Certify- A Characterization and Validation Tool for Behavioral Models

Weifeng Li, University of Arkansas, wxl002@uark.edu
Omair Abbasi, University of Arkansas, omair.abbasi@gmail.com
Naveed S. Hingora, University of Arkansas, nhingor@uark.edu
Yongfeng Feng, University of Arkansas, yfeng@uark.edu

H. A. Mantooth
University of Arkansas
mantooth@uark.edu

ABSTRACT

Device modeling plays an important role in VLSI circuit design because computer-aided circuit analysis results are only as accurate as the models used. This indicates a need for robust tools that can facilitate the testing, validation and characterization procedure of semiconductor device models. Certify, the graphical tool for model characterization and validation, is a step in this direction [1]. The software architecture and different modules of Certify have been described in this paper.

1. INTRODUCTION

As semiconductor devices scale into the nanometer range, the fabrication process of these device structures becomes more time consuming and expensive. As a result, modeling is having an increasingly important role, as one can perform simulations much faster with much less cost, and different device geometries can be investigated before their actual fabrication.

Writing a model which can accurately depict the characteristics of a real device is a challenging task. An efficient model should not only include the device characteristics in normal conditions of operation but should also be able to accurately depict the device behavior under worst case conditions such as high temperature ranges, extreme conditions etc. There are various programming languages in which such models can be written. They can be written in popular programming languages such as C and C++. Models for most popular circuit simulator SPICE (and its variants), are written in C. A second choice is writing these models in Hardware Description Languages (HDLs). The syntax of HDLs is easier to understand as they deal specifically with the problems of describing hardware. Various HDLs are available today for different purposes. VHDL and Verilog were invented to describe digital systems. Then came MAST [2], VHDL-AMS [3] and Verilog-AMS [4] which are used to describe analog and mixed–signal systems. Though HDLs simplified the model creation process significantly, the modeler still had no escape from customs and practices of coding and debugging [1].

Writing a model solves just half of the problem. The other half is testing, characterizing and validating the model. The process of extracting the optimum values of model parameters is called model characterization. Once the model is characterized it has to be validated by simulating it under various test conditions and then comparing the results with the results of actual device. Modelers greatly feel the need for a tool which can facilitate the whole characterization and validation process. This paper describes a software tool called Certify that is specifically designed to fulfill this need. Certify reads semiconductor device models written in the Common Model eXchange (CMX) language in order to test and characterize them [5][6]. The modeling tool Paragon was developed as a graphical editor for this format [7]. In addition, different hardware description languages such as Verilog-A, Verilog-AMS, VHDL-AMS, or MAST can be converted to CMX using the commercial version of Paragon, ModLyng tool of Lynguent, Inc [8]. Using Certify, a user can work in a virtual test bench environment. Virtual test benches can be added and deleted in a highly user friendly Graphical User Interface (GUI). Experiments for each test bench can be defined and whole recipes can be saved. Certify has an in-built optimizer that can be used to optimize the model parameters and thus, characterize the model.

2. CERTIFY

Certify UI consists of a test-bench editor, an experiment editor, analyses dialogs, and an optimizer dialog. It is fully integrated with Saber Simulator [9] and partially integrated with Virtual Test Bed (VTB) simulator [10]. Certify makes calls to ModLyng to extract the model information in CMX format. The elaborator module of Certify calls the appropriate simulator in order to test and characterize the desired model. Figure 1 shows a brief overview of how the different components of Certify interact with ModLyng and the simulators. The following sections describe each component in further detail.

