AMD

# FORMAL VERIFICATION OF A GPU SHADER SEQUENCER

VAIBHAV TENDULKAR, MTS SILICON DESIGN ENGINEER

CHIRAG DHRUV, SR. MANAGER SILICON DESIGN ENGINEER

ADVANCED MICRO DEVICES INC.

DESIGN AUTOMATION CONFERENCE 2019, DESIGNER TRACK INVITED SESSION

# AGENDA

Shader Sequencer Introduction

Formal Verification Methodology

Case Study Results

Conclusions

# ABOUT AMD GRAPHICS

- Responsible for delivering an array of products catering to following segments:
  - Discrete
  - SoCs
  - Semi-custom
  - Compute
  - Embedded

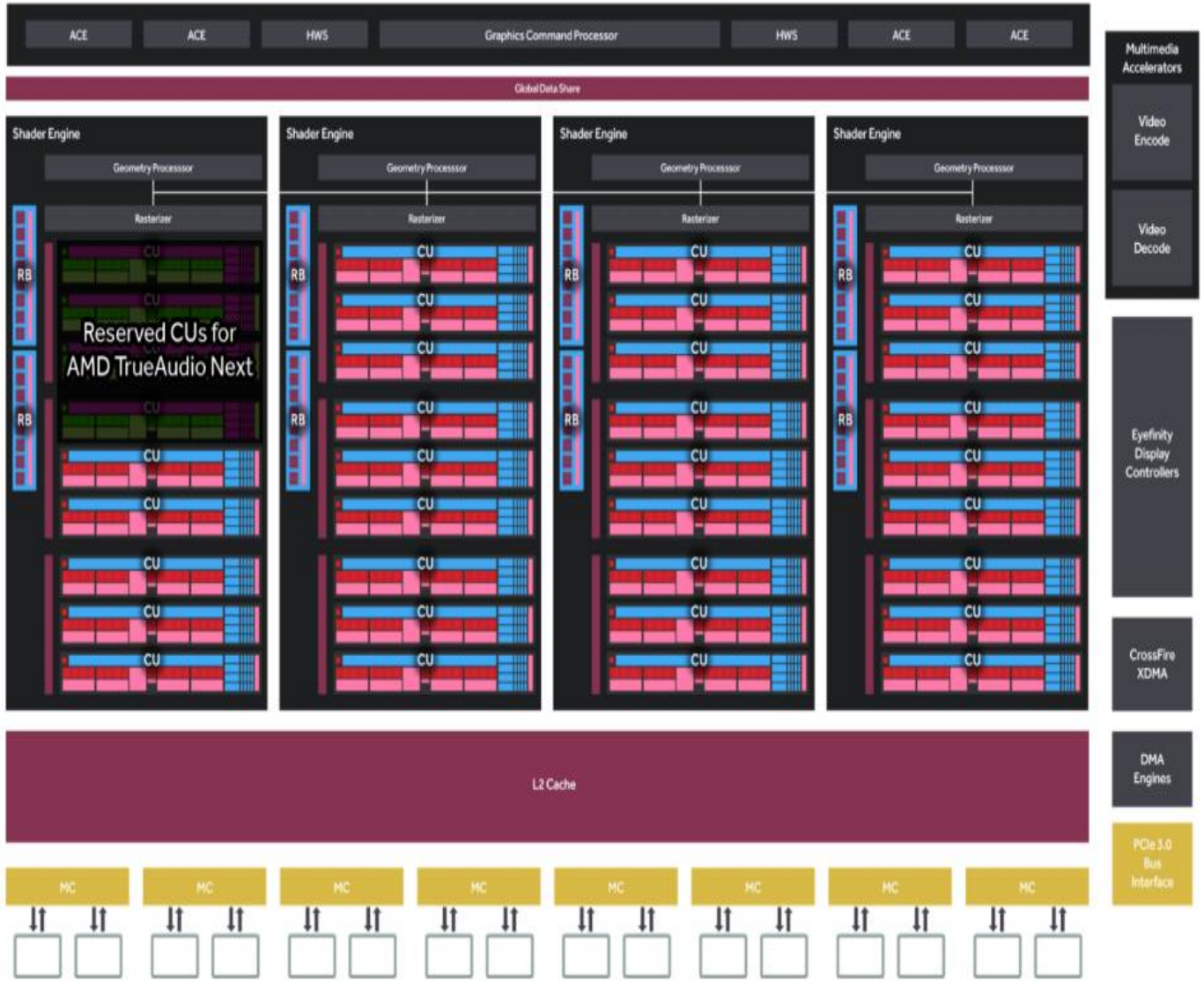
- Current generation of products in the market are based of “Vega” and “Polaris” architectures.

