2017 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

An Efficient and Modular Approach for Formally Verifying Cache implementations

M Achutha KiranKumar V

Abhijith A Bharadwaj

Bindumadhava S S

intel

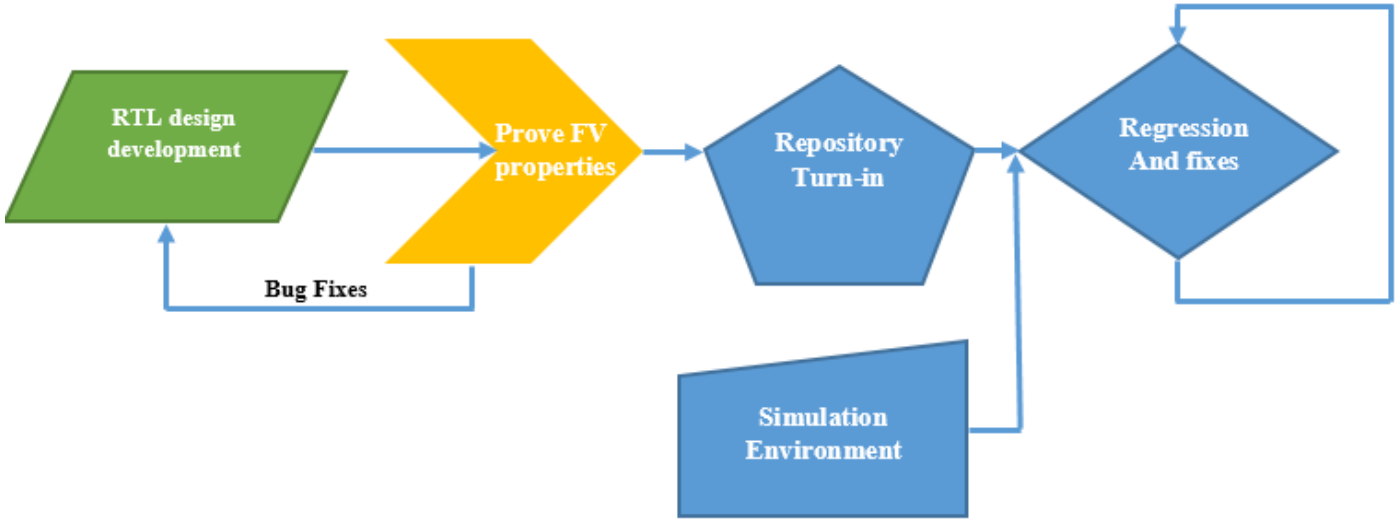
AGENDA

Why?
- Adopted FV Flow

How?
- FV enabling on cache design

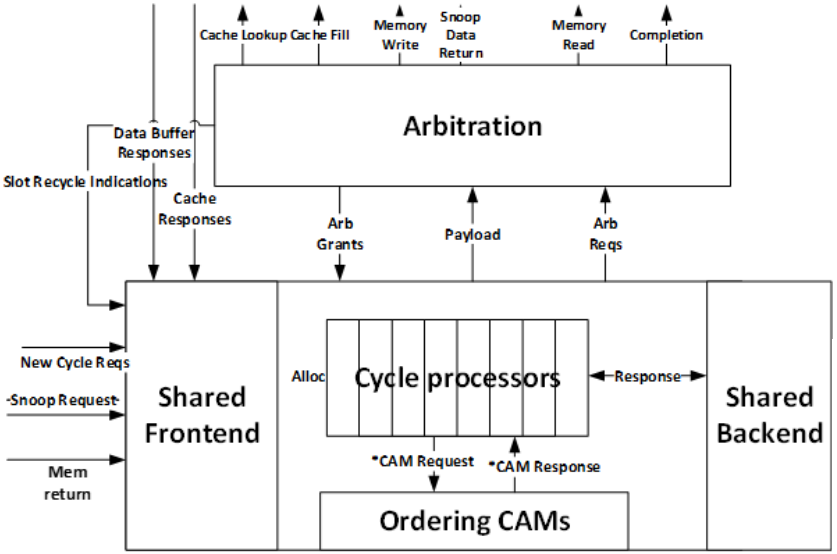
What was Achieved?
- Results of FV Design Exercise

