BUG REPORT

Team Members:
Poornima Gurushanthappa
Sourav Sudhir
Prateek Sasikumar
Ganesh Prabhu Sathyanarayana

Bug 1

1. Failing Test name: mesi_state

2. Test description: Checks for all MESI FSM state transitions. Bug was found when Modified to Shared transition was triggered. A write to address 0x4000004 was issued from CPU0 making the MESI state to modified and then a read to the same address was issued from CPU1. This should trigger a lv2 write from CPU0 and then the data needs a read from CPU1.

3. Failing checker/assertion that helped you identify the bug:
   Score Board failure, when Multiple bus activity was observed for CPU1 i.e., multiple reads were observed.

4. Brief description of your diagnosis: When a read miss happened for CPU1, bus_rdx signal was made high on CPU0 instead of bus_rd. bus_rd should be made high, since CPU1 issued a read and not a write. After observing this, the bus_rd and bus_rdx signals were traced back and found that when inst_cache_lv1_unicore_1 was instantiated, bus_rd and bus_rdx signal assignment were swapped.

5. Erroneous RTL file name: cache_lv1_multicore.sv

6. Lines of RTL file responsible for the bug
   Lines 175 and 176
   175 .bus_rd(bus_rdx),
   176 .bus_rdx(bus_rd),

7. Corrected RTL code
   175 .bus_rd(bus_rd),
   176 .bus_rdx(bus_rdx),

Bug 2

1. Failing Test name: mesi_state

2. Test description: Checks for all MESI FSM state transitions. Bug was found when Modified to Invalid transition was triggered. A write to address 0x4000004 was issued from CPU0 making the MESI state to modified and then a write to the same address was issued from CPU1. This should trigger a lv2 write from CPU0 and then the modified data needs to be read to CPU1 cache and new data should be written. MESI state should be update from Modified to Invalid Transition

3. Failing checker/assertion that helped you identify the bug :
4. $past(current_mesi_snoop == MODIFIED) && $rose(bus_rdx) |=> (updated_mesi_snoop == INVALID);

5. Brief description of your diagnosis:
6. Mesi FSM state machine code was checked for the case when bus_rdx signal is made high and the current_mesi_snoop is in Modified state, it was observed that the updated_mesi_snoop was Modified state.

7. Erroneous RTL file name : mesi_fsm_lvl1.sv

8. Lines of RTL file responsible for the bug
9. Line 109
10. updated_mesi_snoop = MODIFIED;

11. Corrected RTL code
12. updated_mesi_snoop = INVALID;

Bug 3

1. Failing Test name: read_miss_icache

2. Test description: Issue a read to ICACHE and the data needs to be read from lv2 cache.

3. Failing checker/assertion that helped you identify the bug:
   ```
   cpu_rd |-> ##[1:$](|data_bus_cpu_lv1);
   ```

4. Brief description of your diagnosis: Data was not being read when read was issued to Icache. When signals at *_il (instruction level) were checked it was observed the cpu_rd_il was not being asserted. Design files were checked to find places where cpu_rd_il was being modified and found that the assign condition was not right.

5. Erroneous RTL file name: cache_lv1_unicore.sv

6. Lines of RTL file responsible for the bug
   Line81
   ```
   assign cpu_rd_il = (addr_bus_cpu_lv1 <= (IL_DL_ADDR_BOUND & 32'h0fffffff))? cpu_rd : 1'b0;
   ```

7. Corrected RTL code
   ```
   assign cpu_rd_il = (addr_bus_cpu_lv1 <= IL_DL_ADDR_BOUND)? cpu_rd : 1'b0;
   ```

Bug 4

1. Failing Test name: read_miss_icache

2. Test description: Issue a read to ICACHE and the data needs to be read from lv2 cache.

3. Failing checker/assertion that helped you identify the bug:
   $rose(lv2_rd) |-> ##[1:$] (|data_bus_lv1_lv2);

4. Brief description of your diagnosis:
   The assertion indicated that no data was being read from Lv2 cache even though data_in_bus_lv1_lv2 was made high. So we checked if all the steps are being followed on a read miss. On cache read miss, bus_lv1_lv2_req_proc_il was made high and grant was provided by the arbiter by asserting bus_lv1_lv2_gnt_proc. Even though lv2_rd bus was asserted, but data was not being read on the data_bus_lv1_lv2. data_in_bus_lv1_lv2 was not driven to 1 on Simvision. When we traced back the signals we found that data_in_bus_lv1_lv2 was tied to 1 when main_func_lv1_il was instantiated in cache_block_lv1_il.sv file.

