

## Unit-1

### Verilog as HDL

Verilog has a variety of constructs as part of it. All are aimed at providing a functionally tested and a verified design description for the target FPGA or ASIC.

The language has a dual function – one fulfilling the need for a design description and the other fulfilling the need for verifying the design for functionality and timing constraints like propagation delay, critical path delay, slack, setup, and hold times.

### Levels of Design Description

The components of the target design can be described at different levels with the help of the constructs in Verilog.

In Verilog HDL a module can be defined using various levels of abstraction. There are four levels of abstraction in verilog.

They are: 1. Circuit Level 2. Gate Level 3. Data Flow Level 4. Behavioral Level

### Circuit Level

At the circuit level, a switch is the basic element with which digital circuits are built. Switches can be combined to form inverters and other gates at the next higher level of abstraction. Verilog has the basic MOS switches built into its constructs, which can be used to build basic circuits like inverters, basic logic gates, simple 1-bit dynamic and static memories. They can be used to build up larger designs to simulate at the circuit level, to design performance critical circuits.

The below Figure1 shows the circuit of an inverter suitable for description with the switch level constructs of Verilog.

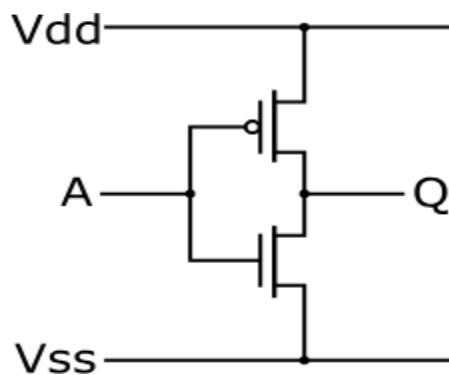


Figure 1 CMOS inverter

## Gate Level

At the next higher level of abstraction, design is carried out in terms of basic gates. All the basic gates are available as ready modules called “Primitives.” Each such primitive is defined in terms of its inputs and outputs. Primitives can be incorporated into design descriptions directly. Just as full physical hardware can be built using gates, the primitives can be used repeatedly and judiciously to build larger systems.

Figure 2 shows an AND gate suitable for description using the gate primitive of Verilog.

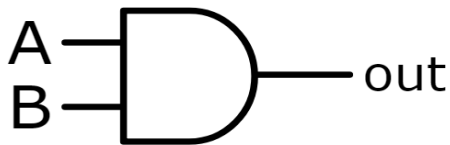


Figure 2 AND gate symbol

The gate level modeling or structural modeling as it is sometimes called is akin to building a digital circuit on a bread board, or on a PCB. One should know the structure of the design to build the model here. One can also build hierarchical circuits at this level. However, beyond 20 to 30 of such gate primitives in a circuit, the design description becomes unwieldy; testing and debugging become laborious.

## Data Flow

Data flow is the next higher level of abstraction. All possible operations on signals and variables are represented here in terms of assignments. All logic and algebraic operations are accommodated. The assignments define the continuous functioning of the concerned block. At the data flow level, signals are assigned through the data manipulating equations. All such assignments are concurrent in nature. The design descriptions are more compact than those at the gate level.

Figure 3 shows an A-O-I relationship suitable for description with the Verilog constructs at the data flow level.

$$e = \overline{a.b + c.d}$$

Figure 3 An A-O-I gate represented as a data flow type of relationship.

## Behavioral Level

Behavioral level constitutes the highest level of design description; it is essentially at the system level itself. With the assignment possibilities, looping constructs and conditional branching possible, the design description essentially looks like a “C” program.

A module can be implemented in terms of the design algorithm. The designer no need to have any knowledge of hardware implementation.

The statements involved are “dense” in function. Compactness and the comprehensive nature of the design description make the development process fast and efficient.

Figure 4 shows an A-O-I gate expressed in pseudo code suitable for description with the behavioral level constructs of Verilog.

<p>If (<i>a, b, c</i> or <i>d</i> changes) Compute <i>e</i> as <math display="block">e = a.b + c.d</math></p>
-----------------------------------------------------------------------------------------------------------------------

Figure . 4 An A-O-I gate in pseudo code at behavioral level.

## The Overall Design Structure in Verilog

The possibilities of design description statements and assignments at different levels necessitate their accommodation in a mixed mode. In fact the design statements coexisting in a seamless manner within a design module is a significant characteristic of Verilog. Thus Verilog facilitates the mixing of the above-mentioned levels of design. A design built at data flow level can be instantiated to form a structural mode design. Data flow assignments can be incorporated in designs which are basically at behavioral level.

## **Concurrency**

In an electronic circuit all the units are to be active and functioning concurrently. The voltages and currents in the different elements in the circuit can change simultaneously. In turn the logic levels too can change. Simulation of such a circuit in an HDL calls for concurrency of operation.

A number of activities – may be spread over different modules – are to be run concurrently here. Verilog simulators are built to simulate concurrency. (This is in contrast to programs in the normal languages like C where execution is sequential.)

Concurrency is achieved by proceeding with simulation in equal time steps. The time step is kept small enough to be negligible compared with the propagation delay values. All the activities scheduled at one time step are completed and then the simulator advances to the next time step and so on. The time step values refer to simulation time and not real time. One can redefine timescales to suit technology as and when necessary and carry out test runs.

In some cases the circuit itself may demand sequential operation as with data transfer and memory-based operations. Only in such cases sequential operation is ensured by the appropriate usage of sequential constructs from Verilog HDL.

## **Simulation and Synthesis**

The design that is specified and entered as described earlier is simulated for functionality and fully debugged. Translation of the debugged design into the corresponding hardware circuit (using an FPGA or an ASIC) is called “synthesis.”

The tools available for synthesis relate more easily with the gate level and data flow level modules [Smith MJ]. The circuits realized from them are essentially direct translations of functions into circuit elements.

In contrast many of the behavioral level constructs are not directly synthesizable; even if synthesized they are likely to yield relatively redundant or wrong hardware. The way out is to take the behavioral level modules and redo each of them at lower levels. The process is carried out successively with each of the behavioral level modules until practically the full design is available as a pack of modules at gate and data flow levels (more commonly called the “RTL level”).

## Programming Language Interface (PLI)

PLI provides an active interface to a compiled Verilog module. The interface adds a new dimension to working with Verilog routines from a C platform. The key functions of the interface are as follows:

- One can read data from a file and pass it to a Verilog module as input. Such data can be test vectors or other input data to the module. Similarly, variables in Verilog modules can be accessed and their values written to output devices.
- Delay values, logic values, etc., within a module can be accessed and altered.
- Blocks written in C language can be linked to Verilog modules.

## MODULE

Any Verilog program begins with a keyword – called a “module.” A module is the name given to any system considering it as a black box with input and output terminals as shown in Figure 1. The terminals of the module are referred to as ‘ports’. The ports attached to a module can be of three types:

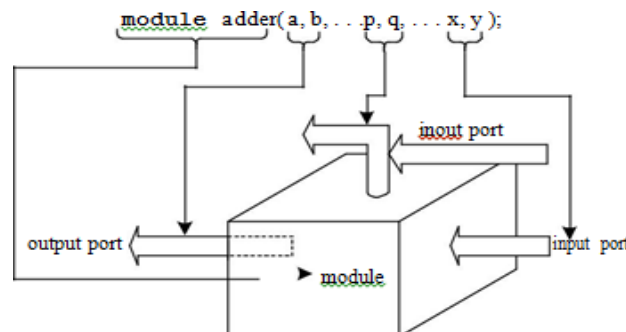


Figure 1 Representation of a module as black box with its ports.

- input ports through which one gets entry into the module; they signify the input signal terminals of the module.
- output ports through which one exits the module; these signify the output signal terminals of the module.
- inout ports: These represent ports through which one gets entry into the module or exits the module; These are terminals through which signals are input to the module sometimes; at some other times signals are output from the module through these.

Whether a module has any of the above ports and how many of each type are present depend solely on the functional nature of the module. Thus one module may not have any port at all; another may have only input ports, while a third may have only output ports, and so on.

All the constructs in Verilog are centered on the module. They define ways of building up, accessing, and using modules. The structure of modules and the mode of invoking them in a design are discussed here.

A module comprises a number of “lexical tokens” arranged according to some predefined order. The possible tokens are of seven categories:

- White spaces
- Comments
- Operators
- Numbers
- Strings
- Identifiers
- Keywords

The rules constraining the tokens and their sequencing will be dealt with as we progress. For the present let us consider modules. In Verilog any program which forms a design description is a “module.” Any program written to test a design description is also a “module.” The latter are often called as “stimulus modules” or “test benches.” A module used to do simulation has the form shown in Figure 2. Verilog takes the active statements appearing between the “module” statement and the “endmodule” statement and interprets all of them together as forming the body of the module. Whenever a module is invoked for testing or for incorporation into a bigger design module, the name of the module (“test” here) is used to identify it for the purpose.

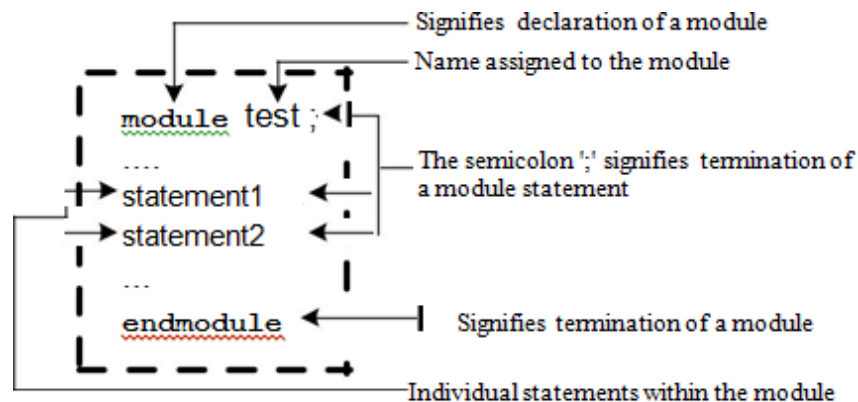


Figure 2 Structure of a typical simulation module.

## LANGUAGE CONSTRUCTS AND CONVENTIONS IN VERILOG

### Introduction

The constructs and conventions make up a software language. A clear understanding and familiarity of these is essential for the mastery of the language. Verilog has its own constructs and conventions [IEEE, Sutherland]. In many respects they resemble those of C language [Gottfried].

Any source file in Verilog (as with any file in any other programming language) is made up of a number of ASCII characters. The characters are grouped into sets — referred to as “lexical tokens.” A lexical token in Verilog can be a single character or a group of characters. Verilog has 7 types of lexical tokens- operators, keywords, identifiers, white spaces, comments, numbers, and strings.

### Case Sensitivity

Verilog is a case-sensitive language like C. Thus sense, Sense, SENSE, sENse,... etc., are all related as different entities / quantities in Verilog.

### Keywords

The keywords define the language constructs. A keyword signifies an activity to be carried out, initiated, or terminated. As such, a programmer cannot use a keyword for any purpose other than that it is intended for. All keywords in Verilog are in small letters and require to be used as such (since Verilog is a case-sensitive language). All keywords appear in the text in New Courier Bold-type letters.

### Examples

module -- signifies the beginning of a module definition.  
endmodule -- signifies the end of a module definition.  
begin -- signifies the beginning of a block of statements.  
end -- signifies the end of a block of statements.  
if -- signifies a conditional activity to be checked  
while -- signifies a conditional activity to be carried out.

### Identifiers

Any program requires blocks of statements, signals, etc., to be identified with an attached nametag. Such nametags are identifiers. It is good practice for us to use identifiers, closely related to the significance of variable, signal, block, etc., concerned. This eases understanding and debugging of any program.

e.g., clock, enable, gate\_1, . . .

There are some restrictions in assigning identifier names. All characters of the alphabet or an underscore can be used as the first character. Subsequent characters can be of alphanumeric type, or the underscore (`_`), or the dollar (`$`) sign – for example

`name`, `_name`. `Name`, `name1`, `name_$`, . . . -- all these are allowed as identifiers

`name aa` -- not allowed as an identifier because of the blank ( "`name`" and "`aa`" are interpreted as two different identifiers)

`$name` -- not allowed as an identifier because of the presence of "`$`" as the first character.

`1_name` -- not allowed as an identifier, since the numeral "`1`" is the first character

`@name` -- not allowed as an identifier because of the presence of the character "`@`".

`A+b m` not allowed as an identifier because of the presence of the character "`+`".

### White Space Characters

Blanks (`\b`), tabs (`\t`), newlines (`\n`), and formfeed form the white space characters in Verilog. In any design description the white space characters are included to improve readability. Functionally, they separate legal tokens. They are introduced between keywords, keyword and an identifier, between two identifiers, between identifiers and operator symbols, and so on. White space characters have significance only when they appear inside strings.

### Comments

Comments can be inserted in the code for readability and documentation. There are two ways to write comments. A one-line comment starts with `//`. Verilog skips from that point to the end of line. A multiple-line comment starts with `/*` and ends with `*/`. Multiple-line comments cannot be nested. However, one-line comments can be embedded in multiple-line comments.

```
a = b && c; // This is a one-line comment
```

```
/* This is a multiple line
```

```
comment */
```

```
/* This is /* an illegal */ comment */
```

```
/* This is //a legal comment */
```



## Operators

Operators are of three types: unary, binary, and ternary. Unary operators precede the operand. Binary operators appear between two operands. Ternary operators have two separate operators that separate three operands.

`a = ~ b;` // ~ is a unary operator. b is the operand

`a = b && c;` // && is a binary operator. b and c are operands

`a = b ? c : d;` // ?: is a ternary operator. b, c and d are operands

## Number Specification

There are two types of number specification in Verilog: sized and unsized.

### Sized numbers

Sized numbers are represented as `<size> '<base format> <number>`.

`<size>` is written only in decimal and specifies the number of bits in the number. Legal base formats are decimal ('d or 'D), hexadecimal ('h or 'H), binary ('b or 'B) and octal ('o or 'O). The number is specified as consecutive digits from 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f. Only a subset of these digits is legal for a particular base. Uppercase letters are legal for number specification.

<code>4'b1111</code>	//	This is a 4-bit	binary number
<code>12'habc</code>	//	This is a	12-bit hexadecimal number
<code>16'd255</code>	//	This is a	16-bit decimal number.

### Unsize numbers

Numbers that are specified without a `<base format>` specification are decimal numbers by default. Numbers that are written without a `<size>` specification have a default number of bits that is simulator- and machine-specific (must be at least 32).

`23456` // This is a 32-bit 'hc3 // This is a 32-bit 'o21 // This is a 32-bit

decimal number by default hexadecimal number octal number

### X or Z values

Verilog has two symbols for unknown and high impedance values. These values are very important for modeling real circuits. An unknown value is denoted by an x. A high impedance value is denoted by z.

12'h13x // This is a 12-bit hex number; 4 least significant bits unknown

6'hx // This is a 6-bit hex number

32'bz // This is a 32-bit high impedance number

An x or z sets four bits for a number in the hexadecimal base, three bits for a number in the octal base, and one bit for a number in the binary base. If the most significant bit of a number is 0, x, or z, the number is automatically extended to fill the most significant bits, respectively, with 0, x, or z. This makes it easy to assign x or z to whole vector. If the most significant digit is 1, then it is also zero extended.

### **Negative numbers**

Negative numbers can be specified by putting a minus sign before the size for a constant number. Size constants are always positive. It is illegal to have a minus sign between <base format> and <number>. An optional signed specifier can be added for signed arithmetic.

-6'd3 // 8-bit negative number stored as 2's complement of 3  
-6'sd3 // Used for performing signed integer math  
4'd-2 // Illegal specification

### **Underscore characters and question marks**

An underscore character "\_" is allowed anywhere in a number except the first character. Underscore characters are allowed only to improve readability of numbers and are ignored by Verilog.

A question mark "?" is the Verilog HDL alternative for z in the context of numbers.

12'b1111\_0000\_1010 // Use of underline characters for readability

4'b10?? // Equivalent of a 4'b10zz

### **Strings**

A string is a sequence of characters that are enclosed by double quotes. The restriction on a string is that it must be contained on a single line, that is, without a carriage return. It cannot be on multiple lines. Strings are treated as a sequence of one-byte ASCII values.

"Hello Verilog World" // is a string

"a / b" // is a string

## Value Set or Logic Values

Verilog supports four values and eight strengths to model the functionality of real hardware. The four value levels are listed in Table below.

Value Level	Condition in Hardware Circuits
0	Logic zero, false condition
1	Logic one, true condition
x	Unknown logic value
Z	High impedance, floating state

## Strengths

The logic levels are also associated with strengths. In many digital circuits, multiple assignments are often combined to reduce silicon area or to reduce pin-outs. To facilitate this, one can assign strengths to logic levels. Verilog has eight strength levels – four of these are of the driving type, three are of capacitive type and one of the hi-Z type.

In addition to logic values, strength levels are often used to resolve conflicts between drivers of different strengths in digital circuits. Value levels 0 and 1 can have the strength levels listed in Table below

Strength Level	Type	Degree
supply	Driving	strongest
strong	Driving	↑
pull	riving	
large	Storage	
weak	Driving	
medium	Storage	
small	Storage	
highz	High Impedance	weakest

If two signals of unequal strengths are driven on a wire, the stronger signal prevails.

For example, if two signals of strength strong1 and weak0 contend, the result is resolved as a strong1. If two signals of equal strengths are driven on a wire, the result is unknown. If two signals of strength strong1 and strong0 conflict, the result is an x. Strength levels are particularly useful for accurate modeling of signal contention, MOS devices, dynamic MOS, and other low-level devices.

## Data Types

The data handled in Verilog fall into two categories:

- (i) Net data type
- (ii) Variable data type

The two types differ in the way they are used as well as with regard to their respective hardware structures. Data type of each variable or signal has to be declared prior to its use. The same is valid within the concerned block or module.

### Nets

A net signifies a connection from one circuit unit to another. Such a net carries the value of the signal it is connected to and transmits to the circuit blocks connected to it. If the driving end of a net is left floating, the net goes to the high impedance state. A net can be specified in different ways.

**wire:** It represents a simple wire doing an interconnection. Only one output is connected to a wire and is driven by that.

**tri:** It represents a simple signal line as a wire. Unlike the wire, a tri can be driven by more than one signal outputs.

Nets are one-bit values by default unless they are declared explicitly as vectors. The terms wire and net are often used interchangeably.

### Variable Data Type

A variable is an abstraction for a storage device. It can be declared through the keyword `reg` and stores the value of a logic level: 0, 1, x, or z. A net or wire connected to a reg takes on the value stored in the reg and can be used as input to other circuit elements. But the output of a circuit cannot be connected to a reg. The value stored in a reg is changed through a fresh assignment in the program.

time, integer, real, and realtime are the other variable types of data; these are dealt with later.

### Time

Verilog simulation is done with respect to simulation time. A special time register data type is used in Verilog to store simulation time. A time variable is declared with the keyword `time`. The width for time register data types is implementation-specific but is at least 64 bits. The system function `$time` is invoked to get the current simulation time.

```
time save_sim_time; // Define a time variable save_sim_time initial
```

```
save_sim_time = $time; // Save the current simulation time
```

## Scalars and Vectors

Entities representing single bits — whether the bit is stored, changed, or transferred — are called “scalars.” Often multiple lines carry signals in a cluster — like data bus, address bus, and so on. Similarly, a group of regs stores a value, which may be assigned, changed, and handled together. The collection here is treated as a “vector.”

Figure below illustrates the difference between a scalar and a vector. `wr` and `rd` are two scalar nets connecting two circuit blocks `circuit1` and `circuit2`. `b` is a 4-bit-wide vector net connecting the same two blocks. `b[0]`, `b[1]`, `b[2]`, and `b[3]` are the individual bits of vector `b`. They are “part vectors.”

A vector reg or net is declared at the outset in a Verilog program and hence treated as such. The range of a vector is specified by a set of 2 digits (or expressions evaluating to a digit) with a colon in between the two. The combination is enclosed within square brackets.

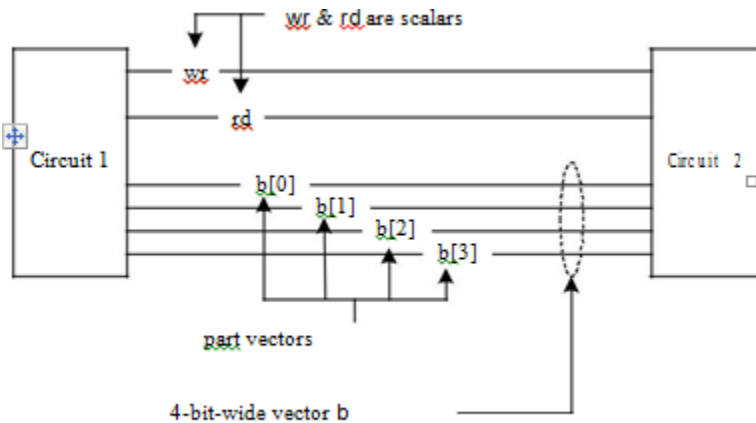


Figure Illustration of scalars and vectors.

Examples:

```
wire[3:0] a; /* a is a four bit vector of net type; the bits are designated as a[3], a[2], a[1] and a[0]. */
```

```
reg[2:0] b; /* b is a three bit vector of reg type; the bits are designated as b[2], b[1] and b[0]. */
```

```
reg[4:2] c; /* c is a three bit vector of reg type; the bits are designated as c[4], c[3] and c[2]. */
```

```
wire[-2:2] d; /* d is a 5 bit vector with individual bits designated as d[-2], d[-1], d[0], d[1] and d[2]. */
```

Whenever a range is not specified for a net or a reg, the same is treated as a scalar – a single bit quantity. In the range specification of a vector the most significant bit and the least significant bit can be assigned specific integer values. These can also be expressions evaluating to integer constants – positive or negative.

Normally vectors – nets or regs – are treated as unsigned quantities. They have to be specifically declared as “signed” if so desired.

#### Examples

```
wire signed[4:0] num;// num is a vector in the range -16 to +15.
```

```
reg signed [3:0] num_1; // num_1 is a vector in the range -8 to +7.
```

## Gate Level Modeling

### Introduction

Digital designers are normally familiar with all the common logic gates, their symbols, and their working. Flip-flops are built from the logic gates. All other functionally complex and more involved circuits can also be built using the basic gates. All the basic gates are available as “Primitives” in Verilog. Primitives are generalized modules that already exist in Verilog [IEEE]. They can be instantiated directly in other modules.

### And Gate Primitive

The AND gate primitive in Verilog is instantiated with the following statement:

```
and g1 (O, I1, I2, . . . , In);
```

Here ‘and’ is the keyword signifying an AND gate. g1 is the name assigned to the specific instantiation. O is the gate output; I1, I2, etc., are the gate inputs. The following are noteworthy:

- The AND module has only one output. The first port in the argument list is the output port.
- An AND gate instantiation can take any number of inputs — the upper limit is compiler-specific.
- A name need not be necessarily assigned to the AND gate instantiation; this is true of all the gate primitives available in Verilog.

### Truth Table of AND Gate Primitive

The truth table for a two-input AND gate is shown in Table below It can be directly extended to AND gate instantiations with multiple inputs. The following observations are in order here:

Truth table of AND gate primitive

		Input 1			
		0	1	X	z
Input 2	0	0	0	0	0
	1	0	1	X	x
	x	0	x	X	x
	z	0	x	X	x

- If any one of the inputs to the AND gate instantiation is in the 0 state, its output is also in the 0 state. It is irrespective of whether the other inputs are at the 0, 1, x or z state.
- The output is at 1 state if and only if every one of the inputs is at 1 state.
- For all other cases the output is at the x state.
- Note that the output is never at the z state – the high impedance state. This is true of all other gate primitives as well.

## Module Structure

In a general case a module can be more elaborate. A lot of flexibility is available in the definition of the body of the module. However, a few rules need to be followed:

- The first statement of a module starts with the keyword `module`; it may be followed by the name of the module and the port list if any.
- All the variables in the ports-list are to be identified as inputs, outputs, or inouts. The corresponding declarations have the form shown below:

```
f    Input a1, a2;
f    Output b1, b2;
f    Inout c1, c2;
```

The port-type declarations here follow the module declaration mentioned above.

- The ports and the other variables used within the body of the module are to be identified as nets or registers with specific types in each case. The respective declaration statements follow the port-type declaration statements.

Examples:

```
wire a1, a2, c;
reg b1, b2;
```

The type declaration must necessarily precede the first use of any variable or signal in the module.

- The executable body of the module follows the declaration indicated above.
- The last statement in any module definition is the keyword `“endmodule”`.
- Comments can appear anywhere in the module definition.



## Other Gate Primitives

All other basic gates are also available as primitives in Verilog. Details of the facilities and instantiations in each case are given in Table below. The following points are noteworthy here:

- In all cases of instantiations, one need not necessarily assign a name to the instantiation. It need be done only when felt necessary – say for clarity of circuit description.
- In all the cases the output port(s) is (are) declared first and the input port(s) is (are) declared subsequently.
- The buffer and the inverter have only one input each. They can have any number of outputs; the upper limit is compiler-specific. All other gates have one output each but can have any number of inputs; the upper limit is again compiler-specific.

Table for Basic gate primitives in Verilog with details

Gate	Mode of instantiation	Output port(s)	Input port(s)
AND	<b>and</b> ga ( o, i1, i2, . . . i8);	o	i1, i2, . .
OR	<b>or</b> gr ( o, i1, i2, . . . i8);	o	i1, i2, . .
NAND	<b>nand</b> gna ( o, i1, i2, . . . i8);	o	i1, i2, . .
NOR	<b>nor</b> gnr ( o, i1, i2, . . . i8);	o	i1, i2, . .
XOR	<b>xor</b> gxr ( o, i1, i2, . . . i8);	o	i1, i2, . .
XNOR	<b>xnor</b> gxn ( o, i1, i2, . . . i8);	o	i1, i2, . .
BUF	<b>buf</b> gb ( o1, o2, .... i);	o1, o2, o3, . .	i
NOT	<b>not</b> gn (o1, o2, o3, . . . i);	o1, o2, o3, . .	i

### Example for a typical A-O-I gate circuit

The commonly used A-O-I gate is shown in Figure 1 for a simple case. The module and the test bench for the same are given in Figure 2. The circuit has been realized here by instantiating the AND and NOR gate primitives. The names of signals and gates used in the instantiations in the module of Figure 2 remain the same as those in the circuit of Figure 1. The module `aoi_gate` in the figure has input and output ports since it describes a circuit with signal inputs and an output. The module `aoi_st` is a stimulus module. It generates inputs to the `aoi_gate` module and gets its output. It has no input or output ports.

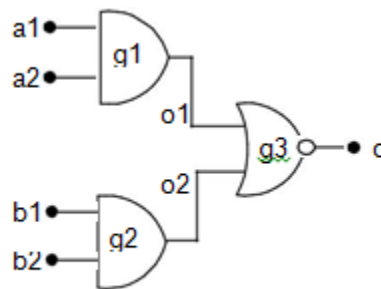


Figure for a typical A-O-I gate circuit.

```
/*module for the aoi-gate of figure 1 instantiating the gate primitives – fig 2*/
module aoi_gate(o,a1,a2,b1,b2);

input a1,a2,b1,b2; // a1,a2,b1,b2 form the input //ports of the module

output o; //o is the single output port of the module

wire o1,o2; //o1 and o2 are intermediate signals //within the module

and g1(o1,a1,a2); //The AND gate primitive has two and g2(o2,b1,b2);

// instantiations with assigned //names g1 & g2.

nor g3(o,o1,o2); //The nor gate has one instantiation with assigned name g3.

endmodule

//Test-bench for the aoi_gate above
module aoi_st;
reg a1,a2,b1,b2;

//specific values will be assigned to a1,a2,b1, // and b2 and these connected
//to input ports of the gate insatntiations;
```

```

//hence these variables are declared as reg
wire o;
initial
begin
a1 = 0;
a2 = 0;
b1 = 0;
b2 = 0;
#3 a1 = 1;
#3 a2 = 1;
#3 b1 = 1;
#3 b2 = 0;
#3 a1 = 1;
#3 a2 = 0;
#3 b1 = 0;
end
initial #100 $stop;//the simulation ends after //running for 100 tu's.
initial $monitor($time , " o = %b , a1 = %b , a2 = %b , b1 = %b ,b2 = %b ",o,a1,a2,b1,b2);
aoi_gate gg(o,a1,a2,b1,b2);
endmodule

```

### Tri-State Gates

Four types of tri-state buffers are available in Verilog as primitives. Their outputs can be turned ON or OFF by a control signal. The direct buffer is instantiated as Bufif1 nn (out, in, control);

The symbol of the buffer is shown in Figure

1. We have

- out as the single output variable
- in as the single input variable and
- control as the single control signal variable.

When  
control = 1,  
out = in.

When  
control = 0,  
out=tri-stated

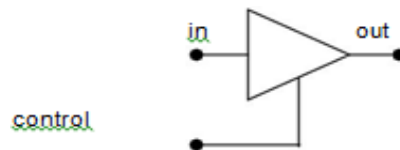
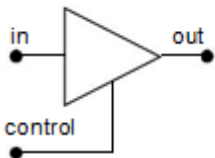
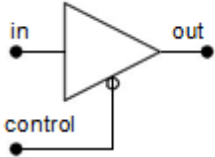
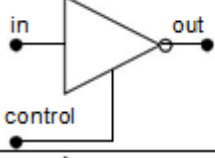
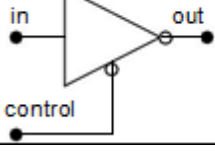


Figure 1 A tri-state buffer.

out is cut off from the input and tri-stated. The output, input and control signals should appear in the instantiation in the same order as above. Details of bufif1 as well as the other tri-state type primitives are shown in Table 1.

In all the cases shown in Table 1, out is the output; in is the input, and control, the control variable.

**Table 1 Instantiation and functional details of tri-state buffer primitives**

Typical instantiation	Functional representation	Functional description
<code>bufif1 (out, in, control);</code>		Out = in if control = 1; else out = z
<code>bufif0 (out, in, control);</code>		Out = in if control = 0; else out = z
<code>notif1 (out, in, control);</code>		Out = complement of in if control = 1; else out = z
<code>notif0 (out, in, control);</code>		Out = complement of in if control = 0; else out = z

### Array of Instances of Primitives

The primitives available in Verilog can also be instantiated as arrays. A judicious use of such array instantiations often leads to compact design descriptions. A typical array instantiation has the form

```
and gate [7 : 4 ] (a, b, c);
```

where a, b, and c are to be 4 bit vectors. The above instantiation is equivalent to combining the following 4 instantiations:

```
and gate [7] (a[3], b[3], c[3]), gate [6] (a[2], b[2], c[2]), gate [5] (a[1], b[1], c[1]), gate [4] (a[0], b[0], c[0]);
```

The assignment of different bits of input vectors to respective gates is implicit in the basic declaration itself. A more general instantiation of array type has the form

```
and gate[mm : nn](a, b, c);
```

where  $mm$  and  $nn$  can be expressions involving previously defined parameters, integers and algebra with them. The range for the gate is  $1+(mm-nn)$ ;  $mm$  and  $nn$  do not have restrictions of sign; either can be larger than the other.

## Gate Delays

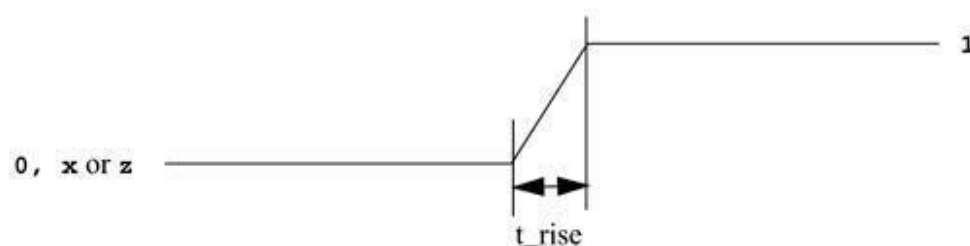
Until now, we described circuits without any delays (i.e., zero delay). In real circuits, logic gates have delays associated with them. Gate delays allow the Verilog user to specify delays through the logic circuits. Pin-to-pin delays can also be specified in Verilog.

### Rise, Fall, and Turn-off Delays

There are three types of delays from the inputs to the output of a primitive gate.

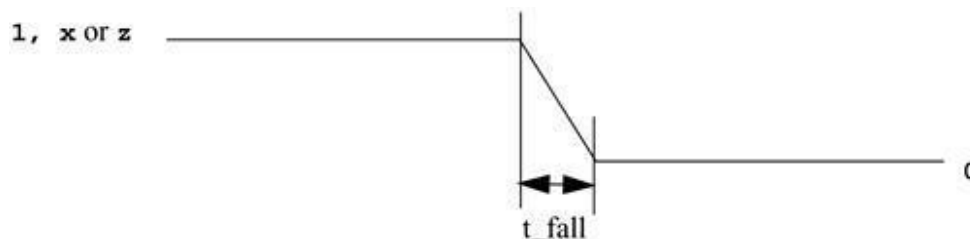
#### Rise delay

The rise delay is associated with a gate output transition to a 1 from another value.



#### Fall delay

The fall delay is associated with a gate output transition to a 0 from another value.



#### Turn-off delay

The turn-off delay is associated with a gate output transition to the high impedance value (z) from another value.

If the value changes to x, the minimum of the three delays is considered.

Three types of delay specifications are allowed. If only one delay is specified, this value is used for all transitions. If two delays are specified, they refer to the rise and fall delay values. The turn-off delay is the minimum of the two delays. If all three delays are specified, they refer to rise, fall, and turn-off delay values. If no delays are specified, the default value is zero.

## Example--Types of Delay Specification

```
//Delay of delay_time for all transitions
and #(delay_time) a1(out, i1, i2);

// Rise and Fall Delay Specification.

and #(rise_val, fall_val) a2(out, i1, i2);

// Rise, Fall, and Turn-off Delay Specification

bufif0 #(rise_val, fall_val, turnoff_val) b1 (out, in, control);
```

Examples of delay specification are shown below.

```
and #(5) a1(out, i1, i2); //Delay of 5 for all transitions and #(4,6) a2(out, i1, i2); // Rise
= 4, Fall = 6

bufif0 #(3,4,5) b1 (out, in, control); // Rise = 3, Fall = 4, Turn-off = 5
```

## Dataflow Modeling

### Introduction

For small circuits, the gate-level modeling approach works very well because the number of gates is limited and the designer can instantiate and connect every gate individually. Also, gate-level modeling is very intuitive to a designer with a basic knowledge of digital logic design. However, in complex designs the number of gates is very large. Thus, designers can design more effectively if they concentrate on implementing the function at a level of abstraction higher than gate level. Dataflow modeling provides a powerful way to implement a design. Verilog allows a circuit to be designed in terms of the data flow between registers and how a design processes data rather than instantiation of individual gates. Later in this chapter, the benefits of dataflow modeling will become more apparent.

With gate densities on chips increasing rapidly, dataflow modeling has assumed great importance. No longer can companies devote engineering resources to handcrafting entire designs with gates. Currently, automated tools are used to create a gate-level circuit from a dataflow design description. This process is called logic synthesis. Dataflow modeling has become a popular design approach as logic synthesis tools have become sophisticated. This approach allows the designer to concentrate on optimizing the circuit in terms of data flow. For maximum flexibility in the design process, designers typically use a Verilog description style that combines the concepts of gate-level, data flow, and behavioral design. In the digital design community, the term RTL (Register Transfer Level) design is commonly used for a combination of dataflow modeling and behavioral modeling.

## Continuous Assignments

A continuous assignment is the most basic statement in dataflow modeling, used to drive a value onto a net. This assignment replaces gates in the description of the circuit and describes the circuit at a higher level of abstraction. The assignment statement starts with the keyword `assign`. The syntax of an assign statement is as follows.

```
continuous_assign ::= assign [ drive_strength ] [ delay3 ]  
                    list_of_net_assignments ;
```

```
list_of_net_assignments ::= net_assignment { , net_assignment }
```

```
net_assignment ::= net_lvalue = expression
```

Notice that drive strength is optional and can be specified in terms of strength levels. The default value for drive strength is `strong1` and `strong0`. The delay value is also optional and can be used to specify delay on the assign statement. This is like specifying delays for gates. Delay specification is discussed in this chapter. Continuous assignments have the following characteristics:

1. The left hand side of an assignment must always be a scalar or vector net or a concatenation of scalar and vector nets. It cannot be a scalar or vector register.
2. Continuous assignments are always active. The assignment expression is evaluated as soon as one of the right-hand-side operands changes and the value is assigned to the left-hand-side net.
3. The operands on the right-hand side can be registers or nets or function calls. Registers or nets can be scalars or vectors.
4. Delay values can be specified for assignments in terms of time units. Delay values are used to control the time when a net is assigned the evaluated value. This feature is similar to specifying delays for gates. It is very useful in modeling timing behavior in real circuits.

## Examples of Continuous Assignment

```
Continuous assign. out is a net. i1 and i2 are nets. assign out = i1  
& i2;
```

```
Continuous assign for vector nets. addr is a 16-bit vector net
```

```
addr1 and addr2 are 16-bit vector registers.
```

```
assign addr[15:0] = addr1_bits[15:0] ^ addr2_bits[15:0];
```

Concatenation. Left-hand side is a concatenation of a scalar net and a vector net.

```
assign {c_out, sum[3:0]} = a[3:0] + b[3:0] + c_in;
```

### Implicit Continuous Assignment

Instead of declaring a net and then writing a continuous assignment on the net, Verilog provides a shortcut by which a continuous assignment can be placed on a net when it is declared. There can be only one implicit declaration assignment per net because a net is declared only once.

In the example below, an implicit continuous assignment is contrasted with a regular continuous assignment.

```
//Regular continuous assignment
```

```
wire out;
```

```
assign out = in1 & in2;
```

```
//Same effect is achieved by an implicit continuous assignment wire out =  
in1 & in2;
```

### Implicit Net Declaration

If a signal name is used to the left of the continuous assignment, an implicit net declaration will be inferred for that signal name. If the net is connected to a module port, the width of the inferred net is equal to the width of the module port.

```
wire i1, i2;
```

```
assign out = i1 & i2; //Note that out was not declared as a wire
```

```
                //but an implicit wire declaration for out //is done  
                by the simulator
```

### Delays

Delay values control the time between the change in a right-hand-side operand and when the new value is assigned to the left-hand side. Three ways of specifying delays



in continuous assignment statements are regular assignment delay, implicit continuous assignment delay, and net declaration delay.

### Regular Assignment Delay

The first method is to assign a delay value in a continuous assignment statement. The delay value is specified after the keyword assign. Any change in values of in1 or in2 will result in a delay of 10 time units before recomputation of the expression in1 & in2, and the result will be assigned to out. If in1 or in2 changes value again before 10 time units when the result propagates to out, the values of in1 and in2 at the time of recomputation are considered. This property is called inertial delay. An input pulse that is shorter than the delay of the assignment statement does not propagate to the output.

```
assign #10 out = in1 & in2; // Delay in a continuous assign
```

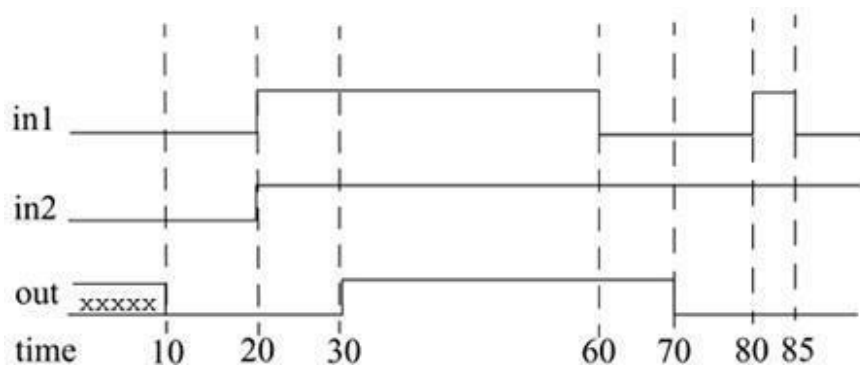


Figure: Delays

The above waveform is generated by simulating the above assign statement. It shows the delay on signal out. Note the following change:

When signals in1 and in2 go high at time 20, out goes to a high 10 time units later (time = 30).

When in1 goes low at 60, out changes to low at 70.

However, in1 changes to high at 80, but it goes down to low before 10 time units have elapsed.

Hence, at the time of recomputation, 10 units after time 80, in1 is 0. Thus, out gets the value 0. A pulse of width less than the specified assignment delay is not propagated to the output.

### **Implicit Continuous Assignment Delay**

An equivalent method is to use an implicit continuous assignment to specify both a delay and an assignment on the net.

```
//implicit continuous assignment delay
```

```
wire #10 out = in1 & in2;
```

```
//same as
```

```
wire out;
```

```
assign #10 out = in1 & in2;
```

The declaration above has the same effect as defining a wire out and declaring a continuous assignment on out.

### **Net Declaration Delay**

A delay can be specified on a net when it is declared without putting a continuous assignment on the net. If a delay is specified on a net out, then any value change applied to the net out is delayed accordingly. Net declaration delays can also be used in gate-level modeling.

```
//Net Delays
```

```
wire # 10 out;
```

```
assign out = in1 & in2;
```

```
//The above statement has the same effect as the following.
```

```
wire out;
```

```
assign #10 out = in1 & in2;
```

## Expressions, Operators, and Operands

Dataflow modeling describes the design in terms of expressions instead of primitive gates. Expressions, operators, and operands form the basis of dataflow modeling.

### Expressions

Expressions are constructs that combine operators and operands to produce a result.

Examples of expressions. Combines operands and operators  $a \wedge b$

```
addr1[20:17] + addr2[20:17] in1 | in2 ;
```

### Operands

Some constructs will take only certain types of operands. Operands can be constants, integers, real numbers, nets, registers, times, bit-select (one bit of vector net or a vector register), part-select (selected bits of the vector net or register vector), and memories or function calls (functions are discussed later).

```
integer count, final_count;
```

```
final_count = count + 1; //count is an integer operand
```

```
real a, b, c;
```

```
c = a - b; //a and b are real operands
```

```
reg [15:0] reg1, reg2;
```

```
reg [3:0] reg_out;
```

```
reg_out = reg1[3:0] ^ reg2[3:0]; //reg1[3:0] and reg2[3:0] are //part-select  
register operands
```

```
reg ret_value;
```

```
ret_value = calculate_parity(A, B); //calculate_parity is a //function  
type operand
```

## Operators

Operators act on the operands to produce desired results. Verilog provides various types of operators.

`d1 && d2` // `&&` is an operator on operands `d1` and `d2`  
`!a[0]`  
// `!` is an operator on operand `a[0]`

`B >> 1` // `>>` is an operator on operands `B` and `1`

## Operator Types

Verilog provides many different operator types. Operators can be arithmetic, logical, relational, equality, bitwise, reduction, shift, concatenation, or conditional. Some of these operators are similar to the operators used in the C programming language. Each operator type is denoted by a symbol. The following table shows the complete listing of operator symbols classified by category.

Table: Operator Types and Symbols

Operator Type	Operator Symbol	Operation Performed	Number of Operands
Arithmetic	*	multiply	two
	/	divide	two
	+	add	two
	-	subtract	two
	%	modulus	two
	**	power (exponent)	two
Logical	!	logical negation	one
	&&	logical and	two
		logical or	two
	>	greater than	two

Relational	<	less than	two
	>=	greater than or equal	two
	<=	less than or equal	two
Equality	==	equality	two
	!=	inequality	two
	===	case equality	two
	!==	case inequality	two

Bitwise	~	bitwise negation	one
	&	bitwise and	two
		bitwise or	two
	^	bitwise xor	two
	^~ or ~^	bitwise xnor	two
Reduction	&	reduction and	one
	~&	reduction nand	one
		reduction or	one
	~	reduction nor	one
	^	reduction xor	one
	^~ or ~^	reduction xnor	one

Shift	>>	Right shift	Two
	<<	Left shift	Two
	>>>	Arithmetic right shift	Two
	<<<	Arithmetic left shift	Two
Concatenation	{ }	Concatenation	Any number
Replication	{ { } }	Replication	Any number
Conditional	?:	Conditional	Three

Let us now discuss each operator type in detail.

### Arithmetic Operators

There are two types of arithmetic operators: binary and unary.

#### Binary operators

Binary arithmetic operators are multiply (\*), divide (/), add (+), subtract (-), power (\*\*), and modulus (%). Binary operators take two operands.

```
A = 4'b0011; B = 4'b0100; // A and B are register vectors
D = 6; E = 4; F=2 // D and E are integers
```

```
A * B // Multiply A and B. Evaluates to 4'b1100
```

```
D / E // Divide D by E. Evaluates to 1. Truncates any fractional part.
A + B // Add A and B. Evaluates to 4'b0111
```

```
B - A // Subtract A from B. Evaluates to 4'b0001 F = E **  
F; //E to the power F, yields 16
```

If any operand bit has a value x, then the result of the entire expression is x. This seems intuitive because if an operand value is not known precisely, the result should be an unknown.

```
in1 = 4'b101x;
```

```
in2 = 4'b1010;
```

```
sum = in1 + in2; // sum will be evaluated to the value 4'bx
```

Modulus operators produce the remainder from the division of two numbers. They operate similarly to the modulus operator in the C programming language.

```
13 % 3 // Evaluates to 1
```

```
16 % 4 // Evaluates to 0
```

```
-7 % 2 // Evaluates to -1, takes sign of the first operand
```

```
7 % -2 // Evaluates to +1, takes sign of the first operand
```

### **Unary operators**

The operators + and - can also work as unary operators. They are used to specify the positive or negative sign of the operand. Unary + or ? operators have higher precedence than the binary + or ? operators.

```
-4 // Negative 4
```

```
+5 // Positive 5
```

Negative numbers are represented as 2's complement internally in Verilog. It is advisable to use negative numbers only of the type integer or real in expressions. Designers should avoid negative numbers of the type <sss> '<base> <nnn> in expressions because they are converted to unsigned 2's complement numbers and hence yield unexpected results.

```
//Advisable to use integer or real numbers -10 /
```

```
5// Evaluates to -2
```

```
//Do not use numbers of type <sss> '<base> <nnn>
```

```
-'d10 / 5// Is equivalent (2's complement of 10)/5 = (232 - 10)/5
```

where 32 is the default machine word width.

This evaluates to an incorrect and unexpected result

### Logical Operators

Logical operators are logical-and (&&), logical-or (||) and logical-not (!). Operators && and || are binary operators. Operator ! is a unary operator. Logical operators follow these conditions:

Logical operators always evaluate to a 1-bit value, 0 (false), 1 (true), or x (ambiguous). If an operand is not equal to zero, it is equivalent to a logical 1 (true condition). If it is equal to zero, it is equivalent to a logical 0 (false condition). If any operand bit is x or z, it is equivalent to x (ambiguous condition) and is normally treated by simulators as a false condition. Logical operators take variables or expressions as operands. Use of parentheses to group logical operations is highly recommended to improve readability. Also, the user does not have to remember the precedence of operators.

```
Logical operations A = 3;
```

```
B = 0;
```

```
A && B // Evaluates to 0. Equivalent to (logical-1 && logical-0) A || B //
```

```
Evaluates to 1. Equivalent to (logical-1 || logical-0) !A// Evaluates to 0.
```

```
Equivalent to not(logical-1)
```

```
!B// Evaluates to 1. Equivalent to not(logical-0)
```

### Unknowns

```
A = 2'b0x; B = 2'b10;
```

```
A && B // Evaluates to x. Equivalent to (x && logical 1)
```



```
// Expressions
```

```
(a == 2) && (b == 3) // Evaluates to 1 if both a == 2 and b == 3 are true.
```

```
// Evaluates to 0 if either is false.
```

## Relational Operators

Relational operators are greater-than (>), less-than (<), greater-than-or-equal-to (>=), and less-than-or-equal-to (<=). If relational operators are used in an expression, the expression returns a logical value of 1 if the expression is true and 0 if the expression is false. If there are any unknown or z bits in the operands, the expression takes a value x. These operators function exactly as the corresponding operators in the C programming language.

```
A = 4, B = 3
```

```
X = 4'b1010, Y = 4'b1101, Z = 4'b1xxx
```

```
A <= B // Evaluates to a logical 0
```

```
A > B // Evaluates to a logical 1
```

```
Y >= X // Evaluates to a logical 1
```

```
Y < Z // Evaluates to an x
```

## Equality Operators

Equality operators are logical equality (==), logical inequality (!=), case equality (===), and case inequality (!==). When used in an expression, equality operators return logical value 1 if true, 0 if false. These operators compare the two operands bit by bit, with zero filling if the operands are of unequal length. Table below lists the operators.

It is important to note the difference between the logical equality operators (==, !=) and case equality operators (===, !==). The logical equality operators (==, !=) will yield an x if either operand has x or z in its bits. However, the case equality operators (===, !==) compare both operands bit by bit and compare all bits, including x and z. The result is 1 if the operands match exactly, including x and z bits. The result is 0 if the operands do not match exactly. Case equality operators never result in an x.

**Table: Equality Operators**

Expression	Description	Possible Logical Value
<code>a == b</code>	a equal to b, result unknown if x or z in a or b	0, 1, x
<code>a != b</code>	a not equal to b, result unknown if x or z in a or B	0, 1, x
<code>a === b</code>	a equal to b, including x and z	0, 1
<code>a !== b</code>	a not equal to b, including x and z	0, 1

`A = 4, B = 3`

`X = 4'b1010, Y = 4'b1101`

`Z = 4'b1xxz, M = 4'b1xxz, N = 4'b1xxx`

`A == B // Results in logical 0`

`X != Y // Results in logical 1`

`X == Z // Results in x`

`Z === M // Results in logical 1 (all bits match, including x and z)`

`Z === N // Results in logical 0 (least significant bit does not match) M !== N // Results in logical 1`

## Bitwise Operators

Bitwise operators are negation ( $\sim$ ), and ( $\&$ ), or ( $|$ ), xor ( $\wedge$ ), xnor ( $\wedge\sim$ ,  $\sim\wedge$ ). Bitwise operators perform a bit-by-bit operation on two operands. They take each bit in one operand and perform the operation with the corresponding bit in the other operand. If one operand is shorter than the other, it will be bit-extended with zeros to match the length of the longer operand. Logic tables for the bit-by-bit computation are shown in Table. A z is treated as an x in a bitwise operation. The exception is the unary negation operator ( $\sim$ ), which takes only one operand and operates on the bits of the single operand.

**Table: Truth Tables for Bitwise Operators**

bitwise and	0	1	x
0	0	0	0
1	0	1	x
x	0	x	x

bitwise or	0	1	x
0	0	1	x
1	1	1	1
x	x	1	x

bitwise xor	0	1	x
0	0	1	x
1	1	0	x
x	x	x	x

bitwise xnor	0	1	x
0	1	0	x
1	0	1	x
x	x	x	x

bitwise negation	result
0	1
1	0
x	x

Examples of bitwise operators are shown below.

X = 4'b1010, Y = 4'b1101

Z = 4'b10x1

$\sim$ X // Negation. Result is 4'b0101

X & Y // Bitwise and. Result is 4'b1000

X | Y // Bitwise or. Result is 4'b1111

```
X ^ Y // Bitwise xor. Result is 4'b0111
```

```
X ^~ Y // Bitwise xnor. Result is 4'b1000
```

```
X & Z // Result is 4'b10x0
```

It is important to distinguish bitwise operators  $\sim$ ,  $\&$ , and  $|$  from logical operators  $!$ ,  $\&\&$ ,  $||$ . Logical operators always yield a logical value 0, 1, x, whereas bitwise operators yield a bit-by-bit value. Logical operators perform a logical operation, not a bit-by-bit operation.

```
// X = 4'b1010, Y = 4'b0000
```

```
X | Y // bitwise operation. Result is 4'b1010
```

```
X || Y // logical operation. Equivalent to 1 || 0. Result is 1.
```

### Reduction Operators

Reduction operators are and ( $\&$ ), nand ( $\sim\&$ ), or ( $|$ ), nor ( $\sim|$ ), xor ( $\wedge$ ), and xnor ( $\sim\wedge$ ,  $\wedge\sim$ ). Reduction operators take only one operand. Reduction operators perform a bitwise operation on a single vector operand and yield a 1-bit result. The difference is that bitwise operations are on bits from two different operands, whereas reduction operations are on the bits of the same operand. Reduction operators work bit by bit from right to left. Reduction nand, reduction nor, and reduction xnor are computed by inverting the result of the reduction and, reduction or, and reduction xor, respectively.

```
// X = 4'b1010
```

```
&X //Equivalent to 1 & 0 & 1 & 0. Results in 1'b0
```

```
|X //Equivalent to 1 | 0 | 1 | 0. Results in 1'b1
```

```
^X //Equivalent to 1 ^ 0 ^ 1 ^ 0. Results in 1'b0
```

```
//A reduction xor or xnor can be used for even or odd parity
```

```
//generation of a vector.
```

The use of a similar set of symbols for logical (!, &&, ||), bitwise (~, &, |, ^), and reduction operators (&, |, ^) is somewhat confusing initially. The difference lies in the number of operands each operator takes and also the value of results computed.

### Shift Operators

Shift operators are right shift ( >> ), left shift ( << ), arithmetic right shift ( >>> ), and arithmetic left shift ( <<< ). Regular shift operators shift a vector operand to the right or the left by a specified number of bits. The operands are the vector and the number of bits to shift. When the bits are shifted, the vacant bit positions are filled with zeros. Shift operations do not wrap around. Arithmetic shift operators use the context of the expression to determine the value with which to fill the vacated bits.

```
// X = 4'b1100
```

```
Y = X >> 1; //Y is 4'b0110. Shift right 1 bit. 0 filled in MSB position.
```

```
Y = X << 1; //Y is 4'b1000. Shift left 1 bit. 0 filled in LSB position.
```

```
Y = X << 2; //Y is 4'b0000. Shift left 2 bits.
```

```
integer a, b, c; //Signed data types
```

```
a = 0;
```

```
b = -10; // 00111...10110 binary
```

```
c = a + (b >>> 3); //Results in -2 decimal, due to arithmetic shift
```

Shift operators are useful because they allow the designer to model shift operations, shift-and-add algorithms for multiplication, and other useful operations.

### Concatenation Operator

The concatenation operator ( { , } ) provides a mechanism to append multiple operands. The operands must be sized. Unsized operands are not allowed because the size of each operand must be known for computation of the size of the result. Concatenations are expressed as operands within braces, with commas separating the operands. Operands can be scalar nets or registers, vector nets or registers, bit-select, part-select, or sized constants.

```
// A = 1'b1, B = 2'b00, C = 2'b10, D = 3'b110
Y = {B , C} // Result Y is 4'b0010
Y = {A , B , C , D , 3'b001} // Result Y is 11'b10010110001
Y = {A , B[0], C[1]} // Result Y is 3'b101
```

### Replication Operator

Repetitive concatenation of the same number can be expressed by using a replication constant. A replication constant specifies how many times to replicate the number inside the brackets ( { } ).

```
reg A;
```

```
reg [1:0] B, C;
```

```
reg [2:0] D;
```

```
A = 1'b1; B = 2'b00; C = 2'b10; D = 3'b110;
```

```
Y = { 4{A} } // Result Y is 4'b1111
```

```
Y = { 4{A} , 2{B} } // Result Y is 8'b11110000
```

```
Y = { 4{A} , 2{B} , C } // Result Y is 8'b1111000010
```

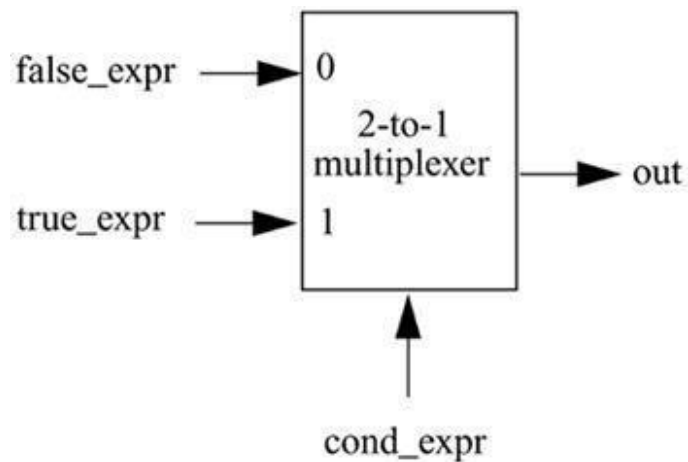
### Conditional Operator

The conditional operator(?:) takes three operands.

Usage: condition\_expr ? true\_expr : false\_expr ;

The condition expression (condition\_expr) is first evaluated. If the result is true (logical 1), then the true\_expr is evaluated. If the result is false (logical 0), then the false\_expr is evaluated. If the result is x (ambiguous), then both true\_expr and false\_expr are evaluated and their results are compared, bit by bit, to return for each bit position an x if the bits are different and the value of the bits if they are the same.

The action of a conditional operator is similar to a multiplexer. Alternately, it can be compared to the if-else expression.



Conditional operators are frequently used in dataflow modeling to model conditional assignments. The conditional expression acts as a switching control.

```
//model functionality of a tristate buffer
```

```
assign addr_bus = drive_enable ? addr_out : 36'bz;
```

```
//model functionality of a 2-to-1 mux
```

```
assign out = control ? in1 : in0;
```

Conditional operations can be nested. Each `true_expr` or `false_expr` can itself be a conditional operation. In the example that follows, convince yourself that  $(A==3)$  and `control` are the two select signals of 4-to-1 multiplexer with `n`, `m`, `y`, `x` as the inputs and `out` as the output signal.

```
assign out = (A == 3) ? ( control ? x : y) : ( control ? m : n) ;
```

### Operator Precedence

Having discussed the operators, it is now important to discuss operator precedence. If no parentheses are used to separate parts of expressions, Verilog enforces the following precedence. Operators listed in Table are in order from highest precedence to lowest precedence. It is recommended that parentheses be used to separate expressions except in case of unary operators or when there is no ambiguity.

**Table: Operator Precedence**

<b>Operators</b>	<b>Operator Symbols</b>	<b>Precedence</b>
Unary	+ - ! ~	Highest precedence
Multiply, Divide, Modulus	* / %	
Add, Subtract	+ -	
Shift	<< >>	
Relational	< <= > >=	
Equality	== != === !==	
Reduction	&, ~& ^ ^~  , ~	
Logical	&& 	
Conditional	?:	Lowest precedence



# Behavioral Modeling

## Introduction

Behavioral modeling is the highest level of abstraction in the Verilog HDL. The other modeling techniques are relatively detailed. They require some knowledge of how hardware or hardware signals work. The abstraction in this modeling is as simple as writing the logic in C language. This is a very powerful abstraction technique. All that a designer need is the algorithm of the design, which is the basic information for any design.

Most of the behavioral modeling is done using two important constructs: initial and always. All the other behavioral statements appear only inside these two structured procedure constructs.

## The Initial Construct

The statements which come under the initial construct constitute the initial block. The initial block is executed only once in the simulation, at time 0. If there is more than one initial block, then all the initial blocks are executed concurrently. The initial construct is used as follows:

```
initial
begin
reset=1'b0;
clk=1'b1;
end
or
initial
clk = 1'b1;
```

In the first initial block there are more than one statements hence they are written between begin and end. If there is only one statement then there is no need to put begin and end.

## The always construct

The statements which come under the always construct constitute the always block. The always block starts at time 0, and keeps on executing all the simulation time. It works like a infinite loop. It is generally used to model a functionality that is continuously repeated.

```
always
#5clk=~clk;
initial
clk = 1'b0;
```

The above code generates a clock signal clk, with a time period of 10 units. The initial blocks initiates the clk value to 0 at time 0. Then after every 5 units of time it toggled, hence we get a

time period of 10 units. This is the way in general used to generate a clock signal for use in test benches.

```
always@(posedge clk, negedge reset)
begin
a = b + c;
  d = 1'b1;
end
```

In the above example, the always block will be executed whenever there is a positive edge in the clk signal, or there is negative edge in the reset signal. This type of always is generally used in implement a FSM, which has a reset signal.

```
always @(b,c,d)
begin
  a = ( b + c ) * d;
  e = b | c;
end
```

In the above example, whenever there is a change in b, c, or d the always block will be executed. Here the list b, c, and d is called the sensitivity list.

