

B.E.

Sixth Semester Examination, Dec-2009

Digital System Design (EE310E)

Note : Attempt any *five* questions. All questions carry equal marks.

Q. 1. (a) Discuss various types of data objects and data types.

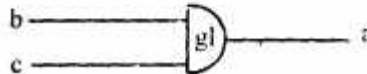
Ans. Various Types of Data Objects and Data Type :

(i) Value Set :

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.

In addition to logic values, strength levels are often used to resolve conflicts between drivers of different strength's in digital circuits.

(ii) Nets : Nets represent connections between hardware elements. Just as in real circuits, nets have values continuously driven on them by the outputs of devices that they are connected to.



(iii) Registers : Registers represents data storage elements. Registers retain value until another value is placed onto them. A register does not need a driver verilog registers do not need a clock as hardware registers do.

(iv) Vectors : The left number in the squared brackets is always the most significant bit of the vector.

(v) Integer, Real and Time Register Data Types :

(a) Integer : An integer is a general purpose register data type used for manipulating quantities. Integers are declared by the keyword integer.

(b) Real : Real number constants and real register data types are declared with the keyword real.

(c) 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.

(vi) Arrays : Arrays are allowed in verilog for reg, integer, time, real realtime and vector register data types.

(vii) Memories : In digital simulation, one often need to model register files, RAMs and ROMs. Memories are modeled in verilog simply as a one-dimensional array of registers.

(viii) Palometers : Verilog allows contents to be defined in a module by the keyword parameter.

(ix) Strings : Strings can be stored in reg the width of the register variable must be large enough to hold the string.

Q. 1. (b) What are operators? Explain all six different types.

Ans. Operators : 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 lan-

guage.

(i) **Arithmetic Operators** : There are two types of arithmetic operators : (a) Binary (b) Unary.

(a) **Binary Operators** : Binary arithmetic operators are multiply (*), divide (/), add (+), subtract (-), power (**) and modulus (%).

(b) **Unary Operators** : The operators + and - 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.

(ii) **Logical Operators** : Logical operators are logical - and (&&), logical -or (||) and logical - not (!). Operators && and || are binary operator operator ! is a unary operator.

(iii) **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 return a logical value of 1 if the expression is true and 0 if the expression is false.

(iv) **Equality Operators** : Equality operators are logical equality (==), logical inequality (!=), care equality (==) and care in equality (!=) when used in an expression, equality operators return logical nature 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.

(v) **Bitwise Operators** : Bitwise operators are negation (~) and (&) or (1), X or (^), X nor (~^). 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.

(vi) **Reduction Operators** : Reduction operators and (&), nand (~&), or (1) nur (~1), Xor (^), and X nor (~^, ~^~) reduction operators take only one operand. Reduction operators perform a bitwise operation on a single vector operand and yield a 1-bit result.

Q. 2. Write VHDL code for 4 : 1 multiplexer using :

(a) Case statement

(b) IF then else statement

(c) Process

Ans. VHDL code for 4 : 1 Multiplexer :

(a) **Case Statement** : The case statement can also act like a many-to-one multiplexer. To understand this, let us model the 4 to 1 multiplexer module mux_to_1 (out, i0, i1, i2, i3, i4, s1, s0); // port declarations from the I/O diagram output out;

```
input i0, i1, i2, i3;
input s1, s0;
reg out;
always @(s1 or s0 or i0 or i1 or i2 or i3)
```

Case ({s1, s0}) // switch based on concatenation of control signals,

```
2'd0 : out = i0;
2'd1 : out = i1;
2'd2 : out = i2;
2'd3 : out = i3;
default : $display ("Invalid control signals");
```

```
end case
end module
```

(b) If Then Else Statement :

```
// Type 1 statements
if (lock) buffer = data;
if (enable) out = in;
// Type 2 statement
if (number_queued < Max_Q_Depth)
begin
    data_queue = data;
    number_queued = number_queued + 1;
end
```

