Enabling True System-Level, Mixed-Signal Emulation

Nimay Shah
Director, Design Verification
Analog Devices
Wilmington, MA, USA
nimay.shah@analog.com

Paul Wright
Principal DV Engineer
Analog Devices
Limerick, Ireland
paul.wright@analog.com

Pranav Dhayagude
Senior DV Engineer
Analog Devices
Wilmington, MA, USA
pranav.dhayagude@analog.com

Raj Mitra
Product Engineering Architect
Cadence Design Systems
Burlington, MA, USA
mitra@cadence.com

Adam Sherer
Account Technical Executive
Cadence Design Systems
Burlington, MA, USA
asherer@cadence.com

***Abstract*-Emulation is ubiquitous for verifying and validating complex silicon systems, comprising of a full software stack driving highly intricate hardware. However, as some of these silicon systems move towards the intelligent edge, the underlying hardware becomes exceedingly mixed signal with the integration of sensors, real-world interfaces, high-speed data convertors, buck/boost regulators, etc. Traditional emulation techniques only support synthesizable digital logic. As a result, the scope of what can be verified or validated, and to what extent is limited. This means that software driven chip configuration that goes all the way down to a primitive hardware elements or complex calibration loops and low power techniques involving the full software stack, cannot be fully verified prior to tapeout. This is an absolute necessity in today's complex systems at the cutting-edge manufacturing technologies owing to the cost of unplanned tape-outs and the pressure of delivering first-pass sampleable silicon to customers. The novel techniques presented in this paper focus on removing this limitation and enabling analog/mixed-signal behavioral modeling methods, thereby enabling "true" system-level, mixed-signal emulation.**

I. Introduction

The advent of SoCs that run basic software stacks on processor cores really challenged validation techniques that relied on silicon samples for developing and testing software. Increasing tapeout costs, and pressures of time-to-market implied that the software had to be co-developed with hardware and validated pre-silicon. Doing so with traditional dynamic simulation-based tools and techniques is highly impractical, and in majority of the cases, not feasible, given prohibitive runtimes. Hardware emulation emerged as the answer to this challenge and became an important part of the Verification 2.0 wave, delivering speed-up of several orders of magnitude over traditional simulation. Since then, emulation has gone through several advancements, such as hybrid emulation or simulation acceleration, advent of speed bridges for emulating with various real-life interfaces and protocols, etc.

Fast forward to today, silicon systems (SoCs, Chiplets, 2.5/3D ICs, SiPs, etc.) are now even more complicated and programmable than ever, with multiple heterogenous processing cores running full-fledged operating systems, DSP, and AI/ML algorithms. To make matters more interesting, a majority of these also have a considerable mixed-signal component to them as data analytics and processing gets increasingly closer to the intelligent edge. This is true of all end-markets, automotive, industrial, communications, healthcare, to name a few. Since hardware emulation can only support synthesizable digital logic, a lot of features of these complex systems cannot be fully validated pre-silicon. Further, running a true AMS (or Analog Mixed Signal Simulation) where the digital simulator simulates the digital content, and an analog simulator simulates the analog content (the two simulators synchronized in time using vendor specific technologies) limits the number of tests that can be run. This is primarily due to the runtime limitations imposed by the analog simulator. Other approaches, such as, standard FPGA-based emulation, hybrid simulation & emulation, or representing floating point real datatypes via equivalent fixed-point bit models [1] have not proven to be sufficient or easy to use. These either result in a major hit to emulation throughput, emulation capacity or require a major change to well-established analog modeling techniques, such as SystemVerilog Real Number Modeling (RNM) and the associated Digital-Mixed Signal (DMS) verification methodology [2]. The novel, innovative techniques presented in this paper aim to address the challenge of enabling true system-level emulation, including synthesis of mixed-signal models without the limitations of prior works in this area.

