
Tufts University

School of Engineering
Department of Computer and Electrical Engineering



EE103 - Introduction to VLSI Design

Fall 2013 — Dr. Denis Daly

Lab #3: Simulation of Logic with Verilog-XL

Due: Tuesday, November 12th, 2013

Name: Noah Kurinsky

Email: Noah.Kurinsky@tufts.edu

Date Submitted: November 12, 2013

Objective

The objective of this lab was to perform functional simulations of static and dynamic logic circuits in Cadence using the Verilog-XL simulation suite. In conjunction with the gate primitive simulations performed, the first part of the 8-bit carry look ahead adder was designed and simulated using only the primitive gates developed in this lab. The goal was to develop comfort and fluency with gate-level verilog, using it to model static and dynamic gates, and to illustrate the power of reusable gate modules.

Introduction

Verilog is a hardware description language (HDL) used to model digital logic at many levels, ranging from gate implementation (as it is used here) to purely functional implementation, which assumes a digital system comes with a handful of built in functions. A new component can be created as a module in Verilog, which can then be reused many times to build modular devices such as adders or buffers which are many bits wide. Verilog can also model at the transistor and CMOS level, and includes functions for simulating various types of CMOS connections and wires, from ideal (just on or off) to resistive versions of CMOS structures to wires which may retain their values or allow them to decay at characteristic rates.

In this lab, we make use of all of these functions. We stick to mostly gate-level modeling, and employ the built-in gate primitives to construct realistic gates with simple delays using the `#` identifier. This combined with the built in modules `nand`, `nor`, `xor` and `not`, which generalize to any number of inputs, allow us to construct gates with specific numbers of inputs and varying degrees of delay. The use of the `wire` and `reg` keywords allow for the connection of multiple components within a module. MOS and CMOS structures include `nmos`, `pmos`, `cmos`, `rnmos`, `rpmos`, and `rcmos`, each of which simulate the associated structure, either as resistive or as a switch (specified by the `r` at the beginning of the keyword). The modules generated in this lab are referred to as functional views, to differentiate them from the schematic and symbol views which are also regularly generated with cadence.

To simulate functional views, the Verilog-XL simulation suite is employed to feed specified inputs into a module and parse the verilog code in order to calculate the output signals produced by a module. The input pattern is specified through a “stimulus” file, which contains initialization for each input signal, commands to change signal value, and the delay between subsequent command executions. The stimulus file is also written in verilog, and uses the `#` sign to specify delay between commands. Individual bits can be assigned by setting a signal equal to a given bit value, where the syntax specifying a value is a bit is `1'b`. For example, assigning a bit with value 0 to variable `A` after 10 time units would look like `#10 A = 1'b0`; where the final 0 specifies the desired value; this command would exist on one line, subsequent commands on following lines. Verilog-XL parses these commands and creates input and output waveforms using the functional views it is simulating, and then stores them in a database, which we can then plot to test if the functional circuits operate as expected. Adding delay to our functional view gives us an easy way to estimate cumulative gate delay without having to run a full cadence simulation, as was done in the previous lab.

Procedure

There are three distinct sections of this lab, labeled parts I-III, which focus on different aspect of verilog gate modeling. In part I, basic gates are constructed and functional views are written to represent common logic gates; these are simulated using Verilog-XL to ensure desired functionality. In part II, the basic structure for the adder is developed, and the design is crudely simulated in verilog to illustrate whether the design functions as intended, and where there may be room for improvement in later implementation and cadence circuit simulation. In part III, a dynamic D-Latch is simulated with a leaky storage node, and the latch is

tested for decay times which are shorter and longer than a clock period to determine stability. The DLatch is then designed to be immune to the effects of leakage.

1 Part I

In this part, functional views were constructed for the Inverter, 2- and 3-Input NAND gates, 2- and 3-Inputs NOR gates, and the 2-Input XOR gate, and symbols were created for each gate to allow the use of the functional view in verilog simulations for circuits built with these gate primitives. The main purpose for implementing these gate primitives was to create gates which mimic the delay characteristics of actual CMOS gates, in order to give structures constructed from these gates somewhat accurate delay characteristics. Because the particular input and output structures are arbitrary, the delay implemented in these gates assumes a parasitic delay of 0 and an electric effort of 1, such that the total delay for each gate amounts to its logical effort alone. The logical efforts for the gates implemented in this lab can be seen in table 1.

	Inverter	NAND2	NOR2	NAND3	NOR3	XOR2
g	1	4/3	5/3	5/3	7/3	4

Table 1: Logical effort for CMOS gates sized for unit-transistor rise and fall resistances.

Each of the functional views were constructed from a delay equal to the logical effort for that gate, and the delay-less gate module standard to verilog, with all inputs and outputs specified. The functional views were then simulated with Verilog-XL to ensure that they computed the correct logical function, and were inspected to see how the delay affected the performance of the gate. For these one-gate primitives, where rise and fall time are identical, we expect only to see delays equal to those specified in the verilog, however we need to ensure that they function properly.

Three stimulus files were written to standardize the test input signals, one for one input (only the inverter), one for two inputs, and one for three inputs (shown in appendix A, sections 1, 2 and 3). The clock period for each stimulus signal was set to 20ns; the writeup specified 50ns, however the delays were not visible for the single gates for this large period, so I reduced the period to allow delays to be more pronounced. In the results section, the symbol and functional view for each gate is presented, and then the output from the simulation is shown along with the Verilog-XL simulation window output, which displays simulation event number and calculation time.

2 Part II

In this part, Nick Davis and I designed and simulated the 8-bit Look-Ahead Carry Adder, using the functional views written in Part I to construct a structure which could then be simulated in Verilog-XL. For more details on the procedure and results, see the joint report.

3 Part III

In this section of the lab, a functional view for a dynamic D-latch was constructed using a resistive CMOS transmission gate, two inverters, and a `triereg` link which simulates a leaky register node. The `triereg` module allows for the specification of both rising and falling delays as well as decay time, which is the parameter which controls the leakage rate of the node. During simulation, decay times of 2ns and 20ns were chosen to test versions of the latch which should be stable over a 10ns hold time versus those which should decay during the hold time. A symbol was also generated for this functional view. The inverters and the

CMOS gate were also given unit time delays as in Part I. After simulating the leaking dynamic latch, a static latch invulnerable to leakage was also designed, but not simulated.

Schematics and Transient Simulation Results

1 Part I: Gate Primitives

Here I present the functional views for the Inverter, NAND gates, NOR gates, and XOR gate, the symbols, and Verilog-XL simulation results.

1.1 Inverter

For the inverter, the verilog “not” module was employed, as seen in the functional view in figure 2. The delay was set to one nanosecond, as the logical effort for the inverter is one, and the inverter input is Vin with the output is Vout (these are labels inherited from the tutorial). The symbol for the inverter can be seen in figure 1.

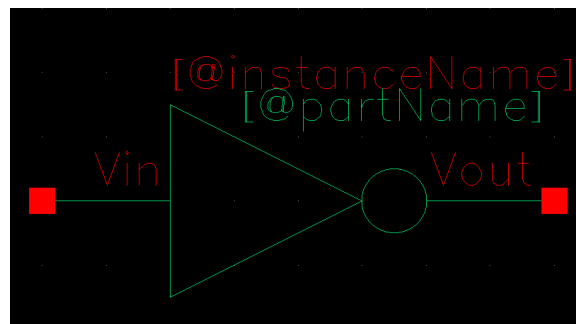


Figure 1: Schematic symbol associated with the inverter functional view developed in this lab.

```
module Inverter ( Vout , Vin );

    input Vin;
    output Vout;
    not #1 n1 ( Vout , Vin );

endmodule
```

Figure 2: Functional view for the inverter gate.

The inverter was simulated with Verilog-XL using the stimulus file in Appendix A1, and the output from the Verilog-XL window at the end of the simulation can be seen in figure 3. The output waveforms can be seen in figure 4. Features to note here are that, after 1ns, the input and output are indeed the inverted versions of each other, however the output is undefined for the first second of simulation and there is delay in signal propagation from the input to the output, as per the functional view design.

