Unique Methodology to Streamline the Checking of Design Tie-offs

Varun Ramesh/Amol Bhinge,
NXP Semiconductors

Jay Dutt,
Synopsys

September 29, 2016
SNUG Austin

Agenda

- Our Verification Challenge – Design Tie-offs

  Why explore Certitude?

  Introducing Certitude

  Introducing – Our Flow - Expected / Unexpected status of faults

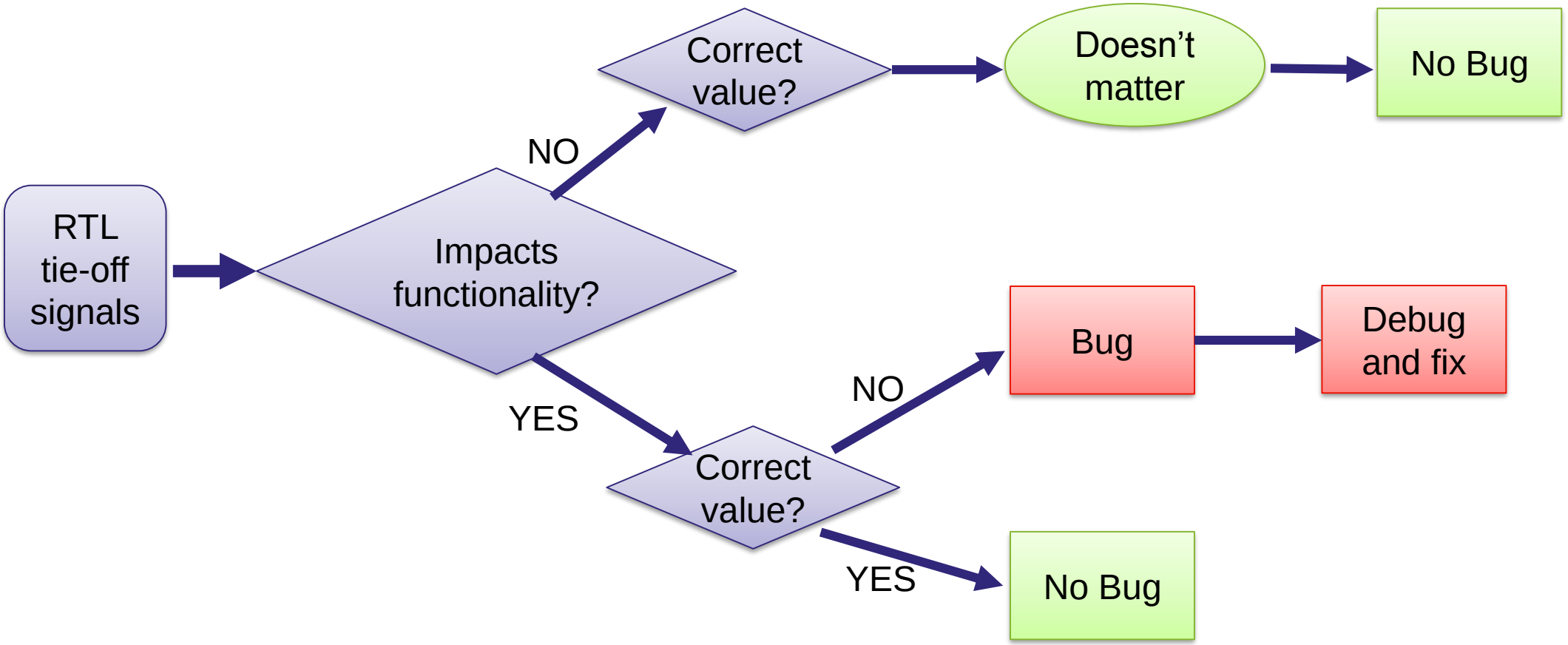
  Analysis of Unexpected faults

  Limitations and Enhancements Requested

  Conclusion and Q&A

Our Verification Challenge – Design Tie-Off

- Tied-off signals may lead to functional bugs if tied to the wrong value



Our Verification Challenge

- Confident that there are no wrong values?

