FPGA Implementation Validation and Debug

Rohit Goel
FPGA Solutions R&D
Systems Design Division
Mentor Graphics

Co Authors:

Rakesh Jain
FPGA Solutions Marketing
Systems Design Division
Mentor Graphics

Aman Rana
FPGA Solutions R&D
Systems Design Division
Mentor Graphics

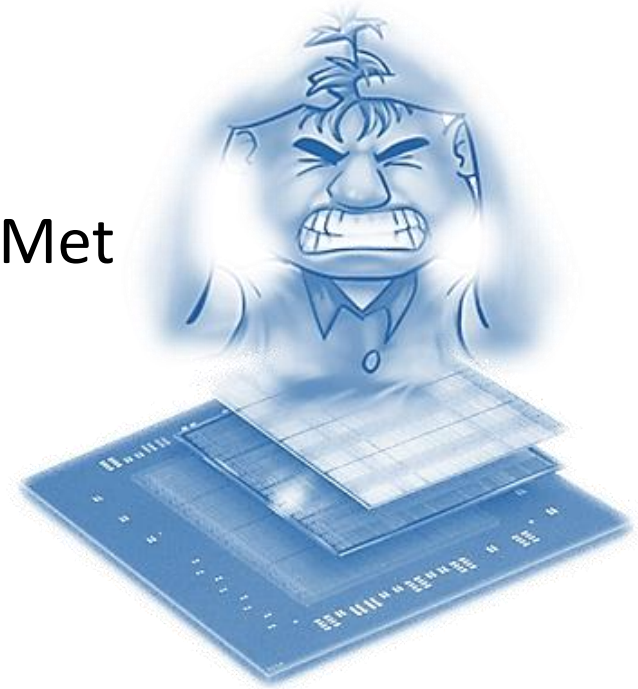
Ankit Goel
FPGA Solutions R&D
Systems Design Division
Mentor Graphics

AGENDA

- **Validation Challenge**
- Existing Methodology
- Improved Methodology
- Design for Validation - Assisted Debug Flow
- Introduction to *Precise-Validate*

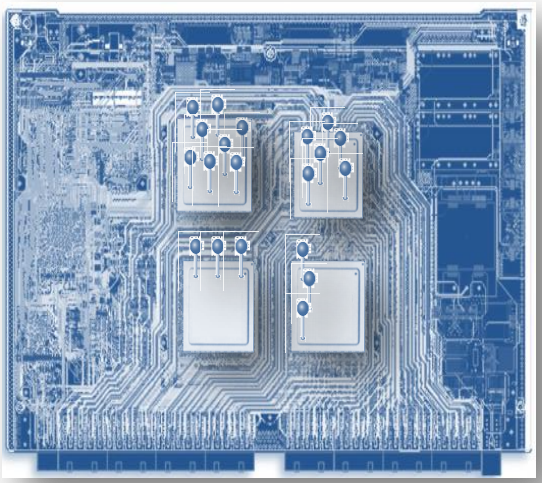
What is a Successful FPGA Design?

- Large Complex FPGA Designs ⇔ NOT Push-Button
  - Extensive Verification Cycle
  - Multiple Implementation Iterations
- Performance and Area Requirements Met
  - Now What?
- System Integration Testing
  - What if the design does not work?
    - Where to start looking?
    - How to find the problem?

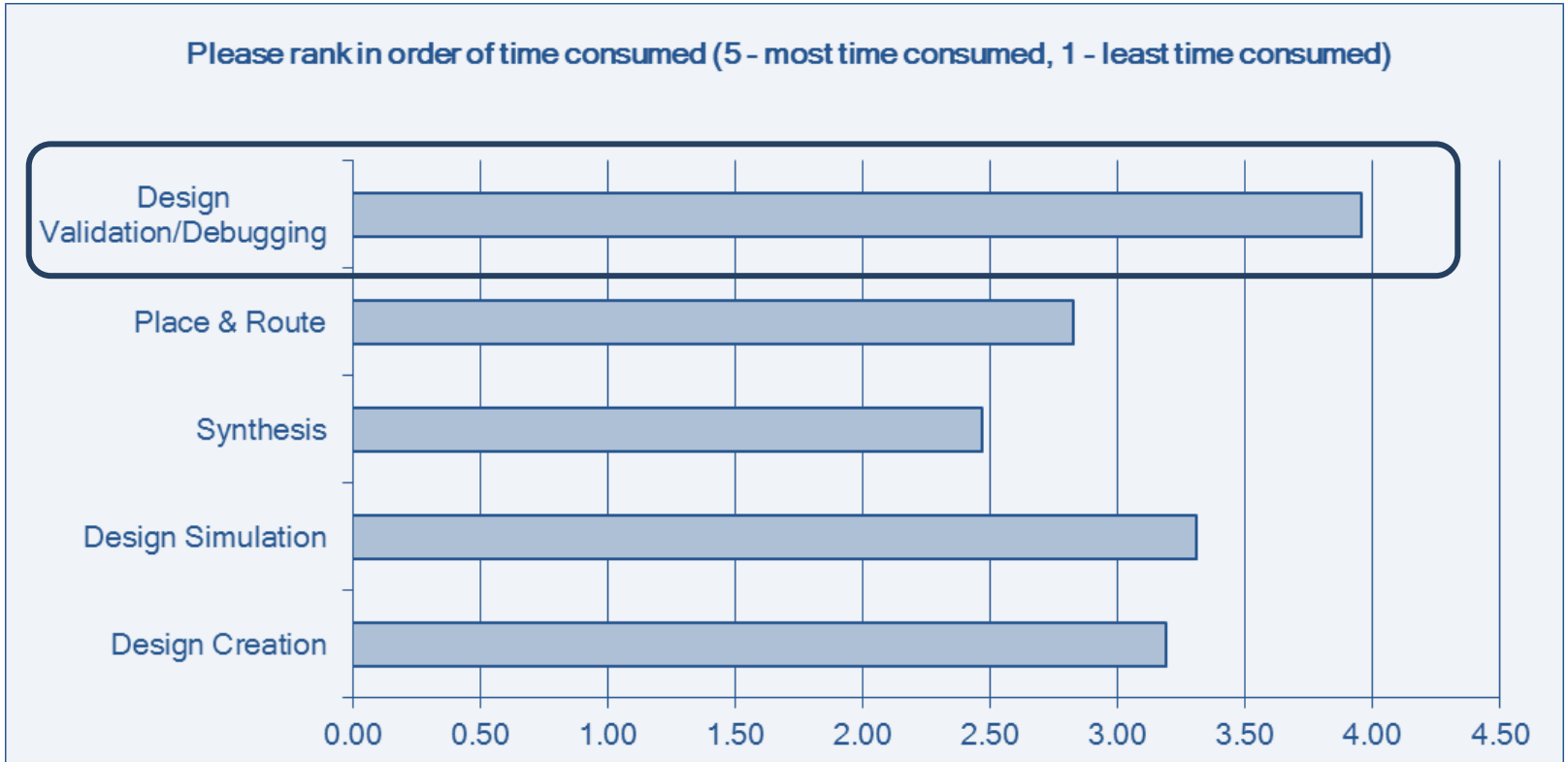


System Validation

- Challenge
  - Quickly & easily instrument validation structures to resolve system design issues in large complex FPGA
- Solution
  - Instrument Validation IP as part of Design
  - In-system FPGA Validation Framework
  - Design for validation
    - Probe for a variety of issues
      - Functional issues
      - Performance issues
      - FPGA-PCB integration issues….
    - Fast iteration cycles

