

Quantifying Electrical System Worst-Case Performance Prior to Prototype Test and Production

Michael Jensen
Mentor Graphics Corp.

ABSTRACT

Electronics now control or drive a large part of automotive system design and development, from audio system enhancements to improvements in engine and drive-train performance, and innovations in passenger safety. Industry estimates suggest that electronic systems account for more than 30% of the cost of a new automobile and represent approximately 90% of the innovations in automotive design. As electronic content increases, so does the possibility of electronic system failure and the potential for compromised vehicle safety. Even when designed properly, electronics can be the weakest link in automotive system performance due to variations in component reliability and environmental conditions. Engineers need to understand worst-case system performance as early in the design process as possible. While traditional electronic design flows use modeling and simulation to analyze system performance prior to prototype, test, and production, limits in model fidelity and simulator capability limit worst-case design evaluation options. However, with the right combination of modeling techniques and simulation capabilities, it is possible to improve model and simulation fidelity to include advanced worst-case effects such as aging, and high and low temperature exposure. By modeling a basic diode and using a standard bridge rectifier circuit example, this paper illustrates important methods and considerations for modeling and analyzing the advanced worst-case performance of electronic systems.

CITATION: Jensen, M., "Quantifying Electrical System Worst-Case Performance Prior to Prototype Test and Production," *SAE Int. J. Passeng. Cars – Electron. Electr. Syst.* 9(1):2016, doi:10.4271/2016-01-0074.

INTRODUCTION

Industry analysts suggest that most automotive systems increasingly depend on some form of electronic control, processing, or monitoring, and that this trend shows no sign of slowing. [1] [2] With vehicle electronic content on the rise, development teams are challenged first with getting the nominal design right, and second with ensuring circuits and systems operate as expected, despite variations in manufacturing tolerances and the reaction to physical effects such as aging and thermal exposure.

Virtual prototyping, once an electronic system design luxury, is now a design flow necessity. Modeling and simulation let engineers ask and answer important system performance questions before committing resources to hardware prototype and test. Device/system models form the questions; simulation provides the answers. Getting the question right is just as important as calculating the answer. Design teams, therefore, need to ensure they select the best tools for asking and answering system questions, not only when designing to nominal system requirements, but also when considering worst-case performance questions caused by variations in design tolerances. The process requires choosing the right modeling platform, using the selected platform to create simulation models that account for tolerance variation and are equal to the worst-case analysis task, selecting simulation tools that adequately exercise the system model in the critical areas of interest, and automating the potentially laborious task of covering multiple worst-case performance scenarios.

MODELING PLATFORMS

SPICE format models are popularly available from many component vendors and often perform well for standard circuit analyses. However, SPICE models often have a limited performance range and therefore are not adequate for modeling variation and evaluating worst-case system performance. For detailed worst-case system evaluation, design teams need a more flexible language-based modeling approach.

There are many modeling language options to choose from. Some are proprietary to a tool vendor; others are very general and make the modeling process too difficult. There is a class of languages, however, that are tuned to hardware modeling. These are often referred to as Hardware Description Languages, or HDLs. Some HDLs cover a specific modeling discipline, such as Verilog and VHDL which are optimized for modeling digital behavior. Complex systems, however, often depend on a mix of design technologies and therefore require a more general purpose HDL. HDLs in this category support analog, digital, and mixed-technology modeling. While there are some vendor-specific languages available in this class, design teams are best served by choosing an industry standard language such as VHDL-AMS (IEEE Standard 1076.1) to avoid being locked-in to a single-vendor modeling solution. VHDL-AMS is tuned for modeling analog, mixed-signal, and mixed-technology system behavior and is used to illustrate the worst-case modeling techniques exemplified in this paper.

VHDL-AMS INTRODUCTION

VHDL-AMS supports modeling device behavior beyond nominal device physics, including the option to model component variability. It is a very structured yet flexible language with a broad range of features for modeling mixed-signal and mixed-technology systems. Using VHDL-AMS, design teams can move up and down the model abstraction ladder, from general functional level to detailed implementation level models, as well as across the model technology spectrum including electrical and non-electronic device modeling (sometimes referred to as "mechatronic system modeling").

