Universal Verification Methodology (UVM)-based Random Verification through VCS and CustomSim in Analog Mixed-signal Designs for Faster Coverage Closure

Warren Anderson
Ravi Ram
AMD
Vijay Akkaraju
Synopsys
The Problem

• Full verification of analog+digital functions in a mixed-signal design

• FastSPICE has significant limits:
  – Long sim times (12+ hours per test) results in:
    • Only a small fraction of regression run in mixed mode. (often less then 5%)
    • Significant coverage holes
  – Limited randomization
    • SPICE models do not lend themselves to today’s verification flows

• Future designs expected to be more complex
The Solution

Use Verilog-AMS for mixed signal models and simulations:

• Pros:
  – Accuracy approaches SPICE
  – Simulation time >10x faster, better resulting in:
    • More tests run in mixed-signal mode.
    • More coverage points hit
  – Enables randomization on the analog side

• Challenges
  – Development time for Verilog-AMS models
  – Verilog-AMS to transistor equivalence checking
Outline of this talk

• Target design
• Verification environment
• Verilog-AMS models
• Metrics and measurements
• Summary and conclusions
Design

- PHY layer of an AMD APU processor DDR sub-system
- Bidirectional bus between processor and DRAM memory
- Analog circuits:
  - Output driver
  - Input receiver
  - Input line impedance termination
  - Calibration
  - Reference voltage generation and distribution
  - Current generation and distribution
  - DLL
- Each of the above has an digital-analog interface
DDR PHY Design
Outline of this talk

• Target design
• Verification environment
• Verilog-AMS models
• Metrics and measurements
• Summary and conclusions
Universal Verification Methodology (UVM)

• System Verilog based, industry standard verification methodology.
  – Maintained by, “Accellera” – standards organization for the EDA industry.

• Developed to:
  – Increase design verification productivity by code reuse.
  – Optimize testbench and testcase development time.
  – Consistent Verification flows across projects, designs, and blocks.
Verification Environment

- Adapted a UVM flow
- Separate agent for each 32 bit channel
- Each agent has an individual:
  - Sequencer
  - Driver
  - Monitor
- Randomization:
  - Through a global config file
  - System Verilog constraint construct
- Mixed-signal modeling:
  - Swap Verilog with a Verilog-AMS model
  - Run the same sequences
  - Randomization for analog using model parameters
Verification Environment
Randomization of analog portion using model parameters

- **Goal:** use UVM to configure the AMS model
  - AMS models use real variables to control behavior
- **Current LRM:**
  - Real variable randomization not legal
- **Workaround flow:**
  - Use System Verilog constraint construct for integers
  - Change SV constrained integers to reals and feed AMS model
- **Example code:**

  ```
  real vswitch;
  vswitch=real ’( $urandom range ( xVswitch ) ) / yVswitch ;
  ```

- **Enable randomization:**
  - From command line or default constraints
- **Future:**
  - Move to “real random” when SV LRM support standardized
Outline of this talk

- Target design
- Verification environment
- Verilog-AMS models
- Metrics and measurements
- Summary and conclusions
Mixed Signal Coding Architecture

• Model structure is key to enable
  – Simple Verilog-AMS coding
  – Schematic equivalence checking

• Design hierarchy for Verilog-AMS modules
  – Define Verilog-AMS module boundary around the analog function only
    • Place as much digital function as possible in digital-only modules
  – Schematics must match hierarchy and pins
  – Verilog-AMS module or module groups must represent a function that can be simulated
    • Best if a schematic characterization block
Verilog-AMS Model Example
Verilog-AMS Example: output driver

• Verilog-AMS model boundary selected to contain only the analog function
  – Driver pullup or pulldown individually
• Hierarchy and pins match between schematic and Verilog-AMS model
• Module represents a function to be characterized in SPICE simulations
  – Output functionality
  – Drive strength versus compensation code
Verilog-AMS Example: output driver

parameter real Rbank = 240.0;
// Resistance of bank, scalable across different drive strength banks
integer BaseWeight = 64 * 3.5 / 2.0;
// Unit of weight is the LSB of the binary legs
real Rbank_unit = 240.0;
// Unit cell used for calibration of the driver versus code curve here
real Rint = 132.0;
// Portion of the driver that is series resistance

real Rscale, Tscale, Ncal_ena;
always @* begin
    Rscale = (Rbank / Rbank_unit) / PFactorResProc / PFactorResTemp;
    Tscale = PFactorXtorProc * PFactorXtorTemp / (Rbank / Rbank_unit);
    Ncal_ena = BaseWeight + {8{ng_dat}} & ncal[7:0];
end
Verilog-AMS Example: output driver

electrical pad_int;
// Intermediate node between the pad and supply
analog begin
    // Resistor portion
    V(PAD_VIO, pad_int) <+ I(PAD_VIO, pad_int) * Rint * Rscale;

    // Transistor portion
    I(pad_int, VSS) <+ Ncal_ena * Tscale *
    ( xivp4 * V(pad_int, VSS) +
      xivp3 * V(pad_int, VSS) * V(pad_int, VSS) +
      xivp2 * V(pad_int, VSS) * V(pad_int, VSS) *
      V(pad_int, VSS) +
      xivp1 * V(pad_int, VSS) * V(pad_int, VSS) *
      V(pad_int, VSS) * V(pad_int, VSS) );
end
UVM and Verilog-AMS Linkage

