



Digital Logic  
Design with FPGA

# Design Flow with Quartus

Simulation & HW Debug



# Outline

- Modeling digital systems
- Simulation in typical design flow
  - Simulation tools
  - Timing simulation
- Hardware debug
  - In-System Sources and Probes (ISSP)
  - Signal Tap Internal Analyzer

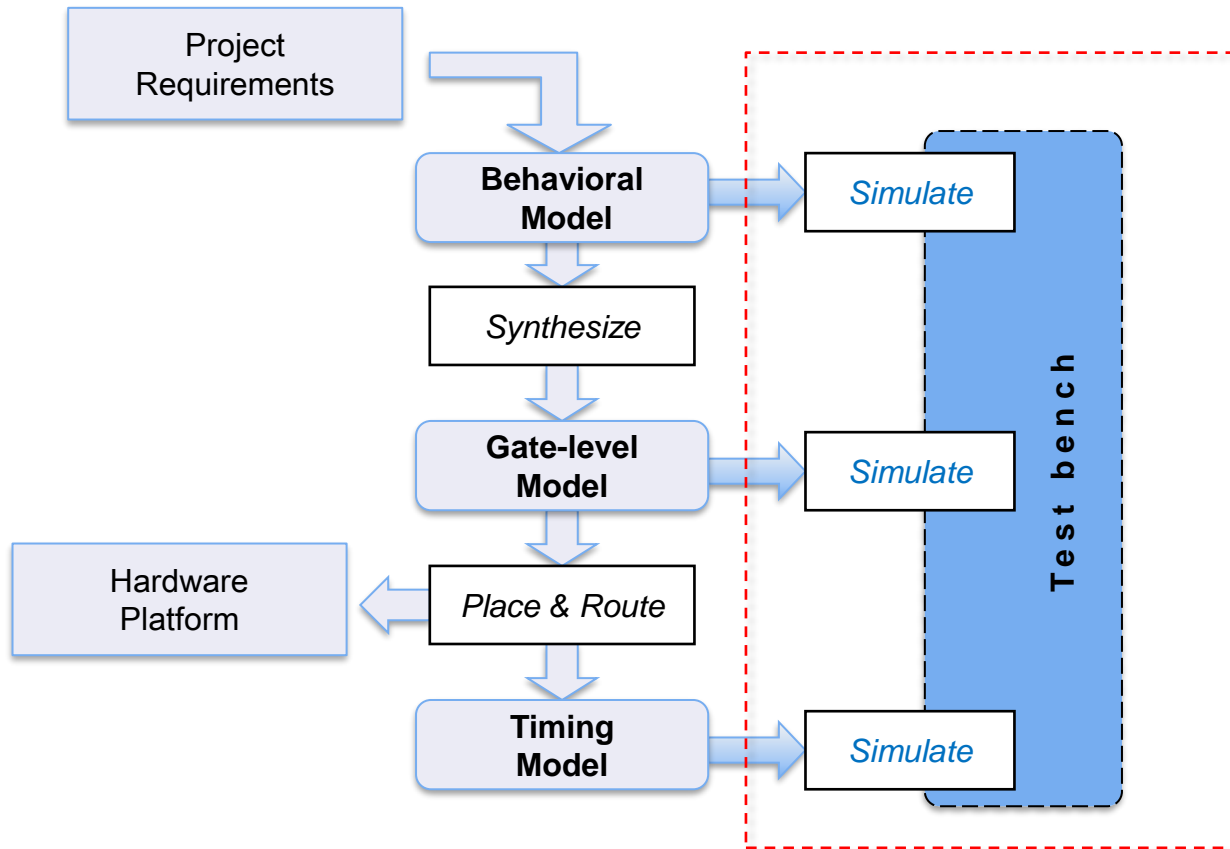


# Modeling Digital Systems

- Simulation and modeling at the system level
  - performance analysis
- Behavior specification at the algorithmic level
  - preliminary functional verification of algorithms
  - division into hardware and software
  - high-level synthesis
- Simulation behavioral models of standard elements
- Functional simulations at the system/package level
  - full
  - bus
- Synthesizable models at the RTL (register transfer level)
  - full functional specification of the project
- Model of the system environment (testbench)
- Simulation models of library cells from integrated circuit manufacturers (VITAL standard) –
  - time verification of ASIC/FPGA systems



