An Automated and Scalable Monitor-and-Checker solution for SMMU Verification in Multi-Die SoCs

Junha Jeon, Namyoung Kim, Hyunman Park, Jaein Hong, Moonki Jun, Sungcheol Park, Sanghune Park

Foundry Business, Samsung Electronics Co., Ltd.
1, Samsung-ro, Giheung-gu, Yongin-si, Gyeonggi-do, Republic of Korea, 17113

Email: **junha96.jeon@samsung.com**

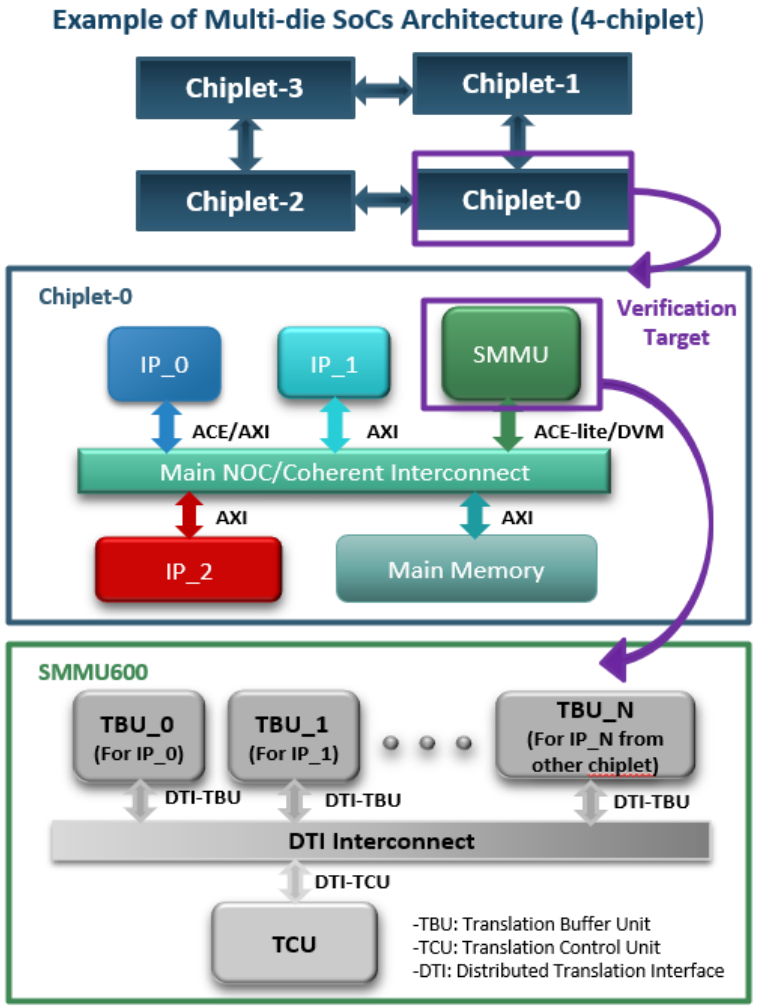
62 THE CHIPS TO SYSTEMS CONFERENCE

SAMSUNG

SPONSORED BY CEDA® IEEE Council on Electronic Design Automation SIGDA acm

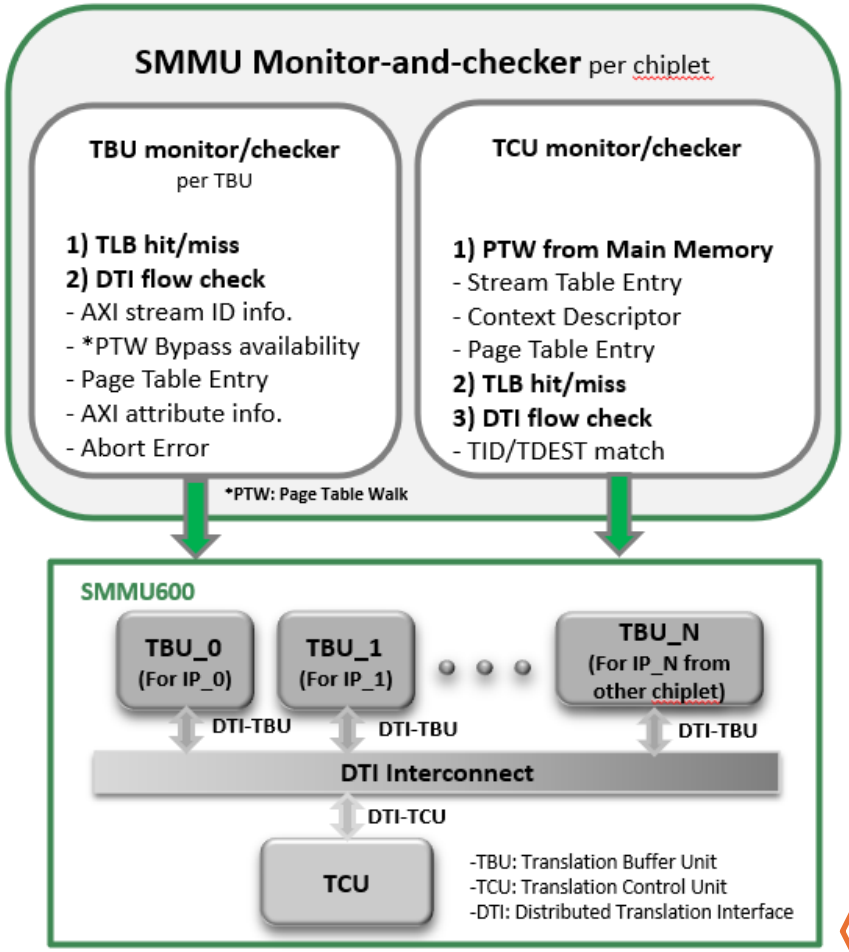
Motivation

- **Multi-Die SoCs amplify verification complexities**, since each die has its own interconnects and resources that must work together seamlessly within one system
- Configuring **TBU,TCU and DTI interconnect of SMMU600 to each SoC architecture is especially challenging** in Multi-Die scenarios
- Existing Solution often do not catch important issues like DTI data integrity problems or TLB hit/miss accuracy because **the difficulty of designing DTI interconnect architecture has increased**
- An automated and scalable monitor-and-checker solution addresses gaps in existing methods, **supports diverse SMMU configurations, and improves verification coverage and system reliability**



SMMU Monitor-and-Checker Solution