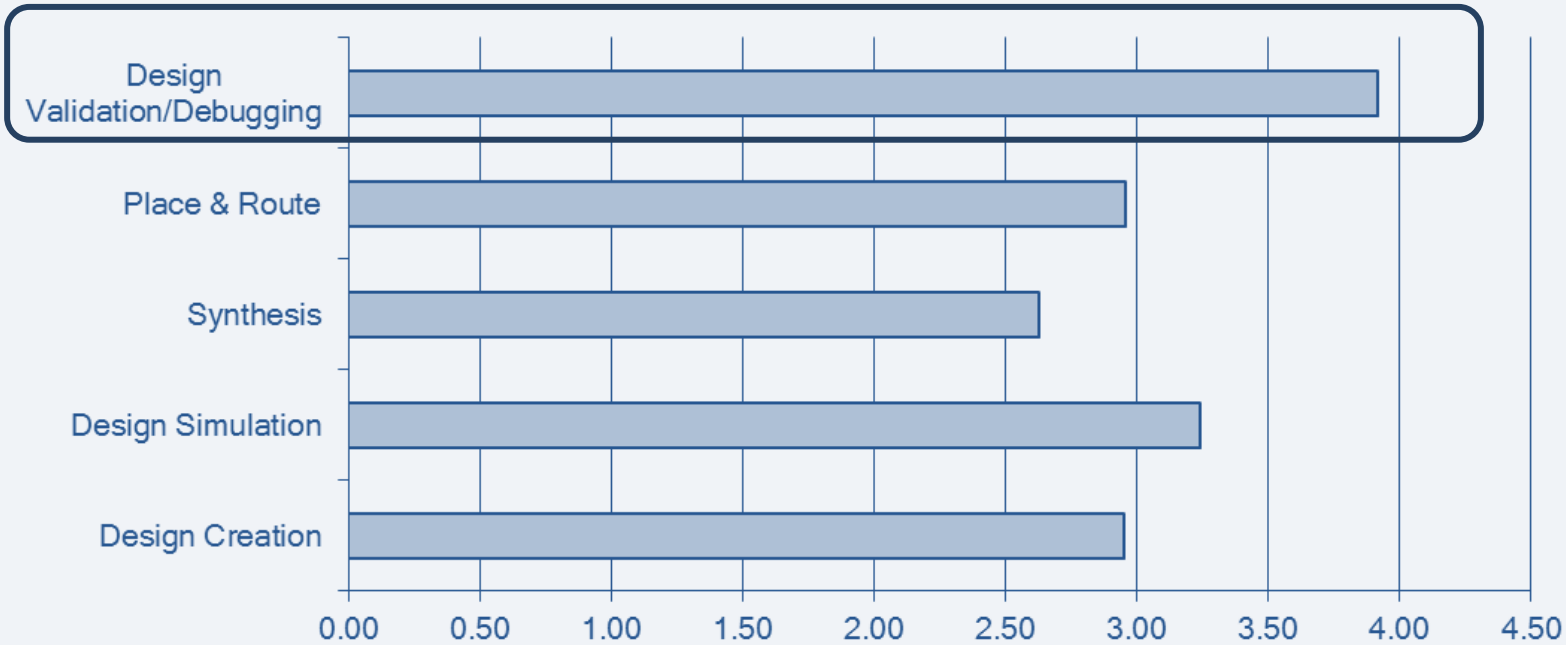


Considerable Time Spent on Debugging



Source: Tech Focus Media/Mentor Graphics FPGA Survey

Difficulty or Challenge Ranking



Source: Tech Focus Media/Mentor Graphics FPGA Survey

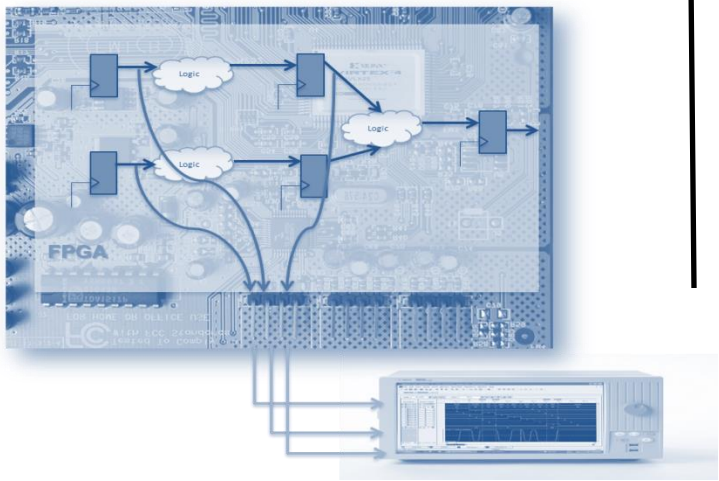
AGENDA

- Validation Challenge
- **Existing Methodology**
- Improved Methodology
- Design for Validation - Assisted Debug Flow
- Introduction to *Precise-Validate*

Existing Methodology

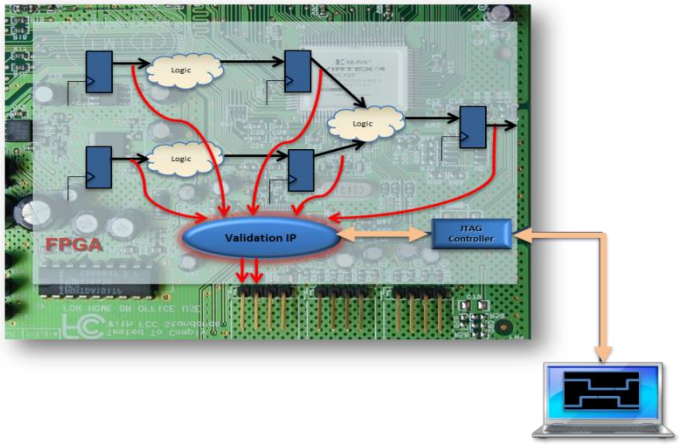
Traditional FPGA Debug

- Using Oscilloscope
- Internal Signals are pulled to I/O
- Signals are monitored using oscilloscope

