Novel Mixed Signal Verification Methodology using complex UDNs

Rakesh Dama, Ravi Reddy, Andy Vitek
Roche Sequencing Solutions
2841 Scott Blvd
Santa Clara - 95050

***Abstract*- Mixed signal verification using discrete modeling of analog circuits is highly efficient for designs where simulation speed dictates functional verification efficacy. System Verilog provides User Defined Nettype (UDN) for modeling analog nets. Users write customized resolution functions for different types of nets depending on the net's behavior – such as sum, average, max, min and other types of resolutions. But there are instances where the same net has to be resolved differently at run time. Also, instead of writing multiple resolution functions as the need arises, modeling efficiency can be improved by using standardized resolution functions. To achieve this, we propose a methodology using structure type UDN, i.e. complex UDN, along with a set of standardized resolution functions. Towards this, a resolution function that resolves the output of drivers according to different scenarios has been developed. In addition, the approach enables modeling difficult-to-model circuits such as a two-way switch.**

I. Introduction

Thorough verification of mixed signal circuits has become indispensable owing to the increased use of analog circuits in SoCs. While verification approaches using co-simulation and continuous modeling languages such as Verilog-AMS accurately model the analog circuits, they require significant effort and simulation clock cycles as seen in Fig. 1. When simulation performance and portability across multiple levels of SoC are required, discrete modeling of the mixed signal circuits and verification methodology based on discrete real number modeling (RNM) is the best approach. Currently, System Verilog provides features such as wreal ports and User Defined Nettypes (UDN) to model analog values and nets. Wreal is used to represent single driver nets, while UDN is used to represent multi-driver nets. UDN requires users to write a resolution function that reduces the driven values from multiple drivers to a single final value. A resolution function could do summation, averaging, pick a minimum or maximum value, or any other custom resolution. System Verilog (2012) also provides a struct data-type that can be used with UDNs and enables passing a set of values between ports. All the ports connected to a UDN must be of the 'same type', where 'same type' means the same data-type and the same resolution function. This would seem to limit the possibility of resolving a net value to only one functional type (i.e. only add, only average etc.). Available standard resolution functions from EDA vendor (Cadence) resolve to a single functional type or mimic an electric equivalent port (EEnet). While these are sufficient for most common scenarios, there are instances where the same net behaves differently depending on the circuit topology at run time. This paper proposes the use of struct type UDN (Complex UDN) to accomplish resolution on the fly. In addition, modeling efficiency can be improved by using a defined set of UDNs and a list of port types, across the design.

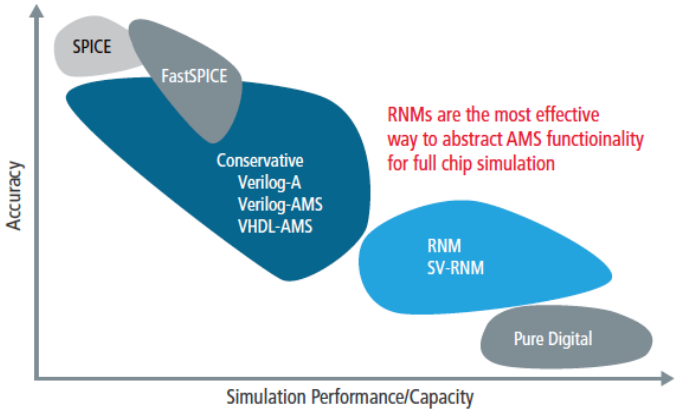

Figure 1. Model accuracy and performance of mixed-signal simulation [1].

II. Analog Ports and UDNs to Model Analog Nets

Examining an analog circuit, one could find that analog ports generally belong to one of the following types: voltage source (V_OUT), current source (I_OUT), pass-through port (V_PT, I_PT) connecting V_OUT or I_OUT to downstream receivers (such as a switch), storage port (V_CAP) such as capacitive nodes in parallel and receiver ports (V_IN, I_IN). Among these ports, several valid combinations of connectivity are possible. The resulting value of voltage or current on a net connecting different ports depends on the type of the ports, and other factors such as impedance on the drivers and receivers. To model the behavior of the analog nets, UDN feature in SV could be used. Assuming ideal sources, the resolution for a net connecting V_OUT to V_IN would be a single value if all the V_OUTs have the same value, else an unknown value. The resolution for a net connecting I_OUT to I_IN would be the algebraic sum of the I_OUT currents. If a net connects V_CAPs, then the resolution would be a weighted average of the voltage on the capacitors. But SV requires that if a net is declared as a particular UDN type, then all the ports connected to it must be of the same UDN type (i.e. same data type and resolution function). This limits the modeling of other combinations of connectivity between different port types if separate resolution functions are written for different port combinations.

While Cadence EEnet provides an electrical equivalent of a net [2] which can be used to connect voltage and current sources using complex UDN (`struct {real V, I, R} EEstruct`), it still isn't possible to model nets connected to storage nodes and two-way switches using the EEnet [3]. To encompass the missing features of the EEnet, our approach defines a 'generic UDN' (using complex UDN). Our methodology uses this generic UDN and other scalar UDNs (simple UDNs), as appropriate for the modeling scenario at hand. The simple UDNs include 'wreal4state' and 'wrealsum' nettypes. The generic UDN is a complex UDN, which includes the port type as part of the port information along with voltage, current, resistance and capacitance information of the port. The generic UDN resolution function takes the information about port type of all ports connected and resolves the net accordingly. In brief, the generic UDN computes the total input resistance/capacitance on a net. Using this information, current entering into each receiver port and total current/voltage on the net are computed. In addition, new port types can be defined and used in the resolution function without the need to redeclare net type of the ports.

