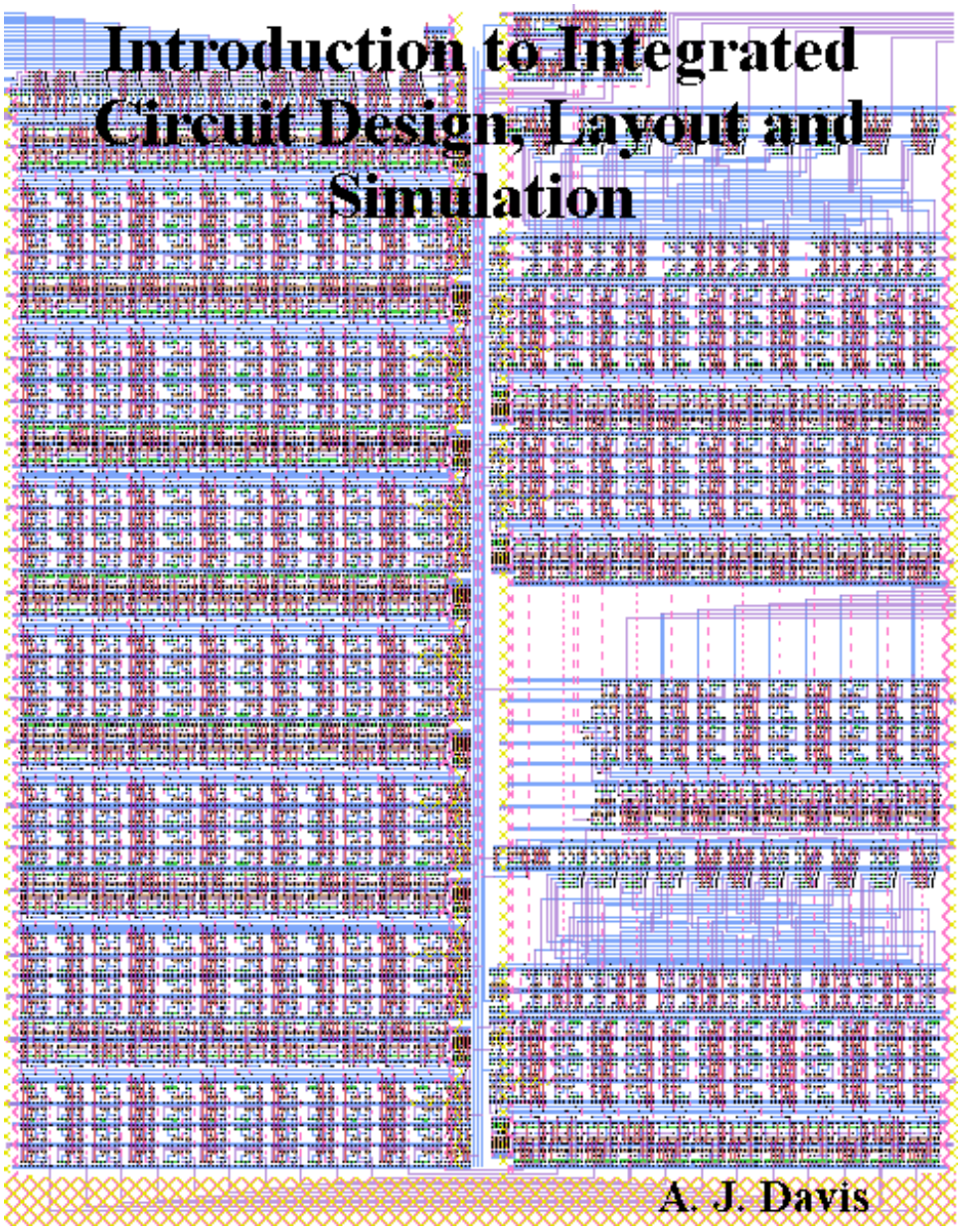
Introduction to Integrated Circuit Design, Layout and Simulation



A. J. Davis

Contents

List of Figures

Preface

The development of an integrated circuit designer is a very long journey. It is important to realize that a single course in this field does not make a person an expert; it is one step on a very long road. The goal here is to prepare students well enough so that on average, each individual who successfully completes the course has achieved a minimum level of competence.

Students need to become effective in the employment of our CAD tool suite. The CAD tools in this course consist of a circuit simulator, *HSPICE*, a layout editor, *magic* and a switch-level digital simulator, *irsim*. Verification at the primitive cell level MUST be done with a circuit simulator, HSPICE in our case.

By completing the exercises from chapters 1-6, the student is literally *walked-through* all of the steps necessary to design a full custom integrated circuit. Students develop their own cells from hand-crafted transistors and they will use these cells to construct larger, hierarchical circuits.

The semester project is sub-divided into two distinct phases: (1) *individual component* and (2) a *final semester project*. The *individual component* requires a similar effort to what was contained in chapters 1-6. Again, students must take things from the hand-crafted transistor cells up through the final pad frame. This assignment is discussed in chapter 7. The *final semester project* can be done individually or in groups. At the University of Rhode Island, I prefer to assign common class projects using groups of only two students. If each individual has successfully completed all of the exercises then two individuals should be able to work efficiently. Occasionally, I will allow a more experienced graduate student to work alone or or even on an M.S. thesis project.

Students are expected to actually employ techniques for manufacturability and economics based upon yield in the completion of their projects. We have tried to incorporate guidance from the ABET visit to the extent that practical restrictions will allow. In order to minimize confusion I have included an appendix which contains the background necessary to understand the role of economics in integrated circuit design.

In future releases of this book I hope to incorporate some of the successful projects graduate students have completed under my supervision. In at least one of these chapters I would like to include an exercise on the verification of the actual integrated circuit. We have fabricated additional ICs with the permission of the MOSIS educational program and we have also constructed some printed circuit boards to facilitate such exercises. It is also possible that I will incorporate additional CAD tools at a later time. At the present time I

feel that exercises on the measurements of first Silicon are the highest priority.

The goal of this work as far as the individual student is concerned is to provide the very best quality experience in a first course in integrated circuit design.

A. J. Davis, July 2005

Acknowledgments

I am very grateful to Professors R. Vaccaro and F. Boudreaux-Bartels of the University of Rhode Island for giving me the opportunity to teach ELE 447. I would like to thank professors J. Daly and G. Fischer for review of this work; thanks to W. Yung for useful suggestions and working through the labs as my TA. I would also like to thank P. Kasturi for literally providing the serial multiplier and the testing.

I would like to thank Gail and the staff at Campus Copy & Design for making the printing seamless.

Special thanks to all of the students I have had in ELE 447 and their suggestions.

I would also like to thank C. Charpea De-Pie for useful comments and observations.

Chapter 1

HSpice Part A

Lab Objectives:

§1. To familiarize students with a basic understanding of the most important types of circuit analysis available using HSPICE and SPICE simulators.

§2. To provide instruction for the representation and testing of actual circuits in HSPICE/SPICE.

§3. To provide a tutorial on HSPICE/SPICE commands, such as, include statements, sub-circuits, parameters, etc.

§4. To provide an overview of the level 3 and BSIM models and how they influence digital integrated circuit design. Also provide an overview of customized models.

Pre Lab:

1. Read Chapter 1 of the textbook pp. 1-25.

2. Read this lab. Write out the net lists prior to the lab.

3. Review "Recent SPICE Parameters" on the ELE 447 Course web page. Download the "Level 3 Model HSPICE File" and the "BSIM3 Model HSPICE File".

1.1 Lab SPICE Part A Instructions.

You will be required to analyze 3 inverters, each with $L_n = L_p = 2.0\lambda$ and $W_n = 4.0\lambda$. The W_p values will be 4λ, 8λ and 12λ, respectively. Each of the 3 inverters will be simulated using the BSIM3 parameter set provided on the course web page for Home Work 1A (and also on the lab web page).

1.1.1 Pre-Lab

1) Compute β_n, β_p using the BSIM3 parameter sets provided on the ELE448 web page (the same set used for Home Work 1A); for β_n, use an n-channel (W/L) of $4\lambda/2\lambda$. For β_p, use the following $\frac{W}{L}$s: $4\lambda/2\lambda$, $8\lambda/2\lambda$ & $12\lambda/2\lambda$. This means that you will be required to compute 1 value of β_n and 3 values of β_p for each parameter set. Set the capacitive load to $100fF$.

2) Compute $\left(\frac{\beta_n}{\beta_p}\right)$ for each inverter.

3) Compute $(\frac{\beta_n}{\beta_p})$, compute the inverter threshold voltage using equation (A.5).

4) Copy the BSIM3 parameter sets over to your home directory. Use the .include "modelName.xxx" directive to access the model.

5) Attempt to run a DC simulation and a transient simulation for at least one of the inverters. Note (Home Work 1 A)

1.1.2 Lab: DC Analysis

6) Measure the inverter thresholds using HSPICE DC analysis. You will need to measure the DC transfer characteristic, V_{out} vs. V_{in}), for each of the 3 inverters. Find the inverter threshold using the graphical method described in your home work example problem.

1.1.3 Lab: Transient Analysis

7) Measure the rising delay, falling delay, rise time and fall time for each of the 3 inverters using each of the two parameter sets. Then compute the propagation delay:

$$\tau_p = \left(\frac{\tau_{dr} + \tau_{df}}{2}\right) \tag{1.1}$$

8) Re-simulate the inverter for $(W/L)_n$ of $4\lambda/2\lambda$ and $(W/L)_p$ of $8\lambda/2\lambda$ using a capacitive load of $1pF$. Measure the rise time and fall time. Estimate the value of R_p and R_n.

1.1.4 Lab: Operating Point and Transient Analysis

9) Insert another n-channel device between the source of the n-channel transistor in the inverter and the ground. This provides 2 n-channel devices in

series and a single p-channel device. Draw the schematic and write out the net list. This is shown in figure 1.2. What are the substrates connected to in the stacked n-channel transistors ?? Connect the lower inverter gate to V_{DD}. Connect the inverter input to 2.5 Volts. Use $W_p = 8.0\lambda$.

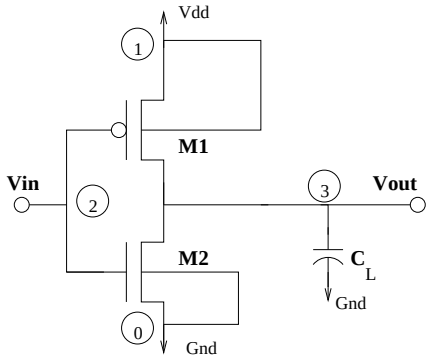


Figure 1.1: Net list labeling for the CMOS inverter with a capacitive load.

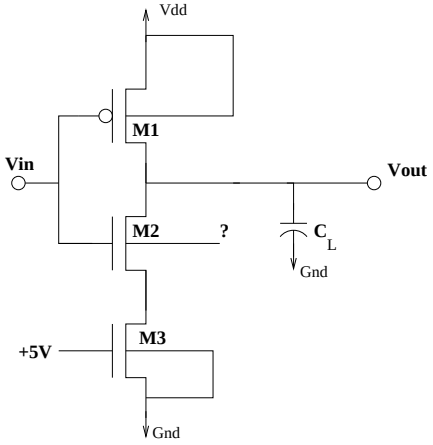


Figure 1.2: Modified inverter for part 6.

i) Compare the value of V_{BS} for the n-channel transistor represented by the net list in figure 1.1 with this inverter. What is different ?

ii) Measure the rising delay, the falling delay, the rise and the fall time of this inverter. Compute the propagation delay. Compare this result with

the inverter from figure A.3 which uses $W_p = 12\mu m$. Is this different ? Why ?

1.2 Problems.

1) The nMOS transistor depicted in figure A.4 has its current flowing from the drain (A) to the source (B). Suppose the direction of the current is reversed; in other words, reversal means that current now flows from the region labeled (B) to the region labeled (A). What happens to the drain and source ? How is the substrate biased (e.g. what voltage) ?

2) Assume the device in figure A.4 is now a pMOS transistor. Can you identify the source and drain terminals ? What is the substrate biased ?

3) If we assume that $V_{tn} = -V_{tp}$, what ratio $\left(\frac{\beta_n}{\beta_p}\right)$ is required for $V_{INVth} = \left(\frac{V_{DD}}{2}\right)$?

4) Using your resulting β_n and β_p from the previous problem, find W_n and W_p assuming that $\mu_n = 2.5\mu_p$.

1.3 Lab Report Instructions

The lab reporting is designed to be as painless as possible. You must tar-zip your working directory and submit it according to instructions provided in the lab; this compressed directory tree, once expanded should provide reproducible results for anything performed during the lab. An electronic copy of your lab report, in pdf-format, should be included in your compressed directory.

The written report should be produced on a word processor and should also be made available as a hard copy. Please be sure to include your name.

You must summarize your results from the lab as illustrated in the tables below:

1) The first tables, Tables 1.1 & 1.2 can and should be completed prior to your scheduled lab date. The information required in Table 1.1 can be obtained from the parameters TOX, U0 and VT0 included in the CMOSN and

CMOSP models used in your lab (for both Level 3 and BSIM3). The units are provided for your convenience and these are standard to these SPICE models. Please note that the oxide thickness, t_{ox} or TOX, will be identical for the respective n-channel and p-channel transistor.

Model Parameter	BSIM3 Model	Units
t_{ox}		m
μ_n		$\frac{cm^2}{Volt-Sec}$
μ_p		$\frac{cm^2}{Volt-Sec}$
V_{T0_n}		$Volts$
V_{T0_p}		$Volts$

Table 1.1: Model parameters from the BSIM3 file.

The values in the second table, Table 1.2, will require the insertion of the parameters obtained from Table 1.1 into the equations listed in this chapter. You will need to track the units and record them properly.

Model Parameter	BSIM3 Model	Units
C_{ox}		
κ_{Pn}		
κ_{Pp}		
β_n		
$\beta_p(W_p = 4\mu m)$		
$\beta_p(W_p = 8\mu m)$		
$\beta_p(W_p = 12\mu m)$		

Table 1.2: Computed model parameters from the BSIM3 file.

2) The inverter voltage thresholds, e.g. the trip-points, are to be recorded in Tables 1.3. Note that this table requires you to compute the inverter voltage thresholds and also record your measurements from the lab. The measured values are obtained using the graphical methods described earlier in this chapter.

3) Timing measurements using the transient analysis are to be recorded in Table 1.4. Note that the propagation delay, τ_p, is obtained from the average of the rising and falling delay times (e.g. τ_{dr} & τ_{df}). Make these results accurate to 3 digits.

4) Using the values of R_p and R_n, compute the rise time and fall time for each of the inverters. You will need to now use the $100fF$ load for C_L; R_p can be

Model	BSIM3		Units
Inverter Ratio	Computed	Measured	
$\left(\frac{\beta_n}{\beta_p}\right)_1$			$Volts$
$\left(\frac{\beta_n}{\beta_p}\right)_2$			$Volts$
$\left(\frac{\beta_n}{\beta_p}\right)_3$			$Volts$

Table 1.3: Inverter threshold voltages (e.g. trip points) vs. $\left(\frac{\beta_n}{\beta_p}\right)$.

scaled in proportion with W_p in order to account for the other values used for W_p. Compare these values with the simulated values for rise and fall time.

Inverter Ratio	τ_{dr}	τ_{df}	τ_p	T_{rise}	T_{fall}	
$\left(\frac{\beta_n}{\beta_p}\right)_1$						$nSec$
$\left(\frac{\beta_n}{\beta_p}\right)_2$						$nSec$
$\left(\frac{\beta_n}{\beta_p}\right)_3$						$nSec$

Table 1.4: Timing measurements vs. $\left(\frac{\beta_n}{\beta_p}\right)$ for BSIM3 model parameters.

5) The last circuit from part 6, figure 1.2 contains 3 transistors, a pair of series connected n-channel transistors and one p-channel transistor. Record the transistor threshold voltage, (V_T), from HSPICE (operating point analysis) for each transistor. Compaire both n-channel transistors with the V_{T0} value obtained from the model parameter list (which you have listed in Table 1.1) in Table 1.5.

Model	BSIM3			Units
Transistor	V_{T0}	V_{BS}	V_T	
M1				$Volts$
M2				$Volts$
M3				$Volts$

Table 1.5: Transistor thresholds obtained from operating point analysis.

6) Record the delay and rise time/fall time measurements (using transient analysis) in Table 1.6 below:

Model Parameter Set	τ_{dr}	τ_{df}	τ_p	T_{rise}	T_{fall}	
Level 3						$nSec$
BSIM3						$nSec$

Table 1.6: Timing measurements of final circuit.

7) Review the problems following the lab.

References

[1] D. O. Pederson, "SPICE: Simulation program with integrated circuit emphasis," in *Proceedings of the* 16^{th} *Mid West Symposium on Circuits & Systems*, (Waterloo, Canada), April 1973.

[2] L. W. Nagel, "Spice 2: A computer program to simulate semiconductor circuits," ERL Memo ERL-M520, University of California, Berkeley, May 1975.

[3] R. Kielkowski, *Inside SPICE*. McGraw-Hill, 1998.

[4] MicroSim, Inc. (OrCad/Cadence), San Jose, CA., *PSPICE User's Guide*, 1995.

[5] Meta-Software, Inc. (AVANT!), Campbell, CA., *Star-HSPICE User's Manual 1998.2*, 1998.

[6] H. Shichman and D. A. Hodges, "Modeling and simulation of Insulated-Gate Field-Effect Transistor switching circuits," *IEEE Journal of Solid-State Circuits*, vol. SC-3, pp. 285–289, September 1968.

[7] D. Frohman-Bentchkowski and A. S. Grove, "On the effect of mobility variation on MOS device characteristics," *Proceedings of the IEEE*, vol. 56, 1968.

[8] V. G. K. Reddi and C. T. Sah, "Source to drain resistance beyond pinch-off in metal-oxide-semiconductor transistors (MOST)," *IEEE Transactions on Electron Devices*, vol. 3, pp. 139–141, 1965.

[9] S. Liu and L. W. Nagel, "Small-signal MOSFET models for analog circuit design," *IEEE Journal of Solid-State Circuits*, vol. SC-17, pp. 983–998, December 1982.

[10] B. J. Sheu, D. L. Scharfetter, P.-K. Ko, and M.-C. Jeng, "BSIM: Berkeley short-channel IGFET model for MOS transistors," *IEEE Journal of Solid-State Circuits*, vol. SC-22, pp. 558–566, August 1987.

[11] W. Liu, *MOSFET Models for SPICE Simulation including BSIM3v3 & BSIM4*. Wiley, 2001.

[12] K. Martin, *Digital Integrated Circuit Design*. Oxford, 2000.

Chapter 2

HSpice Part B

Lab Objectives:

§1. To gain experience in the design and simulation of a ring oscillator.

§2. To provide basic techniques for the simulation of loop structures.

§3. To continue to learn how to use the analysis tools available in HSPICE/SPICE in an efficient manner. Also to learn HSPICE/SPICE "short cuts" using .include, .subckt, .measure and .param directives.

Pre Lab:

1. Complete the Lab assignment for HSpice Part A.

2. Read this lab. Perform necessary computations and prepare the net lists.

3. Review "Recent SPICE Parameters" on the ELE 447 web page. Download the "Level 3 Model HSPICE File" and the "BSIM3 Model HSPICE File". Notice there are 2 sets of Level 3 and BSIM3 parameter files for this lab.

2.1 Background.

Benchmarking is of paramount importance in digital design. Sub-circuits with timing specifications which impact the overall system must be carefully analyzed. The speed of a simple circuit common to all wafer fabrication runs can be used in estimating timing limits for a particular design. Inverters are often used for this purpose since the performance of most static gates can be estimated from the performance of this circuit.

The approach used in the previous lab was to measure the time delay of a single inverter with a known load capacitance. This is difficult to compare with a physical circuit since gate delays are not easy to measure directly. An alternative approach is to measure the total delay of a number of cascaded inverters (e.g. a chain). Still, a more practical approach, which is the topic of this lab, is to measure the frequency of an oscillator realized from inverters in

a closed loop (e.g. connecting the last inverter output in the chain back to the first inverter input).

Oscillators are circuits which generate waveforms at a fixed frequency. A ring oscillator can be constructed by connecting an odd number of inverters in a loop. If properly designed, a loop of inverters can be made to change state at a constant frequency. The propagation delay for a single inverter can easily be inferred from the frequency of a ring oscillator since they are related. This means that the frequency of the same ring oscillator (e.g. with an identical number of equally sized inverters) will serve as a comparative measure of the possible speed over wafers from the same process.

The frequency of a given ring oscillator will vary with each wafer fabricated. A unique set of level 3 and BSIM3 parameters are extracted from each wafer. MOSIS will compare the simulated results using HSPICE with the measured results for the 31 stage ring oscillator. On the ELE 447 web page, under *Recent SPICE parameters* you will find the frequency of oscillation for a 31 stage ring oscillator in the 1.6 μm, double poly process. This is measured for each wafer fabricated. If enough wafers are sampled then the *fast*, *slow* and *typical* ring oscillator frequencies can be estimated.

A CMOS ring oscillator is an example of a circuit which can cause problems when simulating with SPICE. The major obstacle is successfully computing the operating point while the inverters are connected in a ring. This tends to lead to convergence/non-convergence errors. There are simple solutions to eliminate these problems. One solution is to vary the power supply voltage during the start of the simulation. Another technique is to assign voltage levels to each inverter output since SPICE allows one to set either initial conditions or node voltages at time $t = 0$. Still another solution, the one we will use for this lab, is to break the loop at time $t = 0$. This is done long enough to allow SPICE to compute the operating point prior to the simulation. Once a sufficient amount of time has passed the loop is then closed and the ring oscillator can be simulated.

Obtaining good agreement between the simulation and measurement of the ring oscillator is not a simple task. Additional buffers and monolithic components must be added to provide a reliable means of comparing the measurements to the predictions from the circuit simulation. Reliable measurements of the oscillator frequency can be performed albeit difficult.

We will take a closer look at the ring oscillator in the next section.

2.2 The Ring Oscillator

When several inverters are cascaded (e.g. the output of the preceding inverter is connected to the input of the succeeding inverter) and the output of the final inverter is connected to the input of the first inverter forming a loop there are two possibilities:

(1) Bi-State Stability: The value of V_1, once set, is stable for a loop with an even number of inverters. Thus, the logic state at the output of each inverter is fixed as long as the supply voltages are maintained (assuming that

there are no external voltages applied to V_1 or V_2). Since V_1 and V_2 can each have values of a logic "1" or a logic "0", a loop constructed from a pair of inverters will be inherently stable once the voltages, V_1 and/or V_2, are set to either the supply rail or ground. The simplest example is the static latch shown in figure 2.1.

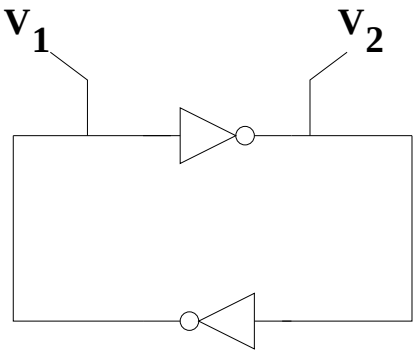


Figure 2.1: A loop formed by 2 inverters will store V_1 and V_2.

(2) Oscillation: If an odd number of inverters are connected in a loop the output of each inverter, V_2 for example, will toggle from a logic "1" to a logic "0" and back continuously; it is an oscillator. This can easily be seen for the loop of 3 inverters shown in figure 2.2.

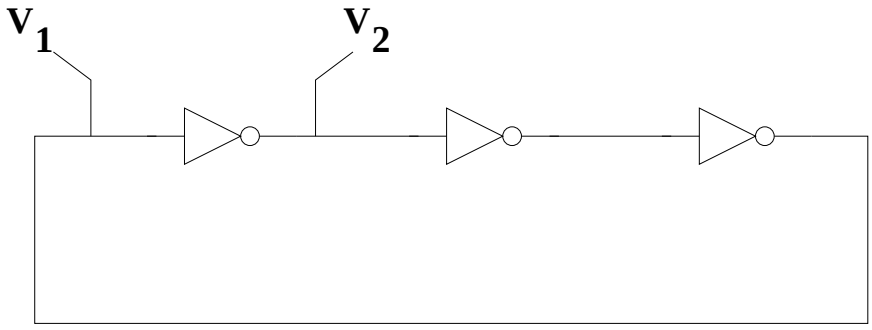


Figure 2.2: A loop formed by 3 inverters.

A loop composed of an odd number of inverters is known as a "ring oscillator".

Analysis of the ring oscillator is quite simple. Let us assume that we have n inverters, where n is odd. The output of the first inverter will be a logic "1",

when its input is a logic "0". If take the output of the inverters which follow all the way back to the first integrator input, we will find that the input is now a logic "1", forcing a logic "0" at its output. If we assume that the delay of each inverter is identical and given by τ_d, then, for n inverters the a total delay given by

$$\tau_{d,n} = n\tau_d \tag{2.1}$$

for each logic state change in a particular inverter in the loop. The oscillation period, T, would require 2 logic state changes for a given inverter and is expressed as

$$T = 2n\tau_d \tag{2.2}$$

yielding the oscillator frequency,

$$f = \frac{1}{T} = \frac{1}{2n\tau_d} \tag{2.3}$$

Equation (2.3) relates the oscillator frequency, f, to the number of inverters, n, and the delay of each specific inverter, τ_d. Increasing the delay of the inverter or increasing the number of inverters in the loop will result in lowering the oscillator frequency.

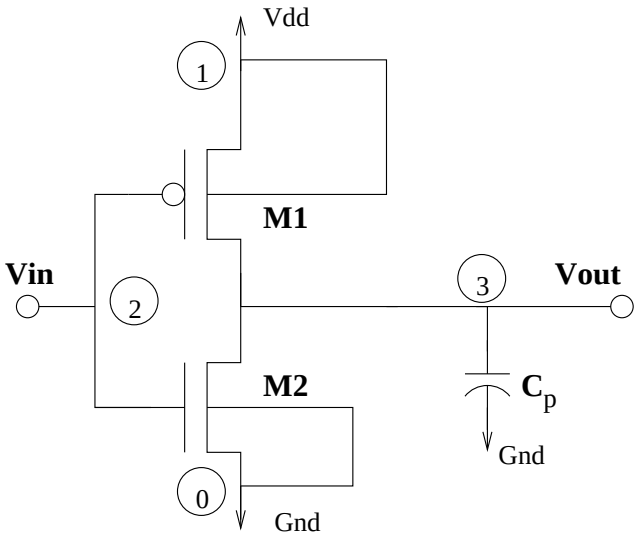


Figure 2.3: CMOS inverter with the netlist nodes labeled.