- **A Monitor for real-time tracking** and **a Checker to ensure configuration accuracy and data integrity**
- **Continuous monitoring and checking of functions throughout the test**
  - Check the Page Table Walk (PTW), AXI attributes (AxCACHE, AxDOMAIN), and DTI data integrity
- **Enhance the verification of exception error scenarios and handling**
  - Check abort errors for all incoming transactions to the TBU slave
- **Performance monitoring and reporting**
  - Assist in checking TLB hit ratio and other factors to assess the performance effect
  - Provide the trace log of the performance scenario for the performance verification report



Practical Usage of the Monitor-and-Checker

- **Monitor-and-Checker SystemVerilog code snippet for TBU**

* Stream ID

– Save AXI stream ID info. regarding all transaction incoming to TBU slave interface

```
always @(negedge tbu_clk) begin
    if(tbu_tbs_awvalid && tbu_tbs_awready) begin
      if(TR_num > ACCEPT_NUM) begin
        TR_num = TR_num-ACCEPT_NUM;
      end
      else begin
        TR_num++;
      end
      TLB_hit_state[TR_num] = 0;
      awaddr_s[TR_num] = tbu_tbs_awaddr;
      awmmusid_s[TR_num] = tbu_tbs_awmmusid;
      awmmussid_s[TR_num] = tbu_tbs_awmmussid;
      awmmussidv_s[TR_num] = tbu_tbs_awmmussidv;
      awmmusecsid_s[TR_num] =tbu_tbs_awmmusecsid;
```

