A novel virtual prototyping methodology for timing-accurate simulation of AMS circuits

A novel virtual prototyping methodology for timing-accurate simulation of AMS circuits

Teo Vallone, Hayri Verner Hasou, Ernesto Colizzi, Sara Vinco and Davide Zoni

***Abstract*—Nowadays, analog-mixed-signal (AMS) circuits are at the core of a large variety of devices targeting automotive, medical, communication, and energy applications. Despite their ubiquity, AMS design methodologies are not automated, rely on long manual iterations, and leverage slow SPICE-level simulation as golden standard. Moreover, SPICE simulations cannot help in verifying timing checks on the digital elements of the circuit, thus allowing possible escaped bugs in the final device. In this scenario, virtual prototypes are used to abstract the modeling of the AMS circuit to boost simulation speed, favor design reuse, and allow early evaluation of the design choices at the cost of a reduced accuracy.**

This paper presents a novel virtual prototyping methodology for AMS circuits. Starting from the netlist of an AMS circuit and the description of the target technology library, the methodology automatically generates the SystemC models for the digital elements, extended with additional timing checks. To deliver the final timing-accurate AMS simulation, the generated SystemC models are then integrated into a co-simulation framework where the analog parts of the circuit are still simulated using SPICE. Experimental results demonstrated the validity of the proposed solution to deliver timing-accurate AMS simulations. The methodology can identify and check five timing violation classes for the digital parts of the circuit that are unchecked at SPICE level.

***Index Terms*—Design automation, ICs, AMS, SPICE, SystemC**

I. Introduction

The ever-increasing demand for advanced mobile communication, automotive, and medical devices fuels the need for modern and complex analog-mixed-signal (AMS) circuits that are at the core of the modern digital world.

Despite their ubiquity, the methodologies to design and verify AMS circuits represent a critical bottleneck. In the past, digital and analog parts have been verified by means of different design flows, while the complexity of modern devices integrating analog and digital functionalities on the same chip impose the use of novel analog-on-top design methodologies for AMS circuit design [1].

On one hand, digital design methodologies are keeping the pace with the continuous evolution of digital circuits offering a vast set of tools, mostly automated, to support the entire design flow. On the other hand, analog design methodologies are not automated for the majority of their design steps and still heavily rely on the expertise of the HW engineers, thus hindering the evolution of analog-on-top design flows [1].

Apart from the lack of automation, analog-on-top methodologies suffer from two other limitations [2]. First, AMS timing verification is achieved through slow SPICE simulations that are, however, considered a golden standard that cannot be neglected. Second, SPICE simulations are not able to reliably represent the possible timing-based issues which are typical of the digital blocks in the AMS designs. If said issues remain unverified throughout the whole design flow, this will likely lead to unexpected behaviors in the final product.

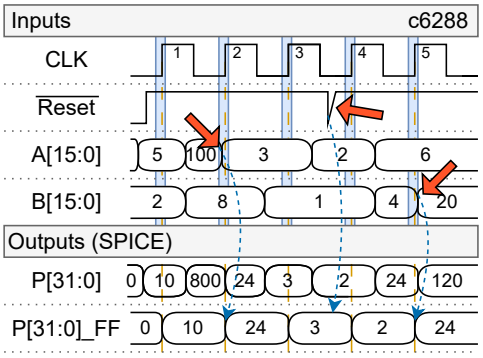


Fig. 1. Examples of timing issues that remain unchecked in the SPICE flow.

An example is provided in Fig. 1, which represents the behavior of the circuit c6288, a 16-bit multiplier from the ISCAS-85 benchmark suite. A and B are the two 16-bit inputs, while P is the 32-bit output. To highlight timing issues, we appended a flip-flop (FF) to each output. We recreated three typical timing issues:

- *setup time violation - (cycle 1)*. By toggling one of the inputs (i.e., the input A) right before the sensitive edge of the clock (i.e., the rising edge), we may not have enough time for the new signal to be correctly sampled by the FF. As a consequence, we won't be perfectly certain on the FF output during the next clock cycle.
- *reset minimum pulse width violation - (cycle 3)*. If the reset input is activated (i.e., pulled to 0) for a very short time, we won't know for certain whether the FF outputs have been brought to the default value (e.g., 0) or not.
- *hold time violation - (cycle 5)*. By toggling one of the inputs (i.e., the input B) right after the sensitive edge of the clock (i.e., the rising edge), the new value may be able to propagate through FFs and reach the output, as the FFs may still be sampling. We'll have another uncertainty, this time during the current clock cycle.

Ideally SPICE should highlight such uncertainties on the outputs. However, simulations show that each output converges to either '1' or '0', though this doesn't mean that we'll reliably

obtain the same results every time, as long as we keep violating these timing constraints.

