Advanced Sample Programs in CompuScope SDKs

There are three GaGe Software Development Kits (SDKs) for user programming of GaGe CompuScope cards: one for C/C#, MATLAB, and LabVIEW. Each of these SDKs includes advanced sample programs that are provided in a separate sub-folder and that are not described within the standard CompuScope SDK documentation. These Advanced sample programs are described within this document. This document describes the advanced sample programs in a generic fashion that is not specific to any of the three SDKs.

The table below lists which sample programs are supported under which programming language. Some sample programs for VisualBasic.NET and Delphi are provided within the C/C# SDK. The user may use the C Sample programs as a guide to construct sample programs that are unavailable for VB.NET and Delphi. Usage of some advanced sample programs requires installation of optional eXpert firmware. These programs are indicated by an asterisk (*).

	C	C#	VB .Net	LabVIEW	MATLAB	CVI	Delphi
Simple	X	X	X	X	X	X	X
Acquire	X	X	X	X	X	X	X
Coerce	X	X		X	X	X	
ComplexTrigger	X	X		X	X	X	
DeepAcquisition	X	X		X	X	X	
MultipleRecords	X	X		X	X	X	
MultipleSystems	X	X		X	X	X	
Average*	X	X		X	X	X	
FFT*	X	X		X	X	X	
FIR*	X	X		X	X	X	
MinMaxDTC*	X					X	
CsPrf	X	X				X	
GageStream2Disk	X	X		X	X		
Callback	X					X	
Events	X	X				X	
AsTransfer	X	X				X	
AdvMulRec	X			X		X	
AdvMulRecEx	X	X		X	X		

To determine which CompuScope models are supported by which advanced sample programs, please check the online Advanced Functionality Matrix at:

http://www.gage-applied.com/products/eXpert_FPGA_technology/eXpert_compatibility_matrix.htm

Optional Firmware images

Some CompuScope models have the ability to be reconfigured with optional alternative firmware allowing on-board processing of waveform data before they are transferred to PC RAM. Currently, these firmware options include Finite Impulse Response (FIR) Filtering, Signal Averaging, and Peak Detection. When a CompuScope is updated with an optional firmware image, the image information is stored in CompuScope non-volatile memory. This memory has space for three firmware images: the standard CompuScope operating image and up to two optional images. The contents of non-volatile memory may be queried by an application using the CsGet(hSystem, CS_PARAMS, CS_EXTENDED_OPTIONS, &i64ExOptions) call, where hSystem is the CompuScope system handle. CS_PARAMS and CS_EXTENDED_OPTIONS are constants defined in CsDefines.h and CsExpert.h respectively. I64ExOption is a 64-bit integer type variable that is filled with the results of the query. The lower 32 bits of the i64ExOption will contain information about first alternative image and the higher 32 bits will contain information about the second one. There are corresponding calls to query available firmware in MATLAB (CsMl_GetExtendedOptions.m) and LabVIEW (CsLv_GetExtendedOptions.vi).

Based on the information about available firmware images, the user application can decide which image to load. Firmware images are loaded from any SDK by bitwise ORing the CompuScope mode (1 for "Single", 2 for "Dual" and 4 for "Quad") with a specific constant that indicates the image number.   For instance, to specify an alternative image from C, either the constants CS_MODE_USER1 or CS_MODE_USER2 (for images #1 or #2) should be bitwise ORed with the u32Mode member of CSACQUSITIONCONFIG structure before it is used in the CsSet() call. (Note that the firmware image is not actually loaded until a call is made to CsCommit()).

Each of the CompuScope SDKs (C/C#, MATLAB and LabVIEW) provides a programming example for each optional firmware image (currently FIR filtering and signal averaging). The programming sequence for the loading each image, which is described above for C, is illustrated within each programming example

GageAverage

Usage of the signal averaging optional firmware image allows repetitive waveform acquisitions to be rapidly averaged, in order to reduce random noise. In the past, signal

averaging required waveforms to be downloaded for averaging within the host PC's CPU so that averaging was limited by the data transfer speed. With the signal averaging firmware, repetitive waveforms are averaged within the firmware with no data transfer required until up to 1024 averages have been performed. Consequently, much higher repetition rates may be achieved.

Once the signal averaging firmware has been loaded, the user adjusts the number of averages to be acquired by using the variable for the Number of Records, which is normally used to select the number of records to be acquired in a Multiple Record acquisition. With the signal averaging firmware loaded, the CompuScope hardware co-adds this number of consecutive waveforms, instead of stacking them in on-board memory as is usually done in Multiple Record Mode.

Co-added waveform data are stored within a 32-bit format buffer within the on-board firmware. The resulting averaged waveform must therefore be transferred as a 32-bit data buffer. With the averaging firmware loaded, the Sample Size, Sample Resolution and Sample Offset values are changed to reflect the 32-bit data format. Querying these parameters after the firmware is loaded will return the updated values. Co-added waveform data must be divided by the number of waveform averages in order to obtain the averaged waveform. Updated Sample Resolution and Sample Offset values may then be used for waveform voltage conversion.

Each SDK contains an advanced sample program that uploads the signal averaging image, performs a signal averaging acquisitions with an adjustable number of averages, and then displays or stores the resulting averaged waveform.

For the Normal eXpert Signal Averaging operation described above, a single averaged waveform is stored within CompuScope FPGA memory, after which it must be downloaded to PC RAM. Some newer CompuScope models are equipped with a superior eXpert Averaging functionality called *Multiple Record Averaging*. This functionality allows multiple averaged waveforms to be stacked in on-board memory, as are non-averaged waveforms in Multiple Record Mode. For instance, the user could select to average 4096 Samples waveform 1000 times each but then stack 10,000 of these averaged waveforms in on-board memory, after which all 10,000 averaged waveforms are downloaded to PC RAM.. Normal Averaging operation is achieved simply by selecting to stack only one averaged waveform in on-board memory.

GageFFT

Fourier Transform analysis is a powerful signal processing technique that allows analysis of time-domain waveforms in the frequency domain. Practically speaking, sampled time-domain waveforms are generally transformed into Fourier frequency domain spectra using the Fast Fourier Transform (FFT) Algorithm.

Historically, Fourier transformation of time-domain waveforms from CompuScope digitizers was achieved by executing the FFT algorithm on the host PC in which the CompuScope

hardware was installed. Unlike comparatively simple processing operations, such as signal averaging or FIR filtering, FFT calculation is computationally intensive and therefore puts a heavy processing load on the host PC.

In order to enable rapid Fourier spectral analysis without loading the host PC, GaGe has created the on-board eXpert FFT firmware option, which executes FFT calculations on time-domain waveform data acquired by CompuScope hardware. With the eXpert FFT firmware, CompuScope acquisition proceeds as usual in Single or Multiple Record acquisition mode. Once data acquisition is complete and waveform data reside in on-board CompuScope acquisition memory, the data are downloaded through an FFT algorithm with the CompuScope's on-board FPGA so that the host PC directly receives FFT spectra into its PC RAM target buffer. In fact, the user may elect to bypass the FFT processing for troubleshooting so that the regular time-domain data are transferred.

The FFT firmware actually consists of three separate FPGA images: one each for the FFT analysis of 512, 1024 or 2048 time-domain waveform points. The user manages the loading of these three images as usual using the CompuScope Manager Software utility.

Details of the operation of the FFT firmware is contained within an accompanying document called "FFTReadMe.TXT", as well as in comments throughout the FFT Sample program source code within each of the three Software Development Kits (C/C#, LabVIEW and MATLAB).

After acquisition, the user may select to download a specified number of acquired waveform records through the FFT firmware, starting from a selectable first record. Within each record, the user may specify to begin download of waveform data from a selectable starting point within the acquired waveforms. The number of points upon which the FFT is applied is determined by the loaded image and may be 512, 1024 or 2048 points.

The time-domain waveform data are entered into the FFT algorithm as Real values with the Imaginary parts set equal to zero. Consequently, the Real and Imaginary parts of the Fourier spectra are always respectively symmetric and asymmetric. That is:

$$\mathrm{Re}\{\Im(i)\} = \mathrm{Re}\{\Im(N-i)\}$$
$$\mathrm{Im}\{\Im(i)\} = -\,\mathrm{Im}\{\Im(N-i)\}$$
$$0 < i < N$$

where $\Im(i)$ is the complex Fourier amplitude in the ith frequency bin. This fact may be used to calculate the Power spectrum using only half of the downloaded data, which is done within the FFT SDK sample programs.

