CESE4005 Lab 1

Transistors, Logic Gates, and Digital Circuits

September 11, 2024

Introduction

In this Lab the basics of CMOS logic and digital design will be introduced. If you already have experience with these topics, the Lab should fly by. However, some of the CESE students will have had less interaction with actual computer hardware at this tiny scale. To better understand why computer hardware is designed the way it is, it is essential to understand what it looks like at all levels of the technology. We will spare you device physics, but we will go into the limitations of CMOS transistors and what this means for design choices at the lowest level of circuit design.]

To do so, you will use Virtuoso, a Cadence licensed tool. To access Virtuoso with a license, you will log into the course server via X2GO using the username and password provided to you at the start of the lab. This setup is explained in the next Section.

We will start the lab by making a simple Inverter circuit. This should introduce you to the use of the Cadence tool Virtuoso, which will be used in the rest of the lab. Next, we expand this experience by making a 2-input NAND and AND logic gate. Then, we will increase the complexity and work on a Full Adder circuit, followed by a Ripple Carry Adder which allows us to add multi-bit binary numbers.

Along the way you should see that even though we like to call them digital circuits, the fundamental components and values in the circuits are all analog. All we do is try to make these analog devices behave as close to digital switches as possible.

To guide you through these observations we have added questions to the lab manual. These questions are marked in red:

Question 0.0 - This is a question.

At the end of every section, you are requested to ask a T.A. to check your work. Put all your answers/screenshots in an answer sheet (.doc, .docx, .pdf, or similar), so they can be checked easily in one go.

Bonus

At the end of Section 4, two Bonus questions are added, marked in *Green*. Answer these questions (and have them checked by a T.A.) to earn bonus points.

Setup

Server Access via VPN

The activities for this lab will be performed using Cadence licensed tools. To access these tools, you will need to ssh into the course server. To be able to do so, you will either need to be connected to the TU Delft local network (ethernet, eduroam does not work), or else you need to connect to the TU Delft VPN. If you use the VPN, you can connect to the server anywhere you want.

The TU Delft recommends using eduVPN: eduVPN download . Download this program, and use your TU Delft credentials to log in. Ensure the VPN connection is active before continuing (i.e. switch it *on* in the eduVPN window).

SSH into Server using X2GO

As we will be running graphical interfaces for this lab, it is best to use X2GO to connect to the server. Download the X2GO client (X2GO download), open it, and create a new Session. This will open a menu window as shown in Figure 1.