III. Generic User Defined Nettype

The proposed generic UDN uses a complex UDN with a struct data type shown in Fig. 2. The struct data type includes a variable to store the type of the analog port from a predefined set. The port types are chosen to be majorly of three classifications – active drivers, pass through ports and secondary drivers. Active drivers are the strongest drivers such as a voltage source and current source. Pass through ports are ports of modules that just connect two branches of active sources, e.g. switch. Secondary drivers are weaker drivers, which drive only if no active sources are driving the net, e.g. capacitor. All active sources are assumed to be ideal sources and non-ideal behavior shall be implemented inside a module's behavioral model. This means connection between a voltage source (V_OUT) and current source (I_OUT) is illegal in this methodology. This assumption is made with an understanding that scenarios where voltage source and current source driving the same net are rare (or not needed).

```
1
2  typedef struct {
3
4     // { V_OUT, I_OUT, V_PASSTHRU, I_PASSTHRU,  V_CAP, UNKNOWN}
5     PORT_TYPE   port_type;
6
7     // Vars to assign driven values.
8     real        v_value ;
9     real        i_value ;
10    real        g_value ;
11    real        c_value ;
12
13    // Vars to store the resolved value of all active sources.
14    real        v_active;
15    real        i_active;
16    real        g_active;
17    real        c_active;
18
19 }  GENERIC_PORT;
20
```

Figure 2. Generic UDN data type.

The resolution function of the generic UDN uses the port type to determine the strongest driver. Active drivers are stronger than secondary drivers and secondary drivers are stronger than pass through ports. Once the strongest port type is determined, the resolved value of the net is calculated accordingly. For example, if the port type of the net is V_OUT then a single voltage value is resolved. If port type is determined to be V_CAP, then a weighted average is calculated. Further, when a given net is resolved to be of a stronger type, but the stronger type is not driving, i.e. `wrealZState (`wZ), then the calculation is done according to the next weaker port type on the net. This enables connecting a variety of port types to the same net and resolving the behavior of the net on the fly. As an added feature, checks are done inside the resolution function for the legality of connections. For example, it is illegal to connect V_OUT port types and I_OUT port types to a net. However, we chose to enable such checks only at time zero for performance reasons.

IV. Methodology Using The Generic UDN

To enable user to easily choose port types and configure ports, a Graphical User Interface (GUI) was developed. The GUI enables user to select the port type and record other port parameters including resistance and capacitance values. The GUI reads-in an SV file containing the module declaration and port list. Each port of the module can be individually configured. After configuring the ports as needed, the program writes out code necessary for UDN usage and additional scalar variables for easier manual debug. Most use cases require simple UDNs, while a few instances require the use of complex UDNs. Using simple UDNs when possible improves simulation efficiency. This is due to the fact that the generic UDN resolution function requires more computation, comparatively. Fig. 3 shows the GUI to configure ports of a module.

