Passive Device Verilog Models For Board And System-Level Digital Simulation

Clifford E. Cummings

Sunburst Design, Inc.
cliffc@sunburst-design.com

Anthony M. Nady II

anthony.m.nady@whiz.to

ABSTRACT

Board and system-level simulations require a simulation model for each component in the design. Simulation models for digital components are commercially available from LMC and Cadence, and can be readily created by in-house simulation-model groups, but some of the more difficult simulation models to acquire are the seemingly trivial passive devices that appear on a schematic. This paper details a bi-directional resistor model which is configurable from a schematic. Examples and benchmarks are included. This paper also details a configurable power supply model.

1. Introduction

While doing board-level simulation with LMC ECL simulation models at Tektronix, we experienced difficulties associated with our simple **`rtran`** resistor model. Specifically, it appeared we needed different resistor models for series-termination, parallel-termination and pull-up resistor configurations.

During subsequent board-level simulations using an LMC PLD device, we discovered potential problems with our simple VCC and GND models.

Our goal was to create passive device models that were efficient, versatile, and possessed intelligent default behavior.

2. The Resistor Model

Two problems that we had experienced with earlier design tools included:

- Unidirectional resistor models - design engineers object to directional schematic resistor symbols.
- Different models for different uses - design engineers do not want to use different schematic symbols to represent pullup and termination resistors.

To address the above problems, we set the following model goals:

- One Schematic symbol for all resistor configurations.
- Bi-directional symbol and model.
- Intelligent default setting.
- If possible, faster default setting option.

3. Driving LMC ECL Gates

Figure 1 shows an example of an LMC ECL model driven with four different input strengths: **`supply`**, **`strong`**, **`pull`** and **`weak`**, and Figure 2 shows the corresponding simulation results (using **`$monitor`** with **`%v`** control).

Note that at time l0ns, all schematic inputs are set to **`0`**, but the output strength of the **`weak`**-driven **`buf`** gates is driven to a **`Pu0`** strength.

Similarly at time 40ns, all schematic inputs are set to one, but the output strength of the **`weak`**-driven **`buf`** gates is driven to a **`Pu0`** strength, while the output strength of the pull-driven **`buf`** gates are now at **`PuX`** (due to the **`Pu0`** back-drive from the LMC ECL device, conflicting with the **`Pu1`** drive from the **`buf`** gates).

In order to successfully drive a **`1`** onto the LMC device, the input must at least be of **`strong`** drive strength. If the input were driven through a series-termination resistor modeled as an **`rtran`** device, only **`pull`** strength would be presented to the LMC device inputs.

