Statistical Eye Analysis Implemented in VHDL-AMS

IBIS Summit, DesignCon 2007

February 1, 2007

Arpad Muranyi
Signal Integrity Engineering
Intel Corporation
arpad.muranyi@intel.com

Background

- **A VHDL-AMS implementation of the Peak Distortion Analysis (PDA) algorithm was demonstrated at the October 2006 IBIS Summit in Tokyo, and at the 2006 IEEE International Behavioral Modeling and Simulation Conference (BMAS):**
  - http://www.vhdl.org/pub/ibis/summits/oct06b/muranyi.pdf
  - http://www.bmas-conf.org/2006/4.7_presentation.pdf
  - http://www.bmas-conf.org/2006/4.7_project.zip
- **PDA predicts the worst eye opening, but**
  - what is the probability of getting an eye that size (or any other size)?
  - i.e. what is the bit error rate (BER) of a certain channel?
- **Numerous statistical methods have been developed by companies**
  - mostly in C, Matlab, or other proprietary languages
  - a well known and publicly available package is "StatEye":
    http://www.stateye.org/index.php
  - these solutions do not work in commonly used signal integrity (SI) tools
  - models from multiple vendors may not work together in the same channel design
- **The challenge is to bring these algorithms to the familiar signal integrity (SI) tools and to make high speed SERDES driver and receiver modeling possible in a standardized format (IBIS)**