To solve the above-mentioned limitations, Virtual Prototypes (VPs) at the Electronic System Level (ESL) of abstraction emerged as a viable and estabilished industrial practice. VPs leverage SystemC (and its extensions, e.g. SystemC-AMS) to describe the functional level of the AMS circuit offering faster simulations, design reuse, and early stage abstract models of the AMS device to support preliminary analysis and novel design concepts [3], [4]. However, current virtual prototyping methodologies work at high level, thus offering behavioral circuit models without accurate timing checks. Moreover, to the best of our knowledge, there is no methodology allowing to automatically abstract a SPICE level component into a VP, thus preventing the full reuse of production-grade intellectual properties (IPs) to shape new device concepts.

Contributions - This work presents a novel virtual prototyping design methodology for AMS circuits with four main contributions to the state of the art:

- *Automatic SystemC accurate models for digital SPICE-level netlists*. Starting from the Liberty timing specification file (Lib) of the technology library and the SPICE netlist of the AMS circuits, the methodology creates a timing accurate SystemC model for each digital block of the AMS circuit, plus the top level SystemC code instantiating and binding them.
- *Accurate timing checks for digital element*. For each SystemC model of the digital part, the methodology automatically creates timing checks, leveraging the information in the Liberty timing specification file.
- *Accurate co-simulation of AMS circuits*. The AMS circuits are co-simulated by using SPICE and SystemC simulators for the analog and the digital parts, respectively. This solution supports accurate timing checks for the digital elements, while still ensuring SPICE-level accuracy for the analog parts of the circuit.
- *Reproducible results*. The entire methodology and the related test-cases are publicly available for repeatability and to foster further investigations and extensions.

Structure of the manuscript - The paper is organized in four parts. Section II reviews the state of the art targeting the virtual prototyping methodologies to improve the accurate evaluation at early stage of analog and mixed signal circuits. Section III discussed the proposed virtual prototyping methodology. Section IV presents the experimental results. Conclusions and future works are drawn in Section V.

II. State of the art

The state of the art contains several contributions to improve the standard AMS design flow targeting specific and difficult design steps, e.g., verification, circuit abstraction, and automatic place and route. However, to the best of our knowledge, none of the proposed solutions targets a SystemC-SPICE co-simulation framework where the automatically generated SystemC models of the digital parts of the circuit implement accurate timing checks.

[5] presents an automated place and route flow for analog-mixed signal designs that leverages the standard digital layout tools. Each atomic analog cell is wrapped to be used within the digital design flow to automatically generate the place and route for the AMS circuit. The experimental results, which include a VCO and a strongARM comparator, demonstrated that the solution allows to speed up the early stage layout evaluation for AMS circuits with relaxed constraints. [6] proposes a solution to automatically generate the top-level for AMS circuits. Starting from a list of fully characterized cells, the engineer declares the interconnections. Then, the methodology automatically generates the netlist for the circuit's top-level. [7] and [8] propose a reinforcement learning-based algorithm to synthesize analog IC topologies. Similarly to the RTL synthesis in digital circuits, the methodologies create the topology of the AMS circuits starting from the design constraints and the target AMS technology library. [9] presents a machine learning algorithm that receives a generic AMS circuit and the corresponding technology library to deliver a feasible set of parameters for the circuit fulfilling its design constraints in terms of area and performance.

[10] presents a system-level verification methodology for AMS circuits. To enhance the predictability of the functional verification, the methodology performs a sensitivity analysis considering different corners.

[11] aims to optimize the leakage power due to spare cells. Starting from the technology library, the methodology leverages the state-dependent information of such cells to determine the set of input values which lead to the lowest dissipation. [12] proposes a static timing analysis (STA) tool for specific quantum digital circuits starting from a SPICE topology and the timing information from the library. [13] presents a methodology to characterize full custom cells considering the size of the transistors in each cell. The tool works at SPICE level by extracting the equivalent RC network for each cell.

Several state-of-the-art contributions highlight the importance of using virtual prototypes to model AMS circuits [3], [14]–[16]. However, all the works aims to use SystemC-AMS to deliver a high-level abstraction of complex AMS circuits for early stage evaluation, design concept creation and evaluation without considering the possibility of automatically abstracting complex SPICE-level netlists in SystemC to support advanced verification techniques.

III. Methodology

Fig. 2 outlines the proposed methodology. The starting point (left) are a netlist description of an AMS system and a library of Liberty files, i.e., files describing standard cell libraries. The inputs are processed as follows:

1) the Liberty files and the netlist are used to generate the SystemC implementation of the digital part, made of:

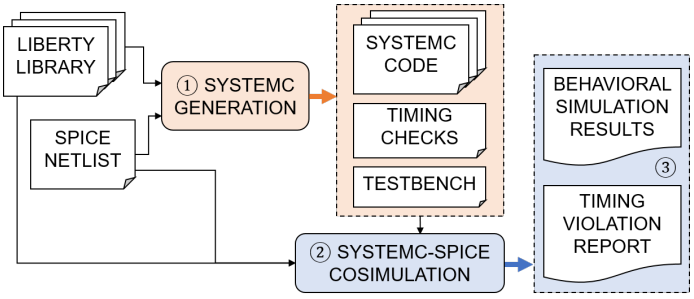


Fig. 2. Proposed flow.

