

SV_LUT: A SystemVerilog Look Up Table package for developing complex AMS Real Number Modeling

FNU Farshad
Ulkasemi Inc.
20045 Stevens Creek Blvd,
Suite 2B
Cupertino, CA 95014
farshad@ulkasemi.com

Shafaitul Islam Surush
Ulkasemi Inc.
20045 Stevens Creek Blvd,
Suite 2B
Cupertino, CA 95014
surush@ulkasemi.com

Simul Barua
Ulkasemi Inc.
20045 Stevens Creek Blvd,
Suite 2B
Cupertino, CA 95014
simul@ulkasemi.com

Abstract- Modern mixed systems on chips (SoC) require firm integration between analog and digital domains. Pre-silicon verification of analog and digital subsystems at the block and SoC levels is a must for ensuring the first working silicon. Unlike digital circuits, analog circuits do not yet have a standard verification methodology [1-3]. Usually, the spice-based analog circuits are replaced with faster event-driven behavioral models developed using Verilog-AMS or SystemVerilog in the verification environment. In recent years, SystemVerilog based real number modeling (SV-RNM) and user-defined nettype (SV-UDN) have been gaining popularity due to easier integration with the universal verification methodology (UVM) testbench [3]. Pre-computed look-up tables (LUT) are widely used to model the behavior of complex analog circuits [4]. These LUTs are usually populated using data from external sources like MATLAB. Unlike Verilog-AMS [5], SystemVerilog has no built-in support for creating LUT from external sources. This paper uses parameterized SystemVerilog macros to demonstrate a SystemVerilog package named `sv_lut_pkg` based mechanism for LUT table creation, population, and fetch values. The capabilities of the `sv_lut_pkg` package are exhibited by developing a LUT-based PTAT core model using SV-RNM flow.

I. INTRODUCTION

Ensuring first-time working silicon for complex mixed-signal SoC, and pre-silicon verification of analog and digital subsystems at the SoC and block levels is essential. In the SoC verification environment, spice-based analog circuits are converted into faster event-driven behavioral models using Verilog-AMS and SystemVerilog. Self-checking UVM-based testbenches is a de facto standard for verifying digital sub-systems. However, AMS verification has yet to develop a methodology. Several methodologies have been proposed for developing behavioral models for analog circuits using HDL languages [1-3]. Among them, SV-RNM and SV-UDN based methods are widely used, as they can be integrated into the UVM testbench without any wrapper [3].

Modeling complex analog circuit behavior, such as the characteristics of solar cells or Li-battery pre-computed data from external sources like MATLAB-based LUT, is widely used [6,7]. While Verilog-AMS has a built-in function named `$table_model()` as per section 9.21 of [5] to support LUT to import data from the `.tbl` file, SystemVerilog has no built-in function to support such functionality. This paper discusses the implementation of `sv_lut_pkg`, a SystemVerilog package that employs SV parameterized macros to create LUTs and import data from CSV files. The decision to use CSV format is based on its widespread use for exporting data from SPICE and MATLAB simulations, as opposed to `.tbl` format, which is not natively supported by these tools. The LUT also supports data interpolation capabilities for addressing unknown values between two known data points. To avoid any proprietary concerns, a simple proportional to absolute temperature (PTAT) core design is chosen to demonstrate the capabilities and integration process of `sv_lut_pkg` package macros in SV-RNM flow. The schematic of the PTAT core is developed using open-source software `quesstudio` [8]. Temperature vs. PTAT current values are exported from the simulated waveform into a CSV file. This file is then imported into the model to construct the LUT using `sv_lut_pkg` package macros. The SystemVerilog testbench passes the temperature as a parameter to the SV-RNM model. This parameter is used as an index to the LUT for fetching the corresponding PTAT current value.

II. OVERVIEW OF SV_LUT_PKG

Before discussing how `sv_lut_pkg` works, it is important to address some preliminary concepts, like the construction of the LUT structure and the role of the CSV files in defining the data points of the LUT. The CSV file contains exported behavioral data points containing signal value changes dumped from external sources like SPICE or MATLAB simulation waveforms. The goal of the LUT is to parse this CSV and map these data points in

searchable SV arrays done during the population of the LUT phase. The structure of the LUT is a SystemVerilog Struct containing two dynamic array/queue members. One dynamic array/queue stores the string column headers of the CSV file named `col_names[$]`. Another member is a 2D SV dynamic array/queue named `val[$][$]`, which is used to store the data of the CSV file. This way, the entire CSV file can be parsed and converted into a two-dimensional SV array that acts as a table that can be searched based on index values. So, the first indices of each row e.g., for a particular row x , `[x][0]` indices in the `val[$][$]` will contain the index values of the CSV file, and the rest of the indices, i.e., `[x][1]` and onward, will contain the data points of other CSV columns. The data is fetched from the LUT using one-to-one mapping. For example, a single index column `[x][0]` for row x is used to fetch a single data column value `[x][y]`, where y is the corresponding column index and $y > 0$. Fig.1 illustrates this conversion process visually.

