Real Portable Models for System/Verilog/A/AMS

Bill Ellersick
Analog Circuit Works, Inc.

This talk presents a mixed-signal modeling methodology that uses discrete-time real variables to represent voltages and currents, and is portable across the variants of Verilog.

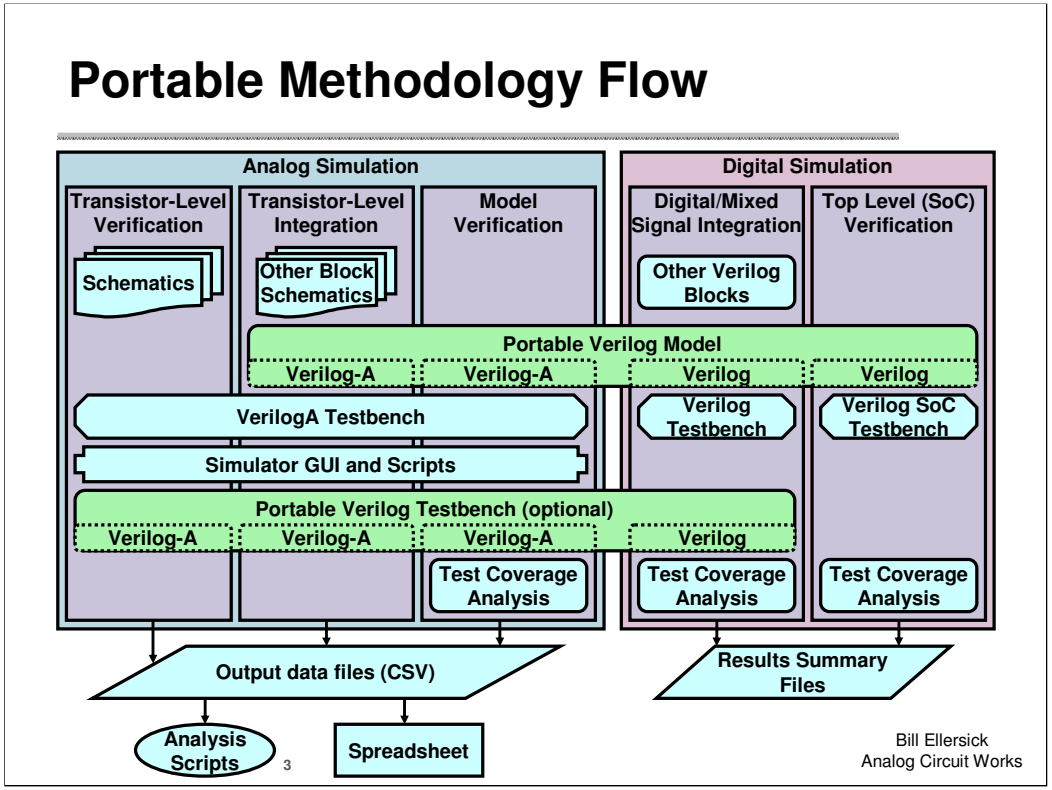
Introduction

- EDA tools now widely support standards
- Real Portable Methodology
  - Portable across processes and tools
  - Rapid discrete-time real-valued behavioral simulation
    - Time honored approach: commonly used in e.g. Matlab
  - Verification of model vs. transistor-level design
  - Enhanced productivity, portable, reusable
- Can develop IP with one set of tools
  - Integrate into SoC with another set of tools
  - Benefits IP providers, users, tool providers

2

Bill Ellersick
Analog Circuit Works

The methodology described in this paper is enabled by EDA tools that now widely support standards, and is based on proven techniques, commonly used for system simulation e.g. in Matlab.
The models described in this paper are portable across processes and tools, and the discrete-time real-valued code simulates rapidly.
Because the models are portable, they can be verified in an analog simulation environment to ensure they match the behavior of the transistor-level designs.
In addition, the use of high-level Verilog code in the models enhances productivity and enables portable and reusable models.
The standards-based models allow mixed-signal IP to be developed with one set of tools, and integrated into SoCs with another set of tools.
This benefits IP providers, who can reach a broader set of customers, it benefits IP users who can use IP from any provider, and it benefits EDA tool providers who can switch customers to their tools with less disruption.



