Modeling Tools Built Upon the HDL Foundation

H. Alan Mantooth

Department of Electrical Engineering, University of Arkansas, Fayetteville, AR, USA 72701

Abstract

Hardware description languages, mainly Verilog and VHDL including their analog and mixed-signal extensions, represent a significant investment by the electronic design automation community. Hardware description language technology promises productivity advances such as a medium for intellectual property exchange, model portability, model productivity, greater design collaboration, top-down design for AMS, AMS synthesis, and richer mixed-level, mixed-signal simulation for improved simulation throughput. Modeling tools represent a step toward completion of the dwelling the EDA community seeks to build upon the hardware description language foundation. One such environment of tools is described in this tutorial on Paragon[†]. Paragon will be described and demonstrated on both behavioral models using multiple HDLs and compact device modeling applications involving Verilog-A primarily.

Keywords

Paragon, ModLyng™, modeling tools, hardware description language

1. Introduction and the HDL Foundation

In 1987, the author first used the MAST® modeling language in what came to be known initially as an analog hardware description language (HDL) and later as a mixed-signal HDL. MAST was a proprietary language and was used in conjunction with the Saber simulator [1]. MAST demonstrated distinct advantages for analog and mixed-signal modeling by allowing the user to focus primarily on model behavior issues, not the entrapments of the simulator's algorithms or data structures. Saber and MAST together represented the first widely available circuit simulation platform that separated models from the simulator in this liberating fashion.

Digital HDLs were already standardized by the late 1980s. Two languages emerged in the market: VHDL and Verilog. These languages were seen as valuable to multi-level digital hardware representation, but the biggest impetus to their use came when additional tools such as logic synthesis came of age. VHDL and Verilog were/are the input languages for these tools and the design advantages of learning and using such a language became too compelling to ignore.

In the early 1990s the efforts toward standardization of analog extensions to the standard digital HDLs began. The first effort culminated in 1999 with the approved standard analog and mixed-signal extensions to VHDL [2]. This new language is referred to as VHDL-AMS and is a superset of VHDL. Later such extensions were added and approved to Verilog [3]. This language is known as Verilog-AMS and is a superset of Verilog. However, during the 1990s a different language emerged from Cadence to address the modeling of analog behavior beyond that of SPICE-like netlists much as MAST had done with Saber. This language is known as Verilog-A [4]. Recently, Verilog-A had extensions added to it in support of compact modeling of semiconductor devices [5].

Clearly, a substantial investment has been made by members of the electronic design automation (EDA) community (industry and academia) in defining and creating viable HDL technology. These investments have been made by the "language lawyers"[1] for standardization, model developers for libraries, and simulator architects for platforms that enable model separation from analog circuit simulation algorithms. With such a significant investment, it is imperative that the returns and the promises be realized. And yet, as recently as this year, two major IEEE conferences have had forums where the value of analog/mixed-signal (AMS) HDL technology has been debated [6, 7]. Such debates no longer rage regarding the digital HDLs. They have become part of the fabric of digital design and are taught in most electrical and computer engineering curricula. In fairness, both of these forums were held at conferences primarily focused on a single market segment – the semiconductor market and IC design – but the fact that the EDA industry is even asking the question is an indictment. Certainly, analog synthesis, a very difficult prospect, would do for AMS HDLs what digital synthesis did for Verilog and VHDL, but that solution has not been achieved yet.

AMS technology has been more widely adopted in the transportation industry where the ability to concurrently model and simulate multiple disciplines (e.g., electro-mechanical, electro-thermal, electro-hydraulic) has been a big advantage, but even in this arena there are barriers to adoption. AMS flows have been adopted by some IC design houses. Many view it as the future of analog/mixed-signal design, but the

[†] Portions of this research were supported by the National Science Foundation under grants EEC-0088011, the Office of Naval Research under Subaward No. USC 01-636, DARPA under grant N66001-01-1-8919, and the SRC under grant SRC05-463.