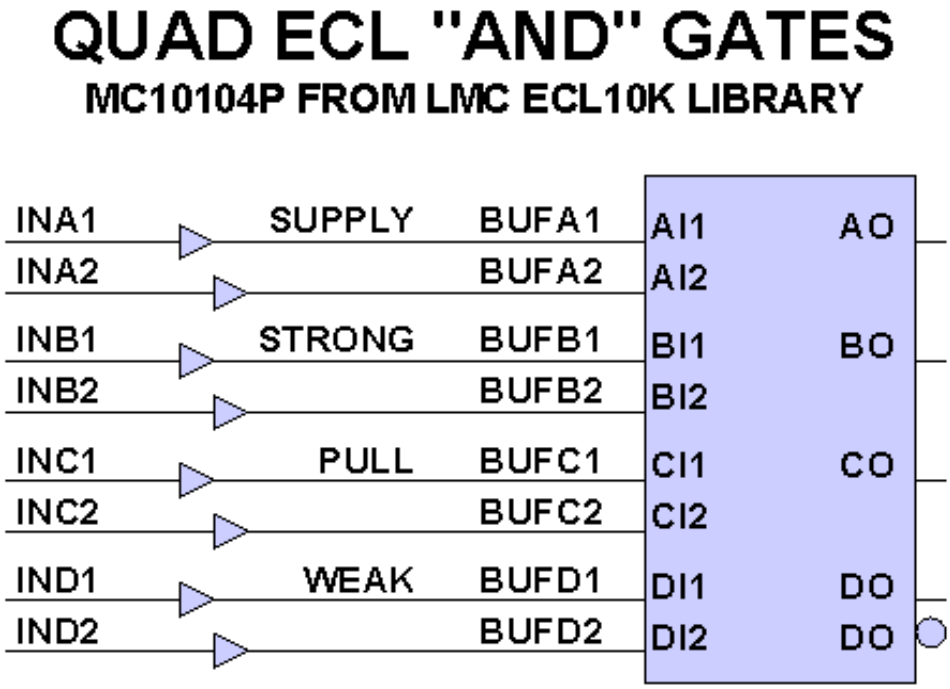


Figure 1 - LMC ECL model driven by different strengths

	INA PORTS	SUPPLY BUFFERS DRIVE	INB PORTS	STRONG BUFFERS DRIVE	INC PORTS	PULL BUFFERS DRIVE	IND PORTS	WEAK BUFFERS DRIVE
0.00ns	z&z	SuX&SuX	z&z	StX&StX	z&z	PuX&PuX	z&z	Pu0&Pu0
10.00ns	**0&0**	**Su0&Su0**	**0&0**	**St0&St0**	**0&0**	**Pu0&Pu0**	**0&0**	**Pu0&Pu0**
20.00ns	0&1	Su0&Su1	0&1	St0&St1	0&1	Pu0&PuX	0&1	Pu0&Pu0
30.00ns	1&0	Su1&Su0	1&0	St1&St0	1&0	PuX&Pu0	1&0	Pu0&Pu0
40.00ns	**1&1**	**Su1&Su1**	**1&1**	**St1&St1**	**1&1**	**PuX&PuX**	**1&1**	**Pu0&Pu0**
50.00ns	0&0	Su0&Su0	0&0	St0&St0	0&0	Pu0&Pu0	0&0	Pu0&Pu0

Figure 2 - LMC ECL input drive simulation results

4. Series Termination Resistors

Figure 3 shows the same LMC ECL device, driven through wires on the first set of inputs, through default resistors on the second set of inputs, and through resistors with attached **`STRENGTH=WIRE`** property on the third and fourth set of inputs.