- cells instantiation as automatically generated SystemC modules, extended with timing checks to identify timing violations;
- automatically generated SystemC top level, reproducing the block's topology;
- a simulation testbench, generated by the user through a manually configured GUI;

2) the SystemC code is either simulated by itself, in case we're only interested in analyzing the digital part, or co-simulated with SPICE, used for the analog portion, through state-of-the-art solutions;
3) the simulation generates behavioral simulation results, augmented with a timing violations report.

A. Input files

The first input to the methodology is a *library of Liberty files* describing the characterization of the standard cells [17]. An example of Liberty file is sketched in Fig. 3.b. Each Liberty file describes a specific PVT (Process, Voltage, Temperature) corner of operation, to test and simulate the circuits both in nominal conditions and under the most extreme application-specific circumstances (lines 4-5).

The file contains general information, such as the units of measurement of the cells' parameters (line 3), as well as the standard cells' descriptions. Each cell is organized in a hierarchical structure, including both general information, like area occupation, and pin-specific data like equivalent capacitance values or the output's boolean function. In addition, we also have multi-dimensional matrices describing relevant parameters, like power and timing, based on other variables in the circuit, such as the output capacitive load (e.g., a 2-D matrix for fall transition times, lines 12-13). This allows us to better reflect the actual post-silicon implementation compared to having a singular scalar value. Fig. 3.a shows the schematic of the cell described at lines 6-13 of Fig. 3.b.

The second input is a *netlist file* (Fig. 4.a), that contains a high-level view with one or multiple connected blocks. The file begins with transistor-level descriptions for the standard cells (lines 1-5), and continues with the blocks' gate-level topologies, i.e. the delineation of how the instantiated cells are connected within the block itself (lines 7-8).

B. SystemC code generation

1) Cell instantiation as SystemC modules: Given the initial Liberty library, the user chooses the Liberty file of interest, reflecting the desired operating conditions. The chosen Liberty file is split into sections, each corresponding to the description of a standard cell. Such sections are then used to generate *one SystemC module per cell* (Fig. 3.c) as follows:

- the cell name becomes the name of the SC_MODULE, to maintain a tight correspondence (line 1);
- cell pins become input ports (sc_in) or output ports (sc_out), depending on their direction tag. The name of the pin is used as name of the corresponding port. Any parameter of a pin is used in the constructor of the SC_MODULE, e.g., to keep track of port capacitance (lines 2-6);
- the logic function associated to each output port is used to populate a corresponding $logic\_fun()$ function, by converting pin expressions into operations on input and output ports (line 10). The $logic\_fun()$ function is executed as a method in case of any update of the input pins of the SC_MODULE (i.e., A, B or C).

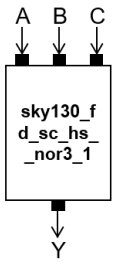
In order to preserve timing information on the signals, the $logic\_fun()$ function does not write directly on the cell output ports, but rather on internal signals (i.e., out, line 10). The value written on such signals is then propagated to the output port (i.e., Y) by respecting the rise and fall delays specified in the Liberty file (lines 11-12), through the activation of a second SystemC process (omitted in the figure). This allows to keep track of the timing information also in the higher level SystemC code.

2) Top level generation: The subsequent step is the generation of the top-level SystemC equivalent view, in charge of instantiating the cells and of reproducing their topology. An excerpt is reported in Fig. 4.b.

The necessary information is extrapolated from the netlist file. The methodology performs an initial filtering action to consider only blocks made entirely from cells contained in the Liberty file: any other component, such as a resistance, a capacitance or a transistor, would be translatable to SystemC, but it would not allow to perform a proper timing analysis.

The netlist provides the topology of the blocks, in terms of instantiated cells and their connection through nets. This allows to generate the SystemC top level as a number of SystemC modules (i.e., the used standard cells) connected via signals (i.e., the network connections described by the netlist).

In detail, the terminals of the selected block are used to generate the input and output ports of the SystemC top level: this allows to receive stimuli from a testbench and simulate the system. Then, for each cell used in the block's section of the netlist, the top level instantiates the corresponding SystemC module (lines 2-3). In the netlist file, the connections (nets) between ports of different cells are described using a position-based system: the first net/pin in an instantiated cell will be connected to the first pin that was stated in the cell declaration (e.g., in Fig. 4.a, $net\_\_235$ is associated to pin B of the



a.

```
1.  library ("sky130_fd_sc_hs__tt_025C_1v80") {
2.    technology("cmos");
3.    capacitive_load_unit(1.0,"pf"); …
4.    operating_conditions ("typ") {
5.      voltage : 1.8; temperature : 25.0; … }
6.    cell ("sky130_fd_sc_hs__nor3_1") {
7.      pin ("A") { capacitance:0.00292; direction:"input"; …}
8.      pin ("B") { capacitance:0.00308; direction:"input"; …}
9.      pin ("C") { capacitance:0.00289; direction:"input"; …}
10.     pin ("Y") { direction:"output"; function:"(!A&!B&!C)";…}
11.   timing () {
12.     cell_fall ("delay_template13x16") {
13.       values(" …,49.9,…,221.1,…);}
```