- URG report – gets us a list of all the tie-offs -- Unreachable
- Traditional Method 1: Manually review against the design specification
  - But since manual, reviewing mistakes can occur

- Traditional Method 2: Regressions/testcases are expected to fail if any of the unreachables are tied to wrong value
  - Verification environment robust enough to detect the wrong values and fail the testcases?
  - Testcases focus mainly on connectivity and overall functionality – not tie-offs

- Neither of these methods can provide the high confidence that we need

Agenda

Our Verification Challenge – Design Tie-offs

- Why explore Certitude?

Introducing Certitude

Introducing – Our Flow - Expected / Unexpected status of faults

Analysis of Unexpected faults

Limitations and Enhancements Requested

Conclusion and Q&A

Why we explored Certitude

- Certitude's most important feature – signal faulting
  - Applies to Unreachable signals? - Yes

- "BitNegation" fault category makes most sense
  - Since the signals are tied to constant values, faulting ➔ negating
  - Rarely used category among the port faults
- Certitude lets us run our regression suite on the fault-instrumented design
- The fault report helps us identify if the faulting was useful -
  - Detected: At least one test failed
  - Non-Detected: The fault propagated to a Primary Output , all tests passed
  - Non-Propagated: The fault did not propagate to a primary output, all tests passed
- Just need to match the Certitude results against our "expectations"

Why we explored Certitude (cont'd)

- Already has a proven flow in our verification methodology
- Uses testcases from our existing regression
  - Saves time of writing new testcases
- Certitude report clearly indicate checker weaknesses which results in fixes to our verification environment
- Verdi integration automatically shows waveform comparison between faulty and reference simulations on relevant signals
  - Allows faster root causing for checker weaknesses
- Provides an objective measure of overall verification robustness

Agenda

Our Verification Challenge – Design Tie-offs

Why explore Certitude?

- Introducing Certitude

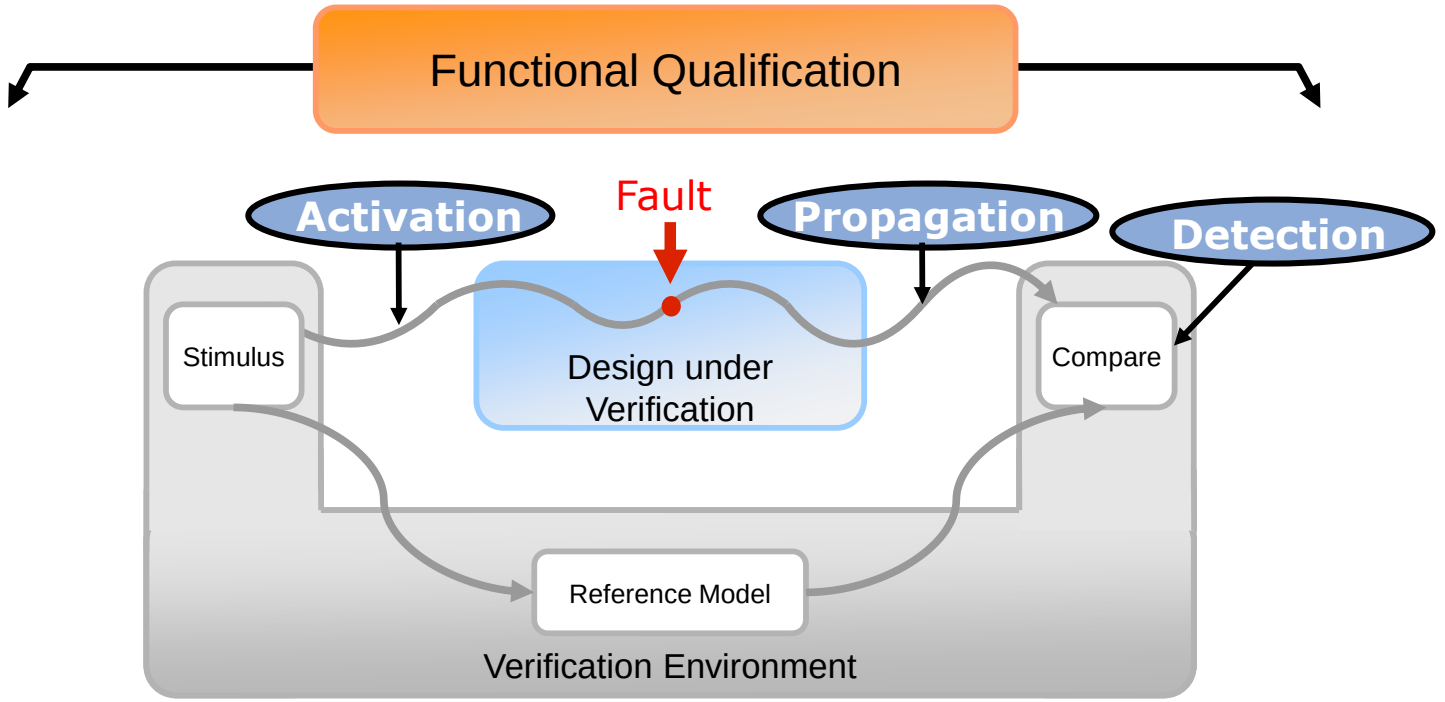
Introducing – Our Flow - Expected / Unexpected status of faults

