mV on probe x1.

Parameter

iCh[in]:

Type: int

Specifies the zero-based index of the DSO channel. Ex: 0 for channel 1.

iVoltDiv_uV[in]:

Type: int

Specified the voltage division of the channel, unit: uV.

Available voltage division
10V, 5V, 2V, 1V, 500mV, 200mV, 100mV, 50mV, 20mV, 10mV, 5mV, 2mV

Return value

Return TRUE if success; otherwise, return FALSE.

BOOL uDsoSDKSetVoltPos(int iCh, int iPosition)
BOOL uDsoSDKGetVoltPos(int iCh, int & iPosition)

Set/Read channel label position. The default value is 8000, means center of waveform.

Parameter

iCh[in]:

Type: int

Specifies the zero-based index of the DSO channel. Ex: 0 for channel 1.

iPosition[in]:

Type: int

Specified the channel label position. Input 0 for the bottom of waveform; 16000 for

the top of waveform.

Return value

Return TRUE if success; otherwise, return FALSE.

BOOL uDsoSDKSetVoltOfs(int iCh, int iVoltOfs_uV)

BOOL uDsoSDKGetVoltOfs(int iCh, int & iVoltOfs_uV)

Set/Read DSO voltage offset for specified channel. The default value is 0V.

(Only Available for TravelScope series)

Parameter

iCh[in]:

Type: int

Specifies the zero-based index of the DSO channel. Ex: 0 for channel 1.

iVoltOfs_uV[in]:

Type: int

Specified the voltage offset value, unit: uV.

Return value

Return TRUE if success; otherwise, return FALSE.

Trigger Setting

There's no default trigger setting, users must select one of the follow trigger condition before capture any waveform.

BOOL uDsoSDKSetEdgeTrig(int iSrc, int iSlope, __int64 i64Threshold_uV)

Set the trigger condition to Edge trigger.

Parameter

iSrc[in]:

Type: int

Parameter		Description
TRIG_SOURCE_CH1	0	Trigger source from channel 1.
TRIG_SOURCE_CH2	1	Trigger source from channel 2.
TRIG_SOURCE_EXT	2	Trigger source from external trigger in.

iSlope[in]:

Type: int

Parameter	Description

TRIG_EDGE_RISING	0	Rising edge
TRIG_EDGE_FALLING	1	Falling edge
TRIG_EDGE_EITHER	2	Either edge
TRIG_EDGE_ALTERNATE	3	Alternate edge

i64Threshold_uV[in]:

Type: __int64

Trigger threshold, unit: uV.

Return value

Return TRUE if success; otherwise, return FALSE.

BOOL uDsoSDKSetVideoTrig(int iSrc, int iMode, int iScanline)

Set the trigger condition to Video trigger.

Parameter

iSrc[in]:

Type: int

Parameter		Description
TRIG_SOURCE_CH1	0	Trigger source from channel 1.
TRIG_SOURCE_CH2	1	Trigger source from channel 2.
TRIG_SOURCE_EXT	2	Trigger source from external trigger in.

iMode[in]:

Type: int

Parameter		Description
TRIG_VIDEO_MODE_SCANLINE	0	Trigger on specified Scan line.
TRIG_VIDEO_MODE_ANYFIELD	1	Trigger on any field.
TRIG_VIDEO_MODE_ODDFIELD	2	Trigger on odd field.
TRIG_VIDEO_MODE_EVENFIELD	3	Trigger on even field.

iScanline[in]:

Type: int

Specified scan line number, only available when selecting scan line trigger.

Return value

Return TRUE if success; otherwise, return FALSE.

BOOL uDsoSDKSetRuntTrig(int iSrc, int iPolarity, int iCompareType, int iEqualRange, __int64 i64Width_ps, __int64 i64ThresholdA_uV, __int64 i64ThresholdB_uV)

Set trigger condition to runt trigger. (Only available for TravelScope series)

Parameter

iSrc[in]:

Type: int

Parameter		Description
TRIG_SOURCE_CH1	0	Trigger source from channel 1.
TRIG_SOURCE_CH2	1	Trigger source from channel 2.
TRIG_SOURCE_EXT	2	Trigger source from external trigger in.

iPolarity[in]:

Type: int

Parameter		Description
TRIG_RUNT_POLARITY_HIGHPULSE	0	Trigger on positive runt.
TRIG_RUNT_POLARITY_LOWPULSE	1	Trigger on negative runt.
TRIG_RUNT_POLARITY_EITHER	2	Trigger on any runt.

iCompareType[in]:

Type: int

Parameter		Description
TRIG_RUNT_CMP_LESS	0	Runt width lesser than specified width.
TRIG_RUNT_CMP_GREAT	1	Runt width greater than specified width.
TRIG_RUNT_CMP_EQUAL	2	Runt width equal to specified width.
TRIG_RUNT_CMP_NOT_EQUAL	3	Runt width not equal to specified width.

iEqualRange[in]:

Type: int

Specified the runt width tolerance range, standard setting is 5 (5%), the acceptable error will be 10ms x 5% = 500ns when specified width is 10ms.

i64Width_ps[in]:

Type: __int64

Runt width setting, unit: ps. Input 0 when no width compare required.

i64ThresholdA_uV[in]:

Type: __int64

Trigger threshold A, unit: uV.

i64ThresholdB_uV[in]:

Type: __int64

Trigger threshold B, unit: uV.

