Help Volume

Utilities: File Out Tool

Using the File Out Tool

The File Out tool lets you save measurement data to a file in ASCII, Internal or Fast Binary format (see page 12). You can then reload the file using the *File In* tool, or export the data file to a debugger or spreadsheet application for post-processing.

See Also

File In Tool (see the *File In Tool* help volume)

"Fast Binary Data File Format" on page 16

Main System Help (see the *Agilent Technologies 16700A/B-Series Logic Analysis System* help volume)

Glossary of Terms (see page 35)

Contents

Contents

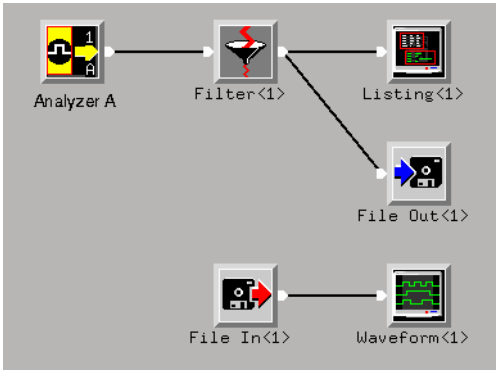
1

Using the File Out Tool

Overview of File Out Tool

The following example shows a measurement configuration using both the File In and File Out tools. Using the File Out tool allows you to save files that can be reopened for display and post-process at a later time using the File In tool. The sequence of events for the following configuration is as follows.

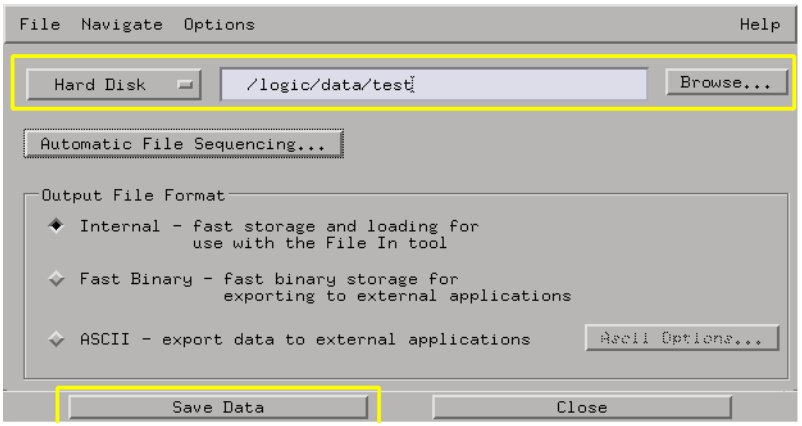
- An analyzer (Analyzer A) is configured to trigger and capture data.
- The Pattern Filter is configured to filter out unwanted data.
- The remaining data is displayed in the Listing tool.
- The same data is saved to a file using the File Out tool.
- The File In tool reloads the saved file from the File Out tool, and displays the data in the Waveform tool.



Saving a Single Data File

The File Out tool saves the measurement data that is currently available at its input to a file. New measurement data is presented to the File Out tool for saving each time you press *Run*. Use the process below to save the data file.

1. In the Workspace window, connect the File Out tool to the output of the analyzer providing data.
2. Run the analyzer to capture data.
3. Access the File Out window by selecting the Navigate menu.
4. In the File Out window, select hard disk or flexible disk as the data file's destination.
5. Type the destination path, or use "Browse..." to select a directory in the file system.
6. Select the desired Output File Format (see page 12).
7. If you choose ASCII, configure the desired ASCII Options (see page 15).
8. Select *Save Data*.



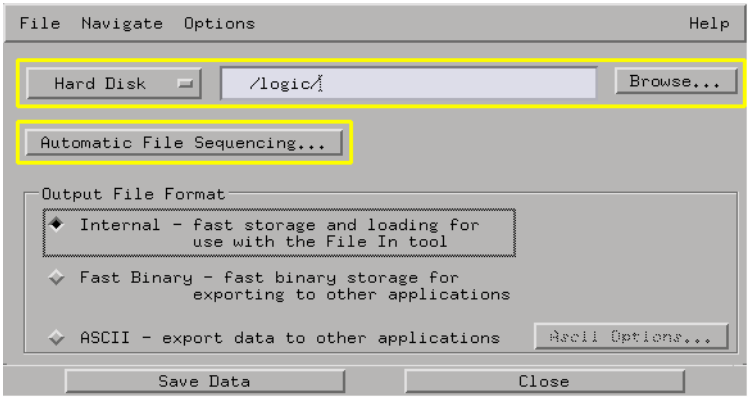
See Also "Saving Multiple Data Files" on page 9

"Overview of File Out Tool" on page 6

Saving Multiple Data Files

New measurement data is presented to the File Out tool each time you press *Run*. *Automatic File Sequencing* allows you to save File Out data at each run without pressing *Save Data*.

1. Set up the File Out tool as described in "Saving a Single Data File" on page 7.
2. Select Automatic File Sequencing... (see page 10) and configure the desired file saving sequence.



See Also

"Saving a Single Data File" on page 7

Automatic Sequencing

To control how the File Out tool will save data files, choose one of these options in the Automatic Sequencing dialog.

No automatic data saving with every run

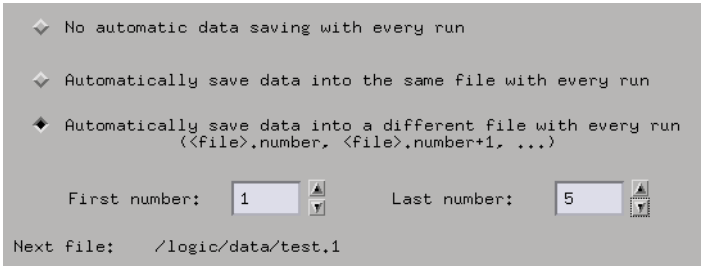
To save current data to the specified file, select *Save Data* in the File Out tool's main window.

Automatically save data to the same file with every run

At every run, data is automatically saved to the filename specified in the File Out tool main window. If you do not change the name of the file, the data in the file will be overwritten with the next run's data. You do *not* need to press *Save Data* for this option.

Automatically save data into a different file with every run

This selection automatically saves one file per run, for the selected number of runs. The *First Number* field indicates the number of the next run to be saved. The *Last Number* field sets the maximum number of runs that are saved. As files are saved, the filename is incremented by one. You do *not* need to select *Save Data* for this option. See Example of a File Sequence (see page 11)



Example of a saved ASCII data file (see page 11)

File Sequence Example

This example shows a list of generated files in a directory when the *First Number* was set to 1, and the *Last Number* was set to 5.

