plug & play instruments
oscilloscopes

Cleverscope Ltd
Phone +64 9 524 7456
Mob +64 21 1777 367
Email support@cleverscope.com
21A Lancing Rd, Sandringham
P.O. Box 26-527
Auckland 1025
New Zealand

V2.8
26 Apr 2018

Cscope Control Driver DLL Description v2.8

1 Summary

The Cscope Control Driver DLL is used by text based languages to communicate with the Cleverscope CS328A acquisition unit. The DLL can be used in both x32 and x64 bit environments.

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 Express C++ 2008, C# 2008, and Visual Basic 2008. We have deliberately used older environments, as newer toolsets continue to open older version projects. See section 6, which describes how the SimpleScope application is put together.

The Labview based document **Cscope Control Driver vi description** is very useful as it shows step by step what inputs and output are needed for several capture scenarios.

The VS2008 C++ version of SimpleScope also includes the ability to capture from two connected Cleverscope Acquisition Units (CAU) - either using a link cable, or as independent units, and with internal sample clock, or external sample clock. See the Simple Scope description in section 6. The VS2008 C++ version also includes the updated v2.8 function calls.

V2.8 adds further function results to CscopeFunction and adds the entirely new CscopeSpectrum to return spectrum information. See the procedure definitions for more information.

2 Important Information

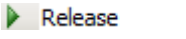
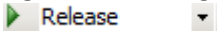
1. To use the application you need to have these components installed:
   - The National Instruments LV7.1 Run Time library. This happens automatically when you install Cleverscope, so make sure Cleverscope has been installed on the target machine.
   - NI-Visa, and the USB port linked to the driver. If you did a full Cleverscope install (rather than just the demo), NI-Visa and the USB links will have been installed.
   - If you are using Visual Studio, you will need .Net Version 3.5 Runtime installed. You can download this from the Microsoft website (http://download.microsoft.com/download/2/0/e/20e90413-712f-438c-988e-fdaa79a8ac3d/dotnetfx35.exe).

   An full environment installer is available on the Resources page of the Cleverscope website, as 'Control Driver Installer xxxx.zip'. See the installation pdf inside the zip for instructions. This installer can be used where the Cleverscope application has **not** been installed.

2. The DLL uses the STD CALL method of parameter passing. Ensure your environment is setup this way.

3. For Visual Studio, you will get a Managed Design Assistant (MDA) 'Loader Lock' error if you have the MDA loader lock enabled. The NI Labview Run Time library uses managed code, and will cause an error if the MDA loader lock is enabled. NI say this will not cause a problem. To avoid errors, turn off the Loader Lock - Find the menu item Debug/Exceptions.
   In the Exceptions dialog, expand Managed Debugging Assistants and find LoaderLock. Now uncheck the check box next to LoaderLock.

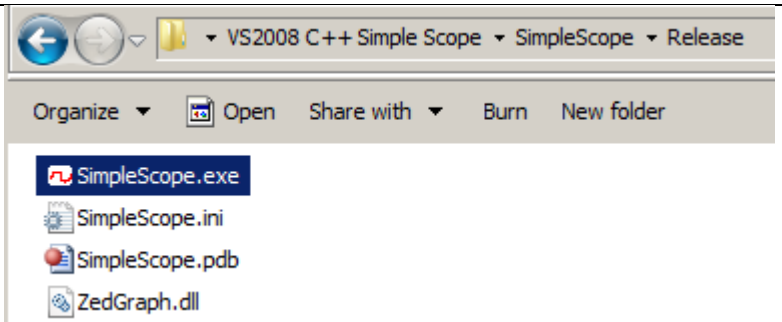
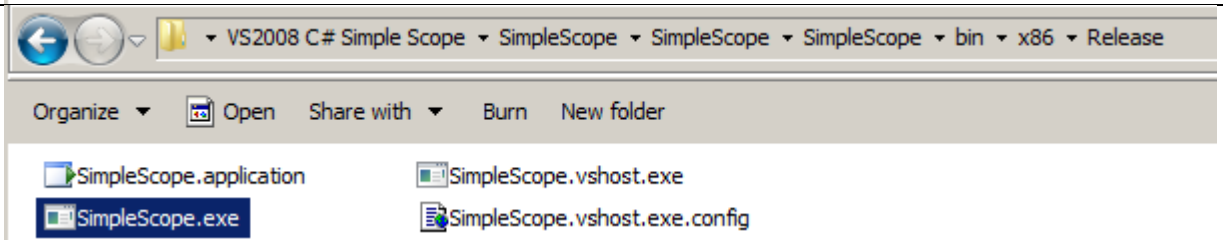
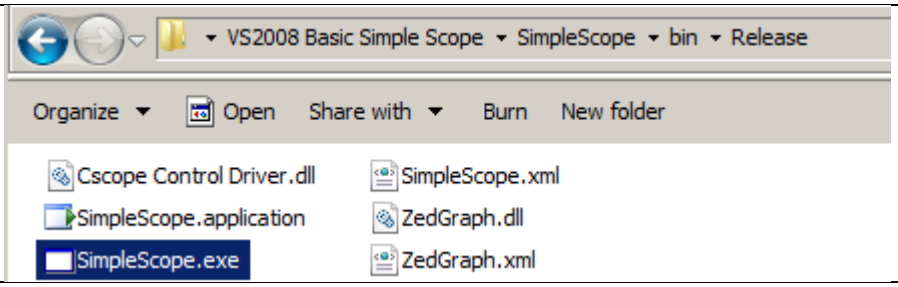
4. You will have to put copies of the two files 'Cscope Control Driver.dll' and 'FTD2XX.dll' in Windows/System32 and Windows/SysWOW64 (for 64 bit systems) as the most general case, or put them in the same directory as your development files, and in the directory with the resulting .exe. Only one copy of 'Cscope Control Driver.dll is included in the distribution, at the root of the distribution zip. If you use the installer, the dll's are automatically copied to the system32 directory. You **will** have to copy the files to the SysWOW64 directory yourself. You can use the CopyDLL.bat batch file to transfer required files to both directories.

5. The CscopeControlDriver DLL is a 32 bit object. To use it, you will have to target any applications you build to be x32 based.
   a. In Visual C++ select Win32 as the target () on the tool bar, or use the Configuration manager (Build/Configuration Manager..) to do this.
   b. In Visual C# select the x86 target ().
   c. In Visual Basic Express, the default is AnyPC, which works fine. The default is located in the SimpleScope\SimpleScope.vbproj file, which is an XML file. The XML file contains this line: *<Platform Condition=" '$(Platform)' == '' ">AnyCPU</Platform>*

   Cleverscope has verified operation of SimpleScope in both x32 and x64 bit versions of Windows 7.

6. The Visual Studio applications use Zedgraph, a toolset for plotting graphs. This is an Open Source toolset available here: http://sourceforge.net/projects/zedgraph/. Click the Download button to obtain component. It is included in the Cleverscope DLL distribution. You can put the zedgraph.dll in your windows\system32 directory (and SysWow64 if you are running a x64 bit system), or in the application folder.

7. Using the Microsoft Visual Studio 2008 Express environments, with the project in a 'SimpleScope' folder, you will find operating executables in the following locations:

C++	
C#	
Basic	

3 Control Driver Versions

Doc 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.
2.0	14 Feb 09	Major change to underlying driver. It now supports multiple connected Cleverscope Acquisition Units simultaneously (up to 8 standard, email for more), and includes much improved error display and recovery. In addition the new driver supports both USB and Ethernet. The 40 ms wait in the 'wait for samples' call has been removed – it is now up to the calling application to wait after starting an acquire. The SimpleScope software has been restructured to show how to call the driver for maximum throughput with minimum CPU loading.
2.1	10 Nov 09	Driver now supports 32 connected units, calibration tables for old and new firmware versions. Improved documentation. Included information on new Function command. Update to Bandwidth values.
2.2	7 May 10	Added the CscopeFunction procedure. This procedure is used to make function calls that were previously made using variants on the CscopeControlDriver FrameNumber and NumFrames values. CscopeFunction now provides a Double out value, and an array of bytes into and out of the function to provide transfer of structures. The existing Function calls to CscopeControlDriver still work to maintain backward compatibility, but all new function calls will use CscopeFunction. The CscopeFunction will be used to provide increased utility of the rear link port. This iteration can be used to send and receive a serial data stream. In the TD1 acquire structure, `UnitsAreLinked` has been replaced with `LinkPort`. The extra parameters `FunctionResult, LinkStart, LinkTimebase, LinkTimer, LinkSetup` have been added – see the descriptions below.
2.4	1 Mar 11	Added Calibrate function to the Cscope Function facility. Calibration supported for Once yearly, Standard, and Signal generator.
2.5	16 Jun 12	Improvements to use of multiple connected Acquisition Units
2.6	18 Mar 13	Improved accuracy of number of samples to be transferred. Improved firmware (6463 required) for capture co-ordination. Removed bug in non-rentrant procedure which caused occasional sampling stoppage.
2.7	15 Mar 14	Supports Internal, Constant External and Variable External sampling clocks. Requires firmware 6467.
2.8	26 Apr 18	V2.8 adds further function results to CscopeFunction and adds the entirely new CscopeSpectrum to return spectrum information. See the procedure definitions for more information.

4 Cscope Control Driver Files

The Cscope Control Driver comes as four files:

- Cscope Control Driver.h
  This header file is used by C++ and C# to define the prototypes for the structures and procedures 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.

- FTD2XX.DLL
  This dll is used by Cscope Control Driver.dll to communicate with the CS328 (Classic). It is required.