[1] a glib reference to those highly skilled experts that are adept at defining consistent language semantics to the finest degree of detail while also being clairvoyant enough to perceive future pitfalls if such issues are not dealt with properly

adoption has been lethargic. One explanation for this glacial movement is that by itself AMS HDLs do not represent a significant productivity enhancement as compared to how analog IC designers achieve tape-out today. A second, related explanation is that analog IC designers are still successful doing it "like they've always done it" even with added time-to-market pressures, increased performance demands driven by embedded digital systems, and challenges posed by constantly advancing process technology. Finally, another explanation is that the deployment of AMS HDL technology has not yet reached a level of comfort for the analog IC design community to feel it is ready to be relied upon. They use SPICE for transistor level analysis and are heavily invested in its use. They rely on transistor models derived from the foundry to verify that their designs perform as desired and work over process corners and temperature. They have an increasing need for behavioral modeling, but the barriers to the adoption of AMS HDL technology as the solution include:

(1) the learning curve for the HDL and
(2) integration with their favored SPICE tools because they need transistor simulations along with their behavioral models.

Barrier (2) above is a simulator feature issue and not the topic of this paper. However, it is important to point out that many AMS HDL simulators and their associated environments have yet to achieve analysis and post-simulation analysis features found in SPICE tools. This certainly does not encourage adoption. The good news is that this is an active area of development and will be addressed.

To delve deeper into barrier (1) above, it is not merely the learning of a new language that poses an obstacle – although for the majority of analog designers this is real – it is in understanding how simulators work internally and composing the model in a way less likely to cause problems. Another part of the learning curve of AMS HDLs is adjusting their thinking from that of a schematic-driven functional way of thinking to an expression-based approach. To illustrate these points consider that an analog IC designer knows that a pair of diodes can provide a limiting function in a model they are creating, but the associative *if-then-else* structure in an HDL is easily misconstructed such that either the simulator will have convergence problems or, more fundamentally, it simply is not what the designer intended. Frustration mounts very quickly in a designer when simulations of a model fail to converge, particularly when this model is not even the topic of design. It may merely be a representation of a bias circuit or a dynamic load configuration needed in order to proceed with the design activity. Frustration continues to mount when very little exists in the way of tools and methods to diagnose and remedy such problems. In truth, these issues exist for all users of AMS HDL technology today and will be the focus of the modeling tools described in this paper.

The next section of the paper will describe the modeling tool Paragon including what it is, how it is used, and how it addresses the elements of barrier (1) [8, 9]. Section 3 focuses on a couple of examples while Section 4 concludes by reviewing some of the promises that HDL technology makes and how Paragon helps make these promises come true.