The characteristics of the inverter which influence τ_d will have a direct effect on the oscillator frequency. The inverter you studied in last weeks lab is shown in figure 2.3. Parasitic capacitances, which depend on the geometry of the source

and drain, will affect performance. Since these are typically not known during the early design phases, the load capacitor, C_p, will represent the parasitics. If the device sizes of the inverter, $(W/L)_p$ and $(W/L)_n$, are increased the inverter will pull up and pull down faster for a fixed capacitive load. Thus, when τ_d is reduced, the oscillator frequency will increase. On the other hand, when C_p is increased, τ_d will increase and the frequency of oscillation will be reduced. This relationship is given by equation (2.4), where the frequency of oscillation is inversely proportional to the capacitive loading on the inverter.

$$f \propto \frac{1}{C_p} \tag{2.4}$$

Care must be taken when assuming that C_p remains fixed when transistor geometries are increased. Increasing in W_p or W_n will increase the gate and drain capacitance for the p-channel and n-channel transistors. This means that C_p actually will increase when W_p or W_n increases. The question of a "speed up" depends upon the ratio of the output current of the inverter and the new value of C_p. If the change in C_p resulting from increasing W_p and/or W_n is small relative to its original value or if there is an additional capacitor (e.g. poly2-poly1 capacitor) which is much larger than the sum of the drain capacitances the assumption is reasonable. In most cases the only capacitive load is that of the transistor parasitics; this means performance will be limited at best when transistor widths are increased.

The ring oscillator is often used as a clock source for digital circuits and for some sampled, analog circuits. There are practical limitations which have not been discussed and will need to be considered when the ring oscillator is employed as a clock circuit. At least 7 integrators are needed to guarantee that a a ring oscillator implemented with CMOS inverters will work reliably. The power supply can have an influence on frequency stability and clock jitter. This is especially true when using CMOS integrators[1]. One improvement is to replace the CMOS inverter with a differential inverter[2]. The advantages of the differential inverter: (1) a ring oscillator can be formed with an even number of differential integrators and (2) most important, differential circuits offer better noise immunity and are less sensitive to power supply noise[1, 2].

2.3 Simulation of the Ring Oscillator in SPICE.

In this lab you will be analyzing a ring oscillator realized by a loop of 11 identical CMOS inverters. In the previous lab you learned about netlists. You should certainly be able to write the netlist for 11 inverters; however, managing a netlist composed of many identical circuits becomes a cumbersome task. Suppose you wish to change a device geometry ? Then you will need to change that value in 11 or more lines within the netlist. Suppose you wish to realize a ring oscillator this time with 31 inverters ? One can easily see that it is simpler to build a single inverter netlist; this enables the realization of a repeated circuit using only a single netlist and some calls to that netlist. The *sub-circuit* statement

provides a reusable, unique circuit. This simplifies the task of using multiple copies of the same circuit and minimizes simulation problems due to errors in the netlist.

2.3.1 The .subckt Statement.

If there is a circuit which is to be used repetitively then a sub-circuit can be formed. This will allow you to reuse a circuit that you specify once. Should you make a mistake with the circuit then you will only need to correct a single part of the netlist.

To implement a sub-circuit, you will place the circuit netlist between the .subckt statement and the .ends statement. The .subckt statement must contain the circuit name and the external node(s)[3]. There is virtually no limit in the complexity of the circuit contained within the .subckt and .ends statements[3]. For example, to place the inverter in figure 3 within a sub-circuit, the following lines are needed:

```
.subckt INVA Vout Vin Vdd Vss
*     D      G      S      SS     Model      W        L *
 M1   Vout   Vin    Vdd    Vdd    CMOSP    W=10u    L=2u
 M2   Vout   Vin    Vss    Vss    CMOSN    W=4u     L=2u
 CL1  Vout   Vss    .1pF
.ends INVA
```

These lines must be placed within your netlist. With the sub-circuit statements included into the netlist, you can now use:

```
X1   node(Vout)   node(Vin)   node(Vdd)   node(Vss)   INVA
```

for each instance of the inverter, INVA. The Xn device name is reserved for sub-circuits[3] and there must be a unique number for each device. The node numbers, e.g. node(Vout), refer to the numbers or characters used to describe the node connection external to the sub-circuit. Node connections within the sub-circuit are local only to the sub-circuit[3]. The modularity can be extended by placing the inverter sub-circuit into a separate file, then using the .include statement to load the sub-circuit into the main netlist.

2.3.2 Netlist Implementation of the Ring Oscillator.

The netlist for the 11 inverters is quite simple. However, you must leave enough time for SPICE to compute the operating point. This can be done by incorporating a switch to open while the operating point is calculated, and then close the loop after a short time. A simple solution is to place an n-channel pass transistor between the 11^{th} and the 1^{st} integrators. At the start of the simulation the switch will be open. After 5-10 nsec, the switch close forming the loop. This approach is illustrated in figure 2.4.

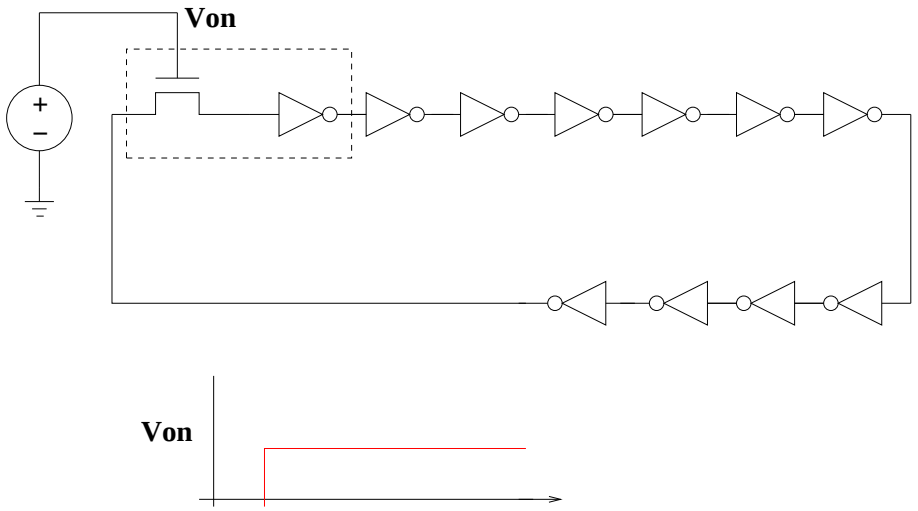


Figure 2.4: The SPICE implementation of the 11 stage ring oscillator.

You will need a signal source at the gate of the n-channel transistor. The wave form for the signal source is also given in figure 2.4. For a short time the gate of the n-channel pass transistor switch is in the logic "0" state, or at ground. After the first few nano-seconds, the gate of the n-channel switch moves from the logic "0" to the logic "1" state, or V_{dd}. The waveform at the gate of the n-channel switch can be synthesized either from the pulse or the piece-wise linear signal source.

2.3.3 .MEASURE Statements

It is possible to automatically measure specific results of a simulation. The advantage of doing this is that HSPICE can be run directly from the command line. This allows one to simulate a circuit while varying parameters much more efficiently than by analyzing the graphs by hand.

Refer to figure 2.5):

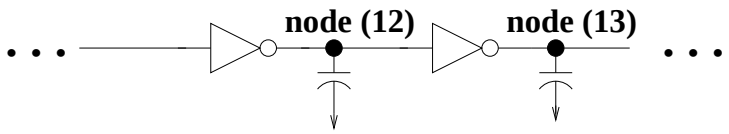


Figure 2.5: Measure statement example.

The following statements will allow one to measure the ring oscillator rising and falling delays, the period and the frequency at several nodes. This can be

accomplished by using the following statements (this example assumes the node names given in figure 2.5):

```
.Measure   TDlyf   Trig   V(10)   VAL=2.5V   Rise=2
+                  Targ   V(11)   VAL=2.5V   Fall=2
.Measure   TDlyr   Trig   V(10)   VAL=2.5V   Fall=2
+                  Targ   V(11)   VAL=2.5V   Rise=2
.Measure    T1     Trig   V(12)   VAL=2.5V   Rise=1
+                  Targ   V(12)   VAL=2.5V   Rise=2
.Measure    T2     Trig   V(12)   VAL=2.5V   Rise=2
+                  Targ   V(12)   VAL=2.5V   Rise=3
.Measure    T3     Trig   V(12)   VAL=2.5V   Rise=3
+                  Targ   V(12)   VAL=2.5V   Rise=4
.Measure    T4     Trig   V(12)   VAL=2.5V   Rise=4
+                  Targ   V(12)   VAL=2.5V   Rise=5
.Measure    T5     Trig   V(12)   VAL=2.5V   Rise=5
+                  Targ   V(12)   VAL=2.5V   Rise=6
.Measure   Freq1   Param='1/T1'
.Measure   Freq2   Param='1/T2'
.Measure   Freq3   Param='1/T3'
.Measure   Freq4   Param='1/T4'
.Measure   Freq5   Param='1/T5'
```

You will need to incorporate these measure statements into your spice deck in order to complete this lab quickly.

2.3.4 Accurate Inveter Measurements

We will also need to measure the common inverter used in the ring oscillator. In the previous lab we measured several inverters using the input signal directly. This will yield an inaccurate result when we are interested in making comparisons between the measured inverter delay and the inverter delay within the ring oscillator. The rise and fall time of the signal driving the inverter will change the delay. This can be adjusted with the pulse statement but there are additional problems; one problem is arriving at a reasonable estimate of the initial rise and fall time. Since the ring oscillator inverter is driven by an identical inverter, it makes more practical sense to drive the inverter we are interested in measuring with a buffer. A simple buffer can be constructed from a cascade of inverters which are identical to the inverter we are testing. This is shown in figure 2.6.

This is easily done using the sub circuit statement in the same way it was described for the ring oscillator frequency measurement.

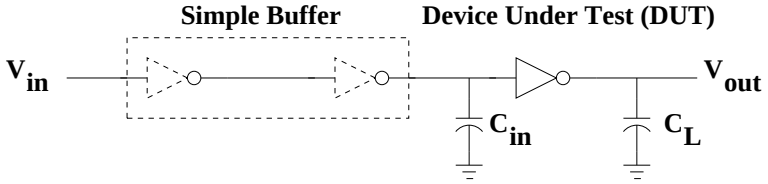


Figure 2.6: Measure statement example.

2.4 Lab SPICE Part B Instructions.

In this lab we will want to directly measure inveter delays in our ring oscillator. We also want to measure the frequency and period of the ring oscillator at several nodes. We want to also make a direct measurement of an inveter delay. Finally, we will compare the inverter delay estimated from the ring oscillator period with that of the two direct inverter delay measurements.

You will be looking at three 11 stage ring oscillators. The first two ring oscillators will use the n-channel pass transistor switch to open and close the loop. The parasitic drain capacitance will be simulated with a load capacitor. In the third ring oscillator we will replace the the pass transistor and one inverter with a NAND gate. The NAND gate will allow us to open and then close this loop. The inverters and the NAND gate will employ source and drain geometries which will allow HSPICE to compute the parasitic capacitances.

2.4.1 Prelab

1) Download all 4 sets of model parameters. You should have the two parameter sets from the previous lab with two additional sets for this lab.

2) For the inverters in the ring oscillator, use: W_n=4λ, W_p=8λ and for L_n = L_p=2λ Scale these values using scale=0.3u with the option statement (in your netlist); this means that W_n=1.2μm, W_p=2.4μm and L_n = L_p=0.6μm.

3) Copy TOX, U0 and VT0 for each set of model parameters. Compute β_n,β_p, and the inverter thresholds for each set of model parameters (note: you have already completed this for the first parameter set in the previous lab).

4) Create your sub circuit using the load capacitance of 75 fF (from figure 2.3).

5) Complete the netlist with the measure statements; get it debugged and working.

2.4.2 Lab: Ring Oscillator 1

6) Prior to performing all of the measurements, use cscope to verify that your measure statements are actually yielding accurate results. Also, plot the

voltage vs. time for the output of the pass transistor switch. Save this plot for your report.

7) For each set of model parameters measure the inverter threshold, the rising delay and the falling delay, using the circuit described in figure 2.6. This can be performed using the measure statement.

8) For each set of model parameters measure the inverter rising delay, falling delay, the oscillator period and frequency. Perform the period and frequency measurements at several nodes. This will also be performed using measure statements.

2.4.3 Lab: Ring Oscillator 2

9) Replace the n-channel pass transistor switch and the first inverter in your ring oscillator (the area inside the box in figure 2.4) with a nand gate as shown in figure 2.7.

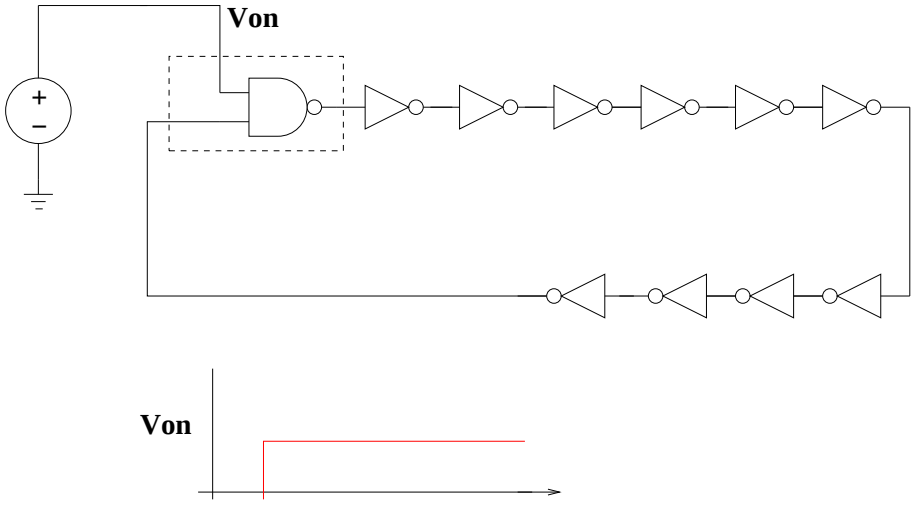


Figure 2.7: The 11 stage ring oscillator using a nand gate reset.

10) Use the following sub circuits for the inverter and the nand gate for the ring oscillator depicted in figure 2.7. Notice that the dimensions are in λ; use the scale directive to properly handle the device sizes.

.subckt INVA Vout Vin Vdd Vss

```
*     D      G     S     SS     Model     W     L     Area D    Per D    Area S    Per S
M1    Vout   Vin   Vdd   Vdd    CMOSP     W=8   L=2   AD=48     PD=20    AS=48     PS=20
M2    Vout   Vin   Vss   Vss    CMOSN     W=4   L=2   AD=24     PD=16    AS=24     PS=16
.ends INVA

.subckt NANDA Vnand Va Vb Vdd Vss
*     D      G     S     SS     Model     W     L     Area D    Per D    Area S    Per S
M1    Vnand  Va    Vdd   Vdd    CMOSP     W=8   L=2   AD=24     PD=6     AS=48     PS=20
M2    Vnand  Vb    Vdd   Vdd    CMOSP     W=8   L=2   AD=24     PD=6     AS=48     PS=20
M3    Vnand  Va    V2    Vss    CMOSN     W=8   L=2   AD=48     PD=20    AS=12     PS=3
M4    V2     Vb    Vss   Vss    CMOSN     W=8   L=2   AD=12     PD=3     AS=48     PS=20
.ends NANDA
```

11) For each set of model parameters measure the inverter threshold, the rising delay and the falling delay, using the circuit described in figure 2.6. This can and should be performed using the measure statement. Compute the propagation delay.

12) For each set of model parameters measure the inverter rising delay, falling delay, the oscillator period and frequency. Perform the period and frequency measurements at several nodes. This should also be performed using measure statements. Compute the propagation delay and compute the estimate of the propagation delay.

13) Compute the ratio of the propagation delay from the ring oscillators to that of the propagation delay measured for each inverter.

2.5 Problems.

1) Which type of analysis did you use to measure the period of the ring oscillator ? Why ?

2) Explain why the propagation delay estimated from the ring oscillator is greater than that directly measured from the inverter. How could this be quantified for all three ring oscillators analyzed ?

3) Find the maximum and minimum output voltages from your cscope plot of the pass transistor output. Can you explain why this happens ?

4) In the final part of the lab, when the value of C_L was changed, compute the expected frequency of the ring Oscillator for each C_L. Is it in good agreement ?? Please explain.

5) Construct a level 1 model. Analyze the DC transfer characteristic of the inverter with $W_n = 1.2\mu m$, $W_p = 3\mu m$ and $L_n = L_p = 0.6\mu m$. Simulate the same inverter using the first set of level 3 and BSIM3 parameters provided. Then overlay the 3 DC transfer characteristics. Use the following level 1 model parameters in your include statement:

```
.MODEL    CMOSN    NMOS    LEVEL=1    TOX=3.0500E-08    VTO=0.7000    UO=500
.MODEL    CMOSP    PMOS    LEVEL=1    TOX=3.0500E-08    VTO=-0.7000   UO=250
```

2.6 Report Format

1) The first tables, Tables 2.1 & 2.2 can and should be completed prior to your scheduled lab date; this is nearly identical to what was done in the previous lab. In fact, much of what had been completed for the first two model sets used in that lab can be copied. The information required in Table 1.1 can be obtained from the parameters TOX, U0 and VT0 included in the CMOSN and CMOSP models used in your lab (for both Level 3 and BSIM3). The units are provided for your convenience and these are standard to these SPICE models. Please note that the oxide thickness, t_{ox} or TOX, will be identical for the respective n-channel and p-channel transistor.

Model Parameter	BSIM3	BSIM3	Units
t_{ox}			m
μ_n			$\frac{cm^2}{Volt-Sec}$
μ_p			$\frac{cm^2}{Volt-Sec}$
$VT0_n$			$Volts$
$VT0_p$			$Volts$

Table 2.1: Model parameters from BSIM3 models.

The values in the second table, Table 1.2, will require the insertion of the parameters obtained from Table 1.1 into the equations listed in this chapter. You will need to track the units and record them properly.

2) Summarize the measurements of each ring oscillator (and each inverter) for each model. This information is illustrated in Tables 2.3 through 2.4.

Table 2.3 summarizes the results for Ring Oscillator-1. Table 2.4 summarizes the results for Ring Oscillator-2 using the inverter with drain and source geometries added.

Model Parameter	BSIM3	BSIM3	Units
C_{ox}			
κ_{Pn}			
κ_{Pp}			
β_n			
β_p			

Table 2.2: Computed model parameters from Level 3 & BSIM3 models.

inverter load	Ring Oscillator Circuit						Inverter-1			Ratios	
	F	T	$\hat{\tau}_{dp}$	τ_{df}	τ_{dr}	τ_p	τ_{df}	τ_{dr}	τ_p	$\frac{ta\hat{u}_p}{\tau_{p-inv}}$	$\frac{\tau_{p-r}}{\tau_{p-inv}}$
Model	MHz	nSec	nSec	nSec	nSec	nSec	nSec	nSec	nSec		
t56-b.BSIM3											
t55-q.BSIM3											

Table 2.3: Analysis of Ring Oscillator-1 for inv-1.

C_L Geom.	Ring Oscillator Circuit						Inverter-1			Ratios	
	F	T	$\hat{\tau}_{dp}$	τ_{df}	τ_{dr}	τ_p	τ_{df}	τ_{dr}	τ_p	$\frac{ta\hat{u}_p}{\tau_{p-inv}}$	$\frac{\tau_{p-r}}{\tau_{p-inv}}$
Model	MHz	nSec	nSec	nSec	nSec	nSec	nSec	nSec	nSec		
t56-b.BSIM3											
t55-q.BSIM3											

Table 2.4: Analysis of Ring Oscillator-2 for inv-2.

3) Complete the problems at the end of the lab write up.

References

[1] D. J. Johns and K. Martin, *Analog Integrated Circuit Design*. John Wiley & Sons, 1997.

[2] A. W. Buchwald and K. Martin, "High-speed voltage-controlled oscillator with quadrature outputs," *Electronics Letters*, vol. 27, pp. 309–310, February 1991.

[3] Meta-Software, Inc. (AVANT!), Campbell, CA., *Star-HSPICE User's Manual 1998.2*, 1998.

Chapter 3

Magic Part A

Lab Objectives:

§1. To learn basic CMOS NWell processing steps and how it relates to the layers in the layout editor.

§2. To provide some instruction on the layout of individual, parallel and series n-channel and p-channel transistors.

§3. Begin training using the magic layout editor, its most basic tools.

§4. To provide instruction and exercise in the synthesis of simple logic cells from hand-crafted transistors.

§5. To gain experience in layout verification via HSPICE/SPICE simulation of extracted layouts.

Pre Lab:

1. Read Chapter 2 of the textbook pp. 35-59. Pay special attention to the section 2.3 *CMOS Layout and Design Rules*, pp. 48-59.

2. Study the layout of inverters, Nand Gates and geometries. Read over homework problem (9) from homework 1.

3. Read this lab carefully.

4. Try running magic and executing basic magic commands. Read magic tutorials 1, 2 and 3 (in tutorial 3, read up to section 2.5). Try some of the exercises in the tutorials provided with magic.

3.1 Background

Integrated circuit design requires a sophisticated set of computer aided design (CAD) tools. The previous labs provided insight into circuit simulation using HSPICE. Circuit simulation, albeit important, is just one software package

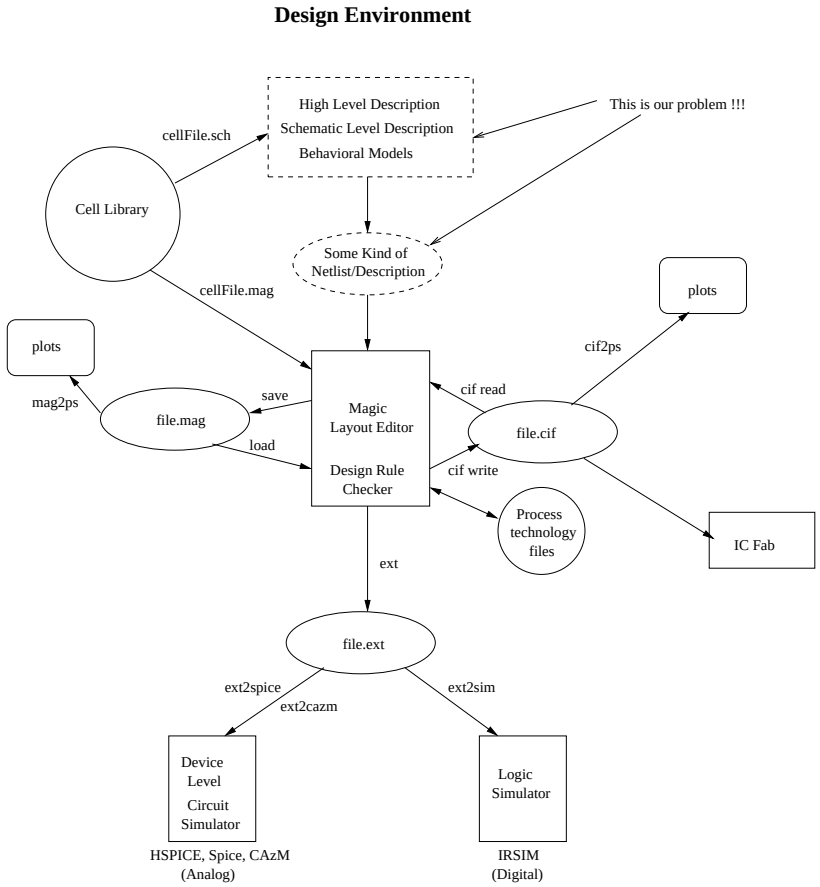


Figure 3.1: The Design Environment for ELE 447.

within a suite of tools. Figure 3.1 provides a complete description of the design environment we will be using in this course.

There must be a high-level description (HLD) of the IC. As a design nears completion, every aspect will be contained within the HLD. The HLD can be composed of more than one tool. If we were designing a digital filter, for example, MATLAB m-files would typically be employed in the earliest design phase. A VHDL editor or some other type of HLD language becomes necessary as the design increases in complexity. This allows for digital simulation of the HLD. Since the projects in ELE 447 are simple in scope, our HLD will consist of a carefully labeled schematic. The schematic is translated to a drawing of the abstract layers needed be the foundry to fabricate the circuits. These drawings are known as layouts and the automated tool for drawing layouts is a layout editor. Our layout editor, which is the topic of this and the next 3 labs, is called

magic[1].

Magic provides a mask oriented description of the HLD circuit. The layout editor is hierarchical; this extends the complexity of its design capabilities and it allows for the inclusion of library cells. The magic layout editor uses the Mead and Conway scalable design rules[2] based on dimensionless units (λ). The layers in magic differ from the actual mask layers. While some are identical magic includes pseudo layers. In addition, some layers are added automatically when the mask layers are prepared for fabrication. The interactive design rule checker immediately alerts the user of design rule violations. Many different sets of design rules can be included. This allows magic to accommodate many different IC fabrication technologies. Design rules and mask layer translation are facilitated by tech files as shown in figure 3.1. Layouts can be extracted to circuit simulators like HSPICE/SPICE and logic simulators such as irsim. Parasitics within the layout can be included via the extraction software. The mask sets can be formatted for the CalTech Intermediate Format (CIF) or GDSII. Magic layouts can also be plotted as shown in figure 3.1.

Magic cannot be completely digested in a single day. Our approach is to provide you with several projects which gradually use most of the features needed to fabricate a design. We start with simple gates at the cell level; not all of the features of magic are needed in this first lab.

In the sections that follow, a brief review of the development of magic is in section 2. The most important layers you will need to construct logic gates are introduced in section 3, Magic Layers. Section 4 provides guidance for drawing transistors in magic. The lab instructions and questions are in sections 5 & 6.

3.2 A Brief History of Magic

Magic was developed during the early 1980's at the University of California, Berkeley[1]. The software development was funded by the U.S. Government which required that the software be made available to the public free of charge with the source code. Magic Version 4 was distributed on tape along with other Berkeley Software tools in 1986. The first versions of magic were not compatible with X-Windows, which was being developed under the name "Project Athena" during the mid 1980s. Later versions, 6 and higher, incorporated X windows. The Government funding for the software development and maintenance ended in the early 1990s.

The popularity of magic has soared during the late 1990s. With the widening popularity of the Web and public domain Unix software, the number of magic users has increased significantly. Public domain developers have extended the life of magic well past the last official build, version 6.5. The latest release of magic is 7.1 and it is expected to continue to evolve.

3.3 Magic Layers

In magic, grid-spacing is measured in λ, dimensionless, scalable units. The relationship between λ and the minimum feature size is:

Minimum Drawn Feature Size = 2λ

Thus, for the 1.2 μm process, $\lambda = 0.6$ μm. The layers in magic (with the exception of pads) conform to this relation. This allows for scalable design rules[2, 3, 4]. The most important layers in magic for drawing cells in the 1.2 μm double-poly CMOS process are shown in figure 3.2.

