

**PA20 Series**  
**PA40 Series**

**Logic State Analyzers**

**Interface Library**  
**Programmer's Guide**



Copyright © 2000 SofTec Microsystems®

DC00231

## **SofTec Microsystems**

via Roma, 1

33082 Azzano Decimo (PN) ITALY

Tel: (+39) 0434 640 729

Fax: (+39) 0434 632 695

E-mail (general information): [info@softecmicro.com](mailto:info@softecmicro.com)

E-mail (marketing department): [marketing@softecmicro.com](mailto:marketing@softecmicro.com)

E-mail (technical support): [support@softecmicro.com](mailto:support@softecmicro.com)

Web: <http://www.softecmicro.com>

### **Important**

SofTec Microsystems reserves the right to make improvements to PA20 and PA40 Series Logic analyzers, their documentation and software routines, without notice. Information in this manual is intended to be accurate and reliable. However, SofTec Microsystems assumes no responsibility for its use; nor for any infringements of rights of third parties which may result from its use.

SOFTEC MICROSYSTEMS WILL NOT BE LIABLE FOR DAMAGES RESULTING FROM LOSS OF DATA, PROFITS, USE OF PRODUCTS, OR INCIDENTAL OR CONSEQUENTIAL DAMAGES, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.

### **Notice of Conformity**

This product is in conformity with the EN-55022, EN-50082-1, ENV-50140 specifications and therefore it complies with the EMC Directive 89/336/EEC.

### **Trademarks**

Product and company names listed are trademarks or trade names of their respective companies.

# Contents

<b>OVERVIEW .....</b>	<b>4</b>
NOTES .....	4
INSTALLATION .....	4
USING THE INTERFACE LIBRARY FUNCTIONS .....	4
RETURN VALUES OF INTERFACE LIBRARY FUNCTIONS.....	5
<b>LIBRARY FUNCTIONS REFERENCE.....</b>	<b>6</b>
<i>PA_CloseCommunication</i> .....	6
<i>PA_GetAcquisitionBufferSize</i> .....	6
<i>PA_GetAddressOnTheFly</i> .....	6
<i>PA_GetAnalyzerMode</i> .....	6
<i>PA_GetAnalyzerStatus</i> .....	7
<i>PA_GetExternalClock</i> .....	7
<i>PA_GetGlitch</i> .....	7
<i>PA_GetInstrumentID</i> .....	8
<i>PA_GetLPTBaseAddress</i> .....	8
<i>PA_GetNumAcquiredSamples</i> .....	8
<i>PA_GetNumLPTPorts</i> .....	8
<i>PA_GetPreTriggerBufferSize</i> .....	9
<i>PA_GetStorageFilter</i> .....	9
<i>PA_GetTimebase</i> .....	9
<i>PA_GetTrigger</i> .....	9
<i>PA_InitInstrument</i> .....	10
<i>PA_ReadBuffer</i> .....	10
<i>PA_SetAcquisitionBufferSize</i> .....	10
<i>PA_SetAnalyzerMode</i> .....	11
<i>PA_SetExternalClock</i> .....	11
<i>PA_SetPreTriggerBufferSize</i> .....	11
<i>PA_SetStorageFilter</i> .....	12
<i>PA_SetTimebase</i> .....	12
<i>PA_SetTrigger</i> .....	12
<i>PA_StartAcquisition</i> .....	13
<i>PA_StopAcquisition</i> .....	13
<b>ACQUISITION EXAMPLE.....</b>	<b>14</b>

# Overview

## Notes

This documentation deals with low level interfacing between user written PC applications and the PA20 and PA40 Series Logic State Analyzers. This documentation assumes you already read the PA20 and PA40 Series Logic State Analyzers User's Manual, provided with your instrument. All of the examples provided in this documentation are written in C, unless otherwise reported.

## Installation

The Communication DLL floppy disk contains:

- The pa24x.dll file (the main DLL file—must be copied into your <WINDOWS>\SYSTEM directory);
- The pa24x.lib file (the library file for Visual C applications—must be copied into your source code directory);
- The pa24x.h file (the include file for Visual C applications—must be copied into your source code directory);
- The pa24x.bas file (the include file for Visual Basic applications—must be copied into your source code directory).