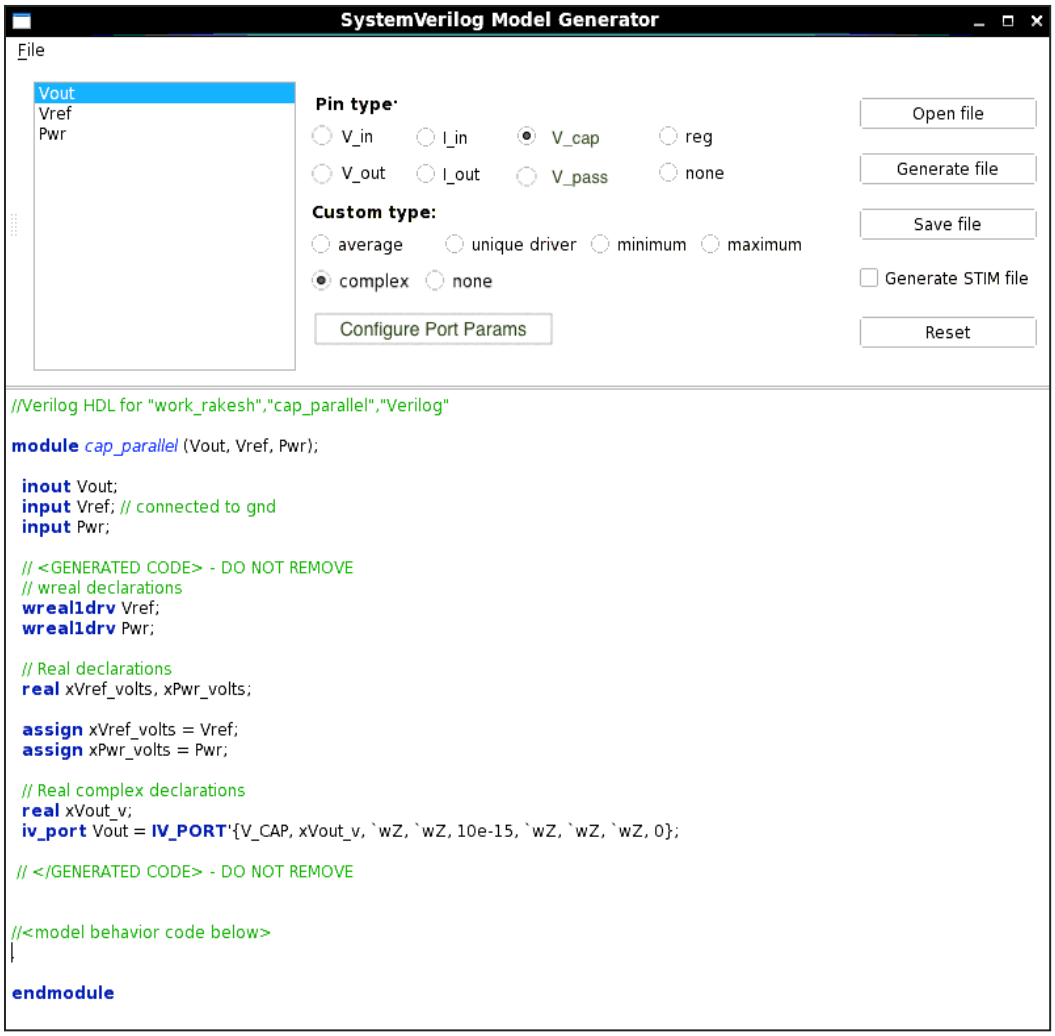

Figure 3. GUI to configure ports in a module.

V. MODELING AN SAR ADC

A 4-bit SAR ADC has been modeled using this methodology. Notably, the charge redistribution on the capacitors has been modeled. Inside the capacitor bank (CAP_BANK in Fig. 5), the capacitor ports connecting to common node (Vcommon) are selected as V_CAP. Port type selection for ports of different modules in the CAP_BANK is shown in Fig. 4. The switches connecting to the bottom plates of the capacitor and common node are modeled as one-way, ideal switches that transfer input voltage (from voltage sources) to the output. So, the output port of the switch is chosen as V_OUT. As can be seen in Fig. 4, the net 'Vcommon' has different types of ports connected to it, i.e. V_CAP, V_OUT and V_IN (of the comparator). Accordingly, the resolved value of the net is equal to Vref when SW1 is closed. When SW1 is open, the resolved value is the weighted average of the voltage on the parallel capacitors C1, C2, C3, C4 and Cdum with C4:C3:C2:C1:Cdum = 8:4:2:1:1. The ports of the remaining modules in the SAR ADC (shown in Fig. 5) are modeled as digital ports or as simple UDNs.

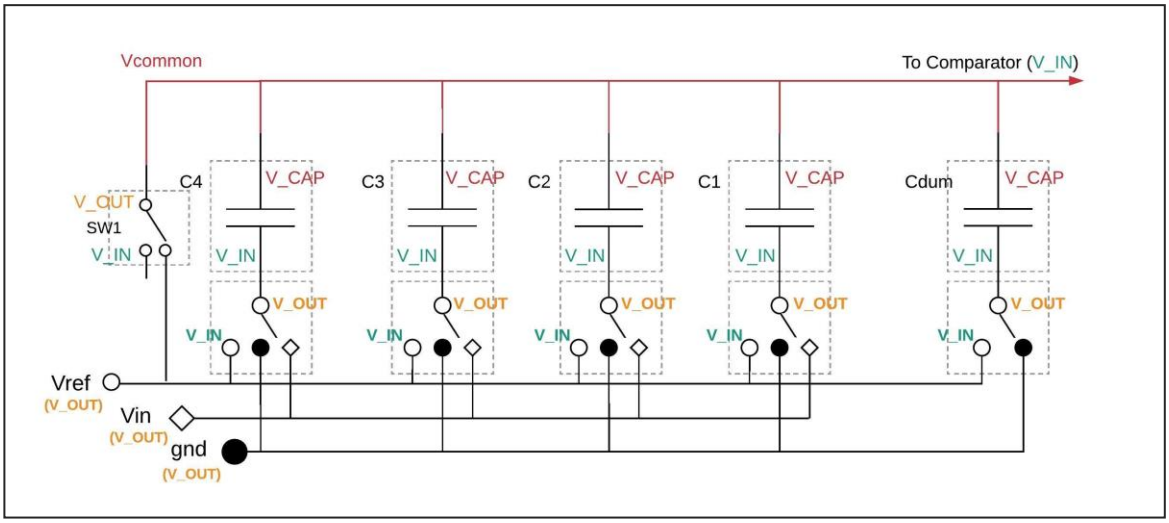

Figure 4. Port types in the capacitor bank to model charge redistribution.

