Verifying C++ Firmware Sequences In UVM Environment

Ashwini Holla
AMD, Markham

October 1st, 2015
Ottawa

Agenda

Introduction

Testbench Updates

Firmware Vs RTL Verification

Scoreboard For Firmware

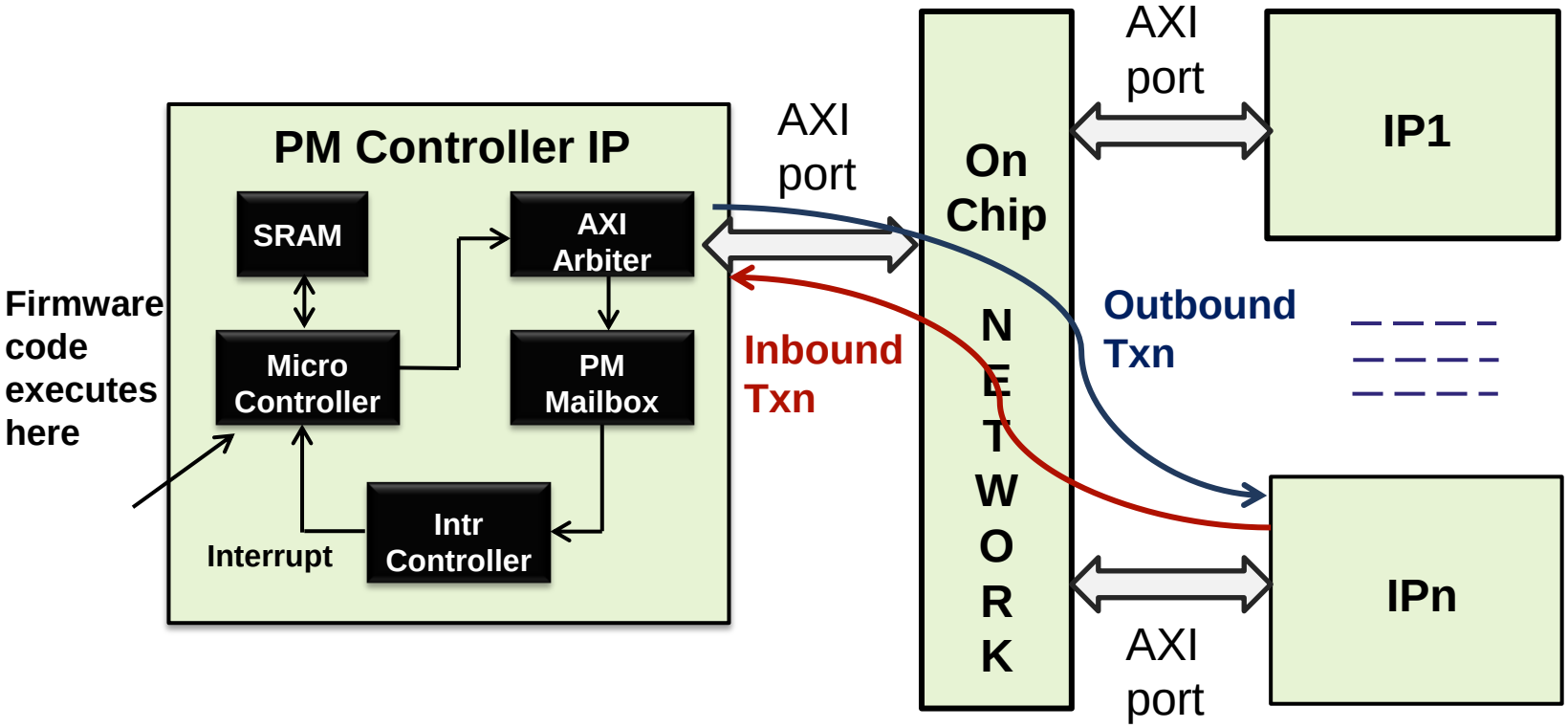
Conclusion

Introduction

- What is firmware?
  - C++ code executing on microcontroller
- Power management firmware in AMD designs
  - Dynamically manages the power states in the entire design
- Need for VCS verification
  - Firmware is an AXI agent
  - Involves interactions with multiple components
  - Needs to be simulated in VCS with other agents/sub-blocks to verify datapath and connectivity in the system
  - Post Si bring-up times are increasing steadily due to unqualified firmware

