WAVECREST
Be certain of the signal you send.

Production Application Programming Interface (PAPI)

Reference Manual

200212-04 REV A

This page intentionally left blank.

WAVECREST CORPORATION continually engages in research related to product improvement. New material, production methods, and design refinements are introduced into existing products without notice as a routine expression of that philosophy. For this reason, any current *WAVECREST* product may differ in some respect from its published description but will always equal or exceed the original design specifications unless otherwise stated.

WAVECREST CORPORATION
7626 GOLDEN TRIANGLE DRIVE
EDEN PRAIRIE, MINNESOTA 55344
(952) 831-0030
(800) 733-7128
WWW.WAVECREST.COM

Table of Contents

(cont'd)

This page intentionally left blank.

Purpose and Organization of this Manual

The *WAVECREST* SIA-3000 and *GigaView™* software have the ability to run automated tests or control the SIA-3000 remotely through a workstation or PC. This manual covers the Production Application Programming Interface (PAPI) method.

Section 1 introduces the user to the elements of an application utilizing the *WAVECREST* PAPI software. This section will aid in getting PAPI set up and ready to compile into applications. There is also a simple example demonstrating the basic PAPI commands and concepts that can be applied to any measurements with any SIA-3000 tool.

Section 2 provides information, in greater detail, pertaining to the basic measurement functions that comprise PAPI. This section should help the developer gain a basic understanding of the measurement commands in PAPI and serve as a reference for the variety of data structures used to pass information to and from the SIA-3000.

Section 3 is a function reference for any remaining functions not addressed in Section 2. Functions for setting up patterns, calibration and making low-level GPIB calls are among the calls listed in this section. Most functions addressed in Section 3 are for advanced PAPI usage or for making low-level GPIB calls. Some mandatory functions for getting started and basic PAPI usage are `COMM_InitDev()` and `COMM_CloseDev()` in Section 3-1 as well as `FCNL_PulsFind()` in Section 3-2. Section 3-7 addresses the definition of groups for defining advanced measurement sequences. It is not necessary to utilize the group functionality for basic PAPI applications.

The best approach for the beginning PAPI developer is to review Section 1, followed by Sections 2-1 and 2-2. Once this is complete, go through the following process when referring to the PAPI manual:

- Choose an SIA-3000 tool and the desired parameters/results
- Refer to the appropriate sub-section of Section 2 for the selected tool (i.e. Histogram – Section 2-25).
- Review the input and output parameters for the structure, the functions that apply to that tool and the simple example. Refer to Sections 2-3 through 2-14 for information on interpreting any sub-structures within the data structure for the tool.
- Refer to the application in Section 1-7, replacing any tool specific calls and structures with your own
- Refer to Section 3 and the Appendices as needed for explanations of other functions

Appendix A lists error codes.

Appendix B shows what the sample program in Chapter 1 might look like if written as a Visual Basic subroutine.

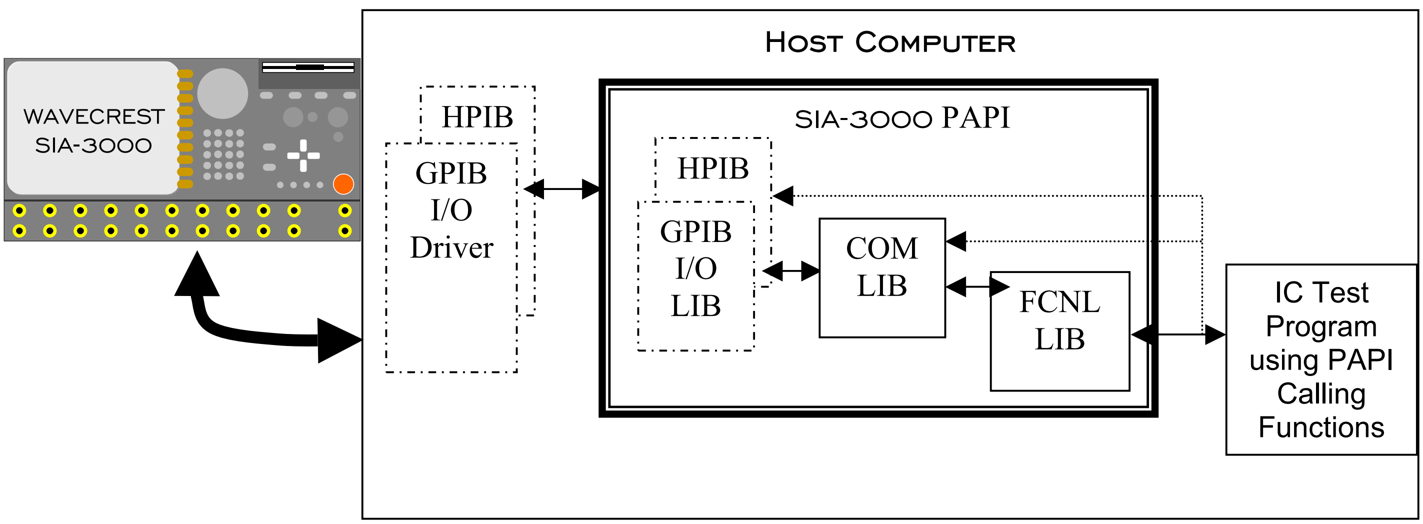
Appendix C lists changes to the measurement window structures and sub-structures for all supported revisions of PAPI.

This page intentionally left blank.

WAVECREST has implemented the Production Applications Programming Interface (PAPI) to provide direct access to the algorithms available in the SIA-3000™. This Production API allows programmers to quickly integrate the functionality available in the SIA-3000 with their own applications. Many tedious tasks such as GPIB interfacing and memory management are eliminated. A layered approach is utilized which provides access to all the statistics and plot data available. This API is cross platform. Versions for Microsoft® Windows as well as many UNIX platforms are available. The PAPI also provides routines to utilize configurations established with the SIA-3000 software to streamline the transition from laboratory characterization to production floor. The PAPI is compatible with SIA-3000 *GigaView*™ software.

1-1 ELEMENTS of an APPLICATION Using the *WAVECREST* PRODUCTION API

A typical application using the *WAVECREST* PAPI can be seen in the following figure.



The *WAVECREST* PAPI is divided into three layers. The I/O layer provides a hardware abstraction layer to isolate the higher-level algorithms from the hardware itself. Although GPIB and HPIB are the only physical medium supported at this time, this abstraction layer provides templates for custom I/O routines.

The communication layer is an intermediate layer between the functional layer and the hardware abstraction layer and provides functions such as polling and data requests. The FCNL (functional) layer provides high-level functionality such as implementing the standard windows contained in the SIA-3000 system, pulse-find and interpreting plot arrays.

1-2 FUNCTION CALL STRUCTURES

As function calls are listed throughout the manual, they will appear in the following format:

Function Name

`long __stdcall FCNL_PtnName ( char sPtnName[], char *name )`

This function is used to assist an application load the pattern file into the required measurement structure. This function is included to assist when programming in Microsoft Visual Basic. When programming in C, the data array can be accessed directly.

Function Description

INPUTS

Input variables used

sPtnName - Location where pattern name will be updated. Memory needs to be allocated by the caller.

*name - Name of pattern to load into measurement structure.

OUTPUTS

Output variables used

Returns SIA_SUCCESS if operation is successful or a negative value to indicate error.

FCNL_PtnName (sPtnName[], *k28.5_pttn) //this function will change the pattern loaded //to the pattern pointed to by the pointer //*k28.5_pttn*. *k28.5_pttn* is user definable.

Sample code

Sample code comments

A few helpful notes:

NOTE: `__stdcall` *and* `DllCall` *are part of the function definitions in the header file but can essentially be ignored. They are utilized to provide options when building and using DLLs on Microsoft® Windows. They are implemented to allow the same header file to be used for building the DLL and importing the DLL, ensuring consistent declarations.*

NOTE: Many of the measurement window structures contain padding fields. These fields are usually called `lPad1`, `lPad2`, *... or* `lPadLoc1`, `lPadLoc2`, *... and are used to insure that variables are placed in the same absolute locations within the structure regardless of compiler padding which varies from system to system. These fields are only used to take up space, and can be safely ignored.*

1-3 FILES INCLUDED IN THE *WAVECREST* PRODUCTION API

The *WAVECREST* PAPI consists of ten header files and associated libraries. The header files are platform independent while the libraries are platform dependent. Libraries for Microsoft® Windows applications are provided in the form of run-time Dynamic Link Libraries while Libraries for UNIX applications are provided in both static and shared forms.

In addition to the header and library files, sample application source code and *makefiles* are also provided. There is also a directory containing various dataCOM patterns. Files are located on the CDROM in the following directory locations:

Name	Size	Type
hp10x		File Folder
hp9x		File Folder
patns		File Folder
solaris2		File Folder
sunos		File Folder
win32		File Folder
ReadMe.txt	5 KB	Text Document
sample.c	8 KB	WordPad Document
WCcomm.h	6 KB	H File
WCcust1.h	2 KB	H File
WCcust2.h	2 KB	H File
WCcust3.h	2 KB	H File
WCfcDefs.h	9 KB	H File
WCfcnl.h	7 KB	H File
WCfcStrc.h	38 KB	H File
WCgpib.h	2 KB	H File
WChpib.h	1 KB	H File
WCio.h	3 KB	H File

1-4 *WAVECREST* PRODUCTION API INSTALLATION

To install the *WAVECREST* PAPI, first create a target directory on the host system. Copy the files from the *WAVECREST* PAPI CDROM contained in the base directory as well as those from the particular platform directory to the newly created target directory.

1-5 BUILDING THE SAMPLE APPLICATION

Before attempting to build the sample application, the supported compiler should be installed and properly configured. This may include modifying the PATH environment variable so that the compiler's executable can be launched from a command line. It may also involve setting INCLUDE and LIB environment variables so that the standard include files and libraries may be located by the compiler. Consult the compiler documentation for further information.

To build the sample application on UNIX, execute the following from a command prompt:

```
make
```

To build the sample application on Microsoft® Windows, execute the following from a command prompt:

```
nmake
```

1-6 EXECUTING THE SAMPLE APPLICATION

Before attempting to execute the sample application, the supported GPIB interface card must be installed and properly configured on the host workstation. (Consult the interface card manufacturer's documentation for further information.) The *WAVECREST* SIA-3000 should be powered on, attached via GPIB cable to the host workstation, with CAL OUT connected to IN1 and CAL $\overline{\text{OUT}}$ connected to IN2.

NOTE: Support is included for both National Instruments and SICL interface libraries on the Linux platform. The only required change is that your application must be linked against the PAPI library ***libWChpb.so*** instead of ***libWCgpb.so*** when using the SICL libraries. The makefile included with the Linux sample application includes a detailed explanation of the compilation changes required in order to utilize the SICL interface.

To execute the sample application, issue the following from a command prompt:

```
./sample
```

NOTE: Preceding the application name with "./" ensures that the executable is launched even if the current directory is not included in the search path on UNIX.

If the sample application is successfully executed, the program should produce an output similar to the following:

```
Single Histogram Mean: 50.392295ns
Single Histogram Sdev: 2.185318ps
Strike ENTER to continue
```

Congratulations! You have just built and ran your first application using the *WAVECREST* Production API.

1-7 REVIEWING THE SAMPLE APPLICATION

Let's examine the sample application in more detail.

STEP 1 - Declare Required Include Files and Input Channels

The *WAVECREST* PAPI utilizes a number of custom structures which are declared in the supplied "include" files. In this example, IN1 and IN2 on the SIA-3000™ are declared as measurement inputs.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "../wccomm.h"
#include "../wcfcnl.h"

/* Uncomment for SUNOS                                */
/*#define SUNOS 1                                     */
#if (WIN32 | SUNOS | SOLARIS2 | LINUX)
#define APIDEVTYPE              GPIB_IO
#define DEVICENAME              "dev5"
#else
#if (HPUX)
#define APIDEVTYPE              HPIB_IO
#define DEVICENAME              "hpib,5"
#endif
#endif

/* Define channel inputs for illustration purposes */
#define IN_1                    1
#define IN_2                    2

int main(int argc, char *argv[])
  {
```

STEP 2 - Allocate Required Structures

Each tool has a specific structure and several function calls to facilitate the data acquisition process. These structures contain input information concerning how to acquire the data, and output data as a result of the acquisition.

```
  DCOM dcom;
  HIST hist;
  JITT jitt;
  long ApiDevId, retn = 0;
  char cmnd[256];

  /* Avoid compiler warnings */
  argc; argv;
```

STEP 3 - Initialize The Structures

Before utilizing a Tool Structure, it must be initialized. This initialization may involve two or more parts.

The first part is to zero out the array using the standard memset() function. This step should only be performed once immediately after the structure is allocated and prior to it being used, as information concerning dynamic memory allocation is subsequently added to the structure.

The second part is to call the function intended to initialize each of the particular structure parameters to their default values. In this case the FCNL_Defxxxx() function is called. This insures that all parameters contain reasonable values.

The final step is to manually modify any parameters from their default values. Great care should be used when manually adjusting parameters to ensure that valid values are used.

NOTE: lChanNum contains start channel in the lower 16 bits and stop channel in the upper 16 bits.

```
/* Initialize our structures */
memset ( &hist, 0, sizeof ( HIST ) );
FCNL_DefHist ( &hist );
memset ( &jitt, 0, sizeof ( JITT ) );
FCNL_DefJitt ( &jitt );
memset ( &dcom, 0, sizeof ( DCOM ) );
FCNL_DefDcom ( &dcom );

/* To measure propagation delay between IN_1 and IN_2, these inputs are identified within a
bitfield */
hist.tParm.lChanNum = IN_1 + (IN_2 << 16);
hist.tParm.lStopCnt = 1;
hist.tParm.lFuncNum = FUNC_TPD_PP;

/* Make Known Pattern w/ Marker measurements using a simple clock pattern */
strcpy(&dcom.sPtnName[0], "clock.ptn");
dcom.tParm.lChanNum = IN_1;
dcom.tParm.lAutoArm = ARM_EXTRN;
dcom.tParm.lExtnArm = IN_2;

/* Measure High Frequency Modulation (Rising Edge, Triangular FFT window) */
jitt.tParm.lFuncNum = FUNC_TT_P;
jitt.tFfts.lWinType = FFT_TRI;
jitt.lAutoFix = 1;
```

STEP 4 - Initializing the SIA-3000

COMM_InitDev() must be called once at the beginning of your application to pass information concerning the remote configuration. The initialization values shown may need to be altered if a non-standard configuration is used. See Section 3.1.1 for complete details concerning configuration options.

All PAPI functions return a non-zero value in the event of an error. These error codes are defined in the supplied include files. A successful call to COMM_InitDev() must be accomplished before any other calls to the *WAVECREST* PAPI.

```
/* Initialize device */
  if ( ( ApiDevId = COMM_InitDev ( APIDEVTYPE, DEVICENAME ) ) < 1 )
    {
    printf("\nCOMM_InitDev() failed...\n");
    goto Error;
    }

  /* Turn on calibration source */
  if ( ( retn = COMM_TalkDev ( ApiDevId, ":CAL:SIG 10MSQ" ) ) != SIA_SUCCESS)
    {
    printf("\nCOMM_TalkDev() failed...\n");
    goto Error;
    }
```

STEP 5 - Perform PulseFind

In this exercise, the calibration signals are used to provide a signal. FCNL_PulsFnd requires two parameters. The first parameter is the ApId number returned from the COMM_InitDev function call. The second parameter is a pointer to one of the PARM structures (initialized in step 3).

```
  /* Go ahead and perform a pulsefind */
  if ( ( retn = FCNL_PulsFnd ( ApiDevId, &hist.tParm ) ) != SIA_SUCCESS)
    {
    printf("\nFCNL_PulsFnd() failed...\n");
    goto Error;
    }
```

STEP 6 - Perform Measurement and Return Statistics

A single call is made to perform the acquisition. Information concerning how to acquire the data is drawn from the HIST structure, and output data as a result of the acquisition is also returned in the HIST structure. If an error occurs during the acquisition a non-zero value is returned. See Appendix A for definition of error codes.

Note that the *WAVECREST* PAPI performs its own dynamic memory allocation as required. The calling application does not need to concern itself with memory management. However, since dynamic memory allocation information is contained within the structure, the supplied cleanup functions detailed below must be utilized in order to avoid memory leaks.

Acquisition functions may be called repeatedly with the same Tool Structure. When doing so the output results contained within the structure are simply overwritten. Any dynamic memory previously allocated is re-utilized. Using the same Tool Structure over and over again has the desirable attribute of reducing the memory fragmentation that would occur if memory was allocated, freed, and reallocated repeatedly.

```
/* Perform a measurement and return the statistics */
if ( ( retn = FCNL_RqstPkt ( ApiDevId, &hist, WIND_HIST ) ) != SIA_SUCCESS)
  {
  printf("\nFCNL_RqstPkt() failed...\n");
  goto Error;
  }
/* Now retrieve the plot structures for the previous measurement */
/* This call is not necessary unless you want the plot data */
if ( ( retn = FCNL_RqstAll ( ApiDevId, &hist, WIND_HIST ) ) != SIA_SUCCESS)
  {
  printf("\nFCNL_RqstAll() failed...\n");
  goto Error;
  }
```

STEP 7 - Print Results

Results to be printed are drawn directly from the HIST structure. Note that all results are returned in the units of Hertz, Volts, and seconds. Therefore a conversion factor may be required in order to display the results in more appropriate units. For complete details on the HIST structure, see Section 2-25.

```
/* Print the results */
printf("Single Histogram Mean: %lfns\n", hist.dNormAvg * 1e9);
printf("Single Histogram Sdev: %lfps\n", hist.dNormSig * 1e12);
```

STEP 8 - Perform a dataCOM Acquisition

This is an example of a dataCOM acquisition. FCNL_RqstPkt retrieves the data and FCNL_RqstAll returns all of the plot data. For complete details on the dataCOM Tool and Structure, see Section 2-20.

```
if ( ( retn = FCNL_RqstPkt ( ApiDevId, &dcom, WIND_DCOM ) )  != SIA_SUCCESS)
  {
  printf("\nFCNL_RqstPkt() failed...\n");
  goto Error;
  }
if ( ( retn = FCNL_RqstAll ( ApiDevId, &dcom, WIND_DCOM ) )  != SIA_SUCCESS)
  {
  printf("\nFCNL_RqstAll() failed...\n");
  goto Error;
  }
```

STEP 9 - Cleanup and Terminate Application

Before terminating the application, the supplied cleanup functions should be called. FCNL_ClrHist and FCNL_ClrJitt frees any dynamic memory which may have been allocated and clears out the structure. COMM_CloseDev() closes the remote device driver. After this cleanup has been performed the application may terminate normally.

```
Error:
/* Return an error message if we had a problem */
  if ( retn )
    printf ( "Return Code: %i\n", retn );

/* Perform any cleanup and exit */
FCNL_ClrHist ( &hist );
FCNL_ClrJitt ( &jitt );
FCNL_ClrDcom ( &dcom );
COMM_CloseDev (ApiDevId);
Printf("Strike ENTER to continue…");
Fgets(cmnd, sizeof(cmnd), stdin);
return (retn);
}
```

1-8 WHERE TO GO FROM HERE

This completes your introduction to the *WAVECREST* PAPI. You should have installed the software, built a basic application and reviewed its composition. You should now have a basic understanding of the underlying framework, and be ready to leverage that understanding to further explore the interface. Subsequent chapters present additional detail concerning the structures and functions provided with the *WAVECREST* PAPI.

SECTION 2 – TOOL-SPECIFIC COMMANDS AND STRUCTURES

2-1 INTRODUCTION

There are 29 tools currently supported in the Production API. These tools, or measurement windows, perform all measurement functions of the SIA-3000 as well as all calculations based on the measurements. All of these tools are represented in software to enable easy measurement programming over GPIB. For any particular measurement, simply select the appropriate tool, program the necessary settings and then execute the measurement command.

All measurements are handled by sending a measurement window structure containing all input parameters to a calling function, which initiates the measurement. Each of the measurement window structures is specific to one of the standard acquisition tools contained in the GigaView software. Additional sub-structures are also defined that are used within these standard measurement window structures. Beginning with Section 2-3, the additional structures are defined. The measurement window structures and commands are detailed for the standard acquisition tools starting with Section 2-15.

Please note that many of the measurement window structures contain padding fields. These fields are usually called `lPad1`, `lPad2`,… *or* `lPadLoc1`, `lPadLoc2`,… *and are used to insure that variables are placed in the same absolute locations within the structure regardless of compiler padding which varies from system to system. These fields are only used to take up space, and can be safely ignored.*

Section 2-2 outlines the calling functions that are used to initiate a measurement and to retrieve the data from the instrument. The commands in Section 2-2 are completely independent of the measurement window structure to be used and are used with all of the structures. Once the measurement has been successfully completed, the results are returned in the output section of the same measurement window structure.

The basic process for conducting a measurement is as follows:

1. Initialize a window structure. This means that memory must be allocated, variables declared and the structure set to defaults.
2. Modify any structure elements as needed for the given measurement. Typical modifications include channel number, pattern file name (if data), number of measurements and triggering information.
3. Call a measurement command. Use one of the measurement commands from Section 3.2 and pass it the window structure defined in 1 and 2.
4. Parse the window structure for the results. Once the measurement is completed, the command will return any error messages or a SIA_SUCCESS if measurement was completed successfully.

If the program is to be done in a production environment, some attention needs to be paid to the memory handling. In step 1, we allocated memory for the structure. If this is done repeatedly without clearing the memory, this will result in a memory overflow error during run time. This can be avoided by either moving the memory declarations to a section of the program that is executed only once. Be sure to execute an appropriate `FCNL_Clrxxxx()` command when the structure is no longer needed. This only needs to be done once at the end of the program. Alternatively, memory can be allocated and cleared on a per-run basis although this will have a huge impact on test time.

2-2 MEASUREMENT COMMANDS

There are three basic commands used to execute a measurement: `FCNL_RqstPkt`, `FCNL_RqstAll` and `FCNL_MultPkt`. The `FCNL_RqstPkt` command is used to perform a measurement where only the statistical result is desired. The `FCNL_RqstAll` command is used to perform a measurement where the plot data is desired. The `FCNL_MultPkt` command is used when the same measurement is to be executed on multiple channels. Again, the process is to define the measurement window structure then pass it to one of these three commands for measurement execution. Each of these three commands requires the device ID and the window structure as an input.

long __stdcall FCNL_RqstPkt (long ApiDevId, void *pData, long nType)

Use this function to perform data acquisitions with a particular tool (Histogram, dataCOM, etc.). Information on how to acquire the data is drawn from the tool structure, and statistical output data resulting from the acquisition is returned in the tool structure. Acquisition functions may be called repeatedly with the same tool structure. When doing so, the output results contained within the structure are overwritten and any previously allocated dynamic memory is re-utilized. Each measurement window structure is defined in Section 3.3. As shown in the example, a measurement window structure is allocated in memory, then modified for the given measurement and passed to the command for measurement execution. The results are stored in the measurement window structure that was used by the FCNL_RqstPkt command. *To retrieve the structure's plot data, use FCNL_RqstAll().*

INPUTS

ApiDevid - Contains the API Device ID of the device. This value can be from 1 to 31.

pData - Pointer to a particular tool structure like HIST, DCOM, etc. to hold the input and output values.

nType - Flag specifying the type of the request such as WIND_HIST, WIND_JITT etc. as described in section 3.1 in the column “Tool Type”.

OUTPUTS

Returns SIA_SUCCESS upon successful completion or a specific error code (negative value) indicating what type of error occurred.

EXAMPLE

```
memset ( &hist, 0, sizeof ( HIST ) );          //Allocate memory for measurement structure
FCNL_DefHist ( &hist );                        //set structure to defaults
hist.tParm.lFuncNum = FUNC_PER;                //Select period meas function of histogram tool
hist.tParm.lChanNum = 1;                       //Select channel number 1
hist.tParm.lStrtCnt = 1;                       //start on first edge after arm
hist.tParm.lStopCnt = 2;                       //stop measurement on second rising edge
FCNL_RqstPkt ( ApiDevId, &hist, WIND_HIST );   //execute the measurement.
```

long __stdcall FCNL_RqstAll (long ApiDevId, void *pData, long nType)

This function is for getting the plot data of a particular type of measurement- like histogram that was done immediately prior to this request. This command is kept separate from the measurement command to minimize test time when the plot data is not desired. Once this command is executed, the plot data can be extracted from the measurement window histogram. *See Section 2-3 for information on the PLTD structure and Section 2-40 for an example on extracting plot data from a measurement window structure.*

INPUTS

ApiDevid - Integer containing the API Device ID of the device. This value can be from 1 to 31.

pData - Pointer to a particular tool structure like HIST, DCOM, etc. that contains the input/output and plot values.

nType - Flag specifying the type of the request, such as WIND_HIST, WIND_JITT, etc.

OUTPUTS

Returns SIA_SUCCESS upon successful completion or a specific error code (negative value) indicating what type of error occurred.

EXAMPLE

```
FCNL_RqstPkt ( ApiDevId, &hist, WIND_HIST );        //execute the measurement.
FCNL_RqstAll ( ApiDevId, &hist, WIND_HIST );        //get plot data
```

long __stdcall FCNL_MultPkt (long ApiDevId, void *pData, long nType, long nRefChn, long nChns)

Use this function to perform pseudo-parallel data acquisitions with a particular tool (Histogram, dataCOM, etc.) on multiple channels. Measurement setup is contained in the first element of the array of structures pointed to by *pData. Results of the measurement are contained in the array structures. Only the structure needs to be defined. All other structures will be copied from the first array structure. In the example below, two structures are created (hist[0] to hist[1]) and defined as type HIST. Then, only the first element, hist[0], is modified with the desired measurement setup parameters. The calling function will copy the info in hist[0] to hist[1].

INPUTS

ApiDevid - Contains the API Device ID of the device. This value can be from 1 to 31.

pData - Pointer to an array of particular tool structures such as HIST, DCOM, etc. to hold the input and output values

nType - Flag specifying the type of tool structure: WIND_HIST, WIND_JITT etc.

nRefChn - Specifies the reference channel for channel-to-channel measurements. For single-channel measurements, set to 0.

nChns - Bit field specifying the channels to measure. Set Bit0 to measure channel 1, Bit1 to measure channel 2, etc.

OUTPUTS

Returns SIA_SUCCESS upon successful completion or a specific error code (negative value) indicating what type of error occurred.

EXAMPLE

```
static HIST hist[2];                                //declare 2 window structures of type HIST
memset ( &hist[0], 0, sizeof ( HIST ) );            //clear the memory for first structure
FCNL_DefHist ( &hist[0] );                          //Set first structure to defaults.
hist[0].tParm.lFuncNum = FUNC_PER;                  //declare measurement to be made
hist[0].tParm.lStrtCnt = 1;                         //declare the start count of the measurement
hist[0].tParm.lStopCnt = 2;                         //declare the stop count of the measurement
FCNL_MultPkt(ApiDevId, &hist[0], WIND_HIST, 0, 3) //execute the measurement on channel 1
                                                    //and channel 2. Note that the nRefChn field
                                                    //is set to 0 since no Ref Channel used.
```

2-3 PLOT DATA STRUCTURE

This is an output structure used to hold the necessary information to construct a view of the measurement that was performed. For example, the histogram tool can return a histogram plot.

In order to optimize performance the plot data itself is returned in the measurement window structure only when FCNL_RqstAll() is called. The plot statistics are valid, but the pointer dData will be invalid until FCNL_RqstAll() transfers the plot data, stores it locally, and assigns the dData pointer to this local copy. The PLTD structure can then be used by a plotting utility to display the plot information. The plot data may be manipulated directly from the PLTD structure, or FCNL_GetXval() and FCNL_GetYval() may be called for simplicity.

See section 2-2 for more information about the FCNL_RqstAll() command and section 2-1 for higher level Plot utility functions.

The data is organized by linear indexing of the x-axis and assignment of one element of X for each element in the y-axis data array. The y-coordinate is extracted from the dData array, while the x-coordinate may be calculated using the number of points in the array and the x-axis extents.

This formula is used to calculate an X value for a given index (0 <= index < plot.lNumb):

X = (plot.dXmax – plot.dXmin) * (double) index / (double) (plot.lNumb - 1) + plot.dXmin;

```
typedef struct
  {
  double *dData;              /* Pointer to y-axis data array          */
  long    lNumb;              /* Number of valid data points           */
  long    lRsvd;              /* Used to track memory allocation       */
  long    lPad1;
  double  dXmin, dXmax;       /* X-axis values for ends of data array   */
  double  dYmin, dYmax;       /* Min/Max values in y-axis data array    */
  double  dYavg, dYstd;       /* Average/1-Sigma values for data array  */

  long    lXminIndx;          /* Used by histograms to indicate         */
  long    lXmaxIndx;          /* location of first and last valid bins  */

  long    lYminIndx;          /* Indicates the location where the       */
  long    lYmaxIndx;          /* min/max values occur in data array     */

  double  dAltXmin, dAltXmax; /* Alternate X-axis values, if applicable */
  } PLTD;
```

dData Pointer to y-axis data array.
LNumb Number of valid data points.
LRsvd Used to track memory allocation.
dXmin,dXmax X-axis values for ends of data array.
dYmin,dYmax Min & Max values in Y-axis data array.
dYavg,dYstd Average & 1-Sigma values for data array.
lXminIndx,lXmaxIndx Used by histograms to indicate location of first and last valid bins.
lYminIndx,lYmaxIndx Indicates the location where the Min & Max values occur in data array.
dAltXmin,dAltXmax Alternate X-axis values, if applicable. For graphs where it makes sense an alternate X-axis unit may be calculated. Examples include time or index on a Clock High Frequency Modulation Analysis 1-sigma plot, or unit interval or time on a Datacom Known Pattern With marker bathtub plot. If no applicable alternate unit is defined these variables will both be set to zero.

2-4 ACQUISITION PARAMETER STRUCTURE

An acquisition parameter structure is contained in every measurement window structure. It is an input structure that holds common information for a variety of tool measurements such as channel number, voltage, and sample size. For some simple tools, information such as start and stop counts will also be drawn from this structure. For more algorithm-based tools these values may be computed as needed.

```
typedef struct
  {
  long    lFuncNum;         /* Function to measure                      */
  long    lChanNum;         /* Channel to measure                       */
  long    lStrtCnt;         /* Channel start count                      */
  long    lStopCnt;         /* Channel stop count                       */
  long    lSampCnt;         /* Sample size                              */
  long    lPadLoc1;
  double  dStrtVlt;         /* Start voltage                            */
  double  dStopVlt;         /* Stop voltage                             */
  long    lExtnArm;         /* Arm when external is selected            */
  long    lPadLoc2;

  long    lOscTrig;         /* O-scope trigger                          */
  long    lOscEdge;         /* O-scope rise/fall trig                   */

  long    lFiltEnb;         /* Filter enable                            */
  long    lPadLoc3;
  double  dFiltMin;         /* Filter minimum                           */
  double  dFiltMax;         /* Filter maximum                           */

  long    lAutoArm;         /* Auto arm enable/mode                     */
  long    lArmEdge;         /* Arm rise/fall edge                       */
  long    lGatEdge;         /* Gate rise/fall edge                      */
  long    lPadLoc4;
  double  dArmVolt;         /* Arm user voltage                         */
  double  dGatVolt;         /* Gate voltage                             */
  long    lGateEnb;         /* Enable gating                            */
  long    lCmdFlag;         /* Command flag for timestamping, etc..     */

  long    lFndMode;         /* Pulse find mode                          */
  long    lFndPcnt;         /* Pulse find percent                       */
  long    lPadLoc5;
  long    lPadLoc6;
  long    lPadLoc7[2][6];

  long    lTimeOut;         /* Timeout in sec's, if negative it's ms    */
  long    lArmMove;         /* Arming delay in steps [can be +/-]       */
  long    lNotUsed[2];
  } PARM;
```

lFuncNum Function to measure, use any of the following:

```
2-Channel:     FUNC_TPD_PP    TPD +/+
               FUNC_TPD_MM    TPD -/-
               FUNC_TPD_PM    TPD +/-
               FUNC_TPD_MP    TPD -/+
1-Channel:     FUNC_TT_P      Rising edge time
               FUNC_TT_M      Falling edge time
               FUNC_PW_P      Positive pulse width
               FUNC_PW_M      Negative pulse width
               FUNC_PER       Period
               FUNC_FREQ      Frequency
               FUNC_PER_M     Period Minus
Default:       FUNC_PER
```

lChanNum Channel to measure, the minimum value is 1, the maximum is based on the system configuration. For two channel TPD measurements, the lower 16 bits define the start channel and the upper 16 bits defines the stop channel. In the Oscilloscope tool, channels are designated by a bitfield, implying that multiple channels can be measured at the same time. (example: If 1ChanNum=3, channels 1 and 2 will be measured)
Default: 1

lStrtCnt Channel start count; the valid range is from 1 to 10,000,000.
Default: 1

lStopCnt Channel stop count; the valid range is from 1 to 10,000,000.
Default: 2

lSampCnt Sample size; the valid range is from 1 to 950,000.
Default: 300

dStrtVlt Start voltage sets the reference voltage used to initiate the time measurement. The valid range is +/-2.0 volts.
Default: 0.0

dStopVlt Stop voltage sets the reference voltage used to terminate the time measurement. The valid range is +/-2.0 volts.
Default: 0.0

lExtnArm Channel to use for external arming. Only used if lAutoArm is set to ARM_EXTRN. The minimum is 1, the maximum is based on the system configuration.
Default: 1

lOscTrig Channel to use for oscilloscope trigger.
Default: 1

lOscEdge Edge to use to trigger oscilloscope, use any of the following: EDGE_FALL, EDGE_RISE.
Default: EDGE_RISE

lFiltEnb Filter enable, any non-zero value enables filters.
Default: 0

dFiltMin Filter minimum in seconds, only used if lFiltEnb is non-zero; valid range is +/-2.49 seconds.
Default: -2.49

dFiltMax Filter maximum in seconds, only used if lFiltEnb is non-zero; valid range is +/-2.49 seconds.
Default: +2.49

lAutoArm Auto arm enable and mode, use any of the following:
ARM_EXTRN Arm using one of the external arms
ARM_START Auto-arm on next start event
ARM_STOP Auto-arm on next stop event
Default: ARM_STOP

lArmEdge Arming edge to use, only used if lAutoArm is set to ARM_EXTRN and may be either EDGE_FALL or EDGE_RISE.
Default: EDGE_RISE

lGateEdge Edge to use when external arming gate is enabled; only used if lAutoArm is set to ARM_EXTRN and may be either EDGE_FALL or EDGE_RISE.
Default: EDGE_RISE

dArmVolt Arm1 voltage, the valid range is +/-2.0 volts and is only used if lAutoArm is set to ARM_EXTRN.
Default: 0.0

dGatVolt Arm2 voltage, the valid range is +/-2.0 volts and is only used if lAutoArm is set to ARM_EXTRN.
Default: 0.0

lGateEnb Enable external arm gating on the currently selected external arming channel; any non-zero value enables gating.
When gating is enabled, the arming edge and reference voltages of the current external arm channel are associated with gating.
Default: 0

lCmdFlag	Bitfield containing modifiers. Most of the bits are reserved for internal use and should be left to zero. However, the following bits are provided for enabling user selectable options. CMD_PATNMARK (1<<4) Use PM50 card as arm source on the selected external arming channel. CMD_BWENHANCED (1<<10) Apply Bandwidth Enhancement algorithm to scope data. This is only appropriate if a stationary waveform relative to the trigger is available. Default: 0
lFndMode	Pulse find mode, may be one of the following: PFND_FLAT Use flat algorithm for pulse-find calculation. PFND_PEAK Use peak value for pulse-find calculation. Default: PFND_PEAK
lFndPcnt	Pulse find percentage, may be one of the following: PCNT_5050 Use 50/50 level for pulse-find calculation. PCNT_1090 Use 10/90 level for pulse-find calculation. PCNT_9010 Use 90/10 level for pulse-find calculation. PCNT_USER Do NOT perform pulse-find, manual mode. When this mode is selected, valid voltages must be loaded in the dStrtVlt, dStopVlt, dArmVolt and dGatVolt parameters. PCNT_2080 Use 20/80 level for pulse-find calculation. PCNT_8020 Use 80/20 level for pulse-find calculation. Default: PCNT_5050
lTimeOut	Seconds for timeout before returning an error. A positive number is used to indicate a value in seconds, a negative number is used to indicate a value in milliseconds (Ex: -100 indicates 100ms.) The range of valid times is 10ms to 50s. Default: 2
lArmMove	This variable controls an arming delay that can be applied to either an external arm source, or the channel itself if auto-arming is enabled. Values in the range of –40 to 40 are acceptable (each step represents a 25ps delay from nominal). <u>Arm Delay (ns)</u> <u>Index Value</u> 19.0 -40 ... ... 19.75 -10 ... ... 20.0 0 ... ... 21.0 40 Default: -10
lNotUsed[n]	Formerly DSM channel select, no longer used.

void __stdcall FCNL_DefParm (PARM *parm)

This function is used to fill a PARM structure with default values that are reasonable. It is not necessary to clear a PARM structure using the standard memset() function prior to calling this function since no dynamic memory allocation exists within this structure.

INPUTS

parm - Pointer to a PARM structure. Memory needs to be allocated by the caller.

OUTPUTS

None.

2-5 TAILFIT RESULT STRUCTURE

This output structure holds the results of a TailFit algorithm execution. This structure is imbedded in all of the measurement structures that use the TailFit algorithm to separate Random Jitter and Deterministic Jitter from a histogram of measurements. Should the measurement come to completion without a successful TailFit, re-execute the measurement to acquire more data.

```
typedef struct
  {
  long    lGood;              /* Flag to indicate successful tail-fit   */
  long    lPad1;
  SIDE    tL, tR;             /* Individual left/right tail-fit data    */
  double  dDjit;              /* Deterministic jitter, from both sides  */
  double  dRjit;              /* Random jitter, average from both sides */
  double  dTjit;              /* Total jitter, calculated from bathtub  */
  } TFIT;
```

lGood Flag to indicate successful tail-fit. This flag will be set to a one if the TailFit algorithm successfully separated RJ and DJ from within the histogram of measurements.

tL, tR Structures of type SIDE, defined below, containg individual left & right tail-fit data.

dDjit Total Deterministic jitter, from both sides.

dRjit Total Random jitter, average from both sides.

dTjit Total jitter, calculated from bathtub curve.

2-6 SINGLE SIDE OF TAILFIT STRUCTURE

This output structure is used within the TFIT structure to contain all of the results of a Tail-Fit pertaining to one side of the measurement histogram. This structure contains side specific RJ and DJ information as well as Chi-squared data defining the "goodness of fit" criteria.

```
typedef struct
  {
  double  dCoef[ 3 ];         /* Used by WavGetTfit() to generate       */
                              /* idealized tail-fit curves              */
  double  dDjit;              /* Deterministic jitter, this side only   */
  double  dRjit;              /* Random jitter, this side only          */
  double  dChsq;              /* ChiSquare indicator, goodness of fit   */
  double  dLoValu, dHiValu;   /* Xval range over which tail was fitted  */
  double  dMuValu;            /* Projected Xval where mu was determined */
  double  dEftvDj, dEftvRj;   /* Effective jitter if calculated         */
  double  dTjit;              /* Total jitter, calculated from bathtub  */
  } SIDE;
```

dCoef Coefficient used to generate idealized tail-fit curves.

dDjit Deterministic jitter, this side only.

dRjit Random jitter, this side only.

dChsq ChiSquare indicator, goodness of fit.

dLoValu,dHiValu range over which tail was fitted.

dMuValu Projected dXval where mu was determined.

dEftvDj,dEftvRj Holds the effective jitter values if calculated. To calculate the effective jitter, lFndEftv must contain a non-zero value. Since the effective jitter is calculated by optimizing a curve-fit, a result is not guaranteed. If the curve-fit fails, a negative value will be returned in these variables.

2-7 SPECIFICATION LIMIT STRUCTURE

This input structure is used by the Datacom Known Pattern With Marker Tool to contain the parameters for **tRateInf**, **tDdjtInf** and **tRjpjInf.** This tool uses these specifications when setting up the measurement for capturing bit rate, DDJ and RJ/PJ spectra respectively.

```
typedef struct
  {
  long    lSampCnt;          /* Sample size to use                      */
  long    lPad1;
  double  dMaxSerr;          /* LIM_ERROR if this std. error exceeded  */
  long    lPtnReps;          /* Patterns to sample across               */
  long    lPad2;
  } SPEC;
```

lSampCnt Sample size to use when acquiring data
Valid Entries: 1 to 10,000,000
Default: 100

dMaxSerr Value of standard error which is tolerated, used to identify wrong pattern or other setup error.
Valid Entries: any integer greater than or equal to 0
Default: 0.5

lPtnReps Patterns to sample across. The larger this number is the more accurate the measurement will be with regards to absolute time measurements. This is due to the effect of aver
Valid Entries: 1 -
Default: rRateInf - 10
dDdjtInf - 1
dRjpjInf - 1

lPad1,lPad2 Internal parameters, do not modify.

2-8 DDJ+DCD DATA STRUCTURE

This output structure contains all of the measurement data used to calculate DDJ+DCD in the Datacom Known Pattern With Marker Tool. This tool contains a pointer to an array of DDJT structures with an element for each transition in the pattern.

```
typedef struct
  {
  double  dMean;              /* Average value for this span           */
  double  dVars;              /* Variance value for this span          */
  double  dMini;              /* Minimum value for this span           */
  double  dMaxi;              /* Maximum value for this span           */
  double  dDdjt;              /* Static displacement for this span (UI) */
  double  dFilt;              /* DDJT after LPF is applied (UI)        */
  long    lNumb;              /* Number of measures in this span       */
  long    lPad1;
  } DDJT;
```

dMean	Average value for this span. This is the time elapsed from the first edge in the pattern to transition associated with this structure. In an ideal signal (one which contains no jitter), this value would be an integer multiple of the bit period. Any deviation there of is considered jitter and becomes an element of the DDJ+DCD histogram.
dVars	Variance value for this span. This is net deviation of the mean to the ideal bit transition.
dMini	Minimum value for this span. This is the earliest transition for this bit period. It defines the earliest transition for this location in the pattern.
dMaxi	Maximum value for this span. This is the latest transition for this bit period. It defines the latest transition for this location in the pattern.
dDdjt	Static displacement for this span (UI).
dFilt	DDJT after HPF is applied (UI).
lNumb	Number of measures in this span.

2-9 PATTERN STRUCTURE

The pattern structure is used internally by the system as part of the measurement process. When tools are used that reference a pattern, they have a member called sPtnName in their binary packet. This field holds the name of the pattern file that is to be used. Whenever a binary packet is sent which contains a new value in sPtnName, a new internal representation is loaded.

```
typedef struct
  {
  char    *bHex;           /* Pointer to raw hex data                  */
  short   *iPos;           /* Pointer to run length encoded data       */
  short   *iCnt;           /* Pointer to start/stop counts to use      */
  double  *dCal;           /* Pointer to calibration data if present */
  long    lLpat;           /* The length of pattern in UI              */
  long    lEpat;           /* The edge count of pattern pos or neg     */
  double  dCalUI;          /* Cal data taken at this unit interval     */
  } PATN;
```

2-10 FFT WINDOW AND ANALYSIS STRUCTURE

This is an input structure used to specify the type of windowing function to use when generating an FFT. It also contains information for an average calculation that is performed on the resulting FFT for some specific tools such as Low Frequency Modulation Analysis.

```
typedef struct
  {
  long    lWinType;          /* Window type, use FFT constants above   */
  long    lPadMult;          /* Power of 2 to use for padding (0 - 5)  */
  double  dCtrFreq;          /* Frequency to assess yavg in plot array */
  double  dRngWdth;          /* Width over which to assess yavg        */
  double  dAlphFct;          /* Alpha factor for Kaiser-Bessel window  */
  } FFTS;
```

lWinType Window type, use one of the following:

FFT_RCT	Rectangular window
FFT_KAI	Kaiser-Bessel window
FFT_TRI	Triangular window
FFT_HAM	Hamming window
FFT_HAN	Hanning window
FFT_BLK	Blackman window
FFT_GAU	Gaussian window
Default:	FFT_KAI

lPadMult Power of 2 to use for padding (0 - 5)
Default: 4

dCtrFreq Frequency over which to assess dYavg in plot array (Hz)
Default: 100.0

dRngWdth Width over which to assess dYavg (Hz)
Default: 10.0

dAlphFct Alpha factor when using Kaiser-Bessel window
Default: 8.0

2-11 QTYS STRUCTURE

QTYS is an output structure used to return scope results.

```
typedef struct
  {
  double  dMaxVolts;
  double  dMinVolts;
  double  dAvgVolts;
  double  dPkPkVolt;
  double  dRmsVolts;
  double  dTopVolts;
  double  dBtmVolts;
  double  dMidVolts;
  double  dAmplVolt;
  double  dOvrShoot;
  double  dUndShoot;
  double  dMaskFail;
  double  dMaskRgn1;
  double  dMaskRgn2;
  double  dMaskRgn3;
  double  dMaskTotl;
  MEAS    mRiseTime;
  MEAS    mFallTime;
  } QTYS;
```

dMaxVolts	Vmax in Volts
dMinVolts	Vmin in Volts
dAvgVolts	Vavg in Volts
dPkPkVolt	Vpk-pk (Vmax – Vmin) in Volts
dRmsVolts	Vrms in Volts
dTopVolts	Vtop in Volts, flat top
dBtmVolts	Vbase in Volts, flat base
dMidVolts	Vmid (Vtop + Vbase) / 2 in Volts
dAmplVolt	(Vtop – Vbase) in Volts
dOvrShoot	Vovershoot in Volts
dUndShoot	Vundershoot in Volts
dMaskFail	Total Mask violations
dMaskRgn1	Mask Violations in Region 1
dMaskRgn2	Mask Violations in Region 2
dMaskRgn3	Mask Violations in Region 3
dMaskTotl	Total Mask hits, both In and Outside the Mask
mRiseTime	Structure holding Risetime information
mFallTime	Structure holding Falltime information

2-12 MEAS STRUCTURE

MEAS is an output structure used to return scope rise/fall time results.

```
typedef struct
  {
  long    lGood;
  long    lPad1;
  double  dValu;
  double  dXpnt[2];
  double  dYpnt[2];
  } MEAS;
```

lGood	Flag indicates valid output data in structure.
DValu	Field holds rise or fall time result
dXpnt[2]	The starting and ending threshold location in secs.
dYpnt[2]	The starting and ending threshold location in Volts.

2-13 OHIS STRUCTURE

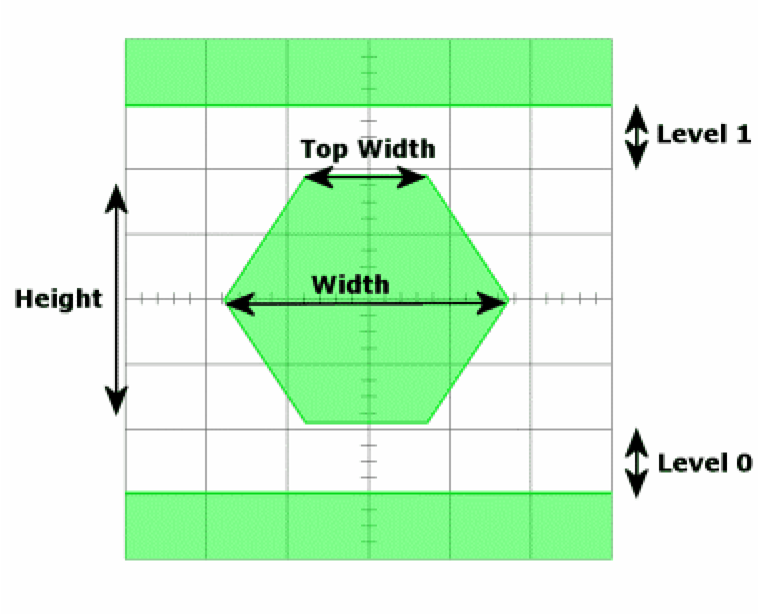
OHIS is an output structure used to return oscilloscope histogram results.

```
typedef struct
  {
  PLTD    tPlot;
  long    lCoun;
  long    lPad1;
  double  dAver;
  double  dMini;
  double  dMaxi;
  double  dSdev;
  double  dEpsl;
  double  dVars;
  } OHIS;
```

tPlot	Plot structure that holds the histogram representation
lCoun	Count of the total number of hits in the histogram
dAver	Average of all the data contained in the histogram
dMini	Minimum of all the data contained in the histogram
dMaxi	Maximum of all the data contained in the histogram
dSdev	Standard deviation of all the data contained in the histogram
dEpsl,dVars	Used internally, DO NOT ALTER!

2-14 MASK STRUCTURE

MASK is an input structure that is used to specify an Eye Mask to be used in the Scope Tool.




```
typedef struct
  {
  /* Absolute voltages */
  double  dVmask;
  double  dVoffs; /* No longer used */
  double  dV1pas;
  double  dTmask;
  double  dToffs; /* No longer used */
  double  dTflat;
  double  dV0pas;
  /* Relative voltages */
  double  dXwdUI;
  double  dXflUI;
  double  dYiPct;
  double  dV1Rel;
  double  dV0Rel;
  } MASK;
```

dVmask Absolute width of mask in secs.

dVoffs No longer used, this field can be ignored

dV1pas Distance from the top of the mask to the upper region in Volts.

dTmask Absolute position of the center of the mask in secs.

dToffs No longer used, this field can be ignored

dTflat Width of the top and bottom flats of the mask in secs.

dV0pas Distance from the bottom of mask to the lower region in Volts.

dXwdUI Relative width of mask in UI

dXflUI Relative width of the top and bottom flats of the mask in UI

dYiPct Height of inner region of mask relative to the data, expressed as %

dV1Rel Distance from top of inner region to top region expressed as a % of data height

dV0Rel Distance from bottom of inner region to bottom region expressed as a % of data height

2-15 KPWM STRUCTURE

KPWM is a measurement structure used by some of the PCI Express and Serial ATA tools.