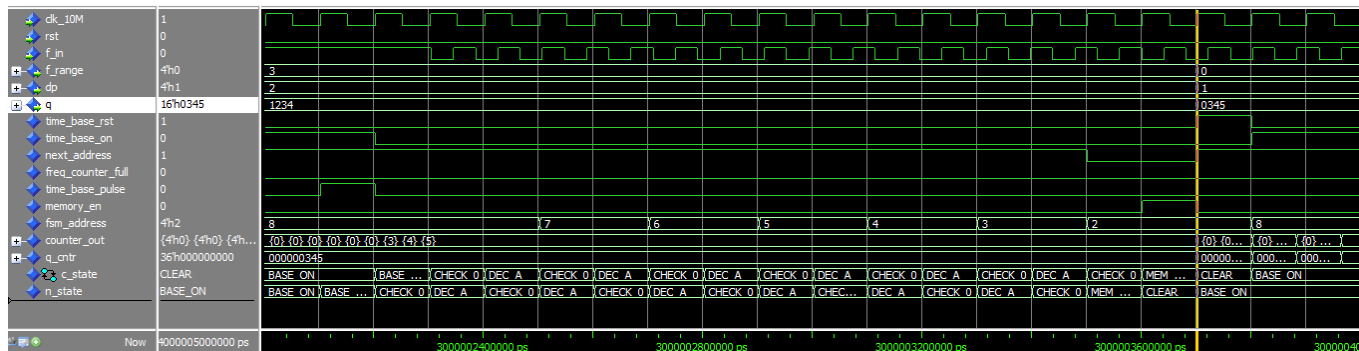
# Simulation in typical design flow





# Simulation

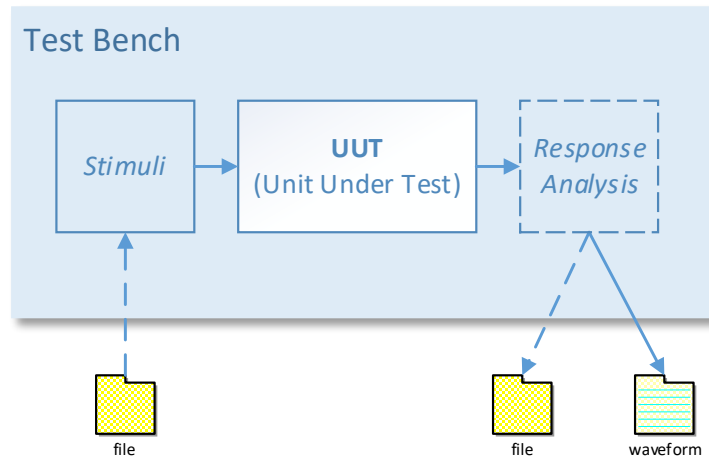
- Provide results that are impossible to measure in HW prototype
- Include wide range of analyses
- Reduce development costs
- Minimize time-to-market
- Delivering visibility of all signals in design
- ...
- Designer has to create stimulus that matches device behavior
- Can take very long time to run for large designs





# HDL Testbench

- **Testbench** is a specific design unit described in HDL
- It has no ports and is intended to simulate the designed device (UUT)
- Defines forces (stimuli) and how to interpret simulation results
- Creating a testbench:
  - written "by hand" by the designer
  - generated automatically using graphic editors based on the description of UUT ports





# HDL Testbench

```
6  library IEEE;
7  use IEEE.STD_LOGIC_1164.ALL;
8  use std.env.all;
9
10 entity d_cntr4ceo_tb is
11     generic(T: time:= 1 ns);
12 end entity d_cntr4ceo_tb;
13
14 architecture behav of d_cntr4ceo_tb is
15     signal clk,rst: std_logic := '0';
16     signal ce,ceo,tc : std_logic;
17     signal q: std_logic_vector(3 downto 0);
18
19     constant PERIOD : delay_length := 20*T;
20
21     procedure clk_gen(signal s: out std_logic; period: delay_length := 10 ns) is
22     begin
26
27     procedure pulse (signal s: out std_logic; Hpulse,Lspace: delay_length) is
28     begin
32
33     procedure stop_after_falling_edge(signal trig: in std_logic; delay: delay_length := 0 ns) is
34     begin
41
42     procedure stop after Nfalling edge(signal trig: in std logic; delay: delay length := 0 ns; N: natural:=2) is
44     begin
57
58 begin
59
60     UUT: entity work.d_cntr4ceo
61     port map(clk,rst,ce,tc,ceo,q);
62
63     pulse(ce,105*PERIOD,2*PERIOD);
64     pulse(rst,2*PERIOD,12*PERIOD);
65     clk_gen(clk,PERIOD);
66     stop_after_Nfalling_edge(ceo,PERIOD,5);
67
68 end architecture behav;
```

Entity without I/O

Stimulus definition

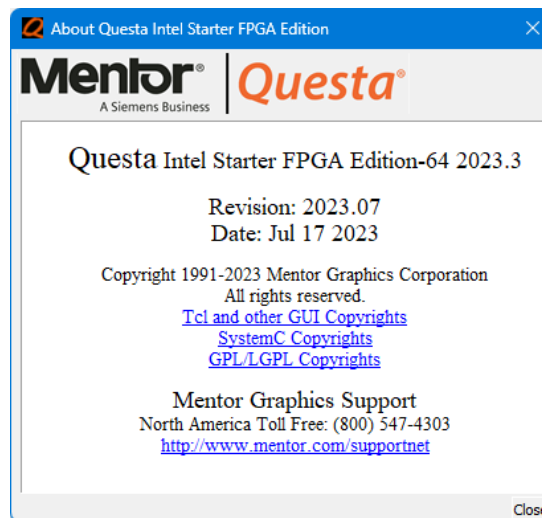
UUT instantiation

Stimulus run



# Questa Overview

- Multi-language HDL simulation environment
- It can be used independently or Quartus can create startup scripts
  