b. LIBERTY FILE

```
1.  SC_MODULE(sky130_fd_sc_hs__nor3_1){
2.    sc_in<sc_logic> A; sc_in<sc_logic> B; sc_in<sc_logic> C;
3.    sc_out<sc_logic> Y;
4.    …
5.    struct states{
6.          double a_cap = 0.00292;
7.          sc_time t_fall_a = sc_time(49.9,SC_PS);
8.          sc_time t_rise_a = sc_time(221.1,SC_PS); … }
9.    void logic_fun(){
10.     out = (~A.read()&~B.read()&~C.read());
11.     if (A.event() && A.read()==0) evt.notify(s.t_fall_a);
12.     if (A.event() && A.read()==1) evt.notify(s.t_rise_a);
13.} … } … }
```

c. SYSTEMC MODULE DECLARATION FOR CELL sky130_fd_sc_hs__nor3_1

Fig. 3. Schematic (a), excerpt of Liberty file (b) and corresponding generated SystemC module (c) for cell $sky130\_fd\_sc\_hs\_\_nor3\_1$.

```
1.  .SUBCKT sky130_fd_sc_hs__nor3_1 A B C Y VGND VNB VPB VPWR
2.  MMP0 VPWR A sndPA VPB pfet_01v8 m=1 w=1.12 l=0.15 mult=1 sa=0.265
3.  + sb=0.265 sd=0.28 topography=normal area=0.063 perim=1.14
4.  …
5.  .ENDS sky130_fd_sc_hs__nor3_1
6.  …
7.  X328 A__15 net__235 net__318 … sky130_fd_sc_hs__nor3_1
8.  X329 B__0 A__15 net__235 net__319 … sky130_fd_sc_hs__a31oi_1
9.  …
```

a. NETLIST FILE

```
1.  SC_MODULE(TopLevel6288){
2.    struct components{
3.      sky130_fd_sc_hs__nor3_1& i_sky130_fd_sc_hs__nor3_1_X328;
4.      sky130_fd_sc_hs__a31oi_1& i_sky130_fd_sc_hs__a31oi_1_X329;
5.      sc_core::sc_signal<sc_logic>& net235;
6.      …
7.      i_sky130_fd_sc_hs__nor3_1_X328 -> B(net__235);
8.      i_sky130_fd_sc_hs__a31oi_1_X329 -> A3(net__235);
9.      … } }
```

b. SYSTEMC TOP LEVEL FILE

Fig. 4. Excerpt of netlist file (a) and corresponding generated top level SystemC code (b).

standard cell $sky130\_fd\_sc\_hs\_\_nor3\_1$). To reproduce such binding, the nets used for pin connection are instantiated as SystemC signals (sc_signal, line 5), and used for port binding of the corresponding SystemC modules (lines 7-8).

3) Timing checks generation: The flexibility and modularity of SystemC is exploited to insert timing checks in the stardard cells. The SystemC modules of specific standard cells, e.g., flip flops, are extended to collect the time of reception of any update on each input to detect timing violations. In the case of flip flops, this allows us to check, e.g., whether the data input D toggled too close to the sensitive clock edge, thus not being stable for a sufficient duration before being sampled.

4) Testbench generation: The last step to allow simulation of the SystemC code is the generation of a testbench, providing stimuli to the overall topology. The testbench consists of a sc_main file used to instantiate the top level and to run the simulation. The testbench declares one signal per each port of the top level, to have a complete binding of the ports of the latter. Then, it starts the simulation and changes the values of the top level input ports over time.

The generation of such stimuli can be hand-written by the designer. However, to ease simulation setup, we provide a support GUI that allows the designer to describe the inputs as bit-toggles described as:

- for non-periodic signals, couples of (time, value);
- for periodic signals, as four parameters: first edge, number of periods, time at 1, time at 0.

The GUI is supported by an automatic tool, that generates the corresponding SystemC code. The output SystemC code is a sequence of invocations of sc_start, that starts simulation for a given amount of time until the next bit toggle, followed by updates of the top level inputs.

In parallel to this, the script will also generate the equivalent testbench in SPICE, where instead the inputs are considered as separate flows and where each time instance refers to the start of simulation, unlike SystemC-based testbenches. This allows to easily compare the results and highlight the differences between the two solutions.

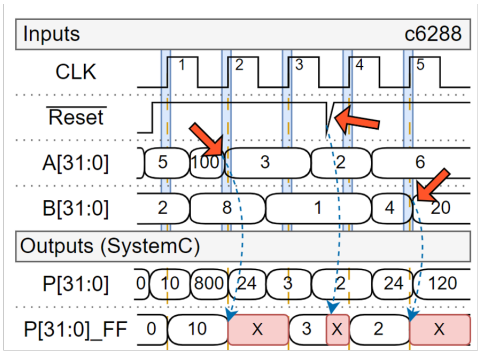
C. SystemC-SPICE cosimulation results

The cosimulation of SystemC and SPICE is a problem that has been discussed and resolved in research with the integration of the SystemC solver with SPICE-based tools [18], [19]. Thus, we do not propose any novel integration flow, but rather rely on available co-simulation support.

