Verification of Digitally Calibrated Analog Systems with Verilog-AMS Behavioral Models

Robert O. Peruzzi, Ph. D.
Agere Systems, Allentown, PA
Peruzzi@IEEE.com
Member, IEEE

ABSTRACT

This paper describes the use of behavioral models to verify the design of digitally calibrated analog/mixed-signal systems, to large for either a circuit simulator or a traditional digital simulator. An important goal of using models in this way is to reduce the likelihood of human error resulting in integrated circuit imperfections. A specific example of a calibrated gain stage, with a full set of behavioral models illustrates the modeling and simulation approach.

1. INTRODUCTION

As the size and complexity of integrated circuits increases, it becomes a critical task to find and eliminate occurrences of human error in the design and integration of the constituent subcircuits. This paper is intended to be a practical discussion of how to create and use behavioral models to verify mixed analog/digital systems that use digital signal processing to calibrate the analog signal path. Behavioral models make it possible to simulate such a system in its entirety. Without behavioral models, one is limited to creating a family of overlapping test benches in an attempt to simulate all of the interfaces.

1.1. Why Calibration

A trend in integrated circuits is to combine analog and digital circuitry on the same die. A concurrent trend is to shrink the dimensions of device geometries for digital circuitry in order to increase speed as well as to lower cost by decreasing area. Power supply voltages decrease because of physical limitations of the finer geometries as well as to decrease power dissipation. Analog design challenges resulting from these trends include reduced headroom and increased susceptibility to noise.

Using open loop analog is a solution to the headroom problem. A mixed analog/digital calibration system replaces analog feedback.

The calibrator converts an analog quantity (voltage, current or time interval) to digital and uses it to calculate some decision criterion. It makes the decision to increase or decrease some controlling parameter (voltage, current, capacitance etc.) to adjust the measured analog quantity. It applies the amount of increase or decrease digitally by means of a D/A converter or switch. The calibrator re-measures the analog quantity of interest and the process repeats until the measured quantity is within acceptable limits.

Calibration executes at start up and periodically thereafter and provides feedbacks that mitigates offset, gain, frequency, or filter corner frequency errors, reduces drift with temperature, power supply voltage and aging.

1.2. Why Behavioral Modeling

Digital calibration of analog circuitry is well known and accepted [1-4], but an entire integrated circuit may be difficult to verify by simulation. A traditional circuit simulator might possibly simulate a system comprising the core analog circuit, A/D converter, signal processing digital circuitry, D/A converter, and analog switches. However, a typical calibrated system shares its A/D converter and signal processing digital circuitry between multiple core analog building blocks. It becomes impractical if not impossible to verify the entire system with a traditional circuit simulator.

Reducing the likelihood of human error is an important motivation for using behavioral modeling to verify the circuit. Errors inevitably occur when there are opportunities for miscommunication between designers. In a large design team, especially one that is widely distributed functionally and geographically, the risk of human error is high. In particular, the following is a list of types of design errors:

1.2.1 Faulty Calibration Algorithm

There may be a fundamental flaw in the semantics of the calibration plan. An example of a gross error would be failing to power up the target before applying the test signal. An example of a subtle error would be a counter wrapping around to zero on overflow, causing wildly unexpected results. A simulation platform that allows runtime-efficient testing of many cases can expose such a flaw.

1.2.2 Bus Bit Order Errors

A common miscommunication is reversal of bit ordering. A digital subcircuit may produce output DATA [7:0], and the analog destination expects DATA [1:8]. Human errors occur despite the best intentions and diligence, methodology documents, design rules, checklists and daylong public design reviews.

1.2.3 Digital Control Signal Polarity Errors

Another miscommunication error is polarity reversal. Confusion can result between the meaning of enable and power-down signals, and their active-high and active-low nature. Again, failures occur despite rules and safeguards.

1.2.4 Digital Signal Integrity Errors

Timing violations in digital circuitry such as setup time or hold time can result in inconsistent states on signals passed to an analog circuit. These can be a nightmare to debug in the field or laboratory because errors can appear at only certain clock rates, power supplies or temperatures. Using a reasonably pessimistic digital simulator will catch timing violations, with either static timing analysis or transient analysis. Errors can occur when the complete set of digital verifications are not performed on a block of digital circuitry appearing in a mixed-signal schematic.

1.2.5 Bias Current Errors

Different team members may design the analog signal path and its bias generator. A common miscommunication error occurs when the signal path designer assumes the wrong current polarity, or magnitude. Another type of error occurs when multiple designers inadvertently connect to the same bias current source.

