Practical Considerations for Real Valued Modeling of High Performance Analog Systems

Dushyant Juneja
Senior CAD Engineer
Analog Devices
Bangalore, India
Dushyant.Juneja@analog.com

Siddharth Prabhu
CAD Development Engineer
Analog Devices
Wilmington, MA, USA
Siddharth.Prabhu@analog.com

Syam Veluri
Principal CAD Architect
Analog Devices
Wilmington, MA, USA
Syam.Veluri@analog.com

Abstract

Real Valued Models (RVMs) have lately gained attention as one of the most promising behavioral/functional verification approaches for modern mixed signal SoCs. Aptly so, since RVMs provide, amidst other benefits, seamless integration with state-of-the-art digital verification methods such as Metric Driven Verification (MDV). However, high performance analog modeling is one of the challenging areas for RVM usage. The paper elaborates on these challenges both in modeling and other aspects such as testbench integration and development, and presents workarounds towards the same.

I. Introduction

Real-Valued Modelling is a behavioral modelling technique that can be used to simulate the functions of analog blocks within a digital simulation. The concept is to allow a digital simulation to operate with a real-number valued nets, wires and not limiting to internal module variables. This also provides analog-like characteristics for modules designed to work with those values in a mixed-signal system.

This technique is not something new to our industry and it has been supported in languages dating back to Verilog-A. However, the technique is becoming more widespread among verification teams involved with complex mixed-signal designs and introduction of standards like SystemVerilog and Universal Verification Methodology (UVM). In recent days it is very evident on how much improvements has been made in high-level abstraction modelling languages like SystemVerilog, for nets (user-defined nettypes, UDTs) to carry real-number values. Also latest enhanements to support the idea of multiple drivers on a net with user-defined resolution functions (UDRs).

The key advantage of real valued modelling is speed over accuracy and modeling details captured. These real-valued models do not have to incorporate analog or iterative solvers but use simple straightforward transfer-functions to emulate the behavior of analog circuits in a digital-like environment. The recommended abstraction level is much higher and closer to sub-system level blocks where there is significant digital/analog signal interactions across mixed-signal boundary. These models being event-driven in nature, it suits for faster top-level digital functional verification environment. These models do fit well in various other verification flows, also provide ability to perform assertions on real-valued nets, collect coverage details and perform analysis.

The work intends to highlight experiences with RVMs, especially using the SystemVerilog 2012 standard that comes with rich RVM updates. Section II discusses the issues faced in modeling common analog topologies such as feedback amplifiers and DACs, and develops the case for the usage of advanced RVM features. Section III further talks about second order concerns such as block level connectivity/netlisitng and type-coercion in using RVMs.

Section IV discusses issues with RVM based testbench development. Though the work has been implemented using SystemVerilog, the concepts are generalized and can be applied for all HDLs with RVM support. SystemVerilog lookalike psuedo-code based syntaxes are used throughout the work in favor of emphasizing the concept, instead of the concerned syntaxes.

II. Analog Modeling Concerns

Most digital HDLs such as SystemVerilog and VHDL rely on discrete time, event driven evaluation of concerned parts in the design. Analog designs, on the other hand, use techniques such as continuous time feedbacks. Even with the evolution of real number models, the underlying event driven simulation method remains and fails to completely capture every analog effect. As a result, these models does not perform accurately or satisfactorily for blocks that involve complex analog interactions. Also the models are not useful for detailed low-level modeling or very close to block level simulation.

The current section validates the above statement with the aid of some simple analog topologies. The work goes ahead to describe the intuitive workarounds found in the process, while also justifying the need for complex RVM nettypes.

A. Feedback Circuits

Analog circuits usually employ generous use of feedbacks in various forms. Examples range from simple, continuous time circuits such as feedback amplifiers to complex mixed signal circuits such as PLLs and calibration loops. While being deceptively simple, former pose a major challenge for digital modeling.