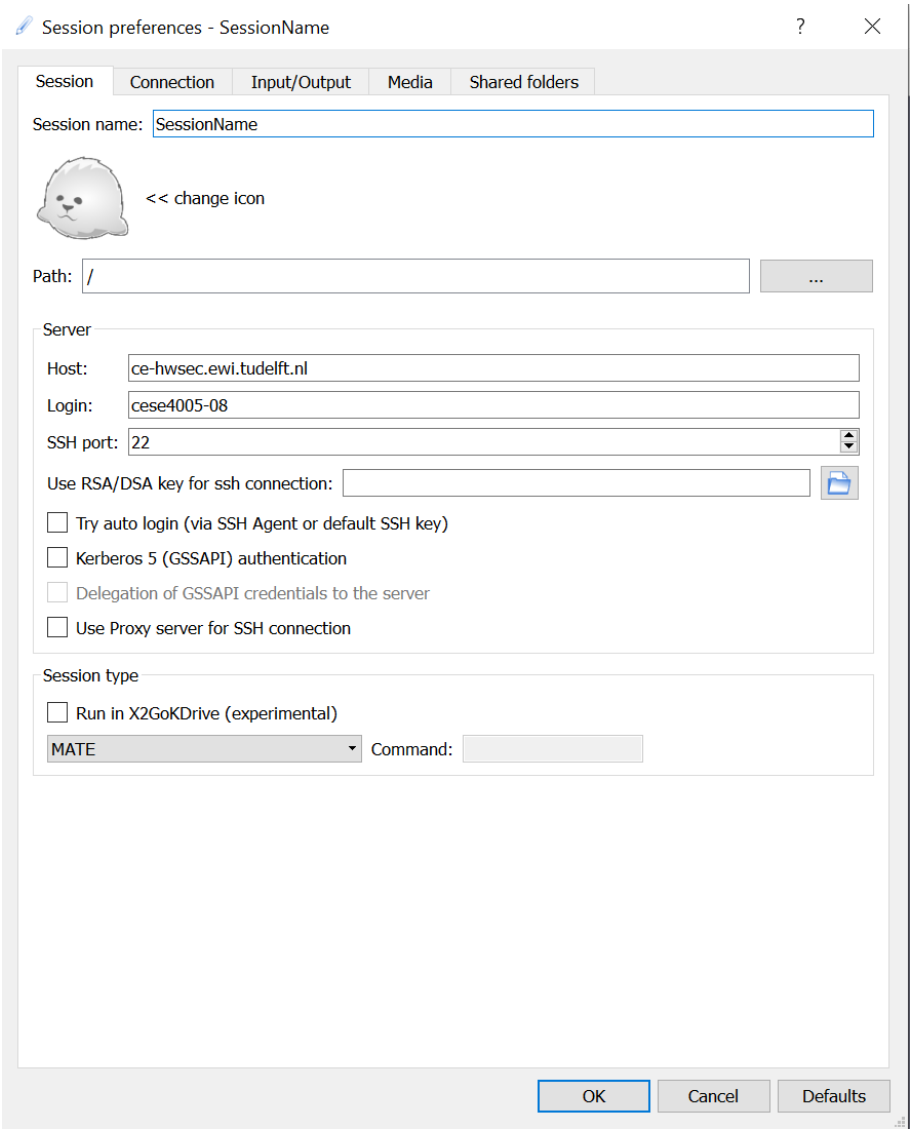

Figure 1: X2GO New Session Menu

In this menu leave all as default except:

- Under the Session tab:
  - Set the Session Name to a unique name
  - Set the Host to: *ce-hwsec.ewi.tudelft.nl*
  - Set the Login field to your provided login name (cese4005-XX)
  - Change the Session type to "MATE"
- Under the Connection tab: Set the Connection Speed to WAN

- Under the Media tab make sure “Enable sound support” and “Client side printing supported” are disabled

Press OK. This creates the Session on the right. Click the session, and log in with the provided password. This should start up a new window with graphical access to the server. At the top a black icon is present to open the Terminal.

Some Notes

- To prevent resource overuse, please properly exit X2GO when done with the lab activities. Do not leave it running, and do not just close the window. Close all programs, exit the session, and then close the window. Note that after 4 hours of inactivity, your session will be closed automatically, so unsaved work may be lost.
- X2GO can be a bit finicky when using multiple displays. When closing the X2GO window, ensure it is currently in your main display (e.g. the display of your laptop). Otherwise when starting the session again the window may open in a ghost display, with no method to get it back except to connect to a secondary display.

1 Introduction to Virtuoso by Making an Inverter

In this Section the most basic/important features of the Cadence tool Virtuoso will be shown for circuit design. Virtuoso is an all-in-one program suite for Integrated Circuit (IC) design. It is widely used in academia and industry, mainly because it allows for seamless integration between all the different tools needed for IC design.

To get you familiar with Virtuoso, you will use the Schematic Editor to design the most basic of digital circuits; the inverter. We will make a simple test setup for the inverter, and simulate it to observe its behavior.

1.1 Virtuoso

Virtuoso should be installed on the server under the provided account. In the Terminal, in your home directory (`/data/home/cese4005-XX`) run:

```
$ virtuoso
```

This will start the CIW (Command Interpreter Window), from which all tools are accessible. Press *Tools >Library Manager*. This opens an overview of all libraries available. You should see a set of pre-loaded libraries in the "Library" list. Libraries are collections of cells (components, circuits, etc.). If you click on a library, all the cells in it will be visible in the Cells list. In turn, you can click on a Cell to show the different Views available for the Cell. These Views are different representations of that one cell.

To keep organized, create a new Library by clicking *File >New >Library* in the Library Manager. Give the new library the name *lab1* and click OK. A popup will ask you about the new library's technology file. Select *Attach to an existing technology library* and press OK. From the list, select *analogLib* and press OK. This makes it so that Virtuoso knows all Cells we create in this library are supposed to reference technology information from this other library *analogLib*.

Go to the newly created library, and click *File >New >Cell View*. Fill the name "inverter" in for the Cell, and press OK. This will create a Schematic view for our new component and open it using the Schematic Editor. (Note: if any prompts pop up regarding licensing, simply press *Always* to ensure you retrieve a valid license). The Schematic Editor allows us to place instances of existing Cells and connect them together, thereby creating larger circuits.

The following is a list of keyboard shortcuts to perform the most important actions. Note that each of these actions are also available via the buttons in the GUI at the top:

i	Add instance	c	Copy
w	Create wire	p	Add pin
f	Fit design to screen	SHIFT+z	Zoom out
l	Add label for wire	CTRL+z	Zoom in
q	Show instance properties		

1.2 Building the Inverter

The basic building blocks of almost all digital circuits are CMOS (complementary metal-oxide semiconductor) transistors. These are semiconductor-based devices that perform the task of a switch, and can be scaled to incredibly small sizes with relatively little issues, hence their use in ICs. The two main types of CMOS transistors are the PMOS and NMOS, which are complementary counterparts of each other.

Place an instance of an NMOS transistor by pressing "i", selecting the *analogLib* library, and within the library selecting the *nmos4* cell. Press Enter and click to place. The schematic representation of the NMOS has four terminals: the Drain at the top, the Source at the bottom, the Gate to the side, and finally a Bulk connection. The NMOS conducts more current between its Drain and Source with a higher voltage on its gate. The Bulk can be used for biasing, but this is outside the scope of the course.