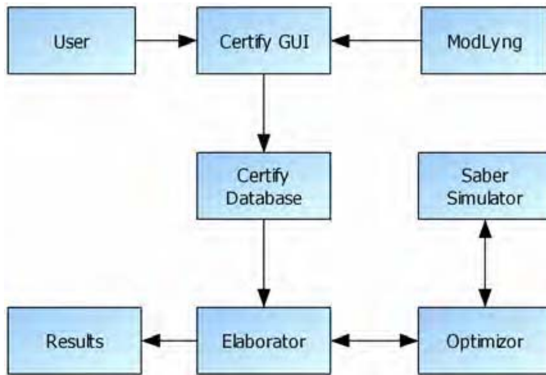

Figure. 1. Block diagram showing the interaction between Certify, ModLyng and Saber Simulator

2.1. Certify GUI

Figure 2 shows the test bench editor of Certify, which is also the main window. Users can create test benches, run the whole recipe, and set constraints for model parameter optimization. Users can also save and load the whole recipe. As the procedures for testing different devices are mostly the same, the users can reuse the saved recipes by simply replacing the model files and netlists.

The test-bench recipe is composed of the device model under test. Every individual test-bench consists of a netlist containing the device, and experiments to test the device. One can add multiple benches on the test bench editor.

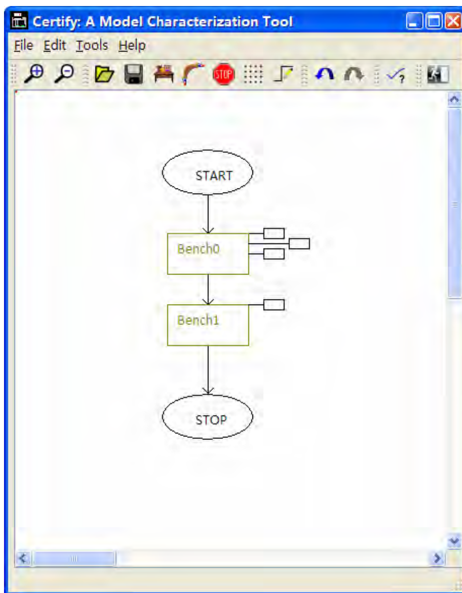

Figure. 2. Certify main window or test bench editor

The small buds on the right of test benches are experiments. Users can add multiple experiments to one test bench by right clicking the test bench block. By double clicking the experiment buds, the experiment editor can be invoked. Figure 3 shows the screenshot of the experiment editor.

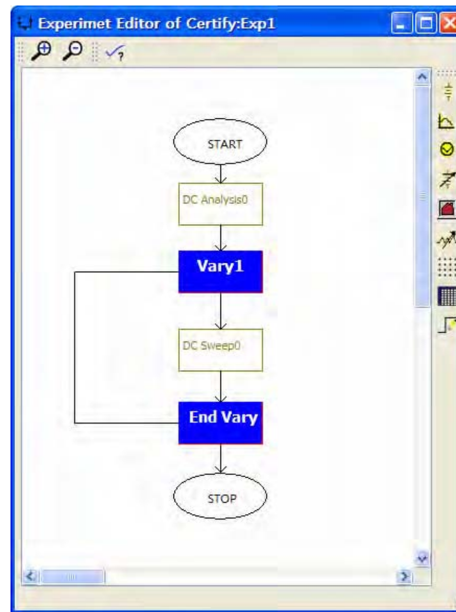

Figure. 3. Experiment Editor of Certify

On the experiment editor users can define experiments to simulate the circuit and compare the measured data to the simulated data. The various analyses implemented in Certify include DC Analysis, AC Analysis, Transient Analysis, and DC Transfer Analysis. Users can also perform vary function to these analyses, which can sweep part parameter values and execute the analyses at each swept value within a user-specified range [9]. After the simulation, the user can also choose to get a result report. Figure 4 shows the DC Transfer analysis dialog.