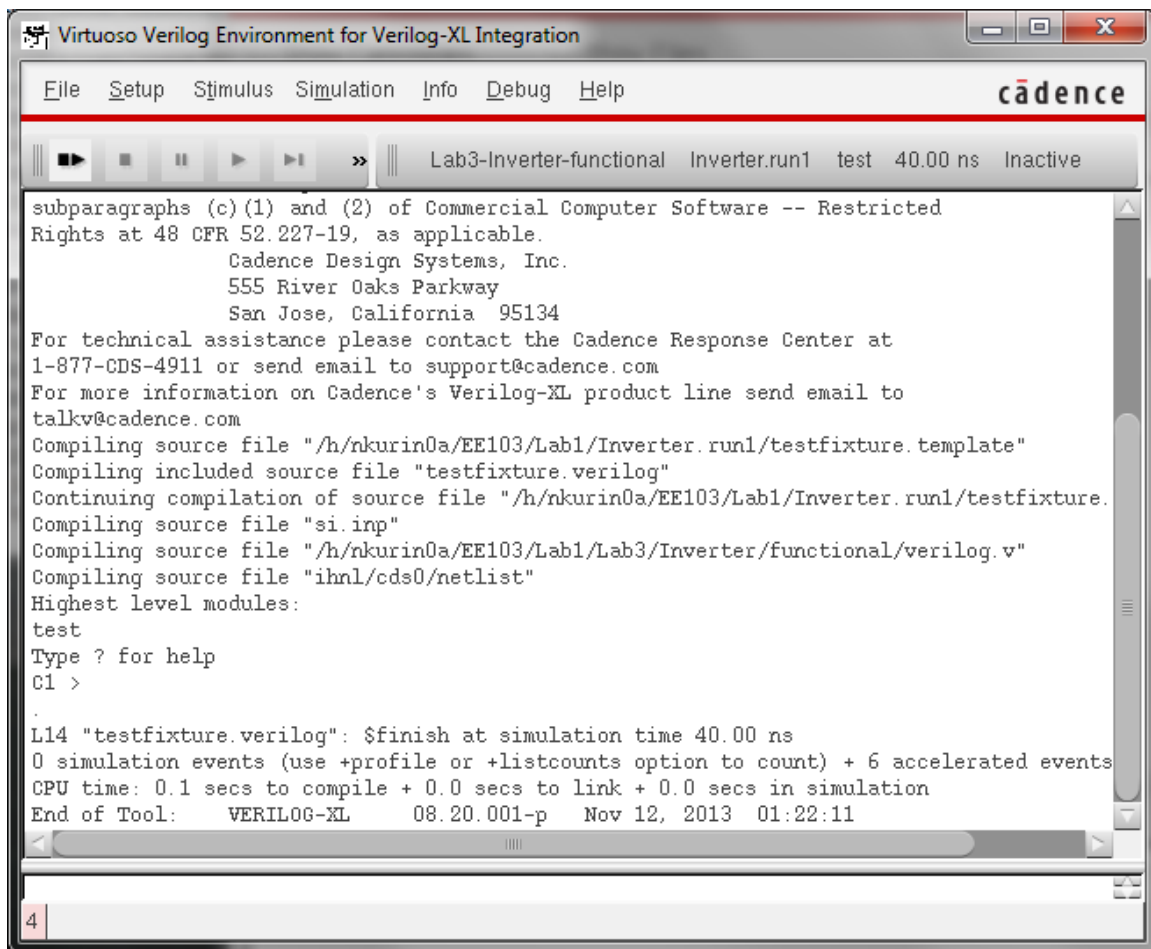


Figure 3: The output within the Verilog-XL simulation window for the inverter simulation, after running first running the initialization and final simulation.

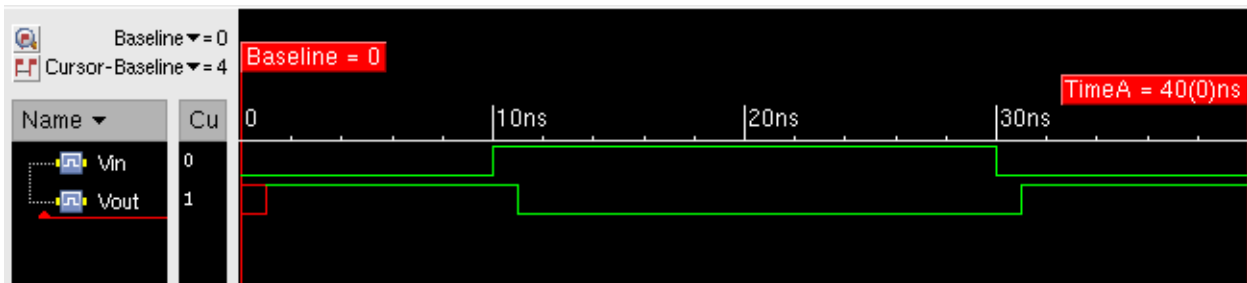


Figure 4: Output from the verilog inverter simulation of the verilog code shown above, cropped from the waveform viewer. Vin refers to the inverter input, Vout is the inverter output. Above, the 1 time unit delay can be seen in the difference between when Vin and Vout transition between values.

1.2 2-Input NAND Gate

For the 2-Input NAND gate, the verilog “nand” module was employed, as seen in the functional view in figure 6. The delay was set to 1.333ns, as the logical effort for the 2-Input NAND is 4/3. The NAND inputs are A and B, respectively, and the output is Y. The associated symbol for the NAND2 can be seen in figure 5.

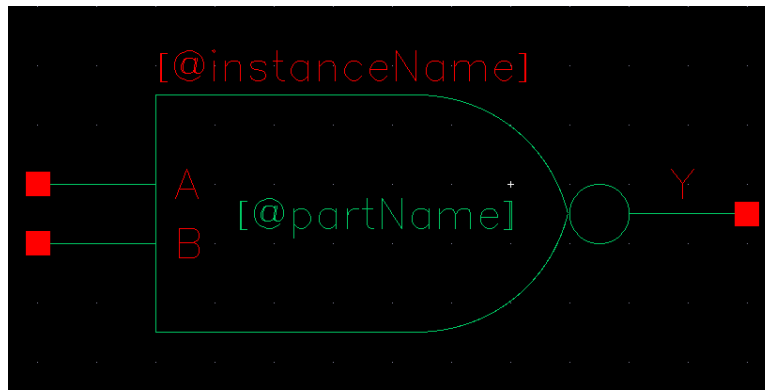


Figure 5: Schematic symbol associated with the 2-Input NAND functional view developed in this lab.

```

module NAND2 ( Y, A, B );

    output Y;
    input A;
    input B;
    nand #(1.333) n1(Y,A,B);

endmodule

```

Figure 6: Functional view for the NAND2 gate

The NAND2 was simulated with Verilog-XL using the stimulus file in Appendix A2 for a two-input gate, and the output from the Verilog-XL window at the end of the simulation can be seen in figure 7. The output waveforms can be seen in figure 8. Features here to note are that the delay of slightly more than one time unit can be seen across all transitions, and again the output is undefined at the very beginning of the simulation. That being said, the gate computes the correct output function and the output is consistently valid after the propagation delay.

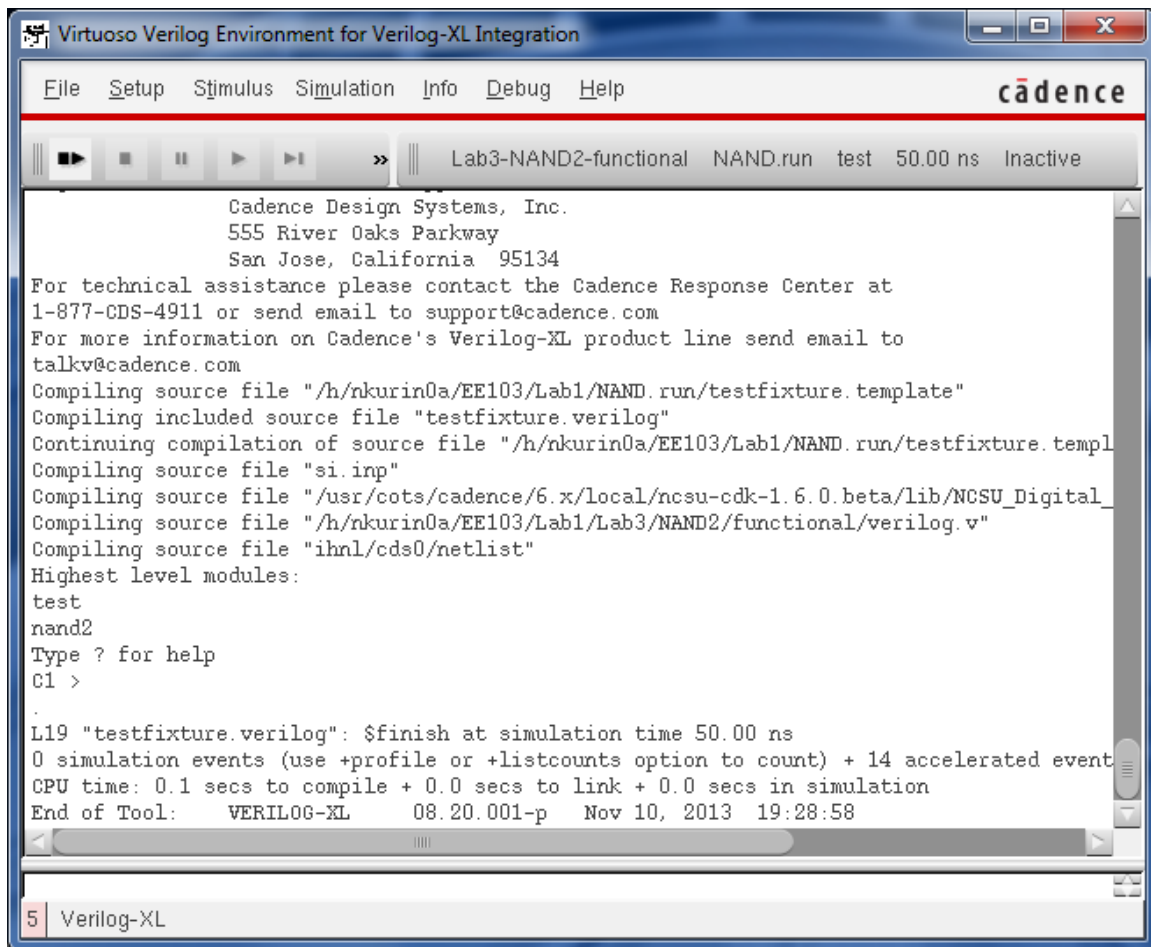


Figure 7: The output within the Verilog-XL simulation window for the NAND2 simulation, after running first running the initialization and final simulation.

