NPR COLLEGE OF ENGINEERING AND TECHNOLOGY.

EC1354

VLSI DESIGN

DEPT/ YEAR/ SEM: ECE/ III/ VI

PREPARED BY: Ms. S. THENMOZHI/ Lecturer/ECE

SYLLABUS

EC1354 – VLSI DESIGN

UNIT I MOS TRANSISTOR THEORY AND PROCESS TECHNOLOGY

NMOS and PMOS transistors – Threshold voltage – Body effect – Design equations–Second order effects – MOS models and small signal AC characteristics – Basic CMOSTechnology

UNIT II INVERTERS AND LOGIC GATES

NMOS and CMOS inverters – Stick diagram – Inverter ratio – DC and transientcharacteristics – Switching times – Super buffers – Driving large capacitance loads –CMOS logic structures – Transmission gates – Static CMOS design – Dynamic CMOSdesign

UNIT III CIRCUIT CHARACTERISATION AND PERFORMANCE ESTIMATION

Resistance estimation – Capacitance estimation – Inductance – Switching characteristics– Transistor sizing – Power dissipation and design margining – Charge sharing – Scaling

UNIT IV VLSI SYSTEM COMPONENTS CIRCUITS AND SYSTEM LEVEL PHYSICAL DESIGN

Multiplexers – Decoders – Comparators – Priority encoders – Shift registers – Arithmeticcircuits – Ripple carry adders – Carry look ahead adders – High-speed adders –Multipliers – Physical design – Delay modeling – Cross talk – Floor planning – Powerdistribution – Clock distribution – Basics of CMOS testing

UNITV VERILOG HARDWARE DESCRIPTION LANGUAGE

Introduction to FPGA- Xilinx FPGA,-Xilinx 2000-Xilinx 3000 -Overview of digital design with Verilog HDL – Hierarchical modeling concepts– Modules and port definitions – Gate level modeling– Data flow modeling – Behavioral modeling – Task & functions – Test bench

TEXT BOOKS

1. Neil H. E. Weste and Kamran Eshraghian, “Principles of CMOS VLSI Design”,2nd edition, Pearson Education Asia, 2000.
2. John P. Uyemura, “Introduction to VLSI Circuits and Systems”, John Wiley andSons, Inc., 2002.
3. Samir Palnitkar, “Verilog HDL”, 2nd Edition,Pearson Education, 2004.

REFERENCES

1. Eugene D. Fabricius, “Introduction to VLSI Design”, TMH International Editions,1990.
2. Bhasker J., “A Verilog HDL Primer”, 2nd Edition, B. S. Publications, 2001.
3. Pucknell, “Basic VLSI Design”, Prentice Hall of India Publication, 1995.
4. Wayne Wolf, “Modern VLSI Design System on chip”, Pearson Education, 2002.

UNIT I

MOS TRANSISTOR THEORY AND PROCESS TECHNOLOGY

- NMOS transistors.
- PMOS transistors.
- Threshold voltage.
- Body effect.
- Design equations.
- Second order effects.
- MOS models and small signal AC characteristics.
- Basic CMOS Technology.

INTRODUCTION:

VLSI stands for "Very Large Scale Integration". This is the field which involves packing more and more logic devices into smaller and smaller areas. Thanks to VLSI, circuits that would have taken board full of space can now be put into a small space few millimeters across! This has opened up a big opportunity to do things that were not possible before. VLSI circuits are everywhere ... your computer, your car, your brand new state-of-the-art digital camera, the cell-phones, and what have you. All this involves a lot of expertise on many fronts within the same field, which we will look at in later sections.

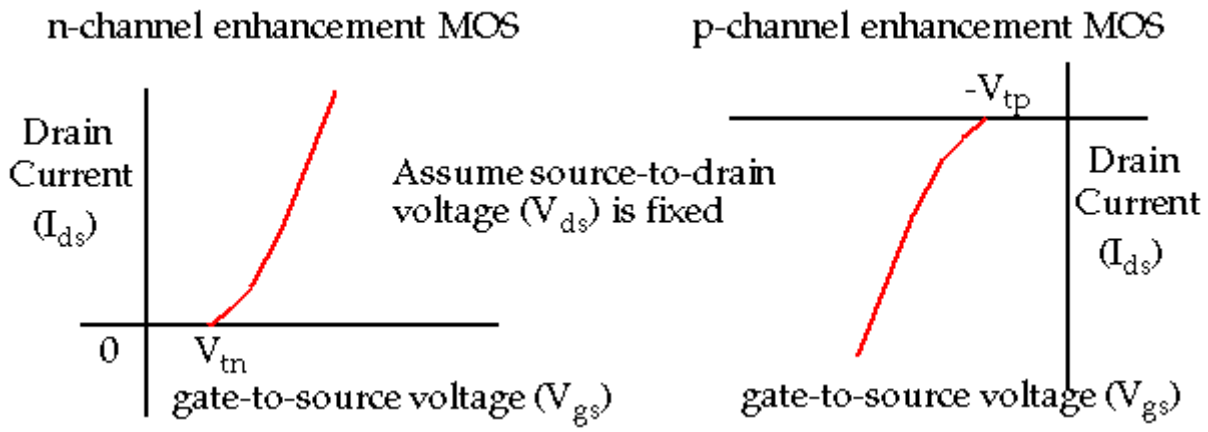
VLSI has been around for a long time, there is nothing new about it ... but as a side effect of advances in the world of computers, there has been a dramatic proliferation of tools that can be used to design VLSI circuits. Alongside, obeying Moore's law, the capability of an IC has increased exponentially over the years, in terms of computation power, utilization of available area, yield. The combined effect of these two advances is that people can now put diverse functionality into the IC's, opening up new frontiers. Examples are embedded systems, where intelligent devices are put inside everyday objects, and ubiquitous computing where small computing devices proliferate to such an extent that even the shoes you wear may actually do something useful like monitoring your heartbeats! These two fields are kind related, and getting into their description can easily lead to another article.

MOS TRANSISTOR DEFINITIONS:

- n-type MOS: Majority carriers are electrons.
- p-type MOS: Majority carriers are holes.

- Positive/negative voltage applied to the gate (with respect to substrate) enhances the number of electrons/holes in the channel and increases conductivity between source and drain.

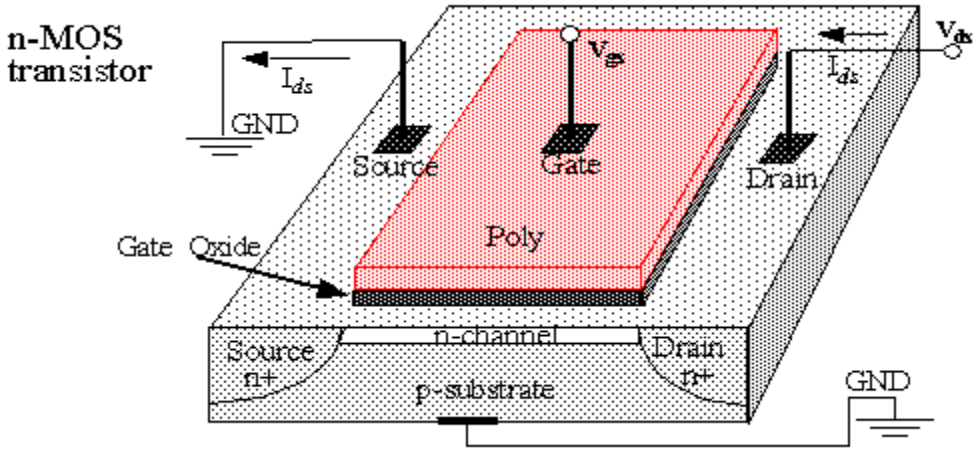
- $V_t$defines the voltage at which a MOS transistor begins to conduct. For voltages less than V_t(threshold voltage), the channel is cut off.



MOS TRANSISTOR DEFINITIONS:

- In normal operation, a positive voltage applied between source and drain (V_{ds}).
- No current flows between source and drain (I_{ds}= 0) with V_{gs}= 0 because of back to back pn junctions.

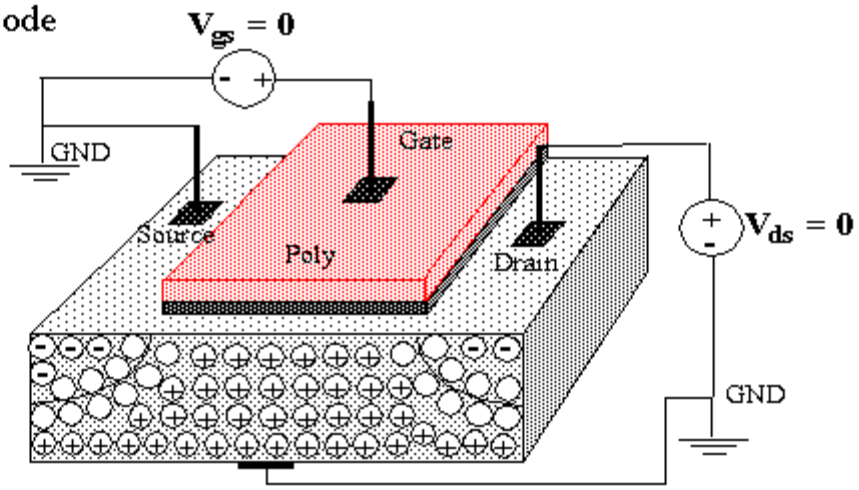
- For n-MOS, with V_{gs}> V_{tn}, electric field attracts electrons creating channel.
- Channel is p-type silicon which is inverted to n-type by the electrons attracted by the electric field.



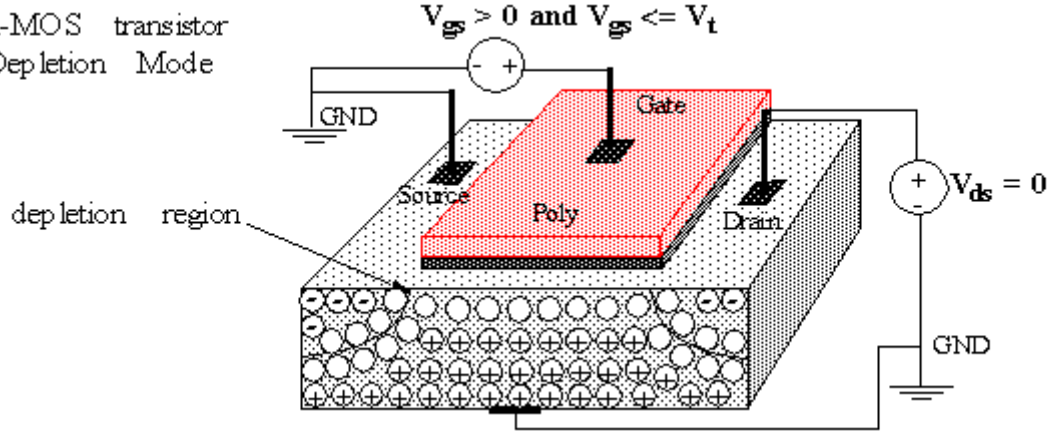
n-MOS ENHANCEMENT TRANSISTOR PHYSICS:

- Three modes based on the magnitude of V_{gs}: accumulation, depletion and inversion.

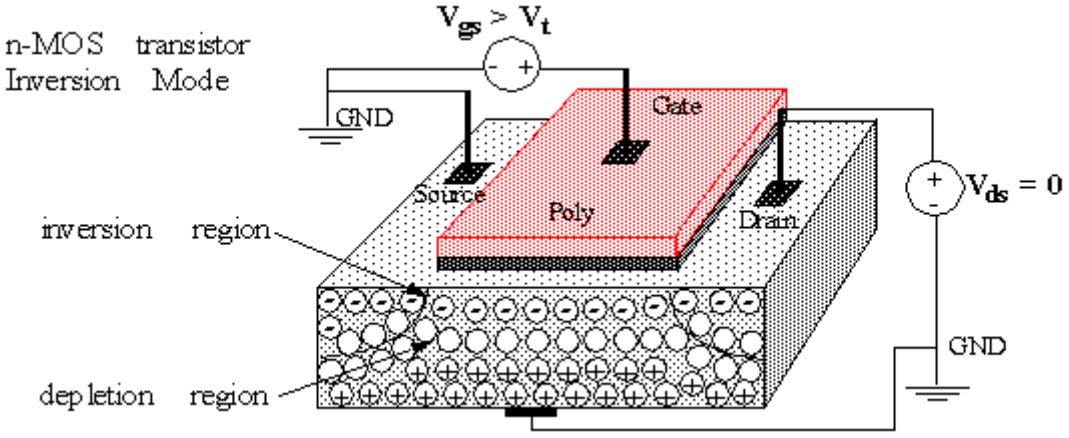
n-MOS transistor
Accumulation Mode



n-MOS transistor
Depletion Mode



n-MOS transistor
Inversion Mode

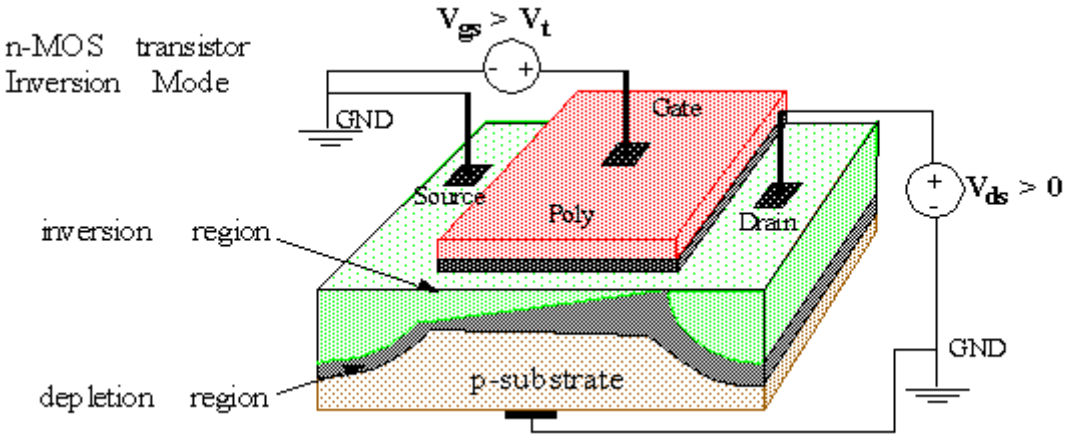


n-MOS ENHANCEMENT TRANSISTOR:

- With V_{ds}non-zero, the channel becomes smaller closer to the drain.

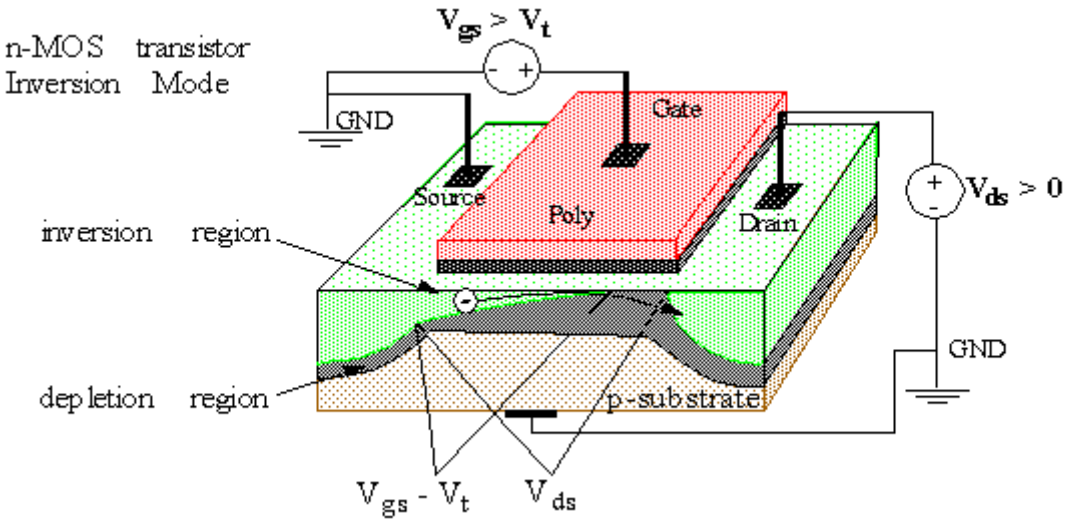
- When $V_{ds} <= V_{gs} - V_t$(e.g. V_{ds}= 3V, V_{gs}= 5V and V_t= 1V), the channel reaches the drain (since $V_{gd} > V_t$).

- This is termed **linear**, **resistive**or **nonsaturated**region. $I_{ds}$is a function of both $V_{gs}$and V_{ds}.



- When $V_{ds} > V_{gs} - V_t$(e.g. V_{ds}= 5V, V_{gs}= 5V and V_t = 1V), the channel is pinchedoff close to the drain (since $V_{gd} < V_t$).

- This is termed saturated region. $I_{ds}$is a function of V_{gs}, almost independent of V_{ds}.



- MOS transistors can be modeled as a voltage controlled switch. $I_{ds}$is an important parameter that determines the behavior, e.g., the speed of the switch.

The parameters that affect the magnitude of I_{ds}.

- The distance between source and drain (channel length).
- The channel width.
- The threshold voltage.
- The thickness of the gate oxide layer.
- The dielectric constant of the gate insulator.
- The carrier (electron or hole) mobility.

SUMMARY OF NORMAL CONDUCTION CHARACTERISTICS:

- Cut-off: accumulation, $I_{ds}$is essentially zero.
- Nonsaturated: weak inversion, I_{ds}dependent on both $V_{gs}$and V_{ds}.
- Saturated: strong inversion, $I_{ds}$is ideally independent of V_{ds}.

THRESHOLD VOLTAGE:

- $V_t$is also an important parameter. What effects its value?

- Most are related to the material properties. In other words, V_t is largely determined at the time of fabrication, rather than by circuit conditions, like I_{ds}.

- For example, material parameters that effect V_t include:

- The gate conductor material (poly vs. metal).
- The gate insulation material (SiO_2).
- The thickness of the gate material.
- The channel doping concentration.

- However, V_t is also dependent on

- V_{sb} (the voltage between source and substrate), which is normally 0 in digital devices.
- Temperature: changes by -2mV/degree C for low substrate doping levels.

- The expression for threshold voltage is given as:

$$V_t = \underbrace{2\phi_b + \frac{\sqrt{2\varepsilon_{Si} q N_A 2\phi_b}}{C_{ox}}}_{\text{Ideal threshold voltage}} + \underbrace{V_{fb}}_{\text{Flat band voltage}} \qquad \text{where} \quad \phi_b = \frac{kT}{q}\ln\left(\frac{N_A}{N_i}\right)$$

Bulk potential

and

N_A: Density of the carriers in the doped semiconductor substrate.

N_i: The carrier concentration of intrinsic (undoped) silicon.

$$N_i = 1.45 \times 10^{10}\, cm^{-3} \text{ (at 300 degrees K)}$$

k: Boltzman's constant. T: temperature. q: electronic charge.

$$\frac{kT}{q} = 25\, mV \text{ (at 300 degrees K)}$$

ε_{Si}: permittivity of silicon

$$\varepsilon_{Si} = 1.06 \times 10^{-12} \text{ Farads/cm}$$

C_{ox}: gate-oxide capacitance.

$$C_{ox} = \frac{\varepsilon_{ox}}{t_{ox}}$$

$$V_t = \underbrace{2\phi_b + \frac{\sqrt{2\varepsilon_{Si} q N_A 2\phi_b}}{C_{ox}}}_{\text{Ideal threshold voltage}} + \underbrace{V_{fb}}_{\text{Flat band voltage}}$$

and

$$V_{fb} = \phi_{ms} - \frac{Q_{fc}}{C_{ox}}$$

where Q_{fc} represents the fixed charge due to imperfections in silicon-oxide interface and doping.

and ϕ_{ms} is work function difference between gate material and silicon substrate ($\phi_{gate} - \phi_{Si}$).

Typical values of V_{fb} for n/p transistor is **-0.9V** (with $N_A = 10^{16}$ cm^{-3}) and **-0.2V**. (See text for examples).

- Typical values of V_t for n and p-channel transistors are +/- 700mV.
- From equations, threshold voltage may be varied by changing:

- The doping concentration (N_A).
- The oxide capacitance (C_{ox}).
- Surface state charge (Q_{fc}).

- Also it is often necessary to adjust V_t.
- Two methods are common:

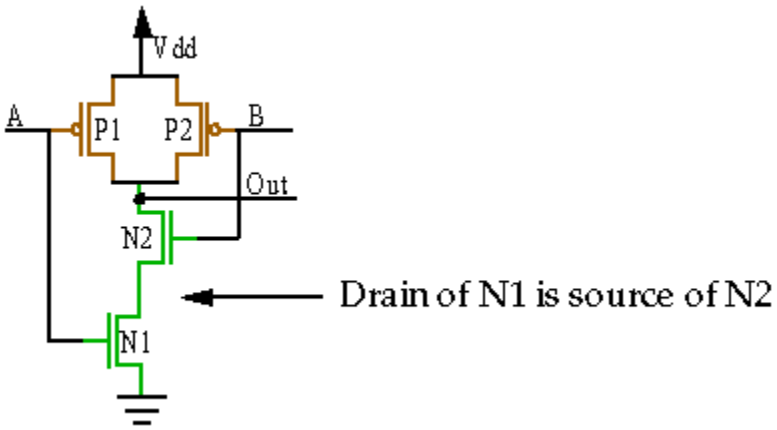
- Change Q_{fc} by introducing a small doped region at the oxide/substrate interface via ion implantation.

- Change C_{ox} by using a different insulating material for the gate.
  - A layer of Si_3N_4 (silicon nitride) with a relative permittivity of 7.5 is combined with a layer of silicon dioxide (relative permittivity of 3.9).
  - This results in a relative permittivity of about 6.

- For the same thickness dielectric layer, C_{ox} is larger using the combined material, which lowers V_t.

BODY EFFECT:

- In digital circuits, the substrate is usually held at zero.
  - The sources of n-channel devices, for example, are also held at zero, except in cases of series connections, e.g.,



- The source-to-substrate (V_{sb}) may increase at this connections, e.g. $V_{sbN1} = 0$ but $V_{sbN2} /= 0$.
- V_{sb} adds to the channel-substrate potential:

$$V_t = 2\phi_b + \frac{\sqrt{2\varepsilon_{Si} q N_A \left| 2\phi_b + V_{sb} \right|}}{C_{ox}} + V_{fb}$$

BASIC DC EQUATIONS:

- Ideal first order equation for cut-offregion:

$$I_{ds} = 0 \quad \text{when} \quad V_{gs} \leq V_t$$

- Ideal first order equation for linearregion:

$$I_{ds} = \beta\left[(V_{gs} - V_t)V_{ds} - \frac{V_{ds}^2}{2}\right] \quad \text{when} \quad 0 < V_{ds} < V_{gs} - V_t$$

- Ideal first order equation for saturationregion:

$$I_{ds} = \beta\frac{(V_{gs} - V_t)^2}{2} \quad \text{when} \quad 0 < V_{gs} - V_t \leq V_{ds}$$

- with the following definitions:

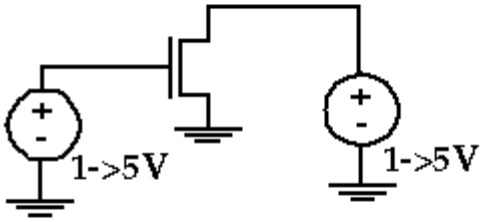
$$\beta = \frac{\mu\varepsilon}{t_{ox}}\left(\frac{W}{L}\right)$$

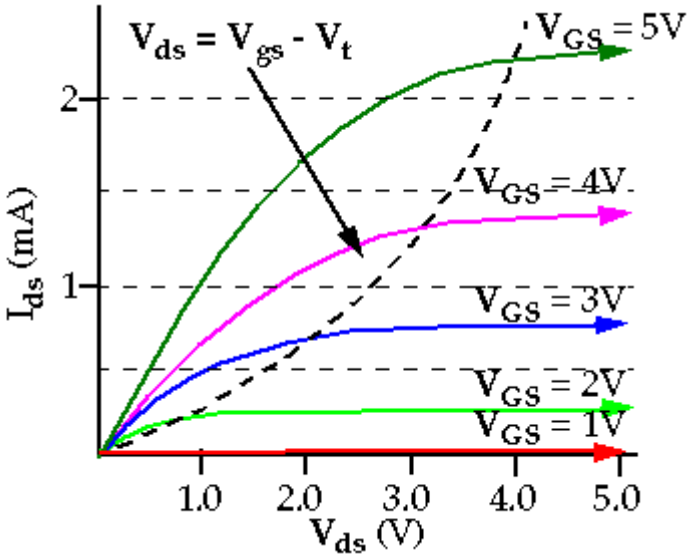
μ = surface mobility of the carriers.
ε = permittivity of the gate insulator.
t_{ox} = thickness of the gate insulator.
W and L are the width and length of channel.

- Process dependent factors: $\frac{\mu\varepsilon}{t_{ox}}$ or μC_{ox} where $C_{ox} = \frac{\varepsilon}{t_{ox}}$.

- Geometry dependent factors: W and L.

- Voltage-current characteristics of the n- and p-transistors.





Beta calculation

- Transistor beta calculation example:
  - Typical values for an n-transistor in 1 micron technology:

$$\mu_n = 500\,\text{cm}^2/\text{V-sec}$$
$$\varepsilon = 3.9\,\varepsilon_0 = 3.9 \times 8.85 \times 10^{-14}\ \text{F/cm (permittivity of silicon dioxide)}$$
$$t_{ox} = 20\,\text{nm}$$

  - Compute beta:

$$\beta_n = \frac{500 \times 3.9 \times 8.85 \times 10^{-14}}{0.2 \times 10^{-5}} \frac{W}{L} = 88.5 \frac{W}{L} \mu A / V^2$$

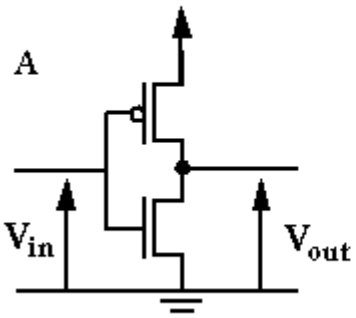
  - How does this beta compare with p-devices:

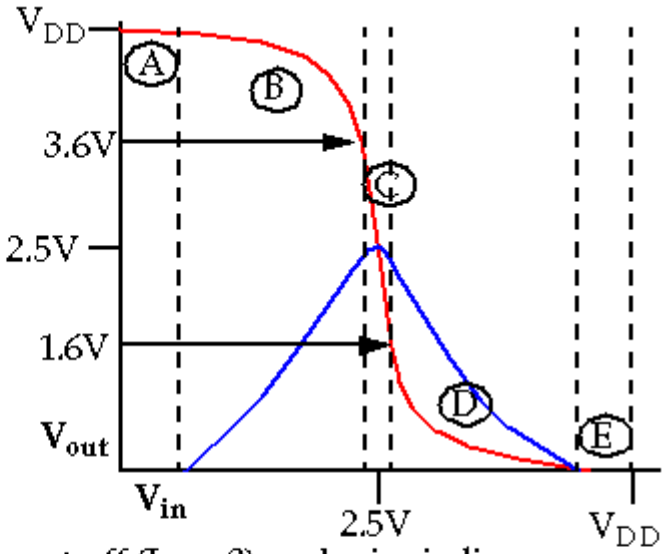
$$\beta_p = \frac{180 \times 3.9 \times 8.85 \times 10^{-14}}{0.2 \times 10^{-5}} \frac{W}{L} = 31.9 \frac{W}{L} \mu A / V^2$$

- n-transistor gains are approximately 2.8 times larger than p-transistors.

Inverter voltage transistor characteristics

- Inverter DC characteristics





Ⓐ $0 \le V_{in} \le V_{tn}$;n-device is cut off (I_{dsn}=0), p-device in linear.

Ⓑ $V_{tn} < V_{in} < V_{DD}/2 - \Delta$;n-device is in sat., p-device in linear.

Ⓒ $V_{DD}/2 - \Delta \le V_{in} \le V_{DD}/2 + \Delta$;n-device is in sat., p-device in sat.

Ⓓ $V_{DD}/2 + \Delta < V_{in} < V_{DD} + V_{tp}$;n-device is in linear, p-device in sat.

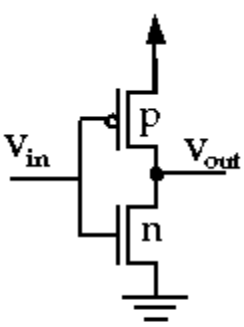
Ⓔ $V_{DD} + V_{tp} \le V_{in} \le V_{DD}$;n-device is in linear, p-device in cut off (I_{dsp}=0).

Beta Ratios

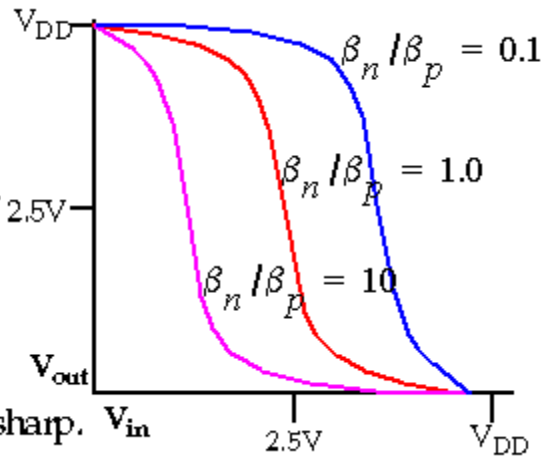
- Region C is the most important region. A small change in the input voltage, V_{in}, results in a LARGE change in the output voltage, V_{out}.

- This behavior describes an amplifier, the input is amplified at the output. The amplification is termed transistor gain, which is given by beta.

- Both the n and p-channel transistors have a beta. Varying their ratio will change the characteristics of the output curve.



Beta ratio of n and p-channel transistors varied over two orders of magnitude.



As ratio is decreased, curve shifts to the right, but the output transition remains sharp.

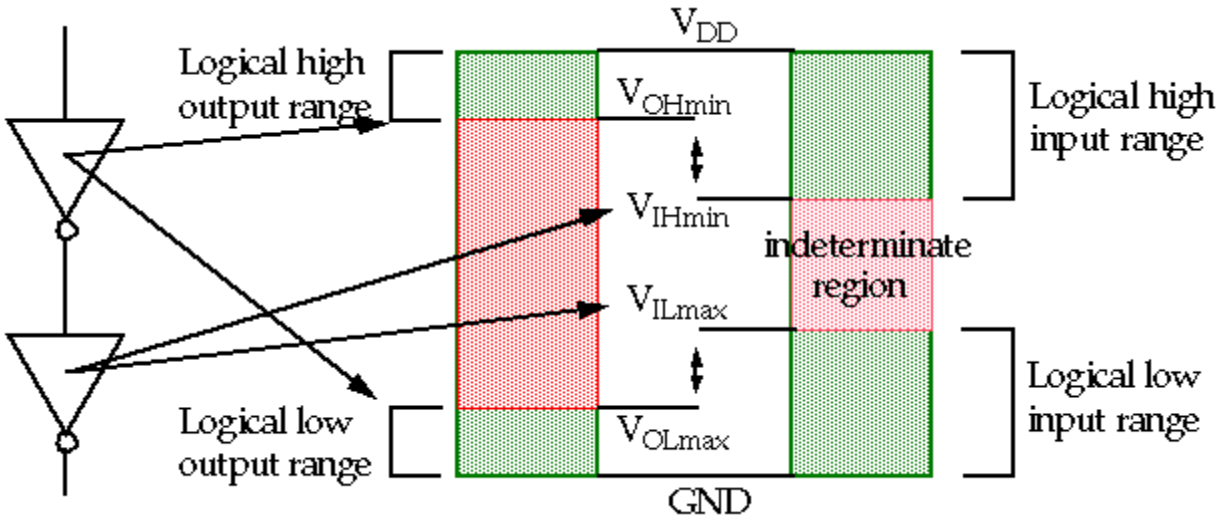
- Therefore, the
  - $\frac{\beta_n}{\beta_p}$
- does not affect switching performance.
- What factor would argue for a ratio of 1 for $\frac{\beta_n}{\beta_p}$?
  - Load capacitance !
- The time required to charge or discharge a capacitive load is equal when $\frac{\beta_n}{\beta_p} = 1$.
- Since beta is dependent W and L, we can adjust the ratio by changing the sizes of the transistor channel widths, by making p-channel transistors **wider** than n-channel transistors.

