Correct Methods For Adding Delays To Verilog Behavioral Models

Clifford E. Cummings
Sunburst Design, Inc.
15870 SW Breccia Drive
Beaverton, OR 97007
cliffc@sunburst-design.com

Abstract

Design engineers frequently build Verilog models with behavioral delays. Most hardware description languages permit a wide variety of delay coding styles but very few of the permitted coding styles actually model realistic hardware delays. Some of the most common delay modeling styles are very poor representations of real hardware. This paper examines commonly used delay modeling styles and indicates which styles behave like real hardware, and which do not.

1.0 Introduction

One of the most common behavioral Verilog coding styles used to model combinational logic is to place delays to the left of blocking procedural assignments inside of an always block. This coding style is flawed as it can either easily produce the wrong output value or can propagate inputs to an output in less time than permitted by the model specifications.

This paper details delay-modeling styles using continuous assignments with delays, and procedural assignments using blocking and nonblocking assignments with delays on either side of the assignment operator.

To help understand delay modeling, the next section also includes a short description on inertial and transport delays, and Verilog command line switches that are commonly used to simulate a model that is neither a fully inertial-delay model nor a fully transport-delay model.

2.0 Inertial and transport delay modeling

Inertial delay models only propagate signals to an output after the input signals have remained unchanged (been stable) for a time period equal to or greater than the propagation delay of the model. If the time between two input changes is shorter than a procedural assignment delay, a continuous assignment delay, or gate delay, a previously scheduled but unrealized output event is replaced with a newly scheduled output event.

Transport delay models propagate all signals to an output after any input signals change. Scheduled output value changes are queued for *transport delay* models.

Reject & Error delay models propagate all signals that are greater than the error setting, propagate unknown values for signals that fall between the reject & error settings, and do not propagate signals that fall below the reject setting.

For most Verilog simulators, reject and error settings are specified as a percentage of propagation delay in multiples of 10%.

Pure *inertial delay* example using reject/error switches.
Add the Verilog command line options:

```
+pulse_r/100 +pulse_e/100
```

reject all pulses less than 100% of propagation delay.

Pure *transport delay* example using reject/error switches.
Add the Verilog command line options:

```
+pulse_r/0 +pulse_e/0
```

pass all pulses greater than 0% of propagation delay.

Semi-realistic delay example using reject/error switches.
Add the Verilog command line options:

```
+pulse_r/30 +pulse_e/70
```

reject pulses less than 30%, propagate unknowns for pulses between 30-70% and pass all pulses greater than 70% of propagation delay.

3.0 Blocking assignment delay models

Adding delays to the left-hand-side (LHS) or right-hand-side (RHS) of blocking assignments (as shown in Figure 1) to model combinational logic is very common among new and even experienced Verilog users, but the practice is flawed.

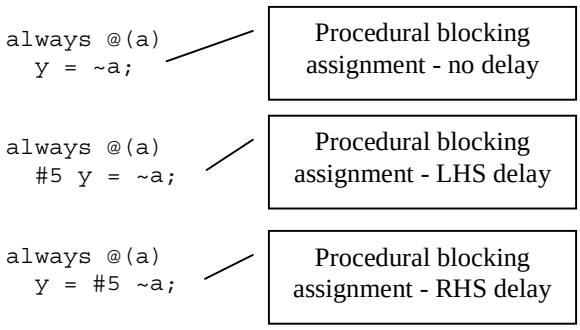

Figure 1 - Blocking Assignments with Delays

For the **adder_t1** example shown in Figure 2, the outputs should be updated 12ns after input changes. If the **a** input changes at time 15 as shown in Figure 3, then if the **a**, **b** and **ci** inputs all change during the next 9ns, the outputs will be updated with the latest values of **a**, **b** and **ci**. This modeling style has just permitted the **ci** input to propagate a value to the **sum** and **carry** outputs after only 3ns instead of the required 12ns propagation delay.

```
module adder_t1 (co, sum, a, b, ci);
  output       co;
  output [3:0] sum;
  input  [3:0] a, b;
  input        ci;
  reg          co;
  reg    [3:0] sum;

  always @(a or b or ci)
    #12 {co, sum} = a + b + ci;
endmodule
```

Figure 2 - LHS Blocking Assignment

Adding delays to the left hand side (LHS) of any sequence of blocking assignments to model combinational logic is also flawed.

