Schematic Debugging Framework for the Multi-Simulator based Verification of the Mixed Signal Design

Freescale Semiconductor
Sylwester Warecki

Session 6.15

Presented at

cadence designer network

cdn LIVE™

Silicon Valley 2007

Schematic Debugging Framework for the Multi-Simulator based Verification of the Mixed Signal Design

Session: 6.15

Sylwester Warecki
Sylwester.Warecki@freescale.com
Freescale Semiconductor
Tempe Arizona, USA

CDNLIVE Conference
San Jose, California
10-12 September 2007

Abstract

Quality assurance in modern mixed-signal designs provides opportunities not only for above-average products but mostly for substantial savings due to increased yields and improved reliability of the products.

In the case of fully custom designs such as ASIC, the pre-layout validation plays especially important role due to costly redesign and limited automation. Such validation is performed on many levels starting with mathematical or behavioral models and ending on such effects as electro-migration (EM), electro static discharge (ESD), or hot carrier injection (HCI).

As elimination of device stress became one of the highest priorities in design verification flow, special methods such as safe operating area (SOA) checks were introduced in many simulators on the market.

However, due to the lack of a standard, which would be implemented across different EDA solutions, designers are left with incompatible implementations and often unsupported options.

The presented schematic debugging framework tries to address these problems by providing technology device-driven solution with a support for multiple simulators and analysis types. The tool has been integrated into Cadence Analog Design Environment (ADE) and binds design, simulation, SOA checks and waveform analysis into a single environment. It combines minimal setup overhead with flexibility of sophisticated checks.

During the session the implementation of tool infrastructure, encountered problems and successful approaches will be discussed. In addition, advantages of the tool recognized by designers as well as proposed future extensions will be covered.

Introduction

Validation of analog and mixed signal circuits can be performed in many phases of the design. Depending on the aspect of operation being evaluated such as SOA, ESD or HCI, designers are presented with various levels of automation.

Manual analysis of operating points of each of the devices requires significant amount of time and is particularly error prone. In effect, most of the devices are NOT checked for their nominal operating conditions thus inviting common bugs, circuit malfunction, physical damage or even destruction. Debugging such problems on already manufactured circuit is costly, time consuming and can better be described as a guesswork.

Yield related analysis and verification of proper operation of the circuit – involving numerous simulation for several process corners – is often overwhelming and any additional checks (for example for SOA conditions) are often not performed. This quite often is due to an extensive setup and lack of standards between simulators especially when the simulators come from different vendors. Lack of

portable and matching checks as well as a limited syntax of specific simulator is also a reason for simplified or even non existent validation process.

Removal of the uncertainty before the integrated circuit is manufactured, and decreasing the project cost usually caused by re-spins and missed schedules was a primary driver for development of circuit quality validation tool - Schematic Debugging Framework (SDF). The first version of SDF was presented in June 2003.

The SDF combines device characterization knowledge with the unified multi simulator environment. It has been incorporated into Freescale internal Cadence tools environment and together with technology specific files provides straightforward and yet sophisticated framework for circuit validation and debugging.

The analyses performed by the tool are based on simulation results of a given circuit with specific rules provided for each of the devices in a particular technology. Higher flexibility of checks is accomplished by a specialized rule editor, which allows for user-defined expressions for an arbitrary device in any library. The editor will be presented in point 3.

SDF is used as a final step in the schematic analysis flow for its pre-layout validation. SDF can significantly reduce cost of the IC development by decreasing the number of re-spins and very costly silicon debugging. In turn it allows for a reduction of ' time-to-market of the manufactured parts. As a final benefit, designs verified with SDF demonstrate an increased durability and an extended time-to-failure (TTF) due to the reduction of stress on the device. This particular feature is especially important for automotive products, lifetime pf which has to reach minimum of 10 years.

Outline

The following paragraphs will introduce the SDF utility. They will cover basic tool operation together with technology and user requirements. Integration into Analog Design Environment (ADE) for analog and mixed signal analysis will be presented. In addition several benefits of incorporating the tool into the standard design flow together with simple steps required for tool deployment, configuration and customization including user private rules, filters and setup files will be discussed.

Project Development Flow

Figure 1 presents a typical project development flow. Starting with a concept, a design goes through its implementation in form of high level description (blocks) later turned into either schematics or some sort of hardware description language (HDL). The RTL blocks are often delivered separately.

The design is translated into layout either through compilation or (for analog and mixed signal designs) with manual effort.