NOISE MARGINS:

- A parameter that determines the maximum **noise**voltage on the input of a gate that allows the output to remain stable.

- Two parameters, Low noise margin (NM_L) and High noise margin (NM_H).

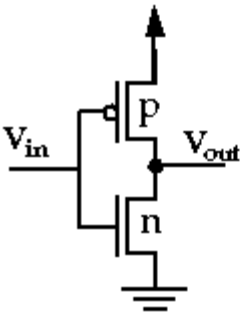
- NM_L= difference in magnitude between the max LOW output voltage of the driving gate and max LOW input voltage recognized by the driven gate.



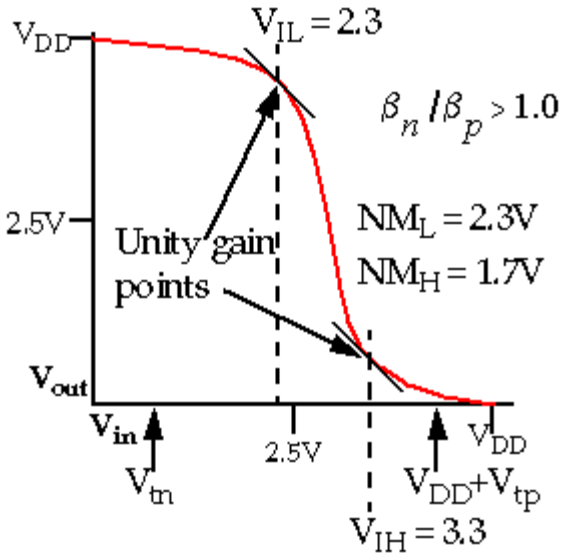
- Ideal characteristic: $V_{IH}= V_{IL}= (V_{OH}+V_{OL})/2$.

- This implies that the transfer characteristic should switch abruptly (high gain in the transition region).

- V_{IL} found by determining unity gain point from V_{OH}.

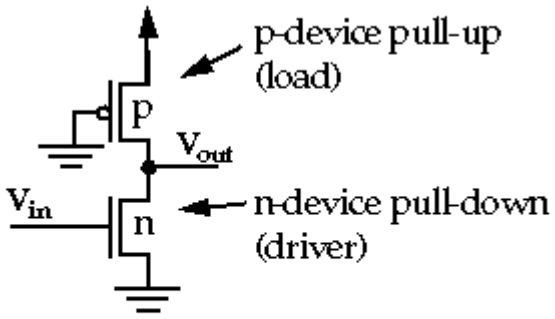


Assume output of driving gate is stable at supply voltage, e.g.,
$V_{OH} = 5V$
$V_{OL} = 0V$

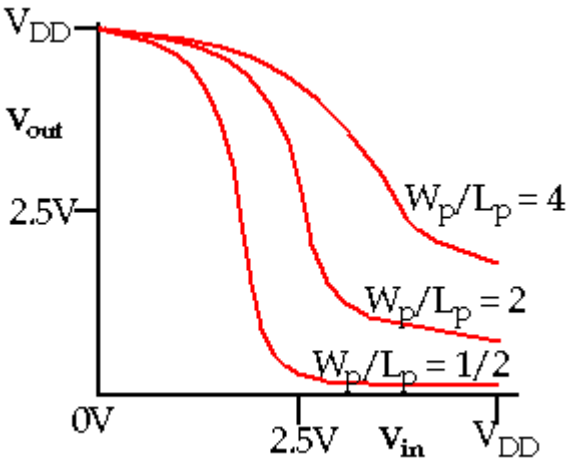


Noise margins are often compromised to improve speed.

Pseudo-nMOS Inverter



When driver is on, steady-state current flows - not a good choice for low-power circuits.

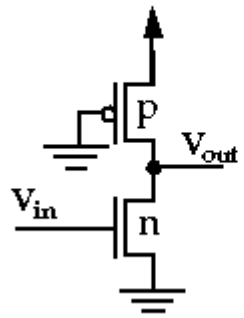


- Therefore, the shape of the transfer characteristic and the V_{OL} of the inverter is affected by the ratio $\frac{\beta_n}{\beta_p}$.
- In general, the low noise margin is considerably worse than the high noise margin for Pseudo-nMOS.

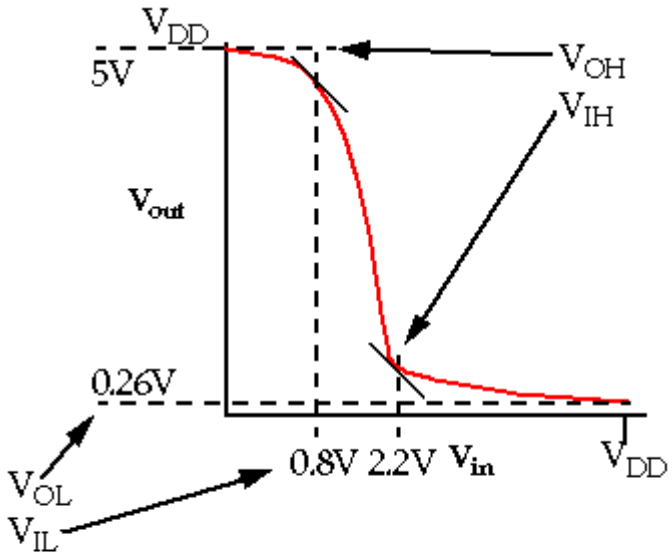
- Pseudo-nMOS was popular for high-speed circuits, static ROMs and PLAs.

Pseudo-nMOS

- Example: Calculation of noise margins:



Pseudo-nMOS inverter



$NM_H = V_{OH} - V_{IH} = 5V - 2.2V = 2.8V$

$NM_L = V_{IL} - V_{OL} = 0.8V - 0.26V = 0.54V$ (This is quite a bit worse than NM_H)

- The transfer curve for the pseudo-nMOS inverter can be used to calculate the noise margins of identical pseudo-nMOS inverters

MOSFET Small Signal Model and Analysis:

•Just as we did with the BJT, we can consider theMOSFET amplifier analysis in two parts:

•Find the DC operating point

•Thendetermine theamplifier outputparametersforvery small input signals.

IMPORTANT QUESTIONS

PART A

1. What are the four generations of Integration circuits?
2. Mention MOS transistor characteristics?
3. Compare pMOS & nMOS.
4. What is threshold voltage?
5. What are the different operating modes of MOS transistor?
6. What is accumulation mode?
7. What is depletion mode?
8. What is inversion mode?
9. What are the three operating regions of MOS transistor?
10. What are the parameters that affect the magnitude of drain source current?
11. Write the threshold voltage equation for nMOS transistor.
12. What are the functional parameters of threshold equation?
13. How the threshold voltage can be varied?
14. What are the common methods used to adjust threshold voltage?
15. Write the threshold voltage equation for pMOS transistor.
16. What is body effect?
17. Write the MOS DC equation.
18. What are the second order effects of MOS transistor?
19. What is sub threshold current?
20. What is channel length modulation?
21. What is mobility variation?
22. What is drain punch through?
23. What is impact ionization?
24. What is cut off region?
25. What is non linear region?
26. What is CMOS technology?
27. What are the advantages of CMOS over nMOS technology?
28. What are the advantages of CMOS technology?

29. What are the disadvantages of CMOS technology?
30. What are the four main CMOS technologies?
31. What are the advantages of n-well process?
32. What are the disadvantages of n-well process?
33. What is twin tub process?
34. What is the process flow of twin tub method?
35. What are the advantages of twin tub process?
36. What are the disadvantages of twin tub process?
37. What is SOI?
38. What are the advantages of SOI process?
39. What are the disadvantages of SOI process?
40. What is the various etching processes used in SOI process?

PART B

1. Explain the operation of nMOS enhancement transistor.
2. Explain the operation of pMOS enhancement transistor.
3. Derive the threshold voltage equation of nMOS transistor with and without body effect.
4. Derive the threshold voltage equation of pMOS transistor with and without body effect.
5. Derive the MOS DC equations of an nMOS.
6. Derive the equation for the threshold voltage of a MOS transistor and threshold voltage in terms of flat band voltage.
7. Derive expressions for the drain to source current in the non-saturated and saturated regions of operation of an nMOS transistor.
8. Explain the second order effects of MOSFET.
9. Discuss the small signal model of an nMOS transistor.
10. What is channel length modulation & body effect? Explain how body effect affects the threshold voltage.
11. What are the various features of CMOS technology?
12. Explain the n-well CMOS process in detail.
13. Explain the p-well CMOS process in detail.
14. Explain the twin tub process in detail.
15. Explain the SOI process in detail.

UNIT II

INVERTERS AND LOGIC GATES

- NMOS and CMOS inverters.
- Stick diagram.
- Inverter ratio.
- DC and transient characteristics.
- Switching times.
- Super buffers.
- Driving large capacitance loads.
- CMOS logic structures.
- Transmission gates.
- Static CMOS design.
- Dynamic CMOS design.

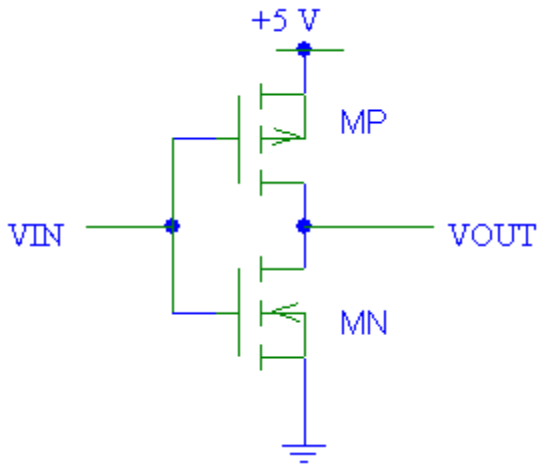
INTRODUCTION

CMOS inverters (Complementary NOSFET Inverters) are some of the most widely used and adaptable MOSFET inverters used in chip design. They operate with very little power loss and at relatively high speed. Furthermore, the CMOS inverter has good logic buffer characteristics, in that, its noise margins in both low and high states are large.

This short description of CMOS inverters gives a basic understanding of the how a CMOS inverter works. It will cover input/output characteristics, MOSFET states at different input voltages, and power losses due to electrical current.

CMOS INVERTER:

A CMOS inverter contains a PMOS and a NMOS transistor connected at the drain and gate terminals, a supply voltage VDD at the PMOS source terminal, and a ground connected at the NMOS source terminal, were VIN is connected to the gate terminals and VOUT is connected to the drain terminals.(See diagram). It is important to notice that the CMOS does not contain any resistors, which makes it more power efficient that a regular resistor-MOSFET inverter. As the voltage at the input of the CMOS device varies between 0 and 5 volts, the state of the NMOS and PMOS varies accordingly. If we model each transistor as a simple switch activated by VIN, the inverter's operations can be seen very easily:



Transistor "switch model"

The switch model of the MOSFET transistor is defined as follows:

MOSFET	Condition on MOSFET	State of MOSFET
NMOS	Vgs<Vtn	OFF

NMOS	Vgs>Vtn	ON
PMOS	Vsg<Vtp	OFF
PMOS	Vsg>Vtp	ON

When VIN is low, the NMOS is "off", while the PMOS stays "on": instantly charging VOUT to logic high. When Vin is high, the NMOS is "on and the PMOS is "on: draining the voltage at VOUT to logic low.

This model of the CMOS inverter helps to describe the inverter conceptually, but does not accurately describe the voltage transfer characteristics to any extent. A more full description employs more calculations and more device states.

Multiple state transistor model

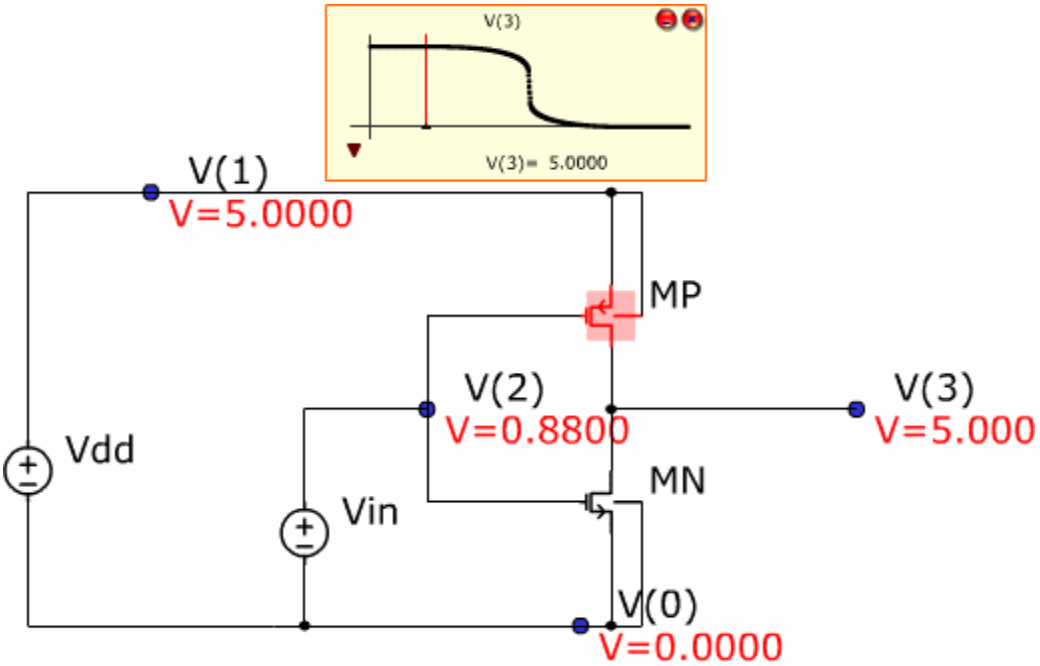
The multiple state transistor model is a very accurate way to model the CMOS inverter. It reduces the states of the MOSFET into three modes of operation: Cut-Off, Linear, and Saturated: each of which have a different dependence on Vgs and Vds. The formulas which govern the state and the current in that given state is given by the following table:

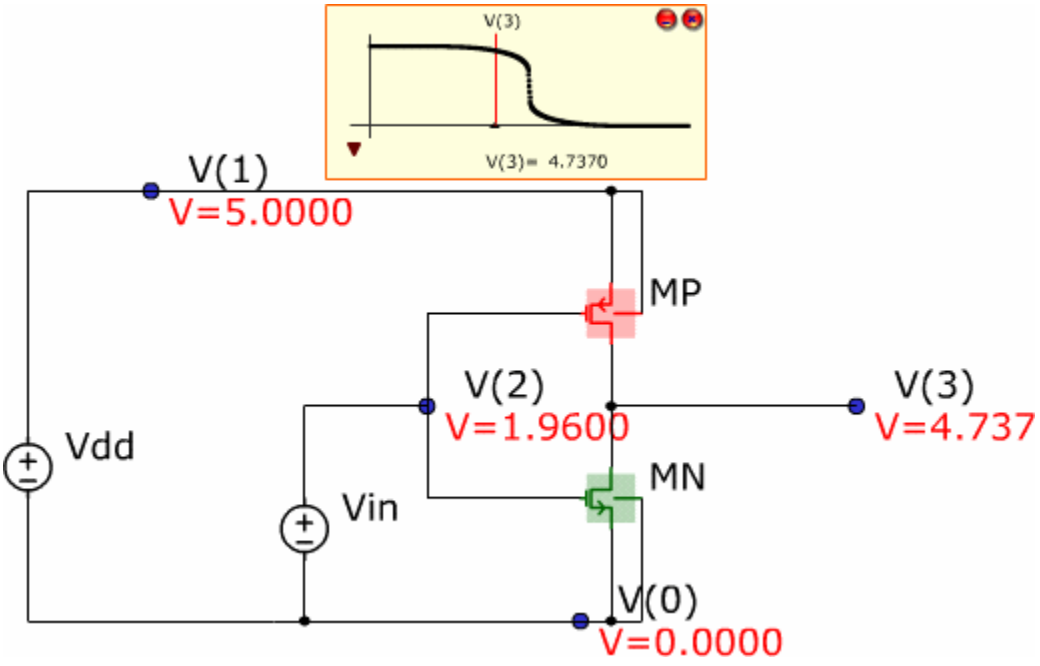
NMOS Characteristics			
	Condition on V_{GS}	Condition on V_{DS}	Mode of Operation
$I_D = 0$	$V_{GS}<V_{TN}$	All	Cut-off
$I_D = k_N [2(V_{GS} - V_{TN}) V_{DS} - V_{DS}^2]$	$V_{GS}>V_{TN}$	$V_{DS}<V_{GS} -V_{TN}$	Linear
$I_D = k_N (V_{GS} - V_{TN})^2$	$V_{GS}>V_{TN}$	$V_{DS} >V_{GS} -V_{TN}$	Saturated
PMOS Characteristics			
	Condition on V_{SG}	Condition on V_{SD}	Mode of Operation
$I_D = 0$	$V_{SG}< -V_{TP}$	All	Cut-off

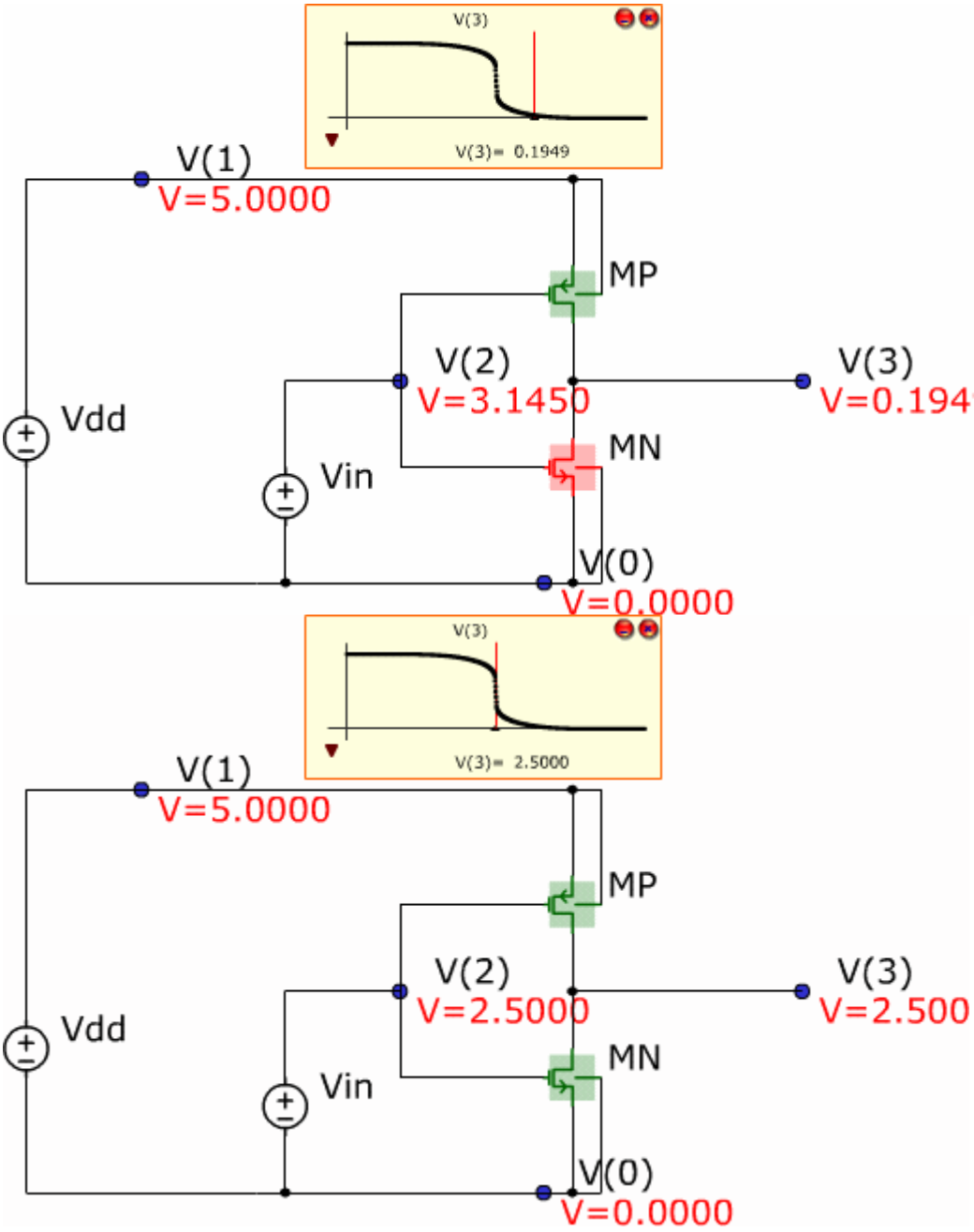
$I_D = k_P [2(V_{SG} + V_{TP}) V_{SD} - V_{SD}^2]$	$V_{SG} > -V_{TP}$	$V_{SD} < V_{SG} + V_{TP}$	Linear
$I_D = k_P (V_{SG} + V_{TP})^2$	$V_{SG} > -V_{TP}$	$V_{SD} > V_{SG} + V_{TP}$	Saturated

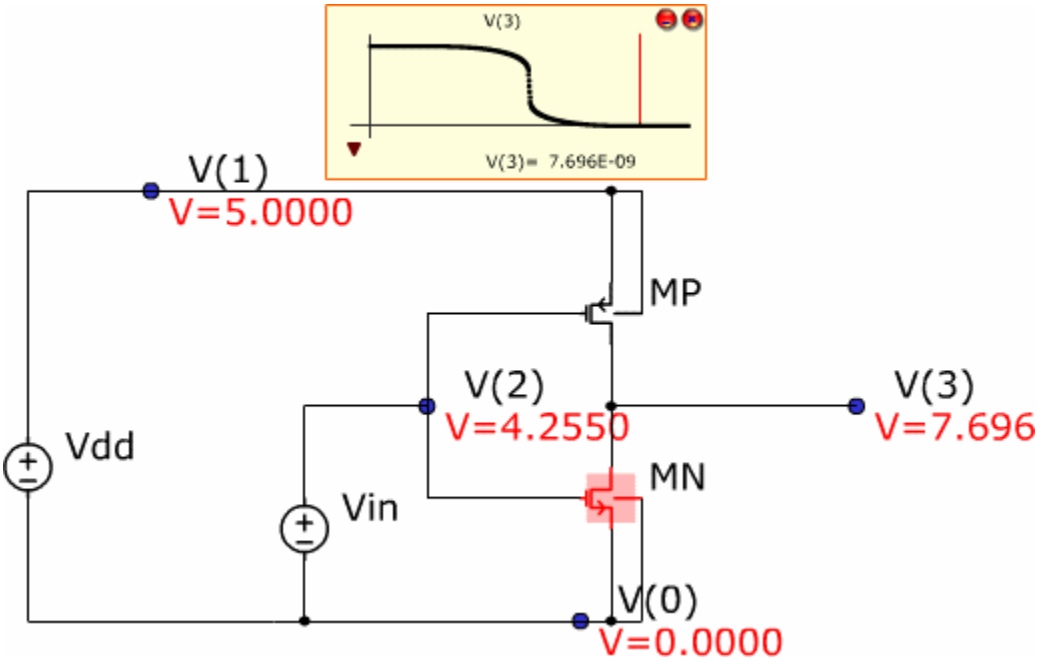
In order to simplify calculations, I have made use of an internet circuit simulation device called "MoHAT." This tool allows the user to simulate circuits containing a few transistors in a simple and visually appealing way. The circuits shown below show the state of each transistor (black for cut-off, red for linear, and green for saturation) accompanied by the voltage transfer characteristic curve (VOUT vs. VIN). The vertical line plotted on the VTC corresponds to the value of VIN on the circuit diagram. The following series of diagrams depict the CMOS inverter in varying input voltages ranging from low to high in ascending order.

	Table of figures	
figure	mode of operation	Logic output level
1	VIN < VIL	High
2	VIN < VIL	High
3	VIL < VIN <VIH	<undetermined>
4	VIN > VIH	Low
5	VIN > VIH	Low


V(3)
V(3)= 5.0000
V(1)
V=5.0000
MP
V(2)
V=0.8800
V(3)
V=5.000
Vdd
MN
Vin
V(0)
V=0.0000

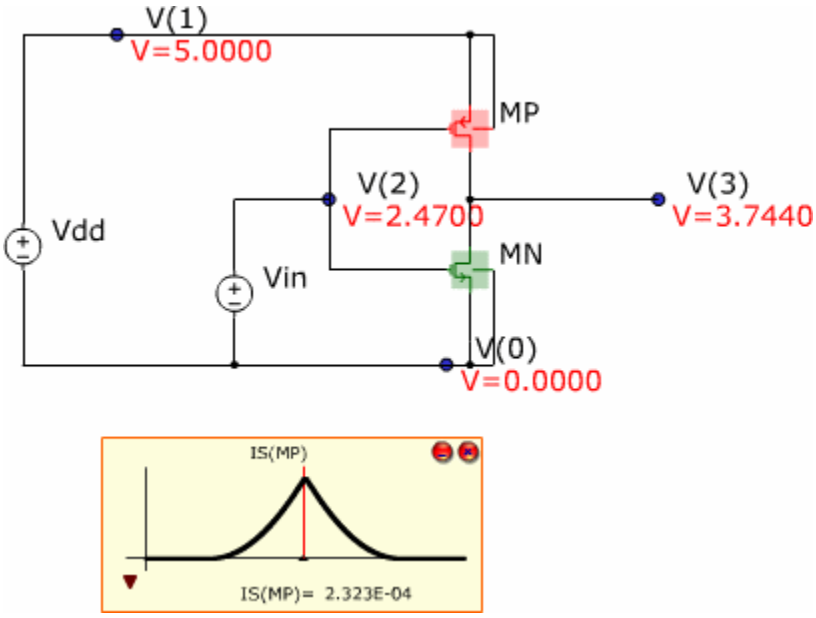

V(3)
V(3)= 4.7370
V(1)
V=5.0000
MP
V(2)
V=1.9600
V(3)
V=4.737
Vdd
MN
Vin
V(0)
V=0.0000


V(3)
V(3)= 0.1949
V(1)
V=5.0000
MP
V(2)
V=3.1450
V(3)
V=0.194
Vdd
MN
Vin
V(0)
V=0.0000
V(3)
V(3)= 2.5000
V(1)
V=5.0000
MP
V(2)
V=2.5000
V(3)
V=2.500
Vdd
MN
Vin
V(0)
V=0.0000



Power dissipation analysis of CMOS inverter

As mentioned before, the CMOS inverter shows very low power dissipation when in proper operation. In fact, the power dissipation is virtually zero when operating close to VOH and VOL. The following graph shows the drain to source current (effectively the overall current of the inverter) of the NMOS as a function of input voltage. Note that the current in the far left and right regions (low and high VIN respectively) have low current, and the peak current in the middle is only .232mA (a 1.16mW power dissipation).



Conclusion

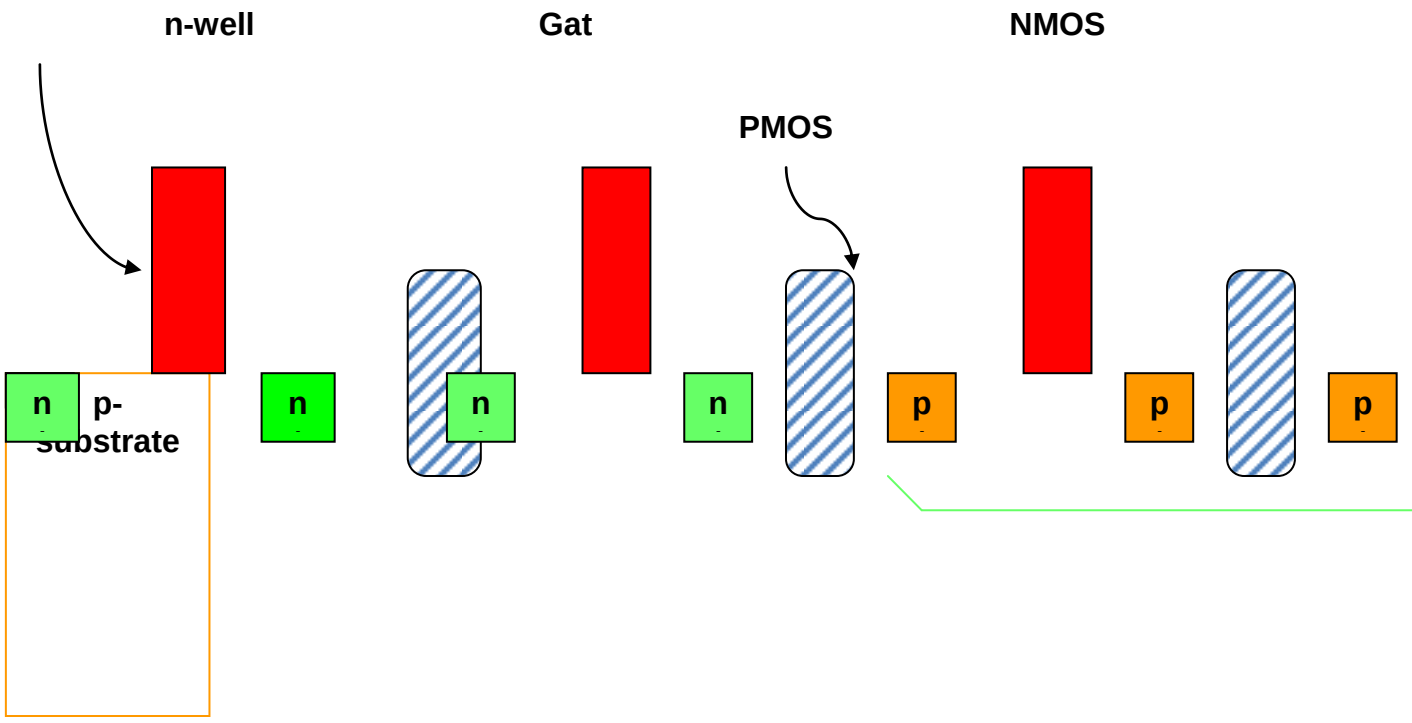
The CMOS inverter is an important circuit device that provides quick transition time, high buffer margins, and low power dissipation: all three of these are desired qualities in inverters for most circuit design. It is quite clear why this inverter has become as popular as it is.