The FFT spectral data are transferred as N+1 32-bit values, where N= 512, 1024 or 2048, depending on the FPGA image. The first N values are the complex Fourier amplitudes for the N frequency bins. The upper and lower 16-bits of each 32-bit word respectively contain the signed 16-bit values for the Real and Imaginary parts of the Fourier Amplitude. The Real and Imaginary parts of the spectra are often respectively called the In-phase and Quadrate spectral components (I&Q). In fact, only the upper 14-bits of the 16-bit Real and Imaginary values

contain Spectral Information and the lower two bits must be masked to zero in order to extract the correct 14-bit values, as is illustrated within the FFT SDK sample programs.

The last 32-bit value transferred for a given Fourier Spectrum contains an Exponent Factor, which must be extracted from the 32-bit value as illustrated within the FFT SDK sample programs. The programs illustrate how the Exponent Factor must be used to obtain the correct normalized spectral amplitudes. The Exponent Factor is generated within the FPGA FFT algorithm in order to ensure that the dynamic range of the Fourier Spectrum values is optimally exploited.

A diagram illustrating the layout of the transferred data is shown below. The N 14-bit FFT values are shown with their Real part (I) and Imaginary part (Q) occupying bits #18 to #31 and #2 to #15, respectively. (DC = "Don't Care" and N is the number of points in the FFT, which is 512, 1024 or 2048). The last value contains the Exponent Factor, EXP, embedded as bits #2 to #7.
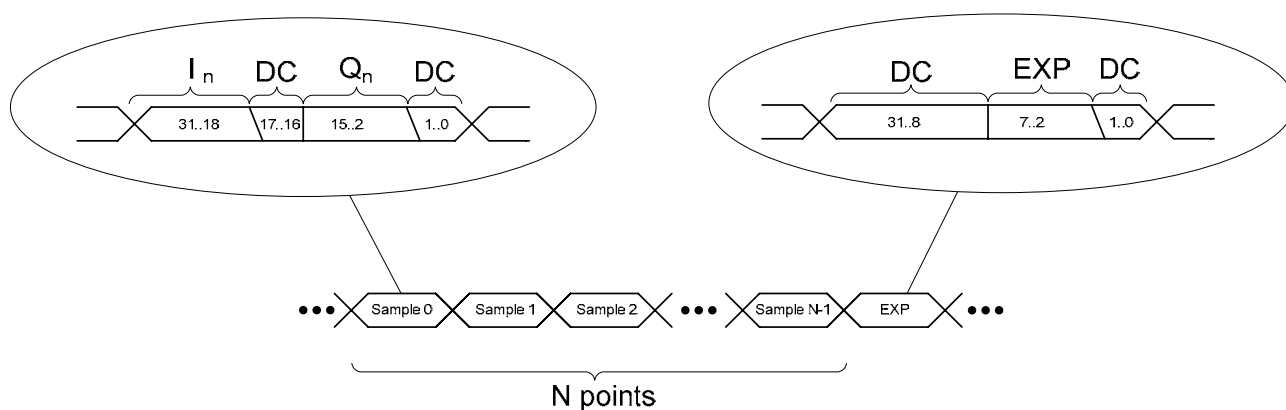
The user may select to apply a time-domain Windowing function to the time-domain waveform data before Fourier Transformation. The SDK programs allow selection of several standard Windowing functions, such as Hamming and Blackman-Harris Windowing functions. The user may also configure an arbitrary Windowing function. The span of values for the Windowing function values is the same for all CompuScope models and is -8191 to +8192. The user must calculate a normalization factor, which is the sum of all windowing coefficients, and must enter this factor along with the Exponent Factor in order to calculate the normalized Fourier Amplitudes, as illustrated within the FFT SDK sample programs.

A Block is defined as the complete data set for a Fourier Spectrum. As discussed, this includes the N 32-bit FFT values, where N=512, 1024 or 2048, plus one 32-bit value containing the Exponent Factor. Consequently, the Block Size is N+1. Accordingly, whenever the user allocates memory to hold FFT Spectra, they must allocate space a Block Size buffer for each spectrum.

The eXpert FFT firmware significantly reduces the processing load on the host CPU. In addition, the user may exploit the firmware's ability to transfer multiple spectra in one PCI

transfer, thereby reducing the overhead operations required to set up multiple PCI transfers. By selecting the optimal transfer mode, the user can maximize overall data throughput.

GaGe has created two separate Multi-Block Transfer Modes in order to allow maximal data throughout. The first mode is activated by setting the FFTMR flag in the FFT data structure to 0. This mode allows the user to obtain in one PCI transfer FFT spectra from contiguous time-domain regions within one acquired record. The acquisition may be done in either Single Record or Multiple Record Acquisition Modes.

As example, consider a user who has pre-loaded the 2048 point FFT expert image and who has performed a 30,000 sample acquisitions in Single Record Mode. Since 30,000/2048 = 14.7, the user would be able to obtain, in a signal PCI transfer, 14 FFT Spectra blocks of 14 contiguous portions of the waveform ,starting from any desired starting point. In the PCI Data transfer call, therefore, the user must request the transfer of a number of values equal to: 14 X BlockSize = 14 X (2048 +1) = 28,686 Samples. If the acquisition had been performed in Multiple Record, the same process could be repeated for each Multiple Record waveform.

Using the second option, the user sets the FFTMR flag in the FFT data structure to 1. This mode only operates if a Multiple Record Acquisition has been performed. The mode allows the transfer to transfer a single FFT Spectral Block from different Multiple Record waveforms in one single PCI transfer. For instance, consider the acquisition of 1,000 waveforms of 3072 Samples each in Multiple Record Mode. The user can elect, for example, to transfer Fourier transfer performed upon 2048 samples beginning form sample #2000 within each of the 1000 Multiple Record waveforms. These 1,000 spectra would then all be transferred within a single PCI transfer operation. The mode allows transfer of only a single spectrum from each record. If the user wants to transfer multiple spectra from each waveform, then he must set FFTMR=0.

In comparing FFT spectra obtained using the eXpert FFT firmware and those calculated in software, the user may find slight discrepancies between the two results. These discrepancies result because the eXpert FFT calculations are done using 14-bit precision integers, while FFTs performed by software algorithms generally use 23-bit precision or more. The calculation differences result, for instance, in a higher base-line spectral noise of the eXpert FFT spectra. These discrepancies are insignificant in most applications.

The eXpert FFT firmware may be operated by the CompuScope C/C#, LabVIEW or MATLAB SDK. From the LabVIEW SDK, the user may perform a Single Record or Multiple Record acquisition and then the eXpert FFT VI normalizes and displays resulting Fourier power spectra in graphical form. If the user selects to download multiple waveforms, the resulting spectra are averaged together to reduce spectral noise. This averaging is done simply as a convenient means of illustrating the multi-Block Transfer. The user may easily modify the VI to handle each spectrum separately, without averaging.

From the C/C# SDK, transferred Fourier spectra resulting from a Single or Multiple Record acquisition are stored in ASCII text files, which may be of three separate types. In the first type of file, the raw unprocessed 32-bit values from the transfer are stored. This file type is best where real-time processing is not necessary and the user wants to maximize the repetitive waveform acquisition rate. In the second type of ASCII file, Power spectral amplitudes are

stored as percentages with respect to full scale. The third type of file stores these same Power spectral amplitudes in dB, where:

$$dB = 10 \times \log_{10}\left(\frac{Power\ Spectral\ Amplitude}{100\,\%}\right).$$

GageFIR

Finite Impulse Response (FIR) filtering of waveform signals is a powerful method for removing unwanted signal features (like noise) and emphasizing signal features of interest. Unlike signal averaging, multiple repetitive waveforms need not be acquired and an FIR filtering algorithm may be applied to a single waveform data set.

The general form of the FIR filter is:

$$Y_i = \sum_{j=0}^{N} A_j X_{i-j}$$

where:

$\{X_i\}$ is the *input data set,*
$\{Y_i\}$ is the *output data set,*
$\{A_j\}$ is the *set of FIR filter coefficients* ($0 \leq j < N$)
and N is the *number of taps* and is equal to the number of coefficients.

The CompuScope FIR firmware image allows up to 20 distinct tap coefficients to be used. FIR filtering is implemented as a numerical *convolution* algorithm. Since both waveform data and tap coefficients are real values with no imaginary components, the FIR filtering algorithm may be used to implement a numerical *correlation* algorithm, simply by reversing the order of the tap coefficients.

During data transfer to the PCI bus, the FIR filtering algorithm is applied to waveform data that have already been acquired into CompuScope acquisition memory. Since the waveform data in CompuScope memory remain unfiltered, different filters may be applied to this same raw waveform data. This is done simply by modifying the tap coefficients and downloading the waveform data again.