Introduction

- Firmware as an AXI agent in the system



Introduction

- Verification platforms for firmware
  - VCS
  - Instruction set simulator (simulation model for the micro-controller)
  - Post silicon bring-up

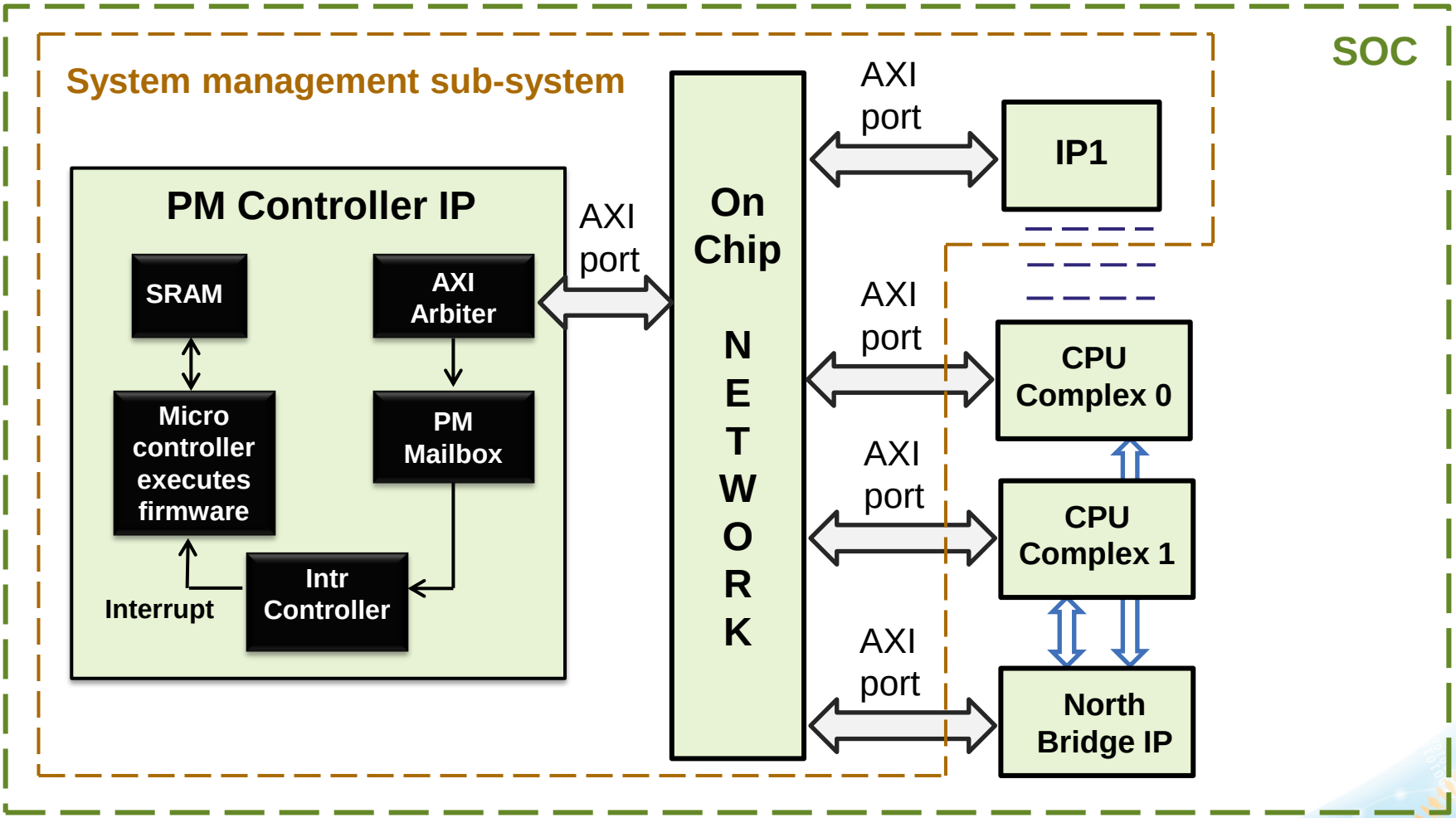
Platform	**Cycle Accurate**	**Debug Turnaround**	**What is covered?**
VCS/UVM testbench	Yes	Medium	Hardware interactions
Instruction Set Simulator	No	Fast	Mathematical computations
Post-Silicon bring-up	Yes	Medium	Real time scenarios