(c) Process :

```
Entity 4 : 1 multiplexer IS
    GENERIC (td_reset, td_in : Time := 4NS);
    PORT (rest, div, clk, INB, F_qout; BUFFER BIT := '0');
    END mux;

ARCHITECTURE behaving at OF Mux IS
    BEGIN
        PROCESS (CLK)
            BEGIN
                IF (Clk = '0' AND Clk EVENT) THEN
                    If reset = '1' THEN
                        q_out C = '0' AFTER + d_reset;
                    ELSE
                        qout <= din AFTER + d_in;
                    END IF;
                END IF;
            END PROCESS;
        END behavioural;
```

Q. 3. Using structural modelling style, write VHDC code for :

(a) Full adder

(b) 16 : 1 multiplexer.

Ans. (a) Full Adder :

```
library ieee;
use ieee.std_logic_1164.all;
```

```
use ieee.std_logic_arith. all;
use ieee.std_logic_unsigned. all;
entity Adder is
generic (n : natural := 2);
port C A : in std_logic_vector
(n-1 down to 0);
B : in std_logic_vector (n-1 down to 0);
carry : out std_logic;
sum out std_logic_vector (n-1 down to 1)
);
end ADDER;
```

Architecture below of ADDER is

-- define a temporary signal to store the result.

Signal result : std_logic_vector (n down to 0);

```
begin
-- the 3rd bit should be carry.
Result <= ('0' & A) + ('0' & B);
sum <= result (n-1 down to 0);
carry <= result (n);
end behaviour;
```

(b) 16 : 1 multiplexer :

```
Library ieee;
use ieee.std_logic_1164 all;
entity Mux is
Port C 13 : in std_logic_vector (2 down to 0);
12 : in std_logic_vector (2 down to 0);
11 : in std_logic_vector (2 down to 0);
10 : in std_logic_vector (2 down to 0);
5 : in std_logic_vector (1 down to 0);
0 : out std_logic_vector (2 down to 0);
};
end mux
architecture behaved of mux is
begin
process (13, 12, 11, 10, 5)
```

Use case statement:

Case s is

```

When "00" => 0 <= 10;
When "01" => 0 <= 11;
When "10" => 0 <= 12;
When "11" => 0 <= 13;
When others => 0 <= 13;
When others => 0 < "zzz";
end case;
end process;
end behind;
architecture behv z of mux is
begin
----- use when ..... else statement.
0 <= 10 when S = "00" else
11 when S = "01" else
12 when S = "10" else
13 when S = "11" else
"zzz";
end behv 2;

```

Q. 4. Differentiate between array and loops. Explain with the help of examples.

Ans. Arrays : Arrays are allowed in verilog for reg, integer, time real, relative and vector register data types. Multi-dimensional arrays can also be declared with any number of dimensions. Array of nets can also be used to connect ports of generated instances. Each element of the array can be used in the same fashion as a scalar or vector net.

Examples of assignments to elements of arrays discussed :

```

count[5] = 0; //Reset 5th element of array of count variables
chk-point[100] = 0; // Reset 100th time check point value
port_id[3] = 0; //Reset 3rd element (a 5-bit value) of port_id array.
matrix[1][0] = 33559; //set value of element indexed by
[1][0] to 33559 array_4d[0][0][0][15:0] = 0; // clear bits 15:0 of the
register // accessed by indices [0][0][0][0]
port_id = 0; // illegal syntax. Attempt to write the entire array.
matrix[1] = 0; //Illegal syntax-Attempt to write [1][0]...[1][255].

```

Loops : There are four types of looping statement, while, for, repeat and forever. The syntax of these loops is very similar to the syntax of loops in the C programming language. All looping statements can appear only inside an initial or always block.