The rest of the paper is organized as follows: the next section describes some related work involving fixed-point modeling and explains the downside of such an approach. We then describe the high-level methodology and takes a deeper dive into supported and unsupported constructs of the SV LRM within the enhanced compiler that can handle RNMs in section III. It will also include code examples and workarounds as needed so that the simulation and emulation models yield equivalent functionality and accuracy. Section IV describes the application of these techniques on a basic PLL, as well as a real-life SoC, a 4T4R transceiver running full-blown Ubuntu OS on an application-class, multi-core processor environment. We will also give examples of some RNMs that comprise of the signal chain and auxiliary analog circuits that constitute this DUT. Next, we will take a deeper dive into results obtained, including waveform comparisons between simulation and emulation to illustrate the accuracy. The runtime for each solver will be highlighted to show the performance gains achieved. We finally conclude in Section V, presenting a summary, our conclusions and future work.

II. Related Work

As cited in the introduction, fixed-point bit models have been shown to enable mixed-signal emulation but have not proven to be sufficient or easy to use. An example of this modelling approach was applied to an Image Sensor [3]. The Image Sensor shown in the center of Fig. 1 was originally model using RNM and executed in simulation. Those models could not run directly on standard FPGA-based and purpose-built emulation platforms due to their inability to directly process real numbers. Fixed-point bit models can replace the real number models but there are multiple factors to consider. One of the most significant is the accuracy versus emulation model size tradeoff. The greater the number of bits, the more accurate the model but that also creates a larger emulation footprint. For FPGA-based emulators, if the model grows too large, routing and timing can impact the ability to emulate the model. For purpose-built emulation platforms, the size of the emulation model can lead to longer compile times affecting the debug turn-around time when used in a production verification flow.

The fixed-point bit models implemented for the Image Sensor in Fig. 1 enabled the emulation of 5.6s of real system function in 2.1 hrs. of emulation time. Of course, other commonplace techniques of transforming a simulation environment to emulation, like replacing VIP drivers/monitors with synthesizable BFMs, replacing memories with their synthesizable equivalents, etc. must be deployed in addition to synthesizable, fixed-point RNMs to realize this gain. However, the key message is that RNM synthesizability is the cornerstone to successfully realize this use-case in emulation. This demonstrates the potential value of system-level, mixed-signal emulation even though it required months of work to recode the model for emulation and a large emulation model size. While the speed is attractive, the engineering cost to create the fixed-point model, to connect the verification environment to that model, to maintain synchronization with the RNM simulation models, and validate the overall model accuracy often preclude the development of these models. Any work that can directly make the original RNMs synthesizable without having to recode them, would thus be of immense value.

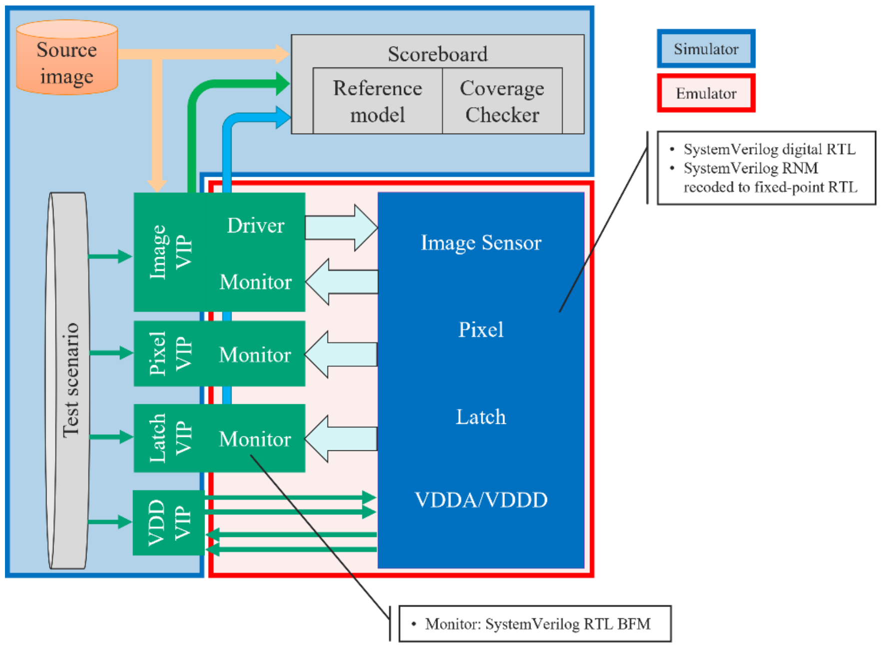


Figure 1. Image Sensor Emulation use-case deploying fixed-point models in simulation-acceleration mode