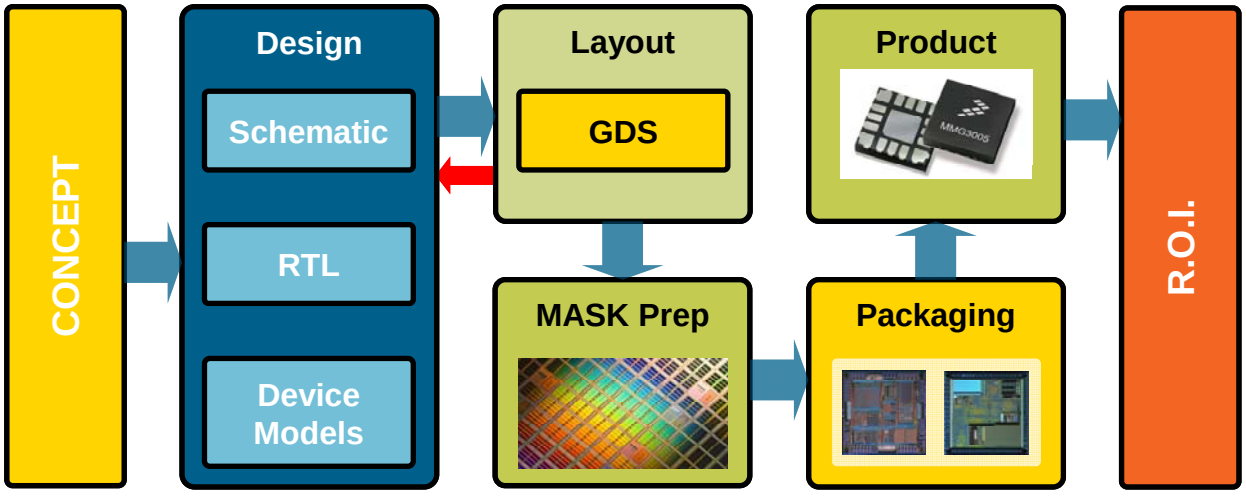

Figure 1: Example of Product Development Flow. Red arrows represent a request to modify design.

GDS files are produced for mask preparation step. Manufactured wafers are then scored and sent for packaging and testing. The final product in this case is an integrated chip.

For high return on investment (ROI) consecutive steps (especially transitions marked here with arrows) should be smooth (no delays) and should not be repeated. Each step has to be carefully executed with minimum amount of problems.

In reality, the full project cycle is a repetitive process with the design modifications being introduced on almost every stage.

One rule however prevails – if the modifications have to be done towards the end of the process, then they become very costly diminishing this way the ROI.

Anticipation of the problems in manufacturing, stands behind specialized tools for Layout validation : design rule checks (DRC) layout versus schematic (LVS) and layout parasitic extraction (LPE). Modern DRC engines also support design for manufacturing (DFM) rules which are directly effecting yield.

At the same time DRC style validation of device operations is limited and not supported by all simulators.

Validation Problem

From the designers' perspective, partial solutions to the design validation were already available. Several simulators had partial checks of operating conditions already available either through models (like warnings in Spectre) or through specific checks (like dchecks statements in UltraSim® or assertions in Spectre®). Simulators of other EDA vendors also include some form of checks.

Unification of these checks became a necessity as a single and uniform solution was necessary for a company providing models and technology support for more than a single simulator.

A tool that would furnish the validation needs of the designers through a well integrated interface together with a reliable setup and minimal effort from the designer's point of view became a necessity. Some of the reasons presented by designers are discussed below.

Known design cycle problems

Design reuse

A typical problem found in various designs is an existence of numerous errors caused by design reuse. Reusing schematics from similar technologies or even from previous version of the same technology's library without detailed verification of all imported devices often becomes a major reason of problems that go undetected until manufacturing stage.

For example leaving an isolation settings from high voltage technology in a default off-state (typically it is a CDF parameter translated to later on to PCELL dimensions) while migrating to the low voltage technology results in an undetected isolation violation. The problem will not show up in any simulation of the circuit containing the device as the isolation rules are most often not included in the model.

Such a mistake left unnoticed is very costly because it cannot be simply modified by a metal mask fix. The faulty device's dimensions require new layout as they are physically too small and therefore all masks have to be re-done.

Electro-Static Discharge

Another problem which has to be addressed in all top level designs is the Electro-Static Discharge. The ESD does not belong to the typical circuit operation conditions. In many cases no simulations checking top-level for ESD are performed as the designs rely on the ESD protection circuits built into the pads. However verification of the circuit behavior in the stressed conditions is necessary as it can detect vulnerabilities not seen during regular simulation.

In addition, the ESD protection circuits also need to be verified themselves in a systematic way to resolve possible bugs between ESD library revisions.

Operating Region

Some of the instances are scheduled to operate only in the specific state – for example saturation. Leaving saturation state may indicate problem in the design, which can cause damage or for example increased power consumption. The latter case became recently a problem in

portable devices which need to minimize power dissipation.

Hot Carrier Injection

Hot carrier injection (used in flash memories [1]) is a problem of NMOS device operating under specific conditions in which 'hot carriers' become trapped in the insulation layer (the thin oxide THX). These conditions involve high speed electrons colliding with the silicone lattice of the drain.

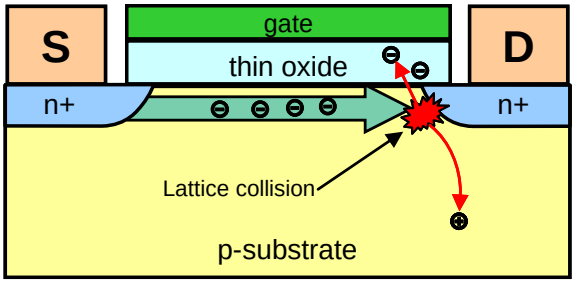

Figure 2: Hot Carrier Injection, electrons hit the lattice and get deflected to the thin oxide.

Hot Carrier Injection, electrons hit the lattice and get deflected to the thin oxide. This causes a shift in device operating point (V_T shift) and therefore decreases its conformance to nominal values. Over time the changes in device characteristics are irreversible and ultimately damage the device. Figure 3 depicts the HCI process.

In case of high concentration of electrons trapped in thin oxide, a gate leakage can also occur. The process of hot carrier injection is well described in the literature. More information about it can be found in (see [2]).

Detection of the HCI is an example of a complex rule which does not fall into simple cross probed voltage check.

