



ISTD 50.002 Computation Structures

Lab #1 CMOS

[General Information](#)

[Due Date](#)

[Receiving Credit for the Lab](#)

[Courseware](#)

[Introduction to JSim](#)

[Starting JSim](#)

[User Interface](#)

[Overview](#)

[Fig 1: Overview of the JSim user interface](#)

[Waveform Window](#)

[JSim netlist format](#)

[Part 1: Characterising MOSFETs](#)

[Setup](#)

[Fig 2: Schematic to make measurements of an nFET](#)

[Task: Making Measurements on the MOSFETS](#)

[Part A: MOSFET “on”](#)

[Part B: MOSFET “off”](#)

[Part 2: Gate-level Timing](#)

[Setup and Introduction](#)

[Defining Circuit Elements Using “.subckt”](#)

[Using Subcircuits](#)

[Shared Nodes](#)

[Symbolic Dimensions](#)

[Task: Design Issues](#)

[Part C](#)

[Part D](#)

[Task: Contamination and Propagation](#)

[Task E](#)

[Task F](#)

[Part 3: CMOS Logic Gate Design](#)

[Task G](#)

[Lab1checkoff.jsim](#)

Modified by: Kenny Choo, Natalie Agus, Oka Kurniawan 2020.

General Information

Due Date

Refer to the [course handout](#) for due date.

Receiving Credit for the Lab

To receive credit for the lab:

1. Key in the answer for Part 1 and 2 on eDimension >> Week 1 Tab >> Lab folder **(7pts)**
2. For Part 3, submit your answer online via .jsim checkoff **(1pt)**
3. Then, you need to complete Lab 1 Quiz (see course handout to find out when this will be conducted)
4. Your **effective credit** will be: $\frac{\min(\text{LabQuizGrade}, 6)}{6} \times \text{total_Lab1Points}$

Courseware

The lab software is written in Java and should run on any Java Virtual Machine supporting JDK 1.7 or higher. You can download the courseware for your Linux or Windows machine at home. You can download the courseware [here](#).

Introduction to JSim

In this lab, we will be using a **simulation program**, JSim, to make measurements of an N-channel MOSFET (or “nFET” for short). JSim uses mathematical models of circuit elements to make predictions of how a circuit will behave both statically (DC analysis) and dynamically (transient analysis). The model for each circuit element is parameterised, e.g., the MOSFET model includes parameters for the length and width of the MOSFET, as well as many parameters that characterize the physical aspects of the manufacturing process. For the models we are using, the manufacturing parameters have been derived from measurements taken at the integrated circuit fabrication facility, and so the resulting predictions are quite accurate.

The (increasingly) complete JSim documentation can be found [here](#). But we will try to include pertinent information for JSim in each lab writeup.

Starting JSim

Extract 50002.zip, open it and simply *double-click* jsim.jar.