# AMD GPU ARCHITECTURES

- GPU Features
  - Responsible for processing either a pixel, vertex, or compute shader.
  - Must deal with massive work items grouped into workloads that execute the same shader program.
  - SIMD execution.

- GPU Verification Challenges
  - Mega-parallelism.
  - Execution can be interrupted by a host of async events.
  - Involves a large instruction set defined to process both graphics and compute workloads.

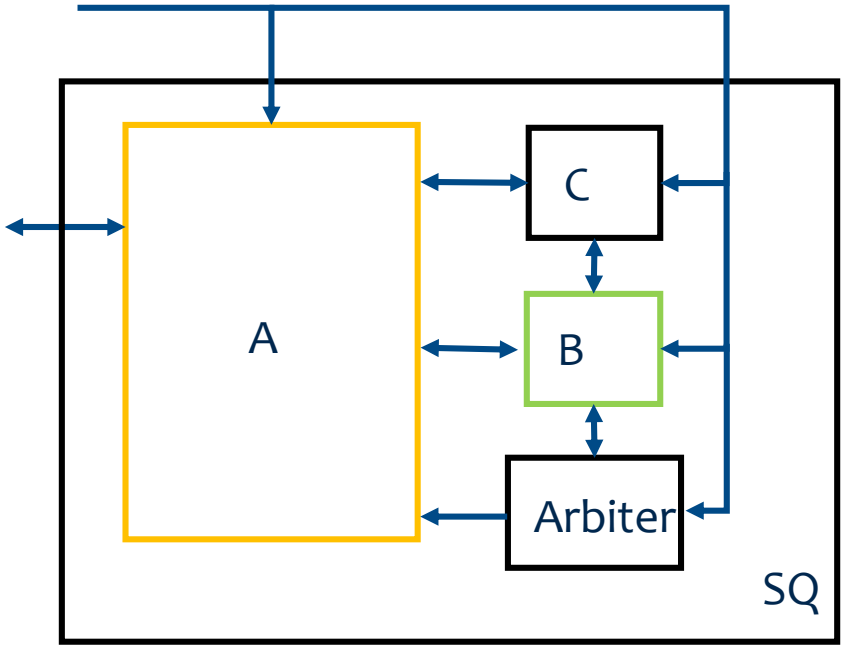
- Motivation for Formal Verification
  - Left-shift bug discovery.
  - Find simulation-resistant bugs.
  - Build formal methodology and in-house flow.




Citation: https://www.amd.com/system/files/documents/polaris-whitepaper.pdf

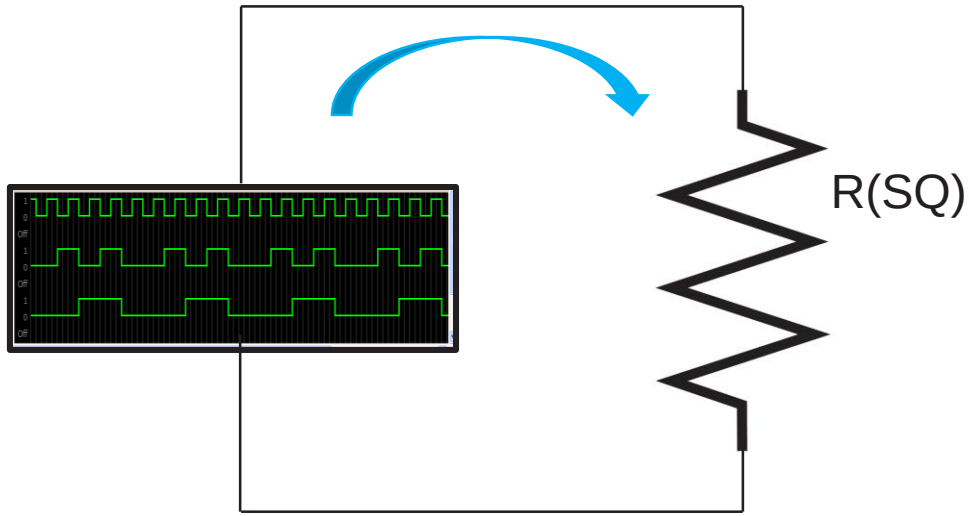
# SHADER SEQUENCER

- Sequencer (SQ) requirements:
  - Sequences and dispatches instructions for a SIMD.
  - Fetch Instructions for each workload active on that SIMD.
  - Resolve dependencies, buffer and decode the instructions for each workload.
  - Arbitrate between different workloads and grant the winner workload the access to instruction issue channels.
  - Execute or issue the instructions to different blocks based on their type.



# MOTIVATION FOR FORMAL

- Sequencer is prone to **Simulation Resistance**
  - Parallel processing of fetch/decode/issue for many variations of instructions.
  - Many permutations of pipeline flow to handle all possible concurrent events.
    - Asynchronous events
    - Configuration changes
    - Stalls
    - Control Flow
  - Many of these events require the SQ to alter its instruction issue flow accordingly.
  - The permutations of these events occurring alongside SQ fetching, decoding, and issuing different flavors of instructions are large and may be difficult to cover completely using only a high-level simulation approach.