Similar to the NMOS, place a PMOS transistor above the NMOS. (see library *analogLib*, cell *pmos*). The PMOS transistor has its Source at the top and Drain at the bottom instead. The PMOS conducts more current with a lower voltage on its gate. This complementary behavior is what we use to make digital circuits.

Connect the PMOS and NMOS transistors like in Figure 2, and add labels for the Power (VDD) and Ground (VSS) to the wires as shown, as well as an IN and OUT. Note how we connect the Bulk of the NMOS to VSS and the Bulk of the PMOS to VDD. This biases the transistors optimally. When making more circuits in this lab, simply connect the Bulks ot the transistors in this way.

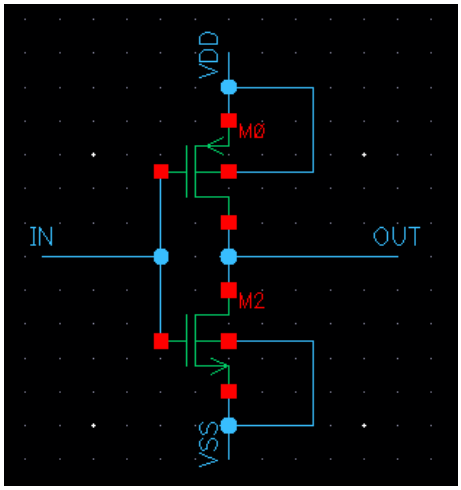

Figure 2: Basic Inverter structure.

This configuration forms a basic inverter. If the IN voltage is close to VDD (high, or '1'), the NMOS will have a low resistance and conduct current well (the switch is open). On the other hand, the PMOS will be in a high resistance state and barely conduct current (the switch is closed). As a result, the connection between VSS and OUT has low resistance, while the connection between VDD and OUT has a much higher resistance. Thus, the voltage at OUT will be much closer to VSS than VDD, i.e. low, or '0'. If IN were to be low ('0') the opposite would happen. This means the voltage on IN will be inverted at OUT.

1.3 Creating the Simulation Setup

We will now simulate what happens when we set these voltages at the Inverter terminals. However, first we want to go up one level of abstraction. This means seeing the Inverter as a standalone Cell, so we do not have to deal with all the wiring that we just did anymore; the inverter is finished.

To do so, we first need to create input/output Pins for the Cell. Press the Create Pin button,

set the Direction to *inputOutput*, and create a Pin for IN, OUT, VDD, and VSS. Now, to ensure all wiring is correct, click the Save button with the checkmark (*Check and Save*). Always press this button before progressing with any step. The result should look as in Figure 3.

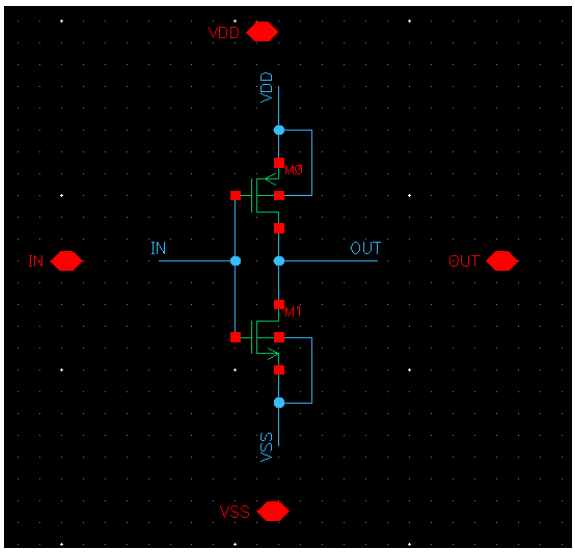

Figure 3: Inverter including Pins.

To go up a level of abstraction, click *Create >Cellview >From Cellview*. Click OK to start creating a Symbol for the Cell . In the Symbol Generation Options window, you should see fields to put Pins at the left, right, top and bottom, and you should see a list of the Pins you created: IN, OUT, VDD and VSS. Put the IN pin left, OUT right, VDD top, and VSS bottom, and press OK. This will create a box-shaped symbol for our Inverter, which will be opened in the *Symbol Editor*. You can adjust this box to make it look more like an inverter using the drawing tools, or leave it as-is. Again, always press *Check and Save*, after which you can close the *Symbol Editor* window.

In the Library created earlier, create a new Cell called *testbench*. Open the new cell, open the instance placement menu, navigate to your library *lab1*, and place an instance of the newly created inverter. This will be our device under test. We now need to take care of the connections to each Pin:

1.3.1 VDD and VSS

We need to set up a voltage source for VDD and VSS. To do so, place two *vdc* (DC voltage sources) instances from the *analogLib* library, as well as a *gnd* (ground) instance. Connect the negative terminals of the voltage sources to the ground component with wires. Additionally, make wire stubs at their positive terminals; label one as VDD and the other as VSS. By cliking on the sources, one can adjust their parameters in the Property Editor on the left. For now we are only interested in their *DC voltage*; set this to a variable *VDD* for the source connected to label VDD, and to *0* for the other source. This should look like Figure 4.

