Mixed-Signal Tutorial VCS® AMS Mixed-Signal Simulation Using Verilog Top

SYNOPSYS®

Contents

Overview

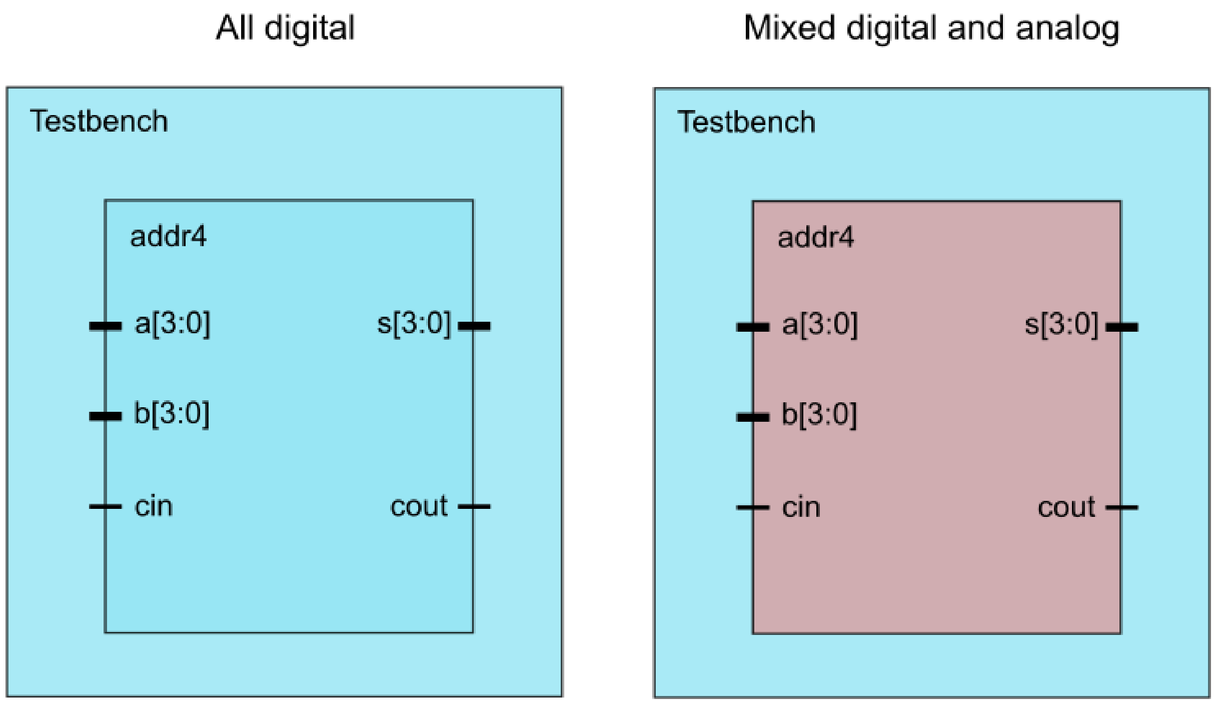
This tutorial demonstrates a simple VCS® AMS (CustomSim™ + VCS) mixed-signal simulation. The design is configured to use a top-level Verilog testbench that instantiates a SPICE cell.

Design Files

This tutorial uses a 4-bit adder circuit with three inputs (a, b, and cin) and two outputs (s and cout). The cin input is the carry-in signal, the s output is the sum of a, b, and cin, and the cout output is the carry-out signal.

A Verilog testbench instantiates the adder circuit and drives its inputs with stimulus. The testbench also compares the circuit outputs against the expected values.

Figure 1 *Tutorial Design*



The tutorial includes three script files to run the simulation:

- `runVcs` : An all-digital simulation run script that models the testbench and the adder in Verilog.
- `runVcsAms` : A mixed-signal VCS AMS simulation run script that uses a top-level Verilog testbench and includes a SPICE block.
- `runVcsAms_3step` : A mixed-signal VCS AMS simulation run script that uses a top-level Verilog testbench and includes a SPICE block in a three-step flow.

The following table describes the files contained in the tutorial.

File	Description
adder.v	Verilog behavioral description of the 4-bit adder
testbench.v	Verilog testbench for the 4-bit adder
addr4.spi	SPICE subcircuit definition of the 4-bit adder
vcsAD.init	Mixed-signal control file
vcsAD_3step.init	Mixed-signal control file for three-step flow
xa.cfg	CustomSim configuration file
runVcs	VCS-only compile and run script
runVcsAms	VCS AMS compile and run script with two-step flow
runVcsAms_3step	VCS AMS compile and run script with three-step flow
CLEAN	A script that removes all result and intermediate directories generated during the run. The original files in the tutorial are not deleted.
README	Brief description of this tutorial

Setting up the Environment for CustomSim and VCS

To run the tutorial, you must set the path and environment variables to run the CustomSim and VCS tools. To set the variables:

1. Source the `CSHRC_xa` shell script file provided in the CustomSim installation directory to set the path to the CustomSim executable. Use the `SHRC_xa` shell script if you are using the Bourne, Korn, or Bash shells.

```
# C-shell
% source /xa_installation_directory/CSHRC_xa

# sh / bash / ksh shell
$ source /xa_installation_directory/SHRC_xa
```

2. Set the `VCS_HOME` environment variable to the VCS installation directory and set the path variable to the VCS installation bin directory. The remaining steps are written for the C-shell.

```
% setenv VCS_HOME /vcs_installation_directory
% set path = ($VCS_HOME/bin $path)
```

3. For Synopsys Licenses, set the SNPSLMD_LICENSE_FILE environment variable to the license host and port, or the license file.

```
% setenv SNPSLMD_LICENSE_FILE port@host
```

```
- or -
% setenv SNPSLMD_LICENSE_FILE license_file_name
- for example -
% setenv SNPSLMD_LICENSE_FILE \
128@synopsys:7129@synopsys:$LM_LICENSE_FILE
```

4. Set the `VERDI_HOME` environment variable and include the path to Verdi in the UNIX path. The Verdi tool is needed to write out a digital FSDB file.

```
% setenv VERDI_HOME /verdi_installation_directory
% set path = ($VERDI_HOME/bin $path)
```

5. Set the path to the Custom WaveView executable. The Custom WaveView tool is used to view the waveforms created during the mixed-signal simulation.

```
% set path = (custom_waveview_installation_directory/bin $path)
```

Running an All-Digital (VCS-only) Simulation

The `runVcs` script contains commands to compile the digital design and run it. The following lines in the script compile the design:

```
# Compile
echo "verilog compilation"
vcs \
  -full64 \
  testbench.v \
  adder.v   \
  -debug_access+all \
  -l ./results_vcs/vcs.log \
  -o simvVcs
```

The VCS tool parses and compiles the testbench.v and adder.v files and applies the following options:

- The `-full64` option enables compilation and simulation in 64-bit mode.
- The `-debug_access+all` option enables debugging visibility and control, line-by-line simulation stepping, support for breakpoints, and other features.
- The `-l` option specifies the name of the log file (./results_vcsAms/vcs.log).
- The `-o` option specifies the name of the executable that is generated during compilation (simvVcs).

The following lines in the `runVcs` script run the simulation:

```
# Simulate
echo "run simulation - simvVcs"
simvVcs \
  +COMPARE \
  -l ./results/simvVcs.log
```

The `simvVcs` executable, generated by the `vcs` command, is invoked to run the simulation. The `+COMPARE` option is passed to the `simvVcs` executable and the testbench (testbench.v) checks for the existence of this parameter. If the `+COMPARE` parameter is specified, a comparison between the simulation results and expected values is performed.

Run the digital simulation with the `runVcs` script as follows:

```
% runVcs
```

Once the simulation is finished, open Custom WaveView and load the vcs.fsdb output file to view the waveforms as described in the next section.

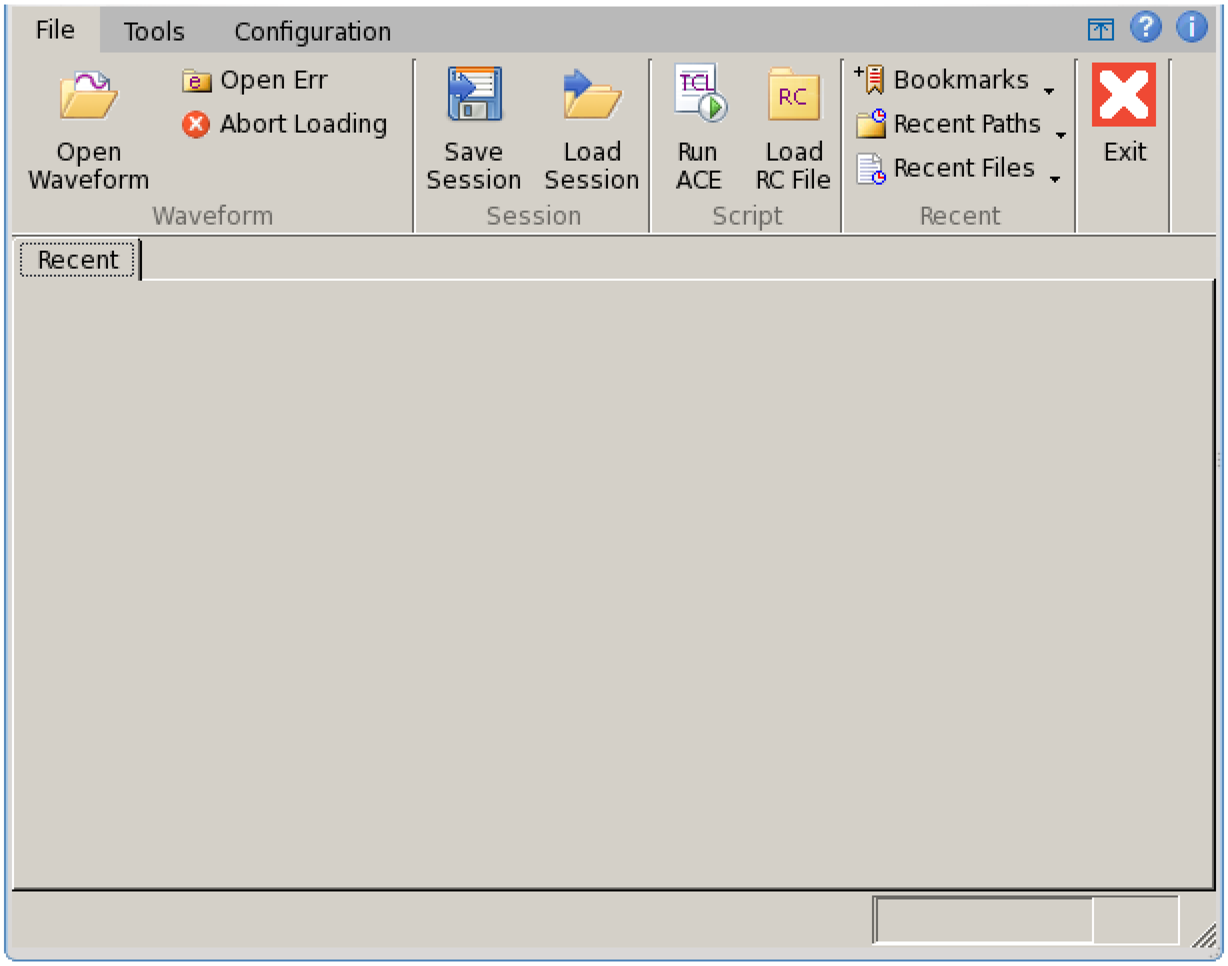
Viewing Waveforms in Custom WaveView

1. Run the following command in the UNIX command line to open the Custom WaveView GUI:

```
% wv &
```

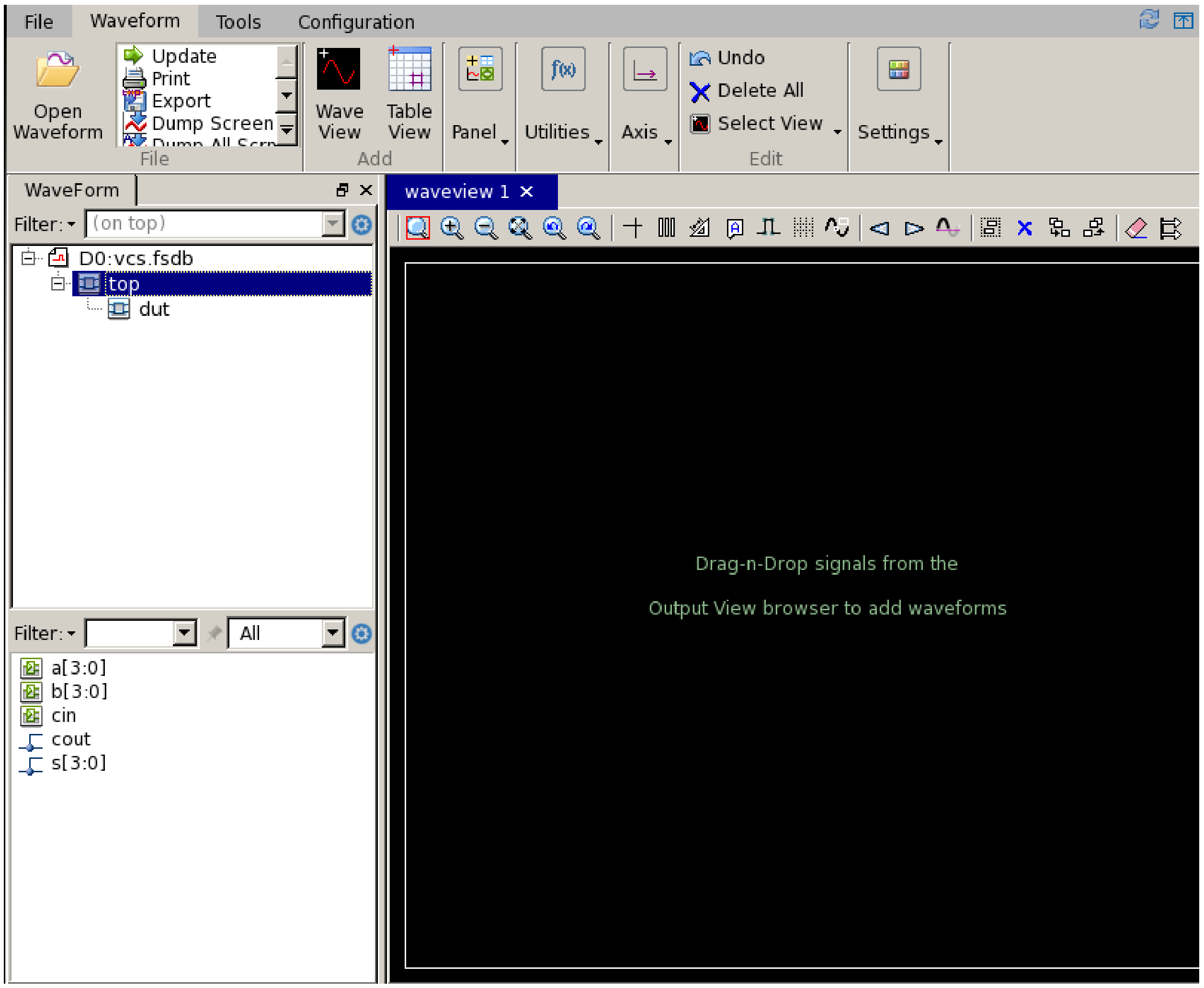
The `wv` command starts the Custom WaveView tool as shown in the following screenshot:

Figure 2	*Custom WaveView*



2. Click **Open Waveform** and load the results_vcs/vcs.fsdb file.
3. Double-click the “D0:vcs.fsdb” label in the top left of the **WaveForm** tab to view the hierarchy. Next, double-click the “top” label to view the next level of hierarchy.

Figure 3 *View Hierarchy*



Drag-n-Drop signals from the

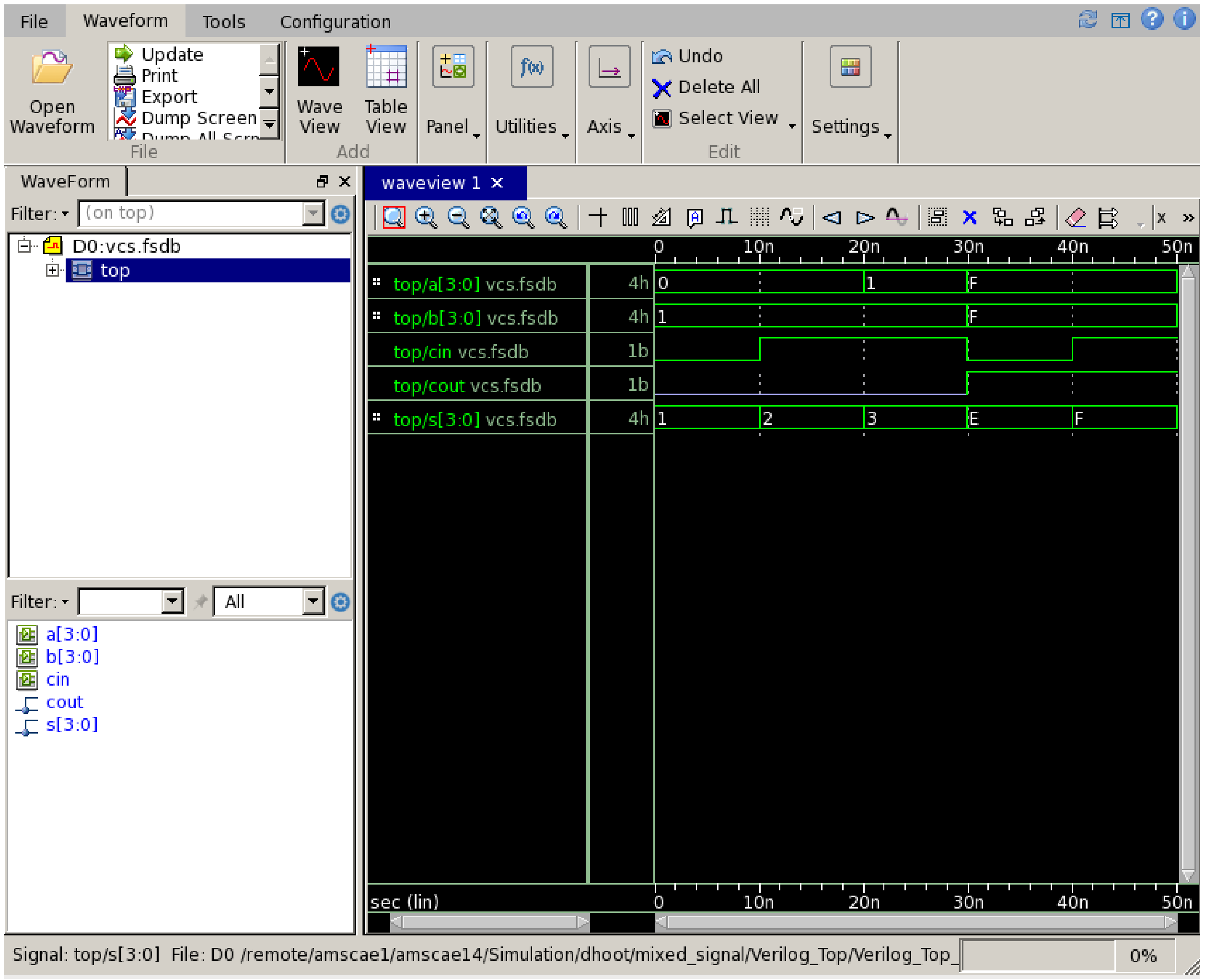
Output View browser to add waveforms

The Custom WaveView tool displays the simulation hierarchy in the top-left panel and signals for the selected hierarchy in the bottom-left panel.

4. Drag a module label from the hierarchy browser panel (top-left) of the **WaveForm** tab into the output view to display all input and output signals for the module. Alternatively, click a hierarchy label in the hierarchy browser and drag or double-click a signal labels from the signal list panel (bottom left) into the waveform view to display a signal waveform.

The following screenshot shows the waveforms in the “top” module.

Figure 4 *Top Module Waveforms*



Signal: top/s[3:0] File: D0 /remote/amscae1/amscae14/Simulation/dhoot/mixed_signal/Verilog_Top/Verilog_Top_

Verify the results for the adder by checking that the s[3:0] and cout output waveforms display the correct sum based on the a[3:0], b[3:0], and cin input signals.

Running a Two-Step Mixed-Signal VCS AMS Simulation

The mixed-signal VCS AMS simulation can be run in a two-step flow or a three-step flow. The two-step flow used in the `runVcsAms` script includes both the compilation and simulation steps.

The following lines in the `runVcsAms` script compile the design:

```
# Compile
echo "verilog compilation"
vcs \
  -full64 \
  testbench.v \
  adder.v   \
  -debug_access+all \
  -ad=vcsAD.init \
  -l ./results_vcsAms/vcs.log
```

The compilation options are similar to the options in the `runVcs` script with the following changes.

The `-ad` option specifies a mixed-signal simulation control file and directs the `vcs` command to include both Verilog files and SPICE files when creating the simulation executable. The vcsAD.init mixed-signal simulation control file contains user-specified mixed-signal commands. In this example, the `vcs` command builds the default executable named “simv”, because no output file name is specified with the `-o` option. The `-l` option specifies ./results_vcsAms/vcs.log as the output log file name.

The vcsAD.init file contains the following mixed-signal commands:

```
bus_format _%d;
use_spice -cell addr4;
choose xa -n addr4.spi -c xa.cfg -o results_vcsAms/xa/xa;
```

These mixed-signal commands are described below.

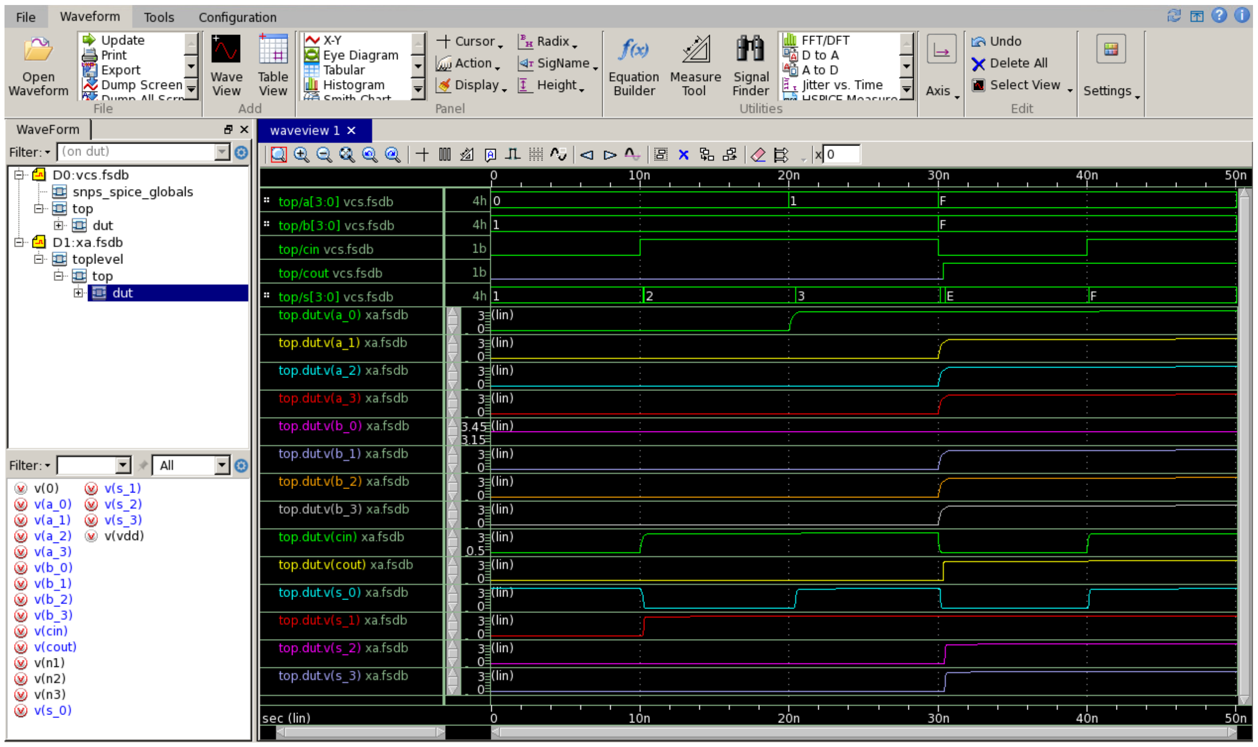
- `bus_format`: Specifies the character format for buses in the SPICE netlist. In this example, `bus_format _%d` specifies that SPICE ports such as a_3, a_2, a_1, and a_0 can be represented as a bus named “a[3:0]”. This allows a bus port named “a” in the Verilog testbench to map to SPICE ports a_3, a_2, a_1, and a_0 in the SPICE netlist.
- `use_spice`: Directs the tool to use a SPICE view for the specified cell. In this example, `use_spice –cell addr4` forces all instances of addr4 in the design to use the SPICE view of the cell.
- `choose`: Specifies the analog simulation engine and command-line options for the analog simulation. In this example, the xa CustomSim simulation engine is specified and the addr4.spi SPICE file and xa.cfg CustomSim configuration file are used by CustomSim.

To run the mixed-signal simulation (two-step flow), use the `runVcsAms` script.

```
% runVcsAms
```

Once the simulation is finished, open the Custom WaveView tool, load the results_vcsAms/vcs.fsdb digital output file from the VCS tool, and the results_vcsAms/xa/xa.fsdb analog output file from the CustomSim tool to view the waveforms as discussed in the previous section.

Figure 5 *Mixed-Signal Waveforms*



Running a Three-Step Mixed-Signal VCS AMS Simulation

The three-step flow performs analysis, elaboration, and simulation as separate steps. The `runVcsAms_3step` script contains the commands to analyze, elaborate, and run the mixed-signal design. During design analysis, the tool verifies the syntax of Verilog files and generates intermediate files for design elaboration. Any syntax errors in Verilog netlists are reported during this step.

The following lines in the `runVcsAms_3step` script analyze the design:

```
echo "verilog analysis - vlogan"
vlogan \
  -full64 \
  testbench.v \
  adder.v    \
  -l ./results_vcsAms_3step/vlogan.log
```

In this example, the `vlogan` command analyzes the Verilog files.

During design elaboration, the design hierarchy is constructed based on the information obtained during the design analysis step. At this stage, incorrect port connectivity or missing definitions for instantiated blocks in Verilog or SPICE are identified and reported.

The following lines in the script elaborate the design:

```
# Elaborate
echo "design elaboration - vcs"
vcs \
  -full64 \
  top \
  -debug_access+all \
  -ad=vcsAD_3step.init \
  -l ./results_vcsAms_3step/vcs.log \
  -o simv_3step
```

The specifications and commands in the vcsAD_3step.init mixed-signal control file are identical to those used in the two-step flow, except for the output directory specified by the `-o` option.

```
bus_format _%d;
use_spice -cell addr4;
choose xa -n addr4.spi -c xa.cfg -o results_vcsAms_3step/xa/xa;
```

To run the mixed-signal simulation (three-step flow), use the `runVcsAms_3step` script:

```
% runVcsAms_3step
```

Both two-step and three-step flows generate identical outputs. The waveforms generated in three-step flow can be viewed in a similar manner to two-step flow. Load the results_vcsAms_3step/vcs.fsdb and results_vcsAms_3step/xa/xa.fsdb files to view the digital and analog waveforms respectively.

Understanding Mixed-Signal Simulation Report Files

In addition to the waveform output, mixed-signal simulation generates a set of report files and writes the reports to a directory with a .msv extension. Use the report files to confirm and debug the mixed-signal configuration. The files can be used to understand how the tool interprets and connects the mixed-signal interfaces in the design.

