## EXPERIMENTNO–03

### DesignVerilogHDLtoimplementsimplecircuitsusingstructural,Dataflowand Behavioural model.

**Verilogcode:**

|  |  |  |
| --- | --- | --- |
| **Structural-** | **Data Flow-** | **BehaviouralModel-** |
| module logicg(a, b, y0, y1, y2, y3, y4, y5, y6);  input a, b;  output y0, y1, y2, y3, y4, y5, y6;  not g1(y0, a);  and g2(y1, a, b);  or g3(y2, a, b);  nand g4(y3, a, b);  nor g5(y4, a, b);  xor g6(y5, a, b);  xnor g7(y6, a, b);  endmodule | module ldataflow(a, b, y0, y1, y2, y3, y4, y5, y6);  input a, b;  output y0, y1, y2, y3, y4, y5, y6;  assign y0 = !a;  assign y1 = a & b;  assign y2 = a | b;  assign y3 = !(a & b);  assign y4 = !(a | b);  assign y5 = a ^ b;  assign y6 = !(a ^ b);  endmodule | module logicgB(a,b, y0, y1, y2, y3, y4, y5, y6);  input a, b;  output reg y0, y1, y2, y3, y4, y5, y6;  always @(\*)  y0 = !a;  y1 = a & b;  y2 = a | b;  y3 = !(a & b);  y4 = !(a | b);  y5 = a ^ b;  y6 = !(a ^ b);  end  endmodule |

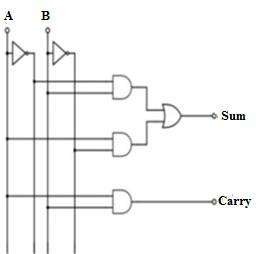
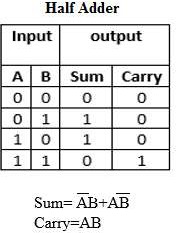
|  |
| --- |
|  |

## EXPERIMENTNO–04

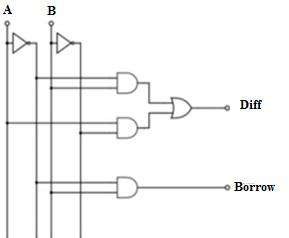
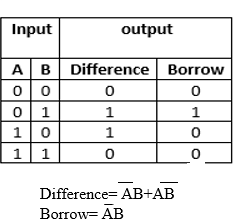
### Design Verilog HDL to implement Binary Adder-Subtractor – Half and FullAdder, Half and Full Subtractor.

### UsingBasic Gates

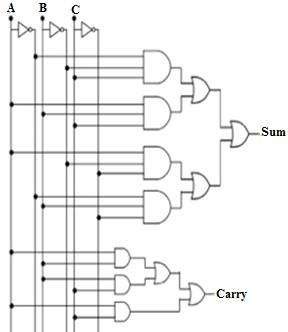
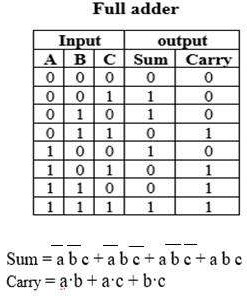
**Half Adder**

****

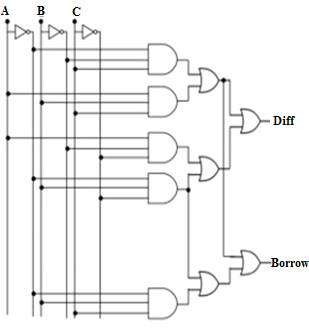
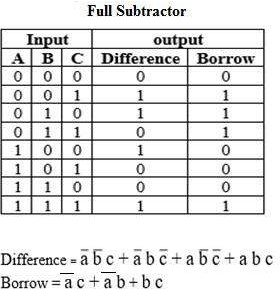
**HalfSubtractor**