III. Our Approach – Solutions, Overall Methodology And Application

Our approach to an end-to-end mixed-signal emulation solution is via innovative enhancements to RNM coding guidelines combined with compiler enhancements and the subsequent proof of those guidelines with a real-world example. These guidelines were implemented on an emulation hardware platform and the associated front-end compilers, to demonstrate the efficacy of the RNMs via execution on emulation hardware. These compilers now support critical RNM features including SV-2009 real datatype, SV-2012 nettypes and resolution functions (includes UDNs such as *EEnet*), delays, floating-point operations, basic math operations, etc. The compiler output is still mapped to compute elements in hardware which limits the use of RNM behavior language features like dynamic arrays and loop limits. However, these are easy to work around without significant issues. As a result of this approach, the overall emulation methodology that the end user is familiar with is relatively unchanged and only requires some additional options and configuration. Below are the examples of some classes of supported constructs and known limitations. This is not an exhaustive list.

A. Datatype support

All standard 'real' datatypes, such as, *real, shortreal, realtime* are supported, along with their unpacked arrays and structs. SV-2012 nettypes are also fully supported, including resolution functions for multi-driver scenarios. However, the resolution functions must be synthesizable. This means that it cannot use any dynamic arrays, which are quite common. These can easily be replaced with fixed-sized arrays. Vendor-specific packages of commonly used nettypes are also supported.

B. Delay support

Pound (#) delays are typically not synthesizable. However due to the ubiquitous usage of such delays in RNM, support for these has been enabled. Statements such as the ones below are fully supported:

- always process with time delay and no event.
  Example:
  ```
  always #10 a = b;
  ```
- always process with time delay and explicit event.
  Example:
  ```
  always @(a or b) begin
    #10;
    y = a + b;
  end
  ```
- Continuous assignment with a single delay.
  Example:
  ```
  assign #10 y = a + b;
  ```

Limitation: Rise/Fall/Off delays specified with Verilog gate primitives are not supported. These need to be transformed into one of the three constructs above.

For example: *not #(RiseDelay, FallDelay) (O, o_reg) ;* is not supported. However, this can be easily transformed into a supported construct as follows:

```
always @(posedge o_reg) begin
  #RiseDelay;
  o_temp = ~o_reg
end
always @(negedge o_reg) begin
  #FallDelay;
  o_temp = ~o_reg;
end
assign O = o_temp;
```

C. Floating-point operations, Math functions and Conversion functions

Operators with real operands in Table 11-1 in the IEEE Std 1800-2017 [4] are supported in DUT except for the inside operator. Majority of the real math functions in Table 20-4 in the IEEE Std 1800-2017 [4] are supported with some minor caveats. Section 20.5 in the IEEE Std 1800-2017 [4] describes system functions for converting values to and from real number values - *$rtoi, $itor, $realtobits, $bitstoreal, $shortrealtobits, $bitstoshortreal* are all supported.

D. DMS Connectivity

All industry standard simulators over the years have built various features into their compilers and elaborators to help with DMS connectivity mismatches between the high-connection and low-connection ends of any segment in the design hierarchy. These features are typically not part of the SystemVerilog LRM and are commonly referred to as (a) datatype coercion and (b) automatic Connect Module (CM) or Interface-Element (IE) insertion. The front-end compiler for the emulation platform has been enhanced to incorporate datatype coercion to a great degree. However, it still is not a 1:1 match with the simulator's capabilities. CM insertion is not supported. To proactively identify and fix any connection mismatches that would cause issues for the emulation build, we developed a VPI and a post-processing Python script to trace every connection from the top-level IO of the design hierarchy to the IO on leaf instances (instances with no further underlying hierarchy). The VPI dumps the high connections and low connections starting from the top-level IO to IO at each intermediate hierarchy all the way to the IO on the leaf cells. The Python script reads all these high connections and low connections and assembles the signal paths, identifying the signal's type (logic, interconnect, coerced wreal, real variable etc.) for every signal in the hierarchy. The signals are then written into an Excel spreadsheet, identifying if there are segments in the hierarchy of a given signal path that might cause a problem during building the emulation object. Logic, wreal and other custom nettypes endpoints are identified vividly.

