

## Project 3: Cadence Tools, part 1 (10%)

ENEE 359a: Digital VLSI Design, Spring 2007

Assigned: Thursday, March 1; Due: Tuesday, March 27

### 1. Purpose

The objective of this project is to familiarize yourself with the different programs included in Cadence. The Cadence suite is a huge collection of programs for different CAD applications from VLSI design to high-level DSP programming. The suite is divided into different “packages,” and for VLSI design, the packages we will be using are the IC package and the DSMSE package (we’ll talk about these as we go along).

Specifics: Design and SPICE verification of inverter, NAND, NOR and MUX gates using the TSMC 0.25um process (also using SCMOS rules and not native-TSMC rules).

### 2. Setup

1. The first thing to do is to setup your directory structure properly so that you can use the required libraries used for the design flow. Download the two files enee359a\_files.tar and IIT\_stdcells.tar.gz from the Project Distribution directory (on course website).
2. In the root of your home directory, extract IIT\_stdcells.tar.gz. (First execute ‘gunzip IIT\_stdcells.tar.gz’ then untar it by doing ‘tar -xvf IIT\_stdcells.tar’)
3. Create your working directory. For the rest of this walkthrough, the assumed name of the working directory will be enee359a. Copy the file enee359a\_files.tar to the enee359a directory and untar it there. (Again, do a ‘tar -xvf enee359a\_files.tar’).
4. Change any occurrence of the string ‘USERNAME’ in the configuration file ‘cds.lib’ into your username and then save as ‘cds.lib’. One way to do this is to use the following: ‘sed ‘s/USERNAME/your\_username/g’ cds.lib.orig > cds.lib’, where you replace “your\_username” with your real username. This uses the program sed and redirects its output to the file cds.lib.

### 3. Creation of Simple Standard Cells

#### 3.1 Inverter design

1. The first step is to setup our environment so that we can access the necessary Cadence programs. Go to the enee359a directory, type ‘tap cds-v’ and then type ‘source /software/cadence/setncsu.’ This executes a script that will setup our environment properly. (The programs we will be using are in the IC, or Integrated Circuit Design, package, but we are also going to use the enhancements to this package that are provided by NCSU)
2. Invoke the main frontend program by doing ‘icfb &’. At least two windows will pop up. Close everything except the icfb window and the Library Manager window.
3. The next step is to create a library where you will store your designs. In Library Manager, go to File->New->Library. In the name field, type in “simple\_cells” (this is actually just an arbitrary name). In the Path field, type in the full path for the enee359a directory (For glue, this would be /homes/<username>/enee359a). In the Technology Library field, choose “Attach to existing tech library” and then choose TSMC 0.30u CMOS025. Click on OK. You should now see the simple\_cells library in the Library column of the Lib Manager.

4. We now start with creating the schematic of a simple inverter. When finished, the schematic should look similar to the one shown in Figure 1. Click on the simple\_cells library then go to File->New->Cellview. In the window that pops up, type in “inv” (without the quotes. Also, the name is again arbitrary) in the Cell Name field. For the View Name field, you can either directly type “schematic” to inform cadence what type of design you’re making, or choose “Composer - Schematic” in the Tool options menu. Click on OK. A window titled Virtuoso Schematic Editing will now open. This is the Cadence program that we will be using whenever we want to design circuits on a schematic level.
5. The next step is to add the NMOS and PMOS transistors of the inverter. Go to the menu option Add->Instance and the Add Instance window should appear. (Important comment: using Cadence becomes a whole lot easier if you know the short cut keys that activate different tools and options. In this case, the shortcut key for adding an instance is ‘i’). Click on Browse to bring up the Library Browser window. In the library field, choose “NCSU\_Analog\_Parts”, then locate the “nmos” cell in the Cell column and the “symbol” view in the View column. In the Add Instance window that pops up, you can modify the different parameters of the NMOS you’re instantiating. Mostly, you’ll be fiddling around with the Width parameter, but in this case, the default value of 450nm (which is the minimum size allowable for this technology) will suffice. Click once on the schematic window to instantiate one mosfet. After clicking, notice that an NMOS symbol is still attached to the cursor. This means that the “Add Instance” command is still active and clicking again will instantiate another transistor. To deactivate the present command, just press ESC.
6. Do the same thing to instantiate a PMOS transistor, but make its width twice that of the NMOS (in this case, 900n). Place the PMOS above the NMOS.
7. Some tips on navigating: you can use the arrow keys to pan around your design. To zoom out, either use the toolbar, or use shift-z. The shortcut for zooming in is ctrl-z, and if you want to fit your entire design in the schematic window, just press ‘f’. Shortcut keys are case-sensitive, so pressing f is different from shift-F. Weirdly enough, the CAPS LOCK doesn’t affect the shortcut keys, i.e. pressing ‘f’ with CAPS on will be interpreted as a lower case f.
8. The next step is to instantiate the connections to the power and ground rails. To do this, repeat the same steps as instantiating a transistor, but this time, locate the “vdd” and the “gnd” cells and place them accordingly. Any node in your schematic that is attached to these nodes are implicitly electrically connected. In this case, it doesn’t really matter since only one node each are connected to both vdd and gnd, but in larger circuits, these symbols make the schematic a lot neater by not requiring explicit wire connections between each vdd/gnd connection (Note: a lot of times, only the Add Instance button appears and you have to click on Browse to show the Library Browser. If you have an idea of what cell you want to instantiate and from what library, you can directly modify the fields in the Add Instance window to get the cells and cellviews you want).

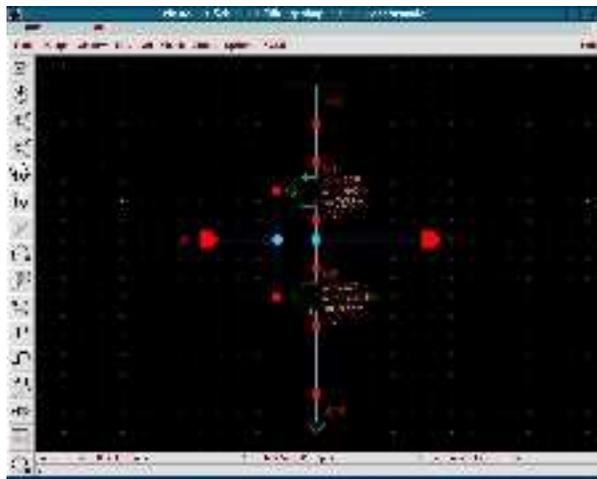


Figure 1: Finished transistor-level schematic of an inverter.