****

**FullAdder**

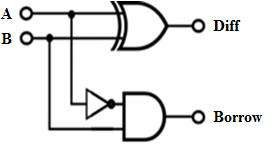
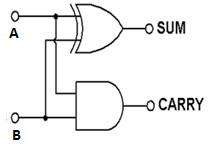
****

### FullSubtractor

****

**UsingEX-OR Gates**

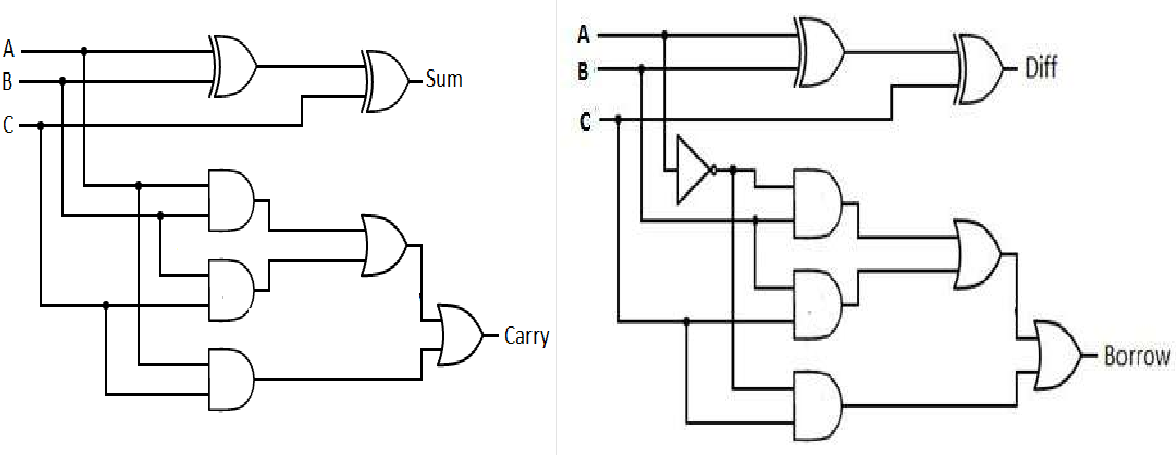
**Half Adder HalfSubtractor**

****

**Sum=A**⨁**B Diff=A**⨁**B**

**Carry=AB Borrow=A’B**

**FullAdder FullSubtractor**

****

**Sum=A**⨁**B**⨁**C Diff=A**⨁**B**⨁**C**

**Carry=AB+BC+AC Borrow=A’B+BC+A’C**

**Verilogcode:**

|  |  |
| --- | --- |
| **Half Adder-** | **Half Substractor-** |
| module halfadder(a,b,sum,carry);  input a, b,  output sum, carry  assign sum = a ^ b;  assign carry = a & b;  endmodule | module HS(a,b,diff,borrow); input a, b,  output diff, borrow  assign diff = a ^ b; assign borrow = ~a & b; endmodule |
|  |
|  |
|  |
|  |
|  |
| **Full Adder-** | **Full Substractor-** |
| module FA(a,b,c,sum,carry);  input a, b, c,  output sum, carry  assign sum = a ^ b ^ c;  assign carry = (a & b) | (b & c) | (c & a); endmodule | module FS(a,b,c,diff,borrow);  input a, b, c,  output diff, borrow  assign diff = a ^ b ^ c;  assign borrow = (~a & b) | (b & c) | (~a & c); endmodule |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

## EXPERIMENTNO–05

### Design Verilog HDL to implement Decimaladder

**Aim:**ToDesign VerilogHDLto implementDecimal adder.

**Theory:**BCDorBinarycodeddecimalisawayofrepresenting decimaldigitsinbinaryform. Generally 4 bits are used to represent values 0 to 9.

|  |  |
| --- | --- |
| **Decimal Digit** | **BCD 8421** |
| 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 |

|  |  |
| --- | --- |
| **Decimal Digit** | **BCD 8421** |
| 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 |

ABCDadder,takesintwoBCDnumbersasinputs,andoutputsaBCDdigitalongwithacarryoutput. If the sum of the BCD digits is less than or equal to 9, then we don't have a problem. But if its greater than9,wehavetoconvertitintoBCDformat.Thisisdonebyadding6tothesumandtakingonlythe least significant 4 bits. The MSB bit is output as carry.