Testbench Updates

Testbench Updates

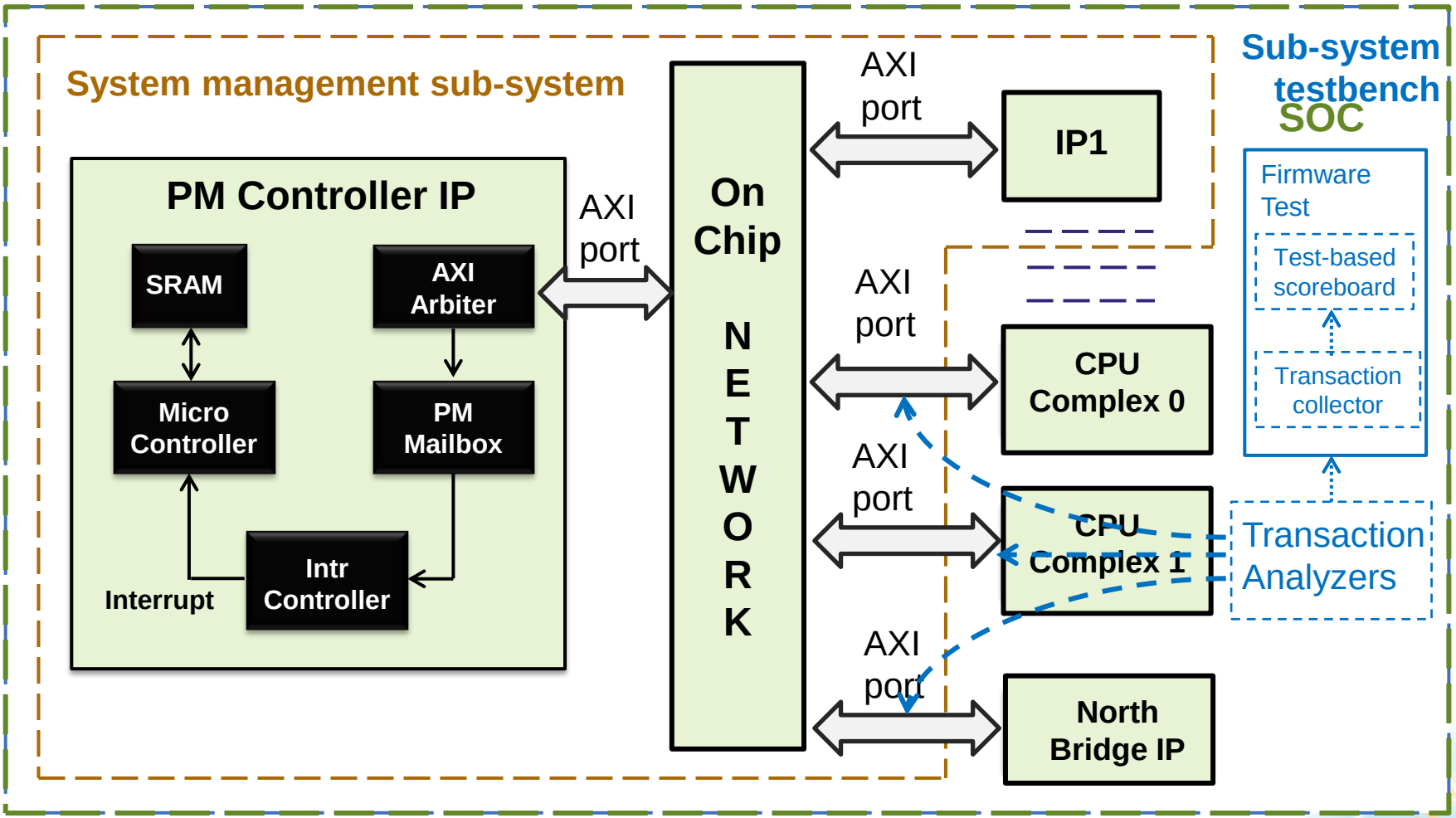
Verification Scope

- Firmware development begins in system management sub-system



Testbench Updates

System management sub-system verification



Testbench Updates

Testbench components

- AXI BFM
  - Mimics the actions of an AXI agent on the network
  - Can initiate read/write to an AXI agent and receive read/write from an AXI agent
  - Pre-developed verification component
  - Easy modelling of non-RTL components

- Transaction Analyzer
  - Generic implementation using uvm_subscriber class
  - Used for detecting transactions at key AXI interfaces and qualifying firmware sequence
  - Used for determining when to set up response registers in AXI BFMs (to mimic IP behavior)

Testbench Updates

Transaction analyzer using uvm_subscriber class

```
class core_axi_slv_export #(type T = axi_write_req_txn, type
parent_ptr = firmware_top_env) extends uvm_subscriber;
parent_ptr ptr;
bit [`NUM_CORE_PER_COMPLEX-1 : 0] core_id;
bit [`NUM_OF_COMPLEXES-1 : 0] complex_id;
…
…

ptr.core_prefill_queue[complex_id][core_id].push_back
                                              (transaction);

endfunction
endclass : core_axi_slv_export
```

Uniquely identify the subscriber component

Populate component variables in an iterative structure from the parent class

Use in a forever loop in base test to process transactions at key interfaces

Firmware Vs RTL Verification

Firmware Vs RTL Verification

- Similarities
  - Testbench infrastructure (firmware, an AXI agent is RTL)
  - Stimulus in the tests
  - Verification for datapath and connectivity

- Differences
  - Verification for correctness is the distinguishing factor
  - Firmware qualification is a sequence check
  - Cannot be covered with IP checkers