Example

data_file.1
data_file.2
data_file.3
data_file.4
data_file.5

ASCII Data File Example

This example shows a saved ASCII data file:

```
16505_Data_Header_Begin
Frame 5:Slot B:MACHINE 1:State Number
Decimal
32
Frame 5:Slot B:MACHINE 1:count
Hex
8
16505_Data_Header_End
1          00000000
2          00000001
3          00000010
4          00000011
5          00000100
6          00000101
7          00000110
8          00000111
9          00001000
10         00001001
11         00001010
12         00001011
13         00001100
14         00001101
15         00001110
```

Loading & Saving File Out Configurations

File Out settings are saved to a configuration file along with system settings.
See:

- Loading Configuration Files (see the *Agilent Technologies 16700A/B-Series Logic Analysis System* help volume)
- Saving Configuration Files (see the *Agilent Technologies 16700A/B-Series Logic Analysis System* help volume)

NOTE: The *Load Configuration* window can be accessed via File->Load Configuration.
The *Save Configuration* window can be accessed via File->Save Configuration.

File Out Data Formats

The File Out tool saves data in Internal, ASCII or Fast Binary format. Data files saved with the File Out tool retain their original format when loaded by the File In tool.

Internal The *Internal* format is a normalized data type used internally by the logic analysis system. Normalized data maintains its time correlation and alignment when used within the analyzer tools. The *Internal* format is recommended when used within the analyzer because of its fast storage and retrieval speed. Also, symbols you have created are stored with the data.

ASCII The *ASCII* format (American Standard Code for Information Interchange), is a common format for exporting data. The ASCII format is recommended if you want to export data to external tools such as debuggers or spreadsheets. ASCII format files require data conversion

and result in comparatively larger files. ASCII format is not recommended for use within the analyzer system when storage/ retrieval speed is an issue.

Fast Binary

The Fast Binary data format lets you export analyzer data for post-processing. The Fast Binary data format saves faster than ASCII, and can be parsed using programming tools.

NOTE: It is recommended that you save ASCII data in hexadecimal format. Hexadecimal makes most efficient use of the ASCII 8-bit format. Binary data storage uses an 8-bit ASCII string to represent each character, thus increasing the file size by a factor of 8.

Printing the File Out Window

The print window lets you print just the File Out tool window. Use this operation if you want a hard copy or electronic record of configurations and data currently displayed in the viewing area of the File Out window.

NOTE: Only the currently displayed viewing area of the File Out window is printed. If any data or configuration fields appear off the screen, scroll the desired data or configuration fields into the window's viewing area before printing.

1. Optional - configure the Print Options (see the *Agilent Technologies 16700A/B-Series Logic Analysis System* help volume) if desired. Print Options includes print destination, file format type, filename autoincrement, and color/b&w; pixel mapping.
2. In the File Out tool menu bar, select *File*, then select *Print This Window*. The print output will be as configured in the Print Options in step 1.

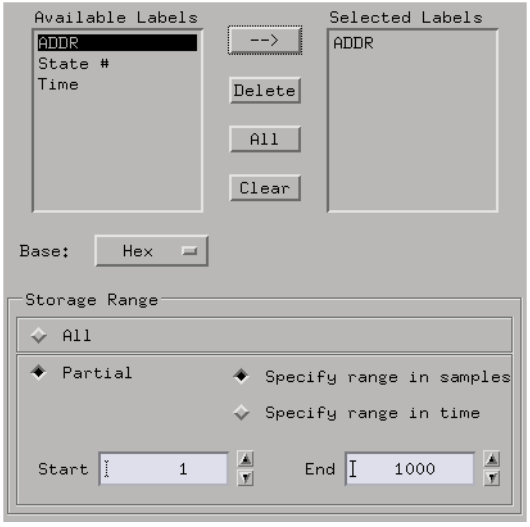
See Also

Setting Print Options (see the *Agilent Technologies 16700A/B-Series Logic Analysis System* help volume)

Set Up the Printer (see the *Agilent Technologies 16700A/B-Series Logic Analysis System* help volume)

Setting the ASCII Options

1. From the ASCII Output Options dialog, select the desired labels, then select the right-arrow field. This places the desired labels in the *Selected Labels* box.
2. Select the numeric base field and select the desired base. See note below.
3. Select either *All* or *Partial* as the Storage Range.
4. If you select *Partial*, choose either *Samples* or *Time* as the data unit.
5. If you select *Partial*, edit the *Start* and *End* value fields to set the partial storage range.



NOTE: It is recommended that you save ASCII data in hexadecimal format. Hexadecimal makes most efficient use of the ASCII 8-bit format. Compared to hex, storing data in binary means each character is represented as an 8-bit ASCII string, thus increasing the file size by a factor of 8.

Fast Binary Data File Format

These pages describe the Fast Binary File Format that you can use to quickly export and import data for use in post-processing environments.

To print a hard copy, select File->Print, then Current and Subtopics.

A "C" program containing parse code for the Fast Binary Data File Format is located on your system at: /logic/demo/fastOutReader/fast_reader.c

Data Organization

The data is organized in a hierarchy of several individual objects representing different aspects of the acquired data. The analyzer data is basically organized by your defined 'labels' and channels called a LabelEntry. Multiple labels/channels from each analyzer is then grouped into a DataSet. All the labels in a DataSet have the same number of samples and have the same X-axis(Abscissa) values. Then one or more DataSets are grouped into a DataGroup. These are discussed in more detail below. The FileOut tool generates or writes out one DataGroup object.

Object IDs

There are several object types that have an object id associated with them. Some objects are referenced more than once within the DataGroup. Where this happens on the first occurrence of an object is written out, and subsequent objects have only their id written out.

The Fast Binary File Format is a mixture of these binary and ascii objects:

- "DataGroup" on page 17
- "DataSet" on page 18

DataGroup

The top level object is the DataGroup. This object contains a general header information and a list of DataSet objects:

```
<DataGroup> ::=
    <DataGroup Header>
    Number of DataSets => "%d\n"
    <DataSet #1>
    <DataSet #2>
        ...
```

DataGroup Header

The DataGroup header is an object that identifies the file as a fast binary data file. It contains the format version and the information that describes how the DataSets are related to each other:

```
<DataGroup Header> ::=
    FileId => "HPLogic_Fast_Binary_Format_Data_File\n"
    FileVersion => "%d %d\n"
    Correlation Bits  => "%d [%d]*\n"
    Cross Correlation id's
        => "%d %d\n"
```

- FileVersion - The file version fields represent the major and minor version numbers. These are used to identify different versions of the fast data format. Where the format has changed will be noted with the corresponding version numbers.
- Correlation Bits - The cross correlation id's are used when correlating multiple DataGroups. DataGroups that are time correlatable will have the same Time Correlation Id. In the same way, DataGroups that are state correlatable will have the same State Correlation Id. An id of -1 indicates cross correlation information is not available.

```
Bit#     Description
 0       Time Correlatable
 1       State Correlatable
```

Time correlatable indicates that the data within all the DataSets are time aligned. State correlatable indicates that each sample is state aligned across all DataSets, i.e. the DataSets are sample synchronized. A DataGroup with only one DataSet that has timing information will be both time and state correlatable.

- Cross Correlation ID's - The cross correlation id's are used when correlating multiple DataGroups. DataGroups that are time correlatable will have the same Time Correlation Id. In the same way, DataGroups that are state correlatable will have the same State Correlation Id. An id of -1 indicates cross correlation information is not available.

