



Connecting Devices™

Quick Start Guide

ABCC SPI Protocol Analyzer

A Plugin for Saleae Logic Software



Version history

Version	History	Author	Date
1.00	Initial release	Jon Carrier	2016-09-24
1.01	Updates to resynchronize with current plugin behavior	Jon Carrier	2018-01-10

About this document

This quick start guide documents the installation, features, issues, and general use of the ABCC SPI Analyzer plugin for Saleae Logic Software.

Using the compiled plugin obtained from this project, a developer can analyze bi-directional ABCC SPI communication and present the data in a more human readable way using the Saleae 'Logic' software and hardware.

This plugin parses SPI communication according to the ABCC SPI messaging protocol. Each field within the SPI telegram is added as a multi-layered bubble-text within the Logic software. This means basic markup is displayed when zoomed-out and very detailed information is displayed when zoomed-in on the capture. By coupling this protocol analyzer with other physical signals from the target device (such as CAN, Serial, GPIO, etc.) this tool can help solve difficult-to-diagnose issues that arise during initial development efforts while keeping manual interpretation of SPI data to a minimum.

Table of Contents

1	Software Overview	4
1.1	System Requirements	4
1.2	Installation	4
1.3	Running the Analyzer	4
2	Feature List	6
3	ABCC SPI Protocol Plugin Overview	7
3.1	Bubble Text	7
3.1.1	Multi-Layer Bubble Text	7
3.2	Markers	8
3.3	Measurement	9
3.4	Analyzer Settings Window	10
3.4.1	Network Type	11
3.4.2	Index - Errors	11
3.4.3	Index – Network Timestamp	11
3.4.4	Index - Anybus Status	11
3.4.5	Index - Application Status	12
3.4.6	Index - Message	12
3.4.7	Message Data Priority	13
3.4.8	Process Data Priority	13
3.4.9	Advanced Settings	14
3.5	Export Options	15
4	Troubleshooting	16
4.1	USB Ports	16
4.2	System Memory	16

1 Software Overview

1.1 System Requirements

Recommended:

- Saleae Logic (software) version 1.2.17
 - o Other versions will work so long as it is compatible with Logic SDK version 1.1.32.
- Saleae Logic analyzer (hardware)
- Operating system
 - o Microsoft Windows 7 or newer
 - o Apple OS X 10.7 Lion+ (or macOS)
 - o Ubuntu 12.04.2+
- Precompiled ABCC SPI Analyzer plugin

1.2 Installation

To install this plugin, it is assumed that the user has previously installed the Saleae Logic software using the default installation options.

For 64-bit versions of Windows:

Please locate and copy the “x64” version of the plugin found in:

“.\plugins\Win64\AbccSpiAnalyzer64.dll”

And copy to:

“C:\Program Files\Saleae LLC\Analyzers\”

For 32-bit versions of Windows:

Please locate and copy the “x86” version of the plugin found in:

“.\plugins\Win32\AbccSpiAnalyzer.dll”

And copy to:

“C:\Program Files\Saleae LLC\Analyzers\”

For 64-bit versions of Ubuntu:

Please locate and copy the “x64” version of the plugin found in:

“.\plugins\Linux64\AbccSpiAnalyzer64.so”

And copy to the “.\Analyzers” folder of the Logic software installation path

For 32-bit versions of Ubuntu:

Please locate and copy the “x86” version of the plugin found in:

“.\plugins\Linux32\AbccSpiAnalyzer.so”

And copy to the “.\Analyzers” folder of the Logic software installation path

For Mac OS X:

Please locate and copy the plugin found in:

“.\plugins\OSX\AbccSpiAnalyzer64.dylib”

And copy to the “.\Analyzers” folder of the Logic software installation path

1.3 Running the Analyzer

If properly installed, the analyzer will appear in the “Analyzers” sub-window’s “add” option (+) as shown in the figure below.

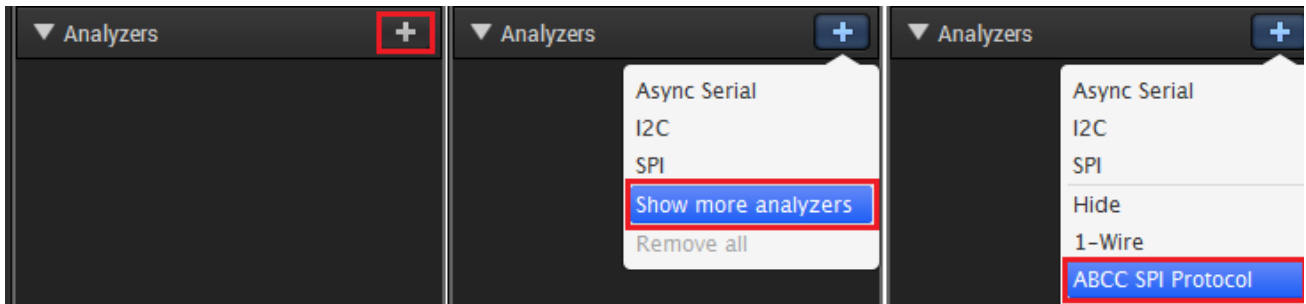


Figure 1 : Adding the analyser

After the analyzer has been selected, the settings window will be displayed as shown in the figure below. The default indexing options provide the most basic and, perhaps, most commonly used options, but can be changed according to your needs.

