LAB TUTORIAL 2 – HSPICE

Original: F. Campi, Adapted By: A. Tino

Background: ENSC450 focuses on building digital computation systems as dedicated ICs using MOSFET transistors. To obtain an in-depth understanding of these systems and the cells which constitute their functionality, we must learn how voltages, which represent digital signals, are processed by these MOSFET devices.

Objective: We have synthesized designs using an opensource stdcell library called OpenPDK. We will now attempt to create our own library and re-synthesize the core from Lab1 to contrast results. This tutorial will guide you through SPICE simulations, using Synopsys HSPICE, exploring cells and their electrical nature during circuit computation. You will use the EDA to observe how voltages and currents in the cell transform from input stimulus to the desired output(s). Further explanations regarding the purpose of Lab2 are also provided in the Appendix of this document.

Setting up the environment

In order to build our working environment, login to your Unix machine with the credentials provided to you. Open a terminal and reserve this for working with HPSICE: Right-click the background of your screen, and select the option "Open in Terminal".

Change your directory to your working area (ensc450 folder), and create a directory called **HSPICE** i.e.

```
cd ensc450
mkdir HSPICE
cd HSPICE
```

In the HSPICE directory, we will create the following subfolders:

```
mkdir examples
```

HPSICE is state-of-the-art commercial software that we will use for educational purposes, as we have been granted a specific commercial license. Again, please note that we are not allowed to perform any industrial or research projects with these licenses.

To use HSPICE in our Unix terminal, we must set up the appropriate environment, i.e. give appropriate values to **environment variables**. Such variable will specify:

a) Where the OS will find the HSPICE executable and its supported commands
b) Where our educational licenses are for the HSPICE software

To setup the license for HSPICE, use the unix command:

```
source /etc/profile.d/ensc-cmc.csh
source /CMC/setups/FE_setup.csh
```

Note that this command must be performed for every shell that requires HSPICE.

After performing this setup command, you may check that the variables set the correct paths for the OS:

a) To verify that the system knows where the correct path to HSPICE is located:

which hspice

should provide the response: /Lnx_STC/tools/synopsys/hspice_vD-2010.03-SP2/hspice/bin/hspice

b) Verify that the system can correctly identify our license server (IP address of Synopsys license server) which we have been granted access to on our Unix machines:

echo $LM_LICENSE_FILE

should provide the response: 6056@ensc-zener.ensc.sfu.ca:7056@ensc-zener.ensc.sfu.ca.....

Note: You can use the same terminal for all the tools you wish to use, however sometimes the configurations may clash. Therefore it is advisable to use one terminal per tool, or set of tools, you wish to use. Ex: reserve one terminal for logic synthesis (phase 1), another for HSPICE simulations (phase 2), one for layout (3rd phase) etc.

Note: The same advice applies to folders (i.e. one folder per tool/design phase) within your ensc450 folder. That is, it is strongly advised to reserve one folder for each circuit you simulate. Keep a tidy file system so you may develop a clearer understanding of the course content.

In this tutorial, we will explore an example circuit definition, provided in the reference folder:

/CMC/setups/ensc450/HSPICE/examples

Part I: Building .cir files for HSPICE

I. Running HSPICE

As the case of majority of EDA tools, we will utilize ascii text scripts to provide HSPICE with instructions to simulate a circuit and observe a GUI-based waveform result. Due to the nature of this course, we will mostly be interested in **transient circuit analysis** (circuit voltage analysis over time).

As HSPICE processes "circuit" files, we will use the **.cir** extension to name our input files. Before getting started, you will require an HSPICE *configuration file* to be present in your home directory. Copy the meta.cfg file to your home folder using the following command:

cp /CMC/setups/ensc450/meta.cfg ~

In general, HSPICE will print a large textual output to the terminal (aka standard output) when are working with circuit files. Therefore it may be a good idea to redirect the output to a file which you can open and make reference to when necessary. You may redirect output in HSPICE by using the following command(s) (just given as an example here for later):

hspice example1.cir OR

hspice example1.cir |& tee -l example1.out → then you may able to view this output using:

less example1.out (or vim example1.out)

Once the hspice command executes, check the file to verify that the simulation has completed correctly. In particular, look for any warning or error messages.

