Example1: Write a Verilog description for a **2** to **4** decoder with one high enable as described in section 4.5.3. Use (a) behavioral modeling (b) dataflow modeling .

Solution

(a) Using behavioral modeling:

Note that {] is concatenate operator in Verilog.

```
module decoder(Y3, Y2, Y1, YO, A, B, en);
      //Define inputs and outputs
      input A, B;
      input en;
      reg Y3, Y2, Y1, Y0;
      output Y3, Y2, Y1, Y0;

      always @(A or B or en)
      begin
            //Use behavioral method for decoder
            if (en ==1)
            begin
                  case ({A,B))
                        2'bOO: {Y3,Y2,Yl,Y0} =4'b000l;
                        2'bOl: {Y3,Y2,Yl,Y0) =4'b00l0;
                        2'blO: {Y3,Y2,Yl,Y0} =4'b0l00;
                        2'bll: {Y3,Y2,Yl,Y0] =4'bl000;
                        default: {Y3,Y2,Yl,Y0)= 4'bxxxx;
                  endcase
            end
            if (en ==0)
                  {Y3,Y2,Yl,Y0) =4'bOOOO;
            end
      end
endmodule
```

(b) Using dataflow modeling:

```
module decoder(E, X, Y, Z0, Z1, Z2, Z3);
      output ZO, Z1, Z2, Z3;
      input E, X, Y;
      assign Z0 =E &-X &-Y;
      assign Z1 =E &-X &Y;
      assign Z2 =E &X &-Y;
      assign Z3 =E &X &Y;
endmodule
```

Example2: Write a Verilog description for a 4-bit ripple-carry adder using (a)structural modeling and (b)behavioral modeling.

(a)Using structural modeling:

```
module half-adder (s,c, x, y);
      output s,c;
      input x,y;
      xor (s,x,Y);
      and (c,x,Y);
endmodule

module full-adder (sum, cout, x, y, z);
      output sum,cout;
      input x,y,z;
      wire sl,cl,c2;
      half-adder Bl(sl,cl,x,y);
      half-adder B2 (sum,c2, sl, z);
      or (cout, cl, c2);
endmodule
```

```
module fulladd4(sum, c-out, a, b, c-in);
      output [3:0] sum;
      input [3:0} a, b;
      input c-in;
      output c-out;
      //Internal nets
      wire cl, c2, c3;

      //Instantiate four 1-bit full adders.
      full-adder fa0(sum[0], cl, a[0], b[0], c-in);
      full-adder fa1 (sum[l], c2, a[l], b[l], cl);
      full-adder fa2(sum[2], c3, a[2], b[2], c2);
      full-adder fa3 (sum[3], c4, a[3], b[3], c3);
endmodule
```

(b) Using behavioral modeling:

```
module adder4 (cout, s, a, b, cin);
      output cout;
      output[3:0] s;
      input [3:0] a, b;
      input cin;
      always @(a or b or cin)
      begin
            {cout, s }=atbtcin;
      end
endmodule
```