The **adder_t7a** example shown in Figure 4 places the delay on the first blocking assignment and no delay on the second assignment. This will have the same flawed behavior as the **adder_t1** example.

The **adder_t7b** example, also shown in Figure 4, places the delay on the second blocking assignment and no delay on the first. This model will sample the inputs on the first input change and assign the outputs to a temporary location until after completion of the specified blocking delay. Then the outputs will be written with the old temporary output values that are no longer valid. Other input changes within the 12ns delay period will not be evaluated, which means old erroneous values will remain on the outputs until more input changes occur.

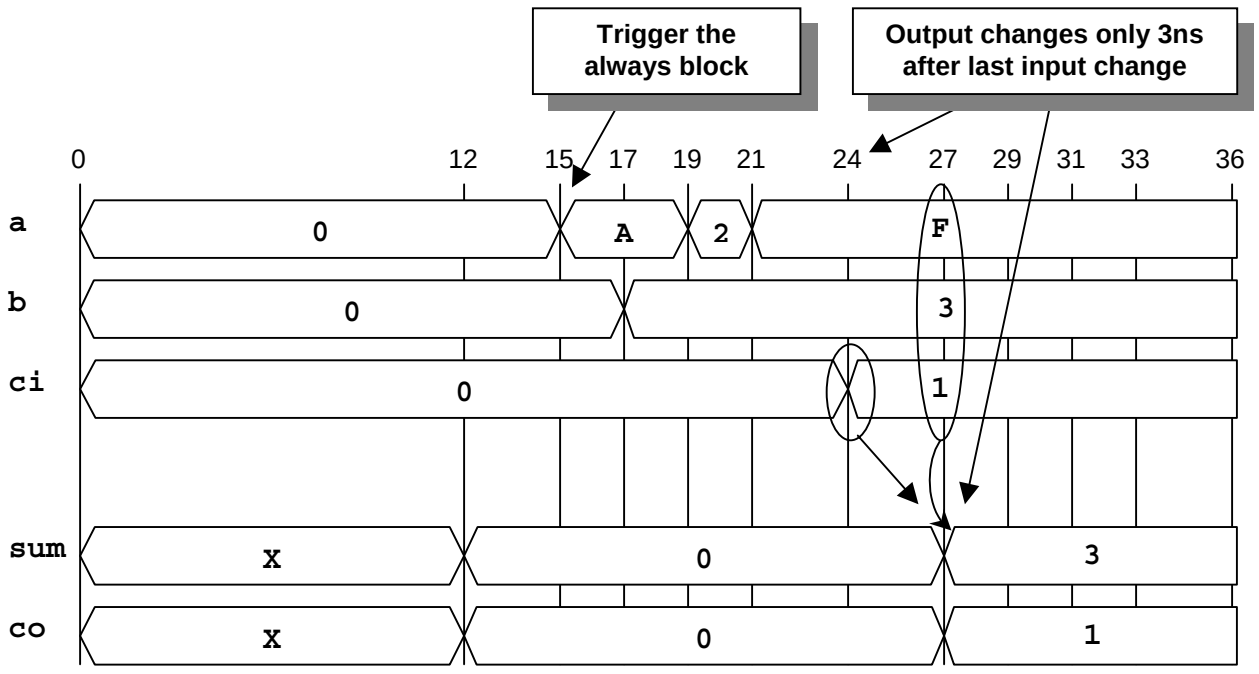

Figure 3 - Waveforms for adder_t1.v Example

These adders do not model any known hardware.

Modeling Guideline: do not place delays on the LHS of blocking assignments to model combinational logic. This is a bad coding style.