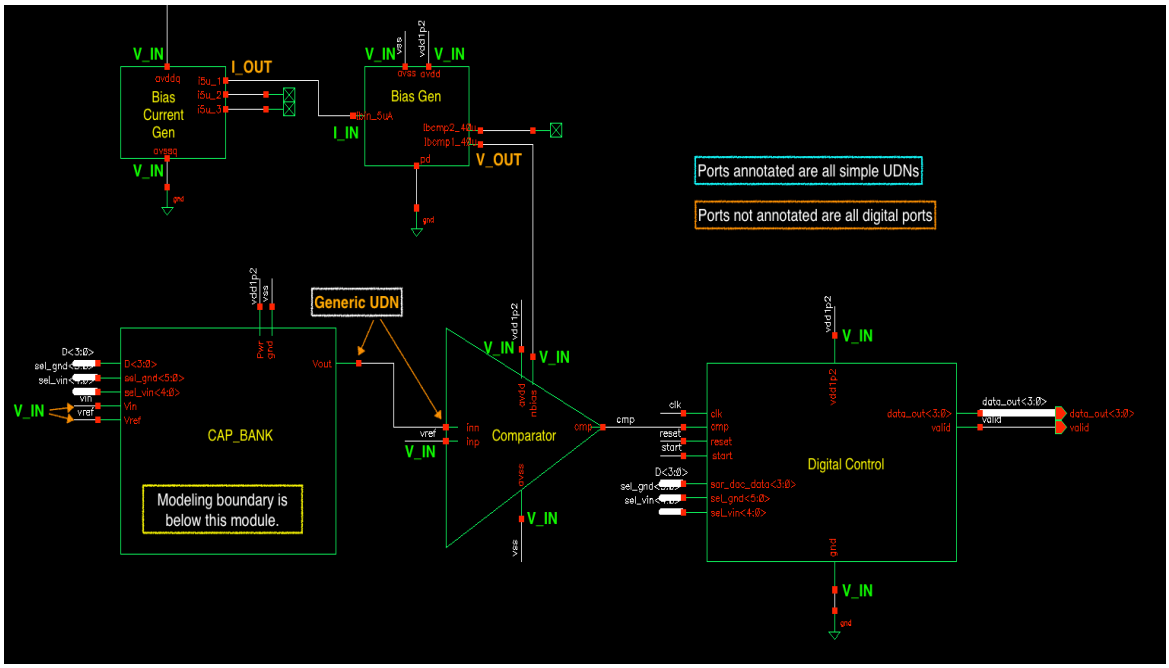

Figure 5. Schematic of SAR ADC with port types annotated.

To begin the process of Analog to Digital conversion, the Vin is sampled by connecting the bottom plates of capacitors to Vin and top plates to Vref (Fig. 4). Since the net Vcommon is being driven by a voltage source Vref, the resolved value is Vref and is stored irrespective of the previous voltage on the capacitors. Once sampling is done, the Vcommon node is left floating (`wZ) and bottom plates of capacitors are connected to ground. This traps a charge equal to C*(Vin–Vref) with a voltage at the top plate equaling '– (Vin–Vref)'. The resolution function detects that V_OUT driver (Vref) is `wZ and evaluates the resolved value as a weighted average. After the sampling step, the bottom plate of each capacitor is connected to Vref in a sequence (MSB to LSB). Depending on the output of the Comparator module, the capacitor is left connected to Vref or connected back to ground. If Vref is connected to a capacitor's bottom plate, then the charge across the capacitor is modeled as C*(– (Vin–Vref)+Vref). The resolution function computes resulting final voltage using the charge across each capacitor. System Verilog simulation waveforms in Fig. 6 show this happening on each clock cycle. Vref is chosen as 0.8V and with a 4-bit output, each LSB is equal to 0.05V. As can be seen in Fig. 6, a Vin of 0.05V results in data_out as 4'b0001 and a Vin of 0.15 gives data_out as 4'b0011. Fig. 7 shows Spice simulation versus RNM simulation of the conversion process of two data samples. Fig. 8 shows an overlay of analog and real waveforms of the Vcommon node.

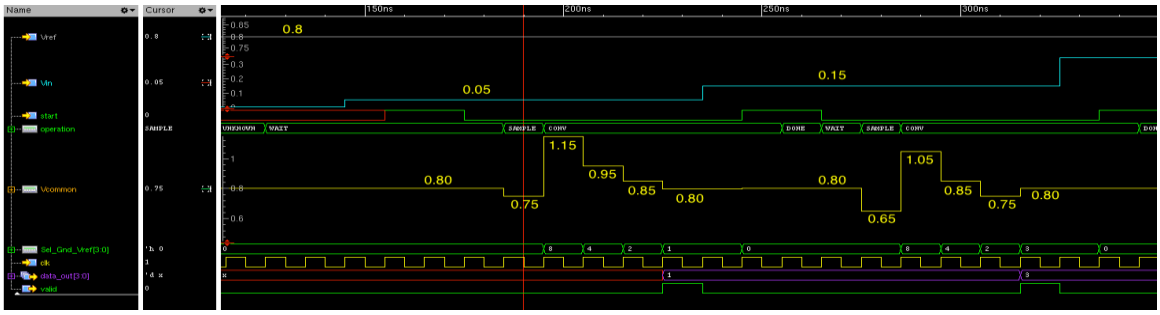

Figure 6. RNM simulation of the SAR ADC using generic UDN.

