AnalyserC C library

User's Guide

Table of Content

Table of Tables

References

[]

History

Version	Date	Description
1.00	01-May-2006	Initial revision (preliminary)
1.01	15-May-2006	First release
1.02	09-Nov-2006	Update for software version 1.04
1.03	31-May-2007	Update with edge trigger functions
1.04	29-Aug-2007	Added trigger positioning function / description
1.05	13-Nov-2007	Update for GP Series Introduction
1.06	16-Feb-2010	Review for release 1.08f.
1.07	09-Aug-2010	Added function prefixes and multi-device support
1.08	20-Jan-2012	Added IO voltage selection
1.09	20-Sep-2013	Added analyzer abort function

1 Introduction

The AnalyserC C library is a specialised C library used with the GP Series device in Analyser mode of operation. It provides a serie of 'pure C' functions to configure and control the GP Series device from within a C/C++ compatible environment. As opposed to the GP Series device C++ libraries, this library offers a 'pure C' interface with each function, which is often easier to integrate from within any external environment.

This library calls itself other libraries and functions to manage the low level transfer of data between the GP Series device and the host PC. Schematically, any session using the AnalyserC library starts by connecting itself to the 'Smart Router' application delivered with the 8PI Control Panel. This application manages the different client connections to the GP Series device and handles priorities between the processes and applications. On the other side, it is responsible for the actual data transfers onto the USB connection.

For advanced information and support, please submit your requests to: support@byteparadigm.com.

2 AnalyserC C Library

2.1 Functions quick Reference Table

Table 1 gives a list of the functions callable from the Analyser, in the same order as in the header file: AnalyserC.h.

Table 1: Quick reference table of Analyser procedures (by functionality)

Function prototype	Description
`int  ana_AnalyserC(void)`	Initialises the AnalyserC library session and creates a first instance of the AnalyserC library.
`int  ana_CreateInstance(void)`	Creates and additional instance of the library.
`void ana_SelectInstance(int Handle)`	Selects a library instance.
`int  ana_SelectDevice(char *pSerNum)`	Selects a device using the serial number.
`void ana_SelectIOVoltage(int IOVoltage)`	Selects the user interface voltage.
`short ana_Ver(void)`	Returns the software version.
`bool ana_IsDeviceReady(void);`	Checks if device is ready to operate.
`bool ana_IsRunning(void)`	Checks if the device is busy and running.
`void ana_Terminate(void);`	Closes an instance of the AnalyserC library. A call to this function is mandatory for each instance of the library before closing the application.
`void ana_SetReqClock(int Freq);`	Defines the operating clock frequency (in Hz).
`int  ana_GetReqClock(void);`	Returns the requested operating clock frequency (Hz).
`int  ana_GetSynthClock(void);`	Returns the synthesised operating clock frequency (Hz). It can differ from the requested frequency due to the limited accuracy of the frequency divider.
`int  ana_GetSynthOutClock(void);`	Returns the achieved frequency (Hz) of the output clock. This value depends on the operating clock frequency and the output clock ratio.
`void ana_SetOutputClock(bool Output);`	Enables or disables the generation of the output clock
`bool ana_GetOutputClock(void);`	Gets the status of the output clock generation (enabled or disabled).
`void ana_UseIntClock(void);`	Configures the device to operate with the internal operating clock as reference.
`void ana_UseExtClock(void);`	Configures the device to operate with an externally supplied operating clock as reference.
`bool ana_GetClockSource(void);`	Returns the selected clock source.
`void ana_SetClockEdge(bool Pos);`	Defines the phase relation between the output clock and the device internal clock.
`bool ana_GetClockEdge(void);`	Returns the phase relation between the device internal clock and the generated output clock.
`void ana_SetClockSampling(bool RisingEdge);`	Defines the clock edge used to sample incoming data.
`bool ana_GetClockSampling(void);`	Returns the clock edge currently used to sample data.
`void ana_SetOutClockRatio(int Ratio);`	Defines the integer ratio between the operating clock frequency and the output clock frequency.
`int ana_GetOutClockRatio(void);`	Returns the current value of the output clock ratio.
`void ana_SetClockDisablePLL(bool Disable);`	Forces disabling the PLL or let the device enable it automatically when possible.
`bool ana_GetClockDisablePLL(void);`	Gets the operating mode of the PLL (disabled / automatic).
`void ana_SetInternalTrigger(bool Internal);`	Defines the triggering mode (internal or external).

