# Quick Start Guide

## ABCC SPI Protocol Analyzer

### A Plugin for Saleae Logic Software

# 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

## 1.1  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
    - Exception codes and information
    - Application error status events
    - Anybus error status events
    - Fragmented packets or clocking error events
    - CRC32 error events
    - Retransmissions (partial support)
    - Buffer full (warning event)
  - New messages (response, command, error response) with support for indication of SPI message fragmentation protocol and message segmentation.
- Analyzer export options (uses CSV format with ',' delimiter by default)
  - Export of:
    - All frame data
    - All message data
    - All process data
- Simulation mode
  - Standard simulation
    - Produces a file object communication with various random events including error events.
    - Conveys plugin's version information.
  - ABCC SDK log file simulation
    - Generate a sequence of SPI packets conveying object messaging and Anybus Status information.
  - User configurable options for modifying underlying signaling, timing, and messaging characteristics.

# 1.2  System Requirements

**Recommended:**

- Saleae Logic (software)
    - Recommended Version: 1.2.29
    - Other versions will work so long as it is compatible with Logic SDK version 1.1.32.
    - NOTE: At this time, Saleae Logic V2 is not supported.
- Saleae Logic analyzer (hardware)
- Operating system
    - Microsoft Windows 7 through Windows 10
        - NOTE: At this time Window 11 has compatibility issues with version 1.2.29. Version 1.2.40 resolves this compatibility issue but Saleae dropped support for the decoded protocols subwindow for this version. An alternative solution for using 1.2.29 on Windows 11 would be to use **WSLg** and attach the Saleae Logic interface via **usbpid.exe**.
    - Mac OS X 10.7 Lion+ (or macOS)
    - Ubuntu 12.04.2+
- Precompiled ABCC SPI Analyzer plugin

## 1.3  Installation

To install this plugin, it is assumed that the user has previously installed the Saleae Logic software using the default installation options. It is also assumed the default installation path is used (where applicable). Copy the corresponding plugin source to the indicated destination path for the operating system being used as indicated in the table below.

| OS | PLUGIN SOURCE | DESTINATION PATH |
|---|---|---|
| **WINDOWS 64-BIT** | `.\plugins\Win64\AbccSpiAnalyzer64.dll` | `C:\Program Files\Saleae LLC\Analyzers\` |
| **WINDOWS 32-BIT** | `.\plugins\Win32\AbccSpiAnalyzer.dll` | `C:\Program Files\Saleae LLC\Analyzers\` |
| **LINUX 64-BIT (UBUNTU)** | `.\plugins\Linux64\AbccSpiAnalyzer64.so` | `.\Analyzers` folder of the Logic software installation path |
| **LINUX 32-BIT (UBUNTU)** | `.\plugins\Linux32\AbccSpiAnalyzer.so` | `.\Analyzers` folder of the Logic software installation path |
| **MACOS (OSX)** | `.\plugins\OSX\AbccSpiAnalyzer64.dylib` | `.\Analyzers` folder of the Logic software installation path |

# 2 ABCC SPI Protocol Plugin Overview

## 2.1  Running the Analyzer

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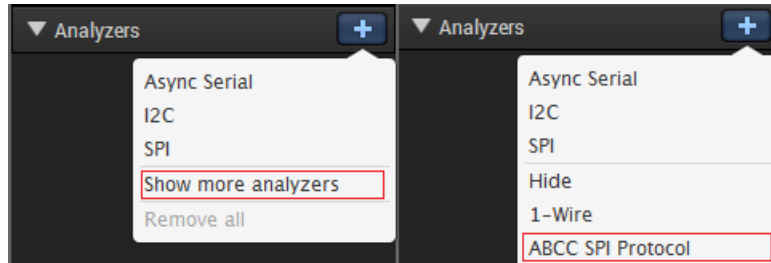