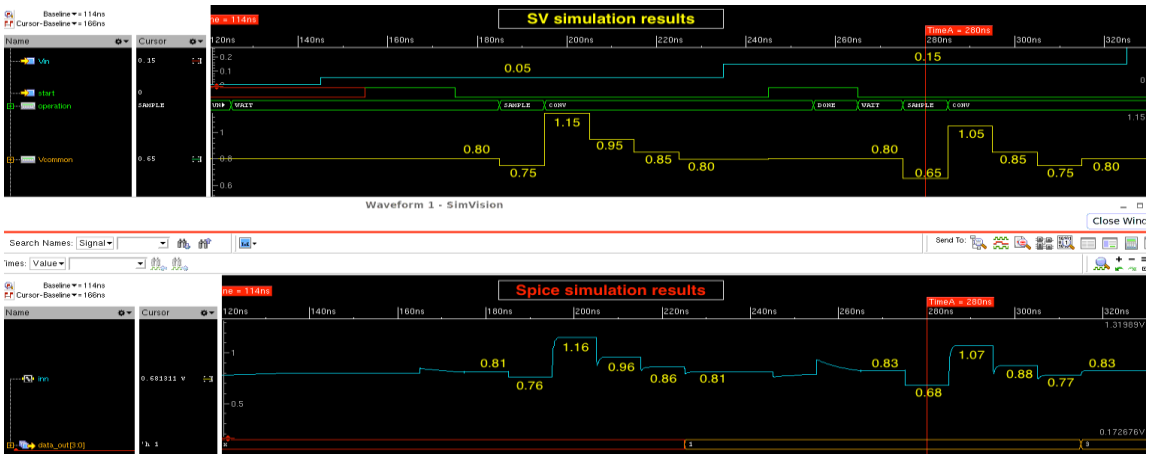

Figure 7: Waveforms of RNM simulation and Spice co-simulation.

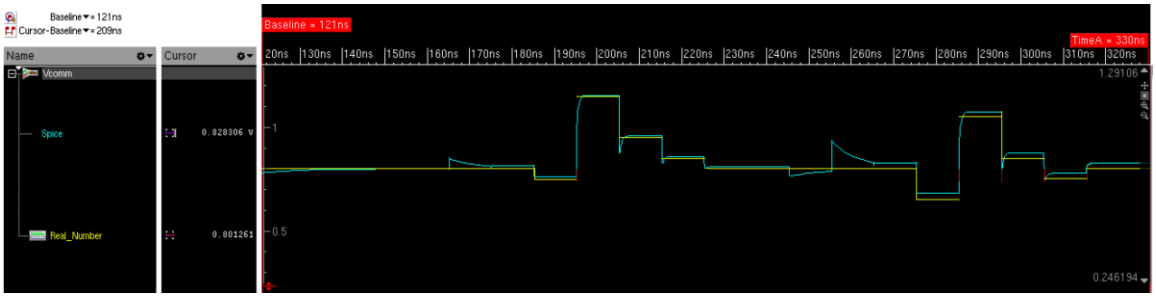

Figure 8: Overlay of analog and real waveforms of Vcommon (shown separate in Fig. 7).

Table 1 shows performance metrics calculated from a 512-point FFT of the time domain data (512 samples) from both Spice and RNM simulations. To generate time domain data, a sine wave of 0.55MHz with an amplitude of 250mV, sampled at a rate of 11.1MSPS, was used. Simulation performance was on par with wreal SV-RNM [2]. The performance gain over Spice simulation was a factor of 250 (using a test case to convert 512 samples of data). Fig. 9 and Fig. 10 show the input signal to the ADC and output of the ADC, for Spice and RNM simulations respectively. Fig. 11 shows the overlap of the outputs of the ADC for Spice and RNM. Fig. 12 shows the overlapped FFT of Spice and RNM simulations.

TABLE I
ADC PERFORMANCE METRICS - RNM VS SPICE

	RNM	Spice
SNR (dB)	20.05	18.16
THD (%)	10.9	10.2
SINAD	16.63	15.89
ENOB	3.03	2.72
Simulation Time (seconds)	4.7	1183.3

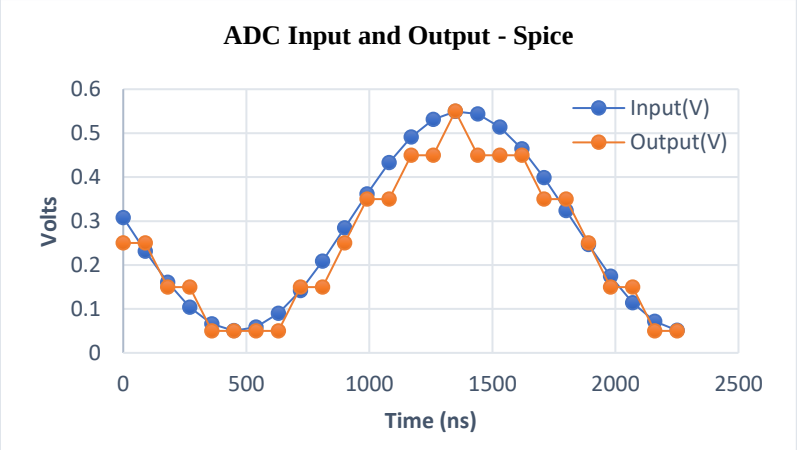

Figure 9: Input and output of ADC - Spice.