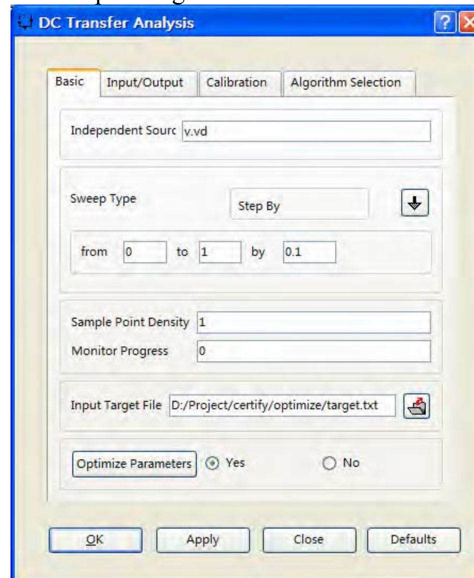

Figure. 4. DC Transfer Analysis Dialog

Once the model file is imported, users can call the parameter spreadsheet to set values of desired parameter for the optimization as shown in figure 5.

Figure. 5. Model Parameter Spreadsheet

After setting up the parameter values, the optimizer can be invoked by clicking the "Run" button of the parameter spreadsheet. The optimizer can alter parameter values for simulation and optimization. It can also draw graphs of the target file and simulated result for comparison.

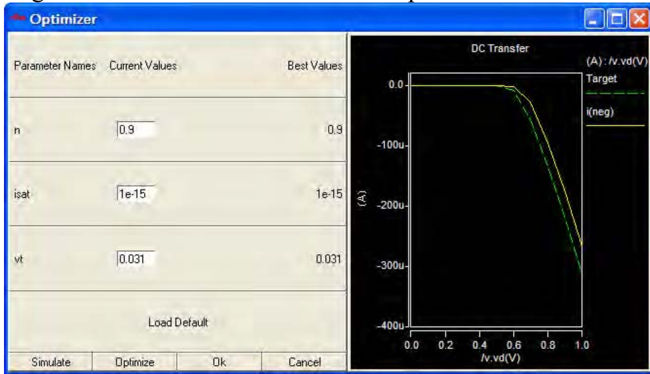

Figure. 6. GUI of the optimizer

2.2. Import models and extract model parameters

Certify is using ModLyng to get parameter names and values from model files. Figure 7 shows how certify interacts with ModLyng.

Figure. 7. the process of Certify to get model parameters

ModLyng is able to read AMS models written in various HDLs such as Verilog-A, Verilog-AMS, VHDL-AMS, or MAST. In the meantime, ModLyng itself is a powerful tool to create AMS models and can be saved in Common Model eXchange (CMX) format and can be converted to different HDLs.

The information Certify will need are the model parameter names and their initial values. When users click the parameter spreadsheet button, Certify will call ModLyng to parse and extract information from the model file. After ModLyng returns the parameter names and values, the parameter spreadsheet will come up as shown in figure 5.

Models written in languages other than MAST need to be converted to MAST by using ModLyng to perform Saber simulations. For the VTB simulator, the model needs to be already present in the VTB library before executing the simulation task.

2.3. Process of model characterization/optimization

Figure 8 shows the methodology of the model characterization. Experiments are setup with a test circuit containing the device model under test with initial parameter values. Then simulation is performed, and the simulation results are compared with target data. The comparison is made by a cost function. If the cost function value is minimum, the corresponding model parameters are optimized. If not, model parameter values are altered and the simulation is performed again. The comparison between simulation results and target data are made again. This process continues until the optimum match is acquired. This technique is also referred to as the simulator-in-loop method [11].

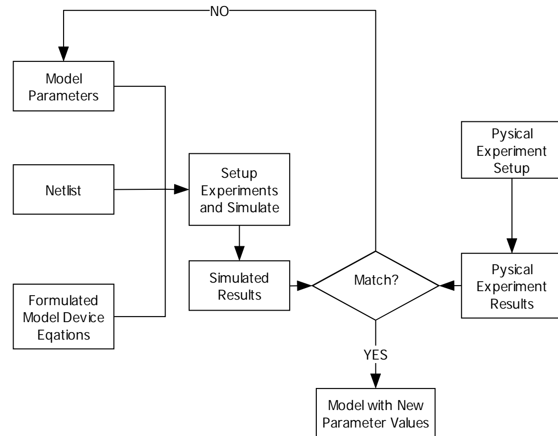

Figure. 8. Model characterization methodology [1]

Figure 9 shows the detailed procedure of optimization. The strategy is to change one parameter value at a time. Other parameters are temporarily fixed while the chosen one is being optimized. If the user opts for the automated procedure, Certify will optimize the parameters one by one. The user can also choose which parameter to be optimized.

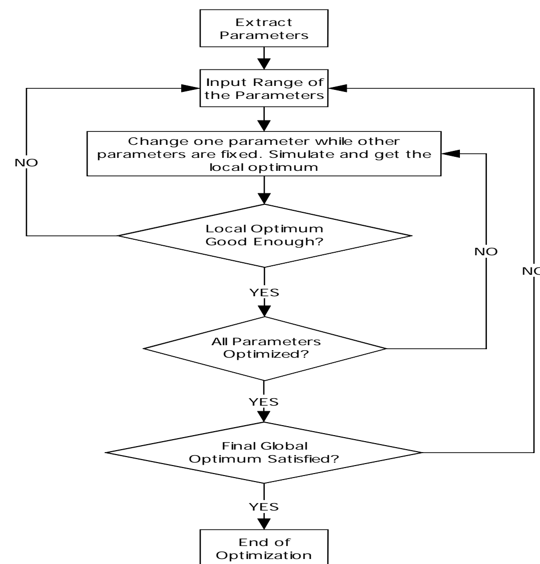

Figure. 9. Optimization Flowchart

2.4. Simulator Integration

Certify has been fully integrated with Saber simulator and partially integrated with the VTB simulator. At test-bench run-time the user is given the option to choose either of the two simulators to perform the simulation as shown in the following figure.

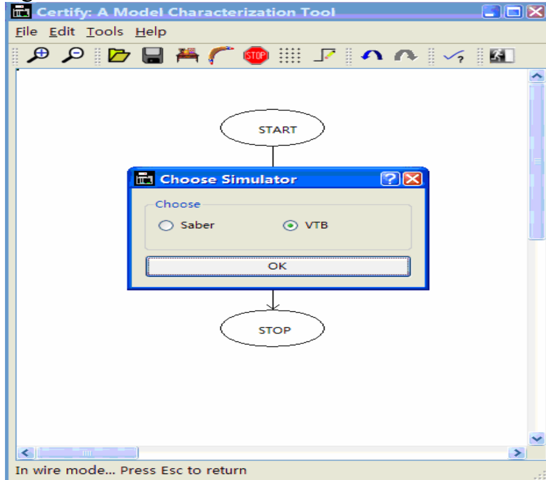

Figure. 10. Simulator Option Dialog

2.4.1. Integration with Saber Simulator

Certify has an in-built module called elaborator that uses the information from the Certify Database to get the desired results. For the Saber simulator, the elaborator converts all of the information in the analyses blocks of the database into Saber specific commands. A ".scs" extension analysis file is created for every analysis inside an experiment and this file is passed on to the Saber simulator for simulation. Thus, different analyses are performed for different calls of Saber simulator. A sample (.scs) script for a dc sweep analysis with vary over the primary voltage source is shown in the following figure.

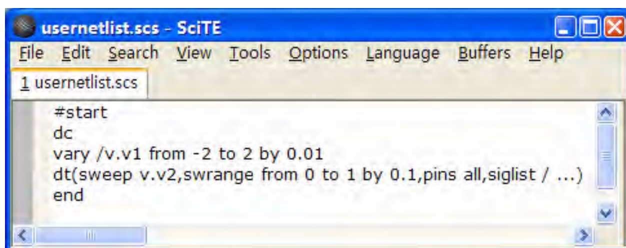

Figure. 11. Saber Script (.scs) File

The GUI of optimizer (figure 6) is created by an ".aim" script file generated by the module called AIMoptimizer. This module gathers information about the model parameters from the parameter spreadsheet, and gets the simulation commands from the module elaborator to generate the script. Using this optimizer, Certify can call the Saber simulation to alter parameter values and perform simulation. Figure 12 shows an example of the generated .aim script file.

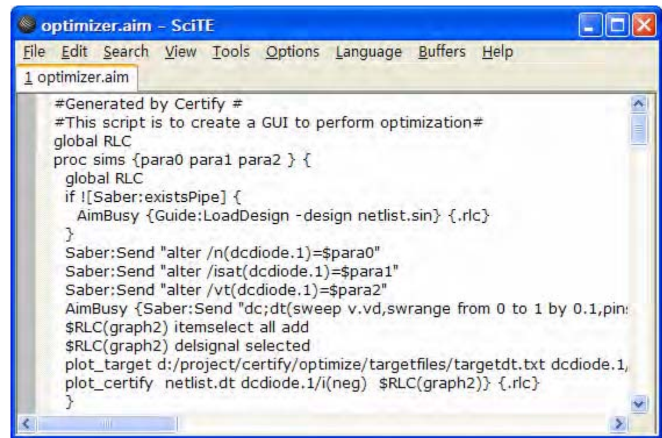

Figure. 12. AIM Script (.aim) File

2.4.2. Integration with VTB

When the option VTB is exercised at test-bench run time, a second elaborator in Certify is invoked that carries on the task of extracting the analysis information from the database file and calling a C# executable that in turn controls the VTB simulator. Work is underway to be able to tweak more simulation parameters in VTB and also, go beyond the transient analysis capability of VTB to implement DC sweep and AC analysis.

3. SOFTWARE DESIGN

Certify was written in Python programming language. Python is an *interpreted, interactive, object-oriented* programming language [12]. The GUI of Certify was developed using PyQt toolkit. PyQt is a set of Python bindings for the Qt toolkit [13]. The bindings are implemented as a set of Python modules. Qt is primarily a GUI toolkit.

3.1 Architecture Design of Certify

The software architecture of Certify is described through a data flow diagram. Figure 13 shows the architecture of Certify. Test Bench Editor is the main window of Certify, which gathers the inputs from the users and it calls the tools and modules such as schematic and drawing manager, database manager, elaborator and experiment editor to capture the recipe and run the experiment. Experiment editor is called by test bench editor and it defines the analysis to be done on the circuit. Schematic and drawing manager manage the objects and the drawing activities on the canvas of test bench editor and experiment editor. Database Manager writes all information of Certify in a database file during save operation and loads all information during load operation. Elaborator reads the certify database and implements functions to interact with the Saber/VTB simulator and the optimizer to run the whole test-bench recipe.

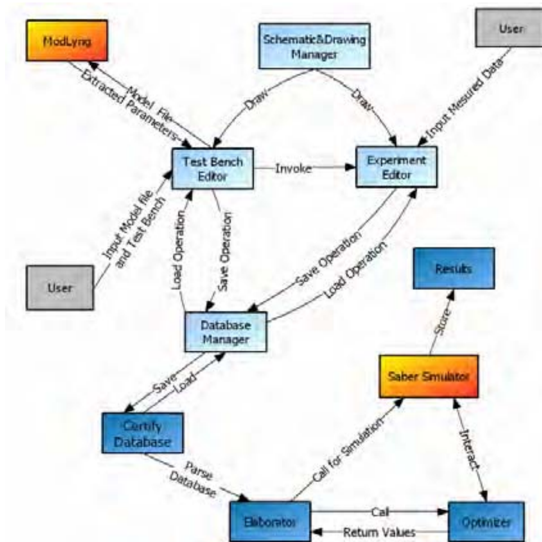

Figure. 13. Architecture of Certify [1]

3.2 GUI Design of Certify

Figure 14 shows the components of the Certify GUI and how they are bond together. Schematic objects module is a sub module of the schematic and drawing manager. It defines all of the objects drawn in both the test bench editor and the experiment editor. The objects include the ellipses representing the start and end point of the flow, the rectangles representing the test benches and the analyses blocks, the small buds that represent experiments and invoke the experiment editor, and the connector to connect different objects to form the recipe flow. The parameter spreadsheet sub module reads the information of the device model from ModLyng and forms a spreadsheet to control the model parameters for characterization. The analysis dialog module connected to the experiment editor defines the data structures of the analysis dialogs. The Model Characterization Tool (MCT) properties module defines the common frame for all the analysis dialogs such as the dialog size, the function of the "Apply", "OK" and "Cancel" buttons. All this information can be saved by database manager in XML format.

