实验 3 状态机实验

1 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

2 实验环境

AMD Vivado2022.2

3 原理说明

FIFO 也即 first in first out, 先进先出, 在本实验中主要采取了一种新的, memory 型的变量, 由此我们可以利用地址来进行索引, 进而进行写入和读出操作, 也利用程序内部的 input 和 output 的 enable 和 valid 信号来进行写入和读出的控制, 在接下来的接口定义中有进一步说明

4 接口定义

在本次实验的模块中, 接口都是完全相同的, 因此合成一个来说明, 接口如下

4.1 序列检测器

```
input clk,
// 时钟信号
input rstn,
// 异步复位信号，但是在实际设计中是同步复位，但不影响结果

input [7:0] data_in,
//8位输入信号(这个输入在实验三中变成16位)
input input_valid,
//外部对输入的控制信号

output reg [15:0] data_out,
```

```
//16位输出信号(这个输出在实验三中变成8位)
input output_enable
//外部对输出的控制信号
```

另外, 在本次实验中还有比较重要的设计文件内部的变量, 具体如下

```
reg [15:0]mem [31:0]
//表示存储，在本次实验的三个子实验中，都是深度位16位的32个数据

reg write_state
//这个信号用于控制写入低8位还是高8位,每次写入之后取反(在实验1,2中)
reg read_state
//这个信号用于控制读出低8位还是高8位,每次读出之后取反(在实验3中)

reg write_addr
//这个信号用于控制写入的地址，每次写入之后加1
reg read_addr
//这个信号用于控制读出的地址，每次读出之后加1

reg input_enable
//设计文件内部的写入控制信号，在本实验中用于检测是否写满
reg output_valid
//设计文件内部的读出控制信号，在本实验中用于检测是否读空
```

5 调试过程以及结果

本次实验整体而言较为顺利, 但是在实验一中发现设计文件并不能很好地实现读出的过程, 经过调试发现是因为所写的逻辑功能有问题, 一开始在 if-else 语句中产生了不正确地, 矛盾的语句, 因此只能读入不能写出, 后来修改了逻辑语句的位置, 就可以很好地实现功能, 接下来给出三个子实验的波形图和对它们的说明, 此外, 本次实验也需要注意调节 testbench 中随机数变化的频率.

值得是注意的是, 在实验二和实验三种需要保证写完了之后可以从头继续重新读, 因此在这里采用了一种增加 write_period 和 read_period, 用于记写满和读空次数的变量, 通过比较这两个变量就可以实现多重的写入与读出

5.1 FIFO: 写满才能读出

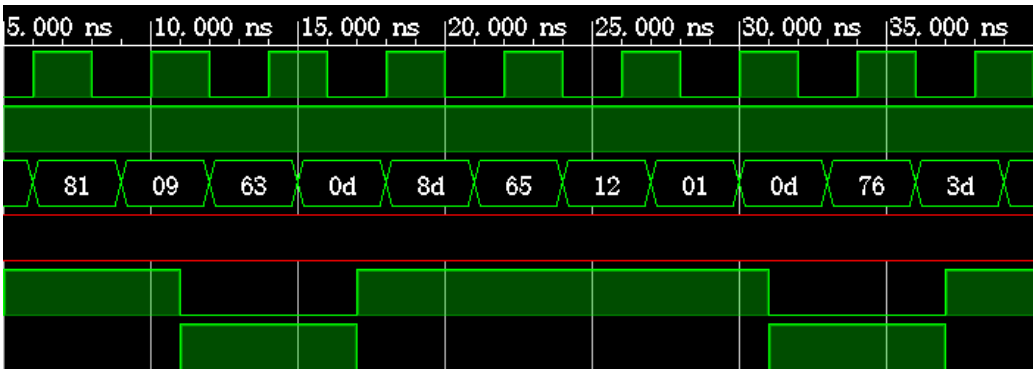

图 1: FIFO1: 写入的初始阶段

在这个波形图中, 一开始读入的是 81 信号, 后来是 09 信号, 在 write_addr=1 中写入了 0981 在这个波形图中输出有效, 而读出的第一个数据就是第一个写入的数据 0981, 可以初步