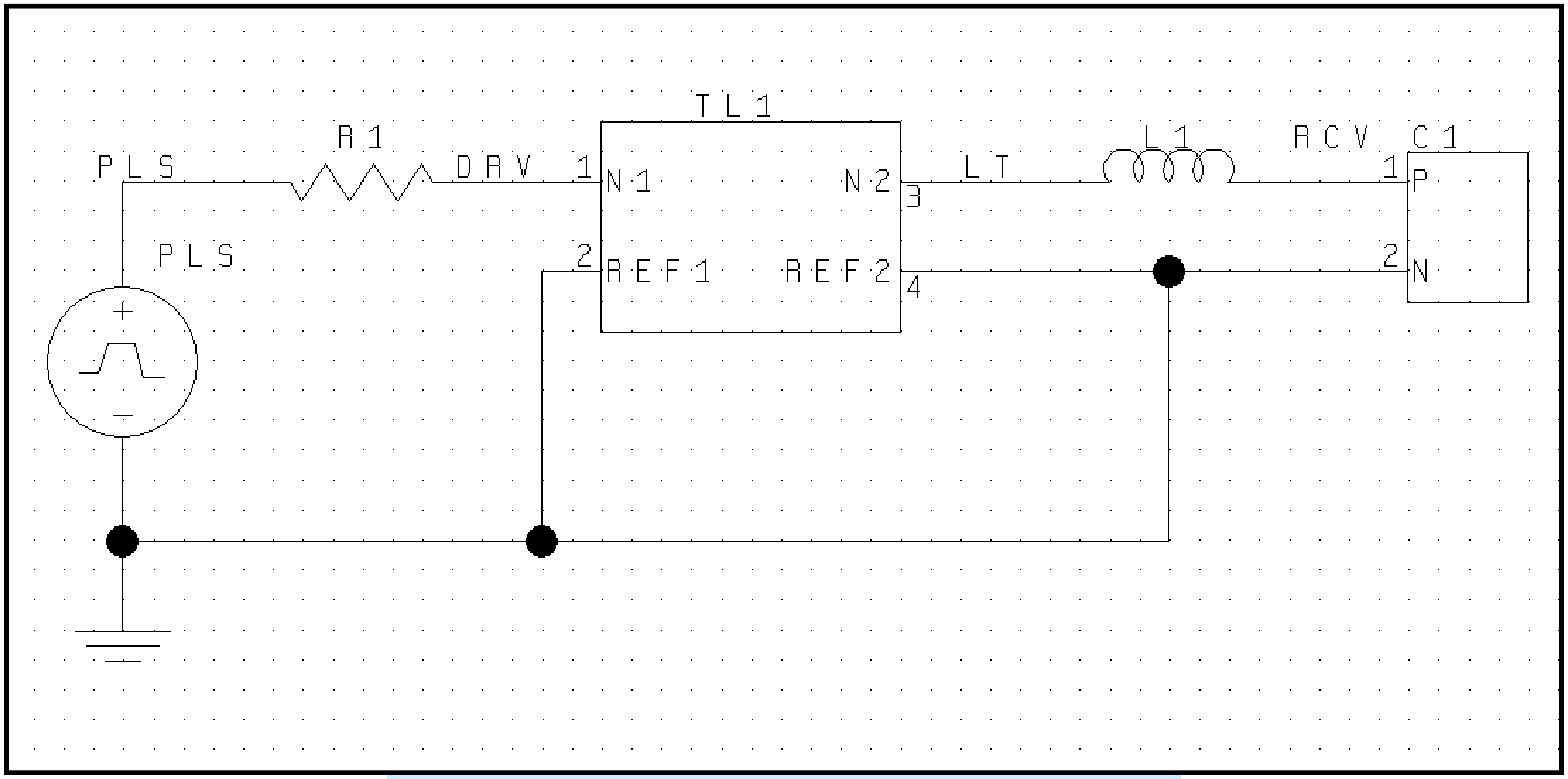
Purpose

- **Show that statistical eye and BER algorithms can be implemented very well using the VHDL-AMS (or Verilog*-AMS) languages**
- **Some of the advantages of using *-AMS:**
  - VHDL-AMS and Verilog*-AMS are standardized languages
  - the IBIS standard (v4.1 and higher) has links to both languages
  - there are several simulators which support them already
  - Tx and Rx models from different vendors would work together
- **However, *-AMS modeling is not gaining popularity very fast in the SI community ☹**
- **The purpose of this presentation is to encourage *-AMS modeling and to help model makers and users to get started**
- **Although this presentation demonstrates a working model, its goal is not to provide a complete solution**
  - the purpose is to show the concepts only
  - (i.e. it is full of bugs that are waiting to be fixed) ☺
- **Currently implemented in VHDL-AMS only**

The simulation setup

- **The top level test circuit is the same as the one used in the previous PDA algorithm presentation**
  - both implementations are based on a pulse response
  - the function of the top level circuit is to generate that pulse response, consisting of:
    - a Thevenin “driver” using a pulse voltage source and a resistor
    - an ideal T-line
    - a series inductor
    - a capacitor acting as a “receiver”
  - schematics and details are shown on the next page
- **For this study I didn’t care whether the channel circuit was realistic or not, as long as it gave a reasonable pulse response**

Schematics of the simulated circuit



Pulse:	**1 V, 200 ps wide, 1 ps edge**
R1:	**100 Ω**
T-line:	**100 Ω, 0.5 ns, lossless**
L1:	**10 nH**
C1:	**1 pF**

Overview of the operation of the model

- **A normal time domain simulation is started**
  - the first 3 ns of the simulation generates the pulse response
  - the 3000 points of the pulse response waveform are stored in a "real_vector"
  - a 1 ps fixed time step is used for simplicity
  - a 200 ps pulse width is used which yields a 200 ps wide eye (200 points)

- **The simulation is continued for another 200 ps to perform the "analysis" and plot the eye**
  - the functions of the algorithm are executed at each of these iterations
  - 200 vertical eye slices (probability density function vs. voltage) are calculated
  - the raw data is printed into a file (in Matlab format) for 3D plotting
  - a family of eye contours are generated for 2D plotting in the simulator
  - (the innermost eye contour is identical to the worst eye obtained with PDA)

  - the statistical algorithm uses only the digital engine (very fast)
  - the analog engine is only used to generate the pulse response and to plot the results from the digital engine