# AGENDA

Shader Sequencer Introduction

Formal Verification Methodology

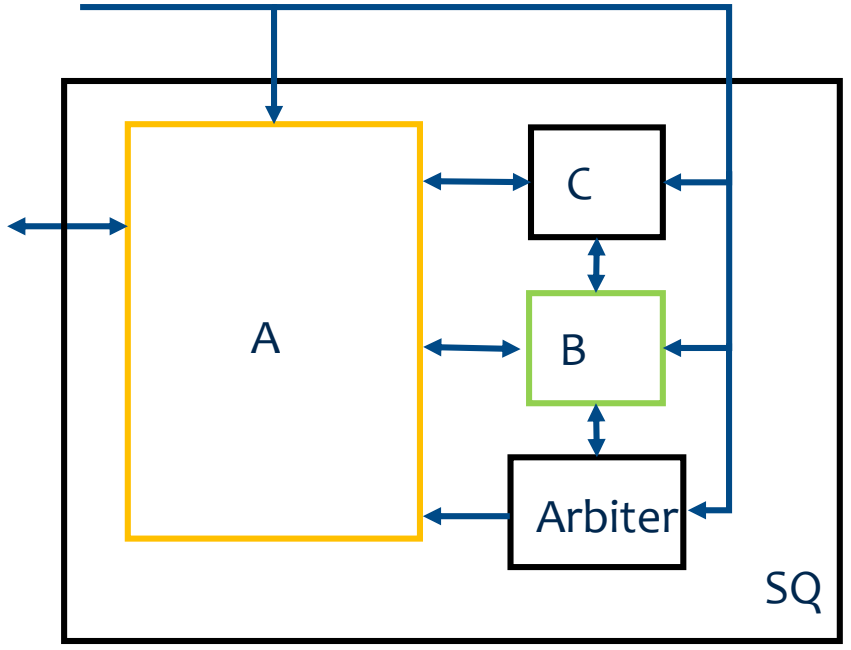
Case Study Results

Conclusions

# DECOMPOSITION FOR FORMAL PROPERTY VERIFICATION

Sub-blocks selected for formal verification:

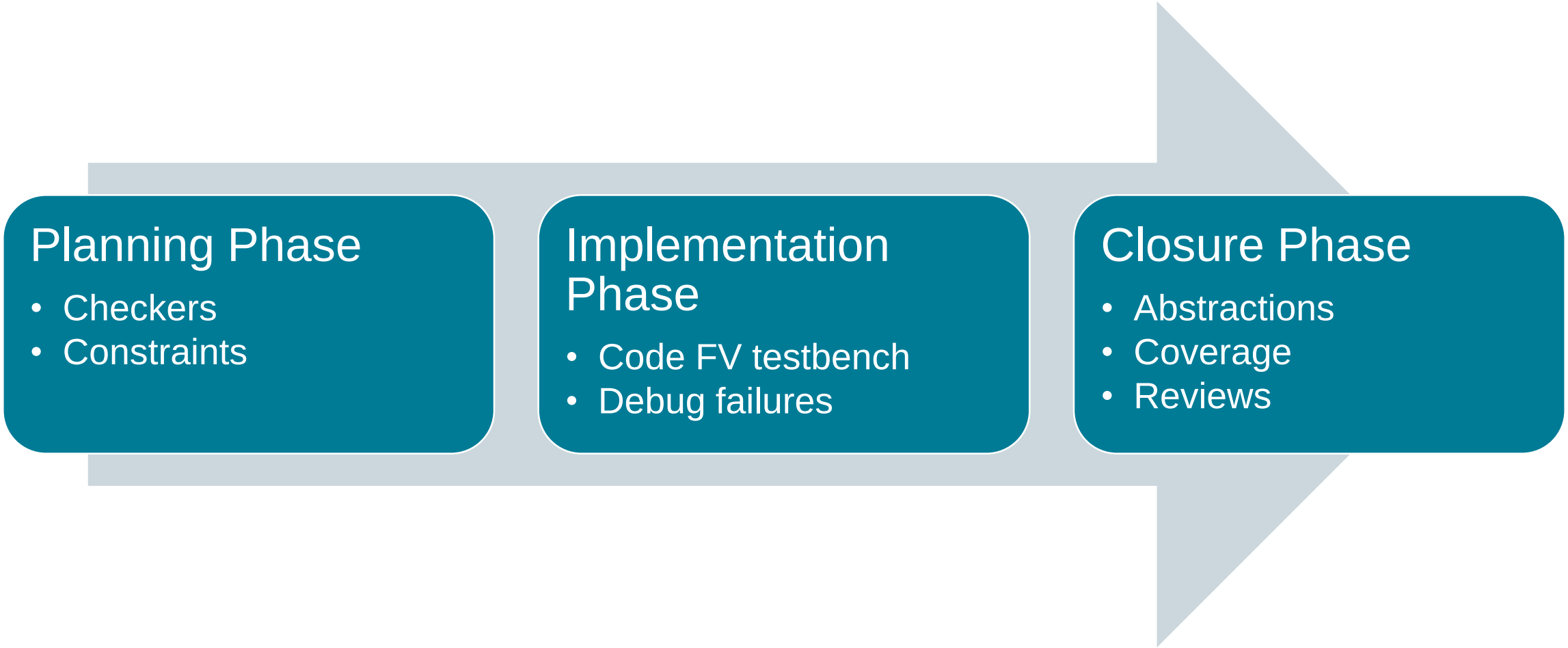
- ❑ Block_A
- ❑ Block_B
- ❑ Arbiter



# FORMAL VERIFICATION PLAN

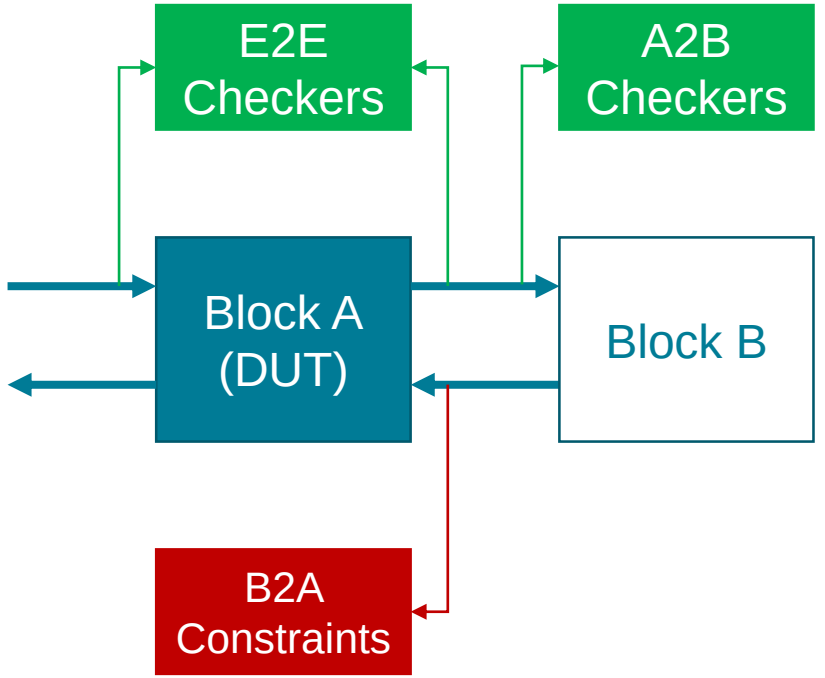
| Block | Formal Application | Checkers Planned |
|---|---|---|
| Block_A | Sign-off | 56 |
| Block_B | Bug hunting | 29 |
| Arbiter | Bug hunting | 32 |
| Total | | 117 |

# FORMAL VERIFICATION SIGN-OFF METHODOLOGY



# CHECKERS AND CONSTRAINTS STRATEGY

- Partitioning into 2 groups
  - Interface Properties
    - Formalize inter-block communication contracts
    - All properties written as asserts
      - Convert to assumes as necessary with tool directives
    - Naming convention for signals between Blocks A and B
      - Use suffix A2B or B2A
      - When verifying Block A:
        - A2B are checkers
        - B2A are constraints
    - Run interface properties in higher-level simulation
      - Shader core level
  - End-to-end Properties
    - End-to-end functions (e.g., arbitration scheme)
    - Data integrity
    - Forward progress