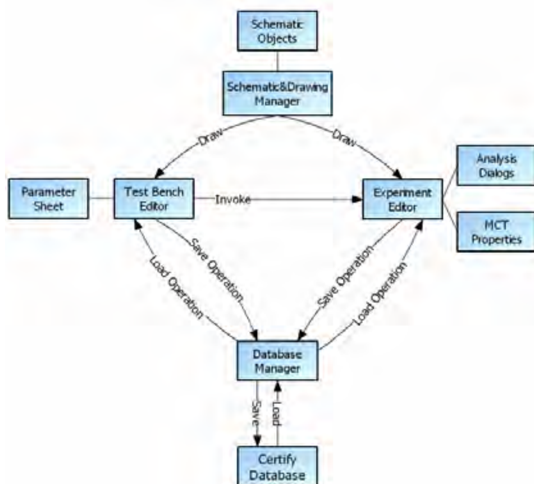

Figure. 14. Control Flow of Certify GUI [1]

3.3 Data Structure Design of Certify

There is a set of data structures associated with each object in a canvas. These data structures are mostly Python strings, lists and dictionaries. Listing 1 shows a part of the actual code for the data structure of properties of DC analysis. In this example, all data structures are strings and are initialized to some value when they are created.

Listing 1 a part of the data structure of DC analysis properties

```
#dcAnalysisDataStruc()
#
#this data struc will be initialized if it is a dc analysis block
def dcAnalysisDataStruc(self):
    self.samplePointDensity="0"
    self.monitorProgress=""
    self.holdNodes=""
    self.releaseHoldNodes="yes"
    self.displayAfterAnalysis="no"
    self.optimizeParameters="no"
    self.signalList=""
    self.waveformsAtPins="Across Variables Only"
```

Certify uses the build in data type "dictionary" in python to store the data. Dictionary is called "associative memories" or "associative arrays" in some other languages and it is formed of unsorted *key: value* pairs [12]. Listing 2 shows some part of the code of the method called "createProperDict" which was implemented as part of data processing for Certify. This method creates dictionaries whose keys contain the name of an attribute (of some node) to be used in an XML file and the value of the dictionary as the value of that attribute. In other words, it is the mapping between the internal data structures in the code and their names in the output XML file.

Listing 2 Part of the code using dictionaries

```
#createProperDict()
#
#creates proper dictionaries which hold the data name and their value
#if you need to add anything to the database just add another item to the dic.
#with key as the name you want in database and value as the corresponding data struc
def createProperDict(self):
    #self.dataDic={}
    self.basic={}
    self.io={}
    self.calibration={}
    self.algorithmSelection={}
    self.integrationControl={}
    if self.analysisType=="DC":
        #data struc for basic
        self.basic["SamplePointDensity"]=self.samplePointDensity
        self.basic["MonitorProgress"]=self.monitorProgress
        self.basic["HoldNodes"]=self.holdNodes
        self.basic["ReleaseHoldNodes"]=self.releaseHoldNodes
```