**Figure 1 : Adding the analyzer**

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, the most commonly used options, but can be changed according to the user's needs.
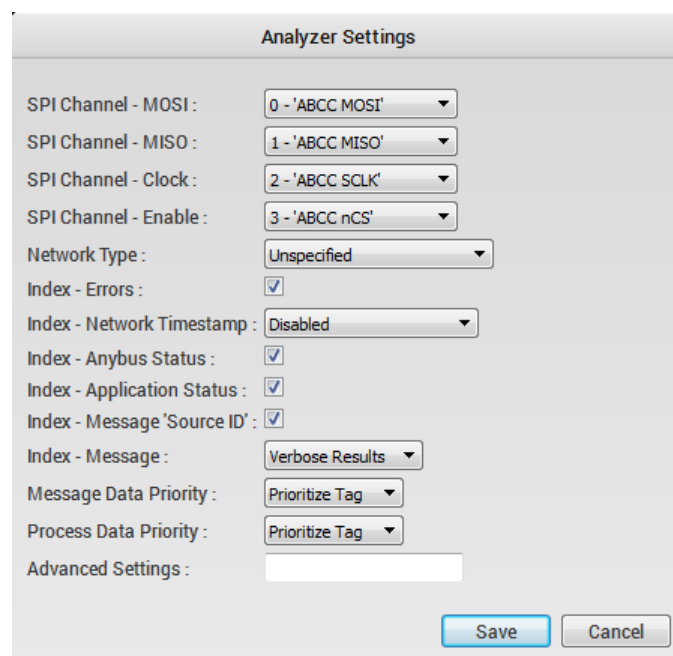


**Figure 2 : Analyzer settings window**

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.*

## 2.2 Framing (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.
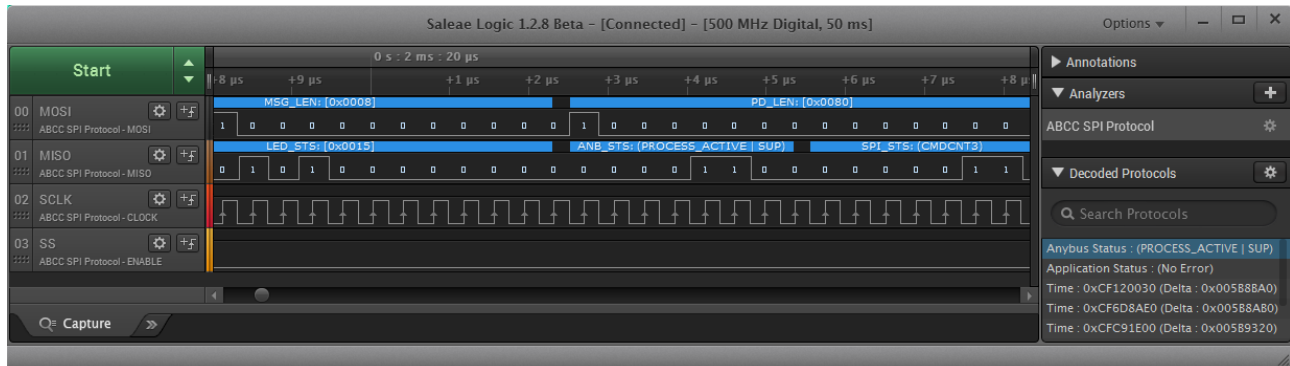


**Figure 3: Example capture using the plugin**

### 2.2.1    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.
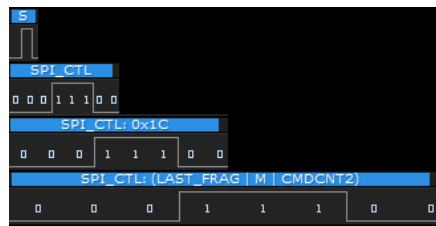


**Figure 4: Example showing multi-layered bubble text for the SPI Control Field**

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
  - This character will either be the first character of the "tag" or an "alert" represented by '!'.
  - The currently used tag names are defined in the following section.
- All frames will contain a "tag" bubble-text entry
  - 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.
  - In the case of an "alert", the tag will be prefixed by "!ALERT –".
- All frames will contain a "value"
  - This value will use an appropriate "display radix" option for a given field. Fields that are naturally numeric values will not be permitted to display as a "character" value and will restrict to using hex-format when the user sets the "display radix" option as ASCII from 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 presence of parenthesis or square brackets encapsulating the information following the "tag". A "tag" and the information that follows is always separated by a colon ':'.