- Two versions of the Questa\* Intel FPGA simulator available:
  - **Questa\* Intel® FPGA Edition**
    - Licenses are required and must be purchased
    - 2-6X slower than Questa Core
  - **Questa\* Intel® FPGA Starter Edition**
    - Licenses are required but are free
    - 40% slower than paid edition
  
- Both the free and paid editions require licenses for performing elaboration and simulation, but not for compilation







# Questa Overview

Digital Logic Design with FPGA

The screenshot displays the Questa! Starter FPGA Edition software interface, showing the simulation environment for a design named "simple\_f\_meter\_tb/UUT/simple\_fsm/c\_state".

**Simulation Structure:** A tree view on the left shows the design hierarchy, including components like "simple\_f\_meter", "UUT", "time\_base\_count...", "freq\_counter", "simple\_fsm", "proc\_fsm", "proc\_memory", "memory", and various "line\_" and "std\_" components.

**FSM View:** A state machine diagram in the center shows states: IDLE, CLEAR, BASE\_ON (highlighted in red), BASE\_OFF, and MEM\_WRITE. Transitions are labeled with conditions like "Cond: 1" and "Cond: ((reset == '1'))".

**Dataflow View:** A block diagram on the right shows the internal dataflow of the "proc\_fsm" block, including signals like "BASE\_ON", "n\_state", "reset", "c\_state", "time\_base\_pulse", "memory\_en", "time\_base\_on", and "time\_base\_rst".

**Waveforms View:** A timing diagram at the bottom right shows the signals "time\_base\_pulse", "c\_state", "memory\_en", "n\_state", "time\_base\_on", and "time\_base\_rst" over time. A cursor is positioned at 41077929928 ps.

**Command/Transcript Window:** A window at the bottom left shows the simulation transcript, including commands like "Loading ieee.std\_logic\_arith(body)", "VSIM4> run 1 us", and "VSIM5> run 1 ms".



# Questa Simulation Types

Simulation Type	Description	Occurs
RTL	Simulation of an RTL design consisting of one or more RTL files. The RTL files can instantiate low level blocks, such as primitives, basic IP functions, and ATOMs.	Can perform before synthesis
Post-Synthesis (Gate-Level)	The Quartus EDA Netlist Writer tool generates the post-synthesis netlist. The post-synthesis netlist is a netlist of low level blocks called ATOMs. The post-synthesis netlist is a purely functional netlist.	Must perform after synthesis
Post-Fit (Gate-Level)	The Quartus EDA Netlist Writer can generate a Verilog HDL or VHDL gate-level netlist after the Fitter stage completes (post-fit netlist). The post-fit netlist is a netlist of ATOMs that the Fitter placed and routed on the FPGA device. The post-fit netlist is a purely functional netlist. <b>Note:</b> The post-fit netlist includes chip locations of ATOM instances in commented lines. The post-synthesis netlist does not include this data.	Must perform after fitting

**Note:** The Quartus Prime software supports post-fit functional simulation, but does not support post-fit timing simulation.



# In-System Sources and Probes (ISSP)

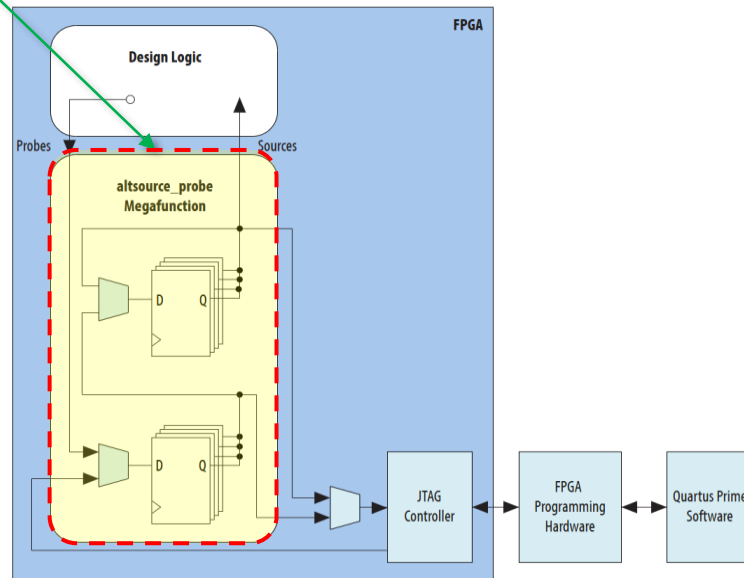
- Hardware and Software Requirements
  - Intel Quartus Prime Lite Edition
  - Download Cable (USB-Blaster download cable or ByteBlaster cable)
  - Intel FPGA development kit or user design board with a JTAG connection to device under test
  
- The In-System Sources and Probes Editor supports the following device families:
  - Arria series
  - Stratix series
  - Cyclone series
  - MAX series
  
- ✓ Quickly set signal to constants: pins or internal nodes
- ✓ Easily monitor signals (no-triggered continuous display)
- ✓ Works on actual hardware
- × Not-triggered – might miss activity!