Testbench Guideline: placing delays on the LHS of blocking assignments in a testbench is reasonable since the delay is just being used to time-space sequential input stimulus events.

```
module adder_t7a (co, sum, a, b, ci);
  output       co;
  output [3:0] sum;
  input  [3:0] a, b;
  input        ci;
  reg          co;
  reg    [3:0] sum;
  reg    [4:0] tmp;

  always @(a or b or ci) begin
    #12 tmp       = a + b + ci;
        {co, sum} = tmp;
  end
endmodule


module adder_t7b (co, sum, a, b, ci);
  output       co;
  output [3:0] sum;
  input  [3:0] a, b;
  input        ci;
  reg          co;
  reg    [3:0] sum;
  reg    [4:0] tmp;

  always @(a or b or ci) begin
        tmp       = a + b + ci;
    #12 {co, sum} = tmp;
  end
endmodule
```

Figure 4 - Multiple LHS Blocking Assignments

3.1 RHS blocking delays

Adding delays to the right hand side (RHS) of blocking assignments to model combinational logic is also flawed.

```
module adder_t6 (co, sum, a, b, ci);
  output       co;
  output [3:0] sum;
  input  [3:0] a, b;
  input        ci;
  reg          co;
  reg    [3:0] sum;

  always @(a or b or ci)
    {co, sum} = #12 a + b + ci;
endmodule
```

Figure 5 - RHS Blocking Assignment

For the **`adder_t6`** example shown in Figure 5, the outputs should be updated 12ns after input changes. If the **`a`** input changes at time 15, the RHS input values will be sampled and the outputs will be updated with the sampled value, while all other **`a`**, **`b`** and **`ci`** input changes during the next 12ns will not be evaluated. This means old erroneous values will remain on the outputs until more input changes occur.

The same problem exists with multiple blocking assignments when delays are placed on the RHS of the assignment statements. The **`adder_t11a`** and **`adder_t11b`** examples shown in Figure 6 demonstrate the same flawed behavior as the **`adder_t6`** example .

None of the adder examples with delays on the RHS of blocking assignments behave like any known hardware.

Modeling Guideline: do not place delays on the RHS of blocking assignments to model combinational logic. This is a bad coding style.

Testbench Guideline: do not place delays on the RHS of blocking assignments in a testbench.

General Guideline: placing a delay on the RHS of any blocking assignment is both confusing and a poor coding style. This Verilog coding practice should be avoided.

```
module adder_t11a (co, sum, a, b, ci);
  output       co;
  output [3:0] sum;
  input  [3:0] a, b;
  input        ci;
  reg          co;
  reg    [3:0] sum;
  reg    [4:0] tmp;

  always @(a or b or ci) begin
    tmp       = #12 a + b + ci;
    {co, sum} =     tmp;
  end
endmodule


module adder_t11b (co, sum, a, b, ci);
  output       co;
  output [3:0] sum;
  input  [3:0] a, b;
  input        ci;
  reg          co;
  reg    [3:0] sum;
  reg    [4:0] tmp;

  always @(a or b or ci) begin
    tmp       =     a + b + ci;
    {co, sum} = #12 tmp;
  end
endmodule
```

Figure 6 - Multiple RHS Blocking Assignments

4.0 Nonblocking assignment delay models

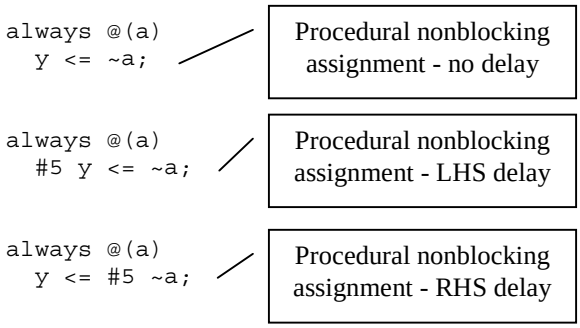

Figure 7 - Nonblocking Assignments with Delays

Adding delays to the left-hand-side (LHS) of nonblocking assignments (as shown in Figure 7) to model combinational logic is flawed.

The same problem exists in the **adder_t2** example shown in Figure 8 (nonblocking assignments) that existed in the **adder_t1** example shown in Figure 2 (blocking assignments). If the **a** input changes at time 15, then if the **a**, **b** and **ci** inputs all change during the next 9ns, the outputs will be updated with the latest values of **a**, **b** and **ci**. This modeling style permitted the **ci** input to propagate a value to the **sum** and **carry** outputs after only 3ns instead of the required 12ns propagation delay.

```
module adder_t2 (co, sum, a, b, ci);
  output       co;
  output [3:0] sum;
  input  [3:0] a, b;
  input        ci;
  reg          co;
  reg    [3:0] sum;

  always @(a or b or ci)
    #12 {co, sum} <=  a + b + ci;
endmodule
```

Figure 8 - LHS Nonblocking Assignment

It can similarly be shown that adding delays to the left hand side (LHS) of any sequence of nonblocking assignments to model combinational logic is also flawed.