ConsiderthebelowBCD addition:

1001 +1000 =10001

9 + 8 = 17

Theoutput 17 is morethan 9. So weadd 6 to it. So weget,

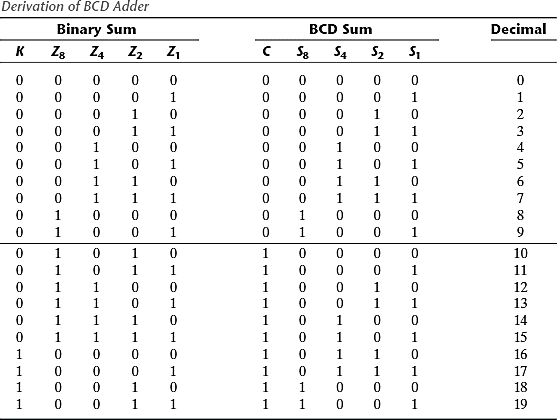
17+6=23 (inbinary 23 is 10111)

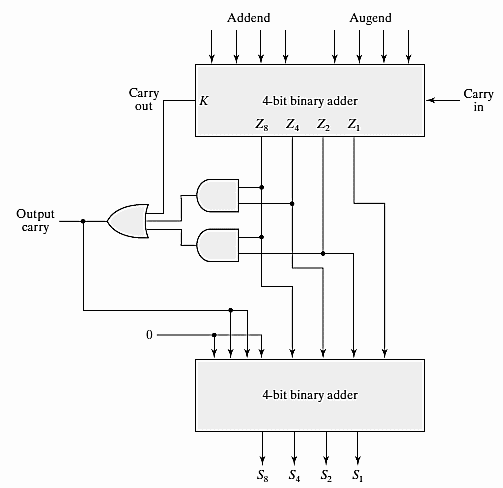
Now the least significant 4 bits (which is "0111")represent the units digit and the MSB(4th bit which is '1') bit represents the tens digit. Sincethe range of input is 0 to 9, the maximum output is 18. If you consider a carry it becomes 19. This means at the output side we need a 4 bit sum and a 1 bit carry to represent the most significant digit. These binary numbers are listed below

Theconditionfora correction andanoutputcarry can be expressed bythe BooleanFunction

***C*=*K*+*Z*8*Z*4+ *Z*8*Z*2**

When C = 1, it is necessary to add 0110 to the binary sum and provide an output carry for the next stage. A BCD adder that adds two BCD digits and produces a sum digit in BCD is shown in Figure below.





**Logic Diagram:**

**module deciadder(a, b, carry\_in, sum, carry);**

**input [3:0] a, b;**

**input carry\_in;**

**output [3:0] sum;**

**output carry;**

**reg [4:0] sum\_temp;**

**reg [3:0] sum;**

**reg carry;**

**always @(a, b, carry\_in)**

**begin**

**sum\_temp = a + b + carry\_in;**

**if (sum\_temp > 9) begin**

**sum\_temp = sum\_temp + 6;**

**carry = 1;**

**sum = sum\_temp[3:0];**

**end else begin**

**carry = 0;**

**sum = sum\_temp[3:0];**

**end**

**end**

**endmodule**

## EXPERIMENTNO–06

### DesignVerilogprogramtoimplementDifferenttypesofmultiplexerlike2:1,4:1 and 8:1.

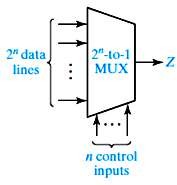
**Aim:**DesignVerilogprogramto implementDifferenttypesofmultiplexerslike2:1,4:1and 8:1.

