
Snapshot User Guide (UG016)



Copyrights, Trademarks and Disclaimers

Copyright © 2018 Achronix Semiconductor Corporation. All rights reserved. Achronix, Speedcore, Speedster, and ACE are trademarks of Achronix Semiconductor Corporation in the U.S. and/or other countries. All other trademarks are the property of their respective owners. All specifications subject to change without notice.

NOTICE of DISCLAIMER: The information given in this document is believed to be accurate and reliable. However, Achronix Semiconductor Corporation does not give any representations or warranties as to the completeness or accuracy of such information and shall have no liability for the use of the information contained herein. Achronix Semiconductor Corporation reserves the right to make changes to this document and the information contained herein at any time and without notice. All Achronix trademarks, registered trademarks, disclaimers and patents are listed at <http://www.achronix.com/legal>.

Achronix Semiconductor Corporation

2903 Bunker Hill Lane
Santa Clara, CA 95054
USA

Website: www.achronix.com
E-mail : info@achronix.com

Table of Contents

Chapter - 1: Overview	6
Chapter - 2: Snapshot General Description	7
Features	7
Triggers	7
Trigger Examples	9
Names.snapshot	10
Chapter - 3: Snapshot Interface	11
Snapshot Macros	11
JTAG Pins	11
Snapshot User Port List	12
Snapshot Parameter List	14
Startup Trigger Parameters	15
Parameter Impact on Core Logic Utilization	16
Snapshot Verilog Interface	17
Snapshot VHDL Interface	18
Chapter - 4: Snapshot Example	20
Clock Constraints (SDC File)	21
Example Verilog RTL	21
Chapter - 5: Probing in a Hierarchical Design	25
Overview	25
Module Declarations	25
Example	26
Chapter - 6: Running the Snapshot User Interface	31
Accessing the Snapshot Debugger	32
Open the ACE GUI and Select Your Project	32
Open the Snapshot Debugger	33
Configuring the Trigger Pattern	33
Configuring the Trigger Mode	34
Configuring Trigger Patterns	34

- Configuring the Monitor Signals 38
 - Naming captured signal data 38
- Configuring the Test Stimuli 39
 - Setting Stimuli Values Using the Table 39
 - Setting Multiple Stimuli Values as a Bus 40
- Configuring Advanced Options 41
 - Pre-Store 41
 - Trigger Pattern Match Behavior 42
 - User Clock Frequency 42
 - Configure output file locations 42
- Collecting Samples of the User Design 43
 - Using the Startup Trigger 43
 - Arming the Snapshot Debugger 43
- Saving/Loading Snapshot Configurations 44
- Running Snapshot in Batch Mode 45
- Revision History 48

Chapter - 1: Overview

Snapshot is the real-time design debugging tool for Achronix FPGAs and cores. The Snapshot debugger, which is embedded in the ACE software, delivers a practical platform to observe the signals of a user's design in real-time. To use the Snapshot debugger, the Snapshot macro needs to be instantiated inside the user's RTL. After instantiating the macro and programming the device, the user will be able to debug the design through the Snapshot Debugger GUI within ACE, or via the `run_snapshot` TCL command API.

The Snapshot macro can be connected to any logic signal mapped to the Achronix core, to monitor and potentially trigger on that signal. Monitored signal data is collected in real time in regular BRAMs, prior to being transferred to the ACE Snapshot GUI. The Snapshot macro has configurable monitor width and depth, as well as other configuration parameters, to allow user control over resource usage. The ACE Snapshot GUI interacts with the hardware via the JTAG interface: interactively specified trigger conditions are transferred to the design, and collected monitor data is transferred back to the GUI, which displays the data using a builtin waveform viewer.

The figure below shows the components involved in a Snapshot debug session.

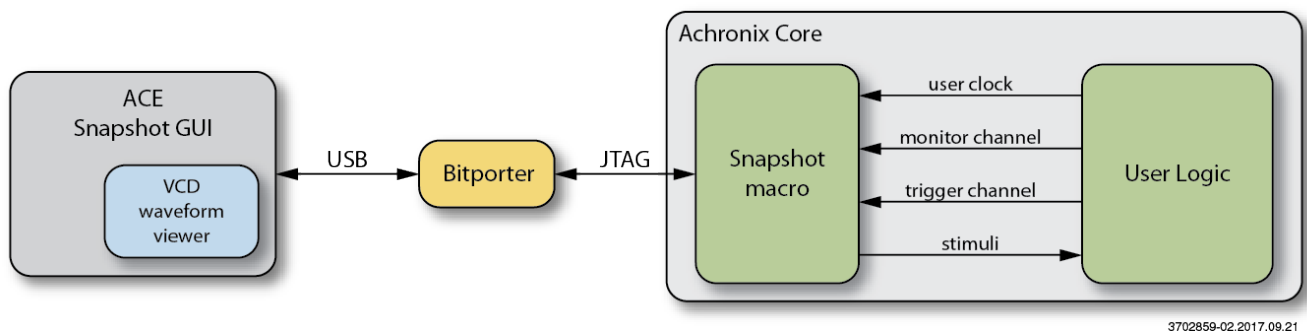


Figure 1: Snapshot Overview

Chapter - 2: Snapshot General Description

Features

The Snapshot macro samples user signals in real time, storing the captured data in one or more BRAMs. The captured data is then communicated through the JTAG interface to the ACE Snapshot GUI.

The implementation supports the following features:

- Monitor channel capture width of 1 to 4064 bits of data.
- Monitor channel capture depth of 512 to 16384 samples of data at the user clock frequency.
- Trigger channel width of 1 to 40 bits.
- Supports up to three separate sequential trigger conditions. Each trigger condition allows for the selection of a subset of the trigger channel, with AND or OR functionality.
- Bit-wise support for edge- (rise/fall) or level-sensitive triggers.
- The ACE Snapshot GUI allows specification of trigger conditions and circuit stimuli at runtime.
- An optional initial trigger condition, specified in RTL parameters, to allow capture of data immediately after startup, before interaction with the ACE Snapshot GUI.
- A stimuli interface, 0 to 512 bits wide, that allows the user to drive values into the Achronix core logic from Snapshot. Stimuli values are specified with the ACE Snapshot GUI and made available before data capture.
- Optionally, the data capture can include values before the trigger occurred. This "pre-store" amount can be specified in increments of 25% of the depth.
- Captured data is saved in a standard VCD waveform file. The ACE Snapshot GUI includes a waveform viewer for immediate feedback.
- The VCD waveform file includes a timestamp for when the Snapshot was taken.
- ACE automatically extracts the names of the monitored signals from the netlist, for easy interpretation of the waveform.
- A repetitive trigger mode, in which repeated Snapshots are taken and collected in the same VCD file.
- The JTAG interface can be shared with the user design.
- A TCL batch/script mode interface is provided via the `run_snapshot` TCL command

Triggers

The Snapshot macro has a trigger channel input with a width from 1 to 40 bits. Any subset of these inputs can be used to trigger a Snapshot. While the set of potential trigger bits is determined at design time, the choice of actual trigger condition is made at runtime using the ACE Snapshot GUI. All monitor and trigger inputs are sampled at the rising edge of `user_clk`. Trigger conditions are evaluated based on these sampled values.

A *trigger condition* specifies one of the following for each of the trigger input bits:

- don't-care ("X") – the value of the bit is ignored
- 0 – the bit matches if the input is 0
- 1 – the bit matches if the input is 1

- rising edge ("R") – the bit matches when it changes from 0 to 1 in consecutive samples
- falling edge ("F") – the bit matches when it changes from 1 to 0 in consecutive samples

Each bit is evaluated independently to determine whether it is a match or not. The results are then either ANDed (all bits, except don't-cares, must match at the same time) or ORed (the trigger matches if any bit matches).

A simple state diagram for Snapshot is shown below. The arm action is initiated from the ACE Snapshot GUI (after specifying the trigger conditions). When armed, Snapshot waits for the trigger condition to become true. Once triggered, monitor data is collected until the internal buffer is filled. The trigger point is always part of the Snapshot waveform, but if requested, a certain amount of pre-store data preceding the trigger point is collected as well. This storage is useful for seeing the events leading up to the trigger occurrence.

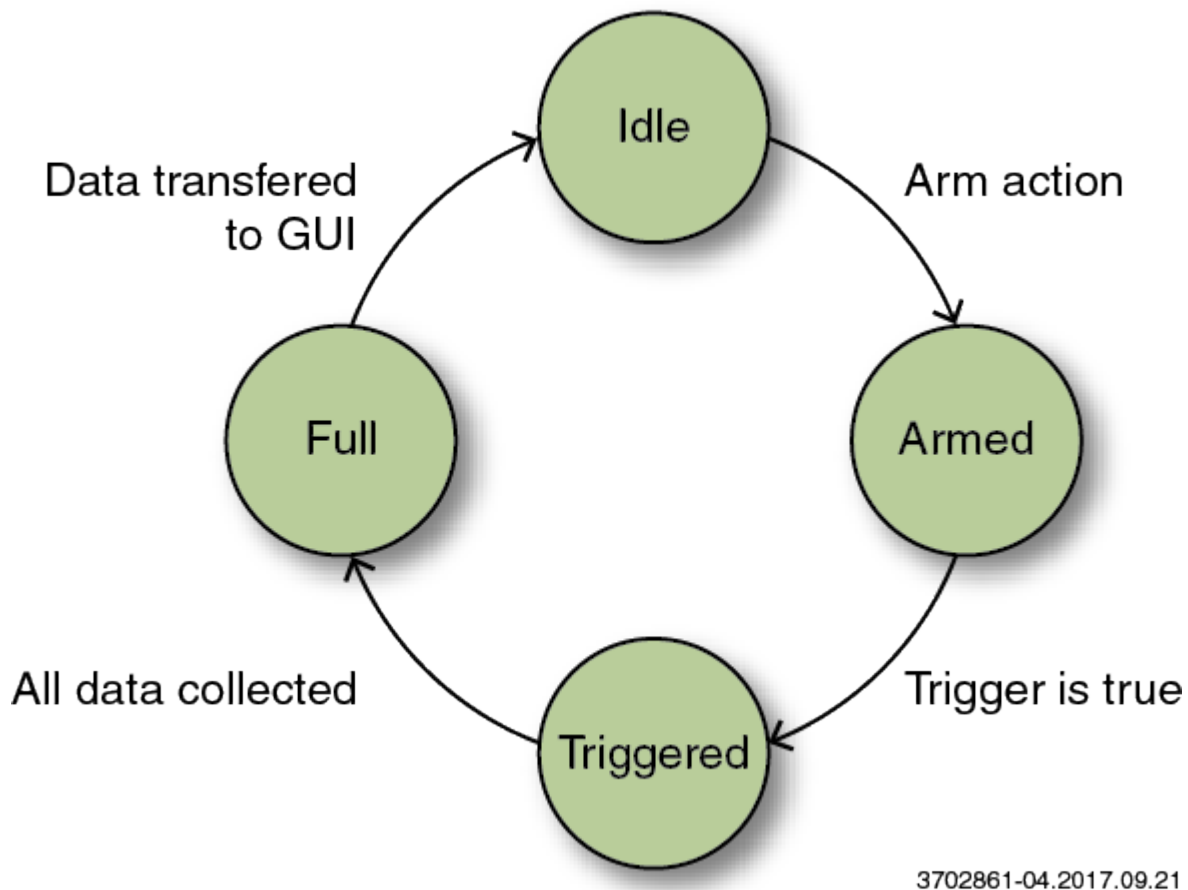
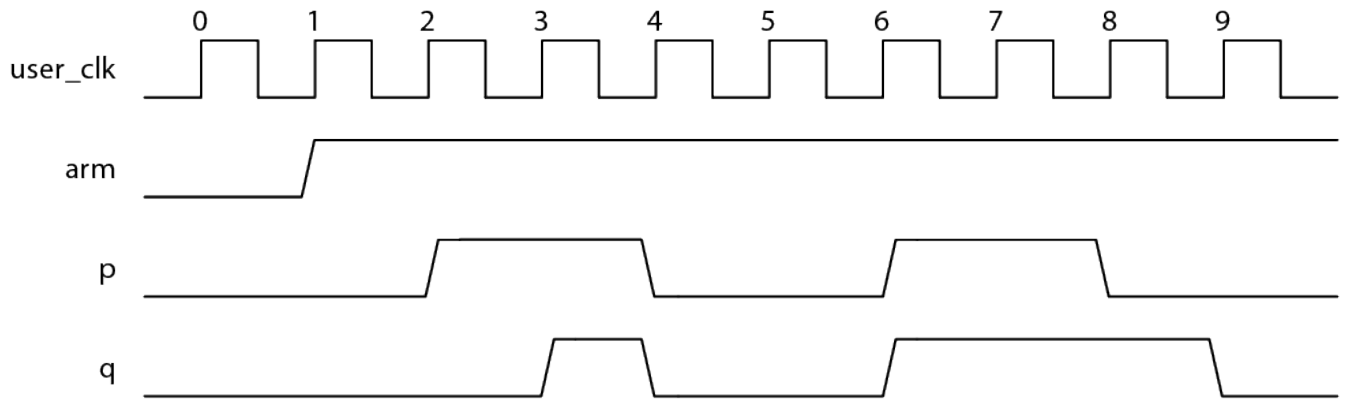


Figure 2: State Transitions of Snapshot Macro

Up to three sequential trigger conditions can be specified. Snapshot waits until the first trigger condition evaluates to true. Once that occurs, it waits for the second condition, etc. The earliest time at which the second trigger can be detected is the clock cycle following the occurrence of the first trigger. The occurrence of the last condition is the Snapshot trigger point, at which the state changes to "triggered". The final trigger point is always part of the Snapshot waveform, but whether the earlier triggers are part of the waveform depends on the pre-store amount.

