Statistical Eye Analysis Implemented in VHDL-AMS

Arpad Muranyi
Mentor Graphics Corporation
8383 158th Avenue NE, Suite 350
Redmond, WA 98052
(503) 685-0440

arpad_muranyi@mentor.com

ABSTRACT
This paper introduces the reader to a VHDL-AMS (an IEEE-endorsed standard modeling language, standard 1076.1) implementation of the basic Statistical Eye Analysis algorithm. The main purpose of this paper is to show how such algorithms can be implemented using VHDL-AMS, and not to discuss the derivation details, novelty or usefulness of the algorithms. The example code that accompanies this paper is fully functional, but many more features and refinements need to be added to it to make it ready for real-world design work. The topic of this paper is essentially a continuation of the presentation given by the author on "Peak Distortion Analysis Implemented in VHDL-AMS"[1].

1. INTRODUCTION
While Peak Distortion Analysis (PDA)[2] is useful to predict the worst possible eye opening of a communication channel, PDA cannot tell what the probability is of getting an eye size like that, or what the Bit Error Rate (BER) is for that channel. However, the principles used in the PDA algorithms can be combined with statistical techniques to predict the probability of various eye openings, and/or the BER of the channel. Numerous analysis tools have been developed by various organizations and companies using such algorithms, but most of them are written in general purpose programming languages (such as C, C++ or similar), or various mathematical software packages (such as MATLAB® or Mathematica®). The implementation of these algorithms in the VHDL-AMS modeling language makes statistical algorithms and models directly available in Signal Integrity (SI) analysis tools. In addition, since VHDL-AMS is a standard hardware description language, these models are portable and compatible between SI tools.

The VHDL-AMS code presented in this paper is based on the algorithms described in the popular open source statistical analysis package, StatEye, which was initially developed using MATLAB® by Anthony Sanders and Edoardo Prete of Infineon[3]. As stated in the Abstract, this paper does not attempt to provide a complete replica of all the features and capabilities found in StatEye. The primary goal is to provide a good starting point for further development using the basics of StatEye.

2. METHODOLOGY OVERVIEW
The code example consists of a top level test circuit which is made up from various circuit elements and blocks as follows. A Piecewise Linear (PWL) voltage source provides a pulse waveform through a series resistor into an ideal transmission line. On the other end of the transmission line, there is a series inductor and a block representing a receiver, containing a capacitive load. A schematic drawing of the circuit is shown in Figure 1.

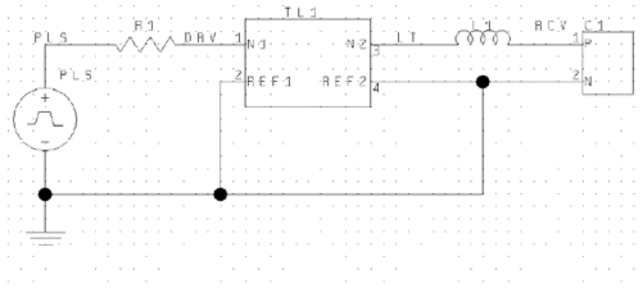

Figure 1 Top Level Test Circuit

The block representing the receiver contains three VHDL-AMS architectures. One is a simple capacitance model, called "ideal". The remaining two, called "StatEye_on" and "StatEye_off" are variants of the Statistical Eye algorithm. The two slightly different versions of the algorithm were implemented in order to be able to measure the simulation time of the Statistical Eye algorithm alone more accurately.

The values of the circuit elements are parameterized. The waveforms shown in this paper were generated using the following parameters:

pulse source:	1 volt, 200 ps width, 1 ps edge
series resistor:	100 Ω
transmission line:	100 Ω, 0.5 ns delay
inductor:	10 nH
receiver capacitance:	1 pF

These values were chosen rather arbitrarily. The goal was to generate a waveform at the receiver's input that looks very much like a pulse response of a typical communication channel. (See the top portion of Figure 2 below).

Using this test circuit, a normal time domain simulation is started. The waveform obtained at the receiver's input corresponds to the pulse response of the channel. While the simulation is running, this pulse response is saved in a vector inside the receiver block at a fixed sampling rate. The waveforms shown below were generated with a 1 ps sampling rate (fixed time step) and the length of the pulse response was 3 ns.

At the end of the pulse response simulation (3 ns) the VHDL-AMS code in the receiver block begins to process the waveform stored in the vector, and the remaining part of the time domain simulation is used to display the results of the Statistical Eye algorithm, which is the eye contours corresponding to the various BER levels. Since the pulse width used for the waveforms of this paper was 200 ps, the eye waveforms have a total width of 200 ps and therefore the last portion of the simulation is a little over 200 ps long. The number of eye contours calculated and plotted depends on an input parameter passed into the Statistical Eye algorithm. With appropriate parameter values, the innermost eye contour will be identical to the worst eye opening that is obtained with the PDA algorithm. Figure 2 shows the waveforms of the pulse response (top half) and a collection of eye contours which are the result of the Statistical Eye algorithm (bottom half).

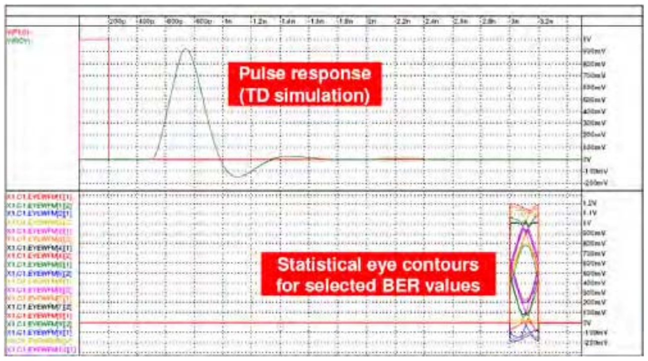

Figure 2 Simulation Results, Pulse Response and Eye Contours

For better readability, Figure 3 shows an enlarged version of the eye contour plot from the bottom half of Figure 2.

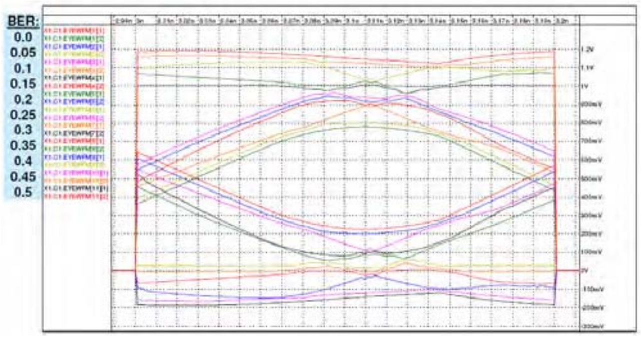

Figure 3 Eye Contours Enlarged

A special feature was added to the VHDL-AMS code to enable the viewing of the results in 3-dimensional plots with third party tools. When this portion of the code is enabled, the results of the Statistical Eye algorithm will be written into a data file in MATLAB® format. A 3D representation of the eye contour plot in Figure 3 is shown here in Figure 4.

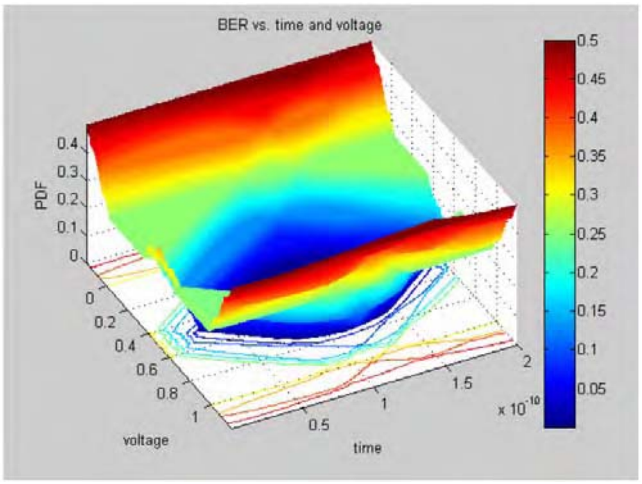

Figure 4 3D view of the Statistical Eye Results

2.1 FUNCTION DESCRIPTIONS

The following sections give a short description for each of the functions and processes in the VHDL-AMS code.

