Equivalence Checking of System-Level and SPICE-Level Models of Static Nonlinear Circuits

Kemal Çağlar Coşkun[ш] Muhammad Hassan[ш,*] Rolf Drechsler[ш,*]
[ш]Institute of Computer Science, University of Bremen, 28359 Bremen, Germany
[*]Cyber-Physical Systems, DFKI GmbH, 28359 Bremen, Germany
muhammad.hassan@dfki.de {kcoskun,drechsle}@uni-bremen.de

***Abstract*—Recently, *Signal Flow Graphs* (SFGs) have been successfully leveraged to show equivalence for linear analog circuits at system-level and SPICE-level. However, this is clearly not sufficient as the true complexity stems from nonlinear analog circuits. In this paper, we go beyond linear analog circuits, i.e., we extend the SFGs and develop the *Modified Signal-Flow Graph* (MSFG), to show equivalence between system-level and SPICE-level representations of static nonlinear analog circuits. First, we map the nonlinear circuits to MSFGs. Afterwards, graph simplification and functional approximation (in particular *Legendre polynomials*) techniques are used to create minimal MSFG and canonical MSFG. This enables us to compare the MSFGs even if they have vastly different structures. Finally, we propose a *similarity metric* that calculates the similarity between SPICE-level and system-level models. By successfully applying the proposed equivalence checking technique to benchmark circuits, we demonstrate its applicability.**

***Index Terms*—equivalence checking, formal verification, nonlinear circuits, circuit analysis, flow graphs, approximation methods**

I. Introduction

The increasing complexity of analog circuits and the growing system integration of analog and digital circuits have emerged as a bottleneck to analog design verification. The prohibitively long simulation time of SPICE-level models is a significant obstacle in this regard [1]. Despite being slow, SPICE-level simulations [2], which are often used with manual inspection of the results, are nonetheless regarded as the gold standard and cannot be ignored. To achieve much higher simulation performance and early design verification of the *Design Under Verification* (DUV), different levels of circuit design abstraction and alternative representations of the circuit, such as behavioral models, can be used.

As a result, top-down design principles are becoming more prevalent in analog systems. In this sense, *Virtual Prototyping* (VP) at the *Electronic System Level* (ESL) is heavily used today [1], [3]–[6]. With a speed boost of approximately $100,000$ times over SPICE-level simulations [1], the *Timed Data Flow* (TDF) *Model of Computation* (MoC) offered in SystemC AMS provides a good trade-off between precision and simulation speed at the system-level, offers a design refinement process, and enables early verification.

However, the absence of equivalence checking methodologies for SystemC AMS and SPICE-level models is one of the major obstacles to the adoption of SystemC AMS system-level models. Equivalence checking identifies the functional equivalency of two implementations of a design. The implementations may use various description environments, such as transistor netlists and system-level languages, as well as various abstraction levels. Although equivalence checking techniques are well established in the digital domain [7]–[9], formal or at least formalized verification techniques are lacking in analog circuit design routines [10]–[18]. When discussing equivalence checking strategies, we generally consider state-space coverage, model checking, and reachability. Regardless of the specific strategy, there is little confidence in the use of system-level analog models. As a result, it becomes challenging to solely rely on SystemC AMS system-level models.

Contribution: Recently, *Signal Flow Graphs* (SFGs) have been successfully leveraged to show equivalence for linear analog circuits at system-level and SPICE-level [19], [20]. However, this is clearly not sufficient as the true complexity stems from nonlinear behaviors. In this paper, we go beyond linear analog systems, i.e., we expand the SFG techniques proposed in [19], [20] and propose a novel methodology to show equivalency between system-level and SPICE-level models of static nonlinear circuits. We make the following concrete contributions:

1) We extend SFGs to *Modified Signal-Flow Graphs* (MSFGs), as a mutually representative form for nonlinear SPICE-level and system-level models.
2) We automatically transform MSFGs to equivalent forms, i.e., a minimal MSFG and a canonical MSFG, with graph simplification operations and approximation techniques. Here, we use *Legendre polynomial series* in particular to approximate the functional description.
3) We introduce a *similarity metric* that quantitatively measures the percentage of equivalence between SPICE-level and system-level models. This enables us to clearly see the quality of system-level models.
4) The methodology spans the complete class of static nonlinear *Single-Input Single-Output* (SISO) analog circuits.
5) We demonstrate the applicability on a common-source amplifier circuit, a *Phase-locked Loop* (PLL) charge-pump circuit from [16], and a squaring circuit.