2. The Paragon Modeling Tool

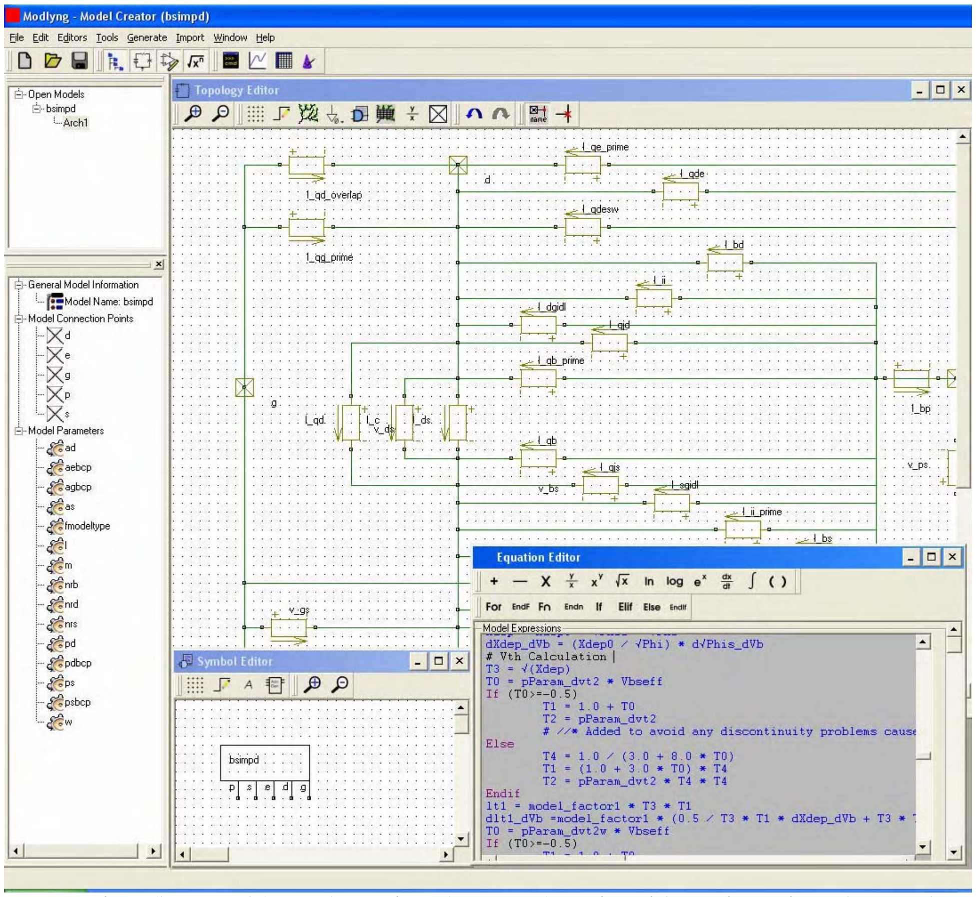
Paragon[2] is a graphical modeling environment that consists of interfaces that segment the modeling process along logical macromodeling lines of thinking. It utilizes a mixture of schematic, browsing and textual interfaces as illustrated in Fig. 1 to expedite model creation and visualization of the effects present within a model. While Fig. 1 is the illustration of a semiconductor device model, Paragon is easily applied to the modeling of higher-level circuit and system blocks. Both will be described in the examples in Section 3.

For the purpose of describing the use of Paragon, an "outside-in" modeling process that begins with describing the model interface (i.e., connections or ports, name of model, model parameters) and flows to the internal structural and behavioral details will be followed. However, no restrictions are placed on the user as to what modeling steps are performed when. Referring again to Fig. 1, the leftmost portion of the screenshot illustrates the browser/editor for loading models (upper left) and editing the interface of the current model (lower left). A special set of dialogs appear for creating or modifying model parameters that allow the user to specify model parameter types, units, default values, and numerical ranges of validity. When appropriate, such as in the case of units and types, built-in or standard values are accessible from the dialog. In deference to semiconductor device modeling, Paragon also allows the user to distinguish between process parameters and instance parameters for the model. In fact, Paragon is able to import a .MODEL card to expedite the task of model parameter entry and defaulting for process parameters.

Models consist of structure (or topology) and behavior in general. The top-right portion of the screenshot of Fig. 1 shows a schematic view of the internal structure of the model being created. This window is referred to as the topology editor. The behavior is reflected in the expressions (lower right) used to describe either a block in the structure or a set of expressions that are used to derive internal variables from the model parameters and other model variables. A Paragon model can consist purely of equations and essentially no structure, a structure with associated equations as in Fig. 1, or a hierarchical model with a structure only (i.e., a macromodel).

