Accessing DesignWare® Sensor and Control IP Subsystem Resources

In an OVM/UVM Testbench using RAL

Stefan Neumann
Intel Ireland Ltd.

June 23rd, 2015
SNUG UK

Agenda

Design Overview and Integration Challenge

Accessing Internal Subsystem Resources

Controlling Resources Independently

Results

Design Overview and Integration Challenge

Design Overview

DUT

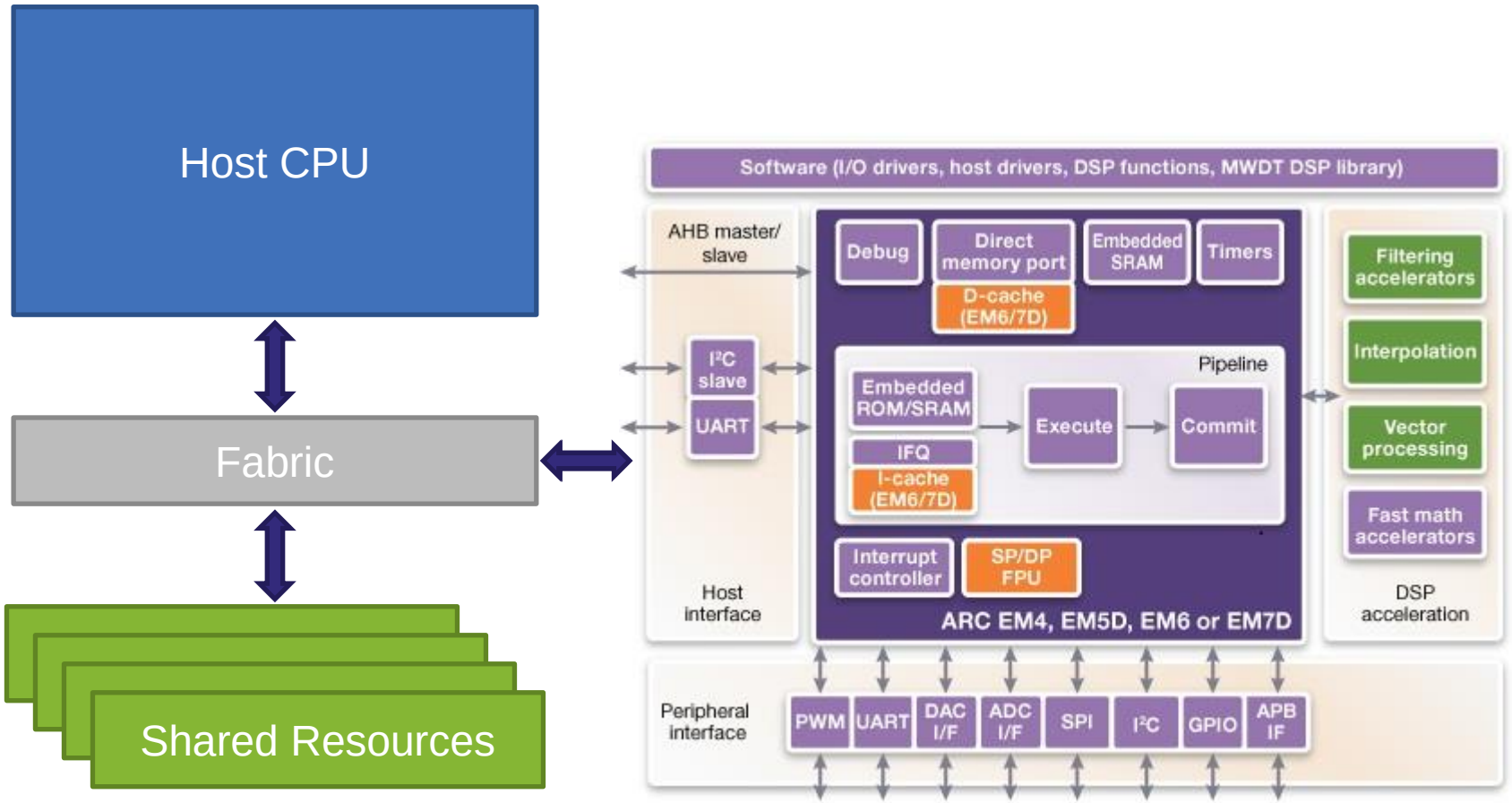


Image source: https://www.synopsys.com/dw/ipdir.php?ds=sensor_subsystem

Integration Challenge

- Sensor and Control IP Subsystem is an autonomous design
  - Runs its own firmware, controlling all closely coupled resources
  - Internals may only be accessible by the embedded ARC processor itself or via debug port