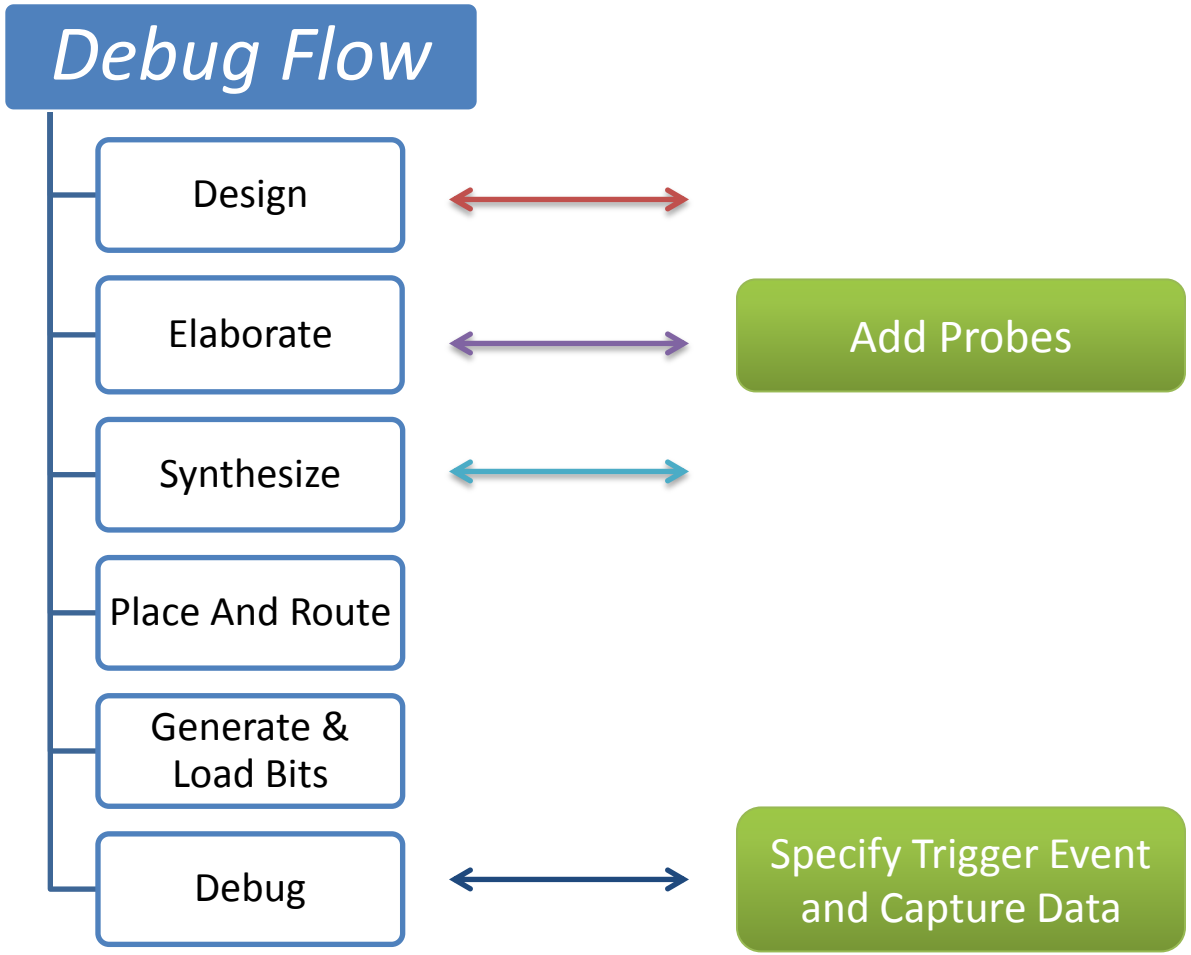


Existing methodology

- Using Desktop/Laptop
- Using Validation IP
- Internal Signals are connected to validation IP
- JTAG is used to interface with Validation IP



Existing Methodology – Broader View



Shortcoming of Existing Methodology

- Lack of Seamless Push-Button Flow
- Difficulty in Specifying hardware event
- Lack of Vendor Independence
- Limited FPGA resources

Lack of Seamless Flow

- Changes the HDL (Not for all methodologies)
  - Manual change in HDL is required
  - Desired signals have to be pulled out for connection
- Limited signal visibility
  - Generate statements
- Some tools exist to automate this
  - Automate pulling out of signals and connection
  - You can also write scripts to do it
  - However issue with such approach is language support

Difficulty in Specifying H/W Event

- What is an Event ?
  - An event in hardware is a point where a potential issue might occur
- How can I specify Event ?
  - Every tool has its own way of specifying Event
- Can I correlate with Hardware cycles ?
  - No, because every Event is seen as an isolated event
  - No way of correlating even if multiple events are specified using separate validation IP

Limited FPGA Resources

- Limited resources
  - Slices, Flops and RAMs
- Number of FPGA I/O Pins are also limited
  - Only a handful of pins are available as GPIO on board
  - It means only a small set of signals can be seen at a given time hence more iterations
- Can degrade design
  - Routing internal signals to validation IP or I/O pins
  - Routing signal from different part of FPGA to a common block