II. Related Work

Several equivalence checking methods were compared in the surveys [10], [11] and it was concluded that many methods

This work was supported in part by the German Federal Ministry of Education and Research (BMBF) within the project AUTOASSERT under contract no. 16ME0117.

TABLE I
A COMPARISON OF THE RELATED WORKS

Source	Approach	Verification Coverage	Applicable Circuits
[12]–[15]	State-space-based	Only at finite number of locations in the state-space	Dynamic nonlinear
[16]–[18]	Simulation-based	Only for finite number of input signals	Dynamic nonlinear
[19], [20]	Structural analysis	Complete coverage	Dynamic Linear
Proposed work	Structural analysis	Complete approximative coverage	Static nonlinear

attempted to strike a balance between excessive cautiousness and completeness, and that defining the coverage metrics was challenging.

In [12], the vector fields of two models were compared on a point grid for state-space-based equivalence checking. The approach initially worked for SISO circuits with differential equations, but was expanded to models with differential-algebraic equations in [13] and multi-input multi-output circuits in [15]. Although it can be applied to a wide range of circuits, certain crucial dynamics can be missed, since the points on the grid are set distances apart on the canonical state space.

Simulation-based equivalence checking approaches such as [16]–[18], focus on three main topics: mapping techniques for comparing signals in separate abstraction levels, decreasing the number of input stimuli to shorten computation times, and automatic input generation for coverage optimization. However, since only a finite number of simulations can be performed, and due to the continuous nature of analog circuits, full coverage cannot be obtained with such approaches.

A graph-based equivalence checking approach was developed in [19], [20] where SPICE-level and system-level models are mapped to each other through graph simplification techniques. This methodology achieves full coverage, but only works for linear circuits.

Table I provides a brief comparison between these various works. As seen in Table I, some methods in the existing literature do not check equivalence with complete coverage of behavior. Therefore, these methods might return false positives when the models behave differently in an overlooked gap. Other methods achieve complete coverage but in the expanse of generality, since these only work on linear circuits.

Our proposed methodology tries to strike a balance while extending the graph-based approaches toward static nonlinear circuits. While we still preserve complete coverage, we tolerate some approximations to increase generality.

III. PRELIMINARIES

In this section, we first provide a brief overview of the methods we use for comparing nonlinear functions. Then, we give a running example that we will use later to illustrate the proposed methodology. For brevity, we do not give a proper introduction to SystemC AMS; instead, we recommend the SystemC AMS user guide [4].

A. Comparison of Nonlinear Functions

Nonlinear functions can take many different forms and structures, as a result, direct comparison is generally not possible. To address this issue, we approximate the functions with *Legendre polynomials*, which are orthogonal for unity weight in $[-1, 1]$, as follows:

$$f_{\text{app}}(x) = \sum_{n=0}^{N} a_n L_n^{(s)}(x) \tag{1}$$

where $L_n^{(s)}$ is a transformed form of the *Legendre polynomial* L_n such that it is orthonormal for unity weight in $[a, b]$. The coefficients of this polynomial series, given as,

$$c = \begin{bmatrix} a_0 & a_1 & \cdots & a_N \end{bmatrix} \tag{2}$$

are then used to calculate the *similarity metric*.

Since it is common for analog circuits to have maximum absolute ratings for their inputs and outputs, or to have a defined operation region, these can be used to define the interval $[a, b]$.

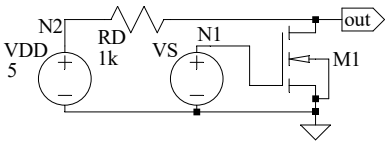

Fig. 1. Running example: *Common-Source* (CS) amplifier with components *Resistor* (RD), *MOSFET* (M1), and *Voltage Source* (VDD, VS).

```
1  void cs::processing() {
2    double v_i = 2.85 + 85 * (v_VS - 0.75);
3    v_out.write(std::max(0.0,std::min(v_i,5.0))); }
```

Fig. 2. The CS amplifier's system-level implementation in SystemC AMS.

B. Running Example: Common-Source Amplifier

To motivate the development of this novel equivalence checking methodology and to illustrate how it works, we consider the single-stage common-source amplifier (Fig. 1) taken from [21]. Such amplifiers are used in many applications such as audio crossovers, sensor circuits, and controller circuits.