Many sets of standard FIR filter coefficients or *filter core*s are symmetric, meaning that *Aj = A-j*. The FIR filtering firmware image allows up to 39 symmetric tap coefficients to be loaded. In this case, the FIR filter calculation is modified to be:

$$Y_i = \sum_{j=0}^{N} \left(A_j X_{i-j} + A_j X_{i+j} \right)$$

As for the signal averaging firmware, FIR filtered data are returned in a 32-bit data format. With the firmware loaded, the Sample Size, Sample Resolution and Sample Offset values are changed accordingly.

In addition, a coefficient scaling factor that scales all tap coefficients is provided. The idealized coefficients *Aj* that are listed above are related to the coefficients, *i16CoefFactor[j]*, that are loaded to the FIR filtering image as follows:

$$A_j = \frac{i16CoefFactor[j]}{u32Factor}$$

where *u32Factor* must be a power of 2 with limits listed below. The default value of *u32Factor* used within the FIR filtering sample programs is 32768.

Since the idealized coefficients, *Aj*, are floating point values and generally have absolute values less than 1, greater numerical precision on these coefficients may be obtained by increasing the value of *u32Factor*. However, using a larger value increases the risk that the FIR filtered data values will exceed the available 32-bit width of the output data buffer. Optimal selection of *u32Factor* also requires knowledge of the amplitude of the acquired signal, since larger signal amplitudes will lead to earlier overload of the 32-bit output data buffer.

As an example, consider a symmetric moving average filter core with 39 constant coefficients. Let us assume further that the data may cover the whole 14-bit ADC range of a CompuScope 14200. In this case, summing full scale data points 39 times requires an extra 6 bits, since $2^6 = 64$. This leaves only $32 - (14 + 6) = 12$ bits. Consequently, in order to guarantee no output data overload, *u32Factor* should be no larger than $2^{12} = 4096$.

From C, the FIR operation is configured by a call to CsSet(hSystem, CS_FIR_CONFIG, &FirConfig), where hSystem is the CompuScope system handle, CS_FIR_CONFIG is a constant defined in CsDefines.h and FirConfig is a variable of the type CS_FIR_CONFIG_PARAMS that is defined in CsExpert.h. This call does not require a commit action and takes effect immediately.

The parameters for configuration of the FIR filtering algorithm are specified within a variable of type CS_FIR_CONFIG_PARAMS:

Field name	Type	Description
u32Size	uIn32	Total size, in Bytes, of the structure
bEnable	BOOL	Enable FIR. If Disabled, a unity filter is used
bSymmetrical39th	BOOL	If true, assume that the coefficients are part of a 39-tap symmetrical filter core
u32Factor	uInt32	Scaling factor used for all coefficients. Allowed values are $2^{(2*n+1)}$ $1 <= n <= 10$
i16CoefFactor	int16 [16]	Core coefficients are represented in a fixed point format. This array contains numerators, while the denominator is stored in the u32Factor field

Each SDK contains an advanced sample program that uploads the FIR filtering image, performs an acquisition with FIR filtering, and then displays or stores the resulting waveform.

GageMinMaxDtc (Peak Detection)

The eXpert Peak Detection firmware option allows on-board detection of the minimum and maximum amplitudes that occur within a waveform, along with their positions within the waveform. Calculated Peak Information Sets for each waveform are accumulated within the CompuScope FPGA for PCI download. The data reduction associated with transforming the raw waveform data into the compact Peak Information Set correspondingly reduces the PCI data traffic so that a faster repetitive capture rate may be accommodated. This section describes operation of the eXpert Peak detection firmware from the C programming environment. Please note: the terms Peak detection and MinMax detection are used interchangeably. The terms Peak Information Set and MinMax Segment Info structure are also used interchangeably.

Usage of Peak Detection from C

In the "Advanced" folder within the "C Samples" folder of the CompuScope C/C# SDK is a Visual C sample project called GageMinMaxDtc that operates CompuScope hardware using the eXpert Peak Detection firmware option. This project configures the CompuScope hardware and does an acquisition using the eXpert Peak Detection firmware until a pre-set number of waveforms have been acquired. The project stores a selectable number Peak Information Sets in an ASCII file.

GageMinMaxDtc receives input configuration settings from an INI file, as do standard C SDK programs. Standard CompuScope input parameters are listed within the INI files and are the same as those documented in the C SDK. The following parameters are added for control of Peak Detection acquisitions.

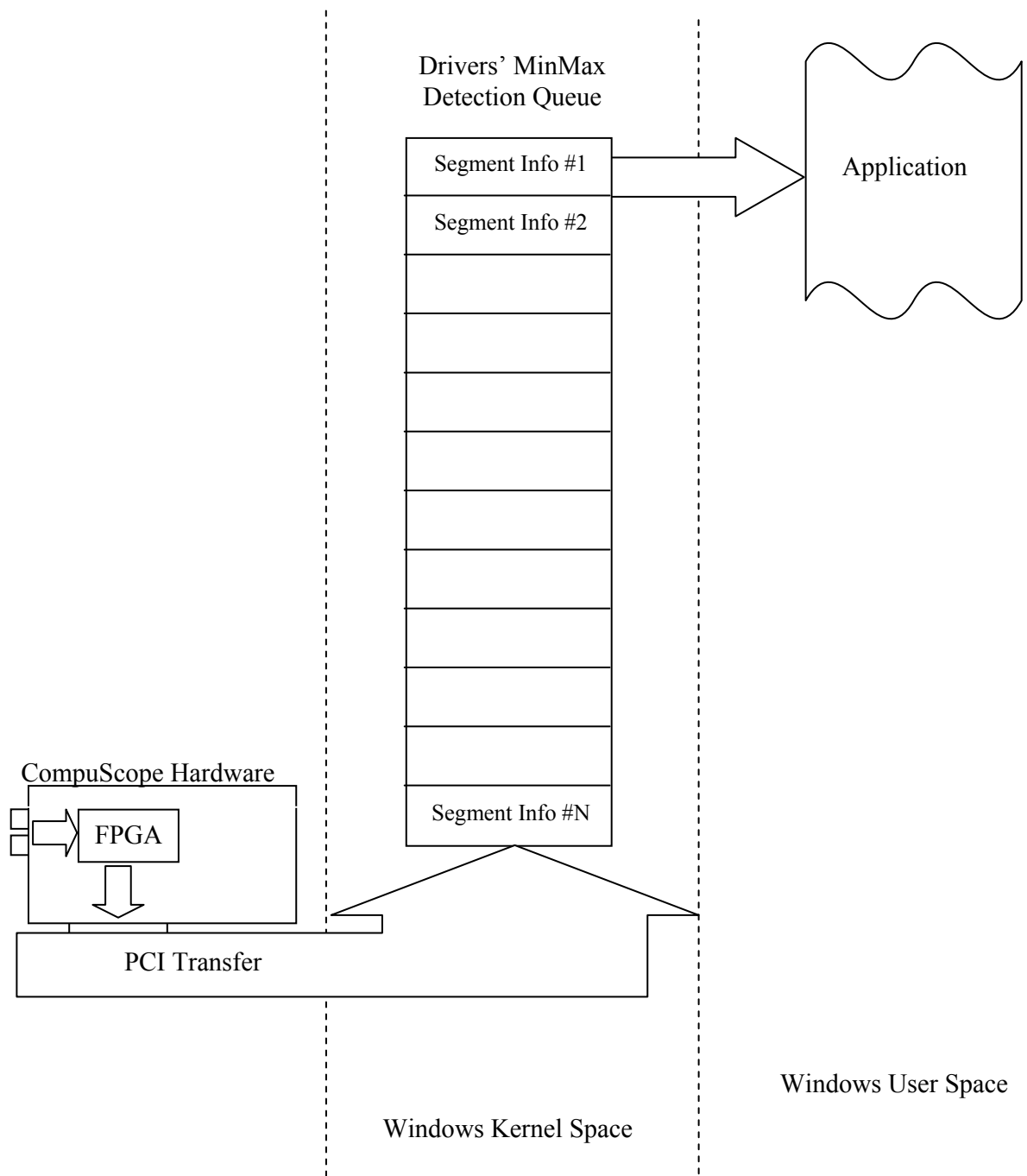**SegmentCount** - Determines the number of waveforms to be acquired by the program.

QueueSize - Sets the number of entries within MinMax Detection Queue that is described below.

LastSegmentSave - Determines the number of peak data sets to be stored in the output TXT file. If this value exceeds SegmentCount, then the most recently acquired segments are those that are stored.

TsResetMode - This setting determines when the Time Stamping counter is reset. When it is set to 0, the Time Stamping Counter is reset only once at the start of the acquisition sequence. When it is set to 1, the Time Stamping Counter is reset at the beginning of each segment acquisition. For peak detect operation, this setting should always be 0.