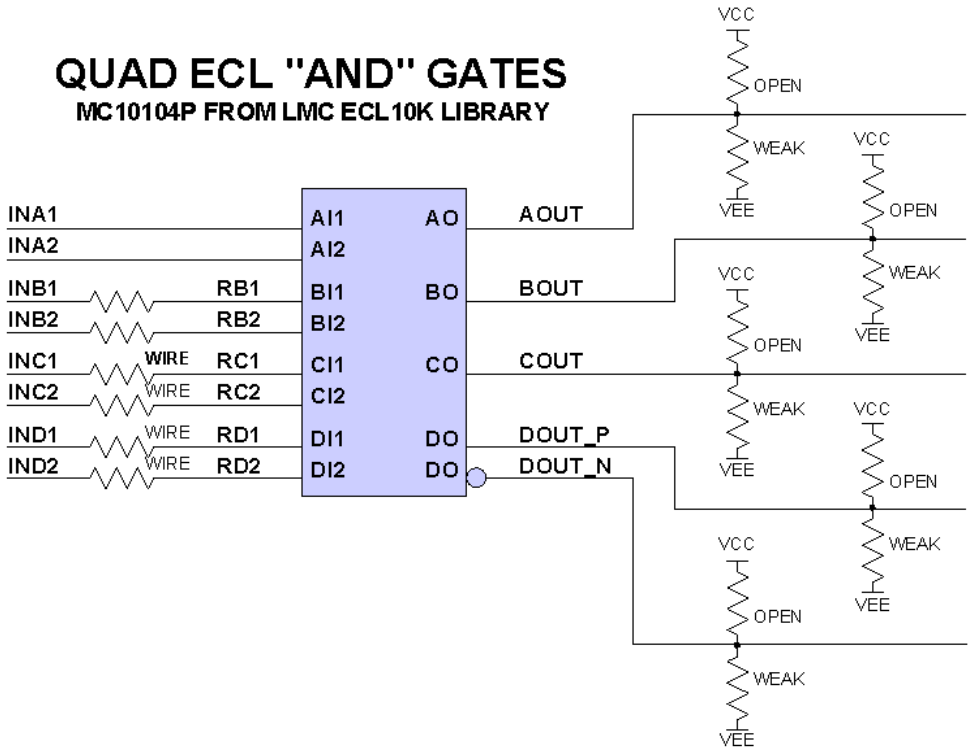


Figure 3 - LMC ECL model with series and parallel termination resistors

	INA	AOUT	INB	RB	BOUT	INC	RC	COUT	IND	RD	DOUT
0.00ns	z&z	StX	z&z	Pu0&Pu0	StX	z&z	Pu0&Pu0	StX	z&z	Pu0&Pu0	StX|StX
4.00ns	z&z	We0	z&z	Pu0&Pu0	We0	z&z	Pu0&Pu0	We0	z&z	Pu0&Pu0	We0|St1
10.00ns	0&0	We0	0&0	Pu0&Pu0	We0	0&0	St0&St0	We0	0&0	St0&St0	We0|St1
20.00ns	0&1	We0	0&1	Pu0&PuX	We0	0&1	St0&St1	We0	0&1	St0&St1	We0|St1
30.00ns	1&0	We0	1&0	PuX&Pu0	We0	1&0	St1&St0	We0	1&0	St1&St0	We0|St1
40.00ns	**1&1**	We0	**1&1**	**PuX&PuX**	We0	**1&1**	**St1&St1**	We0	**1&1**	**St1&St1**	We0|St1
44.00ns	**1&1**	**St1**	**1&1**	**PuX&PuX**	**StX**	**1&1**	**St1&St1**	**St1**	**1&1**	**St1&St1**	**St1|We0**
50.00ns	0&0	St1	0&0	Pu0&Pu0	StX	0&0	St0&St0	St1	0&0	St0&St0	St1|We0
54.00ns	0&0	We0	0&0	Pu0&Pu0	We0	0&0	St0&St0	We0	0&0	St0&St0	We0|St1

Figure 4 - Resistor driven simulation results

Figure 4 shows the corresponding simulation results (using **$monitor** with **%v** control).

At time 40ns, all schematic inputs are set to **1**, and after a 4ns propagation delay (time 44ns), the **A**, **C** and **D**-positive (**DOUT_P**) outputs are all driving a **Stl**, while the **B** AND-gate, whose inputs were driven through default resistors, has a **StX** at its output (note the **PuX&PuX** on the **B** AND-gate inputs).

The **STRENGTH=WIRE** property caused the resistor model to behave like a short circuit (**WIRE**) between the resistor terminals; thus, allowing a **strong** signal to pass directly to the LMC ECL input (and dominate the **Pu0** back-drive).

5. Parallel Termination Resistors

Also visible in the last example (Figure 3) is the technique for configuring parallel termination resistors.

If simple resistors, using only **rtran** Verilog primitives, had been used for the output parallel terminations, a **Pu1** strength from the pullup would conflict with a **Pu0** strength from the pulldown, and would generate a **PuX** any time the outputs are not driven with **strong** strength signals (which is the case when the LMC ECL outputs go low).

Thus parallel terminations also require unique treatment. Parallel terminations can frequently be modeled as open circuits using the **STRENGTH=OPEN** property attached to the appropriate resistors, or one of the terminations might be modeled using either the **STRENGTH=WEAK** or default **STRENGTH=PULL** resistor properties.

6. Four Different Resistor Models

We created and tested four different Verilog resistor models which could be modified by a per-instance schematic property, to emulate series- and parallel-terminations, while the default unmodified resistor model would behave like a pull-up resistor. All four models can be simplified by adding **+define+SETRPULL** to the Verilog command line. This directive, in conjunction with the coded **`ifdef** condition, will cause the model to be compiled with a simple **rtran** device, which is adequate for digital designs that only use pullup resistors.

