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

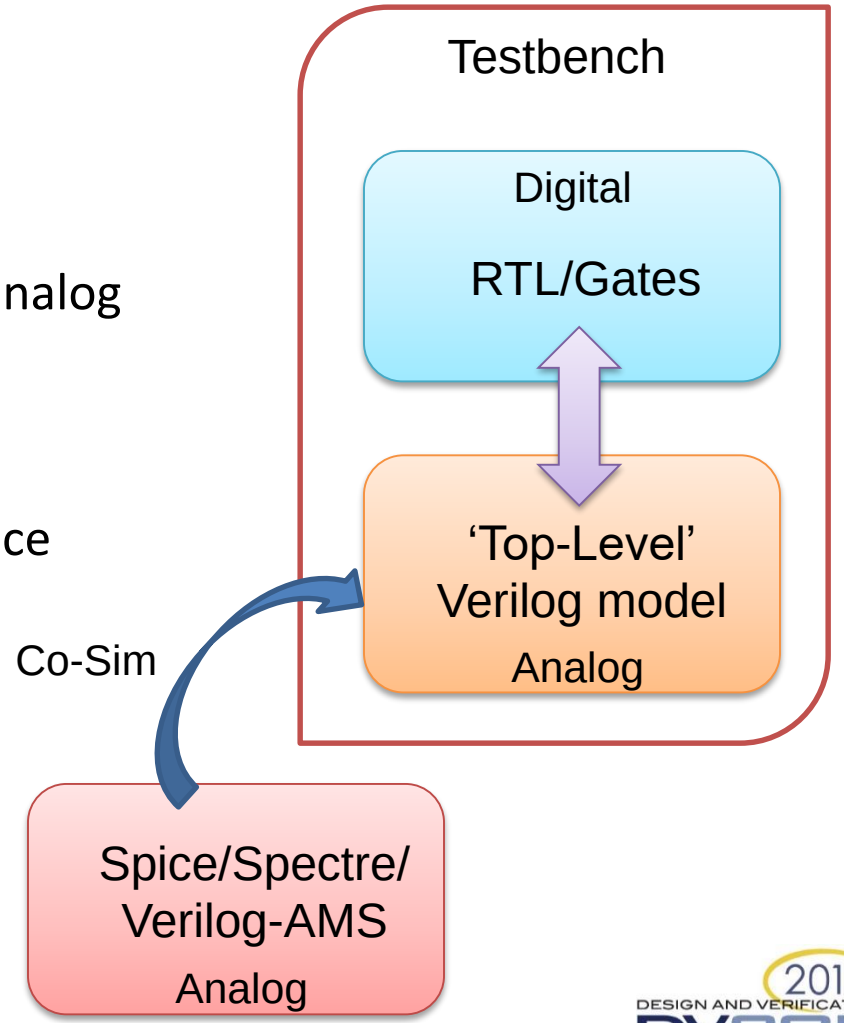
AMS Flow Overview

- 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

Traditional Verilog Testbench

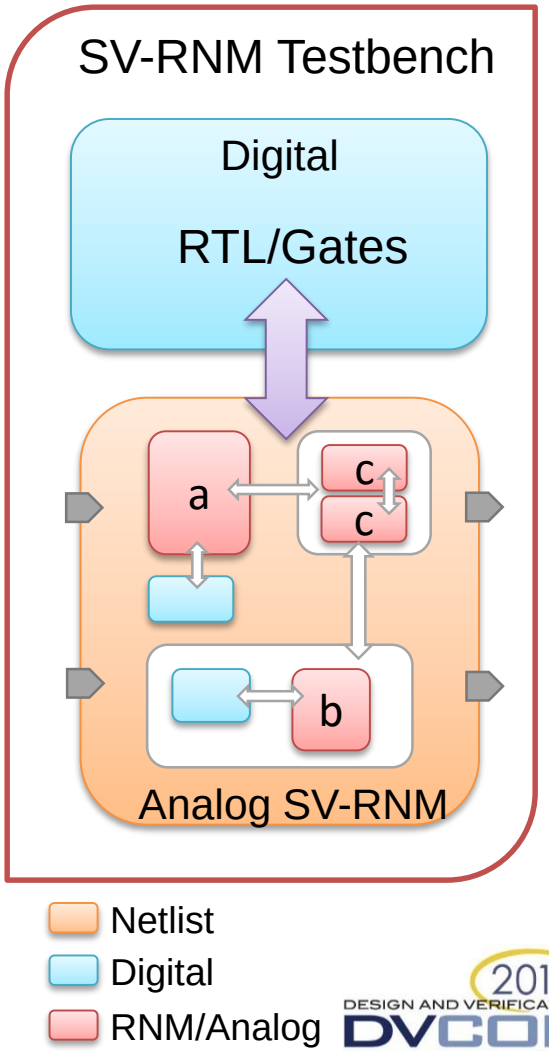
- Digital Verification
  - RTL/Gates
- Analog Model
  - High-level behavioural model of analog
  - Limited by Verilog-2001 language
- Co-Simulation
  - Model ‘swapped’ for Top-level spice