- SID: Stream ID
- SSID: Sub-stream ID
- SSIDV: Sub-stream ID valid
- SECSID: Secure Stream ID

* TBU TLB hit/miss

– Check whether TBU TLB hit or miss has occurred (DTI request would be sent to TCU if TLB miss occurs)

```
@(negedge tcu_clk)
      do begin
        if(!tcu_aresetn) begin
          $display("[SMMU MONITOR] TCU reset de-asserted by RESET triggering,
can't do QTW operation, Checker break from now");
          TLB_hit_state[TR_num] = 2'h2;
          break;
        end
        else if((tbu_tbm_awaddr == awaddr_s[TR_num]) && tbu_tbm_awvalid) begin
          $display("[%t][SMMU MONITOR] VA(0x%x) address is TBU TLB hit, don't
need to request TCU", $time, awaddr_s[TR_num][47:00]);
          TLB_hit_state[TR_num] = 2'h1;
          break;
        end
        #(TCU_FREQ);
      end while(!(tcu_tvalid_dti_dn && tcu_tready_dti_dn && (tcu_tid_dti_dn ==
DTI_TID_DN) && (tcu_tdata_dti_dn[3:0] == 4'h2)));
```

Practical Usage of the Monitor-and-Checker

- **Monitor-and-Checker SystemVerilog code snippet for TBU**

*** DTI flow check**

1) Check DTI request data for AXI stream ID information

- Checker make error log if DTI data is different with stream ID

```
dti_tdata_dn[TR_num] = tcu_tdata_dti_dn;
if(awmmusid_s[TR_num] != dti_tdata_dn[TR_num][63:32]) begin
  $error("[%t][SMMU MONITOR] Incoming AWMMUSID(%x) is different with
DTI_TDATA_DN(%x)", $time, awmmusid_s[TR_num], dti_tdata_dn[TR_num][63:32]);
end
if(awmmussid_s[TR_num] != dti_tdata_dn[TR_num][95:76]) begin
  $error("[%t][SMMU MONITOR] Incoming AWMMUSSID(%x) is different with
DTI_TDATA_DN(%x)", $time, awmmussid_s[TR_num], dti_tdata_dn[TR_num][95:76]);
end
```

2) Check DTI response data for PTW and AXI attribute

- Monitor confirm PTW bypass availability and PTW

- If page table walk enables, check AXI attribute like cacheability/ shareability

```
if(dti_tdata_up[TR_num][99:96] == 4'b1111) begin
  $display("[%t][SMMU MONITOR] Cacheability: Inner Write-Back non-transient, R/W
allocation", $time);
end
else if(dti_tdata_up[TR_num][99:96] == 4'b1100) begin
  $display("[%t][SMMU MONITOR] Cacheability: Inner Write-Back non-transient, R/W
no-allocation", $time);
end
else if((dti_tdata_up[TR_num][99:98] == 2'b10)|(dti_tdata_up[TR_num][99:98] ==
2'b00)) begin
  $display("[%t][SMMU MONITOR] Cacheability: Inner Write-Through", $time);
end
else if(dti_tdata_up[TR_num][99:98] == 2'b01) begin
  $display("[%t][SMMU MONITOR] Cacheability: Inner Non-cacheable", $time);
end
```

Practical Usage of the Monitor-and-Checker

- **Monitor-and-Checker SystemVerilog code snippet for TBU**

* Issuing TR via TBU master interface

Check the success of transaction issuing from TBU master I/F

- If PTW abort or SMMU hang occurs, it would be failed

```
do begin
  #(TBU_FREQ);
  current_time_1 = $realtime;
  if(current_time_1 - start_time_1 > limit_time) begin
    $display("[%t][SMMU MONITOR] Timeout error of TBM issuing transaction: VA(%x)", $time, tbu_tbs_awaddr);
    break;
  end
  else if((tbu_tbm_awaddr != awaddr_s[TR_num]) && tbu_tbm_awvalid && tbu_tbm_awready) begin
    $error("[%t][SMMU MONITOR] VA(%x) to PA(%x) write TR fail!!", $time, awaddr_s[TR_num], tbu_tbm_awaddr);
  end
end while(!((tbu_tbm_awaddr == awaddr_s[TR_num]) && tbu_tbm_awvalid && tbu_tbm_awready));
```