Adders modeled with LHS nonblocking assignments do not model any known hardware.

Modeling Guideline: do not place delays on the LHS of nonblocking assignments to model combinational logic. This is a bad coding style.

Testbench Guideline: nonblocking assignments are less efficient to simulate than blocking assignments; therefore, in general, placing delays on the LHS of nonblocking assignments for either modeling or testbench generation is discouraged.

4.1 RHS nonblocking delays

Adding delays to the right hand side (RHS) of nonblocking assignments (as shown in Figure 9) will accurately model combinational logic with *transport delay*s.

In the **adder_t3** example shown in Figure 9, if the **a** input changes at time 15 as shown in Figure 10 (next page), then all inputs will be evaluated and new output values will be queued for assignment 12ns later. Immediately after the outputs have been queued (scheduled for future assignment) but not yet assigned, the always block will again be setup to trigger on the next input event. This means that all input events will queue new values to be placed on the outputs after a 12ns delay. This coding style models combinational logic with *transport delays*.

```
module adder_t3 (co, sum, a, b, ci);
  output       co;
  output [3:0] sum;
  input  [3:0] a, b;
  input        ci;
  reg          co;
  reg    [3:0] sum;

  always @(a or b or ci)
    {co, sum} <= #12 a + b + ci;
endmodule
```

Figure 9 - RHS Nonblocking Assignment

Recommended Application: Use this coding style to model behavioral delay-line logic.

Modeling Guideline: place delays on the RHS of nonblocking assignments only when trying to model transport output-propagation behavior. This coding style will accurately model delay lines and combinational logic with pure *transport delays*; however, this coding style generally causes slower simulations.

Testbench Guideline: This coding style is often used in testbenches when stimulus must be scheduled on future clock edges or after a set delay, while not blocking the assignment of subsequent stimulus events in the same procedural block.