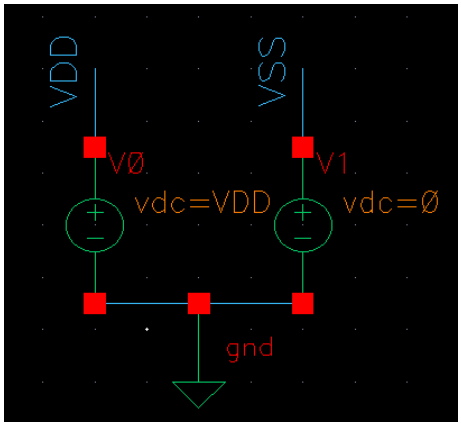

Figure 4: Voltage Sources

Connect the VDD and VSS wires to the Inverter.

1.3.2 IN

We want to stimulate the inverter using a variable voltage source, so we can see changes in time. To do this, place a *vsource* instance from *analogLib*. In its settings, ensure the *Source type* is set to *bit*; this will enable us to pass a string of '0' and '1' as a pattern which will be output by the source. Additionally, set:

- Zero value: 0 V
- One value: VDD V
- Period of waveform: 1n s
- Pattern Parameter data: 01010

Ensure the *vsource* has its negative terminal connected to the VSS net, and its positive terminal to the IN Pin of the Inverter.

1.3.3 OUT

As we want currents to flow, we cannot leave the output Pin of the inverter open. Instead, we will connect a capacitor (*cap* in *analogLib*) to the output as a test load. Ensure its negative terminal is connected to the VSS net, and its positive to the OUT pin of the inverter. Additionally, set the capacitive value to *5f* (5 femtoFahrad). This is equivalent to driving approximately 5 other logic gates. Note: keep this as the output load for this whole lab to ensure we can consistently check your results.

1.3.4 Final Test Setup

The test setup should look something like Figure 5. Make sure that for the following Sections the test setup looks the same.

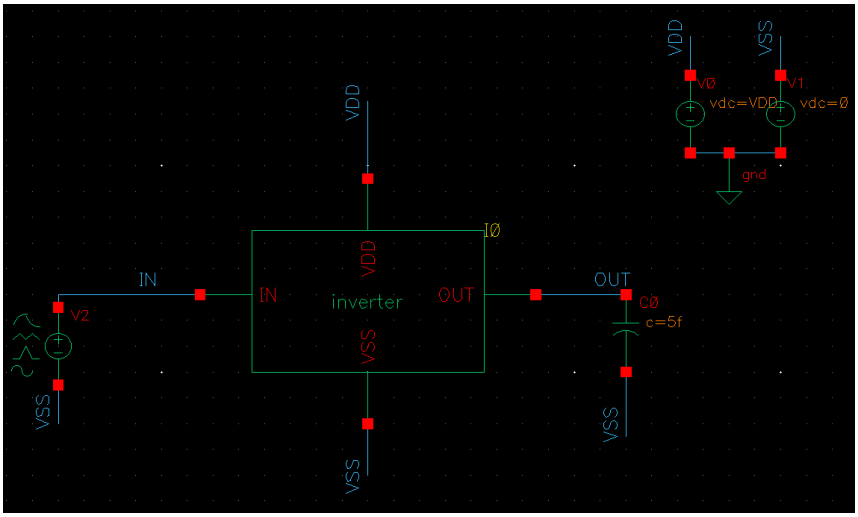

Figure 5: Inverter Test Setup

Note: Conveniently, if you want to connect components together you do not need to connect them via physical wires. Instead, if you add a bit of wire to the Pins of both components, and give them the exact same label, they will be connected as well. This can keep schematics looking more organized.

Question 1.1 - Take a screenshot of your test setup and paste it in the answer sheet. Ensure that the setup is valid by pressing "Check and Save".

1.4 Running the Simulation

To simulate, press *Launch >ADE L*. This will open the Analog Design Environment window. You may get some licensing prompts; simply keep clicking the *Always* button until they stop popping up. In the ADE L we will need to perform a few steps before running the simulation:

- Analysis Type: click *Analysis >Choose* and pick "tran" (transient) analysis, set a Stop Time of 5n for 5 nanoseconds, and set the Accuracy Defaults to conservative (simulate with highest accuracy). Press OK.
- Design Variables: Click *Variables >Copy From Cellview*. All device parameters left as variables will now appear in the "Design Variables" list. In our case, we set the voltage of the *vsource* and *vdc* to be VDD, which will be recognized by Virtuoso as a variable, so it should appear in the list. Set it to the nominal voltage for the technology we are using: 700m (700 mV).
- Outputs: Click *Outputs >To Be Plotted >Select On Design*. This will allow you to select wires to be saved and plotted from the simulation by clicking on them. We want to plot the input and output of the inverter. When done selecting wires, press escape.
- Model Files: The transistors from *analogLib* are empty models without behavior. To add behavioral information, click *Setup >Model Libraries*. Add the model file located at "/data/home/cese4005-files/7nm_TT_160803.pm"; The Model Library Setup should look as in Figure 6. Press OK.