### Objectives:

1. DevelopVerilogprogramfor2:1,4:1,8:1multiplexerdesign.
2. Validatethroughsimulationforaccuracyand efficiencyincircuitimplementation.

### Theory:

* + Amultiplexer(ordataselector,abbreviatedasMUX)hasagroupofdatainputs(2n)andagroup of control inputs (n) (also called as select inputs).
  + Thecontrolinputsareusedtoselectoneofthedatainputsandconnectittotheoutputterminal.



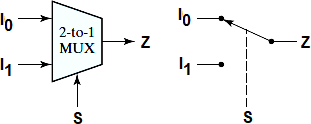
#### Figure6.1: General blockdiagram of 2n: 1multiplexer

* + Ageneralblock diagramof 2n : 1multiplexeris shownin the fig.3.6.
  + A2to1multiplexer requires1selectinput,4to1multiplexersrequire2selectinputsand8to

1multiplexersrequire3selectinputs.

**2:1Multiplexer**

* + A2 to1 multiplexerhas 2 datainputs, 1select inputand 1 output.



#### Figure6.2:2 to1Multiplexerand Switch Analog

* + When the select (control) input A is 0, the switch is in the upper position and the MUX output is Z = I0.
  + When the select (control) input A is 1, the switch is in the lower position and the MUX output is Z = I1.
  + Inotherwords,aMUXactslikeaswitchthatselectsoneofthedatainputs(I0orI1)and transmits it to the output.

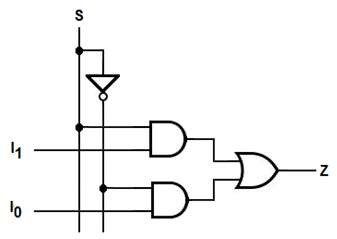
#### TruthTable:

|  |  |
| --- | --- |
| **Select(S) Ou** | **tput (Z)** |
| 0 | I0 |
| 1 | I1 |

* + Thelogicequation forthe2-to-1MUX can be written as:

𝐙=𝐒̅𝐈𝟎+𝐒𝐈𝟏

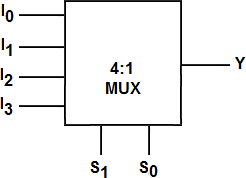
**Logic Diagram:**



**Figure6.3:Logic diagram of 2:1 Multiplexer**

### 4:1Multiplexer

* + A4 to1 multiplexerhas 4 datainputs, 2select inputsand 1 output.
  + The 4 to 1 MUX acts like a four-position switch that transmits one of the four inputs to the output.



#### Figure6.4:4 to 1Multiplexer

* + Twoselect(control)inputs(S1andS0)areneededtoselectoneofthefourinputs.Ifthecontrol inputsareS1S0=00,theoutputisI0;similarly,forthecontrolinputs01,10,and11giveoutputs of I1, I2, and I3, respectively.

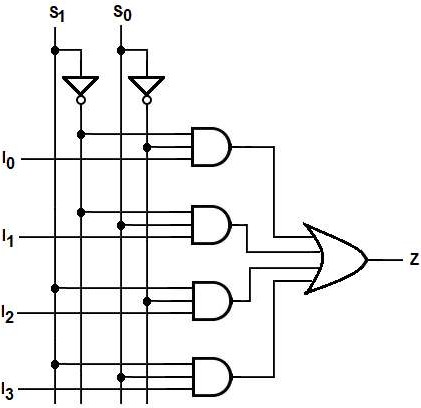
#### TruthTable:

|  |  |  |
| --- | --- | --- |
| **elect(S1)** | **Select(S0)** | **Output(Z)** |
| 0 | 0 | I0 |
| 0 | 1 | I1 |
| 1 | 0 | I2 |
| 1 | 1 | I3 |