- Use UVM to randomize analog variables
  - Process, voltage, temperature sensitivity
  - Analog behavioral properties: offset, noise, INL

```verilog
real PFactorResProc = 1.0 from [0.9:1.12];
// Process factor for resistance process variation
// Minimum value is max R, maximum value is min R
real PFactorResTemp = 1.0 from [0.97:1.0];
// Process factor for resistance temperature variation
// T=0 is 1.0, T=100 is min value
real PFactorXtorProc = 1.0 from [0.74:1.29];
// Process factor for transistor process variation
// Min value is slow process, max value is fast process
real PFactorXtorTemp = 1.0 from [0.85:1.0];
// Process factor for transistor temperature variation
// T=0 is 1.0, T=100 is min value
```
Model Fit: output driver

Solid: SPICE
Dashed: AMS

- FF, 0C, min R
- TT, 0C, typ R
- SS, 100C, max R

Current (10^-3 A) vs. Voltage (V)
Mixed signal coding philosophy

• Code only for end goal
  – Functional verification in this case

• Write models to include only the behavior that has a functional impact
  – No need to model all behaviors that could be modeled
    • Edge rates
    • Delays
    • Bandwidth

• Keeps the analog portion of the simulation to a minimum
**Example: input receiver**

- **ESD CDM Device**
- **RxIn_VIO**
- **R_{ESD}**
- **Internal VREF Generator**
- **Amplifier Chain**
  - **DFE 0**
  - **DFE 1**
  - **Nominal**
  - **Folded-cascode + cml2cmos**
- **Data Slice**
  - **Dll_ClkData (½ rate clk)**
  - **Dll_ClkDataX (½ rate clk)**
- **Edge Slice (CDR)**
  - **Rx_RcvEdgeEven**
  - **Rx_RcvEdgeOdd**
  - **D3 Mode (or RDQS Strobe Data mode)**
  - **Rx_RcvDataOdd**

**Diagram Details**:
- **x2**
- **Low-speed DDR3 (even/odd)**
- **Rx_RcvBypass**
- **Retiming FF**
- **Rx_RcvDataEven**
- **Rx_RcvDataOdd**
- **Csr_VrefDnom<6:0>**
- **Csr_VrefDfe<6:0>**
- **Csr_VrefMargining**

**Networks**:
- **EDF Ctrl**
- **DQ**
- **Edge**
- **Retiming FF**
- **Rx_RdqsStrobe[X]**
Verilog-AMS Example: input receiver

- Verilog-AMS model boundary selected to contain only the analog function
  - Receiver amplifier, mux switches, comparator, reference generator
- Hierarchy and pins match
- Module represents a function to be characterized in SPICE simulations
  - Amplifier function and analog parameters
    - Gain, offset, noise as parameters
  - Reference DAC step size, error as parameters
Code Example: receiver comparator

module ddr_comparator(input p, input n, output out);
    real CmprOffset = 0.0m;  //default
    real CmprHyst = 1.0m;  //default
    real CmprDelay = 0.0;  //default
    electrical p,n,vid;
    reg out;
    real threshhi = CmprOffset + (0.5*CmprHyst);
    real threshlo = CmprOffset - (0.5*CmprHyst);
    always @(above(V(vid)-threshhi)) begin
        #(CmprDelay/1e-12) out = 1'b1;
    end
    always @(above(threshlo-V(vid))) begin
        #(CmprDelay/1e-12) out = 1'b0;
    end
    analog begin
        @(initial_step) begin
            V(vid) <+ V(p,n);
        end
    end
endmodule
Outline of this talk

• Target design
• Verification environment
• Verilog-AMS models
• Metrics and measurements
• Summary and conclusions
V-AMS Vs Verilog simulation

- Example: I/O driver signal to the DRAM
  - Voltage-mode driver with de-emphasis equalization
  - 8-bit data bursts
Verilog-AMS simulation results
# V-AMS Vs Spice simulation

<table>
<thead>
<tr>
<th>Test Name</th>
<th>Verilog-AMS wall-clock time</th>
<th>FastSPICE wall clock time</th>
<th>Simulation speed-up</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output impedance calibration cycle</td>
<td>3 minutes</td>
<td>28 hours</td>
<td>560x</td>
</tr>
<tr>
<td>Output driver functionality</td>
<td>6 seconds</td>
<td>30 minutes</td>
<td>300x</td>
</tr>
</tbody>
</table>
Outline of this talk

• Target design
• Verification environment
• Verilog-AMS models
• Metrics and measurements
• Summary and conclusions
Summary and Conclusions

• Verilog-AMS extremely beneficial to increase verification coverage in mixed-signal designs
  – Enables large number of functional test cases within a standard verification framework
  – Improved simulation time over FastSPICE
  – If models written correctly, accuracy loss compared to SPICE mode can be contained

• Link to UVM enhances coverage through randomization across the analog space
  – Vary process, temperature, offset, noise, etc.
Next Steps

• Apply Verilog-AMS models to more projects
  – Models can be re-used in many cases
  – Advantage that models can be put in place well before circuit design is completed

• Add more variables to the Verilog-AMS randomization to increase verification test coverage

• Enhance the environment when LRM updates come through