Analysis of Unexpected faults

Limitations and Enhancements Requested

Conclusion and Q&A

Introducing Certitude



- Inserts “artificial bugs” called *faults* into the design
- Measures if your verification environment can **activate**, **propagate**, and **detect** the faults
- Identifies specific verification holes
- Provides an **objective** measure of overall verification robustness

Fault types

- RTL Faults

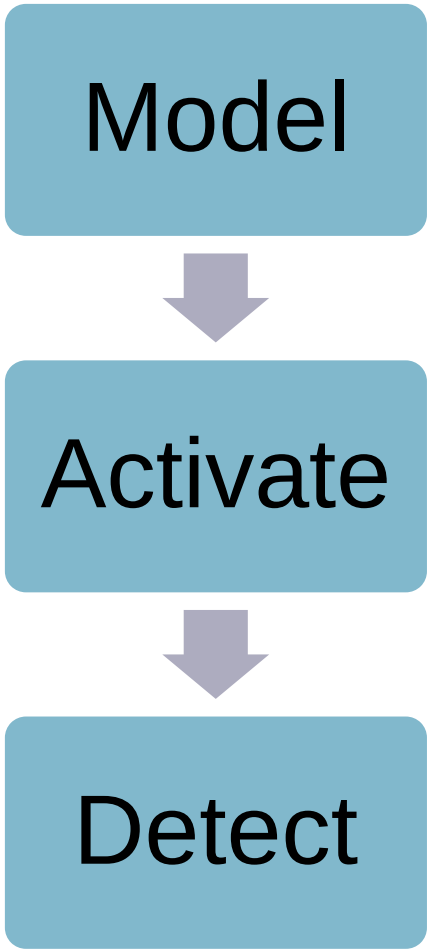
Type	Original code	Changed into
DeadAssign	assign o = i ; o <= i ; o = i ;	assign o = o ; /* code removed */ /* code removed */
ConditionFalse	if (cond)	if (0)
ConditionTrue	If (cond)	if (1)
NegatedCondition	If (cond)	if (!cond)
ElseDead	else	else if (0)

CaseItemDead	case (sel)
	  1 : <statement_block>
		case (sel)
		  1 : /* code removed */ ;