9. The next step is to add I/O pins to the schematic so that signals can go in and out (or both) of this design. Go to Add->Pin (shortcut: 'p') and type in "A" in the Pin Names field. Make sure that the direction is "input" and then place the pin near the gates of the transistors. Do this again for the output by typing "Y" and then changing the pulldown menu option to "output."
10. The last step to finish the schematic is to connect all the instances properly using wires. Go to Add->Wire(narrow) (shortcut: 'w') and do the connections. The easiest way to do this is to use "wire snapping." Notice that when the wire command is active, a yellow diamond can be seen in the design showing where the nearest valid wire connection can be made. Pressing 's' in the keyboard snaps the wire directly to that connection, and the diamond jumps to the next nearest valid connection. Try using this method to finish the connections. If you make mistakes, you can undo it by using the undo command (shortcut: 'u'), or by deactivating the wire command and then selecting the wire you want to delete and then deleting it (shortcut: 'del'). (Important comment: At this point, it is really important to realize that most commands in Cadence stay active after you use them, like the Add Instance example. This can be quite frustrating if you have the delete command active and then click on one of your cells, unintentionally deleting it. You can see which command is presently active by looking at the bottom part of the Schematic Window. Again, pressing ESC once will deactivate the present command, but also keep in mind that Cadence commands are nested, so you may have to press ESC multiple times to make sure that no command is active so you can actually select something. This command nesting is very frustrating at first, but becomes very useful when you get the hang of it).
11. When finished, click on the uppermost button in the left toolbar to do a "Check and Save". This performs rudimentary check of the design (stuff like unfinished connections, floating wires, etc.), and saves the design if no errors or warning are found. Otherwise, you are prompted whether or not you want to proceed with the save. For this simple design, you shouldn't have any warnings whatsoever.
12. At this point, you have finished creating the transistor-level schematic of an inverter. Assuming it's working (we'll verify this later), this can now be used in higher level designs. Obviously, we don't want to use the transistor-level schematics in higher-level designs because they are cumbersome to use and require some thinking before their functions are deduced. Although an inverter is easy enough to recognize, more complicated circuits like XOR gates are significantly tougher. Instead, we will create a "symbol" cellview that we can use to represent the inverter. Go to Design->Create Cellview->From Cellview. Use the default values in the window that pops up and click on OK. A new window titled "Virtuoso Symbol Editing" should pop up showing a rectangular block that has two pins, A and Y. These pins directly correspond to the pins you defined earlier in the schematic. This symbol should be good enough for our use. (To those who want to make their inverters look better, go to Add->Import Symbol and add the symbol cellview of the inverter from the Analog Digital Parts Library. After importing the symbol, delete the rectangular version.). Save the symbol cellview by clicking on the uppermost button in the toolbar, then close the window.
13. We now use SPICE to verify the functionality of our inverter. The first step to do this is to create a new schematic cellview that we will use as the testbench for our inverter. In the Lib Manager window, create a new schematic cellview in the simple\_cells library called inv\_test (again, an arbitrary name). (To do this, make sure that the library simple\_cells is selected, then go to File->New->Cellview then specify inv\_test as the Cell name and choose the Composer-Schematic tool.)
14. In the schematic window that pops up, press 'i' to add an instance and then go to the simple\_cells library. Since we've created a symbol cellview for the inverter, we can now use it in a schematic. Add two instances of the inverter you've created, placing them side by side with the intention of cascading them together.
15. We now have to instantiate input sources that will allow us to test the functionality of our device-under-test (DUT). It is simple in this case because any square wave with the right voltage will enable us to determine correctness. For more complicated circuits, creating a testbench becomes more involved. To instantiate a square wave pulse train, press 'i' (to add an instance) and add the symbol cellview of vpulse from the NCSU\_Analog\_Parts library. Modify the parameters of this voltage source in the Add Instance window by using the following values: Voltage1 = 2.5, Voltage2 = 0 Rise time = 100p, Fall time = 100p, Pulse width = 2n, Period = 4n. (Note: When modifying parameters, only enter the value and the unit modifier, not the actual unit. For example, when entering time values for the period, 4n was used and not 4ns. The "s" is automatically put in afterwards.

Entering 4ns will result in errors during the SPICE simulation because “ns” is not a valid number modifier, while “n” is interpreted correctly as nano or  $10 \times 10^{-9}$ . Create an instance of vpulse to near the input of the leftmost inverter.

16. To make the simulation more realistic, we also simulate an output load that is to be driven by the second inverter. In this case, we instantiate a capacitor to simulate the capacitive load presented by other circuits connected to the output. Add an instance of “pcapacitor” from the NCSU\_Analog\_Parts library. For the capacitance value, instead of specifying a capacitance value, we can make it variable by entering an arbitrary variable name in the Capacitance field. For the rest of the discussion, the variable name is assumed to be Cload. Note that this concept of using variables for use in SPICE simulation can be applied to any device parameter, including that for the transistors. Instead of directly specifying our transistor parameters, for example gate width, we could have used variables instead so we can change them on the fly during SPICE simulations. Of course, this is useful mostly for parameter study and optimizing your design. In final physical designs, these values will always have to be set to some correct value. Create the capacitor near the output of the second inverter.
17. The next step is to include a DC power source that will supply power to your circuit. Instantiate the “vdc” cell from the NCSU\_Analog\_Parts library and specify a voltage of 2.5 in the DC Voltage field. Add this voltage source anywhere in the schematic. Like what you did for the inverter supply rails, instantiate a “vdd” and “gnd” symbol near the DC source. When connected, this will distribute the voltage source to any part of the circuit that is connected to vdd. Although no other nodes in the testbench will seem to be connected to vdd, the transistors in the inverter schematic will be connected to this global vdd by virtue of having connections to an instance of the “vdd” cell. This is the same for gnd. Instantiate two other gnd cells for use of the vpulse and the capacitor.
18. Finish the connection of the entire schematic by connecting them using wires. The final schematic should look similar to Figure 2, with the exception of the labels. To create the labels attached to the wires, go to Add->Wire Name (shortcut: l) and in the Names field, type in “IN OUT1 OUT2” and successively click on the corresponding wires as shown in Figure 2. This step assigns a user-defined name to specific wires. Without this step, the software will automatically name the wires (with cryptic names such as net020) and it would be more confusing later on when we display the voltage waveforms of wires we are interested in.
19. Before proceeding with SPICE simulation do a “Check and Save” first.