The example circuit is designed for a linear gain of -85 and an operation domain of $V_{V_S} \in [0.73\,\text{V}, 0.77\,\text{V}]$ for the input. The supply voltage is assumed to be 5 V. The behavior of the MOSFET in the saturation region is modeled as

$$I_D = \frac{1}{2} K_P \frac{W}{L} (V_{GS} - V_{TO})^2 (1 + \lambda V_{DS})$$

where $K_P = 13.4225\,\text{mAV}^{-2}$, $L = 0.5\,\mu\text{m}$, $W = 50\,\mu\text{m}$, $V_{TO} = 0.7\,\text{V}$, and $\lambda = 0.1\,\text{V}^{-1}$. The resistance R_D is set to $1\,\text{k}\Omega$ to approximate the linear gain specification. The circuit is modeled in LTSpice [22] and its netlist is used in the proposed methodology.

The system-level implementation of the circuit models the saturation of the circuit and its desired linear behavior as,

$$V_i = 2.85 - 85\,(V_{V_S} - 0.75)$$
$$V_{\text{out}} = \begin{cases} 5\,, & V_i \geq 5 \\ 0\,, & V_i \leq 0 \\ V_i\,, & \text{otherwise} \end{cases} \tag{3}$$

where V_{i} stands for some intermediate variable. The SystemC AMS implementation is given in Fig. 2.

In the next section, we present our graph-based equivalence checking methodology for static nonlinear circuits.

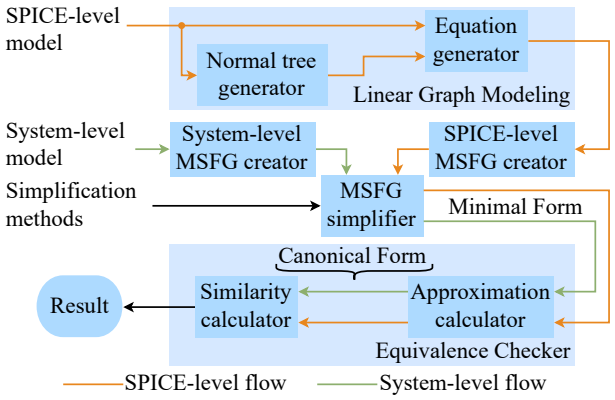

Fig. 3. Overview of the proposed equivalence checking methodology

IV. Signal-Flow Driven Equivalence Checking Methodology

In this section, we propose our MSFG-based novel equivalence checking methodology for system-level and SPICE-level circuits. First, we give an overview of our proposed methodology and introduce MSFGs. Later, we describe methods to create and simplify an MSFG. The simplified MSFGs are then compared using the techniques introduced in Section III-A. Finally, we apply this methodology to our running example from Section III-B.

A. Methodology Overview

The block-diagram overview of our equivalence checking methodology is given in Fig. 3. The linear graph modeling method [23], which consists of the normal tree generator and the equation generator blocks, is used on the SPICE-level model to generate a complete set of explicit equations. After this method, MSFGs for the SPICE-level and system-level descriptions are created with the SPICE-level MSFG creator and system-level MSFG creator, respectively. The MSFG simplifier then simplifies both MSFGs to a minimal form with the simplification methods described in Section IV-E. The only remaining functions of the minimal MSFGs are then approximated by the *Approximation Calculator*. Finally, the *Similarity Calculator* calculates the *similarity metric* as explained in Section IV-F. All of these computations are performed automatically and statically, and no simulations are involved.

B. Modified Signal-Flow Graphs

SFGs were introduced in [24] as a graph-based representation of a system of explicit algebraic equations as shown in Eq. (4):

$$\bar{x} = f(\bar{x}, \bar{u}) \tag{4}$$

where $\bar{x}$ is an array of variables and $\bar{u}$ is an array of inputs. Although it was introduced as a general representation that would also work for nonlinear equations, the SFG is commonly restricted to linear equations [25].

The original SFG structure is not well suited for this work, since it is restricted to either complicated functions (functions with multiple distinct operators) of a single variable, or simple, multivariate functions. Particularly, the edges of the SFG, which always go from a single source node to a single destination node, can represent only single-variable functions. On the other hand, the nodes of the SFG, which are simultaneously used to represent variables, can only represent simple functions with single operators such as $\sum$ or $\prod$. However, many circuit components such as MOSFETs are modeled with functions that are both complicated and multivariate. Therefore, neither edges nor nodes of SFGs can represent them. To support complicated, multivariate functions, we introduce the MSFG. In the MSFG, the nodes are used exclusively to represent functions, and the variables are moved to the edges.