Adopted FV Flow



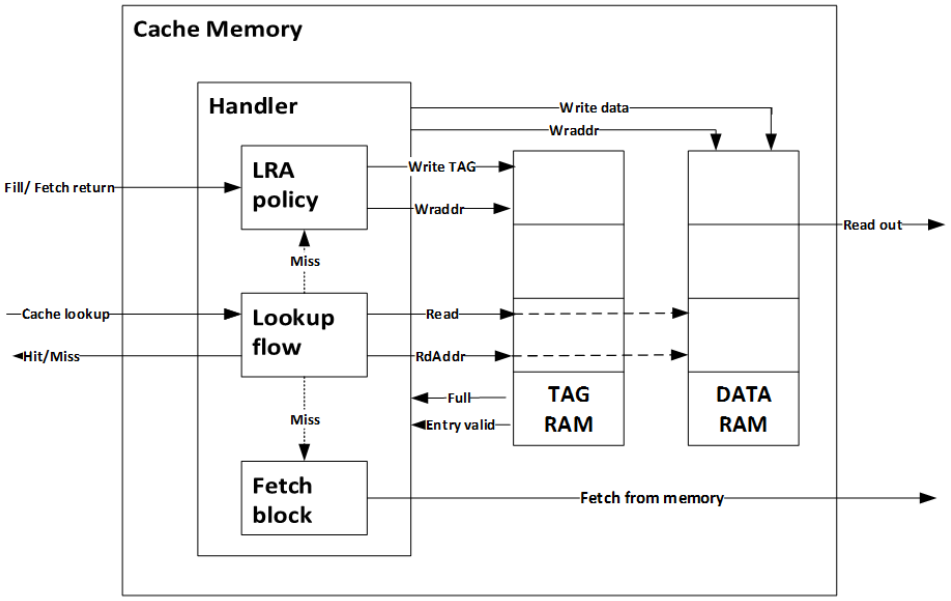
- Formal needs no introduction:
  - Formal Design Exercise empowers stronger RTL design
  - Modular approach on “Formal” design development
  - Stronger design turnin through Formal design exploration
  - The FV environment was integrated with the DV repo
  - New Cache Controller designed for latest project