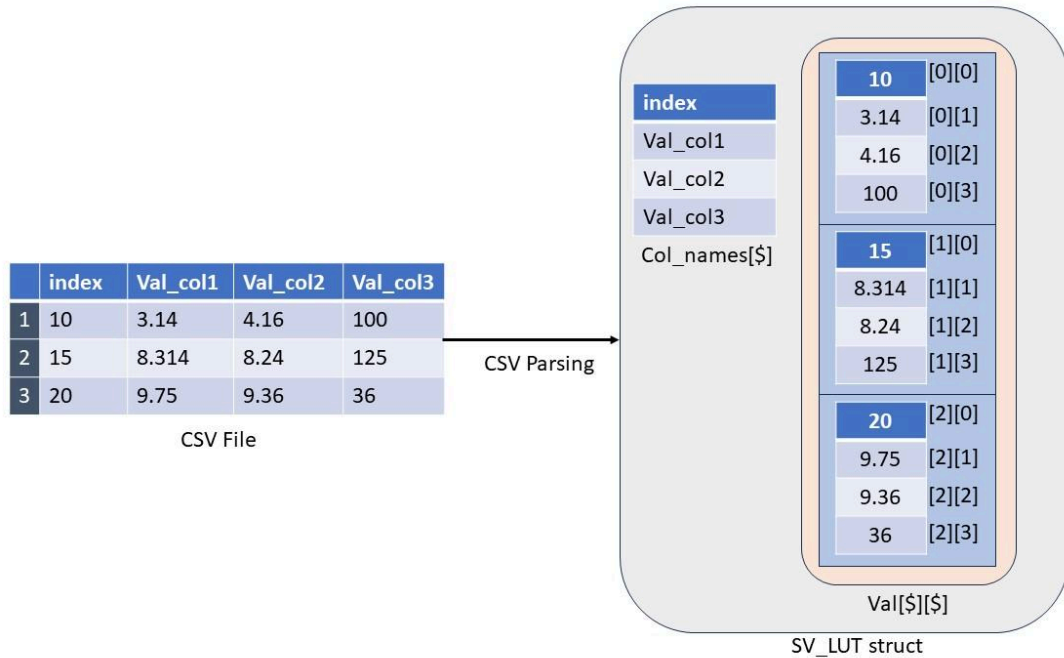


Figure 1. Visual illustration of CSV to SV_LUT struct conversion process

Parameterized preprocessor macros and functions are used to construct, populate, and fetch data values of LUT. The `sv_lut_pkg` encapsulates all of them. Two subsections sketch the functionality of the package. Section A focuses on the structure of the LUT and the macros responsible for populating it from the CSV file. Section B illustrates the mechanism for fetching values from the table. This subsection also shows the interpolation-based method for predicting unknown values.

A. Methods for defining and populating SV_LUT

LUT is defined as a SystemVerilog 'struct' data type containing the following members: a string queue "col_name" and a two-dimensional real queue "val". "col_name" stores the CSV headers where "val" contains the column value. The LUT is defined and populated by the preprocessor macro "POP_LUT". The "lut_struct_t" SV-Struct data type inside the macro defines the table. Furthermore, the SystemVerilog function "read_csv" parses the CSV file and stores values in the struct "lut_name" to populate it. The following arguments are used in the macro:

- lut_name: the name of the LUT.
- csv_file: full path to CSV file for parsing and populating LUT

Fig. 2 shows the structure and population mechanism of LUT.

```

testbench.sv  sv_lut_pkg.sv
1 package sv_lut_pkg;
2 typedef struct {
3     string col_name[$];
4     real val[$][$];
5 } lut_struct_t;
6
7 // create lut and populate it from csv file
8 // parameters: lut_name = name of lut variable
9 //                csv_file = full path to csv file
10 `define POP_LUT(lut_name, csv_file) \
11     lut_struct_t lut_name; \
12     int ``lut_name``_x_indx_col_num = 0; \
13     real ``lut_name``_x_indx_val_tol = 0.001; \
14     initial begin \
15         assert(read_csv(`"csv_file`", lut_name)); \
16     end

```

Figure 2. Code snippet demonstrating LUT definition and population mechanism

Based on the requirement of the RNM modeling, we can create multiple LUTs from parsing multiple CSV files. Given that the “lut_name” and csv_file parameters used in the POP_LUT() macro are unique for each of the LUTs.

