Dataflow model verilog code for simple SOP equation $Y = (AB) + (CD)$

→
```
module SOPequation (A, B, C, D, Y);
input A, B, C, D;
output Y;
assign Y = (A & B) + (C & D);
endmodule
```

Verilog CODE for 4 to 1 Mux using data flow modeling

→
```
module mux4to1 ( A, B, D0, D1, D2, D3, Y);
input A, B, D0, D1, D2, D3;
output Y;
assign Y = A ? (B ? D3 : D2) : (B ? D1 : D0);
endmodule
```

Verilog code for 3 to 8 decoder using data flow modeling

→
```
module decoder (output [0:7]D, input A, B, C, enable);
assign D[0] = (~A & ~B & ~C & enable),
       D[1] = (~A & ~B & C & enable),
       D[2] = (~A & B & ~C & enable),
       D[3] = (~A & B & C & enable),
       D[4] = (A & ~B & ~C & enable),
       D[5] = (A & ~B & C & enable),
       D[6] = (A & B & ~C & enable),
       D[7] = (A & B & C & enable);
endmodule
```

Verilog code for 2to1 Mux using data flow model

→
```
module 2to1Mux ( A, D1, D0, Y);
input A, D1, D0;
output Y;
assign Y = A ? (D1 : D0);
endmodule
```

Verilog code of 2to1 Mux using behavioural model (if-else statements)

→
```
module 2to1Mux (A, D0, D1, Y);
input A, D0, D1;
output Y;
reg Y;
always @ (A or D0 or D1)
    if (A==1) Y = D1;
    else Y = D0;
endmodule
```

Using case statement :-

```
module 2to1Mux(A, D0, D1, Y);
input A, D0, D1;
output Y;
reg Y;
always @ ( A or D0 or D1)
 case (A)
     0 : Y = D0;
     1 : Y = D1;
 endcase
endmodule
```

Verilog code for 4to1mux using behavioural modeling.

→
```
module 4to1mux( A, B, D0, D1, D2, D3, Y);
input A, B, D0, D1, D2, D3;
output Y;
reg Y;
always @ (A or B or D0 or D1 or D2 or D3)
    case({A, B}) // * concatenation of A and B */
    0: Y = D0;
    1: Y = D1;
    2: Y = D2;
    3: Y = D3;
endcase
endmodule
```

Verilog code for demux1to4 using behavioural model / 2 to 4 decod

```
module demux1to4 ( A , D , Y) ;
 input [1:0] A ;
 input D ;
 output [3:0] Y ;
 reg [3:0] Y ;
 always @ (A or D)
    case ({D, A}) // concentation of A and D and D is MSB
       3'b100 : Y = 4'b0001;
       3'b101 : Y = 4'b0010;
       3'b110 : Y = 4'b0100;
       3'b111 : Y = 4'b1000;
        default : Y = 4'b0000;
    endcase
endmodule
```

Verilog code for SRLatch.

```
module SRLatch (S, R , EN , Q);
 input S, R , EN ;
 output q ;
 reg Q ;
 always @ ( EN or S or R)
     if (EN) Q = S | (~R & Q) ;
endmodule
```

D FF positive edge triggerd and negative.

```
module DFFpos (D, C, Q);
input D, C ; // C is clock
output q ;
 reg q ;
always @(posedge C)
  q = D ;
endmodule
```

```
module DFFneg ( D, C , Q);
input D, C ;
output Q ;
reg Q ;
always @(negedge C)
  Q = D ;
endmodule
```

clocked D FF :-

→
```
module DFF (D, EN, Q);
input D, EN; //
output Q;
reg Q;
always @ (EN or D)
  if (EN == 1)  Q = D;
endmodule
```

clocked SR FF :-

→
```
module SRFF (S, EN, R, Q)
input S,R,EN;
output Q;
reg Q;
always @ (EN or S or R)
    if (EN == 1)   Q = S | (~R & Q);
endmodule
```

Verilog code that converts D FF to SR FF

→
```
module SRFFneg(S, R, C, Q);
input S,R, C; // C is clock
output Q;
wire DSR;
assign DSR = S|(~R&Q); // characteristic eqn for SR FF
DFFneg D1(DSR, C, Q); // initiating neg edge trigg D FF
endmodule

module DFFneg(D, C, Q);
input D, C;
output Q;
reg Q;
always @(negedge C)
    Q = D;
endmodule
```