DetectorResetMode - This parameter determines where the Peak detection algorithm should begin looking for peaks within the acquired waveform. When DetectorResetMode is set to 0, the Peak detection (also called MinMax detection) algorithm is reset at the trigger position within an acquired segment. This way, all peaks will be detected within post-trigger waveform data only. When DetectorResetMode is set to 1, the MinMax detection algorithm is reset immediately at the start of segment acquisition so that MinMax peaks may be detected within pre- or post-trigger data.

In order to understand operation of the eXpert Peak Detection firmware option it is important to understand the hardware/firmware/software architecture, which is illustrated in the Figure below. First, waveform acquisition by the CompuScope is triggered, as usual. Instead of storing the raw data in CompuScope on-board memory, the raw waveform data are analysed within the on-board FPGA to determine the peak parameters. These parameters are assembled into the Peak Information Set (also called the "MinMax Segment Info structure" in the API function descriptions). Peak Information Sets are accumulated within the FPGA and are periodically PCI transferred by the driver to the "MinMax Detection Cue", which is a Windows Kernel Level buffer that may accommodate a number of Peak Information Sets that is specified by the QueueSize value in the INI file.

Drivers' MinMax
Detection Queue

| Segment Info #1 |
| Segment Info #2 |
| |
| |
| |
| |
| |
| |
| |
| |
| |
| Segment Info #N |

Application

CompuScope Hardware

FPGA

PCI Transfer

Windows User Space

Windows Kernel Space

Peak Information Sets are constantly added to the MinMax Detection Cue as they become available. The user application polls the MinMax Detection Cue to check if new sets are available. When available, new sets are transferred from the Kernel Level MinMax Detection Cue to an Application Level buffer, where they may be accessed by the application for analysis, display or storage.

The contents of the Peak Information Set (or MinMax Segment Info structure) and the size of each entry are shown in the table below.

PEAK INFORMATION SET (8 BYTES + 16 BYTES + 24 BYTES × NUMBER OF CHANNELS)					
NAME	DESCRIPTION	TYPE	SIZE	GROUP NAME	GROUP SIZE
Structure Size	The size of the base structure for compatibility purposes	uInt32	4 Bytes	Header	8 Bytes
Number of channels	The number of channel information sets	uInt32	4 Bytes		
Trigger Number	The trigger count, which may be used to account for missed triggers, if any	uInt32	4 Bytes	Trigger information set	16 Bytes
Reserved		uInt32	4 Bytes		
Trigger Time-Stamp	The Time-Stamp counter output that marks the occurrence time of the trigger event.	int64	8 Bytes		
Max Amplitude	The maximum value that occurs within the waveform data set	int16	2 Bytes	Channel information set	24 Bytes
Min Amplitude	The minimum value that occurs within the waveform data set	int16	2 Bytes		
Reserved		uInt32	4 Bytes		
Max Time-Stamp	The Time-Stamp counter output that marks the occurrence time of the maximum	int64	8 Bytes		
Min Time-Stamp	The TimeStamp counter output that marks the occurrence time of the minimum	int64	8 Bytes		

The Peak Information Set contains the Min and Max Amplitude information in units of raw ADC samples. Also included are the Time-Stamp time values for the trigger event, and the MinMax positions. These values come from the on-board Time-Stamping counter and may be converted to absolute time values with knowledge of the Time-Stamping counter's clock source frequency, which may be obtained by the driver, as usual.

A distinctive parameter in the Peak Information Set is the Trigger Number parameter. It is possible that, while the Peak detection algorithm is processing a waveform caused by a given trigger, another trigger event occurs. This trigger event and its associated peak information will thus be missed by the CompuScope hardware. In this event, however, the Trigger Number value will be incremented by the missed trigger. Consequently, the user will find that the Trigger number between consecutive Peak Information Sets increased by 2 instead of by 1 and so will know that a trigger was missed and can correctly account for it.

From the above chart, we may calculate the size of a Peak Information Set as:

24 Bytes + 24 Bytes × Number of active channels

Since waveform acquisitions typically consist of thousands of Bytes of data or more, the Peak Detection firmware clearly leads to a significant reduction in data volume and the associated PCI transfer traffic. This reduction allows much higher repetitive waveform capture rates to be achieved without trigger losses.

In order to provide further clarification, important steps within the GageMinMaxDtc sample project are described below. Within GageMinMaxDtc, configuration of standard CompuScope input parameter settings is done as usual.

1) Create the MinMax Detection Queue

   This is done by making a call to CsExpertCall() with a function Id of EXFN_CREATEMINMAXQUEUE within the FunctionParams structure. Once this function has been executed, the MinMax Detection Cue is set up and the application will be able to receive the handle of *hDataAvailableEvent* event.

2) Set the acquisition mode to CS_MODE_USER1 (or CS_MODE_USER2)

   The step loads the Peak Detection image from the CompuScope on-board flash memory to the CompuScope FGPA. The user must first determine whether the Peak Detection image resides in the USER1 or USER2 flash location. This may be determined from CompuScope Manager or from the driver.

   This example code loads the USER1 image to the FPGA:

   ```
   CsAcqConfig.u32Mode = CS_MODE_USER1 | CS_MODE_DUAL;
   CsSet(hSystem, CS_ACQUISITION, &CsAcqCfg);
   CsDo(hSystem, ACTION_COMMIT);
   ```

3) Start acquisition.

   Example:
   ```
   CsDo(hSystem, ACTION_START);
   ```

4) Wait for the *hDataAvailableEvent* event.

5) Once the event is signalled, call CsExpertCall() with the function Id EXFN_GETSEGMENTINFO to retrieve a single MinMax Segment Info structure entry (Peak Information Set) from the driver.

6) Repeat calling CsExpertCall() with the function Id EXFN_GETSEGMENTINFO until receipt of error CS_SEGMENTINFO_EMPTY. This error indicates that all available Peak Information Sets have been transferred to the application.

7) Process, display and/or store the data, if necessary

8) Return to step 4

9) The MinMax Detection application will continue until the number of Peak Information Sets acquired is equal to the pre-set number specified by SegmentCount in the INI file. This criterion may be easily changed according to the requirement.

In a MinMax Detection acquisition, the MinMax Segment Info structure entries that come from CompuScope Hardware are in RAW format. The GageMinMaxDtc sample project will convert these MinMax Segment Info structure entries to a readable format and save them in the MinMax Detection Queue.

The MinMax Detection Queue acts as a FIFO buffer. The driver saves new MinMax Segment Info structure entries at the bottom of the queue and the user application reads old MinMax Segment Info structure entries at the top of queue. Whenever the application retrieves a MinMax Segment Info structure entry, it removes that MinMax Segment Info structure entry from the queue, leaving space for the driver to save new incoming MinMax Segment Info structure entries.

If the application is not fast enough to remove MinMax Segment Info structure entries from the driver's MinMax Detection Queue, the queue will become full. When the queue is full, there is no more space for the driver to save new MinMax Segment Info structure entries, thus any new RAW MinMax Segment Info structure entries coming from CompuScope hardware will be discarded. When this happens, the driver will notify the application via the *hSWFifoFull* event.

There are actually three potential ways of missing waveform triggers and the associated MinMax Segment Info structure entries (Peak Information Sets). First, a trigger may be missed if it occurs while a previous waveform is being acquired. Second, if the Peak Information Sets within the FPGA are not transferred quickly enough, they may accumulate and exceed the capacity of the FPGA and some sets will be lost. Finally, as discussed in the previous paragraph, sets will be lost if the MinMax Detection Queue is not purged frequently enough by the controlling application. Any and all of these loss situations may be detected simply by checking the Trigger Number within the Peak Information Set. If the difference between the Trigger Numbers from successive Peak Information Sets is greater that 1, then the excess is exactly equal to the number of missed sets, regardless of the loss mechanism.

GageCsPrf

GageCsPrf is an advanced sample program that may be used to evaluate the repetitive capture performance of CompuScope hardware. The program makes a series of repetitive acquisitions using the specified settings and provides subsequent timing measurements. All timing measurements are done using the QueryPerformanceCounter Windows API timing function. The total time required to complete a single acquisition is provided along with its inverse, the Pulse Repeat Frequency (PRF). In addition, the timings of the separate operations that occur within one acquisition are provided. Repetitive acquisition sequences are repeated using different acquisition depths and timing results are provided for acquisition depths.

GageCsPrf is based upon the GageAcquire sample program and uses similar controls. Only controls that are different from those of GageAcquire are described here. See the GageAcquire documentation for the common controls.