User Interface

Overview

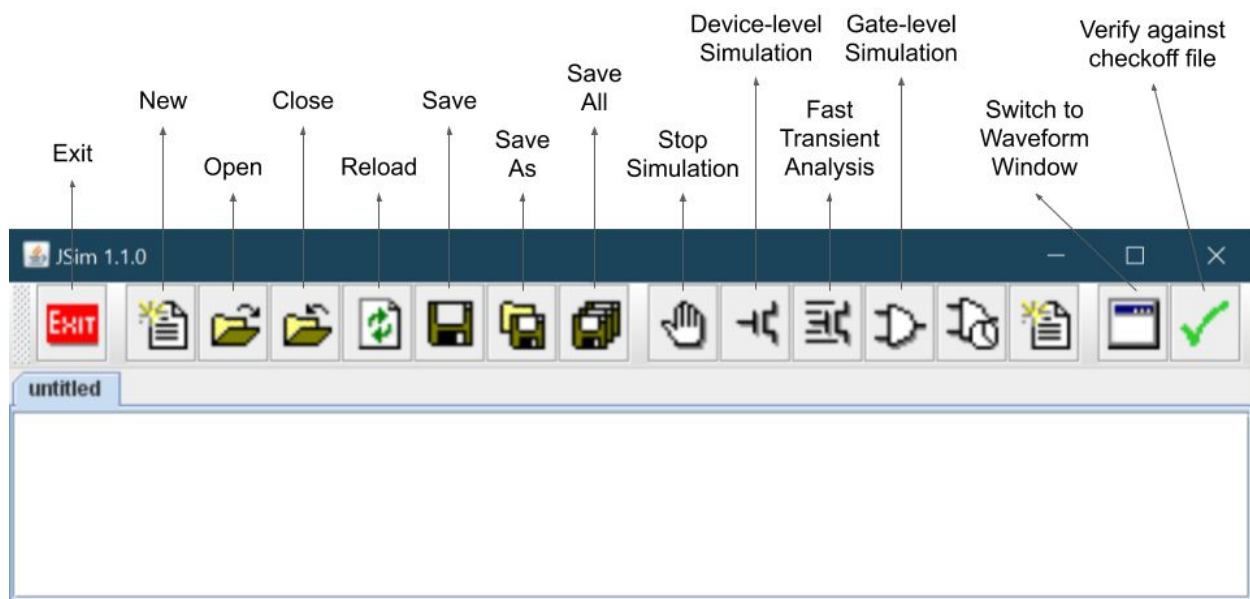






Fig 1: Overview of the JSim user interface

	<u>Stop Simulation</u> Stops a running simulation and displays whatever waveform information is available.
	<u>Device-level Simulation</u> Use a Spice-like circuit analysis algorithm to predict the behaviour of the circuit described by the current netlist. After checking the netlist for errors, JSim will create a simulation network and then perform the requested analysis (i.e., the analysis you asked for with a “.dc” or “.tran” control statement). When the simulation is complete the waveform window is brought to the front so that the user can browse any results plotted by an “.plot” control statements.
	<u>Fast Transient Analysis</u> This simulation algorithm uses more approximate device models and solution techniques than the device-level simulator but should be much faster for large designs. For digital logic, the estimated logic delays are usually within 10% of the predictions of device-level simulation. This simulator only performs transient analysis.
	<u>Gate-level Simulation</u> This simulation algorithm only knows about gates and logic values (instead of devices and voltages). We will use this feature later in the term when trying to simulate designs that contain too many MOSFETs to be simulated at the device level.



Switch to Waveform Window

In the waveform window this button switches to the editor window. Of course, you can accomplish the same thing by clicking on the border of the window you want in front, but sometimes using this button is less work.



Verify against checkoff file

Using information supplied in the checkoff file, check for specified node values at given times. If all the checks are successful, you may submit the circuit to the online assignment system.

Waveform Window

The waveform window shows various waveforms in one or more “channels.” Initially one channel is displayed for each “.plot” control statement in your netlist. If more than one waveform is assigned to a channel, the plots are overlaid on top of each using a different drawing color for each waveform. If you want to add a waveform to a channel simply add the appropriate signal name to the list appearing to the left of the waveform display (the name of each signal should be on a separate line). You can also add the name of the signal you would like displayed to the appropriate “.plot” statement in your netlist and rerun the simulation. If you simply name a node in your circuit, its voltage is plotted. You can also ask for the current through a voltage source by entering “I(Vid)”.

The waveform window has several other buttons on its toolbar:



Select the number of displayed channels; choices range between 1 and 16

You can zoom and pan over the traces in the waveform window using the controls found along the bottom edge of the waveform display



Zoom In / Out / Fit in Window

The scrollbar at the bottom of the waveform window can be used to scroll through the waveforms. You can recentre the waveform display about a particular point by placing the cursor at that position and pressing “c”.

JSim netlist format

The JSim netlist format is quite similar to that used by [SPICE](#), a well-known circuit simulator. Each line of the netlist is one of the following:

1. **Comment line**

- a. Indicated by an “*” (asterisk) as the first character.
- b. Comment and blank lines are ignored when JSim processes your netlist
- c. C++/Java style comments can also be used
 - i. “//”, all characters starting with this and to the end of the line are ignored.
 - ii. “/*” and “*/”, any lines or parts of lines enclosed by these are ignored.

2. **Continuation line**

- a. Indicated by a “+” as the first character.