In the Verilog 2000, we can replace always @(b,c,d) with always @(\*), it is equivalent to include all input signals, used in the always block. This is very useful when always blocks are used for implementing the combination logic.

## **OPERATIONS AND ASSIGNMENTS:**

The design description at the behavioral level is done through a sequence of assignments. These are called 'procedural assignments' – in contrast to the continuous assignments at the data flow level. Though it appears similar to the assignments at the data flow level discussed in the last chapter, the two are different. The procedure assignment is characterized by the following:

- The assignment is done through the "=" symbol (or the "<=" symbol) as was the case with the continuous assignment earlier.
- An operation is carried out and the result assigned through the "=" operator to an operand specified on the left side of the "=" sign – for example,  $N = \sim N$ ;
- Here the content of reg N is complemented and assigned to the reg N itself. The assignment is essentially an updating activity.
- The operation on the right can involve operands and operators. The operands can be of different types – logical variables, numbers – real or integer and so on.

### **wait CONSTRUCT**

The wait construct makes the simulator wait for the specified expression to be true before proceeding with the following assignment or group of assignments. Its syntax has the form

```
wait (alpha) assignment1;
```

alpha can be a variable, the value on a net, or an expression involving them. If alpha is an expression, it is evaluated; if true, assignment1 is carried out. One can also have a group of assignments within a block in place of assignment1.

Example:

```
wait(clk) #2 a = b;
```

The simulator waits for the clock to be high and then assigns b to a with a delay of 2 ns. The assignment will be refreshed as long as the clk remains high.

The below code shows one version of the up-down counter module along with a test bench. It is a modification of the up down counter uses a wait construct. It has an enable input En. The counter is active and counts only when En = 1.

#### **Verilog code:**

```
module ctr_wt(a,clk,N,En);
input clk,En;
input[3:0]N;
output[3:0]a;
reg[3:0]a;
initial a=4'b1111;
always
begin
wait(En)
@(negedge clk)
a=(a==N)?4'b0000:a+1'b1;
end
endmodule
```

#### **Test Bench**

```
module tst_ctr_wt;
reg clk,En;
reg[3:0]N;
wire[3:0]a;
ctr_wt c1(a,clk,N,En);
initial
begin
clk=0;N=4'b1111;En=1'b0;#5 En=1'b1;#20 En=1'b0;
end
always
#2 clk=~clk;
initial #35 $stop;
initial $monitor($time,"clk=%h,En=%b,N=%b,a=%b",clk,En,N,a,);
endmodule
```

## Procedural Assignments

Procedural assignments are used for updating reg, integer, time, real, realtime, and memory data types. The variables will retain their values until updated by another procedural assignment. There is a significant difference between procedural assignments and continuous assignments. Continuous assignments drive nets and are evaluated and updated whenever an input operand changes value. Whereas procedural assignments update the value of variables under the control of the procedural flow constructs that surround them.

The LHS of a procedural assignment could be:

- reg, integer, real, realtime, or time data type.
- Bit-select of a reg, integer, or time data type, rest of the bits are untouched.
- Part-select of a reg, integer, or time data type, rest of the bits are untouched.
- Memory word.

Concatenation of any of the previous four forms can be specified.

When the RHS evaluates to fewer bits than the LHS, then if the right-hand side is signed, it will be sign-extended to the size of the left-hand side.

There are two types of procedural assignments: blocking and non-blocking assignments.

**Blocking assignments:** A blocking assignment statements are executed in the order they are specified in a sequential block. The execution of next statement begins only after the completion of the present blocking assignments. A blocking assignment will not block the execution of the next statement in a parallel block. The blocking assignments are made using the operator =.

```
initial
begin
  a = 1;
  b = #5 2;
  c = #2 3;
end
```

In the above example, a is assigned value 1 at time 0, and b is assigned value 2 at time 5, and c is assigned value 3 at time 7.

**Non-blocking assignments:** The nonblocking assignment allows assignment scheduling without blocking the procedural flow. The nonblocking assignment statement can be used whenever several variable assignments within the same time step can be made without regard to order or dependence upon each other. Non-blocking assignments are made using the operator <=.

Note: <= is same for less than or equal to operator, so whenever it appears in a expression it is considered to be comparison operator and not as non-blocking assignment.

```
initial
begin
  a <= 1;
  b <= #5 2;
  c <= #2 3;
end
```

In the above example, a is assigned value 1 at time 0, and b is assigned value 2 at time 5, and c is assigned value 3 at time 2 (because all the statements execution starts at time 0, as they are non-blocking assignments).

### Conditional (if-else) Statement

The condition (if-else) statement is used to make a decision whether a statement is executed or not. The keywords if and else are used to make conditional statement. The conditional statement can appear in the following forms.

```
if ( condition_1 )
  statement_1;

if ( condition_2 )
  statement_2;
else
  statement_3;

if ( condition_3 )
  statement_4;
else if ( condition_4 )
  statement_5;
else
  statement_6;

if ( condition_5 )
begin
  statement_7;
  statement_8;
end
else
begin
  statement_9;
```

```
    statement_10;  
end
```

Conditional (if-else) statement usage is similar to that if-else statement of C programming language, except that parenthesis are replaced by begin and end.

### **Case Statement**

The case statement is a multi-way decision statement that tests whether an expression matches one of the expressions and branches accordingly. Keywords case and endcase are used to make a case statement. The case statement syntax is as follows.

```
case (expression)  
    case_item_1: statement_1;  
    case_item_2: statement_2;  
    case_item_3: statement_3;  
    ...  
    ...  
    default: default_statement;  
endcase
```

If there are multiple statements under a single match, then they are grouped using begin, and end keywords. The default item is optional.

Case statement with don't cares: casez and casex

casez treats high-impedance values (z) as don't cares. casex treats both high-impedance (z) and unknown (x) values as don't cares. Don't-care values (z values for casez, z and x values for casex) in any bit of either the case expression or the case items shall be treated as don't-care conditions during the comparison, and that bit position shall not be considered. The don't cares are represented using the ? mark.

## Loop Statements

There are four types of looping statements in Verilog:

```
forever  
repeat  
while  
for
```

### Forever Loop

Forever loop is defined using the keyword `forever`, which continuously executes a statement. It terminates when the system task `$finish` is called. A forever loop can also be ended by using the `disable` statement.

```
initial  
begin  
    clk = 1'b0;  
    forever #5 clk = ~clk;  
end
```

In the above example, a clock signal with time period 10 units of time is obtained.

### Repeat Loop

Repeat loop is defined using the keyword `repeat`. The repeat loop block continuously executes the block for a given number of times. The number of times the loop executes can be mentioned using a constant or an expression. The expression is calculated only once, before the start of loop and not during the execution of the loop. If the expression value turns out to be zero, then it is treated as zero, and hence loop block is not executed at all.

```
initial  
begin  
    a = 10;  
    b = 5;  
    b <= #10 10;  
    i = 0;  
    repeat(a*b)  
    begin  
        $display("repeat in progress");  
        #1 i = i + 1;  
    end  
end
```

```
end  
end
```

In the above example the loop block is executed only 50 times, and not 100 times. It calculates (a\*b) at the beginning, and uses that value only.

## While Loop

The while loop is defined using the keyword while. The while loop contains an expression. The loop continues until the expression is true. It terminates when the expression is false. If the calculated value of expression is z or x, it is treated as a false. The value of expression is calculated each time before starting the loop. All the statements (if more than one) are mentioned in blocks which begins and ends with keyword begin and end keywords.

```
initial  
begin  
  a = 20;  
  i = 0;  
  while (i < a)  
  begin  
    $display("%d",i);  
    i = i + 1;  
    a = a - 1;  
  end  
end
```

In the above example the loop executes for 10 times. (Observe that a is decrementing by one and i is incrementing by one, so loop terminated when both i and a become 10).

## For Loop

The For loop is defined using the keyword for. The execution of for loop block is controlled by a three step process, as follows:

Executes an assignment, normally used to initialize a variable that controls the number of times the for block is executed.

Evaluates an expression, if the result is false or z or x, the for-loop shall terminate, and if it is true, the for-loop shall execute its block.

Executes an assignment normally used to modify the value of the loop-control variable and then repeats with second step.



Note that the first step is executed only once.

```
initial
begin
  a = 20;
  for (i = 0; i < a; i = i + 1, a = a - 1)
    $display("%d",i);
end
```

The above example produces the same result as the example used to illustrate the functionality of the while loop.

Examples:

1. Implementation of a 4x1 multiplexer.

```
module mux4_1 (out, in0, in1, in2, in3, s0, s1);
```

```
output out;
```

```
// out is declared as reg, as default is wire
```

```
reg out;
```

```
// out is declared as reg, because we will
```

```
// do a procedural assignment to it.
```

```
input in0, in1, in2, in3, s0, s1;
```

```
// always @(*) is equivalent to
```

```
// always @( in0, in1, in2, in3, s0, s1 )
```

```
always @(*)
```

```
begin
```

```
  case ({s1,s0})
```

```
    2'b00: out = in0;
```

```
    2'b01: out = in1;
```

```
    2'b10: out = in2;
```

```
    2'b11: out = in3;
```

```
    default: out = 1'bx;
```

```
  endcase
```

```
end
```

```
endmodule
```

## 2. Implementation of a full adder.

```
module full_adder (sum, c_out, in0, in1, c_in);

output sum, c_out;
reg sum, c_out

input in0, in1, c_in;

always @(*)
  {c_out, sum} = in0 + in1 + c_in;

endmodule
```

## 3. Implementation of a 8-bit binary counter.

```
module ( count, reset, clk );

output [7:0] count;
reg [7:0] count;

input reset, clk;

// consider reset as active low signal

always @( posedge clk, negedge reset)
begin
  if(reset == 1'b0)
    count <= 8'h00;
  else
    count <= count + 8'h01;
end

endmodule
```

Implementation of a 8-bit counter is a very good example, which explains the advantage of behavioral modeling. Just imagine how difficult it will be implementing a 8-bit counter using gate-level modeling. In the above example the incrementation occurs on every positive edge of the clock. When count becomes 8'hFF, the next increment will make it 8'h00, hence there is no need of any modulus operator. Reset signal is active low.

## Block Statements

Block statements are used to group two or more statements together, so that they act as one statement. There are two types of blocks:

- Sequential block.
- Parallel block.

### Sequential block:

The sequential block is defined using the keywords `begin` and `end`. The procedural statements in sequential block will be executed sequentially in the given order. In sequential block delay values for each statement shall be treated relative to the simulation time of the execution of the previous statement. The control will pass out of the block after the execution of last statement.

### Parallel block:

The parallel block is defined using the keywords `fork` and `join`. The procedural statements in parallel block will be executed concurrently. In parallel block delay values for each statement are considered to be relative to the simulation time of entering the block. The delay control can be used to provide time-ordering for procedural assignments.

The control shall pass out of the block after the execution of the last time-ordered statement. Note that blocks can be nested. The sequential and parallel blocks can be mixed.

### Block names:

All the blocks can be named, by adding `: block_name` after the keyword `begin` or `fork`.

The advantages of naming a block are:

It allows to declare local variables, which can be accessed by using hierarchical name referencing. They can be disabled using the `disable` statement (`disable block_name;`).

## SWITCH LEVEL MODELING

### Introduction

In today's environment the MOS transistor is the basic element around which a VLSI is built. Designers familiar with logic gates and their configurations at the circuit level may choose to do their designs using MOS transistors.

Verilog has the provision to do the design description at the switch level using such MOS transistors.

Switch level modeling forms the basic level of modeling digital circuits.

The switches are available as primitives in Verilog; they are central to design description at this level. Basic gates can be defined in terms of such switches. By repeated and successive instantiation of such switches, more involved circuits can be modeled – on the same lines as was done with the gate level primitives.

### BASIC TRANSISTOR SWITCHES

Consider an NMOS transistor of the depletion type. When used in a digital circuit, it can be in one of three modes:

- $V_G < V_S$  where  $V_G$  and  $V_S$  are the gate and source voltages with respect to the drain: The transistor is OFF and offers very high impedance across the source and the drain. It is in the z state.
- $V_G = V_S$ : The transistor is in the active region. It presents a resistance between the source and the drain. The value depends on the technology. Such a resistive state of the transistor can be modeled in Verilog. A transistor in this mode can be represented as a resistance in Verilog – as pull1 or pull0 depending on whether the drain is connected to supply1 or source is connected to supply0.
- $V_G > V_S$ : The transistor is fully turned on. It presents very low resistance between the source and drain.
- An enhanced-mode NMOS transistor also has the above three modes of operation.
- It is OFF when  $V_G = V_S$ . It is moderately ON or in the active region when  $V_G$  is slightly greater than  $V_S$ , representing a resistive (pull1 or pull0) mode of operation. When  $V_G$  is sufficiently greater than  $V_S$ , the transistor is in the on state representing very low resistance. Similar modes are possible for the PMOS transistor also.

### Basic Switch Primitive

Different switch primitives are available in Verilog.

Consider an nmos switch. A typical instantiation has the form

```
nmos (out, in, control);
```

nmos – a keyword – represents an NMOS transistor functioning as a switch.

The switch has three terminals – in, out, and control.

NMOS transistor symbol shown in below figure 1 with three terminals-

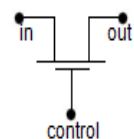


Figure 1 An NMOS switch with terminals

When the control input is at 1 (high) state, the switch is on. It connects the input lead to the output side and offers zero impedance.

When the control input is low, the switch is OFF and output is left floating (z state).

If the control is in the z or the x state, output may take corresponding values.

The keyword `pmos` represents a PMOS transistor functioning as a switch.

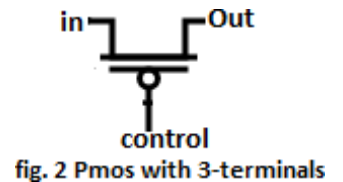
The PMOS switch has three terminals (see Figure 2).

A typical instantiation of the switch has the form

```
pmos (out, in, control);
```

When the control is at 1 (high) state, the switch is off. Output is left floating.

When control is at 0 (low) state, the switch is on, input is connected to output, and output is at the same state as input.



## Resistive Switches

`nmos` and `pmos` represent switches of low impedance in the on-state. `rnmos` and `rpmos` represent the resistive counterparts of these respectively. Typical instantiations have the form

```
rnmos (output1, input1, control1);
```

```
rpmos (output2, input2, control2);
```

The `rnmos` if the `control1` input is at 1 (high) state, the switch is ON and functions as a definite resistance. It connects `input1` to `output1` through a resistance. When `control1` is at the 0 (low) state, the switch is OFF and leaves `output1` floating.

The `rpmos` switch is ON when `control2` is at 0 (low) state. It inserts a definite resistance between the input and the output signals but retains the signal value.

`rpmos` and `rnmos` are resistive switches, they reduce the signal strength when in the on state. The reduced strength is mostly one level below the original strength.

The `rpmos` and `rnmos` switches function as unidirectional switches; the signal flow is from the input to the output side.

## pullup and pulldown

A MOS transistor functions as a resistive element when in the active state. Realization of resistance in this form takes less silicon area in the IC as compared to a resistance realized directly. `pullup` and `pulldown` represent such resistive elements.

A typical instantiation here has the form

```
pullup (x);
```

Here the net `x` is pulled up to the `supply1` through a resistance. Similarly, the instantiation

```
pulldown (y);
```

pulls `y` down to the `supply0` level through a resistance. The `pullup` and `pulldown` primitives can be used as loads for switches or to connect the unused input ports to VCC or GND, respectively. They can also form loads of switches in logic circuits.

The default strengths for pullup and pulldown are pull1 and pull0 respectively. One can also specify strength values for the respective nets. For example,

```
pullup (strong1) (x)
```

specifies a resistive pullup of net x to supply1. One can also assign names to the pullup and pulldown primitives. Thus

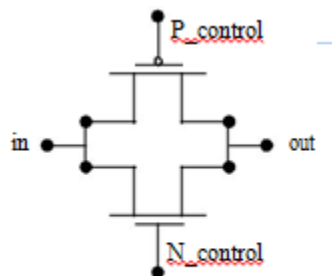
```
pullup (strong1) rs(x)
```

represents an instantiation of pullup designated rs having strength strong1.

## CMOS SWITCH

A CMOS switch is formed by connecting a PMOS and an NMOS switch in parallel – the input leads are connected together on the one side and the output leads are connected together on the other side. Figure 10.15 shows the switch so formed. It has two control inputs:

- N\_control turns ON the NMOS transistor and keeps it ON when it is in the 1 state.
- P\_control turns ON the PMOS transistor and keeps it ON when it is in the 0 state.



**Figure 10.15** A CMOS switch formed by connecting a PMOS transistor and an NMOS transistor in parallel.

The CMOS switch is instantiated as shown below.

```
cmos csw (out, in, N_control, P_control );
```

Significance of the different terms is as follows:

cmos: The keyword for the switch instantiation

csw: Name assigned to the switch in the instantiation

out: Name assigned to the output variable in the instantiation

in: Name assigned to the input variable in the instantiation

N\_control: Name assigned to the control variable of the NMOS transistor in the instantiation

P\_control: Name assigned to the control variable of the PMOS transistor in the instantiation

## BI-DIRECTIONAL GATES

Verilog has a set of primitives for bi-directional switches as well. They connect the nets on either side when ON and isolate them when OFF. The signal flow can be in either direction. None of the continuous-type assignments at higher levels dealt with so far has a functionality equivalent to the bi-directional gates. There are six types of bi-directional gates.

- tran                    rtran
- tranif1                rtanif1
- tranif0                rtranif0

### tran and rtran

The tran gate is a bi-directional gate of two ports. When instantiated, it connects the two ports directly. Thus the instantiation

```
tran (s1, s2);
```

connects the signal lines s1 and s2. Either line can be input, inout or output. rtran is the resistive counterpart of tran.

### tranif1 and rtanif1

tranif1 is a bi-directional switch turned ON/OFF through a control line. It is in the ON-state when the control signal is at 1 (high) state. When the control line is at state 0 (low), the switch is in the OFF state. A typical instantiation has the form

```
tranif1 (s1, s2, c );
```

Here c is the control line. If c=1, s1 and s2 are connected and signal transmission can be in either direction.

rtranif1 is the resistive counterpart of tranif1. It is instantiated in an identical manner.

### tranif0 and rtranif0

tranif0 and rtranif0 are again bi-directional switches. The switch is OFF if the control line is in the 1 (high) state, and it is ON when the control line is in the 0 (low) state. A typical instantiation has the form

```
tranif0 (s1, s2, c);
```

With the above instantiation, if c = 0, s1 and s2 are connected and signal transmission can be in either direction. If c = 1, the switch is OFF and s1 and s2 are isolated from each other.

rtranif0 is the resistive counterpart of tranif0.

## INSTANTIATIONS WITH STRENGTHS AND DELAYS

Propagation delays can be specified for switch primitives on the same lines as was done with the gate primitives in Chapter 5. For example, an NMOS switch instantiated as

```
nmos g1 (out, in, ctrl );
```

has no delay associated with it. The instantiation

```
nmos (delay1) g2 (out, in, ctrl );
```

has delay1 as the delay for the output to rise, fall, and turn OFF. The instantiation

```
nmos (delay_r, delay_f) g3 (out, in, ctrl );
```

has delay\_r as the rise-time for the output. delay\_f is the fall-time for the output. The turn-off time is zero. The instantiation

```
nmos (delay_r, delay_f, delay_o) g4 (out, in, ctrl );
```

has delay\_r as the rise-time for the output. delay\_f is the fall-time for the output delay\_o is the time to turn OFF when the control signal ctrl goes from 0 to 1. Delays can be assigned to the other uni-directional gates (rcmos, pmos, rpmos, cmos, and rcmos) in a similar manner. Bi-directional switches do not delay transmission – their rise- and fall-times are zero. They can have only turn-on and turn-off delays associated with them. tran has no delay associated with it.

```
tranif1 (delay_r, delay_f) g5 (out, in, ctrl );
```

represents an instantiation of the controlled bi-directional switch. When control changes from 0 to 1, the switch turns on with a delay of delay\_r. When control changes from 1 to 0, the switch turns off with a delay of delay\_f.

```
transif1 (delay0) g2 (out, in, ctrl );
```

represents an instantiation with delay0 as the delay for the switch to turn on when control changes from 0 to 1, with the same delay for it to turn off when control changes from 1 to 0. When a delay value is not specified in an instantiation, the turn-on and turn-off are considered to be ideal that is, instantaneous. Delay values similar to the above illustrations can be associated with rtranif1, tranif0, and rtranif0 as well.



## INSTANTIATIONS WITH STRENGTHS AND DELAYS

In the most general form of instantiation, strength values and delay values can be combined. For example, the instantiation

```
nmos (strong1, strong0) (delay_r, delay_f, delay_o ) gg (s1, s2, ctrl) ;
```

means the following:

- It has strength strong0 when in the low state and strength strong1 when in the high state.
- When output changes state from low to high, it has a delay time of delay\_r.
- When the output changes state from high to low, it has a delay time of delay\_f.
- When output turns-off it has a turn-off delay time of delay\_o.

rnmos, pmos, and rpmos switches too can be instantiated in the general form in the same manner. The general instantiation for the bi-directional gates too can be done similarly.

## SYSTEM TASKS, FUNCTIONS, AND COMPILER DIRECTIVES

A number of facilities in Verilog relate to the management of simulation; starting and stopping of simulation, selectively monitoring the activities, testing the design for timing constraints, etc., are amongst them. Although a variety of such constructs is available in Verilog.

### PARAMETERS

Verilog defines parameter as a constant value that is declared within structure of module. The constant value signifies timing values, range of variables, wires e.t.c.

The parameter values can be specified and changed to suit the design environment or test environment. Such changes are effected and frozen at instantiation.

The assigned values cannot change during testing or synthesis.

Two types of parameters are of use in modules: specparam and defparam.

**Specparam** : Parameters related to timings, time delays, rise and fall times, etc., are technology-specific and used during simulation. Parameter values can be assigned or overridden with the keyword “specparam” preceding the assignments.

**Defparam**: Parameters related to design, bus width, and register size are of a different category. They are related to the size or dimension of a specific design; they are technology-independent. Assignment or overriding is with assignments following the keyword “defparam”.

### Timing-Related Parameters

The constructs associated with parameters are discussed here through specific design or test modules.

Example: Module of a half-adder with delays assigned to the output transitions; a test bench is also included in the figure.

```
module ha_1(s,ca,a,b);  
  input a,b; output s,ca;  
  xor #(1,2) (s,a,b);  
  and #(3,4) (ca,a,b);  
endmodule
```

```
//test-bench  
module tstha;  
  reg a,b; wire s,ca;  
  ha_1 hh(s,ca,a,b);  
  initial  
  begin
```

```

a=0;b=0;
end
always
begin
#5 a=1;b=0;
#5 a=0;b=1;
#5 a=1;b=1;
#5 a=0;b=0;
end
initial $monitor($time , " a = %b , b = %b ,out carry = %b , outsum = %b ",a,b,ca,s);
initial #30 $stop;
endmodule

```

### Parameter Declarations and Assignments

Declaration of parameters in a design as well as assignments to them can be effected using the keyword "Parameter." A declaration has the form

```
parameter alpha = a, beta = b;
```

Where

- parameter is the keyword,
- alpha and beta are the names assigned to two parameters and
- a, b are values assigned to alpha and beta, respectively.

In general a and b can be constant expressions. The parameter values can be overridden during instantiation but cannot be changed during the run-time. If a parameter assignment is made through the keyword "localparam," its value cannot be overridden.

## PATH DELAYS

The delay between source pin (input or inout) and destination pin (ouput or inout) of module is called module path delay.

Verilog has the provision to specify and check delays associated with total paths – from any input to any output of a module. Such paths and delays are at the chip or system level. They are referred to as “module path delays”.

Constructs available make room for specifying their paths and assigning delay values to them – separately or together.

## Specify Blocks

Module paths are specified and values assigned to their delays through specify blocks. They are used to specify rise time, fall time, path delays pulse widths, and the like. A “specify” block can have the form shown in Figure

### **specify**

```
specparam rise_time = 5, fall_time = 6;
```

```
(a =>b) = (rise_time, fall_time);
```

```
(c => d) = (6, 7);
```

### **endspecify**

The block starts with the keyword “specify” and ends with the keyword “endspecify”. Specify blocks can appear anywhere within a module.

## Module Paths

Module Path delays are assigned in Verilog within the keywords specify and endspecify. The statements within these keywords constitute a specify block.

Module paths can be specified in different ways inside a specify block.

### Parallel connection

Every path delay statement has a source field and a destination field.

A parallel connection is specified by the symbol => and is used as shown below.

Usage: ( <source\_field> => <destination\_field>) = <delay\_value>;

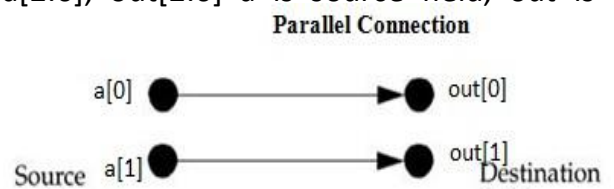
In a parallel connection, each bit in source field connects to its corresponding bit in the destination field.

If the source and the destination fields are vectors, they must have the same number of bits; otherwise, there is a mismatch. Thus, a parallel connection specifies delays from each bit in source to each bit in destination.

### Example: Parallel Connection

(a => out) = 9; //bit-to-bit connection. Both a and out are single-bit

// vector connection. Both a and out are 4-bit vectors a[2:0], out[2:0] a is source field, out is destination field.



```
// for three bit-to-bit connection statements.
```

```
(a[0] => out[0]) = 9;
```

```
(a[1] => out[1]) = 9;
```

```
(a[2] => out[2]) = 9;
```

```
//illegal connection. a[4:0] is a 5-bit vector, out[3:0] is 4-bit.
```

```
//Mismatch between bit width of source and destination fields
```

```
(a => out) = 9; //bit width does not match.
```

### Full connection

A full connection is specified by the symbol `*>` and is used as shown below.

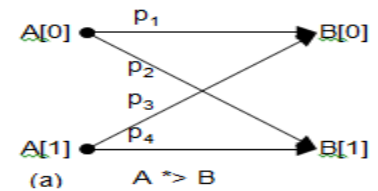
Usage: ( `<source_field> *> <destination_field>` ) = `<delay_value>`;

In a full connection, each bit in the source field connects to every bit in the destination field. If the source and the destination are vectors, then they need not have the same number of bits. A full connection describes the delay between each bit of the source and every bit in the destination.

Example:

Figure below illustrates a case of all possible paths from

a 2-bit vector A to another 2-bit vector B; the specification implies 4 pa



We can write the module M with pin-to-pin delays, using specify blocks as follows:

```
// Parallel connection
```

```
module M (out, a, b, c, d);
```

```
output out;
```

```
input a, b, c, d;
```

```
wire e, f;
```

```
//Specify block with path delay statements
```

```
specify
```

```
(a => out) = 9;
```

```
(b => out) = 9;
```

```
(c => out) = 11;
```

```
(d => out) = 11;
```

```
endspecify
```

```
//gate instantiations
```

```
and a1(e, a, b);
```

```
and a2(f, c, d);
```

```
and a3(out, e, f);
```

```
endmodule
```

```
//Full Connection
```

```
module M (out, a, b, c, d);
```

```
output out;
```

```
input a, b, c, d;
```

```
wire e, f;
```

```
specify
```

```
(a,b *> out) = 9;
```

```
(c,d *> out) = 11;
```

```
endspecify
```

```
and a1(e, a, b);
```

```
and a2(f, c, d);
```

```
and a3(out, e, f);
```

```
endmodule
```

## MODULE PARAMETERS

Module parameters are associated with size of bus, register, memory, ALU, and so on. They can be specified within the concerned module but their value can be altered during instantiation. The alterations can be brought about through assignments made with defparam. Such defparam assignments can appear anywhere in a module.

### Example

The parameter msb specifies the ALU size — consistently in the input and the output vectors of the ALU. The size assignment has been made separately through the assignment statement

```
parameter msb = 3;
```

The ALU module with its size declared as a parameter.

```
module alu (d, co, a, b, f, cci);
  parameter msb=3;
  output [msb:0] d; output co;
  wire[msb:0]d;
  input cci;
  input [msb : 0 ] a, b;
  input [1 : 0] f;
  specify
  (a,b=>d)=(1,2);
  (a,b,cci*>co)=1;
  endspecify
  assign {co,d}= (f==2'b00)?(a+b+cci):((f==2'b01)?(a-b):((f==2'b10)?{1'bz,a^b}:{1'bz,~a}));
endmodule
```

## SYSTEM TASKS AND FUNCTIONS

Verilog has a number of System Tasks and Functions defined in the LRM (language reference manual).

They are for taking output from simulation, control simulation, debugging design modules, testing modules for specifications, etc.

A “\$” sign preceding a word or a word group signifies a system task or a system function.

### Output Tasks

A number of system tasks are available to output values of variables and selected messages, etc., on the monitor. Out of these \$monitor and \$display tasks have been extensively used.

### Display Tasks

The \$display task, whenever encountered, displays the arguments in the desired format; and the display advances to a new line.

### \$strobe Task:

When a variable or a set of variables is sampled and its value displayed, the \$strobe task can be used; it senses the value of the specified variables and displays them.

The \$strobe task is executed as the last activity in the concerned time step. It is useful to check for specific activities and debug modules.

### Example:

```
initial #9 $strobe ("at time %t, di=%b, do=%b", $time, di, do);
```

### \$monitor Task:

\$monitor task is activated and displays the arguments specified whenever any of the arguments changes.

### \$stop and \$finish Tasks:

The \$stop task suspends simulation. The compiled design remains active; simulation can be resumed through commands available in the simulator.

In contrast \$finish stops simulation, closes the simulation environment, and reverts to the operating system.

### \$random Function:

A set of random number generator functions are available as system functions.

One can start with a seed number (optional) and generate a random number repeatedly. Such random number sequences can be fruitfully used for testing.

## Compiler directives

Compiler directives are special commands, beginning with ```, that affect the operation of the Verilog simulator.

### Time Scale

``timescale` specifies the time unit and time precision. A time unit of 10 ns means a time expressed as say #2.3 will have a delay of 23.0 ns. Time precision specifies how delay values are to be rounded off during simulation. Valid time units include s, ms, us ( $\mu$ s), ns, ps, fs.

Only 1, 10 or 100 are valid integers for specifying time units or precision. It also determines the displayed time units in display commands like `$display`.

#### Syntax

```
`timescale time_unit / time_precision;
```

#### Examples

```
`timescale 1 ns/1 ps // unit =1ns, precision=1/1000ns
```

```
`timescale 1 ns /100 ps // time unit = 1ns; precision = 1/10ns;
```

### ``define`

A macro is an identifier that represents a string of text. Macros are defined with the directive ``define`, and are invoked with the quoted macro name as shown in the example. Verilog compilers will substitute the string for the macro name before starting compilation. Many people prefer to use macros instead of parameters.

The define directive in Verilog is similar to `#define` in c-language.

#### Syntax

```
`define macro_name text_string;
```

```
... `macro_name ...
```

#### Example

```
`define add_lsb a[7:0] + b[7:0]
```

```
`define N 8 // Word length
```

```
wire [`N -1:0] S;
```

```
assign S = 'add_lsb; // assign S = a[7:0] + b[7:0];
```

### Include Directive

Include is used to include the contents of a text file at the point in the current file where the include directive is. The include directive is similar to the C/C++ include directive.

#### Syntax

```
`include file_name;
```

#### Example

```
module x;
```

```
`include "dclr.v"; // contents of file "dclr.v" are put here
```



### **USER-DEFINED PRIMITIVES (UDP):**

The primitives available in Verilog are all of the gate or switch types. Verilog has the provision for the user to define primitives –called “user defined primitive (UDP)” and use them.

The designers occasionally like to use their own custom-built primitives when developing a design. Verilog provides the ability to define User- Defined Primitives (UDP). These primitives are self-contained and do not instantiate other modules or primitives. UDPs are instantiated exactly like gate-level primitives.

UDPs are basically of two types –combinational and sequential. A combinational UDP is used to define a combinational scalar function and a sequential UDP for a sequential function.

### **Combinational UDPs:**

A combinational UDP accepts a set of scalar inputs and gives a scalar output. An inout declaration is not supported by a UDP. The UDP definition is on par with that of a module; that is, it is defined independently like a module and can be used in any other module.

```
primitive udp_and(out, a, b);
output out;
input a, b;
table
    // a b: Out;
    0 0: 0;
    0 1: 0;
    1 0: 0;
    1 1: 1;
endtable
endprimitive
```

### **Sequential UDPs:**

Any sequential circuit has a set of possible states. When it is in one of the specified states, the next state to be taken is described as a function of the input logic variables and the present state. A sequential UDP can accommodate all these.

```
primitive latch(q, d, clock, clear); // d-latch
output q;
reg q; //q declared as reg to create internal storage input d, clock, clear;
initial q = 0; //initialize output to value 0
table //state table
//d clock clear: q : q+ ;
?? 1 : ? : 0 ; //clear condition;
1 1 0 : ? : 1; //latchq =data=1
0 1 0 : ? : 0; //latchq =data=0
? 0 0 : ? : - ; //retain original state if clock = 0
endtable
endprimitive
```

## **UNIT-II**

## Fundamentals of MOSFETs

In 1958, Jack Kilby built the first integrated circuit flip-flop with two transistors at Texas Instruments. In 2008, Intel's Itanium microprocessor contained more than 2 billion transistors and a 16 Gb Flash memory contained more than 4 billion transistors. This corresponds to a compound annual growth rate of 53% over 50 years. No other technology in history has sustained such a high growth rate lasting for so long.

This incredible growth has come from steady miniaturization of transistors and improvements in manufacturing processes. Most other fields of engineering involve tradeoffs between performance, power, and price. However, as transistors become smaller, they also become faster, dissipate less power, and are cheaper to manufacture. This synergy has not only revolutionized electronics, but also society at large. The processing performance once dedicated to secret government supercomputers is now available in disposable cellular telephones. The memory once needed for an entire company's accounting system is now carried by a teenager in her iPod. Improvements in integrated circuits have enabled space exploration, made automobiles safer and more fuel efficient, revolutionized the nature of warfare, brought much of mankind's knowledge to our Web browsers, and made the world a flatter place. Fig. 1.1 shows annual sales in the worldwide semiconductor market. Integrated circuits became a \$100 billion/year business in 1994. In 2007, the industry manufactured approximately 6 quintillion ( $6 \times 10^{18}$ ) transistors, or nearly a billion for every human being on the planet. Thousands of engineers have made their fortunes in the field. New fortunes lie ahead for those with innovative ideas and the talent to bring those ideas to reality. During the first half of the twentieth century, electronic circuits used large, expensive, power-hungry, and unreliable vacuum tubes. In 1947, John Bardeen and Walter Brattain built the first functioning point contact transistor at Bell Laboratories. It was nearly classified as a military secret, but Bell Labs publicly introduced the device the following year.

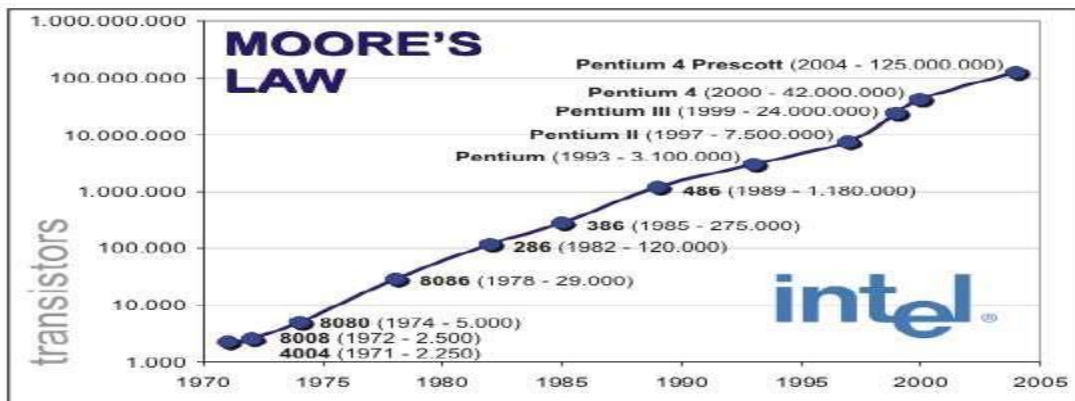
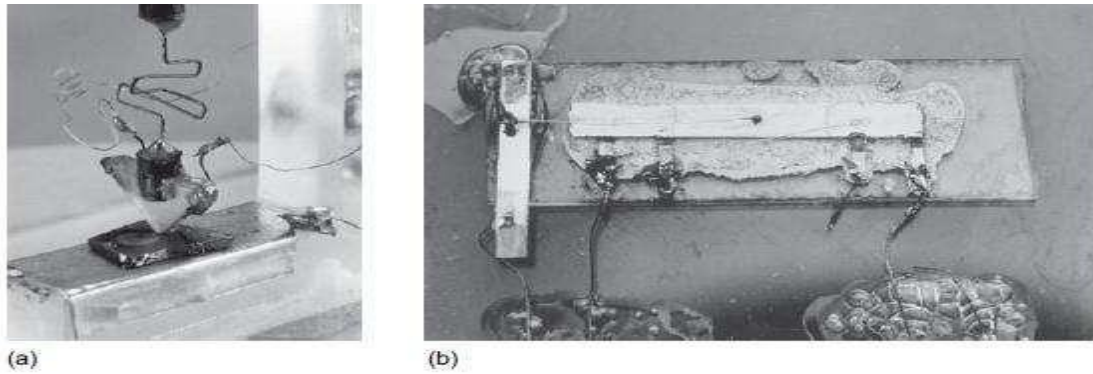


Fig 1.1

Ten years later, Jack Kilby at Texas Instruments realized the potential for miniaturization if multiple transistors could be built on one piece of silicon. Figure 1.2(b) shows his first prototype of an integrated circuit, constructed from a germanium slice and gold wires. The invention of the transistor earned the Nobel Prize in Physics in 1956 for Bardeen, Brattain, and their supervisor William Shockley. Kilby received the Nobel Prize in Physics in 2000 for the invention of the integrated circuit. Transistors can be viewed as electrically controlled switches with a control terminal and two other terminals that are connected or disconnected depending on the voltage or current applied to the control. Soon after inventing the point contact transistor, Bell Labs developed the bipolar junction transistor. Bipolar transistors were more reliable, less noisy, and more power-efficient.



**FIGURE 1.2** (a) First transistor (Property of AT&T Archives. Reprinted with permission of AT&T.) and (b) first integrated circuit (Courtesy of Texas Instruments.)

Early integrated circuits primarily used bipolar transistors. Bipolar transistors require a small current into the control (base) terminal to switch much larger currents between the other two (emitter and collector) terminals. The quiescent power dissipated by these base currents, drawn even when the circuit is not switching, limits the maximum number of transistors that can be integrated onto a single die. By the 1960s, Metal Oxide Semiconductor Field Effect Transistors (MOSFETs) began to enter production. MOSFETs offer the compelling advantage that they draw almost zero control current while idle. They come in two flavors: nMOS and pMOS, using n-type and p-type silicon, respectively. The original idea of field effect transistors dated back to the German scientist Julius Lilienfeld in 1925 and a structure closely resembling the MOSFET was proposed in 1935 by Oskar Heil . but materials problems foiled early attempts to make functioning devices. In 1963, Frank Wanlass at Fairchild described the first logic gates using MOSFETs . Fairchild's gates used both nMOS and pMOS transistors, earning the name Complementary Metal Oxide Semiconductor, and CMOS. The circuits used discrete transistors but consumed only nanowatts of power, six orders of magnitude less than their bipolar counterparts. With the development of the silicon planar process, MOS integrated circuits became attractive for their low cost because each transistor occupied less area and the fabrication process was simpler . Early commercial processes used only pMOS transistors and suffered from poor performance, yield, and reliability. Processes using nMOS transistors became common in the 1970s . Intel pioneered nMOS technology with its 1101 256-bit static random access memory and 4004 4-bit microprocessor, as shown in Figure 1.1. While the nMOS process was less expensive than CMOS, nMOS logic gates still consumed power while idle. Power consumption became a major issue in the 1980s as hundreds of thousands of transistors were integrated onto a single die. CMOS processes were widely adopted and have essentially replaced nMOS and bipolar processes for nearly all digital logic applications. In 1965, Gordon Moore observed that plotting the number of transistors that can be most economically manufactured on a chip gives a straight line on a semilogarithmic scale . At the time, he found transistor count doubling every 18 months. This observation has been called *Moore's Law* and has become a self-fulfilling prophecy. Figure 1.1 shows that the number of transistors in Intel microprocessors has doubled every 26 months since the invention of the 4004. Moore's Law is driven primarily by *scaling* down the size of transistors and, to a minor extent, by building larger chips. The level of integration of chips has been classified as small-scale, medium-scale, large-scale, and very large scale. *Small-scale integration* (SSI) circuits, such as the 7404 inverter, have fewer than 10 gates, with roughly half a dozen transistors per gate. *Medium-scale integration* (MSI) circuits, such as the 74161 counter, have up to 1000 gates. *Large-scale integration*

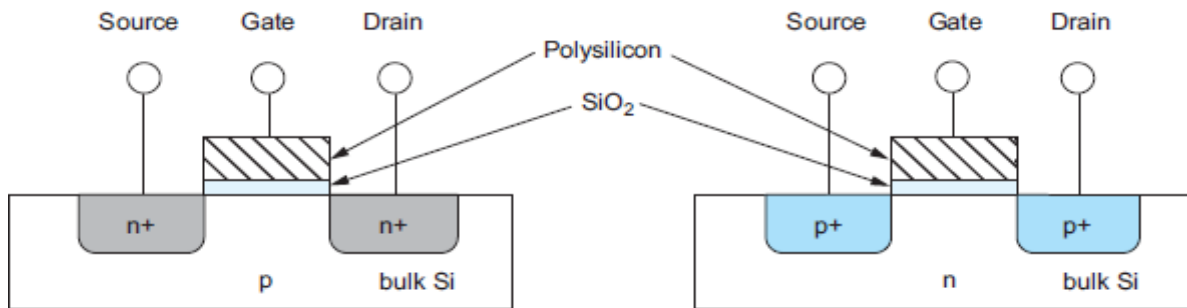
(LSI) circuits, such as simple 8-bit microprocessors, have up to 10,000 gates. It soon became apparent that new names would have to be created every five years if this naming trend continued and thus the term *very large-scale integration* (VLSI) is used to describe most integrated circuits from the 1980s onward. A corollary of Moore's law is *Dennard's Scaling Law* [Dennard74]: as transistors shrink, they become faster, consume less power, and are cheaper to manufacture. Intel microprocessor clock frequencies have doubled roughly every 34 months. This frequency scaling hit the power wall around 2004, and clock frequencies have leveled off around 3 GHz. Computer performance, measured in time to run an application, has advanced even more than raw clock speed. Presently, the performance is driven by the number of cores on a chip rather than by the clock. Even though an individual CMOS transistor uses very little energy each time it switches, the enormous number of transistors switching at very high rates of speed has made power consumption a major design consideration again. Moreover, as transistors have become so small, they cease to turn completely OFF. Small amounts of current leaking through each transistor now lead to significant power consumption when multiplied by millions or billions of transistors on a chip. The feature size of a CMOS manufacturing process refers to the minimum dimension of a transistor that can be reliably built. The 4004 had a feature size of 10  $\mu\text{m}$  in 1971. The Core 2 Duo had a feature size of 45 nm in 2008. Manufacturers introduce a new process generation (also called a technology node) every 2–3 years with a 30% smaller feature size to pack twice as many transistors in the same area. Feature sizes down to 0.25  $\mu\text{m}$  are generally specified in microns ( $10^{-6}$  m), while smaller feature sizes are expressed in nanometers ( $10^{-9}$  m). Effects that were relatively minor in micron processes, such as transistor leakage, variations in characteristics of adjacent transistors, and wire resistance, are of great significance in nanometer processes. Moore's Law has become a self-fulfilling prophecy because each company must keep up with its competitors. Obviously, this scaling cannot go on forever because transistors cannot be smaller than atoms. Dennard scaling has already begun to slow. By the 45 nm generation, designers are having to make trade-offs between improving power and improving delay. Although the cost of printing each transistor goes down, the one-time design costs are increasing exponentially, relegating state-of-the-art processes to chips that will sell in huge quantities or that have cutting-edge performance requirements. However, many predictions of fundamental limits to scaling have already proven wrong. Creative engineers and material scientists have billions of dollars to gain by getting ahead of their competitors. In the early 1990s, experts agreed that scaling would continue for at least a decade but that beyond that point the future was murky. In 2009, we still believe that Moore's Law will continue for at least another decade. The future is yours to invent.

### **Basics of MOSFET:**

A Metal-Oxide-Semiconductor (*MOS*) structure is created by superimposing several layers of conducting and insulating materials to form a sandwich-like structure. These structures are manufactured using a series of chemical processing steps involving oxidation of the silicon, selective introduction of dopants, and deposition and etching of metal wires and contacts. Transistors are built on nearly flawless single crystals of silicon, which are available as thin flat circular wafers of 15–30 cm in diameter. CMOS technology provides two types of transistors (also called *devices*): an n-type transistor (*nMOS*) and a p-type transistor (*pMOS*). Transistor operation is controlled by electric fields so the devices are also called Metal Oxide Semiconductor Field Effect Transistors (*MOSFETs*) or simply *FETs*. Cross-sections and symbols of these transistors are shown in Figure 1.3. The n+ and p+ regions indicate heavily doped n- or p-type silicon.

Each transistor consists of a stack of the conducting *gate*, an insulating layer of silicon dioxide ( $\text{SiO}_2$ , better

known as glass), and the silicon wafer, also called the *substrate*, *body*, or *bulk*. Gates of early transistors were built from metal, so the stack was called metaloxide- semiconductor, or MOS.

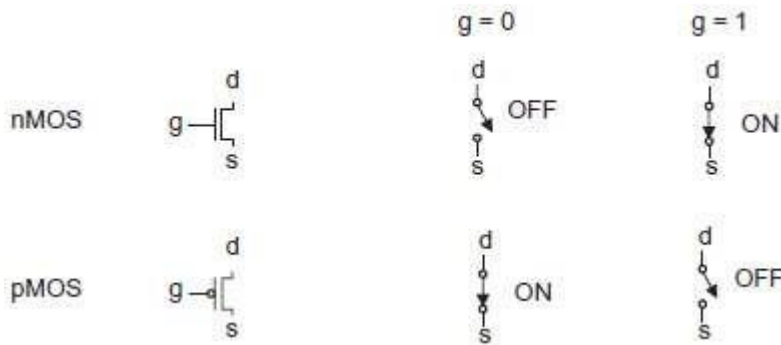


**FIGURE 1.3** nMOS transistor and pMOS transistor

Since the 1970s, the gate has been formed from polycrystalline silicon (*polysilicon*), but the name stuck. (Interestingly, metal gates reemerged in 2007 to solve materials problems in advanced manufacturing processes.) An nMOS transistor is built with a p-type body and has regions of n-type semiconductor adjacent to the gate called the *source* and *drain*. They are physically equivalent and for now we will regard them as interchangeable. The body is typically grounded. A pMOS transistor is just the opposite, consisting of p-type source and drain regions with an n-type body. In a CMOS technology with both flavors of transistors, the substrate is either n-type or p-type. The other flavor of transistor must be built in a special *well* in which dopant atoms have been added to form the body of the opposite type. The gate is a control input: It affects the flow of electrical current between the source and drain. Consider an nMOS transistor. The body is generally grounded so the p–n junctions of the source and drain to body are reverse-biased. If the gate is also grounded, no current flows through the reverse-biased junctions. Hence, we say the transistor is OFF. If the gate voltage is raised, it creates an electric field that starts to attract free electrons to the underside of the Si–SiO<sub>2</sub> interface. If the voltage is raised enough, the electrons outnumber the holes and a thin region under the gate called the *channel* is inverted to act as an n-type semiconductor. Hence, a conducting path of electron carriers is formed from source to drain and current can flow. We say the transistor is ON. For a pMOS transistor, the situation is again reversed. The body is held at a positive voltage. When the gate is also at a positive voltage, the source and drain junctions are reverse-biased and no current flows, so the transistor is OFF. When the gate voltage is lowered, positive charges are attracted to the underside of the Si–SiO<sub>2</sub> interface. A sufficiently low gate voltage inverts the channel and a conducting path of positive carriers is formed from source to drain, so the transistor is ON. Notice that the symbol for the pMOS transistor has a bubble on the gate, indicating that the transistor behavior is the opposite of the nMOS. The positive voltage is usually called *VDD* or *POWER* and represents a logic 1 value in digital circuits. In popular logic families of the 1970s and 1980s, *VDD* was set to 5 volts. Smaller, more recent transistors are unable to withstand such high voltages and have used supplies of 3.3 V, 2.5 V, 1.8 V, 1.5 V, 1.2 V, 1.0 V, and so forth. The low voltage is called *GROUND* (*GND*) or *VSS* and represents a logic 0. It is normally 0 volts. In summary, the gate of an MOS transistor controls the flow of current between the source and drain. Simplifying this to the extreme allows the MOS transistors to be viewed as simple ON/OFF switches. When the gate of an nMOS transistor is 1, the transistor is ON and there is a conducting path from source to drain. When the gate is low, the nMOS transistors are OFF and almost zero current flows from source to drain. A pMOS transistor is just the opposite, being ON when the gate is low and OFF when the gate is high. This



switch model is illustrated in Figure 1.4, where  $g$ ,  $s$ , and  $d$  indicate gate, source, and drain. This model will be our most common one when discussing circuit behavior.



**FIGURE 1.4** Transistor symbols and switch-level models

Even when transistors are nominally OFF, they leak small amounts of current. Leakage mechanisms include subthreshold conduction between source and drain, gate leakage from the gate to body, and junction leakage from source to body and drain to body, as illustrated in Figure 2.19 [Roy03, Narendra06]. Subthreshold conduction is caused by thermal emission of carriers over the potential barrier set by the threshold. Gate leakage is a quantum-mechanical effect caused by tunneling through the extremely thin gate dielectric. Junction leakage is caused by current through the p-n junction between the source/drain diffusions and the body.

In processes with feature sizes above 180 nm, leakage was typically insignificant except in very low power applications. In 90 and 65 nm processes, threshold voltage has reduced to the point that subthreshold leakage reaches levels of 1s to 10s of nA per transistor, which is significant when multiplied by millions or billions of transistors on a chip. In 45 nm processes, oxide thickness reduces to the point that gate leakage becomes comparable to subthreshold leakage unless high-k gate dielectrics are employed. Overall, leakage has become an important design consideration in nanometer processes.

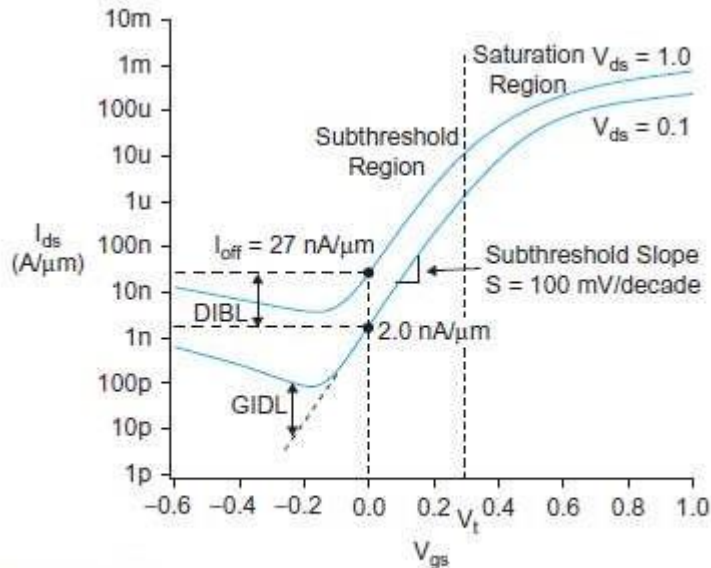
**Subthreshold Leakage** The long-channel transistor I-V model assumes current only flows from source to drain when  $V_{gs} > V_t$ . In real transistors, current does not abruptly cut off below threshold, but rather drops off exponentially, as seen in Figure 2.20. When the gate voltage is high, the transistor is strongly ON. When the gate falls below  $V_t$ , the exponential decline in current appears as a straight line on the logarithmic scale. This regime of  $V_{gs} < V_t$  is called *weak inversion*. The *subthreshold leakage current* increases significantly with  $V_{ds}$  because of drain-induced barrier lowering. There is a lower limit on  $I_{ds}$  set by drain junction leakage that is exacerbated by the negative gate voltage.

### **MOSFETs in the Sub-threshold Region (i.e. a bit below $V_T$ )**

In the depletion approximation for n-channel MOS structures we have neglected the electrons beneath the gate electrode when the gate voltage is less than the threshold voltage,  $V_T$ . We said that it is only when the gate voltage is above threshold that they are significant, and that they are then the dominant negative charge under the gate. Furthermore, we say that above threshold all of the gate voltage in excess of  $V_T$  induces electrons in the channel.

As MOS integrated circuit technology has evolved to exploit smaller and smaller device structures, it has become increasingly important in recent years to look more closely at the minority carriers present under the gate when the gate voltage is less than threshold, i.e. in what is called the “sub-threshold” region. These carriers cannot be totally neglected, and play an important role in device and circuit performance. At first they were viewed primarily as a problem, causing undesirable “leakage” currents and limiting circuit performance. Now it

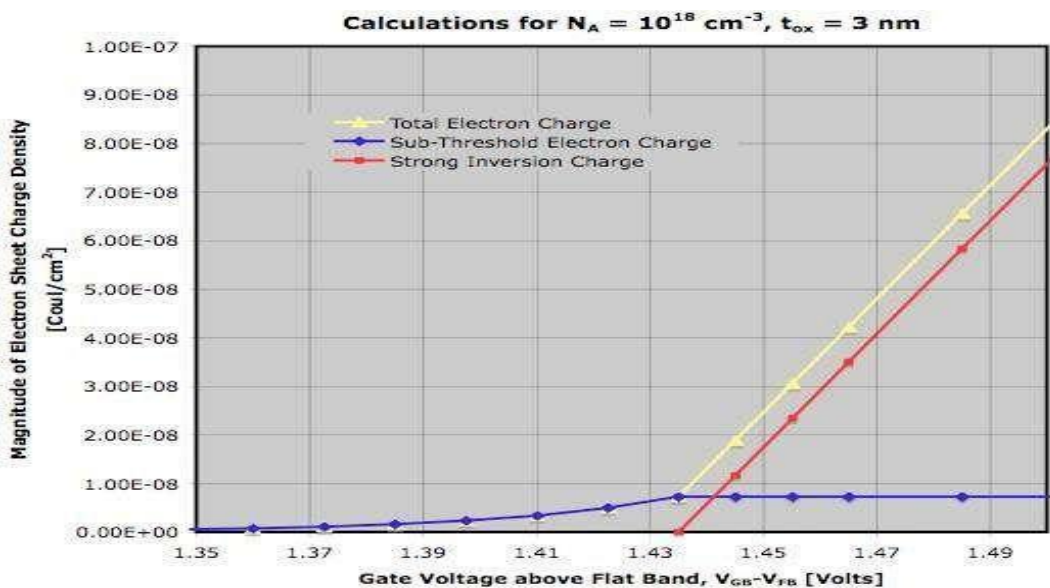
is recognized that they also enable a very useful mode of MOSFET operation, and that the sub-threshold region of operation is as important as the traditional cut-off, linear, and saturations regions of operation. To begin our study of the sub-threshold region, we will first quickly review the electrostatics of the MOS capacitor, and the electrostatic potential profile predicted by the depletion approximation model.



**FIGURE 2.20** I-V characteristics of a 65 nm nMOS transistor at 70 °C on a log scale

**Fig 1.5**

Then we will use this result to derive a more accurate expression than that in Equation 1 for  $qN^*$  below threshold, and use the resulting expression to, among other things, assess the assumption that the contribution of the mobile electrons underneath the gate to the net charge density in the depletion region is negligible compared to the contribution from the ionized acceptors. Finally we will look at the current-voltage characteristic of a MOSFET operating in the sub-threshold region, and merge it with our earlier model so that we then have a model in which the mobile electron charge is taken into account and the drain current is no longer identically zero when  $v_{GS}$  is less than  $V_T$ .

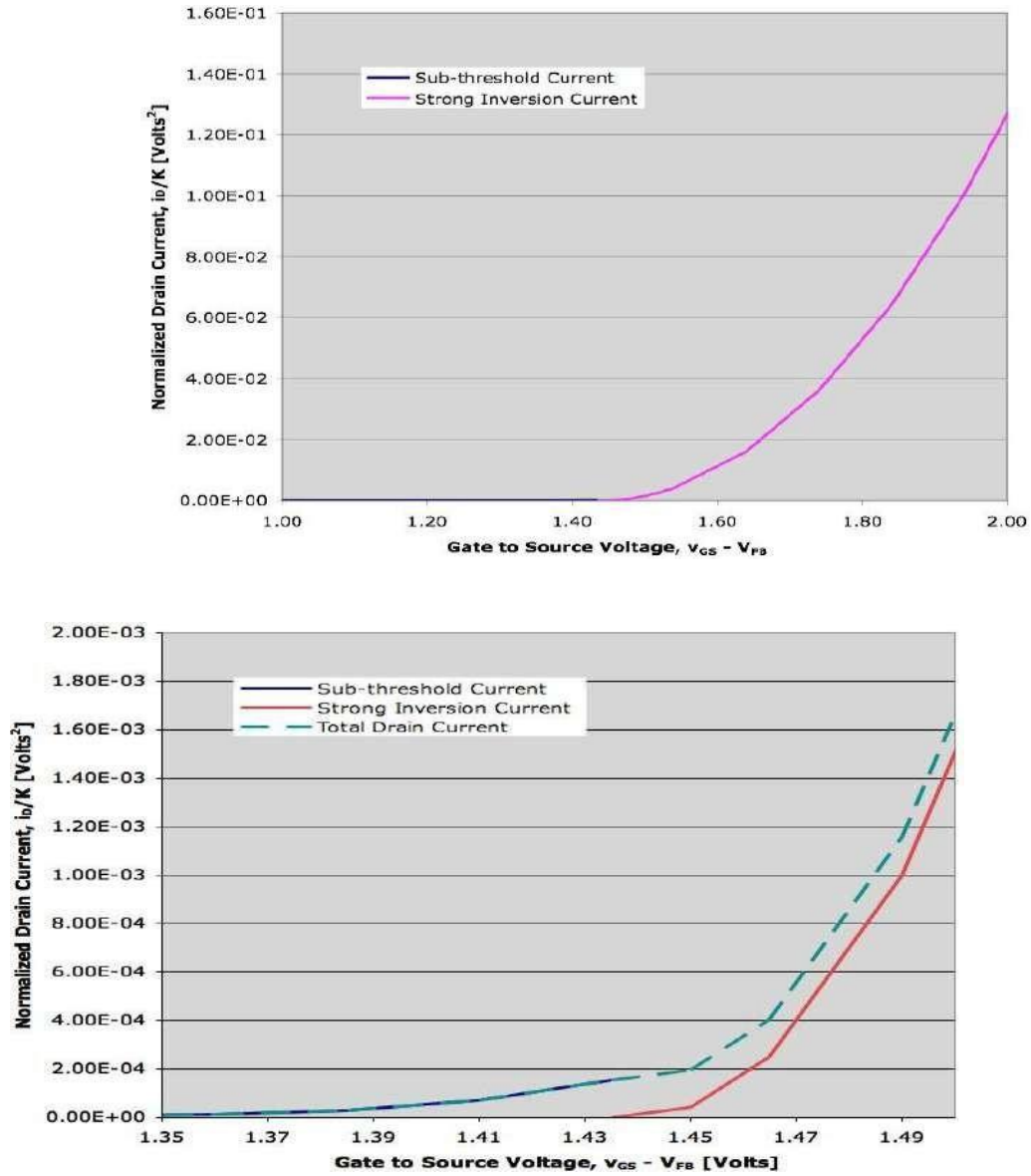




**FIGURE 1.6**

The electron sheet charge density under the gate with a gate voltage in the vicinity of threshold. The blue curve corresponds to the sub-threshold weak-inversion charge [Eq. 12], and the red curve is the strong inversion charge from traditional depletion approximation modeling. The sum is plotted in the yellow curve.