DataSet

The DataSet is a collection of label or channels. All of the labels in a dataset have a common X-axis(Abscissa) information.

```
<DataSet> ::=
    Number of Label Entries  => "%d\n"
    <Label Entry #1>
       . . .
    <Label Entry #n>
    <Abscissa Data Type>
    <Time Correlation Info>
    <State Correlation Info>
    Origin Path  => "`%s'\n"
    DataSet ID, Run ID
                 => "%d %d\n"
    Begin and End Time
                 => "%d %d\n"
    Start Sample
                 => "%d\n"
    Last Sample
                 => "%d\n"
```

- Origin Path - This is a string that describes the "path" that the data was processed through, i.e. what tools were used to process this data.
- DataSet ID, Run ID - These id's are used to cross correlate and integrate multiple data sets. The DataSet ID identifies this data set. The Run ID is an id used to correlate with other data sets. The way it is used is that if two or more data sets have the same Run ID, then they have the possibility of being correlated in some way. To determine if the data sets are truly correlated, you must examine their Time Correlation Info (see page 30) and State Correlation Info (see page 31).

- Begin and End Time - These values show the time at which the data was acquired, both the time when the analysis began, and the time at which the trace ends. The time value is the number of seconds since midnight, January 1st, 1970.
- Start Sample - This value is the first sample number in the acquisition.
- Last Sample - This value is the last sample number in the acquisition.

IntegralData

The IntegralData structure is used to represent signed and unsigned n-bit data. n is the bit width of the label, or the width of this object when shared among several labels. Like label data, there are several objects of IntegralData which is determined by the first line of this object. The objects defined so far are:

"IntegralArray<signed8>"

"IntegralArray<signed16>"

"IntegralArray<signed32>"

"IntegralArray<signed64>"

"IntegralArray<unsigned8>"

"IntegralArray<unsigned16>"

"IntegralArray<unsigned32>"

"IntegralArray<unsigned64>"

"BitPackedData"

"BitBlockData"

"BitBlock"

"PagedBitBlock"

"PagedIntegralData<type size>"

"StringData"

The above list can be broken down into four basic formats, IntegralArray, BitPackedData, BitBlockData, and Strings. Strings are used for textual data. The IntegralArray is used for labels that are exactly 8, 16, 32, or 64 bits wide. The BitPackedData is used for all other label widths. The BitBlockData is similar to BitPackedData in that it contains the raw data for multiple labels. The difference is that the bits associated with the label may not be consecutive or in order.

Paged objects are the same as their non-paged counter parts except that they may contain a filename instead of data. This indicates that the data is located in the named file instead of the current file being read. The additional files will be formatted with a header describing that the file is a data file, and then the actual data will follow with exactly the same format as if it were in the original file. This is being done because a fair number of computer systems have a 2GB file size limit. There are situations where the original Fast Binary Data File would be greater than 2GB, hence these paged objects.

IntegralArray<type size>

```
IntegralArray<type size> ::=
    Integral Data Type       => "IntegralArray<type size>\n"
    Integral Data ID         => "%d\n"
    (if ID has not already been processed)
    Length                   => "%d\n"
    StuckOne, StuckZero      => "%s %s\n"
    Raw Bytes                => n bytes
```

- Type and Size - Type is the sign-ness of the label data values, i.e. "signed" or "unsigned". Size is the width in bits of the label.
- Length - Length is the number of samples that is associated with this label.
- StuckOne and StuckZero - StuckOne indicates which bits, if any, of the label have a value of 1 for all of the samples. This value has the same width as the label does, which can be larger than 32 bits, so therefore is formatted as a string(%s). StuckZero indicates bits with a value of 0.
- Raw Bytes - A raw section of bytes that is n bytes long. n = length * size in bytes. The block of data can be viewed as an array of size length with each element be size bits wide. Then access the data using a zero based array index.

BitPackedData

```
BitPackedData ::=
```

```
    Integral Data Type        => "BitPackedData\n"
    Integral Data ID          => "%d\n"
    (if ID has not already been processed)
    Start Bit, Width, Inverted
                              => "%d %d %d\n"
    StuckOne, StuckZero       => "%s %s\n"
    BytesPerLine, IntegralPerLine
                              => "%d %d\n"
    DataBlock                 => &<IntegralArray&<type size>>
```

- Start Bit, Width, Inverted - Start Bit is starting bit position within the bitblock for this label. Width is the number of bits used by this label. Inverted is a flag indicated whether the data is inverted.
- StuckOne and StuckZero - StuckOne indicates which bits, if any, of the label have a value of 1 for all of the samples. This value has the same width as the label does, which can be larger than 32 bits, so therefore is formatted as a string(%s). StuckZero indicates bits with a value of 0.
- BytesPerLine and IntegralPerLine - These two fields indicate the width of the bitblock data in bytes and in terms of the base type of the IntegralArray.
- DataBlock - DataBlock is an IntegralArray that can be shared by multiple labels. Each label that occupies this DataBlock has a starting position and it's width, so that each line/sample within the datablock is wide enough to support all labels sharing this DataBlock.

BitBlockData

```
BitBlockData ::=
    Integral Data Type        => "BitBlockData\n"
    Integral Data ID          => "%d\n"
    (if ID has not already been processed)
    Extractor                 => <Label Extractor>
    StuckOne, StuckZero       => "%s %s\n"
    DataBlock                 => <BitBlock> or <PagedBitBlock>
```

- Extractor - The Label extractor is used to map the bits of a label definition to specific bits within a bitblock.
- StuckOne and StuckZero - StuckOne indicates which bits, if any, of the label have a value of 1 for all of the samples. This value has the same width as the label does, which can be larger than 32 bits, so therefore is formatted as a string(%s). StuckZero indicates bits with a value of 0.
- DataBlock - DataBlock is either a <BitBlock> or a <PagedBitBlock> that is shared by multiple labels. For each label that uses this bitblock, a label extractor is used to access the bitblock to retrieve the appropriate data associated with that particular label.

BitBlock

```
BitBlock ::=
    Integral Data Type      => "BitBlock\n"
```

```
    reserved                => "%d\n%d\n"
    Integral Data ID        => "%d\n"
    (if ID has not already been processed)
    Length, BytesPerLine    => "%u %u\n"
    Raw Bytes               => n bytes
```

- Integral Data ID - This is the Objects id. An id value of 0 indicated that there is no data associated with this object. If the ID has not been seen earlier in the file, the object information follows. If the id has been seen, then the next object follows.
- Length and BytesPerLine - Length is the number of samples (rows) of data. BytesPerLine is the number of bytes is each sample or row.
- Raw Bytes - A raw section of bytes that is n bytes long. n = length * bytesPerLine. The block of data can be viewed as an array of samples, with each sample being bytesPerLine wide. This array is then accessed using a zero based array index.