STICK DIAGRAM:

CMOS Layers

- n-well process
- p-well process
- Twin-tub process

n-well process

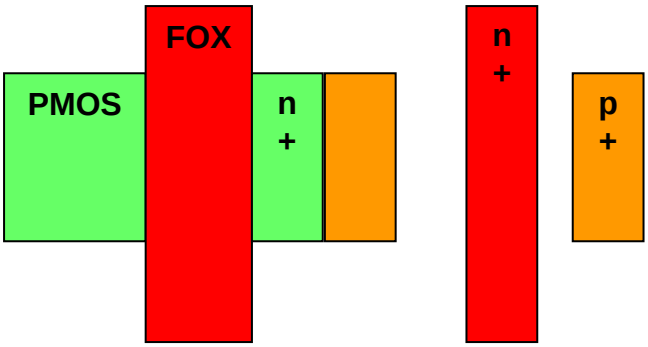


MOSFET Layers in an n-well process

LAYER TYPES:

- **p-substrate**
- **n-well**
- **n+**
- **p+**
- **Gate oxide**
- **Gate (polysilicon)**
- **Field Oxide**
    1. **Insulated glass.**
    2. **Provided electrical isolation.**

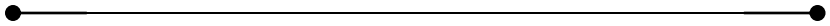
TOP VIEW OF THE FET PATTERN:



NMOS PMOS

Metal Interconnect Layers:

- Metal layers are electrically isolated from each other
- Electrical contact between adjacent conducting layers requires contact cuts and vias



Basic Gate Design:

- Both the power supply and ground are routed using the Metal layer
- n+ and p+ regions are denoted using the same fill pattern. The only difference is the n-well
- Contacts are needed from Metal to n+ or p+

Stick Diagrams:

- Cartoon of a layout.
- Shows all components.
- Does not show exact placement, transistor sizes,wire lengths, wire widths, boundaries, or any other form of compliance with layout or design rules.
- Useful for interconnect visualization, preliminary layoutlayout compaction, power/ground routing, etc.

Points to Ponder:

- be creative with layouts
- sketch designs first
- minimize junctions but avoid long poly runs
- have a floor plan for input, output, power and ground locations

DC & TRANCIENT CHARACTERISTICS:

DC Response:

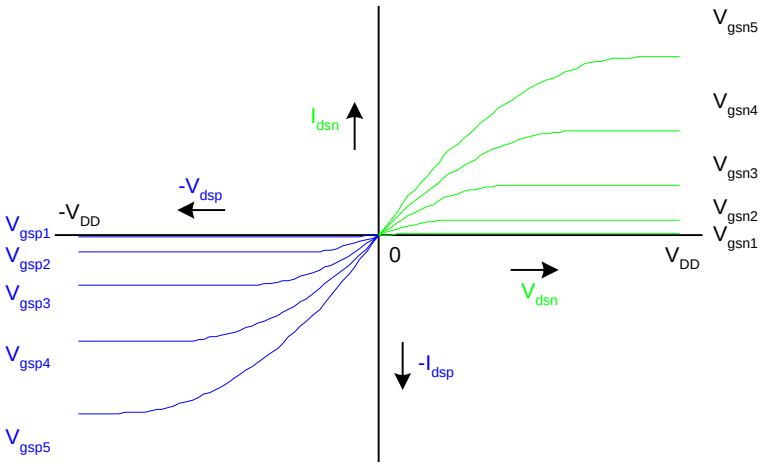
- DC Response: V_{out} vs. V_{in} for a gate
- Ex: Inverter
  - When $V_{in} = 0$ -> $V_{out} = V_{DD}$
  - When $V_{in} = V_{DD}$ -> $V_{out} = 0$
  - In between, V_{out} depends on
  - transistor size and current
  - By KCL, must settle such that
  - $I_{dsn} = |I_{dsp}|$

- We could solve equations
- But graphical solution gives more insight

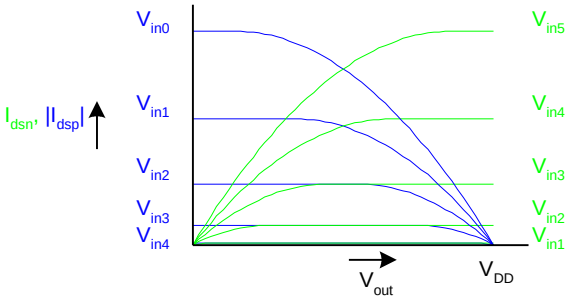
TRANSISTOR OPERATION:

- Current depends on region of transistor behavior
- For what V_{in} and V_{out} are nMOS and pMOS in
  - Cutoff?
  - Linear?
  - Saturation?

I-V Characteristics:



Current vs. V_{out}, V_{in}:

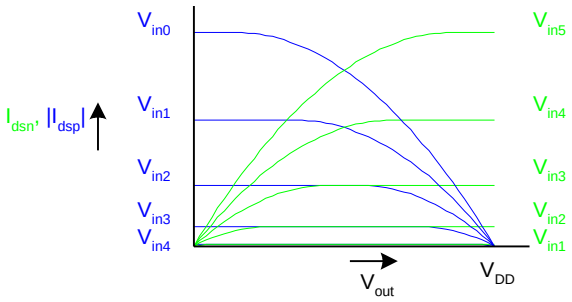


Load Line Analysis:

For a given V_{in}:

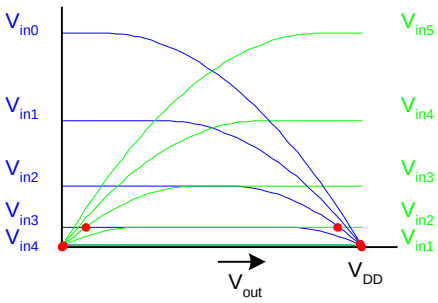
Plot I_{dsn}, I_{dsp} vs. V_{out}

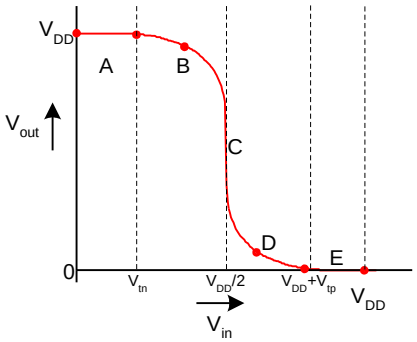
V_{out} must be where |currents| are equal in



DC Transfer Curve:

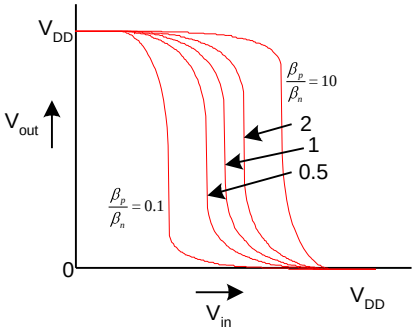
Transcribe points onto V_{in} vs. V_{out} plot



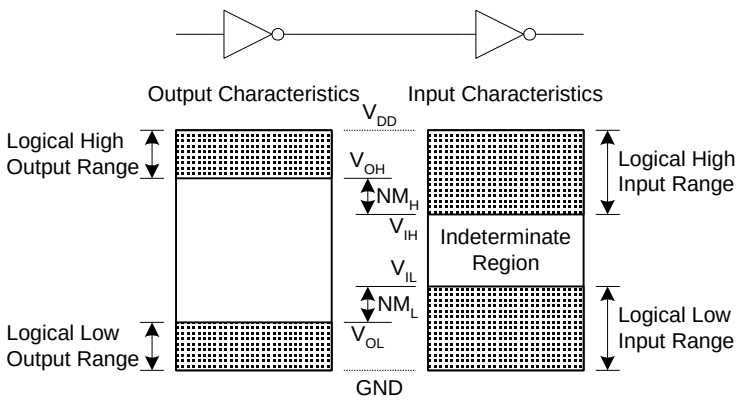


Beta Ratio:

- If $b_p / b_n \neq 1$, switching point will move from $V_{DD}/2$
- Called *skewed* gate
- Other gates: collapse into equivalent inverter

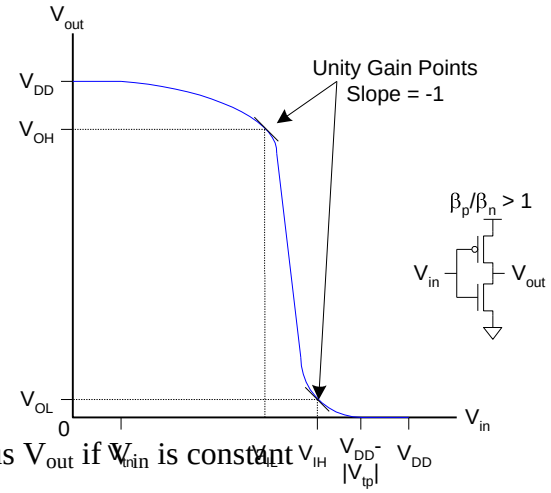


Noise Margins:



Logic Levels:

- To maximize noise margins, select logic levels at
  - unity gain point of DC transfer characteristic



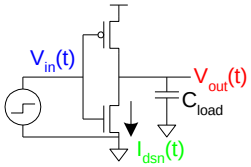
Transient Response:

- DC analysis tells us V_{out} if V_{in} is constant
- Transient analysis tells us $V_{out}(t)$ if $V_{in}(t)$ changes
  - Requires solving differential equations
- Input is usually considered to be a step or ramp
  - From 0 to V_{DD} or vice versa

Inverter Step Response:

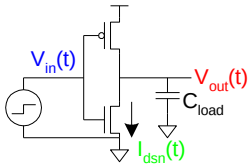
- Ex: find step response of inverter driving load cap

$$V_{in}(t) = u(t - t_0)V_{DD}$$
$$V_{out}(t < t_0) = V_{DD}$$
$$\frac{dV_{out}(t)}{dt} = -\frac{I_{dsn}(t)}{C_{load}}$$

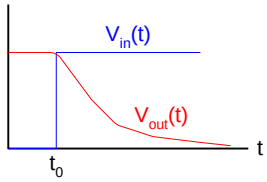


Inverter Step Response:

$$V_{in}(t) = u(t - t_0)V_{DD}$$
$$V_{out}(t < t_0) = V_{DD}$$
$$\frac{dV_{out}(t)}{dt} = -\frac{I_{dsn}(t)}{C_{load}}$$



$$I_{dsn}(t) = \begin{cases} 0 & t \le t_0 \\ \frac{\beta}{2}\left(V_{DD} - V_t\right)^2 & V_{out} > V_{DD} - V_t \\ \beta\left(V_{DD} - V_t - V_{out}(t)/2\right)V_{out}(t) & V_{out} < V_{DD} - V_t \end{cases}$$



Delay Definitions:

t_{pdr}: *rising propagation delay*

From input to rising output crossing $V_{DD}/2$

t_{pdf}: *falling propagation delay*

From input to falling output crossing $V_{DD}/2$

t_{pd}: *average propagation delay*

$t_{pd} = (t_{pdr} + t_{pdf})/2$

t_r: *rise time*

From output crossing 0.2 V_{DD} to 0.8 V_{DD}

t_f: *fall time*

From output crossing 0.8 V_{DD} to 0.2 V_{DD}

t_{cdr}: *rising contamination delay*

From input to rising output crossing $V_{DD}/2$

t_{cdf}: *falling contamination delay*

From input to falling output crossing $V_{DD}/2$

t_{cd}: *average contamination delay*

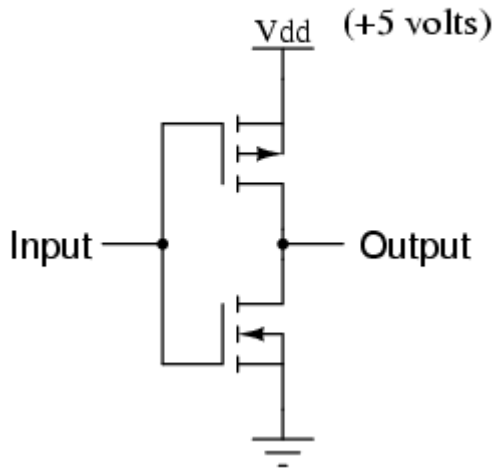
$t_{pd} = (t_{cdr} + t_{cdf})/2$

CMOSgate circuitry

Up until this point, our analysis of transistor logic circuits has been limited to the TTL design paradigm, whereby bipolar transistors are used, and the general strategy of floating inputs being equivalent to "high" (connected to V_{cc}) inputs -- and correspondingly, the allowance of "open-collector" output stages -- is maintained. This, however, is not the only way we can build logic gates.

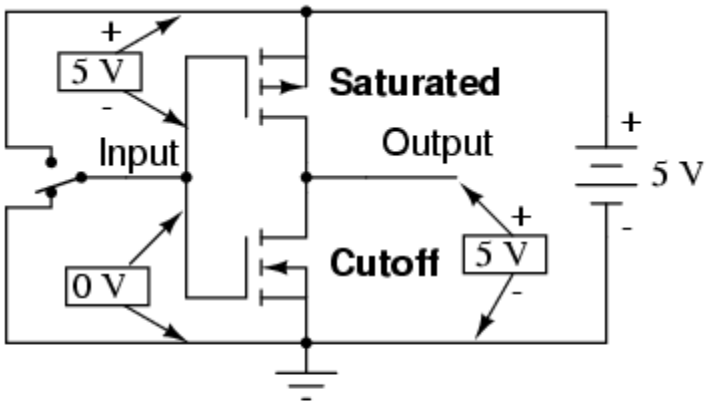
Field-effect transistors, particularly the insulated-gate variety, may be used in the design of gate circuits. Being voltage-controlled rather than current-controlled devices, IGFETs tend to allow very simple circuit designs. Take for instance, the following inverter circuit built using P- and N-channel IGFETs:

Inverter circuit using IGFETs



Notice the "V_{dd}" label on the positive power supply terminal. This label follows the same convention as "V_{cc}" in TTL circuits: it stands for the constant voltage applied to the drain of a field effect transistor, in reference to ground.

Let's connect this gate circuit to a power source and input switch, and examine its operation. Please note that these IGFET transistors are E-type (Enhancement-mode), and so are normally-off devices. It takes an applied voltage between gate and drain (actually, between gate and substrate) of the correct polarity to bias them on.

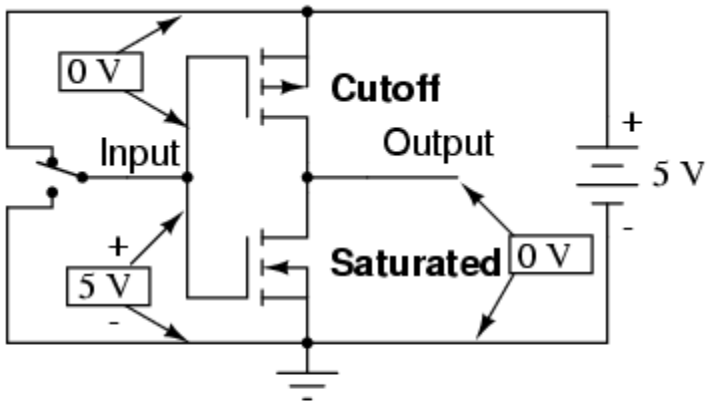


Input = "low" (0)
Output = "high" (1)

The upper transistor is a P-channel IGFET. When the channel (substrate) is made more positive than the gate (gate negative in reference to the substrate), the channel is enhanced and current is allowed between source and drain. So, in the above illustration, the top transistor is turned on.

The lower transistor, having zero voltage between gate and substrate (source), is in its normal mode: off. Thus, the action of these two transistors are such that the output terminal of the gate circuit has a solid connection to V_{dd} and a very high resistance connection to ground. This makes the output "high" (1) for the "low" (0) state of the input.

Next, we'll move the input switch to its other position and see what



happens:

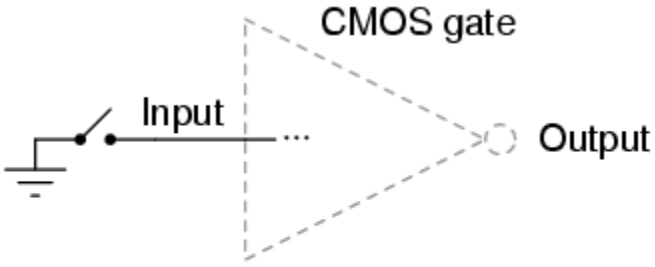
Input = "high" (1)
Output = "low" (0)

Now the lower transistor (N-channel) is saturated because it has sufficient voltage of the correct polarity applied between gate and substrate (channel) to turn it on (positive on gate, negative on the channel). The upper transistor, having zero voltage applied between its gate and substrate, is in its normal mode: off. Thus, the output of this gate circuit is now "low" (0). Clearly, this circuit exhibits the behavior of an inverter, or NOT gate.

Using field-effect transistors instead of bipolar transistors has greatly simplified the design of the inverter gate. Note that the output of this gate never floats as is the case with the simplest TTL circuit: it has a natural "totem-pole" configuration, capable of both sourcing and sinking load current. Key to this gate circuit's elegant design is the complementary use of both P- and N-channel IGFETs. Since IGFETs are more commonly known as MOSFETs (Metal-Oxide-Semiconductor Field Effect Transistor), and this circuit uses both P- and N-channel transistors together, the general classification given to gate circuits like this one isCMOS: Complementary Metal Oxide Semiconductor.

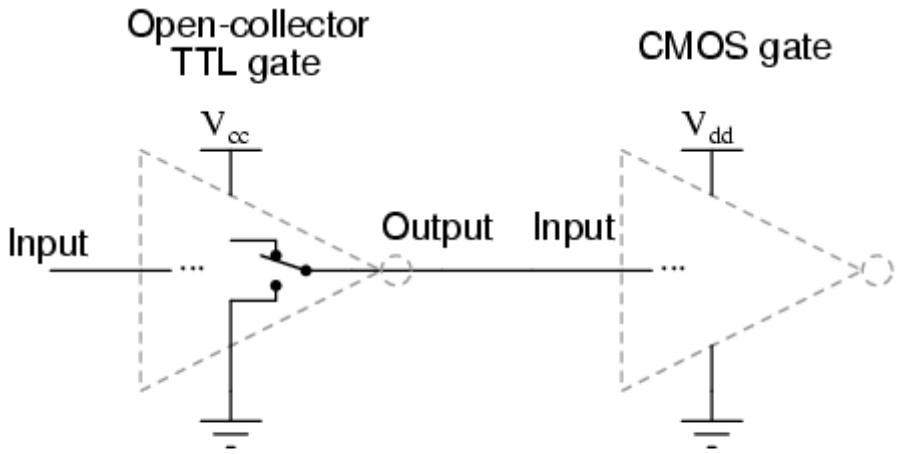
CMOS circuits aren't plagued by the inherent nonlinearities of the field-effect transistors, because as digital circuits their transistors always operate in either the saturated or cutoff modes and never in the active mode. Their inputs are, however, sensitive to high voltages generated by electrostatic (static electricity) sources, and may even be activated into "high" (1) or "low" (0) states by spurious voltage sources if left floating. For this reason, it is inadvisable to allow a CMOS logic gate input to float under any circumstances. Please note that this is very different from the behavior of a TTL gate where a floating input was safely interpreted as a "high" (1) logic level.

This may cause a problem if the input to a CMOS logic gate is driven by a single-throw switch, where one state has the input solidly connected to either V_{dd} or ground and the other state has the input floating (not connected to anything):



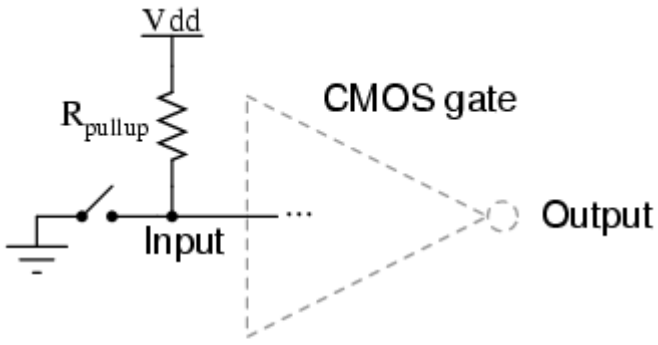
When switch is closed, the gate sees a definite "low" (0) input. However, when switch is open, the input logic level will be uncertain because it's floating.

Also, this problem arises if a CMOS gate input is being driven by an open-collector TTL gate. Because such a TTL gate's output floats when it goes "high" (1), the CMOS gate input will be left in an uncertain state:



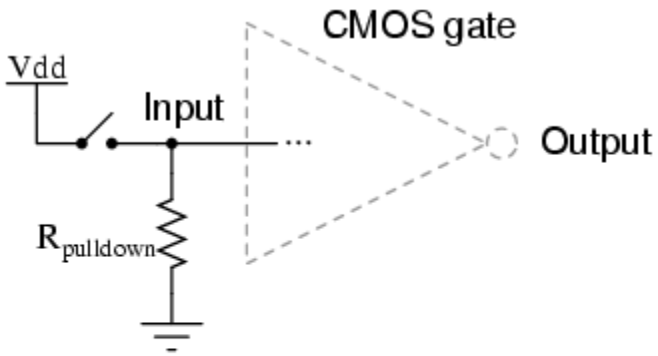
When the open-collector TTL gate's output is "high" (1), the CMOS gate's input will be left floating and in an uncertain logic state.

Fortunately, there is an easy solution to this dilemma, one that is used frequently inCMOS logic circuitry. Whenever a single-throw switch (or any other sort of gate output incapable of both sourcing and sinking current) is being used to drive aCMOS input, a resistor connected to either V_{dd} or ground may be used to provide a stable logic level for the state in which the driving device's output is floating. This resistor's value is not critical: 10 kΩ is usually sufficient. When used to provide a "high" (1) logic level in the event of a floating signal source, this resistor is known as a pull up resistor:



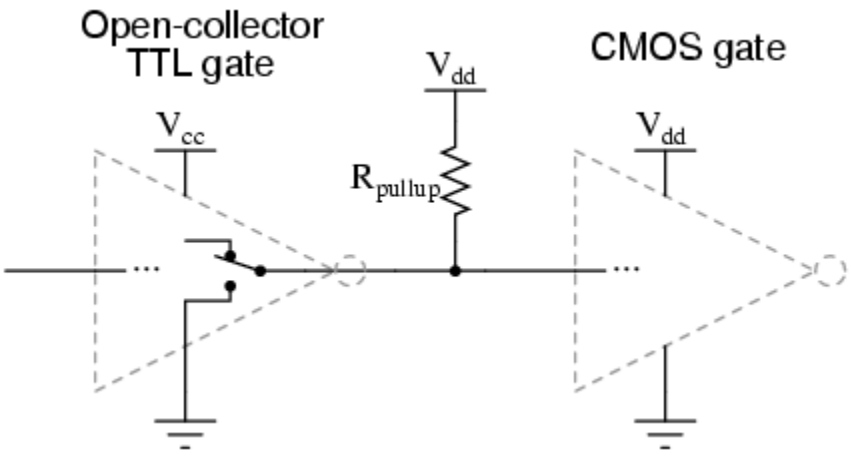
When switch is closed, the gate sees a definite "low" (0) input. When the switch is open, R_{pullup} will provide the connection to Vdd needed to secure a reliable "high" logic level for the CMOS gate input.

When such a resistor is used to provide a "low" (0) logic level in the event of a floating signal source, it is known as a pulldown resistor. Again, the value for a pulldown resistor is not critical:

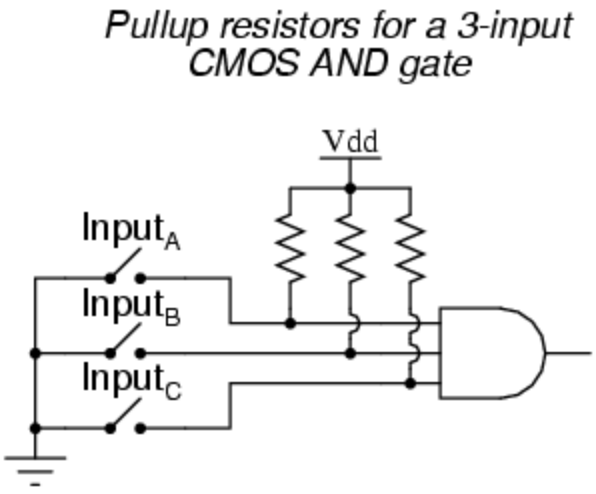


When switch is closed, the gate sees a definite "high" (1) input. When the switch is open, $R_{pulldown}$ will provide the connection to ground needed to secure a reliable "low" logic level for the CMOS gate input.

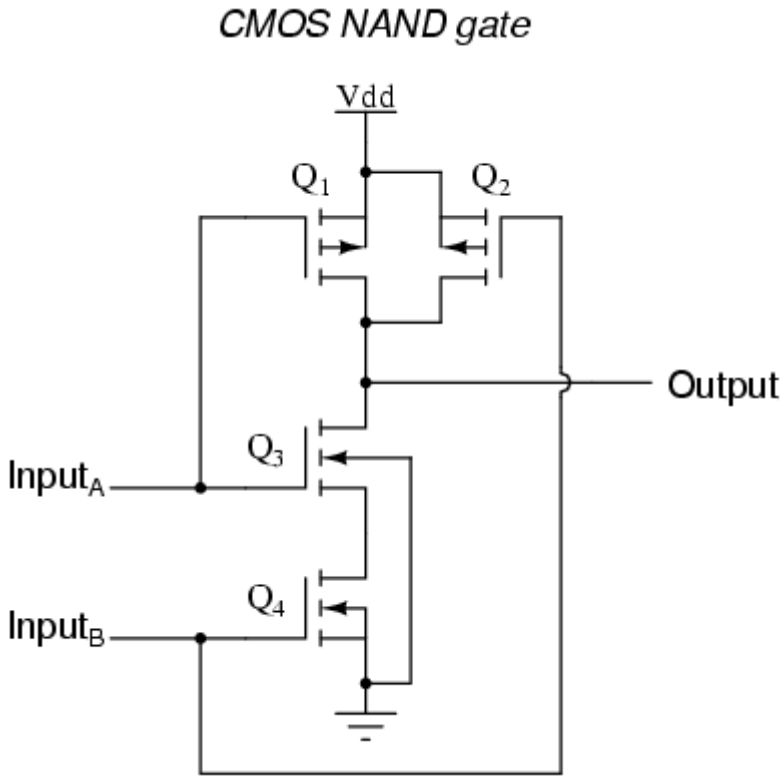
Because open-collector TTL outputs always sink, never source, current, pullup resistors are necessary when interfacing such an output to a CMOS gate input:



Although theCMOS gates used in the preceding examples were all inverters (single-input), the same principle of pullup and pulldown resistors applies to multiple-input CMOS gates. Of course, a separate pullup or pulldown resistor will be required for each gate input:

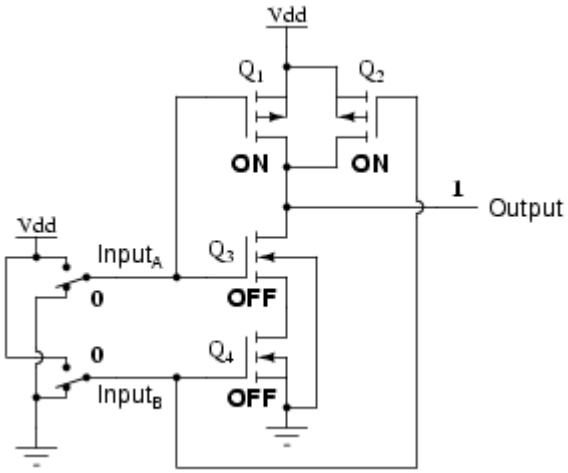


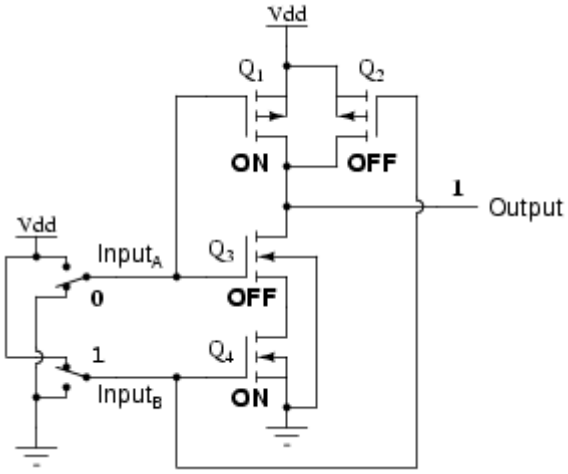
This brings us to the next question: how do we design multiple-input CMOS gates such as AND, NAND, OR, and NOR? Not surprisingly, the answer(s) to this question reveal a simplicity of design much like that of the CMOS inverter over its TTL equivalent.
For example, here is the schematic diagram for a CMOS NAND gate:

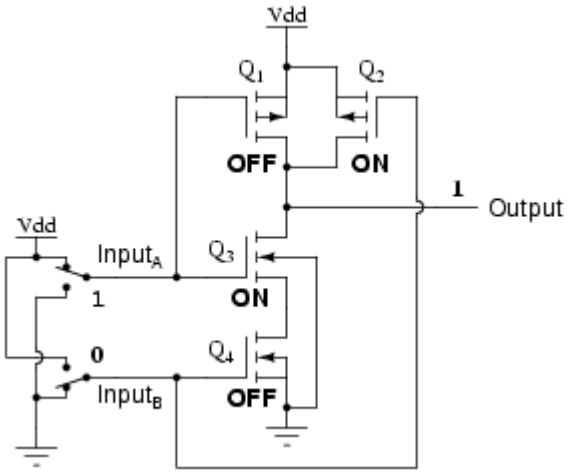


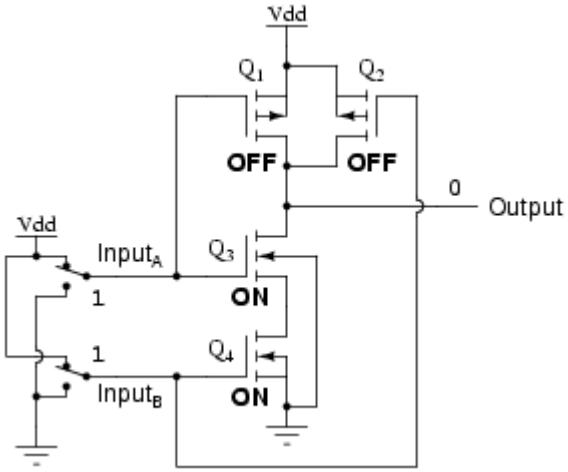
Notice how transistors Q_1 and Q_3 resemble the series-connected complementary pair from the inverter circuit. Both are controlled by the same input signal (input A), the upper transistor turning off and the lower transistor turning on when the input is "high" (1), and vice versa. Notice also how transistors Q_2 and Q_4 are similarly controlled by the

same input signal (input B), and how they will also exhibit the same on/off behavior for the same input logic levels. The upper transistors of both pairs (Q_1 and Q_2) have their source and drain terminals paralleled, while the lower transistors (Q_3 and Q_4) are series-connected. What this means is that the output will go "high" (1) if either top transistor saturates, and will go "low" (0) only if both lower transistors saturate. The following sequence of illustrations shows the behavior of this NAND gate for all four possibilities of input logic levels (00, 01, 10, and 11):