Negated	3'b001	3'b110
FlipFirst	3'b001	3'b101
FlipLast	3'b001	3'b000
Operator	&& || + …	|| && - …

Verification Improvement Mode

Identify weakness of verification environment



- **Parse RTL Files to determine faults to insert**
- **Search for unreachable faults**
- **Determine cones of influence**
- **Create Instrumented RTL Files for next 2 phases**

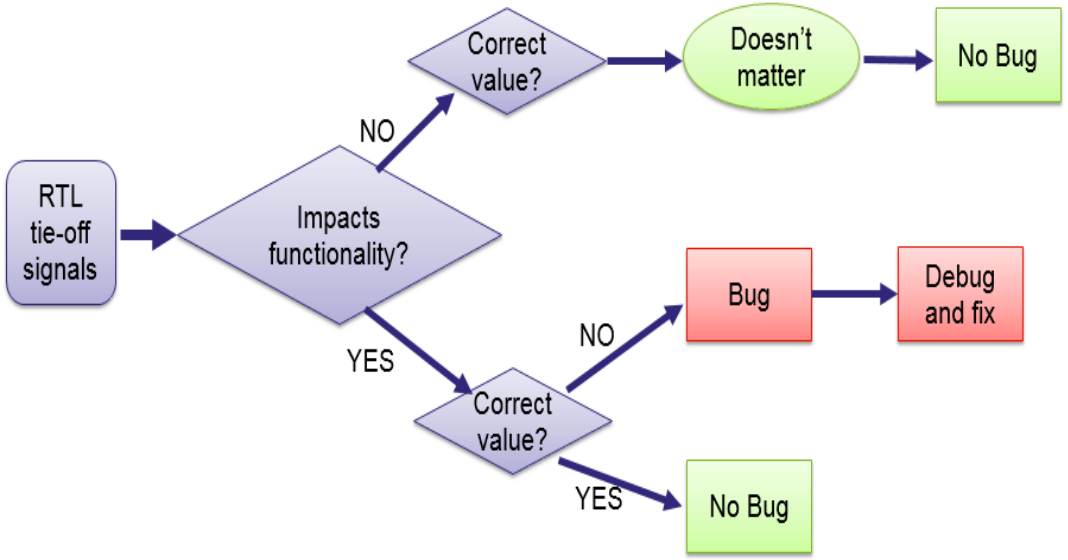
- **Run each test once**
- **Determine which tests activate each fault**
- **Determine which faults are Not Activated**
- **Identify the run time for each test**