PagedBitBlock

```
PagedBitBlock ::=
    Integral Data Type      => "PagedBitBlock\n"
    reserved                => "%d\n%d\n"
    Integral Data ID        => "%d\n"
    (if ID has not already been processed)
    UseFile, Filename       => "%d `%s`\n"
    reserved                => "%d %d\n"
    Length, BytesPerLine    => "%u %u\n"
    Raw Bytes               => n bytes
```

- Integral Data ID - This is the Objects id. An id value of 0 indicated that there is no data associated with this object. If the id has not been seen earlier in the file, the object information follows. If the id has been seen, then the next object follows.
- UseFile and Filename - UseFile is a boolean that indicates whether to use a file or not. The filename is a full path name. A value of 0 means that the data continues on the next line. A value of 1 indicates that the data is located in the associated file. When reading the additional file, filename, the first section is a file comment. The second section starts with the string "HPLogic_Additional_Data_File&\n", then the rest of this Integral Data Object follows, i.e. Length and BytesPerLine and Raw Bytes.
- Length and BytesPerLine - Length is the number of samples (rows) of data. BytesPerLine is the number of bytes for each sample or row.
- Raw Bytes - A raw section of bytes that is n bytes long. n = length * bytesPerLine. The block of data can be viewed as an array of samples, with each sample being bytesPerLine wide. This array is then accessed using a zero based array index.

PagedIntegralData<type size>

```
PagedIntegralData<type size> ::=
    Integral Data Type      => "PagedIntegralData<type size>\n"
    Integral Data ID        => "%d\n"
    (if ID has not already been processed)
    UseFile, Filename       => "%d '%s'\n"
    reserved                => "%d %d\n"
    Length, BytesPerLine    => "%u %u\n"
    Raw Bytes               => n bytes
```

- Type and Size - Type is the sign-ness of the label data values, i.e. "signed" or "unsigned". Size is the width in bits of the label. Valid bit widths are 8, 16, 32, 64.
- Integral Data ID - This is the Objects id. An id value of 0 indicated that there is no data associated with this object. If the ID has not been seen earlier in the file, the object information follows. If the id has been seen, then the next object follows.
- UseFile and Filename - UseFile is a boolean that indicates whether to use a file or not. The filename is a full path name. A value of 0 means that the data continues on the next line. A value of 1 indicates that the data is located in the associated file. When reading the additional file, filename, the first section is a file comment. The second section starts with the string "HPLogic_Additional_Data_File&\n", then the rest of this Inegral Data Object follows, i.e. Length and BytesPerLine and Raw Bytes.
- Length and BytesPerLine - Length is the number of samples (rows) of data. BytesPerLine is the number of bytes for each sample or row. BytesPerLine will be either 1, 2, 4, 8.
- Raw Bytes - A raw section of bytes that is n bytes long. n = length * bytesPerLine. The block of data can be viewed as an array of samples, with each sample being bytesPerLine wide. This array is then accessed using a zero based array index.

Strings

```
Strings ::=
    Integral Data Type      => "StringData\n"
    Integral Data ID        => "%d\n"
    (if ID has not already been processed)
    Length                  => "%d\n"
    String Block            => "%d %s [%d %s]*"
```

- Length - Length is the number of strings contained in the data.
- String Block - The block of data can be viewed as two sections. First, a value is read which represents the number of characters in the current string, say x. The following x bytes are the actual string itself. Then immediately following is a character count of the next string, and so on.

Label Extractor

```
<Label Extractor> ::=
    Bytes, Width, Inverted       => "%u %u %d\n"
    Mask                         => "%x [%x]* \n"
    reserved                     => "%d\n%d\n"
    HaveReorder                  => "%d\n"
    Reorder                      => <Reorder>
```

- Bytes, Width, and Inverted - Bytes is the size of the mask. Width is the number of bits that are set in the mask, also the number of bits of the associated label. Inverted indicates if the label has negative polarity, i.e. does the data need to be complemented after extraction.

- Mask - The mask is an array of bytes that specifies which bits of the bitblock are needed for this label. The bytes of this mask corresponds to the bytes of each sample. A bit set to a 1 in the mask indicates that the corresponding bit in the sample should be used. The data is stored in big Endian format, hence byte 0 is the most significant and continues till byte n. For example, bytes=2, width=7:

```
           Byte 0    Byte 1
          111111
      bit 54321098  76543210
Mask:     10010011  00101010
Sample:   11100000  01111000
          --------  --------
Value:    1  0  00    1 1 0       =>  1000110  => 0x46
```

  If inverted is true or 1, then the value would become 0111001(0x39)

- HaveReorder - HaveReorder is a flag that indicates whether the bits need to be reordered. Reordering occurs after the extraction takes place. If the label bits are configured with bit reordering this field will have a non-zero value, and the reorder object will follow. If haveReorder is zero, nothing will follow.

- Reorder

```
Endian Flags     => "Endian16: %d Endian32: %d Endian64: %d "
and Width        => "Endian128: %d Width: %d\n"
Bit Order Map    => [ "%d %d\n" or " %s %s\n" ](width times)
```

  The reorder object describes the desired label bit ordering. The appropriate Endian flags are set true when little endian is specified. Width specifies the number of bits that are defined in this label, i.e. it should be the same as the width specified in the parent Extract object. When a custom bit ordering is specified, all the Endian flags will be false/0. If all Endian flags are zero then there will be a bit order map following the flags. The bit order map consists of two arrays that are width long, i.e. there is an array element for each bit that is set in the mask. The first array is the map from display bits to channel/bitblock bits, the second array is the map from

channel bits to display bits.The arrays are of type int if the width is less than or equal to 32, otherwise the array is setup as strings to be processed with any method that you have for handling greater than 32 bit integers. The arrays are maps that map the bit(index of array) you are working with to the bit position(value of array) that it should be moved to. For example, a 4 bit label:

```
Channel bit     Display bit
     0              2
     1              0
     2              1
     3              3

      (index)    Array1   Array2
         0        0010     0100
         1        0100     0001
         2        0001     0010
         3        1000     1000
```

Label Data

The label data can be one of several objects. The object is determined by the first string of the label data object. Following the 'type' string is the raw binary data of the label. Then comes a bitset that describes the attributes of this label. The currently defined label data objects are:

```
"NoData"
"States"
"StateCount"
"Glitch"
"Analog"
"TextLines"
```

- NoData - This is the label data object that is empty!

```
<NoData> ::=
    Label Data Type   => "NoData\n"
    Label Attribute bitset
                      => "%d [%d]* \n"
```

- States - This is the label data object that is most common. This is used to represent sampled data of n-bit width. The <IntegralData> will contain the actual sample data and is accessed with a zero based index.

```
<States> ::=
    Label Data Type   => "States\n"
    <IntegralData>
    new line          => "\n"
    Label Attributes bitset
                      => "%d [%d]* \n"
```

- StateCount - This label data object is used to store state counts when acquiring data with state tagging turned on.

```
<StateCount> ::=
    Label Data Type    => "StateCount\n"
    <IntegralData>
    new line           => "\n"
    Label Attributes bitset
                       => "%d [%d]* \n"
```