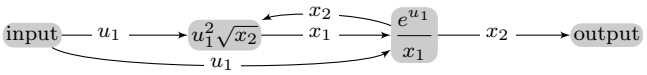

Fig. 4. The *Modified Signal-Flow Graph* (MSFG) of Equation (5).

As an example, consider the MSFG in Fig. 4 and its equivalent system of explicit algebraic equations given in Eq. (5). The variables u_1, x_1, and x_2 are on the edges of the MSFG, whereas the functions are in the nodes. Nodes labeled "input" and "output" are the only types of nodes that do not represent a function. The output variable of a node function, i.e., the left-hand side of the explicit equation, is called the outgoing variable of the node. The edges represent the dependency between the functions and variables and are labeled with the outgoing variable of their source node. For example, the outgoing variable of the node $\left(u_1^2\sqrt{x_2}\right)$ is x_1 and the edge labeled x_1 represents the dependency of its destination node $\left(\frac{e^{u_1}}{x_1}\right)$ to x_1.

$$x_1 = u_1^2\sqrt{x_2}\,, \quad x_2 = \frac{e^{u_1}}{x_1} \tag{5}$$

C. Creating an MSFG from System-Level Descriptions

MSFGs for system-level descriptions can be created directly from programming code, since these have a single variable on the left-hand side and are already in the form of Eq. (4). Equations in this form can be directly transformed into an MSFG as explained in Section IV-B. The methodology supports all functions that are supported by the underlying symbolic engine. These include but are not limited to functions such as multiplication, exponentiation, logarithmic functions, sinusoidal functions, and piecewise (i.e. if statements) functions.

D. Creating an MSFG from SPICE-Level Descriptions

To create an MSFG from the SPICE-level model, equations in the form of Eq. (4) must be obtained. For this purpose, we use the linear graph modeling method because it generates equations that are readily in the form of Eq. (4), whereas other methods generate implicit equations in the form of

$$f(\cdot) = g(\cdot)$$

that would require us to do further processing.

The linear graph modeling method uses the voltages on and currents through the circuit components as variables when creating equations. This is in contrast to nodal-analysis and loop-analysis, which use node voltages and loop currents, respectively.

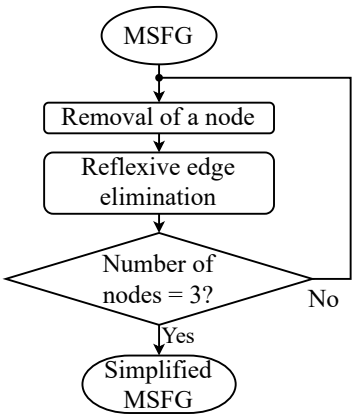

Fig. 5. Overview of MSFG simplification process

In the linear graph modeling technique, an explicit equation for each variable is generated from either elemental models, Kirchhoff's voltage law, or Kirchhoff's current law. To determine which of these will be used to generate the explicit equation, a special type of minimum spanning tree of the circuit graph, called the *normal tree*, is generated. The normal tree generator does this by creating a blank graph, and then repetitively adding the edges of the circuit graph in the following priority order until a minimum spanning tree is reached: voltage sources, capacitors, resistors, switches, diodes, gate-to-drain connected MOSFETs, other MOSFETs, inductors, and current sources. The difference between the circuit graph and the normal tree is called the tree links. It is required that all voltage sources are on the normal tree and all current sources are on the tree links. Otherwise, a contradiction (e.g. parallel voltage sources) occurs.

To determine whether elemental models, Kirchhoff's voltage law, or Kirchhoff's current law should be used to generate the explicit equation for a variable, the following rules from [23] are used together with the normal tree and tree links:

- For voltages on the normal tree, use elemental models.
- For currents on the normal tree, use Kirchhoff's current law.
- For voltages on the tree links, use Kirchhoff's voltage law.
- For currents on the tree links, use elemental models.

The generated equations in the form of Eq. (4) are then used to construct an MSFG, as explained in Section IV-B.

E. Reducing the MSFG

The MSFGs of the system-level and SPICE-level descriptions are reduced to a minimal form by the block called "MSFG simplifier" (Fig. 3). The overview of this process is given in Fig. 5 and consists of the simplification rules given below. These are applied until a simplified graph with only one node between its input and output nodes is obtained.