Scoreboard For Firmware

Scoreboard For Firmware

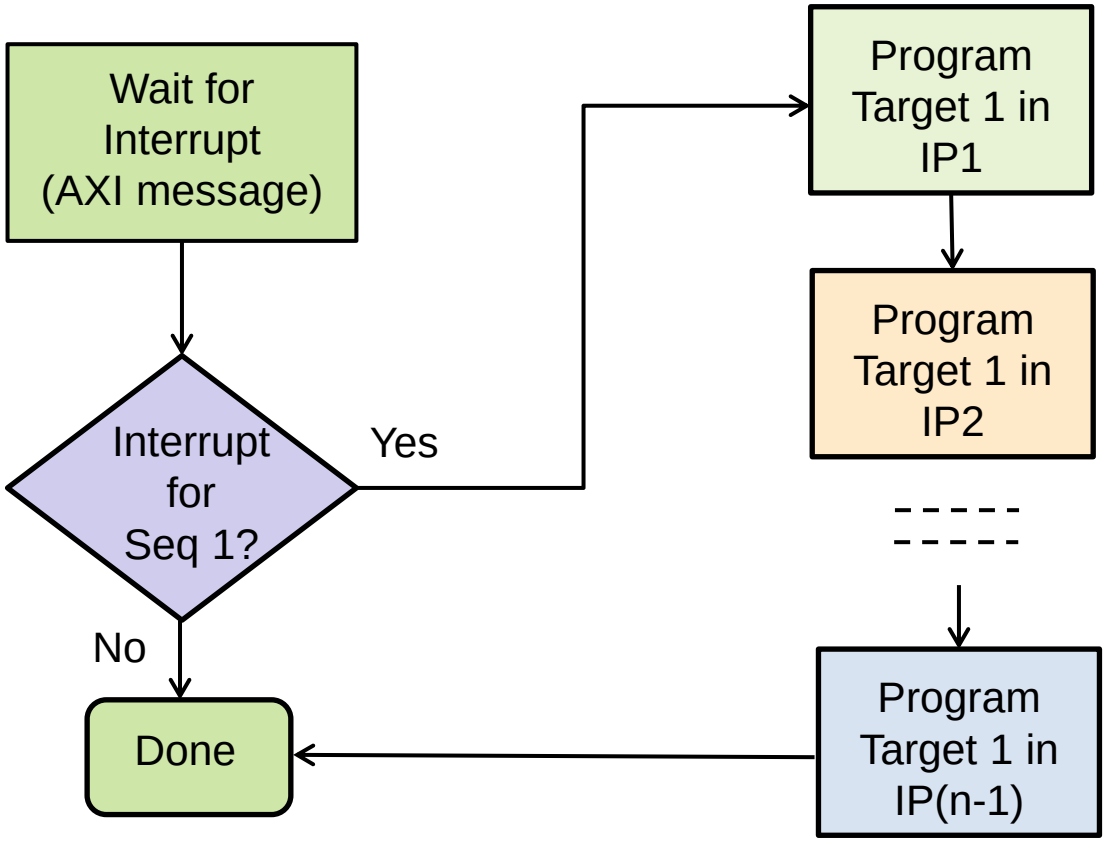
Overview

- Need for firmware scoreboard
  - Does the firmware execute a sequence correctly?
- Can we use standard UVM scoreboard?
  - UVM scoreboard is suitable for protocol checking
  - Knowledge of what firmware must execute is in the test, scoreboard must therefore be in the test
  - Sequence check in a multi-IP sub-system is complex with standard scoreboard
- What does firmware scoreboard involve?
  - Transaction collector using analysis ports
  - **Test-based enumerated scoreboard**

Scoreboard For Firmware

Test-based enumerated scoreboard

- Step by step procedure
  1. Read the firmware specification, identify key programming steps



Scoreboard For Firmware

Test-based enumerated scoreboard

2. Create an enumerated structure for the targets in the sequence

```
typedef enum {
        Target_1,
        …
        …
        Target_N,
} pm_event;
```

3. Create the scoreboard (SCBD) as a hash variable

   Key – targets in the sequence; Value – TRUE or FALSE

```
typedef  enum {FALSE=0, TRUE=1} bool;          //Key
bool pm_event_scbd [pm_event];                 //Actual SCBD
bool pm_event_scbd_exp [pm_event];             //Expected SCBD
pm_event my_pm_event;
```

Scoreboard For Firmware

Test-based enumerated scoreboard

4. Populate the expected SCBD hash at the start of the test

```
pm_event_scbd_exp = '{
                        Target_1 : TRUE,
                        Target_2 : FALSE,
                        …
                        Target_N : TRUE
                        }
```

5. Mark the actual SCBD during transaction collection

```
if (txn.addr == <interesting addr 1>) begin
        pm_event_scbd [Target_1] = TRUE;
end
```