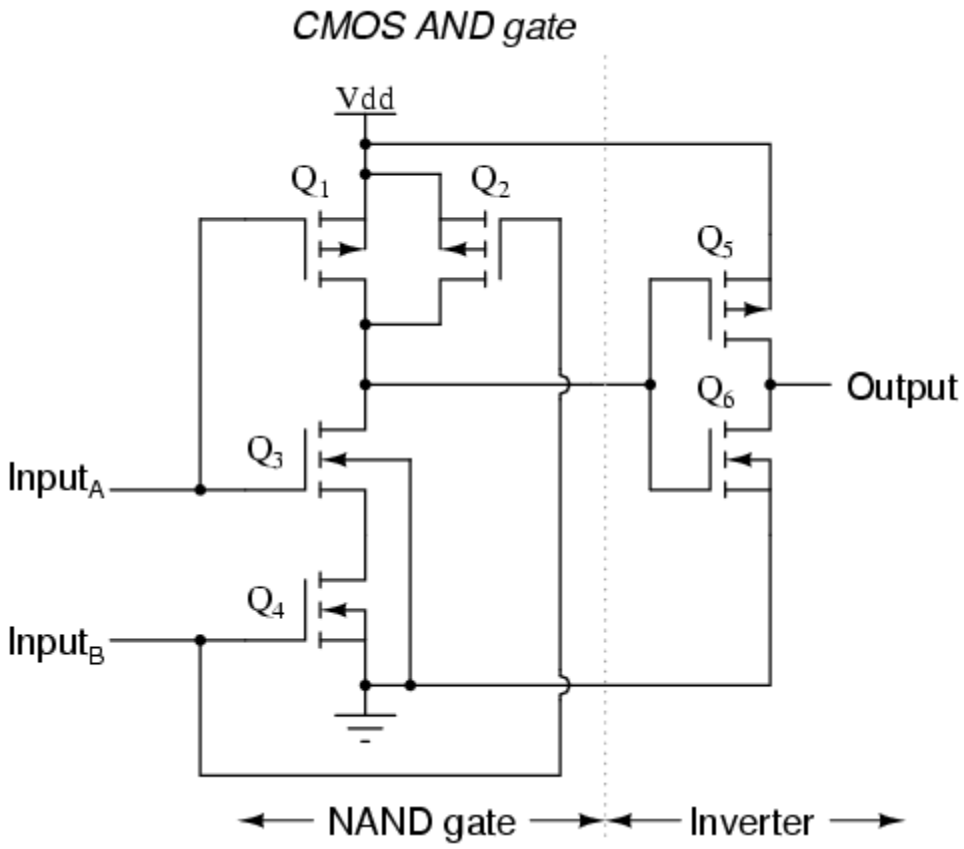




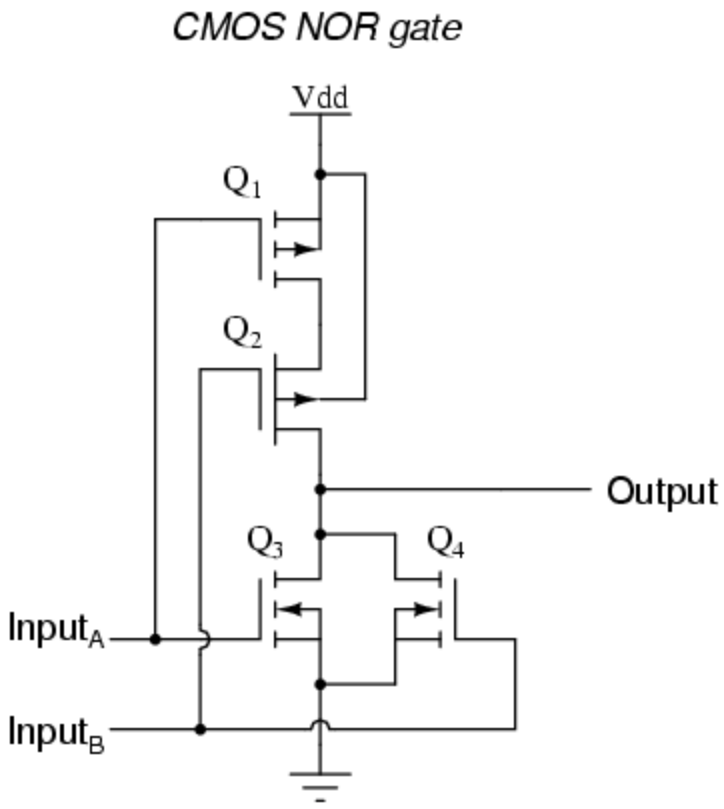




As with the TTL NAND gate, the CMOSAND gate circuit may be used as the starting point for the creation of an AND gate. All that needs to be added is another stage of transistors to invert the output signal:

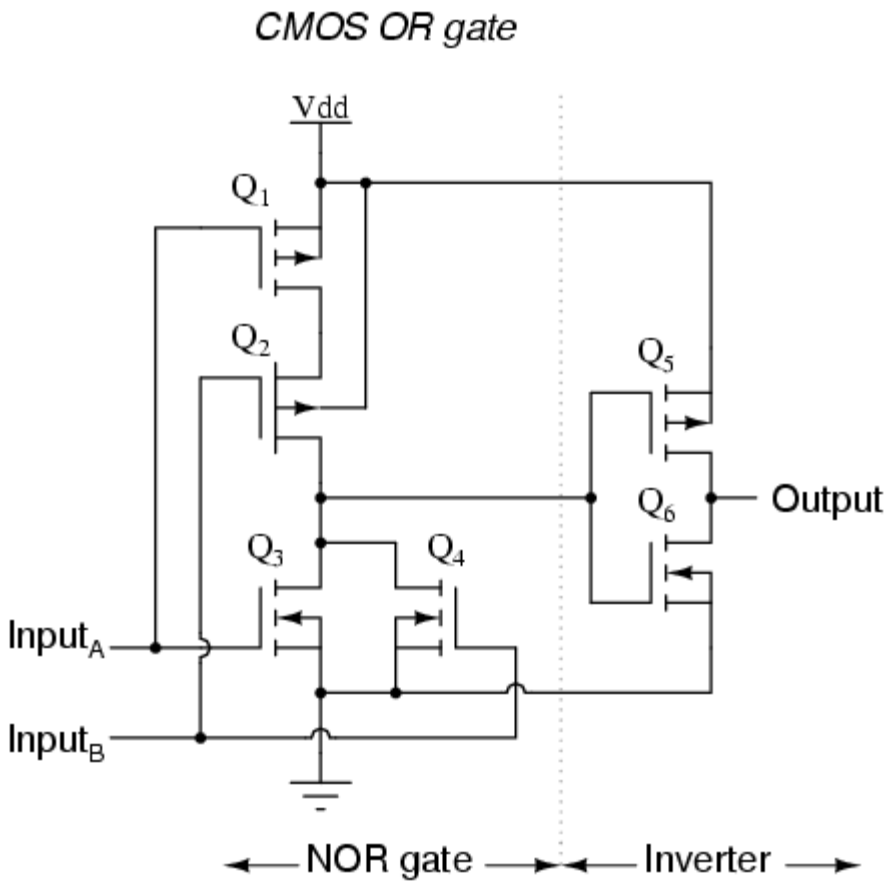


A CMOS NOR gate circuit uses four MOSFETs just like the NAND gate, except that its transistors are differently arranged. Instead of two paralleled sourcing (upper) transistors connected to V_{dd} and two series-connected sinking (lower) transistors connected to ground, the NOR gate uses two series-connected sourcing transistors and two parallel-connected sinking transistors like this:



As with the NAND gate, transistors Q_1 and Q_3 work as a complementary pair, as do transistors Q_2 and Q_4. Each pair is controlled by a single input signal. If either input A or input B are "high" (1), at least one of the lower transistors (Q_3 or Q_4) will be saturated, thus making the output "low" (0). Only in the event of both inputs being "low" (0) will both lower transistors be in cutoff mode and both upper transistors be saturated, the conditions necessary for the output to go "high" (1). This behavior, of course, defines the NOR logic function.

The OR function may be built up from the basic NOR gate with the addition of an inverter stage on the output:



Since it appears that any gate possible to construct using TTL technology can be duplicated in CMOS, why do these two "families" of logic design still coexist? The answer is that both TTL and CMOS have their own unique advantages.

First and foremost on the list of comparisons between TTL and CMOS is the issue of power consumption. In this measure of performance, CMOS is the unchallenged victor. Because the complementary P- and N-channel MOSFET pairs of a CMOS gate circuit are (ideally) never conducting at the same time, there is little or no current drawn by the circuit from the V_{dd} power supply except for what current is necessary to source current to a load. TTL, on the other hand, cannot function without some current drawn at all times, due to the biasing requirements of the bipolar transistors from which it is made.

There is a caveat to this advantage, though. While the power dissipation of a TTL gate remains rather constant regardless of its operating state(s), a CMOS gate dissipates more power as the frequency of its input signal(s) rises. If a CMOS gate is operated in a static (unchanging) condition, it dissipates zero power (ideally). However, CMOS gate circuits draw transient current during every output state switch from "low" to "high" and vice versa. So, the more often a CMOS gate switches modes, the more often it will draw current from the V_{dd} supply, hence greater power dissipation at greater frequencies.

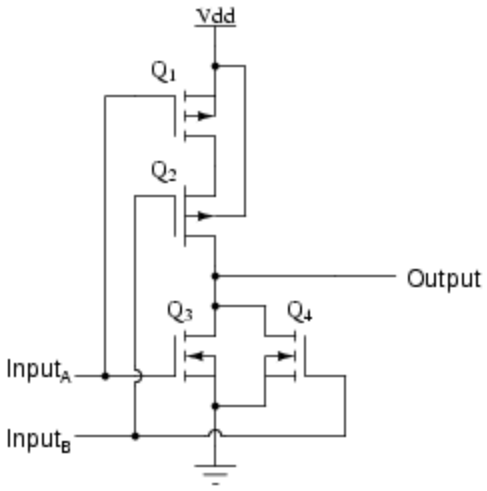
A CMOS gate also draws much less current from a driving gate output than a TTL gate because MOSFETs are voltage-controlled, not current-controlled, devices. This means that one gate can drive many more CMOS inputs than TTL inputs. The measure of how many gate inputs a single gate output can drive is called fan-out.

Another advantage that CMOS gate designs enjoy over TTL is a much wider allowable range of power supply voltages. Whereas TTL gates are restricted to power supply (V_{cc}) voltages between 4.75 and 5.25 volts, CMOS gates are typically able to operate on any voltage between 3 and 15 volts! The reason behind this disparity in power supply voltages is the respective bias requirements of MOSFET versus bipolar junction transistors. MOSFETs are controlled exclusively by gate voltage (with respect to substrate), whereas BJTs are current-controlled devices. TTL gate circuit resistances are precisely calculated for proper bias currents assuming a 5 volt regulated power supply. Any significant variations in that power supply voltage will result in the transistor bias currents being incorrect, which then results in unreliable (unpredictable) operation. The only effect that variations in power supply voltage have on a CMOS gate is the voltage definition of a "high" (1) state. For a CMOS gate operating at 15 volts of power supply voltage (V_{dd}), an input signal must be close to 15 volts in order to be considered "high" (1). The voltage threshold for a "low" (0) signal remains the same: near 0 volts.

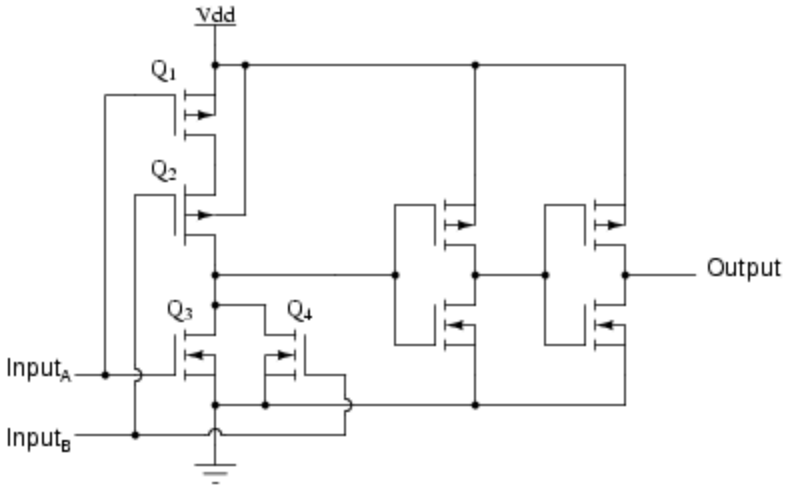
One decided disadvantage of CMOS is slow speed, as compared to TTL. The input capacitances of a CMOS gate are much, much greater than that of a comparable TTL gate -- owing to the use of MOSFETs rather than BJTs -- and so a CMOS gate will be slower to respond to a signal transition (low-to-high or vice versa) than a TTL gate, all other factors being equal. The RC time constant formed by circuit resistances and the input capacitance of the gate tend to impede the fast rise- and fall-times of a digital logic level, thereby degrading high-frequency performance.

A strategy for minimizing this inherent disadvantage of CMOS gate circuitry is to "buffer" the output signal with additional transistor stages, to increase the overall voltage gain of the device. This provides a faster-transitioning output voltage (high-to-low or low-to-high) for an input voltage slowly changing from one logic state to another. Consider this example, of an "unbuffered" NOR gate versus a "buffered," or B-series, NOR gate:

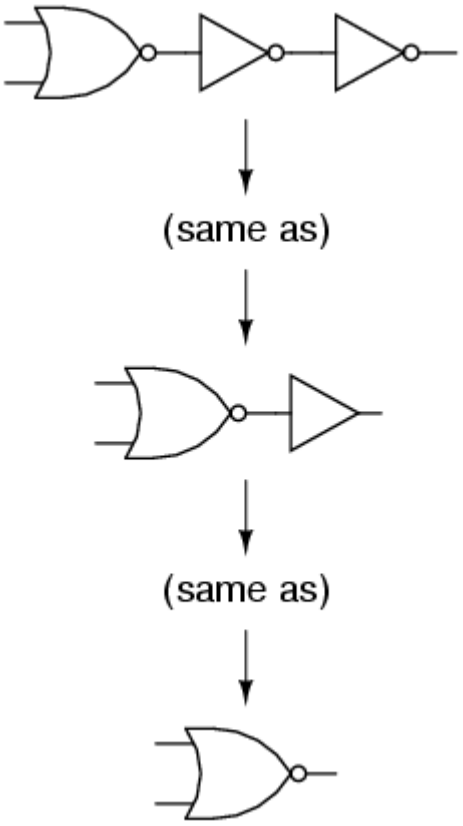
"Unbuffered" NOR gate



"B-series" (buffered) NOR gate



In essence, the B-series design enhancement adds two inverters to the output of a simple NOR circuit. This serves no purpose as far as digital logic is concerned, since two cascaded inverters simply cancel:



However, adding these inverter stages to the circuit does serve the purpose of increasing overall voltage gain, making the output more sensitive to changes in input state, working to overcome the inherent slowness caused by CMOS gate input capacitance.

IMPORTANT QUESTIONS

PART A

1. Define noise margin.
2. Define Rise Time.
3. What is body effect?
4. What is low noise margin?
5. What is stick diagram?
6. What are Lambda (O) -based design rules?
7. Define a super buffer.
8. Give the CMOS inverter DC transfer characteristics and operating regions
9. Give the various color coding used in stick diagram?
10. Define Delay time
11. Give the different symbols for transmission gate.
12. Compare between CMOS and bipolar technologies.
13. What are the static properties of complementary CMOS Gates?
14. Draw the circuit of a nMOS inverter
15. Draw the circuit of a CMOS inverter

PART B

1.List out the layout design rule. Draw the physical layout for one basic gate and

two universal gates. (16)

2. Explain the complimentary CMOS inverter DC characteristics. (16)

3. Explain the switching characteristics of CMOS inverter.(16)

4. Explain the concept of static and dynamic CMOS design (8)

5. Explain the construction and operation of transmission gates (8)

6. Derive the pull up to pull down ratio for an nMOS inverter driven through one or more pass transistors.

7. Explain super buffer.

8. Explain the inverter ration of nMOS device.

UNIT III

CIRCUIT CHARACTERISATION AND PERFORMANCE ESTIMATION

- Resistance estimation.
- Capacitance estimation.
- Inductance.
- Switching characteristics.
- Transistor sizing.
- Power dissipation and design margining.
- Charge sharing.
- Scaling.

INTRODUCTION:

In this chapter, we will investigate some of the physical factors which determine and ultimately limit the performance of digital VLSI circuits. The switching characteristics of digital integrated circuits essentially dictate the overall operating speed of digital systems. The dynamic performance requirements of a digital system are usually among the most important design specifications that must be met by the circuit designer. Therefore, the switching speed of the circuits must be estimated and optimized very early in the design phase.

The classical approach for determining the switching speed of a digital block is based on the assumption that the loads are mainly capacitive and lumped. Relatively simple delay models exist for logic gates with purely capacitive load at the output node; hence, the dynamic behavior of the circuit can be estimated easily once the load is determined. The conventional delay estimation approaches seek to classify three main components of the gate load, all of which are assumed to be purely capacitive, as: (1) internal parasitic capacitances of the transistors, (2) interconnect (line) capacitances, and (3) input capacitances of the fan-out gates. Of these three components, the load conditions imposed by the interconnection lines present serious problems.

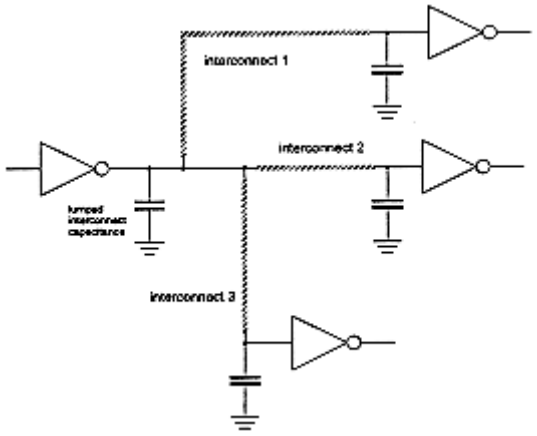


Figure-1: An inverter driving three other inverters over interconnection lines.

Figure 1 shows a simple situation where an inverter is driving three other inverters, linked over interconnection lines of different length and geometry. If the total load of each interconnection line can be approximated by a lumped capacitance, then the total load seen by the primary inverter is simply the sum of all capacitive components described above. The switching characteristics of the inverter are then described by the charge/discharge time of the load capacitance, as seen in Fig.2. The expected output voltage waveform of the inverter is given in Fig.3, where the propagation delay time is the primary measure of switching speed. It can be shown very easily that the signal propagation delay under these conditions is linearly proportional to the load capacitance.

In most cases, however, the load conditions imposed by the interconnection line are far from being simple. The line, itself a three-dimensional structure in metal and/or polysilicon,

usually has a non-negligible resistance in addition to its capacitance. The length/width ratio of the wire usually dictates that the parameters are distributed, making the interconnect a true transmission line. Also, an interconnect is very rarely “alone”, i.e., isolated from other influences. In real conditions, the interconnection line is in very close proximity to a number of other lines, either on the same level or on different levels. The capacitive/inductive coupling and the signal interference between neighboring lines should also be taken into consideration for an accurate estimation of delay.

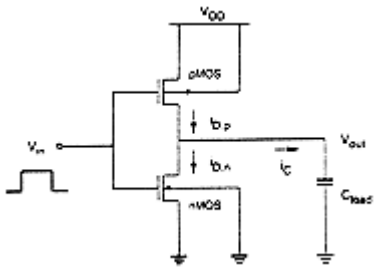


Figure-2: CMOS inverter stage with lumped capacitive load at the output node.

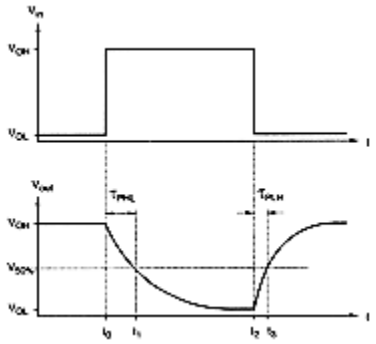


Figure-3: Typical input and output waveforms of an inverter with purely capacitive load.

2 The Reality with Interconnections

Consider the following situation where an inverter is driving two other inverters, over long interconnection lines. In general, if the time of flight across the interconnection line (as determined by the speed of light) is shorter than the signal rise/fall times, then the wire can be modeled as a capacitive load, or as a lumped or distributed RC network. If the interconnection lines are sufficiently long and the rise times of the signal waveforms are comparable to the time of flight across the line, then the inductance also becomes important, and the interconnection lines must be modeled as transmission lines. Taking into consideration the RLCG (resistance, inductance, capacitance, and conductance) parasitic (as seen in Fig.4), the signal transmission across the wire becomes a very complicated matter, compared to the relatively simplistic lumped- load case. Note that the signal integrity can be significantly degraded especially when the output impedance of the driver is significantly lower than the characteristic impedance of the transmission line.

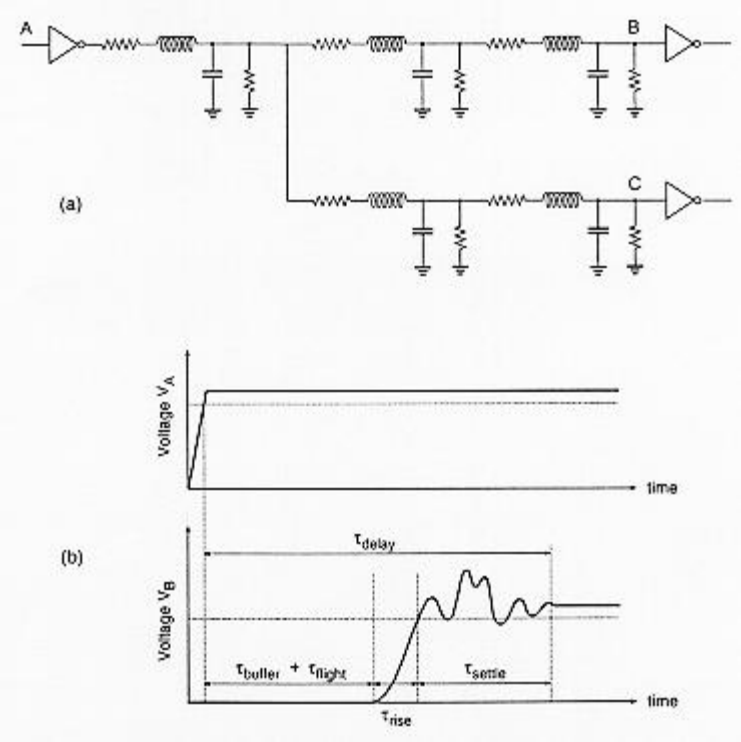


Figure.4: (a) An RLCG interconnection tree. (b) Typical signal waveforms at the nodes A and B, showing the signal delay and the various delay components.

The transmission-line effects have not been a serious concern in CMOS VLSI until recently, since the gate delay originating from purely or mostly capacitive load components dominated the line delay in most cases. But as the fabrication technologies move to finer (sub-micron) design rules, the intrinsic gate delay components tend to decrease dramatically. By contrast, the overall chip size does not decrease - designers just put more functionality on the same sized chip. A 100 mm2 chip has been a standard large chip for almost a decade. The factors which determine the chip size are mainly driven by the packaging technology, manufacturing equipment, and the yield. Since the chip size and the worst-case line length on a chip remain unchanged, the importance of interconnect delay increases in sub-micron technologies. In addition, as the widths of metal lines shrink, the transmission line effects and signal coupling between neighboring lines become even more pronounced.

This fact is illustrated in Fig.5, where typical intrinsic gate delay and interconnect delay are plotted qualitatively, for different technologies. It can be seen that for sub-micron technologies, the interconnect delay starts to dominate the gate delay. In order to deal with the implications and to optimize a system for speed, the designers must have reliable and efficient means of (1) estimating the interconnect parasitic in a large chip, and (2) simulating the time-domain effects. Yet we will see that neither of these tasks is simple - interconnect parasitic extraction and accurate simulation of line effects are two of the most difficult problems in physical design of VLSI circuits today.

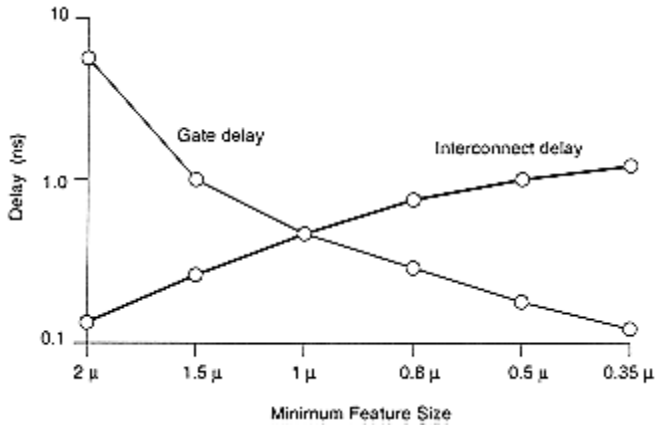


Figure.5: Interconnect delay dominates gate delay in sub-micron CMOS technologies.

Once we establish the fact that the interconnection delay becomes a dominant factor in CMOS VLSI, the next question is: how many of the interconnections in a large chip may cause serious problems in terms of delay. The hierarchical structure of most VLSI designs offers some insight on this question. In a chip consisting of several functional modules, each module contains a relatively large number of local connections between its functional blocks, logic gates, and transistors. Since these intra-module connections are usually made over short distances, their influence on speed can be simulated easily with conventional models. Yet there are also a fair amount of longer connections between the modules on a chip, the so-called inter-module connections. It is usually these inter-module connections which should be scrutinized in the early design phases for possible timing problems. Figure 6 shows the typical statistical distribution of wire lengths on a chip, normalized for the chip diagonal length. The distribution plot clearly exhibits two distinct peaks, one for the relatively shorter intra-module connections, and the other for the longer inter-module connections. Also note that a small number of interconnections may be very long, typically longer than the chip diagonal length. These lines are usually required for global signal bus connections, and for clock distribution networks. Although their numbers are relatively small, these long interconnections are obviously the most problematic ones.

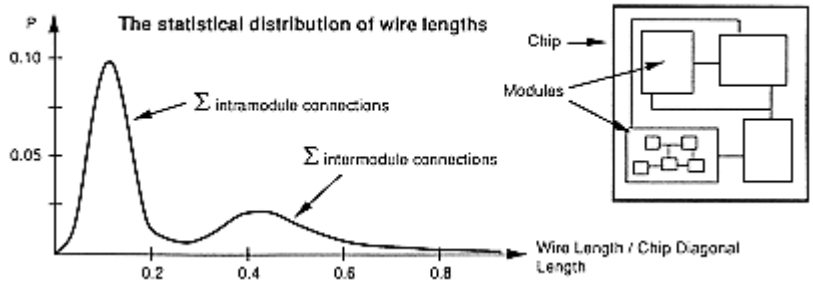


Figure.6: Statistical distribution of interconnection length on a typical chip.

To summarize the message of this section, we state that: (1) interconnection delay is becoming the dominating factor which determines the dynamic performance of large-scale systems, and (2) interconnect parasitics are difficult to model and to simulate. In the following sections, we will concentrate on various aspects of on-chip parasitics, and we will mainly consider capacitive and resistive components.

3 MOSFET Capacitances

The first component of capacitive parasitics we will examine is the MOSFET capacitances. These parasitic components are mainly responsible for the intrinsic delay of logic gates, and they can be modeled with fairly high accuracy for gate delay estimation. The extraction of transistor parasitics from physical structure (mask layout) is also fairly straight forward.

The parasitic capacitances associated with a MOSFET are shown in Fig.7 as lumped elements between the device terminals. Based on their physical origins, the parasitic device capacitances can be classified into two major groups: (1) oxide-related capacitances and (2) junction capacitances. The gate-oxide-related capacitances are Cgd

(gate-to-drain capacitance), Cgs (gate-to-source capacitance), and Cgb (gate-to-substrate capacitance). Notice that in reality, the gate-to-channel capacitance is distributed and voltage dependent. Consequently, all of the oxide-related capacitances described here change with the bias conditions of the transistor. Figure 8 shows qualitatively the oxide-related capacitances during cut-off, linear-mode operation and saturation of the MOSFET. The simplified variation of the three capacitances with gate-to-source bias voltage is shown in Fig. 9.

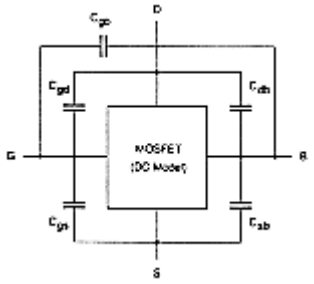


Figure7: Lumped representation of parasitic MOSFET capacitances.

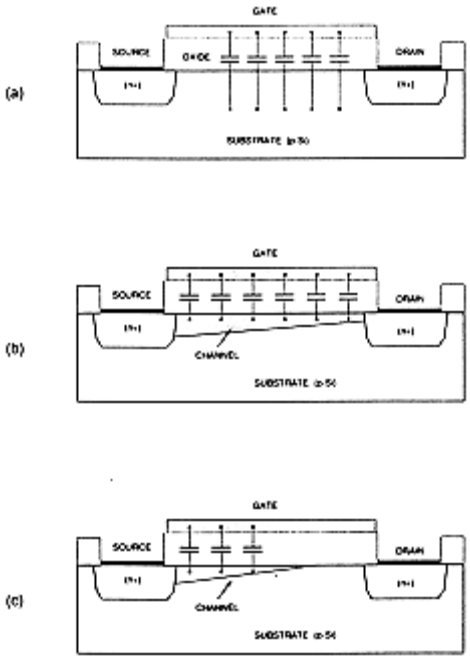


Figure 8: Schematic representation of MOSFET oxide capacitances during (a) cut-off, (b) linear- mode operation, and (c) saturation.

Note that the total gate oxide capacitance is mainly determined by the parallel-plate capacitance between the polysilicon gate and the underlying structures. Hence, the magnitude of the oxide-related capacitances is very closely related to (1) the gate oxide thickness, and (2) the area of the MOSFET gate. Obviously, the total gate capacitance decreases with decreasing device dimensions (W and L), yet it increases with decreasing gate oxide thickness. In sub-micron technologies, the horizontal dimensions (which dictate the gate area) are usually scaled down more easily than the horizontal dimensions, such as the gate oxide thickness. Consequently, MOSFET transistors fabricated using sub-micron technologies have, in general, smaller gate capacitances.

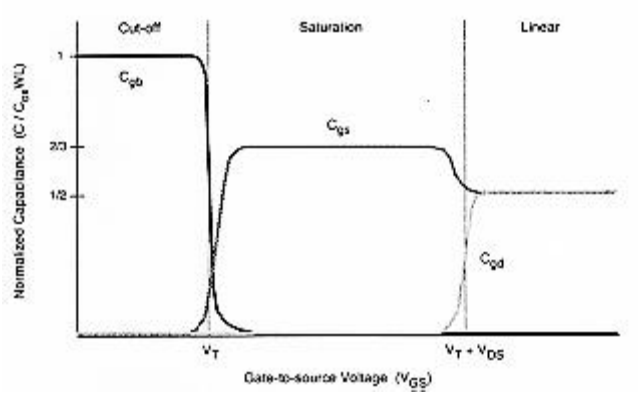


Figure 9: Variation of oxide capacitances as functions of gate-to-source voltage.