6.1. r1.v and r2.v Verilog Models

The first two resistor models have an identical **WIRE** mode of operation. A **tranifl** gate is enabled if the **STRENGTH=WIRE** property is set, while the other two paths through the model are disabled.

The **PULL** mode is modeled either by using an **rtranifl** gate, or by using a **tranifl** in series with an **rtran** gate. Both provide one level of strength reduction.

The **WEAK** mode is modeled using the same idea as the **PULL** mode, except that one more **rtran** gate has been added to this path to provide two levels of strength reduction.

The **OPEN** mode is accomplished by disabling all three paths between the resistor ports.

6.2. r3.v Verilog Model

The **r3.v** models the **WIRE** mode by enabling all three **tranifl** gates, effectively shorting out the parallel **rtran** gates.

The **PULL** and **WEAK** modes are accomplished by turning off one or two of the enables respectfully, causing the signal flow to be directed through one or two **rtran** gates.

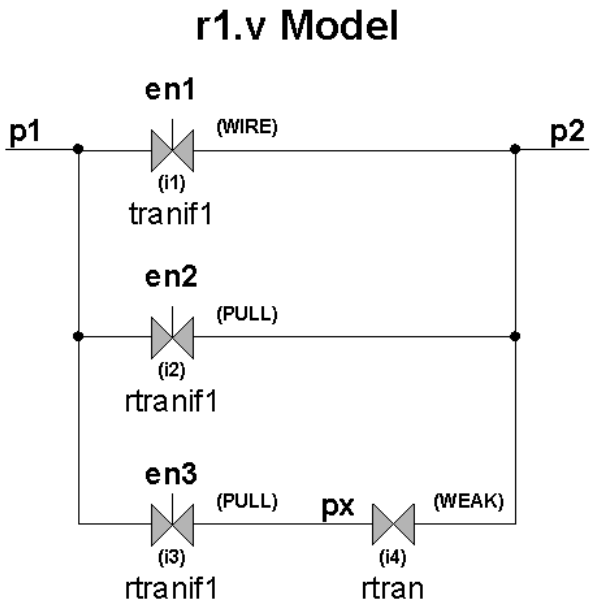
Just like the **r1.v** and **r2.v** models, the **OPEN** mode is accomplished by disabling all three **tranifl** gates; thereby creating an open circuit between the resistor ports.

6.3. r4.v Verilog Model

The **r4.v** model uses the same idea as the **rl.v** and **r2.v** models, only this time bi-directional **tran** gates are replaced by unidirectional gates.

This model was attempted at the suggestion of non-Cadence Verilog vendors, whose implementations did not yet support bi-directional Verilog switches. It was further suggested that the unidirectional gates would run much faster than the bi-directional gates.