This file contains model descriptions for the NMOS and PMOS in the 7nm FinFET technology node, which we will use as an example today.

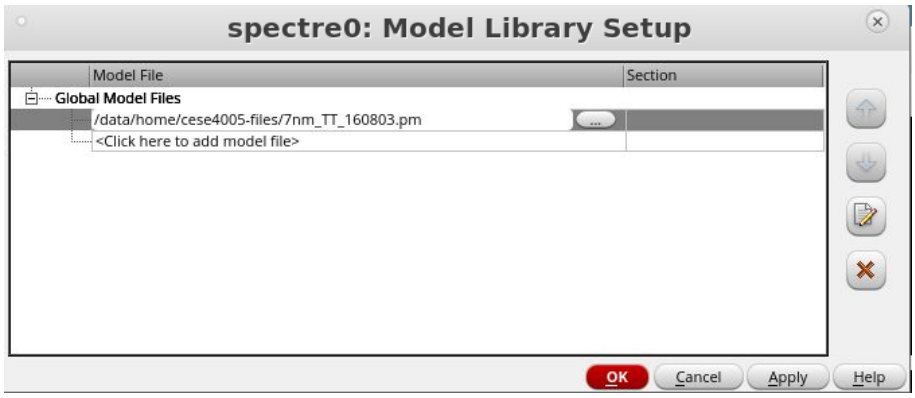

Figure 6: Model Library Setup

Press the Run Button (Green arrow) to run the simulation. A simulation log should open, as well as a Visualization and Analysis window. In the latter the waveforms for the selected signals should be visible. Take your time to explore the waveform viewer. You can select/hide signals, zoom in/out, etc.

Verify that the inverter behaved as expected; is the OUT signal an inverted version of IN?

Question 1.2 - Take a screenshot of the output waveforms and include them in the answer sheet.

Question 1.3 - Zoom into the transition point at 1ns. There should be a delay between IN changing and OUT stabilizing to a new value. This is the *fall time*. Measure how long it takes between the IN signal changing and the OUT signal stabilizing to within 1 mV of the desired voltage. Additionally, measure the *rise time* around the 2ns mark.

Question 1.4 - As observed in the previous question, there are delays betwen IN and OUT changing. What causes this delay?

Question 1.5 - What could you change in the test setup (not the inverter) to decrease this delay?

Question 1.6 - You should see a difference between the rise and fall times reported in Question 1.3. What causes this difference? (Hint: the PMOS and NMOS transistor in our inverter have the same width and length.)

– Ask a T.A to check your work so far –

2 Hierarchical Digital Design and Making a NAND and AND gate

In this Section you will expand upon the previous by building two more complex logic gates. First, you will build a NAND gate in a similar fashion to the Inverter. Then, you will combine the NAND and Inverter to create an AND gate. Finally, we will go into the typical structure of CMOS logic gates, and why this is the way it is.

Question 2.1 - Give the truth tables for the NAND and the AND gate.

Inputs		Outputs	
A	B	NAND	AND
0	0		
0	1		
l	0		
1	1		

2.1 NAND Gate

Create a new Cell called NAND. Similar to the Inverter, use NMOS and PMOS transistors to make a schematic of a NAND. You can try to come up with the structure yourself, but it is also readily available on the internet. Note: To comply with typical port namings, the two input Pins of the NAND should be called A and B, and the output Pin should be called Y. Note: Always connect the Bulk Pin of the PMOS to VDD, and of the NMOS to VSS.

Question 2.2 - Take a screenshot of the NAND circuit and put it in the answer sheet.

Now create a testbench for the NAND. You can expand upon the testbench made in the Inverter section. The only changes needed are to the inputs: Copy the *vsource* component so you can control both inputs of the NAND gate (A and B). Then, ensure the *Pattern Parameter Data* fields of the two sources are filled in as '0011' and '0101' respectively. This will ensure all possible input combinations are addressed. Now simulate the NAND gate like you simulated the Inverter.

Question 2.3 - Take a screenshot of the output waveform and put it in the answer sheet. To show all waveforms separately, press the *Split all strips* button in the top right (see Figure 7).

Figure 7: Split all strips button

Question 2.4 - What is the fall time of the output signal? You should observe a difference with the fall time of the Inverter. What is this difference and why do you think this is the case?

2.2 AND Gate

In this section you will create an AND gate by combining the NAND and Inverter you have made previously.

Create a new Cell, called AND, and place an instance of your NAND and an instance of your Inverter. Connect the two together to form an AND gate, and finish the new Cell with Pins.

Replace the NAND gate in your testbench from the previous subsection with this new AND gate and simulate it.

Question 2.5 - Take a screenshot of the AND circuit and put it in the answer sheet.