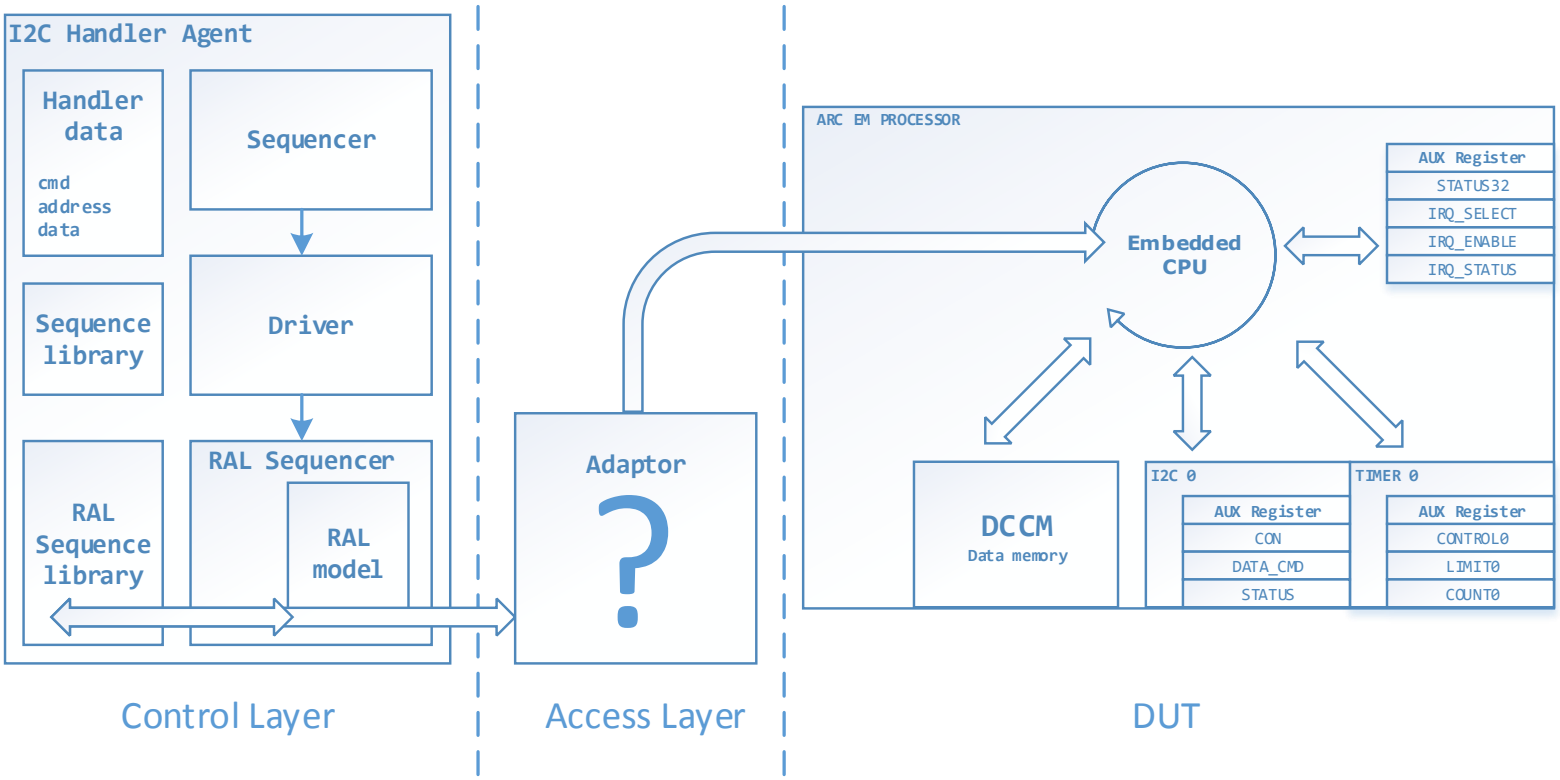
- Confidence tests do not cover complex SoC scenarios
  - Additional test content needs to be developed in Assembly/C/C++
  - Test content is static in nature and does not interact with the TB
  - SoC behavior almost requires full qualified FW code