Return value

Return TRUE if success; otherwise, return FALSE.

BOOL uDsoSDKSetWidthTrig(int iSrc, int iPolarity, int iCompareType, int iEqualRange, __int64 i64Width_ps, __int64 i64Threshold_uV)

Set trigger condition to width trigger. (Only available for TravelScope series)

Parameter

iSrc[in]:

Type: int

Parameter		Description
TRIG_SOURCE_CH1	0	Trigger source from channel 1.
TRIG_SOURCE_CH2	1	Trigger source from channel 2.
TRIG_SOURCE_EXT	2	Trigger source from external trigger in.

iPolarity[in]:

Type: int

Parameter		Description
TRIG_WIDTH_POLARITY_HIGHPULSE	0	Trigger on high pulse.
TRIG_WIDTH_POLARITY_LOWPULSE	1	Trigger on low pulse.
TRIG_WIDTH_POLARITY_EITHER	2	Trigger on any pulse.

iCompareType[in]:

Type: int

Parameter		Description
TRIG_WIDTH_CMP_LESS	0	Pulse width lesser than specified width.
TRIG_WIDTH_CMP_GREAT	1	Pulse width greater than specified width.
TRIG_WIDTH_CMP_EQUAL	2	Pulse width equal to specified width.
TRIG_WIDTH_CMP_NOT_EQUAL	3	Pulse width not equal to specified width.

iEqualRange[in]:

Type: int

Specified the pulse width tolerance range, standard setting is 5 (5%), the acceptable error will be 10ms x 5% = 500ns when specified width is 10ms.

i64Width_ps[in]:

Type: __int64

Pulse width setting, unit: ps.

i64Threshold_uV[in]:

Type: __int64

Trigger threshold, unit: uV.

Return value

Return TRUE if success; otherwise, return FALSE.

BOOL uDsoSDKSetPatternTrig(int iSrcA, int iSrcB, int iNotA, int iNotB, int iAndOr, __int64 i64Width_ps, __int64 i64ThresholdA_uV, __int64 i64ThresholdB_uV)

Set trigger condition to pattern trigger. (Only available for TravelScope series.)

Parameter

iSrcA[in]:

Type: int

Trigger source A.

Parameter		Description
TRIG_SOURCE_CH1	0	Trigger source from channel 1.
TRIG_SOURCE_CH2	1	Trigger source from channel 2.
TRIG_SOURCE_EXT	2	Trigger source from external trigger in.

iSrcB[in]:

Type: int

Trigger source B.

Parameter		Description
TRIG_SOURCE_CH1	0	Trigger source from channel 1.
TRIG_SOURCE_CH2	1	Trigger source from channel 2.
TRIG_SOURCE_EXT	2	Trigger source from external trigger in.

iNotA[in]:

Type: int

Sets the signal logic for input source A, H = high true, L = low true.

iNotB[in]:

Type: int

Sets the signal logic for input source B, H = high true, L = low true.

iAndOr[in]:

Type: int

Sets which logic function to apply to the input signals.

Parameter		Description
TRIG_PATTERN_ANDOR_AND	0	Logic AND
TRIG_PATTERN_ANDOR_OR	1	Logic OR

i64Width_ps[in]:

Type: __int64

Specified width, unit: ps.

i64ThresholdA_uV[in]:

Type: __int64

Trigger threshold A, unit: uV.

i64ThresholdB_uV[in]:

Type: __int64

Trigger threshold B, unit: uV.

Return value

Return TRUE if success; otherwise, return FALSE.

BOOL uDsoSDKSetStateTrig(int iSrcA, int iSrcB, int iNotA, int iNotB, int iAndOr, __int64 i64ThresholdA_uV, __int64 i64ThresholdB_uV)

Set trigger condition to state trigger. (Only available for TravelScope series.)

Parameter

iSrcA[in]:

Type: int

Trigger source A.

Parameter		Description
TRIG_SOURCE_CH1	0	Trigger source from channel 1.
TRIG_SOURCE_CH2	1	Trigger source from channel 2.
TRIG_SOURCE_EXT	2	Trigger source from external trigger in.

iSrcB[in]:

Type: int

Trigger source B.

