UNIT-4
DATAFLOW LEVEL AND SWITCH LEVEL MODELLING

Syllabus: Introduction**,** continuous assignment structures, delays and continuous assignments, assignment to vectors, basic transistor switches, CMOS switch, Bidirectional gates and time delays with switch primitives, instantiations with strengths and delays strength contention with trireg nets.

INTRODUCTION

- **Gate level design** description makes use of the gate primitives available in Verilog. These are repeatedly and judiciously instantiated to achieve the full design description.
- **Behavioral level** modeling constitutes design description at an abstract level. it can be described at a functional level itself instead of getting bogged down with implementation details.
- **Data flow level** description of a digital circuit is at a higher level. It makes the circuit description more compact as compared to design through gate primitives.
- The operations are carried out on the operand(s) in singles or in combinations [IEEE]. The results are assigned to nets. The operand- operation-assignments representing data flow are carried out repeatedly to complete the design description

CONTINUOUS ASSIGNMENT STRUCTURES

- A simple two input AND gate in data flow format has the form

assign c = a && b;

Here "**assign**" is the keyword carrying out the assignment operation. This type of assignment is called a ***continuous assignment.***

a and b are operands – typically single-bit logic variables.

"&&" is a logic operator. It does the bit-wise AND operation on the two operands a and b.

"=" is an assignment activity carried out.

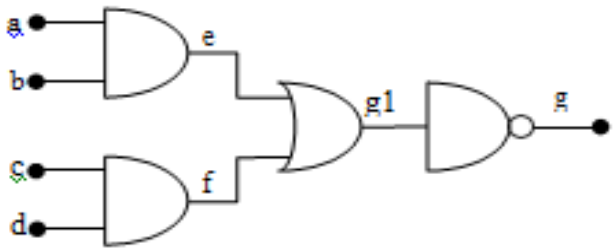
c is a net representing the signal which is the result of the assignment.

```
module andgdf(c,a,b);
output c;
input a,b;
wire c;
assign c = a&&b;
endmodule
```

A module with an AND gate at the data flow level .

```
assign e = a&&b, f = c&&d, g1 = e|f, g = ~g1;
```

```
assign e = a & b, f = c & d;
assign g1 = e|f, g = ~g1;
```




```
assign e  = a & b;
assign f  = c & d;
assign g1 = e ! f;
assign g  = ~g1;
```

Combining Assignment and Net Declarations

The assignment statement can be combined with the net declaration itself making the assignment implicit in the net declaration itself. Thus the two statements

wire c;

assign c = a & b; can be combined as **wire** c = a & b;

Example1:

module aoi1(g,a,b,c,d);
output g;
input a,b,c,d;
wire e,f,g1,g;
assign e = a && b,f = c && d, g1 = e||f, g=~g1;
endmodule

Example2:

```
module aoi2(g,a,b,c,d);
output g;
input a,b,c,d;
wire g;
wire     e= a && b;
 wire    f= c && d;
 wire    g1 = e||f;
assign g = ~g1;
endmodule
```

Continuous Assignments and Strengths

- A net to which a continuous assignment is being made can be assigned strengths for its logic levels.
- The procedure is akin to the strength allocation to the outputs of primitives.
- The assignment to g can be combined with the wire declaration into a single statement as **wire** (**pull1, strong0**)g = ~g1;

Example 3:

```
module aoi3 (g, a, b, c, d);
output g;
input a, b, c, d;
wire g;
wire e  = a &&b;
wire f  = c & &d;
wire g1 = e || f;
assign (pull1, strong0)g = ~g1;
endmodule
```

DELAYS AND CONTINUOUS ASSIGNMENTS

Delays can be incorporated at the data flow level in different ways

Example1:

```
wire c, a, b;

assign #2   c = a & b;
```

Example2:

```
wire a, b;

wire #2 c = a & b;
```

- The assignment takes effect with a time delay of 2 time steps. If a or b changes in value, the program waits for 2 time steps, computes the value of c based on the values of a and b at the time of computation, and assigns it to c.
- In the interim period, a or b may change further, but c changes and takes the new value only 2 time steps after the change in a or b initiates it.