Trigger Examples



3702861-05.2017.09.22

Figure 3: Trigger Example Waveform

The above waveform shows two user signals, p and q. The table below gives several examples of trigger conditions, with the time of the corresponding trigger point. Unless otherwise specified, assume only one trigger condition is specified, and all unmentioned trigger signals are "X". Snapshot is armed at time t = 1.

Table 1: Trigger Examples

Trigger Condition	Trigger Point	Explanation
p=X and q=X	1	The trigger condition with all signals X (don't-care) is always true. This condition is equivalent to "immediate mode" in the Snapshot GUI
p=0 and q=0	1	
p=1 and q=1	3	Note that the trigger point is the time at which the condition becomes true, not the time at which a flop might sample the condition.
p=R and q=R	6	Rising edge triggers. Although p = R occurs at t = 2 and q = R at t = 3, they only occur simultaneously at t = 6.
p=R and q=0	2	
p=R and q=1	6	
p=1 or q=1	2	
trigger1: p=1 and q=1 trigger2: p=0 and q=0	4	Trigger1 occurs at t = 3, then trigger2 occurs at t = 4. The latter is the trigger point.
trigger1: p=1 and q=1 trigger2: p=1 and q=1	6	The trigger point is not t = 3, because trigger2 must occur after trigger1, nor is t = 4 because that matches p = 0 and q = 0.

Trigger Condition	Trigger Point	Explanation
trigger1: p=1 and q=1 trigger2: p=X and q=X	4	The trigger point is not t = 3, because trigger2, while trivially true, must occur after trigger1.

Names.snapshot

The Snapshot macro connects to the user design with buses `i_monitor`, `i_trigger`, and `i_stimuli`. However, it would be cumbersome to debug a design if all signals were referred to as simply `i_monitor[0]`, `i_monitor[1]`, etc. Therefore, during the ACE "prepare" flow step, ACE analyzes the netlist to determine the user's signal names. The result is saved in a Snapshot configuration file, `names.snapshot`. The Snapshot GUI loads this configuration file automatically if there is an active project.

Because the name extraction occurs after RTL synthesis, sometimes names may have been modified by Synplify. It may help to use the `syn_preserve` or `syn_keep` synthesis attributes to prevent names from being changed. The ACE Snapshot GUI also enables editing of the signal names and has buttons to load and save configuration files.

Chapter - 3: Snapshot Interface

Snapshot Macros

There are two variants of the Snapshot macro, `ACX_SNAPSHOT` and `ACX_SNAPSHOT_UNIT`. Both variants have the same interface to the user design, but differ in the way they connect to the JTAG interface. Most designs will simply use `ACX_SNAPSHOT`. However, designs that already use the JTAG TAP Controller Functions for other reasons, should use `ACX_JTAP_UNIT` instead to allow sharing of the JTAG interface between Snapshot and the user design.

The figure below shows the relation between `ACX_SNAPSHOT` and `ACX_SNAPSHOT_UNIT`, as well as the interface ports.

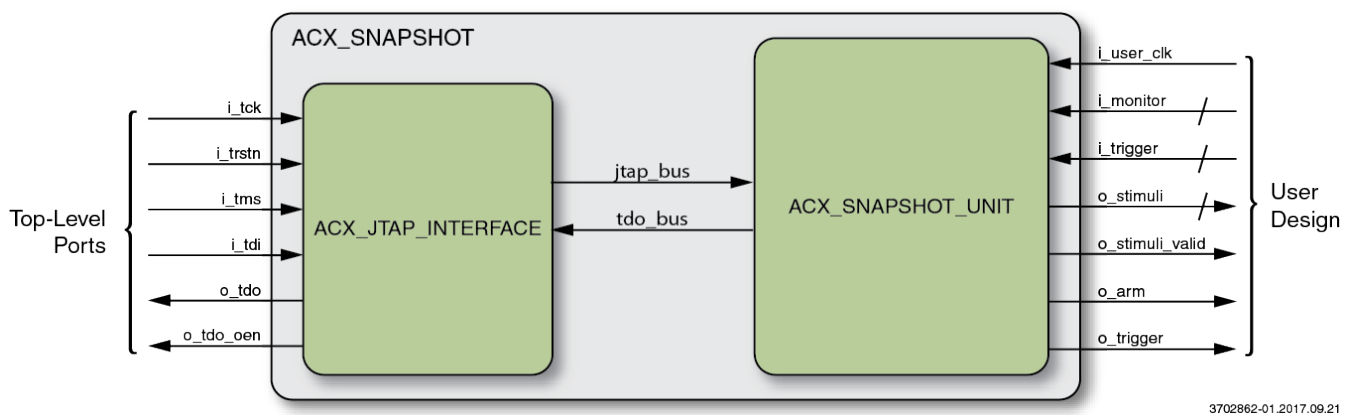


Figure 4: Snapshot Macro Block Diagram


JTAG Pins

The JTAG interface pins of `ACX_SNAPSHOT` map directly to hardware pins. In the user design, these must connect to top-level ports of the RTL *without* insertion of IPINs or OPINs (for a Speedcore instance), or pads (for Speedster FPGAs).

Table 2: JTAG Pin Description for `ACX_SNAPSHOT`

Pin Name	Type	Description
i_tck	Input	JTAG test clock
i_trstn	Input	JTAG test reset (active low)
i_tdi	Input	JTAG test data in
i_tms	Input	JTAG test mode select

Pin Name	Type	Description
o_tdo	Output	JTAG test data out
o_tdo_oen ^(†)	Output	Active-low output enable for o_tdo

Table Note
 † The signal o_tdo_oen only exists in Speedcore products to be combined with o_tdo to drive a tri-state pad. Achronix stand-alone FPGAs already include the tri-state pad as driver of o_tdo.

ACX_SNAPSHOT_UNIT has the same user interface as ACX_SNAPSHOT, but allows the sharing of the JTAG interface with the user design through the JTAG TAP controller functions. This functionality is described in more detail in the "JTAG TAP Controller Functions" of *Speedcore IP Component Library User Guide* (UG065) or the *Speedster22i Macro Cell Library* (UG021).

Table 3: JTAP Pin Description for ACX_SNAPSHOT_UNIT

Pin Name	Type	Description
i_jtap_bus	Input	Input from ACX_JTAP_INTERFACE (of type jtap_bus_tp) shared with other ACX_JTAP_UNIT instances.
i_tdo_bus	Input	Input matching the o_tdo_bus output of an ACX_JTAP_UNIT instance to allow the chaining of units. Tie low when not used.
o_tdo_bus	Output	Output to drive i_tdo_bus of ACX_JTAP_UNIT or ACX_JTAP_INTERFACE.

Snapshot User Port List

The Snapshot user side interface consists of the pins that connect directly to the user design to be monitored. This interface is identical for ACX_SNAPSHOT and ACX_SNAPSHOT_UNIT.

Table 4: Pin Descriptions of Snapshot Macro

Pin Name	Type	Description
i_monitor[MONITOR_WIDTH-1:0]	Input	1–4064 bit monitor channel. These input signals can be any signal present in the user design. They are captured when a trigger occurs and their values stored in the output VCD waveform file.

Pin Name	Type	Description
i_trigger[TRIGGER_WIDTH-1:0]	Input	1–40 bit trigger channel. These inputs can be used to trigger a capture event (the trigger condition is specified at runtime using these signals). This input is used and must be connected to the user design logic if the STANDARD_TRIGGERS parameter is set to 0. If STANDARD_TRIGGERS is 1, the input i_trigger is ignored, and the Snapshot trigger detect logic is connected internally to i_monitor [TRIGGER_WIDTH-1:0].
i_user_clk	Input	User clock (same as user design clock). All monitor and trigger inputs are sampled at the rising edge of this clock. This clock must be running for Snapshot to work, and the design must meet timing with respect to this clock.
o_stimuli[STIMULI_WIDTH-1:0] ^(†)	Output	0–512 bits of test stimuli. The user can drive the value of this bus via the Snapshot GUI when arming Snapshot. These signals can be used as test inputs to the user design. The outputs o_stimuli are only valid when o_stimuli_valid is high; at other times they may change arbitrarily.
o_stimuli_valid ^(†)	Output	Asserted high when the signals o_stimuli are valid and stable. The signal o_stimuli_valid is raised just before a Snapshot capture is started and remains high at least until all data has been captured. This signal will be de-asserted and reasserted again before the next Snapshot capture. The user design can detect the rising edge of o_stimuli_valid to determine when new input stimuli are available.
o_arm ^(†)	Output	Asserted high when Snapshot starts waiting for the trigger condition. This signal asserts at least ARM_DELAY cycles after o_stimuli_valid to give the user design time to react to the stimuli.
o_trigger ^(†)	Output	The output o_trigger is rarely used. It asserts high INPUT_PIPELINING + 5 cycles after the trigger condition occurs. This signal is provided as an optional trigger for external instruments, for example, an oscilloscope. No OUTPUT_PIPELINING is added.

Table Note

† These outputs are in the i_user_clk domain and can be used in the design-under-test (DUT) to create desired events to be observed.

Snapshot Parameter List

These parameters define the size and functionality of Snapshot.

Table 5: Parameter Definitions

Parameter	Default Value	Defined Value
DUT_NAME	"none_specified"	Field provided to the user to help distinguish Snapshot logic instances in different designs. This string is printed in the Snapshot log file whenever a Snapshot capture is taken. Maximum length is 128 characters.
MONITOR_WIDTH	40	Monitor channel width. Sets the number of signals to be monitored by Snapshot. The valid range is 1 to 4064 bits.
MONITOR_DEPTH	1024	The number of consecutive data samples (user_clk cycles) in a single Snapshot, captured from the i_monitor bus. Valid values range from 512 to 16384. The implementation rounds this number up as required by the supported BRAM sizes.
TRIGGER_WIDTH	40	Trigger channel width. The valid range is 1 to 40 bits.
NUM_TRIGGERS	3	The maximum number of sequential triggers to compile into the Snapshot circuit. Setting this parameter to a lower number decreases the Achronix core logic resources needed for Snapshot. During a Snapshot debug session, the user may configure up to NUM_TRIGGERS sequential triggers. Valid values range from 1 to 3.
STANDARD_TRIGGERS	1	If the STANDARD_TRIGGERS parameter value is set to 1, then the i_trigger input is ignored, and instead i_monitor [TRIGGER_WIDTH-1:0] is used as trigger signals. If the STANDARD_TRIGGERS parameter value is set to 0, then the i_trigger [TRIGGER_WIDTH-1:0] input is used as trigger signals.
STIMULI_WIDTH	20	Number of stimuli signals output to the user design. The valid range is 0 to 512 bits.
INPUT_PIPELINING	3	Adds the specified number of pipeline stages to the i_monitor and i_trigger signals to enable fast i_user_clk speeds. This parameter has no effect on the collected data (the .vcd file), or on the point where the trigger occurs.
OUTPUT_PIPELINING	0	Adds the specified number of pipeline stages to the o_arm, o_stimuli, and o_stimuli_valid outputs to enable fast i_user_clk speeds. This parameter has no effect on the collected data (the .vcd file), or on the point where the trigger occurs.

Parameter	Default Value	Defined Value
ARM_DELAY	1	Delay between assertion of o_stimuli_valid and o_arm. The o_arm output signal indicates when Snapshot starts waiting for the trigger condition. This signal asserts at least ARM_DELAY cycles after o_stimuli_valid to give the user design time to react to the stimuli.
ENABLE_EDGE_TRIGGERS	1	When set to 1, both edge-sensitive (rise/fall) and level-sensitive (1/0) trigger conditions may be used during a Snapshot debug session. When set to 0, only level-sensitive trigger conditions may be used. Setting to 0 decreases the Achronix core logic resources needed for Snapshot.

Startup Trigger Parameters

Normally, trigger conditions are specified via the ACE Snapshot GUI prior to taking a capture. However, that makes it hard to observe conditions that occur right after startup. As alternative, an initial trigger condition can be specified using parameters. When INITIAL_TRIGGER is set, Snapshot is armed immediately after startup and waits for the initial trigger condition. The ACE Snapshot GUI has a separate Startup Trigger button to collect the captured data.

Since initial triggers have virtually no circuit overhead, they are enabled by default with a don't-care trigger. With these defaults, the Startup Trigger button collects data from the start of user mode (or as close to the start as possible). Snapshot needs a few clock cycles to initialize before it can collect data or detect trigger conditions. For Speedcore instances, this delay is three cycles if MONITOR_DEPTH \leq 1024; otherwise it is six cycles. Signals will not be monitored during those few cycles unless INPUT_PIPELINING is used. If INPUT_PIPELINING is at least 3 (for small depth) or 6 (for larger depth), data is collected from the start of user mode.

Table 6: Snapshot Startup Trigger Parameters

Parameter	Default Value	Defined Value
INITIAL_TRIGGER	1	Enables a startup trigger condition. Set the other INITIAL_* parameters to specify the trigger condition. When INITIAL_TRIGGER is 1, Snapshot automatically arms right after startup. If INITIAL_TRIGGER is 0, the INITIAL_* parameters are ignored, and Snapshot waits in the Idle state until the user arms Snapshot via the ACE GUI or TCL interface.
INITIAL_NUM_TRIGGERS	1	Number of sequential triggers to use for the startup trigger. Valid range is 1 to NUM_TRIGGERS.
INITIAL_TRIGGER1	X's	INITIAL_TRIGGER1 is specified as a sequence of characters "0" for level 0, "1" for level 1, "R" for rising edge, "F" for falling edge, "X" for don't care, with one character per trigger bit, similar to the binary value specified for a bus in the ACE GUI. For example, if TRIGGER_WIDTH is set to 5, INITIAL_TRIGGER1 could be set to "11XR0" to define the trigger pattern.

