

Automating Analog Verification in a Mixed-Mode Simulation

Michael Laramie

asicNorth

MikeLaramie@asicnorth.com

ABSTRACT

This paper discusses a mixed-signal verification methodology that uses VCS, NanoSim, and a custom communications backplane to verify a mixed-signal device. In a standard mixed-mode NanoSim-VCS simulation, there are no built-in functions for the VCS simulator to measure the analog voltages and currents at the same resolution as the NanoSim simulator. Without this capability, automating the verification of analog signals is difficult and inefficient. The custom communications backplane between VCS and NanoSim allows the analog voltages and currents to be measured and verified using functions and tasks in a Verilog testbench.

Table of Contents

| | |
|--|-----------|
| 1.0 Introduction | 4 |
| 2.0 Simulators and their API extensions | 4 |
| 3.0 Limitations in NanoSim-VCS simulations | 6 |
| 3.1 Cross-module references | 6 |
| 3.2 Graphical output | 10 |
| 3.3 HSPICE .measure statements | 10 |
| 4.0 Previous solution and limitations | 10 |
| 5.0 Custom communications backplane | 11 |
| 5.1 Instantiating the backplane in a design | 15 |
| 5.2 Backplane initialization | 18 |
| 5.3 Simulation flow | 18 |
| 6.0 Example backplane usage | 20 |
| 6.1 DAC | 20 |
| 6.2 Automated mixed-signal verification | 20 |
| 7.0 Conclusions and Recommendations | 23 |
| 8.0 References | 23 |

Table of Figures

| | |
|--|-----------|
| Figure 1 : Standard NanoSim-VCS simulation environment | 4 |
| Figure 2 : NanoSim-VCS simulation environment with backplane..... | 6 |
| Figure 3 : Old XMR access..... | 7 |
| Figure 4 : Enhanced XMR access..... | 8 |
| Figure 5 : Resolution of NanoSim, Verilog, and logic signals..... | 9 |
| Figure 6 : Backplane components | 12 |
| Figure 7 : Example verification task | 13 |
| Figure 8 : Backplane message flow..... | 14 |
| Figure 9 : Analog circuit with ADFMI probe circuits..... | 15 |
| Figure 10 : Verilog testbench with PLI measure functions..... | 16 |
| Figure 11 : Example VCS command line options | 17 |
| Figure 12 : Example partition map | 17 |
| Figure 13 : Simulation flow..... | 19 |
| Figure 14 : 4-bit DAC architecture and partitioning..... | 20 |
| Figure 15 : HSPICE netlist..... | 21 |
| Figure 16 : Verilog testbench | 22 |

1.0 Introduction

The importance of automated mixed-signal verification has increased due to the increasing complexity of mixed-signal chips. Manually reviewing analog waveforms is a reasonable method for verifying the operation of a few integrated DACs, ADCs, and other analog components. However, the feasibility of manually reviewing analog waveforms quickly decreases as the number of programmable analog components increases on mixed-signal chips. A recent project required the verification of a mixed-signal chip with approximately two hundred 8-bit registers that controlled various analog functions including voltage DACs, current DACs, variable resistors, variable capacitors, and more.

The cost/speed ratio of microprocessors continues to decrease so the existing mixed-signal verification tools need to be enhanced to utilize this continually increasing resource. By enabling automated analog verification tests, the time consuming and error prone manual verification steps can be replaced by faster and more accurate automated tests. In other words, expensive engineering time can be used for the more challenging tasks while inexpensive CPUs handle the more mundane aspects of the verification process. The ability to measure analog voltages and currents in a Verilog testbench also brings the verification methodology closer to the ideal mixed-signal verification environment where the analog/digital boundary appears completely seamless.

2.0 Simulators and their API extensions

Mixed-signal chips are partitioned into digital and analog sections for mixed-mode simulations. A logic simulator such as VCS simulates the digital sections, while a circuit simulator such as NanoSim simulates the analog sections. A co-simulation environment handles the communication between the digital and analog simulations. The NanoSim-VCS co-simulation environment (shown in figure 1) is actually a direct kernel-to-kernel interface between VCS and NanoSim that is optimized for performance.

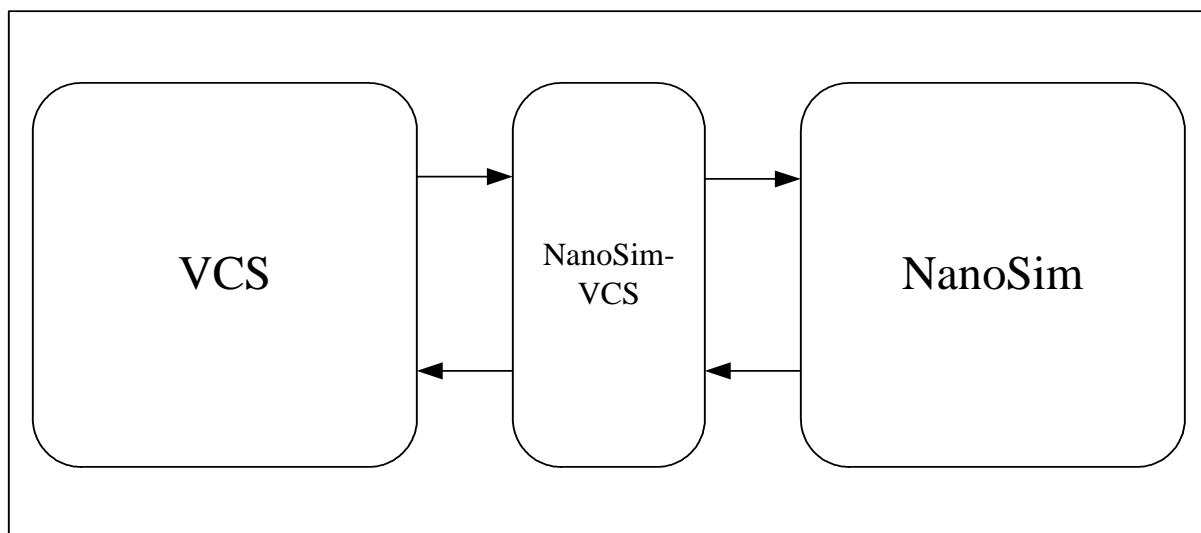


Figure 1: Standard NanoSim-VCS simulation environment

The top-level testbench that generates the digital stimuli and checks the digital outputs for the chip is typically written in Verilog. The NanoSim-VCS co-simulation environment does support designs with a top-level SPICE netlist and mixtures of Verilog and SPICE netlists that form doughnut hierarchies. However, the SPICE “language” does not have the verification capabilities of Verilog so the top-level testbench in a mixed-signal verification environment is usually written in Verilog. The well known Verilog Programming Language Interface (PLI) can also be used to extend the testbench functionality to handle very complex digital pattern generation and automated verification.

The NanoSim simulator has a very flexible Tcl interface for writing custom procedures to enhance its simulation capabilities. However, VCS becomes the primary simulator in a mixed-mode simulation so accessing NanoSim’s interactive Tcl interface is impossible or at least very difficult. NanoSim also supports Verilog-A behavioral models, but the Verilog-A language does not include a programming interface for the C language. Fortunately, there is another less prominent programming interface for the NanoSim simulator. The Analog Digital Functional Modeling Interface (ADFMI) is a proprietary modeling extension developed by Synopsys and used primarily for adding event-driven and functional models to NanoSim.

The NanoSim-VCS co-simulation environment does not have any unique APIs, but both PLI and ADFMI functions can be used in mixed-mode simulations. Since PLI and ADFMI functions can both be implemented in the C language, the natural choice is to implement a communications backplane between VCS and NanoSim in C as well.

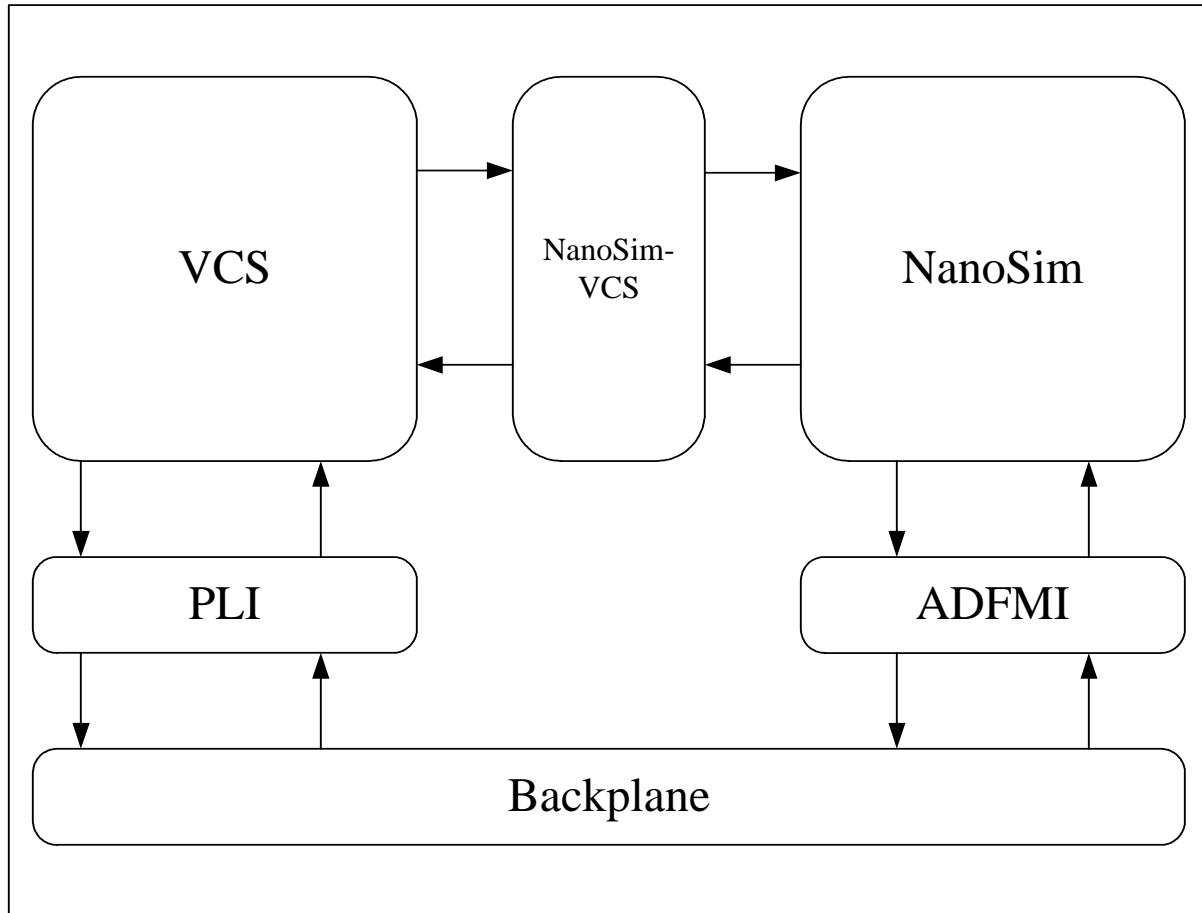


Figure 2: NanoSim-VCS simulation environment with backplane

As shown in figure 2, the backplane adds another communication path between the VCS and NanoSim simulators. The backplane uses standard UNIX Inter Process Communication (IPC) protocols to handle all the communication between the simulators. This means the backplane can be easily enhanced to do much more than just pass voltage and current values between the simulators.

3.0 Limitations in NanoSim-VCS simulations

3.1 Cross-module references

Hierarchical references also know as cross-module references (XMRs) are frequently used in Verilog simulations to measure and verify the logical state of any node in the design. Prior to release 2003.12, XMRs in NanoSim-VCS simulations were limited to the boundary of the analog cells partitioned to the NanoSim simulator. This meant that the VCS simulator could not access any of the internal analog signals, as shown in figure 3.

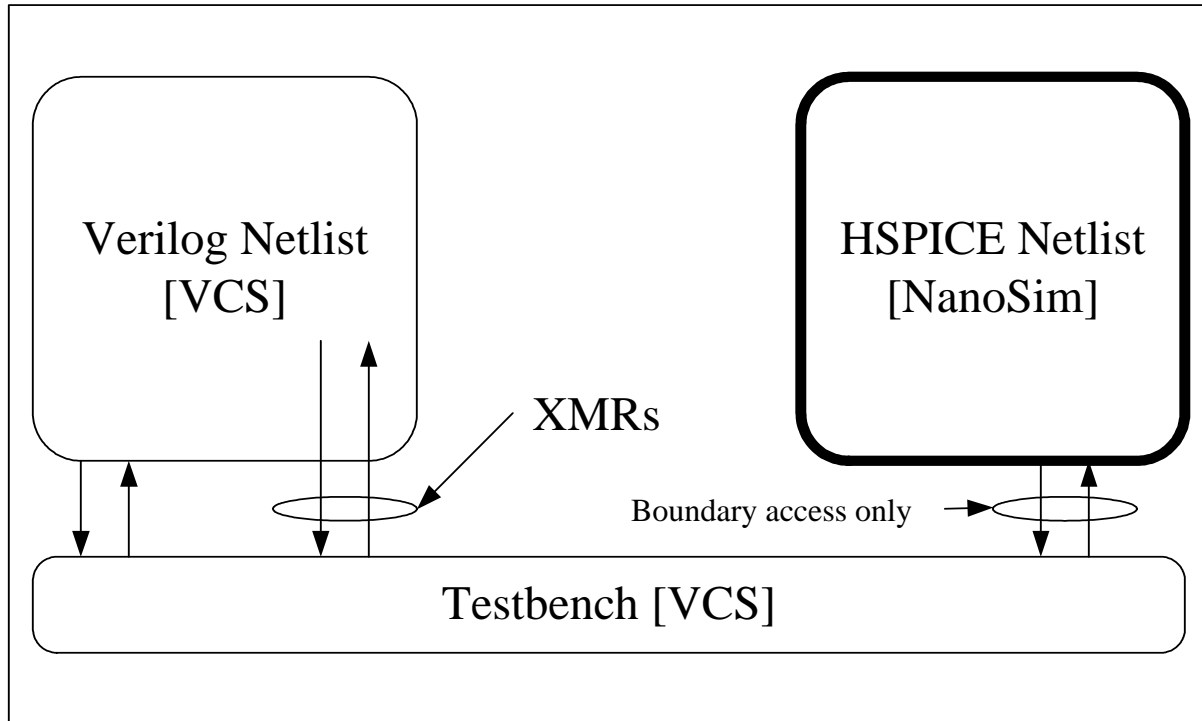


Figure 3: Old XMR access

The XMR capability has been enhanced in the current release (shown in figure 4), but XMRs are still limited to measuring only the logical state of a node. The resistor map used for the boundary nodes is not used for the internal nodes so only a logical 0 or 1 can be returned. Since the Verilog signal strength information is not returned for the internal nodes, the strength levels cannot be used to detect voltage levels within the logic 0 or 1 state defined by the voltage threshold.

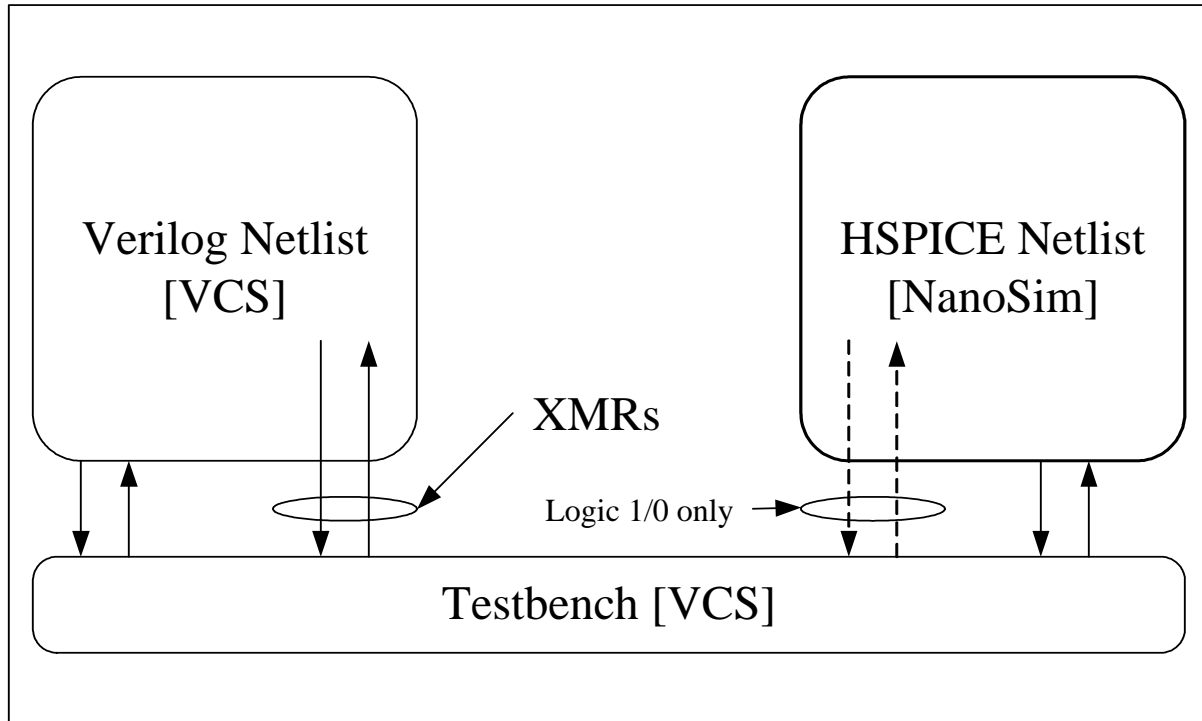


Figure 4: Enhanced XMR access

Figure 5 shows the resolution of an analog signal with a 1 mV resolution in NanoSim compared to a Verilog signal and a logic signal. The figure clearly shows that too much information is lost to automate analog verification tests when an analog signal is converted to a logical 0 or 1. Even though the voltage thresholds can be adjusted at the beginning of each simulation for each node, this still does not return enough information to easily automate the analog measurements. For example, 17 threshold values and therefore 17 different simulations would be required to verify each DC output voltage for a simple 4-bit DAC.

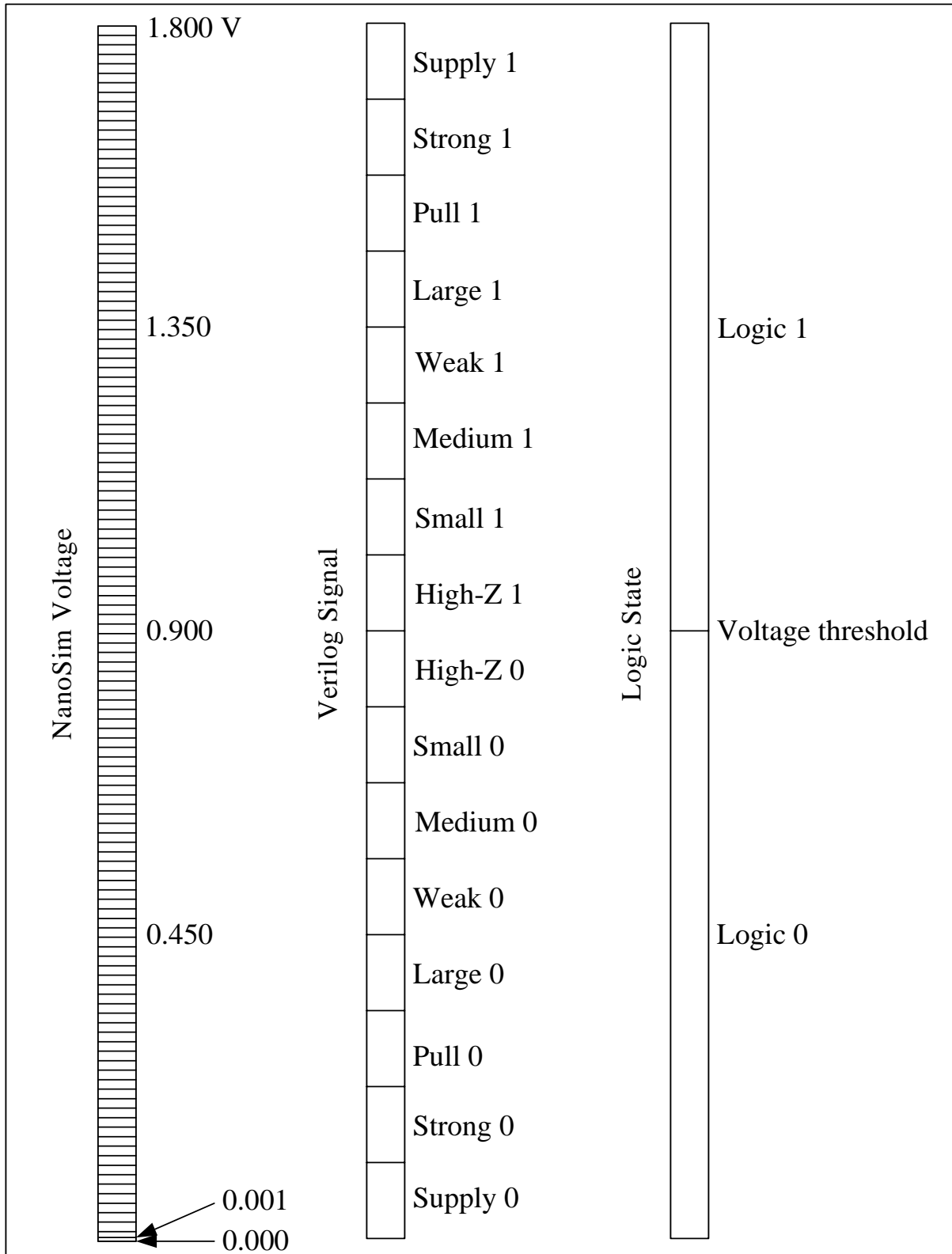


Figure 5: Resolution of NanoSim, Verilog, and logic signals

In general XMRs cannot be used to measure node currents. The voltage thresholds could be adjusted to indirectly detect the current if there is an ideal resistor load to a known voltage. This method would still require many threshold values, and the currents of interest in analog circuits are typically the transistor currents (i.e. mirrored currents and tail currents) not ideal resistor currents.

The new XMR capabilities can be very useful in mixed-mode simulations, but they are not sufficient for developing fully automated tests for the analog portions of a mixed-signal design.

3.2 Graphical output

NanoSim can be configured to save the waveform for any node voltage or current in the design. However, this data is not available to the Verilog testbench during the simulation so all the waveform data has to be post processed to verify the correct results. Manually viewing the waveforms is time consuming and error prone. Automated post processing is possible, but some sort of synchronization has to be maintained with the digital signals coming from the Verilog testbench otherwise minor changes in the testbench can cause all the post processing tests to fail.

The graphics files for analog signals can also become very large for even a relatively short digital simulation so it is very easy to exceed the maximum file size allowed by the operating system. The file size can be managed by using the NanoSim configuration commands to limit the scope of the signals to be printed or to save different time slices of the simulation in different output files. However, this becomes cumbersome to manage when there are approximately 200 testbenches in the verification test suite.

3.3 HSPICE .measure statements

HSPICE .measure statements can also be included in the HSPICE netlist to automate some of the verification tests. However, the .measure statements can become very complicated and some sort of synchronization still has to be maintained with the digital signals coming from the Verilog testbench. The results from the .measure statements are also not available to the Verilog testbench during the simulation so some sort of post processing would still have to be done.

4.0 Previous solution and limitations

In previous mixed-signal verification efforts, the verification tests were partitioned according to the design hierarchy and the capability of the simulation tools. For example due to simulation time constraints and simulation accuracy limits, it does not make sense to verify the full analog function of a DAC in the top-level chip simulations. However, the system-level functionality (i.e. signal connectivity, bus directions, and DC voltage levels) must be verified. Please see “Analog/Mixed-Signal Verification Methodology using NanoSim and VCS-ACE” (VCS-ACE is now known as NanoSim-VCS) from SNUG Boston 2002 for a full description of this methodology.

The previously implemented verification flow deviated from the ideal methodology by adding verification tests that used NanoSim’s Tcl interface to automate the higher-level analog tests.

The NanoSim/Tcl verification tests were fully automated so full regression tests could be run on a regular basis and for all the major design changes.

The NanoSim/Tcl verification environment works great for sequentially sequenced tests. For example program a group of registers, wait a fixed amount of time, and then verify the voltages and currents are within expected limits. However, the NanoSim/Tcl verification tests become cumbersome and inefficient for event-driven tests. For example, program a group of registers, apply the system clock, wait for a lock indicator or some other event, and then verify the analog signals.

The ‘always @ (event)’ Verilog construct is the ideal way to handle these event-driven tests. However, the NanoSim-VCS co-simulation environment does not have any built-in functions that would allow a Verilog testbench to measure analog voltages and currents. So there is no way to fully leverage the Verilog testbench environment for mixed-signal verification.

5.0 Custom communications backplane

A custom communications backplane was developed to address the limitations encountered during previous mixed-signal verification projects. When the backplane is instantiated in the design, VCS can measure analog voltages and currents at the same resolution as NanoSim. The backplane uses ADFMI functions to interface with NanoSim, PLI functions to interface with VCS, and a controller written in the C language to tie all the components together as shown in figure 6.

Three primary ADFMI functions and subcircuits (anBackplane, anProbeNodeVoltage, and anProbeNodeCurrent) were developed to enable communication with the NanoSim simulator. All public functions and subcircuits were prefixed with ‘an’ (asicNorth) to avoid name collisions. The anBackplane subcircuit starts the backplane controller at the beginning of the simulation, and the probe subcircuits report the voltages and currents during the simulation phase. The ADFMI subcircuits can be instantiated at any level of the analog circuit, but would typically be placed at the top level. Hierarchical SPICE references are used to connect the probe subcircuits to the voltages and currents in the analog circuit.

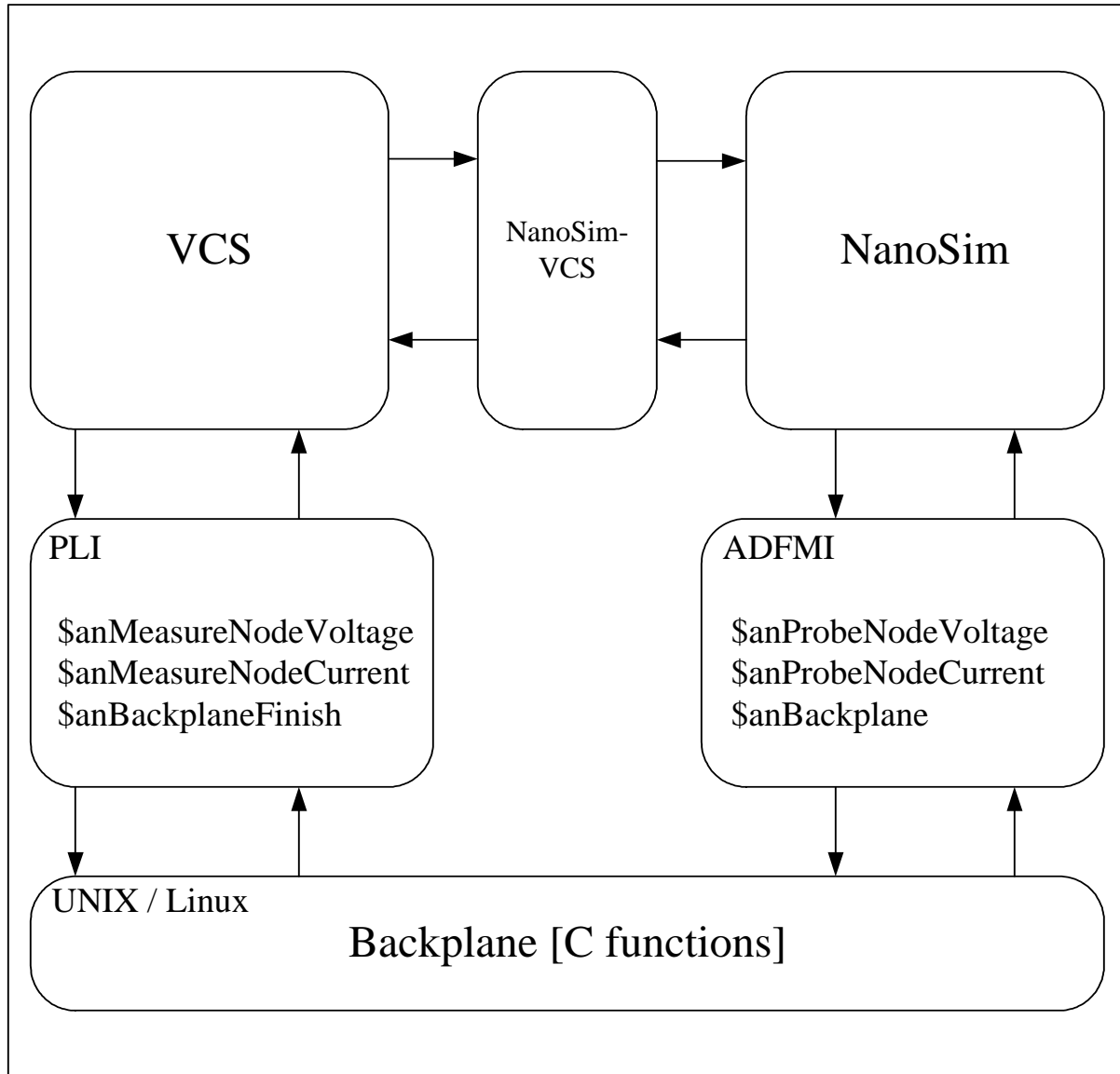


Figure 6: Backplane components

There were also three primary PLI functions (`anMeasureNodeVoltage`, `anMeasureNodeCurrent`, and `anBackplaneFinish`) developed to enable communication with the VCS simulator. The measure functions can be used anywhere in the Verilog testbench to obtain the present voltage or current for a node in the analog circuit, and the `anBackplaneFinish` function is used to safely shut down the backplane. The voltages and currents are returned as real register data types to the Verilog test bench so they can be easily verified or used in a more complicated equation to evaluate the analog circuit. Figure 7 shows a simple task that can be used to verify a voltage returned by the backplane is within a given tolerance of the target voltage.

```

task assertNodeVoltageTolerance;

    input nodeName, voltage, target, tolerance;

    reg [1024*8:1] nodeName;
    real voltage;
    real target;
    real tolerance, lo, hi;

    begin

        lo = target * ( 1 - tolerance );
        hi = target * ( 1 + tolerance );

        if ( voltage >= lo && voltage <= hi )
            begin
                $display( "Voltage OK at node %s = %f V within %f
                    <-> %f V", nodeName, voltage, lo, hi );
            end
        else
            begin
                $display( "Error: Voltage at node %s = %f V not
                    within %f <-> %f V", nodeName, voltage,
                    lo, hi );
            end
        end

    end

endtask

```

Figure 7: Example verification task

The backplane controller enables communication between NanoSim and VCS by parsing and acting on all the messages generated by the ADFMI and PLI functions. Figure 8 shows the message flow between the backplane and the simulators. The present backplane message protocol is very straightforward and allows node voltages and currents to be stored and retrieved.

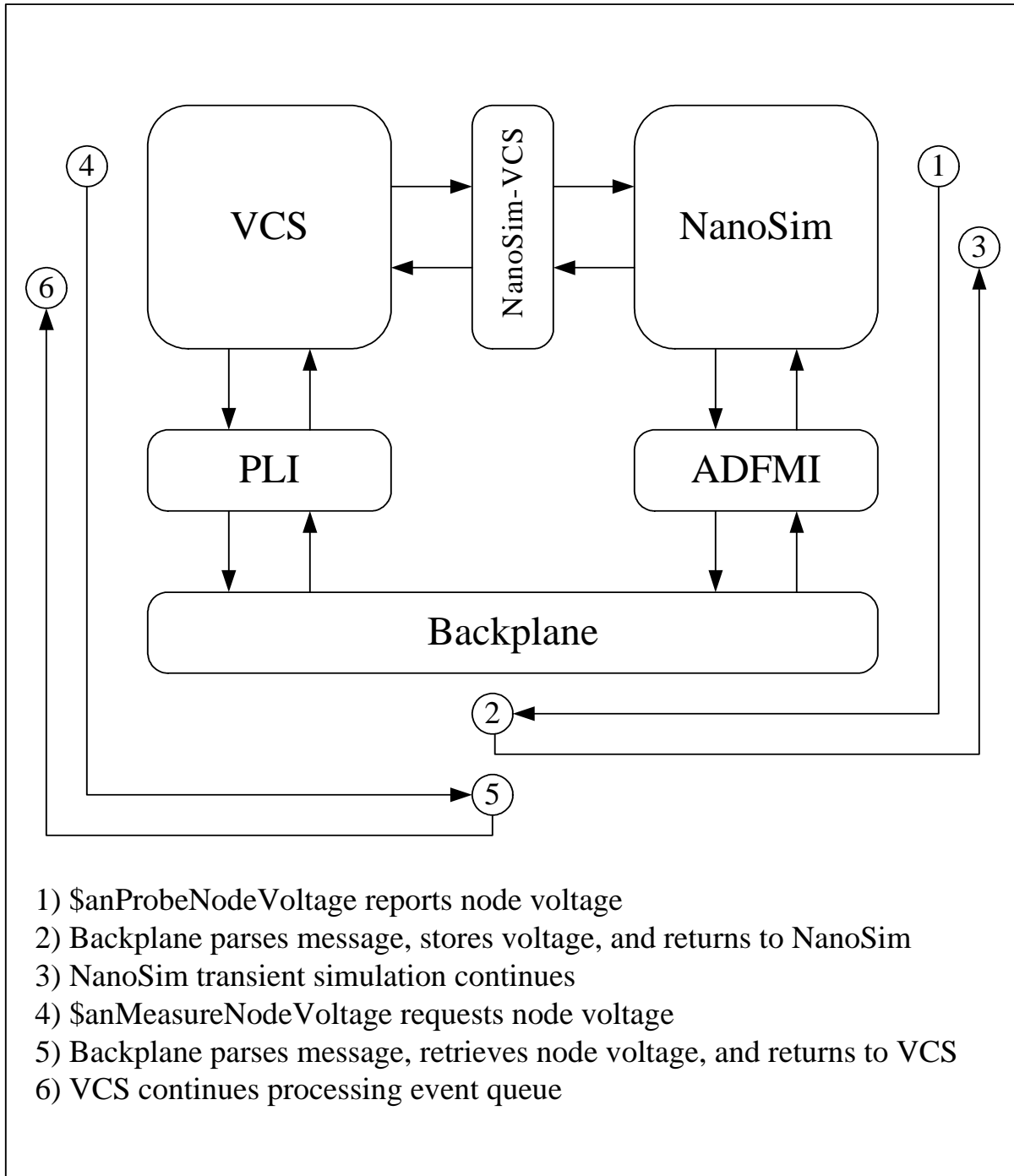


Figure 8: Backplane message flow

5.1 Instantiating the backplane in a design

The backplane can be used in a mixed-signal design by simply adding the probe subcircuits to the analog circuit, adding the measure functions to the Verilog testbench, and modifying the VCS and NanoSim command line options for compiling the simulation. Figure 9 shows a typical analog circuit with the subcircuits required to probe voltages and currents. Figure 10 has an example Verilog testbench that uses the measure function and the previous verification task to verify some of the analog voltages. Finally figures 11 and 12 show the command line options and partition file used to compile the Verilog testbench and include the ADFMI and PLI functions.

```
.SUBCKT ISEG D0N D0P D1N D1P D2N D2P D3N D3P VN VP
XISRC_1 VN VP D0N D0P AVDD ISRC
XISRC_2 VN VP D1N D1P AVDD ISRC
XISRC_3 VN VP D2N D2P AVDD ISRC
XISRC_4 VN VP D3N D3P AVDD ISRC

RResistor_1 VN AGND 50
RResistor_2 VP AGND 50

VV_dc_1 AVDD AGND DC=1.8
VV_dc_2 AGND GND DC=0

X0 AGND anBackplane
X1 VP anProbeNodeVoltage
X2 VN anProbeNodeVoltage
X3 AVDD anProbeNodeVoltage
X4 XISRC_1.N5 anProbeNodeVoltage

.ENDS
```

Figure 9: Analog circuit with ADFMI probe circuits

```
initial begin

    Data = 2'b00;

    #5;
    voltage = $anMeasureNodeVoltage( "tb.u0.iseg.v1" );
    assertNodeVoltageTolerance( "tb.u0.iseg.v1", voltage,
                                0.5, 0.1 );

    #5;
    voltage = $anMeasureNodeVoltage( "tb.vp" );
    $display( "VP = %f V", voltage );

    voltage = $anMeasureNodeVoltage( "tb.vn" );
    $display( "VN = %f V", voltage );

    voltage = $anMeasureNodeVoltage( "avdd" );
    assertNodeVoltageTolerance( "avdd", voltage, 1.8, 0.1 );

    Data = 2'b01;

    #5;
    voltage = $anMeasureNodeVoltage( "tb.u0.iseg.v1" );
    assertNodeVoltageTolerance( "tb.u0.iseg.v1", voltage,
                                1.0, 0.1 );

    $anBackplaneFinish();

    $finish;

end
```

Figure 10: Verilog testbench with PLI measure functions

```

# Compile VCS/NanoSim test bench
vcs ${vDir}/tb.v \
    ${libDir}/verilog/nanosim/ISEG.v \
    +incdir+${vDir}+ \
    +incdir+${libDir}/verilog/an_lib+ \
    +libext+.v+ \
    -y ${libDir}/verilog/an_lib \
    -y ${vDir} \
    -V -PP +vcsd \
    +vpi +acc \
    -P ${pliDir}/anBackplane.tab \
    ${pliDir}/anBackplanePLI.c \
    ${pliDir}/anMeasureNodeCurrent.c \
    ${pliDir}/anMeasureNodeVoltage.c \
    ${pliDir}/anAssertNodeVoltageTolerance.c \
    ${pliDir}/anBackplaneFinish.c \
    +ad=partition.map

```

Figure 11: Example VCS command line options

```

choose nanosim
  -nspice
    /backplane/library/configuration/tsmcModels_typ.inc
    /backplane/library/configuration/nsSetup.sp
    /backplane/library/spice/nanosim/ISEG.sp
  -FM
    /backplane/ADFMI/anBackplaneADFMI.c
  -C
    /backplane/library/configuration/DAC4.cfg;

partition -cell ISEG;

```

Figure 12: Example partition map

5.2 Backplane initialization

At the beginning of the mixed-mode simulation, VCS initializes every node to “X” for registers and “Z” for wires unless there is an initialization value or function in the Verilog files. NanoSim initializes all the nodes at the analog/digital interface to “0” unless an initial value is passed in from the VCS initialization process, and then NanoSim starts its DC initialization process. NanoSim allows the ADFMI subcircuits to have different behaviors for the DC initialization, kickoff, and transient simulation phases of the circuit simulation. The anBackplane subcircuit starts the backplane during the DC initialization phase, and has no function during the kickoff and simulation phases. The probe subcircuits do some internal initialization during the DC initialization phase and then register their corresponding circuit nodes and voltage/current values with the backplane during the kickoff phase.

5.3 Simulation flow

After the backplane and analog circuits have been initialized, the simulation proceeds using the standard Verilog event queue and the NanoSim transient simulation phase. A highly simplified simulation flow would go something like this:

- VCS determines the values for all the digital outputs / analog inputs at a given time step. Values for the digital inputs / analog outputs feeding the digital circuits come from DC initialization or the most recent NanoSim results.
- NanoSim uses the updated analog inputs / digital outputs for the next transient simulation step, and updates the values for the analog outputs / digital inputs.
- Repeat until the end of the simulation (\$finish or simulation time limit in .tran statement).

The ADFMI probe subcircuits will report any voltage or current changes to the backplane during the NanoSim transient simulation phase. When processing returns to the VCS simulator, the PLI measurement functions can retrieve the voltage and current values stored by the backplane.

Figure 13 shows the backplane operation in this simplified simulation flow.

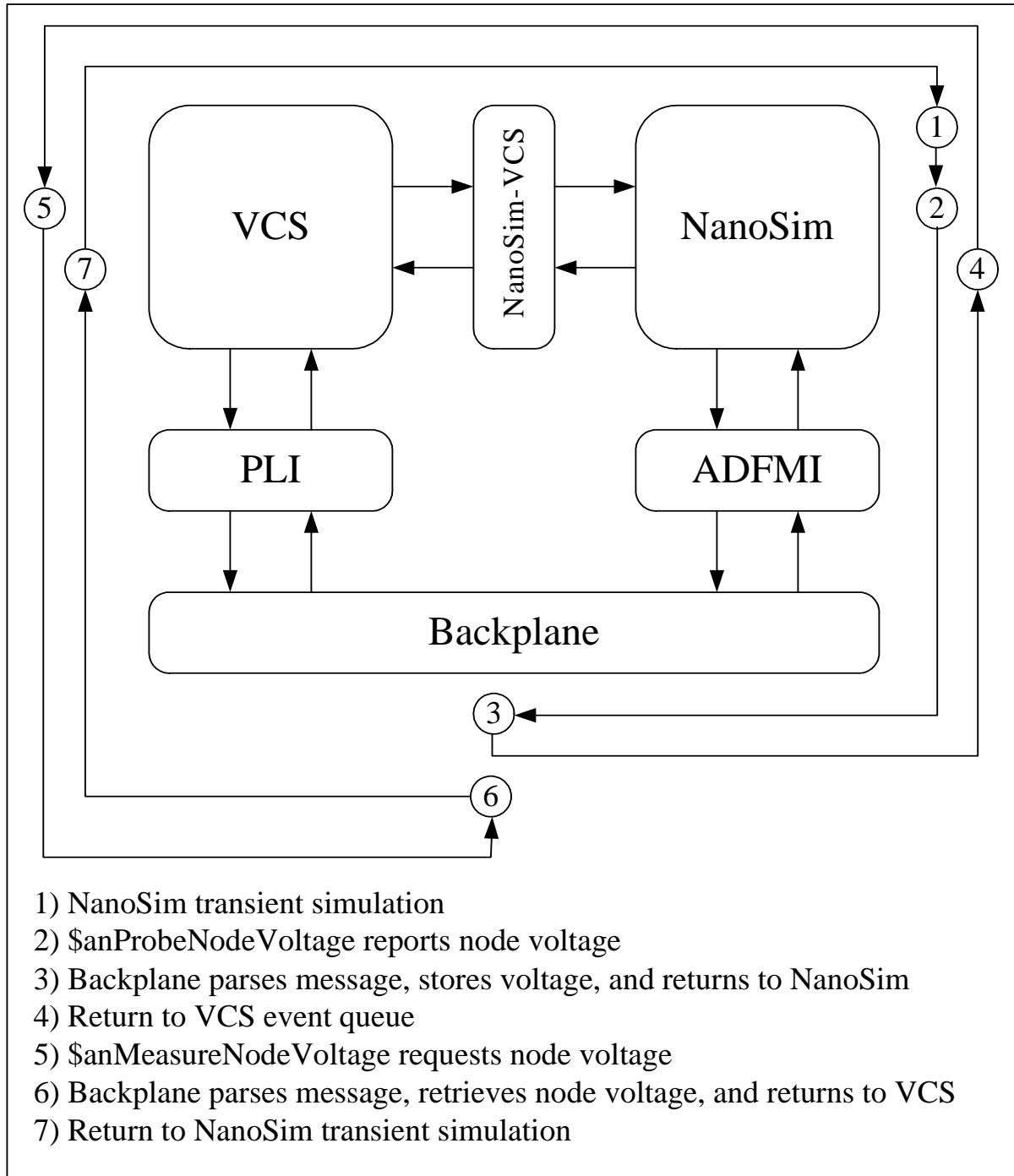


Figure 13: Simulation flow

6.0 Example backplane usage

This section shows the top-level components of a simple mixed-signal verification using the backplane for an ideal 4-bit DAC.

6.1 DAC

The ideal 4-bit DAC consists of a 4-bit to 16-bit thermometer decoder that turns on the correct number of segmented current sources to drive the DAC output. Each segmented current source is an ideal current source and a pair of current steering switches. Figure 14 shows how the 4-bit DAC is partitioned into a digital and analog section for the mixed-mode simulation. VCS simulates the digital domain consisting of the decoder and the top-level testbench while NanoSim simulates the segmented current source array.

