# FREE SOFTWARE FOR ANALOG AND DIGITAL DESIGN

Dušan N. Grujić[1]

1: School of Electrical Engineering, University of Belgrade, 11000, Belgrade, Serbia, e-mail: dusan.grujic@etf.rs

## Abstract

This paper presents an overview of free and open-source software tools for analog and digital design, with emphasis on integrated circuit design. Software tools are essential to harness the full potential of modern CMOS processes, and to enable the design of increasingly complex chips in the first place. Features and limitations of existing software tools are reviewed, as well as missing tools and features important for complex chip design. Some thoughts on improvement of existing, and arguments for development of new, tools are given. Most of the suggested improvements and development of new tools refer to analog design flow because it is not complete, and in its current form cannot be used for the design of high-performance chips. A major obstacle in using free and open-source software tools for chip design is the lack of support from foundries. Problems with foundry-provided process design kits in proprietary formats are discussed, and possible solutions are suggested.

**Keywords:** [open-source, EDA, integrated circuits, analog, digital. ]

## 1 Introduction

Few years after the very first monolithic integrated circuit was made by Robert Noyce in 1959, Gordon Moore has made a prediction in 1965 that complexity will exponentially increase ”at least for 10 years”. Exponential trend has been sustained for more than 50 years, resulting in unparalleled growth of complexity and performance of integrated circuits. It has been a long journey from humble 2250 transistors in Intel 4004, designed in 1971, to tens of billions of transistors in 2021, an increase in number of transistors of more than seven orders of magnitude in 50 years!

Early on, increasing complexity of integrated circuits was welcomed - more transistors could be integrated into a single chip, reducing the overall system size, cost and power consumption. Exponential growth eventually became a problem, as number of transistors overwhelmed engineers, and it became clear that new methodologies and tools are needed. New design methodologies were needed to cope with exponentially increasing chip complexity and to take full advantage of CMOS scaling. Software tools were needed to help engineers focus on design, and automate it as much as possible. Automation was important not just to free engineers from tedious and error-prone tasks, but to make the design and verification of complex chips possible at all.

In late ’70s Conway and Mead have observed that all CMOS design rules, across process nodes and different foundries, can be expressed in terms of integer multiples of lambda units, where lambda is usually half of the minimum feature size [1]. Introduction of lambda-based rules has enabled the design of scalable, process independent, designs and has triggered a VLSI revolution.

Cells designed with lambda-based design rules are not optimal, but are portable across foundries and process nodes. They can be designed once and reused in any CMOS process - Conway formulated that such designs have a ”timeless quality”. Due to phenomenal pace of CMOS process development

in 80s and 90s, in many cases better performance could be achieved by porting a lambda-based design to a new process by simple geometry scaling, than optimizing for an existing process.

Another big step in VLSI revolution was separation of chip design and manufacture. Semiconductor foundries, as they were first called by Mead, are providing manufacturing services to fabless chip design companies. The importance of this concept can be seen from the fact that most of the companies in the semiconductor business are fabless. An excellent read on VLSI revolution is Lynn Conway's personal accounts of events that triggered VLSI revolution is [1].

Fast forward forty years into 2021 many things have changed. Cost and complexity of advanced CMOS nodes has has left only a few foundries following Moore's law to 10 nm and below. Other foundries have stopped development of new nodes, trying to enhance the yield and enrich existing processes with new devices - a concept which is sometimes called "more than Moore".

Demand for lowest chip area and power consumption, increasingly complex design rules, and slower pace of process development have made lambda-based cell design less attractive. Instead of using scaled cells in all process nodes, foundries usually provide a library of digital cells optimized for a given process. Digital designs are synthesized from description in a hardware description language (HDL), such as Verilog or VHDL, or some higher level of abstraction. In a sense, not much has changed, only the "timeless quality" of a design has shifted from lambda-based layout to a more abstract representation in HDL.

Complexity of analog designs has also increased over time - not as dramatic as in digital designs - but still in orders of magnitude. Most CMOS processes are optimized primarily for digital circuits, and analog designs have to "fit in". This creates unique challenges for analog designer because analog/RF circuits don't always benefit from process scaling. Scaled processes offer transistors with higher operating frequency, but at reduced supply voltage and lower intrinsic gain, having detrimental effect on analog performance. In RF designs, area of integrated inductors does not scale with process at all because inductance is determined solely by geometry of windings.