ASSIGNMENT TO VECTORS

- The continuous assignments are equally applicable to vectors. A single statement can describe operations involving vectors wherever possible.

Concatenation of Vectors: One can concatenate vectors, scalars, and part vectors to form other vectors. The concatenated vector is enclosed within braces. Commas separate the components –scalars, vectors, and part vectors.

{2{p}}= {p, p}

{2{p}, q} = {p, p, q}

{a, 3 {2{b , c}, d}} = {a, b, c, b, c, d, b, c, b, c, d, b, c, b, c, d }

Example: Eight Bit adder

```
module add_8(a,b,c);
input[7:0]a,b;
output[7:0]c;
assign c = a + b ;
Endmodule
```

OPERATORS

Unary Operators

Operator type	Symbol	Remarks
Logical negation	!	Only for scalars
Bit-wise negation	~	For scalars and vectors
Reduction AND	&	For vectors – yields a single bit output
Reduction NAND	~&	
Reduction OR	|	
Reduction NOR	~|	
Reduction XOR	^	
Reduction XNOR	~^ or ^~	

Binary Operators

Arithmetic operators and their symbols

Operand type	Symbol	Remarks
Multiplication	*	
Division	/	The result is x if the denominator is zero
Modulus	%	
Addition	+	
Subtraction	–	

Binary logical operators and their symbols

Operator type	Symbol	Possible output value
AND	&&	Single-bit output
OR	||	

Relational operators and their symbols

Operator type	Symbol	Possible output value
Greater than	>	Single-bit output
Less than	<	
Greater than or equal to	>=	
Less than or equal to	<=	

Equality operators and their symbols

Operand symbol	Description of operand	Possible logical value of result
==	(The symbol comprises two consecutive equal signs.) If the two operands are equal bit by bit, the result is 1 (true); else the result is 0 (false). If either operand has a **x** or **z** bit, the result is **x**.	0, 1, or **x**
!=	(The symbol comprises of an exclamation mark followed by an equal sign.) A bit-by-bit comparison of the two operands is made. The result is a 1 if there is a mismatch for at least one bit position.	0, 1, or **x**
===	(The symbol comprises of three consecutive equal signs.) The operand bits can be 0, 1, **x**, or **z**. If the two operands match on a bit by bit basis, the result is a 1 (true) bit; else it is 0 (false). Note that the result is never **x** here.	0 or 1
!==	(The symbol comprises an exclamation mark followed by 2 consecutive equal signs). The operand bits can be 0, 1, **x**, or **z**. If the two operands do not match on a bit by bit basis, the result is a 1 (true) bit; else it is 0 (false). Note that the result is never **x** here.	0 or 1

Bit-wise logical operators and their symbols

Operator type	Symbol	Possible result
AND	&	0, 1, or **x**
OR	|	
XOR	^	
XNOR	~^ or ^~	

Shift type operators and their symbols

Operand	Typical usage	Operation
>>	A >> b	The set of bits representing A are shifted right repeatedly b times.
<<	A<< b	The set of bits representing A are shifted left repeatedly b times.

Ternary operator: Verilog has only one ternary operator – the conditional operator. It checks a condition and does a branching. It is a versatile and powerful operator. It enhances the potential of design description substantially. The general form is

A?b:c

The conditional operation is made up of two operators – "?" and ":" – and three operands. The two operands separate the three operators in the order shown

"A" is evaluated first.

If A is true, b is evaluated.

If A is false, c is evaluated.