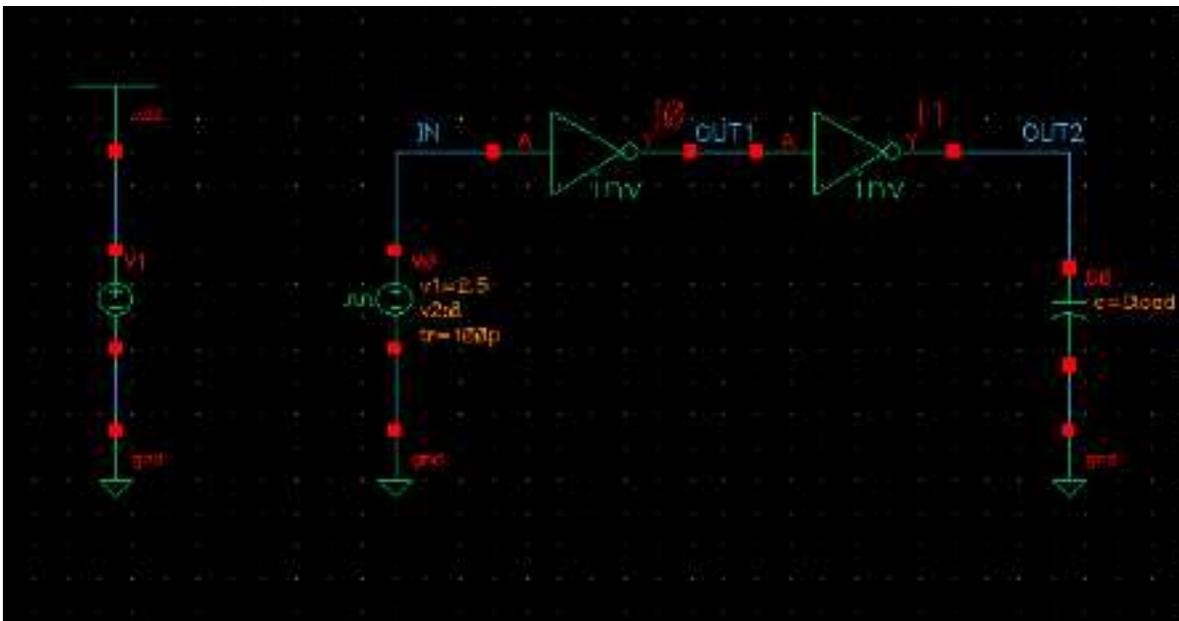


Figure 2: Final schematic of the testbench used to test our inverters.

20. We now use SPICE simulation to verify whether our inverter is functionally correct. There are different ways to do this in Cadence, depending on what SPICE engine you want to use (i.e. HSPICE, Spectre, PSpice, etc...). In our case, we are going to use the Analog Design Environment Spectre SPICE frontend for Cadence. Go to Tools->Analog Environment.
21. We first need to configure Analog Design Environment so that it works the way we want it to. Go to Setup->Simulator/Directory/Host and choose spectreS in the Simulator pulldown menu. This instructs Analog Design Environment to use the SpectreS SPICE engine. Click on OK. Next, we need to make sure that Cadence can locate the SPICE model files for our transistors. Go to Setup->Model Path. For processes older than TSMC 0.25um, all the model files are included as part of the NCSU distribution. For TSMC 0.25um, the model files are included as part of the enee359a\_files.tar tarfile you were given. Type in the complete path for the enee359a directory (i.e. /homes/<user-name>/enee359a) in the New Directory field and make sure that it is the first entry in the Directories Field. Click on OK when you're finished. Next, we now need to tell SPICE what kind of analysis to perform. Go to Analyses->Choose and then make sure that 'tran' is selected (for transient analysis). Make sure that the Enabled check box in the lower left corner is active, and then specify a stop time of 15n. Click on OK. Remember that the period of our input pulse is 4ns, so this should show us roughly 4 output cycles. We now need to tell Analog Design Environment which waveforms we are interested in. Go to Outputs->To Be Plotted->Select on Schematic and then proceed to the schematic of our testbench and click on the wire names IN, OUT1, and OUT2. The colors of the label and wires should change to indicate that they have been selected. Go back to the Analog Design Environment window and you should see these signals enumerated in the Outputs field. Normally, we should be ready to start the simulation, but in this case, we need to specify values to variables we've defined earlier. Go to Variables->Copy from Cellview. In the Design Variables field, the variable Cloud should appear. Double click it, and then enter a value of 20f in the Value(Expr) field. Click on Change then OK. The value should be reflected in the Design Variables field. We can now proceed with the SPICE simulation by clicking on the green stoplight button in the right toolbar. The text output of the SPICE run should be seen in the icfb window, along with any errors that may have occurred.
22. If everything goes well, a window titled Waveform Window should pop up showing the three overlapping waveforms IN, OUT1 and OUT2. Maximize this window, then click the Switch Axis Mode button in the toolbar (second from the bottom). This should show the three waveforms on different axes and should look similar to Figure 3. (Ordering is unimportant)
23. Some comments: Looking at the waveforms, notice that unlike the two other waveforms, IN is very linear. It is important to realize that aside from the rise and fall time constraints we specified for

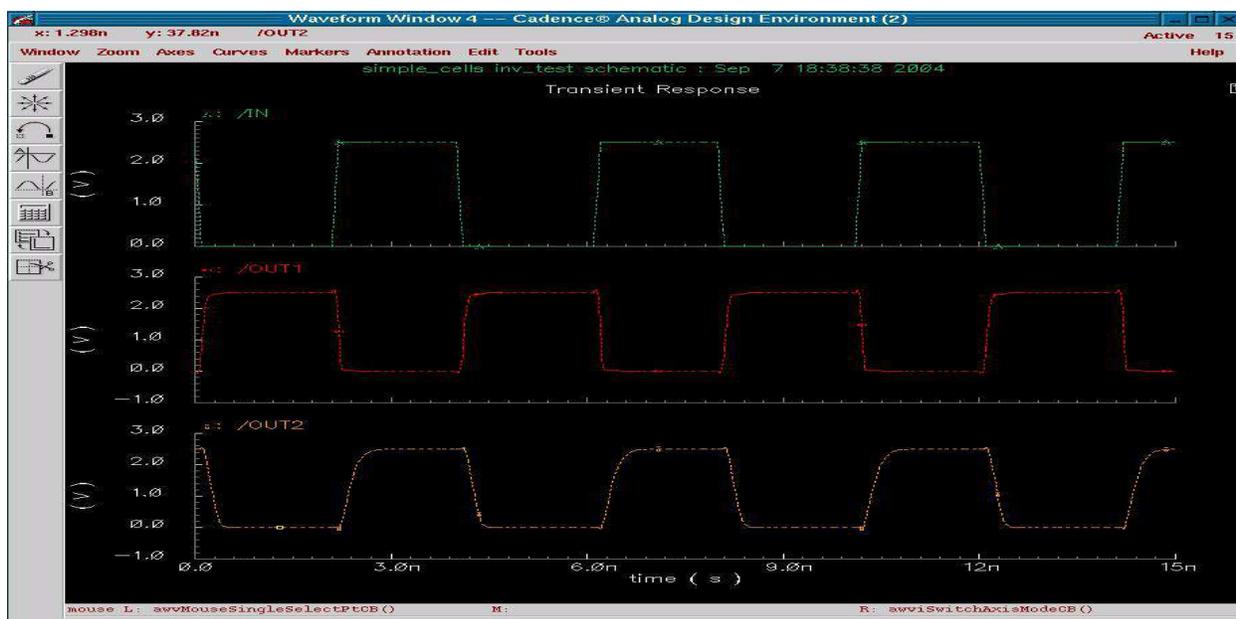


Figure 3: Voltage output waveforms from the inverter testbench.