Design Introduction

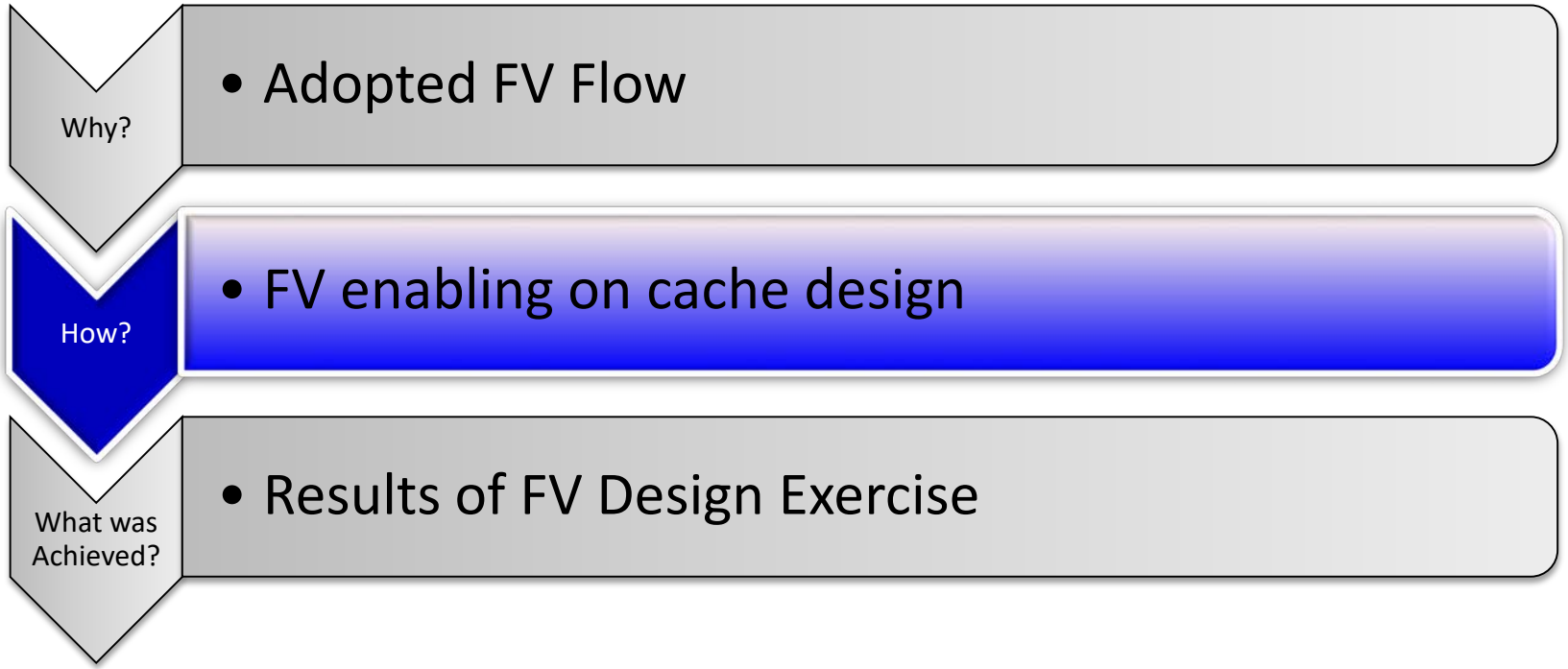


Cache FrontEnd

Cache BackEnd



AGENDA


Why?
• Adopted FV Flow
How?
• FV enabling on cache design
What was Achieved?
• Results of FV Design Exercise

Environment constraints and requirements

- Interactions with external agents:
  - Every interaction must be acknowledged individually. Else, the design hangs.
- External memory (Main):
  - The main memory is expected to be slow
  - Each memory request is accompanied by a ‘Tag’ (Physical address in the cache), which needs to be returned with the associated data.
  - Out of order and appropriate acknowledgements possible, else data corruption or hangs.
- Credit management:
  - Credit based FIFO at the input should not overrun, else such transactions will be dropped.

Initial Design Exercise

Structural checks on FSM:

- FSM State must never be X.
- A state must never attempt multiple arcs simultaneously.
- All valid arc transitions possible
- Invalid arc transitions never occur
- Covering that all the states are visited
- Eventually the state machine moves out of the current state

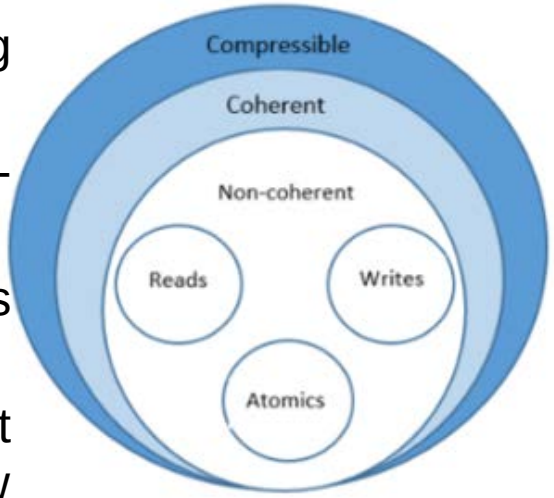
For example:

- NO_MULTIPLE_ARCS : assert property ((FSM_now == state_X) |-> $onehot0(FSM_nxt == state_Y, FSM_nxt == state_Z));
- NO_DEADLOCK : assert property ((FSM_now == state_X) |=> s_eventually (FSM_now != state_X));

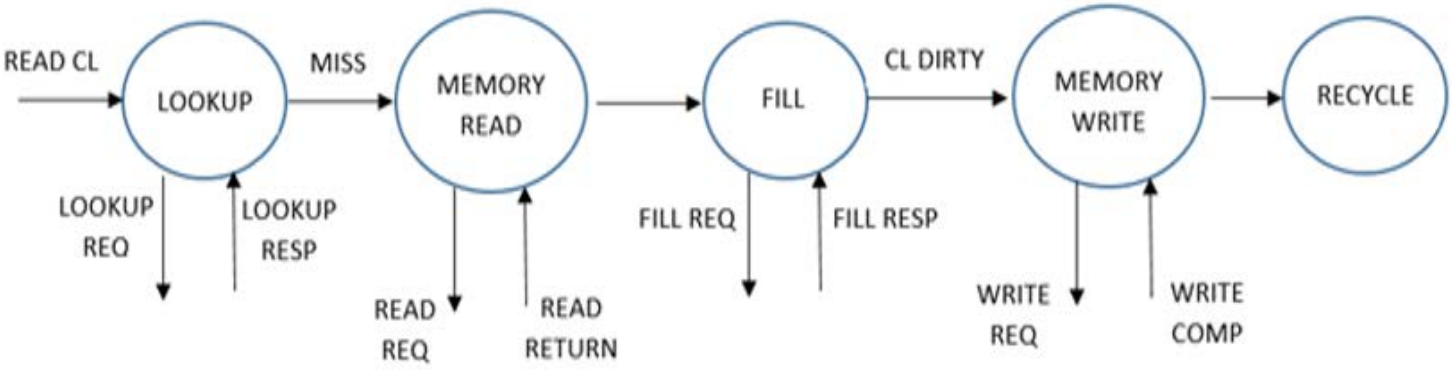
Cache Front End FV

Design breakup:

- Design was bifurcated into chunks of increasing complexity and functionality .
- Verification phase was broadly divided into Non-Coherent cycles, Coherent and Compressible cycles.
- The operations were broadly classified as Reads from a CL, Write to a CL and Atomics operations
- The FE interactions would adhere to a 'Request (REQ)-Acknowledge (ACK)' protocol, as shown below



Design breakup



CL read flow

Cache Front End FV – Contd.

Type	Description
Assume	A REQ is always followed by an ACK
Assume	ACK eventually appears only if requested (REQ)
Assume	No ACK for DELAY number of cycles after a REQ
Assume	If several different REQs request simultaneously for the same ACK, each REQ is individually ACK'd
Assert	No spurious REQs when a REQ is already in progress
Assert	If PS is the present state during a REQ, then NS should be the next state after an ACK
Cover	REQ and an ACK sequence appears

Req-Ack macro

Assertions:

- Micro operation transition checks:A transition would be triggered by a response from an external agent/cache BE. Integrated into the macro.
- No spurious requests: FE does not launch spurious/multiple requests to either an external agent/cache BE. Potential cache data corruption.
- BE does not recycle when active: Whenever a micro-op is active, the FE should not relinquish its possession of the CL. Potential deadlocks.
- Lookups forwarded to the BE only once: Whenever a micro-op is active, the FE looks up the data in the BE only once, in order to avoid collisions

Cache Back End FV

Safety properties:

- **`EVERY_LOOKUP_ACKNOWLEDGED.**
- **`READ_RETURN_IFF_LKUP**.
- **`LKUP_RETURNS_ONLY_M_CL**: Only M number of CL(s) is(are) returned in a lookup cycle, where M is an argument to be passed to the macro.
- **`FIRST_DEPTH_LKUPS_MISS**: Right out of reset or after a cache Flush (cache invalidate), the first DEPTH number of different lookups always return a MISS. This assertion is critical in checking both the reset and flush functionality.
- **`EVERY_HIT_RETURNS_DATA**: Every lookup that is a HIT results in a read return being sent to the FE in M cycles (specified by the user).
- **`HIT_DOES_NOT_WRITE**: Any lookup that is a HIT will not result in a write_enable to the TAG_RAM
- **`EVERY_MISS_GETS_WRITTEN**: Every address lookup that's a MISS results in that address being written into the TAG RAM.
- **`FETCH_ON_EVERY_MISS**.
- **`MISS_AFTER_FLUSH**: Any lookup after a flush returns as a MISS.

Cache Back End FV Contd.

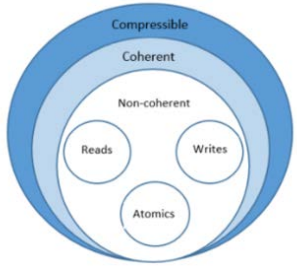
Performance property for LRU policy:

- A flag (lra_fv) is used to indicate that the window for LRA checks.
  - The flag sets when a lookup for ADDR_FV returns as a MISS.
  - The flag resets when counter miss_fv == DEPTH
  - The flag resets on a flush.
- A counter (miss_fv) is used to store the number of lookups that MISS.
  - The counter initializes to 0 out of design reset or on a flush.
  - The counter increments whenever lra_fv == 1 & lookup == MISS
  - Counter resets to 0 when lra_fv resets.

The assertions–

- **HIT_CONDITION**: assert property (lookup && lkup_addr == ADDR_FV && lra_fv |-> HIT)
- **MISS_CONDITION**: assert property (lookup && lkup_addr == ADDR_FV && !lra_fv |-> MISS)

Cache Back End FV Contd.



Bug Hunting:

- Once the verification scope was augmented greatly convergence issues kicked in.
- Several modifications were made to the design environment to suit the bug hunting process
  - Response (ACK) to any REQ to arrive within 7 cycles (delay=3, min latency)

