TUTORIAL:
Achieving Portable Stimulus with
Graph-Based Verification

25 September 2014
Tutorial Objectives

• Provide overview of technical requirements that is driving us towards a portable stimulus standard

• Describe Graph Based Verification
  – What is a graph?
  – How tests modeled with graphs
  – How graphs enable portable tests
  – Verification reuse from IP to subsystem to full-chip
Today’s Agenda and Presenters

• Introductions
  Josef Derner
  5 mins

• Do we Need it?
  Holger Horbach
  20 mins

• Portable Stimulus for SW
  Frederic Krampac
  25 mins

• Portable and Efficient Graph-Based Tests
  Staffan Berg
  25 mins

• Conclusion
  Staffan Berg
  5 mins
Introducing Today’s Presenters

• IBM
  – Holger Horbach, Verification Engineer

• Breker
  – Frederic Krampac, Sr Applications Engineer

• Mentor Graphics
  – Staffan Berg, European Applications Engineer FV
Portable Tests with Graph-Based Scenario Models

Hopes, Dreams and Aspirations

Frederic Krampac
Breker Verification Systems
Agenda

• Motivation
• Defining Verification Intent
• Reachability Analysis
• Verification Intent Coverage
• Composability
• UVM transactions vs. Software Driven Tests
Motivation

• Separate verification intent from testbench implementation

• Verification Intent covers both stimulus and checks

• Define verification intent once, use it at each stage of verification
Motivation: Portable Tests

Portable across verification engines

Portable Tests with Graph Based Scenario Models

DVCon 2014 Portable Stimulus - Graph Based Verification
Defining Verification Intent

- Feature – capability from spec
  - Check – what precisely must be checked in TB
    - Stimulus – how to sensitize check
Defining Intent: Verifying a Car

Input Constraints to test “forward” outcome
Input Constraints to test “stop” outcome

Three Types of Goals
- Sequence Goal – evaluate ALL children
- Select Goal – evaluate ONE child
- Leaf Goal – has NO children

DVCon 2014 Portable Stimulus - Graph Based Verification
Reachability Analysis

Unreachable Case Detected before Simulation
Verification Intent Coverage

Target A: Hit all input ports

Target B: Hit both error conditions

Target C: Cross A x B (8 total paths)
Automatic Coverage Closure

Automatically Close Coverage Targets
Example: “cross 2 choices and walk all 8 paths”
“To make A happen, generate sequence B = \text{B1 B2 B3}; to make B1 happen, generate C1 C2 C3; to make B2 happen, generate C3”

“Generate sequence B = \text{B1 B2 B3}”

“When A occurs, check that B occurs”

“Generate sequence B = \text{B1 B2 B3}”

“When B occurs, check that A occurs”

“Generate sequence B = \text{B1 B2 B3}”

“To make A happen, generate sequence B = \text{B1 B2 B3}”
Example Design

Serial I/O device
Sequence
- CONFIG: TX/RX mode
- TX_DATA: writing it triggers TX transfer
- RX_DATA: received data
- STATUS: TX/RX completion, CRC ERROR bit

Bridge
- Connects 16-bit sources to 32-bit targets
- Ensures back-to-back 16-bit accesses within 32-bit access
- Address translation

Requests

Responses

16-bit requests

Source 1

Source 2

Bridge 1

Bridge 2

Target 1

Target 2

Target 3

Example Design

DVCon 2014 Portable Stimulus - Graph Based Verification
Traditional Constrained Random

Scoreboard / Functional Coverage unaware of stimulus intent
- not covered
- covered but not checked

Need careful constraints / sequences to hit all cases

Difficult to measure which paths have been exercised
Begin with desired check

Target 1

Randomize Target 2 inputs for check

Delegate each required Target 2 input

Randomize Bridge 2 inputs for what Target 2 needs

Delegate each required Bridge 2 input, etc.