Parameter		Description
TRIG_SOURCE_CH1	0	Trigger source from channel 1.
TRIG_SOURCE_CH2	1	Trigger source from channel 2.
TRIG_SOURCE_EXT	2	Trigger source from external trigger in.

iNotA[in]:

Type: int

Sets the signal logic for input source A, H = high true, L = low true.

iNotB[in]:

Type: int

Sets the signal logic for input source B, H = high true, L = low true.

iAndOr[in]:

Type: int

Sets which logic function to apply to the input signals.

Parameter		Description
TRIG_PATTERN_ANDOR_AND	0	Logic AND
TRIG_PATTERN_ANDOR_OR	1	Logic OR

i64ThresholdA_uV[in]:

Type: __int64

Trigger threshold A, unit: uV.

i64ThresholdB_uV[in]:

Type: __int64

Trigger threshold B, unit: uV.

Return value

Return TRUE if success; otherwise, return FALSE.

BOOL uDsoSDKSetTrigCouple(int iTrigCoupleFlag)

BOOL uDsoSDKGetTrigCouple(int & iTrigCoupleFlag)

Set/Read trigger coupling mode. The default value is REJECT_NONE. (Only available for TravelScope series.)

Parameter

iTrigCoupleFlag[in]:

Type: int

Parameter		Description
REJECT_NONE	0	None
REJECT_HF	1	High Frequency Reject
REJECT_LF	2	Low Frequency Reject
REJECT_NOISE	3	Noise Reject

Return value

Return TRUE if success; otherwise, return FALSE.

BOOL uDsoSDKCaptureEx( )

Start the DSO capturing.

Return value

Return TRUE if success; otherwise, return FALSE.

Remark

All the capture parameters and trigger setting must be completed before starting capture.

BOOL uDsoSDKStop()

Stop the DSO capturing.

Return value

Return TRUE if success; otherwise, return FALSE.

BOOL uDsoSDKForceTrig()

Force DSO to trigger.

Parameters

iDev[in]:

Type: int

Specifies the zero-based index of the of DSO devices.

Return value

Return TRUE if success; otherwise, return FALSE.

Remark

When trigger mode is in normal mode or single shot mode and trigger fails, call this function will force DSO to trigger.

BOOL uDsoSDKReadIniFile(LPCSTR szFilePath)

Set the DSO capture parameters and trigger setting from .ini file.

Parameter

szFilePath[in]:

Type: LPCSTR

Ini file path, either relative path (../DSOini.ini) or absolute path (D:\Dso.ini) are acceptable here.

Return value

Return TRUE if success; otherwise, return FALSE.

Remark

Please refer to DSO.ini or Dso_7Dev.ini for detail text format and description.

Reading DSO Capturing Status

int uDsoSDKGetErrorCodeEx()

Retrieves the last-error code value.

Return value

Parameter		Description
DSO_ERROR_BADPOINTER	0x01	User inputs bad pointer or buffer without enough space.
DSO_UNDEFINED_PARAMETER	0x02	User inputs an undefined parameter.
DSO_CHANNELINDEX_OUT_RANGE	0x03	User inputs a channel index > current initialed device number x 2.
DSO_DEVICEINDEX_OUT_RANGE	0x04	User inputs a device index > current initialed device number.
DSO_UNSUPPORT_VOLTDIV	0x05	User inputs an unsupported voltage division.
DSO_VOLTPOS_OUT_RANGE	0x06	Input voltage position is out of range.
DSO_TRIGPOS_OUT_RANGE	0x07	Input trigger position is out of range.
DSO_TREASHOLD_OUT_RANGE	0x08	Input voltage threashold is out of range.
DSO_WAIT_TIME_OUT_RANGE	0x09	Input trigger wait time is out of range.
DSO_TRIGGER_NOT_SUPPORT	0x0A	Trigger mode not supported in current device.
DSO_VIDEO_TRIG_ERROR	0x0B	Video trigger cannot be combined with Equivalent mode.
DSO_TRIG_WAIT_ERROR	0x0C	Trigger wait cannot be combined with Equivalent mode.
DSO_RECORD_LENGTH_ERROR	0x0D	Record length must be multiple of 8, and the available range is different from DSO model.
DSO_NO_HARDWARE	0x0E	No Dso Hardware found.
DSO_UNKNOWN_ERROR	0x0F	Unknown error.
DSO_SAMPLE_RATE_ERROR	0x10	User inputs an unsupported sample rate.
DSO_MEMORY_NOT_ENOUGH	0x11	SDK fail to allocate enough memory.
DSO_MEASURE_TYPE_ERROR	0x12	Selected an invalid measurement type.