All the data contained in the data structures is converted into a Document Object Model (DOM) [14] tree by the Database Manager when the user saves the information. A DOM implementation presents an XML document as a tree structure or allows client code to build such a structure from scratch. It then gives access to the structure through a set of objects which provide well-known interfaces. This DOM tree is finally saved in the form of a XML file. Python has a built-in XML parser in a module called "xml.dom.minidom." This module was used in Database Manager to save and load all the information to and from the database. Figure 14 shows a part of the saved XML file.


```
- <Model MastModel="None" MultipleFiles="Yes">
  - <TestBench Name="Bench0"
    NetlistFileName="c:/paragon/project/certify/fwdcertify/Mosfet.sin">
    - <Experiment Name="Exp1">
      - <DC_Analysis Name="DC Analysis0">
          <Basic DisplayAfterAnalysis="no" HoldNodes=""
          MonitorProgress="" OptimizeParameters="no"
          ReleaseHoldNodes="yes" SamplePointDensity="0"/>
          <IO EndPointDataFile="" EndPointFile="dc" EndPointPlotFile=""
          InitialPointFile="zero" SignalList="" UseInitialConditions="yes"
          WaveformsAtPins="Across Variables Only"/>
          <Calibration Calibrate="no" FunctionValueTolerance="1n"
          MaxADIterations="3" MaxEventIterations="5000"
          TimeResolution="1p" VariableDeltaTolerance="1u"/>
          <AlgorithmSelection AlgorithmStepping="yes"
          FirstOrOnlyAlgorithm="No Ramping"
          MaximumNewtonIterations="100" NewtonStepDensity="1"
          NewtonStepLimiting="yes"/>
        </DC_Analysis>
```


Figure. 14. Part of the saved data in XML format

4. CONCLUSION

The tool described in this paper enables the modelers to quickly and effectively validate and characterize a semiconductor device model. Standard validation and characterization recipes can be created and stored. These recipes can be re-used again for other models. This saves lot of time. Also, all simulation parameters can be saved and the user doesn't need to fill the simulation information every time a similar type of simulation needs to be performed.

Certify is using ModLyng to read the model parameters and values from various HDLs in order to test and optimize a model using Saber or VTB simulator. By using ModLyng, models written in other languages can also be translated into MAST, which make it possible for Saber simulator to simulate the circuits.

REFERENCES

[1] O. Abbasi, *Certify: A Tool for Model Characterization and Validation*, Masters Thesis, University of Arkansas, Fayetteville, Arkansas, May 2006.

[2] *MAST – Analog, Mixed – Technology and Mixed – Signal HDL for Sa*ber. *www.synopsys.com/products/mixedsignal/saber/mast_ds.html*

[3] P. J. Ashenden, G. D. Peterson, D. A. Teegarden, *The System Designer's Guide to VHDL-AMS*, Morgan Kaufmann Publishers, San Francisco, CA, 2002.

[4] *Accellera Verilog Analog Mixed-Signal Group. www.eda.org/verilog-ams/*

[5] Common Model eXchange (CMX), *cmx.sourceforge.com*

[6] A. M. Francis, V. Chaudhary, H. A. Mantooth, "Compact semiconductor device modeling using higher level methods" *Proceedings of IEEE 2004 International Symposium on Circuits and Systems*, vol.5, 23-26 May 2004, pp 109-112.

[7] V. Chaudhary, A. M. Francis, X. Huang, H. A. Mantooth, "Paragon-A Mixed-Signal Behavioral Modeling Environment," *Proceedings of IEEE 2002 Communications, Circuits and Systems and West Sino Expositions*, vol 2, 29 June-1 July 2002, pp 1315- 1321

[8] ModLyng™ Integrated Development Environment, *www.lynguent.com/products/modlyng.html*

[9] Saber® Simulator Guide Reference Manual, *www.synopsys.com/products/mixedsignal/saber/saber.html*

[10] Virtual Test Bed Professional (VTB Pro), *www.vtbpro.com/index.htm*

[11] H. Gunupudi, *Survey of Optimization Algorithms for Model Parameter Extraction*, Masters Thesis, University of Arkansas, Fayetteville, Arkansas, Dec. 2005.

[12] *Fred L. Drake,* Python Tutorial, *http://docs.python.org/tut/tut.html*

[13] *QT overview*, Trolltech. *http://www.trolltech.com/products/qt/*

[14] W3C Document Object Model (DOM). *http://www.w3.org/DOM/*