**The MOSFET Drain Current in the Sub-threshold Region:**



**FIGURE 1.7**

The drain current,  $i_D$ , for gate biases from just below to just above threshold illustrating the relative sizes of the diffusion (sub-threshold) and drift (strong inversion) components of  $i_D$ . Figure 1.7 a shows the situation with a course voltage scale, while Figure 1.7 b has smaller increments and thus focuses in more on the curves near  $V_T$ .

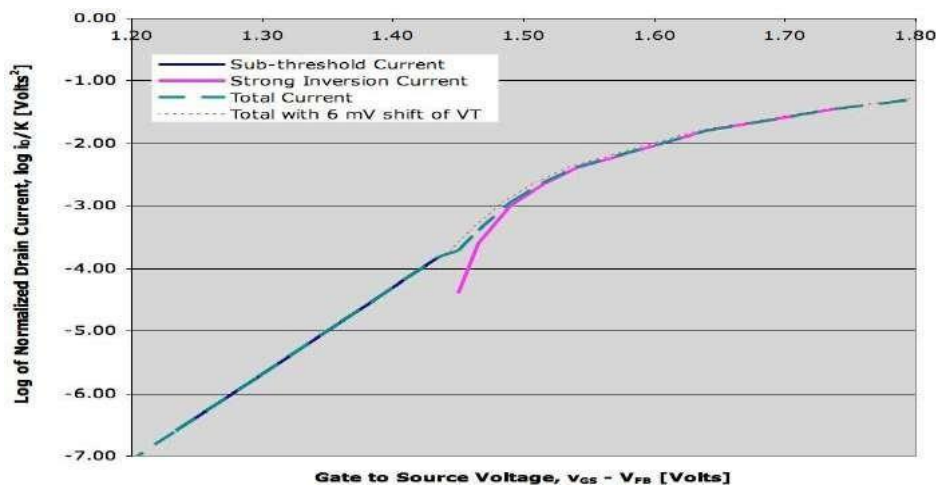


FIGURE 1.8

The drain current,  $i_D$ , for gate biases from just below to just above threshold as in Figure 1.7b, but now plotted on a log-linear graph so the nature of the small, sub-threshold current and its variation with input voltage,  $V_{GS}$ , is more clear. The simple depletion approximation model we are using, our neglect of drift currents below threshold, and our use of a saturated diffusion current above threshold, are all reasonable approximations, and are better the further we are above or below threshold. We should expect them to have limitations very near threshold, however, and in particular we should not demand too much from them as we pass from  $v_{GS} < V_T$  to  $v_{GS} > V_T$ . The curves look rather smooth in Figure 1.7 b, but we see a slight kink on the log scale of Figure 1.8. Interestingly, this kink can be smoothed nicely by using the adjusted threshold in the saturation current expression, as shown in Fig. 1.8, but is largely coincidence as the 6 mV shift came from looking at the total gate charge in the sub-threshold region, not the inversion layer in strong inversion. Just the same, our simple models accurately reflect the physics, and do an outstanding job no matter how one looks at it.

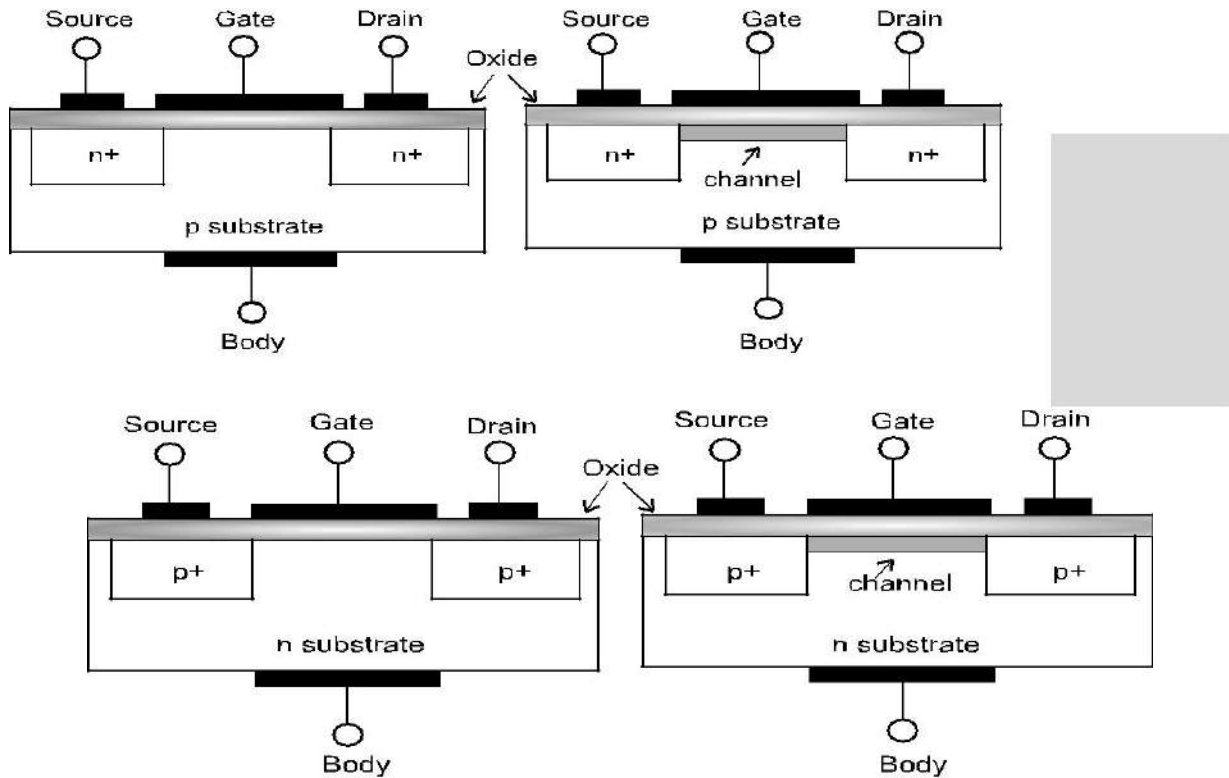
### ENHANCEMENT AND DEPLETION MODE MOS TRANSISTORS

MOS Transistors are built on a silicon substrate. Silicon which is a group IV material is the eighth most common element in the universe by mass, but very rarely occurs as the pure free element in nature. Pure silicon has no free carriers and conducts poorly. But adding dopants to silicon increases its conductivity. If a group V material i.e. an extra electron is added, it forms an n-type semiconductor. If a group III material i.e. missing electron pattern is formed (hole), the resulting semiconductor is called a p-type semiconductor.

A junction between p-type and n-type semiconductor forms a conduction path. Source and Drain of the Metal Oxide Semiconductor (MOS) Transistor is formed by the “doped” regions on the surface of chip. Oxide layer is formed by means of deposition of the silicon dioxide ( $\text{SiO}_2$ ) layer which forms as an insulator and is a very thin pattern.

Gate of the MOS transistor is the thin layer of “polysilicon (poly)”; used to apply electric field to the surface of silicon between Drain and Source, to form a “channel” of electrons or holes. Control by the Gate voltage is achieved by modulating the conductivity of the semiconductor region just below the gate. This region is known as the channel. The Metal–Oxide–Semiconductor Field Effect Transistor (MOSFET) is a transistor which is a voltage-controlled current device, in which current at two electrodes, drain and source is controlled by the action of an electric field at another electrode gate having in-between semiconductor and a very thin metal oxide layer. It is used for amplifying or switching electronic signals. The Enhancement and Depletion mode MOS transistors are further classified as N-type named NMOS (or N-channel MOS) and P-type named PMOS (or P-channel MOS) devices. Figure 1.9 shows the MOSFETs along with their enhancement and depletion

modes.



**Figure 1.9:** (c) Enhancement P-type MOSFET (d) Depletion P-type MOSFET

The depletion mode devices are doped so that a channel exists even with zero voltage from gate to source during manufacturing of the device. Hence the channel always appears in the device. To control the channel, a negative voltage is applied to the gate (for an N-channel device), depleting the channel, which reduces the current flow through the device. In essence, the depletion-mode device is equivalent to a closed (ON) switch, while the enhancement-mode device does not have the built in channel and is equivalent to an open (OFF) switch. Due to the difficulty of turning off the depletion mode devices, they are rarely used

**Working of Enhancement Mode Transistor** The enhancement mode devices do not have the in-built channel. By applying the required potentials, the channel can be formed. Also for the MOS devices, there is a threshold voltage ( $V_t$ ), below which not enough charges will be attracted for the channel to be formed. This threshold voltage for a MOS transistor is a function of doping levels and thickness of the oxide layer.

**Case 1:  $V_{gs} = 0V$  and  $V_{gs} < V_t$**  The device is non-conducting, when no gate voltage is applied ( $V_{gs} = 0V$ ) or ( $V_{gs} < V_t$ ) and also drain to source potential  $V_{ds} = 0$ . With an insufficient voltage on the gate to establish the channel region as N-type, there will be no conduction between the source and drain. Since there is no conducting channel, there is no current drawn, i.e.  $I_{ds} = 0$ , and the device is said to be in the **cut-off region**. This is shown in the Figure 1.7 (a).

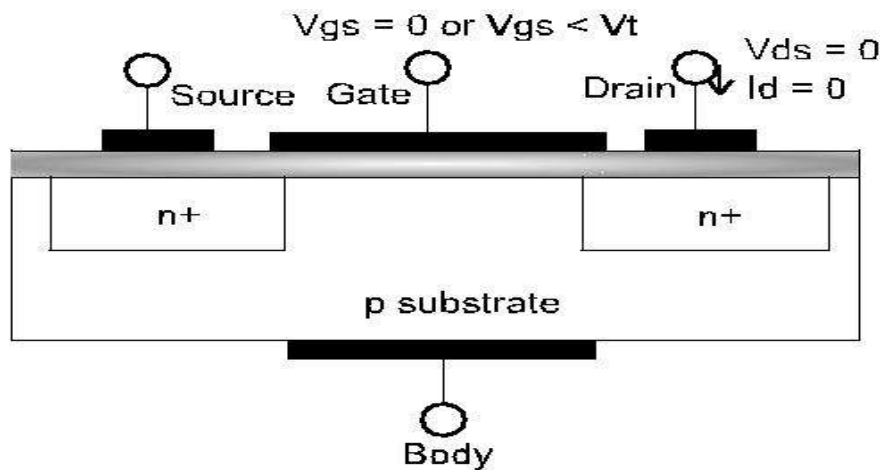
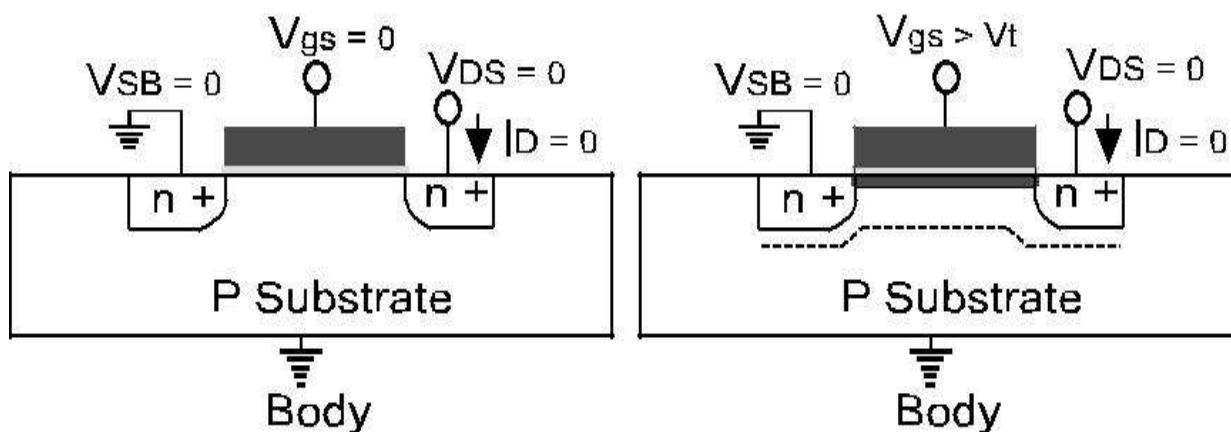


Figure 2.0 (a)

**Case 2:  $V_{gs} > V_t$**  When a minimum voltage greater than the threshold voltage  $V_t$  (i.e.  $V_{gs} > V_t$ ) is applied, a high concentration of negative charge carriers forms an inversion layer located by a thin layer next to the interface between the semiconductor and the oxide insulator. This forms a channel between the source and drain of the transistor. This is shown in the Figure 2.0 (b).



**Figure 2.0:** (b) Formation of a Channel

A positive  $V_{ds}$  reverse biases the drain substrate junction, hence the depletion region around the drain widens, and since the drain is adjacent to the gate edge, the depletion region widens in the channel. This is shown in Figure 2.0(c). This results in flow of electron from source to drain resulting in current  $I_{ds}$ . The device is said to operate in **linear region** during this phase. Further increase in  $V_{ds}$ , increases the reverse bias on the drain substrate junction in contact with the inversion layer which causes inversion layer density to decrease. This is shown in Figure 2.0 (d). The point at which the inversion layer density becomes very small (nearly zero) at the drain end is termed pinch-off. The value of  $V_{ds}$  at pinch-off is denoted as  $V_{ds,sat}$ . This is termed as **saturation region** for the MOS device. Diffusion current completes the path from source to drain in this case, causing the channel to exhibit a high resistance and behaves as a constant current source.

The MOSFET  $I_D$  versus  $V_{DS}$  characteristics ( $V$ - $I$  Characteristics) is shown in the Figure 2.0.

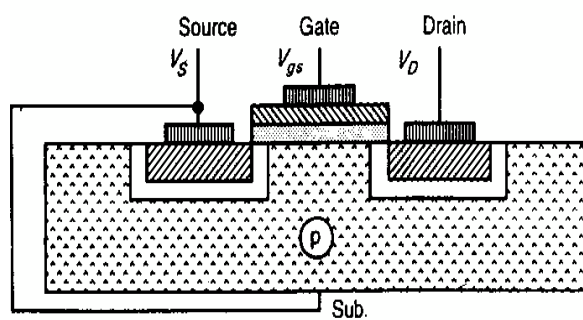
For  $V_{GS} < V_t$ ,  $I_D = 0$  and device is in cut-off region. As  $V_{DS}$  increases at a fixed  $V_{GS}$ ,  $I_D$  increases in the linear region due to the increased lateral field, but at a decreasing rate since the inversion layer density is decreasing. Once pinch-off is reached, further increase in  $V_{DS}$  results in increase in  $I_D$ ; due to the formation of the high field region which is very small. The device starts in linear region, and moves into saturation region at higher  $V_{DS}$ .

The MOSFET transistor has become one of the most important devices used in the design and construction of integrated circuits. Its thermal stability and other general characteristics make it extremely popular in computer circuit design. The basic principle of the MOSFET is that the source-to-drain current (SD current) is controlled by the gate voltage, or better, by the gate electric field. The electric field induces charge (field effect) in the semiconductor at the semiconductor-oxide interface. Thus the MOSFET is a voltage-controlled current source.

**Basic MOS transistors** with the doping concentration of transistor two types of MOS transistors are available as NMOS transistor and PMOS transistor. With their mode of operation further they are classified as depletion mode transistor and enhancement mode transistor.

**NMOS enhancement mode transistor**

nMOS devices are formed in a p-type substrate of moderate doping level. The source and drain regions are formed by diffusing n-type impurities through suitable masks into these areas. Thus source and drain are isolated from one another by two diodes and their connections are made by a deposited metal layer. The basic block diagrams of nMOS enhancement mode transistor is shown in figure.



(a) nMOS enhancement mode transistor

Fig 2.1

If the gate terminal is connected to a positive voltage (a minimum voltage level of **threshold voltage**) with respect to the source, then the electric field established between the gate and the substrate which gives a charge inversion region in the substrate under the gate insulation and a **conduction path** or **channel** is formed between source and drain, but no current flows between source and drain ( $V_{ds}=0$ ). When current flows in the channel by applying a voltage  $V_{ds}$  between source and drain there must be a voltage (IR) drop =  $V_{ds}$  along the channel.

This results that the voltage between gate and channel varying with distance along the channel with the voltage being a maximum of  $V_{gs}$  at the source end. The effective gate voltage is  $V_g = V_{gs} - V_t$ . To invert the channel at the drain end there will be voltage is available upto when  $V_{gs} - V_t > V_{ds}$ . For all voltages  $V_{ds} < V_{gs} - V_t$  the device is in the non-saturated region.

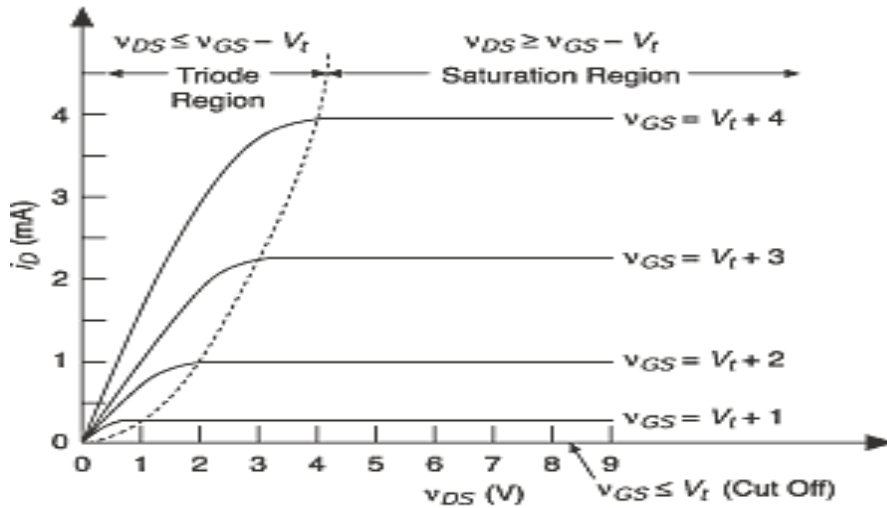


Fig 2.2

When  $V_{ds}$  is increased to a level greater than  $V_{gs} - V_t$ , if the voltage drop =  $V_{gs} - V_t$  takes place over less than the whole length of the channel near the drain, there is insufficient electric field available to give rise to an inversion layer to create the channel. Then the voltage is called '**pinch-off**' voltage. At this stage the diffusion current completes the path from source to drain and the channel exhibits a high resistance and behave as constant current source, This region is known as '**saturation**' region.

### nMOSdepletion mode transistor

The basic block diagram of nMOS depletion mode transistor is shown in figure. In depletion mode transistor the **channel** is established even the voltage  $V_{gs} = 0$  by implanting suitable impurities in the region between source and drain during manufacture and prior to depositing the insulation and the gate.

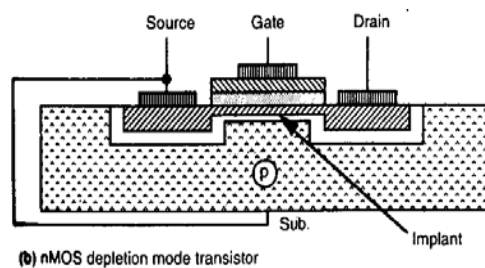


Fig 2.3

At this stage the source and drain are connected by a conducting channel, but the channel may now be closed by applying a suitable negative voltage to the gate. In both enhancement and depletion mode cases, variations of the gate voltage allow control of any current flow between source and drain.

### DRAIN-TO-SOURCE CURRENT $I_{ds}$ versus VOLTAGE $V_{ds}$ RELATIONSHIPS)

The whole concept of the MOS transistor evolves from the use of a voltage on the gate to induce a charge in the channel between source and drain, which may then be caused to move from source to drain under the influence of an electric field created by voltage  $V_{ds}$  applied between drain and source. Since the charge induced is dependent on the gate to source voltage  $V_{gs}$ , then  $I_{ds}$  is dependent on both  $V_{gs}$  and  $V_{ds}$ . Consider a structure, as in Figure 2.1, in which  $n^-$  electrons will flow source to drain:

$$I_{ds} = -I_{sd} = \frac{\text{Charge induced in channel } (Q_c)}{\text{Electron transit time } (\tau)} \quad (2.1)$$

First, transit time:

$$\tau_{sd} = \frac{\text{Length of channel } (L)}{\text{Velocity } (v)}$$

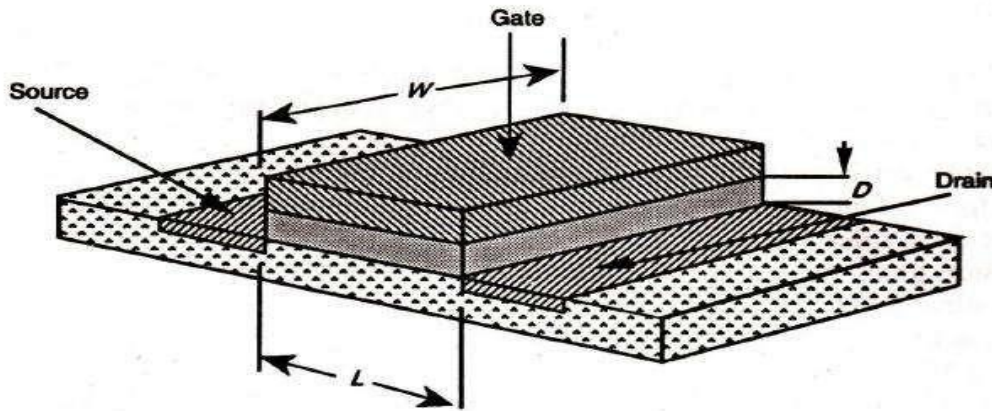


FIGURE 2.1 nMOS transistor structure.

but velocity

$$v = \mu E_{ds}$$

where

$\mu$  = electron or hole mobility (surface)

$E_{ds}$  = electric field (drain to source)

Now

$$E_{ds} = \frac{V_{ds}}{L}$$

so that

$$v = \frac{\mu V_{ds}}{L}$$

Thus

$$\tau_{sd} = \frac{L^2}{\mu V_{ds}} \quad (2.2)$$

Typical values of  $\mu$  at room temperature are:

$$\mu_n \doteq 650 \text{ cm}^2/\text{V sec (surface)}$$

$$\mu_p \doteq 240 \text{ cm}^2/\text{V sec (surface)}$$



### 2.1.1 The Non-saturated Region

Charge induced in channel due to gate voltage is due to the voltage difference between the gate and the channel,  $V_{gs}$  (assuming substrate connected to source). Now note that the voltage along the channel varies linearly with distance  $X$  from the source due to the  $IR$  drop in the channel (see Figure 1.5) and assuming that the device is not saturated then the average value is  $V_{ds}/2$ . Furthermore, the effective gate voltage  $V_g = V_{gs} - V_t$  where  $V_t$  is the threshold voltage needed to invert the charge under the gate and establish the channel.

Note that the charge/unit area =  $E_g \epsilon_{ins} \epsilon_0$ . Thus induced charge

$$Q_c = E_g \epsilon_{ins} \epsilon_0 WL$$

where

$E_g$  = average electric field gate to channel

$\epsilon_{ins}$  = relative permittivity of insulation between gate and channel

$\epsilon_0$  = permittivity of free space

(Note:  $\epsilon_0 = 8.85 \times 10^{-14} \text{F cm}^{-1}$ ;  $\epsilon_{ins} \doteq 4.0$  for silicon dioxide)

Now

$$E_g = \frac{\left( (V_{gs} - V_t) - \frac{V_{ds}}{2} \right)}{D}$$

where  $D$  = oxide thickness.

Thus

$$Q_c = \frac{WLE_{ins}\epsilon_0}{D} \left( (V_{gs} - V_t) - \frac{V_{ds}}{2} \right) \quad (2.3)$$

Now, combining equations (2.2) and (2.3) in equation (2.1), we have

$$I_{ds} = \frac{\epsilon_{ins}\epsilon_0\mu}{D} \frac{W}{L} \left( (V_{gs} - V_t) - \frac{V_{ds}}{2} \right) V_{ds}$$



or

$$I_{ds} = K \frac{W}{L} \left( (V_{gs} - V_t) V_{ds} - \frac{V_{ds}^2}{2} \right) \quad (2.4)$$

in the non-saturated or resistive region where  $V_{ds} < V_{gs} - V_t$  and

$$K = \frac{\epsilon_{ins} \epsilon_0 \mu}{D}$$

The factor  $W/L$  is, of course, contributed by the geometry and it is common practice to write

$$\beta = K \frac{W}{L}$$

so that

$$I_{ds} = \beta \left( (V_{gs} - V_t) V_{ds} - \frac{V_{ds}^2}{2} \right) \quad (2.4a)$$

which is an alternative form of equation (2.4).

Noting that gate/channel capacitance

$$C_g = \frac{\epsilon_{ins} \epsilon_0 W L}{D} \text{ (parallel plate)}$$

we also have

$$K = \frac{C_g \mu}{W L}$$

so that

$$I_{ds} = \frac{C_g \mu}{L^2} \left( (V_{gs} - V_t) V_{ds} - \frac{V_{ds}^2}{2} \right) \quad (2.4b)$$

which is a further alternative form of equation (2.4).

Sometimes it is convenient to use *gate capacitance per unit area*  $C_0$  (which is often denoted  $C_{ox}$ ) rather than  $C_g$  in this and other expressions. Noting that

$$C_g = C_0 W L$$

we may also write

$$I_{ds} = C_0 \mu \frac{W}{L} \left( (V_{gs} - V_t) V_{ds} - \frac{V_{ds}^2}{2} \right) \quad (2.4c)$$

### 2.1.2 The Saturated Region

*Saturation* begins when  $V_{ds} = V_{gs} - V_t$ , since at this point the  $IR$  drop in the channel equals the effective gate to channel voltage at the drain and we may assume that the current remains fairly constant as  $V_{ds}$  increases further. Thus

$$I_{ds} = K \frac{W}{L} \frac{(V_{gs} - V_t)^2}{2} \quad (2.5)$$

or, we may write

$$I_{ds} = \frac{\beta}{2} (V_{gs} - V_t)^2 \quad (2.5a)$$

or

$$I_{ds} = \frac{C_g \mu}{2 L^2} (V_{gs} - V_t)^2 \quad (2.5b)$$

We may also write

$$I_{ds} = C_0 \mu \frac{W}{2 L} (V_{gs} - V_t)^2 \quad (2.5c)$$

The expressions derived for  $I_{ds}$  hold for both enhancement and depletion mode devices, but it should be noted that the threshold voltage for the nMOS depletion mode device (denoted as  $V_{td}$ ) is *negative*.

Typical characteristics for nMOS transistors are given in Figure 2.2. pMOS transistor characteristics are similar, with suitable reversal of polarity.

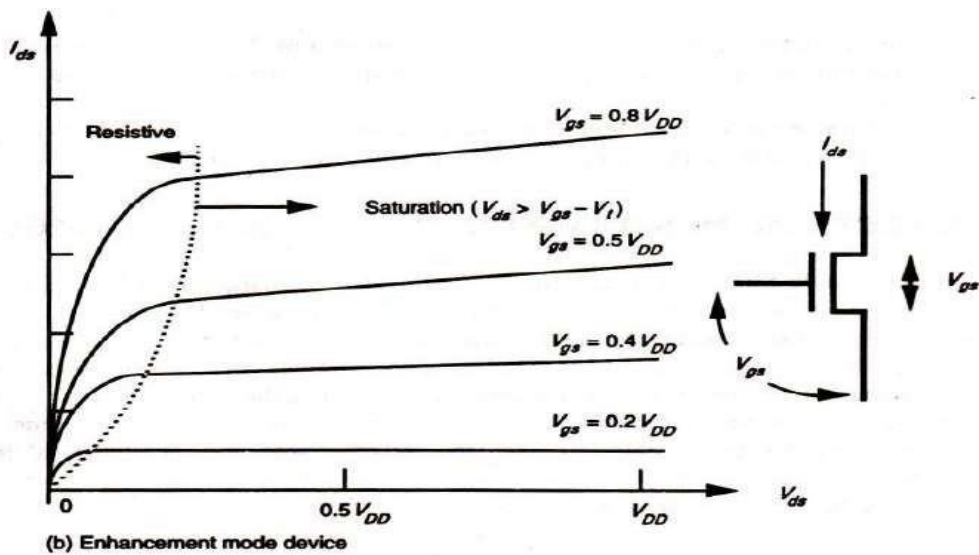
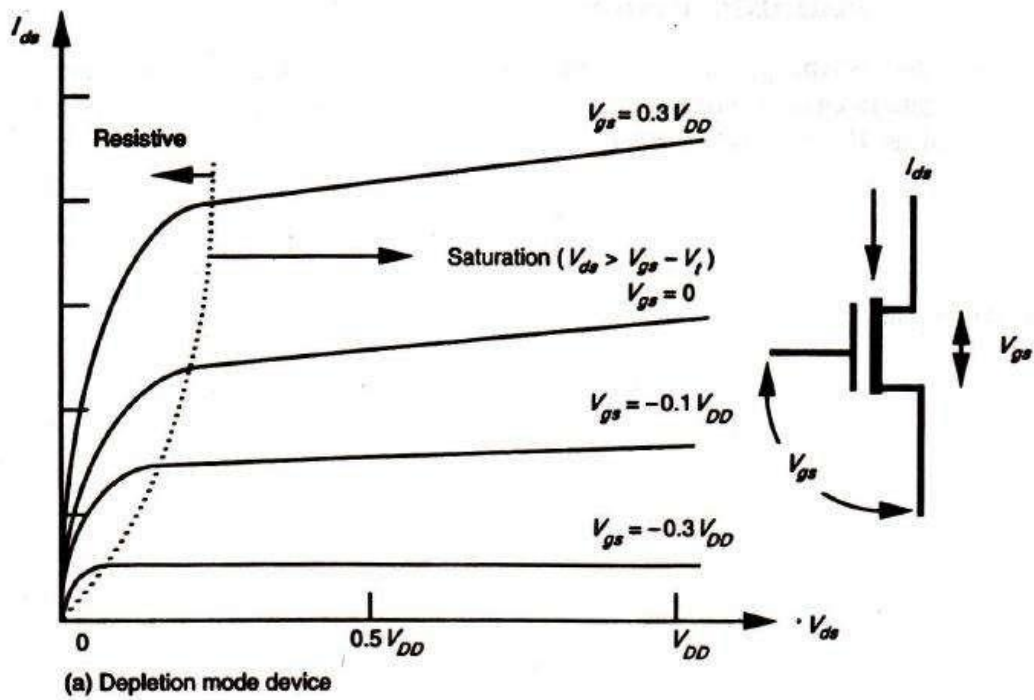


FIGURE 2.2 MOS transistor characteristics.

## 2.2 ASPECTS OF MOS TRANSISTOR THRESHOLD VOLTAGE $V_t$

The gate structure of a MOS transistor consists, electrically, of charges stored in the dielectric layers and in the surface to surface interfaces as well as in the substrate itself.

Switching an enhancement mode MOS transistor from the off to the on state consists in applying sufficient gate voltage to neutralize these charges and enable the underlying silicon to undergo an inversion due to the electric field from the gate.

Switching a depletion mode nMOS transistor from the on to the off state consists in applying enough voltage to the gate to add to the stored charge and invert the 'n' implant region to 'p'.

The threshold voltage  $V_t$  may be expressed as:

$$V_t = \phi_{ms} \frac{Q_B - Q_{SS}}{C_0} + 2\phi_{fN} \quad (2.6)$$

where

$Q_B$  = the charge per unit area in the depletion layer beneath the oxide

$Q_{SS}$  = charge density at Si:SiO<sub>2</sub> interface

$C_0$  = capacitance per unit gate area

$\phi_{ms}$  = work function difference between gate and Si

$\phi_{fN}$  = Fermi level potential between inverted surface and bulk Si.

Now, for polysilicon gate and silicon substrate, the value of  $\phi_{ms}$  is negative but negligible, and the magnitude and sign of  $V_t$  are thus determined by the balance between the remaining negative term  $\frac{-Q_{SS}}{C_0}$  and the other two terms, both of which are positive. To evaluate  $V_t$ , each term is determined as follows:

$$Q_B = \sqrt{2\epsilon_0\epsilon_{Si}qN(2\phi_{fN} + V_{SB})} \text{ coulomb/m}^2$$

$$\phi_{fN} = \frac{kT}{q} \ln \frac{N}{n_i} \text{ volts}$$

$$Q_{SS} = (1.5 \text{ to } 8) \times 10^{-8} \text{ coulomb/m}^2$$

depending on crystal orientation, and where

$V_{SB}$  = substrate bias voltage (negative w.r.t. source for nMOS, positive for pMOS)

$q = 1.6 \times 10^{-19}$  coulomb

$N$  = impurity concentration in the substrate ( $N_A$  or  $N_D$  as appropriate)

$\epsilon_{si}$  = relative permittivity of silicon  $\doteq 11.7$

$n_i$  = intrinsic electron concentration ( $1.6 \times 10^{10}/\text{cm}^3$  at 300°K)

$k$  = Boltzmann's constant =  $1.4 \times 10^{-23}$  joule/°K

The *body effects* may also be taken into account since the substrate may be biased with respect to the source, as shown in Figure 2.3.

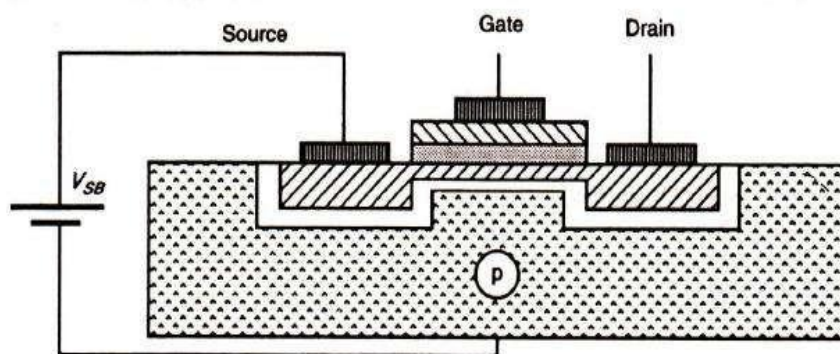


FIGURE 2.3 Body effect (nMOS device shown).

Increasing  $V_{SB}$  causes the channel to be depleted of charge carriers and thus the threshold voltage is raised.

Change in  $V_t$  is given by  $\Delta V_t \doteq \gamma(V_{SB})^{1/2}$  where  $\gamma$  is a constant which depends on substrate doping so that the more lightly doped the substrate, the smaller will be the body effect.

## MOSFET parasitic capacitances :

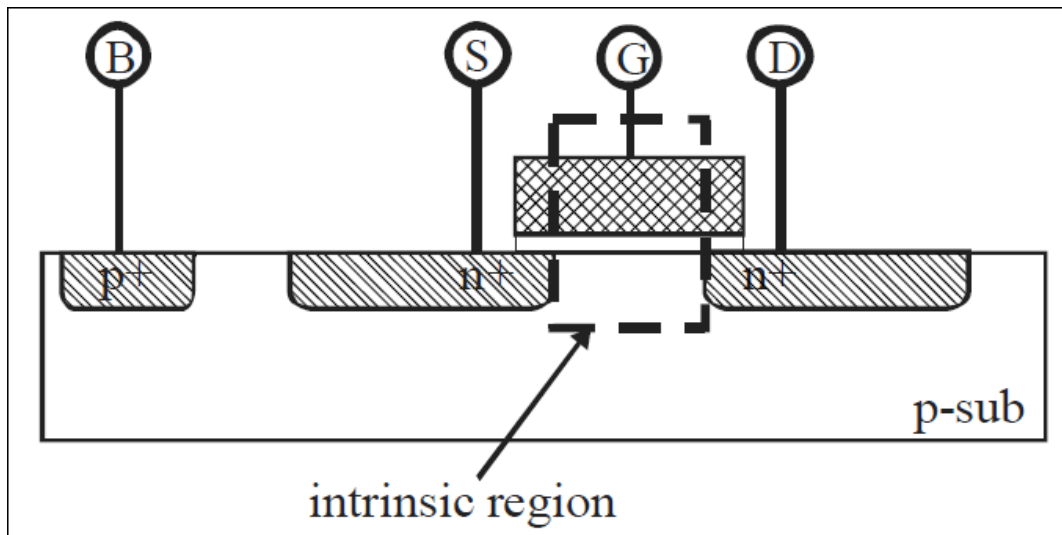
These are subdivided into two general categories:

1. extrinsic capacitances
2. intrinsic capacitances.

Extrinsic capacitances are associated with regions of the transistor outside the dashed line.

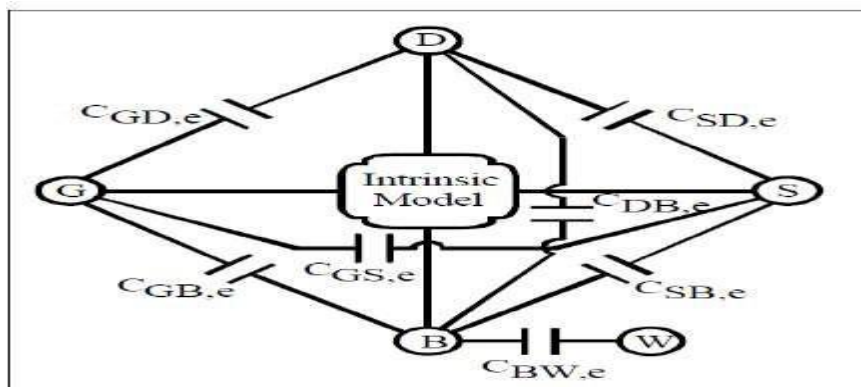
Intrinsic capacitances are all those capacitances located within the boxed region.





### EXTRINSIC CAPACITANCES

Extrinsic capacitances are modeled by using lumped capacitances, each of which is associated with a region of the transistor's geometry. One capacitor is used between each pair of transistor terminals, plus an additional capacitor between the well and the bulk if the transistor is fabricated in a well.

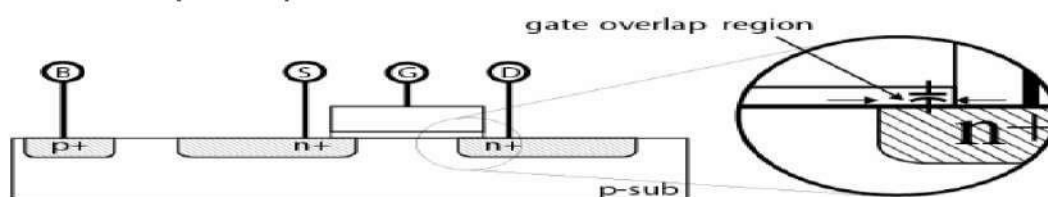


### Extrinsic Capacitance Types

- Overlap capacitances that are mostly dependent on geometry.
- Junction capacitances that are dependent on geometry and on bias.

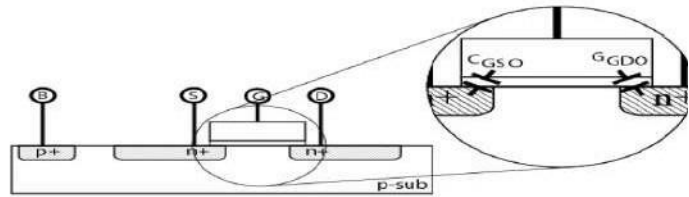
### Gate Overlap Capacitances

There is some overlap between the gate and the source and the gate and the drain. This overlap area gives rise to the gate overlap capacitances.



## Gate-Source/Drain Overlap Capacitances

- The overlap between the gate and the source and the gate and the drain gives rise to the gate overlap capacitances denoted by  $C_{GSO}$  and  $C_{GDO}$  for the gate-to-source overlap capacitance and the gate-to-drain overlap capacitance respectively.
- The overlap capacitances  $C_{GSO}$  and  $C_{GDO}$  are proportional to the width,  $W$ , of the device and the amount that the gate overlaps the source and the drain, typically denoted as "LD" in SPICE parameter files.



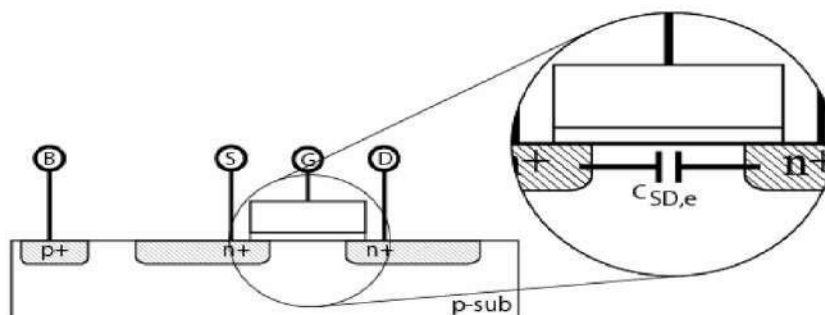
- The overlap capacitances of the source and the drain are often modeled as linear parallel-plate capacitors, since the high dopant concentration in the source and drain regions and the gate material implies that the resulting capacitance is largely bias independent.

## Gate-Source/Drain Overlap Capacitances

- It turns out that fringing field lines add significantly to the total capacitance.
- Estimates of the fringing field capacitances based on measurements are normally used.
- The gate-to-drain overlap capacitances are generally given as measured parameters in the MOSFET model files.
- There are values for NMOS MOSFETs and PMOS MOSFETs.
- The values are "per-width" values.

## Source-Drain Capacitance

- Accurate models of short channel devices may include the capacitance that exists between the source and drain region of the MOSFET. The source-drain capacitance is denoted as  $C_{SD,e}$ .



# Source-Drain Capacitance

- Although the source-drain capacitance originates in the region normally associated with intrinsic capacitance, it is still referred to as an extrinsic capacitance.
- The value of this capacitance is difficult to calculate because its value is highly dependent upon the source and drain geometries. For longer channel devices,  $C_{SD,e}$  is very small in comparison to the other extrinsic capacitances, and is therefore normally ignored.

## Diode Capacitance

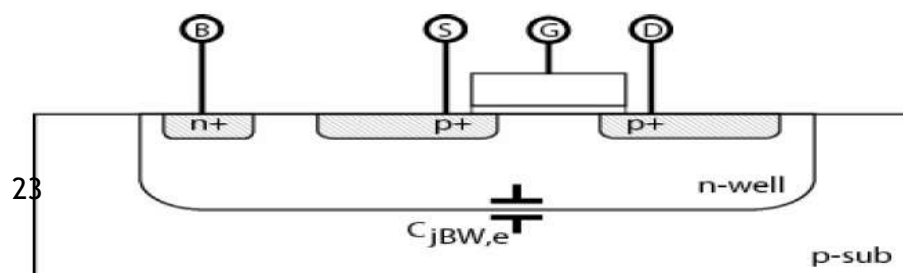
When a reverse voltage is applied to a PN junction, the holes in the p-region are attracted to the anode terminal and electrons in the n-region are attracted to the cathode terminal. The resulting region contains almost no carriers, and is called the depletion region. The depletion region acts similarly to the dielectric of a capacitor. The depletion region increases in width as the reverse voltage across it increases. If we imagine that the diode capacitance can be likened to a parallel plate capacitor, then as the plate spacing (i.e. the depletion region width) increases, the capacitance should decrease. Increasing the reverse bias voltage across the PN junction therefore decreases the diode capacitance.

**Source/Drain-Bulk Junction Capacitances** At the source region there is a source-to bulk junction capacitance,  $C_{jBS,e}$ , and at the drain region there is a drain-to-bulk junction capacitance,  $C_{jBD,e}$ .

**Source/Drain-Bulk Junction Capacitances** The junction capacitances can be calculated by splitting the drain and source regions into a “side-wall” portion and a “bottom-wall” portion.

n The capacitance associated with the side wall portion is found by multiplying the length of the side-wall perimeter (excluding the side contacting the channel) by the effective sidewall capacitance per unit length. The capacitance for the bottom-wall portion is found by multiplying the area of the bottomwall by the bottom-wall capacitance per unit area.

**Well-Bulk Junction Capacitance** If the MOSFET is in a well, a well-to-bulk junction capacitance,  $C_{jBW,e}$ , must be added. The well-bulk junction capacitance is calculated similarly to the source and drain junction capacitances, by dividing the total well-bulk junction capacitance into side-wall and bottom-wall components. If more than one transistor is placed in a well, the well-bulk junction capacitance should only be included once in the total model.



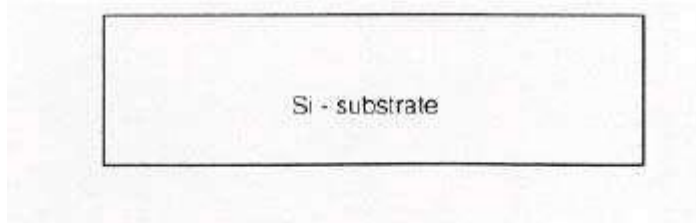
**UNIT-II**  
**VLSI Design styles**



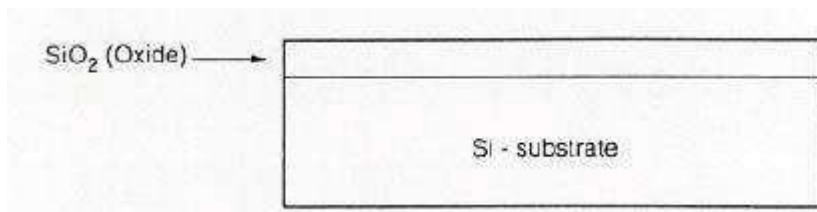
## nMOS FABRICATION

Fabrication is the process to create the devices and wires on a single silicon chip.

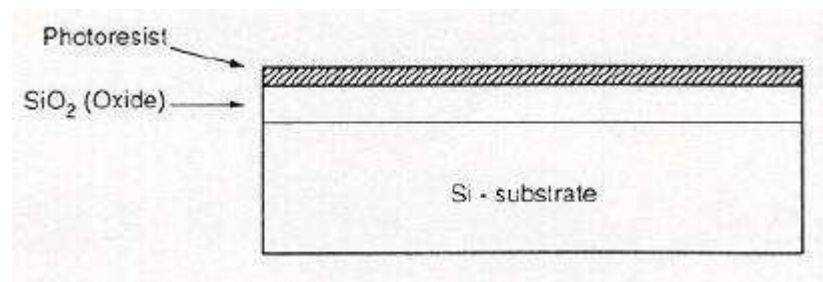
- The process starts with a silicon substrate of high purity into which the required p-impurities are introduced.



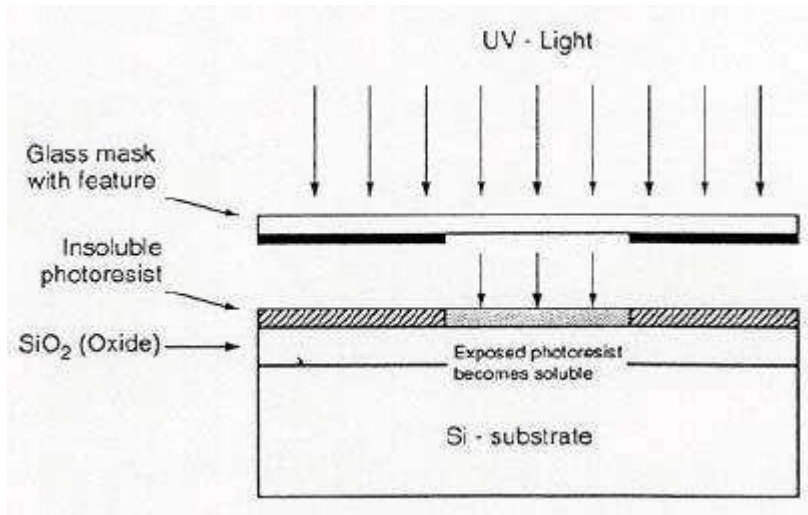
- A layer of silicon dioxide ( $\text{SiO}_2$ ) is grown all over the surface of the wafer to protect the surface and acts as a barrier to dopants during processing and provide a generally insulating substrate onto which other layers may be deposited and patterned.



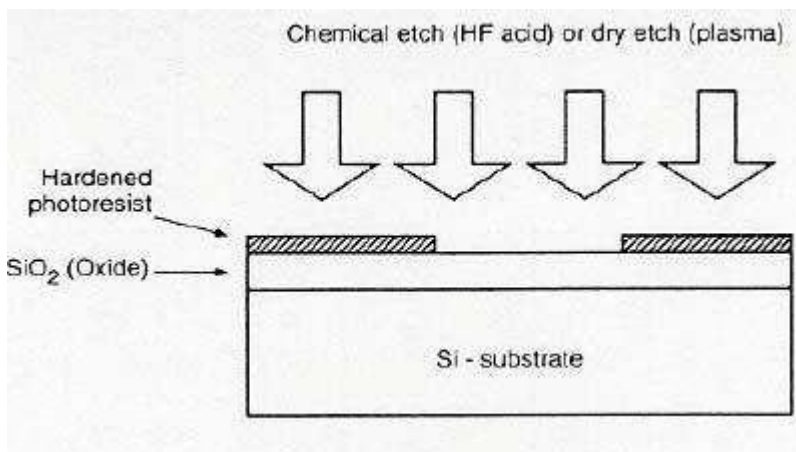
- The surface is now covered with a photoresist which is deposited onto the wafer and spun to achieve an even distribution of the required thickness.



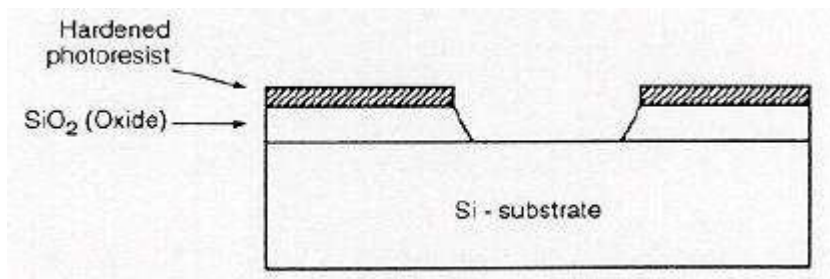
- The photoresist layer is then exposed to ultraviolet light through a mask which defines those regions into which diffusion is to take place together with transistor channels.



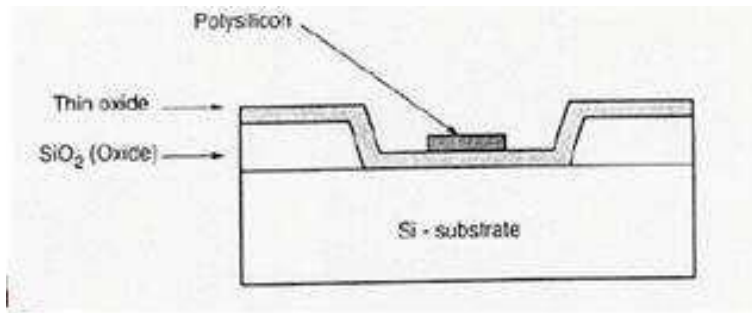
- These areas are subsequently readily etched away together with the underlying silicon dioxide so that the wafer surface is exposed in the window defined by the mask.



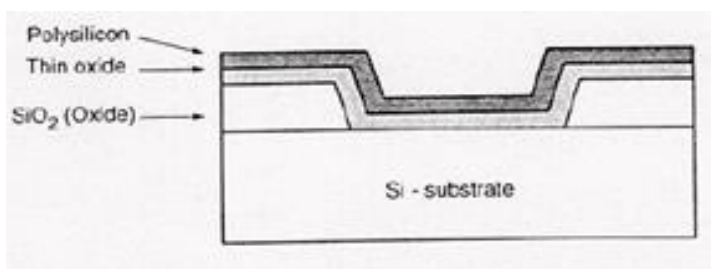
- The remaining photoresist is removed and a thin layer of SiO<sub>2</sub> is grown over the entire chip surface and then polysilicon is deposited on top of this to form the gate structure.



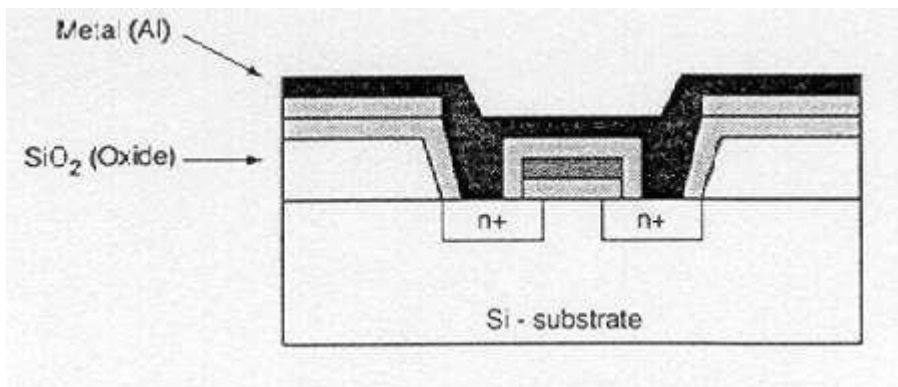
- The polysilicon layer consists of heavily doped polysilicon deposited by chemical vapour deposition (CVD),



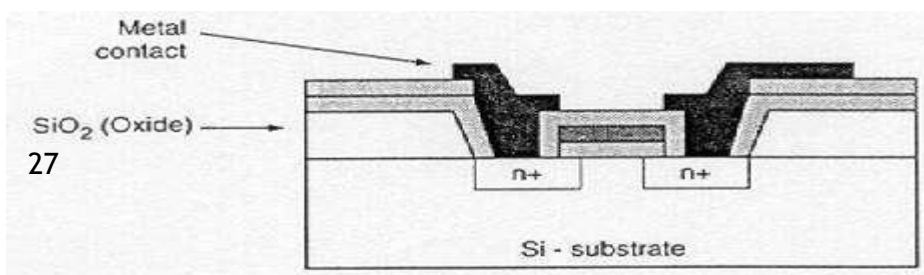
- Further photoresist coating and masking allows the polysilicon to be patterned and then the thin oxide is removed to exposed areas into which n-type impurities are to be diffused to form the source and drain.



- Diffusion is achieved by heating the wafer to a high temperature and passing a gas containing the desired n-type impurity over the surface.



- Thick oxide ( $\text{SiO}_2$ ) is grown over all again and is then masked with photoresist and etched to expose selected areas of the polysilicon gate and the drain and source areas where connections are to be made.



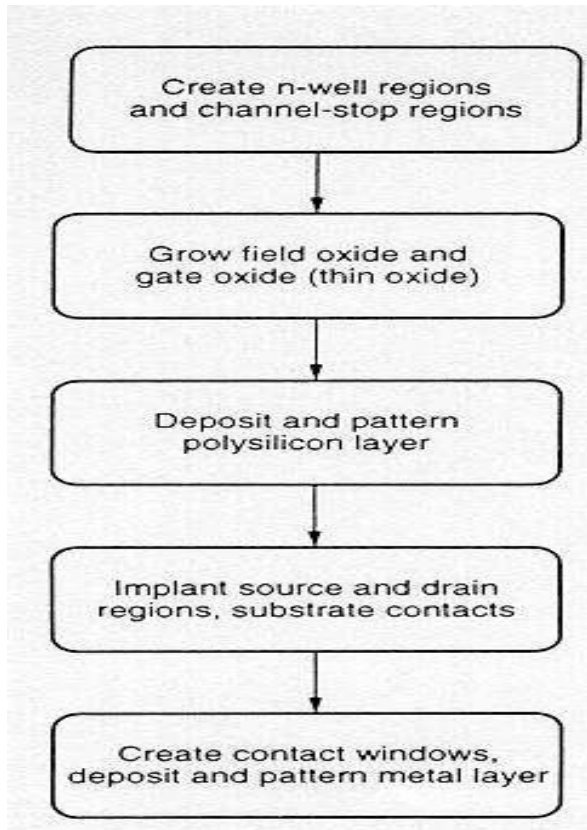
- The whole chip then has metal deposited over the surface to a thickness typically of 1 $\mu$ m. This metal layer is then masked and etched to form the required interconnection pattern.

## **cMOS fabrication**

- CMOS Technology depends on using both N-Type and P-Type devices on the same chip.

The two main technologies to do this task are:

- P-Well (*Will discuss the process steps involved with this technology*)
  - The substrate is N-Type. The N-Channel device is built into a P-Type well within the parent N-Type substrate. The P-channel device is built directly on the substrate.
  - N-Well
  - The substrate is P-Type. The N-channel device is built directly on the substrate, while the P-channel device is built
  - Two more advanced technologies to do this task are: Becoming more popular for sub-micron geometries where device performance and density must be pushed beyond the limits of the conventional p & n-well CMOS processes.
  - Twin Tub
  - Both an N-Well and a P-Well are manufactured on a lightly doped N-type substrate.
  - Silicon-on-Insulator (SOI) CMOS Process
  - SOI allows the creation of independent, completely isolated nMOS and pMOS transistors virtually side-by-side on an insulating substrate.
- The simplified process sequence for the fabrication of CMOS integrated circuits on a p-type silicon substrate is shown.
- The process starts with the creation of the n-well regions for pMOS transistors, by impurity implantation into the substrate.
  - Then, a thick oxide is grown in the regions surrounding the nMOS and pMOS active regions.
  - The thin gate oxide is subsequently grown on the surface through thermal oxidation.
  - These steps are followed by the creation of n<sup>+</sup> and p<sup>+</sup> regions (source, drain and channel-stop implants).
  - Finally the metallization is created (creation of metal interconnects).



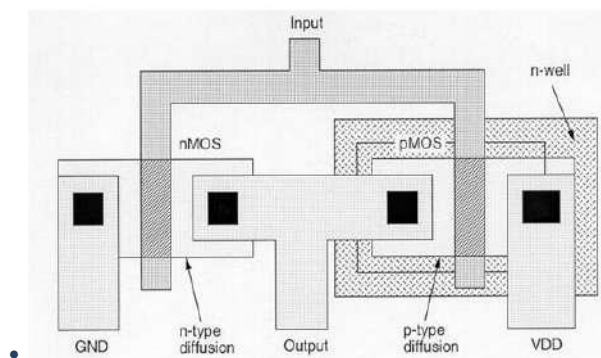
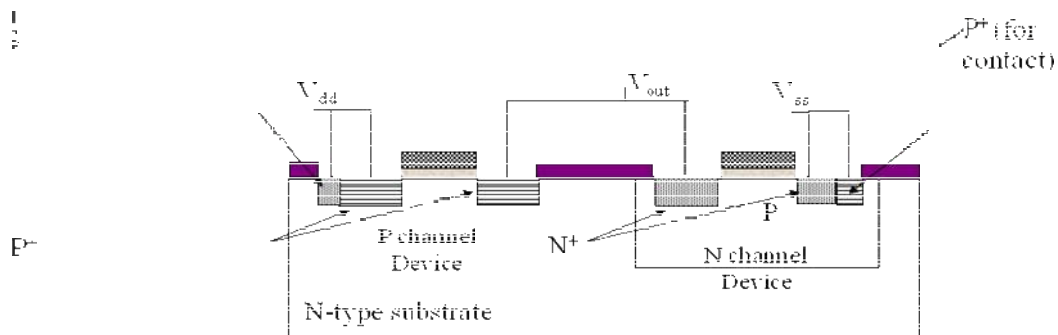
### n-well process

- The n-well CMOS process starts with a moderately doped (impurity concentration  $\sim 10^{16}/\text{cm}^3$ ) p-type silicon substrate. Then, an initial thick “*field*” oxide layer ( $5000\text{\AA}$ ) is grown on the entire surface.
- The first lithographic mask defines the n-well region. Donor atoms, usually phosphorus, are implanted through this window in the oxide. Once the n-well is created, the active areas of the nMOS and pMOS transistors can be defined.
- Following the creation of the n-well region, a thick field oxide is grown around the transistor active regions, and a thin gate oxide ( $25\text{\AA}$ ) is grown on top of the active regions
- The polysilicon layer ( $3000\text{\AA}$ ) is deposited using chemical vapor deposition (CVD) and patterned by dry **plasma etching**. The created polysilicon lines will function as the gate electrodes of the nMOS and the pMOS transistors and their interconnects
- Using a set of two masks, the n<sup>+</sup> and p<sup>+</sup> **Source** and **Drain** regions are implanted into the substrate and into the n-well, respectively.
- The ohmic contacts to the substrate and to the n-well are implanted in this process step
- An insulating silicon dioxide layer is deposited over the entire wafer using CVD ( $5000\text{\AA}$ ). This is for *passivation*, the protection of all the active components from contamination.
- The contacts are defined and etched away to expose the silicon or polysilicon contact windows.

