Vivado Verilog Practice Codes

// 2 X 4 DECODER

```
module TwoTo4Decoder(
   input In0,
   input In1,
   input Sel,
   output reg Out0,
   output reg Out1,
   output reg Out2,
   output reg Out3
);
   always @(*) begin
      if (~Sel) begin
         Out0 = 1'b1;
         Out1 = 1'b1;
         Out2 = 1'b1;
         Out3 = 1'b1;
      end else begin
         case ({In1, In0})
            2'b00: {Out0, Out1, Out2, Out3} = 4'b0111;
            2'b01: {Out0, Out1, Out2, Out3} = 4'b1011;
            2'b10: {Out0, Out1, Out2, Out3} = 4'b1101;
            2'b11: {Out0, Out1, Out2, Out3} = 4'b1110;
            default: {Out0, Out1, Out2, Out3} = 4'b1111;
         endcase
      end
   end
endmodule
```

```
1   module Twoto4decoder(
2     input [1:0] Inp,
3     input en,
4     output [3:0] outp
5   );
6
7     reg [3:0] out;
8
9     always @(en or Inp) begin
10      if (~en)
11        out <= 4'b1111;
12      else begin
13        case (Inp)
14          2'b00 : out <= 4'b0111;
15          2'b01 : out <= 4'b1011;
16          2'b10 : out <= 4'b1101;
17          2'b11 : out <= 4'b1110;
18          default : out <= 4'b0000;
19        endcase
20      end
21    end
22
23    assign outp = out;
24
25  endmodule
26
```

// 2 x 1 MULTIPLEXER

```
module two_by_one_mux(
   input in0,
   input in1,
   input sel,
   output reg outp
);
   always @(*) begin
      if (sel)
         outp = in1;
      else
         outp = in0;
   end
endmodule
```

// 4 x 1 MULTIPLEXER

```
module four_by_1_Multiplexer(
   input in0, in1, in2, in3,
   input [1:0] sel,
   output reg out
);
   always @(*) begin
      case (sel)
         2'b00: out = in0;
         2'b01: out = in1;
         2'b10: out = in2;
         2'b11: out = in3;
         default: out = 1'b0;
      endcase
   end
endmodule
```

// AND GATE

```
module and_gate(
   input a, b,
   output z
);
   assign z = a & b;
endmodule
```

// FULL ADDER

```
module full_adder(
   input x, y, cin,
   output sum, cout
);
   wire a, c, d;
   xor (a, x, y);
   xor (sum, a, cin);
   and (c, x, y);
   and (d, a, cin);
   or  (cout, c, d);
endmodule
```

// HALF ADDER

```
module half_adder(
   input x, y,
   output sum, cout
);
   xor (sum, x, y);
   and (cout, x, y);
endmodule
```

// OR GATE

```
module or_gate(
   input a, b,
   output z
);
   assign z = a | b;
endmodule
```

// XNOR GATE

```
module xnor_gate(
   input a, b,
   output y
);
   assign y = ~(a ^ b);
endmodule
```

// XOR GATE

```
module xor_gate(
   input a, b,
   output y
);
   assign y = a ^ b;
endmodule
```

// 8-to-3 ENCODER

```
module eight_to_three_encoder(
 input [7:0] in,
 output reg [2:0] out
);
 always @(*) begin
   case (in)
     8'b00000001: out = 3'b000;
     8'b00000010: out = 3'b001;
     8'b00000100: out = 3'b010;
     8'b00001000: out = 3'b011;
     8'b00010000: out = 3'b100;
     8'b00100000: out = 3'b101;
     8'b01000000: out = 3'b110;
     8'b10000000: out = 3'b111;
     default: out = 3'bxxx; // Undefined input
   endcase
 end
endmodule
```