Cleverscope Ltd
Phone +64 9 524 7456
Fax +64 9 524 7457
Email support@cleverscope.com
28 Ranfurly Rd, Epsom
P.O. Box 26-527
Auckland 1003
New Zealand

V1.4
15 October 2008

Cscope Control Driver DLL Description

Summary

The Cscope Control Driver DLL is used by text based languages to communicate with the Cleverscope CS328A acquisition unit. We provide an example 'SimpleScope' application to show use of the driver. The example is available for NI Labview, NI Labwindows, Borland Delphi 5, Borland C++ Builder 6, Microsoft Visual Studio C++ 2005, and Microsoft Visual Studio C# 2005. We have deliberately used older environments, as newer toolsets continue to support and open older version projects. See the document "Cleverscope Simple Scope application.pdf" to see how the SimpleScope application is put together. A Labview application 'Bandpass Response' shows use of the driver for a multi-frame capture application.

The Cscope Control Driver comes as three files:

- Cscope Control Driver.h
  This header file is used by C++ and C# to define the prototypes for the structures and functions in Cscope Control Driver. When using Microsoft VS C#, the header items needs to be converted to managed data structures. The utility "P/Invoke Wizard" can help with this. Similarly a conversion is required for Delphi, and "HeadConv" by Bob Swart can help. For Microsoft VS, you will have to use 'Project/Add Existing Item…" to include the file in the project.

- Cscope Control Driver.dll
  This contains the actual driver. It needs to be linked with the project. See the programming examples to see how the DLL has been linked. For Microsoft VS, you will have to use 'Project/Add Existing Item…" to include the file in the project.

- Cscope Control Driver.lib
  This is the library file, and is required for the C variants. For Delphi C++ builder, you will need to convert the standard library into Borland format. The 'implib.exe' utility is provided for this purpose. The example includes a pre-converted library. You will only need to convert if you use Labview to rebuild the Control Driver. Other environments use the .lib file directly.

Changes

Version	Date	Change
1.0	1 Feb 2005	Initial Cscope Control Driver released
1.4	15 Sep 2008	Sample value format changed from Double to Float (Single), to reduce memory usage. Added Num_Frames value to driver to report the number of frames transferred in a multi-frame capture and transfer. Made small changes to the acquire structure – the order and contents after 'Trigger2Source' has changed. **Important**: The driver now waits up to 40ms for a trigger when using the **Wait for samples** Control Driver Function. After 40ms, the call times-out. The wait blocks the thread, but relinquishes control to the operating system. This maximizes throughput.