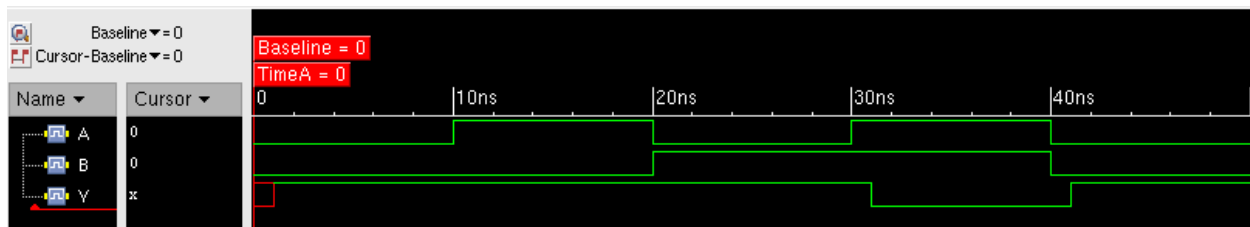


Figure 8: Output from the verilog 2-input nand gate simulation, with A and B the inputs set according to the two-input stimulus file, and Y being the NAND gate output. Here we see the delay at all transitions slightly higher than 1ns, but that all output is the correct NAND2 gate signal for times longer than the propagation delay.

1.3 3-Input NAND Gate

For the 3-Input NAND gate, the verilog “nand” module was employed, as seen in the functional view in figure 10. The delay was set to 1.666ns, as the logical effort for the 3-Input NAND is 5/3. The NAND inputs are A, B, and C, respectively, and the output is Y. The associated symbol for the NAND3 can be seen in figure 9.

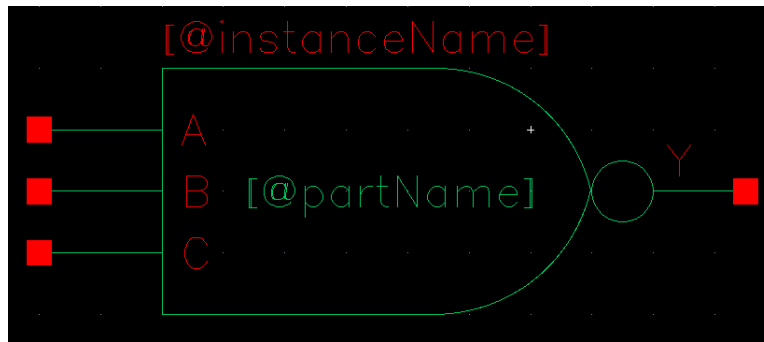


Figure 9: Schematic symbol associated with the 3-Input NAND functional view developed in this lab.

```

module NAND3 ( Y, A, B, C );

    input C;
    output Y;
    input A;
    input B;
    nand #(1.666) n1(Y,A,B,C);

endmodule

```

Figure 10: Functional view for the NAND3 gate.

The NAND3 was simulated with Verilog-XL using the stimulus file in Appendix A3 for a three-input gate, and the output from the Verilog-XL window at the end of the simulation can be seen in figure 11. The output waveforms can be seen in figure 12. Here, as in the previous simulations, the gate performs the correct logical computation after the propagation delay, which here is slightly higher than expected, at around 2 seconds; this may just be a function of delays needing to be in integer multiples of time step. Notice that the NAND only goes low for one of the 8 input combinations, meaning that this gate, though it has a significant switching delay, will not always manifest this delay; only in a few situations will the gate truly need to switch.

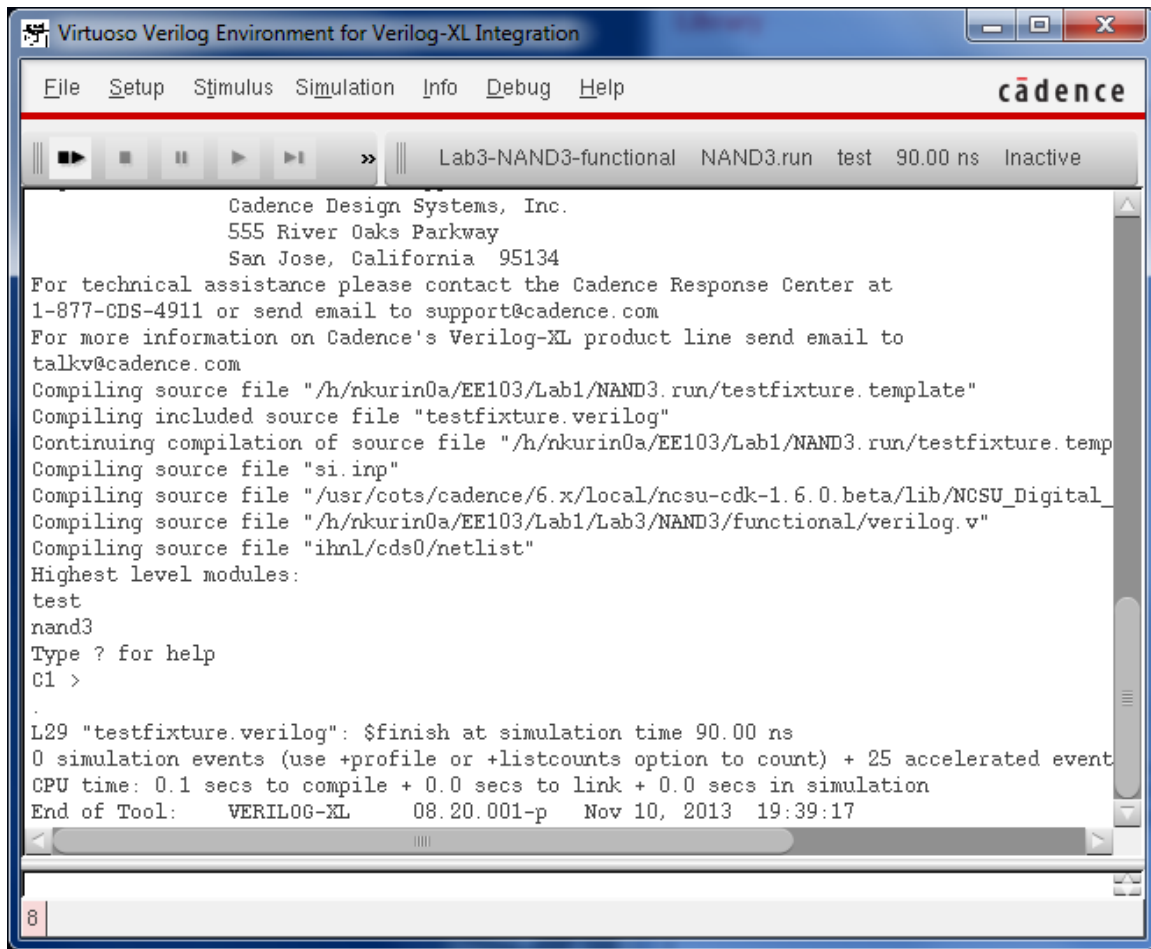


Figure 11: The output within the Verilog-XL simulation window for the NAND3 simulation, after running first running the initialization and final simulation.

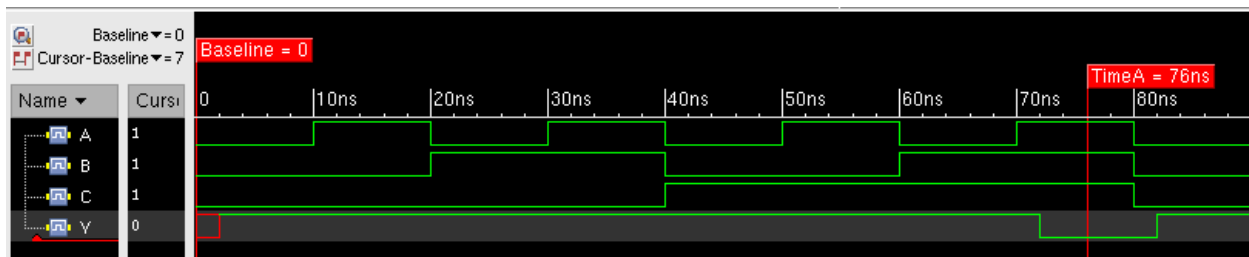


Figure 12: Output from the verilog 3-input nand gate simulation, with A, B and C the inputs set according to the three-input stimulus file, and Y being the NAND gate output. Here we see the delay at all transitions slightly higher than 2ns, but that all output is the correct NAND3 gate signal for times longer than the propagation delay. For instance, the only time the output drop sot zero is marked with the flag labeled “Time A”, and in the cursor column on the right you can see that A=B=C=1 so Y=0.

1.4 2-Input NOR Gate

For the 2-input NOR gate, the “nor” verilog module was employed with a delay of 1.666ns as can be seen in the functional view in figure 14. The delay matches that of the 2-input NOR logical effort of 5/3, although by now we expect this will be rounded up to 2 by Verilog-XL. The symbol created for this functional view can be seen in figure 13.