  - Used for Signoff
- Not scalable…



Why Use SystemVerilog RNMs?

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

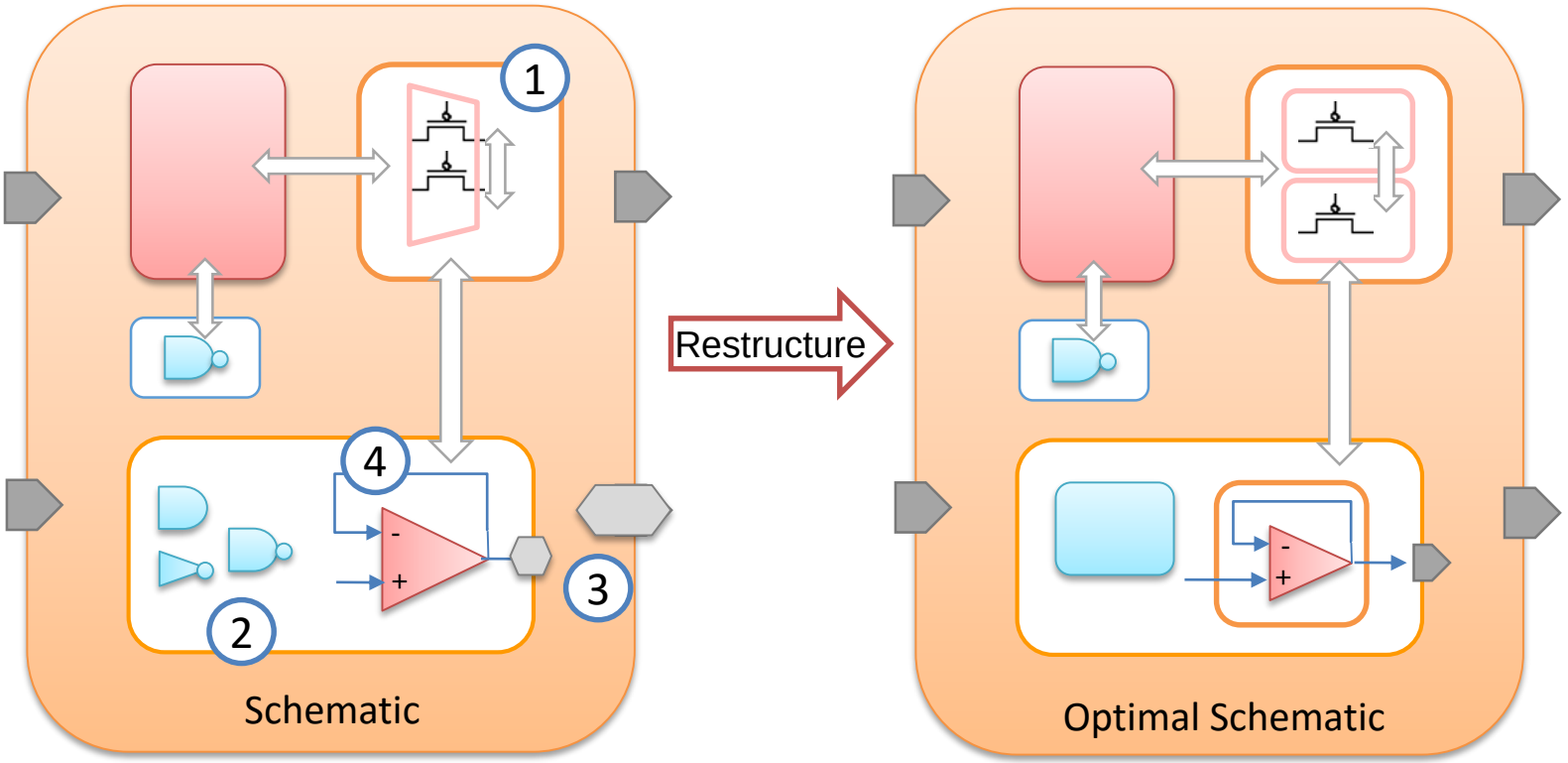
- Using SV as a modelling language
  - SystemVerilog has real-ports
    - 'real' used to model analog (V,I) nodes
  - Fine-grained modelling of 'simple' analog building blocks
  - Can be incorporated into regressions
  - Fast yet accurate simulation