- b. Treated as if they had been typed at the end of the previous line (without the “+” of course)
- c. No limit to length of an input line, but breaking long lines using “+” makes it easier to edit and understand
- d. “+” also continues comment lines

3. **Control statement**

- a. Indicated by “.” (period) as the first character.
- b. Provides information about how the circuit is to be simulated

4. **Circuit element**

- a. Indicated by a letter as the first character, that represents the *type* of circuit element. e.g. “r” for resistor, “c” for capacitor, “m” for MOSFET, “v” for voltage source.
- b. Remainder of line specifies which circuit nodes connect to which device terminals and any parameters needed by that type of circuit element. For example, the following line describes a 1000Ω resistor called “R1” that connects to nodes A and B.

```
R1 A B 1k
```

Note that the numbers can be entered using engineering suffixes for readability. Common suffixes are kilo:“k”=1000, micro:“u”=1E-6, nano:“n”=1E-9 and pico:“p”=1E-12

Part 1: Characterising MOSFETs

Make some measurements of an nFET by hooking it up to a couple of voltage sources to generate different values for V_{GS} and V_{DS} . Our end goal is to obtain the VTC plot of the NFET. Recall that we learn this in the [Digital Abstraction](#) chapter in our lecture.

Setup

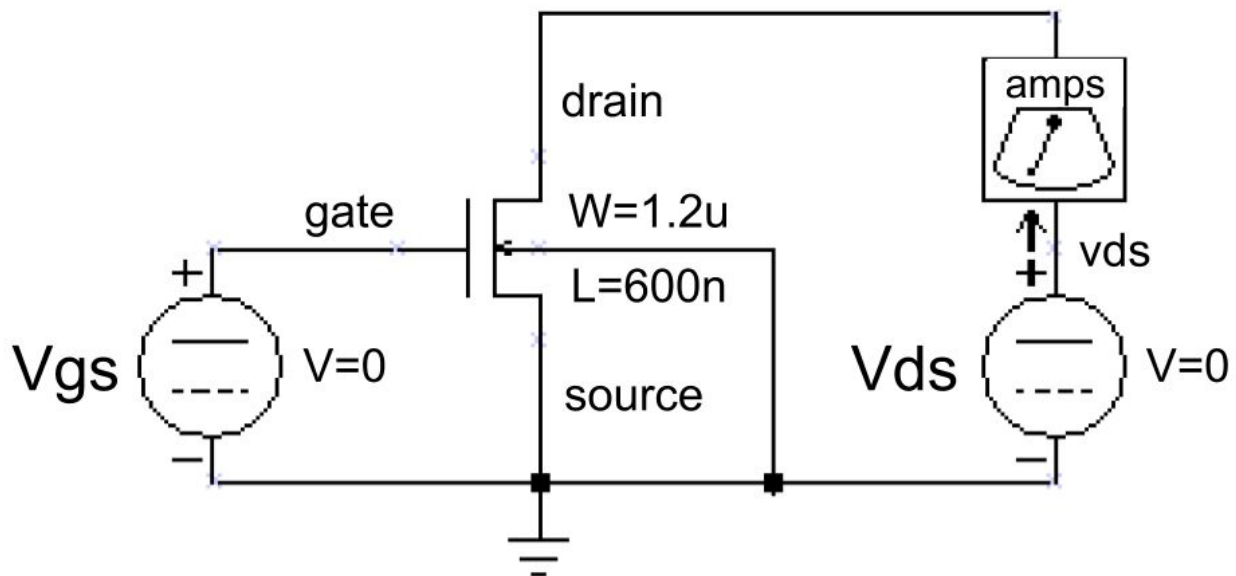


Fig 2: Schematic to make measurements of an nFET

We have included an ammeter (built from a 0V voltage source) so we can measure I_{DS} , the current flowing through the MOSFET from its drain terminal to its source terminal. Here's the **translation of the above schematic into our netlist format**:

Line	Netlist
1	* plot I_{ds} vs. V_{ds} for 5 different V_{gs} values
2	.include "/50002/nominal.jsim"
3	Vmeter vds drain 0v
4	Vds vds 0 0v
5	Vgs gate 0 0v
6	* N-channel MOSFET used for our test
7	M1 drain gate 0 0 NENH W=1.2u L=600n
8	.dc Vds 0 5 .1 Vgs 0 5 1
9	.plot I(Vmeter)

Note: the line numbers are only given for ease of referencing the description that follows below.