Once the model equations, topology, and interface information are entered, the model code can be generated. However, Paragon contains some model checking functions that can be executed to determine

[2] Paragon is being commercialized by Lynguent, Inc. under a license agreement from the University of Arkansas. The commercial name of the tool is ModLyng^{TM}. All screenshots in this paper are taken from ModLyng.



```
dXdep_dVb = (Xdep0 ⁄ √Phi) * d√Phis_dVb
# Vth Calculation |
T3 = √(Xdep)
T0 = pParam_dvt2 * Vbseff
If (T0>=-0.5)
        T1 = 1.0 + T0
        T2 = pParam_dvt2
        # //* Added to avoid any discontinuity problems cause
Else
        T4 = 1.0 ⁄ (3.0 + 8.0 * T0)
        T1 = (1.0 + 3.0 * T0) * T4
        T2 = pParam_dvt2 * T4 * T4
Endif
lt1 = model_factor1 * T3 * T1
dlt1_dVb =model_factor1 * (0.5 ⁄ T3 * T1 * dXdep_dVb + T3 * 
T0 = pParam_dvt2w * Vbseff
If (T0>=-0.5)
```

Fig. 1. Equation editor (lower right), topology editor (top center), and model interface editor showing the partially depleted BSIMSOIv2.2 MOSFET model.

continuity of key functions in the model or to perform consistency checking of equations and variables.

If a MAST, Verilog-A, or VHDL-AMS model already exists, then it can be imported into Paragon[3]. Recently, Modelica models can be imported through a tool written by a partner institution in Russia [10]. At this point the model topology may need to be arranged in a logical fashion, but the model is present and can be easily modified, analyzed and eventually *debugged* in this environment. Of course, the model can be output in any of the languages supported by Paragon if no language-specific concepts have been employed. If a Verilog-AMS model is imported and it does employ a semantic not representable in VHDL-AMS or other languages, then only Verilog-AMS code generation will be enabled. In this way, even though a guiding principle behind Paragon's concept is language independent model entry, it does not have to behave as a "common denominator" modeling environment. It is designed to operate in the context of the major HDLs as required.

In order to address model checking issues and to reduce simulation time, Paragon has a rich and growing set of analysis and utility methods. An abstract syntax tree (AST) is created for each model to identify and represent the inter-relationships between different time-varying variables and constants of the model. The AST is analyzed to determine functional and time dependencies in the model. This enables the generation of efficient and readable code. The model import mechanism also utilizes a combination of the above mentioned utilities and distinguishes between sequential and simultaneous blocks of the model. It also identifies undefined variables before actually saving the model to the database. These and other features differentiate the model importer from basic language translators. So far, continuous time models have been the focus for model importing while research effort is ongoing to include event-driven behavior.

A significant advantage of Paragon is that it produces readable, standardized hardware description language code that captures design intent and removes the common implementation errors that result in unnecessary iterations. Paragon can generate Verilog-A, Verilog-AMS, VHDL-AMS, MAST, and some C-based languages for special purpose simulators. Paragon models are easier to support than hundreds or thousands of lines of code due to the mixed graphical and textual

[3] Paragon only supported MAST import, but ModLyng supports the import of all three languages.

representation in the tool. The learning curve for these tools is not steep due to the fact that Paragon is exceptionally intuitive to use as a result of using familiar user interface approaches such as schematic capture.

Outputs such as symbols for use in design environments and model documentation in HTML format can be generated from Paragon as well to expedite model use. Paragon utilizes a generic XML schema, which enables the capture of information specific to model data. The use of XML, which is open source and a standardized format, allows easy data interchange and formatting. Many standard language translation tools like Extensible Style Sheet Language Transformations (XSLT) [11] can be used to manipulate the data and convert it into a desired target format.

In addition to the HDL code generation functionality, Paragon can also be used in combination with other tools like the ADMS [12, 13] and MCAST [14] model compilers to generate low level C code for target simulators. One can imagine the amount of time, effort and complexity involved in writing and debugging huge models such as those in the BSIM family in either C or an HDL like Verilog-A before actually deploying in simulations. The usefulness and capacity of high level advanced modeling tools like Paragon was illustrated by generating two BSIMSOI models (version 2.2 and 3.2) and validating against the built-in Spectre model in [15]. Another example of Paragon's capacity is illustrated by the VBIC implementation as described in [16]. In each case, the Verilog-A code generated by Paragon was fed into ADMS to compile it into C for the compiled model interface of Spectre.

