# AN 1005: Signal Tap Logic Analyzer Getting Started Tutorial

Updated for Intel® Quartus® Prime Design Suite: **23.4**

**Answers to Top FAQs:**

Q ***What is Signal Tap?***
A About This Application Note on page 3

Q ***What do I need to use this document?***
A Tutorial Prerequisites on page 3

Q ***How do I use Signal Tap?***
A Signal Tap Tutorial Walkthrough on page 6

Q ***How do I program for Signal Tap?***
A Programming the FPGA for Signal Tap on page 7

Q ***How do I set the trigger conditions for capture?***
A Set Trigger Conditions on page 12

Q ***Where can I go for more Signal Tap Info?***
A Design Debugging with Signal Tap

# Contents

Send Feedback

# 1. About This Application Note

This application note demonstrates how to debug a design with the Signal Tap Logic Analyzer. The Signal Tap logic analyzer, available in the Intel® Quartus® Prime Pro Edition software, captures and displays the real-time signal behavior in an Intel FPGA design.

You can use the Signal Tap logic analyzer to probe and debug the behavior of internal signals during normal device operation, without requiring I/O pins or external lab equipment. For more information on using all of the features of the Signal Tap logic analyzer, refer to the following available resources.

**Related Information**

- Intel Quartus Prime Pro Edition User Guide: Debug Tools
- Intel FPGA Technical Training

## 1.1. Tutorial Prerequisites

Use of this tutorial requires the following:

- Installation and basic familiarity with the Intel Quartus Prime Pro Edition software version 23.4 or later. The Intel Quartus Prime software includes the Signal Tap logic analyzer and the Programmer.
- The Intel Arria® 10 SX SoC Development Kit, or a design board with a JTAG connection to the device under test (DUT).
- An Intel FPGA Download Cable II (USB-Blaster II), for communication between the FPGA and the Intel Quartus Prime software.
- The Intel Arria 10 FPGA - Signal Tap Logic Analyzer Getting Started Design Example, available from the Intel FPGA Design Store.

**Related Information**

- Intel Quartus Prime Pro Edition User Guide: Getting Started
- Intel FPGA Download Center
- Intel Arria 10 SX SoC Development Kit
- Intel FPGA Download Cable II

## 1.2. Signal Tap Tutorial Design Description

The design for this tutorial consists of the `counter_50M` 26-bit counter that counts to 49,999,999 and rolls over to 0.

For every 49,999,999 of `counter_50M`, `counter_10` increases by 1 from 0 to 9 and rolls over to 0.

The output register `seven_seg` corresponds to the value of `counter_10` from 0 – 9 in the form of a common anode value for a seven segment display.

Table 1. Value Representation for count_binary and seven_seg Registers

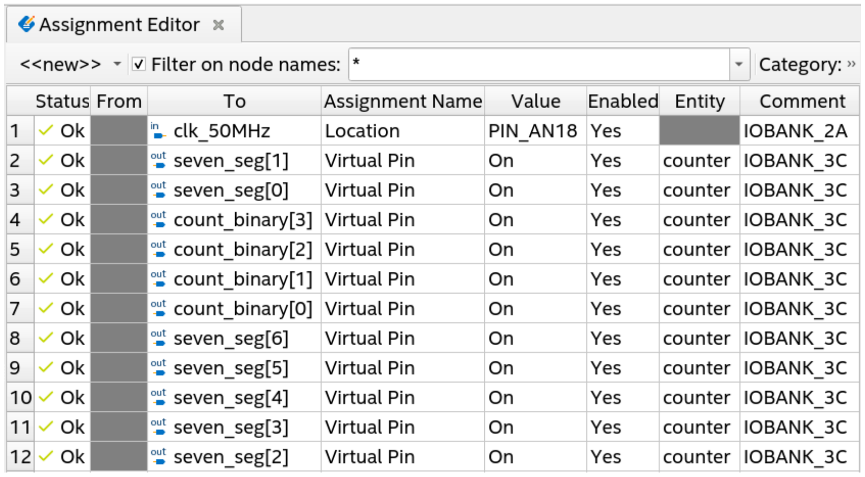
| Decimal Representation | Binary Representation [3:0] | Seven Segments [6:0] |
|---|---|---|
| 0 (zero) | 0000 | 1000000 |
| 1 (one) | 0001 | 1111001 |
| 2 (two) | 0010 | 0100100 |
| 3 (three) | 0011 | 0110000 |
| 4 (four) | 0100 | 1001001 |
| 5 (five) | 0101 | 0010010 |
| 6 (six) | 0110 | 0000010 |
| 7 (seven) | 0111 | 1111000 |
| 8 (eight) | 1000 | 0000000 |
| 9 (nine) | 1001 | 0010000 |

At the board level, the tutorial design connects the clock to a 50 MHz source, and connects the outputs to the `seven_seg` and `count_binary` registers. These outputs do not connect to any physical pins, but only to virtual pins. This example uses virtual pins because the Intel Arria 10 SX SoC Development Kit does not include a seven segment display.