## Tag Descriptions

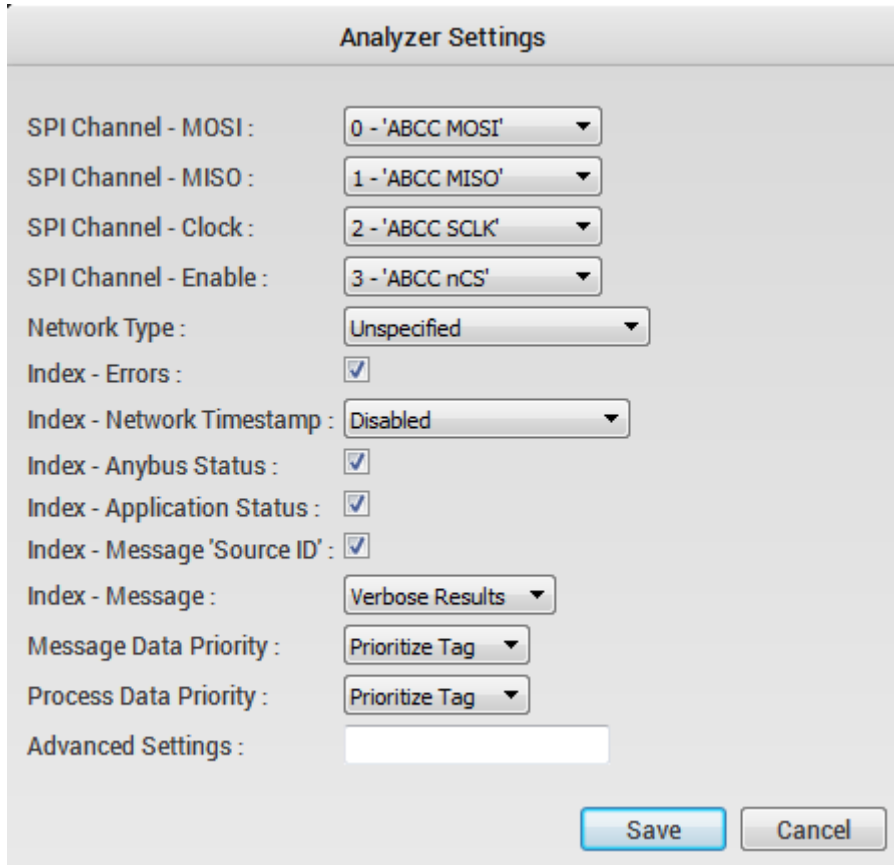| TAG | NAME | ADDITIONAL DETAILS |
|---|---|---|
| !ALERT | Alert Indicator | Prepended to another tag as indicator of important events (commonly to signal an error of some kind) |
| FRAG | Fragmented Packet | Indication of an incomplete ABCC SPI packet detected. |
| SPI_CTL | SPI Control Field | (MOSI) SPI control byte |
| RES | Reserved Field | - |
| MSG_LEN | Message Data Length Field | (MOSI) Length in WORDs (16-bit) of the "Message Data Field" |
| PD_LEN | Process Data Length Field | (MOSI) Length in WORDs (16-bit) of the "Process Data Field" |
| LED_STS | LED Status Field | (MISO) LED status |
| ANB_STS | Anybus Status Field | (MISO) Anybus status |
| SPI_STS | SPI Status Field | (MISO) SPI status |
| APP_STS | Application Status Field | (MOSI) Application status |
| INT_MSK | Interrupt Mask Field | (MOSI) Mask specifies which interrupt events are generated by the module. |
| TIME | Network Time Field | (MISO) Network time reported from the module |
| MSG_SIZE | Message Size Field | Size of the message data in bytes. This corresponds to the actual payload of the message which may exceed MSG_LEN, in which case message fragmentation is used. |
| SRC_ID | Message Source ID | Source ID for a message transaction |
| OBJ | Object | Host/Module object being accessed |
| INST | Instance | Instance of the object being accessed |
| CMD | Command Message | Message contains a command. |
| RSP | Response Message | Message contains a response. |
| ERR_RSP | Error Response Message | Message contains an error response. Error code is present in message data. |
| EXT | Command Extension | 16-bit Command extension data. |
| MD | Message Data | Valid Message Data (message data byte offset < message size) |
| ERR_CODE | Error Response Code | Indication of an Error Response Code (first byte) |
| -- | Data Not Valid | Indication of "data not valid". Used in leftover message data. |
| PD | Process Data | Indicates either read/write process data |
| CRC32 | CRC32 Checksum | 32-Bit CRC of the channel's SPI data |
| PAD | Pad Field | (MOSI) Dummy Data |

## 2.3  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 is described below.

| 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) |
| ✕ | X | Enable | Used to indicate one of the following protocol events occurred: Anybus status change (including error and exception states), message buffer full, application status change, and SPI packet retransmission. |
| • | 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 at least 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 acquisition error. |
| ▲ | Up Arrow | SCLK | Used to indicate the clock sample phase. |
| 1 | One | MOSI, MISO | Used to indicate the sample point fed into the byte acquisition logic. In this case a logic '1' was read. |
| 0 | Zero | MOSI, MISO | Used to indicate the sample point fed into the byte acquisition logic. In this case a logic '0' was read. |

**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 mitigate this issue as much as possible, 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.