This diagram illustrates the flow of the portable modeling methodology in this paper.
On the left, transistor-level verification of a mixed signal block is driven by a Verilog-A testbench along with non-portable simulator scripts and graphical interface operations.
As the transistor-level design progresses, behavioral models are used to accelerate the simulation of multiple blocks, using the same testbench and scripts to configure and verify circuit operation.
The behavioral models are verified with the analog simulator, again using the same testbench and scripts, to ensure the models accurately represent the transistor-level behavior. If the testbench is portable, Verilog code coverage tools can be used to ensure that the Verilog model is fully exercised by the Verilog testbench to ensure high quality models.
With a verified model, digital simulation with Verilog logic blocks can be performed, and can take advantage of a portable testbench to configure and verify the mixed-signal blocks.
Finally, top-level verification of an SoC is possible with the efficient models in this paper, ensuring that top-level connections and architecture are correct.

Organization

- Introduction
  - Methodology Flow
- Organization
- Real Behavioral Models
  - Discrete-time modeling with Verilog reals and integers
  - ADC with input RC Filter Example (adcX)
  - Real Portable Behavioral Models
  - Verification and Debugging
- Conclusions

4

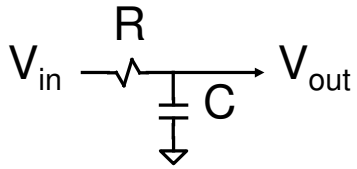
Bill Ellersick
Analog Circuit Works

The rest of the paper will describe real-valued behavioral models, illustrating them with an example consisting of an ADC with an input RC filter.
A method for making the models portable across Verilog variants is presented, following by a discussion of testbenches and debugging of the models, and then the talk is concluded.

Real Behavioral Models

- Real variable models
  - Difference equations
  - $\Delta V_{out} = \Delta t\ I_r / C$
  - $I_r = (V_{in} - V_{out}) / R$
- Accurate for small Δt
- Much faster sims
- Verilog good for models

- Analog models (slow)
  - Differential equations
  - $dV = dt\ I_r / C$
  - $I_r = (V_{in} - V_{out}) / R$
- Simulator chooses Δt



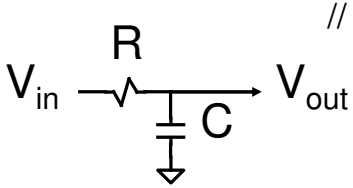
5

Bill Ellersick
Analog Circuit Works

The real-valued and analog models for an RC filter are shown here.
The real-valued model uses difference equations, where the change in voltage on a capacitor is equal to the time interval times the current flowing in the resistor divided by the capacitance. The analog model uses differential equations that are quite similar, where dv/dt is equal to the current divided by the capacitance. The equations for the current in the resistor are identical, equal to the voltage across the resistor divided by the resistance.
The real-valued model is accurate as long as an appropriately small delta-t is used, while the analog simulator chooses delta-t.
However, the real-valued equations need only be executed once when the inputs change significantly, while the analog differential equations must be iteratively executed to converge on a solution at each time step.
And it turns out that Verilog is a nice language to write real-valued models in.

Verilog Model for RC Filter

```
real Vin;              // input voltage
real Ir;               // resistor current
real Vout;             // output voltage
real lsttim;           // last time model was evaluated
initial begin
  Vout = 0.0;
  lsttim = 0.0;
end
Ir = (Vin – Vout) / R;                          // resistor model
Vout = Vout + Ir * ($abstime – lsttim) / C;// cap model
lsttim = $realtime;                             // save time
```



6

Bill Ellersick
Analog Circuit Works

The RC filter is easily modeled in Verilog, with the analog input voltage represented by real input values, and the resistor current and output voltage represented by internal real values.
After initializing the values, the current in the resistor is computed, and the change in the capacitor voltage is equal to the resistor current times the time interval divided by the capacitance.