Parameter	Default Value	Defined Value
INITIAL_TRIGGER2	X's	Specifies the second startup trigger using the same format as INITIAL_TRIGGER1. Snapshot waits for INITIAL_TRIGGER2 after INITIAL_TRIGGER1 has occurred. This parameter is ignored if INITIAL_NUM_TRIGGERS < 2.
INITIAL_TRIGGER3	X's	Specifies the third startup trigger using the same format as INITIAL_TRIGGER1. Snapshot waits for INITIAL_TRIGGER3 after INITIAL_TRIGGER2 has occurred. This parameter is ignored if INITIAL_NUM_TRIGGERS < 3.
INITIAL_USE_AND_1	1	When set to 1, the INITIAL_TRIGGER1 pattern matches the input trigger data if ALL of the trigger bits match the trigger pattern (AND logic). When set to 0, the INITIAL_TRIGGER1 pattern matches the input trigger data if ANY of the trigger bits matches the trigger pattern (OR logic). In both cases, don't-care bits (marked "X") are ignored. However, if all INITIAL_TRIGGER1 bits are "X" (don't-care), this parameter must be set to 1.
INITIAL_USE_AND_2	1	Similar to INITIAL_USE_AND_1, but for INITIAL_TRIGGER2.
INITIAL_USE_AND_3	1	Similar to INITIAL_USE_AND_1, but for INITIAL_TRIGGER3.
INITIAL_PRE_STORE	1	Amount of pre-store data to cache and output prior to the trigger event. Valid values are 0 (no pre-store), 1 (25% pre-store), 2 (50% pre-store), and 3 (75% pre-store). If the startup trigger occurs before INITIAL_PRE_STORE clock cycles have occurred, by necessity less pre-store data is collected.

Parameter Impact on Core Logic Utilization

The following parameters impact the amount of core logic resources required for the Snapshot circuit:

- MONITOR_WIDTH
- MONITOR_DEPTH
- TRIGGER_WIDTH
- NUM_TRIGGERS
- ENABLE_EDGE_TRIGGERS
- STIMULI_WIDTH
- INPUT_PIPELINING
- OUTPUT_PIPELINING

Note



The Snapshot resource utilization is really only important for designs that take up most of the FPGA, or in very small FPGA cores. If a design that includes Snapshot is pushing the limits of (or exceeding) core fabric resources, adjust the parameters above to reduce the size of the Snapshot logic.

Below is a rough estimate of the number of gates required based on these parameters:

- The number of BRAMs/BRAMFIFOs must be sufficient to store $\text{MONITOR_WIDTH} \times \text{MONITOR_DEPTH}$ bits.
- Input pipelining consumes roughly $(\text{MONITOR_WIDTH} + \text{TRIGGER_WIDTH}) \times \text{INPUT_PIPELINING}$ flip-flops.
- Output pipelining consumes roughly $\text{STIMULI_WIDTH} \times \text{OUTPUT_PIPELINING}$ flip-flops.
- The trigger circuit requires roughly $\text{NUM_TRIGGERS} \times 5 \times \text{TRIGGER_WIDTH}$ flip-flops. The number of flip-flops can be reduced by setting NUM_TRIGGERS to 1 or 2, by reducing the width, or by disabling edge triggers. Edge triggers account for roughly 40% of the trigger circuit.

Note

For high-speed circuits, input or output pipelining may be required to meet performance.

Snapshot Verilog Interface

```
// - MONITOR_DEPTH will be rounded up to the next value supported by
//   this implementation.
// - If STANDARD_TRIGGERS is 1, the i_trigger input is ignored and instead
//   i_monitor[TRIGGER_WIDTH - 1 : 0] are used as trigger signals.
// - Stimuli are valid only when o_stimuli_valid is true; at other times
//   o_stimuli are not stable.
// - o_arm indicates when Snapshot starts waiting for the trigger condition.
//   This happens at least ARM_DELAY cycles after o_stimuli_valid, to give
//   the user design time to react to the stimuli.
// - INPUT_PIPELINING is added to i_monitor and i_trigger, to make it easier
//   to collect high-frequency signals from various locations. Likewise,
//   OUTPUT_PIPELINING is added to o_stimuli, o_stimuli_valid, and o_arm.
//   Note that these parameters have *no impact* on the collected data
//   (the vcd file) or on the point where the trigger occurs.
// - To set a startup trigger condition, set INITIAL_TRIGGER to 1, then
//   set the INITIAL_ parameters to specify the trigger condition.
//   INITIAL_TRIGGER1 is a sequence of characters "0", "1", "R", "F", "X", one
//   character per bit, similar to the binary value specified for a bus
//   in the ACE GUI.
// - The o_trigger output is seldom used. It goes high INPUT_PIPELINING + 5
//   cycles after the trigger condition occurred. This signal is provided
//   as a trigger for external equipment such as a scope. No output
//   pipelining is added.
// - SNAPSHOT_MODE is used for development.

`default_nettype none
`timescale 1 ps / 1 ps
module ACX_SNAPSHOT #(
    localparam integer max_dut_name_chars = 128,
    parameter [8*max_dut_name_chars-1 : 0] DUT_NAME = "none_specified",
    parameter integer MONITOR_WIDTH = 40,    // >= 1
    parameter integer MONITOR_DEPTH = 1024, // 1024 .. 16384
    parameter integer TRIGGER_WIDTH = 40,    // 1..40
    parameter integer NUM_TRIGGERS = 3,      // 1..3
    parameter bit STANDARD_TRIGGERS = 1,    // use i_monitor instead of i_trigger

```

```

parameter integer STIMULI_WIDTH = 20, // <= 512
parameter integer INPUT_PIPELINING = 3, // for i_monitor and i_trigger
parameter integer OUTPUT_PIPELINING = 0, // for o_stimuli(_valid) and o_arm
parameter integer ARM_DELAY = 1, // between o_stimuli_valid and o_arm
parameter bit ENABLE_EDGE_TRIGGERS = 1,

parameter bit INITIAL_TRIGGER = 0, // set startup trigger condition
parameter [1:0] INITIAL_NUM_TRIGGERS = 1, // 1..NUM_TRIGGERS
parameter [8*TRIGGER_WIDTH-1 : 0] INITIAL_TRIGGER1 = {TRIGGER_WIDTH{8'h58}},
parameter [8*TRIGGER_WIDTH-1 : 0] INITIAL_TRIGGER2 = {TRIGGER_WIDTH{8'h58}},
parameter [8*TRIGGER_WIDTH-1 : 0] INITIAL_TRIGGER3 = {TRIGGER_WIDTH{8'h58}},
parameter bit INITIAL_USE_AND_1 = 1, // 1 = AND, 0 = OR
parameter bit INITIAL_USE_AND_2 = 1,
parameter bit INITIAL_USE_AND_3 = 1,
parameter [1:0] INITIAL_PRE_STORE = 1, // 0, 1, 2, 3 (= 0, 25%, 50% 75%)

parameter integer SNAPSHOT_MODE = 0
) (
    // jtag connections, must be connected to top-level ports
    input wire i_tck,
    input wire i_trstn,
    input wire i_tms,
    input wire i_tdi,
    output wire o_tdo,

    // signals to/from user design
    input wire i_user_clk,
    input wire [MONITOR_WIDTH-1 : 0] i_monitor,
    input wire [TRIGGER_WIDTH-1 : 0] i_trigger, // if !STANDARD_TRIGGERS
    output wire [STIMULI_WIDTH-1 : 0] o_stimuli,
    output wire o_stimuli_valid,
    output wire o_arm,
    output wire o_trigger // for external devices
);

```

Snapshot VHDL Interface

```

component ACX_SNAPSHOT is
    generic (
        DUT_NAME      : string      := "none_specified";
        MONITOR_WIDTH  : natural     := 40;          -- >= 1
        MONITOR_DEPTH  : natural     := 1024;        -- 1024 ... 16384
        TRIGGER_WIDTH   : natural     := 40;          -- 1 ... 40
        NUM_TRIGGERS    : natural     := 3;           -- 1, 2, 3
        STANDARD_TRIGGERS: std_logic := '1';         -- use "i_monitor" instead of
        "i_trigger"

        STIMULI_WIDTH   : natural     := 20;          -- <= 512
        INPUT_PIPELINING: natural     := 3;           -- FOR i_monitor AND i_trigger
        OUTPUT_PIPELINING: natural    := 0;           -- FOR o_stimuli(_valid) AND
        o_arm

```

```

        ARM_DELAY      : natural := 1;          -- BETWEEN o_stimuli_valid AND
o_arm
        ENABLE_EDGE_TRIGGERS : std_logic := '1';
        INITIAL_TRIGGER      : std_logic := '0'; -- SET STARTUP TRIGGER
CONDITION
        INITIAL_NUM_TRIGGERS : std_logic_vector (1 downto 0) := "01"; -- 1, 2,
3
        INITIAL_TRIGGER1      : string :=
"XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"; -- NUMBER OF CHARACTERS SHOULD BE
TRIGGER_WIDTH. VALID CHARACTERS ARE X, 0, 1, R, AND F.
        INITIAL_TRIGGER2      : string :=
"XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"; -- NUMBER OF CHARACTERS SHOULD BE
TRIGGER_WIDTH. VALID CHARACTERS ARE X, 0, 1, R, AND F.
        INITIAL_TRIGGER3      : string :=
"XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX"; -- NUMBER OF CHARACTERS SHOULD BE
TRIGGER_WIDTH. VALID CHARACTERS ARE X, 0, 1, R, AND F.
        INITIAL_USE_AND_1      : std_logic := '1';    -- 1 = AND, 0 = OR
        INITIAL_USE_AND_2      : std_logic := '1';    -- 1 = AND, 0 = OR
        INITIAL_USE_AND_3      : std_logic := '1';    -- 1 = AND, 0 = OR
        INITIAL_PRE_STORE      : std_logic_vector (1 downto 0) := "00"; -- 0, 1,
2, 3 ( = 0, 25%, 50%, 75%)
        SNAPSHOT_MODE          : natural := 0 -- reserved
    );
    port ( --- JTAG connections, must be connected to TOP-LEVEL ports ---
        i_tck      : in  std_logic;
        i_trstn     : in  std_logic;
        i_tms       : in  std_logic;
        i_tdi       : in  std_logic;
        o_tdo       : out std_logic;
        --- SIGNALS to/from USER DESIGN ---
        i_user_clk   : in  std_logic;
        i_monitor    : in  std_logic_vector (MONITOR_WIDTH-1 downto 0);
        i_trigger    : in  std_logic_vector (TRIGGER_WIDTH-1 downto 0);
        o_stimuli    : out std_logic_vector (STIMULI_WIDTH-1 downto 0);
        o_stimuli_valid : out std_logic;
        o_arm        : out std_logic;
        o_trigger    : out std_logic
    );
end component;

```

Chapter - 4: Snapshot Example

The following is a complete example of a simple user design with Snapshot. The user design consists of two counters and has the following features:

- `counter_a[7:0]` counts from 0 to `limit_a` repeatedly
- `limit_a[7:0]` can be set dynamically with the Snapshot stimuli
- `counter_b[15:0]` 16-bit counter (wraps around)
- External reset or reset via Snapshot stimuli

In order to use the Snapshot logic in a user design, the technology-specific Snapshot Verilog file from the Achronix libraries must be included:

```
`include "speedster<technology>/common/speedster<technology>_snapshot_v3.v"
```

Where `<technology>` is replaced with the target technology library name.

Two clocks are required by the Snapshot macro:

- `i_user_clk` – this clock is provided by the user design and is used to sample the user design signals.
- `i_tck` – the IEEE 1149.1 JTAG based TCK used to read the data out from the Snapshot macro through the JTAG TDO port.

Snapshot evaluates triggers and collects data at the rate of the `user_clk`, whose frequency must be declared in the SDC file.

The design must meet timing with respect to `user_clk`. Even if timing failures in the user design are deemed acceptable, their existence might hide timing failures in the Snapshot logic. Instead, "acceptable" timing failures must be made explicit with exceptions in the SDC file. If the Snapshot logic itself does not meet timing, consider increasing the `INPUT_PIPELINING` and `OUTPUT_PIPELINING` parameters.

The JTAG clock (`i_tck`) for Snapshot must be declared as a 10 MHz clock (period 100 ns). It is recommended that this frequency is also specified during synthesis, because Synplify may over-optimize this slow logic.

The example below includes `CLK_IPIN` and `IPIN` instances for a Speedcore device. For a standalone FPGA, use `IPAD` instances instead. Pad placement appropriate to the device and board must be specified. JTAG signals must not be connected to IPINs or IPADs.

The Snapshot macro should be instantiated in the user design, as shown below, and connected to the signals that may need to be observed. Next, the design is synthesized with Synplify and run through the ACE flow to generate a bitstream. Once the Achronix device has been programmed with the bitstream, the Snapshot debugger tool may be used from the ACE GUI or in batch mode via the ACE TCL interface.

