A Designer-Customizable Design Environment for Analog/Mixed-Signal Circuit Design

M. S. Toth
2268 S. 12th St.
Allentown, PA 18103
(610) 709-1239
mst@agere.com

R. V. Booth
1247 S. Cedar Crest Blvd.
Allentown, PA 18103
(610) 709-2324
rvbooth@agere.com

1. Introduction

Analog/Mixed-Signal (AMS) circuit design is an iterative process involving changes to component values, circuit topology and other aspects of a particular design, with the goal of achieving all of the required specifications. Modern circuit design is typically supported by a suite of computer tools for schematic entry, circuit simulation, and data analysis software. Many vendors supply integrated tool platforms where various software tools are linked together. While these platforms do supply much of the required design support, there are many times when it is necessary to work outside of the platforms with other tools. These platforms also enforce a certain design style, although there are usually some limited possibilities for user-customization. The iterative aspect of AMS design almost requires that customization features be used to shorten the work required for common tasks. Once a design has been completed, it must be exercised (by simulation) in a detailed way across processing, temperature, and supply variations, in order to flush out problems that might actually appear in the field. Often a designer acquires a good idea about which particular process/temperature/supply condition is the most problematic only after some of the initial work with the design. After this initial phase, it is most efficient to continue design at this worst-case condition. Finally, a prominent feature of the design world is that each designer tackles problems in different ways: in different order, with different tools, with different styles. All of these aspects of AMS design point to a possibly better solution than the standard vendor platforms: a flexible, customizable, and extensible environment, where a designer designs the platform himself.

For netlist-driven design, some of the most frequent design operations are: creating a netlist after modifications to the design, modifying the netlist to reflect an optical shrink in processing, designing the computer experiments, running computer simulations, plotting results, analyzing results, and using the results to improve the design. In this paper, we describe a Tcl/Tk-based design environment, which is used to control and integrate the various design tools, as well as provide additional analysis and plotting capability. The design environment is called Camelot, and has demonstrably improved designer efficiency, shortened design time, and improved product robustness.

2. Analog/Mixed-Signal Circuit Design Tools

At the core of AMS circuit design is a transistor-level analog circuit simulator. At Agere, that tool is Celerity, named for its designed-in transient simulation speed. In addition to Celerity's superior simulation speed, convergence, accuracy and analysis features, is its extensibility. Celerity is based on the Tcl/Tk extension language[1] - it is a Tcl/Tk extension.[1] The Tcl/Tk extension language was used for Celerity because it provided a robust and easy way to glue together portions of the tool, and it provided customization and extension capability.

For schematic capture and netlist creation, Agere employs several tools. Schema is one of these; developed originally at Bell Laboratories, it is still widely employed at Agere. It is GUI-based, and is extensively user-customizable, although this capability is somewhat complex.

For waveform viewing and manipulation, Agere widely employs AdvPlot, which was developed in conjunction with Celerity (and its predecessor, Advice). AdvPlot can be programmed and driven by scripts, allowing it to be controlled in a very useful non-GUI manner. In addition, it provides numerous waveform analysis options.

Camelot was originally developed as a tool for device-characterization, device model parameter-extraction, and data analysis tool. It has also been extensively used as a platform for designing the compact-models which are embedded in Celerity. Just like Celerity, Camelot is a Tcl/Tk extension. This common background provided a straight-forward communications path between the two

1. Effective Tcl/Tk Programming : Writing Better Programs in Tcl and Tk (Addison-Wesley Professional Computing Series) by Mark Harrison, Michael J. McLennan Paperback - 432 pages (November 25, 1997) Addison-Wesley Pub Co; ISBN: 0201634740

tools, based on a TCP/IP extension to Tcl/Tk.

Celerity's user-interface is command-line based: a prompt is displayed, the user types in a (interactive/simulation) command, and a carriage-return signifies the end of a command. Camelot also provides a command-line interface, in addition to other use modes: script-sourcing, script-execution, custom graphical-user interface. The GUI capability is provided by Tcl/Tk, and since Celerity is a Tcl/Tk extension this ability has always been available: early versions of Celerity included a basic GUI interface, but the developers found that it was not as popular as the command-line interface, and its development did not continue.

