Leveraging Portable Stimulus across Domains and Disciplines

Pradeep Salla, Mentor Graphics India
Tom Anderson, Breker Verification Systems
Karthick Gururaj, Vayavya Labs Pvt. Ltd.
Background

• The industry has recognized the need for portable verification test cases (stimulus+results+coverage)
• “Portable Stimulus” Proposed Working Group started within Accellera in May 2014
  – List of 100+ technical requirements developed
• Portable Stimulus Working Group began March 2015
  – Defined “use cases” that a standard must satisfy
• Call for contributions closes on September 16, 2015
• First version of standard expected in mid-2016
PART 1: ABSTRACTION LEVEL, PORTABILITY AND REUSE

PRADEEP SALLA, Mentor Graphics India
Test Creation Challenges
Block to System

• Large state spaces
• Require scenarios, not just transactions

• Multiple verification platforms
  – Simulation
  – Emulation
  – Prototype

• Multiple languages
  – SystemC
  – SystemVerilog
  – Embedded software

• Must have automation
• Must have reuse
Test Creation Requirements

• Need to generate lots of test sequences

• Need to generate scenarios, not just lots of transactions
  – Test more-complex functionality
  – Sequences of operations
  – Multi-stream scenarios

• Need visibility and control over these scenarios
  – Ensure we ‘cover’ the scenario space
  – Select an appropriate subset of the scenario space

• Scenarios are difficult to create – even at block level

• Need to have the option to migrate these scenarios forward
  – SoC level
  – System level
Stimulus Creation Techniques

- Directed tests
  - Focused, user-created
  - Captures data and control flow
  - Low-productivity

- Constrained-random tests
  - Open loop, automation driven
  - Captures data
  - High-productivity

- Graph-based goal-driven tests
  - Flexible focus, automation driven
  - Captures data and control flow
  - High-productivity, goal driven
Portable Stimulus Specification

Goals

- Single input specification
  - Stimulus
  - Goals
  - Expected results

- Automated test creation

- Reusable
  - Across environments
  - Across disciplines

- Tools generate specifics
Scenario Example

• Send PCIe TLP

• Inject a link event (Optional)

• Send another PCIe TLP

• Cover all combinations of: TLP types (30), link events
Directed-Random Test
UVM Sequence implementation

- Difficult to write
  - Depends on test writer skill
- Difficult to review
  - No high-level view – just code
- Difficult to control
  - Hard to constrain from outside
  - Hard to tell what occurs
- Difficult to migrate/reuse
  - Specific implementation
  - Specific language
  - Specific execution env

```cpp
typedef enum { EV1, EV2, EV3, EV_NONE } event_t;

class pcie_scenario_seq;

    task body();
    pcie_sequence_item tlp1 = pcie_sequence_item::type_id::create();
    pcie_sequence_item tlp2 = pcie_sequence_item::type_id::create();
    event_t ev;

    forever begin
        start_item(tlp1); tlp1.randomize(); finish_item(tlp1);
        std::randomize(ev);

        case (ev)
            EV1: run_ev1_seq();
            EV2: run_ev2_seq();
            EV3: run_ev3_seq();
            EV_NONE: // Do nothing
        endcase

        start_item(tlp2); tlp2.randomize(); finish_item(tlp2);
    endtask
endclass
```
Graph-Based Portable Stimulus
PCIe Scenario Graph

• Easy to write
  – High-level specification
  – Efficient

• Easy to review
  – Hierarchical
  – Expand to see elements of interest

• Easy to control
  – Specialize from outside
  – Goal-driven generation

• Easy to migrate/reuse
  – Declarative language is
  – Specific language
  – Specific execution env

© Accellera Systems Initiative
Test Specification Fundamentals

• What is legal
  – Universe of what could happen
  – Captures both data and scenario
  – ‘unexpected’ cases

• What to target
  – Cases of specific interest
  – What to verify today, during this test
Graph-Based Stimulus Description
Captures data and data relationships