5 Cscope Control Driver

This is the content of the cscope control driver.h file for C or C++. This module is required for interface to the Cleverscope DLL. The complete environment for defining a Cleverscope Acquisition Unit (CAU) setup is defined by the **acquiredefn** structure. When making changes to the CAU state, change the parameters that need updating and increment the *ValueChanged* parameter. Three procedures - CscopeControlDriver , CscopeFunction, and CscopeSpectrum - are used to access and return values from any connected CAU's. Up to 32 CAU's can be controlled by CscopeControlDriver. See the following sections for more information.

Check out SimpleScope in Section 6 to see how we have implemented a simple oscilloscope.

5.1 Cscope Control Driver.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 SampleInterval;
      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;
      unsigned char LinkPort;
      unsigned char 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;
      unsigned long CAUSerNumHi;
      unsigned long CAUSerNumLo;
      double FunctionNumber;
      double FunctionParameter;
      double FunctionResult;
      unsigned long LinkStart;
      unsigned long LinkTimebase;
      unsigned long LinkTimer;
      unsigned long LinkSetup;
      double Spare1;
      double Spare2;
      double Spare3;
      double Spare4;
      } TD1;

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


void __stdcall CscopeControlDriver(long AcquisitionUnit,
      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, unsigned short *CAUStatus, TD2 *errorOut);

void __CscopeFunction(long AcquisitionUnit, unsigned short FunctionCommand, double Parameter,
      unsigned char LinkDataSend[], long LinkDataSendAllocSpace, double *FunctionResult, double *ResultA,
      double *ResultB, unsigned char LinkDataReceived[], long LinkDataReceivedAllocSpace, TD1 *errorOut)
```

```
void __CscopeSpectrum(long AcquisitionUnit, unsigned short SpectrumType, unsigned short FFTWindow,
      LVBoolean *dBOn, LVBoolean *DegreesOn, LVBoolean *UnwrapPhase, double *dF, long *NumFreqBins,
      double ChanAAmpGain[], long ChanAAmpGainAllocSpace, double ChanBAmpPhase[],
      long ChanBAmpPhaseAllocSpace, double ChanAImPhase[], long ChanAImPhaseAllocSpace,
      double ChanBImPhase[], long ChanBImPhaseAllocSpace, TD1 *errorOut)

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

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

#pragma pack(pop)
```

5.2 Cscope Driver Procedures

Cscope driver provides four procedures:

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

- **CscopeFunction**
  This procedure is used to exercise system level functions, derive mathematical values based on the current signal, and gain access to calibration and the link port.

- **CscopeSpectrum**
  This procedure returns the spectrum (in four formats) of the currently acquired or replayed signal.

- **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.

5.3 Using CscopeControlDriver

This is the main user procedure. Parameters are:

```
void __stdcal  CscopeControlDriver(long AcquisitionUnit,
     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, unsigned short *CAUStatus, TD2 *errorOut);
```

To use the CscopeControlDriver carry out these steps:

1. Set the Acquisition Unit number (0..31). Call the DLL with the **Inititialize (**0) command. You will need to **Inititialize** each Cleverscope slot used.
2. Use the **CAU status** (6) command until the status is 'Open'.
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 *AcquireDefinition* structure.
4. Use a timed loop that achieves the desired throughput. Maximum throughput is typically 40 updates per second (25 msec intervals). If the sequence
   cmd **Acquire**– repeat {wait 25ms then cmd **Wait for samples**} until *GotSamples* = 1
   is used, maximum throughput will be reached. Call the **Wait for samples** (3) command until *GotSamples* = 1. The data will now be in the data array. The driver uses a runtime that operates in the background.
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. Close this Acquisitions Unit by using the **Close** (5) command. Following the close command the CAU Status will return 'Runtime Closed'.
8. After having closed all other Acquisition Unit slots used, finish by calling the **Finish** (9) command. The **Finish** command removes the runtime from memory.

Notes:

1. The driver will automatically take the next lowest available USB port if more than one CS328 or CS328A are connected, and IntfSource = 0, or USB. The port/CAU binding is repeatable through PC restarts.
2. The CAU Status value may be used to check if the Cleverscope is available and turned on. If after an Init, the CAU status does not change to Open, the CAU is not available.
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**.**

5.3.1 Parameters

Acquisition Unit

Input: Signed 32 bit number.
This value sets the Cleverscope Acquisition Unit (CAU) being addressed (0..31). Up to 32 simultaneous CAU's can be controlled at the same time (unless Max Units is increased). Each connected acquisition unit occupies one slot. If a slot is not occupied it uses no additional memory – the CAU runtime support is dynamically loaded when the CAU is opened.

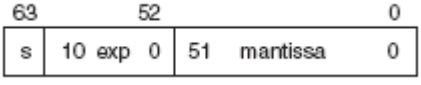
Command

Input: Unsigned 16 bit value.
Values are:

Value	Command	Description
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. Calling 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	**Close**	Call this to close down and remove the runtime for the currently selected acquisition unit. After calling close the CAU status will be returned as 'Runtime Closed'.
6	**Status**	Calling the Control Driver with the status command just returns the current connection status. Status values are 'Runtime Closed', 'Closed', 'Open' and 'Fault' (0..3), and . See below.
7	**Function**	This value is used to retrieve specific values about the connected Cleverscope Acquisition Unit. This command has been superseded by the Cscope Function vi.
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.
9	**Finish**	Call this to close down any remaining unclosed runtime components, and to remove the control driver from memory.

ReplayStartTime

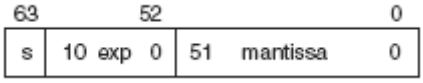
Input: Double



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

Input: 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

Input: 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.

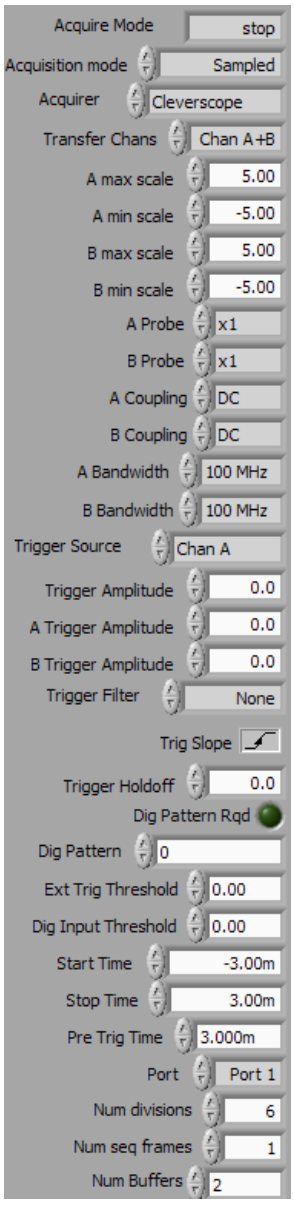
Frame Number

Input: Signed 32 bit number.

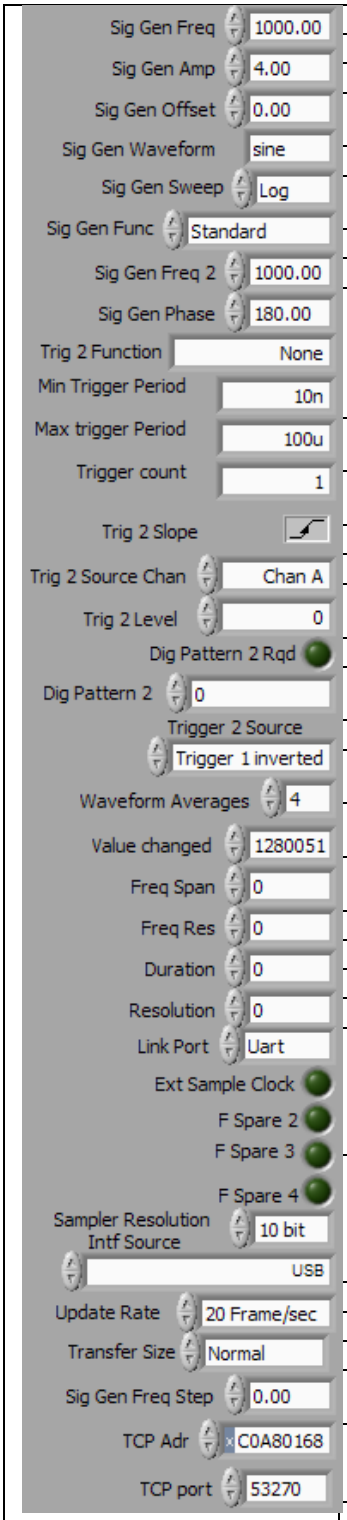
This value specifies which frame to return from a multi frame replay. The default is 0.

AcquireDefinition

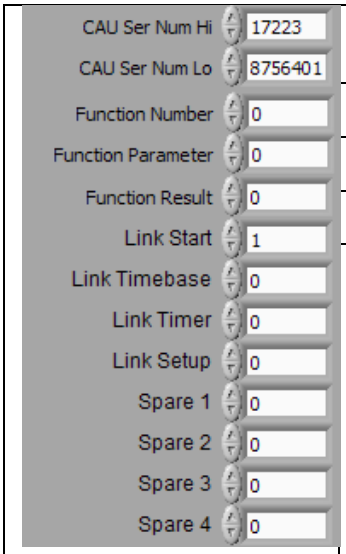
This is TD1, the structure of which is given in the header



Item	Description	Data Type
Acquire Mode	How to acquire: 0 = Single, 1= automatic, 2 = triggered, 3 = stop Make sure this is 3 (stop) when initializing the driver.	U16
Acquisition Mode	Method of acquisition: 0 = sampled, 1= Peak captured, 2 = Filtered, 3= Repetitive, 4= Waveform avg, (for which 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 = x10, 2 = x100, 3 = x1000, 4 = x20, 5 = x50, 6 = x200.	U16
B probe	B Probe Multiplier - Same as for A Probe	U16
A Coupling	A Coupling, 0 = AC, 1= DC	U16
B Coupling	B Coupling, 0 = AC, 1= DC	U16
A Bandwidth	Bit 0 - Global Filter enable, 0 = no filter, 1 = use filter Bit 2:1 - Pre-filter frequency 0 = No filter, 1 = 20 MHz filter Bit 3: - If true, use the moving average (MA) filter Bit 4: - Reserved Bits 7:5 - Filter time constant, in taps: 000 = no filter, 001 = 40ns, 010 = 80ns, 011 = 160ns 100 = 320ns, 101 = 640ns, 110 = 1280ns, tap111 = reserved MA For the moving average only the channel A moving average value is used, and it also used for Channel B	U16
B Bandwidth	Bit 0 - Global Filter enable, 0 = no filter, 1 = use filter Bit 2:1 - Pre-filter frequency 0 = No filter, 1 = 20 MHz filter Bit 3: - If true, use the moving average (MA) filter Bit 4: - Reserved Bits 7:5 from A Bandwidth are also used for B Channel MA filter setting	U16
Trigger Source	Sets trigger source. 0 = A chan, 1 = B chan, 2 = Ext Trigger, 3 = Dig Input, 4 = Link Port	U16
Trigger Amplitude	Level at which to trigger for Channel A or Channel B (as set by Trigger Source). The external or digital input thresholds are set separately below.	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 (<250kHz), 2 = Hi Pass (>500 kHz), 3 = noise. (Test signal 20% FSD sine wave). Normal hysteresis is 2.5%. Noise hysteresis is 7.5%	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. Range is –22 .. + 22 seconds.	Double
Stop Time	Sets the stop time relative to the trigger. Range is –22 .. + 22 seconds. Resolution is 10 ns.	Double
SampleInterval	Time interval between samples, in s units when streaming data to disk.	Double
Port	Not used in driver.	U16
Num divisions	Not used in driver.	I16
Num seq frames	Sets the number of frames captured sequentially. If waveform avg method of capture set to 1. 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	P-P 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, 3 = T1~2 > max, 4 = Count T1, 5 = Count T2, 6 = T1~2 < min , then count T2, 7 = min<= T1~2 <= max then count T2, 8 = T1~2 > max then count T2, 9 = T1 OR 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 = Link Port	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 in acquisition unit if acquisition mode = waveform avg. Values are 1, 4, 16, 64 and 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
Link Port	Defines how the Link port will be used. 0 means debug Uart, 1 means the Link Port outputs trigger, 2 is the port is disabled, 3 means this is a trigger slave cleverscope link port, 4 means this is a trigger master cleverscope link port, 5 means Uart Port, 6 means SPI Port, 7 means I2C port, 8 means used as 3 bit digital port, 9 means use Sig Gen to output arbitrary.	U8
Ext Sample Clock	0 means use internal 100 MHz sample clock. 1 means use external fixed frequency sample clock, 2 means external variable frequency sample clock (no stability check made). Clock must be a sine or square wave, with 45-55% duty cycle, amplitude 0.5V – 3V p-p, biased to 0V or CMOS logic levels. The external clock frequency must be in the range 1 - 110 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 – Open specified IP address, 2 = Find first connected cleverscope, 3 = find Cleverscope with given serial number.	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



CAU Ser Num Hi / Lo	Cleverscope Acquisition Unit serial number split into two U32 values. The upper two ascii characters are stored in the lower 16 bits of CAU ser Num Hi. The lower 4 ascii characers are stored in CAU ser Num Lo.	U32 U32
Function Number	Should not be used direct – used by Cscope Function.vi for Function number transfer	Double
Function Parameter	Should not be used direct – used by Cscope Function.vi for Function parameter transfer	
Function Result	Should not be used direct – used by Cscope Function.vi for Function result transfer	
Link Start	Used to define how the message stored in the link buffer is transmitted: Bit 0 = Transmit when link_port data received Bit 1 = transmit at start of sampling Bit 2 = transmit on trigger Bit 3 = transmit on completion of sampling. Bit 4 = transmit on timer Bit 5 = transmit on digital pattern 1 Bit 6 = transmit on digital pattern 2 Bit 7 = transmit on external trigger rising Bit 8 = transmit on external trigger falling Bit 9 = transmit on link input rising Bit 10 = transmit on link input falling Bit 11 = transmit again as soon as previous transmit completes.	
Link Timebase	Clock used to run uart, i2C, spi, or digital outputs. In 1/70Mhz units. = 14.29 ns. eg 9600 baud = 7292 (an error of 0.4 parts in 9600). 115,200 baud = 608, with an error of 70 parts in 115,200.	
Link Timer	timer used for repeated messages, in 10us units. Bit 31 of the timer is enable. 0 = disable timer, 1 = enable, and start timer	
Link Setup	Setup bits for the different ports UArt dependant setup values. Bit 0 = standard(0), inverted (1) output Bits 2:1 - flow control 0 = no flow control, 1 = ASCII Xon/Xoff flow control 2 = Link Input low stops transmitting 3 = Digital Input 8 low stops transmitting SPI dependant setup. Bit 4 = CPHA = 0 means data is latched on the leading edge of CLK, and changes on the trailing edge = 1 means data is latched on the trailing edge of CLK, and changes on the leading edge Bit 5 = CPOL = 0 means the clock is low while idle, = 1 means it is high while idle. Digital Setup: Bit 8 = Sampled/Timed = 0, means digital output at sample rate = 1. means digital output after count time_base values. Sig Gen setup: Bits 14:12 - Sig gen Destination 0 = DC Offset (uses DC offset as low frequency ARB) 1 = DC gain (amplitude modulation) 2 = Frequency Register (frequency modulation) 28 bit 3 = Phase Register (phase modulation). 4 = Select Freq 0 / Freq 1 (1 bit) 5 = Select Phase 0/ Phase 1 (1 bit) Bit 15: = Sampled/Timed = 0, means sig gen output at sample rate . = 1 means sig gen output after count time_base values.	

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

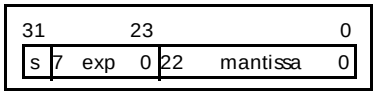
Returned Value – pointer at U32.
Returns the number of samples in the sample array.

NumFrames

Returned Value – pointer at I32.
a. 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.
b. For a Function this is the result of the function call.

ChanAData[]

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

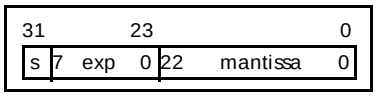


ChanAAllocSpace

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 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.

CAU Status

Output - Unsigned 16 bit integer

This value is returned after every call to the control driver.

Status values are `c_runtime_closed, c_closed, c_open, c_fault, c_fault_closed, c_open_pending`

1. At first power up, with a CAU Unit not initialized, The CAU status will be 'Runtime Closed'.
2. Following a call to Initialize the Control Driver, the CAU status will change to 'Closed.
3. When the connection is made (via USB or Ethernet), the CAU status changes to 'Open Pending'.
4. After the full AcquireDefinition has been transferred to the CAU, the status will change to 'Open'.
5. After you have called Command **Close** the CAU Status will change to c_closed.
6. After you have called Command **Finish**, the CAU Status will change to c_runtime_closed.
7. While closed you may continue to call initialize, with a new interface specification, if needed.
8. If a Fault occurs during a transfer (for example loss of power to the CAU), the Fault status will be returned once, and then the runtime will be automatically be closed, and the status returns 'Runtime Closed'. You will need to Initialize the CAU unit again.

ErrorOut

Returned pointer. Defines any errors using the TD2 data structure.

5.4 CscopeFunction

The **CscopeFunction** procedure is used to recover Cleverscope Acquisition Unit (CAU) information, and to control the Link Port.
NOTE: you will have to set the Link Port to the required destination (eg Link Port = 5 for UART).