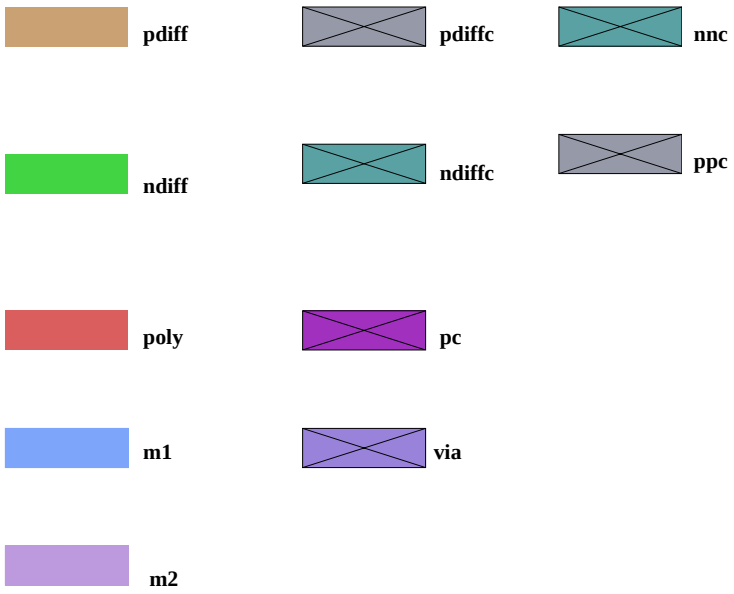


Figure 3.2: Magic layers.

The layers used to realize transistors and to route signals are listed in table 3.1:

Magic Abbr.	Layer Name
pdiff	p-diffusion
ndiff	n-diffusion
poly	polysilicon
m1	metal 1
m2	metal 2

Table 3.1: Routing/Active Layers in Magic.

The n-diffusion and p-diffusion layers are collectively referred to as “active” layers. This terminology reflects that every time polysilicon crosses an active

layer a transistor is formed. The N-Well layer which is needed for the p-channel transistor is not shown but it is available in magic. This layer is not frequently used in digital designs because it is automatically generated when the magic layout is converted into CIF or GDSII for wafer processing. The metal-1 and metal-2 layers provide for routing signals and buses.

Electrical connections must be made between the layers used to paint transistors and routing lines. When diffusion or polysilicon are connected to a metal layer, the electrical connection is referred to as a contact. Two metal layers are connected by a special contact known as a via.

The layers in magic which provide electrical connections between different active/routing layers for the 1.2 μm CMOS process are listed in table 3.2:

Magic Abbr.	Layer Name
pdiffc, ndiffc	diffusion contact (p or n) between diffusion and m1
nnc, ppc	substrate contacts connect the substrate to m1
pc	poly contact connects polysilicon to m1
via	via connects m1 to m2

Table 3.2: Connecting Layers in Magic.

LAYER	CIF LAYER NAME	CALMA NUMBER
Well	CWG	14
N-well	CWN	1
P-well	CWP	2
Active	CAA	3
Select	CSG	15
P-select	CSP	8
N-select	CSN	7
Poly	CPG	4
Poly Contact	CCP	45
Poly 2 (Electrode)	CEL	5
Electrode Contact	CCE	55
Active Contact	CCA	35
Metal 1	CMF	10
Via	CVA	11
Metal 2	CMS	12
Overglass	COG	13

Table 3.3: Mask layers for 1.2 μm double-poly CMOS process.

There are additional layers and contacts provided by magic. Sub-micrometer semiconductor processes have 4-6 metal layers while analog processes, like the 1.2 μm CMOS we will be using, have a second polysilicon layer. The additional metal layers are used for routing. In an analog IC process resistors can be fabricated by a serpentine polysilicon line. Capacitors are typically formed using the two polysilicon layers. Polysilicon is often referred to as "poly".

The abstract layers within magic allow greater flexibility than designing directly with the mask layers. However, prior to submitting a design for fabrication the abstract layers must be converted to the mask layers. Standard file formats adopted by foundries are used to represent the mask level design. Two common formats are CIF and GDSII. Table 3.3 lists the CIF and GDS II layer assignments.

The CIF layers can be viewed in magic. There are two ways to do this: (1) create a cif file for submission then open the cif file in magic or (2) use the command :cif see layer to view a specific layer. It is important to recognize the differences between the magic layout editor and the masks used for fabrication.

3.4 MOS Transistors in Magic

NMOS transistors are formed by crossing the n-diffusion layer with the polysilicon layer. PMOS transistors are formed by crossing the p-diffusion layer with the polysilicon layer. This is illustrated in figure 3.3. Magic must create a new layer in the area where polysilicon crosses diffusion. This is necessary to prevent heavy doping in the channel. Typically, the channel has some doping for threshold adjustments, but it is significantly different than what is required for the source and drain connections.

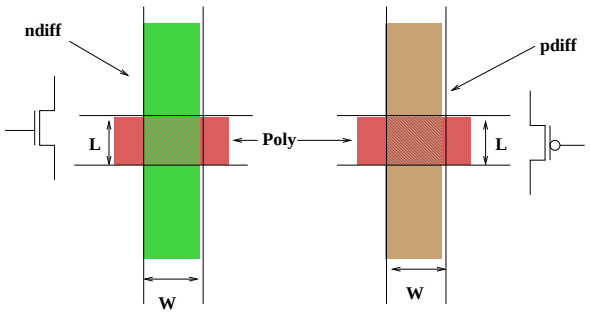


Figure 3.3: Magic layout representation of an n-channel and p-channel transistor.

Many of the simple gates, e.g. NAND, NOR, AND, OR require parallel and series connections of n-channel and p-channel transistors. Figure 3.4 shows several possible options for connecting series transistors: (1) connect the drain and source using metal 1 (m1); this requires two diffusion contacts, (2) connect the drain and source with a single diffusion contact or (3) make a direct connection using only diffusion.

The latter option is desirable in minimizing the size of the drain and source since the respective parasitic capacitance will grow in proportion to their size. This will not be possible if you need to make a metal connection between the transistors. A metal connection will require you to leave enough room for a

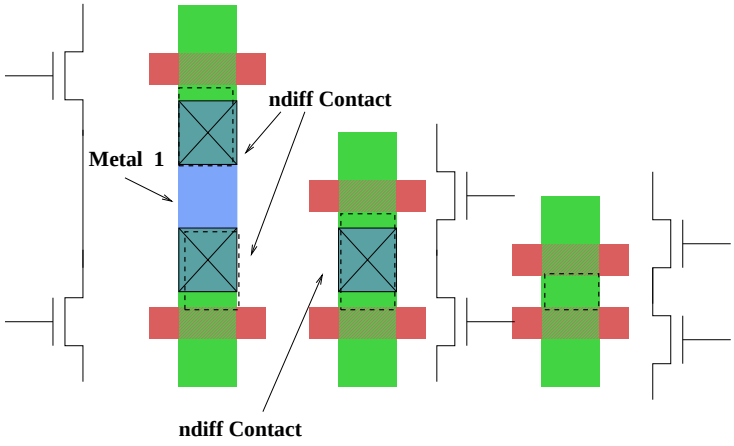


Figure 3.4: Possible layout geometries for series transistors.

diffusion contact. You should always avoid the first approach unless there is a good reason to do so since there is no additional utility and it decreases area-efficiency and speed.

The dashed lines for each alternative in figure 3.4 show the region of the drain and source contributing to their respective capacitance. Thus, the first option is almost never used since the capacitance is much larger. The third option minimizes the effective capacitance by minimizing the geometry and combining the drain of the first transistor with the source of the second transistor. If a metal connection is required, the second option must be selected. We will learn how these connections affect the design of simple gates.

3.5 Lab Instructions

In this lab you will design, extract and simulate a CMOS inverter, a 2 input CMOS NAND gate and a 4 input CMOS NAND gate. It is important to read over these instructions and to prepare for this lab by completing tutorials 1, 2 and 3. Instructions are provided but they are not explicit. There are multiple options for almost any drawing step in magic. You are expected to identify the option which suits you best on your own time prior to the laboratory period.

3.5.1 Magic Commands

The documentation for using magic is enclosed in reference [1]. The sections which describe the use of actual commands within magic are contained in 11 tutorials. You will notice that our web page provides links to all tutorials. For this lab, it is important that you read over tutorials 1, 2, and 3. Links are also

provided to these specific tutorials in the Magic Part A Web Page in order for you to find them and read them thoroughly.

The following section contains the instructions for this lab. Whenever an instruction is started with a ":", this is a command that you can type in on the magic command line. It will be assumed that you understand how to draw boxes, paint inside boxes, copy drawings, move drawings, undo mistakes, re-do undo's, orient layouts and the associated macros. In addition, it is expected that you understand how to switch between magic tool bars. You should know what the wiring tool is and how to use it.

In this lab, you will be provided with nearly all of the drawing instructions necessary to complete your layouts. This is done in order to allow you to focus on the use of the magic commands needed in order to realize simple cells. You should practice this lab at home after the lab is completed. It is especially important to do this if you experience problems in accomplishing this lab assignment. After this lab, it will be assumed that you can ALWAYS draw a layout, extract it and verify the layout using HSPICE.

3.5.2 Layout of the CMOS Inverter

The layout and HSPICE verification of a simple CMOS inverter can be completed by following the steps outlined:

1) start magic; use the command: magic447.

2) Draw a box 8λ high x 12λ wide.

3) :paint pdiff

4) Draw a box exactly 15 λ below the first one, 4λ high x 12λ wide.

5) :paint ndiff.

6) Draw the V_{dd} rail:

   i) Draw a box 4λ high and 12λ wide exactly 4λ above the p-diffusion.

   ii) :paint m1

7) Draw the ground rail:

   i) Draw a box, 4λ high x12λ wide, exactly 4λ below the n-diffusion.

   ii) :paint m1

   The layout in the drawing palette in magic should now be identical to the layout shown in figure 3.5.

8) Draw a box 8λ high x 4λ wide at the leftmost edge of the p-diffusion.

9) :paint pdiffc

10) Draw a box 8λ high x 4λ wide at the rightmost edge of the p-diffusion.

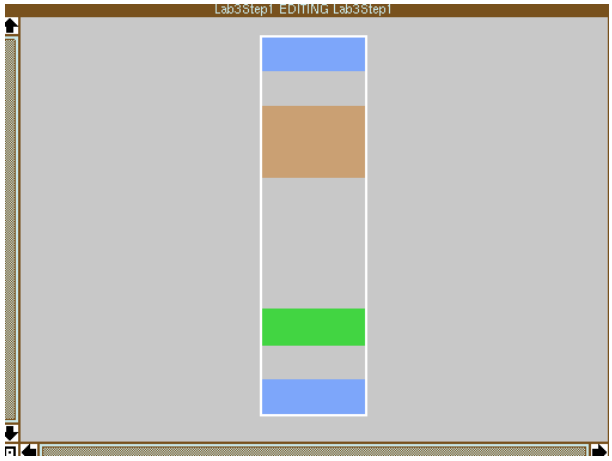


Figure 3.5: First steps in the layout of a CMOS inverter.

11) :paint pdiffc

12) Draw a box 4λ high x4λ wide at the leftmost at edge of the n-diffusion.

13) :paint ndiffc

14) Draw a box 4λ high x 4λ wide at the rightmost at edge of the n-diffusion.

15) :paint ndiffc

16) Draw a box 2λ wide x 33λ high in the center (1 λ separation between diffusion contacts & 1 λ above the n-diffusion).

17) :paint poly (be sure to use correct overhang)

18) Draw the following M1 connections (use the wiring tool in magic):

    i) $Vdd -> $ p-diffusion; the M1 width should be 4λ.

    ii) Gnd $->$ n-diffusion; the M1 width should be 4λ.

    iii) Connect the Drains - use M1 with a width of 3λ.

    The layout in the drawing palette should now be identical to the layout shown in figure 3.6.

19) Make the n-substrate connections for the p-channel MOS transistor

    i) Draw a 4λx4λ box on the leftmost side of the V_{dd} rail.

    ii) :paint nnc

    iii) Draw a 4λx4λ box on the rightmost side of the V_{dd} rail.

    iv) :paint nnc

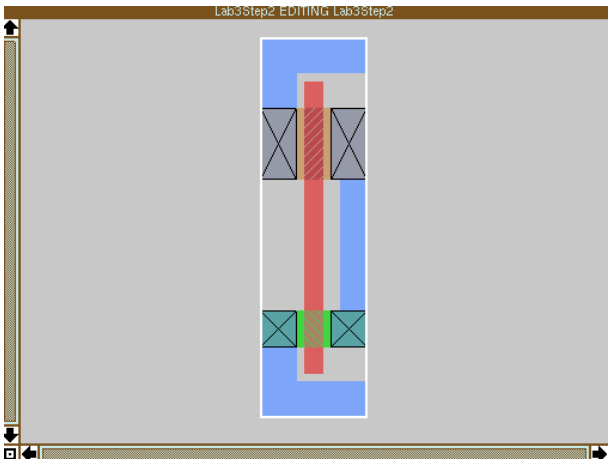


Figure 3.6: Partially completed inverter with m1 connections.

20) Make the p-substrate connections for the n-channel MOS transistor

   i) Draw a 4λx4λ box on the leftmost side of the Gnd rail.
   ii) :paint ppc
   iii) Draw a 4λx4λ box on the rightmost side of the Gnd rail.
   iv) :paint ppc

21) Place labels:

   i) V_{dd} - make a 4λx4λ box on first n-substrate contact.
   ii) :lab Vdd cen
   iii) Gnd - make a 4λx4λ box on first p-substrate contact.
   iv) :lab Gnd cen
   v) In - make a 0x0 box on poly
   vi) :lab In t
   vii) Out - make a 0x0 box on M1 (drain connections)
   viii) :lab Out t

22) :drc check (final drc check)

23) :drc why

24) correct any design rule errors (if there are any) The layout in the drawing palette should now be identical to the layout shown in figure 3.7.

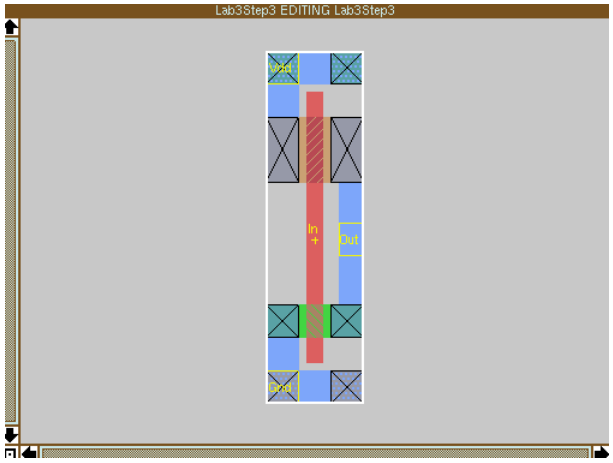


Figure 3.7: Completed inverter with substrate connections.

25) :save inverter

When a drc error occurs the layout editor will identify the error by placing white dots at the offending spot as shown in figure 3.8. Design rule (drc) errors are displayed interactively. DRC checking can be run by typing the macro "y" instead of :drc why. Whenever :drc y is activated, magic will either reply with "no errors found" or it will make a list of errors. The error/rule number can be identified by checking the rules on the web page. In order for the error to be identified using :drc y, the area where the error

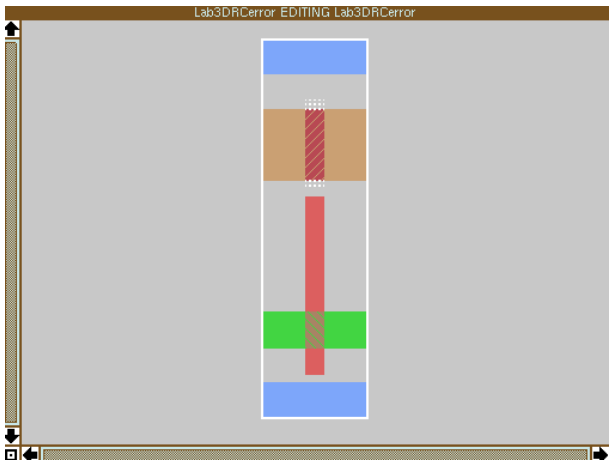


Figure 3.8: Example: interactive design rule checker (drc) violations.

occurs must be inside the box. Otherwise, you will receive the message "no

errors found" when no errors exist inside the box.

26) extract the inverter (from inside magic):

   i) :exttospice format hspice

   ii) :ext

   iii) :exttospice

   iv) exit magic

27) HSPICE simulation:

   i) in your main hspice file, use .include "inverter.sp".

   ii) use the BSIM parameters provided.

By now, you are all probably experts in HSPICE. However, it is still possible to make errors in HSPICE when you attempt to simulate extracted layouts. Check that the model names are consistent. BSIM model names are CMOSN and CMOSP, for the n-channel and p-channel transistors while the extractor uses nfet and pfet. You must edit the net list of the inverter in order to change the model names for each transistor in the netlist.

3.5.3 Layout of the NAND Gate

1) start magic.

2) Draw a box 8 λ high x 20 λ wide.

3) :paint pdiff

4) Draw a box, exactly 15λ below the first one, which is 8λ high x 20λ wide.

5) :paint ndiff.

6) Draw the V_{dd} rail:

   i) Draw a box 4λ high and 20λ wide, exactly 4λ above the p-diffusion.

   ii) :paint m1

7) Draw the ground rail:

   i) Draw a box, 4λx20λ, exactly 4λ below the n-diffusion.

   ii) :paint m1

   The layout in the drawing palette should now be identical to the layout shown in figure 3.9.

8) Draw a box, 8λx4λ at the leftmost edge of the p-diffusion.

9) :paint pdiffc

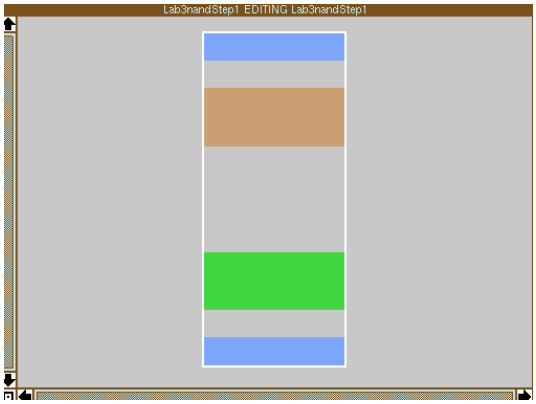


Figure 3.9: Nand gate power rails & diffusion layers drawn.

10) Draw a box 8λx4λ at the rightmost edge of the p-diffusion.

11) :paint pdiffc

12) Draw a box 8λx4λ at the center of the p-diffusion, leaving 4λ between the box and the existing p-diffusion contacts.

13) :paint pdiffc

14) Draw a box, 8λx4λ at the leftmost edge of the n-diffusion.

15) :paint ndiffc

16) Draw a box, 8λx4λ at the rightmost edge of the n-diffusion.

17) :paint ndiffc

    The layout in the drawing palette should now be identical to the layout shown in figure 3.10.

18) M1 connections (use the wiring too):

    i) $V_{dd} -> $ p-diffusion - m1, width 4λ, leftmost and rightmost sides.

    ii) Gnd $->$ n-diffusion - m1, width 4λ, only on the leftmost side.

    iii) Drains $->$ m1, width 4λ, is run between the middle p-diffusion contact and the rightmost n-diffusion contact.

19) Draw the poly lines:

    i) Draw a box, 2λ wide and 37λ on the left side of the center p-diffusion contact (leave 1λ separation between the box and the p-diffusion contacts; it should be 1λ above the ground rail).

    ii) :paint poly

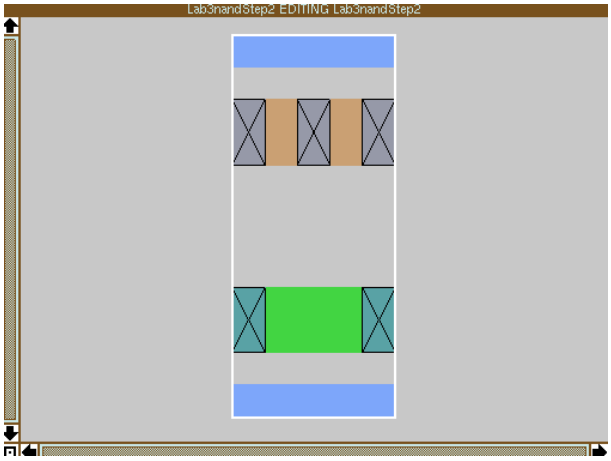


Figure 3.10: Nand gate power rails & diffusion layers drawn.

iii) Draw a box, 2λ wide and 32λ on the right side of the center p-diffusion contact (leave 1λ separation between the box and the p-diffusion contacts; it should be 1λ above the ground rail).

iv) :paint poly

The layout in the drawing palette should now be identical to the layout shown in figure 3.11.

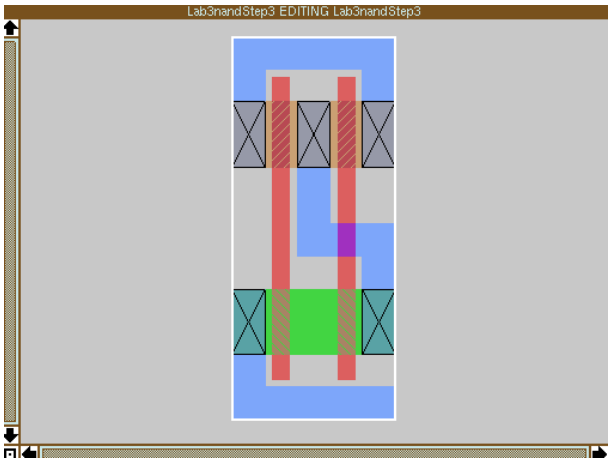


Figure 3.11: Nand gate power rails & diffusion layers drawn.

20) Make the n-substrate connections for the p-channel MOS transistor:

i) Draw a 4λx4λ box on the leftmost side of the V_{dd} rail.

ii) :paint nnc

iii) Draw a 4λx4λ box on the rightmost side of the V_{dd} rail.

iv) :paint nnc

v) Draw a 4λx4λ box on the middle of the V_{dd} rail, leaving 4λ between n-substrate contacts.

vi) :paint nnc

21) Make the p-substrate connections for the n-channel MOS transistor

i) Draw a 4λx4λ box on the leftmost side of the Gnd rail.

ii) :paint ppc

iii) Draw a 4λx4λ box on the rightmost side of the Gnd rail.

iv) :paint ppc

v) Draw a 4λx4λ box on the middle of the V_{dd} rail, leaving 4λ between p-substrate contacts.

vi) :paint ppc

22) Place labels:

i) V_{dd} - make a 4λx4λ box on first n-substrate contact.

ii) :lab Vdd cen

iii) Gnd - make a 4λx4λ box on first p-substrate contact.

iv) :lab Gnd cen

v) A - make a 0x0 box on poly.

vi) :lab A t

vii) B - make a 0x0 box on poly.

viii) :lab B t

ix) nand - make a 4λx4λ box on M1 (drain connections).

x) :lab nand cen

23) perform the final drc check.

The layout in the drawing palette should now be identical to the layout shown in figure 3.12.

24) :save nand

25) extract the nand gate.

i) :exttospice format hspice

ii) :ext

iii) :exttospice

iv) exit magic

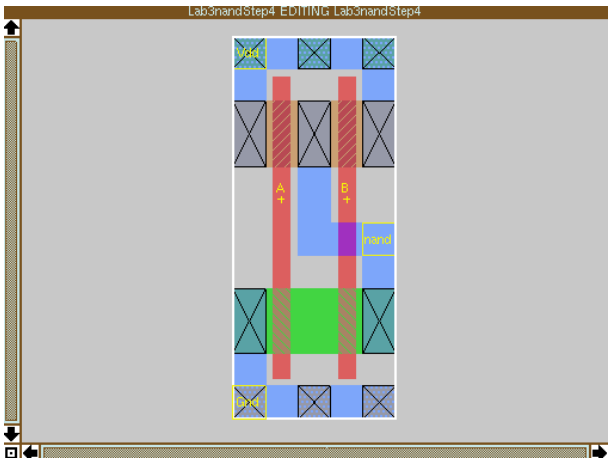


Figure 3.12: Completed Nand gate.

26) HSPICE simulation:

i) in your main hspice file, use .include “nand.sp”

ii) use the BSIM parameters provided

3.5.4 Design a 4-input, CMOS nand gate

You will realize a 4-input nand gate by modifying the layout of your 2-input nand gate. Don’t worry about geometries, leave the pdiff and ndiff sizes identical.

1) Start magic.

2) :load nand

3) use the copy command to expand the nand gate.

   The layout in the drawing palette should now be identical to the layout shown in figure 3.13.

4) Now, complete the alterations in order to realize the 4-input nand gate. The layout in the drawing palette should now be identical to the layout shown in figure 3.14.

5) Perform the drc check.

6) :save nand4

7) Extract the circuit.

8) Simulate it in hspice.

Congratulations !! By completing this lab you have now become a magician (apprentice level) !!

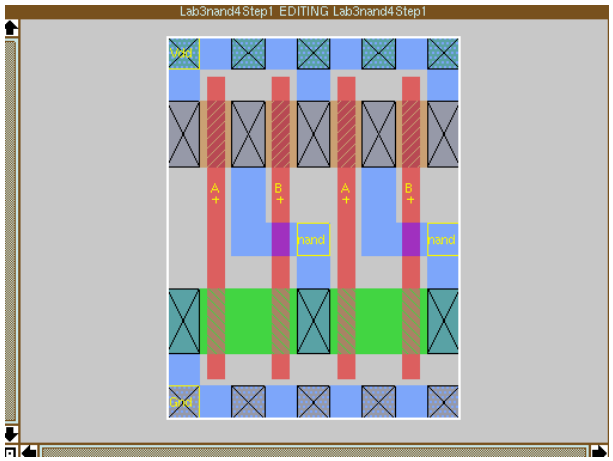


Figure 3.13: Copied structure of 2-input nand gate.