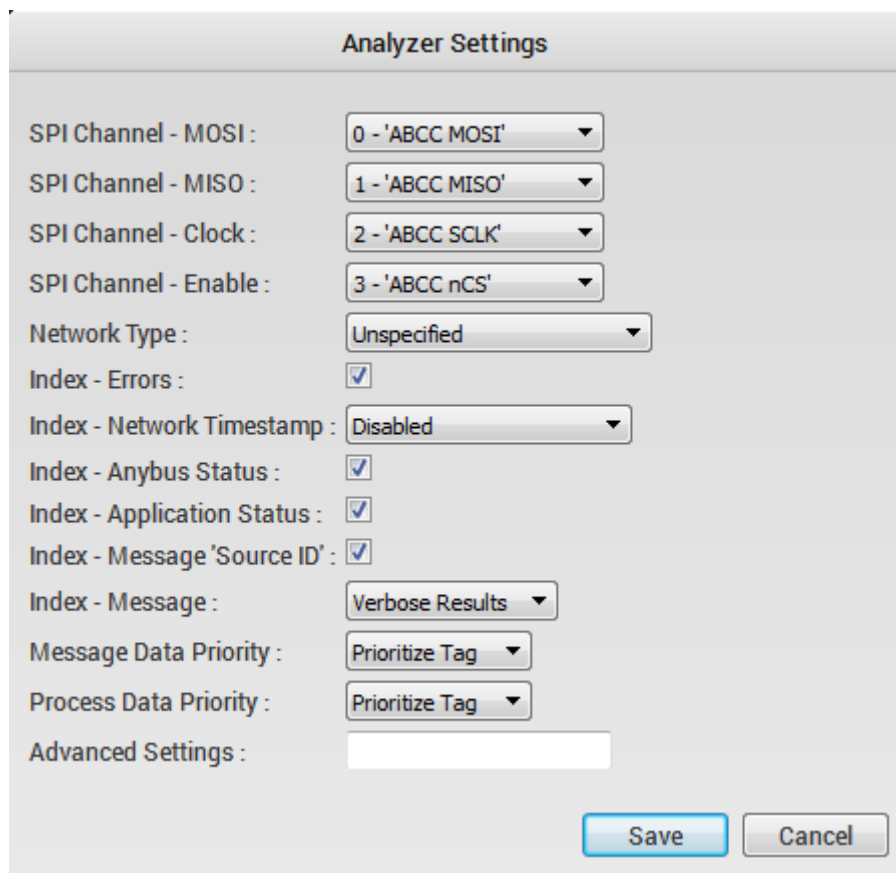


Figure 2 : Analyzer settings window

Select the corresponding channels that are associated with the physical MOSI, MISO, Clock (SCLK), and Enable (CS/SS) lines to the ABCC. The enable channel is optional and will function in 3-wire mode if set to 'None'.

IMPORTANT NOTE:

When using the plugin in "3-wire" mode there are minimum "idle gap" and maximum "clock idle high" timing requirements (>10us and <5us respectively). Also, the ABCC's 3-wire SPI mode only supports "clock idle high" configuration and is enforced in the analyzer logic in such a configuration. See discussion on "Advanced Settings" for details on how to analyze 4-wire without monitoring the enable signal.

2 Feature List

The ABCC SPI Protocol Analyzer Plugin consists of the following features:

- Multi-layered framing of ABCC SPI fields
- 3-wire or 4-wire SPI support
- Indexed and searchable results (with analyzer filter options) for:
 - Network timestamp (with packet counter)
 - Anybus Status change events
 - Application Status change events
 - Error events (logical or protocol-wise)
 - Error response messages
 - Application error status events
 - Anybus error status events
 - SPI CPOL/CPHA/SS settings mismatch
 - CRC32 error events
 - Retransmissions (partial support)
 - Buffer full (warning event)
 - New messages (response, command, error response) with support for indication of SPI fragmentation protocol.
- Analyzer export options (uses CSV format with '\t' delimiter)
 - Export of:
 - All frame data
 - All message data
 - All process data

3 ABCC SPI Protocol Plugin Overview

3.1 Bubble Text

Each of the bubble-text frames (a term used by the Saleae Logic SDK that refers to the blue bars seen in the software) correspond to fields within an ABCC SPI packet. When applicable, these fields are processed and converted into human readable enumerations of the flags and values that make up the underlying ABCC communication. For instance, the bubble text frame shown in the figure below shows “ANB_STS: (PROCESS_ACTIVE | SUP)”. This indicates that the ABCC is in the process active state and is being supervised.