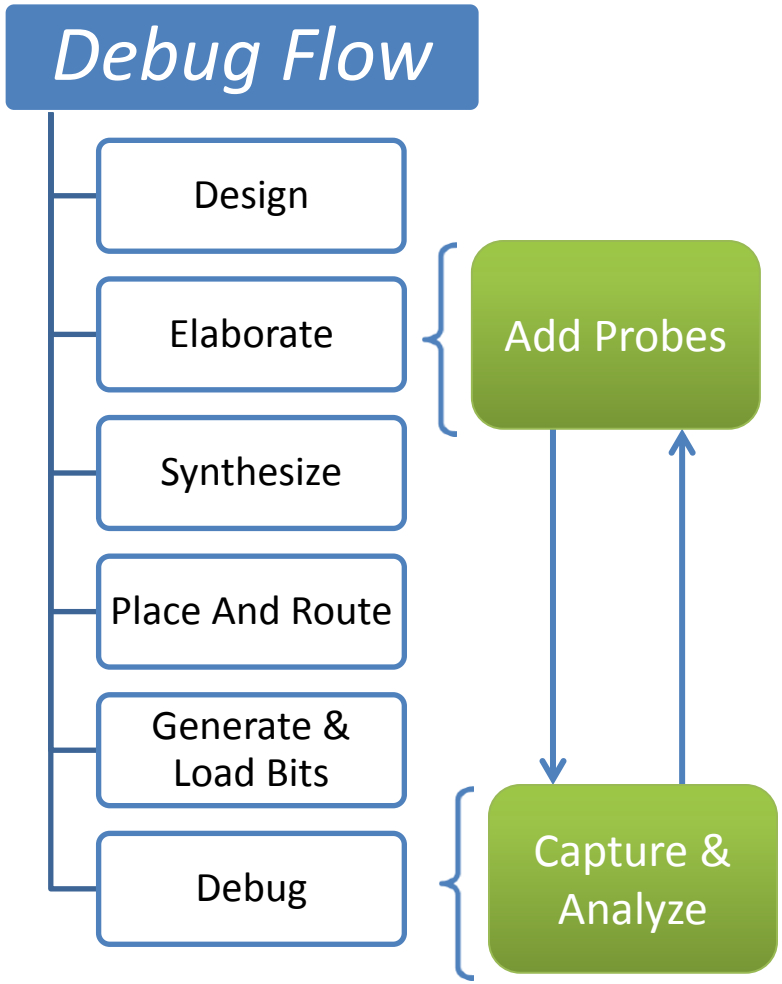
Lack of Vendor Independence

- Solutions commonly used today are vendor dependent
- Requires training and expertise development on different flows
- Every solution has a different flow
  - Adding Signals for Probing
  - Specifying Validation IP behavior
  - Specifying and capturing trigger Events

AGENDA

- Validation Challenge
- Existing Methodology
- **Improved Methodology**
- Design for Validation - Assisted Debug Flow
- Introduction to *Precise-Validate*

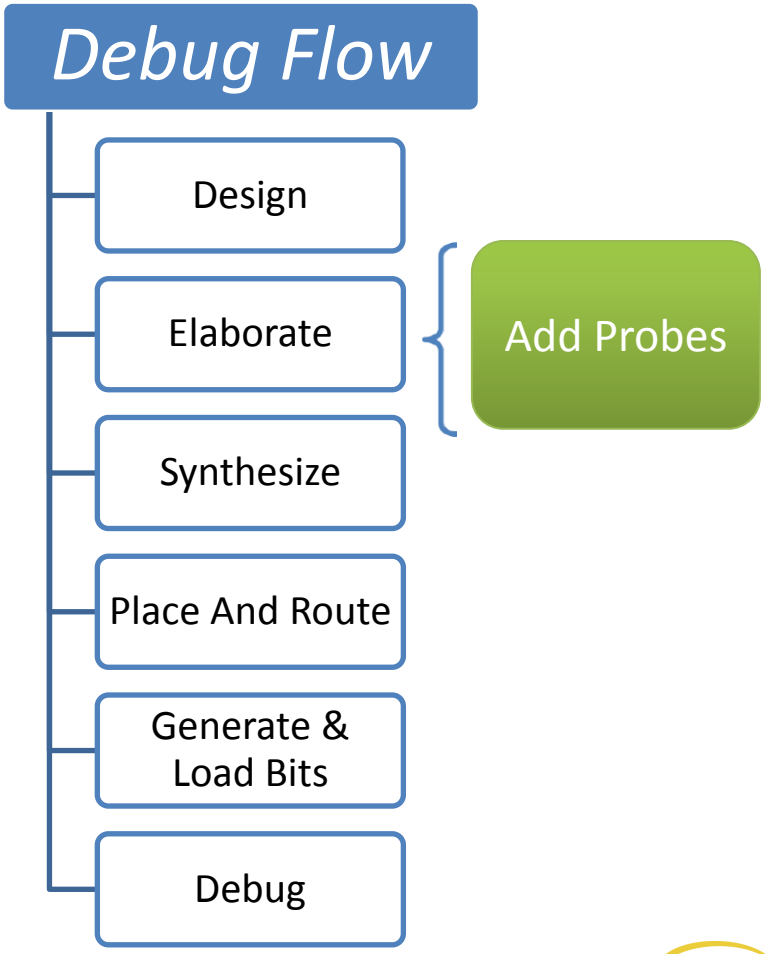
Improved Methodology



- Debug flow is tightly integrated with synthesis tool
- Leverages synthesis tool capabilities

Seamless Flow

- Signals are selected after elaborate stage
  - Names are preserved or similar to HDL names
  - Gives almost 100% visibility
- No HDL change
  - Signals selected for probing are connected during Synthesis



Easy Specification of H/W Event

- Allows user to write expression like normal HDL expression

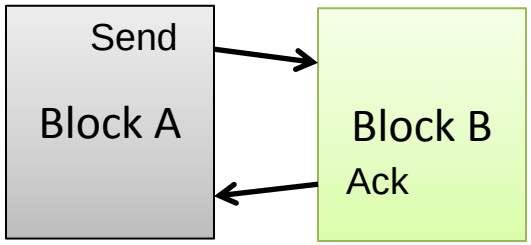
```
(signal_a == 1'b1 || signal_b == 2'b00) && signal_c == 3'b101
```

  - Possible by using the language parser of synthesis tool
  - Expression written in Verilog specific format
- Allows user to write expression using actual signal name
- Ease of use
  - Helps user in writing better expression suited to his needs

Easy Specification of H/W Event (cont'd)

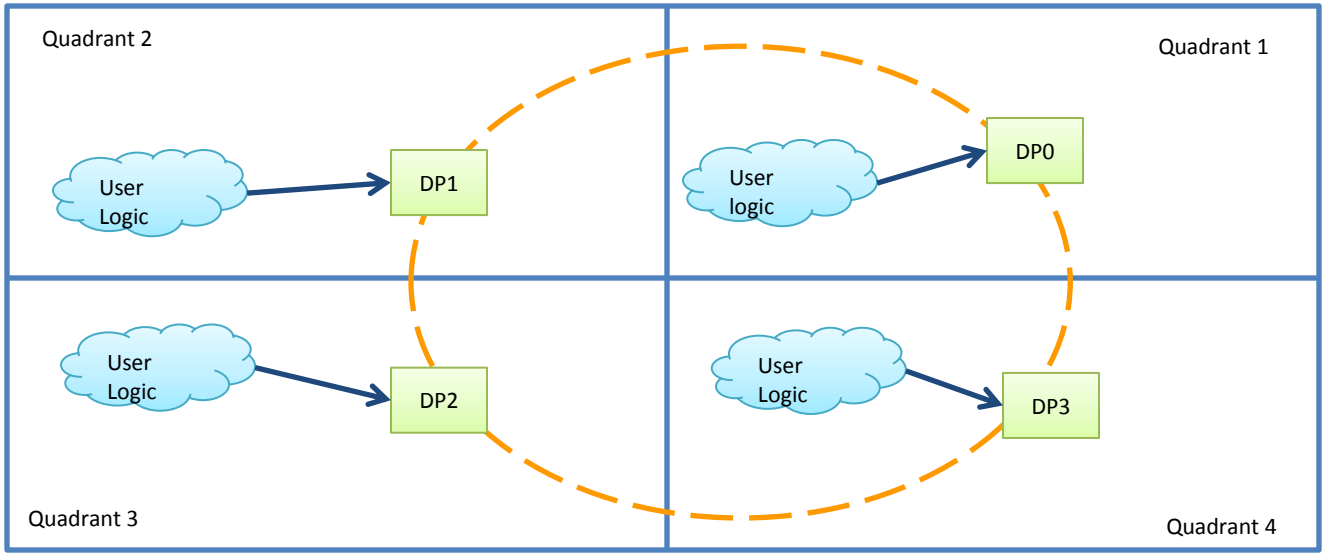
- Correlation of events with actual design run-time.
  - Events will no longer be treated as isolated events
  - Tells exact design time or clock count at which a particular events occur
  - A simple yet powerful attribute which can help in lot of different ways
  - Example usage
    - A simple handshaking Protocol