```
void __stdcall CscopeFunction(long AcquisitionUnit,
     unsigned short FunctionCommand, double Parameter,
     unsigned char LinkDataSend[], long LinkDataSendAllocSpace,
     double *FunctionResult, double *ResultA, double *ResultB,
     unsigned char LinkDataReceived[], long LinkDataReceivedAllocSpace,
     TD1 *errorOut);
```

5.4.1 Parameters

Acquisition Unit

Input: Signed 32 bit number.
This value sets the Cleverscope Acquisition Unit (CAU) being addressed (0..31). Typically up to 32 simultaneous CAU's can be controlled at the same time. Each connected acquisition unit occupies one slot. If a slot is not occupied it uses no additional memory – the CAU runtime support is dynamically loaded when the CAU is opened.

FunctionCommand

Input: Unsigned 16 bit value.

Defines the command executed:

Value	Function Name	Description
0	Get Serial Num	Returns the serial number of the attached CAU
1	Get Firmware Ver	Returns the version number of the firmware in the CAU
2	Get Driver Ver	Returns the version number of the Cscope Control Driver (currently 2.2, rendered as 22)
3	Get Resolution	Returns the native bit resolution of the attached CAU
4	Get Frame Length	Returns the space allocated to each frame in the CAU, in samples.
5	Get Temperature	Returns the temperature inside the CAU in 0.1 °C units (35.2 deg C is returned as 352).
6	Start Link Send	Starts transmission of active message already stored in the acquisition unit.
7	Send Link Data	Sends the data contained in LinkDataSend to the active message. If bit 0 of LinkStart is set to 1, the message will also be transmitted out the port.
8	Read Link Data	Reads a message received into LinkDataReceived.
9	Active Message	Use the parameter value to set the Active message.
10	Calibrate	Use the parameter to define what sort of calibrate action – see below
11	Calibrate Set ref	Use to set the calibration reference if it is not 2.048V (2.048 is the default, no need to set).
12	Sampling Status	Used to return current sampling status. 0 = idle, 1 = pre-sampling (waiting for trigger) 2 = post-sampling (got trigger)
13	Tf or dT Capture	Defines the capture specification as either Tstart - Tstop and attempt to retrun n values, or Tstart with dt sample spacing for n values
	The following functions evaluate the function over the captured Chan A and Chan B signals, and return the results to Result A and Result B	
14	Sig DC	Signal DC value
15	Sig RMS	Signal RMS value
16	Sig Max	Signal Maximum value
17	Sig Min	Signal Minimum value
18	Sig Pk-Pk	Signal Peak to Peak value
19	Sig Std Dev	Signal Standard Deviation value
20	Sig Freq	Signal Frequency based on highest amplitude frequency component in spectra
21	Sig Amplitude	Amplitude of the Frequency with highest amplitude. It is expressed a voltage if parameter = 0, or dBV if parameter = 1
22	Sig Pulse Frequency	Signal Frequency based on edge to edge period
23	Sig Duty Cycle	Signal Duty Cycle as ratio of positive - negative edge time over positive to positive edge time
24	Sig Pulse Period	Inverse of Sig Pulse Frequency
25	Sig Pulse Length	Signal period of first pulse of opposing edges
26	Sig Rise Time	Rise time of first positive going edge
27	Sig Fall Time	Fall time of first negative going edge
28	Sig 1 Level	Voltage of average high level ignoring over shoot
29	Sig 0 Level	Voltage of average low level ignoring under shoot
30	Sig V Swing	Difference between high level and low level voltages
31	Sig Overshoot	Difference between peak voltage and high level voltage
32	Sig Slew Rate	Slew rate of rising edge in V/us
33	Sig Freq 2nd Harmonic	Frequency equal to 2 x Sig Freq
34	Sig Amp 2nd Harmonic	Amplitude of second harmonic frequency. It is expressed a voltage if parameter = 0, or dBV if parameter = 1
35	Sig Freq 3rd	Frequency equal to 3 x Sig Freq

	Harmonic	
36	Sig Amp 3nd Harmonic	Amplitude of third harmonic frequency. It is expressed a voltage if parameter = 0, or dBV if parameter = 1
37	Sig Sinad	Ratio of (Signal + Noise + Distortion Power) / (Noise + Distortion Power) expressed in dB
38	Sig THD	Total Harmonic Distortion is the Root of the sum of Harmonic amplitudes divided by the fundamental amplitude. It is expressed a percentage if parameter = 0, or dB if parameter = 1
39	Sig HD 2+3	Root of the summed squares of 2nd and 3rd harmonic amplitudes. It is expressed a voltage if parameter = 0, or dBV if parameter = 1

Parameter

Input: Double.
A parameter that may be used by the Function.

LinkDataSend

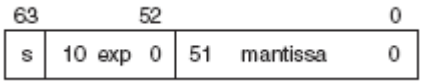
A pointer to a byte array that contains the data that will be sent to the acquisition unit to be stored to the active message. The first two bytes are the length of the send message (Least significant byte followed by Most significant byte). Length bytes follow. A length of 0 is valid.

LinkDataSendAllocSpace

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

FunctionResult

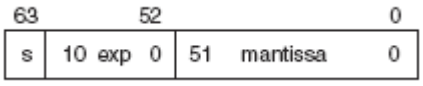
Returned Value – Pointer at Double



This value contains the double format return value from a function.

ResultA

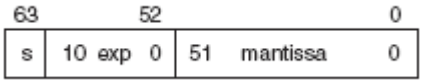
Returned Value – Pointer at Double



This value contains the double format return value for ResultA, which is the result for Channel A for the functions **Sig DC** to **Sig HD 2+3.**

ResultB

Returned Value – Pointer at Double



This value contains the double format return value for ResultB, which is the result for Channel B for the functions **Sig DC** to **Sig HD 2+3.**

LinkDataReceived

A pointer to a byte array that contains the data received from the serial port. The first two bytes are the length of the received message (Least significant byte followed by Most significant byte). Length bytes follow. A length of 0 is valid.

LinkDataReceivedAllocSpace

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

ErrorOut

Returned pointer. Defines any errors using the TD2 data structure.

5.4.2 Doing Calibration

The Calibrate Function (10) uses the **Parameter** to define the calibrate actions. Parameter values are:

Value	Description
0	Idle
1	Start Standard Calibration
2	Start Ground Ofs measurement – channel A
3	Start Ground Offset measurement – channel B
4	Start Baseline measurement
5	Start Ref A measurement
6	Start Ref B measurement
7	Save HW values (used internal to driver – do not use)
8	Start Signal generator Calibration
9	Start External Trigger calibration
10	Start Digital Input Calibration
11	Do Calibrate measurements
12	Set Calibration reference (only use if reference is not 2.048V)

Items 1-10 are used to start an action – call them once.
Item 11 – Do Calibrate is called repeatedly (at >= 50 msec intervals) to carry out the calibration.

The **FunctionResult** returns a real number from 0..100 as a progress indicator. 100 means finished. -1 means an error occurred.

To calibrate the Cleverscope Acquisition unit you need to carry out these steps:

Standard Calibration

Disconnect all cables from Chan A and Chan B.

1. Set **FunctionCommand** to Calibrate.
2. Parameter = Start Calibrate, then run CscopeFunction
3. Parameter = Do Calibrate Measurements, call at >=50msec intervals
4. Wait until FunctionResult is either 100.0 or -1.
   100.0 means complete. Results are automatically saved to the acquisition unit. -1 means calibration failed.

Once yearly calibration

1. Set **FunctionCommand** to Calibrate.
2. Connect gound plugs to both Channels A and B.
3. Parameter = Start Ground Ofs measurement – channel A, then run CscopeFunction
4. Parameter = Do Calibrate Measurements, call at >=50msec intervals
5. Wait until FunctionResult is either 100.0 or -1. 100.0 means ready.
6. Parameter = Start Ground Ofs measurement – channel B, then run CscopeFunction
7. Parameter = Do Calibrate Measurements, call at >=50msec intervals
8. Wait until FunctionResult is either 100.0 or -1. 100.0 means ready.
9. Remove ground plugs
10. Parameter = Start Baseline measurement, then run CscopeFunction
11. Parameter = Do Calibrate Measurements, call at >=50msec intervals
12. Wait until FunctionResult is either 100.0 or -1. 100.0 means ready.
13. Attach 2.048V reference to Channel A.
14. Parameter = Start Ref A measurement – channel A, then run CscopeFunction
15. Parameter = Do Calibrate Measurements, call at >=50msec intervals
16. Wait until FunctionResult is either 100.0 or -1. 100.0 means ready.
17. Attach 2.048V reference to Channel B.

18. Parameter = Start Ref B measurement – channel B, then run CscopeFunction
19. Parameter = Do Calibrate Measurements, call at >=50msec intervals
20. Wait until FunctionResult is either 100.0 or -1.
    100.0 means complete. -1 means calibration failed.

5.4.3 Signal Generator calibration

1. Set **FunctionCommand** to Calibrate.
2. Connect 1x probe or BNC cable between Signal Generator output and Channel A.
3. Parameter = Start Signal generator Calibration, then run CscopeFunction
4. Parameter = Do Calibrate Measurements, call at >=50msec intervals
5. Wait until FunctionResult is either 100.0 or -1.
   100.0 means complete. Results are automatically saved to the acquisition unit.

Cscope Spectrum

