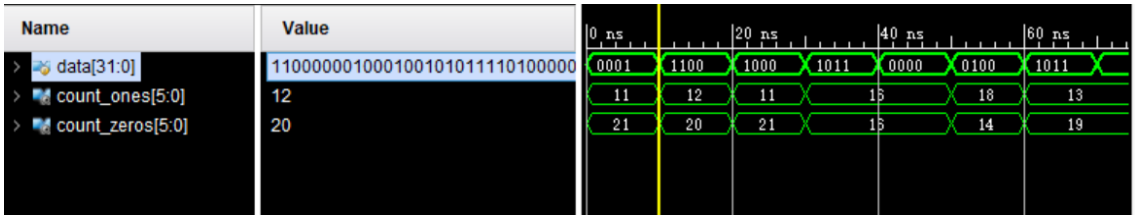
1.请自行设计一个模块，完成统计 32 位 2 进制数 0 和 1 出现的次数。

```
module count01(
    input [31:0] data,
    output reg [5:0] count_ones,
    output reg [5:0] count_zeros
);

integer i;

always @(*) begin
    count_ones = 0;
    count_zeros = 0;
    for(i = 0; i < 32; i = i + 1) begin
        count_ones = count_ones + data[i]; // 直接加data[i]，因为
        count_zeros = count_zeros + ~data[i]; // 使用非操作符来统
    end
end

endmodule
```

```
module tb_count01;
    reg [31:0] data;
    wire [5:0] count_ones;
    wire [5:0] count_zeros;
count_bits mycount (data,count_ones,count_zeros);

    initial begin
    repeat(10)begin
      data=$random % 33'b1_0000_0000_0000_0000_0000_0000_0000_0000;
      #10;
    end
    end
endmodule
```



输入数据 (data): data 的值为 11000000100010010101111010000。

手动计算 1 和 0 的数量:

- 1 的数量: 11000000100010010101111010000 中有 12 个 1。
- 0 的数量: 11000000100010010101111010000 中有 20 个 0。

波形图中的值:

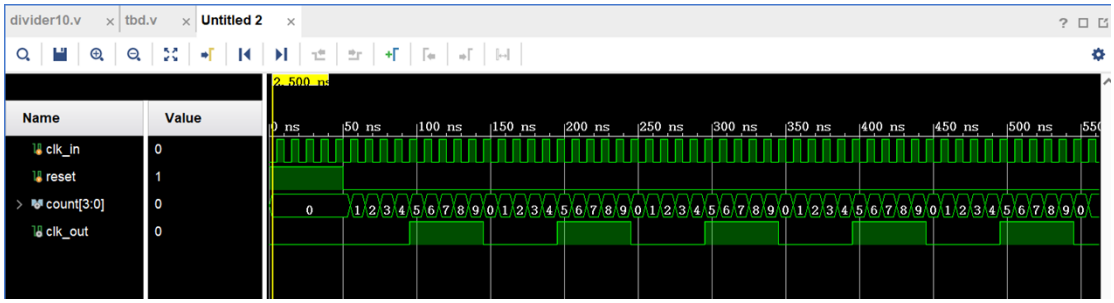
- count_ones 的值为 12。
- count_zeros 的值为 20。

从波形图中可以看出，count_ones 和 count_zeros 的值与手动计算的结果一致，这表明模块正确地统计了 32 位二进制数中 1 和 0 的数量。

2.请使用 always 块语句，实现一个十分频器，divider10(input clk_in,input reset,output count,output clk_out)。其功能可以理解为将时钟降频为原来的 10 分之一（思路：对 clk_in 进行 count 计数，count 取值 0~9,count 数到 5 时，clk_out 由 1 变 0,count 数到 10 时自动归零同时 clk out 由 0 变 1)。

```v
module divider10(
    input clk_in,
    input reset,
    output reg [3:0] count,
    output reg clk_out
);

// 内部逻辑
always @(posedge clk_in or posedge reset) begin
    if (reset) begin
        count <= 4'b0;
        clk_out <= 1'b0;
    end else begin
        count <= count + 1'b1;
        if (count == 4 || count == 9) begin
            clk_out <= ~clk_out;
        end
        if (count == 9) begin
            count <= 4'b0;
        end
    end
end

endmodule
```

```v
module tbd_tb();
reg clk_in, reset;
wire [3:0] count;
wire clk_out;

divider10 mydiv (
    .clk_in(clk_in),
    .reset(reset),
    .count(count),
    .clk_out(clk_out)
);

always #5 clk_in = ~clk_in;

initial begin
    clk_in <= 0;
    reset <= 1;

    #50 reset <= 0;

    #500 $finish; // 继续仿真500时间单位后结束
end

initial begin
    $monitor("Time: %t | clk_in: %b | reset: %b | count: %b | clk_out: %b",
             $time, clk_in, reset, count, clk_out);
end

endmodule
```



仿真结果显示，当 reset 信号为高电平时，计数器 count 和输出信号 clk_out 均保持为 0 不变，表明模块正确响应复位。复位解除后，随着输入时钟 clk_in 的上升沿，计数器递增至 9 时重置，并在计数值为 4 和 9 时翻转 clk_out，实现 clk_out 周期为 clk_in 十倍的效果，即成功将输入时钟频率降低了十分之一，验证了该十分频电路设计的有效性和准确性。

总结：

通过这两次实验，我不仅掌握了如何根据需求选择合适的逻辑结构（组合逻辑与同步时序逻辑），还学会了构建有效的测试平台以自动化验证设计的正确性和可靠性。同时，也提升了我对波形图及仿真结果的解读能力，能够从中提取有价值的信息指导设计优化，并积累了应对和解决设计过程中遇到问题的经验。这些技能对于深入理解数字系统设计的基础知识至关重要，同时也为未来处理更复杂的项目打下了坚实的基础。