II. Writing a .cir file

At the beginning, you will be asked to compose standard CMOS gates based on what is described in the course slides and the textbook. Feel free to reference a design when building your stdcell gate with the libraries available in: /CMC/setups/ensc450/HSPICE/libraries

For now, let's copy the inv.cir file found in /CMC/setups/ensc450/HSPICE/examples/inv.cir using the following command (ensure you are in your ensc450/HSPICE folder):

cp /CMC/setups/ensc450/HSPICE/examples/inv.cir .

Using a text editor of your choice (vim, gedit, emacs ….), open the file, referred to as a **SPICE deck.** Observe its contents. Circuit (.cir) files are ASCII text files that can be written with any editor. All spice commands are composed by a single line. To make our text clearer, we can write a command line in multiples lines using '+'. Adding a '+' at the beginning of the line indicates that the current line is a continuation of the previous.

A .cir files is composed of the following sections:

a) First Line
Inverter: Basic Example

The first line is always intended as comment/title for the .cir file and is ignored by hspice (so don't put useful commands here). Comments in HSPICE are denoted with a * thereafter

b) Library References

.include /CMC/setups/ensc450/HSPICE/cmosp18/rules.inc
.LIB '/CMC/setups/ensc450/HSPICE/cmosp18/log018.l' TT

To make our circuit files simpler, we can import information from other files. In our example, we **include** a "rules" file, pointing to the design information and parameters for our chosen technology (180nm), and MODELS that specify device behaviour.

Not all transistors within a given technology behave the same. Since models also specify timing and power consumption, we have the choice of specifying what "type" (referred to as "corner") we want to use in our simulation:

- **TT** means **Typical** - for timing analysis of average devices
- **SS** means **slow** devices - when performing conservative timing analysis
- **FF** means **fast** devices

Next, the **.protect** statement limits the standard output when hpsice reads the transistor model(s), and therefore we use it to avoid a very long and complex output file.

c) Power/Supply Sources

*** Supply Sources**
.param pwr=1.2
.temp 25
Vvdd vdd 0 dc pwr
Vvdds vdds 0 dc pwr
Vgnd gnd 0 dc 0
Vgnds gnds 0 dc 0

This section of the file allows us to explicitly specify and/or change the reference power supplies used in the design, as well as specify any biasing value (vdd**s**/gnd**s**) for the transistors in our cell. The node 0 conventionally represents ground in Spice, where any other voltage is a value specified with a numeric value.

A parameter (.param) is a value that we define once, and may reuse it as a variable throughout the .cir file. In this case we define the param **pwr** to specify the reference voltage as a variable.

The .temp specification is used to specify the temperature of operation of the circuit. We will use this as a standard temperature in our lab.

Next, any capital letter suffix, followed by a variable name (i.e. **V**vdd), specifies an independent element in Spice which we are defining. In the case of **Vvdd, V** signifies we would like to create an independent voltage source, with the name **vdd**, which may range from 0V dc to pwr (i.e. 1.2V). The same assignments are applicable to the remaining lines (**cards** in a SPICE deck).

d) Logic

The next section of the file defines the logic which composes the cell. In the simple case of an inverter, the cell is composed of a nmos and pmos transistor, and a reference capacitance load that our example has set to 20pF. Note it is possible to tune the strength of the inverter by varying the width of the MOSFETs (more on this during our "**Delay**" lecture).

Typically, Wm# (minimum width of a channel) is used in multiples to specify strength, and is defined in the **rules.inc** file. The default channel width for the 180nm technology we are using is 220nm.

A component may be added to a .cir file as:

Element_Name ports [library item name] parameters

As mentioned previously in the supply sources section (**V**vdd), the first letter (suffix) of the element name specifies the kind of component. Other components include: resistance which

starts with an **R**, capacitance with **C**, inductance with **L**, and MOS devices with **M**. Library items (custom items that have been previously defined) start with an **X**. In the case of a basic inverter:

Xpmos vdd a z vdd pt_st w=2*Wn#
Xnmos gnd a z gnd nt_st

pt_st and nt_st inherit from **st** (stacked]) MOS devices which have been defined in the library /CMC/setups/ensc450/HSPICE/libraries as .inc type files (files to be "included" in another design). Please refer to **hspice_macros.pdf** which describe the different transistor models. In the meta.cfg file which you copied, the variable ddlpath specifies this library folder, and may be adjusted to include other macros from your stdcells if needed.