The Communication DLL floppy disk also contains the following files (which are required for the low level control of the parallel port resources):

- tvicport.vxd;
- TVicPort.dll;
- tvicport.sys.

If you are working under Windows 95/98, just copy the files tvicport.vxd and TVicPort.dll into the <WINDOWS>\SYSTEM directory.

If you are working under Windows NT/2000, you must do the following:

1. Log in as Administrator;
2. Copy the file TVicPort.dll into the <WINNT>\SYSTEM directory;
3. Copy the file tvicport.sys into the <WINNT>\SYSTEM32\DRIVERS directory;
4. Create the following keys in the Registry:
  - HKEY\_LOCAL\_MACHINE\System\CurrentControlSet\Services\TvicPort, Key = "ErrorControl", Value = 0x00000001, Type = DWORD
  - HKEY\_LOCAL\_MACHINE\System\CurrentControlSet\Services\TvicPort, Key = "Type", Value = 0x00000001, Type = DWORD
  - HKEY\_LOCAL\_MACHINE\System\CurrentControlSet\Services\TvicPort, Key = "Start", Value = 0x00000002, Type = DWORD
  - HKEY\_LOCAL\_MACHINE\System\CurrentControlSet\Services\TvicPort, Key = "Group", Value = "Extended Base", Type = String
  - HKEY\_LOCAL\_MACHINE\System\CurrentControlSet\Services\TvicPort, Key = "Parameters", Value = "", Type = String
5. Reboot the PC.

## Using the Interface Library Functions

PA20 and PA40 Series Logic State Analyzers can be controlled without the PA24X user interface. The Communication DLL (131-00137) allows you to set up the instrument and perform data acquisition from within your application. The Communication DLL contains C written routines, and can be used to interface the analyzer from within, for example, a Microsoft® Visual C® or Visual Basic® application, as well as any other programming language that supports the DLL mechanism.

When you control the analyzer within your own application, you will follow the steps indicated below:

1. **Initialize the instrument.**  
To communicate with the analyzer you need to open a parallel port resource. Furthermore, to ensure consistent, repeatable performance, you need to initialize the analyzer in a known state. The initialization procedure must be done every time the instrument is powered on.
2. **Set up acquisition settings.**  
Before acquiring data, you must specify various analyzer's settings, such as acquisition mode (State/Timing), trigger specification, acquisition buffer length, etc.
3. **Acquire data.**

Once the instrument is set up and acquisition settings are specified, you can begin capturing data. Note that when you change the analyzer configuration, any data already captured is most likely invalid.

4. **Analyze data.**

After the analyzer has completed an acquisition, you can transfer the acquisition buffer contents from the analyzer's internal memory to a local buffer in the PC, and perform any kind of analysis on the data.

5. **Repeat the steps 2 through 4 as long as necessary.**

6. **Close the communication with the instrument.**

Closing the communication with the instrument frees the parallel port resource used during communication.

## Return Values of Interface Library Functions

Most of the Interface Library functions return a BOOL value which indicates whether the function has been successfully executed (return value = **TRUE**) or not (return value = **FALSE**). In the latter case it is possible to get extended error information by calling the function **PA\_GetAnalyzerError**:

```
PA_GetAnalyzerError (unsigned char *error);
```

The following table summarizes the possible error codes.