# 2.4 Analyzer Settings Window

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



**Figure 5: Available options for analyzer**

## 2.4.1 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".

## 2.4.2 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:

- 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"
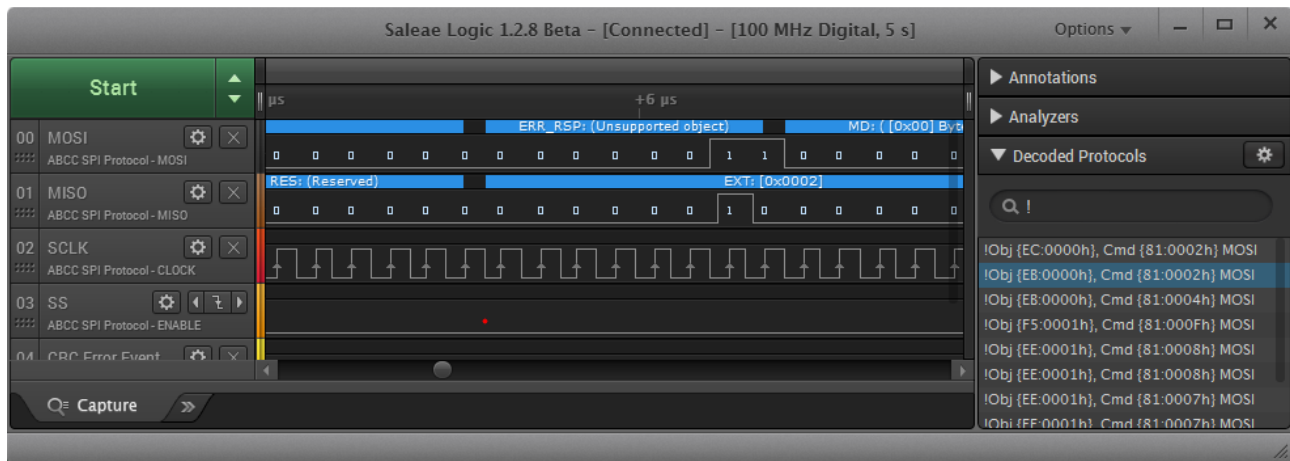- Anybus Status is in an "error state"

**Figure 6: Error indexing**

## 2.4.3    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.

## 2.4.4    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.

## 2.4.5    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.

## 2.4.6 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 7: Example of verbose decoding**

**Compact Decoding**

| | |
|---|---|
| **ENTRY** | MOSI-OBJ {03:0001H}, CMD {01:0003H} |
| **MEANING** | A new message for object 0x03 (Network object) instance 0x0001 with command byte 0x01 (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. |

| | |
|---|---|
| **ENTRY** | MISO-OBJ {03:0001H}, RSP {01:0003H} |
| **MEANING** | This message is identical to the previous example except that it represents a response message. Notice that the bit-field (within the command byte) responsible for indication of whether a message contains a command or a (error) response is filtered from the raw value here. This is done to improve the flexibility of searching the decoded results. |

| | |
|---|---|
| **ENTRY** | MOSI-OBJ {F8:0001H}, CMD {01:0006H}++ |
| **MEANING** | 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. |

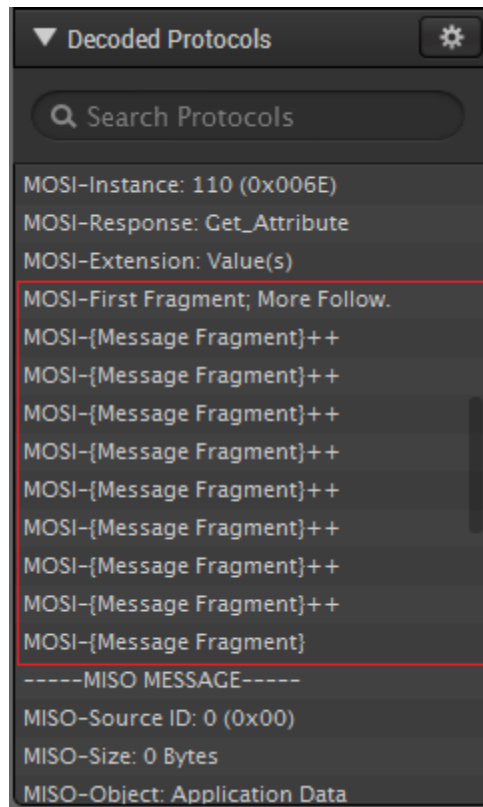| | |
|---|---|
| **ENTRY** | MOSI-{MESSAGE FRAGMENT}++ |
| **MEANING** | Indicates the corresponding message data is a continuation of the ABCC's SPI fragmentation protocol. The presence of "++" at the end of the entry indicates that this is message is not the "Last Fragment". An example of this is shown in the figure below. |