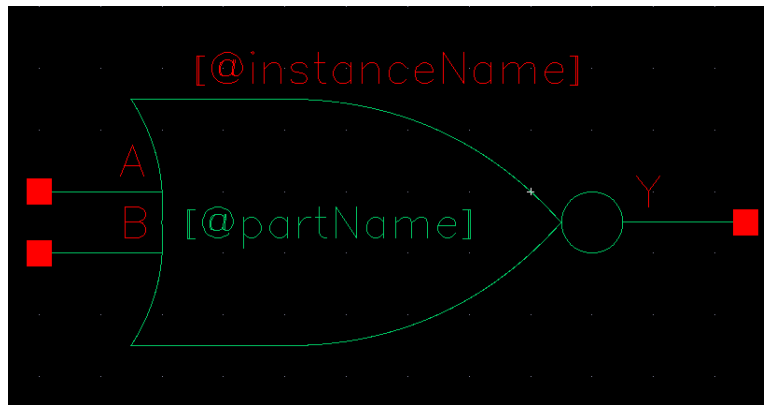


Figure 13: Schematic symbol associated with the 2-Input NOR functional view developed in this lab.

```

module NOR2 ( Y, A, B );

    output Y;
    input A;
    input B;
    nor #(1.666) n1(Y,A,B);
endmodule

```

Figure 14: Functional view for the NOR2 gate.

The NOR2 was simulated with Verilog-XL using the stimulus file in Appendix A2 for a two-input gate, and the output from the Verilog-XL window at the end of the simulation can be seen in figure 15. The output waveforms can be seen in figure 16. Here, as in the previous simulations, the gate performs the correct logical computation after the propagation delay, which here is slightly higher than expected, at around 2 seconds. One thing to notice about this simulation is that so long as one input is on, there is effectively 0 delay; for a path where one signal is expected to remain on much of the time, this would be a good gate to use despite the delay being higher than the NAND gate.

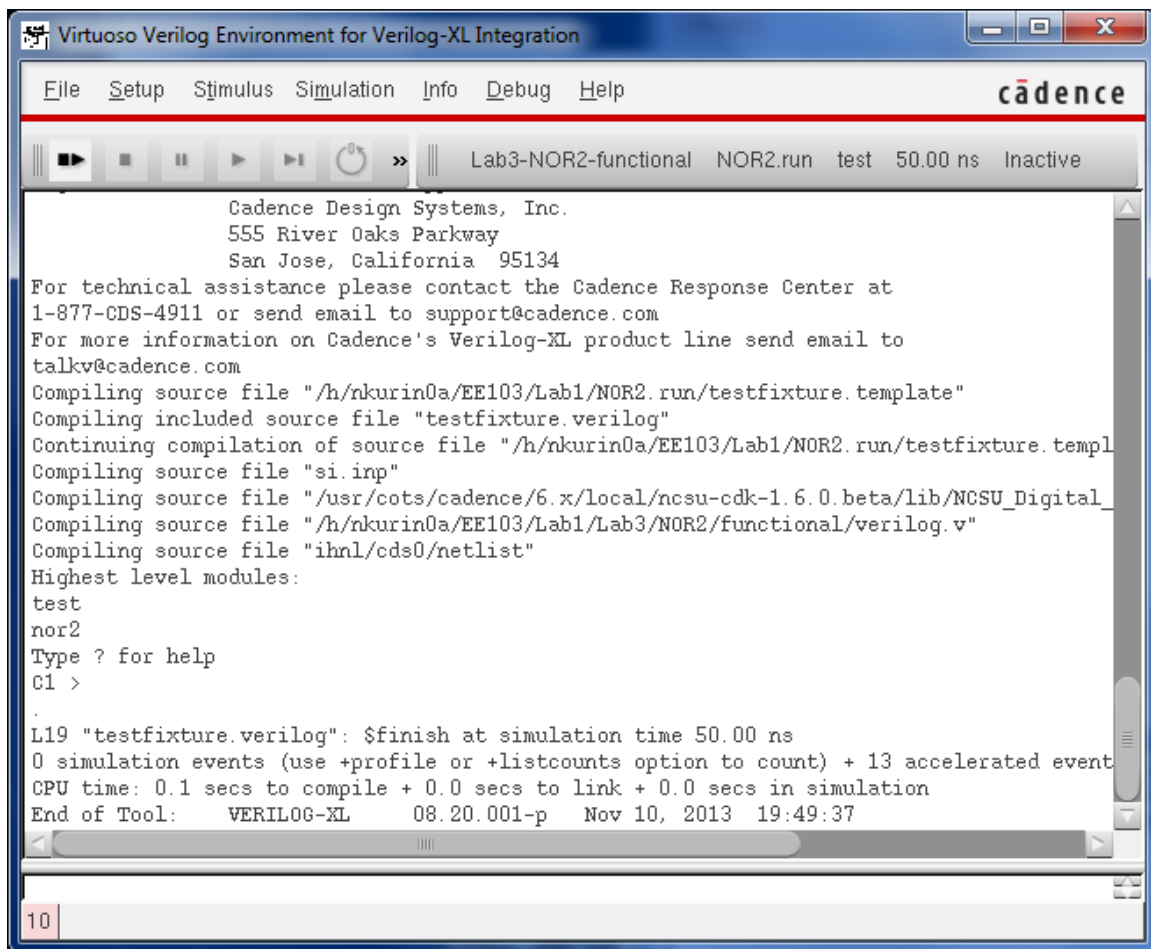


Figure 15: The output within the Verilog-XL simulation window for the NOR2 simulation, after running first running the initialization and final simulation.

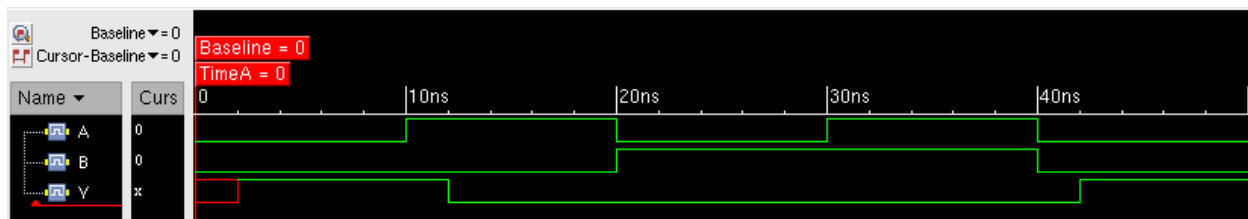


Figure 16: Output from the verilog 2-input nor gate simulation, with A and B the inputs set according to the two-input stimulus file, and Y being the NOR gate output. Here we see the delay at all transitions slightly lower than 2ns, but that all output is the correct NOR2 gate signal for times longer than the propagation delay; specifically, the output is low except for when both are 0. The delay seems a bit longer than the expected 5/3, which may be a result of integer delay requirements.

1.5 3-Input NOR Gate

For the 3-Input NOR gate, the verilog “nor” module was employed, as seen in the functional view in figure 18. The delay was set to 2.333ns, as the logical effort for the 3-Input NOR is 7/3. The NOR inputs are A, B, and C, respectively, and the output is Y. The associated symbol for the NOR3 can be seen in figure 17.

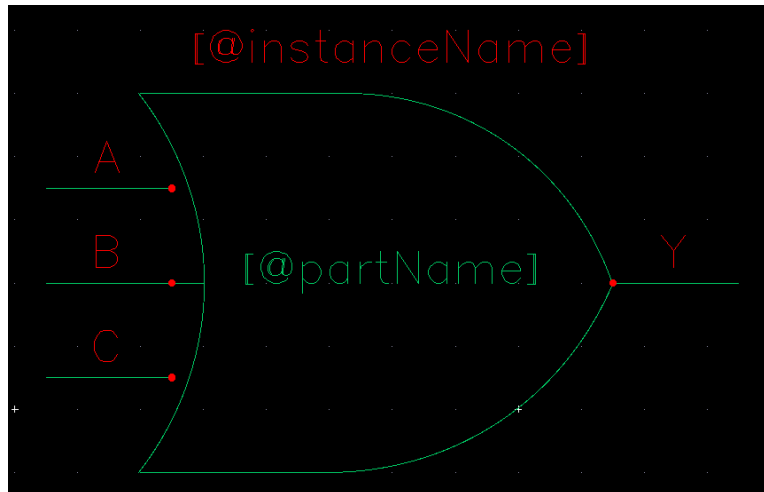


Figure 17: Schematic symbol associated with the 3-Input NOR functional view developed in this lab.

```
module NOR3 ( Y, A, B, C );

    input C;
    output Y;
    input A;
    input B;
    nor #(2.333) n1(Y,A,B,C);
endmodule
```

Figure 18: Functional view for the NOR3 gate.

The NOR3 was simulated with Verilog-XL using the stimulus file in Appendix A3 for a three-input gate, and the output from the Verilog-XL window at the end of the simulation can be seen in figure 19. The output waveforms can be seen in figure 20. Here, the gate performs the correct logical computation after the propagation delay, which here is slightly lower than expected, at around 2 seconds, as previously discussed. Notice that the NOR only goes high for one of the 8 input combinations, meaning that this gate, though it has a significant switching delay, will not always manifest this delay; only in a few situations will the gate truly need to switch.