1.2.6 Reference Voltage Errors

Similarly, a designer may connect to a different reference voltage than intended. Another error scenario occurs when a designer attempts to draw current from an un-buffered voltage reference.

To solve these problems, this paper describes modeling strategies and provides Verilog-AMS examples that can expose these types of faults.

The remainder of this paper is comprises three sections. Section 2 presents an ideal analog gain stage with digital calibration and describes its constituent blocks. Section 3 begins by describing the AMS models representing the subcircuits of the calibrated gain stage, referring to the model listings in the appendix. Section 3 also discusses how the models overcome certain difficulties with the AMS simulator, and gives suggestions for using multiple model views of circuit blocks for flexible use in verification. Section 4 concludes with a summary of how the models address each of the error modes listed in section 1.2. The appendix is a code listing of the AMS models, somewhat condensed for space.

2. A CALIBRATED ANALOG CIRCUIT

An example of a digitally calibrated gain stage is in Figure 1. This fictional system is not practical or realizable as shown, but is a useful vehicle to demonstrate modeling techniques.

The circuit comprises

- An amplifier with digitally controlled coarse gain, current controlled fine gain and current controlled offset, and other digital features
- An analog multiplexer to select normal input or input from a calibration voltage
- An analog switch to connect or not connect the output to the ADC input multiplexer
- An ADC with multiplexed input
- Digital signal processing logic controls all the switches and multiplexers, the ADC and the DACs, and generate the DAC words.
- DACs to create the gain control and offset control currents and a DAC to create the calibration test voltage.

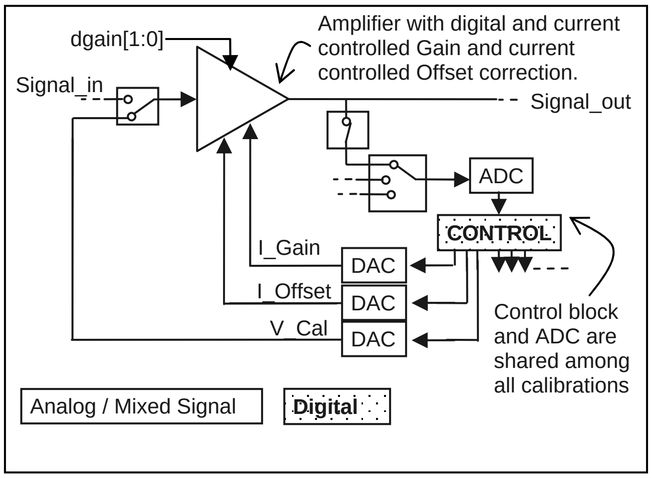
dgain[1:0]
Amplifier with digital and current controlled Gain and current controlled Offset correction.
Signal_in
Signal_out
ADC
CONTROL
I_Gain
DAC
I_Offset
DAC
V_Cal
DAC
Control block and ADC are shared among all calibrations
Analog / Mixed Signal
Digital

Figure 1: A digitally calibrated analog gain stage.

In normal operation, the signal path is from Signal_in through the amplifier to Signal_out, with the I_Gain and I_Offset DACs operating, and the ADC and the V_Cal DAC powered down. As illustrated in Figure 1 the gain stage is in calibration mode. The amplifier input connects to V_Cal, and Signal_out connects to the ADC.

For verification simulations, the CONTROL subcircuit may be represented by either behavioral or RTL Verilog models or gate level netlists, with or without parasitics. Verilog AMS models represent the amplifier, analog switches, ADC and DAC.

3. AMS MODELS AND DESCRIPTIONS

Verilog AMS models for the amplifier, switches, ADC and DACs, somewhat abridged, are listed in the appendix. This section presents and discusses the models.

3.1 Model Overviews

This section describes each of the models. The documentation found in [5] is an excellent reference for further understanding the modeling approach and code syntax.

3.1.1 Amplifier

The complete Verilog-AMS model for the amplifier shown at the top of Figure 1 is in Listing 1 in the Appendix. The device is off when input pdn is low. Two dgain bits control coarse gain. Currents i_gain and i_offset calibrate fine gain and offset. The frequency compensation requirement is different for certain settings of dgain under control of the dcomp bit. A 2.5 Volt power supply, 15-uA bias current and 0.6-V reference are required. In the "initial begin" section the compiler directive variable cal_sim controls the use of gain and offset errors.