Error Code	Description
ERR_S00	Cannot establish bidirectional communication. The parallel port must be configured as bidirectional.
ERR_S01	Instrument not detected. Make sure that both the parallel cable is connected and the analyzer is turned on and retry.
ERR_S02	Power supply off. Make sure that both the parallel cable is connected and the analyzer is turned on and retry.
ERR_S03	Instrument's communication controller is not responding.
ERR_S04	PC Parallel port not compatible with the instrument.
ERR_S05 to ERR_S08	Instrument configuration error.
ERR_S09	Data transfer error. Try again and, if the trouble persists, use another parallel port.
ERR_S0A	Instrument RAM error.
PA_ERR_IODRIVER	I/O communication driver error.
PA_ERR_LOSSCFG	Instrument configuration lost. Make sure the power supply is provided correctly.
PA_ERR_ALREADY RUN	A command has been requested while the instrument was acquiring data.
PA_ERR_INV_TBASE	Invalid timebase setting.
PA_ERR_MEMORYSIZE	Invalid acquisition buffer size.
PA_ERR_TRIGGERPOS	Invalid trigger position setting. The trigger position setting is greater than the acquisition buffer.
PA_ERR_READPARAM	The <b>PA_ReadBuffer</b> function tried to read data from an acquisition buffer position greater than the actual acquisition buffer length. Use the <b>PA_GetNumAcquiredSamples</b> function to determine the actual number of samples actually stored.
PA_ERR_INV_CHANNEL	Invalid channel specification. Channel specification range from 0 to 15 for 16-channel analyzer models, and from 0 to 31 for 32-channel analyzer models.
PA_ERR_FILT_LEV	Invalid storage filters level. See the <b>PA_SetStorageFilter</b> function.
PA_ERR_FILT_OP	Invalid storage filters operator. See the <b>PA_SetStorageFilter</b> function.
PA_ERR_INV_ANALYZERMODE	Invalid analyzer mode specification. See the <b>PA_SetAnalyzerMode</b> function.
PA_ERR_INV_CLKEDGE	Invalid external clock edge specification. See the <b>PA_SetExternalClock</b> function.
PA_ERR_INV_PATTCHR	Invalid trigger pattern specification. See the <b>PA_SetTrigger</b> function.
PA_ERR_INV_EDGECHR	Invalid trigger edge specification. See the <b>PA_SetTrigger</b> function.
PA_ERR_TRG_MODE	Invalid trigger mode setting. See the <b>PA_SetTrigger</b> function.
PA_ERR_TRG_OP	Invalid trigger operator setting. See the <b>PA_SetTrigger</b> function.
PA_ERR_TRG_SRC	Invalid trigger source type setting. See the <b>PA_SetTrigger</b> function.
PA_ERR_TRG_CNT	Invalid trigger source occurrence setting. See the <b>PA_SetTrigger</b> function.

# Library Functions Reference

## PA\_CloseCommunication

**Include file:**

```
#include "pa24x.h"
```

**Function prototype:**

```
BOOL PA_CloseCommunication (void);
```

**Return value:**

**TRUE:** the function was successful.

**FALSE:** an error occurred. Call the **PA\_GetAnalyzerError** function to get extended error information.

**Description:**

Closes the communication with the instrument and frees the parallel port resource used during communication.

## PA\_GetAcquisitionBufferSize

**Include file:**

```
#include "pa24x.h"
```

**Function prototype:**

```
BOOL PA_GetAcquisitionBufferSize (unsigned long *size);
```

**Return value:**

**TRUE:** the function was successful.

**FALSE:** an error occurred. Call the **PA\_GetAnalyzerError** function to get extended error information.

**Description:**

Gets the current acquisition buffer length (in samples) setting. The **size** variable will contain the setting. To change the acquisition buffer length, use the **PA\_SetAcquisitionBufferSize** function.

## PA\_GetAddressOnTheFly

**Include file:**

```
#include "pa24x.h"
```

**Function prototype:**

```
BOOL PA_GetAddressOnTheFly (unsigned long *addr);
```

**Return value:**

**TRUE:** the function was successful.

**FALSE:** an error occurred. Call the **PA\_GetAnalyzerError** function to get extended error information.

**Description:**

During an acquisition, the acquisition buffer is filled with the sampled data. The first location on the acquisition buffer to be written is the location 0, and successive write operations occur to locations 1, 2, 3, up to the acquisition buffer length. Call the **PA\_GetAddressOnTheFly** function during an acquisition to get the acquisition buffer location that is being written. The **addr** variable will contain a value between 0 and the current acquisition buffer length. You can use this function to display a progress bar while the instrument is acquiring data (especially at high values of the timebase setting).