• Scalar types
  – Signed and unsigned integer types
  – Enumerated types

• Composite data structures
  – ‘struct’, supports type extension

• Aggregate data types
  – Single and multi-dimensional fixed-size arrays

• Variables can be input or output
  – Output variables (default) send values to the environment
  – Input variables bring values in from the environment

• Constraints
  – Algebraic expressions, inside, if/else, foreach, etc

```
struct my_struct1 {
    meta_action A[unsigned 3:0];
    meta_action B[unsigned 3:0];
}

struct my_struct2 extends my_struct1 {
    meta_action C[unsigned 3:0];

    constraint c {
        C < A;
    }
}
```
Graph-Based Stimulus Description
Captures test scenario control flow

• Captures process of stimulus generation
  – Sequences of operations
  – Choices
  – Loops

• Branch-specific constraints
  – Conditional execution
  – Partitions scenario structurally

```c
constraint a_eq_b dynamic {
    inst3.A == inst3.B
}

my_graph = init repeat {
    inst1
    if {inst1.A == 5} (inst2) |
    if {inst1.A >= 5} (inst3) |
    if {inst1.A == 6} (a_eq_b inst3)
    infact_checkcov
};
```
Bringing it all Together
PCIe TLP Rules/Graph

• Import TLP specification

• Define scenario with
  – Two TLP items
  – Inject event between

• Run scenario in a loop

```c
rule_graph pcie_scenario_seq {
    import "pcie_sequence_item.rseg";
    action init;
    interface do_item(pcie_sequence_item);

    set event_t[enum EV1, EV2, EV3, EV_NONE];

    struct pcie_scenario {
        pcie_sequence_item tlp1, tlp2;
        meta_action ev[event_t];

        action run_ev1_seq, run_ev2_seq, run_ev3_seq;

        symbol inject_ev = ev
            if {ev == EV1} run_ev1_seq | if {ev == EV2} run_ev2_seq |
            if {ev == EV3} run_ev3_seq | if {ev == EV_NONE} eta;

        pcie_scenario =
            do_item(tlp1)
            inject_ev
            do_item(tlp2);
    }

    pcie_scenario s1;
    pcie_scenario_seq = init repeat {
        s1
    };
}
```
Test Selection and Prioritization
Coverage Strategy

- Coverage strategy expresses test scenario goals
  - Key stimulus values, combinations, sequences

- Flexible
  - Prioritize certain goals
  - Combine random/systematic generation

- Reactive
  - Adapts to changes in verification environment state

- Efficient
  - Automatically suppresses redundant stimulus
  - Reaches goals 10-100x faster than pure-random generation
Test Selection and Prioritization

Target value specification

• Variable domains can be divided using ‘bins’
  – Split a value range into N bins
  – Split a value range into bins of size N

• Coverage constraints select target space with expressions
  – Only active when targeting the coverage goal
  – Prioritizes specific combinations
  – Full legal space reachable otherwise

• Example: A x B
  – Full legal space is 256 (16 * 16)
  – Constraint A < B selects 120 combinations
Bringing it all Together
PCIe Scenario Coverage Goals

• Total scenario space: $6.75 \times 10^{167}$
  – TLP1 fields X events X TLP2 fields

• Target combination of
  – TLP1 type
  – Event type
  – TLP2 type

• Total targeted scenarios: 3600

• Efficient scenario generation
  – No redundancy, predictable completion
Graphs and Automation
Distributed Verification

• **Dynamically partitions goals across simulation runs**
  – Enables distribution in parallel and/or series
  – Each simulation uniquely contributes to coverage

• **Shortens regression runs**
  – Maximize compute resource utilization
  – Linear speedup with more machines

■ **Repeatable for debug**

■ **Compatible with compute-farm management software**
Graphs and Reuse
Leverage Existing SystemVerilog Classes

- Reads
  - SV classes
    - Fields, constraints
    - Covergroups

- Creates
  - Graph Rules
  - Coverage strategy

- Leverages existing SV
- Raises abstraction level
  - Import transactions
  - Build complex scenarios

© Accellera Systems Initiative
Graphs and Reuse
Compose Hierarchical Scenarios

- Easily compose larger scenarios
- Over-constrain to shape
- Two back-to-back scenarios
  - Customized TLP types
  - Customized events
- 28,800 scenarios
Graph-Based Portable Stimulus Specification
Raise the Abstraction Level