E. Other Limitations

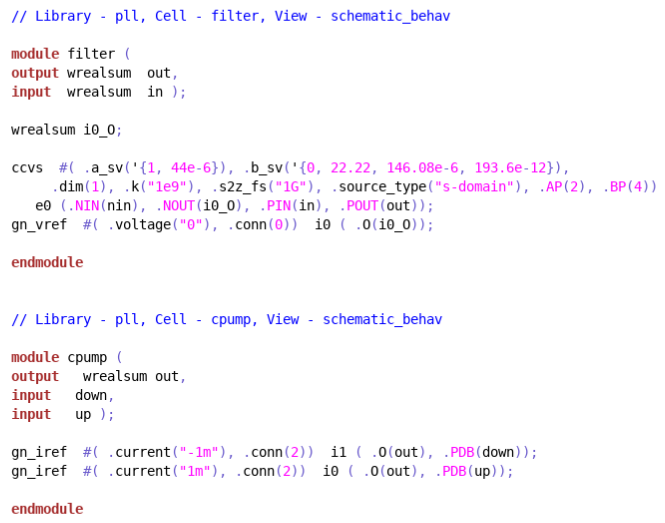
All standard limitations of emulation systems related to synthesizability apply unless otherwise stated (example: # delays are typically non-synthesizable but are supported with this novel methodology as clarified above). In addition, following constructs that might be important from RNM & DMS point of view are not supported: (a) Verilog gate primitives (b) Dynamic arrays (c) SV constructs operating on real data types: concurrent assertions, immediate assertions, coverage (d) Force and release operations of real datatypes.

IV. Test Vehicles and Results

A. Charge Pump Phase Locked Loop (CP-PLL)

As a basic proof-of-concept, we used a basic charge-pump PLL. This is a predominantly analog circuit – which is perfect to prove out the RNM synthesizability aspects of the enhanced compiler. It also helped us with validating the modeling guidelines, quantifying the changes required and identifying any necessary enhancements to our tools and methods.

Fig. 2, below is a code snippet from the PLL netlist showing the top-level modules for the loop filter ($filter$) and the charge pump ($cpump$). The loop filter is implemented using a $H(s)$ implementation of the Current-Controlled Voltage-Source (CCVS) from the reusable ADI schematic model library that can take in an array of poles (a_sv) and zeros (b_sv). These arrays are typically dynamic in nature. However, due to emulation limitations, we made these static by declaring additional model parameters, AP and BP. The filter itself is implemented as a bilinear transformation with some complex floating-point operations. The snippet also shows the use of the $wrealsum$ datatype, which is a commonly used, simple UDN.



```
// Library - pll, Cell - filter, View - schematic_behav

module filter (
output wrealsum  out,
input  wrealsum  in );

wrealsum i0_O;

ccvs  #( .a_sv('{1, 44e-6}), .b_sv('{0, 22.22, 146.08e-6, 193.6e-12}),
     .dim(1), .k("1e9"), .s2z_fs("1G"), .source_type("s-domain"), .AP(2), .BP(4))
   e0 (.NIN(nin), .NOUT(i0_O), .PIN(in), .POUT(out));
gn_vref  #( .voltage("0"), .conn(0))  i0 ( .O(i0_O));

endmodule


// Library - pll, Cell - cpump, View - schematic_behav

module cpump (
output   wrealsum out,
input   down,
input   up );

gn_iref  #( .current("-1m"), .conn(2))  i1 ( .O(out), .PDB(down));
gn_iref  #( .current("1m"), .conn(2))  i0 ( .O(out), .PDB(up));

endmodule
```

Figure 2. Code snippet from the PLL showing top-level modules for the loop filter (filter) and charge pump (cpump)

