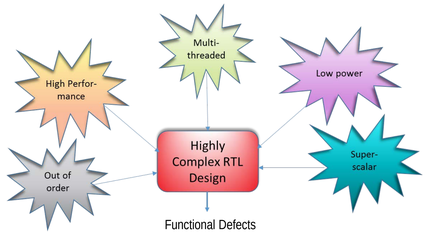
# IDeALS For All – Intelligent Detection and Accurate Localization of Stalls

**Pallavi Jesrani**
**Advanced Micro Devices, Inc., 2485 Augustine Drive, Santa Clara, CA 95054**

## Introduction




- High demands on modern pipelined systems lead to extremely complex RTL designs
- This leads to greater number of Functional Defects
- This work focuses on the specific type of Functional Defects that result in Stalls

## Design Under Verification (DUV)

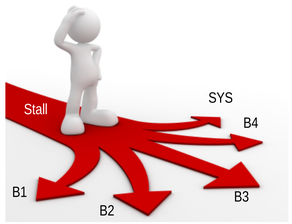

Consider a system consisting of 4 Blocks named B1, B2, B3, and B4 as shown above. The major flow of information/data is indicated by the arrows from B1 to B2 and so forth. Each block can include a number of pipeline stages.

## Stalls - Detection

- Deadlock
  Processing is completely stopped as all components are waiting on one another and there is no forward progress.
- Livelock
  Same sequence of events is continuously repeated as processing retries or loops back to an initial state without making any forward progress.

Extremely challenging to debug or root cause in a post-silicon environment. **Need to detect** all such defects in pre-silicon verification.

## Stalls - Localization



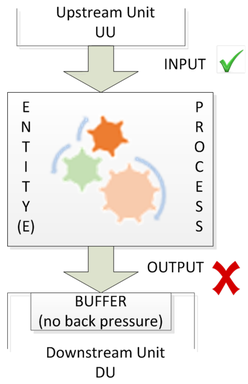

Narrowing down the **location of the RTL bug** to a specific block/unit/IP

- Increased Debug Efficiency.
- Visibility into health of design even before debug.
- Faster bug fix turnarounds.
- Faster design convergence.
- Fewer post silicon issues.

## IDeALS

- We present the IDeALS approach for Intelligent Detection and Accurate Localization of Stalls.
- It includes the following sections prefixed with IDeALS.
- We also present our implementation of this approach in our X86 microprocessor core design.

## IDeALS - Unit/Block level Timeouts

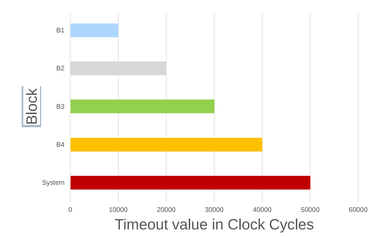



As per the IDeALS approach, a unit timeout fail should be flagged only if:

- the unit **has received input** from upstream units,
- it is **not encountering any back-pressure** from downstream unit(s),
- but it still is **not producing any output** for a predetermined (X) number of cycles .

## IDeALS - Staggering unit level timeout checks when used in higher level system environment

- Downstream unit may also supply some input.
- Accounting for all secondary/tertiary inputs may result in shadowing of the RTL that can be a maintenance nightmare.
- Second-level measure to avoid any mis-categorization due to lack of modelling of secondary/tertiary inputs.
- **Staggered** as per general flow of information.
- **Relative relationship** needs to be maintained for accurate localization.




## IDeALS - System Level Timeout

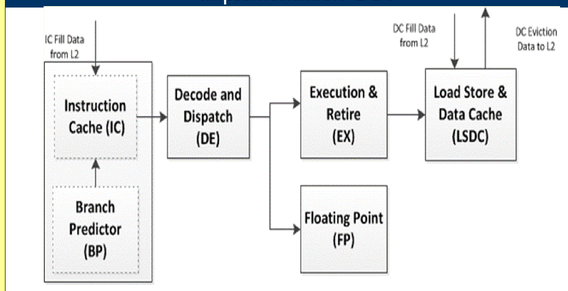
To capture any fails that may potentially fall through block level timeout checks. Examples of this are:

- Deadlock scenarios - all the units are waiting on each other.
- The system is waiting upon some external trigger to make forward progress.
- Implementation limitations of unit level timeouts.

## IDeALS Timeout Value Configuration

- **Iterative** process – runtime configurability is invaluable
- **Too small** a timeout value can lead to **increased chances of false fails** - progress is slow due to legitimate reasons but no livelock or deadlock
- **Too big** of a value - sim failing with **extremely generic fails like simulation or job timeout fails**. Also **highly inefficient** since we are then wasting precious simulation cycles.
- Knowledge of individual blocks and system under test is invaluable.
- Can start with the system level check and work our way backwards.
- If the same unit level timeouts are also utilized in unit level testbenches, we can leverage from those values as well.

## Implementation DUV

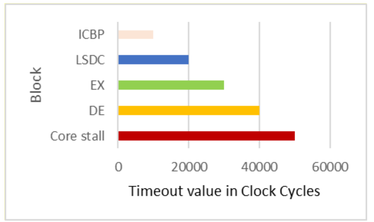



Simplified high-level block diagram of the design under verification (DUV). The DUV is the core engine of a deeply pipelined, superscalar, out-of-order X86 microprocessor core.

## Implementation Details

An IC stall would be flagged only if:

- IC was receiving input from BP in the form of predictions and not waiting on getting a response from upstream units like a level 2 cache (L2) or a page translation (TLB) response from LSDC
- The instruction byte buffer in downstream DE unit is not full
- However, the IC still does not produce output in the form of valid fetch packets to send downstream to DE in X number of cycles.




The relationship chosen for the different unit timeouts is as seen in above figure following the general flow of information in the core engine System level timeout was implemented as a Core level instruction stall.

## Attribution



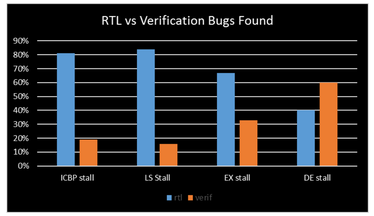
## Results - Accuracy

- Row => the bugs found by a specific unit timeout check.
- Column => the different units.
- The diagonal elements indicate the number of bugs where the localization was accurate.

| IDeALS Implementation | | Unit with RTL bug | | | |
|---|---|---|---|---|---|
| | | ICBP | LS | EX | DE |
| TIMEOUT CHECK | ICBP stall | 88% | 7% | 4% | 1% |
| | LS stall | 0 | 96% | 2% | 2% |
| | EX retire stall | 0 | 40% | 50% | 10% |
| | Dispatch stall | 25% | 0% | 0% | 75% |

- EX retire stall check had a limitation whereby it was not accounting for the lack of input coming in from the LS and hence ended up finding some LS bugs in the process. This is the reason for 40% LS bugs found by this check.

## IDeALS – Results - Effectiveness




The number of RTL bugs found by a check versus the number of verification bugs (bugs in the check itself).

## In-Progress and Future work

- **Dynamic scaling** of timeouts to avoid false positives
  - Legitimate cases where forward progress is very slow – but not stalled
  - Need to communicate to downstream timeout checks as well
- **Fine-Grained**
  - Refine approach to a specific flow within the unit/block
- Use in **Emulation** environment
  - Checks can be implemented in synthesizable system verilog
  - Helps increase the observability and visibility

## Conclusion

In this work, we presented the IDeALS approach to detect and localize stalls in complex pipelined systems. We have seen promising results from our implementation of this approach. This methodology has helped to:

- Increase Debug Efficiency.
- Provide more visibility into health of design even before debug.
- Reduce bug fix turnarounds.
- Hasten design convergence.
- Reduce number of post silicon issues.

## Acknowledgements

The results presented in this paper have been obtained by the combined effort of many talented engineers across multiple AMD locations.