• Declarative scenario specification
  – Raises abstraction level
  – Efficient scenario capture

• Easy to review
  – Hierarchical description
  – Communicate across disciplines

• Enables reuse
  – Reuse existing SystemVerilog description
  – Reuse graph descriptions

• Enables automation
  – Efficient scenario generation
PSWG Breaking News

• On September 8, Cadence, Mentor, and Breker issued a joint press release
  – Three companies providing joint contribution to PSWG
  – Contributing proven technology and leveraging expertise

• Why this combined effort?
  – Avoids a long process to evaluate multiple contributions
  – Accelerates development of an Accellera standard
  – Accelerates adoption of portable stimulus in the industry

• What’s next?
  – Joint contribution to PSWG by September 16 deadline
  – BoF breakfast tomorrow 0800-0845 – share your thoughts!
PART 2: GRAPH-BASED SCENARIO MODELS
PSWG Vision

Proposed Portable Stimulus Diagram

Scope (Integration)
- Middleware (Graphics, Audio, etc..)
- OS & Drivers
- Bare Metal SW
- System on Chip (HW + SW)
- Sub-System
- IP

User
- Architect
- HW Developer
- Analog Developer
- SW Developer
- Verification Engineer
- SW Test Engineer
- Post-silicon Validation Engineer

Abstract Portable Stimulus Model
- Syntax/Concepts/Semantics
- Use Case Verification
- Visualization
- Runtime Portable Semantics

Tools (Secret Sauce)

Verification Environment
- UML/SysML
- SystemC
- HVL/UVM (SV, e)
- C/C++
- AMS

Platform
- Virtual Platform
- Simulation
- Emulation
- FPGA Prototype
- Silicon Board

APIs

© Accellera Systems Initiative 2015
Breker’s Vision

Horizontal (Platform) Portability

Simulation

Emulation

Prototyping

Silicon

Vertical (Integration) Portability

Transaction

Software Driven

SoC

Subsystem

IP
UVM ≠ Portable Stimulus

• UVM VIP components
  – Can be reused from project to project in same scope
  – Hard to reuse from IP to chip
  – Can’t reuse in hardware
• UVM is “pushing on a rope”
  – Hard to exercise deep state purely from chip inputs
• SoC verification limited
  – No link to code running in the embedded processors
  – No support for software-driven verification
Graphs = Test Case Composition

“Generate sequence $B = B_1 B_2 B_3$”

“When $B$ occurs, check that $S$ occurs”

“When $U$ occurs, check that $S$ occurs”

“To make $S$ happen, generate sequence $B_1 B_2$; to make $B_1$ happen, generate $U_1 U_2 U_3$; to make $B_2$ happen, generate $U_3$”

“To make $S$ happen, generate sequence $B = B_1 B_2 B_3$”

“When $B$ occurs, check that $S$ occurs”
Graphs Enable Portable Stimulus

• Graph-based scenario model
  – Abstract view of the design and the verification space
  – “Begin with the end in mind”
  – Built-in coverage metrics
  – Automatic coverage closure (nodes, arcs, etc.)

• Vertical portability
  – Combine IP graphs for full chip: test case composition
  – 100% scenario model reuse
  – Same approach works for chaining IP blocks in SoCs
  – Automatic test generation for transactions and embedded C/C++

• Horizontal portability
  – Automatic test generation from simulation to silicon
Example: Digital Camera SoC
Digital Camera Use Cases

Display Controller ➔ SD Card Controller Write ➔ Camera ➔ CCD

Display ➔ SD Card Controller Read ➔ SD Card

SD Card Controller Write ➔ SD Card Controller Read

Photo Processor Decode ➔ SD Card Controller Read

Photo Processor Encode ➔ Camera
Multi-Processor Scheduling
Aggressive Memory Scheduling

Have Verified Smartphones, GPUs, Servers, and More!
Example: Networking SoC

Constrain packet type, errors, length, etc.