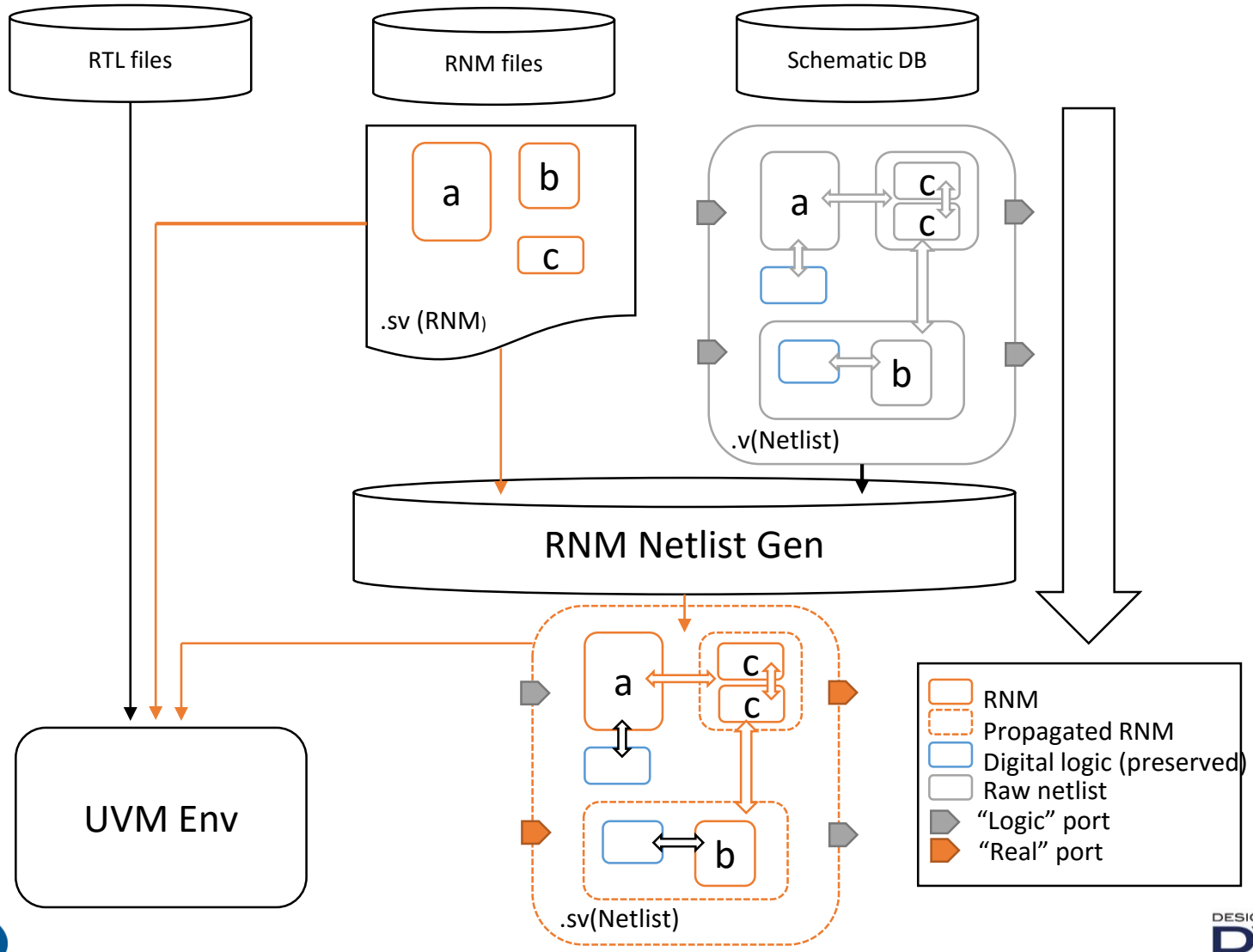
Optimising Analog Schematics

- 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

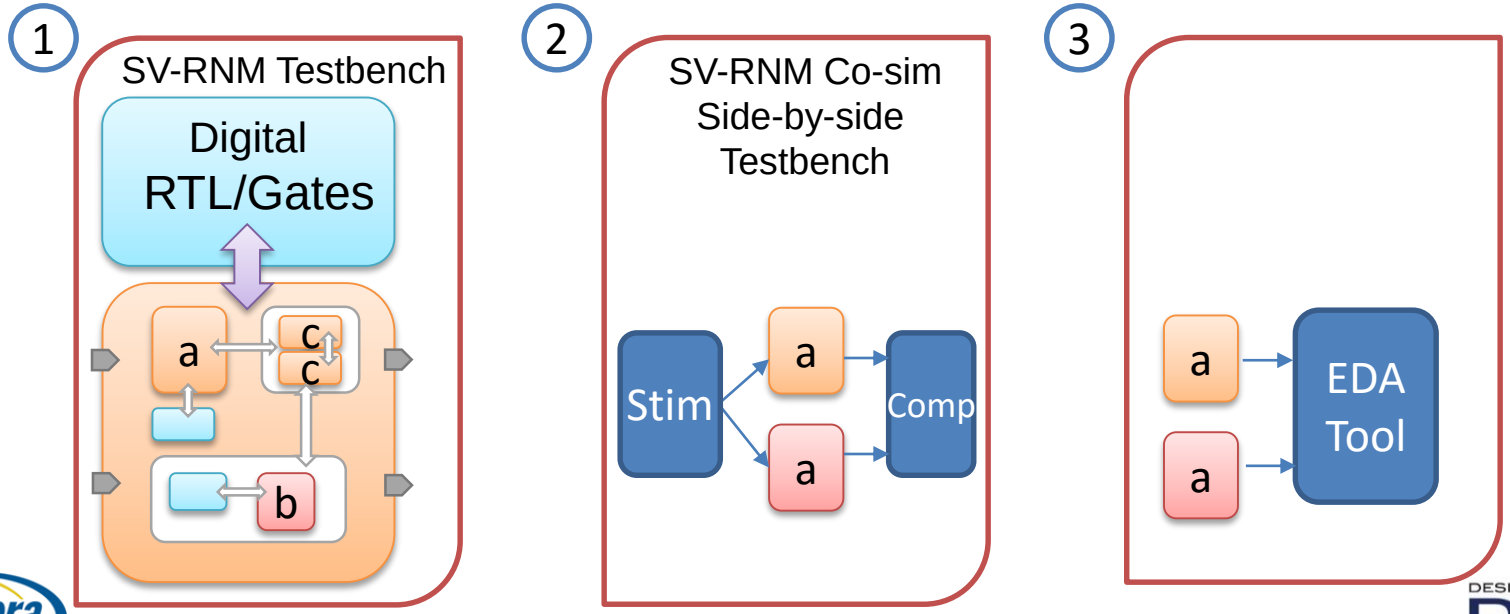


Netlist Modification Flow



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