- As an example1, consider the assignment statement

assign y = w ? x : z;

where w, x, y and z are binary bits. If the bit w is true (1), y is assigned the value of x: otherwise – that is, if w is false (0) – y is assigned the value of z. The assignment statement here multiplexes x and z onto y; w is the control signal here

- As an example2, ALU can be defined in a compact manner using the ternary operator. **assign** d = (f==add)?(a+b): ((f==subtract)?(a-b): ((f==compl)?~a: ~b));

In the example here, f is taken as a control word. If it is equal to the number add, d is to be equal to the sum of a and b. If f is equal to the number subtract, d is to be equal to the difference between a and b. If it is equal to the number compl, d is to be the complement of a.

Verilog Program for ALU in Data flow modeling

```
module alu_df1 (d, co, a, b, f,cci);
output [3:0] d;
output co;
wire[3:0]d;
wire co;
 input cci;
input [3 : 0 ] a, b;
input [1 : 0] f;
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
```

Operator Priority

The table brings out the order of precedence. The order of precedence decides the priority for sequence of execution and circuit realization in any assignment.

Unary operators	! & ~& | ~| ^ ~^ + –	Highest precedence
Binary operator	* ? /	
	+ -	
	<< >>	
	< <= > >=	
	== != === !==	
	& ^ ~^	
	|	
	&&	
	||	
Ternary operators	? :	Lowest precedence

SWITCH LEVEL MODELING

INTRODUCTION

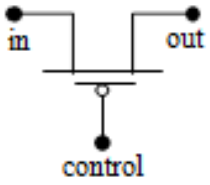
- The MOS transistor is the basic element around which a VLSI is built. Designers familiar with logic gates and their configurations at the circuit level may choose to do their designs using MOS transistors.
- Verilog has the provision to do the design description at the switch level using such MOS transistors, which is the theme of the present chapter.
- Switch level modeling forms the basic level of modeling digital circuits. The switches are available as primitives in Verilog

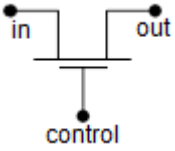
BASIC SWITCH PRIMITIVES

Different switch primitives are available in Verilog

nmos switch primitives as **nmos** (out, in, control);

pmos switch primitives as **pmos** (out, in, control);





nmos switch **pmos switch**

(Data) input	Control (input) 0	1	X	z
0	Z	0	L	L
1	Z	1	H	H
X	Z	X	X	X
z	z	z	z	z

(Data) input	Control (input) 0	1	X	z
0	Z	0	L	L
1	Z	1	H	H
X	Z	X	X	X
z	z	z	z	z

Resistive Switches: nmos and **pmos** represent switches of low impedance in the on-state. **rnmos** and **rpmos** represent the resistive counterparts of these respectively.

rnmos (output1, input1, control1);

rpmos (output2, input2, control2);

It inserts a definite resistance between the input and the output signals but retains the signal value. The **rpmos** and **rnmos** switches function as unidirectional switches; the signal flow is from the input to the output side.

strength levels

Output-side strength levels for different input-side strength values of rnmos, rpmos, and rcmos switches .

Input strength	Output strength
Supply – drive	Pull – drive
Strong – drive	Pull – drive
Pull – drive	Weak – drive
Weak – drive	Medium – capacitive
Large – capacitive	Medium – capacitive
Medium – capacitive	Small – capacitive
Small – capacitive	Small – capacitive
High impedance	High impedance

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.

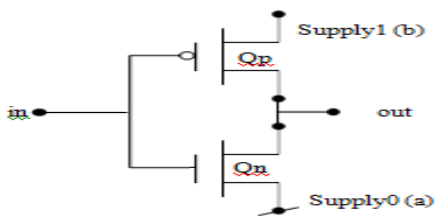
- **pullup** (x); Here net x is pulled up to the **supply1** through a resistance.

- **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 *V*CC or GND, respectively.