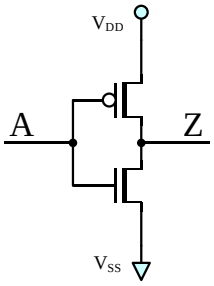
Before we proceed, open the file where the pmos and nmos macros have been defined, i.e.
For pmos: /CMC/setups/ensc450/HSPICE/libraries/pt_st.inc
For nmos: /CMC/setups/ensc450/HSPICE/libraries/nt_st.inc

Using the pmos file as an example, you will see the transistor is defined as:
.macro pt_st d g s b

Implying that we are developing a macro called pt_st, the first parameter is **d** (signifying the drain voltage), **g** for gate, **s** for source, and **b** for body. Thus when including the pt_st (pmos device) in our stdcell definition:

Xpmos vdd a z vdd pt_st w=2*Wn#

We assign vdd to the pmos drain and body connections (defined in the Supply Source section), node **a** to the gate, and connect the source to node **z**. Similarly, looking at the nmos, the drain and body are connected to ground, where node a and z are connected to the gate and source, respectively. Accordingly, we have built a circuit that resembles the following:



Note: although it is proper convention that Vdd be the pmos source, and gnd the nmos source, the inverter will still function correctly if connected to the drain instead. However it is recommended to stick to proper conventions.

A detailed description of general macros and their use are available in /CMC/setups/ensc450/HSPICE/manuals/hspice_macros.pdf.

e) Stimuli

In any EDA simulation context, the stimuli applied to a device/circuit are fundamental. Stimuli represent input variations which the logic will process and compute within a given time frame, consuming power, and producing a result. Varying the input stimuli will provide different power consumption and timing results which we may use in our library for a given stdcell.

Power, timing, and area of our circuit also determines whether we have designed an efficient circuit, and may be verified by applying different input stimuli. This stimuli however **needs to have a significant meaning** in order to accurately assess our results.

In HPSICE, we provide stimuli with **PWL (piece-wise linear function)** syntax that draws detailed digital signals with respect to time. We must carefully select the transition ramp (i.e. input transition time) of our signals as this will significantly impact the behaviour of our logic. Syntax for PWL is as follows:

Vname N1 N2 PWL(T1 V1 T2 V2 T3 V3 ….)

Vname signifying a voltage **V** type and its variable **name**, connected from node1 (N1) to node2 (N2). At t1 we would like to apply a voltage of v1, at t2, apply a voltage of v2 etc, creating our exact input transition time to voltage signal applied to the gate. As an example, let's apply a PWL stimulus to our inverter - a 1ns 0-to-1 input step connected to the INV input node a:

VA a 0 PWL(0n 0 9ns 0 10ns pwr)

This implies we have created a voltage source VA, one side connected to ground, and the other connected to node a. In terms of time, at 0ns, 0V will be output from the source to the inverter's node a, at 9ns 0V will be output, and suddenly at 10ns, "pwr" V will output.

f) Simulation Parameters Statement

After describing the circuitry in the .cir file, we need to convey to HSPICE what to do with the circuit through simulation. This may be accomplished with specific statements that describe the type of simulation and expected output from the tool. Spice supports various simulations, the most significant being **transient, ac** and **dc.**

In our context as digital designers building stdcell libraries, we are mostly interested in evaluating v(t) and i(t) curves = **transient analysis**. The syntax for transient analysis is as follows:

.TRAN TSTEP TSTOP <TSTART>

Meaning start (TSTART) and stop (TSTOP) times, simulating TSTEP units in between these two times. in our case, we have:

.tran 0.01ps 40ns START=0

Which signifies 0.01ps times steps between 0 to 40ns

g) Measurements and Waveform Analysis

Our SPICE simulation has two types of results:

i. **Waveform** – we may use this to inspect the i(t) and v(t) waveform plots generated for each node of the spice network.

ii. **Textual output** – we may use this to verify that HSPICE performed a specific **measurement** which will be available for use when the tool outputs the results.