2.1.1 Process "GetCursorIndex"

Locates the peak of the pulse response waveform which is assumed to coincide with the center of the eye (in time).

2.1.2 Process "Ticker"

Saves the points of the pulse response waveform (during the first 3 ns of the TD simulation at a 1 ps time interval, a total of 3000 points in our example). After the pulse response part of the simulation is done (3 ns), it calls the MakeEyeContours function (200 times at 1 ps intervals, determined by the pulse width, which is 200 ps in this study). MakeEyeContours and all of its associated functions make up the statistical eye algorithm and are located in the file "StatEye_functions.vhd".

2.1.3 Function "MakeEyeContours"

Partitions the pulse response waveform into windows of size BitWidth and passes an array of one sample from each of these windows into the function called Convolve.

2.1.4 Function "Convolve"

Convolves all of the pre/post cursor samples to generate the probability distribution function (PDF) of the inter symbol interference (ISI) only and calls the MakeCPDF function.

2.1.5 Function "MakeCPDF"

Combines the logic 1 and 0 cursors with the ISI PDF, generates the cumulative probability distribution functions (CPDF) by integration, generates the eye contours for 2D plotting in the simulator, and calls the Make3Dfile function to save the CPDF data to a file.

2.1.6 Function "Make3Dfile"

Generates the output data in MATLAB® syntax for a cell array containing the data and writes it to a file.

3. BENCHMARKS

The simulation times of the various configurations for the above test case were recorded in Table 1.

Table 1 Top Level Run Results

	ON with file write	**On without file write**	**OFF**
Device evaluations	70.0 % (5s 48ms)	76.0 % (4s 826ms)	79.5 % (4s 983ms)
Factorizations	2.6 % (200ms)	2.8 % (179ms)	3.5 % (220ms)
Resolutions	0.1 % (9ms)	0.5 % (29ms)	0.3 % (19ms)
Convergence checks	0.4 % (29ms)	0.3 % (19ms)	0.5 % (29ms)
Logic kernel solve	---	---	0.2 % (9ms)
Logic generated processes	11.3 % (859ms)	4.7 % (300ms)	1.9 % (119ms)
Logic generated memory allocations	6.4 % (489ms)	6.6 % (420ms)	5.3 % (329ms)
Waveform display	1.4 % (109ms)	1.4 % (89ms)	1.1 % (69ms)
Other routines	7.7 % (590ms)	7.6 % (479ms)	7.7 % (483ms)
Total	100.0 % (7s 640ms)	100.0 % (6s 348ms)	100.0 % (6s 268ms)

The numbers in Table 1 reveal that writing to the file takes the longest time. The statistical eye algorithm increases the CPU time of the digital engine, but it is very fast (adds only 181 ms). It takes more time to simulate the pulse response with the analog circuit solver than calculating the statistical eye contours with the digital kernel.

4. FUTURE WORK

The work discussed in this paper implements only the basic statistical algorithms. No higher order effects, such as cross talk and jitter are included. To make this capability useful in real-life design work, additional algorithms to take such effects into account will have to be added. These algorithms are readily available, and can be easily implemented in the VHDL-AMS code presented in this paper.

5. REFERENCES

[1] A. Muranyi, "Peak Distortion Analysis Implemented in VHDL-AMS", *2006 IEEE International Behavioral Modeling and Simulation Conference*, http://www.bmas-conf.org/2006/4.7_presentation.pdf., Sep. 2006.

[2] B. Casper, "Peak Distortion ISI Analysis", Circuits Research Lab, Intel Corporation, http://download.intel.com/education/highered/signal/ELCT865/Class2_15_16_Peak_Distortion_Analysis.ppt., Aug. 2003.

[3] A. Ghiasi, S. Anderson, "Using StatEye for IEEE Backplane Evaluation", IEEE 802.3ap Task Force, http://www.ieee802.org/3/ap/public/signal_adhoc/ghiasi_01_0904.pdf, Sep 2004.