Review scientific paper
https://doi.org/10.33180/InfMIDEM2024.404

Informacije MIDEM
Journal of Microelectronics,
Electronic Components and Materials
Vol. 54, No. 4(2024), 271 – 281

Free Software Support for Compact Modelling with Verilog-A

Árpád Bűrmen, Tadej Tuma, Iztok Fajfar, Janez Puhan, Žiga Rojec, Matevž Kunaver, Sašo Tomažič

University of Ljubljana, Faculty of Electrical Engineering

Abstract: Verilog-A is the analog subset of Verilog-AMS - a hardware description language for analog and mixed-signal systems. Verilog-A is commonly used for the distribution of compact models of semiconductor devices. for such models to be usable a Verilog-A compiler is required. The compiler converts the model equations into a form that can be used by the simulator. Such compilers have been supplied with commercial simulators for many years now. Free software alternatives are much more scarce and limited in the features they offer. The paper gives an overview of Verilog-A, Free software Verilog-A compilers, and Free software/Open source simulators that can simulate compact models defined in Verilog-A. Advantages and disadvantages of individual compilers and simulators are highlighted.

Keywords: analog circuits, compact models, Verilog-A, compiler, simulation

Odprtokodna programska oprema za uporabo kompaktnih modelov v jeziku Verilog-A

Izvleček: Verilog-AMS je opisni jezik za mešana analogno-digitalna vezja. Verilog-A je njegov podsklop, ki je namenjen opisu analognih vezij. Pogosto ga uporabljamo za distribucijo kompaktnih modelov polprevodniških elementov. Da bi take modele lahko uporabili v simulatorju vezij, potrebujemo prevajalnik za Verilog-A. Ta pretvori model v obliko, ki jo simulator lahko uporabi pri izračunu odziva vezja. Prevajalniki za Verilog-A so že dlje časa sestavni del tržnih programskih paketov za simulacijo vezij. Odprtokodnih alternativ je manj in podpirajo samo del specifikacije jezika. Članek poda pregled odprtokodnih prevajalnikov in simulatorjev s podporo za kompaktne medele opisane v jeziku Verilog-A s poudarkom na prednostih in slabostih posameznih prevajalnikov in simulatorjev.

Ključne besede: analogna vezja, kompaktni modeli, Verilog-A, prevajalnik, simulacija

** Corresponding Author's e-mail: arpad.buermen@fe.uni-lj.si*

1 History of Verilog-A

Verilog-A is a hardware description language (HDL) for analog circuits. It is based on Verilog, which by itself is a HDL for digital circuits. The history of Verilog [1] dates back to 1980s when Gateway Design Automation introduced the language. In 1990 the language was acquired by Cadence Design Systems. The language was transferred to public domain where it was supported and extended by Open Verilog International (OVI). In 2000 Accellera Systems Initiative was founded from the merger between OVI and VHDL International and has been managing the language to date.

Verilog has been standardised by IEEE as standards 1364-1995 (Verilog-95) [2], 1364-2001 (Verilog-2001) [3], and 1364-2005 (Verilog-2005) [4]. Since then Verilog has evolved into SystemVerilog which offers new design (data lifetime specification, more advanced data types, new procedural blocks, and interfaces) and verification features (new data types, object-oriented programming model, generation of constrained random values, assertions, coverage, and synchronisation primitives). SystemVerilog has been standardised by IEEE as standards 1800-2005 (superset of Verilog-2005) [5] and 1800-2009 (SystemVerilog 2009) [6] with fur-

How to cite:
Á. Bűrmen et al., "Free Software Support for Compact Modelling with Verilog-A", Inf. Midem-J. Microelectron. Electron. Compon. Mater., Vol. 54, No. 4(2024), pp. 271–281

ther updates in 2012 (1800-2012) [7], 2017 (1800-2017) [8], and 2023 (1800-2023) [9].

In parallel to the evolution of Verilog a new HDL for analog/mixed signal systems was being developed. Verilog-A, which is a HDL for purely analog systems, was released in 1996 by OVI [10]. Its syntax was based on the syntax of Verilog, but the language constructs were designed for describing analog systems in terms of ordinary differential equations (ODE). The language was primarily created to standardize the Spectre simulator's behavioral language in times when it was facing competition from VHDL that was getting analog capabilities via incorporating analog HDL languages like MAST [13]. Verilog-A was developed with a more advanced language in mind - one that would be capable of describing analog, as well as, mixed-mode systems. The language was released in 1998 and was deemed Verilog-AMS (version 1.3). In the year 2000 version 2.0 of Verilog-AMS was released. Since then Verilog-AMS has beed updated in 2009 (version 2.3.1), 2014 (version 2.4.0) [11], and 2023 (Verilog-AMS 2023) [12]. Currently work is underway to merge Verilog-AMS with SystemVerilog to produce SystemVerilog-AMS [14]. Verilog-A and Verilog-AMS did not become IEEE standards and have remained under the oversight of Accellera. Modern Verilog-A is the analog subset of Verilog-AMS.

Free software [15] alternatives for Verilog-A are important, among other things, because they make Verilog-A and the compact models defined in Verilog-A available to a wide audience without having to pay the high cost of commercial tools. Free software means that the users have the freedom to run, copy, distribute, study, change and improve the software. It is usually licensed under the GNU General Public License (GPL) or some other compatible license. Free software must not be confused with free software (lowercase). The latter means only that the price of the software is zero and does not give its users the same freedom as Free software. All Free software is Open source [16], but every Open source software is not Free software. Open source licenses can be more restrictive that GPL. Free software has great impact. As an example, consider the importance, usefulness, and implications of the GNU C Compiler [17] or Free software for Verilog-95 simulation (i.e. Verilator, [18]).