Cscope Control Driver.h

//This is the format of the cscope control driver.h file for C or c++

```
#include "extcode.h"
#pragma pack(push)
#pragma pack(1)

#ifdef __cplusplus
extern "C" {
#endif
typedef struct {
      unsigned short AcquireMode;
      unsigned short AcquisitionMode;
      unsigned short Acquirer;
      unsigned short TransferChans;
      double AMaxScale;
      double AMinScale;
      double BMaxScale;
      double BMinScale;
      unsigned short AProbe;
      unsigned short BProbe;
      unsigned short ACoupling;
      unsigned short BCoupling;
      unsigned short ABandwidth;
      unsigned short BBandwidth;
      unsigned long TriggerSource;
      double TriggerAmplitude;
      double ATriggerAmplitude;
      double BTriggerAmplitude;
      unsigned short TriggerFilter;
      LVBoolean TrigSlope;
      double TriggerHoldoff;
      LVBoolean DigPatternRqd;
      unsigned long DigPattern;
      double ExtTrigThreshold;
      double DigInputThreshold;
      double StartTime;
      double StopTime;
      double PreTrigTime;
      unsigned short Port;
      short NumDivisions;
      short NumSeqFrames;
      long NumBuffers;
      double SigGenFreq;
      double SigGenAmp;
      double SigGenOffset;
      unsigned short SigGenWaveform;
      unsigned short SigGenSweep;
      unsigned short SigGenFunc;
      double SigGenFreq2;
      double SigGenPhase;
      unsigned short Trig2Function;
      double MinTriggerPeriod;
      double MaxTriggerPeriod;
      unsigned long TriggerCount;
      LVBoolean Trig2Slope;
      unsigned long Trig2SourceChan;
      double Trig2Level;
      LVBoolean DigPattern2Rqd;
      unsigned long DigPattern2;
      unsigned short Trigger2Source;
      long WaveformAverages;
      long ValueChanged;
      double FreqSpan;
      double FreqRes;
      double Duration;
      double Resolution;
      LVBoolean UnitsAreLinked;
      LVBoolean ExtSampleClock;
      LVBoolean FSpare2;
      LVBoolean FSpare3;
      LVBoolean FSpare4;
      unsigned short SamplerResolution;
      unsigned short IntfSource;
      unsigned short UpdateRate;
```

```
      unsigned short TransferSize;
      double SigGenFreqStep;
      unsigned long TCPAdr;
      unsigned long TCPPort;
      double NSpare3;
      double NSpare4; } TD1;

typedef struct {
      LVBoolean status;
      long code;
      LStrHandle source;
      } TD2;


void __stdcall CscopeControlDriver(
      unsigned short Command, double ReplayStartTime, double ReplayStopTime,
      long SamplesInReplay, long FrameNumber, TD1 *AcquireDefinition,
      LVBoolean *GotSamples, double *T0, double *dT,
      unsigned long *NumSamples, unsigned long *NumFrames,
      float ChanAData[], long ChanAAllocSpace,
      float ChanBData[], long ChanBAllocSpace,
      unsigned short DigitalInputData[], long DigInpAllocSpace,
      TD2 *errorOut);


long __cdecl LVDLLStatus(char *errStr, int errStrLen, void *module);

#ifdef __cplusplus
} // extern "C"
#endif
```

Cscope Driver Functions

Cscope driver provides two functions:

CscopeControlDriver
This function is used to communicate with the acquisition unit, configure it, and retrieve samples.

LCDLL status
This function is used to verify that the DLL loaded properly, and if not, what the error is.

LVBoolean is a U8. 0 means false, 1 means true.

CcsopeControlDriver

This is the main user function. Parameters are:

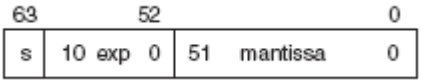
Command

Unsigned 16 bit value.
Values are:

- 0 - **Inititialize**. Call this once to initialise the acquisition system. Further calls are ignored.
- 1 – **Acquire**. Call to acquire data as defined by the Acquire Definition and other parameters.. Caalling acquire automatically updates the acquisition unit with any changed acquire values.
- 2 – **Replay**. Call this to re-decimate the capture buffer, and return new samples, based on the SamplesIn Replay, ReplayStartTime and ReplayStopTime values.
- 3 – **Wait for samples**. Call this to check if a trigger has occurred, and the samples are available. The Value GotSamples is set true when all the samples have been received. The call will wait up to 40ms for a trigger. After 40ms, the call times-out, returning false. The wait blocks the thread, but relinquishes control to the operating system during the wait. This maximizes throughput.
- 4 – **Update.** This call updates acquisition unit values if the acquisition unit is not acquiring, or is waiting for a trigger. Can be used to update the signal generator values for example.
- 5 – **Finish**. Call this to close down the acqusition system
- 8 - **Get Frames**. Gets a multi-frame sequence as one array. The value num_samples is the number of samples in one frame. The value num_frames are the number of frames included in the array. After sending the command, call 'Wait for Samples' until the samples are transferred.

ReplayStartTime

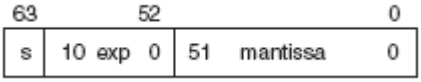
Double.