DSO_MEASURE_RANGE_ERROR	0x13	Measurement Start/End range error.
DSO_MEASURE_DATA_ERROR	0x14	Measurement pre-scan data error, cannot calculate valid result.
DSO_GROUP_INDEX_OUT_RANGE	0x15	The group number is limited from 0 to 19.
DSO_MAIN_DLL_NOT_FOUND	0x16	DsoRun.dll not exist.
DSO_MAIN_DLL_FUNCTION_ERR	0x17	DsoRun.dll version too old.
DSO_MAIN_DLL_LOAD_ERR	0x18	SDK failed to load DsoRun.dll.
DSO_GET_TEMP_NAME_ERR	0x19	SDK can not get Temp file name.
DSO_GET_TEMP_DIR_ERR	0x1A	SDK can not get the Temp folder directory.
DSO_FILE_COPY_ERR	0x1B	SDK failed to copy DsoRun.dll to Temp folder.
DSO_NOT_INITIAL	0x1C	DSO Hardware not initialed.
DSO_GROUP_PARAMETER_ERR	0x1D	DSO Group Parameter Error.
DSO_FILE_NOT_EXIST	0x1E	Target File doesn't exist.
DSO_REINITIAL	0x1F	The DSO group had been initialed, call DSO shutdown fucntion before reinitial.
DSO_USB_DRIVER_ERROR	0x20	Some errors happened in DSO USB driver layer, please re-plug the USB cable to solve this problem.
DSO_USB_UNPLUG	0x21	The USB cable has been unplugged.
DSO_STACK_ID_ERROR	0x22	The input DSO stack serial number is invalid. Please refer to the user manual for the correct format.
DSO_MUST_NOT_INITIAL	0x23	Environment settings must be done before DSO initialization.
DSO_READ_EMPTY	0x24	DSO data is empty, need to call uDsoSDKCapture before read data.
DSO_SEQUENCE_ERROR	0x25	Functions initial setting not ready, need to call the initial function before use.

int uDsoSDKGetStatus(int iDev)

Retrieves the capturing status from specified DSO.

Parameter

iDev[in]:

Type: int

Specifies the zero-based index of the of DSO devices.

Return value

Parameter	Description
9	The DSO is filling the pretrigger portion of the waveform.

5	The DSO is waiting for a valid trigger signal to occur.
3	The DSO is filling the posttrigger portion of the waveform.
1	The DSO is sending data to PC.
0	Data capture completed.
32	USB driver error.
64	USB device unplugged.

BOOL uDsoSDKDataReady()

Retrieves the capturing status from all DSO.

Return value

Return TRUE if the DSO capturing is finished. Otherwise, return FALSE.

Retrieving the DSO Waveform and Measurement Function

SDK.dll is using RAW data format for reading and measurement function, and Double data format (mV) for the FFT function. The convertion functions are provided in SDK.dll to convert the RAW to Double (mV or uV) data format.

BOOL uDsoSDKReadExRaw(int iDev, int* piFlag, short* lpsData, double pdbYofsA[2], double pdbYofsB[2], double pdbYMul[2])

Retrieving the DSO waveform and calibration parameter from specified device.

Parameter

iDev [in]:

Type: int

Specifies the zero-based index of the of DSO devices.

piFlag [out]:

Type: int *

Returning the capture status of DSO, each state can be combined with OR operation. Input a NULL pointer to ignore the status.

Parameter		Description
DSDF_ETSMODE	0x02000	Current waveform is captured in ETS mode.
DSDF_DBLMODE	0x04000	Current waveform is captured in single channel mode with 200MSa/s. (Only apply to DS1000 series.)
DSDF_TIMEOUT	0x08000	There's no valid trigger occurred within trigger wait time.
DSDF_TRIGSRC_CH1	0x10000	Indicate the trigger source is from Channel 1
DSDF_TRIGSRC_CH2	0x20000	Indicate the trigger source is from Channel 2

lpsData [out]:

Type: short *

The Waveform buffer. The 2-channel's waveform will be read at the same time regardless the channel display status. The former half of the buffer is for Ch1, and the other half is for Ch2. The required buffer size can be calculated as below:

Buffer Size = Record Length * 2 Channel * Sizeof (short)

Record Length:

According to the record length set when capturing, unit: sample point.

pdbYofsA, pdbYofsB, pdbYMul [out]:

Type: double *

Calibration parameter for true voltage value calculation.

Return value

Return TRUE if success; otherwise, return FALSE.

Remark

Example for retrieving waveform and voltage calculation:

```
short       *pWaveData;
short       *pCH1, *pCH2;
int         iData_CH1, iData_CH2;
pWaveData = new short[2 * iRecordLength];      // Prepare the waveform buffer
pCH1 = m_pWaveData;                            // Set CH1 waveform pointer
pCH2 = m_pWaveData + iRecordLength;            // Set CH2 waveform pointer

// Calculate the voltage value for the first point of the waveform.
iData_CH1 = ((pCH1[0]– dYOfsA[0] ) * dYMul[0] + dYOfsB[0] – VOffset) * Probe;
iData_CH2 = ((pCH2[0]– dYOfsA[1] ) * dYMul[1] + dYOfsB[1] – VOffset) * Probe;
```

The calculation formula and waveform data format will be the same regardless the channel display status.