2 Using Verilog-A for compact modelling

Verilog-A is commonly used for the distribution of compact models (CM) of semiconductor devices. Compact models provide the equations linking terminal currents to terminal voltages of circuit components like MOSFETs, bipolar transistors, diodes, etc. The usual approach to formulating circuit equations is modified nodal analysis (MNA) [19] where as many as possible branch currents are explicitly expressed and substituted into Kichoff current law equations.

In every circuit one Kirchoff current law equation (KCL) can be constructed for each node, with the exception of the reference node. Each node is associated with a nodal voltage (potential) which becomes an unknown in the system of equations. An exception to this is the reference node whose nodal voltage is assumed to be zero. Branch currents that cannot be explicitly expressed are kept as unknowns in the system of equations. In circuit simulation such branch currents are treated as the associated quantities of so-called flow nodes. For each flow node an equation has to be added to the system. This equation is obtained from the constitutive relation of the element where the aforementioned branch resides and has a similar role as the KCL equation of an ordinary node. Flow nodes are typically used for modelling voltage sources and inductors.

Equations of a model are formulated as ordinary differential equations (ODE). Let us assume a circuit element has n nodes (ordinary and flow nodes) of which the first $m \leq n$ nodes are terminals. All terminals are assumed to be ordinary nodes. The associated quantities of the nodes are considered to be the independent variables and their values are listed in vector **x**. Let **y** denote the vector of terminal currents where a current is assumed to be positive if it flows into the corresponding terminal. Components of **y** that correspond flow nodes or internal nodes are assumed to be 0.

Let **g(x)** and **q(x)** denote two vector valued (nonlinear) functions of independent variables. These two functions represent the resistive and the reactive contributions to the equations associated with the aforementioned *n* nodes. For ordinary nodes the components of **g(x)** and **q(x)** correspond to resistive currents flowing from the nodes and charges accumulated at the nodes, respectively. For flow nodes they correspond to voltages and fluxes. The resistive currents are assumed to be positive if they flow outward from a node. After a Verilog-A compact model is compiled its equations are formulated as

$$\mathbf{y} = \mathbf{g}(\mathbf{x}) + \frac{\mathrm{d}}{\mathrm{d}t}\mathbf{q}(\mathbf{x}) \tag{1}$$

As an example, let us consider a semiconductor diode in Figure 1. The model has two terminals (A and C) and one internal node (Ai). It comprises a linear resistor (R_S) that models the series resistance of a diode and a core

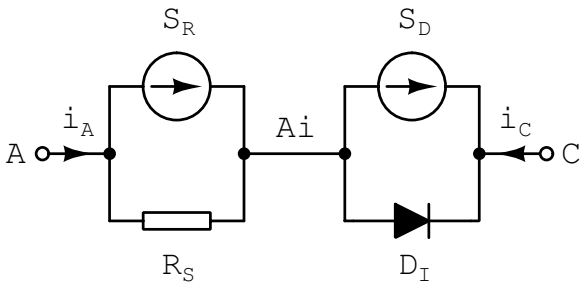

Figure 1: Model of a semiconductor diode. Noise sources S_D and S_R are treated separately by Verilog-A.

diode that models the nonlinear diode characteristic and its charge storage. The noise generated by the diode and its series resistance is modelled by noise sources with power spectral densities S_R and S_D. Vectors **y** and **x** in equation (1) can be written as $\mathbf{y} = \begin{bmatrix} i_A & i_C & 0 \end{bmatrix}^{\mathrm{t}}$ and $\mathbf{x} = \begin{bmatrix} v_A & v_C & v_{Ai} \end{bmatrix}^{\mathrm{T}}$. The two nonlinear vector-valued functions are

$$\mathbf{g}(\mathbf{x}) = \begin{bmatrix} R_S^{-1}\left(v_A - v_{Ai}\right) \\ -i_D\left(v_{Ai} - v_C\right) \\ R_S^{-1}\left(v_{Ai} - v_A\right) + i_D\left(v_{Ai} - v_C\right) \end{bmatrix} \tag{2}$$

$$\mathbf{q}(\mathbf{x}) = \begin{bmatrix} 0 & -q_D\left(v_{Ai} - v_C\right) & q_D\left(v_{Ai} - v_C\right) \end{bmatrix}^T \tag{3}$$

To simplify expressions let us neglect the diode's junction capacitance and assume it exhibits only diffusion capacitance. Then functions i_D and q_D can be written as

$$i_D(u) = I_S\left(\exp\left(\frac{u}{V_T}\right) - 1\right) \tag{4}$$

$$q_D(u) = \tau\frac{I_S}{V_T}\exp\left(\frac{u}{V_T}\right) \tag{5}$$

The power spectral densities of the two noise sources are $S_R = 4kT / R_S$ and $S_D = 2qi_D\left(v_{Ai} - v_C\right) + K_f i_D\left(v_{Ai} - v_C\right)^{A_f} / f$. I_S, τ, K_f, and A_f are diode parameters and V_T is the thermal voltage (kT/q). The Boltzmann constant, the absolute temperature, and the electron charge are denoted by k, T, and q, respectively. The core diode noise source (S_D) comprises a frequency-independent shot noise component and a flicker noise component whose power spectral density is inversely proportional to the frequency. The Verilog-A code defining the diode model is

```
 `include "constants.vams"
 `include "disciplines.vams"
module diode(A,C);
    inout A, C;
    electrical A, C, AI;
    parameter real Is = 1e-14 from [0:inf];
    parameter real Rs = 0.1 from (0:inf];
    parameter real Tau = 1e-6 from [0:inf];
    parameter real Kf = 1e-12 from [0:inf];
    parameter real Af = 1 from (0:inf];
    real VT, id, qd, g;
    analog begin
      VT = `P_K*$temperature/`P_Q;
      id = Is*(exp(V(AI, C)/VT)-1);
      g = Is/VT*exp(V(AI, C)/VT);
      qd = Tau*g;
      I(A, AI) <+ V(A, AI) / Rs;
      I(AI, C) <+ id + ddt(qd);
      I(A, AI) <+ white_noise(4*`P_K*$temperature/
      Rs, "rs");
      I(AI, C) <+ white_noise(2*`P_Q*id, "id");
      I(AI, C) <+ flicker_noise(Kf*pow(id, Af), 1, "flicker");
    end
endmodule
```