# ISSP

The ISSP system consists of the ALTSOURCE\_PROBE IP core and an interface to control the ALTSOURCE\_PROBE IP core instances during run time

- ISSP Editor consists of a probe function and interface to control the instances during run-time
- Allows an easy way to drive and sample signals in hardware
- Operates over JTAG
- Each ISSP instance can view/probe up to 512 signals
- Each ISSP instance can drive/source up to 512 signals



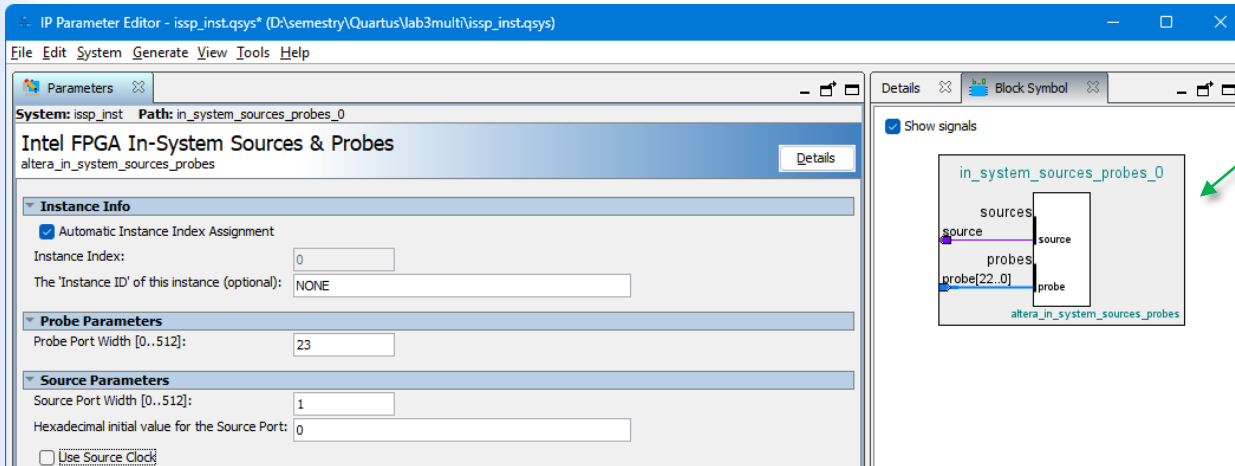
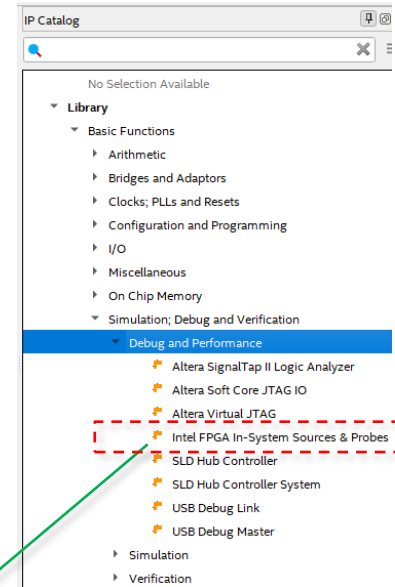
ISSP block diagram



# ISSP

## Using In-System Sources and Probes

- Create ISSP IP instance using the IP Parameter Editor
- Instantiate in design and compile project
- Program target device
- Create and use ISSP Editor (.spf file) to control sources and probes



ISSP IP core setup

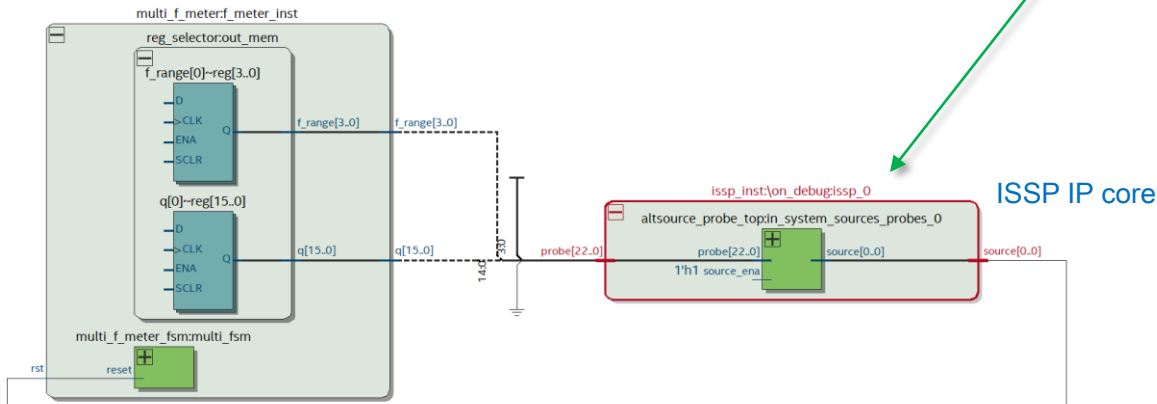


# ISSP

## Using In-System Sources and Probes

- Create ISSP IP instance using the IP Parameter Editor
- [Instantiate in design and compile project](#)
- Program target device
- Create and use ISSP Editor (.spf file) to control sources and probes

```
57 on_debug:  
58   if DEBUG generate  
59     issp_0: issp_inst port map(source => source, probe => q(22 downto 0));  
60     rst <= source(0);  
61   end generate;  
62 no_debug:  
63   if not DEBUG generate  
64     rst <= not key(0);  
65   end generate;
```