BOOL uDsoSDKRawToDbl_mv(short* lpsSrc, double * lpdbDst, int iLength, double dbYofsA, double dbYofsB, double dbYMul, double dbVOffset_uv, int iProbe)

BOOL uDsoSDKRawToDbl_uv(short* lpsSrc, double * lpdbDst, int iLength, double dbYofsA, double dbYofsB, double dbYMul, double dbVOffset_uv, int iProbe)

Convert the RAW Data format to Double (mV) or Double (uV) format.

Parameter

lpsSrc[in]:

Type: short *

Input the DSO waveform data in RAW data format.

lpdbDst[out]:

Type: double *

Convert and output the DSO waveform data in Double data format.

iLength[in]:

Type: int

Waveform data length, unit: sample point.

dbYofsA, dbYofsB, dbYMul[in]:

Type: int

Calibration value for the waveform.

dbVOffset_uv[in]:

Type: double

Waveform voltage offset value, unit: uV.

iProbe[in]:

Type: int

Current probe attenuation ratio.

Return value

Return TRUE if success; otherwise, return FALSE.

BOOL uDsoSDKGetFFTData(double* lpdbSrc_mv, int iRecordLength, int iType, int iWindow, double* lpdbFFT)

Fast Fourier transform for the DSO waveform in Double data format.

Parameter

lpdbSrc_mv[in]:

Type: double *

DSO waveform in Double (mV) format.

iRecordLength[in]:

Type: int

Waveform length, unit: sample point.

iType[in]:

Type: int

Specified the FFT scale.

Parameter	Description

FFT_TYPE_LINEAR_RMS	0	Linear RMS
FFT_TYPE_DBV_RMS	1	dbV RMS
FFT_TYPE_DBM_RMS	2	dBm RMS

iWindow[in]:

Type: int

Specified the FFT window.

參數		說明
FFT_WINDOW_RECTANGULAR	0	Rectanglar
FFT_WINDOW_BLACKMAN	1	Blackman
FFT_WINDOW_HANN	2	Hann
FFT_WINDOW_HAMMING	3	Hamming
FFT_WINDOW_HARRIS	4	Harris
FFT_WINDOW_TRIANGULAR	5	Triangular
FFT_WINDOW_COSINE	6	Cosine
FFT_WINDOW_LANCZOS	7	Lanczos
FFT_WINDOW_GUASS	8	Guass

***lpdbFFT[out]:**

Type: double

Data buffer to save the waveform data. The requiring size is the same with the waveform length.

Return value

Return TRUE if success; otherwise, return FALSE.

LPDSMEAS Structure

```
typedef struct _MEASUREDATA
{
    __int64    i64ScData;
    int        iRecordLength;
    int        iVoltDiv[2];
    double     dYOfsA[2];
    double     dYOfsB[2];
```

```
    double    dYMul[2];       // Volt(mV) = ( Raw Data - dYOfsA ) * dYMul + dYOfsB
    double    dVOfs[2];
    int       iProbe[2];
    LPWORD    lpwWaveData[2];
}DSMEAS, FAR *LPDSMEAS;
```

The structure will store both target and compare channel's parameters, the compare channel's parameters will only be used in some channel compare measurements, such as rising delay and phase delay…etc.

i64ScData:

Sampling rate of the waveform, unit: Sa/s.

iRecordLength:

Record length of the waveform, unit: sample point.

iVoltDiv:

The voltage division of the target and compare channel, unit: uV.

dYOfsA、dYOfsB、dYMul:

The calibration parameter of the target and compare channel.

dbVoltOfs:

The voltage offset of the target and compare channel, unit: uV.

iProbe:

The probe attenuation of the target and compare channel, ex: x1, x10, x100…

lpwWaveData:

The waveform data of target and compare channel in RAW data format.

BOOL uDsoSDKMeasurement(int * piType, int iStart, int iEnd, LPDSMEAS lpDsMeas, bool * pfForceStop, bool * pfResult, double * dbValue)

Calculate the waveform with specified measurement types.

Parameter

piType[in]:

Type: int *

Input the required measurement index array, and put MEASURE_END at the end of the array.