Behavioral Constraints
Composing Software Driven Tests
Composing Software Driven Tests

Prerequisites

Data Flow

Display Controller

Photo Processor

SD Card Controller

SD Card

Display
Composing Software Driven Tests

Constraint: \(PP.\text{quality} == SD.\text{quality}\)
Composing Software Driven Tests

Display Controller

SD Card Controller Read

Photo Processor Decode

SD Card Controller Write

Photo Processor Encode

Display

SD Card

Camera

CCD

SD Card

TrekSoC System Services

Concurrent Scheduling

Memory Allocation

Interrupt Handling

I/O Management

Debug

Result Checking

DVCon 2014 Portable
Stimulus - Graph Based Verification
Multi-Processor Scheduling

- **Cell**
  - **Driver Scenario**
    - **Application Scenario**
      - **Application Scenario**

- **test.c**
  - SD Card Controller Read
  - Photo Processor Decode
  - Display Controller
  - Camera
  - Photo Processor Encode
  - SD Card Controller Write

- **test_cpu1.c**
  - SD Card Controller Read
  - Photo Processor Decode
  - Display Controller
  - Camera
  - Photo Processor Encode
  - SD Card Controller Write

- **test_cpu2.c**
  - Camera
  - SD Card Controller Read
  - Display Controller
  - SD Card Controller Write

- **test_cpu3.c**
  - Camera
  - SD Card Controller Read
  - Photo Processor Decode
  - Display Controller
  - SD Card Controller Write

DVCon 2014 Portable Stimulus - Graph Based Verification
Memory Scheduling
UVM Transaction vs Software Driven Tests

Portable across verification engines

- Portable from unit to full chip
- Portable from unit to full chip
- Portable from unit to full chip
- Portable from unit to full chip

Simulation
Emulation
Prototyping
Post Silicon

SoC
Subsystem
IP
Summary

- Portable stimulus must be extended to checks and coverage -> portable tests
- Graph-based scenario models provide portable tests
- Efficient coverage closure of stimulus and checks
- Portability
  – From unit to cluster/subsystem to full-chip
  – From simulation to hardware platforms to silicon
  – Both transactional and SoC software-driven tests
Portable and Efficient Graph-Based Tests

Staffan Berg
Agenda

• Modeling Tests with Graphs

• Portable Stimulus with Graphs

• Graph-Based Stimulus Applications
Agenda

- Modeling Tests with Graphs
- Portable Stimulus with Graphs
- Graph-Based Stimulus Applications
Stimulus Specification Fundamentals

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

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

• Stimulus scenario described using **Rules**
  – Captures data and control flow aspects of test scenario
  – Describes legal stimulus scenario space
  – Efficient description mechanism

• Rules are compiled into **Graphs**
  – Visual representation of the stimulus model
  – Easy to review

```
rule_graph trans_eng {
  action init, infact_checkcov;

  struct trans {
    meta_action data [unsigned 7:0];
    meta_action addr [unsigned 15:0];
    constraint on_addr {addr < 0x8000}
  }

  trans trans_inst;
  interface fill_trans(trans);

  trans_eng = init repeat {
    fill_trans(trans_inst)
    infact_checkcov
  };
}
```
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

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

```c
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

```
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
};
```
Test Selection and Prioritization

Coverage Strategy

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

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

- Reactive
  - Adapts to changes in the DUT or the verification environment state
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
AXI4 Adjacency Example
Efficient description at transaction level

• Struct captures transaction
  – Key transaction fields
  – Constraints

```c
rule_segment {
    struct axi4_master_rw_transaction {
        meta_action read_or_write=axi4_rw_e[enum READ, WRITE];
        meta_action addr[unsigned 31:0];
        meta_action burst=axi4_burst_e[enum AXI4_FIXED, AXI4_INCR, AXI4_WRAP];
        meta_action burst_length[unsigned 7:0];
        meta_action size=axi4_size_e[enum AXI4_BYTES_1, AXI4_BYTES_2, AXI4_BYTES_3, AXI4_BYTES_4];