Function prototype	Description
bool ana_GetInternalTrigger(void);	Returns the current triggering mode.
void ana_SetEdgeTrigger(bool Enable)	Selects the triggering mode (edge or level)
bool ana_GetEdgeTrigger(void)	Returns the triggering mode (edge or level)
int ana_SetTriggerPos(int Sample)	Defines the position of the trigger in the run – that is the number of samples before the trigger.
int ana_GetTriggerPos(void)	Returns the programmed trigger position.
void ana_SetCtrlTrigMask(short *pMask);	Defines the triggering mask to select the control lines to use as trigger inputs.
void ana_GetCtrlTrigMask(short *pCtrlTrigMask);	Returns the triggering mask.
void ana_SetCtrlTrigPattern(short *pPattern);	Defines the pattern to detect on the trigger inputs to generate the trigger event.
void ana_GetCtrlTrigPattern(short *pTrigPattern);	Returns the triggering pattern.
void ana_EnDefaultCtrl(void);	Enables applying default static levels on the control lines.
void ana_DisDefaultCtrl(void);	Disables applying a default level on the control lines.
bool ana_GetDefaultCtrlEn(void);	Returns the status of the default level feature.
void ana_SetCtrlMaskIn(short *pMaskIn);	The control mask selects the control lines on which the default static levels have to be applied.
void ana_GetCtrlMaskIn(short *pMaskIn);	Returns the control mask value.
void ana_SetCtrlDefaultVal(short *pDefaultVal);	Defines the default level to apply on the control lines.
void ana_GetCtrlDefaultVal(short *pCtrlDefaultVal);	Returns the current default level applied on the control lines.
void ana_SetDataMaskIn(short *pMaskIn);	Selects the data lines enabled as inputs.
void ana_GetDataMaskIn(short *pMaskIn);	Returns the data mask value.
int ana_AnalyserRun(int NrSamples, bool StepStart, bool Message);	Starts collecting data with the GP Series device and store them in the host memory.
int ana_AnalyserRunH(int NrSamples, bool StepStart, bool Message int Handle);	Starts collecting data with the GP Series device (using the device linked to the specified library instance) and store them in the host memory.
int ana_MakeConfFileTemplate(char *FileName);	Creates a configuration file template.
int ana_ReadConfFileAnalyser(char *FileName, bool Message);	Reads the device controls, configuration and data header from a file.
int ana_WriteConfFile(char *FileName);	Writes the current configuration to a file.
int ana_GetLastErr(void);	Returns the last error detected during the data transfer.
void ana_ResetCfg(void);	Reset the device to its default configuration.
void ana_Unsupervised(bool Enable);	Enables or disables the unsupervised operating mode.
int ana_GetCapturedData(int nr, short *CapturedData);	Returns selected data stored in memory with the previous run.
void ana_SetExportAutoSave(bool AutoSave);	Enables / disables the 'auto save' option.
bool ana_GetExportAutoSave(void);	Returns the 'auto save' option status (enabled / disabled).
void ana_SetExportFileType(int Type);	Defines the 'auto save' option file type.

Function prototype	Description
`int ana_GetExportFileType(void);`	Returns the file type selected for the 'auto save' option.
`int ana_ExportRawBinFile(char *FileName);`	Exports the data collected with the Analyser to a raw binary file.
`int ana_ExportRawTextFile(char *FileName);`	Exports the data collected with the Analyser to a raw text file.
`int ana_ExportVCDFile(char *FileName);`	Export the data collected with the Analyser to a VCD file.
`void ana_Abort(void);`	Aborts the pending run

2.2 Functions details

This section gives a detailed description of each function available to control the GP Series device Analyser operating mode with the AnalyserC library. The functions are listed in alphabetical order.

