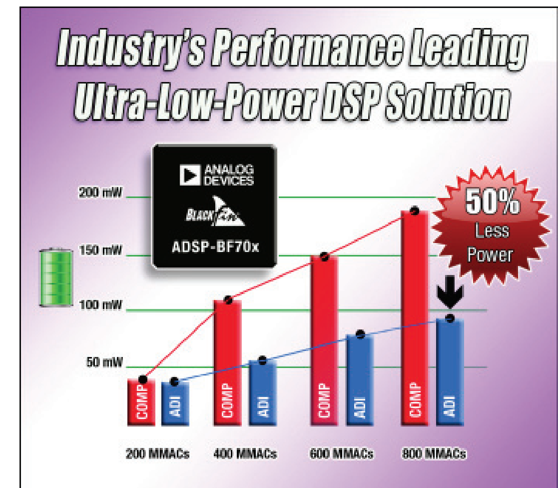
UVM, VMM and Native SV: Enabling Full Random Verification at System Level

Ashok Chandran, Analog Devices

Introduction

- Latest Product in Blackfin Processor Family – BF70x
  - Enhanced Core
  - Low Power System
  - Memory
  - Peripherals
  - Security



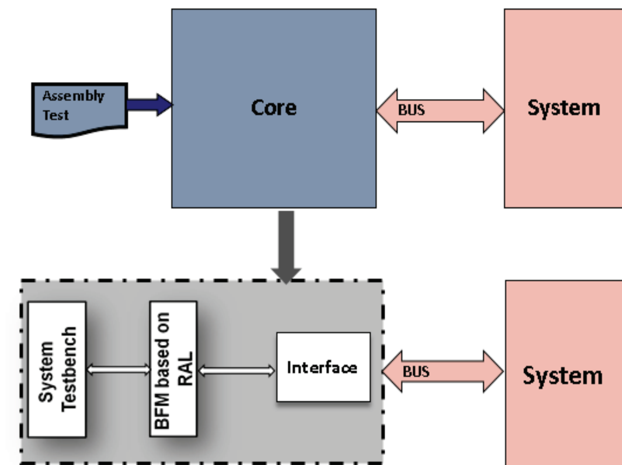
Overall DV Methodology

- Bottom Up Approach
  - MDV for blocks
  - Spec->Executable Vplan-> CRV >Coverage->QA
  - Significant set of VIP in VMM/UVM/SV
- Focus on System Level Verification
  - Constrained random verification at system level
  - Complemented by Formal for connectivity, glue logics

© Accellera Systems Initiative

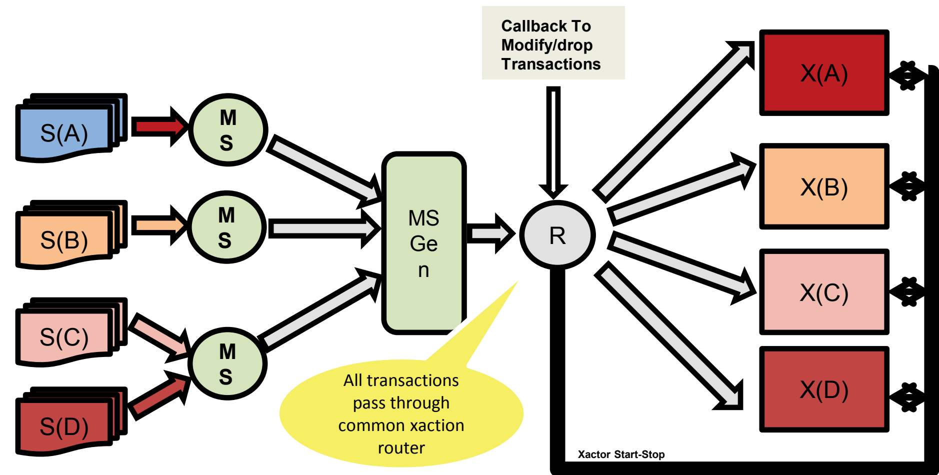
System Level Approach

- Core Independent Testbench
  - Why ? – We work with Blackfin, ARM, Sharc Cores
  - Uses BFM Driver for MMR and Memory access initiated by Core
  - Enables peripheral and system oriented verification
- Core System DV
  - Dynamic Backdoor Memory load with random code generation
  - Actual core executes random code (cache, data, execute)
- Arbitration Mechanism between BFM and actual core