5. Erroneous RTL file name: cache_block_lv1_il.sv

6. Lines of RTL file responsible for the bug
   Line 171
   171: .data_in_bus_lv1_lv2 (1'b1)

7. Corrected RTL code
   Line 171
   .data_in_bus_lv1_lv2 (data_in_bus_lv1_lv2)

Bug 5

1. Failing Test name: read_miss_icache

2. Test description: Issue a read to ICACHE and the data needs to be read from lv2 cache.

3. Failing checker/assertion that helped you identify the bug:
   $rose(data_in_bus_cpu_lv1_il) |-> ##[1:$] (|data_bus_cpu_lv1);
   Score board error indicating expected and read data don't match.

4. Brief description of your diagnosis:
   The assertion indicated that no data was being read on data_bus_cpu_lv1 even though data_in_bus_cpu_lv1_il high. After fixing bug 4 where data was not being read on data_bus_lv1_lv2, we were able to observe the data being read from lv2 cache into data_bus_lv1_lv2. However, the data was not propagated to data_bus_cpu_lv1 after read hit which follows once read miss is addressed. On checking all the signals, we realised that all the steps are being followed for both read miss and read hit case. On a read miss, the read data is updated in cache_var and the same is used to write data to data_bus_cpu_lv1 on read hit. The issue was narrowed down to some problem with updation of cache_var. When the design file was checked, we found that the width of cache_var was wrong and that being the reason for data not being written to lv1 cache.

5. Erroneous RTL file name: main_func_lv1_il.sv

6. Lines of RTL file responsible for the bug
   Line 59
   59 : reg [CACHE_DATA_WID - 1   : 0] cache_var   [0 : CACHE_DEPTH/4 - 1];

7. Corrected RTL code
   Line 59
   59 : reg [CACHE_DATA_WID - 1   : 0] cache_var   [0 : CACHE_DEPTH - 1];

Bug 6

1. Failing Test name: mesi_state

2. Test description Checks for all MESI FSM state transitions. Bug was found when Exclusive to Modified transition was triggered. A read to address 0x40000400 was issued from CPU0 making the MESI state to exclusive and then a write to the same address was issued from CPU0. MESI state should be updated from Exclusive to Modified state.

3. Failing checker/assertion that helped you identify the bug :
   $past(current_mesi_snoop == Exclusive) && $rose(cpu_wr) |=> (updated_mesi_snoop == MODIFIED);

4. Brief description of your diagnosis:
   Mesi FSM state machine code was checked for the case when cpu_wr signal is made high and the current_mesi_snoop is in Exclusive state, it was observed that the updated_mesi_snoop remained in Exclusive state.

5. Erroneous RTL file name : mesi_fsm_lvl1.sv

6. Lines of RTL file responsible for the bug
   Line 72
   updated_mesi_proc = EXCLUSIVE;

7. Corrected RTL code
   Line 72
   updated_mesi_proc = MODIFIED;

Bug 7

1. Failing Test name: mesi_state

2. Test description: Checks for all MESI FSM state transitions. Bug was found when Shared to Invalid (on snoop side) and Shared to Modified transition (on proc side). A read to address 0x40000500 was issued from CPU0 and a read to same address was issued by CPU1 making the mesi state to shared. Then a write to the same address was issued from CPU0. Data should be read from both CPU0 and CPU1 on read and data should be written to CPU0 cache on write.