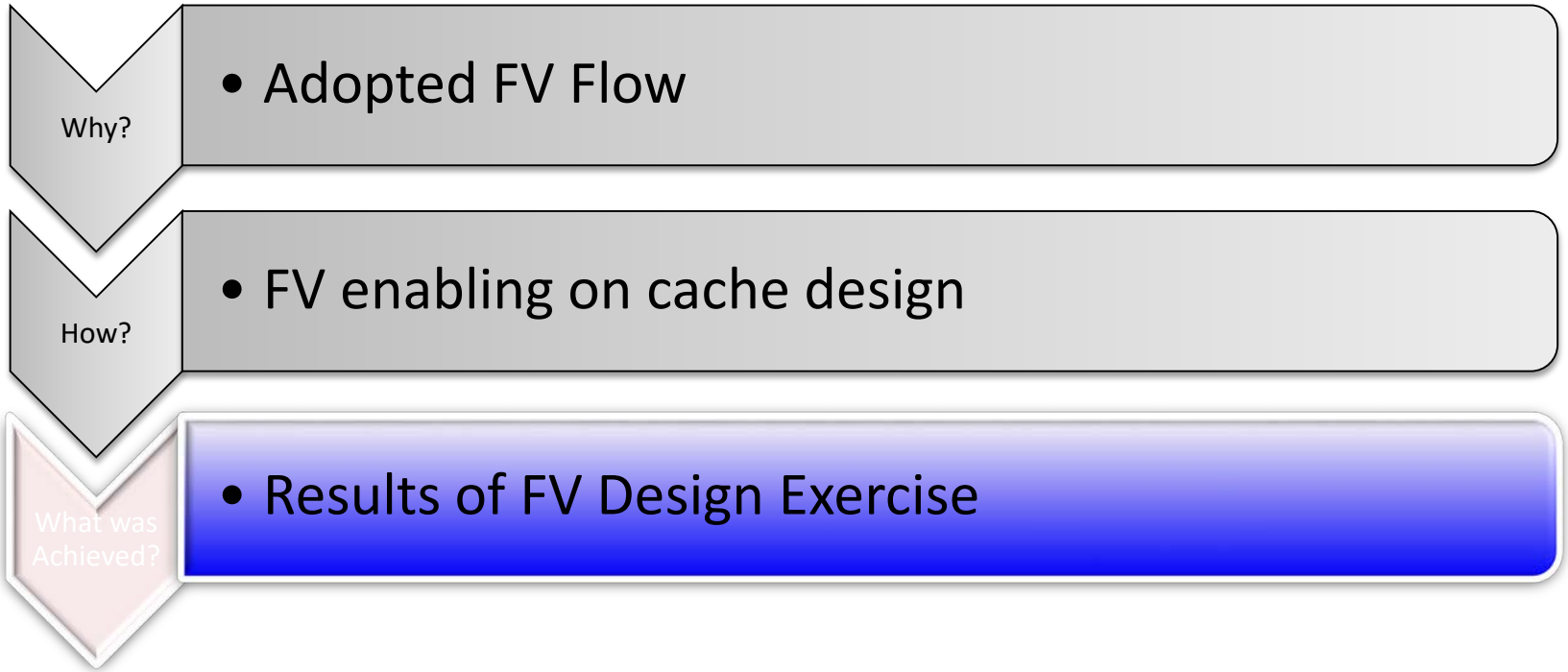
REQ_EVENTUALLY_ACKD: assume property (@ (clk) (REQ) |-> strong(##[delay:(delay+4)] ACK)); \

  - All liveness assertions made deterministic. A CL was expected to execute all transactions and recycle within a hundred cycles.

CL_GETS_RELEASED: assert property (@ (clk) CL_acquired |-> strong (##[1:100] (CL_got_released)[->1]));

- Force design to deadlock
  - NO_DEADLOCK: cover property ((FSM != STATE_1) [*DELAY]); //Pushes the design into a single state
  - DEADLOCK_CHK: assert property (FSM != STATE_1); // Prove the property from the failure for NO_DEADLOCK. A failure might be a bug.