Example: Forever loop.

```
// Example 1 : Clock generation
```

```
// Use forever loop instead of always block reg. clock;
initial
begin
    clock = '1' b0i
    Forever #.10 clock = ~ clock // clock
    with period of 20 units
end

// Example 2; synchronize two register values at every positive edge of
// clock
reg clock;
reg x, y;
initial
forever @ (posedge clock) x = y;
```

Q. 5. Write short notes on :

(a) Functions and procedures

(b) Sub program over loading.

Ans. (i) Functions : Functions are declared with the keywords function, and end function. Functions are used if all of the following conditions are true for the procedure.

- * There are no delay, timing or event control constructs in the procedure.
- * The procedure returns a single value.
- * There is at least one input argument.
- * There are no output or in out argument.
- * There are no nonblocking assignments.

Automatic (Recursive) Functions :

Functions are normally used non-recursively. If a function is called concurrently from two locations.

Constant Functions : A constant function is a regular verilog HDL function, but with certain restrictions. These functions can be used to reference complex values and can be used instead of constants.

Signed Functions : Signed functions allow signed operations to be performed on the function return value.

Example:

```
Module to;
--
// signed function declaration
// Returns a 64 bit signed value
Function signed [63 : 0] compute-signed (input [63 : 0] vector);
--
--
```

```

end function
--
// call to the signed function from the higher module.
If (compute_signed (vector) < -3)
begin
--
end
--
end module

```

(ii) Procedure : The value placed on a variable will remain unchanged until another procedural assignment updates the variable with a different value. Where one assignment statement can cause the value of the right-hand-side expression to be continuously placed onto the left-hand-side net.

- * A reg., integer, real or time register variable or a memory element.
- * A bit select of these variables.
- * A part select of these variables.
- * A concatenation of any of the above.

(h) Subprogram Over Loading : Overloading is a useful mechanism for using the same name for subprograms that perform the same operations on data of different types. VHDL allows overloading of user-defined subprograms, standard functions and operators.

Example : For AND OR and inversion operations in the git logic value system. As before, the high-impedance 'Z' value is treated as a (1) by all three functions.

a :	0	1	Z	X
b :	0	0	0	0
1	0	1	1	x
Z	0	1	1	x
X	0	1	x	x

$$Z = a.b$$

Type qit is ('0', '1', 'Z', 'X')

Type qit_2d is ARRAY (qit, qit) of qit;

Type qit_2d ARRAY (qit) of qit.

--

FUNCTION "AND" (a, b : qit) RETURN qit;

FUNCTION "OR" (a, bi qit) RETURN qit;