```
typedef struct
  {
  /* Input parameters  */
  PARM    tParm;              /* Contains acquisition parameters        */
  FFTS    tFfts;              /* FFT window and analysis parameters     */
  char    sPtnName[ 128 ];    /* Name of pattern file to be used        */
  long    lAcqEdge;           /* Reference Edge and RJ+PJ measure edge  */
                              /* Could be: EDGE_FALL or EDGE_RISE       */
  long    lOneEdge;           /* If true, DCD+ISI is rise or fall only  */
  long    lQckMode;           /* Enable quick mode, external arm only   */
  long    lIntMode;           /* Interpolation mode, non-zero is linear */
  long    lErrProb;           /* Error probability for Total Jitter     */
                              /* Valid range is ( -1 to -16 )           */
  long    lHeadOff;           /* Header offset, external arming only    */
  double  dCornFrq;           /* Corner Frequency for RJ+PJ             */

  SPEC    tRateInf;           /* Parameters to acquire Bit Rate         */
  SPEC    tDdjtInf;           /* Parameters to acquire DCD+DDJ          */
  SPEC    tRjpjInf;           /* Parameters to acquire RJ+PJ            */

  double  dLpfFreq;           /* Low pass filter corner frequency       */
  double  dHpfFreq;           /* High pass filter corner frequency      */
  double  dLpfDamp;           /* Low pass filter 2nd order damp_factor  */
  double  dHpfDamp;           /* High pass filter 2nd order damp_factor */
  long    lLpfMode;           /* LPF mode, see constants above          */
  long    lHpfMode;           /* HPF mode, see constants above          */

  long    lFndEftv;           /* Flag to attempt effective jitter calc  */
  long    lMinEftv;           /* Min probability for effective fit: -4  */
  long    lMaxEftv;           /* Max probability for effective fit: -12 */

  long    lFiltEnb;           /* Enable IDLE character insertion filter */
  long    lQckTjit;           /* Fast total jitter calc - no bathtubs!  */
  long    lPllComp;           /* Enable PLL Curve Spike Compensation    */
  long    lPad0;

  /* Output parameters */
  long    lGood;              /* Flag indicates valid data in structure */
  PATN    tPatn;              /* Internal representation of pattern     */

  double  dWndFact;           /*****************************************/
  long    lMaxStop;           /*  These values are all used internally  */
  long    lPtnRoll;           /*             DO NOT ALTER!              */
  long    lAdjustPW;          /*****************************************/
  long    lPad1;

  double  dBitRate;           /* Bit Rate that was measured             */
  DDJT   *tDdjtData;          /* Raw DCD+DDJ measurements               */
  long    lDdjtRsvd;          /* Used to track memory allocation        */
  double *dRjpjData;          /* Raw variance data                      */
  long    lRjpjRsvd;          /* Used to track memory allocation        */
  long   *lPeakData;          /* Tracks detected spikes in RJ+PJ data   */
  long    lPeakNumb;          /* Count of detected spikes               */
  long    lPeakRsvd;          /* Used to track memory allocation        */
```

```
long    lHits;              /* Total samples for DDJT+RJ+PJ combined  */
double  dDdjt;              /* DCD+DDJ jitter                         */
double  dDjit;              /* Deterministic jitter                   */
double  dRjit;              /* Random jitter                          */
double  dPjit;              /* Periodic jitter                        */
double  dTjit;              /* Total jitter                           */
double  dEftvLtDj;          /* Effective jitter when enabled          */
double  dEftvLtRj;
double  dEftvRtDj;
double  dEftvRtRj;

PLOT    tRiseHist;          /* DCD+DDJ histogram of rising edges      */
PLOT    tFallHist;          /* DCD+DDJ histogram of falling edges     */
PLOT    tNormDdjt;          /* DCD+DDJvsUI for external arming only   */
PLOT    tHipfDdjt;          /* High Pass Filtered DCD+DDJvsUI         */
PLOT    tLopfDdjt;          /* Low  Pass filtered DCD+DDJvsUI         */
PLOT    tBathPlot;          /* Bathtub plot                           */
PLOT    tEftvPlot;          /* Effective Bathtub plots, if enabled    */
PLOT    tSigmNorm;          /* 1-Sigma plots                          */
PLOT    tFreqNorm;          /* Frequency plots                        */
} KPWM;
```

tParm A structure of type PARM that contains acquisition parameters. The PARM structure is discussed in full detail in Section 2-4.

tFfts A structure of type FFTS that contains the setup parameters for the FFT. See Section 2-10 for further details on FFTS structures.

sPtnName A character array containing the name of pattern file to be used, the file must exist in the pattern directory (C:\VISI\) on the SIA3000 or else an error will be returned. The first time a measurement is performed the pattern is loaded in structure tPatn.
Valid Entries: a valid file name (including extension)
Default: "k285.ptn"

lAcqEdge Reference Edge and RJ+PJ measure edge: EDGE_FALL or EDGE_RISE.
Default: EDGE_RISE

lOneEdge This parameter is used to enable a special mode where only rising or falling edges are used to access DCD+ISI, as is the case for the special PCI Express Clock Tool. Setting this parameter to 1 will enable this special mode.
Valid Entries: 0 – disable single edge mode
1 – enable single edge mode
Default: 0

lQckMode Parameter used to enable Quick Mode. QuickMode uses a sparse sample of data points for the PJ and RJ estimates. In this mode, the accuracy of these estimates is greatly reduced depending on the application. Setting this structure element to 1 enables quick mode, valid with external arm only.
Valid Entries: 0 – disable quick capture mode
1 – enable quick capture mode
Default: 0

lIntMode Parameter used to enable linear Interpolation mode for RJ & PJ estimate. RJ & PJ are calculated based on the frequency data of the noise. Since data points are captured only on the single polarity transitions, interpolation must be performed between sample points. There are two types of interpolation available in the SIA3000: linear and cubic. Setting this parameter to 1 will enable linear interpolation; otherwise, cubic interpolation will be used.
Valid Entries: 0 – use cubic interpolation in FFT data

1 – use linear interpolation in FFT data

Default: 0

lErrProb Error probability level for Total Jitter. Total Jitter is calculated based on the desired Error Probability level. This value is used in conjunction with the bathtub curve after the successful completion of a tail-fit in order to project the value of Total Jitter.

Valid Entries: -1 to -16

Default: -12

lHeadOff Header offset parameter, for use in packet-ized data which may have a frame header before the test pattern. This offset value can be used to skip past header information and into the repeating data pattern stream. This can be useful when analyzing data from disk drives when the pattern marker may be synchronized with the start of frame data.

Valid Entries: 0 to 10,000,000-pattern length I

Default: 0 (indicating no header present)

dCornFrq Corner Frequency for RJ & PJ estimate in Hertz. This value is used in conjunction with the Bit Rate and pattern to determine the maximum stop count to be used to acquire RJ & PJ data. A lower value increase acquisition time.

Valid Entries: Bit-Rate /10,000,000 to Bit-Rate I

Default: 637e3 (637kHz – Fibre Channel 1X)

tRateInf A structure of type SPEC used by the Bit Rate measurement. The structure holds measurement specific parameters such as sample count, pattern repeats and maximum standard error. See Section 2-7 for a description of the SPEC structure and its elements.

tDdjtInf A structure of type SPEC used by the Data Dependant Jitter (DDJ) measurement. The structure holds measurement specific parameters such as sample count, pattern repeats and maximum standard error. See Section 2-7 for a description of the SPEC structure and its elements.

tRjpjInf A structure of type SPEC used by RJ & PJ estimate. The structure holds measurement specific parameters such as sample count, pattern repeats and maximum standard error. See Section 2-7 for a description of the SPEC structure and it's elements.

dLpfFreq Low pass filter frequency in Hertz. This is only valid when lLpfMode is enabled.

dHpfFreq High pass filter frequency in Hertz. This is only valid when lHpfMode is enabled.

dLpfDamp Low pass damping factor. This is only valid when lLpfMode is enabled, and a 2nd order filter is selected.

dHpfDamp High pass damping factor. This is only valid when lHpfMode is enabled, and a 2nd order filter is selected.

lLpfMode Low pass filter mode. One of the following may be used:

Valid Entries: FILTERS_DISABLED
BRICKWALL_FILTER
ROLLOFF_1STORDER
ROLLOFF_2NDORDER
PCIX_CLOK_FILTER

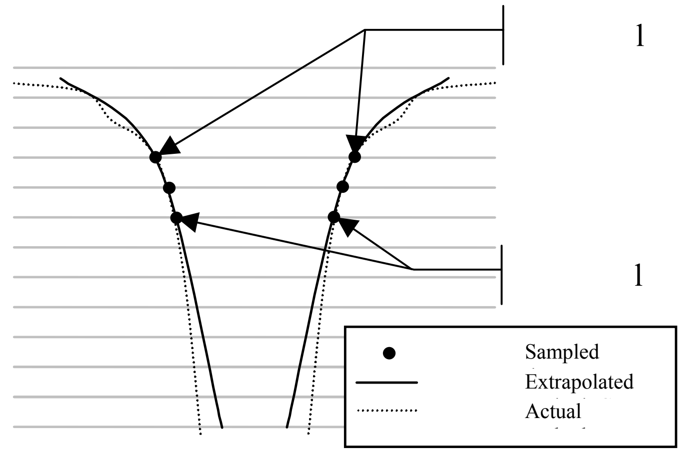
Default: FILTERS_DISABLED

lHpfMode High pass filter mode. One of the following may be used:

Valid Entries: FILTERS_DISABLED
BRICKWALL_FILTER
ROLLOFF_1STORDER
ROLLOFF_2NDORDER
PCIX_CLOK_FILTER

Default: FILTERS_DISABLED

lFndEftv Flag to indicate that an effective jitter calculation is to be attempted. Effective Jitter is a means of estimating the effective deterministic jitter as it relates to a .5 error probability. This is done by first capturing the bathtub curve using conventional RJ & DJ estimation techniques; then, extrapolating from a few points in the bathtub curve to the .5 error probability level to estimate effective DJ. Effective RJ is extracted based on the curve that was fitted to the sample points. These values should only be used to correlate to a BERT Scan measurement and should not be used as a vehicle for quantifying jitter. This technique was developed to allow BERT systems to correlate with SIA3000 results.



Extrapolated Bathtub curve versus real bathtub curve as seen by BERT

Valid Entries: 0 – disable effective jitter estimate
1 – enable effective jitter estimate
Default: 0

lMinEftv, lMaxEftv Defines the error rates at which the eye width calculation will be used in the estimating effective jitter components. lMinEftv and lMaxEftv define points on the bathtub curve from which the extrapolated RJ curve is traced. Then, where this extrapolated curve intersects the .5 error probability, the effective DJ is calculated.
Valid Entries: -1 to –16 (indicating 10^{-1} to 10^{-16} error rate)
Default: -4 and –12 (lMaxEftv: 10^{-4} BER, lMinEftv: 10^{-12} BER)

lFiltEnb Flag to enable IDLE character insertion filter. When enabled any edge measurements that are not within ± 0.5 UI will be discarded. This filter is used in systems, which may insert an idle character from time to time to compensate for buffer under-run/overrun issues. In those instances where an idle character was inserted during a measurement, the edge selection may be off. If this parameter is greater than or equal to one, the filter is enabled and measurements that differ from the mean by ± 0.5 UI will be discarded.
Valid Entries: 0 – disable idle character filter
1 – enable idle character filter
Default: 0

lQckTjit Flag to indicate a fast total jitter calculation will be performed using simple linear calculation of Total Jitter instead of convolving the DJ Probability Density Functions and the RJ Probability Density Functions. This calculation is based on the formula [TJ = DJ + n*RJ] where DJ and RJ are measured, and n is the multiplier based on a theoretical Gaussian distribution
Valid Entries: 0 do not use convolution for TJ est.
1 Convolve DJ and RJ for TJ est.
Default: 0

lPllComp Enable PLL Curve Spike Compensation. If a low frequency spike is detected in the Power Spectral Density (FFT) plot, it is automatically removed and it's energy is dispersed evenly across the rest of the Power Spectral Density.
Default: 0

lGood Flag indicates valid output data in structure. A positive value in this parameter indicates that the measurement was completed successfully, and, valid data can be extracted from this structure.

tPatn Structure of type PATN which holds all of the pattern information with regards to pattern length, pattern content, marker placement relative to location in pattern and other pattern specific metrics. (See Section 2-9 for a detailed description of the PATN structure elements.) This is an internal structure that the system uses to store pattern information and does not need to be altered by the user. The first time a measurement is performed the pattern is loaded into tPatn which is used internally for all subsequent acquisition and analysis.

dBitRate The bit rate is measured and placed in this field (Hertz).

lHits Total samples taken to calculate DDJ, RJ, and PJ values combined. Gives an indication of the actual data to support the calculated total jitter number.

dDdjt DCD+DDJ measurement in seconds. This measurement is taken from the mean deviation of each pattern edge from it's ideal location. All deviations are placed in a histogram and the peak-peak value from this histogram is placed in this structure location.

dDjit Deterministic jitter measurement, in seconds. This is the DCD+DDJ summed with the Periodic Jitter.

dRjit Random jitter estimate, in seconds.

dPjit Periodic jitter measurement, in seconds.

dTjit Total jitter estimate, in seconds.

dEftvLtDj Effective Deterministic(eDJ) jitter estimate, in seconds, for the left side of the bathtub curve. Total effective DJ is calculated by adding dEftvLtDj to dEftvRtDj. In order to calculate the effective jitter the flag lFndEftv must be enabled. Since the effective jitter is calculated by optimizing a curve-fit to the bathtub curve, a result is not guaranteed. If the curve-fit is unsuccessful, a negative value will be returned in this variable.

dEftvLtRj Effective Random(eRJ) jitter estimate, in seconds, for the left side of the bathtub curve. Total effective RJ is calculated by averaging dEftvLtRj and dEftvRtRj. In order to calculate the effective jitter the flag lFndEftv must be enabled. Since the effective jitter is calculated by optimizing a curve-fit to the bathtub curve, a result is not guaranteed. If the curve-fit is unsuccessful, a negative value will be returned in these variables.

dEftvRtDj Effective Deterministic(eDJ) jitter estimate, in seconds, for the right side of the bathtub curve. Total effective DJ is calculated by adding dEftvLtDj to dEftvRtDj. In order to calculate the effective jitter the flag lFndEftv must be enabled. Since the effective jitter is calculated by optimizing a curve-fit to the bathtub curve, a result is not guaranteed. If the curve-fit is unsuccessful, a negative value will be returned in this variable.

dEftvRtRj Effective Random(eRJ) jitter estimate, in seconds, for the right side of the bathtub curve. Total effective RJ is calculated by averaging dEftvLtRj and dEftvRtRj. In order to calculate the effective jitter the flag lFndEftv must be enabled. Since the effective jitter is calculated by optimizing a curve-fit to the bathtub curve, a result is not guaranteed. If the curve-fit is unsuccessful, a negative value will be returned in this variable.

tRiseHist Structure of type PLOT which contains all of the plot information for generating a DCD+DDJ histogram of rising edges. See Section 2-3 for details concerning the PLOT structure and its elements.

tFallHist Structure of type PLOT which contains all of the plot information for generating a DCD+DDJ histogram of falling edges. See Section 2-3 for details concerning the PLOT structure and its elements.

tNormDdjt Structure of type PLOT which contains all of the plot information for generating a DCD+DDJ versus UI plot. This plot is only valid in Pattern Marker mode. See Section 2-3 for details concerning the PLOT structure and its elements.

tHipfDdjt Structure of type PLOT which contains all of the plot information for generating an DCD+DDJ versus UI plot with the DCD+DDJ High Pass Filter enabled. This plot is only valid in Pattern Marker Mode and dDdjtHpf is a non-negative number. *(For a discussion on the High Pass Filter Function for DCD+DDJ data, see dDdjtHpf above.)* When dDdjtHpf is enabled, the dDdjt value is calculated based on applying the dDdjtHpf filter. See Section 2-3 for details concerning the PLOT structure and its elements.

tLopfDdjt Structure of type PLOT \which contains all of the plot information for generating an DCD+DDJ versus UI plot with the DCD+DDJ Low Pass Filter enabled. This plot is only valid in Pattern Marker Mode and dDdjtLpf is a non-negative number. *(For a discussion on the Low Pass Filter Function for DCD+DDJ data, see dDdjtLpf above.)* See Section 2-3 for details concerning the PLOT structure and its elements.

tBathPlot Structure of type PLOT which contains all of the plot information for generating a Bathtub curve. See Section 2-3 for details concerning the PLOT structure and its elements.

tEftvPlot Structure of type PLOT which contains all of the plot information for generating an Bathtub curve based on Effective Jitter if lFndEftv is set and a valid fit is obtained. *(For a detailed description of Effective Jitter, see lFndEftv above.)* See Section 2-3 for details concerning the PLOT structure and its elements.

tSigmNorm Structure of type PLOT which contains all of the plot information for generating an 1-Sigma versus UI plot. *(x-axis can be converted to time from UI based on dBitRate value.)* This plot describes the standard deviation for each accumulated time sample. See Section 2-3 for details concerning the PLOT structure and its elements.

tFreqNorm Structure of type PLOT which contains all of the plot information for generating a Jitter versus Frequency plot. See Section 2-3 for details concerning the PLOT structure and its elements.

The following parameters are for internal use only. They are presented for reference only. Do not try to read the values or parse the structures nor try to write the various locations.

dWndFact, lMaxStop, lPtnRoll, lAdjustPW These values are for internal use only, DO NOT ALTER or try to use.

tDdjtData Structure which contains the raw DCD+DDJ measurements. This value is for internal use only, DO NOT ALTER or try to use.

lDdjtRsvd Used to track memory allocation for tDdjtData structures. This value is for internal use only, DO NOT ALTER or try to use.

dRjpjData Raw variance data used for the calculation of RJ and PJ. This structure is for internal use only, DO NOT ALTER or try to use.

lRjpjRsvd Used to track memory allocation for dRjpjData values. This value is for internal use only, DO NOT ALTER or try to use.

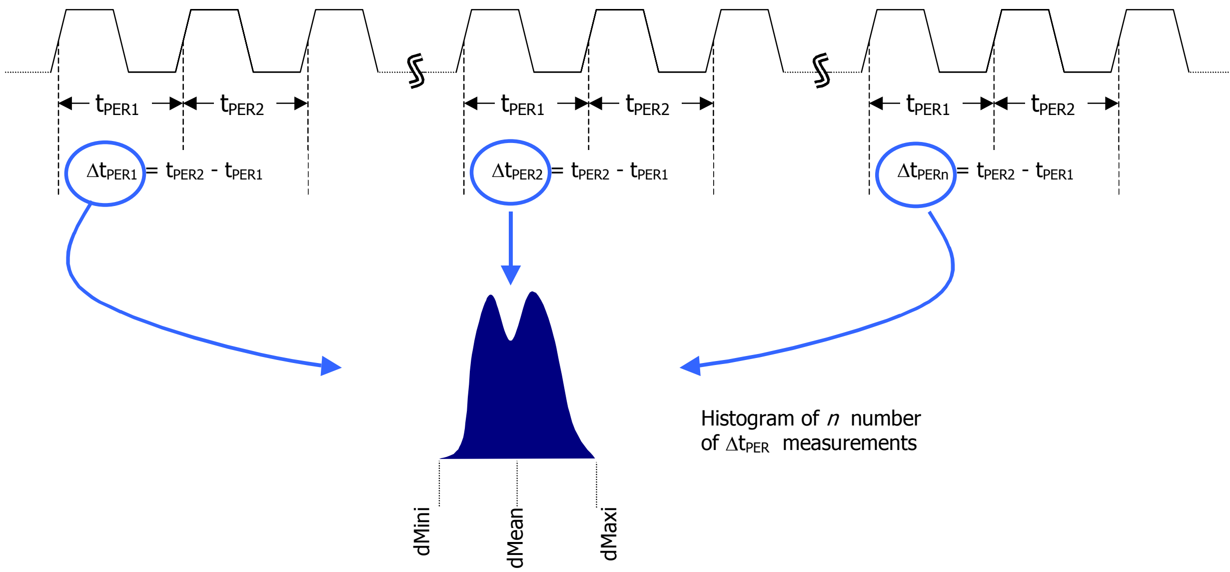
lPeakData Tracks detected spikes in RJ+PJ data. This value is for internal use only, DO NOT ALTER or try to use.

lPeakNumb Count of detected spikes, indicates the number of values in the lPeakData array.

lPeakRsvd Used to track memory allocation for lPeakData values. This value is for internal use only, DO NOT ALTER or try to use.

2-16 ADJACENT CYCLE JITTER TOOL

The Adjacent Cycle Jitter tool is used to capture period deviation information for two adjacent cycles. This measurement is called out in a few standards as a means to estimate short-term jitter. Although this metric has limited value in the physical world, it is a required measurement in many PLL test standards.




```
typedef struct
  {
  /* Input parameters  */
  PARM    tParm;              /* Contains acquisition parameters        */
  double  dUnitInt;           /* Unit Interval to assess Total Jitter   */
  long    lPassCnt;           /* Acquisitions so far, set to 0 to reset */
  long    lErrProb;           /* Error probability for Total Jitter     */
                              /* Valid range is ( -1 to -16 )           */
  long    lTailFit;           /* If non-zero a tail-fit will be tried   */
  long    lForcFit;           /* If non-zero use the force-fit method   */
  long    lMinHits;           /* Minimum hits before trying tail-fit    */
  long    lFndEftv;           /* Flag to attempt effective jitter calc  */
  long    lMinEftv;           /* Min probability for effective fit: -4  */
  long    lMaxEftv;           /* Max probability for effective fit: -12 */
  long    lAutoFix;           /* If true perform a pulsefind as req'd   */
  long    lDutCycl;           /* If non-zero make duty cycle measurement*/
  /* Output parameters */
  long    lGood;              /* Flag indicates valid data in structure */

  long    lMeasCnt;           /* Number of hits in measured normal data */
  double  dMeasMin;           /* Minimum value in measured normal data  */
  double  dMeasMax;           /* Maximum value in measured normal data  */
  double  dMeasAvg;           /* Average value of measured normal data  */
  double  dMeasSig;           /* 1-Sigma value of measured normal data  */

  long    lNormCnt;           /* Hits in adjacent cycle normal data     */
  long    lPad1;
  double  dNormMin;           /* Min. in adjacent cycle normal data     */
  double  dNormMax;           /* Max. in adjacent cycle normal data     */
  double  dNormAvg;           /* Avg. of adjacent cycle normal data     */
  double  dNormSig;           /* 1-Sig of adjacent cycle normal data    */
```

```
long    lTotlCnt;          /* # of hits in measured accumulated data */
long    lPad2;
double  dTotlMin;          /* Min. in measured accumulated data      */
double  dTotlMax;          /* Max. in measured accumulated data      */
double  dTotlAvg;          /* Avg. of measured accumulated data      */
double  dTotlSig;          /* 1-Sig of measured accumulated data     */

long    lAcumCnt;          /* Hits in adjacent cycle accumulated data*/
long    lPad3;
double  dAcumMin;          /* Min. in adj. cycle accumulated data    */
double  dAcumMax;          /* Max. in adj. cycle accumulated data    */
double  dAcumAvg;          /* Avg. of adj. cycle accumulated data    */
double  dAcumSig;          /* 1-Sig of adj. cycle accumulated data   */

double  dDutyMax;          /* Maximum value of duty cycle measurement*/
double  dDutyMin;          /* Minimum value of duty cycle measurement*/
double  dDutyAvg;          /* Average value of duty cycle measurement*/

long    lBinNumb;          /*****************************************/
long    lPad4;             /*  These values are all used internally  */
double  dLtSigma[PREVSIGMA];/*   as part of the measurement process   */
double  dRtSigma[PREVSIGMA];/*            DO NOT ALTER!               */
double  dFreq;             /*****************************************/

PLTD    tNorm;             /* Histogram of prev. adj. cycles         */
PLTD    tAcum;             /* Histogram of all adj. cycles combined  */
PLTD    tMaxi;             /* Histogram of max across all adj. cycles*/
PLTD    tBath;             /* Bathtub curves determined from PDF     */
PLTD    tEftv;             /* Effective Bathtub curves if enabled    */
TFIT    tTfit;             /* Structure containing tail-fit info     */
} ACYC;
```

tParm A structure of type PARM that contains acquisition parameter. The PARM is discussed in full detail in Section 2-4.

dUnitInt Unit Interval (UI) in seconds to assess Total Jitter as a percent of UI. Set this parameter as the metric against which TJ will be evaluated as a percentage. It is displayed as the span of the x-axis in a bathtub curve. This parameter is only used if tail-fit is enabled.
Valid Entries: any number greater than 0 which represents the time (in secs) of a bit period or unit interval.
Default: 1e-9 (1ns)

lPassCnt This parameter is a bi-directional structure element that tracks the number of acquisitions since last reset. This flag can be read after an execution or set prior to an exectution. Setting this parameter to 0 essentially resets this register. A measurement can be performed repeatedly with the same HIST structure. In this case, data is then accumulated in the tAcum and tMaxi plot structures. When lPassCnt is set to 0 the tAcum and tMaxi plot structures are flushed. It will be automatically incremented by the next measurement.
Valid Entries: any integer greater than or equal to 0
Default: 0

lErrProb Error probability level for Total Jitter. Total Jitter is calculated based on the desired Error Probability level. This value is used in conjunction with the bathtub curve after the successful completion of a tail-fit in order to project the value of Total Jitter.

Valid Entries: -1 to -16
Default: -12

lTailFit Flag to indicate whether to perform a TailFit on data in tAcum data array. If non-zero, a tail-fit will be attempted on the tAcum data array. The lGood element of the tTfit structure will indicate if the TailFit was successful. Any positive interger for this parameter will initiate the TailFit algorithm.
Valid Entries: 0 – disable TailFit
1 – enable TailFit
Default: 0

lForcFit If non-zero uses the force-fit method. If set to zero, the measurement will continue to loop until a reasonably accurate TailFit can be achieved.
Valid Entries: 0 – do not use force fit.
1 – force a fit using lMinHits number of hits.
Default: 0

lMinHits Minimum hits before attempting a tail-fit in 1000's; the default is 50. The larger the number the more likely a valid tailfit will be found.
Valid Entries: any integer ≥ 50
Default: 50

lFndEftv Flag to indicate that an effective jitter calculation is to be attempted. This is necessary for those instances in which correlation to a BERT scan is necessary. In all other practical applications, this parameter and it's resultant measurement should be ignored.
Valid Entries: 0 – do not estimate effective jitter values
1 – calculate effective jitter values
Default: 0

lMinEftv, lMaxEftv Defines the range of the bathtub curve that is to be used to calculate an effective jitter value.
Valid Entries: -1 to –16 with lMinEftv < lMaxEftv
Default: -4 for MaxEftv and –12 for MinEftv

lAutoFix Flag indicating whether to perform a pulse-find as required. Setting this value to any integer greater than zero tells the measurement to perform a pulse find if needed. The system will know if a measurement was recently performed and if a pulse find is necessary.
Valid Entries: 0 – No pulsefind prior to measurement
1 – Pulsefind if the measurement mode changed.
Default: 0

lDutCycl Flag to indicate whether to perform a duty cycle measurement. This measurement is done using three time measurement markers. It measures the time elapsed from a rising edge to falling edge to rising edge. This measurement is performed tParm.SampCnt number of times.
Valid Entries: 0 – do not perform a Duty Cycle measurement
1 – perform a Duty Cycle measurement.
Default: 0

lGood Flag indicates valid output data in structure.

lMeasCnt Number of hits in measured normal data.

dMeasMin Minimum period measurement as captured from the latest execution of adjacent cycle jitter measurement.

dMeasMax Maximum period measurement as captured from the latest execution of adjacent cycle jitter measurement.

dMeasAvg Average period measurement as captured from the latest execution of adjacent cycle jitter measurement.

dMeasSig Standard deviation (1σ) of period measurements as captured from the latest execution of the measurement.

lNormCnt	Number of measurements captured in latest adjacent cycle jitter execution.
dNormMin	Minimum measured value of adjacent cycle period deviation. This value indicates the smallest amplitude of period change between two adjacent periods. This value is most likely a negative number indicating that the measurement is actually the largest decrease in period between two adjacent periods.
dNormMax	Maximum measured value of adjacent cycle period deviation. This value indicates the largest amplitude of period change between two adjacent periods. This value is most likely a positive value indicating that this register contains the largest increase in periods between two adjacent periods. To identify the overall largest change in periods, compare the absolute value of dNormMin and dNormMax.
dNormAvg	Average value of adjacent cycle period deviation. This value should be zero indicating that the period amplitude on average is remaining fixed. If this value is something other than zero, the period was shifting during the measurement. In most cases, the period of a clock signal will have instantaneous amplitude deviations (also known as jitter) but on average, the periods tend toward the same amplitude.
dNormSig	Standard deviation (1σ) of adjacent cycle jitter measurements as captured from the latest execution of the measurement.
lTotlCnt	Number of hits in measured accumulated period measurement data. This accumulation is of the absolute period measurements and not the adjacent cycle jitter measurements.
dTotlMin	Minimum period measurement found in the accumulated data.
dTotlMax	Maximum period measurement found in the accumulated data.
dTotlAvg	Average period measurement found in the accumulated data.
dTotlSig	Standard deviation (1σ)of period measurements found in the accumulated data.
lAcumCnt	Number of measurements in adjacent cycle jitter accumulated data.
dAcumMin	Minimum adjacent cycle jitter measurement found in accumulated data.
dAcumMax	Maximum adjacent cycle jitter measurement found in accumulated data.
dAcumAvg	Average value of adjacent cycle jitter found in accumulated data.
dAcumSig	Standard deviation (1σ) of **accumulated** adjacent cycle jitter data.
tNorm	Structure of type PLTD containing all of the necessary information to draw a Histogram of latest adjacent cycle jitter measurements from most recent execution. See Section 2-3 for details of the PLTD structure and its elements.
tAcum	Structure of type PLTD containing all of the necessary information to draw a Histogram of accumulated data from all adjacent cycle acquisitions. See Section 2-3 for details of the PLTD structure and its elements.
tMaxi	Structure of type PLTD containing all of the necessary information to draw a Histogram with the maximum number of occurrences of a given measurement in all previous executions of adjacent cycle jitter. See Section 2-3 for details of the PLTD structure and its elements.
tBath	Structure of type PLTD containing all of the necessary information to draw a Bathtub curve based on the Probability Density Function (PDF) of DJ and RJ as measured by the TailFit

routine (if enabled.) The data in this structure is only valid when a successful tail-fit has been performed. See Section 2-3 for details of the PLTD structure and its elements.

tEftv Structure of type PLTD containing all of the necessary information to draw an Effective Jitter Bathtub curve based on the amplitude of effective DJ and effective RJ. The data in this structure is only valid if lFndEftv is set and a valid fit is obtained. See Section 2-3 for details of the PLTD structure and its elements.

tTfit Structure of type TFIT containing all of the TailFit information (including plot and limits.) This structure is only valid when a successful tail-fit has been performed. See Section 2-3 for details of the TFIT structure and its elements.

lBinNumb, **dLtSigma**, **dRtSigma**, **dFreq** Used internally, DO NOT ALTER!

void __stdcall FCNL_DefAcyc (ACYC *acyc)

This function is used to fill the acyc structure for the Adjacent Cycle Jitter tool with reasonable default values. It is recommended that this function be called initially even if parameters within the structure are to be adjusted manually, and may be called repeatedly to reestablish initial conditions; however, this will impact test time.

Before calling this function, zero out the ACYC structure using the standard memset() function to ensure that any information pertaining to dynamic memory allocation is cleaned out prior to using the structure.

INPUTS

acyc - Pointer to a ACYC structure. Memory needs to be allocated by the caller.

OUTPUTS

None.

void __stdcall FCNL_ClrAcyc (ACYC *acyc)

This function frees any dynamic memory that may have been allocated during previous data acquisitions and clears out the acyc structure.

INPUTS

acyc - Pointer to a ACYC structure. Memory needs to be allocated by the caller.

OUTPUTS

None.

EXAMPLE

```
#define TRUE 1
#define FALSE 0
static ACYC cyc2cyc;                             //declare cyc2cyc to be a structure of
                                                 //type ACYC
memset ( &cyc2cyc, 0, sizeof ( ACYC ) );         //clear the memory for cyc2cyc
FCNL_DefAcyc (&cyc2cyc);                         //set histogram structures to default
                                                 //values
cyc2cyc.tparm.lChanNum = 1;                      //capture waveform on channel 1
cyc2cyc.tparm.lSampCnt = 10,000;                 //measure 10,000 samples per burst
cyc2cyc.lTailFit = TRUE;                         //indicate TailFit desired
cyc2cyc.lMinHits = 50,000;                       //don't attempt a TailFit until at least
                                                 //50,000 measurements have been
                                                 //accuired.
cyc2cyc.lDutCycl = TRUE;                         //Measure true duty cycle my measuring
                                                 //successive edges.
FCNL_RqstPkt ( ApiDevId, &cyc2cyc, WIND_ACYC );  //execute the measurement.
FCNL_RqstAll ( ApiDevId, & cyc2cyc, WIND_ACYC ); //get plot data
//print the worst case period decrease between two adjacent cycles.
printf("Maximum Period Decrease in sample is %d\n",ABS(cyc2cyc.dNormMin));
//print the worst case period increase between two adjacent cycles within the sample.
printf("Maximum Period Increase in sample is %d\n",ABS(cyc2cyc.dNormMax));
FCNL_ClrAcyc (&cyc2cyc);                         //deallocate the structure
```

2-17 CLOCK ANALYSIS TOOL

This tool combines a few different measurement tools in the SIA-3000. By doing this, a large number of useful results can be displayed quickly. The lMeas parameter allows you to toggle on or off certain measurements. The measurement settings provide the best configuration to a variety of users.

This ease of use means that there is less control over individual settings. There may be instances where there is the need to have more control over a specific measurement. An example would be changing the trigger delay on the oscilloscope, or measuring a histogram over two periods rather than single period jitter. Another example would be to find very low frequency jitter below the (clock/1667) low cutoff frequency of this tool. If you need access to more configuration settings, use one of the individual tools instead.

```
typedef struct
  {
  PARM    tParm;              /* Contains acquisition parameters       */
  long    lPass;              /* Acquisitions so far, set to 0 to reset */
  long    lPcnt;              /* Amount +/- 50% to calc. rise/fall time */
  long    lHiRFmV;            /* Absolute rise/fall voltage if lPcnt<0  */
  long    lLoRFmV;            /* Absolute rise/fall voltage if lPcnt<0  */
  long    lMeas;              /* Measure flag, see defines above        */
  long    lInps;              /* Input selection, see defines above     */
  double  dAttn[POSS_CHNS];   /* Attenuation factor (dB) - per channel  */
  long    lGood;              /* Flag indicates valid data in structure */
  long    lPad0;
  long    lHistCnt[POSS_CHNS];/* Number of hits in accumulated edge data*/
  double  dHistMin[POSS_CHNS];/* Minimum value in accumulated edge data */
  double  dHistMax[POSS_CHNS];/* Maximum value in accumulated edge data */
  double  dHistAvg[POSS_CHNS];/* Average value of accumulated edge data */
  double  dHistSig[POSS_CHNS];/* 1-Sigma value of accumulated edge data */
  double  dPwPl[POSS_CHNS];   /* Pulsewidth plus                        */
  double  dPwMn[POSS_CHNS];   /* Pulsewidth minus                       */
  double  dFreq[POSS_CHNS];   /* Carrier frequency                      */
  double  dDuty[POSS_CHNS];   /* Duty Cycle                             */
  double  dPjit[POSS_CHNS];   /* Periodic jitter on N-clk basis         */
  double  dCorn[POSS_CHNS];   /* Corner Frequency used for measurement  */

  long    lBinNumb[POSS_CHNS];/*****************************************/
  double  dWndFact[POSS_CHNS];/*  These values are all used internally  */
  double  dLtSigma[POSS_CHNS][PREVSIGMA];/*  DO NOT ALTER!              */
  double  dRtSigma[POSS_CHNS][PREVSIGMA];/******************************/

  QTYS    qNorm[POSS_CHNS];   /* Normal channel quantities              */
  QTYS    qComp[POSS_CHNS];   /* Complimentary channel  quantities      */
  QTYS    qDiff[POSS_CHNS];   /* Differential quantities                */
  QTYS    qComm[POSS_CHNS];   /* Common (A+B) quantities                */
  TFIT    tTfit[POSS_CHNS];   /* Structure containing tailfit info     */

  long    lPeakNumb[POSS_CHNS];/* Count of detected spikes              */
  long    lPeakRsvd[POSS_CHNS];/* Used to track memory allocation       */
  long   *lPeakData[POSS_CHNS];/* Tracks detected spikes in RJ+PJ data  */

  PLTD    tNorm[POSS_CHNS];   /* Normal channel voltage data            */
  PLTD    tComp[POSS_CHNS];   /* Complimentary channel voltage data     */
  PLTD    tDiff[POSS_CHNS];   /* Differential voltage data              */
  PLTD    tComm[POSS_CHNS];   /* Common (A+B) voltage data              */
  PLTD    tHist[POSS_CHNS];   /* Histogram of all acquires combined     */
  PLTD    tShrt[POSS_CHNS];   /* Total Jitter for SHORT Cycles          */
  PLTD    tLong[POSS_CHNS];   /* Total Jitter for LONG Cycles           */
  PLTD    tBoth[POSS_CHNS];   /* Total Jitter for LONG & SHORT Cycles   */
```

```
PLTD    tFftN[POSS_CHNS];   /* Frequency plot data on 1-clock basis   */
PLTD    tSave[POSS_CHNS];   /* Average Frequency plot before scaling  */
} CANL;
```

tParm A structure of type PARM that contains acquisition parameter. The PARM is discussed in full detail in Section 2-4.

lPassCnt This parameter is a bi-directional structure element that tracks the number of acquisitions since last reset. This flag can be read after an execution or set prior to an execution. Setting this parameter to 0 essentially resets this register. It will be automatically incremented when a measurement is performed.
Valid Entries: any integer greater than or equal to 0
Default: 0

lPcnt This field specifies the voltage thresholds to be used when calculating rise and fall times. The voltage thresholds are assumed to be symmetrical about the 50% threshold, and this is the distance from the 50% threshold to the starting and ending thresholds. For example if this field is equal to 30, then 20% and 80% thresholds are used. If this field is equal to 40, then 10% and 90% thresholds are used. The absolute voltage levels used are based on the previous pulsefind minimum and maximum voltages. If this field is negative, then the absolute rise and fall thresholds are taken from the following fields lHiRFmV and lLoRFmv.
Default: 30

lHiRFmV Absolute rise/fall voltage if lPcnt<0, in units of mV
Default: +250

lLoRFmV Absolute rise/fall voltage if lPcnt<0, in units of mV
Default: -250

lMeas Measure flag, this is a bitfield which may be created by combining any or all of the following constants:
CANL_MEAS_RISEFALL – Rise and Fall times are calculated
CANL_MEAS_VTYPICAL – Vtop and Vbase are calculated
CANL_MEAS_VEXTREME – Vmin and Vmax are calculated
CANL_MEAS_OVERUNDR – Overshoot and Undershoot are calculated
CANL_MEAS_WAVEMATH – Vavg and Vrms are calculated
CANL_MEAS_TAILFITS – Enables Histogram tailfits
CANL_MEAS_PERIODIC – Yields Hi-Freq Mod. results
Default: All of the above are included

dAttn[n] Attenuation factor in dB, this is provided to allow the results to be scaled to compensate for external attenuation from sources such as probes.
Default: 0

lGood Flag indicates valid data in structure

lHistCnt[n] Number of hits in accumulated edge data, per channel

dHistMin[n] Minimum value in accumulated edge data, per channel

dHistMax[n] Maximum value in accumulated edge data, per channel

dHistAvg[n] Average value of accumulated edge data, per channel

dHistSig[n] 1-Sigma value of accumulated edge data, per channel

dPwPl[n] Pulsewidth plus, per channel

dPwMn[n] Pulsewidth minus, per channel

dFreq[n] Carrier frequency, per channel

dDuty[n] Duty Cycle, per channel

dPjit[n] Periodic jitter on N-clk basis, per channel

dCorn[n] Corner Frequency used for measurement, per channel

lBinNumb[n],dWndFact[n],dLtSigma[n][m],dRtSigma[n][m] These values are for internal use only, DO NOT ALTER or try to use.

qNorm[n] + Input channel quantities, per channel
qComp[n] - Input channel quantities, per channel
qDiff[n] Differential quantities, per channel
qComm[n] Common (A+B) quantities, per channel
tTfit[n] Structure containing tailfit info, per channel

lPeakNumb[n] Count of detected spikes, per channel
lPeakRsvd[n] Used to track memory allocation, per channel
lPeakData[n] Tracks detected spikes in RJ+PJ data, per channel

tNorm[n] Normal channel voltage data, per channel
tComp[n] Complimentary channel voltage data, per channel
tDiff[n] Differential voltage data, per channel
tComm[n] Common (A+B) voltage data, per channel
tHist[n] Histogram of all acquires combined, per channel
tShrt[n] Total Jitter for SHORT Cycles, per channel
tLong[n] Total Jitter forCycles, per channel
tBoth[n] Total Jitter for& SHORT Cycles, per channel
tFftN[n] Frequency data on 1-clock basis, per channel
tSave[n] Average Frequency before scaling, per channel

void __stdcall FCNL_DefCanl (CANL *canl)

This function is used to fill the canl structure for the Clock Analysis tool with reasonable default values. It is recommended that this function be called initially even if parameters within the structure are to be adjusted manually, and may be called repeatedly to reestablish initial conditions; however, this will impact test time.

Before calling this function, zero out the CANL structure using the standard memset() function to ensure that any information pertaining to dynamic memory allocation is cleaned out prior to using the structure.

INPUTS

canl - Pointer to a CANL structure. Memory needs to be allocated by the caller.

OUTPUTS

None.

void __stdcall FCNL_ClrCanl (CANL *canl)

This function frees any dynamic memory that may have been allocated during previous data acquisitions and clears out the canl structure.

INPUTS

canl - Pointer to a CANL structure. Memory needs to be allocated by the caller.

OUTPUTS

None.

EXAMPLE

```
static CANL clk;                                    //declare clk to a structure of type
                                                    //CANL
memset ( &clk, 0, sizeof ( CANL ) );                //clear the memory for clk structure
FCNL_DefCanl ( &clk);                               //set clk structures to default values
FCNL_RqstPkt ( ApiDevId, &clk, WIND_CANL );         //execute the measurement.
FCNL_RqstAll ( ApiDevId, &clk, WIND_CANL );         //get plot data
FCNL_ClrCanl ( &clk);                               //deallocate the structure
```

2-18 CLOCK STATISTICS TOOL

The Statistics panel displays the results of several basic clock parameters: mean, minimum, maximum, 1-sigma, peak-to-peak, hits, frequency and duty cycle. Also displayed are the measured Vstart, Vstop as well as the Vp-p, Vmax and Vmin of the input channels.

The Statistics panel provides a summary of the statistics from a single histogram of measurements of the chosen function (period, rise-time, fall-time, positive pulse width and negative pulse width). The tool reports the clock frequency with 9 digits of precision. Duty cycle is displayed in this tool.

```
typedef struct
  {
  /* Input parameters  */
  PARM    tParm;              /* Contains acquisition parameters        */
  long    lPfnd;              /* Force a pulse-find before each measure */
  long    lQckMeas;           /* If true skip frequency and voltages    */
  /* Output parameters */
  long    lGood;              /* Flag indicates valid data in structure */
  long    lPad1;
  double  dPwPavg;            /* Contains the PW+ average value         */
  double  dPwPdev;            /* Contains the PW+ 1-Sigma value         */
  double  dPwPmin;            /* Contains the PW+ minimum value         */
  double  dPwPmax;            /* Contains the PW+ maximum value         */
  double  dPwMavg;            /* Contains the PW- average value         */
  double  dPwMdev;            /* Contains the PW- 1-Sigma value         */
  double  dPwMmin;            /* Contains the PW- minimum value         */
  double  dPwMmax;            /* Contains the PW- maximum value         */
  double  dPerPavg;           /* Contains the PER+ average value        */
  double  dPerPdev;           /* Contains the PER+ 1-Sigma value        */
  double  dPerPmin;           /* Contains the PER+ minimum value        */
  double  dPerPmax;           /* Contains the PER+ maximum value        */
  double  dPerMavg;           /* Contains the PER- average value        */
  double  dPerMdev;           /* Contains the PER- 1-Sigma value        */
  double  dPerMmin;           /* Contains the PER- minimum value        */
  double  dPerMmax;           /* Contains the PER- maximum value        */

  double  dDuty;              /* Contains the returned duty cycle       */
  double  dFreq;              /* Contains the carrier frequency         */
  double  dVmin;              /* Pulse-find Min voltage                 */
  double  dVmax;              /* Pulse-find Max voltage                 */
  } CLOK;
```

tParm	A structure of type PARM that contains acquisition parameter. The PARM is discussed in full detail in Section 2-4.
lPfnd	If true force a pulse-find before each measure
lQckMeas	If true skip frequency and voltages
lGood	Flag indicates valid output data in structure.
dPwPavg	Contains the PW+ average value
dPwPdev	Contains the PW+ 1-Sigma value
dPwPmin	Contains the PW+ minimum value
dPwPmax	Contains the PW+ maximum value
dPwMavg	Contains the PW- average value
dPwMdev	Contains the PW- 1-Sigma value
dPwMmin	Contains the PW- minimum value
dPwMmax	Contains the PW- maximum value
dPerPavg	Contains the PER+ average value

dPerPdev	Contains the PER+ 1-Sigma value
dPerPmin	Contains the PER+ minimum value
dPerPmax	Contains the PER+ maximum value
dPerMavg	Contains the PER- average value
dPerMdev	Contains the PER- 1-Sigma value
dPerMmin	Contains the PER- minimum value
dPerMmax	Contains the PER- maximum value
dDuty	Contains the returned duty cycle
dFreq	Contains the carrier frequency
dVmin	Pulse-find Min voltage
dVmax	Pulse-find Max voltage

void __stdcall FCNL_DefClok (CLOK *clok)

This function is used to fill the clok structure for the Clock Statistics tool with reasonable default values. It is recommended that this function be called initially even if parameters within the structure are to be adjusted manually, and may be called repeatedly to reestablish initial conditions; however, this will impact test time.

Before calling this function, zero out the CLOK structure using the standard memset() function to ensure that any information pertaining to dynamic memory allocation is cleaned out prior to using the structure.

INPUTS

clok - Pointer to a CLOK structure. Memory needs to be allocated by the caller.

OUTPUTS

None.

void __stdcall FCNL_ClrClok (CLOK *clok)

This function frees any dynamic memory that may have been allocated during previous data acquisitions and clears out the clok structure.

INPUTS

clok - Pointer to a CLOK structure. Memory needs to be allocated by the caller.

OUTPUTS

None.

EXAMPLE

```
static CLOK clkstat;                                 //declare clkstat to a structure of type
                                                     //CLOK
memset ( &clkstat, 0, sizeof ( CLOK ) );             //clear the memory for clkstat structure
FCNL_DefClok ( &clkstat);                            //set clkstat structures to default values

FCNL_RqstPkt ( ApiDevId, &clkstat, WIND_CLOK );      //execute the measurement.
FCNL_RqstAll ( ApiDevId, &clkstat, WIND_CLOK );      //get plot data

FCNL_ClrClok ( &clkstat);                            //deallocate the structure
```

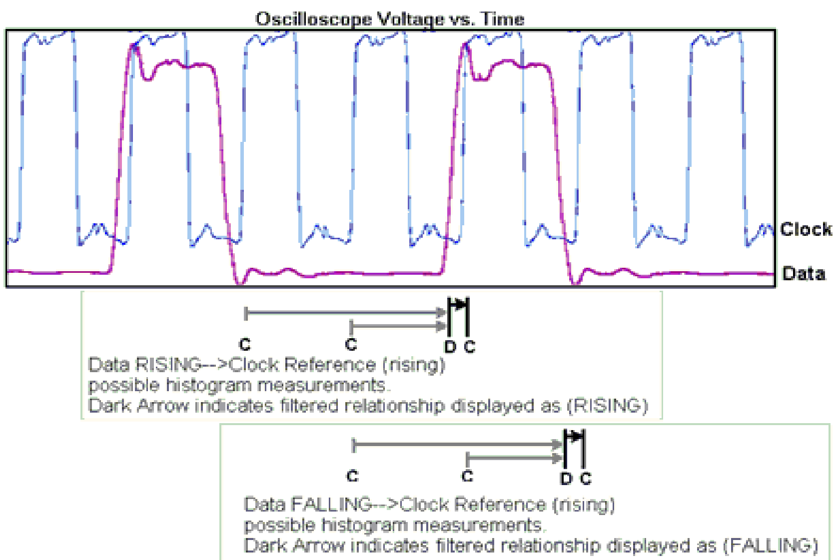
2-19 DATABUS TOOL

With the SIA-3000 Signal Integrity Analyzer and GigaView Databus software, single-ended and differential clock and data signals can be characterized for timing, clock and data jitter, clock-to-data skew, channel-to-channel skew and Bit Error Rate (BER) on up to ten channels in parallel. The analysis is done using one reference clock and up to nine data channels. Users can input the setup and hold specifications. Setup and Hold violations can be measured based on the actual mean of the data histogram referenced to the clock edge.

For each data lane there are two histograms: one showing the transitions before the clock edge and one showing the transitions after the clock edge. The tool also applies statistical long term BER in the form of a bathtub curve. This measurement is used to determine long-term system reliability. If the jitter is too high, the tool will indicate a failure.

The following example shows the Data signal connected to Channel 1 and Bit Clock Signal connected to Channel 2. Therefore, two histograms can be made. One histogram represents a measurement of Data RISING edges to clock reference edge, the other represents Data FALLING edges to the clock reference edge.

These histograms would show many modes or distributions because there are many possible relationships between clock and data edges. These histograms are filtered to show only those times that relate to the measured Data edges closest in time to the Reference Clock Edge.




```
typedef struct
  {
  /* Input parameters  */
  long    lClokChn;          /* Reference Clock channel                */
  long    lChanNum;          /* Bitfield indicating channels to measure*/
  double  dSetTime;          /* Setup time to assess PASS/FAIL         */
  double  dHldTime;          /* Hold time to assess PASS/FAIL          */
  double  dEyeSpec;          /* Eye opening size to assess PASS/FAIL   */
  double  dUserVlt[POSS_CHNS];/* Array of user voltages                */
  EYEH    tDbus;             /* Contains acquisition parameters        */
  /* Output parameters */
```

```
  long    lGood;              /* Flag indicates valid data in structure */
  long    lPad1;
  double  dDutCycl;           /* Duty cycle measurement of clock signal */
  HIST    tHist;              /* Contains output data for clock channel */
  EYEH    tEyeh[POSS_CHNS];   /* Contains output data for enabled chans */
                              /* The following are bitfields indicating */
                              /* PASS/FAIL [0/1] for each channel       */
  long    lTypclSetHldPF;     /* Means of histograms to setup/hold time */
  long    lEyeOpenSpecPF;     /* Eye opening spec (jitter only)         */
  long    lWorstSetHldPF;     /* Histogram means w/jitter to setup/hold */
                              /* The following indicate PASS only if all*/
                              /* selected channels PASS [Pass=1;Fail=0] */
  long    lTypclSetHldAll;    /* Means of histograms to setup/hold time */
  long    lEyeOpenSpecAll;    /* Eye opening spec (jitter only)         */
  long    lWorstSetHldAll;    /* Histogram means w/jitter to setup/hold */
  } DBUS;
```

lClokChn Reference Clock channel
Default: 2

lChanNum Bitfield indicating channels to measure
Default: 1

dSetTime Setup time to assess PASS/FAIL
Default: 5e-10

dHldTime Hold time to assess PASS/FAIL
Default: 5e-10

dEyeSpec Eye opening size to assess PASS/FAIL, in UI
Default: 0.6

dUserVlt[n] Array of user voltages
Default: 0.0

tDbus This is the same structure as is defined in the Random Data With Bitclock tool. It contains all the acquisition parameters that are used for the measurement, with the exception of those defined directly above.
Default: See Random Data With Bitclock Tool

lGood Flag indicates valid data in structure

dDutCycl Duty cycle measurement of clock signal

tHist This is the same structure as is defined for the Histogram Tool. It contains all the output data for the clock channel.

tEyeh[n] This is an array of the same structures as are defined in the Random Data With Bitclock tool. It contains all the output data for each of the channels which a measurement is performed on.

lTypclSetHldPF Means of histograms to setup/hold time, this is a bitfield indicating PASS/FAIL [0/1] for each channel

lEyeOpenSpecPF Eye opening spec, this is a bitfield indicating PASS/FAIL [0/1] for each channel

lWorstSetHldPF Histogram means w/jitter to setup/hold, this is a bitfield indicating PASS/FAIL [0/1] for each channel

lTypclSetHldAll Means of histograms to setup/hold time, this is a bitfield indicating PASS/FAIL [0/1] for each channel

lEyeOpenSpecAll Eye opening spec (jitter only) , this is a bitfield indicating PASS/FAIL [0/1] for each channel

lWorstSetHldAll Histogram means w/jitter to setup/hold, this is a bitfield indicating PASS/FAIL [0/1] for each channel

void __stdcall FCNL_DefDbus (DBUS *dbus)

This function is used to fill the dbus structure for the DataBus tool with reasonable default values. It is recommended that this function be called initially even if parameters within the structure are to be adjusted manually, and may be called repeatedly to reestablish initial conditions; however, this will impact test time.

Before calling this function, zero out the DBUS structure using the standard memset() function to ensure that any information pertaining to dynamic memory allocation is cleaned out prior to using the structure.

INPUTS

dbus - Pointer to a DBUS structure. Memory needs to be allocated by the caller.

OUTPUTS

None.

void __stdcall FCNL_ClrDbus (DBUS *dbus)

This function frees any dynamic memory that may have been allocated during previous data acquisitions and clears out the dbus structure.

INPUTS

dbus - Pointer to a DBUS structure. Memory needs to be allocated by the caller.

OUTPUTS

None.

EXAMPLE

```
static DBUS databus;                              //declare clkstat to a structure of type
                                                  //DBUS
memset ( &databus, 0, sizeof ( DBUS ) );          //clear the memory for databus structure
FCNL_DefDbus ( &databus);                         //set databus structures to default values

FCNL_RqstPkt ( ApiDevId, &databus, WIND_DBUS );  //execute the measurement.
FCNL_RqstAll ( ApiDevId, &databus, WIND_DBUS );  //get plot data in tEyeh[n]

FCNL_ClrDbus ( &databus);                         //deallocate the structure
```

2-20 DATACOM BIT CLOCK AND MARKER TOOL

This tool can operate either with the Clock Recovery option installed or with an external bit clock applied to another input. A pattern marker is necessary and is possibly derived from the data pattern generator. But, in many cases, this signal is not externally available and it is useful to have the SIA-3000 Pattern Marker (PM50) option. The pattern requirements are such that it needs to be a repeating pattern.