Event 1 : Block A transmit Send
Event 2 : Block A receives Ack



Efficient use of FPGA Resources

- Using localize debug points.
  - Signals lying on different quadrant of FPGA should not be clubbed
  - Validation IP pipelined to make sure design timing is not effected



Efficient use of FPGA Resources (cont'd)

- Highly optimized customizable monitors
  - More flexibility to control area utilization

Example 1

i. signal_A == 3'b1

ii. (signal_a == 1'b1 || signal_b == 2'b00) && signal_c == 3'b101

Example 2

i. signal_a == 1'b1 || signal_b == 2'b00)

ii. signal_c >= 2'b001 || signal_d <= 4'b1010

- Flexible use of RAM
  - No limitation on RAM size
  - Use of RAM is optional

Vendor Independence

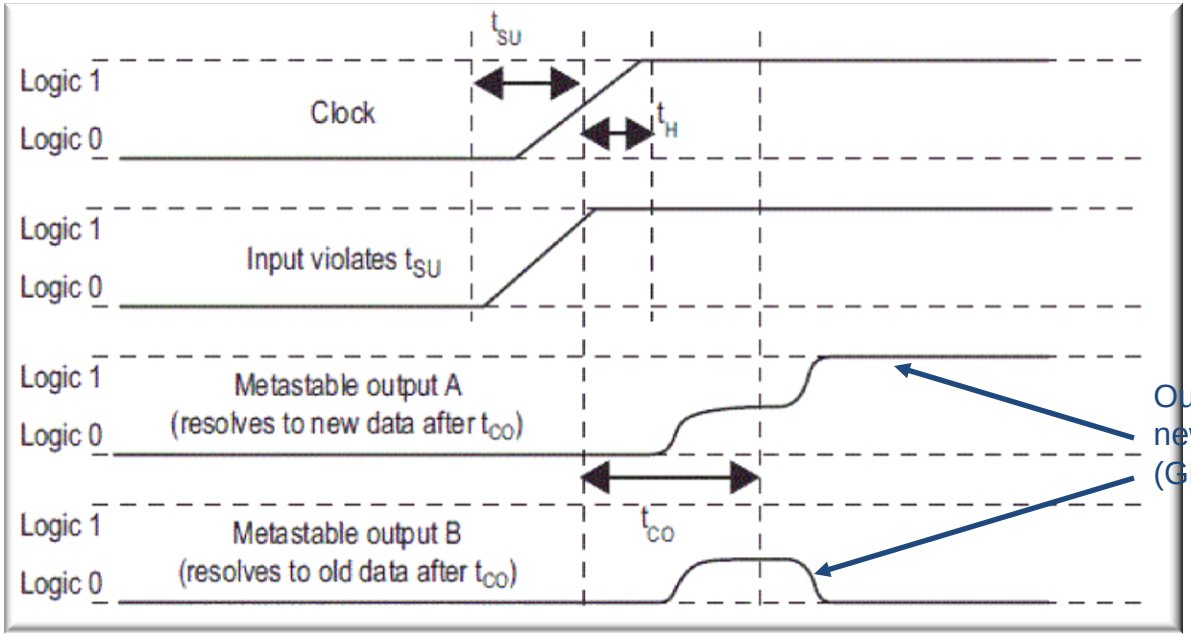
- Validation IP is common for all vendors
- Using synthesis tool for instrumentation
  - Flow remains same for different vendors

Metastability

- Motivation
  - Growing number of complex multi-clock domain designs finding their way into FPGAs
  - Data transfer across different clock domains (CDC) requires careful handling at RTL level to avoid metastability issues
  - Improper or insufficient timing constraints during implementation may also result in timing violations on board
  - Functional issues due to metastability appear totally random and hence are hard to debug

Metastability

- Typical output of a metastable Flip-Flop

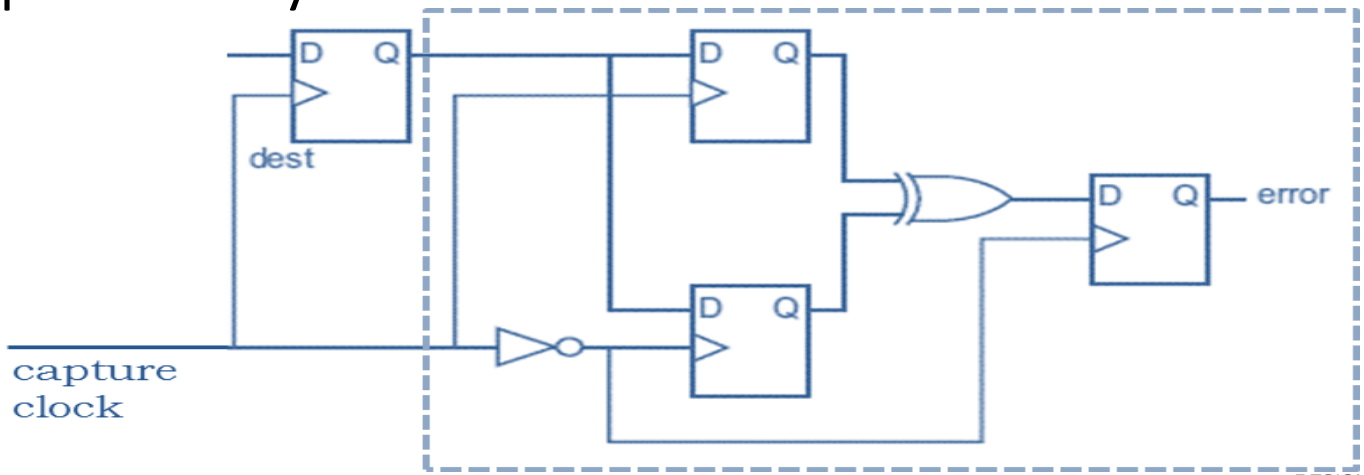


Output may settle to either new state or old state (Glitch)

