Assignment1

Arnav Yadnopavit
EE24BTECH11007

April 21, 2025

Code 1

```
module encoder(
        input [3:0] in,output reg [1:0] out,output reg valid
);
always @(*) begin
casez (in)
        4'b1zzz: begin out = 2'b11; valid = 1; end
        4'b01zz: begin out = 2'b10; valid = 1; end
        4'b001z: begin out = 2'b01; valid = 1; end
        4'b0001: begin out = 2'b00; valid = 1; end
        default: begin out = 2'b00; valid = 0; end
endcase
end
endmodule
```

Explanation 1

This module implements a 4-to-2 priority encoder using the `casez` statement in Verilog. It accepts a 4-bit input signal named `in`, and produces a 2-bit binary output `out` and a 1-bit `valid` signal.

The encoder assigns priority in the order `in[3]`>`in[2]`>`in[1]`>`in[0]`. That means if multiple inputs are high at the same time, the encoder outputs the position of the highest-order high bit.

The `casez` statement allows "don't care" bits (written as `z`) in pattern matching, which makes it easier to define priority logic. Inside each `casez` branch, `out` is assigned a 2-bit binary value corresponding to the position of the high input, and `valid` is set to 1. If no bits are high, the `default` case sets `valid` to 0 and `out` to 0.

This design is often used in digital circuits where fewer output lines are needed, but input signal priorities must be preserved.

Code 2

```
module counter(
        input clk,input reset,input enable,output reg  [3:0]count
);

always @(posedge clk) begin
        if (reset)
                count<=4'b0000;
        if (enable)
                casez(count)
                        4'bzzz0:begin count[0]<=1;end
                        4'bzz01:begin count[1:0]<=2'b10;end
```

```
                        4'bz011:begin count[2:0]<=3'b100;end
                        4'b0111:begin count[3:0]<=4'b1000;end
                        4'b1111:begin count[3:0]<=4'b0000;end
                endcase
end
endmodule
```

Explanation 2

This module describes a 4-bit up counter with three control inputs: `clk` (clock), `reset` (asynchronous reset), and `enable` (count enable). It produces a 4-bit output `count`.

The counter behavior is governed by a clock edge using a `posedge clk` sensitivity. When `reset` is asserted high, the counter resets to 0. This happens asynchronously on the next positive clock edge.

When `enable` is high, a `casez` block checks the current value of `count` and updates it to the next specific value. Unlike a traditional binary counter using `count <= count + 1`, this implementation manually specifies each state transition using partial bit updates such as `count[1:0] <= 2'b10` or full assignments like `count[3:0] <= 4'b1000`.

The `casez` allows matching specific patterns with don't care bits (`z`), making it efficient for conditionally updating parts of the vector.

This approach offers flexibility in defining custom count sequences. It is useful when specific transitions are desired that deviate from normal binary counting, and where control signals like `enable` and `reset` must be considered.

Code 3

```
module EvenParity(
    input [7:0] data,
    output parity
);
    assign parity = ~^data;
endmodule
```

Explanation 3

This module generates an even parity bit for an 8-bit input. It has a single input vector `data[7:0]` and a single output `parity`.

Even parity ensures that the total number of 1's (including the parity bit itself) is even. To compute this, the code uses the XOR reduction operator `^data`, which applies XOR across all bits of the input vector. The result of `^data` is 1 if the number of 1's is odd, and 0 if even.

Then, the result is inverted using `~` to generate the even parity bit. Thus, if the original data has an odd number of 1's, the parity becomes 1, making the total even. If the original data has an even number of 1's, the parity becomes 0, maintaining even parity.

This module is commonly used in digital communication systems for basic error detection. For codes,testbenches refer to
`https://github.com/ArnavYadnopavit/DigitalSystemLabEE1501/tree/main/Assignmnets/Assignment1`

Thank You