```
typedef struct
  {
  PARM    tParm;               /* Contains acquisition parameters        */
  char    sPtnName[ 128 ];     /* Name of pattern file to be used        */
  long    lPassCnt;            /* Acquisitions so far, set to 0 to reset */
  long    lHeadOff;            /* Header offset, external arming only    */
  long    lFftMode;            /* 0=NoFFT, 1=Fc/1667, 2=Use dCornFrq     */
  long    lMinHits;            /* Minimum hits before trying tail-fit    */
  long    lTailFit;            /* If non-zero a tail-fit will be tried   */
  long    lErrProb;            /* Error probability for Total Jitter     */
                               /* Valid range is ( -1 to -16 )           */
  double  dBitRate;            /* Bit Rate, may be specified or measured */
  double  dCornFrq;            /* Corner Frequency for RJ+PJ             */
  double  dMaxSerr;            /* LIM_ERROR if this std. error exceeded  */
  long    lGood;               /* Flag indicates valid data in structure */

  long    lBinNumb;            /*****************************************/
  long    lMaxStop;            /*                                       */
  long    lPtnRoll;            /*                                       */
  long    lFallAdj;            /*  These values are all used internally  */
  long    lClokAdj;            /*   as part of the measurement process   */
  long    lLeftCnt;            /*             DO NOT ALTER!              */
  long    lRghtCnt;            /*                                       */
  double  dWndFact;            /*                                       */
  double  dDdjMove;            /*                                       */
  double  dLtSigma[PREVSIGMA];/*                                        */
  double  dRtSigma[PREVSIGMA];/*****************************************/

  double  dHistMed;            /* Total Jitter Histogram median location */
  double  dLeftMed;            /* Left Edge Histogram median location    */
  double  dRghtMed;            /* Right Edge Histogram median location   */
  long    lAcumHit;            /* Accumulated Histogram hits             */
  long    lPassHit;            /* Histogram hits for this pass only      */
  TFIT    tTfit;               /* Structure containing tail-fit info     */

  PATN    tPatn;               /* Internal representation of pattern     */
  long    lPeakNumb;           /* Count of detected spikes               */
  long    lPeakRsvd;           /* Used to track memory allocation        */
  long   *lPeakData;           /* Tracks detected spikes in RJ+PJ data   */
  long    lDdjtRsvd;           /* Used to track memory allocation        */
  DDJT   *tDdjtData;           /* Raw DCD+DDJ measurements               */
  long    lPad1;

  PLTD    tRiseHist;           /* DCD+DDJ histogram of rising edges      */
  PLTD    tFallHist;           /* DCD+DDJ histogram of falling edges     */
  PLTD    tNormDdjt;           /* DCD+DDJvsUI for external arming only   */
  PLTD    tTotlHist;           /* Histogram of all acquires combined     */
  PLTD    tLeftHist;           /* Leftmost Histogram                     */
  PLTD    tRghtHist;           /* Rightmost Histogram                    */
  PLTD    tBathPlot;           /* Bathtub curves determined from PDF     */
  PLTD    tSigmPlot;           /* 1-Sigma vs. span plot                  */
  PLTD    tFreqPlot;           /* Jitter vs. frequency plot              */
  } RCPM;
```

tParm A structure of type PARM that contains acquisition parameters. The PARM structure is discussed in full detail in Section 2-4.

sPtnName A character array containing the name of pattern file to be used, the file must exist in the pattern directory (C:\VISI\) on the SIA3000 or else an error will be returned. The first time a measurement is performed the pattern is loaded into structure tPatn.
Valid Entries: a valid file name (including extension)
Default: "k285.ptn"

lPassCnt This parameter is a bi-directional structure element that tracks the number of acquisitions since last reset. This flag can be read after an execution or set prior to an execution. Setting this parameter to 0 essentially resets this register. It will be automatically incremented when a measurement is performed.
Valid Entries: any integer greater than or equal to 0
Default: 0

lHeadOff Header offset parameter, for use in packet-ized data which may have a frame header before the test pattern. This offset value can be used to skip past header information and into the repeating data pattern stream. This can be useful when analyzing data from disk drives when the pattern marker may be synchronized with the start of frame data.
Valid Entries: 0 to 10,000,000-pattern length I
Default: 0 (indicating no header present)

lFftMode 0=NoFFT, 1=Fc/1667, 2=Use dCornFrq
Default: 0

lMinHits Minimum hits before trying tail-fit
Default: 0

lTailFit If non-zero a tail-fit will be tried
Default: 1

lErrProb Error probability level for Total Jitter. Total Jitter is calculated based on the desired Error Probability level. This value is used in conjunction with the bathtub curve after the successful completion of a tail-fit in order to project the value of Total Jitter.
Valid Entries: -1 to -16
Default: -12

dBitRate Bit Rate, may be specified or measured
Default: 2.5e9

dCornFrq Corner Frequency for RJ & PJ estimate in Hertz. This value is used in conjunction with the Bit Rate and pattern to determine the maximum stop count to be used to acquire RJ & PJ data. A lower value increase acquisition time.
Valid Entries: Bit-Rate /10,000,000 to Bit-Rate I
Default: 637e3 (637kHz – Fibre Channel 1X)

dMaxSerr An error is returned if this std. error is exceeded
Default: 0.5

lGood Flag indicates valid data in structure

lBinNumb,lMaxStop,lPtnRoll,lFallAdj,lClokAdj,lLeftCnt,lRghtCnt
dWndFact,dDdjMove,dLtSigma[n],dRtSigma[n] These values are for internal use only, DO NOT ALTER or try to use.

dHistMed Total Jitter Histogram median location

dLeftMed Left Edge Histogram median location

dRghtMed Right Edge Histogram median location

lAcumHit Accumulated Histogram hits

lPassHit Histogram hits for this pass only

tTfit	`Structure containing tail-fit info`
tPatn	`Internal representation of pattern`
lPeakNumb	`Count of detected spikes`
lPeakRsvd	`Used to track memory allocation`
lPeakData	`Tracks detected spikes in RJ+PJ data`
lDdjtRsvd	`Used to track memory allocation`
tDdjtData	`Raw DCD+DDJ measurements`
tRiseHist	`DCD+DDJ histogram of rising edges`
tFallHist	`DCD+DDJ histogram of falling edges`
tNormDdjt	`DCD+DDJvsUI for external arming only`
tTotlHist	`Histogram of all acquires combined`
tLeftHist	`Leftmost Histogram`
tRghtHist	`Rightmost Histogram`
tBathPlot	`Bathtub curves determined from PDF`
tSigmPlot	`1-Sigma vs. span plot`
tFreqPlot	`Jitter vs. frequency plot`

void __stdcall FCNL_DefRcpm (RCPM *rcpm)

This function is used to fill the rcpm structure for the Datacom Bit Clock and Marker tool with reasonable default values. It is recommended that this function be called initially even if parameters within the structure are to be adjusted manually, and may be called repeatedly to reestablish initial conditions; however, this will impact test time.

Before calling this function, zero out the RCPM structure using the standard memset() function to ensure that any information pertaining to dynamic memory allocation is cleaned out prior to using the structure.

INPUTS

rcpm - Pointer to a RCPM structure. Memory needs to be allocated by the caller.

OUTPUTS

None.

void __stdcall FCNL_ClrRcpm (RCPM *rcpm)

This function frees any dynamic memory that may have been allocated during previous data acquisitions and clears out the rcpm structure.

INPUTS

rcpm - Pointer to a RCPM structure. Memory needs to be allocated by the caller.

OUTPUTS

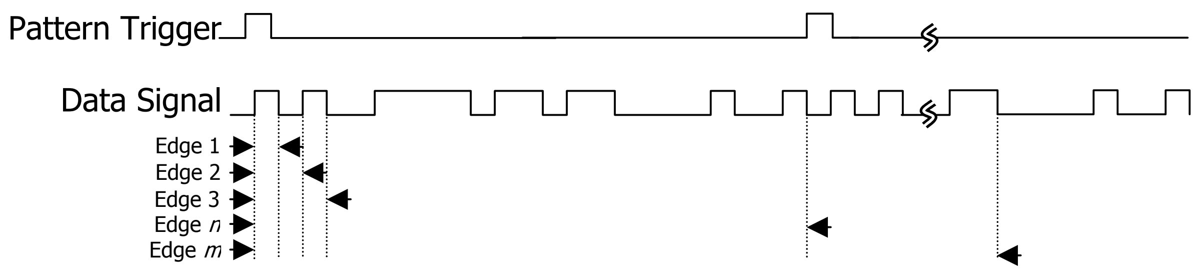
None.

EXAMPLE

```
static RCPM bcam;                                       //declare bcam to a structure of type
                                                        //RCPM
memset ( &bcam, 0, sizeof ( RCPM ) );                   //clear the memory for bcam structure
FCNL_DefRcpm ( &bcam);                                  //set bcam structures to default values

FCNL_RqstPkt ( ApiDevId, &bcam, WIND_RCPM );            //execute the measurement.
FCNL_RqstAll ( ApiDevId, &bcam, WIND_RCPM );            //get plot data

FCNL_ClrRcpm ( &bcam);                                  //deallocate the structure
```

2-21 DATACOM KNOWN PATTERN WITH MARKER TOOL

The Datacom Known Pattern With Marker Tool is used to measure jitter on serial communication signals. This tool is not protocol specific and works with all communication standards that rely on jitter separation to define jitter limits for compliance. Such standards include: Fibre Channel, Gigabit Ethernet, the XAUI layer of 10G Ethernet, SFI 4, SFI 5, XFP, RapidIO, PCI Express and Serial ATA. This tool requires that a pattern trigger be available either externally from the test environment or internally from the PM50. Measurements are made based on this diagram. Each measurement is from the first edge after the pattern trigger to each subsequent edge in the pattern. DDJ is based on edges 1 through *n*, where *n* is the last edge in the pattern. PJ and RJ estimates are based on edges 1 through *m* where *m* is last edge measured based on the prescribed cutoff frequency.




```
typedef struct
  {
  /* Input parameters  */
  PARM    tParm;              /* Contains acquisition parameters        */
  char    sPtnName[ 128 ];    /* Name of pattern file to be used        */
  long    lAcqMode;           /* Mask defining modes for RJ+PJ acquire  */
                              /* Bit3:PW- Bit2:PW+ Bit1:Per- Bit0:Per+  */
  long    lRndMode;           /* Enable random mode, auto-arming only   */
  long    lQckMode;           /* Enable quick mode, external arm only   */
  long    lIntMode;           /* Interpolation mode, non-zero is linear */
  long    lGetRate;           /* If non-zero Bit Rate will be measured  */
                              /* Not valid for random mode              */
  long    lTailFit;           /* Count of tailfits, see constants above */
                              /* Not valid when auto-arming             */
  long    lErrProb;           /* Error probability for Total Jitter     */
                              /* Valid range is ( -1 to -16 )           */
  long    lPassCnt;           /* Acquisitions so far, set to 0 to reset */
  long    lFftAvgs;           /* 2^fft_avgs averages used to smooth FFT */
  long    lFitPcnt;           /* Automode suceed %, see constants above */

  SPEC    tRateInf;           /* Parameters to acquire Bit Rate         */
  SPEC    tDdjtInf;           /* Parameters to acquire DCD+DDJ          */
  SPEC    tRjpjInf;           /* Parameters to acquire RJ+PJ            */

                              /* Negative values disable these filters  */
  double  dDdjtLpf;           /* Low pass DCD+DDJ filter frequency      */
  double  dDdjtHpf;           /* High pass DCD+DDJ filter frequency     */
  double  dRjpjFmn;           /* Minimum integration limit for RJ+PJ    */
  double  dRjpjFmx;           /* Maximum integration limit for RJ+PJ    */

  double  dBitRate;           /* Bit Rate, may be specified or measured */
  double  dCornFrq;           /* Corner Frequency for RJ+PJ             */
  long    lHeadOff;           /* Header offset, external arming only    */

  long    lFndEftv;           /* Flag to attempt effective jitter calc  */
  long    lMinEftv;           /* Min probability for effective fit: -4  */
  long    lMaxEftv;           /* Max probability for effective fit: -12 */
```

```
long    lFiltEnb;          /* Enable IDLE character insertion filter */
long    lQckTjit;          /* Fast total jitter calc - no bathtubs!  */
long    lTfitCnt;          /* Sample count per pass when tailfitting */
/* Output parameters */
long    lGood;             /* Flag indicates valid data in structure */
PATN    tPatn;             /* Internal representation of pattern     */

double  dWndFact;          /*****************************************/
long    lMaxStop;          /*  These values are all used internally  */
long    lCmpMode;          /*                                        */
long    lPosRoll;          /*            DO NOT ALTER!               */
long    lNegRoll;          /*                                        */
long    lAdjustPW[ 2 ];    /*****************************************/

DDJT   *tDdjtData;         /* Raw DCD+DDJ measurements               */
long    lDdjtRsvd;         /* Used to track memory allocation        */
double *dMeasData[ 2 ];    /* Raw allmeas histogram when auto-arming */
long    lMeasRsvd[ 2 ];    /* Used to track memory allocation        */
double *dRjpjData[ 4 ];    /* Raw variance data                      */
long    lRjpjRsvd[ 4 ];    /* Used to track memory allocation        */
double *dTfitData[ 4 ];    /* Raw tail-fit data if used              */
long    lTfitRsvd[ 4 ];    /* Used to track memory allocation        */
long   *lPeakData[ 4 ];    /* Tracks detected spikes in RJ+PJ data   */
long    lPeakNumb[ 4 ];    /* Count of detected spikes               */
long    lPeakRsvd[ 4 ];    /* Used to track memory allocation        */
double *dFreqData[ 4 ];    /* Raw FFT output when averaging          */
long    lFreqRsvd[ 4 ];    /* Used to track memory allocation        */
double *dTailData[ 4 ];    /* Raw tailfit FFT output when averaging  */
long    lTailRsvd[ 4 ];    /* Used to track memory allocation        */

long    lHits;             /* Total samples for DDJT+RJ+PJ combined  */
long    lPad2;
double  dDdjt;             /* DCD+DDJ jitter                         */
double  dRang;             /* Pk-Pk of allmeas histogram for auto-arm*/
double  dRjit[ 4 ];        /* Random jitter, for enabled modes       */
double  dPjit[ 4 ];        /* Periodic jitter, for enabled modes     */
double  dTjit[ 4 ];        /* Total jitter, for enabled modes        */
double  dEftvLtDj[ 4 ];    /* Effective jitter when enabled          */
double  dEftvLtRj[ 4 ];
double  dEftvRtDj[ 4 ];
double  dEftvRtRj[ 4 ];

PLTD    tRiseHist;         /* DCD+DDJ histogram of rising edges      */
PLTD    tFallHist;         /* DCD+DDJ histogram of falling edges     */
PLTD    tRiseMeas;         /* Rising  allmeas histo. auto-arm only   */
PLTD    tFallMeas;         /* Falling allmeas histo. auto-arm only   */
PLTD    tNormDdjt;         /* DCD+DDJvsUI for external arming only   */
PLTD    tHipfDdjt;         /* High Pass Filtered DCD+DDJvsUI         */
PLTD    tLopfDdjt;         /* Low  Pass filtered DCD+DDJvsUI         */
PLTD    tBathPlot[ 4 ];    /* Bathtub plots, for enabled modes       */
PLTD    tEftvPlot[ 4 ];    /* Effective Bathtub plots, if enabled    */
PLTD    tSigmNorm[ 4 ];    /* 1-Sigma plots, for enabled modes       */
PLTD    tSigmTail[ 4 ];    /* 1-Sigma tail-fits, for enabled modes   */
PLTD    tFreqNorm[ 4 ];    /* Frequency plots, for enabled modes     */
PLTD    tFreqTail[ 4 ];    /* Tail-fit FFT plots, for enabled modes  */
} DCOM;
```

tParm A structure of type PARM that contains acquisition parameters. The PARM structure is discussed in full detail in Section 2-4.

sPtnName A character array containing the name of pattern file to be used, the file must exist in the pattern directory (C:\VISI\) on the SIA3000 or else an error will be returned. The first time a measurement is performed the pattern is loaded into structure tPatn.

```
Valid Entries: a valid file name (including extension)
Default:       "k285.ptn"
```

lAcqMode Measurement mode for Random Jitter (RJ) and Periodic Jitter (PJ) estimate. To calculate RJ and PJ, variance data for each transition must be captured. This variance data is then passed through an FFT to create the frequency response. Since rise time and fall time may be asymmetrical, bogus frequency components could be inserted into the RJ & PJ records if both rising and falling edges were used in the data records. Since the frequency response will be calculated based on the records, the slew rate effect must be eliminated from the data. To do this, we force the measurement to either capture only rising edges or falling edges for this data record. For completeness, the start of the measurement could be either a rising or a falling edge. This parameter allows the user to select the polarity of both the reference edge and the measured edge in the data signal. The user can select all permutations of rising and falling edges. This parameter is parsed as a 4-bit binary value with each bit representing a possible permutation. A value of b1111 would indicate that the measurement is to be run using all permutations.

```
Valid Entries: b0001 – rising edge to rising edge
               b0010 – falling edge to falling edge
               b0100 – rising edge to falling edge
               b1000 – falling edge to rising edge
Default:       b0001 – rising edge to rising edge
```

lRndMode Parameter used to enable Random Mode. This parameter is only used in conjunction with RAND structures as used in the Random Data Tool. This parameter enables random mode, valid when auto-arming only. Setting this parameter to 1 will enable Random Mode.

```
Valid Entries: 0 – disable random data mode
               1 – enable random data mode
Default:       0
```

lQckMode Parameter used to enable Quick Mode. QuickMode uses a sparse sample of data points for the PJ and RJ estimates. In this mode, the accuracy of these estimates is greatly reduced depending on the application. Setting this structure element to 1 enables quick mode, valid with external arm only.

```
Valid Entries: 0 – disable quick capture mode
               1 – enable quick capture mode
Default:       0
```

lIntMode Parameter used to enable linear Interpolation mode for RJ & PJ estimate. RJ & PJ are calculated based on the frequency data of the noise. Since data points are captured only on the single polarity transitions, interpolation must be performed between sample points. There are two types of interpolation available in the SIA3000: linear and cubic. Setting this parameter to 1 will enable linear interpolation; otherwise, cubic interpolation will be used.

```
Valid Entries: 0 – use cubic interpolation in FFT data
               1 – use linear interpolation in FFT data
Default:       0
```

lGetRate Parameter used to enable Bit Rate measurement. Knowledge of the pattern enables the instrument to measure from one transition in the pattern to the same edge several pattern repeats later. If this function is disabled, an appropriate value must be supplied in dBitRate variable. This function is NOT available when using random mode.

Valid Entries: 0 – use user specified bit rate
1 – measure bit rate from data

Default: 0

lTailFit Parameter used to enable TailFit algorithm for RJ estimate. The TailFit algorithm yields the highest level of accuracy when calculating an RJ estimate. However, millions of samples must be taken in order to perform an accurate TailFit. Valid with external arm only. The number of TailFits to be performed is based on the value assigned to this parameter. In practice, only a small sampling of edges need to be analyzed for RJ content. The smallest sample is three. The edges selected are the first edge in the pattern, the middle edge and the last edge. This allows a reasonable span of frequency content. It is assumed that the noise components can be approximated by a continuous function (as is generally the case.) If the RJ changes over frequency, there will be a delta between the different samples. A change in value of less than 5% between adjacent points is considered acceptable. If the delta is larger, more TailFit points should be taken.

Valid Entries:	DCOM_NONE	Do not perform a TailFit
	DCOM_AUTO	Perform TailFits until the delta Between successive fits < 5%.
	DCOM_FIT3	Perform 3 TailFits
	DCOM_FIT5	Perform 5 TailFits
	DCOM_FIT9	Perform 9 TailFits
	DCOM_FIT17	Perform 17 TailFits
	DCOM_ALL	Perform TailFit on every edge
Default:	DCOM_NONE	

lErrProb Error probability level for Total Jitter. Total Jitter is calculated based on the desired Error Probability level. This value is used in conjunction with the bathtub curve after the successful completion of a tail-fit in order to project the value of Total Jitter.

Valid Entries: -1 to -16

Default: -12

lPassCnt This parameter is a bi-directional structure element that tracks the number of acquisitions since last reset. This flag can be read after an execution or set prior to an execution. Setting this parameter to 0 essentially resets this register. It will be automatically incremented when a measurement is performed.

Valid Entries: any integer greater than or equal to 0

Default: 0

lFftAvgs This variable is used to calculate the number of averages to use in the FFT. Increasing the number of averages reduces the background noise associated with the FFT algorithm. The number of averages is calculated based on the equation:

AVERAGES = 2^n where n = lFftAvgs

Valid Entries: any integer greater than or equal to 0

Default: 0 (indicating 2^0 averages = 1 execution.)

tRateInf A structure of type SPEC used by the Bit Rate measurement. The structure holds measurement specific parameters such as sample count, pattern repeats and maximum standard error. See Section 2-7 for a description of the SPEC structure and its elements.

tDdjtInf A structure of type SPEC used by the Data Dependant Jitter (DDJ) measurement. The structure holds measurement specific parameters such as sample count, pattern repeats and maximum standard error. See Section 2-7 for a description of the SPEC structure and its elements.

tRjpjInf A structure of type SPEC used by RJ & PJ estimate. The structure holds measurement specific parameters such as sample count, pattern repeats and maximum standard error. See Section 2-7 for a description of the SPEC structure and it's elements.

dDdjtLpf Low pass DCD+DDJ filter frequency in Hertz, negative value disables filter. This filter allows the user to apply a low pass filter function to the DCD+DDJ data to approximate the low pass filtering effects that would be present on the receiver or in the transmission line. The low pass filter is basically the bandwidth of the transmission line and the input bandwidth of the receiver. This is only valid when external arming is enabled.
Valid Entries: 0 to the Carrier Frequency (F_c) or –1 to disable.
Default: -1 (indicating the filter is off.)

dDdjtHpf High pass DCD+DDJ filter frequency in Hertz, a negative value disables filter. This filter allows the user to apply a high pass filter function to the DCD+DDJ data to approximate the high pass filtering effects that would be present on the receiver or in the transmission line. The High Pass filter is basically the PLL's response to the DCD+DDJ. Since the data will be clocked into the de-serializer by the PLL, the response of the PLL to the DCD+DDJ will become apparent as a function of the PLL to the de-serializer. This is only valid when external arming is enabled.
Valid Entries: 0 to the Carrier Frequency (F_c) or –1 to disable.
Default: -1 (indicating the filter is off.)

dRjpjFmn Minimum integration limit for RJ+PJ in Hertz, a negative value disables filter. This filter is used post-measurement as a means of focusing the RJ & PJ estimates on specific frequency bands with in the FFT. This filter is not normally used in a production program and should be left disabled.
Valid Entries: 0 to the Carrier Frequency (F_c) or –1 to disable.
Default: -1 (indicating the filter is off.)

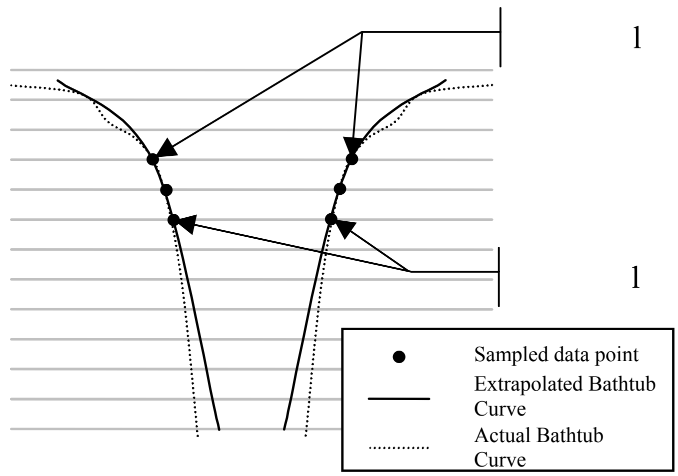
dRjpjFmx Maximum integration limit for RJ+PJ in Hertz, a negative value disables filter. This filter is used post-measurement as a means of focusing the RJ & PJ estimates on specific frequency bands with in the FFT. This filter is not normally used in a production program and should be left disabled.
Valid Entries: 0 to the Carrier Frequency (F_c) or –1 to disable.
Default: -1 (indicating the filter is off.)

dBitRate A bi-directional variable that allows the user to specify the bit rate or read back what the SIA3000 measured as the bit rate. If lGetRate is non-zero the bit rate is measured and placed in this field. If lGetRate is set to zero an the bit rate is read by the software from this field. This value must be supplied when Random mode is being used.
Valid Entries: 0 to the maximum bit rate of channel card
Default: 0 (indicating bit rate will be measured.)

dCornFrq Corner Frequency for RJ & PJ estimate in Hertz. This value is used in conjunction with the Bit Rate and pattern to determine the maximum stop count to be used to acquire RJ & PJ data. A lower value increase acquisition time.
Valid Entries: Bit-Rate /10,000,000 to Bit-Rate I
Default: 637e3 (637kHz – Fibre Channel 1X)

lHeadOff Header offset parameter, for use in packet-ized data which may have a frame header before the test pattern. This offset value can be used to skip past header information and into the repeating data pattern stream. This can be useful when analyzing data from disk drives when the pattern marker may be synchronized with the start of frame data.
Valid Entries: 0 to 10,000,000-pattern length I
Default: 0 (indicating no header present)

lFndEftv Flag to indicate that an effective jitter calculation is to be attempted. Effective Jitter is a means of estimating the effective deterministic jitter as it relates to a .5 error probability. This is done by first capturing the bathtub curve using conventional RJ & DJ estimation techniques; then, extrapolating from a few points in the bathtub curve to the .5 error probability level to estimate effective DJ. Effective RJ is extracted based on the curve that was fitted to the sample points. These values should only be used to correlate to a BERT Scan measurement and should not be used as a vehicle for quantifying jitter. This technique was developed to allow BERT systems to correlate with SIA3000 results.



Extrapolated Bathtub curve versus real bathtub curve as seen by BERT

Valid Entries: 0 – disable effective jitter estimate
1 – enable effective jitter estimate
Default: 0

lMinEftv, lMaxEftv Defines the error rates at which the eye width calculation will be used in the estimating effective jitter components. lMinEftv and lMaxEftv define points on the bathtub curve from which the extrapolated RJ curve is traced. Then, where this extrapolated curve intersects the .5 error probability, the effective DJ is calculated.
Valid Entries: -1 to –16 (indicating 10^{-1} to 10^{-16} error rate)
Default: -4 and –12 (indicating 10^{-4} BER for lMaxEftv and 10^{-12} BER for lMinEftv)

lFiltEnb Flag to enable IDLE character insertion filter. When enabled any edge measurements that are not within ± 0.5 UI will be discarded. This filter is used in systems, which may insert an idle character from time to time to compensate for buffer under-run/overrun issues. In those instances where an idle character was inserted during a measurement, the edge selection may be off. If this parameter is greater than or equal to one, the filter is enabled and measurements that differ from the mean by ± 0.5 UI will be discarded.
Valid Entries: 0 – disable idle character filter
1 – enable idle character filter
Default: 0

lQckTjit Flag to indicate a fast total jitter calculation will be performed using simple linear calculation of Total Jitter instead of convolving the DJ Probability Density Functions and the RJ Probability Density Functions. This calculation is based on the formula [TJ = DJ + n*RJ] where DJ and RJ are measured, and n is the multiplier based on a theoretical Gaussian distribution

```
Valid Entries: 0  do not use convolution for TJ est.
               2  Convolve DJ and RJ for TJ est.
Default:       0
```

lGood Flag indicates valid output data in structure. A positive value in this parameter indicates that the measurement was completed successfully, and, valid data can be extracted from this structure.

tPatn Structure of type PATN which holds all of the pattern information with regards to pattern length, pattern content, marker placement relative to location in pattern and other pattern specific metrics. (See Section 2-9 for a detailed description of the PATN structure elements.) This is an internal structure that the system uses to store pattern information and does not need to be altered by the user. The first time a measurement is performed the pattern is loaded into tPatn which is used internally for all subsequent acquisition and analysis.

dHits Total samples taken to calculate DDJ, RJ, and PJ values combined. Gives an indication of the actual data to support the calculated total jitter number.

dDdjt DCD+DDJ measurement in seconds. This measurement is taken from the mean deviation of each pattern edge from it's ideal location. All deviations are placed in a histogram and the peak-peak value from this histogram is placed in this structure location.

dRang Peak-to-peak of "All-Measurements" histogram. This histogram is part of the random data analysis package and should not be used as a metric of jitter measurement. Numbers captured in this tool are for comparison purposes only and only coincidentally share some terminology with jitter measurements.

dRjit[n] Random jitter estimate, in seconds, for each of the enabled acquire modes. Each mode's RJ estimate is kept separate since the data came from frequency information derived from different FFTs.

dPjit[n] Periodic jitter measurement, in seconds, for each of the enabled acquire modes. Each enabled acquire mode's PJ measurement is kept separate since the data came from frequency information derived from different FFTs.

dTjit[n] Total jitter estimate, in seconds, for each of the enabled acquire modes. Each mode's TJ estimate is kept separate since the data came from frequency information derived from different FFTs.

dEftvLtDj[n] Effective Deterministic(eDJ) jitter estimate, in seconds, for the left side of the bathtub curve. Total eDJ is calculated by adding dEftvLtDj to dEftvRtDj. Each of the enabled acquire modes is stored in the appropriate array location as specified in the table below. In order to calculate the effective jitter the flag lFndEftv must be enabled. Since the effective jitter is calculated by optimizing a curve-fit to the bathtub curve, a result is not guaranteed. If the curve-fit is unsuccessful, a negative value will be returned in this variable.

dEftvLtRj[n] Effective Random(eRJ) jitter estimate, in seconds, for the left side of the bathtub curve. Total eRJ is calculated by averaging dEftvLtRj and dEftvRtRj. Each of the enabled acquire modes is stored in the appropriate array location as specified in the table below. In order to calculate the effective jitter the flag lFndEftv must be enabled. Since the effective jitter is calculated by optimizing a curve-fit to the bathtub curve, a result is not guaranteed. If the curve-fit is unsuccessful, a negative value will be returned in these variables.

dEftvRtDj[n] Effective Deterministic(eDJ) jitter estimate, in seconds, for the right side of the bathtub curve. Total eDJ is calculated by adding dEftvLtDj to dEftvRtDj. Each of the enabled acquire modes is stored in the appropriate array location as specified in the table below. In order to calculate the effective jitter the flag lFndEftv must be enabled. Since the effective jitter is calculated by optimizing a curve-fit to the bathtub curve, a result is not guaranteed. If the curve-fit is unsuccessful, a negative value will be returned in this variable.

dEftvRtRj[n] Effective Random(eRJ) jitter estimate, in seconds, for the right side of the bathtub curve. Total eRJ is calculated by averaging dEftvLtRj and dEftvRtRj. Each of the enabled acquire modes is stored in the appropriate array location as specified in the table below. In order to calculate the effective jitter the flag lFndEftv must be enabled. Since the effective jitter is calculated by optimizing a curve-fit to the bathtub curve, a result is not guaranteed. If the curve-fit is unsuccessful, a negative value will be returned in this variable.

tRiseHist Structure of type PLTD which contains all of the plot information for generating a DCD+DDJ histogram of rising edges. See Section 2-3 for details concerning the PLTD structure and its elements.

tFallHist Structure of type PLTD which contains all of the plot information for generating a DCD+DDJ histogram of falling edges. See Section 2-3 for details concerning the PLTD structure and its elements.

tRiseMeas Structure of type PLTD (See Section 2-3) which contains all of the plot information for generating an all-measurements histogram of rising edges. This plot is only valid when using random mode. This histogram is for informational use and qualitative assessment. Numbers originating from this measurement methodology are not to be confused with jitter measurements.

tFallMeas Structure of type PLTD which contains all of the plot information for generating an all-measurements histogram of falling edges. This plot is only valid when using random mode. This histogram is for informational use and qualitative assessment. Numbers originating from this measurement methodology are not to be confused with jitter measurements. See Section 2-3 for details concerning the PLTD structure and its elements.

tNormDdjt Structure of type PLTD which contains all of the plot information for generating a DCD+DDJ versus UI plot. This plot is only valid in Pattern Marker mode. See Section 2-3 for details concerning the PLTD structure and its elements.

tHipfDdjt Structure of type PLTD which contains all of the plot information for generating an DCD+DDJ versus UI plot with the DCD+DDJ High Pass Filter enabled. This plot is only valid in Pattern Marker Mode and dDdjtHpf is a non-negative number. *(For a discussion on the High Pass Filter Function for DCD+DDJ data, see dDdjtHpf above.)* When dDdjtHpf is enabled, the dDdjt value is calculated based on applying the dDdjtHpf filter. See Section 2-3 for details concerning the PLTD structure and its elements.

tLopfDdjt Structure of type PLTD \which contains all of the plot information for generating an DCD+DDJ versus UI plot with the DCD+DDJ Low Pass Filter enabled. This plot is only valid in Pattern Marker Mode and dDdjtLpf is a non-negative number. *(For a discussion on the Low Pass Filter Function for DCD+DDJ data, see dDdjtLpf above.)* See Section 2-3 for details concerning the PLTD structure and its elements.

tBathPlot[n] Structure of type PLTD which contains all of the plot information for generating a Bathtub curve. There is one structure and associated plot for each of the acquisition modes specified in lAcqMode. See Section 2-3 for details concerning the PLTD structure and its elements.

tEftvPlot[n] Structure of type PLTD which contains all of the plot information for generating an Bathtub curve based on Effective Jitter if lFndEftv is set and a valid fit is obtained. *(For a detailed description of Effective Jitter, see lFndEftv above.)* There is one structure and associated plot for each of the acquisition modes specified in lAcqMode. See Section 2-3 for details concerning the PLTD structure and its elements.

tSigmNorm[n] Structure of type PLTD which contains all of the plot information for generating an 1-Sigma versus UI plot. *(x-axis can be converted to time from UI based on dBitRate value.)* This plot describes the standard deviation for each accumulated time sample. There is one structure and associated plot for each of the acquisition modes specified in lAcqMode. See Section 2-3 for details concerning the PLTD structure and its elements.

tSigmTail[n] Structure of type PLTD which contains all of the plot information for generating a 1σ TailFit results versus UI plot. *(x-axis can be converted to time from UI based on dBitRate value.)* Each successful TailFit will be displayed as a data point and connected to adjacent TailFit samples. The plot value represents the overall RJ for the given amount of accumulated UI. This plot is only valid if tail-fit is enabled. . There is one structure and associated plot for each of the acquisition modes specified in lAcqMode. See Section 2-3 for details concerning the PLTD structure and its elements.

tFreqNorm[n] Structure of type PLTD which contains all of the plot information for generating a Jitter versus Frequency plot. There is one structure and associated plot for each of the acquisition modes specified in lAcqMode. See Section 2-3 for details concerning the PLTD structure and its elements.

tFreqTail[n] Structure of type PLTD which contains all of the plot information for generating a 1σ TailFit results versus frequency plot. This plot is only valid if tail-fit is enabled. There is one structure and associated plot for each of the acquisition modes specified in lAcqMode. See Section 2-3 for details concerning the PLTD structure and its elements.

The following parameters are for internal use only. They are presented for reference only. Do not try to read the values or parse the structures nor try to write the various locations.

dWndFact, lMaxStop, lCmpMode, lPosRoll, lNegRoll, lAdjustPW These values are for internal use only, DO NOT ALTER or try to use.

tDdjtData Structure which contains the raw DCD+DDJ measurements. This value is for internal use only, DO NOT ALTER or try to use.

lDdjtRsvd Used to track memory allocation for tDdjtData structures. This value is for internal use only, DO NOT ALTER or try to use.

dMeasData Raw all-measurements histogram data, only valid when auto-arming is used. This structure is for internal use only, DO NOT ALTER or try to use.

lMeasRsvd Used to track memory allocation for dMeasData values. This value is for internal use only, DO NOT ALTER or try to use.

dRjpjData Raw variance data used for the calculation of RJ and PJ. This structure is for internal use only, DO NOT ALTER or try to use.

lRjpjRsvd Used to track memory allocation for dRjpjData values. This value is for internal use only, DO NOT ALTER or try to use.

dTfitData Raw tail-fit data if tail-fit data is enabled and successful, as indicated by the lGood variable in the tTfit structure being non-zero. This structure is for internal use only, DO NOT ALTER or try to use.

lTfitRsvd Used to track memory allocation for dTfitData values. This value is for internal use only, DO NOT ALTER or try to use.

lPeakData Tracks detected spikes in RJ+PJ data. This value is for internal use only, DO NOT ALTER or try to use.

lPeakNumb Count of detected spikes, indicates the number of values in the lPeakData array.

lPeakRsvd Used to track memory allocation for lPeakData values. This value is for internal use only, DO NOT ALTER or try to use.

dFreqData Raw FFT output when averaging is enabled. This structure is not normally directly access by an application program. This value is for internal use only, DO NOT ALTER or try to use.

lFreqRsvd Used to track memory allocation for dFreqData values. This value is for internal use only, DO NOT ALTER or try to use.

dTailData Raw tail-fit FFT output when tail-fit and averaging are both enabled. This structure is not normally directly access by an application program. This value is for internal use only, DO NOT ALTER or try to use.

lTailRsvd Used to track memory allocation for dTailData values. This value is for internal use only, DO NOT ALTER or try to use.

void __stdcall FCNL_DefDcom (DCOM *dcom)

This function is used to fill the dcom structure for the Datacom Known Pattern with Marker tool with reasonable default values. It is recommended that this function be called initially even if parameters within the structure are to be adjusted manually, and may be called repeatedly to reestablish initial conditions; however, this will impact test time.

Before calling this function, zero out the DCOM structure using the standard memset() function to ensure that any information pertaining to dynamic memory allocation is cleaned out prior to using the structure.

INPUTS

dcom - Pointer to a DCOM structure. Memory needs to be allocated by the caller.

OUTPUTS

None.

void __stdcall FCNL_ClrDcom (DCOM *dcom)

This function frees any dynamic memory that may have been allocated during previous data acquisitions and clears out the dcom structure.

INPUTS

dcom - Pointer to a DCOM structure. Memory needs to be allocated by the caller.

OUTPUTS

None.

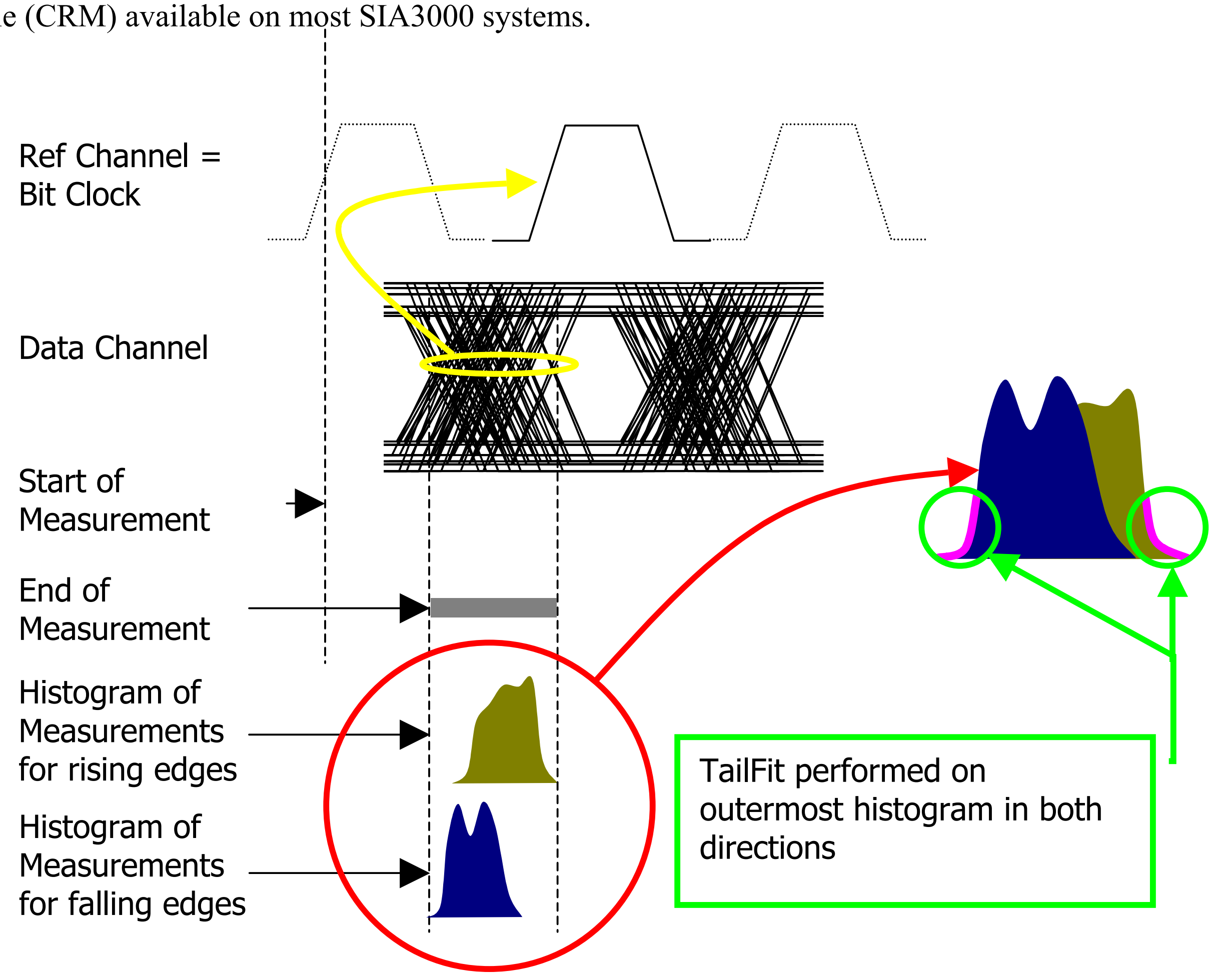
EXAMPLE

```
#define TRUE 1
static DCOM dataJit;                                    //declare dataJit to be a structure of
                                                //type DCOM
memset ( &dataJit, 0, sizeof ( DCOM ) );        //clear the memory for dataJit structure
FCNL_DefDcom ( &dataJit);                       //set dataJit structure to default values
                                                //NOTE:  dataJit.tparm, dataJit.tRateInf,
                                                //dataJit.DdjtInf, dataJit.tRjpjInf,
                                                //dataJit.tPatn and dataJit.tDdjtData
                                                //are also set to defaults by this
                                                //command.
dataJit.tParm.lChanNum = 1;                             //Set channel number to 1
dataJit.tparm.lExtnArm = 2;                             //Set Pattern Marker to Channel 2
dataJit.tParm.lSamCnt = 500;                            //Capture 500 measurements per pass.
dataJit.tParm.lAutoArm = ARM_EXTRN;                     //Set to External Arming mode
strcpy(&dataJit.sPtnName[0], "cjtpat.ptn");             //Use k28.5 pattern
dataJit.lTailFit = DCOM_AUTO;                           //Perform TailFit for RJ estimate.  Let
                                                //SIA3000 decide how many TailFit
                                                //samples to take.
dataJit.tRateInf.SampCnt = 10000;               //Set sample count for BitRate meas. To
dataJit.tRateInf.PtnReps = 100;                 //10,000 and Pattern Repeats to 100 for
                                                //improved DDJ measurement accuracy.
dataJit.dCornFrq = 637000;                      //Set Corner Frequency to 637kHz
dataJit.lQckTjit = TRUE;                        //Use simple calc for TJ for faster result.

FCNL_RqstPkt ( ApiDevId, & dataJit, WIND_DCOM );        //execute the measurement.
FCNL_RqstAll ( ApiDevId, & dataJit, WIND_DCOM );        //get plot data

//Print Total Jitter Estimate.
If (dataJit.lGood>0) printf("\nTJ = %d\n",dataJit.dTjit[0]);

FCNL_ClrDcom ( &dataJit);                       //deallocate the structure
```

2-22 DATACOM RANDOM DATA WITH BIT CLOCK TOOL

The Datacom Random Data With Bit Clock Tool is used to measure jitter from a reference clock to a data signal. This measurement setup is the same as the setup used by an oscilloscope when generating an Eye Diagram or for Eye Mask testing. The measurement starts out with a quick frequency measurement for the reference clock. Based on this information, the algorithm finds the next clock transition and establishes data filters that limit the data to only those transitions that are within a ± 0.5 UI window of the expected clock. This means that the software will throw out any measurements that are not valid and belong to a different location in the pattern. Then, the instrument measures from the bit clock to the data channel and generates two histograms of measurements, one for each polarity of the data signal. Then, the histograms are overlaid and the right most and left most edges are used to perform a TailFit for RJ/DJ separation.

Eye Histogram Tool is used primarily for long data patterns (greater than 2k in length) or for fully random data streams in which no repeating pattern is available. The bit clock for this measurement could be placed on any one of the other input channels or may come from the optional Clock Recovery Module (CRM) available on most SIA3000 systems.



Measurement methodology for Eye Histogram Measurements.

```
typedef struct
  {
  /* Input parameters  */
  PARM    tParm;              /* Contains acquisition parameters        */
  long    lPassCnt;           /* Acquisitions so far, set to 0 to reset */
  long    lRefEdge;           /* Referenced to: EDGE_FALL or EDGE_RISE  */
  long    lErrProb;           /* Error probability used Total Jitter    */
                              /* Valid range is ( -1 to -16 )           */
  long    lClokSmp;           /* Sample size while acquiring clock rate */
  long    lFiltSmp;           /* Sample size when finding filter limits */
  long    lTailFit;           /* If non-zero a tail-fit will be tried   */
  long    lForcFit;           /* If non-zero use the force-fit method   */
  long    lMinHits;           /* Minimum hits before trying tail-fit    */
  long    lFndEftv;           /* Flag to attempt effective jitter calc  */
  long    lMinEftv;           /* Min probability for effective fit: -4  */
  long    lMaxEftv;           /* Max probability for effective fit: -12 */
  long    lDdrClok;           /* Non-zero for double data rate clocks   */
  double  dMinSpan;           /* Minimum span between edges in seconds  */
  long    lFiltOff;           /* Filter offset in %UI (100 to -100)     */
  long    lKeepOut;           /* If non-zero use tailfit keep out below */
  double  dKpOutLt;           /* Keep out value for left side           */
  double  dKpOutRt;           /* Keep out value for right side          */
  /* Output parameters */
  long    lGood;              /* Flag indicates valid data in structure */
  long    lRiseCnt;           /* Number of hits in rising edge data     */
  long    lFallCnt;           /* Number of hits in falling edge data    */
  long    lPad2;
  double  dDataMin;           /* Minimum value relative to clock edge   */
  double  dDataMax;           /* Maximum value relative to clock edge   */
  double  dDataSig;           /* 1-Sigma of all values relative to clock*/
  double  dAvgSkew;           /* Average of all values relative to clock*/
  double  dUnitInt;           /* Measured Unit Interval                 */

  long    lUnitOff;           /*****************************************/
  long    lSpanCnt;           /*                                        */
  double  dRiseMin;           /*  These values are all used internally  */
  double  dRiseMax;           /*   as part of the measurement process   */
  double  dFallMin;           /*                                        */
  double  dFallMax;           /*                                        */
  long    lRiseBin;           /*             DO NOT ALTER!              */
  long    lFallBin;           /*                                        */
  double  dLtSigma[PREVSIGMA];/*                                        */
  double  dRtSigma[PREVSIGMA];/*                                        */
  double  dAltMean;           /*****************************************/

  PLTD    tRise;              /* Histogram of rising edge data          */
  PLTD    tFall;              /* Histogram of falling edge data         */
  PLTD    tBoth;              /* Histogram of combined edge data        */
  PLTD    tRiseProb;          /* Probability Histogram of rising edges  */
  PLTD    tFallProb;          /* Probability Histogram of falling edges */
  PLTD    tBothProb;          /* Probability Histogram of combined edges*/
  PLTD    tBath;              /* Bathtub curves determined from PDF     */
  PLTD    tEftv;              /* Effective Bathtub curves if enabled    */
  TFIT    tTfit;              /* Structure containing tail-fit info     */
  } EYEH;
```

tParm A structure of type PARM that contains acquisition parameter. The PARM is discussed in full detail in Section 2-4. Be sure to either set the following parameters in **tParm** for a successful EyeHistogram Tool execution or review the default settings:

lChanNum This is a 32 bit word that represents the channel for this measurement. The upper 16 bits define which channel will be used as the reference edge (or bit clock) the lower 16 bits are used for identifying the channel to be measured. It is best to manipulate the channel selection field using HEX format or by using binary shift functions. ***See sample code at the end of this section for an example of using binary shift function in the channel declaration.*** in HEX format, simply enter the reference channel number in the first two bytes and the measured channel in the last two bytes such that 0x000m000n would indicate a reference channel of m and a measured channel of n (in hexadecimal format) where m and n are elements of the set {1,2,3,4,5,6,7,8,9,a}. For example, 0x00050003 would indicate that channel 5 was the channel with the bit clock signal and channel 3 was the channel with the data signal. The default for tParm.lChanNum within a EYEH structure is 0x00010002 indicating that the reference channel is defaulted to channel 1 and the measured channel is set to 2.

dStrtVlt Since measurements are made from the data signal to the next clock signal, the start of measurement is the data signal and thus dStrtVlt controls the threshold level for the data channel. It is typically best to leave this variable at the default and allow Pulse Find to establish the 50% level at which to test the device. However, there are two cases in which this may not be desirable. First, in a production environment, it may be too time-consuming to perform a Pulse Find each time the test is to be executed. All of the parts should have roughly the same voltage characteristics (if they are passing parts) and will most likely have the same threshold settings. Second, in some cases, it might be desirable to account for any slew rate issues by adjusting the threshold voltage to the cross point. A simple script can be written to identify the cross point prior to testing.

dStopVlt Since measurements are made from the data signal to the next clock signal, the stop of measurement is the reference clock signal and thus dStopVlt controls the threshold level for the clock channel. It is typically best to leave this variable at the default and allow Pulse Find to establish the 50% level at which to test the device. In a production environment, this value can be forced by turning pulse find off and setting this parameter.

lPassCnt This parameter is a bi-directional structure element that tracks the number of acquisitions since last reset. This flag can be read after an execution or set prior to an execution. Setting this parameter to 0 essentially resets this register. It will be automatically incremented when a measurement is performed.
Valid Entries: any integer greater than or equal to 0
Default: 0

lRefEdge Parameter to define the polarity of the clock edge which will be used as the reference.
Valid Entries: EDGE_FALL reference clock to data measurements tothe falling edge of the clock signal.
EDGE_RISE reference clock to data measurements to the rising edge of the clock signal.
Default: EDGE_RISE

lErrProb Exponent of Bit Error Probability (BER) to which Total Jitter will be calculated if TailFit is enabled. TJ is calculated based on the convolution of DJ and RJ out to 10^n BER where n = lErrProb.,
Valid Entries: Any integer from -1 to -16
Default: -12

lClokSmp Sample size while acquiring clock rate.
Valid Entries: Any integer less than or equal to 1,000,000
Default: 10000.

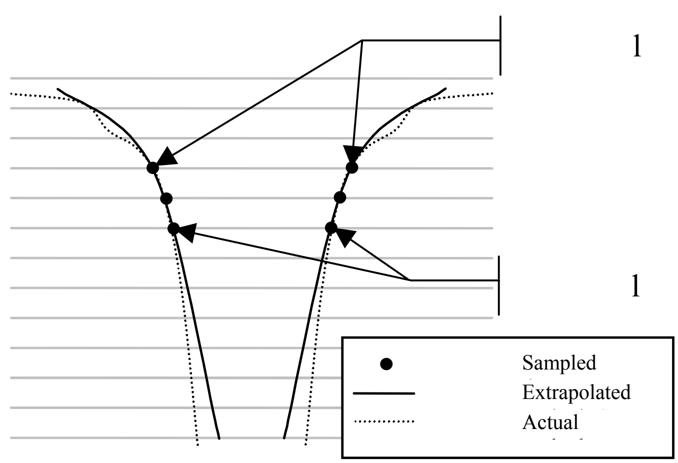
lFltSmp Sample size when finding filter limits
Valid Entries: Any integer less than or equal to 1,000,000
Default: 1000.

lTailFit Flag to indicate whether to perform a TailFit on data in the rising and falling data histograms. If non-zero, a tail-fit will be attempted. The lGood element of the tTfit structure will indicate if the TailFit was successful. Setting this structure element to 1 will initiate the TailFit algorithm.
Valid Entries: 0 – disable TailFit algorithm
1 – enable TailFit algorithm
Default: 0

lForcFit Flag to indicate whether to force a TailFit on a fixed sample size or to continue acquiring data until a sufficient amount of data has been collected resulting in a high level of confidence in the accuracy of the TailFit on the given sample. If selected, the TailFit algorithm will make a single attempt at fitting Gaussian tails to the tail regions of the histograms after acquiring the minimum number of samples as defined by lMinHits.
Valid Entries: 0 continue acquiring data until chi squared (X^2) estimate indicates a good TailFit was accomplished.
1 perform tail fit on only lMinHits amount of data.
Default: 0

lMinHits Minimum number of samples (in thousands) to acquire prior to attempting a TailFit.
Valid Entries: any positive integer less than or equal to 100,000
Default: 50

lFndEftv Flag to indicate that an effective jitter calculation is to be attempted. Effective Jitter is a means of estimating the effective deterministic jitter as it relates to a .5 error probability. This is done by first capturing the bathtub curve using conventional RJ & DJ estimation techniques; then, extrapolating from a few points in the bathtub curve to the .5 error probability level to estimate effective DJ. Effective RJ is extracted based on the curve that was fitted to the sample points. These values should only



Extrapolated Bathtub curve versus real bathtub curve as seen by BERT

be used to correlate to a BERT Scan measurement and should not be used as a vehicle for quantifying jitter. This technique was developed to allow BERT systems to correlate with SIA3000 results.
Valid Entries: 0 – disable effective jitter estimate
1 – enable effective jitter estimate
Default: 0

lMinEftv, lMaxEftv Defines the error rates at which the eye width calculation will be used in the estimating effective jitter components. lMinEftv and lMaxEftv define points on the bathtub curve from which the extrapolated RJ curve is traced. Then, where this extrapolated curve intersects the .5 error probability, the effective DJ is calculated.
Valid Entries: -1 to –16 (indicating 10^{-1} to 10^{-16} error rate)
Default: -4 and –12 (indicating 10^{-4} BER for lMaxEftv and 10^{-12} BER for lMinEftv)

dMinSpan Minimum delay between reference clock and measured edges. This parameter will skip a sufficient number of edges to measure the data transitions that are at least dMinSpan (in seconds) away from the reference clock. This parameter is used to correlate with oscilloscopes, which have a trigger delay of at least 20ns (typ.). It is not typically used in a production environment.
Valid Entries: 0 to 1.0
Default: 0

lFiltOff This allows an offset to be made to the filter that is used to isolate histogram data to within 1 UI of the bit clock. The filter is established on the first pass by the instrument, and can normally be left alone. However, in the presence of large amounts of jitter it may be necessary to tweak this value slightly. The offset is entered as a percentage of UI, and a value in the range of +/-100 is valid.
Valid Entries: -100 to +100
Default: 0

lGood Flag indicates valid output data in structure.

lRiseCnt Number of hits in rising edge data.

lFallCnt Number of hits in falling edge data.

dDataMin Minimum value relative to clock edge.

dDataMax Maximum value relative to clock edge.

dDataSig 1-Sigma of all values relative to clock.

dAvgSkew Average of all values relative to clock.

dUnitInt Measured Unit Interval, this is based on the clock.

tRise Structure of type PLTD which contains all of the plot information to generate a Histogram of rising-edge data to next reference clock measurements. See Section 2-3 for details of the PLTD structure and its elements.

tFall Structure of type PLTD which contains all of the plot information to generate a Histogram of falling-edge data to next reference clock measurements. See Section 2-3 for details of the PLTD structure and its elements.

tRiseProb Structure of type PLTD which contains all of the plot information to generate a probability histogram of rising-edge data to next reference clock measurements. The amplitude of each point in the probability histogram is normalized to the probability of a given measurement occurring as opposed to the total number of measurements made with the given result. See Section 2-3 for details of the PLTD structure and its elements.

tFallProb Structure of type PLTD which contains all of the plot information to generate a probability histogram of falling-edge data to next reference clock measurements. The amplitude of each point in the probability histogram is normalized to the probability of a given measurement occurring as opposed to the total number of measurements made with the given result. See Section 2-3 for details of the PLTD structure and its elements.

tBath Structure of type PLTD which contains all of the plot information to generate a bathtub curve based on Probability Density Function derived from histogram data and RJ estimate from TailFit algorithm. . See Section 2-3 for details of the PLTD structure and its elements.

tEftv Structure of type PLTD which contains all of the plot information to generate a bathtub curve based on the estimate of effective Deterministic Jitter (eDJ) and effective Random Jitter (eRJ) derived from the true data bathtub curve. This plot is only available when lFndEftv is set and a valid fit is obtained. See Section 2-3 for details of the PLTD structure and its elements.

tTfit A structure of type TFIT containing tail-fit info. See Section 2-5 for details of the TFIT structure and its elements.

lUnitOff, dRiseMin, dRiseMax, dFallMin, dFallMax, lRiseBin, lFallBin, dLtSigma, dRtSigma, lSpanCnt
These values are all used internally, DO NOT ALTER!

void __stdcall FCNL_DefEyeh (EYEH *eyeh)

This function is used to fill the eyeh structure for the Datacom with Bit Clock tool with reasonable default values. It is recommended that this function be called initially even if parameters within the structure are to be adjusted manually, and may be called repeatedly to reestablish initial conditions; however, this will impact test time.

Before calling this function, zero out the EYEH structure using the standard memset() function to ensure that any information pertaining to dynamic memory allocation is cleaned out prior to using the structure.

INPUTS

eyeh - Pointer to a EYEH structure. Memory needs to be allocated by the caller.

OUTPUTS

None.

void __stdcall FCNL_ClrEyeh (EYEH *eyeh)

This function frees any dynamic memory that may have been allocated during previous data acquisitions and clears out the eyeh structure.

INPUTS

eyeh - Pointer to a EYEH structure. Memory needs to be allocated by the caller.

OUTPUTS

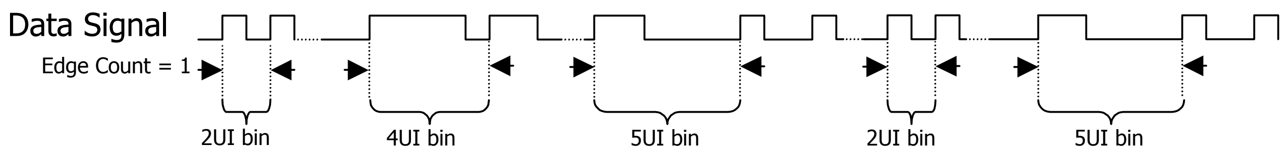
None.

EXAMPLE

```
#define TRUE 1
#define FALSE 0

static EYEH eyehist;                                    //declare eyehist to be a structure of
                                                //type EYEH
memset ( &eyehist, 0, sizeof ( EYEH ) );        //clear the memory for eyehist structure
FCNL_DefEyeh ( &eyehist);                       //set eyehist structure to default values
                                                //NOTE:  eyehist.tparm, are also set to
                                                //defaults by this command.
eyehist.tParm.lChanNum = 1 | (2<<16);                   //Set ch 1 for data and ch 2 for ref clk
eyehist.tParm.lSampCnt = 50,000;                //Set sample size to 50k
eyehist.lTailFit = TRUE;                        //Enable TailFit for RJ estimate
eyehist.ForcFit = TRUE;                         //Force the fit with first 50k samples
eyehist.MinHits = 50,000;                       //set minimum samples to 50k

FCNL_RqstPkt ( ApiDevId, & eyehist, WIND_EYEH );        //execute the measurement.