* Diagram from Altera White-paper

Metastability detector

- Existing solutions
  - Intuitive circuit that compares output at negative edge and next positive edge
  - Ineffective for scenarios where metastable output settles before negative edge
  - So probability of catching metastability is around ½ approximately



Proposed Metastability Detector

- Efficiency is independent of clock period
- All case where metastable output settles after negative edge of clock is detected
- Probability of catching metastable state when output settles before negative edge is high which is null for existing solutions
- Hence overall probability will be higher as compared to ½ for existing solutions

AGENDA

- Validation Challenge
- Existing Methodology
- Improved Methodology
- **Design for Validation - Assisted Debug Flow**
- Introduction to *Precise-Validate*

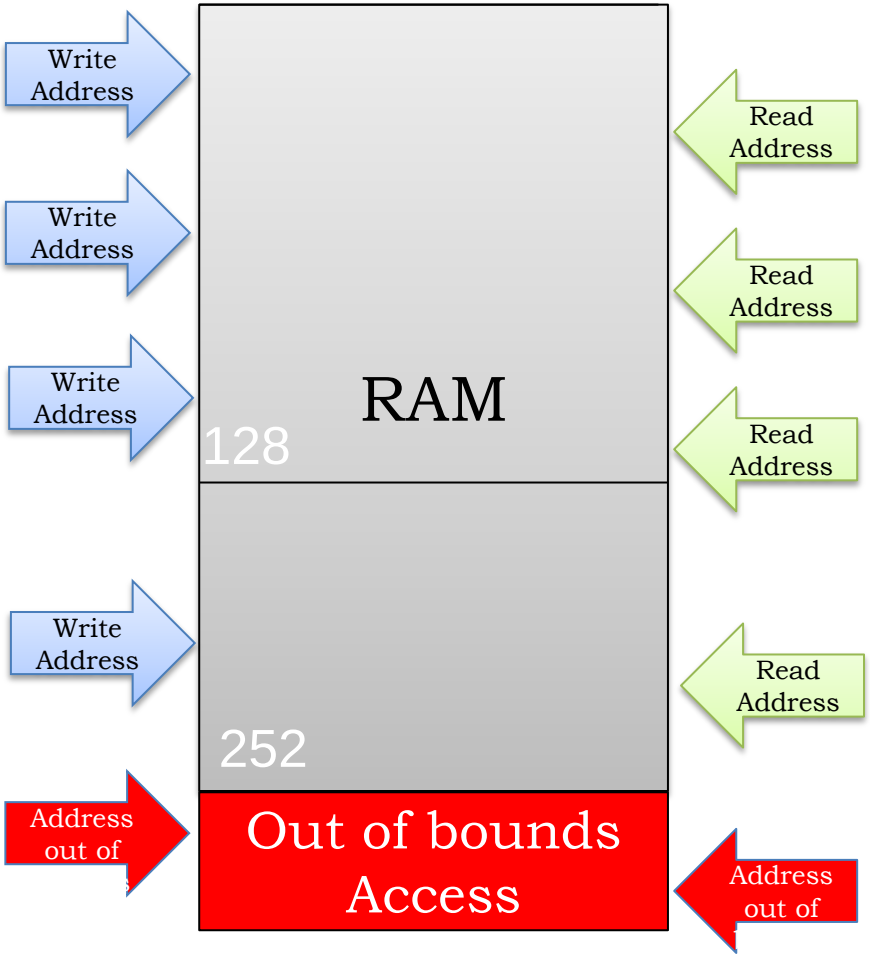
Design Validation – Assisted Debug Flow

- An unexplored area for FPGA validation
- Automatic identification of design issue areas
- Issues are sometime common across different design
  - If common issues are ironed out earlier design cycle get reduce
- Automatic instrumentation of validation IP for such issues
- Validates not only functionality but also quality of design

Assisted Debug Flow (cont'd)

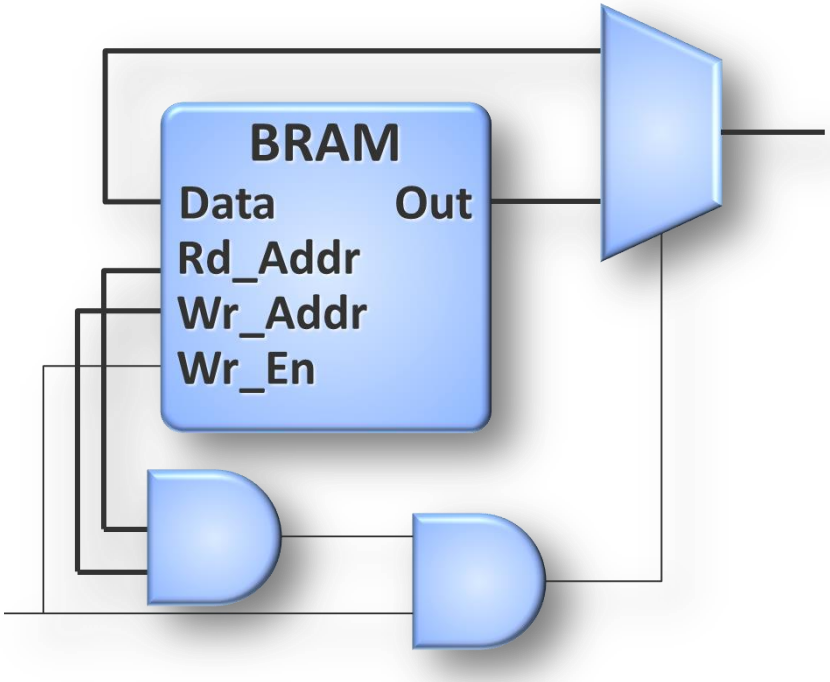
RAM Array bound

- RAM
  - Extensively used in the designs.
- RTL example
  - reg [7:0]RAM [0:252]
  - rd_addr [7:0]
  - wr_addr[7:0]
- Invalid data access