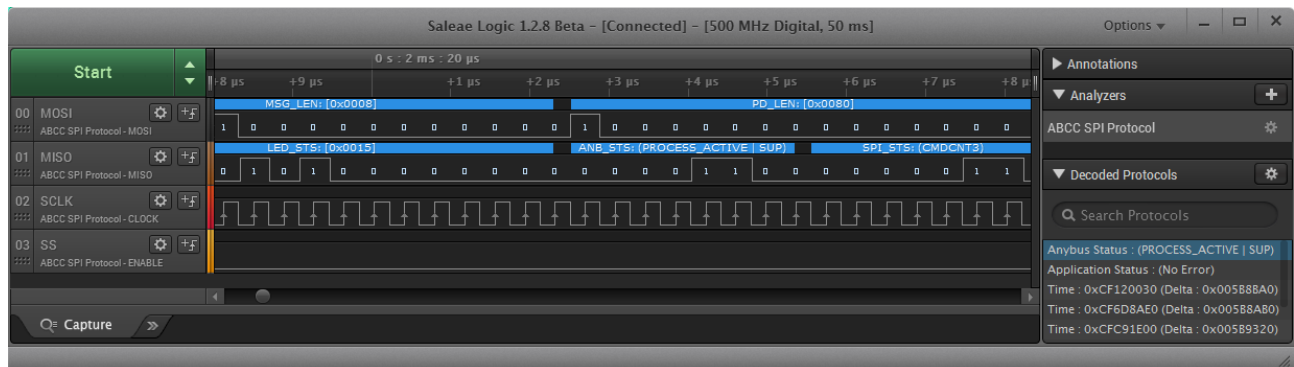


Figure 3: Example capture using the plugin

3.1.1 Multi-Layer Bubble Text

The figure below illustrates the multi-layered bubble text that the plugin supports. This example shows how the SPI Control field looks when zoomed out, and how it may look when sufficiently zoomed-in, revealing more verbose information about the state of the field. Additionally, when sufficiently zoomed-in, the logic signal will be drawn with ‘0’ and ‘1’ indicators over the exact sample point in the corresponding channel which reflect the state of the signal during that sample period.

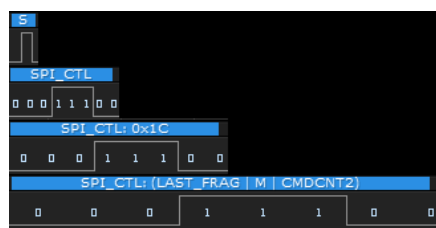


Figure 4: Example showing multi-layered bubble text for the SPI Control Field

To determine if the last, most detailed, bubble-text is currently being displayed the user can make the following observations:

- All frames will contain a single-character bubble-text entry
 - o This character will either be the first character of the “tag” or an “alert” represented by ‘!’.
 - o The currently used tag names include:

TAG	NAME
RES	Reserved
LED_STS	LED Status
ANB_STS	Anybus Status
SPI_STS	SPI Status
TIME	Network Time
MD_SIZE	Message Data Field Size

SRC_ID	Source Identifier
OBJ	Object
INST	Instance
CMD	Command
RSP	Response
ERR_RSP	Error Response
EXT	Command Extension (EXT0/EXT1)
MD	Message Data
ERR_CODE	Response Error Code
PD	Process Data
CRC32	32-bit Cyclic Redundancy Check
PAD	Pad
SPI_CTL	SPI Control
PD_LEN	Process Data Field Length
APP_STS	Application Status
INT_MSK	Interrupt Mask
--	Data Not Valid

- All frames will contain a “tag” bubble-text entry
 - o This entry is meant to be short but readable enough for the user to determine its meaning when sufficiently familiar with the ABCC SPI telegram fields.
 - o In the case of an “alert”, the tag will be prefixed by “!ALERT -”.
- All frames will contain a “value”
 - o This value will obey the “display radix” options that are part of the Logic software.
- Optionally, frames may contain a descriptive/enumerated bubble-text entry
 - o A user can reasonably determine if there are deeper levels of bubble-text available by the present of parenthesis or square brackets encapsulating the information following the “tag”. A “tag” and the information that follows is always separated by a colon ‘:’.

3.2 Markers

Each analyzer plugin is responsible for drawing their own markers onto channels; this is an optional feature of a plugin which can be used in a variety of ways to provide higher-level/at-a-glance understanding of a capture or finely detailed indication of events. This plugin uses markers in both ways; the usage of each marker will be described below.

