CS39001 - COMPUTER ORGANIZATION LABORATORY

INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Binary Counter: Behavioural Design and Structural Design

Authors:
Ashish Rekhani : 20CS10010
Kartik Pontula : 20CS10031

Date: September 14, 2022

Introduction

In this assignment, we design a 4-bit binary counter with two different approaches in Verilog, and execute the code on the Spartan-3 FPGA board.

Binary Counter

A binary counter is a circuit that generates binary numbers starting from 0 and increments the binary number sequentially.
Example: 4-bit binary counter with initial value 0 generates the following sequence:

$$0000_2, 0001_2, 0010_2, ..., 1111_2, 0000_2, 0001_2, ...$$

Here the counter resets back to 0 after reaching its maximum.

Clock Divider

The clock frequency of the Spartan-3 FPGA board is 4 MHz, and hence the time period between every increment of the counter is 250 ns. This duration is extremely small compared to the persistence period of human vision (50 ms), and as a consequence the incrementing of the counter would be unnoticable (all LEDs would appear to glow simultaneously).
To resolve this, we use a **Clock Divider** module that lowers the clock frequency to 20 Hz or lower, the minimum frequency of persistence of vision.
To implement this, we store the binary value of the clock cycle count and extract the bit at the n'th position from right, where:

$$\begin{aligned} \text{n} &= log_2(\frac{\text{Frequency of FPGA}}{\text{Frequency of vision persistence}}) \\ &= log_2(\frac{4 \text{ MHz}}{20 \text{ Hz}}) \\ &\approx 18 \end{aligned} \tag{1}$$

n shall be used as the new clock cycle count. This means that if n=18, 2^{18} clock cycles constitutes 1 reduced clock cycle.
We have chosen to set n=19 to slightly lower the increment speed.

Having taken care of the vision persistence problem, we proceed to implement the counter in two different ways:

(a) Behavioural Design

In the behavioural design of the counter we write a simple Verilog module that:

- Increments the counter at the +ve edge of every divided clock cycle
- Resets the counter to 0 after it reaches 1111_2 or if the asynchronous reset input is entered

```
module bitCounter(
input clk,
input rstn,
output reg [3:0] count
    );
     wire newclk;
     initial count = 0;

     //always @ (posedge clk) begin
    ClockDivider cd(.clk(clk), .newclk(newclk));
     //nd
      // for asynchronous reset, we add rstn in always condition
      // reset is linked to p108
     always@ (posedge(newclk), posedge(rstn))
        begin
         if (rstn) count <= 4'b0000;
         else if (count == 4'b1111) count <= 4'b0000;
         else count <= count + 1;
        end
endmodule
```

Figure 1: Behavioural up-counter implemented in bitCounter.v

(b) Structural Design

In this method we employ the 4-bit adder module built in earlier assignments, along with a D Flip-Flop module.

AddByOne Module

Here we utilize the **Carry Lookahead Adder** module but specialized for increments. We set A = counter, B = 0, and $c_{in} = 0001_2$. The equations for this specialized adder are much simpler because the Generate signals are all zero.

$$G_i = A_i \cdot B_i = A_i \cdot 0 = 0$$

Hence the output formulae are:

$$C_0 = 1$$
$$C_1 = P_0$$
$$C_2 = P_1 \cdot P_0 \cdot C_0$$
$$C_3 = P_2 \cdot P_1 \cdot P_0 \cdot C_0$$

```
module AddbyOne(
    input [3:0] count, output [3:0] out
    );

    wire [3:0] P;
    wire [3:0] C;
    // we use the LCA in previous assigment but with fixed value of 2nd operand as 4'b0000
    // we input the value of c_in as 1
    assign P = count;
    assign C[0] = 1'b1;
    // G in LCA is assigned value A & B but since B == 0, G = 0
    // so the updated values of C are as follows
    assign C[1] = P[0];
    assign C[2] = P[1] & P[0] & C[0];
    assign C[3] = P[2] & P[1] & P[0] & C[0];
    // assign c_out = P[3] & P[2] & P[1] & P[0] & C[0];

    assign out = P^C;
endmodule
```

Figure 2: Implementation of AddByOne.v

D Flip-Flop Module

A D Flip-Flop (DFF) is a circuit that can store the value of its previous state. At every clock cycle, the output Q is assigned the same value as D, and Q' assigned the complement of D. This assignment happens only at the end of each clock cycle. There is also an asynchronous reset input that resets Q to 0 and Q' to 1.