Since GageCsPrf does not store acquired data, the *SaveFileName* key within the INI file is ignored. The timing results file name is specified by the *ResultsFile* key in the *PrfConfig* branch in the INI file. Also, the *TransferLength* key in the *Application* branch, along with the *SegmentSize* and *Depth* keys in the *Acquisition* branch, are ignored since the *Depth* is internally adjusted by *GageCsPrf*. Internally, the *Depth* is always selected as a power of two. The range of internally selected *Depth* values is bound by the smallest power of two that is greater than or equal to the value of the *StartDepth* key in the *PrfConfig* branch and the highest power of two that does not exceed the value of the *FinishDepth* key in the *PrfConfig* branch.

Results are stored in a tab delimitated text file. The first 5 lines describe the measurement configuration, such as the CompuScope model number and memory size, acquisition configuration and number of acquisitions in one repetitive capture sequence (loop count). These lines are followed by 7 columns of data. The first column (*depth*) specifies the size of the acquisition. The remaining columns are the result of the timing measurements. *Total time* is the time required by the complete acquisition sequence and *PRF* is its inverse, the repeat frequency of acquisitions. *Start time* is the time required for execution of the CsDo (ACTION_START) method. In this time the CompuScope system will perform all necessary operations to start an acquisition. *Busy time* is the time taken by the data acquisition itself. Please note that this time includes waiting for the trigger event. Consequently, if the trigger event is infrequent, the measured *Busy time* will be long. For measurement of the maximum PRF, trigger from a fast signal source, such as a sine wave with a frequency of 1 MHz or more. *Transfer time* and *transfer rate* describe data transfer from one channel. Please note that the data *transfer time* includes two components: a fixed transfer set-up overhead time and the actual data transfer duration, which is proportional to the data volume. As data volume increases, the importance of the overhead time diminishes. Consequently, the calculated aggregated *transfer rate* improves with the *Depth*.

Within GageCsPrf, the power-saving mode is enabled. This is fine for all CompuScope models except the CS82G and CS8500. For these models, power-saving mode will severely reduce repetitive capture performance (PRF). In order to get the best PRF from these models,

you must disable power-saving mode. This is done by adding 128 to the Mode value within the PRF.ini file. For single-channel mode, use "Mode=129" and for dual-channel mode, use "Mode=130".

By following the coding illustrated in GageCsPrf, a user can achieve the fastest possible repetitive capture performance from CompuScope hardware.

GageStream2Disk

Several Gage customers require their CompuScopes to acquire fast repetitive waveforms and store them as quickly as possible to hard drive. CompuScope hardware allows for fast PCI data transfer, extremely fast hard drive arrays are commercially available and MS Windows provides powerful multi-threading capability. Optimally combining these elements, however, to fully exploit their performance is not straightforward. In order to allow Gage customers to realize the best possible performance, Gage has created *GageStream2Disk*. This sample program uses a flexible, CompuScope driver-level engine and that allows the fastest possible repetitive CompuScope capture to hard drive.

A GageStream2Disk sample is now available as an Advanced Sample Program within CompuScope SDKs for C/C#, LabVIEW and MATLAB. The program will support waveform acquisition and storage sequence from any single CompuScope system (single card or Master/Slave system) constructed using any CompuScope model. User may even launch concurrent instances of Gage Stream2Disk in order to simultaneously acquire data from independent CompuScope systems. The sample programs are all relatively thin shells that call the GageStream2Disk engine, which operates from within the CompuScope drivers. Performance is therefore virtually identical from all three SDKs. The engine incorporates the following features to maximize overall throughput:

- Multi-threading for concurrent, parallel operations where possible – for example, storing previously acquired waveforms to hard drive while new waveforms are being acquired

- Pre-opening of target waveform files to avoid costly file open operations during acquisition sequence

- Block transfer size from CompuScope hardware optimized for most efficient usage of PC RAM under Windows

- Optimized storage file transfer size for fast storage rate and easy management under Windows

- Hardware interrupt usage to avoid costly repetitive polling operations

The GageStream2Disk engine uses a lot of system resources. While CPU processing power per se is not important, the host system should be equipped a fast PCI bus, a fast memory bus, a fast hard drive and as much PC RAM as possible. Specifically, the PC systems should be

equipped at least an amount of PC RAM that is equal to the number of active CompuScope channels multiplied by the 128 MB. Further, before running any GageStream2Disk sample program, the user should close all unnecessary Windows applications, since these will divert resources from GageStream2Disk.

From the GageStream2Disk sample program all SDKs, the Gage Stream2Disk operation method is the same. Before activating the GageStream2Disk engine, the sample program must first initialize the drivers and commit CompuScope configuration settings, as usual. Next, the program activates the GageStream2Disk engine and initializes it with acquisition settings (e.g. number of waveforms to be acquired, data of interest that is to be transferred). The initialization also validates input parameter and allocates buffers required for GageStream2Disk acquisitions. Finally, initialization creates all Windows folders and pre-opens all SIG files that will be required for the acquisitions. Next, the program starts the GageStream2Disk acquisition sequence, so that the CompuScope hardware begins acquiring waveforms and storing them to files. Normally, the process continues until the requested number of waveforms has been acquired and stored to hard drive. From C, all control of the GageStream2Disk engine is invoked using the CsExpertCall() API call with different u32ActionId constants. The user may issue the following commands and queries:

1. COMMAND INITIALIZE (implemented by the CsExpertCall() with u32ActionId = EXFN_DISK_STREAM_INITIALIZE): Activate and initialize the GageStream2Disk engine. Allocate required resources and validates input GageStream2Disk parameters. Create target folders and pre-open target SIG files.

2. COMMAND START (implemented by the CsExpertCall() with u32ActionId = EXFN_DISK_STREAM_START): Start the GageStream2Disk acquisition sequence, which will normally terminate only once all waveforms are acquired and stored to hard drive files.

3. COMMAND STOP (implemented by the CsExpertCall() with u32ActionId = EXFN_DISK_STREAM_STOP): Stop the GageStream2Disk acquisition sequence after completing any acquisition that is in progress and storing all resultant waveform files to hard drive.

4. COMMAND CLOSE (implemented by the CsExpertCall() with u32ActionId = EXFN_DISK_STREAM_CLOSE): De-activate the GageStream2Disk engine. Free all allocated system resources, including active CompuScope handles.

5. QUERY STATUS (implemented by the CsExpertCall() with u32ActionId = EXFN_DISK_STREAM_STATUS): Returns status that indicates if GageStream2Disk acquisition sequence is finished.

6. QUERY # ACQUISITIONS (implemented by the CsExpertCall() with u32ActionId = EXFN_DISK_STREAM_ACQ_COUNT): Return the number of acquisitions that have been completed since the GageStream2Disk acquisition sequence has started.

7. QUERY # FILES (implemented by the CsExpertCall() with u32ActionId = EXFN_DISK_STREAM_WRITE_COUNT): Return the number of files that have been stored since the GageStream2Disk acquisition sequence has started.

As discussed, adjustment of configuration settings (e.g. sampling rate, input range) for GageStream2Disk is done in a similar fashion as for the standard SDK sample programs. However, there are some additional setup variables that have been added for GageStream2Disk. These variables are described below only for Stream2Disk.INI configuration file for the C Sample Stream2Disk program but corresponding variables exist for the other SDKs.

Within Stream2Disk.INI, the *AcqCount* variable determines the total number of acquisitions that will be performed by GageStream2Disk. Only completion of this number of acquisition will terminate the GageStream2Disk engine, unless it encounters an error or is intentionally stopped or aborted. The *Channels* variable selects the channels whose data will be stored to hard drive files. For instance, if a user is only interested in the even channel on an 8-channel CompuScope card, he can set *Channels=2,4,6,8*.

As in the standard SDK samples, GageStream2Disk allows adjustment of the *StartPosition* and *TransferLength*, which determine how much waveform data are to be downloaded from which point in the waveforms. In addition, GageStream2Disk allows the user to select how many records to download and from which starting record for Multiple Record acquisitions. For instance, if the user does Multiple Record acquisitions acquiring 120 Records, he can choose to download and store only 10 records starting from record #40 by setting *RecordStart=40* and *RecordCount=10*.

Finally, there are three separate Trigger TimeOut Settings in GageStream2Disk, all of which are expressed in milliseconds and all of which are disabled by a value of -1. The *TriggerTimeOut* in the *Acquisition* key within the INI file, which is standard for all C SDK programs, is a hardware Trigger TimeOut. There is a high–speed counter on the CompuScope hardware that forces a trigger event if the counter value corresponding to the Trigger TimeOut is exceeded. The advantage of this hardware Trigger TimeOut is that has sub-microsecond precision and is also very accurate and deterministic, as compared to a software Trigger TimeOut. The disadvantage, however, is that the hardware counter will eventually roll over, since it has finite width.