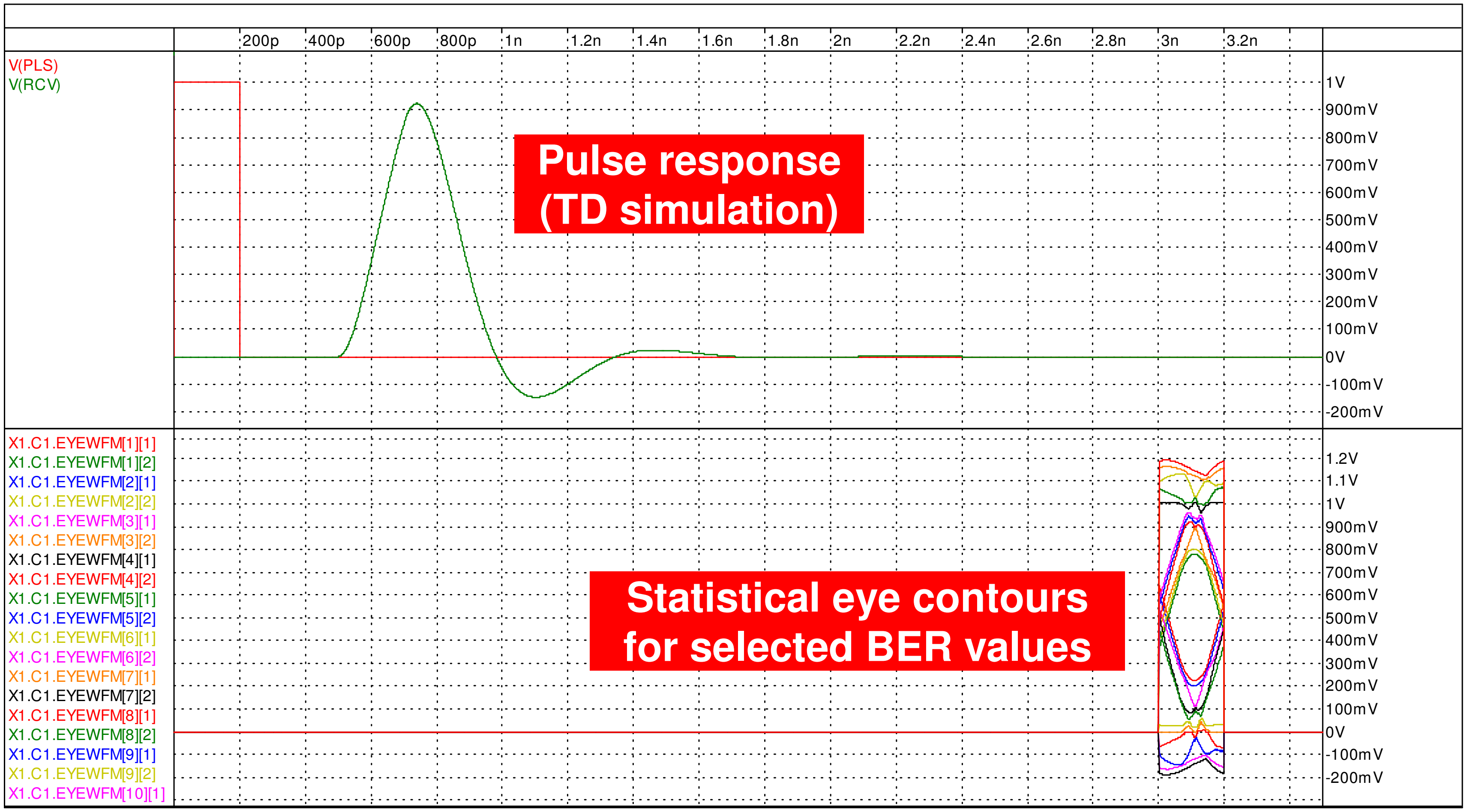
Flow - top level circuits

- **Test bench**
  - the circuit shown on the schematics
  - these are located in the file called **“IBIS_StatEye_library.vhd”**
- **The receiver model, C_StatEye contains three VHDL-AMS architectures**
  - **“ideal”**, a simple capacitor model
  - **“StatEye_on”**, a capacitor model with processes to activate the algorithm
  - **“StatEye_off”**, a capacitor model with processes that pretend to activate the algorithm but don’t (for more reasonable benchmarking)
- **The two processes in C_StatEye**
  - **“GetCursorIndex”**, locates the peak of the pulse response waveform which is assumed to coincide with the center of the eye (in time)
  - **“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)
  - after 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”**

Flow - functions

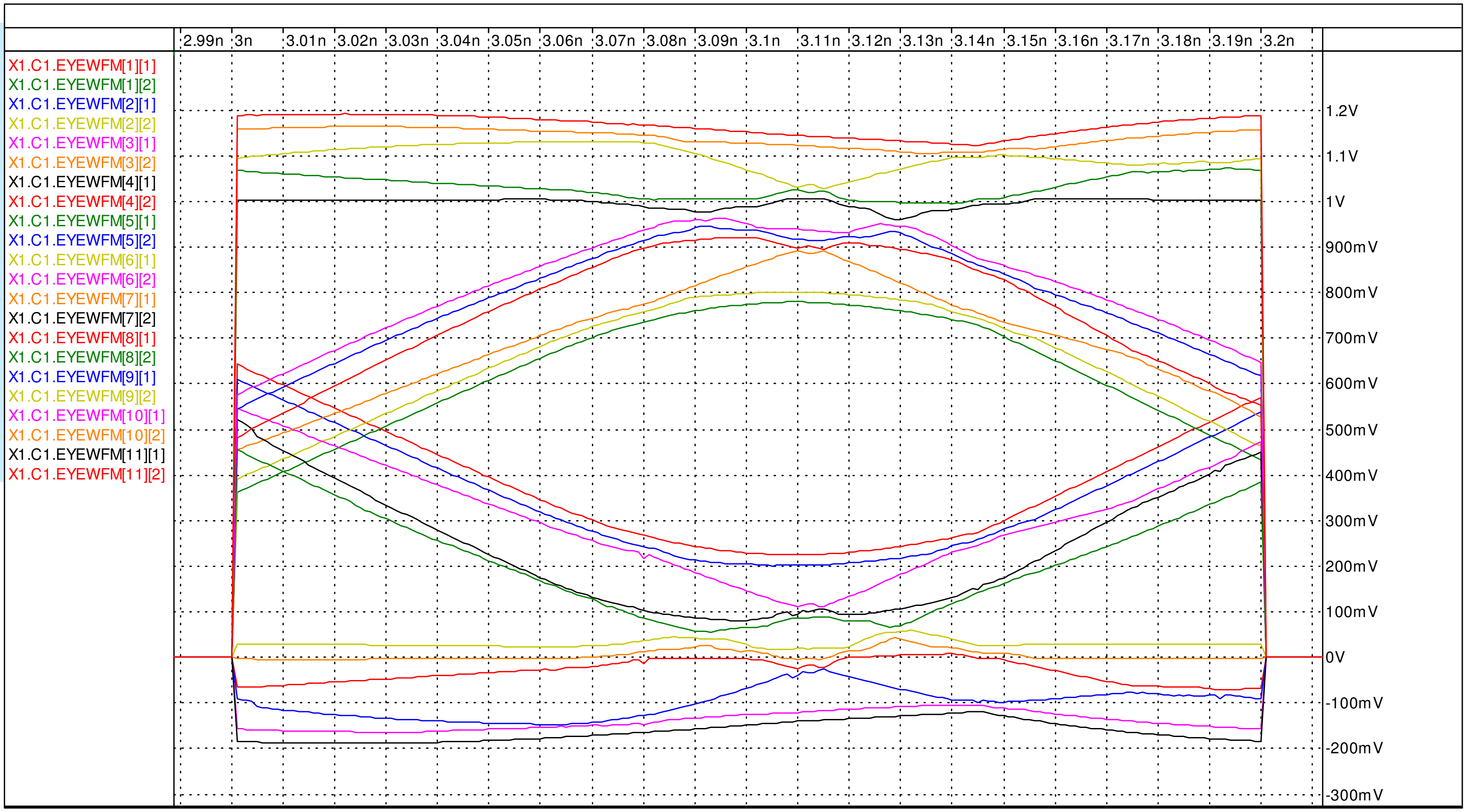
- **MakeEyeContours**
  - partitions the pulse response waveform into windows of size BitWidth and passes an array of one sample from each of these windows into the function called Convolve
- **Convolve**
  - convolves all of the pre/post cursor samples to generate the probability distribution function (PDF) of the inter symbol interference (ISI) only
  - calls the MakeCPDF and CPDFtoPlot functions
- **MakeCPDF**
  - combines the logic 1 and 0 cursors with the ISI PDF
  - generates the cumulative probability distribution function (CPDF) by integration
- **CPDFtoPlot**
  - calls Make3Dfile to print raw data to a file
  - extracts the contour values for the BERvalues requested and returns these all the way up to the top level to be plotted in the simulator's plotter
- **Make3Dfile**
  - rounds off the voltage axis values towards the center of the eye for making 3D plotting easier in Matlab
  - generates the Matlab syntax for a cell array containing the data and writes it to a file