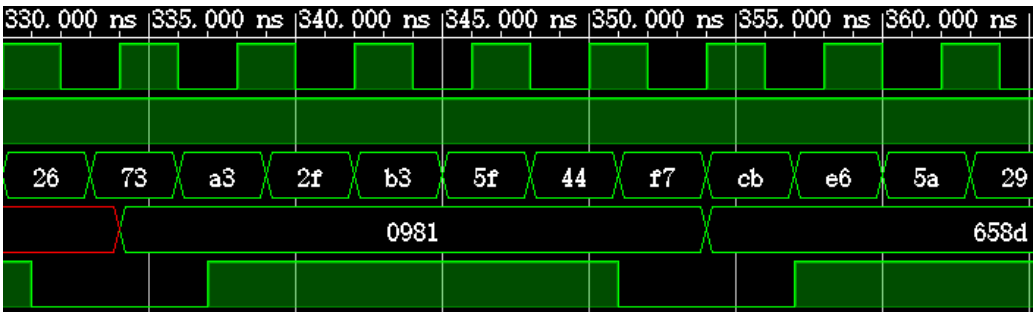

图 2: FIFO1: 读出的初始阶段

判断所设计的 fifo 是满足我们实验要求的

5.2 FIFO: 读写不影响

在这个实验中需要注意的是不能读出空的 fifo, 所以在读出的时候需要保证 read_addr 要小于 write_addr, 波形图如下 可以看到, 一开始写入 24, 之后是 81, 之后再第一个读出有效的