1) Simplification Operations for MSFGs:

a) Node removal with substitution: All nodes except the input nodes, output nodes, and the parent nodes of output nodes are removed during the simplification process. To remove a node $(f_i(\cdot))$ with outgoing variable x_i,

- for every child node $(f_c(x_i, \dots))$ the variable x_i is replaced by $(f_i(\cdot))$,
- for all parent nodes, the edge from the parent node $(f_p(\cdot))$ to $(f_i(\cdot))$ is redirected to go from $(f_p(\cdot))$ to all child nodes $(f_c(x_i, \dots))$,
- the node $(f_i(\cdot))$ and all the edges leaving it are deleted.

b) Reflexive edge elimination: Reflexive edges are edges that come out from and go to the same node. There are two ways to remove a reflexive edge x_i at node $(f_i(x_i, \dots))$. Symbolic solving for the unknown x_i is the first option and is used whenever possible, where equation

$$x_i = f_i(x_i, \dots)$$

is solved for x_i to get

$$x_i = f_{new}(\cdot)$$

where $f_{new}(\cdot)$ does not depend on x_i. The function $f_i(x_i, \dots)$ is then replaced by $f_{new}(\cdot)$ and the reflexive edge is removed. The second option is used whenever symbolic solving is not possible, where $f_{new}(\cdot)$ is approximated with polynomial interpolation such that

$$f_{\text{new}}(\cdot) = \sum_{n=0}^{N} a_n L_n^{(s)}(\cdot)$$

where $L_n^{(s)}(\cdot)$ is defined in the same way as in Section III-A.

The only remaining functions of the SPICE-level and system-level minimal MSFGs are called $f_{\text{spi}_{\text{min}}}(u)$ and $f_{\text{sys}_{\text{min}}}(u)$, where min stands for minimal, u is the input to the MSFG, and *spi* and *sys* stand for SPICE-level and system-level, respectively. Next, we create the canonical MSFG from the minimal MSFG we just obtained and introduce a method to compute similarity.

F. Canonical MSFG and Similarity Metric

To create the canonical MSFGs, the functions $f_{\text{spi}_{\text{min}}}(u)$ and $f_{\text{sys}_{\text{min}}}(u)$ are replaced by their polynomial approximations $f_{\text{spi}_{\text{app}}}(u)$ and $f_{\text{sys}_{\text{app}}}(u)$, as given in Eq. (1). The coefficients c_{spi} and c_{sys} as defined in Eq. (2) can then be used to calculate the similarity by computing their Euclidean distance,

$$d = \|c_{\text{spi}} - c_{\text{sys}}\|_2 \tag{6}$$

and finally, calculating the *similarity metric* as

$$s = 1 - \frac{d}{\|c_{\text{sys}}\|_2} \tag{7}$$

We demonstrate our methodology on our running example in the next section.

G. Illustration

In this section, we will use the amplifier circuit from Fig. 1 to illustrate our methodology. First, the normal tree generator is used to create the circuit's normal tree (bold tree in Fig. 6). It can be validated that this tree is indeed a minimum spanning tree by checking that all nodes of the circuit are also nodes of the normal tree without forming any loops. Also, since the voltage sources and the resistor were included in the normal

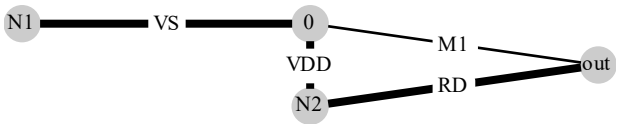

Fig. 6. Graph of the CS amplifier circuit. The normal tree is emphasized with bold edges.

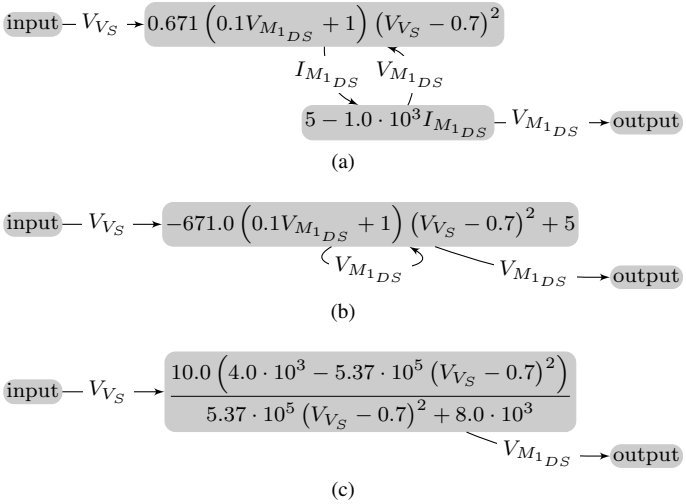

Fig. 7. Some MSFGs of the CS amplifier: **(a)** The MSFG after 5 simplification steps. **(b)** The MSFG after removal of a node from Fig. 7a. **(c)** Minimal MSFG.

tree but the MOSFET was not, it is validated that the priority order as described in Section IV-D was obeyed.