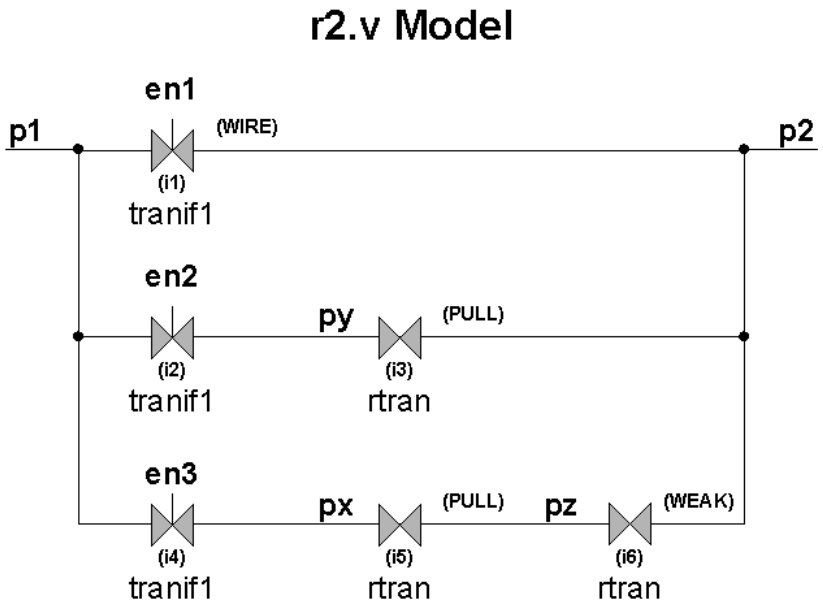
Note: unidirectional gates could not model the **WIRE** mode, since conflicting signals would drive both ports to a **StX** state.




```
// r1.v resistor model, bi-directional, configurable from schematic.
//
// Use Verilog command-line option +define+SETRPULL to
// speedup simulation which only use pullup resistors.
module r1 (p1, p2);
  inout p1, p2;
  parameter STRENGTH = "PULL";
  `ifdef SETRPULL
    rtran i2 (p1, p2);                          // PULL
  `else
    reg en1, en2, en3;
    tranif1  i1 (p1,p2,en1);                    // WIRE
    rtranif1 i2 (p1,p2,en2);                    // PULL
    rtranif1 i3 (p1,px,en3); rtran i4 (px,p2);  // WEAK

    initial case (STRENGTH)
      "WIRE"  :   {en1,en2,en3} = 3'b100;
      "PULL"  :   {en1,en2,en3} = 3'b010;
      "WEAK"  :   {en1,en2,en3} = 3'b001;
      "OPEN"  :   {en1,en2,en3} = 3'b000;
      default : begin
                  `ifdef NOMSG
                  `else
                    $display("%m: %s strength unknown, using PULL", STRENGTH);
                  `endif
                  {en1,en2,en3} = 3'b010;
                end
    endcase
  `endif
endmodule
```

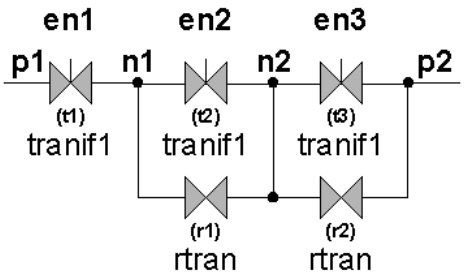
Figure 5 - The "r1" resistor model




```
// r2.v resistor model, bi-directional, configurable from schematic.
//
// Use Verilog command-line option +define+SETRPULL to
// speedup simulation which only use pullup resistors.
module r2 (p1, p2);
  inout p1, p2;
  parameter STRENGTH = "PULL";
  `ifdef SETRPULL
    rtran i2 (p1, p2);                          // PULL
  `else
    reg en1, en2, en3;
    tranif1  i1 (p1,p2,en1);                    // WIRE
    rtranif1 i2 (p1,py,en2); rtran i3 (py,p2);  // PULL
    rtranif1 i4 (p1,px,en3); rtran i5 (px,pz);  // WEAK
                             rtran i6 (pz,p2);
    initial case (STRENGTH)
      "WIRE"  :   {en1,en2,en3} = 3'b100;
      "PULL"  :   {en1,en2,en3} = 3'b010;
      "WEAK"  :   {en1,en2,en3} = 3'b001;
      "OPEN"  :   {en1,en2,en3} = 3'b000;
      default : begin
                  `ifdef NOMSG
                  `else
                    $display("%m: %s strength unknown, using PULL", STRENGTH);
                  `endif
                  {en1,en2,en3} = 3'b010;
                end
    endcase
  `endif
endmodule
```