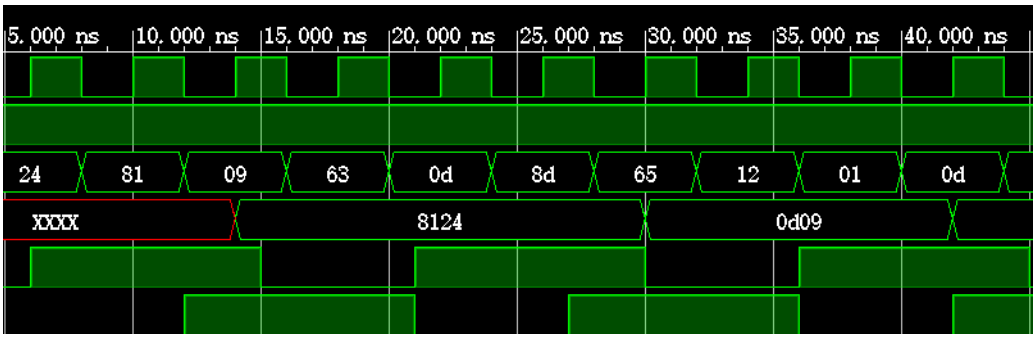

图 3: FIFO2

时候读出 8124, 满足实验要求

5.3 FIFO: 写入读出频率比

在这个实验中转化为每次写入 16 位, 之后输出 8 位, 调节 testbench 使得写入和读出的频率为 2:3, 结果如下

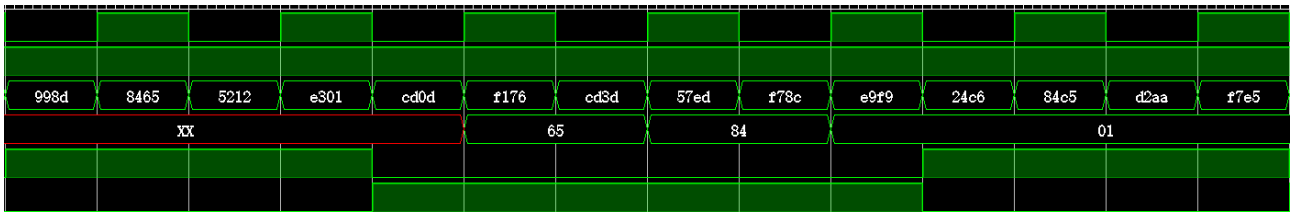

图 4: FIFO3

一开始写入 8465, 之后读出 65, 在读出 84, 之后进入写入的阶段, 输出一直是 01 保持不变

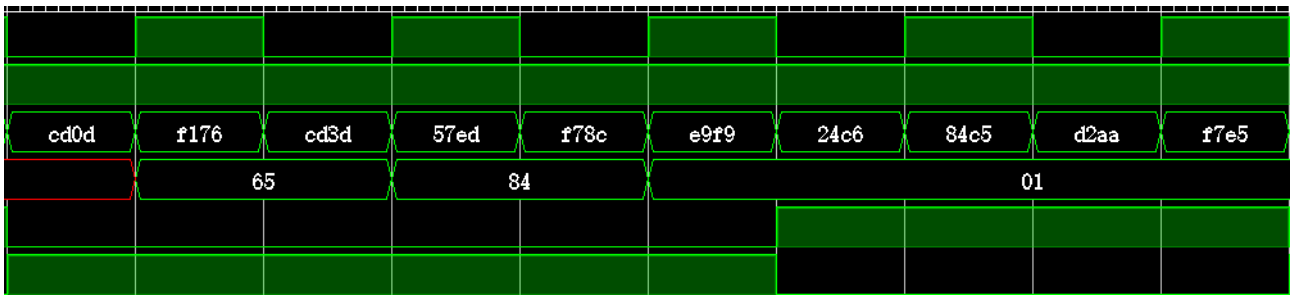

图 5: FIFO3: 分频展示

这个波形图展示了我们的写入和读出的频率为 2:3, 说明我们的设计基本符合条件

6 实验总结

在本次实验中, 学习了 fifo 的原理以及设计方法, 另外值得一提的是, 我们对 memory 的使用也对未来有很大的帮助, 在一定程度上掌握了有关地址索引的只是. 另外, 在本次实验中对 testbench 的调试占较大一部分, 对 testbench 的设计与调试有了更深的理解

7 源代码

7.1 设计文件

7.1.1 FIFO1: 写满才能读出

```
module fifo(
    input clk,
    input rstn,

    input [7:0] data_in,
    input input_valid,

    output reg [15:0] data_out,
    input output_enable
    //input_valid和output_enable是外部的控制信号
    );


    reg [15:0] mem [31:0];
    //表示每组数据的长度为16,一共有32组数据
    reg [6:0] write_addr;
    //表示写入时候的地址, 因为一共有32组,
    //所以在这里需要用5位来表示数据地址
    reg [6:0] read_addr;

    reg write;//用于表示当前fifo的状态是写入还是读出
    reg read;
    reg input_enable;
    reg output_valid;

    reg write_state;
```

```
27      //因为是8位8位写入，
28      //所以用这样一个数据来表示写入在低8位还是高8位
29
30
31      always @(negedge rstn or posedge clk)begin
32          if(rstn==0)begin
33              write_addr <= 6'b000000;
34              //初始化读入和读出，这个保证了先入先出
35              read_addr <= 6'b000000;
36              input_enable <= 1;
37              //表示一开始是从读入开始的
38              output_valid <= 0;
39              //表示一开始不读出， 只写入
40              write_state <= 0;
41          end
42          else begin
43          if(write == 1)begin//读到是写入状态
44                  if(write_state == 0)begin
45                      write_state <= ~write_state;
46                      mem[write_addr][7:0] <= data_in[7:0];
47                  end
48                  else begin
49                      write_state <= ~write_state;
50                      mem[write_addr][15:8] <= data_in[7:0];
51                      if(write_addr < 31)begin
52                      //如果是前32个广度， 继续读入
53                          write_addr <= write_addr+1;
54                      end
55                      else begin
56                      //发现地址超过31,说明写满了，转化为读出
57                          input_enable <= 0;
58                          output_valid <= 1;
59                          write_addr <= 0;
60                      end
61                  end
62          end
63          else if(read == 1)begin
```

```
            data_out[15:0] <= mem[read_addr][15:0];
            if(read_addr < 31)begin
                read_addr <= read_addr+1;
            end
            else begin
                read_addr <= 0;
                input_enable <= 1;
                output_valid <= 0;
            end
        end
        else begin
        end
        end
    end

        always @(*) begin
        write = input_valid && input_enable;
        end

        always @(*) begin
        read = output_valid && output_enable;
        end

endmodule
```