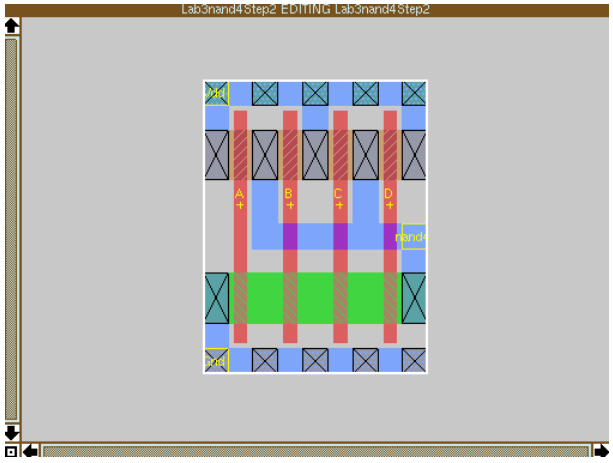


Figure 3.14: Completed 4-input Nand gate.

3.6 Problems

1) Find the gate, source and drain dimensions of the inverter in λ's. What are they in μm's ?

2) Find the gate, source and drain dimensions of the 2 input Nand Gate in λ's. What are they in μm's ?

3) Start magic. Load your inverter. Type :cif see CWN; what CIF layer is this ??

4) Find the gate, source and drain dimensions of the 4 input Nand Gate in λ's. What are they in μm's ?

5) If you examine the gate dimensions of the 2 input nand gate what would you assume for the ratio of $\frac{\mu_n}{\mu_p}$ is ?? Is this true for the inverter cell ?? Why/Why not ??

References

[1] R. M. Mayo, M. H. Arnold, W. S. Scott, D. Stark, and G. T. Hamachi, "1990 DECWRL/livermore magic release," Tech. Rep. WRL Research Report 90/7, Digital: Western Research Lab, Palo Alto, CA, September 1990.

[2] C. Mead and L. Conway, *Introduction to VLSI Systems*. Addison-Wesly, 1980.

[3] N. H. E. Weste and K. Eshraghian, *Principals of CMOS Design: A Systems Perspective*. Addison-Wesly, 1993.

[4] K. Martin, *Digital Integrated Circuit Design*. Oxford, 2000.

Chapter 4

Magic Part B

Lab Objectives:

§1. To learn disciplined, design procedures.

§2. To continue to learn how to layout cells for a cell library.

§3. To provide instruction and exercise in the synthesis of simple logic cells from hand-crafted transistors.

§4. To gain experience in layout verification via SPICE simulation of extracted layouts.

Pre Lab:

1. Read Chapter 2 of the textbook pp. 48-59.
2. Review Magic Tutorials #1, #2 & #3.
3. Read this lab carefully.
4. Have the schematics drawn and labeled in xfig (or equivalent drawing tool) and READY at the START of your lab.

4.1 Background

Successful IC designers must possess the ability to design full custom logic cells at the transistor level. Designers will often need to design customized circuits despite having a cell library. Regardless of how complete a particular library might it is impossible to anticipate every need. The geometry of a particular design might require changes in the shape of a cell which is extensively reused. This can often save a very large area. It is also possible that a design may require a specific cell which is not available for general use; again the designer needs to provide a custom circuit. When the device technology is scaled down to a more advanced process with smaller minimum feature sizes the entire cell library must be redesigned. This can often lead to significant layout changes since the

more advanced technologies often allow additional layout features which will dramatically change the basic cells. It is also possible that the designs of the present library will no longer work in the new technology due to second-order effects.

The basic storage elements, latches and flip-flops create the foundation of sequential logic circuits. Digital systems often employ combinational logic for operations and sequential logic for control. Sequential logic circuits are also used in high-speed arithmetic functions. For example, registers must be used to buffer a cascade of adders. Since adders are asynchronous, there is a risk that the adder will operate on the data before it becomes available. Registers synchronized with the clock and other events solve this problem. Registers are composed of single-bit flip-flops and latches are composed of single-bit storage elements.

In this lab you will be building and verifying simple storage elements, namely latches and flip-flops. These circuits are important in any digital synthesis and they also serve as useful starting points for students to aid in developing their skills in layout. Later in the course we will discuss latches and flip-flops; it helps to know something about them prior to the discussions. We will review the basics, then start by designing latches based on pass transistor logic. We will construct master-slave flip-flops which use these latches as building blocks. All of the circuits will need to be verified at the circuit level, in Spice.

4.2 Latches & Flip-Flops

A single-bit latch is the most fundamental storage element used in logic design. The simplest controllers depend upon some sequential operation. A latch offers the ability to synchronize a given state change with other related events. In addition, a logic state can be "held" for an indefinite period; hence the latch provides storage. In contrast to the single-bit latch, standard logic gates are *memoryless*, meaning that the logic equation does not consider past logic states. In general a latch can be *synchronous*, if its states change with the system clock or it can be *asynchronous* if its states change independent of the system clock.

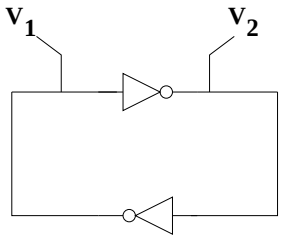


Figure 4.1: A static latch synthesized from two inverters can hold the voltage levels V_1 and V_2.

All static latches are synthesized with a pair of logic gates connected in a loop. The value of the inverter outputs will not change (unless power is removed); hence the term static latch. The simplest example of a static latch is composed of a pair of inverters as shown in figure 4.1. You will remember from the SpiceB lab, when ring oscillators were discussed, that a loop formed by an even number of inverters is stable in two states; e.g. *bistable*. For example, once the voltage, V_1, in figure 4.1 rises above or below the inverter logic threshold the output state, V_2 will be determined. If V_1 remains unchanged, as shown in the figure, then V_2 will actually force V_1 to remain at its original value due to the regenerative property of inverters connected in this manner. In order to make a practical latch control for writing and holding data in the latch must be supplied. This will be the main topic of this section.

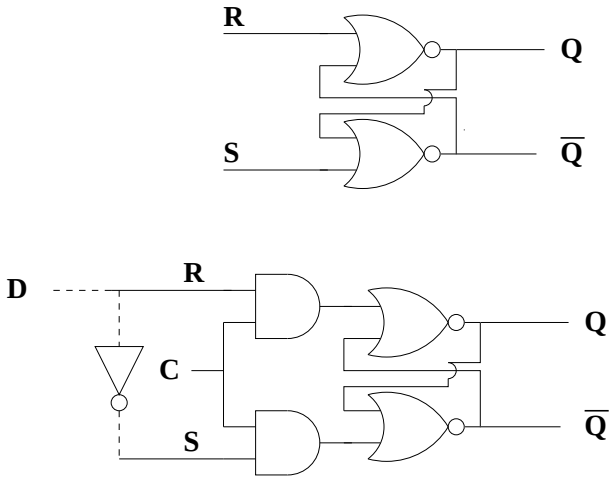


Figure 4.2: Simple RS latch, clocked RS latch and clocked D-latch examples.

The most common first example of a latch shown to students is the R-S Latch (sometimes called the R-S Flip-Flop). The R-S latch can be realized with NAND gates or NOR gates. A version of this latch using NOR gates is shown in the upper half of figure 4.2. This approach to latch synthesis is selected because the fundamental units available to students in a first logic design course are typically AND, OR and XOR gates (or their inverted forms). The R-S latch is asynchronous; however, it can easily be modified to a synchronous version by incorporating additional logic gates. A version of the clocked R-S latch shown in the lower half of figure 4.2. The R-S latch is not often used. The undefined state when R and S are equal to a logic "1" presents some problems. These problems can be overcome by forcing the R and S inputs to always be inverted versions of each other. This is commonly known as the *D-latch*. The D-latch can be synthesized directly from the R-S latch; this is shown in the lower half of figure 4.2 by connecting the inverter between the "R" & the "S" inputs.

There are in fact many implementation options available for one to synthesize

a D-latch. One can use a combination of NOR gates, AND gates and inverters as shown in figure 4.2. It is also possible to synthesize a latch by opening and closing the feedback loop in the static latch. For example, if we were to add a switch to open and close the loop in figure 4.1 we would have the option of changing the value of the bit stored. A common way to introduce this feature is to add a 2:1 multiplexer (MUX). There are two possible options: *positive or negative phasing*. The mux select to write can be a logic "1", which would indicate a positive phased latch or it could be a logic "0" which would indicate a negative phased latch. Both examples are shown in figure 4.3. The positive phased latch is written when the "W" line is high and the bit is stored when the "W" line is low. The negative latch uses a complementary write level; in other words data is written when the "W" line is held low and stored when the "W" line is high. It is easily seen that when the "W" line is "active", meaning that the latch is being written, that the bit at V_{in} is immediately seen at the Q and $\bar{Q}$ outputs. In other words, the latch is said to be *transparent* when the "W" line is active. This property is shared by all latches.

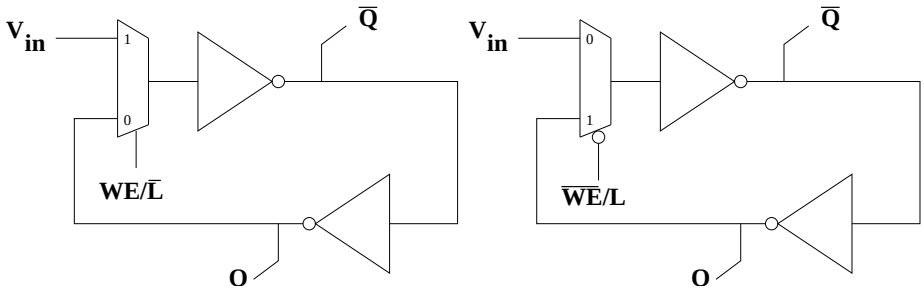


Figure 4.3: Simple RS latch, clocked RS latch and clocked D-latch examples.

Flip-Flops differ from latches in two principal features. First, flip-flops are *edge-sensitive* while latches are *level-sensitive*. Data is typically written on during the rising edge of the clock. While the data is being written the output is held during the rising edge of the clock. The output of the flip-flop will change during the falling edge (please note that data can be written on the falling edge which would mean that the data at the output would only change on the rising edge of the clock). This brings us to the second difference between flip-flops and latches; flip-flops are not *transparent*. This means that the input bit is not seen at the output when data is written to the flip-flop. There is also a difference in the clocked delay since a latch has a minimum delay of $\frac{T}{2}$ while the flip-flop has a minimum delay of a full clock period.

There are several ways to introduce edge triggering. Some gates and a pair of inverters can be used as shown in figure 4.4. There are many problems which make this type of flip-flop impractical for IC design. The *master-slave* flip-flop is the approach that is most frequently in IC design. A master-slave flip-flop is realized by cascading opposite phased latches which use a common clock signal. A common choice is to cascade a positive latch with a negative latch. There are

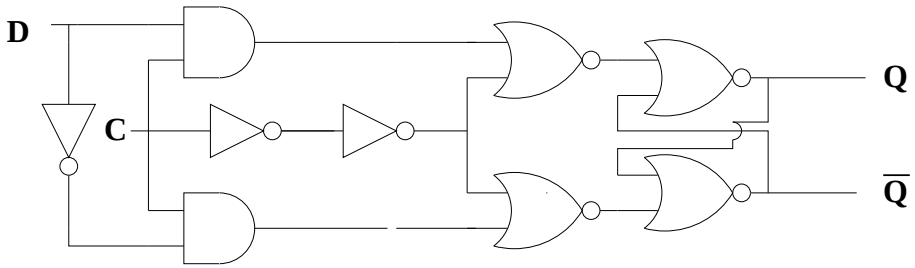


Figure 4.4: D flip-flop using inverters for edge-triggering.

two examples illustrated in figure 4.5. The first version combines the clocked D latches described earlier. The opposite clock phasing is realized by inverting the clock signal. The second version uses the muxed latch implementations. The positive latch is realized by using a multiplexer which writes when the clock is high and the negative latch is realized using an inverted version of the MUX.

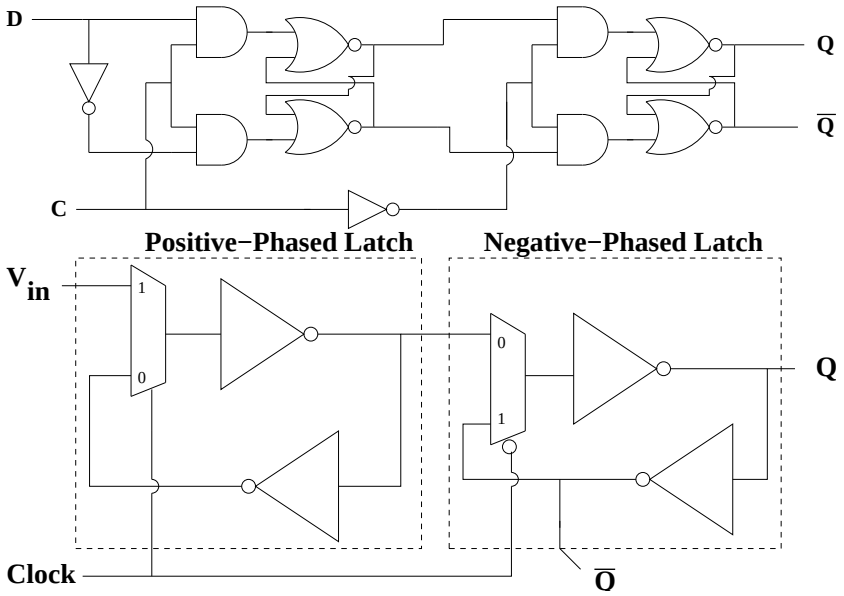


Figure 4.5: Two versions of master-slave D flip-flops.

All of the above are static *D Flip-Flop*s. D flip-flops are the most common versions used in integrated circuit design. The *JK Flip-Flop* is used in some control applications. The JK flip-flop is similar to the R-S flip-flop; however, the input state "J=1, K=1" is defined (as opposed to being undefined in the R-S flip-flop). A *T Flip-Flop* can be realized from a J-K flip-flop if "J=K=1". This means that the flip-flop will toggle in when "J" and "K" are both set to the logic level "1". All versions of flip-flops can be realized from each other;

e.g. a D flip-flop, with the addition of some logic gates, can function as a J-K flip-flop.

4.3 Static Latches Implemented with Pass Transistor Logic

We will be synthesizing multiplexed latches described in figure 4.3. The multiplexer can be implemented with pass transistors or transmission gates. This is typically chosen over a CMOS implementation due to the number of transistors required to realize a CMOS multiplexer (approximately 10). Only two transistors are required if n-channel and p-channel devices are used to implement the switches. This is illustrated in figure 4.6. Practical latches should use an additional inverter to obtain the Q output. Since the voltage levels will not reach the supply rails it is safer not to use the inverter tied to the pass transistor to drive any other logic elements. The logic state can actually be changed by the load if a designer is not careful. *Please note that the reason for the dashed lines is to indicate that we will not add the additional inverter for this lab since our ultimate goal is to use these to synthesize flip-flops.*

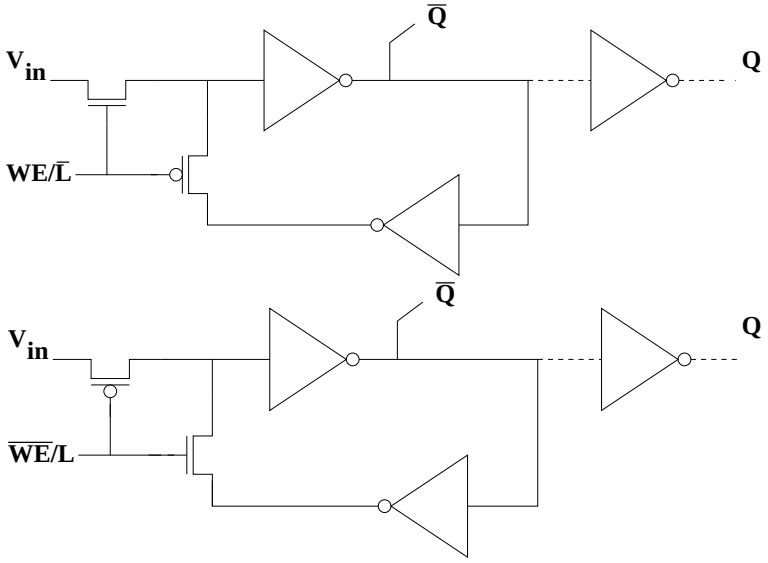


Figure 4.6: A complimentary pair of static latches using pass transistor logic.

Resets can also be added to these latch structures. A common addition to the latch is an active high or active low reset. This can easily be incorporated by replacing one of the latch inverters with a nor gate or a nand gate as shown in figure 4.7. The reset works by placing the latch in a state where the output will be fixed to a value regardless of what is stored on the latch; typically this

is a logic low.

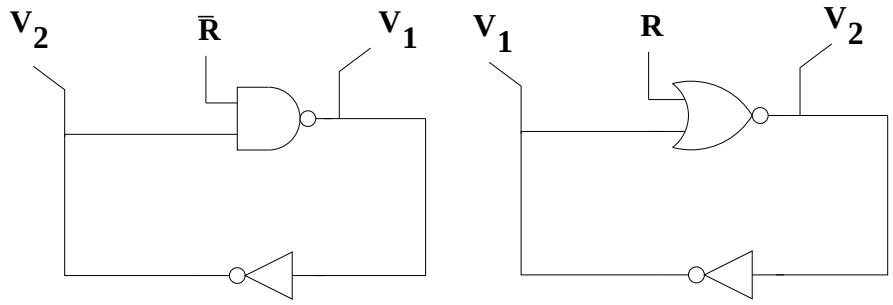


Figure 4.7: Re-settable latch with a nand and a nor gate.

We will now discuss how to realize master-slave D flip-flops from the latches we have synthesized.

4.4 Pass Transistor Implementations of Static D-Flip-Flops & T-Flip-Flops

The version of the master-slave static D flip-flop using multiplexers described earlier in figure 4.5 can be directly implemented by cascading the complimentary latches described in the previous section and depicted in figure 4.6. The resulting flip-flop is shown in figure 4.8. It is also worth noting that this version of the D-Flip-Flop can be used to implement the T-Flip-Flop by connecting the $\bar{Q}$ output to the "D" input shown by the dashed-line in figure 4.8. In the "T" flip-flop configuration the clock input is renamed to the "T" input and the "D" input is no longer required.

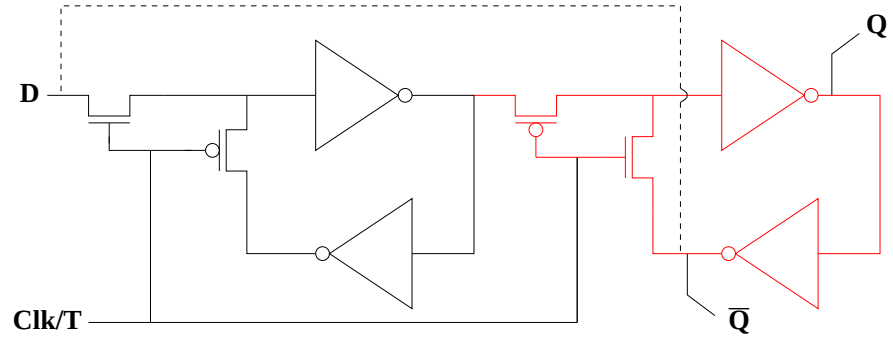


Figure 4.8: A T/D-Flip-Flop synthesized from two complimentary latches.

The resets described in figure 4.7 are implemented using a NAND gate in the master flip-flop. This is illustrated in figure 4.9. The flip-flop will not be reset for $\frac{1}{2}$ clock period after the reset changes from high to low since the reset state is not immediately applied to the second latch. If an immediate reset is needed, then an additional nand gate can be included in the second latch in figure 4.9. This will provide a low at the Q output and the contents will be immediately erased. The dashed line indicates how this could be converted to a T-Flip-Flop.

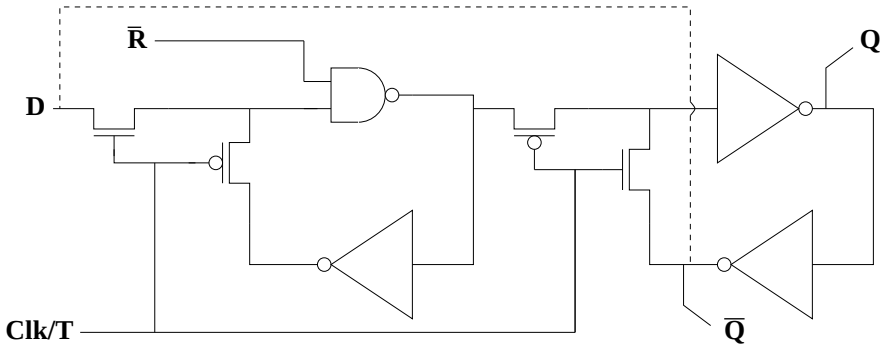


Figure 4.9: T/D-Flip-Flop with reset.

The T-Flip-Flop with a T input enable can be synthesized from an XOR gate and a D-Flip-Flop as shown in figure 4.10. The Q output and the T input are XOR 'ed together. When T is high the flip-flop will toggle and when T is low, the output will not toggle.

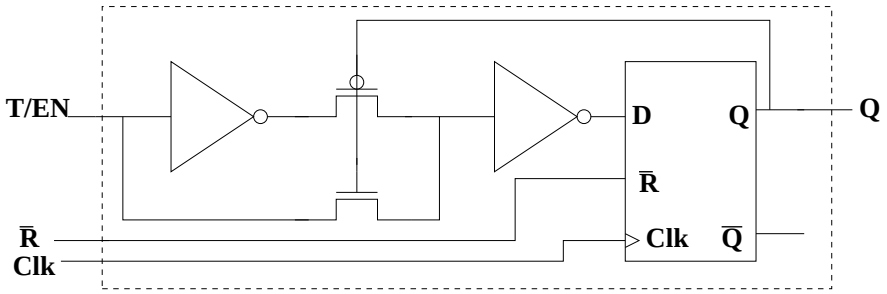


Figure 4.10: A second version of a T-Flip-Flop synthesized from an XOR gate and a D-Flip-Flop.

4.5 Cell Design Procedures

You will be designing D and T flip-flops for this lab. The flip-flop, unlike the simple gates you have been synthesizing, will require a methodical approach. This is necessary in order to realize the logic element with an efficient layout and to verify the layout netlist in Spice.

1) Create a high-level description of the cell.
   a) Logic/Gate level description.
   b) Output waveform as a function of the input waveform(s) and control signals.
2) Draw the schematic.
   a) Draw the transistor-level schematic from the gate-level description.
   b) Verify that the schematic realizes the correct logic function.
   c) Select appropriate $\frac{W}{L}$'s for each individual gate. Assume $\mu_n = 2\mu_p$.
   d) Label the schematic.
      i) Label the input(s), output(s) and control signals.
      ii) Label other nodes in the net list which are needed for testing (these can be removed once the testing is complete.
3) Layout the cell.
   a) Create the sub-cells first. For example, the flip-flops can be synthesized from complimentary latches. Draw these first.
   b) Place subcells. Make sure that the labels in the layout match the labels from the schematic.
4) Verify the layout.
   a) The layout must pass DRC checking.
   b) Visually check the connections in magic using the select tool.
   c) extract the layout netlist.
   d) Simulate the in Spice.
   e) Compare the results to the expected output waveform vs. input waveform.

4.6 Summary

Common approaches to implementing latches and flip-flops have been discussed. These are part of the primitive cell library; all must be verified using a circuit simulator which in our case will be Spice. We have selected multiplexer based latches which use simple pass transistors to realize the switching operations

needed to write and hold data. While these circuits are work well in the 1.2 μm process, they are not without problems in more current technologies. First, single pass transistors, especially the p-channel transistors can cause some problems as features sizes become smaller. It is possible to have an overlap in time where both devices actually will conduct if the rise time of the control signals (e.g. write, clock) are not sharp enough. Remember that MOS transistors will conduct even when the switch voltage is below the threshold value. The solution for problems like this is to move to a two-phase, non-overlapping clock. This will prevent both switches from potentially conducting at the same time. The next problem issue is driving large loads. The need for an additional inverter to provide the Q output in figure 4.7 has been discussed earlier (please realize this also holds for the flip-flops we will build). Designers need to guarantee that the latch and flip-flop will work for large fan outs. This can be accomplished by providing the correct load at the $\bar{Q}$ output in figure 4.7 and the Q outputs in figures 4.8-4.10 in a Spice simulation. It is possible that the static latches and flip-flops we are building could have the stored value changed. The solution is to add additional inverters or to buffer the output. A third problem results from the fact that we are using pass transistors. This will affect voltage levels at the input to the inverter fed by the pass transistors. One potential problem is static power consumption if the circuit is to be used for low power. Switching the pass transistors to transmission gates is a good start to solving such problems. The main thing to remember is that although the pass transistors are much more area-efficient, this is not a CMOS circuit !! So you need to be careful when voltage levels and power consumption are important. This again underscores the reason why a circuit simulator is used rather than a digital simulator. One other useful test is to simulate a delay line using multiple flip-flops; be sure to test for storing a logic "1" since the logic "0" provides a less useful result.

4.7 Pre-Lab

In order for people to move quickly through this lab we will start with a general structure which can be used to synthesize complimentary latches and the xor gate. You should create in magic and save PRIOR to your lab meeting. This will be checked at the beginning of the lab. This is shown in figure 4.12.

The cell in figure 4.12 is 40λ high and 44λ wide. The separation between the V_{dd} and Gnd power rails is 32λ. This will provide the "footprint" of the xor gate. In order to obtain a working "footprint" of the latch cells you will need to work on routing. A particular routing problem occurs with the clocks and some of the intermediate signals. People often will not see how to efficiently handle the routing of signals, clock lines, etc. quickly. In order to help with this it is worthwhile to mention that you will also need 3 buses when you are designing the latches (these are not needed for the xor gate). The actual "footprint" for the latch cells (with the exception of the resettable latch) is shown in figure 4.12. The size of the previous cell will be increased in height from 40λ to 61λ. This assumes 4λ wide buses with 3λ separation.

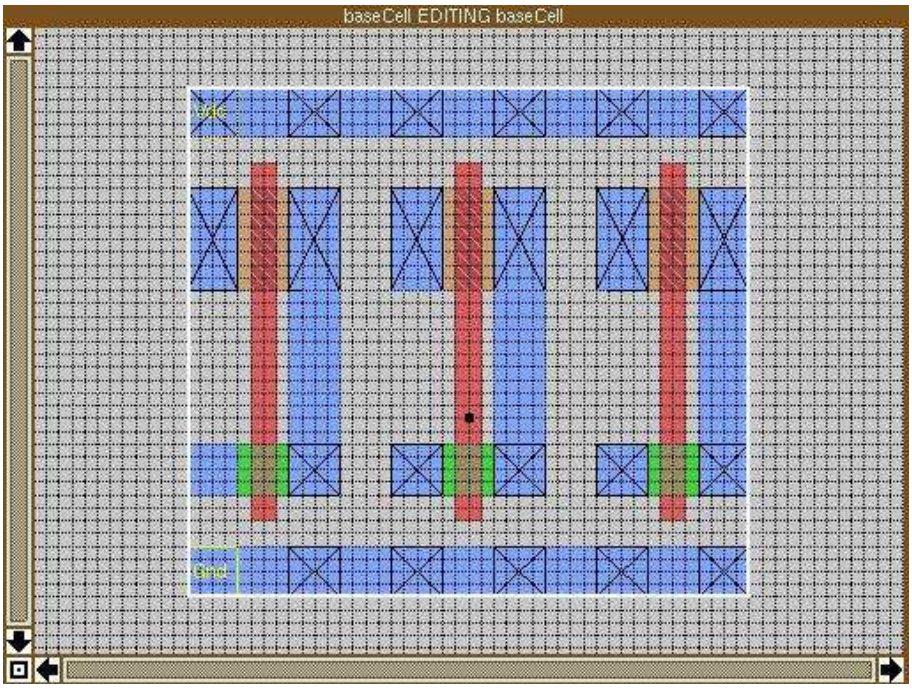


Figure 4.11: Your base cell.

Think about how you will need to make the connections in order to realize each of the latches. It is also important for you to devote some time to thinking about how to modify the baseCell.mag in order to incorporate the nand gate at the output.

1) Sketch the input, output and control waveforms before lab.

2) Prepare the magic layout figure 4.11.

3) You should have the connections for each latch figured out prior to your lab.

4) You also should have a layout solution to allow for replacing the inverter with the nand gate; to add the reset.

4.8 Lab Instructions

This lab will take two weeks.

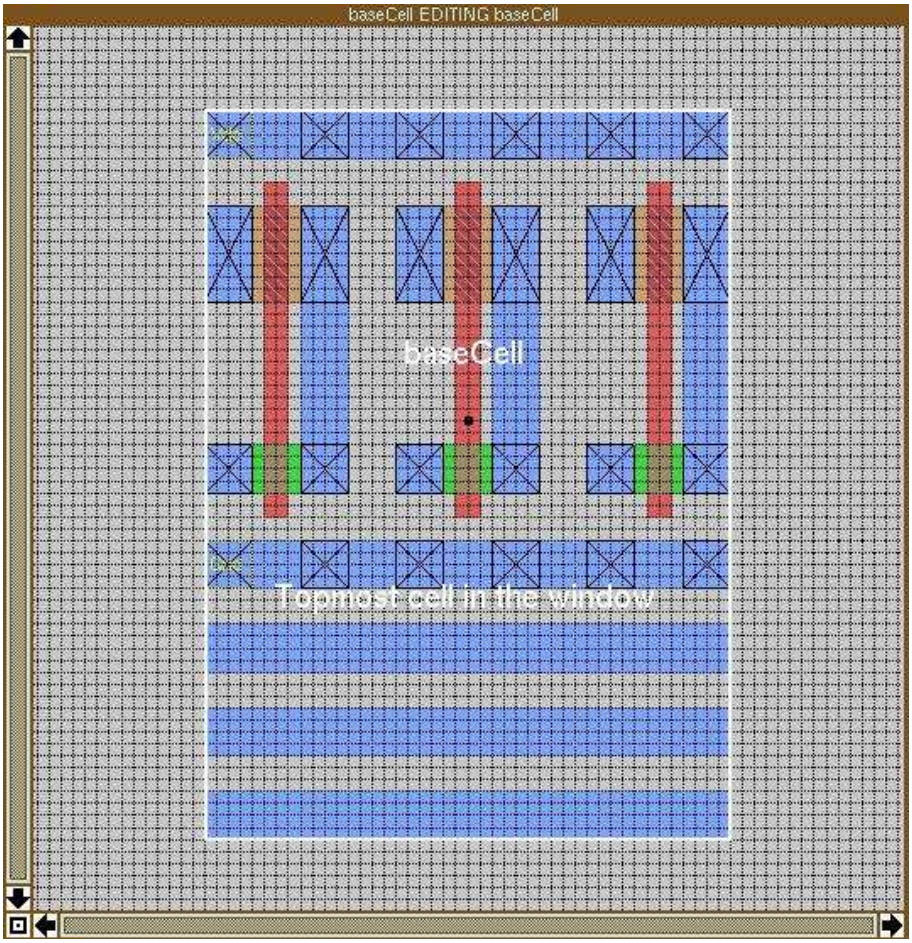


Figure 4.12: Your base cell with busses added.

4.8.1 Week 1

1) Design and layout the complimentary latches shown in figure 4.6 (make this a flat layout). You will use the baseCell.mag which should be prepared prior to the lab. Remember to rename the baseCell.mag each time you use it. Extract the layout and verify the design with Spice.

2) Modify the baseCell.mag to accommodate a nand gate at the output; save it as baseCell2.mag first.

3) Modify the positive phased-latch to incorporate the reset using the NAND gate described in figure 4.7 (right hand side). Be sure that you save the previous version of the latch because you will need it to get full credit for this lab. Extract the layout and verify the design with Spice.

4) Design and layout the xor gate described in figure 4.10. Extract the layout and verify the design with Spice.

5) Modify the complimentary latches to make them more area efficient. Re-verify the layouts in Spice. Remember to save your previous work so it is all available. This is not only important for your grade; it will also keep you from going insane in the event the modified version does not work.

6) Reduce the area for the xor gate. Verify in Spice.

4.8.2 Week 2

4.8.3 Week 2 Pre-Lab

1) You should continue to work on shrinking the size of your latch designs; you should also be certain that the latches you will use to construct the flip-flops have been thoroughly verified via Spice simulation PRIOR to the second lab.

2) Sketch the timing for each of the flip-flops you will build. Be sure to use your timing sketch in the verification and provide it in your lab report.

4.8.4 Week 2 Lab

1) Design the D-Flip-Flop in figure 4.9(make this a flat layout). Extract the layout and verify the design with Spice.

2) Modify the design the D-Flip-Flop to create a T-Flip-Flop as shown in figure 4.9. Include the D-Flip-Flop as a subcell. Extract the layout and verify the design with Spice.

3) Design the T-Flip-Flop in figure 4.10 using the D-Flip-Flop you designed in part 1. Include the D-Flip-Flop as a subcell. Extract the layout and verify the design with Spice.

4.9 Reporting

1) Tar zip all magic files, *.ext files and Spice files into a file named: $MagicBYourName.tar.gz$.

2) Copy this file to public directory listed on the web page for this lab.

3) Submit the hand drawn schematics; the label names in the schematics should match the label names at appropriate points in the layout and the simulation net lists.

4) Printout the layouts and simulation/verification results for each circuit designed.

Chapter 5

Magic Part C

Lab Objectives:

§1. To provide instruction in hierarchical design.

§2. To learn about layout with subcells and subcell wiring techniques.

§3. To provide training on the use of multiple windows and cell hierarchys in magic.

§4. Introduction to irsim, the digital simulator.

Pre Lab:

1. Review Magic Tutorials #4 & #5.
2. Read the demo on irsim and the manual provided on the ELE 447 web page.
3. Attempt to extract some of your earlier cells, namely the and gate and the T-Flip-Flops; use ext2sim and simulate them in irsim prior to attending this lab. Follow the examples in the lab write-up.
4. Read this lab carefully; Have the schematics drawn in xfig and READY at the START of your lab.

5.1 Background

The most practical method of designing analog or digital ICs is to synthesize complex systems from library of common, reusable primitive circuits. The primitive circuits form a cell library which is common to many projects. This implies a hierarchy of at least two levels: one at the top level where cells are used as building blocks and one at the cell library level. A given library cell might be used many times in a design; by restricting the designer's access to "read-only", problems can also be divided into groups at either the top level or at the primitive cell level. This dramatically simplifies both the design and verification of

ICs with large numbers of transistors. Most CAD systems and layout editors will provide hierarchical sub-division of circuits and sub-circuits. In magic hierarchical cells are implemented through the use of a sub-cell. Sub-cells can be read using the magic command ":getcell". In magic the hierarchy can be expanded indefinitely thus accommodating as many levels as needed for a given project.

Larger, more complex ICs to be broken down into smaller sub-circuits which require fewer steps to verify. Good designers will always employ many hierarchical levels to simplify testing and debugging. The hierarchy in magic should be ALWAYS be used; cells should NEVER be copied from the cell library, directly loaded into a layout where it can be modified (perhaps an unintended result). If a very large and complex system is constructed in a *flat* layout, meaning that the entire design is kept on a single level, then one cannot guarantee that two of the same simple cells will yield the same performance.

More than one drawing palette is often required when you are using the layout editor to handle more than one type of cell. This situation occurs so frequently that most layout editors will also provide designers with multiple palettes. This feature is referred to as multiple windows in the magic layout editor.

Digital simulation is an important tool for the verification of complex digital circuits. The synthesis of large digital systems becomes UN-wielding for a circuit simulator like SPICE or HSPICE. The time it takes to simulate increases; small problems can turn into big ones due to extraction errors. This is acceptable for small circuits with few transistors; however, when the number of transistors grows and the function is purely digital (meaning that the only 2 states on any given clock are a logic high or a logic low, e.g. V_{DD} and Ground) then a digital simulator is the only practical option.

The digital simulator in our CAD suite is known as irsim. Irsim was developed in the late 1980s by two researchers in RISC machines at Stanford University, Arturo Salz and Professor Mark Horowitz[1]. Professor Horowitz is now the director of the Computer Systems Laboratory at Stanford University. Irsim was a descendant of research and previous switch-level simulators developed during that period[2, 3]. Several improvements were made to irsim and revisions to the software were made through the 1990s. Irsim is public domain software and it is often used in conjunction with the magic layout editor.

In this lab you will be synthesizing a ripple counter and a carry-save counter using the T Flip-Flop cells developed in the previous lab as well as simple gates from the first magic lab. These circuits will be verified with irsim.

5.2 Binary Counters

Counters are one of the most common circuits in digital systems. Applications of counters include controllers, program counters, frequency dividers and state machines. Counters are synthesized from T flip-flops. When you first learned about counters they were probably implemented using J-K flip-flops with the

J-input and K-input both connected to V_{dd}. A J-K flip-flop connected in this manner is also a T flip-flop.

A counter is a special case of a *state machine*. A state machine is a sequential logic circuit which executes an algorithm. The algorithm is completed by stepping through a finite number of steps (or states). Each state is stored in a memory, typically, a flip-flop. Thus, a 2-bit counter is a state machine which uses 2 memory elements. Minimally, an implementation of a state machine requires state memory for its *present state* and logic to find the *next state*. The present states and next states for a simple 2-bit counter are illustrated in Table 5.1.

Present State	Next State
00	01
01	10
10	11
11	00

Table 5.1: State Table for A Simple Counter.

Generally, state machines, even counters, can be a great deal more complicated than the simple 2-bit counter. In this example, the next state is totally determined from the present state. General state machines can also use inputs in addition to state memory in finding the next state. Although the simple counter runs through all of its states, a general state machine need not use all of its states each time it runs. The general state machine is illustrated in figure 5.1[4]. The dashed lines are part of the general state machine which are not used for the simple counter counter example.

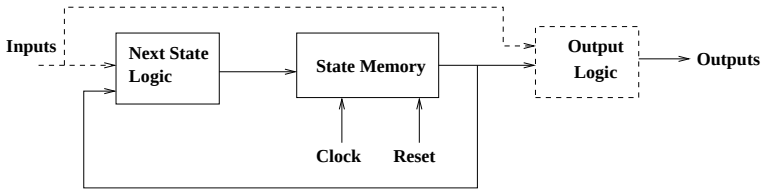


Figure 5.1: General description of a state machine.

Counters can be static or dynamic. If a counter is synthesized with dynamic flip-flops, then it will be referred to as a dynamic counter. If a counter is synthesized with static flip-flops then it is referred to as a static counter. Static flip-flops will always have a low resistance connection to either V_{DD} or ground for a given logic state while dynamic counters rely on the parasitic capacitance at nodes within the circuit to store charge to maintain a logic level. Because charge will eventually leak, the selection of static or dynamic counters depends mainly on the speed needed. For example, a dynamic counter might be a poor choice if it used as a machine controller or as a timer circuit in a wristwatch since these circuits will require maintaining a count or state for 10 mSec or even

longer. Static counters are used exclusively for such applications. On the other hand, if you are building a circuit which is expected to run at high speeds then a dynamic counter might be a better choice. However, you must include the testing performed (during the prototype phase and production) to be certain that low counter speeds are not required. Otherwise, some tests may fail.

We will be looking at specific asynchronous and synchronous static, binary counters in the sections which follow.

5.2.1 Asynchronous Ripple Counter

A ripple-counter is synthesized from T flip-flops by connecting the Q-output of the preceding flip-flop to the T input of the next flip-flop. The T-input of the first flip-flop is connected to the clock. The 4-bit ripple counter is shown in figure 5.2 using a T flip-flop. A reset for the counter is also provided by connecting all of the T flip-flop resets globally.

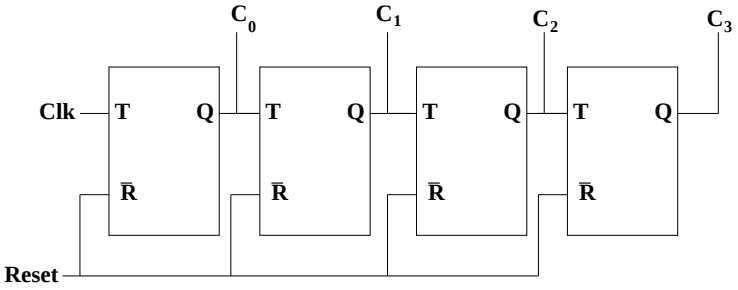


Figure 5.2: Asynchronous, ripple counter.

The ripple counter is asynchronous, meaning that the value of each bit used to count will not change state at the same instant following the clock transition. Information from the least significant bit will be carried to the next significant bit and this is repeated until the information reaches the most significant bit; information is *rippled* from LSB to MSB hence the name *ripple* counter.

The first flip-flop in figure 5.2 will toggle on every clock period. The second flip-flop will change state on every second clock period. The third flip-flop will change on every fourth clock period. This continues by powers of 2. Because the first flip-flop toggles once per clock period the maximum speed of the counter is limited to the maximum speed of the first flip-flop.

The ripple counter can be used to divide the frequency of the clock in powers of 2 since the output of the n^{th} bit will be a square wave with a period 2^{nth} times longer that the clock period. Thus, one application of the ripple counter is frequency division.

5.2.2 Synchronous Serial-Carry Counter

In a synchronous counter the state transitions take place on each clock cycle. This means that each T flip-flop must have a direct clock connection. One could construct a simple synchronous counter from T flip-flops with enables. The output of the preceding stage is connected to the enable of the succeeding stage. The T input is connected to the clock signal for each T flip-flop. The speed of this counter is limited by the time it takes for the LSB to serially reach the MSB. This problem can be resolved to some degree by "anding" ALL of the previous outputs to create the enable signal at each succeeding stage. The price will be the multiple input AND gates.

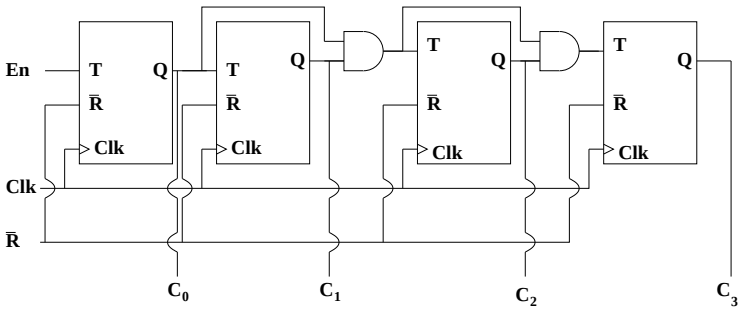


Figure 5.3: A synchronous, serial-carry counter.

The 4-bit serial-carry counter shown in figure 5.3 synchronizes the state transitions with the clock with clocked T flip-flops. Two-input AND gates combine the output of the preceding stage with the output of the present stage. This counter only requires an additional AND gate for all but the first and final stage.

The serial-carry counter in figure 5.3 ANDs the output of a flip-flop with its input to form a carry-out logic signal. With the exception of the first flip-flop, this is repeated for each binary digit. The speed limitation is the transition of the count which changes the logic state of all AND gates. The ripple counter is actual faster for the same given circuit technology.

5.3 Sub-cells, Multiple Windows in Magic & irsim

The layout of systems with sub-cells requires proficiency with the rules magic employs for hierarchical design. The use of multiple windows and changing the edit window is also important. Finally, efficient use of irsim is needed to verify larger, digital designs.

The following parts of this section provide exercises aimed at helping you to become familiar with sub-cells, multiple windows (both in magic) and irsim.

This is provided for you to do OUTSIDE of the lab.

You are expected to have read tutorials #5 and #6 as well as the irsim documentation. You should have the AND gate layout and irsim simulation complete at the start of this lab. You should also have the irsim simulation of both T flip-flops completed. All of these tasks should be completed by your scheduled lab time and day. You will be expected to know these parts of magic and irsim.

5.3.1 The AND gate

The AND gate is synthesized from the NAND gate and the inverter. Since we have created a NAND gate and an inverter in Magic Part A, you will use those sub-cells for this exercise.

Open magic without selecting a file. Use the following magic commands:

```
:getcell nand_12
:getcell inv_12
```

to load the nand and inverter as sub-cells.

Select the cell, then use the "x" macro to view the contents of the cell. Use the "X" macro (e.g. "shift-x") to remove the contents of the cell from view. You are allowed to move the cell, but, the level that you are operating on does not permit direct modification of the nand or the inverter cells.

Open a window with the inverter, place the cursor in this window. Make this window the edit window using the following magic commands:

```
:op inverter,
use the "s" macro with the cursor in the inverter window
:edit
```

Notice that you will need to stretch the inverter in order to make the power rails compatible with the NAND gate. Stretch the inverter using the "s" macro and the "shift-w,e" macros. Once the rails are identical in height, save the inverter cell.

It is desirable to overlap the diffusion contacts of the NAND gate and the inverter. Doing this directly will cause a drc error because the diffusion contacts are different sizes. Thus, we will need to re-size the diffusion contacts on the NAND gate. Use the following magic commands to accomplish this task:

```
:op nand
use the the "s" macro with the cursor in the nand window
:edit
```

The NAND gate should now be in the edit window. Modify the n-diffusion contacts in your nand gate. Make them the same size as the inverter cell n-diffusion contacts. Once you have done this, make sure there are no drc errors. Save the nand gate once this task is complete.

Now we are ready to work on the AND gate level. Place the cursor in the window with the nand and inv sub-cells. Make it the edit window using "s" macro, and the :edit command. You should now be editing the in the original window. Here, we will overlay the diffusion contacts of the nand and the inverter in order to minimize the area of the nand gate. Notice, however, that the nand gate must be "flipped". Select the nand cell. Use the "x" macro. Use the following magic commands:

```
:sideways
```

View the NAND gate using the "x" macro. Move the cells together, overlapping the diffusion.

You will now need to connect the output of the nand to the input of the inverter. Using the wiring tool, extend m1 from the nand output to just before the inverter input. Since the inverter input is poly, you will need a poly contact (overlap the poly by 1 λ when placing the poly contact to avoid a drc error).

Now, we have the and gate in a hierarchical design. The problem is that with such a simple gate, we do not want it to be hierarchical. To save a hierarchical cell as a flat file (flat meaning a single cell instead of a cell composed of sub-cells), use the command:

```
:flatten and_12
```

This will save the contents of this cell in the file

```
"and_12.mag"
```

.

Open the AND gate cell using:

```
:op and_12
the "s" macro
:edit
```

Erase the nand and the In, and Out labels. Replace the inverter output label with the "and" label. Now save the AND gate as:

```
and_12.mag
```

Make sure it is free of drc errors.

Use :ext to extract the and gate.

Verification will be carried out using irsim. You must convert magic's ext file to a file readable by irsim. From inside magic, type:

```
$\%$ :exttosim format SU
$\%$ :ext
$\%$ :exttosim and_12
$\%$ :exit
```

This will create new files with several extensions. One file will have a ".sim" file extension. Irsim uses a command file for simulation instructions. Copy the command file provided for the AND gate. The name of the command file is:

```
‘‘and_12.cmd’’
```

Now, you can run irsim. At the prompt use:

```
[\%]> irsim -s scmos100.prm and_12.sim -and_12.cmd
```

Review the result on the screen. Does the simulation appear to be correct for the AND gate ??

Sub-cells can also be selected from within the edit window. Return to the early steps where the NAND gate and the inverter cells are both opened as sub-cells; select each cell and make it the edit cell using a single window.

5.3.2 T-Flip-Flop

Simulate each of the T flip-flops designed in the previous lab. You will need to open each flip-flop in magic and then use the :ext command to extract each cell. You will again use ext2sim create the simulation files for each T-Flip-Flop.

You will need to copy the irsim command files: "tff1.cmd" and "tff2.cmd" supplied on the web page for this lab. To simulate the T flip-flops, at the prompt type:

```
$[\%]> irsim -s scmos100.prm tffx_12.sim -tffx_12.cmd$
```

Review the result on the screen.

5.4 Top Down Design Procedures

In this lab you will be designing two types of counters. In each case the counter will be designed using sub-cells. Although the counter is a fairly simple logic circuit, it will serve to illustrate the methods one should use for larger systems. In larger systems we tend to design from the "top down". In contrast, when we designed the flip-flops, which is true of all cell design, we employed more of a "bottoms up" approach. Thus, for a project which would include the design of some or all of the sub-cells as well as the final product, two different activities would take place.

Verification only makes sense if there is something to verify the cell or the subsystem or the entire design (and later the layout) against. This is generally true for the design of large systems and individual cells though the methods are somewhat inverses of each other. There is one fundamental relationship which will be identical for both cell design and high-level system design: You should have some expectation of how the circuit will function when it is working correctly LONG BEFORE starting the layout. In other words, you should know the basic timing waveforms and how the circuit will function; at least

at the "behavioral level". If you don't know this then how will you interpret the simulation results for an extracted cell ?? This sounds very simple; so simple that it should be automatically understood. However, there are countless examples which can be cited where this basic rule is violated and the system becomes nearly impossible to debug. When this happens, much of the work must be re-done.

One goal of this lab is to drive students in the direction where they WILL NOT START a layout without first developing an understanding of the detailed operation of the circuits they are attempting to build.

1) Create a high-level description of the system.

2) Divide the entire system into major subsystems.

   a) Simulate the entire system.
   b) Simulate the sub-systems and back-annotate results into the full system.

3) Generate a high-level logic design (Schematic, logic block or VHDL).

   a) Inventory the unique cells required.
   b) Update and/or create the new cells (if any are required).

4) For Each Subsystem.

   a) Decide on a hierarchy (1 level, 2 level, etc.).
   b) Prepare a detailed verification for each hierarchical level.
   c) Place Cells, adjust, overlap, flip , etc. before committing to wiring.
   d) Complete wiring.
   e) Place Labels.
   f) Fix drc violations.
   g) Verification.

      i) extract layout.
      ii) convert extracted files to irsim format (ext2sim).
      iii) simulate in irsim.

iv) compare simulation result to part (b)'s prediction.

5) Assemble subsystems.

i) Label the input(s), output(s) and control signals.
ii) Label other nodes needed for testing.

6) Layout the cell.

a) Create the sub-cells first. For example, the flip-flops can be synthesized from complimentary latches. Draw these first.
b) Place sub-cells. Make sure that the labels in the layout match the labels from the schematic.

7) Verify the layout.

a) The layout must pass DRC checking.
b) Visually check the connections in magic using the select tool.
c) extract the layout netlist.
d) Simulate the in HSpice.
e) Compare the results to the expected output waveform vs. input waveform.

5.5 Lab Instructions

1) Design the 4-bit ripple counter shown in figure 5.2 using the resettable T flip-flop from the previous lab. Perform the layout using the magic *array* command (described in Tutorial #4). Extract it and use ext2sim to create the irsim files. The irsim command file:

`‘‘rCounter_12.cmd’’`

can be copied from the web page for this lab.

2) Design the 4-bit synchronous counter shown in figure 5.3 using the T flip-flop with the T enable from the previous lab. Extract it and use ext2sim to create the irsim files. The irsim command file:

`‘‘scCounter_12.cmd’’`

can be copied from the web page for this lab.

5.6 Problems

1) A true enable should halt the counter without resetting it. This means that when the "En" input transitions from high to low the count is held independent of the clock. When the "En" input transitions back to high the counter should resume counting from the count it held. Does the enable on the serial carry counter meet this specification ?

2) How could the irsim script:

   ``scCounter_12.cmd'',

   be modified in order to investigate this problem ?

3) What modifications, if any, are required to make the enable function according to the specification provided in problem (1) ? (your answer should be based upon the result provided in the previous problem).

5.7 Pre-Lab/Reporting

1) Have the irsim simulation of the AND and both T flip-flops completed.

2) Have the block diagrams drawn and labeled for the layout.

3) Sketch the input, output and control waveforms before lab.

4) Tar zip all magic files, *.ext files and irsim files into a file named: $MagicCYourName.tar.gz$.

5) Copy this file to the public directory listed on the web page for this lab.

References

[1] A. Salz and M. Horowitz, "IRSIM: An incremental MOS switch-level simulator," in *Proceedings of the* 26^{th} *ACM/IEEE Design Automation Conference*, pp. 173–178, June 1989.

[2] R. E. Bryant, "MOSSIM: A switch-level simulator for MOS LSI," in *Proceedings of the* 18^{th} *ACM/IEEE Design Automation Conference*, pp. 786–790, July 1981.

[3] C.-Y. Chu and M. Horowitz, "Charge-sharing models for switchlevel simulation," *IEEE Transactions on Computer-Aided Design*, vol. CAD6, pp. 1053–1061, November 1987.

[4] C. R. Clare, *Designing Logic Systems Using State Machines.* McGraw-Hill, 1973.

Chapter 6

Magic Part D

Lab Objectives:

§1. To provide instruction on pad frames and how this is handled in magic.

§2. To provide instruction on the use of input and output pad buffers.

§3. To learn about wiring pad frames and verification at the pad frame level.

§4. Introduction to irsim, the digital simulator.

§5. To create cif files correctly.

Pre Lab:

1. Read this lab carefully.
2. Look at frame12, bufferx8 and bufferx3 prior to the lab.

6.1 Background

Communication paths between the circuit board and the integrated circuit (IC) for input/output (I/O) signals must be considered in order to make use of monolithic circuits. For example, if you fabricate the counter built in the previous lab then you would need to establish I/O connections between the labeled areas in the layout to a power supply, a clock and a logic analyzer in order to verify operation of the circuit in Silicon. It is common practice to place the die containing the circuit (e.g. the IC) in a hermetically sealed package. The pins or leads are often wire-bonded to the die. An alternative to this approach would be to wire bond from a circuit board to the bare die. MOSIS supplies a standard 40 pin DIP package with a cover which can be lifted in order to inspect the die. This package is obviously intended for prototype circuits.

