

## ECE 733 Homework 2

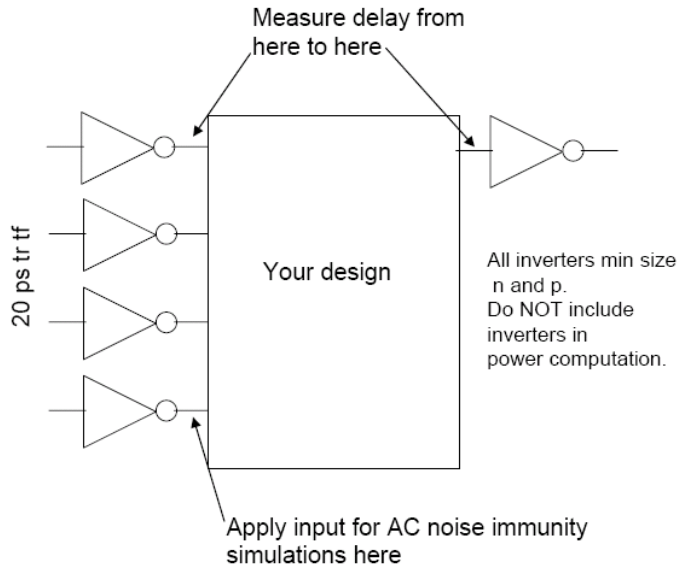
Complete the summary sheet on the next page as the first page of your homework.

### Question 1

Design combinational logic to capture the “all ones” function for a 4-bit input (i.e.  $out=1$  if all for of  $In_3, In_2, In_1,$  and  $In_0 = 0$ ). Design the logic in two different logic families – EEPL and standard domino logic (assume all inputs are rising only for the latter). Make reasonable efforts to optimize the EEPL design to minimize power consumption, and the domino logic to minimize delay. Describe how you achieved these goals. For each design, characterize the following:

- Worst-case delay.
- Average power consumed over all possible input transitions when driven by, and loaded with, minimum size inverters. (Use a 200 ps Clock period). Do not include the inverters in the power computation. Apply 20 ps edges at the inverter inputs.
- Power.delay product.
- Energy.delay product.
- For the EEPL design, determine the worst case DC noise margin.
- For the domino design, determine the worst-case AC noise immunity curve (the “low” curve only since inputs can only rise). Use triangular shaped pulses from 10 ps to 200 ps wide (at their base), and an appropriate range of voltage swings. Apply these inputs directly to the gate, not through the inverters. The pulse passes, if the output does not exceed  $V_{IL}$ , otherwise it fails – the noise is excessive. The AC Noise Immunity curve is the boundary between pass and fail. Leave the load inverter in the circuit, however.

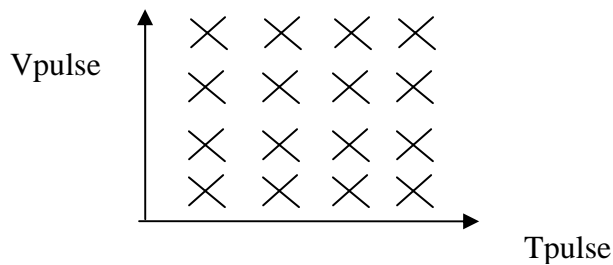
[60 points]



The inverters with min size n and pfets are included so as to normalize the input voltage drive and load for “your design”. They are NOT part of the design.

**Notes:**

In this homework, you have to design a computer experiment to answer at least one question. In order to obtain the AC Noise Immunity curve, you need to do a number of simulations with different noise pulse widths and heights. I suggest you initially perform a Uniform sample in order to get an idea, eg. As follows:



You then might need to perform a second grid of simulations on the pass/fail boundary to home in on the actual curve. There are several commands available in Hspice that simplify the conduct of such computer experiments. I encourage you to look them up in the Hspice manual and consider using them. These are attached as an appendix (a-e are relevant to this homework).

**Name:**

**Student ID:**

I certify I did not copy these circuits, nor allowed mine to be copied (sign).

