Report on Verilog Designs: Priority Encoder, Up Counter, and Even Parity Generator

EE24BTECH11024

April 11, 2025

1 Introduction

This report explains three digital design implementations written in Verilog. The designs include:

1. A 4-to-2 Priority Encoder.
2. A 4-bit Up Counter with Enable and Asynchronous Reset.
3. An 8-bit Even Parity Generator.

Each section discusses the functionality, design choices, and code implementation details that fulfill the given specifications. In addition, truth tables and Karnaugh maps are included to illustrate the underlying logic, and a formal proof is provided for the parity generator.

2 4-to-2 Priority Encoder

2.1 Specifications

- **Inputs:** $in[3:0]$ — A 4-bit input signal.
- **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 that is set to 1 if any input bit is high, otherwise 0.
- **Priority Order:** $in[3] > in[2] > in[1] > in[0]$

2.2 Code Explanation

The Verilog module uses continuous assignments with reduction and logical operators to implement the priority encoder.

Listing 1: 4-to-2 Priority Encoder

```
module top_module(input wire [3:0] in, output wire [1:0] out, output valid
    );
    assign valid = (|in[3:0]);
    assign out[0] = (in[3] | ((~in[2]) & in[1]));
    assign out[1] = in[3] | in[2];
endmodule
```

2.3 Truth Table

The following truth table summarizes the operation of the encoder. (Here, "x" denotes a don't-care condition.)

Table 1: Truth Table for the 4-to-2 Priority Encoder

in_3	in_2	in_1	in_0	valid	out_1	out_0
0	0	0	0	0	0	0
0	0	0	1	1	0	0
0	0	1	x	1	0	1
0	1	x	x	1	1	0
1	x	x	x	1	1	1

2.4 Karnaugh Maps

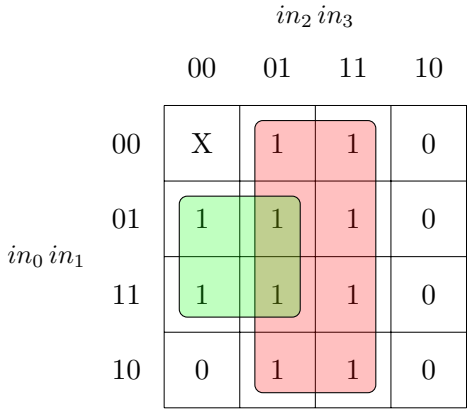
2.4.1 Karnaugh Map for $out[1]$

For $out[1] = in[3] + in[2]$, the output is 1 whenever the upper two bits are not both 0. In our 4-variable Karnaugh map (with columns for $in_3\, in_2$ and rows for $in_1\, in_0$), the minterms for which $out[1] = 1$ correspond to indices 4 through 15.

$in_1\, in_0$ \ $in_3\, in_2$	00	01	11	10
00	X	1	1	1
01	0	1	1	1
11	0	1	1	1
10	0	1	1	1

2.4.2 Karnaugh Map for $out[0]$

For $out[0] = in[3] + ((\ in[2]) \cdot in[1])$, the output is 1 when $in[3] = 1$ (minterms 8 through 15) or when $in[3] = 0$, $in[2] = 0$, and $in[1] = 1$ (minterms 2 and 3).



2.5 Discussion

- The `valid` signal is generated by a reduction OR operator which produces a 1 if any input bit is high.
- The Karnaugh maps illustrate how the functions for $out[0]$ and $out[1]$ are derived from the inputs, obeying the priority: if $in[3] = 1$ the outputs are 11; if only $in[2] = 1$ then 10; and so on.

3 4-bit Up Counter with Enable and Asynchronous Reset

3.1 Specifications

- **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.

3.2 Truth Table (Toggle Conditions and Next State)

The following truth table (written in a style similar to the provided example) shows the present state, the toggle conditions for each flip-flop, and the resulting next state when $enable = 1$.

Present State				T Flip-Flops				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

Table 1: State table for the 4-bit Up Counter with Enable ($enable = 1$).