# AGENDA

Shader Sequencer Introduction

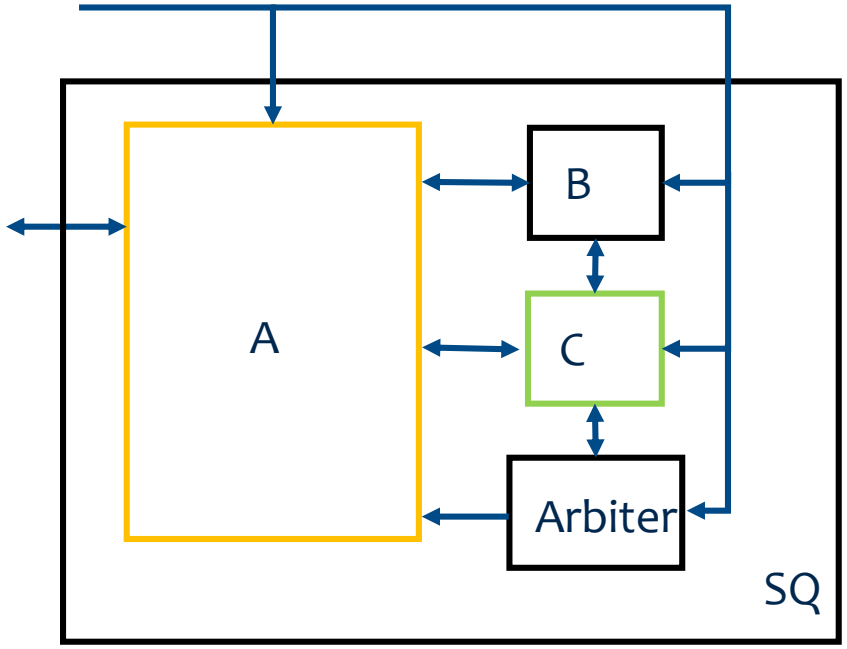
Formal Verification Methodology

Case Study Results

Conclusions

# BLOCK_A PRIMARY FUNCTIONS

- Fetch and retry instructions.
- Send instructions downstream.
- Handle special operation requests from downstream.
- Implement the Program Counter (PC).
- Note: Branching poses a challenge



# BLOCK_A E2E CHECKER EXAMPLES

| Category | Checker | Description |
|---|---|---|
| Sending of Requests | Valid Requests | Requests shall be sent for a given wave only when all conditions are satisfied.<br>• Between start and end of wave, space available in memory, etc. |
| | Resource Utilization | If any request satisfies conditions to be sent for a given wave,<br>then a request for any wave shall be sent in that cycle. |
| | Priority | If any request satisfies conditions to be sent for a given wave,<br>then a lower priority request shall not be sent. |
| | Tag | Requests shall be sent with tag corresponding to certain rules. |
| Program Counter | Correctness | PC return data for a given wave shall be equal to the previous instruction's PC for that wave. |
| Sending of Instructions | Valid Instructions | Instructions shall be sent for a given wave when only when instruction data is pending for that wave. |
| | Forward Progress | If instruction data is pending for a given wave, it shall be sent within a known bound, excluding cycles with higher priority instructions. |
| | Data Integrity | Instruction data sent shall be equal to that received. |

# BLOCK_A ABSTRACTIONS

- Instruction retry timer
  - Comes into play if a certain instruction needs to be re-issued due to lack of address translation.
  - Maximum value changed from a fixed large value to a configurable value.
  - Allowed verification of rollover condition at lower depths.

# BLOCK_A COVERAGE

All coverage holes were fixed prior to project release.

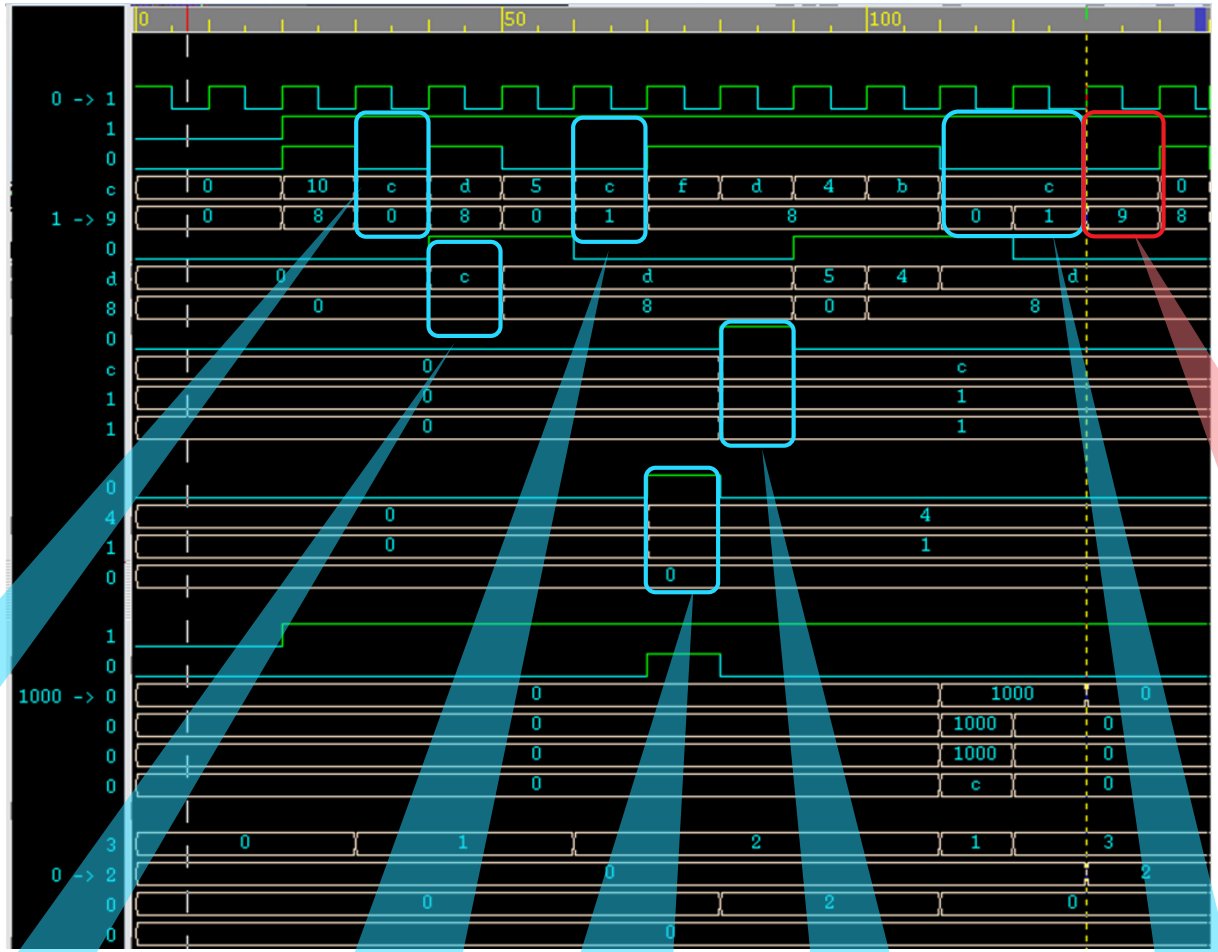
| Reachability | | | |
|---|---|---|---|
| **Type** | **Sub-Total** | **Unreachable** | **Covered** |
| Line | 3,970 | 8 | 3,952 (99.55%) |
| Condition | 5,305 | 43 | 5,204 (98.10%) |
| Toggle | 79,934 | 330 | 79,216 (99.10%) |
| RTL Cover properties | 1,049 | 1 | 889 (84.75%) |