Note



When the user design is run through the ACE place and route flow, a Snapshot configuration file will be generated in `<ace_project_dir>/<active_impl_dir>/output/names.snapshot`. This file contains all the signal names connected to Snapshot (automatically extracted from the user design), along with monitor, trigger, and stimuli width settings based on the user RTL, clock frequency based on the user SDC constraints, etc. This file is automatically loaded in the Snapshot Debugger View in the ACE GUI to configure Snapshot whenever the active implementation in the ACE session changes.

Clock Constraints (SDC File)

Both the JTAG TCK clock and the Snapshot user clock must be defined in the user SDC clock constraints:

```
# Snapshot JTAG clock: 10MHz
create_clock -period 100 [get_ports i_tck] -name tck
set_clock_groups -asynchronous -group {tck}

# User design clock; example: 100MHz
create_clock -period 10 [get_ports i_clk_100] -name i_clk_100
set_clock_groups -asynchronous -group {i_clk_100}
```

Example Verilog RTL

```
// Copyright (c) 2017 Achronix Semiconductor Corp.
// All Rights Reserved.
`include "speedster16t/common/speedster16t_snapshot_v3.v"

`default_nettype none
`timescale 1ps/1ps
module snapshot_counter_v3_sc (
    // jtag ports:
    input wire i_tck,
    input wire i_trstn,
    input wire i_tms,
    input wire i_tdi,
    output wire o_tdo,
    output wire o_tdo_oen,

    // user design ports:
    input wire i_clk_100,
    input wire i_rst_n
);

/***** clock *****/

wire clk;
CLK_IPIN ipin_i_clk_100(.din(i_clk_100), .dout(clk));

/***** stimuli *****/

// Snapshot stimuli are only valid when stimuli_valid is high.
wire stimuli_valid;
reg [2:0] stimuli_valid_d; // for edge detection/stretching

always @(posedge clk)
begin
    stimuli_valid_d <= (stimuli_valid_d < 1) | stimuli_valid;
end
```

```

/***** reset *****/

wire rst_n;
IPIN #(
    .mode(`DEF_IPIN_MODE_CLOCKED)
) ipin_i_reset_n (
    .din(i_rst_n),
    .clk(clk),
    .ce(1'b1),
    .rstn(1'b1),
    .dout(rst_n)
);

reg [3:0] pipe_i_rst_n; // pipeline from i_rst_n pad, for timing
wire do_reset; // set via stimuli (active-high)

// At edge detection of stimuli_valid, do_reset is a reset input (active-high).
// While do_reset stays high, we de-assert reset after 3 cycles.
reg reset_n = 0;
always @(posedge clk)
begin
    pipe_i_rst_n <= (pipe_i_rst_n << 1) | rst_n;

    if (stimuli_valid && !stimuli_valid_d[2])
        reset_n <= pipe_i_rst_n[3] && !do_reset;
    else
        reset_n <= pipe_i_rst_n[3];
end

/***** user circuit *****/

// The main user design consists of two counters.
// counter_a : 8-bit counter with configurable period. The period is set
//             by setting limit_a via the Snapshot stimuli. Default
//             limit_a = 62 (hence counter_a has default period 63).
// counter_b : 16-bit counter

reg [7:0] limit_a = 62;
reg [7:0] counter_a = 0; // counts 0..limit_a
reg [15:0] counter_b = 0;

always @(posedge clk)
begin
    if (!reset_n)
        begin
            counter_a <= 0;
            counter_b <= 0;
        end
    else
        begin
            if (counter_a == limit_a)
                counter_a <= 0;
        end
    end
end

```

```

        else
            counter_a <= counter_a + 1;
            counter_b <= counter_b + 1;
        end
    end

    wire [7:0] limit_a_in; // set via stimuli; if not 0, value for limit_a
    always @(posedge clk)
    begin
        if (stimuli_valid && limit_a_in != 0)
            limit_a <= limit_a_in;
        end

    /***** snapshot *****/

    localparam MONITOR_WIDTH = 38;
    localparam TRIGGER_WIDTH = 38;
    wire [MONITOR_WIDTH-1 : 0] monitor;
    wire arm;

    assign monitor = {
        counter_b,
        counter_a,
        limit_a,
        arm,
        stimuli_valid,
        reset_n
    };

    localparam STIMULI_WIDTH = 9;
    wire [STIMULI_WIDTH-1 : 0] stimuli;
    assign {
        do_reset,
        limit_a_in
    } = stimuli;

    ACX_SNAPSHOT #(
        .DUT_NAME("counter_v3"),
        .MONITOR_WIDTH(MONITOR_WIDTH),
        .MONITOR_DEPTH(2000), // will be rounded up
        .TRIGGER_WIDTH(TRIGGER_WIDTH),
        .NUM_TRIGGERS(3),
        .STANDARD_TRIGGERS(1), // use i_monitor as trigger input
        .STIMULI_WIDTH(STIMULI_WIDTH),
        .INPUT_PIPELINING(3),
        .OUTPUT_PIPELINING(0),
        .ARM_DELAY(2),
        .ENABLE_EDGE_TRIGGERS(1)
    ) x_snapshot (
        .i_tck(i_tck),
        .i_trstn(i_trstn),
        .i_tms(i_tms),
        .i_tdi(i_tdi),

```

```
.o_tdo(o_tdo),  
.o_tdo_oen(o_tdo_oen),  
  
.i_user_clk(clk),  
.i_monitor(monitor),  
.i_trigger(), // not used if STANDARD_TRIGGERS = 1  
.o_stimuli(stimuli),  
.o_stimuli_valid(stimuli_valid),  
.o_arm(arm),  
.o_trigger()  
);  
  
endmodule
```


Chapter - 5: Probing in a Hierarchical Design

Overview

Snapshot provides the ability to probe signals deep within a hierarchical design without the need to modify every level of RTL, i.e., pulling the signals through the hierarchy up to the top level.

A special macro allows the user to define which signals are to be probed within the deeply embedded module. These probe points are then matched at the top-level module where Snapshot is instantiated. Synplify or ACE (depending on usage) aligns the deeply embedded and top-level signals, providing access to the embedded signals without having to explicitly bring them to the top level through multiple levels of RTL.

This method uses modules ACX_PROBE_CONNECT and ACX_PROBE_POINT. There are two options for using them:

- Use a user-defined tag to associate an ACX_PROBE_POINT with an ACX_PROBE_CONNECT
- Use a hierarchical pin name (with wildcards) to associate pins with an ACX_PROBE_CONNECT

Then monitor the ACX_PROBE_CONNECT output with Snapshot.

Generally, the method with tags is preferred, because it is often hard to determine the full hierarchical name of a pin. The method with a pin name is useful for tapping a signal from a macro that cannot be edited (for example, probing inside the library macros or third-party IP).

Note



The first method, with tags, uses Synplify `syn_hyper_source` and `syn_hyper_connect` instances; error messages may refer to those terms.

Module Declarations

```
module ACX_PROBE_POINT #(
    parameter integer width = 1, // set to input width
    parameter tag = ""           // set to unique string
) (
    input [width-1:0] din
);
endmodule
```

An ACX_PROBE_POINT takes as input one or more signals that need to be observed with Snapshot. It is instantiated in the hierarchy at the point where these signals are available.

```
module ACX_PROBE_CONNECT #(
    parameter integer width = 1, // must match width of source
    parameter tag = "",         // must match tag of source
    parameter pin = "",         // "instance:pin" or "instance:bus", wildcards allowed
    parameter must_connect = 1'b1 // whether missing source is error or warning
) (
    output [width-1:0] dout
```

```
);
endmodule
```

The output of an ACX_PROBE_CONNECT instance are then monitored with Snapshot. This instance can be created in the same module as the ACX_SNAPSHOT instance. The software uses the tag string to find a matching ACX_PROBE_POINT, then replaces both modules with a direct connection between the input of ACX_PROBE_POINT and the output of ACX_PROBE_CONNECT.

Alternatively, for cases where it is not possible to insert an ACX_PROBE_POINT, ACX_PROBE_CONNECT can be used with a hierarchical pin name instead of a tag. The ACE `find` command may be useful when trying to determine hierarchical names.

Example

This code below is the same as the design shown in [Snapshot Example \(see page 20\)](#), but places the user design (the two counters) inside a separate module, `counters`. One can easily imagine that this module is designed to compute some function (`all_zero` in this example) without necessarily exposing the counters themselves. But during debugging, counter values need to be observed to verify correctness. Rather than adding ports to expose the counters, maybe for many levels of hierarchy, probe points can be used instead.

As mentioned, using probe points with tags is preferred, but for the sake of the example, a probe point was only placed on `counter_a`; `counter_b` is identified by pin name.

Nested module with local counters

```
`default_nettype none
`timescale 1ps/1ps
module counters (
    input clk,
    input reset_n,
    input [7:0] limit_a,
    output all_zero
);

/***** user circuit *****/

// The main user design consists of two counters.
// counter_a : 8-bit counter with configurable period.
// counter_b : 16-bit counter

reg [7:0] counter_a = 0; // counts 0..limit_a
reg [15:0] counter_b = 0;

always @(posedge clk)
begin
    if (!reset_n)
        begin
            counter_a <= 0;
            counter_b <= 0;
        end
    else
        begin
            if (counter_a == limit_a)
                counter_a <= 0;
        end
    end
end
```

```

        else
            counter_a <= counter_a + 1;
            counter_b <= counter_b + 1;
        end
    end

    assign all_zero = (counter_a == 0 && counter_b == 0);

    /***** probe points for Snapshot *****/

    ACX_PROBE_POINT #(
        .width(8),
        .tag("counter_a")
    ) probe_counter_a (
        .din(counter_a)
    );

endmodule // counters
`default_nettype wire

```

Next, at the top level where Snapshot is instantiated, we create matching ACX_PROBE_CONNECT instances that use the same tags. For counter_b the pin name method is used to create the connection. While counter_a and counter_b are seemingly driven by ACX_PROBE_CONNECT, behind the scenes they will be connected to the real counters.

Top-level module with Snapshot

```

`include "speedster16t/common/speedster16t_snapshot_v3.v"

`default_nettype none
`timescale 1ps/1ps
module snapshot_counter_v3_sc (
    // jtap ports:
    input wire    i_tck,
    input wire    i_trstn,
    input wire    i_tms,
    input wire    i_tdi,
    output wire   o_tdo,
    output wire   o_tdo_oen,

    // user design ports:
    input wire    i_clk_100,
    input wire    i_rst_n
);

    /***** clock *****/

    wire clk;
    CLK_IPIN ipin_i_clk_100(.din(i_clk_100), .dout(clk));

    /***** stimuli *****/

    // Snapshot stimuli are only valid when stimuli_valid is high.

```

```

wire stimuli_valid;
reg [2:0] stimuli_valid_d; // for edge detection/stretching

always @(posedge clk)
begin
    stimuli_valid_d <= (stimuli_valid_d << 1) | stimuli_valid;
end

/***** reset *****/

wire rst_n;
IPIN #(
    .mode(`DEF_IPIN_MODE_CLOCKED)
) ipin_i_reset_n (
    .din(i_rst_n),
    .clk(clk),
    .ce(1'b1),
    .rstn(1'b1),
    .dout(rst_n)
);

reg [3:0] pipe_i_rst_n; // pipeline from i_rst_n pad, for timing
wire do_reset; // set via stimuli (active-high)

// At edge detection of stimuli_valid edge, do_reset is a reset input (active-high).
// While do_reset stays high, we de-assert reset after 3 cycles.
reg reset_n = 0;
always @(posedge clk)
begin
    pipe_i_rst_n <= (pipe_i_rst_n << 1) | rst_n;

    if (stimuli_valid && !stimuli_valid_d[2])
        reset_n <= pipe_i_rst_n[3] && !do_reset;
    else
        reset_n <= pipe_i_rst_n[3];
end

/***** counter limit *****/

reg [7:0] limit_a = 62;

wire [7:0] limit_a_in; // set via stimuli; if not 0, value for limit_a
always @(posedge clk)
begin
    if (stimuli_valid && limit_a_in != 0)
        limit_a <= limit_a_in;
end

/***** user circuit *****/

wire all_zero;

```

```

counters x_counter (
    .clk(clk),
    .reset_n(reset_n),
    .limit_a(limit_a),
    .all_zero(all_zero)
);

/***** snapshot *****/

localparam MONITOR_WIDTH = 38;
localparam TRIGGER_WIDTH = 38;
wire [MONITOR_WIDTH-1 : 0] monitor;
wire arm;

wire [7:0] counter_a;
ACX_PROBE_CONNECT #(
    .width(8),
    .tag("counter_a")
) probe_counter_a (
    .dout(counter_a)
);

wire [15:0] counter_b;
ACX_PROBE_CONNECT #(
    .width(16),
    .pin("*.counter_b[*]:q")
) probe_counter_b (
    .dout(counter_b)
);

assign monitor = {
    counter_b,
    counter_a,
    limit_a,
    all_zero,
    arm,
    stimuli_valid,
    reset_n
};

localparam STIMULI_WIDTH = 9;
wire [STIMULI_WIDTH-1 : 0] stimuli;
assign {
    do_reset,
    limit_a_in
} = stimuli;

ACX_SNAPSHOT #(
    .DUT_NAME("v3"),
    .MONITOR_WIDTH(MONITOR_WIDTH),
    .MONITOR_DEPTH(3000), // will be rounded up
    .TRIGGER_WIDTH(TRIGGER_WIDTH),
    .NUM_TRIGGERS(3),