"// I Power down pin…" defines behavior for states 1 (**real** en_step = 1.0) and 0 (en_step = 0.0) of the pdn input, and also defines behavior in case of error conditions when pdn = X or Z, providing coverage for error mode 1.2.4 "digital signal integrity". The bus variable logic_ok has bits to monitor the integrity of every digital input. Skipping ahead to segment *"// I.A"* in the analog section, the discontinuous product *"en_step * noFault_step"* gets smoothed by the transition function into the continuous real variable en, which scales the output to zero when powered down or when there's a power supply, bias or logic fault.

Regarding transition(), sometimes you can get away without smoothing when the testbench is small and the transient simulation is short. The likelihood of analog convergence errors increases with the size of the circuit under test – as does the difficulty in finding and fixing the cause of non-convergence.

Secondly, a real variable changing in the digital section of an AMS model does not necessarily trigger an analog solution point. In a short transient simulation, the max timestep sometimes will camouflage the time interval between the digital real variable change and its effect in the analog environment. Use of the transition filter explicitly defines an analog solution point

"//II. Digitally controlled nominal gain…" defines the nominal amplifier gain. The case statement includes all legal values of dgain and the default detects signal integrity faults. The ngain_step is smoothed to ngain in the analog section.

" //III. Digitally controlled compensation for low gain"
The effect of frequency compensation is not easy to model behaviorally, and if one did model its effect, the model would not convey verification information in a useful format. A better approach is to design and simulate the amplifier using the circuit simulator, then knowing the effect of the dcomp input when properly programmed, verify the controllability and integrity of dcomp. Whenever dcomp changes, an INFO message displays the value of dcomp. If there is a dcomp fault, the model displays an ERROR message and shuts down the amplifier. Shutting down the amplifier because the compensation control bit is floating may not be physically accurate but the objective here is to call attention to a connectivity fault.

" // IV. Monitor vdd"
Two instances of the *"@(above…)"* function trigger if the power supply is over or under 2.4 V at time zero, or if the power supply crosses 2.4 V at any time in the simulation. Power supply greater than 2.4 V sets power_ok and allows normal operation. Two more instances of *"@(above…)"* could add an upper limit to the power supply. One may check connectivity to ground in the same way. Skip ahead to *"// IV.A"* in the analog section. The vdd pin is terminates resistively to gnd and a power supply current is assigned when powered up. A collection of AMS models written in this manner can help tally total power supply current under various operating modes. Measure with the circuit simulator and assign currents in the AMS model.

" // V. Monitor bias current"
Two instances of the *"@(above…)"* function trigger if the bias current is not within tolerance, and set the corresponding bit of the bias_ok variable. Incorrect current will clear the bias_ok bit, causing the output to shut down and the display of an ERROR message. The currents terminate through an arbitrary 100 Ohms to gnd in the analog section.

" // VII. Continuously check for errors"
Variable all_ok is the logical AND of the power, bias and logic integrity monitor variables, and controls the real variable noFault_step.

*" // analog **begin**"*
With the AMS simulator, all currents and voltages terminate to ground by default (through minimum resistance or conductance) if not explicitly terminated. In this model, i_gain and i_offset are explicitly, if arbitrarily, terminated to gnd through 100 Ohms. Factors are calculated to correct gains from 0.98 to 1.02 V/V, and offsets from -20 to 20 mV. After calibration, effective gain approaches nominal gain and effective offset approaches zero.

This model modifies the ideal linear gain equation (after gain and offset errors and corrections) to allow for hard limiting to vdd and gnd, bandwidth limiting to an arbitrarily selected bandwidth of 1 GHz, and 100 Ohms output impedance (1 GOHM output impedance when powered down).

3.1.2 Two-Pole Switch
The Verilog-AMS model (abridged for space) for the two-pole switch at the input of the amplifier in Figure 1 is in Listing 2 in the Appendix. When sel1 (select 1) is low, there is a low impedance path between in0 and out, and a high impedance between in1 and out. When sel1 is high, in1 connects to out with low impedance.

The model checks the signal integrity of sel1, and both paths are high impedance if it is at fault. Power and ground connections do not appear in this model for the sake of brevity.

The modeling strategy for this idealized model is to make the switch resistance either the maximum or the minimum resistance for Verilog AMS, and to exponentially transition between the base-10 logarithms of these extremes. This approach allows continuous, fast transitions without convergence problems.