The *TriggerTimeOut* in the *AppConfig* key is a software Trigger TimeOut. Its use is necessary when the required Trigger TimeOut exceeds the rollover limit of the hardware Trigger TimeOut (typically over an hour to several hours, depending upon the CompuScope model and sampling rate). When using this software Trigger TimeOut, the user should disable the hardware Trigger TimeOut by setting it to -1.

The final timeout, *StatusTimeout*, is an intentional software latency introduced so that the QUERY operation listed above (QUERY STATUS, QUERY # ACQUISITIONS and QUERY # FILES) are not called in a loop at a rate sufficient to compromise the speed of the GageStream2Disk data transfer and storage operations. The default setting of 500 milliseconds ensures that the driver will not poll the status of GageStream2Disk more than

twice a second, which will not significantly affect operation and is usually a sufficient status update rate for the user.

Gage Stream2Disk stores waveforms acquired by CompuScope hardware in GageScope SIG files, which are binary data files with a 512 byte information header. For easy management under Windows, Gage has limited the maximum SIG files size created by GageStream2Disk to a maximum of 256 Megabytes. (This number may be increased in future, but the principles discussed below will still apply). In order to optimize system memory usage, GageStream2Disk internally limits the size of a single block data transfer from the CompuScope hardware to a maximum of 128 Megabytes per active channel. Consequently, the user should have a bare minimum of 128 Megabytes of PC RAM per active channel. Of course, this bare minimum does not leave any room for the O/S or for applications, so that significantly more than this minimum is highly recommended for optional operation.

GageStream2Disk stores all acquired waveform data within GageScope SIG files that reside within a folder structure designed for easy management. The folder structure is created upon initialization of the GageStream2Disk engine. First, all files are placed within a Master folder local to the GageStream2Disk application, whose name is specified by the *FolderName* variable (Default= "Signal Files"). Next, folders within the Master folder are created for data from different active CompuScope channels. These folders are given names in the from YYYY-MM-DD_HH-MM-SS_CHANXX, where YYYY-MM-DD and HH-MM-SS are the system date and time when the folders are created upon initialization and XX is the CompuScope channel number. Within these folders, a series of sequential folders called *Folder.001, Folder.002, Folder.003, etc.* are created. The contents of each of these folders is limited to a maximum of 16,000 SIG files and initialization creates a sufficient number of folders to accommodate the entire GageStream2Disk acquisition sequence. Within these folders, SIG files called *File-00000.SIG, File-00001.SIG, etc.* are created and opened so that they are ready to accept data from the subsequent GageStream2Disk acquisition sequence. We describe below the three most common GageStream2Disk use cases and describe how the resultant waveform data are stored in each case.

The most common use case is acquisition of many relatively short waveforms (a few hundred Samples ~ 100 MegaSamples) by CompuScope hardware in Single Record Mode. In this case, the GageStream2Disk engine will generally pack many single waveforms into a single Gage SIG file. All but the last SIG file will contain the same number of records. The last SIG file will most likely contain a smaller number of left-over records. As an example, consider the user who configures GageStream2Disk to acquire twenty thousand waveforms of 65536 Samples each in Single Record Mode from a 14-bit CompuScope, where each Sample will occupy 2 data bytes. In this case, GageStream2Disk will fill each 256 MB SIG File with 2048 records, since $(256 \times 1024 \times 1024 \text{ MB})/(65536 \text{ Samples/waveform})/(2 \text{ Bytes/Sample}) = 2048$ waveforms. GageStream2Disk would create nine such SIG files and the final tenth SIG file would contain only the remaining $20{,}000 - 9 \times 2048 = 1568$ records.

The second case is acquisition of relatively long waveforms (>256 Megabytes) by CompuScope hardware in Single Record Mode. Since the waveform size exceeds the maximum file size, the GageStream2Disk engine will divide the waveform into multiple contiguous SIG files. For example, a user might want to perform sixteen acquisitions of 1,100,000,000 Samples each using a 12-bit CompuScope. Because each 12-bit Sample

occupies 2 data bytes, GageStream2Disk will only allow storage of 128 MegaSamples per SIG file. In this case, the 1. 1 Billion Samples of data from a single acquisition will be divided across nine SIG files. The first eight files will contain 128 MegaSamples, and the ninth file will contain the remaining $1,100,000,000 - 8 \times 128 \times 1024 \times 1024 = 26,258,176$ Samples.

These nine SIG files will contain completely contiguous data so that, for example, the first Sample in the third SIG file was acquired directly after the last Sample in the second SIG file. The relationship amongst the nine contiguous SIG files is indicated by the Start Address contained within the SIG files. As is often the case, let us assume that the data requested for storage by the user began at the Trigger Address, which is always presented by the software as Trigger Address 0. In this case, the first SIG file will contain a Start Address of 0, which indicates that the first Sample within the file is the Trigger Point. The second SIG file, however, will contain a Start Address of $-128 \times 1024 \times 1024 = -134,217,728$, which correctly indicates that the Start Address occurred 134,217,728 before the first Sample in the second file. The Start Address will decrease by 134,217,728 Samples for each successive SIG file in the nine file set.

Since the user requested sixteen acquisitions of 1. 1 Billion Samples each, each of the 12 acquisitions will create its own set of nine contiguous files. While each set of nine files represents a continuous acquisition within no breaks between files, between each file set there will be a time-gap as acquisition ceases while data are downloaded from the CompuScope hardware and stored to files.

In Multiple Record Mode, GageStream2Disk stores files in a similar fashion as in Single Record Mode. GageStream2Disk always tries to store all records from a single Multiple Record acquisition within a single SIG file. However, GageStream2Disk never mixes records from different Multiple Record acquisitions within the same SIG file. Division of data from a single Multiple Record is avoided, if possible. If the total Multiple Record data volume exceeds 256 MB, GageStream2Disk will distribute records across multiple SIG files, as in the first case above, while not splitting records if possible. The rare extreme Multiple Record case occurs when the user acquires very long Multiple Records. In this case, GageStream2Disk is forced to split the records across multiple files. For example, if a user chooses to acquire ten Multiple Records of 600,000,000 Samples each from an 8-bit CompuScope, GageStream2Disk will create ten contiguous SIG files sets – one set for each record. Each set will consist of three SIG Files, the first two of size 256 MB and the last one containing the roughly 100 MB of remaining left-over data.

The GageScope SIG files created by Gage Stream2Disk contain an informative 512 Byte header, followed by the CompuScope waveform data in binary. Gage provides the C CsConvertFromSigHeader() method, which decodes the header (from a buffer into which a SIG file header has transferred) and fills a structure with relevant header variables. These variables may then be used to select and interpret subsequent SIG file waveform data. Users can also use the CsConvertToSigHeader() method to create their own SIG file headers. Corresponding methods exist in the other SDKs.

Key elements that retard GageStream2Disk's maximum achievable repetitive acquisition rate are the time required to download data from the CompuScope hardware to PC RAM and the

time to transfer data from PC RAM to the target hard drive. In almost all cases, it is the latter element that is the primary bottleneck to data throughput. Maximizing the storage rate to drive requires that certain optimization steps.

First of all, users should be wary of high storage rate claims by hard drive manufacturers. Usually, these claims refer to the data storage rate that may be maintained by the drive during burst storage. Most drive devices are equipped with a limited amount of cache memory. Usually, manufacturers' throughput claims refer only to burst storage volumes that do not exceed this cache size, which is typically a few MB While very high storage rates may be realized into this disk cache memory, as soon as the cache is full, the rate drops dramatically, since the cache contents must be more slowly transferred to the actual drive media. What the Gage user really requires is this slower *sustained* storage rate, which is usually not provided by the hard drive manufacturer.

For best data throughput, the users should dedicate a specific drive as a storage target and should not store other files on this drive. Most importantly, the user should make sure not to store GageStream2Disk data on the same drive upon which the Windows Operating System or Page File is installed. The constant read/write housekeeping operations that Windows performs in normal course will seriously compromise the storage rate of Gage Stream2Disk.

For best Disk performance, the user should obtain hard disk device drivers from the hard drive manufacturer, rather than using the generic drivers included in Windows, which typically will not provide the fastest throughput for your manufacturer's drive. Finally, disk throughput may be improved dramatically by configuring multiple hard drives in a RAID disk array. A RAID disk array may realize sustained data throughput rates of 100 MB/s and more.