Question 2.6 - Take a screenshot of the output waveform and put it in the answer sheet.

2.3 The Complementary Structure of CMOS

With CMOS logic, we almost always connect the NMOS transistors to VSS as they are much better at propagating a low/zero voltage, while we connect the PMOS transistors to VDD as they are better suited for propagating high voltages. This is of course a simplified explanation, but it will suffice to understand this Section.

This means any gate we create has an inherently inverting nature. This means in conventional CMOS design we create Inverters, NAND, NOR, XNOR gates, etc., instead of Buffers, AND, OR, and XOR gates. To create the latter gates we typically invert the output of the former using an Inverter, like we did for the AND gate in the previous subsection.

However, this seems like quite the hassle. In this subsection we will answer the question you might have: why do we not directly make an AND gate? To do so, create a new Cell, *ANDdirect*, and build the circuit as shown in Figure 8. If the NMOS and PMOS behaved as perfect switches, this would be the smallest AND circuit you could make.

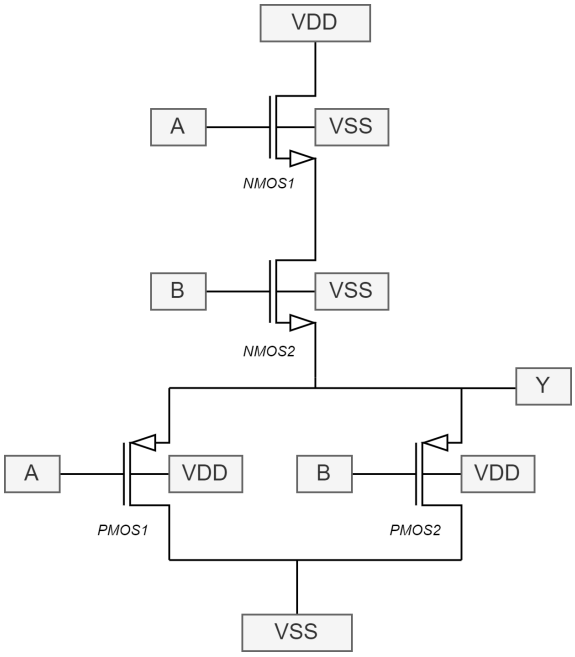

Figure 8: Diagram of (wrong) AND circuit.

Replace the AND gate in your testbench from the previous subsections with this ANDdirect Cell and simulate it the same.

Question 2.7 - Take a screenshot of the output waveform and put it in the answer sheet.

Question 2.8 - The simulation output should look much less clean than in the previous sections; explain in what way this is the case. Why would this be a problem when designing digital circuits?

– Ask a T.A to check your work so far –

3 More Complex Circuits: the Full Adder

Binary addition is a very common operation that is performed in digital circuits. The full adder is a standard digital block that sums up two operand bits and a carry-in bit, and produces a sum bit and a carry-out bit. This is illustrated in Figure 9.

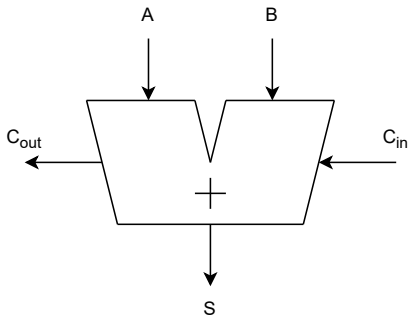

Figure 9: Full Adder symbol

In this section, the full adder is simulated on its own. In section 4, many full adders are connected to each other to perform the addition of two 8-bit numbers.

Question 3.1 - Complete the truth table of the Full-Adder

Inputs			Outputs	
A	B	C_in	Sum	C_out
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

Figure 10 shows the gate-level schematic of the Full Adder. This schematic includes 3 different gates: AND-gates, OR-gates and XOR-gates (exclusive OR).

Start by loading a library we have prepared for you. As shown in figure 10, on the library manager, go to *Edit >Library Path...* There, add a library called "lab_library" with path "/data/home/cese4005-files/lab_library" like shown in figure 12. Press *File >Save* and close the Libary Path Editor window.

Create a new schematic cell-view with the name "Full_Adder". On this cell-view, you are tasked with implementing the circuit shown in figure 10. Under the library "lab_library" you are provided with pre-designed AND-, OR-, and XOR-gates. Start by instantiating the gates you need. After