- Glitch - This object is used for labels that have data that was acquired with glitch mode turned on. The sample <IntegralData> contains the sample data just like the State object. The glitch <IntegralData> contains the glitch data. There is a one-to-one correspondence between the sample and glitch data. When the glitch data is true, a 1 in any bit position, indicates that a glitch occurred on the corresponding sample.

```
<Glitch> ::=
    Label Data Type    => "Glitch\n"
    Sample <IntegralData>
    new line           => "\n"
    Glitch <IntegralData>
    new line           => "\n"
    Label Attributes bitset
                       => "%d [%d]* \n"
```

- Analog - The object is for analog type data. The integral array contains the digital quantization-levels of the sampled signal. The vertical header contains the information to convert the digital data to voltage levels.

```
<Analog> ::=
    Label Data Type    => "Analog\n"
    <IntegralData>
    new line           => "\n"
    <VerticalHeader>
    Analog units       => "%s\n"
    Label Attributes bitset
                       => "%d [%d]* \n"
```

- TextLines - The object is for text type data. The integral array contains text strings of data.

```
<Text> ::=
    Label Data Type    => "TextLines\n"
    Lines <Strings>
    new line           => "\n"
    Label Attributes bitset
                       => "%d [%d]* \n"
```

Label Data Attributes Bitset

The label data attributes is a bitset that represents the different attributes this data has. At this time there are 22 label data attributes:

```
Bit#      Description
  1       Label has width
  2       Label has length(samples)
```

```
 3       Label has Glitch Data
 4       Label has a Vertical Header
 5       Label Data has trigger
 6       Range
 7       Label Data has Statistical data
 8       Label Data has Integral data
 9       Label Data is in floating point
10       Label Data is signed
11       Label Data is unsigned
12       Label Data is Periodic
13       Label has symbols
14       ValueItor
15       GlitchItor
16       SampledData
17       VersusTime
18       VersusFreq
19       Analog
20       Digital
21       StateCounts
22       TextLines
```

Label Entry

The Label Entry structure is the information associated with one data label. This structure contains the label name, label data, label attributes, and symbol information.

```
<Label Entry> ::=
    Label ID      => "%d\n"
    (if Label ID has not already been processed)
        Label Name    => "'%s'\n"
        <Label Data>
        new line          => "\n"
        Label Attribute Bits  => "%d [%d]*\n"
```

- Label ID - There is a unique identifier for each label. When the same label is used in multiple DataSets, the label data is written only once. If this label's data has already been written to the file then the label name, label data, and label attributes are not included in this record.
- Label Attribute Bitset - These attributes are used by various tools within the instrument to identify certain labels. Most of these are for inverse assembler. The only one that might be of interest is StateCount which is set when doing state with state-tags type of acquisitions.

```
Bit#     Description
  0        TypeAddress
  1        TypeAddressB
  2        TypeAddressC
  3        TypeAddressD
  4        TypeData
  5        TypeDataB
  6        TypeDataC
  7        TypeDataD
  8        TypeStatus
  9        TypeStatusB
```

```
10        TypeStatusC
11        TypeStatusD
12        TypeClock
13        TypeClockPos
14        TypeClockNeg
15        TypeClockBoth
16        TypeSampleEdge
17        TypeSampleEdgePos
18        TypeSampleEdgeNeg
19        TypeSampleEdgeBoth
20        TypeControl
21        StateCount
22        Sequencer
```

Vertical Header

The vertical header is used to describe information about a signal that goes beyond just digital data. At this time, the only specific application is to describe the signal from an scope. As with other polymorphic objects, this object supports several types of objects which is specified by the first string/line. The possible types at this time are:

"DefaultOrdinateHeader"

"ScopeHeader"

- DefaultOrdinateHeader

```
<DefaultOrdinateHeader> ::=
    Header Type => "DefaultOrdinateHeader\n"
```

- ScopeHeader

```
<ScopeHeader> ::=
    Header Type            => "ScopeHeader\n"
    YIncrement, YOrigin, YReference, NumBits
                           => "%f %f %d %d\n"
```

- YOrigin - is the voltage value at center screen.
- YIncrement - the voltage difference between consecutive data values.
- YReference - the value that specifies the data value at center screen, where YOrigin occurs.
- NumBits - the number of bits for the width of the scope data.

Abscissa Data Type (x-axis information)

The Abscissa Data object contains the time and/or state information that describes how the data "flows" in this data set. The state portion of this object describes the number of samples and where the trigger is located within the samples. If there is time associated with this data set, then the time portion describes any timing information. This is a polymorphic object and as such, the first line determines which object is actually used. The current possible objects are:

- AbscissaData - The Abscissa Data object is a base object that describes the number of samples and the trigger position for all of the labels (LabelEntry) contained in this data set. The abscissa attributes give the characteristics of the abscissa data.

```
<AbscissaDataType> ::=
    Abscissa Data Type      => "AbscissaData\n"
    Number of Samples, Trigger Position
                            => "%d %d\n"
    Abscissa Attribute bitset
                            => "%d [%d]* \n"
```

- Periodic - The Periodic object is used when the timing information is based on periodic sampling.

```
<AbscissaDataType> ::=
    Abscissa Data Type      => "Periodic\n"
    Number of Samples, Trigger Position
                            => "%d %d\n"
    Origin, Increment       => "%s %s\n"
    Abscissa Attribute bitset
                            => "%d %d\n"
```

  - Origin is the time of the first sample.
  - Increment is the time between samples.

  These two times are defined in terms of strings because these values are normally kept as 64-bit signed integers. A value of 1 represents 1 pico-second.

- TimeTags / PagedTimeTags - The TimeTags object is used for any timing information that is not periodic.

```
<AbscissaDataType> ::=
    Abscissa Data Type        => "TimeTags\n"
    Number of Samples, Trigger Position
                              => "%d %d\n"
    Time Values               => IntegralArray&<signed64>
    Abscissa Attribute bitset => "%d [%d]* \n"
```

The PagedTimeTags object is also used for any timing information, but it has one additional data element, Time Values Flag. If this flag is zero, then the Time Values element is skipped!

```
<AbscissaDataType> ::=
    Abscissa Data Type        => "PagedTimeTags\n"
    Number of Samples, Trigger Position
                              => "%d %d\n"
    Time Values Flag          => "%d\n"
    (if TimeValuesFlag is non-zero, then Time Values)
    Time Values               => IntegralArray<signed64>
    Abscissa Attribute bitset => "%d [%d]* \n"
```

- Time Values - The time is kept in 64 bit signed integers. These integers represent the number of pico-seconds. The time value is an array of times and is accessed with a zero based index. There is a one-to-one correspondence between the sample array and this time value array.