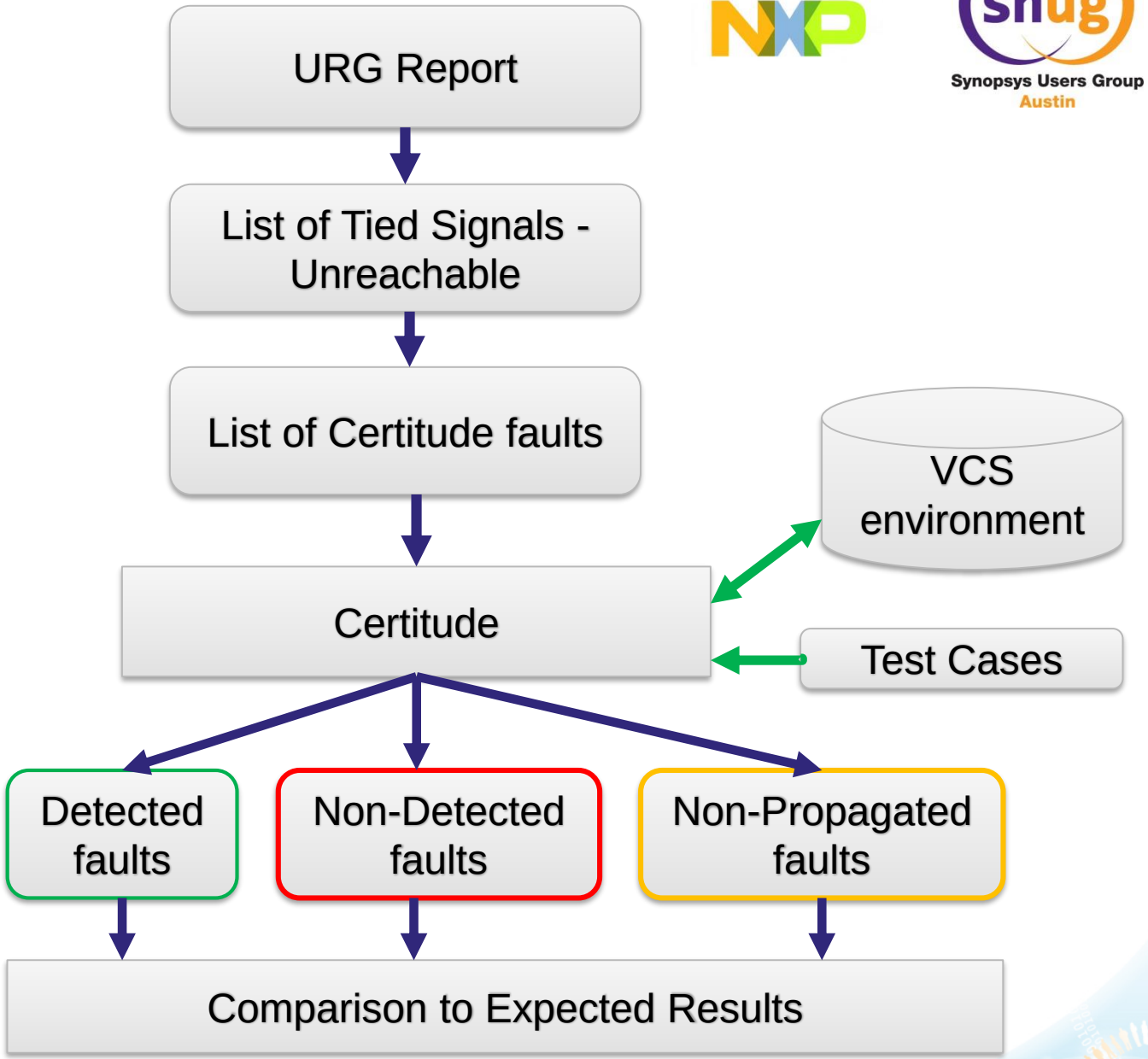
- **Insert each fault into the RTL**
- **Simulate with tests that activated fault**
- **Identify Detected, Not Detected, and Not Propagated faults**

Agenda

- Our Verification Challenge – Design Tie-offs
- Why explore Certitude?
- Introducing Certitude
- ❖ Introducing – Our Flow - Expected / Unexpected status of faults
- Analysis of Unexpected faults
- Limitations and Enhancements Requested
- Conclusion and Q&A

Certitude flow for Unreachable signals





Starting from the URG list

- ❑ Identify Unreachable signals of interest from the URG
- ❑ Each bit of signal – a potential bug

Signal	Toggle	Toggle 1->0	Toggle 0->1	Direction	Comment
I_aiop_atsi_rd_Ar_Prot[1]	Unreachable	Unreachable	Unreachal	INPUT	
I_aiop_atsi_rd_Ar_User[7:0]	Unreachable	Unreachable	Unreachal	INPUT	
I_aiop_atsi_rd_Ar_User[25:18]	Unreachable	Unreachable	Unreachal	INPUT	
I_aiop_atsi_rd_Ar_User[30:28]	Unreachable	Unreachable	Unreachal	INPUT	
I_aiop_atsi_wr_Aw_Prot[1]	Unreachable	Unreachable	Unreachal	INPUT	
I_aiop_atsi_wr_Aw_User[7:0]	Unreachable	Unreachable	Unreachal	INPUT	
I_aiop_atsi_wr_Aw_User[25:18]	Unreachable	Unreachable	Unreachal	INPUT	
I_aiop_atsi_wr_Aw_User[30:28]	Unreachable	Unreachable	Unreachal	INPUT	
I_aiop_ctlu_rd_Ar_Prot[1]	Unreachable	Unreachable	Unreachal	INPUT	
I_aiop_ctlu_rd_Ar_User[7:0]	Unreachable	Unreachable	Unreachal	INPUT	
I_aiop_ctlu_rd_Ar_User[25:18]	Unreachable	Unreachable	Unreachal	INPUT	