```

```

        .STANDARD_TRIGGERS(1), // use i_monitor as trigger input
        .STIMULI_WIDTH(STIMULI_WIDTH),
        .INPUT_PIPELINING(3),
        .OUTPUT_PIPELINING(0),
        .ARM_DELAY(2),
        .ENABLE_EDGE_TRIGGERS(1)
    ) x_snapshot (
        .i_tck(i_tck),
        .i_trstn(i_trstn),
        .i_tms(i_tms),
        .i_tdi(i_tdi),
        .o_tdo(o_tdo),
        .o_tdo_oen(o_tdo_oen),

        .i_user_clk(clk),
        .i_monitor(monitor),
        .i_trigger(), // not used if STANDARD_TRIGGERS = 1
        .o_stimuli(stimuli),
        .o_stimuli_valid(stimuli_valid),
        .o_arm(arm),
        .o_trigger()
    );


endmodule // snapshot_counter_v3_sc
`default_nettype wire

```

Chapter - 6: Running the Snapshot User Interface

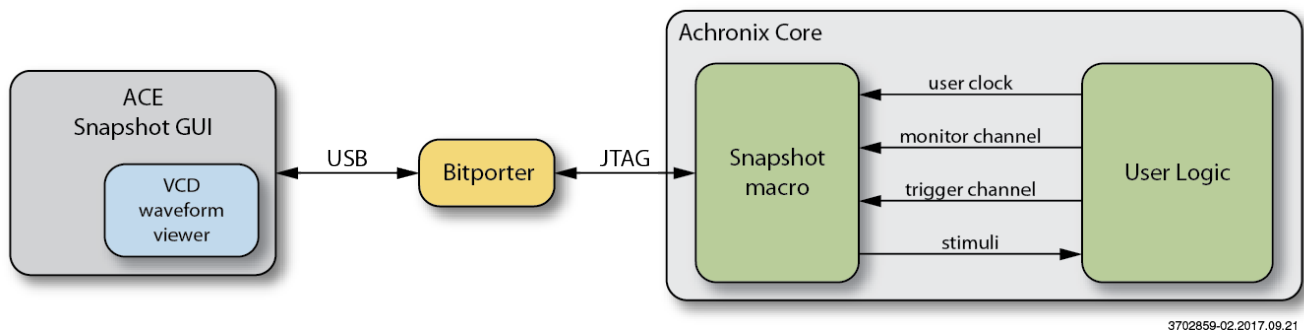


The JTAG connection must be configured before using the Snapshot Debugger!

ACE interacts with the FPGA using the JTAG interface through a Bitporter pod or FTDI FT2232H device. This JTAG interface must be properly configured in ACE before using the Snapshot Debugger view. The configuration is managed using the Configure JTAG Connection Preference Page, which is easily accessible by pressing the **Configure JTAG Interface** () button in the Snapshot Debugger view. See Configuring the JTAG Connection for more details.

Snapshot is the real-time design debugging tool for Achronix FPGAs. Snapshot, which is embedded in the ACE Software, delivers a practical platform to evaluate the signals of a user's design in real-time, and optionally send stimuli to the user's design.

To utilize the Snapshot debugger tool, the Snapshot macro must be instantiated inside the RTL for the Design-Under-Test (DUT). After instantiating the macro and programming the device, the user will be able to debug the design in the ACE GUI using the Snapshot Debugger view and the VCD Waveform Editor, found within the Bitporter perspective.



3702859-02.2017.09.21

Figure 5: Snapshot Communication with the Snapshot Debugger View within ACE (Running on the Host PC)

When instantiated in a design, the Snapshot macro can be used to interface with any logic mapped to the Achronix FPGA core. The Snapshot macro provides a JTAG / JTAP interface to control/observe debug logic mapped to the core. This allows the ACE Snapshot Debugger view, which drives the JTAG interface, to control / observe the signals associated with the debug logic.

Within the ACE GUI, the Snapshot Debugger view allows a designer to configure an embedded Snapshot Debugger core, interactively arm the core, and generate a VCD waveform output of the collected samples. By default, the generated VCD waveform output will be displayed in the ACE Editor Area using the VCD Waveform Editor. The VCD output can also be read into a third-party waveform viewer.

At a high level, to utilize Snapshot the user must first:

1. instantiate the Snapshot macro `ACX_SNAPSHOT` in the user's design
2. synthesize the design
3. place and route the design in ACE
4. generate the Bitstream for the design in ACE
5. configure ACE's JTAG connection to the FPGA (see Configuring the JTAG Connection)

6. program the Achronix device with the Bitstream

- use of the ACE GUI's Download View is documented in the section Playing a STAPL File (Programming a Device)
- use of the `acx_stapl_player` executable on the command-line is documented in the *Bitstream Programming and Debug Interface User Guide* (UG004)

Once those prerequisite steps are complete, the ACE GUI's Snapshot Debugger View allows the user to evaluate /interact with the running design in real-time

The following sections will further explain Snapshot and guide the user through the process.

Accessing the Snapshot Debugger

Open the ACE GUI and Select Your Project

Open the ACE GUI tool, and load or activate your project in the Projects View as shown below. See the Loading Projects, Setting the Active Implementation, and Working with Projects and Implementations sections for details.

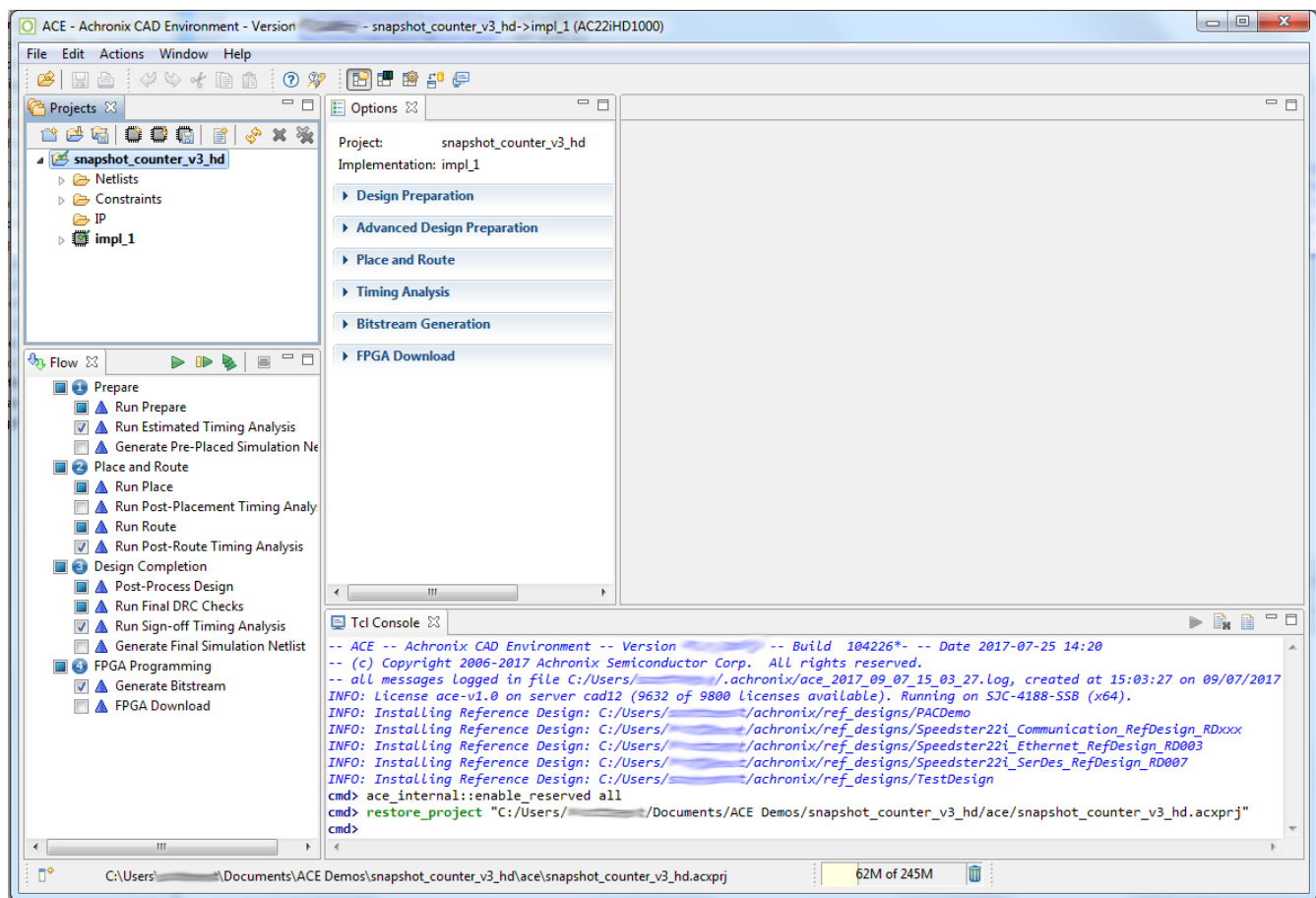




Figure 6: ACE Tool

Open the Snapshot Debugger

Click the toolbar button to change to the Programming and Debug Perspective () as described in the Working with Perspectives section. The Snapshot Debugger View should be visible by default, as shown below. If not, you can click **Window**→**Show View**→**Snapshot Debugger** from the main menu bar.

The Snapshot Debugger View should have automatically loaded the default Snapshot configuration file for your project that was generated when you ran your design through place and route, located in `<ace_project_dir>/<active_impl_dir>/output/names.snapshot`. If it loaded, you will see the correct signal names from your user design in the Trigger Channels, Monitor Channels, and Stimuli tables. If it did not automatically load, you can click the **Load Snapshot Configuration** () toolbar button in the Snapshot Debugger View to browse to the location of your preferred *.snapshot configuration file, or manually enter the signal names, channel widths, etc to match your design.

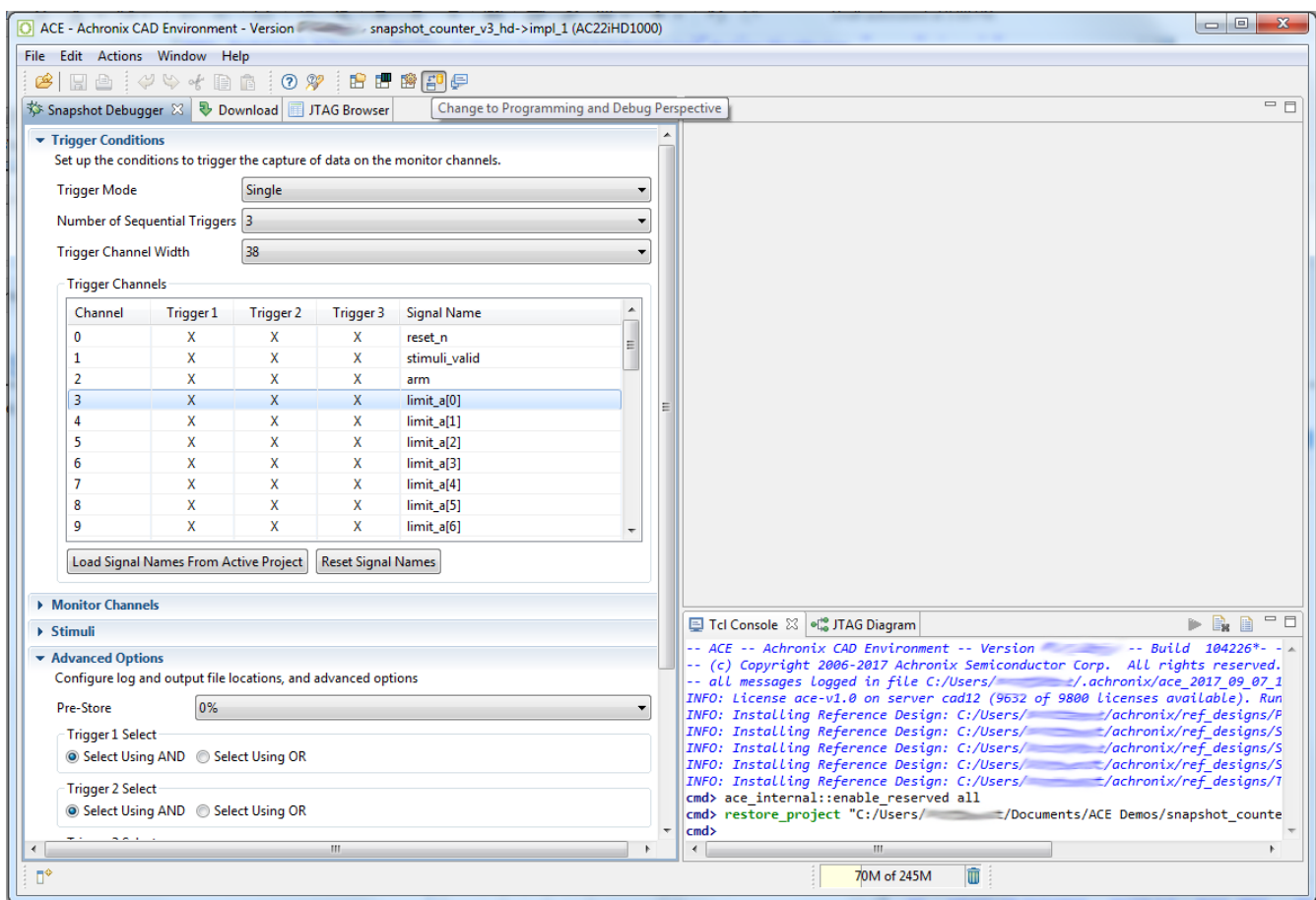


Figure 7: Snapshot Debugger View

Configuring the Trigger Pattern

The Trigger Channel signal names are automatically configured to the correct values when the names.snapshot file is loaded. The names.snapshot file is generated during design preparation (the **Run Prepare Flow Step**), which contains the user design signal names connected to Snapshot, along with the trigger width and the maximum number of sequential triggers.