Real Behavioral Advantages

- Rapid simulation even for large SoCs
- Can easily model parasitic coupling
  - Inductors are simple too: ∫Idt becomes IΔt
- No need for iterative analog SPICE engine
  - Simulate as fast as logic simulators; no convergence issues
- System/Verilog/A/AMS allow reals on ports
  - Portable across platforms/tools/companies/processes
  - Most ports transfer a voltage or a current: one real value
    - Falls short of true electrical modeling, but usually equivalent
  - Allows real interactions between models
  - Ports connections can really be verified
  - (and multiple reals can be surreptitiously passed)

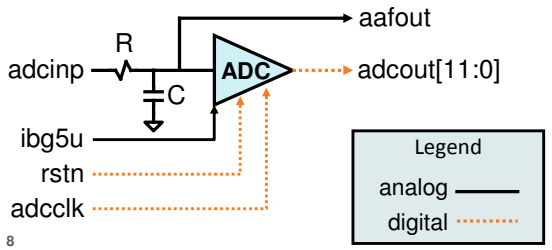
7

Bill Ellersick
Analog Circuit Works

The real-valued behavioral models have a number of advantages: they simulate rapidly enough to be used in top-level SoC simulations, and can even model parasitic coupling.
Even inductors can be modeled, with the integral of I dt becoming merely I times delta-t.
The real-valued models avoid the need for an iterative analog SPICE engine and the accompanying convergence issues, and the models simulate as fast as logic.
Most System Verilog, digital Verilog, Verilog-A and Verilog-AMS simulators now support real values on input and output ports, allowing models that are portable across tools and companies.
It turns out that most ports in mixed-signal circuits transfer either a voltage or a current, so only one real value is required. This falls short of true electrical modeling, but is usually equivalent, and allows real interactions between models using the actual connections between their ports. And there are techniques to pass multiple real values between models when necessary.

Verilog and Verilog-A

- Both Verilog and Verilog-A support 3 key features
  1. Coupling real or binary values in and out of module ports
     (electrical discipline in Verilog-A supports I and V in and out)
  2. Specifying when to update values
  3. Equations to update real and integer values
- Simple but comprehensive example (adcX)
  – ADC with input RC filter



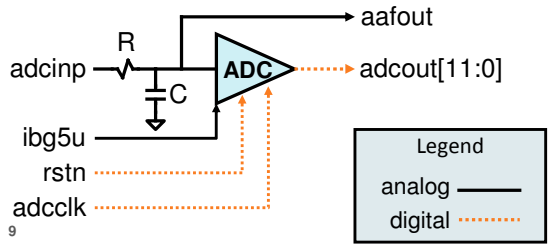
8

Bill Ellersick
Analog Circuit Works

This paper focuses on the differences between digital Verilog and Verilog-A, since SystemVerilog and Verilog-AMS are supersets of these languages.
Both of the languages support the 3 key features needed to create mixed signal models. They both allow coupling of real or binary values in and out of module ports, with the electrical discipline in Verilog-A supporting voltage and current through each port.
Both languages can specify when to update values in the models.
And the languages share common syntax for equations to update the real and integer values.
The modeling methodology is illustrated with a simple but comprehensive model of an ADC with an input RC filter that includes an analog input voltage to the RC filter, an analog output voltage from the RC filter, an ADC that samples and converts the input to a 12-bit digital output, an analog input current, and digital reset and clock inputs.

adcX Input/Output Declarations

(Almost Pure) Verilog	Verilog-A
wire [11:0] adcout;	electrical [11:0] adcout;
real aafout;	electrical aafout;
wire real ibg5u;	electrical ibg5u;
wire real adcinp;	electrical adcinp;
wire adcclk;	electrical adcclk;
wire rstn;	electrical rstn;



9

Bill Ellersick
Analog Circuit Works

Let's look at the Verilog mode code for this example. After declaring the module inputs and outputs, the Verilog model declares the digital outputs as wires, the analog output voltage as a real, the analog input voltage and current as wire real's, and the digital reset and clock inputs as wires. The Verilog-A model declares all of these as electrical and accesses the voltages or currents on the ports inside the models.

adcX RC Filter (discrete-time real)

Verilog

```
initial #1e-10 forever begin          // evaluate ~10 times highest freq
  ires = (adcinp - aafout_v) / Raaf; // resistor current
          // capacitor difference equation C = V + I * deltaT / C
  aafout_v = aafout_v + ires * ($realtime-lsttim) / Caaf;
  lsttim = $realtime;                 // save time
#1e-10; end
```