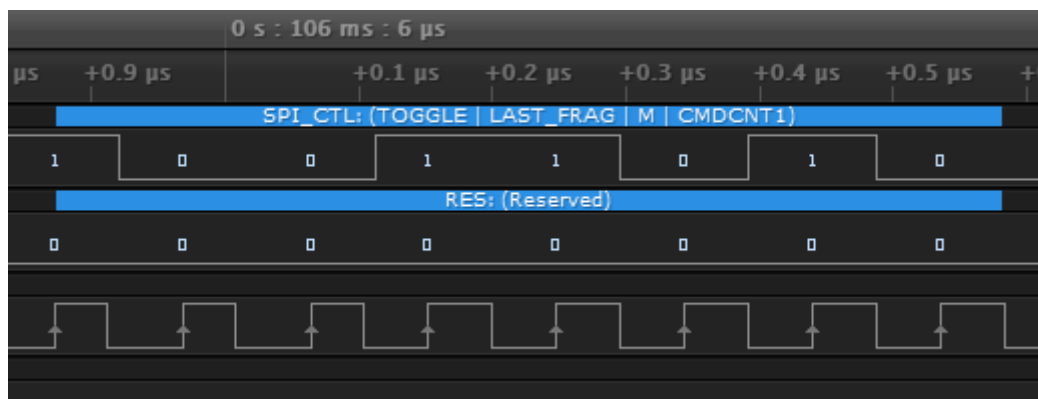


Figure 5: Example illustrating the usage of sample point markers

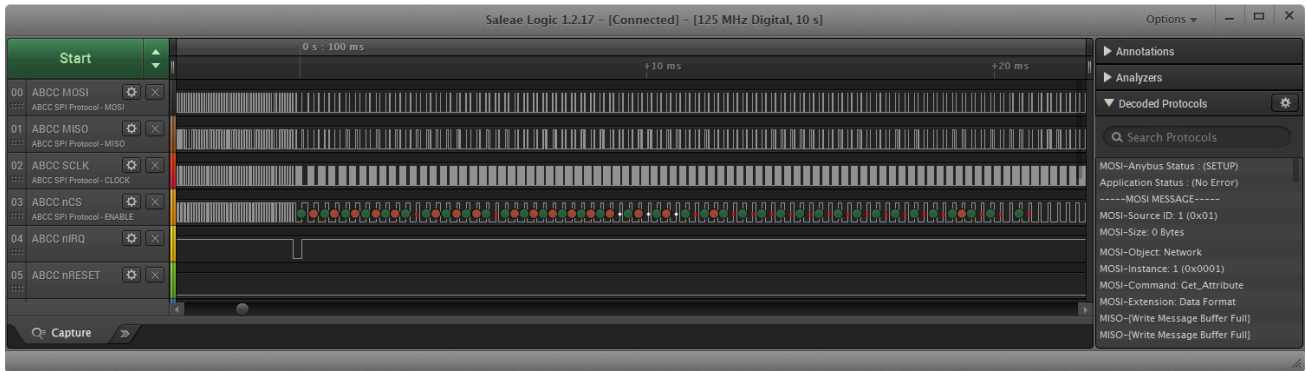












Figure 6: Example illustrating the usage of markers on the SPI enable channel

Note: To help with locating the actual error event, “error markers” (dots, squares, or x’s are placed on the Slave Select signal where the beginning of the bubble-text frame associated with the error event occurred. The current version of Saleae Logic software has some issues with resolving markers that are drawn in close proximity to each other. To work around this issue such markers are relocated to the slave select line to help maximize the likelihood that the marker will be drawn on the screen. Ideally, a future update of the Logic software would remove this limitation. In such an event, these markers may be relocated to reside on the physical channel where the error event was detected.

Marker	SDK Name	Applicable Channels	Usage
	Start	Enable	Used to indicate an ABCC message command
	Stop	Enable	Used to indicate an ABCC message response (no error)
	Dot	Enable	Used to indicate ABCC message fragmentation
	Square	Enable	Used to indicate that multiple events have occurred (no errors)
	Error Dot	Enable	Used to signal an ABCC error response message.
	Error Square	SCLK, Enable	Used to indicate a clocking/protocol violation or that multiple events have occurred with atleast one error event.
	Error X	Enable	Used primarily to indicate an error in the CRC32 field or to signal that a packet has been “cancelled” from further processing due to byte aquisition error.
	Up Arrow	SCLK	Used to indicate the clock sample phase.
	One	MOSI, MISO	Used to indicate the sample point fed into the byte acquisition logic. In this case a logic ‘1’ was read.
	Zero	MOSI, MISO	Used to indicate the sample point fed into the byte acquisition logic. In this case a logic ‘0’ was read.