- ❑ Fault each bit of these signals
- ❑ Enable all these faults
- ❑ Set expectation for each fault

- ❑ All 3 above, done by extra scripts with Certitude command

- ❑ Run regression
- ❑ Look at Certitude reports for fault status
- ❑ Actual status not important for debug
- ❑ Actual VS Expected is important for debug

Expected and Non-expected results

- Our Challenge: Up front prediction of our expected results versus actual Certitude results
  - Need a way to pass the expectation as one of the fault attributes to Certitude

- Scenario 1: The tied-off signal has no expected functional impact
  - Expect all tests to pass with the faulty design
  - Predict the faults will be Non-Detected
  - If fault is Detected ?? Not expected -- Root cause the source of this problem (Problem with design?? Problem with Specs??)

- Scenario 2: The tied-off signal has expected functional impact
  - We expect at least one test to fail with the fault injected
  - We predict the fault will be Detected
  - If fault is Non-Detected ?? Not expected ➔ weak or missing checkers

Majority of fault results were Expected

- This is a section of our Expectations table showing Expected results, designated by “match”

Signal Name	Description	AxUSER AXI3	AxUSER AXI4	AxUSER ACE-Lite	FDMA expected	FDMA actual	FDMA compare
SRCID[0]	Source ID. See Section 5.5, Source ID Assignments	0	0	0	D	D	match
SRCID[1]	Source ID. See Section 5.5, Source ID Assignments	1	1	1	D	D	match
SRCID[2]	Source ID. See Section 5.5, Source ID Assignments	2	2	2	D	D	match
SRCID[3]	Source ID. See Section 5.5, Source ID Assignments	3	3	3	D	D	match
SRCID[4]	Source ID. See Section 5.5, Source ID Assignments	4	4	4	D	D	match
SRCID[5]	Source ID. See Section 5.5, Source ID	5	5	5	D	D	match

Agenda

Our Verification Challenge – Design Tie-offs

Why explore Certitude?

Introducing Certitude

Introducing – Our Flow - Expected / Unexpected status of faults

- Analysis of Unexpected faults

Limitations and Enhancements Requested

Conclusion and Q&A

Analysis of Non-expected results

- Scenario 1 Example: A Non-expected result was found for Ax_User bit 21
  - Expectation = Non-Detected, since this bit had no functional impact
  - However, the Negated fault on this bit was Detected
  - This is a section of our Expectations table, showing the “nomatch” on bit 21

Signal Name	Description	AxUSER AXI3	AxUSER AXI4	AxUSER ACE-Lite		FDMA expect	FDMA actual	FDMA compare
SPARE	Spare	18	18	18		ND	ND	match
SNOOP[0]	Snoop: Defined in ARM AMBA4 ACE-Lite Specification.	19	19	-		ND	ND	match
SNOOP[1]	Snoop: Defined in ARM AMBA4 ACE-Lite Specification.	20	20	-		ND	ND	match
SNOOP[2]	Snoop: Defined in ARM AMBA4 ACE-Lite Specification.	21	21	-		ND	D	nomatch
SNOOP[3]	Snoop: Defined in ARM AMBA4 ACE-Lite Specification.	22	22	-		ND	ND	match

Analysis of Non-expected results (cont'd)