This value specifies, in seconds, the start time of the samples to be returned in the decimated replay from the sample buffer. If the start time is outside the actual available buffer start and stop times (relative to the trigger), the start time will be clipped to either the beginning or end of the buffer, as necessary.

ReplayStopTime

Double.



This value specifies, in seconds, the stop time (inclusive) of the samples to be returned in the decimated replay from the sample buffer. If the start time is outside the actual available buffer start and stop times (relative to the trigger), the start time will be clipped to either the beginning or end of the buffer, as necessary.

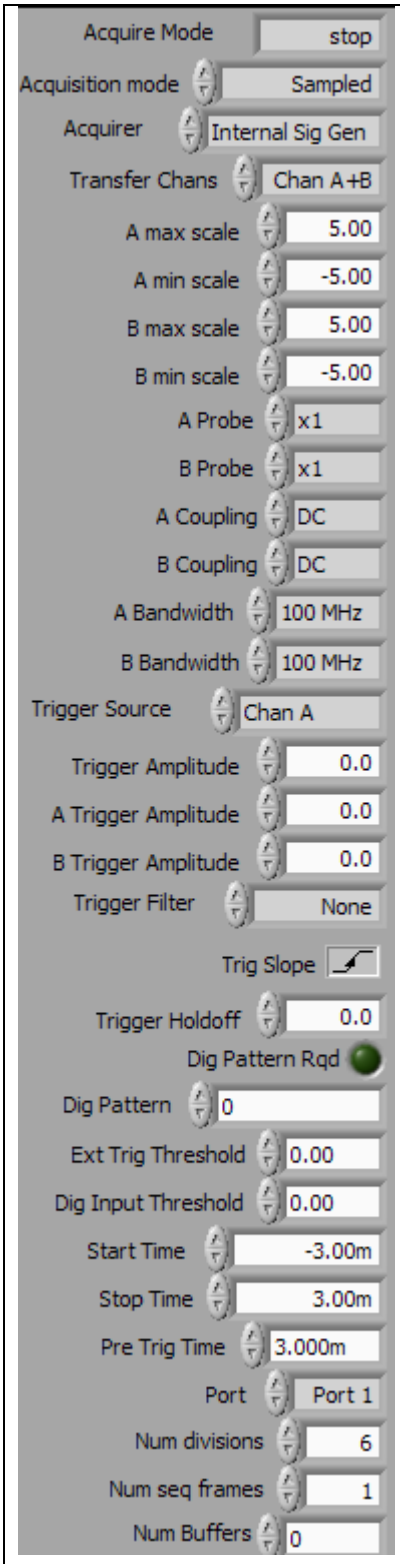
SamplesInReplay

Signed 32 bit number.

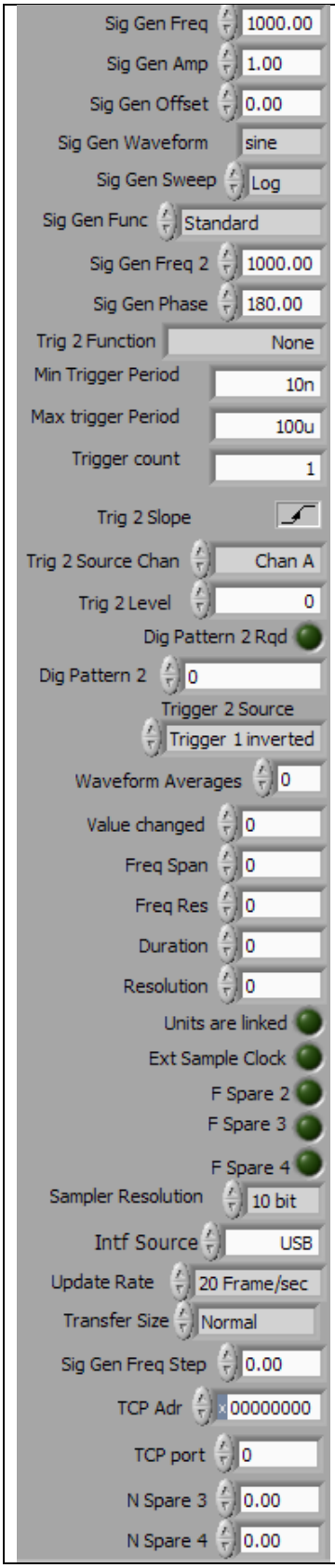
This value specifies the number of samples that will be returned in the decimated replay from the sample buffer. Values may vary from 0 to the size of a frame. If you request more samples than in a frame, the number will be set to the frame size. The maximum size is the acquisition storage size (4 or 8M) divided by 2.