VHDL-AMS models are divided into two sections: an ENTITY and an ARCHIECTURE (caps added for emphasis). The ENTITY defines how the model connects to other models in a system, and the parameters available to characterize the model's behavior. The ARCHITECTURE defines how the model behaves using equations, algorithms, and other behavioral descriptions. To illustrate these entity/architecture concepts, consider an ideal diode with the following mathematical description:

$$i = Isat * (e(v/vt) - 1.0) \quad (1)$$

Refer to the *Definitions/Abbreviations* section at the end of this paper for variable names and definitions used throughout this paper.

Assuming the two pins for the diode are named "p" and "n", the model entity can be written as:

```
entity diode is
  generic (
    v_knee : voltage      := 0.75;
    i_knee : current      := 50.0e-3;
    v_lin  : voltage      := 0.85;
    i_lin  : current      := 1.0
  );
  port (
    terminal p, n : electrical
  );
end entity diode;
```

Items in the *generic* section are parameters that users can define to customize the diode's performance. The architecture starts by defining the voltage and current relationships between the diode pins (p and n). After calculating useful constants, the architecture implements the diode equation shown in [Equation 1](#).

```
architecture ideal of diode is
  quantity v across i through p to n;
  constant Isat : current := i_lin/Exp(v_lin/vt);
  constant vt : voltage :=
    (v_lin - v_knee)/Log(i_lin/i_knee);
begin
  i == Isat*(Exp(v/vt) - 1.0);
end architecture ideal;
```

This diode model could be more complex, but is sufficient to illustrate basic modeling and simulation techniques for analyzing worst-case performance of electronic systems. A more detailed explanation of VHDL-AMS capabilities is beyond the scope of this paper. Internet and print resources are readily available, including the definitive language reference *The System Designer's Guide to*

VHDLAMS [3], for more information. VHDL-AMS is supported by several commercially available simulators including the SystemVision® modeling and analysis environment (Mentor Graphics®), Saber® (Synopsys®), and Simplorer® (ANSYS®).

MODELING MANUFACTURING VARIABILITY

Key to analyzing system worst-case performance is modeling variability. Non-language based modeling techniques such as SPICE have limited flexibility. But languages like VHDL-AMS support features that help engineers model complex device behavior. To illustrate this flexibility, in this section the diode model example is improved by adding tolerances to the knee voltage. For simulation, tolerances and distributions are typically defined for a parameter by adding information to a component's symbol. More control over variability is possible, however, when manufacturing tolerances and distributions are integrated into the model definition.

For maximum flexibility, the model should support definitions for asymmetric tolerances about a parameter's nominal value. For the diode model, this is simply done by adding parameters to the generics section that define plus and minus tolerances for a selected parameter, in this case the knee voltage:

```
v_knee_tol_hi : real := 5.0;
v_knee_tol_lo : real := 5.0;
```

In this case, the values are defined as real numbers, are assumed to represent a percentage, and are each assigned a default of 5%. The generic section in the model becomes:

```
generic (
  v_knee : voltage      := 0.75;
  i_knee : current      := 50.0e-3;
  v_lin  : voltage      := 0.85;
  i_lin  : current      := 1.0;
  v_knee_tol_hi : real := 5.0;
  v_knee_tol_lo : real := 5.0;
);
```

These tolerance definitions can be combined within the model architecture in different ways to affect the nominal value of the diode's knee voltage. For this paper, the percentages will be treated as a tolerance pair, which can be represented in the architecture as:

```
constant v_knee_tol_hi_sum : real := v_knee_tol_hi;
constant v_knee_tol_lo_sum : real := v_knee_tol_lo;
constant v_knee_tolpair : tolerance_pair :=
  (v_knee_tol_lo_sum/100.0,
   v_knee_tol_hi_sum/100.0);
```

Here the *_sum constants represent the sum of multiple tolerances, which is an easy way to support additional tolerance variations due to effects such as aging and high and low temperature effects - see the *Age Tolerancing* section in this paper. These *_sum constants are then combined in a constant called v_knee_tolpair of type tolerance_pair. Note that the v_knee_pair constant divides each of the *_sum constants by 100, turning the percent into a number between 0.0 and 1.0 for calculating the toleranced values used to increase or decrease the nominal knee voltage.