Byte array sent to driver

Design

Packet Generator

VIP Driver

Constraints

uvm_tlm_generic_payload

UVM Testbench
Packet Generator Graph

IP address, port, TCP sequence, etc.

Graph-based scenario models scalable to an arbitrary number of header encapsulations
Automatic Coverage Closure

Now Running in Simulation on Chip with 1.3 Million Graph Nodes!
Example: Many-Core SoC
Cache Coherency Graphs

State Transition Table

<table>
<thead>
<tr>
<th>Current State</th>
<th>Cmd</th>
<th>Next State</th>
<th>Outputs / Expects</th>
</tr>
</thead>
<tbody>
<tr>
<td>C0 C1 L2 CCU</td>
<td>LD</td>
<td>C1</td>
<td>fetch</td>
</tr>
<tr>
<td>I I I I</td>
<td>C0</td>
<td>ST</td>
<td>snoop C1</td>
</tr>
<tr>
<td>M S S S</td>
<td>C1</td>
<td>LDEX</td>
<td>snoop C0</td>
</tr>
<tr>
<td>M S S S</td>
<td>VIP</td>
<td>ST EX</td>
<td>invalidate C1</td>
</tr>
</tbody>
</table>

State Machine

Graph

© Accellera Systems Initiative
Scheduling Concurrent Scenarios

Graph

\[ N \] Transition Scenarios

\[ N \] Graph Traversals

Concurrent Scenario Test Case

Schedule Memory
Interleave and Pack
Resolve Dependencies

Now Running in Silicon on Design with 96 ARM Cores!
Cache Coherency Test Case

Directed Coherency Test…

… vs. Generated Test Case
Breker Portable Stimulus Solution

Horizontal Reuse

Simulation

Emulation

Prototyping

Silicon

Vertical Reuse

SoC

Transaction

Subsystem

IP

UVM Testbench

SoC RTL

CPU

Memory

Photo Processor

Camera

Display Controller

SD Card Controller

VIP

VIP

VIP

UVM Testbench

SoC RTL

VIP

Memory

VIP

Photo Processor

Camera

Display Controller

SD Card Controller

VIP

VIP

VIP

Trek Family

SoC Scenario Model

Cam

Sys

PP

DC

SD

TrekApps

IP Scenario Model

© Accellera Systems Initiative
PSWG Breaking News

• On September 8, Cadence, Mentor, and Breker issued a joint press release
  – Three companies providing joint contribution to PSWG
  – Contributing proven technology and leveraging expertise

• Why this combined effort?
  – Avoids a long process to evaluate multiple contributions
  – Accelerates development of an Accellera standard
  – Accelerates adoption of portable stimulus in the industry

• What’s next?
  – Joint contribution to PSWG by September 16 deadline
  – BoF breakfast tomorrow 0800-0845 – share your thoughts!
KARTHICK GURURAJ, Vayavya Labs Pvt. Ltd.

PART 3: HW/SW INTERFACE AND PS
The Story so far...

• Importance of Portability of test-cases
  – To different environments
  – And different platforms

• Graph based scenario models to capture test intent

Is it all there to it?
A Motivating Example

Test bench (in SV)

DMAC (DUT, RTL in SV)

Simulation

Test cases (in C, executing on host PC)

DMAC (DUT, synthesized on an Emulator)

Accelerated simulation

Test cases (in C, executing on host PC)

DMAC (DUT, synthesized on FPGA)

FPGA, w/ external PCIe master

Test cases (in C, executing on embedded core)

DMAC (DUT, synthesized on FPGA)

FPGA, w/ embedded core
Need for HW-SW Interface in PS

Hardware-software Interface spec is important for “real portability” across environments
Different Environments in PS

Different manifestations of the same driver functionality!
HW/SW interface spec – current practice

• Described in module reference manual
• **Informal in nature** – prone to misinterpretations and ambiguity
  – At best, only programmable registers captured formally in XLS/XML/custom format
• Correctness of the specification is a function of review thoroughness

• Not surprisingly, even **publically available versions of datasheets** have few bugs...
27.2.3.3 I2S Basic Transmit/Receive Data Flow

After the module-initialization (reset and clock programming), program the corresponding bits of the following registers, preferably in the same sequence:

1. **I2S_CTRL**: Set FIFOs to Tx/Rx-mode as required: set I2S_CTRL[27]=0/1 for FIFO1 to be in Tx/Rx-mode and I2S_CTRL[30]=1/0 for FIFO2 to be in Tx/Rx-mode.
2. **I2S_FIFO_SCR**: Clear the FIFOs before every new transaction and program the attention-levels.
3. **I2S_TIMING**: Program to match the required sampling-rate based on the clock-source chosen on I2S_div clock.