Fig. 3, below shows the top-level of the PLL and how the filter and charge pump (cpump) modules get connected with the output of the charge pump going to the input of the filter. The main things to notice here is how this connection happens via a $wire$ datatype. Since the compiler supports coercion, this automatically get coerced to the appropriate datatype ($wrealsum$ in this case) at emulation build time. There is no need to explicitly declare this top-level signal as wrealsum. This might not appear to be a major issue for this smaller example but would be a huge showstopper for any real-life applications. The snippet also shows the use of the on-board emulator clock.

```
module pll (  );

wire vss, i4_out, vc;  /*don't need to be declared wrealsum as coercion is supported*/
bit ckin ;
wire logic ckdiv ;
wire logic ckout ;
wire up, i0_OB, down ;

pfd i1 ( .down(down), .up(up), .ckdiv(ckdiv), .ckin(ckin));
vco i2 ( .out(ckout), .vc(vc));
filter i3 ( .out(vc), .in(i4_out));
cpump i4 ( .out(i4_out), .down(down), .up(up));
divby32 i5 ( .ckout(ckdiv), .ckin(ckout));
lclock i0 ( .ckin(ckin), .i0_OB(i0_OB));
gn_vref  #( .voltage("0"), .conn(0))  i7 ( .O(vss));

`ifdef IXCOM_COMPILE
IXCclkgen #(500) ckg(mclk);  // 1ns CAKE master clock
`endif

endmodule
```

Figure 3. Code snippet showing top-level PLL netlist

Fig. 4, below shows the comparison of the control voltage, Vc, the input to the Voltage Controlled Oscillator (VCO), between the simulation and emulation cases. Assuming simulation as the reference, the error is less than 0.0423%.

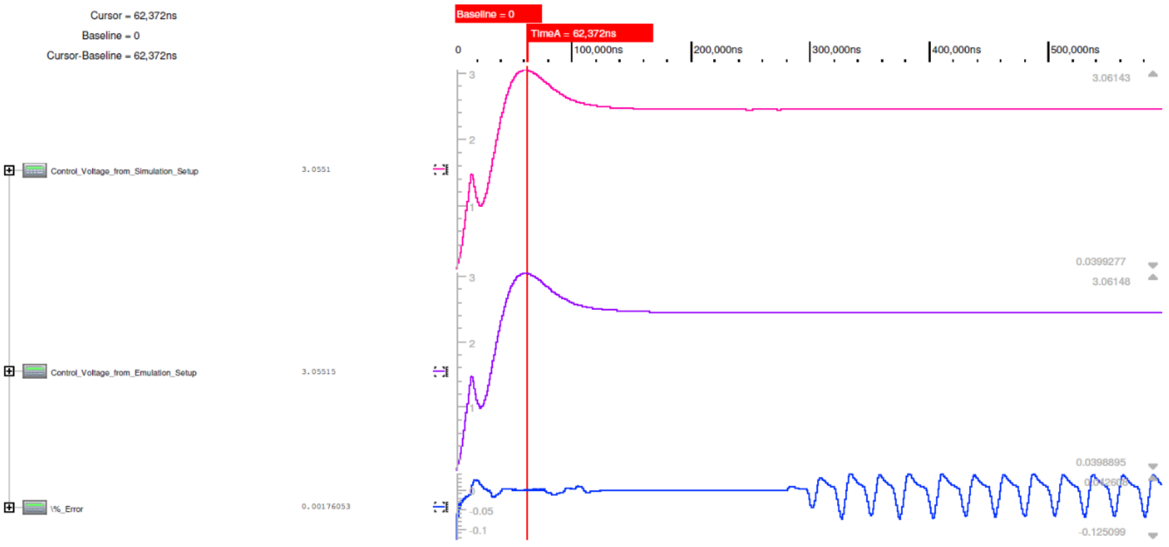


Figure 4. Comparison of CP-PLL VCO Control Voltage between Simulation and Emulation

The PLL pipe-cleaning exercise helps us test several pieces of critical, required functionality. It uses several nettypes, current modeling, and a bilinear transform implementation as part of the loop filter to implement $H(s)$ function which has several complex floating-point operations. It also uses delays, some of which had to be recoded for emulation compatibility. Some of the bilinear transform functions also had to be recoded to convert dynamic arrays into fixed-size static arrays.