3. Examples

The first example to be described is a semiconductor device model (BSIMSOI). This leverages off of the information presented in Fig. 1. Several state-of-the-art device models have been implemented over the past year including BSIMSOI, EKV, VBIC, and a SiC JFET power device model [17-21].

The BSIMSOI model consists of over 15,000 lines of C code as implemented for SPICE. It represents the most significant stress test for Paragon due to its complexity and size. The BSIMSOI model consists of as many as six external pins (or ports): drain (d), gate (g), source (s), external body (p), back gate (e), and an optional thermal pin. The thermal pin was not required for the project that this model was created for. An additional internal body node (b) exists within the model as can be seen in the model topology of Fig. 1 on the far right-hand side. The model creation process was started by entering the ports and then importing a .MODEL card populated with the default values of all process parameters. This import process resulted in the *fmodeltype* entry in the model parameters list of Fig. 1. The instance parameters were input directly using the user interface.

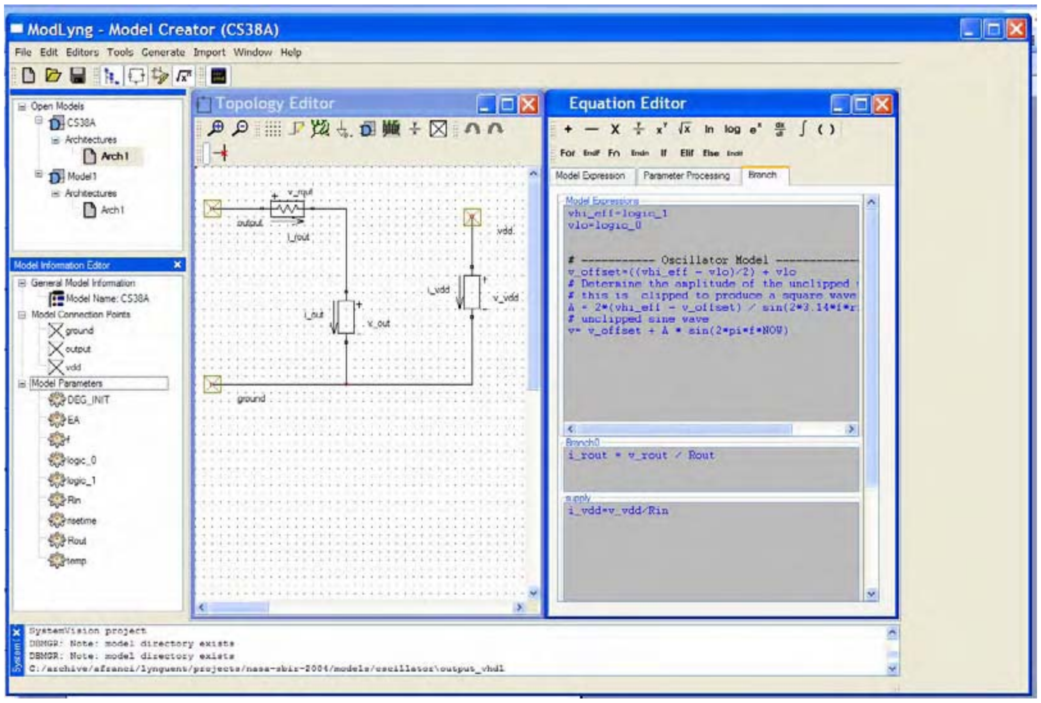
Next, all of the model expressions involving model parameter manipulation were entered into the equation editor. These expressions are easily stripped out from the Berkeley C code and pasted into the editor for speed and syntactical correctness. At this point, the longest task of the modeling effort is the input of the model topology and the equations that go with those blocks (i.e., branch constitutive equations) or the equations that algebraically compute internal variables needed for the branch constitutive equations. For a model the size of BSIMSOI this takes a couple of days normally. The challenge is that in Paragon one is capturing the model in more of a macro or object-oriented style using the topology as the driving implementation vehicle. Whether the original Berkeley model is implemented from such a diagram or not, this object-orientation is not reflected in the C implementation. Therefore, up until now, this has been a manual process that is error-prone requiring careful model validation against a known good model. This validation typically takes a few days itself and more if serious errors have been made.