Note: The SystemVision simulator from Mentor Graphics is used to simulate the examples in this paper; and the tolerance_pair type is defined in a separate VHDL-AMS package for this simulator. A discussion of packages is beyond the scope of this paper, but details are available in the language reference mentioned earlier.[3]

For the next step, this tolerance pair is used to calculate knee voltage values for the diode. Due to the nature of statistical variation, the knee voltage tolerances must be combined with distribution information in order to properly determine the statistical spread of knee voltage values. For this paper, a *normal probability density function* (i.e. normalpdf) is applied to values for the knee voltage:

```
constant v_knee_pdf : voltage :=
  normalpdf(v_knee, v_knee_tolpair,
    "v_knee_pdf", "voltage");
```

Note: As with the tolerance_pair type mentioned earlier, the "normalpdf" distribution is defined in a separate VHDL-AMS package for the SystemVision simulator. For a general discussion about a VHDL-AMS statistical package, refer to SAE Standard J2748 [4].

This new knee voltage value must now be added to the diode performance calculations. The original value for the diode thermal voltage was calculated as:

```
constant vt : voltage :=
  (v_lin - v_knee)/Log(i_lin/i_knee);
```

Substituting the new v_knee_pdf calculation for v_knee, the calculation for vt becomes:

```
constant vt : voltage :=
  (v_lin - v_knee_pdf)/Log(i_lin/i_knee);
```

Adding these constant calculations to the original diode model, the new architecture becomes:

```
architecture ideal of diode is
  quantity v across i through p to n;
  constant vt : voltage :=
    (v_lin - v_knee_pdf)/Log(i_lin/i_knee);
  constant Isat : current := i_lin/Exp(v_lin/vt);
  constant v_knee_tol_hi_sum : real :=
    v_knee_tol_hi;
  constant v_knee_tol_lo_sum : real :=
    v_knee_tol_lo;
  constant v_knee_tolpair : tolerance_pair :=
    (v_knee_tol_lo_sum/100.0,
     v_knee_tol_hi_sum/100.0);
  constant v_knee_pdf : voltage :=
    normalpdf(v_knee, v_knee_tolpair,
      "v_knee_pdf", "voltage");
begin
  i == Isat*(Exp(v/vt) - 1.0);
end architecture ideal;
```

The diode model can now be used to simulate parameter variation, in this case the knee voltage, due to variations in manufacturing processes. Manufacturing variability is an important consideration in device performance, but variations due to physical effects are equally important for systems where safety and high-reliability are critical.

MODELING PHYSICAL EFFECTS VARIABILITY

With the diode model setup to account for manufacturing variability, the next step is to model variation due to physical effects. Physical effects can include many phenomena such as variability due to device aging, and high and low temperature exposure. This section illustrates how to model these effects by adding aging effects to the diode's knee voltage parameter. Here the aging effect is assumed to be a change in parameter tolerance, but could easily be represented by more complex in-model calculations. For simplicity, assume aging is characterized by a tolerance increase, and is represented by an additional user defined parameter added to the generics section with a default of 10%:

```
v_knee_tol_age : real := 10.0;
```

This definition represents a symmetrical tolerance about the nominal knee voltage value, though an asymmetric tolerance could be defined by adding an additional aging tolerance and then specifying which tolerance represents a knee voltage increase, and which represents a knee voltage decrease.

Adding the aging tolerance to the *_sum values in the architecture is a simple operation, but then aging would be included in all statistical calculations, which is not the desired behavior. The age influenced tolerancing should be defined in a way that supports toggling the effect on and off. Adding a single user defined parameter to the entity, and a simple calculation to the architecture, enables user control of the aging tolerance. The entity addition represents a simple "toggle" parameter that can be set to 0.0 or 1.0:

```
age_enable : real := 0.0;
```

The age_enable parameter's default is 0.0 so the age tolerancing effect is not included in normal statistical calculations. To control whether or not the aging tolerance is included in a statistical analysis, the v_knee_tol_age parameter is multiplied by the age_enable value using a simple equation in the diode model's architecture:

```
v_knee_tol_age_val : real :=
  age_enable * v_knee_tol_age;
```

Now the change in knee voltage tolerance due to aging can be added to the change in tolerancing defined earlier. The definitions for the

```
constant v_knee_tol_hi_sum : real :=
  v_knee_tol_hi + v_knee_tol_age_val;
constant v_knee_tol_lo_sum : real :=
  v_knee_tol_lo + v_knee_tol_age_val;
```

With these simple additions to the entity and architecture, the diode model can now be used to simulate the effects of manufacturing variation, as well as the added variation due to the physical effect of aging.

These modeling techniques can be applied to any model parameter. To summarize, the general steps for adding manufacturing variation to a model parameter include:

1. Create the nominal performance model. Include parameters so the model can be customized to perform as a specific device.