**Figure 8: Message fragmentation protocol**

## 2.4.7    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"
     o    Use this option when only interested in knowing where the message data begins and ends within the telegram.
-    "Prioritize Data"
     o    Use this option when interested in seeing each adjacent message data byte.

## 2.4.8    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"
     o    Use this option when only interested in knowing where the process data begins and ends within the telegram.
-    "Prioritize Data"
     o    Use this option when interested in seeing each adjacent message data byte.

## 2.4.9    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
<?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 name="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 name="3-wire-on-4-channels">0</Setting>

    <!-- "export-delimiter" allows the user to define a delimiter to use when using one of the
    plugin's supported export options. Any single (visible) character is accepted as a delimiter.
    A tab-delimiter can be specified by using "\t" without the quotes. -->
    <Setting name="export-delimiter">,</Setting>

    <!-- "clocking-alert-limit" provides the user with a way to restrict how many times a
    "clocking alert" event will be reported by the plugin. This can be useful in cases where a
    device's SPI chipselect line may "appear" to toggle at the same time the SPI clock transitions
    high after receiving the last bit in the SPI transaction. This event results in a
    "clocking alert" notification. This type of event is more of an informational warning/error
    event, and as long as the user understands the reason and deems the behavior acceptable, then
    this setting may be set to a value >=0. Setting the value to 0 disables the notification
    entirely. While any value above zero (datatype is S32) will set a maximum count for producing
    the notification in the processed results. An invalid value or value <0 indicates that there is
    no limit. Users of the ABCC StarterKit may encounter this scenario since the chipselect line is
    deasserted close in time (~8ns) with the SPI clock returning to the "clock idle high" state.
    This results in a higher than normal sampling time requirement to resolve the two events
    occurring at different points in time. Sampling more than 125MSamples/second is necessary to
    avoid this issue; thus, a general recommendation for this scenario is to use sampling rates of
    250MS/s or higher. If this is not possible due to hardware limitations of the PC/analyzer then
    it is recommended to set this parameter to a value >= 0 to limit how many times the error is
    reported. -->
    <Setting name="clocking-alert-limit">-1</Setting>

    <!-- "expand-bit-frames" provides a workaround for a limitation found in Saleae Logic software
    version 1.2.x. Specifically, the software does not properly navigate to the associated bubble
    text entry when clicking on a "Decoded Protocols" entry for a frame less than 8 samples.
    Currently, this workaround is only applicable for error frames; any error frame less than 8
    samples will be expanded to 8 samples to restore the linkage. This feature could have
    undesirable side effects in rare circumstances. In case of abnormal behavior, it may be necessary
    to disable this feature. -->
    <Setting name="expand-bit-frames">1</Setting>

    <!-- "simulation" provides various options for generating simulated ABCC SPI communication.
    There are two primary modes supported: "standard simulation" and "log file simulation".
    "Standard simulation" involves a general hardcoded procedure for file object communication. This
    simulation functions on various pseudorandom number generators for randomly creating different
    events including various error events. This mode also conveys the plugin's version information.
    "Log file simulation" involves parsing a standard ABCC SDK log file and generating ABCC
    SPI packets that convey these messages. The SDK message logging is activated via
    ABCC_CFG_DEBUG_MESSAGING in abcc_drv_cfg.h. The target platform must have support for
```

```
    ABCC_PORT_DebugPrint() in abcc_sw_port.h. -->
    <Setting name="simulation">
        <!-- Path to the log file to simulate. No quotes, backslash/forward slashes are acceptable.
        Empty or invalid paths will disable "log file simulation" and instead "standard simulation"
        mode will be executed. -->
        <LogFilePath></LogFilePath>

        <!-- Default ABCC state (integer) for "log file simulation". Use one of the raw values
        specified in ABP_AnbStateType. Invalid values will default to SETUP state. -->
        <LogFileDefaultAnbState>0</LogFileDefaultAnbState>