System Level Approach

- What we had - VMM based SoC Level Testbench
  - Has worked well for past projects
- VMM Features Used
  - Register/memory Model
  - Memory Allocation
  - MS Scenarios



A,B,C and D represent different IO
S is a scenario of the IO
MS represents a multi-stream scenario
MS gen is a MS scenario generator
X is the xactor which handles the scenario
R is the xaction router and xactor manager

Adopting UVM - Challenges

- Industry trend - UVM is now preferred for block DV
- Challenges for us in moving to UVM:
  - Top level was VMM based
  - Wide variety of VIPs
    - Huge set of VMM based VIPs – Scenario Driven
    - SV and pure Verilog Testbenches (file based)
    - New UVM Testbenches – Sequence Driven
  - Goals
    - Randomize and execute across all VIPs independent of the Methodology
    - Manage shared system resources effectively between all components
    - Layered approach to ensure reusability

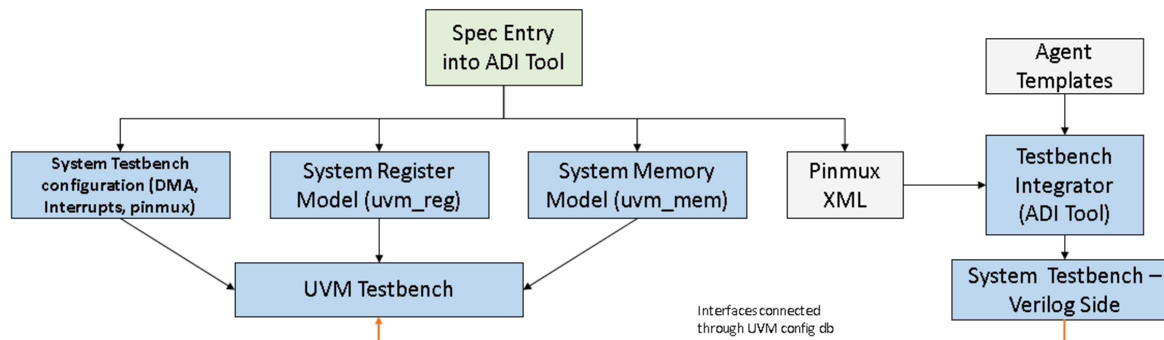
Unify the methodologies by a new layer on top for multi-peripheral verification

Key Requirements

- System Resource Management
  - Shared resources should be managed
  - Configuring system resources should be simplified
    - Standard DMA, Interrupts, Pinmux, Interconnect priorities
- Enabling VMM blocks inside UVM sequences
- Block to system reuse of UVM agents
- Multi-Peripheral Wrapper
  - Allow to run VMM and UVM block together

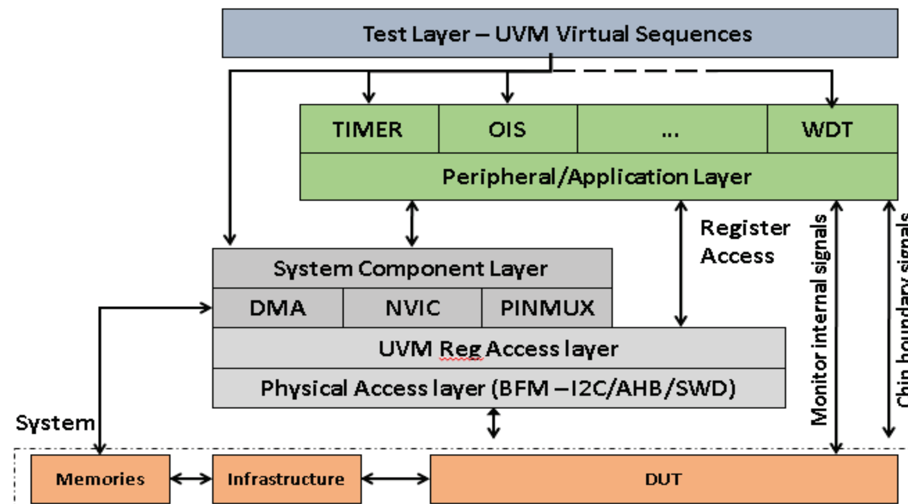
System Level UVM Platform

- Migrated Top Level to UVM
  - Takes time, but was an investment for future
- Testbench Integration Approach Improved
  - Components for UVM testbench – register model, mem, configuration are generated from spec directly
  - Verilog side is instantiated for multiple instances and stitched by script based on pinmux spec – UVM config db helps pass interfaces



System Level UVM Platform

- Layered Approach
  - Made migration to UVM easier
  - Xactors were switched in place with UVM components
  - New system transaction for UVM components



System Level UVM Platform

- Register Model Migration
  - Had been using reg model from VMM
  - Created the toplevel register and memory model in UVM
  - The access methods were identical
  - Scripts to change functions which differed