3.3 Measurement

The current generation of Saleae Logic hardware adds analog input functionality and can perform basic analog measurements. This helps in spot-checking signal quality when troubleshooting a new hardware design. For more serious analog measurement a traditional oscilloscope should be used.

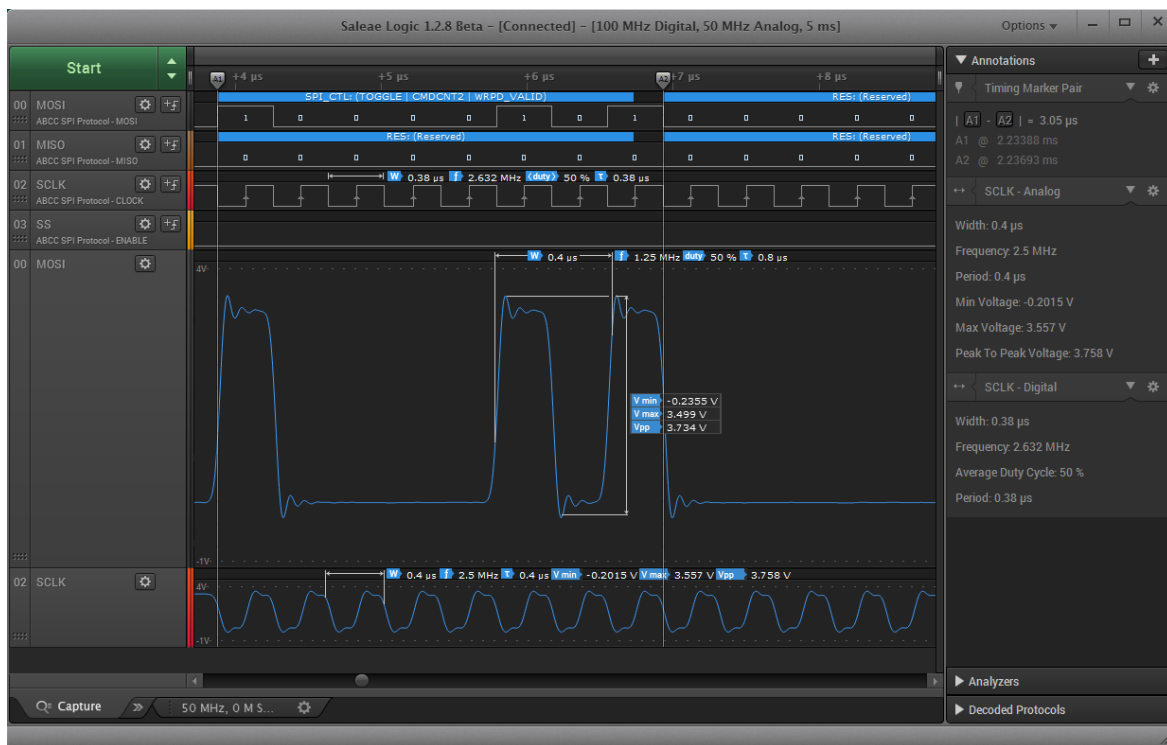
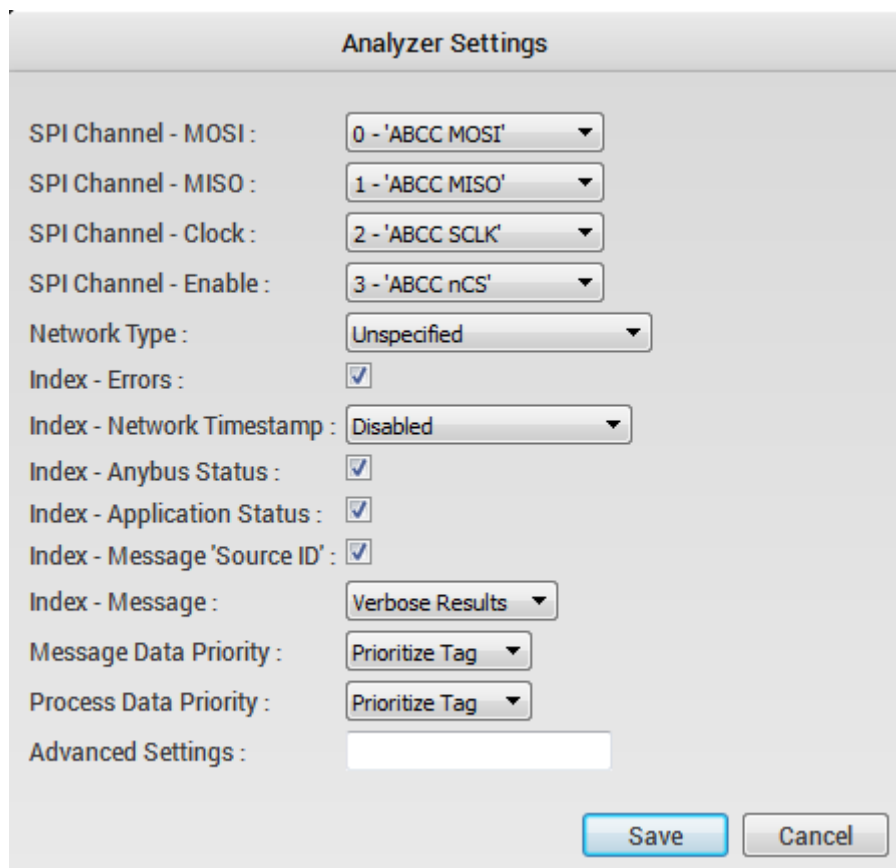