Now we consider the voltage-dependent source-to-substrate and drain-to-substrate capacitances, Csb and Cdb. Both of these capacitances are due to the depletion charge surrounding the respective source or drain regions of the transistor, which are embedded in the substrate. Figure 10 shows the simplified geometry of an n-type diffusion region within the p-type substrate. Here, the diffusion region has been approximated by a rectangular box, which consists of five planar pn-junctions. The total junction capacitance is a function of the junction area (sum of all planar junction areas), the doping densities, and the applied terminal voltages. Accurate methods for estimating the junction capacitances based on these data are readily available in the literature, therefore, a detailed discussion of capacitance calculations will not be presented here.

One important aspect of parasitic device junction capacitances is that the amount of capacitance is a linear function of the junction area. Consequently, the size of the drain or the source diffusion area dictates the amount of parasitic capacitance. In sub-micron technologies, where the overall dimensions of the individual devices are scaled down, the parasitic junction capacitances also decrease significantly.

It was already mentioned that the MOSFET parasitic capacitances are mainly responsible for the intrinsic delay of logic gates. We have seen that both the oxide-related parasitic capacitances and the junction capacitances tend to decrease with shrinking device dimensions, hence, the relative significance of intrinsic gate delay diminishes in sub-micron technologies.

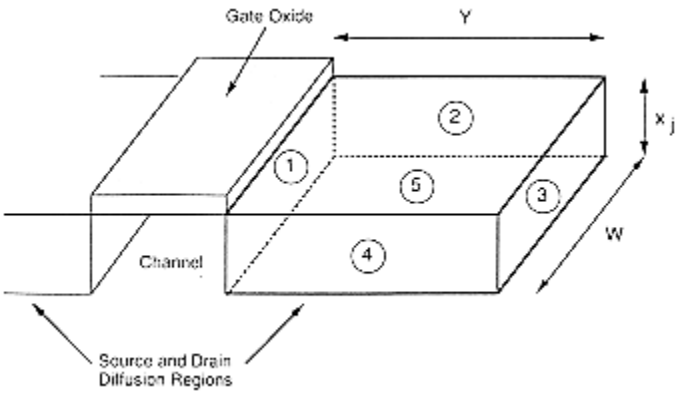


Figure10: Three-dimensional view of the n-type diffusion region within the p-type substrate.

4 Interconnect Capacitance Estimation

In a typical VLSI chip, the parasitic interconnect capacitances are among the most difficult parameters to estimate accurately. Each interconnection line (wire) is a three dimensional structure in metal and/or polysilicon, with significant variations of shape, thickness, and vertical distance from the ground plane (substrate). Also, each interconnect line is typically surrounded by a number of other lines, either on the same level or on different levels. Figure 4.11 shows a possible, realistic situation where interconnections on three different levels run in close proximity of each other. The accurate estimation of the parasitic capacitances of these wires with respect to the ground plane, as well as with respect to each other, is obviously a complicated task.

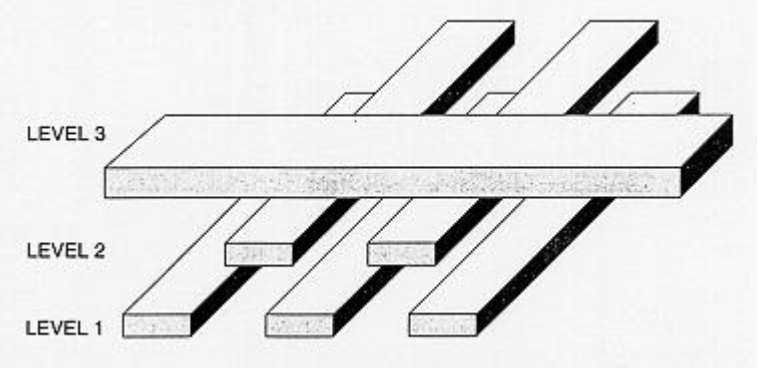


Figure 11: Example of six interconnect lines running on three different levels.

Unfortunately for the VLSI designers, most of the conventional computer-aided VLSI design tools have a relatively limited capability of interconnect parasitic estimation. This is true even for the design tools regularly used for sub-micron VLSI design, where interconnect parasitics were shown to be very dominant. The designer should therefore be aware of the

physical problem and try to incorporate this knowledge early in the design phase, when the initial floorplanning of the chip is done.

First, consider the section of a single interconnect which is shown in Fig12. It is assumed that this wire segment has a length of (l) in the current direction, a width of (w) and a thickness of (t). Moreover, we assume that the interconnect segment runs parallel to the chip surface and is separated from the ground plane by a dielectric (oxide) layer of height (h). Now, the correct estimation of the parasitic capacitance with respect to ground is an important issue. Using the basic geometry given in Fig 12, one can calculate the parallel-plate capacitance Cpp of the interconnect segment. However, in interconnect lines where the wire thickness (t) is comparable in magnitude to the ground-plane distance (h), fringing electric fields significantly increase the total parasitic capacitance (Fig13).

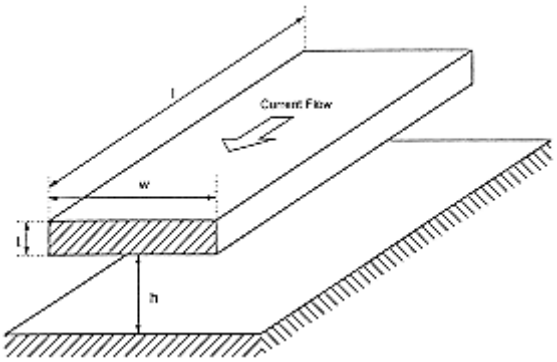


Figure12: Interconnect segment running parallel to the surface, used for parasitic capacitance estimations.

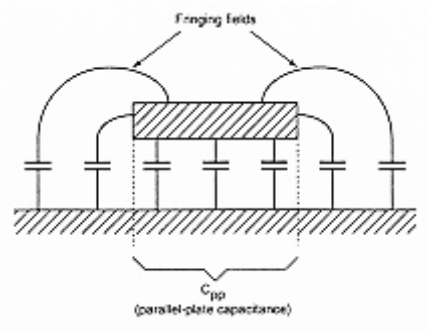


Figure 13: Influence of fringing electric fields upon the parasitic wire capacitance.

Figure 14 shows the variation of the fringing-field factor FF = C_{total}/C_{pp}, as a function of (t/h), (w/h) and (w/l). It can be seen that the influence of fringing fields increases with the decreasing (w/h) ratio, and that the fringing-field capacitance can be as much as 10-20 times larger than the parallel-plate capacitance. It was mentioned earlier that the sub-micron fabrication technologies allow the width of the metal lines to be decreased somewhat, yet the thickness of the line must be

preserved in order to ensure structural integrity. This situation, which involves narrow metal lines with a considerable vertical thickness, is especially vulnerable to fringing field effects.

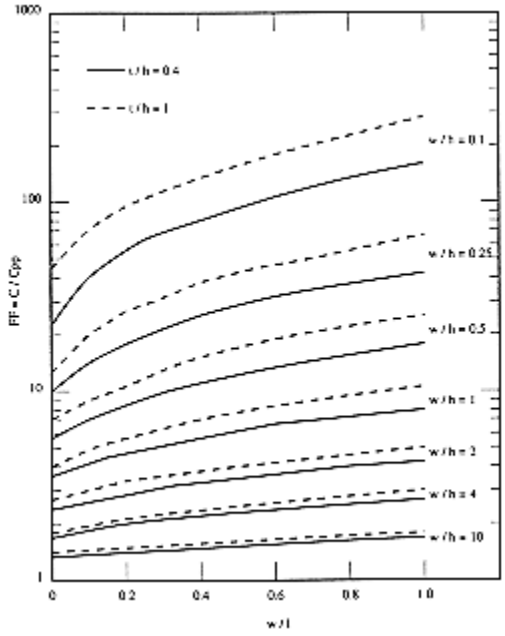


Figure-14: Variation of the fringing-field factor with the interconnect geometry.

A set of simple formulas developed by Yuan and Trick in the early 1980's can be used to estimate the capacitance of the interconnect structures in which fringing fields complicate the parasitic capacitance calculation. The following two cases are considered for two different ranges of line width (w).

$$C = \varepsilon \left[\frac{\left(w - \frac{t}{2} \right)}{h} + \frac{2\pi}{\ln\left(1 + \frac{2h}{t} + \sqrt{\frac{2h}{t}\left(\frac{2h}{t} + 2 \right)} \right)} \right] \quad for \quad w \geq \frac{t}{2} \tag{4.1}$$

$$C=\varepsilon\left[\frac{w}{h}+\frac{\pi\left(1-0.0543\cdot\frac{t}{2h}\right)}{\ln\left(1+\frac{2h}{t}+\sqrt{\frac{2h}{t}\left(\frac{2h}{t}+2\right)}\right)}+1.47\right] \quad for \quad w<\frac{t}{2} \tag{4.2}$$

These formulas permit the accurate approximation of the parasitic capacitance values to within 10% error, even for very small values of (t/h). Figure 4.15 shows a different view of the line capacitance as a function of (w/h) and (t/h). The linear dash-dotted line in this plot represents the corresponding parallel-plate capacitance, and the other two curves represent the actual capacitance, taking into account the fringing-field effects.

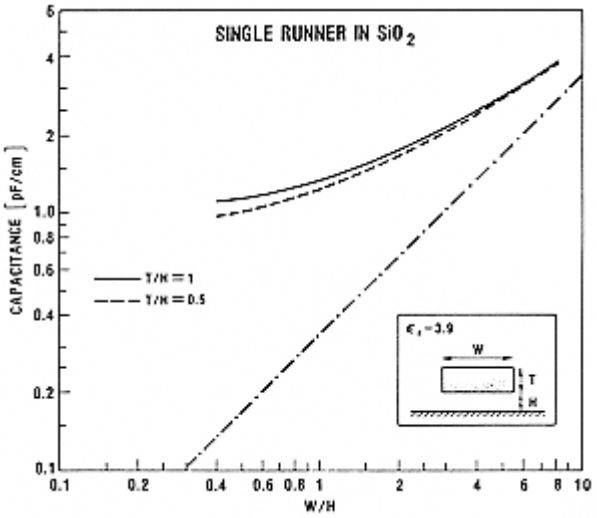


Figure-15: Capacitance of a single interconnect, as a function of (w/h) and (t/h).

Now consider the more realistic case where the interconnection line is not “alone” but is coupled with other lines running in parallel. In this case, the total parasitic capacitance of the line is not only increased by the fringing-field effects, but also by the capacitive coupling between the lines. Figure 16 shows the capacitance of a line which is coupled with two other lines on both sides, separated by the minimum design rule. Especially if both of the neighboring lines are biased at ground potential, the total parasitic capacitance of the interconnect running in the middle (with respect to the ground plane) can be more than 20 times as large as the simple parallel-plate capacitance. Note that the capacitive coupling between neighboring lines is increased when the thickness of the wire is comparable to its width.

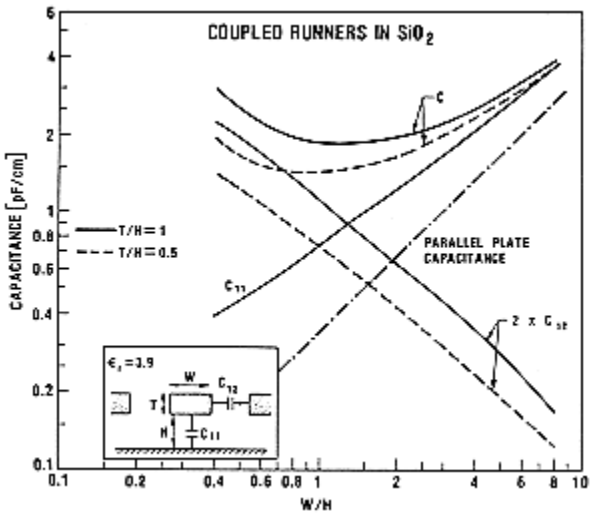


Figure-16: Capacitance of coupled interconnects, as a function of (w/h) and (t/h).

Figure 17 shows the cross-section view of a double-metal CMOS structure, where the individual parasitic capacitances between the layers are also indicated. The cross-section does not show a MOSFET, but just a portion of a diffusion region over which some metal lines may pass. The inter-layer capacitances between the metal-2 and metal-1, metal-1 and polysilicon, and metal-2 and polysilicon are labeled as Cm2m1, Cm1p and Cm2p, respectively. The other parasitic capacitance components are defined with respect to the substrate. If the metal line passes over an active region, the oxide thickness underneath is smaller (because of the active area window), and consequently, the capacitance is larger. These special cases are labeled as Cm1a and Cm2a. Otherwise, the thick field oxide layer results in a smaller capacitance value.

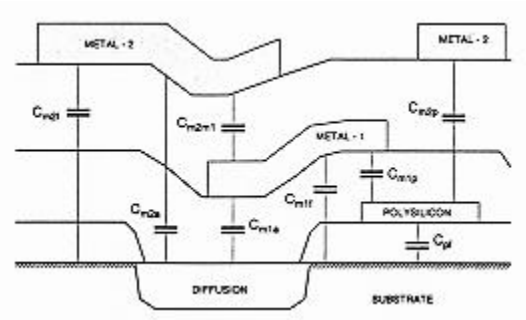


Figure-17: Cross-sectional view of a double-metal CMOS structure, showing capacitances between layers.

The vertical thickness values of the different layers in a typical 0.8 micron CMOS technology are given below as an example.

Field oxide thickness	0.52	um	
Gate oxide thickness	16.0	nm	
Poly 1 thickness	0.35	um	(minimum width 0.8 um)
Poly-metal oxide thickness	0.65	um	
Metal 1 thickness	0.60	um	(minimum width 1.4 um)
Via oxide thickness	1.00	um	
Metal 2 thickness	1.00	um	(minimum width 1.6 um)
n+ junction depth	0.40	um	
p+ junction depth	0.40	um	
n-well junction depth	3.50	um	

The list below contains the capacitance values between various layers, also for a typical 0.8 micron CMOS technology.

Poly over field oxide	(area)	0.066	fF/um2
Poly over field oxide	(perimeter)	0.046	fF/um
Metal-1 over field oxide	(area)	0.030	fF/um2
Metal-1 over field oxide	(perimeter)	0.044	fF/um
Metal-2 over field oxide	(area)	0.016	fF/um2
Metal-2 over field oxide	(perimeter)	0.042	fF/um
Metal-1 over poly	(area)	0.053	fF/um2
Metal-1 over poly	(perimeter)	0.051	fF/um
Metal-2 over poly	(area)	0.021	fF/um2
Metal-2 over poly	(perimeter)	0.045	fF/um
Metal-2 over metal-1	(area)	0.035	fF/um2
Metal-2 over metal-1	(perimeter)	0.051	fF/um

For the estimation of interconnect capacitances in a complicated three-dimensional structure, the exact geometry must be taken into account for every portion of the wire. Yet this requires an unacceptable amount of computation in a large circuit, even if simple formulas are applied for the calculation of capacitances. Usually, chip manufacturers supply the area capacitance (parallel-plate cap) and the perimeter capacitance (fringing-field cap) figures for each layer, which are backed up by measurement of capacitance test structures. These figures can be used to extract the parasitic capacitances from the mask layout. It is often prudent to include test structures on chip that enable the designer to independently calibrate a process to a set of design tools. In some cases where the entire chip performance is influenced by the parasitic capacitance of a specific line, accurate 3-D simulation is the only reliable solution.

5 Interconnect Resistance Estimation

The parasitic resistance of a metal or polysilicon line can also have a profound influence on the signal propagation delay over that line. The resistance of a line depends on the type of material used (polysilicon, aluminum, gold ...), the dimensions of the line and finally, the number and locations of the contacts on that line. Consider again the interconnection line shown in Fig12. The total resistance in the indicated current direction can be found as

$$R_{metal} = \rho \cdot \frac{l}{w \cdot t} = R_{sheet}\left(\frac{l}{w}\right) \tag{4.2}$$

where the greek letter ro represents the characteristic resistivity of the interconnect material, and Rsheet represents the sheet resistivity of the line, in (ohm/square). For a typical polysilicon layer, the sheet resistivity is between 20-40 ohm/square, whereas the sheet resistivity of silicide is about 2- 4 ohm/square. Using the formula given above, we can estimate the total parasitic resistance of a wire segment based on its geometry. Typical metal-poly and metal-diffusion contact resistance values are between 20-30 ohms, while typical via resistance is about 0.3 ohms.

In most short-distance aluminum and silicide interconnects, the amount of parasitic wire resistance is usually negligible. On the other hand, the effects of the parasitic resistance must be taken into account for longer wire segments. As a first-order approximation in simulations, the total lumped resistance may be assumed to be connected in series with the total lumped capacitance of the wire. A much better approximation of the influence of distributed parasitic resistance can be obtained by using an RC-ladder network model to represent the interconnect segment (Fig18). Here, the interconnect segment is divided into smaller, identical sectors, and each sector is represented by an RC-cell. Typically, the number of these RC-cells (i.e., the resolution of the RC model) determines the accuracy of the simulation results. On the other hand, simulation time restrictions usually limit the resolution of this distributed line model.

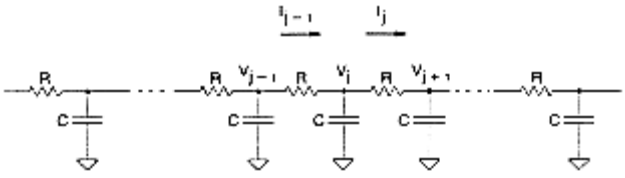


Figure-18: RC-ladder network used to model the distributed resistance and capacitance of an interconnect.

INDUCTANCE:

A 60 GHz cross-coupled differential LC CMOS VCO is presented in this paper, which is optimized for a large frequency tuning range using conventional MOSFET varactors. The MMIC is fabricated on digital 90 nm SOI technology and requires a circuit area of less than 0.1 mm^2 including the 50 Ω output buffers. Within a frequency control range from 52.3 GHz to 60.6 GHz, a supply voltage of 1.5 V and a supply current of 14 mA, the circuit delivers a very constant output power of –6.8 ± 0.2 dBm and yields a phase noise between –85 to -92 dBc/Hz at 1 MHz frequency offset.

INTRODUCTION:

Over the last years, the speed gap between leading edge III/V and CMOS technologies has been significantly decreased. Today, SOI CMOS technologies allow the efficient scaling of the transistor gate length resulting in a f_t and f_{max} of up to 243 GHz and 208 GHz, respectively [1-2].

This makes the realization of analog CMOS circuits at microwave frequencies possible leading to promising market perspectives for commercial applications. Circuits such as a 26-42 GHz low noise amplifier [3] and a 30-40 GHz mixer [4] have been realized demonstrating the suitability of SOI CMOS technologies for analog applications at millimeter wave frequencies.

To the best knowledge of the authors, the highest oscillation frequency of a CMOS VCO reported to date is 51 GHz [5]. The circuit uses 0.12 μm bulk technology and has been optimized for a high oscillation frequency. Since high oscillation frequency and high tuning range are contrary goals, the achieved tuning range at fixed supply voltage of 1.5 V is less than 1.5%. Due to the high process tolerances of aggressively scaled CMOS technologies, the oscillation frequency can significantly vary compared to the nominal value. Thus, in practice, a much higher tuning range is desired to allow a compensation of these variations.

A 40 GHz SOI CMOS VCO with 1.5 V supply voltage, a phase noise of -90 dBc/Hz at 1 MHz offset and a high tuning range of 9% has been reported [6]. The high tuning range is achieved by applying special accumulation MOS varactor diodes having a high capacitance control range $c_R = C_{vmax}/C_{vmin}$ of 6 [7]. These varactors require additional processing steps making the technology more expensive. An off-chip bias-T is required for the buffer amplifier to minimize the loading of the oscillator core.

TABLE ICOMPARISON WITH STATE-OF-THE-ART VCOs

Technology/ Speed	Center frequency	Tuning range	Phase noise @1MHz offset	Supply power	Ref.
SiGe HBT/f_t=120GHz	43GHz	11.8%	-96dBc	3V×121mA	[8]
0.12μm CMOS/n.a.	51GHz	2%	-85Bc/Hz	1.5V×6.5mA	[5]
0.13μm SOI CMOS/f_{max}=168GHz	40GHz	9%	-90dBc/Hz	1.5V×7.5mA*	[6]
90nm SOI CMOS/f_{max}=160GHz	57GHz	16%	-90dBc/Hz	1.5V×14mA	This work
	60GHz	14%	-94dBc/Hz	1.2V×8mA	

In this paper, a fully integrated 60 GHz SOI CMOS VCO is presented, which applies conventional MOSFET varactors. To allow process variations and high yield, the circuit is optimized for high frequency tuning range. Target applications are commercial wideband WLAN and optical transceivers operating around 60 GHz. Despite the high oscillation frequency, which to the best knowledge of the authors is the highest reported to date for a CMOS based oscillator, a high tuning range of more than 10 % is achieved with varactors having a c_R of only 2.

A comparison with recently reported silicon based VCOs is given in TABLE I.

II. TECHNOLOGY

The VCO was fabricated using a 90 nm IBM VLSI SOI CMOS technology featuring a metal stack with 8 metal layers. A thin isolation layer between the active region and the substrate allows a relatively high substrate resistivity of 13.5 ± 5 Ωcm without increasing the threshold voltages V_{th} of the FETs required for digital applications. Thus, relatively high Q factors and operation frequencies can be achieved for the passive devices, which are mandatory for analog applications and oscillators. The possibility of highly integrated single chip solutions makes this technology well suited for future commercial applications.

A. FETs

In Fig. 1, the simplified small signal equivalent circuit of a typical n-channel FET is shown. The device with V_{th} of 0.25 V and gate width w of 16 μm is biased in class A operation yielding a f_t and f_{max} of approximately 150 GHz and 160 GHz, respectively, for the experimental hardware.

A

Fig. 1. Simplified small signal equivalent circuit of MOSFET with w = 16 μm at V_{gs} = 0.5V, V_{ds} = 1V and I_{ds} = 4mA.

B. Passive devices

The upper copper metal is used for the realization of the inductive transmission lines. It has the largest distance to the lossy substrate and the highest metal thickness, which are approximately 7 μm and 1 μm, respectively. The line width is 4 μm yielding a characteristic impedance of 95 Ω and an inductance of 0.9 nH/mm. For a line with length of 100 μm, an inductance of 90 pH and a quality factor of 20 at 60 GHz was extracted from measurements.

Unfortunately, the used commercial VLSI process does not feature special varactor diodes with large tuning ranges as reported in [7]. However, the process provides thick oxide MOSFET capacitors allowing a C_R of approximately 2 and quality factors around 5 at 60 GHz.

The three top metals are used for the realization of the signal pads, which have a size of 54 μm x 50 μm. A low insertion loss of approximately 0.5 dB was measured at 60 GHz.

II. CIRCUIT DESIGN

The circuit was simulated using SOI BSIM FET large signal models, and lumped equivalent circuits for the inductive lines, varactors, interconnects and pads. The applied software tool is Cadence.

B

Fig. 2. Simplified VCO circuit schematics, C_{var} = [25-50fF], L_R = 90nH, $w_o = w_b$ = 16μm, w_{dc} = 32μm, R_b = 75μm, R_{dc1} = 1.4kΩ, R_{dc1} = 2kΩ.

The simplified circuit schematics of the cross-coupled differential LC oscillator is shown in Fig. 2. A common drain output buffer is used since it has a high input and low output impedance thereby minimizing the loading of the oscillator core and allowing 50 Ω output matching. A voltage divider generates the required V_{gs} of the current source transistor.

The frequency tuning range of the circuit is given by

$$\Delta\omega = \frac{1}{\sqrt{L_R (C_{v\min} + C_p)}} - \frac{1}{\sqrt{L_R (C_{v\max} + C_p)}} \quad (1)$$

with

$$C_p = C_{o,in} + C_{o,out} + C_{b,in} + C_L \quad (2)$$

as the total load of the oscillator core mainly consisting of the input and output capacitance of the oscillator core transistors $C_{o,in}$ and $C_{o,out}$, the input capacitance of the buffer transistor $C_{b,in}$ and the parasitic capacitance of the inductive line C_L.

To reach a high frequency control range, large varactor and small transistor sizes have to be chosen. However, decreasing of the oscillator transistor size w_o lowers the g_m required to compensate the losses of the resonator. This limits the maximum oscillation frequency since the losses increase with frequency. The minimum size of the buffer transistor w_b is determined by the output power of the oscillator core. Consequently, the transistor and varactor sizes were optimized to ensure oscillation up to 65 GHz and to reach a frequency tuning range of more than 10%.

With values of $C_{o,in} \approx$ 25 fF, $C_{b,out} \approx$ 3.75 fF, $C_{o,out} \approx$ 10 fF, $C_L \approx$ 3 fF, L_R = 90 pH, C_{vmin} = 25 fF and C_{vmax} = 50 fF, a tuning range from 55.4 GHz to 64.9 GHz can be calculated from Eqs. (1) and (2).

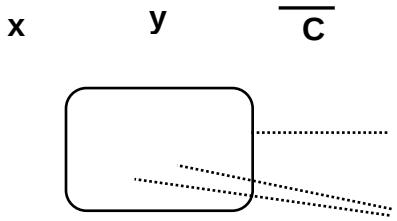


Fig. 3. Photograph of the VCO chip with overall size of 0.3 mm × 0.25 mm.

Higher oscillation frequencies are possible by proper downscaling of C_V, L_r and w_o. We assume that w_b can not be decreased due to large signal constraints. The decrease of w_o and L_r is limited since the elements determine the loop gain required for loss compensation and stable oscillation. A significant increase of the oscillation frequency can be achieved by decreasing of C_V. Unfortunately, this decreases the frequency tuning range as indicated in Eq. (1). For a frequency tuning range of 2 % and 0%, maximum oscillation frequencies of approximately 76 GHz and 80 GHz, respectively, would be possible. This is a interesting insight concerning the exploration of this technology and topology towards highest frequencies. However, in practice, this tuning range would be too small to compensate significant process variations thereby limiting the yield. Consequently, to have a reasonable tolerance margin, the circuit was optimized for a lower oscillation frequency of 60 GHz, which is still very high for a CMOS VCO.

A photograph of the compact MMIC is shown in Fig. 3. The overall chip size is 0.3 mm × 0.25 mm.

III. RESULTS

Measurements were performed on-wafer using an HP 8565E spectrum analyzer and an HP11974V pre-selected mixer.

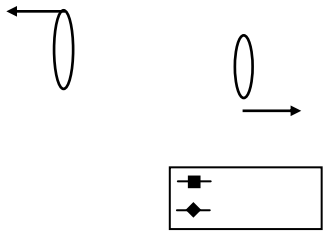


Fig. 4. Measured frequency tuning range and varactor potential versus tuning voltage.

A. Tuning range

The high tuning range of the circuit of 55.5-62.9 GHz and 52.3-60.6 GHz at supply voltages of V_{dd} = 1.2 V and V_{dd} = 1.5 V, respectively are shown in Fig. 4 including the effective varactor potential. The corresponding supply currents are 8 mA and 14 mA, respectively. The achieved tuning range agrees well with the theoretical considerations made in Section II.

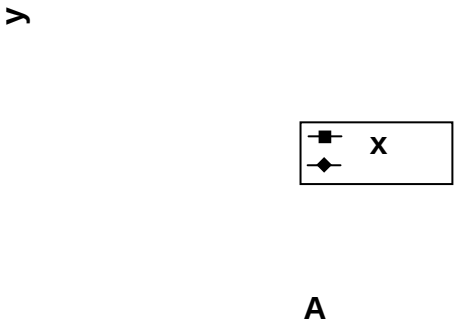


Fig. 5. Measured output power versus tuning voltage.

B. Output power

The measured output power versus tuning voltage is illustrated in Fig. 5. The losses of the cables and the RF probe were taken into account. A very constant output power of –6.8 ± 0.2 dBm was measured at V_{dd} = 1. 5V. An output power between –14 to -18.5 dBm was measured at the lower bias with V_{dd} = 1.2V.

C. Phase noise

A typical output spectrum is depicted in Fig. 6. At V_{dd} = 1.2 V and a frequency of approximately 60 GHz, a phase noise of –94 dBc/Hz was measured.

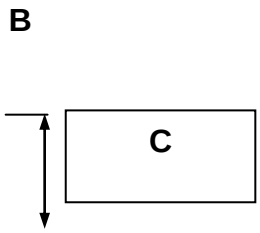


Fig. 6. Measured output spectrum at 59.9 GHz, L: phase noise V_{dd} = 1.2V and V_{tune} = 2V.

The measured phase noise at 1 MHz versus tuning range is shown in Fig. 7. Within the full tuning range, the phase noise is less than -85 dBc/Hz, which is sufficient for many applications.

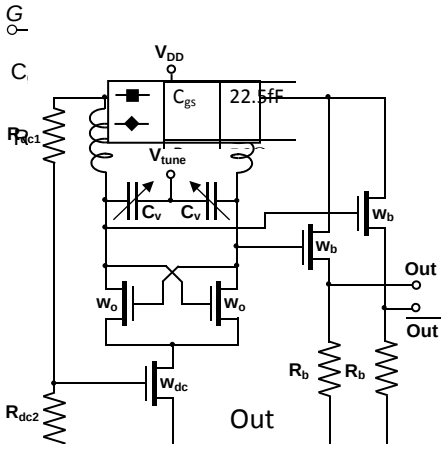


Fig. 7. Measured phase noise at 1 MHz offset versus tuning voltage.

A 60 GHz SOI CMOS VCO has been presented in this paper. Despite this high oscillation frequency, which to the best knowledge of the authors is the highest reported to date for a CMOS oscillator, an excellent tuning range of higher than 10% has been achieved with conventional MOSFET varactors. Due to the high tuning range, the circuit allows a compensation of strong process variations, which is important for aggressively scaled CMOS technologies. Furthermore, the circuit has low phase noise, a compact size and a moderate power consumption. Thus, the VCO is well suited for commercial applications operating in accordance to future WLAN and optical transceivers.

TRANSISTOR SIZING:

For a simple CMOS inverter, both the static and dynamic characteristics of the circuit were dependent on the transistor properties and V_t. It is therefore clear that designers must take care to control these parameters to ensurethat circuits work well. In practice, there is only a limited amount of control available to the designer. The threshold voltage cannot (or should not) be modified at will by the designer. V_t is strongly dependent on the doping of the substrate material, the thickness of the gate oxide and the potential in the substrate. The doping in the substrate and the oxide thickness are physical parameters that are determined by the process designers, hence the circuit designer is unable to modify them. The substrate potential can be varied at will by the designer, and this will influence V_t via a mechanism known as the body effect. It is unwise to tamper with the substrate potential, and for digital design the bulk connection is always shorted to V_{SS} for NMOS and V_{DD} for PMOS.since the carrier mobility ($\lceil_n$ or $\lceil_p$) is a fixed property of the semiconductor (affected by process), $\sum_{ox}$ is a fixed physical parameter and t_{ox} is set by process. The designer is therefore left with the two dimensional parameters, W and L. In principle both W and L can be varied at will by the chip designer, provided they stay within the design rules for the minimum size of any feature. However, the smaller L, the larger and smaller the gate capacitance, hence the quicker the circuit, so with few exceptions designers always use the smallest possible L available in a process. For example, in a 0.1 $\lceil$m process the gate length will be 0.1 $\lceil$m for virtually every single transistor on the chip. To conclude, the only parameter that the circuit designer can manipulate at will is the transistor gate width, W, and much of the following discussion will deal with the effect of modifying W and making the best selection.