This situation has led us to develop a Verilog-A model import mechanism that is currently being tested with BSIM3 and BSIM4. The development has focused on Verilog-A for two main reasons. Verilog-A is readily generated from the Berkeley C code and it is commonly used as an implementation format for semiconductor device models. This will reduce the implementation time in Paragon for *existing* semiconductor models to a couple of hours allowing for manual rearrangement of the default place-and-route of the model topology that gets automatically generated.

Once the model is implemented in Paragon the utility of the environment can be brought to bear on the device model. For example, the continuity checking algorithm can be employed to analyze complex equations in the model. Relationships within the model can be visualized. Alternative output formats can be considered such as VHDL-AMS. Enhancements to the model can be considered by a larger audience of model developers than those familiar with SPICE codes. The project that led to the BSIMSOI version 2.2 implementation was focused on adding radiation effects to the base model. These effects included total dose and single event effects. These enhancements were made and the model regenerated in Verilog-A. The model was then compiled using ADMS and linked into Spectre through its compiled model interface.

To conclude the semiconductor device modeling illustration, once the topology, equations and interface are input, the code can be generated in Verilog-A, VHDL-AMS, and MAST at the present time. By using model compilers these codes can be converted into C and therefore made to run quite efficiently.

The second example is the behavioral model creation of a commercial-off-the-shelf (COTS) quartz crystal oscillator. A behavioral model of the C38SA device (200 °C, 7.3728MHz) from Sentry Manufacturing was required as part of a larger, high temperature system. The basic behavioral model was to be modified to include temperature and performance degradation effects as a function of time and temperature (i.e., aging effects).



```
vhi_eff=logic_1
vlo=logic_0


# ----------- Oscillator Model ------------
v_offset=((vhi_eff - vlo)/2) + vlo
# Determine the amplitude of the unclipped
# this is  clipped to produce a square wave
A = 2*(vhi_eff - v_offset) / sin(2*3.14*f*r
# unclipped sine wave
v= v_offset + A * sin(2*pi*f*NOW)
```

```
i_rout = v_rout / Rout
```

```
i_vdd=v_vdd/Rin
```

```
SystemVision project
DBMGR: Note: model directory exists
DBMGR: Note: model directory exists
C:/archive/afranci/lynguent/projects/nasa-sbir-2004/models/oscillator\output_vhdl
```

Fig. 2. C38SA quartz oscillator model as defined in Paragon.

Fig. 2 shows the screenshot of the base oscillator model without thermal or aging effects. A much simpler model conceptually than the semiconductor device model, it still consists of the same basic elements: interface, topology and equations. In this case the output voltage, frequency and waveshape are modeled according to experimental measurements made of the device. The jitter of this oscillator was insignificant (even over temperature) and thus neglected in the model. In this case, the model was generated using VHDL-AMS and simulated using SystemVision^{TM}. Fig. 3 shows the basic simulation results as compared to measurements. While the ringing was not captured in this simple model, the basic functionality is there.

Next, the lognormal behavior of the oscillator model's output high voltage as a function of time and temperature was modeled according to [22]. This amounted to adding a few extra equations and parameters to the model. The limited experimental data taken allowed for a basic aging model for the oscillator. The oscillator does not operate for very long beyond its 200 °C rating. The experimental data indicated that the leading indicator of failure occurred when the output voltage dropped by 20 %, indicating that oscillator failure was imminent. The voltage drops rapidly once it falls by 20 %, analogous to the voltage on a battery when discharging. The nominal output voltage was about 5 V.

The oscillator model with aging was simulated at 220 °C and 250 °C and accurately predicted oscillator failure at these temperatures. For 220 °C the device failed at approximately 73 hours and at 250 °C it failed at 15.5 hours – both within a few percent of the measured data. Another experimentally validated simulation predicted that if the oscillator was first operated for 10 hours at 250 °C, then it would only operate for approximately 35 hours at 220 °C before degrading to a point of failure thus indicating that the model was capable of predicting thermally-induced failures as a function of time at temperature.