Figure 7: Example taking physical measurements

3.4 Analyzer Settings Window

The figure below shows the default state of the analyzer settings.



The 'Analyzer Settings' dialog box contains the following settings:

- SPI Channel - MOSI : 0 - 'ABCC MOSI'
- SPI Channel - MISO : 1 - 'ABCC MISO'
- SPI Channel - Clock : 2 - 'ABCC SCLK'
- SPI Channel - Enable : 3 - 'ABCC nCS'
- Network Type : Unspecified
- Index - Errors : ☒
- Index - Network Timestamp : Disabled
- Index - Anybus Status : ☒
- Index - Application Status : ☒
- Index - Message 'Source ID' : ☒
- Index - Message : Verbose Results
- Message Data Priority : Prioritize Tag
- Process Data Priority : Prioritize Tag
- Advanced Settings : (empty text field)

Buttons: Save, Cancel

Figure 8: Available options for analyzer

3.4.1 Network Type

This option allows the plugin to have an awareness of the network type which makes it possible to report network specific details. For example, this feature is used to provide the instance name of each instance in the Network Configuration object. The default state of this option is “Unspecified”.

3.4.2 Index - Errors

When enabled, this option will index each “error” event. The default state of this option is “enabled”. To make searching for issues as easy as possible, all error events are prefixed with a “!”. An error is classified as any of the following:

- SPI settings mismatch (CPOL, CPHA, Slave-Select Active Level, etc.)
- Protocol Fragmentation
 - o Not to be confused with ABCC SPI fragmentation protocol. This error is indicated when Slave-Select de-asserts before a complete ABCC transaction completes.
- SPI Clocking Errors
- Error Response Messages
- Application Status is in an “error state”
- Anybus Status is in an “error state”

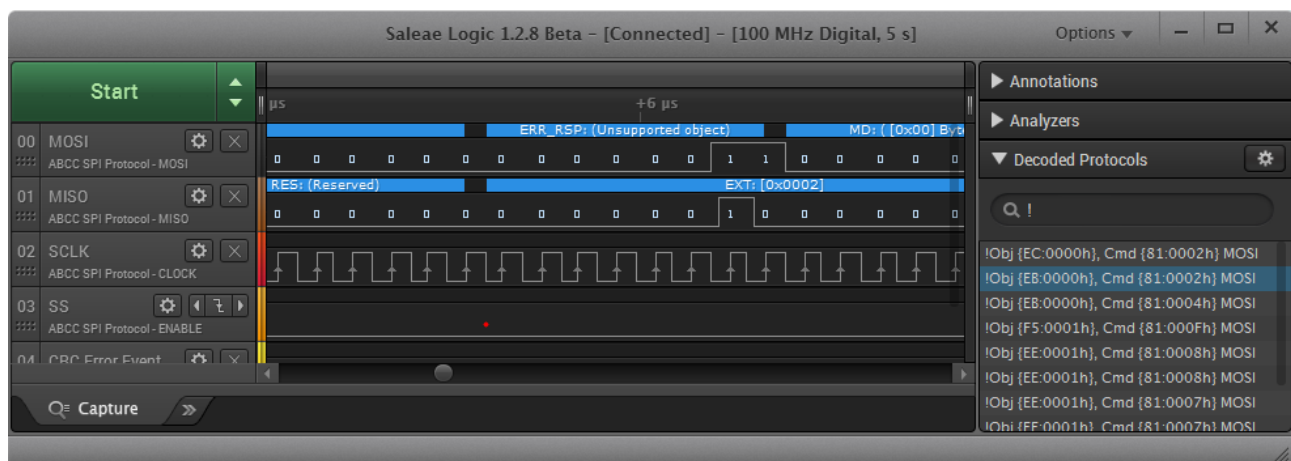


Figure 9: Error indexing

3.4.3 Index – Network Timestamp

When enabled, this option will index each “network timestamp” received from the MISO ABCC message. The default state of this option is “disabled”.