B. Methods for fetching values from LUT

Once the LUT is generated and populated, the user can use the SV_LUT() preprocessor macro to fetch the respective value using the aforementioned arguments.

- x_index_val: Value of the index column used to look up the corresponding value of y_index argument
- y_index: defines column number in the lookup table used in correspondence with x-index. This column number is associated with the column number in the CSV file. Essentially, this parameter maps the index and output CSV column together i.e., one-to-one mapping.
- lut_name: same as the lut_name parameter in POP_LUT() preprocessor macro
- interpolation_type: Declares LUT interpolation method. The value for this parameter is the same as the value defined in section 9.21 of [5], i.e., “1L” for selecting “linear interpolation,” “1C” for selecting cubic spline interpolation, etc.

Fig. 3 illustrates the SV_LUT() preprocessor macro and its underlying functions,

```

testbench.sv  sv_lut_pkg.sv
80 // main lut function
81 // parameters:
82 //   x_indx: (string) lookup index of look up table
83 //   lut_name: ()
84 //   y_indx: (string) return index selector for LUT
85 //   interpolation_type: (string) L: linear interpolation, C: cuble spline interpolation
86 //   x_indx_col_num: (int) column number for x_index
87 `define SV_LUT(x_indx_val, lut_name, y_indx, interpolation_type) \
88     lut_fetch_val(x_indx_val, lut_name, y_indx, interpolation_type, ``lut_name``_x_indx_col_num,
89     ``lut_name``_x_indx_val_tol);
89
90     function real lut_fetch_val(real x_indx, lut_struct_t lut, int y_indx_col, string intrpol_type,
91     int x_indx_col_num=0, real x_indx_val_reltol=0.01);

```

Figure 3. Code snippet demonstrating SV_LUT macro and its underlined function

The lut_fetch_val() function employs all the logic responsible for fetching the y_index_val values corresponding to x_index_val values. As the output value is fetched from the LUT based on the “y_indx” parameter representing CSV data column number, it is possible to fetch different column values using single x_index_val by calling SV_LUT() macro with different “y_indx” parameters. At this juncture, the user has to specify the interpolation method that will be used to make predictions on the unknown data points using the parameter “interpolation_type”. Currently, linear interpolation logic is implemented in this scope, with a plan to add cubic spline interpolation support later. The authors are also planning to add support for the extrapolation features.

As real numbers are used for indexing the LUT, using an exact match in the LUT index is challenging. To address this issue, the SV_LUT employs a relative tolerance of 1% while matching the index value. This tolerance level can be adjusted with the parameter “x_indx_val_reltol” of the lut_fetch_val() function.

III. APPLICATION OF SV_LUT_PKG IN AMS MODELING

Three looming sections utilize the PTAT core design to demonstrate the capabilities and integration process of sv_lut_pkg package macros in SV-RNM flow. Section A provides an overview of the schematic design along with the simulation waveform and a snapshot of CSV document data exported from the results. Section B illustrates the design of the SV-RNM model using LUT to model the circuit's behavior. Section C establishes the results.

A. Overview of PTAT core design

The PTAT core design consists of vdd, gnd, and out ports for power, ground, and PTAT current output, respectively. The schematic is simulated for the temperature range of -50 °C to 125 °C. The schematic diagram and simulation waveform are shown in Fig. 4.