Integration Challenge (continued)

- Provide a reusable verification environment to allow:
  - Configuration and control of the Sensor Subsystem in a dynamic environment using RAL
  - Horizontal reuse of components for different configurations
  - Vertical reuse and seamless integration into SoC environment

- How to synchronize RAL register model updates with the design without frontdoor accesses path?
  - Debug port available but too slow for simulations

Integration Challenge (continued)



- Our proposal:
  - Turn the embedded processor into an active TB component used as a frontdoor path

Accessing Internal Subsystem Resources

Access Layer Architecture Overview

Access Layer

- The Access Layer consists of:
  - Assembly program (Test Daemon) running on the processor to interpret and execute requests from the TB
  - OVM/UVM handler agent to control the Test Daemon state (Test Daemon Handler Agent)

- The handler agent utilizes RAL backdoor accesses on a subset of general purpose register (GPRs) to:
  - Store command and status information
    - Address and data for register and memory accesses
  - Call available Test Daemon tasks

Test Daemon

- The Test Daemon is a small Assembly program acting as an interface between the TB and the embedded processor
  - Exposes data path of the processor as a frontdoor access path
  - Highly optimized for performance

```
.align 32; inf_loop:                    //Infinite Loop
bbit0 STATUS_REG, 31, inf_loop
bi    [CMD_REG]                         //Branch to Task
.align 32; set_aux_reg_cmd:             //Write Aux Reg Task
sr    DATA_REG,   [ADDR_REG]
mov   STATUS_REG, 0                     //Task successful
b     inf_loop
.align 32; get_aux_reg_cmd:             //Read Aux Reg Task
lr    DATA_REG,   [ADDR_REG]
mov   STATUS_REG, 0                     //Task successful
b     inf_loop
```

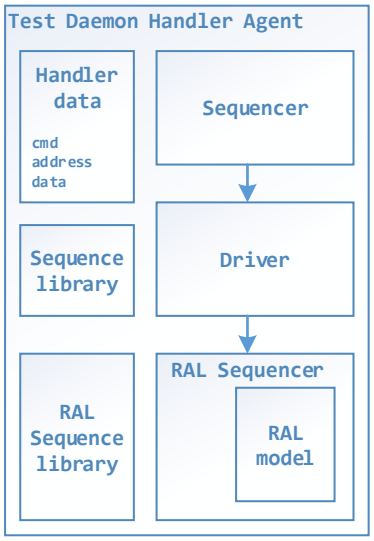
Test Daemon (continued)

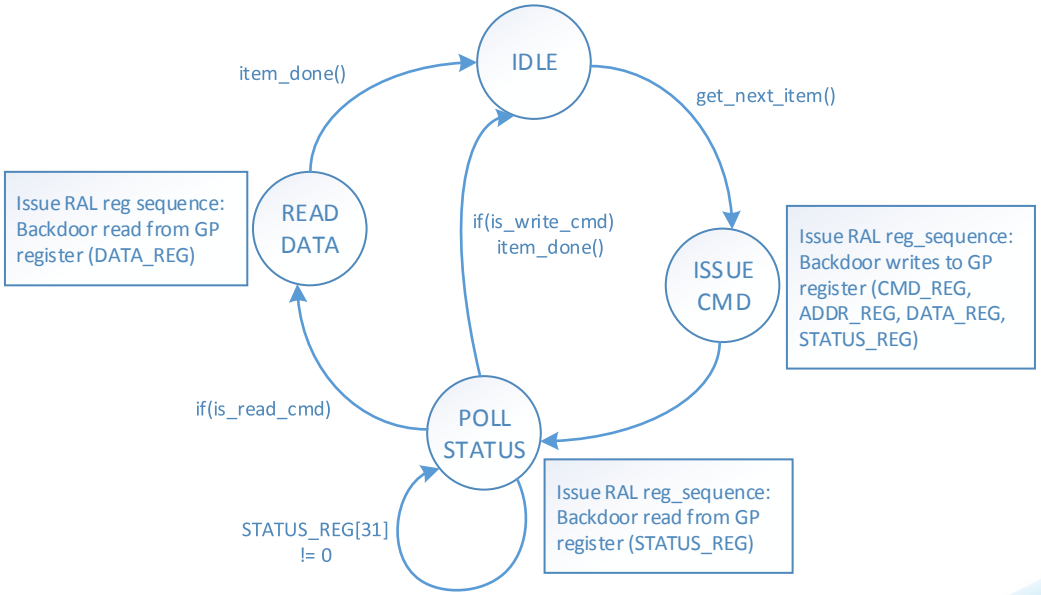
- Full test daemon provides tasks to:
  - Access auxiliary register and memory space
  - Configure timers
  - Configure interrupt (e.g. global enable/disable)
  - Transition between sleep, halt and power states