Once all models formulate their equations along the lines of (1) the system of equations describing a circuit can easily be assembled. For each circuit element the components of vector **y** corresponding to terminals are simply added to the KCL equations of nodes to which these terminals are connected. Rows of (1) that correspond to internal nodes and complement the circuit's KCL equations as extra equations.

Verilog-A is capable of describing all aspects of a device covered by a legacy SPICE3 model implemented in C. There are several advantages in using Verilog- A. The models are significantly shorter. This arises from two facts. Writing a model in C can require many lines of code for expressing concepts that are expressed with a single line in Verilog-A. Secondly, Verilog-A compilers automatically derive the expressions for the derivatives of functions **g** and **q** with respect to components of **x**. These expressions must be formulated manually in SPICE3 models and can easily double the amount of C code that needs to be written. Manual implementation of derivatives is error prone. Incorrectly implemented derivatives result in convergence problems during simulation which can arise only under certain circumstances and are thus not easily detectable.

Table 1: Length in lines of code (l) and the number of parameters (p) for various MOSFET compact device models.

Compact model	released	language	l	p	l/p
BSIM3 3.2.4	2001	C	14176	439	32
BSIM3 3.3.0	2005	C	13741	441	31
BSIM4 4.5.0	2005	C	23882	789	30
BSIM4 4.8.2	2020	C	27561	926	30
BSIM4 4.8 (Cogenda)	2019	Verilog-A	12591	897	14
BSIM6 6.0.0	2013	Verilog-A	3628	757	4.8
BSIM-BULK 107.1.0	2022	Verilog-A	4992	1073	4.7

Table 1 lists selected BSIM3, BSIM4 and BSIM-BULK (BSIM6) models [20]. The number of parameters of a model (p) is closely correlated with the size of the model expressed as lines of code (l). It is evident that SPICE3 models implemented in C are significantly more verbose than models implemented in Verilog-A. Models that are implemented in Verilog-A since their inception (BSIM6, BSIM-BULK) comprise roughly 6 times fewer lines of code per parameter than SPICE3 models implemented in C (BSIM3, BSIM4). Even models translated from a SPICE3 model (e.g. Cogenda BSIM4 4.8 [21]) comprise less than half the amount of code per parameter compared to SPICE3 models.

Modern device models, like BSIM-BULK, BSIM-SOI, HICUM, MEXTRAM, etc., are all released by their developers (mostly universities) in Verilog-A. Compact Model Coalition (CMC) [22] performs quality checks and verifies if the released models comply with standards.

3 Interfacing with a simulator

Simulators typically require from a model to compute the contributions to KCL equations (given by vector-valued functions **g** and **q**) for a given vector of independent variables (**x**). The system of first order differential equations is assembled as discussed in section 2 and can be expressed as

$$\mathbf{g}^*\left(\mathbf{x}^*\right)+\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{q}^*\left(\mathbf{x}^*\right)=0 \tag{6}$$

where vector **x*** is obtained by meaningfully merging vectors of independent variables corresponding to individual circuit elements (**x**) because an independent circuit variable can appear in multiple circuit elements. Functions **g*** and **q*** are obtained by adding up contributions from circuit components (**g** and **q**) depending on the way their terminals are connected to the circuit's nodes. Each node corresponds to one KCL equation. Contributions of grounded terminals are ignored. The last *n* - *m* components of each element's **g** and **q** correspond to extra equations. These equations complement the set of KCL equations to form the circuit's system of equations.

Numerical algorithms employed by simulators depend on the derivatives of **g*** and **q*** with respect to the independent variables **x***. These derivatives are gathered in the Jacobian matrices **G*** and **C*** whose components are given by

$$G_{ij}^*=\frac{\partial g_i^*}{\partial x_j^*} \tag{7}$$

$$C_{ij}^*=\frac{\partial q_i^*}{\partial x_j^*} \tag{8}$$

Because **g*** and **q*** were constructed by adding contributions from vector valued functions **g** and **q** matrices **G*** and **C*** can be constructed by adding contributions from Jacobian matrices **G** and **C** computed for functions **g** and **q**, respectively. To summarize, an analog device model must compute the Jacobian matrices **G** and **C** alongside **g** and **q** for a given vector of independent variables **x**.