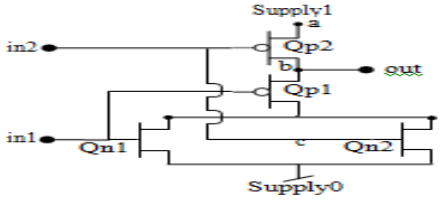
CMOS INVERTER

```
module inv (in, out );
output out;
input in;
supply0 a;
supply1 b;
nmos (out, a, in );
pmos (out, b, in);
endmodule
```

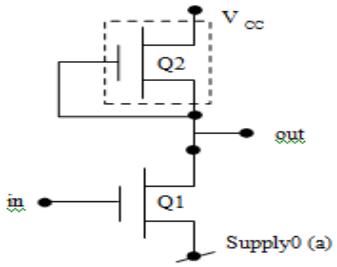


CMOS NOR

```
module npnor 2(out, in1, in2 );
output out;
input in1, in2;
supply1 a;
supply0 c;
wire b;
pmos(b, a, in2), (out, b, in1);
nmos(out, c, in1), (out, c, in2) ;
endmodule
```

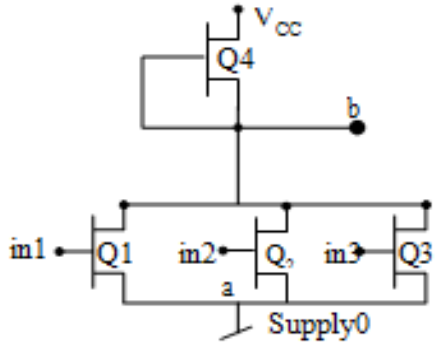


NMOS Inverter with Pull up Load




```
module NMOSinv(out,in);
 output out;
input in;
supply0 a;
pullup (out);
nmos(out,a,in);
endmodule
```

NMOS Three Input NOR Gate

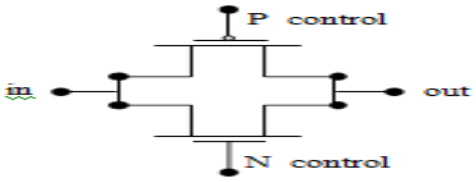


```
module nor3NMOS(in1,in2,in3,b);
output b;
input in1,in2,in3;
supply0 a;
pullup(b);
wire b;
nmos(b,a,in1),(b,a,in2),(b,a,in3);
endmodule
```

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. The CMOS switch is instantiated as shown below.

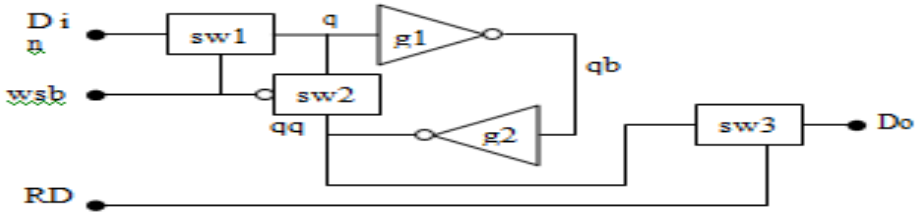
cmos csw (out, in, N_control, P_control);

```
module CMOSsw(out,in,n_ctr,p_ctr);
output out; input in,n_ctr,p_ctr;
nmos gn(out,in,n_ctr);
pmos gp(out,in,p_ctr);
endmodule
```

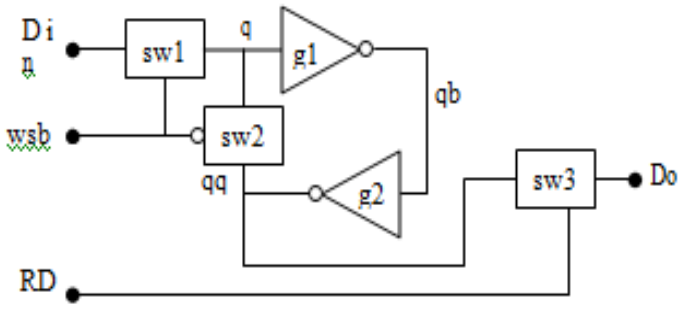


A RAM Cell

- A basic RAM cell with facilities for writing data, storing data, and reading data.
- When switch sw2 is ON, qb – the output of inverter g1 – forms the input to the inverter g2 and vice versa.
- The g1-g2 combination functions as a latch and freezes the last state entry before sw2 turns on. The step-by-step function of the cell is as