Line	Description
1, 6	A comment

2	A control statement that directs JSim to include a netlist file containing the MOSFET model parameters for the manufacturing process we will be targeting this semester. The pathname shown must be modified to point at where your nominal.jsim file is located.
3-5	<p>These specify three voltage sources; each voltage source specifies the <i>two terminal nodes</i> and the <i>voltage</i> we want between them. Note that the reference node for the circuit (marked with a ground symbol in the schematic) is always called “0”. The “v” following the voltage specification is not a legal scale factor and will be ignored by JSim—it is included just to remind ourselves that the last number is the voltage of the voltage source. All three sources are initially set to 0 volts but the voltage for the Vds and Vgs sources will be changed later when JSim processes the “.dc” control statement.</p> <p>We can ask JSim to plot the current through voltage sources which is how we’ll see what I_{DS} is for different values of V_{GS} and V_{DS}. We could just ask for the current of the Vds voltage source, but the sign would be wrong since JSim uses the convention that positive current flows from the positive to negative terminal of a voltage source. So we introduce a 0-volt source with its terminals oriented to produce the current sign we’re looking for.</p>
7	This is the MOSFET , where we have specified in order the names of the drain, gate, source and substrate nodes. The next item names the set of model parameters JSim should use when simulating this device; “NENH” for an nFET, and “PENH” for a p-channel MOFET (“pFET”). The final two entries specify the width and length of the MOSFET. Note that the dimensions are in microns (1E-6 meters) since we’ve specified the “u” scale factor as a suffix. Do not forget the “u” or your MOSFETS will be meters long! You can always use scientific notation (e.g., 1.2E-6) if suffixes are confusing.
8	A control statement requesting a DC analysis of the circuit made with different settings for the Vds and Vgs voltage sources: the voltage of Vds is swept from 0V to 5V in .1V steps, and the voltage of Vgs is swept from 0V to 5V in 1V steps. Altogether 51 * 6 separate measurements will be made.
9	A control statement that gets JSim to plot the current through the voltage source named “Vmeter”. JSim knows how to plot the results from the dual voltage sweep requested on the previous line: it will plot $I(Vmeter)$ vs. the voltage of source Vds for each value of voltage of the source Vgs—there will be 6 plots in all, each consisting of 51 connected data points.

After you enter the netlist above, you might want to save your efforts for later use by using the “save file” button. To run the simulation, click the “device-level simulation” button on the toolbar. After a pause, a waveform window will pop up where we can take some measurements. As you move the mouse over the waveform window, a moving cursor will be displayed on the first waveform above the mouse’s position and a readout giving the cursor coordinates will appear in the upper left hand corner of the window. To measure the delta between two points, position the mouse so the cursor is on top of the first point. Now click left and drag the mouse (i.e., move the

mouse while holding its left button down) to bring up a second cursor that you can then position over the second point. The readout in the upper left corner will show the coordinates for both cursors and the delta between the two coordinates. You can return to one cursor by releasing the left button.

Task: Making Measurements on the MOSFETS

Part A: MOSFET “on”

To get a sense of how well the channel of a turned-on MOSFET conducts, let us **estimate the effective resistance** of the channel while the MOSFET is in the linear conduction region. We'll use the $V_{GS} = 5V$ curve (the upper-most plot in the window). The equation at the linear region is given by:

$$I_D = \mu_n C_{ox} \frac{W}{L} \left[(V_{GS} - V_{th}) V_{DS} - \frac{V_{DS}^2}{2} \right] (1 + \lambda V_{DS})$$

The actual effective resistance is given by $\delta V_{DS} / \delta I_{DS}$ and clearly depends on which V_{DS} we choose. Let's use $V_{DS} = 1.2V$. We could determine the resistance graphically from the slope of a line tangent to the I_{DS} curve at $V_{DS} = 1.2V$. But we can get a rough idea of the channel resistance by determining the slope of a line passing through the origin and the point we chose on the I_{DS} curve, i.e., $1.2V / I_{DS}$.