Operating Conditions

Any manufactured IC is exposed to various environmental conditions which will either change its operation, cause malfunction or even cause physical damage. Most common factors are temperature, radiation, humidity, pressure and vibration. Other sources of failure include stress (mechanical and electrical) fatigue (often due to thermal expansion) and typical diffusion process that is present in any silicon based chip.

Not all of these conditions can be included in the simulation of the device, however introduction of stress checks (SOA), which include reliability information, can improve device performance in the field.

In most cases safe operation is recognized as voltage limitation which increases reliability and mean TTF. However, not always limiting of device operating conditions is necessary. Increasing voltage range, for example, can broaden the sensitivity or increase device power. In some circumstances, the device's real operating time can be calculated in seconds or even milliseconds (for example airbags) making the more stringent assumptions (lower voltage) for a given technology too limiting.

On the other hand, devices going through heating-cooling cycles (near car's engine) must be extremely resilient to stress, and their reliability limits have to be set in much stricter way.

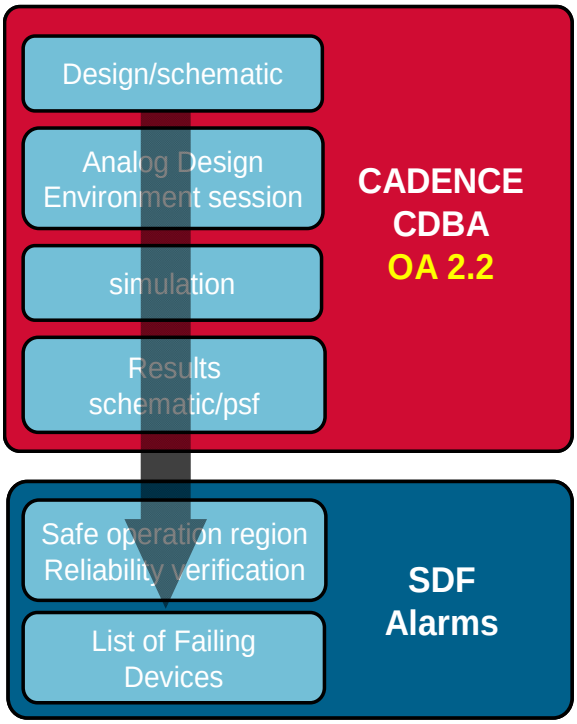

Figure 3: SDF Tool Flow

Integration of SDF Flow

Both aspects presented above can be addressed by providing a specific set of rules describing device operation. Incorporation of

these rules into a standard design flow, makes the products cheaper, better designed and protects company from unnecessary recalls. Figure 3 presents a desired flow integration that could fill the gap in the pre-layout validation stage with utilization of the existing mechanisms: design hierarchical schematics, simulations and saved states. The OA2.2 marked in yellow was a later addition to the project.

Figure 4 presents the GUI realization of SDF tool with 3 distinctive steps involving ADE setup, SDF setup and the SDF alarm browser.

The control of the SDF engine and specific techniques used for optimized execution are done through SDF setup involving several tabbed dialogs.

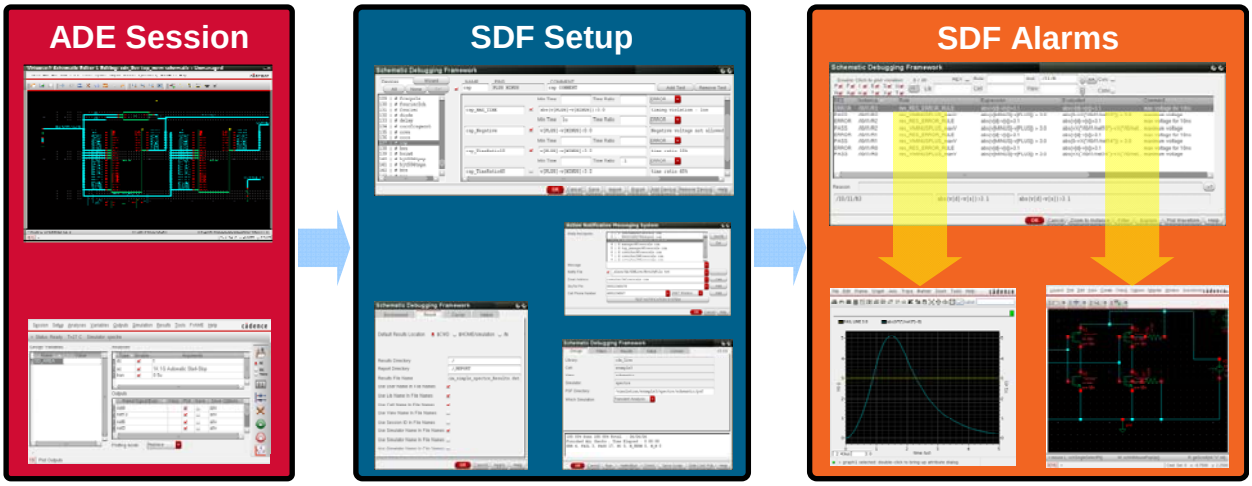

Figure 4: SDF Tool Main Components

SDF Setup

As it is presented in Figure 4 the SDF tool is integrated into ADE from which it obtains all necessary for the execution parameters. Since the tool is invoked from the ADE window its integration with ADE was a necessary step in development.

Integration with ADE

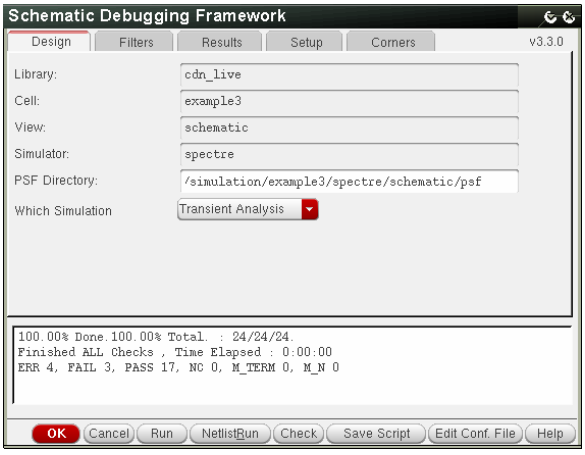

Figure 5: Main SDF Run dialog

- Top cell,
- Selected simulator,
- Used PSF/SST2 directory,
- The Simulation type

The bottom section of the dialog is used for the progress updates. It provides details about number of checks done so far and expected time to finish the current validation process. The run time estimate helps designers to better plan their own time.

Simulator support

Currently the standard version as well as Verilog counterpart of both commercial (including Spectre™ and UltraSim™) and in-house simulators are supported by the tool.

Verilog support

Verilog and VerilogA are supported only up to the port level of the design blocks. The internal variables of Verilog blocks are inaccessible to the tool as these are not stored in the PSF file in the acceptable form.

Although the Verilog data is not retrievable, the design can still be checked up to the port level, which means that the Verilog block is treated as a black box with wires attached. Currents and voltages on these ports can still be measured from the outside and therefore they can be used in the expressions. a special technique had to be used for mapping of the Verilog block ports.

Design Hierarchy support

The SDF tool was designed to support both Analog and Mixed signal simulations. The first style of simulation is purely schematic based and requires only top level schematic (Top-Design) and two lists :

- switch view list
- stop view list

the stop view list ends the hierarchy search while the schematic view list, shows the views which the search engine should not descent into. The mechanism is identical with the ADE hierarchy search.

In case of Mixed signal simulation, which includes Verilog/VerilogA blocks – a config view is necessary to describe the hierarchy. The SDF is able to parse the config views and extract the hierarchy from these, thus enabling the mixed signal simulation and its further analysis.

The selection of the hierarchical parser is automatic and does not require user's input. The information is extracted from the ADE session, and therefore it follows the setup already prepared by the user.

All names in the filter fields (both the setup and the result browser) support standard “slash” notation. Such notation is kept common between different simulators and the translation to specific notation used inside the PSF file is done again automatically by the tool.

Simulation types

The SDF tool concentrates on 4 simulation types:

- Transient
- DC-OP
- DC-Sweep
- AC-Sweep

The support of the given type of simulation is dependent on the simulator used. For example Ultrasim(TM) provides only transient option (DC-OP still has unresolved problems).

Speed Considerations, Filter Options

Since the SDF tool needs to parse large PSF files, to speed up the process, a designer can narrow the evaluation of the circuit only to its specific sections. Both cellview name filters and block hierarchical path filters are available. Figure 6 shows the example of available filters. Multiple blocks and Multiple cellviews can be specified.

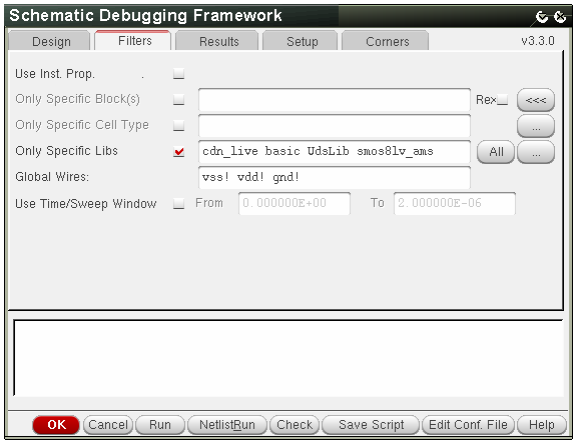

Figure 6: Filtering options of the SDF

The time window is another enhancements that provided a way to narrow down the analysis to the portion of simulation which is interesting to the designer. a typical example is the problem of charge pumps that achieve full operating conditions only towards the last 5% of the simulation. By default the time window shows the time limits selected in ADE so that they can be easily updated.

Running the engine

The SDF tool can be run in one of 3 modes. Run, Netlist and Run and Check.

First and second option invoke simulation and netlisting and simulation respectively Third option is most often used as it utilizes the results already stored in the PSF directory. Selecting that option brings the PSF results selection dialog which shows only valid results.

Additional efficiency related options (including cache) are available in the SDF Options dialog (not presented here).

Multiple Setups

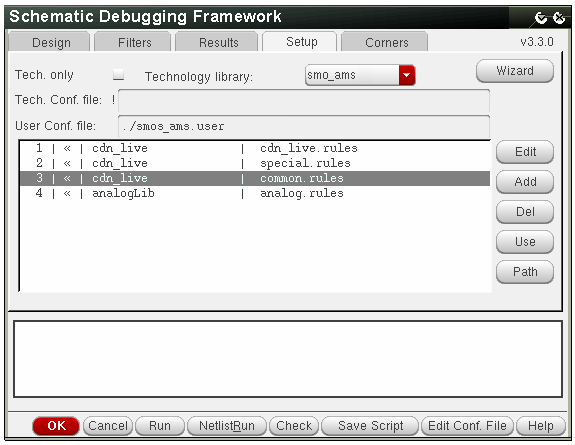

Figure 7: Multiple Setup for multiple libraries

Figure 7 illustrates multiple setups, which can be simultaneously used by designer for description of several libraries and even the same library elements. In this example 3 setup files are utilized each of which can be assigned to the specific library and turn on/off at will. The multiple setup files can contain different set of rules that correspond to more or less stringent conditions. The setup file can represent original library setup or a shadow setup.

SDF Alarm Editor

The next important element of the SDF Setup is the SDF alarm editor. The editor provides means of generating the SDF Alarm files or editing existing ones.

Alarm Expressions Generation

By default SDF Alarm files are delivered with the technology library and are available to the user without any special selections. To verify the alarm conditions or to select only special cases, user can open the Alarm Editor dialog. Figure 8 presents the SDF Alarm Editor with the standard Cadence *analogLib* library as a reference.

The dialog was designed to provide further automation and better integration with the rest of SDF framework. Both for user libraries and for the technology library, user can create arbitrary number of expressions. Each expression belongs to 1 of 3 categories: warnings, infos and errors (marked yellow). To accommodate more sophisticated needs the expression evaluation can be gated through a timing window (Min Time) – this option is available in some of the simulators (for example Spectre). For more complex conditions additional attribute can be set (Time Ratio), which allows for duty cycle type checks. That option is not available as a check in other solutions.

Figure 8 Example of SDF Alarm Editor.

Utilization of the dialog instead of text editing the setup files provides one more advantage to the user, it verifies the syntax of the expressions and saves the setup file always in a correct format. This way, unnecessary mistakes are reduced to minimum, allowing both designers and developers to concentrate on the issues at hand and not complexities of the setup files syntax.

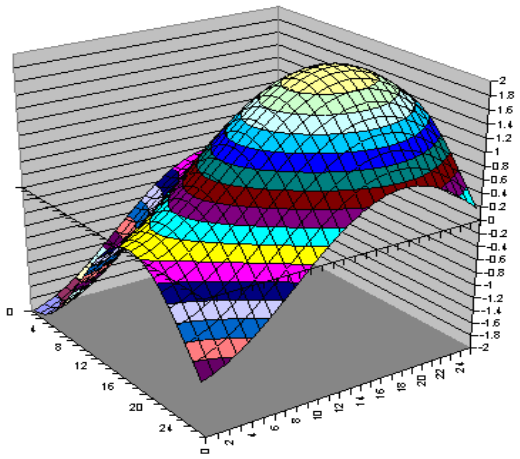

Figure 9: violation space

Utilization of the time window gating function with the time ratio, allows for very accurate examination of the phenomenon mentioned before - the HCI conditions.

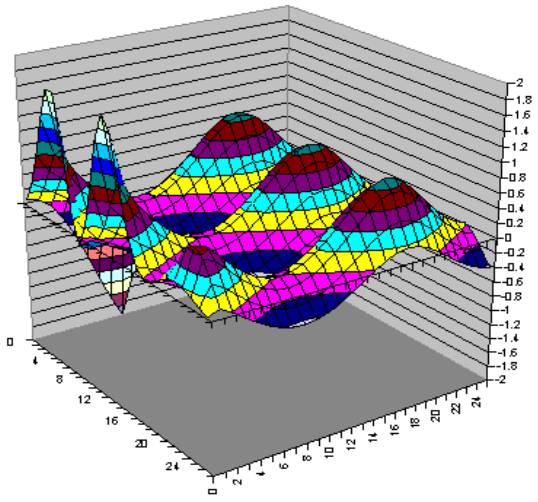

Figure 10: violation

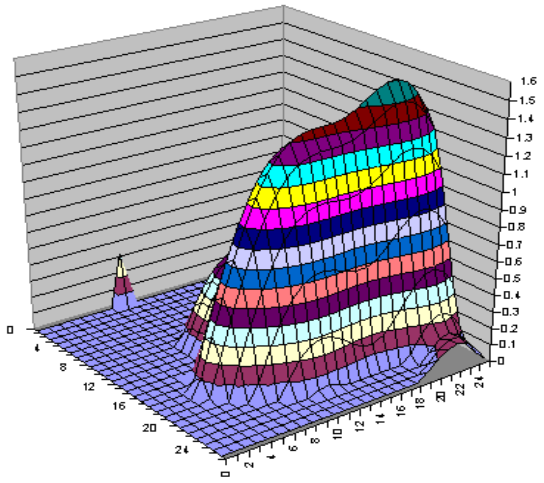

Figure 11: Resulting Violation Space

Complex Rules

The SDF Alarm editor allows for creation of complex dependencies between parameters and simulated values of the sub-circuit. These complex relations are often described in 2-D or 3D space. Figure 9 presents the violation space described for a 2D case. Figure 10 shows the simulation results for the given pair of variables (2-variable DC-sweep), while Figure 11 shows the effective alarm space.

It is important to understand that in the course of the transient simulation for example, only a few of the nodes of the violation graph can be reached as the transitions will be represented by a line rather than surface.

An example of a complex alarm - HCI

HCI rules belong the non-trivial expressions. Trivial expressions can be described with following formula:

$$V_{XY} > V_0$$

where V0 is a constant and X and Y represent terminals of the device.

The HCI conditions require equations that verify relations between 3 or more terminals of a given device.

Methods describing the device HCI are complex (see [4]) and the degradation evaluation methods became subject of several US patents .

A simplified analysis of the HCI process of the electron injection into the oxide space can provide a model of that phenomenon based on just 2 major factors:

- the source-drain voltage, which is responsible for the electric field accelerating the electrons to the speeds resulting in high energy collisions with the lattice
- the gate potential which, attracts the electrons, towards the gate and traps them in the oxide.

The force (proportional to the potential of the gate) helps electrons, which effectively land in one of three regions:

- close to substrate – a shallow penetration
- in the middle of oxide – oxide degradation or memory function (in case of flash)
- crossing the oxide – leakage current

As the HCI damage is considered mostly in the second case (degrading Vth) when electrons become trapped, the formula defining the forbidden region of operation can be represented as follows:

$$V_{GB} \approx a + b * V_{DS}$$

which results in the expression:

```
(v[g]-v[b]>a+b0*(v[d]-v[s]))&&
(v[g]-v[b]<a+b1*(v[d]-v[s]))
```

where a, b0 and b1 represent appropriate coefficients for a given technology.

The conditions can also be subjected to timing gating (for example tD>200ns). Figure 13 shows how time gating can be achieved.

Alarm Expression syntax

Below an example of simple rules used in the notation of the alarm expressions is presented:

- Terminal names are accessed through square brackets : example v[s] is a voltage of the source terminal with the reference to 0-gnd.
- Cdf parameters are accessed with dollar sign : for example $iso_105V represents isolation rule 105 volts. At the same time $l refers to a length.
- Simulation parameters and variables are accessed through a @ symbol : for example @temp represents the temperature (note: in some simulators temp cannot be used as a variable).
- Conditional expressions are available with “and“ && and “or” || support.
- Calculator functions (sin, atan etc) are available.

An Example Circuit.

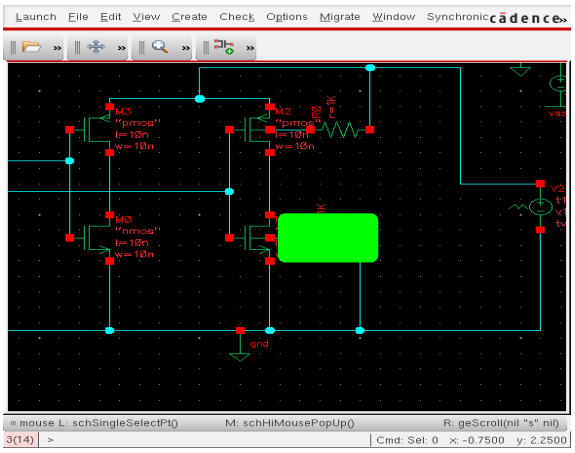

Figure 12: example circuit

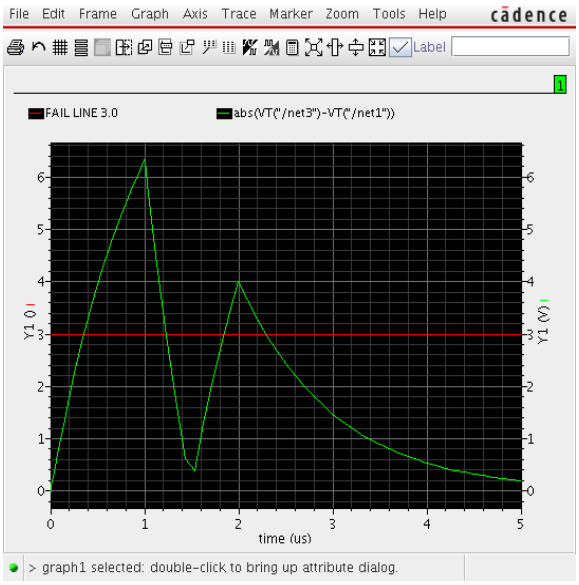

Figure 13: Example of time dependent violation

Example of a violation detected by the SDF is shown in Figure 13. The violation will not trigger the *cap_MAX_TIME* from Figure 13, however it will trigger the *cap_Time_Ratio10* as the time

spent in the offending region crosses 10% of the total time of the simulation.

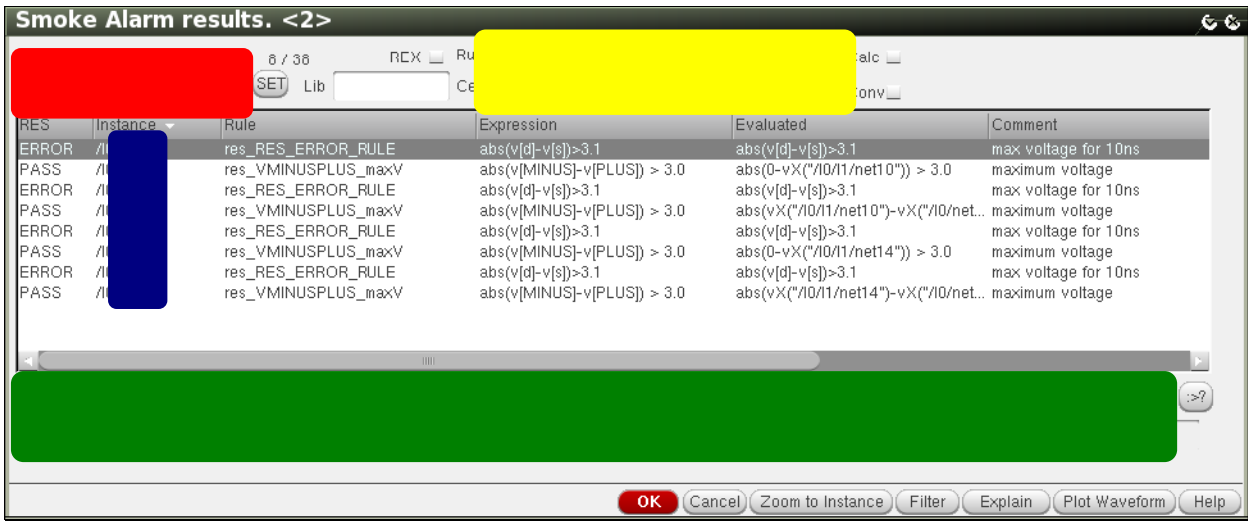

Figure 14: Example of SDF Alarm Results Browser

SDF Alarm Developer Support

To minimize the time needed for development of the Alarm file, special wizard has been provided. The wizard can generate all basic checks for all devices in the library, which have a symbol view.

It is important to use wizard especially in case of devices which have numerous inputs. The wizard can generate all combinations of inputs reducing manual overload for minimum. In most cases (which are simple voltage checks) what is left for the SDF Alarm file developer, is to fill in the actual values for limiting voltages

Result Browser

Violation Debugging

As in any other aspect of design, the debugging of encountered problem (in this case SDF violation) must be clear, precise and fast. All these aspects were taken into account during development of the main debugging window of the SDF tool. Designer is able to narrow down the problem by using a set of filters and then follow the problem to its source.

Filters

Filter fields are marked in Figure 14 with yellow and red frames. The first set of filters (red section) due to a limited dialog space has been compacted to a set of single letter shorts. Meanings of these abbreviations are explained in a separate dialog accessed through the SET button. The filtering options in this section refer to a type of results, errors, warnings, unfinished checks, evaluation errors, syntax errors, used file and suspicious conditions such as for example *always fails*, and *never fails*. The last two conditions are especially useful during debugging of the SDF Alarm files, the *always* and *never* conditions can show when the rules are not correctly defined. For example the following rule:

```
abs(v[s]-v[d])==0.6
```

seems to be a valid expression however in practice it will never fail. The reason is the representation of the floating point numbers. This is because the values often become truncated and direct comparison of 2 floating point numbers cannot result in true statement (see [6]). Most likely the desired formula needs one more abs():

```
abs(abs(v[s]-v[d])-0.6)<1e-5
```

Here the comparison is done correctly as it allows for a margin of error. Thanks to the mentioned special category of results : Never Fails – the designer is able to catch not obvious problems and modify the expressions accordingly.

The second filter (yellow section) allows for extraction of results for only specific sub-cells in the design.

Providing the instance partial hierarchical name in the Inst box allows the browser to filter out the names which do not contain such string. In the presented example only 8 violations for 4 instances are shown out of total of 38. The section marked blue, shows the matching portion of the instance names. The reported alarms can be also filtered by device library, cell and view names independently. Finally they can be filtered with the Rule Name. All filtering is turned off by default.

In addition internally developed *Form Messaging System* speeds up multiple selections in case of large result arrays. This helps to keep the speed of the tools GUI with the pace of a designer.

Alarm Details

The simple True/False information about the existing violation might not be enough for designer to make an informed decision about fixing the cell. To quickly find out how the specified condition is violated, a plot of the result can be accessed by double clicking on the result. Figure 16 shows such a waveform generated for the failing condition marked in Figure 8.

Another important feature of the SDF Alarm Results Browser is the access to the schematic design also directly from the Results window. In this case pressing *Zoom to instance* opens a schematic with focus on the violating instance.

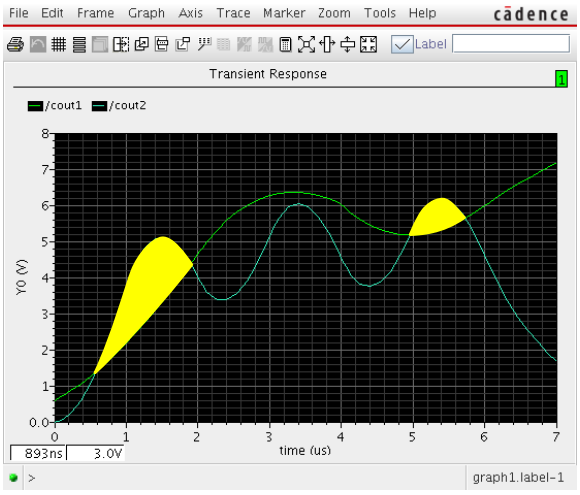

Figure 15: Example of complex violation regions

The bottom portion (marked green) of the Results dialog is used for clarification of obtained results. It shows the original equations, the translated expression and the reason for error. The *Reason* field is especially useful in case of calculation problems. Most common are the problems in the formulas containing wrong terminal name.