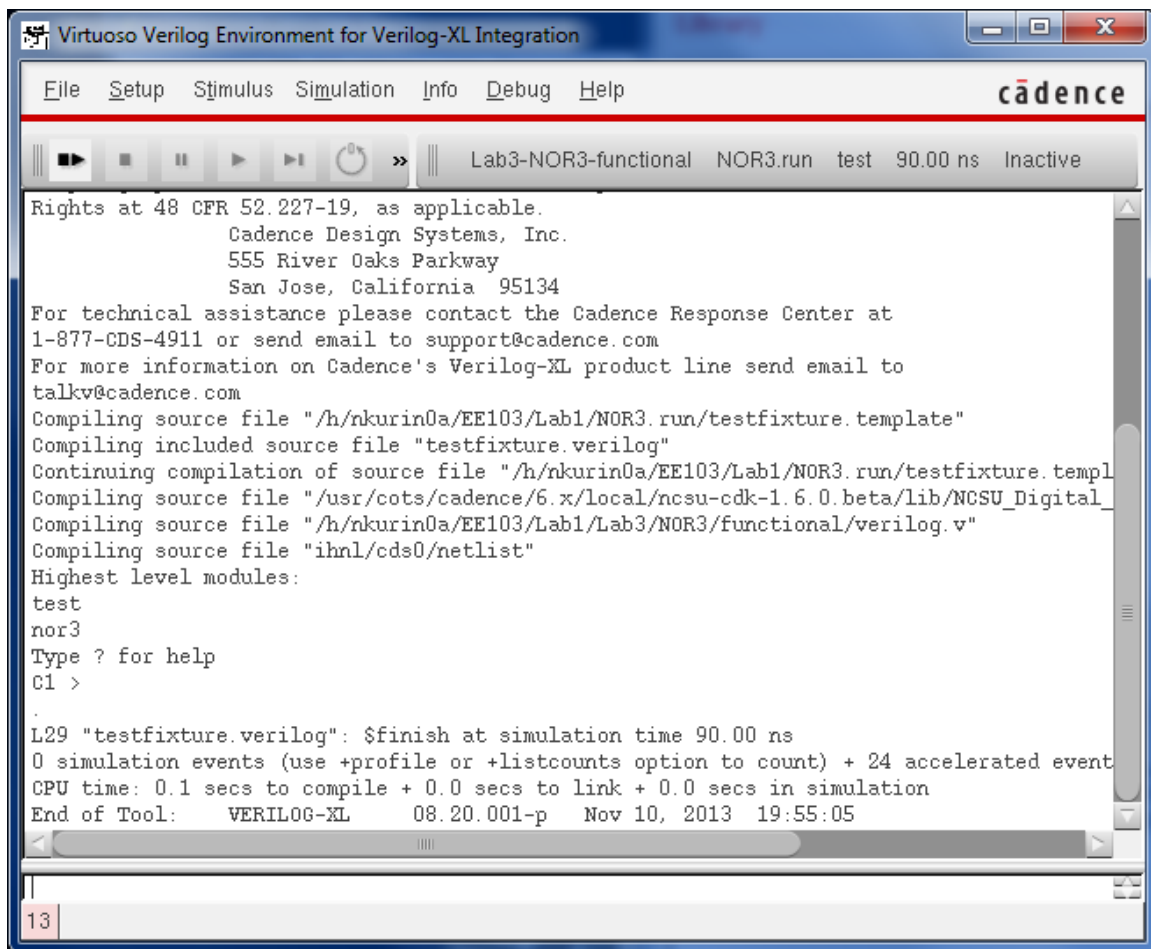


Figure 19: The output within the Verilog-XL simulation window for the NOR3 simulation, after running first running the initialization and final simulation.

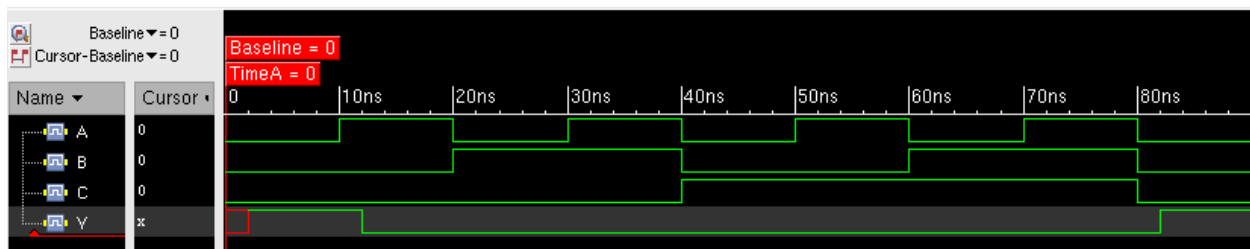


Figure 20: Output from the verilog 3-input nor gate simulation, with A, B, and C the inputs set according to the three-input stimulus file, and Y being the NOR gate output. Here we see the delay at all transitions slightly higher than 2ns, but that all output is the correct NOR3 gate signal for times longer than the propagation delay; specifically, the output is low except for when all three inputs are 0, at the start and end of the simulation. The delay seems a bit shorter than the expected 7/3, which may be a result of integer delay requirements.

1.6 2-Input XOR Gate

For the 2-input XOR gate, the verilog “xor” module was used, as seen in the functional view in figure 22. Here a gate delay of 4 was used, corresponding to the logical effort of the gate; this is twice as high as all the other delays due to the increased complexity of the XOR gate’s internals. The symbol created for this functional view can be seen in figure 21.

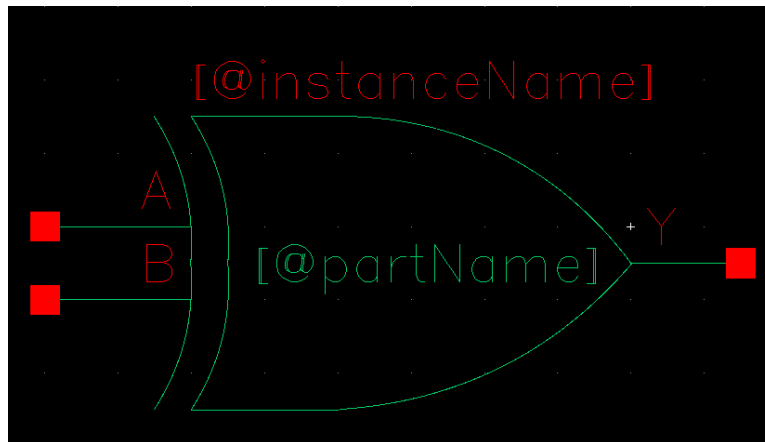


Figure 21: Schematic symbol associated with the 2-Input XOR functional view developed in this lab.

```
module XOR2 ( Y, A, B );

    output Y;
    input A;
    input B;
    xor #(4) x1(Y,A,B);

endmodule
```

Figure 22: Functional view for the XOR2 gate.

The XOR2 was simulated with Verilog-XL using the stimulus file in Appendix A2 for a two-input gate, and the output from the Verilog-XL window at the end of the simulation can be seen in figure 23. The output waveforms can be seen in figure 24. The delay approaches half of the input stability period here, and we see a shrinking region during which the XOR gate has a valid output; taken at random, there is only slightly better than a 50 percent chance that a given output is valid for this clocking rate. Here we also see that switching is more frequent than the other gates, with one half up and one half down values. This gate has high delay and high switching probability; it should most likely be avoided when possible.

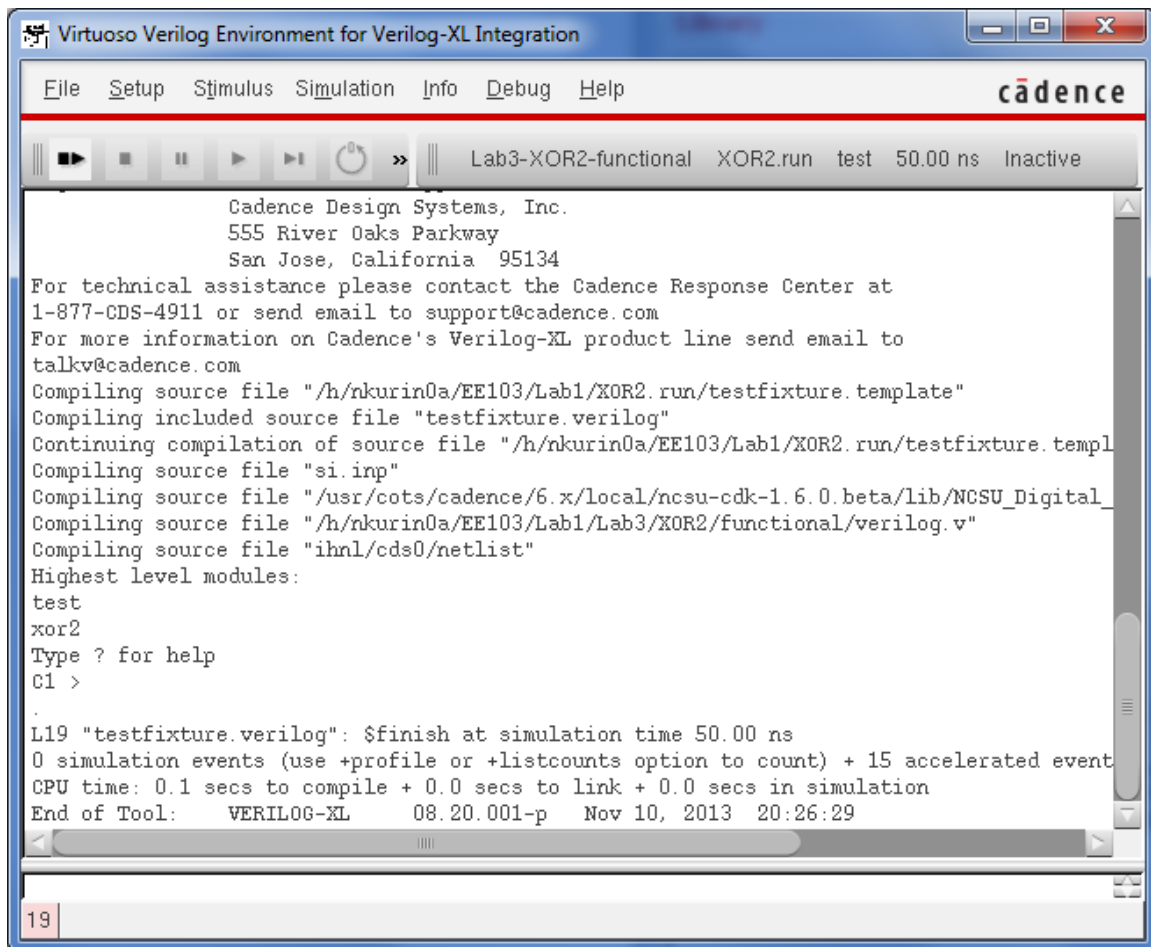


Figure 23: The output within the Verilog-XL simulation window for the XOR2 simulation, after running first running the initialization and final simulation.