3.3 Code Explanation

The counter is implemented by cascading T flip-flops. Each T flip-flop toggles when its T input is 1. The toggle inputs are defined (assuming $enable = 1$) as follows:

$$T_0 = \texttt{enable}$$
$$T_1 = \texttt{enable} \cdot Q_0$$
$$T_2 = \texttt{enable} \cdot Q_0 \cdot Q_1$$
$$T_3 = \texttt{enable} \cdot Q_0 \cdot Q_1 \cdot Q_2$$

Thus, the next state is the present state incremented by one modulo 16.

Listing 2: 4-bit Up Counter

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

    wire t0, t1, t2, t3;

    assign t0 = enable;
    assign t1 = enable & count[0];
    assign t2 = enable & count[0] & count[1];
    assign t3 = enable & count[0] & count[1] & count[2];

    Tflipflop ff0(clk, t0, reset, count[0]);
    Tflipflop ff1(clk, t1, reset, count[1]);
    Tflipflop ff2(clk, t2, reset, count[2]);
    Tflipflop ff3(clk, t3, reset, count[3]);

endmodule
```

```
16
17 module Tflipflop(input clk, input T, input reset, output reg Q);
18
19     always @(posedge clk, posedge reset) begin
20         if (reset == 1) begin
21             Q <= 1'b0;
22         end
23         else begin
24             if (T == 1) begin
25                 Q <= ~Q;
26             end
27         end
28     end
29
30 endmodule
```

3.4 Discussion

- Each row shows how the present state and the corresponding toggle conditions determine the next state.
- For a standard binary counter, the next state is simply the present state plus one modulo 16.

4 8-bit Even Parity Generator

4.1 Specifications

- **Input:** $data[7:0]$ (labeled as $in[7:0]$) — An 8-bit input vector.
- **Output:** *parity* — A single-bit output representing the even parity bit.

4.2 Code Explanation

The parity generator uses the Verilog reduction XOR operator to compute the even parity bit.

Listing 3: 8-bit Even Parity Generator

```
1 module top_module(input wire [7:0] in, output out);
2     assign out = ^in;
3 endmodule
```

4.3 Proof that XORing All Bits Generates Even Parity

The key idea behind using the XOR operator ($\oplus$) to generate the even parity bit relies on its properties:

- **Associativity:** $(a \oplus b) \oplus c = a \oplus (b \oplus c)$.
- **Commutativity:** $a \oplus b = b \oplus a$.
- **Identity:** $a \oplus 0 = a$.
- **Self-Inverse:** $a \oplus a = 0$.

When we compute

$$p = b_0 \oplus b_1 \oplus \cdots \oplus b_7,$$

this is equivalent to summing the bits modulo 2:

$$p = \sum_{i=0}^{7} b_i \pmod 2.$$

- If there is an even number of 1's, the pairs of 1's cancel each other ($1 \oplus 1 = 0$) and thus $p = 0$.
- If there is an odd number of 1's, one unpaired 1 remains, yielding $p = 1$.

Proof by Induction:

1. **Base Case:** For a single bit b_0, $p = b_0$. If $b_0 = 0$, the parity is even (0); if $b_0 = 1$, the parity is odd (1).
2. **Inductive Step:** Assume that for n bits,

   $$p_n = b_0 \oplus b_1 \oplus \cdots \oplus b_{n-1},$$

   the result is 0 when there is an even number of 1's and 1 when odd. Adding the $(n+1)^{\text{th}}$ bit b_n,

   $$p_{n+1} = p_n \oplus b_n.$$

   If $b_n = 0$, then $p_{n+1} = p_n$ (parity unchanged); if $b_n = 1$, the parity flips, exactly as expected.

Thus, the reduction XOR operation produces a parity bit p that is 0 if an even number of 1's exist and 1 if odd. When appended to the data, this ensures an even overall count of 1's.

5 Conclusion

This report detailed the design and implementation of three Verilog modules:

1. The **4-to-2 Priority Encoder** uses combinational logic to determine the highest-priority active input. Its functionality is illustrated via a truth table and Karnaugh maps for $out[0]$ and $out[1]$.
2. The **4-bit Up Counter** is implemented using cascaded T flip-flops with an enable and an asynchronous reset, as shown in the detailed state table.
3. The **8-bit Even Parity Generator** employs the reduction XOR operator to compute the parity bit, with an illustrative 4-variable Karnaugh map and a formal induction proof.

Each design meets its specifications with a modular approach, demonstrating key digital design techniques applicable to a wide range of electronic systems. The click **here** for viewing the Codes.