4. Conclusions

This tutorial paper has described the use of the Paragon modeling tool that is built to operate on the HDL foundation and reduce the barriers to widespread usage of these technologies by making them easier to effectively use, improve the resulting models created, and provide a richer set of debugging capability in the future. By way of conclusion it is instructive to review some of the promises that HDL technology espouses to see where things stand at this point. A representative list is shown below:

1. Medium for intellectual property (IP) exchange
2. Model portability
3. Design collaboration (ease of collaboration between customers-suppliers, companies, design groups)
4. Top-down design for AMS (medium for expressibility of analog and mixed-signal blocks during design exploration phase)
5. Analog Synthesis (AMS needed for the models)
6. Mixed-level and Mixed-signal Simulation

From the six items listed above, modeling tools such as Paragon help to directly address four of them (2, 3, 4, 6). Modeling tools such as Paragon help to insure model portability, encourage design collaboration and top-down design methods because of the ease of making models and sharing them (even if encrypted), and certainly promote mixed-level simulation. The choice of whether to use HDLs as a medium for IP exchange may depend on a number issues, but certainly once the decision is made to use HDLs then modeling tools will greatly help to facilitate IP exchange in the same way encrypted models can be delivered from customer to supplier, for example. Lastly, experts in analog synthesis have indicated a need for modeling tools to enable their simulation-based approaches. However, tools like Paragon need more research and additional utility to address these requirements because they are driven from a bottom-up behavioral model generation need such as described in [23].

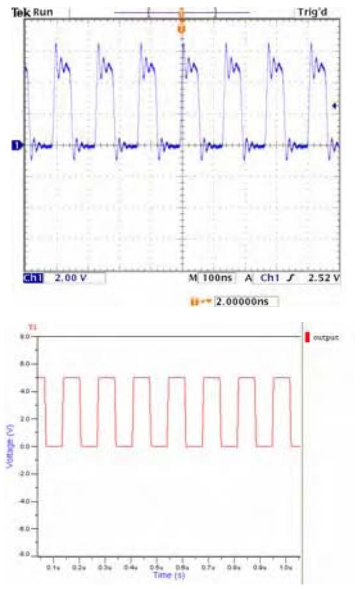


Fig. 3. Output waveforms of the C38SA quartz oscillator and the behavioral model.

5. Acknowledgements

The author would like to acknowledge and express his gratitude to all of the students that have contributed to the development of Paragon in the MSCAD Laboratory at the University of Arkansas over the past six years: O. Abbasi, A. Austin, V. Chaudhary, Y. Feng, M. Francis, C. Gathercole, H. Gunupudi, N. Hingora, X. Huang, J. Kutchka, P. Mallick, J. Mao, H. Nguyen, E. Pettis, C. Vemulapally, and W. Zheng.

6. References

[1] H. A. Mantooth and M. Fiegenbaum, *Modeling with an Analog Hardware Description Language*, Kluwer Academic Publishers, Norwell, MA, 1995.

[2] 1076.1-1999 IEEE Standard VHDL Analog and Mixed-Signal Extensions Language Reference Manual, IEEE Press, ISBN 0-7381-1640-8.

[3] Accellera, "Verilog-AMS Language Reference Manual – Analog & Mixed-Signal Extensions to Verilog HDL," Version 2.2, Nov. 2004. http://www.eda.org/verilog-ams

[4] D. Fitzpatrick, I. Miller, *Analog Behavioral Modeling with Verilog-A*, Kluwer Academic Publishers, Norwell, MA, 1997.

[5] Accellera, "Proposed Verilog-A Language Extensions for Compact Modeling," version 9, 23 pgs, Aug. 2004. http://www.eda.org/verilog-ams

[6] Accellera Breakfast and Panel Discussion, "Design and Verification: Can the Analog/Mixed-Signal (AMS) Standard Bridge the Chasm?," Anaheim Hilton, Anaheim, CA, June 15, 2005.