IMPORTANT QUESTIONS

PART A

1. What are the issues to be considered for circuit characterization and performance estimation?
2. Give the formula for resistance of a uniform slab of conducting material.
3. What are the factors to be considered for calculating total load capacitance on the output of a CMOS gate?
4. What are the components of Power dissipation?
5. What is meant by path electrical effort?
6. Define crosstalk.
7. Define scaling.
8. What are the factors to be considered for transistor scaling?
9. Define constant voltage scaling.
10. What are the sources to be considered for design margin?

PART-B(16Marks)

1. Explain in detail about the following:

   a)Resistance estimation (8)

   b)Inductance estimation (8)

2. Explain about routing capacitance with neat diagram. (16)

3. With a neat diagram explain about power dissipation. (16)

4. Briefly explain about the following:

   a) C
   MOS transistor sizing (8)

   b) D
   esign margining (8)

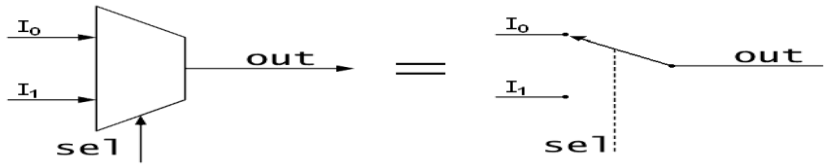
5. Explain about scaling of MOS transistor dimensions and charge sharing. (16)

UNIT IV
VLSI SYSTEM COMPONENTS CIRCUITS AND SYSTEM LEVEL PHYSICAL DESIGN

- **Multiplexers**
- **Decoders**
- **Comparators**
- **Priority encoders**
- **Shift registers**
- **Arithmetic**
- **circuits**
- **Ripple carry adders**
- **Carry look ahead adders**
- **High-speed adders**
- **Multipliers**
- **Physical design**
- **Delay modeling**
- **Cross talk**
- **Floor planning**
- **Power distribution**
- **Clock distribution**
- **Basics of CMOS testing**

MULTIPLEXER:

In electronics, a multiplexer or mux (occasionally the terms muldex or muldem are also found for a combination multiplexer-demultiplexer) is a device that performs multiplexing; it selects one of many analog or digital input signals and forwards the selected input into a single line. A multiplexer of 2^n inputs has n select lines, which are used to select which input line to send to the output.



An electronic multiplexer makes it possible for several signals to share one device or resource, for example one A/D converter or one communication line, instead of having one device per input signal.

On the other end, a demultiplexer (or demux) is a device taking a single input signal and selecting one of many data-output-lines, which is connected to the single input. A multiplexer is often used with a complementary demultiplexer on the receiving end.

An electronic multiplexer can be considered as a multiple-input, single-output switch, and a demultiplexer as a single-input, multiple-output switch. The schematic symbol for a multiplexer is an isosceles trapezoid with the longer parallel side containing the input pins and the short parallel side containing the output pin. The schematic on the right shows a 2-to-1 multiplexer on the left and an equivalent switch on the right. The sel wire connects the desired input to the output.

Functional coding for Multiplexer

```
module mux1(out,a,b,sel);

input a;

input b;

inputsel;

output out;

    wirenotsel;
```

wire y0,y1;

not g0(notsel,sel);

and g1(y0,a,notsel);

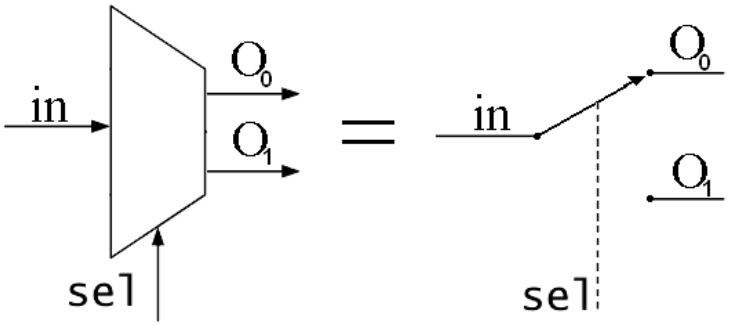
and g2(y1,b,sel);

or g3(out,y0,y1);

endmodule

Functional coding for Demultiplexer:

In telecommunications, a multiplexer is a device that combines several input information signals into one output signal, which carries several communication channels, by means of some multiplex technique. A demultiplexer is in this context a device taking a single input signal that carries many channels and separates those over multiple output signals.



In telecommunications and signal processing, an analog time division multiplexer (TDM) may take several samples of separate analogue signals and combine them into one pulse amplitude modulated (PAM) wide-band analogue signal. Alternatively, a digital TDM multiplexer may combine a limited number of constant bit rate digital data streams into one data stream of a higher data rate, by forming data frames consisting of one timeslot per channel.

In telecommunications, computer networks and digital video, a statistical multiplexer may combine several variable bit rate data streams into one constant bandwidth signal, for example by means of packet mode communication. An inverse multiplexer may utilize several communication channels for transferring one signal.

Functional coding for Demultiplexer:

Module demux(y0,y1,y2,y3,s0,s1,d);

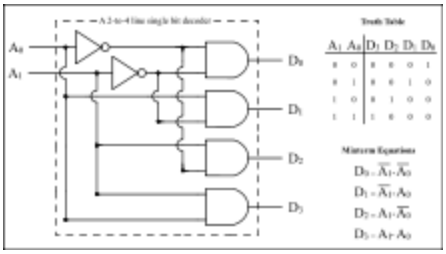
```
input s0,s1,d;
output  y0,y1,y2,y3;
not (s0bar,s0);
not (s1bar,s1);
and (y0,d,s0bar,s1bar);
and (y1,d,s0bar,s1);
and (y2,d,s0,s1bar);
and (y3,d,s0.s1);
endmodule
```

Decoder

A decoder is a device which does the reverse of an encoder, undoing the encoding so that the original information can be retrieved. The same method used to encode is usually just reversed in order to decode.

In digital electronics, a decoder can take the form of a multiple-input, multiple-output logic circuit that converts coded inputs into coded outputs, where the input and output codes are different. e.g. n-to-2^n, binary-coded decimal decoders. Enable inputs must be on for the decoder to function, otherwise its outputs assume a single "disabled" output code word. Decoding is necessary in applications such as data multiplexing, 7 segment display and memory address decoding.

The example decoder circuit would be an AND gate because the output of an AND gate is "High" (1) only when all its inputs are "High." Such output is called as "active High output". If instead of AND gate, the NAND gate is connected the output will be "Low" (0) only when all its inputs are "High". Such output is called as "active low output".



Example: A 2-to-4 Line Single Bit Decoder

A slightly more complex decoder would be the n-to-2^n type binary decoders. These type of decoders are combinational circuits that convert binary information from 'n' coded inputs to a maximum of 2^n unique outputs. We say a maximum of 2^n outputs because in case the 'n' bit coded information has unused bit combinations, the decoder may have less than 2^n outputs. We can have 2-to-4 decoder, 3-to-8 decoder or 4-to-16 decoder. We can form a 3-to-8 decoder from two 2-to-4 decoders (with enable signals).

Similarly, we can also form a 4-to-16 decoder by combining two 3-to-8 decoders. In this type of circuit design, the enable inputs of both 3-to-8 decoders originate from a 4th input, which acts as a selector between the two 3-to-8 decoders. This allows the 4th input to

enable either the top or bottom decoder, which produces outputs of D(0) through D(7) for the first decoder, and D(8) through D(15) for the second decoder.

A decoder that contains enable inputs is also known as a decoder-demultiplexer. Thus, we have a 4-to-16 decoder produced by adding a 4th input shared among both decoders, producing 16 outputs.

Functional coding for Decoder:

```
Module dec(d, a, b, e);
output[3:0] d;
input a;
input b;
input e;
wire abar,bbar,ebar;
not n1(abar,a);
not n2(bbar,b);
not n3(ebar,e);
nand n4(d[0],abar,bbar,ebar);
nand n5(d[1],abar,b,ebar);
nand n6(d[2],a,bbar,ebar);
nand n7(d[3],a,b,ebar);
endmodule
```

Digital comparator

A digital comparator or magnitude comparator is a hardware electronic device that takes two numbers as input in binary form and determines whether one number is greater than, less than or equal to the other number. Comparators are used in a central processing units (CPU) and microcontrollers. Examples of digital comparator include the CMOS 4063 and 4585 and the TTL 7485 and 74682-'89.

The analog equivalent of digital comparator is the voltage comparator. Many microcontrollers have analog comparators on some of their inputs that can be read or trigger an interrupt.

Comparator truth tables

inputs Outputs

A	B	A < B	A = B	A > B

0	0	0	1	0
0	1	1	0	0
1	0	0	0	1
1	1	0	1	0

The operation of a two bit digital comparator can be expressed as a truth table

Functional coding for COMPARATOR:

```
module program 4(A,B,AltB,AeqB,Agt B);
input[3:0]A,B;
output  AltB,AeqB,Agt B;
reg Alt B,AeqB,Agt B;
always @(A or B)
begin
Alt B<=(A<B);
Aeq B<=(A==B);
Agt B<=(A>B);
end
endmodule

module comp (x,y,z,a,b);
inputa,b;
output  x,y,z;
reg x;
reg y;
reg z;
always @(aorb)
begin
x<=1'b0;
y<1'b0;
z<=1'b0;
if (a==b)
x<="b1";
else if (a>b)
 y<=1'b1;
else if (a<b)
z<=1'b1;
end
endmodule
```

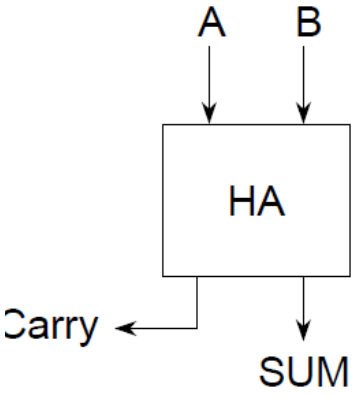
VLSI Circuit Design

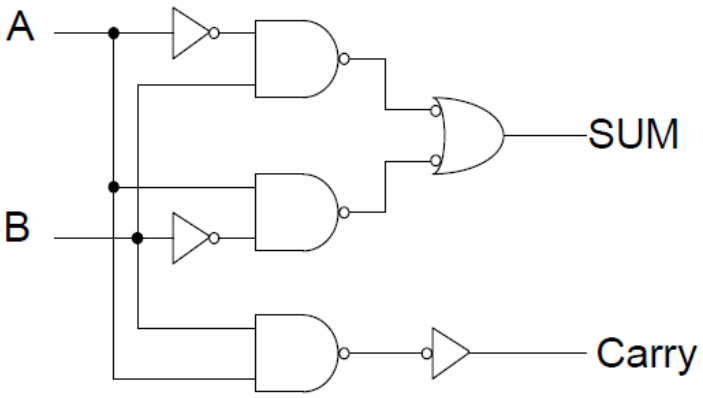
- **Data Path Operators**
  - **Adder, Parity Generator, Comparators**
  - **Counters, ALUs, Multiplier**
- **Memory Elements**
  - **SRAM, Register Files - FIFOs, LIFOs, SIPOs - Serial Access Memory**
- **Control Circuits**
  - **Finite State Machines - Logic Implementation**

Data Path - Adder

- **2`s complement Arithmetic**
- **Half Adder / Full Adder**
- **n-bit Adder**
- **Fully Parallel Addres**
- **Carry Look-Ahead Adders**
- **Carry-Completion-Detection Adders**
- **Carry Save Adders**
- **Binary Subtraction**
- **Arithmetic Overflow Detection**

Data Path - Adder - Half Adder





A	B	S	Co
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

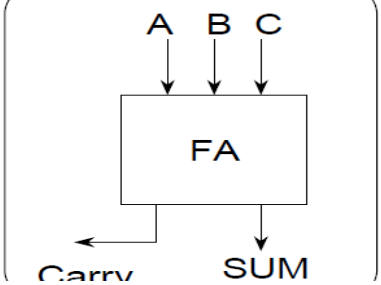
$S = A \oplus B$

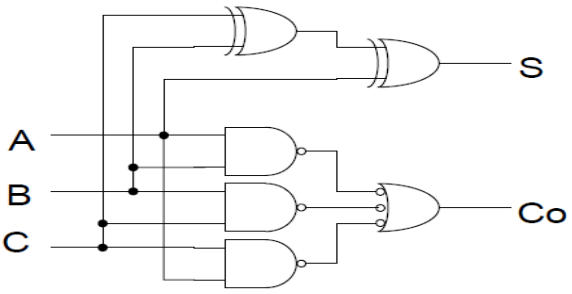
$Co = AB$

A, B - Inputs
S - Sum
Co - Carry

$S = A \oplus B \oplus C$

$Co = AB + BC + CA$

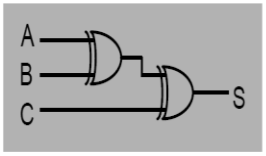


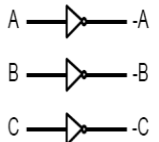


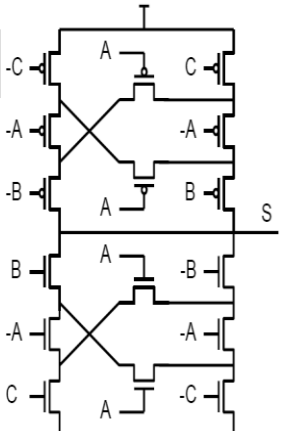
$S = A \oplus B \oplus C$

$S = C(AB + \overline{A}\,\overline{B}) + \overline{C}(A\overline{B} + \overline{A}B)$

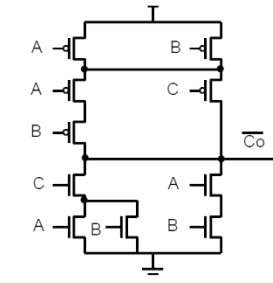
$\overline{S} = \overline{C(AB + \overline{A}\,\overline{B}) + \overline{C}(A\overline{B} + \overline{A}B)}$



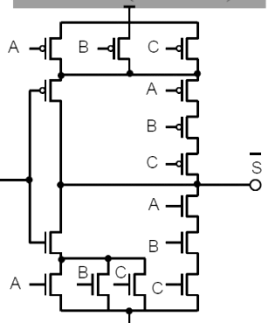


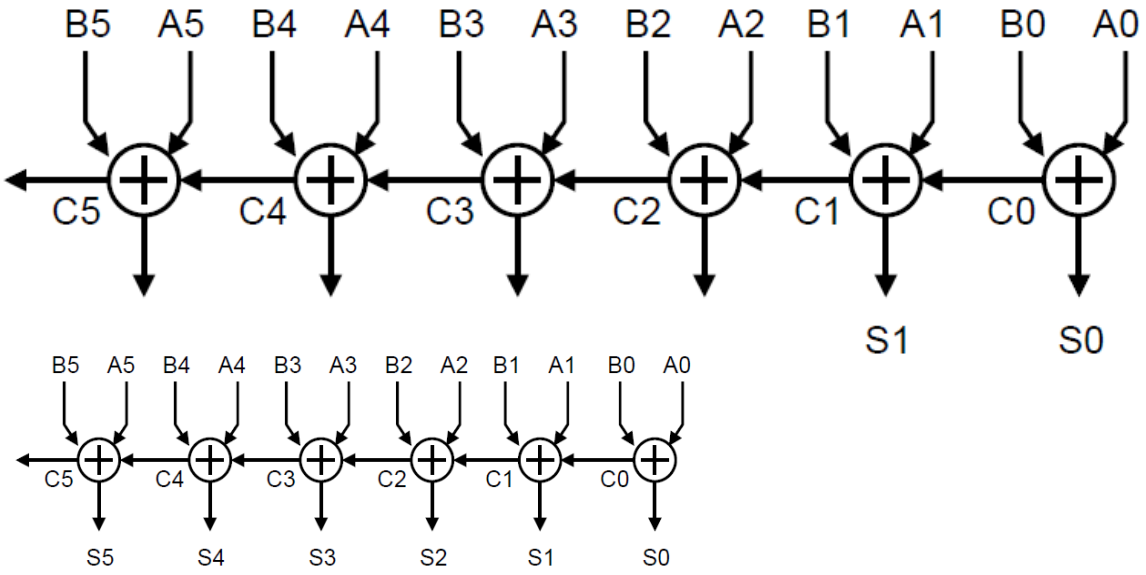


$\overline{Co} = \overline{AB + C(A+B)}$



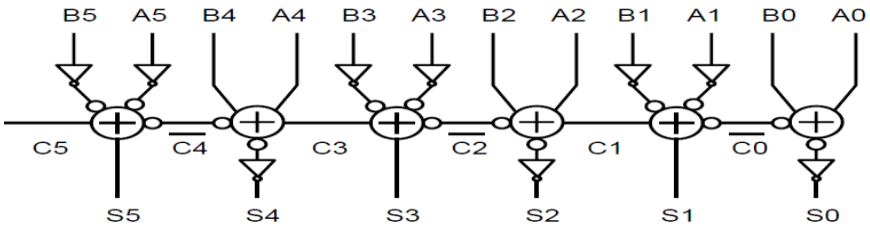
$\overline{S} = \overline{ABC + (A+B+C)\overline{Co}}$





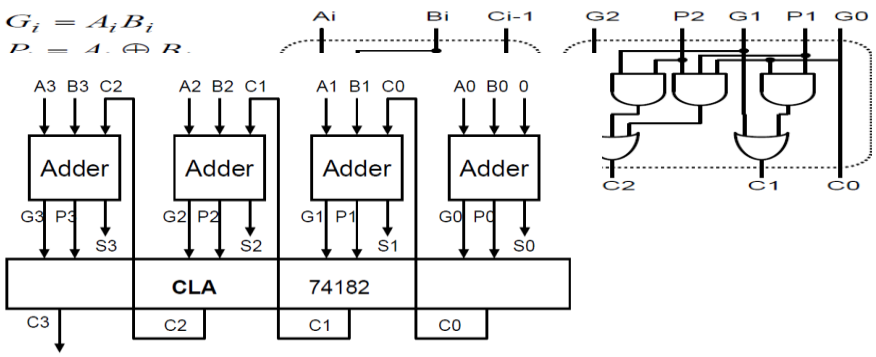
Carry is propagated or ripple through the length of the adder unit.

Data Path - Ripple Carry (Bit-Parallel) Adder



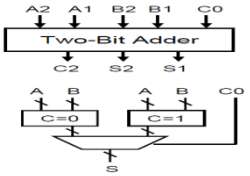
Alternate positive and negative logic.

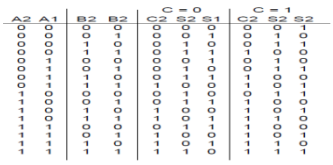
Data Path - Carry Look Ahead (CLA) Adder



High-speed adders

Data Path - Two-Bit Adder (Carry Select Adder)



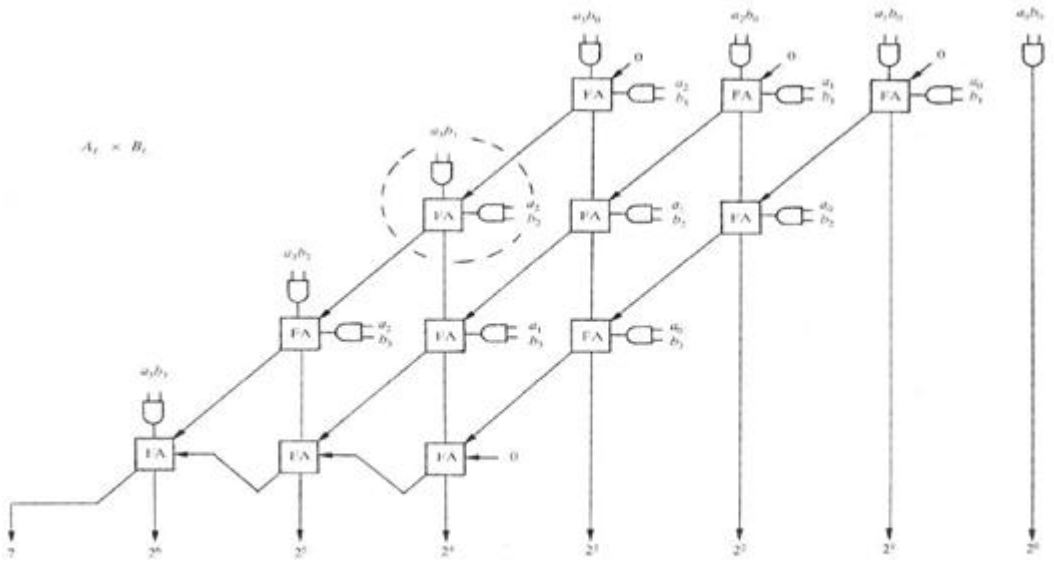


A Cascade Multiplier

This web page publishes a C++ program that generates a cascade multiplier. The design for this multiplier comes from Joseph J.F. Cavanagh's excellent book on computer arithmetic, Digital Computer Arithmetic: Design and Implementation, 1984. Sadly this book is out of print. My copy of this book was printed before books were commonly printed on acid free paper and the pages are starting to yellow. Although this book is almost twenty years old, computer arithmetic remains the same. The only thing that has changed is the density of VLSI chips. Hardware modules like the cascade multiplier, discussed here, consumed too much hardware area in some cases. Given current chip density, a 32x32 multiplier can be included in a microprocessor.

When this multiplier was implemented a large structural Verilog test case for Quick turn's Sim Server hardware accelerated behavioral simulation system. It was also fascinated by the tiling and recursive nature of the multiplier. The multiplier implemented here is not the most efficient in its use of hardware nor the fastest in terms of logic levels. Cavanagh shows how 4X4 adder slices can be combined with a look ahead carry tree (a so called Wallace tree) to build larger multipliers, which are faster than this multiplier for larger bit widths.

The figure below have been referring to as a cascade multiplier. The multiplier shown here takes two 4-bit operands (e.g., its a 4 X 4 multiplier) and generates an 8-bit result. The FA elements in the diagram are full adders. This version of the multiplier does unsigned multiplication. A slight variation on this design will perform signed integer multiplication.




```
reg [63:0] product, product_temp;

reg [31:0] multiplier_copy;
reg [63:0] multiplicand_copy;
reg negative_output;

reg [5:0] bit;
wire ready = !bit;

initial bit = 0;
initial negative_output = 0;

always @( posedge clk )

if( ready ) begin

bit = 6'd32;
product = 0;
```

```
product_temp = 0;
multiplicand_copy = (!sign || !multiplicand[31]) ?
{ 32'd0, multiplicand } :
{ 32'd0, ~multiplicand + 1'b1};
multiplier_copy = (!sign || !multiplier[31]) ?
multiplier :
~multiplier + 1'b1;

negative_output = sign &&
((multiplier[31] && !multiplicand[31])
||(!multiplier[31] && multiplicand[31]));

end
else if ( bit > 0 ) begin

if( multiplier_copy[0] == 1'b1 ) product_temp = product_temp +
multiplicand_copy;

product = (!negative_output) ?
product_temp :
~product_temp + 1'b1;

multiplier_copy = multiplier_copy >> 1;
multiplicand_copy = multiplicand_copy << 1;
bit = bit - 1'b1;

end
endmodule
```

Physical design

In integrated circuit design, physical design is a step in the standard design cycle which follows after the circuit design. At this step, circuit representations of the components (devices and interconnects) of the design are converted into geometric representations of shapes which, when manufactured in the corresponding layers of materials, will ensure the required functioning of the components. This geometric representation is called integrated circuit layout. This step is usually split into several sub-steps, which include both design and verification and validation of the layout.

Modern day Integrated Circuit (IC) design is split up into Front-end design using HDL's, Verification and Back-end Design or Physical Design. The next step after Physical Design is the Manufacturing process or Fabrication Process that is done in the Wafer Fabrication Houses. Fab-houses fabricate designs onto silicon dies which are then packaged into ICs.

Each of the phases mentioned above have Design Flows associated with them. These Design Flows lay down the process and guide-lines/framework for that phase. Physical Design flow uses the technology libraries that are provided by the fabrication houses. These technology files provide information regarding the type of Silicon wafer used, the standard-cells used, the layout rules, etc.

Technologies are commonly classified according to minimal feature size. Standard sizes, in the order of miniaturization, are 2µm, 1µm , 0.5µm , 0.35µm, 0.25µm, 180nm, 130nm, 90nm, 65nm, 45nm, 28nm, 22nm, 18nm... They may be also classified according to major manufacturing approaches: n-Well process, twin-well process, SOI process, etc

FLOOR PLANNING:

The RTL of the chip is assigned to gross regions of the chip, input/output (I/O) pins are assigned and large objects (arrays, cores, etc.) are placed.

- **Logic synthesis:**

  The RTL is mapped into a gate-level netlist in the target technology of the chip.

- **Placement:**

  The gates in the netlist are assigned to non overlapping locations on the die area.

- **Logic/placement refinement:**

  Iterative logical and placement transformations to close performance and power constraints.

- **Clock insertion:**

  Clock signal wiring is (commonly, clock trees) introduced into the design.

- **Routing:**

  The wires that connect the gates in the netlist are added.

- **Post wiring optimization:**

  Performance (timing closure), noise (signal integrity), and yield (Design for manufacturability) violations are removed.

- **Design for manufacturability:**

  The design is modified, where possible, to make it as easy and efficient as possible to produce. This is achieved by adding extra vias or adding dummy metal/diffusion/poly layers wherever possible while complying to the design rules set by the foundry.

- **Final checking:**

  Since errors are expensive, time consuming and hard to spot, extensive error checking is the rule, making sure the mapping to logic was done correctly, and checking that the manufacturing rules were followed faithfully.

- **Tape out and mask generation:**

  The design data is turned into photo masks in mask data preparation

Delay Specification

•No delay specified then default is 0
•If one delay, then used for all transitions

•If two delays, then refers to rise and fall in order. Turn-off is min of these two.
•If three delays, then refers to rise, fall and turn-off in order.

Delay Specification (Examples)

•and #(5) a1(out,i1,i2);
•and #(4,6) a2(out,i1,i2);
•bufif0 #(3,4,5) b1 (out,i1,i2);

Additional Control

•min/typ/max delays for each of these values
•and #(4:5:6) a1 (out,i1,i2);
•and #(4:5:6, 5:6:7) a2 (out,i1,i2);
•and #(4:5:6, 5:6:7, 7:8:9) a3(out,i1,i2);
•+mindelays, +maxdelays, +typdelaysare compile options
•verilogtest.v +typdelays

Simulators

•Pessimistic in delay calculation
–For eg. 0, 1 or z to x is the minimum of rise, fall and turnoff delays.
–Tend to make the signal as undefined as possible
–This is necessary to simulate the worst case conditions

Clock distribution

- **Goals:**
  - **deliver clock to all memory elements with acceptable skew;**
  - **deliver clock edges with acceptable sharpness.**
- **Clocking network design is one of the greatest challenges in the design of a large chip.**

Clock distribution tree

- **Clocks are generally distributed via wiring trees.**
- **Want to use low-resistance interconnect to minimize delay.**

- **Use multiple drivers to distribute driver requirements—use optimal sizing principles to design buffers.**

Clock lines can create significant cross

Floor planning

- **Develop a wiring plan. Think about how layers will be used to distribute important wires.**
- **Sweep small components into larger blocks. A floor plan with a single NAND gate in the middle will be hard to work with.**
- **Design wiring that looks simple. If it looks complicated, it is complicated.**
- **Design planar wiring. Planarity is the essence of simplicity. It isn't always possible, but do it where feasible (and where it doesn't introduce unacceptable delay).**
- **Draw separate wiring plans for power and clocking. These are important design tasks which should be tackled early.**

Structure of a typical package

Boundary scan

- **Boundary scan is a technique for testing chips on boards. Pads on chips are arranged into a scan chain that can be used to observe and control pins of all chips.**
- **Requires some control circuitry on pads along with an on-chip controller and boundary-scan-mode control pins.**

IMPORTANT QUESTIONS

PART-A (2 Marks)

1. Write the difference between encoder and priority encoder.

2. Draw the CMOS implementation of 4-to-1 MUX using transmission gates.

3. Give the verilog coding for 4-bit magnitude comparator.

4. Draw the wheel floor plan

5. What is meant by clock distribution?

6. Design a circuit for finding the 9's compliment of a BCD number using 4-bit binary adder and some external logic gates

7. What is physical verification?

8. Mention the levels at which testing of a chip can be done

9. What are the approaches in design for testability?

10. What is known as boundary scan register?

11. Design a set of CMOS gates to implement the sum function.

PART-B (16 Marks)

1. Give the design procedure for 8 bit carry look ahead adder. (16)

2. Design a multiplier for the given sequence: (16)

3. Draw the basic physical design for the inverter AND, OR and half-adder. (16)

4. Explain in detail about manufacturing of test principles. (16)

5. Explain the concept of clock distribution and power distribution. (16)

UNIT V
FPGA AND VERILOG HARDWARE DESCRIPTION LANGUAGE

- **Introduction to FPGA**
- **Xilinx FPGA**
- **Xilinx 2000**
- **Xilinx 3000**
- **Overview of Digital Design with Verilog HDL**
- **Hierarchical modeling concepts**
- **Modules and Port definitions**
- **Gate level modeling**
- **Data flow modeling**
- **Behavioral modeling**

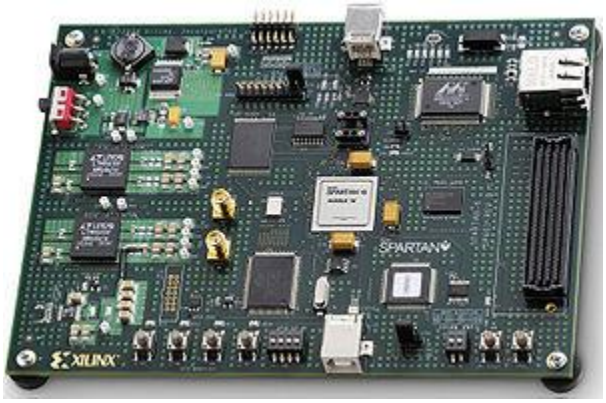
INTRODUCTION:

FPGA" redirects here. It is not to be confused with Flip-chip pin grid array.A Field-programmable Gate Array (FPGA) is an integrated circuit designed to be configured by the customer or designer after manufacturing—hence "field-programmable". The FPGA

configuration is generally specified using a hardware description language (HDL), similar to that used for an application-specific integrated circuit (ASIC) (circuit diagrams were previously used to specify the configuration, as they were for ASICs, but this is increasingly rare). FPGAs can be used to implement any logical function that an ASIC could perform. The ability to update the functionality after shipping, partial re-configuration of the portion of the design[1] and the low non-recurring engineering costs relative to an ASIC design (notwithstanding the generally higher unit cost), offer advantages for many applications

FPGA" redirects here. It is not to be confused with Flip-chip pin grid array.

An Altera Stratix IV GX FPGA



An example of an FPGA programming/evaluation board

FPGAs contain programmable logic components called "logic blocks", and a hierarchy of reconfigurable interconnects that allow the blocks to be "wired together"—somewhat like many (changeable) logic gates that can be inter-wired in (many) different configurations. Logic blocks can be configured to perform complex combinational functions, or merely simple logic gates like AND and XOR. In most FPGAs, the logic blocks also include memory elements, which may be simple flip-flops or more complete blocks of memory.