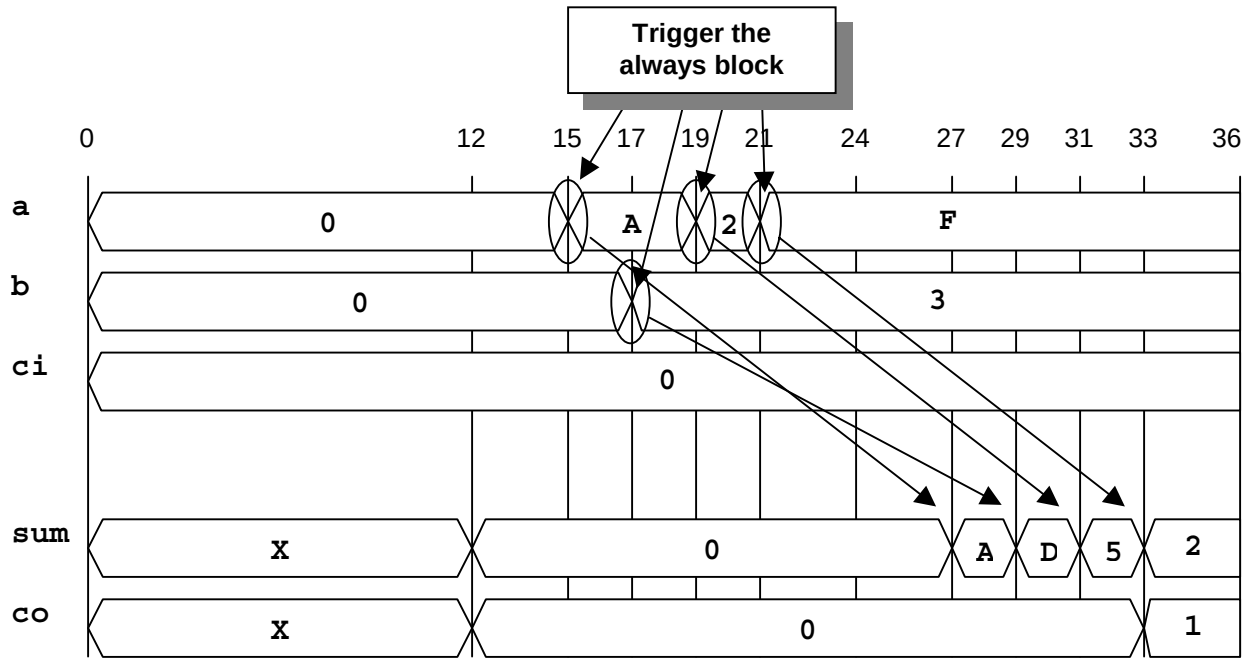

Figure 10 - Waveforms for adder_t3 Example

4.2 Multiple RHS nonblocking delays

Adding delays to the right hand side (RHS) of multiple sequential nonblocking assignments to model combinational logic is flawed, unless all of the RHS input identifiers are listed in the sensitivity list, including intermediate temporary values that are only assigned and used inside the always block, as shown in Figure 11.

For the **adder_t9c** and **adder_t9d** examples shown in Figure 11, the nonblocking assignments are executed in parallel and after **tmp** is updated, since **tmp** is in the sensitivity list, the always block will again be triggered, evaluate the RHS equations and update the LHS equations with the correct values (on the second pass through the always block).

Modeling Guideline: in general, do not place delays on the RHS of nonblocking assignments to model combinational logic. This coding style can be confusing and is not very simulation efficient. It is a common and sometimes useful practice to place delays on the RHS of nonblocking assignments to model clock-to-output behavior on sequential logic.

Testbench Guideline: there are some multi-clock design verification suites that benefit from using multiple nonblocking assignments with RHS delays; however, this coding style can be confusing, therefore placing delays on the RHS of nonblocking assignments in testbenches is not generally recommended.

```
module adder_t9c (co, sum, a, b, ci);
  output       co;
  output [3:0] sum;
  input  [3:0] a, b;
  input        ci;
  reg          co;
  reg    [3:0] sum;
  reg    [4:0] tmp;

  always @(a or b or ci or tmp) begin
    tmp       <= #12 a + b + ci;
    {co, sum} <=     tmp;
  end
endmodule


module adder_t9d (co, sum, a, b, ci);
  output       co;
  output [3:0] sum;
  input  [3:0] a, b;
  input        ci;
  reg          co;
  reg    [3:0] sum;
  reg    [4:0] tmp;

  always @(a or b or ci or tmp) begin
    tmp       <=     a + b + ci;
    {co, sum} <= #12 tmp;
  end
endmodule
```

Figure 11 - Multiple Nonblocking Assignments with Delays

5.0 Continuous assignment delay models

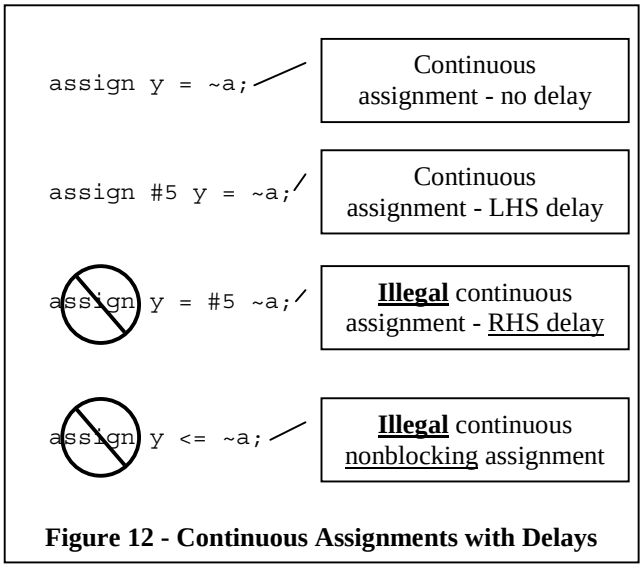

Figure 12 - Continuous Assignments with Delays

Adding delays to continuous assignments (as shown in Figure 12) accurately models combinational logic with *inertial delays* and is a recommended coding style.