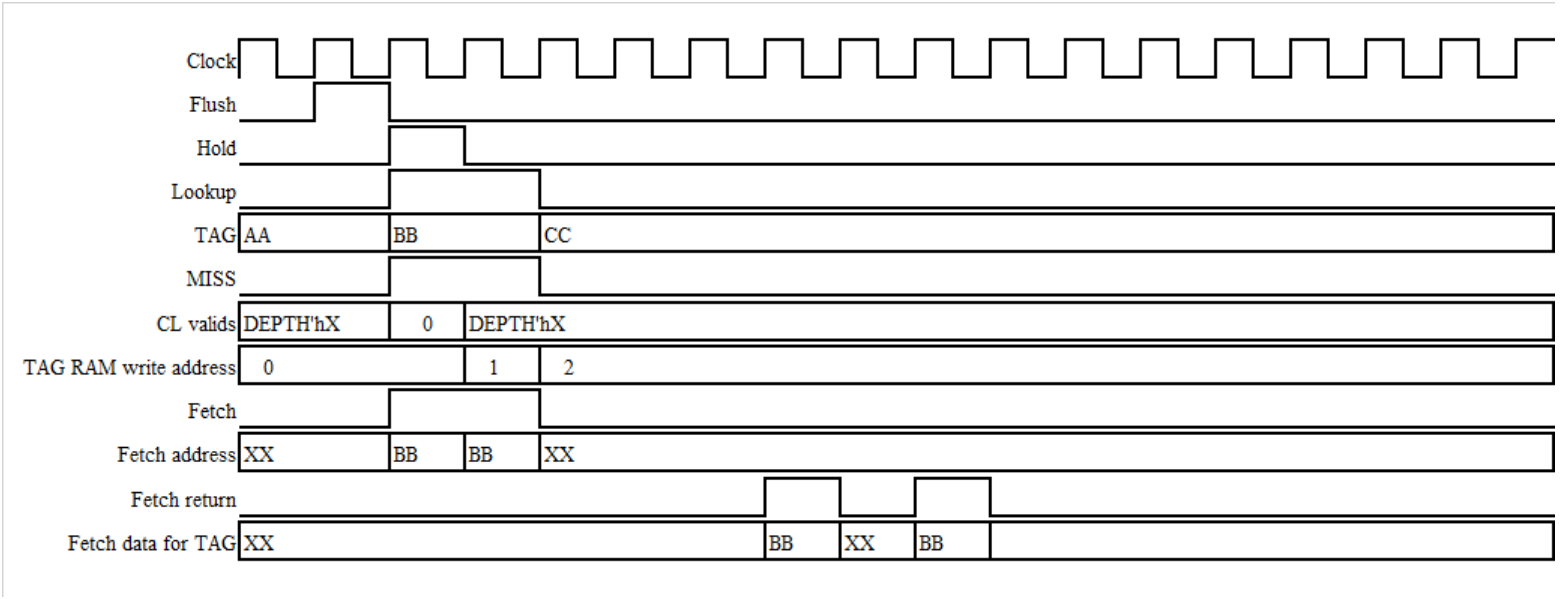
AGENDA


Why?
• Adopted FV Flow
How?
• FV enabling on cache design
What was Achieved?
• Results of FV Design Exercise

Results

A. Quality of issues uncovered:

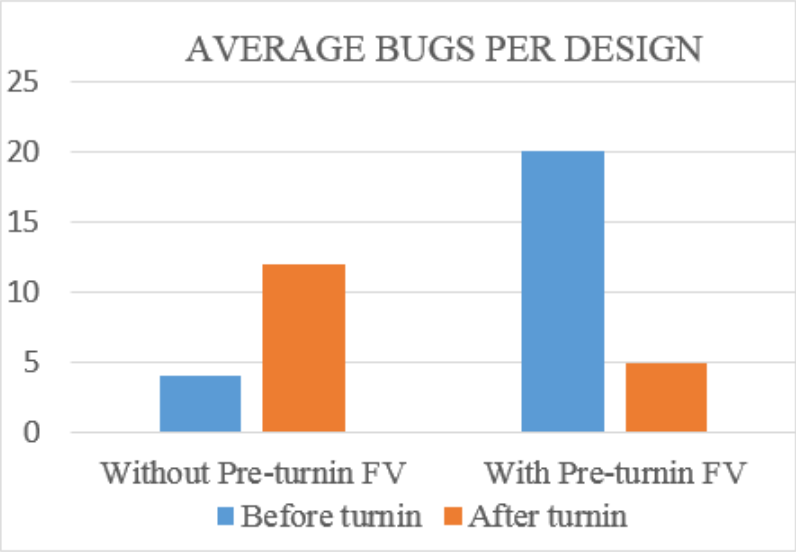
- 30 different cycle types composing all the non-coherent and coherent cache accesses were fully covered.
- The various FV strategies uncovered 25 different issues, ranging from minor typos to improper ordering assumptions in RTL that would not be exposed in simulation



Results

B. Return On the Investments (ROI):

- Without FV, the traditional simulation methodology found around 70% of the bugs after the turn-in for similar units.
- Using FV before turn-in, 80% of the issues found were before turn-in which reduced code churn to a great extent.
- On the areas FV'd, no bugs were logged after the Design Validation environment came out of reset.



2017 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION UNITED STATES

Q & A