Abscissa Attributes bitset

```
Bit#     Description
  1       Abscissa has Length
  2       Abscissa has HorizontalHeader
  3       Abscissa has TimeCorrelation
  4       Abscissa has TriggerRow
  5       Abscissa is TimeTag
  6       Abscissa is StateTag
  7       Abscissa is SamplePeriod
  8       Range
  9       Abscissa is Periodic
 10       Abscissa has TimeItor
 11       Abscissa has StateNumberItor
 12       Abscissa consists of SampledData
```

Time Correlation Info

The Time Correlation info object describes how this data set is time correlated to other data sets.

```
<Time Correlation Info>::=
    Correlation Type          => "TimeCorrelationInfo\n"
    TimeCorrelationType, Source, CorrelationTime
                              => "%d %d %s\n"
```

- TimeCorrelationType - The type of time correlation for this data set. Correlation can be of 3 different types. No time correlation means there is no time correlation information for this data set. Conditional time correlation means that the data set can be time correlated under certain circumstances with data from another machine. Unconditional time correlation means that the data is unconditionally correlated with another data set as it is from the same machine.

```
0  No Time Correlation
1  Conditional Time Correlation
2  Unconditional Time Correlation
```

- Source - The source is the machine id from which the data set was obtained.
- CorrelationTime - The correlation time is a zero-relative time based on the data set that is considered the *trigger* data set. If the time is non-zero, this is the amount of time offset to use to correlate this data from the *trigger* data's time.

State Correlation Info

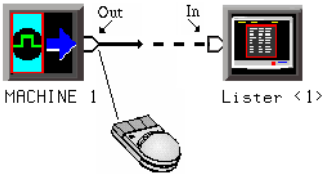
The State Correlation info object describes how this data set is state correlated to other data sets.

```
<State Correlation Info>::=
    Correlation Type => "StateCorrelationInfo\n"
    Offset           => "%s\n"
```

- Offset - The correlation offset is a zero-relative number of states based on the data set that is considered the `trigger' data set. This value represents how many states to offset from the `trigger' data set to correlate the data. (Note: this value may be invalid. It is recommended that no dependence is placed on this field.)

Connecting Tools Together

If you *drag and drop* tools into open space in the workspace, you must create a data path between the tools by connecting their output and input ports.

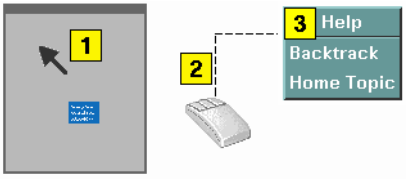


To Connect Output and Input Ports

1. *Point* to the tool output port.
2. Press and hold, then move the cursor over to a tool input port, then release.
   You should now have a line, representing a data path, drawn between tool data ports.

Help - How to Navigate Quickly

1. Place mouse cursor anywhere in a help window.
2. Press the right mouse button.
3. Select desired destination.



You can also access all navigation and search commands from the help window menu bar.

Help - System Overview

The help system is divided into *System* Help, and *Tool* Help. All help is designed to be task oriented and specific to the window where tasks are performed.

Links, in most cases, are confined to topics in the specific window where help was requested. However, some links do go up to system-level topics from specific tool windows. When this occurrs, a second help window will appear. Since the definition of future tools is hard to link to, there are no links going from the system level down to specific tools.

System Help

The system help is accessed through the *Help* field in the menu bar of the main system window. It offers help on system-level topics and operations.

Tool Help

As you add new software and hardware tools to the system, the tool specific help is added. Tool specific help is accessed through the *Help* field in the menu bar of the specific tool windows.

Using Help

In addition to system and tool help, there is help on help called *Using Help*. Using Help shows you how to navigate and search the help systems and to print help topics. Using Help is accessed through the *Help* field in all windows.

absolute Denotes the time period or count of states between a captured state and the trigger state. An absolute count of -10 indicates the state was captured ten states before the trigger state was captured.

acquisition Denotes one complete cycle of data gathering by a measurement module. For example, if you are using an analyzer with 128K memory depth, one complete acquisition will capture and store 128K states in acquisition memory.

analysis probe A probe connected to a microprocessor or standard bus in the device under test. An analysis probe provides an interface between the signals of the microprocessor or standard bus and the inputs of the logic analyzer. Also called a *preprocessor*.

analyzer 1 In a logic analyzer with two *machines*, refers to the machine that is on by default. The default name is *Analyzer<N>*, where N is the slot letter.

analyzer 2 In a logic analyzer with two *machines*, refers to the machine that is off by default. The default name is *Analyzer<N2>*, where N is the slot letter.

arming An instrument tool must be armed before it can search for its trigger condition. Typically, instruments are armed immediately when *Run* or *Group Run* is selected. You can set up one instrument to arm another using the *Intermodule Window*. In these setups, the second instrument cannot search for its trigger condition until it receives the arming signal from the first instrument. In some analyzer instruments, you can set up one analyzer *machine* to arm the other analyzer machine in the *Trigger Window*.

asterisk (*) See *edge terms*, *glitch*, and *labels*.

bits Bits represent the physical logic analyzer channels. A bit is a *channel* that has or can be assigned to a *label*. A bit is also a position in a label.

card This refers to a single instrument intended for use in the Agilent Technologies 16600A-series or 16700A/B-series mainframes. One card fills one slot in the mainframe. A module may comprise a single card or multiple cards cabled together.

channel The entire signal path from the probe tip, through the cable and module, up to the label grouping.

click When using a mouse as the

pointing device, to click an item, position the cursor over the item. Then quickly press and release the *left mouse button*.

clock channel A logic analyzer *channel* that can be used to carry the clock signal. When it is not needed for clock signals, it can be used as a *data channel*, except in the Agilent Technologies 16517A.

context record A context record is a small segment of analyzer memory that stores an event of interest along with the states that immediately preceded it and the states that immediately followed it.

context store If your analyzer can perform context store measurements, you will see a button labeled *Context Store* under the Trigger tab. Typical context store measurements are used to capture writes to a variable or calls to a subroutine, along with the activity preceding and following the events. A context store measurement divides analyzer memory into a series of context records. If you have a 64K analyzer memory and select a 16-state context, the analyzer memory is divided into 4K 16-state context records. If you have a 64K analyzer memory and select a 64-state context, the analyzer memory will be divided into 1K 64-state records.

count The count function records periods of time or numbers of state transactions between states stored in memory. You can set up the analyzer count function to count occurrences of a selected event during the trace, such as counting how many times a variable is read between each of the writes to the variable. The analyzer can also be set up to count elapsed time, such as counting the time spent executing within a particular function during a run of your target program.

cross triggering Using intermodule capabilities to have measurement modules trigger each other. For example, you can have an external instrument arm a logic analyzer, which subsequently triggers an oscilloscope when it finds the trigger state.

data channel A *channel* that carries data. Data channels cannot be used to clock logic analyzers.

data field A data field in the pattern generator is the data value associated with a single label within a particular data vector.

data set A data set is made up of all labels and data stored in memory of any single analyzer machine or

instrument tool. Multiple data sets can be displayed together when sourced into a single display tool. The Filter tool is used to pass on partial data sets to analysis or display tools.

debug mode See *monitor*.

delay The delay function sets the horizontal position of the waveform on the screen for the oscilloscope and timing analyzer. Delay time is measured from the trigger point in seconds or states.