//Print Total Jitter Estimate.
If (eyehist.lGood>0) printf("\nTJ = %d\n",eyehist.tTfit.dTjit);

FCNL_ClrEyeh ( &eyehist);                       //deallocate the structure
```

2-23 DATACOM RANDOM DATA WITH NO MARKER TOOL

The Datacom Random Data With No Marker Tool is used to estimate jitter components on random data signals without the benefit of a repeating data pattern or access to a bit clock. This tool is used primarily to capture relative jitter amplitudes and is not considered an accepted means of accurately measuring jitter components on a data signal. For accurate jitter measurements on data signals, it is imperative to have a repeating pattern and a pattern trigger or have access to a bit clock. This tool, the Random Data Tool, is prone to inaccuracies when periodic jitter is present and data dependent jitter is present on the signal. This tool does not take into account any PJ amplitude when estimating Total Jitter. Secondly, this tool may underestimate the amplitude of DDJ due to data binning errors.



Example of Random Data utility when edge count equals 1. In a complete execution of the random data utility, edge count will range from 1 to FC/(4*FM) where FC is the carrier frequency and FM is the modulation cutoff frequency.

To capture jitter information, this tool measures time from randomly selected transitions in the pattern to a subsequent edge in the pattern some “n” number of transitions after the start of the measurement. “n” is swept from a count of 1 to a count as defined by the carrier frequency and the desired cutoff frequency. Once all of the measurements are captured, the data is binned according to their proximity to integer multiples of the bit period. (For example, all measurements within ± .5UI of 5xbit-period are placed in the 5UI bin.) Then, each bin is parsed for statistical information including jitter and mean offset from ideal. The mean offset is used to estimate Data Dependent Jitter (DDJ). As such, the location of the mean for a given bin’s histogram could be artificially inflated based on combining measurements from transitions which are not from the same point in the data pattern. The above example shows a given burst of measurements where the edge count was equal to 1. During the course of the complete measurement, the edge count will be varied from an initial value of 1 to a final value determined based on the bit rate and the intended cutoff frequency. Each is bin is also sorted based on edge count and polarity in an attempt to maximize accuracy of DDJ estimate. Once all of the data is captured, the mean of each histogram for each sub-bin is compared to an ideal bit clock and the deviation is taken as Data Dependant Jitter. All DDJ estimates are combined to determine the peak to peak spread of DDJ. Then, the algorith selects appropriate edge counts to create a histogram from which to capture TailFit information in an attempt to estimate RJ. Based on the users selection of the structure element tDcom.lTailFit.

The structure used in this tool incorporates a Datacom Known Pattern With Marker structure. In other words, this tool basically creates a “wrapper” structure around the dataCOM structure which has settings unique to the random data tool.

To estimate Random Jitter (RJ) on a random signal without the benefit of a reference clock, the random data tool uses TailFit on sampled data histograms from various amounts of accumulated bit periods. The precision of the measurement is increased as the number of different accumulations used is increased. There is a significant increase in test time for increasing the number of tailfit points. As such, the user can specify 4 different setting selections or have the instrument dynamically decide which to use (AUTO). In AUTO mode, the tool first performs 3 tailfits (maximum count, minimum count and middle count) and checked to see if the deviation between adjacent RJ measurements is less than the percentage specified in lPcnt. If the deviation is greater, the instrument will perform two more TailFit measurements between the three already taken. Again, the instrument will check adjacent RJ estimates and decide whether to capture additional interstitial samples.

```
typedef struct
  {
  /* Input parameters  */
  long    lCoun;              /* Count of tailfits, see constants above */
  long    lPcnt;              /* Automode suceed %, see constants above */
  DCOM    tDcom;              /* DCOM structure holds most information  */
  /* Output parameters */
  long    lGood;              /* Flag indicates valid data in structure */
  long    lPad1;
  double  dDjit;              /* Deterministic jitter value             */
  double  dRjit;              /* Random jitter value                    */
  double  dTjit;              /* Total jitter value                     */
  PLTD    tSigmTail;          /* 1-Sigma plot using tail-fits           */
  } RAND;
```

lCoun This parameter selects the number TailFit iterations to be captured. This number can be any of 3, 5, 9 or 17. In RAND_AUTO mode, the user can choose to have the instrument dynamically decide the number based on the deviation of adjacent RJ estimates. The instrument will start with 3 TailFits and increase the count based on the value specified in lPcnt.

Valid Entries: RAND_AUTO - Continue to perform tailfits until RJ is within some percentage of the previous pass.

RAND_AUTO -	Continue to perform tailfits until RJ is within some percentage of the previous pass.
RAND_FIT3 -	Perform 3 tailfits
RAND_FIT5 -	Perform 5 tailfits
RAND_FIT9 -	Perform 9 tailfits
RAND_FIT17 –	Perform 17 tailfits

lPcnt Target maximum amount of deviation between adjacent RJ estimates. Each RJ estimate is calculated based on a histogram of accumulated bit periods. Then, each RJ is compared with the RJ estimate of the adjacent accumulations. The percentage difference is compared with this entry to determine if the RJ estimate is valid.

RAND_PCNT5	RJ within 5% of adjacent estimates
RAND_PCNT10	RJ within 10% of adjacent estimates
RAND_PCNT25	RJ within 25% of adjacent estimates
RAND_PCNT50	RJ within 50% of adjacent estimates

tDcom Structure of type DCOM which specifies most of the input and output parameters necessary for a data signal analysis. *See D-3 for more details on the DCOM structure and the elements described below.* The user will need to review all of the default parameters of the DCOM structure and decide which to change. The following entities from the DCOM structure are valid for use with the random data tool:

tDcom.tParm Acquisition parameter sub structure.

tDcom.AcqMode Acquire Mode (rise-rise, rise-fall, fall-rise, fall-fall)

tDcom.lRndMode Enable/Disable Random Mode

tDcom.lErrProb Error Probably level to which TJ is to be calculated.

tDcom.lPassCnt Number of passes using same RAND structure since
tDcom.lFftAvgs Number of FFTs to capture and average
tDcom.tDdjtInf SPEC structure used to set up DDJ measurement.
tDcom.dBitRate Bit Rate of data signal under test.
tDcom.dCornFrq Corner Frequency as specified by given standard
tDcom.lFndEftv Enable/Disable Effective Jitter measurements
tDcom.lMinEftv Minimum BER point in Bathtub curve used for Effective Jitter.
tDcom.lMaxEftv Maximum BER point in Bathtub curve used for Effective Jitter.
tDcom.lQckTjit Enable Quick TJ estimate rather than convolving RJ+DDJ for TJ.
tDcom.lGood Flag to indicate valid data results exist in structure.
tDcom.dHits total number of measurements made
tDcom.dDdJt peak-peak amplitude of DDJ
tDcom.dRang peak-peal of all measurements histogram.
tDcom.dRjit[n] RJ estimate for each possible mode.
tDcom.dPjit[n] PJ estimate for each possible mode.
tDcom.dTjit[n] TJ estimate for each possible mode.
tDcom.dEftvLtDj[n] Effective DJ estimate for left or short cycle side.
tDcom.dEftvLtRj[n] Effective RJ estimate for left or short cycle side.
tDcom.dEftvRtDj[n] Effective DJ estimate for right or long cycle side.
tDcom.dEftvRtRJ[n] Effective RJ estimate for right or long cycle side.
tDcom.tRiseHist PLTD structure of DDJ histogram for rising edges
tDcom.tFallHist PLTD structure of DDJ histogram for falling edges
tDcom.tRiseMeas PLTD structure of "All Measurements" of rising edges.
tDcom.tFallMeas PLTD structure of "All Measurements" of falling edges.
tDcom.tBathPlot[n] PLTD structure of bathtub curves for each measurement mode.
tDcom.tEftvPlot[n] PLTD structure of Effective Jitter for each measurement mode.
tDcom.tSigmNorm[n] PLTD structure of standard Deviation (1σ) versus time.
tDcom.tSigmTail[n] PLTD structure of 1σ versus time using TailFit for RJ.
tDcom.tFreqNorm[n] PLTD structure of 1σ versus frequency.
tDcom.tFreqTail[n] PLTD structure of 1σ versus frequency using TailFit for RJ.

lGood Flag indicates valid output data in structure.

dDjit Deterministic Jitter estimate. This value is based strictly on the Data Dependant Jitter calculation and does not account for any Periodic Jitter since it is impossible to accurately separate Periodic Jitter in the FFT results when DDJ is present.

dRjit Random Jitter estimate. This value comes from the series of TailFits that were performed on the accumulated jitter data.

dTjit Total Jitter estimate. This value is the convolution of the DDJ probability density function captured in dDjit and the RJ estimate captured in dRjit.

tSigmTail Structure of type PLTD containing information necessary to create a plot of RJ (based on the TailFit results) and 1-σ (standard deviation) as a function of accumulated bit periods. See Section 2-3 for details of the PLTD structure and its elements.

void __stdcall FCNL_DefRand (RAND *rand)

This function is used to fill the rand structure for the Datacom Random Data With No Marker tool with reasonable default values. It is recommended that this function be called initially even if parameters within the structure are to be adjusted manually, and may be called repeatedly to reestablish initial conditions; however, this will impact test time.

Before calling this function, zero out the RAND structure using the standard memset() function to ensure that any information pertaining to dynamic memory allocation is cleaned out prior to using the structure.

INPUTS

rand - Pointer to a RAND structure. Memory needs to be allocated by the caller.

OUTPUTS

None.

void __stdcall FCNL_ClrRand (RAND *rand)

This function frees any dynamic memory that may have been allocated during previous data acquisitions and clears out the rand structure.

INPUTS

rand - Pointer to a RAND structure. Memory needs to be allocated by the caller.

OUTPUTS

None.

EXAMPLE

```
#define TRUE 1
#define FALSE 0
static RAND rdataJit;                                   //Declare rdataJit to be a structure of
                                                //type RAND
memset ( &rdataJit, 0, sizeof ( RAND ) );       //Clear the memory for rdataJit structure
FCNL_DefRand ( &rdataJit);                      //Set rdataJit structure to default values
                                                //NOTE:  rdataJit.tdcom and all of the
                                                //DCOM substructures (including tparm)
                                                //are also set to defaults by this
                                                //command.
rdataJit.tDcom.tParm.lChanNum = 1;                      //Set channel number to 1
rdataJit.tDcom.tParm.lSamCnt = 500;                     //Capture 500 measurements per pass.
rdataJit.tDcom.dCornFrq = 637000;               //Set Corner Frequency to 637kHz
rdataJit.lCoun = RAND_AUTO;                             //Set TailFit count to aotomatic mode.
rdataJit.lPcnt = RAND_PCNT10;                           //Set target deviation maximum to 10%

FCNL_RqstPkt ( ApiDevId, & rdataJit, WIND_RAND );       //execute the measurement.
FCNL_RqstAll ( ApiDevId, & rdataJit, WIND_RAND );       //get plot data

//Print Total Jitter Estimate.
If (rdataJit.lGood>0) printf("\nTJ = %d\n",rdataJit.dTjit);

FCNL_ClrRand ( &rdataJit);                      //deallocate the structure
```

2-24 FIBRE CHANNEL COMPLIANCE TOOL

The Fibre Channel Compliance Tool utilizes the Datacom Known Pattern with Marker Tool for the measurements. In addition to the data signal to be analyzed, this tool requires a pattern marker to be connected to the Arm Channel. If your SIA-3000 is equipped with the PM-50 option, the marker signal will be generated on the card and no additional input signals are required for making a measurement. The Marker signal has an edge relative to the same bit of the pattern each time the marker occurs. Since no bit-clock is used, analysis of jitter is independent of clock-jitter effects, and because the Arm is not a trigger, any jitter on the marker will not transfer to the measurement of the Data.

For an in depth description on Known Pattern With Marker measurement theory, refer to the Known Pattern With Marker quick reference guide.

```
typedef struct
  {
  /* Input parameters  */
  double  dAttn;              /* Attenuation factor (dB)               */
  DCOM    tDcom;              /* DCOM structure holds most information  */
  /* Output parameters */
  long    lGood;              /* Flag indicates valid data in structure */
  long    lPad0;
  PLTD    tNrmScop;           /* Normal channel voltage data           */
  PLTD    tCmpScop;           /* Complimentary channel voltage data     */
  } FCMP;
```

dAttn Attenuation factor in dB, this is provided to allow the results to be scaled to compensate for external attenuation from sources such as probes.
Default: 0

tDcom Structure of type DCOM which specifies most of the input and output parameters necessary for a data signal analysis. The user will need to review all of the default parameters of the DCOM structure and decide which to change.

lGood Flag indicates valid data in structure

tNrmScop Normal channel voltage data

tCmpScop Complimentary channel voltage data

void __stdcall FCNL_DefFcmp (FCMP *fcmp)

This function is used to fill the fcmp structure for the Fibre Channel Compliance tool with reasonable default values. It is recommended that this function be called initially even if parameters within the structure are to be adjusted manually, and may be called repeatedly to reestablish initial conditions; however, this will impact test time.

Before calling this function, zero out the FCMP structure using the standard memset() function to ensure that any information pertaining to dynamic memory allocation is cleaned out prior to using the structure.

INPUTS

fcmp - Pointer to a FCMP structure. Memory needs to be allocated by the caller.

OUTPUTS

None.

void __stdcall FCNL_ClrFcmp (FCMP *fcmp)

This function frees any dynamic memory that may have been allocated during previous data acquisitions and clears out the fcmp structure.

INPUTS

fcmp - Pointer to a FCMP structure. Memory needs to be allocated by the caller.

OUTPUTS

None.

EXAMPLE

```
static FCMP fibre;                              //declare fibre to a structure of type
                                                //FCMP
memset ( &fibre, 0, sizeof ( FCMP ) );          //clear the memory for fibre structure
FCNL_DefFcmp ( &fibre);                         //set fibre structures to default values

FCNL_RqstPkt ( ApiDevId, &fibre, WIND_FCMP );   //execute the measurement.
FCNL_RqstAll ( ApiDevId, &fibre, WIND_FCMP );   //get plot data (including tDcom plots)

FCNL_ClrFcmp ( &fibre);                         //deallocate the structure
```

2-25 FOLDED EYE DIAGRAM TOOL

The Folded Eye Tool is designed to provide an eye mask test to be applied to a repeating pattern. This allows a DSP Bandwidth Extension algorithm to be applied to improve the apparent front end performance. See the SIA-3000 User Manual for additional information concerning the Bandwidth Extension.

```
typedef struct
  {
  /* Input parameters  */
  PARM    tParm;             /* Contains acquisition parameters        */
  long    lPassCnt;          /* Acquisitions so far, set to 0 to reset */
  long    lPatnLen;          /* Pattern length in bit periods          */
  long    lScopRes;          /* Scope resolution in ps increments      */
  long    lInps;             /* Input selection, see defines above     */
  long    lVoff;             /* Voltage offset (mV)     - per channel  */
  long    lVdif;             /* Differential offset (mV)- per channel  */
  MASK    tMask;             /* Structure which holds mask definition  */
  double  dMargin;           /* Margin in percentage [-1.0 to 1.0]     */
  double  dBitRate;          /* Bit Rate, must be specified            */
  double  dAttn;             /* Attenuation factor (dB)                */
  /* Output parameters */
  long    lGood;             /* Flag indicates valid data in structure */
  long    lPad2;
  double  d1stEdge;          /* This value is used internally          */
  double  dNrmPkpk;          /* Vpp for Normal Channel Eye Diagrams    */
  double  dCmpPkpk;          /* Vpp for Complimentary Eye Diagrams     */
  double  dDifPkpk;          /* Vpp for Differential Eye Diagrams      */
  QTYS    qNorm;             /* Normal channel quantities              */
  QTYS    qComp;             /* Complimentary channel  quantities      */
  QTYS    qDiff;             /* Differential channel  quantities       */
  PLOT    tNrmScop;          /* Normal channel voltage data            */
  PLOT    tCmpScop;          /* Complimentary channel voltage data     */
  PLOT    tDifScop;          /* Differential voltage data              */
  char   *bNrmData;          /* Eye diagram of normal data             */
  long    lNrmRsvd;          /* This value is used internally          */
  char   *bCmpData;          /* Eye diagram of complimentary data      */
  long    lCmpRsvd;          /* This value is used internally          */
  char   *bDifData;          /* Eye diagram of differential data       */
  long    lDifRsvd;          /* This value is used internally          */
  } FEYE;
```

tParm A structure of type PARM that contains acquisition parameter. tParm is discussed in full detail in a previous section.

lPassCnt This parameter is a bi-directional structure element that tracks the number of acquisitions in the data set. This flag can be read after an execution or set prior to an execution. Setting this parameter to 0 essentially resets the accumulated data on the instrument. The value in the returned structure will be automatically incremented by the instrument.
Valid Entries: any integer greater than or equal to 0
Default: 0

lPatnLen This parameter configures the number of UI that are measured and folded into the Eye Mask.
Valid Entries: any integer greater than or equal to 1
Default: 40

lScopRes This parameter configures the sample interval and is entered in units of picoseconds.
Valid Entries: any integer greater than or equal to 1

Default: 2

lInps Input selection, can be any of the following:
SCOP_INPS_NORM +Input Only
SCOP_INPS_COMP –Input Only
SCOP_INPS_DIFF +Input minus -Input
Default: SCOP_INPS_DIFF

lVoff Offset voltage used for scope acquire, specified in mV
Default: 0

lVdif Differential offset voltage used for display, specified in mV
Default: 0

tMask MASK Structure which holds mask definition. See the definition above.
Defaults:

```
tMask.dXwdUI = 0.40
tMask.dXflUI = 0.20
tMask.dYiPct = 0.60
tMask.dV1Rel = 0.20
tMask.dV0Rel = 0.20
tMask.dVmask = 64e-3
tMask.dTmask = 700e-12
tMask.dV1pas = feye->tMask.dVmask * 0.75
feye->tMask.dV0pas = feye->tMask.dVmask * 0.75
tMask.dTflat = feye->tMask.dTmask * 3.0 / 7.0
```

dMargin Margin in percentage for Eye Mask [-1.0 to 1.0]
Default: 0

dBitRate Bit Rate, must be specified
Default: 2.5e9

dAttn Attenuation factor in dB, this is provided to allow the results to be scaled to compensate for external attenuation from sources such as probes.
Default: 0

lGood Flag indicates valid data in structure

d1stEdge Used internally, DO NOT ALTER!

dNrmPkpk Vpp for normal Channel scope data

dCmpPkpk Vpp for complimentary Channel scope data

dDifPkpk Vpp for differential Channel scope data

qNorm Normal channel quantities

qComp Complimentary channel quantities

qDiff Differential channel quantities

tNrmScop Normal channel voltage data, last pass only

tCmpScop Complimentary channel voltage data, last pass only

tDifScop Differential channel voltage data, last pass only

bNrmData, lNrmRsvd, bCmpData, lCmpRsvd, bDifData, lDifRsvd for internal use only, DO NOT ALTER or try to use.

void __stdcall FCNL_DefFeye (FEYE *feye)

This function is used to fill the feye structure for the Folded Eye tool with reasonable default values. It is recommended that this function be called initially even if parameters within the structure are to be adjusted manually, and may be called repeatedly to reestablish initial conditions; however, this will impact test time.

Before calling this function, zero out the FEYE structure using the standard memset() function to ensure that any information pertaining to dynamic memory allocation is cleaned out prior to using the structure.

INPUTS

feye - Pointer to a FEYE structure. Memory needs to be allocated by the caller.

OUTPUTS

None.

void __stdcall FCNL_ClrFeye (FEYE *feye)

This function frees any dynamic memory that may have been allocated during previous data acquisitions and clears out the feye structure.

INPUTS

feye - Pointer to a FEYE structure. Memory needs to be allocated by the caller.

OUTPUTS

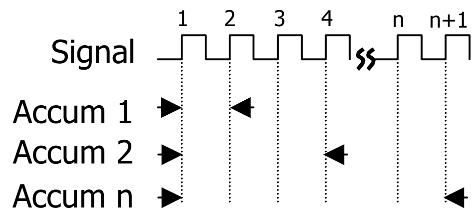
None.

EXAMPLE

```
static FEYE feye;                                 //declare feye to a structure of type
                                                  //FEYE
memset ( &feye, 0, sizeof ( FEYE ) );             //clear the memory for FEYE structure
FCNL_DefFeye ( &feye);                            //set FEYE structures to default values

FCNL_RqstPkt ( ApiDevId, &feye, WIND_FEYE );      //execute the measurement.
FCNL_RqstAll ( ApiDevId, &feye, WIND_FEYE );      //get plot data

FCNL_ClrFeye ( &feye);                            //deallocate the structure
```

2-26 HIGH FREQUENCY MODULATION ANALYSIS TOOL

The High Frequency Modulation Analysis Tool is used typically for frequency analysis of noise on clock and clock-like signals (101010…). The controls for the tool deal primarily with measurement setup, corner frequency selection and normalization technique.



This tool will take several randomly selected time measurements using Accumulated Time Analysis (ATA). The data can be displayed in the time domain (accumulated jitter versus time) or in the frequency domain (jitter versus frequency). This latter plot is used to identify spectral peaks in the noise which may indicate modulation and can typically be attributed to crosstalk or EMI effects.

The Jitter Analysis Tool can be set up to calculate RJ and DJ of a clock signal over a specified frequency band (typically the corner frequency to ½ the clock rate) and separate the DJ by frequency content. The DJ measured in this tool is strictly Periodic Jitter.

```
typedef struct
  {
  /* Input parameters  */
  PARM    tParm;             /* Contains acquisition parameters        */
  FFTS    tFfts;             /* FFT window and analysis parameters     */
  long    lIncStop;          /* Increase stop count by this value      */
  long    lMaxStop;          /* Maximum stop count to collect data     */
  long    lAutoFix;          /* If true calculate the above parameters */
  long    lPad1;
  double  dCornFrq;          /* Corner Frequency for RJ+PJ             */
  double  dRjpjFmn;          /* Minimum integration limit for RJ+PJ    */
  double  dRjpjFmx;          /* Maximum integration limit for RJ+PJ    */
  long    lFftAvgs;          /* 2^fft_avgs averages used to smooth FFT */
  /* Output parameters */
  long    lGood;             /* Flag indicates valid data in structure */
                             /*****************************************/
  double  dWndFact1Clk;      /* These values are used internally       */
  double  dWndFactNClk;      /*              DO NOT ALTER!             */
                             /*****************************************/
  PLTD    tSigm;             /* Contains the 1-Sigma plot array        */
  PLTD    tPeak;             /* Contains the ( max - min ) plot array  */
  PLTD    tFft1;             /* Frequency plot data on 1-clock basis   */
  double  dPjit1Clk;         /* Periodic jitter on 1-clk basis         */
  double  dRjit1Clk;         /* Random jitter on 1-clk basis           */
  long   *lPeakData1Clk;     /* Tracks detected spikes in RJ+PJ data   */
  long    lPeakNumb1Clk;     /* Count of detected spikes               */
  long    lPeakRsvd1Clk;     /* Used to track memory allocation        */
  long    lPad2;
  PLTD    tFftN;             /* Frequency plot data on N-clock basis   */
  double  dPjitNClk;         /* Periodic jitter on N-clk basis         */
  double  dRjitNClk;         /* Random jitter on N-clk basis           */
  long   *lPeakDataNClk;     /* Tracks detected spikes in RJ+PJ data   */
  long    lPeakNumbNClk;     /* Count of detected spikes               */
  long    lPeakRsvdNClk;     /* Used to track memory allocation        */
  long    lPad3;
  double  dFreq;             /* Carrier frequency                      */
  } JITT;
```

tParm A structure of type PARM that contains acquisition parameter. tParm is discussed in full detail in Section 2-4.

tFfts A structure of type FFTS that contains the setup parameters for the FFT. See Section 2-10 for further details on FFTS structures.

lIncStop Timing resolution of Accumulated Time Analysis. This value will define the highest frequency component that will be observed (low-pass filter function approximated by a brick wall)
Valid Entries: tParm.lStopCnt to lMaxStop.
Default: 1

lMaxStop Maximum number of accumulated periods to acquire. This value defines the low frequency cut off for this measurement. The larger this number is, the more lower-frequency modulation content can be observed. Furthermore, the larger this number is, the more data that is taken and the longer the test time.
Valid Entries: tParm.StopCnt to 10,000,000
Default: 256

lAutoFix Flag to indicate whether to use dCornFrq or lMaxStop to indicate the low-frequency cutoff. If the value is of this parameter is greater than zero, dCornFrq will be used to calculate the stop count. If this parameter is equal to zero, lMaxStop will be used.
Valid Entries: 0 – no pulsefind prior to measurement
1 –pulsefind if the measurement mode changed.
Default: 0

dCornFrq Corner Frequency for RJ & PJ estimate in Hertz. This value is used in conjunction with the measured clock frequency (F_{CM}) to determine the maximum number of accumulated periods used to acquire. A lower value increases acquisition time while capturing more low frequency data.
Valid Entries: F_{CM} /10,000,000 to F_{CM} I
Default: 637e3 (637kHz – Fibre Channel 1X)

dRjpjFmn High-pass digital filter function in Hertz for calculating RJ and DJ. A negative value disables filter. The accuracy of low frequency modulation measurements can be improved by setting the measurement corner frequency lower than the desired corner frequency and then using this filter for the RJ and PJ estimate.
Valid Entries: -1 to dCornFreq or Clock Frequency ÷ lMaxStop
Default: -1

dRjpjFmx Low-pass Digital filter function in Hertz for calculating RJ and DJ. A negative value disables filter. This filter is used as a post-processing filter applied to the measured data to limit high frequency information present in the data when calculating RJ-DJ estimate.
Valid Entries: -1 to Clock Frequency ÷ lIncStop
Default: -1

lFftAvgs This variable is used to calculate the number of averages to use in the FFT. Increasing the number of averages reduces the background noise associated with the FFT algorithm. The number of averages is calculated based on the equation:
AVERAGES = 2^n where n = lFftAvgs
Valid Entries: any integer greater than or equal to 0
Default: 0 (indicating 2^0 averages = 1 execution.)

lGood Flag indicates valid output data in structure. A positive value in this parameter indicates that the measurement was completed successfully, and, valid data can be extracted from this structure.

dWndFact1Clk, dWndFactNClk These values are for internal use only, DO NOT ALTER or try to use.

tSigm A structure of type PLTD containing the 1-Sigma plot array. This plot is used to observe the standard deviation (1σ) of accumulated jitter versus time. See Section 2-3 for details of the PLTD structure elements.

tPeak A structure of type PLTD containing the peak-to-peak Accumulated jitter versus time plot array. See Section 2-3 for details of the PLTD structure elements.

tFft1 A structure of type PLTD containing the Accumulated jitter versus frequency with amplitudes normalized to their effect on 1-clock. This is sometimes referred to as accumulated period jitter. See Section 2-3 for details of the PLTD structure elements.

dPjit1Clk Amplitude of the largest spectral component in the normalized accumulated jitter versus frequency (1-clock PJ estimate).

dRjit1Clk Random jitter calculated based on filter functions (if enabled) and Normalized Accumulated Jitter versus frequency plot (RJ as a function of 1-clock FFT).

lPeakData1Clk For internal use only, DO NOT ALTER or attempt to interpret.

lPeakNumb1Clk Count of detected spikes observed in the normalized Accumulated Jitter versus frequency plot. (spectral peaks in 1-clock FFT)

lPeakRsvd1Clk for internal use only, DO NOT ALTER or try to use.

tFftN A structure of type PLTD containing the Accumulated Jitter versus Frequency plot data. The amplitudes show the total amplitude of the modulation and is referred to as "N-clock" mode in reference to edge deviation due to a given modulation tone relative to an ideal clock. This is sometimes referred to as accumulated edge jitter. See Section 2-3 for details of the PLTD structure elements.

dPjitNClk Amplitude of the largest spectral component in the accumulated jitter versus frequency plot. (N-clock PJ estimate).

dRjitNClk Random jitter calculated based on filter functions (if enabled) and Accumulated Jitter versus frequency plot (RJ as a function of n-clock FFT).

lPeakDataNClk For internal use only, DO NOT ALTER or attempt to interpret.

lPeakNumbNClk Count of detected spikes observed in the accumulated jitter versus frequency plot. (spectral peaks in n-clock FFT)

lPeakRsvdNClk for internal use only, DO NOT ALTER or try to use.

dFreq Measured clock frequency.

void __stdcall FCNL_DefJitt (JITT *jitt)

This function is used to fill the jitt structure for the High Frequency Modulation tool with reasonable default values. It is recommended that this function be called initially even if parameters within the structure are to be adjusted manually, and may be called repeatedly to reestablish initial conditions; however, this will impact test time. Before calling this function, zero out the `JITT` structure using the standard `memset()` function to ensure that any information pertaining to dynamic memory allocation is cleaned out prior to using the structure.

INPUTS

jitt - Pointer to a JITT structure. Memory needs to be allocated by the caller.

OUTPUTS

None.

void __stdcall FCNL_ClrJitt (JITT *jitt)

This function frees any dynamic memory that may have been allocated during previous data acquisitions and clears out the jitt structure.

INPUTS

jitt - Pointer to a JITT structure. Memory needs to be allocated by the caller.

OUTPUTS

None.

EXAMPLE

```
#define TRUE 1
static JITT hfm;                                 //declare hfm to be a structure of
                                                 //type JITT
memset ( &hfm, 0, sizeof ( JITT ) );             //clear the memory for hfm structure
FCNL_DefJitt ( &hfm);                            //set hfm structure to default values
                                                 //NOTE:  hfm.tparm & hfm.tFfts
                                                 //are also set to defaults by this
                                                 //command.
hfm.tparm.lChanNum = 1;                          //perform measurement on CH1
hfm.tparm.lSampCnt = 500;                        //measure 500 different samples per
                                                 //accumulated edge
hfm.lAutoFix = TRUE;                             //use dCornFrq instead of lMaxStop
hfm.dCornFrq = 2e6;                              //set corner frequency to 2MHz
FCNL_RqstPkt ( ApiDevId, &hfm, WIND_JITT );      //execute the measurement.
FCNL_RqstAll ( ApiDevId, &hfm, WIND_JITT );      //get plot data

FCNL_ClrJitt ( &hfm);                            //deallocate the structure
```

2-27 HISTOGRAM TOOL

The histogram tool is used for displaying the statistical distribution of a given measurement. Measurements made with this tool are limited to repetitive signal measurements such as clock period, duty cycle, pulse width, rise time, fall time, propagation delay and frequency. This tool is typically used for displaying the statistical distribution of thousands of measurements. Important distribution parameters can be calculated based on the data including: RMS, peak to peak, Random Jitter (RJ), Deterministic Jitter (DJ) and Total Jitter (TJ).

```
typedef struct
  {
  /* Input parameters  */
  PARM    tParm;             /* Contains acquisition parameters        */
  double  dUnitInt;          /* Unit Interval to assess Total Jitter   */
  long    lPassCnt;          /* Acquisitions so far, set to 0 to reset */
  long    lErrProb;          /* Error probability for Total Jitter     */
                             /* Valid range is ( -1 to -16 )           */
  long    lTailFit;          /* If non-zero a tail-fit will be tried   */
  long    lForcFit;          /* If non-zero use the force-fit method   */
  long    lMinHits;          /* Minimum hits before trying tail-fit    */
  long    lFndEftv;          /* Flag to attempt effective jitter calc  */
  long    lMinEftv;          /* Min probability for effective fit: -4  */
  long    lMaxEftv;          /* Max probability for effective fit: -12 */
  long    lAutoFix;          /* If true perform a pulsefind as req'd   */
  long    lKeepOut;          /* If non-zero use tailfit keep out below */
  double  dKpOutLt;          /* Keep out value for left side           */
  double  dKpOutRt;          /* Keep out value for right side          */
  long    lPad0;             /* Output parameters                      */
  long    lGood;             /* Flag indicates valid data in structure */

  long    lPad1;
  long    lNormCnt;          /* Number of hits in normal edge data     */
  double  dNormMin;          /* Minimum value in normal edge data      */
  double  dNormMax;          /* Maximum value in normal edge data      */
  double  dNormAvg;          /* Average value of normal edge data      */
  double  dNormSig;          /* 1-Sigma value of normal edge data      */

  long    lPad2;
  long    lAcumCnt;          /* Number of hits in accumulated edge data*/
  double  dAcumMin;          /* Minimum value in accumulated edge data */
  double  dAcumMax;          /* Maximum value in accumulated edge data */
  double  dAcumAvg;          /* Average value of accumulated edge data */
  double  dAcumSig;          /* 1-Sigma value of accumulated edge data */

  long    lBinNumb;          /*****************************************/
  long    lPad3;             /*  These values are all used internally  */
  double  dLtSigma[PREVSIGMA];/*   as part of the measurement process   */
  double  dRtSigma[PREVSIGMA];/*            DO NOT ALTER!              */
  double  dFreq;             /*****************************************/

  PLTD    tNorm;             /* Histogram of previous acquisition      */
  PLTD    tAcum;             /* Histogram of all acquires combined     */
  PLTD    tMaxi;             /* Histogram of max across all acquires   */
  PLTD    tBath;             /* Bathtub curves determined from PDF     */
  PLTD    tEftv;             /* Effective Bathtub curves if enabled    */
  PLTD    tShrt;             /* Total Jitter for SHORT Cycles          */
  PLTD    tLong;             /* Total Jitter for LONG Cycles           */
  PLTD    tBoth;             /* Total Jitter for LONG & SHORT Cycles   */
  TFIT    tTfit;             /* Structure containing tail-fit info     */
  } HIST;
```

tParm A structure of type PARM that contains acquisition parameters. tParm is discussed in full detail in Section 2-4.

dUnitInt Unit Interval (UI) in seconds to assess Total Jitter as a percent of UI. Set this parameter as the metric against which TJ will be evaluated as a percentage. It is displayed as the span of the x-axis in a bathtub curve. This parameter is only used if tail-fit is enabled.
Valid Entries: any number greater than 0 which represents the time (in seconds) of a bit period or unit interval.
Default: 1e-9 (1ns)

lPassCnt This parameter is a bi-directional structure element that tracks the number of acquisitions in the data set. This flag can be read after an execution or set prior to an execution. Setting this parameter to 0 essentially resets the accumulated data on the instrument. The value in the returned structure will be automatically incremented by the instrument.
Valid Entries: any integer greater than or equal to 0
Default: 0

lErrProb Error probability level for Total Jitter. Total Jitter is calculated based on the desired Error Probability level. This value is used in conjunction with the bathtub curve after the successful completion of a tail-fit in order to project the value of Total Jitter.
Valid Entries: -1 to -16
Default: -12

lTailFit Flag to indicate whether to perform a TailFit on data in tAcum data array. If non-zero, a tail-fit will be attempted on the **tAcum** data array. The lGood element of the tTfit structure will indicate if the TailFit was successful. Any positive interger for this parameter will initiate the TailFit algorithm.
Valid Entries: 0 – disable TailFit
1 – enable TailFit
Default: 0

lForcFit If non-zero uses the force-fit method. If set to zero, the measurement will continue to loop until a reasonably accurate TailFit can be achieved.
Valid Entries: 0 – do not use force fit.
1 – force a fit using lMinHits number of hits.
Default: 0

lMinHits Minimum hits before attempting a tail-fit in 1000's; the default is 50. The larger the number the more likely a valid tailfit will be found.
Valid Entries: any integer ≥ 50
Default: 50

lFndEftv Flag to indicate that an effective jitter calculation is to be attempted. This is necessary for those instances in which correlation to a BERT scan is necessary. In all other practical applications, this parameter and it’s resultant measurement should be ignored.
Valid Entries: 0 – do not estimate effective jitter values
1 – calculate effective jitter values
Default: 0

lMinEftv, lMaxEftv Defines the range of the bathtub curve that is to be used to calculate an effective jitter value.
Valid Entries: -1 to –16 with lMinEftv < lMaxEftv
Default: -4 for MaxEftv and –12 for MinEftv

lAutoFix Flag indicating whether to perform a pulse-find as required. Setting this value to any integer greater than zero tells the measurement to perform a pulse find if needed. The system will know if a measurement was recently performed and if a pulse find is necessary.
Valid Entries: 0 – no pulsefind prior to measurement
1 –pulsefind if the measurement mode changed.
Default: 0

lGood Flag indicates valid output data in structure. This parameter does not indicate success of TailFit measurement only whether a valid time measurement was performed and valid measurement data was placed in tNorm, tAcum and tMaxi.

lNormCnt Number of measurements in tNorm plot array.

dNormMin, dNormMax Minimum and maximum values in tNorm plot array.

dNormAvg Average value of distribution in tNorm plot array.

dNormSig Standard Deviation (1-Sigma (1σ)) value of distribution in tNorm plot array.

lAcumCnt Number of hits of distribution in tAcum plot array.

dAcumMin, dAcumMax Minimum and maximum values of distribution in tAcum plot array.

dAcumAvg Average value of distribution in tAcum plot array.

dAcumSig 1-Sigma value of distribution in tAcum plot array.

lBinNumb, dLtSigma, dRtSigma These values are for internal use only, DO NOT ALTER or try to use.

tNorm A structure of type PLTD containing a Histogram of data from latest acquisition only. See Section 2-3 for further details on PLTD structures.

tAcum A structure of type PLTD containing Histogram of data from all acquisitions combined. See Section 2-3 for further details on PLTD structures.

tMaxi A structure of type PLTD containing Histogram with the maximum value obtained for every particular bin across all of the acquisitions performed so far. See Section 2-3 for further details on PLTD structures.

tBath A structure of type PLTD containing Bathtub curves determined from PDF, only valid when a successful tail-fit has been performed. See Section 2-3 for further details on PLTD structures.

tEftv A structure of type PLTD containing Effective Bathtub curves if lFndEftv is set and a valid fit is obtained. Effective Bathtub curves are used for correlation to BERT scan only. See Section 2-3 for further details on PLTD structures.

tTfit A structure of type TFIT containing tail-fit info; only valid when a successful tail-fit has been performed. See end of chapter for additional details. See Section 2-3 for further details on TFIT structures.

void __stdcall FCNL_DefHist (HIST *hist)

This function is used to fill the hist structure for the Histogram tool with reasonable default values. It is recommended that this function be called initially even if parameters within the structure are to be adjusted manually, and may be called repeatedly to reestablish initial conditions; however, this will impact test time.

Before calling this function, zero out the HIST structure using the standard memset() function to ensure that any information pertaining to dynamic memory allocation is cleaned out prior to using the structure.

INPUTS

hist - Pointer to a HIST structure. Memory needs to be allocated by the caller.

OUTPUTS

None.

void __stdcall FCNL_ClrHist (HIST *hist)

This function frees any dynamic memory that may have been allocated during previous data acquisitions and clears out the hist structure.

INPUTS

hist - Pointer to a HIST structure. Memory needs to be allocated by the caller.

OUTPUTS

None.

EXAMPLE

```
#define TRUE 1
static HIST histogram;                          //declare histogram to be a structure of
                                                //type HIST
memset ( &histogram, 0, sizeof ( HIST ) );      //clear the memory for histogram str.
FCNL_DefHist ( &histogram);                     //set histogram structures to default
                                                //values
histogram.tparm.lChanNum = 1;                   //capture waveform on channel 1
histogram.tparm.lFuncNum = FUNC_PER;            //set measurement to be period
histogram.tparm.lStrtCnt = 1;                   //measure from first edge to second
histogram.tparm.lStpCnt = 2;                    //edge
histogram.tparm.lSampCnt = 10,000;              //measure 10,000 samples per burst
histogram.lPassCnt = 0;                         //reset pass count to zero
histogram.lTailFit = TRUE;                      //indicate TailFit desired
histogram.lMinHits = 50,000;                    //don't attempt a TailFit until at least
                                                //50,000 measurements are
                                                //accumulated
histogram.lAutoFix = TRUE;                      //perform pulse find initially if needed.
FCNL_RqstPkt ( ApiDevId, &histogram, WIND_HIST );//execute the measurement.
FCNL_RqstAll ( ApiDevId, &histogram, WIND_HIST );//get plot data

FCNL_ClrHist ( &histogram);                     //deallocate the structure
```

2-28 INFINIBAND TOOL

This tool is similar to the Random Data With Bitclock Tool, but also provides voltage information.

```
typedef struct
  {
  /* Input parameters  */
  long    lVoff;              /* Offset voltage used for scope acquire  */
  long    lPad1;
  double  dAttn;              /* Attenuation factor (dB)                */
  EYEH    tEyeh;              /* EYEH structure holds most information  */
  /* Output parameters */
  long    lGood;              /* Flag indicates valid data in structure */
  long    lPad2;
  PLTD    tNrmScop;           /* Normal channel voltage data            */
  PLTD    tCmpScop;           /* Complimentary channel voltage data     */
  PLTD    tDifScop;           /* Differential voltage data              */
  PLTD    tComScop;           /* Common (A+B) voltage data              */
  } INFI;
```

lVoff Offset voltage used for scope acquire, specified in mV

dAttn Attenuation factor in dB, this is provided to allow the results to be scaled to compensate for external attenuation from sources such as probes.

Default: 0

tEyeh This is the same structure as is defined in the Random Data With Bitclock tool. It contains all the acquisition parameters and all the output results associated with this measurement, with the exception of those defined directly above.

Default: See Random Data With Bitclock Tool

lGood Flag indicates valid data in structure

tNrmScop Normal channel voltage data

tCmpScop Complimentary channel voltage data

tDifScop Differential voltage data

tComScop Common (A+B) voltage data

void __stdcall FCNL_DefInfi (INFI *infi)

This function is used to fill the infi structure for the Infiniband Compliance tool with reasonable default values. It is recommended that this function be called initially even if parameters within the structure are to be adjusted manually, and may be called repeatedly to reestablish initial conditions; however, this will impact test time.

Before calling this function, zero out the INFI structure using the standard memset() function to ensure that any information pertaining to dynamic memory allocation is cleaned out prior to using the structure.

INPUTS

infi - Pointer to a INFI structure. Memory needs to be allocated by the caller.

OUTPUTS

None.

void __stdcall FCNL_ClrInfi (INFI *infi)

This function frees any dynamic memory that may have been allocated during previous data acquisitions and clears out the infi structure.

INPUTS

infi - Pointer to a INFI structure. Memory needs to be allocated by the caller.

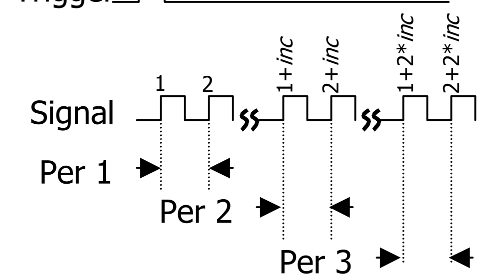
OUTPUTS

None.

EXAMPLE

```
static INFI iband;                               //declare iband to a structure of type
                                                 //INFI
memset ( &iband, 0, sizeof ( INFI ) );           //clear the memory for iband structure
FCNL_DefInfi ( &iband);                          //set iband structures to default values

FCNL_RqstPkt ( ApiDevId, &iband, WIND_INFI );    //execute the measurement.
FCNL_RqstAll ( ApiDevId, &iband, WIND_INFI );    //get plot data (including tEyeh)

FCNL_ClrInfi ( &iband);                          //deallocate the structure
```

2-29 LOCKTIME ANALYSIS TOOL



The Locktime Analysis tool is used to analyze timing measurement variation as a function of location in pattern. This is important when measuring periods, pulse widths, slew rates and propagation delay right after an event such as a reset, power-up, data bus read/write, chip enable, ref clock enable etc. Common measurements include PLL lock time and cross talk sensitivity to specific functionalities occurring on the DUT. The Locktime Analysis Tool makes several measurements of the same event after a trigger and then can increment to the next event. For example, a period measurement could be made on the first clock pulse after a trigger occurs. This measurement could be made hundreds of times. Then, this tool automatically will increment to the next clock period and measure that one hundred times. This is repeated for as many sequential periods as desired. The increment and the number of measurements is programmed by the user.

```
typedef struct
  {
  /* Input parameters  */
  PARM    tParm;              /* Contains acquisition parameters        */
  FFTS    tFfts;              /* FFT window and analysis parameters     */
  long    lIncStrt;           /* Increase start count by this value     */
  long    lMaxStrt;           /* Maximum start count to collect data    */
  long    lAnlMode;           /* Relationship of start and stop counts  */
                              /* Use one of:  ANL_FNC_FIRST             */
                              /*              ANL_FNC_PLUS1             */
                              /*              ANL_FNC_START             */
  long    lAutoFix;           /* If true calculate the above parameters */
  long    lSpanCnt;           /* The span across which to measure       */
  long    lDataPts;           /* The data points within span to measure */
  /* Output parameters */
  long    lGood;              /* Flag indicates valid data in structure */
  long    lPad1;
  PLTD    tTime;              /* Time domain plot data                  */
  PLTD    tDerv;              /* 1st derivative of time domain plot data*/
  PLTD    tFftT;              /* Frequency domain plot data             */
  PLTD    tFftD;              /* Frequency domain of 1st derivative     */
  PLTD    tSigm;              /* Contains the 1-Sigma plot array        */
  PLTD    tPeak;              /* Contains the ( max - min ) plot array  */
  PLTD    tMini;              /* Contains the Minimum plot array        */
  PLTD    tMaxi;              /* Contains the Maximum plot array        */
  double  dSigmAvg;           /* Average 1-Sigma value                  */
  double  dSigmMin;           /* Minimum 1-Sigma value                  */
  double  dSigmMax;           /* Maximum 1-Sigma value                  */

  double  dTimePos;           /* Maximum increase between time values   */
  double  dTimeNeg;           /* Maximum decrease between time values   */
  long    lTimePosLoc;        /* Index to max increase between values   */
  long    lTimeNegLoc;        /* Index to max decrease between values   */

  double  dDervPos;           /* Maximum increase between 1st deriv's   */
  double  dDervNeg;           /* Maximum decrease between 1st deriv's   */
  long    lDervPosLoc;        /* Index to max incr. between 1st deriv's */
  long    lDervNegLoc;        /* Index to max decr. between 1st deriv's */

  double  dFreq;              /* Carrier frequency                      */
  } FUNC;
```

tParm A structure of type PARM that contains acquisition parameter. The PARM structure is discussed in full detail in Section 2-4.

tFfts A structure of type FFTS that contains the setup parameters for the FFT. See Section 2-10 for further details on FFTS structures.

lIncStrt Resolution of successive time measurements. This parameter defines the number edges to skip between successive measurements. Increase start count by this value, the default is 1. Data is collected for start counts ranging from tParm.lStrtCnt to lMaxStrt.
Valid Entries: 1 to lMaxStrt
Default: 1

lMaxStrt Maximum start count used. The start count will be incremented from the value in tParm.lStrtCnt to lMaxStrt in step size of lIncStrt.
Valid Entries: tParm.StrtCnt to 10,000,000
Default: 250

lAnlMode Relationship of start and stop counts. In general, this measurement is done either on a single channel measuring successive cycles' slew rate, period or pulse width. As such, the stop count will always be either equal to the start count or one more than the start count in the case of period measurements.

Valid Entries:	ANL_FNC_PLUS1	Stop Count = Start Count + 1 Use this for period measurements
	ANL_FNC_START	Stop Count = Start Count Use this for skew, slew rate and pulse width
Default:	ANL_FNC_PLUS1	

lAutoFix If set to 1, calculate the number of measurements skipped and the total number of measurements based on lSpanCnt and lDataPts plus information measured on the live data signal.

Valid Entries:	0	use lMaxStrt, tParm.lStrtCnt & lIncStrt to calculate the stop counts for each measurement.
	1	use lSpanCnt, DataPts and measured data from signal to calculate the stop counts for each measurement.
Default:	0	

lSpanCnt The total number of edges across which to measure. This is the maximum delay count for a measurement and is synonymous with lMaxStrt.
Valid Entries: 1 to 10,000,000-tParm.StrCnt
Default: 1000

lDataPts The total data points within span to measure. If every data point is to be measured such that the start and stop counters are incremented by one, then lDataPts must equal lSpanCnt. The
Valid Entries: 1 to lSpanCnt
Default: 100

lGood Flag indicates valid output data in structure.

tTime A structure of type PLTD containing the time domain plot data. See Section 2-3 for details on the PLTD structure elements.

tDerv A structure of type PLTD containing 1st derivative of time domain plot data. See Section 2-3 for details on the PLTD structure elements.

tFftT A structure of type PLTD containing Frequency domain plot data. See Section 2-3 for details on the PLTD structure elements.

tFftD	A structure of type PLTD containing Frequency domain of 1st derivative plot data. See Section 2-3 for details on the PLTD structure elements.
tSigm	A structure of type PLTD containing 1-Sigma plot array. See Section 2-3 for details on the PLTD structure elements.
tPeak	A structure of type PLTD containing the (max - min) plot array. See Section 2-3 for details on the PLTD structure elements.
tMini	A structure of type PLTD containing the Minimum plot array. See Section 2-3 for details on the PLTD structure elements.
tMaxi	A structure of type PLTD containing the Maximum plot array. See Section 2-3 for details on the PLTD structure elements.
dSigmAvg	Average 1-Sigma value.
dSigmMin	Minimum 1-Sigma value.
dSigmMax	Maximum 1-Sigma value.
dTimePos	Maximum increase between time values.
dTimeNeg	Maximum decrease between time values.
lTimePosLoc	Index to maximum increase between values.
lTimeNegLoc	Index to maximum decrease between values.
dDervPos	Maximum increase between 1st derivative values.
dDervNeg	Maximum decrease between 1st derivative values.
lDervPosLoc	Index to maximum increase between 1st derivative values.
lDervNegLoc	Index to maximum decrease between 1st derivative values.
dFreq	Carrier frequency.

void __stdcall FCNL_DefFunc (FUNC *func)

This function is used to fill the func structure for the Locktime tool with reasonable default values. It is recommended that this function be called initially even if parameters within the structure are to be adjusted manually, and may be called repeatedly to reestablish initial conditions; however, this will impact test time.

Before calling this function, zero out the FUNC structure using the standard memset() function to ensure that any information pertaining to dynamic memory allocation is cleaned out prior to using the structure.

INPUTS

func - Pointer to a FUNC structure. Memory needs to be allocated by the caller.

OUTPUTS

None.

void __stdcall FCNL_ClrFunc (FUNC *func)

This function frees any dynamic memory that may have been allocated during previous data acquisitions and clears out the func structure.

INPUTS

func - Pointer to a FUNC structure. Memory needs to be allocated by the caller.

OUTPUTS

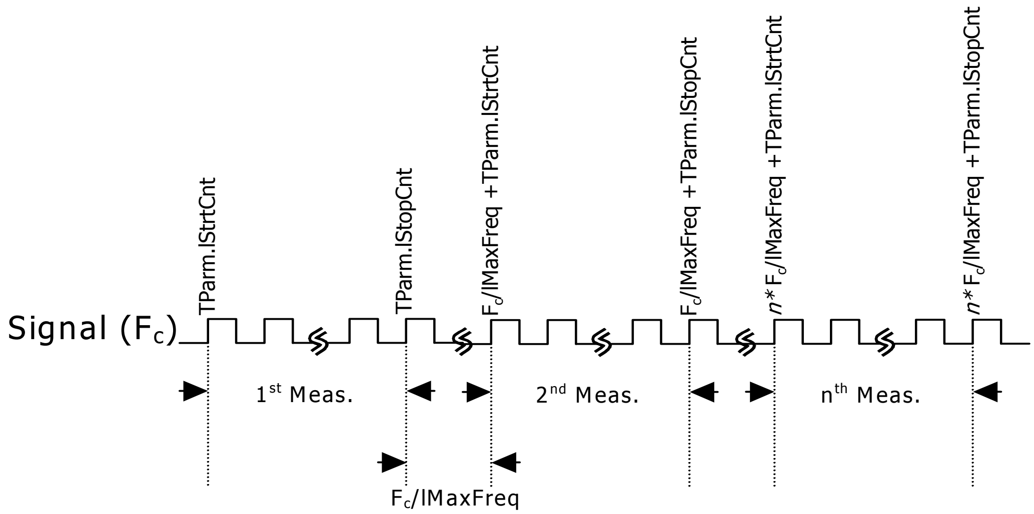
None.

EXAMPLE

```
#define TRUE 1
static FUNC funcAnal;                               //declare funcAnal to be a structure of
                                                    //type FUNC
memset ( &funcAnal, 0, sizeof ( FUNC ) );          //clear the memory for funcAnal structure
FCNL_DefFunc ( &funcAnal);                         //set funcAnal structure to default values
                                                    //NOTE:  funcAnal.tparm & funcAnal.tFfts
                                                    //are also set to defaults by this
                                                    //command.
funcAnal.tparm.lChanNum = 1;                       //perform measurement on CH1
funcAnal.tparm.lSampCnt = 500;                     //measure 500 different samples per
                                                    //offset from trigger
funcAnal.lIncStrt = 1;                             //set increment between successive
                                                    //period measurements to 1
funcAnal.lMaxStrt = 1000;                          //Capture all period measurements
                                                    //after the trigger up to and including
                                                    //the period 1000 cycles after the
                                                    //trigger.