- Scenario 1 Example: Analysis
  - Turns out from the IP guide, tying this bit to the opposite value is invalid.
    - The fault on this bit 21 presented an unsupported combination of values on the ACE-Lite interface.
    - Clearly documented to make IP go into unpredictable behavior
    - The combination was - (AWBAR=0, AWSNOOP=10, AWDOMAIN=100) at a ACE-Lite interface, which is illegal according to the ACE-lite protocol
    - Only ACE can get such transactions ➔ Block goes into unpredictable behavior
  - This was a learning experience as we gained a better understanding of the Interconnect IP specifications

Analysis of Non-expected results (cont'd)

- Scenario 2 example: Non-expected result for bits Ar_User bits 7 through 0
  - Expectation = Detection
  - However, actually the faults were Non-Propagated

Signal Name	Description	AxUSER AXI3	AxUSER AXI4	AxUSER ACE-Lite	CE expect	CE actual	CE compare
SRCID[0]	Source ID. See Section 5.5, Source ID Assignments	0	0	0	D	NP	no match
SRCID[1]	Source ID. See Section 5.5, Source ID Assignments	1	1	1	D	NP	no match
SRCID[2]	Source ID. See Section 5.5, Source ID Assignments	2	2	2	D	NP	no match
SRCID[3]	Source ID. See Section 5.5, Source ID Assignments	3	3	3	D	NP	no match
SRCID[4]	Source ID. See Section 5.5, Source ID Assignments	4	4	4	D	NP	no match
SRCID[5]	Source ID. See Section 5.5, Source ID Assignments	5	5	5	D	NP	no match
SRCID[6]	Source ID. See Section 5.5, Source ID Assignments	6	6	6	D	NP	no match
SRCID[7]	Source ID. See Section 5.5, Source ID Assignments	7	7	7	D	NP	no match

Analysis of Non-expected results (cont'd)

- Scenario 2 example: Analysis
  - These bits carried the SourceIDs for initiators all the way to slaves
  - This is a clear case of missing stimulus from this particular initiator
  - We enhanced the test suite to add a test for this initiator
  - We updated the checks for the SourceIDs propagating to slaves

  - These fixes made the testbench better able to detect bugs

Agenda

Our Verification Challenge – Design Tie-offs

Why explore Certitude?

Introducing Certitude

Introducing – Our Flow - Expected / Unexpected status of faults

Analysis of Unexpected faults

- Limitations and Enhancements Requested

Conclusion and Q&A

Limitations and Enhancements requested

- General:
  - One-step flow for faulting tie-offs (unreachable signals).
    - Work-around: Multiple steps with URG report analysis and subsequent faulting.
  - As of today, Certitude in Native-VCS mode is not supported with VCS Partition Compile flow.
    - Work-around: Mitigating with monolithic compile flow.
  - Certitude in the default Stand-alone mode was the only option
    - Needs to have the instrumented files in a separate folder (since Read/Write capability is required)
    - Work-around: Checked out the original files and added R/W permissions
  - The testcase list still needs extra manual edits
    - Work-around: For example: Replace spaces with colons

Limitations and Enhancements requested

- Project Specific:
  - Tight coupling between URG report and Certitude required
  - Selecting signals to fault -
    - Example:
      - 1 signal each from 3 Interfaces – AWUSER[21:0] ➔ 22 * 3 = 66 bits
      - Copy paste them manually to Certitude config files
      - Command for identifying fault number for each bit corresponding to activation
      - Putting them in a script -- fault_enable.tcl
      - Used – “user_data” attribute of fault to record the expectation – inside a script – fault_expectation.tcl

    - Desired Solution/Enhancement:
      - Certitude be able to pull the signals from URG or Verdi
      - Enable user to select the bits to be faulted
      - And automatic fault number identification by Certitude
      - And hopefully a GUI to input the expectation for each fault

Limitations and Enhancements requested (cont'd)