7.1.2 FIFO2: 读写不影响

```
module fifo2(
    input clk,
    input rstn,

    input [7:0] data_in,
    input input_valid,

    output reg [15:0] data_out,
    input output_enable
    //input_valid和output_enable是外部的控制信号
```

```
    );


    reg [15:0] mem [31:0];
    //表示每组数据的长度为16,一共有32组数据
    reg [6:0] write_addr;
    //表示写入时候的地址， 因为一共有32组，
    //所以在这里需要用5位来表示数据地址
    reg [6:0] read_addr;

    reg write;//用于表示当前fifo的状态是写入还是读出
    reg read;
    reg input_enable;
    reg output_valid;

    reg write_state;
    //因为是8位8位写入，
    //所以用这样一个数据来表示写入在低8位还是高8位
    reg [3:0]write_period;
    reg [3:0]read_period;


    always @(negedge rstn or posedge clk)begin
        if(rstn==0)begin
            write_addr <= 6'b000000;
            //初始化读入和读出，这个保证了先入先出
            read_addr <= 6'b000000;
            input_enable <= 1;
            //表示一开始都可以读入， 读出
            output_valid <= 1;
            write_state <= 0;
            write_period <= 0;
            read_period <=0;
        end
        else begin
        if(write == 1)begin//读到是写入状态
                if(write_state == 0)begin
```

```
48                      write_state <= ~write_state;
49                      mem[write_addr][7:0] <= data_in[7:0];
50                  end
51                  else  begin
52                      write_state <= ~write_state;
53                      mem[write_addr][15:8] <= data_in[7:0];
54                      if(write_addr < 31)begin//如果是前32个广度，继续读入
55                          write_addr <= write_addr+1;
56                      end
57                      else   begin
58                      write_addr <= 0;    //后面的写满，从头再写
59                      write_period = write_period +1;//表示写完了一圈
60                      end
61              end
62          end
63          if(read == 1)begin
64          //write和read并不是if-else的两端，
65          //在这种语法下二者应该可以同时进行
66              if((read_addr < write_addr) || (read_period < write_period)) be
67              //用于判断接下来要读出的地址是不是没有写入数据
68                  data_out[15:0] = mem[read_addr][15:0];
69                  if(read_addr < 31) begin //检测是不是读空
70                      read_addr <= read_addr+1;
71                  end
72                  else begin
73                      read_addr <= 0;//表示读完了一个周期
74                      read_period = read_period+1;//读的周期加1
75                  end
76              end
77              else begin//如果要读出的是空的，就不读出
78              end
79          end
80          end
81      end
82
83
84          always @(*) begin
```

```
        write = input_valid && input_enable;
        end

        always @(*) begin
        read = output_valid && output_enable;
        end

endmodule
```

7.1.3 FIFO3: 写入读出频率比

```
module fifo3(
    input clk,
    input rstn,

    input [15:0] data_in,
    input input_valid,

    output reg [7:0] data_out,
    input output_enable
    //input_valid和output_enable是外部的控制信号
    );


    reg [15:0] mem [31:0];
    //表示每组数据的长度为16,一共有32组数据
    reg [5:0] write_addr;
    //表示写入时候的地址，因为一共有32组，
    //所以在这里需要用5位来表示数据地址
    reg [5:0] read_addr;

    reg write;//用于表示当前fifo的状态是写入还是读出
    reg read;
    reg input_enable;
    reg output_valid;
    reg [3:0]write_period;
    reg [3:0]read_period;
```

```
27
28      reg read_state;//因为是8位8位写入，
29      //所以用这样一个数据来表示写入在低8位还是高8位
30
31
32      always @(negedge rstn or posedge clk)begin
33          if(rstn==0)begin
34              write_addr <= 5'b000000;
35              //初始化读入和读出，这个保证了先入先出
36              read_addr <= 5'b000000;
37              input_enable <= 1;
38              //表示一开始都可以读入， 读出
39              output_valid <= 1;
40              read_state <= 0;
41              read_period <= 0;
42              write_period <= 0;
43          end
44          else begin
45          if(write == 1)begin//读到是写入状态
46              mem[write_addr][15:0] <= data_in[15:0];
47              //每次写入16位数据
48                      if(write_addr < 31)begin
49                      //如果是前32个广度， 继续读入
50                          write_addr <= write_addr+1;
51                      end
52                      else   begin
53                      write_addr <= 0;
54                      write_period = write_period + 1;
55                      //如果已经写满了，就不能继续写了
56                      end
57              end
58          end
59          if(read == 1)begin
60          //write和read并不是if-else的两端，
61          //在这种语法下二者应该可以同时进行
62              if(read_addr < write_addr || read_period < write_period) begin
63              //用于判断接下来要读出的地址是不是没有写入数据
```

```
                if(read_state == 0) begin
                    read_state <= ~read_state;
                    data_out[7:0] <= mem[read_addr][7:0];
                end
                else begin
                    read_state <= ~read_state;
                    data_out[7:0] <= mem[read_addr][15:8];
                    if(read_addr < 31)begin
                        read_addr <= read_addr+1;
                    end
                    else begin
                    read_addr <= 0;
                    read_period = read_period+1;
                    end
                end
            end
            else begin//如果要读出的是空的，就不读出
            end
        end
        end

        always @(*) begin
        write = input_valid && input_enable;
        end

        always @(*) begin
        read = output_valid && output_enable;
        end

endmodule
```