A virtual pin is an I/O element that the Intel Quartus Prime Compiler temporarily maps to a logic element during compilation, rather than mapping to a pin. For more information on using virtual pins, refer to *Defining Virtual Pins* in *Intel Quartus Prime Pro Edition User Guide: Design Optimization*.

If you are using other boards or devices, you can change `clk_50MHz` to another general-purpose clock. To change the pin for `clk_50MHz`, click **Assignments ➤ Assignment Editor** after opening the project.

Figure 1. Pins Used By Signal Tap Tutorial Design



|  | Status | From | To | Assignment Name | Value | Enabled | Entity | Comment |
|---|---|---|---|---|---|---|---|---|
| 1 | ✓ Ok |  | clk_50MHz | Location | PIN_AN18 | Yes |  | IOBANK_2A |
| 2 | ✓ Ok |  | seven_seg[1] | Virtual Pin | On | Yes | counter | IOBANK_3C |
| 3 | ✓ Ok |  | seven_seg[0] | Virtual Pin | On | Yes | counter | IOBANK_3C |
| 4 | ✓ Ok |  | count_binary[3] | Virtual Pin | On | Yes | counter | IOBANK_3C |
| 5 | ✓ Ok |  | count_binary[2] | Virtual Pin | On | Yes | counter | IOBANK_3C |
| 6 | ✓ Ok |  | count_binary[1] | Virtual Pin | On | Yes | counter | IOBANK_3C |
| 7 | ✓ Ok |  | count_binary[0] | Virtual Pin | On | Yes | counter | IOBANK_3C |
| 8 | ✓ Ok |  | seven_seg[6] | Virtual Pin | On | Yes | counter | IOBANK_3C |
| 9 | ✓ Ok |  | seven_seg[5] | Virtual Pin | On | Yes | counter | IOBANK_3C |
| 10 | ✓ Ok |  | seven_seg[4] | Virtual Pin | On | Yes | counter | IOBANK_3C |
| 11 | ✓ Ok |  | seven_seg[3] | Virtual Pin | On | Yes | counter | IOBANK_3C |
| 12 | ✓ Ok |  | seven_seg[2] | Virtual Pin | On | Yes | counter | IOBANK_3C |

**Related Information**

Defining Virtual Pins

# 2. Signal Tap Tutorial Walkthrough

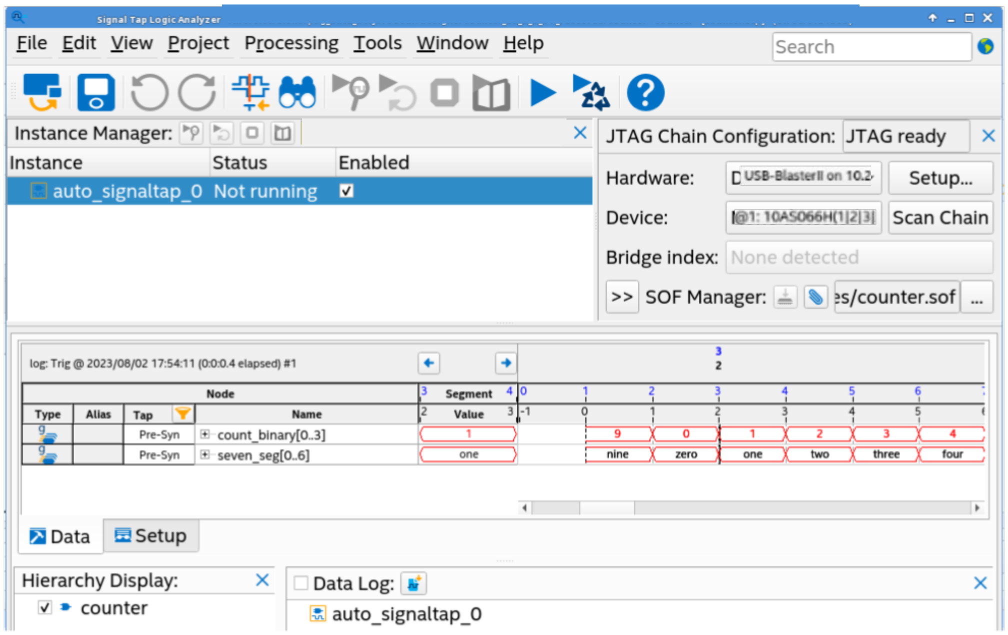
The following steps describe how to program the tutorial design and probe and debug the behavior of internal signals during normal device operation using the Signal Tap logic analyzer.