`int ana_AnalyserC (void)`

parameters:	none
returns:	A handle to the initialised library instance.
description:	Initialises an Analyser session and creates a first instance of the AnalyserC library. This function must be called at the start of any session using the AnalyserC library. It enables the control of the device by 'connecting' the C session as a client to the 'Smart Router'.

`int ana_AnalyserRun (int NrSamples, bool StepStart, bool Message)`

parameters:	NrSamples: integer value that specifies the number of samples to be collected. StepStart: must be set to 0. Used to support advanced functions – please contact support@byteparadigm.com for more information. Message: boolean enabling or disabling the message pop-ups during run.
returns:	An error code integer. 0 if successful.
description:	Directs the sampling of data from the GP Series device system connector. The specified number of samples are collected from the GP Series device system connector and transferred to the host PC memory. If the AutoSave option is selected, this procedure also automatically saves the collected samples, with the chosen predefined options.
see also:	SetDataMaskIn, ExportRawTextFile, ExportRawBinFile, ExportVCDFile, SetExportAutoSave, SetExportFileType, SetExportFileName.

`int ana_AnalyserRunH (int NrSamples, bool StepStart, bool Message, int Handle)`

parameters:	NrSamples: integer value that specifies the number of samples to be collected. StepStart: must be set to 0. Used to support advanced functions – please contact support@byteparadigm.com for more information. Message: boolean enabling or disabling the message pop-ups during run. Handle: handle of a library instance
returns:	An error code integer. 0 if successful.
description:	Directs the sampling of data from the GP Series device system connector. The device linked to the specified handle is used. The specified number of samples are collected from the GP Series device system connector and transferred to the host PC memory. If the AutoSave option is selected, this procedure also automatically saves the collected samples, with the chosen predefined options.

see also:	SetDataMaskIn, ExportRawTextFile, ExportRawBinFile, ExportVCDFile, SetExportAutoSave, SetExportFileType, SetExportFileName.

int ana_CreateInstance(void)

parameters:	none
returns:	A handle to the created library instance.
description:	Creates an additional instance of the library. This in needed when using more than one device at the same time. Each instance of the library can be linked to a different device with the ana_SelectDevice function.

void ana_DisDefaultCtrl (void)

parameters:	none.
returns:	
description:	Disables the application of a forced default level on the control lines.
see also:	EnDefaultCtrl.

void ana_EnDefaultCtrl (void)

parameters:	none.
returns:	
description:	Enables the application of default static levels on the control lines. When this feature is enabled, static levels can be set in the selected control lines. Valid for File or Static modes.
conditions:	To enable the default level feature for the control lines, the control sequence feature has to be enabled.
see also:	DisDefaultCtrl.

int ana_ExportRawBinFile (char *FileName)

parameters:	*FileName: specifies the function output file name (including path)
returns:	0 when the file save completed successfully.
description:	Saves the last sampling run to a file, formatted as raw binary data.
see also:	AnalyserRun.

int ana_ExportRawTextFile (char *FileName)

parameters:	*FileName: specifies the function output file name (including path)
returns:	0 when the file save completed successfully.
description:	Saves the last sampling run to a file, formatted as raw text data.
see also:	AnalyserRun.

int ana_ExportVCDFile (char *FileName)

parameters:	*FileName: specifies the function output file name (including path)
returns:	0 when the file save completed successfully.
description:	Saves the last sampling run to a file, formatted as value change dump (VCD) vector information.
see also:	AnalyserRun.

int ana_GetCapturedData (int nr, short *CapturedData)

parameters:	nr: rank to the data to return *CapturedData: pointer to a short holding the requested data.