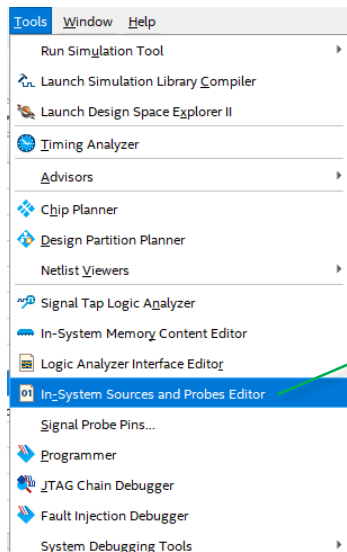


# ISSP

## Using In-System Sources and Probes

- Create ISSP IP instance using the IP Parameter Editor
- Instantiate in design and compile project
- Program target device
- Create and use ISSP Editor (.spf file) to control sources and probes

Digital Logic Design with FPGA



**ISSP manager**

Instance Manager: Ready to acquire

Probe read interval: Current interval: 0 samples per second

Event log: Maximum size: 512

Automatic (selected)

Save data to event log (checked)

Write source data: Continuously

Index	Instance ID	Status	Sources	Probes	Name
0		Not running	1	23	

**Probes/sources**

Index	Type	Alias	Name	Data
P[22..0]			probe[22..0]	2295E3h
P22			probe[22]	0
P21			probe[21]	1
P20			probe[20]	0
P19			probe[19]	0
P18			probe[18]	0
P17			probe[17]	1
P16			probe[16]	0
P15			probe[15]	1
P14			probe[14]	0
P13			probe[13]	0
P12			probe[12]	1
P11			probe[11]	0
P10			probe[10]	1
P9			probe[9]	0

**Live Waveforms View**

JTAG Chain Configuration: JTAG ready

Hardware: USB-Blaster [USB-0]

Device: @1: 10M50DA([ES]/10M50DC)

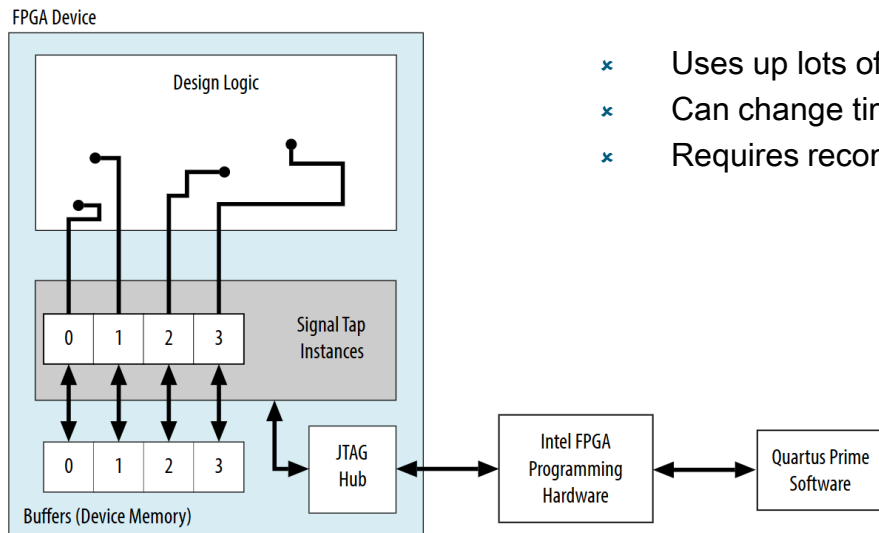
File: try/Quartus/lab3multi/output\_files/multi\_f.sof



# Signal Tap

The **Signal Tap Logic Analyzer** captures and displays real-time signal behavior in an FPGA design using a defined clock signal

- ✓ Easily monitor signals – using simple to elaborate triggering schemes
- ✓ No external equipment required
- ✓ Don't need to figure out stimulus since its based on actual hardware



- ✗ Uses up lots of memory resources inside the FPGA
- ✗ Can change timing of design
- ✗ Requires recompile witch takes time





# Signal Tap

- Hardware and Software Requirements
  - Signal Tap Logic Analyzer (following software includes the Signal Tap):
    - Intel Quartus Prime Design Software
    - Intel Quartus Prime Lite Edition
    - Alternatively, use the Signal Tap [standalone](#) software and standalone Programmer software
  - Download Cable (USB-Blaster download cable or ByteBlaster cable)
  - Intel FPGA development kit or user design board with a JTAG connection to device under test