FCNL_RqstPkt ( ApiDevId, & funcAnal, WIND_FUNC );  //execute the measurement.
FCNL_RqstAll ( ApiDevId, & funcAnal, WIND_FUNC );  //get plot data

FCNL_ClrFunc ( &funcAnal);                         //deallocate the structure
```

2-30 LOW FREQUENCY MODULATION ANALYSIS TOOL

The Low Frequency Modulation Analysis tool is used to analyze low frequency modulation on clock signals. It uses its internal time stamp capability to identify when a given measurement is made. This tool combines the actual time measurements with the relative time each measurement was made to identify low frequency modulation components. This tool can be used for modulation frequencies below 120kHz.




```
typedef struct
  {
  /* Input parameters  */
  PARM    tParm;             /* Contains acquisition parameters        */
  FFTS    tFfts;             /* FFT window and analysis parameters     */
  long    lAutoFix;          /* If true calculate the above parameters */
  long    lPad1;
  double  dMaxFreq;          /* Maximum Frequency that is desired      */
  long    lFftAvgs;          /* 2^fft_avgs averages used to smooth FFT */
  /* Output parameters */
  long    lGood;             /* Flag indicates valid data in structure */
  PLTD    tTime;             /* Time domain plot data                  */
  PLTD    tStmp;             /* Time stamp array, not normally plotted */
  PLTD    tFft1;             /* Frequency plot data on 1-clock basis   */
  PLTD    tFftN;             /* Frequency plot data on N-clock basis   */
  double  dCarFreq;          /* Carrier frequency                      */
  double  dSmpRate;          /* Sampling rate                          */
  double  dFftNdBc;          /* dBc assessed on 1-clock FFT data       */
  } TDIG;
```

tParm A structure of type PARM that contains acquisition parameters. The PARM structure is discussed in full detail in Section 2-4. tParm.lStampTm is enabled for this tool by default. All other defaults listed in Section 2-4 apply.

tFfts A structure of type FFTS that contains the FFT setup parameters such as window type and padding factor. See Section 2-10 for further details.

lAutoFix This tool uses tParm.lSampCnt to define the number of measurements to make and the span of tParm.lStrCnt to tParm.lStopCnt to define the maximum frequency observed in the FFT plots. If this structure element is set to 1, then tParm.StrCnt and

	tParm.lStopCnt will be calculated based on dMaxFreq plus information measured on the live data signal. Valid Entries: 0 – use tParm data 1 – calculate tParm data using dMaxFreq Default: 0
dMaxFreq	Maximum Frequency information that is desired.
lFftAvgs	This variable is used to calculate the number of averages to use in the FFT. Increasing the number of averages reduces the background noise associated with the FFT algorithm. The number of averages is calculated based on the equation: AVERAGES = 2^n where n = lFftAvgs Valid Entries: any integer greater than or equal to 0 Default: 0 (indicating 2^0 averages = 1 execution.)
lGood	Flag to indicate valid output data is in structure.
tTime	A structure of type PLTD containing the time domain plot data. See Section 2-3 for details on the PLTD structure elements.
tStmp	A structure of type PLTD containing time stamp data plot data. This is not normally plotted. See Section 2-3 for details on the PLTD structure elements.
tFft1	A structure of type PLTD containing the Frequency plot data with frequency amplitude roll off of 20dB/decade from the sampling Nyquist Frequency. This plot is typically used for debug purposes only. See Section 2-3 for details on the PLTD structure elements.
tFftN	A structure of type PLTD containing the Frequency plot data with amplitudes representing the cumulative effect of the frequency component. See Section 2-3 for details on the PLTD structure elements.
dCarFreq	Carrier frequency.
dSmpRate	Sampling rate.
dFftNdBc	dBc assessed on 1-clock FFT data.

void __stdcall FCNL_DefTdig (TDIG *tdig)

This function is used to fill the `tdig` structure for the Low Frequency Modulation tool with reasonable default values. It is recommended that this function be called initially even if parameters within the structure are to be adjusted manually, and may be called repeatedly to reestablish initial conditions; however, this will impact test time.

Before calling this function, zero out the `TDIG` structure using the standard `memset()` function to ensure that any information pertaining to dynamic memory allocation is cleaned out prior to using the structure.

INPUTS

tdig - Pointer to a TDIG structure. Memory needs to be allocated by the caller.

OUTPUTS

None.

void __stdcall FCNL_ClrTdig (TDIG *tdig)

This function frees any dynamic memory that may have been allocated during previous data acquisitions and clears out the tdig structure.

INPUTS

tdig - Pointer to a TDIG structure. Memory needs to be allocated by the caller.

OUTPUTS

None.

EXAMPLE

```
#define TRUE 1
static TDIG TimDig;                                    //declare timDig to be a structure of
                                                  //type TDIG
memset ( &TimDig, 0, sizeof ( TDIG ) );           //clear the memory for timDig structure
FCNL_DefTdig ( &TimDig);                          //set timDig structure to default values
                                                  //NOTE:  timDig.tparm & timDig.tFfts
                                                  //are also set to defaults by this
                                                  //command.
TimDig.tParm.lChanNum = 1;                        //Set channel number to 1
TimDig.tparm.lStrtCnt = 1;                        //Measure from 1st rising edge
TimDig.tParm.lStopCnt = 10000;                    //to 10,000th rising edge for each meas.
TimDig.tParm.lSampCnt = 100000;                   //capture 100,000 measurements per
                                                  //pass
TimDig.lFftAvgs = 3;                                   //Perform 23 passes or 8 total passes
                                                  //with which to average data in FFT.

FCNL_RqstPkt ( ApiDevId, & TimDig, WIND_TDIG );        //execute the measurement.
FCNL_RqstAll ( ApiDevId, & TimDig, WIND_TDIG );        //get plot data

FCNL_ClrTdig ( &TimDig);                          //deallocate the structure
```

2-31 OSCILLOSCOPE TOOL

The Oscilloscope Tool is typically used to view the waveform of a signal relative to a trigger. In a diagnostic environment, this tool is essential when debugging any signal measurement challenge. In a production environment, this capability is used to make voltage measurements on signals such as amplitude, glitch energy, overshoot and undershoot. This section describes the structure used to initiate a waveform capture. This is the original measurement window structure for conducting an oscilloscope measurement, and was later replaced by the Scope Tool, but is still supported for legacy operations.

```
typedef struct
  {
  /* Input parameters  */
  PARM    tParm;              /* Contains acquisition parameters        */
  FFTS    tFfts;              /* FFT window and analysis parameters     */
  long    lStrt;              /* Start time (ps), 20,000 to 100,000,000 */
  long    lStop;              /* Stop time (ps),  20,000 to 100,000,000 */
  long    lIncr;              /* Time increment (ps), minimum is 10     */
  /* Output parameters */
  long    lGood;              /* Flag indicates valid data in structure */
  PLTD    tTime[ POSS_CHNS ]; /* Time domain plot of voltage data       */
  PLTD    tFreq[ POSS_CHNS ]; /* Frequency domain plot of voltage data  */
  PLTD    tNorm[ POSS_CHNS ]; /* Normal channel voltage data (3000 only)*/
  PLTD    tComp[ POSS_CHNS ]; /* Complimentary  voltage data (3000 only)*/
  } OSCI;
```

tParm A structure of type PARM that contains acquisition parameter. See Section 2-4 for further details concerning this structure.

tFfts A structure of type FFTS that contains setup parameters for the FFT window. These parameters needs to be set if the user is interested in capturing the spectrum analysis on the waveform. See Section 2-10 for further details concerning this structure.

lStrt Start time in picoseconds.
Valid Entries: (24,000 to 100,000,000)
Default: 24,000

lStop Stop time in picoseconds
Valid Entries: (24,000 to 100,000,000)
Default: 100,000

lIncr Resolution of time base in picoseconds. Maximum Resolution is equal to the window width (lStop - lStrt), such that only 2 data points would be captured.
Valid Entries: (10 to *window width*)
Default: 500

lGood Flag indicates waveform capture was successful and valid output data is in the structure.

tTime[*n*] A structure of type PLTD which contains the differential time domain plot of voltage data for channel *n*. See Section 2-3 for further details on PLTD structures.

tFreq[*n*] A structure of type PLTD which contains the differential frequency domain plot of voltage data for channel *n*. See Section 2-3 for further details on PLTD structures.

tNorm[*n*] A structure of type PLTD which contains the single ended time domain plot of the positive channel voltage information for channel *n*. See Section 2-3 for further details on PLTD structures.

tComp[*n*] A structure of type PLTD which contains the single ended time domain plot of the negative channel voltage information for channel *n*. See Section 2-3 for further details on PLTD structures.

void __stdcall FCNL_DefOsci (OSCI *osci)

This function is used to fill the osci structure for the Oscilloscope tool with reasonable default values. It is recommended that this function be called initially even if parameters within the structure are to be adjusted manually, and may be called repeatedly to reestablish initial conditions; however, this will impact test time.

Before calling this function, zero out the OSCI structure using the standard memset() function to ensure that any information pertaining to dynamic memory allocation is cleaned out prior to using the structure.

INPUTS

osci - Pointer to a OSCI structure. Memory needs to be allocated by the caller.

OUTPUTS

None.

void __stdcall FCNL_ClrOsci (OSCI *osci)

This function frees any dynamic memory that may have been allocated during previous data acquisitions and clears out the osci structure.

INPUTS

osci - Pointer to a OSCI structure. Memory needs to be allocated by the caller.

OUTPUTS

None.

EXAMPLE

```
static OSCI oscope;                             //declare oscope to a structure of type
                                                //OSCI
memset ( &oscope, 0, sizeof ( OSCI ) );         //clear the memory for oscope structure
FCNL_DefOsci ( &oscope);                        //set oscope structures to default values

oscope.tparm.lChanNum = 1;                      //capture waveform on channel 1
oscope.tparm.lOscTrig = 2;                      //trigger on channel 2
oscope.tparm.lOscEdge = EDGE_RISE;              //trigger on rising edge of channel 2
oscope.tparm.lFndPcnt = PCNT_5050;              //set trigger level at 50% point

oscope.lStrt = 200;                             //start waveform capture at 200ps
oscope.lStop = 10,000;                          //stop waveform capture at 10ns
oscope.lIncr = 10;                              //set resolution to 10ps (this means
                                                //that there will be 980 points in
                                                //oscope.tTime[1].data array

FCNL_RqstPkt ( ApiDevId, &oscope, WIND_OSCI );  //execute the measurement.
FCNL_RqstAll ( ApiDevId, &oscope, WIND_OSCI );  //get plot data

FCNL_ClrOsci ( &oscope);                        //deallocate the structure
```

2-32 PCI EXPRESS 1.1 WITH HARDWARE CLOCK RECOVERY TOOL

The PCI Express 1.1 with Hardware Clock Recovery Tool provides both timing and amplitude compliance measurements using the SIA3000 Multirate Clock Recovery Option. This tool accurately determines device performance by quantifying both random and deterministic jitter components.

```
typedef struct
  {
  /* Input parameters  */
  long    lCompPnt;          /* Compliance Point 0-RX 1-TX            */
  long    lPcnt;             /* Amount +/- 50% to calc. rise/fall time */
  long    lHiRFmV;           /* Absolute rise/fall voltage if lPcnt<0  */
  long    lLoRFmV;           /* Absolute rise/fall voltage if lPcnt<0  */
  long    lIdleOk;           /* Common mode idle voltages are valid    */
  long    lPad0;
  double  dAttn;             /* Attenuation factor (dB)                */
  RCPM    tRcpm;             /* Contains acquisition parameters        */
  /* Output parameters */
  long    lGood;             /* Flag indicates valid data in structure */
  long    lPad1;
  double  dEyeOffs;
  double  dXmnDiff;
  double  dXmxDiff;
  double  dVdiffPP;          /* Pk-pk differential voltage             */
  double  dVdRatio;          /* De-emphaisis voltage ratio             */
  double  dOpnEyeT;          /* Eye opening                            */
  double  dMedEyeT;          /* Median to max jitter                   */
  double  dOpnEyeT1M;        /* Eye opening @ 10^-6 BER                */
  double  dTranVolts;        /* Vpp for Transition Eye                 */
  double  dDeemVolts;        /* Vpp for De-Emphasis Eye                */

  double  dVcommonAc;        /* V?x-cm-acp                             */
  double  dVcommonDc;        /* V?x-cm-dc                              */
  double  dVcmDcActv;        /* V?x-cm-dc-active-idle-delta            */
  double  dVcmIdleDc;        /* V?x-cm-idle-dc                         */
  double  dVcmDcLine;        /* V?x-cm-dc-line-delta                   */
  double  dVcmDcDpls;        /* V?x-cm-dc-d+                           */
  double  dVcmDcDmin;        /* V?x-cm-dc-d-                           */
  double  dVIdleDiff;        /* V?x-idle-diffp                         */

  QTYS    qNorm;             /* Normal channel quantities              */
  QTYS    qComp;             /* Complimentary channel  quantities      */
  PLTD    tNrmScop;          /* Normal channel voltage data            */
  PLTD    tCmpScop;          /* Complimentary channel voltage data     */
  char   *bTranEye;
  long    lTranRsv;
  char   *bDeemEye;
  long    lDeemRsv;
  } PCIM;
```

lCompPnt

```
Compliance Point, may be one of the following constants:
PCIX_RX_MODE – Receive Mode
PCIX_TX_MODE – Transmit Mode
PCIX_RX_CARD – Receive Add-In Card Mode
PCIX_TX_CARD – Transmit Add-In Card Mode
PCIX_RX_SYST – Receive System Card Mode
PCIX_TX_SYST – Transmit System Card Mode
```

Default: PCIX_RX_MODE

lPcnt This field specifies the voltage thresholds to be used when calculating rise and fall times. The voltage thresholds are assumed to be symmetrical about the 50% threshold, and this is the distance from the 50% threshold to the starting and ending thresholds. For example if this field is equal to 30, then 20% and 80% thresholds are used. If this field is equal to 40, then 10% and 90% thresholds are used. The absolute voltage levels used are based on the previous pulsefind minimum and maximum voltages. If this field is negative, then the absolute rise and fall thresholds are taken from the following fields lHiRFmV and lLoRFmv.
Default: 30

lHiRFmV Absolute rise/fall voltage if lPcnt<0, in units of mV
Default: +250

lLoRFmV Absolute rise/fall voltage if lPcnt<0, in units of mV
Default: -250

lIdleOk This flag is set by the system when an Idle Mode measurement is successfully made. The results are then applied in subsequent measurements. Set this flag to zero to invalidate the previous Idle Mode measurement results, and force a new Idle measurement to be made using the command :PCIM:IDLE? Before the common mode idle voltages are applied once again.
Default: 0

dAttn Attenuation factor in dB, this is provided to allow the results to be scaled to compensate for external attenuation from sources such as probes.
Default: 0

tRcpm Datacom With Bitclock and Marker Tool which specifies most of the input and output parameters necessary for a data signal analysis. The user will need to review all of the default parameters of the Datacom With Bitclock and Marker Tool and decide which to change.

lGood Flag indicates valid data in structure

dEyeOffs, dXmnDiff, dXmxDiff Used internally, DO NOT ALTER!

dVdiffPP Pk-pk differential voltage

dVdRatio De-emphaisis voltage ratio

dOpnEyeT Eye opening at Bit Error rate 10e-12

dMedEyeT Median to max jitter based on 1 million samples

dOpnEyeT1M Eye opening at Bit Error rate 10e-6

dTranVolts Vpp for Transition Eye

dDeemVolts Vpp for De-Emphasis Eye

dVcommonAc V?x-cm-acp

dVcommonDc V?x-cm-dc

dVcmDcActv V?x-cm-dc-active-idle-delta

dVcmIdleDc V?x-cm-idle-dc

dVcmDcLine V?x-cm-dc-line-delta

dVcmDcDpls V?x-cm-dc-d+

dVcmDcDmin V?x-cm-dc-d-

dVIdleDiff V?x-idle-diffp

qNorm Normal channel quantities

qComp Complimentary channel quantities

tNrmScop Normal channel voltage data

tCmpScop Complimentary channel voltage data

bTranEye,lTranRsv, bDeemEye,lDeemRsv Used internally, DO NOT ALTER!

void __stdcall FCNL_DefPcim (PCIM *pcim)

This function is used to fill the pcim structure for the PCI Express Compliance tool with reasonable default values. It is recommended that this function be called initially even if parameters within the structure are to be adjusted manually, and may be called repeatedly to reestablish initial conditions; however, this will impact test time.

Before calling this function, zero out the PCIM structure using the standard memset() function to ensure that any information pertaining to dynamic memory allocation is cleaned out prior to using the structure.

INPUTS

pcim - Pointer to a PCIM structure. Memory needs to be allocated by the caller.

OUTPUTS

None.

void __stdcall FCNL_ClrPcim (PCIM *pcim)

This function frees any dynamic memory that may have been allocated during previous data acquisitions and clears out the pcim structure.

INPUTS

pcim - Pointer to a PCIM structure. Memory needs to be allocated by the caller.

OUTPUTS

None.

EXAMPLE

```
static PCIM pcim;                                //declare pcim to a structure of type
                                                 //PCIM
memset ( &pcim, 0, sizeof ( PCIM ) );            //clear the memory for pci structure
FCNL_DefPcim ( &pcim);                           //set pci structures to default values

FCNL_RqstPkt ( ApiDevId, &pcim, WIND_PCIM );     //execute the measurement.
FCNL_RqstAll ( ApiDevId, &pcim, WIND_PCIM );     //get plot data (includes tRcpm)

FCNL_ClrPcim ( &pcim);                           //deallocate the structure
```

2-33 PCI EXPRESS 1.1 WITH SOFTWARE CLOCK RECOVERY TOOL

The PCI Express 1.1 with Software Clock Recovery Tool provides both timing and amplitude compliance measurements using the SIA3000. This tool accurately determines device performance by quantifying both random and deterministic jitter components.

```
typedef struct
  {
  /* Input parameters  */
  long    lCompPnt;          /* Compliance Point 0-RX 1-TX             */
  long    lPcnt;             /* Amount +/- 50% to calc. rise/fall time */
  long    lHiRFmV;           /* Absolute rise/fall voltage if lPcnt<0  */
  long    lLoRFmV;           /* Absolute rise/fall voltage if lPcnt<0  */
  long    lIdleOk;           /* Common mode idle voltages are valid    */
  long    lPass;             /* Acquisitions so far, set to 0 to reset */
  double  dAttn;             /* Attenuation factor (dB)                */
  KPWM    tKpwm;             /* Contains acquisition parameters        */
  /* Output parameters */
  long    lGood;             /* Flag indicates valid data in structure */
  long    lTtlHits;
  double  dEyeOffs;
  double  dHistMed;
  double  dXmnDiff;
  double  dXmxDiff;
  double  dVdiffPP;          /* Pk-pk differential voltage             */
  double  dVdRatio;          /* De-emphaisis voltage ratio             */
  double  dOpnEyeT;          /* Eye opening                            */
  double  dMedEyeT;          /* Median to max jitter                   */
  double  dOpnEyeT1M;        /* Eye opening @ 10^-6 BER                */
  double  dTranVolts;        /* Vpp for Transition Eye                 */
  double  dDeemVolts;        /* Vpp for De-Emphasis Eye                */

  double  dVcommonAc;        /* V?x-cm-acp                             */
  double  dVcommonDc;        /* V?x-cm-dc                              */
  double  dVcmDcActv;        /* V?x-cm-dc-active-idle-delta            */
  double  dVcmIdleDc;        /* V?x-cm-idle-dc                         */
  double  dVcmDcLine;        /* V?x-cm-dc-line-delta                   */
  double  dVcmDcDpls;        /* V?x-cm-dc-d+                           */
  double  dVcmDcDmin;        /* V?x-cm-dc-d-                           */
  double  dVIdleDiff;        /* V?x-idle-diffp                         */

  QTYS    qNorm;             /* Normal channel quantities              */
  QTYS    qComp;             /* Complimentary channel  quantities      */
  PLTD    tNrmScop;          /* Normal channel voltage data            */
  PLTD    tCmpScop;          /* Complimentary channel voltage data     */
  PLTD    tTtlHist;          /* Total Histogram of median-to-max data  */
  char   *bTranEye;
  long    lTranRsv;
  char   *bDeemEye;
  long    lDeemRsv;
  } EXPR;
```

lCompPnt Compliance Point, may be one of the following constants:

PCIX_RX_MODE - Receive Mode
PCIX_TX_MODE - Transmit Mode
PCIX_RX_CARD - Receive Add-In Card Mode
PCIX_TX_CARD - Transmit Add-In Card Mode
PCIX_RX_SYST - Receive System Card Mode
PCIX_TX_SYST - Transmit System Card Mode
Default: PCIX_RX_MODE

lPcnt This field specifies the voltage thresholds to be used when calculating rise and fall times. The voltage thresholds are assumed to be symmetrical about the 50% threshold, and this is the distance from the 50% threshold to the starting and ending thresholds. For example if this field is equal to 30, then 20% and 80% thresholds are used. If this field is equal to 40, then 10% and 90% thresholds are used. The absolute voltage levels used are based on the previous pulsefind minimum and maximum voltages. If this field is negative, then the absolute rise and fall thresholds are taken from the following fields lHiRFmV and lLoRFmv.
Default: 30

lHiRFmV Absolute rise/fall voltage if lPcnt<0, in units of mV
Default: +250

lLoRFmV Absolute rise/fall voltage if lPcnt<0, in units of mV
Default: -250

lIdleOk This flag is set by the system when an Idle Mode measurement is successfully made. The results are then applied in subsequent measurements. Set this flag to zero to invalidate the previous Idle Mode measurement results, and force a new Idle measurement to be made using the command :EXPR:IDLE? Before the common mode idle voltages are applied once again.
Default: 0

lPass This parameter is a bi-directional structure element that tracks the number of acquisitions since last reset. This flag can be read after an execution or set prior to an execution. Setting this parameter to 0 essentially resets this register. It will be automatically incremented when a measurement is performed.
Valid Entries: any integer greater than or equal to 0
Default: 0

dAttn Attenuation factor in dB, this is provided to allow the results to be scaled to compensate for external attenuation from sources such as probes.
Default: 0

tKpwm Known Pattern With Marker Tool which specifies most of the input and output parameters necessary for a data signal analysis. The user will need to review all of the default parameters of the Known Pattern With Tool and decide which to change.

lGood Flag indicates valid data in structure

lTtlHits Total hits collected in the Total Jitter Histogram

dHistMed Median location for the Total Jitter Histogram

dEyeOffs, dXmnDiff, dXmxDiff Used internally, DO NOT ALTER!

dVdiffPP Pk-pk differential voltage

dVdRatio De-emphaisis voltage ratio

dOpnEyeT Eye opening at Bit Error rate 10e-12

dMedEyeT Median to max jitter based on 1 million samples

dOpnEyeT1M Eye opening at Bit Error rate 10e-6

dTranVolts Vpp for Transition Eye

dDeemVolts Vpp for De-Emphasis Eye

dVcommonAc V?x-cm-acp

dVcommonDc V?x-cm-dc

dVcmDcActv V?x-cm-dc-active-idle-delta

dVcmIdleDc V?x-cm-idle-dc

dVcmDcLine V?x-cm-dc-line-delta

dVcmDcDpls V?x-cm-dc-d+

dVcmDcDmin V?x-cm-dc-d-
dVIdleDiff V?x-idle-diffp
qNorm Normal channel quantities
qComp Complimentary channel quantities
tNrmScop Normal channel voltage data
tCmpScop Complimentary channel voltage data
tTtlHist Total Jitter Histogram data
bTranEye,lTranRsv, bDeemEye,lDeemRsv Used internally, DO NOT ALTER!

void __stdcall FCNL_DefExpr (EXPR *expr)

This function is used to fill the expr structure for the PCI Express Compliance tool with reasonable default values. It is recommended that this function be called initially even if parameters within the structure are to be adjusted manually, and may be called repeatedly to reestablish initial conditions; however, this will impact test time.

Before calling this function, zero out the EXPR structure using the standard memset() function to ensure that any information pertaining to dynamic memory allocation is cleaned out prior to using the structure.

INPUTS

expr - Pointer to a EXPR structure. Memory needs to be allocated by the caller.

OUTPUTS

None.

void __stdcall FCNL_ClrExpr (EXPR *expr)

This function frees any dynamic memory that may have been allocated during previous data acquisitions and clears out the expr structure.

INPUTS

expr - Pointer to a EXPR structure. Memory needs to be allocated by the caller.

OUTPUTS

None.

EXAMPLE

```
static EXPR expr;                               //declare expr to a structure of type
                                                //EXPR
memset ( &expr, 0, sizeof ( EXPR ) );           //clear the memory for expr structure
FCNL_DefExpr ( &expr);                          //set expr structures to default values

FCNL_RqstPkt ( ApiDevId, &expr, WIND_EXPR );    //execute the measurement.
FCNL_RqstAll ( ApiDevId, &expr, WIND_EXPR );    //get plot data (includes tRcpm)

FCNL_ClrExpr ( &expr);                          //deallocate the structure
```

2-34 PCI EXPRESS 1.1 CLOCK ANALYSIS TOOL

The PCI Express 1.1 Clock Analysis Tool provides both timing and amplitude compliance measurements for PCI Express Reference Clocks using the SIA3000. This tool accurately determines device performance by quantifying both random and deterministic jitter components.

```
typedef struct
  {
  /* Input parameters  */
  long    lPcnt;             /* Amount +/- 50% to calc. rise/fall time */
  long    lHiRFmV;           /* Absolute rise/fall voltage if lPcnt<0  */
  long    lLoRFmV;           /* Absolute rise/fall voltage if lPcnt<0  */
  long    lPad0;
  double  dAttn;             /* Attenuation factor (dB)                */
  KPWM    tKpwm;             /* Contains acquisition parameters        */
  /* Output parameters */
  long    lGood;             /* Flag indicates valid data in structure */
  long    lPad1;
  double  dRiseRate;         /* Rising edge rate (V/ns)                */
  double  dFallRate;         /* Falling edge rate (V/ns)               */
  double  dDifMaxVin;        /* Differential Input High Voltage        */
  double  dDifMinVin;        /* Differential Input Low Voltage         */

  double  dPeriodPpm;        /* Average Clock Period Accuracy          */
  double  dPeriodMin;        /* Absolute Period Minimum                */
  double  dPeriodMax;        /* Absolute Period Maximum                */
  double  dCycl2Cycl;        /* Cycle to Cycle Jitter                  */
  double  dVmaxSingl;        /* Absolute Max input voltage             */
  double  dVminSingl;        /* Absolute Min input voltage             */
  double  dDutyCycle;        /* Duty Cycle                             */
  double  dRFMatches;        /* Rising Rate to Falling Rate Matching   */
  double  dMaxJitt1M;        /* Maximum Pk-Pk Jitter @ 10^-6 BER       */

  QTYS    qNorm;             /* Normal channel quantities              */
  QTYS    qComp;             /* Complimentary channel  quantities      */
  QTYS    qDiff;             /* Differential channel  quantities       */
  PLTD    tNrmScop;          /* Normal channel voltage data            */
  PLTD    tCmpScop;          /* Complimentary channel voltage data     */
  PLTD    tDifScop;          /* Differential channel voltage data      */
  } PCLK;
```

lPcnt This field specifies the voltage thresholds to be used when calculating rise and fall times. The voltage thresholds are assumed to be symmetrical about the 50% threshold, and this is the distance from the 50% threshold to the starting and ending thresholds. For example if this field is equal to 30, then 20% and 80% thresholds are used. If this field is equal to 40, then 10% and 90% thresholds are used. The absolute voltage levels used are based on the previous pulsefind minimum and maximum voltages. If this field is negative, then the absolute rise and fall thresholds are taken from the following fields lHiRFmV and lLoRFmv.
Default: 30

lHiRFmV Absolute rise/fall voltage if lPcnt<0, in units of mV
Default: +250

lLoRFmV Absolute rise/fall voltage if lPcnt<0, in units of mV
Default: -250

lPad0 Used internally, DO NOT ALTER!

dAttn	Attenuation factor in dB, this is provided to allow the results to be scaled to compensate for external attenuation from sources such as probes. Default: 0
tKpwm	Known Pattern With Marker Tool which specifies most of the input and output parameters necessary for a data signal analysis. The user will need to review all of the default parameters of the Known Pattern With Marker Tool and decide which to change.
lGood	Flag indicates valid data in structure
lPad1	Used internally, DO NOT ALTER!
dRiseRate	Rising edge rate (V/ns)
dFallRate	Falling edge rate (V/ns)
dDifMaxVin	Differential Input High Voltage
dDifMinVin	Differential Input Low Voltage
dPeriodPpm	Average Clock Period Accuracy expressed in Parts Per Million
dPeriodMin	Absolute Period Minimum in seconds
dPeriodMax	Absolute Period Maximum in seconds
dCycl2Cycl	Cycle-To-Cycle Jitter in seconds
dVmaxSingl	Absolute Max Single-Ended input voltage
dVminSingl	Absolute Min Single-Ended input voltage
dDutyCycle	Duty Cycle expressed as a percentage
dRFMatches	Rising Rate to Falling Rate Matching expressed as a Percentage
dMaxJitt1M	Maximum Pk-Pk Jitter @ 10^-6 BER
qNorm	Normal channel quantities
qComp	Complimentary channel quantities
qDiff	Differential (IN - /IN) channel quantities
tNrmScop	Normal channel voltage data
tCmpScop	Complimentary channel voltage data
tDifScop	Differential (IN - /IN) channel voltage data

void __stdcall FCNL_DefPclk (PCLK *pclk)

This function is used to fill the pclk structure for the PCI Express Compliance tool with reasonable default values. It is recommended that this function be called initially even if parameters within the structure are to be adjusted manually, and may be called repeatedly to reestablish initial conditions; however, this will impact test time.

Before calling this function, zero out the PCLK structure using the standard memset() function to ensure that any information pertaining to dynamic memory allocation is cleaned out prior to using the structure.

INPUTS

pclk - Pointer to a PCLK structure. Memory needs to be allocated by the caller.

OUTPUTS

None.

void __stdcall FCNL_ClrPclk (PCLK *pclk)

This function frees any dynamic memory that may have been allocated during previous data acquisitions and clears out the pclk structure.

INPUTS

pclk - Pointer to a PCLK structure. Memory needs to be allocated by the caller.

OUTPUTS

None.

EXAMPLE

```
static PCLK pclk;                                 //declare pclk to a structure of type
                                                  //PCLK
memset ( &pclk, 0, sizeof ( PCLK ) );             //clear the memory for pclk structure
FCNL_DefPclk ( &pclk);                            //set pclk structures to default values

FCNL_RqstPkt ( ApiDevId, &pclk, WIND_PCLK );      //execute the measurement.
FCNL_RqstAll ( ApiDevId, &pclk, WIND_PCLK );      //get plot data (includes tRcpm)

FCNL_ClrPclk ( &pclk);                            //deallocate the structure
```

2-35 PCI EXPRESS 1.0a TOOL

The PCI Express 1.0a Tool provides both timing and amplitude compliance measurements in any environment, system or IC, electrical or optical. Compliance tests can be completed in seconds with a simple pass/fail indication for each test parameter. It is the most comprehensive and easy to use signal integrity test solution on the market today.

The PCI Express 1.0a Tool accurately determines device performance by quantifying random and deterministic jitter components. In addition, the PCI Express 1.0a Tool can quickly isolate and quantify unwanted deterministic jitter due to crosstalk and EMI with a spectral view of jitter as well as perform Eye Diagram analysis for a quick qualitative view of device performance.

```
typedef struct
  {
  /* Input parameters  */
  long    lCompPnt;           /* Compliance Point 0-RX 1-TX            */
  long    lPcnt;              /* Amount +/- 50% to calc. rise/fall time */
  long    lHiRFmV;            /* Absolute rise/fall voltage if lPcnt<0  */
  long    lLoRFmV;            /* Absolute rise/fall voltage if lPcnt<0  */
  long    lIdleOk;            /* Common mode idle voltages are valid    */
  long    lPad0;
  double  dAttn;              /* Attenuation factor (dB)                */
  RCPM    tRcpm;              /* Contains acquisition parameters        */
  /* Output parameters */
  long    lGood;              /* Flag indicates valid data in structure */
  long    lPad1;
  double  dEyeOffs;
  double  dXmnDiff;
  double  dXmxDiff;
  double  dVdiffPP;           /* Pk-pk differential voltage             */
  double  dVdRatio;           /* De-emphaisis voltage ratio             */
  double  dOpnEyeT;           /* Eye opening                            */
  double  dMedEyeT;           /* Median to max jitter                   */

  double  dVcommonAc;         /* V?x-cm-acp                             */
  double  dVcommonDc;         /* V?x-cm-dc                              */
  double  dVcmDcActv;         /* V?x-cm-dc-active-idle-delta           */
  double  dVcmIdleDc;         /* V?x-cm-idle-dc                         */
  double  dVcmDcLine;         /* V?x-cm-dc-line-delta                   */
  double  dVcmDcDpls;         /* V?x-cm-dc-d+                           */
  double  dVcmDcDmin;         /* V?x-cm-dc-d-                           */
  double  dVIdleDiff;         /* V?x-idle-diffp                         */

  QTYS    qNorm;              /* Normal channel quantities              */
  QTYS    qComp;              /* Complimentary channel  quantities      */
  PLTD    tNrmScop;           /* Normal channel voltage data            */
  PLTD    tCmpScop;           /* Complimentary channel voltage data     */
  char   *bTranEye;
  long    lTranRsv;
  char   *bDeemEye;
  long    lDeemRsv;
  } PCIX;
```

lCompPnt Compliance Point, may be one of the following constants:
PCIX_RX_MODE - Receive Mode
PCIX_TX_MODE - Transmit Mode
PCIX_RX_CARD - Receive Add-In Card Mode
PCIX_TX_CARD - Transmit Add-In Card Mode

PCIX_RX_SYST - Receive System Card Mode
PCIX_TX_SYST - Transmit System Card Mode
Default: PCIX_RX_MODE

lPcnt This field specifies the voltage thresholds to be used when calculating rise and fall times. The voltage thresholds are assumed to be symmetrical about the 50% threshold, and this is the distance from the 50% threshold to the starting and ending thresholds. For example if this field is equal to 30, then 20% and 80% thresholds are used. If this field is equal to 40, then 10% and 90% thresholds are used. The absolute voltage levels used are based on the previous pulsefind minimum and maximum voltages. If this field is negative, then the absolute rise and fall thresholds are taken from the following fields lHiRFmV and lLoRFmv.
Default: 30

lHiRFmV Absolute rise/fall voltage if lPcnt<0, in units of mV
Default: +250

lLoRFmV Absolute rise/fall voltage if lPcnt<0, in units of mV
Default: -250

lIdleOk This flag is set by the system when an Idle Mode measurement is successfully made. The results are then applied in subsequent measurements. Set this flag to zero to invalidate the previous Idle Mode measurement results, and force a new Idle measurement to be made using the command :PCIX:IDLE? Before the common mode idle voltages are applied once again.
Default: 0

dAttn[n] Attenuation factor in dB, this is provided to allow the results to be scaled to compensate for external attenuation from sources such as probes.
Default: 0

tRcpm Datacom With Bitclock and Marker Tool which specifies most of the input and output parameters necessary for a data signal analysis. The user will need to review all of the default parameters of the Datacom With Bitclock and Marker Tool and decide which to change.

lGood Flag indicates valid data in structure

dEyeOffs, dXmnDiff, dXmxDiff Used internally, DO NOT ALTER!

dVdiffPP Pk-pk differential voltage

dVdRatio De-emphaisis voltage ratio

dOpnEyeT Eye opening

dMedEyeT Median to max jitter

dVcommonAc V?x-cm-acp

dVcommonDc V?x-cm-dc

dVcmDcActv V?x-cm-dc-active-idle-delta

dVcmIdleDc V?x-cm-idle-dc

dVcmDcLine V?x-cm-dc-line-delta

dVcmDcDpls V?x-cm-dc-d+

dVcmDcDmin V?x-cm-dc-d-

dVIdleDiff V?x-idle-diffp

qNorm Normal channel quantities

qComp Complimentary channel quantities

tNrmScop Normal channel voltage data

tCmpScop Complimentary channel voltage data

bTranEye,lTranRsv, bDeemEye,lDeemRsv Used internally, DO NOT ALTER!

void __stdcall FCNL_DefPcix (PCIX *pcix)

This function is used to fill the pcix structure for the PCI Express Compliance tool with reasonable default values. It is recommended that this function be called initially even if parameters within the structure are to be adjusted manually, and may be called repeatedly to reestablish initial conditions; however, this will impact test time.

Before calling this function, zero out the PCIX structure using the standard memset() function to ensure that any information pertaining to dynamic memory allocation is cleaned out prior to using the structure.

INPUTS

pcix - Pointer to a PCIX structure. Memory needs to be allocated by the caller.

OUTPUTS

None.

void __stdcall FCNL_ClrPcix (PCIX *pcix)

This function frees any dynamic memory that may have been allocated during previous data acquisitions and clears out the pcix structure.

INPUTS

pcix - Pointer to a PCIX structure. Memory needs to be allocated by the caller.

OUTPUTS

None.

EXAMPLE

```
static PCIX pci;                                 //declare pci to a structure of type
                                                 //PCIX
memset ( &pci, 0, sizeof ( PCIX ) );             //clear the memory for pci structure
FCNL_DefPcix ( &pci);                            //set pci structures to default values

FCNL_RqstPkt ( ApiDevId, &pci, WIND_PCIX );      //execute the measurement.
FCNL_RqstAll ( ApiDevId, &pci, WIND_PCIX );      //get plot data (includes tRcpm)

FCNL_ClrPcix ( &pci);                            //deallocate the structure
```

2-36 PHASE NOISE TOOL

The Phase Noise tool allows users to measure phase noise in clock/oscillator sources. By simply choosing the highest frequency to be displayed and the frequency resolution, the tool will measure and display the phase noise spectrum. This tool reports the phase noise values at common offset frequencies.

The Phase Noise tool is used to show the amplitude and frequency of phase noise relative to the carrier signal frequency. This tool measures the fluctuations in the phase of a signal caused by time domain instabilities. Fast and easy phase noise measurements of oscillators and PLL devices can be easily correlated to other noise effects on the signal.

The sensitivity of the tool is limited by hardware and is dependent on f0 and Maximum Freq. Alternate methods of characterizing random noise in clock sources are available in the SIA-3000.

```
typedef struct
  {
  /* Input parameters  */
  PARM    tParm;              /* Contains acquisition parameters        */
  FFTS    tFfts;              /* FFT window and analysis parameters     */
  long    lAutoFix;           /* If true calculate the above parameters */
  long    lPad1;
  double  dMaxFreq;           /* Maximum Frequency that is desired      */
  double  dFreqRes;           /* Frequency resolution that is desired   */
  long    lFftAvgs;           /* 2^fft_avgs averages used to smooth FFT */
  /* Output parameters */
  long    lGood;              /* Flag indicates valid data in structure */
  PLTD    tTime;              /* Time domain plot data                  */
  PLTD    tStmp;              /* Time stamp array, not normally plotted */
  PLTD    tFft1;              /* Frequency plot data on 1-clock basis   */
  PLTD    tPhas;              /* Phase noise plot in dBc/Hz             */
  double  dCarFreq;           /* Carrier frequency                      */
  double  dSmpRate;           /* Sampling rate                          */
  double  dValByDec[DECADES]; /* Phase Noise by Decade, first is 10Hz   */
                              /* last is fMax, zero means illegal value */
  } PHAS;
```

tParm A structure of type PARM that contains acquisition parameter. The PARM structure is discussed in full detail in Section 2-4.

tFfts A structure of type FFTS that contains the FFT setup parameters such as window type and padding factor. See Section 2-10 for further details.

lAutoFix If true calculate some of the above tParm parameters
Default: 0

dMaxFreq Maximum Frequency that is desired
Default: 1000.0

dFreqRes Frequency resolution that is desired
Default: 1.0

lFftAvgs 2^fft_avgs averages used to smooth FFT
Default: 2

lGood Flag indicates valid data in structure

tTime Time domain plot data

tStmp Time stamp array, not normally plotted

tFft1 Frequency plot data on 1-clock basis

tPhas Phase noise plot in dBc/Hz

dCarFreq `Carrier frequency`
dSmpRate `Sampling rate`
dValByDec[n] `Phase Noise by Decade, first is 10Hz last is fMax, zero means illegal value`

void __stdcall FCNL_DefPhas (PHAS *phas)

This function is used to fill the phas structure for the Phase Noise tool with reasonable default values. It is recommended that this function be called initially even if parameters within the structure are to be adjusted manually, and may be called repeatedly to reestablish initial conditions; however, this will impact test time.

Before calling this function, zero out the PHAS structure using the standard memset() function to ensure that any information pertaining to dynamic memory allocation is cleaned out prior to using the structure.

INPUTS

phas - Pointer to a PHAS structure. Memory needs to be allocated by the caller.

OUTPUTS

None.

void __stdcall FCNL_ClrPhas (PHAS *phas)

This function frees any dynamic memory that may have been allocated during previous data acquisitions and clears out the phas structure.

INPUTS

phas - Pointer to a PHAS structure. Memory needs to be allocated by the caller.

OUTPUTS

None.

EXAMPLE

```
static PHAS phase;                                  //declare phase to a structure of type
                                                    //PHAS
memset ( &phase, 0, sizeof ( PHAS ) );              //clear the memory for phase structure
FCNL_DefPhas ( &phase );                            //set phase structures to default values

FCNL_RqstPkt ( ApiDevId, &phase, WIND_PHAS );       //execute the measurement.
FCNL_RqstAll ( ApiDevId, &phase, WIND_PHAS );       //get plot data

FCNL_ClrPhas ( &phase );                            //deallocate the structure
```

2-37 PLL ANALYSIS TOOL

The PLL Analysis tool permits users to study characteristics and parameters of a 2nd-order PLL. With a simple set of variance measurements, the tool can extract information such as damping factor, natural frequency, input noise level, lock range, lock-in time, pull-in time, pull-out range and noise bandwidth. The tool also presents a transfer function and Bode plots up to the natural frequency, as well as a plot of the poles and zero for a 2nd-order PLL.