Figure 6 - The "r2" resistor model

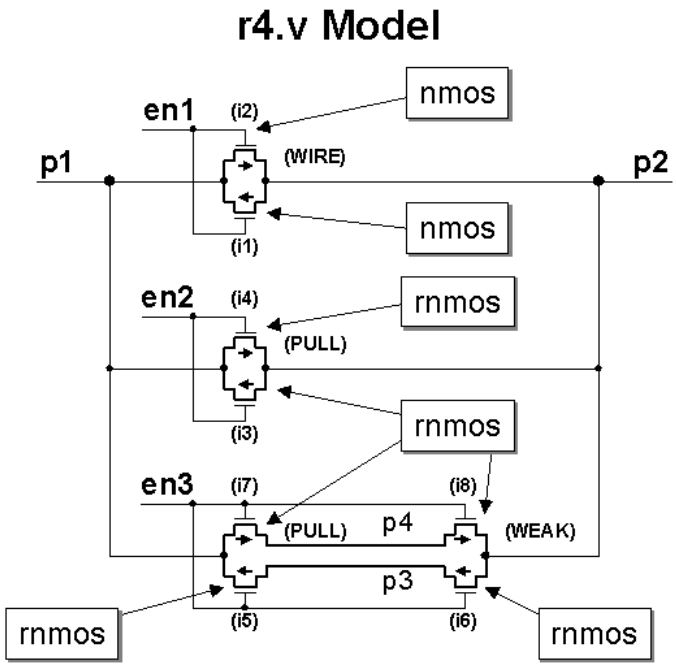
r3.v Model




```
// r3.v resistor model, bi-directional, configurable from schematic.
//
// Use Verilog command-line option +define+SETRPULL to
// speedup simulation which only use pullup resistors.
module r3 (p1, p2);
  inout p1, p2;
  parameter STRENGTH = "PULL";
  `ifdef SETRPULL
    rtran r1 (p1, p2);
  `else
    reg en1, en2, en3;
    tranif1  t1 (p1,n1,en1);
    tranif1  t2 (n1,n2,en2);
    tranif1  t3 (n2,p2,en3);
    rtran    r1 (n1,n2);
    rtran    r2 (n2,p2);

    initial case (STRENGTH)
      "WIRE"  :   {en1,en2,en3} = 3'b100;
      "PULL"  :   {en1,en2,en3} = 3'b010;
      "WEAK"  :   {en1,en2,en3} = 3'b001;
      "OPEN"  :   {en1,en2,en3} = 3'b000;
      default : begin
                  `ifdef NOMSG
                  `else
                    $display("%m: %s strength unknown, using PULL", STRENGTH);
                  `endif
                  {en1,en2,en3} = 3'b010;
                end
    endcase
  `endif
endmodule
```

Figure 7 - The "r3" resistor model




```
// r4.v resistor model, bi-directional, configurable from schematic.
//
// Use Verilog command-line option +define+SETRPULL to
// speedup simulation which only use pullup resistors.
module r4 (p1, p2);
  inout p1, p2;
  parameter STRENGTH = "PULL";
  `ifdef SETRPULL
    rnmos i3 (p1, p2, 1'b1); rnmos i4 (p2, p1, 1'b1); // PULL
  `else
    reg en1, en2, en3;
    nmos  i1 (p1, p2, en1); nmos  i2 (p2, p1, en1);  // WIRE
    rnmos i3 (p1, p2, en2); rnmos i4 (p2, p1, en2);  // PULL
    rnmos i5 (p1, p3, en3); rnmos i6 (p3, p2, en3);  // WEAK
    rnmos i7 (p2, p4, en3); rnmos i8 (p4, p1, en3);

    initial case (STRENGTH)
      "WIRE"  :   {en1,en2,en3} = 3'b100;
      "PULL"  :   {en1,en2,en3} = 3'b010;
      "WEAK"  :   {en1,en2,en3} = 3'b001;
      "OPEN"  :   {en1,en2,en3} = 3'b000;
      default : begin
                  `ifdef NOMSG
                  `else
                    $display("%m: %s strength unknown, using PULL", STRENGTH);
                  `endif
                  {en1,en2,en3} = 3'b010;
                end
    endcase
  `endif
endmodule
```

Figure 8 - The "r4" resistor model

7. Resistor Model Benchmarks

The four preceding resistor models were benchmarked by tying together the inputs of 2,000 resistors, and monitoring the 2,000 outputs for a few input transitions.

The 2,000-resistor simulations were conducted with attached **STRENGTH** properties, including a misspelled **STRENGTH** property (**ERR**), and with the Verilog command line **+SETRPULL** pullup default setting. The results are outlined in the table below.