Parameter		Description
MEASURE_FREQ	0	Frequency
MEASURE_PERIOD	1	Period

MEASURE_VMAX	2	Voltage maximum
MEASURE_VMIN	3	Voltage minimum
MEASURE_VHIGH	4	Voltage high
MEASURE_VLOW	5	Voltage low
MEASURE_VPP	6	Voltage peak to peak
MEASURE_VAMP	7	Voltage amplitude
MEASURE_RMS	8	Voltage RMS
MEASURE_VMEAN	9	Voltage mean
MEASURE_HIGH_DUTY	10	High duty cycle percentage
MEASURE_LOW_DUTY	11	Low duty cycle percentage
MEASURE_HIGH_PERIOD	12	High pulse period
MEASURE_LOW_PERIOD	13	Low pulse period
MEASURE_RISETIME	14	Rise time
MEASURE_FALLTIME	15	Fall time
MEASURE_POVERSHOOT	16	Positive overshoot
MEASURE_NOVERSHOOT	17	Negative overshoot
MEASURE_VMID	18	Voltage mid
MEASURE_CYCLE_RMS	19	First cycle RMS
MEASURE_CYCLE_MEAN	20	First cycle voltage mean
MEASURE_END	-1	Measurement end mark

iStart[in]:

Type: int

Specified the measurement start position, input 0 for the start of waveform. Unit: sample point.

iEnd[in]:

Type: int

Specified the measurement end position, input 0 for the start of waveform. Unit: sample point.

lpDsMeas[in]:

Type: LPDSMEAS

Input the measurement structure, including the waveform data and parameters.

pfForceStop[in]:

Type: bool *

The force stop flag. Users can stop the measurement calculation in multi-thread architecture by set this flag to TRUE.

pfResult[out]:

Type: bool *

Returning the calculation success flag of each measurement. Return TRUE for success, return FALSE when calculation failed.

pdbValue[out]:

Type: double *

Returning the result of each measurement.

Return value

Return TRUE if success; otherwise, return FALSE.

Remark

Example for measuring frequency and RMS in the same time.

Parameter setting:

```
int iType[3] = {
    MEASURE_FREQ,
    MEASURE_RMS,
    MEASURE_END
};
bool fResult[3] = {0};
double dbValue[3] = {0};
```

After excuted the measurement function:

fResult[0] represent the calculation success flag of MEASURE_FREQ.

fResult[1] represent the calculation success flag of MEASURE_RMS.

dbValue[0] represent the measurement value of MEASURE_FREQ.

dbValue[1] represent the measurement value of MEASURE_RMS.

Function Generator Control Functions

BOOL uDsoSDKFGSetting(int iDev, LPVOID lpvData)

Set FG. (Only available for TravelScope series.)

Parameter

iDev[in]:

Type:int

Specifies the zero-based index of the of DSO devices.

lpvData[in]:

Type:LPVOID

Input the FG parameter structure.

FG Function Select:

Parameter		Description
FGSDK_DATA_CH1 FGSDK_DATA_CH2	0x0000 0x0005	Set the Carrier waveform of specified channel. Corresponding structure: ***CarrierParameter***
FGSDK_CW_MODE_CH1 FGSDK_CW_MODE_CH2	0x0001 0x0006	Set the operation mode to Continuous mode. Corresponding structure: ***NoParameter***
FGSDK_MODUL_MODE_CH1 FGSDK_MODUL_MODE_CH2	0x0002 0x0007	Set the operation mode to Modulation mode. Corresponding structure: ***ModulationParameter***
FGSDK_SWEEP_MODE_CH1 FGSDK_SWEEP_MODE_CH2	0x0003 0x0008	Set the operation mode to Sweep mode. Corresponding structure: ***SweepParameter***
FGSDK_BURST_MODE_CH1 FGSDK_BURST_MODE_CH2	0x0004 0x0009	Set the operation mode to Burst mode. Corresponding structure: ***BurstParameter***
FGSDK_TRIG_DATA	0x000A	Set the Burst/Sweep trigger parameter. Corresponding structure: ***TriggerParameter***
FGSDK_SENDPTN	0x000B	Set the FG to send the pattern. Corresponding structure: ***NoParameter***
FGSDK_PHASERESET	0x000C	Set the FG to reset the waveform phase between two channels. Corresponding structure: ***NoParameter***
FGSDK_ARRAY_MODE	0x1000	Combine with other parameter to access the FG setting by using array instead of data structure.