The electrical connections to the die are NEVER made directly to the circuit itself. The wires from the package are simply too large. There is a pad frame which provides a larger conductive contact for each electrical connection. There are many pad frames in our library. You will be using a pad frame which supplies

40 pads which wrap around the 2.2mm x 2.2mm pad frame. This is compatible with the 40 pin DIP supplied by MOSIS. Pad dimensions are on the order of 100-200 μm for each dimension in order to provide enough area to attach a wire. Pads are often square.

Additional components designed specifically for I/O must be included for reliable operation. The gates in MOS devices are not able to cope with a great deal of charge. For example, the static electricity build up which shocks you from time to time is enough to puncture the oxide causing permanent damage to the IC. Clocked signals should be buffered to guarantee correct interpretation of the logic level.

In this lab you will learn how to make connections to the pad frame from our cell library. This is a final step in any IC design project. You will also learn how to create a CIF file which contains the exact masks used in fabricating the IC.

6.2 Pad Frames & Protective Diodes

The collection of pads and protective diodes is referred to as a pad frame. There are several pad frames in the cell library. The one we are interested in is frame12.mag. It can be accessed using the load command (inside magic):

:load frame12

Open magic and load the pad frame. Select the pad frame by placing the mouse in the middle and using the "s" macro. Then use the "x" macro to expand the pad frame. You will notice additional circuitry (sub-circuits) between the pads in the pad frame layout. These are protective diodes. If you do not notice the diodes, zoom into an area just larger than one of the pads. Use the "X" macro (to unexpand subcells) and repeat the "x" macro. You will see the diodes. If you use the "cntrl-x" macro you will also see the name of the cell containing the diode.

The diodes protect the circuit by absorbing current from any voltage greater than V_{dd} (+5 Volts, in our case) or less than Gnd. Figure 6.1 shows how the diode connection is made.

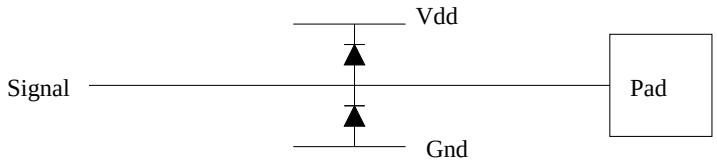


Figure 6.1: Diode protection at the pad.

The pad frame contains 40 pads. Each pad will provide a connection to a pin on the 40 PIN DIP package. The pad frame, frame12, is shown in figure 6.2 below. Notice that there are 2 strips of metal1 which surround the inner pad

frame. The inner strip is the *Gnd* rail and the outer strip is the V_{dd} rail. The V_{dd} rail is connected to the pads in the upper right and lower left corners. The *Gnd* rail is connected to the both pads in opposite corners. The V_{dd} and *Gnd* rails provide power for the diodes and buffers.

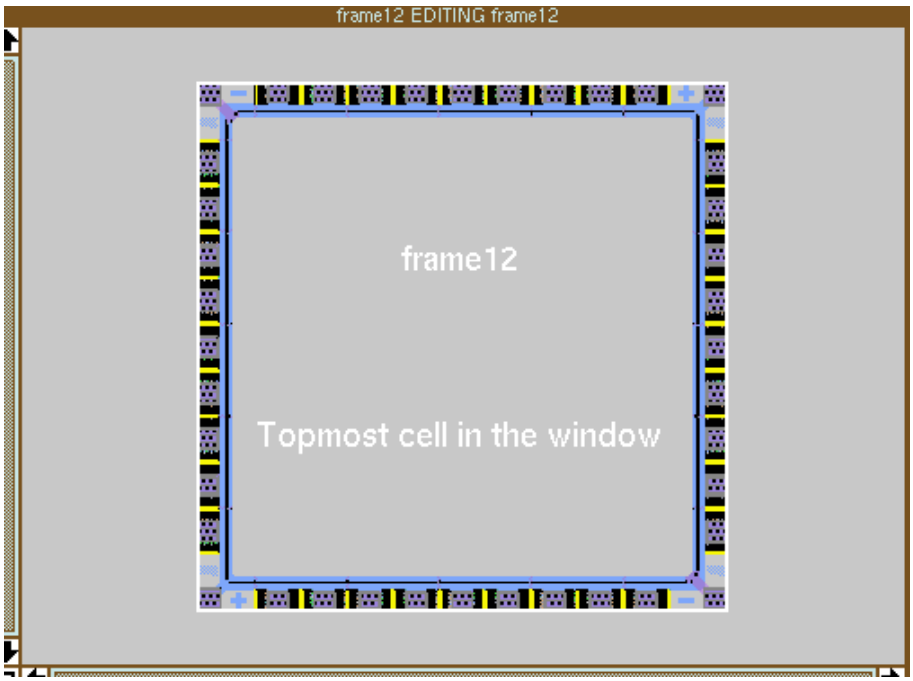


Figure 6.2: The pad frame: frame12.

The pads in the pad frame contain metal1, metal2, via and overglass. There is a glass layer which covers the entire die. The glass above the pads is cut away in order to allow for an electrical contact between the wire and each pad. Thus, the overglass layer is intended to cut an opening in the glass where it is drawn. There is a layer in magic named pad. The dimensions of the pad layer are given in μm and they do not scale. The relationship between the pad location in the frame to the pin location on the package is given in Figure 6.3.

Notice that the corner pads for the V_{dd} rail connect to pins 5 and 25 while the pads for the *Gnd* rail connect to pins 15 and 35. The numbers of the remaining pins can easily be determined from the schematic in figure 6.3.

6.3 Buffers

The wires bonded to the pads present very large capacitive loads on the order of several hundred fF to 1 pF or more depending on what the pin must drive. Since most gates drive loads which are significantly less than those at the pins

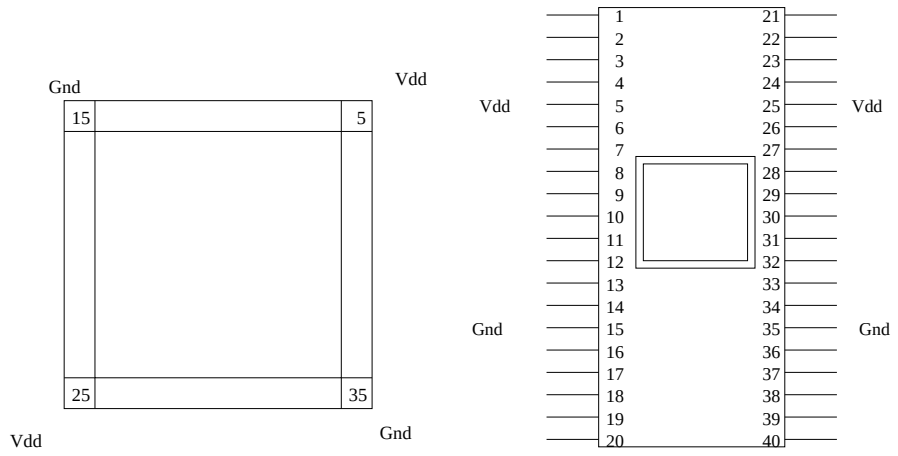


Figure 6.3: Schematics describing pad-to-pin connections.

output signals cannot be directly driven from the outputs of your circuit. As you already know, delays are directly proportional to the capacitive loading on a gate; thus, a load which is one or two orders of magnitude larger will present a proportional increase in the delay time going on and off the chip.

The solution to this problem is buffer amplifiers. A digital buffer amplifier provides the increased current necessary to drive the outputs at the pins. Buffers are simple amplifier circuits and they can be synthesized from inverters. Smaller buffers resemble a cascade of inverters while larger ones have additional complexity.

It is a good design practice to buffer critical input signals like clock inputs. Clocks are often routed using metal and poly layers. As we will see, these layers also increase the capacitive load. Thus, buffering is important. In addition, buffering input signals like a clock will increase the noise immunity. Please make sure that you ONLY buffer DIGITAL signals with the buffers from our library.

Magic files for 2 buffers, bufferx3.mag and bufferx8.mag are supplied on the lab web page. Download the buffers and store them in the directory where your pad frame for this lab will be stored. Download the diode3.mag file. The bufferx3 is to be used for the input clock and the bufferx8 is to be used for the counter outputs. These are intended to be used as subcells.

6.4 CIF Files

The final step for an IC project is the submission of the CIF file. You should recall that the CIF file contains the actual mask layers used by the foundry. The CIF format was described in this lab manual for Magic Part A.

CIF file generation is straightforward:

1) Perform a drc check on the design.

2) Simulate the design using irsim. Place the labels on the pads in order to obtain the highest level of confidence.

3) Verify the design from the pads via irsim.

4) Perform a final drc check.

5) Set the CIF output style:

:cif ostyle lambda=0.6(nwell)

6) Generate the CIF file:

:cif write myproject

To read the CIF file, do the following:

1) Start magic.

2) Set the CIF input style:

:cif istyle lambda=0.6(nwell)

3) Input the CIF file:

:cif read myproject

6.5 Lab Instructions

1) Modify the serial-carry counter built in last weeks lab. Provide the correct enable control.

2) Construct the pad frame for the counter.

a) Start magic.

b) :load frame12

c) Fully expand frame12.

d) :getc MYscCounter

e) Place the counter in the middle of the pad frame.

f) Make the following connections:

   i) Connect the counter V_{dd} to pin 39; label it.
   ii) Connect the counter ground to the pad frame ground rail.
   iii) Connect the counter Enable to pin 22; label it.
   iv) Connect the counter Reset to pin 24; label it.
   v) Verify the connections.

g) Connect the counter Clock signal to pin 23:

   i) place a box and cursor near pin 23.
   ii) :getc bufferx3
   iii) Connect the bufferx3 V_{dd} to the pad frame V_{dd} rail.
   iv) Connect the bufferx3 *Gnd* to the pad frame *Gnd* rail.
   v) Connect the bufferx3 input to pin 24.
   vi) Connect the bufferx3 output to the counter clock signal.
   vii) Verify the connections.

h) Connect the counter counter outputs, Q0, Q1 Q2 Q3 to pins 14, 12, 10 & 8:
   For each counter output:

   i) place a box and cursor the output pin.
   ii) :getc bufferx8
   iii) Connect the bufferx8 V_{dd} to the pad frame V_{dd} rail.
   iv) Connect the bufferx8 *Gnd* to the pad frame *Gnd* rail.
   v) Connect the bufferx8 output to the output pin.
   vi) Connect the bufferx8 input to the counter output.
   vii) Verify the connections.

i) Check the location pin connections in the pad frame. You should have the pad frame connections placed in order to supply the package connections shown in figure 6.4.

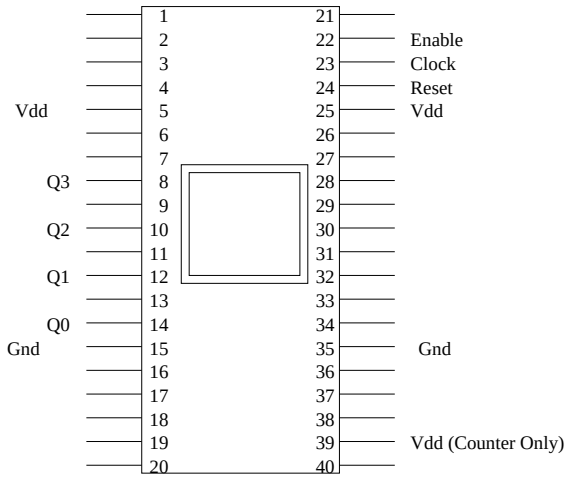


Figure 6.4: Schematic describing pin connections.

j) Save the design as CounterName

k) Perform drc check

l) Verify each pin connection and each counter connection with the select macro.

6.6 Reporting

1) Use mag2ps to print the completed counter alone.

2) Use mag2ps to print the completed counter & pad frame.

3) tarball the entire directory.

4) Tar zip all magic files, *.ext files and irsim files into a file named: $MagicBYourName.tar.gz$.

5) Copy this file to public directory listed on the web page for this lab.

Chapter 7

4 Bit Dynamic Sequencer

Lab Objectives:

The design, layout and verification of a 4-bit sequencer will be completed. The sequencer will be connected in a pad frame and made ready for design submission in the MOSIS 1.2μm double-poly CMOS process. The sequencer will have the following salient features:

§1. Reset, and $Enable/\overline{Halt}$.

§2. Parallel load from external output pins.

§3. Up-count and down-count capability.

§4. A parallel load feature which also works with the down counter.

§5. Introduction to irsim, the digital simulator.

7.1 Introduction

A sequencer is a counter which works in a manner similar to that of a program counter. A sequencer, like a counter, will count from 0 up to the full binary range it supports. The $Enable/\overline{Halt}$ command will halt the counter at a particular count. Once the counter is taken out of the halt state, it will continue to count in its programmed direction. Sequencers select memory locations which are to be read by a controller. Instructions are not always read in a sequential order. Sometimes there is a "jump" command. A jump command requires the reading of an address which is unrelated to the next state of the counter. This means that the sequencer must be capable of reading a non-sequential address. The mechanism for this operation is to switch the next state to a word which is provided by an external input, e.g. a parallel load function. The parallel load allows the sequencer to read its next instruction from input pins instead of the next count in the counter. Finally, this sequencer will also count down, e.g. count from 15 down to 0. The sequencer must accommodate the parallel load and the $Enable/\overline{Halt}$ capability when counting down.

7.2 Sequencer Specifications

The serial-carry counter (figure 7.1)is the major subsystem in the sequencer. The changes required in the implementation of the Magic D lab must be included for the $Enable/\overline{Halt}$ function to operate. The parallel load will need to be incorporated. Finally, the down counting can easily be included by selecting the T-Flip-Flop's $\overline{Q}$ output instead of the Q output.

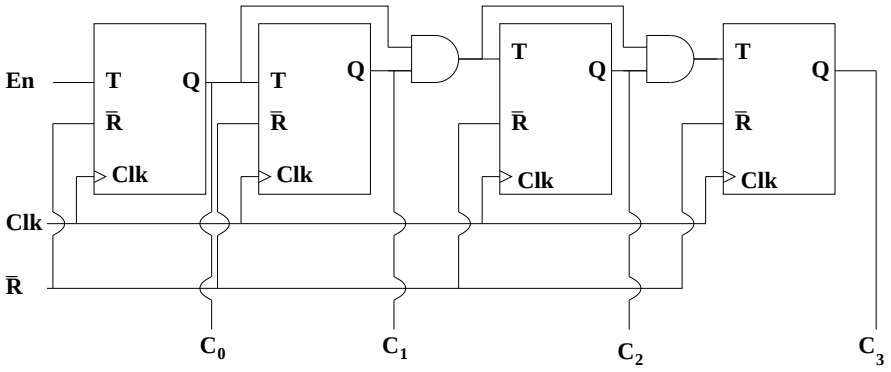


Figure 7.1: A synchronous, serial-carry counter.

You will need to develop a high level design. The pin out of the sequencer is given in figure 7.2. Notice that there are inputs for the word which is loaded via the parallel load. In addition, you must also provide a switch, e.g. the 2:1 mux, to implement this function.

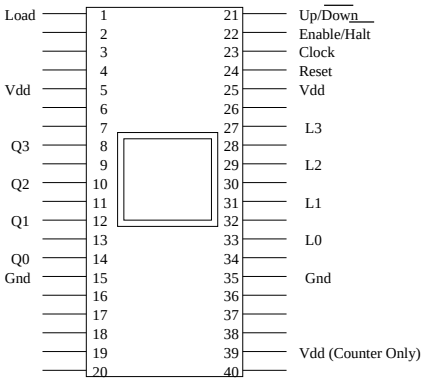


Figure 7.2: External pin diagram for sequencer.

The sequencer you are designing will employ a dynamic serial-carry counter. You will recall that you previously designed static counters by virtue of the

dynamic flip-flops used for the state control. A dynamic counter will instead utilize dynamic flip-flops.

The implementation of your sequencer will require some new cells. Thus, there are two primitive cells you will need to design to complete this project: the 2:1 mux and the dynamic T flip-flop. The selection of an up count or a down count could be accomplished by wiring nearly every signal to the pad frame. This places the burden on the circuits operating outside this integrated circuit. Instead, you must provide an on-chip solution. You will need several sets of 2:1 multiplexers in order to incorporate this feature.

7.2.1 Dynamic Latches & Dynamic Flip-Flops

The D-latch and D-flip-flop in figure 7.3 serve as examples of dynamic storage elements. The dynamic latch is synthesized from an inverter and a switch. The switch is either a pass transistor or a transmission gate. Typically, a transmission gate is used to conserve layout area. The latch is written when the switch is closed. The dynamic latch will maintain the voltage level at the last instant the switch was closed when the switch opens. It is assumed that the binary value is stored when the switch is open. The voltage level at the input to the inverter is held by storing charge on the parasitic capacitance between the switch and the inverter. The capacitor, C_p, represents the combined parasitic capacitance from the drain of the MOS switch and the gate of the CMOS/nMOS inverter. Thus the dynamic latch maintains its voltage level by storing charge on a parasitic capacitor while the static latch maintains its voltage level by providing a direct path to either V_{dd} or Gnd, saturating the output at the rails. You should also note that the output of the latch in figure 7.3 is inverted.

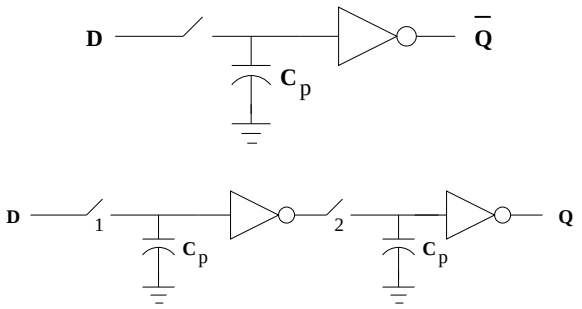


Figure 7.3: General dynamic D-latch & dynamic D-flip-flop.

The dynamic master-slave flip-flop in figure 7.3 is synthesized from two dynamic latches (also shown in figure 7.3) with opposite phased switches (hence the #1 & #2). The parasitic capacitances between the switch and the inverter is again represented by C_p for the D-flip-flop in figure 7.3. While this capacitor is usually omitted from schematics it must always be assumed to exist since the

charge stored on C_p contains the value of the latch or the flip-flop.

The advantage of dynamic flip-flops (and latches) is that they are both faster and significantly smaller than static flip-flops (and latches). The disadvantage is that pass transistors are not ideal switches. For example, junction leakage at the drain will eventually drain the stored charge, thereby changing the stored value. For high clock speeds this is usually not a problem; however, for clock periods greater than 200 μSec dynamic flip-flops and latches are unreliable. The debugging of dynamic logic is also more difficult. It is possible that due to unexpected charge sharing when the switch changes state and/or junction leakage that unpredictable behavior can result. A static latch or flip-flop will almost always work, even if poorly designed. This is not the case for dynamic storage elements.

A simple version of the dynamic D-flip-flop and T-flip-flop can be constructed using the approach outlined above. The resulting flip-flops are shown in figure 7.4. Notice that n-channel and p-channel transistors are used for the switches. This provides the opposite clock phasing without the need for a second clock signal.

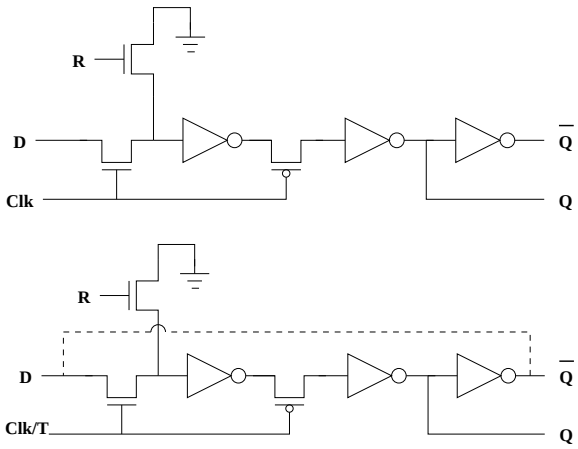


Figure 7.4: Resettable dynamic D & T flip flops.

The T-flip-flop in figure 7.4 requires that a $\overline{Q}$ output is available to be fed back to the D input. The $\overline{Q}$ output in the flip-flop requires an additional inverter. Otherwise, there will only be a Q output.

The T-flip-flop with an enable can be constructed from the D-flip-flop in figure 7.4. This is shown in figure 7.5. Note the resemblance to the static T-flip-flop designed in the Magic B lab. Obviously, the layout of the dynamic D-flip-flop will be very different than the one from Magic B, although the schematics symbols are identical at this level.

The T-flip-flop in figure 7.5 implemented with the D-flip-flop in figure 7.4 will be used in the serial-carry counter.

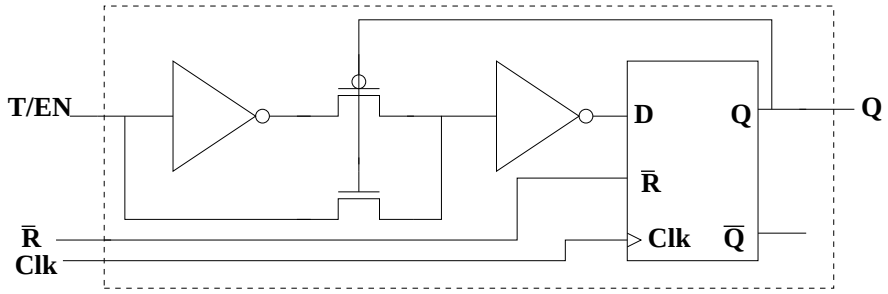


Figure 7.5: T-Flip-Flop with Enable.

7.2.2 2:1 Multiplexer

The 2:1 multiplexer can be implemented as shown in figure 7.6. The use of n-channel pass transistors imposes some driving limitation on the multiplexer. That is the reason for depicting the optional inverter at the output. This will of course invert the multiplexer output.

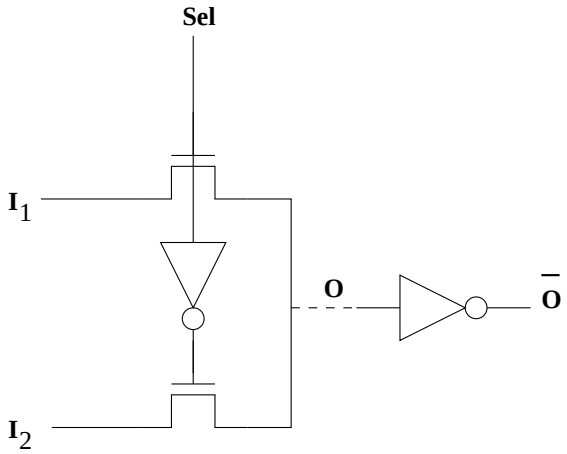


Figure 7.6: 2:1 multiplexer.

7.3 Summary

The circuit you are designing is implemented with dynamic flip-flops. While dynamic flip-flops dramatically increase speed and reduce area there are problems which a good designer will examine with a circuit simulator. These problems can become more pronounced as the feature sizes shrink. Clock feed through

can cause these types of latches to literally lose data. For example, if the clock has a rise time which is greater than the delay through an inverter, then it is possible for dynamic flip-flop can become *transparent*, meaning that the input is not properly latched. Inverters in deep sub-micron technology have delays on the order of 100 psec. Providing/guaranteeing shorter rise times at the clock inputs to the flip-flops will become more difficult to obtain. You should also consider that the clocks in these circuits must be routed to many places and that risk increases as the size grows. Minimally, the inverter thresholds will need to be shifted to minimize this problem. Replacing the pass transistors with transmission gates is only a partial solution. A two phase clocking system is needed in order to eliminate the clock feed through problem. This is the most likely design approach used in deep sub-micron technologies. While these circuits have their limitations they are very compact and can still be made to be useful if the designer is aware of the problems and takes proper measures to minimize the risks presented.

7.4 Instructions

1) Design a top level block diagram of the sequencer. Include all of the functions listed in sections 1 and 2.

2) Design the high-level tests of the sequencer (assume that you will use irsim).

3) Design the T-flip-flop with the enable.

   i) Draw the layout of the D-flip-flop from figure 7.4 in magic.

   ii) Extract and simulate the D-flip-flop in HSpice; verify the correct operation.

   iii) Design the T-flip-flop in figure 7.5 using the dynamic D-flip-flop.

   iv) Extract and simulate this T-flip-flop; verify the correct operation.

4) Design the T-flip-flop with the enable.

   i) Draw the layout of the 2:1 Mux shown in figure 7.6.

   ii) Extract it and verify the correct operation using HSpice.

5) Design the sequencer using the necessary 2:1 multiplexers and the 4-bit serial-carry counter. Implement the counter with the T-flip-flops you verified in HSpice.

6) Perform the layout of the sequencer.

7) Extract the layout of the sequencer and verify correct operation in irsim using the test designed in step #2.

8) Load frame12 and save it as ProjectI.

9) Include the sequencer and appropriate buffers as subcells. Wire the pad frame using the I/O pins specified in figure 7.2.

10) Verify the operation of the sequencer at the pad frame level using the pads in irsim.

11) Provide a separate D-flip-flop and T-flip-flop for lab testing; the unused pins on the 40 pin DIP will allow a sufficient number of connections. Make sure that you provide buffers for clock inputs and flip-flop outputs.

7.5 Reporting

1) Provide the block diagrams and test plan.

2) Provide plots of the D-flip-flop, the T-flip-flop, the 2:1 multiplexer, the spice netlists and the simulation results.

3) Provide plots of the sequencer and the sequencer in the pad frame.

4) Provide the irsim scripts and test results (from irsim).

5) Tar zip all magic files, *.ext files and irsim files into a file named: $ProjectIndYourName.tar.gz$.

6) Copy the "tarballed" directory to the public directory listed on the web page for this lab.

Appendix A

Introduction to HSPICE (or SPICE) in IC Modeling

§1. To familiarize students with a basic understanding of the most important types of circuit analysis available using HSPICE and SPICE simulators.

§2. To provide instruction for the representation and testing of actual circuits in HSPICE/SPICE.

§3. To provide a tutorial on HSPICE/SPICE commands, such as, include statements, sub-circuits, parameters, etc.

§4. To provide an overview of the level 3 and BSIM models and how they influence digital integrated circuit design. Also provide an overview of customized models.

Pre Lab:

1. Read Chapter 1 of the textbook pp. 1-25.
2. Read this lab. Write out the net lists prior to the lab.
3. Review "Recent SPICE Parameters" on the ELE 447 Course web page. Download the "BSIM3 Model HSPICE File".