* + Thelogicequationfor4-to-1 multiplexercanbewritten as

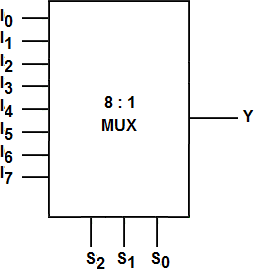
𝐙=𝐒𝟏𝐒𝟎𝐈𝟎+𝐒𝟏𝐒𝟎𝐈𝟏+𝐒𝟏𝐒𝟎𝐈𝟐+𝐒𝟏𝐒𝟎𝐈𝟑

**Logic Diagram:**



### 8:1Multiplexer

* + An8to1multiplexerhas8datainputs,3selectinputsand1output. 8to1MUXselectsoneof eight data inputs using three select inputs



#### Figure6.5:8 to 1Multiplexer

* + The 8-to-1 MUX acts like an eight-position switch that transmits one of the eight inputs to the output.

#### TruthTable:

|  |  |  |  |
| --- | --- | --- | --- |
| **elect(S2)** | **Select(S1)** | **Select(S0)** | **Output(Z)** |
| 0 | 0 | 0 | I0 |
| 0 | 0 | 1 | I1 |
| 0 | 1 | 0 | I2 |
| 0 | 1 | 1 | I3 |
| 1 | 0 | 0 | I4 |
| 1 | 0 | 1 | I5 |
| 1 | 1 | 0 | I6 |
| 1 | 1 | 1 | I7 |

**Verilogcode:**

|  |  |  |
| --- | --- | --- |
| **2:1 MUX** | **4:1 MUX** | **8:1 MUX** |
| module two\_to\_one\_MUX (I0, I1, S, Y);  input I0;  input I1;  input S;  output Y;    wire S\_bar, w1, w2;  not g1(S\_bar, S);  and g2(w1, I0, S\_bar);  and g3(w2, I1, S);  or g4(Y, w1, w2);  endmodule | module four\_to\_one\_MUX(I1, I2, I3, I4, S1, S2, Y);  input I1, I2, I3, I4;  input S1, S2;  output Y;  assign Y = ((!S1) & (!S2) & I1) |  ((!S1) & (S2) & I2) |  ((S1) & (!S2) & I3) |  ((S1) & (S2) & I4);  endmodule | module eight\_to\_one\_MUX (I1, I2, I3, I4, I5, I6, I7, I8, S1, S2, S3, Y);  input I1, I2, I3, I4, I5, I6, I7, I8; input S1, S2, S3; output Y;  assign Y = ((!S1) & (!S2) & (!S3) & I1) |  ((!S1) & (!S2) & S3 & I2) |  ((!S1) & S2 & (!S3) & I3) |  ((!S1) & S2 & S3 & I4) |  (S1 & (!S2) & (!S3) & I5) |  (S1 & (!S2) & S3 & I6) |  (S1 & S2 & (!S3) & I7) |  (S1 & S2 & S3 & I8);  endmodule |
|  |  |

**Aim:**

## EXPERIMENTNO–07

### DesignVerilogprogramtoimplementtypesofDe-Multiplexer.

ToDesignVerilogprogramtoimplementDifferent typesofDe-multiplexerlike1:2,1:4and 1:8.

### Objectives:

1. ImplementVerilogprogramfor1:2,1:4,1:8Demultiplexer designs.
2. Verifyfunctionalitythroughsimulationforaccuracyandperformance assessment.

### Theory:

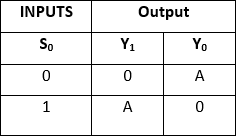
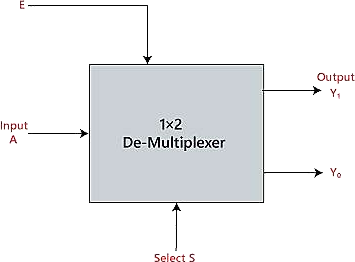
A De-multiplexer is a combinational circuit that has only 1 input line and 2Noutput lines. Simply, the multiplexeris asingle-input and multi-output combinational circuit. Theinformation is receivedfrom the single input lines and directed to the output line. On the basis of the values of the selection lines, the input will be connected to one of these outputs. De-multiplexer is opposite to the multiplexer.