For the **adder_t4** example shown in Figure 13, the outputs do not change until 12ns after the last input change (12ns after all inputs have been stable). Any sequence of input changes that occur less than 12ns apart will cause any future scheduled *output-event* (output value with corresponding assignment time) to be replaced with a new output-event.

```
module adder_t4 (co, sum, a, b, ci);
  output       co;
  output [3:0] sum;
  input  [3:0] a, b;
  input        ci;

  assign #12 {co, sum} = a + b + ci;
endmodule
```

Figure 13 - Continuous Assignment with Delay

Figure 14 shows the output waveforms for a simulation run on the **adder_t4** code shown in Figure 13. The first **a**-input change occurs at time 15, which causes an output event to be scheduled for time 27, but a change on the **b**-input and two more changes on the **a**-input at times 17, 19 and 21 respectively, cause three new output events to be scheduled. Only the last output event actually completes and the outputs are assigned at time 33.

Continuous assignments do not "queue up" output assignments, they only keep track of the next output value and when it will occur; therefore, continuous assignments model combinational logic with *inertial delays*.

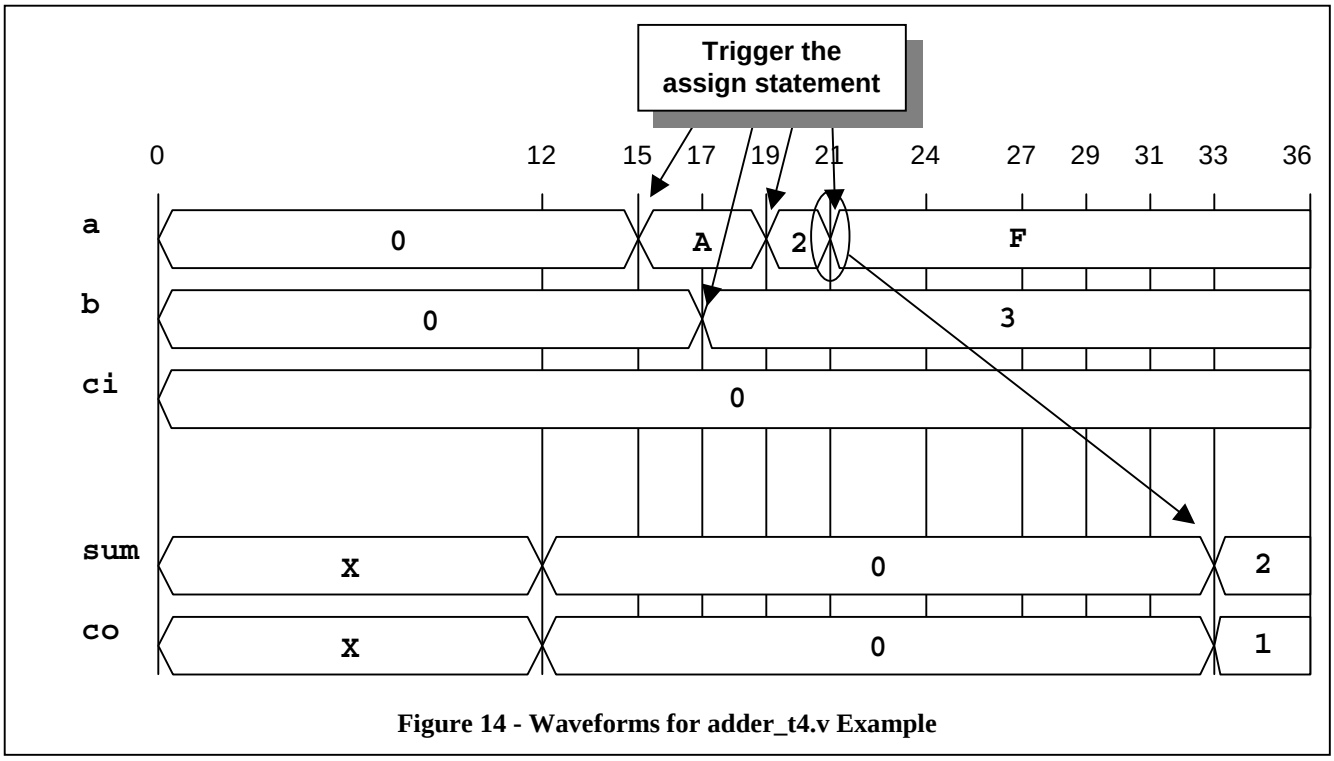

Figure 14 - Waveforms for adder_t4.v Example

5.1 Multiple continuous assignments

It can similarly be shown that modeling logic functionality by adding delays to continuous assignments, whose outputs are used to drive the inputs of other continuous assignments with delays, as shown in Figure 15, also accurately models combinational logic with *inertial delays*.

```
module adder_t10a (co, sum, a, b, ci);
  output       co;
  output [3:0] sum;
  input  [3:0] a, b;
  input        ci;
  wire   [4:0] tmp;

  assign     tmp       = a + b + ci;
  assign #12 {co, sum} = tmp;
endmodule

module adder_t10b (co, sum, a, b, ci);
  output       co;
  output [3:0] sum;
  input  [3:0] a, b;
  input        ci;
  wire   [4:0] tmp;

  assign #12 tmp       = a + b + ci;
  assign     {co, sum} = tmp;
endmodule
```

Figure 15 - Multiple Continuous Assignments

5.2 Mixed no-delay always blocks and continuous assignments

Modeling logic functionality in an always block with no delays, then passing the always block intermediate values to a continuous assignment with delays as, shown in Figure 16, will accurately model combinational logic with *inertial delays*.

```
module adder_t5 (co, sum, a, b, ci);
  output       co;
  output [3:0] sum;
  input  [3:0] a, b;
  input        ci;
  reg    [4:0] tmp;

  always @(a or b or ci) begin
    tmp = a + b + ci;
  end

  assign #12 {co, sum} = tmp;
endmodule
```

Figure 16 - No-Delay Always Block & Continuous Assignment