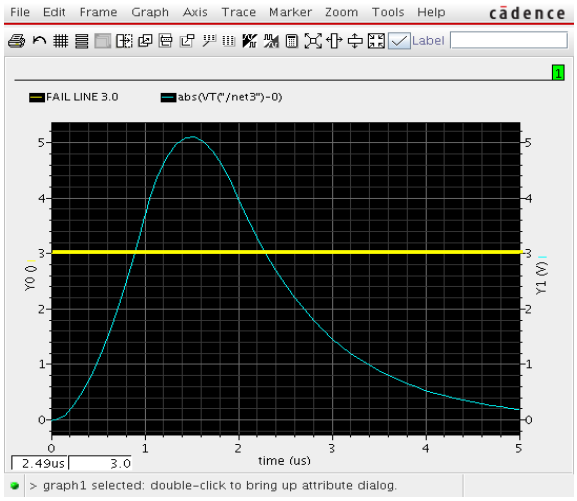

Figure 16: Example of Violation plot

SDF Tool Extra Freatures

Notifications

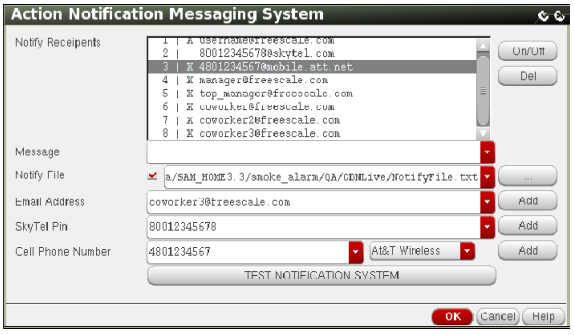

Figure 17: Notification - messaging system

The messaging system provides further improvements to designers experience with the tool. The SDF framework can notify multiple receivers about several stages of the check completion. In case of the checks which require overnight run, it is important to be able to stop or restart the tool if running it ends with failure (for example in case of an unsuccessful simulation of a block any further analysis requires designer's manual intervention).

Components of the SDF

The tool components of the SDF are presented in Figure 18. Each of these components has to

realize a specific function inside the Cadence Design Environment.

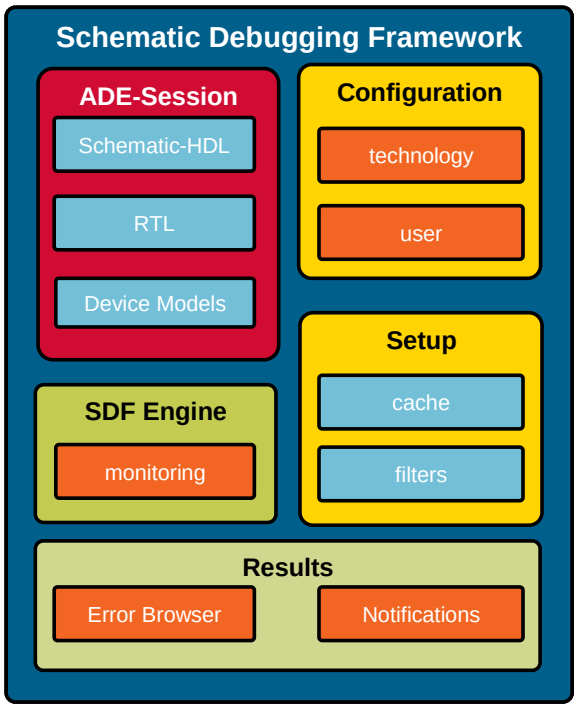

Figure 18: Components of the SDF

Summary

The issues presented in this session represent only a part of numerous design validation issues. Many of them can be accurately addressed by providing appropriate methodology with strict design flow steps. As it has been shown, incorporating the SDF tool into such flow, can enhance the flow and increase designer's productivity.

Support of CDF parameters, design variables, terminal currents and voltages, provides a way of generating both simple and complex expressions, which can cater different design needs. At the same time a helper mechanism in form of several wizards turns the chore of writing the SDF Alarm files into a quick and yet highly reliable task.

In addition unification of the approach to different simulators allows for simplification of validation steps in case of design migration.

All mentioned features make the SDF a great choice for designers as it complements existing flows and allows for reuse of already generated test cases, minimizing this way the setup overhead.

The key reason for success of the SDF utility is partitioning of the configuration process into two separate tasks. The first, realized by a team of experts (the device characterization group), gave solid and reliable set of SDF Alarm rules, the second, performed by designers, allowed for full flexibility in selecting checks and minimization of unnecessary overhead. Due to this division, the designers get a full product, which is ready to work out of the box with minimum effort on their side.

Providing such infrastructure makes sure that *batteries are included*.

Bibliography

[1] “Constant-charge-injection programming for 10-MB/s multilevel AG-AND flash memories”, Kurata, H.; Saeki, S.; Kobayashi, T.; Sasago, Y.; Kawahara, T.;Symposium on VLSI Circuits Digest of Technical Papers, 2002.

[2] “Physical model of drain conductance, g_d, degradation of NMOSFET's due to interface state generation by hot carrier injection”, Kurachi, I.; Nam Hwang; Forbes, L.; IEEE Transactions on Electron Devices, Volume 41, Issue 6, June 1994

[3] “Reliability effects on MOS transistors due to hot-carrier injection”, Kueing-Long Chen; Saller, S.A.; Groves, I.A.; Scott, D.B.;IEEE Transactions on Electron Devices, 1885

[4] Virtuoso® UltraSim Simulator User Guide version 6.1, November 2006, Cadence Design Systems Inc.

[5] Virtuoso® Circuit Simulator User Guide version 6.1, June 2006, Cadence Design Systems Inc.

[6] “SKILL Language Reference”, Cadence Design Systems, June 2004

[7] “SKILL Language User Guide”, Cadence Design Systems, June 2004

[8] Virtuoso® Analog Design Environment SKILL Reference, 2007, Cadence Design Systems Inc.