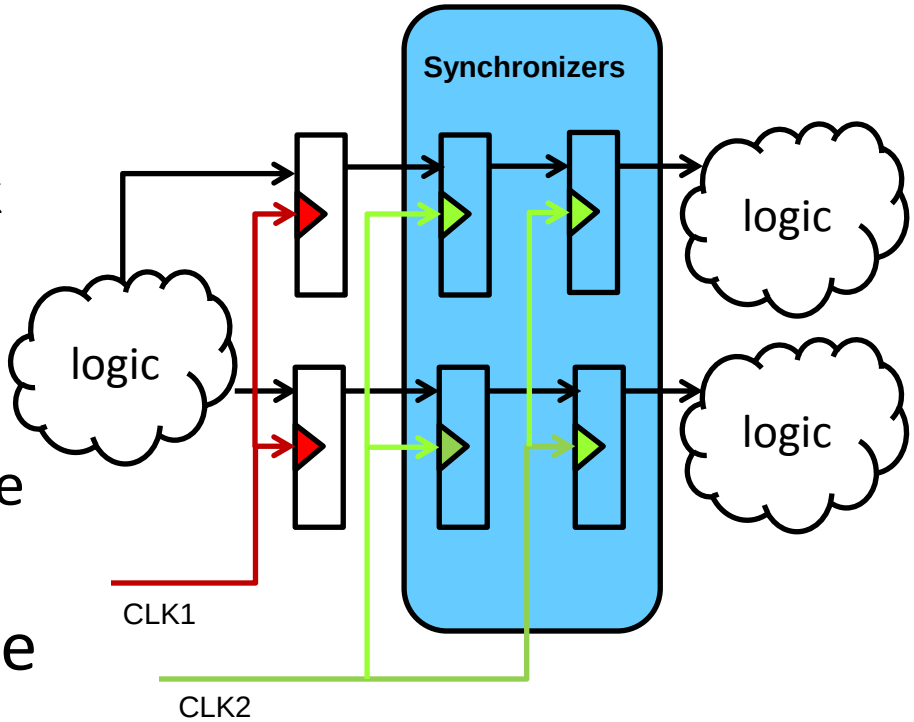
Assisted Debug Flow (cont'd)

- RAM Rd/Wr Collision
  - FPGA architecture does not guarantee proper functionality
  - Synthesis adds logic to insure proper functionality
    - Logic adds extra area
  - Use instrumentation to prove functionality without extra logic



Assisted Debug Flow (cont'd)

- Clock domain crossing
  - Signal transfer across clock domains requires proper synchronizers to avoid metastability
  - 2 or 3 flop synchronizers are commonly used
- All N-flop synchronizers are automatically detected
  - Metastability is checked at output of synchronizer



Assisted Debug Flow (cont'd)

- Auto identification of common design issues
  - Array bound check
  - Ram Rd/Wr Collision
  - Metastability detector across clock domain crossing
- Automating the flow of instrumenting validation IP
  - All signals which are essential for design issues are automatically added as probe
  - All connections are made in memory and validation IP becomes part of design

AGENDA

- Validation Challenge
- Existing Methodology
- Improved Methodology
- Design for Validation - Assisted Debug Flow
- **Introduction to *Precise-Validate***

Introduction to *Precise-Validate*

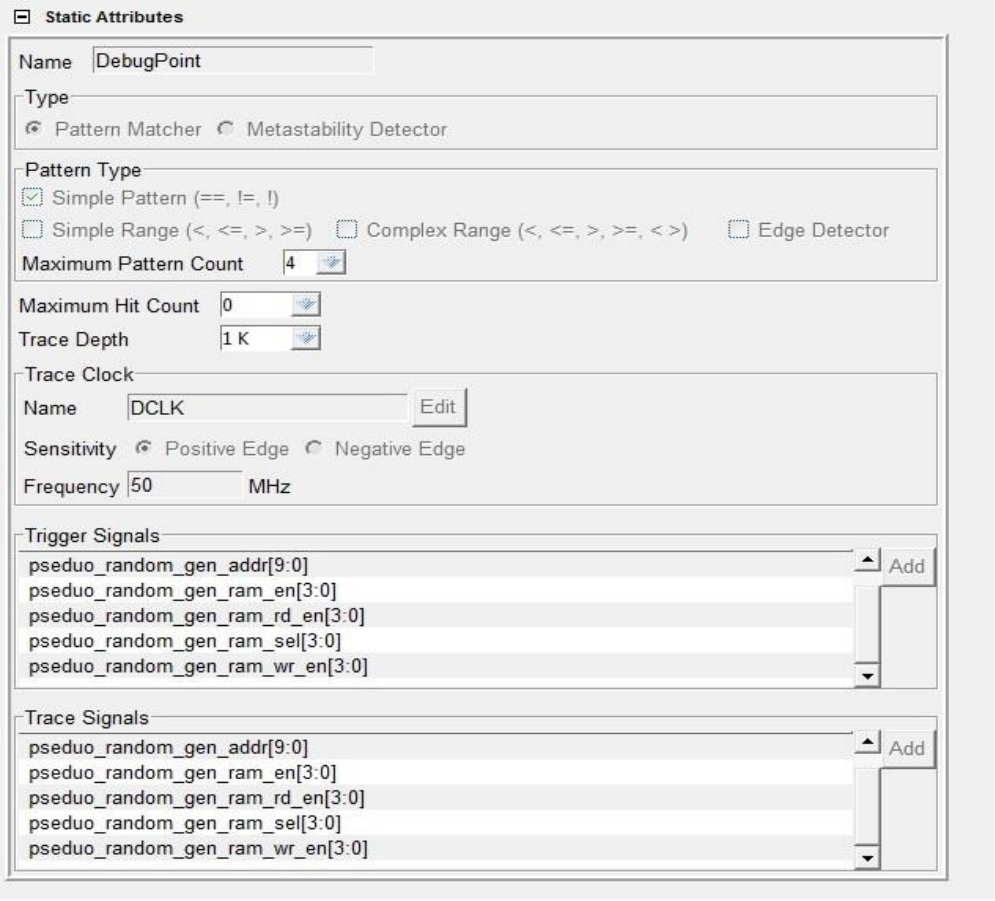
- Vendor independent FPGA validation
- Seamless push-button instrumentation
  - No HDL modifications
- System speed FPGA validation
  - Real clock speed to catch real errors
- Dynamic trigger expression modification
- Virtually unlimited visibility
- User defined probe
- Automatic probe

Probe Points

Implement	Name	Type	Status
Yes	Probe_1	PM	Disabled
No	Probe_2	PM	N/A
Yes	Probe_3	PM	Enabled
Yes	Probe_4	PM	Enabled
Yes	Probe_5	PM	Enabled
No	Probe_6	PM	N/A
No	Probe_7	PM	N/A
Yes	Probe_8	PM	Enabled
Yes	Probe_9	PM	Enabled