These contact windows are necessary to complete the circuit interconnections using the metal layer, which is patterned in the next step.

- Metal (aluminum,  $>5000\text{\AA}$ ) is deposited over the entire chip surface using metal evaporation, and the metal lines are patterned through etching.
- Since the wafer surface is non-planar, the quality and the integrity of the metal lines created in this step are very critical and are ultimately essential for circuit reliability.
- The composite layout and the resulting cross-sectional view of the chip, showing one nMOS and one pMOS transistor (built-in n-well), the polysilicon and metal interconnections.
- The final step is to deposit a full  $\text{SiO}_2$  passivation layer ( $5000\text{\AA}$ ), for protection, over the chip, except for wire-bonding pad areas.

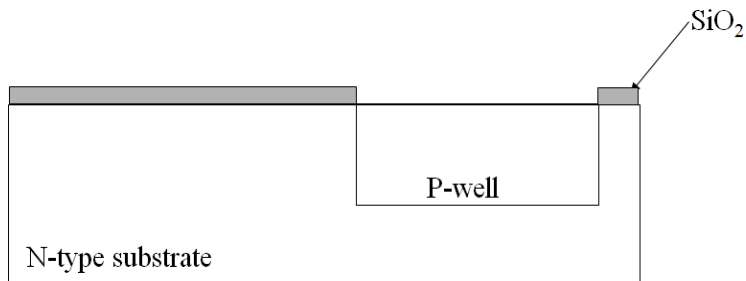
P-substrate



### p-well process

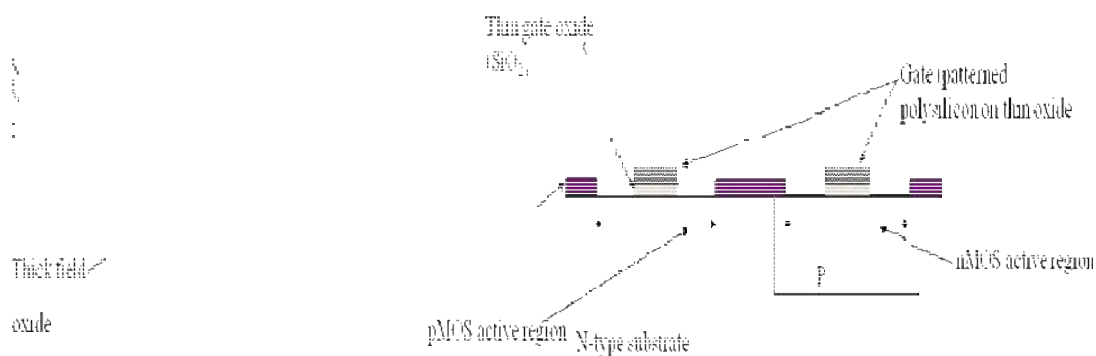
- ~~P~~well on N-substrate
- N-type substrate
- Oxidation, and mask (MASK 1) to create P-well ( $4-5\mu\text{m}$  deep)

- P-well doping
- P-well acts as substrate for nMOS devices.
- The two areas are electrically isolated using thick field oxide (and often
- isolation implants [not shown here])

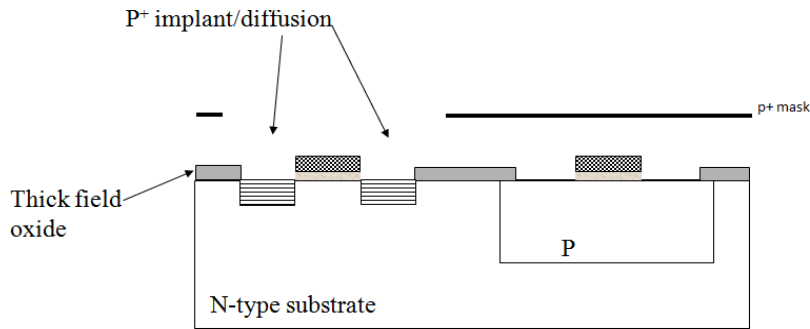


### Polysilicon Gate Formation

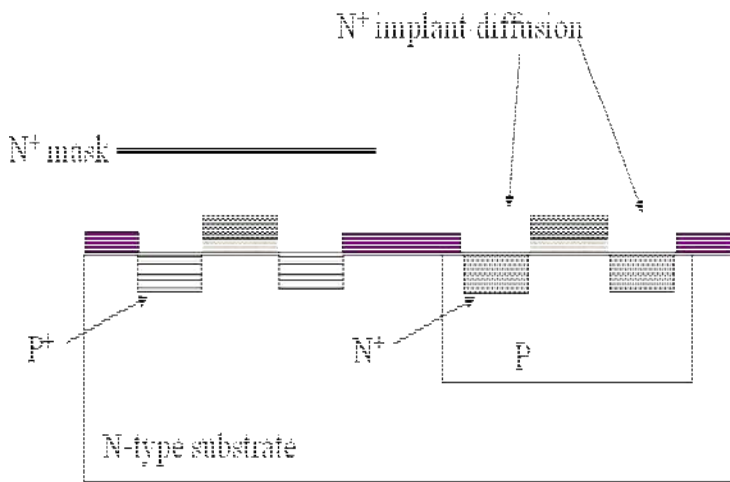
- Remove p-well definition oxide
- Grow thick field oxide
- Pattern (MASK 2) to expose nMOS and pMOS active regions
- Grow thin layer of SiO<sub>2</sub> (~0.1μm) gate oxide, over the entire chip surface
- Deposit polysilicon on top of gate oxide to form gate structure
- Pattern poly on gate oxide (MASK 3)



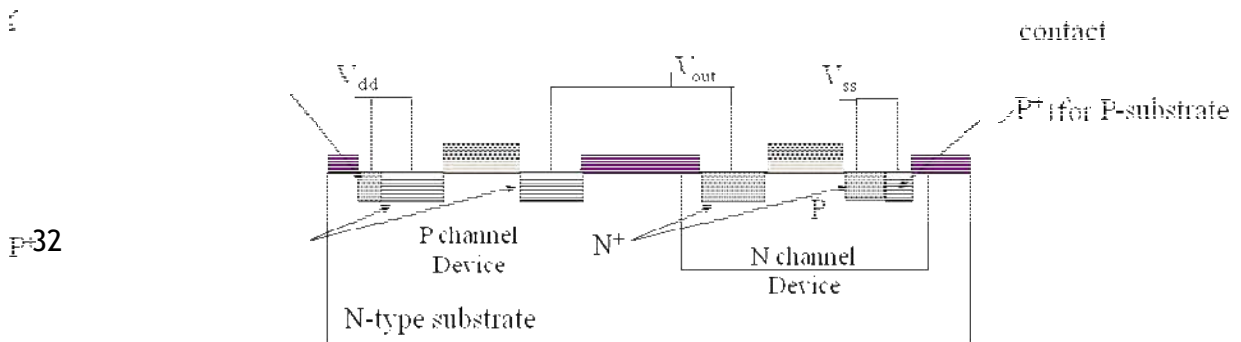
- nMOS P<sup>+</sup> Source/Drain diffusion – self-aligned to Poly gate
- Implant P<sup>+</sup> nMOS S/D regions (MASK 4)



- pMOS N+ Source/Drain diffusion – self-aligned to Poly gate
- Implant N+ pMOS S/D regions (MASK 5 – often the inverse of MASK 4)



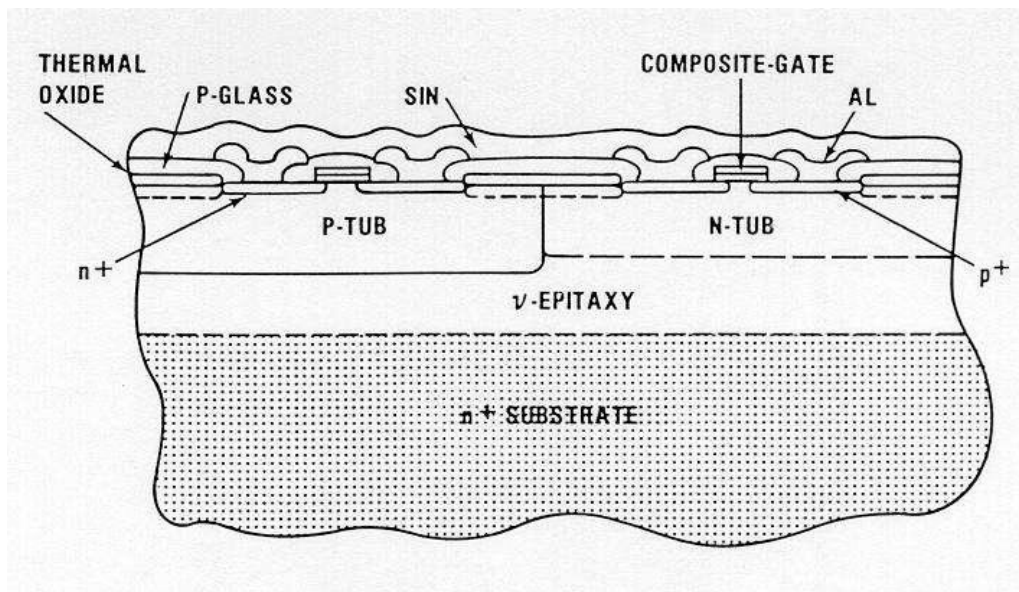
- pMOS N+ Source/Drain diffusion, contact holes & metallisation
- Oxide and pattern for contact holes (MASK 6)
- Deposit metal and pattern (MASK 7)
- Passivation oxide and pattern bonding pads (MASK 8)
- P-well acts as substrate for nMOS devices.
- Two separate substrates : requires two separate substrate connections
- Definition of substrate connection areas can be included in MASK 4/MASK5





## Twin-Tub (Twin-Well) CMOS Process

This technology provides the basis for separate optimization of the nMOS and pMOS transistors, thus making it possible for threshold voltage, body effect and the channel transconductance of both types of transistors to be tuned independently. Generally, the starting material is a n+ or p+ substrate, with a lightly doped epitaxial layer on top. This epitaxial layer provides the actual substrate on which the n-well and the p-well are formed. Since two independent doping steps are performed for the creation of the well regions, the dopant concentrations can be carefully optimized to produce the desired device characteristics. The Twin-Tub process is shown below.



In the conventional p & n-well CMOS process, the doping density of the well region is typically about one order of magnitude higher than the substrate, which, among other effects, results in unbalanced drain parasitics. The twin-tub process avoids this problem.

### Noise Margin

*Noise margin* is closely related to the DC voltage characteristics [Wakerly00]. This parameter allows you to determine the allowable noise voltage on the input of a gate so that the output will not be corrupted. The specification most commonly used to describe noise margin (or *noise immunity*) uses two parameters: the *LOW* noise margin, *NML*, and the *HIGH* noise margin, *NMH*. *NML* is defined as the difference in maximum *LOW* input voltage recognized by the receiving gate and the maximum *LOW* output voltage produced by the driving gate.

$$NM_L = V_{IL} - V_{OL}$$

The  $V_{OH}$  value of *NMH* is the difference between the minimum *HIGH* output voltage of the driving gate and the minimum *HIGH* input voltage recognized by the receiving gate.

Thus,

$$NM_H = V_{OH} - V_{IH}$$

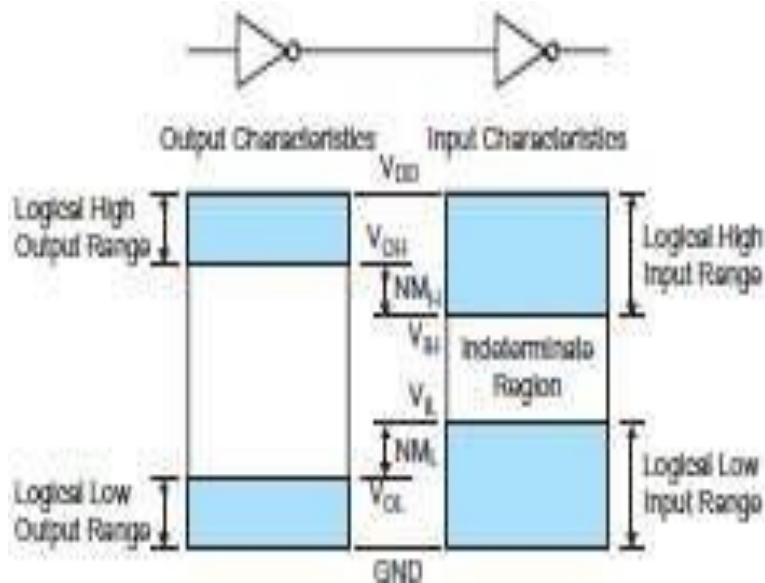
where

$V_{IH}$  = minimum HIGH input voltage

$V_{IL}$  = maximum LOW input voltage

$V_{OH}$  = minimum HIGH output voltage

$V_{OL}$  = maximum LOW output voltage

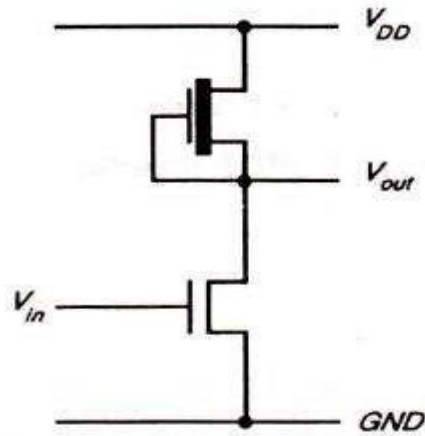


Inputs between  $V_{IL}$  and  $V_{IH}$  are said to be in the *indeterminate region* or *forbidden zone* and do not represent legal digital logic levels. Therefore, it is generally desirable to have  $V_{IH}$  as close as possible to  $V_{IL}$  and for this value to be midway in the “logic swing,”  $V_{OL}$  to  $V_{OH}$ . This implies that the transfer characteristic should switch abruptly; that is, there should be high gain in the transition region. For the purpose of calculating noise margins, the transfer characteristic of the inverter and the definition of voltage levels  $V_{IL}$ ,  $V_{OL}$ ,  $V_{IH}$ , and  $V_{OH}$  are shown in Figure above. Logic levels are defined at the unity gain point where the slope is  $-1$ . This gives a conservative bound on the worst case static noise margin .

### THE nMOS INVERTER

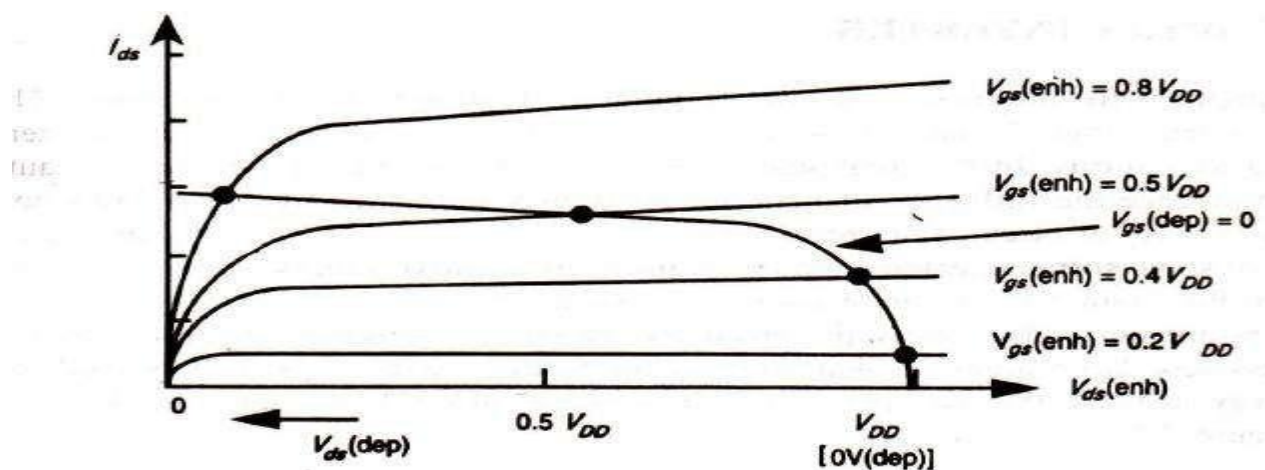
A basic requirement for producing a complete range of logic circuits is the inverter. This is needed for restoring logic levels, for *Nand* and *Nor* gates, and for sequential and memory circuits of various forms . The basic inverter circuit requires a transistor with source connected to ground and a load resistor of some sort connected from the drain to the positive supply rail  $V_{DD}$ . The output is taken from the drain and the input applied between gate and ground. Resistors are not conveniently produced on the silicon substrate; even modest values occupy excessively large areas so that some other form of load resistance is required. A convenient way to solve this problem is to use a depletion mode transistor as the load, as

shown in Figure 2.5.



**FIGURE 2.5 nMOS inverter.**

- With no current drawn from the output, the currents  $I_{ds}$  for both transistors must be equal.
- For the depletion mode transistor, the gate is connected to the source so it is always on and only the characteristic curve  $V_{gs} = 0$  is relevant.
- In this configuration the depletion mode device is called the pull-up (p.u.) and the enhancement mode device the pull-down (p.d.) transistor.
- To obtain the inverter transfer characteristic we superimpose the  $V_{gs} = 0$  depletion mode characteristic curve on the family of curves for the enhancement mode device, noting that maximum voltage across the enhancement mode device corresponds to minimum voltage across the depletion mode transistor.
- The points of intersection of the curves as in Figure *f-6* give points on the transfer characteristic, which is of the form shown in Figure 2.7.
- Note that as  $V_{in}(=V_{gs}$  p.d. transistor) exceeds the p.d. threshold voltage current begins to flow. The output voltage  $V_{out}$  thus decreases and the subsequent increases in  $V_{in}$  will cause the p.d. transistor to come out of saturation and become resistive. Note that the p.u. transistor is initially resistive as the p.d. turns on.



$$V_{ds}(enh) = V_{DD} - V_{ds}(dep) = V_{out}$$

35

$V_{gs}(enh) = V_{in}$  . . . intersection points give transfer characteristic

**FIGURE 2.6 Derivation of nMOS Inverter transfer characteristic.**

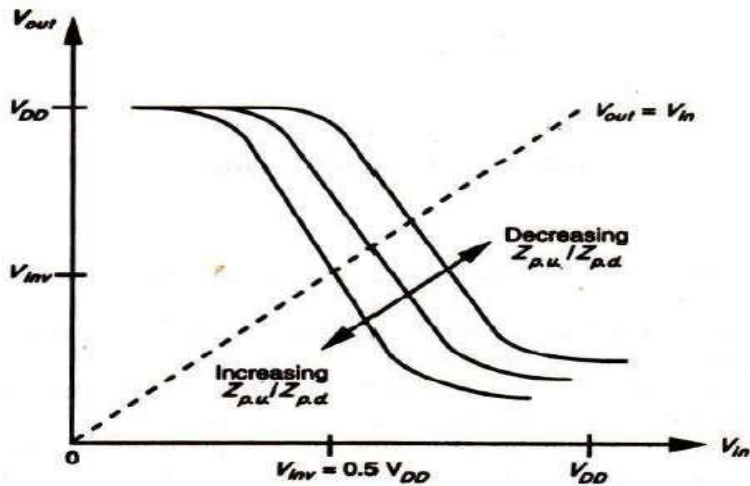


FIGURE 2.7 nMOS Inverter transfer characteristic.

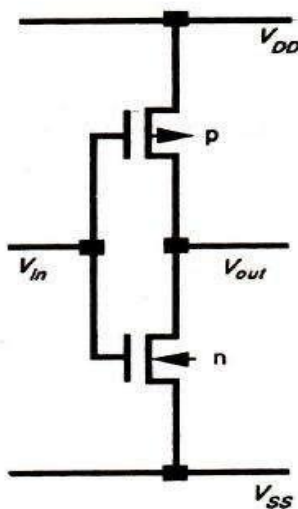
during transition, the slope of the transfer characteristic determines the gain:

$$\text{Gain} = \frac{\delta V_{out}}{\delta V_{in}}$$

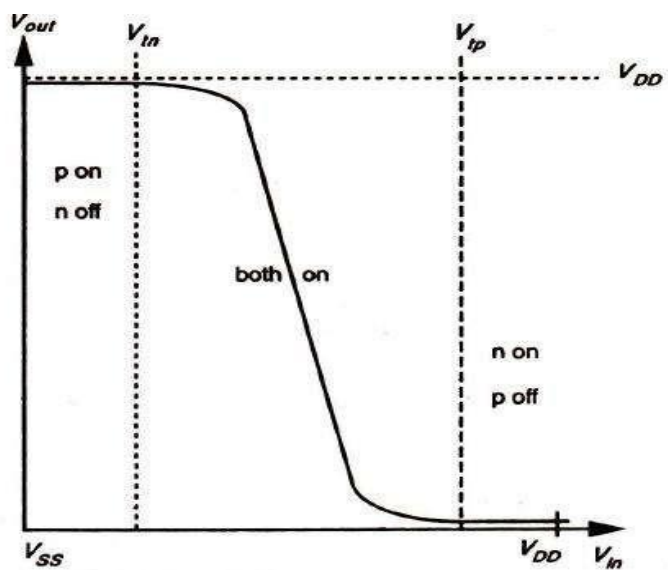
The point at which  $V_{out} = V_{in}$  is denoted as  $V_{inv}$  and it will be noted that the transfer characteristics and  $V_{inv}$  can be shifted by variation of the ratio of pull-up to pull-down resistances (denoted  $Z_{p.u.}/Z_{p.d.}$  where  $Z$  is determined by the length to width ratio of the transistor in question).

### THE CMOS INVERTER

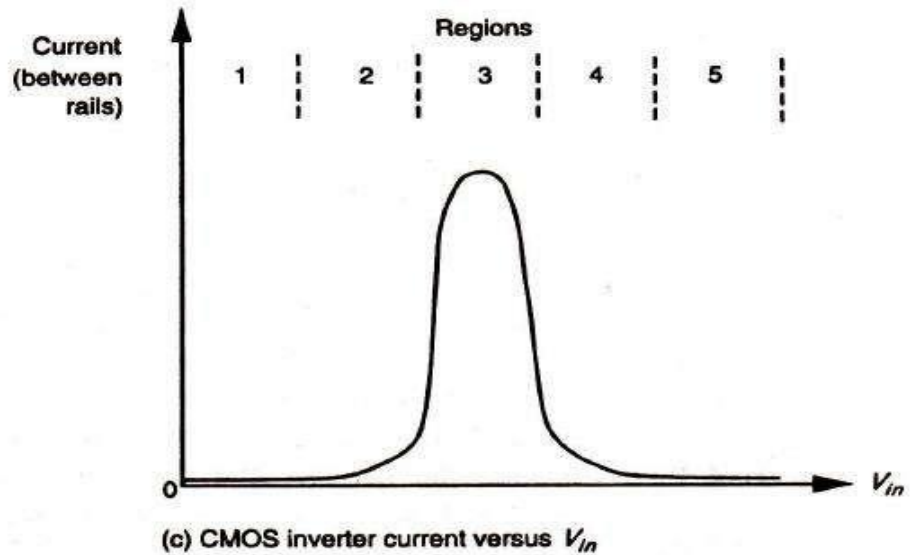
- (a) No current flow either for logical 0 or for logical 1 inputs.
- (b) Full logical 1 and 0 levels are presented at the output.
- (c) For devices of similar dimensions the p-channel is slower than the n-channel device.



36 (a) Circuit



(b) Transfer characteristic



**FIGURE 2.14 Complementary transistor pull-up (CMOS).**

The general arrangement and characteristics are illustrated in Figure 2.14. We have seen (equations 2.4 and 2.5) that the current/voltage relationships for the MOS transistor may be written

$$I_{ds} = K \frac{W}{L} (V_{gs} - V_t) V_{ds} - \frac{V_{ds}^2}{2}$$

in the resistive region, or

$$I_{ds} = K \frac{W}{L} \frac{(V_{gs} - V_t)^2}{2}$$

in saturation. In both cases the factor  $K$  is a technology-dependent parameter such that

$$K = \frac{\epsilon_{ins} \epsilon_0 \mu}{D}$$

The factor  $W/L$  is, of course, contributed by the geometry and it is common practice to write

$$\beta = K \frac{W}{L}$$

so that, for example

$$I_{ds} = \frac{\beta}{2} (V_{gs} - V_t)^2$$

in saturation, and where  $\beta$  may be applied to both nMOS and pMOS transistors as follows:

$$\beta_n = \frac{\epsilon_{ins} \epsilon_0 \mu_n}{D} \frac{W_n}{L_n}$$

$$\beta_p = \frac{\epsilon_{ins} \epsilon_0 \mu_p}{D} \frac{W_p}{L_p}$$

where  $W_n$  and  $L_n$ ,  $W_p$  and  $L_p$  are the n- and p-transistor dimensions respectively. With regard to Figures 2.14(b) and 2.14(c), it may be seen that the CMOS inverter has five distinct regions of operation.



Considering the static conditions first, it may be seen that in *region 1* for which  $V_{in} = \text{logic } 0$ , we have the p-transistor fully turned on while the n-transistor is fully turned off. Thus no current flows through the inverter and the output is directly connected to  $V_{DD}$  through the p-transistor. A good logic 1 output voltage is thus present at the output.

In *region 5*  $V_{in} = \text{logic } 1$ , the n-transistor is fully on while the p-transistor is fully off. Again, no current flows and a good logic 0 appears at the output.

In *region 2* the input voltage has increased to a level which just exceeds the threshold voltage of the n-transistor. The n-transistor conducts and has a large voltage between source and drain; so it is in saturation. The p-transistor is also conducting but with only a small voltage across it, it operates in the unsaturated resistive region. A small current now flows through the inverter from  $V_{DD}$  to  $V_{out}$ . If we wish to analyze the behavior in this region, we equate the p-device resistive region current with the n-device saturation current and thus obtain the voltage and current relationships.

*Region 4* is similar to region 2 but with the roles of the p- and n-transistors reversed. However, the current magnitudes in regions 2 and 4 are small and most of the energy consumed in switching from one state to the other is due to the larger current which flows in region 3.

*Region 3* is the region in which the inverter exhibits gain and in which both transistors are in saturation.

The currents (with regard to Figure 2.14(c)) in each device must be the same: since the transistors are in series, so we may write

$$I_{dsp} = -I_{dsn}$$

where

$$I_{dsp} = \frac{\beta_p}{2} (V_{in} - V_{DD} - V_{tp})^2$$

and

$$I_{dsn} = \frac{\beta_n}{2} (V_{in} - V_{tn})^2$$

from whence we can express  $V_{in}$  in terms of the  $\beta$  ratio and the other circuit voltages and currents

$$V_{in} = \frac{V_{DD} + V_{tp} + V_{tn}(\beta_n/\beta_p)^{1/2}}{1 + (\beta_n/\beta_p)^{1/2}} \quad (2.10)$$

Since both transistors are in saturation, they act as current sources so that the equivalent circuit in this region is two current sources in series between  $V_{DD}$  and  $V_{SS}$  with the output voltage coming from their common point. The region is inherently unstable in consequence and the changeover from one logic level to the other is rapid.

If  $\beta_n = \beta_p$  and if  $V_{in} = -V_{ip}$ , then from equation (2.10).

$$V_{in} = 0.5 V_{DD}$$

This implies that the changeover between logic levels is symmetrically disposed about the point at which

$$V_{in} = V_{out} = 0.5 V_{DD}$$

since only at this point will the two  $\beta$  factors be equal. But for  $\beta_n = \beta_p$  the device geometries must be such that

$$\mu_p W_p / L_p = \mu_n W_n / L_n$$

Now the mobilities are inherently unequal and thus it is necessary for the width to length ratio of the p-device to be two to three times that of the n-device, namely

$$W_p / L_p \approx 2.5 W_n / L_n$$

However, it must be recognized that mobility  $\mu$  is affected by the transverse electric field in the channel and is thus dependent on  $V_{gs}$  (and thus on  $V_{in}$  in this case). It has been shown empirically that the actual mobility is

$$\mu = \mu_z (1 - \phi (V_{gs} - V_t))^{-1}$$

$\phi$  is a constant approximately equal to 0.05,  $V_t$  includes any body effect, and  $\mu_z$  is the mobility with zero transverse field. Thus a  $\beta$  ratio of 1 will only hold good around the point of symmetry when  $V_{out} = V_{in} = 0.5 V_{DD}$ .

The  $\beta$  ratio is often unimportant in many configurations and in most cases minimum size transistor geometries are used for both n- and p-devices. Figure 2.15 indicates the trends in the transfer characteristic as the ratio is varied. The changes indicated in the figure would be for quite large variations in  $\beta$  ratio (e.g. up to 10:1) and the ratio is thus not too critical in this respect.

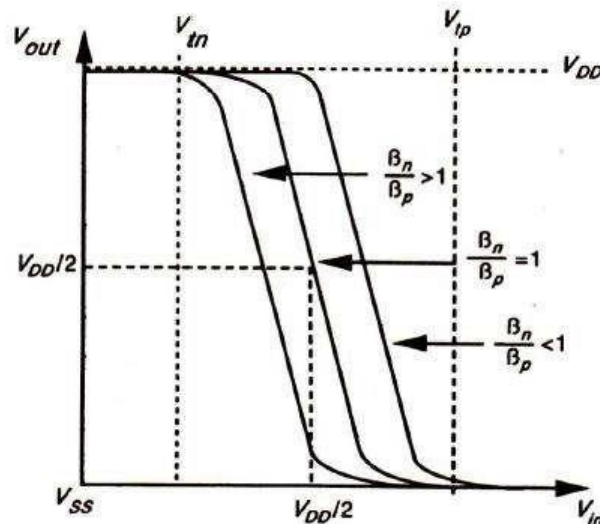
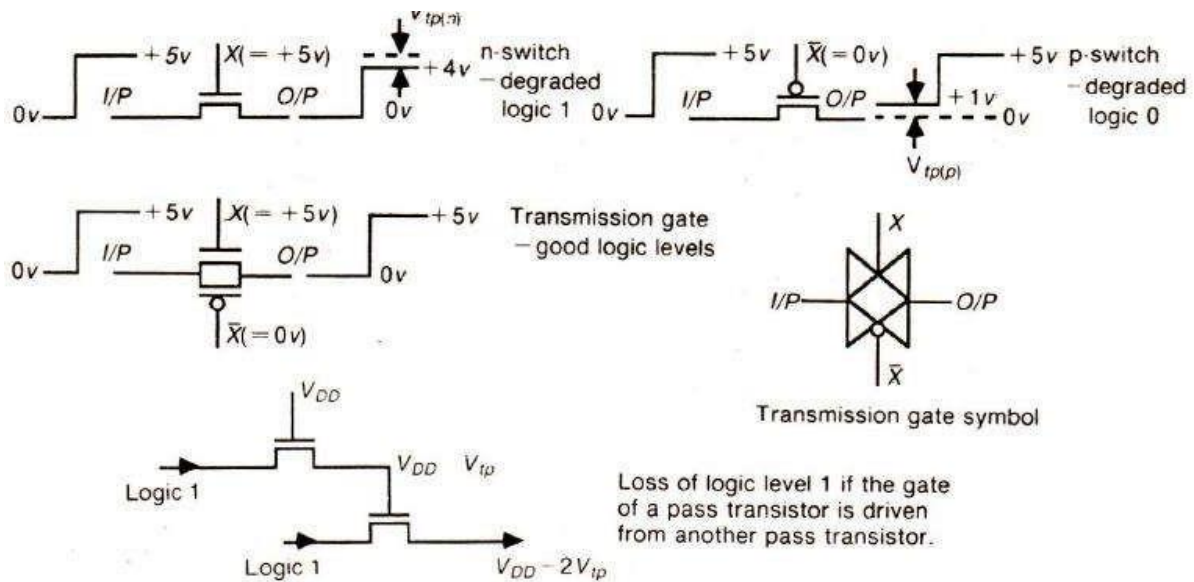


FIGURE 2.15 Trends in transfer characteristic with  $\beta$  ratio.

Switches and switch logic may be formed from simple n- or p-pass transistors or from transmission gates (complementary switches) comprising an n-pass and a p-pass transistor in parallel as shown in Figure 6.2. The reason for adopting the apparent complexity of the transmission gate, rather than using a simple n-switch or p-switch in most CMOS applications, is to eliminate the undesirable threshold voltage effects which give rise to the loss of logic levels in pass transistors as indicated in Figure 6.2. No such degradation occurs with the transmission gate, but more area is occupied and complementary signals are needed to drive it. 'On' resistance, however, is lower than that of the simple pass transistor switches.

When using nMOS switch logic, there is one restriction which must always be observed: *no* pass transistor gate input may be driven through *one or more pass transistors* (see Figure 6.2). As shown, logic levels propagated through pass transistors are degraded by threshold voltage effects. Since the sign~l out of pass transistor *T1* does not reach a full logic 1, but rather a voltage one transistor threshold below a true logic 1, this degraded voltage would not permit the output of *T2* to reach an acceptable logic 1 level.



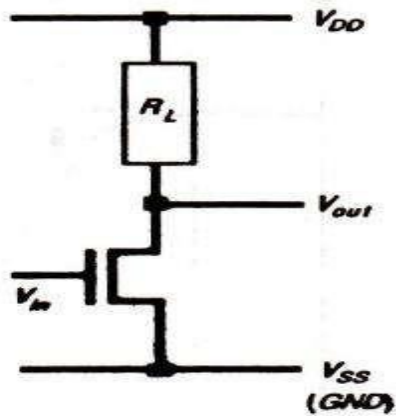
**FIGURE 6.2** Some properties of pass transistors and transmission gates.

### ALTERNATIVE FORMS OF PULL-UP

Up to now we have assumed that the inverter circuit has a depletion mode pull-up transistor as its load. There are, however; at least four possible arrangements:

1. *Load resistance RL* (Figure 2.11 ). This arrangement is not often used because of the large space requirements of resistors produced in a silicon substrate.

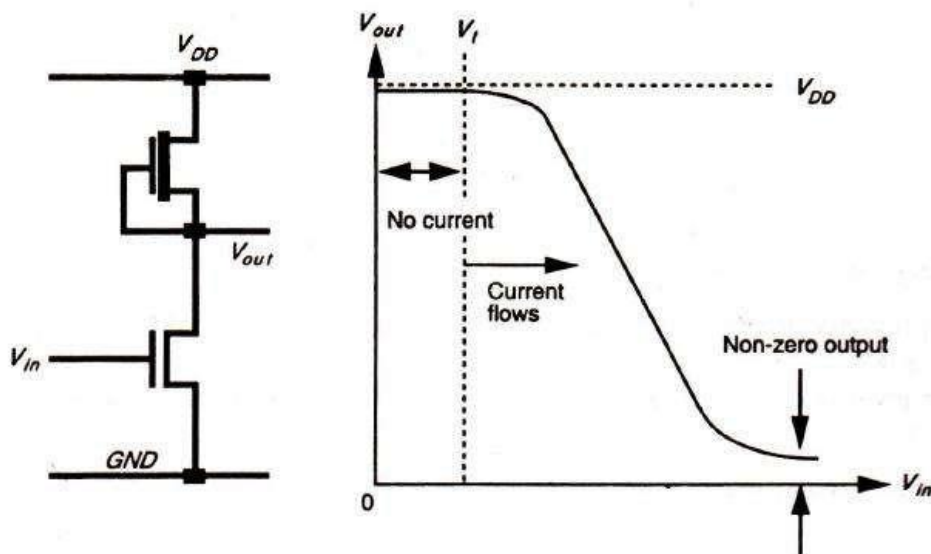




**FIGURE 2.11 Resistor pull-up.**

2. *nMOS depletion mode transistor pull-up* (Figure 2.12).

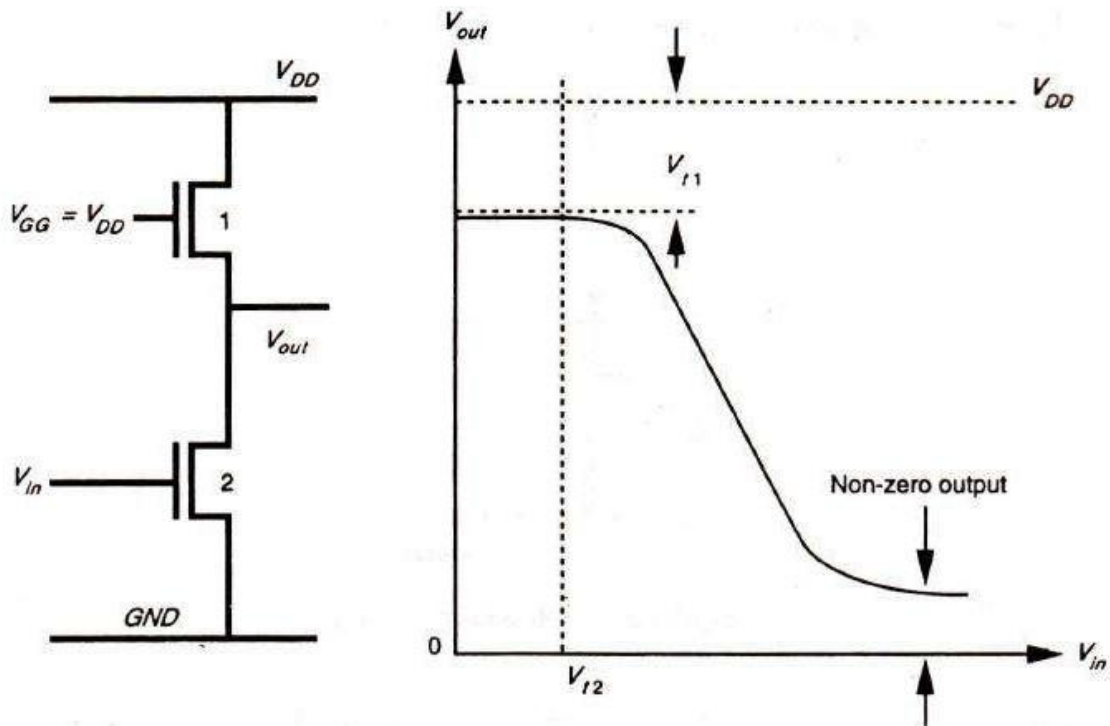
- (a) Dissipation is high, since rail to rail current flows when  $V_{in} = \text{logical } 1$ .
- (b) Switching of output from 1 to 0 begins when  $V_{in}$  exceeds  $V_t$  of p.d. device.
- (c) When switching the output from 1 to 0, the p.u. device is non-saturated initially and this presents lower resistance through which to charge capacitive loads.



**FIGURE 2.12 nMOS depletion mode transistor pull-up and transfer characteristic.**

3. *nMOS enhancement mode pull-up* (Figure 2.13).

- (a) Dissipation is high since current flows when  $V_{in} = \text{logical } 1$  ( $V_{GG}$  is returned to  $V_{DD}$ ).
- (b)  $V_{out}$  can never reach  $V_{DD}$  (logical 1) if  $V_{GG} = V_{DD}$  as is normally the case.



**FIGURE 2.13** nMOS enhancement mode pull-up and transfer characteristic.

(c)  $V_{GG}$  may be derived from a switching source, for example, one phase of a clock, so that dissipation can be greatly reduced.

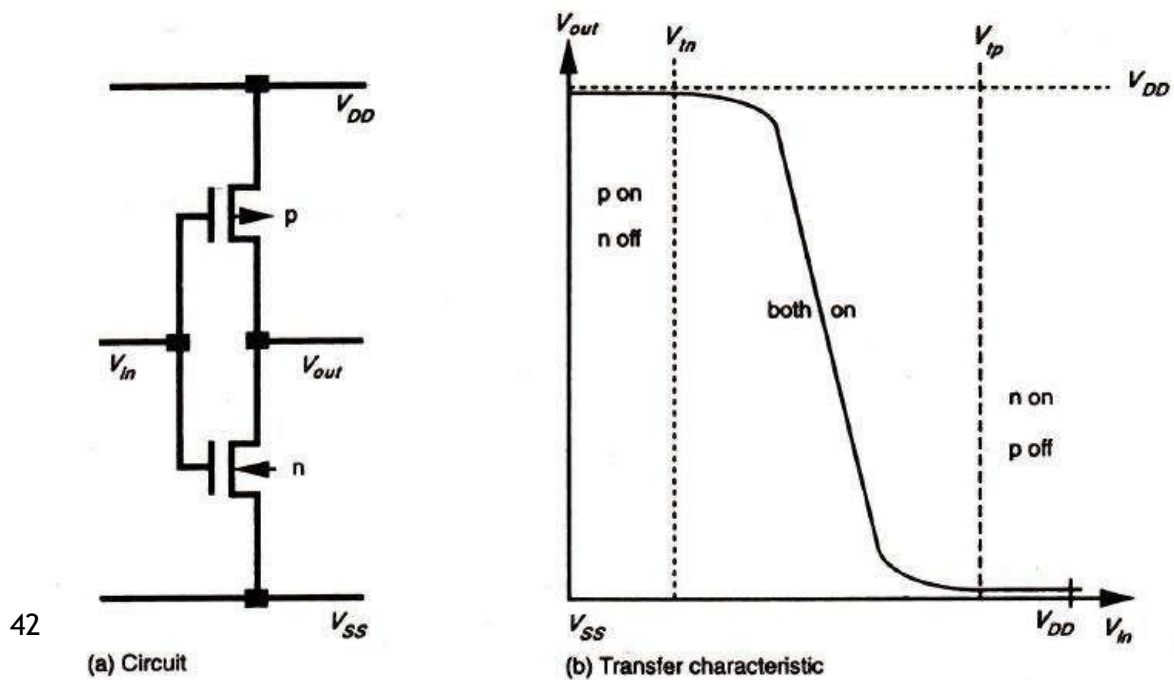
(d) If  $V_{GG}$  is higher than  $V_{DD}$  then an extra supply rail is required.

4. *Complementary transistor pull-up (CMOS)* (Figure 2.14).

(a) No current flow either for logical 0 or for logical 1 inputs.

(b) Full logical 1 and 0 levels are presented at the output.

(c) For devices of similar dimensions the p-channel is slower than the n-channel device.



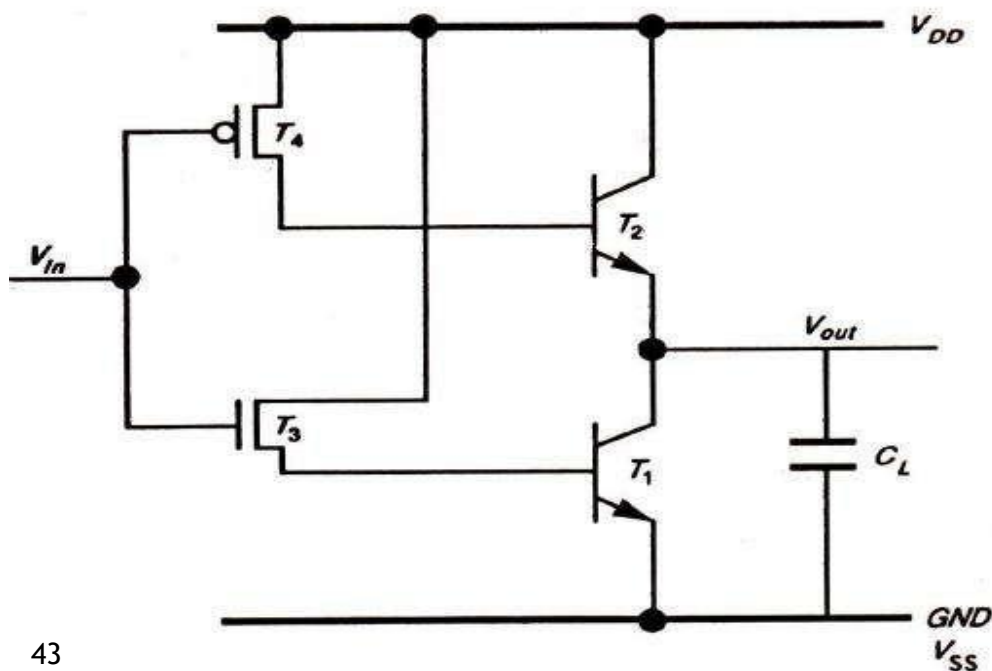
**FIGURE 2.14 Complementary transistor pull-up (CMOS).**

### BICMOS Inverters

As in nMOS and CMOS logic circuitry, the basic logic element is the inverter circuit. When designing with BiCMOS in mind, the logical approach is to use MOS switches to perform the logic function and bipolar transistors to drive the output loads. The simplest logic function is that of inversion, and a simple BiCMOS inverter circuit is readily set out as shown in Figure 2.17.

It consists of two bipolar transistors  $T_1$  and  $T_2$  with one nMOS transistor  $T_3$ , and one pMOS transistor  $T_4$ , both being enhancement mode devices. The action of the circuit is straight forward and may be described as follows:

- With  $V_{in} = 0$  volts (GND)  $T_3$  is off so that  $T_1$  will be non-conducting. But  $T_4$  is on and supplies current to the base of  $T_2$  which will conduct and act as a current source to charge the load  $C_L$  toward +5 volts ( $V_{DD}$ ). The output of the inverter will rise to +5 volts less the base to emitter voltage  $V_{BE}$  of  $T_2$ .
- With  $V_{in} = +5$  volts ( $V_{DD}$ )  $T_4$  is off so that  $T_2$  will be non-conducting. But  $T_3$  will now be on and will supply current to the base of  $T_1$  which will conduct and act as a current sink to the load  $C_L$  discharging it toward 0 volts (GND). The output of the inverter will fall to 0 volts plus the saturation voltage  $V_{CEsat}$  from the collector to the emitter of  $T_1$ .
- $T_1$  and  $T_2$  will present low impedances when turned on into saturation and the load  $C_L$  will be charged or discharged rapidly.

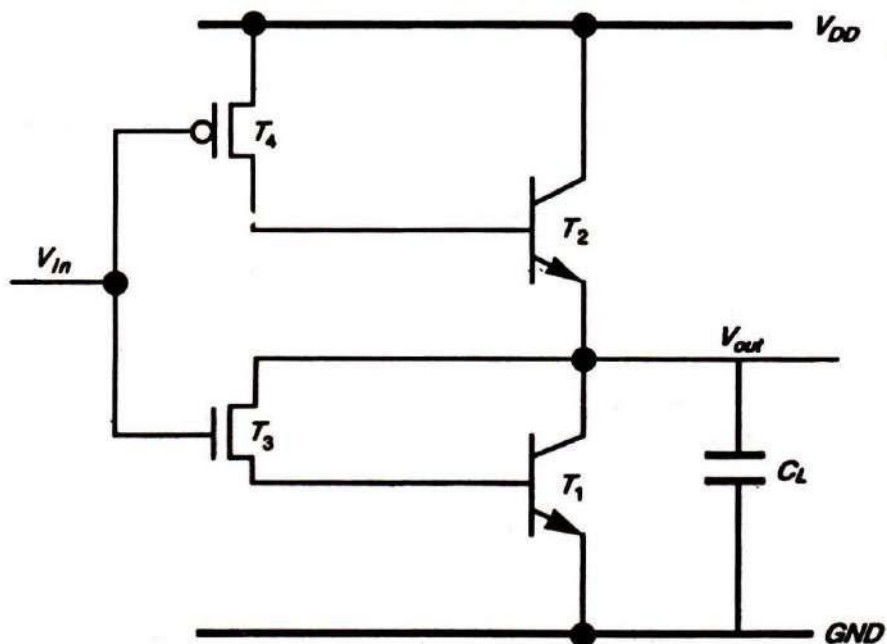


**FIGURE 2.17 A simple BiCMOS inverter.**

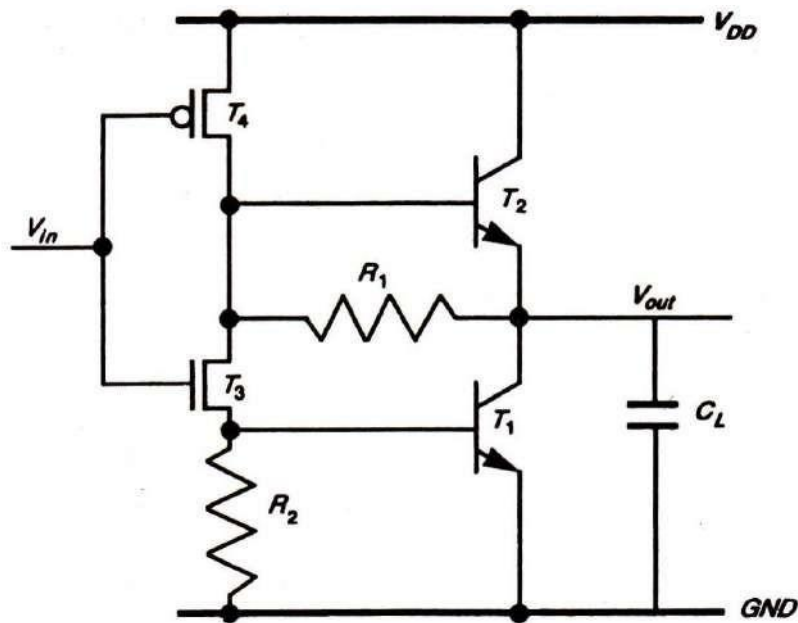
- The output logic levels will be good and will be close to the rail voltages since  $V_{CEsat}$  is quite small and  $V_{BE}$  is approximately + 0.7 volts.
- The inverter has a high input impedance.
- The inverter has a low output impedance.
- The inverter has a high current drive capability but occupies a relatively small area.
- The inverter has high noise margins.

However, owing to the presence of a DC path from  $V_{DD}$  to  $GND$  through  $T_3$  and  $T_1$  this is not a good arrangement to implement since there will be a significant static current flow whenever  $V_{in} = \text{logic 1}$ . There is also a problem in that there is no discharge path for current from the base of either bipolar transistor when it is being turned off. This will slow down the action of this circuit. An improved version of this circuit is given in Figure 2.18, in which the DC path through  $T_3$  and  $T_1$  is eliminated, but the output voltage swing is now reduced, since the output cannot fall below the base to emitter voltage  $V_{BE}$  of  $T_1$ .

An improved inverter arrangement, using resistors, is shown in Figure 2.19. In this circuit resistors provide the improved swing of output voltage when each bipolar transistor is off, and also provide discharge paths for base current during turn-off. The provision of on chip resistors of suitable value is not always convenient and may be space-consuming, so that other arrangements-such as in Figure 2.20-are used. In this circuit, the transistors  $T_5$  and  $T_6$  are arranged to turn on when  $T_2$  and  $T_1$  respectively are being turned off.



**FIGURE 2.18** An alternative BiCMOS inverter with no static current flow.



**FIGURE 2.19 An improved BiCMOS inverter with better output logic levels.**

In general, BiCMOS inverters offer many advantages where high load current sinking and sourcing is required.

#### **MOS LAYERS :**

MOS design is aimed at turning a specification into masks for processing silicon to meet the specification. We have seen that MOS circuits are formed on four basic layers-n-diffusion- diffusion, polysilicon, and metal, which are isolated from one another by thick or thin(thinox) silicon dioxide insulating layers. The thin oxide (thinox) mask region includes n-diffusion, p- diffusion, and transistor channels. Polysilicon and thinox regions interact so that a transistor is formed where they cross one another. In some processes, there may be a second metal layer and also, in some processes, a second polysilicon layer. Layers may deliberately joined together where contacts are formed. It is also clear that the basic MOS transistor properties can be modified by the use of an implant within the thinox region and this is used in nMOS circuits to produce depletion mode transistors. The BiCMOS technology is developed by including the bipolar transistors in this design process by the addition of extra layers to a CMOS process.

#### **STICK DIAGRAMS :**

A stick diagram is a diagrammatic representation of a chip layout that helps to abstract a model for design of full layout from traditional transistor schematic. Stick diagrams are used to convey the layer information with the help of a color code .For example, in the case of nMOS design, green color is used for n-diffusion, red for polysilicon, blue for metal, yellow for implant, and black for contact areas. Monochrome encoding is also used in stick diagrams to represent the layer information. The monochrome encoding chosen is shown in

figure(a).

COLOR	STICK ENCODING	LAYERS	MASK LAYOUT ENCODING	GIF LAYER
GREEN		n-diffusion (n+ active) *Thinex*		ND
RED		Polysilicon		NP
BLUE		Metal 1		NM
BLACK		Contact out		NC
GRAY	NOT APPLICABLE	Overglass		NG
nMOS ONLY YELLOW		Implant		NI
nMOS ONLY BROWN		Buried contact		NB
FEATURE	FEATURE (STICK)	FEATURE (SYMBOL)	FEATURE (MASK)	
n-type enhancement mode transistor				
Transistor length to width ratio L:W should be shown.				
n-type depletion mode transistor nMOS only				
Source, drain and gate labelling will not normally be shown.				

**Figure 1: NMOS encodings**

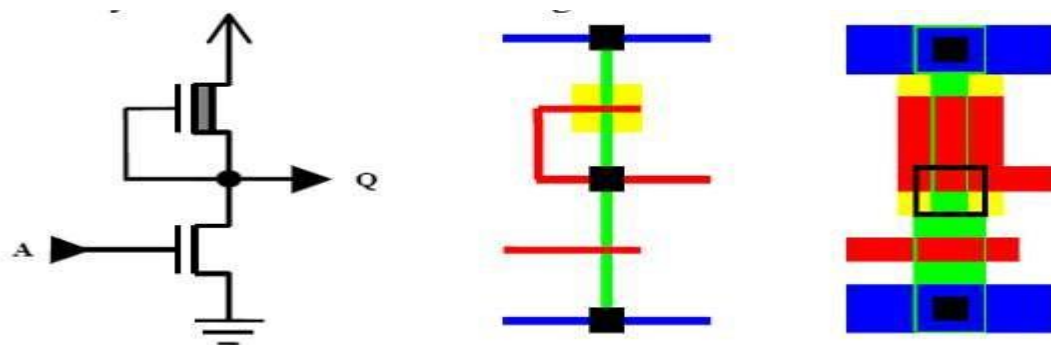


The layout of stick diagrams faithfully reflects the topology of the actual layout in silicon. The color encoding is compatible with color terminals, printers, and plotters having quite simple color palettes. Using color workstations, the mask areas are usually color filled while pen plotters produce color outlines only.

COLOR	STICK ENCODING	LAYERS	MASK LAYOUT ENCODING	CIF LAYER
GREEN	Encoding as in Color plate 1(a)	n-diffusion (n <sup>+</sup> active) Thinox*	Encoding as in Color plate 1(a)	CAA or CNA
RED		Polysilicon		CPF
BLUE		Metal 1		CMF
BLACK		Contact out		CC
GRAY		Oxideglass		COG
YELLOW (STICK)	green outline here for clarity	p-diffusion (p <sup>+</sup> active)	p <sup>+</sup> mask	CAA or CPA
YELLOW	Not shown on diagram	p <sup>+</sup> mask	either or	CPF
DARK BLUE OR PURPLE		Metal 2		CMS
BLACK		VIA		CVA
BROWN	Demarcation line p-well edge is shown as a demarcation line in stick diagrams	p-well		CPW
BLACK		V <sub>DD</sub> or V <sub>SS</sub> contact		CC
FEATURE	FEATURE (STICK)	FEATURE (SYMBOL)	FEATURE (MASK)	
n-type enhancement mode transistor (as in Color plate 1(a)) Transistor length to width ratio L:W may be shown.	Demarcation line 			
p-type enhancement mode transistor	Demarcation line 			
Note: p-type transistors are placed above and n-type below the demarcation line.				

Figure 2: CMOS encodings

Fig. Encodings for a simple metal nMOS process(color).



49

Figure 4: nMOS depletion load inverter

Stick diagram for n-MOS transistor is Shown above. The two parallel rails indicate VDD and GND

### nMOS Design Style :

To understand the design rules for nMOS design style , let us consider a single metal, single polysilicon nMOS technology.

The layout of nMOS is based on the following important features.

- n-diffusion [n-diff.] and other thin oxide regions [thinox] (green) ;
- polysilicon 1 [poly.]-since there is only one polysilicon layer here (red);
- metal 1 [metal]-since we use only one metal layer here (blue);
- implant (yellow);
- contacts (black or brown [buried]).

A transistor is formed wherever poly. crosses n-diff. (red over green) and all diffusion wires (interconnections) are n-type (green).

When starting a layout, the first step normally taken is to draw the metal (blue) VDD and GND rails in parallel allowing enough space between them for the other circuit elements which will be required. Next, thinox (green) paths may be drawn between the rails for inverters and inverter- based logic as shown in Fig. below. Inverters and inverter-based logic comprise a pull-up structure, usually a depletion mode transistor, connected from the output point to VDD and a pull- down structure of enhancement mode transistors suitably interconnected between the output point and GND. This is illustrated in the Fig.(b). remembering that poly. (red) crosses thinox (green) wherever transistors are required. One should consider the implants (yellow) for depletion mode transistors and also consider the length to width (L : W) ratio for each transistor. These ratios are important particularly in nMOS and nMOS- like circuits.

