Extending UVM Methodology for Verifying Mixed-Signal Components

by
Abhisek Verma
Synopsys
8th October 2012
Target Audience & Applications

- SoC Integrator
- Analog IP developers

Domains
- Automotive
- Communication
- Multimedia
- PHYs
- LP
- IPs

Applications
- ADC, DAC
- AMPs, Comparators
- PLLs
- Low Power Controller
- Voltage ref
- PHYs
- Custom IPs

Time-based discrete verification
AMS Testbench Technology

• Technology for mixed-signal SoC functional verification
  – Constrained-random verification
  – SystemVerilog

• Enhance VMM/UVM with new constructs for analog blocks

• Provide AMS coverage in SV Testbench Environment

VMM /UVM

AMS TB
UVM-AMS Testbench Overview

*Technology for mixed-signal SoC functional verification*

**Basic Usage**
- Electrical ↔ Real conversion
- Asynchronous analog events
- AMS toggle coverage

**Intermediate usage**
- AMS SystemVerilog assertions
- AMS SystemVerilog testbench
- AMS Checker Library
- SystemVerilog Real Number Modeling

**Advanced usage**
- UVM AMS testbench
- AMS Source generators
- Analog signals can be modeled as real and grouped together in SV interface

```sv
interface ana_comp_if(input bit clk);
    real vin;
    bit o1, o2, o3, o4;
endinterface

module tester;
    reg [31:0] in;
    reg [31:0] out;
    wire [31:0] clk;
    ana_comp_if if;

    always @(posedge clk)
        if.in = in;
        if.out = out;

    initial begin
        #500 in = 0;
        while (1) begin
            #1000 in = next [31:0]; // Drive a random value to the ADC input
        end
    end

    initial begin
        #500 clk = 1'b0;
        while (1) begin
            #1000 clk = ~clk; // Clock generation
        end
    end

endmodule
```

- DUT and testbench are instantiated and connected using SV interface
- Necessary e2r and r2e models inserted automatically

```sv
module tb;
    logic clk=1'b0;
    ana_comp_if if;

    always @ (posedge if.clk)
        if.in = next [31:0]; // Drive a random value to the ADC input

    always @ (posedge clk)
        if.in = if.in + 1;

    A2DConverter dut if.dut;
        if.dut.in = if.dut.in;
```

- DUT should be driven from the testbench, using the SV interface
- ... Better use model available with AMS Testbench base classes
Immediate Assertions

Synchronous immediate assertion of analog node

```verilog
class always @(posedge clk)
    assert (top.vref.analog_node <= 1.8)
    else $error("Node is greater than VDD");
```

Asynchronous immediate assertion of analog node

```verilog
class always @(snps_cross($snps_get_volt(top.ev)-0.6,1))
    assert (top.vref.analog_node <= 1.8)
    else $error("Node is greater than VDD");
```
## UVM-AMS Testbench Checkers

<table>
<thead>
<tr>
<th>Checkers</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sv_ams_threshold_checker</td>
<td>Checks that analog signal remains within a given high and low threshold. Can perform this check synchronously or asynchronously</td>
</tr>
<tr>
<td>sv_ams_stability_checker</td>
<td>Checks that analog signal remains below or above a given threshold. Can perform this check synchronously or asynchronously</td>
</tr>
<tr>
<td>sv_ams_slew_checker</td>
<td>Checks that analog signal rises/falls with a given slew rate(+/− tolerance). Can perform this check synchronously or asynchronously</td>
</tr>
<tr>
<td>sv_ams_frequency_checker</td>
<td>Checks that analog signal frequency is within a given tolerance</td>
</tr>
</tbody>
</table>
Primary Use Model: Functional Verification of an SoC

With AMS TB, it could now contain analog blocks
Other use:
Verifying Analog IP before SoC integration
UVM-AMS Testbench Generators

- Topology to support SV-Top, VAMS / Spice leaves
- Reference model in Verilog-AMS
- Automatic insertion of d2a/a2d for bits
- Constrainable uvm_real to drive analog nodes
- AMS Testbench components (voltage, current)
AMS Testbench Generators
UVM

Sine Voltage Gen
• Vmax=1.0V,
• Vmin=-1.0V
• F=1.0MHz

Construct sin Wave generator. Default is auto-run throughout run_phase()

```java
class my_env extends uvm_component;

  ... sv_ams_sine_voltage_gen#(-1.0, +1.0, 1.0E6) sGen_IN;

  function void build_phase(uvm_phase phase);
    super.build_phase(phase);
    uvm_resource_db#(virtual ams_src_if)::
      set("*", "uvm_ams_src_if", aif, this);
    sGen_IN = sv_ams_sine_voltage_gen#
      (-1.0, +1.0, 1.0E6)::type_id::create("sine", this);
  endfunction
```
UVM-AMS Testbench Benefits

• Clear methodology for mixed-AMS SoC verification
  – Enable consistent digital and analog verification
  – Tight integration between VCS and CustomSim

• Covers Holes in mixed-AMS Block-level verification
  – Synchronous and Analog Asynchronous verification
  – Usage of Ref models for self-checking and quicker simulation

• Ecosystem
  – Verification planning
  – Regressions
  – Coverage convergence
  – Self-checking