```
void CscopeSpectrum(long AcquisitionUnit, unsigned short SpectrumType, unsigned
    short FFTWindow, LVBoolean *dBOn, LVBoolean *DegreesOn,
    LVBoolean *UnwrapPhase, double *dF, long *NumFreqBins,
    double ChanAAmpGain[], long ChanAAmpGainAllocSpace,
    double ChanBAmpPhase[], long ChanBAmpPhaseAllocSpace,
    double ChanAImPhase[], long ChanAImPhaseAllocSpace,
    double ChanBImPhase[], long ChanBImPhaseAllocSpace, TD1 *errorOut)
```

Cscope Spectrum is used to extract the spectrum from the Channel A and B signals received to the PC. By doing a replay on an already captured signal you can change the Frequency resolution, and the Frequency Span of the spectrum captured.

5.4.4 Frequency Span and Resolution

As an example we will capture $T_{capture}$ = 20msec of signal with $N_{capture}$ = 10,000 samples.

The Frequency Resolution, $F_{res} = 1/ T_{capture}$ = 1/20msec = 50 Hz. This is dF
The Frequency Span, F_{span} = Fres x $N_{capture}$ / 2 = 50 x 10,000 / 2 = 250 kHz.

So when capturing to display a spectrum, make sure the capture period is long enough to capture the required frequency resolution, and the number of samples is high enough to achieve the frequency span. After capturing you can use replay to transfer new sample sets. Remember that Cleverscope will always capture as many samples as possible. So if you set the capture time as 20 msecs, but only transfer 1000 samples, there will still be 2M or 4M samples in the Cleverscope buffer. You can do a replay with 100k samples, and get much wider Frequency Span. In addition if you intend to use only a portion of the frequency span, you will get a lower noise floor if you use a wider span, and only select the bandwidth of interest from this. This is because you are minimizing aliasing of signals above the frequency span into the bandwidth of interest.

The VI outputs Num Freq Bins = $N_{capture}/2$. The frequency resolution is dF.

5.4.5 CscopeSpectrum.vi parameters

Parameters are:

5.4.6 AcquisitionUnit

Input: Signed 32 bit number.
This value sets the Cleverscope Acquisition Unit (CAU) being addressed (0..31).

5.4.7 SpectrumType

Input: U16

Value	Function Name	Description
0	RMS	produce the RMS spectrum
1	Power	produce the Power spectrum
2	Gain Phase	Produce the Gain-Phase response of Chan A/Chan B
3	Re-Im	Produce the RMS spectrum but in Real-Imaginary format

5.4.8 FFTWindow

Input: U16
The FFT Window is applied to the signal to minimize the effects of the discontinuous beginning and end of the signal capture record. The window is applied using convolution in the time domain. Each convolution function has a different name.

Value	Window Name
0	None
1	Hanning
2	Hamming
3	Blackman-Harris
4	Exact Blackman
5	Blackman
6	Flat Top
7	4 Term B-Harris
8	7 Term B-Harris
9	Low Sidelobe

FFT Window Characteristics

Window	-3 dB Main Lobe Width (bins)	-6 dB Main Lobe Width (bins)	Maximum Side Lobe Level (dB)	Side Lobe Roll-Off Rate (dB/decade)
Uniform (None)	0.88	1.21	-13	20
Hanning (Hann)	1.44	2.00	-32	60
Hamming	1.30	1.81	-43	20
Blackman-Harris	1.62	2.27	-71	20
Exact Blackman	1.61	2.25	-67	20
Blackman	1.64	2.30	-58	60
Flat Top	2.94	3.56	-44	20

Example FFT Window selection

The choice of window to use depends on the type of signal being examined, and the desired trade-off between Frequency resolution, spectral leakage and amplitude accuracy. The following table gives a first approximation to the best type of window to use:

Signal Content	Best frequency resolution	Best spectral leakage	Best amplitude accuracy
Sine wave or combination of sine waves	Hanning	Exact Blackman	Flat Top
Narrowband random signal (vibration data)	Hanning	Blackman Harris	Blackman Harris
Broadband random (white noise)	Uniform (None)	Blackman	Uniform
Closely spaced sine waves	Uniform, Hamming	Hamming	Blackman
Excitation signals (Hammer blow)	Uniform (None)		
Unknown content	Hanning		

5.4.9 dBOn

Input: Boolean
If dB On is true, voltages are output as dBV, and powers as dBW, except for the Re+IM which are always output as voltages. Otherwise voltages are output as Volts, and powers as Watts (into 1 ohm).

5.4.10 DegreesOn

Input: Boolean
If Degrees On is true, phase is output as degrees, otherwise phase is output as radians.

5.4.11 UnwrapPhase

Input: Boolean
If Unwrap Phase is true, a transition approaching -180 degrees which would normally fold over to +180 degrees continues negative, for example to -190 degrees. The same goes for approaching +180 degrees, which would normally folder over to -180 degrees continues positive, for example to +190 degrees. If false, then fold over is used.

5.4.12 ChanAImPhase

Output - array of double real
Returns the Channel A array of Imaginary or Phase values dependant on the Spectrum Type:

Value	Function Name	Chan A Im/Phase
0	RMS	Phase portion of Chan A RMS spectrum, output in degrees if **Degrees On**, else radians
1	Power	Phase portion of Chan A Power spectrum, output in degrees if **Degrees On**, else radians
2	Gain Phase	Not Used
3	Re-Im	Imaginary portion of Chan A Re+Im spectrum

5.4.13 ChanBImPhase

Output - array of double real
Returns the Channel B array of Imaginary or Phase values dependant on the Spectrum Type:

Value	Function Name	Chan B Im/Phase
0	RMS	Phase portion of Chan B RMS spectrum, output in degrees if **Degrees On**, else radians
1	Power	Phase portion of Chan B Power spectrum, output in degrees if **Degrees On**, else radians
2	Gain Phase	Not Used
3	Re-Im	Imaginary portion of Chan B Re+Im spectrum

5.4.14 dF

Output: Double
The frequency interval between Frequency Bins, equal to the Frequency Resolution

5.4.15 ChanAAmpGain

Output - array of double real
Returns the Channel A array of Amplitude or Gain values dependant on the Spectrum Type:

Value	Function Name	Chan A Amp/Gain
0	RMS	Amplitude portion of Chan A RMS spectrum. Output in Volts or dBV, dependant on **dB On**.
1	Power	Power portion of Chan A Power spectrum. Output in Watts or dBW, dependant on **dB On**, for a load of 1 ohm.
2	Gain Phase	Gain portion of Chan A/Chan B Gain/Phase spectrum. Output in Gain or dB, dependant on **dB On**.
3	Re-Im	Real portion of Chan A Re+Im spectrum, in Volts

5.4.16 ChanBAmpPhase

Output - array of double real
Returns the Channel B array of Amplitude or Phase values dependant on the Spectrum Type:

Value	Function Name	Chan B Amp/Gain
0	RMS	Amplitude portion of Chan B RMS spectrum. Output in Volts or dBV, dependant on **dB On**.
1	Power	Power portion of Chan B Power spectrum. Output in Watts or dBW, dependant on **dB On**, for a load of 1 ohm.
2	Gain Phase	Phase portion of Chan A/Chan B Gain/Phase spectrum. Output in degrees if **Degrees On**, else radians
3	Re-Im	Real portion of Chan B Re+Im spectrum, in Volts

5.4.17 NumFreqBins

Output: Integer 32
The number of frequency bins included in the spectrum output. For a Half Sided spectrum such as output here, the Bin Frequencies run from 0 (DC) to dF * $[(N_{capture}/2)-1]$

6 Simple Scope application example

This application example uses the Microsoft Visual Studio Express 2008 C++ code environment to produce a working executable to make a simple Oscilloscope application with Oscilloscope and Signal generator controls. Further examples using NI Labview, NI Labwindows, Borland Delphi and C++ Builder, and Microsoft Visual Studio Express C# and Visual Basic are also provided. All of the examples use the same structures, variable names and program flow.

The application may be used as an example of interfacing to the DLL using C code.

Note the important information in section 2.

Simplescope can be used to control several connected CAUs, though the results are only displayed for one of them. Simply change the UnitID to connect to another unit.

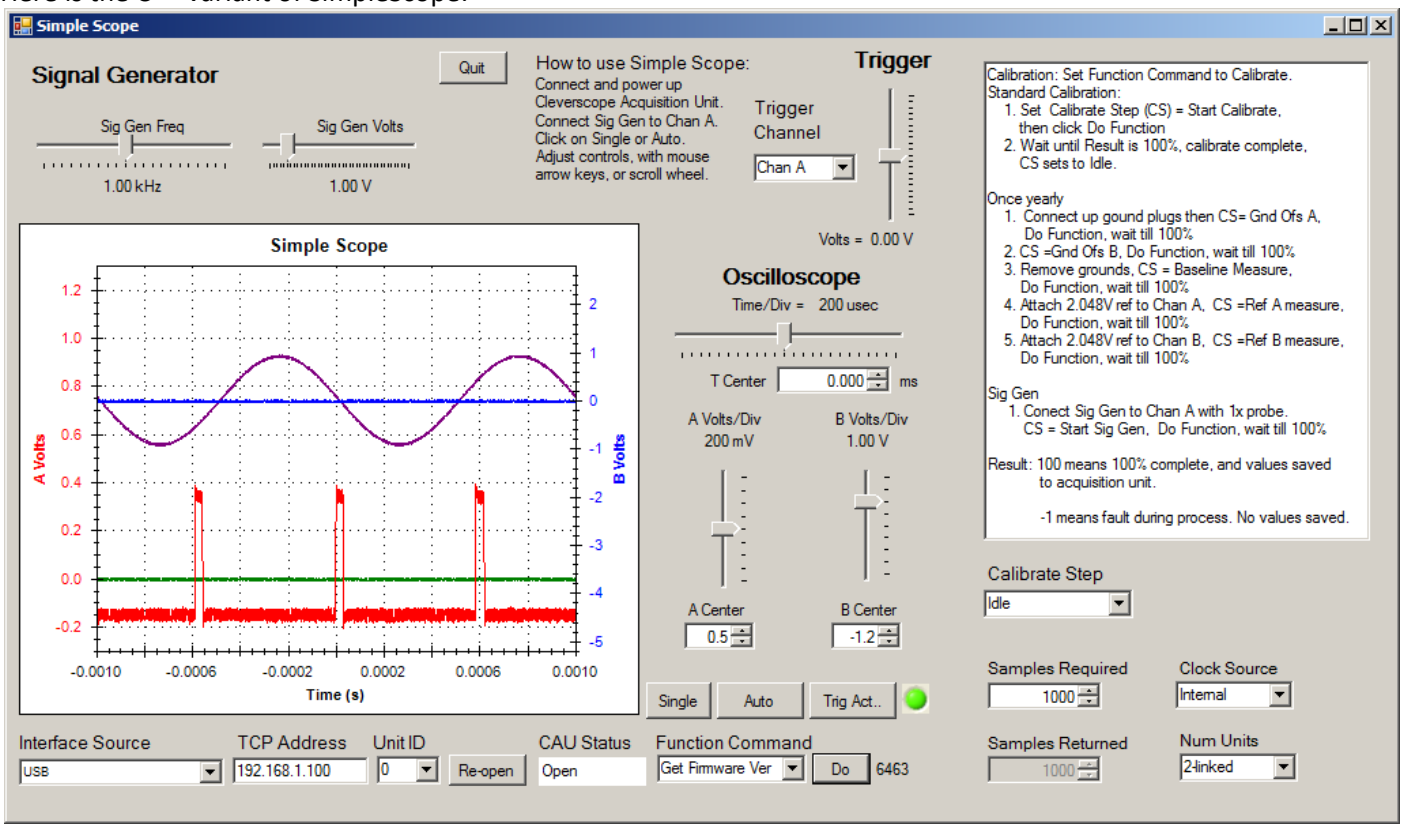
The C++ version of SimpleScope has been extended over the other text language versions, in that it can control either 1 or 2 CAU's simultaneously, and display 4 channels. These additional controls have been provided:

1. Num Units
   Connect 1 unit, 2 independent units, or 2 linked units.
2. Clock Source
   Use the internal sampling clock, or use an external clock source to synchronize sampling across both units.
3. Samples Required
   Define the number of samples required.

Following the description of SimpleScope we include some application information.

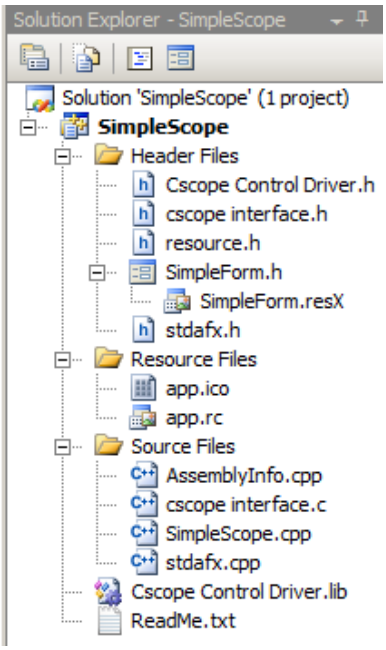
6.1 Front Panel

Here is the C++ variant of SimpleScope:



The application has been setup to capture 2 linked CAU's using the Internal clock source. Signals are connected to Channel A (Master - Red) and Channel B (Slave - purple). We have used the Center controls to shift the graph centers to separate the A and B channels. We just used the *Get Firmware Version* Function to get the firmware version installed, which is 6463. The other controls define amplitude, time, trigger setting and signal generator setup. The Triggered method of capture is Active. Note that the application connects immediately when you start it – so connect a signal to the acquisition unit, and have it on, and connected before running the application.

6.2 Application Structure



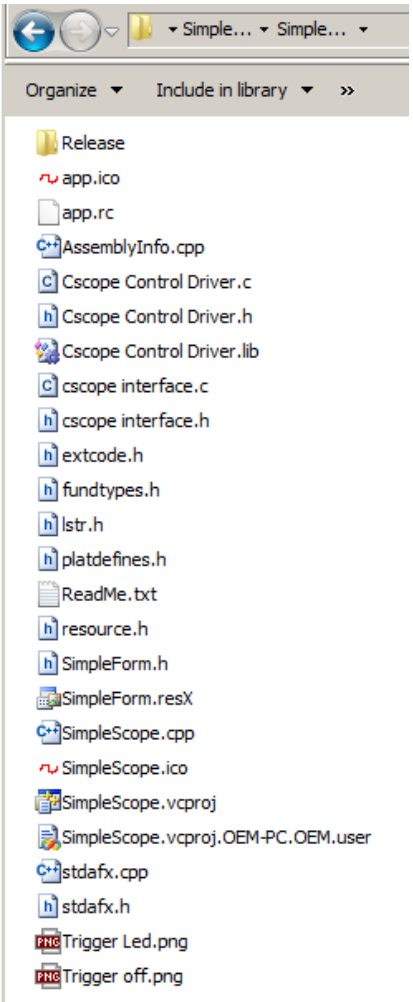
The application structure is shown to the right.

To debug the application you will need the Microsoft Visual Studio Express C++ 2008 or later C development environment. This discussion assumes you can look at the source files with a text editor of some sort.

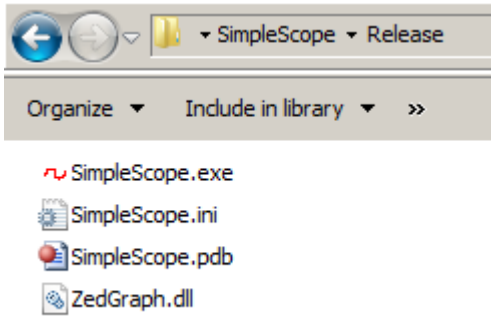
The application form **SimpleForm.h** communicates with the Cleverscope interface file **cscope interface.c** using the **cscope interface.h** header file. **Cscope interface.c** communicates with the acquisition unit hardware via USB or Ethernet using the **cscope control driver.dll** file, which is accessed using the **cscope control driver.h** header file. You will also need the **ftd2xx.dll** file for USB support, and the NI Visa I/O system. For other environments you may need other header files or the cscope.lib file to build an application.

The *SimpleScope* subdirectory holds the application files:

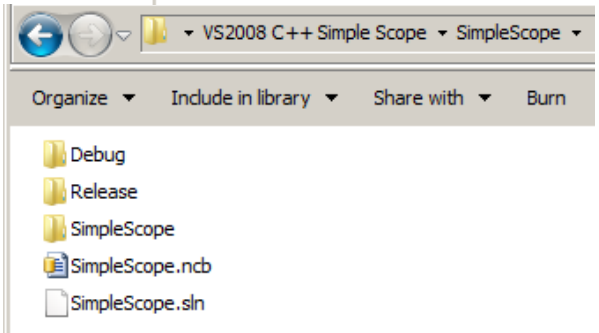
- Cscope Control driver.dll – contains the executable code to control the Cleverscope Acquisition Unit
- Cscope Control driver.h – contains a header file with the C code definitions required to call the DLL. The definitions are based on the STD CALL calling protocol.
- Cscope Control driver.lib – a linker library file used by Visual Studio.
- Cscope interface.c – the C code that is used to talk to the DLL in the example application