returns:	An integer error code. 0 if operation was successful.
description:	Returns a single captured sample as a short integer. *CapturedData points to this sample. The nr parameter selects which data from the last run (start index = 0) is to be returned.
see also:	AnalyserRun

bool ana_GetClockDisablePLL (void)

parameters:	none.
returns:	1 when the device internal PLL is disabled. 0 when the PLL automatic control mode is enabled.
description:	Returns the control mode of the device internal PLL.
see also:	SetClockDisablePLL.

bool ana_GetClockEdge (void)

parameters:	none.
returns:	0 Output data lines are generated in phase with the falling edge of the output clock. 1 Output data lines are generated in phase with the rising edge of the output clock.
description:	Returns the phase relation between the device internal clock and the generated output clock.
see also:	SetClockEdge.

bool ana_GetClockSampling (void)

parameters:	none.
returns:	1 if the rising edge of the clock is used. 0 if the falling edge of the clock is used.
description:	Returns the clock edge currently used to sample the incoming data.
see also:	SetClockSampling.

bool ana_GetClockSource (void)

parameters:	none.
returns:	1 if the internal reference clock is used. 0 if an externally supplied reference clock is used.
description:	Returns the reference clock source used to generate the device operating clock.
see also:	UseIntClock , UseExtClock.

void ana_GetCtrlDefaultVal (short *pDefaultVal)

parameters:	*pDefaultVal: pointer to a short value. This value represents a 6 bits value and ranges from 0 to 63.
returns:	
description:	Returns the current default level pattern defined for the 6 control lines. The value is returned through the *pDefaultVal pointer.
see also:	SetCtrlDefaultVal

void ana_GetCtrlMaskIn (short *pMaskIn)

parameters:	*pMaskIn: pointer to a short value. This value represents a 6 bits mask and ranges from 0 to 63.
returns:	
description:	Returns a mask value representing the control lines enabled as output. The value is returned through the *pMaskIn pointer.
see also:	SetCtrlMaskIn

void ana_GetCtrlTrigMask (short *pCtrlTrigMask)

parameters:	*pCtrlTrigMask: pointer to a short value. This value represents a 6 bits mask and ranges from 0 to 63.
returns:	
description:	Returns the mask applied on the control lines to detect the external trigger. The value is returned through the *pCtrlTrigMask pointer.
see also:	SetCtrlTrigMask

void ana_GetCtrlTrigPattern (short *pCtrlTrigPattern)

parameters:	*pCtrlTrigPattern: pointer to a short value. This value represents a 6 bits pattern and ranges from 0 to 63.
returns:	
description:	Returns the pattern applied on the control lines to generate a trigger event and start applying data samples on the GP Series device system connector. The value is returned through the *pCtrlTrigPattern pointer.
see also:	SetCtrlTrigPattern.

void ana_GetDataMaskIn (short *pMaskIn)

parameters:	*pMaskIn: pointer to a short value. This value represents a 16 bits mask and ranges from 0 to 65535.
returns:	
description:	Returns a mask value representing the data lines enabld as output. The value is returned through the *pMaskIn pointer.
see also:	SetDataMaskOut.

bool ana_GetDefaultCtrlEn (void)

parameters:	none.
returns:	1 when the default level feature is enabled. 0 when the default level feature is disabled.
description:	Returns the status of the 'deafult level' feature for the control lines.
see also:	EnDefaultCtrl.

bool ana_GetEdgeTrigger (void)

parameters:	none.
returns:	
description:	Returns the current triggering mode: 1 for 'edge triggering mode'; 0 for 'level triggering mode'.
see also:	

bool ana_GetExportAutoSave (void)

parameters:	none.
returns:	1 when the AutoSave option is enabled.