demo mode An emulation control session which is not connected to a real target system. All windows can be viewed, but the data displayed is simulated. To start demo mode, select *Start User Session* from the Emulation Control Interface and enter the demo name in the *Processor Probe LAN Name* field. Select the *Help* button in the *Start User Session* window for details.

deskewing To cancel or nullify the effects of differences between two different internal delay paths for a signal. Deskewing is normally done by routing a single test signal to the inputs of two different modules, then adjusting the Intermodule Skew so that both modules recognize the signal at the same time.

device under test The system under test, which contains the circuitry you are probing. Also known as a *target system*.

don't care For *terms*, a "don't care" means that the state of the signal (high or low) is not relevant to the measurement. The analyzer ignores the state of this signal when determining whether a match occurs on an input label. "Don't care" signals are still sampled and their values can be displayed with the rest of the data. Don't cares are represented by the *X* character in numeric values and the dot (.) in timing edge specifications.

dot (.) See *edge terms*, *glitch*, *labels*, and *don't care*.

double-click When using a mouse as the pointing device, to double-click an item, position the cursor over the item, and then quickly press and release the *left mouse button* twice.

drag and drop Using a Mouse: Position the cursor over the item, and then press and hold the *left mouse button*. While holding the left mouse button down, move the mouse to drag the item to a new location. When the item is positioned where you want it, release the mouse button.

Using the Touchscreen:
Position your finger over the item, then press and hold finger to the screen. While holding the finger down, slide the finger along the screen dragging the item to a new location. When the item is positioned where you want it, release your finger.

edge mode In an oscilloscope, this is the trigger mode that causes a trigger based on a single channel edge, either rising or falling.

edge terms Logic analyzer trigger resources that allow detection of transitions on a signal. An edge term can be set to detect a rising edge, falling edge, or either edge. Some logic analyzers can also detect no edge or a *glitch* on an input signal. Edges are specified by selecting arrows. The dot (.) ignores the bit. The asterisk (*) specifies a glitch on the bit.

emulation module A module within the logic analysis system mainframe that provides an emulation connection to the debug port of a microprocessor. An E5901A emulation module is used with a target interface module (TIM) or an analysis probe. An E5901B emulation module is used with an E5900A emulation probe.

emulation probe The stand-alone equivalent of an *emulation module*. Most of the tasks which can be performed using an emulation module can also be performed using an emulation probe connected to your logic analysis system via a LAN.

emulator An *emulation module* or an *emulation probe*.

Ethernet address See *link-level address*.

events Events are the things you are looking for in your target system. In the logic analyzer interface, they take a single line. Examples of events are *Label1 = XX* and *Timer 1 > 400 ns*.

filter expression The filter expression is the logical *OR* combination of all of the filter terms. States in your data that match the filter expression can be filtered out or passed through the Pattern Filter.

filter term A variable that you define in order to specify which states to filter out or pass through. Filter terms are logically OR'ed together to create the filter expression.

Format The selections under the logic analyzer *Format* tab tell the

logic analyzer what data you want to collect, such as which channels represent buses (labels) and what logic threshold your signals use.

frame The Agilent Technologies 16600A-series or 16700A/B-series logic analysis system mainframe. See also *logic analysis system*.

gateway address An IP address entered in integer dot notation. The default gateway address is 0.0.0.0, which allows all connections on the local network or subnet. If connections are to be made across networks or subnets, this address must be set to the address of the gateway machine.

glitch A glitch occurs when two or more transitions cross the logic threshold between consecutive timing analyzer samples. You can specify glitch detection by choosing the asterisk (*) for *edge terms* under the timing analyzer Trigger tab.

grouped event A grouped event is a list of *events* that you have grouped, and optionally named. It can be reused in other trigger sequence levels. Only available in Agilent Technologies 16715A, 16716A, and 16717A logic analyzers.

held value A value that is held until the next sample. A held value can exist in multiple data sets.

immediate mode In an oscilloscope, the trigger mode that does not require a specific trigger condition such as an edge or a pattern. Use immediate mode when the oscilloscope is armed by another instrument.

interconnect cable Short name for *module/probe interconnect cable*.

intermodule bus The intermodule bus (IMB) is a bus in the frame that allows the measurement modules to communicate with each other. Using the IMB, you can set up one instrument to *arm* another. Data acquired by instruments using the IMB is time-correlated.

intermodule Intermodule is a term used when multiple instrument tools are connected together for the purpose of one instrument arming another. In such a configuration, an arming tree is developed and the group run function is designated to start all instrument tools. Multiple instrument configurations are done in the Intermodule window.

internet address Also called Internet Protocol address or IP address. A 32-bit network address. It

is usually represented as decimal numbers separated by periods; for example, 192.35.12.6. Ask your LAN administrator if you need an internet address.

labels Labels are used to group and identify logic analyzer channels. A label consists of a name and an associated bit or group of bits. Labels are created in the Format tab.

line numbers A line number (Line #s) is a special use of *symbols*. Line numbers represent lines in your source file, typically lines that have no unique symbols defined to represent them.

link-level address Also referred to as the Ethernet address, this is the unique address of the LAN interface. This value is set at the factory and cannot be changed. The link-level address of a particular piece of equipment is often printed on a label above the LAN connector. An example of a link-level address in hexadecimal: 0800090012AB.

local session A local session is when you run the logic analysis system using the local display connected to the product hardware.

logic analysis system The Agilent Technologies 16600A-series or 16700A/B-series mainframes, and all tools designed to work with it. Usually used to mean the specific system and tools you are working with right now.

machine Some logic analyzers allow you to set up two measurements at the same time. Each measurement is handled by a different machine. This is represented in the Workspace window by two icons, differentiated by a *1* and a *2* in the upper right-hand corner of the icon. Logic analyzer resources such as pods and trigger terms cannot be shared by the machines.

markers Markers are the green and yellow lines in the display that are labeled *x*, *o*, *G1*, and *G2*. Use them to measure time intervals or sample intervals. Markers are assigned to patterns in order to find patterns or track sequences of states in the data. The x and o markers are local to the immediate display, while G1 and G2 are global between time correlated displays.

master card In a module, the master card controls the data acquisition or output. The logic analysis system references the module by the slot in which the master card is plugged. For example, a 5-card Agilent Technologies 16555D

would be referred to as *Slot C: machine* because the master card is in slot C of the mainframe. The other cards of the module are called *expansion cards*.

menu bar The menu bar is located at the top of all windows. Use it to select *File* operations, tool or system *Options*, and tool or system level *Help*.

message bar The message bar displays mouse button functions for the window area or field directly beneath the mouse cursor. Use the mouse and message bar together to prompt yourself to functions and shortcuts.

module/probe interconnect cable

The module/probe interconnect cable connects an E5901B emulation module to an E5900B emulation probe. It provides power and a serial connection. A LAN connection is also required to use the emulation probe.

module An instrument that uses a single timebase in its operation. Modules can have from one to five cards functioning as a single instrument. When a module has more than one card, system window will show the instrument icon in the slot of the *master card*.