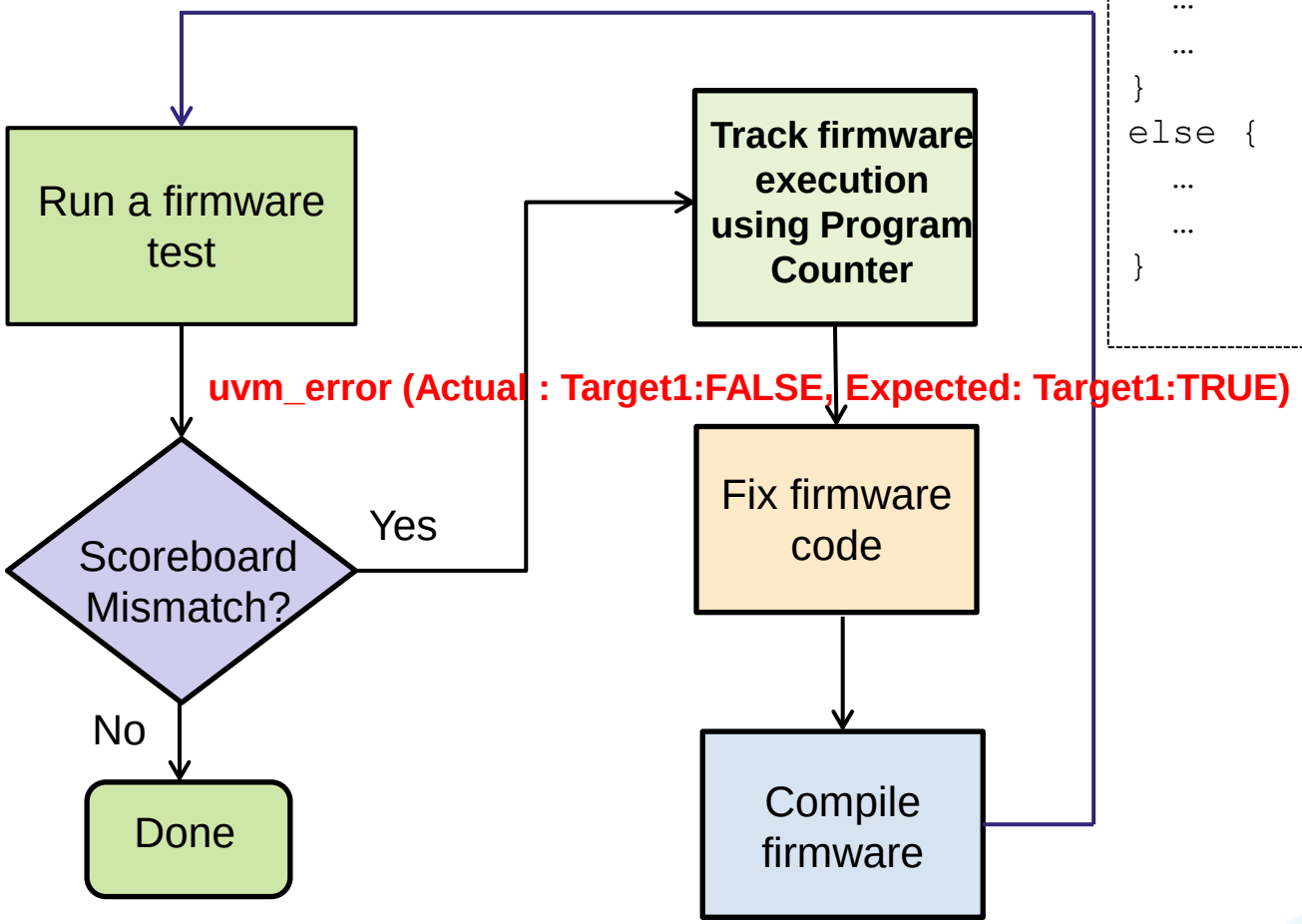
6. At the end of the test, compare expected and actual SCBD hashes

Scoreboard For Firmware

Evaluating failures

- Understanding a test cycle

```
if (cond1){
   prog(targ1);
   …
   …
}
else {
   …
   …
}
```



Conclusion

Conclusion

- Problem point
  - Simulate firmware C++ sequences
  - Verify firmware sequence for correctness
- What did it involve
  - Testbench updates
  - Test-based scoreboard
  - Regression cycle with firmware tests
- Outcome
  - Verify hardware interactions of firmware
  - Deliver clean code and help speed up to post-silicon bring up

Ashwini debugs the firmware bug with the help of test-based scoreboard!

AMD

snug
Synopsys Users Group

Thank You