vpulse, it is an ideal voltage source such that it will be able to maintain the required output voltage regardless of output current. Remember from basic electric circuits that voltage sources have an associated parasitic series resistance. As the current output of the voltage source rises, the voltage drop across the resistance increases and correspondingly, the output of the voltage source decreases. Most voltage sources in SPICE are ideal in that their series resistance is zero and that the source can supply as much current as required. This is well and good during simulation, but remember that we are here to build real circuits -- you have to keep in mind that real circuits won't have the benefit of these "ideal" sources, and you have to be careful on deciding whether or not the ideal characteristics of the sources you're using will affect the realism and accuracy of your result. To give you an idea of how this affects your circuit, take a look at how OUT2 deviates from the ideal during the rise and fall of the waveform. This deviation is obvious because it is driving a significant capacitive load (roughly equal to about the capacitive loads presented by four minimum-sized inverter inputs). Since the current that has to charge/discharge this capacitive load has to go through one of the inverter transistors and its effective ON channel resistance, the voltage cannot change instantaneously but instead undergoes an exponential increase/decay dictated by the RC circuit formed by the MOS ON-resistance and the capacitive load.

24. A lot of things can be done in the Waveform Window to get information out of the displayed waveforms. One useful method is to use cursors to measure absolute time/voltage or their differences between two points. Pressing 'a' should show you a crosshair with "A" in the center. Try moving this crosshair around. Notice that time and voltage information about the center point is displayed in the lower right. Click on a point at the rising edge of IN that is about halfway up (1.25V). Pressing 'b' will allow you to use a second crosshair and make measurements. With two crosshairs active, additional information is displayed showing the time and voltage difference between the two crosshairs and the resulting slope. Click the halfway point of the falling edge of OUT1 caused by the rising edge of IN that you have clicked before. This is a good metric in measuring the delays of your gates and is often called the propagation delay (50% to 50%). (Comment: The propagation delay metric will vary from different manufacturers of cell libraries. Some manufacturers will use 50% to 67% or some other value to make propagation delay characterization easier, but that is unimportant here). You can also zoom in and out of the waveforms using the same shortcut keys you used for Schematic Composer.

### 3.1.1 Questions

What is the delay through the 1st inverter? the 2nd?

What is the rise time & fall time of 1st inverter? the 2nd?

25. As a last step in SPICE simulation for the inverter, we want to do what is called a parametric analysis. This involves varying one parameter in your circuit while keeping all others constant. In our case, we can vary the value of Cload and see how that affects our circuit. In the Analog Design Environment window, go to Tools->Parametric Analysis. Enter Cload in the variable name, 0f in the From field, 200f in the To field, and 10 in the Total Steps field. Enable the Select button and go to Analysis->Start. This now starts different runs of the simulation using different values of Cload for each run. After everything is finished, you should see 10 sets of waveforms in the waveform window similar to Figure 4.
26. In the resulting waveform window, notice that as the value of Cload increases, the deviation of the output waveform becomes more significant until it begins to look like a triangular pulse train instead of a square wave. This makes sense because it will take longer to charge/discharge a larger capacitance, but the frequency of our input waveform doesn't give enough time for the gates to do a complete charge/discharge cycle. Also, notice that for all ten runs, the input waveform stays the same. This isn't remarkable given the previous discussion that ideal sources can supply any value of current and so shouldn't be affected by their load. What is important to see is that the output OUT1 of the first inverter stage also stays the same regardless of the load of the second stage. This makes our lives a whole lot easier by making sure that the problem of driving a specific load is confined to a localized part of the circuit and doesn't propagate back. This bears repeating -- the circuits that drive the input of a gate are mostly isolated from the characteristics of the gate's load.
27. The Analog Design Environment is a very powerful program, and we have only shown you some of the features that can be used by digital designers. For now, this concludes the schematic design and

SPICE verification of the inverter. (Note that the verification itself wasn't emphasized too much because in this case, it was obvious that the inverter was working by noticing that OUT1 is the complement of IN, and OUT2 is the complement of OUT1.)

28. Close all windows except for the Library Manager and icfb. When Analog Design Environment prompts you on whether you want to save the state, choose yes (you can later load this state to save you some time in configuring your run). Of course, all other designs are have to be saved before closing.

**3.1.2 Question:** if  $V_{IH}$  is 2.0V, what is the highest Cload that can be driven correctly? (meaning, yielding a correctly interpreted output waveform) What if  $V_{IH}$  is 2.25V?

### 3.2 NAND, NOR design

After finishing the design of the inverter, the next step is to design two other gates, a NAND and a NOR gate. (Comment: With the inclusion of a D flipflop, these four cells are enough to design any digital logic. Of course, only the functionality is guaranteed, and not the performance, which is the main reason commercial standard cell libraries contain a much wider variety of cells. Technically, a cell containing a NAND and a flipflop is complete and can implement any digital logic since a NAND can be configured to implement any required combinational logic).

1. Using the same steps used for the inverter, create new schematic cellviews for the NAND and NOR gates called 'nand' and 'nor'.
2. Follow the same steps to create the transistor-level schematic of a NAND and NOR gate. For the NAND gate, use a width of 900n for both the NMOS and PMOS. For the NOR gate, use 450n for the NMOS, and 900n for the PMOS.<sup>1</sup>
3. For both the NMOS and NOR gates, use input names 'A' and 'B' and an output 'Y'.
4. Create symbol cellviews for both gates. Again, to those who want a better symbol than the rectangular shape created by default, you can import the nand2 and nor2 symbols from the NCSU\_Digital\_Parts library.