Graphical waveforms are generated and may be very useful when debugging the circuit. It is especially useful for new users trying to make sense of the circuit's behaviour. The GUI waveform also allows the users to perform measurements as well.

To produce a file that can be read by the waveform GUI, specify the **.option post** command. This will enable a dump file with information for the waveform editor, output to a file with extension **.tr0**.

We can specify that we would like to graph the voltages of nodes a and z of the inverter using:

.graph V(a)
.graph V(z)

Once the circuit is known to the designer who tunes parameters to obtain ideal performance, it is useful to run automated simulations with an appropriate script. Once generated, it would not be convenient to inspect all these waveforms manually. The .cir file may specify a specific set of measurements, which are written by HPSICE to the standard output (terminal). More complex measurements are also possible using the **.meas** spice statement as seen in inv.cir (refer to the next tutorial for further details – "Measuring Power and Timing Parameters Using HSPICE").

Waveform Visualization:

The waveform GUI we will use in the labs is from Mentor, called EzWave. The linux command to use this tool is **ezwave &** (remembering that '&' forces Unix to launch a graphic command in the background so the terminal is free to the user for other commands). This section will help you get started with ezwave, but for further details refer to the man pages (in the GUI **Help** menu option).

If the spice simulation concluded correctly, and the **.graph** and **.option** post statement were specified, your working folder will contain a **.tr0** file once you run the command ***hspice inv.cir***. You may open this file in the waveform GUI using **File – Open** or in the terminal type **ezwave inv.tr0 &**. You can navigate the hierarchy of your circuit with the left panel, and select the waveform to display by clicking and dragging it onto the right panel. You may amalgamate the waveforms onto a single panel, or use different panels for clarity (drag wave to the top or bottom of the pre-existing open panel(s)). Do not use the same panel for different physical entities, such as current and voltage.

Note: We have assumed that we can describe any transistor with the macros pt_st and nt_st. This is an approximation, as standard cell layout transistors are not designed in the same manner. The above macros are inflexible in conveying differences in design. Specifically, CMOS logic has **stacked** configurations when two or more transistors of the same kind are connected in series, ex: the NMOS in NAND, or the PMOS in NOR. This allows us to build transistors on the same active diffusion without the need to specify contacts

between them. Consequently, the smaller the transistors, the smaller the parasitic resistance and capacitance. **It is mandatory that stacked transistor models are used in your assignments only when necessary**: recognizing and using stacked transistor macros in place of common (srd/drain) transistors, your circuit will have better performance, and results will be more accurate, as discussed in your lectures.

Detailed documentation on how to model stacked transistors, and other layout-specific features is available in the hspice_macros.pdf

Please reference your lectures when reading the remainder of this document – it will be much easier to comprehend

Many examples and tutorials on SPICE and HPSICE are available online, and in Chapter 8 of your VLSI textbook. You are encouraged to explore and suggest alternative methods for your assignment.

Part II: Measuring Power and Timing Parameters using HSPICE

In the previous section, we learned how to build/describe circuit (.cir) files for HSPICE. In this section, we will learn how **to perform measurements for circuit file simulations**. The SPICE command **.meas** offers complex syntax that is capable of performing most measurement tasks that we require to characterize circuit behaviour. For complete syntax, please refer to the HSPICE reference manual. The following presents a few relevant cases.

TRIG and TARG represent criteria for defining the start (TRIGGER) and end (TARGET) of the measurement period:

.MEAS <AC|DC|TRAN> var_name [func var] [TRIG] [TARG]

Example 1: Measuring Power

Based on the transient analysis selected, we can determine **dynamic power** (if analysed during the transition period), or **static power** (if analysed outside of the transition period) with the track:

.meas tran var_name AVG POWER FROM=xns TO=yns

Where AVG POWER specifies the [func] and [var] respectively. For example, for an inverter whose input is transitioning between 0 and 1 during 9 to 10ns, we can specify:

```
.meas tran leak_pow_in1  avg power  FROM=6ns   TO=8ns
.meas tran leak_pow_in0  avg power  FROM=12ns TO=14ns
.meas tran dyn_pow       avg power  FROM=9ns   TO=10ns
```

Which abides by the command and parameters specified above. Add this to your inv.cir file prior to the .end statement. You will see these measurements output to stdout when HSPICE is run.