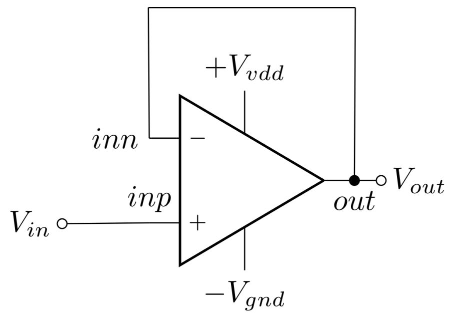

Fig. 1: Basic Non-Inverting Amplifier

The case of a non-inverting amplifier is presented in [1], whereby a simple, stable configuration as shown in Fig. 1 is shown to go unstable in event driven modeling. A basic, hard-limited model of the core amplifier for above figure could have the computation as in Listing 1.

```
// Amplifier gain
real gain = 1000 ;

// Output computation
always @( inp, inm, vdd, gnd ) begin
  tmp_in = inp - inm ;
  tmp_out = gain * tmp_in ;

  // Hard limiting the output
  if ( tmp_out > vdd )      : tmp_out = vdd ;
  else if ( tmp_out < vss ) : tmp_out = vss ;

  #1 out = tmp_out ;
end
```

Listing 1: An Ideal Opamp Model

However, the output can be seen to be oscillating between `vdd` and `vss` values, showing unstable behavior. The cause of the unstable behavior is attributed in [1] to the extra pole added by the time step delay, causing the loop to change as in Fig. 2. The transfer function changes as follows:

$$H_{expected} = G \tag{1}$$

$$H_{actual}(z) = \frac{V_{out}}{V_{in}} = \frac{G}{1 + G \cdot z^{-1}} \tag{2}$$

Here, where G is the gain of the amplifier.

As observed, the intrinsic z-domain pole is at $z = -G$, depicting that -

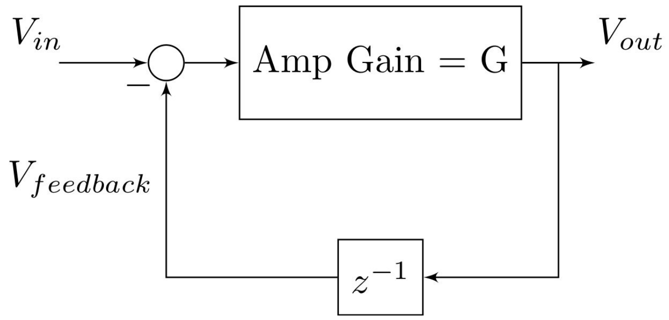

Fig. 2: Equivalent System for the Non-Inverting Amplifier in SV

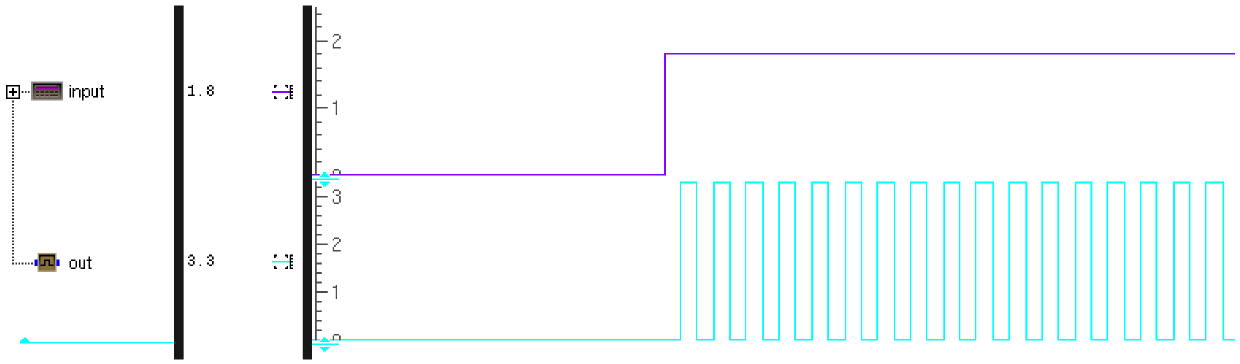

Fig. 3: Unstable output from ideal amplifier modeling. The oscillations are due to hard limits put on amplifier output.

- The system would be unstable for all $G > 1$, which is bound to happen for any normal opamp.
- The instability increases with G - higher the gain of the opamp, more the instability at the output.

The simulated output for the amplifier is as shown in Fig. 3, and shows oscillations for a simple step input, evidencing instability. The pole reflects in Listing 1 as the delay between input and output at line 13. However, this cannot be easily dealt away with. This is because in case of the unity gain feedback amplifier in Fig. 1, it creates a zero delay loop and locks the simulator in delta cycles at time 0. An analog simulator tackles the situation using simultaneous solution at input and output nodes using iterative solvers. However, the facility is difficult to implement with an event driven evaluation, and further can give significant speed penalty.