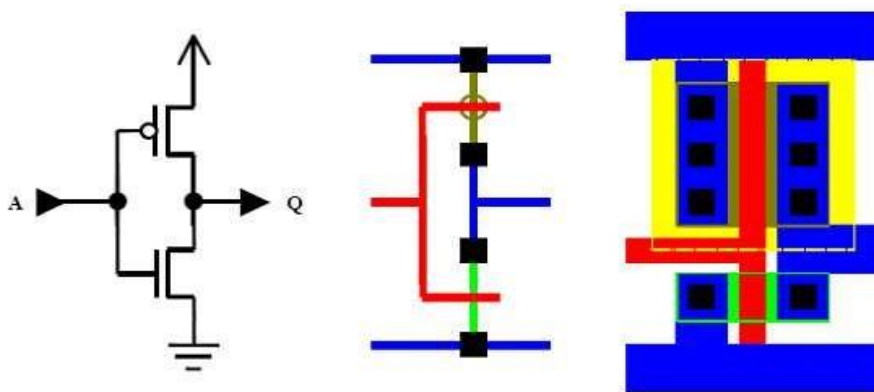


Figure 5: CMOS inverter

Figure 5 shows the schematic, stick diagram and corresponding layout of CMOS inverter



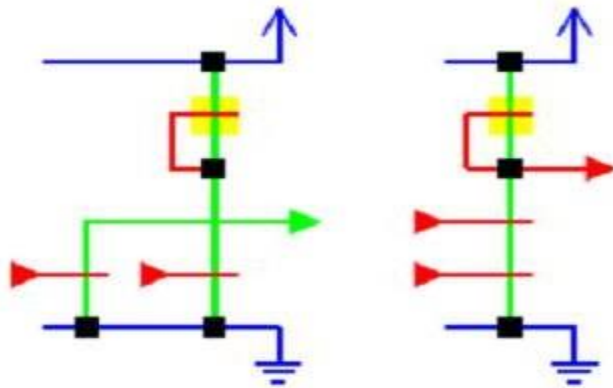


Figure 6: nMOS depletion load NAND and NOR stick diagram

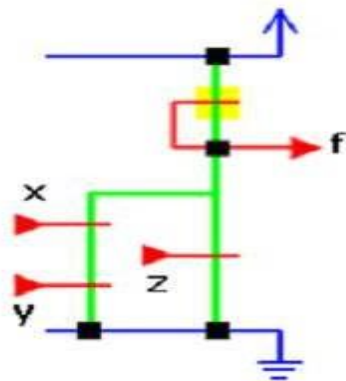
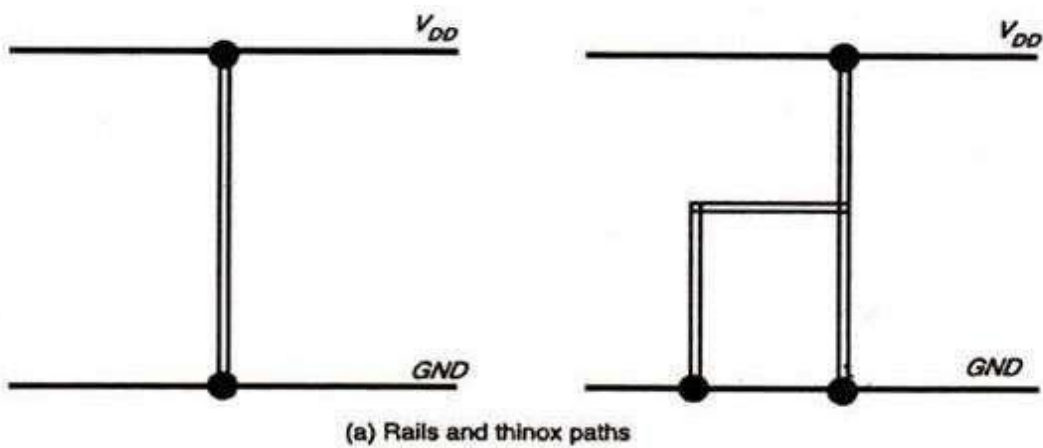


Figure 7: stick diagram of a given function f.

Figure 7 shows the stick diagram nMOS implementation of the function  $f = [(xy) + z]'$



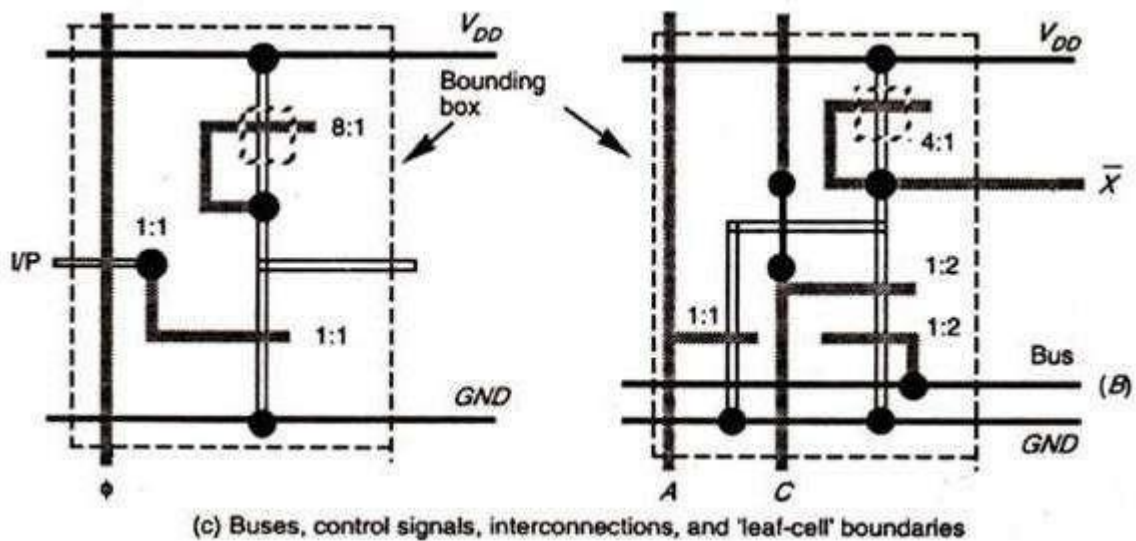
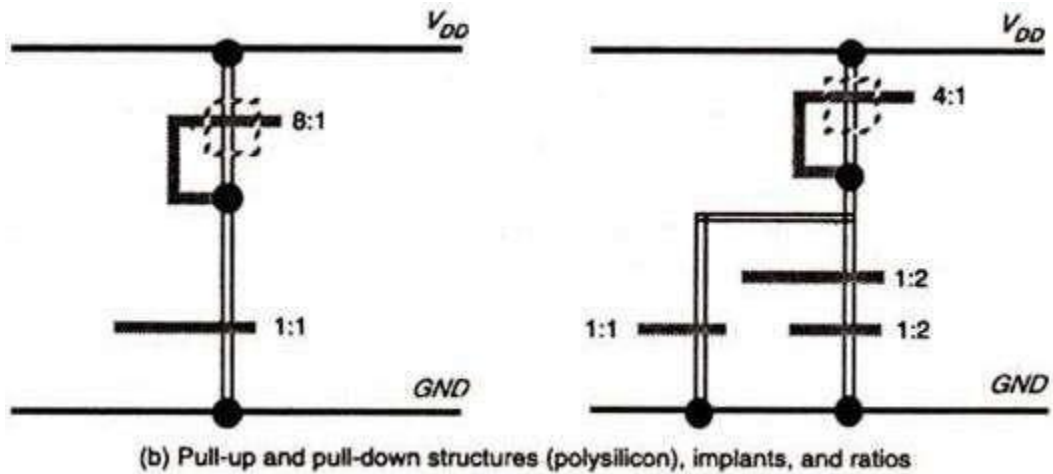
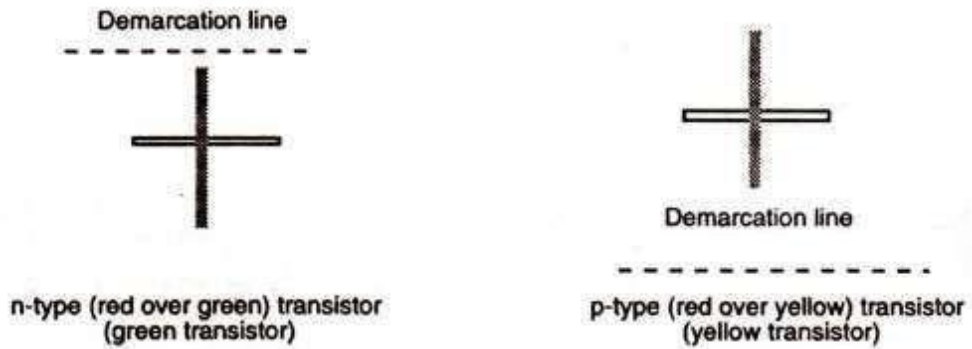


Fig . nMOS stick layout design style

### CMOS Design Style :

The CMOS design rules are almost similar and extensions of n-MOS design rules except the implant (yellow) and the buried contact (brown). In CMOS design Yellow is used to identify p- transistors and wires, as depletion mode devices are not utilized. The two types of transistors 'n' and 'p', are separated by the demarcation line (representing the p-well boundary) above which all p-type devices are placed (transistors and wires (yellow)). The n-devices (green) are consequently placed below the demarcation line and are thus located in the p-well as shown in the diagram below.

Diffusion paths must not cross the demarcation line and n-diffusion and p-diffusion wires must not join. The 'n' and 'p' features are normally joined by metal where a connection is needed. Their geometry will appear wh5e2n the stick diagram is translated to a mask layout. However, one must not forget to place crosses on VDD and Vss rails to represent the substrate and p-well connection respectively.



The design style is explained by taking the example the design of a single bit shift register. The design begins with the drawing of the VDD and Vss rails in parallel and in metal and the creation of an (imaginary) demarcation line in-between, as shown in Fig. below. The n-transistors are then placed below this line and thus close to Vss, while p-transistors are placed above the line and below VDD. In both cases, the transistors are conveniently placed with their diffusion paths parallel to the rails (horizontal in the diagram) as shown in Fig. (b). A similar approach can be taken with transistors in symbolic form.

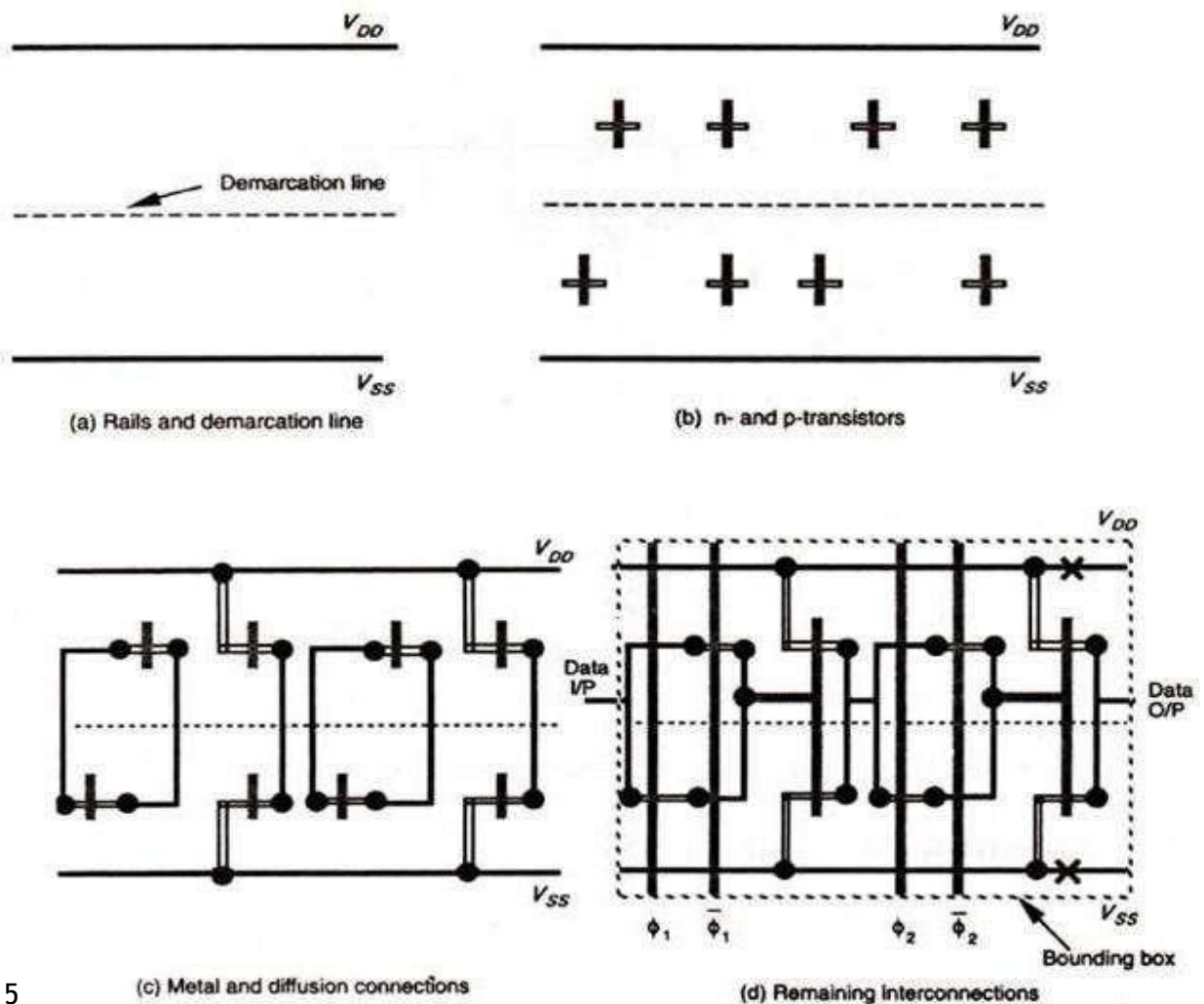


Fig. CMOS stick layout design style (a,b,c,d)

The n- along with the p-transistors are interconnected to the rails using the metal and connect as shown in Fig.(d). It must be remembered that only metal and poly-silicon can cross the demarcation line but with that restriction, wires can run-in diffusion also. Finally, the remaining interconnections are made as appropriate and the control signals and data inputs are added as shown in the Fig.(d).

### **Design Rules and Layout :**

The design rules are formed to translate the circuit design concepts , (usually in stick diagram or symbolic form) into actual geometry in silicon. The design rules are the effective interface between the circuit/system designer and the fabrication engineer. The design rules also help to provide a reliable compromise between the circuit/system designer and the fabrication engineer. In general the circuit designers expect smaller layouts for improved performance and decreased silicon area. On the other hand, the process engineer like those design rules that result in a controllable and reproducible process. In fact there is a need of compromise for a competitive circuit to be produced at a reasonable cost.

One of the important factors associated with design rules is the achievable definition of the process line. For example, it is found that if a 10: 1 wafer stepper is used instead of a 1: 1 projection mask aligner; the level-to-level registration will be closer. Design rules can be affected by the maturity of the process line. For example, if the process is mature, then one can be assured of the process line capability, allowing tighter designs with fewer constraints on the designer.

The simple and well known design rules that are widely used in the design of multiproject chips are 'lambda ( $\lambda$ )-based' design rules developed by Mead and Conway .

### **Lambda-based Design Rules :**

In this Lambda –base design rules all paths in all layers will be dimensioned in  $\lambda$  units and subsequently  $\lambda$  can be allocated an appropriate value compatible with the feature size of the fabrication process. These design rules are such that, if correctly obeyed, the mask layouts will produce working circuits for a range of values allocated to  $\lambda$ . For example,  $\lambda$  can be allocated a value of  $1.0\mu\text{m}$  so that minimum feature size on chip will be  $2\mu\text{m}$  ( $2\lambda$ ). Design rules, also, specify line widths, separations, and extensions in terms of  $\lambda$ . Design rules can be conveniently set out in diagrammatic form as shown in Fig.(a) for wires , and the Fig.(b) for extensions and separations associated with transistor layouts.

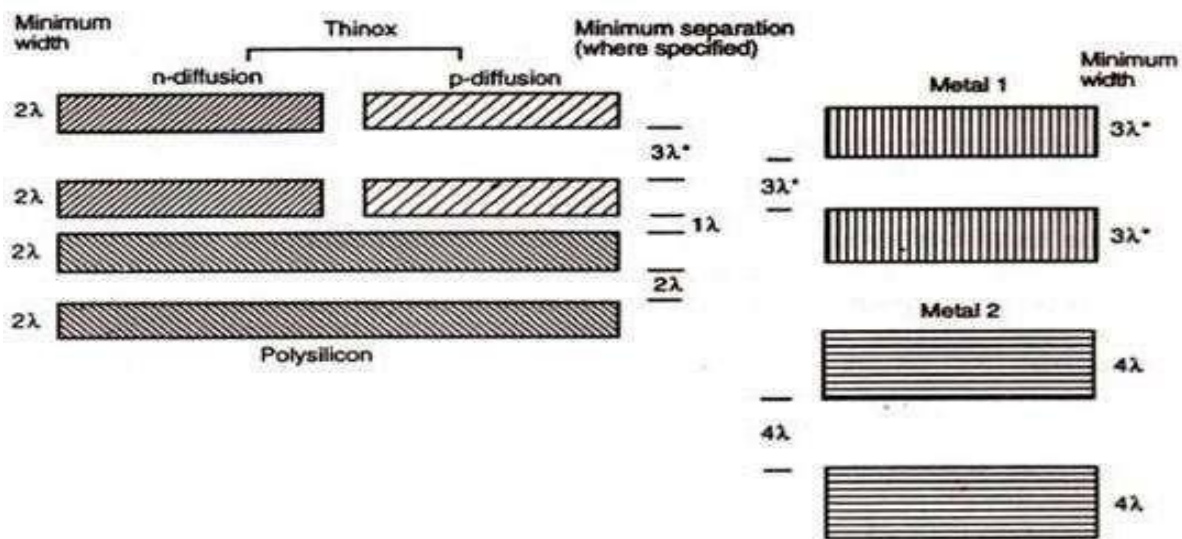


Fig.(a). Design rules for wires (n-MOS and p-MOS)

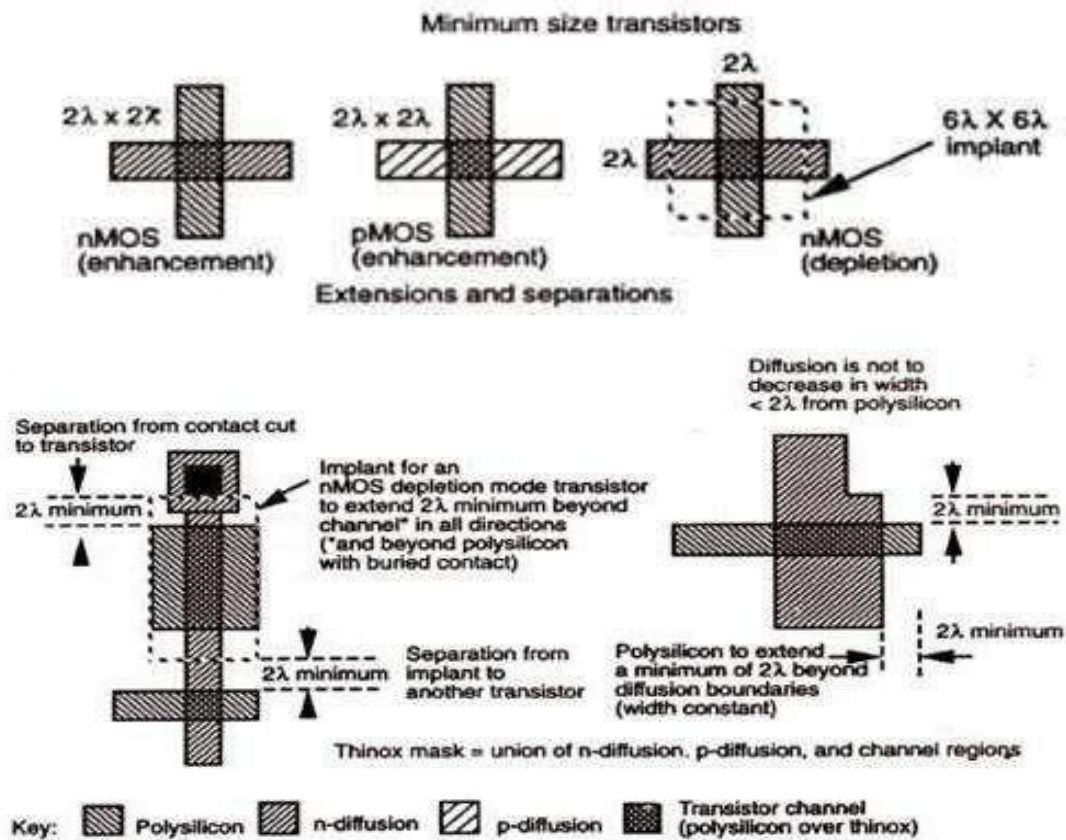


Fig.(b). Transistor design rules (n-MOS, p-MOS and c-MOS)

### Contact Cuts :

While making contacts between poly-silicon and diffusion in nMOS circuits it should be remembered that there are three possible approaches--poly. to metal then metal to diff., or a

buried contact poly. to diff. , or a butting contact (poly. to diff. using metal). Among the three the latter two, the buried contact is the most widely used, because of advantage in space and a reliable contact. At one time butting contacts were widely used , but now a days they are superseded by buried contacts.

In CMOS designs, poly. to diff. contacts are always made via metal. A simple process is followed for making connections between metal and either of the other two layers (as in Fig.a), The  $2\lambda \times 2\lambda$ . contact cut indicates an area in which the oxide is to be removed down to the underlying polysilicon or diffusion surface. When deposition of the metal layer takes place the metal is deposited through the contact cut areas onto the underlying area so that contact is made between the layers.

The process is more complex for connecting diffusion to poly-silicon using the butting contact approach (Fig.b), In effect, a  $2\lambda \times 2\lambda$  contact cut is made down to each of the layers to be joined. The layers are butted together in such a way that these two contact cuts become contiguous. Since the poly-silicon and diffusion outlines overlap and thin oxide under poly- silicon acts as a mask in the diffusion process, the poly-silicon and diffusion layers are also butted together. The contact between the two butting layers is then made by a metal overlay as shown in the Fig.

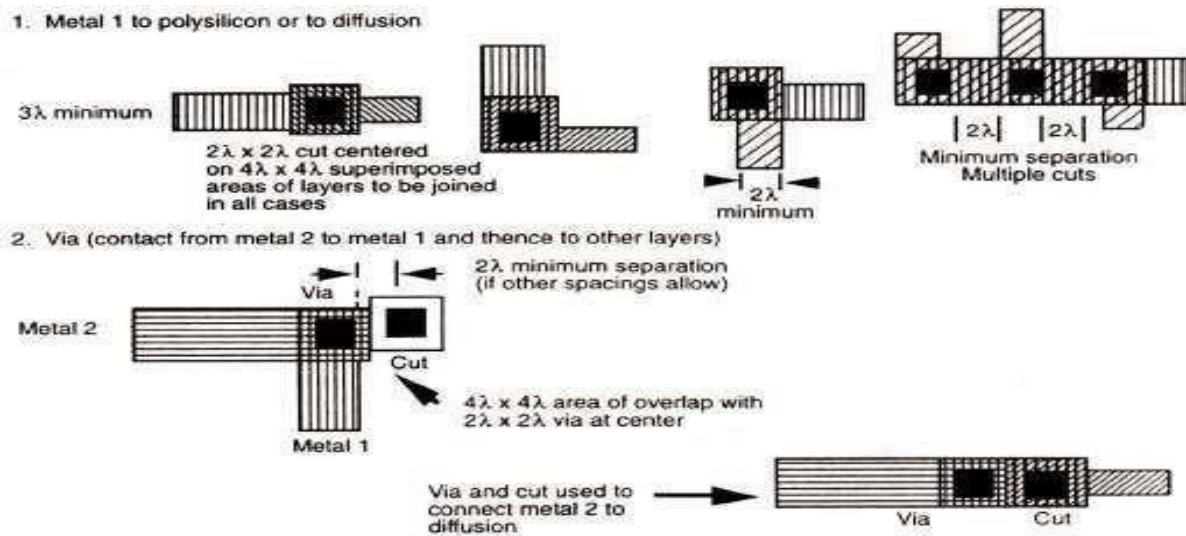


Fig.(a) . n-MOS & C-MOS Contacts



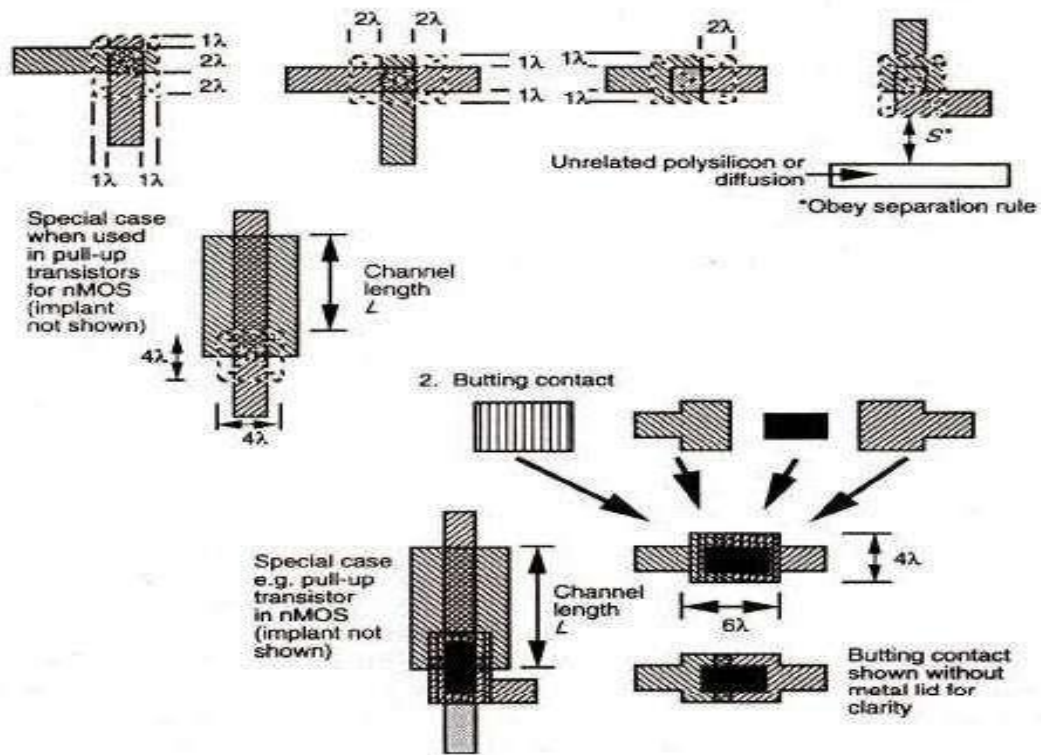


Fig.(b). Contacts poly-silicon to diffusion

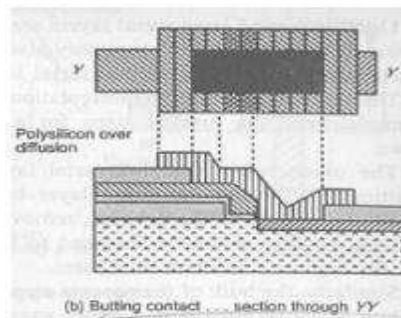


Figure 15: Butting contact

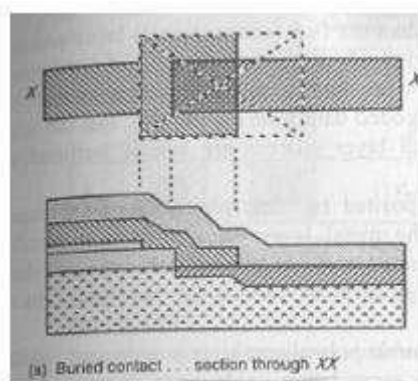


Figure 14: Buried contact

In buried contact basically, layers are joined over a  $2\lambda \times 2\lambda$  area with the buried contact cut extending by  $1\lambda$ , in all directions around the contact area except that the contact cut extension is increased to  $2\lambda$  in diffusion paths leaving the contact area. This helps to avoid the formation of unwanted transistors. So, this buried contact approach is simpler when compared to others. The, poly-silicon is deposited directly on the underlying crystalline wafer. When diffusion takes place, impurities will diffuse into the poly-silicon as well as into the diffusion region within the contact area. Thus a satisfactory connection between poly-silicon and diffusion is ensured. Buried contacts can be smaller in area than their butting contact counterparts and, since they use no

metal layer, they are subject to fewer design rule restrictions in a layout.

### **Double metal MOS process rules :**

In the MOS design rules a powerful design process is achieved by adding a second metal layer. This gives a much greater degree of freedom, in distributing global VDD and Vss(GND) rails in a system. From the overall chip inter-connection aspect, the second metal layer in particular is important and, although the use of such a layer is readily envisaged, its disposition relative to its connection. to other layers using metal1 to metal 2 contacts, called **vias** , can be readily established .

Usually, second level metal layers are coarser than the first (conventional) layer and the isolation layer between the layers may also be of relatively greater thickness. To distinguish contacts between first and second metal layers, they are known as **vias** rather than contact cuts. The second metal layer representation is color coded dark blue (or purple).

The important process steps for a two-metal layer process are given below.

The oxide below the first metal layer is deposited by atmospheric chemical vapor deposition (CVD) and the oxide layer between the metal layers is applied in a similar manner. Depending on the process, removal of selected areas of the oxide is accomplished by plasma etching, which is designed to have a high level of vertical ion bombardment to allow for high and uniform etch rates. Similarly, the bulk of the process steps for a double polysilicon layer process are similar in nature to those already described, except that a second thin oxide layer is grown after depositing and patterning the first polysilicon layer (Poly.1) to isolate it from the now to be deposited second poly. layer (Poly.2). The presence of a second poly. layer gives greater flexibility in interconnections and also allows Poly.2 transistors to be formed by intersecting Poly. 2 and diffusion.

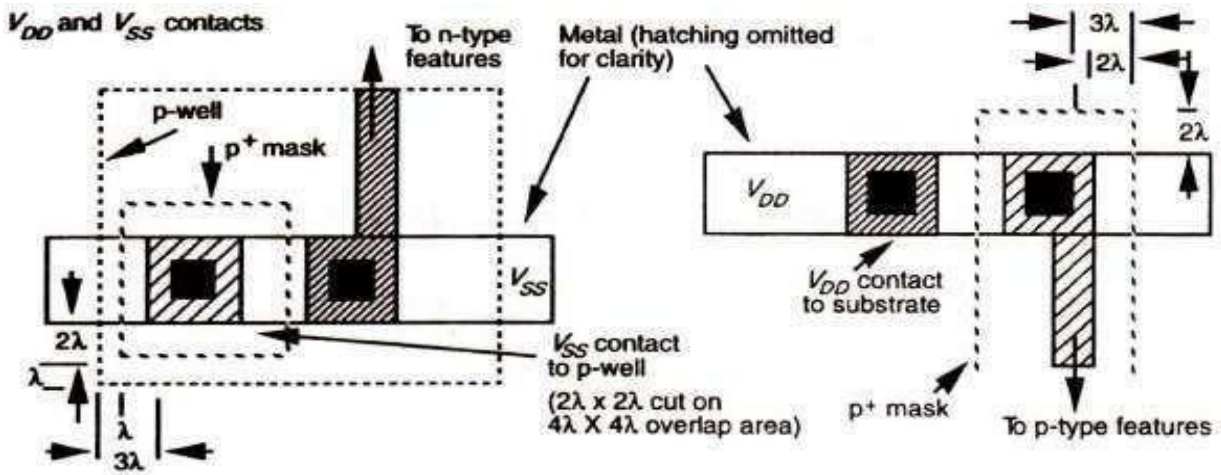
The important features of double metal process are summarized as follows :

- Use the second level metal for the global distribution of power buses, that is, VDD and GND ( Vss), and for clock lines.
- Use the first level metal for local distribution of power and for signal lines.
- Lay out the two metal layers so that the conductors are mutually orthogonal wherever possible.

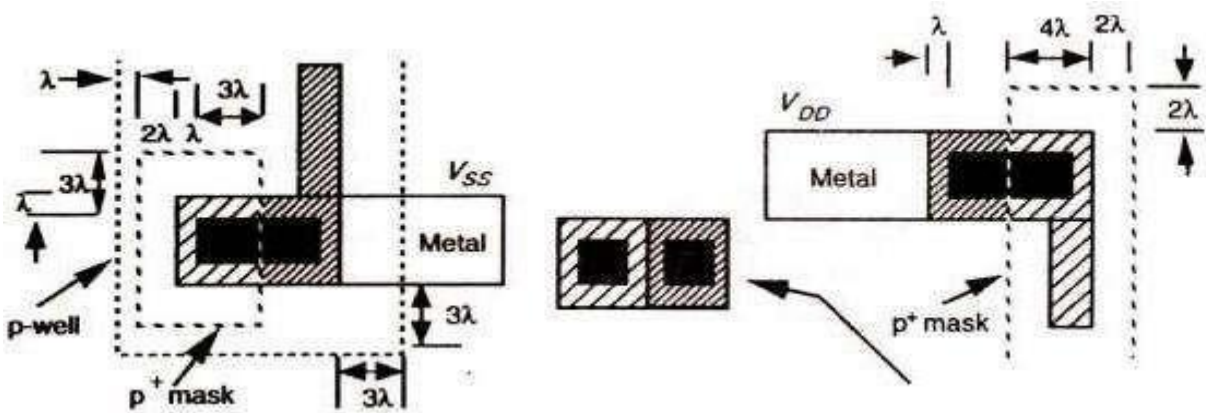
### **CMOS Lambda-based Design Rules:**

The CMOS fabrication process is more complex than nMOS fabrication . In a CMOS process, there are nearly 100 actual set of industrial design rules . The additional rules are concerned with those features unique to p-well CMOS, such as the p-well and p+ mask and the special 'substrate' contacts. The p-well rules are shown in the diagram below.





In the diagram above each of the arrangements can be merged into single split contacts.



From the above diagram it is also clear that split contacts may also be made with separate cuts.

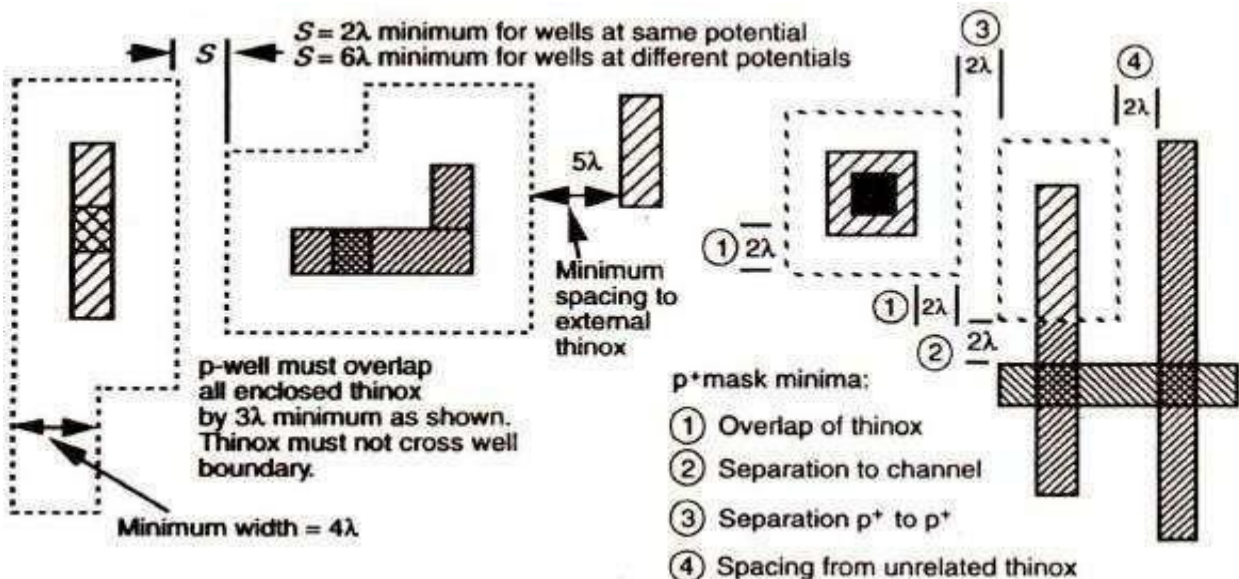


Fig. Particular rules for p-well CMOS Process.

Th5e9CMOS rules are designed based on the extensions of the Mead and Conway concepts and also by excluding the butting and buried contacts the new rules for CMOS design are formed. These rules for CMOS design are implemented in the above diagrams.

**General Observations on the Design Rules :**

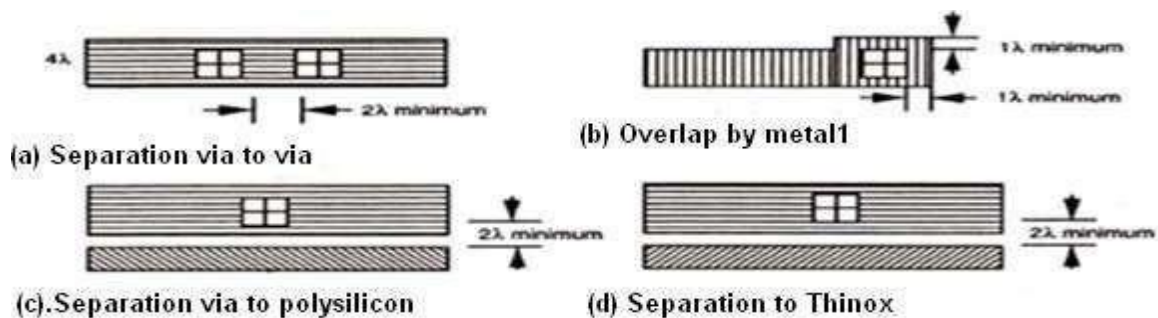
The microscopic dimensions of Silicon circuits always cause some problems in the design process. The major

problem is presented by possible deviation in line widths and in interlayer registration. If the line widths are too small, it is possible for lines to be discontinuous in places. If separate paths in a layer are placed too close together, it is possible that they will merge in places or interfere with each other.

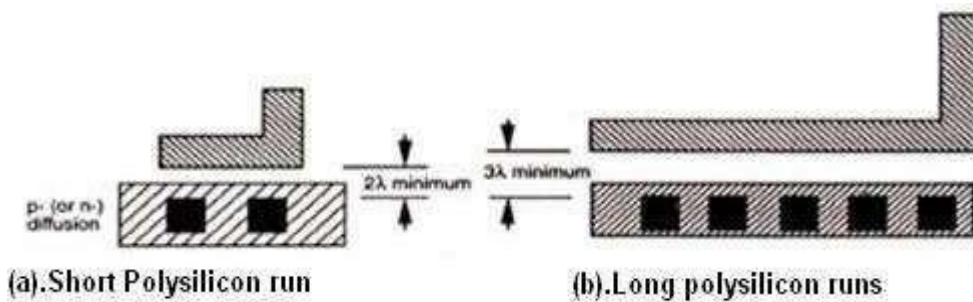
For the lambda-based rules, the design rules are formulated in terms of a length unit  $\lambda$  which is related to the resolution of the process  $\lambda$  may be viewed as a limit on the width deviation of a feature from its ideal 'as drawn' size and also as a bound on the maximum misalignment of any one mask. In the worst case, these effects may combine to cause the relative position of feature edges on different mask levels to deviate by as much as  $2\lambda$  in their interrelationship. Inevitably, a consequence of using the lambda-based concept is that every dimension must be rounded up to whole  $\lambda$  values and this leads to layouts which do not fully exploit the capabilities of the process.

Similar concepts underlie the establishment of 'micron-based' rule sets, but actual dimensions are given so that full advantage can be taken of the fabrication line capabilities and tighter layouts result.

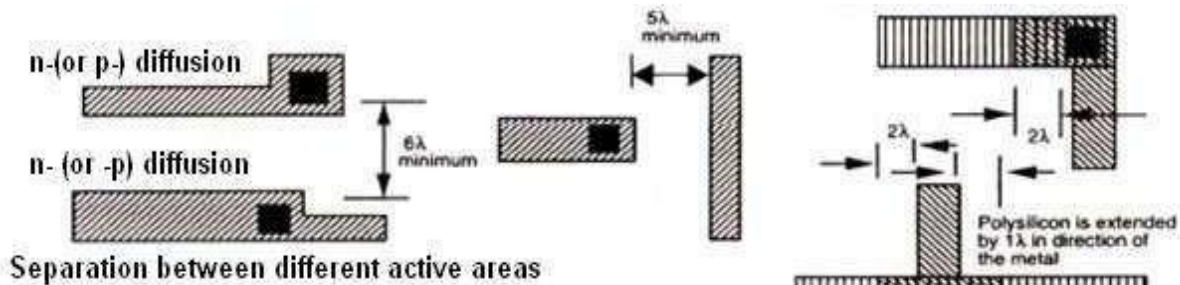
Layout rules, therefore, provide strict guidelines for preparing the geometric layouts which will be used to configure the actual masks used during fabrication and can be regarded as the main communication link between circuit/systems designers and the process engineers engaged in manufacture. The goal of any set of design rules should give optimize yield while keeping the geometry as small as possible without compromising the reliability of the finished circuit. On the questions of yield and reliability, even the conservative nature of the lambda based rules can stand reevaluation when these two factors are of paramount importance. In particular, the rules associated with contacts can be improved upon in the light of experience. Fig.(a) sets out aspects that may be observed for high yield and in high reliability situations. In our proposed scheme of events in creating stick layouts for CMOS, it is assumed that poly. and metal can both freely cross well boundaries and this is indeed the case, but we should be careful to try to exclude poly. from areas which lie within p+ mask areas where possible. The reason for this is that the resistance of the poly. layer is reduced in current processes by  $n^-$  type doping. Clearly the p+ doping which takes place inside the p+ mask will also dope the poly. which is already in place when the p+ doping step takes place. This results in an increase in the  $n^-$  doping poly. resistance which may be significant in certain parts of a system.



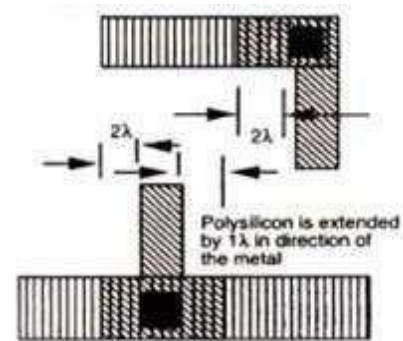
1.Aspects related to vias (Double metal processes)



2. Polysilicon wires separation from cuts



3. Diffusion wires separation from cuts



4. Increase in polysilicon overlap to reduce metal migration effect

The  $3\lambda$  metal width rule is a conservative one but is implemented to allow for the fact that the metal layer is deposited after the others and on top of them and several layers of silicon dioxide, so that the surface on which it sits is quite 'mountainous'. The metal layer is also light-reflective and these factors combine to result in poor edge definition. In double metal the second layer of metal has an even more uneven terrain on which to be deposited and patterned. Hence metal 2 is often wider than metal 1.

Metal to metal separation is also large and is brought about mainly by difficulties in defining metal edges accurately during masking operations on the highly reflective metal. All diffusion processes are such that lateral diffusion occurs as well as impurity penetration from the surface. Hence the separation rules for diffusion allow for this and relatively large separations are specified. This is particularly the case for the p-well diffusions which are deep diffusions and thus have considerable lateral spread. Transitions from thin gate oxide to thick field oxide in the oxidation process also use up space and this is another reason why the lambda-based rules require a minimum separation between thinox regions of  $3\lambda$ . In effect, this implies that the minimum feature size for thick oxide is  $3\lambda$ . The simplicity of the lambda-based rules makes this approach to design an appropriate one for the novice chip designer and also, perhaps, for those applications in which we are not trying to achieve the absolute minimum area and the absolute maximum performance. Because lambda-based rules try 'to be all things to all people', they do suffer from least common denominator effects and from the upward rounding of all process line dimension parameters into integer values of lambda.

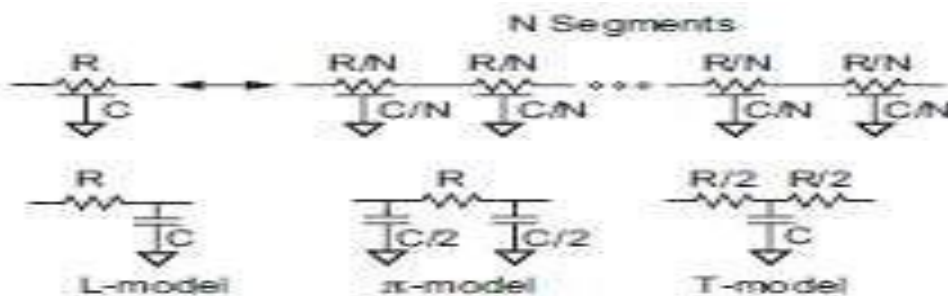
The performance of any fabrication line in this respect clearly comes down to a matter of tolerances and definitions in terms of microns (or some other suitable unit of length). Thus, expanded sets of rules often referred to as micron-based rules are available to the more experienced designer to allow for the use of the full capability of any process. Also, many processes offer additional layers, which again adds to the possibilities presented to the designer. In order to properly represent these important aspects, the next section introduces Orbit Semiconductor's  $2\mu\text{m}$  feature size double metal, double poly, n-well CMOS rules which also offer a BiCMOS capability.

## UNIT - 3

### VLSI Interconnects

The wires linking transistors together are called *interconnect* and play a major role in the performance of modern systems. In the early days of VLSI, transistors were relatively slow. Wires were wide and thick and thus had low resistance. In modern VLSI Processes, transistors switch much faster. Meanwhile, wires have become narrower, driving up their resistance to the point, that in many signal paths, the wire RC delay exceeds gate delay.

A wire is a distributed circuit with a resistance and capacitance per unit length. Its behavior can be approximated with a number of lumped elements. Three standard approximation are the L-model,  $\pi$ -model, and T-model, so-named because of their shapes. Figure shows how a distributed RC circuit is equivalent to  $N$  distributed RC segments of proportionally smaller resistance and capacitance, and how these segments can be modeled with lumped elements.



The L-model is a poor choice because a large number of segments are required for accurate results. The  $\pi$ -model is much better; three segments are sufficient to give results accurate to 3%.

Following are the effects of interconnects

**Delay:** Interconnect increases circuit delay for two reasons. First, the wire capacitance adds loading to each gate. Second, long wires have significant resistance that contributes distributed

RC delay or *flight time*. wire delay grows quadratically with length. Using thicker and wider wires, lower-resistance metals such as copper, and lower-dielectric constant insulators helps, but long wires nevertheless often have unacceptable delay. Repeaters can be used to break a long wire into multiple segments such that the overall delay becomes a linear function of length. Polysilicon and diffusion wires have high resistance, even if silicided. Diffusion also has very high capacitance.

**Energy:** The switching energy of a wire is set by its capacitance. Long wires have significant capacitance and thus require substantial amounts of energy to switch.

**crosstalk :** wires have capacitance to their adjacent neighbors as well as to ground. When wire  $A$  switches, it tends to bring its neighbor  $B$  along with it on account of capacitive coupling, also called *crosstalk*. If  $B$  is supposed to switch simultaneously, this may increase or decrease the switching delay. If  $B$  is not supposed to switch, crosstalk causes noise on  $B$ . We will see that the impact of crosstalk depends on the ratio of  $C_{adj}$  to the total capacitance. Note that the load capacitance is included in the total, so for short wires and large loads, the load capacitance dominates and crosstalk is unimportant. Conversely, crosstalk is very important for long wires.

### Reliability issues in CMOS VLSI

Reliability has always been a concern for integrated circuit designers due to the small size of the devices and the natural variations that occur in manufacturing processes. Modern design-for-manufacturability and design-for yield techniques are based on a fundamental understanding of the failure mechanisms of integrated circuits.



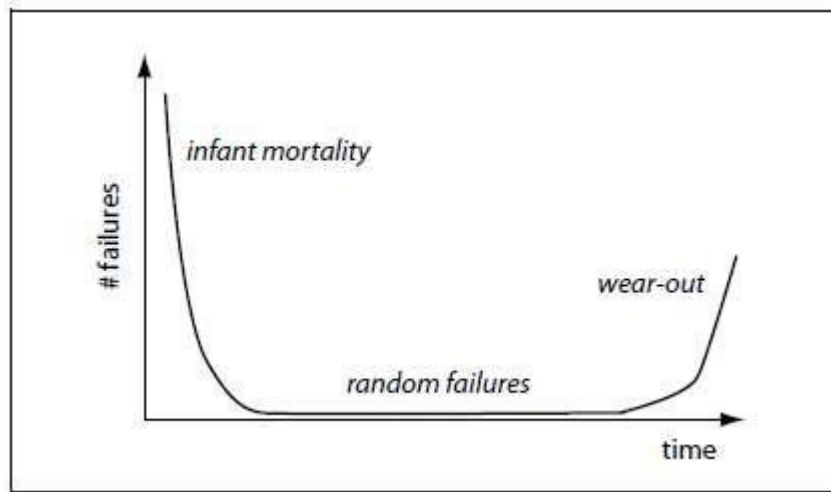


Fig :

Traditional VLSI manufacturing processes yielded chips that were remarkably reliable over a long period. Figure illustrates the general form of failures vs. time for traditional processes. This curve is known as the **bathtub curve** because of its shape—many chips failed in the first few hours of operation, then few failures occurred for years, and finally chips started to fail at a higher rate as they wore out.

Early chip failures are known as **infant mortality**; it may be caused by a variety of fabrication flaws that create marginal structures such as thin wires or malformed transistors. One commonly-used model for chip reliability is an exponential probability for failure .

This model assumes that the failure rate starts high and rapidly decreases. Manufacturers generally **burn in** their chips for some period by running them with power so that marginal chips will fail at the factory rather than in the hands of the customer.

The bathtub curve concerns itself with **hard failures**, meaning permanent functional failures of the chip. **Transient failures**, which cause errors on certain outputs, were not a major concern for quite some time in digital circuits, although they have long been a concern in memories. Transient failures can come from several causes, including bit flips and timing errors.

The most common metric for failure rates is mean time to failure (MTTF). This metric defines the mean time to the next occurrence of a given failure mechanism. Based on MTTF, we can determine other interesting metrics, such as **lifetime**.

Semiconductor manufacturing processes are complex and build many different structures. As a result, several different important failure mechanisms have been identified for traditional VLSI processes

- **diffusion and junctions** Crystal defects, impurity precipitation, mask misalignment, surface contamination.
- **oxides** Mobile ions, pinholes, interface states, hot carriers, time dependent dielectric breakdown.
- **metallization** Scratches and voids, mechanical damage, non-ohmic contacts, step coverage, weak adhesion, improper thickness, corrosion, electromigration, stress migration.
- **passivation** Pinholes and cracks, thickness variations, contamination, surface inversion.

Several mechanisms stand out: **time-dependent dielectric breakdown (TDDB)**, **hot carriers**, **negative bias temperature instability (NBTI)**, electromigration, **stress migration** and **soft errors**. Some of these failure mechanisms target transistors while others come from interconnect.

**TDDB** Time-dependent dielectric breakdown occurs because the electric fields across gate oxides induce stresses that damage the oxide. Small transistors require very thin oxides that are more susceptible to this form of damage. The traditional model for TDDB failure rates is known as Black's equation

$$MTTF = A \times 10^{BE} e^{E_a/kT}$$

In this formula,  $A$  is a constant,  $E_a$  is the activation energy in eV,  $E$  is the electric field intensity in MV/cm,  $A$  is the electric field intensity coefficient in cm/MV,  $k$  is Boltzmann's constant, and  $T$  is the absolute temperature. A hot carrier is a carrier that gains enough energy to jump from the silicon substrate into the gate oxide. As these hot carriers accumulate, they create a space charge in the oxide that affects the transistor's threshold voltage and other parameters. Several factors, such as power supply voltage, channel length, and ambient temperature can affect the rate at which hot carriers are produced.

Negative bias temperature instability is particular to pMOS devices. It refers to shifts in  $V_{th}/g_m$  due to stress that introduces interface states and space charge. Interestingly, this degradation can be reversed by applying a reverse bias to the transistor. As a result, it is not a significant failure mechanism for p-type transistors whose bias voltages change from forward to reverse regularly but is very important for DC-biased transistors.

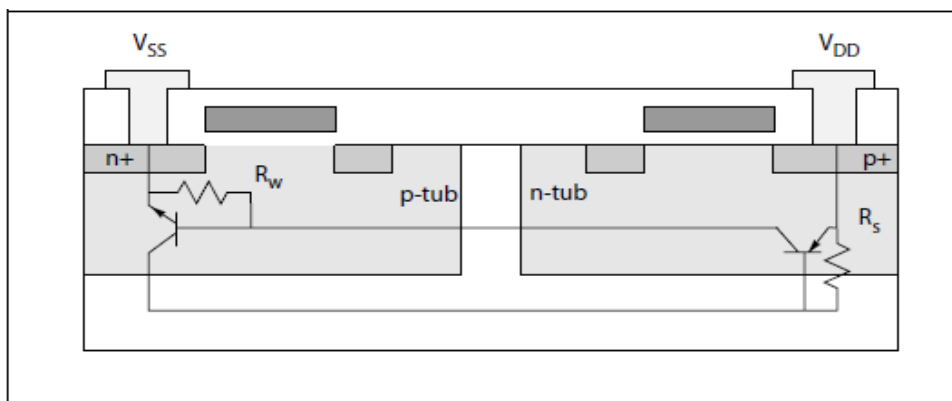
Electromigration is a degenerative failure mechanism for wires that we touched upon before. Aluminum wiring includes grains that carry many defects; these grain boundaries are the most important source of electromigration problems.

*stress migration* Stress migration is caused by mechanical stress and can occur even when no current flows through the wire. These stresses are caused by the different thermal expansion coefficients of the wires and the materials in which they reside. Failures can be caused by long-term exposure to moderate temperatures in the range. Failures can also occur due to short-term stresses at very high temperatures.

*soft errors* Soft errors cause memory cells to change state. Soft errors can be caused by alpha particles that generate excess carriers as they travel through the substrate. The materials used in packages include small amounts of uranium and thorium, which is still enough to cause noticeable rates of soft errors.

### Latching:

Using many tub ties in each tub makes a low-resistance connection between the tub and the power supply. If that connection has higher resistance, parasitic bipolar transistors can cause the chip to **latch-up**, inhibiting normal chip operation.



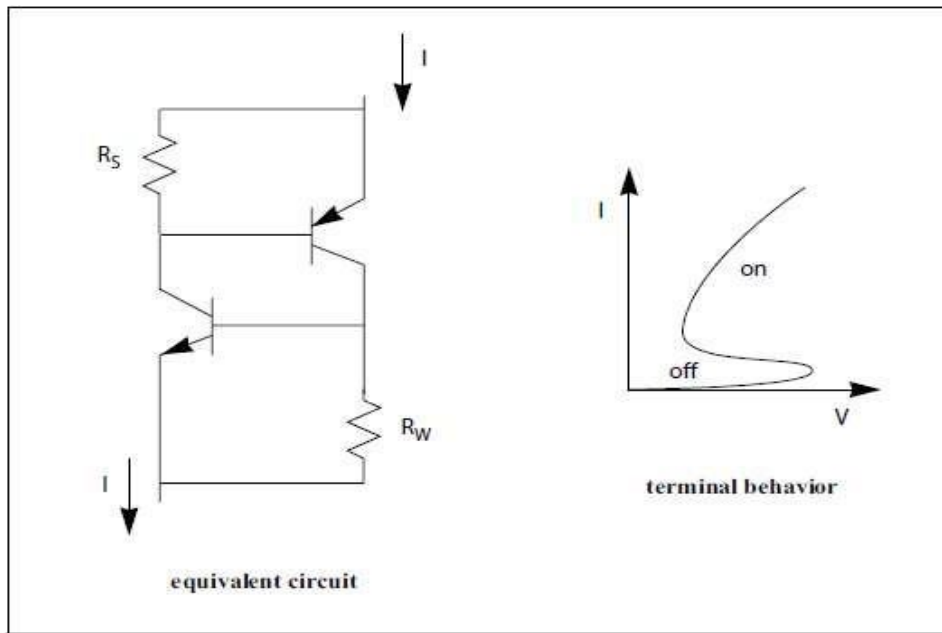
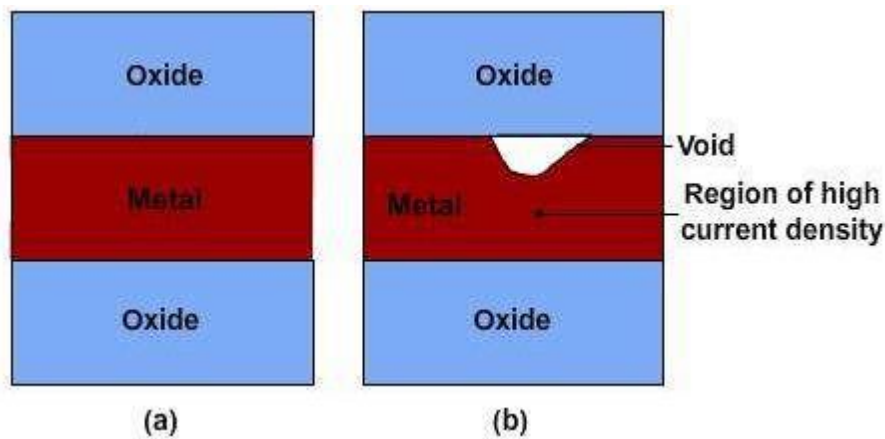


Figure 2-11 shows a chip cross-section which might be found in an inverter or other logic gate. The MOS transistor and tub structures form parasitic bipolar transistors: npn transistors are formed in the p-tub and pnp transistors in the n-tub. Since the tub regions are not physically isolated, current can flow between these parasitic transistors along the paths shown as wires. Since the tubs are not perfect conductors, some of these paths include parasitic resistors; the key resistances are those between the power supply terminals and the bases of the two bipolar transistors.

The parasitic bipolar transistors and resistors create a parasitic **silicon controlled rectifier**, or SCR. The schematic for the SCR and its behavior are shown in Figure 2-12. The SCR has two modes of operation. When both bipolar transistors are off, the SCR conducts essentially no current between its two terminals. As the voltage across the SCR is raised, it may eventually turn on and conducts a great deal of current with very little voltage drop. The SCR formed by the n- and p-tubs, when turned on, forms a high-current, low-voltage connection between VDD and VSS. Its effect is to short together the power supply terminals. When the SCR is on, the current flowing through it floods the tubs and prevents the transistors from operating properly. In some cases, the chip can be restored to normal operation by disconnecting and then reconnecting the power supply; in other cases the high currents cause permanent damage to the chip.

The switching point of the SCR is controlled by the values of the two power supply resistances  $R_S$  and  $R_W$ . Each bipolar transistor in the SCR turns on when its base-to-emitter voltage reaches 0.7 V; that voltage is controlled by the voltage across the two resistors. The higher the resistance, the less stray current through the tub is required to cause a voltage drop across the parasitic resistance that can turn on the associated transistor. Adding more tub ties reduces the values of  $R_S$  and  $R_W$ . The maximum distance between tub ties is chosen to ensure that the chip will not latch-up during normal operation.



**Fig 9.4** Schematic of electromigration causing failure. (a) Early stages. Wire occupies the entire space and the current goes through the metal. Current density is moderate (b) Void formation leads to decreased area available for conduction and increased current density. Acceleration of failure.

In the beginning, the metal would fill the space between the insulators and the current density would be at a certain level (Fig 9.4a). If sufficient atoms move due to electromigration, then a small void will form. Now the all current has to go through the metal and hence near the void region, the current density will increase.

The electrical resistance of the metal line is also higher now, because the electrical resistance is inversely proportional to the cross sectional area. This results in larger heat release and hence higher local temperature. At higher temperatures, the metal atom diffusivity is higher, which makes it easier to ‘push’ the atoms. The increased current density and higher temperature accelerates the formation of voids and finally results in the circuit failure.

The extent of electromigration movement depends on the nature of the material. For example, materials such as copper, tungsten and gold have good electro migration resistance. We must note the difference between electrical resistance and electro migration resistance clearly. Electrical resistance indicates the resistance to movements of electrons. We want good electrical conductors (i.e. low electrical resistance). At the same time, we want materials which have high electro migration resistance.

If the electro migration resistance is poor, then after many hours of operation, the resistance of one or few wires will increase dramatically. It can lead to the failure of the chip. These types of failures, where the chip originally functions well and after few months of operation fails, are called “reliability issues”. This means that the chip appears to be good during testing in the fab, but it is not reliable and after sometime it can fail.

Aluminum has low electromigration resistance. To enhance its electromigration resistance, usually a small amount of Cu is added during the deposition of Al. Similarly, a small amount of Si is also added to Al. This is because of the tendency to dissolve into aluminum and the silicon in the insulator (silicon dioxide) may diffuse into aluminum. If the aluminum is already saturated with silicon, then further dissolution of Si in Al will not be possible. Thus, both Si and Cu are added in small quantities (1% for example) in the aluminum lines to minimize silicon dissolution and electromigration, respectively. Tungsten has a good electromigration resistance and hence the vias or contacts made of tungsten are quite immune to this issue.