Afterwards, the explicit equations Eq. (8) and Eq. (9) are obtained by using this normal tree and the rules described in Section IV-D. The equations for the components on the normal tree are given in Eq. (8),

$$\begin{aligned} V_{R_D} &= 10^3 I_{R_D}\,, \quad I_{R_D} = I_{M1_{DS}}\,, \\ I_{V_{DD}} &= I_{M1_{DS}}\,, \quad\quad I_{V_S} = 0 \end{aligned} \tag{8}$$

whereas the equations for the components on the tree links are given in Equation (9),

$$\begin{aligned} V_{M1_{DS}} &= V_{V_{DD}} - V_{R_D}\,, \qquad V_{M1_{GS}} = V_{V_S}\,, \\ I_{M1_{DS}} &= 0.671\left(0.1V_{M1_{DS}}+1\right)\left(V_{M1_{GS}}-0.7\right)^2 \end{aligned} \tag{9}$$

Since these equations are in the form of Eq. (4), an MSFG can be created without any further modifications.

The MSFG simplifier then simplifies this MSFG with the rules given in Section IV-E. An example of node removal with substitution is seen in the 6th step of the simplification process, where the node $\left(0.671\left(0.1V_{M1_{DS}}+1\right)\left(V_{V_S}-0.7\right)^2\right)$ in Fig. 7a is removed to get Fig. 7b. The next step is an example of reflexive edge elimination, where the reflexive edge $V_{M1_{DS}}$ is removed, to get the minimal MSFG in Fig. 7c. The numbers in the graphs were printed with low precision to save space.

The system-level MSFG in Fig. 8 can be obtained directly from the system-level model in Fig. 2, since its equation given in Eq. (3) is already in the form of Eq. (4). This MSFG requires no further simplification since it is already minimal.

To check whether these MSFGs are similar, we take their final functions and approximate them with Legendre polynomials. The approximated function for the SPICE-level MSFG is Eq. (10), whereas for the system-level MSFG it is Eq. (11).

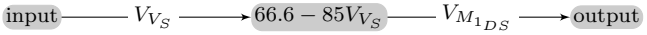

Fig. 8. System-level MSFG of the CS amplifier.

$$\begin{aligned} f_{\text{spi}_{\text{app}}} &= 4L_0^{(s)}(V_{V_S}) - 1.2L_1^{(s)}(V_{V_S}) \\ &\quad - 0.054L_2^{(s)}(V_{V_S}) + 0.01L_3^{(s)}(V_{V_S}) \end{aligned} \tag{10}$$

$$f_{\text{sys}_{\text{app}}} = 4L_0^{(s)}(V_{V_S}) - 1.4L_1^{(s)}(V_{V_S}) \tag{11}$$

The polynomials $L_i^{(s)}$ were scaled for the domain of V_{V_S}, which is $[0.73, 0.77]$ as seen in Section III-B.

The similarity of $f_{\text{spi}_{\text{app}}}$ and $f_{\text{sys}_{\text{app}}}$ is calculated from the Euclidean distance between their coefficients. For this, we first define the coefficient vectors

$$\begin{aligned} c_{\text{spi}} &= \begin{bmatrix} 4 & -1.2 & -0.054 & 0.01 \end{bmatrix} \\ c_{\text{sys}} &= \begin{bmatrix} 4 & -1.4 & 0 & 0 \end{bmatrix} \end{aligned}$$

then, we find the Euclidean distance with Eq. (6) as

$$d = \|c_{\text{spi}} - c_{\text{sys}}\|_2 = 0.223$$

and finally, we calculate the *similarity metric* with Eq. (7) as

$$s = 1 - \frac{d}{\|c_{\text{sys}}\|_2} = 94.76\,\%$$

This high similarity result is expected, since both implementations model the same circuit. On the other hand, a result of less than 100% was also expected, since the SPICE-level model captures nonlinear behavior, whereas the linear system-level model does not. In the next section, we perform more case studies to analyze our methodology further.