Unlikeencoderanddecoder,therearenselectionlinesand2n outputs.So,thereisatotalof2npossible combinations of inputs. De-multiplexer is also treated as **De-mux**.

Therearevarioustypesof De-multiplexerwhichareas follows:

**1×2 De-multiplexer:** In the 1 to 2 De-multiplexer, there are only two outputs, i.e., Y0, and Y1, 1 selectionlines,i.e.,S0,andsingleinput,i.e.,A.Basedontheselectionvalue,theinputwillbeconnected to one of the outputs. The block diagram and the truth table of the 1**×**2 multiplexer are given below.

### Block Diagram: Truth Table:

****

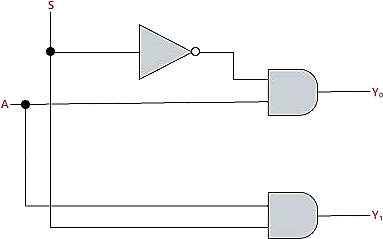
Thelogical expressionoftheterm Yis as follows:

Y0=S0'.A Y1=S0.A

Logicalcircuitoftheaboveexpressionsisgiven below:

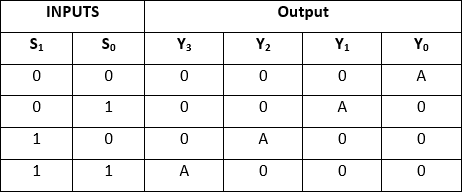
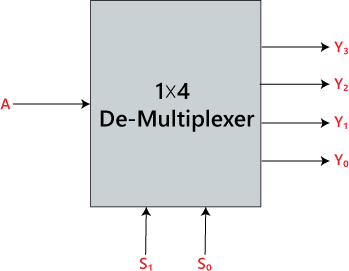
**TruthTable:**

**Block Diagram:**



**1×4 De-multiplexer:** In 1 to 4 De-multiplexer, there are total of four outputs, i.e., Y0, Y1, Y2, and Y3,2selectionlines,i.e.,S0andS1andsingleinput,i.e.,A.Basedonthecombinationofinputswhich are present at the selection lines S0 and S1, the input be connected to one of the outputs. The block diagram and the truth table of the 1**×**4 multiplexer is given below.

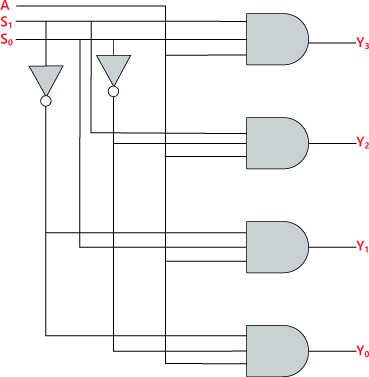
### Block Diagram: Truth Table:

****

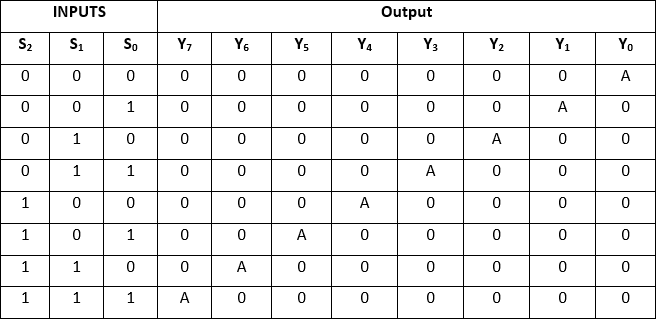
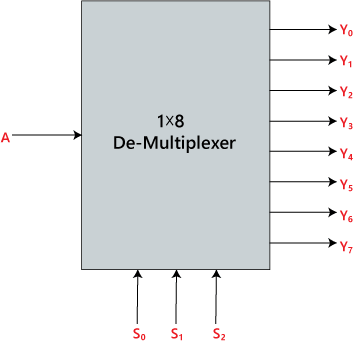
Thelogical expressionoftheterm Yis as follows:

**Y0=S1'S0'A y1=S1' S0A y2=S1S0' A y3=S1 S0 A**

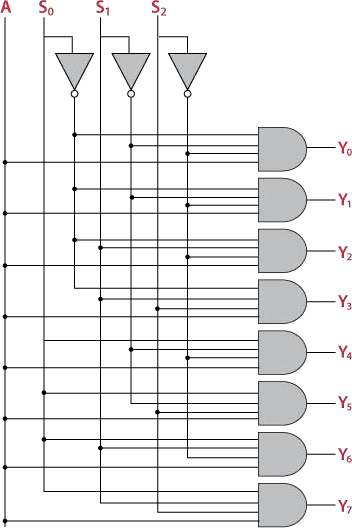
Logicalcircuitoftheaboveexpressionsisgiven below:



**1×8 De-multiplexer:** In 1 to 8 De-multiplexer, there are total of eight outputs, i.e., Y0, Y1, Y2, Y3, Y4, Y5, Y6, and Y7, 3 selection lines, i.e., S0, S1and S2and single input, i.e., A. On the basis of the combination of inputs which are present at the selection lines S0, S1and S2, the input will be connected to one of these outputs. The block diagram and the truth table of the 1**×**8 de-multiplexer is given below.



Logicalcircuitoftheaboveexpressionsisgiven below:



**Verilogcode:**

|  |  |  |
| --- | --- | --- |
| **1:2 DEMUX** | **1:4DEMUX** | **1:8DEMUX** |
| module demux12\_BM(S0,A,Y);  input S0;  input A;  output reg [1:0] Y;  always @(A or S0)  begin  case (S0)  1'b0: Y = {A, 1'b0}; 1'b1: Y = {1'b0, A};  endcase  end  endmodule | module one\_four\_demux\_BM(S, A, Y);  input A;  input [1:0] S;  output [3:0] Y;  reg [3:0] Y;  always @(A or S) begin  case(S)  2'b00: Y = {A, 3'b000};  2'b01: Y = {1'b0, A, 2'b00};  2'b10: Y = {2'b00, A, 1'b0};  default: Y = {3'b000, A};  endcase  end  endmodule | module one\_eight\_demux\_BM(S, A, Y);  input A;  input [2:0] S;  output [7:0] Y;  reg [7:0] Y;  always @(A or S) begin  case(S)  3'b000: Y = {A, 7'b0000000};  3'b001: Y = {1'b0, A, 6'b000000};  3'b010: Y = {2'b00, A, 5'b00000};  3'b011: Y = {3'b000, A, 4'b0000};  3'b100: Y = {4'b0000, A, 3'b000};  3'b101: Y = {5'b00000, A, 2'b00};  3'b110: Y = {6'b000000, A, 1'b0};  default: Y = {7'b0000000, A};  endcase  end  endmodule |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