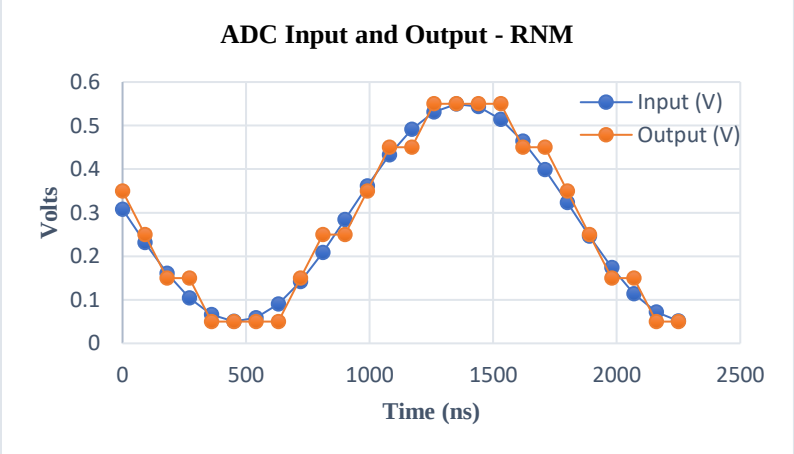

Figure 10: Input and output of ADC -RNM.

Using complex UDN to enable modeling of the SAR ADC at a lower hierarchical level, we were able to easily identify a functional bug where the switch polarity of the MSB capacitor (C4 in Fig. 4) was incorrect. Due to the incorrect polarity, the bottom port of C4 was being connected to 'Gnd' instead of 'Vref' and vice-versa during the conversion process. This bug resulted in an incorrect output of '0.5*Vref', or 8 LSBs, greater than the actual value and our modeling of charge redistribution exposed the bug. Fig. 13 shows waveforms of the correct and 'buggy' data.

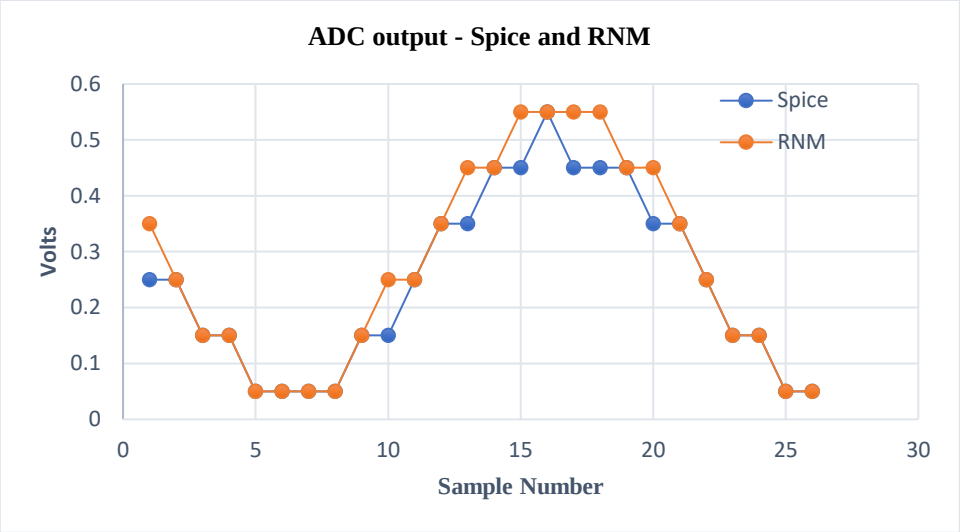

Figure 11: Overlapped ADC outputs of Spice and RNM.