AcquireDefinition

This is TD1, the structure of which is given in the header.



Item	Description	Data Type
Acquire Mode	How to acquire: 0 = Single, 1= automatic, 2 = triggered, 3 = stop	U16
Acquisition Mode	Method of acquisistion: 0 = sampled, 1= Peak captured, 2 = Filtered, 3= Repetitive, 4= Waveform avg If Waveform avg, make sure there are at least waveform avg +1 buffers.	U16
Acquirer	Sets the acquirer to use. Always use 4 = cleverscope	U16
Transfer Chans	Always set to 2 = transfer all channels.	U16
A max scale	Maximum A channel scale value.	Double
A Min scale	Minimum A channel scale value – make lower than max	Double
B max scale	Maximum B channel scale value.	Double
B min scale	Minimum B channel scale value – make lower than max	Double
A probe	A Probe Multiplier 0 = x1, 1 = x 10, 2 = x100, 3 = x1000	U16
B probe	A Probe Multiplier 0 = x1, 1 = x 10, 2 = x100, 3 = x1000	U16
A Coupling	A Coupling, 0 = AC, 1= DC	U16
B Coupling	B Coupling, 0 = AC, 1= DC	U16
A Bandwidth	A Bandwidth, 0 = 25MHz, 1 = 100 MHz	U16
B Bandwidth	B Bandwidth, 0 = 25MHz, 1 = 100 MHz	U16
Trigger Source	Sets trigger source. 0 = A chan, 1 = B chan, 2 = Ext Trigger, 3 = Dig Input, 4 = Rear Input	U16
Trigger Amplitude	Level at which to trigger	Double
A Trigger Amplitude	Not used in driver.	Double
B Trigger Amplitude	Not used in driver.	Double
Trigger Filter	Sets filter on trigger. 0 = None, 1 = Low Pass, 2 = Hi Pass, 3 = noise (2 divisions of hysteresis)	U16
Trig Slope	Sets the trigger slope. 0 = rising, 1 = falling	U8
Trigger Holdoff	Not used in driver.	Double
Dig Pattern Rqd	Sets if the digital pattern qualifies the analog trigger. 0 = not required. 1= required.	U8
Dig Pattern	Sets the digital pattern for digital input triggering. Byte 0 = Select mask, 1= input is used. Byte 1 = Pattern required before trigger Byte 2 = Pattern required to trigger Byte 3 not used. Bit 0 is input 1 .. Bit 7 is input 8	U32
Ext Trig Threshold	Sets the amplitude of the external trigger input, -6..+18V	Double
Dig Inp Threshold	Sets the amplitude of the digital input threshold, 0 .. 10V	Double
Start Time	Sets the start time relative to the trigger, at which acquisition will begin. If positive delayed triggering is used.	Double
Stop Time	Sets the stop time relative to the trigger. Range is –22 .. + 22 seconds. Resolution is 10 ns.	Double
Pre Trig Time	Not used in driver.	Double
Port	Not used in driver.	U16
Num divisions	Set to 10.	I16
Num seq frames	Sets the number of frames captured sequentially. If not waveform avg method of capture set to 1. If waveform avg capture, set to the number of averages used, 4,16,64,128. If capturing sequential frames, set to number of frames to capture.	I16
Num Buffers	Sets the number of buffers allocated for frame capture. Must be at least num waveform averages + 1.	I32