Configuring the Trigger Mode

The **Trigger Mode** option allows the user to select the trigger mode to use when the Arm action is run.

Single

The default trigger mode is **Single**, which means the trigger conditions are programmed in to the ACX_SNAPSHOT macro and then the GUI waits for a single trigger event to occur which matches those trigger conditions, and then a single VCD file is recorded. This option arms Snapshot and captures data only once.

Immediate

If **Immediate** trigger mode is selected, pressing the Arm button results in the same behavior as **Single** trigger mode, except that all 3 trigger patterns are treated as "Don't Care" (X's) so that the trigger event will occur as soon as the Arm button is pressed. This mode is useful to quickly capture the state of the running design without waiting for any trigger pattern to be met.

Repetitive

If **Repetitive** trigger mode is selected, the trigger conditions are programmed in to the ACX_SNAPSHOT macro and samples are captured repetitively until the upper limit of trigger event records is reached. When **Repetitive** trigger mode is selected, an additional set of repetitive trigger mode options will appear to allow the user to configure the number of sequential times Snapshot should be armed repetitively using the configured trigger conditions, and the way in which the output VCD files are managed. This mode is useful when the trigger conditions do not narrow in on the exact data pattern and the pattern you intend to observe occurs sporadically at the trigger conditions. You can let the repetitive trigger mode run for a long period of time, taking several capture records at the trigger conditions, to help find the pattern you are interested in. The user can optionally cancel the remaining Snapshot session once the desired data is captured.

The repetitive trigger Record Limit setting determines how many times (number of records) the GUI will repeatedly Arm the Snapshot debugger and capture samples. The user may set this to automatically run Snapshot up to 128 times.

The repetitive trigger VCD Record Limit setting determines how many Snapshot records to capture in a single VCD file. This essentially concatenates the VCD files from consecutive runs of Snapshot (records) into a single VCD file. The VCD file waveform contains a set of virtual signals to indicate the system timestamp at which each Snapshot record was captured. The user may concatenate up to 10 Snapshot records in a single VCD file.

If the Overwrite VCD File option is selected, the VCD Waveform File name specified in the Advanced Options section will be used to store the output VCD file. The file will be overwritten with the new VCD file each time the VCD record limit is reached. If the Overwrite VCD File option is not selected, then multiple VCD files will be written out and a unique VCD record number will be added to the VCD Waveform File name specified in the Advanced Options section for each VCD. For example, if you set the Record Limit to 8 and set the VCD Record Limit to 2, and set the VCD Waveform file path the `"/snapshot.vcd"`, then Snapshot would output 4 VCD files to `"/snapshot1.vcd"`, `"/snapshot2.vcd"`, `"/snapshot3.vcd"`, `"/snapshot4.vcd"`, each containing 2 Snapshot capture records.

Configuring Trigger Patterns

The Snapshot Debugger can be configured to use a **Trigger Channel Width** of 1 to 40 bits. The value entered in the Snapshot Debugger View must match the value of the `TRIGGER_WIDTH` parameter set on the ACX_SNAPSHOT module in the user design RTL. (This will be the width of the *i_trigger* bus.)

The Snapshot Debugger is capable of handling one to three sequential trigger patterns. The post-trigger data is sampled once the last trigger pattern in the sequence is matched.

The user may specify the number of desired sequential trigger patterns using the **Number of Sequential Triggers** option in the Snapshot Debugger View. If **1** is selected, Trigger 2 and Trigger 3 are ignored. If **2** is selected, Trigger 3 is ignored and Snapshot will trigger when Trigger 1 is matched, followed (on any subsequent clock) by a match on Trigger 2. If **3** is selected, then Snapshot will trigger after a match on Trigger 1, followed (on any subsequent clock) by a match on Trigger2, followed (on any subsequent clock) by a match on Trigger3.

Each sequential trigger is hooked up to the trigger channels on the Snapshot Debugger core. The LSb of the trigger pattern is hooked to trigger channel 0, and the MSb is hooked to upper most trigger channel bit (TRIGGER_WIDTH - 1).

Each sequential trigger is made up of three parts: the pattern mask, the edge mask, and the don't care mask. In the Snapshot Debugger View, these 3 masks are combined for ease of use into a single trigger pattern value, which allows each bit to be specified as **X** (don't care), **R** (rising edge), **F** (falling edge), **0** (level 0), or **1** (level 1). The trigger pattern defines the trigger channel signal conditions that are required to detect a match. If a given trigger channel value is set to X (don't care), then this trigger channel is ignored when computing a match. If a given trigger channel value is set to R (rising edge), then this trigger channel is evaluated as a match when a rising edge of this signal is seen by Snapshot. If a given trigger channel value is set to F (falling edge), then this trigger channel is evaluated as a match when a falling edge of this signal is seen by Snapshot. If a given trigger channel value is set to 1 (level 1), then this trigger channel is evaluated as a match as long as this signal's level is seen as a 1 by Snapshot (it is not edge sensitive). If a given trigger channel value is set to 0 (level 0), then this trigger channel is evaluated as a match as long as this signal's level is seen as a 0 by Snapshot (it is not edge sensitive).



If any active Trigger is configured with as all X's (don't care), the trigger pattern will be a match on the first clock cycle that trigger is evaluated.

The values within a trigger pattern may cause a trigger match event either by AND'ing or OR'ing. If AND'ing, then **all** signal values not masked (set to X) must match their pattern for the trigger match event to occur. If OR'ing, then the trigger match event will occur if **any** of the non-masked (not set to X) signal values match the specified pattern. The AND/OR configuration is set per sequential trigger using the **Select using AND** or **Select using OR** radio buttons. This selection can be different for each sequential trigger.

In the "Trigger Channels" table of the Snapshot Debugger View, the trigger patterns can be viewed and edited.

Setting Pattern Values Using the Table

For each channel, a value of **X** (don't care), **R** (rising edge), **F** (falling edge), **0** (level 0), or **1** (level 1) can be specified via a pull-down menu under each "Trigger" column as shown below.

Trigger Channels

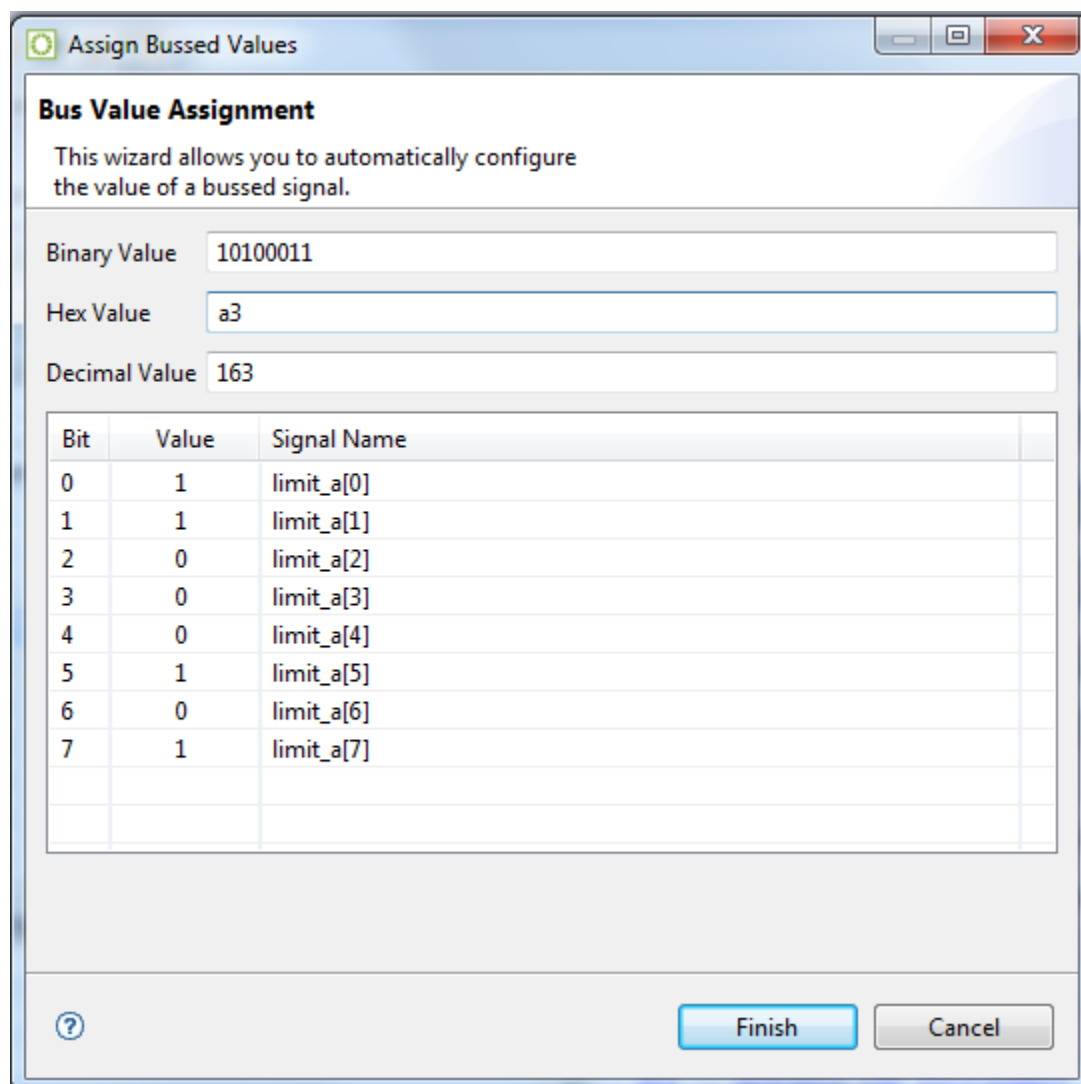
Channel	Trigger 1	Trigger 2	Trigger 3	Signal Name
0	X	X	X	reset_n
1	X	X	X	stimuli_valid
2	X	X	X	arm
3	X	X	X	limit_a[0]
4	X	X	X	limit_a[1]
5	0	X	X	limit_a[2]
6	1	X	X	limit_a[3]
7	R	X	X	limit_a[4]
8	F	X	X	limit_a[5]
9	X	X	X	limit_a[6]

Load Signal Names From Active Project Reset Signal Names

Setting Multiple Pattern Values as a Bus

The Assign Bussed Values Dialog wizard allows the user to assign a value to multiple signals from the SnapShot Debugger view's "Trigger Channels" or "Stimuli Channels" tables as a bus. After configuring the bus in the dialog, the values of each signal are propagated to all the selected signals in the SnapShot Debugger View. There are 2 ways to launch this dialog to allow bus assignment of values:

1. With your mouse, left click to select a single row in the SnapShot Debugger View table which has a bussed signal name (i.e. din[2]). Then right mouse click to edit the **Value by Bus**. This method will automatically find all the other bits in the bus with the same signal name (i.e. din[0], din[1], din[2], etc.) and open the dialog to allow editing of the entire bus of signals.
2. With your mouse, hold CTRL or SHIFT and left click to select multiple rows in the SnapShot Debugger View table. Then right mouse click to edit the **Value by Selection**. This method will open the dialog to allow editing of all selected signals as a bussed value.



Assign Bussed Values

Bus Value Assignment

This wizard allows you to automatically configure the value of a bussed signal.

Binary Value: 10100011

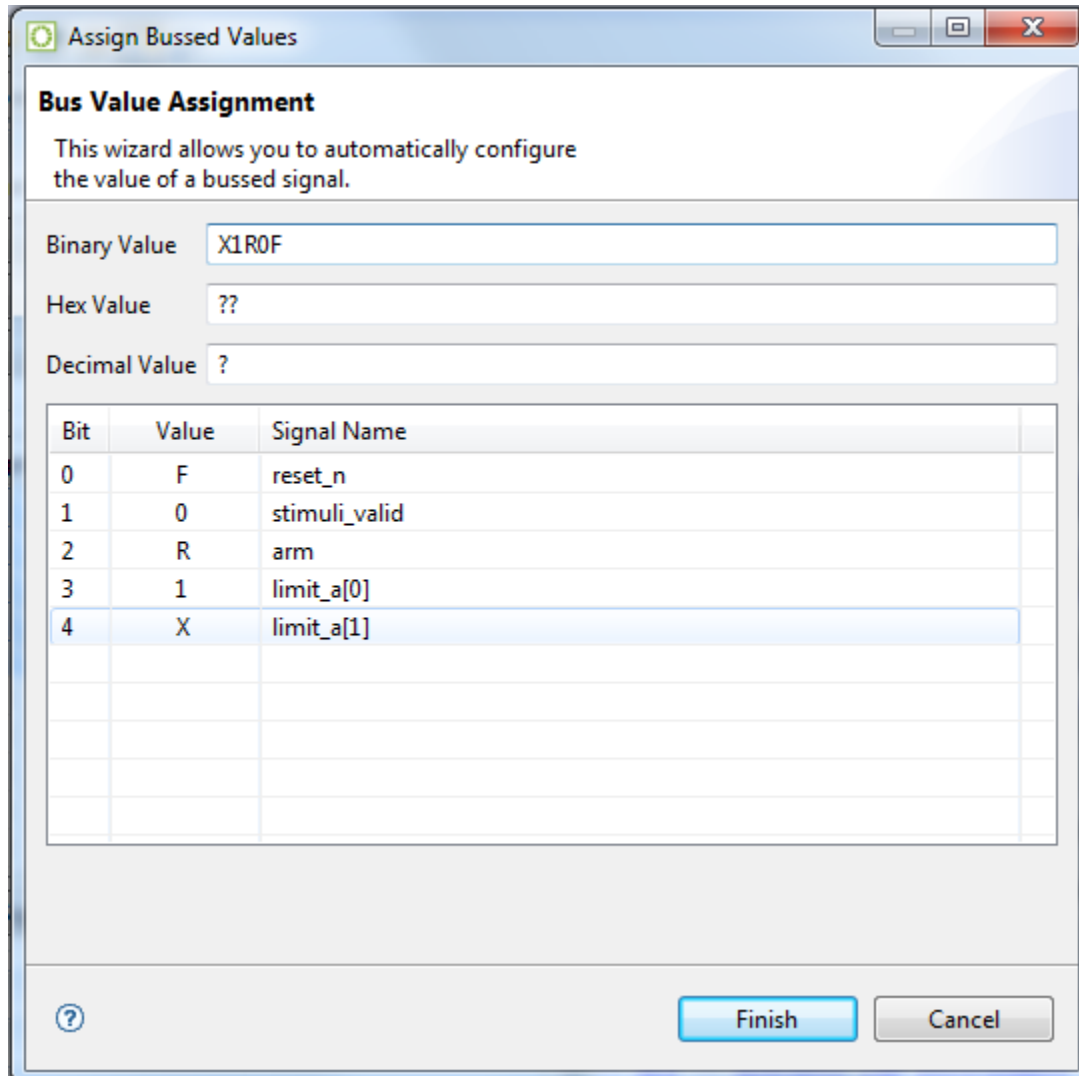
Hex Value: a3

Decimal Value: 163

Bit	Value	Signal Name
0	1	limit_a[0]
1	1	limit_a[1]
2	0	limit_a[2]
3	0	limit_a[3]
4	0	limit_a[4]
5	1	limit_a[5]
6	0	limit_a[6]
7	1	limit_a[7]

