FORMAL VERIFICATION OPTIMIZATIONS

Anamaya Sullerey
INTRODUCTION

- This presentation is a walkthrough of a formal verification exercise on a large and complex block.
- The size and complexity of this block made it too difficult to verify without any modifications.
- The focus of this presentation is on the optimizations that were employed to reduce the state space and complexity.
- These optimizations are not generic but they can be added to the bag of tricks and used where appropriate.
MULTI CORE NETWORK PROCESSOR

Arriving packets are assigned to various cores by a spray function.

These cores take variable amount of time for processing of packets.

Packets are put back in order according to the input order and the firmware running on the network processor cores.

I will go over formal verification of packet ordering block.
BLOCK DIAGRAM

 Queues -> Schedule -> Core pipeline

 Enque

 Network Processor

 Output

 Command state

 Other state
PACKET ORDERING BLOCK FUNCTIONALITY

- Pipeline operations:
  - *enqueue* operation occurs when a packet arrives. This operation adds the token for this packet to a specified queue.
  - *addCmd (add command)* operation occurs when a network processor core sends a command for some token. This operation updates the token’s command state.
  - *execCmd (execute command)* operation occurs when a queue is selected by the scheduler. This operation executes the first command for the head packet of that queue.

- Token commands:
  - *relo (reocate)* command relocates the token from one queue to another.
  - *delete* command frees the token.
  - *transmit* command transmits a copy of the packet represented by the token.
EXAMPLE : LIFE OF TOKEN IN ORDERING BLOCK

- New token
- Enque
- Delete
- Network Processor
- Output
- Queues
- Schedule
- core pipeline
- Command state
- Other state
OVERALL STATE SPACE

- Queues: thousands of queues, each queue with queue pointers and queue element state
- Scheduler: large set of flops, of the order of queues
- Token state: thousands of tokens
- Pipeline: few thousand flops
- **Overall state space is too large**
AREA OF FOCUS: PROVING COHERENCE OF KEY STATE VARIABLES

- Most properties for outputs were framed in terms of input commands and current state.
- Multiple instances of state for same token or queue could be in the pipeline. Pipeline has extensive bypass mechanisms to handle this.
- Proving properties for coherence of state under all possible event combination is the most challenging aspect.
- Remaining slides in this presentation concentrate on methods employed to proving correctness of one those state variables, the command state for tokens.
- Other state variable checks used similar optimizations.
OPTIMIZATION 1: TOKEN STATE REDUCTION

- To prove correctness of token state operations, we need to prove the properties for single non-specific token.
  - Model for token functionality tracks a single token. I will refer to it as the magic token in this presentation.
  - Magic token is selected during reset phase. It’s value is kept unconstrained.

- Any token other than magic token is left unconstrained

- Error isolation in design is a necessary prerequisite for this optimization to work. Error conditions related to one token must not have any affect on other tokens.
IMPACT ON VERIFICATION

- No affect on verification quality
- State space is greatly reduced since most memories are shrunk from few thousand entries to a single entry
OPTIMIZATION 2 : REMOVAL OF SCHEDULER

- Scheduler is built using preverified library elements
- Scheduler can be independently verified.
- Fairness of the scheduler is an orthogonal behavior to correctness of state
- Removal of scheduler does not add risk to the verification of pipeline functionality
IMPACT ON VERIFICATION

- Scheduler functionality needs to be separately verified
- State space is greatly reduced since arbitration logic was adding complexity. This opens up for more optimizations that are discussed later.
This diagram is conceptual. In reality, each arc takes few cycles to execute hence these operations have time overlap.
OPTIMIZATION 3: REMOVING LOOPS IN STATE MODEL

- Long delays in the design compared to the pipeline depth
- Round trip delay to packet processor
  - Initial enqueue to first `addCmd` operation
  - `execCmd` operation acknowledgement to packet processor to new `addCmd` operation from processor. An acknowledgement is a must for at least every other command since there is limited command space per token.
- Delay of scheduler
  - `execCmd` operation to new `execCmd` operation
- During these long delays pipeline gets flushed with respect to the magic token
- The state diagram can be optimized to remove all loops
UNROLLED TOKEN INSTRUCTION STATE DIAGRAM

@enqueue

uninit

{}

@addCmd

{} 

{C0}

{C0,C1}

@execCmd

@execCmd

@addCmd

@addCmd

{C0,C1}

{C0}

{} 

{C0}

{C0,C1}
IMPACT ON VERIFICATION

- Remapping requires many new assumptions to be added. All attributes related to the command state need to be consistent with start state.
  - Before: Command count was set by incoming commands. Scheduler model used it to allow execCmd.
  - After: \((\text{Init state} == \{\}) \Rightarrow (\text{command count} == 0)\) ….

- To mitigate this risk,
  - New assertions are required to verify assumptions on delays. These assertions are used in formal verification and simulation.
  - Review of the remapping is required.
  - Assume-guarantee approach needs to be followed on new assumptions wherever possible.
OPTIMIZATION 4: SEPARATION OF MODELS USED FOR ORTHOGONAL FUNCTIONS

- Token and queue properties are somewhat orthogonal
  - When proving properties for token state updates queue state modeling is kept minimal
  - When proving properties for queue state updates token state modeling is kept minimal

- Token properties are proven using magic token and some queue attributes

- Queue properties are proven using magic queue and some token attributes
IMPACT ON VERIFICATION

- This is just an effective approach for modeling. No impact is made on verification quality.
OPTIMIZATION 5: USING DESIGN INFORMATION

- Example operation:
  - Initial state = \{C0\}, Operations: addCmd and ecexCmd
  - Final state = \{C0\}
  - These two operations come from different interfaces and have different delays

- Initial model did not follow state transition order of DUT. Model state was updated as soon as operation was observed.

- Checks were based on initial state and operations. In this case,
  - Final state should be \{C0\} within N cycles
  - All actions related to execution of C0 should be observed at output interfaces within N cycles

- Assertions could not be proved with default time settings
OPTIMIZATION 5: USING DESIGN INFORMATION

- Model was modified to add exact timing for state update relative to externally observable inputs.
  - addCmd changed model state after 3 clock
  - execCmd changed model state after 4 clocks
  - Design state was always in sync with the model

- Original assertions were retained.

- New assertion for exact state match was added.

- Jasper proved the timed model in less than 10 minutes

- Jasper proved the relaxed model assertions very quickly when using the timed model assertion proof
IMPACT ON VERIFICATION

- Makes verification coverage better. Covers detailed pipeline behavior.
- The overall checks on outputs were written based on initial state, input operations, and final state. Since intermediate state is not used for checks, and the transition sequence modification does not sacrifice verification quality.
OPTIMIZATION 6: REDUCTION OF TOKENS

- State related proofs did not converge with thousands of overall tokens.

- Approach 1: Number of tokens bounded to 64. 64 is much greater than the pipeline depth. Magic token can be any of the 64. Proofs converged in default time.

- Approach 2: Selected few magic tokens, 0, 1, 10, …, max_value. Proofs converged in default time.
IMPACT ON VERIFICATION

- Minimal impact.
- The proof ended successfully for the first time with 32 tokens. Engine N proved the property. In further proofs engine N was used.
CONCLUSION

- With the described set of optimizations the core state properties were proven.

- Other interface related properties were described in terms of model state and commands. With state coherence proven, it was easy to prove these properties.

- Jasper took less than 30 minutes for state related proofs after the optimizations. Other proofs were even faster. This greatly increase the verification pace.
MORE DETAILS OF THE VERIFICATION

- Descriptions simplifies but retains key features
- Verification had other challenges,
  - Pipeline has speculative execution to achieve high performance
  - Even with error isolation, illegitimate cases crop up.
    - Example, “head” attribute of the token is updated when its parent is deleted or relocated.
    - Jasper showed counter examples for assertions for magic token where its parent got deleted more than once.
    - Fixing the behavior of the parent is not the solution, because the tool will cook up a solutions with a misbehaving grandparent, and we will end up modeling behavior of all tokens.