3.1.3 One-Pole Switch
The model listing for the single-pole switch between Signal_out and the ADC does not appear in the appendix for brevity. It uses the same strategy as used in the two-pole model. That is, "enable" controls the impedance between in and out.

3.1.4 Three-Pole Switch
Figure 1 shows a three to one multiplexer connecting to the ADC, but there may be significantly more inputs in a typical calibrated analog system. A model for a three-pole switch is in Listing 3. The model uses the same controlled impedance approach as the previous two switches. The model listing is included in appendix because it illustrates a different type of error verification. Two control bits are enough to control four inputs, but there are only three. The case statement will flag an error if sel = 1'b11, separate from the signal integrity check that sel <= 2'b11.

3.1.5 ADC
Listing four is an abridged model of an ideal ADC. Comments replace segments of code previously discussed. The A/D conversion code executes totally in the digital section of the model.

3.1.6 Current and Voltage DACs
Listing five is an abridged model of a current output DAC with level sensitive digital input. Notice the clock level evaluation code in the "initial begin" section. This is necessary for accurate level sensitive switch modeling. As in the gain_stage model, the real variables are set according to the digital word in the digital domain, and a transition function in the analog domain drives the output. With a level sensitive DAC, the output will follow the digital input if it changes while the clock is high, and will wait for a high clock level if the digital inputs change while the clock is low. The code for a voltage-output DAC is similar.

3.2 Difficulties Overcome
3.2.1 Digital/Analog Simulator Control Sharing
The Verilog-AMS simulator uses a continuous-time analog simulation engine as well as an event-driven discrete-time digital simulation engine. Controlling and maintaining communication between the two simulators presents a difficulty overcome by these models. The method described above of using a "_step" variable to trigger an analog evaluation from a digital event provides a communication path from the digital solver to the analog solver. Conversely, triggering a digital even from the analog domain is done using the "@(above…) function shown in the gain_stage model and description.

3.2.2 Wide Frequency Difference
In normal operation, the analog signal path may be high frequency, but operate at nearly DC during calibration mode. This puts another demand on the simulator, in controlling the maximum time step between analog evaluations. Using the "_step" variables and transition() and @(above…) functions overcomes this difficulty by explicitly setting evaluation points.

3.3 Simpler Model Versions or Views
It is possible to have several levels of model detail available for each block. The listings in the appendix correspond to the most detailed view, used to verify control and calibration of the analog signal path.

In a larger system, the digital circuitry downstream from the analog front end may be ready for verification before the analog front-end design is complete. Owners of the downstream digital circuitry may have no design or verification responsibility for the analog blocks or their calibration, but use the analog blocks as part of their test platform. Such users may substitute a simpler model view, in which nominal analog performance takes place as if after successful calibration.

One approach to accommodating the need for simpler models is to maintain multiple model views of the analog blocks, using a tool such as the Cadence Hierarchy Editor to assemble a test platform with the desired level of detail. Another approach uses a single edition of the model source code, but delineates the more detailed model code between `ifdef, `elseif and `endif compiler directives.

4. CONCLUSIONS

These models can identify all six classes of design flaw. This section reviews the error modes listed in sections 1.2.1 to 1.2.6, pointing out how the models expose these types of errors.

4.1 Faulty Calibration Algorithm

If calibration never takes place because of a gross error such as failure to power-up, or some other sequencing fault in the algorithm, the gain and offset errors would still be evident in Signal_out. Simulating calibration with several sets of initial errors of both polarities, and both small and extreme magnitude are necessary. For completeness, also try error magnitudes too large for complete correction and ensure the calibration routine fails “gracefully”. Try with zero error too, making sure the calibration routine does not degrade a perfect situation.

4. 2 Bus Bit Order Errors

The modeling strategy makes it straightforward to expose any bus bit order errors. For instance, reversal of the dgain input bits to the gain_stage causes the amplitude of Signal_out to be different from the expected. Behaviors depending on bit ordering appear in each model.

4. 3 Digital Control Signal Polarity Errors

Behaviors depending on digital control signal polarity are part of each model. Verify by comparing observed output to the expected output. In the case of the dcomp input to gain_stage, the INFO message in the log file will show if the resulting block input has the expected polarity.

4.4 Digital Signal Integrity Errors

Digital integrity errors show up in the simulation log file and stop normal simulation output. All the models are written such that logic input faults become evident.