?

Finish Cancel



See Assign Bussed Values Dialog for more information on this dialog.

Configuring the Monitor Signals




The Monitor Signals are automatically configured to the correct values when the `names.snapshot` file is loaded. The `names.snapshot` file is generated during design preparation (the **Run Prepare** Flow Step), which contains the user design signal names connected to Snapshot, along with the monitor width and number of samples.

The value of **Monitor Channel Width** in the SnapShot Debugger view must be configured to match the value of the `MONITOR_WIDTH` parameter of the `ACX_SNAPSHOT` instance inside the RTL of the design being debugged. (This will be the width of the `i_monitor` bus.)

The value of **Number of Samples** in the SnapShot Debugger view should be configured to match the value of the `MONITOR_DEPTH` parameter of the `ACX_SNAPSHOT` instance inside the RTL of the design being debugged. If the value in the GUI does not match the value in the RTL, the value from the RTL will be used and a warning will be printed in the Snapshot log file.

Naming captured signal data

Custom signal names for each channel can be entered under the "Signal Name" heading within the "Monitor Channels" table. The signal/bus names in the "Monitor Channels" table are then used as labels on the captured signal data in the VCD waveform output, and will be visible in the VCD Waveform Editor.

Multiple signals can be combined into a bus by selecting multiple rows in the "Monitor Channels" table, right-clicking on a selected signal row to bring up a popup context menu, and selecting **Assign Bus Name** () from the context menu to bring up the Assign Bussed Signal Names Dialog. After configuring the bus in the Assign Bussed Signal Names Dialog, the bus name and indices are propagated to all the previously-selected signals. To select a contiguous range of rows, select the first signal, hold the Shift key, and select the last signal. To select a non-contiguous set of rows, select the first signal, then while holding down the Ctrl key on the keyboard, select the other signals.

Signal names may be returned to their defaults by selecting the **Reset Signal Names** button under the "Monitor Channels" table. Note that this resets all signal names in the table at once, not just the currently selected rows/signals.

The **Load Signal Names From Active Project** button loads the `names.snapshot` file generated during design preparation (the **Run Prepare** Flow Step), which renames all signals with their project-specific names, and also loads the project-specific default settings for monitor width, user clock frequency, default log and vcd file path, etc.

Configuring the Test Stimuli



The stimuli channel signal names are automatically configured to the correct values when the `names.snapshot` file is loaded. The `names.snapshot` file is generated during design preparation (the **Run Prepare** Flow Step), which contains the user design signal names connected to Snapshot, along with the stimuli width.

Snapshot has the capability to send 0 to 512 bits of test stimuli (the `ACX_SNAPSHOT` macro output signal `o_stimuli`) to the Design Under Test (DUT). This data is sent once per arming session, is only valid while the `o_stimuli_valid` signal is high.

This `o_stimuli` output is optional, and need not be connected to the DUT - it may safely be left floating when the user wants to use Snapshot to only read signals.

The value of **Stimuli Channel Width** in the SnapShot Debugger view must be configured to match the value of the `STIMULI_WIDTH` parameter of the `ACX_SNAPSHOT` instance inside the RTL of the design being debugged. (This will be the width of the `o_stimuli` bus.)

In the "Stimuli Channels" table of the Snapshot Debugger View, the stimuli values can be viewed and edited.

Setting Stimuli Values Using the Table

For each channel, an output value of **0** (level 0), or **1** (level 1) can be specified via a pull-down menu under the "Value" column as shown below.

Stimuli Channels

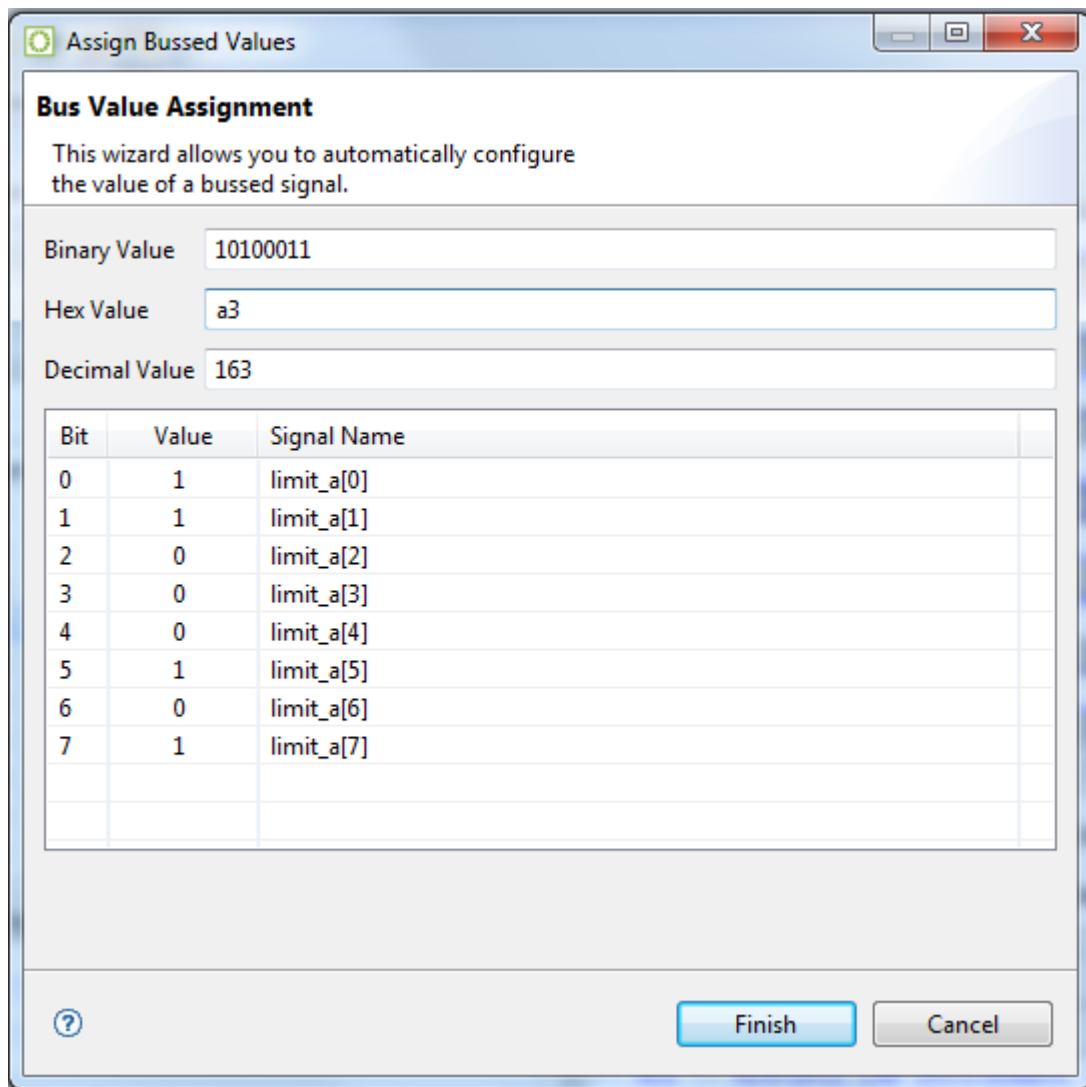
Channel	Value	Signal Name
0	0	dut_stimuli[0]
1	0	dut_stimuli[1]
2	0	dut_stimuli[2]
3	0	dut_stimuli[3]
4	0	dut_stimuli[4]
5	0	dut_stimuli[5]
6	1	dut_stimuli[6]
7	0	dut_stimuli[7]
8	0	reset_n

Load Signal Names From Active Project Reset Signal Names

Setting Multiple Stimuli Values as a Bus

The Assign Bussed Values Dialog wizard allows the user to assign a value to multiple signals from the SnapShot Debugger view's "Stimuli Channels" table as a bus. After configuring the bus in the dialog, the values of each signal are propagated to all the selected signals in the SnapShot Debugger View. There are 2 ways to launch this dialog to allow bus assignment of values:

1. With your mouse, left click to select a single row in the SnapShot Debugger View table which has a bussed signal name (i.e. din[2]). Then right mouse click to edit the **Value by Bus**. This method will automatically find all the other bits in the bus with the same signal name (i.e. din[0], din[1], din[2], etc.) and open the dialog to allow editing of the entire bus of signals.
2. With your mouse, hold CTRL or SHIFT and left click to select multiple rows in the SnapShot Debugger View table. Then right mouse click to edit the **Value by Selection**. This method will open the dialog to allow editing of all selected signals as a bussed value.



See Assign Bussed Values Dialog for more information on this dialog.

Configuring Advanced Options

Pre-Store

In the Snapshot Debugger View, the **Pre-Store** setting configures the portion of samples that are collected before the trigger, and (indirectly) how many are collected after the trigger.

For example, assume the user has configured Snapshot to use a monitor depth of 1024 samples. See the table below:

Table 7: Effect of "Pre-store" on samples collected before and after the trigger event

"Pre-Store" value	Samples collected before trigger	Samples collected after trigger
0%	0	1024

"Pre-Store" value	Samples collected before trigger	Samples collected after trigger
25%	256	768
50%	512	512
75%	768	256

When a **Pre-Store** value other than **0%** is selected, the `.vcd` file will contain a signal `snapshot_pre_store` that transitions (goes low) at the point where the (last sequential) trigger event occurred. Thus, users may easily find the trigger event without needing to actually count the samples.

Trigger Pattern Match Behavior

The values within a trigger pattern may cause a trigger match event either by AND'ing or OR'ing. If AND'ing, then **all** signal values not masked (set to X) must match their pattern for the trigger match event to occur. If OR'ing, then the trigger match event will occur if **any** of the non-masked (not set to X) signal values match the specified pattern. The AND/OR configuration is set per sequential trigger using the **Select using AND** or **Select using OR** radio buttons. This selection can be different for each sequential trigger.

User Clock Frequency

The **Frequency** field must be configured to match the `user_clk` frequency in the target user design, which typically matches the timing constraint set in the SDC file of the design being debugged. The value from the user design SDC file will be set automatically in the `names.snapshot` file when an active implementation is available. The frequency value entered in the Snapshot GUI (or `.snapshot` configuration file) will determine the time (in picoseconds) for all signals shown in the captured VCD file. All samples are captured at the rising edge of the Snapshot `user_clk` signal.

Configure output file locations

The final Snapshot configuration steps specify the locations of the output files which will contain the log messages and sample data collected by Snapshot.



File Paths Relative To Chooses whether the **Log File** and **Waveform File** paths are understood to be relative to the **Active Project's** directory or to the **Working Directory**. (Only matters when the file paths provided are relative paths, and not absolute paths.)

Log File configures the file name and path for the log file generated by the Snapshot Debugger run. The associated **Browse** button provides a directory/file selection dialog for the selection of a location different from the default. (The default will be '`<active_impl_dir>/log/snapshot.log`', or if there is no Active Project and Implementation, then '`<user_home>/snapshot.log`'.) If an error occurs during setup or while reading back the sample information, the Snapshot log file will contain the error messages.

Waveform File configures the file name and path for storing downloaded sample waveform information from the Snapshot Debugger core in VCD format. The **Browse** button allows for the selection of a location different from the default. (The default will be '`<active_impl_dir>/output/snapshot.vcd`', or if there is no active implementation, then '`<user_home>/snapshot.vcd`'.)