GageASTransfer, GageEvents, GageCallback

These sample programs illustrate advanced synchronization techniques for multi-threaded applications. These techniques are essential to creating a complex application for real time data analysis or for operating multiple inter-related instruments. This is because these techniques allow a multi-threaded application to perform other tasks while CompuScope hardware is busy acquiring or transferring data without the usual need to poll its status. Without the need for polling, data acquisition and transfer do not tax the CPU, leaving it free to perform other operations. Nevertheless, these techniques add significant complexity to the overall application design, making it prone to errors such as thread deadlock. Consequently, usage of these techniques should not be considered unless they are truly required.

As the name suggests, GageASTransfer illustrates asynchronous data transfer. When called, the standard CsTransfer method does not exit and return until the requested data transfer operation is complete and all data have been transferred into the target buffer. By contrast, when CsTransferAS is called, it returns immediately after initiating the data transfer, which is then left to finish in the background. While data are being transferred, the controlling application may do something else, even though the data transfer is not yet complete.

Completion of the data transfer is signalled by the "end of the transfer" event. Progress of the data transfer may be checked by CsTransferASResult method. GageASTransfer is a non-multi-threaded C application that polls the "end-of-transfer" event while checking the transfer status every 100 milliseconds.

GageEvents is a multi-threaded sample program that illustrates usage of the notification events that can be assigned to specific operation of the CompuScope, allowing synchronization between different threads of execution. GageEvents uses the "end-of-busy" and "end-of-transfer" event notifications to trigger appropriate operations. In parallel, GageEvents processes older waveform data to determine the minimum and maximum points within the waveform. The handling of events that is illustrated within GageEvents is the recommended method of synchronization in a multi-threaded C application.

Some environments, such as Visual Basic and LabWindows/CVI, do not allow the programmer to create multi-threaded programs directly. These environments, however, do provide a functionality called "Callbacks", which allows the programmer to associate a callback function with notification of an event. The thread associated with the callback is then launched within the CompuScope driver. Use of callbacks has limitations. For instance, only one callback function may be executed at a time. Synchronization using callbacks is illustrated by the GageCallback sample program.

Advanced Multiple Record Sample program

All modern CompuScope models allow for the acquisition of pre-trigger data in Multiple Record mode. The user is able to set up a Multiple Record Segment Size that is larger than the post-trigger depth so that pre-trigger data may be accumulated. In addition, trigger Time Stamp values are logged and may be retrieved for each Multiple Record.

The improved Multiple Record features complicate the storage of Multiple Record data in CompuScope on-board memory. For example, the trigger position may be located anywhere within the Multiple Record segment memory and a memory footer exists to store information such as Time-Stamp data. In order to manage these complications, standard GaGe sample programs in all CompuScope SDKs are designed to download Multiple Records one-at-a-time in a software loop. The CsTransfer() method internally manages on-board storage complications and extracts the data for a single Multiple Record segment. While this technique is easy to use, aggregate PCI data transfer rate is compromised, since software overhead of initiating separate transfers for each Multiple Record is introduced. In applications where rapid repetitive Multiple Record acquisitions are required, this software overhead can limit performance. In order to provide the fastest possible download of Multiple Record data, GaGe has provided the AdvMultipleRecord sample program for the C/C# SDK.

The AdvMultipleRecord program requires the same input files and provides the same output files as GageMultipleRecord – the standard Multiple Record C SDK sample program. That is, the CompuScope configuration settings are read from an INI file and the output files are individual Multiple Record .DAT files. The difference is that the AdvMultipleRecord internally downloads all Multiple Record data in one PCI data transfer, rather than using a

separate PCI transfer for each Multiple Record segment. Only after all data transfer is complete are the records parsed by AdvMultipleRecord for storage in individual Multiple Record DAT files.

The coding functionality within AdvMultipleRecord and GageMultipleRecord is identical up until the subroutine call to SaveMulRecRawData() within AdvMultipleRecord. The SaveMulRecRawData() subroutine calls the CsExpertCall()subroutine using an ActionId that is equal to the constant EXFN_RAWMULREC_TRANSFER. This call to CsExpertCall() transfers all segment data for all active channels from a Multiple Record acquisition to the specified memory buffer in a single PCI data transfer operation. The buffer must have been previously allocated using the GetMulRecRawDataBufferSize() subroutine, which also calls CsExpertCall().

Once all the raw Multiple Record data have been transferred to an internal buffer, the records are parsed within a loop whose index cycles through the number of Multiple Records. Within the loop, the CompuScope API method or function called CsRetrieveChannelFromRawBuffer() extracts the data for a single record file, whose attributes are specified by variables within the InData structure. The data for the extracted record are returned within the buffer pointed to by pBuffer. Also returned is Time-Stamp data along with other information in the OutData structure. The Multiple Record waveform data are optionally converted into Volts and then are stored within a DAT file with a corresponding header.

The user may easily modify the AdvMultipleRecord program so that the complete raw data buffer data is stored to a binary file. This way, the user does not need to waste time parsing data during repetitive Multiple Record acquisitions. After the acquisitions are all finished, the user can reload the raw data files and extract the data of interest without sacrificing measurement time.

There is also a LabVIEW sample program called GageAdvMulRec.VI in the CompuScope LabVIEW SDK. The VI front panel and functionality is similar to that of GageMultipleRecord.VI. The main difference is that GageAdvMulRec.VI downloads the raw data buffer for all Multiple Records in one transfer. This raw buffer is then parsed by a special VI called CsLv_ParseAdvMulRecData.VI into a 2-dimensional output array with dimensions that are equal to the Number of Records and the Number of Samples per Record. The user may modify the main VI to perform the parsing in post-processing after the repetitive acquisition experiment is complete.

AdvMulRecEx

Historically, Gage CompuScopes have allowed for the collection of a large and non-predetermined amount of pre-trigger data in Multiple Record Mode. In order to achieve this capability, CompuScopes have configured their acquisition memory as multiple circular buffers of size *SegmentSize* in Multiple Record mode. The disadvantage of configuring each Segment memory as a circular buffer is that this complexity practically required Multiple Records to be downloaded to PC RAM one-at-a time with a separate transfer for each Multiple Record, rather than downloading all Multiple Records in one data transfer. This one-at-a-time record download does not cause a significant problem for long Records (more

than 10,000 Samples or so). However, for shorter record lengths, the overhead of setting up multiple PCI transfers (one per record) can seriously reduce the aggregate data download rate.

In order to significantly improve the download rate for Multiple Record acquisitions with short records, Gage has implemented a separate Multiple Record operation mode on some CompuScope models called *Rectangular Multiple Record Mode*. Instead of being configured as multiple circular buffers, the memory is configured in a rectangular fashion. Memory contents consist of a rectangular array in which each row contains a single waveform record and each column corresponds to a specific Sample within each waveform record. (In fact, the architecture is a little more complex since Trigger Timestamp Data are stored a the end of each waveform.) With this architecture, all records from a Multiple Record Mode may be downloaded in one transfer and the resultant data buffer may be easily parsed to extract single records or may be stored directly for future analysis. A diagram of the on-board memory architecture in rectangular mode is shown below for the acquisition of M records of N Samples each.

Record #1 Sample #1	Record #1 Sample #2	Record #1 Sample #3	..	Record #1 Sample #N-1	Record #1 Sample #N	TimeStamp #1
Record #2 Sample #1	Record #2 Sample #2	Record #2 Sample #3	..	Record #2 Sample #N-1	Record #2 Sample #N	TimeStamp #2
Record #3 Sample #1	Record #3 Sample #2	Record #3 Sample #3	..	Record #3 Sample #N-1	Record #3 Sample #N	TimeStamp #3
…	…	…	..	…	…	…
Record # M-1 Sample #1	Record # M-1 Sample #2	Record # M-1 Sample #3	..	Record # M-1 Sample #N-1	Record # M-1 Sample #N	TimeStamp # M-1
Record #N Sample #1	Record #M Sample #2	Record #N Sample #3	..	Record #M Sample #N-1	Record #M Sample #N	TimeStamp #M

The maximum number of records that may be acquired in Multiple Record Mode is:

Max Number of Records ≈ Max Acquisition Memory per channel / (#Samples per Record)

The equation is not exactly correct because space is required for some inter-record information, such as Timestamp values. The amount of inter-record information is dependent upon the CompuScope model.

Acquisition using Rectangular Multiple Record Mode is illustrated within the AdvMulRecEx sample program within each CompuScope SDK. The configuration of Rectangular Multiple Record Mode is the same as for Circular Multiple Record Mode with some difference in the interpretation of the input parameters. As usual, the Post Trigger *Depth* specifies how many Samples will be acquired after each trigger. The *Segment Size*, however, determines the amount of waveform data that will always be acquired for each Record. Consequently, a pre-trigger data volume of (*SegmentSize – Depth*) will be acquired for each record – even if the

CompuScope must ignore triggers in order to acquire this pre-trigger data volume. Another important element is that there is a maximum allowed amount of pre-trigger data, which is equal to 128 kiloSamples/(Number of Active Channels).