a :	0	1	Z	X
b :	0	1	1	x
1	1	1	1	1
Z	1	1	1	1
X	x	1	1	x

$$Z = a + b$$

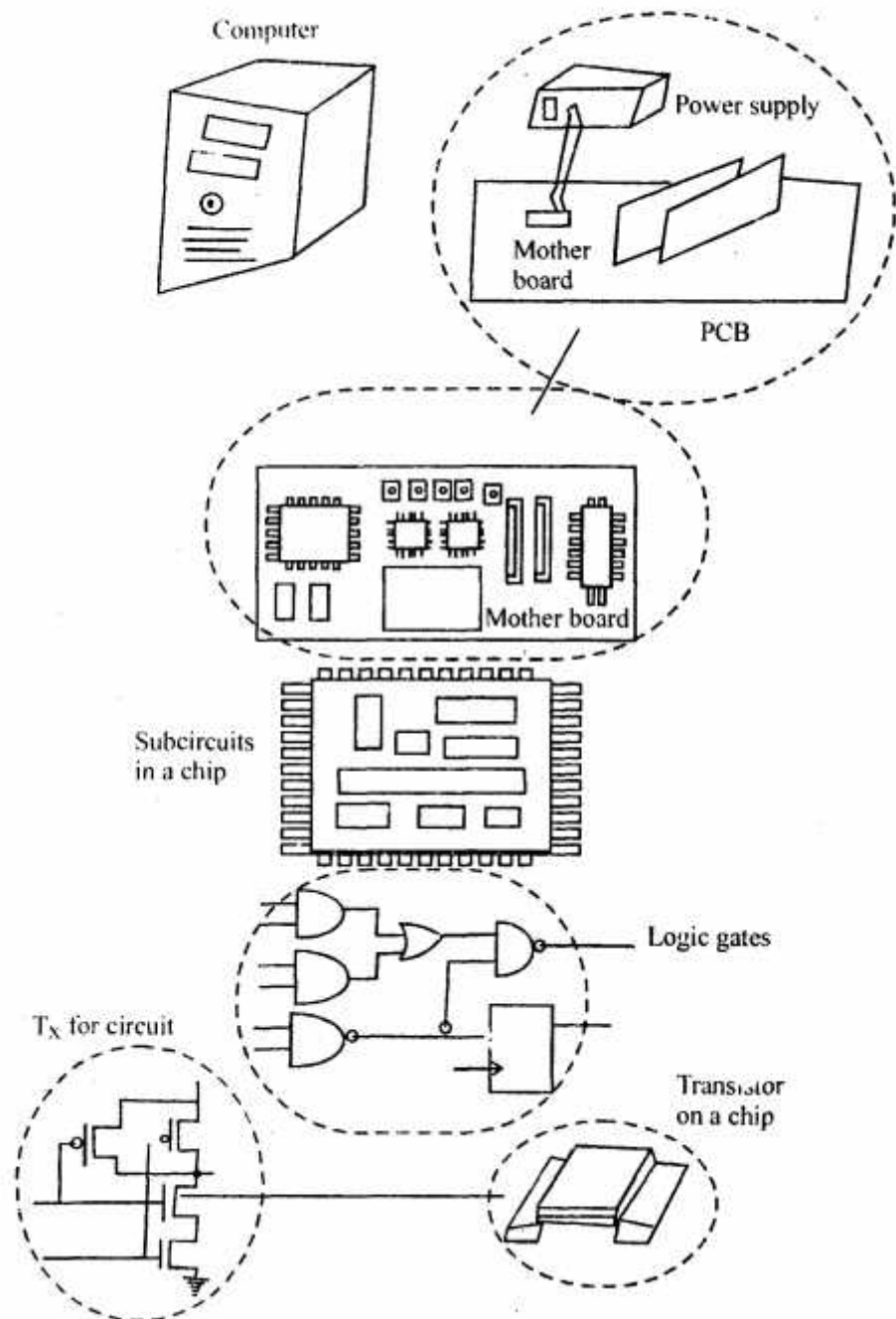
a :	1
1	0
Z	0
X	x

$$Z = a'$$

FUNCTION "NOT" (a : qit) RETURN qit;

Q. 6. (a) Discuss briefly basic components of micro computer system.

Ans. Basic Components of Micro Computer System :



To understand the role that logic circuits play in digital systems, consider the structure of a typical computer. The computer case houses a number of printed circuit boards (PCBs), a power supply and storage units like a hard disk and DVD or CD-ROM devices. Each unit is plugged into a main PCB called the motherboard. The mother board holds several integrated circuit chips and it provides slots for connecting other PCBs. such as audio, video and network boards.

The structure of an integrated circuit chip. The chip comprises a number of subcircuits which are interconnected to build the complete circuit. Examples of subcircuits are those that perform arithmetic operations, store data or control the flow of data. Each of these subcircuits is a logic circuit. A logic circuit comprises a network of connected logic gates. Each logic gate performs a very simple function and more complex operations are realised by connecting gates together. Logic gates are built with transistors which in turn are implemented by fabricating various layers of materials on a silicon chip. The circuit is specified, how the circuits are designed to minimum cost or maximum speed of operation and how the circuits can be tested to ensure correct operation.

Q. 6. (b) Write VHDL code for :

(i) JK Flip Flop

(ii) ALU.

Ans. (i) JK Flip Flop VHDL Code :

```
entity JK_FF is
    port C clock :      in std_logic;
        J, K :          in std_logic;
        Select :        in std_logic;
        Q, Q bar        out std_logic;
);
end Jk_FF;
architecture bev of JK_FF is
    -- define the useful signals here
    signal state : std_logic;
        signal input : std_logic_vector (1 down to 0);
    begin
        -- combine inputs into vector
        input <= J & K;
        P : process (clock, reset) is begin
            if(reset = '1') then
                state <= '0';
            else if(rising_edge(clock)) then
                -- compare to the truth table case (input) is
                when "11" =>
```

```
state <= not state;
when "10" =>
state <= '1';
when "01" =>
state <= '0';
null;
end case;
end if;
end process;
-- concurrent statements
Q <= state;
Q bar C = not state
end behv;
```

(ii) ALU :

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.std_logic_arith.all;
```

entity ALU is

```
port C      A : in std_logic_vector (1 down to 0);
            B : in std_logic_vector (1 down to 0);
            Sel : in std_logic_vector (1 down to 0);
            Res : Out std_logic_vector (1 down to 0);
```

);

end ALU;

architecture behv of ALU is begin

process (A, B, xes)

begin

-- use case statement to achieve

-- different operations of ALU

Case xes is