It is instructive at this stage to realize the effect of the step delay. The delay effectively introduces a 180°phase difference for the ideal amplifier. However, every opamp has it's own intrinsic delay, manifesting as the bandwidth of the opamp. This gives a sense of delay to the amplifier. If the digital simulator time step can be lesser than this, the phase difference could be reduced to make the simulation stable and closer to the analog simulation. The limit for the simulation time step can be found via mathematical analysis as in [4], [5], as follows. While the result itself is well known and applied [4], a mathematical treatment is presented for the sake of intution and deriving further insights into what happens during the simulation.

The transfer function for a band-limited opamp is given as:

$$H_{amplifier}(s) = \frac{G(dc)}{1 + s\tau} \tag{3}$$

Here, $G(dc)$ is the DC gain of the amplifier. If the simulator time step can be defined as δ, the transfer function for the unstable system can be mapped to continuous time domain using s to z domain map ($z = e^{s\delta}$) as follows:

$$H_{system}(z) = \frac{H_{amplifier}}{1 + H_{amplifier} \cdot z^{-1}} \tag{4}$$

$$\implies H'_{system}(s) = \frac{H_{amplifier}}{1 + H_{amplifier} \cdot e^{-s\delta}} \tag{5}$$

Substituting and rearranging terms, this provides us with the following root locus equation for stability of the system:

$$1 + s\tau + G(dc) \cdot e^{s\delta} = 0 \tag{6}$$

The above equation can be intuitively verified as follows -

- For $\delta \to 0$, as in the case of an ideal analog simulator, the equation reduces to $1 + s\tau + G(dc) = 0$, which is stable and is the expected equation for non-inverting amplifier.
- For $\tau \to \infty$, as in the case of an ideal amplifier, the equation reduces to $s \approx -\frac{1+G(dc)}{\tau}$ for a finitely small δ, thus proving instability proportional to $G(dc)$.

For stability, the above root locus needs to have negative poles. The larger the magnitude of the poles, the closer is the system to ideal simulation. The equation is not easy to solve via direct methods, but can be expanded using Taylor's series:

$$e^{s\delta} = 1 + \frac{s\delta}{1!} + \frac{(s\delta)^2}{2!} + \dots \tag{7}$$

$$\implies 1 + s\tau + G(dc) \cdot e^{s\delta} \equiv 1 + s\tau + G(dc)\Big(1 + \frac{s\delta}{1!} + \frac{(s\delta)^2}{2!} + \dots\Big) \tag{8}$$

In the present case, δ can be made as small as the limit of the simulator, generally 1 fs. Ignoring second order terms, we land into the following relation -

$$s = \frac{G+1}{G\delta - \tau} \tag{9}$$

Hence, the system would be stable when -

$$G\delta - \tau < 0 \implies \delta < \frac{\tau}{G} \tag{10}$$

In other words, the time step needs to be calculated from the characteristics of the circuit for reliable simulation. In application, this manifests as ensuring that the `timescale` is small enough to accommodate the unity gain bandwidth of the amplifier. More complicated feedbacks would rely on the bandwidth for the complete loop including feedback, in case the feedback exercises its own characteristics. It is worth noticing at this point that while specifying smallest time step might look like the best solution, it will have a severe penalty on simulation time.

1) System bandwidth and timescale needs to be controlled and kept in sync for reliable simulation.
2) Assumption that $s\delta \to 0$. Hence $\delta \ll \tau/G$ in practice.
3) The output will have an artificial error due to the finitely small time step of the simulator. δ may need to be further tightened to bring this under controllable limits, at the cost of simulation time.