In addition to digital functions, some FPGAs have analog features. The most common analog feature is programmable slew rate and drive strength on each output pin, allowing the engineer to set slow rates on lightly loaded pins that would otherwise ring

unacceptably, and to set stronger, faster rates on heavily loaded pins on high-speed channels that would otherwise run too slow. Another relatively common analog feature is differential comparators on input pins designed to be connected to differential signaling channels. A few "mixed signal FPGAs" have integrated peripheral Analog-to-Digital Converters (ADCs) and Digital-to-Analog Converters (DACs) with analog signal conditioning blocks allowing them to operate as a system-on-a-chip.[5] Such devices blur the line between an FPGA, which carries digital ones and zeros on its internal programmable interconnect fabric, and field-programmable analog array (FPAA), which carries analog values on its internal programmable interconnect fabric.

History

The FPGA industry sprouted from programmable read-only memory (PROM) and programmable logic devices (PLDs). PROMs and PLDs both had the option of being programmed in batches in a factory or in the field (field programmable), however programmable logic was hard-wired between logic gates.

In the late 1980s the Naval Surface Warfare Department funded an experiment proposed by Steve Casselman to develop a computer that would implement 600,000 reprogrammable gates. Casselman was successful and a patent related to the system was issued in 1992.

Some of the industry's foundational concepts and technologies for programmable logic arrays, gates, and logic blocks are founded in patents awarded to David W. Page and LuVerne R. Peterson in 1985.

Xilinx Co-Founders, Ross Freeman and Bernard Vonderschmitt, invented the first commercially viable field programmable gate array in 1985 – the XC2064. The XC2064 had programmable gates and programmable interconnects between gates, the beginnings of a new technology and market. The XC2064 boasted a mere 64 configurable logic blocks (CLBs), with two 3-input lookup tables (LUTs). More than 20 years later, Freeman was entered into the National Inventors Hall of Fame for his invention.

Xilinx continued unchallenged and quickly growing from 1985 to the mid-1990s, when competitors sprouted up, eroding significant market-share. By 1993, Actel was serving about 18 percent of the market.

The 1990s were an explosive period of time for FPGAs, both in sophistication and the volume of production. In the early 1990s, FPGAs were primarily used in telecommunications and networking. By the end of the decade, FPGAs found their way into consumer, automotive, and industrial applications.

FPGAs got a glimpse of fame in 1997, when Adrian Thompson, a researcher working at the University of Sussex, merged genetic algorithm technology and FPGAs to create a sound recognition device. Thomson's algorithm configured an array of 10 x 10 cells in a Xilinx FPGA chip to discriminate between two tones, utilizing analogue features of the digital chip. The application of genetic algorithms to the configuration of devices like FPGA's is now referred to as Evolvable hardware.

APPLICATIONS:

Applications of FPGAs include digital signal processing, software-defined radio, aerospace and defense systems, ASIC prototyping, medical imaging, computer vision, speech recognition, cryptography, bioinformatics, computer hardware emulation, radio astronomy, metal detection and a growing range of other areas.

FPGAs originally began as competitors to CPLDs and competed in a similar space, that of glue logic for PCBs. As their size, capabilities, and speed increased, they began to take over larger and larger functions to the state where some are now marketed as full systems on chips (SoC). Particularly with the introduction of dedicated multipliers into FPGA architectures in the late 1990s, applications which had traditionally been the sole reserve of DSPs began to incorporate FPGAs instead.

FPGAs especially find applications in any area or algorithm that can make use of the massive parallelism offered by their architecture. One such area is code breaking, in particular brute-force attack, of cryptographic algorithms.

FPGAs are increasingly used in conventional high performance computing applications where computational kernels such as FFT or Convolution are performed on the FPGA instead of a microprocessor.

The inherent parallelism of the logic resources on an FPGA allows for considerable computational throughput even at a low MHz clock rates. The flexibility of the FPGA allows for even higher performance by trading off precision and range in the number format for an increased number of parallel arithmetic units. This has driven a new type of processing called reconfigurable computing, where time intensive tasks are offloaded from software to FPGAs.

The adoption of FPGAs in high performance computing is currently limited by the complexity of FPGA design compared to conventional software and the turn-around times of current design tools.

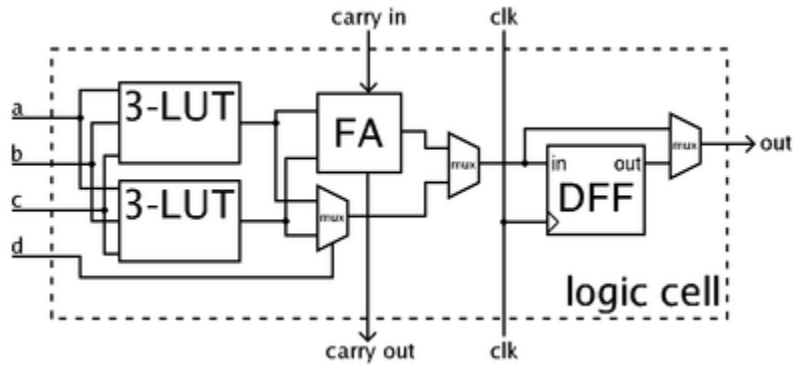
Traditionally, FPGAs have been reserved for specific vertical applications where the volume of production is small. For these low-volume applications, the premium that companies pay in hardware costs per unit for a programmable chip is more affordable than the development resources spent on creating an ASIC for a low-volume application. Today, new cost and performance dynamics have broadened the range of viable applications.

ARCHITECTURE:

The most common FPGA architecture consists of an array of logic blocks (called Configurable Logic Block, CLB, or Logic Array Block, LAB, depending on vendor), I/O pads, and routing channels. Generally, all the routing channels have the same width (number of wires). Multiple I/O pads may fit into the height of one row or the width of one column in the array.

An application circuit must be mapped into an FPGA with adequate resources. While the number of CLBs/LABs and I/Os required is easily determined from the design, the number of routing tracks needed may vary considerably even among designs with the same amount of logic. For example, a crossbar switch requires much more routing than a systolic array with the same gate count. Since unused routing tracks increase the cost (and decrease the performance) of the part without providing any benefit, FPGA manufacturers try to provide just enough tracks so that most designs that will fit in terms of LUTs and IOs can be routed. This is determined by estimates such as those derived from Rent's rule or by experiments with existing designs.

In general, a logic block (CLB or LAB) consists of a few logical cells (called ALM, LE, Slice etc). A typical cell consists of a 4-input Lookup table (LUT), a Full adder (FA) and a D-type flip-flop, as shown below. The LUT are in this figure split into two 3-input LUTs. In normal mode those are combined into a 4-input LUT through the left mux. In arithmetic mode, their outputs are fed to the FA. The selection of mode are programmed into the middle mux. The output can be either synchronous or asynchronous, depending on the programming of the mux to the right, in the figure example. In practice, entire or parts of the FA are put as functions into the LUTs in order to save space.



Simplified example illustration of a logic cell

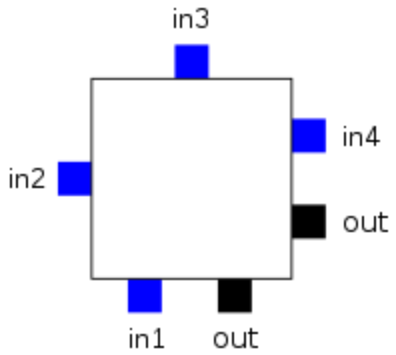
ALMs and Slices usually contain 2 or 4 structures similar to the example figure, with some shared signals.

CLBs/LABs typically contain a few ALMs/LEs/Slices.

In recent years, manufacturers have started moving to 6-input LUTs in their high performance parts, claiming increased performance.

Since clock signals (and often other high-fan-out signals) are normally routed via special- purpose dedicated routing networks in commercial FPGAs, they and other signals are separately managed.

For this example architecture, the locations of the FPGA logic block pins are shown below.



Logic Block Pin Locations

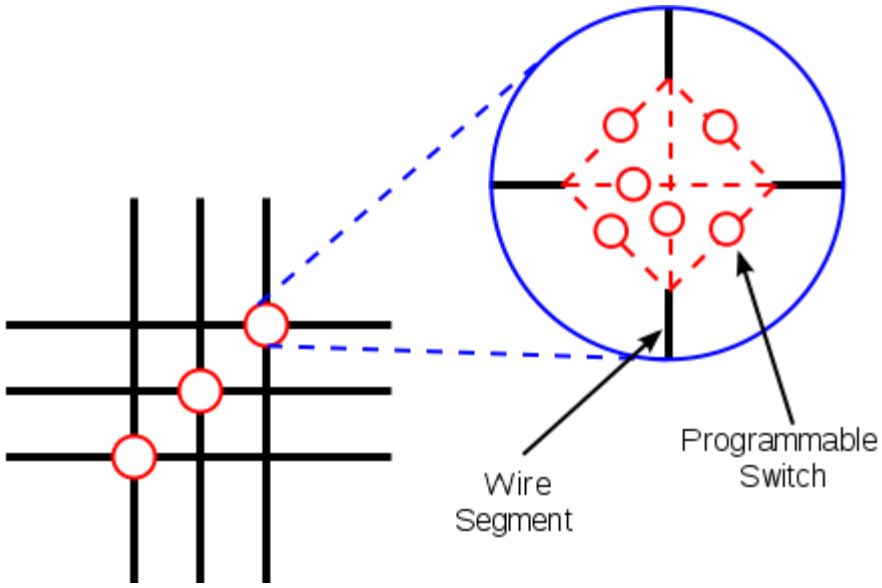
Each input is accessible from one side of the logic block, while the output pin can connect to routing wires in both the channel to the right and the channel below the logic block.

Each logic block output pin can connect to any of the wiring segments in the channels adjacent to it.

Similarly, an I/O pad can connect to any one of the wiring segments in the channel adjacent to it. For example, an I/O pad at the top of the chip can connect to any of the W wires (where W is the channel width) in the horizontal channel immediately below it.

Generally, the FPGA routing is unsegmented. That is, each wiring segment spans only one logic block before it terminates in a switch box. By turning on some of the programmable switches within a switch box, longer paths can be constructed. For higher speed interconnect, some FPGA architectures use longer routing lines that span multiple logic blocks.

Whenever a vertical and a horizontal channel intersect, there is a switch box. In this architecture, when a wire enters a switch box, there are three programmable switches that allow it to connect to three other wires in adjacent channel segments. The pattern, or topology, of switches used in this architecture is the planar or domain-based switch box topology. In this switch box topology, a wire in track number one connects only to wires in track number one in adjacent channel segments, wires in track number 2 connect only to other wires in track number 2 and so on. The figure below illustrates the connections in a switch box.



Switch box topology

Modern FPGA families expand upon the above capabilities to include higher level functionality fixed into the silicon. Having these common functions embedded into the silicon reduces the area required and gives those functions increased speed compared to building them from primitives. Examples of these include multipliers, generic DSP blocks, embedded processors, high speed IO logic and embedded memories.

FPGAs are also widely used for systems validation including pre-silicon validation, post-silicon validation, and firmware development. This allows chip companies to validate their design before the chip is produced in the factory, reducing the time-to-market.

FPGA design and programming

To define the behavior of the FPGA, the user provides a hardware description language (HDL) or a schematic design. The HDL form is more suited to work with large structures because it's possible to just specify them numerically rather than having to draw every piece by hand. However, schematic entry can allow for easier visualisation of a design.

Then, using an electronic design automation tool, a technology-mapped netlist is generated. The netlist can then be fitted to the actual FPGA architecture using a process called place-and-route, usually performed by the FPGA company's proprietary place-and-route software. The user will validate the map, place and route results via timing analysis, simulation, and other verification methodologies. Once the design and validation process is complete, the binary file generated (also using the FPGA company's proprietary software) is used to (re)configure the FPGA.

Going from schematic/HDL source files to actual configuration: The source files are fed to a software suite from the FPGA/CPLD vendor that through different steps will produce a file. This file is then transferred to the FPGA/CPLD via a serial interface (JTAG) or to an external memory device like an EEPROM.

The most common HDLs are VHDL and Verilog, although in an attempt to reduce the complexity of designing in HDLs, which have been compared to the equivalent of assembly languages, there are moves to raise the abstraction level through the introduction of alternative languages. National Instrument's LabVIEW graphical programming language (sometimes referred to as "G") has an FPGA add-in module available to target and program FPGA hardware. The LabVIEW approach drastically simplifies the FPGA programming process.

To simplify the design of complex systems in FPGAs, there exist libraries of predefined complex functions and circuits that have been tested and optimized to speed up the design process. These predefined circuits are commonly called IP cores, and are available from FPGA vendors and third-party IP suppliers (rarely free, and typically released under proprietary licenses). Other predefined circuits are available from developer communities such as Open Cores (typically released under free and open source licenses such as the GPL, BSD or similar license), and other sources.

In a typical design flow, an FPGA application developer will simulate the design at multiple stages throughout the design process. Initially the RTL description in VHDL or Verilog is simulated by creating test benches to simulate the system and observe results. Then, after the synthesis engine has mapped the design to a netlist, the netlist is translated to a gate level description where simulation is repeated to confirm the synthesis proceeded without errors. Finally the design is laid out in the FPGA at which point propagation delays can be added and the simulation run again with these values back-annotated onto the netlist.

Basic process technology types

- **SRAM - based on static memory technology. In-system programmable and re-programmable. Requires external boot devices. CMOS.**
- **Antifuse - One-time programmable. CMOS.**
- **PROM - Programmable Read-Only Memory technology. One-time programmable because of plastic packaging.**
- **EPROM - Erasable Programmable Read-Only Memory technology. One-time programmable but with window, can be erased with ultraviolet (UV) light. CMOS.**
- **EEPROM - Electrically Erasable Programmable Read-Only Memory technology. Can be erased, even in plastic packages. Some but not all EEPROM devices can be in-system programmed. CMOS.**
- **Flash - Flash-erase EPROM technology. Can be erased, even in plastic packages. Some but not all flash devices can be in-system programmed. Usually, a flash cell is smaller than an equivalent EEPROM cell and is therefore less expensive to manufacture. CMOS.**
- **Fuse - One-time programmable. Bipolar.**

Major manufacturers

Xilinx and Altera are the current FPGA market leaders and long-time industry rivals. Together, they control over 80 percent of the market with Xilinx alone representing over 50 percent.

Both Xilinx and Altera provide free Windows and Linux design software.

Other competitors include Lattice Semiconductor (SRAM based with integrated configuration Flash, instant-on, low power, live reconfiguration), Actel (Antifuse, flash-based, mixed-signal), Silicon Blue Technologies (extremely low power SRAM-based FPGAs with option integrated nonvolatile configuration memory), Achronix (RAM based, 1.5 GHz fabric speed) who will be building their chips on Intel's state-of-the art 22nm process[34], and Quick Logic (handheld focused CSSP, no general purpose FPGAs).

In March 2010, Tabula announced their new FPGA technology that uses time-multiplexed logic and interconnect for greater potential cost savings for high-density applications.

Overview of Digital Design with Verilog HDL -Evolution of Computer-Aided Digital Design

Digital circuit design has evolved rapidly over the last 25 years. The earliest digital circuits were designed with vacuum tubes and transistors. Integrated circuits were then invented where logic gates were placed on a single chip. The first integrated circuit (IC) chips were SSI (Small Scale Integration) chips where the gate count was very small. As technologies became sophisticated, designers were able to place circuits with hundreds of gates on a chip. These chips were called MSI (Medium Scale Integration) chips. With the advent of LSI (Large Scale Integration), designers could put thousands of gates on a single chip. At this point, design processes started getting very complicated, and designers felt the need to automate these processes. Electronic Design Automation(EDA)techniques began to evolve.

Chip designers began to use circuit and logic simulation techniques to verify the functionality of building blocks of the order of about 100 transistors. The circuits were still tested on the bread board, and the layout was done on paper or by hand on a graphic computer terminal. With the advent of VLSI (Very Large Scale Integration) technology, designers could design single chips with more than 100,000 transistors. Because of the complexity of these circuits, it was not possible to verify these circuits on a breadboard. Computer aided techniques became critical for verification and design of VLSI digital circuits.

Computer programs to do automatic placement and routing of circuit layouts also became popular. The designers were now building gate-level digital circuits manually on graphic terminals. They would build small building blocks and then derive higher-1. The earlier edition of the book used the term CAD tools. Technically, the term Computer-Aided Design(CAD) tools refers to back-end tools that perform functions related to place and route, and layout of the chip. The term Computer-Aided Engineering (CAE) tools refers to tools that are used for front-end processes such HDL simulation, logic synthesis, and timing analysis.

Designers used the terms CAD and CAE interchangeably. Today, the term Electronic Design Automation is used for both CAD and CAE. For the sake of simplicity, in this book, we will refer to all design tools as EDA tools

level blocks from them. This process would continue until they had built the top-level block. Logic simulators came into existence to verify the functionality of these circuits before they were fabricated on chip. As designs got larger and more complex, logic simulation assumed an important role in the design process. Designers could iron out functional bugs in the architecture before the chip was designed further.

Emergence of HDLs

For a long time, programming languages such as FORTRAN, Pascal, and C were being used to describe computer programs that were sequential in nature. Similarly, in the digital design field, designers felt the need for a standard language to describe digital circuits. Thus, Hardware Description Languages(HDLs) came into existence. HDLs allowed the designers to model the concurrency of processes found in hardware elements. Hardware description languages such as Verilog

HDL and VHDL

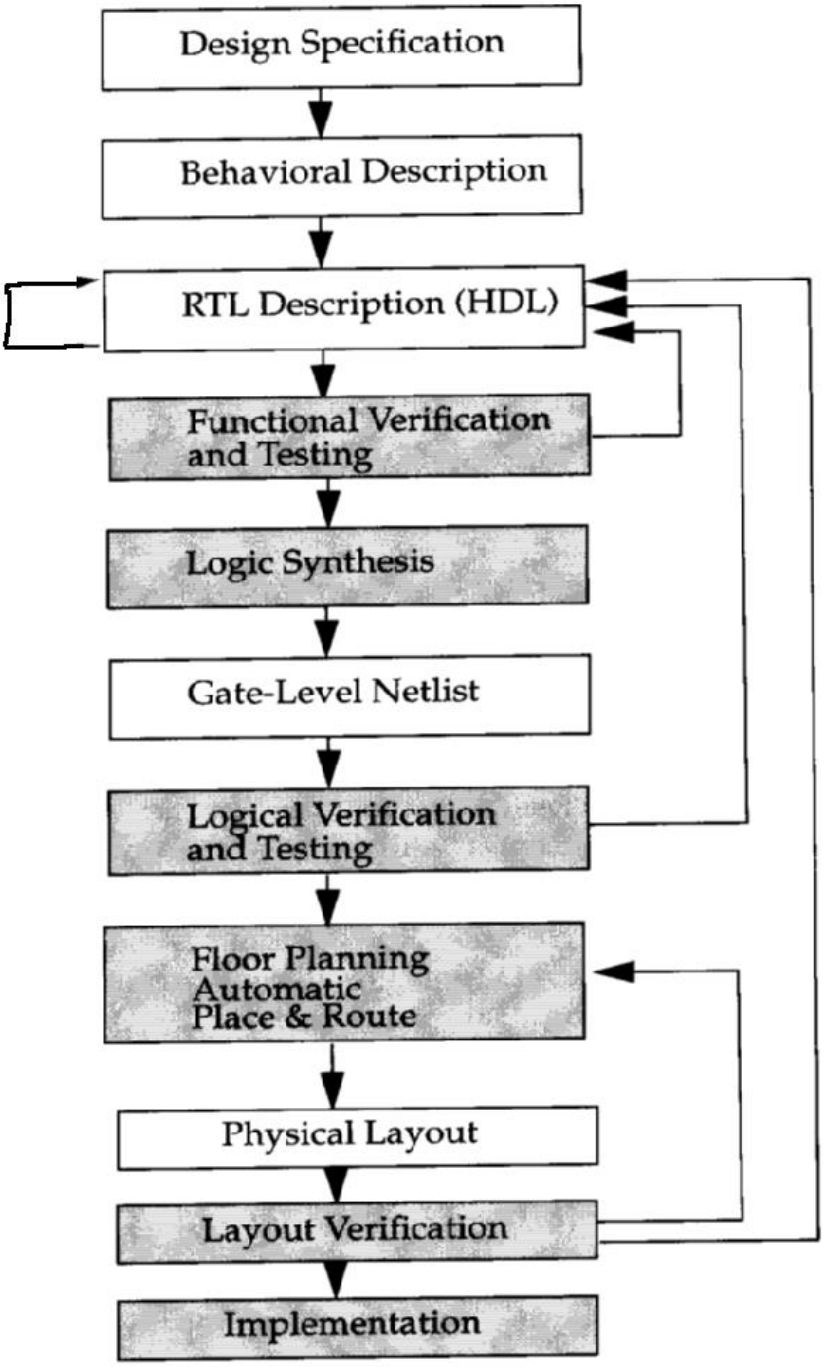
Verilog HDL originated in 1983 at Gateway Design Automation. Later, VHDL was developed under contract from DARPA. Using both Verilog and VHDL simulators to simulate large digital circuits quickly gained acceptance from designers. Even though HDLs were popular for logic verification, designers had to manually translate the HDL-based design into a schematic circuit with interconnections between gates. The advent of logic synthesis in the late 1980s changed the design methodology radically. Digital circuits could be described at a register transfer level (RTL) by use of an HDL. Thus, the designer had to specify how the data flows between registers and how the design processes the data. The details of gates and their interconnections to implement the circuit were automatically extracted by logic synthesis tools from the RTL description. Thus, logic synthesis pushed the HDLs into the forefront of digital design. Designers no longer had to manually place gates to build digital circuits. They could describe complex circuits at an abstract level in terms of functionality and data flow by designing those circuits in HDLs. Logic synthesis tools would implement the specified functionality in terms of gates and gate interconnections. HDLs also began to be used for system-level design. HDLs were used for simulation of system boards, interconnect buses, FPGAs (Field Programmable Gate Arrays), and PALs (Programmable Array Logic). A common approach is to design each IC chip, using an HDL, and then verify system functionality via simulation. Today, Verilog HDL is an accepted IEEE standard. In 1995, the original standard IEEE 1364-1995 was approved. IEEE 1364-2001 is the latest Verilog HDL standard that made significant improvements to the original standard.

Overview of Digital Design with Verilog HDL

Typical Design Flow

A typical design flow for designing VLSI IC circuits is shown in Figure 1-1.

Unshaded blocks show the level of design representation; shaded blocks show processes in the design



flow.

Figure Typical Design Flow Behavioral Description RTL Description (HDL) Gate-Level Netlist Logic Synthesis/Floor Planning Physical Layout Automatic

Implementation Layout Verification Place and Route Functional Verification Logical Verification and Testing and Testing Design Specification Timing Verification

Verilog HDL: A Guide to Digital Design and Synthesis

The design flow shown in Figure is typically used by designers who use HDLs. In any design, specifications are written first. Specifications describe abstractly the functionality, interface, and overall architecture of the digital circuit to be designed. At this point, the architects do not need to think about how they will implement this circuit. A behavioral description is then created to analyze the design in terms of functionality, performance, and compliance to standards, and other high-level issues. Behavioral descriptions are often written with HDLs.

The behavioral description is manually converted to an RTL description in an HDL. The designer has to describe the data flow that will implement the desired digital circuit. From this point onward, the design process is done with the assistance of EDA tools

Logic synthesis tools convert the RTL description to a gate-level netlist. A gate-level netlist is a description of the circuit in terms of gates and connections between them. Logic synthesis tools ensure that the gate-level netlist meets timing, area, and power specifications. The gate-level netlist is input to an Automatic Place and Route tool, which creates a layout. The layout is verified and then fabricated on a chip.

Thus, most digital design activity is concentrated on manually optimizing the RTL description of the circuit. After the RTL description is frozen, EDA tools are available to assist the designer in further processes. Designing at the RTL level has shrunk the design cycle times from years to a few months. It is also possible to do many design iterations in a short period of time.

Behavioral synthesis tools have begun to emerge recently. These tools can create RTL descriptions from a behavioral or algorithmic description of the circuit. As these tools mature, digital circuit design will become similar to high-level computer programming. Designers will simply implement the algorithm in an HDL at a very abstract level. EDA tools will help the designer convert the behavioral description to a final IC chip. It is important to note that, although EDA tools are available to automate the processes and cut design cycle times, the designer is still the person who control show the tool will perform. EDA tools are also susceptible to the "GIGO: Garbage In Garbage Out" phenomenon. If used improperly, EDA tools will lead to inefficient designs.

Thus, the designer still needs to understand the nuances of design methodologies, using EDA tools to obtain an optimized design.

New EDA tools have emerged to simulate behavioral descriptions of circuits. These tools combine the powerful concepts from HDLs and object oriented languages such as C++. These tools can be used instead of writing behavioral descriptions in Verilog HDL.

Importance of HDLs

HDLs have many advantages compared to traditional schematic-based design.

• Designs can be described at a very abstract level by use of HDLs. Designers can write their RTL description without choosing a specific fabrication technology. Logic synthesis tools can automatically convert the design to any fabrication technology. If a new

technology emerges, designers do not need to redesign their circuit. They simply input the RTL description to the logic synthesis tool and create a new gate-level netlist, using the new fabrication technology. The logic synthesis tool will optimize the circuit in area and timing for the new technology.

•By describing designs in HDLs, functional verification of the design can be done early in the design cycle. Since designers work at the RTL level, they can optimize and modify the RTL description until it meets the desired functionality. Most design bugs are eliminated at this point. This cuts down design cycle time significantly because the probability of hitting a functional bug at a later time in the gate-level netlist or physical layout is minimized.

• Designing with HDLs is analogous to computer programming. A textual description with comments is an easier way to develop and debug circuits. This also provides a concise representation of the design, compared to gate-level schematics. Gate-level schematics are almost incomprehensible for very complex designs. HDL-based design is here to stay.

With rapidly increasing complexities of digital circuits and increasingly sophisticated EDA tools, HDLs are now the dominant method for large digital designs. No digital circuit designer can afford to ignore HDL-based design.

Popularity of Verilog HDL:

Verilog HDL has evolved as a standard hardware description language. Verilog HDL offers many useful features for hardware design.

•Verilog HDL is a general-purpose hardware description language that is easy to learn and easy to use. It is similar in syntax to the C programming language. Designers with C programming experience will find it easy to learn Verilog HDL.

New tools and languages focused on verification have emerged in the past few years. These languages are better suited for functional verification. However, for logic design, HDLs continue as the preferred choice.

Verilog HDL: A Guide to Digital Design and Synthesis

•Verilog HDL allows different levels of abstraction to be mixed in the same model. Thus, a designer can define a hardware model in terms of switches, gates, RTL, or behavioral code. Also, a designer needs to learn only one language for stimulus and hierarchical design.

•Most popular logic synthesis tools support Verilog HDL. This makes it the language of choice for designers.

•All fabrication vendors provide Verilog HDL libraries for post logic synthesis simulation. Thus, designing a chip in Verilog HDL allows the widest choice of vendors.

•The Programming Language Interface (PLI) is a powerful feature that allows the user to write custom C code to interact with the internal data structures of Verilog. Designers can customize a Verilog HDL simulator to their needs with the PLI.

Trends in HDLs:

The speed and complexity of digital circuits have increased rapidly. Designers have responded by designing at higher levels of abstraction. Designers have to think only in terms of functionality. EDA tools take care of the implementation details. With

designer assistance, EDA tools have become sophisticated enough to achieve a close to-optimum implementation.

The most popular trend currently is to design in HDL at an RTL level, because logic synthesis tools can create gate-level netlists from RTL level design. Behavioral synthesis allowed engineers to design directly in terms of algorithms and the behavior of the circuit, and then use EDA tools to do the translation and optimization in each phase of the design. However, behavioral synthesis did not gain widespread acceptance. Today, RTL design continues to be very popular. Verilog HDL is also being constantly enhanced to meet the needs of new verification methodologies. Formal verification and assertion checking techniques have emerged. Formal verification applies formal mathematical techniques to verify the correctness of Verilog HDL descriptions and to establish equivalency between RTL and gate-level netlists.

However, the need to describe a design in Verilog HDL will not go away. Assertion checkers allow checking to be embedded in the RTL code. This is a convenient way to do checking in the most important parts of a design. New verification languages have also gained rapid acceptance. These languages combine the parallelism and hardware constructs from HDLs with the object oriented nature of C++. These languages also provide support for automatic stimulus creation, Overview of Digital Design with Verilog HDL, checking, and coverage. However, these languages do not replace Verilog HDL. They simply boost the productivity of the verification process. Verilog HDL is still needed to describe the design.

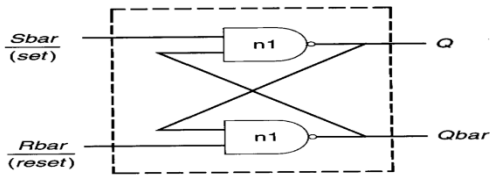
For very high-speed and timing-critical circuits like microprocessors, the gate-level netlist provided by logic synthesis tools is not optimal. In such cases, designers often mix gate-level description directly into the RTL description to achieve optimum results. This practice is opposite to the high-level design paradigm, yet it is frequently used for high-speed designs because designers need to squeeze the last bit of timing out of circuits, and EDA tools sometimes prove to be insufficient to achieve the desired results. Another technique that is used for system-level design is a mixed bottom-up methodology where the designers use either existing Verilog HDL modules, basic building blocks, or vendor-supplied core blocks to quickly bring up their system simulation. This is done to reduce development costs and compress design schedules. For example, consider a system that has a CPU, graphics chip, I/O chip, and a system bus. The CPU designers would build the next-generation CPU themselves at an RTL level, but they would use behavioral models for the graphics chip and the I/O chip and would buy a vendor-supplied model for the system bus. Thus, the system-level simulation for the CPU could be up and running very quickly and long before the RTL descriptions for the graphics chip and the I/O chip are completed.

Components of a Verilog Module

- **A module definition always begins with the keyword module.**
- **The module name, port list, port declaration, and optional parameters must come first in a module definition.**
- **The five components within a module are:**
  - **variable declarations,**
  - **dataflow statements,**
  - **instantiation of lower modules,**
  - **behavioral blocks and,**
  - **tasks or functions.**
- **These components can be in any order and at any place in the module definition.**

- **The endmodule statement must always come last in a module definition.**
- **Verilog allows multiple modules to be defined in a single file.**
- **The modules can be defined in any order in the file.**

To understand the components of the module shown above, consider a simple example of an SR latch.