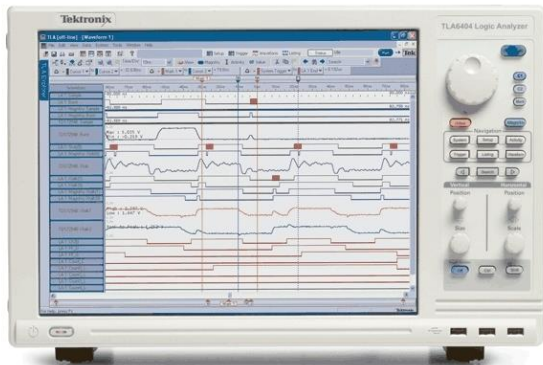
**Note:** During data acquisition, the memory blocks in the device store the captured data, and then transfer the data to the logic analyzer over a JTAG communication cable.

## Opening the Standalone Signal Tap Logic Analyzer GUI

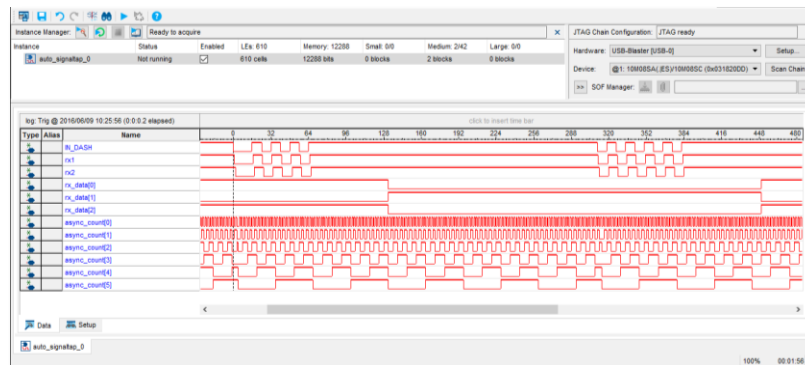
```
quartus_stpw <stp_file.stp>
```



# Signal Tap LA vs. External LA



- System-level debug
- Can store large quantities of data
- Flexible trigger condition

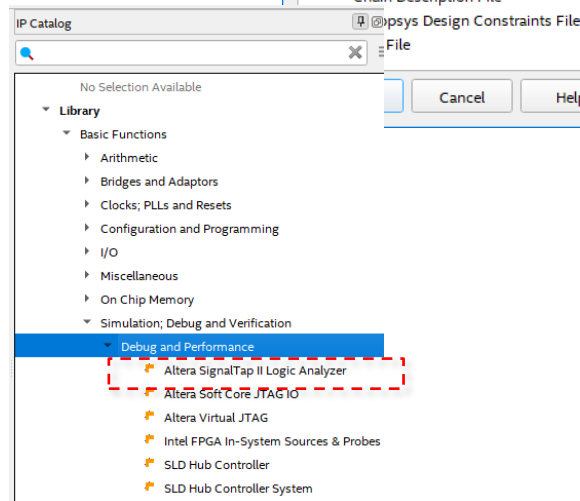
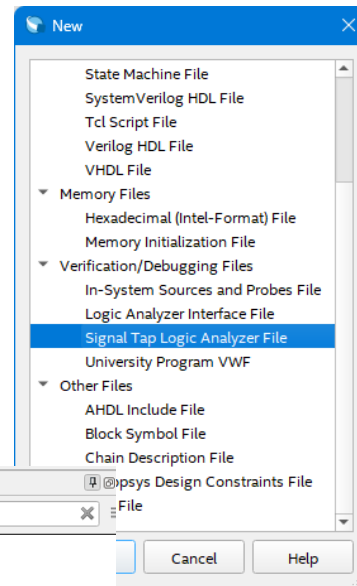


- Tap signals buried deep in the design
- No unassigned I/Os or routing needed
- Comes free with Quartus
- No external test equipment needed
- Tap new signals with the same board by recompiling, reprogramming (no re-spin!)



# Signal Tap

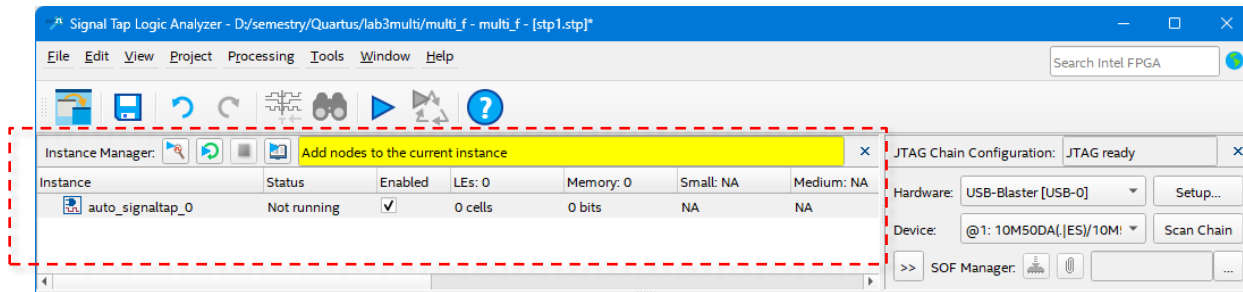
- Create Signal Tap instance
  - Use Signal Tap file (.stp)
    - Creates a file separate from design files
    - Convenient features and GUI
  - Use IP Catalog
    - Manually instantiate IP core directly into HDL code
    - Ties the ELA to the signals directly in RTL