## 2.1. Step 1: Getting Started

Follow these steps to copy the tutorial reference design files to your working environment, compile the design, and open the Signal Tap logic analyzer:

1. Obtain the Intel Arria 10 FPGA - Signal Tap Logic Analyzer Getting Started Design Example from the Intel FPGA Design Store. Download the `.qar` file for the tutorial design.
2. In the Intel Quartus Prime software, click **File ➤ Open Project** to open the `counter_stp.qar` project archive file.
3. To compile the design, click **Processing ➤ Start Compilation**.
4. When compilation is complete, close the Timing Analyzer window that displays by default.

**Figure 2. Signal Tap Logic Analyzer Window**


Signal Tap Logic Analyzer
File Edit View Project Processing Tools Window Help
Search
Data
Setup

Instance Manager:

| Instance | Status | Enabled |
|---|---|---|
| auto_signaltap_0 | Not running | ✓ |

JTAG Chain Configuration: JTAG ready

Hardware: USB-BlasterII on 10.2

Device: @1: 10AS066H(1|2|3|

Bridge index: None detected

SOF Manager: es/counter.sof

log: Trig @ 2023/08/02 17:54:11 (0:0:0.4 elapsed) #1

| Type | Alias | Tap | Name | Value |
|---|---|---|---|---|
| | | Pre-Syn | count_binary[0..3] | 1 |
| | | Pre-Syn | seven_seg[0..6] | one |

Hierarchy Display: ✓ counter

Data Log: auto_signaltap_0

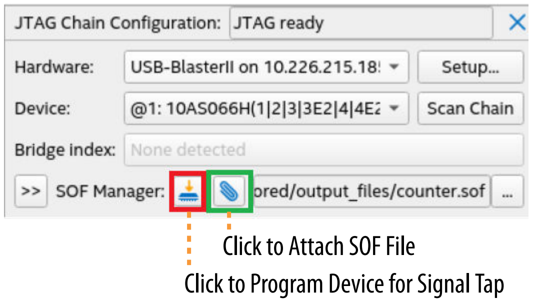
5. To open the Signal Tap logic analyzer, click **Tools ➤ Signal Tap Logic Analyzer**.

## 2.2. Step 2: Programming the FPGA for Signal Tap Use

Before programming the FPGA for use with Signal Tap, you must first create the connection between the Signal Tap logic analyzer and the Intel Arria 10 SX SoC Development Kit board via the Intel FPGA Download Cable II (USB-Blaster II), as the following steps describe:

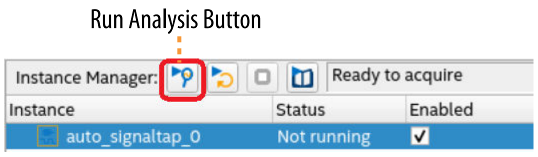
1. Connect the power supply to the Intel Arria 10 SX SoC Development Kit board.
2. Connect the USB Blaster cable between your PC USB port and the USB Blaster port on the development board.
3. In the Signal Tap Logic Analyzer window, under **JTAG Chain Configuration**, specify the following JTAG settings for this tutorial:
   - **Hardware—USB-BlasterII**
   - **Device—10AS066H**
   - **SOF Manager**—<*project*>`/output_files/counter.sof`

**Figure 3. JTAG Chain Configuration in Signal Tap**


JTAG Chain Configuration: JTAG ready
Hardware: USB-BlasterII on 10.226.215.18!
Device: @1: 10AS066H(1|2|3|3E2|4|4E2
Bridge index: None detected
SOF Manager: ored/output_files/counter.sof

4. To program the board for Signal Tap use, click the **Program Device** button.
5. When programming is complete, Signal Tap displays the 'Ready to acquire' status under **Instance Manager**.
6. To run the Signal Tap analysis, click the **Run Analysis** button.

**Figure 4. Run Analysis Button**



Ready to acquire

| Instance | Status | Enabled |
|---|---|---|
| auto_signaltap_0 | Not running | ✓ |

7. After analysis complete, the data acquisition appears in the **Data** tab.

   The trigger for the data acquisition was set to 9 for `count_binary[0..3]`. Data acquisition begins or is triggered when the value of `count_binary[0..3]` is 9, and `seven_seg[0..6]` vector starts displaying its corresponding value.

### Related Information

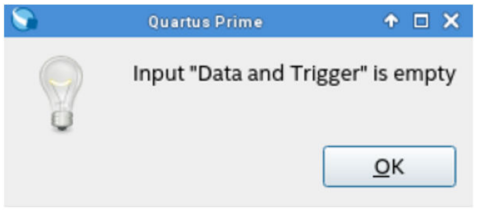
Intel Quartus Prime Pro Edition User Guide: Programmer

## 2.3. Step 3: Create a New Signal Tap File

Follow these steps to create a new Signal Tap file (`.stp`) that appropriately defines the Signal Tap settings for this tutorial exercise:

1. In the Signal Tap Logic Analyzer window, click **File ➤ New**.
2. In the **New File from Template** dialog box, under **Quick Start Group**, select **Default (default selection)**, and then click the **Create** button.
3. To save the Signal Tap file, click **File ➤ Save As**, and save the file as `stp_exercise.stp`. Click **OK** when the following warning appears:

**Figure 5. Empty Input Warning**



Input "Data and Trigger" is empty

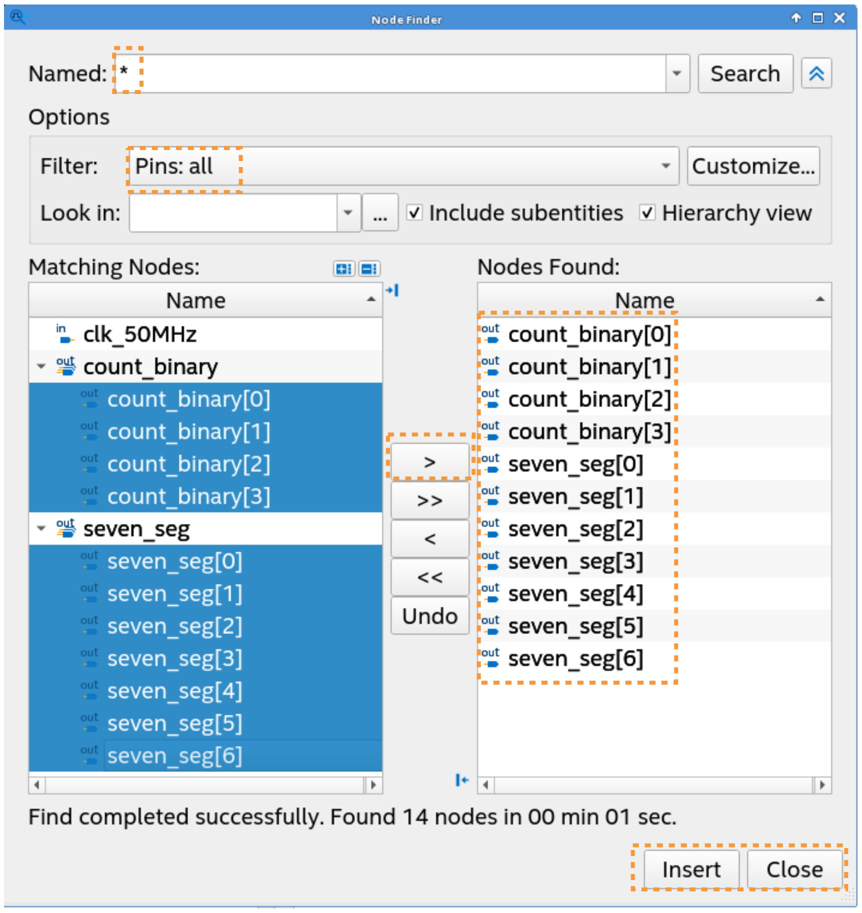
4. Click **Yes** when prompted to add the file to the current project.
5. In the Signal Tap **Setup** tab, double-click anywhere to open the Node Finder.
6. In the Node Finder, specify the following search parameters:

**Table 2. Node Finder Search Parameter Values**

| Search Parameter | Value |
|---|---|
| **Named** | * |
| **Filter** | **Pins: All** |

7. Click the **Search** button. The search results that match the search parameters appear in the **Matching Nodes** list.
8. In the matching nodes list, select the **count_binary [3:0]** and **seven_seg [6:0]** signals, as the following shows:

**Figure 6. Selected count_binary [3:0] and seven_seg [6:0] Signals**



Find completed successfully. Found 14 nodes in 00 min 01 sec.

9. Click the **Copy** (>) button to copy the **count_binary [3:0]** and **seven_seg [6:0]** signals to the **Nodes Found** list.
10. Click the **Insert** button to insert the **Nodes Found** into the **Data** and **Setup** tabs.
11. Click the **Close** button to close the Node Finder. The signals now appear on the **Data** and **Setup** tabs of the Signal Tap GUI.

## 2.4. Step 4: Add the Acquisition Clock

Follow these steps to specify the reference clock that Signal Tap uses during signal acquisition:

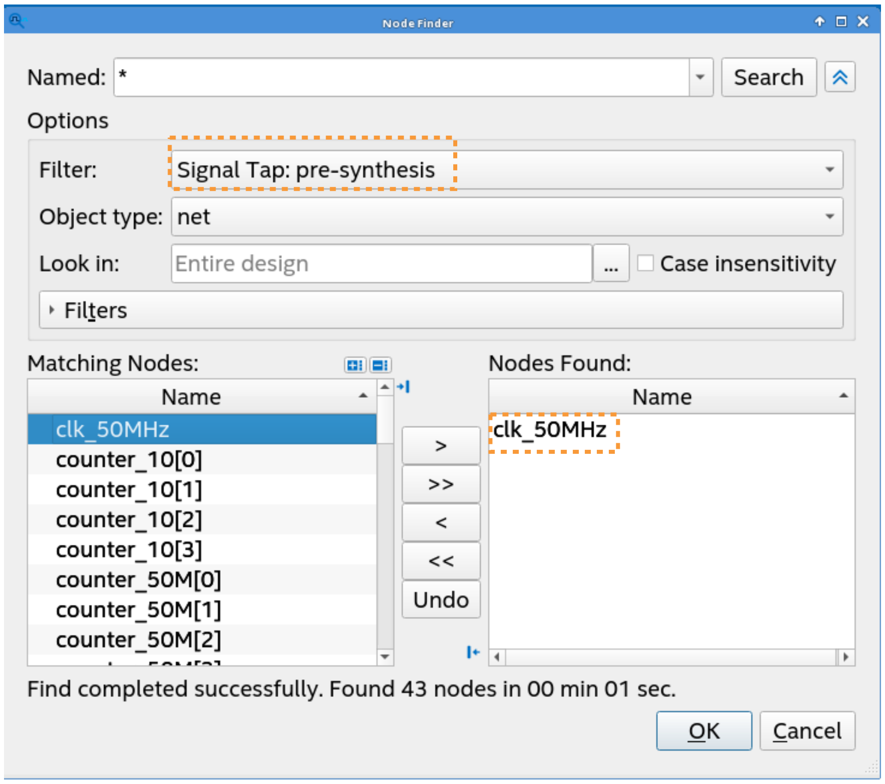
1. In the Signal Tap **Signal Configuration** pane, click (**...**) to open the Node Finder.
2. In the Node Finder, specify the following search parameters:

**Table 3. Node Finder Search Parameter Values**

| Search Parameter | Value |
|---|---|
| **Named** | * |
| **Filter** | **Signal Tap: pre-synthesis** |

3. Click the **Search** button.
4. In the **Matching Nodes** list, select the **clk_50MHz** signal, as the following shows:

**Figure 7. Selected clk_50MHz Signal**



Named: *

Options

Filter: Signal Tap: pre-synthesis

Object type: net

Look in: Entire design

Case insensitivity

Filters

Matching Nodes:

| Name |
|---|
| clk_50MHz |
| counter_10[0] |
| counter_10[1] |
| counter_10[2] |
| counter_10[3] |
| counter_50M[0] |
| counter_50M[1] |
| counter_50M[2] |

Nodes Found:

| Name |
|---|
| clk_50MHz |

Find completed successfully. Found 43 nodes in 00 min 01 sec.

5. Click the **Copy** (>) button to copy the **clk_50MHz** signal to the **Nodes Found** list.
6. Click the **Insert** button. This clock is now the Signal Tap acquisition clock.
7. Click the **Close** button.

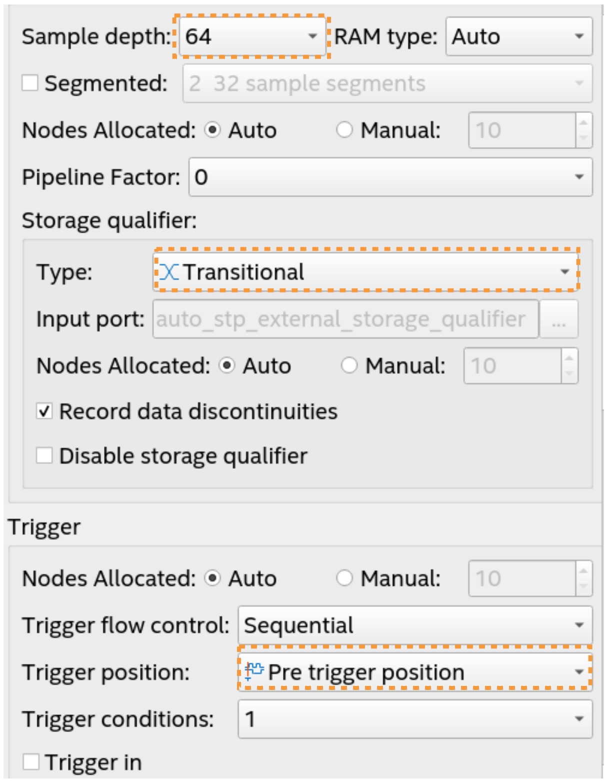
## 2.5. Step 5: Add Storage Parameters

Storage parameters define the number of samples the Signal Tap logic analyzer captures and stores, how Signal Tap organizes the sample, and the location of the sample with respect to the trigger activation.