B. 4T4R Transceiver SoC

Next, we turned our attention to a real-life SoC – a 4T4R ORAN transceiver running a complex Ubuntu OS over complex, highly mixed-signal hardware. Current approach with emulation of this system was to black box the entire analog portion. Selective AMS blocks would be replaced with non-functional, overly simplistic models of just their configuration registers and the use-cases being run on emulation would simply check for the resulting register configuration bits and not for the actual "real" signal output. As for complex calibration loops, there is not really an effective way to verifying these end-to-end as they rely on feedback from the analog and an appropriate response from the digital or all the way from the software/firmware itself. The baseline, digital-only emulation setup that we use as a starting point is in "simulation acceleration" mode i.e., a portion of the UVM testbench (non-synthesizable) runs on the standard CPU-based simulation engine and the entire DUT, as well as synthesizable portions of the testbench run on the emulation platform. A lot of optimizations have been incorporated into this setup over a few generations of the product, ensuring maximum possible speedup and efficiency with the emulation platform. These include well established techniques such as replacing pure simulation-based Verification IPs (VIPs) with their emulation-friendly equivalents, Accelerated VIPs (AVIPs) [5], optimizing the interface between the simulator and emulator using platform-specific proprietary techniques, optimizing the DUT build, et cetera.

The first RNM that we white-boxed was the bias generator. All the analog blocks rely on correct configuration and functioning of this block for their proper operation. The bias generator provides 50μA bias current to the main oscillator on the device. Its core section includes a bandgap reference. From this reference, the various bias currents are generated. These can be trimmed, some currents are constant, others are PTAT (Proportional to Absolute Temperature). The model, however, does not consider temperature dependencies. The trim of the core bandgap reference has not been implemented in the new or old models. The model implements a set voltage output which can be brought to the off state by powering down the block. The block contains a comparator which can be used to trim some of the bias currents relative to one another. The trimming routine is controlled by a state machine activated in response to a register write.

Several modeling techniques are used to produce the overall bias circuit behavioral model. These are described next:

- Category 1: Model Library - There were several standard components used from the Analog Devices simulation library. These components have views which can operate across multiple simulators. These include voltage sources, voltage-controlled voltage sources, OR gates, AND gates, comparators etc. The necessary subset of this library has been updated to ensure that they are emulation compliant.
- Category 2: Extracted Look Up Table (LUT) - The bias currents are generated in direct proportion to an input voltage from the bandgap reference. The ratio between the voltages and currents can be trimmed. Such a circuit can be modelled using an extracted look up table. An internally developed Analog Devices application was used to extract the LUT and generate the model. The output model types are SV compatible. This application had been used on previous Analog Devices projects; the SV code generated by it required no modification for use with emulation.
- Category 3: Handwritten System Verilog - The use of handwritten models was restricted only to those cases where absolutely needed – such as those where $`ifdef \ldots `else \ldots `endif$ constructs are required to differentiate between emulation and simulation behavior. Cases where this differentiation is required are (a) simpler behavior for emulation vs. simulation to avoid complexities (b) manually managing signal disciplines and (c) inserting hand-instantiated connected modules. This is an exceedingly small subset of the overall model of the bias generator.