4.5 Bias Current Errors

The models do not allow incorrect bias current levels or polarities. Whenever the current is outside of limits, the model shuts down and an ERROR message indicates the incorrect bias.

4.6 Reference Voltage Errors

In the same manner, the models do not allow incorrect voltage references without shutting down and displaying an ERROR message. Drawing current from a reference with a source resistance causes a voltage drop, which is detectable.

REFERENCES

[1] Bailey, James; Franck, Stephen, US Patent 20,050,270,092, Calibration technique for variable-gain amplifiers.

[2] Matsuzawa,A., "Is the Golden Age of Analog Circuit Design Over?", ISSCC 2004 Panel. www.ssc.pe.titech.ac.jp/materials/ISSCC04_panel_matsu_homepage.pdf

[3] Moon, Un-Ku. Song, Bang-Sup. “Background digital calibration techniques for pipelined ADCs”, *IEEE Transactions on Circuits and Systems II: Express Briefs*, Vol. 44, No. 2, Feb 1997, pp 102.

[4] Pastre, Marc, Kayal, Maher. *Methodology for the Digital Calibration of Analog Circuits and Systems with Case Studies* Series: The International Series in Engineering and Computer Science , Vol. 870, 2006.

[5] Kundert, Kenneth S., Zinke, Olaf, *The Designer’s Guide to Verilog AMS*, Kluwer Academic Publishers, 2004.

APPENDIX: Verilog-AMS Models

Listing 1. Amplifier Model

```
`include "constants.vams"
`include "disciplines.vams"
module gain_stage (vin, pdn, dgain, dcomp, i_gain,
i_offset, ibias, vref, vdd, gnd, vout);
  input vin, pdn, dgain, dcomp, i_gain, i_offset, ibias,
vref, vdd, gnd;
  input  [1:0] dgain;
  electrical ibias, , i_gain, i_offset, vin, vout,  vdd,
gnd;
  logic [1:0] dgain;
  logic pdn, dcomp;

    // Nominal gain from digital control
  real ngain_step, ngain;
    // Arbitrary errors to test calibration
  real gain_err, os_err;
    // Correction factors from calibration
  real gain_corr, os_corr;
    // Effective gain and offset
  real egain, eos;
    // For power down control
  real en_step, en;
    // For verifying input integrity
  reg power_ok;
  reg [1:2] bias_ok;
  reg [1:3] logic_ok;
  wire all_ok;
  real noFault_step;
    // Internal versions of the output
  electrical out_raw, out_clip, out_bw;
  real BW, ROUT;
  initial begin
    power_ok = 1'b1;
    bias_ok = 2'b11;
    logic_ok = 3'b111;
      // All step variables passed to transition filter
      // must be initialized
    noFault_step = 1.0;
    en_step = 0.0;
    ngain_step = 1.0;
    BW = 2 * 3.414 * 1e9;
    `ifdef cal_sim       // If running calibration test
      gain_err = 1.01; // 1% arbitrary gain error
      os_err = 0.01;   // 10 mV arbitrary offset error
    `else              // No errors otherwise
      gain_err = 1.0;
      os_err = 0.0;
    `endif
  end                  // end of initial begin section
 // I. Power down pin  (Also see I.A)
  always @(pdn) begin
    logic_ok[1] = 1'b1;
    if (pdn == 1'b1)
      en_step = 1.0;
    else if (pdn == 1'b0)
      en_step = 0.0;
    else begin         // React to fault condition on
pdn
      en_step = 0.0;
      logic_ok[1] = 1'b0;
        $display(
          "ERROR : %g : %m : Bad logic: pdn = %1b ",
        $realtime, pdn);
    end
  end
  //   II. Digitally controlled nominal gain: ngain
  // Here assigned as step, smoothed in analog
  // section with transition filter. (See II.A)
  always @(dgain) begin
    logic_ok[2] = 1'b1;
    case (dgain)
      2'b00: ngain_step = 1.0;
      2'b01: ngain_step = 2.0;
      2'b10: ngain_step = 4.0;
      2'b11: ngain_step = 8.0;
      default begin  // Fault condition on dgain bits
        logic_ok[2] = 1'b0;
        ngain_step = 1.0;
          $display(
         "ERROR : %g : %m : Bad logic: dgain = %2b ",
          $realtime, dgain);
       end
     endcase
  end
  //   III. Digitally controlled compensation for low
gain
  // Behavior is too subtle to model physically
  // Verify path connectivity with INFO messages
  always @(dcomp) begin
    logic_ok[3] = 1'b1;
    if (dcomp <= 1'b1) begin
      $display(
"INFO : %g : %m : You have written: dcomp = %1b
",
      $realtime, dcomp);
    end
    else begin
      logic_ok[3] = 1'b0;
      $display(
       "ERROR : %g : %m : Bad logic: dcomp = %1b ",
      $realtime, dcomp);
    end
  end
```