* PTW abort check

Check PTW abort error from DTI response data

- TCU could not get the page table from main memory

```
do begin
  #(TCU_FREQ);
  current_time = $realtime;
  if(current_time-start_time > limit_time) break;
end while(!((tcu_tvalid_dti_up) && (tcu_tready_dti_up) && (tcu_tdest_dti_up == DTI_TDEST_UP)));
dti_tdata_up[TR_num] = tcu_tdata_dti_up;
//DTI resp context
if((dti_tdata_up[TR_num][3:0] == 4'h1) && tcu_tvalid_dti_up) begin
  $error("[%t][SMMU MONITOR] TCU abort error -> Check if address fault occur", $time);
end
```

Practical Usage of the Monitor-and-Checker

- **Monitor-and-Checker SystemVerilog code snippet for TCU**

* TCU TLB hit/miss

Save page table from main memory to TCU

```
rdata_qtw[PT_req_num] = tcu_rdata_qtw;
if(rdata_qtw[PT_req_num][1:0] == 2'h1) begin
  axaddr_PTE[PT_req_num][47:21] = rdata_qtw[PT_req_num][47:21]; //[23:21]
descriptor address value = 0
end
else if(rdata_qtw[PT_req_num][65:64] == 2'h1) begin
  axaddr_PTE[PT_req_num][47:21] = rdata_qtw[PT_req_num][111:85];
end
else if(rdata_qtw[PT_req_num][129:128] == 2'h1) begin
  axaddr_PTE[PT_req_num][47:21] = rdata_qtw[PT_req_num][175:149];
end
else if(rdata_qtw[PT_req_num][193:192] == 2'h1) begin
  axaddr_PTE[PT_req_num][47:21] = rdata_qtw[PT_req_num][239:213];
end
```