**Results Summary:**

<b>Parameter</b>	<b>Delay</b>	<b>Power</b>	<b>Power.Delay product</b>	<b>Noise Margin/Immunity</b>
<b>EEPL</b>				
<b>Domino</b>				@20ps:  @ 50ps:

(Note: Energy.delay can be reported in the main body, not here.)

Show each design below, clearly labeling transistor sizes. Put the two required transient waveforms on the next two pages after that. Attach everything else required as supporting documentation.

# 1. Introduction

HSPICE installation directory:

**/afs/bp.ncsu.edu/dist/synopsys\_hspice/Z-2007.03/hspice/**

In the HSPICE installation directory, there is a lot of useful information available. The three most

***./docs** This directory contains all the manuals for HSPICE.*

File Name	Manual Title	Description
hspice_sa.pdf	HSPICE Simulation and Analysis User Guide	This is the main manual. It describes how to simulate the circuit design in a very organized way. This manual should cover everything except some further details that are not necessarily related to the simulator itself and covered in other manual.
hspice_cmdref.pdf	HSPICE and RF Command Reference	This is a dictionary-type command reference. Very convenient for a quick command look-up.
hspice_si.pdf	HSPICE Signal Integrity Guide	This manual covers anything related to signal integrity simulation, from basics to application, such as transmission line modeling and IBIS.
hspice_appl.pdf	HSPICE Applications Manual	This is another tutorial-type manual that provides application examples and explains how they works.

***./demo** This directory contains HSPICE example netlist used in the manuals.*

The directory structure and description is listed in Chapter 20 Running Demonstration Files of hspice\_sa.pdf.

## 2. Some useful functions or commands

It is not necessary to memorize all the commands. But it will certainly help if you know where to look for reference when you need them.

Note that, throughout the manual, the netlist file being simulated is called “input listing” and the output file is called “output listing”.

### a. kreset (unix command)

```
>kreset -1
```

After certain period of time, AFS will kick you out of the system by terminating your “ticket”. That is, you still have the connection to the machine you login to, but you can no longer access your AFS space, including course/research locker. To mitigate this problem, use command **kreset -1** to enter the maximum allowable

ticket holding time (1275 minutes) and avoiding being kicked out during long simulation.

### b. Running directory (HSPICE setup)

It is better to run simulation in local disk space rather than on remote volume, such as AFS or course/research locker. Not only does the remote disk space have much less quota, but it also causes slower simulation. Especially when you have a large .tran output listing to be constantly written, the simulation time would be limited to the network speed, not the CPU speed.

- If you own a machine, you can create your own directory under /local and run simulation in **/local/username/hspice-sim/**
- If you do not own a machine, **/tmp** is a great place to put simulation files. I used to use **/tmp/username/hspice-sim-date/** for simulation purposes. Notice that this **/tmp** directory would be erased by system (possibly everyday), so do make copies of files.

### c. Multi-thread simulation (HSPICE simulation)

```
>hspice_mt -mt 2 -i circuit.sp -o out.lis
```

- Use **hspice\_mt** instead of **hspice** (add **hspice** first, of course)
- **-mt** will enable the multi-thread capability
- The number **2** following **-mt** indicates the number of cores you want the simulator to utilize. If you have 8 cores, you can put up to 8 of them to work simultaneously. However, it works best if you have multiple **.alter** command or a very long netlist, such as 50,000 components.
- Following the **-i** is the input listing (normally .sp file)
- Following the **-o** is the output listing
- In case your HSPICE is not properly setup and you c, the hspice\_mt executable is located in /ncsu/hspice/hspice/linux/hspice\_mt

### d. .alter (HSPICE command)

- This command enables the simulation to be re-run with different parameters, different sweep setup, or different library
- Example: Assume you have a input listing with **.tran** analysis, right before **.end** , insert the following text

```
...  
.alter  
.param a=1 b=1 c=1 d=1  
.alter  
.param a=2 b=2 c=2 d=2  
.end
```