Camelot is used to control Celerity by starting up a slave process, issuing commands to the simulator, and receiving the results for post-processing. Since Camelot is Tcl/Tk-based, a wide range of commands are available, including such programming control structures as **`while`**, **`for`**, and **`foreach`**. Within a control loop, for example, one might change various circuit component values and execute simulation commands. Celerity does itself provide limited looping capability by a special **`.LOOP`** simulation command. However, the parameters which can be swept are limited, nested loops are difficult to implement, and interrupts can put the tool in an undefined state.

In this paper, we describe a customized designer-produced design environment which was enabled by Camelot. It consists of several Tcl/Tk scripts which help the circuit designer control the simulator and view results in an efficient way. Using this interface, the designer can focus more of his attention on circuit design rather than manipulation of the simulator and the plotting package. Compilation and post-processing of simulation results is straightforward. Processing case-combinations and simulation temperature are mouse-selectable. Each analysis is defined as needed and once defined, can be triggered by a GUI-button. The analyses which are designed at simulation time are stored in a single database which can be used and modified later.

A major feature of this design environment is that it is circuit-specific, in recognition of the fact that every circuit has unique simulation needs. While vendor-supplied platforms attempt to provide all design resources in one large system, a circuit-specific platform is much simpler, easier to navigate, and easier to maintain.

3. Camelot/Celerity Design Environment

Figure 1. is a schematic of the Camelot/Celerity design environment. In the figure, software tools are pictured as ellipse, files are pictured as rectangles, GUI-panels are pictured as rounded boxes. At the center of the arrangement is the Camelot/Celerity interface: a two-way communications pipe where Camelot issues simulation commands to Celerity and Celerity returns the results of the command back. To initialize the environment, a start-up script is sourced immediately after invoking Camelot. This script provides procedures which initialize variables, and create the two GUI panels: one for selection of process libraries and temperature, and the other for simulation analyses. The start-up script also provides generic analysis commands.

A typical design project involves numerous cycles of the following sequence: a circuit is initially designed or modified using the schematic capture tool, and a new netlist is then created. The netlist is read into Celerity after libraries for the models are specified. Assuming no errors occur at this stage, the designer sets necessary circuit parameters, such as input voltages and currents. Then an analysis is performed, and results are viewed or post-processed. In

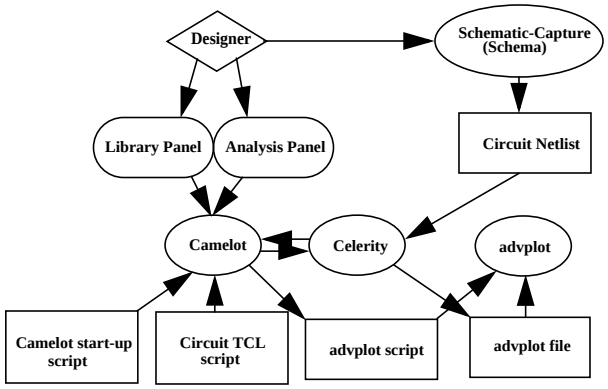

Figure 1: Camelot/Celerity Design Environment

manual form, these cycles can easily get out of hand, since interactive commands are changing the parameters of the circuit. Even in batch script form, it is difficult to maintain a database of analyses.

The Camelot/Celerity interface automatically maintains, in script form, the series of operations which are necessary to perform all of the various analyses. The interface also allows interactive changes to the circuit since Celerity is always running and available. This interactivity makes it possible to sweep almost any circuit parameter the designer is interested in.

The database of analyses is shown as the Circuit TCL Script in Figure 1. It is specific to the circuit which is under investigation, and must be written by the designer at the time of the investigation. Writing the script is straight-

forward, and at the beginning of the investigation, an initial Circuit script can be created by procedure, so that hand coding is minimized.

An example screen is shown in Figure 2. The circuit analysis panel has 15 different analysis commands in the form of a button. The first time this panel is created it contains three analyses: operating point, small signal analysis, and node voltages. Additional analyses are added by the designer, as needed. Selecting one with the mouse executes the analysis for a temperature and library file shown in the library panel. More than one library can be selected at one time. If more than one library is selected, the analysis requested is done for each library and the results are collected in one output file which is then viewed automatically by the plotting package.

For each of the three default analyses, results are displayed in a new window, as shown in Figure 4 for the default Operating Point analysis. The print button sends the text directly to the printer.

4. Methodology

Figure 3 shows a typical design flow during iterative fine-tuning. First, the schematic is modified using the schematic-capture tool. After editing the schematic, a new netlist is created. At this stage additional modifications to the netlist are often required. For example, the circuit may be designed in a parent technology but ultimately fabricated in an optical shrink of the technology. For design, this mapping can be accomplished by altering dimensions of devices in the parent technology. At Agere, additional tools are available for this purpose.

To read the circuit into the simulator, the library files must first be specified, then a read-in simulator command invoked. Errors in read-in caused by errors in the schematic require returning to the schematic modification step. Once a circuit is correctly read into the simulator, simulator commands are issued from a simulation script. The script usually includes the simulation command to read in the netlist. Post-process-

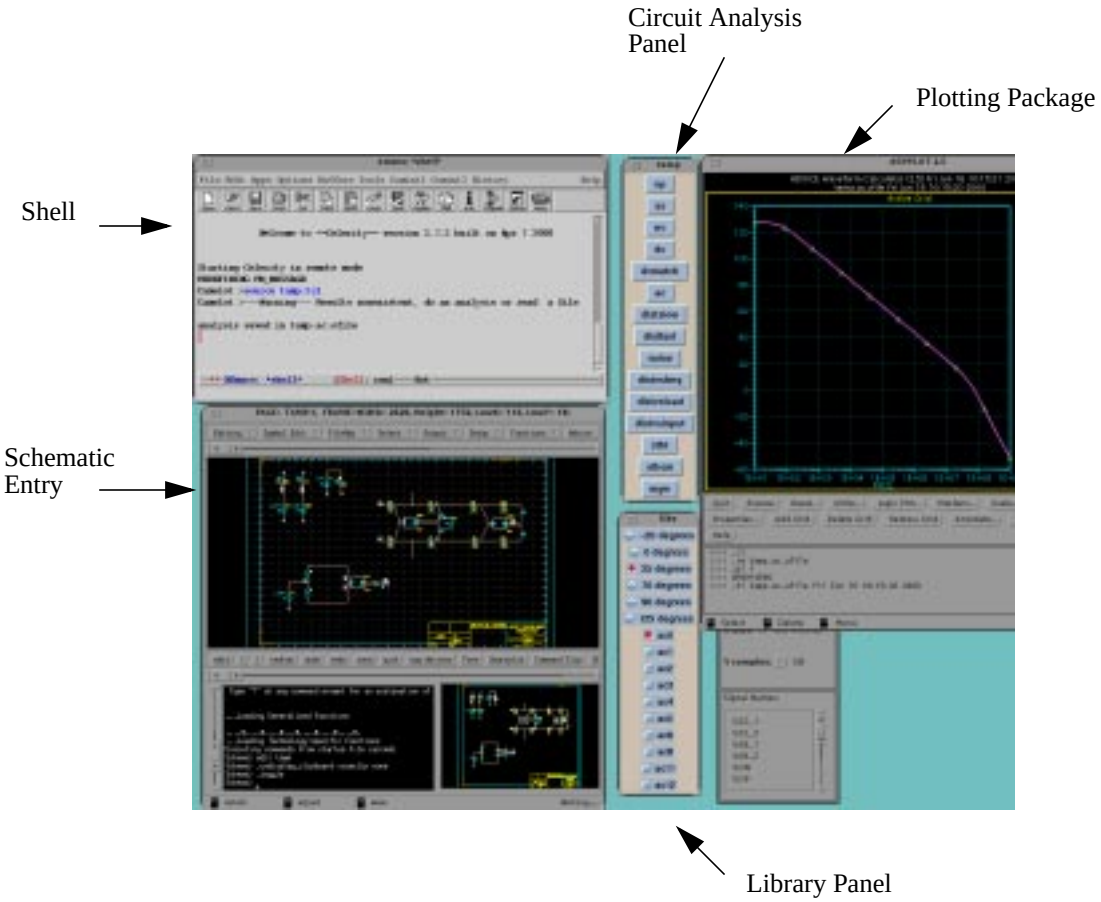

Figure 2: Screen Shot of Camelot/Celerity Environment

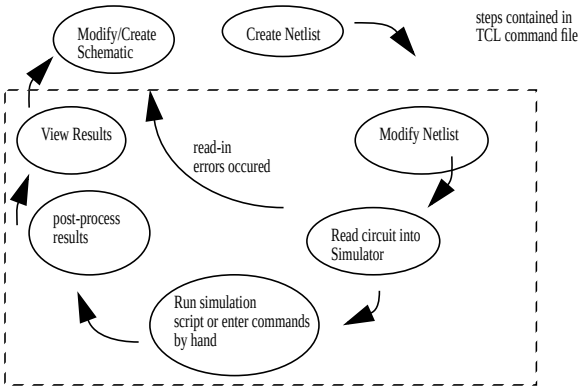

Figure 3: Design Flow

ing can also be included in the script.

Viewing results involves starting the plotting tool, reading the file into the plotter, and selecting the desired waveform. A script is also possible to use with the plotting package in many cases.

In the Camelot/Celerity design environment, all five steps shown in the dotted box are triggered by a single mouse click. The Tcl/Tk command file which is used to condense these steps into one is the only script the designer needs to maintain and use for all his analyses. Different analyses such as noise, and distortion, are managed in a switch statement. This makes it easier to maintain simulation scripts.

Figure 5 shows how these steps are performed in the Tcl/Tk command script. The **`view`** procedure in Step 5 is a pre-defined Camelot procedure which creates a script for the AdvPlot plotting package and then runs the plotter. The name of the desired waveform can be set as can be seen in Step 3. Setting the variable “name” to the name of the signal causes the plotting package to display this waveform automatically. It is easy to add these procedures as needed. In this way the system evolves to fit the needs of the designer.

The post-processing code shown in Figure 5 is used to create a plot of distortion vs. frequency, shown in Figure 6.

5. Using the interface

To reduce the overhead required to create a circuit-specific simulation environment, the designer can make use of the **`makeproc`** utility procedure. This procedure creates a starting command script, suitable for beginning the circuit characterization. Once the command script for a particular

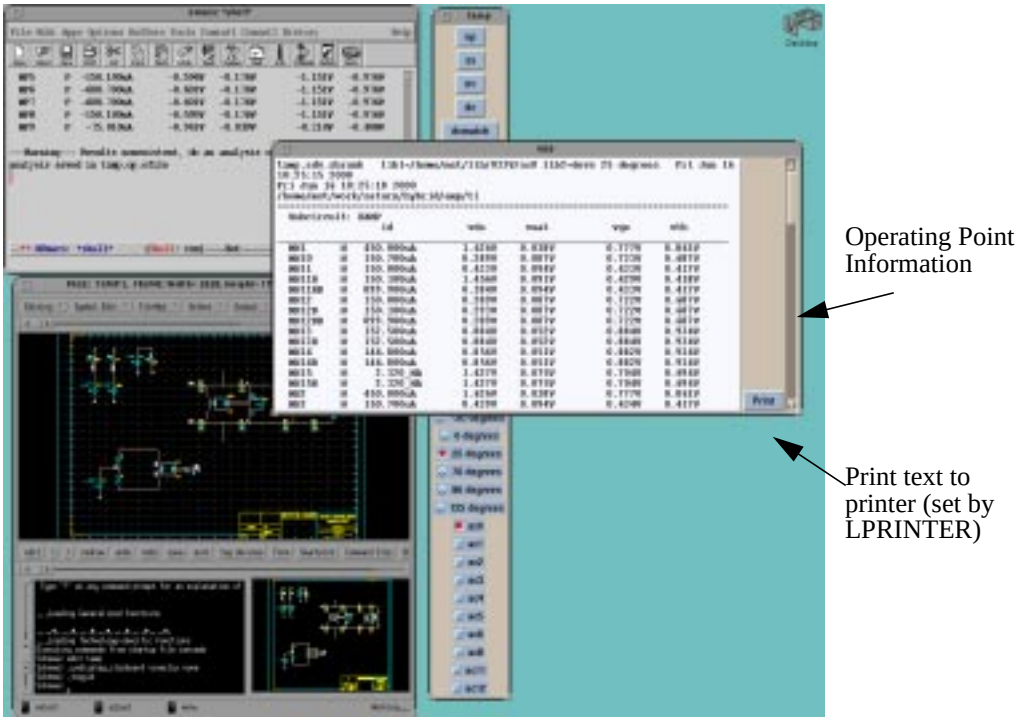

Figure 4: Screen Shot after performing Operating Point Analysis

Step 1: Modifying the netlist for an optical shrink

```
catch {exec ttcad shrink -o 1 tamp.adv}
```

Step 2: Reading circuit into simulator

```
rd tamp.pl.adv.shrunk "${libpath}/$libi" devs
```

Step 3: Running analysis

```
switch $anal {
    dc {
      c command ".dc vin -.1 .1 .001"
      set name VPN
    }
}
```

Step 4: Post-processing results

```
set r [c command ".newfour 1meg VPN"]
set i [lsearch $r "total"]
set d [lindex $r [expr $i+4]]
if {$d==0} {
    set d 0.0001
}
puts $fptr "$vin $d [expr 20*log10($d/100)]"
```

Step 5: Viewing Results

```
view $name tamp.$anal.ofile
```

Figure 5: Examples of Tcl/Tk in the command script

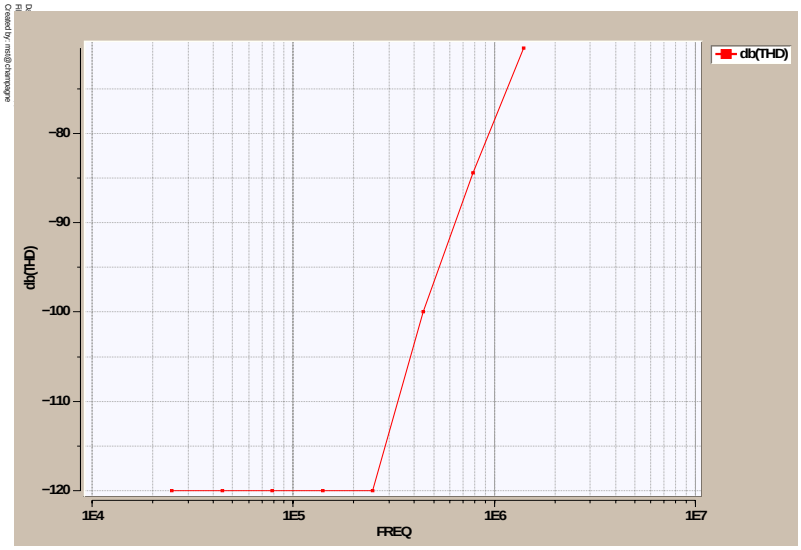

Figure 6: Distortion vs. Frequency

circuit has been created, other analyses are easily added manually. Appendix A lists an example of a command script that was created by **makeproc.**

The interface allows a designer to cleanly switch to post-layout sub-circuits, if they are available, so that parasitic capacitances and resistances can be taken into account. For each analysis, results are stored, so that they can readily be used in other analyses. For viewing simulation results, In addition to the **`view`** procedure described above, the interface provides its own plotting package and data-analysis routines. Two very convenient utilities for operating-point design are provided by the **`node-voltages`** and **`mosfets`** procedures. **`node-voltages`** displays a legible, sorted list of operating-point voltages. **`mosfets`** displays DC and small-signal operating point information for each MOSFET in the design, sorted by subcircuit of appearance. These utilities are uniquely provided by the Camelot/Celerity interface, since the information which is displayed comes directly from the simulation engine. For stability design, **`pzview`** is a built-in utility for displaying the poles and zeros in the design is provided. The utility cancels identical poles and zeroes for a particular transfer function, so that the basic features of the plot are not obscured. An example pole-zero plot is shown in Figure 7.

Recently, we have written a [incr tcl] Test Circuit class (TestCkt) which makes it easier to add circuit measurements to the set of tests in the circuit file. The instantiation and configuration of the class essentially replaces the command script described earlier. The test circuit class describes separate measurements for performing various analyses such as operating-point design, small-signal analysis, and stability analysis using the pole-zero Celerity command, . The TestCkt methods embed, monitor, measurement, and command simplify the work of writing the various measurrements. For the voltage-regulator example shown below, the temperature/case/ analysis panel and analysis windows are shown in Figure 8.