Check whether TCU TLB hit or miss

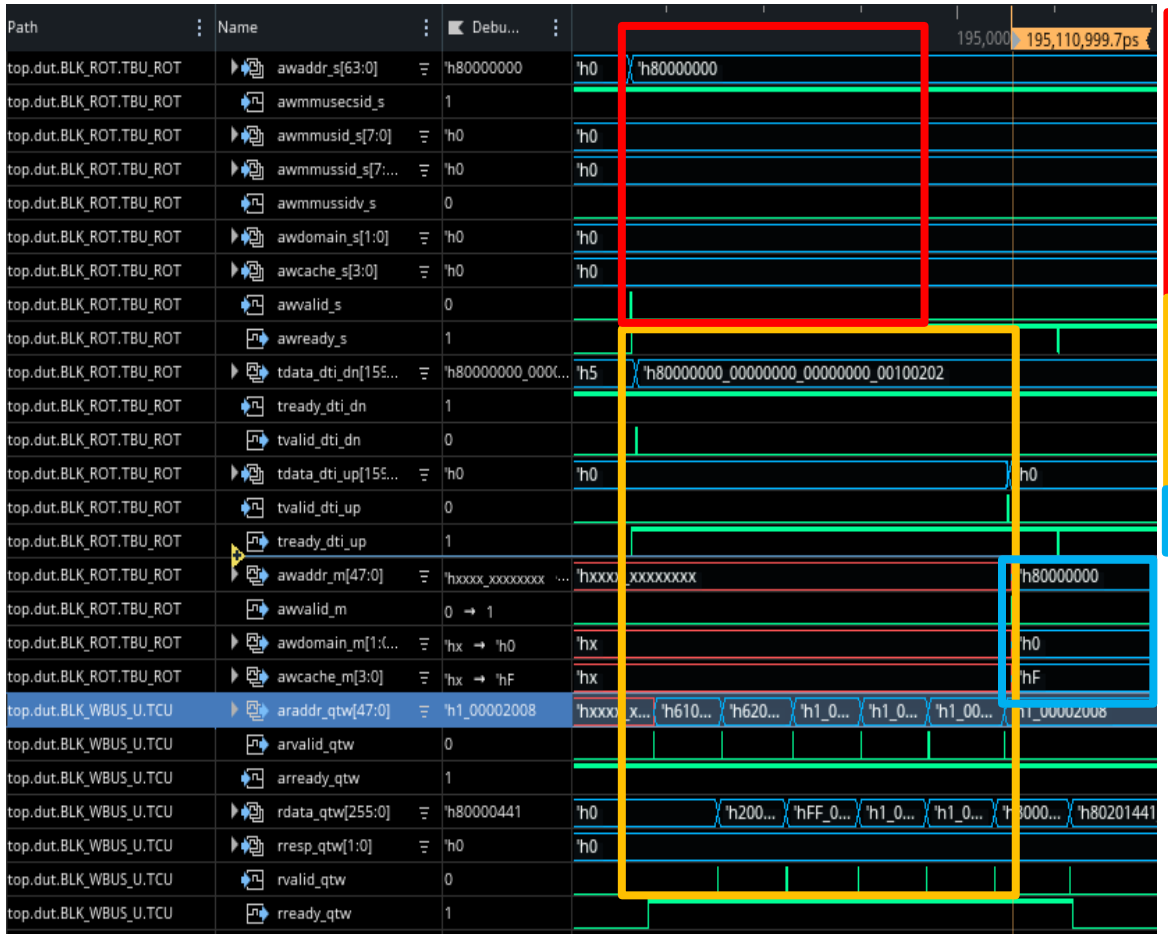
Error log would be printed if TCU already have any specific page table regarding previously accessed address range

```
always @(negedge tcu_clk) begin
  if((tcu_tdata_dti_dn[3:0] == 4'h2) && tcu_tvalid_dti_dn && tcu_tready_dti_dn) begin
    for(i=1; i<=PT_req_num; i++) begin
      if(tcu_tdata_dti_dn[143:117] == axaddr_PTE[PT_req_num]) begin
        $display("[%t][SMMU MONITOR] VA(%x) of DTI_TID_DN(%x) is TCU TLB hit",
$time, tcu_tdata_dti_dn[143:96], tcu_tid_dti_dn);
        TCU_TLB_hit = 1;
      end
    end
  end
…
    do begin
      $error("[%t][SMMU MONITOR] VA(%x) of DTI_TID_DN(%x) is TCU TLB hit, but
PTW happened", $time, tcu_tdata_dti_dn[143:96], tcu_tid_dti_dn);
      …
      #(TCU_FREQ);
    end while(!((tcu_tdest_dti_up == tcu_tid_dti_dn) && (tcu_tdata_dti_up == 4'h2) &&
tcu_tvalid_dti_up && tcu_tready_dti_up));
…
end
```