## EXPERIMENTNO–08

### DesignVerilogprogramforimplementingvarioustypesofFlip-FlopssuchasSR, JK and D

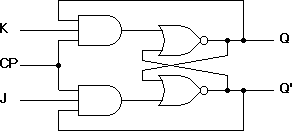
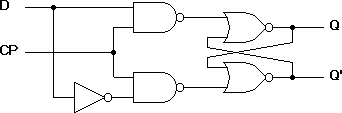
**Table8.1:Flip-flopsandtheirproperties**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Flip- Flop**  **Name** | **Flip-Flop Symbol** | **Characteristic Table** | | | | | **Characteristic Equation** | **ExcitationTable** | | | | | |
|  |  |  | | | | | Q(next)= S + R’Q SR = 0 |  | | | | | |
| SR |  | **S** | **R** | **Q(next)** |  |  | **Q** | **Q(next)** | **S** | **R** |  |
| 0 | 0 | Q | 0 | 0 | 0 | X |
| 0 | 1 | 0 | 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | ? | 1 | 1 | X | 0 |
|  |  |  |  | | |  | Q(next)=JQ’  +K’Q |  |  | | | |  |
| JK |  | **J** | **K** | **Q(next)** |  |  | **Q** | **Q(next)** | **J** | **K** |  |
| 0 | 0 | Q | 0 | 0 | 0 | X |
| 0 | 1 | 0 | 0 | 1 | 1 | X |
| 1 | 0 | 1 | 1 | 0 | X | 1 |
| 1 | 1 | Q’ | 1 | 1 | X | 0 |

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| D |  |  | | | | Q(next)=D |  | **Q** | **Q(next)** | **D** |  |
|  | **D** | **Q(next)** |  | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 0 |
|  | | | | 1 | 1 | 1 |

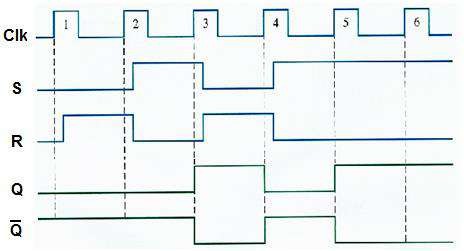
### Logic diagram

**Figure8.1:SR (a)NOR Gatelatch (b) Symbol**

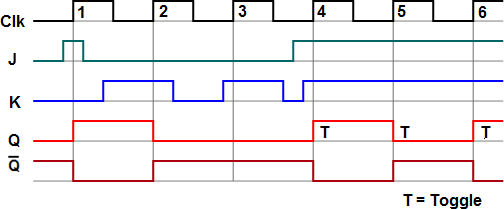
****

### Graphs

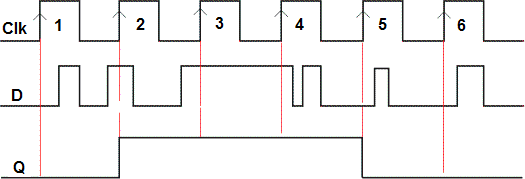
#### Figure8.2:D-FlipFlop Figure8.3:JK FlipFlop

****

**Figure8.4: Timingdiagramof SR Flip-Flop**



**Figure8.5: Timingdiagramof JK Flip-Flop**



**Figure8.6: Timingdiagramof DFlip-Flop**

**Verilogcode:**

|  |  |  |
| --- | --- | --- |
| **1.SRFlip-Flop** | **2.JK FlipFlop** | **3.DFlip Flop** |
| module srlax(S,R,Q,Q\_bar);  input wire S;  input wire R;  output reg Q;  output wire Q\_bar;  always @(S, R);  begin  if (S && ~R);  Q <= 1'b1;  else if (~S && R);  Q <= 1'b0;  else if (S == 0 && R == 0);  Q <= Q;  else if (S == 1 && R == 1);  Q <= 1'bz;  end  assign Q\_bar = ~Q;  endmodule | module JK\_FF(j,k,q,q\_bar);  input wire j;  input wire k;  output reg Q;  output wire Q\_bar;  always @(j, k) begin  if (j && ~k)  Q <= 1'b1;  else if (~j && k)  Q <= 1'b0;  else if (j == 0 && k == 0)  Q <= Q;  else if (j == 1 && k == 1)  Q <= ~Q;  end  assign Q\_bar = ~Q;  endmodule | module D\_FF(d,q,q\_bar);  input wire D;  output reg Q;  output wire Q\_bar;  always @(D) begin  if (D == 0)  Q <= 0;  else  Q <= 1;  end  assign Q\_bar = ~Q;  endmodule |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |