Design of a Switch-Level Analog Model for Verilog

Thomas J. Sheffler
Rambus Inc.
Los Altos, CA
tom.sheffler@sbcglobal.net

ABSTRACT

This paper describes a modeling extension to Verilog called "Switch-Level Analog." It is inspired by the switch-level transistor modeling facility of Verilog, but extends the value domain from Logic to Reals and is based on linear relationships between the currents of branches and the voltages of nodes, rather than the charge relationships of nodes of earlier switch-level models. This capability allows the modeling of many types of modern circuit blocks that exploit the current-source (saturation mode) and resistive (linear mode) properties of transistors. The model is implemented as a PLI library. Modeling examples and performance data are presented.

1 INTRODUCTION

Verilog is the workhorse language for the modeling of digital designs. In recent years, more and more "mostly digital" designs include some amount of non-digital circuitry. Unfortunately, the ability of Verilog to model non-digital phenomena is severely limited by its inability to transmit non-digital values over Verilog wires.

This paper describes a PLI (Programming Language Interface) library that adds an event-driven analog modeling facility to Verilog. With the library, Verilog wires can name nodes in an electrical network. Electrical network primitives are provided allowing the modeling of register-controlled variable sources, resistors, dependent sources and switches. This repertoire is sufficient to model many types of analog architectures that occur in real chip designs. With such a modeling capability, the verification of the chip-level digital and "coarse" analog architecture can proceed before transistor-level implementations are available.

The model presented here is event-driven: it responds to changes in Verilog real-valued registers and delivers value-change events back through the Verilog scheduler. The model is derived from the existing switch-level model of Verilog, which is explicitly based on the notion of connected-component subnetworks [2,10]. Value changes on the inputs to a subnetwork induce the evaluation of the steady-state of a subnetwork.

The subnetwork model extends to unknown values such that if any inputs are unknown the value of an entire subnetwork is simply unknown. This formulation gives a meaningful basis for the analysis of "X"-values in some types of analog systems.

1.1 A PLI Library for Linear Systems

We have extended the techniques of [3,11] and implemented a PLI library for modeling networks of variable linear circuit elements. In previous work [6,7] we reported on verification issues involving mixed-signal integration, and showed how to use the library as a modeling tool. In this paper, we focus on its implementation and performance.

Our implementation uses the VPI programming interface [9]. The first level of the library defines system functions for modeling circuit devices including resistors, current sources, voltage sources and switches. Verilog registers, whose values may be updated dynamically, assign time-varying values to these elements.

In this model, an electrical node is named by a Verilog wire. A wire describes one endpoint of a multi-terminal net in Verilog. Using PLI functions to trace connectivity, we define all of the aliases of the multi-terminal net to refer to the same electrical node. In this manner, wiring hierarchy in the Verilog database connects electrical nodes in the expected way. This is the same technique used by [3].

A variable resistor is instantiated with the executable call to "$resistor." An example appears in Figure 1. Wires w1 and w2 identify two electrical nodes, and the call to "$resistor" creates an electrical circuit element across them. The value of the resistor is dynamically controlled by "rval," which is called the "control register" of the resistor. In the example, the initial value of "rval" is set before instantiating the resistor. Subsequent assignments to the value of "rval" during simulation change the value of the resistor, which changes the steady-state of its electrical network.

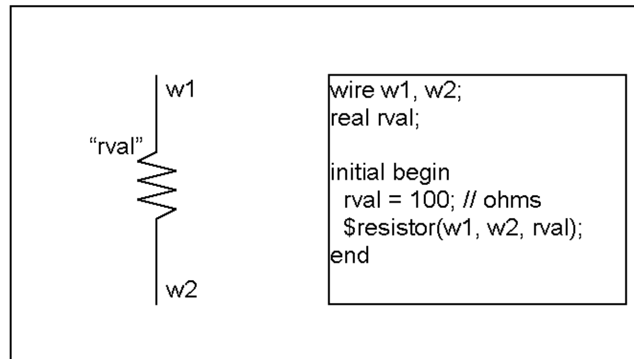

Figure 1. Resistor Instantiation

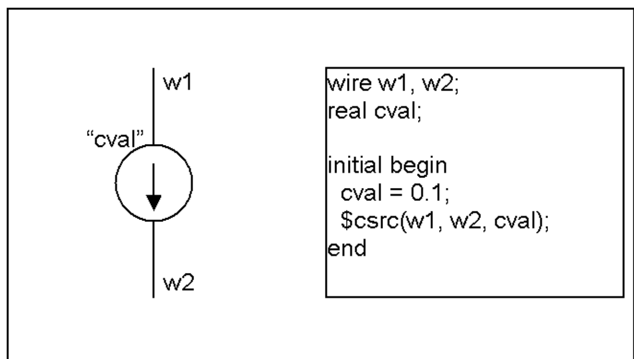

Figure 2. Current Source Instantiation

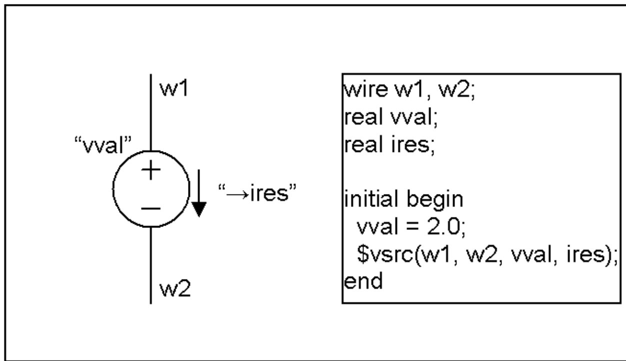

Figure 3. Voltage Source Instantiation

A variable current source is instantiated as shown in Figure 2. In this example, the Verilog register "cval" is the control register of the current source. The amount of current flowing through the current source may be dynamically changed by re-assigning the value of register "cval."

A variable voltage source may likewise be instantiated as shown in Figure 3. A voltage source has one control register and also has a "result register" which is used to report the amount of current flowing through the voltage source in the steady-state solution of the subnetwork. In the example, the control register is "vval" and the result register is "ires."

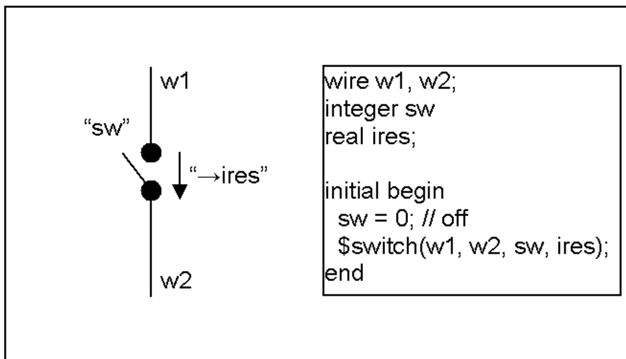

Figure 4. Switch Instantiation

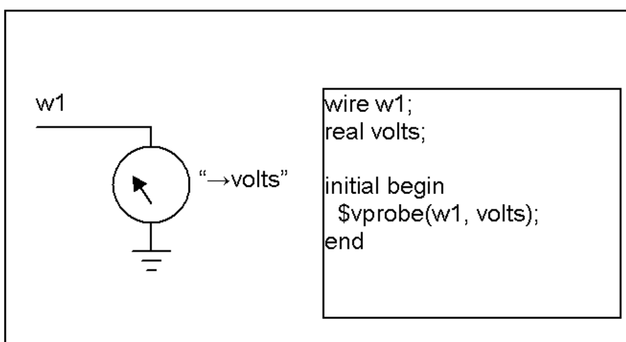

Figure 5. Voltage Probe Instantiation

An ideal electrical switch is instantiated as illustrated in Figure 4. The control register of a switch is a Logic value, and selects whether the switch is ON (1) or OFF (0). An ideal switch has zero resistance (a short circuit) when ON and zero admittance (an open circuit) when OFF [1].

A primitive is also provided for observing a voltage on an electrical node and reporting it in a Verilog register. An example of the use of a voltage probe is shown in Figure 9. In this fragment, "volts" is a result register. Whenever the steady-state operating point of the voltage on the electrical node named by "w1" changes due to a change in the value of an electrical element in its subnetwork, the value of "volts" will be changed to reflect the new value.

2 Implementation Notes

A complete description of the implementation of the library is not possible in this space. Rather, we highlight some of the important points one would need to know to implement the library presented here.

2.1 An Overview of Data-Structure Creation

The Verilog PLI provides hooks for registering callbacks at various times in the parsing of the netlist and the running of the simulator. Each device instantiation corresponds to a so-called user-defined "system function." The callback points we used were "compile", "start_of_sim " and "calltf."

The compile callback is called for each system function during parsing of the netlist. Because the compile phase of Verilog execution may occur in a process separate from the simulation run-time, one cannot reliably allocate memory (or otherwise build a data-structure) during this phase. Thus, during this phase we check the number of arguments for each device and confirm that they are the correct type. We are, however, allowed to register selected callbacks for each system function, and it is here that we register the "start_of_sim" callback for each device.

The "start_of_sim" callback is called at simulation time 0, and PLI code is allowed full use of the memory allocation system. For each circuit device, we build a device data structure, and record references to the Verilog objects describing the wires and registers associated with each device. It is also during this phase that we resolve wire name aliases to electrical nodes (see the next section).

The "calltf" callback corresponds to the invocation of the system function corresponding to the instantiation of the circuit device in an "initial" block. It is here that we chose to perform global operations that need to occur over the collection of all devices. The following steps are performed here:

- Partitioning:

  The electrical network (now described in terms of devices and nodes) is partitioned into subnetworks. A connected components algorithm is run. The rules defining the partitioning are the following:

  - An electrical element can be in only one subnetwork.
  - Two electrical elements are in the same subnetwork if they share a node.

- Build Matrices:

  The devices provided include common linear devices and a special device: the switch. The structure of the matrix is determined here, and memory is allocated. Initial values from the control registers provide initial values for the stamp associated with each device.

- Schedule control register callbacks:

  For each control register, a value-change callback is registered with the Verilog simulator. A control register value-change results

in the subnetwork being marked for re-evaluation (system solution) at the end of the current time step. After solution, the result registers associated with each subnetwork are updated. This event causes value changes to propagate through the system to logic, and other electrical subnetworks.

2.2 Using Verilog Wires to Name Circuit Nodes

We adopt the technique of [3] to use Verilog wires to name electrical nodes. In our system, a reference to a wire, w1, is translated into a list of the string-valued Verilog path names of all of its aliases in the netlist. (Finding these aliases is performed once for each wire and requires a connected-components traversal of the nets and ports in the database.) Each of these names becomes an entry in a global hash table that maps path names to an electrical node object.

During instantiation of the electrical devices, the pathname of each wire is first looked up in the hash table. If the name is there, the corresponding node object is retrieved directly. If it is not, all of the aliases of that wire are added to the hash table and a node object is created. The process is performed at the beginning of simulation only. During simulation, the node objects are used directly, and the hash table is no longer needed.

2.3 Global Supplies, Implicit Sources, and GND

A facility is defined to identify global supplies and implicit sources. The datum node, "top.GND" is a special node defined in the database. During the partitioning process, device terminals attached to this node and its aliases are ignored. Thus, two subnetworks connected through GND remain separate.

A similar facility has been defined for global supplies and implicit sources. An implicit source, like "top.VDD" is a wire that is defined in the top level of the database. A configuration file gives its voltage value. Wires connected to implicit sources are ignored in the partitioning process, similar to GND. A reference to an implicit source in a subnetwork causes the automatic instantiation of a source device in that subnetwork.

2.4 Matrix Stamps

For each subnetwork, a matrix formulation of the DC steady-state of the subnetwork system is formulated at simulation time 0, using the initial values of the control registers. We chose to adopt the modified nodal analysis formulation described in [5]. The introduction of the stamps for each linear device (source, resistor, op-amp or dependent source) is performed as described there.

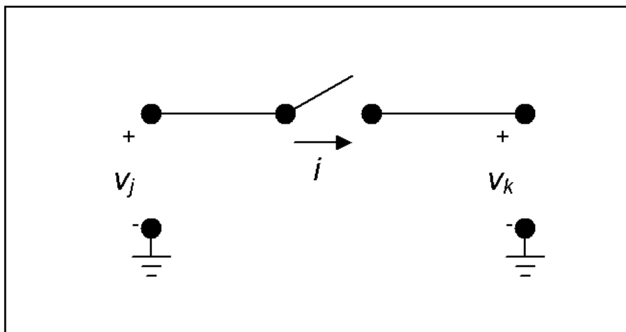

Figure 6. Switch Definition

The stamp for the switch, is novel, but has been reported elsewhere [4]. Figure 6 defines the quantities associated with a switch. A switch connected to nodes j and k, introduces one of two linear constraints into the system describing the steady-state value of the subnetwork. If the switch is open, the switch is a zero-valued current source, introducing the constraint $I_{jk} = 0$. If the switch is closed, the switch is a zero-valued voltage source, introducing the constraint $V_j - V_k = 0$.

For each switch S, we introduce a new independent variable X_i: the branch current through the switch. The row z, corresponding to variable X_i has the following form as shown in Figure 7, depending on the value of s.

	v_j	v_k	i	rhs
j			1	
k			-1	
z	1	-1		

Stamp for Closed switch

	v_j	v_k	i	rhs
j			1	
k			-1	
z			1	

Stamp for Open Switch

Figure 7. Switch Matrix Stamps

Thus, for the open switch, the stamp describes the constraint that the branch-current must be zero, and that the switch introduces no constraint between the node voltages of the switch. For a closed switch, it describes the constraint that the node voltages of the switch must sum to zero, but that the switch introduces no constraint on the branch current through the switch.

In our implementation, the system is re-solved each time a control register changes by performing an entire LU factorization and solve. In [4], the authors describe an iterative approach for handling switch transitions. We have not experimented with that formulation.

2.5 Efficient Update of Matrix Stamps

The update of the steady-state values of a subnetwork is initiated by a value change on a control register. To ensure that only the stamp values that need to be changed are actually changed, a map from control registers to stamp elements is maintained. We handle the updates for linear elements and switches differently.

For the linear elements, each entry in the matrix corresponding to the element's stamp is potentially a linear combination of contributions from other elements. Thus, it is possible to adjust the stamp by the change in magnitude of the control register and maintain the correct linear combination of contributions of related elements without also visiting their control registers.

A change in a switch value requires only modifying the three elements in its row to change its constraint from a zero-valued current source to a zero-valued voltage source.

2.6 The Simulation Algorithm

A sketch of the event-driven simulation model is described as follows. At simulation time zero, the steady-state of each subnetwork is computed using the initial values of the control registers. When a real-valued control register posts a value-change event, its attached subnetwork is scheduled for re-evaluation at the current time step. Re-evaluation consists of adjusting the stamps

related to changed values and recomputing the steady state. The result registers corresponding to watched branch currents and node voltages are then posted with new values.

Unknown (X) values are handled in the following way. If any inputs to a subnetwork are unknown, then the entire subnetwork is marked as unknown, and unknown values are propagated to all of its outputs. This is similar to the way in which X values propagate in Boolean switch-level networks in Verilog.

3 MODELING

The SLA (Switch-Level Analog) library adds two key capabilities to Verilog useful for the modeling of mixed analog/digital systems. They are:

1. The ability to send real values along wires,
2. A resolution function for these wires based on the solution of KCL for linear networks having flows and potentials.

One way to use the library is to model the Thevenin or Norton equivalents of drivers and loads on the periphery of modules, and to do the bulk of behavioral modeling using Verilog behavioral code. This has performance advantages, as most of the design is modeled in plain old Verilog and only subnetworks connecting modules are modeled using linear constructs. (Voltage sources, current sources and resistors are necessary to model Thevenin and Norton equivalents. Switches help model circuits whose logical topology changes as a result of digital control.) Figure 8 illustrates the general idea.

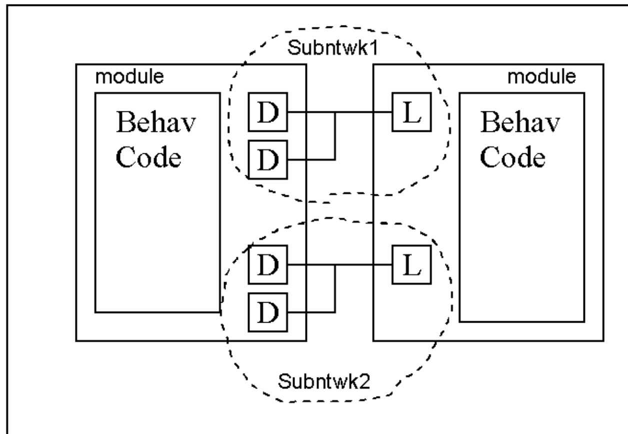

Figure 8. Modeling of Drivers and Loads

A module with digitally controlled analog drivers is modeled using SLA components for its output drivers, here labeled with a “D.” Another module observes these drivers with its loads, here labeled with an “L.” The hierarchy of the design wires together the drivers and loads as shown. Each connected component of SLA devices will form its own subnetwork. Each subnetwork will be evaluated when the control values of the drivers change. Small subnetworks are encouraged as they can be solved quickly.

Because each subnetwork is separate, the evaluation schedule of each may proceed on its own clock. For us, this is an important performance consideration, because each driver normally operates on its own clock phase. Without partitioning, a change in any control value would cause the re-evaluation of all analog values in the entire design.

3.1 Modeling a Differential Current-Steering Output Driver with Programmable Swing

Equalization (or pre-emphasis) is a technique used in an output driver to compensate for data losses and reflections on a noisy channel [12]. Such reflections cause interference between adjacent bit-times, which is called inter-symbol interference (ISI). In the frequency domain, equalization reduces the low-frequency components of the transmitted signal as appropriate for the transmission medium.

A Differential Current-Steering Output Driver with Equalization (DCSODwE) modulates the magnitudes of the bits transmitted on the channel to manage the frequency components of the signal. Rather than transmit one of only two voltage levels, representing a logical "0" and a logical "1", the DCSODwE implements a recurrence equation, where the voltage value produced at each bit time is a weighted sum of the data bits *before* (and possibly *after*) the current bit, x_i. (Bit x_i is called the cursor.) Equation 1 defines such a recurrence.

(Eq 1) $V_i = a_0\, x_I + a_1\, x_{i-1} + a_2\, x_{i-2}$

The output driver with equalization builds on a simpler structure: a differential output driver with programmable swing. This section describes the simple driver. A following section uses the components of the simple driver to assemble the output driver with equalization.

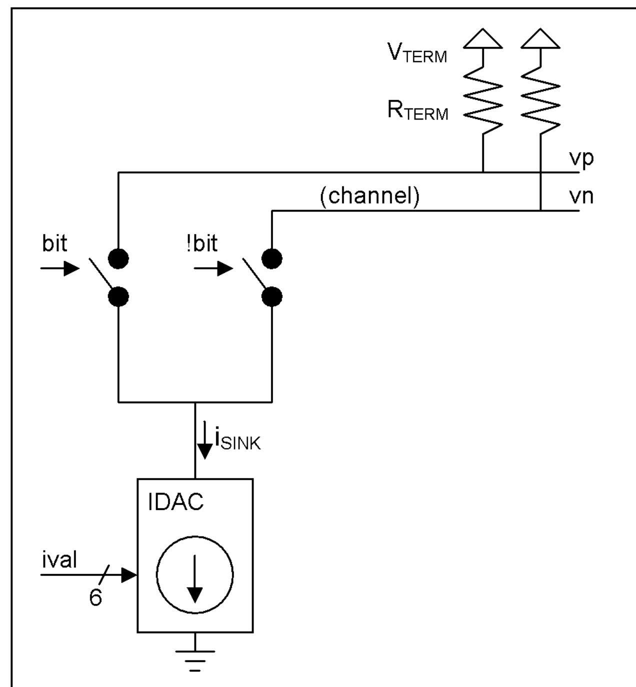

Figure 9. Differential Output Driver

The architecture of a non-equalizing differential current-steering output driver (DCSOD) (including its termination components) is shown in Figure 9. The current source produces a current called iSINK that passes through one of two switches. The bit stream (value "bit") and its complement (value "!bit") open and close the switches, which are implemented as appropriately biased transistors. The termination voltage, vTERM and the termination pullup resistors reside outside of the driver, on the far end of the channel. Because the current is always flowing through one of the legs, this architecture is called "current-steering."

When the driver is transmitting a logical "1", the switch controlled by "bit" is closed and vp is pulled down to

$V_{LOW} = V_{TERM} - (R_{TERM} * i_{SINK})$.

The switch controlled by the complement of "bit" remains open, and vn floats high to

$V_{HIGH} = V_{TERM}$.

Conversely, if the driver is transmitting a logical "0", then vn is pulled down and vp floats high. (In use, the signals vp and vn are routed as differential signals to provide noise immunity in the electrical environment in which the PHY is operating.)

Many modern PHYs provide a means to tune the magnitude of the current, iSINK. To do this digitally, a component called an IDAC is used. The IDAC is a form of digital-to-analog converter that produces a current determined by a digital control value.

An example Verilog definition for the IDAC of the output driver appears in Figure 10. The call to the PLI function "$csrc" instantiates the variable circuit element. The body of the "always" block implements the simulation behavior of the IDAC, changing the value of the current in response to a change in a digital input.

```
module idac (inout wire out,
             input [5:0] ival);

  real isink;

  initial begin
    isink = 0.0; // initial current
    $csrc(out, GND, isink);
  end

  always @(ival)
    case (ival)  // decode value
      5'b00000: isink = 0.1;
      5'b00001: isink = 0.15;
      ...
      5'b11111: isink = 1.65
    endcase
endmodule
```

Figure 10. IDAC Behavioral Model Code Listing

3.2 Output Driver with Equalization

The addition of the equalization capability adds multiple stages to the output driver. The architecture of a three-stage DCSODwE (Differential Current-Steering Output Driver with Equalization) is shown in Figure 11. Here, banks of the simple output driver are ganged together, each contributing to the pulldown swing of the differential outputs. The flip-flops are used to produce the bit stream x_i, x_{i-1} and x_{i-2}; these are normal digital components. The current source magnitude of each stage is programmable through the values of ival0, ival1 and ival2. This architecture implements the following mixed-signal recurrences.

(Eq 2) $vp_i = V_{TERM} – (ival_0\ x_i + ival_1\ x_{i-1}\ ival_2\ x_{i-2})$

and

(Eq 3) $vn_i = V_{TERM} – (ival_0\ !x_i + ival_1\ !x_{i-1}\ ival_2\ !x_{i-2})$

4 Experimental Results

We modeled the output drivers of a testchip PHY using the electrical elements shown here, and modified an existing purely-digital testbench to one that checked the voltage-based recurrence values of Equation 2 on pin "vp". Our testbench wrote to the registers controlling the coefficient values (idac0, idac1 and idac2) and generated a data stream.

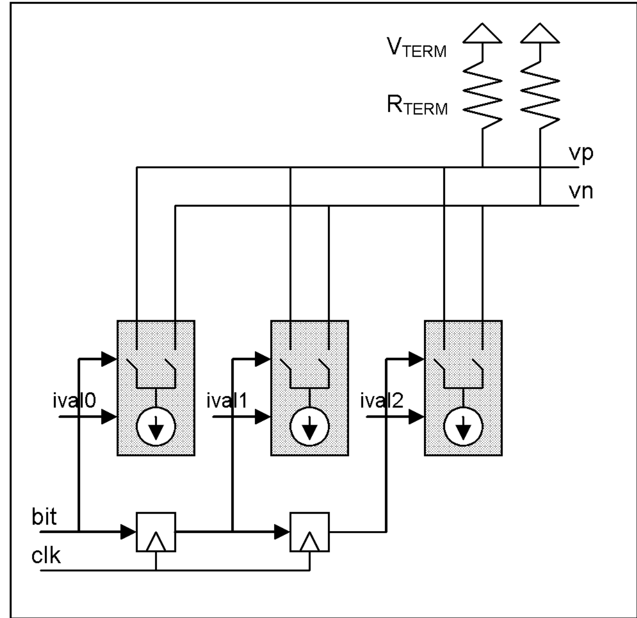

Figure 11. Three-Stage Output Driver with Equalization

Figure 12 shows the resulting piecewise-constant waveforms from such a system. The signal at the top shows the digital-only output of the pre-existing testbench. The second signal is the transmitted "analog" value, and the third is the analog signal as sampled by a receive clock. The fourth signal is the "expect" value as given by the definition recurrence. The fifth signal is the numerical difference between the sampled and the expect value, which remains zero (within a small floating point tolerance) through the course of the simulation, indicating no error.

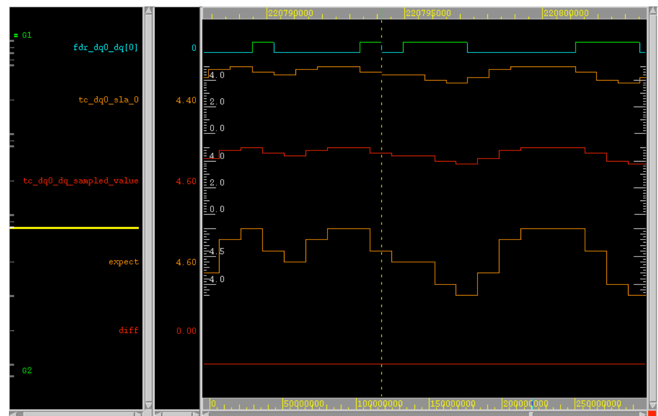

Figure 12. Waveform Capture from Good System

4.1 Performance

We measured the computational overhead of our approach. Each electrical subnetwork should require roughly one matrix inversion to evaluate its steady state, and we were interested in how much CPU time this would cost. The other main use of time is PLI (VPI) overhead.

As a baseline, the PHY system without any analog modeling required 251 CPU seconds to run a complete test. The full system has a three stage output driver, but we also wanted to observe the performance of systems with simpler output drivers and compare the results. We created systems with one, two, and three-stage output drivers and measured the CPU time used using both dense and sparse matrix solvers. Our dense matrix solver is a straightforward LU decomposition with full pivoting. For our sparse matrix solver we used Meschach [8].

Table 1 gives information about these four configurations, including the number of electrical devices, the order of the matrices generated, the number of digital events (toggles) and the number of calls to the matrix solve routines.

Figure 13 presents simulation time data collected for the three configurations as a graph, with the sparse and dense solvers compared side by side for each configuration. We measured the total simulation time, and the CPU time used by the numerical matrix inversion. The rest of the time (minus the baseline) was lumped into a category called "Other." Significant uses of time in this category are PLI overhead, and for the sparse case, the lookup of matrix elements by a search function on their indices.

For the full three-stage output driver, using the sparse solver, the total simulation time rose to 448 seconds. It is interesting that only 64 seconds of this time was spent inverting the matrix. The other 133 seconds is dedicated to PLI interfacing, and searching the matrix for entries. For this same configuration, the dense solver devoted 86 seconds to overhead; this amount is all PLI interfacing time, since the matrix entry time update is insignificant.

It is interesting to compare the dense implementation for the extremely small subnetwork resulting from a single stage output driver. For this case, the total simulation time is less than that for the sparse case. Notably the "other" time is much smaller, because the matrix elements are located using direct index arithmetic. Our dense LU algorithm is not very well optimized, actually, and we think we could make it better. This suggests that for very small subnetworks a dense matrix solver might be preferred over a sparse one.

TABLE I. TEST CONFIGURATIONS

	No drivers	1 driver	2 drivers	3 drivers
Devices	0	216	345	432
Order	0	7 (x36)	10 (x36)	13 (x36)
Switch toggles	0	1.1M	2.2M	3.4M
Solves	0	573K	852K	998K

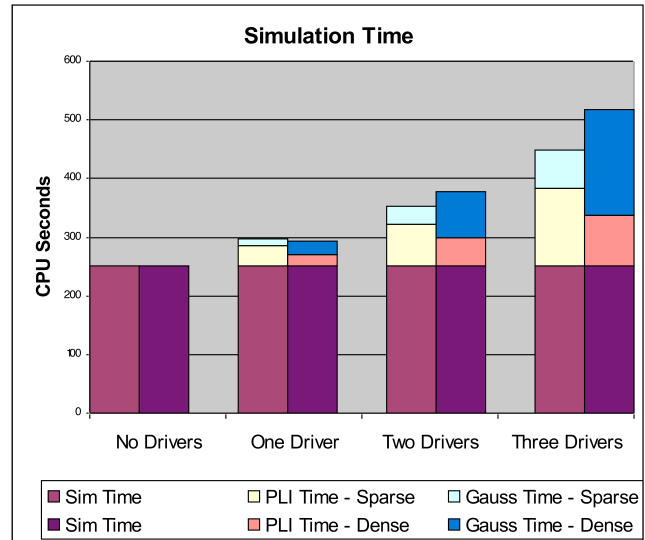

Figure 13. Simulation Times

5 CONCLUSION

This paper described the design and application of a switch-level analog extension to Verilog. We showed an example mixed-signal subsystem and described how to model and verify its functional behavior using the extension. With simulation performance data, we showed that the computational overhead is reasonable for the output drivers shown.

6 REFERENCES

[1] David Bedrosian and Jiri Vlach. *Time-Domain Analysis of Networks with Internally Controlled Switches.* IEEE Transactions on Circuits and Systems. Vol 39. No 3. 1992.

[2] Randal E. Bryant. *Boolean Analysis of MOS Circuits.* IEEE TCAD, 6(4), pp. 634-649, Jul. 1987.

[3] Chris S. Jones, Jeff McNeal and Ross Segelken. *Sending Analog Values Along Digital Wires.* In Proceedings DVCon, 2007.

[4] Vanco Litovski, Milan Savic and Seljko Mrcarica. *Electronic Circuit Simulation with Ideal Switches.* HAIT Journal of Science and Engineering B, Volume 2, Issues 3-4, pp. 476-495. 2005.

[5] Lawrence T. Pillage, Ronald A. Rohrer and Chandramouli Visweswariah. *Electronic Circuit and System Simulation Methods.* McGraw-Hill, TX. December 1, 1994.

[6] Thomas J. Sheffler. *Mixed-Signal Integration: Functional Verification in the Presence of Linear Analog Components.* In Proceedings DesignCon 2008.

[7] Thomas J. Sheffler. *Functional Verification in the Presence of Linear Analog Components.* In Proceedings DVCon, 2008.

[8] David Stewart. *Meschach*. http://www.math.uiowa.edu/~dstewart/meschach/

[9] Stuart Sutherland. *The Verilog PLI Handbook.* Springer. 2002.

[10] Verilog-XL User Guide. Product Version 5.1. Sept 2003.

[11] Carl Werner, Claus Hoyer, Andrew Ho et. al. *Modeling, Simulation, and Design of a Multi-Mode 2-10Gb/s Fully Adaptive Serial Link System.* In Proceedings 2005 Custom Integrated Circuits Conference.

[12] Jared L. Zerbe, Carl W. Werner, Vladimir Stojanovic et. al. *Equalization and Clock Recovery for a 2.5-10-Gb/s 2-PAM/4-PAM Backplane Transceiver Cell.* IEEE Journal of Solid-State Circuits. Vol 38, No. 12. 2003.