EE1501 Assignment 1

Arjun Pavanje

April 2025

1 4-to-2 Priority Encoder

- **Inputs:** A 4-bit input signal `in[3:0]`.
- **Outputs:**
  - `out[1:0]`: A 2-bit binary output indicating the position of the highest-priority input that is high.
  - `valid`: A 1-bit output signal set to 1 if any input bit is high; otherwise, it is set to 0.

The priority order of the inputs is: `in[3]` > `in[2]` > `in[1]` > `in[0]`.

Approach

The logic is implemented using a `casez` statement which allows for the use of don't-care conditions (denoted by `z` in the input).

- When `in[3]` is high (`4'b1zzz`), the output `out` is set to `3` and `valid` is set to 1.
- When `in[2]` is high (`4'b01zz`), the output `out` is set to `2` and `valid` is set to 1.
- When `in[1]` is high (`4'b001z`), the output `out` is set to `1` and `valid` is set to 1.
- When `in[0]` is high (`4'b0001`), the output `out` is set to `0` and `valid` is set to 1.
- In the default case (when no input is high), the output `out` is set to `0` and `valid` is set to 0.

`casez` saves us from writing out all the cases by letting use don't cares. This way we can intelligently specify `out, valid` for a few select cases and it works for the entire range of values.

Code

```
module top_module (
    input [3:0] in,
    output reg [1:0] out,
    output reg[0:0] valid);
    always @(*) begin
        casez (in)
            4'b1zzz: begin
                out = 2'd3;
                valid = 1;
            end
            4'b01zz: begin
                out = 2'd2;
                valid = 1;
            end
            4'b001z: begin
                out = 2'd1;
                valid = 1;
            end
            4'b0001: begin
                out = 2'd0;
                valid = 1;
            end
            default: begin
                out = 2'd0;
                valid = 0;
            end
        endcase
    end
endmodule
```

2 4-bit Up Counter (Synchronous)

- **Inputs:**
  - `clk`: Clock signal.
  - `reset`: Asynchronous reset signal (active high).
  - `enable`: Control signal to enable counting.
- **Output:**
  - `count[3:0]`: A 4-bit output representing the current count value.

Approach

- The counter is initialized to zero at the beginning using the `initial` block.
- The counting behavior is controlled by the `enable, clock` signal, which allows the counter to increment when high.
- When `reset` is high, the counter is asynchronously reset to zero.

- On each positive edge of the `clk`, `enable` is checked. If it is high, the counter value increments using Karnaugh-Maps logic (T-flipflops used)

State Table

Present State				**T**				**Next State**			
Q_3	Q_2	Q_1	Q_0	T_3	T_2	T_1	T_0	Q'_3	Q'_2	Q'_1	Q'_0
0	0	0	0	0	0	0	1	0	0	0	1
0	0	0	1	0	0	1	1	0	0	1	0
0	0	1	0	0	0	0	1	0	0	1	1
0	0	1	1	0	1	1	1	0	1	0	0
0	1	0	0	0	0	0	1	0	1	0	1
0	1	0	1	0	0	1	1	0	1	1	0
0	1	1	0	0	0	0	1	0	1	1	1
0	1	1	1	1	1	1	1	1	0	0	0
1	0	0	0	0	0	0	1	1	0	0	1
1	0	0	1	0	0	1	1	1	0	1	0
1	0	1	0	0	0	0	1	1	0	1	1
1	0	1	1	0	1	1	1	1	1	0	0
1	1	0	0	0	0	0	1	1	1	0	1
1	1	0	1	0	0	1	1	1	1	1	0
1	1	1	0	0	0	0	1	1	1	1	1
1	1	1	1	1	1	1	1	0	0	0	0

Karnaugh Map for T_3

		Q_1Q_0			
		00	01	11	10
	00	0	0	0	0
	01	0	0	1	0
Q_3Q_2	11	0	0	1	0
	10	0	0	0	0

$$T_3 = Q_2Q_1Q_0$$

Karnaugh Map for T_2

Q_3Q_2 \ Q_1Q_0	00	01	11	10
00	0	0	1	0
01	0	0	1	0
11	0	0	1	0
10	0	0	1	0

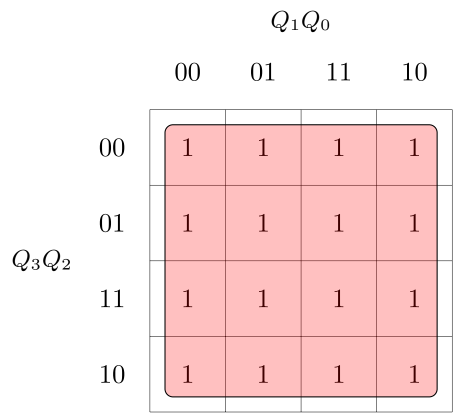
$$T_2 = Q_1Q_0$$

Karnaugh Map for T_1

Q_3Q_2 \ Q_1Q_0	00	01	11	10
00	0	1	1	0
01	0	1	1	0
11	0	1	1	0
10	0	1	1	0

$$T_1 = Q_0$$

Karnaugh Map for T_0



$$T_0 = 1$$

Code

```
module top_module(
  input clk,
  input reset,
  input enable,
  output reg[3:0] count
);
    initial begin
    count = 3'd0;
    end
    always @(posedge clk or posedge reset) begin
        if (reset)
            count = 3'd0;
        else if (enable) begin
            count[0] <= ~count[0];
            count[1] <= count[1] ^ count[0];
            count[2] <= count[2] ^ (count[0] & count[1]);
            count[3] <= count[3] ^ (count[0] & count[1] & count[2])
    ;
        end
    end
endmodule
```

3 Even Parity Generator

- **Input:**
  - `data[7:0]`: An 8-bit input vector representing the data for which parity is to be calculated.
- **Output:**

– `parity`: A single-bit output representing the even parity bit.

Approach

The even parity bit is calculated by first performing a bitwise XOR operation across all the bits of the input `data`. This ensures that parity is 0 when number of 1's is even in the number.

`parity = (^ data)`

generates the even parity bit for the 8-bit input `data`.

Code

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