Sig Gen Freq	Set the signal generator frequency in Hz. Range is 0.003..10e6 Hz.	Double
Sig Gen Amp	Amplitude of signal generator output. Range is 0..8V	Double
Sig Gen Offset	Offset of signal generator output. Range is –5..+5V	Double
Sig Gen Waveform	Sets the signal generator waveform. 0 = sine, 1= triangle, 2 = square, 3 = DC, 4 = 0V.	U16
Sig Gen Sweep	Not used in driver	U16
Sig Gen Func	0 means normal sig gen use, 1 means step the sig gen upwards by Sig Gen Freq Step automatically following a trigger.	U16
Sig Gen Freq 2	Not used in driver.	Double
Sig Gen Phase	Not used in driver.	Double
Trig 2 Function	Sets the use of Trigger 2. 0 = Not used, 1 = T1~2 < min, 2 = min<= T1~2 <= max, T1~2 > max, 3 = Count T1, 4 = Wait for T1, then count T2. T1~2 = time duration from trigger 1 to trigger 2.	U16
Min Trigger Period	Sets the min period. 0..22 secs, resolution is 10 ns.	Double
Max Trigger Period	Sets the max period. 0..22 secs, resolution is 10 ns.	Double
Trigger Count	Sets the number of counts for counting. 0..4,294,967,295	U32
Trig 2 slope	Sets the slope for trigger 2. 0 = rising, 1 = falling	U8
Trig 2 Source han	Sets the trigger 2 source channel. 0 = A chan, 1 = B chan, 2 = Ext Trigger, 3 = Dig Input, 4 = Rear Input	U16
Trig 2 Level	Sets the trigger 2 threshold level.	Double
Dig Pattern 2 Rqd	Sets if Trigger 2 is qualified by the pattern.	U8
Dig Pattern 2	Defines the trigger 2 digital pattern.	U32
Trigger 2 Source	Defines the trigger 2 source – 0 = Trigger 1 inverted, 1= Use the Trigger 2 definition	U16
Waveform Averages	Sets how many waveforms to average if acquisition mode = waveform avg. Values are 0 = 4, 1 = 16, 2 = 64, 3 = 128.	I32
Value Changed	Change this value to cause the driver to check for changes in all the values in this data structure. If not changed, data structure values will not update.	I32
Freq Span	Not used in driver	Double
Freq Res	Not used in driver	Double
Duration	Not used in driver	Double
Resolution	Not used in driver	Double
Units are linked	0 means not linked, 1 means linked, and Link port is active	U8
Ext Sample Clock	0 means use internal 100 MHz sample clock. 1 means use external sample clock. Clock must be a sine or square wave, with 45-55% duty cycle, amplitude 0.3V – 3V p-p, biased to 0V or CMOS logic levels. The external clock range currently supported is 10 – 49 MHz.	U8
Fspare 2	Reserved for future use	U8
Fspare 3	Reserved for future use	U8
Fspare 4	Reserved for future use	U8
Sampler Resolution	Sets the sampler resolution to be used, 0 = 10 bits, 1 = 12 bits, 2 = 14 bits. Will clip to maximum resolution available.	U16
IntfSource	Source for connections – 0 = USB, 1 = Ethernet	U16
Update Rate	Not used in driver	U16
Transfer Size	Use 0 to transfer one frame. Use 6 to transfer all the frames in a sequential capture as one array. See num frames value in next section.	
Sig Gen Freq Step	Frequency increment used when acquisition unit automatically steps the signal generator frequency following a trigger, if Sig gen Func = 1.	Double
TCPAdr	TCP address of acquisition unit. Format is bb.bb.bb.bb	U32
TCPPort	TCP port used for acquisition unit.	U32
NSpare3	Reserved for future use	Double
NSpare4	Reserved for future use	Double

GotSamples

Returned value – pointer at U8
Returns 0 if samples are not yet all received. 1 = received the values.