hierarchy.rpt

The hierarchy.rpt file lists the hierarchical paths to all cells in the design together with their cell names as shown in the following example.

```
#  () is used to denote digital_cell, e.g. (digital_ff)
#  <> is used to denote analog_cell, e.g. <analog_ff>
#  {} is used to denote ams_cell, e.g. {ams_ff}
#  [] is used to denote verilog-A, e.g. [verilog-A]
top(top).dut<addr4>.x1<addr>.x1<xor2>.x1<inv>
top(top).dut<addr4>.x1<addr>.x1<xor2>.x2<inv>
top(top).dut<addr4>.x1<addr>.x1<xor2>.x3<inv>
top(top).dut<addr4>.x1<addr>.x1<xor2>.x4<xfer>
```

interface_element.rpt

The interface_element.rpt file contains information about the interface elements used at the mixed-signal boundary. The file also provides a list of resistance map files used in the design. The following is a section from the simv.msv/interface_element.rpt file.

```
Total number of resistors added by interface elements = 9
Total number of capacitors added by interface elements = 9

rmap_file 1 = /global/synopsys/xa_2018.09-sp2/include/rmapAD.init

d2a hiv=3.3v lov=0v node=top.dut.a_0;
d2a hiv=3.3v lov=0v node=top.dut.a_1;
d2a hiv=3.3v lov=0v node=top.dut.a_2;
d2a hiv=3.3v lov=0v node=top.dut.a_3;
d2a hiv=3.3v lov=0v node=top.dut.b_0;
d2a hiv=3.3v lov=0v node=top.dut.b_1;
d2a hiv=3.3v lov=0v node=top.dut.b_2;
d2a hiv=3.3v lov=0v node=top.dut.b_3;
d2a hiv=3.3v lov=0v node=top.dut.cin;
a2d loth=1.65v hith=1.65v node=top.dut.cout;
a2d loth=1.65v hith=1.65v node=top.dut.s_0;
a2d loth=1.65v hith=1.65v node=top.dut.s_1;
```

In the mixed-signal simulations, the circuit stimulus is generated by the Verilog testbench and connects to the addr4 SPICE view. To connect the digital testbench to the SPICE netlist, d2a interface elements are inserted by the tool. Similarly, the outputs of the SPICE view are used in the testbench and are connected by a2d interface elements.

mview.rpt

The mview.rpt file lists all cells in the design that have more than one view as shown in the following example:

```
; Has unresolved modules?
0
; For SPICE: case l by default
; Lists of modules: Verilog SPICE Verilog-A Adfmi
addr4 addr4 * *
; Verilog Top Module(s)
1 top
;done
```

Note that the addr4 module has both Verilog and SPICE views available.

port.rpt

The port.rpt file contains information about how ports are mapped when both a SPICE view and an HDL view are available for use by the simulator. The following is a section of the simv.msv/port.rpt file.

```
------------- Multiple cell views -------------

--------------- Cell: addr4 -------------------

   subckt addr4:  file "addr4.spi" line "57" ports=14
   verilog addr4: file "adder.v" line "2" ports=14

   verilog module addr4([3:0]a, [3:0]b, cin, [3:0]s, cout);
     input     a, b, cin;
     output    s, cout;
   subckt addr4 a_3, a_2, a_1, a_0, b_3, b_2, b_1, b_0,
         cin, s_3, s_2,
         s_1, s_0, cout

   use_spice -cell addr4 port_map(
               *=>snps_by_name);

   port_dir -cell addr4(
     input  a, b, cin;
     output s, cout
  );            //derived from verilog
```

The `port_map` entry in the `use_spice` command contains `*=>snps_by_name`; this indicates that all HDL ports without a specific mapping assignment are mapped to SPICE ports by their port name. If some ports are mapped using a `port_map` command in the mixed-signal control file, those are also listed together with the default mappings.

use_cell_view.rpt

This file lists all the cells that match each of the `use_spice`, `use_verilog`, and `use_vhdl` commands. The following is a section of the simv.msv/use_cell_view.rpt file.

```
#=====================================================
# Command used on line 2 of mixed-signal control file
# vcsAD.init followed by instances partitioned by that
# command
#=====================================================
use_spice -cell addr4;

# top.dut
```

Summary

This tutorial provides a basic overview of mixed-signal simulation and uses an example design and scripts to perform all-digital and mixed-signal simulations on a Verilog Top design:

- All-digital VCS-only simulation
- Mixed-signal VCS AMS two-step flow
- Mixed-signal VCS AMS three-step flow

In addition, the tutorial describes using the Custom WaveView waveform viewer to view the simulation results and lists the available report files generated during simulation.

Refer to the *Mixed-Signal Simulation User Guide* and *CustomSim User Guide* for more information about mixed-signal and CustomSim simulation flows and commands.