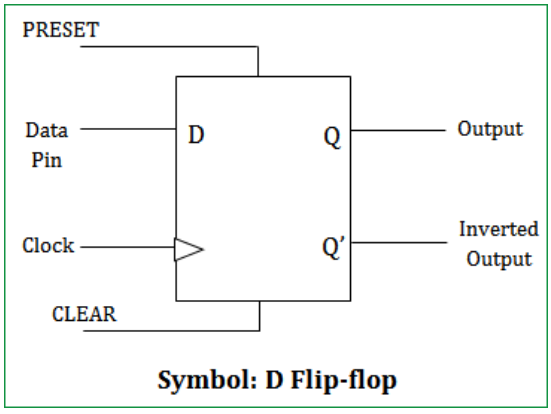

Figure 3: Diagram of a DFF.

Clock	D	Q	Q’
-ve edge	0	Q	Q’
-ve edge	1	Q	Q’
+ve edge	0	0	1
+ve edge	1	1	0

Table 1: Truth Table of DFF

```
module DFF(d,clk,rstn,q,qbar);
    input d, clk, rstn;
    output reg q, qbar;
    always @(posedge clk)
    begin
        if(rstn== 1)begin
            q <= 0;
            qbar <= 1;
            end
        else begin
            q <= d;
            qbar <= ~d;
end
    end
endmodule
```

Figure 4: Implementation of DFF.v

Overall Implementation

Finally to create the binary counter structurally, we utilize 4 DFFs to keep track of the old counter value, then apply the AddByOne module to increment. We also make sure the ClockDivider module is used for visibility of increments.

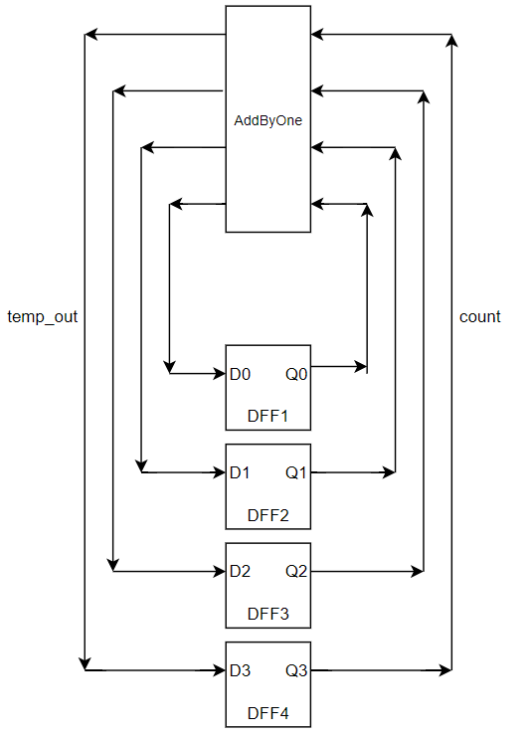

Figure 5: Basic outline of structural design

```
module CounterCLA(
input clk,
input rstn,
output [3:0] count
    );

    wire newclk;
    wire [3:0] Q, Qbar, temp_out;

    //always @ (posedge clk) begin
    ClockDivider cd(.clk(clk), .newclk(newclk));
    // for asynchronous reset, we add rstn in always condition
    //end
    DFF dff1(.d(temp_out[0]), .clk(newclk), .rstn(rstn), .q(Q[0]), .qbar(Qbar[0]));
    DFF dff2(.d(temp_out[1]), .clk(newclk), .rstn(rstn), .q(Q[1]), .qbar(Qbar[1]));
    DFF dff3(.d(temp_out[2]), .clk(newclk), .rstn(rstn), .q(Q[2]), .qbar(Qbar[2]));
    DFF dff4(.d(temp_out[3]), .clk(newclk), .rstn(rstn), .q(Q[3]), .qbar(Qbar[3]));

    assign count = Q;
    AddbyOne ad1(count,temp_out);

endmodule
```

Figure 6: Implementation of counterCLA.v