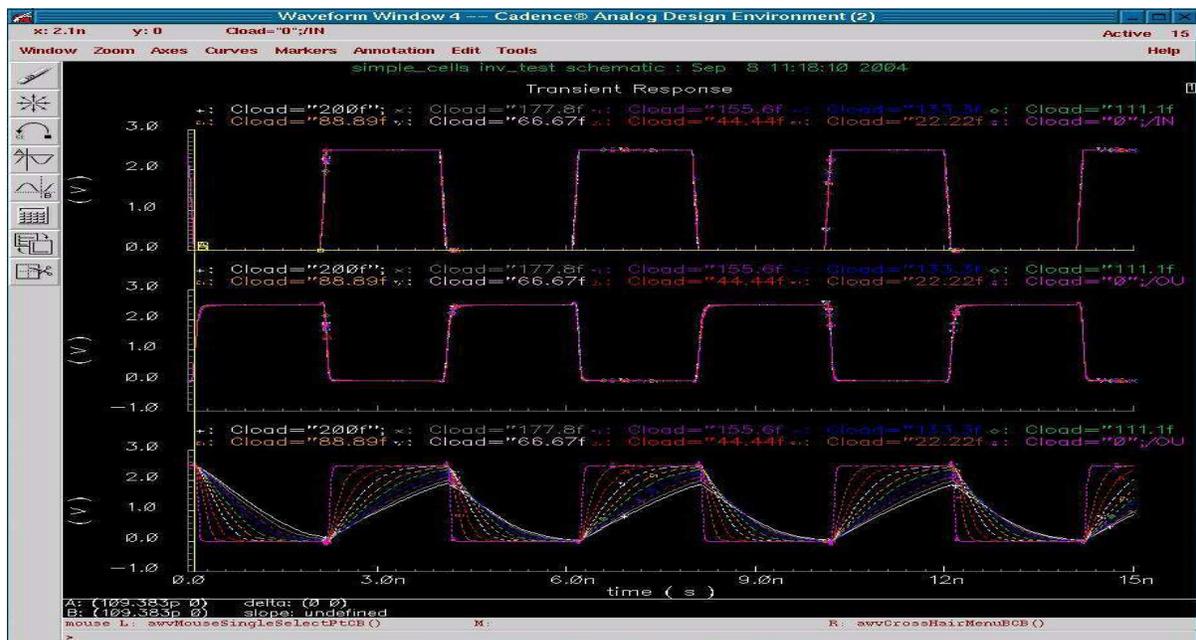


Figure 4: Parametric Analysis of the inverter testbench with output capacitance, Cload, varied from 0f to 200f.

5. After creating the schematic and symbol cellviews of the NAND and NOR inverter, the next step is to perform SPICE simulations. To do this, we have to create another testbench schematic. Create a schematic called 'nand\_nor\_test'. Instantiate a nand and a nor gate.<sup>1</sup> Also create a capacitor for each output and a gnd connection (capacitor value set again to variable Cload). Like what you did for the schematic, instantiate a 'vdc' cell with the DC voltage set to 2.5V, along with the necessary vdd and gnd connections. The next step is to supply the gates with iinputs such that all possible combinations (or more practically, all interesting and important combinations) of inputs. This is a little bit more complicated than an inverter because we are now concerned about two inputs, but is still easy enough to do since all the possible input combinations (just four of them -- 00, 01, 10 and 11) can easily be generated. To do this, we can simply instantiate two 'vpulse' sources and make sure that the frequency of one source is exactly twice that of the other one. Instantiate a 'vpulse' source with the following common parameters: Voltage1 = 0, Voltage2 = 2.5, rise time=falltime=100p. For one vpulse, set Pulse width = 2n and Period = 4n. For the other vpulse, use Pulse width = 4n, Period = 8n to generate a square wave with half the frequency (twice the period) of the first. Connect the first vpulse to the A input of both gates, and the other vpulse to the B inputs. Instantiate gnd connections for both gates and then finish up everything by connecting the nodes with wires and labeling the inputs A, B and the outputs as OUT\_nand and OUT\_nor. Your final schematic should look similar to that of Figure 5.
6. Now that the schematic testbench has been created, we can now invoke SPICE. Again, go to Tools->Analog Environment to invoke our SPICE front-end. Configure Analog Design Environment the same way you did for the inverter simulation. (Summary: Go to Setup->Simulator/Directory/Host and choose spectreS in the pulldown menu; Go to Setup->Model Path and specify the enee359a directory; Go to Variables->Copy From Cellview and change the value of Cload to 20f; Go to Analyses->Choose and make sure that 'tran' is selected, that the 'Enabled' checkbox is on, and enter a stop time of 15n. Another way to do this is to load the state from a previous run. This retrieves most of the state so that only a few of the configuration steps need to be done) Also, to plot the correct waveforms, go to Outputs->To Be Plotted->Select on Schematic and choose the nets A, B, OUT\_nand and OUT\_nor. Start the simulation by clicking on the green stoplight toolbar button.
7. If everything goes okay, the waveform window should appear. Again, maximize it and click on the 'Switch Axis Mode'. You should see something similar to Figure 6.

### 3.2.1 Questions:

What are the rise and fall times of the NAND gate?

What are the rise and fall times of the NOR gate?

8. From the output, you should be able to determine whether or not your nand and/or nor gates are working. In the case of Figure 6, both gates are working since the NOR output is "1" only when both inputs are "0", and the NAND output is "0" only when both inputs are "1". The most interesting features of the output are the output glitches in the NAND output. These transient glitches happened during the input transition where both inputs happen to be high.

---

1. When sizing transistors for standard cells, it is desirable to size them such that the PMOS network and the NMOS network have the same equivalent resistance so that rise and fall times of the output waveform will be roughly equal. For the inverter, the PMOS was made twice as wide as the NMOS, using the assumption that the carrier mobilities within a PMOSFET is only half that of an NMOSFET so that a PMOS with the same width and length as another NMOS will have roughly twice the resistance. Making the PMOS twice as wide will result in roughly equal ON-resistance and correspondingly, equal drive strength. This also holds true for more complex gates by using the concept of series and parallel resistance combinations and accounting for the worst case scenarios. For the NAND gate, since the two NMOS transistors are in series, the effective width will only be half of the two widths. By making the width 900n, the effective width is still made equal with that of a minimum-sized inverter. For the PMOS network, although the parallel combination of two 900n transistors will result in an equivalent width of 1.8um, the worst case is used where only one transistor is on such that the output resistance becomes 900n, again equal to that of a minimum-sized inverter.

1. Note that sometimes, it takes quite a bit of time for Cadence to refresh its list of libraries such that the nand and/or nor cells you created are still not visible in the Library browser. One way around this is to directly type in the name in the 'Add Instance' window, i.e. specify the simple\_cells library, nand as the cell name, and symbol as the view.

9. After making sure that your gates are working correctly, close all windows again except for icfb and Lib Manager.

### 3.3 MUX Design: creating a new cell using Gate-level design

Now that you've had some experience in creating basic cells at the transistor-level, the next step is to use the gates you've created in a gate-level design that isn't simply a testbench. The next exercise is to create a multiplexer using the basic standard cells we've created.