Collecting Samples of the User Design

Using the Startup Trigger

The Startup Trigger feature requires that the end user has configured the initial startup trigger parameters on the ACX_SNAPSHOT macro to enable the Startup Trigger feature, and that the Arm Snapshot action has not been executed since the bitstream has been programmed. By clicking the **Capture Startup Trigger** () button, the Snapshot Debugger View will connect to the running ACX_SNAPSHOT circuit over JTAG and wait for the startup trigger condition to be met, retrieve the trace buffer contents, and output a VCD file. This feature is useful to capture trigger events that happen very soon after the Achronix FPGA enters user mode. Once the **Arm Snapshot** () button is pressed, the Startup trigger conditions and any existing trace buffer contents are cleared. The Startup Trigger feature may only be used once after programming the bitstream.


Arming the Snapshot Debugger

Once all the fields in the Snapshot Debugger View are configured, and the design is running on the target device, Snapshot is ready to be Armed.

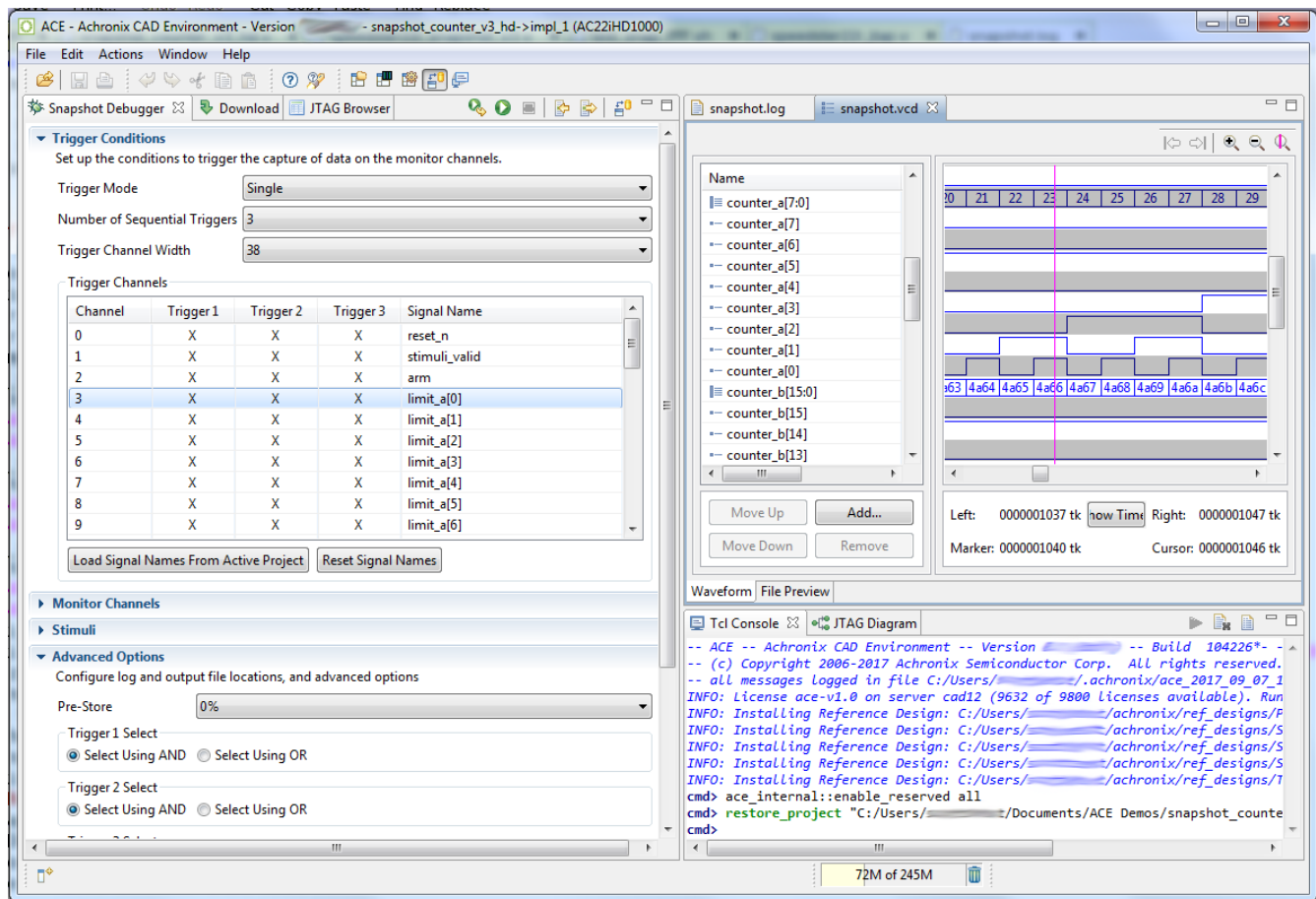
Select the **Arm** button [or the **Arm Snapshot** () button in the SnapShot Debugger view's toolbar], and the ACE Snapshot Debugger will send the configuration data (including the optional *Stimulus*) to the ACX_SNAPSHOT circuit running on the Achronix device, wait for the trigger condition(s) to be met, retrieve the trace buffer contents, and output a VCD file as well as a LOG file.

Once Armed, Snapshot will begin to analyze the already-executing design in real-time.

The Snapshot Log file and Snapshot Waveform file are populated with the captured results, and the files are opened in ACE. (The log file will open in an ACE Text Editor, while the waveform (`.vcd`) file will open in the ACE VCD Waveform Editor.) If an error occurs during Snapshot Debugger configuration or while reading back the sampled information (trace buffer), the Snapshot Log file will contain the relevant error messages, and the Snapshot Waveform file will not be created/updated.


The **Cancel** () button aborts the Snapshot Arming process. The Snapshot Log file will be updated, but the Snapshot Waveform file will not be created/updated if Cancel is pressed. Cancel is useful if you accidentally send in trigger conditions that are never matched.


If using **Repetitive** Trigger Mode, Snapshot will repetitively execute the Arm action for the number of records specified, or until Cancel is pressed. See Configuring the Trigger Pattern for details on the Repetitive Trigger feature.



Saving/Loading Snapshot Configurations

Users may wish to re-use an existing known-good Snapshot configuration (the collection of settings in the Snapshot Debugger View) at a later date, or in batch mode.

Snapshot configurations may be saved to a Snapshot configuration file (with the `.snapshot` file extension) using the **Save Snapshot Configuration** () button found in the Snapshot Debugger View's toolbar.

These Snapshot configurations may then be loaded later by using the **Load Snapshot Configuration** () button, found in the Snapshot Debugger View's toolbar.

 Previously saved Snapshot configuration files are necessary to run Snapshot in Batch Mode.



When a user design containing the `ACX_SNAPSHOT` macro completes the Flow Step **Run Prepare**, a names `.snapshot` configuration file is automatically generated. This file contains harvested information from the design including the monitor width, monitor depth, monitored signal names, trigger width, maximum number of triggers, trigger signal names, stimuli width, stimuli signal names, and user clock frequency. When an Active Project and Implementation is available, the Snapshot Debugger View automatically loads the implementation's names `.snapshot` file to pre-populate the relevant fields of the view. Note that when generated, the file contains only a subset of a complete Snapshot configuration, and thus a generated names `.snapshot` file should not be used to drive Snapshot in Batch Mode via Tcl.

The names.snapshot configuration file can be loaded as a starting point to map the Snapshot RTL configuration into the Snapshot Debugger View. The Snapshot settings can be further customized and saved as custom Snapshot configuration files for later use.

Running Snapshot in Batch Mode

It is also possible to run Snapshot from ACE in batch mode. To do so, use the TCL command `run_snapshot`. Note that `run_snapshot` requires the use of a previously-saved Snapshot configuration file (`.snapshot`), and allows some values to be overridden from the TCL commandline. See the `run_snapshot` command in the TCL Command Reference section for further details.

The Snapshot configuration file may be edited manually in a text editor, or be configuring the Snapshot Debugger View in the ACE GUI and saving the Snapshot configuration.

Example Snapshot Configuration File

```
#Snapshot Configuration File
#Tue Sep 12 13:52:54 PDT 2017
files_relative_to_project=1
frequency=322.0
log_file=./impl_1/log/snapshot.log
monitor_ch0.name=reset_n
monitor_ch1.name=stimuli_valid
monitor_ch10.name=limit_a[7]
monitor_ch11.name=counter_a[0]
monitor_ch12.name=counter_a[1]
monitor_ch13.name=counter_a[2]
monitor_ch14.name=counter_a[3]
monitor_ch15.name=counter_a[4]
monitor_ch16.name=counter_a[5]
monitor_ch17.name=counter_a[6]
monitor_ch18.name=counter_a[7]
monitor_ch19.name=counter_b[0]
monitor_ch2.name=arm
monitor_ch20.name=counter_b[1]
monitor_ch21.name=counter_b[2]
monitor_ch22.name=counter_b[3]
monitor_ch23.name=counter_b[4]
monitor_ch24.name=counter_b[5]
monitor_ch25.name=counter_b[6]
monitor_ch26.name=counter_b[7]
monitor_ch27.name=counter_b[8]
monitor_ch28.name=counter_b[9]
monitor_ch29.name=counter_b[10]
monitor_ch3.name=limit_a[0]
monitor_ch30.name=counter_b[11]
monitor_ch31.name=counter_b[12]
monitor_ch32.name=counter_b[13]
monitor_ch33.name=counter_b[14]
monitor_ch34.name=counter_b[15]
monitor_ch4.name=limit_a[1]
monitor_ch5.name=limit_a[2]
monitor_ch6.name=limit_a[3]
```

```

monitor_ch7.name=limit_a[4]
monitor_ch8.name=limit_a[5]
monitor_ch9.name=limit_a[6]
monitor_width=38
num_samples=4096
num_triggers=3
pre_store=0%
repetitive_trigger.override_vcd=0
repetitive_trigger.record_limit=10
repetitive_trigger.vcd_record_limit=10
snapshot_version=3
stimuli=110010100
stimuli_ch0.name=stimuli[0]
stimuli_ch1.name=stimuli[1]
stimuli_ch2.name=stimuli[2]
stimuli_ch3.name=stimuli[3]
stimuli_ch4.name=stimuli[4]
stimuli_ch5.name=stimuli[5]
stimuli_ch6.name=stimuli[6]
stimuli_ch7.name=stimuli[7]
stimuli_ch8.name=do_reset
stimuli_ch9.name=stimuli_ch9
stimuli_width=9
trigger1=XXXXXXXXXXXXXXXXXXXX00110101XXXXXXXXXXXX
trigger1.select_using_and=1
trigger2=XXXXXXXXXXXXXXXXXXXX1111R000XXXXXXXXXXXX
trigger2.select_using_and=1
trigger3=XXXXXXXXXXXXXXXXXXXXXXXFXXXXXXXXXXXXXX
trigger3.select_using_and=1
trigger_ch0.name=reset_n
trigger_ch1.name=stimuli_valid
trigger_ch10.name=limit_a[7]
trigger_ch11.name=counter_a[0]
trigger_ch12.name=counter_a[1]
trigger_ch13.name=counter_a[2]
trigger_ch14.name=counter_a[3]
trigger_ch15.name=counter_a[4]
trigger_ch16.name=counter_a[5]
trigger_ch17.name=counter_a[6]
trigger_ch18.name=counter_a[7]
trigger_ch19.name=counter_b[0]
trigger_ch2.name=arm
trigger_ch20.name=counter_b[1]
trigger_ch21.name=counter_b[2]
trigger_ch22.name=counter_b[3]
trigger_ch23.name=counter_b[4]
trigger_ch24.name=counter_b[5]
trigger_ch25.name=counter_b[6]
trigger_ch26.name=counter_b[7]
trigger_ch27.name=counter_b[8]
trigger_ch28.name=counter_b[9]
trigger_ch29.name=counter_b[10]
trigger_ch3.name=limit_a[0]

```

```
trigger_ch30.name=counter_b[11]
trigger_ch31.name=counter_b[12]
trigger_ch32.name=counter_b[13]
trigger_ch33.name=counter_b[14]
trigger_ch34.name=counter_b[15]
trigger_ch4.name=limit_a[1]
trigger_ch5.name=limit_a[2]
trigger_ch6.name=limit_a[3]
trigger_ch7.name=limit_a[4]
trigger_ch8.name=limit_a[5]
trigger_ch9.name=limit_a[6]
trigger_mode=Single
trigger_width=38
vcd_file=./impl_1/output/snapshot.vcd
```

Revision History

Version	Date	Description
1.0	April 5, 2013	<ul style="list-style-type: none">Initial Achronix release.
1.1	April 17, 2013	<ul style="list-style-type: none">Updated module name to ACX_SNAPSHOT.
1.2	July 12, 2016	<ul style="list-style-type: none">Modified name of document to not be Speedster22i specific.
1.3	July 17, 2016	<ul style="list-style-type: none">Ported document to Confluence and re-drew figures.Modified monitor/trigger bus widths to the original 36-bit variants.Put in information on multiple Snapshot instances through a single JTAG port and the new feature to display bus values in timing waveforms.
1.4	August 2, 2016	<ul style="list-style-type: none">Included section on Probing in a Hierarchical Design (see page 25).Updated parameter list and corrected wording in various sections.
2.0	September 24, 2017	<ul style="list-style-type: none">Extensive reworking and updating of the content to reflect newly available features as part of the Snapshot version 3 release, including startup trigger, edge triggering, repetitive trigger mode, configurable monitor and stimuli widths.
2.1	October 23, 2018	<ul style="list-style-type: none">Snapshot General Description: (see page 7) Minor updates to the Triggers (see page 7) section.Snapshot Interface (see page 11): Corrections to the parameter table and additional descriptions.