Verilog-XL 1.6b - SPARCstation2						
Using r1.v model	**PULL**	**WIRE**	**WEAK**	**OPEN**	**ERR**	**+SETRPULL**
Memory usage (bytes)	11,957,460	11,957,460	11,957,460	11,957,460	11,957,452	4,222,404
Simulation events	46,033	44,033	46,033	22,033	46,033	14,033
Compile time (secs)	2.6	2.6	2.5	2.6	2.5	2.6
Link time (secs)	33.7	33.7	33.7	33.6	33.7	8.1
Simulation time (secs)	16.6	16.6	16.7	17.4	16.7	1.3
Total CPU Time (secs)	**52.9**	**52.9**	**52.9**	**53.6**	**52.9**	**12.0**
Using r2.v model	**PULL**	**WIRE**	**WEAK**	**OPEN**	**ERR**	**+SETRPULL**
Memory usage (bytes)	13,981,480	13,981,480	13,981,480	13,981,396	13,981,472	4,222,404
Simulation events	77,033	72,033	72,033	26,033	72,033	14,033
Compile time (secs)	2.5	2.5	2.5	2.6	2.6	2.6
Link time (secs)	53.6	53.7	53.6	53.6	53.6	8.1
Simulation time (secs)	32.2	32.1	32.2	32.6	32.1	1.2
Total CPU Time (secs)	**88.1**	**88.1**	**88.1**	**88.8**	**88.3**	**11.9**
Using r3.v model	**PULL**	**WIRE**	**WEAK**	**OPEN**	**ERR**	**+SETRPULL**
Memory usage (bytes)	12,645,480	12,645,480	12,645,480	12,645,460	12,645,472	4,222,404
Simulation events	56,033	54,033	58,033	24,033	56,033	14,033
Compile time (secs)	2.6	2.6	2.5	2.5	2.5	2.6
Link time (secs)	18.8	18.8	18.8	18.8	18.8	8.2
Simulation time (secs)	5.5	5.4	5.7	3.3	5.6	1.2
Total CPU Time (secs)	**26.9**	**26.8**	**27.0**	**24.6**	**26.0**	**12.0**
Using r4.v model	**PULL**	**WIRE**	**WEAK**	**OPEN**	**ERR**	**+SETRPULL**
Memory usage (bytes)	14,185,868	14,017,196	14,185,868	14,185,868	14,185,860	4,972,904
Simulation events	30,096	16,034	30,096	18,042	30,096	14,096
Accelerated events	68,059	24,053	94,059	30,059	68,059	33,225
Compile time (secs)	2.5	2.5	2.6	2.6	2.7	3.0
Link time (secs)	17.4	17.4	17.7	17.8	17.4	7.2
Simulation time (secs	203.4	51.5	203.0	74.3	219.1	55.8
Total CPU Time (secs)	**223.3**	**71.4**	**223.3**	**94.7**	**239.1**	**66.0**

Figure 9 - Benchmark simulation 2000 resistors

For overall speed, the **r3.v** resistor model was a clear winner, and used only 6% more memory than the **r1.v** resistor model.

The results of the benchmark testing produced two surprises:

1. We thought that the parallel path resistor models (**r1.v** and **r2.v**) would be more efficient than the **r3.v** model. Such was not the case.
2. Based on information received from non-Cadence Verilog vendors, we thought that the unidirectional switch resistor model might be more efficient than the bi-directional-switch counter parts. In fact, the benchmark demonstrated that the unidirectional-switch models performed much worse. We recognize that the unidirectional-switch models may perform much better with another vendor's version of Verilog.

We also attempted to run the benchmarks with the **+switchxl** Verilog command line switch, but this caused our memory usage to increase to the point where the benchmark tests appeared to be memory-disk swapping, and test-run times exceeded our patience level.

In every case, using the Verilog command line switch **+define+SETRPULL** achieved significant speed and memory utilization improvements; therefore, if a schematic design only uses pullup-type resistors, this switch can improve simulation efficiency by deleting unnecessary logic.