Of course, the channel resistance depends on the dimensions of the MOSFET we used to make the measurement. For MOSFETs, their I_{DS} is proportional to W/L where W is the width of the MOSFET (1.2 microns in this example) and L is the length (0.6 microns in this example). When reporting the effective channel resistance, it's useful to report the *sheet resistance*, i.e., the resistance when $W/L = 1$. That way you can easily estimate the effective channel resistance for other size devices by scaling the sheet resistance appropriately. Since $W/L = 2$ for the device you measured, it conducted twice as much current and has half the channel resistance as a device with $W/L = 1$, so you need to double the channel resistance you computed above in order to estimate the effective channel sheet resistance.

Record down the value for the effective channel sheet resistance you calculated from that measurement. You need this information to get **Checkoff 1**. You can access the checkoffs 1 to 7 for Lab 1 through **eDimension**.

Part B: MOSFET “off”

Now let us see how well the MOSFET turns “off.” Take some measurements of I_{DS} at various points along the $V_{GS}=0V$ curve (the bottom-most plot in the window). Notice that they are not zero! MOSFETs do conduct minute amounts of current even when officially “off”, a phenomenon called “*subthreshold conduction*”. *While negligible for most purposes, this current is significant if we are trying to store charge on a capacitor for long periods of time (this is what DRAMs try to do).* Make a measurement of I_{DS} when $V_{GS}=0V$ and $V_{DS}=2.5V$. Based on this measurement report how long it would take for a .05pF capacitor to discharge from 5V to 2.5V, i.e., to change from a valid logic “1”

to a voltage in the forbidden zone. Recall from Physics II (10.005) that $Q = CV$, so we can estimate the discharge time as $\Delta t = C (\Delta V / I_{OFF})$. So if our MOSFET switch controls access to the storage capacitor, you can see we will need to refresh the capacitor's charge at fairly frequent intervals.

Record down the estimated discharge time. You will need this information for **Checkoff 2**.

Part 2: Gate-level Timing

Setup and Introduction

Defining Circuit Elements Using “.subckt”

The following JSim netlist shows you how to define your own circuit elements using the “.subckt” statement:

```
* circuit for Lab#1, parts C thru F
.include "/50002/nominal.jsim"

* 2-input NAND: inputs are A and B, output is Z
.subckt nand2 a b z
MPD1 z a 1 0 NENH sw=8 sl=1
MPD2 1 b 0 0 NENH sw=8 sl=1
MPU1 z a vdd vdd PENH sw=8 sl=1
MPU2 z b vdd vdd PENH sw=8 sl=1
.ends

* INVERTER: input is A, output is Z
.subckt inv a z
MPD1 z a 0 0 NENH sw=16 sl=1
MPU1 z a vdd vdd PENH sw=16 sl=1
.ends
```

The “.subckt” statement introduces a new level of netlist. All lines following the “.subckt” up to the matching “.ends” statement will be treated as a self-contained subcircuit. This includes model definitions, nested subcircuit definitions, electrical nodes and circuit elements. The only parts of the subcircuit visible to the outside world are its terminal nodes which are listed following the name of the subcircuit in the “.subckt” statement:

```
.subckt name terminals...
* internal circuit elements are listed here
.ends
```

In the example netlist, two subcircuits are defined: “nand2” which has 3 terminals (named “a”, “b” and “z” inside the nand2 subcircuit) and “inv” which has 2 terminals (named “a” and “z”).

Using Subcircuits

Once the definitions are complete, you can create an instance of a subcircuit using the “X” circuit element:

```
Xid nodes... name
```

where *name* is the name of the circuit definition to be used, *id* is a unique name for this instance of the subcircuit and *nodes...* are the names of electrical nodes that will be hooked up to the terminals of the subcircuit instance. There should be the same number of nodes listed in the “X” statement as there were terminals in the “.subckt” statement that defined *name*. For example, here’s a short netlist that instantiates 3 NAND gates (called “g0”, “g1” and “g2”):

```
Xg0 d0 ctl z0 nand2  
Xg1 d1 ctl z1 nand2  
Xg2 d2 ctl z2 nand2
```

The node “ctl” connects to all three gates; all the other terminals are connected to different nodes. Note that any nodes that are *private* to the subcircuit definition (i.e., nodes used in the subcircuit that don’t appear on the terminal list) will be unique for each instantiation of the subcircuit. For example, there is a private node named “1” used inside the nand2 definition. When JSim processes the three “X” statements above, it will make three independent nodes called “xg0.1”, “xg1.1” and “xg2.1”, one for each of the three instances of nand2. There is no sharing of internal elements or nodes between multiple instances of the same subcircuit. The example netlist above uses “vdd” whenever a connection to the power supply is required.

