A UVM Based Methodology for Processor Verification

Abhineet Bhojak, Stephan Herrmann
Tejbal Prasad
Freescale Semiconductor
Processor Verification Challenges

- Different kind of instructions and excessive number of GPRs leading to massive functional space and we need to target the pertinent
- Presence of asymmetric and out-of-order pipelines
  - Various hazards (e.g. RAW, WAR, WAW, Branches)
- Dedicated Hardware Accelerators in parallel with pipeline
- Debug hooks for the ease of debug

**Stimuli Generator**
- Most important
- Need of multiple tests

**Program generation**
- Hazard scenario
- Accelerator
- Jump & Loop cmd

**Debug**
- Debug Hooks for localization of failure
Existing Technologies

- Random test pattern generators (RTPG) and Test plan automation tools
  - Define a specific scenario description language and take as declarative input architecture and micro-architecture
  - Uses sophisticated CSP solver with bias to generate test programs
  - there is a significant learning curve involved to leverage these RTPG’s in a project schedule along with a considerable cost factor

- Formal verification
  - Useful and efficient in some cases
  - it requires significant mathematics skill and computational resources to relate to the scenarios and analyze them

- Pure directed testing
  - Gives confidence on different functionalities
  - Achieving desired coverage may take large amount of time
PROPOSED METHODOLOGY

• Efficient constrained random stimuli generation mechanism for creating meaningful and highly reusable scenarios

• Focus on running top level use cases with minimum efforts to achieve high confidence

• Reducing the debug time for better time-to-market

• A methodology for processor verification using the open sources UVM, SV & C/C++.
Proposed Stimulus Generation Flow

- A fine blend of Top Down control and Bottom layer intelligence
- Better control over random stimuli and high reuse

1. **Scenario**
   - Skeleton of the program is generated
   - Size of the program is controlled
   - Data for the program is controlled
   - Provides the Top Down Control

2. **Program Generator**
   - It does the decision making based on Scenario level information
   - It Randomizes atomic transaction based on fixed/random type or to a fixed/random instruction

3. **Atomic Transaction**
   - It randomizes all the fields and pack them into one instruction.
   - Bottom layer intelligence
     - Takes care of infinite loop
     - Does instruction operand interlinking
     - Extension for instruction grouping for better reuse
Bottom Layer Intelligence

Interesting scenario

Potential Hazard Candidate

Conventional Way
Randomize the whole program
Use foreach constraint to make relation between instruction operand

Innovative Way
Randomize one instruction at a time.
Keep copy of last few instruction.
While randomizing current instruction 1\textsuperscript{st} decide to what depth (rel\_depth) you want to link it
Use last instruction copy & rel\_depth to decide the current instruction operand

\textit{load address}_1, R3
\textit{load address}_2, R4
\textit{add} R3, R4, R5
\textit{store} R5, \textit{address}_3

How to put in this intelligence w/o complicating constraint solver?

© Accellera Systems Initiative 6
Bottom Layer Intelligence

Operand Interlinking

```java
class processor_transaction_atomic extends uvm_sequence_item;

// represent instruction groups
rand processorMasterCommandT master_cmd;
// represents individual instructions
rand processorCommandT processor_command_type;

// Variables for construction the final instruction
bit [31:0] processor_instruction;
rand bit [7:0] processor_input_operand_1;
rand bit [7:0] processor_input_operand_2;
rand bit [7:0] processor_destination_address;

constraint processor_master_command_decoding {
    (master_cmd) inside
    { /*define the valid instruction groups here */};
    (master_cmd == MASTER_ADD) ->
    processor_command_type == PROCESSOR_CMD_S_ADD;
    (master_cmd == SCALAR_COMMANDS) -> (processor_command_type inside
    {PROCESSOR_CMD_S_MOV,PROCESSOR_CMD_S_ADD});
}

constraint processor_operand_interlinking {
    (dest_op_relation) inside { INDEP, DEPO, DEPI }
    (dest_op_relation == INDEP) -> {};
    (dest_op_relation == DEPO) ->
    {processor_destination_address == last_operand_type};
    (dest_op_relation == DEPI ) ->
    {processor_destination_address == 2nd_last_operand_type};
}

function generate_instruction_data();
    if(processor_command_type == PROCESSOR_CMD_S_ADD)
    begin
        processor_instruction[23:16] = processor_destination_address;
        processor_instruction[15:8] = processor_input_operand_1;
        processor_instruction[7:0] = processor_input_operand_2;
    end
endfunction : generate_instruction_data

endclass : processor_transaction_atomic
```
Proposed flow in action

Loop:

MOV R1, VAL

JMP LEN1

SUB R1, 1

BZ End

JMP LEN2

SUB R1, 1

BNZ Loop

HALT

Jump Length constraints

INSTRUCTION GROUPS

BRANCH
• BEQ
• BLT
• BGT
• BN
• BC

ALU
• ADD
• SUB
• XOR
• OR
• CMP

FILL

Jump Length constraints

W-MARK

LEN2

LENGT H

Infinite Loop Avoidance
Debugging Hooks

- Zero time Reference model Vs pipelined processor
- Checking only at interfaces is not enough for complex scenarios
- Register trace queue based Checker

- Out-of-order execution of pipeline Vs In-order execution of model
- Need checker based on register content change – Data trace checker

- To get the desired confidence running directed use cases is a must
- Switch based flow for directed stimulus which uses program, data/images and configuration as file based input

Debug cannot be an afterthought.

Localization of Failure

Scenario Replication
Debugging Hooks
Evaluation of the Proposed Flow

Design Complexity
Scalar, Vector & Matrix operation, 9 ALUs, 4 Multiplier, ~256 GPRs & Hardware Accelerator like SORT, HISTOGRAM etc

Verification
30 man weeks of effort, Verification Environment created from Scratch, ~200 test /15 K runs, ~10 k functional cover points, 200 odd defects were found

First Pass Success
No additional bugs found after IP signoff.
Silicon has been evaluated - considered to be a first pass success.
Thank You
Q & A