Listing 1. Amplifier Model (Cont.)

```
// IV. Monitor vdd (See IV.A)
  always @(above (V(vdd) - 2.4)) begin
    power_ok = 1'b1;
      $display (
        "INFO : %g : %m : VDD ON. vdd = %g",
      $realtime,V(vdd));
  end
  always @(above (2.4 - V(vdd))) begin
    power_ok = 1'b0;
      $display (
        "INFO : %g : %m : VDD OFF. vdd = %g",
      $realtime, V(vdd));
  end
  // V. Monitor bias current (See V.A)
  always @(above(abs(V(ibias) - 1.5e-3) - 0.1e-3))
       // |err| > tol.
    bias_ok[1] = 1'b0;
  always @(above(0.1e-3 - abs(V(ibias) - 1.5e-3)))
         // |err| < tol.
    bias_ok[1] = 1'b1;
  // Respond to incorrect current
  always @(negedge bias_ok[1]) begin
    if (bias_ok[1] == 1'b0)
      $display(
  "ERROR : %g : %m : Bad bias current: ibias = %g",
      $realtime, V(ibias)/100.0);
  end
  // VI. Monitor reference voltage
  // Expected voltage is 0.6 +/- 0.03
       // When  |err| > tolerance:
  always @(above(abs(V(vref) - 0.6) - 0.03))
     bias_ok[2] = 1'b0;
      // When  |err| < tolerance:
  always @(above(0.03 - abs(V(vref) - 0.6)))
     bias_ok[2] = 1'b1;
  // Respond to incorrect voltage
  always @(negedge bias_ok[2]) begin
    if (bias_ok[2] == 1'b0)
      $display(
"ERROR : %g : %m : Bad ref. voltage: vref = %g",
      $realtime, V(vref));
  end
    // VII. Continuously check for errors
  assign all_ok = & {power_ok, bias_ok, logic_ok};
    // VIII. If no errors... (Also see I.A)
  always @(all_ok) begin
    if (all_ok == 1'b1) noFault_step = 1.0;
    else noFault_step = 0.0;
  end
```

```
analog begin
    // I.A Continuous scale factor disables the output
    // when powered down or during a fault condition
  en = transition(en_step * noFault_step, 0.0, 1n,
1n);

    // II.A Smooth steps on ngain
    ngain = transition(ngain_step, 0.0, 1n, 1n);
    // Terminate i_gain and i_offset
    V(i_gain) <+ I(i_gain) * 100.0;
    V(i_offset) <+ I(i_offset) * 100.0;
    // Develop correction factors
 gain_corr = I(i_gain) * 4e3 + 0.96; // 0.98 to 1.02
V/V
 os_corr = I(i_offset) * 4e3 - 0.04; // -20 to 20 mV
   // Effective gain: nominal gain, error and correction
    egain = ngain * gain_err * gain_corr;
    // Effective offset, including error and correction
    eos = os_err + os_corr;

    // Signal path:  Linear gain equation
    V(out_raw) <+ (V(vin) * egain + eos) * en;
    // Hard limit to ground and power supply
    if (V(out_raw) > V(vdd))
      V(out_clip) <+ V(vdd);
    else if (V(out_raw) < V(gnd))
      V(out_clip) <+ V(gnd);
    else
      V(out_clip) <+ V(out_raw);
    // Bandwidth limited
    V(out_bw) <+ (V(out_clip)-ddt(V(out_bw)/BW));
    // Output resistance
    ROUT = 100 * (1.0 - en) * 1e9;
    // Final output
    I(vout, out_bw) <+ V(vout, out_bw) / ROUT;

 // IV.A  Typical power supply currents from circuit
 // simulation. Assigned here as a "book keeping"
aid.
 // 1 nA leakage plus 1 mA when enabled
    I(vdd, gnd) <+ V(vdd, gnd) / 2.5e9 + 1e-3 * en;

 // V.A Develop bias current voltage over 100 Ohms
    V(ibias) <+ I(ibias) * 100.0;

  end   // end of analog begin

endmodule
```