Simulations were set up to verify that the above claim stands for the case of a non-inverting amplifier. The respective pole was emulated using bilinear transform, as follows,

```
real gain = 1000 ;

real tau = 1/(2*3.14*1e9);           // Pole at 1 GHz
real pole_term, a, b ;               // For pole emulation
real prev_vin=0, prev_vout=0, timestep = 1e-15 ;
```

```

always #1 begin                           // Continuous time evaluation required
   tmp_in = inp - inm ;

   // First order bilinear transform
   pole_term = 2 * tau / timestep;
   a = (1 + pole_term);
   b = (1 - pole_term);

   // Output equation
   tmp_out = (gain*tmp_in + gain*prev_vin - (b*prev_vout))/a;

   // For next iteration
   prev_vout = tmp_out ;
   prev_vin = tmp_in ;
   out = tmp_out ;
end
```

Listing 2: Amplifier modeling with inherent delay (or finite bandwidth)

The corresponding outputs are shown in Fig. 4. As seen, the band-limited amplifier output is stable, as opposed to unstable output observed in the earlier case (Fig. 3).

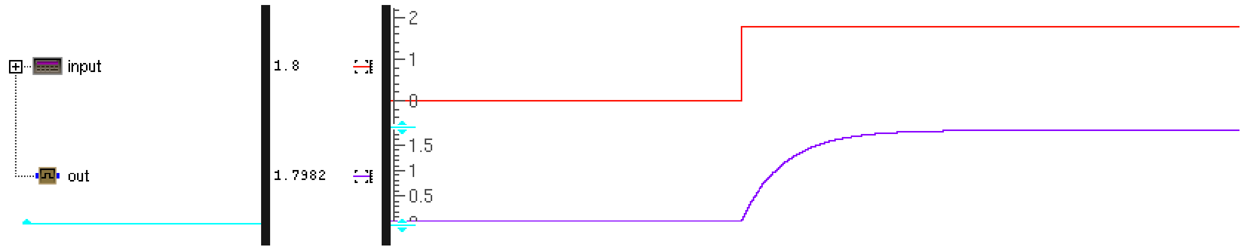

Fig. 4: Stable output obtained from adding bandwidth and time step control. The 1% error in output is owing to finitely small step size.

The case however, acts **only** for simple feedbacks. A slightly more complex feedback, say a resistive feedback for instance, would need some way to describe the resistance. This cannot be easily done using the vanilla `real` data type. Further, the gain itself for say an inverting amplifier configuration is controlled via voltage-current interactions with the feedback impedance and the amplifier, hence mandating a richer data type for the signal.

B. Identifying Unintended Contention/Unknown States

Consider the case of a PLL circuit (Fig. 5). The oscillator (VCO), Divider and the phase-frequency detector (PFD) are blocks with purely logic outputs. The feed-forward path starts typically with a phase-frequency detector followed by a charge pump. The charge pump itself is implemented using defined switches that push respective current values as in the following listing.

```
real output_current = 1m ; // 1 mA output current
always @( up, down ) begin
  up_real = ( up === 1'b1 ) ? output_current : 0 ;
  down_real = ( down === 1'b1 ) ? -1*output_current : 0 ;
  out = up_real + dn_real ;
end
```

Listing 3: Charge pump model

The model basically converts the `up` and `down` signals to output current counterparts, and sums them up to generate the current at `out` port. Note the use of === to emphasize exact match, as opposed to == which would match `X, Z` against logic high as well.

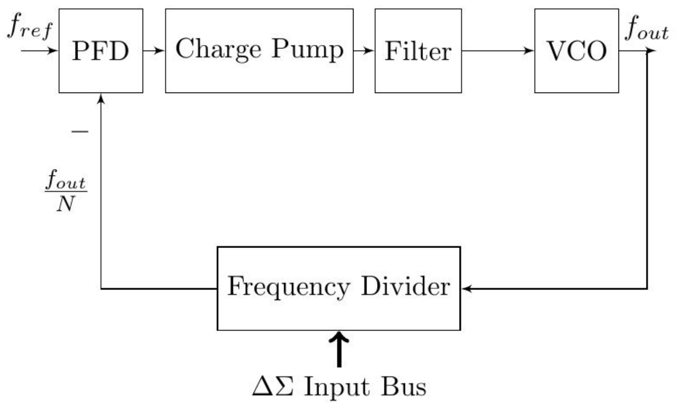

Fig. 5: PLL block diagram.

Assume that at some point in the simulation, the oscillator output goes to `X` due to an unintentional bug in the oscillator. It is expected here that the unknown state propagate through the divider and cause a corruption in the loop, or trigger some alarm so that the bug can be highlighted. However, with the above model, the charge pump actually 'eats up' the unknown state at PFD output and returns a deterministic current at the output. The situation does not improve by allowing `X, Z` to match the logic high states (using == instead of ===).