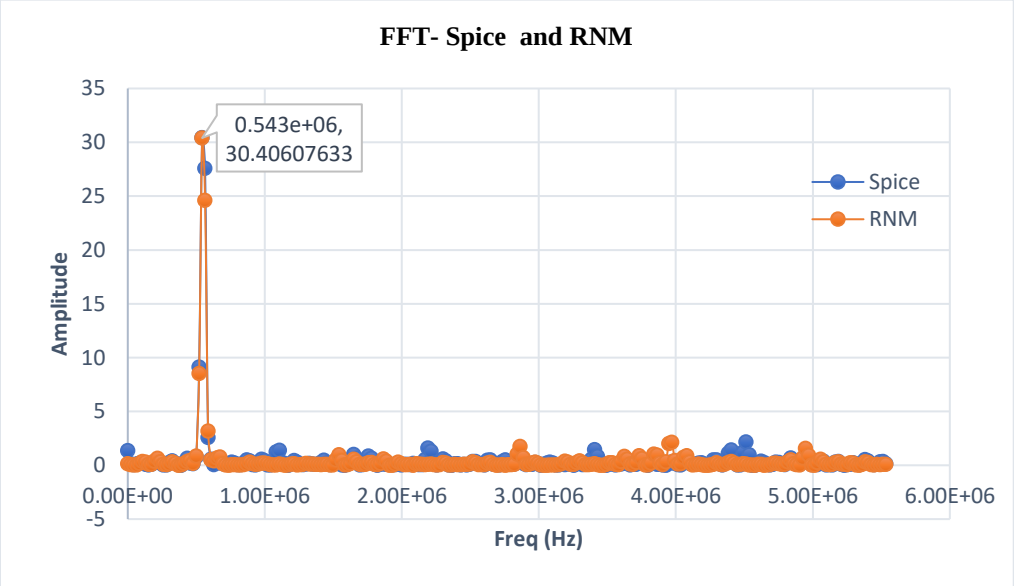

Figure 12: Overlapped FFT of Spice and RNM

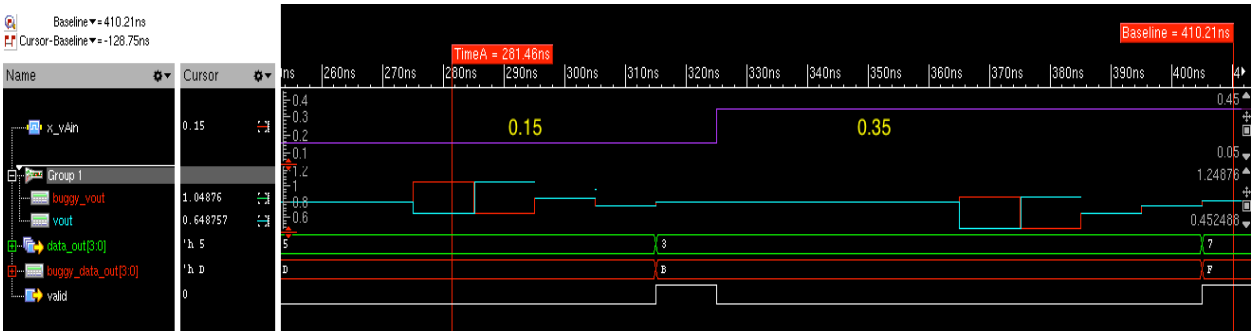

Figure 13: Actual and 'Buggy' Waveforms (Using Complex UDN Real Number Modeling)

VI. Modeling a Bidirectional Switch

Using this approach, a bidirectional switch was implemented for both voltage and current carrying nets. The switch implementation works by presenting the resolved resistance, current and voltage of all external sources from one port to the other port using V_PT ports as seen in Fig. 14. Using this modeling, we achieve a better functionality of a bidirectional switch compared to Cadence wtran implementation. A wtran bidirectional switch only accepts a scalar value [3]. Due to this, when connecting two current carrying nets, a wtran switch implementation would be unable to account for the input impedance of current sinks on either side. For example, consider the sample circuit where two branches of current are connected, as shown in Fig. 15. The wtran implementation would be unable to account for current flowing into the sinks R1 and R2, and resolution would be an algebraic sum of the current values of sources. The complex UDN switch accounts for the current sinks and accurately computes the resultant currents on each net. Simulation results are shown in Fig.16. However, at present, connecting the switches in series doesn't work.

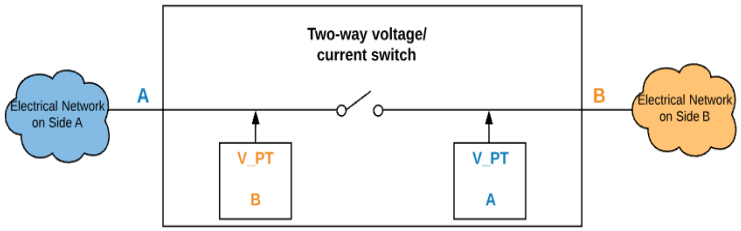

Figure 14. Illustration of implementation of a two-way switch.

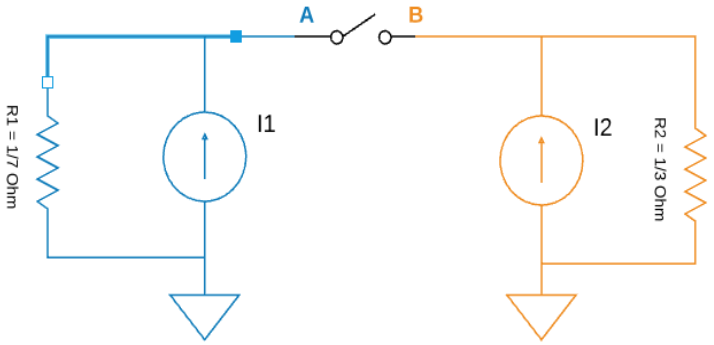

Figure 15. An example circuit for two-way current switch.

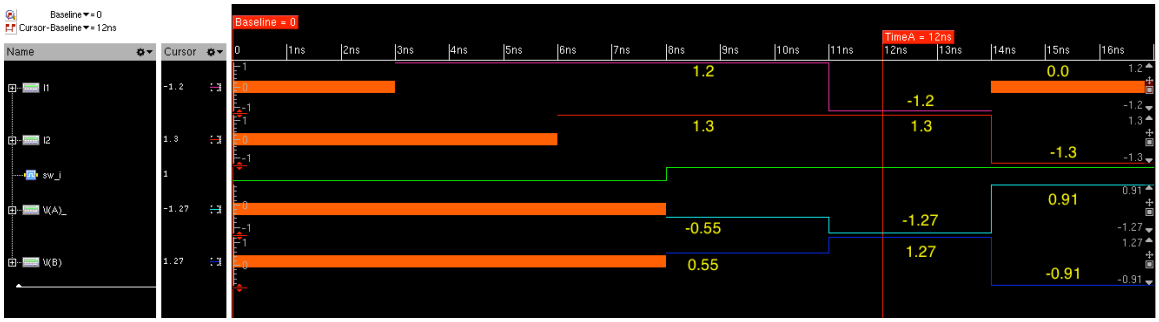

Figure 16. SV simulation result for the circuit in Fig. 14.