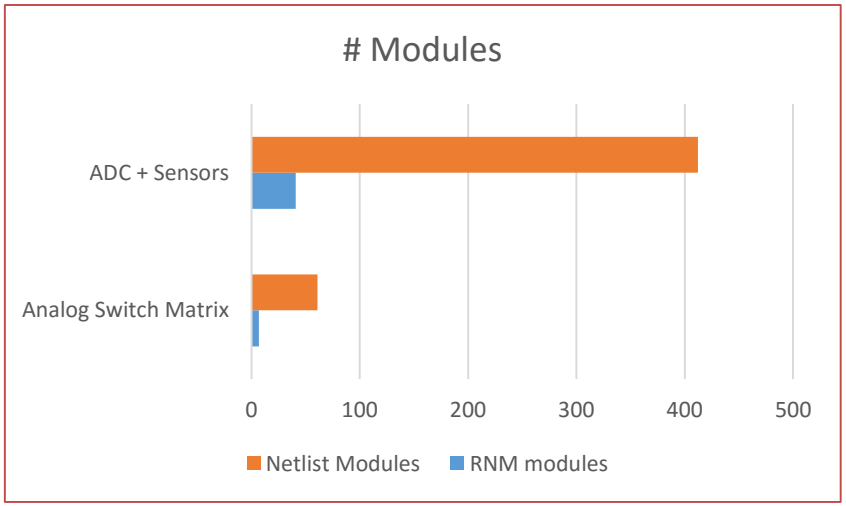
- However…
  - Partial support in some simulators (to date)
  - Extra license costs - AMS type licenses

SV-RNM Results

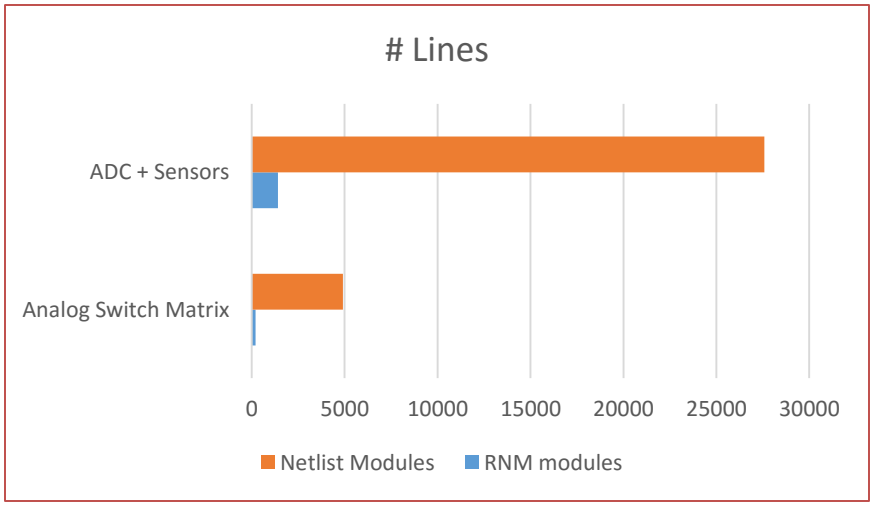
- Performance:
  - Closed-loop calibration of a complex analog block
    - SV-RNM: 16 blocks in parallel – **5 minutes**
    - Co-Simulation: 1 block – **18+ hours**
- Issues Found
  - Analog schematic hook-up bugs
  - Calibration algorithm mismatches / feedback delays
  - Many polarity/sign inversions
- 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