T0

Returned Value – pointer at double.
Returns the start time of the waveform being replayed relative to the trigger, which is time 0, in seconds.

dt

Returned Value – pointer at double.
Returns the interval between successive samples, in seconds.

NumSamples

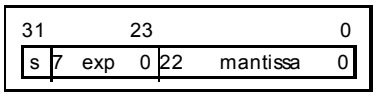
Returned Value – pointer at U32.
Returns the number of samples in the sample array.

NumFrames

Returned Value – pointer at U32.
Returns the number of frames that the sample array is segmented into – only used when returning all the frames in a sequential capture in one transfer. As an example, assuming 2000 samples per frame, and 100 frames sequentially captured, one data array of 200,000 samples will be returned, being composed of 100 segments of 2000 samples.

ChanAData[]

Returned value – pointer to Array of Single (Float). Channel A values.
Values are stored as:

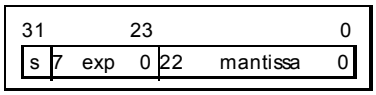


ChanAAllocSpace

Input value – I32
Used to declare to the DLL how much space has been allocated to the Chan A Data array. The data array will be clipped if insufficient space.

ChanBData[]

Returned value – pointer to Array of Single (Float). Channel B values.
Values are stored as:



ChanBAllocSpace

Input value – I32
Used to declare to the DLL how much space has been allocated to the Chan B Data array. The data array will be clipped if insufficient space.

DigitalInputData

Returned value – Array of U16. Digital Input values.

Each U16 contains the bit values corresponding as follows:
In 1 = Bit 0.. In8 = Bit 7

DigInpAllocSpace

Input value – I32
Used to declare to the DLL how much space has been allocated to the DigitalInputs Data array. The data array will be clipped if insufficient space.

ErrorOut

Defines any errors using the TD2 data structure.

Extcode.h

The Extcode.h header files defines the following:

```
typedef uInt8 LVBoolean;
#define LVBooleanTrue           ((LVBoolean)1)
#define LVBooleanFalse   ((LVBoolean)0)
#define LVTRUE                  LVBooleanTrue                   /* for CIN users */
#define LVFALSE                         LVBooleanFalse

typedef struct {
        int32     cnt;          /* number of bytes that follow */
        uChar     str[1];       /* cnt bytes */
        } LStr, *LStrPtr, **LStrHandle;
```

Using the DLL

To use the DLL carry out the following steps:

1. Allocate memory for the data arrays.
2. Call the DLL with the **Inititialize (**0) command.
3. Setup the Acquire Definition, and call using the **Acquire** (1) command. The Acquire call automatically updates the acquisition unit to the contents of the acquire structure.
4. Use a timed loop that achieves the desired throughput. Maximum thoughput is typically 20 updates per second (50msec intervals). Call the **Wait for samples** (3) command until GotSamples = 1. The data will now be in the data array. Note that the call may delay up to 40msec for a trigger event to occur. During the wait, the active thread hibernates and returns control to the operating system.
5. If you want to replay another portion of the acquired data, use the **Replay** (2) command followed by **Wait for samples** (3) to check for the samples being transported. Any returned signal subset will be clipped to the start and end times specified when the acquire was made.
6. If you want to update the acquisition unit, without making an acquisition, or while waiting for a trigger, use the **Update** (4) command. You can control the signal generator this way.
7. Finally finish by calling the **Finish** (5) command.