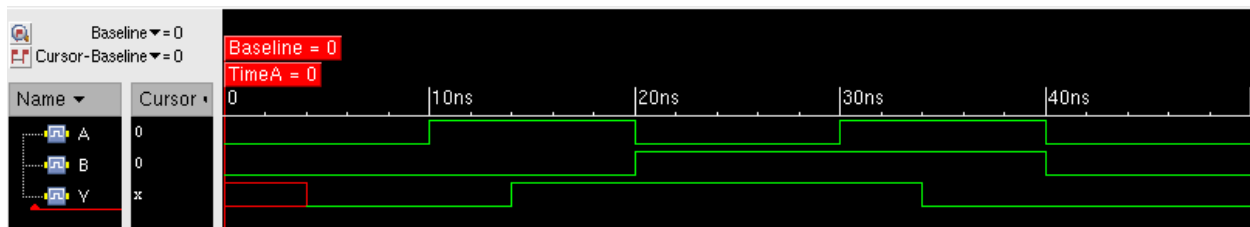


Figure 24: Output from the verilog 2-input xor gate simulation, with A and B the inputs set according to the two-input stimulus file, and Y being the XOR gate output. Here we see the delay at all transitions is higher than previous simulations at 4ns, but that all output is the correct XOR2 gate signal for times longer than the propagation delay; specifically, the output is low except for when only one input is 1. The delay, in approaching the magnitude of the pulse period, has begun to blur the cause/effect line for this gate.

2 Part III: Dynamic D-latch

The cadence schematic for the dynamic D-latch can be seen in figure 25 (the symbol is shown in figure 26). Here we see a circuit centered around a CMOS transmission gate. On the left, the data line is fed into the CMOS transmission gate, which is controlled by the clock C and its inverted pair. The output from the gate is fed to an inverter, from which the inverted stored bit can be read, and an additional inverter provides a non-inverted output. The storage here is performed by the first inverter's parasitic capacitance, and depending on technology and layout, will leak away at various rates.

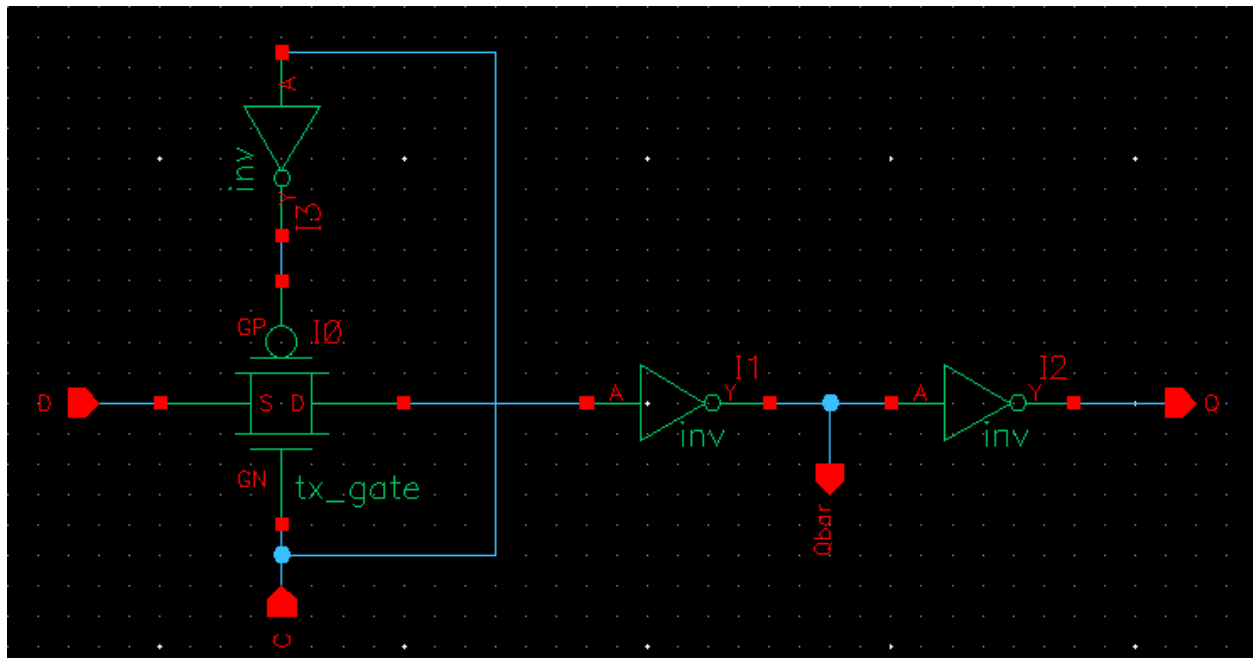


Figure 25: Schematic for the dynamic D-latch as it was created in cadence. On the left is the data input D , which is fed into a transmission gate, gated by a clock C and the inverted version of that clock. This is followed by a connection to two series inverters, from which the normal and inverted outputs are read. The symbol in figure 26 was generated from this schematic.

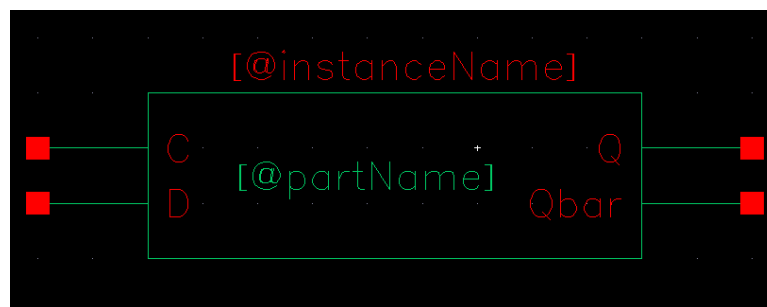


Figure 26: Schematic symbol associated with the `DLatch` functional view developed in this lab. Here, D marks the input node, C is the clock signal, and Q and $Qbar$ are the stored value and its inverted version. The latch is opaque when the clock is 0.

In order to model the effect of the decay rate of this capacitance, we constructed a functional view in verilog (shown in figure 27) to simulate the behavior of the latch. This functional view utilizes a “resistive”

CMOS switch which weakens the input slightly, and implements it with a 1ns delay. The storage aspect of the latch is achieved by using a trireg which allows its value to decay after a preset number of time units. I used 2ns and 20ns as my two decay times, as one is longer than hold time and the other is shorter. Finally, the two inverters are implemented using “not” gates with delay of 1ns.

```

module DLatch  ( Q, Qbar , C, D );

    input C;
    input D;
    output Qbar ;
    output Q;
    wire Cbar ;
    trireg #(0,0,20) sto ;

    not n1 (Cbar,C);
    rcmos #1 nml( sto ,D,C, Cbar );
    not #1 n2 (Qbar , sto );
    not #1 n3 (Q,Qbar );

endmodule

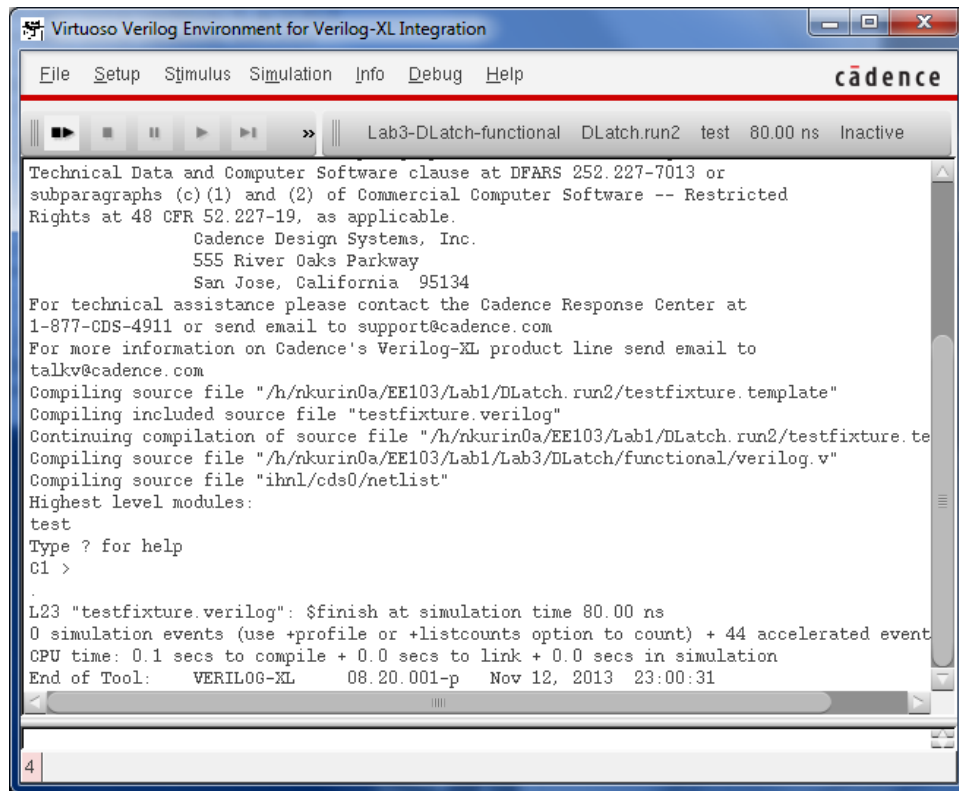
```