Verilog-A

```
@(timer(1e-10,1e-10)) begin          // evaluate ~10 times highest freq
  ires = (V(adcinp) - aafout_v) / Raaf;  // resistor current
          // capacitor difference equation C = V + I * deltaT / C
  aafout_v = aafout_v + ires * ($realtime-lsttim) / Caaf;
  lsttim = $realtime;                 // save time
end
```

- **Assignments identical; Inputs similar; when to update different**
- **No Verilog-A equivalent to always @(adcinp): periodic loop used**

10

Bill Ellersick
Analog Circuit Works

The core of a real-valued RC filter model is quite similar in Verilog and Verilog-A, with differences in the syntax that specifies when to update the model values. In Verilog, an initial delay and a forever loop with a delay at the end to set the period is used, while the Verilog-A model simply uses the timer function with delay and period inputs. In addition, the Verilog-A model specifies V(adcinp) to access the input voltage, while the Verilog model directly accesses the real input value. Notice how similar the model code is for these two languages which most of us view as incompatible. This is largely due to the discrete-time modeling method, which is supported in both languages but is not the norm in Verilog-A.

adcX ADC Model Excerpts

- Strikingly similar code: due to discrete-time approach

Verilog

```
always @(posedge adcclk or negedge rstn) begin
    vadc = adcinp-Voff+refv0;    // sample input, subtract Voff-refv0
    for(i=11; i>0; i=i-1) begin
     if(vadc > refv) begin
       adcout_int[i] = 1;       // output logic high if vadc > refv
       vadc = vadc - refv;      // and subtract refv from vadc
     end else begin
       adcout_int[i] = 0;       // else output logic low
```

Verilog-A

```
@(cross(V(adcclk)-(vdd_v+vss_v)/2.0, 1)
    vadc = V(adcinp)-Voff+refv0; // sample input, subtract Voff-refv0
    generate i(11,0) begin
     if(vadc > refv) begin
       adcout_int[i] = 1;       // output logic high if vadc > refv
       vadc = vadc - refv;      // and subtract refv from vadc
     end else begin
       adcout_int[i] = 0;       // else output logic low
```

11

Bill Ellersick
Analog Circuit Works

The model code for the ADC is also strikingly similar, again primarily with differences in the specification of when to update the model values. The Verilog model uses the @posedge construct, while the Verilog-A model uses the cross() function to detect when the input clock voltage rises past mid-supply. A for loop is used in Verilog to set integers to 0 or 1 to represent the 12 output bits, while in Verilog-A the generate function is used. Note that integers are used to represent logic values inside the models, since Verilog-A doesn't support boolean values.

adcX Output Assignments

Verilog

```
assign adcout[0] = adcout_int[0];
  …
assign adcout[11] = adcout_int[11];
always @( aafout_v ) aafout =  aafout_v;
```

Verilog-A

```
V(adcout[0]) <+ transition((adcout_int[0] ? vdd_v:vss_v),100e-12);
  …
V(adcout[11]) <+ transition((adcout_int[11] ? vdd_v:vss_v),100e-12);
V(aafout) <+  aafout_v;
```

12

Bill Ellersick
Analog Circuit Works

Finally, the model outputs are assigned based on the internal model values. In Verilog, logic outputs are directly assigned, while in Verilog-A, the transition filter is used to set logic output voltages to Vdd when the internal integer values are high, and to Vss when they are low. The analog output voltage is directly assigned in Verilog when the internal value changes, and in Verilog-A the <+ contribution assignment is used.

Real Portable Behavioral Models

- Mixed signal simulators continue to be developed
  - Remain slow, expensive, finicky
- Instead, use discrete-time features common to Verilog/A
  - Common code to manipulate reals and integers
  - Use powerful `define to write portable models
- Portable models can be verified w/analog simulator
  - Then can confidently use in digital sims at break-neck speeds
- Tested on Verilog-A,Verilog,Verilog-AMS,SystemVerilog
  - Ideally, standards and simulator syntax will converge
  - In the meantime, we can benefit from existing similarities
- Real portable testbenches for Verilog system simulation
  - replace higher level sim. languages; produce accurate models

13

Bill Ellersick
Analog Circuit Works

Since mixed signal simulators remain slow, expensive and temperamental, the models presented in this paper rely on features common to Verilog and Verilog-A, using common code to manipulate real and integer values, and taking advantage of the powerful `define construct to bridge the differences in syntax.
This allows portable models that can be verified in the analog simulation environment and then confidently used in digital simulations to rapidly and accurately verify SoCs.
The models in this paper have been verified on Verilog-A, Verilog, Verilog-AMS and SystemVerilog simulators. Ideally, standards and simulator syntax will converge, but in the meantime, we can benefit from the existing similarities.
System simulation can also benefit from portable models and testbenches, which can replace higher level simulation languages such as Matlab while producing accurate models to guide mixed signal circuit implementation.