8. VCC and GND Models

There are also subtle decisions to be made when creating a model for power supplies. If VCC and GND are modeled with **supply** drive strengths, a design with a gate output tied to either VCC or GND may simulate without error or warning, but the problematic gate may bum-up on the actual circuit board.

Conversely, modeling VCC and GND with **strong** strengths poses different problems as we discovered with the following example.

This LMC PLD device is programmed to be a 2-bit up/down counter and NAND gate: with inputs, **GIN1**, **GIN2**, and output **GOUT**.

In this circuit, our GND model drove a **strong0** strength, and of the outputs of this PLD device were always unknown.

The grounded unused configurable **F2**, **F3**, **F6** and **F7** pins happened to be driving a **St1** against the **St0** **GND**, producing a **StX**, which was fed back to the **OE/Il9** input, and disabled all outputs.

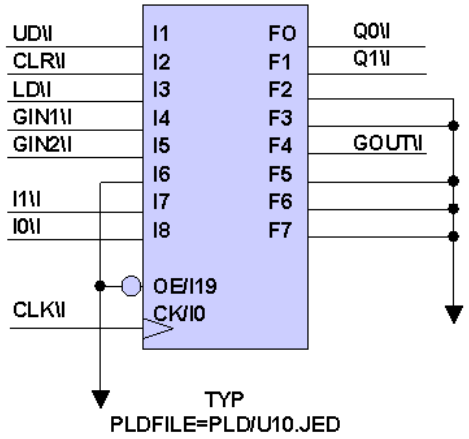


Figure 10 - "strong" strength GND model

The apparent, simple solution is to always drive **supply** strengths on power and ground symbols; however, in this case, the simulation would have passed, but the device would have failed on the prototype board.

For this reason, the Tektronix supply model still drives **strong** outputs by default, but can be reconfigured from the Verilog command line to drive **supply** strengths by using the **+define+SETSUPPLY0** command switch.

```
module supply(out);
  output out;
  reg    outreg;
  parameter COMPONENT = 0;
  `ifdef SETSUPPLY0
    wire (supply0, supply1) out = outreg;
  `else
    wire (strong0, strong1) out = outreg;
  `endif
  initial outreg = COMPONENT;
endmodule
```

Figure 11 - supply.v Verilog code

9. Conclusions

The resistor and power supply models outlined in this paper accomplished our overall goal of creating simplified Verilog models and schematic symbols with powerful configuration options.

Other passive device Verilog models have been written, using the same behavioral/structural model controlling techniques described in this paper.

10. Revision 1.1 (November 2004) - What Changed?

The original source file for this paper had been lost so the paper was re-captured and cleaned up.

Both authors were Tektronix employees at the time the paper was first released but both have moved to new companies.

11. Author & Contact Information

Cliff Cummings, President of Sunburst Design, Inc., is an independent EDA consultant and trainer with 22 years of ASIC, FPGA and system design experience and 12 years of Verilog, synthesis and methodology training experience.

Mr. Cummings, a member of the IEEE 1364 Verilog Standards Group (VSG) since 1994, is the only Verilog and SystemVerilog trainer to co-develop and co-author the IEEE 1364-1995 & IEEE 1364-2001 Verilog Standards, the IEEE 1364.1-2002 Verilog RTL Synthesis Standard and the Accellera SystemVerilog 3.0 & 3.1 Standards.

Mr. Cummings holds a BSEE from Brigham Young University and an MSEE from Oregon State University.

Sunburst Design, Inc. offers Verilog, Verilog Synthesis and SystemVerilog training courses. For more information, visit the www.sunburst-design.com web site.

Email address: cliffc@sunburst-design.com

Tony Nady, is an overall cool dude and expert design engineer. Tony's author information will be further updated in future versions of this paper.

Email address: anthony.m.nady@whiz.to

An updated version of this paper can be downloaded from the web site:

www.sunburst-design.com/papers

(Data accurate as of November 2nd, 2004)