Figure 27: Functional view for the dynamic D-latch. Along with the usual inputs and outputs we saw in Part I, here we also have a wire and a trireg. The trireg acts as the leaking node as described in the introduction, and the wire acts as a variable without longterm storage; that is, it holds a value only in relation to another value, and if the value of the related variable changes it must as well (this is the same behavior as a physical wire). The trireg does have storage, which is destroyed after the decay time. Above, the decay time is set to 20ns, and this parameter was changed to 2ns for the short decay time simulation. tracing the connections, we see the clock is inverted, then the rcmos puts “D” to the trireg “sto” when C and Cbar allow it to do so with delay 1ns, and then “sto” is notted into cbar with delay 1ns, and Cbar is notted into C with delay 1ns.

This functional view was simulated in Verilog-XL using the stimulus files shown in appendix A section 4 for each of the two decay times. The output in the simulation window for both decay simulations is shown in figure 29, and on top you can see that the simulator detects the latch signal decaying to an indeterminate state for the 2ns decay setting. Looking at the simulation results in figure 30, we see that the top simulation where the decay rate was less than hold time shows periods of indeterminate output, whereas the bottom managed to store the value until the next refresh cycle. The take away from these simulations is that, for a dynamic latch like this, the decay rate of the stored signal sets the minimum operational frequency for the clock driving such a latch for it to remain valid at all times.

I also simulated the normal latch function using asynchronous D and C signals, read from the associated stimulus file in A4. The output in the simulation window can be seen in figure 28, and the simulation results can be seen in figure 31. Not that no transitions occur after the clock has transitioned to a 0 aside from those which occur only due to propagation error, but when the clock rises again the inputs are then passed through the latch. For these simulations, I used the delay valid for a clock of this period to demonstrate a stable latch.

In figure 32, I have added gates which make the dynamic, leaky latch a static self-sustaining latch which can function for any clock frequency. To achieve long-term stability, I have added a feedback path and an input buffer. The input buffer makes the latch regenerative and reduces output noise. The tristate feedback buffer is off while the latch is transparent, but is on when the latch is opaque, driving the first buffer after



```

Virtuoso Verilog Environment for Verilog-XL Integration
File Setup Stimulus Simulation Info Debug Help cadence
Lab3-DLatch-functional DLatch.run2 test 80.00 ns Inactive

Technical Data and Computer Software clause at DFARS 252.227-7013 or
subparagraphs (c)(1) and (2) of Commercial Computer Software -- Restricted
Rights at 48 CFR 52.227-19, as applicable.
Cadence Design Systems, Inc.
555 River Oaks Parkway
San Jose, California 95134
For technical assistance please contact the Cadence Response Center at
1-877-CDS-4911 or send email to support@cadence.com
For more information on Cadence's Verilog-XL product line send email to
talkv@cadence.com
Compiling source file "/h/nkurin0a/EE103/Lab1/DLatch.run2/testfixture.template"
Compiling included source file "testfixture.verilog"
Continuing compilation of source file "/h/nkurin0a/EE103/Lab1/DLatch.run2/testfixture.te
Compiling source file "/h/nkurin0a/EE103/Lab1/Lab3/DLatch/functional/verilog.v"
Compiling source file "ihnl/cds0/netlist"
Highest level modules:
test
Type ? for help
C1 >
.
L23 "testfixture.verilog": $finish at simulation time 80.00 ns
0 simulation events (use +profile or +listcounts option to count) + 44 accelerated event
CPU time: 0.1 secs to compile + 0.0 secs to link + 0.0 secs in simulation
End of Tool: VERILOG-XL 08.20.001-p Nov 12, 2013 23:00:31

```

Figure 28: The output within the Verilog-XL simulation windows for the DLatch latch-behavior simulation, after running first running the initialization and final simulation.

the output repeatedly with the value initially at the node. The output buffers have been isolated from this feedback loop to further reduce the effect of the input and output on the stored latch value. If we were to simulate this latch in verilog, we might still need a register, however the decay time would only be limited by the gate delays, here only around 2ns; thus the node with the short decay time would be adequate, and a clock of any frequency could be used with the gate instead of having to limit the clock based on gate delay and needing to constantly refresh the latch.

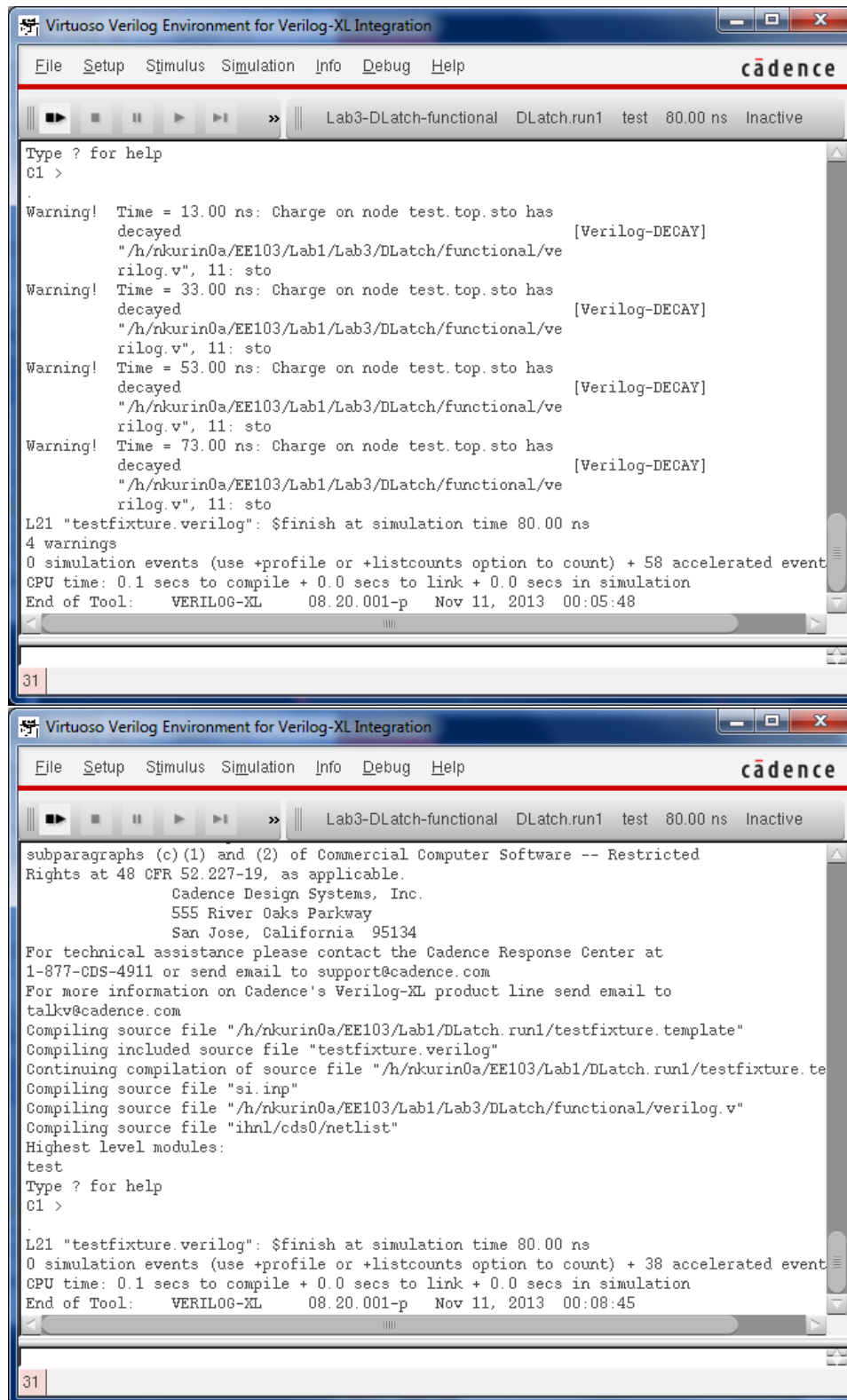


Figure 29: The output within the Verilog-XL simulation windows for each of the two DLatch simulations, after running first running the initialization and final simulation. Note on the top the warnings that one of the signals has decayed, due to the fact that the trireg had a decay time shorter than the hold time required for the latch.



Figure 30: The output from the dynamic D-latch simulations, with the short node decay time simulation on top, and the long decay time on the bottom. Here, C is the clock, which has a period of 20ns so the latch has a desired hold time of 10ns, D is the data line, sto is the register, and C and Cbar are the normal and inverted outputs. On top, we see that only a few nanoseconds after the gate is turned off, the node decays and the latch reaches an indeterminate state; on the bottom, the latch never decays and the outputs are always defined. During the periods where the outputs are valid, though, the output does reflect the last input.