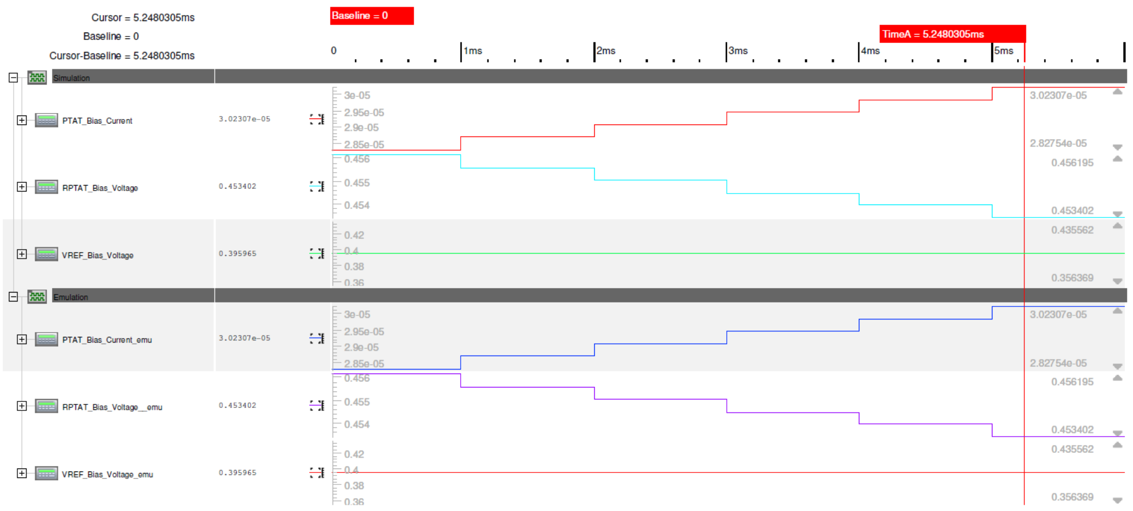


Figure 5. Comparison of some key signals for the transceiver bias generator between simulation and emulation

The bias generator model is automatically netlisted from the analog design environment where the leaf-level models sit next to their transistor-level equivalent circuit. This updated “emulation-friendly” model was then validated by running the test-bench with the transistor-level circuit, RNM model in simulation and RNM model in emulation for various use cases – power-down, trim, etc. In the emulation case, the test-bench runs on the CPU-based simulator and the model runs on the emulator. This is also commonly known as “simulation acceleration” mode. Some of the results are shown in Fig. 5 above.

Once all of these were passing, the bias generator was integrated into the transceiver system-level DUT for a system-level emulation run. One of the interesting cases is the bias generator self-trim routine that is triggered via register writes from the processor core on the SoC. This shows all the key components interacting with each other; software, digital and analog. Fig. 6 below shows the results from the simulation run. The results from the emulation run are shown in Fig. 7. The trim routine executes successfully on both. There are differences in the output of the constraint randomization solver that results in the waveforms not being identical. However, this additional capability of incorporating RNMs on to the emulation platform comes at a cost. The maximum frequency achieved on the emulation platform using full-fledged bias generator RNM drops from 959kHz (with an all-digital bias generator) to 635kHz (33% degradation). For our case, this tradeoff of being able to synthesize RNMs in floating-point fidelity by sacrificing some emulation speedup, is well within bounds and worth it considering all the additional system-level mixed-signal use-cases that can be validated, pre-silicon. The emulation runtime is 45s vs. the pure simulation runtime of 14400s, thereby yielding a 320x speedup. We expect the speed-up using the emulation platform to be significantly higher as we move to more complicated cases, most of which would be runtime prohibitive on a pure DMS simulation setup. We will also continue working on improving the maximum speed achieved on the emulation platform for mixed-signal use-cases.