The above situation justifies the need for a real 'unknown' state, analogous to the digital `X` state. The `real` data type is implemented using floating point operations, and could as well use the `sNan` / `qNaN` to highlight unknown or high impedance states. However, usage of nettypes simplifies the task as well. For instance, a new nettype struct can include a flag for the 'unknown' real signals, as follows:

```
typedef struct{
  real V ; // Voltage value
  real I ; // Current flow value
  logic isX ; // flag for detecting unknown voltages
} VI_with_X ;

// Resolution function to highlight unknowns
function VI_with_X my_resolution_func( input VI_with_X node[] ) ;
  logic result_is_X ;
  int i ;
  i = 0 ;
  for ( i=0 ; i < node.size() ; i++ ) begin
    if ( node[i].isX === 1'b1 )
            result_is_X = 1'b1 ;
    ... // Other stuff for dealing with V and I
  end

  my_resolution_func.V = result_V ; // computed in above foreach
  my_resolution_func.I = result_I ; // computed in above foreach
  my_resolution_func.isX = result_is_X ; // computed in above foreach
endfunction

nettype VI_with_X myNet with my_resolution_func ;
```

Listing 4: A nettype geared towards highlighting unknowns

The connect module between digital and analog boundary need to take care of flagging the `isX` value if the logical input is `X`, and vice-versa. The advantage with this approach, however, is the ease of building the resolution function and connect modules. This is because the unknown state is directly handled via dedicated bit which can be used for decision making. In absence of this measure, the analog-digital interface can instead act as a 'black hole' to any unknown states or unintended configurations arising on the boundary.

C. Ground References and High Impedance

Analog circuits refer all nodes to a ground reference, which is assumed to be at an unmoving '0' potential throughout the operation. The node effectively acts as a stable, zero impedance reference. A high impedance node has the opposite case - it emulates an undriven, floating node which may or may not have a parasitic/remainder voltage. It hence immediately takes the value that is assigned to it.

These are difficult to mimic with a single real signal net. In case of ground references, multiple assignments on will force it to differ from a '0' potential as dictated by respective resolution function/UDR. On the other hand, in case of high impedance, a single real net cannot emulate an undriven node, since the real data type is deterministic in itself.

The need hence is to model impedance for each net, necessitating complex nettypes. A nettype which can handle impedance information, with an impedance based resolution function can aid such a situation, as follows

```
// New user defined type
typedef struct {
  real voltage ;  // The actual signal
  real Zr ;   // Real component of impedance, for resolution function
} my_udt ;

// Resolution function
function my_udt my_res ( input drivers [] ) ;
  int i ;
  real final_v, final_zr ;
  real Zr_max = 1e12 ;  // Consider impedance above this to be effectively floating

  for ( i=0 ; i < drivers.size() ; i ++ )
    if ( Zr = 0 )
      final_v = drivers[i].voltage , final_zr = drivers[i].Zr ;
    else if ( Zr > Zr_max )
      next ; // The driver can be effectively ignored!
    ...
  end // for loop

  my_res.voltage = final_v, my_res.Zr = final_zr ;

endfunction

// Nettype to merge the UDT and UDR for modeling GND and High impedances
nettype my_udt my_nettype with my_res ;
```

Listing 5: A nettype for emulating references and high impedances

The nettype described above effectively takes care of impedance related effects, making it easy to model components such as ideal voltage sources and current sources as well.

III. Structural and Integration Concerns

Integrating RVMs with chip level simulations can be divided into two parts:

1) *Connecting/'stitching' RVMs* into hierarchies as defined by the design/schematics, often called as *netlisting*.
2) *Integrating RVMs into top level/main digital simulation deck.*

Though second order concerns, these can take significant development time owing to related issues. The section discusses these issues and the workarounds found.

Digital flow for mixed-signal designs requires a top-level digital compatible netlist. The netlist should run in a stand-alone digital simulator, and needs to stitch the mixed signal event-driven RVMs as per the design schematics. This can be done manually or via automated tools that translate design schematics to SystemVerilog netlist. While former is easier, the latter is more scalable for large mixed signal designs.

However, one needs to take care of traditional analog design features such as inherited connections from design schematics, which may not be properly translated. Further, the SV netlist itself needs to have more information than the design schematics. Some of these are listed below.