### I2S Control Register

**Offset:** 000h  |  **Read/Write:** R/W  |  **Reset:** 0b00000000xxxxxxxx0000x0000000

<table>
<thead>
<tr>
<th>Bit</th>
<th>Reset</th>
<th>Description</th>
</tr>
</thead>
</table>
| 30  | 0x0   | **TX2_ENABLE**: Configure FIFO-2 as Tx-FIFO  
0 = DISABLE  
1 = ENABLE |
| 27  | 0x0   | **RX2_ENABLE**: Configure FIFO-1 as Rx-FIFO  
0 = DISABLE  
1 = ENABLE |
HW/SW interface spec – current

Table 16. Register Definitions, Page 1 through Page 6 (continued)

<table>
<thead>
<tr>
<th>Page (dec)</th>
<th>Addr (dec)</th>
<th>Addr (hex)</th>
<th>Default (binary)</th>
<th>Bits</th>
<th>Register Title</th>
<th>Register Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0000 0000</td>
<td>[7:3]</td>
<td>Calibration Mode</td>
<td>Calibration Mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>[2]</td>
<td></td>
<td>DDAC Disable</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0 AFE output includes DDAC (digital DAC offset) correction</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1 AFE output does not include DDAC correction</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>[1:0]</td>
<td></td>
<td>Calibration Mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>00 Black Calibration Only</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>01 White and Black Calibration - Full/Binary White Cal</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10 White and Black Calibration - Fine Tuning</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>11 Reserved</td>
</tr>
</tbody>
</table>

Ref: http://e2e.ti.com/support/other_analog/imaging_afes/f/239/t/189361.aspx

Texas Instruments, LM98725 TRM
Not “tomorrow’s” problem anymore

- **Ethernet ➔ Gigabit Ethernet**
  - Multiple process queues
  - TCP segment offloading
  - Checksum calculation in hardware
  - Receive-side scaling

- **USB2.0 ➔ USB 3.0**
  - Event queues instead of interrupts
  - Complex state-machine in programming the hardware
  - (More) Power-saving states
  - And, maintain USB2.0 legacy interface
“All problems in computer science can be solved by another level of indirection.”

David Wheeler
### HW/SW interface spec elements

- Registers
  - Address offset
  - Access type
  - Reset values
  - Bit fields
  - Repeated registers
  - ...
HW/SW interface spec elements

- Registers
- FIFOs
  - For data buffering
  - R/W access via registers
HW/SW interface spec elements

- Registers
- FIFOs
- Register access
  - Direct memory mapped
  - I2C
  - Other IO

Read/Write methods

Register Access

Registers

FIFOs

HAL
HW/SW interface spec elements

- Registers
- FIFOs
- Register access
- Interrupt management
  - Identification
  - Enable, disable, clear
  - Action to be taken
HW/SW interface spec elements

- Registers
- FIFOs
- Register access
- Interrupt management
- Sequences
  - Program the device for performing a specific task (e.g.: transmit)
  - Involves register accesses and data transformation
  - Exact set of sequences is device class dependent
HW/SW interface spec elements

- Registers
- FIFOs
- Register access
- Interrupt management
- Sequences
- Virtual registers
  - Used normally for scatter-gather DMA
  - Structures in memory that the peripheral understands
  - Very similar attributes as “normal” registers
HW/SW interface spec elements

- Registers
- FIFOs
- Register access
- Interrupt management
- Sequences
- Virtual registers
- Descriptor management
  - Virtual registers can be “instantiated” multiple times
HW/SW interface spec elements