Notes:
1. The DLL is called using STD CALL calling conventions.
2. The DLL will automatically take the next lowest available USB serial number if more than one CS328 or CS328A are connected.
3. **ErrorOut** may be used to check for errors.
4. **LCDLL status** may be used to verify that the DLL has loaded correctly before use**.**

Example C code

This example makes use of the "cscope interface.h" provided with the SimpleScope example. Here is the source:

```
#define     max_samples         16384         //This can be any number up to 4194304
#define     t_divisions         10            //This is the number of time divisions across a graph
#define     v_divisions         8             //This is the number of volt divisions up a graph

//Trigger actions:
#define     acq_single          0             //means capture with a trigger
#define     acq_auto            1             //means capture auto - with a trigger if there is one.
#define     acq_stop            3             //means stop capturing.

/* = Sample Oscilloscope Include File ================================== */
int     scope_init (void);
int     scope_close (void);
int     scope_config (double a_div, double b_div, double t_div, int number_of_points, double freq,
double sigvolts, double trigvolts, int trig_chan, unsigned short trigger_action);
int     scope_acquire (void);
int     scope_read_waveform (float a_waveform[max_samples], float b_waveform[max_samples],
        int *num_samples,  double *delta_t, double *t_zero);
int     check_for_samples(void);
```

//Trigger_action defines how we want to trigger - with a trigger, auto, or not trigger.