	0 when the AutoSave option is disabled.
description:	The AutoSave option allows to automatically save the data collected with the AnalyserRun function to a file specified with the SetExportFileName and SetExportFileType funcions. GetExportAutoSave returns the enable/disable status of this option.
see also:	AnalyserRun, SetExportFileName, SetExportFileType, SetExportAutoSave.

int ana_GetExportFileType (void)

parameters:	none.

returns:	An integer representing the type of the file used with the AutoSave option.
	0: Raw text file;
	1: Raw binary file;
	2: VCD file.

description:	The AutoSave option allows to automatically save the data collected with AnalyserRun command to a file specified with the SetExportFileName and SetExportFileType functions. GetExportFileType returns the file type set with the command SetExportFileType.
see also:	AnalyserRun, SetExportFileName, SetExportFileType, SetExportAutoSave.

bool ana_GetInternalTrigger (void)

parameters:	none.
returns:	1 when the internal triggering mode is enabled. 0 when the external triggering mode is enabled.
description:	Returns the triggering mode in use.
see also:	SetInternalTrigger.

int ana_GetLastErr (void)

parameters:	none.
returns:	An integer value.
description:	Returns the last error code returned by the device.
see also:	AnalyserRun.

int ana_GetOutClockRatio (void)

parameters:	none.
returns:	An integer value between 1 and 65535.
description:	Returns the integer ratio between the output clock frequency and the operating clock frequency.
see also:	SetOutClockRatio.

bool ana_GetOutputClock (void)

parameters:	none.
returns:	1 when the output clock is enabled. 0 when the output clock is disabled.
description:	Returns the status of the output clock generation.
see also:	SetOutputClock.

int ana_GetReqClock (void)

parameters:	none.
returns:	An integer value representing a frequency in Hz.
description:	Returns the frequency requested with the function 'SetReqClock', for the device operating clock frequency.

see also:	SetReqClock, GetSynthClock.

int ana_GetSynthClock (void)

parameters:	none.
returns:	An integer value representing a frequency in Hz.
description:	Returns the achieved frequency for the system operating clock. Since the operating clock is generated using an integer ratio to divide a reference clock, not all frequencies can be generated. This function returns the achieved frequency value. This value can differ from the requested clock frequency. If the requested frequency can not be achieved, the device uses the first achievable frequency smaller than the requested one.
see also:	void SetReqClock(int Freq), int GetSynthClock(void).

int ana_GetSynthOutClock (void)

parameters:	none.
returns:	An integer value representing a frequency in Hz.
description:	Returns the achieved frequency of the output clock.
see also:	void SetOutClockRatio(int Ratio)

int ana_GetTriggerPos (void)

parameters:	none.
returns:	A integer representing the trigger position.
description:	Returns the trigger position previously programmed. The trigger position is defined with the sample index at which it is positionned.

bool ana_IsDeviceReady (void)

parameters:	none
returns:	1 if device is ready
description:	Checks if the device is properly connected to the host PC.

bool ana_IsRunning (void)

parameters:	none
returns:	1 if device is running
description:	Checks if the device is busy receiving data.

int ana_MakeConfFileTemplate (char *FileName)

parameters:	FileName: specifies the function output file name (including path).
returns:	0 when file generation is complete.
description:	Creates a standard template to be used as configuration file with the device.

int ana_ReadConfFileAnalyser (char *FileName, bool Message)

parameters:	FileName: specifies the function input file name (including path). Message: when set to 1, enables the pop-up windows for information and error messages.
returns:	0 if read successful; another value if read failed.

description:	Reads the device configuration, the control sequence configuration and the data header from the specified file. Does not expect any set of data defined in the input file.
see also:	

void ana_ResetCfg (void)

parameters:	none.
returns:	
description:	Resets the device to a default state.

int ana_SelectDevice(char *pSerNum)

parameters:	pSerNum: ascii encoded string containing an 11 character serial number
returns:	-1 when the selection fails, a positive of zero value is returned on success
description:	Selects the device based on its serial number. The selected device is associated with the currently selected library instance.

void ana_SelectIOVoltage(int *IOVoltage)

parameters:	IOVoltage: integer value representing the IO voltage, the IO voltage can be internally generated of user applied. The voltage level is defined in millivolts.
returns:	
description:	This function only takes the following predefined values: 3300, 2500, 1800, 1500 and 1200. The nearest value must be selected when the user applies a different external voltage level. For example, set IOVoltage to 2500 when 2.7V is applied. The default value is 3300.