VII. Assumptions and Limitations

Our methodology is based on the following assumptions:

1. All active sources are ideal sources. Output resistance of V_OUT and I_OUT is not used to resolve a net. If non-ideal behavioral implementation is needed, then it shall be implemented in the model of the module using the resistance information of all the receivers extracted from the generic UDN and thereby adjusting the driver's output accordingly.
2. Since all the active sources are modeled as ideal sources, V_* and I_* connections are forbidden.

Some of the limitations using this methodology are as follows:

1. Implementation has been done entirely on Cadence Incisive simulator (version 15.2). Several features such as net coercion, automatic conversion of scalar values to simple UDN and some standard definitions may be different or unavailable on simulators from other vendors.
2. Due to limited availability of debug hooks into the resolution function, and given that the generic UDN resolution function is a heavier function compared to simpler UDN resolution functions, debugging internal aspects of the UDN gets tricky. More support from vendors might ease this.
3. In theory, all the analog nets in the design can be implemented using the generic UDN. However, due to the heavier resolution function, simulation performance may be impacted without much benefit.
4. Although several possible cases of connections between different port types have been implemented, series capacitance behavior hasn't been yet implemented. Two-way switch implementation doesn't allow chaining, i.e. the implementation works only for one switch connecting two branches of active sources.

VIII. Future Enhancements

Analog circuits' low power information is embedded in the schematic, unlike digital circuits (UPF). To aid power aware verification of analog circuits, our methodology envisions to create "power aware" ports by adding additional attributes to the complex UDN struct, such as power domain (PD) name and PD state. Using the power domain information transmitted on the complex UDN, checkers can be written inside the models of analog modules. Power intent is defined in a file - power intent file (PIF). Information such as power domain names, power rail values and power rail names are recorded in the PIF. Within the model of a module, PD name and its state are driven out on output ports (complex UDN) and are received at the input ports. By using the PD name and PD state of the port, and information about the PD for the module or port (from PIF), checkers verify the power intent. For example, consider a scenario where an isolator (ISO) and level shifter (LSF) are missing as shown in Fig. 17. To check if an LSF is needed, a comparison of the power supply voltages of the driver and receiver is done. Pseudo-code for the checker is shown in Fig. 17. This is made possible by making the driver PD information available to the receiver using complex UDN. Similarly, for checking a missing ISO, the PD state information of the driver is used to check if 'in_a' signal is at the isolation value (0 in this example). This approach may significantly reduce the effort required when compared to approaches using formal tools, as it uses the functional context of the signals.

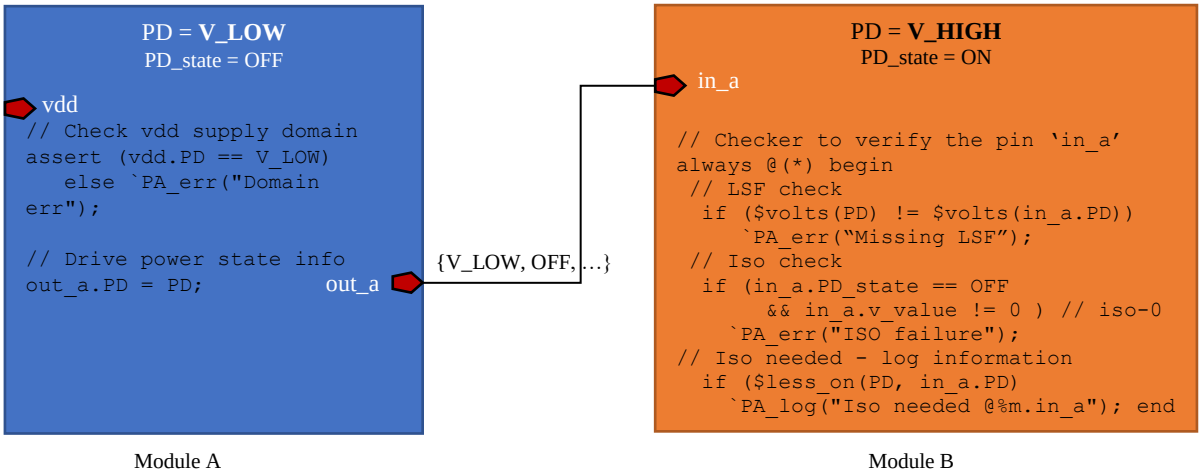

Figure 17. Example of missing LSF and ISO. Power Aware checks.

REFERENCES

[1] S. Balasubramanian and P. Hardee, S*olutions for Mixed-Signal SoC Verification Using Real Number Models*, Cadence Design Systems, 2013.

[2] B. John, T. Ziller, K. Fotouhi and A. Osman, *The How To's of Advanced Mixed-Signal Verification*, DVCON Europe. 2015.

[3] Spectre AMS Designer and Xcelium Simulator Mixed-Signal User Guide, version 18.09, Available: support.cadence.com