SDK - HTMarch.dll Manual
VB 6.0 IDE

Updated: 16-JAN-2017

Note: HTMarch.dll was compiled under VC++6.0.

The following ifdef block is the standard way of creating macros which make exporting from a DLL simpler. All files within this DLL are compiled with the HTMARCH_API symbol defined on the command line. this symbol should not be defined on any project that uses this DLL. This way any other project whose source files include this file see HTMARCH_API functions as being imported from a DLL, wheras this DLL sees symbols defined with this macro as being exported.

```
#ifndef HTMARCH_API
#define HTMARCH_API extern "C" __declspec(dllimport)
#endif
#defineWIN_API __stdcall
```

This unofficial English version of the documentation is a modification of the original by roderick.young@lycos.com . It contains his best guesses at how things actually work. Feedback is welcome.

Note that the 6022BL (Oscilloscope plus Logic Analyzer) also has a library called HTMarch. This document covers the HTMarch for the 6022BE. It is not a simple matter to make one program work for *both* the 6022BE *and* 6022BL. See the BasicScope Design Document for one approach.

Function Introduction

1. HTMARCH_API short WIN_API dsoOpenDevice(unsigned short DeviceIndex)

Return value: Return zero (0) indicates device isn't connected; return one (1) indicates device connected.

Parameter:

DeviceIndex

The first connected device index is 0, and others sequentially numbered.

Remark:

The device whose device index value is judged as DeviceIndex whether connected to PC or not.

Program example:

```
unsigned short nDev = 0;
if(dsoOpenDevice(0) == 1)
        {
        ;// Device connected
        }
Else
        {
        ;// Not detect device
        }
```

2. HTMARCH_API short WIN_API dsoSetVoltDIV(unsigned short DeviceIndex,int nCH,int nVoltDIV);

This sets the input attenuators on the scope so that full scale is the specified value. The scope returns 8-bit samples in the range of 0-255. It is assumed that there are 10 divisions full scale. For example, if nVoltDIV below is set to 5, then a sample value of 0 indicates -5.00 volts, a sample value of 128 indicates 0.00 volts, and a sample value of 255 indicates +5.00 volts. Settings 0, 1, and 2 have 100 mVdiv sensitivity. Settings 3, 4, 5, 6, and 7 have 1V/div sensitivity. For settings 2, 3, 4, and 5, the driver does the necessary scaling to provide samples in the range of 0-255 automatically. For other settings, the user must do the scaling.

Return value: one (1) for setup success and zero (0) for failure.
Parameter:
DeviceIndex
indicates current device index value.
nCH:
Channel index value. 0 stands for CH1, and 1 stands for CH2.
nVoltDIV
Voltage index value. Minimum voltage is 0, and following is the index values for corresponding voltages.
0: 20mV/div
1: 50mV/div
2: 100mV/div
3: 200mV/div
4: 500mV/div
5: 1V/div
6: 2V/div
7: 5V/div
Remark:
The device whose device index value is judged as DeviceIndex whether connected to PC or not.
Program example:
dsoSetVoltDIV(0,0,5); // Set the voltage of CH1 to 1V/div.

3. HTMARCH_API short WIN_API dsoSetTimeDIV(unsigned short DeviceIndex,int nTimeDIV);
Return value: one (0) for setup success and zero (0) for failure.
Parameter:
nDeviceIndex
indicates current device index value.
nTimeDIV
indicates current sampling rate index value, following is the value.
0 ~ 10 : 48MSa/s – recommend 1016 buffer size
11: 16MSa/s – recommend 130048 buffer size
12: 8MSa/s – recommend 130048 buffer size
13: 4MSa/s – recommend 130048 buffer size
14 ~ 24: 1MSa/s – recommend 1047552, 523264, or 130048 buffer size
25: 500KSa/s
26: 200KSa/s
27: 100KSa/s
Remark:
Recommend using the value 10 for a 48 MSa/S rate. The lower numbers might be higher sample rates for compatibility with other models of scope, but are all effectively 48 MSa/S on the 6022BE. dsoSetTimeDIV() must be called to set the sample rate before calling dsoReadHardData(), because the latter function does not actually set the sample rate.

Program example:

4. HTMARCH_API short WIN_API dsoReadHardData(

unsigned short DeviceIndex,
short* pCH1Data,
short* pCH2Data,
unsigned long nReadLen,
short* pCalLevel,
int nCH1VoltDIV,
int nCH2VoltDIV,
short nTrigSweep,
short nTrigSrc,
short nTrigLevel,
short nSlope,
int nTimeDIV,
short nHTrigPos,
unsigned long nDisLen,
unsigned long * nTrigPoint,
short nInsertMode);

Return value: Reading data, return -1 for failure (device disconnected),
Other return values uncertain, see nTrigSweep description below.
return 0 for successful read (trigger found) in normal or single mode
return 1 for no trigger found in auto trigger mode
return 2 in normal or single mode, seems to suggest failed read due to clipping. Returned data will be unscaled, as if in auto mode.
Return 2 for successful read (trigger found) in auto mode

Parameter:

unsigned short DeviceIndex: IN Device index value
short* pCH1Data: OUT CH1 data storage buffer pointer
short* pCH2Data: OUT CH2 data storage buffer pointer
unsigned long nReadLen: IN The length of data to read
short* pCalLevel: IN Proofreading level (reference function dsoGetCalLevel)
int nCH1VoltDIV: IN The voltage of CH1, same as that passed to dsoSetVoltDIV()
int nCH2VoltDIV: IN The voltage of CH2
short nTrigSweep: IN SWPMODE-0: AUTO; 1: Normal; 2: Single
short nTrigSrc: IN Trigger source - 0: CH1; 1: CH2
short nTrigLevel: IN Trigger level (-160 ~ +160 in AUTO, 0 ~ 255 in Normal and Single)
short nSlope: IN Trigger Slope - 0: Rise; 1: Fall
int nTimeDIV: IN Sampling rate, same as that passed to dsoSetTimeDIV()
short nHTrigPos: IN Horizontal trigger position -0 ~ 100 (ignored)
unsigned long nDisLen: IN The length of the display data
unsigned long * nTrigPoint: OUT The index value of returned trigger point
short nInsertMode: IN D-value mode - 0: Step D-value; 1: Line D-value; 2: SinX/X D-value