- **The SR latch has S and R as the input ports.**
- **Q and Qbar as the output ports.**

The SR latch and its stimulus can be modeled

- **In the SR latch definition above, notice that all components need not be present in a module. We do not find variable declaration, dataflow (assign) statements, or behavioral blocks (always or initial).**
- **However, the stimulus block contains module name, wire, reg, and variable declarations, instantiation of lower level modules, behavioral block (initial), and endmodule statement but does not contain port list, port declaration, and data flow (assign) statements.**
- **Thus, all parts except module, module name, and endmodule are optional and can be mixed and matched as per design needs.**

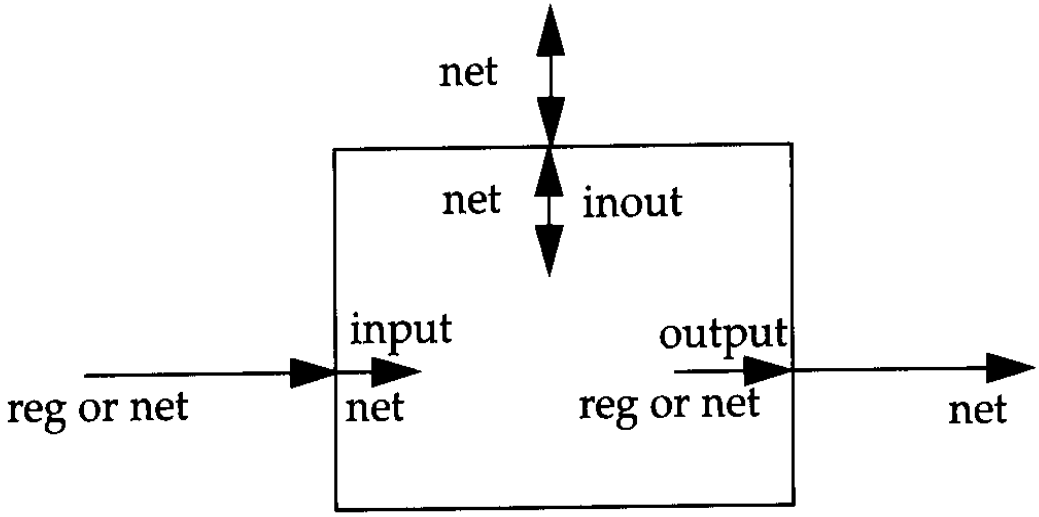
Ports

- **Ports provide the interface by which a module can communicate with its environment.**
- **For example, the input/output pins of an IC chip are its ports.**
- **Ports are also referred as terminals.**

List of Ports

- **A module definition contains an optional list of ports.**
- **If module does not exchange any signals with the environment, there are no ports in the list.**
- **Consider a 4-bit full adder that is instantiated inside a top-level module Top.**

Port Connection Rules



Verilog is quite a rich language and supports various levels of hardware specification of increasing abstraction using the various language constructs available. These levels can be freely mixed in one circuit. A taxononomy of four levels is handy for the current purposes.
1. Structural Verilog: a hierarchic netlist form of the type generated by the CSYN compiler.
2. Continuous Assignment: allows advanced combinatorial expressions, including adders and bus comparison, etc.
3. Synthesisable Behavioral Subset: allows clocked assignment, if-then-else and case statements, resulting in state machine synthesis.
4. The Unsynthesisable Behavioral Remainder: includes programming constructs such as unbounded while loops and fork-join.

Verilog Lexicography and Comments

A Verilog source file contains modules and comments. All whitespace characters (outside of macro definition lines) are optional and ignored, except where adjacent identifiers would elide.
One-line comments are introduced with the revered double slash syntax and terminate at the end of line. Block comments are introduced with slash-star and terminate with star-slash. Block comments may not be nested.

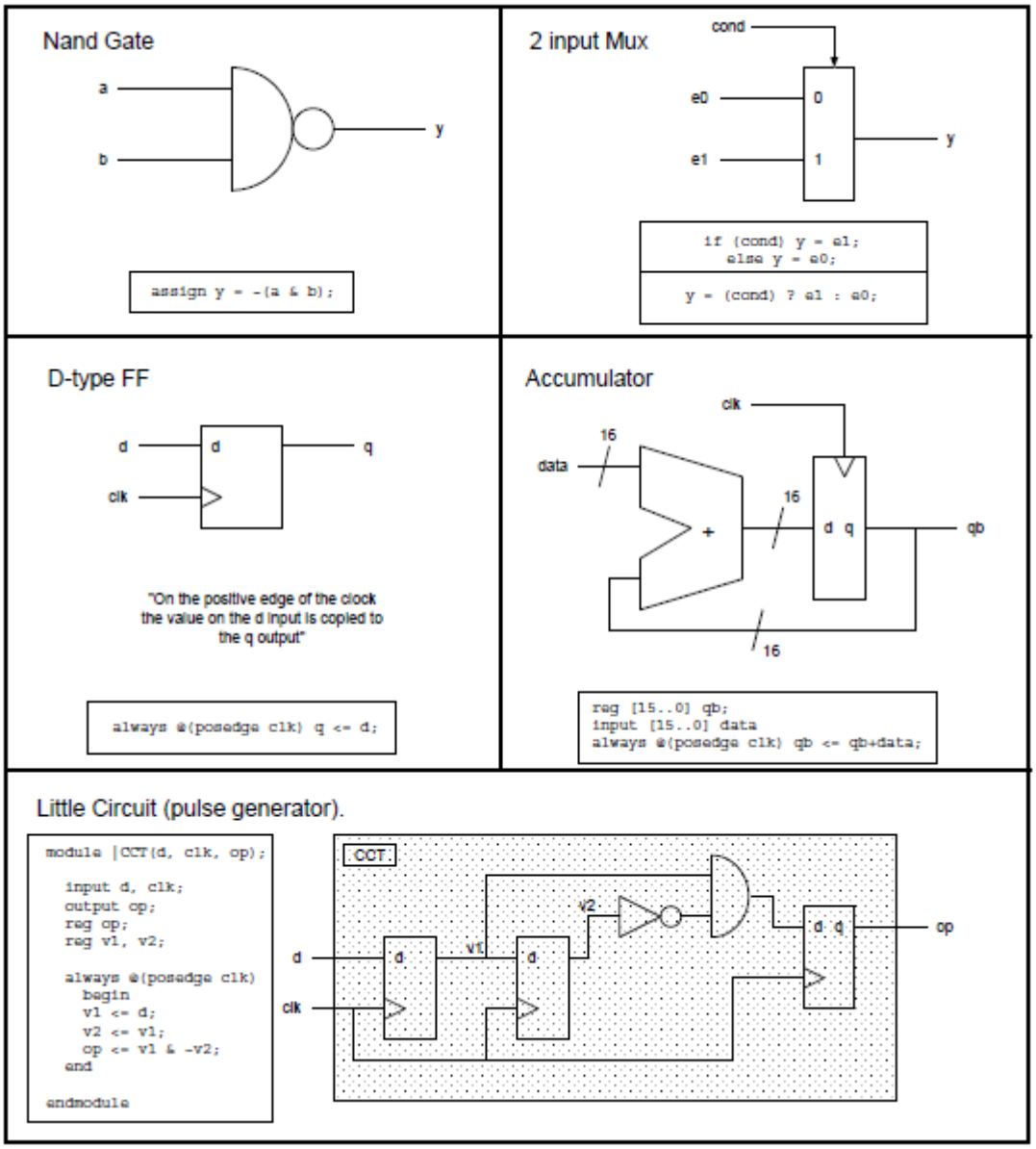
```
/*
* This is a block comment.
*/
// And this is a single line comment.
```

Preprocessor

Verilog has a preprocessor which can be used to define textual macros as follows

```
‘define RED 3
```

‘define BLUE 4
if (a== ‘RED) b <= ‘BLUE;
Note that the reverse quote is used both in macro definitions and in their use.
The preprocessor can also be used to control conditional inclusion of parts of the source file. This is useful is variations are needed depending on whether the design is to be simulated or synthesized.



1
assign y = ~(a & b);
if (cond) y = e1;
else y = e0;
y = (cond) ? e1 : e0;
reg [15..0] qb;
input [15..0] data
always @(posedge clk) qb <= qb+data;
always @(posedge clk) q <= d;

```
module |CCT(d, clk, op);
input d, clk;
output op;
reg op;
reg v1, v2;
always @(posedge clk)
begin
v1 <= d;
v2 <= v1;
op <= v1 & ~v2;
end
endmodule
```

Identifier

Identifier names consist of alphanumeric strings which can also contain underscores and must start with a letter. Case is significant. In this manual, capitalised identifiers are used by convention for module names, but this is not required by the language definition or CSYN.

CSYN maintains separate identifier spaces for modules, nets, macros and instances, allowing the same identifier to be used in more than one context. The net identifier space also includes other classes of identifier that could be used in expressions, such as parameters and integers.

Many postprocessing tools do not maintain such separate namespaces, so it is possible to have clashes within these tools that were not a problem for CSYN.

Module, Port and Net Definitions

A module is the top-level syntactic item in a Verilog source file. Each module starts with the word module and ends with endmodule and between the two are 'module declarative items'.

```
// Here is a module definition, called FRED
module FRED(q, a, b);
input a, b; // These are the inputs
output q; // Make sure your comments are helpful
// ..... Guts of module go here .....
endmodule
```

After the module name comes a list of formal parameters in parenthesis. These are also known as the 'ports'. In the module body, semicolons are used at the end of atomic statements, but not composite statements. Hence there is never one after an endmodule (or an end or an endcase).

Useful module declarative items include the following statements input, output, inout, wire, reg, tri, assign, always and gate and module structural instantiations.

The order of the declarative items is unimportant semantically, except that nets need to be declared before they are referenced.

Input and Output definitions

Each of the ports of a module must be explained using the 3 keywords

• input : signal coming into the module

• output: signal going out of the module
• inout : bidirectional signal.

These declarations take a comma separated list of ports which share a similar direction.

When a modules are instantiated, CSYN checks that no two outputs are connected to each other and that all nets of type wire are driven by exactly one output.

Bus definitions:

A Verilog signal may either be a simple net or else a bus. When an identifier is introduced (using a declaration such as input or tri etc.), if it is given a range, then it is a bus, otherwise it is a simple net. When an identifier which denotes a bus is referred to without giving an index range, then the full range of the bus is implied. Most Verilog operators will operate on both busses and simple nets.

Busses are defined by putting a range in square brackets after the keyword which declares then.

For example

```
// Here is a module definition with two input busses
module FRED(q, d, e);
input [4:0] d, e; // Ports d and e are each five bit busses
output q; // q is still one bit wide
endmodule
```

Note that when a module port is actually a bus, the bus width is not given in the formal parameter list, just its identifier.

The most significant bit is always specified first and the least significant bit index second. This convention is understood by certain built-in arithmetic operators, such as addition. Users, such as die-hard IBM employees who wish to number their bits starting with '1' as the most significant are free to do so, but normally, the first index is the number of bits in the bus minus one and the second index is zero. The least significant index does not have to be zero and this is helpful sometimes - e.g. the least significant two bits are normally missing in an A32, byte addressed address bus. In summary, the numbers use to index a bus are just labels on the bit lanes and do not introduce any implied shifts or bit-reversals when the busses are used in expressions.

Local signal names:

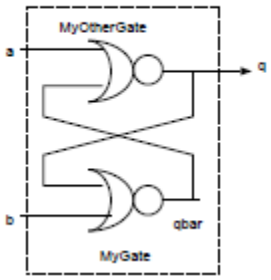
Local signal names are introduced with the wire, reg and tri statements. There are slight variations in the way these types of nets can be used, but the syntax of the three statements is the same.

Here is an example using some wires:

```
// Here are some local nets being defined

module FRED(q, a);
input a;
```

output q;
wire [2:0] xbus, ybus; // Two local busses of three bits each
wire parity_x, parity_y; // Two simple local signals.
...
Local nets which will be driven from the outputs of instantiated modules and by continuous assignments should be of type wire.



Local nets which are to be assigned to from behavioral statements should be of type reg, whereas those assigned in continuous assignments should be type wire.

Outputs which are to assigned to from behavioral statements should be defined twice, first as outputs and then with type reg.

Structural Specification:

One further statement type, beyond the wire and port declarations, is all we need for structural specification of a hierarchical or flat netlist. This is the component instance statement.

The syntax of a submodule or component instance is:
<modname><instancename>(<portlist>) [, <instancename>(<portlist>)] ;
Where the square brackets are meta-syntax to indicate optional repetition and the angle brackets indicate that an identifier is not a keyword. The syntax allows several instances of the same submodule to be specified in one instance statement, if desired.
// Here is a module definition with submodule instances.
module FRED(q, a, b);
input a, b; // These are the inputs
output q; // Make sure your comments are helpful
wire qbar; // Local, unported net
NOR2 mygate(q, a, qbar), myothergate(qbar, b, q);
endmodule
The submodule NOR2 must be defined somewhere, as a leaf module in a technology library, or in the same or another design-specific Verilog source file.
Instance names, such as 'mygate' in the example, are optional. CSYN will make up instance names for instances in its output netlists if they are not provided in the source Verilog. However, this is not recommened since these names are likely to change each time a circuit is compiled and this is often inconvenient if a simulator or placement tool has retained instance name state from one run to another.
Note that explict instantiation of this type at the low level of individual gates is not normally done by a human, unless this is an especially critical section of logic. Such gate-level Verilog is normally

synthesised from the higher levels of Verilog source by CSYN or other tools.

Positional and Named port mapping:

The component instance example above, for NOR2, uses positional port mapping. The writer of the Verilog had to know that in the definition of NOR2, the output net was put first. That is

```
// Here is the signature of NOR2
module NOR2(y, x1, x2);
input x1, x2; // These are the inputs
output y; // Make sure your comments are always helpful
endmodule
```

However, Verilog also allows named port mapping in the instance statement. Using named port mapping, the writer does not have to remember the order of the ports in the module definition, he must remember their names instead. The chance of misconnection through giving the wrong order is eliminated and the compiler is able to generate more helpful error messages when the wrong number of arguments is supplied to an instance.

```
// Here is a module definition with named port mapping.
module FRED(q, a, b);
input a, b; // These are the inputs
output q; // Make sure your comments are helpful
wire qbar; // Local, unported net
NOR2 mygate(.x1(a), .y(q), .x2(qbar)),
myothergate(.y(qbar), .x2(q), .x1(b));
endmodule
```

As can be seen, the syntax of a named port association is

. <formal-name> (<parameter-expression>)

whereas for the positional syntax, we just have a list of parameter expressions.

I use the term 'parameter expression' since the actual parameters, for both syntaxes, do not have to be simply signal names, but for inputs to the submodule, can be any signal expression which results in a signal of the correct width. Signal expressions are explained below.

It is often necessary to pass a single net of a bus, or a subgroup of a bus to or from a module instance. This can be done using the bus sub ranging operators described later.

Continuous Assignment and Signal Expressions:

The level of Verilog specification which comes above netlists and structural instantiation is continuous assignment. The syntax of a continuous assignment within CSYN is assign <signal> = <signal-expression> ;

For example

```
wire p;
assign p = (q & r) | (~r & ~s);
```

The stile (vertical bar) denotes Boolean OR, the ampersand denotes Boolean AND and the tilde denotes inversion. The signals q, r and s must be defined already using one of the declarations input, output, inout, wire, tri or reg.

A continuous assignment is synthesized into a set of gates which achieve the desired logic function.

Logic minimization is applied to the signal expression on the right-hand side, but CSYN does not eliminate explicitly declared nets or their interconnection pattern across continuous assignments.

Verilog allows a short hand that combines a wire declaration with an assignment. The above example can be written more concisely

```
wire p = (q & r) | (~r & ~s);
```

where the assignment to p is combined into the wire statement that introduced it.

Constant numeric expressions:

Numerical constants can be introduced in a number of bases using the syntax:

<width-expr> ' <base><value-expr>

where the base is a single letter:

b : binary
o : octal
h : hexadecimal
d : decimal (the default)

For example

```
wire [7:0] bus1, clipbus;
assign clipbus = (bus1 > 8'd127) ? 8'd127 : bus1;
```

where 8'd127 is the decimal number 127 expressed to a width of eight bits.

If constants are given without a width, CSYN will attempt to provide an appropriate width to use.

If the situation is ambiguous, an error is flagged. CSYN will also accept simple strings of digits and treat them as unsized decimal numbers.

Underscores are allowed in numbers. These are ignored by the compiler but can make the language more readable. For instance

```
wire [23:0] mybus = yourbus & 24'b11110000_11110000_00001111;
```

1.12 Bus subranging and concatenation

To have access to parts of a bus, indexing can be used with a bit field in square brackets. When the index is left out, which is the case in the examples until now, the default width of the bus is used.

examples

```
wire [31:0] dbus;
wire bit7 = dbus[7];
wire [7:0] lowsevenbits = dbus[7:0];
wire [31:0] swapbus = dbus[0:31];
```

It is possible to assign to part of a bus using a set of assignment statements provided that each net of the bus is assigned to exactly once (i.e. the ranges are disjoint). For example

```
wire [11:0] boo;
input [3:0] src;
assign boo[11:8] = src;
assign boo[7:4] = 4'b0;
assign boo[3:0] = src + 4'd1;
```

It is also possible to form anonymous busses using a concatenation of signals. Concatenations are introduced with curly braces and have width equal to the sum of their components. Hence, the previous example can be rewritten more briefly

```
assign boo = { src, 4'b0, src + 4'd1 };
```

Note that all items within a concatenation must have a clearly specified width. In particular, unsized numbers may not be used.

Verilog Operators:

Table 1 defines the set of combinatorial operators available for use in signal expressions in order of binding power. Parenthesis can be used to overcome binding power in the normal sort of way.

All operators can be applied to simple nets or to busses. When a diadic operator is applied where one argument is a simple net and the other a bus, CSYN treats the simple net as the least significant bit of a bus with all other bits zero. This is also the case when busses of unequal size are combined. Verilog has no support for two's complement or sign extension built in—it treats all numbers as unsigned. When signed operation is needed, the user must create such behavior around the built-in operators. (Of course, addition and subtraction work both with unsigned and two's complement numbers anyway).

There are both Boolean and logical versions of the AND and OR operators. These have different effects when applied to busses and also different precedence binding power. The Boolean operators '&' and or '|' produce bitwise ANDs and ORs of their two arguments. The logical operators '&&' and '||' first or-reduce (see section 1.14) their arguments and then combine them in a unit width AND or OR gate.

Unary Reduction:

It is possible to reduce a bus to a wire under an operator: that is, to combine all of the wires of a bus using an associative Boolean diadic operator. The syntax of unary reduction required by CSYN insists on two sets of parenthesis.

(<unary-op> (<signal-expr>))

the unary-operator must be one of the ones in table 2. For example

wire [9:0] mybus;

wire x = (& (mybus));

constructs a ten-input AND gate such that x will be a logical one if the bus contains all ones.

Symbol	Function	Resultant width
~	monadic negate	as input width
–	monadic complement (*)	as input width
!	monadic logic not	unit
*	unsigned binary multiply (*)	sum of arg widths
/	unsigned binary division (*)	difference of arg widths
%	unsigned binary modulus (*)	width of rhs arg
+	unsigned binary addition	maximum of input widths
–	unsigned binary subtraction	maximum of input widths
>>	right shift operator	as left argument
<<	left shift operator	as left argument
==	net/bus comparison	unit
!=	inverted net/bus compare operator	unit
<	bus compare operator	unit
>	bus compare operator	unit
>=	bus compare operator	unit
<=	bus compare operator	unit
&	diadic bitwise and	maximum of both inputs
^	diadic bitwise xor	maximum of both inputs
~^	diadic bitwise xnor (*)	maximum of both inputs
|	diadic bitwise or	maximum of both inputs
&&	diadic logical and	unit
||	diadic logical or	unit
? :	conditional expression	maximum of data inputs

Conditional Expressions:

The conditional expression operator allows multiplexers to be made. It has the syntax
(<switch-expr>) ? <true-expr> : <false-expr>
The value of the expression is the false-expression when the switch-expression is zero and the true expression otherwise. If the switch-expression is a bus, then it is unary-reduced under the OR operator. CSYN does not insist on the parenthesis around the switch-expression, but they are recommended. The association of the conditional expression operator is defined to enable multiple cases to be tested without multiple parentheses.

Examples

```
wire [7:0] p, q, r;
wire s0, s1;
wire [7:0] bus1 = (s0) ? p : q;
wire [7:0] bus2 = (s0) ? p : (s1) ? q : r;
```

The bus comparison predicates (== ! = <><= >=) take a pair of busses and return a unity width result signal. For example

```
wire yes = p > r;
```

All comparison operators in Verilog treat their arguments as unsigned integers. There is no explicit support for basic signed operations on two's complement or sign extensions.

Dynamic Subscription:

CSYN supports dynamic subscription (indexing) of a bus in a signal expression provided that the selected width from the bus is one bit wide. Dynamic subscription is not supported on the let hand side of any assignment. Here is an example of use module

```
TEST();
wire [10:0] bus;
wire [3:0] idx;
...
wire bit = bus[idx];
...
endmodule
```

Behavioral Specification

Verilog HDL contains a full set of behavioral programming constructs, allowing one to write procedural style programs, with ordered assignments (as opposed to continuous assignments). TheCSYN-V2 Verilog compiler supports a subset of the language's behavioral constructs for logic synthesis. These are described in this section. Behavioral statements must be included within an initial declaration or an always declaration. CSYN (release cv2) ignores the contents of initial statements which simply assign to a variable: these may be present and are useful when simulating the source file. The following form of the Verilog always construct is the most useful for RTL designs.

It has the syntax

always @(posedge <clocksignal>) <behavioral-statement>

This statement causes execution of the behavioral statement each time the clock goes from zero to one. An alternative keyword negedge has the expected effect.1.18 begin-end Behavioral Statement this statement is simply the sequencing construct required to extend the range of behavioral statements which include behavioral statements (such as if and always) to cover multiple behavioral statements.

The syntax is begin <behav-statement> [<behav-statement>] end

where multiple instances of the contents of the square brackets may be present. The order of statements inside a begin-end block is important when multiple blocking assignments are made to the same variable. Multiple non-blocking assignments to the same variable are not allowed unless they are in different branches of an if-then-else.

Repeat Behavioral Statement:

The repeat statement is supported by CSYN provided it has a compile-time constant argument and is simply textually expanded into that many copies of its argument statement (which of course can be a block).
The syntax is repeat (<simple-numerical-expression>) <behav-statement>

if-then-else Behavioral Statement

The if statement enables selective execution of behavioral statements. The target behavioral statement (which can be any type of behavioral statement, including further if statements) is executed if the signal expression is non-zero. If (<signal-expr>) <behav-statement> [else <behav-statement>]If the signal expression given for the condition is a bus it is or-reduced so that the target behavioral statement is executed if any of the wires in the bus are non-zero. When the optional else clause is present, the else clause is executed if the signal-expression condition is false.

case Behavioral Statement:

The case statement enables one of a number of behavioral statements to be executed depending on the value of an expression. The syntax of the statement is case (<top-signal-expr>) <case-items> endcase where the case-items are a list of items of the form
<tag-signal-expr> [, <tag-signal-expr>] : <behav-statement>and the list may include one default item of the form default : <behav-statement>The semantic is that the top signal expression is compared against each of the tag signal expressions and the behavioral statement of the first matching tag is executed. If no tags match, the statement provided as the default is executed, if present. Here is an example always @(aal_counter) case (aal_counter) // full_case
0: aal_header <= 8'h00;
1: aal_header <= 8'h17;
2: aal_header <= 8'h2D;
3: aal_header <= 8'h3A;
default: aal_header <= 8'h74;
endcase

There are a number of variations from the case statements found in other languages, such as C. These are: the tag values do not have to be constants, but can be any signal expression; for each group of tag expressions there must be exactly one target statement (begin-end must be used to overcome this) and there is no 'fall-through' from one case section to the next (i.e. the 'break' of C is implied.)1.22 Behavioral Assignment Statements
Behavioral assignment is used to change the values of nets which have type 'reg'. These retain their value until the next assignment to them. CSYN supports two types of behavioral assignment(as well as the continuous assignment of section 1.10).
1.22.1 Non-blocking or 'delayed' assignment Using the <= operator, an assignment is 'non-blocking' (or is 'delayed') in the sense that the value of the assigned variable does not change immediately and subsequent references to it in the same
group of statements (always block) will refer to its old value. This corresponds to the normal behavior of synchronous logic, where each registered net corresponds to a D-type (or broadside register for a bus) clocked when events in the sensitivity list dictate. The syntax is<regnet><= <signal-expression> ;The left hand side signal must have type reg. CSYN supports only one such assignment to a variable to be made within a program, except that multiple assignments can be made if they are in separate clauses of an if-then-else construct. For example, to swap to registers we may use
x <= y;
y <= x;
A variable may only be updated by one delayed assignment per clock cycle.

Blocking or 'immediate' assignment:

Verilog also supports assignments using the '=' operator. These take effect immediately, rather than at the next time step.

Behavioral Example and Style:

To support registered outputs from a module, it is allowed for the same net name to appear both in a reg statement and an output statement.

For example:

```
module EXAMPLE(ck, ex);
input ck; // This is the clock input
output ex;
reg ex, ez; // ex is both a register and an output
reg [2:0] q; // q is a local register for counting.
always @(posedge ck)
begin
if (ez) begin
if (q == 2) ex <= 1; else ex <= ~ex;
q <= q + 3'd1;
end
ez <= ~ez;
end
endmodule
```

The example defines a divide by two flip-flop ez and a three bit counter with clock enabled by ez.
The output ex is also a register. In addition ex is updated twice within the always block, but under different conditions. Since q is a three bit register, it will count 0, 1, 2, ... 7, 0, 1 ...

Unsynthesisable Constructs:

Several statements cannot normally be turned into hardware, but are useful for generating test wrappers to exercise a design under simulation.
Hash Delays The clock modules found in the libraries and the back-annotated modules from XNFTOV contain hash delays. These actually have three forms:

1. In a continuous assignment: 'assign #n v = exp;'

2. Inside a behavioral assignment: 'v <= #n exp;'

3. Between behavioral statements: 'begin s1; #n s2; ... end'
The first two forms introduce a delay between calculating the value of an expression and the assigned variable taking on the new value. The third form actually pauses the flow of a 'thread' of execution. These constructs should not be fed into the synthesizer and should only be used for test harnesses in simulation. Simulator Meta-commands.

The $display command instructs the simulator to print output. This command and others like it are ignored entirely by the synthesizer. The first argument is a string. The string contains tokens introduced with a percent sign that control printing of the remaining arguments. The percent operators available are

• h : print a hex value
• d : print a decimal value
• b : print a binary value
• m : print the surrounding module instance name (does not consume an argument from the list
• t : print the simulation time (must tie up with $time in the argument list).
Here is an example

```
always @(posedge clk) $display("module %m: time %t, bus value=%h",
$time, mybus);
```

This will give output that might look like

```
module SIMSYS_mychip_gate1: time 10445, bus value=4DF
```

A clock Generator
To generate a free running clock, one can use

```
module CLOCK(q);
output q; reg q;
// Toggle ever 50 or so time units.
initial begin q = 0; forever #50 q = !q; end
endmodule
```

A Reset Pulse

To generate a reset pulse at the start of a simulation, one can use

```
module POWER_ON_RESET(q);
output q; reg q;
// Toggle ever 50 or so time units.
initial begin q = 1; #200 q = 0; end
endmodule
```

IMPORTANT QUESTIONS

PART – A (2 Marks)

1. What is Verilog?

2. What are the types of HDL?

3. What are gate primitives?

4. What are the different kinds of the test bench?

5. Give the two blocks in behavioral modeling.

6. Define FSM.

7. What do you mean by Data flow model?

8. What is Switch-level modeling?

9. Define vector in verilog.

10. What are the different types of modeling Verilog?

PART-B (16 Marks)

1. Design a 4-bit carry look ahead adder and write the verilog HDL for it. (16)

2. Design 4X1 multiplexer and write the HDL for it in all four modeling: (16)

3. Briefly explain behavioral modeling (all functions) with an example: (16)

4. Design and develop a project in HDL to compare **x5x4x3x2x1x0** with **y5y4y3y2y1y0**. Check the output by means of test bench. (16)

5. Explain the following with an example:

i) Tasks and functions (4)

ii) Test bench for multiplexer (4)

iii) Difference between always and initial (4)

iv) Blocking and non-blocking statements (4)

UNIVERSITY QUESTIONS

B.E/B.Tech Degree Examination, November/December 2009.

Seventh Semester Electronics and Communication Engineering

EC 1401-VLSI DESIGN Question Paper

(Common to B.E.(Part-Time) Sixth Semester Regulation 2005) (Regulation 2004)

Time: Three hours Maximum:100 marks

Answer all questions.

Part A-(10*2=20 marks)

1.What are the different MOS layers?

2. What are the two types of layout design rules?

3. Define rise time and fall time.

4. What is a pull down device?

5. What are the difference between task and function?

6. What is the difference between === and == ?

7. What is CBIC ?

8. Draw an assert high switch condition if input = 0 and input =1.

9. What do you mean by DFT?

10. Draw the boundary scan input logic diagram.

Part B - (5*16=80 marks)

11.a) Discuss the steps involved in IC fabrication process.(16)

Or

b) Describe n-well process in detail.(16)

12.a)i)Explain the DC characteristics of CMOS inverter with neat sketch.(8)

ii)Explain channel length modulation and body effect.(8)

Or

b)i)Explain the different regions of operation in a MOS transistor.(10)

ii)Write a note on MOS models.(6)

13.a)Explain in detail any five operators used in HDL .(16)

Or

b)i)Write the verilog code for 4 bit ripple carry full adder.(10)
ii)Give the structural description for priority encoder using verilog.(6)

14.a)Explain in detail the sequence of steps to design an ASIC.(16)

Or

b) Describe in detail the chip with programmable logic structures.(16)

15.a)Explain in detail Scan Based Test Techniques.(16)

Or

b) Discuss the three main design strategies for testability.(16)

B.E./B.Tech. DEGREE EXAMINATION, NOVEMBER/ DECEMBER 2007
Seventh semester
(Regulation 2004)
Electronics and Communication Engineering
EC 1401 – VLSI DESIGN
(Common to B.E. part time) sixth Semester Regulation 2005)

Answer all questions
PART A – (10*2=20marks)

1. What are the advantages of SOI CMOS process?
2. Distinguish electrically alterable and non-electrically alterable ROM.
3. Compare nMOS and pMOS.
4. Compare enhancement and depletion mode devices.
5. What is meant by continuous assignment statement in Verilog HDL?
6. What is a task in Verilog?
7. Give the application of PLA.
8. What is meant by a transmission gate?
9. What is the aim of adhoc test techniques?
10. Distinguish functionality test and manufacturing test.

PART B – (5*16=80 marks)

11. (a) (i) Draw and explain the n-well process.
(ii) Explain the twin tub process with a neat diagram.
OR
(b) (i) Discuss the origin of latch up problems in CMOS circuits with necessary diagrams. Explain the remedial measures.
(ii) Draw and explain briefly the n-well CMOS design rules.

12. (a) (i) Derive expressions for the drain to source current in the nonsaturated and saturated regions of operation of an nMOS transistor.
(ii) Define and derive the transconductance of nMOS transistor.
OR
(b) (i) Discuss the small signal model of an nMOS transistor.
(ii) Explain the CMOS inverter DC characteristics.

13. (a) (i) Give a verilog structural gate level description of a bit comparator.
(ii) Give a brief account of timing control and delay in verilog.
OR
(b) (i) Give a verilog structural gate level description of a ripple carry adder.
(ii) Write a brief note on the conditional statements available in verilog.

14. (a) (i) Compare the different types of ASICs.
(ii) Discuss the operation of a CMOS latch.
OR
(b) Explain the ASIC design flow with a neat diagram. Enumerate clearly the different steps involved.

15. (a) Explain the chip level test techniques.
OR
(b) Explain the system level test techniques.