- Registers
- FIFOs
- Register access
- Interrupt management
- Sequences
  - Involves register/virtual register accesses and data transformation
- Virtual registers
- Descriptor management

DPS – Device Programming Sequence
Formal capture of HW/SW interface
Formal capture of HW/SW interface

Sequences

Interrupt management

SystemRDL 1.0/2.0

Descriptor Management

Register Access

SystemRDL 1.0/2.0

Registers

FIFOs

Virtual Registers

SystemRDL 1.0/2.0
DPS Example

MEMORY_SPEC {
    FIFO TXQ[0:1] {
        TYPE = CHANNEL;
        TRANSFER_TYPE = RECEIVE;
    }
    FIFO RXQ[0:1] {
        TYPE = CHANNEL;
        TRANSFER_TYPE = TRANSMIT;
    }
}

VIRTUAL REGISTERS {
    XMIT {
        tdes0[64] {
            FIELD pBuffer<0:63>;
        }
        tdes1[64] {
            FIELD buf_length<0:15>;
            FIELD owner<16:16>;
            FIELD SOP<17:17>;
            FIELD EOP<18:18>;
        }
    }
}

INTERRUPT_SPEC {
    ETH_CTRL_INTR {
        RX_INTR {
            STATUS = ICR.RXT0(0x1);
            ENABLE_FIELD = IMS.RXT0(0x1);
            CLEAR_FIELD = AUTO_CLEAR;
            INT_TYPE = DEVICE_READ;
        }
        TX_INTR {
            STATUS = ICR.TXDW(0x1);
            ENABLE_FIELD = IMS.TXDW(0x1);
            CLEAR_FIELD = AUTO_CLEAR;
            INT_TYPE = DEVICE_WRITE;
        }
    }
}

sequence device_features {
    with DEVICE_CAPABILITIES {
        ETHFEATURES.TSO = 1;
        ETHFEATURES.RX_CSUM = 1;
        INTERFACE = MII | GMII;
    }
}
sequence pre_transmit
{
    input unsigned int buffer[];
    input int length[];

    XMT[first].tdes1.SOP = 0x1;
    XMT[first].tdes0.pkt_len = SUM_OF(length[first:last]);

    foreach ((x in XMT[first:last-1]), (y in range_of(XMT[first:last-1]))) {
        x.pBuffer = buffer[y];
        x.tdes0.buf_length = length[y];
        x.tdes1.owner = 0x1;
        x.pNext = address_of(x+1);
    }

    XMT[last].pBuffer = buffer[last];
    XMT[last].tdes0.buf_length = length[last];
    XMT[last].tdes1.owner = 0x1;
    XMT[last].tdes1.EOP = 0x1;

    // Descriptor chain ends here.
    XMT[last].pNext = 0;

    TXnHDP0 = address_of(XMT[first]);
}
Extended FSM Specification

SM example_sm {

STATE START idle {
    REG1.RESET = 1;
    <ISR_REG1.RCV(1) WHEN (ISR_REG1.ERR == 0), receive_data>;
    <ISR_REG1.RCV(1) WHEN (ISR_REG1.ERR == 1), error_data_transfer>;
}

STATE receive_data {
    length = REG_LEN;
    buffer = RcvFifo(length);
    <ISR_REG1.DTO(1) WHEN (ISR_REG1.ERR == 0), exit_state>;
    <ISR_REG1.DTO(1) WHEN (ISR_REG1.ERR == 1), error_data_transfer>;
}

STATE EXIT exit_state {
    /* no action/cleanup required */
}

/* ... */
}
Scenarios and HW/SW interface

- Captures the test intent
- Uses Driver APIs specified in the Driver API Interface Layer
- Device agnostic, but specific to a device-function (category)
- Is device specific
- Sequences for configuration, initialization, descriptor management, data transfer, ...
- Interrupt handling
- Publish device capabilities

Scenarios/Use Cases

Driver API Interface

HW/SW Interface
Truly Portable Stimulus

DPS – Device Programming Sequence

© Accellera Systems Initiative
Questions

Are you following the PSWG?
What are your challenges?