Remark:

This function reads one buffer of scope data. It does NOT wait for a trigger condition, then return a valid buffer of data. It will grab the specified number of bytes, then always return. In that sense, it always acts in an auto-trigger mode. There is no trigger in the hardware, and no automatic retry if no trigger was found.. If there was no trigger condition present, the calling software must keep calling the function again until a trigger is found. dsoReadHardData() does not seem to locate the trigger position reliably, nor position the buffer with respect to trigger reliably, especially at high sample rates.

For some reason, taking 1012 samples at 48 Msps takes about twice as long as taking 130048 samples at 16 Msps. Possibly the 48 Msps rate has extensive signal processing or interpolation going on.

nTrigSweep theoretically controls the sweep mode, but in practice, seems to do something entirely different. Auto mode seems to return a value of 1 or 2 always. In Auto mode, the returned

data buffers seem to be signed short integers in the range --164 to 163. Also in Auto mode, the data is scaled so that there will be 8 divisions on the screen.. Normal and Single mode seem indistinguishable, each returning 0 for a successful capture, with the data buffer a short integer in the range 0 to 255. The data is scaled so that there are 10 vertical divisions on the display. In Normal and Single mode, if the return value is 2, the data buffer is like that of Auto mode, that is, not normalized to the 0 to 255 range. One theory is that this is the raw, unscaled data, and presented that way because the input was out-of-range and clipped, and the buffer is theoretically should not to be used.
IMPORTANT: Results seem to be erratic unless the proper read length is specified to match the sample rate. Deviate from the recommended values at your own peril. See dsoSetTimeDIV() for recommended read lengths (buffer sizes).

nTrigLevel takes this measurement scale as the returned data, so in Auto mode, should have a value of -160 to +159, and in Normal or Single mode, 0 to 255. This is contrary to the official Hantek SDK documentation at the time of this writing. Also, there appears to be a bug in the Hantek triggering, such that values over +135 in Auto mode lead to no trigger being found. Possibly this also affects highly negative values, too.

nCH1VoltDIV, **and nCH2VoltDIV**. In dsoReadHardData(), these parameters set up the input attenuator. However, dsoSetVoltDIV() sets up additional scaling on the output data. Therefore, dsoSetVoltDIV() must be called at least once every time the volts/div setting changes, in addition to passing the matching values to dosReadHardData().
nTimeDIV must be set appropriately. The value is set to dsoSetTimeDIV() once to set the sample rate, then again in dsoReadHardData() to set the buffer size. If the two values do not match, there may be an access violation in the driver, probably because a buffer overflowed.

nHTrigPos seems to be ignored, probably unimplemented.
nTrigPoint returns the index of the sample on which the trigger occurred, at least in Auto mode. Other modes have not been tested by the original writer of this document. If no trigger was found, nTrigPoint is set to 0.
nDisLen is ignored unless the requested sample rate exceeds the capability of the 6022BE, in which case data is interpolated into the longer buffer. **nInsertMode** is similarly ignored unless needed.

5. HTMARCH_API unsigned short WIN_API dsoGetCalLevel(unsigned short DeviceIndex,short* level,short nLen);

Return value: return zero (0) for success and non-zero for failure.

Parameter:

DeviceIndex
: indicates current device index value.

level
: Proofreading data storage buffer.

nLen
: The length of proofreading data, here=32.

Remark:

Retrieve proofreading data stored in the non-volatile memory of device.

Program example:

```
short nCal[32];
dsoGetCalLevel(0, nCal l,32);
```

6. HTMARCH_API short WIN_API dsoCalibrate(unsigned short nDeviceIndex,int nTimeDIV,int nCH1VoltDIV,int nCH2VoltDIV,short* pCalLevel);

Return value: return zero (0) for success and non-zero for failure.

Parameter:

nDeviceIndex
: indicates current device index value.

nTimeDIV
: Sampling rate

nCH1VoltDIV
: The voltage of CH1

nCH2VoltDIV
: The voltage of CH2

pCalLevel
: Proofreading data memory area

Remark:

It is crucial to have both scope probes shorted to ground when this function is called. This returns an array of proofreading values, which correspond to the zero volt levels on each channel. Generally, these values will be about 128. If they are not, the probes may not be properly grounded. In normal operation, an application will get calibration data from dosGetCalLevel(). nTmeDIV, nCH1VoltDIV, and nCH2VoltDIV seem to have no effect on the results.

7. HTMARCH_API unsigned short dsoSetCalLevel(unsigned short DeviceIndex,short* level,short nLen);

Return value: return zero (0) for success and non-zero for failure.

Parameter:

DeviceIndex
: indicates current device index value.

level
: Proofreading data memory area

nLen
: The length of proofreading data, here is 32.

Remark:

Store calibration data into the non-volatile memory of the device. Typically, dsoSetCalLevel() is called right after calibrating the scope with dsoCalibrate().

Program example:

```
short nLevel[32];
dsoCalibrate(0,11,5,5,nLevel); // When zero reference offset, calibrate it first, then acquire
the calibration data.
dsoSetCalLevel(0, nLevel,32); // Store the calibration date to device.
```