实验 4　FIFO 实验

班级：01　　学号：2021k8009925006　　姓名：冯浩瀚

实验目的:

1. 熟悉 Verilog 编程、调试
2. 熟悉 FIFO 工作原理
3. 实现功能较复杂的数字电路

实验环境:

1、操作系统：Windows 11

2、软件及版本：Vivado 2017.4

原理说明

FIFO：

即 First In First Out，先进先出，先进入队列的数据先输出。

如图，这是一个同步的 FIFO，其中：
1. data_in 为数据输入端口，位宽为 8 位。
2. data_out 为数据输出端口，位宽为 4 位。
3. 同步 FIFO 的缓冲区大小为 64bit。FIFO 的一项数据宽度是 4bits，每次写入 8bit，每次读出 4bit。
4. 每次读出的数据，先读出写入时 8bit 的低 4 位，后读出高 4 位。即写入 8bit 的低四位是第一个数据，高四位是第二个数据。

5. input_valid 和 input_enable 控制着数据的传入。
6. output_valid 和 output_enable 控制着数据的输出。
7. 对于同一组 valid 和 enable 信号而言，只有 2 者同时有效时才会工作，即 input_valid 和 input_enable 同时有效时才读入 data_in 的数据。同理如 output。

请设计一个同步 FIFO 模块，要求如下：

1. 同步的 FIFO 只存在一种工作状态，即要么只接收数据不输出数据，要么只输出数据，不接收数据。
2. FIFO 只有在填满数据之后才向外输出数据，只有在读空之后才允许写入数据。
3. 外部的控制信号 input_valid 和 output_enable 是随机控制的

要满足以上要求，需要：

1. mem 存满之后 input_enable 变为 0，output_valid 变为 1。mem 读空之后 output_valid 变为 0，input_enable 变为 1
2. 因为每次写入 8 位，读出 4 位，所以要引入变量 flag 作为指针记录读出数据次数，如 flag 为奇数时读数据的低四位，flag 为偶数时读数据的高四位。

接口定义：

Input 信号			Output 信号		
名称	说明	位宽	名称	说明	位宽
clk	时钟信号	1	data_out	数据输出端口	4
rstn	同步置位信号	1			
data_in	数据输入端口	8			
input_valid	输入控制信号	1			
output_enable	输出控制信号	1			

调试过程及结果波形：

调试过程

1. 阻塞赋值与非阻塞赋值的误用

非阻塞(Non_Blocking)赋值方式(如 b <= a;)在块结束后才完成赋值操作。b 的值并不是立刻就改变的；阻塞(Blocking)赋值方式(如 b = a;)赋值语句执行完后,块才结束。b 的值在赋值语句执行完后立刻就改变的。

2.

```
if (write_addr == 3'b111) begin
    output_valid = 1'b1;
    input_enable = 1'b0;
    write_addr <= 3'b000;
end
```

```
if(read_addr == 3'b111) begin
    input_enable = 1'b1;
    output_valid = 1'b0;
    read_addr <= 3'b000;
    flag <= 4'b0000;
end
```

在 mem 写满（读空）之后没有将 write_addr(read_addr)归零，导致读写只能进行一轮，无法进行重复读写。

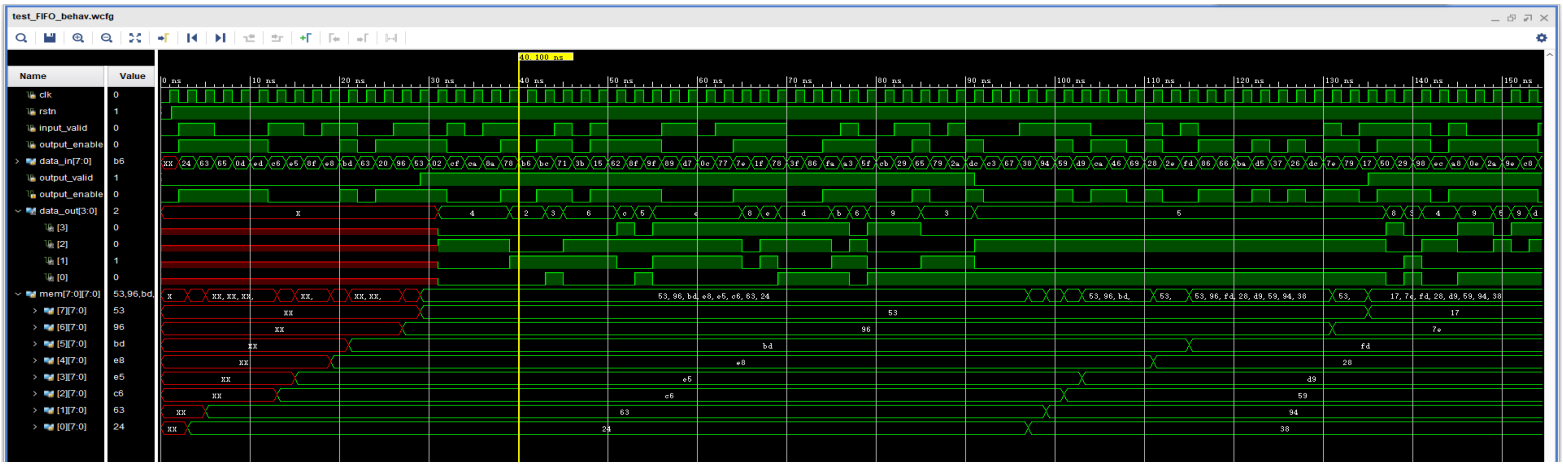
3. write_addr 归零的位置错误，在写满之后归零，导致后续读出的时候，写入仍在进行，违反题目要求的读写不同时进行。