Listing 2. Two-Pole Switch Model

```
module switch2p ( in1, in0, sel1, out);
  inout in1, in0, out;
  input sel1;
  electrical in1, in0, out;
  logic sel1;
    // Internal variables
  real OPEN_LOG = 27;   // log10(open resistance)
  real CLOSE_LOG = -4; // log10(close resistance)
  real R1_LOG, R1, R0_LOG, R0;

  initial begin
    R1_LOG = OPEN_LOG;
    R0_LOG = CLOSE_LOG;
  end

  always @(sel1) begin
    if (sel1 == 1'b1) begin
      R1_LOG = CLOSE_LOG;
      R0_LOG = OPEN_LOG;
    end
    else if (sel1 == 1'b0) begin
      R1_LOG = OPEN_LOG;
      R0_LOG = CLOSE_LOG;
    end
    else begin
      R1_LOG = OPEN_LOG;
      R0_LOG = OPEN_LOG;
      $display(
        "ERROR : %g : %m : Bad logic: sel1 = %1b ",
      $realtime, sel1);
    end
  end

  analog begin
    R1 = exp(transition(R1_LOG, 0.0, 10e-9, 10e-9));
    R0 = exp(transition(R0_LOG, 0.0, 10e-9, 10e-9));
    I(in1, out) <+ V(in1, out) / R1;
    I(in0, out) <+ V(in0, out) / R0;
  end
endmodule
```

Listing 3. Three-Pole Switch Model

```
…
module switch3p  ( in2, in1, in0, sel, out);
  inout in2, in1,in0, out;
  input [1:0] sel;
  electrical in2, in1,in0, out;
  logic [1:0] sel;
  real OPEN_LOG = 27; // log10(open resistance)
  real CLOSE_LOG = -4; // log10(close resistance)
  real R2_LOG, R2, R1_LOG, R1, R0_LOG, R0;
 initial begin
   R2_LOG = OPEN_LOG;
   R1_LOG = OPEN_LOG;
   R0_LOG = CLOSE_LOG;
  end

  always @(sel) begin
    if (sel <= 2'b11) begin
      case (sel)
        2'b00: begin
          R2_LOG = OPEN_LOG;
          R1_LOG = OPEN_LOG;
          R0_LOG = CLOSE_LOG;
        end
        2'b01: begin
          R2_LOG = OPEN_LOG;
          R1_LOG = CLOSE_LOG;
          R0_LOG = OPEN_LOG;
        end
        2'b10: begin
          R2_LOG = CLOSE_LOG;
          R1_LOG = OPEN_LOG;
          R0_LOG = OPEN_LOG;
        end
        default begin
          R2_LOG = OPEN_LOG;
          R1_LOG = OPEN_LOG;
          R0_LOG = OPEN_LOG;
          $display(
         "ERROR : %g : %m : Illegal value: sel = %2b ",
          $realtime, sel);
        end
      endcase
    end
    else begin
      R2_LOG = OPEN_LOG;
      R1_LOG = OPEN_LOG;
      R0_LOG = OPEN_LOG;
      $display(
        "ERROR : %g : %m : Bad logic: sel = %2b ",
      $realtime, sel);
    end
  end

analog begin
    R2 = exp(transition(R2_LOG, 0.0, 10e-9, 10e-9));
    R1 = exp(transition(R1_LOG, 0.0, 10e-9, 10e-9));
    R0 = exp(transition(R0_LOG, 0.0, 10e-9, 10e-9));
    I(in2, out) <+ V(in2, out) / R2;
    I(in1, out) <+ V(in1, out) / R1;
    I(in0, out) <+ V(in0, out) / R0;
  end
endmodule
```