## PA\_GetAnalyzerMode

**Include file:**

```
#include "pa24x.h"
```

**Function prototype:**

```
BOOL PA_GetAnalyzerMode (int *mode);
```

**Return value:**

**TRUE:** the function was successful.

**FALSE:** an error occurred. Call the **PA\_GetAnalyzerError** function to get extended error information.

**Description:**

Retrieves the current analyzer acquisition mode. The **mode** variable will assume the values **PA\_TIMING\_MODE** or **PA\_STATE\_MODE**. To set the analyzer acquisition mode, use the **PA\_SetAnalyzerMode** function.

## PA\_GetAnalyzerStatus

**Include file:**

```
#include "pa24x.h"
```

**Function prototype:**

```
BOOL PA_GetAnalyzerStatus (unsigned char *status);
```

**Return value:**

**TRUE:** the function was successful.

**FALSE:** an error occurred. Call the **PA\_GetAnalyzerError** function to get extended error information.

**Description:**

Retrieves the current analyzer acquisition status. The **status** variable can assume one of the following values: **PA\_S\_STOP** (the analyzer is not acquiring data), **PA\_S\_PREFILL** (the analyzer is filling the pre-trigger buffer), **PA\_S\_TRGWAIT** (the analyzer has filled the pre-trigger buffer and is waiting for a trigger event to occur), **PA\_S\_STORE** (a trigger event has been detected and the analyzer is now filling the post-trigger buffer).

## PA\_GetExternalClock

**Include file:**

```
#include "pa24x.h"
```

**Function prototype:**

```
BOOL PA_GetExternalClock (int *ch, int *edge);
```

**Return value:**

**TRUE:** the function was successful.

**FALSE:** an error occurred. Call the **PA\_GetAnalyzerError** function to get extended error information.

**Description:**

Retrieves the current state mode (external clock) properties. The **ch** variable will contain the input channel which provides the sampling clock, and the **edge** variable will contain the sampling clock edge. See also the **PA\_SetExternalClock** function.

## PA\_GetGlitch

**Include file:**

```
#include "pa24x.h"
```

**Function prototype:**

```
BOOL PA_GetGlitch (unsigned long *glitch);
```

**Return value:**

**TRUE:** the function was successful.

**FALSE:** an error occurred. Call the **PA\_GetAnalyzerError** function to get extended error information.

**Description:**

Call this function after an acquisition in Timing mode has been performed to get glitch information. The binary representation of the contents of the **glitch** variable contains a 1 in correspondence with the position of the channels on which one or more glitches were detected (input channel 0 is the least significant bit). For example, to know if channel **n** contains glitches, use the following test expression: **if (glitch & (0x00000001 << n)) { /\* do something \*/ }**.

**Notes:**

This function can only be called when the analyzer is in Stop mode.

## PA\_GetInstrumentID

**Include file:**

```
#include "pa24x.h"
```

**Function prototype:**

```
BOOL PA_GetInstrumentID (unsigned char *id);
```

**Return value:**

**TRUE:** the function successfully detected the analyzer model.

**FALSE:** an error occurred. Call the **PA\_GetAnalyzerError** function to get extended error information.

**Description:**

Gets the analyzer model connected to the parallel port. Depending on the analyzer model detected, the **id** variable will assume the following values: 0 = PA2016A, 1 = PA2032A, 2 = PA4016A, 3 = PA4032A.

## PA\_GetLPTBaseAddress

**Include file:**

```
#include "pa24x.h"
```

**Function prototype:**

```
unsigned int PA_GetLPTBaseAddress (unsigned char nLPT);
```

**Return value:**

The base address for the specified parallel port.

**Description:**

Gets the base address for the specified parallel port. This value can in turn be used with the **PA\_InitInstrument** function to initialize the instrument connected to the parallel port. The **nLPT** parameter is 1 for LPT1, 2 for LP2, etc. To get the number of parallel ports present in your system, call the **PA\_GetNumLPTPorts** function.

## PA\_GetNumAcquiredSamples

**Include file:**

```
#include "pa24x.h"
```

**Function prototype:**

```
BOOL PA_GetNumAcquiredSamples (unsigned long *nsamples);
```

**Return value:**

**TRUE:** the function was successful.

**FALSE:** an error occurred. Call the **PA\_GetAnalyzerError** function to get extended error information.

**Description:**

Gets the number of actual samples stored in the acquisition buffer. Since an acquisition can be stopped with the **PA\_StopAcquisition** function, the number of valid samples stored in the acquisition buffer may be less than the acquisition buffer length. Always use this function before to call the **PA\_ReadBuffer** function.