The average time for failure, which is called MTF or mean time to fail, depends on the current density and the



temperature. Usually if the wires are made with large grains, that is large crystals then it will not be easy to break the wire. If the size and the arrangement (that is the orientation of the grains) are good then the electro migration resistance will be high and the chip will not fail easily.

In order to test whether a material or chip will function for a long time without failure, frequently the chip is tested at high temperature. This is called high temperature operation test or HTOT. Sometimes the chip is also tested at a high temperature and high humidity. This is called highly accelerated stress test or HAST. Hence in every batch few chips will be tested using this to understand whether the chips made during those processes are prone to failure in the medium term.

The length of the metal line and the current density passing through that determine the MTF. The minimum value of the product of length and current density ( $L * j$ ) needed for electromigration to cause catastrophic failure is called blech product, named after I.A. Blech, who proposed it first. The higher the current density or longer the line, more the chance of failure by electromigration. For a given current density and material choice (Cu or Al), the critical length (minimum length) above which electromigration can cause failure is called Blech length.

## UNIT-IV

# GATE LEVEL DESIGN

## UNIT 4

### Introduction

The module (integrated circuit) is implemented in terms of logic gates and interconnections between these gates. Designer should know the gate-level diagram of the design. In general, gate-level modeling is used for implementing lowest level modules in a design like, full-adder, multiplexers, etc.

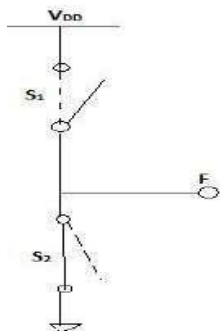
Boolean algebra is used to represent logical (combinational logic) functions of digital circuits. A combinational logic expression is a mathematical formula which is to be interpreted using the laws of Boolean algebra. Now the goal of logic design or optimization is to find a network of logic gates that together compute the combinational logic function we want.

For example, given the expression  $a+b$ , we can compute its truth value for any given values of  $a$  and  $b$ , and also we can evaluate relationships such as  $a+b = c$ . but logic design is difficult for many reasons:

- We may not have a logic gate for every possible function, or even for every function of  $n$  inputs.
- Not all gate networks that compute a given function are alike—networks may differ greatly in their area and speed.
- Thus combinational logic expressions are the specification,
- A **logic gate** is an idealized or physical device implementing a Boolean function, that is, it performs a logical operation on one or more logic inputs and produces a single logic output.
- Logic gates are primarily implemented using diodes or transistors acting as electronic switches, but can also be constructed using electromagnetic relays (relay logic), fluidic logic, pneumatic logic, optics, molecules, or even mechanical elements.
- With amplification, logic gates can be cascaded in the same way that Boolean functions can be composed, allowing the construction of a physical model of all of Boolean logic.
- simplest form of electronic logic is diode logic. This allows AND and OR gates to be built, but not inverters, and so is an incomplete form of logic. Further, without some kind of amplification it is not possible to have such basic logic operations cascaded as required for more complex logic functions.
- To build a functionally complete logic system, relays, valves (vacuum tubes), or transistors can be used.
- The simplest family of logic gates using bipolar transistors is called resistor-transistor logic (RTL). Unlike diode logic gates, RTL gates can be cascaded indefinitely to produce more complex logic functions. These gates were used in early integrated circuits. For higher speed, the resistors used in RTL were replaced by diodes, leading to diode-transistor logic (DTL).
- Transistor-transistor logic (TTL) then supplanted DTL with the observation that one transistor could do the job of two diodes even more quickly, using only half the space.
- In virtually every type of contemporary chip implementation of digital systems, the bipolar transistors have been replaced by complementary field-effect transistors (MOSFETs) to reduce size and power consumption still further, thereby resulting in complementary metal–oxide–semiconductor (CMOS) logic. that can be described with Boolean logic.

## cMOS logic gates and other complex gates

**General logic circuit** Any Boolean logic function ( $F$ ) has two possible values, either logic 0 or logic 1. For some of the input combinations,  $F = 1$  and for all other input combinations,  $F = 0$ . So in general, any Boolean logic function can be realized using a structure as shown in figure.

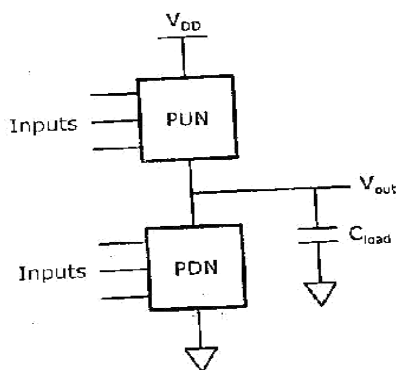


- The switch  $S_1$  is closed and switch  $S_2$  is open for input combinations that produces  $F = 1$ .
- The switch  $S_1$  is open and switch  $S_2$  is closed for input combinations that produces  $F = 1$ .
- The switch  $S_1$  is open and switch  $S_2$  is open for input combinations that produces  $F = 0$ .

Thus the output ( $F$ ) is either connected to  $V_{DD}$  or the ground, where the logic 0 is represented by the ground and the logic 1 is represented by  $V_{DD}$ . So the requirement of digital logic design is to implement the pull-up switch( $S_1$ ) and the pull-down switch( $S_2$ ).

### CMOS static logic

A generalized CMOS logic circuit consists of two transistor nets nMOS and pMOS. The pMOS transistor net is connected between the power supply and the logic gate output called as pull-up network, Whereas the nMOS transistor net is connected between the output and ground called as pull-down network. Depending on the applied input logic, the PUN connects the output node to  $V_{DD}$  and PDN connects the output node to the ground.

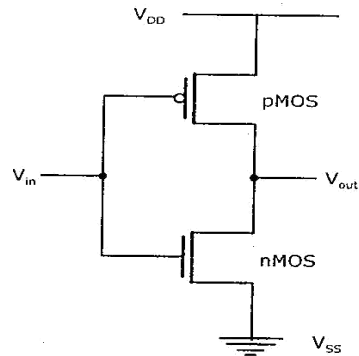


The transistor network is related to the Boolean function with a straight forward design procedure:

- Design the pull down network (PDN) by realizing, AND(product) terms using series-connected nMOSFETs. OR (sum) terms using parallel-connected nMOSFETs.
- Design the pull-up network by realizing, AND(product) terms using parallel-connected nMOSFETs. OR (sum) terms using series-connected nMOSFETs.
- Add an inverter to the output to complement the function. Some functions are inherently negated, such as NAND, NOR gates do not need an inverter at the output terminal.

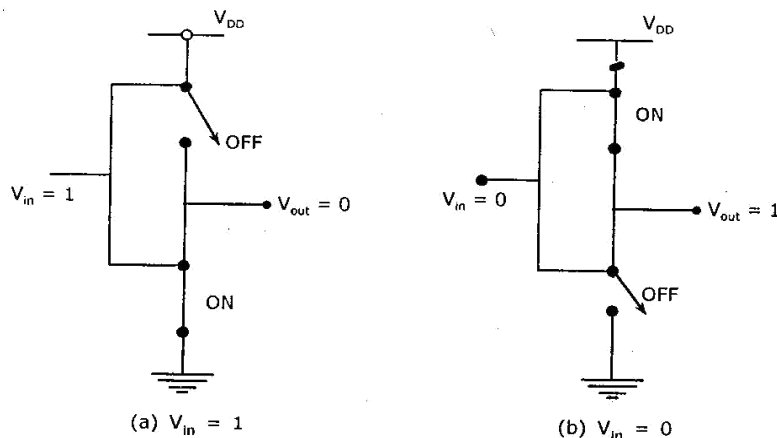
## CMOS inverter

A CMOS inverter is the simplest logic circuit that uses one nMOS and one pMOS transistor. The nMOS is used in PDN and the pMOS is used in the PUN as shown in figure.



### Working operation

- 1) When the input  $V_{in}$  is logic HIGH, then the nMOS transistor is ON and the pMOS transistor is OFF. Thus the output Y is pulled down to ground (logic 0) since it is connected to ground but not to source  $V_{DD}$ .
- 2) When the input  $V_{in}$  is logic LOW, then nMOS transistor is OFF and the pMOS transistor is ON, Thus the output Y is pulled up to  $V_{DD}$ (logic 1) since it is connected to source via pMOS but not to ground.



## CMOS NAND gate

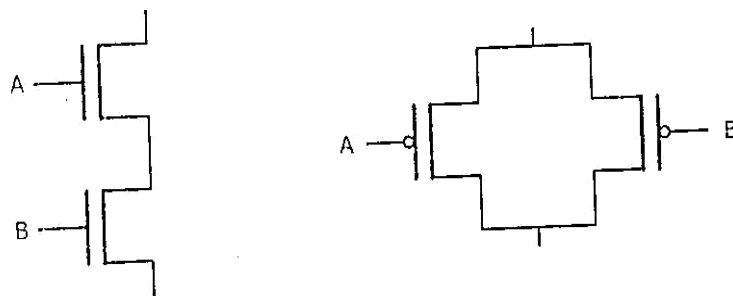
The two input NAND function is expressed by  $Y = \overline{A \cdot B}$

**Step 1** Take complement of Y

$$Y = \overline{A \cdot B} = \overline{A} + \overline{B}$$

**Step 2** Design the PDN In this case, there is only one AND term, so there will be two nMOSFETs in series as shown in figure.

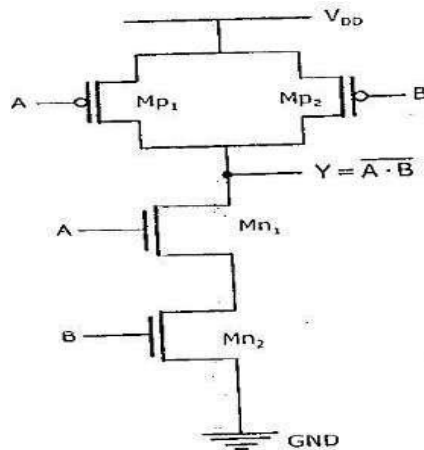
**Step 3** Design the PUN. In PUN there will be two pMOSFETs in parallel, as shown in figure



(a) Pull-down Network Comprising nMOSFETs

(b) Pull-up Network Comprising pMOSFETs

inally join the PUN and PDN as shown in figure which realizes two –input NAND gate. Note that we have realized y, rather tat Y because the inversion is automatically provided by the nature of the CMOS circuit operation,



Working operation

- 1) Whenever at least one of the inputs is LOW, the corresponding pMOS transistor will conduct while the corresponding nMOS transistor will turn OFF. Subsequently, the output voltage will be HIGH.
- 2) Conversely, if both inputs are simultaneously HIGH, then both pMOS transistors will turn OFF, and the output voltage will be pulled LOW by the two conducting nMOS transistors.

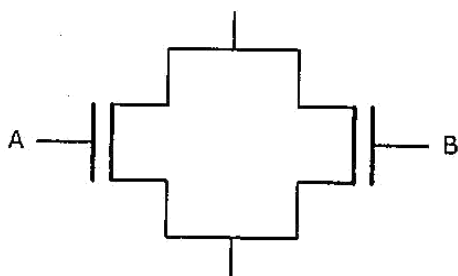
### CMOS NOR gate

The two input NOR function is expressed by  $Y=A+B$

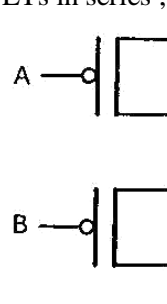
**Step 1** Take complement of Y,  $Y= A+B = A+B$

**Step 2** Design the PDN In this case, there is only one OR term, so there will be two nMOSFETs connected in parallel, as shown in figure.

**Step 3** Design the PUN In PUN there will be two pMOSFETs in series , as shown in figure



(a) Pull-down Network Comprising nMOSFETs

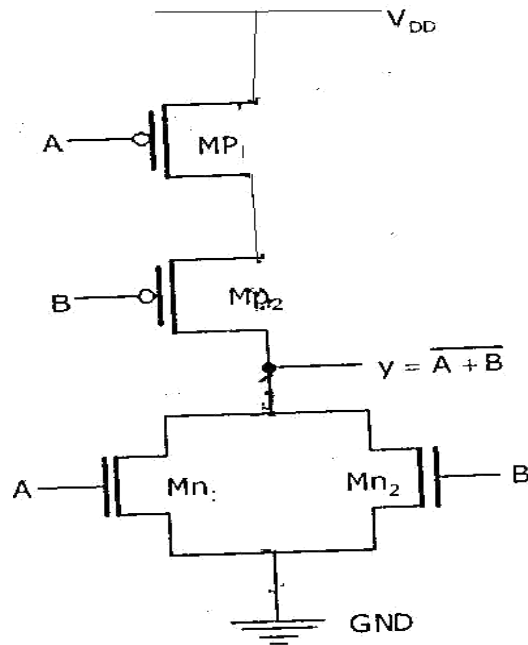


(b) Pull-up Network Comprising pMOSFETs

Finally join the PUN and PDN as shown in figure which realizes two –input NAND gate. Note that we have realized y, rather tat Y because the inversion is automatically provided by the nature of the CMOS circuit operation,

Working operation

- 1) Whenever at least one of the inputs is LOW, the corresponding pMOS transistor will conduct while the corresponding nMOS transistor will turn OFF. Subsequently, the output voltage will be HIGH.
- 2) Conversely, if both inputs are simultaneously HIGH, then both pMOS transistors will turn OFF, and the output voltage will be pulled LOW by the two conducting nMOS transistors.



### Complex gates in CMOS logic

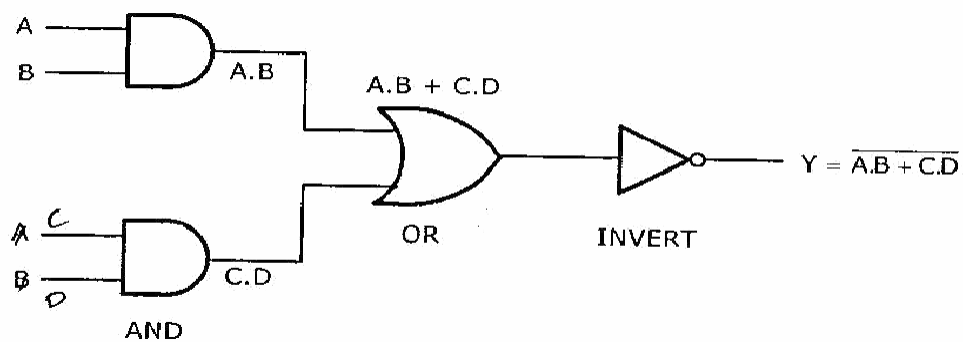
A complex logic gate is one that implements a function that can provide the basic NOT, AND and OR operation but integrates them into a single circuit. CMOS is ideally suited for creating gates that have logic equations by exhibiting the following,

An AOI logic equation is equivalent to a complemented SOP form, while an AOI equation is equivalent to a complemented POS structure. In CMOS, output always produces NOT operation acting on input variable.

#### 1) AOI Logic Function (OR) Design of XOR gate using CMOS logic.

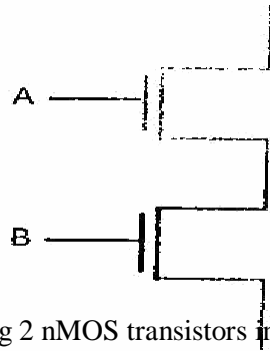
AND-OR-INVERT logic function(AOI) implements operation in the order AND,OR,NOT. For example ,

let us consider the function  $Y = \overline{AB + CD}$  i.e.,  $Y = \text{NOT}((A \text{ AND } B) \text{ OR } (C \text{ AND } D))$  The AOI logic gate implementation for Y

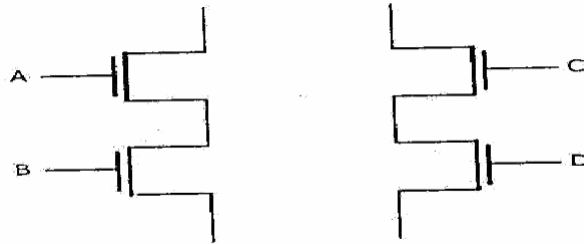


#### CMOS implementation for Y

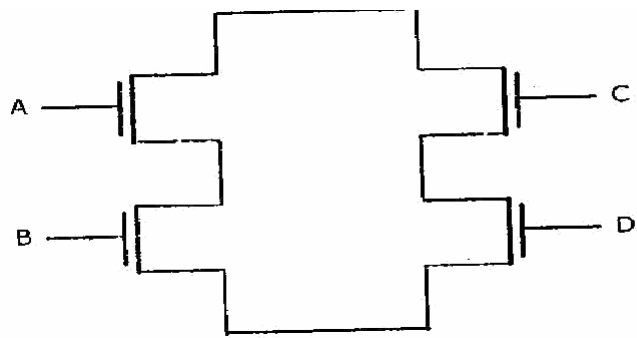
**Step 1:** Draw A.B (AND) function first by connecting 2 nMOS transistors in series.



2: Draw C.D implementation, by using 2 nMOS transistors in series.

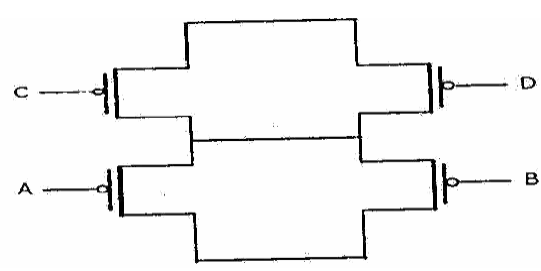


**Step 3:**  $Y = A.B + C.D$  , In this function A.B and C.D are added, for addition , we have to draw parallel connection. So, A.B series connected in parallel with C.D as shown in figure.

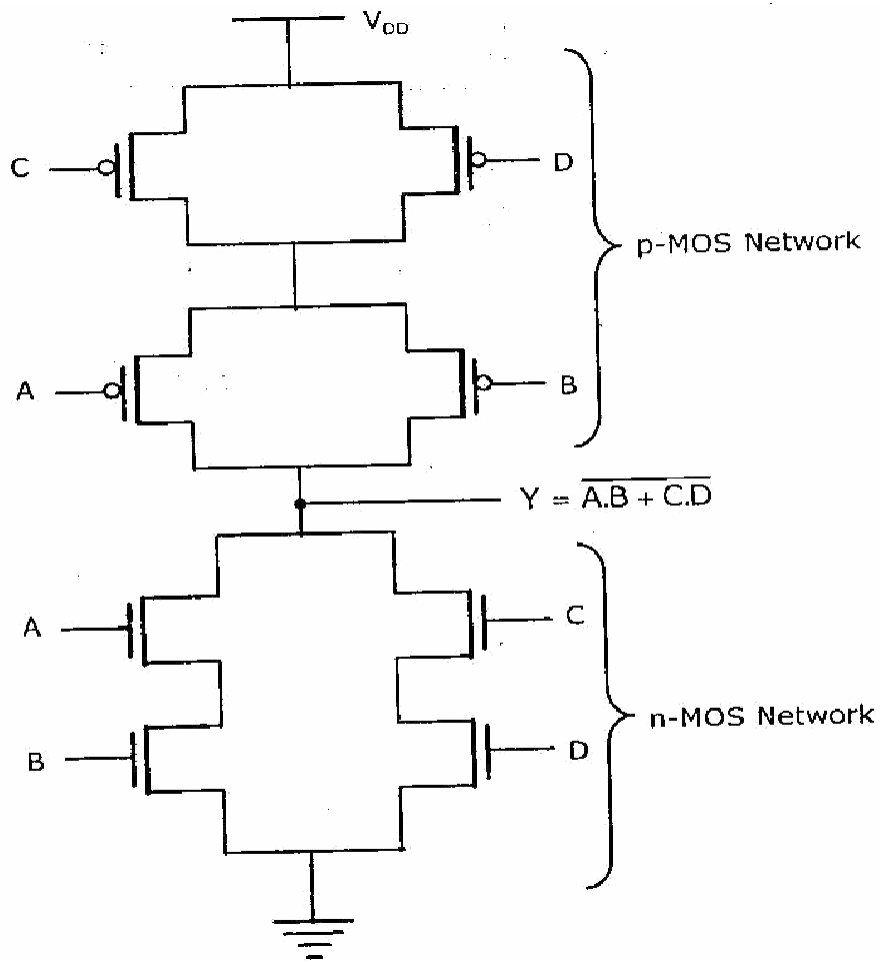


**Step 4:** Draw pMOS connection,

- In nMOS A,B connected in series. So, in pMOS side, A.B should be connected in parallel.
- In nMOS C,D connected in series. So, in pMOS side, C.D should be connected in parallel.
- A.B and C.D networks are connected in parallel in nMOS side. So, in pMOS side, A.B and C.D networks should be connected in series.
- In pMOS multiplication should be drawn in parallel, then addition should be drawn in series as shown in figure.



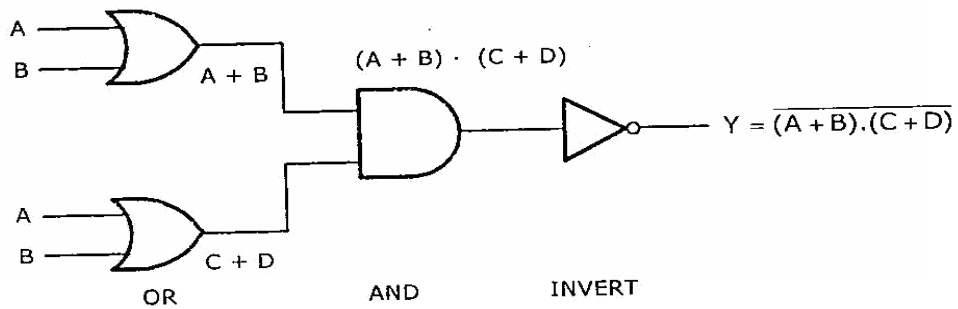
**Step 5:** Take output at the point in between nMOS and pMOS networks.



**1) OAI Logic Function (OR) Design of XNOR gate using CMOS logic.**

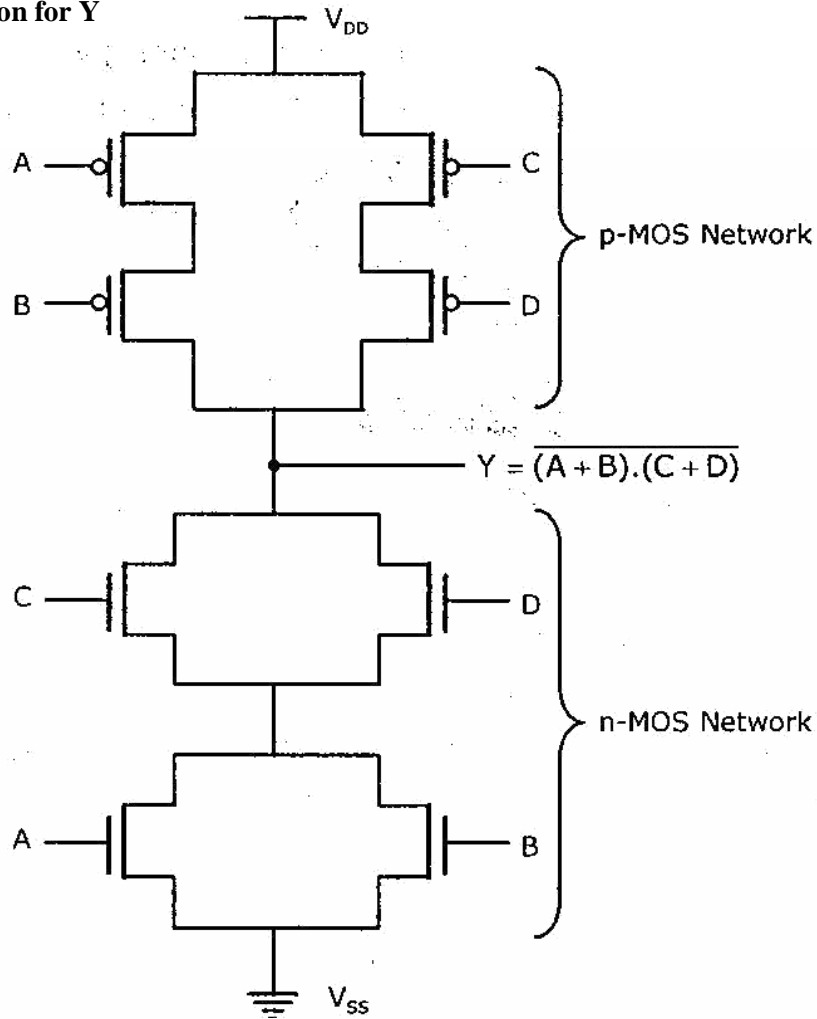
OR-AND-INVERT logic function(AOI) implements operation in the order OR,AND,NOT. For example ,let us consider the function  $Y = (A+B).(C+D)$  i.e.,  $Y = \overline{(A \text{ OR } B) \text{ AND } (C \text{ OR } D)}$

The OAI logic gate implementation for Y





**MOS implementation for Y**

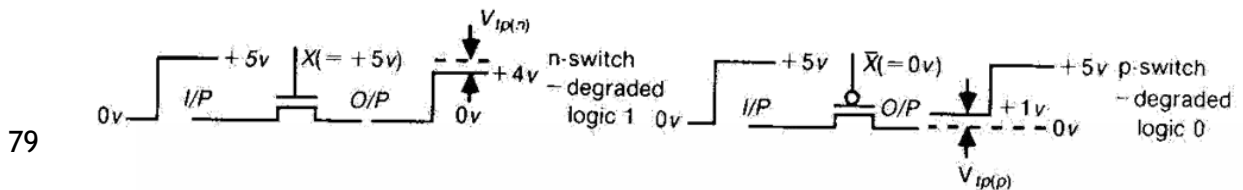


**SWITCH LOGIC**

- 1) Switch logic is mainly based on pass transistor or transmission gate.
- 2) It is fast for small arrays and takes no static current from the supply,  $V_{DD}$ . Hence power dissipation of such arrays is small since current only flows on switching.
- 3) Switch (pass transistor) logic is analogous to logic arrays based on relay contacts, where in path through each switch is isolated from the logic levels activating the switch.

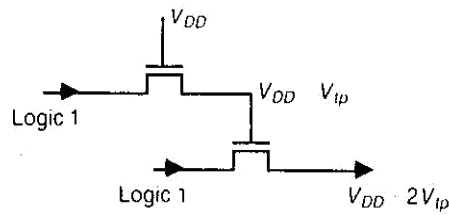
**PASS TRANSISTOR**

- 1) This logic uses transistors as switches to carry logic signals from node to node instead of connecting output nodes directly to  $V_{DD}$  or ground(GND)
- 2) If a single transistor is a switch between two nodes, then voltage degradation equal to  $v_t$  (threshold voltage) for high or low level depends up on nMOS or pMOS logic.



79

- 3) When using nMOS switch logic no pass transistor gate input may be driven through one or more pass transistors as shown in figure.



Loss of logic level 1 if the gate of a pass transistor is driven from another pass transistor.

- 4) Since the signal out of pass transistor T<sub>1</sub> does not reach a full logic 1 by threshold voltage effects signal is degraded by below a true logic 1, this degraded voltage would not permit the output of T<sub>2</sub> to reach an acceptable logic 1 level.

### Advantages

They have topological simplicity.

- 1) Requires minimum geometry.
- 2) Do not dissipate standby power, since they do not have a path from supply to ground.

### Disadvantages

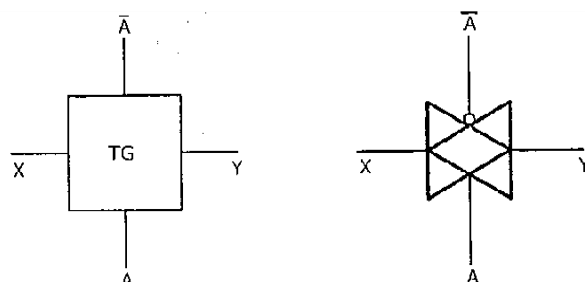
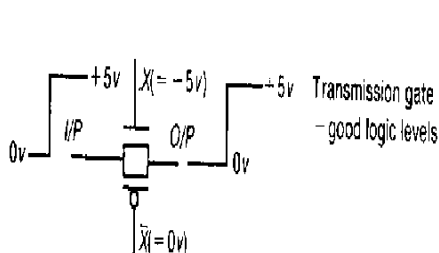
- 1) Degradation in the voltage levels due to undesirable threshold voltage effects.
- 2) Never drive a pass transistor with the output of another pass transistor.

### TRANSMISSION GATE

- 1) It is an electronic element, good non-mechanical relay built with CMOS technology.
- 2) It is made by parallel combination of an nMOS and pMOS transistors with the input at gate of one transistor being complementary to the input at the gate of the other as shown in figure.
- 3) Thus current can flow through this element in either direction.
- 4) Depending on whether or not there is a voltage on the gate, the connection between the input and output is either low resistance or high-resistance, respectively R<sub>on</sub> = 100Ω and R<sub>off</sub> > 5 MΩ.

### Operation

- When the gate input to the nMOS transistor is '0' and the complementary '1' is gate input to the pMOS, thus both are turned off.
- When gate input to the nMOS is '1' and its complementary '0' is the gate input to the pMOS, both are turned on and passes any signal '1' and '0' equally without any degradation.
- The use of transmission gates eliminates the undesirable threshold voltage effects which give rise to loss of logic levels in pass-transistors as shown in figure.



80

### Advantages

- 1) Transmission gates eliminates the signal degradation in the output logic levels.
- 2) Transmission gate consists of two transistors in parallel and except near the positive and negative rails.

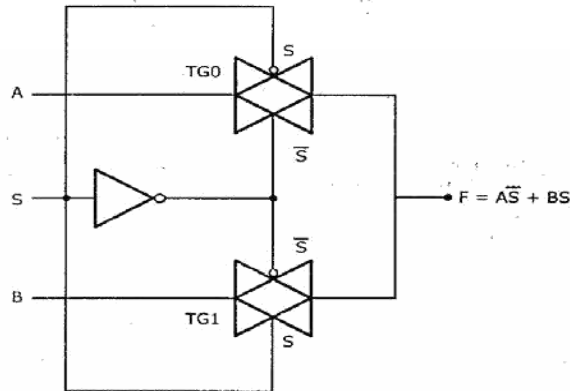
### Disadvantages

- 1) Transmission gate requires more area than nMOS pass circuitry.
- 2) Transmission gate requires complemented control signals.

“ Transmission gate logic can be used to design multiplexers(selector functions)”.

**Design a 2-input multiplexer using CMOS transmission gates.**

Figure shows a 2-input multiplexer circuit using CMOS transmission gate.



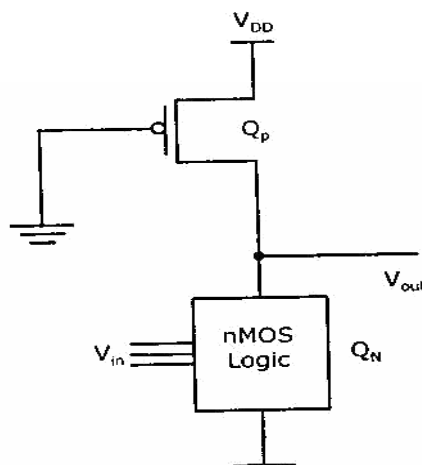
If the control input S is low, the TG0 conducts and the output F is equal to A. On the other hand, if the control input S is high the TG1 conducts and the output F is equal to B.

### ALTERNATIVE GATE CIRCUITS

CMOS suffers from increased area and correspondingly increased capacitance and delay, as the logic gates become more complex. For this reason, designers developed circuits (Alternate gate circuits) that can be used to supplement the complementary type circuits. These forms are not intended to replace CMOS but rather to be used in special applications for special purposes.

### PSEUDO nMOS Logic

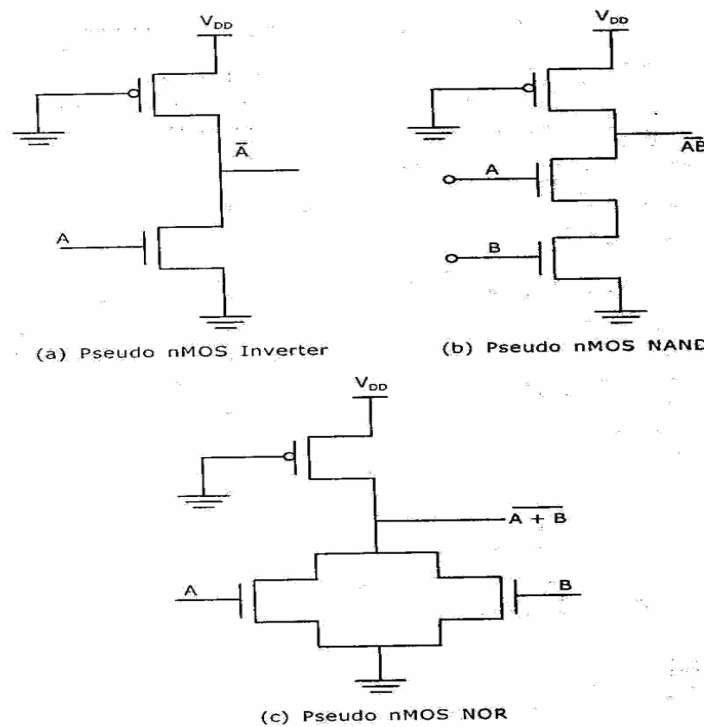
Pseudo nMOS logic is one type of alternate gate circuit that is used as a supplement for the complementary MOS logic circuits. In the pseudo-nMOS logic, the pull up network (PUN) is realized by a single pMOS transistor. The gate terminal of the pMOS transistor is connected to the ground. It remains permanently in the ON state. Depending on the input combinations, output goes low through the PDN. Figure shows the general building block of logic circuits that follows pseudo nMOS logic.



Here, only the nMOS logic (Qn) is driven by the input voltage, while the gate of p-transistor(Qp) is connected

to ground or substrate and  $Q_p$  acts as an active load for  $Q_n$ . Except for the load device, the pseudo-nMOS gate circuit is identical to the pull-down network(PDN) of the complementary CMOS gate.

The realization of logic circuits using pseudo-nMOS logic is as shown in figure.



### Advantages

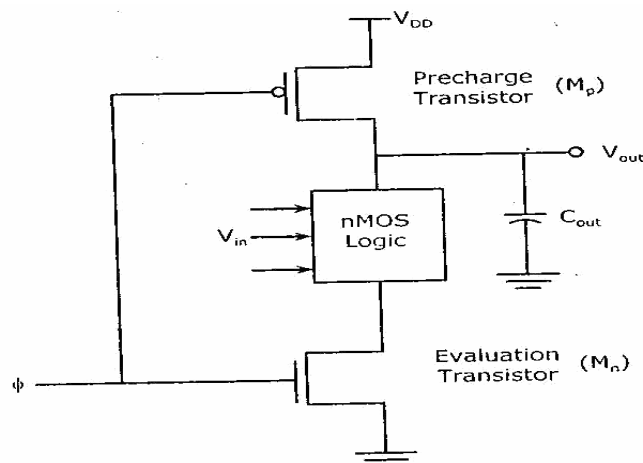
- 1) Uses less number of transistors as compared to CMOS logic.
- 2) Geometrical area and delay gets reduced as it requires less transistors.
- 3) Low power dissipation.

### Disadvantages

- 1) The main drawback of using a pseudo nMOS gate instead of a CMOS gate is that the always on PMOS load conducts a steady current when the output voltage is lower than  $V_{DD}$ .
- 2) Layout problems are critical.

### DYNAMIC CMOS LOGIC

A dynamic CMOS logic uses charge storage and clocking properties of MOS transistors to implement logic operations. Figure shows the basic building block of dynamic CMOS logic. Here the global clock  $\phi$  drives nMOS evaluation transistor and pMOS precharge transistor. A logic is implemented using an nFET array connected between output node and ground.

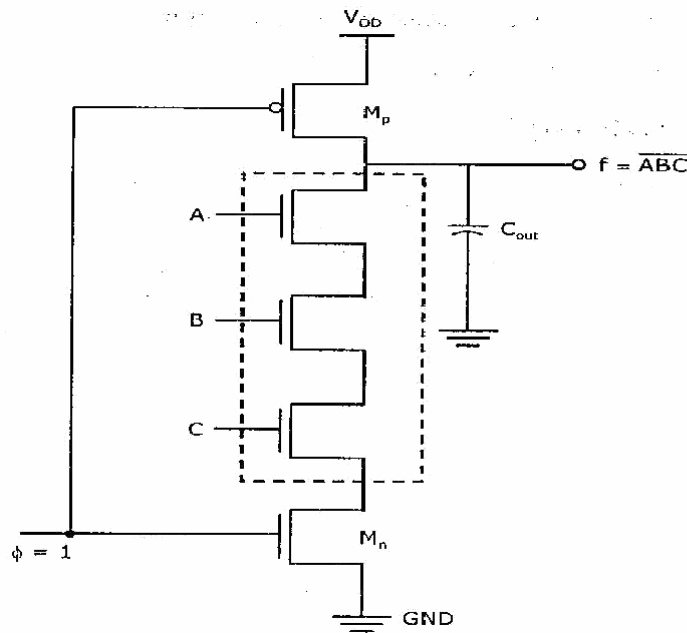


The gate (clock  $\phi$ ) defines two phases, evaluation and precharge phase during each clock cycle.

### Working

- When clock  $\phi = 0$  the circuit is in precharge phase with the pMOS device  $M_p$  ON and the evaluation nMOS  $M_n$  OFF. This establishes a conducting path between  $V_{DD}$  and the output allowing  $C_{out}$  to charge to a voltage  $V_{out} = V_{DD}$ .  $M_p$  is often called the precharge FET.
- When clock  $\phi = 1$  the circuit is in evaluation phase with the pMOS device  $M_p$  OFF and the evaluation nMOS  $M_n$  ON. If the logic block acts like a closed switch the  $C_{out}$  can discharge through logic array and  $M_n$ , this gives a final result of  $V_{out} = 0$ , logically this is an output of  $F = 1$ . Charge leakage eventually drops the output to  $V_{out} = 0$  V which could be an incorrect logic value.

The logic formation is formed by three series connected FETs (3-input NAND gate) is shown in figure.



The dynamic CMOS logic circuit has a serious problem when they are cascaded. In the precharged phase ( $\phi = 0$ ), output of all the stages are pre-charged to logic high. In the evaluation phase ( $\phi = 1$ ), the output of all stages are evaluated simultaneously. Suppose in the first stage, the inputs are such that the output is logic low after the evaluation. In the second stage, the output of the first stage is one input and there are other inputs. If the other inputs of the second stage are such that output of it discharges to logic low, then the evaluated output of the first stage can never make the output of the second stage logic high. This is because, by the time the first stage is being evaluated, output of the second stage is discharged, since evaluation happens simultaneously. Remember that the output cannot be charged to logic high in the evaluation phase ( $\phi = 1$ , pMOSFET in PUN is OFF), it can only be retained in the logic high depending on the inputs.

### Advantages

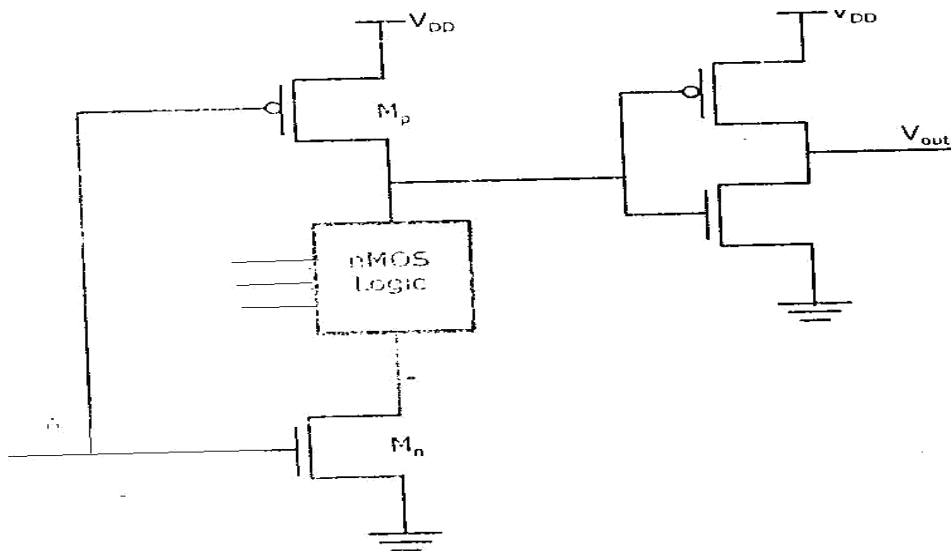
- 1) Low power dissipation.
- 2) Large noise margin.
- 3) <sup>83</sup> Small area due to less number of transistors.\

## CMOS DOMINO LOGIC

Standard CMOS logic gates need a PMOS and an NMOS transistor for each logic input. The pMOS transistors require a greater area than the nMOS transistors carrying the same current. So, a large chip area is necessary to perform complex logic operations. The package density in CMOS is improved if a dynamic logic circuit, called the domino CMOS logic circuit, is used.

Domino CMOS logic is a slightly modified version of the dynamic CMOS logic circuit. In this case, a static inverter is connected at the output of each dynamic CMOS logic block. The addition of the inverter solves the problem of cascading of dynamic CMOS logic circuits.

The circuit diagram of domino CMOS logic structures as shown in figure as follows



A domino CMOS AND-OR gate that realizes the function  $y = AB + CD$  is depicted in figure .

The left hand part of the circuit containing  $M_n, M_p, T_1, T_2, T_3,$  and  $T_4$  forms an AND-OR-INVERTER (AOI) gate. It derives the static CMOS inverter formed by  $N_2$  and  $P_2$  in the right-hand part of the circuit. The domino gate is activated by the single phase clock  $\phi$  applied to the NMOS ( $M_n$ ) and the PMOS ( $M_p$ ) transistors. The load on the AOI part of the circuit is the parasitic load capacitance.

### Working

- When  $\phi = 0$ ,  $M_p$  is ON and  $M_n$  is OFF, so that no current flows in the AND-OR paths of the AOI. The capacitor  $C_L$  is charged to  $V_{DD}$  through  $M_p$  since the latter is ON. The input to the inverter is high, and drives the output voltage  $V_0$  to logic-0.

- When  $\phi = 1$ ,  $M_p$  is turned OFF and  $M_n$  is turned ON. If either (or both) A and B or C and D is at logic-1,  $C_L$  discharges through either  $T_2, T_1$  and  $M_n$  or  $T_3, T_4$  and  $M_p$ . So, the inverter input is driven to logic-0 and hence the output voltage  $V_0$  to logic-1. The Boolean expression for the output voltage is  $Y = AB + CD$ .

**Note :** Logic input can change only when  $\phi = 0$ . No changes of the inputs are permitted when  $\phi = 1$  since a discharge path may occur.

## Advantages

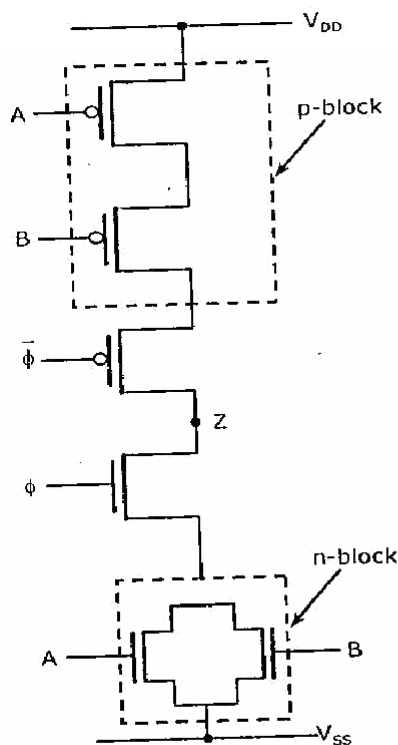
- 1) Smaller areas compared to conventional CMOS logic.
- 2) Parasitic capacitances are smaller so that higher operating speeds are possible.
- 3) Operation is free of glitches since each gate can make one transition.

## disadvantages

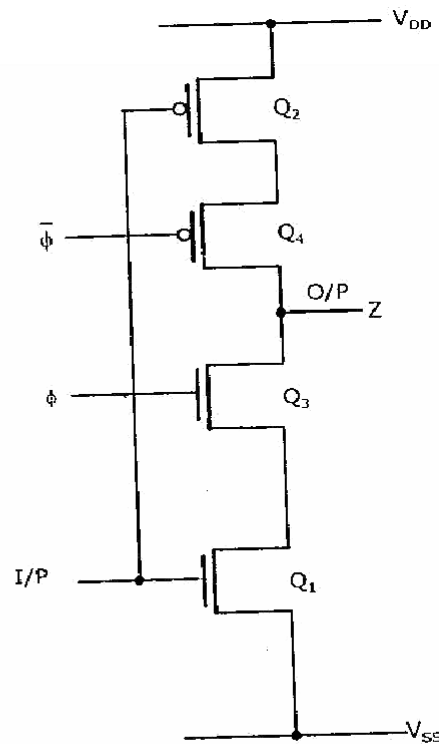
- 1) Non inverting structures are possible because of the presence of inverting buffer.
- 2) Charge distribution may be a problem.

## CLOCKED CMOS LOGIC

The clocked CMOS logic is also referred as C<sup>2</sup>MOS logic. Figure shows the general arrangement of a clocked CMOS (C<sup>2</sup>MOS) logic. A pull-up p-block and a complementary n-block pull-down structure represent p and n-transistors respectively and are used as implement clocked CMOS logic shown in figure. However, the logic in this case is connected to the output only during the ON period of the clock. Figure shows a clocked inverter circuit which is also belongs to clocked CMOS logic family. The slower rise times and fall times can be expected due to owing of extra transistors in series with the output.



(a) Implemented CMOS Logic



(b) Clocked Inverter Circuit

## Working

- When  $\phi = 1$  the circuit acts as an inverter, because transistors Q3 and Q4 are 'ON'. It is said to be in the "evaluation mode". Therefore the output Z changes its previous value.
- When  $\phi = 0$  the circuit is in hold mode, because transistors Q3 and Q4 become 'OFF'. It is said to be in the "precharge mode". Therefore the output Z remains its previous value.

## n-p CMOS LOGIC

Figure shows another variation of basic dynamic logic arrangement of CMOS logic called as n-p CMOS logic. In this logic, the actual logic blocks are alternatively 'n' and 'p' in a cascaded structure. The clock  $\phi$  and  $\phi$  are used alternatively to feed the precharge and evaluate transistors. However, the functions of top and bottom transistors are also alternate between precharge and evaluate transistors.

## Working

- During the pre charge phase  $\phi = 0$ , the output of the n-tree gate, OUT 1 OUT3, are charged to  $V_{DD}$ , while the output of the p-tree gate OUT2 is pre discharged to 0V. Since the n-tree gate connects pMOS pull-up devices, the PUN of the p-tree is turned off at that time.
- During the evaluation phase  $\phi = 1$ , the outputs (OUT1,OUT3) of the n-tree gate can only make a 1- 0 transition, conditionally turning on some transistors in the p-tree. This ensures that no accidental discharge of OUT 2 can occur.
- Similarly n-tree blocks can follow p-tree gates without any problems, because the inputs to the n-gate are pre charged to 0.

## Disadvantages

Here, the p-tree blocks are slower than the n-tree modules, due to the lower current drive of the pMOS transistors in the logic network.

## BASIC CIRCUIT CONCEPTS

In VLSI design the wiring up (interconnection) of circuits takes place through the various conductive layers which are produced by the MOS processing. So, it is necessary to know the resistive and capacitive characteristics of each layer. Concepts such as

- resistance  $R_S$  and a standard unit of capacitance  $\square c_g$  which helps in evaluating the effects of wiring and input and output capacitances.
- The delays associated with wiring with inverters and with other circuitry evaluated in terms of a delay unit  $\tau$ .

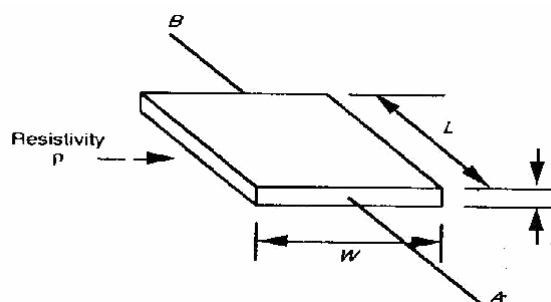
## Sheet Resistance $R_S$

- The sheet resistance is a measure of resistance of thin films that have a uniform thickness.
- It is commonly used to characterize materials made by semiconductor doping, metal deposition, resistive paste printing and glass coating.

Ex: doped semiconductor regions (silicon or polysilicon) and resistors.

- Sheet resistance is applicable to two-dimensional systems where the thin film is considered to be a two-dimensional entity.

Consider a uniform slab of conducting material of resistivity  $\rho$  of width  $W$ , thickness  $t$  and length between faces A&B is  $L$ , as shown in figure.



Consider the resistance  $R_{AB}$  between two opposite faces

86

$$R_{AB} = \frac{\rho L}{A} \text{ ohm}$$

Where A is area of cross section



$$R_{AB} = \frac{\rho L}{tW} \text{ ohm}$$

Consider a case in which  $L = W$ . It means square of resistive material

$$R_{AB} = \frac{\rho}{t} = R_s$$

Where  $R_s$  is ohm per square or sheet resistance.

$$R_s = \frac{\rho}{t} \text{ ohm per square}$$

From the above equation  $R_s$  is independent of the area of square., for example a  $1\mu\text{m}$  per side square slab of the material has same resistance as 1 cm per side square slab of the same material if the thickness is same.

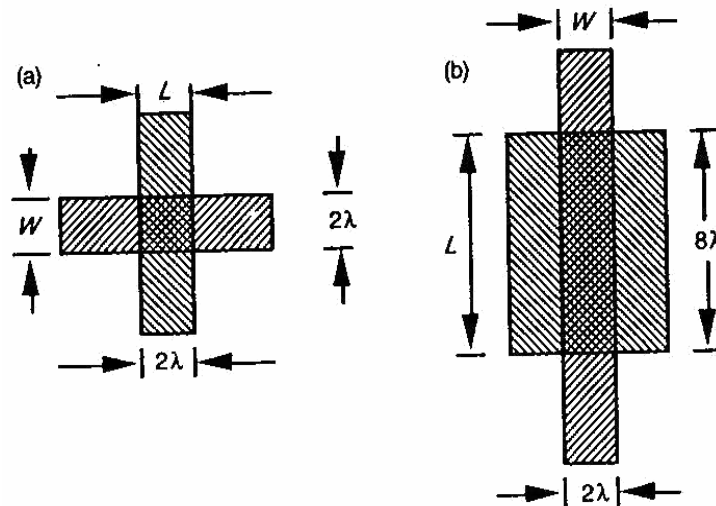
Hence, the resistance of the MOS layers depend on the thickness and the resistivity of the material of the layer.

- The thickness of the metal and polysilicon deposited is known by measuring using four probe method.
- The resistivity of the diffusion layers is measured by measuring the penetration depth of the diffusion regions.

**Sheet resistance concept applied to MOS Transistors and Inverters:**

Consider the transistor structures by distinguish the actual diffusion (active) regions from the channel regions.

The simple n-type pass transistor has a channel length  $L = 2\lambda$  and a channel width  $W=2\lambda$ .



Hence the channel is square and the channel resistance is

$$R = 1 \text{ square} \times R_s \frac{\text{ohm}}{\text{square}} = R_s = 10^4 \text{ ohm}$$

Here the length to width ratio denotes the impedance ( $Z$ ) and is equal to 1:1. Consider another transistor has a channel length  $L = 8\lambda$  and width  $W = 2\lambda$ .

$$Z = \frac{L}{W} = 4$$

Thus, channel resistance

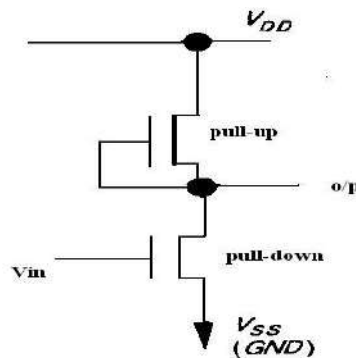
$$R = ZR_s = 4 \times 10^4 \text{ ohm}$$

Typical sheet resistances of MOS layers are tabulated

Layer	Rs ohm per square		
	5μm	Orbit	Orbit 1.2μm
Metal	0.03	0.04	0.04
Diffusion	15 - 100	20 - 45	20 - 45
Silicide	2 - 4	---	---
Polysilicon	15 - 100	15 - 30	15 - 30
n- channel	$10^4$	$2 \times 10^4$	$2 \times 10^4$
p- channel	$2.5 \times 10^4$	$4.5 \times 10^4$	$4.5 \times 10^4$

### Sheet resistance for Inverters

Consider an nMOS inverter has the channel length  $8\lambda$  and width  $2\lambda$  for pull up transistor as shown in figure.



$$L = 8\lambda; W = 2\lambda$$

$$Z = L/W = 4$$

$$\text{Sheet resistance } R = Z.R_s = 4 \times 10^4 = 40 \text{ K}\Omega$$

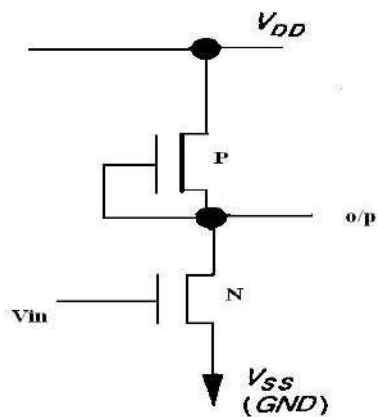
For pull down transistor the channel length  $2\lambda$  and width  $2\lambda$ , then the sheet resistance is

$$R = Z.R_s = 1 \times 10^4 = 10 \text{ K}\Omega$$

Hence  $Z_{p,u}$  to  $Z_{p,d}$  = 4:1 hence the ON resistance between  $V_{DD}$  and  $V_{SS}$  is the total series resistance i.e.,

$$R_{ON} = 40 \text{ K}\Omega + 10 \text{ K}\Omega = 50 \text{ K}\Omega$$

Consider the simple CMOS inverter as shown in figure.



Here the pull up transistor is of p-type device with channel length  $2\lambda$  and width  $2\lambda$ .

$$Z = L/W = 1$$

$$\text{Then Sheet resistance } R_{SP} = Z.R_S = 1 \times 2.5 \times 10^4 = 25 \text{ K}\Omega$$

The pull down transistor is of n-type with channel length  $2\lambda$  and width  $2\lambda$ .

$$Z = L/W = 1$$

$$\text{Hence, Sheet resistance } R_{SN} = Z.R_S = 1 \times 10^4 = 10 \text{ K}\Omega.$$

In this case, there is no static resistance between  $V_{DD}$  and  $V_{SS}$ . Since at any point of time only one transistor is ON, but not both.

When  $V_{in} = 1$ , the ON resistance is  $10\text{K}\Omega$   $V_{in} = 0$ , the ON resistance is  $25\text{K}\Omega$

### Area capacitance of layers

From the concept of the transistors, it is apparent that as gate is separated from the channel by gate oxide an insulating layer, it has capacitance. Similarly different interconnects run on the chip and each layer is separated by silicon dioxide .

For any layer by knowing the dielectric thickness, we can calculate the area capacitance as follows

$$C = \frac{\epsilon_0 \epsilon_{ins} A}{D} \text{ farads}$$

Where,  $A$  is area of the plates ,  $D$  is the thickness of  $\text{SiO}_2$ ,  $\epsilon_0$  is the permittivity of the free space and  $\epsilon_{ins}$  is the relative permittivity of insulator( $\text{SiO}_2$ ).

Typical area capacitance values of MOS circuits

Layer	Value in pF X 10 <sup>-4</sup> / μm <sup>2</sup> (Relative values in brackets )		
	5μm	Orbit	Orbit 1.2μm
Gate to channel	4 (1.0)	8 (1.0)	16 (1.0)
Diffusion	1 (0.25)	1.75 (0.22)	3.75 (0.23)
Polysilicon to substrate	0.4 (0.1)	0.6 (0.075)	0.6 (0.038)
Metal 1 to substrate	0.3 (0.075)	0.33 (0.04)	0.33 (0.02)
Metal 2 to substrate	0.2 (0.05)	0.17 (0.02)	0.17 (0.01)
Metal 2 to metal 1	0.4 (0.1)	0.5 (0.06)	0.5 (0.03)
Metal 1 to poly silicon	0.3(0.075)	0.3 (0.038)	0.3 (0.018)

**Standard unit of capacitance** □c<sub>g</sub>

It is defined as the gate – to – channel capacitance of a MOS transistor having W = L . i.e., standard square as shown in figure. The unit is denoted by □c<sub>g</sub>. □c<sub>g</sub> may be calculated for any MOS process as follows

For **5μm** MOS circuits

$$\text{Area/standard square} = 5\mu\text{m} \times 5\mu\text{m} = 25 \mu\text{m}^2$$

$$\text{Capacitance value} = 4 \times 10^{-4} \text{ pF}/ \mu\text{m}^2$$

Thus, standard value  $\square c_g = 25 \mu\text{m}^2 \times 4 \times 10^{-4} \text{ pF}/\mu\text{m}^2 = 0.01 \text{ pF}$

For **2 $\mu\text{m}$**  MOS circuits

Area/standard square =  $2\mu\text{m} \times 2\mu\text{m} = 4 \mu\text{m}^2$

Capacitance value =  $8 \times 10^{-4} \text{ pF}/\mu\text{m}^2$

Thus, standard value  $\square c_g = 4 \mu\text{m}^2 \times 8 \times 10^{-4} \text{ pF}/\mu\text{m}^2 = 0.01 \text{ pF}$

For **1.2 $\mu\text{m}$**  MOS circuits

Area/standard square =  $1.2\mu\text{m} \times 1.2\mu\text{m} = 1.44 \mu\text{m}^2$

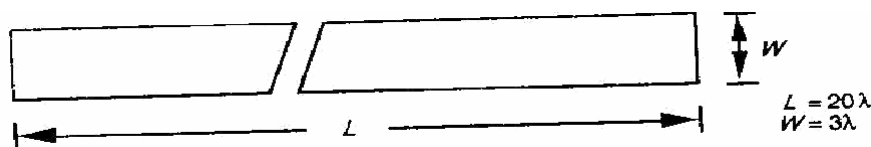
Capacitance value =  $16 \times 10^{-4} \text{ pF}/\mu\text{m}^2$

Thus, standard value  $\square c_g = 1.44 \mu\text{m}^2 \times 16 \times 10^{-4} \text{ pF}/\mu\text{m}^2 = 0.0023 \text{ pF}$

### Calculation for capacitance value

The calculation of capacitance value is established by the ration between the area of interest and the area of standard gate and multiplying this ration by the appropriate relative C value from tabular form. The product will give the required capacitance in  $\square c_g$  units.

Consider the area defined as shown in figure of length  $20\lambda$  and width  $3\lambda$



Area relative to the standard gate

Relative area =  $\text{Area}(L \times W)/\text{standard gate area}$

$$= \frac{20\lambda \times 3\lambda}{2\lambda \times 2\lambda}$$

= 15 .