```
when "00" =>
    Res C = A + B;
When "01" =>
```

```

Res C=[ A + (not B) + 1;
When "10" =>
    Res C = A and B;
When "11" =>
    Res C = A or B;
When others =>
    Res C = "XX";
end case;
end process
end behv

```

Q. 7. (a) Discuss in detail about various types of PLDS.

Ans. PLD's (Programmable Logic Design) has two types :

- (i) Simple PLDs'
- (ii) Complex Programmable logic devices (CPLDs)

(i) **Simple PLD's** : Some of the major manufacturers of SPLD's along with their some of the SPLD products and their www locations.

Manufacturer	SPLD Products	WWW Locator
Altera	Classic	http://www.altera.com
Atmel	PAL	http://www.atmel.com
Cypress	PAL	http://www.cypress.com
Lattice	CAL	http://www.lattice-semi.com
Philips	PLA, PAL	http://www.philips.com
Vantis	PAL	http://www.vantis.com

(ii) **Complex Programmable Logic Devices (CPLD's)** :

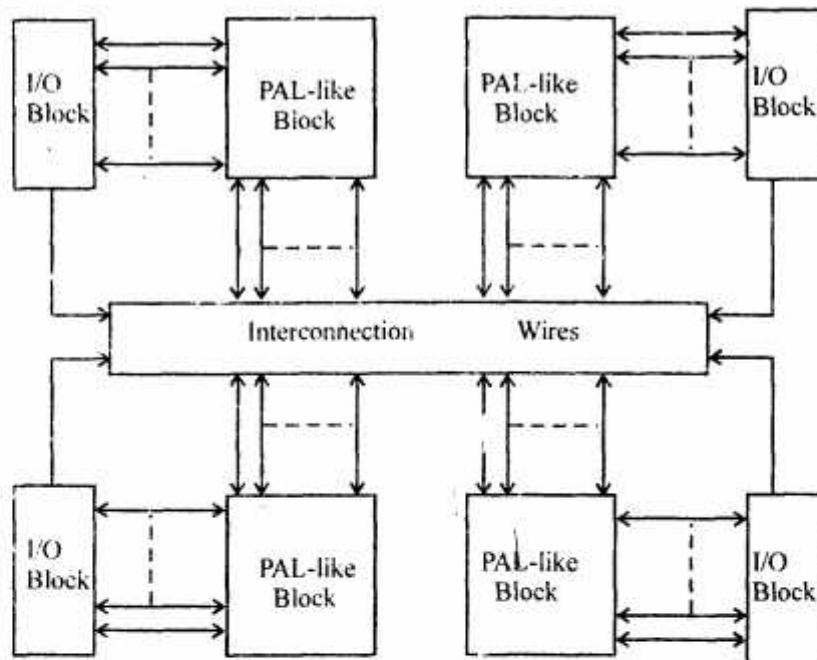
Manufacturer	CPLD Products	www.Locator
Altera	Max 3000, 7000 & 9000	http://www.Altera.com
Atmel	ATF, ATV	http://www.admul.com
Cypus	FLASH 370, Ultra 37000	http://www.cypress.com
Lattice	isp [S] 1000 to 8000	http://www.lattice.com
Philips	XPLA	http://www.philips.com
Vantis	MACH 1 to 5	http://www.vante's.com
Xilinx	XC9500	http://www.xilinx.com

Have 32 total number of inputs and outputs only.

For implementation of circuits that require more inputs and outputs than that are available in a single

SPLD chip, either multiple SPLD chips, either multiple SPLD chips can be employed. The complexity of any digital IC chip can be specified in terms of number of equivalent 2 input NAND gates.

Block Diagram :



Programming : PLDs, SPLD's and CPLD, are implemented using electrically erasable programmable read-only memory (EEPROM) technology. SPLD's chip have small number of pins and can therefore be taken out of the circuit board, without much of inconvenient and put in a programming unit.

CPLD's have large number of pins on the chip package and there pins are fragile and easily bent.

Packaging : CPLD's have a large number of pins, making it impractical to use dual-in line purchasing (DIP). Some of the commonly used packages for CPLDs are :

Plastic-Leaded Chip Carrier (PLCC) : A PLCC package has pins on all the four sides that 'wrap around' the edges of the chip a rather than extending straight down as in the case of a DIP.

Quad Flat Pack (QFP) : A QFP package also has pins on all four sides like a PLCC package, but with pins extending outward from the package with a downward-curving shape.

Ceramic Pin Grid Array (PGA) : It has pins extending straight outwards from the bottom of the package in a grid pattern. It can accommodate a few hundred pins in total.

Q. 7. (b) What is a test bench and what is the purpose of writing a test bench?

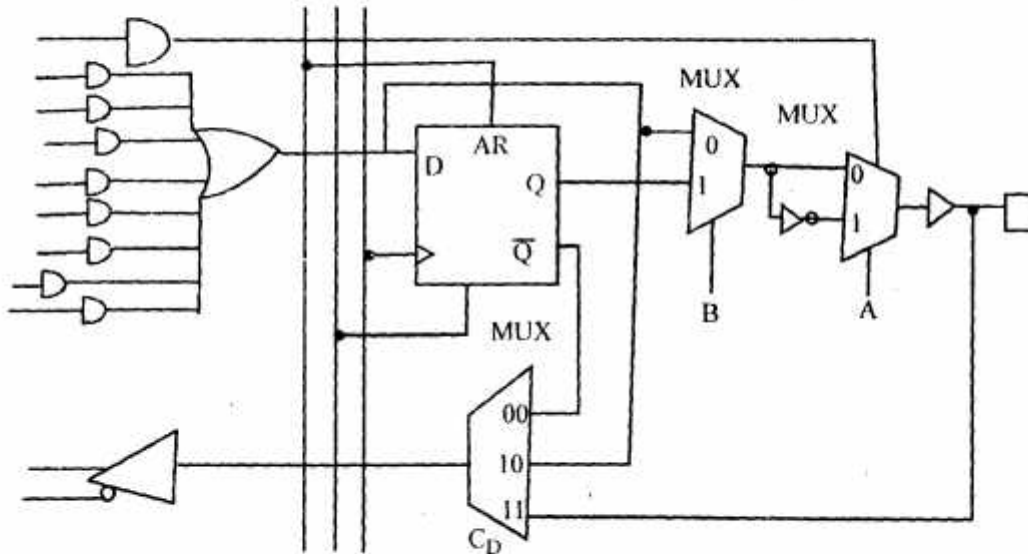
Ans. Test Bench : Testing the nibble-comparator involves generating a test bench description and using it to provide stimulated to the input ports of the 4-bit comparator. A test bench must contain the circuit under test and should have sources for providing data to its inputs. Containment of the nibble-comparator as well as application of waveforms to its inputs can be modeled in VHDL. Development of a test bench for the comparator circuit requires the use of language constructs that are generally not considered to be at the structural level. In order to stay with in the scope. A simple test bench that only requires the use of signal assignment and

component instantaneous.

Q. 8. Write short notes on (any two) :

- (a) PEEL
- (b) Delays
- (c) FPGAs.

Ans. (a) PEEL : Programmable Electrically Erasable Logic (PEEL) devices.

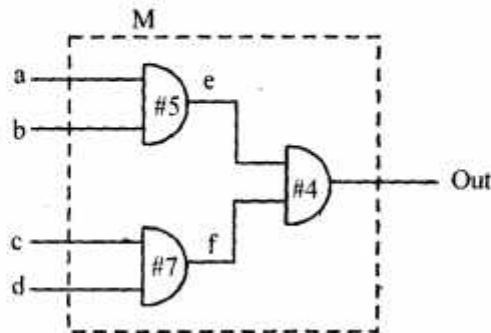


(b) Delays : In real circuits, logic gates have delays associated with them. Pin-to-pin delays can also be specified in verilog.

There are three types of delay models :

- (i) Distributed
- (ii) Lumped
- (iii) Pin-to-pin delays.

(i) Distributed :



Specified on a per element basis, delay values are assigned to individual elements in the circuit.

An Example : Assigned delay values to individual gates or by using delay values in individual assignment statements. When input of any gate change, the output of the gate changes after the delay value specified.

// Distributed delays in gate-level module.

Module M (out, a, b, c, d);

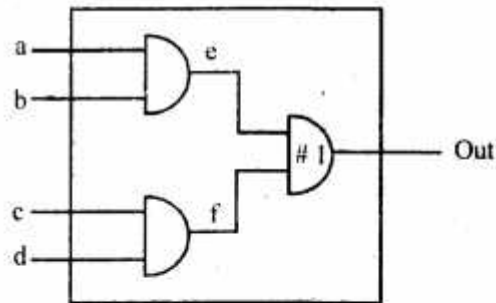
output out;

input a, b, c, d;

wire e, f;

// Delay is distributed to each gate and # 5 at (e, a, b);

(ii) Lumped Delay :



Lumped delays are specified on a per module basis. They can be specified as a single delay on the output gate of the module. The cumulative delay of all paths is lumped at one location.

// Lumped delay model module M (out, a, b, c, d);

Output out;

Input a, b, c, d;

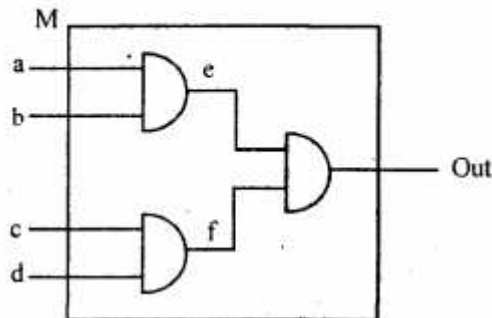
Wire a, f;

and e1 (e, a, b);

and a2 (f, c, d);

and # // a3 (out, e, f); // delay only on the output gate end module.

(iii) Pin-to-Pin Delays :



Delay are assigned individually to paths from each input to each output. Thus, delay can be separately specified for each input output path.

(c) FPGAs (Field Programmable Gate Array) : To increase the effective size and to add more functionality in a single programmable device, alternative architectures have been developed which are known as FPGA's.

An FPGA is composed of a number of relatively independent configurable logic modules, configurable Input/output cells and programmable interconnection paths.