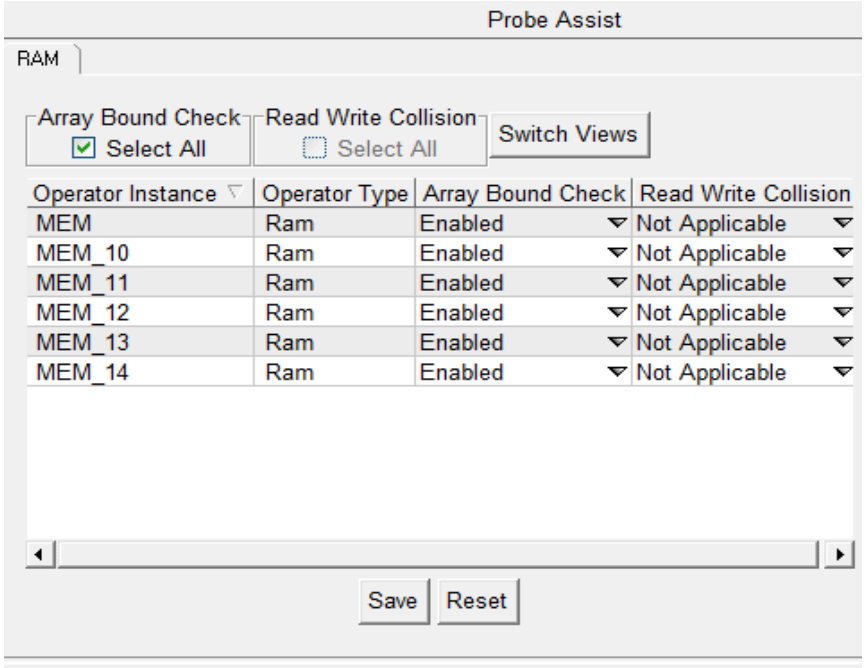
User Defined Probe Instrumentation

- Customizable monitors
- Pipelined architecture
- At-speed
- Pattern matcher
  - Simple Pattern
  - Simple Range
  - Complex Range
  - Edge Detector
- Metastability detector



Automatic Probe Instrumentation

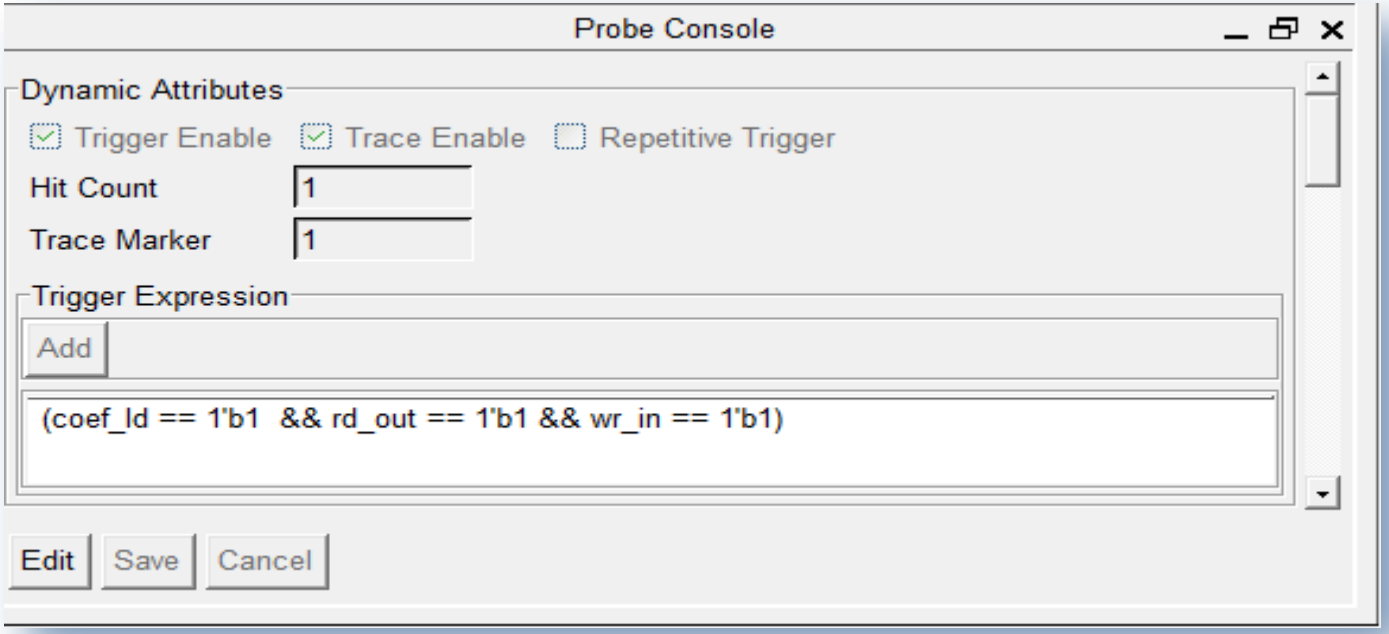
- Probe points are automatically detected
- Controlled instrumentation
- Probe
  - For functionality
  - To validate performance



Operator Instance	Operator Type	Array Bound Check	Read Write Collision
MEM	Ram	Enabled	Not Applicable
MEM_10	Ram	Enabled	Not Applicable
MEM_11	Ram	Enabled	Not Applicable
MEM_12	Ram	Enabled	Not Applicable
MEM_13	Ram	Enabled	Not Applicable
MEM_14	Ram	Enabled	Not Applicable

Dynamic Trigger Expression Configuration

- Re-configure trigger expression without re-implementation
  - Re-program FPGA through JTAG



Trigger Monitor

- Gives bird-eye view for trigger status
- Provides multiple information for every debug point
  - Status
  - Triggers captured
  - Trigger time

Trigger Monitor

Time Unit ns

Debug Points

Name	Type	Status	Trigger(s) Captured	First Trigger Time	Last Trigger Time	First Trigger Clk Count	Last Trigger Clk Count
ram_1_inst.ram_3.MEM	RAM_AUTO	Idle	2	21588045900	21592417010	NA	NA
ram_0_inst.ram_3.MEM	RAM_AUTO	Idle	1	21585604610	21585604610	NA	NA
Training	PM	Idle	1	43163332620	43163332620	2158166631	2158166631

Trigger Search Criteria

Trigger(s) Count Time Clock Count

Start Value 1 End Value 1

View Waveform

Strategy

Name	Status	Trigger(s) Captured	First Trigger Time	Last Trigger Time
Array Bound Check	Idle	1	21592417010	21592417010
Read Write Collision	Idle	1	21588045900	21588045900

Q&A

- Please email *Precise-Validate_mktg@mentor.com* for any queries on methodologies/product.