Example 2: Measuring Timing Properties

In order to determine propagation delay, we need to measure timing from the time of the input transition (a > Vdd/2) to the output transition (Z < Vdd/2 for an inverter). To express this concept, we need a complex representation of TRIG and TARG, versus the FROM and TO cases used above for power. We may use the following:

.meas <AC|DC|tran> var_name TRIG TARG

Where TRIG (or TARG) => TRIG var_name VAL = 'value' <FALL|RISE|CROSS>

In the case of our inverter, we would like to measure **propagation delay from the rising edge** of input 'A' to the falling edge of output 'Z'. We can express this as:

.meas tran trise TRIG v(a) val='pwr*0.5' cross=1 TARG v(z) val='pwr*0.5' cross=1

If we would like to measure the **output delay from the falling edge** of output 'Z':

.meas tran trise trig v(z) val='pwr*0.2' cross=1 targ v(z)='pwr*0.8' cross=1

Similar to power, the results of the measurements are provided in the output of HSPICE.

Sweeping Parameters

There are times when one would like to evaluate the impact of varying a given parameter with respect to power and/or timing. As an example, let's try to evaluate the impact of varying the driving strength of an inverter over time and power. We can assume that increasing the channel will change the Current/Voltage (IV), such that the signal will propagate faster, but at a higher cost in power. Let's try to prove this assumption:

We can force HSPICE to repeat the same simulation, and only changing one parameter by adding the option **SWEEP** to the .**tran** command. For example, we can define the parameter Wn (width of the N-Channel) while imposing the P-Channel=2*N-Channel:

```
.param Wn=Wm#
Xpmos vdd a z vdd pt_st w=2*Wn
Xnmos gnd a z gnd nt_st w=Wn
```

The # implying a flexible parameter, where Wm# is specified in the rules.inc file. We may repeat 4 simulations and measurements for 4 different driving strength values i.e. from Wn# to 4*Wn# (where Wn is the minimum N-Channel allowed by the design rules, specified in rules.inc) with the sweep command:

.tran 0.01ps 40ns START=0 sweep Wn START=Wn# STOP=4*Wn# STEP=Wn#

In your inv.cir file, uncomment the sweep line above (do not include the #), and comment out the previous .tran line i.e. .tran 0.01ps 40ns START=0. Re-run the hspice inv.cir command. Results may be extracted from ezwave or the output of the hspice command.

Many examples and tutorials on SPICE and HSPICE are available online. You are encouraged to explore and suggest alternative methods for your assignment, in addition to the example folder provided.

Part III: Building Component Libraries (sub-circuits)

Most of the digital circuits that we use in our designs are rather complex and may be hierarchical designs. This means that the designs are built from blocks and sub-blocks, and when interconnected, provide the desired functionality of the circuit. Consider a multiplexer or memory cell: the composition of such a cell can consist of NAND gates, inverters, tri-state buffers and/or transmission gates. If we were to describe each cell as a single transistor, our circuit files would be too complex and likely unreadable.

To ease the design phase, it is very convenient to create a library of cells so that we may reuse them later in other designs. We may also want to adjust certain parameters for specific transistors in our cells when reusing the component, and therefore should design our library of components in a parametric manner (although this may not always be possible as shown below).

Libraries are quite simple to build and reuse with HSPICE. Although there are several options, we will cover one option in this tutorial. Reusable sub-blocks are called **sub-circuits** in HSPICE. A sub-circuit has a name, and associated ports (input/output pins). A sub-circuit can be in the same .cir file as the rest of the circuit, or could also be created in a different file and instantiated in the file. The second option is highly recommended as this makes a sub-circuit easier to reuse.