A.1 Background.

Simulation of analog circuits is critical to developing reliable custom integrated circuits. IC design can be subdivided into several tasks: design of the cell library, hierarchical design, layout, design verification and measurements of the actual prototype circuit. The design of the cell library, parts of the hierarchical design where either power and/or timing is critical and verification involve the simulation of analog circuits. Often, when the measured prototype circuit does not function properly or works correctly but does not meet speed or power consumption requirements, circuit simulators are used in the debugging process.

Every digital logic gate is actually an analog circuit. Logic gates, latches and flip flops, for example, have voltages which can assume any value within

a specified power supply range. This means that a good digital IC designer must understand the analog circuit parameters and how they will impact a particular digital system. This cannot be accomplished without effective use of a circuit simulator. The design of a dynamic latch, for example, is best verified in a circuit simulator. It is impossible to verify the operation of memory cells without a circuit simulator. Circuit simulation is important in designing low power gates since the power consumption can be reliably predicted. You will also learn that not all digital cells restore the logic levels to 0 and the supply voltage. This is also not always accounted for during digital simulation.

Important effects due to device Physics are incorporated into a good analog circuit simulator. This is very important because the IC process technology today will minimally yield thousands of devices for a single fabrication cycle. Process parameters due to the device Physics will cause the performance of identical circuits to vary during fabrication. This means that a particular digital system must be able to operate reliably over the variations in the process expected to occur during fabrication runs. In addition, the operating environment, namely ambient temperature and case temperatures (of the IC package) also impact proper operation (especially for semiconductor devices). Since device Physics and temperature are intimately related, analog circuit simulation is the only means of effectively evaluating that a system (in large quantities) will work over the specified temperature range.

Due to the more precise analog modeling of each transistor, circuit simulators will provide accuracy well beyond that of digital simulators; however, circuit simulation has its limits, particularly when the number of transistors is large as is the case in subsystems of any complexity. Since the time steps tend to be nano-seconds or less, 1,000 or more additional computations may be required to model 10^{-6} seconds of activity ! The numerical accuracy is most effectively used to verify smaller cells in a large system.

There are two major types of simulators: digital and analog. A digital simulator only recognizes logic levels, 0 or 5 volts for the “0” and “1” in a 5 volt circuit, for example. Digital simulation is most useful in the verification of very large systems which assume the lowest level functions work properly. Precise timing analysis, measurement of the restoration of logic levels through drivers and comparison of performance over many wafer fabs can only be accomplished with a good circuit simulator.

Intelligent approaches to efficient use of CAD tools must be incorporated when the accuracy of a circuit simulator is needed for a very large digital IC. Since the entire IC is not easily simulated with an analog circuit simulator, then the measurements of a small circuit or several small circuits can be used to infer details of the operation at the IC level. The reason this is possible is that many IC designs involve the reuse of many identical primitive cells in order to realize a complex operation. One simple example is in the simulation of an adder; typically, the carry operations introduce complexity beyond what can be seen in simulating only a single-bit adder.

In ELE 447 we will be using HSPICE.

A.2 The History of SPICE and its Derivatives

The Simulation Program with Integrated Circuit Emphasis (SPICE) was created by Professor Donald Pederson in 1972 and presented at the 16^{th} Mid West Symposium on Circuits & Systems in April, 1973[**?**]. Professor Pederson made the source code for SPICE (known as SPICE1) publicly available from the University of California, Berkeley. SPICE development was funded by the U.S. Government for many years and it was one of the first software packages released into the public domain. Improvements were made and this lead to SPICE version 2 (SPICE2), developed by Dr. Lawrence Nagel[**?**] in 1975. Dr. Nagel started work in circuit modeling as a graduate student and scores of talented graduate students followed adding improvements to future releases of SPICE.

Because of its free cost and wide spread use in universities designers in industry and academia adopted SPICE as their primary circuit simulation tool. Between 1975 and 1983, development continued on advanced versions of SPICE2. In 1983, the final version of SPICE2, version G.6 (SPICE2G.6) was placed in the public domain[**?**]. SPICE1 and SPICE2 were developed in Fortran, which was not the most flexible implementation. During the 1980s, the development of SPICE3 commenced. SPICE3 was written in C and early versions were released in during the 1980s. Although the most recent public version of spice is SPICE3, most modern SPICE simulators trace their starting point to SPICE2[**?**], version 2G.6. The U.S. Government funding for the SPICE3 development was significantly less than what had been available during the development of SPICE2. Less attention was given to the details during the writing and verification of SPICE3. When SPICE3 was tested, there were many problems; some remain unresolved[**?**]. Thus, SPICE2 is often referred to as the "original" SPICE. Professor Pederson is and Dr. Nagel share the credit as the "originators"' of SPICE; however, many people have made valuable contributions. Commercial simulators have worked from SPICE2G.6 while SPICE3 has matured to the point where it is also reliable.

SPICE has become the industry standard for circuit simulation. Even the high-end cad suites which have fully integrated circuit simulators refer to these as *spice like*. Most foundries which fabricate ICs provide SPICE compatible circuit models so there is wide-spread compatibility. There are many commercial versions of SPICE available. This is owed in large-part its release into the public domain; it provided product developers with a common starting point which reduced the time and cost of bringing software to the marketplace.

The choice of a particular commercial version of SPICE depends on the specific interest of the buyer. Developers of commercial products integrate some or all of the features of the public domain versions of SPICE into their products. Since the vendors are under no obligation to include all of the features, the consumer must be aware of improvements and limitations of specific commercial versions. These products tend to target specific areas. For example, for many years P-SPICE[**?**] did not employ the most complex models for sub-micron IC design and currently, these are difficult to get at directly. The direction of PSPICE is to be integrated with commercial circuit board level design pack-

ages. HSPICE[**?**], on the other hand, has always had the less capable graphical interface but it was one of the first circuit simulation tools with reliable models for sub micron transistors in place. Today, many other circuit simulators, such as Smart SPICE, have caught up providing more choices in the selection of commercial SPICE tools. HSPICE and SmartSpice are backward compatible with Berkeley SPICE (the public domain version), with a few minor exceptions.

Public development in circuit simulation today tends to focus more on advanced models of Complimentary Metal Oxide Semiconductor (CMOS) transistors. Over 90% of the integrated circuits in production today are realized CMOS technology. The modeling of *deep sub-micron* CMOS transistors (e.g. transistors with channels much shorter than 500 nm) is a work in progress due to the constant shrinking of device geometries. For many years, the capital investments in this area have supported work at the University of California, Berkeley, also the developer of many of the current short channel models as well as SPICE itself. The results of this work are often integrated within the commercial versions of SPICE simulation tools.

A.3 Description of the SPICE Simulator

The SPICE simulator consists of two distinct numerical analysis tools: the simulation engine itself and element models. The simulation engine requires I-V characteristics for each element in a particular circuit. For any circuit element which has a non-linear I-V characteristic an element model is required. Refer to the circuit in figure A.1. The circuit elements (e.g. E_k) are generalized and can

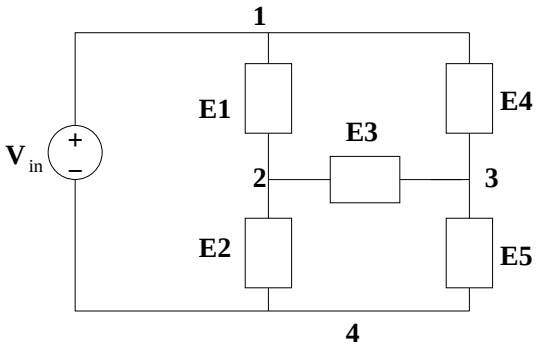


Figure A.1: Example circuit.

refer to one of 3 types of elements: 1) resistor, with a linear I-V characteristic, 2) charge storage element, e.g. a capacitor or inductor or 3) an element with a non-linear I-V characteristic such as a diode or a transistor (although more terminals must be identified). The nodes in the circuit are numbered.

If the circuit elements in figure A.1 are all resistors then the nodal or loop equations are solved in a manner which is numerically equivalent to the techniques you were taught in your circuit analysis course[**?**]. Gaussian Elimination

is used to solve the matrix equations. If one or more of the circuit elements is a charge storage device, such as a capacitor, then the element must have a model which accounts for the I-V characteristic[**?**]. For instance, a DC voltage applied to the terminals of a capacitor will yield a non-linear response as a function of time. SPICE models will tend to linearize the response over small ranges. If the voltages are measured over time, the case during transient analysis, then the model of the capacitor must be solved at each time-step via numerical integration. If one or more of the elements is a non-linear device, such as a diode or a transistor, meaning that there is a non-linear I-V characteristic, then a device model is used and the system of equations can no longer be solved using Gaussian Elimination. The Newton-Raphson algorithm, a non-linear root-finding method, is used to solve the system of equations[**?**]. This is done in a sequence of repeated computational steps where the algorithm will hopefully converge to a solution. This algorithm is not always guaranteed to reach a solution, hence, the infamous non-convergence error message.

When solutions are required over a series of time steps, the case when one is using transient analysis, then, a time-step algorithm is employed. A time step algorithm determines appropriate times to find solutions to the nodal equations. When there is a large change the time-steps are smaller and when not much is expected to change the computations occur less frequently with respect to time[**?**]. Modern SPICE simulators have at least two time-step algorithms[**?**].

The SPICE models consist of parameterized algorithms which help the simulation engine compute nodal voltages and currents for the particular types of analysis required. The semiconductor model parameters are extracted during each wafer fabrication and then incorporated into the SPICE modeling algorithm. The operation of the SPICE simulator is described in figure A.2.

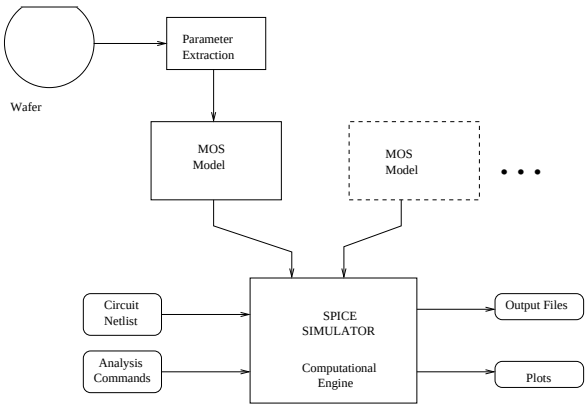


Figure A.2: Spice simulator block diagram.

It is important to realize that there is some variance between individual die on a given wafer. There are also differences between die selected from different

wafers. A unique set of model parameters is extracted from each wafer fab for this reason. In addition there are also model parameter sets known as *corner parameters* which cover the extent of the variance for a particular process.

This section describes the internal workings of a SPICE simulator. The user must communicate with the simulator. This is done by preparing a description of the circuit and the signals used as inputs. In addition there are voltage and current sources, e.g. the net list. The user selects one or more types of analysis, such as DC, AC, transient, etc. The output of the SPICE program provides data and graphical information to aid the user in the analysis and synthesis of electronic circuits.

A.4 Introduction to Analysis Using SPICE

SPICE requires an input file, often referred to as a *SPICE deck*. Before many of you were born, computers did not have files. Instead, each line in a file was entered onto a card and it was encoded with holes punched into the card. Although computers with batch inputs (e.g. card decks) have long since vanished, the name SPICE deck is still used today. Many references to cards and decks are made due to the age of the program. The input file must contain, a title (commented with *), model parameters for semiconductors, a net list, a list of analysis commands, output commands and an end statement, .end.

The information necessary for you to efficiently work with SPICE will be presented in the following sections.

A.4.1 The Net List

The net list provides a line by line description of the components in a circuit. Each component will have a particular name and node connections. Common node names are used to determine how the circuit is connected. The net list also provides device parameters, a model name (if appropriate) and possibly device geometries (for transistors and diodes). In ELE 447 the most important devices will be p-channel and n-channel enhancement mode MOS FETs and capacitors.

A net list can be generated from a hand drawn schematic. Net lists can also be generated via schematic capture from an automated schematic entry program. If the circuit exists in a layout editor, the net list can be extracted from the layout. You will need to know how to generate a net list from a schematic as well as drawing the schematic from a net list. Although this seems like a giant step backward, it is necessary. Later you will be working on the layout of gates. The skills you will develop in interpreting netlists will help develop the background necessary to become competent with the layout editor. Designers need to have some sense of how net lists are assembled in order to visualize the construction of logic cells.

The first step in this process is to draw the circuit to be analyzed. The second step is to label the nodes of the circuit. The node labels in SPICE can be either numbers or words. The node ”0” is hard wired to the ground

connection in SPICE. The inverter in figure A.3 illustrates the node labeling

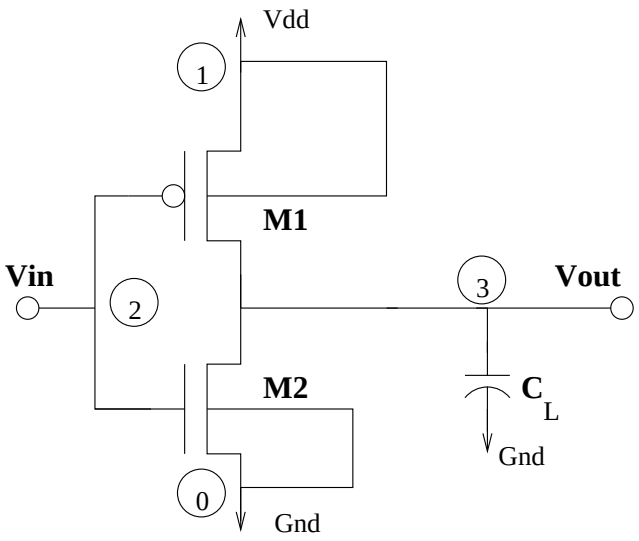


Figure A.3: Net list labeling for the CMOS inverter with a capacitive load.

scheme. Node 1 is tied to V_{dd}. Node 2 is tied to the gates of the inverter. Node zero is tied to the source of the n-channel device and also to the capacitor ground. The drains of the n-channel and p-channel transistors are connected together and to the output capacitor. Capacitors are seldom present in digital integrated circuits. The purpose of the capacitor is to estimate the load on the inverter. The load often contains gates, drains and sources of other transistors, poly silicon lines, metal lines or the pads for output pins. The last, but most important connection, is the bulk substrate. Remember, the MOS transistor is a 4-terminal device.

The net list entry for an MOS device contains the following fields:

Dname drain gate source substrate modelName W L AS PS AD PD

Dname is the device name. You must have a unique device name for each device in the net list. The nodes for the drain, gate source and substrate are then listed. This order NEVER changes. The model name follows. Then the channel width, W, and the channel length, L, are listed. Optional parameters are AS, PS, AD and PD for the source area, source perimeter, drain area and drain perimeter. These are useful in estimating the parasitic capacitance of the source and drain. For this lab, the optional parameters are omitted. We will assume all parasitic capacitances are lumped in C_L.

*	D	G	S	SS	Model	W	L
M1	3	2	1	1	CMOSP	W=4u	L=2u
M2	3	2	0	0	CMOSN	W=4u	L=2u
CL	3	0	0.1	pF			

The net list for the inverter with a capacitive load of .1 pF in figure A.3 is given above. You should be able to write out the net list for any of the simple logic gates.

A.4.2 SPICE MOS Transistor Models

Because the majority of commercial ICs are developed in CMOS, with the *enhancement-mode Insulated Gate Field Effect Transistor* or the *MOS Transistor* (both IGFET and MOS refer to the same device), and the device geometries are constantly shrinking. This leads to a number of *short-channel effects* which tend to cause errors which lead to differences between the simulation result and the measurement of the same circuit fabricated in Silicon. The SPICE answer to this is to advance generalized MOS transistor models. Thus, there are many semiconductor device models provided within HSPICE and most are dedicated to the MOS transistor. Selection of the level parameter indicates the desired model. The Level-1 model, developed by Schichman and Hodges[**?**], provides a very simple description of the MOS transistor based entirely on derivations from simple device physics. The Level-2 model, developed by Grove and Frohman[**?**, **?**], incorporates more detailed device physics. The square law relationship for the I vs V curves is assumed with some adjustments. These models are closely related to the transistor models presented in this course.

While a level 1 model is efficient for *hand calculations*, the resulting predictions do not agree with smaller transistor geometries (e.g. less than $5\mu m$ or so ...). Through comparisons between simulation results and actual Silicon devices, solid-state effects which were neglected become more dominant as the transistor geometries are reduced[**?**]. The Level-3 model[**?**] represents an early attempt at empirical modeling. The device physics of the transistor are approximated, relying on parameters which are a function of the device geometries in order to obtain better agreement with fabricated transistors. The simulation of smaller device geometries evolved into grid-based models, where subsets of empirical parameters are applied for different geometric classes. This is approach is referred to as "binning". Binning introduces new problems due to discontinuities at the boundaries between grids. Numerical smoothing algorithms were incorporated to eliminate the effects due to boundary discontinuities. This lead to the development of what was originally known as the Berkeley Short-Channel IGFET Model (BSIM)[**?**] or BSIM1. Since BSIM1 several newer versions have been introduced, BSIM2, BSIM3 and BSIM4 as device sizes have continued to shrink[**?**]. BSIM models provide the most accurate means of simulating sub-micron, MOS transistors. While they are the most accurate models, this does not necessarily mean that results obtained from BSIM are always in good agreement when predictions are compared with measurements from the fabricated devices. BSIM

should model devices more accurately than earlier SPICE models. There are many factors which can degrade the model parameter accuracy and much of this is beyond the scope of ELE 447.

The level-1, level-2 and level-3 models are referred to as 1^{st} generation models. BSIM1 and BSIM2 models are referred to as 2^{nd} generation models. BSIM3 and BSIM4 are referred to as 3^{rd} generation models[**?**]. In ELE447, we will use Level 3 and BSIM3 models.

From time to time, starting with this lab, you will also be asked to use simple level 1 models. The interesting part of this is that by specifying just a few parameters, results which match your hand calculations quite accurately can be obtained. Note that the unspecified model parameters are also required; default values assumed by the simulator for each parameter not found in the model used by the SPICE deck.

The properties of the MOS devices will change from one wafer fabrication to the next one. It is impossible to keep all parameters constant for each fabrication run. This is the reason model parameters are extracted from wafers following each fabrication. A good designer will select several fabrication runs which represent the "fast", "slow" and typical parameters when characterizing circuits which impact timing. The exact model parameters can be listed in the input file, but it is preferable to use the include statement as follows:

```
.include    "Level3Parameters.txt"     * level 3 model
                      or
.include    "BSIM3Parameters.txt"      * BSIM3 model
```

This will allow you to obtain results for several fabrication runs with several sets of level 3 parameters while only needing to change a single statement in the input file.

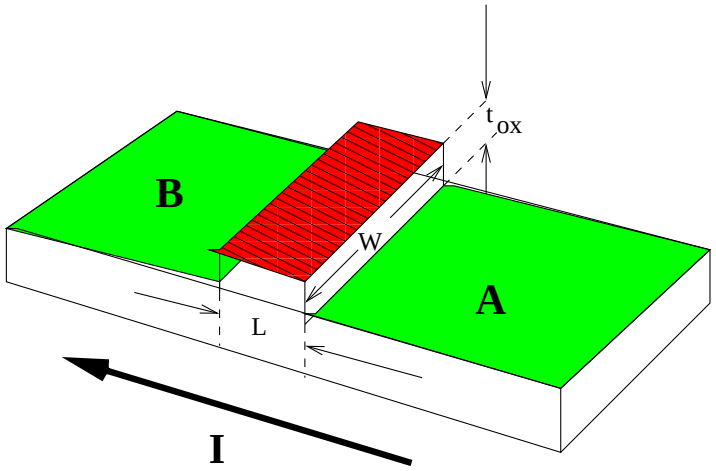


Figure A.4: MOS Transistor.

A simple diagram of an n-channel MOS (nMOS) transistor is illustrated in figure A.4. The green areas represent the highly doped *nType* Silicon (e.g. n^+) *drain* and *source*. The current in the nMOS transistor flows from the drain to the source. Since the MOS transistor is symmetric with respect to the source and drain the direction of current flow determines the source and the drain. In this case the green regions, labeled A and B, are the drain and the source, respectively. The red area represents the *gate* which is separated from the channel by a thin layer of SiO_2, the *oxide* or *gate oxide* layer. The channel lies under the gate oxide is lightly doped, *pType* Silicon (e.g. p^-; white regions); the *substrate* or *Body* of the transistor sets the potential of the channel and the pType Silicon under the source and drain. Since this voltage must be defined there is one additional terminal known as the *substrate or body contact*. nMOS and pMOS transistors are 4-Terminal devices, e.g. G, D, S & SS. Note that the doping types, concentrations and substrate contact are not shown in figure A.4. The distance between the drain and source, (e.g. L), is referred to as the *channel length* and the shortest distance that L can assume is known as the *minimum feature size*. The minimum feature size is one of the most important parameters influencing the operation. A reduction in L leads to better performance at the price of greater difficulty in its fabrication. The *channel width* is identified by the distance labeled W. The number of charge carriers present in the channel is controlled by the vertical electric field between the gate and the channel (which is why the substrate contact is of great importance). The current direction is set by the electric field from the drain to the source.

We will identify the most important nMOS transistor parameters. First, recall that the dimensions of the area under the gate, L and W, represent the channel length and width respectively. Increasing W will increase the current flow through the channel. Increasing L will require the current to flow over a longer distance, thereby reducing the current flow through the channel. The *p-channel MOS transistor* has an identical dependence on the channel length and width. The major difference is the polarity of the terminal voltages and the doping are reversed in the pMOS transistor with respect to those of the nMOS transistor. This is why the combination of an nMOS and pMOS transistors to realize logic gates is referred to as Complimentary MOS, or CMOS logic. One additional fundamental difference between nMOS and pMOS transistors is that the channel conductance is lower for the pMOS transistor by a factor of 2-3 with respect to an nMOS transistor. This is because the charge carriers in a pMOS transistor are holes instead of electrons. The threshold voltage to turn on the p-channel transistor is negative with respect to the source voltage (and substrate voltage).

The β of an MOS transistor relates the transconductance to the device geometries. β_n and β_p are defined as:

$$\beta_n = \kappa_n(W/L)_n, \beta_p = \kappa_p(W/L)_p \tag{A.1}$$

where κ_n and κ_p are the intrinsic channel conductances ($[\frac{Amps}{V^2}]$).

β is directly proportional to W and inversely proportional to L. Using this,

a direct relationship between the current and the device geometry represented by β is given by

$$I \propto \beta \tag{A.2}$$

where I is the current in figure A.4. κ_n and κ_p from equation (A.1) are defined as

$$\kappa_n = \mu_n \left(\frac{\epsilon_{ox}}{t_{ox}}\right), \kappa_p = \mu_p \left(\frac{\epsilon_{ox}}{t_{ox}}\right) \tag{A.3}$$

where μ_n and μ_p are the surface mobility of the n-channel and p-channel MOS transistors, t_{ox} is the oxide thickness under the gate (shown in figure A.4) and ϵ_{ox} is the permittivity of SiO_2. The mobility is the ratio the average drift velocity of the charge carriers in the channel and the electric field applied to the gate.

Finally, the oxide capacitance (per-unit area) of the gate, C_{ox}, is found by dividing the dielectric constant by the oxide thickness

$$C_{ox} = \left(\frac{\epsilon_{ox}}{t_{ox}}\right) \tag{A.4}$$

where $\epsilon_{ox} = 3.9\epsilon_o = 3.45x10^{-13}F/cm$. You should realize that ϵ_o is the permittivity of free space, a fundamental constant.

Models for the n-channel and p-channel transistor level-3 similar to the ones you will be using are shown below. The parameters we have discussed are enclosed in boxes. TOX is the oxide thickness in meters, VTO is the transistor threshold voltage in volts (when the substrate and source are shorted), KP is κ in $[\frac{Amps}{V^2}]$, and U0 is the mobility in $[\frac{cm^2}{Volt-Sec}]$.

```
1.  .MODEL CMOSN NMOS LEVEL=3 PHI=0.700000 [TOX=3.0400E-08] XJ=0.200000U TPG=1
2.  + [VTO=0.6221] DELTA=9.0430E-01 LD=1.1270E-07 [KP=7.6708E-05]
3.  + [UO=675.3] THETA=7.8760E-02 RSH=6.9150E+01 GAMMA=0.6478
4.  + NSUB=1.6310E+16 NFS=5.8890E+11 VMAX=2.1090E+05 ETA=1.2550E-01
5.  + KAPPA=3.5810E-01 CGDO=1.9202E-10 CGSO=1.9202E-10
6.  + CGBO=4.2839E-10 CJ=2.8122E-04 MJ=5.1486E-01 CJSW=1.4887E-10
7.  + MJSW=1.0000E-01 PB=9.6412E-01
8.  .MODEL CMOSP PMOS LEVEL=3 PHI=0.700000 [TOX=3.0400E-08] XJ=0.200000U TPG=-1
9.  + [VTO=-0.8289] DELTA=2.1790E+00 LD=1.1000E-09 [KP=2.1275E-05]
10. + [UO=187.3] THETA=9.8100E-02 RSH=5.9990E+01 GAMMA=0.3337
11. + NSUB=4.3290E+15 NFS=5.9090E+11 VMAX=2.4570E+05 ETA=2.6470E-01
12. + KAPPA=7.7710E+00 CGDO=5.0000E-11 CGSO=5.0000E-11
13. + CGBO=4.3405E-10 CJ=2.8763E-04 MJ=4.4034E-01 CJSW=2.0137E-10
14. + MJSW=1.2309E-01 PB=7.5007E-01
```

A.4.3 Sources, Power, Ground

You will quickly learn that circuits in SPICE do not work well if power and ground connections are missing. The simulator is just as bad as the infamous proto-board, with the major exception being that you will destroy fewer components using SPICE.

If you only apply power and ground the circuit, not much will happen. You must also apply a signal source at the input (hoping the circuit was correctly designed). There are several signal sources in SPICE: the sine wave, a piecewise linear function and the rectangular pulse. For the inverter we would like to provide a rectangular pulse. For example, V_{dd} and V_{in} can be implemented by writing the following statements:

```
Vdd   1   0   5V
Vin   2   0   pulse(0 5 5n 100p 100p 20n 45n)
```

This provides a 5 volt supply with respect to ground. The pulse statement contains the low voltage, high voltage, delay, rise time, fall time, pulse width and period. In the example, the pulse will transition from ground to 5 volts after 5 nsec. It has a 100 psec rise and fall time, it has a duration of 20 nsec and it repeats every 45 nsec.