Shared Nodes

It is sometimes convenient to define nodes that are shared by the entire circuit, including subcircuits; for example, power supply nodes. The ground node “0” is such a node; all references to “0” anywhere in the netlist refer to the same electrical node. The included netlist file nominal.jsim defines another shared node called “vdd” using the following statements:

```
.global vdd  
VDD vdd 0 3.3v
```

The example netlist above uses “vdd” whenever a connection to the power supply is required.

Symbolic Dimensions

The other new twist introduced in the example netlist is the use of symbolic dimensions for the MOSFETs (“SW=” and “SL=”) instead of physical dimensions (“W=” and “L=”). Symbolic dimensions specify multiples of a parameter called SCALE, which is also defined in nominal.jsim:

```
.option SCALE=0.6u
```

So with this scale factor, specifying “SW=8” is equivalent to specifying “W=4.8u.” Using symbolic dimensions is encouraged since it makes it easier to determine the W/L ratio for a MOSFET (the current through a MOSFET is proportional to W/L) and it makes it easy to move the design to a new manufacturing process that uses different dimensions for its MOSFETs. Note that in almost all instances “SL=1” since increasing the channel length of a MOSFET reduces its current carrying capacity, not something we’re usually looking to do.

We’ll need to keep the PN junctions in the source and drain diffusions reverse biased to ensure that the MOSFETs stay electrically isolated, so the substrate terminal of nFET (those specifying the “NENH” model) should always be hooked to ground (node “0”). Similarly the substrate terminal of pFET (those specifying the “PENH” model) should always be hooked to the power supply (node “vdd”).

Task: Design Issues

Part C

To maximize **noise margins** we **want** to have the **transition** in the voltage transfer characteristic (VTC) of the nand2 gate **centered halfway** between ground and the power supply voltage (3.3V) --- *why? Ask yourself, and [review the lecture](#) on this topic.* To determine the VTC for nand2, we’ll perform a dc analysis to plot the gate’s output voltage as a function of the input voltage using the following additional netlist statements:

```
* dc analysis to create VTC
Xtest vin vin vout nand2
Vin vin 0 0v

Vol vol 0 0.3v // make measurements easier!
Voh voh 0 3v // see part (D)

.dc Vin 0 3.3 .005
.plot vin vout voh vol
```

Combine this netlist fragment with the one given at the start of this section and run the simulation. To center the VTC transition, keep the size of the nFET in the nand2 definition as “SW=8 SL=1” and adjust the width of both pFETs until the plots for vin and vout intersect at about 1.65 volts.

Just try different integral widths (i.e, 9, 10, 11, ...). Report the integral width that comes closest to having the curves intersect at 1.65V.

Record down the SW value you found. You will need this information for **Checkoff 3**.

Part D

The **noise immunity** of a gate is the smaller of the low noise margin ($V_{IL} - V_{OL}$) and the high noise margin ($V_{OH} - V_{IH}$). If we specify $V_{OL} = 0.3V$ and $V_{OH} = 3.0V$, what is the largest possible noise immunity we could specify and still have the “improved” NAND gate of part (C) be a legal member of the logic family?

Record down the noise immunity value. You will need this information for **Checkoff 4**.

Hint: to measure the low noise margin, use the VTC to determine what V_{IN} has to be in order for V_{OUT} to be 3V, and then subtract V_{OL} (0.3V) from that number. To measure the high noise margin, use the VTC to determine what V_{IN} has to be in order for V_{OUT} to be 0.3V, and then subtract that number from V_{OH} (3.0V). We’ve added some voltage sources corresponding to V_{OL} and V_{OH} to make it easier to make the measurements on the VTC plot.

NOTE: make these measurements using your “improved” nand2 gate that has the centered VTC, i.e., with the updated widths for the pFETS.