**Notes:**

This function can only be called when the analyzer is in Stop mode.

## PA\_GetNumLPTPorts

**Include file:**

```
#include "pa24x.h"
```

**Function prototype:**

```
int PA_GetNumLPTPorts (void);
```

**Return value:**

The number of parallel ports present in your system.

**Description:**

Gets the number of parallel ports present in your system. This value can in turn be used with the **PA\_GetLPTBaseAddress** function.

## PA\_GetPreTriggerBufferSize

**Include file:**

```
#include "pa24x.h"
```

**Function prototype:**

```
BOOL PA_GetPreTriggerBufferSize (unsigned long *size);
```

**Return value:**

**TRUE:** the function was successful.

**FALSE:** an error occurred. Call the **PA\_GetAnalyzerError** function to get extended error information.

**Description:**

Retrieves the current pre-trigger buffer size (in samples). To set the pre-trigger buffer size (i.e., the trigger position) use the **PA\_SetPreTriggerBufferSize** function.

## PA\_GetStorageFilter

**Include file:**

```
#include "pa24x.h"
```

**Function prototype:**

```
BOOL PA_GetStorageFilter (PA_FILTERS *f);
```

**Return value:**

**TRUE:** the function was successful.

**FALSE:** an error occurred. Call the **PA\_GetAnalyzerError** function to get extended error information.

**Description:**

Retrieves information about the currently used storage filters. See also the **PA\_SetStorageFilter** function.

**Notes:**

Storage filters work both in Timing and in State mode.

## PA\_GetTimebase

**Include file:**

```
#include "pa24x.h"
```

**Function prototype:**

```
BOOL PA_GetTimebase (unsigned long *time_ns);
```

**Return value:**

**TRUE:** the function was successful.

**FALSE:** an error occurred. Call the **PA\_GetAnalyzerError** function to get extended error information.

**Description:**

Retrieves the current timebase setting. See also the **PA\_SetTimebase** function.

## PA\_GetTrigger

**Include file:**

```
#include "pa24x.h"
```

**Function prototype:**

```
BOOL PA_GetTrigger (PA_TRIGGER *trg);
```

**Return value:**

**TRUE:** the function was successful.

**FALSE:** an error occurred. Call the **PA\_GetAnalyzerError** function to get extended error information.

**Description:**

Retrieves the current trigger settings. See also the **PA\_SetTrigger** function.

## PA\_InitInstrument

### Include file:

```
#include "pa24x.h"
```

### Function prototype:

```
BOOL PA_InitInstrument (unsigned int boardaddr, unsigned char ramtest, PROGRESSPROC *callback);
```

### Return value:

**TRUE:** the instrument has been successfully initialized.

**FALSE:** an error occurred. Call the **PA\_GetAnalyzerError** function to get extended error information.

### Description:

Initializes the instrument connected to the specified parallel port.

The **boardaddr** parameter is the parallel port base address, obtained with the **PA\_GetLPTBaseAddress** function.

The **ramtest** parameter specifies whether or not to perform the instrument's RAM self-check (0 = test is not performed, 1 = test is performed).

The **callback** parameter specifies the address of a callback function which will get the progress (an integer from 0 to 100) of the initialization procedure. You can use this feature to display a progress bar while the instrument is being initialized. If this feature is not used, the **callback** parameter must be **NULL**.

If the initialization is successful, the instrument is configured with the following settings: acquisition mode = Timing, Timebase = 25 ns (for 20 MSa/s analyzer models) or 50 ns (for 40 MSa/s analyzer models), acquisition memory = 1K samples, trigger position = 512, storage filters disabled, edge trigger (CHO, rising), trigger mode Auto.

## PA\_ReadBuffer

### Include file:

```
#include "pa24x.h"
```

### Function prototype:

```
BOOL PA_ReadBuffer (unsigned long start_addr, unsigned long *data_buf, unsigned long size);
```

### Return value:

**TRUE:** the function was successful.

**FALSE:** an error occurred. Call the **PA\_GetAnalyzerError** function to get extended error information.

### Description:

Reads the acquisition buffer after an acquisition has been completed. The **start\_addr** parameter must be greater than or equal to 0 and must not exceed the current acquisition buffer length - 1. The **size** parameter is the number of samples to read (at least one sample must be read). The sum **start\_addr + size** must not exceed the current acquisition buffer length - 1. To get the current acquisition buffer length, use the **PA\_GetAcquisitionBufferSize** function.

Acquisition data is transferred to the PC and is stored on the **data\_buf** buffer, which must be large enough to contain all the requested samples. Each sample is stored as an unsigned long value; the binary representation of this value contains a 1 if the corresponding input channel has been sampled at a high logic value, 0 otherwise (input channel 0 is the least significant bit). For example, to get the value of channel **n** at the **s** sample, use the following test expression:

```
logic_value = data_buf [s] & (0x00000001 << n).
```

### Notes:

This function can only be called when the analyzer is in Stop mode.

## PA\_SetAcquisitionBufferSize

### Include file:

```
#include "pa24x.h"
```

### Function prototype:

```
BOOL PA_SetAcquisitionBufferSize (unsigned long size);
```

### Return value:

**TRUE:** the function was successful.

**FALSE:** an error occurred. Call the **PA\_GetAnalyzerError** function to get extended error information.

### Description:

Sets the new acquisition buffer length (in samples) setting. The **size** variable must be in the range 1...131056. To get the current acquisition buffer length, use the **PA\_GetAcquisitionBufferSize** function.

**Notes:**

This function can only be called when the analyzer is in Stop mode.

## PA\_SetAnalyzerMode

**Include file:**

```
#include "pa24x.h"
```

**Function prototype:**

```
BOOL PA_SetAnalyzerMode (int mode);
```

**Return value:**

**TRUE:** the function was successful.

**FALSE:** an error occurred. Call the **PA\_GetAnalyzerError** function to get extended error information.

**Description:**

Sets the current analyzer acquisition mode. The **mode** variable must be **PA\_TIMING\_MODE** or **PA\_STATE\_MODE**. To get the current analyzer acquisition mode, use the **PA\_GetAnalyzerMode** function.

**Notes:**

This function can only be called when the analyzer is in Stop mode.

## PA\_SetExternalClock

**Include file:**

```
#include "pa24x.h"
```

**Function prototype:**

```
BOOL PA_SetExternalClock (int ch, int edge);
```

**Return value:**

**TRUE:** the function was successful.

**FALSE:** an error occurred. Call the **PA\_GetAnalyzerError** function to get extended error information.

**Description:**

Sets the current state mode (external clock) properties. The **ch** variable must contain the input channel which provides the sampling clock (in the range 0...15 for 16-channel analyzer models and 0...31 for 32-channel analyzer models), and the **edge** variable must contain the sampling clock edge. Valid values for the edge setting are **PA\_CLK\_EDGE\_RISE**, **PA\_CLK\_EDGE\_FALL**, **PA\_CLK\_EDGE\_BOTH**.

**Notes:**

This function can only be called when the analyzer is in Stop mode. This function does not change the current analyzer acquisition mode. To change the acquisition mode, call the **PA\_SetAnalyzerMode** function.

## PA\_SetPreTriggerBufferSize

**Include file:**

```
#include "pa24x.h"
```

**Function prototype:**

```
BOOL PA_SetPreTriggerBufferSize (unsigned long size);
```

**Return value:**

**TRUE:** the function was successful.

**FALSE:** an error occurred. Call the **PA\_GetAnalyzerError** function to get extended error information.

**Description:**

Sets the current pre-trigger buffer size (i.e., the trigger position). The **size** parameter represents the number of samples contained in the pre-trigger buffer, and ranges between 0 and the current acquisition buffer size. To set/get the acquisition buffer size, use the **PA\_SetAcquisitionBufferSize** and **PA\_GetAcquisitionBufferSize** functions.

**Notes:**

This function can only be called when the analyzer is in Stop mode.