Listing 4. 8-Bit ADC Model

```
…
`define NUM_ADC_BITS   8
`define tconv #1
module adc8 (vin, clk, pdn, ibias, vref, vdd, gnd,
dout);
  input vin, clk, pdn, ibias, vref  vdd  gnd;
  output  [`NUM_ADC_BITS-1:0] dout;
  electrical vin, ibias, , vdd, gnd;
  logic clk, pdn;
  wire [`NUM_ADC_BITS-1:0] dout;
  // Internal Variables
    // For verifying input integrity, pdn control etc.
…
    // ADC Variables
  parameter real  vmax = 2.2;
  parameter real  vmin = 0.2;
  real sample, vmid, lsb, voffset;
  reg [0:`NUM_ADC_BITS-1] vd;
  integer i, ii, binvalue;
  reg [`NUM_ADC_BITS-1:0] dout_raw;

  initial begin
    … input integrity, power etc. initialization
    … step variable initialization
    vmid = (vmax - vmin) / 2.0;
    lsb  = (vmax - vmin) / (1 << `NUM_ADC_BITS) ;
    voffset = vmin;
    for (i = `NUM_ADC_BITS-1; i >= 0; i = i-1) begin
      vd[i] = 0 ;
    end
  end
  // I. Power down pin  (Also see I.A)
  …
  // II. ADC Action
  always @(negedge clk) begin
    binvalue = 0;
    sample = V(vin) - voffset;
    for ( ii = `NUM_ADC_BITS -1 ; ii>=0 ; ii = ii -1 )
begin
      vd[ii] = 1'b0;
      if (sample > vmid ) begin
        vd[ii] = 1'b1;
        sample = sample - vmid;
        binvalue = binvalue + ( 1 << ii );
      end
      else begin
       vd[ii] = 1'b0;
      end
      sample = sample * 2.0;
    end
 // Complement MSB
    #1 dout_raw[`NUM_ADC_BITS-1]  <=
~vd[`NUM_ADC_BITS-1];
      for (i = `NUM_ADC_BITS-2; i >= 0; i = i-1) begin
        dout_raw[i]  <=   vd[i];
       end
  end
 assign dout = dout_raw & ~pdn & all_ok;

 // III. Catch any illegal clock states
  always @(clk) begin
    logic_ok[2] = 1'b1;
    if (clk <= 1'b1)
      ;    // Do nothing
    else begin
      logic_ok[2] = 1'b0;
      $display(
        "ERROR : %g : %m : Bad logic: clk = %1b ",
      $realtime, clk);
    end
  end
   // IV. Monitor vdd (See IV.A)
  …

  // V. Monitor and respond to bias current (See V.A)
  …

  // VI. Monitor and respond to reference voltage
 …
  // VII. Continuously check for errors
…
  analog begin
    // I.A Smooth en_step to en
    …
    // IV.A  Assign power supply currents
    …
    // V.A Develop bias current
   …
  end
endmodule
```

Listing 5. 7-Bit Current DAC Model

```
…
module daci7 (dbits, clk, pdn, io, ibias, vref, vdd,
gnd);
  input  [6:0] dbits;
  input clk, pdn, ibias, vref, vdd, gnd;
  output io;
  logic  [6:0] dbits;
  logic clk, pdn;
  electrical ibias, , vdd, gnd, io;
  // Internal Variables
    // For verifying input integrity, pdn control etc.
…
    // DAC variables
  real ilsb, ios, idac, i_ibias, dac_step, dac_scale;
```

Listing 5. 7-Bit Current DAC Model (Cont.)

```
initial begin
    … input integrity, power etc. initialization
    … step variable initialization
// Level sensitive latch initialization
    #1 ;
    if (clk == 1'b1) begin
      if (dbits <= 7'b1111111)          // i.e. if no x's
        dac_step = dbits;
      else            // No error messages at startup
        dac_step = 0.0;
    end
  end   // end of initial begin section
  // I. Power down pin  (Also see I.A)
  …
  // II. Level sensitive DAC action (See section II.A)
  always @ (dbits) begin
    logic_ok[2] = 1'b1;
    wait (clk == 1'b1) begin
      if (dbits <= 7'b1111111)          // i.e. if no x's
        dac_step = dbits;
      else begin
        … bad logic response
      end
    end
  end
 // III. Catch any illegal clock states
     …
  // IV. Monitor vdd (See IV.A)
     …
  // V. Monitor bias current and ref. voltage
     …
  // VII. Continuously check for errors
      …
  // VIII. Zero output if error. (Also see I.A)
      …
  analog begin
    // I.A Smooth en_step to en
    …
    // II.A DAC
      //Derive DAC constants from ibias
i_ibias = I(ibias) * 1.0; // Can't use I(ibias) to
calculate
ios = 2.5 * i_ibias;
ilsb = 10.0/(2.0 * 127.0) * i_ibias;
      // DAC action
  dac_scale = transition(dac_step, 0.0, 10e-9, 10e-
9);
  idac = ilsb * dac_scale + ios;
      // P-channel output current
  I(vdd,io) <+ idac * en;
    // IV.A  Assign power supply currents
    …
    // V.A Develop bias current
   …
  end
endmodule
```