A. *Type Coercion*

This is often the need for automated netlisting flows. The `wire` type is used for digital block level interconnects, both implicit and explicit. However, it fails to coerce/inherit nettype information from leaf level modules, being logical in nature. One way to get over this is to post process the netlist and coerce every port to its concerned nettype as defined by the leaf level modules. The SV 2012 LRM, however, comes with the new generic `interconnect` type. This is essentially a type-less net that can be used to inherit type information.

It should be noted that the coercion fails in case of conflicts. At this point, it might be useful to check the source of these conflicts. Unfortunately, this may not be very easy since an `interconnect` does not go well with useful debug tasks such as `$typename`, making this an ardous process.

A legitimate case of nettype conflicts arises for designs such as a testmux, which taps various signals from the chip for test and validation reasons, irrespective of the signal type (digital/analog). A single output signal hence might be fed from digital or analog domain, making it difficult to make a static coercion. This can be worked around by adding appropriate conversion modules within the testmux model that can handle this situation.

To mimic the behavior of analog signals, such event-driven models would capture electrical parameters like voltage, current, impedance, directionality of a net in a single port as user-defined nettype. While netlist creation with such nettypes can be created, the hierarchical coersion of such nets are not well defined or automated in existing netlist flows. This requires some of the hand-edits to the netlist or changing datatypes for the pins in symbol by explicitly mentioning the UDTs. This issue is not limited to module terminals but also any local wire types within the module.

Primitive models or gate-level modeling such as transmission gates, switch primitives are impossible to model with generic usages nettypes and bi-directionality. Inherited connections of designs could be made as optional terminals and this addition would limit having multiple copies of symbol views and add flexibility. Dropping inherited connections if not connected to any nets in design. Port by name and port by order issues (vlog-net 180 warning) while creating explicit netlist, there was mixed-order in nets getting connected to module terminals.

B. *Mixed Signal Buses*

Mixed signal buses pose another issue to RVM integration. There are limitations which make usage of these mixed signal buses difficult for large designs. For instance, a single bus cannot have nettypes and logic nodes and cannot be packed type. On the other hand, making a bus as unpacked creates issues with top level connections.

IV. Testbench Generation Concerns for RVM Based Modules

A. *Constraint Real Randomization*

Constraint real randomization may seldom be required for modeling, but can prove vitally useful for randomized testbench and pattern generation in verification. There would be cases where a random number would be required in models such as connectivity protocol sources, still it may not stretch to the requirement of random floating-point numbers (subset of real numbers). There would be a requirment of randomization of integers that could be achieved through seed or equivalent functions. It is very extreme case of requirement for a constrained real randomization in modeling space. Some EDA tools require special licenses to invoke constraint real randomization generator, one can avoid it doing a little bit of math. A division of two integers could give us a floating-point value by following steps:

1) Randomize two different integer variables
2) Create suitable constraints for the integer type variable
3) Invoke randomize function and perform the math to assign it to a real variable

Listing 6 provides an example of constraint real randomization as one would ideally like it to be. However, it uses limitedly supported constructs (specifically, `rand real` at Line 5). Listing 7 shows similar randomization using division of randomized integers, while employing universally supported constructs. Note that the probability distribution of the randomized numbers may not be preserved in the latter, and one may need to take appropriate steps to ensure a specific distribution if this is a mission critical requirement.

```
/* Emulating rand real with constraints */
/* Constraint - REAL Values to be 0<x<10 */

class xtndA;
  rand real a;  // Useful but limitedly supported construct
  constraint cst1 { a < 10; }
  constraint cst2 { a > 0; }
endclass

module top;

  xtndA ca = new;
  real x;

  initial begin
    repeat (5) begin
      if(ca.randomize() == 1) begin
        x = ca.a;
        $display("REAL Value = %g \n", x);
      end
            else
        $display ("Randomization failed.\n");
    end
  end

endmodule
```

Listing 6: Constraint Real Randomization Workaround

```
/* Emulating rand real constraints using integer division */
/* Constraint - REAL Values to be 0<x<10 */

class xtndA;
  rand int a;  // Universally supported construct
  constraint cst1 { a < 1230; }
  constraint cst2 { a > 0; }
endclass

module top;

  xtndA ca = new;
  real x;

  initial begin
    repeat (5) begin
      if(ca.randomize() == 1) begin
        x = ca.a/123.0;
        $display("REAL Value = %g \n", x);
      end
      else
        $display ("Randomization failed.\n");
    end
  end

endmodule
```