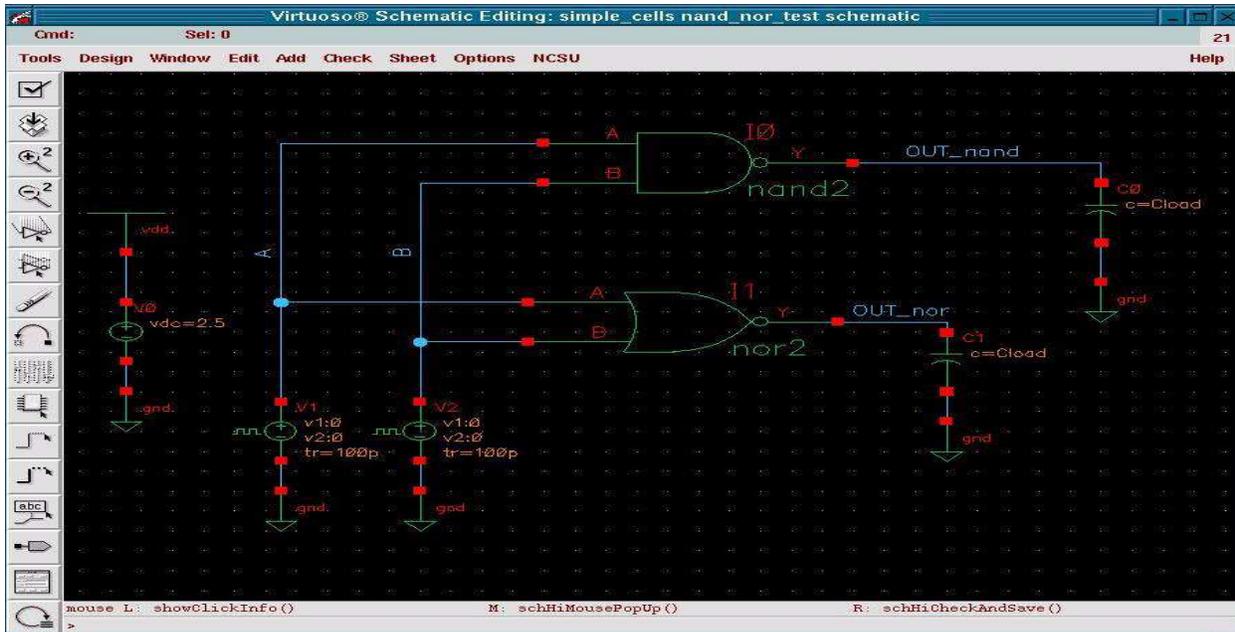


Figure 5: Schematic testbench 'nand\_nor\_test' to test NAND and NOR gates.

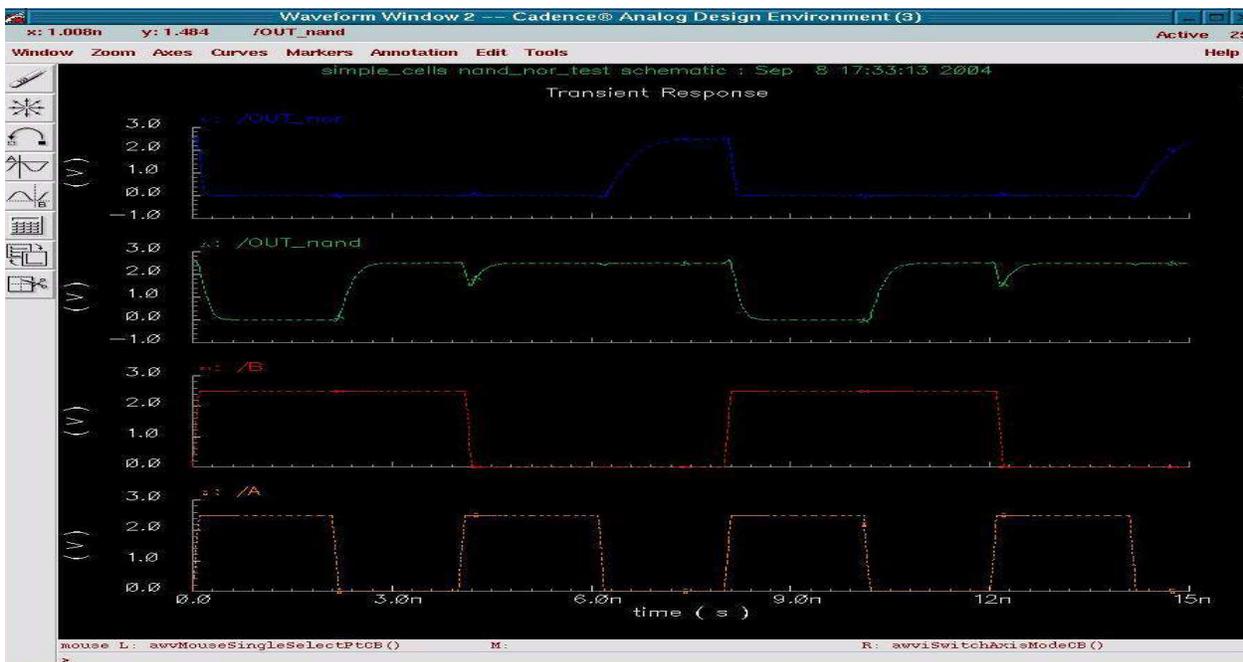


Figure 6: Output voltage waveforms for the nand\_nor\_test SPICE simulation.

1. Create a new schematic cellview called 'mux'. This gate will have three inputs - IN0, IN1, SEL, and one output Y. The following equation describes our desired multiplexer:
2.  $Y = (SEL == 0) ? IN0 : IN1;$
3. In terms of logic equations,  $Y = \sim SEL(IN0) + SEL(IN1)$
4. This equation basically states that the output is equal to IN0 if SEL is "0" and equal to IN1 if SEL is "1". In essence, the input signal SEL selects between IN0 and IN1 which signal to replicate at the output. It is up to you how to implement the logic equation using the three gates available to you. (Hint: The minimal implementation uses 4 gates)
5. After creating the circuit and the necessary input and output pins, save the schematic and then create the symbol cellview (Design->Create Cellview->From Schematic). Although a multiplexer symbol can still be imported from the NCSU\_Digital\_Parts library, some modification is needed to make it correspond with our specific pin names, so we'll just be satisfied with the default rectangular shape this time.
6. The next step is to create a testbench to test the functionality of the multiplexer and then think of a set of input waveforms that can be used to perform this testing. One easy way this could be done is to generate two square waves with distinct frequencies and use another square wave with a much lower frequency to toggle the select input. This way, the output should switch from the first frequency to the second everytime the select input changes. Open the testbench for the nand and nor and instantiate a multiplexer with a capacitive load (value = Cload). Connect IN0 to A and IN1 to B. Label the output OUT\_mux. Next, instantiate a third 'vpulse' source with the same parameters as the first two except for Pulse width (set to 40n) and Period (set to 80n). Label the output of this new 'vpulse' as SEL. Now, go to Design->Save As and change the Cell name to mux\_nand\_nor\_test. Your circuit should now look similar to the one shown in Figure 5. Close this window and open the schematic 'mux\_nand\_nor\_test'.
7. Go to Tools->Analog Environment to proceed with SPICE simulation. Configure everything the same way you did before but this time, set the analysis stop time to 100n. Also choose the outputs to be plotted as A, B, SEL and OUT\_mux. After everything is configured, start the SPICE simulation. If everything goes properly, the output waveforms should look similar to that of Figure 8. Observe that When SEL is high, the output waveform is obviously B, and when SEL is low, the output waveform is obviously A.