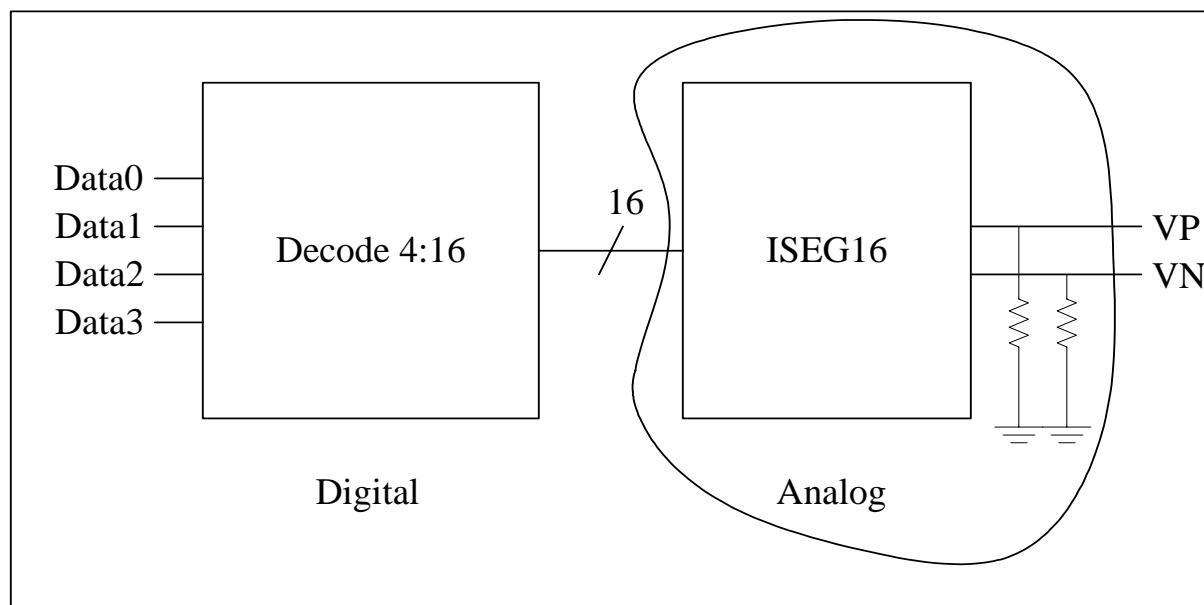


Figure 14: 4-bit DAC architecture and partitioning

6.2 Automated mixed-signal verification

Figure 15 shows the top-level of the HSPICE netlist for the 4-bit DAC with the ADFMI probe subcircuits used to send the output voltages to VCS. The fully automated Verilog testbench is shown in figure 16.

```

.SUBCKT ISEG D0N D0P D1N D1P D2N D2P D3N D3P D4N D4P
+ D5N D5P D6N D6P D7N D7P D8N D8P D9N D9P D10N D10P
+ D11N D11P D12N D13P D14N D14P D15N D15P VN VP

XISRC_0 VN VP D0N D0P AVDD ISRC
XISRC_1 VN VP D1N D1P AVDD ISRC
XISRC_2 VN VP D2N D2P AVDD ISRC
XISRC_3 VN VP D3N D3P AVDD ISRC
XISRC_4 VN VP D4N D4P AVDD ISRC
XISRC_5 VN VP D5N D5P AVDD ISRC
XISRC_6 VN VP D6N D6P AVDD ISRC
XISRC_7 VN VP D7N D7P AVDD ISRC
XISRC_8 VN VP D8N D8P AVDD ISRC
XISRC_9 VN VP D9N D9P AVDD ISRC
XISRC_10 VN VP D10N D10P AVDD ISRC
XISRC_11 VN VP D11N D11P AVDD ISRC
XISRC_12 VN VP D12N D12P AVDD ISRC
XISRC_13 VN VP D13N D13P AVDD ISRC
XISRC_14 VN VP D14N D14P AVDD ISRC
XISRC_15 VN VP D15N D15P AVDD ISRC

RResistor_1 VN AGND 50
RResistor_2 VP AGND 50

VV_dc_1 AVDD AGND DC=1.8
VV_dc_2 AGND GND DC=0

X0 AGND anBackplane
X1 VP anProbeNodeVoltage
X2 VN anProbeNodeVoltage
X3 AVDD anProbeNodeVoltage
X4 XISRC_1.N5 anProbeNodeVoltage

.ENDS

.SUBCKT ISRC IOUTN IOUTP N P AVDD

II_dc_1 AVDD N5 2.4mA
Mpch_1 IOUTN N N5 AVDD pch L=0.18u W=20u M=10
Mpch_2 IOUTP P N5 AVDD pch L=0.18u W=20u M=10

.ENDS

```

Figure 15: HSPICE netlist

```

`timescale 1ns/10ps
module tb;

    real voltage;
    real lsbVoltage = 0.120;
    real idealVoltage;

    reg [3:0] data;

    integer dataVal;

    wire VP;
    wire VN;
    wire [3:0] D = data;

    DAC4 u0 (
        .D0 (D[0]),
        .D1 (D[1]),
        .D2 (D[2]),
        .D3 (D[3]),
        .VP (VP),
        .VN (VN)
    );

    initial begin

        for( dataVal = 0; dataVal < 16; dataVal = dataVal + 1 )

            data = dataVal[3:0];
            idealVoltage = dataVal * lsbVoltage;
            #5;
            voltage = $anMeasureNodeVoltage( "tb.u0.iseg.vp" );
            assertNodeVoltageTolerance( "tb.u0.iseg.vp", voltage,
                idealVoltage, 0.1 );

        end

        $anBackplaneFinish();

        $finish;

    end

endmodule

```

Figure 16: Verilog testbench

7.0 Conclusions and Recommendations

This paper shows that a custom communications backplane between VCS and NanoSim can be developed with only a few ADFMI and PLI functions. Verification engineers can use the backplane to write Verilog testbenches that measure analog voltages and currents and fully automate the analog tests in a mixed-mode simulation. This can eliminate the time consuming and error prone process of manually reviewing analog waveforms as part of the verification process. The backplane also takes the methodology one step closer to the ideal mixed-signal verification environment where the analog/digital boundary appears completely seamless.

The next step is to evaluate the performance impact of the backplane on the mixed-mode simulations. Even if the performance is significantly degraded, the backplane still adds new capabilities that are needed by the mixed-signal verification engineers. So there will be a place for the backplane in the verification toolbox. Performance improvements will come from more efficient backplane code, as well as faster and cheaper CPUs. Synopsys is also currently developing the capability to measure voltages in NanoSim from a Verilog testbench. Performance should not be an issue once this capability is included in the generally released kernel-to-kernel interface.

8.0 References

Synopsys, NanoSim-VCS Manual -- Version V-2003.12, Synopsys, December 2003.

Synopsys, NanoSim Modeling Guide for ADFMI and Verilog-A -- Version V-2003.12, Synopsys, December 2003.

Stuart Sutherland, The Verilog PLI Handbook -- Second Edition, Kluwer Academic Publishers, 2002.

Marc Hutner, Analog/Mixed-Signal Verification Methodology using NanoSim and VCS-ACE -- SNUG Boston 2002, Synopsys, September 2002.

Synopsys, Techniques in Verification of Mixed-Signal and SoC Designs using NanoSim, Synopsys, January 2003.