```
typedef struct
  {
  /* Input parameters  */
  PARM    tParm;              /* Contains acquisition parameters        */
  double  dXiGuess;           /* Initial value for damping factor       */
  double  dWnGuess;           /* Initial value for natural frequency    */
  double  dS0Guess;           /* Initial power spectral density dBc/Hz  */
  double  dInitOff;           /* Initial offset frequency - delta W0    */
  long    lIncStop;           /* Increase stop count by this value      */
  long    lMaxStop;           /* Maximum stop count to collect data     */
  double  dCornFrq;           /* Corner Frequency for Record Length     */
  double  dRecTime;           /* Record Length in units of time (s)     */
  long    lRecUnit;           /* Record length units, must be one of:   */
                              /* 0=lMaxStop, 1=dCornFreq, 2=dRecTime    */
  long    lIniCond;           /* Calc. initial conditions if non-zero   */
  /* Output parameters */
  long    lGood;              /* Flag indicates valid data in structure */
  long    lVfit;              /* Indicates if the variance fit was good */
  double  dDampFct;           /* Damping factor from variance fit       */
  double  dNatFreq;           /* Natural frequency from fit (rad/s)     */
  double  dS0Noise;           /* Noise process power spectral density   */
  double  dChSquar;           /* Chi-square of variance fit             */
  double  dFreq;              /* Carrier frequency                      */
  complex dPole[2], dZero;    /* Poles and zero                         */
  double  dLockRng;           /* Lock Range (rad/s)                     */
  double  dLockInT;           /* Lock-in Time (s)                       */
  double  dPullInT;           /* Pull-in Time (s)                       */
  double  dPullOut;           /* Pull-out Range (rad/s)                 */
  double  dNoiseBW;           /* Noise Bandwidth (rad/s)                */
  PLTD    tSigm;              /* Contains the 1-Sigma plot array        */
  PLTD    tVfit;              /* Resulting variance fit plot array      */
  PLTD    tInit;              /* Initial Conditions variance plot array */
  PLTD    tXfer;              /* PLL Transfer Function plot array       */
  PLTD    tBodeMag;           /* Bode plot magnitude/gain response      */
  PLTD    tBodePha;           /* Bode plot phase response               */
  } APLL;
```

tParm A structure of type PARM that contains acquisition parameter. The PARM structure is discussed in full detail in Section 2-4.

dXiGuess Initial value for damping factor
Default: 0.25

dWnGuess Initial value for natural frequency
Default: 315e3

dS0Guess Initial power spectral density dBc/Hz
Default: -90.0

dInitOff Initial offset frequency - delta W0
Default: 1000.0

lIncStop Increase stop count by this value
Default: 1

lMaxStop	`Maximum stop count to collect data` Default: 1000
dCornFrq	`Corner Frequency for Record Length` Default: 50e3
dRecTime	`Record Length in units of time (s)` Default: 10e-6

lRecUnit	`Record length units, must be one of:`
	`0=lMaxStop, 1=dCornFreq, 2=dRecTime`
	Default: 2

lIniCond	`Calc. initial conditions if non-zero` Default: 1
lGood	`Flag indicates valid data in structure`
lVfit	`Indicates if the variance fit was good`
dDampFct	`Damping factor from variance fit`
dNatFreq	`Natural frequency from fit (rad/s)`
dS0Noise	`Noise process power spectral density`
dChSquar	`Chi-square of variance fit`
dFreq	`Carrier frequency`
dPole[2]	`Location of Poles of transfer function`
dZero	`Location of zero of transfer function`
dLockRng	`Lock Range (rad/s)`
dLockInT	`Lock-in Time (s)`
dPullInT	`Pull-in Time (s)`
dPullOut	`Pull-out Range (rad/s)`
dNoiseBW	`Noise Bandwidth (rad/s)`
tSigm	`Contains the 1-Sigma plot array`
tVfit	`Resulting variance fit plot array`
tInit	`Initial Conditions variance plot array`
tXfer	`PLL Transfer Function plot array`
tBodeMag	`Bode plot magnitude/gain response`
tBodePha	`Bode plot phase response`

`void __stdcall FCNL_DefApll ( APLL *apll )`

This function is used to fill the apll structure for the PLL Analysis tool with reasonable default values. It is recommended that this function be called initially even if parameters within the structure are to be adjusted manually, and may be called repeatedly to reestablish initial conditions; however, this will impact test time.

Before calling this function, zero out the APLL structure using the standard memset() function to ensure that any information pertaining to dynamic memory allocation is cleaned out prior to using the structure.

INPUTS

apll - Pointer to a APLL structure. Memory needs to be allocated by the caller.

OUTPUTS

None.

void __stdcall FCNL_ClrApll (APLL *apll)

This function frees any dynamic memory that may have been allocated during previous data acquisitions and clears out the apll structure.

INPUTS

apll - Pointer to a APLL structure. Memory needs to be allocated by the caller.

OUTPUTS

None.

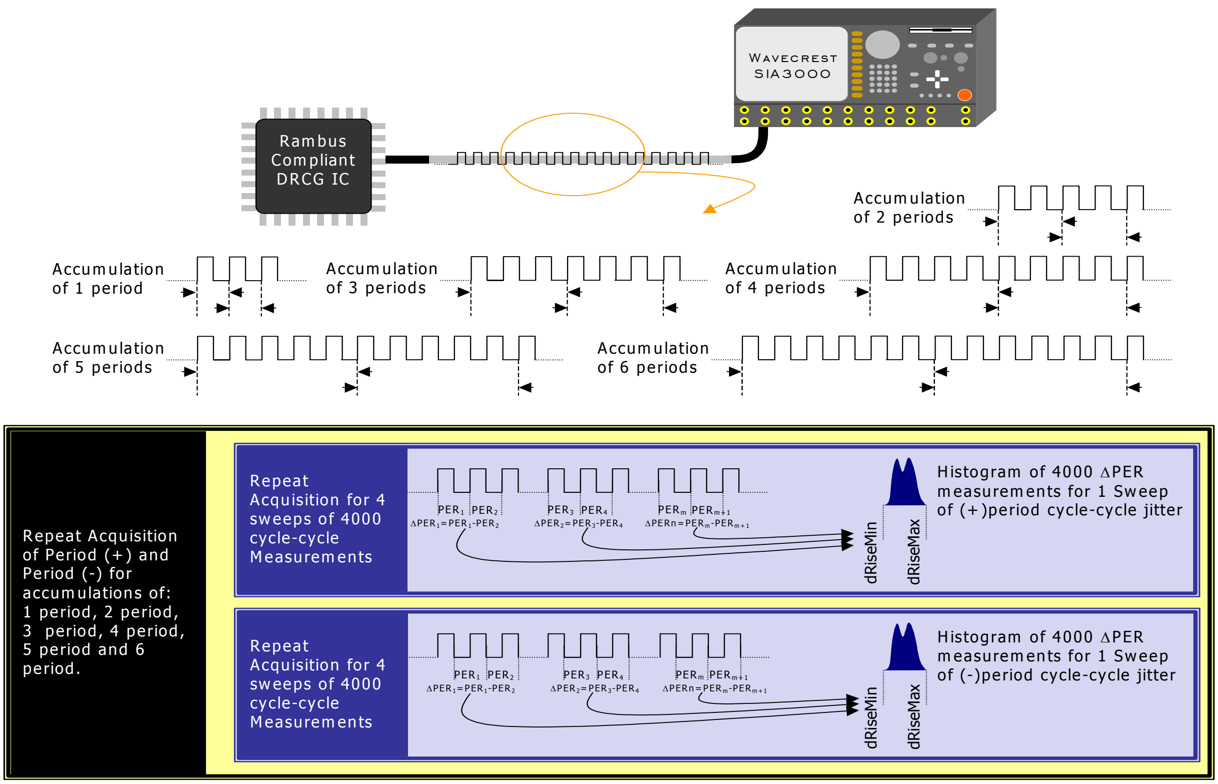
EXAMPLE

```
static APLL pll;                                 //declare pll to a structure of type
                                                 //APLL
memset ( &pll, 0, sizeof ( APLL ) );             //clear the memory for pll structure
FCNL_DefApll ( &pll);                            //set pll structures to default values

FCNL_RqstPkt ( ApiDevId, &pll, WIND_APLL );      //execute the measurement.
FCNL_RqstAll ( ApiDevId, &pll, WIND_APLL );      //get plot data

FCNL_ClrApll ( &pll);                            //deallocate the structure
```

2-38 RAMBUS DRCG TOOL

The Rambus DRCG tool was developed specifically to test Rambus® clock generator chips which have a compliance test that includes adjacent cycle jitter at 6 incremental accumulations for both period polarities. This tool is a true compliance tool such that the specification, as defined by Rambus Corporation, has been incorporated into this tool to validate a DRCG's performance relative to the standard.

The measurement consists of accumulated adjacent cycle jitter measurements (cycle to cycle) for both rising edges and falling edges. The measurement algorithm is depicted above. Each measurement configuration is executed in 4 "sweeps". Each sweep is a burst of 4000 measurements. For a given execution, 4 sweeps of 4000 measurements for both rising and falling edges at 6 different amplitudes of accumulation results in 4x4000x2x6=192,000 measurements. The results are placed in arrays, which are organized by cumulative cycles and sweep number.



DRCG Utility's measurement algorithm

```
typedef struct
  {
  /* Input parameters  */
  PARM    tParm;              /* Contains acquisition parameters        */
  long    lAutoFix;           /* If true perform a pulsefind as req'd   */
  long    lDutCycl;           /* If non-zero make duty cycle measurement*/
  long    lUsrSpec;           /* If non-zero use the specified TJ value */
  long    lPad1;
  double  dSpecVal;           /* User-defined TJ specification          */
  /* Output parameters */
  long    lGood;              /* Flag indicates valid data in structure */
  long    lPass;
  double  dDutyMax;           /* Maximum value of duty cycle measurement*/
  double  dDutyMin;           /* Minimum value of duty cycle measurement*/
```

```
double  dDutyAvg;           /* Average value of duty cycle measurement*/
PLTD    tRiseMax;           /* Minimum deltaT of rising adj. periods  */
PLTD    tRiseMin;           /* Maximum deltaT of rising adj. periods  */
PLTD    tFallMax;           /* Minimum deltaT of falling adj. periods */
PLTD    tFallMin;           /* Maximum deltaT of falling adj. periods */
PLTD    tMaxiLim;           /* Maximum limit per specification        */
PLTD    tMiniLim;           /* Minimum limit per specification        */
double  dRiseMax[DRCG_CYCLES][DRCG_SWEEPS];
double  dRiseMin[DRCG_CYCLES][DRCG_SWEEPS];
double  dFallMax[DRCG_CYCLES][DRCG_SWEEPS];
double  dFallMin[DRCG_CYCLES][DRCG_SWEEPS];
double  dFreq;              /* Carrier frequency                      */
} DRCG;
```

tParm A structure of type PARM that contains acquisition parameter. The PARM is discussed in full detail in Section 2-4.

lAutoFix Flag indicating whether to perform a pulse-find as required. Setting this value to any integer greater than zero tells the measurement to perform a pulse find if needed. The system will know if a measurement was recently performed and if a pulse find is necessary.
Valid Entries: 0 – No pulsefind prior to measurement
1 – Pulsefind if the measurement mode changed.
Default: 0

lDutCycl Flag indicating whether to perform a duty cycle measurement. Measuring three successive transitions, this measurement represents the absolute duty cycle and allows the user to identify the maximum, minimum and average duty cycle.
Valid Entries: 0 – do not perform a duty cycle measurement
1 – perform a duty cycle measurement
Default: 0

lUsrSpec Flag to indicate whether to use a user specified limit for maximum/minimum cycle to cycle jitter or to use the Rambus defined specification. If this flag is set, the parameter specified in dSpecVal will be used as the pass/fail limit for this test.
Valid Entries: 0 – Use Rambus defined specification
1 – Use limit defined in dSpecVal
Default: 0

dSpecVal Test limit used by this tool, depending on the state of lUsrSpec, indicate a pass/fail condition based on the measured cycle to cycle jitter for each pass, polarity and accumulation.

lGood Flag used to indicate valid output data in structure.

dDutyMax, dDutyMin, dDutyAvg Maximum, minimum and average values of duty cycle measurement.

tRiseMax Structure of type PLTD containing all of the necessary information to draw a histogram of data containing the maximum increase in period of adjacent positive periods (periods characterized by a rising edges). See Section 2-3 for details of the PLTD structure and its elements.

tRiseMin Structure of type PLTD containing all of the necessary information to draw a histogram of data containing the maximum decrease in period of adjacent positive periods. See Section 2-3 for details of the PLTD structure and its elements.

tFallMax Structure of type PLTD containing all of the necessary information to draw a histogram of data containing the maximum increase in period of adjacent negative periods (periods characterized by a falling edges). See Section 2-3 for details of the PLTD structure and its elements.

tFallMin Structure of type PLTD containing all of the necessary information to draw a histogram of data containing the minimum deltaT of falling adjacent periods. See Section 2-3 for details of the PLTD structure and its elements.

tMaxiLim Structure of type PLTD containing all of the necessary information to draw a histogram of maximum limits per specification. See Section 2-3 for details of the PLTD structure and its elements.

tMiniLim Structure of type PLTD containing all of the necessary information to draw a histogram of minimum limits per specification. See Section 2-3 for details of the PLTD structure and its elements.

dRiseMax[m][n] A 6x4 array of maximum period increase between two adjacent positive periods organized by the number of accumulated periods and the sweep number. Each execution of this structure results in 6 accumulations and 4 sweeps. (Each sweep is a burst of 4000 measurements.)

dRiseMin[m][n] A 6x4 array of maximum period decrease between two adjacent positive periods organized by the number of accumulated periods and the sweep number. Each execution of this structure results in 6 accumulations and 4 sweeps. (Each sweep is a burst of 4000 measurements.)

dFallMax[m][n] A 6x4 array of maximum period increase between two adjacent negative periods organized by the number of accumulated periods and the sweep number. Each execution of this structure results in 6 accumulations and 4 sweeps. (Each sweep is a burst of 4000 measurements.)

dFallMin[m][n] A 6x4 array of maximum period decrease between two adjacent negative periods organized by the number of accumulated periods and the sweep number. Each execution of this structure results in 6 accumulations and 4 sweeps. (Each sweep is a burst of 4000 measurements.)

dFreq Measured carrier frequency.

void __stdcall FCNL_DefDrcg (DRCG *drcg)

This function is used to fill the drcg structure for the Rambus DRCG tool with reasonable default values. It is recommended that this function be called initially even if parameters within the structure are to be adjusted manually, and may be called repeatedly to reestablish initial conditions; however, this will impact test time.

Before calling this function, zero out the DRCG structure using the standard memset() function to ensure that any information pertaining to dynamic memory allocation is cleaned out prior to using the structure.

INPUTS

drcg - Pointer to a DRCG structure. Memory needs to be allocated by the caller.

OUTPUTS

None.

void __stdcall FCNL_ClrDrcg (DRCG *drcg)

This function frees any dynamic memory that may have been allocated during previous data acquisitions and clears out the `drcg` structure.

INPUTS

drcg - Pointer to a DRCG structure. Memory needs to be allocated by the caller.

OUTPUTS

None.

EXAMPLE

```
#define TRUE 1
#define FALSE 0
#define ACCUM_MAX 6
#define SWEEP_MAX 4


int i,j;
static DRCG rambus;                                 //declare cyc2cyc to be a structure of
                                                    //type ACYC
memset ( &rambus, 0, sizeof ( DRCG ) );             //clear the memory for cyc2cyc
FCNL_DefDrcg (&rambus);                             //set histogram structures to default
                                                    //values
rambus.tparm.lChanNum = 1;                          //capture waveform on channel 1
rambus.lDutCycl = TRUE;                             /Measure true duty cycle my measuring
                                                    //successive edges.

FCNL_RqstPkt ( ApiDevId, &rambus, WIND_DRCG );      //execute the measurement.
FCNL_RqstAll ( ApiDevId, & rambus, WIND_DRCG );     //get plot data

printf("MAX PERIOD DECREASE:  NEGATIVE PERIODS\n"); //Display results for all sweeps and cycles
printf("\tSweep1\tSweep2\tSweep3\tSweep4\n");
for(i=1;i<=ACCUM_MAX;i++)
  {
  printf("%i PER CYC-CYC\t",i);
  for(j=0;j<SWEEP_MAX;j++)
    printf("\t%d",ABS(rambus.dFallMin[i][j]));
  printf("\n"));
  }
printf("MAX PERIOD INCREASE:  NEGATIVE PERIODS\n");
printf("\tSweep1\tSweep2\tSweep3\tSweep4\n");
for(i=1;i<=ACCUM_MAX;i++)
  {
  printf("%i PER CYC-CYC\t",i);
  for(j=1;j<=SWEEP_MAX;j++)
    printf("\t%d",ABS(rambus.dFallMax[i][j]));
  printf("\n"));
  }

FCNL_ClrDrcg (&rambus);                             //deallocate the structure
```

2-39 SCOPE TOOL

The Oscilloscope tool provides a quick and easy display of the signal to be analyzed. The Oscilloscope has many different capabilities. It can capture a waveform, measure voltage parameters, and create eye masks.

```
typedef struct
  {
  /* Input parameters  */
  PARM    tParm;              /* Contains acquisition parameters       */
  long    lVoff[POSS_CHNS];   /* Voltage offset (mV)     - per channel  */
  long    lVdif[POSS_CHNS];   /* Differential offset (mV)- per channel  */
  long    lVcom[POSS_CHNS];   /* Common offset (mV)      - per channel  */
  long    lTper;              /* Time per division (ps)  - all channels */
  long    lTdel;              /* Delay time (ps)         - all channels */
  long    lPcnt;              /* Amount +/- 50% to calc. rise/fall time */
  long    lHiRFmV;            /* Absolute rise/fall voltage if lPcnt<0  */
  long    lLoRFmV;            /* Absolute rise/fall voltage if lPcnt<0  */
  long    lInps;              /* Input selection, see defines above     */
  long    lMeas;              /* Measure flag, see defines above        */
  long    lPass;              /* Acquisitions so far, set to 0 to reset */
  long    lAvgs;              /* 2^lAvgs = averages used to smooth data */
  long    lPad1;
  MASK    tMask;              /* Structure which holds mask definition  */
  double  dMargin;            /* Margin in percentage [-1.0 to 1.0]     */
  double  dHistDly;           /* Histogram horizontal location, seconds */
  double  dHistWid;           /* Histogram horizontal width, seconds    */
  double  dHistVlt;           /* Histogram vertical location, volts     */
  double  dHistHgt;           /* Histogram vertical height, volts       */
  double  dAttn[POSS_CHNS];   /* Attenuation factor (dB) - per channel  */
  /* Output parameters */
  long    lGood;              /* Flag indicates valid data in structure */
  long    lPad2;
  QTYS    qNorm[ POSS_CHNS ]; /* Normal channel quantities              */
  QTYS    qComp[ POSS_CHNS ]; /* Complimentary channel  quantities      */
  QTYS    qDiff[ POSS_CHNS ]; /* Differential quantities                */
  QTYS    qComm[ POSS_CHNS ]; /* Common (A+B) quantities                */
  PLTD    tXval;              /* Xaxis data to go with the voltage data */
  PLTD    tNorm[ POSS_CHNS ]; /* Normal channel voltage data            */
  PLTD    tComp[ POSS_CHNS ]; /* Complimentary channel voltage data     */
  PLTD    tDiff[ POSS_CHNS ]; /* Differential voltage data              */
  PLTD    tComm[ POSS_CHNS ]; /* Common (A+B) voltage data              */
  OHIS    tHorz[ POSS_CHNS ]; /* Horizontal histogram data              */
  OHIS    tVert[ POSS_CHNS ]; /* Vertical histogram data                */
  } SCOP;
```

tParm A structure of type PARM that contains acquisition parameter. The PARM is discussed in full detail in Section 2-4.

LVoff[n] Offset voltage used for scope acquire, specified in mV, one per channel

lVdif[n] Differential offset voltage used for scope acquire, specified in mV, one per channel

lVcom[n] Common mode offset voltage used for scope acquire, specified in mV, one per channel

lTper Time per division specified in ps – applies to all channels, any of the following are valid values:
5000000, 2000000, 1000000, 500000, 200000, 100000, 50000, 20000, 10000, 5000, 2000, 1000, 500, 200, 100, 50

Default: 10000

lTdel Delay time to start specified in ps – applies to all channels
Valid Range: 24,000 to 100,000,000
Default: 24,000

lPcnt This field specifies the voltage thresholds to be used when calculating rise and fall times. The voltage thresholds are assumed to be symmetrical about the 50% threshold, and this is the distance from the 50% threshold to the starting and ending thresholds. For example if this field is equal to 30, then 20% and 80% thresholds are used. If this field is equal to 40, then 10% and 90% thresholds are used. The absolute voltage levels used are based on the previous pulsefind minimum and maximum voltages. If this field is negative, then the absolute rise and fall thresholds are taken from the following fields lHiRFmV & lLoRFmv.
Default: 30

lHiRFmV Absolute rise/fall voltage if lPcnt<0, in units of mV
Default: +250

lLoRFmV Absolute rise/fall voltage if lPcnt<0, in units of mV
Default: -250

lInps Input selection, can be any of the following:
SCOP_INPS_NORM +Input Only
SCOP_INPS_COMP –Input Only
SCOP_INPS_DIFF +Input minus -Input
SCOP_INPS_BOTH +Input and -Input
SCOP_INPS_COMM +Input plus –Input
Default: SCOP_INPS_NORM

lMeas Measure flag, this is a bitfield which may be created by combining any or all of the following constants:
SCOP_MEAS_RISEFALL – Rise and Fall times are calculated
SCOP_MEAS_VTYPICAL – Vtop and Vbase are calculated
SCOP_MEAS_VEXTREME – Vmin and Vmax are calculated
SCOP_MEAS_OVERUNDR – Overshoot and Undershoot are calculated
SCOP_MEAS_WAVEFORM – Vavg and Vrms are calculated
SCOP_MEAS_VERTHIST – Create a vertical histogram
SCOP_MEAS_HORZHIST – Create a horizontal histogram
SCOP_MEAS_EYEMASKS – Apply an Eye Mask Keep In/Out Region
Default: None of the above are included

lPass This parameter is a bi-directional structure element that tracks the number of acquisitions since last reset. This flag can be read after an execution or set prior to an execution. Setting this parameter to 0 essentially resets this register. It will be automatically incremented when a measurement is performed.
Valid Entries: any integer greater than or equal to 0
Default: 0

lAvgs This variable is used to calculate the number of averages to use. Increasing the number of averages reduces the background noise associated with the algorithms. The number of averages is calculated based on the equation:
AVERAGES = 2^n where n = lAvgs
Valid Entries: any integer greater than or equal to 0
Default: 0 (indicating 2^0 averages = 1 execution.)

tMask MASK Structure which holds mask definition. See the definition above.
Defaults:
tMask.dXwdUI = 0.40
tMask.dXflUI = 0.20
tMask.dYiPct = 0.60
tMask.dV1Rel = 0.20

```
tMask.dV0Rel = 0.20
tMask.dVmask = 64e-3
tMask.dTmask = 700e-12
tMask.dV1pas = scop->tMask.dVmask * 0.75
scop->tMask.dV0pas = scop->tMask.dVmask * 0.75
tMask.dTflat = scop->tMask.dTmask * 3.0 / 7.0
```

dMargin `Margin in percentage for Eye Mask [-1.0 to 1.0]`
Default: 0

dHistDly `Histogram Box center horizontal location, seconds`
Default: `120e-9`

dHistWid `Histogram Box horizontal width, seconds`
Default: `160e-9`

dHistVlt `Histogram Box center vertical location, volts`
Default: 0.0

dHistHgt `Histogram Box vertical height, volts`
Default: 1.6

dAttn[n] `Attenuation factor in dB, this is provided to allow the results to be scaled to compensate for external attenuation from sources such as probes.`
Default: 0

lGood `Flag indicates valid data in structure`
qNorm[n] `Normal channel quantities, one for each channel`
qComp[n] `Complimentary channel quantities, one for each channel`
qDiff[n] `Differential quantities, one for each channel`
qComm[n] `Common (A+B) quantities, one for each channel`
tXval `Xaxis data to go with the voltage data`
tNorm[n] `Normal channel voltage data, one for each channel`
tComp[n] `Complimentary channel voltage data, one for each channel`
tDiff[n] `Differential voltage data, one for each channel`
tComm[n] `Common (A+B) voltage data, one for each channel`
tHorz[n] `Horizontal histogram data, one for each channel`
tVert[n] `Vertical histogram data, one for each channel`

void __stdcall FCNL_DefScop (SCOP *scop)

This function is used to fill the scop structure for the Scope tool with reasonable default values. It is recommended that this function be called initially even if parameters within the structure are to be adjusted manually, and may be called repeatedly to reestablish initial conditions; however, this will impact test time.

Before calling this function, zero out the SCOP structure using the standard memset() function to ensure that any information pertaining to dynamic memory allocation is cleaned out prior to using the structure.

INPUTS

scop - Pointer to a SCOP structure. Memory needs to be allocated by the caller.

OUTPUTS

None.

void __stdcall FCNL_ClrScop (SCOP *scop)

This function frees any dynamic memory that may have been allocated during previous data acquisitions and clears out the scop structure.

INPUTS

scop - Pointer to a SCOP structure. Memory needs to be allocated by the caller.

OUTPUTS

None.

EXAMPLE

```
static SCOP scope;                                //declare scope to a structure of type
                                                  //SCOP
memset ( &scope, 0, sizeof ( SCOP ) );            //clear the memory for scope structure
FCNL_DefScop ( &scope);                           //set scope structures to default values

FCNL_RqstPkt ( ApiDevId, &scope, WIND_SCOP );     //execute the measurement.
FCNL_RqstAll ( ApiDevId, &scope, WIND_SCOP );     //get plot data

FCNL_ClrScop ( &scope);                           //deallocate the structure
```

2-40 SERIAL ATA GEN2I & GEN2M TOOL

The SERIAL ATA GEN2I & GEN2M Tool provides both timing and amplitude compliance measurements. It accurately determines device performance by quantifying both random and deterministic jitter components.

```
typedef struct
  {
  /* Input parameters  */
  long    lCompPnt;          /* Compliance Point 0-Gen2i 1-Gen2m       */
  long    lVoff;             /* Offset voltage used for scope acquire  */
  double  dAttn;             /* Attenuation factor (dB)                */
  KPWM    tKpwm;             /* KPWM structure holds most information  */
  /* Output parameters */
  long    lGood;             /* Flag indicates valid data in structure */
  long    lPad2;
  double  dTjit10;           /* TJ @ Fbaud / 10                        */
  double  dRjit10;           /* RJ @ Fbaud / 10                        */
  double  dDjit10;           /* DJ @ Fbaud / 10                        */
  double  dTjit500;          /* TJ @ Fbaud / 500                       */
  double  dRjit500;          /* RJ @ Fbaud / 500                       */
  double  dDjit500;          /* DJ @ Fbaud / 500                       */
  double  dTjit1667;         /* TJ @ Fbaud / 1667                      */
  double  dRjit1667;         /* RJ @ Fbaud / 1667                      */
  double  dDjit1667;         /* DJ @ Fbaud / 1667                      */
  PLTD    tDdjt10;           /* DCD+DDJvsUI @ Fbaud / 10               */
  PLTD    tFreq10;           /* Frequency PLTD @ Fbaud / 10            */
  PLTD    tBath10;           /* Bathtub PLTD @ Fbaud / 10              */
  PLTD    tDdjt500;          /* DCD+DDJvsUI @ Fbaud / 500              */
  PLTD    tFreq500;          /* Frequency PLTD @ Fbaud / 500           */
  PLTD    tBath500;          /* Bathtub PLTD @ Fbaud / 500             */
  PLTD    tDdjt1667;         /* DCD+DDJvsUI @ Fbaud / 1667             */
  PLTD    tFreq1667;         /* Frequency PLTD @ Fbaud / 1667          */
  PLTD    tBath1667;         /* Bathtub PLTD @ Fbaud / 1667            */
  PLTD    tNrmScop;          /* Normal channel voltage data            */
  PLTD    tCmpScop;          /* Complimentary channel voltage data     */
  PLTD    tDifScop;          /* Differential voltage data              */
  PLTD    tComScop;          /* Common (A+B) voltage data              */
} ATA2;
```

lCompPnt Compliance Point, may be one of the following constants:
0 – GEN2I Specification
1 – GEN2M Specification
Default: 0

lVoff Offset voltage used for scope acquire, specified in mV
Default: 0

dAttn Attenuation factor in dB, this is provided to allow the results to be scaled to compensate for external attenuation from sources such as probes.
Default: 0

tKpwm Known Pattern With Marker Tool which specifies most of the input and output parameters necessary for a data signal analysis. The user will need to review all of the default parameters of the Known Pattern With Marker Tool and decide which to change.

lGood Flag indicates valid data in structure

lPad2 Internal parameter, do not modify.

dTjit10 Total Jitter with Fbaud/10 High Pass Filter Applied

dRjit10	Random Jitter with Fbaud/10 High Pass Filter Applied
dDjit10	Deterministic Jitter with Fbaud/10 High Pass Filter Applied
dTjit500	Total Jitter with Fbaud/500 High Pass Filter Applied
dRjit500	Random Jitter with Fbaud/500 High Pass Filter Applied
dDjit500	Deterministic Jitter with Fbaud/500 High Pass Filter Applied
dTjit1667	Total Jitter with Fbaud/1667 High Pass Filter Applied
dRjit1667	Random Jitter with Fbaud/1667 High Pass Filter Applied
dDjit1667	Deterministic Jitter with Fbaud/1667 High Pass Filter Applied
tDdjt10	DCD+DDJvsUI @ Fbaud/10
tFreq10	Frequency plot @ Fbaud/10
tBath10	Bathtub plot @ Fbaud/10
tDdjt500	DCD+DDJvsUI @ Fbaud/500
tFreq500	Frequency plot @ Fbaud/500
tBath500	Bathtub plot @ Fbaud/500
tDdjt1667	DCD+DDJvsUI @ Fbaud/1667
tFreq1667	Frequency plot @ Fbaud/1667
tBath1667	Bathtub plot @ Fbaud/1667
tNrmScop	Normal channel voltage data
tCmpScop	Complimentary channel voltage data
tDifScop	Differential mode (IN - /IN) voltage data
tComScop	Common mode (IN + /IN) voltage data

void __stdcall FCNL_DefAta2 (ATA2 *ata2)

This function is used to fill the ata2 structure for the Serial ATA tool with reasonable default values. It is recommended that this function be called initially even if parameters within the structure are to be adjusted manually, and may be called repeatedly to reestablish initial conditions; however, this will impact test time.

Before calling this function, zero out the ATA2 structure using the standard memset() function to ensure that any information pertaining to dynamic memory allocation is cleaned out prior to using the structure.

INPUTS

ata2 - Pointer to a ATA2 structure. Memory needs to be allocated by the caller.

OUTPUTS

None.

void __stdcall FCNL_ClrAta2 (ATA2 *ata2)

This function frees any dynamic memory that may have been allocated during previous data acquisitions and clears out the ata2 structure.

INPUTS

ata2 - Pointer to a ATA2 structure. Memory needs to be allocated by the caller.

OUTPUTS

None.

EXAMPLE

```
static ATA2 ata2;                              //declare ata2 to a structure of type
                                               //ATA2
memset ( &ata2, 0, sizeof ( ATA2 ) );          //clear the memory for ata2 structure
FCNL_DefAta2 ( &ata2);                         //set ata2 structures to default values

FCNL_RqstPkt ( ApiDevId, &ata2, WIND_ATA2 );   //execute the measurement.
FCNL_RqstAll ( ApiDevId, &ata2, WIND_ATA2 );   //get plot data

FCNL_ClrAta2 ( &ata2);                         //deallocate the structure
```

2-41 SERIAL ATA GEN1X & GEN2X TOOL

The SERIAL ATA GEN1X & GEN2X Tool provides both timing and amplitude compliance measurements. It accurately determines device performance by quantifying both random and deterministic jitter components.

```
typedef struct
  {
  /* Input parameters  */
  long    lCompPnt;          /* Compliance Point, see defines above    */
  long    lVoff;             /* Offset voltage used for scope acquire  */
  double  dAttn;             /* Attenuation factor (dB)                */
  EYEH    tEyeh;             /* EYEH structure holds most information  */
  /* Output parameters */
  long    lGood;             /* Flag indicates valid data in structure */
  long    lPad2;
  PLTD    tNrmScop;          /* Normal channel voltage data            */
  PLTD    tCmpScop;          /* Complimentary channel voltage data     */
  PLTD    tDifScop;          /* Differential voltage data              */
  PLTD    tComScop;          /* Common (A+B) voltage data              */
  } ATAX;
```

lCompPnt Compliance Point, may be one of the following constants:
ATAX_RX_1X_MODE - 1X Receive Mode
ATAX_TX_1X_MODE - 1X Transmit Mode
ATAX_RX_2X_MODE - 2X Receive Mode
ATAX_TX_2X_MODE - 2X Transmit Mode
Default: 0

lVoff Offset voltage used for scope acquire, specified in mV
Default: 0

dAttn Attenuation factor in dB, this is provided to allow the results to be scaled to compensate for external attenuation from sources such as probes.
Default: 0

tEyeh Random Data With Bit Clock Tool which specifies most of the input and output parameters necessary for a data signal analysis. The user will need to review all of the default parameters of the Random Data With Bit Clock Tool and decide which to change.

lGood Flag indicates valid data in structure

lPad2 Internal parameter, do not modify.

tNrmScop Normal channel voltage data

tCmpScop Complimentary channel voltage data

tDifScop Differential mode (IN - /IN) voltage data

tComScop Common mode (IN + /IN) voltage data

void __stdcall FCNL_DefAtax (ATAX *atax)

This function is used to fill the atax structure for the Serial ATA tool with reasonable default values. It is recommended that this function be called initially even if parameters within the structure are to be adjusted manually, and may be called repeatedly to reestablish initial conditions; however, this will impact test time.

Before calling this function, zero out the ATAX structure using the standard memset() function to ensure that any information pertaining to dynamic memory allocation is cleaned out prior to using the structure.

INPUTS

atax - Pointer to a ATAX structure. Memory needs to be allocated by the caller.

OUTPUTS

None.

void __stdcall FCNL_ClrAtax (ATAX *atax)

This function frees any dynamic memory that may have been allocated during previous data acquisitions and clears out the atax structure.

INPUTS

atax - Pointer to a ATAX structure. Memory needs to be allocated by the caller.

OUTPUTS

None.

EXAMPLE

```
static ATAX atax;                                 //declare atax to a structure of type
                                                  //ATAX
memset ( &atax, 0, sizeof ( ATAX ) );             //clear the memory for atax structure
FCNL_DefAtax ( &atax);                            //set atax structures to default values

FCNL_RqstPkt ( ApiDevId, &atax, WIND_ATAX );      //execute the measurement.
FCNL_RqstAll ( ApiDevId, &atax, WIND_ATAX );      //get plot data

FCNL_ClrAtax ( &atax);                            //deallocate the structure
```

2-42 SERIAL ATA TOOL

The SATA Specification requires that jitter measurements be made from Data edge to Data edge across varying spans. The spans are from 0 to 5 UI, and then from 6 to 250 UI. This tool automates these measurements and provides pass/fail results. For the specification point A2, or 25,000 UI, a 1010 pattern is used and the Low frequency modulation tool can be used.

This tool requires no knowledge of the data stream prior to making a measurement. It simply measures data edge to data edge and places the measurements in their relative bins. The bin size is base on the "Bit Rate (Gb/s)" entered into the tool plus or minus 0.5 UI. For example, if a span of 1.12UI is measured, it is placed in the 1UI bin. Some random time later (see SIA-3000 measurement theory) another measurement is made and is 2.34 UI, so it is placed in the 2UI bin. After each bin has sufficient data, a tail-fit is performed on each UI span to get RJ, DJ and TJ at 10-12 BER.

```
typedef struct
  {
  PARM    tParm;              /* Contains acquisition parameters        */
  long    lPassCnt;
  long    lPad1;
  double  dBitRate;           /* Bit Rate, must be specified            */
  /* Output parameters */
  long    lGood;              /* Flag indicates valid data in structure */
  long    lTfit;              /* Flag indicates all tailfits are good   */
  long    lMinHits;           /* Min hits across all DJ spans           */
  long    lPad2;

  long    lSetSave[SATA_TFITS];/*****************************************/
  long    lPad3;               /*                                       */
  long    lBinNumb[SATA_TFITS];/*  These values are all used internally */
  long    lPad4;               /*                                       */
  double  dLtSigma[SATA_TFITS][PREVSIGMA];/*  DO NOT ALTER!             */
  double  dRtSigma[SATA_TFITS][PREVSIGMA];/******************************/

  double  dDjit5, dDjit250;   /* DJ at 5 and 250 spans                  */
  double  dTjit5, dTjit250;   /* TJ at 5 and 250 spans                  */
  long    lHits[SATA_TFITS];  /* Contains count of histogram hits       */
  long    lPad5;               /*                                       */
  TFIT    tTfit[SATA_TFITS];  /* Structure containing tail-fit info     */
  PLTD    tDjit;              /* Determinstic Jitter plot               */
  PLTD    tTjit;              /* Total Jitter plot                      */
  PLTD    tHist[SATA_TFITS];  /* Histograms for specific spans          */
  } SATA;
```

tParm A structure of type PARM that contains acquisition parameter. The PARM is discussed in full detail in Section 2-4.

lPassCnt This parameter is a bi-directional structure element that tracks the number of acquisitions since last reset. This flag can be read after an execution or set prior to an execution. Setting this parameter to 0 essentially resets the accumulated data. A measurement can be performed repeatedly with the same structure. It will be automatically incremented by the next measurement.
Valid Entries: any integer greater than or equal to 0
Default: 0

dBitRate Bit Rate, must be specified
Default: 1.5e9

lGood Flag indicates valid data in structure

lTfit `Flag indicates all tailfits are good`
lMinHits `Min hits across all DJ spans`
lSetSave[n],lBinNumb[n],dLtSigma[n][m],dRtSigma[n][m] `These values are all used internally, DO NOT ALTER!`
dDjit5 `DJ at 5 spans`
dDjit250 `DJ at 250 spans`
dTjit5 `TJ at 5 spans`
dTjit250 `TJ at 250 spans`
lHits[n] `Contains count of histogram hits`
tTfit[n] `Structure containing tail-fit info`
tDjit `Determinstic Jitter`
tTjit `Total Jitter`
tHist[n] `Histograms for specific spans`

void __stdcall FCNL_DefSata (SATA *sata)

This function is used to fill the sata structure for the Serial ATA tool with reasonable default values. It is recommended that this function be called initially even if parameters within the structure are to be adjusted manually, and may be called repeatedly to reestablish initial conditions; however, this will impact test time.

Before calling this function, zero out the SATA structure using the standard memset() function to ensure that any information pertaining to dynamic memory allocation is cleaned out prior to using the structure.

INPUTS

sata - Pointer to a SATA structure. Memory needs to be allocated by the caller.

OUTPUTS

None.

void __stdcall FCNL_ClrSata (SATA *sata)

This function frees any dynamic memory that may have been allocated during previous data acquisitions and clears out the sata structure.

INPUTS

sata - Pointer to a SATA structure. Memory needs to be allocated by the caller.

OUTPUTS

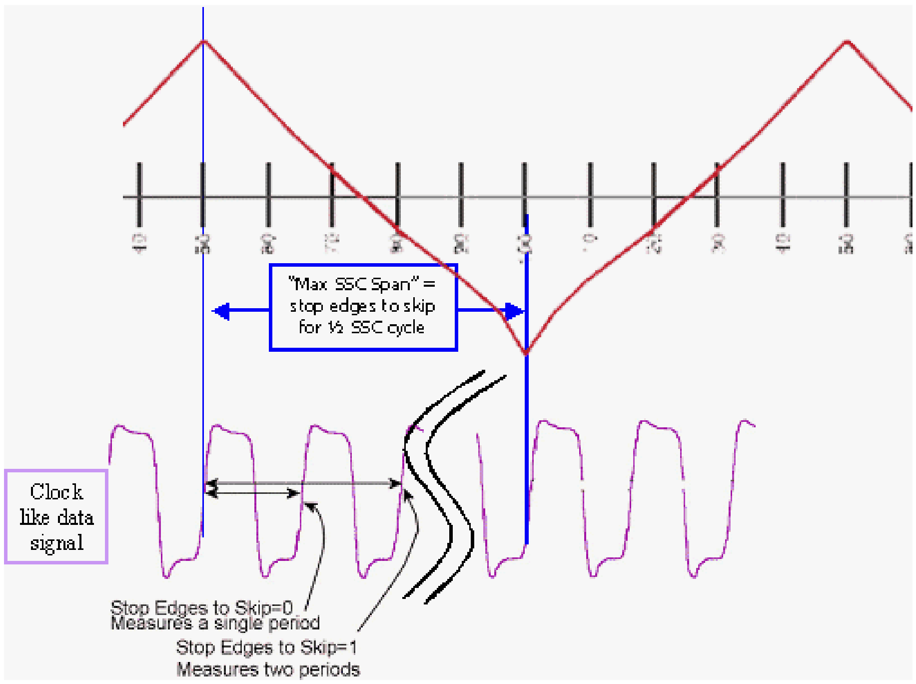
None.

EXAMPLE

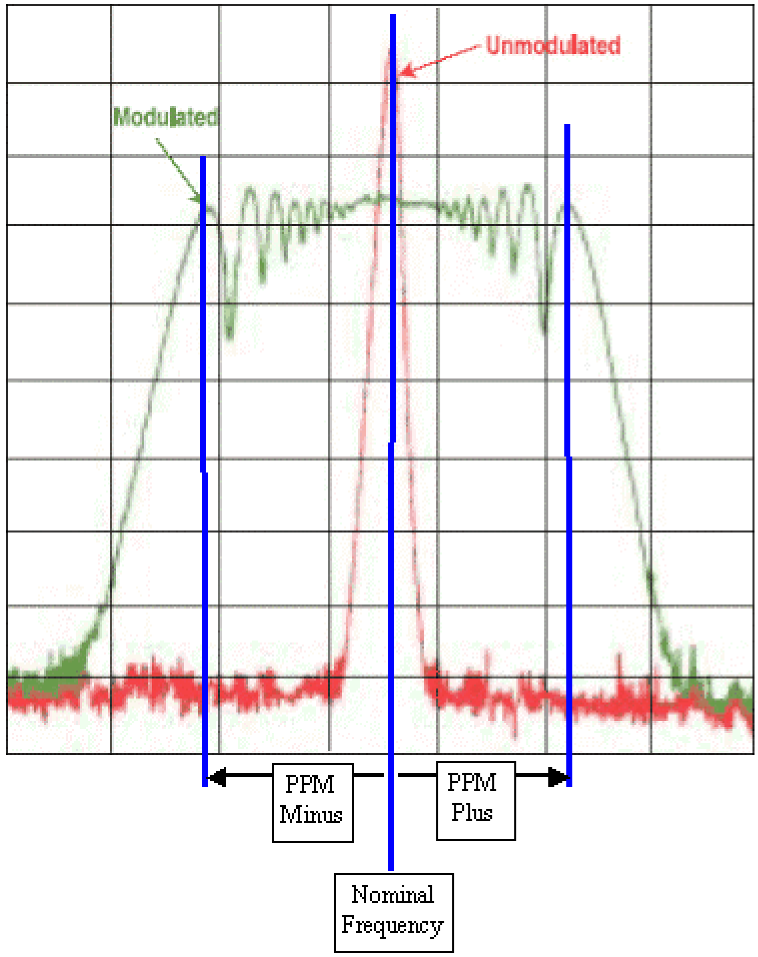
```
static SATA ser_ata;                            //declare ser_ata to a structure of type
                                                //SATA
memset ( &ser_ata, 0, sizeof ( SATA ) );        //clear the memory for ser_ata structure
FCNL_DefSata ( &ser_ata);                       //set ser_ata structures to default values

FCNL_RqstPkt ( ApiDevId, &ser_ata, WIND_SATA );  //execute the measurement.
FCNL_RqstAll ( ApiDevId, &ser_ata, WIND_SATA );  //get plot data

FCNL_ClrSata ( &ser_ata);                       //deallocate the structure
```

2-43 SPREAD SPECTRUM TOOL

The SSC tool will measure the appropriate number of the input clock cycles to see the maximum peak-to-peak deviation due to the SSC profile (see figure below). This will be equal to the fundamental frequency divided by the frequency of ½ the SSC cycle. The tool will search for this maximum deviation within the range of possible SSC frequencies entered in the "Max. SSC Freq. (kHz)" and "Min. SSC Freq. (kHz)" inputs.



The SSC tool will then measure a histogram of this span and determine the PPM deviation form the input "Nominal Freq. (MHz)". The figure below shows what this corresponds to in the frequency domain.




```
typedef struct
  {
  /* Input parameters  */
  PARM    tParm;             /* Contains acquisition parameters        */
  double  dBegFreq;          /* Starting freq to find peak jitter span */
  double  dEndFreq;          /* Ending freq to find peak jitter span   */
  double  dNomFreq;          /* Nominal frequency                      */
  long    lClokDiv;          /* Scaling factor for divided clock       */
  long    lHstSamp;          /* Samples for histogram at peak span     */
  long    lPpmAvgs;          /* 2^lPpmAvgs used to average results     */
  long    lSscStds;          /* Standard used, see above defines       */
  /* Output parameters */
  long    lGood;             /* Flag indicates valid data in structure */
  long    lMaxSpan;          /* Span across which max jitter is found  */
  double  dCarFreq;          /* Measured carrier frequency             */
  double  dModFreq;          /* Apparent jitter modulation frequency   */
  double  dPpmPstv;          /* Parts per million positive             */
  double  dPpmNgtv;          /* Parts per million negative             */
  double  dMeasMin;          /* Minimum value in measured normal data  */
  double  dMeasMax;          /* Maximum value in measured normal data  */
  double  dMeasAvg;          /* Average value of measured normal data  */
  double  dMeasSig;          /* 1-Sigma value of measured normal data  */
  double  dUnitInt;          /* Unit Interval of data signal           */
  PLTD    tHist;             /* Histogram of results for peak freq.    */
  PLTD    tSigm;             /* 1-Sigma data to find max jitter span   */
  } SSCA;
```

tParm	A structure of type PARM that contains acquisition parameter. The PARM is discussed in full detail in Section 2-4.
dBegFreq	Starting freq to find peak jitter span Valid Range: 1e3 to 1e6 Default: 30e3
dEndFreq	Ending freq to find peak jitter span Valid Range: 1e3 to 1e6 Default: 33e3
dNomFreq	Nominal frequency Valid Range: 1e6 to 10e9 Default: 750e6
lClokDiv	Scaling factor for divided clock Valid Range: 1 to 5 Default: 1
lHstSamp	Samples for histogram at peak span Valid Range: 1 to 950,000 Default: 100,000
lPpmAvgs	This variable is used to calculate the number of averages to use. Increasing the number of averages reduces the background noise associated with the algorithm. The number of averages is calculated based on the equation: AVERAGES = 2^n where n = lFftAvgs Valid Entries: any integer greater than or equal to 0 Default: 0 (indicating 2^0 averages = 1 execution.)
lSscStds	Standard used, the following defines apply: SSCA_USER, SSCA_SATA1, SSCA_SATA2, SSCA_PCIX Default: SSCA_SATA1
lGood	Flag indicates valid data in structure
lMaxSpan	Span across which max jitter is found
dCarFreq	Measured carrier frequency
dModFreq	Apparent jitter modulation frequency
dPpmPstv	Parts per million positive
dPpmNgtv	Parts per million negative
dMeasMin	Minimum value in measured normal data
dMeasMax	Maximum value in measured normal data
dMeasAvg	Average value of measured normal data
dMeasSig	1-Sigma value of measured normal data
dUnitInt	Unit Interval of data signal
tHist	Histogram of results for peak freq.
tSigm	1-Sigma data to find max jitter span

void __stdcall FCNL_DefSsca (SSCA *ssca)

This function is used to fill the ssca structure for the SSC tool with reasonable default values. It is recommended that this function be called initially even if parameters within the structure are to be adjusted manually, and may be called repeatedly to reestablish initial conditions; however, this will impact test time.

Before calling this function, zero out the SSCA structure using the standard memset() function to ensure that any information pertaining to dynamic memory allocation is cleaned out prior to using the structure.

INPUTS

ssca - Pointer to a SSCA structure. Memory needs to be allocated by the caller.

OUTPUTS

None.

void __stdcall FCNL_ClrSsca (SSCA *ssca)

This function frees any dynamic memory that may have been allocated during previous data acquisitions and clears out the ssca structure.

INPUTS

ssca - Pointer to a SSCA structure. Memory needs to be allocated by the caller.

OUTPUTS

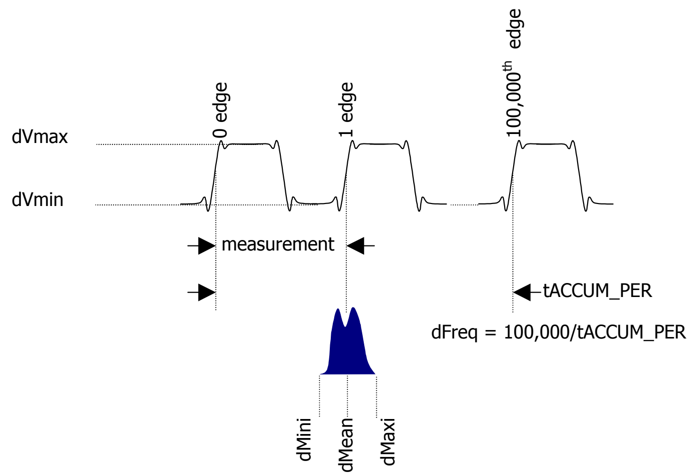
None.

EXAMPLE

```
static SSCA spread;                             //declare spread to a structure of type
                                                //SSCA
memset ( &spread, 0, sizeof ( SSCA ) );         //clear the memory for spread structure
FCNL_DefSsca ( &spread);                        //set spread structures to default values

FCNL_RqstPkt ( ApiDevId, &spread, WIND_CLOK );  //execute the measurement.
FCNL_RqstAll ( ApiDevId, &spread, WIND_CLOK );  //get plot data

FCNL_ClrSsca ( &spread);                        //deallocate the structure
```

2-44 STATISTICS TOOL

The statistics tool is used to capture a few basic parameters of a measurement that the user selected in the tParm structure. The statistics tool will also return voltage parameters of the signal under test. As seen in the accompanying example for a simple period measurement, the number of parameters returned may be more extensive than is typically desired in a production program. For a simple time measurement, it is best to use the histogram tool which can be set to return just the statistics of the interest and not any of the voltage information nor the extra timing measurements as is captured in this tool. There is added test time to capture duty cycle, frequency and the voltage parameters.



Example of a period measurement using the Statistics Utility

```
typedef struct
  {
  /* Input parameters  */
  PARM    tParm;              /* Contains acquisition parameters        */
  long    lPfnd;              /* Force a pulse-find before each measure */
  long    lAutoFix;           /* If true perform a pulsefind as req'd   */
  /* Output parameters */
  long    lGood;              /* Flag indicates valid data in structure */
  long    lPad1;
  double  dMean;              /* Contains the returned average value    */
  double  dMaxi;              /* Contains the returned maximum value    */
  double  dMini;              /* Contains the returned minimum value    */
  double  dSdev;              /* Contains the returned 1-Sigma value    */
  double  dDuty;              /* Contains the returned duty cycle       */
  double  dFreq;              /* Contains the carrier frequency         */
  double  dVmin[ 2 ];         /* Pulse-find Min voltage for Start&Stop  */
  double  dVmax[ 2 ];         /* Pulse-find Max voltage for Start&Stop  */
  } STAT;
```

tParm A structure of type PARM that contains acquisition parameter. The PARM is discussed in full detail in Section 2-4.

lPfnd A flag used to force the execution of a pulse find execution. In normal operation, the SIA3000 dynamically decides whether a pulsefind is necessary based on previous test conditions. In may cases, this is sufficient. However, in a production environment, the previous test may have the same type of voltage settings, however, the devices are different and may have different voltage characteristics and would thus require a pulse find on each device. Be aware that most production test have specified amplitudes at which measurements are to be made such that the programmer must specify the amplitude in tPARM rather than use pulse find to establish test conditions.

Valid Entries: 0 – No pulsefind prior to measurement
1 – perform a pulse find.

Default: 0

lAutoFix Flag to indicate to the system whether pulse find should be performed if needed. This flag essentially enables the "automatic pulse find" capability which will execute a

	pulsefind based on the previous test setup and not with any regard to device-device variations in amplitude. Valid Entries: 0 – No pulsefind prior to measurement 1 – Pulsefind if the measurement mode changed. Default: 0
lGood	Flag used to indicate valid output data in structure.
dMean	Contains the returned average value.
dMaxi	Contains the returned maximum value.
dMini	Contains the returned minimum value.
dSdev	Contains the returned 1-Sigma value.
dDuty	Contains the returned duty cycle of the signal being measured. This is not measured if a two channel measurement is being performed.
dFreq	Contains the frequency of the signal being measured. This is not measured if a two channel measurement is being performed.
dVmin	Min voltage returned from last pulse-find. *It is important to note that the accuracy of this voltage measurement is severely bandwidth limited. For accurate amplitude measurements, use the oscilloscope tool.*
dVmax	Max voltage returned from last pulse-find. *It is important to note that the accuracy of this voltage measurement is severely bandwidth limited. For accurate amplitude measurements, use the oscilloscope tool.*

void __stdcall FCNL_DefStat (STAT *stat)

This function is used to fill the stat structure for the Statistics tool with reasonable default values. It is recommended that this function be called initially even if parameters within the structure are to be adjusted manually, and may be called repeatedly to reestablish initial conditions; however, this will impact test time.

Before calling this function, zero out the STAT structure using the standard memset() function to ensure that any information pertaining to dynamic memory allocation is cleaned out prior to using the structure.

INPUTS

stat - Pointer to a STAT structure. Memory needs to be allocated by the caller.

OUTPUTS

None.

void __stdcall FCNL_ClrStat (STAT *stat)

This function frees any dynamic memory that may have been allocated during previous data acquisitions and clears out the stat structure.

INPUTS

stat - Pointer to a STAT structure. Memory needs to be allocated by the caller.

OUTPUTS

None.

EXAMPLE

```
#define TRUE 1
#define FALSE 0
static STAT statistics;                           //declare statistics to be a structure of
                                                  //type STAT
memset ( &statistics, 0, sizeof ( STAT ) );       //clear the memory for statistics structure
FCNL_DefStat ( &statistics);                      //set statistics structure to default values
                                                  //NOTE:  statistics.tparm, are also set to
                                                  //defaults by this command.
statistics.tParm.lChanNum = 2 | (3<<16);          //Set ch 2 for start and ch 3 for stop
statistics.tParm.lSampCnt = 1,000;                //Set sample size to 1k per burst.
statistics.tParm.lFuncNum = FUNC_TPD_PP;          //make propogation delay meas. from
                                                  //rising edge on ch2 to rising edge on
                                                  //ch 2.
statistics.tParm.lExternArm = 1;                  //use ch1 as arm channel
statistics.tParm.lAutoArm = ARM_EXTRN;            //use External Arming
statistics.tParm.lStrtCnt = 1;                    //start measurement on first edge of
                                                  //ch2 after the arm signal.
statistics.tParm.lStopCnt = 6;                    //stop measurement on sixth edge of
                                                  //ch3 after the arm signal.
FCNL_RqstPkt ( ApiDevId, & statistics, WIND_STAT );  //execute the measurement.

If (statistics.lGood=TRUE)
  {
  printf("\nSkew = %d\n",statistics.dMean);        //Print skew measurement
  printf("\nSkew jitter = %d\n",statistics.dSdev);  //print skew jitter result
  }

FCNL_ClrStat ( &statistics);                      //deallocate the structure
```

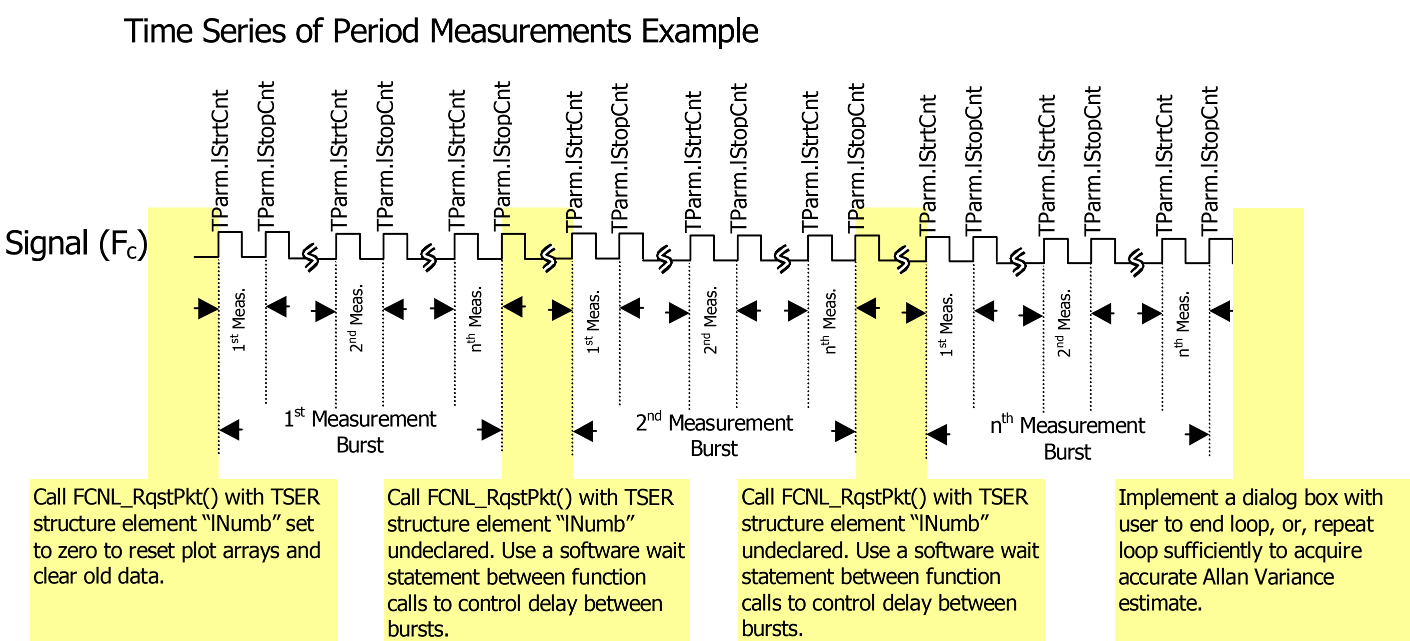
2-45 STRIPCHART TOOL

The Time Series Tool is used to capture timing issues that are occurring at sub Hertz rates. This tool performs a measurement, extracts the statistical information from the measurement burst then waits a defined interval and performs the measurement again. This type of measurement is used in Allan Variance measurements and in real time debugging of various environment parameters (such as VDD, VIL/VIH, timing skew, etc.) and their impact on various time measurements (such as period, jitter, slew rate and modulation). To use this tool, the user must initiate a measurement with the TSER structure in a loop that includes the wait time between measurements.

If this tool is to be used as a debug tool, it is recommended that the plot be redrawn between measurements so as to allow the user to see a real-time display of the successive measurements. It is also recommended that this routine be placed in a user-aborted infinite loop such that the user can initiate and stop a Time Series measurement session.

If this tool is used to simply gather a fixed number of successive measurements and to analyze the variance of the mean/peak-peak/1s/min/max over the said number of successive iterations, then the last execution's plot structures will contain all of the combined results.

In both cases, be sure to initialize the TSER structure element lNumb to zero when the first measurement is performed via a call to FCNL_RqstPkt(). On subsequent calls, be sure to leave the lNumb parameter undeclared so that the tool will continue to accumulate measurements on each successive measurement burst. Measurements are acquired as follows:




```
typedef struct
  {
  /* Input parameters  */
  PARM    tParm;              /* Contains acquisition parameters        */
  long    lNumb;              /* Measurements so far, set to 0 to reset */
  long    lPad1;
  double  dSpan;              /* Span between measurements in seconds   */
  long    lAutoFix;           /* If true perform a pulsefind as req'd   */
```

```
/* Output parameters */
long    lGood;             /* Flag indicates valid data in structure */
double  dYstd;             /* 1-Sigma value calculated on all data   */
double  dAvar;             /* Allan variance calculation             */
                           /*****************************************/
double  dSumm;             /*  These values are all used internally  */
double  dTyme;             /*   as part of the measurement process   */
                           /*             DO NOT ALTER!              */
                           /*****************************************/
PLTD    tMean;             /* Contains the average plot array        */
PLTD    tMini;             /* Contains the minimum plot array        */
PLTD    tMaxi;             /* Contains the maximum plot array        */
PLTD    tTime;             /* Contains the time samples were taken   */
PLTD    tSdev;             /* Contains the 1-Sigma plot array        */
PLTD    tPeak;             /* Contains the ( max - min ) plot array  */
} TSER;
```

tParm A structure of type PARM that contains acquisition parameter. The PARM is discussed in full detail in Section 2-4.

lNumb When implemented correctly, a measurement is performed repeatedly with the TSER structure to generate a Time Series plot of a given measurement. *(User defines measurement parameters in tParm.)*. For the first execution, set lNumb to zero to reset the plot arrays. All subsequent measurements should not assign any value to this structure element. This parameter is automatically incremented by the next measurement and can be read after execution to determine the number of times this structure has been called.
Valid Entries: 0 reset counter and clear all plot data.
Default: Increment previous value.

lAutoFix Flag indicating whether to perform a pulse-find as required. Setting this value to any integer greater than zero tells the measurement to perform a pulse find if needed. The system will know if a measurement was recently performed and if a pulse find is necessary.
Valid Entries: 0 – no pulsefind prior to measurement
1 – pulsefind if the measurement mode changed.
Default: 0

lGood Flag indicates valid output data in structure.

dYstd 1-Sigma, or standard deviation of all data.

dAvar Allan variance estimate.

tMean Structure of type PLTD which contains all of the plot information to generate a diagram of mean values versus iteration number. Use this in PLTD structure in conjunction with the structure tTime to generate a Maximum measurement versus time plot. See Section 2-3 for details of the PLTD structure and its elements.

tMini Structure of type PLTD which contains all of the plot information to generate a diagram of minimum measurement of a given burst versus iteration number. Use this in PLTD structure in conjunction with the structure tTime to generate a Maximum measurement versus time plot. See Section 2-3 for details of the PLTD structure and its elements.

tMaxi Structure of type PLTD which contains all of the plot information to generate a diagram of maximum measurement of a given burst versus iteration number. Use this in PLTD structure in conjunction with the structure tTime to generate a

Maximum measurement versus time plot. See Section 2-3 for details of the PLTD structure and its elements.

tTime Structure of type PLTD which contains all of the time values at which measurements were taken. Use this structure in conjunction with tMini, tMaxi, tSdev, tPeak & tMean to plot said structures as a function of time. . See Section 2-3 for details of the PLTD structure and its elements.

tSdev Structure of type PLTD which contains all of the plot information to generate a diagram of 1-Sigma values of a given burst versus iteration number. Use this in PLTD structure in conjunction with the structure tTime to generate a Maximum measurement versus time plot. See Section 2-3 for details of the PLTD structure and its elements.

tPeak Structure of type PLTD which contains all of the plot information to generate a diagram of peak to peak (maximum measurement – minimum measurement) of a given burst versus iteration number. Use this in PLTD structure in conjunction with the structure tTime to generate a Maximum measurement versus time plot. See Section 2-3 for details of the PLTD structure and its elements.

dSumm, dTyme, dSpan These values are all used internally, DO NOT ALTER!

void __stdcall FCNL_DefTser (TSER *tser)

This function is used to fill the tser structure for the Time Series tool with reasonable default values. It is recommended that this function be called initially even if parameters within the structure are to be adjusted manually, and may be called repeatedly to reestablish initial conditions; however, this will impact test time.

Before calling this function, zero out the TSER structure using the standard memset() function to ensure that any information pertaining to dynamic memory allocation is cleaned out prior to using the structure.

INPUTS

tser - Pointer to a TSER structure. Memory needs to be allocated by the caller.

OUTPUTS

None.

void __stdcall FCNL_ClrTser (TSER *tser)

This function frees any dynamic memory that may have been allocated during previous data acquisitions and clears out the tser structure.

INPUTS

tser - Pointer to a TSER structure. Memory needs to be allocated by the caller.

OUTPUTS

None.

EXAMPLE

```
#define TRUE 1
#define FALSE 0
static TSER tiseries;                              //declare tseries to be a structure of
                                                   //type TSER
memset ( &tseries, 0, sizeof ( TSER ) );           //clear the memory for tseries structure
FCNL_DefTser ( &tseries);                          //set tseries structure to default values
                                                   //NOTE:  tseries.tparm, are also set to
                                                   //defaults by this command.
tseries.tParm.lChanNum = 1;                        //Set ch 1 to be measured channel
tseries.tParm.lSampCnt = 1000;                     //Set sample size to 1k per burst.
tseries.tParm.lFuncNum = FUNC_PER;                 //make period measurements

for (I=0;I<100,000;I++)
  {
  FCNL_RqstPkt ( ApiDevId, &tseries, WIND_TSER );  //execute the measurement.
  plot_data(tseries.tPeak);                        //call subroutine which generates a plot
  }                                                //of the data contained in the
                                                   //tseries.tPeak PLTD structure.