FG Data Structure:

```
typedef struct _CARRIER_PARAMETER
{
    double dbID;          //FG function select index
    int iWaveFunction;    // Wave function
    double dbFreq;        // Frequency (Range: 0.01Hz ~ 1MHz)
    double dbPhase;       // Phase (Range: -180~180 degree)
    double dbDuty;        // Duty cycle (Range: 0~100%)
    double dbLeading;     // Leading time (μs)
```

```
    double dbTrailing;      // Trailing time (μs)
    double dbVoltAmp;       // Voltage amplitude. (Range: 0~2.5V)
    double dbVoltOffset;    // Voltage offset
} CarrierParameter;
```

iWaveFunction:

Parameter		Description
FGSDK_WAVE_DC	0x0000	DC
FGSDK_SINE	0x0001	Sine wave
FGSDK_SQUARE	0x0002	Square wave
FGSDK_TRIANGLE	0x0003	Triangle wave
FGSDK_RAMP	0x0004	Ramp
FGSDK_PULSE	0x0005	Pulse

```
typedef struct _MODULATION_PARAMETER
{
    double dbID;            //FG function select index
    int iWaveFunction;      //Wave function
    double dbFreq;          //Frequency (Range: 0.01Hz ~ 1MHz)
    double dbPhase;         //Phase (Range: -180~180 degree)
    double dbDuty;          //Duty cycle (Range: 0~100%)
    double dbLeading;       //Leading time (μs)
    double dbTrailing;      //Trailing time (μs)
    int iModulType;         //Modulation type
    double dbModulScale;    //Modulation scale
} ModulationParameter;
```

iWaveFunction:

Parameter		Description
FGSDK_WAVE_DC	0x0000	DC wave is not supported in the modulation mode.
FGSDK_SINE	0x0001	Sine wave
FGSDK_SQUARE	0x0002	Square wave
FGSDK_TRIANGLE	0x0003	Triangle wave
FGSDK_RAMP	0x0004	Ramp
FGSDK_PULSE	0x0005	Pulse

iModulType:

Parameter		Description
FGSDK_MODUL_AM	0x0000	AM, modulation scale range: ±100%
FGSDK_MODUL_FM	0x0001	FM, modulation scale range: ± Carrier frequency.
FGSDK_MODUL_PM	0x0002	PM, modulation scale range: ±180 degree
FGSDK_MODUL_ASK	0x0003	ASK, modulation scale range: ±100%
FGSDK_MODUL_FSK	0x0004	FSK, modulation scale range: 0Hz to 1MHz
FGSDK_MODUL_PSK	0x0005	PSK, modulation scale range: ±180 degree

```
typedef struct _SWEEP_PARAMETER
{
    double dbID;            //FG function select index
    bool bSweepMode;        //Sweep mode: repeat/trigger
    bool bSweepType;        //Sweep type: linear/log
    double dbStartFreq;     //Start frequency (Range:1Hz ~ 1MHz)
    double dbStopFreq;      //Stop frequency (Range:1Hz ~ 1MHz)
    double dbSweepTime;     //Sweep time (Range:1μs ~ 100s)
    double dbHoldTime;      //Hold time (Range:1μs ~ 100s)
    double dbReturnTime;    //Return time (Range:1μs ~ 100s)
} SweepParameter;
```

bSweepMode:

Parameter		Description
FGSDK_SWEEP_MODE_TRIG	0x0000	Trigger mode. The FG generates one sweep cycle after each trigger.
FGSDK_SWEEP_MODE_REPEAT	0x0001	Repeat mode. The FG generates sweep cycle repeatedly.

bSweepType:

Parameter		Description
FGSDK_SWEEP_TYPE_LINEAR	0x0000	Sweep frequency will increase / decrese linearly.
FGSDK_SWEEP_TYPE_LOG	0x0001	Sweep frequency will increase / decrese logarithmically.

```
typedef struct _BURST_PARAMETER
```

```
{
    double dbID;            //FG function select index
    int iBurstCycle;        //Burst count (Range:1 ~ 99999)
    int iTrigDelay;         //Trigger delay (Range: 1µs ~ 343s)
} BurstParameter;

typedef struct _TRIG_PARAMETER
{
    double dbID;            //FG function select index
    bool bTrigSource;       //Trigger source
    bool bTrigEdge;         //Trigger edge: rising / falling
    double dbTrigFreq;      //internal trigger freq. (Range: 0.01Hz ~ 1MHz)
} TriggerParameter;
```

bTrigSource:

Parameter		Description
FGSDK_TRIG_TYPE_INT	0x0000	Use internal trigger.
FGSDK_TRIG_TYPE_EXT	0x0001	Use external trigger.

bTrigEdge:

Parameter		Description
FGSDK_EDGE_FALLING	0x0000	Falling edge
FGSDK_EDGE_RISING	0x0001	Rising edge

```
typedef struct _NO_PARAMETER
{
    double dbID;            //FG function select index
}NoParameter;
```

Return value

Return TRUE if success; otherwise, return FALSE.