In the Signal Tap **Signal Configuration** pane, specify the following storage parameters for this example:

Table 4. Storage Parameter Values

| Storage Parameter | Value |
|---|---|
| **Sample depth** | **64** |
| **Storage qualifier** | **Transitional** |
| **Trigger position** | **Pre trigger position** |



Sample depth: 64
RAM type: Auto
☐ Segmented: 2 32 sample segments
Nodes Allocated: ● Auto ○ Manual: 10
Pipeline Factor: 0
Storage qualifier:
Type: Transitional
Input port: auto_stp_external_storage_qualifier
Nodes Allocated: ● Auto ○ Manual: 10
☑ Record data discontinuities
☐ Disable storage qualifier
Trigger
Nodes Allocated: ● Auto ○ Manual: 10
Trigger flow control: Sequential
Trigger position: Pre trigger position
Trigger conditions: 1
☐ Trigger in

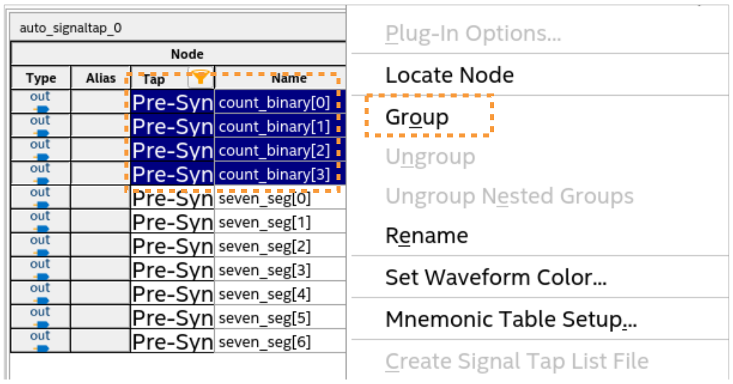
Figure 8. Signal Configuration Parameters

## 2.6. Step 6: Set Trigger Conditions

Follow these steps to set the trigger conditions for Signal Tap capture:

1. In the Signal Tap **Setup** tab, select the **count_binary[0]** to **count_binary[3]** signals, right-click and then click **Group**. This action groups the signals into a bus.

**Figure 9. Grouping count_binary Bus**

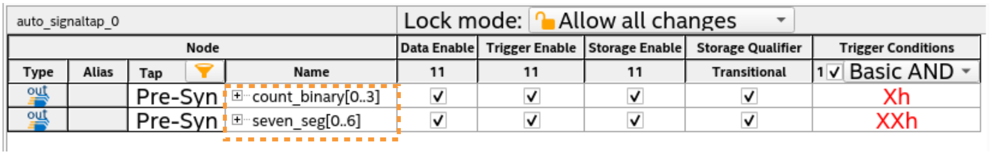


auto_signaltap_0

| Type | Alias | Tap | Name |
|---|---|---|---|
| out | | Pre-Syn | count_binary[0] |
| out | | Pre-Syn | count_binary[1] |
| out | | Pre-Syn | count_binary[2] |
| out | | Pre-Syn | count_binary[3] |
| out | | Pre-Syn | seven_seg[0] |
| out | | Pre-Syn | seven_seg[1] |
| out | | Pre-Syn | seven_seg[2] |
| out | | Pre-Syn | seven_seg[3] |
| out | | Pre-Syn | seven_seg[4] |
| out | | Pre-Syn | seven_seg[5] |
| out | | Pre-Syn | seven_seg[6] |

2. Repeat step 1 to group the signals for **seven_seg[0]** to **seven_seg[6]**.

**Figure 10. Grouping Results**

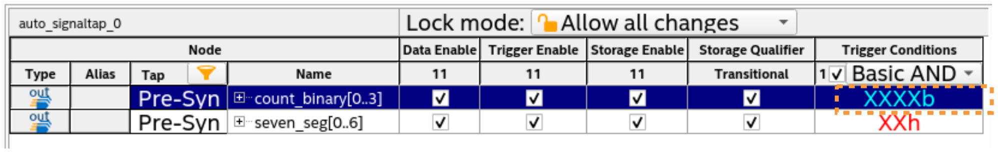


auto_signaltap_0 Lock mode: Allow all changes

| Node | | | | Data Enable | Trigger Enable | Storage Enable | Storage Qualifier | Trigger Conditions |
|---|---|---|---|---|---|---|---|---|
| Type | Alias | Tap | Name | 11 | 11 | 11 | Transitional | 1 ✓ Basic AND |
| out | | Pre-Syn | ⊞ count_binary[0..3] | ✓ | ✓ | ✓ | ✓ | Xh |
| out | | Pre-Syn | ⊞ seven_seg[0..6] | ✓ | ✓ | ✓ | ✓ | XXh |

3. To view the data for **count_binary[0..3]** in binary format, right-click **count_binary[0..3]**, and then click **Bus Display Format ➤ Binary**.