void ana_SelectInstance(int Handle)

parameters:	Handle: Handle to a instance of a previously create library instance.
returns:	
description:	Selects the library instance corresponding to the supplied handle. The selected instance must first be created with ana_CreateInstance.

void ana_SetClockDisablePLL (bool Disable)

parameters:	Disable: boolean value (0 or 1).
returns:	
description:	Forces the disabling of the device internal PLL used for the connector clock generation. When the Disable parameter is set to 0, the PLL will be automatically enabled, if possible. When Disable is set to 1, the PLL is not used.
see also:	GetClockDisablePLL

void ana_SetClockEdge (bool Pos)

parameters:	Pos: boolean value (0 or 1).
returns:	
description:	Defines the phase relation between the output clock and the device internal clock. When the clock edge 'Pos' parameter is set to 1, the output clock is in phase with the device internal clock; when this parameter is set to 0, there is a 180° phase shift between the clocks.
see also:	GetClockContinuity

void ana_SetClockSampling (bool RisingEdge)

parameters:	RisingEdge: boolean value (0 or 1).
returns:	
description:	Defines the clock edge used to sample data.
see also:	GetClockContinuity

void ana_SetCtrlDefaultVal (short *pDefaultVal)

parameters:	*pDefaultVal: pointer to a short value. This value represents a 6 bit value and ranges from 0 to 63.
returns:	
description:	The static default value is applied on the control lines with the first data sent out to the device. The levels remain constant until a new control pattern is defined. If a new default value is defined, it will only be applied on the control lines with the next outgoing data sample. The default value is given as a pointer to a short value equivalent to the binary value of the default value (example: default value = (Binary)011001 ➔ *pDefaultVal points to a short = 25).
see also:	GetCtrlMaskOut

void ana_SetCtrlMaskIn (short *pMaskIn)

parameters:	*pMaskIn: pointer to a short value. This value represents a 6 bit mask and ranges from 0 to 63.
returns:	
description:	The control mask selects the device control lines on which the default static levels have to be applied. When a mask bit is set to 0, the corresponding control line is masked. The mask is given as a pointer to a short value equivalent to the binary value of the mask (example: mask = (Binary)011001 ➔ *pMaskOut points to a short = 25).
see also:	GetCtrlMaskIn

void ana_SetCtrlTrigMask (short *pMask)

parameters:	*pMask: pointer to a short value. This value represents a 6 bit mask and ranges from 0 to 63.
returns:	
description:	When the external triggering mode is used, the trigger mask selects the control lines to be used as trigger inputs. When a mask bit is set to 0, the corresponding control line is masked for triggering. The mask is given as a pointer to a short value equivalent to the binary value of the mask (example: mask = (Binary)011001 ➔ *pMask points to a short = 25).
see also:	GetCtrlTrigMask

void ana_SetCtrlTrigPattern (short *pPattern)

parameters:	*pPattern: pointer to a short value. This value represents a 6 bit pattern and ranges from 0 to 63.
returns:	
description:	Defines the pattern to detect on the trigger inputs to generate the trigger event. The pattern is given as a pointer to a short value equivalent to the binary value of the pattern (example: pattern = (Binary)011001 ➔ *pPattern points to a short = 25).
see also:	GetCtrlTrigPattern

void ana_SetInternalTrigger (bool Internal)

parameters:	Internal: boolean value (0 or 1).
returns:	
description:	Defines the triggering mode. When Internal is set to 1, the internal triggering mode is selected; when Internal is set to 0, the external triggering mode is selected. When internal mode is selected, the device does not wait for any external event to start the data run.
conditions:	

void ana_SetDataMaskIn (short *pMaskIn)

parameters:	*pMaskIn: pointer to a short value. This value represents a 16 bit mask and ranges from 0 to 65535.
returns:	
description:	Applies a mask onto the data input lines to enable / disable them. The mask is given as a pointer to a short value equivalent to the binary value of the mask (example: mask = (Binary)0110111110111101 ➔ *pMaskIn points to a short = 28605).
see also:	GetDataMaskIn