Here is the c code

```
#include "Cscope Control Driver.h"
 #include "cscope interface.h"


#define     c_init              0
#define     c_acquire           1
#define     c_replay            2
#define     c_check             3
#define     c_update            4
#define     c_finish            5

int scope_err;

float                           a_samples[max_samples];      //contains the a channel samples
float                           b_samples[max_samples];      //contains the b channel samples
unsigned short                  dig_samples[max_samples];    //contains teh digotal channel values

long                            samples_required;            //the number of samples to capture and
display
long                            samples_returned;            //actual number of samples returned
double                          dt,t0;                       //holds time increment and start value
static                          TD1             acquire;     //holds the acquire definition
static                          TD2             error;       //holds the error value
static                          LVBoolean       got_samples; //set to 1 when we have samples

//************************************************************
int call_cscope_control_driver(unsigned short command)
//use this routine to call the control driver with a particular command
{
CscopeControlDriver(command, acquire.StartTime, acquire.StopTime, samples_required,frame_number,
&acquire, &got_samples, &t0, &dt, &samples_returned, &frames_returned,
a_samples, sizeof(a_samples),  b_samples, sizeof(b_samples),dig_samples, sizeof(dig_samples),
&error);

scope_err = error.code;
return scope_err;
}
//************************************************************
int         scope_init (void)
//Assumes the scope is connected and opens it.
//Sets up the default values

{
acquire.AcquireMode = 3;                        //don't capture right now
acquire.AcquisitionMode = 1;                    //peak captured
acquire.Acquirer =  4;                          //cleverscope is the acquirer
acquire.TransferChans = 2;                      //transfer both channels
```

```
acquire.AMaxScale = 2;                          // Volts range = +/-2
acquire.AMinScale = -2;
acquire.BMaxScale = 2;
acquire.BMinScale = -2;
acquire.AProbe = 0;                             //x1
acquire.BProbe = 0;                             //x1
acquire.ACoupling = 1;                          //DC
acquire.BCoupling = 1;                          //DC
acquire.ABandwidth = 1;                         //100 MHz
acquire.BBandwidth = 1;                         //100 MHz
acquire.TriggerSource = 0;                      //A Chan trigger
acquire.TriggerAmplitude = 0;                   //Trigger at zero volts
acquire.ATriggerAmplitude = 0;
acquire.BTriggerAmplitude = 0;
acquire.TriggerFilter = 0;                      //No trigger filter
acquire.TrigSlope = 0;                          //rising
acquire.TriggerHoldoff = 0;
acquire.DigPatternRqd = 0;                      //not used
acquire.DigPattern =  0;                        //not used
acquire.ExtTrigThreshold = 0;
acquire.DigInputThreshold = 2;
acquire.StartTime =  -0.005;   //-5 msecs
acquire.StopTime =   0.005;                     //5 msecs
acquire.PreTrigTime = 0.005;
acquire.Port = 0;
acquire.NumDivisions = 10;
acquire.NumSeqFrames = 1;
acquire.NumBuffers = 2;
acquire.SigGenFreq = 1000;                      //1kHz output
acquire.SigGenAmp = 1;                          //1V amplitude
acquire.SigGenOffset = 0;
acquire.SigGenWaveform = 0;                     //sine
acquire.SigGenSweep = 0;
acquire.SigGenFunc = 0;
acquire.SigGenFreq2 = 0;
acquire.SigGenPhase = 0;
acquire.Trig2Function = 0;                      //not used
acquire.MinTriggerPeriod = 0.0000001;
acquire.MaxTriggerPeriod = 1;
acquire.TriggerCount = 1;
acquire.Trig2Slope = 0;
acquire.Trig2SourceChan = 0;
acquire.Trig2Level = 0;
acquire.DigPattern2Rqd = 0;
acquire.DigPattern2 = 0;
acquire.Trigger2Source = 0;
acquire.WaveformAverages = 1;
acquire.ValueChanged = 1;
acquire.SamplerResolution = 0; //0 = 10 bit (1 = 12 bit, 2 = 14 bit).
samples_required = 1000;
return call_cscope_control_driver(c_init);

}

//*****************************************************
void update_values(double a_div, double b_div, double t_div, int number_of_points,
            double freq, double sigvolts, double trigvolts, int trig_chan, unsigned short
            trigger_action)
//updates the acquire variable only. Trigger_action defines single, auto or stop actions.
{
acquire.AMaxScale = v_divisions * a_div / 2;
acquire.AMinScale = - acquire.AMaxScale;
acquire.BMaxScale = v_divisions * b_div / 2;
acquire.BMinScale = - acquire.BMaxScale;
acquire.StopTime = t_divisions * t_div /2;
acquire.StartTime = - acquire.StopTime;
acquire.SigGenFreq = freq;
acquire.SigGenAmp = sigvolts;
acquire.TriggerAmplitude = trigvolts;
acquire.TriggerSource = trig_chan;
samples_required = number_of_points;
acquire.AcquireMode = trigger_action;
acquire.ValueChanged++;
}

//*****************************************************
int scope_config (void)
```

```
//Configures major values for the acquisition unit.
{
return call_cscope_control_driver(c_update);
}

//*************************************************************************

int         scope_acquire (void)
//start an acquisition

{
return call_cscope_control_driver(c_acquire);
}

//*************************************************************************
int scope_read_waveform (float a_waveform[max_samples], float b_waveform[max_samples], int
*num_samples, double *delta_t, double *t_zero)
//returns the last read waveform for the given channel
{
int i;

for (i=0; i<samples_returned; i++)
            {
            a_waveform[i] = a_samples[i];
            }
for (i=0; i<samples_returned; i++)
            {
            b_waveform[i] = b_samples[i];
            }

*delta_t = dt;
*t_zero = t0;
*num_samples = samples_returned;
return scope_err;
}

//*************************************************************************

int check_for_samples(void)
//checks to see if samples have been returned. If so returns 1, else 0

{
call_cscope_control_driver(c_check);
return got_samples;
}

//***********************************************
int scope_close (void)
//closes the scope
{
return call_cscope_control_driver(c_finish);
}

//*************************************************************************

To use this system:
    1.  call scope_init to start the run-time background system working.
    2.  Setup the acquire variable.
    3.  call scope_acquire to start looking for a trigger.
    4.  call check_for_samples to check if samples ready. This command waits up to 40msec for a
        trigger. If true call:
    5.  call scope_read_waveform to get the values. They are in single real format.
    6.  Repeat 2-5 until done.
    7.  If you wish to stop sampling call scope_close.
```