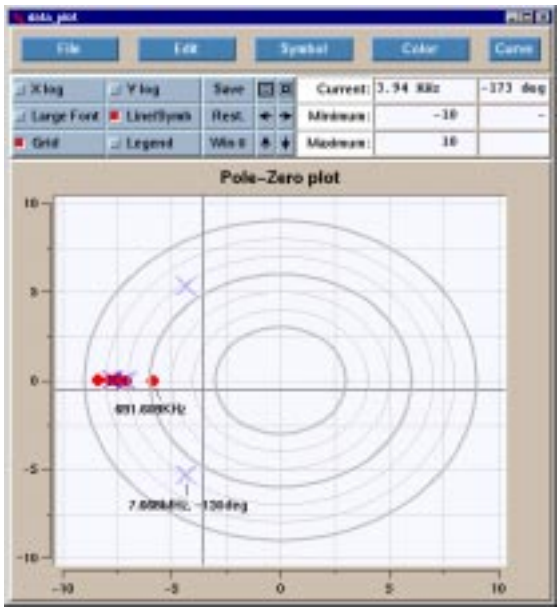

Figure 7: Pole Zero Plot Example

6. Conclusion

In this paper, we have presented an Analog/Mixed-Signal design environment, which allows a circuit designer to completely customize the set of analyses and operating point conditions, according to the circuit under investigation. For AMS circuit design involving iterative design cycles, repetitive characterization, or even optimization, the environment is very efficient, placing many simulation and analysis tools within easy reach of the designer.

The Agere Systems circuit simulation tool, Celerity, is Tcl/Tk-based, which allows a natural communications link to send and receive information between the simulator and control/analysis tool, Camelot. But other tools need not be Tcl/Tk-based to be integrated into the design environment. For example, the plotting package is driven by writing to plotting scripts which are read by the plotting tool. We have also integrated other circuit simulators and analysis tools into the environment. For each circuit, only one simulation/analysis script is modified when new tests are added, which is much easier to maintain than subdirectories of different simulation and post-processing run files.

7. Acknowledgements

The authors would like to thank Don Laturell for his comments while developing the interface.

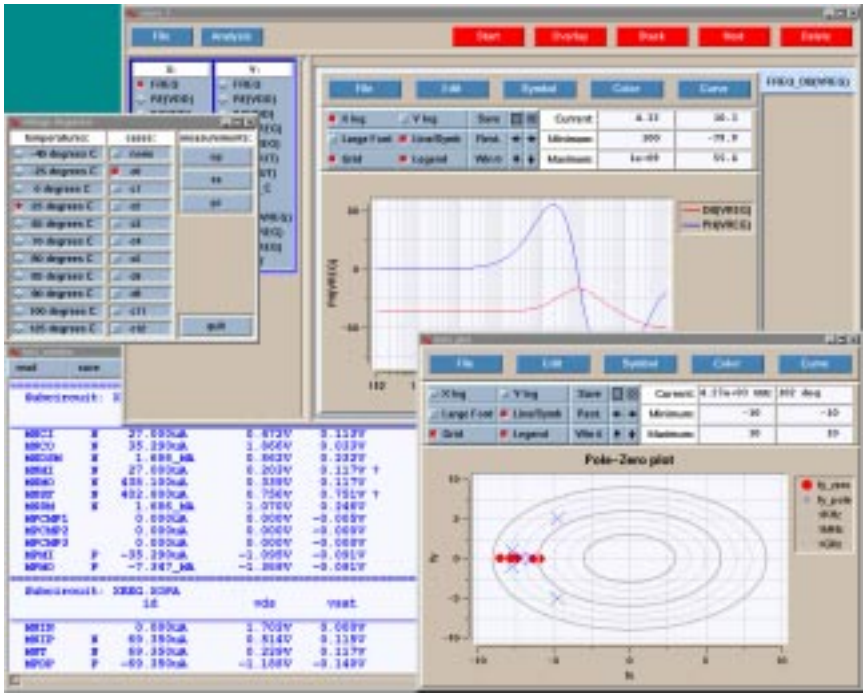

Figure 8: Temperature/Case/Measurement Panel and Analysis Windows created by [incr tcl] script