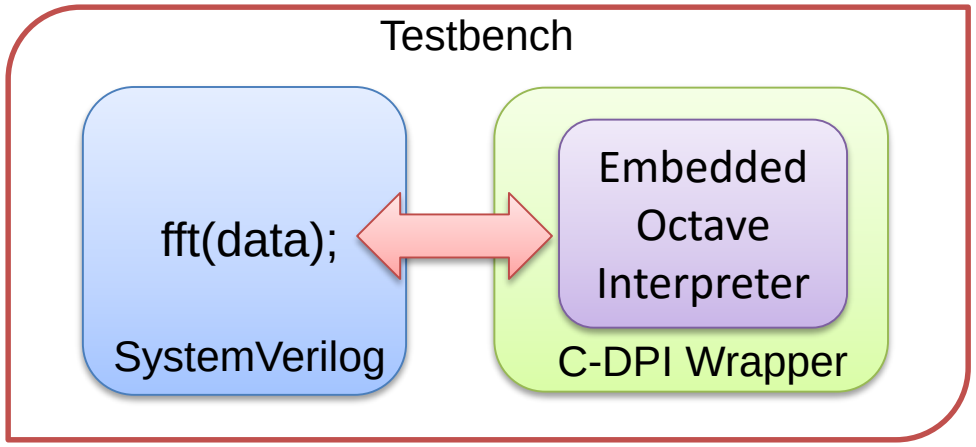
>96% of model netlisted

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



SV-Octave Example

SystemVerilog IF

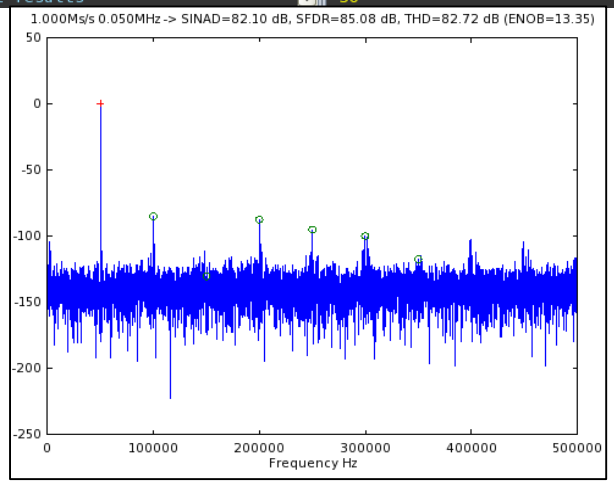
```
 3 // General Octave Setup Functions
 4 import "DPI-C" function void oct_init(string path="");
 5 import "DPI-C" function void oct_exit();
 6
 7 // Call Octave C Wrapper Functions
 8 import "DPI-C" function void oct_fft(input  int  dyn_arr[],
 9                                      output real res_arr[]);
10 ....
11
12 // result struct
13 typedef struct { ... real sfdr; } fft_result;
14
15 // Calls oct_fft function, and returns results in a struct for each eaccess
16 function fft_result fft(input int dyn_arr[]);
17 real res_arr[10];
18 begin
19   oct_fft(dyn_arr, res_arr);
20   fft.sndr = res_arr[0];
21   ....
22 end
23 endfunction
-- INSERT --                                          17,18         Bot
```

```
 7 initial begin
 8     for(int i=0; i<1024; i++) begin   // Gather Data
 9         @(negedge adc_clk);
10         fix_arr[i] = adc_data;
11     end
12     result = fft(fix_arr);            // Call Octave
13     $display("SNR: %f", result.snr);  // Report results
```

SystemVerilog TB

```
 1 #include <octave/octave.h>
 2 #include <svdpi.h>
 3 ....
 4
 5 // Macros
 6 #define SV_ARR_REAL(arr,idx) *(double *)svGetArrElemPtr1(arr,idx)
 7
 8 // Octave FFT Wrapper
 9 extern "C" void oct_fft (const svOpenArrayHandle dyn_arr,
10                          const svOpenArrayHandle res_arr)
11 {
12   // 1) Prepare inputs - arrays are converted to Matrix types for octave
13   Matrix data_in = Matrix(1, len);
14   for(int i=0; i<svLength(dyn_arr, 1); i++) {
15     data_in(i) = SV_ARR_INT(dyn_arr,i);
16   }
17
18   // 2) Call the function
19   octave_value_list func_in;       // create the array (matrix) to input
20   func_in(0) = data_in;            // pass data
21   octave_value_list out = feval("fanalyze", func_in);
22
23   // 3) Return the results
24   for(octave_idx_type i=0; i<out.length(); i++) {
25     if(i < svLength(res_arr,1)) {
26       SV_ARR_REAL(res_arr, i) = out(i).double_value();
27     }
28   }
29 }
30
```