FCNL_ClrTser ( &tseries);                          //deallocate the structure



void plot_data(PLTD *plotstruc)
  {                                                //see section 2-40 for an example of
                                                   //a subroutine which will plot a PLTD
                                                   //structure.
}
```

2-46 RETRIEVING SPIKELISTS

Many of the tools that contain FFT's have the ability to detect and characterize spikes by their frequency and amplitude from within the GUI. The commands used to retrieve the spikelists were designed to remain flexible, and if used properly will adapt from release to release with a simple recompile of your source code. This functionality is supported via the low level GPIB command set and the low level communication functions in Section 4. The spikelist GPIB commands take the following form:

Command syntax- :SPIK`elist:<toolname>(@n)<offset>`

```
<toolname>  is replaced with the same name used with the :ACQUIRE command
(@n)        is used to specify the channel which the spikelist is taken
            from
<offset>    is the length in bytes from the start of a binary packet to
            the pointer to the spikelist to be returned in the same binary
            packet, it is normally calculated from the structure
            definition
```

The correct way to obtain the spikelist is shown in the following example:

```
// Initialize RCPM structure and set to defaults
static RCPM bcam;
memset ( &bcam, 0, sizeof ( RCPM ) );
FCNL_DefRcpm ( &bcam);

//execute the measurement
FCNL_RqstPkt ( ApiDevId, &bcam, WIND_RCPM );

// Create the command to get the spikelist
sprintf(buffer, ":SPIK:CLKANDMARK(@1)%i", (long)&bcam.lPeakData-
(long)&bcam);

// Send the command and place the returned data into the spikelist buffer
COMM_ReqBin ( ApiDevId, buffer, strlen(buffer), spikes, sizeof(spikes) );

// Use the spikelist as required
```

2-47 EXAMPLE OF HOW TO DRAW USING A PLTD STRUCTURE:

```
/*****************************************************************************/
/* DrawPlot() is a function that will plot a graph based on the variables defined */
/* in the PLTD structure passed into this function.                          */
/* (1)  get initial (x,y) coordinates within diagram to start plot.          */
/* (2)  Normalize (x,y) coordinates to amplitudes between 0 and 1 to represent    */
/*      their relative location between [xmin,xmax] or [ymin,ymax] for       */
/*      x coordinates and y coordinates respectively                         */
/* (3)  Initialize the pointer pCdc to the start of the plot in units of pixels   */
/* (4)  step through the data array, normalize the coordinates and pass them to   */
/*      the pCdc function to draw a line to from the previous pCdc location.      */
/* (5) repeat step 4 for all coordinates.                                    */
/* The variables passed into the function are:                               */
/* CDC *pCdc - Windows® pointer to communicate cursor location during plot.       */
/* Crect *wind - Windows® pointer to indicate window size and location in the     */
/*               display. the parameters are in units of pixels top, bottom, left */
/*               and right define the boundaries for the display window. The      */
/*               origin is set to the upper left hand corner with increasing      */
/*               amplitude to the lower right hand corner.                   */
/* PLTD *pldt - Wavecrest plot structure                                     */
/* double xmax - user specified maximum x value for x-axis. This may be larger    */
/*               than pltd.dXmax if a margin is desired. Xmax is in same units as */
/*               the pldt structure's x axis elements.                       */
/* double xmin - user specified minimum x value for x-axis. This may be smaller   */
/*               than pltd.dXmin if a margin is desired. Xmin is in same units as */
/*               the pldt structure's x axis elements.                       */
/* double ymax - user specified maximum y value for y-axis. This may be larger    */
/*               than pltd.dYmax if a margin is desired. Ymax is in same units    */
/*               as the pldt structure's y axis elements.                    */
/* double ymin - user specified minimum x value for x-axis. This may be larger    */
/*               than pltd.dYmin if a margin is desired. Ymin is in same units    */
/*               as the pldt structure's y axis elements.                    */
/*****************************************************************************/


void DrawPlot(CDC *pCdc, CRect *rect, PLTD *plot,
              double xmin, double xmax, double ymin, double ymax)
  {
  long i;
  double x, y;
  double xrange = xmax-xmin;
  double yrange = ymax-ymin;

  x = (plot->dXmin - xmin) / xrange; //normalize first X plot point
  x = (double)(rect.right-rect.left)*x+(double)rect.left; //convert first plot point to
Windows®
                                                          //coordinates in pixels
  y = (plot.dData[0]-ymin)/yrange; //normalize first Y plot point
  y = (double)(rect.bottom-rect.top)*(1.0-y) //convert first plot point to Windows®
    + (double) rect.top;                     //coordinates in pixels. Note, the
                                             //(1.0-y) function is used to account for
                                             //the reverse direction of the coordinate
                                             //system between pixels and the plot elements
  pCdc.MoveTo ((int)x,(int)y); //move display cursor to start of plot

  for ( i = 1; i < plot.lNumb; i++ )
    {
    x = ((plot.dXmax-plot.dXmin)*(double)i //find next x-coordinate
      / (double)(plot.lNumb-1)+plot.dXmin );

    x = (x-xmin)/xrange; //normalize new x-coordinate

    x = (double)(rect.right-rect.left)*x+(double)rect.left; //convert new x-coord to Windows®
                                                            //coordinates in pixels.
    y = ( plot.dData[ i ]-ymin)/yrange; //find next y-coordinate and normalize it

    y = (double)(rect.bottom-rect.top)*(1.0-y) //convert y-coord to Windows® pixel
      + (double) rect.top;                     //coordinates

    pCdc.LineTo((int)x,(int)y); //draw a line from previous cursor
                                //location to new (x,y) coordinates.
    }
  return 0;
  }
```

2-48 DEFINES FOR VALUES IN MEASUREMENT STRUCTURES

The following defines were created to aid in assigning values to various fields in the binary packet structure. They would have been referenced in the above definitions.

```
/* Standard acquire functions */
#define FUNC_TPD_PP   1     /* TPD +/+                2-Chan          */
#define FUNC_TPD_MM   2     /* TPD -/-                2-Chan          */
#define FUNC_TPD_PM   3     /* TPD +/-                2-Chan          */
#define FUNC_TPD_MP   4     /* TPD -/+                2-Chan          */
#define FUNC_TT_P     5     /* Rising edge            1-Chan          */
#define FUNC_TT_M     6     /* Falling Edge           1-Chan          */
#define FUNC_PW_P     7     /* Positive pulse width   1-Chan          */
#define FUNC_PW_M     8     /* Negative pulse width   1-Chan          */
#define FUNC_PER      9     /* Period                 1-Chan          */
#define FUNC_FREQ     10    /* Frequency              1-Chan          */
#define FUNC_PER_M    11    /* Period minus           1-Chan          */
/* Available analysis macros */
#define ANAL_FUNC     0     /* Function analysis macro                */
#define ANAL_JITT     1     /* Jitter analysis macro                  */
#define ANAL_RANG     2     /* Range analysis macro                   */
#define ANAL_CLOK     3     /* PW+/PW-/PER+/PER- macro                */
/* Stop count designators specific to ANAL_FUNC macro */
#define ANL_FNC_FIRST 0     /* Arm start first                        */
#define ANL_FNC_PLUS1 1     /* Start + 1                              */
#define ANL_FNC_START 2     /* Start                                  */
/* Rise/Fall edge designators */
#define EDGE_FALL     0     /* Measurement reference is falling edge  */
#define EDGE_RISE     1     /* Measurement reference is rising edge   */
#define EDGE_BOTH     2     /* Used for DDR in EYEH, DBUS, & FCMP     */
/* Pulsefind mode designators */
#define PFND_FLAT     0     /* Use flat algorithm for pulse-find calc */
#define PFND_PEAK     1     /* Use peak value for pulse-find calc     */
/* Pulsefind percentage designators */
#define PCNT_5050     0     /* Use 50/50 level for pulse-find calc    */
#define PCNT_1090     1     /* Use 10/90 level for pulse-find calc    */
#define PCNT_9010     2     /* Use 90/10 level for pulse-find calc    */
#define PCNT_USER     3     /* Do NOT perform pulse-find; manual mode */
#define PCNT_2080     4     /* Use 20/80 level for pulse-find calc    */
#define PCNT_8020     5     /* Use 80/20 level for pulse-find calc    */
/* Arming mode designators */
#define ARM_EXTRN     0     /* Arm using one of the external arms     */
#define ARM_START     1     /* Auto-arm on next start event           */
#define ARM_STOP      2     /* Auto-arm on next stop event            */
/* Valid lCmdFlag values for special features */
#define CMD_PATNMARK        (1<<4)
#define CMD_BWENHANCED      (1<<10)
/* Constants to assist in setting lArmMove                            */
#define ARMMOVE_MAX_STEP       40
#define ARMMOVE_MIN_STEP      -40
#define ARMMOVE_PICO_PER_STEP  25
/* Used for structure definitions below */
#define POSS_CHNS     10
/* Constants used to identify FFT window type                           */
#define FFT_RCT       0       /* Rectangular window                     */
#define FFT_KAI       1       /* Kaiser-Bessel window                   */
#define FFT_TRI       2       /* Triangular window                      */
#define FFT_HAM       3       /* Hamming window                         */
#define FFT_HAN       4       /* Hanning window                         */
#define FFT_BLK       5       /* Blackman window                        */
```

```
#define FFT_GAU       6      /* Gaussian window                       */
/* Constants used by new scope tool to identify which plot to show */
#define SCOP_INPS_NORM  0
#define SCOP_INPS_COMP  1
#define SCOP_INPS_DIFF  2
#define SCOP_INPS_BOTH  3
#define SCOP_INPS_COMM  4
/* Constants used by new scope tool for measures to calculate */
#define SCOP_MEAS_VEXTREME  (1<<0)
#define SCOP_MEAS_VTYPICAL  (1<<1)
#define SCOP_MEAS_WAVEFORM  (1<<2)
#define SCOP_MEAS_OVERUNDR  (1<<3)
#define SCOP_MEAS_RISEFALL  (1<<4)
#define SCOP_MEAS_VERTHIST  (1<<5)
#define SCOP_MEAS_HORZHIST  (1<<6)
#define SCOP_MEAS_EYEMASKS  (1<<7)
/* Used internally for tailfit algorithm */
#define PREVSIGMA 5
/* Used by Advanced PLL tool */
#define MIN_APLL_INI_DAMP_FCT 1e-3
#define MAX_APLL_INI_DAMP_FCT 10.0
#define MIN_APLL_INI_NAT_FREQ 10.0
#define MAX_APLL_INI_NAT_FREQ 10e9
#define MIN_APLL_INI_NOISEPSD -120
#define MAX_APLL_INI_NOISEPSD -40
/* Used by Phase Noise tool for number of decades to span */
#define DECADES 8
/* Constants for: lTailFit the number of dataCOM tailfits to perform */
#define DCOM_NONE   0
#define DCOM_AUTO   1
#define DCOM_FIT3   2
#define DCOM_FIT5   3
#define DCOM_FIT9   4
#define DCOM_FIT17  5
#define DCOM_ALL    6
#define DCOM_1SIGMA 7
/* Constants for: lFitPcnt the auto-mode percentage to converge within */
#define DCOM_PCNT5   0
#define DCOM_PCNT10  1
#define DCOM_PCNT25  2
#define DCOM_PCNT50  3
/* Constance used for PCI Express mode */
#define PCIX_SCOP_AVG 8
#define PCIX_RX_MODE 0
#define PCIX_TX_MODE 1
#define PCIX_RX_CARD 2
#define PCIX_TX_CARD 3
#define PCIX_RX_SYST 4
#define PCIX_TX_SYST 5
#define PCIX_SPECS 6
#define PCIX_EYE_XDOTS 408
#define PCIX_EYE_YDOTS 630
/* Constants used for Serial ATA tool */
#define SATA_SPANS 250
#define SATA_TFITS 11
/* Constants used to identify which clock analysis measures to calculate
*/
#define CANL_MEAS_RISEFALL  (1<<0)
#define CANL_MEAS_VTYPICAL  (1<<1)
#define CANL_MEAS_VEXTREME  (1<<2)
#define CANL_MEAS_OVERUNDR  (1<<3)
```

```
#define CANL_MEAS_WAVEMATH  (1<<4)
#define CANL_MEAS_TIMEPARM  (1<<5)
#define CANL_MEAS_TAILFITS  (1<<6)
#define CANL_MEAS_PERIODIC  (1<<7)
/* Constants to define the number of random data tailfits to perform */
#define RAND_AUTO   0
#define RAND_FIT3   1
#define RAND_FIT5   2
#define RAND_FIT9   3
#define RAND_FIT17  4
/* Constants for percentage to succeed when Random Data using auto-mode */
#define RAND_PCNT5   0
#define RAND_PCNT10  1
#define RAND_PCNT25  2
#define RAND_PCNT50  3
/* Constants used for Rambus DRCG adjacent cycle tool */
#define DRCG_SWEEPS 4
#define DRCG_CYCLES 6
/* Constants used for Spread Spectrum tool */
#define SSCA_USER  0
#define SSCA_SATA1 1
#define SSCA_SATA2 2
#define SSCA_PCIX  3
/* Constants used for filter selection */
#define FILTERS_DISABLED 0
#define BRICKWALL_FILTER 1
#define ROLLOFF_1STORDER 2
#define ROLLOFF_2NDORDER 3
#define PCIX_CLOK_FILTER 10
```

SECTION 3 – GENERAL COMMAND REFERENCE

The *WAVECREST* Production API provides low level and administrative functions to simplify GPIB operations and provide advanced configuration and measurement options. With the exception of the GPIB functions that initialize device communication via the ApiDevID, these functions are not prerequisite for using the Production API to acquire simple measurements. Most of these routines provide greater flexibility for the advanced user.

This chapter provides a general overview of these functions along with examples for the more commonly used functions. These functions apply to all tools, but may require the pointer to a specific measurement window structure to be passed along with a type identifier (i.e., `WIND_HIST`). For more information regarding specific measurement tools and their corresponding measurement window structures and commands, refer to the previous chapter.

NOTE: `__stdcall` *and* `DllCall` *are part of the function definitions in the header file but can essentially be ignored. They are utilized to provide options when building and using DLLs on Microsoft® Windows. They are implemented to allow the same header file to be used for building the DLL and importing the DLL, ensuring consistent declarations.*

3-1 GPIB COMMUNICATION AND I/O LAYER FUNCTIONS

COMM Layer Functions

The functions in this section provide GPIB bus functionality. GPIB commands may be used in conjunction with Production API commands for advanced functionality. However, COMM_InitDev and COMM_CloseDev are the only functions that must be called in order to utilize the Production API. These functions initialize and close a GPIB connection and acquire an API Device ID through which higher-level Production API measurement functions are called. All other functions are strictly optional unless low-level GPIB functions must be sent or more customized GPIB error handling is required.

Required Functions

`void __stdcall COMM_CloseDev ( long ApiDevId )`

Calls IO_close to close the device specified by ApiDevid.

INPUTS

ApiDevid - API Device ID of the device. This value can be from 1 to 31. The device must have been opened using `COMM_InitDev(..)`.

OUTPUTS

None.

`long __stdcall COMM_InitDev ( long ApiDevTyp, char *devname )`

This function first calls IO_open to open the device specified by devname. Then initializes the device for communication using the COMM library and returns the API Device ID. If an error occurs, a negative number is returned instead.

INPUTS

ApiDevType - An integer value indicating the device type:
HPIB = 0 (HP Systems Only)
GPIB = 1
CUST1 = 11
CUST2 = 12
CUST3 = 13

devname - A pointer to an ASCII string containing a device name.

OUTPUTS

Returns an integer containing the API Device ID or a negative number to indicate an error.

`long __stdcall COMM_ResetDev ( long ApiDevId )`

This function first calls IO_clear to reset the device specified by devname. Then initializes the device for communication using the COMM library. If an error occurs, a negative number is returned instead.

INPUTS

ApiDevid - Integer containing the API Device ID of the device. This value can be from 1 to 31.

OUTPUTS

Returns SIA_SUCCESS upon completion or a negative value to indicate error.

long __stdcall COMM_TalkDev (long ApiDevId, char *cmnd)

This function first clears the response byte of the specified device and then sends the specified command with an "*opc" command appended and waits for the ESB bit in the response byte to be set or LONG_TIME (100) seconds. If the ESB bit is set, the ESR byte is checked for errors. If a timeout occurs or an error is found, a negative value is returned.

INPUTS

ApiDevid - Integer containing the API Device ID of the device. This value can be from 1 to 31.

cmnd - A pointer to a NULL terminated ASCII string containing the command to send.

OUTPUTS

Returns SIA_SUCCESS upon completion or a negative value to indicate error.

long __stdcall COMM_TalkBin (long ApiDevId, char *sCmnd, long lCmnd)

This function first clears the response byte of the specified device and then sends the specified command with an "*opc" command appended and waits for the ESB bit in the response byte to be set or LONG_TIME (100) seconds. If the ESB bit is set the ESR byte is checked for errors. If a timeout occurs or an error is found, a negative value is returned.

INPUTS

ApiDevid - Integer containing the API Device ID of the device. This value can be from 1 to 31.

sCmnd - A pointer to buffer containing the command and binary data to send.

lCmnd - Integer containing the length of sCmnd.

OUTPUTS

Returns SIA_SUCCESS upon completion or a negative value to indicate error.

long __stdcall COMM_ReqAsc (long ApiDevId, char *cmnd, char *sval, long svalLen)

This function first clears the response byte of the specified device and then sends the specified command and waits for the ESB or MAV bit in the response byte to be set or LONG_TIME (100) seconds. If the MAV bit is set, an IO_read of the specified number of bytes minus one (svalLen - 1) is done and the returned NULL terminated ASCII data is placed in the specified location (sval). The ESR byte is then checked for errors. If a timeout occurs or an error is found, a negative value is returned.

INPUTS

ApiDevid - Integer containing the API Device ID of the device. This value can be from 1 to 31.

cmnd - A pointer to a NULL terminated ASCII string containing the command to send.

sval - A pointer to store the returned NULL terminated ASCII data. This buffer must be long enough to hold the expected number of ASCII bytes plus a NULL terminator.

svalLen -An integer containing the length of sval. This value must be the length of or greater than the expected number of bytes returned plus one (1) or the IO_read will not return all the data from the specified device.

OUTPUTS

Returns SIA_SUCCESS upon completion or a negative value to indicate error.

`long __stdcall COMM_ReqBin ( long ApiDevId, char *sCmnd, long lCmnd, char *sRetn, long *lRetn )`

This function first clears the response byte of the specified device and then sends the specified command and waits for the ESB or MAV bit in the response byte to be set or LONG_TIME (100) seconds. If the MAV bit is set, an IO_read of the specified number of bytes (lRetn) is done and the returned binary data is placed in the specified location (sRetn). The ESR byte is then checked for errors. If a timeout occurs or an error is found, a negative value is returned.

INPUTS

ApiDevid - Integer containing the API Device ID of the device. This value can be from 1 to 31.

sCmnd - A pointer to buffer containing the command and binary data to send.

lCmnd - Integer containing the length of sCmnd.

sRetn - A pointer to store the returned binary data. This buffer must be long enough to hold the expected number of bytes.

lRetn - An integer containing the length of sRetn. This value must be the length or greater than the expected number of bytes returned or the IO_read will not return all the data from the specified device.

OUTPUTS

Returns SIA_SUCCESS upon completion or a negative value to indicate error.

`long __stdcall COMM_ReqInt ( long ApiDevId, char *cmnd, long *ival )`

This function first clears the response byte of the specified device and then sends the specified command and waits for the ESB or MAV bit in the response byte to be set or LONG_TIME (100) seconds. If the MAV bit is set, an IO_read is done and the returned ASCII data is converted to a long integer and placed in the specified location (ival). The ESR byte is then checked for errors. If a timeout occurs or an error is found, a negative value is returned.

INPUTS

ApiDevid - Integer containing the API Device ID of the device. This value can be from 1 to 31.

cmnd - A pointer to a NULL terminated ASCII string containing the command to send.

ival - A pointer to a long integer to store the returned value.

OUTPUTS

Returns SIA_SUCCESS upon completion or a negative value to indicate error.

`long __stdcall COMM_ReqDbl ( long ApiDevId, char *cmnd, double *dval )`

This function first clears the response byte of the specified device and then sends the specified command and waits for the ESB or MAV bit in the response byte to be set or LONG_TIME (100) seconds. If the MAV bit is set, an IO_read is done and the returned ASCII data is converted to a double and placed in the specified location (dval). The ESR byte is then checked for errors. If a timeout occurs or an error is found, a negative value is returned.

INPUTS

ApiDevid - Integer containing the API Device ID of the device. This value can be from 1 to 31.

cmnd - A pointer to a NULL terminated ASCII string containing the command to send.

dval - A pointer to a double to store the returned value.

OUTPUTS

Returns SIA_SUCCESS upon completion or a negative value to indicate error.

long __stdcall COMM_PollUntilTrue (long ApiDevId, long mask, time_t tyme)

This function will poll the response byte of the specified device until one of the specified bits becomes true or the specified number seconds elapses.

INPUTS

ApiDevid - Integer containing the API Device ID of the device. This value can be from 1 to 31.

mask - Integer containing the response bits to wait for. Refer to the device documentation for definition of the response bits.

tyme - Integer containing the amount of time in seconds to wait for one of the specified response bits to become true.

OUTPUTS

Returns an integer containing the response byte from ApiDevId (Refer to the device documentation for definition of the response bits.) or a negative value to indicate error.

long __stdcall COMM_PollWhileTrue (long ApiDevId, long mask, time_t tyme)

This function will poll the response byte of the specified device until one of the specified bits becomes true or the specified number seconds elapses.

INPUTS

ApiDevid - Integer containing the API Device ID of the device. This value can be from 1 to 31.

mask - Integer containing the response bits to wait for. Refer to the device documentation for definition of the response bits.

tyme - Integer containing the amount of time in seconds to wait for one of the specified response bits to become true.

OUTPUTS

Returns an integer containing the response byte from ApiDevId (Refer to the device documentation for definition of the response bits.) or a negative value to indicate error.

long __stdcall COMM_PollWithStatUntilTrue (long ApiDevId, long mask, time_t tyme)

This function will poll both the status byte and the response byte of the specified device while all of the specified bits are clear, or the specified number of seconds elapses.

INPUTS

ApiDevid - Integer containing the API Device ID of the device. This value can be from 1 to 31.

mask - Integer containing the response bits to wait for. Refer to the device documentation for definition of the response bits.

tyme - Integer containing the amount of time in seconds to wait for one of the specified response bits to become true.

OUTPUTS

Returns an integer containing the response byte from ApiDevId (Refer to the device documentation for definition of the response bits.) or a negative value to indicate error.

long __stdcall COMM_ClearRspByt (long ApiDevId)

If any of the status indicators are set on the specified device, this function send a "*cls" command to the specified device and waits for the response byte to clear or SHORT_TIME (5) seconds. If the function times out, an error is returned.

INPUTS

ApiDevid - Integer containing the API Device ID of the device. This value can be from 1 to 31.

OUTPUTS

Returns an integer containing the response byte from ApiDevId (Refer to the device documentation for definition of the response bits.) or a negative value to indicate error.

long __stdcall COMM_ReqEsr (long ApiDevId, char *esr)

This function sends a "*esr?" command and waits for the byte to return or SHORT_TIME (5) seconds. If the function times out, an error is returned.

INPUTS

ApiDevid - Integer containing the API Device ID of the device. This value can be from 1 to 31.

esr - A character pointer to the location to store the esr byte.

OUTPUTS

Returns SIA_SUCCESS upon completion or a negative value to indicate error.

long __stdcall COMM_DevChans (long ApiDevId)

INPUTS

ApiDevid - Integer containing the API Device ID of the device. This value can be from 1 to 31. The device must have been opened using COMM_InitDev(..).

OUTPUTS

Returns the number of channels installed in the specified device or a negative number to indicate error.

long __stdcall COMM_DevMarkers (long ApiDevId)

INPUTS

ApiDevid - Integer containing the API Device ID of the device. This value can be from 1 to 31. The device must have been opened using COMM_InitDev(..).

OUTPUTS

Returns the number of pattern markers installed in the specified device or a negative number to indicate error.

char * __stdcall COMM_DevIdn (long ApiDevId)

INPUTS

ApiDevid - Integer containing the API Device ID of the device. This value can be from 1 to 31. The device must have been opened using COMM_InitDev(..).

OUTPUTS

Returns a pointer to the IDN of the specified device or NULL to indicate error.

long __stdcall COMM_GetApiDevId (char *devname, long *ApiDevId)

INPUTS

ApiDevid - Integer pointer to location to return ApiDevId.

devname - A pointer to an ASCII string containing a device name.

OUTPUTS

Returns SIA_SUCCESS upon completion or a negative value to indicate error.

long __stdcall COMM_GetDevName (long ApiDevId, char *devname)

INPUTS

ApiDevid - Integer containing the API Device ID of the device. This value can be from 1 to 31. The device must have been opened using COMM_InitDev(..).

devname - A pointer to location to return device name.

OUTPUTS

Returns SIA_SUCCESS upon completion or a negative value to indicate error.

long __stdcall COMM_ReadFile (long ApiDevId, const char *srcFilename, const char *destFilename, long lFileSize)

Use this function to read back a file from the SIA3000 and save the contents in a specified file on the host.

INPUTS

ApiDevid - Integer containing the API Device ID of the device. This value can be from 1 to 31. The device must have been opened using COMM_InitDev(..).

srcFilename - A pointer to the name of the file to read data from. This file is located on the SIA3000 hard drive.

destFilename - A pointer to the location of the file the data will be saved to.

lFileSize - The known size (in bytes) of the file being read in.

OUTPUTS

Returns SIA_SUCCESS upon completion or a negative value to indicate error.

long __stdcall COMM_SendFile (long ApiDevId, const char *filename)

Use this function to send a file to the SIA3000.

INPUTS

ApiDevid - Integer containing the API Device ID of the device. This value can be from 1 to 31. The device must have been opened using COMM_InitDev(..).

filename - A pointer to the name of the file whose contents will be saved to the SIA3000 in a file with the same name.

OUTPUTS

Returns SIA_SUCCESS upon completion or a negative value to indicate error.

long __stdcall COMM_InitSingleShot (long ApiDevId, char *sCmnd, long lCmnd)

Use this function to configure a device specified by ApiDevId to perform a "Single Shot" measurement. If a timeout occurs or an error occurs, a negative value is returned.

INPUTS

ApiDevid - Integer containing the API Device ID of the device. This value can be from 1 to 31.

sCmnd - A pointer to buffer containing the command and binary data to send.

lCmnd - Integer containing the length of sCmnd.

OUTPUTS

Returns SIA_SUCCESS upon completion or a negative value to indicate error.

`long __stdcall COMM_ReqSingleShot ( long ApiDevId, char *sRetn, long *lRetn )`

Use this function to read the results of the "Single Shot" measurement requested by COMM_InitSingleShot for the device specified by ApiDevId. If result exists, the returned binary data is placed in the location specified by sRetn. If a timeout occurs or an error is found, a negative value is returned.

INPUTS

ApiDevid - Integer containing the API Device ID of the device. This value can be from 1 to 31.

sRetn - A pointer to store the returned binary data. This buffer must be long enough to hold the expected number of bytes.

lRetn - An integer containing the length of sRetn. This value must be the length or greater than the expected number of bytes returned or the IO_read will not return all the data from the specified device.

OUTPUTS

Returns SIA_SUCCESS upon completion or a negative value to indicate error.

`void __stdcall COMM_libver (char *StrPtr)`

This function returns the current COMM library versions (i.e. "2.5.0").

INPUTS

Strptr - Pointer to location to store version string. Memory must be allocated by user.

OUTPUTS

None.

I/O Layer Functions

NOTE: These functions can be used to control other devices using the same I/O protocol (GPIB, HPIB or Custom).

`void __stdcall IO_clear ( int ApiDevid )`

Clears the internal or device functions of the specified device.

INPUTS

ApiDevid - Integer containing the API Device ID of the device. This value can be from 1 to 31. The device must have been opened using IO_open (...).

OUTPUTS

None.

`void __stdcall IO_close ( int ApiDevid )`

Closes the device specified by ApiDevid.

INPUTS

ApiDevid - Integer containing the API Device ID of the device. This value can be from 1 to 31. The device must have been opened using IO_open (...).

OUTPUTS

None.

`int __stdcall IO_count (int ApiDevid)`

This function returns the byte count of the last data transfer operation.

INPUTS

ApiDevTyp - Integer containing the API Device ID of the device. This value can be from 1 to 31.

OUTPUTS

Returns an integer containing the byte count of the last data transfer operation.

`void __stdcall IO_libver (char *StrPtr)`

This function returns the current IO library version (i.e. "2.5.0").

INPUTS

Strptr - Pointer to location to store version string. Memory must be allocated by user.

OUTPUTS

None.

`int __stdcall IO_open ( int ApiDevTyp, char *devname )`

INPUTS

ApiDevType - An integer value indicating the device type:
HPIB = 0 (HP Systems Only)
GPIB = 1
CUST1 = 11
CUST2 = 12
CUST3 = 13

devname - A pointer to an ASCII string containing a device name.

OUTPUTS

Returns an integer containing the API Device ID or a negative number to indicate an error. Opens the device specified by devname and returns the API Device ID. If an error occurs, a negative number is returned instead.

`int __stdcall IO_read ( int ApiDevid, void *buf, long cnt )`

Read a maximum of cnt bytes from ApiDevid and place it in buf. Use IO_count() to check actual number of bytes read.

INPUTS

ApiDevid - Integer containing the API Device ID of the device. This value can be from 1 to 31. The device must have been opened using IO_open (...).

buf - Location to place data read. Must be at least cnt long.

cnt - Number of bytes to try and read.

OUTPUTS

Returns an integer containing the status of the last I/O operation.

`int __stdcall IO_response ( int ApiDevid, char *rsp )`

Get response byte from ApiDevid and place it in rsp. Refer to the device documentation for definition of the response bits.

INPUTS

ApiDevid - Integer containing the API Device ID of the device. This value can be from 1 to 31. The device must have been opened using IO_open (...).

rsp - Location to place response byte.

OUTPUTS

Returns an integer containing the status of the last I/O operation.

`int __stdcall IO_status (int ApiDevid)`

This function returns the status of the last I/O operation. Fourteen bits within the status word are meaningful. Three are used throughout the API:

ERROR - bit 15, hex value 8000, Error detected

TIMEO - bit 14, hex value 4000, Time out

END - bit 13, hex value 2000, END detected.

INPUTS

ApiDevid - Integer containing the API Device ID of the device. This value can be from 1 to 31.

OUTPUTS

Returns an integer containing the status of the last I/O operation.

`void __stdcall IO_trigger ( int ApiDevid )`

Sends a device trigger to the specified device.

INPUTS

ApiDevid - Integer containing the API Device ID of the device. This value can be from 1 to 31. The device must have been opened using IO_open (...).

OUTPUTS

None.

`int __stdcall IO_write ( int ApiDevid, void *buf, long cnt )`

Write cnt bytes from buf to ApiDevid. Use IO_count() to check actual number of bytes written.

INPUTS

ApiDevid - Integer containing the API Device ID of the device. This value can be from 1 to 31. The device must have been opened using IO_open (...).

buf - Location of data to write.

cnt - Number of bytes to write.

OUTPUTS

Returns an integer containing the status of the last I/O operation.

3-2 MEASUREMENT UTILITY FUNCTIONS

The following functions perform actions that will prepare a configured measurement for execution by setting thresholds or timing values based on detection algorithms.

`long __stdcall FCNL_CalcArmDelay ( double dFreq, PARM *tParm )`

This function calculates the Arm Delay for a given input frequency If a math error occurs or an error is found, a negative value is returned.

INPUTS

dFreq - The current test frequency in MHz.

tParm - A pointer to the PARM structure.

OUTPUTS

Returns SIA_SUCCESS upon completion or a negative value to indicate error.

`long __stdcall FCNL_PulsFnd (long ApiDevId, PARM *tParm )`

This function is used to perform a pulse-find operation. The pulse-find feature determines minimum and maximum voltage levels for the channels specified in the PARM structure and sets the voltage thresholds based on the percentage set in the tParm.lFndPcnt field supplied in the PARM structure.

INPUTS

ApiDevid - Integer containing the API Device ID of the device. This value can be from 1 to 31.

tParm - A pointer to the PARM structure.

OUTPUTS

Returns SIA_SUCCESS upon successful completion or a specific error code (negative value) indicating what type of error occurred.

3-3 PATTERN AND PM50 FUNCTIONS

The following functions are related to configuration of patterns. The first function, `FCNL_PtnName()`, will set the pattern file name within a measurement window structure of any tool that requires it. All other functions are related to the configuration of a PM50 to generate a pattern marker.

long __stdcall FCNL_PtnName (char sPtnName[], char *name)

This function is used to load the pattern file name into the required measurement structure. This function is included to assist when programming in Microsoft Visual Basic. When programming in C, it is best to use a sprintf() command to write a character string into the structure element associated with the pattern name.

INPUTS

sPtnName - Location where pattern name will be updated. Memory needs to be allocated by the caller.

name - Name of pattern file to load into measurement structure.

OUTPUTS

Returns SIA_SUCCESS if operation is successful or a negative value to indicate error. Error codes are defined in Appendix B.

EXAMPLE

```
memset(&dcom,0,sizeof(DCOM));                 //allocate memory space for dcom structure
FCNL_DefDcom(&dcom);                          //set dcom structure to defaults
FCNL_Ptn(&dcom.sPtnName[0], "cjtpat.ptn");    //load cjtpat.ptn file into dcom's pattern
                                              //name element.  This command could be
                                              //replaced with a sprintf command when
                                              //programming in C.
```

long __stdcall FCNL_MarkerInit (long ApiDevId, long MarkerId, PMKR *tPmkr)

Use this function to initialize the specified PM50. This must be called before using a PM50 in any application.

INPUTS

ApiDevid - Contains the API Device ID of the device. This value can be from 1 to 31.

MarkerId - Which PM50 card in the system to initialize

tPmkr - Pointer to a PM50 PMKR measurement and control structure

OUTPUTS

Returns SIA_SUCCESS upon successful completion or a specific error code (negative value) indicating what type of error occurred.

long __stdcall FCNL_MarkerReset (long ApiDevId, PMKR *tPmkr)

Use this function to reset the state of the specified PM50.

INPUTS

ApiDevid - Contains the API Device ID of the device. This value can be from 1 to 31.

tPmkr - Pointer to a PM50 PMKR measurement and control structure.

OUTPUTS

Returns SIA_SUCCESS upon successful completion or a specific error code negative value) indicating what type of error occurred.

long __stdcall FCNL_MarkerConfig (long ApiDevId, PMKR *tPmkr)

Use this function to change the configuration of the PM50 specified.

INPUTS

ApiDevid - Contains the API Device ID of the device. This value can be from 1 to 31.

tPmkr - Pointer to a PM50 PMKR measurement and control structure.

OUTPUTS

Returns SIA_SUCCESS upon successful completion or a specific error code (negative value) indicating what type of error occurred.

long __stdcall FCNL_MarkerStatus (long ApiDevId, PMKR *tPmkr)

Use this function to monitor the current state of the specified PM50.

INPUTS

ApiDevid - Contains the API Device ID of the device. This value can be from 1 to 31.

tPmkr - Pointer to a PM50 PMKR measurement and control structure.

OUTPUTS

Returns a value > 0 to indicate the presence of an arming condition on the specified PM50 or an error code (negative value) indicating what type of error occurred.

long __stdcall FCNL_MarkerReadErr (long ApiDevId, PMKR *tPmkr)

Use this function to read bit errors recorded by the specified PM50.

INPUTS

ApiDevid - Contains the API Device ID of the device. This value can be from 1 to 31.

tPmkr - Pointer to a PM50 PMKR measurement and control structure.

OUTPUTS

Returns SIA_SUCCESS upon successful completion or a specific error code (negative value) indicating what type of error occurred.

long __stdcall FCNL_PatternMatch (long ApiDevId, PARM *tParm, const char *filename, double *dBits, long lSize)

Use this function to perform a functional pattern match test and report results to the user.

INPUTS

ApiDevid - Contains the API Device ID of the device. This value can be from 1 to 31.

tParm - Pointer to a PARM acquisition structure.

filename - Name of pattern file to be used for comparison purposes.

dBits - Pointer to a array representing each bit in a pattern.

lSize - Size of array representing each bit in a pattern.

OUTPUTS

Returns SIA_SUCCESS upon successful completion or a specific error code (negative value) indicating what type of error occurred.

3-4 CALIBRATION UTILITY FUNCTIONS

long __stdcall FCNL_GoReq (long ApiDevId, long (*CallBackFunc) (long ApiDevID, char *Prompt), char *Prompt)

This is an internal function used by the calibration functions to allow the programmer to physically change the connections to the instrument either through a matrix or manually with cables. This function requires that a process be running on the SIA3000 which has paused operation and sent a RQC_BIT back. At present, the only functions doing this are the calibration routines. Future expansion of This function waits for the RQC_BIT to be set then sends a :SYST:GO or SYST:NOGO to the specified device based on the return value of CallBackFunc. Only the calibration commands have the ability to set RQC_BIT and monitor :SYST:GO and SYST:NOGO.

INPUTS

ApiDevid - Integer containing the API Device ID of the device. This value can be from 1 to 31.

CallBackFunc - A pointer to a function to call to determine if a :SYST:GO (continue) or :SYST:NOGO (skip) command should be sent to the device (see functional description below). CallBackFunc can be NULL or it must follow these rules:

long CallBackFunc (long ApiDevID, char *Prompt)

It must return an integer value of...
... >0 Send :SYST:GO to device
... 0 Send :SYST:NOGO to device

ApiDevid - Integer containing the API Device ID of the device. This value can be from 1 to 31.

Prompt - A pointer to a string prompt generally specifying what an operator needs to do before the :SYST:GO or :SYST:NOGO command should be sent to the device.

Prompt - A pointer to a string prompt that will be passed to the CallBackFunction generally specifying what an operator needs to do before the :SYST:GO or :SYST:NOGO command should be sent to the device (see functional description below). This parameter is simply passed through and is not checked for NULL.

OUTPUTS

Returns SIA_SUCCESS upon completion or a negative value to indicate error/abort.

EXAMPLE

```
long (*CallBackFunc)(long ApiDevId, char *prompt);  //Declare CalBackFunc to be a pointer to a
                                                    //function with two parameters passed in.
static long ConChan(long ApiDevId, char * Prompt);  //Declare ConChan to be function with two
                                                    //parameters passed in.
main()
  {
  char userPrompt[256];                             //declare userPrompt to be a string of 256
                                                    //characters in length.

  CallBackFunc = ConChan;                           //Let *CallBackFunc() point to ConChan()
  strcpy(userPrompt, "Connect CH1");                //Define userPrompt string.
  FCNL_GoReq (ApiDevId,*CallBackFunc,inpPrompt);    //continue execution of paused calibration
                                                    //command after ConChan is executed and
                                                    //ConChan has returned a 1.
  }
ConChan (long ApiDevID, char *Prompt)               //User defined function that prompts the
  {                                                 //user to "Connect CH1" as defined by the
  char buf[10];                                     //calling function above.

  printf("Ready to execute.  Please %s\n",Prompt);  //display string passed from FCNL_GoReq
  gets(buf);                                        //pause execution until enter key is pressed.
  return (1);                                       //return a value of 1 to allow SIA3000 to
  }                                                 //proceed with calibration routine.
```

long __stdcall FCNL_CalInt (long ApiDevId, long Multiplier)

The internal calibration function will process 10,000,000 samples multiplied by Multiplier, taking 5.5 minutes/10,000,000 samples to complete.

INPUTS

ApiDevid - Integer containing the API Device ID of the device. This value can be from 1 to 31.

Multiplier - Integer containing a value 1 - MAX_CAL_MULT. The selected multiplier, from 1 - MAX_CAL_MULT, sets the calibration period of approximately 5.5 minutes by that factor.

OUTPUTS

Returns SIA_SUCCESS upon completion or a negative value to indicate error.

long __stdcall FCNL_CalDeskew (long ApiDevId, long (*CallBackFunc) (long ApiDevID, char *prompt))

This function will calibrate all the channels installed in the device according to the following conditions determined by the CallBackFunc function:

…If the return value is > 0, the current channel is calibrated.
…If the return value is 0, the current channel is skipped.

INPUTS

ApiDevid - Integer containing the API Device ID of the device. This value can be from 1 to 31.

CallBackFunc - A pointer to a function to call to determine if the channel should be calibrated or skip the channel (see functional description below). CallBackFunc cannot be NULL. It must follow these rules:

long CallBackFunc (long ApiDevID, char *Prompt)

It must return an integer value of...
... >0 Continue with this channel
... 0 Skip this channel

ApiDevid - Integer containing the API Device ID of the device. This value can be from 1 to 31.

Prompt - A pointer to a string prompt generally specifying what an operator needs to do before calibrating the channel.

OUTPUTS

Returns SIA_SUCCESS upon completion or a negative value to indicate error/abort.

Deskew (external) calibration without DC Calibration.

long __stdcall FCNL_CalDeskewDc (long ApiDevId, long (*CallBackFunc) (long ApiDevID, char *prompt))

This function will calibrate all the channels installed in the device according to the following conditions determined by the CallBackFunc function.

...If the return value is > 0, the current channel is calibrated.
...If the return value is 0, the current channel is skipped.

INPUTS

ApiDevid -Integer containing the API Device ID of the device. This value can be from 1 to 31.

CallBackFunc -A pointer to a function to call to determine if the channel should be calibrated or skip the channel (see functional description below). CallBackFunc cannot be NULL. It must follow these rules:

long CallBackFunc (long ApiDevID, char *Prompt)

It must return an integer value of...
... >0 Continue with this channel
... 0 Skip this channel

ApiDevid - Integer containing the API Device ID of the device. This value can be from 1 to 31.

Prompt - A pointer to a string prompt generally specifying what an operator needs to do before calibrating the channel.

OUTPUTS

Returns SIA_SUCCESS upon completion or a negative value to indicate error/abort.

Deskew (external) calibration with DC Calibration.

long __stdcall FCNL_CalStrobe (long ApiDevId)

The strobe calibrarion funtion does an Oscilloscope Strobe calibration.

INPUTS

ApiDevid - Integer containing the API Device ID of the device. This value can be from 1 to 31.

OUTPUTS

Returns SIA_SUCCESS upon completion or a negative value to indicate error.

long __stdcall FCNL_GetCalData(long ApiDevId, long *pChannelCards, double *pDeSkewData)

Use this function to obtain the current external deskew values for all available channels in the device.

INPUTS

ApiDevid - Integer containing the API Device ID of the device. This value can be from 1 to 31.

pChannelCards - Upon successful completion, pointer to variable containing the number of channels in the device

pDeSkewData - Upon successful completion, pointer to an array containing a deskew value for each channel

OUTPUTS

Returns SIA_SUCCESS upon completion or a negative value to indicate error.

long __stdcall FCNL_SetCalData(long ApiDevId, long ChannelCards, double *pDeSkewData)

Use this function to update the current external deskew values for the number of channels specified in the device.

INPUTS

ApiDevid - Integer containing the API Device ID of the device. This value can be from 1 to 31.

ChannelCards - Number of channels to set external deskew values for

pDeSkewData - Pointer to an zero-based indexed array containing the desired deskew values for each channel

OUTPUTS

Returns SIA_SUCCESS upon completion or a negative value to indicate error.

3-5 SIGNAL PATH FUNCTIONS (DSM16, PATH MAPPING AND PATH DESKEW)

***NOTE**: *MuxAddr (1 thru 8) is assigned based on the RS232C output connectors on the USB-to-RS232C interface module.*

`long __stdcall FCNL_DSM16Switch ( long ApiDevId, long MuxAddr, long switch_ON_OFF )`

Use this function to enable or disable the DSM connected to the device specified in ApiDevId.

INPUTS

ApiDevid - Integer containing the API Device ID of the device. This value can be from 1 to 31.

MuxAddr - An integer address identifying DSM to select. The range is 1 to 8, based on information in above note.

switch_ON_OFF - An integer with value 0 to disable the DSM16 front panel buttons and any non zero value or 1 to enable.

OUTPUTS

Returns SIA_SUCCESS upon completion or a negative value to indicate error.

`long __stdcall FCNL_DSM16Ver ( long ApiDevId, long MuxAddr, char *outbuf )`

Use this function to determine the revision level of the DSM connected to the device specified in ApiDevId

INPUTS

ApiDevid - Integer containing the API Device ID of the device. This value can be from 1 to 31.

MuxAddr - An integer address identifying DSM to select. The range is 1 to 8, based on information in above note.

OUTPUT PARAMETER

outbuf - A pointer to a character array which will be filled with the revision level.

OUTPUTS

Returns SIA_SUCCESS upon completion or a negative value to indicate error.

`long __stdcall FCNL_DSM16GetSwitchNumbers ( long ApiDevId, long MuxAddr, char *switchNums )`

Use this function to determine the current configuration of the DSM connected to the device specified in ApiDevId.

INPUTS

ApiDevid - Integer containing the API Device ID of the device. This value can be from 1 to 31.

MuxAddr - An integer address identifying DSM to select. The range is 1 to 8, based on information in above note.

OUTPUT PARAMETER

switchNum - A pointer to a character array which will be filled with the switch numbers currently active in the banks.

OUTPUTS

Returns SIA_SUCCESS upon completion or a negative value to indicate error.

long __stdcall FCNL_DSM16SetSwitchNumber (long ApiDevId, long MuxAddr, long switchNum)

Use this function to reconfigure the switch settings of the DSM connected to the device specified in ApiDevId.

INPUTS

ApiDevid - Integer containing the API Device ID of the device. This value can be from 1 to 31.

MuxAddr - An integer address identifying DSM to select. The range is 1 to 8, based on information in above note.

SwitchNum - Integer containing the switch number to activate. The range for the relays is : 11 to 18 for bank 1 : 21 to 28 for bank 2.

OUTPUTS

Returns SIA_SUCCESS upon completion or a negative value to indicate error.

long __stdcall FCNL_DefPathMap (long path, long DevType, char *DevName, long Channel, long MuxSwitch, long MuxIsADsm)

Use this function to map an unique path (pin number) to an individual channel on a particular device. This function will initialize the device if this had not been done previously.

INPUTS

Path - Number of the path being defined. This value can be from 0 to 511.

DevType - Number that indicates the device type:

HPIB = 0 (HP Systems Only)
GPIB = 1
CUST1 = 11
CUST2 = 12
CUST3 = 13

DevName - A pointer to an ASCII string containing a device name

Channel - A valid SIA channel of the device named in DevName above

MuxSwitch - A flag indicating if an external MUX is included in path.

MuxIsADsm - A flag indicating if a DSM is included in this path.

OUTPUTS

Returns SIA_SUCCESS upon completion or a negative value to indicate error.

long __stdcall FCNL_DefPathDutDeskew (long path, double value)

Use this function to set the external deskew value for the DUT path indicated.

INPUTS

path - Number of the path being defined. This value can be from 0 to 511.

value - DUT Deskew value for this path

OUTPUTS

Returns SIA_SUCCESS upon completion or a negative value to indicate error.

long __stdcall FCNL_DefPathFixDeskew (long path, double value)

Use this function to set the external deskew value for the fixture path indicated.

INPUTS

path - Number of the path being defined. This value can be from 0 to 511.

value - Fixture Deskew value for this path

OUTPUTS

Returns SIA_SUCCESS upon completion or a negative value to indicate error.

long __stdcall FCNL_GetPathDevName (long path, char *DevName)

Use this function to retrieve the device name for the path indicated.

The path must have been defined previously using FCNL_DefPathMap(..).

INPUTS

path - Number of the path being defined. This value can be from 0 to 511.

DevName - A pointer to an ASCII string containing a device name

OUTPUTS

Returns SIA_SUCCESS upon completion or a negative value to indicate error.

long __stdcall FCNL_GetPathDevType (long path, long *DevType)

Use this function to retrieve the device type for the path indicated.

The path must have been defined previously using FCNL_DefPathMap(..).

INPUTS

path - Number of the path being defined. This value can be from 0 to 511.

DevType - Pointer to location that returns the device type:

HPIB = 0 (HP Systems Only)

GPIB = 1

CUST1 = 11

CUST2 = 12

CUST3 = 13

OUTPUTS

Returns SIA_SUCCESS upon completion or a negative value to indicate error.

long __stdcall FCNL_GetPathApiDevId (long path, long *ApiDevId)

Use this function to retrieve the device id for the path indicated.

The path must have been defined previously using FCNL_DefPathMap(..).

INPUTS

path - Number of the path being defined. This value can be from 0 to 511.

DevType - Pointer to location that returns the ApiDevId (a value between 1 and 31)

OUTPUTS

Returns SIA_SUCCESS upon completion or a negative value to indicate error.

long __stdcall FCNL_GetPathChannel (long path, long *Channel)

Use this function to retrieve the channel for the path indicated.

The path must have been defined previously using FCNL_DefPathMap(..).

INPUTS

path - Number of the path being defined. This value can be from 0 to 511.

Channel - Pointer to location that returns the device channel

OUTPUTS

Returns SIA_SUCCESS upon completion or a negative value to indicate error.

long __stdcall FCNL_GetPathMuxSwitch (long path, long *MuxSwitch)

Use this function to indicate the MUX switch index for the path indicated. The path must have been defined previously using FCNL_DefPathMap(..).

INPUTS

path - Number of the path being defined. This value can be from 0 to 511.

MuxSwitch - Pointer to location that returns the MUX switch index

OUTPUTS

Returns SIA_SUCCESS upon completion or a negative value to indicate error.

long __stdcall FCNL_GetPathMuxIsADsm (long path, long *MuxIsADsm)

Use this function to inquire if a DSM is being used as a MUX in this path indicated. The path must have been defined previously using FCNL_DefPathMap(..).

INPUTS

path - Number of the path being defined. This value can be from 0 to 511.

MuxIsADsm - Pointer to location that indicates if a DSM is being used as a MUX in this path.

OUTPUTS

Returns SIA_SUCCESS upon completion or a negative value to indicate error.

long __stdcall FCNL_GetPathDutDeskew (long path, double *value)

Use this function to retrieve the external deskew value for the DUT path indicated. The path must have been defined previously using FCNL_DefPathMap(..).

INPUTS

path - Number of the path being defined. This value can be from 0 to 511.

value - Pointer to location containing the DUT Deskew value for the path indicated

OUTPUTS

Returns SIA_SUCCESS upon completion or a negative value to indicate error.

long __stdcall FCNL_GetPathFixDeskew (long path, double *value)

Use this function to retrieve the external deskew value for the fixture path indicated. The path must have been defined previously using FCNL_DefPathMap(..).

INPUTS

path - Number of the path being defined. This value can be from 0 to 511.

value - Pointer to location containing the fixture Deskew value for the path indicated

OUTPUTS

Returns SIA_SUCCESS upon completion or a negative value to indicate error.

3-6 MISCELLANEOUS RESULT AND STATUS FUNCTIONS

double __stdcall FCNL_GetXval (PLTD *plot, long indx)

This function is used to simplify the process of extracting X-axis information from a PLTD structure. In order to reduce memory requirements, only Y-axis values are contained within PLTD structures. This is permissible since X-axis values represent the independent variable. This function uses the same method for calculating the X-axis values based on the elements in the measurement structure. Results are only valid after a successful call to `FCNL_RqstAll`, `FCNL_MultPkt`, or `FCNL_GrpGetPkt`.

INPUTS

*plot - Pointer to a PLTD structure. Memory needs to be allocated by the caller. This pointer will be the PLTD structure pointer used in the measurement command of interest.

indx - Index from which to determine X-value, range is (0 to tPlot.lNumb - 10).

OUTPUTS

The value is double of the x coordinate.

EXAMPLE

