实验 4　FIFO 实验

Chen-Yuanmeng*

2024/12/5

1　实验目的

1. 熟悉 Verilog 编程、调试
2. 熟悉 FIFO 的工作原理, 增强对非阻塞赋值的理解
3. 学习了解异步时钟信号的处理

2　实验环境

- Microsoft Windows 10.0.19045.5131
- Vivado v2017.4 (64-bit)
- 玉泉路一机房

3　原理说明

先入先出队列 (FIFO), 即 First In First Out, 是先进先出的数据存储队列.

对于一个 FIFO, 其中的数据存储一般用到 `memory` 型, 它用于存储多个相同的位宽的信号. 如下面的代码定义了一个 `memory` 型变量 `addr`, 该变量有 4 组 (序号 0 3), 每组的长度为 5 bits (0 4).

```
reg [4:0] addr [3:0];
```

`input_valid` 和 `input_enable` 控制着数据的传入, `output_valid` 和 `output_enable` 控制着数据的输出. 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` 同时为 1 时才写入 `data_in` 的数据, output 同理.

数据的写入和输出依据时钟信号触发, 因此用到时序逻辑电路和非阻塞赋值 `<=`.

4　接口定义

对于一个同步的 FIFO, 通常有以下接口:

```
module fifo(
    input clk,              // 时钟信号
    input rstn,             // 重置信号
    input [7:0] data_in,    // 输入 (8 bit宽, 可更改)
    input input_valid,      // 表示 data_in 是否有效的标识
    input output_enable,    // 表示 data_out 是否应输出下一个值的标识
    output reg [15:0] data_out, // 输出 (16 bit宽, 可更改)
```

*Email: chenyumeng23@mails.ucas.ac.cn

```
    output reg input_enable, // 表示是否能接受 data_in 的标识
    output reg output_valid // 表示目前输出是否有效的标识
);
```

若为异步 (即两个不同的时钟信号分别控制读和写), 则会有如下改变:

```
module fifo_async(
    input clk_read, // 读操作 时钟信号
    input clk_write, // 写操作 时钟信号
    // 其余一致
)
```

5 调试过程及结果

点击左侧 “SIMULATION” 下的 “Run Simulation”, 以默认设置进行模拟调试. 调试结果分别如图所示:

5.1 同步 FIFO 模块: 先写入后读出

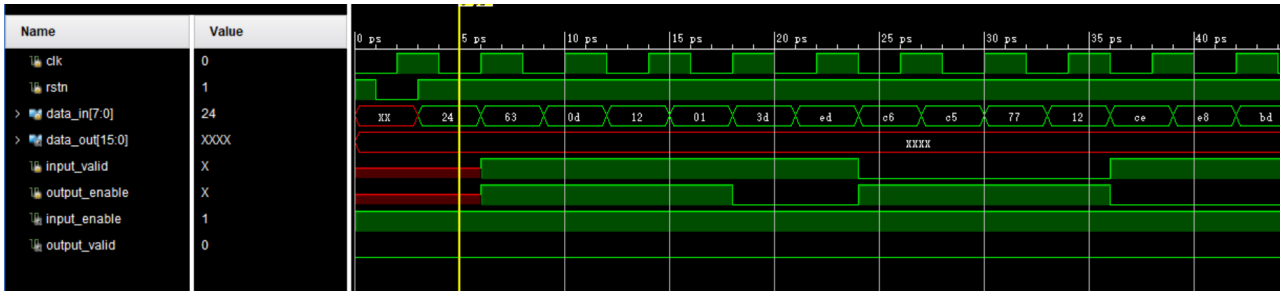

图 1: 先写入后读出的同步 FIFO 模块模拟示意图: 读部分

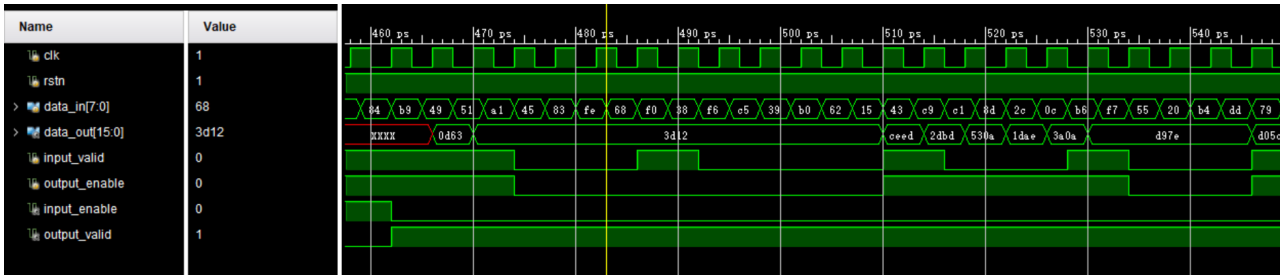

图 2: 先写入后读出的同步 FIFO 模块模拟示意图: 写部分

5.2 同步 FIFO 模块: 写入和读出同时进行

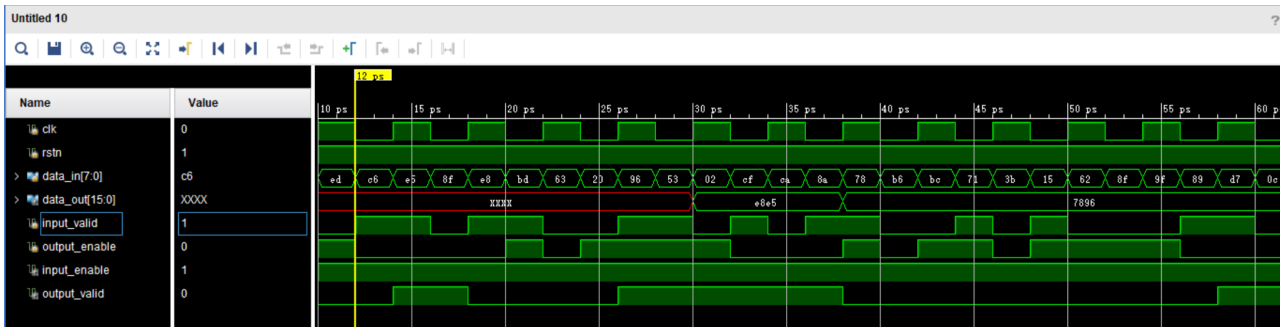

图 3: 写入和读出同时进行的同步 FIFO 模块模拟示意图

5.3 异步 FIFO 模块: 写入和读出同时进行

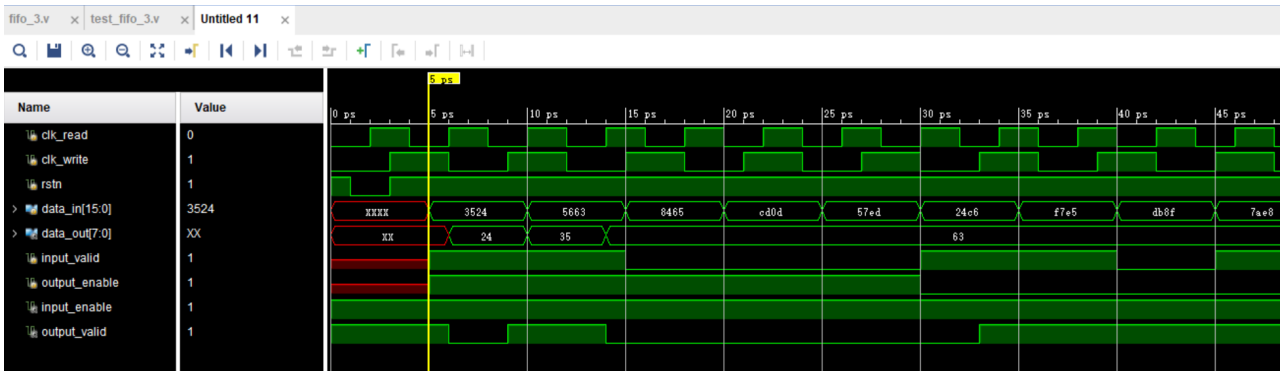

图 4: 写入和读出同时进行的异步 FIFO 模块模拟示意图

6 实验总结

通过本次实验, 我了解并实践了 FIFO 的结构和设计方法, 学习了时序电路和非阻塞赋值的应用, 对时钟信号有了更深刻的了解.