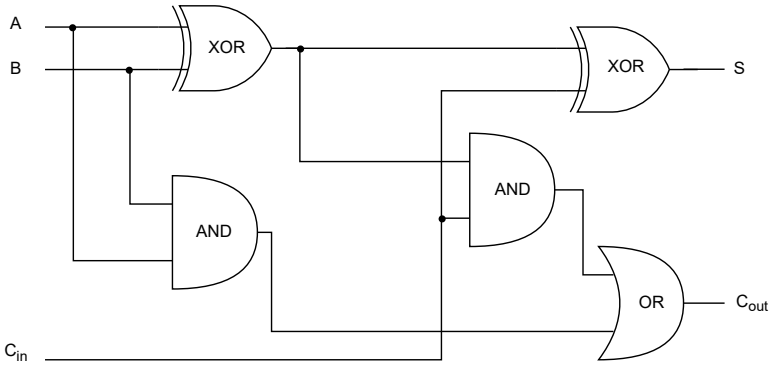

Figure 10: Full Adder gate-level schematic

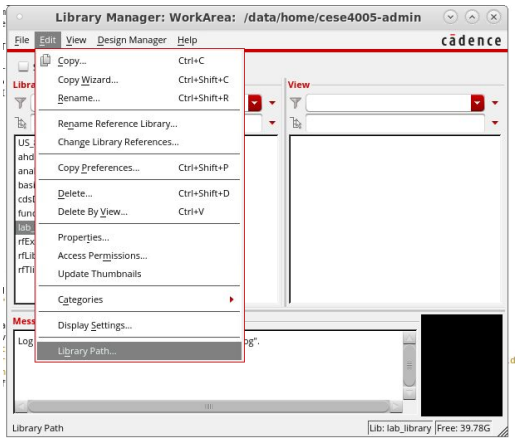

Figure 11: Opening the library path editor

that, connect them to form the circuit shown in figure 10. Then, add pins for the 3 inputs, the 2 outputs and also pins for VDD and VSS.

After finishing creating the circuit, you can then create a symbol for the Full-Adder as done in previous sections.

Due to the fact that the switching activity of transistors takes time, gates have a "propagation delay" between each input and output pair. As shown in figure 13, the propagation delay is measured between the moment of the 50% of input transition to the corresponding 50% of output transition.

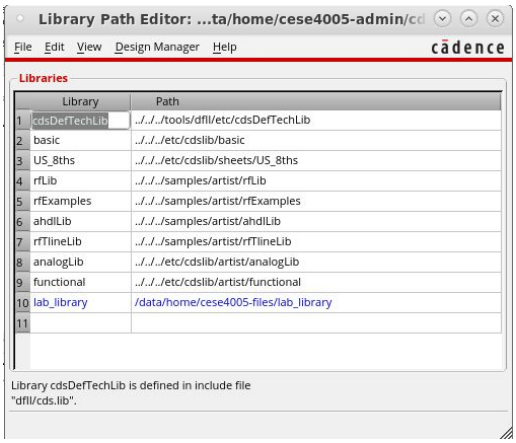

Figure 12: Adding the path to the pre-made library to the library path editor

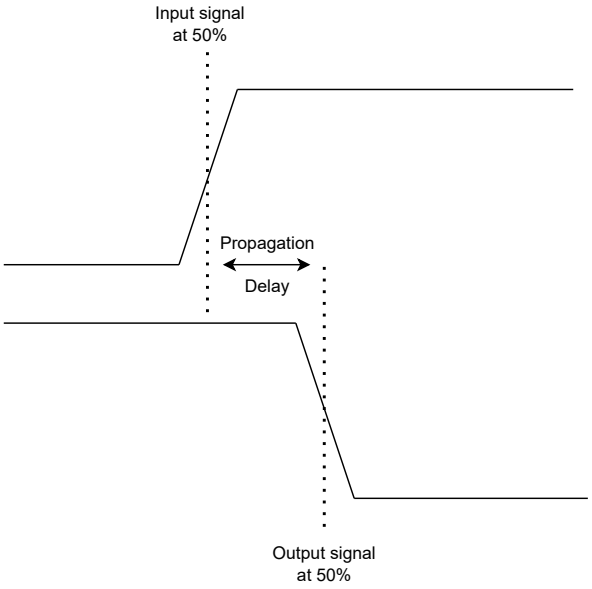

Figure 13: Propagation delay

You are now tasked with measuring the propagation delay between the Carry-in and output ports of the Full-Adder you just designed. Start by creating a new Cell-view schematic that you can call "Full_Adder_Testbench" for example. After it gets opened, you can instantiate the symbol of the Full-Adder you just created. You can connect the inputs, VDD and VSS to "vsource" from "analogLib" and load the outputs with a capacitance of 290 aF ($1aF = 10^{-3}nF$). This capacitance value is picked to replicate the loading introduced in the simulation scenario in section 4. Use ADE L then to conduct a transient simulation.

Question 3.2 - Verify the functionality of the Full-Adder by simulating a scenario that tests the 8 possible combinations of values for A, B and Carry-in and checking the values of the outputs. Provide a clear screenshot of the wave-forms.

Question 3.3 - Measure the propagation delay between the Carry-in and each of the two outputs. Load the outputs with a capacitance of 290 aF. Provide clear screenshots of the wave-forms that allowed you to measure the propagation delay, zoomed-in on the transition moments. You can press M while scrolling through the waveform to put a marker.