# Signal Tap

- Instance Manager
  - Identifies which instance is being edited in the GUI
  - Enable/disable instances quickly
  - Gives status and resource utilization







# Signal Tap

## Nodes List

- Use the Node Finder to add signals to be tapped
- Automatically groups busses together and create custom groups

The screenshot shows the Signal Tap configuration window for 'auto\_signaltap\_0'. The 'Lock mode' is set to 'Allow all changes'. The 'Node' table lists 12 memory locations, each with a 'Data Enable' checkbox checked, a 'Trigger Enable' checkbox checked, and a 'Trigger Conditions' dropdown set to 'Basic AND'. The 'Signal Configuration' panel on the right shows 'Clock' set to 'adc\_clk\_10', 'Sample depth' at 512, 'RAM type' as 'Auto', 'Nodes Allocated' as 'Auto', and 'Type' as 'Continuous'.

Type	Alias	Name	Data Enable	Trigger Enable	Trigger Conditions
R		...meter_inst reg_selector.out_mem q[0]	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	1 ✓ Basic AND
R		...meter_inst reg_selector.out_mem q[1]	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
R		...meter_inst reg_selector.out_mem q[2]	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
R		...meter_inst reg_selector.out_mem q[3]	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
R		...meter_inst reg_selector.out_mem q[4]	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
R		...meter_inst reg_selector.out_mem q[5]	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
R		...meter_inst reg_selector.out_mem q[6]	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
R		...meter_inst reg_selector.out_mem q[7]	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
R		...meter_inst reg_selector.out_mem q[8]	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
R		...meter_inst reg_selector.out_mem q[9]	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
R		...eter_inst reg_selector.out_mem q[10]	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
R		...eter_inst reg_selector.out_mem q[11]	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	

Node Finder

The Node Finder dialog box shows a search for nodes. The 'Named' field is empty. The 'Matching Nodes' list includes 'q\_tmp-0', 'q\_tmp-2', 'd\_cnr4Aceo|gen:2.cnr', 'ceo', 'Equal0-0', 'q\_tmp', 'q\_tmp[0]', 'q\_tmp[1]', 'q\_tmp[2]', 'q\_tmp[3]', 'q\_tmp[0]-3', 'q\_tmp[2]-1', and 'q\_tmp-0'. The 'Nodes Found' list includes 'multi\_f\_meter\_f\_...\_state.MEM\_WRITE', 'multi\_f\_meter\_f\_...\_fsm|address[2]-1', 'multi\_f\_meter\_f\_...\_fsm|address[3]-2', 'multi\_f\_meter\_f\_...\_fsm|address[3]-3', 'multi\_f\_meter\_f\_m\_n:1.cnr|q\_tmp[0]', 'multi\_f\_meter\_f\_m\_n:1.cnr|q\_tmp[1]', 'multi\_f\_meter\_f\_m\_n:1.cnr|q\_tmp[2]', 'multi\_f\_meter\_f\_m\_n:1.cnr|q\_tmp[3]', 'multi\_f\_meter\_f\_m\_n:2.cnr|q\_tmp[0]', 'multi\_f\_meter\_f\_m\_n:2.cnr|q\_tmp[1]', 'multi\_f\_meter\_f\_m\_n:2.cnr|q\_tmp[2]', and 'multi\_f\_meter\_f\_m\_n:2.cnr|q\_tmp[3]'. The 'q\_tmp' node is highlighted in blue. The 'List' button is visible at the top right.



# Signal Tap

- Trigger Conditions and Qualifiers
  - Data Enable: saves signal data (disable to save memory)
  - Trigger Enable: signal is part of the trigger condition (disable to save LEs)

The screenshot shows the Signal Tap configuration interface. The main window displays a table of nodes with columns for Type, Alias, Name, Data Enable, Trigger Enable, and Trigger Conditions. A red dashed box highlights the Data Enable and Trigger Enable columns. The Signal Configuration panel on the right shows settings for Clock, Data, Segmented, Nodes Allocated, Pipeline Factor, and Storage qualifier.