Once a Rectangular Multiple Record Mode acquisition is completed, resulting data are downloaded using the CsTransferEx() API method (or the corresponding method in other SDKs). The most important difference between CsTransferEx() and the older CsTransfer() method is that CsTransferEx() is able to download many Multiple Records in a single data transfer. A second important different is that, unlike CsTransfer(), CsTransferEx() is able to download Multiple Record data from more than one CompuScope channel in a single data transfer.

The input parameters to the CsTransferEx function, whose actual variable names are slightly different for each SDK, are as follows:

Channel number – The channel from which record data are to be downloaded A Flag exists that allows download from all active channels.
Start Record – The first record to be downloaded
Number of Records – The number of records to be downloaded, starting from *Start Record*
Start Sample – The first Sample within each record to be downloaded
Number of Samples – The number of Samples to be downloaded, starting from *Start Sample*, from within each record

While CsTransferEx() allows download of huge volumes of data in a single transfer, the user should be careful not download too much data at one time. Doing this may cause the operating system to start using part of the hard drive as PC RAM, which may seriously retard data throughput. Generally, the user should avoid downloading more than 100 Megabytes of data in a single data transfer. Smaller data volumes may be downloaded in sequential transfers, between which each volume is dispatched to hard drive storage or data analysis algorithms.

Special CompuScope API function for usage with eXpert firmware

CsExpertCall

The **CsExpertCall** function is required for control of certain eXpert firmware features in CompuScope cards.

int32 CsExpertCall(CSHANDLE hCsHandle, VOID *pFunctionParams)

Parameters

       hCsHandle   hHandle of the CompuScope system
       pFunctionParams   The pointer to Function Params structure

Return values

       CS_SUCCESS indicates success.

Remarks

The Function Params structure will be different depending on the action to be performed, but they all have the same form:

```
typedef struct
{
      struct
      {
            uInt32        u32Size;
            uInt32        u32ActionId;
            ...
      } in;

      struct
      {
            .....
      } out;
}
```

FUNCTION PARAM STRUCTURES AND FUNCTION ID TO BE USED WITH CsExpertCall()

EXFN_CREATEMINMAXQUEUE

Call CsExpertCall() with the function Id EXFN_CREATEMINMAXQUEUE to create a MinMax Detection Queue within the Windows Kernel for usage by the driver.

```
typedef struct _CSCREATEMINMAXQUEUE
{
        struct
        {
                uInt32          u32Size;
                uInt32          u32ActionId;
                uInt32          u32QueueSize;
                uInt16          u16DetectorResetMode;

                uInt16          u16TsResetMode;
        } in;

        struct
        {
                HANDLE          *hQueueEvent;
                HANDLE          *hErrorEvent;
                HANDLE          *hSwFifoFullEvent;
        } out;

} CSCREATEMINMAXQUEUE, *PCSCREATEMINMAXQUEUE;
```

Parameters

u32Size	Size of this structure
u32ActionId	The function Id. Must be EXFN_CREATEMINMAXQUEUE
u32QueueSize	Number of SegmentInfo in the Driver MinMax Detection Queue. (e.g. a value of 50 indicates that the driver MinMax Detection Queue can hold up to 50 MinMax Segment Info structure entries before the *hSwFifoFullEvent* event gets signalled).

u16DetectorResetMode	MinMax detector reset mode.
	  0: Reset on Trigger. Peaks detected only in post-trigger data
	  1: Reset on Start of Segment

u16TsResetMode	MinMax Time stamp reset mode
	  0: Reset on Start Acquisition
	  1: Reset on Start of Segment

hDataAvailableEvent	Event for data available in MinMax Detection Queue
hErrorEvent;	Error event
hSwFifoFullEvent;	Event for MinMax Detection Queue full

Remarks

The queue must be created before the Peak Detection image is loaded onto the CompuScope FPGA.

Upon return from this function, the application may receive one or more of the following 3 events:

hDataAvailableEvent
This event will be signalled whenever there is a SegmentInfo item in the MinMax Detection Queue.
As soon as the event is signalled, the application should call CsExpertCall with the function Id EXFN_GETSEGMENTINFO to retrieve the SegmentInfo from the driver.

hErrorEvent
This event will be signalled when a fatal error occurs in the current acquisition.
Once the error is signalled, the current acquisition will be automatically aborted.

hSwFifoFullEvent
The event will be signalled when the MinMax Detection Queue is full.
When the driver MinMaxQueue FIFO is full, any MinMax Segment Info structure entries that come from hardware will be discarded. The current acquisition will continue.
The event remains in a signalled state, unless the application resets it via a call to CsExpertCall with the function Id EXFN_CLEARERRORMINMAXQUEUE.

EXFN_DESTROYMINMAXQUEUE

Call CsExpertCall() with the function Id EXFN_DESTROYMINMAXQUEUE to destroy the driver's MinMax Detection Queue created by EXFN_CREATEMINMAXQUEUE.

```
typedef struct _CSDESTROYMINMAXQUEUE
{
        struct
        {
              uInt32        u32Size;
              uInt32        u32ActionId;
        } in;

} CSDESTROYMINMAXQUEUE, *PCSDESTROYMINMAXQUEUE;
```

Parameters

u32Size	Size of this structure
u32ActionId	The function Id. Must be EXFN_DESTROYMINMAXQUEUE

Remarks
The function fails if it is called when the Peak Detection image is loaded onto the CompuScope FPGA.
The standard image CS_MODE_USER0 must be loaded onto the CompuScope FPGA before destroying the queue.

EXFN_GETSEGMENTINFO

Call CsExpertCall() with the function Id EXFN_GETSEGMENTINFO to retrieve a MinMax Segment Info structure from driver's MinMax Detection Queue, which was created by EXFN_CREATEMINMAXQUEUE.

```
typedef struct _CSPARAMS_GETSEGMENTINFO
{
      struct
      {
            uInt32          u32Size;
            uInt32          u32ActionId;
            uInt32          u32BufferSize;
      } in;

      struct
      {
            MINMAXSEGMENT_INFO       *pBuffer;
      } out;

} CSPARAMS_GESEGMENTINFO, *PCSPARAMS_GESEGMENTINFO;


typedef struct _MINMAXSEGMENT_INFO
{
      uInt32              u32Size;                //size of this structure
      uInt32              u32NumberOfChannels; //Number of channels
      TRIGGERTIMEINFO     TrigTimeInfo;        //The Triggering information
      MINMAXCHANNEL_INFO  MinMaxChanInfo[1];   //The MinMaxInfo for each
                                               channel

}MINMAXSEGMENT_INFO, *PMINMAXSEGMENT_INFO;


typedef struct _TRIGGERTIMEINFO
{
      int64       i64TriggerTimeStamp; //The Time Stamp counter value for
                                       the Trigger Event
      uInt32      u32TriggerNumber;    //The Trigger Number

}TRIGGERTIMEINFO, *PTRIGGERTIMEINFO;


typedef struct _MINMAXCHANNEL_INFO
{
      int16       i16MaxVal;           //Max ADC code within the waveform
      int16       i16MinVal;           //Min ADC code within the waveform
      int64       i64MaxPosition;      //Time Stamp counter value for the
                                       max position
      int64       i64MinPosition;      //Time Stamp counter value for the
                                       min position

} MINMAXCHANNEL_INFO, *PMINMAXCHANNEL_INFO;
```

Parameters

u32Size	Size of this structure
u32ActionId	The function Id. Must be `EXFN_GETSEGMENTINFO`
u32BufferSize	Size of the buffer in bytes.
pBuffer	pointer to the buffer receiving MinMaxSegment info.

Remarks
The MinMaxSegmentInfo size will be different depending on the mode (Dual or Single channel) and the number of cards in Master/Slave system. The *u32BufferSize* must be equal to at least the MinMaxSegmentInfo size.

EXFN_CLEARERRORMINMAXQUEUE

Call CsExpertCall() with the function Id EXFN_CLEARERRORMINMAXQUEUE to reset the event *hSwFifoFullEvent.*

```
typedef struct _CSPARAMS_CLEARERRORMINMAXQUEUE
{
      struct
      {
            uInt32        u32Size;
            uInt32        u32ActionId;
      } in;

} CLEARERRORMINMAXQUEUE, *PCLEARERRORMINMAXQUEUE;
```

Parameters
> *u32Size*   Size of this structure
> *u32ActionId*   The function Id. Must be `EXFN_GETSEGMENTINFO`

Remarks
This function will reset the *hSwFifoFullEvent.* If the driver's MinMax Detection Queue remains full, however, this event will get signalled again.