Comprehensive AMS Verification using Octave, Real Number Modelling and UVM

John McGrath, Patrick Lynch, Ali Boumaalif
Xilinx, AMS Group, Ireland
AMS System Level Sim Challenges

• Simulate analog-digital interaction
  – Calibration, feedback, complex interfaces
  – Closed-loop response of the system

• Realistic Stimulus
  – E.g. for a Data-Converter
    • Single tone, multi-tone, coherent frequencies
    • Random stimulus / frequencies
    • Analog transactions

• Real Performance Analysis
  – Time and Frequency-domain based measurements
  – Analog/Signal-Processing predictor model
  – Analog design coverage/assertions
AMS Flow Overview

• 3 Core Components
  – Automation of Netlist based SV-RNMs
    • System-Verilog Real Number Models
  – SystemVerilog <-> Octave interface
  – UVM environment

• 3 Core Constraints
  – Portable methodology across all simulators
  – Low / Zero cost overhead
  – Performance / Run-time

© Accellera Systems Initiative
Traditional Verilog Testbench

• Digital Verification
  – RTL/Gates

• Analog Model
  – High-level behavioural model of analog
  – Limited by Verilog-2001 language

• Co-Simulation
  – Model ‘swapped’ for Top-level spice
  – Used for Signoff

• Not scalable...

© Accellera Systems Initiative 2015

Accellera Systems Initiative 4
Why Use SystemVerilog RNMs?

• Why use RNMs?
  – Analog schematic is golden
  – Increasing analog design complexity
  – Digitally assisted analog
  – Top-Level co-simulation not feasible

• Using SV as a modelling language
  – SystemVerilog has real-ports
    • ‘real’ used to model analog (V,I) nodes
  – Fine-grained modelling of ‘simple’ analog building blocks
  – Can be incorporated into regressions
  – Fast yet accurate simulation

© Accellera Systems Initiative
Optimising Analog Schematics

- Enables RNM automation, analog + digital modelling

1) Model at lowest Feasible level
2) Partition local analog/digital
3) Use signal flow
4) Encapsulate analog feedback

Schematic

Optimal Schematic

© Accellera Systems Initiative 2015
Netlist Modification Flow

- RTL files
- RNM files
- Schematic DB
- UVM Env

RNM Netlist Gen

.sv (RNM)
.v (.Netlist)

- RNM
- Propagated RNM
- Digital logic (preserved)
- Raw netlist
- "Logic" port
- "Real" port
RNM Validation

• 3 main methods
  1. In-system co-sim – replacing individual RNMs with Spice
  2. Side-by-side module co-simulation – RNM vs Spice
  3. EDA Tools to compare models

• Following Schematic Guidelines
  – Simpler/lower-level models => easier to verify
Modelling Enhancements in SV-2012

• **nettype** enhancement
  – Enables more advanced signal modelling
  – Handles contention with multiple drivers per node

• **interconnect** enhancement
  – Automatically adapts ‘wires’ to type of connected signal
  – Could replace netlist tracing script

• However…
  – Partial support in some simulators (to date)
  – Extra license costs - AMS type licenses
SV-RNM Results

• Performance:
  – Closed-loop calibration of a complex analog block
    • SV-RNM: 16 blocks in parallel – 5 minutes
    • Co-Simulation: 1 block – 18+ hours

• Issues Found
  – Analog schematic hook-up bugs
  – Calibration algorithm mismatches / feedback delays
  – Many polarity/sign inversions

• Enhanced Debug
  – Netlist preserves inter-block dependencies
    • Supplies, bias, enables, register settings, etc.
  – Reduced iteration time between simulations
  – Much faster co-sim bring-up as a result
SV-RNM Summary

• Low-cost, re-usable, high performance

• Other advantages
  – SV-assertions and coverage deep into analog hierarchy
  – Enables granular mix-and-match co-sim
  – Can catch analog bugs

>90% of modules netlisted

>96% of model netlisted
SV-Octave Overview

• GNU Octave
  – Open source Matlab® clone
  – Slower than Matlab, but compatible with mode analysis code

• Why?
  – Analog analysis, stimulus, algorithms already in Matlab code
  – Re-use the golden code from analog designers
  – Directly called from SystemVerilog testbenches

• Advantages of integrating with SV
  – Enables frequency-domain metrics capture
  – Realistic performance measurements at system level
  – Dynamic interaction - no management of vector files
Octave Integration

- **SystemVerilog DPI**
  - Is a C interface for extending SV
- **Octave**
  - Has a C interface for embedding Octave
- ‘Bridge’ SV and Octave via the DPI

```c
fft(data);
```

Testbench

- SystemVerilog
- Embedded Octave Interpreter
- C-DPI Wrapper
SV-Octave Example

SystemVerilog IF

```systemverilog
// General Octave Setup Functions
import "DPI-C" function void oct_init(string path="");
import "DPI-C" function void oct_exit();

// Call Octave C Wrapper Functions
import "DPI-C" function void oct_fft(input int dyn_arr[],
       output real res_arr[]);

// result struct
typedef struct { ... real sfdr; } fft_result;

// Calls oct_fft function, and returns results in a struct for each access
function fft_result fft(input int dyn_arr[]);
begin
  oct_fft(dyn_arr, res_arr);
  fft.sfdr = res_arr[0];
endfunction

// Initial begin
for(int i=0; i<1024; i++) begin // Gather Data
  @(negedge adc_clk);
  fix_arr[1] = adc_data;
end
result = fft(fix_arr); // Call Octave
$display("SNR: ", result.sfdr); // Report result
```

SystemVerilog TB

```
include <octave/octave.h>
#include <svdpi.h>

// Macros
#define SV.ARR_REAL(arr, idx) *(double *)svGetArrayElemPtr(arr, idx)

// Octave FFT Wrapper
extern "C" void oct_fft (const svOpenArrayHandle dyn_arr,
       const svOpenArrayHandle res_arr)
{
  // 1) Prepare inputs - arrays are converted to Matrix types for octave
  Matrix data_in = Matrix(i, len);
  for(int i=0; i<svLength(dyn_arr, 1); i++) {
    data_in(i) = SV.ARR_INT(dyn_arr, i);
  }

  // 2) Call the function
  octave_value_list func_in;  // create the array (matrix) to input
  func_in(i) = data_in;       // pass data
  octave_value_list out = eval("'analyzer', func_in);

  // 3) Return the results
  for(octave_idx_type i=0; i<out.length(); i++) {
    if(i < svLength(res_arr, 1)) {
      SV.ARR_REAL(res_arr, i) = out(i).double_value();
    }
  }
```

DPI-Wrapper
SV-Octave Summary

• Results
  – Full access to all Octave processing from SV testbench
  – 1.6x performance increase over vec-file based methods
  – No files to manage
  – Can use SV randomization for stimulus and processing

• Use with Co-Simulation
  – Provide real stimulus for analog
  – Real system-level performance evaluation
Integration into UVM

- SV-RNM + Octave integrated into UVM Environment
Summary

• Combination of UVM, SV-RNMs and Octave
• Significant increase in analog verification coverage
  – Analog functionality
  – Assertions deep into analog schematic design
• Enables System-Level verification
  – System-level performance checks
• High performance simulation
  – Low run-time
• Low-Cost
  – Only standard simulator licenses used
Questions