        <!-- SPI clock polarity setting (integer). 0 = Clock Idles Low, 1 = Clock Idles High,
        Else = "Auto-mode". "Auto-mode" changes behavior depending on simulation mode. When running
        "log file simulation", auto-mode will use clock-idle-high for 3-wire mode, and clock-idle-low
        for 4-wire mode. When running "standard simulation" mode, auto-mode will use a pseudorandom
        number generator to toggle between clock-idle-high and clock-idle-low to illustrate both modes
        in one simulation run. NOTE: The ABCC's 3-wire mode only supports clock-idle-high, while in
        3-wire mode clock-idle-high will be enforced. -->
        <SpiClockIdleHigh>-1</SpiClockIdleHigh>

        <!-- SPI clock frequency (integer, in Hertz). Values <= 0 or parsing errors will default to
        auto-frequency (this is a mode where the SPI clock frequency is set, when possible, to
        1/10th the sampling frequency). Values > 20MHz will be limited to 20MHz. 3-wire mode will
        enforce a minimum of 100KHz. Please keep in mind the analyzer's configured sample rate, this
        mode of simulation requires the user to consider Nyquist rate just like in real world logic
        analyzer usage. -->
        <SpiClockFrequency>0</SpiClockFrequency>

        <!-- SPI inter-packet gap (integer, in nanoseconds). Values <= 0 or parsing errors will
        default to 15000ns. -->
        <SpiPacketGapNs>0</SpiPacketGapNs>

        <!-- SPI inter-byte gap (integer, in nanoseconds). Values < half the SPI clock period or
        parsing errors will default to half the SPI clock period. -->
        <SpiByteGapNs>0</SpiByteGapNs>

        <!-- SPI spacing between chip-select going active-low and the first clock edge and the
        spacing after the last clock edge before chip-select returns high (integer, in nanoseconds).
        Note, depending on SPI configuration the spacing on one side will have an additional 1/2
        clock period delay. Values <= 0 or parsing errors will default to 1000ns. -->
        <SpiChipSelectDelayNs>0</SpiChipSelectDelayNs>

        <!-- SPI register data size (in bits). This parameter represents the underlying SPI
        controller register data size and is used to determine when to apply "SpiByteGapNs".
        This value must be 8 or 16 bits; other values or parsing errors will default to 8 bits. -->
        <SpiDataSize>0</SpiDataSize>