## PA\_SetStorageFilter

### Include file:

```
#include "pa24x.h"
```

### Function prototype:

```
BOOL PA_SetStorageFilter (PA_FILTERS *f);
```

### Return value:

**TRUE:** the function was successful.

**FALSE:** an error occurred. Call the **PA\_GetAnalyzerError** function to get extended error information.

### Description:

Sets storage filter settings. The **PA\_FILTERS** structure is defined as follows:

### typedef struct

```
{
    int Ch0;      /* F1 source */
    int Level0;   /* F1 level */
    int Op;       /* Operator (PA_SFILT_AND, PA_SFILT_OR, PA_SFILT_DISABLE) */
    int Ch1;      /* F2 source */
    int Level1;   /* F2 level */
} PA_FILTERS;
```

**Ch0** and **Level0** fields describe Storage Filter 1 (F1); **Ch1** and **Level1** fields describe Storage Filter 2 (F2). The **Level0** and **Level1** fields must be set to 0 (level low) or 1 (level high). The **Op** field is the logical operation to be performed on F1 and F2 in order to get the final storage condition and can be either **PA\_SFILT\_AND** or **PA\_SFILT\_OR**. To disable storage filters, set the **Op** field to **PA\_SFILT\_DISABLE**.

### Notes:

This function can only be called when the analyzer is in Stop mode. Storage filters work both in Timing and in State mode.

## PA\_SetTimebase

### Include file:

```
#include "pa24x.h"
```

### Function prototype:

```
BOOL PA_SetTimebase (unsigned long time_ns);
```

### Return value:

**TRUE:** the function was successful.

**FALSE:** an error occurred. Call the **PA\_GetAnalyzerError** function to get extended error information.

### Description:

Sets the current timebase, i.e. the sampling period used when acquiring data in Timing mode. The **time\_ns** parameter is the sampling period in ns, and ranges from 0 to 3276800 in steps of 50 ns (in 20 MSa/s analyzer models) or from 0 to 1638400 ns in steps of 25 ns (in 40 MSa/s analyzer models).

### Notes:

This function can only be called when the analyzer is in Stop mode.

## PA\_SetTrigger

### Include file:

```
#include "pa24x.h"
```

### Function prototype:

```
BOOL PA_SetTrigger (PA_TRIGGER *trg);
```

### Return value:

**TRUE:** the function was successful.

**FALSE:** an error occurred. Call the **PA\_GetAnalyzerError** function to get extended error information.

### Description:

Sets trigger settings. The `PA_TRIGGER` structure is defined as follows:

```
typedef struct
{
    int Mode; /* Trigger mode (PA_TRG_AUTO, PA_TRG_NORM) */
    int Src0Type; /* Source1 type (PA_TRG_PATTERN, PA_TRG_EDGE) */
    char Src0[33]; /* Source1 specification */
    int Operator; /* PA_TRG_AND, PA_TRG_OR, PA_TRG_THEN,
PA_TRG_OCCURED */
    int Src1Type; /* Source2 type (PA_TRG_PATTERN, PA_TRG_EDGE) */
    char Src1[33]; /* Source2 specification */
    unsigned int EvCounter; /* Counter to be used with PA_TRG_OCCURED operator */
} PA_TRIGGER;
```

`Src0Type` and `Src0` fields describe Source1; `Src1Type` and `Src1` fields describe Source2. Each `Src` field is a 33-characters long, null terminated string (32 characters for each input channel plus one character for the `NULL` character; the character at position 0 corresponds to the input channel 0). Depending on the `SrcType` parameter, the corresponding `Src` string must be initialized properly. If the `SrcType` parameter defines an edge (`PA_TRG_EDGE`), each character of the corresponding `Src` string must be `'R'` (rising edge), `'F'` (falling edge), `'B'` (both edges) or `'X'` (don't care). If the `SrcType` parameter defines a pattern (`PA_TRG_PATTERN`), each character of the corresponding `Src` string must be `'0'` (level low), `'1'` (level high) or `'X'` (don't care). The `Operator` parameter specifies the logical operation to be performed on `Src0` and `Src1` in order to get the final trigger specification and can be `PA_TRG_AND`, `PA_TRG_OR`, `PA_TRG_THEN` or `PA_TRG_OCCURED`. In the latter case, `Src0` must be a valid source specification, while `EvCounter` must specify the number of times the `Src0` must be recognized in order to generate a trigger event (`Src1` is ignored). `EvCounter` must be in the range 0...65535.