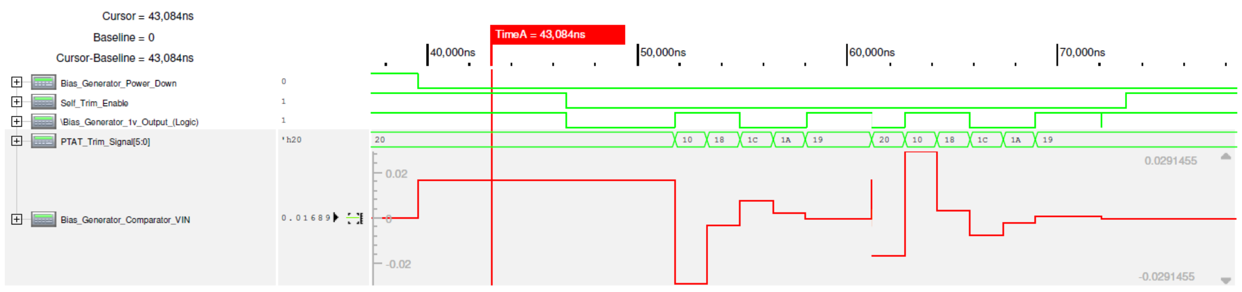


Figure 6. Simulation results for the bias generator self-trim routine

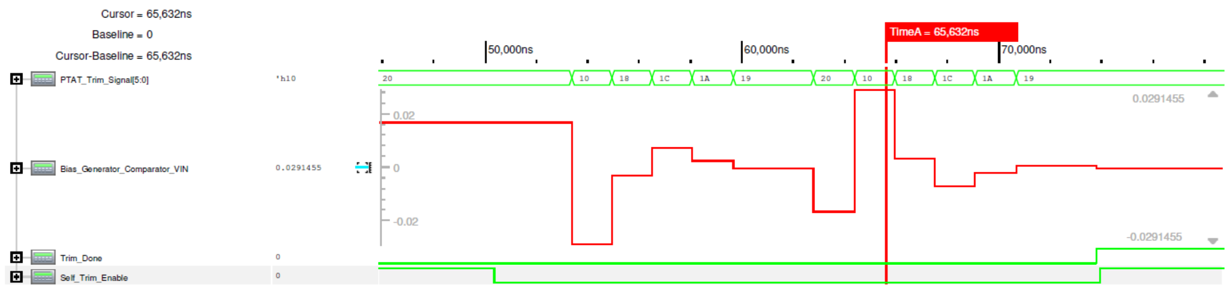


Figure 7. Emulation results for the bias generator self-trim routine

V. Conclusion And Future Work

Being able to synthesize RNMs to floating-point fidelity on standard emulation platforms, opens immense potential for comprehensive pre-silicon validation of complex, mixed-signal silicon systems with the entire software stack at the edge.

As a basic proof-of-concept, we used a basic charge-pump PLL. This is a predominantly analog circuit, which is perfect to prove out the RNM synthesizability aspects of the enhanced compiler. It also helped us validate the modeling guidelines, quantifying the changes required and identifying any necessary enhancements to our tools and methods.

With the real-life transceiver, we can attest that the tools, techniques, and methodology outlined in this work are ready for production use with known limitations that have proven workarounds. We can now reliably and truly validate complex mixed-signal systems, pre-silicon.

As part of future work, Analog Devices will continue to enhance its simulation library to be fully emulation compliant. As a result of this, any models created using this library of building blocks will be emulation-ready without any overhead.

From Cadence's point of view, as tools and emulation hardware platforms advance there will be more support for additional RNM and DMS constructs along with underlying hardware enhancements as needed. Furthermore, the single RNM code base for simulation and emulation will enable engineers to devote more time to full-system verification driving additional innovation resulting in higher quality products.

ACKNOWLEDGMENT

The authors would like to acknowledge the contributions of various people on teams at Analog Devices, Inc. and Cadence Design Systems, Inc. that made this possible.

REFERENCES

[1] F. A. Nothaft et. al., "Pragma-based floating-to-fixed point conversion for the emulation of analog behavioral models," in *2014 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, San Jose, CA, USA, 2014.

[2] S. Herbst, G. Rutsch, W. Ecker and M. Horowitz, "An Open-Source Framework for FPGA Emulation of Analog/Mixed-Signal Integrated Circuit Designs," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, July 2022.

[3] R. Kolker, G. Best and P. Len Orlando III, "Mixed-Signal Emulation Digital Twin for Defense Applications," in *CadenceLive*, Burlington, MA, USA, Sept. 2022.

[4] "IEEE Standard for SystemVerilog--Unified Hardware Design, Specification, and Verification Language," in *IEEE Std 1800-2017 (Revision of IEEE Std 1800-2012)*, 22 Feb. 2018, pp. 1-1315.

[5] "Accelerated VIP," Cadence Design Systems, Inc., [Online]. Available: https://www.cadence.com/en_US/home/tools/system-design-and-verification/verification-ip/accelerated-vip.html.