A.4.4 Types of Analysis Available

There is still not much that will happen in spite of having a net list properly connected to power and ground with an input signal source. You must specify the type of analysis for SPICE to perform, the output format, and specific nodes you will need for the results of a particular analysis. The most important types of analysis are DC, transient and AC. There are also other types of analysis which include the operating point, small signal transfer function and pole-zero analysis.

The DC analysis provides output voltages for a specified range of input voltages. Low frequency operation is assumed. For example:

```
.DC     Vin   0      5   0.1
.print  DC    v(3)
.plot   dc    v(3)
```

This will provide voltages for the circuit at each note based on a DC input voltage ranging from 0 to 5 volts, incremented by .1 volt at each step. The print and plot commands provide the output voltage at node 3 for each input voltage step.

The transient analysis provides time series information. Over a specified time interval, the circuit response to a selected input, such as the rectangular pulse, is recorded. For example:

```
.tran    .1 n   50n
.print   tran   v(2)   v(3)
.plot    tran   v(2)   v(3)
```

The response of the inverter is computed from 0 to 50 nsec in .1 nsec steps. The original input pulse at v(2) and the inverter output v(3) are recorded nu-

merically and in a plot.

A.4.5 SPICE Specific Commands

The option statement is necessary for plotting. This statement needs to be included in the input file:

.option probe post=1

Global device scaling can also be performed. For example, if you need to scale an entire circuit by 0.3u, one would include a scaling statement in the input file:

.option probe post=1 scale=0.3u

A.4.6 Running SPICE (Specific to HSPICE)

HSPICE is run at the command prompt. The printouts from the list file can be obtained directly from the command line. If a printout of a particular analysis is all that is needed then HSPICE can be implemented as follows:

% > hspice inputFile.sp > inputFile.lis

The redirect symbol, >, in unix tells the operating system to write the contents to a file. Without the redirect symbol and the file, the statement, "hspice inputfile.sp", the output will go directly to the computer screen.

HSPICE results can be viewed graphically using the command cscope. Cscope is a graphical interface capable of reading the output plot files generated by HSPICE. You will need to open the plot files once in the graphical tool and then run the program.

A.5 Measurements

A.5.1 rise time & fall time

The rise time is an estimate of the amount of time it takes for the output of a logic gate to transition between the "0" state and the "1" state. The problem with such an estimate is that it is difficult to determine precisely where the transition starts and ends. The most common accepted approach to overcome this is to measure the time interval over the linear portion of the curve, as shown in figure A.5. This approximately measures the slope over the linear portion of the output waveform. If we keep the same V_{1r} and V_{2r}, e.g. the voltage levels, the rise and fall time for a logic gate output will always have the same ΔV,

then we need only record the time, Δt_r. This provides a relative estimate of the speed of a particular gate. By keeping the voltage range constant, direct comparisons between gates can be made. The fall time measurement is defined as the time it takes for the output of a logic gate to transition from the "1" state to the "0" state. This result can be obtained in a manner similar to the method described for the rise time measurement. The same voltage ranges are used. Traditionally, the rise time and fall time voltage ranges are chosen to be 10% of the maximum value and 90% of the maximum value.

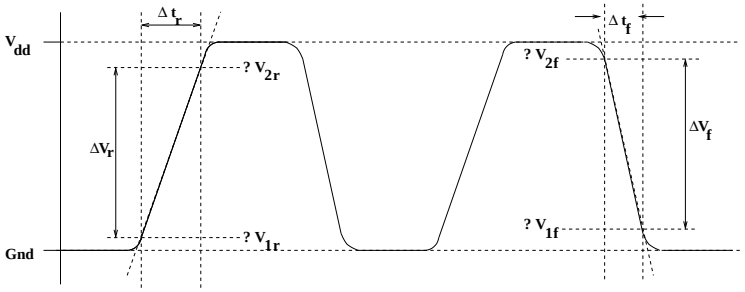


Figure A.5: Rise and fall time.

As technologies change, particularly as device sizes shrink and voltage supplies are lowered, alternative definitions of rise and fall time might be used. It is a good practice to keep these measures consistent when making relative comparisons. We will see other definitions of rise and fall time in our text book[**?**]. Despite the text book definitions, the rise and fall time definitions described earlier in this section will be employed because we will be working with a 5 volt process.

A.5.2 Inverter Threshold

The threshold of a gate is the lowest input voltage which causes a logic transition to start at the output. The gate-threshold of a CMOS inverter, V_{INVth}, is given by

$$V_{INVth} = \frac{V_{tn} + (V_{DD} + V_{tp})\sqrt{\frac{\mu_p(W/L)_p}{\mu_n(W/L)_n}}}{1 + \sqrt{\frac{\mu_p(W/L)_p}{\mu_n(W/L)_n}}} \tag{A.5}$$

where V_{tn}, V_{tp}, μ_n, μ_p, $(W/L)_n$ and $(W/L)_p$ are the n-channel and p-channel device thresholds, mobilities and geometries. Also notice that $\frac{\mu_p(W/L)_p}{\mu_n(W/L)_n}$ is equal to $\left(\frac{\beta_p}{\beta_n}\right)$, since C_{ox} will be identical for both n-channel and p-channel MOS transistors.

The gate threshold can also be found graphically. Plot the transfer curve, V_{out} vs. V_{in} characteristic for the inverter. Then, plot the line $V_{out} = V_{in}$.

The gate threshold is given by the intersection of the transfer curve and the line $V_{out} = V_{in}$ as shown in figure A.6. This method works for any logic gate. There are 2 possible ways to do this in SPICE. Can you think of a way to do this in SPICE ?

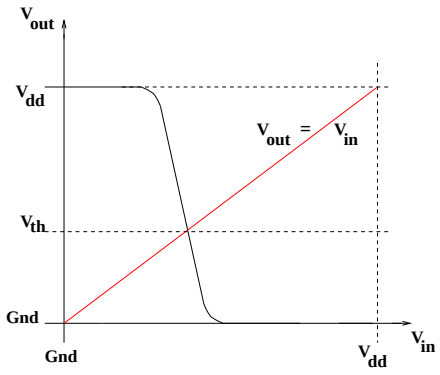


Figure A.6: Graphical method used to find V_{th}.

A.5.3 Gate Delay

The gate delay is defined as the time it takes for the output voltage to reach the input voltage threshold after the instant the input voltage crosses the input voltage threshold. Gate delay is defined separately for both the rising and falling inputs.

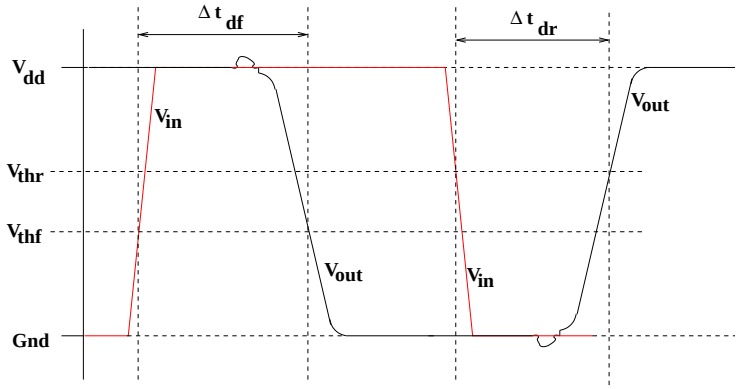


Figure A.7: Rising and falling gate delay time.

This is illustrated in the diagram in figure A.7. Notice that in figure A.7 we have separate delay times for the rising and falling input step. The gate delay definition is sometimes simplified by assuming symmetric thresholds, each at $\left(\frac{V_{DD}}{2}\right)$. The gate delay is then measured from the time that the input crosses $\left(\frac{V_{DD}}{2}\right)$ to the time that the voltage change of the output crosses $\left(\frac{V_{DD}}{2}\right)$. In this lab, we will be interested in finding exact delay times, so, we will measure the logic thresholds of the inverter.

Appendix B

Elementary IC Economics

This appendix provides simple examples which illustrate the steps required to estimate the variable cost per die and the non-recurring engineering costs (NRE) associated with integrated circuit fabrication. The basics of cost analysis are also presented. The goals are:

§1. Obtain preliminary UPC estimates at the die level.

§2. Develop a "reasonable" approach to estimate the total number of working die from a wafer lot.

§3. Estimate the total cost and profit as a function of the number of die fabricated.

B.1 Introduction

The market for integrated circuits is composed of buyers who are very discriminating. The reliability is expected to be extraordinary. The ratio of the number of working die to the number of die fabricated, the yield, is expected to be well above 90%. The development period must be minimized due to short product life cycles (you do not want to have your device ready at a time when the industry has moved away due to obsolescence).

There is a very large investment in the design, layout, and testing before the first production part can be produced. In addition, one must also advertise thus, the marketing and sales need to be included as part of the investment. The investment or fixed cost is often referred to as the non-recurring investment (NRE).

Success of full custom integrated circuit (IC) design is driven by economics; namely volume. The NRE can range from several hundred thousand to many millions of dollars yet the selling price of a single part, e.g. the packaged die sales price often ranges from \$0.50 to \$1.00. The price you expect to pay for a candy bar! That is the price the customer expects for the production integrated circuit. There are two important factors which determine a project to be profitable: (1) the per unit selling price must be greater than the per unit production cost and (2) the demand must be sufficient to earn money on the NRE.

B.2 Break-Even Point Analysis

There are some relations for cost which are used by our colleagues in business.

The first set is the definition of cost, given by

$$Cost = UnitCostXN + FC \tag{B.1}$$

where $UnitCost$ is the cost to produce a single unit, N is the number of units produced and FC is the fixed cost which actually would be the NRE from the previous discussion. They are one in the same.

Profit can be defined as difference between revenue and cost. For simplicity, revenue is the product of the unit price and the number of units sold. For a given number of units sold the profit can be expressed as:

$$Profit = Revenue - Cost = (price - UnitCost)XN - FC \tag{B.2}$$

where *price* is the per-unit price.

One can see that both profit and cost are defined by linear equations; that is, they are of the form $Y = mX + b$ where the value Y as a function of X is found using the slope, m and the intercept, b. This means that there is a variable cost and a fixed cost. There is also a variable profit per unit which is referred to as the *contribution* per unit. Contribution is defined as

$$\begin{aligned} Contribution - per - unit &= (price - UnitCost) \\ TotalContribution &= (price - UnitCost)XN \end{aligned} \tag{B.3}$$

This means that profit can also be defined as

$$Profit = TotalContribution - FC = Contribution - per - unitXN - FC \tag{B.4}$$

From this relation one can see that profit is equal to $-FC$ when $N = 0$ and it becomes positive *only* when the total contribution exceeds the value of FC.

When investing in a product, it is useful to know how many products one must sell before the *investment* produces a profit. This is often expressed in the following way: the number of products sold, N which makes $FC = TotalContribution$, hence a profit equal to zero. This is known as the *Break-Even Point* and the name indicates exactly what it means, e.g. how many units must be sold to recover the investment. The Break-Even Point, BEP, is defined as

$$BEP = \frac{FC}{Contribution - per - unit} = \frac{FC}{price - UnitCost} \tag{B.5}$$

The BEP is a very useful metric for the investor(s). If the total demand for a product is less than the BEP, the investor will not recover the investment; however, if the demand for the product is much larger than the BEP it is possible that the investor will earn a great deal on the investment.

The BEP value is only reliable if FC, $UnitCost$ and *price* can be found with good accuracy. FC and $UnitCost$ can be found from the design costs and the

production cost. The foundry often has good information for one to compute the total cost.

The stakes in this aspect of a business, especially a start up company, are very high. The sales price and demand must be obtained from the sales and marketing people. They have an important job. If their estimates are incorrect, investors can lose a great deal of money. The one thing you must learn about investors in a business plan is that paying them back is priority #1! Investors are not motivated by shear generosity. Their only motivation to invest in the first place is to earn more on their money over time than what they could in a bank account or a mutual fund; who could blame them. If they earn money on your idea, either the same investors will return or investors with greater resources will show up. If they lose their money then you can be sure there will be NO second opportunity. All kidding aside, there is not enough that can be said about being accountable when people place their trust in you.

B.3 IC Wafer and Yield Estimates

In order to estimate the cost of a custom integrated circuit design some information about the wafer fabrication must be known. The wafer is purchased in a lot; this means that the price quoted for several or more wafers. The lot sizes vary depending on the process and the foundry requirements. Thus, the cost per lot and the lot size must be known. The wafer diameter must also be known. Finally, the number of defects per unit area, the defect density, must be known. This number, the defect density, is considered proprietary to the vendor or foundry. It would be a criminal act to distribute proprietary information. Often this type of information will only be provided with a confidentiality or a non-disclosure agreement. We will use an arbitrary defect density value for teaching purposes.

B.3.1 Total Die per Wafer

The first step is to compute the total number of die per wafer. We assume that rectangular die are printed over a circular wafer (the wafer is actually flat on one side). Thus, the number of die per wafer does not quite divide evenly into the wafer area. Die which approach the edges cannot be used; you are also not allowed to combine two or more fractional die into one so these cannot be counted at all. The number of die per wafer can be estimated using[1]

$$N_{totaldie} = \frac{\pi R^2}{A_{die}} - \frac{\pi D}{\sqrt{2A_{die}}} \tag{B.6}$$

where $N_{totaldie}$ is the total number of die, R is the radius of the wafer, D is the diameter of the wafer and A_{die} is the area of the die itself.

It is not certain that the entire wafer will be patterned with die. In fact, this is usually not the case. possible that a number of potential die sites on the

wafer are to be used by the fabricator. So this estimate must be considered to be just that; an estimate.

B.3.2 Yield & Net Die per Wafer

Although the fabrication of die on a Silicon wafer is a very tightly controlled process you will find that not all die will work. The yield is the ratio of the number of working die, N_{netdie}, to N_{die}.

One would like to know what can be expected for the yield prior to fabrication of the wafer lots. Unfortunately, estimation of the yield is not straightforward. Early attempts at estimating yield assumed a uniform defect density throughout the wafer[2]. Binomial distributions and other standard probability density functions were often used. These all assumed that the defect density was uniform[2].

People quickly found that the assumption of uniform defect densities was not realistic making the estimates unusable[2]. Defect densities vary from wafer to wafer. The defect densities tend to be greater near the edges of the wafer and they also tend to occur in clusters. When one thinks of handling either by humans or machines, this is not surprising. Murphy developed a general relation for computing the wafer yield from a non-uniform defect density which is given by[3]

$$Y = \int_0^\infty e^{(-D_o A_{die})} f(D_o) dD_o \tag{B.7}$$

where D_o is the defect density, $f(D_o)$ is a probability density for D_o and A_{die} is the area of the die.

If a triangularly approximation to the Gaussian distribution is used then the yield is found by

$$Y = \left[\frac{1 - e^{(-D_o A_{die})}}{D_o A_{die}} \right]^2 \tag{B.8}$$

If instead $f(D_o)$ is a uniform probability density function then

$$Y = \left[\frac{1 - e^{(-D_o A_{die})}}{2 D_o A_{die}} \right] \tag{B.9}$$

Since Murphy more complex probability distribution functions have been investigated. This has culminated in the following yield expression[2, 4, 1]

$$Y = \left[1 + \frac{D_o A_{die}}{\alpha} \right]^{-\alpha} \tag{B.10}$$

where the constant α is a number which is empirically determined. Typical values are between 1 and 3. The value, $\alpha = 3$ provides reasonable accuracy for commercial CMOS fabrication today[1].

The product of defect density and die area, $D_o A_{die}$, determines the yield. It can be seen that the yield will dramatically decrease in proportion to increasing

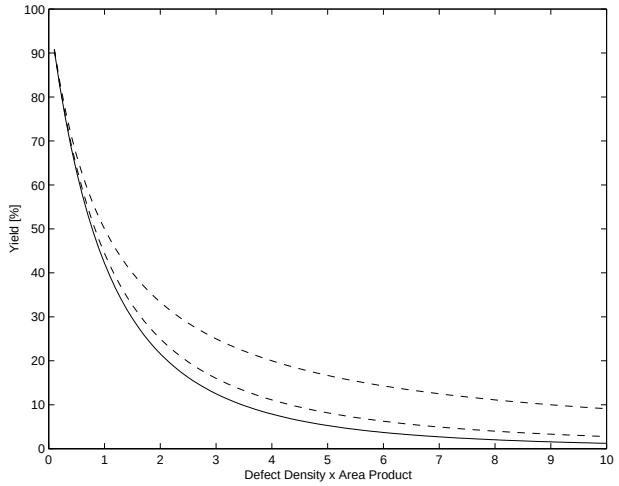


Figure B.1: Yield vs. $D_o A_{die}$ product using equation (B.10) for α=1, 2 & 3.

the die area for a given defect density. This is shown in figure B.1 for several yield expressions. The relation between yield and die area predicted by the expressions for yield is fundamentally true. You pay a big price in yield once the die approaches a certain size. Thus, products which require a larger die can be expected to be more expensive than smaller die.

One aspect not considered is the design itself. The design, transistor geometries, %-area populated with transistors vs. passive components or blank space will differ enough from one design to another and this could also have an impact on the actual yield.

Using equations (B.6) and (B.10) we can find, the total die per wafer, $N_{totaldie}$, the yield, Y, and the net die or good die per wafer, $N_{netdie} = YXN_{totaldie}$.

The estimate of the total and net die per wafer is given in Tables B.1 and B.2 for the a 2.2X2.2 mm die in the AMI 1.2μm and a 1.5x1.5mm die in the AMI 0.5μm process.

Fabrication Process	Wafer Diameter	Dimensions	Area	Die/Wafer
	$[mm]$	$[mm]$	$[mm^2]$	[Total]
1.20 μm	125	2.2 x 2.2	4.84	2,409
0.50 μm	200	1.5 x 1.5	2.25	13,666

Table B.1: Estimate of the total die per wafer.

Notice that the wafer diameter for the 0.5μm process (and smaller die size) provides 5 times as many working die when compared to the 1.2μm process despite a higher defect density (also note these defect densities are arbitrarily

Fabrication Process	Die/Wafer [Total]	Defect Density $[/mm^2]$	Yield Estimate [%]	Net Die Per Wafer
1.20 μm	2,409	0.01	95.31	2,296
0.50 μm	13,666	0.07	86.69	11,847

Table B.2: Estimate of the net die per wafer.

selected and are different from the vendor's true defect density).

The best estimates of the yield are found by producing a large quantity of prototypes; on the order of 500-1,000 die. The yield estimates in Table B.2 were based on an estimated (in this case a fictitious) defect density. Although the results obtained from equation (B.10) are reasonably accurate, measurement of a larger quantity is more precise. It is possible that there are reliability problems due to design practices, etc., which might need to be addressed in the pre-production stage.

B.4 UPC Estimates Based on the Die Only

The UPC on the die level can be estimated for a wafer lot and these results can be extrapolated to cover the desired number of die required. The number of die per wafer are provided from the estimates given in Tables B.1 and B.2.

The cost per die must be computed by dividing the cost per wafer lot by the number of die per wafer lot. The number of die per wafer lot is found by multiplying the net die per wafer by the number of wafers per wafer lot. The Mask set is a paid only for the production of the first wafer lot. The Mask set is a one time fee and is thus considered to be part of the NRE. These results are summarized in Table B.3.

Fabrication Process	Wafer Lot Price [$]	Wafers Per Lot	Total Die	Cost/Die [$]	Mask NRE [$]
1.20 μm	31,000	5	11,480	2.70	52,000
0.50 μm	30,000	5	59,235	0.51	58,000

Table B.3: Estimated cost per die for 'tiny chips" in the 1.20 μm process and the 0.50 μm process.

If one were to use only these values, e.g. the cost/die and the Mask NRE, can be substituted $UnitCost$ and FC, respectively in the expressions derived in section 2. Notice that although there is a very low cost per die that there is a relatively large investment required. The NRE here considers only the Mask Set cost. This is probably a relatively small cost compared to other additional costs such as prototype fabrication/testing, labor for engineers and technicians, marketing, etc. The variable costs, that is the cost/die is not complete due to

the variable cost and yield for the packaging. Although there are many details to include, this still provides a simple example which illustrates how one goes about finding such information. It also illustrates how many die one must sell in order to recover only the mask costs, assuming a contribution of $0.20 per die.

A good deal of the discussion in the two previous sections has been obtained from R. Jaeger in reference[2]. One of the later chapters contains an excellent discussion about the estimation of yield and other interesting information on economics in this area.

B.5 BEP Analysis for 0.5 μm Die

Break-Even analysis provides reliable results only when all of the costs are considered and factored into the calculations. Tables B.4 and B.5 provide additional information about how the non-recurring and per-unit costs are found. The labor is a significant NRE cost driver; this is typical in most budgets. There are other costs which fall under NRE such as board fabrication, prototype IC fabrication, the mask set, the set up costs associated with automatic testing, marketing and other smaller costs.

Cost Line Item	Cost p.Week [$]	Num. of Weeks	FTE	Total [$]
Labor (IC Engineer)	4,000	32	3	384,000
Labor (Tech. Lev 1)	1,500	24	2	72,000
Board Fabrication				28,000
IC Prototype Fab.				45,000
Mask Set				58,000
Testing				43,000
Marketing				125,000
Other				13,000
Total NRE				768,000

Table B.4: Total fixed/non-recurring engineering (NRE) cost.

The per unit cost breakdown, per-unit contribution and the BEPs are given in Table B.5 as a function of yield over the wafer lot. Since the yield is a statistical estimate it is a good practice to quantify the break even point as a function of yield. We are assuming a selling price of $4.23 per chip (die). The BEP is given in number of die and it is also given in wafer lots. It is useful to express the BEP in terms of wafer lots because this is how the die are actually purchased. Notice that if the yield is 70% the BEP is not reached until the purchase of the 8^{th} wafer lot while a yield of 95% reduces the BEP to the 6^{th} wafer lot purchase. The *risk* can also be quantified as a function of the die yield. It is worth pointing out that there is also a yield for packaging because some

working devices can actually be destroyed. This will further reduce the number of working die and thus raise the BEP. It is also possible that some working die are lost in the testing (depending upon the required production testing.

Yield Est. [%]	Net Die p.Wafer Lot	Price p.Die [$]	Testing p.Die [$]	Pkg. p.Die [$]	Total p.Die [$]	Contr. p.Die [$]	BEP Die [units]	BEP Wafer [Lot]
95	64,914	0.46	0.35	1.00	1.81	2.42	317,638	4.89
90	61,497	0.49	0.35	1.00	1.84	2.39	321,047	5.22
87	59,447	0.51	0.35	1.00	1.86	2.37	323,567	5.46
85	58,081	0.52	0.35	1.00	1.87	2.36	324,945	5.59
80	54,664	0.55	0.35	1.00	1.90	2.33	329,445	6.03
75	51,248	0.59	0.35	1.00	1.94	2.29	334,698	6.54
70	47,831	0.63	0.35	1.00	1.98	2.25	340,910	7.13

Table B.5: Other elements of variable cost per die.

One alternative to purchasing wafer lots is to purchase "real estate" on a *multi-project* wafer. This typically works for part volumes of 1,000 and under. The advantage of this approach is that NRE costs such as the mask set are no longer necessary and the overall investment is reduced. The disadvantage is that the per-unit cost is much higher. For the example we have presented a reasonable low-quantity NRE estimate is approximately $\$565,000$ and a per-unit cost for this die size is around $100.00 per die, totaling $101.35 when the testing cost is added. This means that the selling price would need to be over $\$1,000.00$ to bring the BEP below 500 die; it is desirable to break-even during the first fab since the risks are higher (due to a much smaller market) using the *low-volume* alternative.

B.6 Summary

The economics, the market and the profit margin, drive the decision to proceed or cancel the development of a particular integrated circuit. It is obvious that it is not prudent to develop a full custom integrated circuit for an application which requires quantities of 1-10. In fact, the analysis presented overwhelmingly favors high volume production; the larger the demand will be in wafer lots, the better the return on the investment. The time to get a particular product into the market will perturbate this result due to the short product life cycles(e.g. the market price is not time-invariant). This means that most companies doing high volume integrated circuit development will NOT deploy design engineers to projects which are not guaranteed to lead to high volume sales. There is a lower volume market which has evolved for quantities as low as 500-1,000, for customers with applications which do not favor semi-custom or programmable logic devices such as field programmable gate arrays (FPGA)s; however, the return on the investment will be significantly lower. The market exists because

there is a sufficient number of customers who are willing to pay the NRE and there are less expensive alternatives than deep sub-micron technology lines. Due to the higher mask set costs, the most advanced integrated circuit technology, 90-130 nm, only makes economic sense for high volume products starting with commercial memories, microprocessors and FPGAs. The low volume customers will use 250 nm up to $2\mu m$ technologies, more mature but not obsolete, to offset the cost of the most advanced devices.

References

[1] J. M. Rabaey, A. Chandrakasan, and B. Nikolic, *Digital Integrated Circuts*. Prentice Hall, 2^{nd} ed., 2003.

[2] R. C. Jaeger, *Introduction to Microelectronic Fabrication*, vol. V of *Modular Series on Solid-State Devices*. Addison Wesly, 1988.

[3] B. T. Murphy, "Cost-size optima of monolithic integrated circuits," *Proceedings of the IEEE*, vol. 52, pp. 1537–1545, December 1964.

[4] C. H. Strapper, "On yield, fault distributions and clustering of particles," *IBM Journal of Research & Development*, vol. 30, pp. 326–338, May 1986.

Appendix C

Projects

C.1 Estimating Die Sizes for Class Projects

You will perform a similar analysis for your project for the AMI 1.2 μm process. Instead of the die dimensions listed here you will compute the actual dimensions of your project, L and W.

The problem you will have is that the pad frame you are using is not properly sized and is most likely too big. Typcially a project has a rectangular shape which can be found (excluding the buffers for your project) using the magic box command. You will then need to estimate what the rectangular dimensions of your project would be if you were to make a custom pad frame and supply buffers. This is illustrated in figure C.1

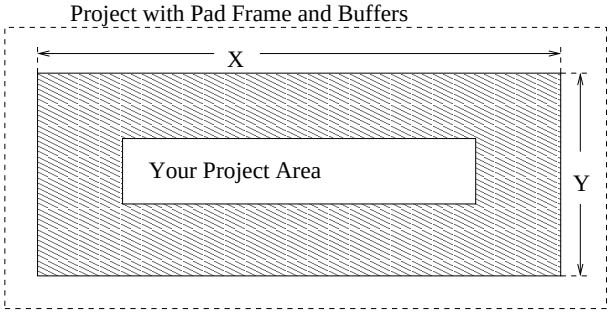


Figure C.1: Estimate of project size.

You will then add a ΔL and a ΔW of 484 λ to account for the pads and buffers in a final product. A spreadsheet in MS excel and Star Office have been provided. This should dramatically reduce your work in estimating the cost of your project.