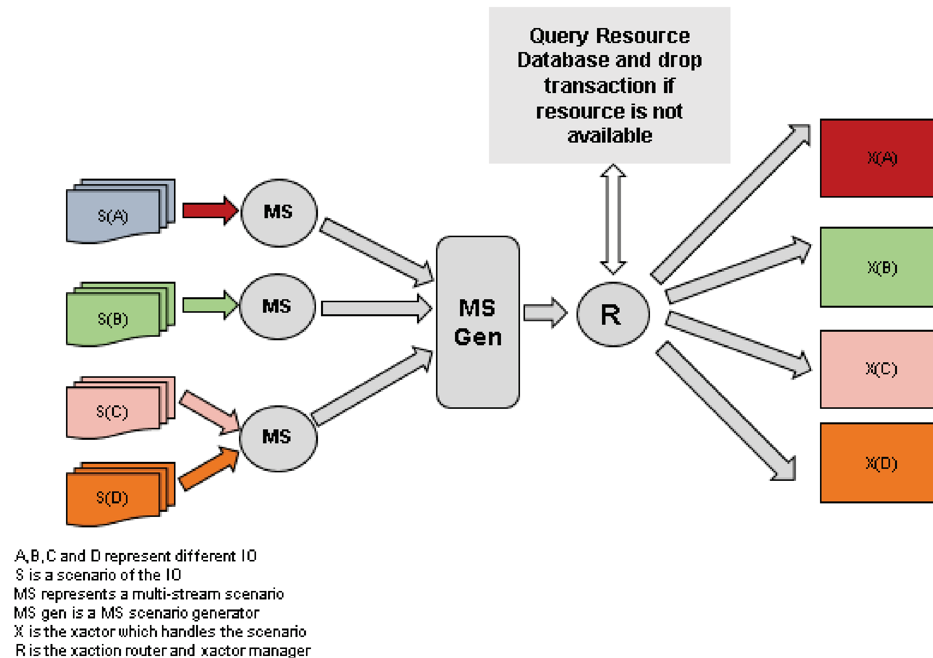
accellera SYSTEMS INITIATIVE

© Accellera Systems Initiative

2014 DESIGN AND VERIFICATION DVCON CONFERENCE AND EXHIBITION INDIA

Resource Allocation

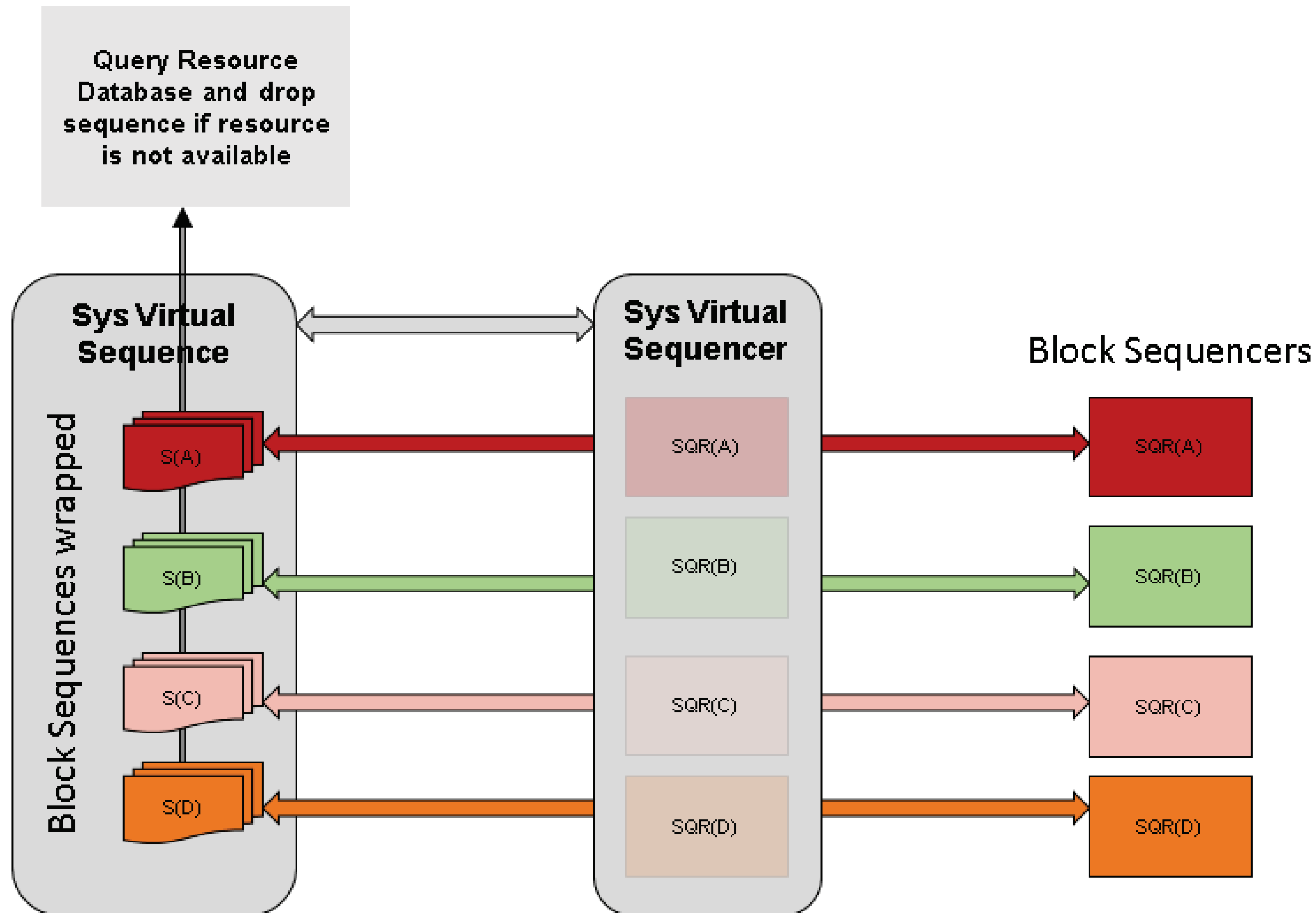
- Environment maintains a resource database
- Resource Management is at input side itself (Lossy generator)
- VMM transaction router queries the DB and drops if resource conflict is present



© Accellera Systems Initiative

Resource Allocation (contd ..)

- In UVM side, system sequence checks resource DB



Resource Allocation (contd ..)

- Ensuring Randomness in allocation
  - For Shared resources, need to ensure that randomly different sources should get the resource
  - Ensured by starting all scenarios and sequences at the same time.
  - Resource checking should be the first action in the system level virtual sequence for a uvm block.

accellera SYSTEMS INITIATIVE

2014 DESIGN AND VERIFICATION DVCON CONFERENCE AND EXHIBITION INDIA

Resource Management

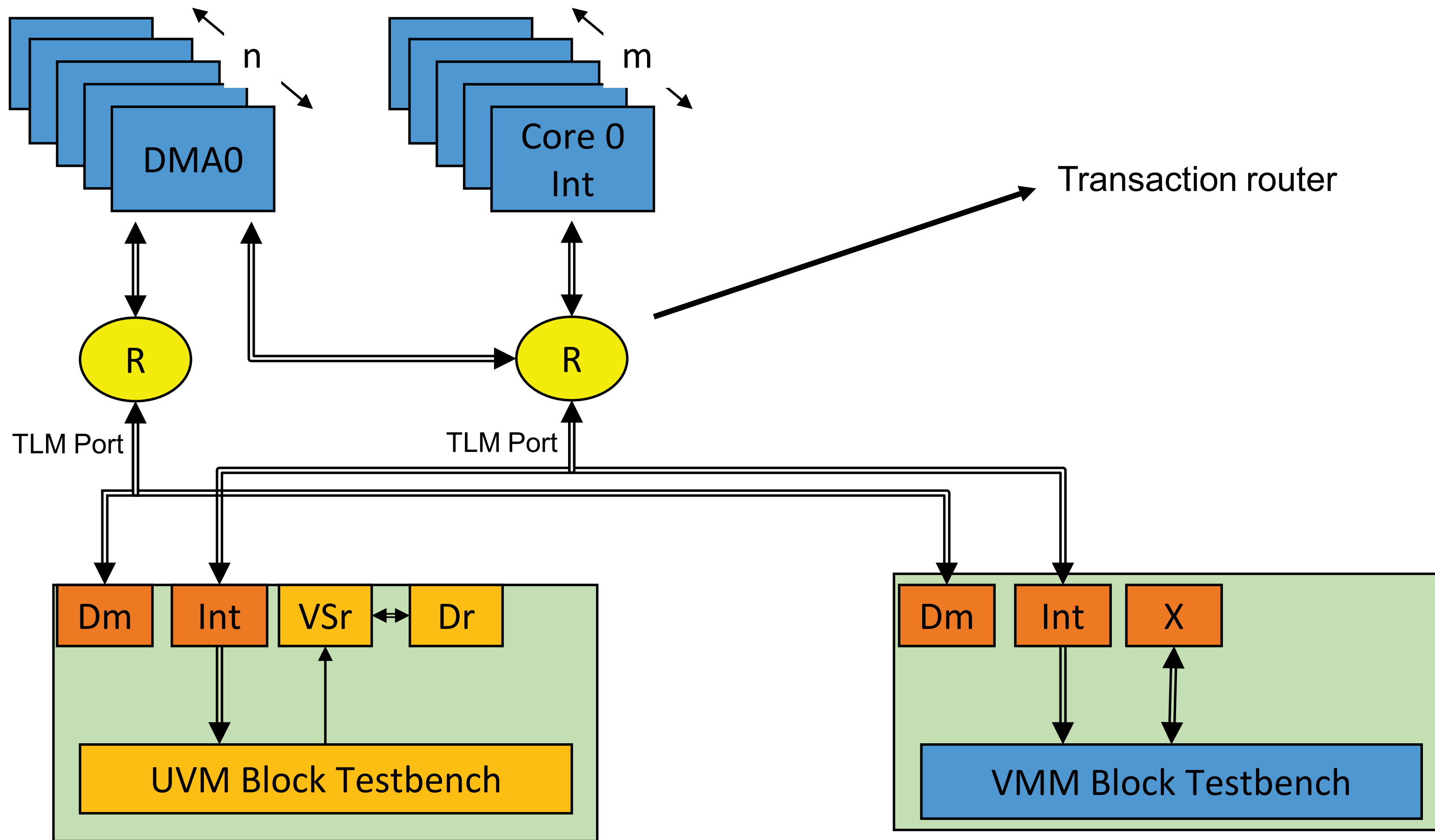
- System resources (which are now allocated to each peripheral) are managed by individual system components
  - DMA, Pinmux, Each interrupt Interface , Interconnect, Fault Management etc
  - Manages all resource requirements for the peripheral
- Block environments communicate to system resources via transactions and utility functions