Practical Usage of the Monitor-and-Checker

- **Continuous monitoring and checking of SMMU function**

Case: Write, TBU/TCU TLB miss, AXI attribute override



Monitoring/Checking result

```
            194326][SMMU MONITOR] WRITE REQ ADDR: 0x000080000000 reached to TBU_ROT
SMMU checker] TBU requests via DTI interface
194363.000ns][SMMU MONITOR] TLB_hit_state[         1] = 0
194363.000ns][SMMU MONITOR] TBU_ROT requests Attribute/Page Table info for write transaction!
194363.000ns][SMMU MONITOR] AWMMUSID = 00
194363.000ns][SMMU MONITOR] AWMMUSSID = 00
194363.000ns][SMMU MONITOR] AWMMUSSIDV = 0
194363.000ns][SMMU MONITOR] AWMMUSECSID = 1
194363.000ns][SMMU MONITOR] Virtual Address = 0000000080000000
SMMU checker] QTW happens at        714ns from the time when traffic reaches to TBS.
195076] UVM_INFO (smmu_cfg_c) [SMMU checker] SMMU QTW operation checker is ended.
            195076][SMMU MONITOR] axaddr_PTE[00000001]: 0x000080000000 fill in TCU TLB
            195080][SMMU MONITOR] TBU_ROT get the page table from TCU, (Page Table: 0x080000000)
            195080][SMMU MONITOR] Shareability: Non-shareable
            195080][SMMU MONITOR] Cacheability: Inner Write-Back non-transient, R/W allocation
195112.000ns][SMMU MONITOR] TR_num:          1
195112.000ns][SMMU MONITOR] VA(0000000080000000) to PA(000080000000) write TR success
```

1) 0x80000000 write TR arrives at TBU slave interface

2) TCU PTW for 0x80000000
Upstream DTI have page table entry/attribute info.

3) 0x80000000 write TR occurs via TBU master interface
AWCACHE changes to 'hF(Write back, R/W allocation)

Practical Usage of the Monitor-and-Checker

- **Performance Monitoring and Reporting**

Case: Read, TBU TLB hit, AXI attribute override

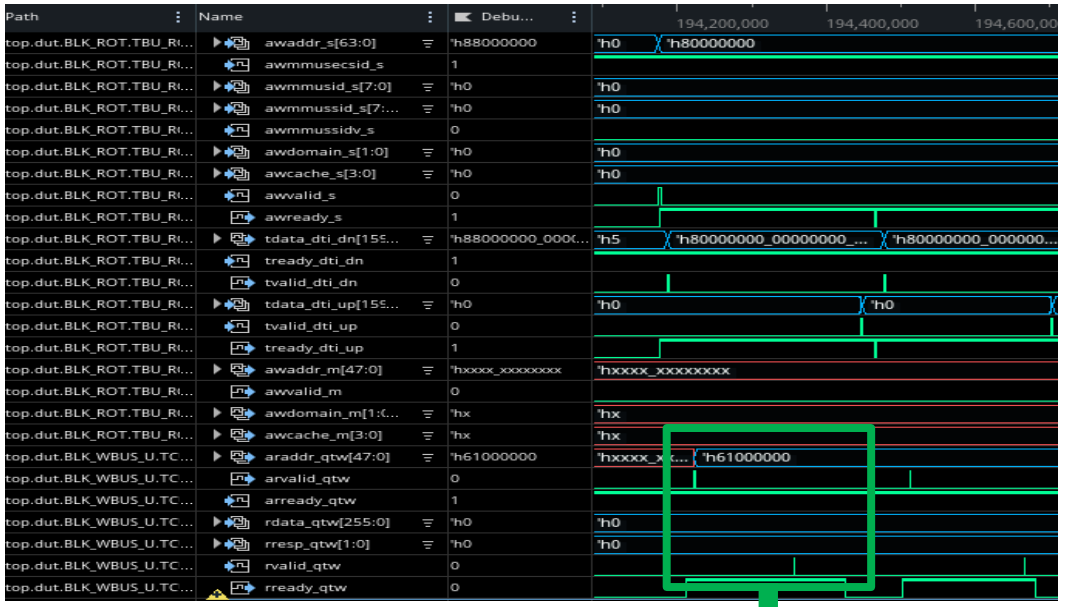
Monitoring result

```
195208][SMMU MONITOR] READ REQ ADDR: 0x000080000000 reached to TBU_ROT
195210][SMMU MONITOR] VA(0x000080000000) address is TBU TLB hit, don't need to request TCU
195210][SMMU MONITOR] TLB_hit_state[          2] = 1
195210][SMMU MONITOR] TR_num:          2
195210][SMMU MONITOR] VA(0000000080000000) to PA(000080000000) read TR success
```