For component reuse, in the case of the inverter: The inverter circuitry must be specified in a single file named with the suffix **.inc** (include). The file's name must be the same as that of the actual sub-circuit. Check the folder /CMC/setups/ensc450/HSPICE/libraries to see many more examples. Let us take the inverter file **iv.inc** as an example, located in the libraries folder. Opening the .inc file, we see the SPICE deck for subcircuit contains the following:

```
.subckt IV  a z  vdd  vdds  gnd  gnds  wp=Awd#  wn=Awd#
Xpmos vdd a z vdd pt_st  w=wp
Xnmos gnd a z gnd nt_st w=wn
.ends
```

To include the inverter in a buffer circuit in our main circuit, we would first need to specify where the library components are located so that HSPICE may read them. This is accomplished using the command:

Option search = "Path_name"

You are strongly encouraged to build your own library components by adding the ".option" search command in your .cir file (see the examples folder files inv_lib.cir and mux_test.cir for a template).

Including .cir Files

A subcircuit may be instantiated in the main .cir file used for simulation as:

```
Xiv a z  vdd  vdds  gnd  gnds IV wn=Awd#
```

Where Awd# is the width passed to the subcircuit. If the subcircuit IV is specified in a different file being run in HPSICE, it must be included in the main circuit file as:

.include <otherfile.cir>

The **.include** command will simply force HSPICE to read and process <otherfile.cir> before the current file, which is equivalent to copying and pasting the content of <otherfile.cir> in the current file.

As mentioned, it is also possible to define circuit files within the same circuit file, or have a second .cir file. We will take a buffer, consisting of two inverters as a full example of a sub-circuit creation and instantiation within the same file. Please refer and copy the example located in: /CMC/setups/ensc450/HSPICE/examples/buf.cir which also sweeps increasing transistor widths.

Part IV: Building a Combinational Liberty (.lib) File

Circuit-level SPICE simulations are an essential tool to describe and characterize circuit behaviour. However due to the size and complexity of modern digital circuits, this may not be directly applicable to our designs. As we saw in the first lab, VLSI designs are described in HDL (VHDL/Verilog) and synthesized by automated tools to form a complex netlist of elementary cells.

Starting from data derived by SPICE simulations of elementary cells, we derive the timing and power behaviour of our complex VLSI design. Our synthesis tool, Synopsys dc_shell, requires the power/timing information of our cells to appropriately select the cells that should be utilized in synthesis – the synthesis process is not simply logic steps, but rather timing/power driven.

Spice-level information of cells is processed by the synthesis tool which converts our HDL to a gate-level netlist. The information required is contained in a **liberty file.** Liberty files are ASCII files that contain all information necessary for dc_shell to translate **behavioural** HDL to a gate-level netlist.

If we want to build a stdcell library with SPICE, and synthesize our HDL code using this library, we must **build a liberty file**. The rest of this tutorial will guide you to generate a definition for a simplified **liberty file**.

Please refer to /CMC/setups/ensc450/HSPICE/lib_Templates/spicelib_180_tt_120_25C.lib for syntax, keeping in mind that your numbers will be different from the contents in the file.

Lookup Tables

Let's consider a given standard cell, an inverter for example with a driving strength of 1, called INVX1. We need to provide the synthesis tool with the following information:

1. Area
2. Propagation delay
3. Leakage power
4. Dynamic power

Area: The area is fixed information, that is, it does not depend on the context of where the cell is being used. On the other hand, to precisely determine the cell area, we would need to perform a layout. For the sake of simplicity, we will simply approximate area with the number of transistors that compose the layout. Area is expressed by default in micrometers squared (um^2), meaning that one transistor is also measured in (um^2). In the case of cells with greater driving strength, we multiply area by the driving strength. **Note that this is a gross approximation.**

Propagation Delay: We define the propagation delay of a cell as the distance in time from when a cell's input crosses Vdd/2, to the time when the output crosses Vdd/2. We can briefly deduce the timing behaviour of a given cell with a SPICE simulation, and realize that it is strongly dependent on:

i. The transition of the input signal, and
ii. The load capacitance, that depends on the parasitic capacitance of the wire, its own diffusion capacitance, and all gate capacitances of the gates which drive the cell.

When dc_shell is building the given target design, it will have all this information and perform the necessary calculations, but at this stage we don't have such information as we consider each cell individually. What we require is a reference case to perform a SPICE simulation and calculate a delay. dc_shell will start from these reference values to **interpolate** the correct delay. This is an approximation, the more reference values we use, the closer the approximation. For simplicity of this lab, we will select two input transition (Ttrin) values and two capacitive load (Cload) values.