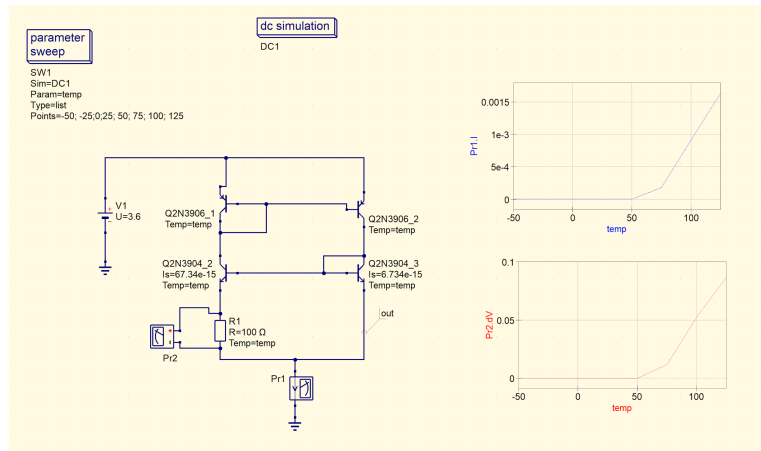


Figure 4. Schematic design and simulation waveforms of temp vs. PTAT current (Pr1.I) and temp vs. PTAT voltage (Pr2.dV)

The simulation waveform data is exported to the “ptat_sim.csv” file for SV-RNM model development.

	A	B
1	temperature (°C)	r Pr1.I (A)
2	-50	4.85E-13
3	-25	1.10E-11
4	0	2.10E-10
5	25	4.42E-09
6	50	5.85E-07
7	75	0.000176578
8	100	0.000911202
9	125	0.00163012
10		

Figure 5. CSV file containing temperature and PTAT current (rPr1.I)

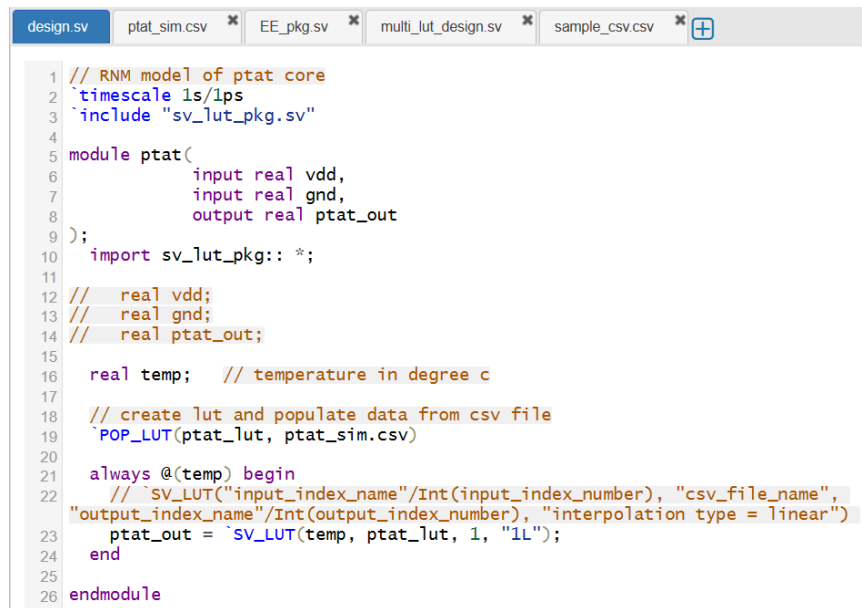
As shown in Fig. 5, the exported CSV file contains PTAT current (rPr1.I) across register R1 for the temperature range of -50°C to 125°C with an incremental step of 25°C.

B. SV RNM modeling demonstration with sv_lut_pkg package

All the input and output electrical ports of the PTAT core are defined as real data types. The model also contains a real variable named “temp”, which will be used to set the temperature from the testbench. The value of the PTAT current will be generated with the help of defined and populated LUT. This step can be done using the POP_LUT macro. The user passes two parameters to this macro: the LUT name and the CSV file. Then, the corresponding value of the PTAT current can be retrieved from the LUT using the SV_LUT() preprocessor macro. Arguments to this macro are as follows:

- x_index_val: The “temp” variable value is used as x_index,
- lut_name: LUT name is set to ptat_lut, same as in the POP_LUT macro,
- y_index: The second column is selected as y_val_col,
- interpolation_type: interpolation type parameter is set to linear using “1L” parameter.