The result of the co-simulation are traces of the signal of interest in a Value Change Dump (.vcd) file format. The identified timing violations are then annotated in a report that displays which signals and which cells are involved in each issue, as well as the total slack of the operation and the type of violation: an example of both results is portrayed in Fig. 5, which replicates the same function as Fig. 1 in a SystemC environment. The red arrows in the traces (top of the figure) represent timing violations, that are detailed in the timing violation report (bottom).

IV. Experimental Analysis

This section discusses the assessment of the proposed methodology: we will first describe the libraries, benchmarks




```
TIMING VIOLATION REPORT
Source | Destination | Slack (ns) | Violation
-----------------------------------------------------------
...
A[2] | P[3]_FF | -1.562 | Setup
...
Reset | P[0]_FF | -0.383 | MPW
...
B[4] | P[6]_FF | -0.711 | Hold
...
```

Fig. 5. Outputs of the SystemC-SPICE co-simulation: signals traces (top) and timing violation report (bottom). The red arrows on the traces highlight the occurred timing violations, that are detailed in the report.

and general setup steps in subsection A, followed by an in-depth analysis of the results, including a brief description of each timing scenario in subsection B.

A. Experimental setup

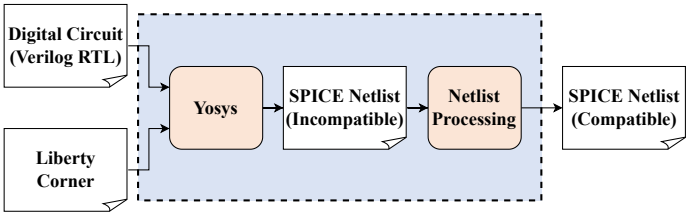


Fig. 6. Top-level view of the setup flow.

To work correctly, out tool needs a complete standard cells library (.lib file), either a single one or a collection of corners, as well as a compatible netlist that includes one or more fully digital blocks. With compatible we mean that each and every cell in the netlist must be associated to one of the cells from the library description, as otherwise we won't have any way to generate a timing and functional-accurate model.

We resorted to generating such netlists semi-automatically starting from high-level behavioral models written in Verilog. This allowed to map directly each cell to the most appropriate one from our libraries during the synthesis process.

As cell library we adopted the skywater130 library [20]–[22], in particular the *sky130-fd-sc-hs* corner with a voltage supply of 1.8V measured at a temperature of 25°C. As Verilog models we used the ISCAS-85 and 74-X series benchmarks suites, developed in [23], [24]. Table II reports a list of the circuits that were used, as well as their I/O and the number of cells included, both pre- and post-synthesis.

As synthesis tool, we used the open-source Yosys tool (OSS CAD SUITE [25]) to create a full set of SPICE netlists that were mostly compatible with our tools. The setup flow is shown in Fig 6, where the additional "processing" step consists of some slight adjustments to the netlist, such as adding a transistor-level declaration for each of the standard cells that were used, to make it fully compatible and ready to be used.

To simulate the SystemC code, we used the COSIDE [26] environment. The mixed-signal co-simulations were also performed within COSIDE thanks to the possibility of instantiating an NGSPICE-based netlist within the environment itself.

TABLE I
PRIMARY TIMING CHECKS IN A DIGITAL DESIGN

Scenario	Detected timing check		Figure no.
	SystemC	Spice	
Setup Time	✓		7
Hold Time	✓		8
Reset Recovery	✓		9
Reset Removal	✓		10
Reset MPW	✓		11

B. Experimental results

Table I reports the complete taxonomy of the timing checks that are supported by the proposed methodology and for which the SPICE simulator does not offer any support. For each timing check, the corresponding figure indexed in Table I details a representative example, obtained from the collected experimental results from the ISCAS-85/74-X benchmarks. Each example is organized in three vertically stacked timing diagrams, where each part reports a specific set of signals of the circuit:

- the `Input` part reports the signals that provoke the timing violations;
- the `SystemC` and the `SPICE` parts report the timing diagrams of the signals that exhibit a wrong behavior due to the timing violation.

The `SystemC` and the `SPICE` parts report the same set of signals. The timing diagrams of the `SystemC` part are obtained by means of a SystemC simulation employing the proposed methodology. In contrast, the timing diagrams of the `SPICE` part are obtained by means of classic SPICE simulations.

In order to highlight the timing violations, we added a chain of two flip-flops to each output of each circuit, as all circuits originally were fully combinatorial. We named `Q0` the output of the first FF and `Q1` the output of the second FF, whose input is `Q0`. The string after the "_" represents the combinatorial output, which corresponds to the input of the first FF `D`.