        <!-- Configures the message data length field (in words). {size} = 0: 8 words (default),
        0 < {size} <= 762: fixed message data length, {size} < 0: dynamic mode. When in SETUP,
        NW_INIT, or WAIT_PROCESS, "dynamic mode" will adapt the message field length based on the
        number of message data bytes the host is requesting to send effectively avoiding
        SPI message fragmentation. If the host is not trying to send a message, then the message
        channel will default to the absolute value of {size}. While in IDLE or PROCESS_ACTIVE the
        message data length will be set to the absolute value of {size}. When receiving a
        message from the module, the message will be adjusted, if necessary, to limit SPI
        message fragmentation. -->
        <SpiMessageDataLength>0</SpiMessageDataLength>
    </Setting>
</AdvancedSettings>
```

| Setting | Datatype | Default state | Default State Meaning |
|---|---|---|---|
| "4-wire-on-3-channels" | BOOL (0/1) | 0 | Disabled, 3-wire mode will be assumed when omitting the plugin's **ENABLE** channel. |
| "3-wire-on-4-channels" | BOOL (0/1) | 0 | Disabled, 4-wire mode will be assumed when assigning the plugin's **ENABLE** channel. |
| "export-delimiter" | STRING | "," | Comma Delimited CSV-files (quotes are not to be included). Only single-character values are accepted. Delimiter must be a visible character. The exception to these two rules is that the user can specify a tab-delimiter via "\t" without the quotes. |
| "clocking-alert-limit" | SINT32 | -1 | Unlimited clocking alert limit. |
| "expand-bit-frames" | BOOL (0/1) | 1 | Error frames < 8 samples will be expanded to 8 samples to restore linkage between "Decoded Protocols" and "Bubble Text". |
| "simulation" | STRUCTURE | N/A | Contains multiple sub-settings. See "Simulation sub-setting" table below. |

| Simulation Sub-Setting | Datatype | Default state | Default State Meaning |
|---|---|---|---|
| LogFilePath | File Path | Empty string | Invalid or empty log file paths will default to standard simulation mode. |
| LogFileDefaultAnbState | SINT32 | 0 | Invalid values (any value not part of ABP_AnbStateType) will default to 0 (SETUP state). |
| SpiClockIdleHigh | SINT32 | -1 | Values not equal to 0 or 1 or parsing errors will default to auto-mode. |
| SpiClockFrequency | SINT32 (Hertz) | 0 | Values <= 0 or parsing errors will default to auto-frequency. |
| SpiPacketGapNs | SINT32 (nanoseconds) | 0 | Values <= 0 or parsing errors will default to 15000ns. |
| SpiByteGapNs | SINT32 (nanoseconds) | 0 | Values < half the SPI clock period will default to half the SPI clock period. |