0x80000000 write TR occurred before read: TBU TLB hit
Don't need PTW of TCU

- **Exception Error Handling**

Case: Abnormal Page Table Walk



Checking result

```
xmsim: *E,ERRSEV (/user/rebelhv5/WS/junha96.jeon_REBELH_ws_21/VERIFICATION/lib/common/smmu_ptw_monitor.sv,174
incr_top.TBU_ROT_ptw_monitor
[194425.000ns][SMMU MONITOR] TCU abort error -> Check if address fault occur
[194463] UVM_ERROR (uvm_sequence_item) [AXI] response err, expected: okay response
```

Bad STE(Stream Table Entry) has an effect on abnormal PTW
TCU informs PTW abort error to TBU via DTI upstream message

Achievement

- **Improvement of SMMU verification quality:**

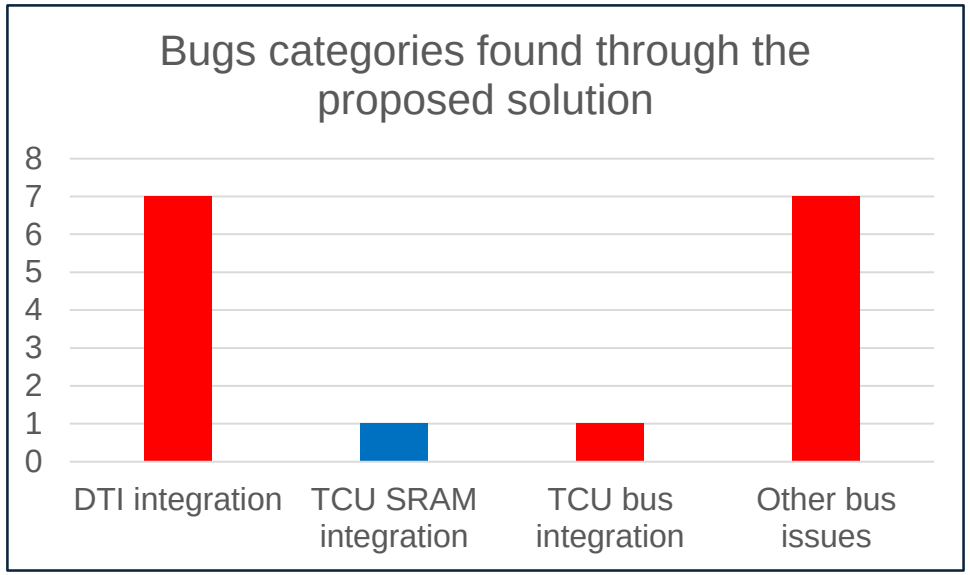
  The new SMMU verification methodology significantly contribute to identifying **16 design bugs related to SMMU** on Multi-Die project.

  1) Function bug: DTI/TCU/Other bus integration

  There are two main causes: one is an integration issue of DTI interconnect, which caused problems with the transmission of down/upstream data, and the other is a bus integration issue for PTW of TCU.

  2) Performance bug: TCU SRAM integration

  Developed solution monitor all transactions, making it useful for identifying performance defects. A bug was discovered when multiple TBUs requested PTW from the TCU, and the cache at each level was supposed to be filled, but wasn't.



Summary

- **Reduction of SMMU verification TAT:**

  The proposed SMMU solution enabled **the quick detection of bus integration and performance bugs**, leading to a notable reduction in verification time.

  1) Integration bug detection

  It is very important to finalize SMMU backbone design before SoC block freeze, as it affects all IPs that use SMMU. In Multi-Die architecture, early SMMU bug detection was more challenging, but the solution actually contributed to **shortening TAT by more than 3 weeks.**

  2) Performance bug detection

  Verifying performance in complex usage scenarios of Multi-Die is becoming an even greater challenge. By quickly detecting performance bug that was difficult to identify with manual debug approach, we were able to **cut verification TAT by 2 months.**