Setup time - Figure 7. This scenario considers the `c432` circuit and the setup timing violation of the synchronous flip-flop, as shown on its output `Q0_PB`. At the end of period `11` of

TABLE II
BENCHMARKS

Benchmark Suite	ISCAS-85										74-X Series			
Benchmark Name	c432	c499	c880	c1355	c1908	c2670	c3540	c5315	c6288	c7552	74181	74182	74283	74L85
Inputs [Bits]	36	41	60	41	33	233	50	178	32	207	9	9	14	11
Outputs [Bits]	7	32	26	32	25	140	22	123	32	108	4	5	8	3
Gates (Pre-Yosys) [#]	160	202	383	546	880	1,193	1,669	2,406	2,406	3,512	19	36	61	33
Gates (Post-Yosys) [#]	89	149	147	154	161	232	438	640	838	727	12	12	37	19
Flip Flops [#]	7	32	26	32	25	140	22	123	32	108	4	5	8	3

the CLK signal, the input to the sequential element violates the setup time (see D_PB). The timing violation impacts the value stored in the flip-flop and, consequently, its output for which the proposed methodology drives an X value (see Q0_PB of the Output (SystemC) section). The undefined value is shown to flow from Q0_PB to Q1_PB in the clock cycle 13. In contrast, the timing diagrams obtained by means of the classic SPICE simulation do not highlight the violation (see Q0_PB and Q1_PB of the Output (SPICE) section).

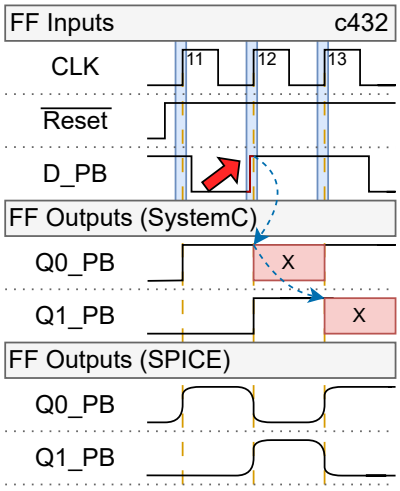


Fig. 7. Circuit c432 from ISCAS-85 benchmark suite with a setup time violation of the signal D_PB. Simulations obtained through COSIDE co-simulation (section FF Outputs (SystemC)) and through a pure SPICE solution (section FF Outputs (SPICE)).

Hold time - Figure 8. This scenario considers the c880 circuit and the hold timing violation of the synchronous flip-flop, as shown on its output Q0_F[5]. At the beginning of period 75 of the CLK signal, the input to the sequential element violates the hold time (see D_F[5]). The timing violation impacts the value stored in the flip-flop and, consequently, its output for which the proposed methodology drives an X value (see Q0_F[5] of the Output (SystemC) section). Similarly to the setup timing violation, the undefined value for Q0_F[5] propagates into Q1_F[5] in clock cycle 76. In contrast, the timing diagrams obtained by means of the classic SPICE simulation do not highlight the violation (see Q0_F[5] and Q1_F[5] of the Output (SPICE) section).

Reset recovery - Figure 9. This scenario considers the c5315 circuit and the reset recovery timing violation of the synchronous flip-flops, as shown on their outputs Q0_OP4

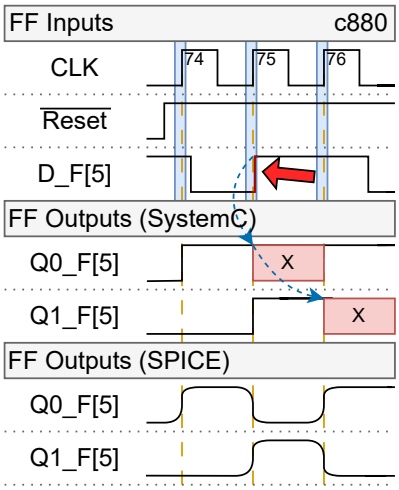


Fig. 8. Circuit c880 from ISCAS-85 benchmark suite, hold time violation of the signal D_PB. Simulations obtained through COSIDE co-simulation (section FF Outputs (SystemC)) and through a pure SPICE solution (section FF Outputs (SPICE)).

and Q1_OP4. The reset signal changes changes at the end of period 33 of the CLK signal, thus violating recovery time (see $\overline{Reset}$). The timing violation impacts the values stored in every flip-flop connected to the same clock and reset signals and, consequently, it impacts their outputs for which the proposed methodology drives an X value (see Q0_OP4 and Q1_OP4 of the Output (SystemC) section). The undefined value driven by Q0_OP4 propagates into Q1_OP4 in the clock cycle 35. In contrast, the timing diagrams obtained by means of the classic SPICE simulation do not highlight the violation (see Q0_OP4 and Q1_OP4 of the Output (SPICE) section).

Reset removal - Figure 10. This scenario considers the 74181 circuit and the reset removal timing violation of the synchronous flip-flops, as shown on their outputs Q0_Y and Q1_Y. The reset signal changes at the beginning of period 21 of the CLK signal, thus violating the removal time (see $\overline{Reset}$). The timing violation impacts the values stored in every flip-flop connected to the same clock and reset signals and, consequently, it impacts their outputs for which the proposed methodology drives an X value (see Q0_Y and Q1_Y of the Output (SystemC) section). The undefined value driven by Q0_Y propagates into Q1_Y in the clock cycle 22. In contrast, the timing diagrams obtained by means of the classic SPICE simulation do not highlight the violation (see Q0_Y and Q1_Y of the Output (SPICE) section in Figure 10).