| SpiChipSelectDelayNs | SINT32 (nanoseconds) | 0 | Values <= 0 or parsing errors will default to 1000ns. |
| SpiMessageDataLength | SINT32 (words) | 0 | Invalid values or value == 0 will default to 8 words. |

# 2.5  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
  - For instance, the trend of an analog input/output signal could be plotted versus the network time.
  - Analyze jitter characteristics of reporting new input data or receiving new output data.
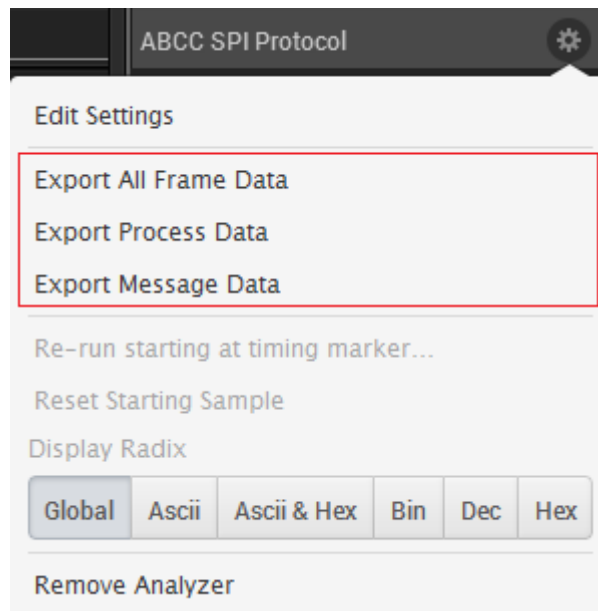- Collect and assemble message data that was sent using SPI fragmentation



**Figure 9: Options for data export**

## 2.5.1    Export All Frame Data
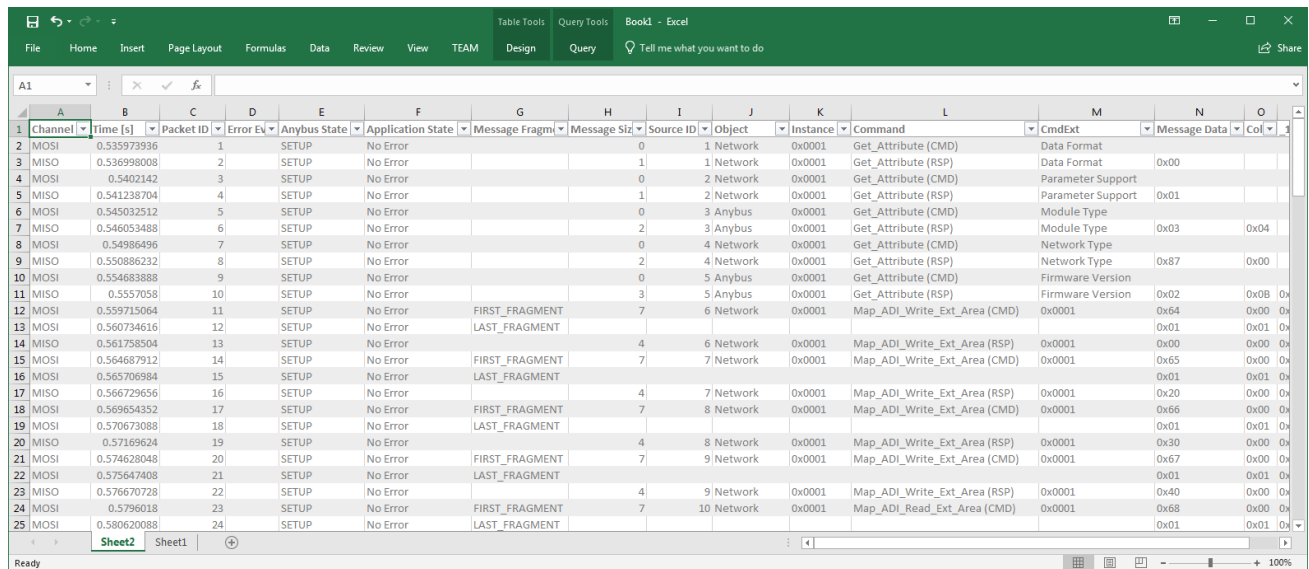
This option will individually export each bubble-text/frame as its own line in a CSV file. This option may be useful to perform additional postprocessing of results in cases where the other two export options may omit desired information.

## 2.5.2    Export Process Data

This option will export information that is most relevant when considering the SPI process data field. Only SPI packets containing new or valid process data, as indicated by the control/status bytes, will be exported.

## 2.5.3     Export Message Data

This option will export information that is most relevant when considering the SPI message data field. Only SPI packets containing a message, as indicated by the control/status bytes, will be exported. This option is the ideal choice when the user wants to view a series of object messaging transactions exchanged between the host and module. 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. Below is an example of exporting message data during the startup process of the ABCC SDK.

# 2.6  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. When using this functionality, keep in mind that the analyzer has anti-aliasing filters which will smooth out the signal and remove out-of-band high frequency content. Sampling around 10x the signal's frequency is recommended if any precise measurements are required. For more serious analog measurement, a traditional oscilloscope is recommended; however, having digitally decoded SPI communication alongside of time-correlated analog waveforms can be invaluable when debugging certain signal quality issues.
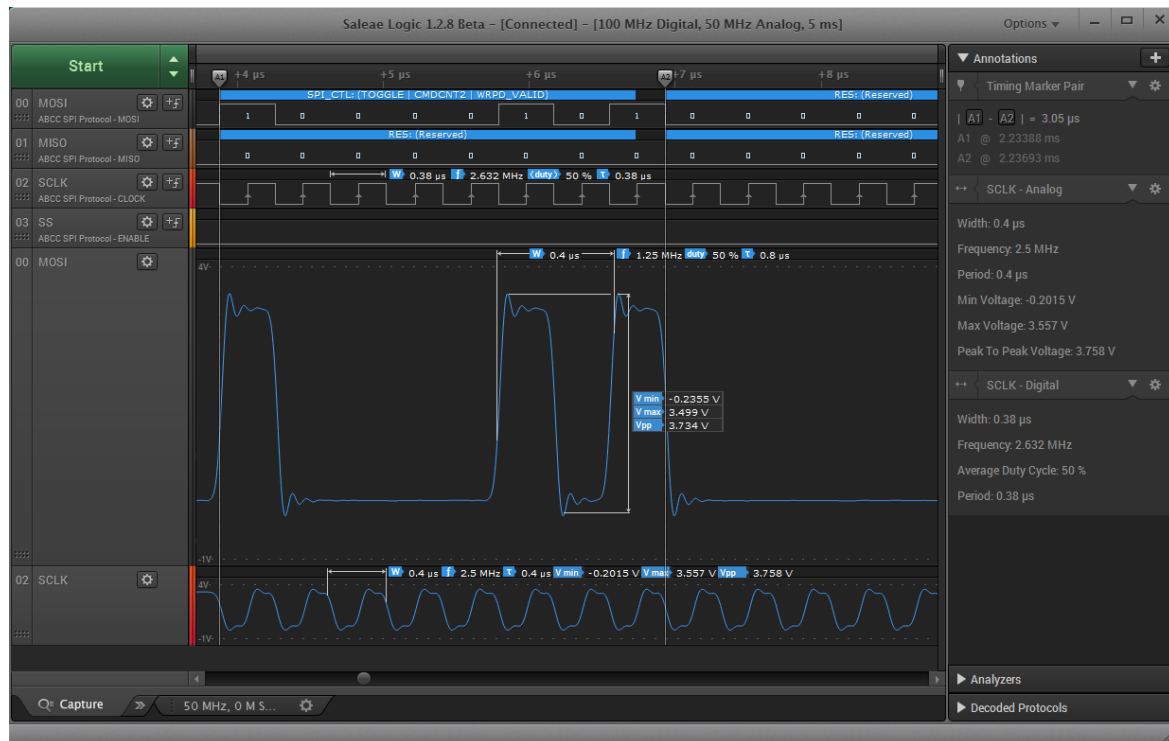


**Figure 10: Example taking physical measurements**

# 3 Troubleshooting

## 3.1  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
    - [Recommended-USB-3-0-host-controller-cards](Recommended-USB-3-0-host-controller-cards)
    - [Latest-USB-3-0-Host-Controller-Drivers](Latest-USB-3-0-Host-Controller-Drivers)

## 3.2  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.