Note: The Saleae Logic software currently does not support adding multiple indexed results to the “Decoded Protocols” sub-window for separate bubble-text entries that exist in the same time-space (even though they reside in separate channels). This means when enabling both “Timestamp Indexing” and “Application Status Indexing” only one entry in the decoded protocols can exist. Currently, priority is given to the “Application Status” events. A future update to the plugin may resolve this limitation.

3.4.4 Index - Anybus Status

When enabled, this option will index each “Anybus status event” received from the MISO ABCC message. The default state of this option is “disabled”. Even when disabled, if error indexing is enabled, any Anybus state related to an error/exception will be indexed.

3.4.5 Index - Application Status

When enabled, this option will index each “Application status event” received from the MISO ABCC message. The default state of this option is “disabled”. Even when disabled, if error indexing is enabled, any application state related to an error/exception will be indexed.

3.4.6 Index - Message

When enabled, this option will index each “New message” in the “Decoded Protocols” sub-window. Retransmissions, commands, and (error) responses all fall into this category. This setting can be set to “compact”, “verbose”, or “disabled”. The default state of this option is “verbose”.

Verbose Decoding

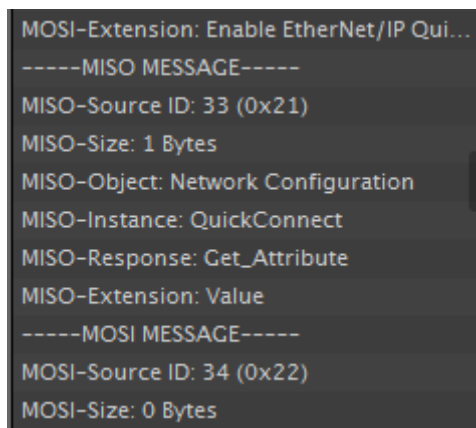


Figure 10: Example of verbose decoding

Compact Decoding

Example Decodes:

Obj {03:0001h}, Cmd {41:0003h} MOSI

A new message for object 0x03 (Network object) instance 0x0001 with command byte 0x41 (Get_Attribute) for attribute 0x03 (Data format). To keep things compact (since the Saleae Logic software does not have a resizable decodes window) the message fields are represented in the decoded section as raw hexadecimal format.

Obj {F8:0001h}, Cmd {01:0006h} MOSI++

Like the previous example this indicates a new message. The presence of “++” at the end of the entry indicates that this is message is using the ABCC’s SPI fragmentation protocol.

{Message Fragment} MOSI++

Indicates the corresponding message data is a continuation of the ABCC’s SPI fragmentation protocol. The presence of “++” at the end of the entry indicates that this is message is not the “Last Fragment”. An example of this is shown in the figure below.

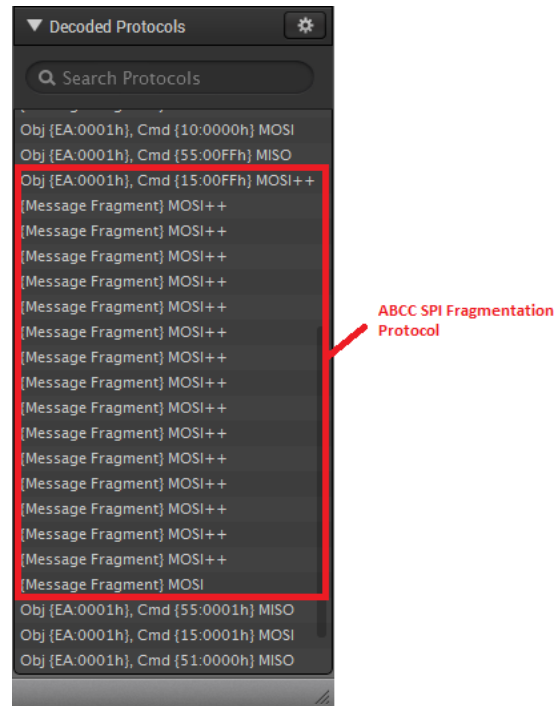


Figure 11: Message fragmentation protocol

3.4.7 Message Data Priority

This option allows the user to prioritize either the “tag” or the “data” for each message data bubble-text entry.

- “Prioritize Tag”
 - Use this option when only interested in knowing where the message data begins and ends within the telegram.
- “Prioritize Data”
 - Use this option when interested in seeing each adjacent message data byte.

3.4.8 Process Data Priority

This option allows the user to prioritize either the “tag” or the “data” for each process data bubble-text entry.

- “Prioritize Tag”
 - Use this option when only interested in knowing where the process data begins and ends within the telegram.
- “Prioritize Data”
 - Use this option when interested in seeing each adjacent message data byte.