accellera SYSTEMS INITIATIVE

© Accellera Systems Initiative

2014 DESIGN AND VERIFICATION DVCON CONFERENCE AND EXHIBITION INDIA

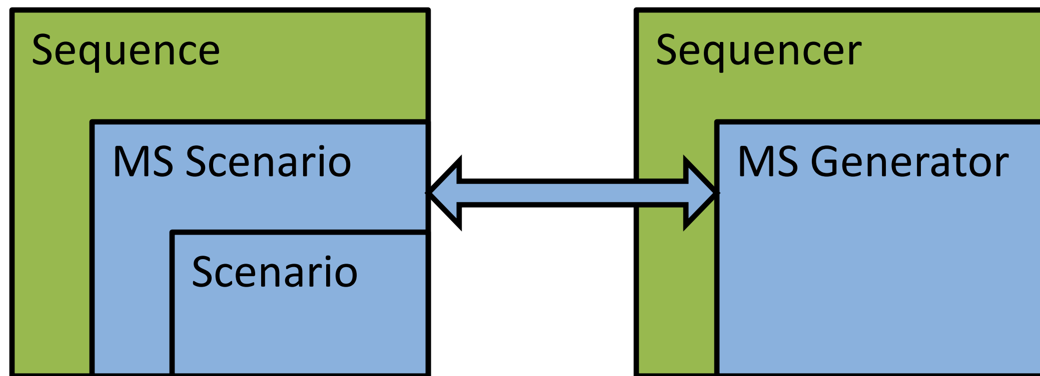
Resource Management (contd …)

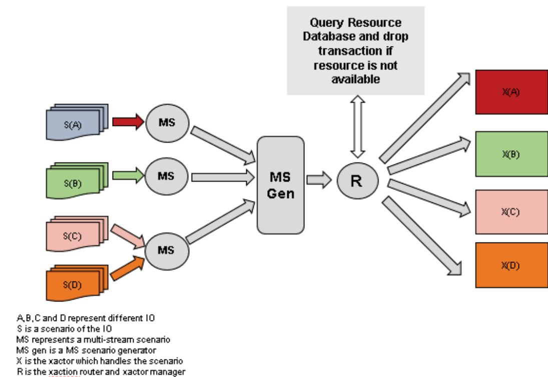


accellera SYSTEMS INITIATIVE
2014 DESIGN AND VERIFICATION DVCON CONFERENCE AND EXHIBITION INDIA

Running VMM block

- Running a VMM Scenario/MS Scenario
  - Macros wrap VMM Scenarios under UVM sequences
  - VMM scenario generators are instantiated inside the syslvl_vir_sequencer