```
FCNL_RqstAll ( ApiDevId, &hist, WIND_HIST );      //execute a histogram based on settings in
                                                  //hist structure as defined in preceding lines
val = FCNL_GetXval( &hist.tAcum, inpIndx);        //get x-value of Accumulated Jitter Plot
                                                  //inpIndx number of units from the
                                                  //minimum x value.
printf("Plot value of hist.tAcum for index %d = %2.18lf ns\n", inpIndx, val*1e9);
```

double __stdcall FCNL_GetYval (PLTD *plot, long indx)

This function is used to simplify the process of extracting Y-axis information from a PLDT structure. It is primarily included to assist when programming in Microsoft Visual Basic. When programming in C, the data array can be accessed directly. This function is called after a successful execution of a measurement. The return value is the Y-value at an X-location offset from X-min by `indx`. Results are only valid after a successful call to `FCNL_RqstAll`, `FCNL_MultPkt`, or `FCNL_GrpGetPkt`.

INPUTS

plot - Pointer to a PLDT structure. Memory needs to be allocated by the caller.

indx - Index from which to determine Y-value, range is (0 to tPlot.lNumb - 10).

OUTPUTS

The value is double the y coordinate.

EXAMPLE

```
FCNL_RqstAll ( ApiDevId, &hist, WIND_HIST );      //execute a histogram based on settings in
                                                  //hist structure as defined in preceding lines
val = FCNL_GetYval( &hist.tAcum, inpIndx);        //get y-value of Accumulated Jitter Plot
                                                  //inpIndx number of units from the
                                                  //minimum y value.
printf("Plot value of hist.tAcum for index %d = %2.18lf ns\n", inpIndx, val*1e9);
```

`long __stdcall FCNL_Diagnostics ( long ApiDevId )`

Use this function to perform a system diagnostics test on the device. If any portion of the test fails, a negative value is returned.

INPUTS

ApiDevid - Integer containing the API Device ID of the device. This value can be from 1 to 31.

OUTPUTS

Returns SIA_SUCCESS upon completion or a negative value to indicate error.

`void __stdcall FCNL_libver (char *StrPtr)`

This function returns the current API version (i.e. "2.5.0").

INPUTS

Strptr - Pointer to location to store version level. Memory must be allocated by user.

OUTPUTS

None

3-7 ADVANCED GROUP MEASUREMENT FUNCTIONS

Grouping of commands provides an advanced Production API technique to further minimize GPIB bus traffic and set up complex sequences including multiple tools and/or channels. Once a group is established, it can be quickly and repeatedly executed.

If the fastest possible test time is desired, then these commands and programming techniques may be of use. Keep in mind that any measurement sequence can be accomplished through repeated calls to the standard FCNL_RqstPkt, FCNL_RqstALL, or FCNL_MultPkt functions. Since the measurement sequences are stored remotely on the GPIB host rather than the SIA-3000, the standard calls will require some GPIB bus overhead each time a measurement is taken. This overhead is reduced via grouping by storing all the measurement configuration information for the group locally on the SIA-3000 instrument.

Configuring a group involves the following steps:

1. Issue the `FCNL_GrpDefBeg` (groupNumber) command
2. Send down various measurement and configuration requests using `FCNL_GrpDefAsc` (command Syntax) or `FCNL_GrpDefPkt` (toolWindow, type, GetPlots?)
3. When finished defining a group, issue `FCNL_GrpDefEnd` (groupNumber)

Then issue:

`FCNL_GrpGetAll`

Then issue:

`FCNL_GrpGetAsc` (dataBuffer, expectedLength) or `FCNL_GrpGetPkt` (toolWindow, type, PlotsRetrieved?) in the same order the corresponding `FCNL_GrpDefAsc` and `FCNL_GrpDefPkt` were originally issued.

NOTE: Nesting of groups is not allowed.

CAUTION: DO NOT intersperse group definitions to multiple devices or call `FCNL_RqstPkt` or `FCNL_MultPkt` in the middle of a group definition. Unpredictable results will occur.

long __stdcall FCNL_GrpDefBeg (long nNumb)

Define a group; the group must be defined only once.

INPUTS

nNumb - Long Integer specifying the index of a group to be defined. A maximum of 20 groups are allowed at present.

OUTPUTS

Returns an integer 0 specifying a success or a negative value to indicate error.

long __stdcall FCNL_GrpDefAsc (char *sCmnd)

This function is for standard ASCII commands to be included in a group.

INPUTS

sCmnd - Pointer to a character array containing the ASCII command string to be used in the group. For the list of commands not allowed in groups please consult the manual.

OUTPUTS

Returns an integer 0 specifying a success or a negative value to indicate error.

long __stdcall FCNL_GrpDefPkt (void *pData, long nType, long bGetPlots)

This function is for setting up for getting data and/or plot values from a measurement like histogram, datacom etc within the scope of a group command.

If bGetPlots is non-zero memory is allocated for plot too and the binary structure will hold the binary plot data when executed later.

INPUTS

pData - Pointer to a data structure like HIST, DCOM etc to hold the input/output/plot values.

nType - Long Integer specifying the type of the request like: WIND_HIST, WIND_JITT etc.

bGetPlots - Long Integer specifying whether to get the plot data.

Zero - no plot data retrieved.

non- zero - get plot data.

OUTPUTS

Returns an integer 0 specifying a success or a negative value to indicate error.

long __stdcall FCNL_GrpDefEnd (long ApiDevId, long nNumb)

Finalize the group definition, for group specified in nNumb.

INPUTS

ApiDevid - Integer containing the API Device ID of the device. This value can be from 1 to 31.

nNumb - Long Integer specifying the index of a group to be defined. A maximum of 20 groups are allowed at present.

OUTPUTS

Returns an integer 0 specifying a success or a negative value to indicate error.

long __stdcall FCNL_GrpGetAll (long ApiDevId, long nNumb)

This function does the measurements and gets the whole block of data.

INPUTS

ApiDevid - Integer containing the API Device ID of the device. This value can be from 1 to 31.

nNumb - Long Integer specifying the index of a group to be defined. A maximum of 10 groups is allowed at present.

OUTPUTS

Returns an integer 0 specifying a success or a negative value to indicate error.

long __stdcall FCNL_GrpGetAsc (void *sBuff, long nSize)

This function gets the ASCII data back corresponding to the FCNL_GrpDefAsc command in the sequence. Refer to the manual for the example program that lists the order in which the commands in a group are defined and used.

INPUTS

sBuff - Pointer to a void to store the ASCII string from this call. Memory to be allocated by the caller.

nSize - Long Integer specifying the number of bytes to fetch.

OUTPUTS

Returns an integer 0 specifying a success or a negative value to indicate error.

long __stdcall FCNL_GrpGetPkt (void *pData, long nType, long bGetPlots)

This function gets the data back corresponding to the FCNL_GrpDefPkt command in the sequence. Refer to the manual for the example program that lists the order in which the commands in a group are defined and used.

This command is mostly used for getting a single histogram/dataCOM etc. data back.

INPUTS

pData - Pointer to a data structure like HIST, DCOM etc to hold the input/output/plot values.

nType - Long Integer specifying the type of the request like: WIND_HIST, WIND_JITT etc.

bGetPlots - Long Integer specifying whether to get the plot data.

Zero - no plot data retrieved.

non- zero - get plot data.

OUTPUTS

Returns an integer 0 specifying a success or a negative value to indicate error.

EXAMPLE

The following example shows how to utilize the group functions together to define a measurement group, and acquire multiple passes of the group. This code is meant to replace Steps 6, 7, and 8 of the Sample.c example given in Section 1.7

STEP 1 – Define a Group

Up to 20 distinct command “groups” can be sent to the SIA-3000™, where any number of commands can be “grouped” together, sent down to the SIA-3000 and executed in the order they are received (“pseudo-parallel” mode). Controlling the SIA3000 with Command Groups significantly reduces any overhead associated with the remote driver (GPIB, HPIB). Refer to the sample program comments or the SIA3000 GPIB Programming Guide for further details regarding command groups.

```
/* Now define a group, the group must only be defined once */
/* There can be up to 20 different groups defined */
if ( ( retn = FCNL_GrpDefBeg ( 1 ) ) != SIA_SUCCESS)
  {
  printf("\nFCNL_GrpDefBeg() failed...\n");
  goto Error;
  }
/* You can have standard ascii commands included in a group */
if ( ( retn = FCNL_GrpDefAsc ( ":ACQ:RUN PER" ) ) != SIA_SUCCESS)
  {
  printf("\nFCNL_GrpDefAsc() failed...\n");
  goto Error;
  }
/* You can also retrieve blocks of binary data */
if ( ( retn = FCNL_GrpDefAsc ( ":MEAS:DATA?" ) ) != SIA_SUCCESS)
  {
  printf("\nFCNL_GrpDefAsc() failed...\n");
  goto Error;
  }
/* And you can also use the structure calls, the zero argument skips plots */
if ( ( retn = FCNL_GrpDefPkt ( &hist, WIND_HIST, 0 ) ) != SIA_SUCCESS)
  {
  printf("\nFCNL_GrpDefPkt() failed...\n");
  goto Error;
  }
/* Ascii & structure calls can be interspersed */
if ( ( retn = FCNL_GrpDefAsc ( ":ACQ:RUN PW+" ) ) != SIA_SUCCESS)
  {
  printf("\nFCNL_GrpDefAsc() failed...\n");
  goto Error;
  }
/* With this structure call, the 1 argument requests all the plot data */
if ( ( retn = FCNL_GrpDefPkt ( &jitt, WIND_JITT, 1 ) ) != SIA_SUCCESS)
  {
  printf("\nFCNL_GrpDefPkt() failed...\n");
  goto Error;
  }
if ( ( retn = FCNL_GrpDefPkt ( &dcom, WIND_DCOM, 1 ) ) != SIA_SUCCESS)
  {
```

```
  printf("\nFCNL_GrpDefPkt() failed...\n");
  goto Error;
  }
/* You can nest multiple ascii commands, but only the last should return data */
if ( ( retn = FCNL_GrpDefAsc ( ":ACQ:FUNC FREQ;:ACQ:COUN 1000;:ACQ:MEAS" ) )
          != SIA_SUCCESS)
  {
  printf("\nFCNL_GrpDefAsc() failed...\n");
  goto Error;
  }
/* Finalize the group definition, for group 1 */
if ( ( retn = FCNL_GrpDefEnd ( ApiDevId, 1 ) ) != SIA_SUCCESS)
  {
  printf("\nFCNL_GrpDefEnd() failed...\n");
  goto Error;
  }

/* The definition doesn't acquire anything; use WavGrpGetAll to acquire */
/* You can loop and re-use the same definition over and over again */
for ( loop = 0; loop < 10; loop++ )
  {
```

STEP 2 – Perform a Group Acquire and Print Results

When the function FCNL_GrpGetAll(deviceID, groupNumber) is called, the group of commands indicated by groupNumber is executed by the SIA-3000 and the measurement results are available to the user in the same order the corresponding measurement commands were defined in that particular group.

```
  /* WavGrpGetAll does the measurements and gets the whole block of data */
  if ( ( retn = FCNL_GrpGetAll ( ApiDevId, 1 ) ) != SIA_SUCCESS)
  {
  printf("\nFCNL_GrpGetAll() failed...\n");
  goto Error;
  }

  /* The following calls parse the individual results out of the group data */
  /* There must be a 1-to-1 correspondence between the definition and these calls */
  if ( ( retn = FCNL_GrpGetAsc ( per, sizeof ( per ) ) ) != SIA_SUCCESS)
  {
  printf("\nFCNL_GrpGetAsc() failed...\n");
  goto Error;
  }
  /* The same method is used for binary blocks from ascii requests */
  if ( ( retn = FCNL_GrpGetAsc ( data, sizeof ( data ) ) ) != SIA_SUCCESS)
  {
  printf("\nFCNL_GrpGetAsc() failed...\n");
  goto Error;
  }

  /* For structure calls, the bGetPlot argument must be the same as in the
     definition */
  if ( ( retn = FCNL_GrpGetPkt ( &hist, WIND_HIST, 0 ) ) != SIA_SUCCESS)
  {
  printf("\nFCNL_GrpGetPkt() failed...\n");
  goto Error;
  }
  if ( ( retn = FCNL_GrpGetAsc ( pw, sizeof ( pw ) ) ) != SIA_SUCCESS)
  {
  printf("\nFCNL_GrpGetAsc() failed...\n");
  goto Error;
  }

  /* If bGetPlot = 1, plots are returned; these can be BIG and will be slower!!! */
  if ( ( retn = FCNL_GrpGetPkt ( &jitt, WIND_JITT, 1 ) ) != SIA_SUCCESS)
  {
  printf("\nFCNL_GrpGetPkt() failed...\n");
  goto Error;
  }
  if ( ( retn = FCNL_GrpGetPkt ( &dcom, WIND_DCOM, 1 ) ) != SIA_SUCCESS)
  {
  printf("\nFCNL_GrpGetPkt() failed...\n");
  goto Error;
  }
  if ( ( retn = FCNL_GrpGetAsc ( freq, sizeof ( freq ) ) ) != SIA_SUCCESS)
  {
```

```
printf("\nFCNL_GrpGetAsc() failed...\n");
goto Error;
}

/* Print the simple ascii command results */
printf("Group Loop %i - Period Measurement: %s\n",      loop + 1, per);
printf("Group Loop %i - Pulsewidth Measurement: %s\n",  loop + 1, pw);
printf("Group Loop %i - Frequency Measurement: %s\n",    loop + 1, freq);
/* Print the start of the binary block of raw data */
printf("Group Loop %i - Raw Period Measurements: %lf, %lf, %lf, ...\n", loop + 1,
       data[0] * 1e9, data[1] * 1e9, data[2] * 1e9);

/* Print out some of the statistics from the HIST and JITT tool structures */
printf("Group Loop %i - Histogram Mean: %lfns\n", loop + 1, hist.dNormAvg * 1e9);
printf("Group Loop %i - Histogram Sdev: %lfps\n", loop + 1, hist.dNormSig * 1e12);
printf("Group Loop %i - 1Clock RJ: %lfps\n",  loop + 1, jitt.dRjit1Clk * 1e12);
printf("Group Loop %i - NClock RJ: %lfps\n",  loop + 1, jitt.dRjitNClk * 1e12);
/* Print the max of the FFT to show how data within a plot is accessed */
printf("Group Loop %i - NClock Plot Max: %lfps\n", loop + 1,
       jitt.tFftN.dData[ jitt.tFftN.lYmaxIndx ] * 1e12);

/* Print the dataCOM tool DJ & RJ values */
printf("Group Loop %i - dataCOM DJ: %lfps\n",  loop + 1, dcom.dDdjt * 1e12);
printf("Group Loop %i - dataCOM RJ: %lfps\n",  loop + 1, dcom.dRjit[0] * 1e12);
```

This page intentionally left blank.

The following code samples are provided in order to aid in getting started using the *WAVECREST* Production API. These code samples are provided for instructional purposes only.

4-1 MODIFYING WINDOW STRUCTURE PARAMETERS

The following code snippet shows how parameters pertaining to a high-level window structure may be modified.

```
/* Allocate window structure */
STAT tStat;

/* Zero out the structure, and initialize to defaults */
memset ( &tStat, 0, sizeof ( STAT ) );
FCNL_DefStat ( &tStat );

/* Change input parameters from default */
tStat.tParm.lFuncNum = FUNC_PW_P;  /* Function PW+ */
tStat.tParm.lChanNum = 2;          /* Channel 2    */
tStat.tParm.lAutoArm = ARM_EXTRN;  /* External Arm */
tStat.tParm.lStrtArm = 2;          /* Start Arm 2  */
tStat.tParm.lStopArm = 2;          /* Stop Arm 2   */
tStat.tParm.lSampCnt = 500;        /* Sample Size  */
tStat.tParm.lStopCnt = 11;         /* Stop Count   */
```

4-2 PERFORMING TAILFIT

The following code snippet shows how a tailfit can be performed in a Histogram Window. Note that it may take many passes for the tailfit to succeed. Therefore you may want to error if not successfully in a certain number of passes. Set the **lPass** parameter to 0 to start a new tailfit analysis.

```
/* Allocate window structure, and initialize to defaults */
HIST tHist;
memset ( &tHist, 0, sizeof ( HIST ) );
FCNL_DefHist ( &tHist );

/* Enable tailfit */
tHist.lTailFit = 1;

/* Loop until tailfit is successful */
while ( !tHist.tTfit.lGood )
  {
  if ( FCNL_RqstPkt(ApiDevId, tHist, WIND_HIST ( &tHist ) )
    goto ErrorHandler;
  }
```

4-3 DRAWING FROM A PLOT STRUCTURE

This code snippet shows how to draw from a plot structure. The example is for Microsoft® Visual C++, but can be modified for other platforms.

```
void DrawPlot ( CDC *pCdc,   // Pointer to device context.
                CRect *wind, // Window to draw within
                             // in device coordinates.
                PLDT *pldt,  // Source plot structure.
                double xmin, // Plot extents to use when
                double xmax, // drawing, this allows a
                double ymin, // margin to be added around
                double ymax )// plot or overlay of plots
  {                          // with differing extents.
  long i;
  double x, y;

  // First plot X point as a percent of window extents
  x =  ( pldt->dXmin - xmin ) / ( xmax - xmin );

  // First plot X point in device coordinates
  x = ( double ) ( wind->right - wind->left )
    * x + ( double ) wind->left;

  // First plot Y point as a percent of window extents
  y = ( pldt->dData[ 0 ] - ymin ) / ( ymax - ymin );

  // First plot Y point in device coordinates
  y = ( double ) ( wind->bottom - wind->top )
    * ( 1.0 - y ) + ( double ) wind->top;

  // Move current location to the first plot point
  pCdc->MoveTo ( ( int ) x, ( int ) y );

  for ( i = 1; i < pldt->lNumb; i++ )
    {
    // Calculate what the next X point is
    x = ( ( pldt->dXmax - pldt->dXmin ) * ( double ) i
      / ( double ) ( pldt->lNumb - 1 ) + pldt->dXmin );

    // This plot X point as a percent of window extents
    x =  ( x - xmin ) / ( xmax - xmin );

    // This plot X point in device coordinates
    x = ( double ) ( wind->right - wind->left )
      * x + ( double ) wind->left;

    // This plot Y point as a percent of window extents
    y = ( pldt->dData[ i ] - ymin ) / ( ymax - ymin );

    // This plot Y point in device coordinates
    y = ( double ) ( wind->bottom - wind->top )
      * ( 1.0 - y ) + ( double ) wind->top;

    // Draw line to this plot point
    pCdc->LineTo ( ( int ) x, ( int ) y );
    }
  }
```

4-4 PERFORMING A DATACOM MEASUREMENT

This code snippet shows how a dataCOM measurement can be taken. Error checking is performed at each step, and several acquisition parameters are overridden. A pulsefind is used to determine suitable voltage levels, and results are printed.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "WCcomm.h"
#include "WCfcnl.h"

long main()
{
   DCOM dcom;
   long ApiDevId, retn = 0;

   /* Initialize device */
   if ( ( ApiDevId = COMM_InitDev ( APIDEVTYPE, DEVICENAME ) ) < 1 )
   {
        fprintf(stderr, "\nUnable to initialize device\n");
        return -1;
   }

   /* Initialize structure to default values */
   memset ( &dcom, 0, sizeof ( DCOM ) );
   FCNL_DefDcom ( &dcom );

   /* Measure on Channel 1; External Arm using Channel 2 */
   dcom.tParm.lChanNum = 1;
   dcom.tParm.lAutoArm = ARM_EXTRN;
   dcom.tParm.lExtnArm = 2;

   /* Select the pattern to use */
   strcpy(&dcom[0].sPtnName, "crpat.ptn");

   /* Do not measure the Bit Rate; assign the Bit Rate to use */
   dcom.lGetRate = 0;
   dcom.dBitRate = 1.0625e9;

   /* Perform a pulsefind */
   if ( ( retn = FCNL_PulsFnd ( ApiDevId, &dcom.tParm ) ) != SIA_SUCCESS)
        goto Error;

   /* Acquire measurement and obtain all values */
   if ( ( retn = FCNL_RqstPkt ( ApiDevId, &dcom, WIND_DCOM ) ) != SIA_SUCCESS)
        goto Error;
   if ( ( retn = FCNL_RqstAll ( ApiDevId, &dcom, WIND_DCOM ) ) != SIA_SUCCESS)
        goto Error;

   /* Print out the dataCOM DJ, RJ and TJ values in picoseconds */
   printf("dataCOM DJ: %lf ps\n",  dcom.dDdjt * 1e12);
   printf("dataCOM RJ: %lf ps\n",  dcom.dRjit[0] * 1e12);
   printf("dataCOM TJ: %lf ps\n",  dcom.dTjit[0] * 1e12);

Error:
   /* Return an error message if we had a problem */
   if ( retn )
      printf ( "Return Code: %i\n", retn );

   /* Perform any cleanup and exit */
   FCNL_ClrDcom ( &dcom );
   COMM_CloseDev (ApiDevId);
   return retn;
}
```

4-5 USING A PM50 PATTERN MARKER IN A DATACOM MEASUREMENT

This example illustrates how to utilize a PM50 Pattern Marker in a dataCOM measurement to determine bit errors and the Bit Error Rate using the PM50's Bit Error Counter.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "WCcomm.h"
#include "WCfcnl.h"

/* Uncomment for SUNOS                                     */
/*#define SUNOS 1                                          */
#if (WIN32 || SUNOS || SOLARIS2)
#define APIDEVTYPE              GPIB_IO
#define DEVICENAME              "dev5"
#else
#if (HPUX)
#define APIDEVTYPE              HPIB_IO
#define DEVICENAME              "hpib,5"
#endif
#endif

#define PATN_WORD_SIZE   20

int main()
{
   DCOM dcom;
   PMKR pmkr;

   char buff[PATN_WORD_SIZE];
   long ApiDevId, MarkerId, indx, bitIndx, retn = 0;

   /* In this example, use the PM50 associated with Input 1 */
   MarkerId = 1;

   /* Initialize our DCOM structure */
   memset ( &dcom, 0, sizeof ( DCOM ) );
   FCNL_DefDcom ( &dcom );
   strcpy(&dcom.sPtnName[0], "cjtpat.ptn");
   dcom.tParm.lChanNum = 1;

   /* Request that the PM50 be used as the arm */
   dcom.tParm.lAutoArm = ARM_EXTRN;
   dcom.tParm.lExtnArm = 1;
   dcom.tParm.lCmdFlag |= CMD_PATNMARK;

   /* Initialize SIA3000 */
   if ( ( ApiDevId = COMM_InitDev ( APIDEVTYPE, DEVICENAME ) ) < 1 )
   {
      printf ( "Unable to initialize SIA3000.  Program terminated.\n" );
      goto Error;
   }
```

```
/* Initialize PM50 */
if ( ( retn = FCNL_MarkerInit ( ApiDevId, MarkerId, &pmkr ) ) != SIA_SUCCESS )
   printf ( "FCNL_MarkerInit:  Return Code: %i\n", retn );

printf( " - Wavecrest Production API - \n - Sample PM50 Application  -\n\n" );

/* PART I: Configure the PM50 for edge count mode */
pmkr.lModeSel = PMKR_EDGE_COUNT;
strcpy(&pmkr.sPtnName[0], "cjtpat.ptn");
if ( ( retn = FCNL_MarkerConfig ( ApiDevId, &pmkr ) ) != SIA_SUCCESS )
   printf ( "FCNL_MarkerConfig:  Return Code: %i\n", retn );

/* Is the PM50 detecting the pattern? */
if ( ( retn = FCNL_MarkerStatus ( ApiDevId, &pmkr ) ) <= 0 )
   printf ( "FCNL_MarkerStatus:  Return Code: %i\n", retn );

/* Perform a pulsefind before making a DCOM measurement */
if ( ( retn = FCNL_PulsFnd ( ApiDevId, &dcom.tParm ) ) != SIA_SUCCESS )
   printf ( "FCNL_PulsFnd:  Return Code: %i\n", retn );
if ( ( retn = FCNL_RqstPkt ( ApiDevId, &dcom, WIND_DCOM ) ) != SIA_SUCCESS )
{
   printf ( "FCNL_RqstPkt:  Return Code: %i\n", retn );
   goto Error;
}

/* Print the results */
printf( " Edge Count Mode\n\n" );
printf( "Pattern: %s\n",            pmkr.sPtnName );
printf( "Edge Count: %i\n",         pmkr.lEdgeCnt );
printf( "dataCOM DCD+ISI: %lfps\n", dcom.dDdjt    * 1e12 );
printf( "dataCOM RJ: %lfps\n\n",    dcom.dRjit[0] * 1e12 );

/* PART II: Configure the PM50 for pattern match mode @ 1.0625 GBit/s */
pmkr.lModeSel = PMKR_PATN_MATCH;
pmkr.lProtSel = PMKR_FC1X;
strcpy(&pmkr.sPtnName[0], "cjtpat.ptn");
if ( ( retn = FCNL_MarkerConfig ( ApiDevId, &pmkr ) ) != SIA_SUCCESS )
   printf ( "FCNL_MarkerConfig:  Return Code: %i\n", retn );

/* Is the PM50 detecting the pattern? */
if ( ( retn = FCNL_MarkerStatus ( ApiDevId, &pmkr ) ) <= 0 )
   printf ( "FCNL_MarkerStatus:  Return Code: %i\n", retn );

/* Perform a pulsefind before making a DCOM measurement */
if ( ( retn = FCNL_PulsFnd ( ApiDevId, &dcom.tParm ) ) != SIA_SUCCESS )
   printf ( "FCNL_PulsFnd:  Return Code: %i\n", retn );
if ( ( retn = FCNL_RqstPkt ( ApiDevId, &dcom, WIND_DCOM ) ) != SIA_SUCCESS )
{
   printf ( "FCNL_RqstPkt:  Return Code: %i\n", retn );
   goto Error;
}
```

```
   /* Print the results */
   printf( " Pattern Match Mode\n\n" );
   printf( "Pattern: %s\n",            dcom.sPtnName );
   printf( "dataCOM DCD+ISI: %lfps\n", dcom.dDdjt    * 1e12 );
   printf( "dataCOM RJ: %lfps\n\n",    dcom.dRjit[0] * 1e12 );

   /* Did the PM50 detect any bit errors? */
   if ( ( retn = FCNL_MarkerReadErr ( ApiDevId, &pmkr ) ) != SIA_SUCCESS )
   {
      printf ( "FCNL_MarkerReadErr:  Return Code: %i\n", retn );
      goto Error;
   }

   /* Print the Bit Error Counter results */
   printf ( "Number of Bit Errors: %i\n",   pmkr.lNumBitErr );
   printf ( "Total Compare Count: %.0lf\n", pmkr.dTtlCmpCnt );
   printf ( "Bit Error Rate: %.10e\n\n",    pmkr.dBitErrRat );

   for ( indx = 0; indx < BEC_ERRS; indx++ )
   {
      BECT *bec = &pmkr.tBerTest[indx];

      if ( bec->lErrBits == 0 )
         break;
      printf ( "Error %i:\n", indx + 1 );
      printf ( " Pattern Repeat: %.0lf\n", bec->dLoopCnt );
      printf ( " Frame Number: %i\n",      bec->lFrameNo );
      printf ( " 20-bit Data in Error: " );

      /* Display each bit, noting bits in error with special characters */
      buff[1] = 0;
      for ( bitIndx = PATN_WORD_SIZE - 1; bitIndx >= 0; bitIndx-- )
      {
         if ( bec->lErrBits & ( 1 << bitIndx ) )
            buff[0] = ( bec->lExpBits & ( 1 << bitIndx ) ) ? 140 : 147;
         else
            buff[0] = ( bec->lExpBits & ( 1 << bitIndx ) ) ? '1' : '0';
         printf ( "%c", buff[0] );
         if (bitIndx == (PATN_WORD_SIZE / 2))
            printf( "  " );
      }
      printf ( "\r\n\n" );
   }

Error:
   /* Perform any cleanup and exit */
   FCNL_ClrDcom ( &dcom );
   COMM_CloseDev ( ApiDevId );
   return retn;
   }
```

5-1 SUPPORTED COMPILERS FOR THE *WAVECREST* PRODUCTION API

The *WAVECREST* Production API was built and is supported using the following compilers. Other compilers may be used and provide satisfactory results, although performance is not guaranteed.

Win32 (Win9x, WinNT 4.0 and Win2k)
Microsoft Visual C++ 5.0 and later
Microsoft C/C++ Optimizing Compiler 11.00
Microsoft Visual Basic 6.0 and later
National Instruments LabVIEW 6.1 and later
HP-UX 9.05, 10.2 and 11i
HP C/ANSI C Developer's Bundle B.10.20.03
Sun 4.1.x (Solaris 1)
SPARCompiler C 3.0.1
Sun 2.5.1 or above (Solaris 2)
SPARCompiler C 3.0.1
LINUX 7.2 and above
GNU compiler gcc version 2.96 or above

5-2 BUILD REQUIREMENTS

When building an application using the *WAVECREST* Production API, the following requirements need to be considered.

5-3 DEVELOPING WITH C++

The define **`CPLUSPLUS`** must be supplied if you are developing a C++ application. This informs the compiler that the module was created as a C library, and does not contain the additional information that is normally contained in a C++ library. If you are developing a standard C application, supplying this define will result in an error. If you are using a command line compiler, this define may be supplied as follows:

```
cl -c -DCPLUSPLUS sample.c
```

5-4 WIN32 (WIN9X, WINNT 4.0, WIN2K AND WINXP)

A static stub library and dynamic library link library (DLL) are supplied for developing under Microsoft Windows. You can link to the static stub library that relieves all the programming of the chores normally associated with linking to a DLL. The DLL libraries must be available in the current directory or somewhere in the PATH in order to execute the application.

The define **`WIN32`** must be supplied to enable options specific to Microsoft Windows platforms. If you are developing within the Visual C++ environment, this define is automatically supplied for you. If you are using a command line compiler, this define may be supplied as follows:

```
cl -c -DWIN32 sample.c
```

5-5 ALL UNIX PLATFORMS

The define **WIN32** must NOT be defined when compiling under UNIX platforms. This *define* enables options that are not suitable under UNIX platforms.

5-6 HP-UX 9.05, HP-UX 10.20 AND HP-UX 11i

The ANSI C compiler must be used. ANSI compatibility is enabled from a command line by specifying the **`-Aa`** option as follows:

`cc -c -Aa –DHPUX –DHP9X` (HP-UX 9.05)

`cc -c -Aa –DHPUX` (HP-UX 10.20)

`cc -c -Aa –DHPUX` (HP-UX 11.i)

Required HPIB support is supplied by linking to the Standard Instrument Control Library. This library must already be installed per manufacturers documentation. This library can be included by adding **`-lsicl`** to the link command. The resulting link command including the Wavecrest API libraries takes the form:

```
cc -Aa sample.o –lWChpb -lWCcu1 -lWCcu2 -lWCcu3 -lWCio –lWCmc -lWCfnl
   -lsicl -lm –o sample
```

5-7 SUN 4.1.X (SOLARIS 1)

The ANSI C compiler must be used. ANSI compatibility is enabled from a command line by using the **`acc`** command as follows:

```
acc -c –DSUNOS sample.c
```

Required GPIB support is supplied by linking to the National Instruments GPIB Library. This library must already be installed per manufacturers documentation. This library can be included by adding **`-lgpib`** to the link command. The resulting link command including the Wavecrest API libraries takes the form:

```
acc sample.o –lWChpb -lWCcu1 -lWCcu2 -lWCcu3 -lWCio –lWCmc -lWCfnl –lgpib
   -o sample
```

5-8 SUN 2.5.1 OR ABOVE (SOLARIS 2)

The standard ANSI C compiler must be used. The command line would appear as follows:

```
cc –c –DSUNOS -DSOLARIS sample.c
```

Required GPIB support is supplied by linking to the National Instruments GPIB Library. This library must already be installed per manufacturers documentation. This library can be included by adding **`-lgpib`** to the link command. The resulting link command including the Wavecrest API libraries takes the form:

```
cc sample.o –lWChpb -lWCcu1 -lWCcu2 -lWCcu3 -lWCio –lWCmc -lWCfnl -lgpib
   -lm -o sample
```

Define	Value	Description
SIA_SUCCESS	0	Success
SIA_ERROR	-1	Communication error with device
MEM_ERROR	-2	Could not allocate required memory
CMD_ERROR	-3	Invalid parameters passed to function
VER_ERROR	-4	Wrong version of software detected
FIT_ERROR	-5	Failure applying tail-fit
LIM_ERROR	-6	Results exceed specified limits
FIO_ERROR	-7	File I/O error
ARM_ERROR	-8	No suitable arm signal detected
TRG_ERROR	-9	No suitable trigger signal detected
USR_ERROR	-10	Operation was terminated by user
UNT_ERROR	-11	Unit Interval data exceeds limits
DDJ_ERROR	-12	DCD+DDJ data exceeds limits
VAR_ERROR	-13	Variance data for RJ+PJ exceeds limits
LRN_ERROR	-14	Learn Mode data exceeds limits
INT_ERROR	-15	Insufficient points for interpolation
TIM_ERROR	-16	Maximum measurement timeout exceeded
PCI_ERROR	-17	PCI bus error
LOK_ERROR	-18	Memory transfer error
CAL_ERROR	-19	Missing or invalid calibration file
SYS_ERROR	-20	System or hardware failure
PTN_ERROR	-21	Indicates an invalid pattern was used
FRQ_ERROR	-22	Channel does not support this Bit Rate
BEC_ERROR	-23	Pattern is too long for BEC comparison
NOI_ERROR	-24	Obtained an invalid Phase Noise result
PAT_ERROR	-25	DCD+ISI calibration pattern is no longer supported
PKT_ERROR	-26	Invalid data returned in binary packet

This page intentionally left blank.

APPENDIX B – VBASIC EXAMPLE

The following example shows what the sample program in Chapter 1 might look like written as a Visual Basic subroutine:

```
Private Sub Sample_Click()
' Start of Sample Program
Dim bHist As HIST
Dim bJitt As JITT
Dim bDcom As DCOM

Dim ApiDevid As Long
Dim Round As Long
Dim retn As Long
Dim avg As Double
Dim data(299) As Double

Dim per As String
Dim pw As String
Dim rise As String
Dim AsciData(255) As Byte
Dim AsciLeng As Long

' Initialize our structures
FCNL_DefHist bHist
FCNL_DefJitt bJitt
FCNL_DefDcom bDcom

' Bitfield of input channels to measure (Channel 1 = lower 16 bits; Channel 2
= upper 16 bits)
' Equivalent to (1 + (2 << 16) in ANSI C
bHist.tParm.lChanNum = 131073
bHist.tParm.lStopCnt = 1
bHist.tParm.lFuncNum = FUNC_TPD_PP

retn = FCNL_PtnName(bDcom.sPtnName(0), "clock.ptn")
If (retn <> SIA_SUCCESS) Then
    GoTo Error:
End If

bDcom.lQckMode = 1
bDcom.tParm.lChanNum = 1
bDcom.tParm.lAutoArm = ARM_EXTRN
bDcom.tParm.lExtnArm = 2

' Initialize device
ApiDevid = COMM_InitDev(GPIB_IO, "dev5")
If (ApiDevid < 1) Then
    GoTo Error:
End If
```

```
' Turn on calibration source
retn = COMM_TalkDev(ApiDevid, ":CAL:SIG 10MSQ")
If (retn <> SIA_SUCCESS) Then
    GoTo Error:
End If

' Go ahead and perform a pulsefind
retn = FCNL_PulsFnd(ApiDevid, bHist.tParm)
If (retn <> SIA_SUCCESS) Then
    GoTo Error:
End If

' Perform a simple measurement and get the average
retn = COMM_ReqDbl(ApiDevid, ":ACQ:RUN PER", avg)
If (retn <> SIA_SUCCESS) Then
    GoTo Error:
End If

' Print the results
mainDisplay.Text = " - Wavecrest Production API - " & _
    vbCrLf & " -   Sample Application     -" & vbCrLf & _
    vbCrLf & "Simple Period Command: " & _
    Format(avg * 1000000000#, "0.000") & "ns" & vbCrLf

' Perform a measurement and return the statistics
retn = FCNL_RqstPkt(ApiDevid, bHist, WIND_HIST)
If (retn <> SIA_SUCCESS) Then
    GoTo Error:
End If

' Now retrieve the plot structures for the previous measurement
' This call is not necessary unless you want the plot data
retn = FCNL_RqstAll(ApiDevid, bHist, WIND_HIST)
If (retn <> SIA_SUCCESS) Then
    GoTo Error:
End If

' Print the results
mainDisplay.Text = mainDisplay & "Single Histogram Mean: " & _
    Format(bHist.dNormAvg * 1000000000#, "0.000") & "ns" & _
    vbCrLf & "Single Histogram Sdev: " & _
    Format(bHist.dNormSig * 1000000000000#, "0.000") & "ps" & vbCrLf

retn = FCNL_RqstPkt(ApiDevid, bDcom, WIND_DCOM)
If (retn <> SIA_SUCCESS) Then
    GoTo Error:
End If

retn = FCNL_RqstAll(ApiDevid, bDcom, WIND_DCOM)
If (retn <> SIA_SUCCESS) Then
    GoTo Error:
End If
```

```
' Now define a group, the group must only be defined once
' There can be up to 20 different groups defined
retn = FCNL_GrpDefBeg(1)
If (retn <> 0) Then
    GoTo Error:
End If

' You can have standard ascii commands included in a group
retn = FCNL_GrpDefAsc(":ACQ:RUN PER")
If (retn <> SIA_SUCCESS) Then
    GoTo Error:
End If

' You can also retrieve blocks of binary data
retn = FCNL_GrpDefAsc(":MEAS:DATA?")
If (retn <> SIA_SUCCESS) Then
    GoTo Error:
End If

' And you can also use the structure calls, the zero argument skips plots
retn = FCNL_GrpDefPkt(bHist, WIND_HIST, 0)
If (retn <> SIA_SUCCESS) Then
    GoTo Error:
End If

' Ascii & structure calls can be interspersed
retn = FCNL_GrpDefAsc(":ACQ:RUN PW+")
If (retn <> SIA_SUCCESS) Then
    GoTo Error:
End If

' With this structure call, the 1 argument requests all the plot data
retn = FCNL_GrpDefPkt(bJitt, WIND_JITT, 1)
If (retn <> SIA_SUCCESS) Then
    GoTo Error:
End If

retn = FCNL_GrpDefPkt(bDcom, WIND_DCOM, 1)
If (retn <> 0) Then
    GoTo Error:
End If

' You can nest multiple ascii commands, but only the last should return data
retn = FCNL_GrpDefAsc(":ACQ:FUNC TT+;:ACQ:COUN 1000;:ACQ:MEAS")
If (retn <> SIA_SUCCESS) Then
    GoTo Error:
End If

' Finalize the group definition, for group 1
retn = FCNL_GrpDefEnd(ApiDevid, 1)
If (retn <> SIA_SUCCESS) Then
    GoTo Error:
End If
```

```
' The definition doesn't acquire anything; use WavGrpGetAll to acquire
' You can loop and re-use the same definition over and over again
For Round = 0 To 1 Step 1
    ' WavGrpGetAll does the measurements and gets the whole block of data
    retn = FCNL_GrpGetAll(ApiDevid, 1)
    If (retn <> SIA_SUCCESS) Then
        GoTo Error:
    End If

    ' The following calls parse the individual results out of the group data
    ' There must be a 1-to-1 correspondence between the definition and these
    ' calls
    retn = FCNL_GrpGetAsc(AsciData(0), 256)
    If (retn <> SIA_SUCCESS) Then
        GoTo Error:
    End If

    For AsciLeng = 0 To 255 Step 1
        per = per & Chr$(AsciData(AsciLeng))
    Next AsciLeng

    ' The same method is used for binary blocks from ascii requests
    retn = FCNL_GrpGetAsc(data(0), 2400)
    If (retn <> SIA_SUCCESS) Then
        GoTo Error:
    End If

    ' For structure calls, the bGetPlot argument must be the same as in the
    ' definition
    retn = FCNL_GrpGetPkt(bHist, WIND_HIST, 0)
    If (retn <> SIA_SUCCESS) Then
        GoTo Error:
    End If

    retn = FCNL_GrpGetAsc(AsciData(0), 256)
    If (retn <> SIA_SUCCESS) Then
        GoTo Error:
    End If

    For AsciLeng = 0 To 255 Step 1
        pw = pw & Chr$(AsciData(AsciLeng))
    Next AsciLeng

    ' If bGetPlot = 1, plots are returned; these can be BIG and will be
    ' slower!!!
    retn = FCNL_GrpGetPkt(bJitt, WIND_JITT, 1)
    If (retn <> SIA_SUCCESS) Then
        GoTo Error:
    End If

    retn = FCNL_GrpGetPkt(bDcom, WIND_DCOM, 1)
    If (retn <> SIA_SUCCESS) Then
        GoTo Error:
    End If
```



```
    retn = FCNL_GrpGetAsc(AsciData(0), 256)
    If (retn <> SIA_SUCCESS) Then
        GoTo Error:
    End If

    For AsciLeng = 0 To 255 Step 1
        rise = rise & Chr$(AsciData(AsciLeng))
    Next AsciLeng

    ' Print the simple ascii command results and print the start of the binary
    ' block of raw data
    mainDisplay.Text = mainDisplay & "Group Loop " & _
        Format(Round + 1, "0") & " - Period Measurement: " & per
    mainDisplay.Text = mainDisplay & vbCrLf & "Group Loop " & _
        Format(Round + 1, "0") & " - Pulsewidth Measurement: " & pw
    mainDisplay.Text = mainDisplay & vbCrLf & "Group Loop " & _
        Format(Round + 1, "0") & " - Risetime Measurement: " & rise
    mainDisplay.Text = mainDisplay & vbCrLf & "Group Loop " & _
        Format(Round + 1, "0") & " - Raw Period Measurements: " & _
        Format(data(0) * 1000000000#, "0.000") & "ns," & _
        Format(data(1) * 1000000000#, "0.000") & "ns," & _
        Format(data(2) * 1000000000#, "0.000") & "ns, ..." & vbCrLf

    ' Print out some of the statistics from the HIST and JITT tool structures
    mainDisplay.Text = mainDisplay & "Group Loop " & _
        Format(Round + 1, "0") & " - Histogram Mean: " & _
        Format(bHist.dNormAvg * 1000000000#, "0.000") & "ns" & _
        vbCrLf & "Group Loop " & _
        Format(Round + 1, "0") & " - Histogram Sdev: " & _
        Format(bHist.dNormSig * 1000000000000#, "0.000") & "ps" & _
        vbCrLf & "Group Loop " & _
        Format(Round + 1, "0") & " - 1Clock RJ: " & _
        Format(bJitt.dRjit1Clk * 1000000000000#, "0.000") & "ps" & _
        vbCrLf & "Group Loop " & _
        Format(Round + 1, "0") & " - NClock RJ: " & _
        Format(bJitt.dRjitNClk * 1000000000000#, "0.000") & "ps" & vbCrLf

    ' Print the max of the FFT to show how data within a plot is accessed and
    ' print the dataCOM tool DJ & RJ values
    mainDisplay.Text = mainDisplay & "Group Loop " & _
        Format(Round + 1, "0") & " - dataCOM DJ: " & _
        Format(bDcom.dDdjt * 1000000000000#, "0.000") & "ps" & _
        vbCrLf & "Group Loop " & _
        Format(Round + 1, "0") & " - dataCOM RJ: " & _
        Format(bDcom.dRjit(0) * 1000000000000#, "0.000") & "ps" & _
        vbCrLf & "Group Loop " & _
        Format(Round + 1, "0") & " - NClock Plot Max: " & _
        Format(FCNL_GetYval(bJitt.tFftN, bJitt.tFftN.lYmaxIndx) * 1000000000000#, "0.000")
& "ps" & vbCrLf
```

```
Next Round

Error:
' Return an error message if we had a problem
If (retn) Then
    mainDisplay.Text = mainDisplay & vbCrLf & "ERROR! Return Code: " &
Format(retn, "0")
End If

' Perform any cleanup and exit
FCNL_ClrHist bHist
FCNL_ClrJitt bJitt
FCNL_ClrDcom bDcom
COMM_CloseDev ApiDevid
End Sub
```

APPENDIX C – PAPI REVISION CHANGES

The following listings provide changes to the measurement window structures and sub-structures for all supported revisions of PAPI. Find the version of *GigaView* or *VISI* that is currently installed on your SIA-3000. All the changes in that section and previous sections (newer versions) show the differences between the latest version of PAPI and the version of PAPI compatible with your SIA-3000.

GIGAVIEW 1.5 CHANGES (FROM GIGAVIEW 1.4)

New Tools

Folded Eye Diagram (`FEYE`)

Measurement Window Structure Changes

Added Input Parameters

None

Sub-Structure Changes

None

GIGAVIEW 1.4 CHANGES (FROM GIGAVIEW 1.3)

New Tools

PCI Express 1.1 Clock Analysis (`PCLK`)

PCI Express 1.1 Hardware Clock Recovery (`PCIM`)

PCI Express ATA 1.1 Software Clock Recovery (`EXPR`)

Serial ATA Gen2i & Gen2m (`ATA2`)

Serial ATA Gen1x & Gen2x (`ATAX`)

Measurement Window Structure Changes

Added Input Parameters

None

Sub-Structure Changes

None

GIGAVIEW 1.3 CHANGES (FROM GIGAVIEW 1.2)

New Tools

Feature Analysis (`FEAT`)

Measurement Window Structure Changes

Added Input Parameters

EYEH – `lFiltOff`

Sub-Structure Changes

None

GIGAVIEW 1.2 CHANGES (FROM GIGAVIEW 1.1)

New Tools

None

Measurement Window Structure Changes

Added Input Parameters

CANL – `lHiRFmV, lLoRFmV, dAttn[POSS_CHNS]`

FCMP – `dAttn`

INFI – `dAttn`

PCIX – `lPcnt, lHiRFmV, lLoRFmV, lIdleOk, dAttn`

SCOP – `lVdif[ POSS_CHNS ], lVcom[ POSS_CHNS ], lHiRFmV, lLoRFmV`

Added Output Parameters

CANL - `qComm[POSS_CHNS], tComm[POSS_CHNS]`

INFI – `tDifScop, tComScop`

PCIX – `dEyeOffs, dXmnDiff, dXmxDiff, dVcommonAc, dVcommonDc, dVcmDcActv, dVcmIdleDc, dVcmDcLine, dVcmDcDpls, dVcmDcDmin, dVIdleDiff, *bTranEye, lTranRsv, *bDeemEye, lDeemRsv`

SCOP – `qComm[ POSS_CHNS ], tComm[ POSS_CHNS ]`

Element Order and Padding

INFI – `tEyeh` moved

INFI – Added `lPad1` and `lPad2`

Sub-Structure Changes

Added Parameters

MASK – `dV0pas, dXwdUI, dXflUI, dYiPct, dV1Rel, dV0Rel`

QTYS – `dMaskRgn1, dMaskRgn2, dMaskRgn3`

Modified Parameters

MASK – `dToffs, dVoffs` are now ignored

MASK – `dVpass` renamed to `dV1pas`

GIGAVIEW 1.1 CHANGES (FROM GIGAVIEW 1.0)

New Tools

Spread Spectrum Clock Analysis (`SSCA`)

Measurement Window Structure Changes

Added Input Parameters

DCOM – `lTfitCnt`

EYEH – `lKeepOut, dKpOutLt, dKpOutRt`

HIST – `lKeepOut, dKpOutLt, dKpOutRt`

Added Output Parameters

EYEH – `tBoth, tBothProb`

Element Order and Padding

HIST – `Added lPad0`

EYEH, RCPM, SIMP and STRP – Added `lPad1`

PCIX – Added `lPad0 and lPad1`

SATA – Added `lPad3`, `lPad4`, and `lPad5`

SCOP – Added `lPad1` and `lPad2`

Sub-Structure Changes

None

GIGAVIEW 1.0 CHANGES (FROM VISI 7.4.0)

NOTE: Beginning with this release, VISI is now called GigiView and a new version numbering system has been started.

New Tools

Clock Analysis (`CANL`)
Infiniband (`INFI`)
PCI Express (`PCIX`)
Recovered Clock / Pattern Marker dataCOM (`RCPM`)
Serial ATA (`SATA`)

Measurement Window Structure Changes

Added Input Parameters

SCOP – `dAttn[ POSS_CHNS ]`

Added Output Parameters

HIST – `tShrt, tLong, tBoth`
SCOP – `qNorm[ POSS_CHNS ], qComp[ POSS_CHNS ], qDiff[ POSS_CHNS ]`

Modified Parameters

SCOP – `qDisp[ POSS_CHNS ]` eliminated. Use `qNorm[ POSS_CHNS ]`.

Sub-Structure Changes

None

VISI 7.4.0 CHANGES (FROM VISI 7.3.0)

New Tools

None

Measurement Window Structure Changes

Added Input Parameters

SCOP – `lVoff[ POSS_CHNS ], dHistDly, dHistWid, dHistVlt, dHistHgt`

Added Output Parameters

SCOP – `tHorz[ POSS_CHNS ], tVert[ POSS_CHNS ]`

Modified Parameters

SCOP – `lMask` eliminated

Sub-Structure Changes

New Structures

Oscilloscope Histogram `(OHIS)`

Added Parameters

QTYS – `dMidVolts`

VISI 7.3.0 Changes (from VISI 7.2.1)

New Tools

Clock Statistics (`CLOK`)
New Oscilloscope (`SCOP`)

Measurement Window Structure Changes

Added Output Parameters
APLL - `tInit`

Sub-Structure Changes

New Structures
Measurement (`MEAS`)
Quantities (`QTYS`)
Mask (`MASK`)

VISI 7.2.1 Changes (from VISI 7.2.0)

NOTE: VISI 7.2.2 and 7.2.1 are identical as far as PAPI structures are concerned.

New Tools

None

Measurement Window Structure Changes

Added Input Parameters
APLL – `dRecTime, lRecUnit, lIniCond`
Modified Parameters
APLL – `lAutoFix` eliminated
Element Order and Padding
APLL – `dCornFrq` moved
APLL – `lPad1` eliminated

Sub-Structure Changes

None

WAVECREST CORPORATION

World Headquarters:
7626 Golden Triangle Drive
Eden Prairie, MN 55344
TEL: (952) 831-0030
FAX: (952) 831-4474
Toll Free: 1-800-733-7128
www.wavecrest.com

West Coast Office:
1735 Technology Drive, Ste. 400
San Jose, CA 95110
TEL: (408) 436-9000
FAX: (408) 436-9001
1-800-821-2272

Europe Office:
Hansastrasse 136
D-81373 München
TEL: +49 (0)89 32225330
FAX: +49 (0)89 32225333

Japan Office:
Otsuka Sentcore Building, 6F
3-46-3 Minami-Otsuka
Toshima-Ku, Tokyo
170-0005, Japan
TEL: +81-03-5960-5770
FAX: +81-03-5960-5773

200212-04 REV A