Software tools are essential to harness the full potential of advanced CMOS processes and cope with design complexity. Following sections present an overview of FOSS tools for analog and digital design flow. Overview covers only a portion of available tools, which are most representative in authors opinion, and highlights missing tools or features.

# 2 Digital Design Flow

Modern digital design flow relies on many software tools to generate a chip layout from a given design. A majority of digital chips are built using libraries of standard cells provided by a foundry. In rare cases, where minimum delay or power consumption are imperative, digital circuits are designed on a transistor level - full custom design. Full custom digital design relies on analog design flow to accurately capture timing and power information, and is very time and resource consuming. As the nature of full custom digital design limits its scope to small designs, or critical parts of larger designs, it will not be considered as a part of digital flow in this paper.

Digital design can be roughly partitioned to logic and physical design phase. In a logic design phase the design is synthesized - design is read-in from a given description, optimized for specified goals, and mapped to a set of available logic gates and sequential elements available in a given standard cell library. Physical design tool takes the synthesized design netlist, and performs iterative place and route (PnR) until design goals are satisfied. Finally, placed and routed design layout is extracted for parasitic resistance and capacitance, timing model is generated, and timing constraints are verified.

Digital circuit can be described at various level of abstraction, from gate level, through register transfer logic (RTL), to abstract descriptions on algorithmic level. Gate level description is rarely used, as it

locks the design to a specific standard cell library and process, and is time consuming, hard to debug and maintain.

Most designs are implemented on RTL level, usually in Verilog or VHDL, where design intent is explicitly stated, while implementation and optimization details are left to synthesis tool, according to specified constraints. Design intent in RTL encompasses specifying signal types, digital logic equations, state machine states and transitions, placement of pipeline registers etc. Synthesis tool deals with details of implementation and optimization, such as common subexpression elimination, resource duplication to satisfy fanout constraints, timing and area, state machine encoding, pipeline retiming etc. Output of synthesis tool is a netlist of gates from a given standard cell library.

Various higher levels of abstraction are also used in digital designs. One of prominent FOSS high level hardware description languages, developed at UC Berkeley, is Chisel [6]. Chisel is based on Scala language with many extension classes, which are used to translate the design to an intermediate representation, optimize it, and emit a Verilog code for synthesis. It is an abstract HDL, in the sense that it can infer signal types, and that it is more expressive than Verilog or VHDL, but is not abstract on algorithmic level. Chisel has attracted a lot of attention in academia and industry because design generators, ranging from block level to full SoC, have been written in it.

Algorithmic level abstraction tool, also called high level synthesis (HLS), takes design intent description in C/C++ (SystemC) or other high level language, and generates all logic equations, state machines, pipeling etc. automatically according to specified constraints. For example, to design a FIR filter, designer only provides filter coefficients and uses a C/C++ library function - the HLS tool takes care of details, such as constant multiplier design, pipelining etc. To the best of author's knowledge, there aren't any widely used algorithmic level digital design FOSS tools.

Simulation of complex digital designs is essential for development and verification. Verilog designs can be simulated with Icarus Verilog [7], while VHDL designs can be simulated with GHDL [8]. Both Icarus and GHDL can simulate any valid language construct, including timing delays and behavioral code. Orders of magnitude faster simulation of Verilog code is possible with Verilator [9] if only a synthesizable subset of Verilog is used in a design. Verilator transforms synthesizable Verilog to optimized C++ code, which can be compiled and used as any software component. Common use case for Verilator is to generate a cycle/bit accurate C++ simulator for CPUs or DSP blocks.

Regardless of whether the digital design is specified in Verilog/VHDL or some higher level tool, eventually it has to be synthesized to standard library gates. Digital design has been made truly vendor-agnostic by standardizing:

- Logic function, timing and power are specified in Liberty library (.LIB) files, containing tables of delay, rise and fall times, and power vs loading capacitance.
- Process information and simplified abstract views of gate layouts are provided in Library Exchange Format (.LEF). Simplified abstract views are derived from layout by keeping only the cell boundary, pin locations and routing blockage information. Complete cell layouts are needed only for LVS, DRC or tapeout.
- Cell placement and design routing is kept in a Design Exchange File (.DEF).

Most foundries provide standard digital libraries under NDA, with SkyWater being a notable exception. SkyWater is providing open-source PDK for a 130 nm CMOS process [10], which can be used in open-source digital flow.

OpenROAD project [2] has produced a complete FOSS digital flow called OpenLane [3] by putting together available tools and developing new ones. OpenLane is not the first complete FOSS digital

flow, but is actively maintained and has gained significant attention.

Design flow in OpenLane consists of design exploration outer loop and synthesis exploration inner loop. Design exploration loop evaluates a complete - synthesized, placed and routed - design. During synthesis exploration the design is repeatedly synthesized with various options, and result of each synthesis run is processed by static timing analysis tool OpenSTA to evaluate its performance.

Synthesis in OpenLane flow is performed by Yosys+ABC [4]. Yosys is currently the most complete open-source synthesis tool, which can synthesize a Verilog design, optimize and map to gates from a given Liberty library. It has been used to synthesize complex digital systems, and is considered to be a robust tool.

However, some features are missing in Yosys. VHDL support requires an external proprietary tool, but work is ongoing to provide open-source alternative. More importantly, advanced synthesis options, such as pipeline retiming, physically aware synthesis, clock domains, support for SDC constraints, etc. are missing. These advanced features are important to achieve better quality of results, comparable to prorietary tools.

Floorplanning, placement, clock tree synthesis, optimization and global routing of synthesized design is performed by OpenROAD applications. Optimizations performed during physical design phase are not only related to cell placement and routing, but also gate resizing to satisfy design constraints. Placed and routed design is exported to design exchange format (DEF).

When design exploration is finished, parasitic resistances and capacitances of routed design are extracted with SPEF Extractor [5], saved in a Standard Parasitic Exchange Format (SPEF) and used for final STA check. SPEF Extractor is a Python script which takes library LEF and design DEF files and calculates parasitic resistance and capacitance of routing. Algorithm used for RC extraction is simple, straight forward, and is likely to produce low accuracy results, so more work, or another extractor, is needed to match the accuracy of proprietary tools.

Design rule check (DRC) is performed by Magic, while the layout versus schematic check is performed by netgen. If there are no errors the design is exported to GDSII file, which is used for chip fabrication.

The described digital flow is complete, in a sense that a FOSS tool exist for every step from RTL to chip layout in GDSII. It is incomplete in a sense that some tools are not adequate for complex designs, and should be improved or replaced.

# 3 Analog Design

Standardization of library formats for digital design (LIB, LEF, DEF) has enabled a true vendor-agnostic design flow. Such standardization does not exist for analog design flow, and foundry-provided process design kits are usually tied to one vendor. Process design kits contain device model decks used for simulation, parametrized primitive device layouts, physical verification (DRC, LVS) rules and process information for parasitic extraction, all of which are needed in analog design flow.

Circuit simulation is a first step in the analog design flow. One of the best known open-source simulator is SPICE, developed at UC Berkeley. SPICE circuit simulator was revolutionary at the time it was introduced. It provided robust algorithms for solving electrical circuits, and an extensible framework to add new device models and analyses. Perhaps most importantly - authors have made the SPICE source code publicly available. open-source derivatives of SPICE, such as Ngspice [11], are still actively developed.

Last version of SPICE released by UC Berkeley (SPICE3F5) was improved and extended over the years, requiring changes of netlist syntax. Changes of netlist syntax were necessary to support convenience features, such as variables and expressions, and core features, such as model binning, new

device types and analyses. Myriad of SPICE-like simulators, both FOSS and propietary, expanded the original SPICE netlist syntax independently, resulting in many incompatible dialects - SPICE Tower of Babel. Proprietary extensions, for example ”A” behavioral devices in LTSpice, are another source of incompatibility. To make matters even more complicated, some proprietary simulators are using proprietary netlist languages.

Incompatible netlist formats, either SPICE dialects or proprietary languages, pose a major problem in using FOSS circuit simulators. To the best of author’s knowledge, there is no technical reason why that must be the case. Most device models, such as BSIM, PSP, etc., are developed by a third party and are freely available. Therefore, model decks for any simulator, i.e. in any netlist format, convey the same information - the only issue seems to be how to read-in model parameters from a netlist.

A possible solution for netlist language incompatibilites is to standardize a netlist language for circuit simulation. Standardization is a colossal effort, and would probably be affected by conflicting interests, so it might take a very long time, if ever, to complete. A step in the right direction would be to develop an unified intermediate representation of analog netlists, and parsers for existing netlist languages, which would decouple simulator front-end from a simulator core. Short-term solution, or rather a duct tape to hold the design flow together, would be to make a translator of netlist in proprietary netlist language to flat SPICE netlist. Flat SPICE netlist would elliminate any issues with hierarchy, parameter evaluation, conditional expressions, etc. and should enable the use of FOSS simulators with already existing PDKs. From a technical point of view, such translators should be feasible, but whether there would be any legal issues is beyond the scope of this paper and author’s expertise.

For an analog circuit netlist to be truly vendor-agnostic, non-standard or behavioral devices should be modeled with Verilog-A instead of using simulator specific custom devices, or (mis)using controlled sources by exploiting some simulator specific implementation. Therefore, Verilog-A support is of paramount importance for circuit simulators, even for core device models. For example, BSIM6 is provided only as a Verilog-A model, in contrast to earlier versions of BSIM.

Support for compiling and dynamic loading of Verilog-A models is a standard feature in proprietary simulators. In principle, Verilog-A models could be used instead of built-in models, effectively decoupling the simulator core from models and thus providing ultimate flexibility. Although possible, such approach is not used for core models due to performance penalty of using automatically generated code, but is useful for custom Verilog-A models. FOSS circuit simulators, such as Ngspice, Qucs and Xyce, rely on a tool called ADMS for compiling a Verilog-A code.

ADMS is a tool which reads a Verilog-A model and outputs code based on rules and code templates given in XML. Processing a Verilog-A model involves parsing, building an abstract representation, which usually requires algebraic manupulations, calculation of derivatives and optimizations, such as node collapsing. Abstract representation is then used to generate simulator-specific code from provided templates. Generated code can be compiled to a shared library, which can be dynamically loaded by Ngspice to add support for new models.

Dynamically adding new device or behavioral models to simulator via compiled Verilog-A code provides ultimate flexibility and is a very important feature. However, ADMS does not support all of Verilog-A constructs, which poses a limitation on which devices or behavioral models can be added. Furthermore, ADMS is not actively developed nor maintained, and is likely to remain that way. In author’s opinion, revival of ADMS development, or a new implementation, should be one of priorities for analog design flow to progress.

Simulator architecture and mathematical formulation can have a significant impact on code maintenance and development, as well on performance and scalability. SPICE derivatives, such as Ngspice, are still using legacy progamming paradigms and code base. For example, SPICE requires that each device model provide a separate function for each type of analysis. Analysis-specific device model

function is then responsible for calculating MNA contributions by calling a numerical integration routine NIntegrate, provided by simulator core. Although used in SPICE derivatives for decades, there are advantages and drawbacks of such approach.

Advantage of requiring that device model provide a separate function for each analysis type is that mathematical formulation and implemenation could be optimized for a specific analysis. As a side effect, it simplifies simulator development since it delegates matrix fill-in to device models. An obvious disadvantage is that introducing a new type of analysis would require updating all device models with a new function. A not-so-obvious disadvantage is that direct matrix fill-in by device models, which rely on simulator-provided numerical integration, couples simulator core algorithms with device models. Such copling complicates development of either simulator core or device models, as any changes might have unforeseen effects and require changes in a large code base.

Direct matrix fill-in hides important details from a simulator, which might be needed in certain types of analyses. For example, mathematical formulation of harmonic balance (HB) analysis requires that a circuit is partitioned to linear and non-linear parts. Separating linear from non-linear branches is difficult when a device model provides only numerical values for matrix fill-in. Similar problems arise in other types of analyses, such as periodic steady state (PSS) analysis and its small signal variants. Aforementioned difficulties could be, in principle, solved within legacy SPICE framework, but it would require more hacking the existing code than developing new features, and making code increasingly difficult to maintain and develop.

Demand for increased integration and low power has resulted in SoCs which contain digital, analog, mixed signal, RF and power converters on the same chip. Harmonic balance and periodic steady state, followed by corresponding small signal and transfer function, analyses are essential for development of such SoCs. Lack of HB, PSS and corresponding small signal and transfer function analyses is a major obstacle for designing highly integrated SoCs.

Besides FOSS SPICE circuit simulator derivatives, there are simulators using modern object oriented programming paradigms and mathematical formulations. Xyce [12] is an open-source simulator developed at Sandia National Laboratories. It is not a SPICE derivative, but a new implementation written in C++, using advantages of object oriented programming paradigm to allow easier development, adding new features, and code which is maintenable in the long term. Mathematical formulation used in Xyce is different than the one used in SPICE. Instead of relying on device models to perform numerical integration and fill-in MNA matrix, as is the case in SPICE-based simulators, Xyce uses Differential-Algebraic Equation (DAE) formulation. DAE formulation, used in Xyce and some proprietary simulators, requires that device models only provide branch contributions and their derivatives, while numerical integration and MNA matrix fill-in is handled by the simulator core algorithms.

SPICE-derivatives and DAE-based simulators eventually solve a system of equations determined by MNA matrix and right-hand side, and it may seem that DAE formulation is a minor reformulation of SPICE algorithms. However, this is not the case - DAE formulation is a major improvement for several reasons. In DAE-based simulators device models are just that - they provide information about branches, calculate branch currents, charges and their derivatives from given terminal voltages - nothing more or less. Everything else - numerical integration, analysis specific circuit partition, MNA fill-in etc. is handled exclusively by the simulator core. This in effect decouples the development of device models and simulator algorithms and analyses. Any improvement in simulator core, such as new analysis type, or a new ODE integration algorithm, does not require any change in device models, and is immediately available. In contrast, implementing a new type of simulation in SPICE-derivatives would require that a new function is provided for each device model.

Device models for DAE-based simulators are available in NXP SiMKit [13], an open-source library providing PSP and other models. Besides an open-source version of SiMKit, there is a proprietary

versions of SimKit which can be used with Cadence Spectre and Keysight ADS. SiMKit uses an API which allows the simulator to dynamically load and query the library for available device models, instantiate a device model with given model parameters, and use it to calculate branch contributions. Such API is general enough to be considered as a good candidate for all models in DAE simulators.

Analog, mixed signal and RF circuits are sensitive to physical implementation effects, and designers go to great lengths to ensure that circuit layout does not degrade performance. Physical implementation effects can roughly be divided into two categories:

- parasitic resistance, capacitance, inductance and inductive coupling,
- geometry-dependent effects that alter the device model.

In almost all parasitic extraction flows inductors and transformers are treated as black boxes, and extraction of only parasitic resistance and capacitance results in acceptable accuracy most of the time, which is also used in digital design flow. Geometry-dependent effects are not extracted at all in digital design flow, since only routing parasitics are extracted. Effect of any geometry-dependent effects are expected to be captured in Liberty library files, either from analog simulation of extracted cells, or from silicon characterization. Accurate extraction of analog, mixed signal and RF circuit layouts cannot rely only on parasitic RC extraction because geometry-dependent effects can have significant impact on performance.

To eliminate, or at least minimize geometry-dependent effects analog designers use many layout techniques, some of which are:

- common centroid layout to minimize the effects of process gradients,
- same source-drain orientation of matched devices to elliminate mismatch due to MOSFET chirality caused by ion implantation at an angle,
- avoididng routing metal, and blocking dummy filler, over poly in matched transistors and resistors to avoid device mismatch due to hydrogen trapping,
- placing dummy devices (resistors, MOS, capacitors) around matched devices to equalize density-dependent effectcs, such as etcing bias, metal dishing etc.,
- placing sensitive circuits, which rely on device matcing, in the middle of the chip to minimize mismatch due to mechanical stress of seal ring, dicing and bonding.

Although all of the above listed effects are well known, and some of them are quantified in foundry-provided process documentation, they are not considered by any device model nor extracted by parasitic extractor. However, some important geometry-dependent effects are included in transistor models, but are not extracted by FOSS parasitic extractors. These effects are well proximity effect (WPE) and length of diffusion effect (LOD).

Well proximity effect captures the change in NMOS device model due to proximity of N well, caused by ion scattering during well implantation. Matching of asymmetrically placed transistors is severely degraded by proximity of N well, and can cause a costly chip re-spin if it is not caught by simulation of extracted circuit with effects of WPE.

Mobility of carriers in MOS transistors is affected by compressive mechanical stress, degrading mobility of carriers in NMOS and enhancing it in PMOS. Stress-dependent mobility is used in many advanced processes to enhance MOS performance by deliberately introducing mechanical strain in

the channel. Mechanical stress is also dependent on the distance from the channel to the nearest shallow trench isolation (STI), equal to the length of source or drain diffusion, hence the name length of diffusion. Difference of LOD lenght in matched devices, and corresponding change of mobility, can result in as much as 20% mismatch in drain current - an error that is unnaceptable in analog design. This LOD-dependent current mismatch cannot be simply trimmed because it is temperature dependent, ultimately resulting in poor performance and chip respin. Device current mismatch could be caught by simulation of extracted circuit, if LOD effects are extracted.

To summarize, RC-only extraction is not adequate for analog, mixed signal and RF designs - geometry-dependent effects have to be extracted as well. Extraction of geometry-dependent effects is a major feature missing in FOSS extractors, and should be considered as an important future improvement.

Another very important feature for RF design, missing in FOSS extractors, is parasitic blocking. It is common for RF devices to be modeled with extrinsic model - it includes intrinsic parasitics of semiconductor device and extrinsic parasitics of local interconnects. Extrinsic model is of great importance for RF desingers, because they can optimize the device size to achieve design goals without having to draw a layout for each device size. Without extrinsic model, designer would have to estimate the local interconnect parasitics and include them as ideal elements, find a candidate device size from simulation, draw a layout and extract it, just to see that parasitic estimate was not correct. To say that optimizing an RF circuit performance without extrinsic model is very tedious and time consuming is an understatement.

Extrinsic device models are great - but there is a problem: they already include local routing, which would be double-counted by parasitic extraction, producing an incorrect extracted circuit. Simply removing the device layout and treating it as a black box does not solve the problem either, because it would exclude the parasitic capacitance from local interconnect to layout drawn by designer, and underestimate parasitics. The only solution is to instruct parasitic extractor to block extraction of certain layers in certain cells, which have already been accounted for in the device model. This feature is present in most proprietary parasitic extractors, but is missing in FOSS extractors.

Even in parasitic extractor does its job perfectly - accurately extracts layout parasitics and geometry-dependent effects, and properly performs parasitic blocking - the resulting netlist of top level design is usually too big for simulation with available resources in a reasonable time. Netlist size problem is usually solved (?) by partitioning the top level design, mixing extracted netlists of critical blocks with schematic, or even behavioral models, of supporting circuits. Putting aside that such approach is error prone, and does not really simulate the whole chip, there are cases where the netlist is still to big. The reason for excessive netlist size it that parasitic extractors generate circuits with too much details, possibly filtering out only very small parasitic resistors (mili Ohms) and capacitors (atto Farads).

Parasitic reduction tools take an extracted circuit netlist and create an equivalent circuit with specified tolerance at maximum frequency of interest. Reduction of extracted circuit size can be dramatic, by more than order of magnitude, without compromising simulation results accuracy. Smaller netlist results in faster simulations, and in some cases enable the simulation to be run at all, especially on top level or large blocks. To the best of author's knowledge there aren't any FOSS parasitic reduction tools available.

Electromagnetic simulators are also missing from FOSS desing flow, with one notable exception - ASITIC [14]. ASITIC is not a general purpose electromagnetic simulator, but rather a magneto- and electro-static simulator for supported geometries. It is free, but is not open-source, and it has been used in both academia and industry for more than twenty years. At the time ASITIC was introduced CMOS processes were mostly used for digital designs, and phrase "CMOS RF" was considered as oxymoron. In such historical context ASITIC was an invaluable tool for CMOS RF pioneers, and many others to follow even into mm-Wave. Last release of ASITIC was in 2001, and it is long overdue for an update.