The simulator will :(1). compile the netlist all the way until the first **.alter**, simulate, and generate first set of output listing and **.tr0** , then (2). compile the netlist with content replaced by the text block written between the first **.alter** and the second **.alter**, simulate, and generate second set of output listing and **.tr1**, then (3). compile the netlist with content replaced by the text block written between the

second **.alter** and the **.end**, simulate, and generate third set of output listing and **.tr2**

- When using awaves or cosmoscope, you can choose which **.tr#** to display
- Use together with **hspice\_mt** command to speed up simulation
- Check out [hspice\\_cmdref.pdf](#) and [hspice\\_sa.pdf](#) for more detail
- Demo files: `/demo/hspice/apps/alter2.sp`

#### e. IF-ELSE Condition-Controlled Netlists (HSPICE command)

- **Page 64, hspice\_sa.pdf**
- Use the IF-ELSE structure to change the circuit topology, expand the circuit, set parameter values for each device instance, select different model cards, reference subcircuits, or define subcircuits in each IF-ELSE block
- Define a parameter to work as a “switch” - to switch on/off a block of netlist, which can **include .include, .lib, .parameter, .model, .subckt**, or more
- For example, assume there are several different schemes to test your circuit, instead of changing all the parameter for each scheme, it is actually better to first set up those scheme by putting them in each “block” to be switched on/off by **IF-ELSE** then use just one single parameter to toggle between them.

```
...
.param ChangeInput=1 $This is the switch
Vp IN_net 0 pwl $This is a impulse
+ Tp1 GND_Value $Start point of the pulse
+ `Tp1+Tedge` VDD_Value
+ `Tp1+Tedge+Twidth` VDD_Value
+ `Tp1+Tedge+Twidth+Tedge` GND_Value $end of the pulse
* This part defines the parameters of the impulse
.if (ChangeInput ==1)
.param Tp1=3n Tedge=200p Twidth=200p
.elseif (ChangeInput ==2)
.param Tp1=5n Tedge=600p Twidth=1n
.else
.param Tp1=7n Tedge=10n Twidth=2n
.endif
.alter
.param ChangeInput =2
.alter
.param ChangeInput =3
.end
```

- Even more powerful when use with **.alter** (see example in [hspice\\_sa.pdf](#))
- Use **.alter** and **IF-ELSE** for test bench configuration, as well as library and circuit.

#### f. Bisection analysis (HSPICE command)

- **Chapter 2, hspice\_appl.pdf**
- Analyzes setup time, hold time, minimum valid pulse width, and maximum toggle frequency

- This analysis requires the netlist to be properly setup, including a DelayTime parameter, a .measure statement, a .tran with optimization, and .OPTION AUTOSTOP to shorten simulation time
- Demo files: /demo/hspice/bisec

#### **g. W-element (HSPICE command)**

- **Chapter 3, hspice\_si.pdf**
- Create transmission line models from (1). Ideal transmission line model (2). measured S-parameter (3). Frequency-dependent table (4) build-in field-solver model, then place a W-element that refer to the specific model created
- Unlike other W-model, the field-solver model needs additional information, such as material and geometry. During the first time netlist being simulated, HSPICE will invoke a built-in field-solver to generate a W-model based on the information given, store that information in the running directory, then compile the rest of the netlist and simulate. The subsequent simulation, as long as W-model doesn't change, will only simulate the netlist without re-invoke the field-solver, which could take some time. Therefore, you can write a input listing just to generate the required W-model, then re-use the outcome everywhere.
- Demo files: /demo/hspice/twline
- Carefully follow the manual and demo files would greatly help understand W-model.

#### **h. S-element (HSPICE command)**

- **Chapter 2, hspice\_si.pdf**
- Use s-parameter to describe a multi-terminal network circuit. Equivalent to n-port in Spectre (Cadence)
- Use **.model** to define the model, then place a S-element that refer to the specific model created
- S-parameter model can be specified within the input listing, or from external files, such as CITI or TOUCHSTONE format
- Demo files: /demo/hspice/sparam