        // Align address for wraps
        constraint addr_align_c {};
        if (burst == AXI4_WRAP) {
            (addr % (1 << size) == 0);
        }
    }
    // Limit burst length for wrapping bursts
    constraint burst_len_c {
        burst_length <= 15;
    }
    if (burst == AXI4_WRAP) {
        burst_length Inside [1, 3, 7, 15];
    }
}
```
AXI4 Adjacency Example
Efficient scenario specification and selection

- Specify target address ranges
  - Four target devices
  - Target four ranges in each device

- Generate transaction sequences
  - R/W, address, burst sequences
  - T2->T1 adjacencies are random
  - Burst length, size random

- Pre-simulation size analysis
  - 9216 sequences selected

- Efficient and flexible generation
Agenda

• Modeling Tests with Graphs

• Portable Stimulus with Graphs

• Graph-Based Stimulus Applications
Graph Reuse and Portability

• Graphs are language and environment-independent
  – Self-contained
  – Describe data and sequence scenario

• Graphs mapped to specific environments
  – Communicate data to/from environment
  – Synchronize execution

• Can reuse graphs across environments
  – Create with UVM, reuse with embedded sw
  – Create with C model, reuse in UVM
Graphs Enable Reuse

Testbench Import leverages existing SystemVerilog

• Imports
  – SV classes
    • Fields, constraints
  – Covergroups

• Creates
  – Rules
  – Coverage strategy
  – Testbench integration

• Leverages existing SV
• Raises abstraction level
  – Import transaction classes
  – Build complex scenarios on top
Integration Example
Transaction-level UVM

- Graph runs in a UVM sequence
- Graph nodes set sequence-item fields
- Each graph iteration produces a transaction
- Graph-based sequence is “just another UVM sequence”
Integration Example

Virtual Sequence UVM

- Graph runs in a UVM virtual sequence
- Graph execution
  - Selects sequence parameters
  - Starts sequences
- Sub-sequences can be
  - Graph-based UVM
  - Random or directed
Integration Example

Embedded software

• Graphs call existing ‘C’ methods
  – Select parameter values
  – Sequence method calls

■ Graphs can run
  — Independently
  — Cooperatively
  — Single/multi-core
  — Single/multi-thread

■ Supports
  — Simulation
  — Emulation
  — Silicon
Integration Example
Embedded Software – Coordinated Scenarios

- Hw/Sw scenarios coordinated via a mailbox
  - Mailbox infrastructure provided

- Scenario graph coordinates activity between core(s)/TB

- Example: Coordinated traffic generation
  - Graph selects scenario for VIP to run
  - Graph selects scenarios for cores to run
Agenda

• Modeling Tests with Graphs

• Portable Stimulus with Graphs

• Graph-Based Stimulus Applications
C Model Verification
Portable stimulus across block-Level environments

• Verify C model for high-level synthesis
  – SystemC simulation environment

• Re-run same tests on RTL result of synthesis
  – SystemVerilog simulation environment
  – UVM testbench

• Graph Benefits
  – Same graph used in both environments
  – Highly-productive test creation SystemC
  – Systematic verification
Test Reuse IP to SoC
Portable stimulus enables vertical reuse

- Run IP-level test in UVM environment
  - Scenario driven via BFM
  - Test targets single IP block

- Re-run test in SoC environment
  - Scenario driven via processor core
  - Run tests for multiple IP blocks in parallel

- Graph Benefits
  - Same test scenario run in both environments
  - More-comprehensive testing at SoC level
Graph-Based Portable Stimulus

- High-Productivity Input Specification
  - Familiar data constructs and constraints
  - Formally captures control flow

- Efficient and flexible execution
  - 10-100x more efficient than random
  - Scalable to a simulation farm

- Portable
  - Existing HVLs
  - Embedded software

- Highly Automatable
  - Import/export
  - Analysis
Portable Stimulus Standardization

- Portable Stimulus Proposed Working Group
  - Launched in May 2014

- PSPWG Charter
  - Investigate need and requirements for a Portable Stimulus standard
  - Report to the Accellera board
    - Recommendation on formation of a working group
    - Scope of the proposed working group

- PSPWG Status
  - Weekly meetings
  - Collecting and organizing requirements
  - Users and vendors actively participating
  - Report to Accellera board in November
Questions

Finalize slide set with questions slide
Thank You!

www.mentor.com

www.brekersystems.com