7.2 激励文件

7.2.1 FIFO1: 写满才能读出

```
module test_sim_fifo(

    );
```

```
4
5       reg clk,rstn;
6       reg [7:0] data;
7       wire [15:0] out;
8       reg input_valid, output_enable;
9
10      fifo test_sim_fifo(
11          .clk(clk),
12          .rstn(rstn),
13          .input_valid(input_valid),
14          .output_enable(output_enable),
15          .data_in(data),
16          .data_out(out)
17       );
18
19       always #2 begin
20          clk = ~clk;
21          end
22
23       initial begin
24          clk = 1'b0;
25          rstn = 1'b1;
26          input_valid = 1'b0;
27          output_enable = 1'b0;
28          #1 rstn = 1'b0;
29          #2 rstn = 1'b1;
30       end
31
32       always begin
33          #3;
34          data = $random()%9'b1_0000_0000;     //输入是8位
35       end
36
37       always begin
38          #5;
39          input_valid = 1'b1;
40          #6;
```

```
        input_valid = 1'b0;
        output_enable = 1'b1;
        #6;
        input_valid = 1'b1;
        output_enable = 1'b0;
        #3;

    end
endmodule
```

7.2.2 FIFO2: 读写不影响

```
module test_fifo2(

);

    reg clk,rstn;
    reg [7:0] data;
    wire [15:0] out;
    reg input_valid, output_enable;

    fifo2 test_sim_fifo(
        .clk(clk),
        .rstn(rstn),
        .input_valid(input_valid),
        .output_enable(output_enable),
        .data_in(data),
        .data_out(out)
     );

     always #2 begin
        clk = ~clk;
        end

     initial begin
        clk = 1'b0;
        rstn = 1'b1;
```

```
        input_valid = 1'b0;
        output_enable = 1'b0;
        #1 rstn = 1'b0;
        #2 rstn = 1'b1;
    end

    always begin
        #4;
        data = $random()%9'b1_0000_0000;     //输入是8位
    end

    always begin
        #6;
        input_valid = 1'b1;
        output_enable = 1'b0;
        #6;
        input_valid = 1'b1;
        output_enable = 1'b1;
        #3;
        input_valid = 1'b0;
        output_enable = 1'b1;
        //$finish;
    end
endmodule
```

7.2.3 FIFO3: 写入读出频率比

```
module test_fifo3(

    );

    reg clk,rstn;
    reg [15:0] data;
    wire [7:0] out;
    reg input_valid, output_enable;

    fifo3 test_sim_fifo(
```

```
11          .clk(clk),
12          .rstn(rstn),
13          .input_valid(input_valid),
14          .output_enable(output_enable),
15          .data_in(data),
16          .data_out(out)
17      );
18
19      always #2 begin
20          clk = ~clk;
21          end
22
23      initial begin
24          clk = 1'b0;
25          rstn = 1'b1;
26          input_valid = 1'b0;
27          output_enable = 1'b0;
28          #1 rstn = 1'b0;
29          #2 rstn = 1'b1;
30      end
31
32      always begin
33          #2;
34          data = $random()%17'b1_0000_0000_0000_0000;     //输入是8位
35      end
36
37      always begin
38          #12;
39          input_valid = 1'b1;
40          output_enable = 1'b0;
41          #8;
42          input_valid = 1'b0;
43          output_enable = 1'b1;
44      end
45  endmodule
```