Type	Alias	Name	Data Enable	Trigger Enable	Trigger Conditions
		...inst reg_selector:out_mem q[0..3]	✓	✓	Ah
		...inst reg_selector:out_mem q[0]	✓	✓	0
		...inst reg_selector:out_mem q[1]	✓	✓	1
		...inst reg_selector:out_mem q[2]	✓	✓	0
		...inst reg_selector:out_mem q[3]	✓	✓	1
		...inst reg_selector:out_mem q[4..7]	✓	✓	Xh
		...meter_inst reg_selector:out_mem q[8]	✓	✓	
		...meter_inst reg_selector:out_mem q[9]	✓	✓	
		...eter_inst reg_selector:out_mem q[10]	✓	✓	
		...eter_inst reg_selector:out_mem q[11]	✓	✓	
		...eter_inst reg_selector:out_mem q[12]	✓	✓	
		...eter_inst reg_selector:out_mem q[13]	✓	✓	
		...eter_inst reg_selector:out_mem q[14]	✓	✓	
		...eter_inst reg_selector:out_mem q[15]	✓	✓	

Signal Configuration:

Clock: adc\_clk\_10

Data

Sample depth: 512 RAM type: Auto

Segmented: 2 256 sample segments

Nodes Allocated:  Auto  Manual: 16

Pipeline Factor: 1

Storage qualifier:

Type: Input port

Input port: auto\_stp\_external\_storage\_qualifier

Nodes Allocated:  Auto  Manual: 16



# Signal Tap

- Trigger Conditions
  - Add up to 10 trigger conditions
  - Choose how every node is compared
  - Choose what action triggers a specific node

Type	Alias	Name	Data Enable	Trigger Enable	Trigger Conditions
		...inst reg_selector.out_mem q[0..3]	✓	✓	1 ✓ Basic AND
		...inst reg_selector.out_mem q[0]	✓	✓	0
		...inst reg_selector.out_mem q[1]	✓	✓	1
		...inst reg_selector.out_mem q[2]	✓	✓	0
		...inst reg_selector.out_mem q[3]	✓	✓	1
		...inst reg_selector.out_mem q[4..7]	✓	✓	Xh
		...meter_inst reg_selector.out_mem q[8]	✓	✓	
		...meter_inst reg_selector.out_mem q[9]	✓	✓	
		...eter_inst reg_selector.out_mem q[10]	✓	✓	
		...eter_inst reg_selector.out_mem q[11]	✓	✓	
		...eter_inst reg_selector.out_mem q[12]	✓	✓	
		...eter_inst reg_selector.out_mem q[13]	✓	✓	
		...eter_inst reg_selector.out_mem q[14]	✓	✓	
		...eter_inst reg_selector.out_mem q[15]	✓	✓	





# Signal Tap

- Signal Configuration
  - Advanced trigger control
  - Select the number of trigger conditions
  - Trigger In/Out options

Type	Alias	Name
		...inst reg_selector.out_mem q[0..3]
		...inst reg_selector.out_mem q[0]
		...inst reg_selector.out_mem q[1]
		...inst reg_selector.out_mem q[2]
		...inst reg_selector.out_mem q[3]
		...inst reg_selector.out_mem q[4..7]
		...meter_inst reg_selector.out_mem q[8]
		...meter_inst reg_selector.out_mem q[9]
		...eter_inst reg_selector.out_mem q[10]
		...eter_inst reg_selector.out_mem q[11]
		...eter_inst reg_selector.out_mem q[12]
		...eter_inst reg_selector.out_mem q[13]
		...eter_inst reg_selector.out_mem q[14]
		...eter_inst reg_selector.out_mem q[15]

Hierarchy Display:

- multi\_f
  - multi\_f\_meter:f\_meter\_inst
    - reg\_selectorout\_mem

Signal Configuration:

Clock:

Data

Sample depth:  RAM type:

Segmented: 2 256 sample segments

Nodes Allocated:  Auto  Manual:

Pipeline Factor:

Storage qualifier:

Type:

Input port:

Nodes Allocated:  Auto  Manual:

Record data discontinuities

Disable storage qualifier

Trigger

Nodes Allocated:  Auto  Manual:

Trigger flow control:

Trigger position:

Trigger conditions:

Trigger in

Pin:

Node:

Instance:

Hard Processor System (HPS) trigger out

Pattern:

Trigger out

Pin:

Instance:

Hard Processor System (HPS) trigger in

Hard Processor System (HPS) event:

Level:

Latency delay:



# Signal Tap

- Data/Setup Window
  - Setup allows configuration of nodes and trigger condition
  - Data shows the acquired signal information

log: Trig @ 2024/05/12 15:47:51 (0:0:0.1 elapsed) click to insert time bar

Type	Alias	Name	-64	-32	0	32	64	96	128	160	192	224	256	288	320	352	384	416	448	
		...inst reg_selector:out_mem q[0..3]																		
		...inst reg_selector:out_mem q[4..7]																		
		...inst reg_selector:out_mem q[8..11]																		
		...st reg_selector:out_mem q[12..15]																		
*		...inst reg_selector:out_mem q[12]																		
*		...inst reg_selector:out_mem q[13]																		
*		...inst reg_selector:out_mem q[14]																		
*		...inst reg_selector:out_mem q[15]																		

**Data** **Setup**

Hierarchy Display: x

- multi\_f
  - multi\_f\_meter:f\_meter\_inst
    - reg\_selector:out\_mem

auto\_sigtap\_0 0% 00:00:00