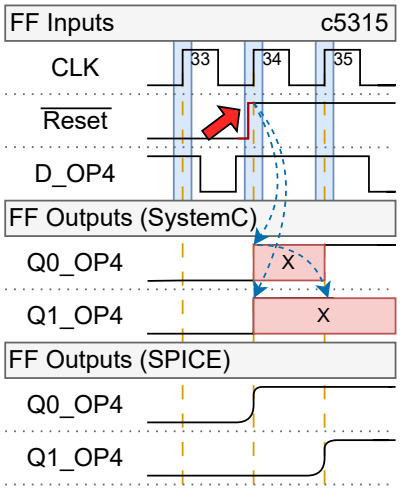


Fig. 9. Circuit `c5315` from ISCAS-85 benchmark suite, `reset recovery` time violation of the signal $\overline{Reset}$. Simulations obtained through COSIDE co-simulation (section `FF Outputs (SystemC)`) and through a pure SPICE solution (section `FF Outputs (SPICE)`).

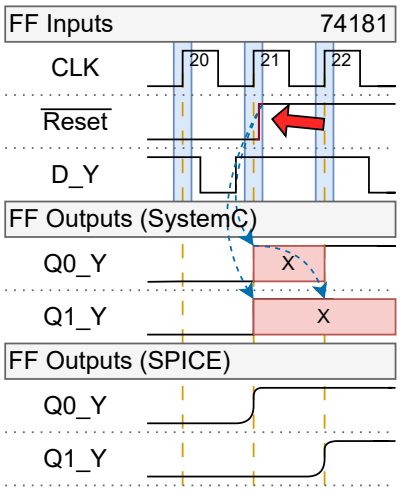


Fig. 10. Circuit `74181` from 74-X series benchmark suite, `reset removal` time violation of the signal $\overline{Reset}$. Simulations obtained through COSIDE co-simulation (section `FF Outputs (SystemC)`) and through a pure SPICE solution (section `FF Outputs (SPICE)`).

Reset minimum pulse width (MPW) - Figure 11. This scenario considers the `74L85` circuit and the MPW timing violation of the synchronous flip-flops, as shown on their outputs `Q0_A>B` and `Q1_A>B`. Considering period `731` of the `CLK` signal, the pulse on the reset signal violates the MPW (see $\overline{Reset}$). The timing violation impacts the values stored in every flip-flop connected to the same reset signal and, consequently, it impacts their output for which the proposed methodology drives an `X` value (see `Q0_A>B` and `Q1_A>B` of the `Output (SystemC)` section). Notably, the output of the flip-flops changes immediately after the MPW timing violation. The undefined value driven by `Q0_A>B` propagates into `Q1_A>B` in the clock cycle `732`. In contrast, the timing diagrams obtained by means of the classic SPICE simulation do not highlight the violation (see `Q0_A>B` and `Q1_A>B` of the `Output (SPICE)` section in Figure 11).

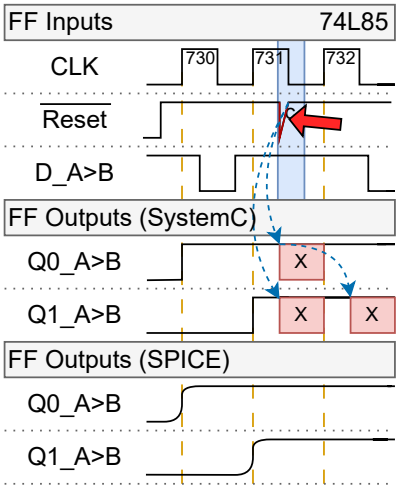


Fig. 11. Circuit `74L85` from 74-X series benchmark suite, `reset MPW` time violation of the signal $\overline{Reset}$. Simulations obtained through COSIDE co-simulation (section `FF Outputs (SystemC)`) and through a pure SPICE solution (section `FF Outputs (SPICE)`).

V. Conclusions

This paper presented a novel virtual prototyping methodology to support the design and simulation of AMS circuits. Starting from the description of the target technology library and the SPICE-level netlist of the design, the proposed methodology automatically generates timing-accurate SystemC models for the digital parts of the circuit. In addition, specific SystemC timing checks are generated as in traditional digital design methodologies. A complete virtual prototyping platform is created to deliver a timing-accurate simulation of the AMS design where the digital and the analog parts are executed at SystemC and SPICE-level, respectively.

The experimental results demonstrated the validity of the proposed solution, both in generating simulatable SystemC code and in identifying timing violations that were not detected by the SPICE simulations, where no timing checks are available for the digital elements.

Notably, we make the entire methodology and the related test cases available to support its adoption while also fostering further investigation and improvements, some of which may include:

- an early stage power consumption analysis of the digital blocks based on the Liberty data;
- even more accurate cell models that keep track of additional topological information (e.g., input transition time, output capacitive load) to determine the most precise values for the signal delays.