For the **`adder_t5`** example shown in Figure 16, the **`tmp`** variable is updated after any and all input events. The continuous assignment outputs do not change until 12ns after the last change on the **`tmp`** variable. Any sequence of always block input changes will cause **`tmp`** to change, which will cause a new output event on to be scheduled on the continuous assignment outputs. The continuous assignment outputs will not be updated until **`tmp`** remains unchanged for 12ns. This coding style models combinational logic with *inertial delays*.

Modeling Guideline: Use continuous assignments with delays to model simple combinational logic. This coding style will accurately model combinational logic with *inertial delays*.

Modeling Guideline: Use always blocks with no delays to model complex combinational logic that are more easily rendered using Verilog behavioral constructs such as "case-casez-casex", "if-else", etc. The outputs from the no-delay always blocks can be driven into continuous assignments to apply behavioral delays to the models. This coding style will accurately model complex combinational logic with *inertial delays*.

Testbench Guideline: Continuous assignments can be used anywhere in a testbench to drive stimulus values onto input ports and bi-directional ports of instantiated models.

6.0 Conclusions

Any delay added to statements inside of an always block does not accurately model the behavior of real hardware and should not be done. The one exception is to carefully add delays to the right hand side of nonblocking assignments, which will accurately model *transport delays*, generally at the cost of simulator performance.

Adding delays to any sequence of continuous assignments, or modeling complex logic with no delays inside of an always block and driving the always block outputs through continuous assignments with delays, both accurately model *inertial delays* and are recommended coding styles for modeling combinational logic.

Author & Contact Information

Cliff Cummings, President of Sunburst Design, Inc., is an independent EDA consultant and trainer with 19 years of ASIC, FPGA and system design experience and nine years of Verilog, synthesis and methodology training experience.

Mr. Cummings, a member of the IEEE 1364 Verilog Standards Group (VSG) since 1994, chaired the VSG Behavioral Task Force, which was charged with proposing enhancements to the Verilog language. Mr. Cummings is also a member of the IEEE Verilog Synthesis Interoperability Working Group.
Mr. Cummings holds a BSEE from Brigham Young University and an MSEE from Oregon State University.

E-mail Address: cliffc@sunburst-design.com
This paper can be downloaded from the web site:
www.sunburst-design.com/papers

(Data accurate as of March 7th, 2001)