[7] CICC Panel Discussion, "Analog Behavioral Modeling: Fantasy, Fad, or Foundation for the Future?," Doubletree Hotel, San Jose, CA, Sept. 20, 2005.

[8] V. Chaudhary, M. Francis, X. Huang, H. A. Mantooth, "Paragon - A mixed-signal behavioral modeling environment," *IEEE Int. Conf. on Communications, Circuits, & Syst. (ICCCAS)*, vol. 2, pp. 1315-1321, Chengdu, China, June 2002.

[9] P. Mallick, M. Francis, C. Vemulapally, A. Austin, H. A. Mantooth, "Achieving language independence with Paragon," *International Workshop on Behavioral Modeling and Simulation (BMAS)*, pp. 149-153, Oct. 2003.

[10] Y. Chernukhin, M. Polenov, C. Vemulapally, E. Solodovnik, H. A. Mantooth and R. Dougal, "Deploying Modelica models into multiple simulation environments," International Workshop on Behavioral Modeling and Simulation (BMAS) – these proceedings, 6 pgs., Sept. 2005.

[11] Extensible Stylesheet Language, http://www.w3.org/TR/xslt

[12] L. Lemaitre, C. McAndrew and S. Hamm, "ADMS – Automatic Device Model Sythesizer," *Proc. IEEE Custom Int. Circ. Conf.*, pp. 27-30, 2002.

[13] ADMS Model Compiler, http://sourceforge.net/projects/mot-adms

[14] B. Wan, B. P. Hu, L. Zhou and C.-J. Shi, "MCAST – An abstract-syntax-tree based model compiler for circuit simulation," *IEEE Custom Integrated Circuits Conf. (CICC)*, pp. 249-252, Sept. 2003.

[15] BSIM3SOI Source code and Documentation, http://www-device.eecs.berkeley.edu/~bsimsoi/

[16] VBIC BJT Model, http://www.designers-guide.org/VBIC/

[17] A. M. Francis, B. O. Woods, H. A. Mantooth, M. Vlach, L. Lemaitre, "A Methodology for Rapid Development and Simulator Integration of Compact Models," Proc. of SRC TECHCON 2005, 6 pgs., Oct. 2005.

[18] M. Francis, V. Chaudhary and H. A. Mantooth, "Compact modeling of semiconductor devices using higher level methods," *IEEE 2004 International Symposium on Circuits and Systems*, vol. 5, pp. v-109 - v-112, May 2004.

[19] V. Chaudhary, M. Francis, W. Zheng, H. A. Mantooth and L. Lemaitre, "Automatic Generation of Compact Semiconductor Device models using Paragon and ADMS," *International Workshop on Behavioral Modeling and Simulation (BMAS)*, pp. 107-112, Oct. 2004.

[20] A. S. Kashyap, C. Vemulapally and H. A. Mantooth, "VHDL-AMS Modeling of Silicon Carbide Power Semiconductor Devices," *IEEE Workshop on Computers in Power Electronics (COMPEL)*, pp. 50-54, Aug 2004.

[21] M. Bucher, C. Lallement, C. Enz, F. Théodoloz and F. Krummenacher, "The EPFL-EKV MOSFET Model Equations for Simulation," Technical Report, Model Version 2.6, June 1997. Revision I, September, 1997, Revision II, July, 1998.

[22] F. R. Nash, *Estimating Device Reliability: Assessment of Credibility*, ch. 7., Kluwer Academic Publishers, Norwell, MA, 1993.

[23] H. A. Mantooth, L. Ren, X. Huang, Y. Feng, W. Zheng, "A survey of bottom-up behavioral modeling methods for analog circuits," *IEEE Proc. Int. Symp. Circuits Syst.*, pp. 910-913, Bangkok, Thailand, May 2003.