The “temp” variable is used as a trigger in the always block. As soon as the temperature is set from the testbench, the model will fetch the corresponding PTAT value from the LUT. Fig. 6 illustrates LUT's generation and integration process in the proposed PTAT core SV-RNM model.



```
1 // RNM model of ptat core
2 timescale 1s/1ps
3 `include "sv_lut_pkg.sv"
4
5 module ptat(
6     input real vdd,
7     input real gnd,
8     output real ptat_out
9 );
10 import sv_lut_pkg:: *;
11
12 // real vdd;
13 // real gnd;
14 // real ptat_out;
15
16 real temp; // temperature in degree c
17
18 // create lut and populate data from csv file
19 POP_LUT(ptat_lut, ptat_sim.csv)
20
21 always @(temp) begin
22     // `SV_LUT("input_index_name"/Int(input_index_number), "csv_file_name",
23     "output_index_name"/Int(output_index_number), "interpolation type = linear")
24     ptat_out = `SV_LUT(temp, ptat_lut, 1, "1L");
25 end
26 endmodule
```

Figure 6. LUT generation process in PTAT core SV-RNM model

C. Simulation results

The simple testbench of Fig. 7 tests the SV-RNM of the PTAT core. Inside the test loop, the “temp” parameter is varied between -50°C to 150°C with an incremental step size of 5°C. The resulting PTAT current is printed out in the console. The simulation results are shown in Fig. 8 and Fig. 9.

```

testbench.sv  sv_lut_pkg.sv * +
5 module tb;
6
7   import sv_lut_pkg::*;
8
9   real vdd, gnd, out;
10
11  ptat DUT(
12    .vdd(vdd),
13    .gnd(gnd),
14    .ptat_out(out)
15  );
16
17  initial begin
18    for(int temp = -50; temp<150; temp=temp+5) begin
19      $display("Setting temp: %0d", temp);
20      DUT.temp = temp;
21      # 100us;
22      $display("V(out):%0e", out);
23    end
24  end
25
26  initial begin
27    vdd = 3;
28    gnd = 0;
29  end
30
31  initial begin
32    $dumpvars(0, DUT);
33    $dumpfile("ptat_sim.vcd");
34  end
35 endmodule

```

Figure 7. SystemVerilog Testbench code for testing the PTAT core SV-RNM model

# V(out):4.850000e-13	# Setting temp: 10	# Setting temp: 65
# Setting temp: -45	# V(out):1.894000e-09	# V(out):1.061808e-04
# V(out):2.588000e-12	# Setting temp: 15	# Setting temp: 70
# Setting temp: -40	# V(out):2.736000e-09	# V(out):1.413794e-04
# V(out):4.691000e-12	# Setting temp: 20	# Setting temp: 75
# Setting temp: -35	# V(out):3.578000e-09	# V(out):1.765780e-04
# V(out):6.794000e-12	# Setting temp: 25	# Setting temp: 80
# Setting temp: -30	# V(out):4.420000e-09	# V(out):3.235028e-04
# V(out):8.897000e-12	# Setting temp: 30	# Setting temp: 85
# Setting temp: -25	# V(out):1.205360e-07	# V(out):4.704276e-04
# V(out):1.100000e-11	# Setting temp: 35	# Setting temp: 90
# Setting temp: -20	# V(out):2.366520e-07	# V(out):6.173524e-04
# V(out):5.080000e-11	# Setting temp: 40	# Setting temp: 95
# Setting temp: -15	# V(out):3.527680e-07	# V(out):7.642772e-04
# V(out):9.060000e-11	# Setting temp: 45	# Setting temp: 100
# Setting temp: -10	# V(out):4.688840e-07	# V(out):9.112020e-04
# V(out):1.304000e-10	# Setting temp: 50	# Setting temp: 105
# Setting temp: -5	# V(out):5.850000e-07	# V(out):1.054986e-03
# V(out):1.702000e-10	# Setting temp: 55	# Setting temp: 110
# Setting temp: 0	# V(out):3.578360e-05	# V(out):1.198769e-03
# V(out):2.100000e-10	# Setting temp: 60	# Setting temp: 115
# Setting temp: 5	# V(out):7.098220e-05	# V(out):1.342553e-03
# V(out):1.052000e-09		# Setting temp: 120
		# V(out):1.486336e-03