void ana_SetEdgeTrigger (bool Enable)

parameters:	Enable: Valid values: 0 or 1 – 0 for 'level trigger', 1 for 'edge trigger'
returns:	
description:	Selects the triggering mode: in edge triggering mode (Enable = 1), the AnalyserRun run will be triggered upon detection of a transition to the programmed trigger pattern. In level triggering mode (Enable = 0), the AnalyserRun run is triggered upon detection of the programmed trigger pattern.
see also:	

void ana_SetExportAutoSave (bool AutoSave)

parameters:	AutoSave: boolean value: 0 to disable the AutoSave option; 1 to enable it.
returns:	
description:	Enables or disables the AutoSave option. This options allows to automatically save the data collected with the AnalyserRun command to a file specified with the SetExportFileName and SetExportFileType functions.
see also:	GetExportAutoSave, AnalyserRun, SetExportFileName, SetExportFileType.

void ana_SetExportFileType (int Type)

parameters:	Type: integer value representing the file type. 0 : raw text file; 1 : raw binary file; 2 : VCD file.
returns:	
description:	Specifies the type of the output file used with the AutoSave option.
see also:	GetExportFileType, AnalyserRun, SetExportFileName, SetExportFileType.

void ana_SetOutClockRatio (int Ratio)

parameters:	Ratio: integer value representing the ratio between the output clock frequency and the operating clock frequency. Valid range:

	1 to 65535.
returns:	
description:	Defines the output clock ratio with respects to the operating clock: Frequency(Output Clock) = Frequency(Operating Clock)/Ratio.

void ana_SetOutputClock (bool Output)

parameters:	Output: boolean value (0 or 1).
returns:	
description:	Enables or disables the generation of an output clock. When Output is set to 1, the output clock is enabled; when Output is set to 0, the output clock is disabled.

void ana_SetReqClock (int Freq)

parameters:	Freq: integer value representing the requested frequency in Hz. Range: from 800 (800 Hz) to 50000000 (50 MHz) for GP-22050/Xpress and 100000000 (100MHz) for the GP-24100.
returns:	
description:	Defines the requested operating clock frequency. The device will actually set the operating clock frequency to the closest frequency available.
see also:	int GetReqClock(void), int GetSynthClock(void).

int ana_SetTriggerPos (int Sample)

parameters:	Sample: integer value representing the index of the sample in the run where the trigger should be positioned.
returns:	An integer error code: 0 is successful; another value if unsuccessful.
description:	Use this function to position the trigger after a given number of samples in the total run. Once the sampling run is over, the corresponding number of samples before the trigger is displayed, together with the rest of the run after the trigger.

void ana_Terminate (void)

parameters:	none
returns:	
description:	Closes an instance of the AnalyserC library and closes the communication with the device. A call to this function is mandatory for each instance of the library before closing the application.

void ana_Unsupervised (bool Enable)

parameters:	Enable: boolean – valid values: 0 or 1.
returns:	
description:	Enables or disables the unsupervised operating mode. In this mode, the traffic part reserved to the control between the host and the device is reduced to a minimum. As a consequence, more bandwidth is available for the data on the USB connection.

void ana_UseExtClock (void)

parameters:	none
returns:	
description:	Configures the device to operate with an externally supplied operating clock as reference.

void ana_UseIntClock (void)

parameters:	none
returns:	
description:	Configures the device to operate with the internal operating clock as reference.

short ana_Ver (void)

parameters:	none
returns:	A short integer value representing the software version.
description:	Returns the software version as a decimal value. Example: software version 1.04 ➔ returns 0x104 converted as a decimal value: 260.

int ana_WriteConfFile (char *FileName)

parameters:	*FileName: specifies the function output file name (including path).
returns:	0 if write successful; another value if write failed.
description:	Writes the device configuration to a file.

void ana_Abort (void)

parameters:	none
returns:	
description:	Aborts the pending run.