| COI | | | |
|---|---|---|---|
| **Type** | **Total** | **Out of COI** | **In COI** |
| Registers | 193 | 16 | 177 (91.71%) |

| Formal Core | | | |
|---|---|---|---|
| **Type** | **Total** | **Out of Formal Core** | **In Formal Core** |
| Registers | 193 | 34 | 159 (82.38%) |

Coverage Holes due to Unused logic and disabled / untested features

# BLOCK_A SIMULATION-RESISTANT BUG EXAMPLE



Cycle 4: instruction fetch for tag-0

Cycle 5: instruction returned for tag-0

Cycle 7: instruction fetch for tag-1

Cycle 8: FLUSH resulting in invalidation of outstanding fetch requests (for tag-1)

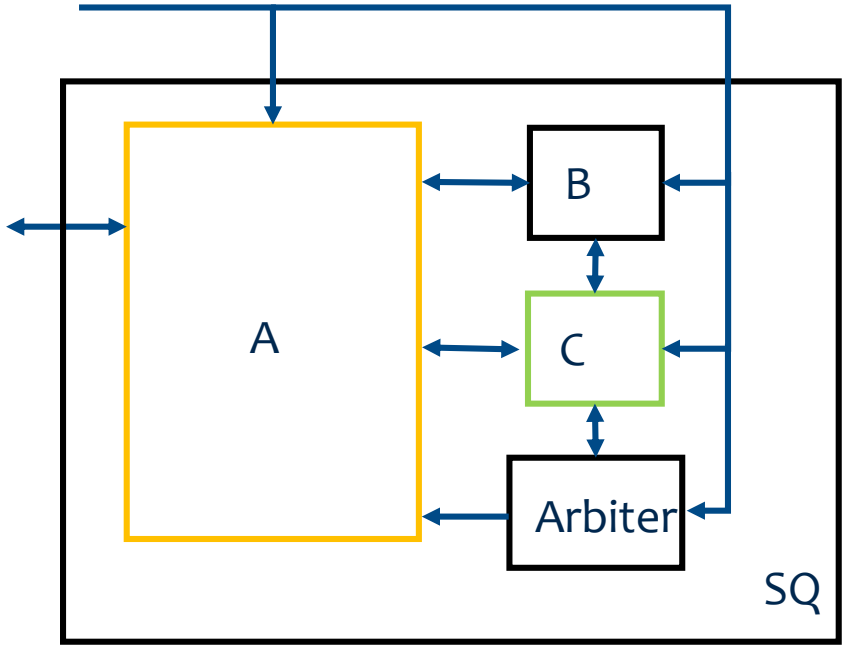
Cycle 9: Negative acknowledgement for fetch request for tag-1