DPI-Wrapper



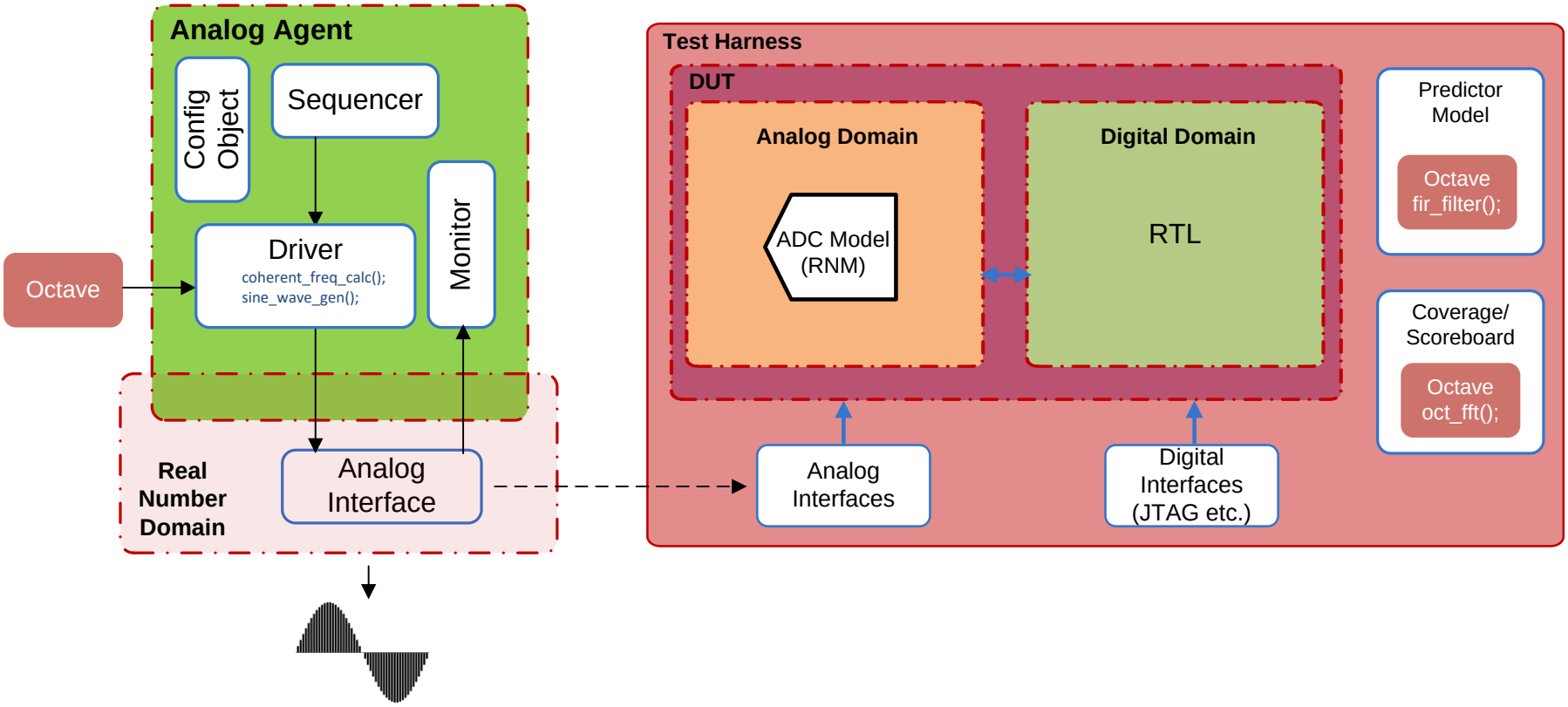
SV-Octave Summary

- 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