With these SPICE simulations, we can then fill in a simple LOOKUP TABLE, which will be used to determine propagation delay for two input transition values, and two capacitance load values. For the moment, we will neglect power factors and focus on the calculation of propagation times. The following is a snippet from the spicelib_180_tt_120_25C.lib file in the /libraries folder:

```
#Writing LUT templates
lu_table_template(Propagation_Delay){
      variable_1: input_net_transition;
      variable_2: total_output_net_capacitance;
      index_1("1 2");
      index_2("10 20");
      values(……);
};
/*end lu_table*/
```

To obtain this table, we need to sweep both Cload and input transition in our simulation environment for the inverter and fill in the values section. Please refer to the file comb_stdcell_liberty2.cir in /CMC/setups/ensc450/HSPICE/examples as a reference, and the appendix on how to read the above lookup table. Also, ensure to go through the spicelib_180_tt_120_25C.lib file line by line. You may use this file as a template to build your own .lib file for your stdcell library.

Note: if dc_shell needs to calculate Cload for every cell in the design, we would need to add a 5th piece of information to the liberty file (i.e. in addition to area, propagation delay, leakage power, dynamic power), which is the input capacitance of our cell.

In this case, we could use a simple SPICE calculation to determine input capacitance, but for the sake of simplicity we will use a standard value of 2pF for each transistor that the gate is driving, multiplied by the driving strength of the cell. Therefore, if our cell is an INVx2 (2x strength) cell, the input capacitance is estimated as 2*2*2 = 8pF (i.e. (strength)*(transistor_gateC*unit_cap(pf))).

Appendix: Lab2 Explanation of Terms

The **main purpose** of lab2 is to design a liberty (.lib) file, representative of the stdcell library you have designed using HSPICE. The .lib will be translated to a .db file using lc_shell, which will be included in your synth.tcl file (versus the nangate_..._slow.db file). You will use your new .db file to re-synthesize your pre-synthesis lab1 code and compare the results.

.lib file structure

The .lib file itself consists of several Look Up Tables (LUTs) which resemble the following structure:

```
cell_fall(Propagation_Delay) {
index_1 ("1, 2");
index_2 ("2, 10");
values ("0.106,0.246","0.231,0.437");
```

This example provides a template for a given cell's tpdf, one of many LUTs contained in the lib file (each table specifying a parameter for a given gate). You must fill out the "values" row, specifying your cell's tpdf. These values are obtained using the .cir/.inc files, changing the input transition times and Cload such that you can sweep each gate in HSPICE to fill in the LUT "values". The dc_shell tool will then read and use the .lib LUT information you provide to determine the best gate(s) to synthesize your design.

In this example, the LUT describes the falling propagation delay for a given cell. Index_1 refers to input transition time, and index_2 to the capacitance load. Accordingly, the value 0.106 signifies a 0.106ns tpdf for the given cell when the input transition time = 1ns, and capacitance load = 2fF. 0.246 signifies a 0.246ns delay for the cell when the input transition time = 1ns, and capacitance load = 10fF, etc. It is your job to fill these values in the liberty file. You may leave the (1,2)(2,10) values as provided in the template, and use these values to sweep your stdcells and fill in the LUTs accordingly.

Note that the .lib has other parameters as well which you must fill (area, etc). Refer to both the tutorial and lab document for specifications, and to **spicelib_180_tt_120_25C.lib** as a good template.

Why do we need .cir and .inc files?

The initial .cir files you create will help you verify the correct functionality per x1 gate, if you have properly sized the transistors for x4 drive, and how to measure various properties such as tpdf, and tpdr.

Once you have verified these factors, you may use .inc files to define your cells, their respective terminals, logic behaviour, and any parameters to pass to the gate (such as width). You may then integrate all your cell definitions into a main .cir file, and sweep for different input transition and Cload values required of the .lib file (versus running hspice n*stdcells*2Cload*2input transition times).

It is highly suggested that the **comb_stdcell_liberty2.cir** file is referred to, as it provides an example of instantiating multiple cells in a main file, and sweeping for necessary variables.