   调试方法：在读空之后再将 write_addr 和 read_addr 都归零。

4. 以 read_addr 是否达到 3'b111 为判断是否读空的标准，导致 mem[7][7:4]（即最后一次写入的数据的高四位）未读出就停止读取。

   调试方法：以 flag 是否达到 4'b1111 为判断是否读空的标准

结果波形：



实验总结:

本实验通过编写 FIFO 进一步提高了对 Verilog 的掌握程度，能用 Verilog 实现功能较复杂的数字电路。并学习了从波形图中获取信息，推断 bug 产生的原因。例如本次实验先后遇到了读写一轮后就停止工作、读写同时进行、剩 4 位数据无法读取的波形，通过观察就能轻松定位 bug，比审查源代码效率提高了很多。

源代码:

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
```

```
// Create Date: 2022/11/29 10:54:31
// Design Name:
// Module Name: FIFO
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
module FIFO(
    input clk,
    input rstn,
    input input_valid,
    input output_enable,
    input [7:0] data_in,
    output reg [3:0] data_out
    );
    reg [7:0] mem [7:0];
    reg [2:0] write_addr;
    reg [2:0] read_addr;
    reg [3:0] flag;
    reg input_enable, output_valid;

    always @(posedge clk or negedge rstn) begin
        if (rstn == 0) begin
            write_addr <= 3'b000;
            flag <= 4'b0000;
            input_enable <= 1'b1;
            output_valid <= 1'b0;
        end
        else begin
            if (input_valid & input_enable) begin
                write_addr <= write_addr + 3'b001;
                mem[write_addr] [7:0] <= data_in[7:0];//使用非阻塞赋值
                if (write_addr == 3'b111) begin
                    output_valid = 1'b1;
                    input_enable = 1'b0;
                end
```

```
                else begin
                    output_valid = 1'b0;
                    input_enable = 1'b1;
                end
            end
            else if (output_valid & output_enable) begin
                flag <= flag + 4'b0001;
                if (!(flag % 2)) begin
                    read_addr = flag / 2;
                    data_out[3:0] = mem[read_addr] [3:0];
                end
                else if (flag % 2) begin
                    read_addr = (flag-1) / 2;
                    data_out[3:0] = mem[read_addr] [7:4];
                end
                //flag 每加 2read_addr 加 1
                if(flag == 4'b1111) begin
                    input_enable = 1'b1;
                    output_valid = 1'b0;
                    read_addr <= 3'b000;
                    flag <= 4'b0000;
                    write_addr <= 3'b000;
                end
                else begin
                    input_enable = 1'b0;
                    output_valid = 1'b1;
                end
            end
        end
    end
endmodule
```

testbench:

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 2022/11/29 17:03:05
// Design Name:
// Module Name: test_FIFO
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
```

```
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module test_FIFO(

    );
    reg clk, rstn, input_valid, output_enable;
    reg [7:0] data_in;
    wire [3:0] data_out;
    FIFO instance_FIFO(
        .clk(clk),
        .rstn(rstn),
        .input_valid(input_valid),
        .output_enable(output_enable),
        .data_in(data_in),
        .data_out(data_out)
    );

    initial begin
        clk = 0;
        rstn = 1;
        input_valid = 0;
        output_enable = 0;
        #0.1 rstn = 0;
        #1.1 rstn = 1;
    end

    always begin
        #1;
        clk = ~clk;
    end

    always begin
        #2;
        data_in=$random()%9'b100000000;
        input_valid=$random()%2;
        output_enable=$random()%2;
```

```
    end

    always begin
    #400;
    $finish;
    end
endmodule
```