Cycle 12-13: instruction fetch for tag-0-1

Cycle 14: Retry request for tag-9 (corresponding to negative ack for invalidated tag-1 in cycle 9)

# BLOCK_B PRIMARY FUNCTIONS

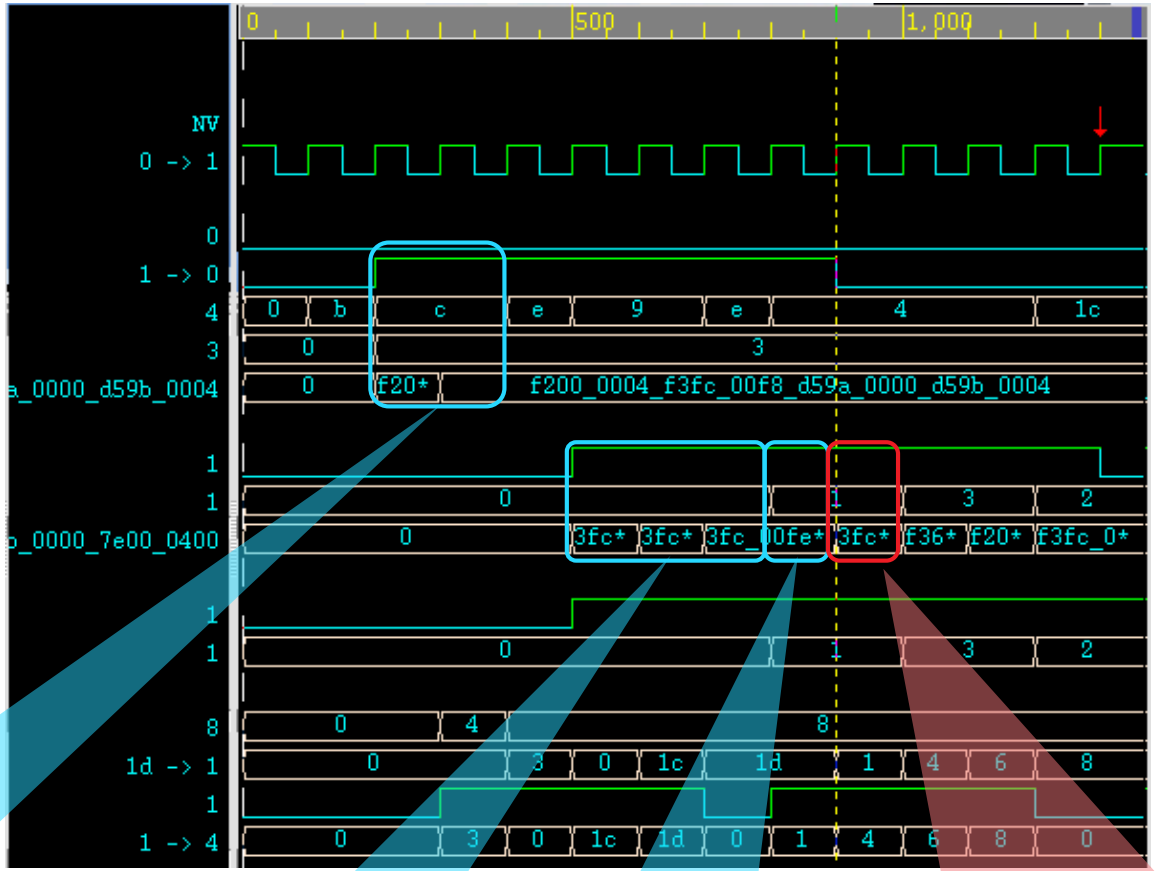
- Instructions are decoded and checked for legality.
- If qualified, instructions are checked for dependencies.
- If cleared, instructions are presented to the Arbiter and held until they are selected by the Arbiter.



# BLOCK_B E2E CHECKER EXAMPLES

| Category | Checker | Description |
|---|---|---|
| Sending of Instructions | Valid Instructions | If a Control Flow op is sent for a wave, the instruction should also be presented to the Arbiter for that wave. |
| | | If NACK op is sent for a wave, the instruction should not be presetedt to the Arbiter for that wave. |
| | | If a wave's instruction has been selected as winner by the Arbiter and the response is pending, no other instruction for that wave should be sent to the Arbiter. |
| | Data Integrity | Instructions sent to the Arbiter for a wave should be the same as those received for that wave. |

# BLOCK_B SIMULATION RESISTANT BUG EXAMPLE




Cycle 3-4: 3 instructions received
- 1st inst: 1st to 4th dword
- 2nd inst: 5th to 6th dword
- 3rd inst: 7th to 8th dword