V. Experimental Evaluation

To demonstrate the broad applicability of our methodology, and to compare it to previously published methods, we conduct two case studies. First, we use the PLL charge-pump circuit from [16] to test our methodology and compare it to past work. Then, we apply our methodology to a squaring circuit. All computations were conducted on an octa-core AMD Ryzen 7 PRO 4750U with 32 GB RAM.

A. PLL Charge-Pump Circuit

In [16], a SPICE-level model and two system-level models, *Model A* and *Model B*, for a PLL charge-pump circuit were given. Our *SPICE-level MSFG creator* (Fig. 3) produced an MSFG with 32 nodes that required 35 steps to simplify, whereas the system-level models resulted in MSFGs with 3 nodes. All models were approximated with *Legendre polynomials* up to 19 degrees, with the resulting coefficients given in Fig. 9. The similarity to the SPICE-level model was found to be 99.58% for *Model A* and 92.64% for *Model B*. In [16], *Model B* was presented as a simpler and inferior model. Our results confirm this. A summary of the results and a comparison with the reference work are given in Table II.

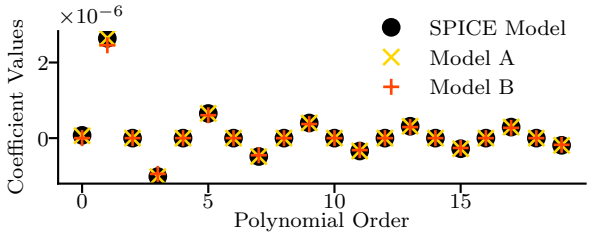

Fig. 9. Coefficients of the polynomial approximations for the PLL charge-pump circuit.

TABLE II
EQUIVALENCE RESULTS FOR THE PLL CHARGE-PUMP MODELS

	Difference		Run Time	
Method	**Model A**	**Model B**	**Model A**	**Model B**
[16]	14.3[a]	21.5[a]	31.6 s	30.4 s
Our work	0.42%	7.36%	1.36 s	

[a] In [16], an absolute difference value is given that cannot be directly compared to our relative values. The values are provided here for reference.

B. Squaring Circuit

As another case study, we tackled the squaring circuit from [26], which has important applications in analog signal processing. The initial MSFG from the SPICE-level circuit was produced with 14 nodes and was simplified in 12 steps. The MSFG created from the system-level model, given as

$$I_o = 4.59 I_{in}^2$$

had 3 nodes. The most significant coefficients were found as

$$c_{\text{spi}} = \begin{bmatrix} 2.3 & 1.9 & 0.42 \end{bmatrix}$$
$$c_{\text{sys}} = \begin{bmatrix} 2.2 & 1.9 & 0.48 \end{bmatrix}$$

and the similarity was found to be 93.88%. We expected this difference since additions in square functions that expand like

$$(x(t) + c)^2 = x(t)^2 + 2c \cdot x(t) + c^2$$

and the feedback interaction between the components prevent the implementation of a pure square function. The run time for this example was 0.32 s.

VI. CONCLUSION

In this study, we developed a novel, graph-based, formal equivalence checking methodology by combining multiple analysis and modification techniques. We then used it to check SPICE-level-to-system-level equivalency for static nonlinear circuits. The running times were short, on the order of seconds, since the methodology is based on static analysis. We quantitatively showed equivalency between the models with the help of a *similarity metric*. We successfully applied the methodology to several case studies.

This paper can be extended in multiple ways. One possibility is to broaden the applicability by changing the approximation method to one that supports dynamic systems. Additionally, determining the necessary number of Legendre polynomials for approximation can be automated. Also, the graph-based representation can be leveraged by using established search methods to pinpoint specific parts of the circuit that cause a mismatch, when the similarity between the models is not 100%. Finally, the manual definition of variable ranges required for polynomial interpolation might be automated with a reachability method.

REFERENCES

[1] M. Barnasconi, "SystemC AMS Extensions: Solving the Need for Speed," *DAC Knowledge center*, May 2010.

[2] L. W. Nagel, "Spice-simulation program with integrated circuit emphasis," *Electronics Research Lab., Univ. of California, Berkeley*, 1973.

[3] C. Grimm, M. Barnasconi, A. Vachoux, and K. Einwich, "An introduction to modeling embedded analog/mixed-signal systems using systemc ams extensions," in *Open SystemC Initiative*, 2008.

[4] M. Barnasconi, C. Grimm, M. Damm, K. Einwich, M. Louërat, T. Maehne, F. Pecheux, and A. Vachoux, "Systemc ams extensions user's guide," *Accellera Systems Initiative*, 2010.