3. Failing checker/assertion that helped you identify the bug:
   $rose(cpu_wr) |=> ##[1:$]$rose(cpu_wr_done);
   $rose(invalidate) |=> ##[1:$] $rose(all_invalidation_done);

4. Brief description of your diagnosis:
   Assertion failures indicated that CPU write was not happening on CPU0 and all_invalidation_done was not being asserted even though a invalidate signal was being asserted. According to the HAS document, on the snoop side when there is a block hit (blk_hit_snoop being asserted), if invalidate is high, then invalidation_done should be made high. This was not happening. The code for Snoop side invalidation request was checked and the condition being checked was if(invalidate & !invalidation_done) .The problem here was invalidation_done was initialised to z and it was never being driven to 0 anywhere. So if(invalidate & !invalidation_done) always failed and again the snoop invalidate request was not being processed.

5. Erroneous RTL file name : main_func_lv1_dl.sv

6. Lines of RTL file responsible for the bug
   Line 130
   invalidation_done   <= 1'bz;

7. Corrected RTL code
   Line 130
   invalidation_done   <= 1'b0;

Bug 8

1. Failing Test name: mesi_state

2. Test description: Checks for all MESI FSM state transitions. Bug was found when Shared to Invalid (on snoop side) and Shared to Modified transition (on proc side). A read to address 0x40000500 was issued from CPU0 and a read to same address was issued by CPU1 making the mesi state to shared. Then a write to the same address was issued from CPU0. Data should be read from both CPU0 and CPU1 on read and data should be written to CPU0 cache on write.

3. Failing checker/assertion that helped you identify the bug :
   $rose(cpu_wr) |=> ##[1:$]$rose(cpu_wr_done);
   $rose(invalidate) |=> ##[1:$] $rose(all_invalidation_done);

4. Brief description of your diagnosis:
   Assertion failures indicated that CPU write was not happening on CPU0 and all_invalidation_done was not being asserted even though a invalidate signal was being asserted. After fixing the bug7(invalidate_done signal not being asserted), invalidation_done was getting asserted for all other CPUs (CPU1, CPU2 and CP3) but all_invalidation_done was not being asserted. So, we back traced the assignment of all_invalidation_done signal and found that the assignment statement was wrong. For CPUs for which bus_lv1_lv2_gnt_proc is raised, invalidate_done is not asserted and for bus_lv1_lv2_gnt_proc is not raised, invalidate_done is asserted. So this would make the all_invalidation_done to 0, even thought invalidation was done for all CPUs.

5. Erroneous RTL file name : cache_lv1_multicore.sv

6. Lines of RTL file responsible for the bug
   Line 79
   assign all_invalidation_done = | (invalidation_done & bus_lv1_lv2_gnt_proc);

7. Corrected RTL code
   Line 79
   assign all_invalidation_done = & (invalidation_done | bus_lv1_lv2_gnt_proc);

Bug 9

1. Failing Test name: lru_replacement

2. Test description: Checks for the pseudo-LRU replacement policy at Cache Lv1. Write requests to addresses 0x40000400, 0x45000400, 0x50000400, 0x55000400 were issued from CPU0 and read to address 0x60000400 was issued from CPU0. This should force a eviction of one of the blocks from Cache since index of all addresses are same, with different tag. The block that should be evicted should be 0x40000400 according to the pseudo LRU replacement policy implemented.

3. Failing checker/assertion that helped you identify the bug:
   Score board failure, proc_evict_dirty_blk_addr and proc_evict_dirty_blk_data mismatch between expected and observed packets.

4. Brief description of your diagnosis:
   Scoard board failure indicated that the expected block to be replaced didn't match the block replaced by the LRU unit. blk_accessed_main and lru_replacement_proc signals were monitored to check for the lru state transitions. According to the HAS document, the access sequence of blk_accessed_main should result in eviction of Block3. However, Block 0 was evicted. When block_acessed_main was 0, the lru_replacement_proc should be 3, but it was asserted to 0. Upon checking the design file, we found that the state definition of BLK3_REPLACEMENT was wrong.

5. Erroneous RTL file name : lru_block_lv1.sv

6. Lines of RTL file responsible for the bug
   Line 29
   parameter BLK3_REPLACEMENT = 3'bx01;

7. Corrected RTL code
   Line 29
   parameter BLK3_REPLACEMENT = 3'b1x1;