Cycle 6-8: Block_B presents 3 instructions in these 3 cycles

Cycle 9: Block_B presents instruction with 1st dword received

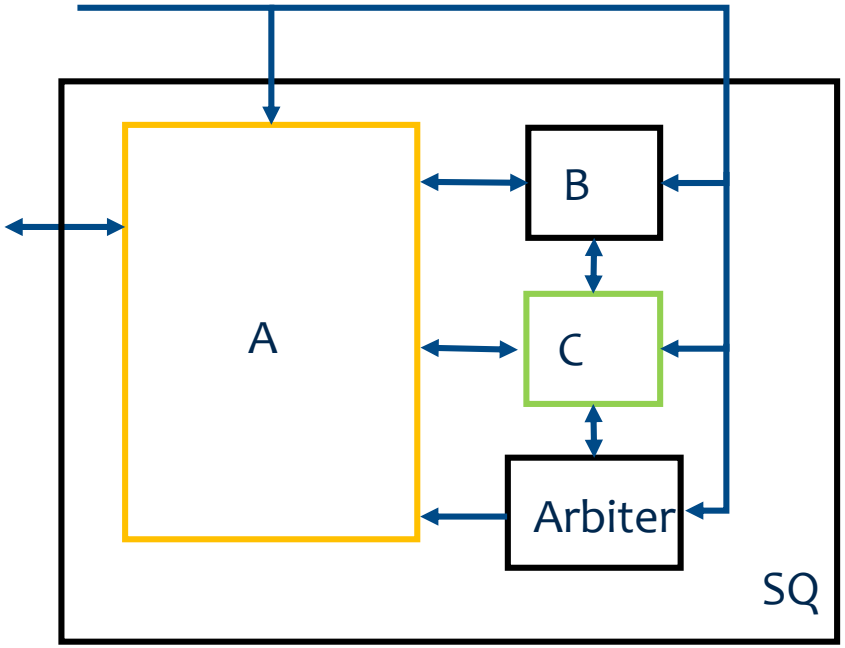
Cycle 10: Block_B presents instruction with a dword **not** received

# BLOCK_B IMPACT – EARLY BUG DISCOVERY

- A new feature that required multiple cycles to send an instruction to the Arbiter was introduced.
- Block_B was misinterpreting the subsequent dwords of the multi-cycle instructions as separate valid instructions.
- FV caught many such scenarios with data integrity checkers.
- **11 counter examples** were reported that led to **8 RTL changes.**
- Helped **left-shift** the bug discovery process for this new feature.

# ARBITER PRIMARY FUNCTIONS

- Makes arbitration decisions between waves trying to access the same issue channel.
- There are four channels based on the different instruction types.
- Arbiter maintains an age table.
  - Updated based on arrival and exit of different waves on the SIMD.



# ARBITER E2E CHECKER EXAMPLES

| Category | Checker | Description |
|---|---|---|
| Sending of Instructions | Valid Instruction Channel Arbitration | There shall be at most so many winners as there are instruction channels. |
| | | If a wave has been picked as the winner to issue on an instruction channel, no other wave with the same instruction type shall be picked as the winner. |
| | | For a wave, if an instruction has come in it shall eventually go out (liveness). |
| | | If a instruction channel is empty, then no valid wave shall currently have that type of instruction in the system. |
| Wave Table Validation | Update | Wave Table shall be updated on specific events in the design. |
| | Contents | The RTL wave table contents shall be identical to the checker's wave table contents. |

# FORMAL VERIFICATION SUMMARY

| Block | Formal Application | Checkers | RTL Bugs |
|---|---|---|---|
| Block_A | Sign-off | 56 | 17 |
| Block_B | Bug hunting | 29 | 16 |
| Arbiter | Bug hunting | 32 | 1 |
| Total | | 117 | 34 |

# AGENDA

Shader Sequencer Introduction

Formal Verification Methodology

Case Study Results

Conclusions

# CONCLUSIONS

- Formal sign-off was effective.
  - **No bugs found after sign-off of Block_A** in higher level simulation or emulation.
- Formal found simulation-resistant bugs.
  - Some bugs **highly unlikely to find otherwise.**
- Left-shift bug discovery
  - Formal may reduce overall verification effort and time by **finding bugs earlier.**
  - Improved simulation strategy by learning from formal bugs.

# FUTURE DIRECTIONS

- Expand use of formal sign-off.
  - Target additional designs areas with simulation-resistance.

- Mature the formal process and capabilities of the team.

# ACKNOWLEDGMENTS


- Elaine Poone, ASIC Design Engineer, AMD Inc.
- Kevin Cheng, ASIC Design Engineer, AMD Inc.

- Ashutosh Prasad, Formal Verification Engineer, Oski Technology
- Chirag Agarwal, Formal Verification Lead, Oski Technology

**Disclaimer**

The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions and typographical errors.

The information contained herein is subject to change and may be rendered inaccurate for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product releases, product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes.

AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION.

AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY DIRECT, INDIRECT, SPECIAL OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF AMD IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.