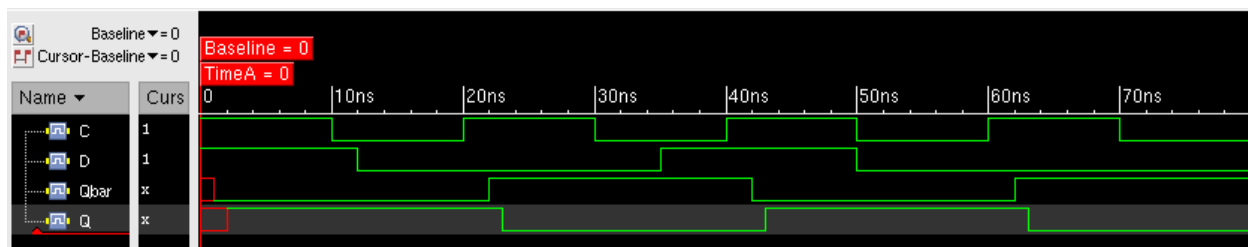


Figure 31: The output from the dynamic D-latch simulation of latch behavior. Here, C is the clock, which has a period of 20ns so the latch has a desired hold time of 10ns, D is the data line, and C and Cbar are the normal and inverted outputs. Stepping through the simulation left to right, we see that the latch is transparent for a clock level of one, but if D changes during the clock's down cycle, Q and Qbar do not reflect this change until after the next rising edge. This behavior is seen three times above for various D transition times.

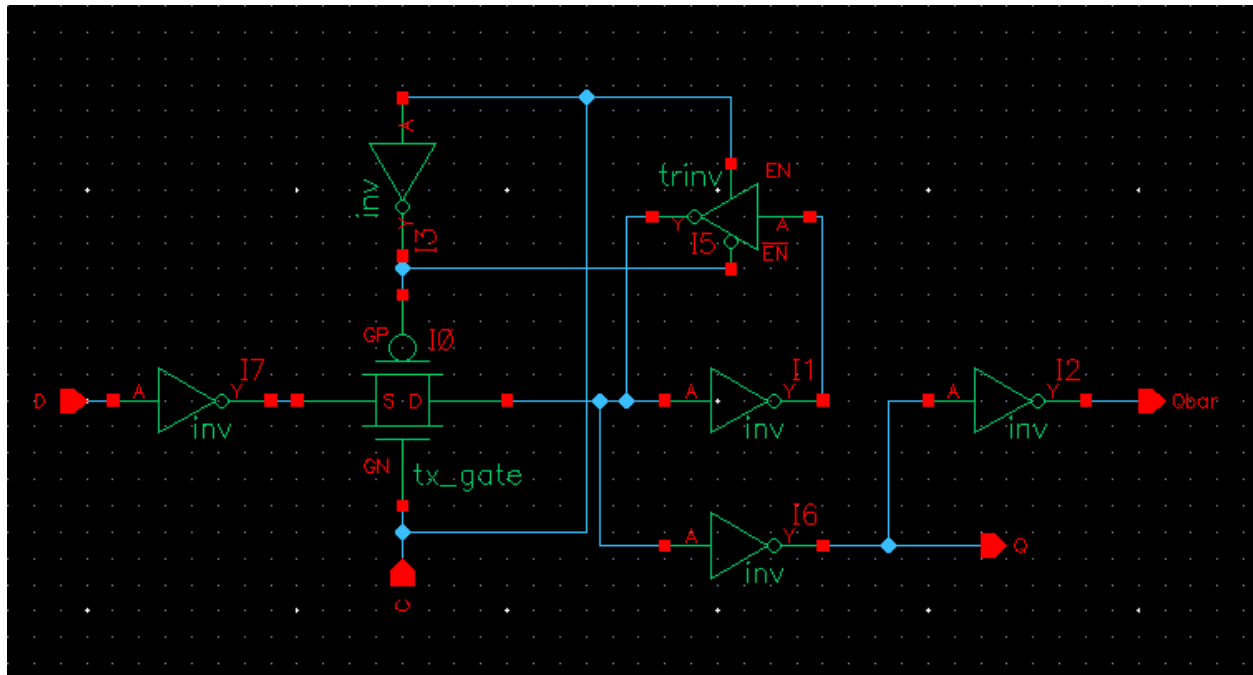


Figure 32: The schematic of the static D-latch, which achieves stability through multiple mechanisms. We have kept the gate, but now include an inverted on the input to stabilize the input, and a tristate inverter feeding the normal output back to the storage node, which is on when the gate is turned off. This feedback loop keeps refreshing the node, preventing it from decaying when the latch is in a storage state. In addition, other inverters isolate this loop from the output to prevent the output from overpowering the tristate and destroying the stored node.

Discussion

In the previous sections, and through the work done for the associated report, we have seen how powerful modularization with verilog functional views can be when designing a large circuit out of even the simplest gates, and how complex delay behavior can be approximated using logical effort. We made some assumptions early on about electrical effort and branching which will most likely not hold for most real gates, however if stages are not sized for maximum delay, this technique is probably a good estimator, as the logical effort relates stages in terms of delay to the inverter's unit delay. This will provide us with a rough delay estimate to begin analysis; it will also allow us to determine where custom gates may be necessary, or transistor level implementation will be more efficient.

In addition, verilog showed itself to be a very capable simulation language, both in terms of modeling gate behavior and generating robust testing signals; this was most poignant for the 8-bit adder. The gate primitive implementation was easy mostly due to the robust nature of the built in gate modules, which allow various numbers of inputs, and the ease with which delays and decays can be added to models. The transistor level modeling was also nice, and allowed for an admittedly roughly but insightful simulation of the dynamic dlatch.

The dlatch simulation and design illustrated a few points. First, relying on parasitic capacitance as a storage element is immensely unreliable; second, that without active feedback mechanisms, gates are both unstable and unpredictable. The initial dlatch design showed itself to be very crude and highly dependent on application, and in most instances the static configuration is much more preferable. It buffers both input and output, and eliminates the effects of all but the most drastic sources of leakage, which might cause problems for non-storage elements as well.

It will be nice to compare these initial verilog simulations to later transistor level cadence simulations to see how accurate they were, and the tools developed in this lab will be invaluable for future modular design of VLSI structures. In the adjunct report, we reviewed many improvements we can make in light of the work done in this lab, and the final project will be much more robust, efficient, and thoroughly designed.

A Stimulus Files

Below are the stimulus files used to run the Verilog simulations of all gate primitives and the DLatch; the file used for the primitives was the same for all gates with the same number of inputs. In order to better resolve the timing delays for these gates, a clock with a 10-ns pulse width was employed (the clock therefore had a period of 20-ns). In other words, stimulus files only switched inputs at time intervals of 10 ns. The D latch fixes the input at each possible value for two full clock cycles before exiting the simulation.

1 One Input

The following corresponds to the sequence $A = 0 \rightarrow 1 \rightarrow 0$

```
initial  
begin  
  
    Vin = 1'b0;  
    #10 Vin=1'b1;  
    #20 Vin=1'b0;  
    #10 $finish;  
  
end
```

2 Two Inputs

The following corresponds to the sequence $BA = 00 \rightarrow 01 \rightarrow 10 \rightarrow 11 \rightarrow 00$

```
initial  
begin  
  
    A = 1'b0;  
    B = 1'b0;  
    #10 A=1'b1;  
    #10 A=1'b0;  
        B=1'b1;  
    #10 A=1'b1;  
    #10 A=1'b0;  
    B=1'b0;  
    #10 $finish;  
  
end
```

3 Three Inputs

The following corresponds to the sequence

$CBA = 000 \rightarrow 001 \rightarrow 010 \rightarrow 011 \rightarrow 100 \rightarrow 101 \rightarrow 110 \rightarrow 111 \rightarrow 000$

```
initial  
begin  
  
    A = 1'b0;  
    B = 1'b0;  
    C = 1'b0;  
  
    #10 A=1'b1;  
    #10 B=1'b1;  
        A=1'b0;  
    #10 A=1'b1;  
    #10 A=1'b0;  
        B=1'b0;  
        C=1'b1;  
    #10 A=1'b1;  
    #10 A=1'b0;  
        B=1'b1;  
    #10 A=1'b1;  
    #10 A=1'b0;  
        B=1'b0;  
        C=1'b0;  
    #10 $finish;  
  
end
```


4 D-Latch

This stimulus file was used for testing the latch behavior:

```
initial  
begin  
  
    C = 1'b1;  
    D = 1'b1;  
    #10 C = 1'b0;  
    #2 D = 1'b0;  
    #8 C = 1'b1;  
    #10 C = 1'b0;  
    #5 D = 1'b1;  
    #5 C = 1'b1;  
    #10 C = 1'b0;  
        D = 1'b0;  
    #10 C = 1'b1;  
    #10 C = 1'b0;  
    #10 $finish;  
  
end
```

This stimulus file was used for testing the leaky node:

```
initial  
begin  
  
    C = 1'b1;  
    D = 1'b1;  
    #10 C = 1'b0;  
    #10 C = 1'b1;  
    #10 C = 1'b0;  
    #10 C = 1'b1;  
        D = 1'b0;  
    #10 C = 1'b0;  
    #10 C = 1'b1;  
    #10 C = 1'b0;  
    #10 $finish;  
  
end
```