Include File for Portable Models

Verilog

```
`define AnalogInput wire real
`define AnalogOutput real
`define LogicInput wire
`define LogicOutput wire
`timescale 1s/1fs     // timescale 1s to match Verilog-A
`define AnalogAssign(sig,val) always @(val) sig = val
`define LogicAssign(sig,val) assign sig = val

`define Posedge(sig) posedge sig

`define Negedge(sig) negedge sig

`define True(sig) (sig)       // logic signal wire in Verilog
`define AtTimerBegin(dly,period) initial #dly forever
    begin
`define TimerEnd(period) #period; end
`define V(sig) sig            // V(sig) real value in Verilog
`define I(sig) sig            // I(sig) real value in Verilog
`define For(i,i1,i2) for(i=i1; (i1<i2?i<=i2:i>=i2);
    i=(i1<i2?i+1:i-1))
```

Verilog-A

```
`define AnalogInput electrical
`define AnalogOutput electrical
`define LogicInput electrical
`define LogicOutput electrical

`define AnalogAssign(sig,val) sig <+ val
`define LogicAssign(sig,val) sig <+ transition((val ?
      vdd_v : vss_v), `_Tdelay, `_Trise, `_Tfall)
`define Posedge(sig) cross(V(sig)-
      (vdd_v+vss_v)/2.0, 1)
`define Negedge(sig) cross(V(sig)-
      (vdd_v+vss_v)/2.0, -1)
`define True(sig) (V(sig)>(vdd_v+vss_v)/2.0) ? 1 : 0)
`define AtTimerBegin(dly,period)
      @(timer(dly,period)) begin
`define TimerEnd(period) end
`define V(sig) V(sig)
`define I(sig) I(sig)
`define For(i,i1,i2) generate i(i1,i2)
```

Bill Ellersick
Analog Circuit Works

14

The portable models are enabled by an include file that contains a couple dozen `define macros to generate syntax specific to the target simulator. These `define's also illustrate the relatively minor syntax differences between the languages, and can inform enhancements to standards and simulators aimed at improving portability of models and testbenches.
In Verilog, analog inputs are wire real's, analog outputs are real's, while logic inputs and outputs are wires. All of these are declared as electrical in Verilog-A.
In Verilog, a timescale with units of 1 second is used to match the timescale in Verilog-A.
Analog outputs are assigned in Verilog with the always @ construct, and with the <+ contribution operator in Verilog-A.
Logic outputs are directly assigned in Verilog, but use the transition filter in Verilog-A to drive the output voltage to either Vdd or Vss.
Edges of logic inputs are detected with posedge and negedge in Verilog, while in Verilog-A the cross() function is used to detect when the input voltage crosses mid-supply.
The `True macro allows the state of logic inputs to be tested, which is trivial in Verilog but requires a comparison to mid-supply in Verilog-A.
The most complex of these simple `define macros generate a periodic event: in Verilog with an initial delay followed by a forever loop with a delay at the end to set the period of the loop, while in Verilog-A the timer() function is simply used with delay and period inputs.
To access input voltages or currents, the real input values are directly accessed in Verilog, while the V() and I() functions are used in Verilog-A.
And finally, a loop in Verilog-A uses the generate() function that either increments or decrements depending on whether the initial value is large or smaller than the final value, while in Verilog explicit code is used to compare the initial and final values to determine if the loop should increment or decrement.
Remarkably, this one slide shows all the syntax translation necessary to produce models that are portable from Verilog to Verilog-A.

Portable Model Excerpts for adcX

```
`LogicOutput [11:0] adcout;
`AnalogOutput aafout;
`AnalogInput ibg5u;
`AnalogInput adcinp;
`LogicInput adcclk;
`LogicInput rstn;
`AtTimerBegin(1e-10,1e-10)          // evaluate ~10 times highest freq
   ires = (`V(adcinp) - aafout_v) / Raaf;      // resistor current
              // capacitor difference equation C = V + I * deltaT / C
   aafout_v = aafout_v + ires * ($realtime-lsttim) / Caaf;
   lsttim = $realtime;                        // save time
`TimerEnd(1e-10)
`Always @(`Posedge(adcclk) or `Negedge(rstn)) begin
 `For(i,11,0) begin
      if(vadc > refv) begin
        adcout_int[i] = 1;             // output logic high if vadc > refv
        vadc = vadc - refv;             // and subtract refv from vadc
      end else begin
        adcout_int[i] = 0;             // else output logic low
 end end
 `LogicAssign(adcout[0],adcout_int[0]);
      …
 `LogicAssign(adcout[11],adcout_int[11]);
 `AnalogAssign(`V(aafout), aafout_v);
```

15

Bill Ellersick
Analog Circuit Works

This slide shows the portable model code for the ADC example that uses the `define macro's to generate syntax for the target simulator.
Ports need merely to be identified as logic or analog inputs or outputs.
A periodic loop for the RC filter is easily set up with the `AtTimeBegin and `TimerEnd macros with delay and period arguments.
The equations for updating the internal model variables are directly portable, except for access to input voltages and currents using the `V() and `I() macros.
Similarly, a loop that executes on the rising edge of the input clock relies on the `Posedge and `Negedge macros, and the iterative loop to generate the 12 output bits uses the `For macro, but the equations are directly portable.
Finally, the outputs assignments are accomplished with either `LogicAssign or `AnalogAssign

Verification and Debugging

- Portable models simplify analog verification
  - Verify portable model matches transistor-level design
  - Accelerate transistor-level integration simulations
- Portable testbenches are possible
  - Reuse configuration/analysis, enable code coverage
- Debugging portable Verilog can be challenging
  - Single-line `define's simplify error messages
  - Simulate pre-processed code to debug
- Complete adcX example with testbench is online
  - www.analogcircuitworks.com

16

Bill Ellersick
Analog Circuit Works

There are clear advantages to portable models, particularly in that they can be verified in the analog simulation environment to ensure that they match the transistor-level behavior. They also accelerate transistor-level integration simulations.
Portable testbenches can be developed with the same methodology, allowing reuse of configuration and analysis code, and enabling code coverage tools to ensure verification quality.
The debugging of portable code can be challenging, since simulator errors refer to input code, while the errors may be due to code in the `define macros. When the cause of the errors is not clear, it can be useful to generate pre-processed code as input to the simulator so that errors are more clearly identified.
If you are interested in the methodology presented in this paper, the complete adcX example with testbench and Makefile is posted on our website.

Conclusions

- Standards-based modeling/simulation
  - Portable, accurate, efficient
  - Proven discrete-time approach used for years in Matlab, etc.
  - Real variables connected through ports in design hierarchy
  - Portable models verified with analog testbenches
- High-level code in models and testbenches
  - Improves productivity and portability
  - Mixed signal and SoC and tool designers benefit
- Standards and simulator improvements
  - Example shows how easily common syntax could be supported
  - Low-end simulator supports portable models already
    - High-end require mixed-signal options for real inputs/outputs
  - Add discrete-time capabilities to Verilog-A (not SPICE->Verilog)

17

Bill Ellersick
Analog Circuit Works

In conclusion, a standards-based mixed-signal modeling methodology has been presented that is portable, accurate, and simulates rapidly. Discrete-time real variables are passed through the ports of models that are verified against transistor-level designs in the analog simulation environment.

The use of high-level Verilog code in the models and testbenches enhances productivity, benefitting almost everyone in this room.

An interesting side effect of this paper is the highlighting of syntax differences between Verilog-based simulators, and the ease with which the syntax differences can be bridged. Surprisingly, a low-end Verilog simulator was just as capable of simulating the mixed-signal models in this paper as high-end simulators, which required Verilog-AMS or SystemVerilog options to couple real values in and out of module ports.

We advocate enhancing the discrete-time capabilities in Verilog-A, rather than improving analog SPICE capabilities in Verilog. I hope to be able to report next year that simulators have adopted more common syntax and that the `define's in this paper are no longer needed.