References

[1] J. Scheible, "Optimized is not always optimal - the dilemma of analog design automation," in *Proc. of ISPD*, 2022, p. 151–158.

[2] P. B. Rock Z. Shi and, P. Birdsong, G. Chaitanya, and K. Jani, "Mixed-signal design verification: Leveraging the best of AMS and DMS," *Proc. of DVCON*, 2022.

[3] F. Pêcheux, C. Grimm, T. Maehne, M. Barnasconi, and K. Einwich, "SystemC AMS based frameworks for virtual prototyping of heterogeneous systems," in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2018, pp. 1–4.

[4] M. Lora, S. Vinco, E. Fraccaroli, D. Quaglia, and F. Fummi, "Analog models manipulation for effective integration in smart system virtual platforms," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 2, pp. 378–391, 2018.

[5] P.-H. Wei and B. Murmann, "Analog and mixed-signal layout automation using digital place-and-route tools," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 29, no. 11, pp. 1838–1849, 2021.

[6] J. Wittmann, C. Wegener, and F. Rigoni, "Automatic analog-on-top chip-level schematic generation based on wire-by-name methodology juergen wittmann, carsten wegener,," *GMM/ITG ANALOG*, pp. 1–5, 2018.

[7] Z. Zhao and L. Zhang, "Analog integrated circuit topology synthesis with deep reinforcement learning," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 12, pp. 5138–5151, 2022.

[8] ——, "Deep reinforcement learning for analog circuit structure synthesis," *Proc. of Design, Automation & Test in Europe Conference*, pp. 1157–1160, 2022.

[9] K. Settaluri, A. Haj-Ali, Q. Huang, K. Hakhamaneshi, and B. Nikolic, "AutoCkt: Deep reinforcement learning of analog circuit designs," *Proc. of Design, Automation & Test in Europe Conference*, pp. 490–495, 2020.

[10] C. Zivkovic and C. Grimm, "Verification of analog/mixed-signal systems with aadd," *GMM/ITG ANALOG*, pp. 1–6, 2018.

[11] B. V. P. V. Kumar, N. S. M. Sharma, K. L. Kishore, and N. Goel, "Leakage power recovery in spare cells by using state dependent leakage tables from library models," *Proc. of Asia Pacific Conference on Postgraduate Research in Microelectronics and Electronics*, pp. 19–24, 2012.

[12] J. A. Delport and C. J. Fourie, "A static timing analysis tool for RSFQ and ERSFQ superconducting digital circuit applications," *IEEE Transactions on Applied Superconductivity*, vol. 28, no. 5, pp. 1–5, 2018.

[13] J. Lee, J. Jung, and Y. Shin, "Fast timing analysis of transistor-level full custom digital circuits," in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2018, pp. 1–4.

[14] M. Barnasconi and S. Adhikari, "Invited: ESL design in SystemC AMS," *Proc. of ACM/EDAC/IEEE Design Automation Conference (DAC)*, pp. 1–5, 2017.

[15] F. Frank and R. Weigel, "Co-simulation of spice netlists and vhdl-ams models via a simulator interface," in *2007 International Symposium on Signals, Systems and Electronics*, 2007, pp. 75–78.

[16] M. Hassan, D. Große, T. Vörtler, K. Einwich, and R. Drechsler, "Functional coverage-driven characterization of rf amplifiers," *Proc. of Forum for Specification and Design Languages (FDL)*, pp. 1–8, 2019.

[17] A. Mishchenko, *Liberty Reference Manual*, https://people.eecs.berkeley.edu/ alanmi/publications/other/liberty07_03.pdf, 2007.

[18] T. Kirchner, N. Bannow, and C. Grimm, "Analogue mixed signal simulation using Spice and SystemC," in *Proc. of the DATE 2009*, 2009, p. 284–287.

[19] Y. Zaidi, C. Grimm, and J. Haase, "Analog behavior refinement in system centric modeling," in *IEEE Behavioral Modeling and Simulation Workshop 2009*, 2009, pp. 31–36.

[20] "Skywater130 library documentation." [Online]. Available: https://skywater-pdk.readthedocs.io/en/main/contents/libraries.html

[21] "Skywater130 libraries and netlists." [Online]. Available: https://github.com/google/skywater-pdk

[22] "Openroad suite." [Online]. Available: https://github.com/The-OpenROAD-Project/OpenROAD

[23] M. Hansen, H. Yalcin, and J. Hayes, "Unveiling the ISCAS-85 benchmarks: a case study in reverse engineering," *IEEE Design & Test of Computers*, vol. 16, no. 3, pp. 72–80, 1999.

[24] "ISCAS 74-x benchmark suites." [Online]. Available: https://web.eecs.umich.edu/ jhayes/iscas.restore/benchmark.html

[25] "Yosys suite version 2023-10-06." [Online]. Available: https://github.com/YosysHQ/yosys

[26] "Coside environment (version 3.0)." [Online]. Available: https://www.coseda-tech.com/coside-overview