Pulse response and statistical eye contours



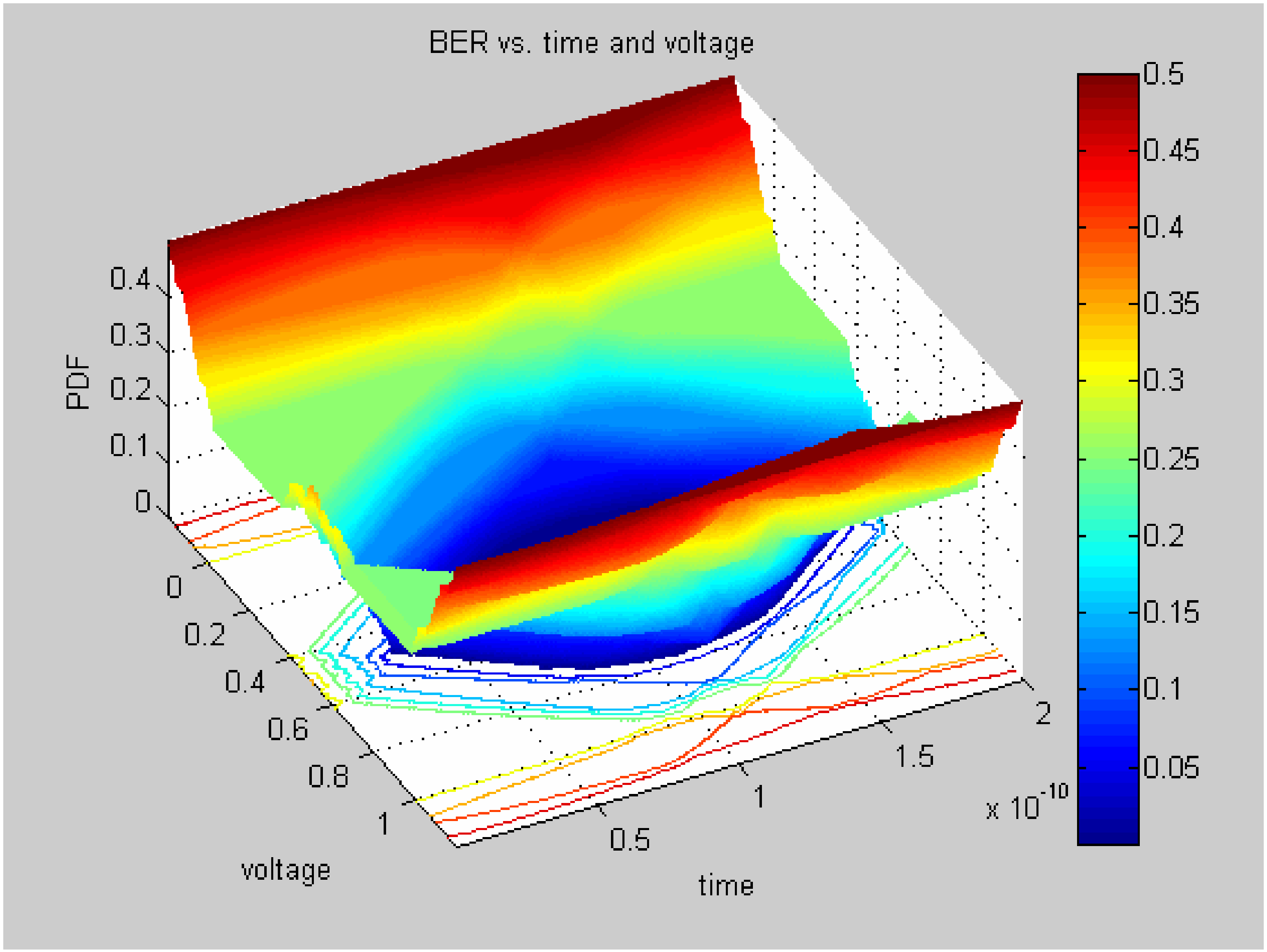
Statistical eye contours (zoomed in)