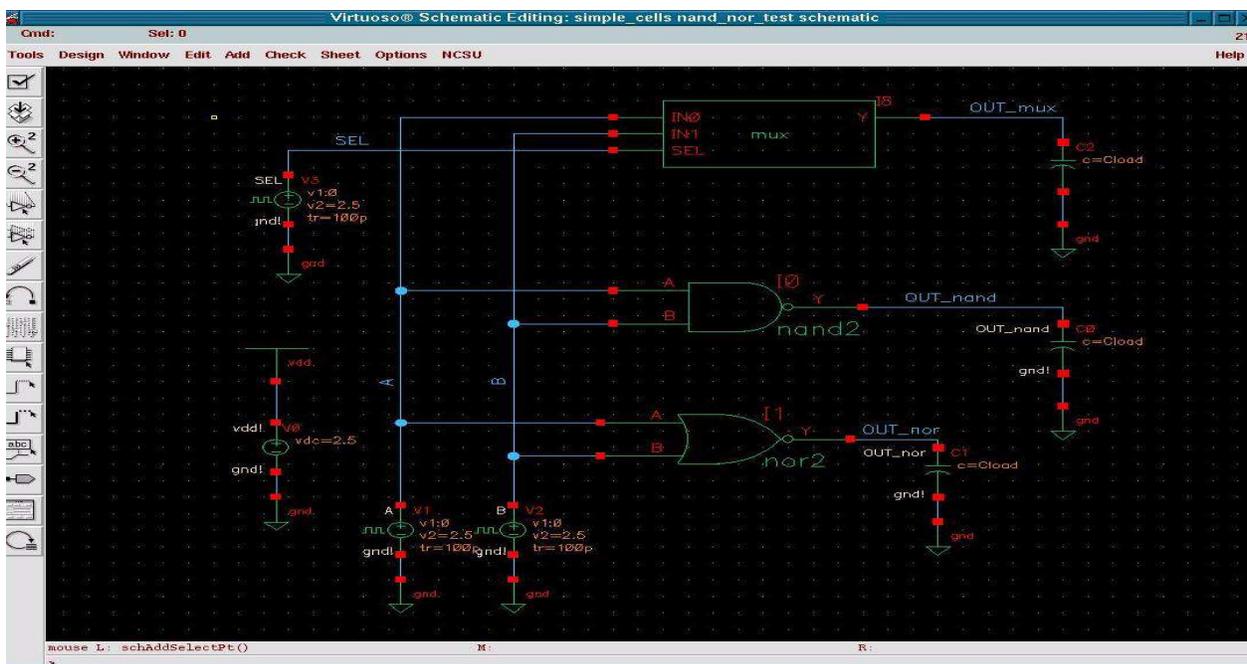


Figure 7: mux\_nand\_nor\_test final schematic.

### 3.3.1 Question: What is the propagation delay through your MUX?

8. Close all the windows except icfb and Lib Manager.

As a review of what we've done so far, we've designed inverter, nand and nor gates at the transistor-level by using schematic design software, created symbols for these gates for use in later gate-level designs, and created testbench for verification. Using these testbenches, we've performed SPICE simulation to verify the functionality of our gates and measure some metrics. Lastly, we've used the basic gates we've created in a gate level design to create a 2-input multiplexer, which we've again verified using SPICE.

At this point, little is stopping you from implementing any kind of combinational circuit by using the three gates you created as building blocks. But as designs become bigger and bigger, gate-level design becomes very cumbersome, and therefore designing at the gate-level isn't advisable for the design of most circuits. (Of course, gate level design is still very important especially when doing full-custom design or gate-level optimization of the critical paths of performance critical circuits)

## 4. 10-bit Fibonacci Counter SPICE verification

This last step for project two is to do a SPICE simulation of a big, gate-level schematic that was synthesized from a verilog RTL code similar to the ones you wrote for the first project. For later projects, you will be the one performing all the necessary steps to go from RTL Verilog to SPICE verification, but for now, everything will be given to you including the testbench.

1. Download the file enee359a\_fibo\_p2.tar and copy it into your working directory (which i assume in my case to be enee359a). Untar this archive by doing a 'tar -xvf enee359a\_fibo\_p2.tar'. This will create a new directory in your working directory named 'fibonacci'. Among the cells created will the main fibonacci cell, which contains a schematic cellview, a symbol cellview, and a layout cellview (which will be used for project3). Another cell is the fibonacci\_test cell, which is the testbench for the 10-bit fibonacci counter. To make these cells accessible from the Library Manager window, in your working directory, add the following line in your cds.lib file (without the quotes): "DEFINE fibonacci /homes/<username>/enee359a/fibonacci". One easy way to do this is to execute the fol-