- Exceptions and Interrupt conditions are recognized by the Test Daemon but ultimately handled by the TB
  - Exceptions will be reported to the TB
  - Test Daemon will always stay responsive
  - TB will always stay in control of transaction flow

Test Daemon Handler Agent

- Handler agent maintains state of the Test Daemon
  - Driver controls state machine and issues RAL backdoor sequences on the virtual RAL sequencer
  - Local RAL model holds GPR backdoor locations
  - Sequence library exposes Test Daemon functionality and provides interface for the Control Layer





Controlling Resources Independently

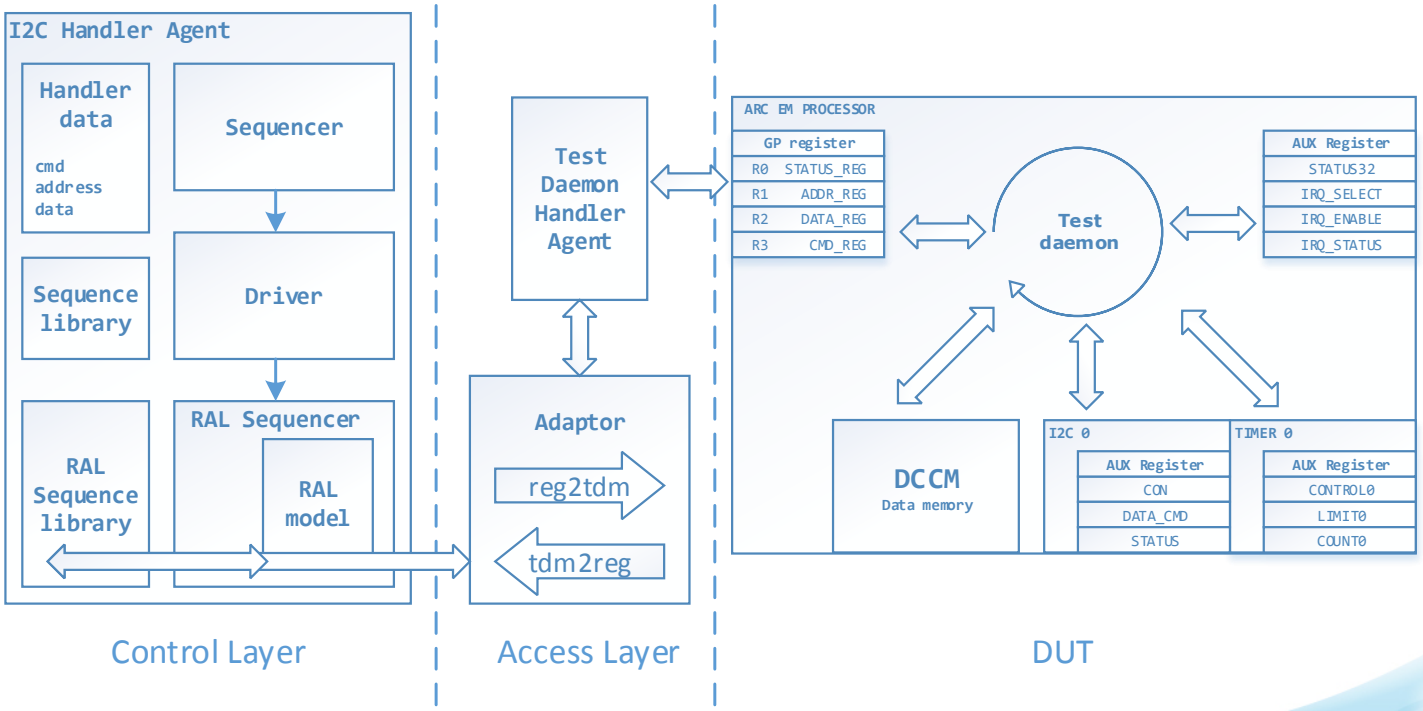
Control Layer Architecture Overview

Control Layer

- The Control Layer consists of:
  - Handler agent components for all available resources present in the Sensor Subsystem
    - Multiple instances for same resource type
  - Interrupt agent, monitoring the external interrupt interface and starting ISR sequences on other handler agents
- The handler agents utilize RAL frontdoor accesses on the interface provided by the Access Layer
- Testbench can control the Sensor Subsystem state using virtual sequences

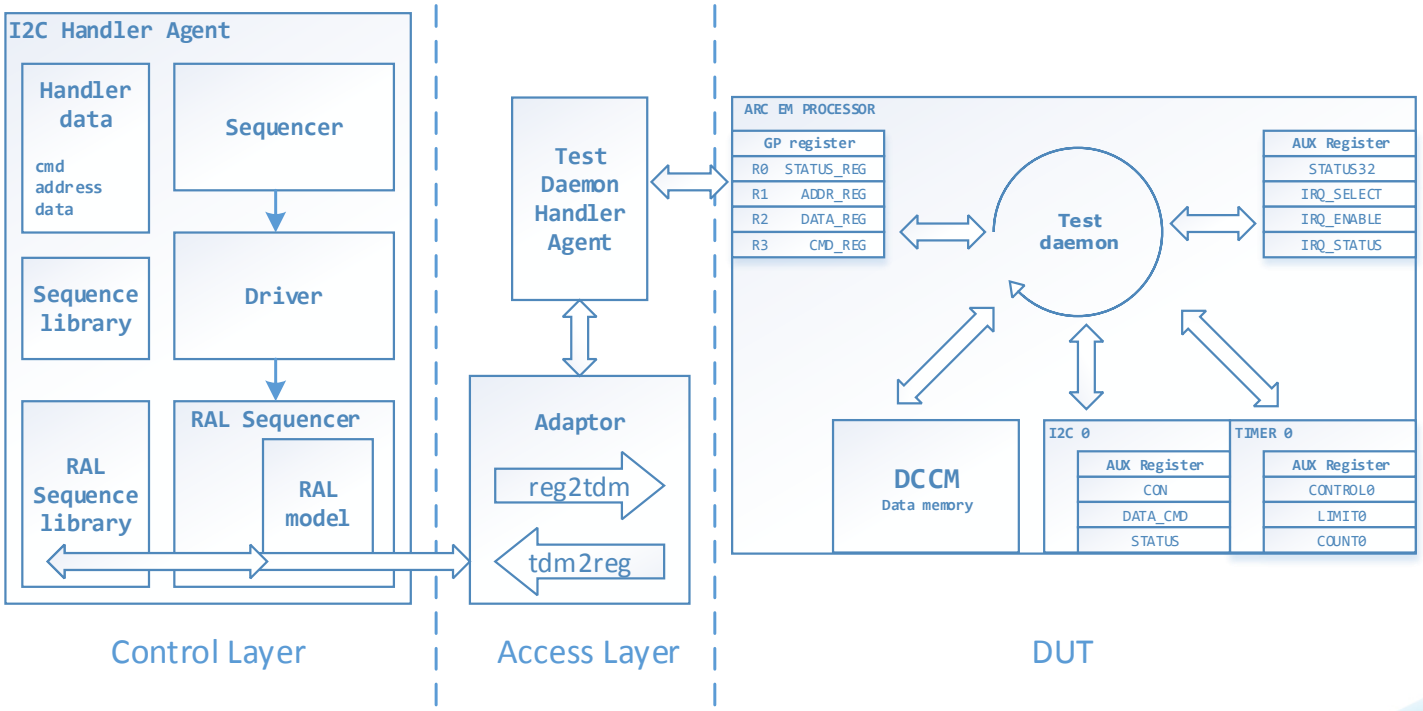
Handler Agent

- Handler agent maintains state of one particular resource
  - Driver controls state machine and issues RAL frontdoor sequences on the virtual RAL sequencer
  - Local RAL model holds register locations and address information
    - Different base address per instance