© Accellera Systems Initiative

Running SV block

- Pure Verilog/SV blocks present for some IO blocks
  - File based – reading from file into Verilog arrays
  - Arrays mapped to memories via UVM register model
  - Read/write via backdoor of register model
- Need to create components to randomize

```
block AGENT_SP {
    bytes 4;
    memory SP_DATA=SP_TX_PRI_DATA (core.data) @none;
    memory SP_DATA=SP_TX_SEC_DATA (core.data_sec) @none;
    memory SP_DATA=SP_RX_PRI_DATA (core.u_rx.data) @none;
    memory SP_DATA=SP_RX_SEC_DATA (core.u_rx.data_sec) @none;
    memory SP_DATA=SP_AGENT_REG (core.sport_agent_reg) @none;
    register SPEN=SPENA_CORE (core.spA_en) @none;
    register SPEN=SPENB_CORE (core.spB_en) @none;
    register SPEN=SPENA_RX (core.u_rx.spA_en) @none;
    register SPEN=SPENB_RX (core.u_rx.spB_en) @none;
    register SPEN=SPENA_FIFO (fifo.spA_en) @none;
    register SPEN=SPENB_FIFO (fifo.spB_en) @none;
}
```

Verilog Array

Agent Register

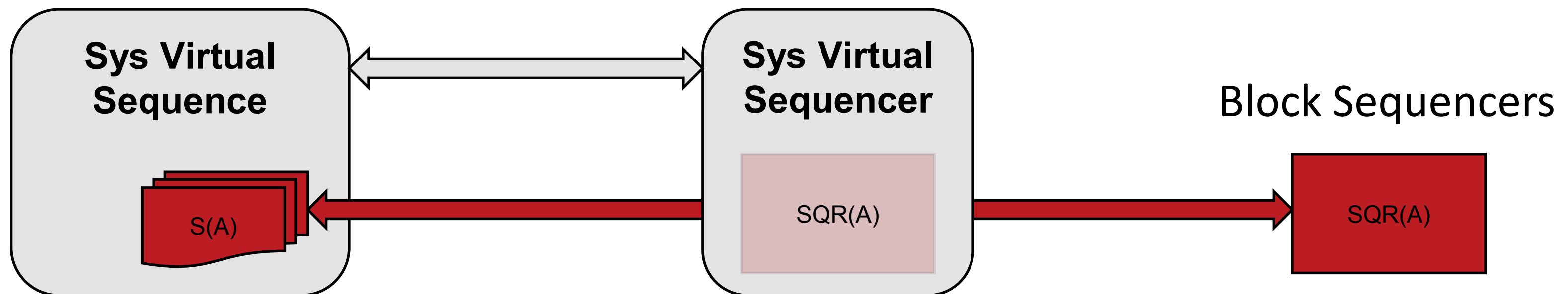
HDL Path

accellera SYSTEMS INITIATIVE

2014 DESIGN AND VERIFICATION DVCON CONFERENCE AND EXHIBITION INDIA

Running UVM blocks

- UVM Sequences can be run directly
- Need to create system virtual sequence
  - Performs system configuration
  - Performs resource allocation
  - Extends from common class to allow multi-peripheral seq



© Accellera Systems Initiative

Multi-Peripheral Sequence

- Enables VMM and UVM blocks to run together
- Macros create virtual sequences for uvm and vmm from common base class
- Sequences are registered inside multi_periph_test
- Randomly picks "N" sequences out of "M" registered items.

```
class syslvl_dap_mdma_test extends syslvl_multi_periph_base_test;
     `uvm_component_utils(syslvl_dap_mdma_test)


     function void build_phase(uvm_phase phase);
          super.build_phase(phase);
          `register_multi_sequence(syslvl_mdma0_sequence_vir_sequence,"MDMA0")
          `register_multi_sequence(dap_rtg_vir_sequence,"DAP0")
     endfunction:build_phase

endclass:syslvl_dap_mdma_test
```

accellera SYSTEMS INITIATIVE

© Accellera Systems Initiative

2014 DESIGN AND VERIFICATION DVCON CONFERENCE AND EXHIBITION INDIA

Results

- Integrated UVM and VMM into single environment
  - Capable of randomly picking UVM/VMM/SV blocks
  - Ensures random resource allocation
- Ease of integration of new peripheral environments
- Reused into other ADI projects
- Many bugs caught through random verification
  - First silicon sampling to customers

Questions