#### Notes:

This function can only be called when the analyzer is in Stop mode.

## PA\_StartAcquisition

#### Include file:

```
#include "pa24x.h"
```

#### Function prototype:

```
BOOL PA_StartAcquisition (void);
```

#### Return value:

**TRUE:** the function was successful.

**FALSE:** an error occurred. Call the `PA_GetAnalyzerError` function to get extended error information.

#### Description:

Starts a single shot acquisition. When the analyzer starts acquiring data, its internal states change to `PA_S_PREFILL`, `PA_S_TRGWAIT`, `PA_S_STORE`. When the analyzer finishes acquiring data, it goes in its internal state `PA_S_STOP`. Use the `PA_GetAnalyzerStatus` function to get information about the progress of the acquisition process.

#### Notes:

This function can only be called when the analyzer is in Stop mode.

## PA\_StopAcquisition

#### Include file:

```
#include "pa24x.h"
```

#### Function prototype:

```
BOOL PA_StopAcquisition (void);
```

#### Return value:

**TRUE:** the function was successful.

**FALSE:** an error occurred. Call the `PA_GetAnalyzerError` function to get extended error information.

#### Description:

Stops the current acquisition. To retrieve how many samples have been stored, use the `PA_GetNumAcquiredSamples` function.

# Acquisition Example

```
/* Acquisition mode = Timing */
/* Timebase = 100 ns */
/* Acquisition buffer = 4K Samples */
/* Pre-trigger buffer = 2K Samples */
/* Storage filters = A5 low */
/* Trigger mode = AUTO */
/* Trigger specification = rising edge of CH0 OR low level of CH2 */

#include "pa24x.h"
#define LPT1 1

/* Analyzer initialization */
BOOL init (void)
{
    PA_FILTERS f;
    PA_TRIGGER t;

    if (!PA_InitInstrument (PA_GetLPTBaseAddress (LPT1), TRUE, NULL))
        return FALSE;
    if (!PA_SetAnalyzerMode (PA_TIMING_MODE))
        return FALSE;
    if (!PA_SetTimebase (100))
        return FALSE;
    if (!PA_SetGlobalMemorySize (4096))
        return FALSE;
    if (!PA_SetPreTriggerPos (2048))
        return FALSE;

    f.Ch0 = 5;
    f.Ch1 = 5;
    f.Level0 = 0;
    f.Level1 = 0;
    f.Op = PA_SFILT_OR;
    if (!PA_SetStorageFilter (&f))
        return FALSE;
    t.Mode = PA_TRG_AUTO;
    t.Src0Type = PA_TRG_EDGE;
    strcpy (t.Src0, "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXR");
    t.Src1Type = PA_TRG_PATTERN;
    strcpy (t.Src1, "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXLXX");
    t.Operator = PA_TRG_OR;
    t.EvCounter = 1;
    if (!PA_SetTrigger (&t))
        return FALSE;
    return TRUE;
}

/* Reads the content of the analyzer acquisition buffer */
/* and puts it in a PC global buffer */
BOOL read_acquisition (unsigned long *mem_buf)
{
    unsigned char status;
    unsigned long nbytes;

    if (!PA_StartAcquisition ())
        return FALSE;

    do
    {
        if (!PA_GetAnalyzerStatus (&status))
```

```

        return FALSE;
    }
    while (!(status == PA_S_STOP));
    if (!PA_GetNBytesAcquired (&nbytes))
        return FALSE;
    if (!PA_ReadMemory (0, mem_buf, nbytes))
        return FALSE;
    return TRUE;
}

/* Main */
void main(void)
{
    unsigned char err;
    unsigned long mem_buffer [4096];

    if(!init ())
    {
        CmdGetAnalyzerError (&err);
        /* To do: display error code */
    }
    else
    {
        while (1)
        {
            if (!read_acquisition (mem_buffer))
            {
                CmdGetAnalyzerError (&err);
                /* To do: display error code */
            }
            else
            {
                /* To do: display waveform data */
            }
        }
    }
}

```