1) consider the area in metal 1

capacitance to substrate = relative area X relative C value (from table)

$$= 15 \times 0.075 \square c_g$$

$$= 1.125 \square c_g$$

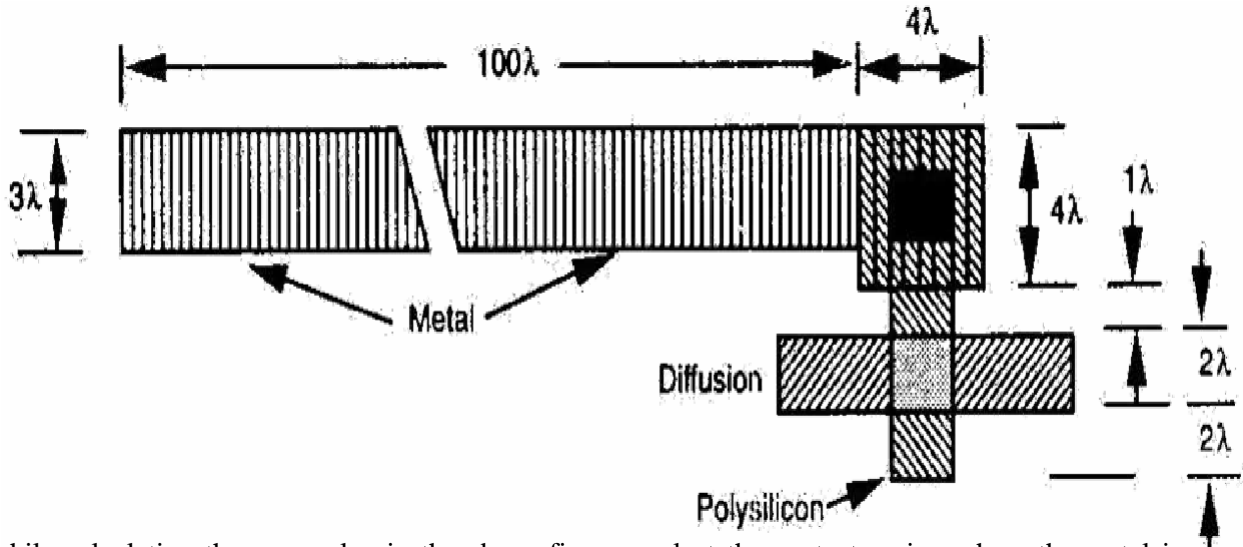
2) consider the same area in polysilicon capacitance to substrate =  $15 \times 0.1 \square c_g$

$$= 1.5 \square c_g$$

3) consider the same area in n- type diffusion capacitance to substrate =  $15 \times 0.25 \square c_g$

$$= 3.75 \square c_g$$

Consider the following structure which occupies more than one layer as shown in figure and calculate the area capacitance value



while calculating the area value in the above figure neglect the contact region where the metal is connected to polysilicon and shielded from the substrate.

(i) consider the metal area

Relative area = Area( L X W )/standard gate area

$$= \frac{100\lambda \times 3\lambda}{4\lambda^2}$$

= 75

metal capacitance = relative area X relative C value (from table)

$$= 75 \times 0.075 \square c_g$$

$$= 5.625 \square c_g$$

3) consider the polysilicon area (excluding the gate region) capacitance to substrate = 15 0.1□c<sub>g</sub>

$$= 1.5 \square c_g$$

3) consider the same area in n- type diffusion capacitance to substrate = 15 X 0.25□c<sub>g</sub>

$$= 3.75$$

We know that the transit time ( $\tau_{sd}$ ) from source to drain

$$\tau_{sd} = \frac{L^2}{\mu_n V_{ds}}$$

Here the  $V_{ds}$  varies as  $C_g$  charges from 0 volts to 63% of  $V_{dd}$  in period  $\tau$ . Thus the average value of  $V_{ds} = 3V$ .

For **5μm** technology

$$\tau_{sd} = \frac{25\mu\text{m}^2 \text{V sec}}{650 \text{ cm}^2 \text{ 3V}} \times \frac{10^9 \text{ n sec cm}^2}{10^8 \mu\text{m}^2}$$

$$\tau_{sd} = 0.13 \text{ n sec}, \tau_{sd} \approx \tau$$

Similarly the transition point of an inverter or gate is 0.5  $V_{DD}$  which is approximately equal to 0.63  $V_{DD}$  (time

constant). From this we can conclude that we can use the transit time and time constant interchangeably and 'stray' capacitances are allowed for doubling the theoretical values calculated.

Thus,  $\tau$  is used as the fundamental time unit and all timings in a system can be assessed in relation to  $\tau$ , Hence for **5 $\mu$ m MOS technology  $\tau = 0.3$  nsec.**

for **2 $\mu$ m MOS technology  $\tau = 0.2$  nsec.**

for **1.2 $\mu$ m MOS technology  $\tau = 0.1$  nsec.**

### INVERTER DELAYS

Consider the basic nMOS inverter has the channel length  $8\lambda$  and width  $2\lambda$  for pull-up transistor and channel length of  $2\lambda$  and width  $2\lambda$  for pull down transistor.

Hence the sheet resistance for pull-up transistor is  $R_{p,u} = 4R_S = 40k\Omega$  and

sheet resistance for pull-down transistor is  $R_{p,d} = 1R_S = 10k\Omega$ .

Since ( $\tau = RC$ ) depends upon the values of R & C, the delay associates with the inverter depend up on whether it is being turned on (or) off. Now, consider a pair of cascaded inverters as shown in figure, then the delay over the pair will be constant irrespective of the sense of the logic level transition of the input to the first .

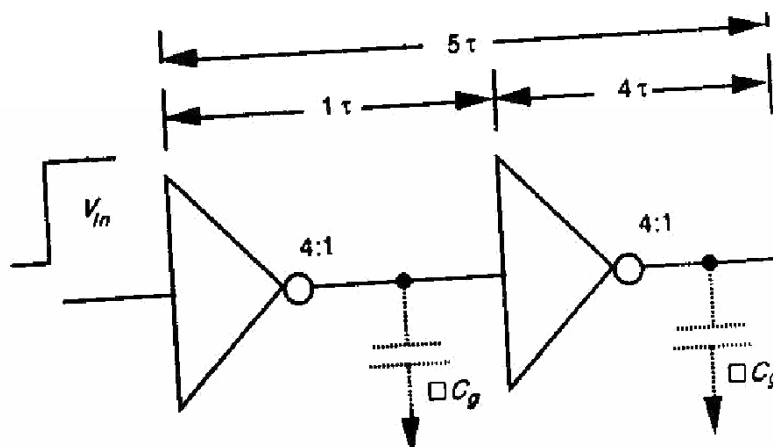
In general, the delay through a pair of similar nMOS inverters is  $T_d = (1 + Z_{p,u}/Z_{p,d}) \tau$

Assume that  $\tau = 0.3$  n sec.

Then , $T_d = (1 + 4) 0.3$

$= 5 \tau$

Thus, the inverter pair delay for inverters having 4:1 ration is  $5\tau$ .



Hence, a single 4:1 inverter exhibits undesirable asymmetric delays, Since the delay in turning ON is  $\tau$  and delay in turning OFF is  $4\tau$ .

### CMOS inverter pair delay

When we consider CMOS inverters, the rules for nMOS inverters are not applicable. But we need to consider the natural ( $R_S$ ) uneven values for equal size pull up p-transistor and the n-type pull down transistors.

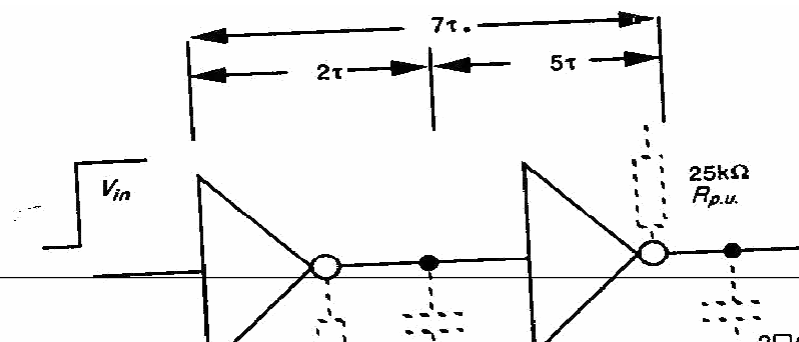


Figure shows the theoretical delay associated with a pair of both n and p transistors lambda based inverters. Here the gate capacitance is double comparable to nMOS inverter since the input to a CMOS inverter is connected to both transistor gate. **NOTE:** Here the asymmetry (uneven) of resistance values can be eliminated by increasing the width of the p-device channel by a factor of two or three at the same time the gate capacitance of p-transistor also increased by the same factor.

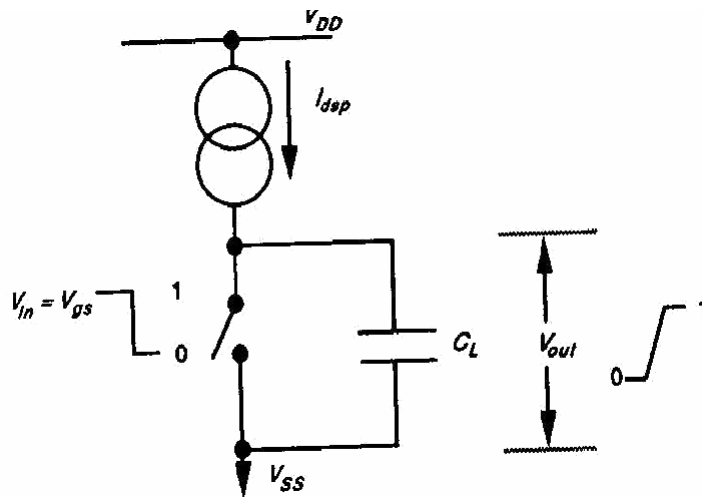
### Formal estimation of CMOS inverter delay

In CMOS inverter by the charging and discharging of a capacitive load  $C_L$ , we can estimate the Rise time and fall time from the following simple analysis.

#### Rise time estimation

In this analysis we assume that the p-device stays in saturation for the entire charging period of the load capacitor  $C_L$ .

Consider the circuit as follows



Saturation current for the p-transistor is given by

$$I_{dsp} = \frac{\beta_p (V_{gs} - |V_{tp}|)^2}{2}$$

This current charges  $C_L$  and since its magnitude is approximately constant, we have

$$V_{out} = \frac{I_{dsp} t}{C_L}$$

Substitute the value of  $I_{dsp}$  in above equation and then the rise time is

$$t = \frac{2 C_L V_{out}}{\beta_p (V_{gs} - |V_{tp}|)^2}$$

Assume that  $t = \tau_r$  when  $V_{out} = V_{DD}$  then

$$\tau_r = \frac{2 V_{DD} C_L}{\beta_p (V_{DD} - |V_{tp}|)^2}$$

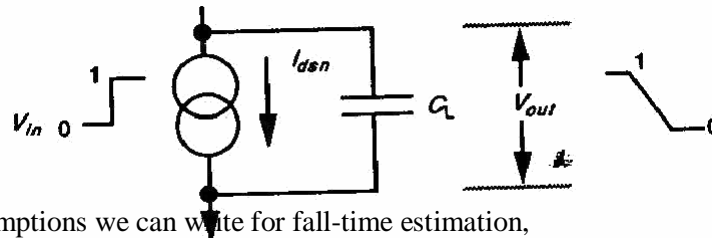


If  $V_{ip} = 0.2V_{DD}$ , then

$$\tau_r \doteq \frac{3C_L}{\beta_p V_{DD}}$$

### Fall time estimation

Consider the circuit for discharge of  $C_L$  through n-transistor as follows



By making similar assumptions we can write for fall-time estimation,

$$\tau_f \doteq \frac{3C_L}{\beta_n V_{DD}}$$

From the above two estimations we can deduce that

$$\frac{\tau_r}{\tau_f} = \frac{\beta_n}{\beta_p}$$

We know that  $\mu_n = 2.5 \mu_p$

$$\beta_n \doteq 2.5\beta_p$$

and hence

So that the rise time is slower by a factor of 2.5 when using minimum size devices for both n & p.

- In order to achieve symmetrical operation using minimum channel length we need to make  $W_p = 2.5 W_n$ .
- For minimum size lambda based geometries this would result in the inverter having an input capacitance of

$$1 \square c_g (\text{n-device}) + 2.5 \square c_g (\text{p-device}) = 3.5 \square c_g$$

From the above equations we can conclude that

1.  $\tau_r$  and  $\tau_f$  are proportional to  $1/V_{DD}$
2.  $\tau_r$  and  $\tau_f$  are proportional to  $C_L$
3.  $\tau_r = 2.5\tau_f$  for equal n and p- transistor geometries.

### Driving Large capacitive loads

when signals are propagated from the chip to off chip destinations we can face problems to drive large capacitive loads. Generally off chip capacitances may be several orders higher than on chip  $c_g$  values.

$$C_L \geq 10^4 c_g$$

Where  $C_L$  denotes offchip load. The capacitances which of this order must be driven through low resistances, otherwise excessively long delays will occur. Large capacitance is presented at the input, which in turn slows down the rate of change of voltage at input.

### Cascaded Inverters as drivers

Inverters to drive large capacitive loads must be present low pull-up and pull down resistance. For MOS

circuits low resistance values imply low L:W ratio (since  $R_s = \frac{\rho L}{tw}$ ). Since length L cannot be reduced below the minimum feature size, the channels must be made very wide to reduce resistance value. Consider N cascaded inverters as on increasing the width factor of 'f' than the previous stage as shown in

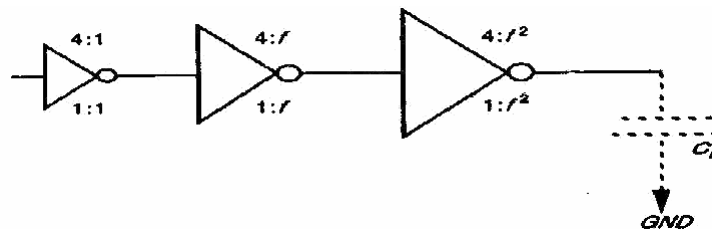


figure.

As the width factor increases, the capacitive load presented at the inverter input increases and the area occupied increases also. It is observed that as the width increases, the number N of stages are decreased to drive a particular value of  $C_L$ . Thus with large f(width), N decreases but delay per stage increases for 4:1 nMOS inverters.

$$\begin{aligned} \text{Delay per stage} &= ft \text{ for } \Delta V_{in} \\ &= 4ft \text{ for } \Delta V_{in} \end{aligned}$$

Where  $\Delta V_{in}$  indicates logic 0 to 1 transition and

$\Delta V_{in}$  indicates logic 1 to 0 transition of  $V_{in}$

Toal delay per nMOS pair =  $4ft$

Similarly delay per CMOS pair =  $7ft$ .

### Calculation for time delay

Let us assume  $y = C_L/c_g = f^N$

Determine the value of f which will minimize the overall delay for a given value of y. Apply logarithms on both sides in the above equation

$$\ln(y) = \ln(f^N)$$

$$\ln(y) = N \ln(f)$$

$$N = \ln(y)/\ln(f)$$

**For N even**

$$\begin{aligned} \text{Total delay} &= N/2 \cdot 5ft \\ &= 2.5 Nft \text{ (nMOS)} \end{aligned}$$

$$\begin{aligned} \text{(Or) toal delay} &= N/2 \cdot 7ft \\ &= 3.5Nft \text{ (CMOS)} \end{aligned}$$

From above relations, we can write

Delay  $\propto N\tau$

$$= \ln(y)/\ln(f) \cdot \tau$$

It can be shown that total delay is minimized if  $f$  assumes the value of  $e$  for both CMOS and nMOS inverters.

Assume  $f = e \Rightarrow N = \ln(y)/\ln(e)$

$$N = \ln(y)$$

Overall delay  $t_d \propto N$  even  $t_d = 2.5 eN\tau$  (nMOS)

(or)  $t_d = 3.5 eN\tau$  (CMOS)

$\hat{I}$  N odd  $t_d = [2.5(N-1) + 1] \tau$  (nMOS)

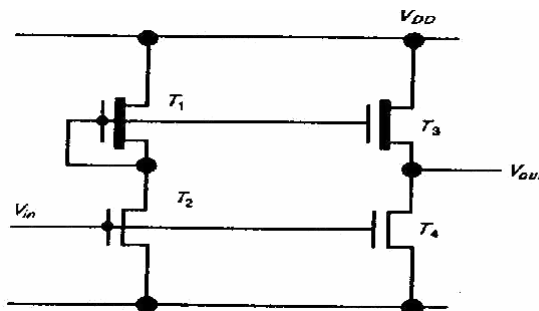
$t_d = [3.5(N-1) + 2] \tau$  (CMOS) (for logical transition 0 to 1)

(or)  $t_d = [2.5(N-1) + 4] \tau$  (nMOS)

$t_d = [3.5(N-1) + 5] \tau$  (CMOS) (for logical transition 1 to 0)

**Super buffers**

Generally the pull-up and the pull down transistors are not equally capable to drive capacitive loads. This asymmetry is avoided in super buffers. Basically, a super buffer is a symmetric inverting or non inverting driver that can supply (or) remove large currents and is nearly symmetrical in its ability to drive capacitive load. It can switch large capacitive loads than an inverter. An inverting type nMOS super buffer as shown in figure.



•Consider a positive going (0 to 1) transition at input  $V_{in}$  turns ON the inverter formed by  $T_1$  and  $T_2$ .

•With a small delay, the gate of  $T_3$  is pulled down to 0 volts. Thus, device  $T_3$  is cut off. Since gate of  $T_4$  is connected to  $V_{in}$ , it is turned ON and the output is pulled down very fast.

For the opposite transition of  $V_{in}$  (1 to 0),  $V_{in}$  drops to 0 volts. The gate of transistor  $T_3$  is allowed to rise to  $V_{DD}$  quickly.

•Simultaneously the low  $V_{in}$  turns off  $T_4$  very fast. This makes  $T_3$  to conduct with its gate voltage approximately equal to  $V_{DD}$ .

•This gate voltage is twice the average voltage that would appear if the gate was connected to the source as in the conventional nMOS inverter.

Now as  $I_{ds} \propto V_{gs}$ , doubling the effective  $V_{gs}$  increases the current and there by reduces the delay in charging at the load capacitor of the output. The result is more symmetrical transition.

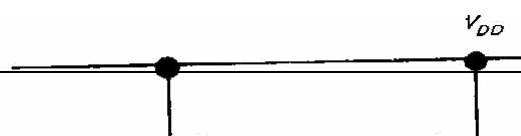
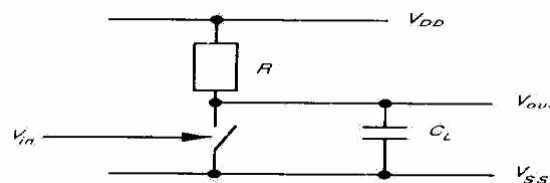


Figure shows the non-inverting nMOS super buffer where the structures fabricated in 5 $\mu$ m technology are capable of driving capacitance of 2pF with a rise time of 5nsec.

### BiCMOS drivers

1. In BiCMOS technology we use bipolar transistor drivers as the output stage of inverter and logic gate circuits.
2. In bipolar transistors, there is an exponential dependence of the collector (output) current on the base to emitter (input) voltage  $V_{be}$ .
3. Hence, the bipolar transistors can be operated with much smaller input voltage swings than MOS transistors and still switch large current.
4. Another consideration in bipolar devices is that the temperature effect on input voltage  $V_{be}$ .
5. In bipolar transistor,  $V_{be}$  is logarithmically dependent on collector current  $I_C$  and also other parameters such as base width, doping level, electron mobility.
6. Now, the temperature differences across an IC are not very high. Thus the  $V_{be}$  values of the bipolar devices spread over the chip remain same and do not differ by more than a few milli volts.

The switching performance of a bipolar transistor driving a capacitive load can be analyzed to begin with the help of equivalent circuit as shown in figure.



The time  $\Delta t$  required to change the output voltage  $V_{out}$  by an amount equal to the input voltage is

$$\Delta t = C_L / g_m$$

Where,

$C_L$  is the load capacitance

$g_m$  is the trans conductance of the bipolar transistor.

The value of  $\Delta t$  is small because the trans conductance of the bipolar transistors is relatively high.

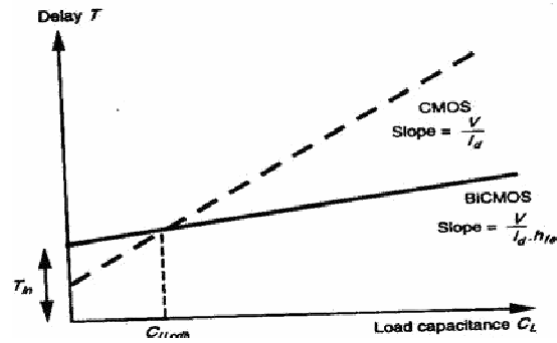
There are two main components which reveals the delay due to the bipolar transistors are  $T_{in}$  and  $T_L$ .

- $T_{in}$  is the time required to first charge the base emitter junction of the bipolar (npn) transistor. This time is typically 2ns for the BiCMOS transistor base driver.

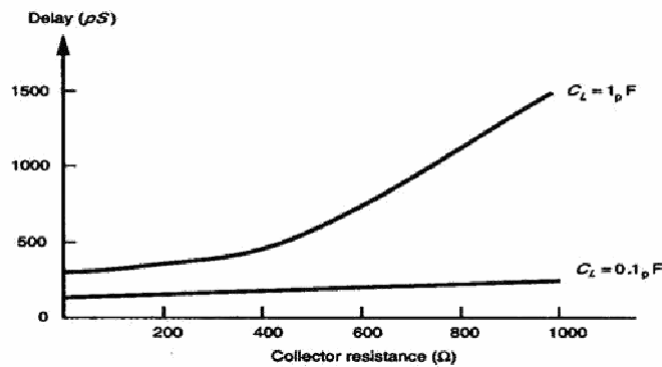
- For the CMOS driver the time required to charge the input gate capacitance is 1ns.

- $T_L$  is the time required to charge the output load capacitance .

- The combined effect of  $T_{in}$  and  $T_L$  is represented as shown in figure.



- Delay of BiCMOS inverter can be described by
- Delay for BiCMOS inverter is reduced by a factor of  $h_{fe}$  as compared with a CMOS inverter.
- In Bipolar transistors while considering delay another significant parameter is collector resistance  $R_c$  through which the charging current for  $C_L$  flows.
- For a high value of  $R_c$ , there is a long propagation delay through the transistor when charging a capacitive load.
- Figure shows the typical delay values at two values of  $C_L$  as follows.



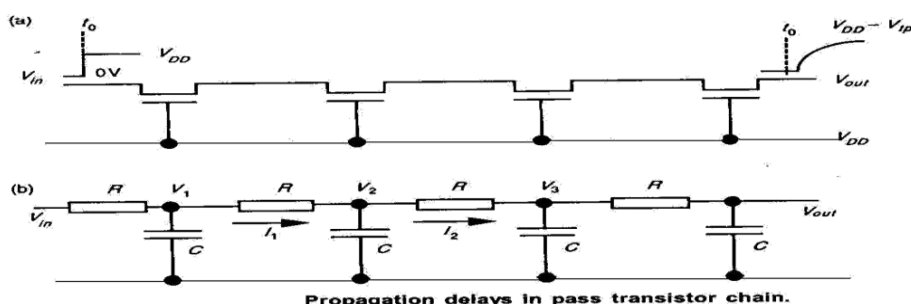
The devices thus have high  $\beta$ , high  $g_m$ , high  $h_{fe}$  and low  $R_c$ . The presence of such efficient and advantageous devices on chip offers a great deal of scope and freedom to the VLSI designer.

### Propagation delays

Propagation delay is the delay in the propagation of the signal created by the change of logical status at the input to create same change at the output.

#### (i) Cascaded pass transistors

Figure shows a chain of four pass transistors driving a capacitive load  $C_L$ . All the gates are supplied by  $V_{DD}$  so that a signal can propagate to the output. The lumped RC equivalent circuit is shown in figure, where each transistor is modeled by a series resistance and capacitance representing the gate-to-channel capacitance and stray capacitances. The minimum value of  $R$  is the turned ON resistance of each enhancement mode pass transistor.



The current through the capacitance at the node with voltage  $V_2$  is

$$C \quad (dV_2 / dt) \approx C \cdot \Delta V_2 / \Delta t$$

The current entering at this node is  $I_1 = (V_1 - V_2)/R$  and the current leaving from this node is  $I_2 = (V_2 - V_3)/R$ . By applying KCL at this node

$$I_C = I_1 - I_2$$

$$C \cdot \Delta V_2 / \Delta t = I_1 - I_2 = ((V_1 - V_2) - (V_2 - V_3)) / R$$

As the number of sections in the network increases, the circuit parameters become distributed.

Assume that  $R$  and  $C$  as the resistance per unit length and the capacitance per unit length respectively.

$$C \Delta V_2 / \Delta t = \Delta(\Delta V_2) / R \Delta X$$

Where  $x$  is the distance along the network from the input.

$$RC \, dv/dt = d/dx \cdot (dv/dx) = d^2V/dx^2$$

The propagation time  $\tau_p$  from a signal to propagate a distance  $x$  is

$$\tau_p \propto X^2$$

By simplifying the analysis if all sheet resistance, gate-to-channel capacitance  $R_s$  and  $C_g$  are lumped together

$$R_{\text{total}} = nr R_s$$

$$C_{\text{total}} = nc C_g$$

Where  $r$  gives relative resistance per section in terms of  $R_s$  and  $c$  gives relative capacitance per section in terms of  $C_g$ . Then the overall delay for  $n$  sections is given by

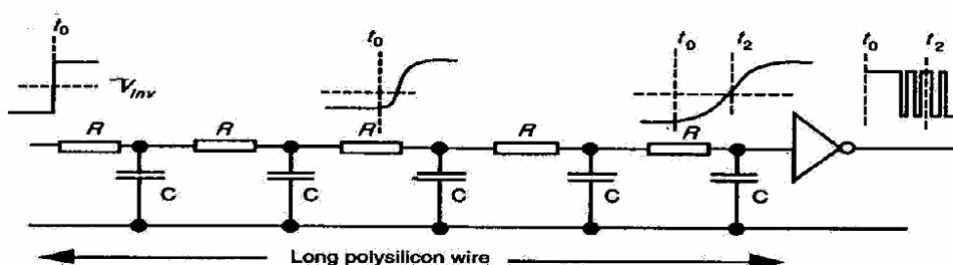
$$\tau_p = n^2 rc(\tau)$$

It can be shown that the signal delay in a section containing  $N$  identical pass transistors driving a matched load ( $C_L = C_g$ ) is  $\tau_p = 0.7 \cdot N(N+1)/2 \cdot RCL$

For large value of  $N$ , the quantity  $(N + 1)$  can be replaced by  $N$ . Since the delay increases with  $N$ , the number of pass transistors is restricted to 4. A cascade of more pass transistors will produce a very slow circuit and the signal needs to be restored by an inverter after every three (or) four pass transistor.

### Design of long polysilicon wires

Long polysilicon wires also contribute distributed series  $R$  and  $C$  as was the case for cascaded pass transistors and in consequence signal propagation is slowed down. This would also be the case for wires in diffusion where the value of  $C$  may be quite high, and for this reason the designer is discouraged from running signals in diffusion except over very short distances.



For long polysilicon runs, the use of buffers is recommended. In general, the use of buffers to drive long polysilicon runs has two desirable effects. First, the signal propagation is speeded up and second there is a reduction in sensitivity to noise. In the diagram the slow rise-time of the signal at the input of the inverter means that the input voltage spends a relatively long time in the vicinity of  $V_{inv}$  so that small disturbances due to noise will switch the inverter state between '0' and '1' as shown at the output point.

Thus, it is essential that long polysilicon wires be driven by suitable buffers to guard against the effects of noise and to speed up the rise-time of propagated signal edges.

### **Wiring capacitances**

The significant sources of capacitance which contribute to the overall wiring capacitance are as follows

#### **(i) Fringing fields**

Capacitance due to fringing field effects can be a major component of the overall capacitance of interconnect wires. For fine line metallization, the value of fringing field capacitance ( $C_{ff}$ ) can be of the same order as that of the area capacitance. Thus,  $C_{ff}$  should be taken into account if accurate prediction of performance is needed.

$$C_{ff} = \epsilon_{SiO_2} \epsilon_0 l \left[ \frac{\pi}{\ln \left\{ 1 + \frac{2d}{t} \left( 1 + \sqrt{1 + \frac{t}{d}} \right) \right\}} - \frac{t}{4d} \right]$$

Where  $l$  = wire length

$t$  = thickness of wire

$d$  = wire to substrate separation.

Then, total wire capacitance

$$C_w = C_{area} + C_{ff}$$

#### **(ii) Interlayer capacitances**

From the definition of capacitance itself, it can be said that there exists a capacitance between the layers due to parallel plate effects. This capacitance will depend upon the layout i.e., where the layers cross or whether one layer underlies another etc., by the knowledge of these capacitances, the accuracy of circuit modeling and delay calculations will be improved. It can be readily calculated for regular structures.

#### **(iii) peripheral capacitance**

1. The source and drain p-diffusion regions form junctions with the n-substrate (or n-well) at well defined and uniform depths.
2. Similarly, the source and drain n-diffusion regions form junctions with p-substrate (or p-well) at well defined and uniform depths.
3. Hence, for diffusion regions, each diode thus formed has associated a peripheral (side wall) capacitance with it.
4. As a whole the peripheral capacitance,  $C_p$  will be the order of pF/unit length. So its value will be greater than  $C_{area}$  of the diffusion region to substrate.

$C_p$  increases with reduction in source or drain area.

Total diffusion capacitance is  $C_{diff} = C_{area} + C_p$

However, as the n and p-active regions are formed by impure implants at the surface of the silicon in case of orbit processes, they have negligible depth. Hence  $C_p$  is quite negligible in them.

Typical values are given in tabular form

Diffusion capacitance	Typical values		
	5 $\mu$ m	2 $\mu$ m	1.2 $\mu$ m
Area C ( $C_{area}$ )	$1.0 \times 10^{-4}$ pF/ $\mu$ m <sup>2</sup>	$1.75 \times 10^{-4}$ pF/ $\mu$ m <sup>2</sup>	$3.75 \times 10^{-4}$ pF/ $\mu$ m <sup>2</sup>
Periphery ( $C_{periph}$ )	$8.0 \times 10^{-4}$ pF/ $\mu$ m <sup>2</sup>	Negligible (assuming implanted regions of negligible depth)	negligible

### Fan – in and Fan-out

**Fan-in:** The number of inputs to a gate is called as fan - in.

**Fan-out:** The maximum number of similar gates that a gate can drive while remaining within the guaranteed specifications is called as fan-out.

#### Effects of Fan-in and Fan-out on propagation delay:

- An additional input to a CMOS logic gate requires an additional nMOS and pMOS i.e., two additional transistors, while in case of other MOS logic gates, it requires one additional transistor.
- In CMOS logic gates, due to these additional transistors, not only the chip area but also the total effective capacitance per gate also increased and hence propagation delay increases.
- Some of the increase in propagation delay time can be compensated by the size-scaling method.
- By increasing the size of the device, its current driving capability can be preserved.
- Due to increase in both of inputs and devices size, the capacitance increases, Hence propagation delay will still increase with fan-in.
- An increase in the number of outputs of a logic gate directly adds to its load capacitances. Hence, the propagation delay increases with fan-out.

Fan-In = Number of inputs to a logic gate

– 4 input NAND has a FI = 4

– 2 input NOR has a FI = 2, etc. (See Fig. a below.)

• Fan-Out (FO)= Number of gate inputs which are driven by a particular gate output

– FO = 4 in Fig. b below shows an output wire feeding an input on four different logic gates

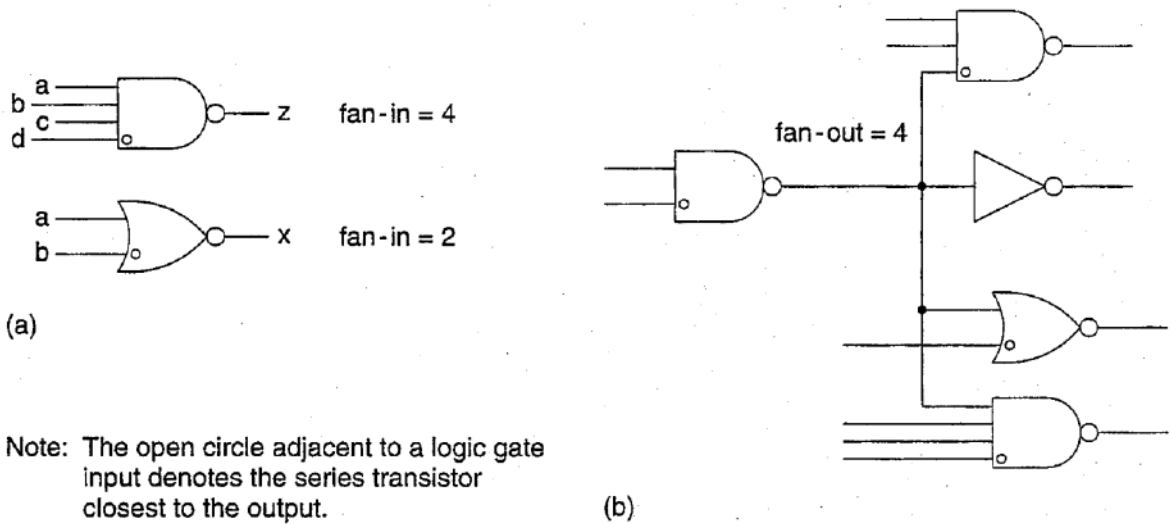
• The circuit delay of a gate is a function of both the Fan-In and the Fan-Out.

Ex. m-input NAND:  $t_{dr} = (R_p/n)(mC_d + C_r + kC_g)$

=  $t_{internal-r} + k t_{output-r}$

- where n = width multiplier, m = fan-in, k = fan-out,  $R_p$  = resistance of min inverter P Tx,  $C_g$  = gate capacitance,  $C_d$  = source/drain capacitance,  $C_r$  = routing (wiring) capacitance.





### Choice of layers

The following are the constraints which must be considered for the proper choice of layers.

1. Since the polysilicon layer has relatively high specific resistance ( $R_s$ ), it should not be used for routing  $V_{DD}$  and  $V_{SS}$  (GND) except for small distances.
2.  $V_{DD}$  and GND ( $V_{SS}$ ) must be distributed only on metal layers, due to the consideration of  $R_s$  value.
3. The capacitive effects will also impose certain restrictions in the choice of layers as follows
  - (i) where fast signal lines are required, and in relation to signals on wiring which has relatively higher values of  $R_s$ .
  - (ii) The diffusion areas have higher values of capacitance to substrate and are harder to drive.
4. Over small equipotential regions, the signal on a wire can be treated as being identical at all points.
5. Within each region the propagation delay of the signal will comparably smaller than the gate delays and signal delays caused in a system connected by wires.

Thus the wires in a MOS system can be modeled as simple capacitors. This concept leads to the establishment of electrical rules (guidelines) for communication paths(wires) as given in tabular form.

Layer	Maximum length of communication wire		
	Lambda based 5 $\mu$ m	$\mu$ m based (2 $\mu$ m)	$\mu$ m – based(1.2 $\mu$ m)
Metal	Chip wide	Chip wide	Chip wide
Silicide	2,000 $\lambda$	NA	NA
Polysilicon	200 $\lambda$	400 $\mu$ m	250 $\mu$ m
Diffusion (active)	20 $\lambda$	100 $\mu$ m	60 $\mu$ m

### **3.6.7 APPLICATIONS OF CPLDs AND FPGAS**

CPLDs and FPGAs are used today in many diverse applications, such as consumer products like DVD players and high-end television sets, controller circuits for automobile factories and test equipment, Internet routers and high-speed network switches, and computer equipment like large tape and disk storage systems.

In a given design situation a CPLD may be chosen when the needed circuit is not very large, or when the device has to perform its function immediately upon application of power to the circuit. FPGAs are not a good choice for this latter case because, as we mentioned before, they are configured by volatile storage elements that lose their stored contents when the power is turned off. This property results in a delay before the FPGA chip can perform its function when turned on.

FPGAs are suitable for implementation of circuits over a large range of size, from about 1000 to more than a million equivalent logic gates. In addition to size a designer will consider other criteria, such as the needed speed of operation of a circuit, power dissipation constraints, and the cost of the chips. When FPGAs do not meet one or more of the requirements, the user may choose to create a custom-manufactured chip as described below.

## Introduction

Most digital functions can be divided into the following categories:

1. Datapath operators 2. Memory elements 3. Control structures 4. Special-purpose cells

- I/O
- Power distribution
- Clock generation and distribution
- Analog and RF

CMOS system design consists of partitioning the system into subsystems of the types listed above. Many options exist that make trade-offs between speed, density, programmability, ease of design, and other variables. This chapter addresses design options for common datapath operators. The next chapter addresses arrays, especially those used for memory. Control structures are most commonly coded in a hardware description language and synthesized.

Datapath operators benefit from the structured design principles of hierarchy, regularity, modularity, and locality. They may use  $N$  identical circuits to process  $N$ -bit data. Related data operators are placed physically adjacent to each other to reduce wire length and delay. Generally, data is arranged to flow in one direction, while control signals are introduced in a direction orthogonal to the dataflow.

Common datapath operators considered in this chapter include adders, one/zero detectors, comparators, counters, shifters, ALUs, and multipliers.

### 5.1 Shifters

Consider a direct MOS switch implementation of a 4X4 crossbar switch as shown in Fig. 5.1.

The arrangement is quite general and may be readily expanded to

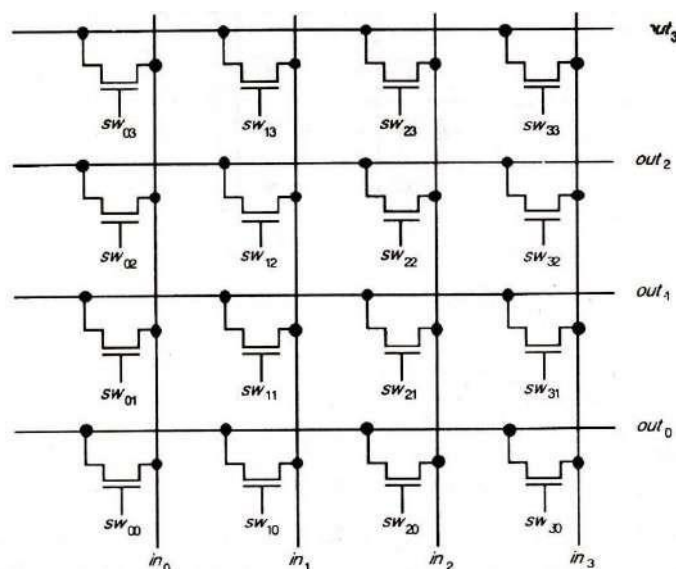


Figure 5.1: 4 x 4 crossbar switch.

accommodate  $n$ -bit inputs/outputs. In fact, this arrangement is an overkill in that any input line can be connected to any or all output lines—if all switches are closed, then all inputs are connected to all outputs in one glorious short circuit. Furthermore, 16 control signals ( $SW_{00}$ - $SW_{15}$ ), one for each transistor switch, must be provided to drive the crossbar switch, and such complexity is highly

undesirable. An adaption of this arrangement) Recognizes the fact that we can couple the switch gates together in groups of four (in this case) and also form four separate groups corresponding to shifts of zero, one, two, and three bits. The arrangement is readily adapted so that the inlines also run horizontally (to confirm the required strategy). The resulting arrangement is known as barrel shifter and a 4X4-bit barrel shifter circuit diagram is given in Fig. 5.2. The interbus switches have their gate inputs connected in staircase fashion in group of four and there are now four shift control inputs which must be mutually exclusive in active state. CMOS transmission gates may be used in place of the simple pass transistor switches if appropriate

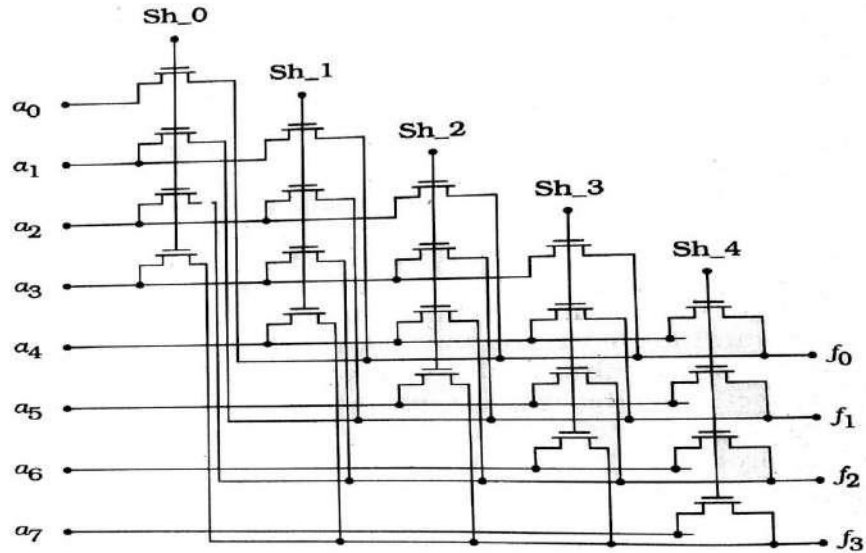


Figure 5.2: Barrel shifter

ADDERS: Addition is one of the basic operation perform in various processing like counting, multiplication and filtering. Adders can be implemented in various forms to suit different speed and density requirements. The truth table of a binary full adder is shown in Figure 5.3, along with some functions that will be of use during the discussion of adders. Adder inputs: A,

C	A	B	A.B (G)	A+B (P)	$A \oplus B$	SUM	CARRY
0	0	0	0	0	0	0	0
0	0	1	0	1	1	1	0
0	1	0	0	1	1	1	0
0	1	1	1	1	0	0	1
1	0	0	0	0	0	1	0
1	0	1	0	1	1	0	1
1	1	0	0	1	1	0	1
1	1	1	1	1	0	1	1

B

Carry input: SUM Carry output: CARRY

Generate signal:  $G(A \cdot B)$ ; occurs when CARRY is internally generated within adder

Propagate signal:  $P(A+B)$ ; when it is 1, C is passed to CARRY. In some adders

$A \oplus B$  is used as the P term because it may be reused to generate the sum term. 5.2.1 Single-Bit Adders

Probably the simplest approach to designing an adder is to implement gates to yield the required majority logic functions.

From the truth table find sum and carry

The direct implementation of the above equations is shown in Fig. 5.4 using the gate schematic and the transistors is shown in Fig. 5.5.

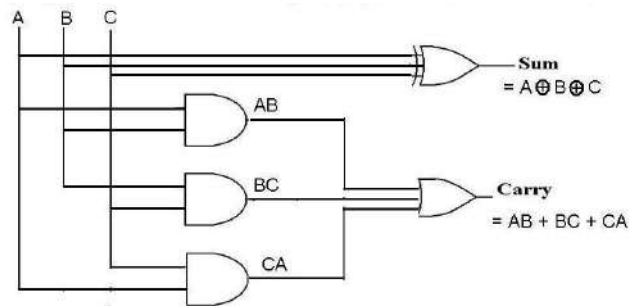


Figure 5.4: Logic gate implementation of 1-Bit adder

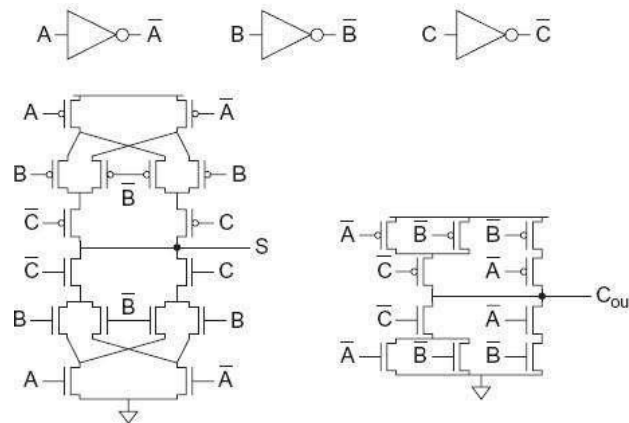


Figure 5.5: Transistor implementation of 1-Bit adder

The full adder of Fig. 5.5 employs 32 transistors (6 for the inverters, 10 for the carry circuit, and 16 for the 3-input XOR). A more compact design is based on the observation that S can be factored to reuse the CARRY term as follows:

$$\text{SUM} = A.B.C + (A + B + C).CARRY$$

$$= A.B.C + (A + B + C).A.B + C.(A + B)$$

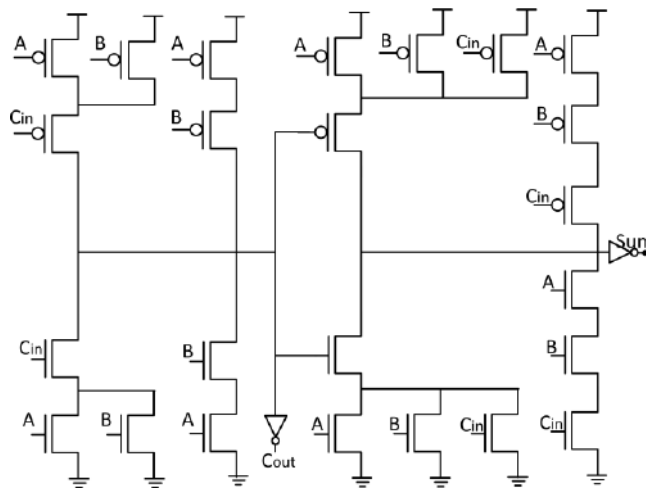


Figure 5.6: Transistor implementation of 1-bit adder

Such a design is shown at the transistor levels in Figure 5.6 and uses only 28 transistors. Note that the pMOS network is complement to the nMOS network.

Here  $C_{in}=C$

### 5.1.2 n-Bit Parallel Adder or Ripple Carry Adder

A ripple carry adder is a digital circuit that produces the arithmetic sum of two binary numbers. It can be constructed with full adders connected in cascaded, with the carry output from each full adder connected to the carry input of the next full adder in the chain. Figure 5.7 shows the interconnection of four full adder (FA) circuits to provide a 4-bit ripple carry adder. Notice from Figure 5.7 that the input is from the right side because the first cell traditionally represents the least significant bit (LSB). Bits  $a_0$  and  $b_0$  in the figure represent the least significant bits of the numbers to be added. The sum output is represented by the bits  $s_0-s_3$ .

### 5.1.3 Carry lookahead adder (CLA)

The carry lookahead adder (CLA) solves the carry delay problem by calculating the carry signals in advance, based on the input signals. It is based on the fact that a carry signal will be generated in two cases: (1) when both bits  $a_i$  and  $b_i$  are 1, or

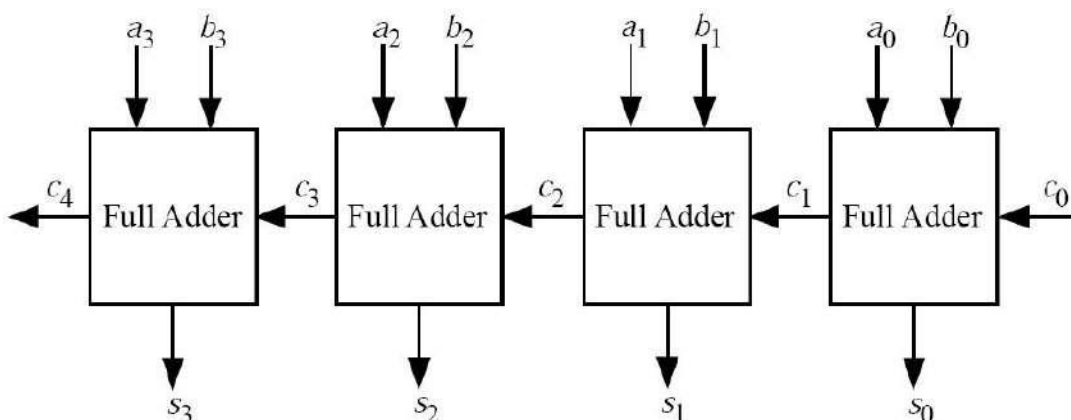


Figure 5.7: 4-bit ripple carry adder

(2) when one of the two bits is 1 and the carry-in is 1. Thus, one can write,

$$c_{i+1} = a_i \cdot b_i + (a_i \oplus b_i) \cdot c_i$$

$$s_i = (a_i \oplus b_i) \oplus c_i$$

The above two equations can be written in terms of two new signals  $P_i$  and  $G_i$ , which are shown in Figure 5.8:

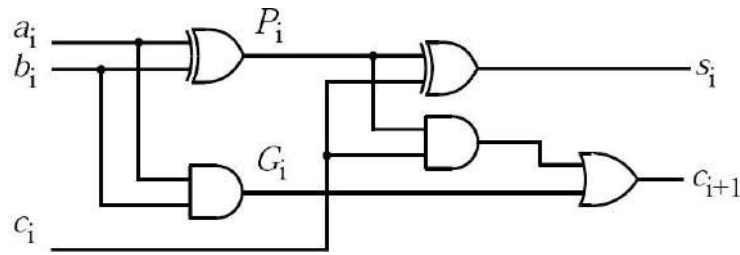


Figure 5.8: Full adder stage at  $i$  with  $P_i$  and  $G_i$  shown

Where  $c_{i+1} = G_i + P_i.c_i$

$$s_i = P_i \oplus c_i$$

$$G_i = a_i.b_i$$

$$P_i = (a_i \oplus b_i)$$

$P_i$  and  $G_i$  are called carry propagate and carry generate terms, respectively. Notice that the generate and propagate terms only depend on the input bits and thus will be valid after one and two gate delay, respectively. If one uses the above expression to calculate the carry signals, one does not need to wait for the carry to ripple through all the previous stages to find its proper value. Let's apply this to a 4-bit adder to make it clear.

Putting  $i = 0, 1, 2, 3$  in  $c_{i+1} = G_i + P_i.c_i$  we get

$$c_1 = G_0 + P_0.c_0$$

$$c_2 = G_1 + P_1.G_0 + P_1.P_0.c_0$$

$$c_3 = G_2 + P_2.G_1 + P_2.P_1.G_0 + P_2.P_1.P_0.c_0$$

$$c_4 = G_3 + P_3.G_2 + P_3.P_2.G_1 + P_3.P_2.P_1.G_0 + P_3.P_2.P_1.P_0.c_0$$

Notice that the carry-out bit,  $c_{i+1}$ , of the last stage will be available after four delays: two gate delays to calculate the propagate signals and two delays as a result of the gates required to implement Equation  $c_4$ .

Figure 5.9 shows that a 4-bit CLA is built using gates to generate the  $P_i$  and  $G_i$  and signals and a logic block to generate the carry out signals according to Equations  $c_1$  to  $c_4$ . Logic gate and transistor level implementation of carry bits are shown in Figure 5.10 The disadvantage of CLA is that the carry logic block gets very complicated for more than 4-bits. For that reason, CLAs are usually implemented as 4-bit modules and are used in a hierarchical structure to realize adders that have multiples of 4-bits.

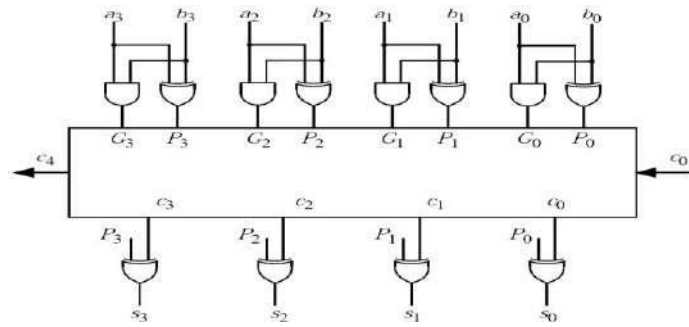
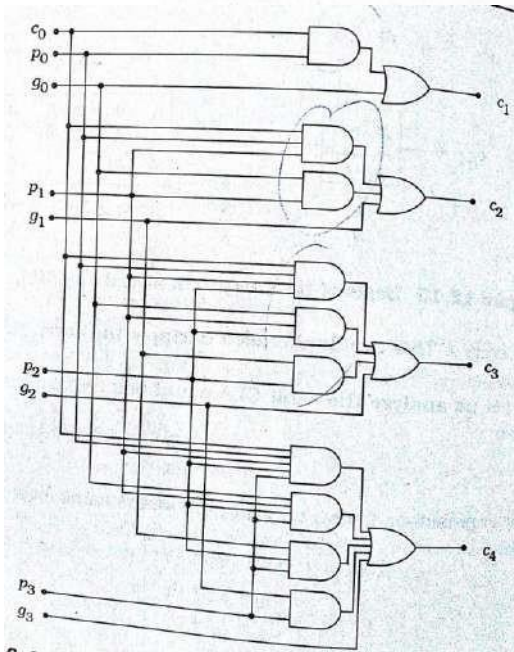
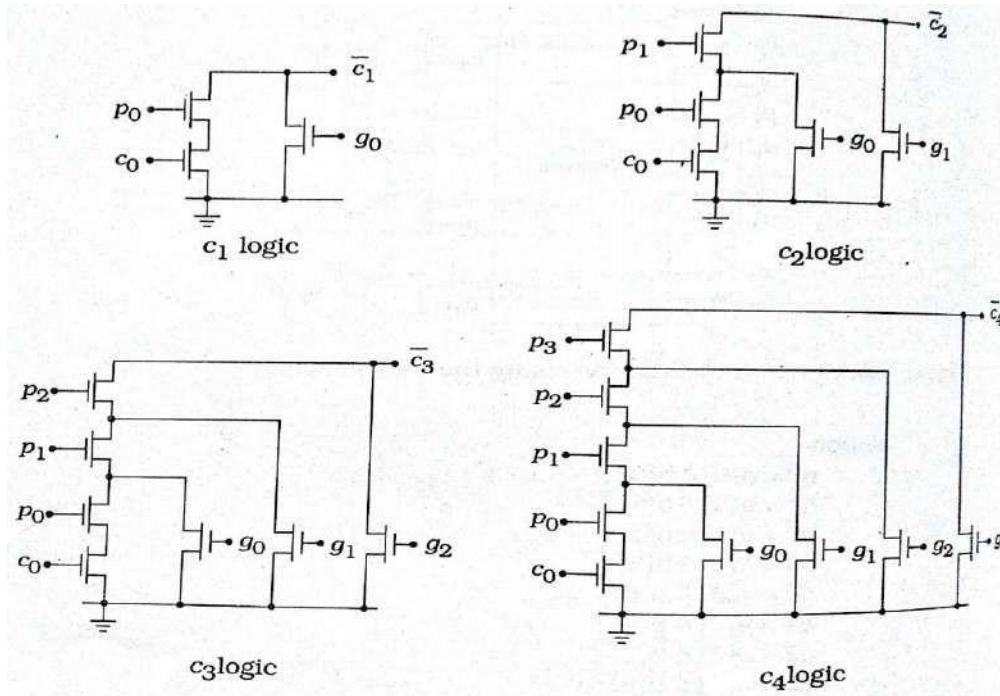
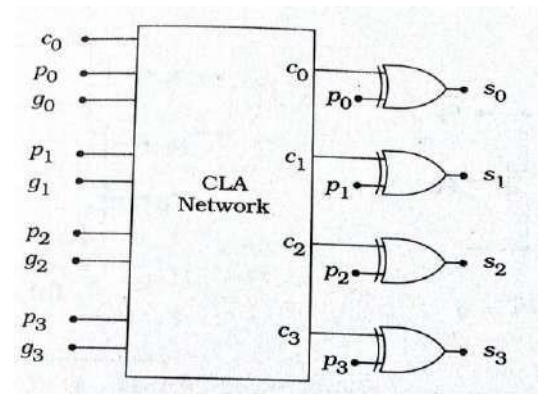


Figure 5.9: 4-bit carry lookahead adder implementation in detail



Logic network for 4-bit CLA carry bits

(c) nFET logic arrays for the CLS terms



(b) Sum calculation using CLA network

Figure 5.10: Carry structures of CLA



### 5.1.4 Manchester carry chain

This implementation can be very performant (20 transistors) depending on the way the XOR function is built. The carry propagation of the carry is controlled by the output of the XOR gate. The generation of the carry is directly made by the function at the bottom. When both input signals are 1, then the inverse output carry is 0. In the schematic of Figure 5.11, the carry passes through a complete

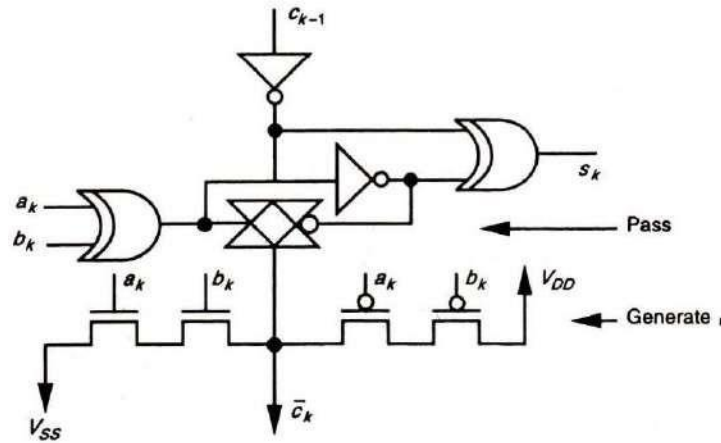


Figure 5.11: An adder element based on the pass/generate concept.

transmission gate. If the carry path is precharged to  $V_{DD}$ , the transmission gate is then reduced to a simple NMOS transistor. In the same way the PMOS transistors of the carry generation is removed. One gets a Manchester cell.

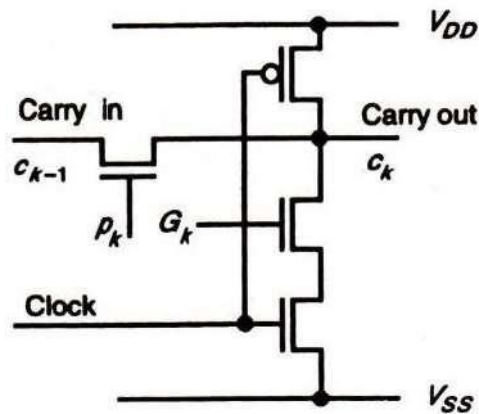


Figure 5.12: Manchester cell

The Manchester cell is very fast, but a large set of such cascaded cells would be slow. This is due to the distributed RC effect and the body effect making the propagation time grow with the square of the number of cells. Practically, an inverter is added every four cells, like in Figure 5.13.

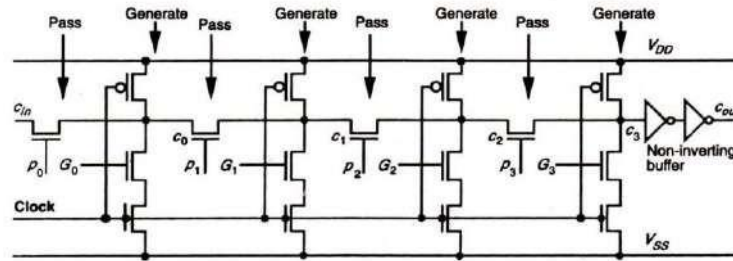


Figure 5.13: Cascaded Manchester carry-chain elements with buffering

## 5.2 Multipliers

In many digital signal processing operations - such as correlations, convolution, filtering, and frequency analysis - one needs to perform multiplication. The most basic form of multiplication consists of forming the product of two positive binary numbers. This may be accomplished through the traditional technique of successive additions and shifts, in which each addition is conditional on one of the multiplier bits. Here is an example.

$$\begin{array}{r}
 \text{multiplicand} \quad 1100 : 12_{10} \\
 \text{multiplier} \quad 0101 : 5_{10} \\
 \hline
 1100 \\
 0000 \\
 1100 \\
 0000 \\
 \hline
 0111100 : 60_{10}
 \end{array}$$

Figure 5.5: 4-bit multiplication

The multiplication process may be viewed to consist of the following two steps:

- ⤴ Evaluation of partial products.
- ⤴ Accumulation of the shifted partial products.

It should be noted that binary multiplication is equivalent to a logical AND operation. Thus evaluation of partial products consists of the logical ANDing of the multiplicand and the relevant multiplier bit. Each column of partial products must then be added and, if necessary, any carry values passed to the next column.