[5] M. Barnasconi, K. Einwich, C. Grimm, T. Maehne, and A. Vachoux, "Advancing the SystemC analog/mixed-signal (AMS) extensions," *Open SystemC Initiative*, 2011.

[6] F. Pêcheux, C. Grimm, T. Maehne, M. Barnasconi, and K. Einwich, "SystemC AMS based frameworks for virtual prototyping of heterogeneous systems," 2018, pp. 1–4.

[7] R. Drechsler, Ed., *Advanced Formal Verification*. Kluwer Academic Publishers, 2004.

[8] P. Molitor and J. Mohnke, *Equivalence checking of digital circuits: fundamentals, principles, methods*. Springer Science & Business Media, 2007.

[9] R. Drechsler, *Formal System Verification*. Springer, 2018.

[10] M. H. Zaki, S. Tahar, and G. Bois, "Formal verification of analog and mixed signal designs: A survey," *Microelectronics Journal*, vol. 39, no. 12, pp. 1395–1404, Dec. 2008.

[11] A. Tarraf, L. Hedrich, N. Kochdumper, M. Rechmal-Lesse, and M. Olbrich, "Equivalence Checking Methods for Analog Circuits Using Continuous Reachable Sets," in *2020 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, Jul. 2020, pp. 7–12.

[12] L. Hedrich and E. Barke, "A formal approach to nonlinear analog circuit verification," in *Proceedings of IEEE International Conference on Computer Aided Design (ICCAD)*, Nov. 1995, pp. 123–127.

[13] L. Hedrich and W. Hartong, "Approaches to Formal Verification of Analog Circuits," in *Low-Power Design Techniques and CAD Tools for Analog and RF Integrated Circuits*. Boston, MA: Springer US, 2001.

[14] W. Hartong, R. Klausen, and L. Hedrich, "Formal Verification for Nonlinear Analog Systems: Approaches to Model and Equivalence Checking," in *Advanced Formal Verification*, R. Drechsler, Ed. Boston, MA: Springer US, 2004, pp. 205–245.

[15] S. Steinhorst and L. Hedrich, "Advanced methods for equivalence checking of analog circuits with strong nonlinearities," *Formal Methods in System Design*, vol. 36, no. 2, pp. 131–147, Jun. 2010.

[16] A. Singh and P. Li, "On behavioral model equivalence checking for large analog/mixed signal systems," in *2010 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Nov. 2010, pp. 55–61.

[17] A. Ain, S. Sanyal, and P. Dasgupta, "A Framework for Automated Feature Based Mixed-Signal Equivalence Checking," in *VLSI Design and Test (VDAT)*, Roorkee, 2017, pp. 779–791.

[18] M. O. Saglamdemir, G. Dundar, and A. Sen, "An analog behavioral equivalence checking methodology for simulink models and circuit level designs," in *International Conference on Synthesis, Modeling, Analysis and Simulation Methods and Applications to Circuit Design (SMACD)*, Istanbul, Sep. 2015.

[19] K. Ç. Coşkun, M. Hassan, and R. Drechsler, "Equivalence Checking of System-Level and SPICE-Level Models of Linear Analog Filters," in *Design and Diagnostics of Electronic Circuits and Systems (DDECS)*, Prague, 2022.

[20] K. Ç. Coşkun, M. Hassan, and R. Drechsler, "Equivalence Checking of System-Level and SPICE-Level Models of Linear Circuits," *Chips*, vol. 1, no. 1, pp. 54–71, Jun. 2022.

[21] B. Razavi, *Design of Analog CMOS Integrated Circuits*. Boston, MA: McGraw-Hill, 2001.

[22] Analog Devices, "Ltspice," https://www.analog.com/en/design-center/design-tools-and-calculators/ltspice-simulator.html.

[23] D. Rowell and D. N. Wormley, *System Dynamics: An Introduction*. Upper Saddle River, NJ: Prentice Hall, 1997.

[24] S. J. Mason, "Feedback Theory-Some Properties of Signal Flow Graphs," *Proceedings of the IRE*, vol. 41, no. 9, pp. 1144–1156, Sep. 1953.

[25] L. P. A. Robichaud, *Signal Flow Graphs and Applications*. Englewood Cliffs, N.J. :, 1962.

[26] N. Beyraghi, A. Khoei, and K. Hadidi, "CMOS design of a four-quadrant multiplier based on a novel squarer circuit," *Analog Integrated Circuits and Signal Processing*, vol. 80, no. 3, pp. 473–481, Sep. 2014.