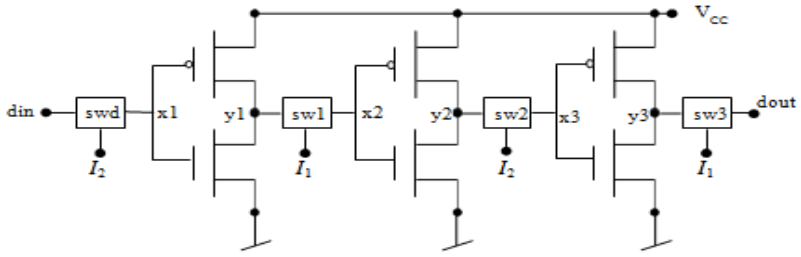
- When wsb (write/store) is high, switch sw1 is ON, and switch sw2 OFF. With sw1 on, input Din is connected to the input of gate g1 and remains so connected.
- When wsb goes low, din is isolated, since sw1 is OFF. But sw2 is ON and the data remains latched in the latch formed by g1-g2. In other words the data Din is stored in the RAM cell formed by g1-g2.
- When RD (Read) goes active (=1), the latched state is available as output Do. Reading is normally done when the latch is in the stored state.




```
module ram_cell(do,din,wsb,rd);
output do;
input din,wsb,rd;
wire sb;
wire q,qq;
tri do;
csw sw1(q,din,wsb),sw2(q,qq,sb),sw3(do,q,rd);
not n1(sb,wsb),n2(qb,q),n3(qq,qb);
endmodule
module csw(out,in,n_ctr);
output out;
input in,n_ctr;
wire p_ctr;
assign p_ctr =~n_ctr;
cmos csw(out,in,n_ctr,p_ctr);
endmodule
```

A Dynamic Shift Register

- Three successive stages of a dynamic shift register. It is operated through a two-phase clock system – I1 and I2. Each stage has a CMOS inverter. Successive stages are given input through CMOS switches (sw1, sw2, *etc*.). I1 and I2 are symmetric clock waveforms in anti-phase. Two successive stages together form one storage element.



- When I2 is ON AND I1 is OFF. Din is input to stage 1 through switch swd. sw1 and sw3 are OFF and sw2 is ON. State of stage 2 (attained when I1 was high last) is coupled as input to stage 3 through switch sw2, and stage 3 takes up the new state.
- In the next half cycle, I1 is ON and I2 is OFF. sw1 and sw3 are ON and sw2 is OFF. State of stage 1 (attained when I2 was high last) is coupled as input to stage 2 through switch sw1 and Do takes up the new state from stage3 through sw3.

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

tran and rtran : The **tran** gate is a bi-directional gate of two ports. When instantiated, it connects the two ports directly.

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 rtranif1: **tranif1** is a bi-directional switch turned ON/OFF through a control line(c). It is in the ON-state when the control signal is at 1 (high) state

tranif1 (s1, s2, c);

tranif0 and rtranif0: **tranif0** and **rtranif0** are again bi-directional switches. The switch is OFF if the control line is in the 1 state, and it is ON when the control line is in the 0 state.

tranif0 (s1, s2, c);

TIME DELAYS WITH SWITCH PRIMITIVES

- **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.

- **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.
- **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 in a similar manner. Bidirectional 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); 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

INSTANTIATIONS WITH STRENGTHS AND DELAYS

- **nmos** (**strong1**, **strong0**) (delay_r, delay_f, delay_o) gg (s1, s2, ctrl) ;
- **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.

STRENGTH CONTENTION WITH TRIREG NETS

- Nets declared as **trireg** can have capacitive storage. Such storage can be assigned one of three strengths – **large**, **medium**, or **small**.
- Driving such a net from different sources can lead to contention