Task: Contamination and Propagation

Now that we have the MOSFETs ratioed properly to maximize noise immunity, let’s measure the contamination time (t_c) and propagation time (t_p) of the nand2 gate. The contamination delay, t_{CD} , for the nand2 gate will be a lower bound for all the t_c measurements we make. Similarly, the propagation delay, t_{PD} , for the nand2 gate will be an upper bound for all the t_p measurements.

Recall that the contamination time is the period of output validity after the inputs have become invalid. So for nand2:

$$\begin{aligned}t_{C-FALL} &= \text{time elapsed from when input} > V_{IL} \text{ to when output} < V_{OH} \\t_{C-RISE} &= \text{time elapsed from when input} < V_{IH} \text{ to when output} > V_{OL} \\t_C &= \min(t_{C-RISE}, t_{C-FALL})\end{aligned}$$

Similarly the propagation time is the period of output invalidity after the inputs have become valid. So for nand2:

$$\begin{aligned}t_{P-RISE} &= \text{time elapsed from when input} \leq V_{IL} \text{ to when output} \geq V_{OH} \\t_{P-FALL} &= \text{time elapsed from when input} \geq V_{IH} \text{ to when output} \leq V_{OL} \\t_P &= \max(t_{P-RISE}, t_{P-FALL})\end{aligned}$$

Following standard practice, we’ll choose the logic thresholds as follows:

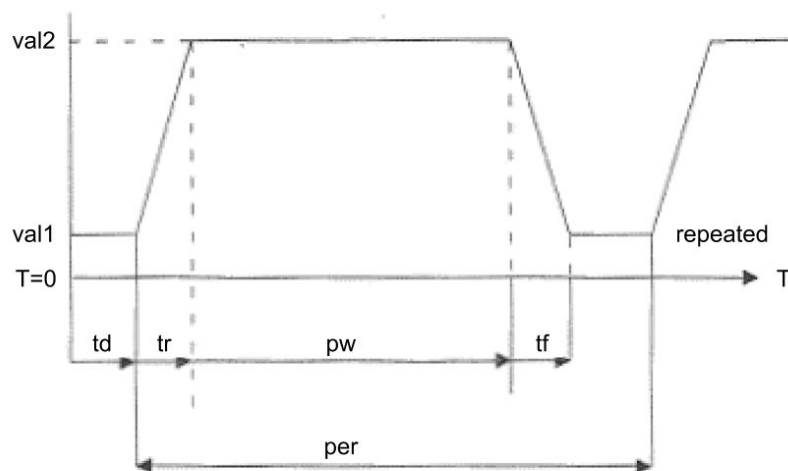
$V_{OL} = 10\%$ of power supply voltage = .3V
 $V_{IL} = 20\%$ of power supply voltage = .6V
 $V_{IH} = 80\%$ of power supply voltage = 2.6V
 $V_{OH} = 90\%$ of power supply voltage = 3V

Review the lecture on [CMOS Technology](#) to refresh your understanding on propagation delay and contamination delay.

You can use a voltage source with either a pulse or piece-wise linear waveform to generate test signals for your circuit. Here's how to enter them in your netlist:

```
Vid output 0 pulse(val1 val2 td tr tf pw per)
```

This statement produces a periodic waveform with the following shape:



Don't forget to specify your times in nanoseconds (use an "n" suffix)! **Do not specify zero rise and fall times** since the simulation will probably fail to converge. To construct a piecewise linear waveform, you need to supply a list of time,voltage pairs:

```
Vid output 0 pwl(t1 v1 t2 v2 ... tn vn)
```

The voltage will be v1 for times before t1, and vn for times after tn.

Task E

Replace the netlist fragment from (C) with the following test circuit that will let us measure various delays:

```

* test jig for measuring tcd and tpd
Xdriver vin nin inv
Xtest vdd nin nout nand2
Cload nout 0 .02pf
Vin vin 0 pulse(3.3,0,5ns,.1ns,.1ns,4.8ns)
  
```

```
Vol vol 0 0.3v // make measurements easier!
Vil vil 0 0.6v
Vih vih 0 2.6v
Voh voh 0 3.0v

.tran 15ns
.plot vin
.plot nin nout vol vil vih voh
```

NOTE: make these measurements using your “improved” nand2 gate that has the centered VTC, i.e., with the updated widths for the pFETs.

We use an inverter to drive the nand2 input since we would normally expect the test gate to be driven by the output of another gate (there are some subtle timing effects that we’ll miss if we drive the input directly with a voltage source). Run the simulation and measure the contamination and propagation delays for both the rising and falling output transitions. (You will need to zoom in on the transitions in order to make an accurate measurement.) Combine as described above to produce estimates for t_c and t_p .

Record down the contamination and propagation delays for both the rising and falling output transitions. You will need this information for **Checkoff 5**.

Task F

We mentioned several times in lecture the desire to have our circuits operate reliably over a wide range of environmental conditions. We can have JSim simulate our test circuit at a different temperature by adding a “.temp” control statement to the netlist. Normally JSim simulates the circuit at room temperature (25°C), but we can simulate the circuit at, say, 100°C by adding the following to our netlist:

```
.temp 100
```

For many consumer products, designs are tested in the range of 0°C to 100°C. **Repeat your measurements of part (E) at 100°C and report your findings. Recompute your estimates for t_c and t_p indicating which measurement(s) determined your final choice for the two delays.**

Record down your findings. You will need this information for **Checkoff 6 and 7**.

Based on your experiment, if a Pentium 4 processor is rated to run correctly at 3Ghz at 100°C, how fast can you clock it and still have it run correctly at room temperature (assuming t_{pD} is the parameter that determines “correct” behavior)? This is why you can usually get away with overclocking your CPU—it’s been rated for operation under much more severe environmental conditions than you’re probably running it at!

Part 3: CMOS Logic Gate Design

Task G

As the final part of this lab, your mission is to design and test a CMOS circuit that implements the function $F(A,B,C) = C + A \cdot B$ using nFETs and pFETs. The truth table for F is shown below:

A	B	C	F(A, B, C)
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Your circuit must contain no more than 8 MOSFETs. Remember that **only nFETs should be used in pulldown circuits** and **only pFETs should be in pullup circuits**. *Hint:* using six MOSFETs, implement the complement of F as one large CMOS gate and then use the remaining two MOSFETs to invert the output of your large gate. Refer to the lecture on [Logic Synthesis](#) for clues.


Enter the netlist for F as a subcircuit that can be tested by the test-jig built into **lab1checkoff.jsim** (a file that we supply and which can be found in the course locker). Note that the checkoff circuitry expects your F subcircuit to have exactly the terminals shown below – the inside circuitry is up to you, but the “.subckt F...” line in your netlist should match exactly the one shown below.

```
.include "/50002/nominal.jsim"
.include "/50002/lab1checkoff.jsim"

... you can define other subcircuits (eg, INV or
NAND gates) here ...

.subckt F A B C Z
... your circuit netlist here
.ends
```

lab1checkoff.jsim contains the necessary circuitry to generate the appropriate input waveforms to test your circuit. It includes a *.tran* statement to run the simulation for the appropriate length of time and a few *.plot* statements which will display the input and output waveforms for your circuit.

For faster simulation, use the  (fast transient analysis) button!

Lab1checkoff.jsim

When you are satisfied your circuit works, you can start the checkoff process by making sure your top-level netlist is visible in the edit window and that you've completed a successful simulation run. Then click on the green checkmark in the toolbar. JSim proceeds to do the following:

1. Verifies that a *.checkoff* statement was found when your netlist was read in.
2. Processes each of the *.verify* statements in turn by retrieving the results of the most recent simulation and comparing the computed node values against the supplied expected values. It will report any discrepancies it finds, listing the names of the nodes it was checking, the simulated time at which it was checking the value, the expected value and the actual value.
3. When the verification process is successful, JSim asks for your 50.002 user name and password (the same ones you use to login to your SUTD account) so it can send the results to the online assignment server. **Please use VPN if you're not connected to the school wifi. Contact IT-Care if you don't know how to use the school VPN.**
4. JSim sends your circuits to the online assignment server, which responds with a status message that will be displayed for you. If you've misentered your username or password you can simply click on the green checkmark to try again.

If you have any difficulties with the checkoff, email the course staff. Remember to schedule a lab checkoff meeting with a member of the course staff after you complete the online checkoff. For details see [Receiving Credit for the Lab](#).