monitor When using the Emulation Control Interface, running the monitor means the processor is in debug mode (that is, executing the debug exception) instead of executing the user program.

panning The action of moving the waveform along the timebase by varying the delay value in the Delay field. This action allows you to control the portion of acquisition memory that will be displayed on the screen.

pattern mode In an oscilloscope, the trigger mode that allows you to set the oscilloscope to trigger on a specified combination of input signal levels.

pattern terms Logic analyzer resources that represent single states to be found on labeled sets of bits; for example, an address on the address bus or a status on the status lines.

period (.) See *edge terms*, *glitch*, *labels*, and *don't care*.

pod pair A group of two pods containing 16 channels each, used to physically connect data and clock signals from the unit under test to the analyzer. Pods are assigned by pairs in the analyzer interface. The number of pod pairs avalaible is determined

by the channel width of the instrument.

pod See *pod pair*

point To point to an item, move the mouse cursor over the item, or position your finger over the item.

preprocessor See *analysis probe*.

primary branch The primary branch is indicated in the *Trigger sequence step* dialog box as either the *Then find* or *Trigger on* selection. The destination of the primary branch is always the next state in the sequence, except for the Agilent Technologies 16517A. The primary branch has an optional occurrence count field that can be used to count a number of occurrences of the branch condition. See also *secondary branch*.

probe A device to connect the various instruments of the logic analysis system to the target system. There are many types of probes and the one you should use depends on the instrument and your data requirements. As a verb, "to probe" means to attach a probe to the target system.

processor probe See *emulation probe*.

range terms Logic analyzer resources that represent ranges of values to be found on labeled sets of bits. For example, range terms could identify a range of addresses to be found on the address bus or a range of data values to be found on the data bus. In the trigger sequence, range terms are considered to be true when any value within the range occurs.

relative Denotes time period or count of states between the current state and the previous state.

remote display A remote display is a display other than the one connected to the product hardware. Remote displays must be identified to the network through an address location.

remote session A remote session is when you run the logic analyzer using a display that is located away from the product hardware.

right-click When using a mouse for a pointing device, to right-click an item, position the cursor over the item, and then quickly press and release the *right mouse button*.

sample A data sample is a portion of a *data set*, sometimes just one point. When an instrument samples the target system, it is taking a single

measurement as part of its data acquisition cycle.

Sampling Use the selections under the logic analyzer Sampling tab to tell the logic analyzer how you want to make measurements, such as State vs. Timing.

secondary branch The secondary branch is indicated in the *Trigger sequence step* dialog box as the *Else on* selection. The destination of the secondary branch can be specified as any other active sequence state. See also *primary branch*.

session A session begins when you start a *local session* or *remote session* from the session manager, and ends when you select *Exit* from the main window. Exiting a session returns all tools to their initial configurations.

skew Skew is the difference in channel delays between measurement channels. Typically, skew between modules is caused by differences in designs of measurement channels, and differences in characteristics of the electronic components within those channels. You should adjust measurement modules to eliminate as much skew as possible so that it does not affect the accuracy of your measurements.

state measurement In a state measurement, the logic analyzer is clocked by a signal from the system under test. Each time the clock signal becomes valid, the analyzer samples data from the system under test. Since the analyzer is clocked by the system, state measurements are *synchronous* with the test system.

store qualification Store qualification is only available in a *state measurement*, not *timing measurements*. Store qualification allows you to specify the type of information (all samples, no samples, or selected states) to be stored in memory. Use store qualification to prevent memory from being filled with unwanted activity such as no-ops or wait-loops. To set up store qualification, use the *While storing* field in a logic analyzer trigger sequence dialog.

subnet mask A subnet mask blocks out part of an IP address so that the networking software can determine whether the destination host is on a local or remote network. It is usually represented as decimal numbers separated by periods; for example, 255.255.255.0. Ask your LAN administrator if you need a the subnet mask for your network.

symbols Symbols represent patterns and ranges of values found on labeled sets of bits. Two kinds of symbols are available:

- Object file symbols - Symbols from your source code, and symbols generated by your compiler. Object file symbols may represent global variables, functions, labels, and source line numbers.
- User-defined symbols - Symbols you create.

Symbols can be used as *pattern* and *range* terms for:

- Searches in the listing display.
- Triggering in logic analyzers and in the source correlation trigger setup.
- Qualifying data in the filter tool and system performance analysis tool set.

system administrator The system administrator is a person who manages your system, taking care of such tasks as adding peripheral devices, adding new users, and doing system backup. In general, the system administrator is the person you go to with questions about implementing your software.

target system The system under test, which contains the microprocessor you are probing.

terms Terms are variables that can be used in trigger sequences. A term can be a single value on a label or set of labels, any value within a range of values on a label or set of labels, or a glitch or edge transition on bits within a label or set of labels.

TIM A TIM (Target Interface Module) makes connections between the cable from the emulation module or emulation probe and the cable to the debug port on the system under test.

time-correlated Time correlated measurements are measurements involving more than one instrument in which all instruments have a common time or trigger reference.

timer terms Logic analyzer resources that are used to measure the time the trigger sequence remains within one sequence step, or a set of sequence steps. Timers can be used to detect when a condition lasts too long or not long enough. They can be used to measure pulse duration, or duration of a wait loop. A single timer term can be used to delay trigger until a period of time after detection of a significant event.

timing measurement In a timing measurement, the logic analyzer samples data at regular intervals according to a clock signal internal to the timing analyzer. Since the analyzer is clocked by a signal that is not related to the system under test, timing measurements capture traces of electrical activity over time. These measurements are *asynchronous* with the test system.

tool icon Tool icons that appear in the workspace are representations of the hardware and software tools selected from the toolbox. If they are placed directly over a current measurement, the tools automatically connect to that measurement. If they are placed on an open area of the main window, you must connect them to a measurement using the mouse.

toolbox The Toolbox is located on the left side of the main window. It is used to display the available hardware and software tools. As you add new tools to your system, their icons will appear in the Toolbox.

tools A tool is a stand-alone piece of functionality. A tool can be an instrument that acquires data, a display for viewing data, or a post-processing analysis helper. Tools are represented as icons in the main window of the interface.

trace See *acquisition*.

trigger sequence A trigger sequence is a sequence of events that you specify. The logic analyzer compares this sequence with the samples it is collecting to determine when to *trigger*.

trigger specification A trigger specification is a set of conditions that must be true before the instrument triggers.

trigger Trigger is an event that occurs immediately after the instrument recognizes a match between the incoming data and the trigger specification. Once trigger occurs, the instrument completes its *acquisition*, including any store qualification that may be specified.

workspace The workspace is the large area under the message bar and to the right of the toolbox. The workspace is where you place the different instrument, display, and analysis tools. Once in the workspace, the tool icons graphically represent a complete picture of the measurements.

zooming In the oscilloscope or timing analyzer, to expand and contract the waveform along the time base by varying the value in the s/Div

field. This action allows you to select specific portions of a particular waveform in acquisition memory that will be displayed on the screen. You can view any portion of the waveform record in acquisition memory.

Index

Index