In time-domain analysis equation (6) must be numerically integrated to obtain a system of nonlinear algebraic equations. When backward Euler integration is used equation (6) becomes

$$\mathbf{g}^*\left(\mathbf{x}^*\left(t_{k+1}\right)\right)+\frac{\mathbf{q}^*\left(\mathbf{x}^*\left(t_{k+1}\right)\right)-\mathbf{q}^*\left(\mathbf{x}^*\left(t_k\right)\right)}{t_{k+1}-t_k}=\mathbf{0} \tag{9}$$

where t_{k+1} is the timepoint for which we are solving the circuit and t_k is the previous timepoint where the circuit's solution is already known. In older simulators (e.g. SPICE3, Gnucap, and QUCS) numerical integration is performed by the device model itself. Consequently models in transient analysis do not compute a separate **q**(**x**). Instead they replace **g**(**x**) and its Jacobian with

$$\mathbf{f}\left(\mathbf{x}\right)=\mathbf{g}\left(\mathbf{x}\right)+\frac{\mathbf{q}\left(\mathbf{x}\right)-\mathbf{q}\left(\mathbf{x}\left(t_k\right)\right)}{t_{k+1}-t_k} \text{ and} \tag{10}$$

$$\mathbf{F}\left(\mathbf{x}\right)=\mathbf{G}\left(\mathbf{x}\right)+\left(t_{k+1}-t_k\right)^{-1}\mathbf{C}\left(\mathbf{x}\right) \tag{11}$$

In this way the code that computes the DC solution can be used without any modification for computing the time-domain solution from

$$\mathbf{f}^*\left(\mathbf{x}\right)=0. \tag{12}$$

Here $\mathbf{f}^*(\mathbf{x})$ is assembled from contributions of individual elements (i.e. $\mathbf{f}(\mathbf{x})$) in the same way as previously $\mathbf{g}^*(\mathbf{x})$ and $\mathbf{q}^*(\mathbf{x})$ have been assembled from $\mathbf{g}(\mathbf{x})$ and $\mathbf{q}(\mathbf{x})$.

This approach violates the separation between the simulator and the models and unnecessarily increases the size of the device model. On the other hand, it also has some advantages besides code reuse between DC and time-domain analysis, like the capability to implement non quasi-static effects in transistor models without adding extra nodes to the circuit (for how this is done in case of a bipolar transistor, see [23]). Serious flaws can also arise from this approach, e.g. the charge conservation problem in early MOSFET models [24, 25]. Charge nonconservation is a serious bug that was facilitated by the capability of handling numerical integration within models themselves. The problem originated from the attempt to formulate model's dynamics with ordinary reciprocal capacitors between nodes instead of charges stored at nodes. Charge non-conservation is impossible to "implement by accident" if charge based modelling is enforced like in Verilog-A.

Modern simulators separate numerical integration from device model evaluation. Using the formulation given by (6) as the basis of a circuit simulator adding new types of analysis becomes a much simpler task. This has been demonstrated in the past by commercial and free simulators, like Spectre [26] and fREEDA [27].

4 Free software compilers for compact models in Verilog-A

This sections gives an overview of Free software Verilog-A compilers where the term Verilog-A compiler is meant in a very broad sense. Two of these compilers (ADMS and OpenVAF) only support a subset of Verilog-A for compact modelling. The third one (Modelgen-Verilog) aims to be a full Verilog-AMS compiler once completed. Since this paper's focus is on compact modelling all three compilers are viable candidates for compiling compact models once their limitations are taken into account.

4.1 ADMS

ADMS (Automatic Device Model Synthesizer) [28] is the oldest of Free software Verilog-A compilers. It was developed by Motorola. At the time of its development MOSFET models were becoming excessively large. Back then the state of the art model (BSIM4) had almost 1000 parameters. The only way to use an advanced MOSFET model was either to use its official Open source implementation for the SPICE3 simulator and accept all the quirks and shortcomings of SPICE3 or implement the model from scratch for the simulator of choice.

Most commercial simulators at the time offered an API (e.g. [26, 29]) via which an external model could be implemented in C. Implementing a model with several hundred parameters involves writing tens of thuosands of lines of C code. Derivatives of currents and charges must be manually implemented. This process is error prone and slow. Furthermore, due to different APIs a large part of the model has to be rewritten for each simulator.

Verilog-A solves these problems since the model has to be implemented only once and the implementation can then be used by all simulators supporting Verilog-A. ADMS was developed as a tool that compiles Verilog-A into a model utilizing the C API of a selected simulator. After defining a new ADMS backend tailored to a specific simulator one can compile arbitrary Verilog-A models (within the limitations of ADMS) for that simulator. The process of compilation with ADMS is fairly slow. For a modern CMC model it can take more than a minute (e.g. for PSPv103 [42]). The generated code must be compiled (usually with a C/C++ compiler) and linked either statically with the simulator (e.g. Xyce [36]) or into a dynamic library that can be loaded by the simulator on-demand (e.g. Spectre [26], Ngspice [33], Gnucap [34]).

ADMS itself is implemented in C utilizing the Glib library [31]. The compiler operates by parsing the Verilog-A code and representing it in the Extensible Markup Language (XML) [30]. The specifications for the code generator (backend) are defined in XSLT, a subset of XSL [32], which is a language for representing XML document transformations. Models generated by ADMS are approximately 20% slower that hand-coded models [28].

Over the years ADMS has been used as the only available Verilog-A solution for compact modelling in Open source simulators like Ngspice [33], Gnucap [34], Quc-

sator [35], and Xyce [36]. Of all the listed simulators Xyce is the most advanced one with the best ADMS support. Nevertheless its ADMS integration has many limitations [37]. CMC models can be handled by ADMS after applying some manual modifications to the model (e.g. [38]).

ADMS is no longer being developed by its author. Development has been taken over by the Qucs project [39]. Contributions to the Git repository since 2017 are scarce and have ceased in 2022.

4.2 OpenVAF

OpenVAF [40] is a fairly recent development. It evolved from VerilogAE [41] whose primary purpose was to ease the process of model parameter extraction by retrieving the model equations from Verilog-A code. OpenVAF translates Verilog-A into a dynamic library with the help of the LLVM library [47]. LLVM emits highly optimized machine code and is generally used for implementing compilers. The resulting dynamic library interfaces with the simulator via the Open Source Device Interface API (OSDI API) [46].