BER:
0.0
0.05
0.1
0.15
0.2
0.25
0.3
0.35
0.4
0.45
0.5



The Rx model has an array parameter (shown on the top left) through which the user can request which and how many eye contours should be plotted. Each contour corresponds to a certain BER.

3D plot of the data generated by the VHDL-AMS code



Simulation statistics

Algorithm: ON OFF

```
        -------------------------------------------------------------------- ----------------------------
        Transient: CPU time spent in:
             - device evaluations                :    1.2 % (551ms)          :  40.0 % (561ms)
analog       - factorizations                    :    0.1 % (50ms)           :   5.0 % (70ms)
             - convergence checks                :    0.0 % (20ms)           :   1.4 % (20ms)
             - logic kernel solve                :    0.0 % (10ms)
digital      - logic kernel propagate            :    0.0 % (9ms)
             - logic generated processes         :   95.8 % (44s 991ms)      :  13.6 % (190ms)
             - logic generated memory allocations :   1.3 % (630ms)          :  20.0 % (280ms)
             - waveform display                  :    0.2 % (89ms)           :   1.4 % (20ms)
             - other routines                    :    1.3 % (601ms)          :  18.6 % (260ms)

             - total                             : 100.0 % (46s 954ms)       : 100.0 % (1s 402ms)
        -------------------------------------------------------------------- ----------------------------
```

Note: These statistics were obtained without saving the data to a file

Summary and future work

- **This experiment implemented only the basic equations of statistical eye analysis**
  - source: “Channel Compliance Testing Utilizing Novel Statistical Eye Methodology”, DesignCon 2004, by Anthony Sanders, Infineon Technologies
    http://www.stateye.org/downloads/channel_compliance.pdf
    (see section “Theoretical Analysis of Receiver Pulse Response”)
  - cross talk is not included
    - need to add coupled T-lines and do more convolutions
  - jitter is not included
    - more sophisticated stimulus and/or processing algorithms may be needed

- **A 3000 point pulse response and a 200 point pulse width**
  - 14 sampling windows convolved and post processed 200 times in less than a minute (on a 1.8 GHz laptop)
  - a better sorting algorithm (used in the convolution function) could reduce CPU time dramatically
  - additional code optimizations could be applied in the implementation

- **Future work may include**
  - implement the same in Verilog-A(MS)
  - implement similar (equivalent) algorithms in the frequency domain