Ex1 Struct type:

```
CarrierParameter sCh1Data = {
    FGSDK_DATA_CH1,     // FG function index
    FGSDK_PULSE,        // Wave function
    1000,               // Frequency
    0,                  // Phase
    35,                 // Duty cycle
    100,                // Leading time
    200,                // Trailing time
    2.0,                // Voltage amplitude
    1.0,                // Voltage offset
};
CarrierParameter sCh2Data = {
    FGSDK_DATA_CH2,     // FG function index
    FGSDK_SINE,         // Wave function
    1500,               // Frequency
    20,                 // Phase
    0,                  // Duty cycle
    0,                  // Leading time
    0,                  // Trailing time
    2.5,                // Voltage amplitude
    1.25,               // Voltage offset
};
ModulationParameter sCh1ModulData = {
    FGSDK_MODUL_MODE_CH1, // FG function index
    FGSDK_SINE,           // Wave function
    100,                  // Frequency
    0,                    // Phase
    0,                    // Duty cycle
    0,                    // Leading time
    0,                    // Trailing time
    FGSDK_MODUL_AM,       // Modulation type
    100,                  // Modulation scale
};
```

```
TriggerParameter sTrigParameter =       {
        FGSDK_TRIG_DATA,                // FG function index
        FGSDK_TRIG_TYPE_INT,            // Trigger source
        FGSDK_EDGE_RISING,              // Trigger edge
        100,                            // Internal Trigger source frequency
};
NoParameter sCh2CWMode = {
        FGSDK_CW_MODE_CH2
};
NoParameter sSendPtn =      {
        FGSDK_SENDPTN
};
//Set Ch1 carrier parameters
puDsoSetting(iDevice, DSS_FGCONTROL, & sCh1Data);
//Set Ch2 carrier parameters
puDsoSetting(iDevice, DSS_FGCONTROL, & sCh2Data);
//Set Ch1 to Modulation mode
puDsoSetting(iDevice, DSS_FGCONTROL, & sCh1ModulData);
//Set Ch2 to Continuous Wave mode
puDsoSetting(iDevice, DSS_FGCONTROL, & sCh2CWMode);
//Set Trigger data
puDsoSetting(iDevice, DSS_FGCONTROL, & sTrigParameter);
//Send FG pattern
puDsoSetting(iDevice, DSS_FGCONTROL, & sSendPtn);
```

Ex2 Array type:

```
double pdbParaCh1[9] = {
    FGSDK_DATA_CH1 | FGSDK_ARRAY_MODE, // FG function index
    FGSDK_PULSE,          // Wave function
    1500,                 // Frequency
    20,                   // Phase
    20,                   // Duty cycle
    30,                   // Leading time
    70,                   // Trailing time
    2.5,                  // Voltage amplitude
    1.25,                 // Voltage offset
};
double pdbModulCh1[9] = {
    FGSDK_MODUL_MODE_CH1| FGSDK_ARRAY_MODE, // FG function index
    FGSDK_SINE,           // Wave function
    100,                  // Frequency
    0,                    // Phase
    0,                    // Duty cycle
    0,                    // Leading time
    0,                    // Trailing time
    FGSDK_MODUL_AM,       // Modulation type
    100,                  // Modulation scale
};
double pdbParaCh2[9] = {
    FGSDK_DATA_CH2 | FGSDK_ARRAY_MODE, // FG function index
    FGSDK_SINE,           // Wave function
    5500,                 // Frequency
    0,                    // Phase
    0,                    // Duty cycle
    0,                    // Leading time
    0,                    // Trailing time
    1.5,                  // Voltage amplitude
    0.75,                 // Voltage offset
};
```

```
double sTrigParameter[4] =
{
    FGSDK_TRIG_DATA | FGSDK_ARRAY_MODE, // FG function index
    FGSDK_TRIG_TYPE_INT,      //Trigger source
    FGSDK_EDGE_RISING,        //Trigger edge
    100,                      //Internal Trigger source frequency
};

double dbCh2CwMode = FGSDK_CW_MODE_CH2;
double dbSendPtn = FGSDK_SENDPTN;

//Set Ch1 carrier parameters
puDsoSetting(iDevice, DSS_FGCONTROL, pdbParaCh1);
//Set Ch2 carrier parameters
puDsoSetting(iDevice, DSS_FGCONTROL, & pdbParaCh2);
//Set Ch1 to Modulation mode
puDsoSetting(iDevice, DSS_FGCONTROL, & pdbModulCh1);
//Set Ch2 to Continuous Wave mode
puDsoSetting(iDevice, DSS_FGCONTROL, & dbCh2CwMode);
//Set Trigger data
puDsoSetting(iDevice, DSS_FGCONTROL, & sTrigParameter);
//Send FG pattern
puDsoSetting(iDevice, DSS_FGCONTROL, & dbSendPtn);
```

Acute Technology Inc.

http://www.acute.com.tw/

Address：6F-7, #12, Ln. 609, Sec. 5, Chongxin Rd., Sanchong Dist., New Taipei City 24159, Taiwan

Tel：+886-2-2999-3275

Fax：+886-2-2999-3276

E-mail: service@acute.com.tw