Internally OpenVAF translates Verilog-A code into an abstract syntax tree (AST). Then it performs several transformations in the steps that follow. The first step resolves undefined references to other parts of the code to produce high-level intermediate representation (HIR). HIR is further processed by constructing a control flow graph, thus defining the execution order of the statements. Symbolic derivatives of expressions with Verilog-A operators ddx and ddt are computed to be later used during the construction of the Jacobians and module's output variables. The result of HIR processing is the medium level intermediate representation (MIR). From MIR the LLVM intermediate representation (IR) is generated. IR is a high-level abstraction of the machine code. LLVM performs several low-level optimizations on IR before emitting machine code for the target platform.

The resulting code is very efficient and faster than the code generated by an ordinary C/C++ compiler from ADMS output. OpenVAF supports a significant part of the Verilog-A specification and can compile all of the CMC models without any manual modifications. There are some limitations, though. The compiler does not

Table 2: Simulation runtimes for various models (taken from [42]). Builtin devices defined in C/C++ are denoted by a dash in the compiler column.

Simulator	Free	Compiler/built-in	t [s]	Comment
HICUM/L2v2p4p0 characteristic				
Ngspice	yes	OpenVAF	9.16	
Ngspice	yes	-	14.64	slow implementation
Xyce	yes	ADMS	36.42	strict convergence checks
Xyce	yes	-	26.56	strict convergence checks
ADS	no	proprietary	8.63	
ADS	no	-	7.01	
Spectre	no	proprietary	52.61	
Spectre	no	-	25.33	
BSIMSOI 4.4.0 characteristic				
Ngspice	yes	OpenVAF	8.47	
Ngspice	yes	-	7.98	manually optimized model
BSIMBULK 106.2 characteristic				
Ngspice	yes	OpenVAF	2.08	
Ngspice	yes	ADMS	3.38	
BSIMBULK 106.2 transient				
Ngspice	yes	OpenVAF	9.47	
Ngspice	yes	ADMS	13.70	
PSP 103.8 inverter				
Ngspice	yes	OpenVAF	20.01	
Ngspice	yes	ADMS	25.07	
PSP 103.8 with ISCAS C7552				
Ngspice	yes	OpenVAF	1200	
Ngspice	yes	ADMS	1500	

support analog events, genvars, hidden states, Laplace filters, paramsets, and hierarchical modules. But since these features are rarely used in compact models the lack of them does not represent a significant shortcoming at this point in time.

OpenVAF has replaced ADMS in Ngspice. It is also used by a free but closed-source simulator Spice Opus [48]. Finally, it is the core part of a novel Free software simulator VACASK [43, 44] for which the devices supported by the simulator are almost exclusively defined in Verilog-A.

Table 2 outlines the performance of OpenVAF-generated models with respect to builtin models (manually coded in C/C++), models generated by ADMS, and models generated by commercial compilers. These results are sparse and not sufficient to reliably determine the compiler that produces the fastest models, but nevertheless, they are a good indicator what one can expect from ADMS and OpenVAF.

Several Verilog-A compilers were tested by using the compiled HICUM model to compute the transistor's characteristics. OpenVAF comes out close to the top, second only to the compiler in ADS [49]. Xyce with ADMS comes out as one of the slowest solutions. This can be largely attributed to more strict convergence checks in Xyce when compared to Ngspice. Ngspice performance on this test problem can be attributed to sub-optimally coded derivatives in the built-in HICUM model.

When compared to a mature and highly optimized manually written builtin model in Ngspice (BSIMSOI 4.4.0) the OpenVAF-compiled model exhibits only 6% slower performance. On the two BSIMBULK test problems (characteristic and transient) the ADMS-compiled model is 45% to 60% slower than the one compiled with OpenVAF. This difference is significantly greater than the difference between models compiled by ADMS and manually coded models (models generated by ADMS are on average 20% slower). On the PSP inverter test problem the ADMS-compiled model is 20% slower than the one compiled with OpenVAF. The large test problem (ISCAS C7552) once again confirms the speed difference between models generated by ADMS and OpenVAF. These two benchmark results, the result obtained with the BSIMSOI model, and the fact that ADMS models are on average 20% slower than hand-coded models indicate that OpenVAF-generated models are roughly as fast as manually coded compact models.

4.3 Modelgen-Verilog

Modelgen-Verilog (MGV) [45] is a Verilog-AMS compiler for the Gnucap [34] circuit simulator. It has been in development since 2023. The ultimate goal of the project is to implement full support for Verilog-AMS in Gnucap. Presently the compiler outputs C++ code that is tightly coupled with the Gnucap simulator. After compiling and linking the code a dynamic library is obtained that can be loaded by Gnucap. The dependence on Gnucap could be removed in the future as backends for other simulators get added.

At the present (June 2024) the compiler seems to be capable of processing some CMC models [50], albeit quite inefficiently since a compiled PSP103 model uses 30 internal nodes, while its Verilog-A source code defines only 17 internal nodes. Consequently, simulations with the generated devices are reportedly slow [50]. A comparison akin to that in Table 2 has not been published yet.

A significant improvement in speed is expected from paramset support. Paramsets substitute most of the model parameters with concrete numbers upon which the expressions are simplified (constant folding) thus significantly reducing the computational burden of model evaluation. Further speedup could be obtained if the analog part of the compiler would implement optimizations akin to those in OpenVAF.

Modelgen-Verilog is a project whose ambitions are much bigger than the topic of this paper. Currently the compiler supports paramsets, analog events, hierarchical models, Verilog-A disciplines, discontinuities, and frequency domain filters. These features are missing in the remaining two Verilog-A compilers. Due to its early stage of development not many optimizations have been applied yet and there is a lot of room for improvement.

5 Free software/Open source simulators supporting compact models in Verilog-A

Table 3 gives a concise overview of the Free software/Open source analog circuit simulators that support compact models defined in Verilog-A. Note that the term Free software cannot be applied to Ngspice because of its license. Despite this Ngspice is still Open source and parts of it are Free software.

Core size of a simulator is the size of the simulator's source code excluding code that defines the device models. Simulators usually offer some kind of parameter sweep which is significantly more efficient than repeatedly running the simulator with a modified input file. Although a sparse linear solver is almost a must for a circuit simulator, not all simulators use one (e.g. Qucsator).

Table 3: Comparison of Free software simulators. Asterisk denotes a feature under development as of September 2024.

	Xyce	Ngspice	VACASK	Gnucap	Qucsator
Language	C++	C	C++	C++	C++
Core size (lines of code)	185500	63800	36700	28600	50300
Verilog-A CM support	ADMS	OpenVAF	OpenVAF	MGV or ADMS	ADMS
Operating point (OP)	yes	yes	yes	yes	yes
Small-signal AC	yes	yes	yes	yes	yes
Transient	yes	yes	yes	yes	yes
Small-signal noise	yes	yes	yes	no	yes
Harmonic balance	yes	no	no*	no	yes
Analyses supported by sweep	all	OP	all	OP	all
Sweep depth	arbitrary	2	arbitrary	arbitrary	1
Analysis/device separation	yes	no	yes	no	no
Sparse solver	yes	yes	yes	yes	no
Parallel evaluation	yes	yes	no	no	no
Parallel solver	yes	no	no	no	no
SPICE devices	yes	yes	no*	yes	partly

The process of simulation can be divided into two steps that in general must be repeated multiple times in order to complete a circuit analysis: evaluation of the circuit's components and solving a system of linear equations. Both steps can take advantage of parallel processing which can speed up the simulation and facilitate the simulation of circuits that are too big to fit on a single computer. Not many simulators exploit parallelism (only Xyce and partly Ngspice).

Finally, for a simulator it is important to provides basic SPICE device models (e.g. Gummel-Poon BJT, MOSFET levels 1-3, and 6, JFET, and MESFET). Mature simulators provide these device models (Xyce, Ngspice, Gnucap) while newer ones do not (VACASK, Qucs).

In the remainder of this section a more detailed description will be given for each one of the mentioned simulators.

5.1 Xyce

Xyce [36] is the most advanced of all the simulators listed in Table 3. Like all modern simulators, Xyce's core separates the device models from analysis implementation which makes it possible to implement a new analysis without having to change the device models. The simulator is capable of accelerating computations via parallel computing. Numerical capabilities are provided by the Trilinos [51] suite of libraries that offer unified wrappers around various state of the art solvers (like KLU). Element evaluation, as well as, certain linear solvers can take advantage of parallel processing. The latter is efficient only for very large circuits. Xyce offers all the standard SPICE circuit analyses, as well as, harmonic balance analysis.

Support for compact models in Verilog-A is provided by ADMS. The development team announced in 2022 [52] that they intend to build their own Verilog-A compiler based on an in-house Python library for (symbolic) differentiation. Since then there has been little news regarding this subject. Currently ADMS in Xyce has many limitations [37], largely due to the nature of ADMS.

5.2 Ngspice

Ngspice [33] is the most commonly used Open source simulator. It is based on the original SPICE3f5 source code in C. The original source code has been significantly extended and many bugs and shortcomings were fixed. One of these shortcomings was the original linear solver library of SPICE3 [55], which by now is no longer competitive. It has been replaced with the much faster KLU library [56].

Unfortunately, as is customary with all SPICE-based simulators, the models are tightly coupled with the circuit analyses. This makes it hard to add new types of analysis without making extensive changes to the large library of device models. Ngspice partly supports parallel evaluation of elements, either on multiple local CPU cores via OpenMP [53], or (for some elements) on a GPU via CUDA [54]. The linear solver, however, is not parallel.

Support for Verilog-A compact models was implemented at first with ADMS. Recently, the OSDI API has been

implemented which in turn makes it possible to use OpenVAF-generated models.

5.3 Gnucap

Gnucap [34] has a long history dating back to 1982. Since then it has been in slow, but steady development. The set of circuit analyses is fairly limited (only operating point/DC sweep, AC, and transient analyses are supported). The separation between the device models and the analyses is not complete as the models still have separate matrix loading functions for the time domain and for the frequency domain. This is alleviated by the fact that Gnucap's models are mostly generated with Modelgen, Gnucap's own model generator, not to be confused with Modelgen-Verilog. Models generated by both model generators are accessed by the simulator through the same API. Another shortcoming of Gnucap is its linear solver which is outdated. On the bright side, the solver offers functionality not available in other Free software circuit simulators because it can do partial solves of matrices when only a part of the matrix changes.
Support for Verilog-A compact models is provided by ADMS. Recently, development of a novel Verilog-AMS capable compiler for Gnucap has started (Modelgen-Verilog [45]). The compiler already supports a large subset of Verilog-AMS.

5.4 Qucsator

Qucsator [35] is a fairly new simulator whose beginnings date back into early 2000s when it started as the Quite universal circuit simulator (Qucs) project's own simulator. The simulator offers operating point/DC, AC, S-parameter, transient, and harmonic balance analysis. The models are tightly coupled with the analyses so implementing a new kind of analysis generally means all device models need to be modified, too. A major shortcoming is the fact that the simulator does not use a sparse linear solver. Instead an ad-hoc dense matrix solver is used, which makes the simulator impractical for anything but the smallest of circuits. Support for Verilog-A compact models is provided by ADMS.

5.5 VACASK

VACASK [43, 44] is a recently published simulator. It separates the models from the analyses thus simplifying the implementation of analyses by avoiding changes in device models. VACASK uses a state of the art linear solver (KLU).

The simulator offers operating point/DC, AC, small-signal transfer function (DC and AC), transient, and noise analysis. Harmonic balance analysis is currently under development, as well as, support for SPICE builtin device models. VACASK supports the OSDI API so that Verilog-A compact models compiled with the OpenVAF compiler can be used. In fact, most of the simulator's device library is implemented in Verilog-A. An exception to this are independent sources, linear controlled sources, and inductive couplings. These elements cannot easily be implemented in the Verilog-A subset supported by OpenVAF if one wants them to provide the same kind of interface as SPICE3 models do.

VACASK is in early stages of development. Preliminary benchmarks indicate that in single CPU mode it runs faster than Xyce, Gnucap, and Ngspice [43].

6 Conclusion

Verilog-A is the analog subset of Verilog-AMS. Over the years Verilog-A has become the de-facto standard for distributing compact models of semiconductor devices. Models implemented in Verilog-A need not specify any derivatives which makes the models significantly shorter and the coding process less errorprone. Verilog-A focuses on the equations describing the behavior of a circuit element. This reduces the size of a compact model by a factor up to 6 compared to SPICE3 compatible C code. Verilog-A compilers can significantly speed up the execution of a model by applying optimizations before the final machine code is emitted. The resulting model can be as fast as the hand-coded version of the model.

Verilog-A compilers are supplied with most commercial simulators. The available alternatives in the realm of Free software are much more scarce. Simulator developers can choose between three alternatives. ADMS is an old solution that requires manual intervention in the model code. OpenVAF is a modern compiler that produces fast models. Both alternatives support only a subset of Verilog-A. OpenVAF is more suitable because it is capable of compiling all public CMC models without modifications. The third alternative (Modelgen-Verilog) is a Verilog-AMS compiler that already supports a large part of the standard despite being in the early stages of development. It is capable of compiling Verilog-A compact models, but the resulting code is somewhat inefficient. Unfortunately its interface currently supports only the Gnucap simulator.

Several Open source and Free software simulators support Verilog-A, ranging from the most advanced one (Xyce), through SPICE3-based Ngspice, and newer simulators like Qucsator, Gnucap, and VACASK. All of these simulators support compact models defined in Verilog-A via one of the three mentioned alternatives.

7 Acknowledgements

This research was funded in part by the Slovenian Research Agency within the research program ICT4QoL—Information and Communications Technologies for Quality of Life, grant number P2-0246.

8 References

1. Flake, P., et.al. Verilog HDL and its ancestors and descendants. Proceedings of the ACM on Programming Languages, vol. 4 (2020), pp. 1–90.
2. 1364-1995 - IEEE Standard Verilog Hardware Description Language, 1996, https://doi.org/10.1109/IEEESTD.1996.81542.
3. 1364-2001 - IEEE Standard Verilog Hardware Description Language, 2001, https://doi.org/10.1109/IEEESTD.2001.93352.
4. 1364-2005 - IEEE Standard for Verilog Hardware Description Language, 2006, https://doi.org/10.1109/IEEESTD.2006.99495.
5. 1800-2005 - IEEE Standard for SystemVerilog: Unified Hardware Design, Specification and Verification Language, 2005, https://doi.org/10.1109/IEEESTD.2005.97972.
6. 1800-2009 - IEEE Standard for SystemVerilog–Unified Hardware Design, Specification, and Verification Language, 2009, https://doi.org/10.1109/IEEESTD.2009.5354441.
7. 1800-2012 - IEEE Standard for SystemVerilog–Unified Hardware Design, Specification, and Verification Language, 2013, https://doi.org/10.1109/IEEESTD.2013.6469140.
8. 1800-2017 - IEEE Standard for SystemVerilog–Unified Hardware Design, Specification, and Verification Language, 2018, https://doi.org/10.1109/IEEESTD.2018.8299595.
9. 1800-2023 - IEEE Standard for SystemVerilog–Unified Hardware Design, Specification, and Verification Language, 2024, https://doi.org/10.1109/IEEESTD.2024.10458102.
10. Verilog-A Language Reference Manual, Analog Extensions to Verilog HDL, https://www.siue.edu/~gengel/ece585WebStuff/OVI VerilogA.pdf, 2024.
11. Verilog-AMS Language Reference Manual, version 2.4.0, June 2024, [online] Available: https://www.accellera.org/images/downloads/standards/v-ams/VAMSLRM-2-4.pdf
12. Verilog-AMS standard, June 2024, [online] Available: https://www.accellera.org/downloads/standards/v-ams
13. Saber(R) MAST Language User Guide, Synopsys, 2008.
14. SystemVerilog AMS (Analog/Mixed-Signal) Working Group, June 2024, [online] Available: https://www.eda.org/activities/workinggroups/system-verilog-ams
15. What is Free Software?, September 2024, [online] https://www.gnu.org/philosophy/free-sw.en.html
16. Open Source Initiative, September 2024, [online] https://opensource.org/
17. GCC, the GNU Compiler Collection, September 2024, [online] Available: https://gcc.gnu.org/
18. Welcome to Verilator, September 2024, [online] Available: https://www.veripool.org/verilator/
19. Ho, C.-W., Ruehli, A., and Brennan, P., The Modified Nodal Approach to Network Analysis. Proc. 1974 Int. Symposium on Circuits and Systems, San Francisco. pp. 505–509.
20. BSIM Group, June 2024, [online] Available: http://bsim.berkeley.edu/
21. VA-BSIM48, June 2024, [online] Available: https://github.com/cogenda/VA-BSIM48
22. Compact Model Coalition, June 2024, [online] Available: https://si2.org/cmc/
23. Weil, P., McNamee, L., Simulation of excess phase in bipolar transistors. IEEE Trans. Circuits Syst., vol. 25 (1978), pp. 114-116.
24. Ward, D. E., Dutton, R. W., A charge-oriented model for MOS transistor capacitances. IEEE J. Solid-State Circuits, vol. 13 (1978), pp. 703-708.
25. Yang, P., Epler, B. D., Chatterjee, P. K., An Investigation of the Charge Conservation Problem for MOSFET Circuit Simulation, IEEE J. Solid-State Circuits, vol. 18 (1983), pp. 128-138.
26. Spectre Circuit Simulator Reference, Cadence, 2003.
27. Steer, M. B., Kriplani, N. M., Luniya, S., Hart, F., Lowry, J., Christoffersen, C. E., fREEDA: An Open Source Circuit Simulator, 2006 International Workshop on Integrated Nonlinear Microwave and Millimeter-Wave Circuits, IEEE, January 2006.
28. Lemaitre, L., McAndrew, C., Hamm, S., ADMS-automatic device model synthesizer, Proceedings of the IEEE 2002 Custom Integrated Circuits Conference, IEEE, May 2002.
29. HSPICE MOSFET Models Manual, Synopsys, 2005.
30. Extensible Markup Language (XML), June 2024, [online] Available: http://www.w3.org/XML.
31. Glib-2.0, June 2024, [online] Available: https://docs.gtk.org/glib/
32. XSL transformations (XSLT), June 2024, [online] Available: http://www.w3.orgITR/xslt
33. Ngspice - open source spice simulator, June 2024, [online] Available: https://ngspice.sourceforge.io/
34. Gnu Circuit Analysis Package - Git Repositories, June 2024, [online] Available: https://savannah.gnu.org/git/?group=gnucap

35. Circuit simulator of the Qucs project, September 2024, [online] Available: https://github.com/Qucs/qucsator
36. Xyce - Parallel electronic simulation, June 2024, [online] Available: https://xyce.sandia.gov/
37. Xyce/ADMS Users Guide, June 2024, [online] Available: https://xyce.sandia.gov/documentation-tutorials/xyce-adms-users-guide/
38. Ngspice user's manual (version 38), June 2024, [online] Available: https://ngspice.sourceforge.io/docs/ngspice-38-manual.pdf
39. ADMS is a code generator for the Verilog-AMS language, June 2024, [online] Available: https://github.com/Qucs/ADMS
40. Kuthe, P., Müller, M., Krattenmacher, M., Schr¨oter, M., OpenVAF Verilog-A Compiler, MOSAK February 25 2022, February 2022.
41. VerilogAE: An Open Source Verilog-A Compiler for Compact Model Parameter Extraction, Journal of Electron Devices Society, vol. 8 (2020), pp. 1416-1423.
42. OpenVAF - Performance, June 2024, [online] Available: https://openvaf.semimod.de/docs/details/performance/
43. Bűrmen, A., VACASK: a Verilog-A Circuit Analysis Kernel, Free Silicon Conference 2024, September 2024, [online] Available: https://wiki.fsi.org/index.php?title=VACASK: a Verilog-A Circuit Analysis Kernel
44. Bűrmen, A., VACASK - Verilog-A Circuit Analysis Kernel, September 2024, [online] Available: https://codeberg.org/arpadbuermen/VACASK
45. Modelgen-Verilog GIT repository, June 2024, [online] Available: https://git.savannah.gnu.org/cgit/gnucap/gnucap-modelgen-verilog.git
46. Open Source Device Interface (OSDI) Specification – working Draft, SemiMod, 2022, June 2024, [online] Available: https://openvaf.semimod.de/osdi/osdi v0p3.pdf
47. The LLVM Compiler Infrastructure, June 2024, [online] Available: https://llvm.org/
48. Puhan, J, B˝urmen, ´A., Fajfar, I., Tuma, T., Spice Opus June 2024, [online] Available: http://www.spiceopus.si/
49. Advanced Design System, 2024, June 2024, [online] https://www.keysight.com/us/en/products/software/pathwave-designsoftware/pathwave-advanced-design-system.html
50. Salfelder, F., Verilog-AMS in Gnucap, Free Silicon Conference (FSiC) 2023, June 2024, [online] Available: http://felix.salfelder.org/gnucap/fsic gnucap23.pdf
51. The Trilinos Project Website, June 2024, [online] Available: https://trilinos.github.io
52. Verley, J., Keiter, E., Xyce/ADMS and the Xyce Verilog-A Path Forward, Q1 2022 MOS-AK Panel, June 2024, [online] Available: https://mosak.org/panel Q1 2022/presentations/Verley Xyce MOS-AK Q1 2022.pdf
53. Chandra, R., et.al., Parallel Programming in OpenMP, Morgan Kaufmann, 2000.
54. Sanders, J., Kandrot, E., CUDA by Example: An Introduction to General-Purpose GPU Programming, Addison-Wesley Professional, 2010.
55. Kundert., K., SPARSE 1.3, June 2024, [online] Available: https://www.netlib.org/sparse/
56. Davis, T. A., Natarajan, E. P., Algorithm 907: KLU, a direct sparse solver for circuit simulation problems. ACM Trans. Math. Softw., vol. 37 (2010), pp. 1-17.

Arrived: 22. 07. 2024
Accepted: 09. 10. 2024