Handler Agent (continued)

- RAL frontdoor sequences get translated to Test Daemon tasks being executed
  - Adaptor class will issue Test Daemon Handler Agent sequences
  - Test Daemon will execute task to update register content



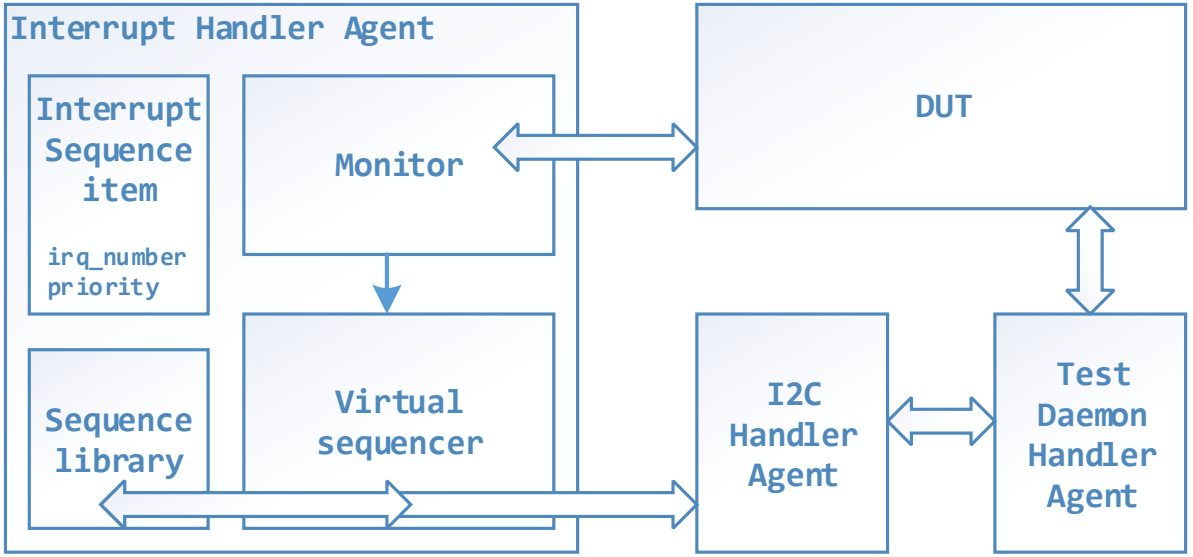
Interrupt Agent

- Interrupt agent handles interrupt conditions by starting ISR sequences on different handler agents
  - ISR sequences and sequencer are registered with the interrupt agent

```
function void env::connect_phase(uvm_phase phase)
    intr_handler_agent_cfg.set_intr_seq(
       I2C_0_ERR,                          //Interrupt src type
       "i2c_handler_agent_err_intr_seq", //Sequence type name
       i2c_handler_agent[0].sequencer);  //Pointer to sequencer
    intr_handler_agent_cfg.set_intr_seq(
       I2C_1_ERR,
       "i2c_handler_agent_err_intr_seq",
       i2c_handler_agent[1].sequencer);
```

Interrupt Agent (continued)

- Interrupt agent handles interrupt conditions by starting ISR sequences on different handler agents
  - Allows for different interrupt priorities and preemption using sequence priorities propagated to the Test Daemon Handler



Results

Results

- Fully controllable Subsystem using OVM/UVM RAL in a constrained random TB
  - No Assembly/C/C++ code dependency other than the Test Daemon
  - Full register abstraction for the handler agent components
    - Multiple instantiations for same resource type
  - Full reuse of RAL sequences for shared components on SoC
    - Either controlled by Host CPU or Sensor Subsystem processor

- Improved performance compared to debug port access
  - 6 to 10 clock cycles on average to execute test daemon task
  - No need to use debug port or SW debugger attached to the simulator

Thank You