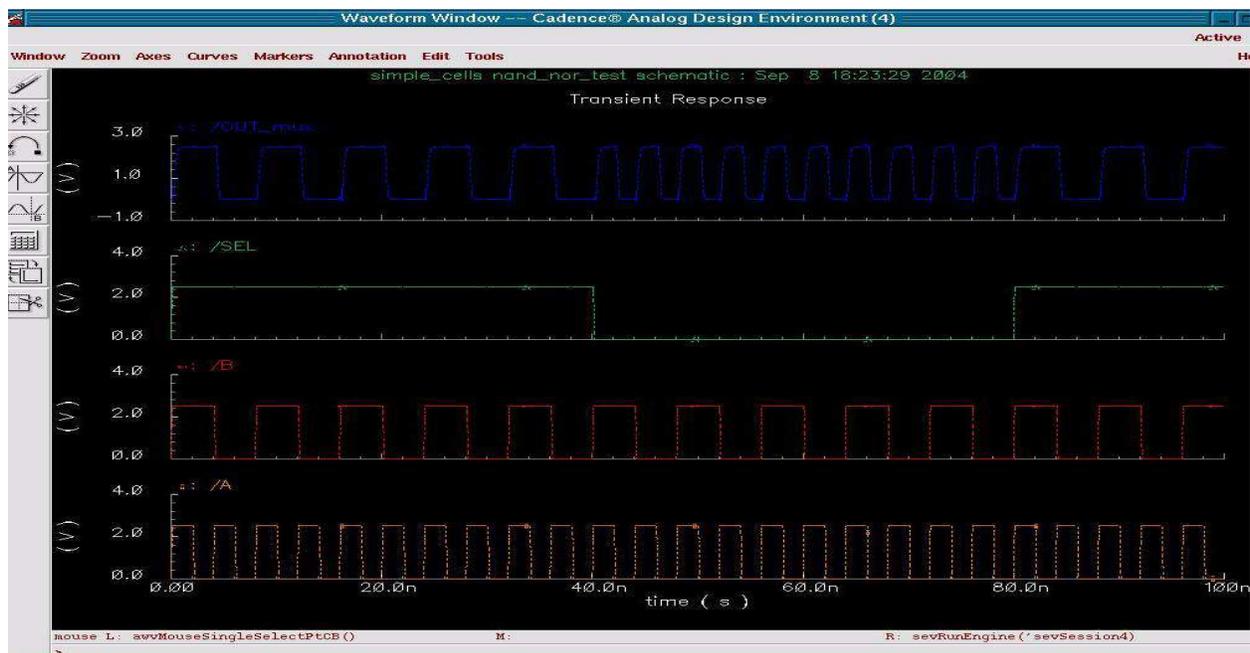


Figure 8: SPICE simulation results of mux\_nand\_nor\_test.

lowing command: `'echo 'DEFINE fibonacci /homes/<username>/enee359a/fibonacci' >> cds.lib'` and replace `<username>` with your username and `'enee359a'` with the name of your working directory. This updates the `cds.lib` file which Library Manager uses as a list of libraries to use. In Lib Manager, go to View->Refresh and then click OK on the new window that pops-up. You should now be able to see the `fibonacci` library in Lib Manager.

2. Try opening the schematic cellview of the `fibonacci` cell to see what an automatically synthesized schematic looks like. Don't worry if you can't make heads or tails out of the schematic. As circuits become bigger and bigger, it becomes harder and harder to intuitively deduce their operation just by glancing at the schematic. It's still possible, but not without spending some time figuring things out. This should emphasize the need for designing circuits at a higher level and then using CAD tools to perform conversion later on in the design process.
3. Open the testbench schematic (`fibonacci_test`) so that we can run it through SPICE. Invoke the Analog Design Environment. Configure it the same way you did for the previous runs but specify an analysis stop type of 75n. Plot the signals `clk`, `reset`, and `fibonacci_out<9:0>`. When finished, start the simulation run and if everything goes correctly, you should see something similar to the waveforms shown in Figure 9. Check that `fibonacci_out<9:0>` goes through the proper fibonacci sequence of  $\{1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, \dots\}$ . Note that the registers in the circuits are only 10-bits so they can only represent a maximum value of 1023 (unsigned). This means that after the value 987, the counter will overflow and produce garbage. This error is acceptable for our application as long as we realize this limitation and we take it into account. Since the only present purpose of this fibonacci counter is to serve as a vehicle to familiarize you with the CAD tools that are used in the VLSI digital design flow, this limitation is perfectly okay.

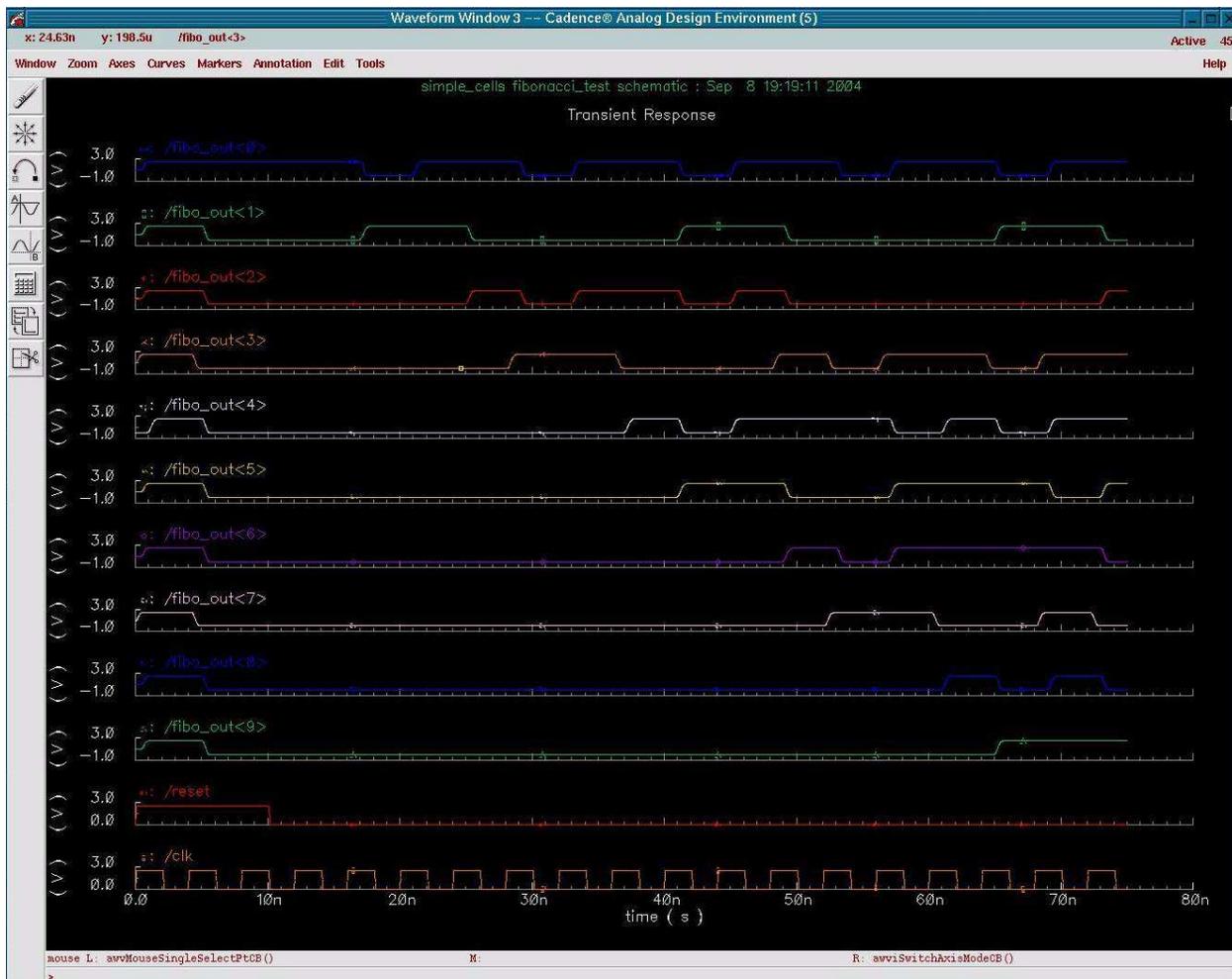


Figure 9: SPICE Output waveforms from `fibonacci_test` .