- Project Specific:
  - Find fault numbers and enabling them ➔ fault_enable.tcl
    - Work-around: Manual

```
faultenable -faultlist=[faultlist -subunitname=work.gio_noc_network.module_gio_noc_p2_cg_clk_Cm_root
-bitindex=18 -formalport=I_ccp_Aw_User -type=InputPortConnectionBitNegated]
```

  - List of faults output by script

```
I_aiop_atsi_wr_Aw_User[7:0]   --        37203 37206 37209 37212 37215 37218 37221 37224
I_aiop_atsi_wr_Aw_User[25:18] --        37149 37152 37155 37158 37161 37164 37167 37170
I_aiop_atsi_wr_Aw_User[30:28] --        37134 37137 37140
I_aiop_ctlu_wr_Aw_User[7:0]   --        93252 93255 93258 93261 93264 93267 93270 93273
I_aiop_ctlu_wr_Aw_User[25:18] --        93198 93201 93204 93207 93210 93213 93216 93219
I_aiop_ctlu_wr_Aw_User[30:28] --        93183 93186 93189
I_aiop_fdma_wr_Aw_User[7:1]   --        41442 41445 41448 41451 41454 41457 41460 41463
I_aiop_fdma_wr_Aw_User[25:18] --        41388 41391 41394 41397 41400 41403 41406 41409
I_aiop_fdma_wr_Aw_User[30:28] --        41373 41376 41379
...verif49/workarea 12$
```

Limitations and Enhancements requested (cont'd)

- Project Specific:
  - Collate the report for expected/non-expected results.
    - Work-around: Manually wrote the fault_status.tcl to export a CSV file

Fault_ID	Output	Input	Expected	Actual	Match?	Number_of Testcases_Propagatedby	Propagating_Testcases
27354		Axi_Aw_User	Detected	NonPropagated	Nomatch	0	
27357		Axi_Aw_User	Detected	NonDetected	Nomatch	6	{-l sim:axi_massive_write_read_ddr
27360		Axi_Aw_User	Detected	NonDetected	Nomatch	6	{-l sim:axi_massive_write_read_ddr
27369		Axi_Aw_User	Detected	NonPropagated	Nomatch	0	
27372		Axi_Aw_User	Detected	NonPropagated	Nomatch	0	
27375		Axi_Aw_User	Detected	NonDetected	Nomatch	6	{-l sim:axi_massive_write_read_ddr

    - Enhancement: Certitude can show Mismatch in its Fault Report.

Agenda

Our Verification Challenge – Design Tie-offs

Why explore Certitude?

Introducing Certitude

Introducing – Our Flow - Expected / Unexpected status of faults

Analysis of Unexpected faults

Limitations and Enhancements Requested

- Conclusion and Q&A

Conclusions

- Certitude's ability to – Bit-negate – the signals has provided promising solution.
  - To build a novel methodology for reviewing Unreachable signals through faulting mechanism
  - To improve coverage on these signals using metrics of Match/No-match vs the expectations

- Timeline to deliver proof of concept, was reasonable.
  - For integration – initially about 3 to 6 weeks
    - Come across issues mentioned earlier
  - Running testcases / generation of Certitude fault reports – about 2 weeks
  - Analysis of report and testbench fixes – about 4 weeks

Conclusions (cont'd)

- We were able to expose weakness in verification environment
  - Adding the missing testcases – which are important : 2 tests
  - Enhance checkers – improve the verification quality to better catch the buggy tie-offs in RTL : for the above 2 tests
  - Helped us improve understanding of the design specifications : Update constraints on the SNOOP bits for the AXI drivers

- This metric based approach is better than the traditional method of manual reviews for tie-offs

- We will appreciate continued, strong co-ownership from Synopsys to help us productize this novel flow within NXP.
  - Limitations/Enhancements need to be addressed in timely manner

- Thank you to Certitude team and users.

Thank You