There are a number of techniques that may be used to perform multiplication. In general, the choice is based on factors such as speed, throughput, numerical accuracy, and area. As a rule, multipliers may be classified by the format in which data words are accessed, namely:-

### Parallel form 1.4.1 Array Multiplication

A parallel multiplier is based on the observation that partial products in the multiplication process may be independently computed in parallel. For example, consider the unsigned binary integers X and Y.

$$X = \sum_{i=0}^{m-1} X_i \cdot 2^i \quad \text{and} \quad Y = \sum_{j=0}^{n-1} Y_j \cdot 2^j$$

The product is found by

$$\begin{aligned} P = X \times Y &= \left( \sum_{i=0}^{m-1} X_i \cdot 2^i \right) \times \left( \sum_{j=0}^{n-1} Y_j \cdot 2^j \right) \\ &= \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} (X_i \cdot Y_j) \cdot 2^{i+j} \\ &= \sum_{k=0}^{m+n-1} P_k \cdot 2^k \end{aligned}$$

Thus  $P_k$  are the partial product terms called summands. There are  $mn$  summands, which are produced in parallel by a set of  $mn$  AND gates.

For 4-bit numbers, the expression above may be expanded as in the table below.

				X3	X2	X1	X0	Multiplicand
				Y3	Y2	Y1	Y0	Multiplier
				X3Y0	X2Y0	X1Y0	X0Y0	
			X3Y1	X2Y1	X1Y1	X0Y1		
		X3Y2	X2Y2	X1Y2	X0Y2			
	X3Y3	X2Y3	X1Y3	X0Y3				
P7	P6	P5	P4	P3	P2	P1	P0	Product

Figure 5.6

An  $n \times n$  multiplier requires

$(n - 1)^2$  full adders,

$n - 1$  half adders, and

$n^2$  AND gates.

The worst-case delay associated with such a multiplier is  $(2n + 1)t_g$ , where  $t_g$  is the worst-case adder delay.

Cell shown in Figure 5.16 is a cell that may be used to construct a parallel multiplier.

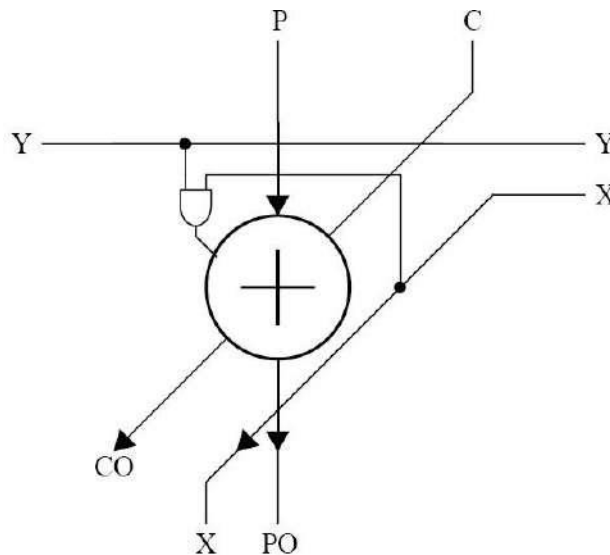


Figure 5.16: Basic cell to construct a parallel multiplier

The  $X_i$  term is propagated diagonally from top right to bottom left, while the  $y_j$  term is propagated horizontally. Incoming partial products enter at the top. Incoming CARRY IN values enter at the top right of the cell. The bit-wise AND is performed in the cell, and the SUM is passed to the next cell below. The CARRY OUT is passed to the bottom left of the cell. Figure 5.17 depicts the multiplier array with the partial products enumerated.

The Multiplier can be drawn as a square array, as shown here, Figure 5.18 is the most convenient for implementation. In this version the degeneration of the first two rows of the multiplier are shown. The first row of the multiplier adders has been replaced with AND gates while the second row employs half-adders rather than full adders. This optimization might not be done if a completely regular multiplier were required (i.e. one array cell). In this case the appropriate inputs to the first and second row would be connected to ground, as shown in the previous slide. An adder with equal carry and sum propagation times is advantageous, because the worst-case multiply time depends on both paths.

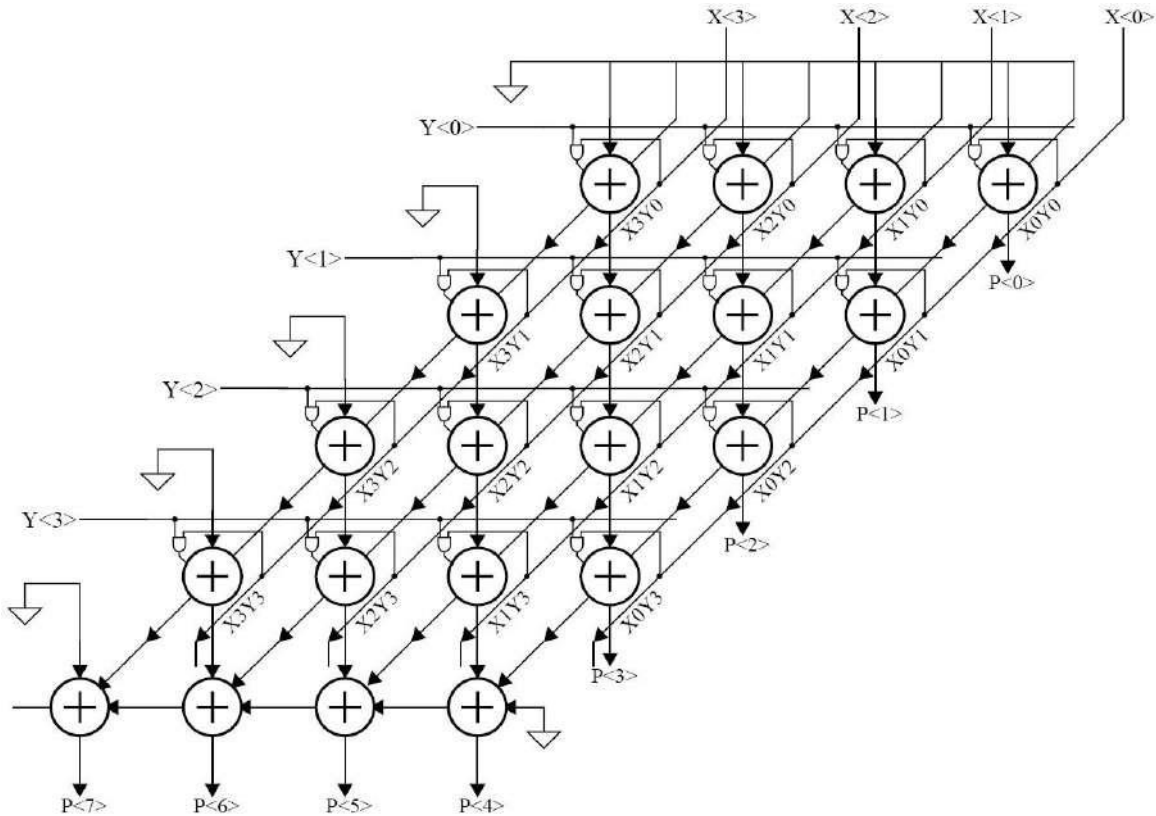


Figure 5.17: Array multiplier

### Wallace Tree Multiplication

If the truth table for an adder, is examined, it may be seen that an adder is in effect a “one’s counter” that counts the number of 1’s on the A, B, and C inputs and encodes them on the SUM and CARRY outputs.

A 1-bit adder provides a 3:2 (3 inputs, 2 outputs) compression in the number of bits. The addition of partial products in a column of an array multiplier may be thought of as totaling up the number of 1’s in that column, with any carry being passed to the next column to the left.

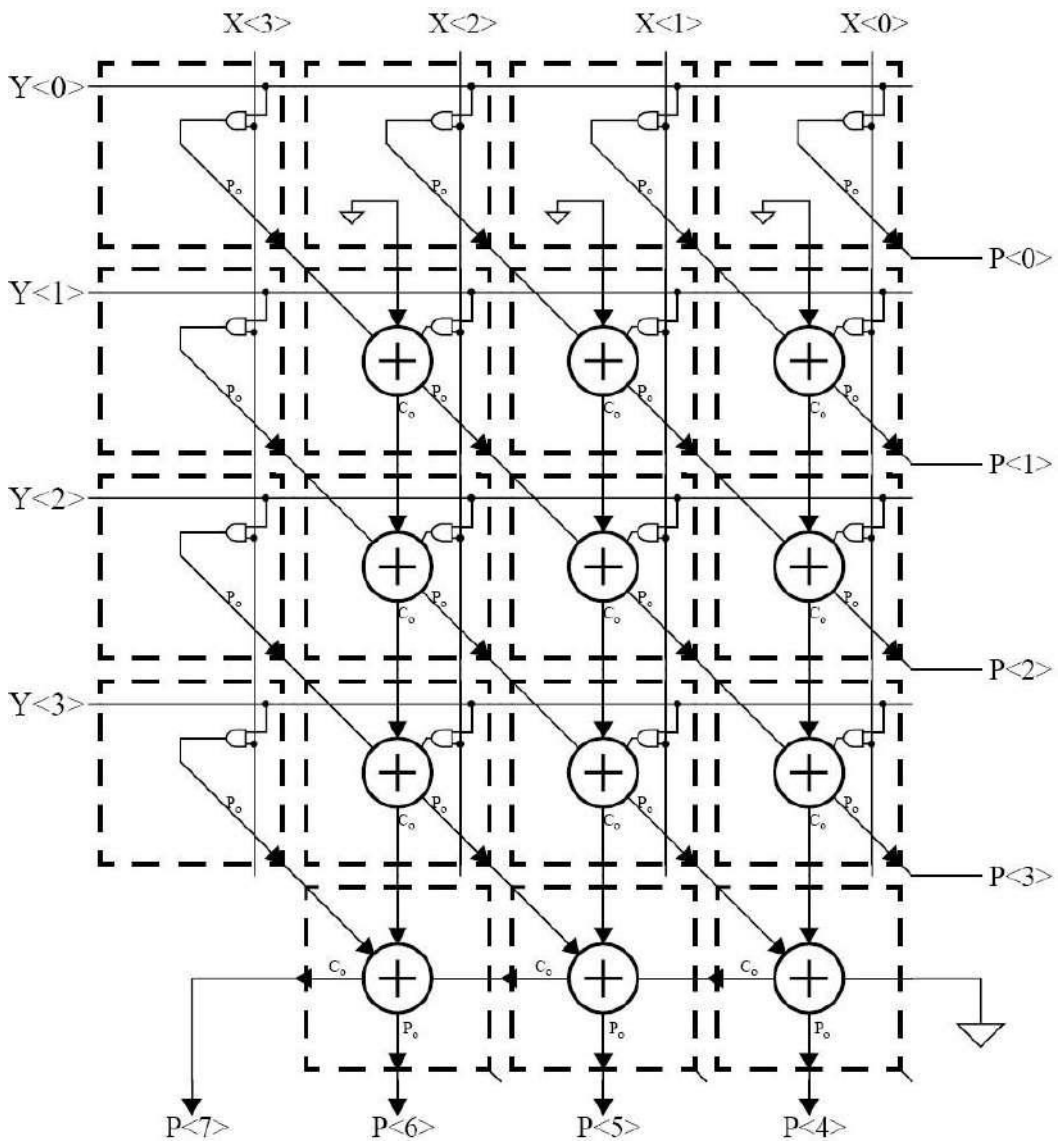


Figure 5.18: Most convenient way for implementation of array multiplier

ABC	Carry/Sum	Number of 1's
000	00	0
001	10	1
010	10	1
011	01	2
100	01	1
101	10	2
110	10	2
111	11	3

Figure 5.7

Example for implementation of 4x4 multiplier(4-bit) using Wallace Tree Multi- plication methods

				X3	X2	X1	X0	Multiplicand
				Y3	Y2	Y1	Y0	Multiplier
				X3Y0	X2Y0	X1Y0	X0Y0	
			X3Y1	X2Y1	X1Y1	X0Y1		
		X3Y2	X2Y2	X1Y2	X0Y2			
	X3Y3	X2Y3	X1Y3	X0Y3				
P7	P6	P5	P4	P3	P2	P1	P0	Product

Figure 5.7: Table to find product terms

Considering the product P3, it may be seen that it requires the summation of four partial products and a possible column carry from the summation of P2.

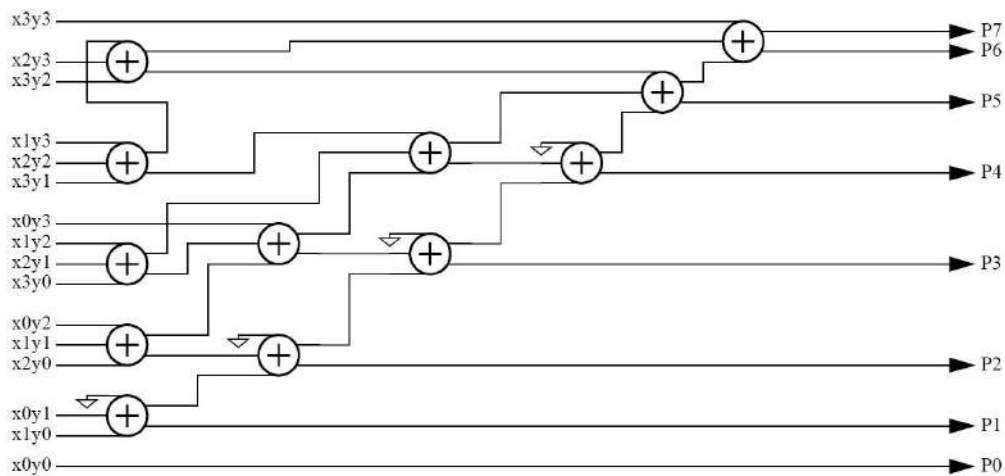


Figure 5.8: Wallace Tree Multiplication for 4-bits

Example for implementation of 6X6 multiplier(4-bit) using Wallace Tree Multiplication methods  
 Consider the 6 x 6 multiplication table shown below. Considering the product P5, it may be seen that it requires the summation of six partial products and a possible column carry from the summation of P4. Here we can see the adders required in a multiplier based on this style of addition. The adders have been arranged vertically into ranks that indicate the time at which the adder output becomes available. While this small example shows the general Wallace addition technique, it does not show the real speed advantage of a Wallace tree. There is an identifiable “array part”, and a CPA part, which is at the top right. While this has been shown as a ripple-carry adder, any fast CPA

						X5	X4	X3	X2	X1	X0	Multiplicand
						Y5	Y4	Y3	Y2	Y1	Y0	Multiplier
						X5Y0	X4Y0	X3Y0	X2Y0	X1Y0	X0Y0	
					X5Y1	X4Y1	X3Y1	X2Y1	X1Y1	X0Y1		
				X5Y2	X4Y2	X3Y2	X2Y2	X1Y2	X0Y2			
			X5Y3	X4Y3	X3Y3	X2Y3	X1Y3	X0Y3				
		X5Y4	X4Y4	X3Y4	X2Y4	X1Y4	X0Y4					
	X5Y5	X4Y5	X3Y5	X2Y5	X1Y5	X0Y5						
P11	P10	P9	P8	P7	P6	P5	P4	P3	P2	P1	P0	Product

Figure 1.22: 6 x 6 multiplication table

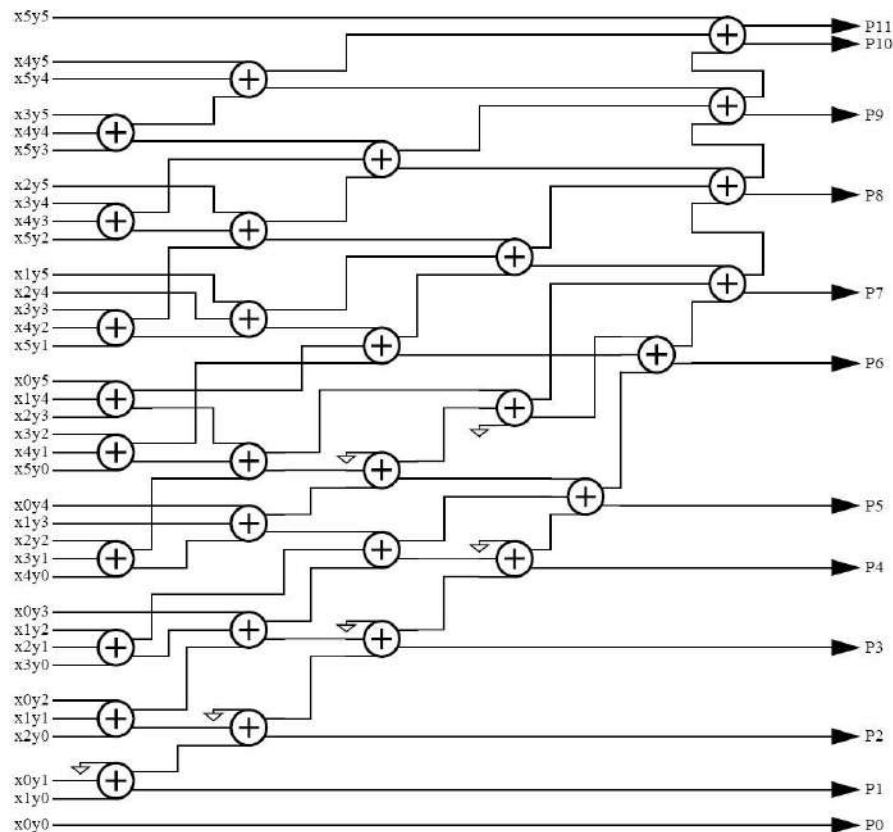


Figure 1.23: Wallace Tree Multiplication for 6-bits

can be used here. The delay through the array addition (not including the CPA) is proportional to  $\log_{1.5}(n)$ , where  $n$  is the width of the Wallace tree.

### Parity generator

1. Parity is a very useful tool in information processing in digital computers to indicate any presence of error in bit information.
2. External noise and loss of signal strength cause loss of data bit information while transporting data from one device to other device, located inside the computer or externally.
3. To indicate any occurrence of error, an extra bit is included with the message according to the total number of 1s in a set of data, which is called parity.
4. If the extra bit is considered 0 if the total number of 1s is even and 1 for odd quantities of 1s in a set



of data, then it is called even parity.

5. On the other hand, if the extra bit is 1 for even quantities of 1s and 0 for an odd number of 1s, then it is called odd parity

A parity generator is a combination logic system to generate the parity bit at the transmitting side.

Table 1.1: Truth table for generating even and odd parity bit

Four bit message $D_3D_2D_1D_0$	Even parity	Odd parity
0000	0	1
0001	1	0
0010	1	0
0011	0	1
01000	1	0
0101	0	1
0110	0	1
0111	1	0
1000	1	0
1001	0	1
1010	0	1
1011	1	0
1100	0	1
1101	1	0
1110	1	0
1111	0	1

If the message bit combination is designated as,  $D_3D_2D_1D_0$  and  $P_e$ ,  $P_o$  are the even and odd parity respectively, then it is obvious from the table that the Boolean expressions of even parity and odd parity are

$$P_e = D_3D_2D_1D_0$$

$$P_o = (D_3D_2D_1D_0)$$

The above illustration is given for a message with four bits of information. However, the logic diagrams can be expanded with more XOR gates for any number of bits.

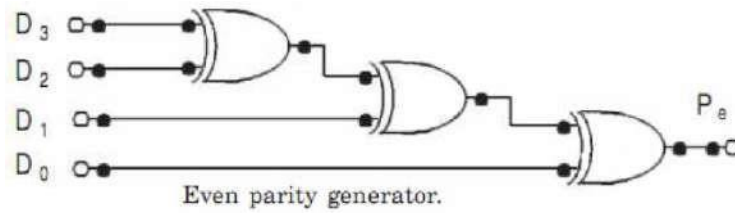


Figure 1.24: Even parity generator using logic gates

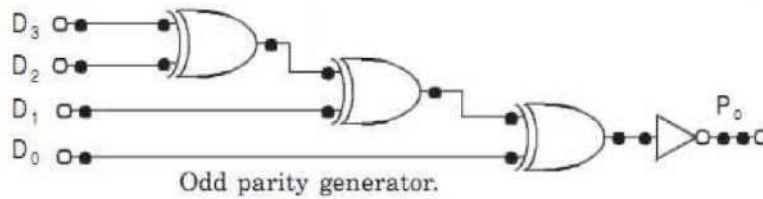


Figure 1.25: Odd parity generator logic gates

**Zero/One detector**

Detecting all ones or zeros on wide N-bit words requires large fan-in AND or NOR gates. Recall that by DeMorgan's law, AND, OR, NAND, and NOR are fundamentally the same operation except for possible inversions of the inputs and/or outputs. You can build a tree of AND gates, as shown in Figure 4.26(b). Here, alternate NAND and NOR gates have been used. The path has  $\log N$  stages.

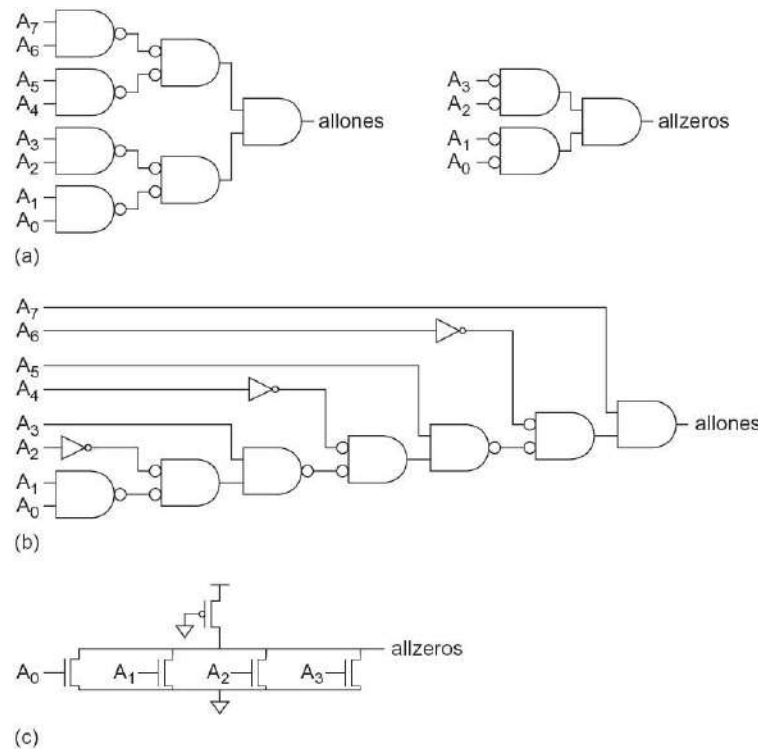


Figure: One/zero detectors (a) All one detector (b) All zero detector (c) All zero detector transistor level representation

## Comparators

Another common and very useful combinational logic circuit is that of the Digital Comparator circuit. Digital or Binary Comparators are made up from standard AND, NOR and NOT gates that compare the digital signals present at their input terminals and produce an output depending upon the condition of those inputs.

For example, along with being able to add and subtract binary numbers we need to be able to compare them and determine whether the value of input A is greater than, smaller than or equal to the value at input B etc. The digital comparator accomplishes this using several logic gates that operate on the principles of Boolean Algebra. There are two main types of Digital Comparator available and these are.

Identity Comparator an Identity Comparator is a digital comparator that has only one output terminal for when  $A = B$  either "HIGH"  $A = B = 1$  or "LOW"  $A = B = 0$

Magnitude Comparator a Magnitude Comparator is a type of digital comparator that has three output terminals, one each for equality,  $A = B$  greater than,  $A > B$  and less than  $A < B$

The purpose of a Digital Comparator is to compare a set of variables or unknown numbers, for example A ( $A_1, A_2, A_3, \dots, A_n$ , etc) against that of a constant or unknown value such as B ( $B_1, B_2, B_3, \dots, B_n$ , etc) and produce an output condition or flag depending upon the result of the comparison. For example, a magnitude comparator of two 1-bits, (A and B) inputs would produce the following three output conditions when compared to each other.

$A > B, A = B, A < B$

Which means: A is greater than B, A is equal to B, and A is less than B

This is useful if we want to compare two variables and want to produce an output when any of the above three conditions are achieved. For example, produce an output from a counter when a certain count number is reached. Consider the simple 1-bit comparator below.

Then the operation of a 1-bit digital comparator is given in the following Truth Table

Inputs		Outputs		
B	A	$A > B$	$A = B$	$A < B$
0	0	0	1	0
0	1	1	0	0
1	0	0	0	1
1	1	0	0	0

From the above table the obtained expressions for magnitude comparator using K-map are as follows

**For**  $A < \overline{\overline{B}} : C = \overline{A}B$

**For**  $A = B : D = \overline{A}B + A\overline{B}$

For  $A > B : E = A\overline{B}$  The logic diagram of 1-bit comparator using basic gates is shown below in Figure 1.24.

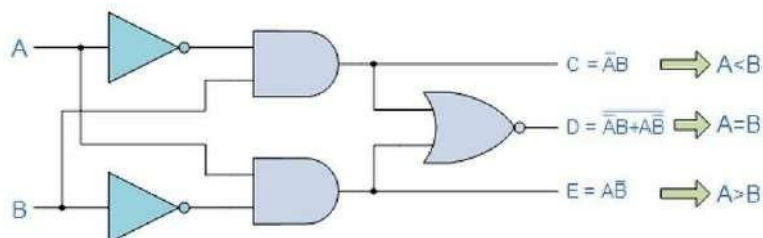


Figure 1.27: 1-bit Digital Comparator

\*\*\* Draw separate diagrams for greater, equality and less than expressions.

## Counters

Counters can be implemented using the adder/subtractor circuits and registers (or equivalently, D flip-flops)

The simplest counter circuits can be built using T flip-flops because the toggle feature is naturally suited for the implementation of the counting operation. Counters are available in two categories

**6. Asynchronous (Ripple counters)** Asynchronous counters, also known as ripple counters, are not clocked by a common pulse and hence every flip-flop in the counter changes at different times. The flip-flops in an asynchronous counter is usually clocked by the output pulse of the preceding flip-flop. The first flip-flop is clocked by an external event.

The flip-flop output transition serves as a source for triggering other flip-flops

i.e the C input (clock input) of some or all flip-flops are triggered NOT by the common clock pulses Eg:- Binary ripple counters, BCD ripple counters

**Synchronous counters** A synchronous counter however, has an internal clock, and the external event is used to produce a pulse which is synchronized with this internal clock.

C input (clock input) of all flip-flops receive the common clock pulses

E.g.:- Binary counter, Up-down Binary counter, BCD Binary counter, Ring counter, Johnson counter,

## Asynchronous Up-Counter with T Flip-Flops

Figure shows a 3-bit counter capable of counting from 0 to 7. The clock inputs of the three flip-flops are connected in cascade. The T input of each flip-flop is connected to a constant 1, which means that the state of the flip-flop will be toggled at each active edge (here, it is

positive edge) of its clock. We assume that the purpose of this circuit is to count the number of pulses that occur on the primary input called Clock. Thus the clock input of the first flip-flop is connected to the Clock line. The other two flip-flops have their clock inputs driven by the  $Q$  output of the preceding flip-flop. Therefore, they toggle their states whenever the preceding flip-flop changes its state from  $Q = 1$  to  $Q = 0$ , which results in a positive edge of the  $Q$  signal.

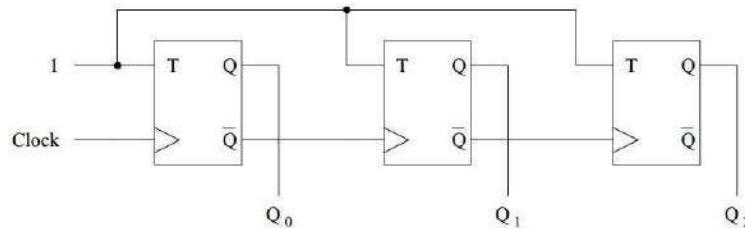


Figure: A 3-bit up-counter.

Note here the value of the count is indicated by the 3-bit binary number  $Q_2Q_1Q_0$ . Since the second flip-flop is clocked by  $Q_0$ , the value of  $Q_1$  changes shortly after the change of the  $Q_0$  signal. Similarly, the value of  $Q_2$  changes shortly after the change of the  $Q_1$  signal. This circuit is a modulo-8 counter. Because it counts in the upward direction, we call it an up-counter. This behavior is similar to the rippling of carries in a ripple-carry adder. The circuit is therefore called an asynchronous counter, or a ripple counter.

### Asynchronous Down-Counter with T Flip-Flops

Some modifications of the circuit in Figure 4.29 lead to a down-counter which counts in the sequence 0, 7, 6, 5, 4, 3, 2, 1, 0, 7, and so on. The modified circuit is shown in Figure 3. Here the clock inputs of the second and third flip-flops are driven by the  $Q$  outputs of the preceding stages, rather than by the  $\bar{Q}$  outputs.

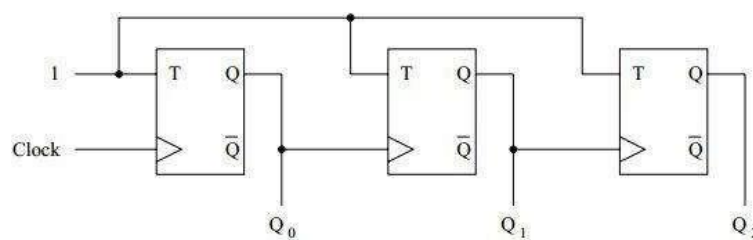


Figure: A 3-bit down-counter.

Although the asynchronous counter is easier to construct, it has some major disadvantages over the synchronous counter.

First of all, the asynchronous counter is slow. In a synchronous counter, all the flip-flops will change states simultaneously while for an asynchronous counter, the propagation delays of the flip-flops add together to produce the overall delay. Hence, the more bits or number of flip-flops in an asynchronous counter, the slower it will be.

Secondly, there are certain "risks" when using an asynchronous counter. In a complex

system, many state changes occur on each clock edge and some ICs respond faster than others. If an external event is allowed to affect a system whenever it occurs (unsynchronized), there is a small chance that it will occur near a clock transition, after some IC's have responded, but before others have. This intermingling of transitions often causes erroneous operations. And the worse this is that these problems are difficult to foresee and test for because of the random time difference between the events.

**Synchronous Counters:**

A synchronous counter usually consists of two parts: the memory element and the combinational element. The memory element is implemented using flip-flops while the combinational element can be implemented in a number of ways. Using logic gates is the traditional method of implementing combinational logic and has been applied for decades.

**Synchronous Up-Counter with T Flip-Flops**

An example of a 4-bit synchronous up-counter is shown in Figure 5. Observing the

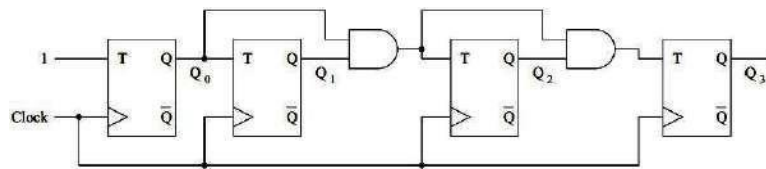


Figure A 4bit synchronous upcounter

Clock Cycle	Q <sub>3</sub>	Q <sub>2</sub>	Q <sub>1</sub>	Q <sub>0</sub>
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1
16	0	0	0	0

Figure: Contents of a 4bit upcounter for 16 consecutive clock cycles

pattern of bits in each row of the table, it is apparent that bit Q<sub>0</sub> changes on each clock cycle. Bit Q<sub>1</sub> changes only when Q<sub>0</sub> = 1. Bit Q<sub>2</sub> changes only when both Q<sub>1</sub> and Q<sub>0</sub> are equal to 1.

Bit  $Q_3$  changes only when  $Q_2 = Q_1 = Q_0 = 1$ . In general, for an n-bit up-counter, a give flip-flop changes its state only when all the preceding flip-flops are in the state  $Q = 1$ . Therefore, if we use T flip-flops to realize the 4-bit counter, then the T inputs should be defined as

$$T_0 = 1$$

$$T_1 = Q_0$$

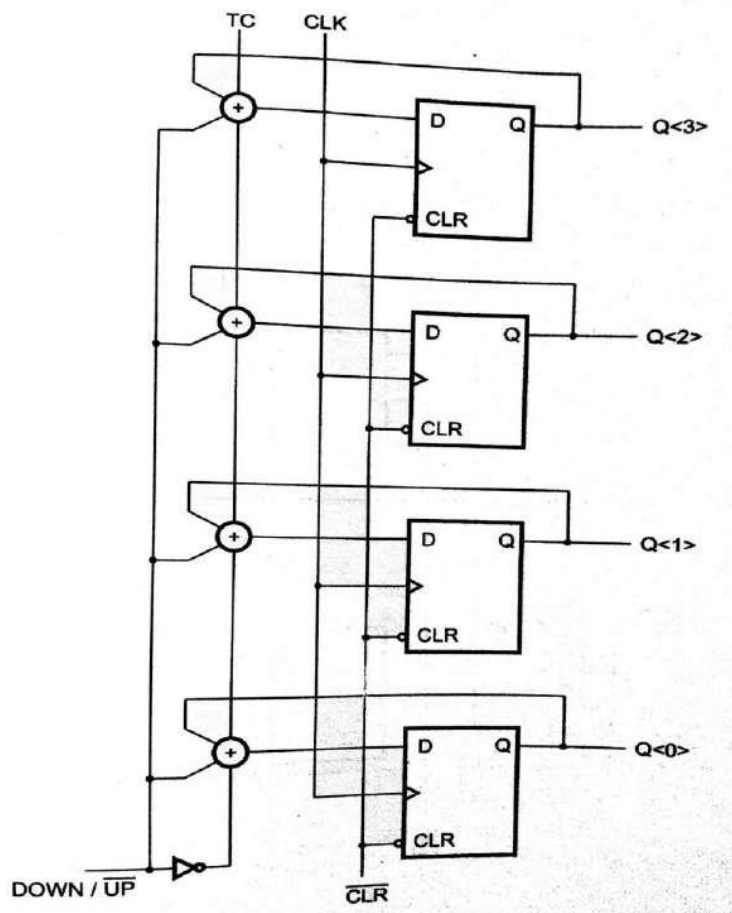
$$T_2 = Q_0Q_1$$

$$T_3 = Q_0Q_1Q_2$$

In Figure 5, instead of using AND gates of increased size for each stage, we use a factored arrangement. This arrangement does not slow down the response of the counter, because all flip-flops change their states after a propagation delay from the positive edge of the clock. Note that a change in the value of  $Q_0$  may have to propagate through several AND gates to reach the flip-flops in the higher stages of the counter, which requires a certain amount of time. This time must not exceed the clock period. Actually, it must be 3less than the clock period minus the setup time of the flip-flops. It shows that the circuit behaves as a modulo-16 up-counter. Because all changes take place with the same delay after the active edge of the Clock signal, the circuit is called a synchronous counter.

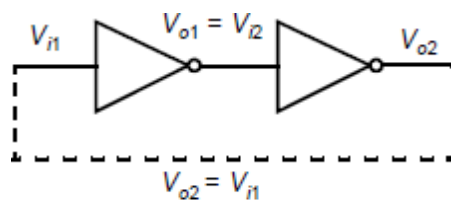
Figure 1.32: Design of synchronous counter using adders and registers

### Static Latches and Registers

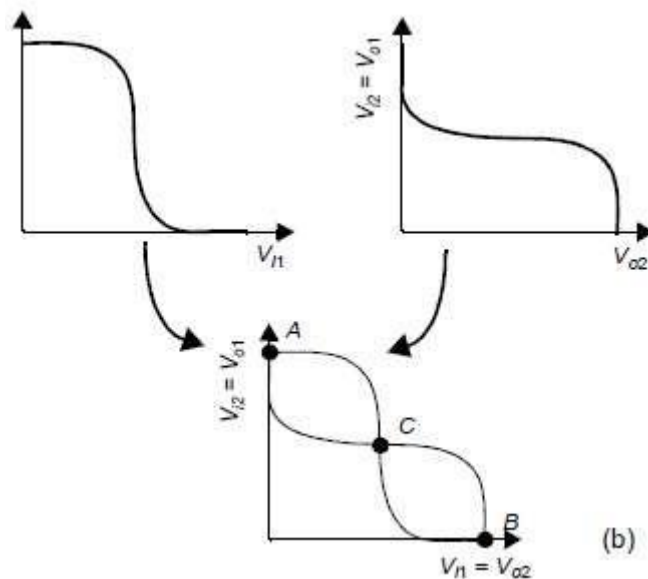


### The Bistability Principle:

Static memories use positive feedback to create a bistable circuit — a circuit having two stable states that represent 0 and 1. The basic idea is shown in Figure 7.4a, which shows two inverters connected in cascade along with a voltage-transfer characteristic typical of such a circuit. Also plotted are the VTCs of the first inverter, that is,  $V_{o1}$  versus  $V_{i1}$ , and the second inverter ( $V_{o2}$  versus  $V_{o1}$ ). The latter plot is rotated to accentuate that  $V_{i2} = V_{o1}$ . Assume now that the output of the second inverter  $V_{o2}$  is connected to the input of the first  $V_{i1}$ , as shown by the dotted lines in Figure 7.4a. The resulting circuit has only three possible operation points ( $A$ ,  $B$ , and  $C$ ), as demonstrated on the combined VTC. The following important conjecture is easily proven to be valid:



(a)



(b)

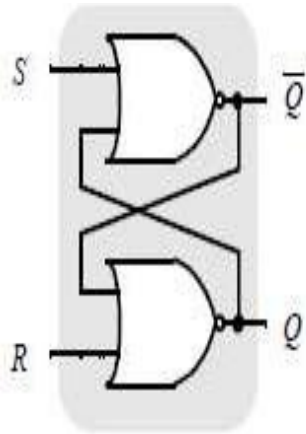
Under the condition that the gain of the inverter in the transient region is larger than 1, only  $A$  and  $B$  are stable operation points, and  $C$  is a metastable operation point.

### SR Flip-Flops

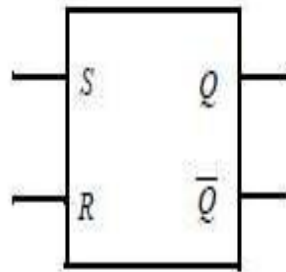
The cross-coupled inverter pair shown in the previous section provides an approach to store a binary variable in a stable way. However, extra circuitry must be added to enable control of the memory states. The wellknown *SR* —or *set-reset*— *flip-flop*, an implementation of which is shown in Figure 7.6a. This circuit is similar to the cross-coupled inverter pair with NOR gates replacing the inverters. The second input of the NOR gates is connected to the trigger inputs ( $S$  and  $R$ ), that make it possible to force the outputs  $Q$  and  $\bar{Q}$  to a given state. These outputs are complimentary (except for the  $SR = 11$  state). When both  $S$  and  $R$  are 0, the flip-flop is in a quiescent state and both outputs retain their value (a NOR gate with one of its input being 0



looks like an inverter, and the structure looks like a cross coupled inverter). If a positive (or 1) pulse is applied to the  $S$  input, the  $Q$  output is forced into the 1 state (with  $\bar{Q}$  going to 0). Vice versa, a 1 pulse on  $R$  resets the flip-flop and the  $Q$  output goes to 0.



(a) Schematic diagram



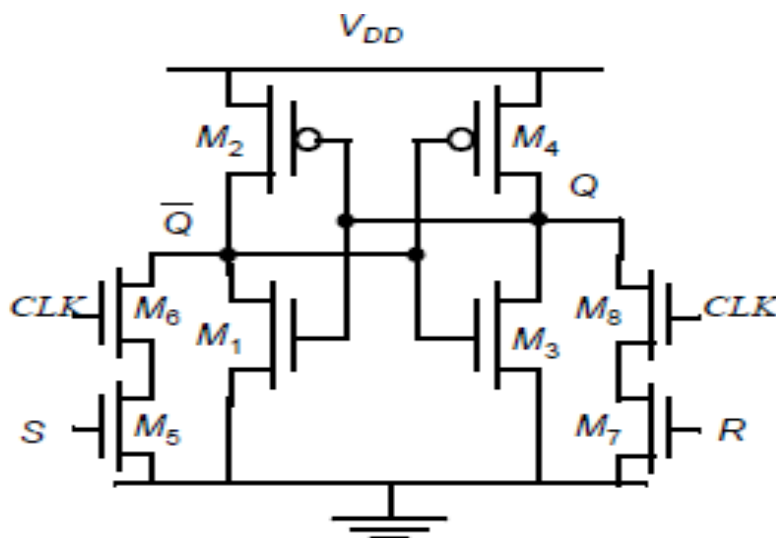
(b) Logic symbol

$S$	$R$	$Q$	$\bar{Q}$
0	0	$Q$	$\bar{Q}$
1	0	1	0
0	1	0	1
1	1	0	0

Forbidden State

(c) Characteristic table

These results are summarized in the *characteristic table* of the flip-flop, shown in Figure 7.6c. The characteristic table is the truth table of the gate and lists the output states as functions of all possible input conditions. When both  $S$  and  $R$  are high, both  $Q$  and  $\bar{Q}$  are forced to zero. Since this does not correspond with our constraint that  $Q$  and  $\bar{Q}$  must be complementary, this input mode is considered to be forbidden. An additional problem with this condition is that when the input triggers return to their zero levels, the resulting state of the latch is unpredictable and depends on whatever input is last to go low. Finally, Figure 7.6 shows the schematics symbol of the  $SR$  flip-flop. The  $SR$  flip-flops discussed so far are asynchronous, and do not require a clock signal. Most systems operate in a synchronous fashion with transition events referenced to a clock. One possible realization of a clocked  $SR$  flip-flop— a *level-sensitive* positive latch—is shown in Figure 7.8.



It consists of a cross-coupled inverter pair, plus 4 extra transistors to drive the flip-flop from one state to

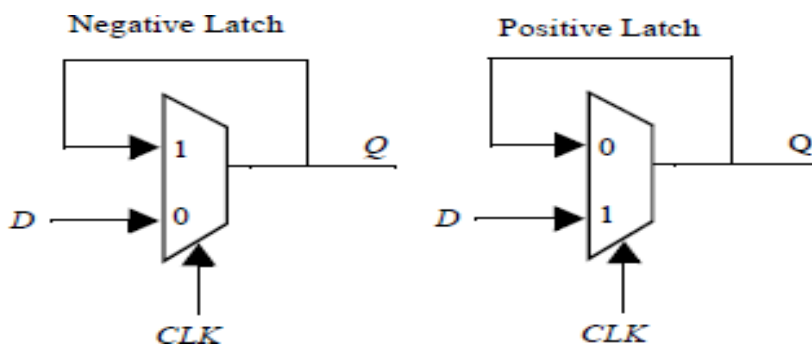
another and to provide clocked operation. Observe that the number of transistors is identical to the implementation of Figure 7.6, but the circuit has the added feature of being clocked. The drawback of saving some transistors over a fully-complimentary CMOS implementation is that transistor sizing becomes critical in ensuring proper functionality. Consider the case where  $Q$  is high and an  $R$  pulse is applied. The combination of transistors  $M4$ ,  $M7$ , and  $M8$  forms a ratioed inverter. In order to make the latch switch, we must succeed in bringing  $Q$  below the switching threshold of the inverter  $M1$ - $M2$ . Once this is achieved, the positive feedback causes the flip-flop to invert states. This requirement forces us to increase the sizes of transistors  $M5$ ,  $M6$ ,  $M7$ , and  $M8$ .

### Multiplexer-Based Latches

There are many approaches for constructing latches. One very common technique involves the use of transmission gate multiplexers. Multiplexer based latches can provide similar functionality to the  $SR$  latch, but has the important added advantage that the sizing of devices only affects performance and is not critical to the functionality.

Figure 7.11 shows an implementation of static positive and negative latches based on multiplexers. For a negative latch, when the clock signal is low, the input 0 of the multiplexer is selected, and the  $D$  input is passed to the output. When the clock signal is high, the input 1 of the multiplexer, which connects to the output of the latch, is selected. The feedback holds the output stable while the clock signal is high. Similarly in the positive latch, the  $D$  input is selected when clock is high, and the output is held (using feedback) when clock is low.

A transistor level implementation of a positive latch based on multiplexers is shown in Figure 7.12. When  $CLK$  is high, the bottom transmission gate is *on* and the latch is



*transparent* - that is, the  $D$  input is copied to the  $Q$  output. During this phase, the feedback loop is open since the top transmission gate is *off*. Unlike the  $SR$  FF, the feedback does not have to be overridden to write the memory and hence sizing of transistors is not critical for realizing correct functionality. The number of transistors that the clock touches is important since it has an *activity factor* of 1. This particular latch implementation is not particularly efficient from this metric as it presents a load of 4 transistors to the  $CLK$  signal.

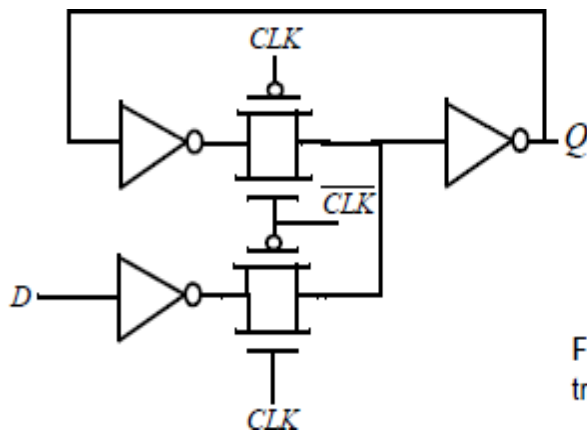
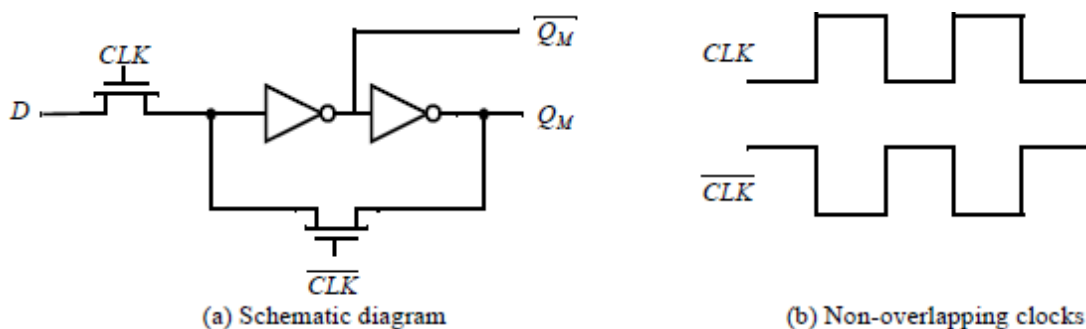


Figure 7.12 Positive latch built using transmission gates.

It is possible to reduce the clock load to two transistors by using implement multiplexers using NMOS only pass transistor as shown in Figure 7.13. The advantage of this approach is the reduced clock load of only two NMOS devices. When  $CLK$  is high, the latch samples the  $D$  input, while a low clock-signal enables the feedback-loop, and puts the latch in the hold mode. While attractive for its simplicity, the use of NMOS only pass transistors results in the passing of a degraded high voltage of  $VDD-V_{Tn}$  to the input of the first inverter. This impacts both noise margin and the switching performance, especially in the case of low values of  $VDD$  and high values of  $V_{Tn}$ . It also causes static power dissipation in first inverter, as already pointed out in Chapter 6. Since the maximum input-voltage to the inverter equals  $VDD-V_{Tn}$ , the PMOS device of the inverter is never turned off, resulting in a static current flow.

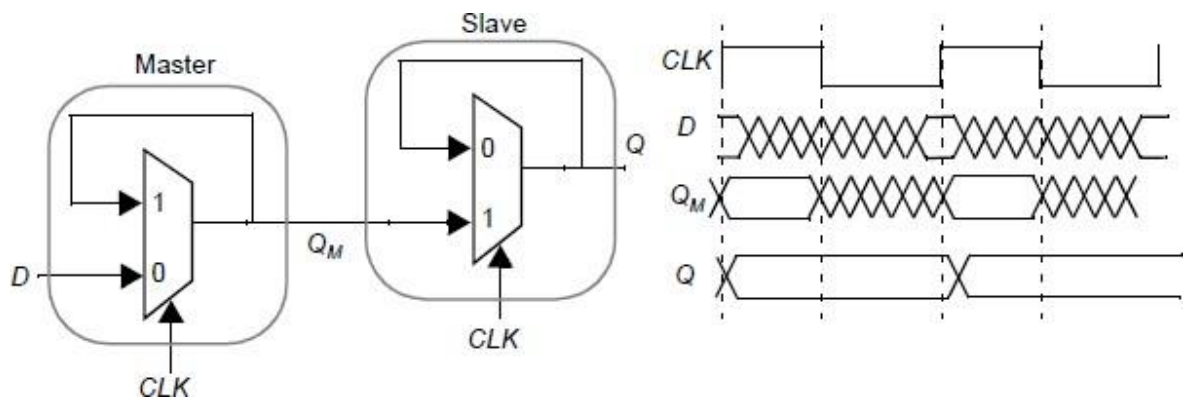
### Master-Slave Edge-Triggered Register

The most common approach for constructing an *edge-triggered* register is to use a *masterslave* configuration, as shown in Figure 7.14. The register consists of cascading a negative latch (master stage) with a positive latch (slave stage). A multiplexer-based latch is used in this particular implementation, although any latch could be used. On the low phase of the clock, the master stage is *transparent*, and the  $D$  input is passed to the master stage output, using feedback. On the rising edge of the clock, the master stage goes into a *hold* mode, and the slave stage starts sampling the output of the master stage ( $Q_M$ ), while the master stage remains in a *hold* mode. Since  $Q_M$  is constant during the high phase of the clock, the output  $Q$  makes only one transition per cycle. The value of  $Q$  is the value of  $D$  right before the rising edge of the clock, achieving the *positive edge-triggered* effect. A *negative edge-triggered* register can be constructed



On the rising edge of the clock, the master slave stops sampling the input, and the slave stage starts sampling. During the high phase of the clock, the slave stage samples the output of the master stage ( $Q_M$ ), while the master stage remains in a *hold* mode. Since  $Q_M$  is constant during the high phase of the clock, the output  $Q$  makes only one transition per cycle. The value of  $Q$  is the value of  $D$  right before the rising edge of the clock, achieving the *positive edge-triggered* effect. A *negative edge-triggered* register can be constructed

using the same principle by simply switching the order of the



positive and negative latch (this is, placing the positive latch first). A complete transistor-level implementation of a the *master-slave positive edge-triggered* register is shown in Figure 7.15. The multiplexer is implemented using transmission gates as discussed in the previous section. When the clock is low ( $CLK = 1$ ),  $T1$  is *on* and  $T2$  is *off*, and the  $D$  input is sampled onto node  $Q_M$ . During this period,  $T3$  is *off* and  $T4$  is *on* and the cross-coupled inverters ( $I5, I6$ ) holds the state of the slave latch. When the clock goes high, the master stage stops sampling the input and goes into a *hold* mode.  $T1$  is *off* and  $T2$  is *on*, and the cross coupled inverters  $I3$  and  $I4$  holds the state of  $Q_M$ . Also,  $T3$  is *on* and  $T4$  is *off*, and  $Q_M$  is copied to the output  $Q$ .

### Dynamic Latches and Registers

Storage in a static sequential circuit relies on the concept that a cross-coupled inverter pair produces a bistable element and can thus be used to memorize binary values. This approach has the useful property that a stored value remains valid as long as the supply voltage is applied to the circuit, hence the name *static*. The major disadvantage of the static gate, however, is its complexity. This results in a class of circuits based on temporary storage of charge on parasitic capacitors. The principle is exactly identical to the one used in dynamic logic — charge stored on a capacitor can be used to represent a logic signal. The absence of charge denotes a 0, while its presence stands for a stored 1. No capacitor is ideal, unfortunately, and some charge leakage is always present.

### Dynamic Transmission-Gate Edge-triggered Registers

A fully dynamic *positive edge-triggered* register based on the *master-slave* concept is shown in Figure 7.24. When  $CLK = 0$ , the input data is sampled on storage node 1, which has an equivalent capacitance of  $C1$  consisting of the gate capacitance of  $I1$ , the junction capacitance of  $T1$ , and the overlap gate capacitance of  $T1$ . During this period, the slave stage is in a *hold* mode, with node 2 in a high-impedance (floating) state. On the rising edge of clock, the transmission gate  $T2$  turns on, and the value sampled on node 1 right before the rising edge propagates to the output  $Q$  (note that node 1 is stable during the high phase of the clock since the first transmission gate is turned *off*). Node 2 now stores the inverted version of node 1. This implementation of an *edge-triggered* register is very efficient as it requires only 8 transistors. The sampling switches can be implemented using NMOS-only pass transistors, resulting in an even-simpler 6 transistor

implementation. The reduced transistor count is attractive for high-performance and low-power systems. The *set-up time* of this circuit is simply the delay of the transmission gate, and corresponds to the time it takes node 1 to sample the  $D$  input. The *hold time* is approximately zero, since the transmission gate is turned *off* on the clock edge and further inputs changes are ignored. The *propagation delay* ( $t_{c-q}$ ) is equal to two inverter delays plus the delay of the transmission gate  $T_2$ . One important consideration for such a dynamic register is that the storage nodes (i.e., the state) has to be refreshed at periodic intervals to prevent a loss due to charge leakage, due to diode leakage as well as sub-threshold currents. In datapath circuits, the refresh rate is not an issue since the registers are periodically clocked, and the storage nodes are constantly updated. Clock overlap is an important concern for this register. Consider the clock waveforms shown in Figure 7.25. During the 0-0 overlap period, the NMOS of  $T_1$  and the PMOS of  $T_2$  are simultaneously on, creating a direct path for data to flow from the  $D$  input of the register to the  $Q$  output. This is known as a *race condition*. The output  $Q$  can change on the falling edge if the overlap period is large — obviously an undesirable effect for a *positive edge-triggered* register.

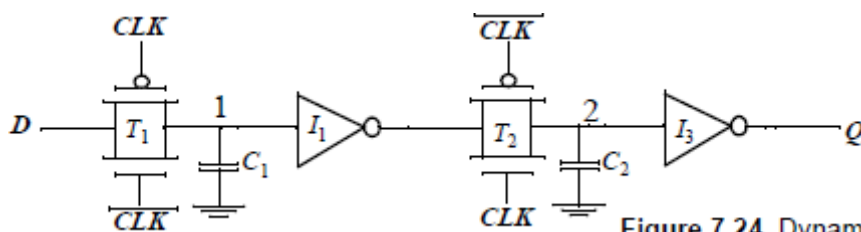


Figure 7.24 Dynamic edge-triggered register.

The same is true for the 1-1 overlap region, where an input-output path exists through the PMOS of  $T_1$  and the NMOS of  $T_2$ . The latter case is taken care off by enforcing a *hold time* constraint. That is, the data must be stable during the high-high overlap period. The former situation (0-0 overlap) can be addressed by making sure that there is enough delay between the  $D$  input and node 2 ensuring that new data sampled by the master stage does not propagate through to the slave stage. Generally the

built in single inverter delay should be sufficient and the overlap period constraint is given

as: The *set-up time* of this circuit is simply the delay of the transmission gate, and corresponds to the time it takes node 1 to sample the  $D$  input. The *hold time* is approximately zero, since the transmission gate is turned *off* on the clock edge and further inputs changes are ignored. The *propagation delay* ( $t_{c-q}$ ) is equal to two inverter delays plus the delay of the transmission gate  $T_2$ .

One important consideration for such a dynamic register is that the storage nodes (i.e., the state) has to be refreshed at periodic intervals to prevent a loss due to charge leakage, due to diode leakage as well as sub-threshold currents. In datapath circuits, the refresh rate is not an issue since the registers are periodically clocked, and the storage nodes are constantly updated.

Clock overlap is an important concern for this register. Consider the clock waveforms

shown in Figure 7.25. During the 0-0 overlap period, the NMOS of  $T1$  and the PMOS of  $T2$  are simultaneously on, creating a direct path for data to flow from the  $D$  input of the register to the  $Q$  output. This is known as a *race condition*. The output  $Q$  can change on the falling edge if the overlap period is large — obviously an undesirable effect for a *positive edge-triggered* register. The same is true for the 1-1 overlap region, where an input-output path exists through the PMOS of  $T1$  and the NMOS of  $T2$ . The latter case is taken care off by enforcing a *hold* time constraint. That is, the data must be stable during the high-high overlap period. The former situation (0-0 overlap) can be addressed by making sure that there is enough delay between the  $D$  input and node 2 ensuring that new data sampled by the master stage does not propagate through to the slave stage. Generally the built in single inverter delay should be sufficient and the overlap period constraint is given as:

$$t_{\text{overlap } 0-0} < t_{T1} + t_{I1} + t_{T2}$$

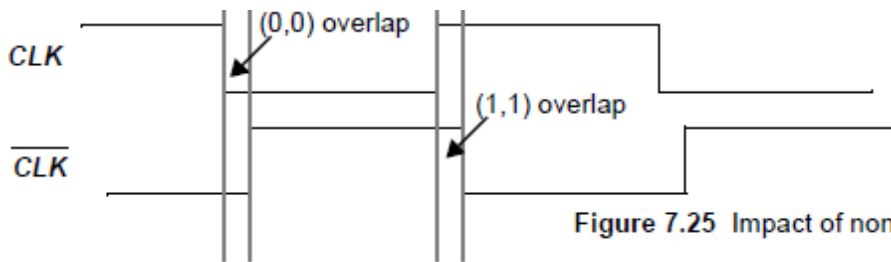


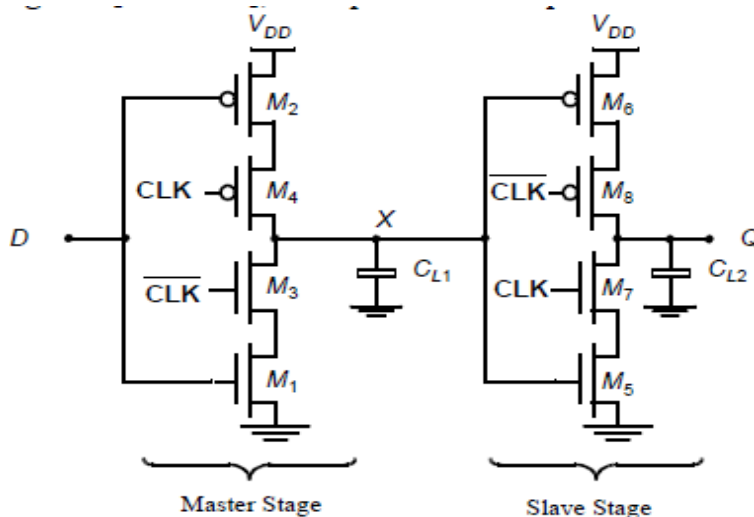
Figure 7.25 Impact of non-overlapping clocks.

Similarly, the constraint for the 1-1 overlap is given as:

$$t_{\text{hold}} > t_{\text{overlap } 1-1}$$

### The C2MOS Register

Figure 7.26 shows an ingenious *positive edge-triggered* register, based on a *master-slave* concept insensitive to clock overlap. This circuit is called the C2MOS (Clocked CMOS) register [Suzuki73], and operates in two phases.



**1.**  $CLK = 0$  ( $CLK = 1$ ): The first tri-state driver is turned on, and the master stage acts as an inverter sampling the inverted version of  $D$  on the internal node  $X$ . The master stage is in the *evaluation mode*. Meanwhile, the slave section is in a high-impedance mode, or in a *hold mode*. Both transistors  $M7$  and  $M8$  are off, decoupling the output from the input. The output  $Q$  retains its previous value stored on the output capacitor  $CL2$ .

**2.** The roles are reversed when  $CLK = 1$ : The master stage section is in hold mode ( $M3$ - $M4$  off), while the second section evaluates ( $M7$ - $M8$  on). The value stored on  $CL1$  propagates to the output node through the slave stage which acts as an inverter.