– Ask a T.A to check your work so far –

4 Microarchitectures and the Effect of using Analog Devices to make Digital Circuits

So far we have introduced the Full-Adder, that can add two bits with a 1-bit carry in, and provide a 1-bit sum and a 1-bit carry out. These Full-Adders can be connected in different ways to perform N-bit addition. For instance, they can be used to make a ripple carry adder, shown in figure 14.

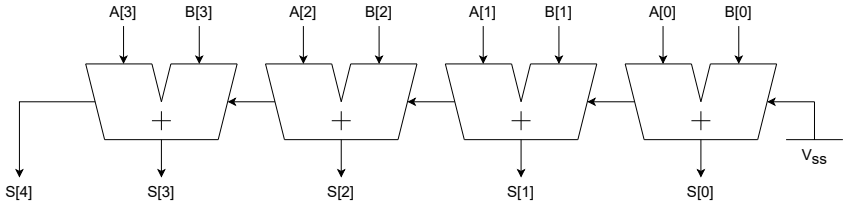

Figure 14: 4-bit ripple carry adder

An N-bit ripple carry adder is designed by chaining N Full Adders. For example, you need 4 Full Adders to build a 4-bit ripple carry adder.

In digital systems, the worst case delay is the maximum time that a change in the input of a system takes to propagate to the output. The critical path is the path that causes the worst case delay. Figure 15 shows an example of a critical path of a digital circuit. In this case, the critical path is from A (or B) to Y, since that path goes through the most gates, which leads the most propagation delay.

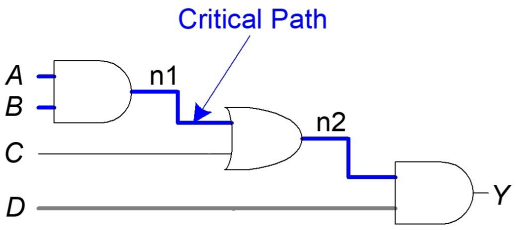

Figure 15: Critical Path of a digital circuit

Question 4.1 - What is the critical path in the circuit shown in figure 14. You should provide a reasoning using your propagation delay measurements of the full adder from the section before.

Question 4.2 - Calculate analytically the propagation delay of an 8-bit Ripple Carry Adder based on your measurements from the previous section. You may assume the propagation delay from the Carry-in to the Carry-out is the same as from A or B to the Carry-out.

– Ask a T.A to check your work so far if you don't want to do the bonus part–

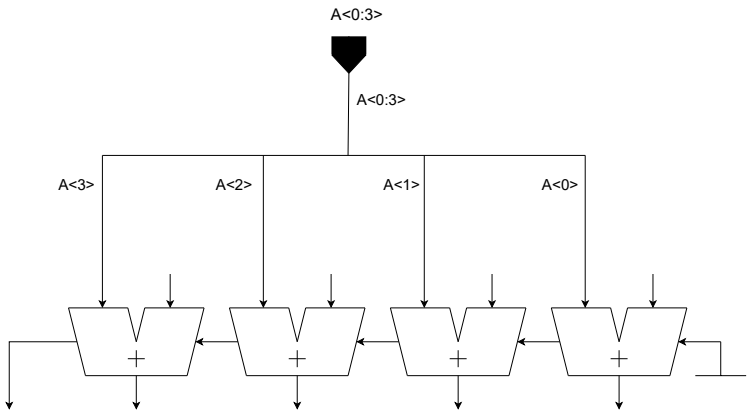

Figure 16: Multi-bit wire naming convention example

Bonus Part

You are now tasked with building and simulating an 8-bit ripple carry adder on virtuoso, to measure its propagation delay.

Question 4.3 (Bonus) - What values can you give to the inputs of the 8-bit ripple carry adder to measure its worst case propagation delay. You need to give two sets of inputs A and B in two different moments of time.

Create a new netlist that you can name "RCA_8" for example (RCA for Ripple Carry Adder). You can then instantiate 8 of the Full-Adders that you created in the section before. Connect them to form an 8 bit Ripple Carry Adder. Create pins for A<0:3>, B<0:3>, S<0:4>using the multi-bit wire naming convection shown in figure 16. Make sure to name the pin, and label the bus that is connected to the pin and the branches of the bus as shown in figure. You can create a label by pressing "l" on your keyboard, and typing the label name. If you don't name or label any one of them, the designed circuit will NOT work properly. After you finish adding the pins and labeling, press "check and save" to see if there are any error or warning messages. More information on the Multi-bit naming convention can be found here: `https://www.eecs.umich.edu/courses/eecs427/f10/Common/busnames.pdf`.

Then, create pins for VDD and VSS, and finish by creating a symbol for your designed Ripple Carry Adder.

After that, create a new cell-view where you are going to perform the simulation. Instantiate the Ripple-Carry Adder and stimulate its inputs using the answers given to question 4.3

Question 4.4 (Bonus) - Simulate and measure the propagation delay of the 8-bit Ripple Carry Adder. Provide clear screenshots that show the wave-forms you used in your calculation. Does the measured value correspond to the calculated one?

– Ask a T.A to check your work for the Bonus –