**Figure 11. Selecting Binary Display Format**

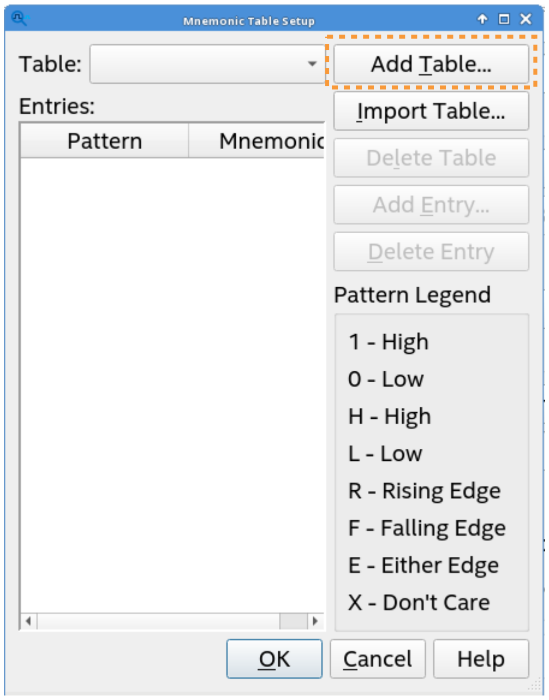


auto_signaltap_0 Lock mode: Allow all changes

| Node | | | | Data Enable | Trigger Enable | Storage Enable | Storage Qualifier | Trigger Conditions |
|---|---|---|---|---|---|---|---|---|
| Type | Alias | Tap | Name | 11 | 11 | 11 | Transitional | 1 ✓ Basic AND |
| out | | Pre-Syn | ⊞ count_binary[0..3] | ✓ | ✓ | ✓ | ✓ | XXXXb |
| out | | Pre-Syn | ⊞ seven_seg[0..6] | ✓ | ✓ | ✓ | ✓ | XXh |

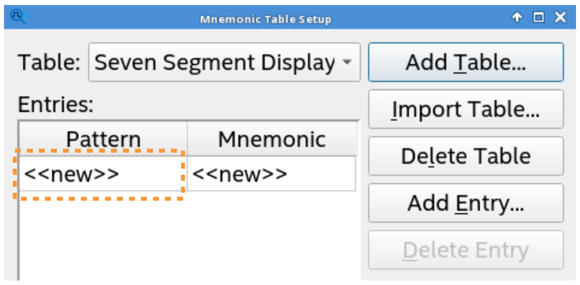
4. To better represent and easily read signals for **seven_seg[0..6]**, right-click **seven_seg[0..6]**, then click **Mnemonic Table Setup**. The **Mnemonic Table Setup** dialog box appears.

**Figure 12. Mnemonic Table Setup**



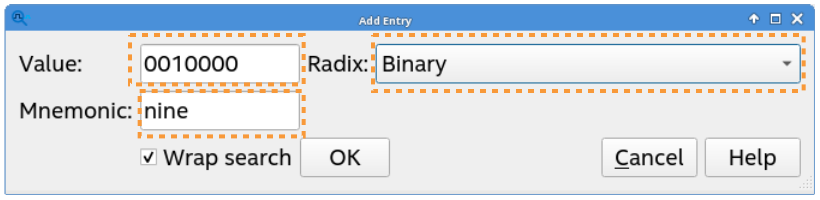
5. Click the **Add Table** button, specify `Seven Segment Display` for the **Name** and a **Width** of 7. Click **OK**.
6. Double-click **<<new>>** in the **Pattern** cell to add the mnemonic pattern.

**Figure 13. New Cell in Mnemonic Table Setup**



7. For example, to add number nine as a seven-segment mnemonic pattern, click **Add Entry**, and then specify the following values in the **Add Entry** dialog box before clicking **OK**:
   - **Value**—`010000`
   - **Radix**—**Binary**
   - **Mnemonic**—`nine`
   - **Wrap search**—Enable

**Figure 14. Mnemonic Table Setup**



Value: 0010000
Radix: Binary
Mnemonic: nine
✓ Wrap search

8. Continue to fill the **Pattern** based on Value Representation for count_binary and seven_seg Registers, as Complete Pattern in Mnemonic Table Setup shows. Click **OK** when complete.

**Figure 15. Complete Pattern in Mnemonic Table Setup**

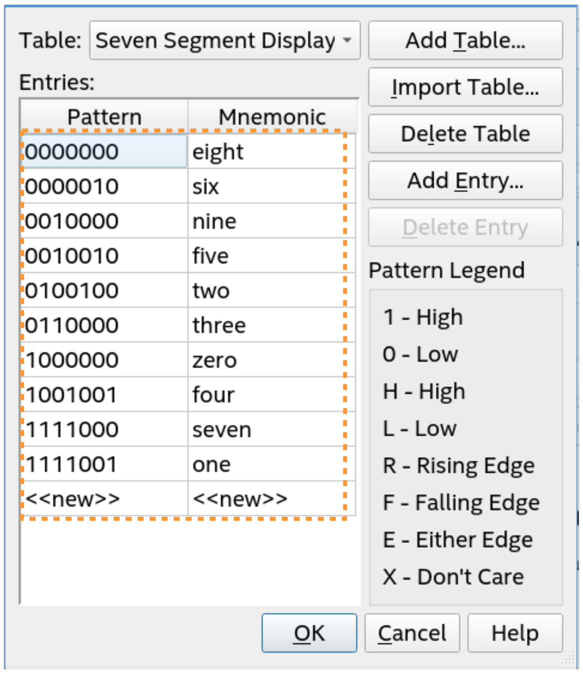


Table: Seven Segment Display

Entries:

| Pattern | Mnemonic |
|---|---|
| 0000000 | eight |
| 0000010 | six |
| 0010000 | nine |
| 0010010 | five |
| 0100100 | two |
| 0110000 | three |
| 1000000 | zero |
| 1001001 | four |
| 1111000 | seven |
| 1111001 | one |
| <<new>> | <<new>> |

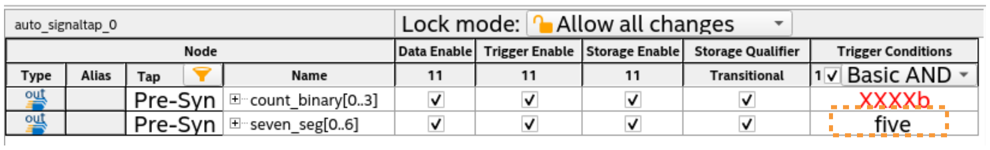
Pattern Legend

| | |
|---|---|
| 1 | High |
| 0 | Low |
| H | High |
| L | Low |
| R | Rising Edge |
| F | Falling Edge |
| E | Either Edge |
| X | Don't Care |

9. To set the display format for `seven_seg[0..6]`, right-click **seven_seg[0..6]** in **Trigger Conditions**, and then click **Bus Display Format ➤ Seven Segment Display : width = 7**.

10. To set **seven_seg[0..6]** as the current trigger conditions, double-click the **seven_seg[0..6]** cell, and then type `five` in the **Trigger Conditions** column.

**Figure 16. Set Current Trigger Conditions**

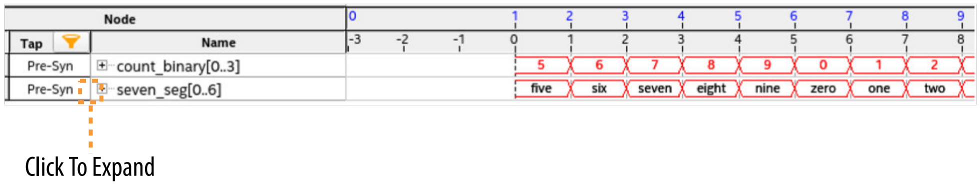


auto_signaltap_0 | Lock mode: Allow all changes

| Node | | | | Data Enable | Trigger Enable | Storage Enable | Storage Qualifier | Trigger Conditions |
|---|---|---|---|---|---|---|---|---|
| Type | Alias | Tap | Name | 11 | 11 | 11 | Transitional | 1 ✓ Basic AND |
| out | | Pre-Syn | ⊞ count_binary[0..3] | ✓ | ✓ | ✓ | ✓ | XXXXb |
| out | | Pre-Syn | ⊞ seven_seg[0..6] | ✓ | ✓ | ✓ | ✓ | five |

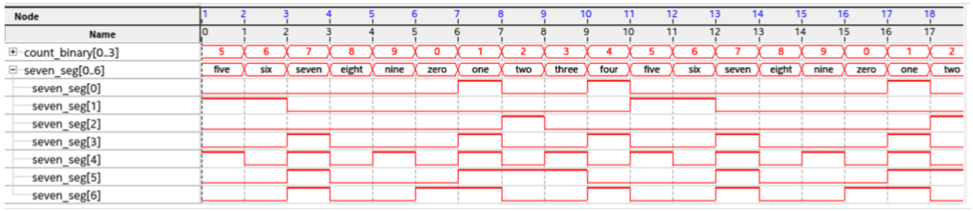
11. To compile the design that includes this Signal Tap instance, click **Processing ➤ Start Compilation** in the Intel Quartus Prime software. Close the Timing Analyzer that automatically appears following successful compilation.
12. In the Signal Tap logic analyzer, program the board and click **Run analysis**. The following figure shows the expected result with `seven_seg[0..6] == five` as the trigger.

**Figure 17. Expected Results**



13. To expand the bus group and view the signal for each sample, click the + icon

**Figure 18. Expanded Group**



# 3. Document Revision History of AN 1005: Signal Tap Logic Analyzer Getting Started Tutorial

| Document Version | Intel Quartus Prime Version | Changes |
|---|---|---|
| 2024.02.15 | 23.4 | First version of document. |