Many things have changed since the last release of ASITIC - thick metal options are common in CMOS processes, modern CPUs have with powerful SIMD instructions, and desktop PCs have more cores than servers twenty years ago. To illustrate the point, some proprietary EM simulators simulate an inductor in full wave mode over a broad frequency range, from DC to beyond self resonant frequency, in less time than it takes ASITIC to simulate the same inductor at a single frequency. Furthermore, proprietary EM simulators can simulate an arbitrary geometry provided in GDSII file. Having in mind that the design of all RF, microwave and high speed IO circuits rely on EM simulation, lack of FOSS EM simulator is a significant drawback.

While on the topic of high-performance RF, microwave or high speed IO, required performance has long surpassed the raw (uncalibrated) peformance of CMOS. Digital circuits have benefited from each generation of scaled CMOS nodes, reducing the power consumption and area, while increasing operating frequency and scale of integration. Process scaling has not been so generous to analog/RF circuits - constant field CMOS scaling has resulted in reduced supply voltage, and consequently reduced available voltage swing and dynamic range. Going into deep sub-micron nodes, mismatch of close to minimum transistor size cannot be considered as a perturbation of a process corner, but rather a large deviation, having detrimental effect on performance and yield. If transistors of minimum, or close to minimum, channel length cannot be used in a given process due to excessive mismatch, there is no point of using it - an older, and more stable CMOS node could be used, at lower cost as well.

Key to using an advanced CMOS node to its full potential is digital calibration, correction and assistance. For example, spurios free dynamic range (SFDR) of RF D/A converter without any digital calibration is determined solely on transistor matching, proportional to inverse square root of device area, in a given CMOS process. High resolution D/A converters require large transistors, resulting in large area, high power consumption and/or limited maximum frequency, to achieve desired SFDR without using digital assitance [15]. Mixed signal simulations are essential for design and characterization of digitally assisted analog/RF circuits.

Currently available mixed signal simulators support only basic simulations - DC, AC, transient, and there is not a single simulator supporting mixed signal PSS simulations. In principle, digital circuits can be simulated on transistor level with available simulators, but in reality it is not feasible due to excessive number of transistors. There have been hacks to perform PSS with digital circuits by compiling Verilog code with Verilator and using it within Verilog-A module [15], but this is not a general approach. Future improvents of FOSS circuit simulators could consider using Verilator for mixed-signal PSS simulation in a general manner.

# 4 Conclusion

An overview of free software for analog and digital design has been presented in this paper. In general, FOSS digital design flow is in much better shape than analog. One of the reasons is that file formats for digital design are standardized, while PDKs for analog design are usually locked to one vendor. Besides an overview of available FOSS tools, some thoughts on possible improvements and missing tools are also given. Hopefully these thoughts will contribute to future directions of development.

Development of EDA tools for chip design requires specialized knowledge and a lot of resources. It is interesting to note that many tools used in FOSS digital and analog flow were developed by individuals or projects outside of universities. Shift of FOSS tool development from universities to individuals/projects might be explained by pressure to publish papers. Some form of academic recognition for development of FOSS tools might restore interest at universities.

# References


[1] L. Conway, Reminiscences of the VLSI Revolution: How a Series of Failures Triggered a Paradigm Shift in Digital Design, in IEEE Solid-State Circuits Magazine, vol. 4, no. 4, pp. 8-31, Dec. 2012, doi: 10.1109/MSSC.2012.2215752.

[2] The OpenROAD Project. https://theopenroadproject.org

[3] OpenLane. https://github.com/The-OpenROAD-Project/OpenLane

[4] Yosys. https://github.com/YosysHQ/yosys

[5] SPEF Extractor. https://github.com/HanyMoussa/SPEF_EXTRACTOR

[6] Chisel. https://www.chisel-lang.org

[7] Icarus Verilog simulator. http://iverilog.icarus.com

[8] GHDL VHDL simulator. http://ghdl.free.fr

[9] Verilator. https://www.veripool.org/verilator

[10] SkyWater 130 nm PDK. https://github.com/google/skywater-pdk

[11] Ngspice. http://ngspice.sourceforge.net

[12] Xyce. https://xyce.sandia.gov

[13] SiMKit. https://www.nxp.com/design/software/models/simkit:SIMKIT

[14] ASITIC. https://rfic.eecs.berkeley.edu/ niknejad/asitic.html

[15] D. Grujic et al., Periodic Steady State Simulation of Mixed-Signal RF Circuits , SSSS Conference 2016.