- Cscope Interface.h – a header file that allows the main application to call procedures in **cscope interface.c**.
- SimpleForm.h – the form that contains the design and source for the application.
- SimpleScope.cpp - the SimpleScope main file used to launch the form.
- SimpleScope.vcproj – Visual Studio project definition and build files.
- Trigger Led.png and Trigger off.png - garphics files used to implement a flashing LED.



Looking in the *Release* sub directory we find



- SimpleScope.exe – the executable generated by the compiler. It runs the example SimpleScope application.
- Zedgraph is an Open Source graphing component. See Section 2.



6.3 Key Functions in cscope interface.c

We have written cscope interface.c to simplify calling and using the control driver. You will probably want to modify the file for your own purposes. We are not C experts, and so we have used globals to provide intermediate storage. If you make a better implementation, please let us know!

These are the key functions:

`uInt16 call_cscope_control_driver(int32 unit_number, uInt16 command);`
This function calls the control driver, using the global variables to pass parameters.

`float64 get_function(int32 unit_number, uInt16 cmd_value, float64 cmd_parameter);`
This function calls a CAU function, and returns the result.

`int scope_init (int32 unit_number);`
This function initializes the acquisition unit. It returns 0 if no error, a positive integer otherwise. You can preset the acquire variable as needed.

`uInt16 cscope_interface(int32 unit_number, uInt16 Intf_source, uInt32 TCP_adr);`
This function sets the CAU interface - USB or ethernet, and unit number.

`int scope_close (int32 unit_number);`
This function closes the acquisition unit. It returns 0 if no error, a positive integer otherwise.

`int scope_finish (void);`
This function completes use of the driver, and removes runtime components. It returns 0 if no error, a positive integer otherwise.

`void update_values(int32 unit_number, float64 a_div, float64 b_div, float64 t_div,float64 a_div_center, float64 b_div_center, float64 t_div_center, int32 number_of_points, float64 freq, float64 sigvolts, float64 trigvolts, int32 trig_chan,uInt16 trigger_action, uInt32 dig_pattern, uInt32 clock_source, uInt32 unit_usage);`

This function is used to update the values of interest to you. You may wish to change it as necessary. It stores the passed value in the global Acquire variable, and in this implementation sets up the Linking between two units correctly.

`int scope_config (int32 unit_number);`
This function updates the configuration inside the acquisition unit when no acquisition is being done. . It returns 0 if no error, a positive integer otherwise.

`int scope_acquire (int32 unit_number);`
This function starts an acquisition based on the acquire variable setup. It returns 0 if no error, a positive integer otherwise.

`uInt16 scope_read_waveform (int32 unit_number, float32* a_waveform[], float32* b_waveform[], int32 *num_samples, float64 *delta_t, float64 *t_zero)`
This function returns the last captured waveforms for Channels A and B, and sampling details directly into the arrays pointed at. It returns 0 if no error, a positive integer otherwise.

`int check_for_samples(int32 unit_number);`
This function returns 1 if a trigger has occurred and samples have been returned, otherwise 0.

`int get_CAU_status (int32 unit_number);`
This function returns the current state of the CAU slot. Status values are:

0. `c_runtime_closed` - The runtime is not yet operational
1. `c_closed` - The CAU runtime has been initialized but the CAU has not been opened yet.

2. c_open – The CAU is open
3. c_fault – The CAU has a fault (loss of power, loss of interface, time out)
4. c_fault_closed – The CAU had a fault, and closed. You will need to init again.
5. c_open_pending – The CAU is open, communications are working, but the complete CAU state has not yet been transferred.

6.4 Key variables in scope interface.c

These are important variables, used for transfer between the CscopeControlDriver and the C++ app.

- acquire_defn - The variable holds the current acquisition configuration. The function **cscope update values** simply manipulates this variable. The Acquire variable holds values for each connected CAU.
- dt - the sample interval, in seconds
- t0 - the start time of the sample set, relative to the trigger point.
- samples[num_acquisition_units*2][max_samples] - arrays to hold the received samples
  pointers to these arrays are used to reduce copying
- dig_samples[max_samples] - array to hold the digital samples

6.5 Simple Scope Operation

Simple scope operation is determined by SimpeForm.h (or Simplescope.c in languages where the form is not the main control). Steps are:

- **SimpleForm_Load**, on loading the form, use the ZedGraph dll to **CreateGraph**, then use **scope_init** to open the Cleverscope Acqusition Unit. Finally call **InitializeGUI** to start the GUI.

- On opening the form, **InitializeGUI** is called to set each control to the correct value. Key operating values are initialized:
  a. single_acquire - Boolean, means acquire a single signal with triggered capture
  b. auto_acquire – Boolean, means acquire multiple signals with auto capture
  c. trigger_acquire - Boolean, means acquire multiple signals with triggered capture
  d. waiting_for_trigger – Boolean, if true, waiting for trigger. Otherwise can start acquire.
  e. Initialize graph controls and user interface controls
  f. Enable the 50ms timer. You will find the timer control on the bottom left of the Form window, below the window scroll bar. The timer calls the **timer1_tick** event every 50 msecs. This procedure is the central controller for the acquisition system.

- On a user control event: Call the event handler. Each event handler deals with a separate control value. When the value is changed, vals_changed is set true.

  Four events are important:
  a. **SingleTrigger** – start a single capture with trigger. Set *single_acquire*. Set the *trigger_action* to either single (if we haven't been waiting for a trigger) or stop (if waiting for a trigger).
  b. **Auto** – start a continuous auto capture. Set *auto_acquire*. Set the *trigger_action* to either auto (if we haven't been capturing) or stop (if capturing).
  c. **Triggered** - start a continuous triggered capture. Set *trigger_acquire*. Set the *trigger_action* to either trigger (if we haven't been capturing) or stop (if capturing).
  d. **QuitButton** – or **OnFormClose** – these events cause the SimpeScope program to exit.

- **Event Timer 1_Tick** – this is the central control function in Simple Scope. It fires every 50 msecs.

  This function proceeds in these stages:

  a. The timer is disabled - if we use more than 50 msecs, we ignore another timer tick.
  b. If we are **not** waiting for a trigger (*single_acquire* or *auto_acquire* or *trigger_acquire* are all false), we stop waiting_for_trigger, and turn off the trigger LED.
  c. We make sure all the attached CAU's are open. We don't do anything else until they are open.
  d. If we **are** waiting for a trigger, we **check_for_samples**. If there are samples, we do a **scope_read_waveform** to read the values into our waveform variables.
  e. If a waveform has been read, we display it to the graph (decimated to 1000 points).
  f. We do this for all connected CAU's.
  g. If we received a waveform, we turn on the Trigger LED, otherwise we turn it off.
  h. If we aren't *waiting_for_trigger*, and one of *single_acquire* or *auto_acquire* or *trigger_acquire* is true, and the CAU's are open, we start an acquire. If any values have changed we use **update_values** to make sure the Acquire variable is up to date before starting the Acquire. We set *waiting_for_trigger* as true.
  i. If we are calibrating, we allow the calibration state system to work.
  j. *If vals_changed* is true, we update the acquire variable using update_values. This does not update the acquisition unit itself. Because of that we do a **scope_config** to force the new values into the CAU. We do this because a user might have changed the controls while an earlier transfer is taking place.
  k. Finally we set the timer enable state to be the same as *f_timer_active* (which can be set to false in other parts of the system. This gives us an exit strategy. If **f_timer_active** is false, we **scope_close**, and then **scope_finish**, and then **exit**. This closes the form. **Scope_finish** and **scope_close** are in **cscope_interface**.
  l. The user can click Quit or the SimpleForm close box, and the **Quitbutton_click** event is called. All these do is set *f_timer_active* false, which causes the eventtimer1 to close down. The **FormClosing** function uses a semaphore (*stop_sync*) to ensure that the timer is not running while the form is closing. This ensures an orderly close down.

6.6 Applying SimpleScope

We can use the SimpleScope application to check out a number of scenarios in using the Cleverscope Acquisition Unit (CAU).

6.6.1 Linked two unit scope with Four Channel capture

Two units can be linked together using the CS1020 link cable. The link cable transfers trigger signals between the two units. In addition an external sampling clock can be used by having the CS810 option fitted at time of manufacture. The external sampling clock means that all 4 channels sample synchronously. Without the external sampling clock, the two units clocks will be different by a small degree, and the sampling points will not be time coherent

Acquire Definition

Master

Item	Explanation
Link Port	Set to 'Master' (4)
Trigger Source	Specifies the Trigger Source as normal, 0 = A chan, 1 = B chan, 2 = Ext Trigger, 3 = Dig Input, 4 = Link Port
Ext Sample Clock	Set to 0 for an internal clock, or 1 for an external fixed frequency sampling clock, 0r 2 for an external variable frequency sampling clock (no stability detection). The clock must be sine or square, 0.5 - 3V p-p amplitude, 1-110 MHz frequency.

Slave

Item	Explanation
Link Port	Set to 'Slave' (3)
Trigger Source	Set to Link Port (4).
Ext Sample Clock	Set to 0 for an internal clock, or 1 for an external fixed frequency sampling clock, 0r 2 for an external variable frequency sampling clock (no stability detection). The clock must be sine or square, 0.5 - 3V p-p amplitude, 1-110 MHz frequency.

The C++ version of SimpleScope can be used to display 4 channels. We did this:

- We assigned two units - Master (0) and Slave (1).
  In **cscope interface.c**, the **update_values** procedure, we setup the 4 channel linkage based on *unit_usage*:

```
acquire[unit_number].LinkPort = 0; //debug UART as default
if (unit_usage == two_linked)
      {
      if (unit_number == slave)
              {
              acquire[unit_number].TriggerSource = 4;              //Link Input
              acquire[unit_number].LinkPort = 3;           //This is a Slave Cleverscope
              }
      else
              {
              acquire[unit_number].LinkPort = 4;           //This is a Master Cleverscope
              }
      }
if (clock_source == external_clk)
      acquire[unit_number].ExtSampleClock = 1;
else
      acquire[unit_number].ExtSampleClock = 0;
```

  You can see that we have setup both the linkage, and the clock source here.

- Back to **SimpleForm**, In **SimpleForm_Load,** if *unit_usage* (a front panel control) is for 2 units, we **scope_init(slave)** as well.
- In **Num_units_Selected_IndexChanged**, if the user selects 2 units while operating, we **scope_init(slave)**.
  In this way we have opened both units as they are required.

- In **timer1_tick**, we check that both CAU's are open before we do anything else.

```
status = get_CAU_status(master);
if (unit_usage>one_unit)
        {
        if (status == c_open)          //only update status if master already open.
                status = get_CAU_status(slave);
        }
if (CAU_status != status)
        {
        CAU_status = status;
        Cscope_Status->SelectedIndex = CAU_status;
        }
```

  The result is that CAU_status contains Open if both units are open.

- Next, if *waiting_for_trigger* (a triggering state variable) = 2, we **check_for_samples(slave)**, and if there are any, display them. After receiving slave samples, we set *waiting_for _trigger* to 1.
- Next, if *waiting_for_trigger* = 1, we **check_for_samples(master)**, and if there are any, display them. After receiving master samples, we set *waiting_for _trigger* to 0 - we have finished.
- If *waiting_for _trigger = 0,* and an acquire has been scheduled by the user clicking on Single, Auto or Triggered, we start another scope_acuire for both the master and the slave. We set *waiting_for _trigger = 2.* We also do update values first, so any user changes get acted on.
- It doesn't matter which order the master or slave acquires are started, or the order in which they are read. It is important that once a CAU reports it has samples that they are read immediately.

6.6.2 Explanation of four channel operation

The Link port has two lines called Link_in and Link_out.

For a 2 unit combination, the slaves Link_out is connected to the masters Link_in, and the masters Link_out is connected to the slaves Link_in.

When a unit is assigned as a **slave**, it sets *Link_out to 0*, which means **not ready**. When it sees an acquire command, and it has started acquiring, it sets *Link_out to 1*, which means that is **ready**.

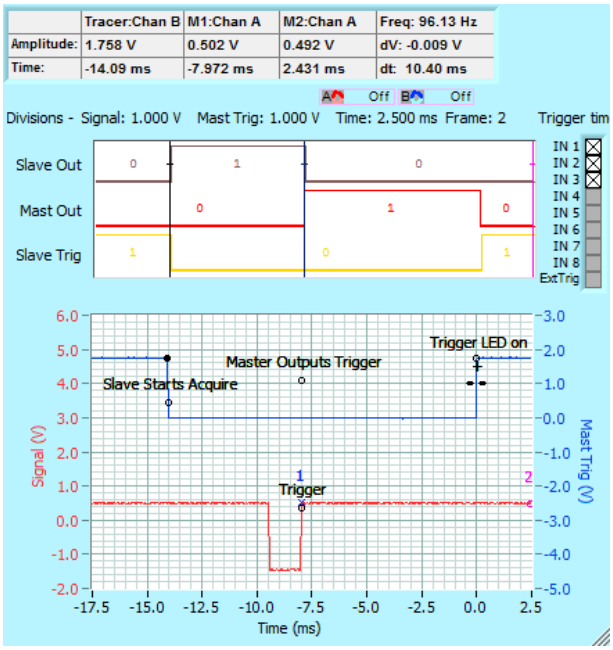
When the master is issued with an acquire command, it outputs outputs *Link_out =0* which means **not triggered** , and waits until the Link_in input is 1 - the slave is ready, and then and starts acquiring. This makes sure that both units are ready before acquisition starts.

When the master sees a trigger from the source programmed, it outputs *Link_out = 1*, which means **triggered**. The slave sees the trigger source as link_input and triggers. As a result it sets *Link_out = 0* - meaning it has seen the trigger, and is no longer ready.

When the master sees Link_in change to 0, it knows that the slave has seen the trigger, and it also sets *Link_out = 0*. The trigger has been captured by both units and both units are now waiting for the samples to be transferred back to the PC. Both units are idle until the next acquire command.

We take special care with the link port timing so that the trigger point is synchronous between both units to +/-10ns.

All this means that it does not matter in what order and how much later samples are transferred to the PC. In fact samples are transferred by the driver to the PC automatically, and you don't see that process. The routine Scope_read_waveform simply returns pointers to the data that is already in the PC. So the order of transfer does not matter.



6.6.3 Acquisition Unit based waveform averaging

The acquisition unit can be used to automatically acquire 4, 16, 64 or 128 frames synchronous with the trigger, and then average across all of them, and return one averaged waveform as a result. The result can be rendered as 10, 12 or 14 bits. In this example we will average 16 frames, and return with 12 bit precision

With Minimum Scope set these values:

Acquire.element	Value	Explanation
Acquisition mode	Waveform averaged (4)	This value sets the type of acquisition to do. We want waveform averaged, so we set the Acquire Definition. Acquisition mode to Waveform avg (4)
Waveform averages	16	We want 16 frames to be averaged together. When a 'Single' trigger is started, the acquisition unit will capture 16 frames in sequence and then carry out the waveform average.
Num Buffers	17	This is the number of buffers assigned for frame capture. This must always be at least one more than the number of frames captured sequentially (because we reserve one buffer for circular sampling to find the next trigger). In this example we set 'Num Buffers' to 17
Num Seq Frames	1	We want to return 1 frame, which is the average of the number of frames given in Waveform averages
Acquire Mode	Single (0)	We use 'Single' (0), which means capture with a trigger, and start a capture only once, but do a sequence, because Num seq frames>1.
Sampler Resolution	12 bit (1)	Set the returned values to have a resolution of 12 bits.

After setting these values in the Acquire Definition, we run Minimum Scope once, and it captures a waveform of 16 averages, with 12 bit resolution.

6.6.4 Filtering the Signal

The Cleverscope Acquisition unit includes a 20 MHz pre-filter and a 40ns ~ 1.28 us moving average filter for filtering the 100 MSPS captured data stream. When enabled the filters provide real-time results. The filters include time delay compensation so that all channels are time aligned. Using the filters improves bit resolution at the expense of sample rate. With a moving average filter time constant of 640ns, a true 14 bit ENOB can be achieved with an effective bandwidth of about 1 MHz.

For this example we will not use the 20 Mhz prefilter, but use a 640ns moving average filter, and return 14 bit filtered results. Channel B will be set to using the 20 MHz pre-filter only.

With Simple Scope set these values in the **cscope interface/scope_init** function:

Acquire.element	Value	Explanation
A Bandwidth	10101001b = 0xA9 (169 dec)	A Bandwidth (changed from firmware 4639). Bit 0 - Global Filter enable, 0 = no filter, 1 = use filter Bit 2:1 - Pre-filter frequency 0 = No filter, 1 = 20 MHz filter, 2 and 3 reserved Bit 3: - If true, use the moving average (MA) filter Bit 4: - If true use exponential (E) filter - mutually exclusive with moving average filter Bits 7:5 - Filter time constant: 000 = no filter 001 = 40ns MA , 20ns E 010 = 80ns MA, 40ns E 011 = 160ns MA, 80ns E 100 = 320ns MA, 160ns E 101 = 640ns MA, 320ns E 110 = 1280ns MA, 640ns E 111 = reserved MA, 1280ns E For the moving average only the channel A moving average value is used, and it also used for Channel B If Bit 0 is 0, then all the other bits are ignored.
B Bandwidth	00000011b = 3	Same description as above.
Sampler Resolution	14 bit (2)	Set the returned values to have a resolution of 14 bits.

Note that when using 14 bit resolution, the 32 bit memory storage system stores 2 x 14 bit analog values, and just 4 bits (In1-4) of the digital signal. Reduce resolution to 12 bits to get all 8 digital channels.

6.6.5 Sequential Capture

Sequential capture is the process of capturing a set of N frames, each started by a separate trigger, and then transferring the samples captured to the PC. The sequential frame capture system requires about 200us between triggers minimum.

Examples of sequential frame capture are in 'Bandpass Response.vi' and 'Minimum seq scope' included with the Cscope Control Driver package. TheBandpass response example uses the acquisition unit sig gen auto step facility to step the signal generator after every trigger. The example steps the signal generator over a user defined frequency range (eg 1M to 3M Hz) in steps (eg 1 kHz), using sequential capture to capture up to 3000 separate triggered frames. It then transfers the sequence to the PC, and does an FFT on each frame, and plots the results to a peak captured graph. This shows the bandpass response of any network that might be between the signal generator output and the Channel A/B inputs. Because all of the capturing and stepping is done automatically in the acquisition unit, this process is fast.

The Simple scope example can be modified to capture and display a sequence of frames. For this example we will assume we capture 200 triggered frames, each of 100us duration, and 1000 samples per frame. We will start capturing 50us before the trigger, and stop 50us after the trigger. We will transfer all the samples.

Setup

We assume the values on amplitude scale, coupling, bandwidth etc are all setup as required. This example only works through the values needed to meet what we want to do.

Acquire Definition

Item	Explanation
Start Time -50us	The capture start time, relative to the trigger point. Can be + or –. The value -10u means start capturing 10us before the trigger. The value 600m means start capturing 600 ms after the trigger (ie delay after the trigger). We use -10u.

Stop Time +50us	The capture stop time relative to the trigger. We will use 40us. The total capture duration will be 50us.
Num Buffers 1000	This is the number of buffers assigned for frame capture. This must always be at least one more than the number of frames captured sequentially (because we reserve one buffer for circular sampling to find the next trigger). In this example we set 'Num Buffers' to 1000 in the Acquire definition.
Num Sequence Frames 200	This is the number of frames we want to capture in sequence. Set this to 200. Now when a 'Single' trigger is started, the acquisition unit will capture 200 frames in sequence and then signal completion.
Acquire Mode 0 (Single)	This specifies how we are going to capture the frames. We use 'Single' (0), which means capture with a trigger, and start a capture only once.
Transfer Size Sequence (6)	This value specifies the size and type of transfer being used. 0 is Normal. The value we want is 'Sequence' (6) which means the acquisition will transfer a full sequence whenever it is told to do a transfer. We set up the Acquire definition to sequence.

Input values to specify

In addition to the Acquire Definition, we have values that specify the playback. For Cleverscope the capture process is setup independently of the playback process (think of a tracker graph displaying 10us of a signal that is 800ms wide). So we need to specify these values:

Item	**Explanation**
Replay Start Time -50u	The replay start time, relative to the trigger point. Can be + or –. The value -5u means start displaying 5us before the trigger. The value 300m means start displaying 300ms ms after the trigger. We use -50u, which is the same as the capture start time (it can be different).
Replay Stop Time +50u	The replay stop time relative to the trigger. We will use 50us. The total replay duration will be 1050us, which is the same as the capture duration.
Num Samples 1000	This specifies how many samples we want in our replay. It may be truncated. The Cleverscope acquisition unit might capture 4M samples for a particular signal. We might only want to playback 20,000. The replay system automatically decimates the samples in the acquisition unit to meet the replay values. In this example we want 1000 samples.
Acquisition Unit 0	We can control up to 32 concurrent units. We only want one for now, so we leave the Acquisition Unit input unwired, to default to 0.

To implement this example, set up the Acquire variable values, and then add an extra section into the **Timer1Tick** function. First we add an extra meaning to the *waiting_for_trigger* state variable. Lets say that 3 means trigger acquired, and now doing a replay.

Modify the existing:

```
if (waiting_for_trigger == 1)    //waiting for master acquire
   {
   if (check_for_samples(master))
            {
             Trig_LED->Visible = true;
             waiting_for_trigger = 3;  //start a replay
             scope_replay (int32 unit_number, replay_start, replay_stop, num_samples,num_frames);
                  //start a replay
             }
            }
```

and add:

```
if (waiting_for_trigger == 3)    //waiting for master replay
   {
   if (check_for_samples(master))
            {
             waiting_for_trigger = 0;  //finished

             for (i=0; i < num_frames; i++)
                   {
                   scope_read_frame (master,i, &a_waveform, &b_waveform, &num_samples, &dT, &T0);
                   //replay all the frames, and do the normal display stuff
                   //....
                   }
            }
   }
```

1. The Acquire Mode is set to single when you click the 'Single Trigger' button. The acquire definition needs to be manually set to include the number of buffers allocated to 1000, in **scope_init**. Once the buffers are allocated we can proceed to acquire up to num_buffers – 1 frames to the buffers.
2. Following Acquire, we call the control driver with 'Wait for Samples' until we see that the samples have been captured. Each frame will have been individually triggered until the number of frames specified have been captured.
3. Once the samples are in CAU memory, we need to transfer them as a frame transfer to the PC (Acquire does include the capability to automatically transfer the latest frame to the PC, but this is not sufficient to transfer all the frames.). This is done by starting a Replay. Because we captured a sequence, the replay will replay the full sequence.
4. After starting the replay we need to make sure all the samples have been transferred to PC memory. We use **Wait for Samples** for this.
   Once we have the samples in PC memory, we can get the frames, one at a time from PC memory, and process them as we wish. Here we display them. We use the **Get Frame** command to do this.

6.6.6 Auto step Signal Generator Control

It is possible to automatically step the signal generator following an acquisition. This is useful for carrying out voltage/frequency sweeps while capturing a sequence. The Labview **Bandpass** response example shows how this was done.

Acquire Definition

Item	Explanation
Sig Gen Func 1	Defines the signal generator function. Without Auto stepping we use 'Standard' (0). We want 'Auto advance' (1) to automatically step the signal generator after an acquisition is complete. This happens in the acquisition unit.
Sig Gen Freq Step 1000	Specifies the step size in Hz. In our example, it is user configurable, and defaults to 1kHz.
Sig Gen Freq	Specifies the start frequency for auto stepping. In our example we use 1MHz

We must update the Acquire Definition and input it to the Cscope Control Driver when calling the driver with the Acquire or Update commands.

Use

At a minimum we need to do these things:

1. Setup the Acquire definition the way we want it.
2. Call the control driver with the command **Acquire** (1). By combining the capture with a sequence (see above), we can capture thousands of frames, and the signal generator will automatically step between each capture. In this way a frequency response can be measured