3.4.9 Advanced Settings

This option can specify an external XML configuration file that contains extra options that may be useful in certain situations. Below is a description of each of these options. The default state of this setting is an empty file path string (this means the default state of the advanced settings will be used).

```
<?xml version="1.0" encoding="utf-8"?>
<AdvancedSettings>

  <!-- "4-wire-on-3-channels" is useful when a 4-wire SPI configuration is used to
  interface with the ABCC but only MOSI, MISO, and SCLK are connected to the Logic
  analyzer. This mode will ignore timing and CPOL/CPHA requirements that the ABCC's
  3-wire mode enforces. -->
  <Setting type="4-wire-on-3-channels">0</Setting>

  <!-- "3-wire-on-4-channels" is useful when a 3-wire SPI configuration is used to
  interface with the ABCC but the user has a spare Logic channel that can be used
  as a dummy place holder for placing the markers that the plugin normally applied
  to the SPI Enable channel. -->
  <Setting type="3-wire-on-4-channels">0</Setting>

</AdvancedSettings>
```

- 4-wire-on-3-channels
 - This setting can be set to 1 to allow the plugin to ignore timing and CPOL/CPHA requirements typically required when using the plugin with only 3 channels specified (i.e. SCLK omitted). This can be useful if the user needs to free-up a logic channel to monitor another signal but would otherwise run into timing or CPOL/CPHA errors reported by the plugin. The default state of this setting is 0 (disabled).
 - It may be required for the user to specify the starting point for the analyzer to begin from in order for the plugin to be synchronized to the actual transactions.
- 3-wire-on-4-channels
 - This setting can be set to 1 to allow the plugin to overlay “markers” on the 4th channel (i.e. dummy slave select channel). This channel does not actually function as a slave select and is only used by the analyzer to draw the markers that otherwise would not be displayed in the capture when using 3-wire mode. The default state of this setting is 0 (disabled).

3.5 Export Options

This plugin supports several options for exporting the captured/processed data. These options make it possible (after a little data manipulation) to:

- Generate process data graphs
 - o For instance, the trend of an analog input/output signal could be plotted versus the network time.
 - o Analyze jitter characteristics of reporting new input data or receiving new output data.
- Collect and assemble message data that was sent using SPI fragmentation

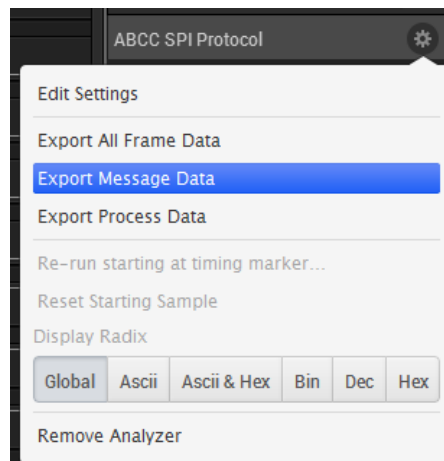


Figure 12: Options for data export

For instance, a file transfer to/from a File System Interface instance could be captured and easily reassembled into a matching file using the exported capture data for verification purposes.

4 Troubleshooting

4.1 USB Ports

In certain cases, the Logic software may report that it cannot keep up using the current sample rate.

To avoid this scenario, try the following:

- Reduce the pre-trigger buffer size (found in Options->Preferences->Capture)
- Reduce the sample rate (as requested by the software)
- Reduce the capture size
- Remove/disable unnecessary channels from capture
- Remove unnecessary USB peripherals from the USB controller that the Logic is connected to.
- Move the Logic hardware to another USB controller
- Replace the USB cable with a shorter and/or higher quality cable. Poor signal quality may introduce bit errors across the USB bus and will require retransmission. Such events may fill up the Logic's buffer which could result in a buffer overrun.
- Check Saleae Q&A links below for details regarding USB3 host drivers and controllers
 - o [Recommended-USB-3-0-host-controller-cards](#)
 - o [Latest-USB-3-0-Host-Controller-Drivers](#)

4.2 System Memory

For the logic to maintain high sampling rates and large capture sizes, the Logic offloads samples into system memory. This means the Logic software increases in RAM utilization with higher sample rates and larger capture sizes. With the current version of the Logic software, the handling of an "out-of-memory" event is less than graceful. When the Logic software runs out of memory, it will simply cash and all unsaved settings or captures will be lost.

To avoid this scenario, try one or more of the following:

- Reduce capture size
- Reduce sample rate
- Remove/disable unnecessary channels from capture
- Save/Close other capture tabs
- Close down unnecessary applications that are consuming system memory (see operating system's task manager for details on utilizations)
- Upgrade system memory

As a rough estimate, 3 seconds of 4-channel capture data at 50MSamples/second with a host using 10Mbps SPI (4-wire) and approximately 3000 SPI packets per second that is fully processed through the plugin could consume around 1.2GB of RAM. The more logic transitions there are the more memory will be utilized due to the way the analyzer compresses logic data.