Listing 7: Constraint Real Randomization using Integer division

V. Results

The usage workarounds suggested in Section II-A were benchmarked for simulation accuracy and speed for the case of non-inverting amplifier circuit. The results obtained were as follows:

TABLE I: Simulation speed and accuracy benchmarks for different approaches

Modeling	Speed	Accuracy
VerilogA/SPICE	1x (Reference)	Reference
RVM with ideal amplifier, no output limiting	0.02x	Not converged
Bandlimited RVM with $\delta \approx 10^{-5} \cdot \tau/G(dc)$	100x	1% error

As observed, the ideal amplifier gotcha actually slowed down the simulation, even for the case where it did come out of delta cycles (that is, where explicit delay is provided in the model). The output of the simulation further depended on the time steps taken by the simulator.

On the other hand, after modifications as discussed in Section II-A, the simulations were faster, though with a manageable 1% loss of accuracy in steady state. The accuracy can be improved further by lowering δ, at the cost of simulation speed.

VI. Conclusion

While identifying RVMs as an important vehicle for enhanced mixed signal verification, the work highlighted perspectives and gotchas in behavioral modeling for common analog topologies. Also presented were the shortcomings and gotchas with conventional modeling approach, especially for capturing subtle analog effects. Workarounds were presented for some of these, especially zero-delay feedbacks and ground references was discussed, using nettype facilities provided by SystemVerilog 2012 standard. While it is difficult to consolidate analog effects into resolution tables, appropriate examples were presented for user defined resolution functions for individual cases. Limitations with the current standard, prominently with real randomization and mixed signal buses were further highlighted.

References

[1] D. Juneja, “On Event Driven Modeling of Continuous Time Systems”, Proc. 2015 28th International Conference on VLSI Design (VLSID 2015), Jan 2015, DOI 10.1109/VLSID.2015.39

[2] J. Tejada, A. Sanchez, “Ultra-low-power power supply system for an IC chip”, United States Patent No. US20120313696 A1, Filed on June 13, 2011.

[3] C. Webber *et al*, “Event driven mixed signal modeling techniques for System-in-Package functional verification”, IEEE Aerospace Conference, 2010, Vol. 1, No. 16, pp. 6-13 March 2010

[4] H. Logemann, “Destabilizing Effects of Small Time Delays on Feedback-Controlled Descriptor Systems”, Elsevier’s Linear Algebra and its Applications, Vol.272, Issues 1-3, Mar. 1998, pp.131-153.

[5] J. W. M. Bergmans, “Effect of Loop Delay on Stability of Discrete-Time PLL”, Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on , Vol.42, No.4, pp.229-231, Apr. 1995.

[6] B. Rezaeian, “Simulation and Verification Methodology of Mixed Signal Automotive ICs”, Lund University, November 2012.

[7] S. Balasubramanian, P. Hardee, “Solutions for Mixed-Signal SoC Verification Using Real Number Models”, Cadence Design Systems, 2013.

[8] “IEEE 1800-2009 Standard: SystemVerilog–Unified Hardware Design, Specification, and Verification Language”, IEEE Standards Association.

[9] “IEEE 1800-2012 Standard: SystemVerilog – Unified Hardware Design, Specification, and Verification Language”, IEEE Standards Association.

[10] M. J. W. Schubert, “An Analog-Node Model for VHDL-Based Simulation of RF Integrated Circuits”, Circuits and Systems I: Regular Papers, IEEE Transactions on , vol.56, no.12, pp.2717-2727, Dec. 2009.

[11] G. Nunn et al, “Using Digital Verification Techniques on Mixed-Signal SoCs with CustomSim and VCS”, Synopsys Inc., March 2011.

[12] N. Khan, Y. Kashai, “From Spec to Verification Closure: a Case Study of Applying UVM-MS for First Pass Success to a Complex Mixed-Signal SoC Design”, Design and Verification Conference 2012.

[13] A. Oppenheim, “Discrete Time Signal Processing, Third Edition”, Pearson Higher Education, Inc. p. 504. ISBN 978-0-13-198842-2.