Figure 8. Snapshot of Simulation results

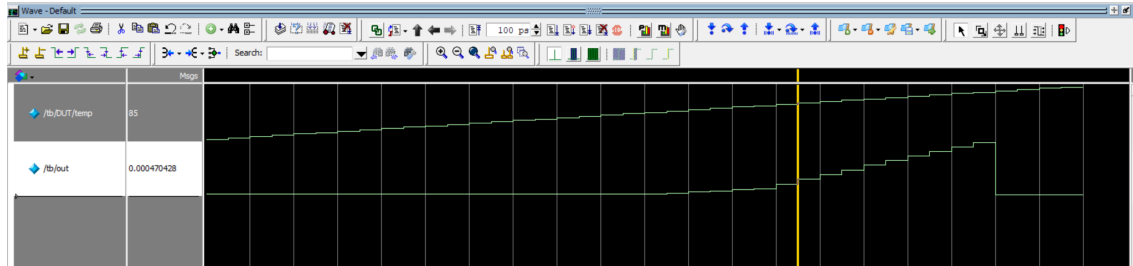


Figure 9. Snapshot of Simulation waveform

The CSV file only contains 8 data points (i.e. -50°C , -25°C , 0°C , 25°C , 50°C , 75°C , 100°C , 125°C). The LUT uses linear interpolation to predict the PTAT current values for all the temperature data points among these 8 data points.

IV. CONCLUSION AND FUTURE WORKS

This paper demonstrates that the proposed `sv_lut_pkg` package is effortless to integrate and consume in traditional SV-RNM flow. Most of the heavy work is done in the background, enabling the verification engineers to model many complex scenarios that are difficult to model otherwise. Current implementation of `sv_lut_pkg` maps single input to fetch single output and does not support mapping of multiple inputs for fetching corresponding output. In addition, the CSV file parser has no data validation capability, as the current work assumes that the data will be exported from external sources like MATLAB and simulation waveforms, with a meager chance of providing invalid data. The lack of data validation can be problematic if the CSV file is manually generated and contains invalid data. Another drawback of the current implementation is that if the CSV file is huge (i.e., greater than 1000 rows and columns), the search mechanism of the LUT takes a long time. These limitations create an opportunity for future scopes of work. For all its limitations, the current version of this package has some outstanding features to contribute to the mixed signal verification arena.

ACKNOWLEDGMENT

The authors would like to thank their parents and colleagues for their encouragement and support.

REFERENCES

- [1] R. Shi, P. Bhinghe, P. Birdsong, G. Chaitanya, and K. Jani, "Mixed-Signal Design Verification: Leveraging the Best of AMS and DMS." Accessed: Sep. 15, 2023. [Online]. Available: <https://dvcon-proceedings.org/wp-content/uploads/Mixed-Signal-Design-Verification-Leveraging-the-Best-of-AMS-and-DMS-2.pdf>
- [2] A. Freitas, "UVM Ready: Transitioning Mixed-Signal Verification Environments to Universal Verification Methodology." Accessed: Sep. 15, 2023. [Online]. Available: https://dvcon-europe.org/wp-content/uploads/sites/14/2022/03/T1_3_paper.pdf
- [3] N. Sonara, N. Mohammad, P. Singh, D. Stoops, J. Fernando, and K. Sudarshana, "SCALABLE, RE-USABLE UVM DMS AMS BASED VERIFICATION METHODOLOGY FOR MIXED-SIGNAL SOCS." Accessed: Sep. 15, 2023. [Online]. Available: <https://dvcon-proceedings.org/wp-content/uploads/scalable-re-usable-uvms-dms-ams-based-verification-methodology-for-mixed-signal-socs.pdf>
- [4] A. A. Youssef, B. Murmann, and H. Omran, "Analog IC design using Precomputed lookup Tables: Challenges and solutions," *IEEE Access*, vol. 8, pp. 134640–134652, Jan. 2020, doi: 10.1109/access.2020.3010875.
- [5] "Verilog-AMS Language Reference Manual," 2014. Available: <https://www.accelera.org/images/downloads/standards/v-ams/VAMS-LRM-2-4.pdf>
- [6] A. Ilyas, M. R. Khan, and M. Ayyub, "Lookup table based modeling and simulation of solar photovoltaic system," *2015 Annual IEEE India Conference (INDICON)*, Dec. 2015, doi: 10.1109/indicon.2015.7443268.
- [7] J. Liobe, "Improving Analog Simulation Speed Using Verilog-A / Verilog-AMS Without Compromising Circuit Accuracy," www.intrinsix.com. <https://www.intrinsix.com/blog/improving-analog-simulation-speed-using-verilog-a/-verilog-ams-without-compromising-circuit-accuracy> (accessed Sep. 15, 2023).
- [8] QucsStudio, "Home - QucsStudio," *QucsStudio*, Aug. 31, 2023. <http://qucsstudio.de/>