7 源代码

7.1 同步 FIFO 模块: 先写入后读出

File: fifo.v

```
`timescale 1ps/1ps

module fifo (
    input clk,
    input rstn,
    input [7:0] data_in,
    input input_valid,
    input output_enable,
    output reg [15:0] data_out,
    output reg input_enable,
    output reg output_valid
);
```

```
    reg [15:0] buffer [31:0];
    // Depth=32; use buffer[i] to refer to depth i

    reg [4:0] line_write = 5'b0;
    reg [4:0] line_read = 5'b0;

    reg writeplace = 1'b0;

    always @(*) begin
        if (line_write == 5'b11111 && writeplace == 1'b1) begin
            input_enable = 0;
            output_valid = 1;
        end
        else begin
            input_enable = 1;
            output_valid = 0;
        end
    end

    always @(posedge clk or negedge rstn) begin
        if (rstn == 0) begin // 复位
            line_write <= 5'b0;
            line_read <= 5'b0;
        end
        else begin
            if (input_enable && input_valid) begin
                if (writeplace) begin
                    buffer[line_write][15:8] <= data_in[7:0];
                    line_write <= line_write + 1;
                    writeplace <= 1'b0;
                end
                else begin
                    buffer[line_write][7:0] <= data_in[7:0];
                    writeplace <= 1'b1;
                end
            end
            else if (output_enable && output_valid) begin
                data_out[15:0] <= buffer[line_read][15:0];
                line_read <= line_read + 1;
            end
        end
    end
endmodule
```

File: test_fifo.v

```
`timescale 1ps/1ps

module test_fifo();
    reg clk, rstn;
    reg [7:0] data_in;
    wire [15:0] data_out;
    reg input_valid, output_enable;
```

```
    wire input_enable, output_valid;

    fifo inst_fifo(
        .clk(clk),
        .rstn(rstn),
        .data_in(data_in),
        .input_valid(input_valid),
        .output_enable(output_enable),
        .data_out(data_out),
        .input_enable(input_enable),
        .output_valid(output_valid)
    );


    always #2 begin
        clk = ~clk;
    end

    initial begin
        clk = 1'b0;
        rstn = 1'b1;
        #1 rstn = 1'b0;
        #2 rstn = 1'b1;
    end

    always begin
        #3;
        data_in = $random() % 9'b1_0000_0000;
    end

    always begin
        #6;
        input_valid = $random() % 2;
        output_enable = $random() % 2;
    end
endmodule
```

7.2 同步 FIFO 模块: 写入和读出同时进行

File: fifo_2.v

```
`timescale 1ps/1ps

module fifo_2 (
    input clk,
    input rstn,
    input [7:0] data_in,
    input input_valid,
    input output_enable,
    output reg [15:0] data_out,
    output reg input_enable,
    output reg output_valid
```

```
);
    reg [15:0] buffer [31:0];
    // Depth=32; use buffer[i] to refer to depth i

    reg [4:0] line_write = 5'b0;
    reg [4:0] line_read = 5'b0;

    reg writeplace = 1'b0;

    always @(*) begin
        // If there is enough to read, then read it
        if (line_write >= line_read && writeplace) begin
            output_valid = 1;
        end
        else begin
            output_valid = 0;
        end

        // If buffer is not full, then writing is allowed
        if (line_write == 5'b11111 && writeplace) begin
            input_enable = 0;
        end
        else begin
            input_enable = 1;
        end
    end

    always @(posedge clk or negedge rstn) begin
        if (rstn == 0) begin // 复位
            line_write <= 5'b0;
            line_read <= 5'b0;
        end
        else begin
            if (input_enable && input_valid) begin
                if (writeplace) begin
                    buffer[line_write][15:8] <= data_in[7:0];
                    line_write <= line_write + 1;
                    writeplace <= 1'b0;
                end
                else begin
                    buffer[line_write][7:0] <= data_in[7:0];
                    writeplace <= 1'b1;
                end
            end

            if (output_enable && output_valid) begin
                if (line_write == line_read) begin
                    data_out[15:0] <= {data_in[7:0], buffer[line_read][7:0]};
                end
                else begin
                    data_out[15:0] <= buffer[line_read][15:0];
                    line_read <= line_read + 1;
                end
```

```
            end
        end
    end

endmodule
```

File: test_fifo_2.v

```
`timescale 1ps/1ps

module test_fifo();
    reg clk, rstn;
    reg [7:0] data_in;
    wire [15:0] data_out;
    reg input_valid, output_enable;
    wire input_enable, output_valid;

    fifo_2 inst_fifo_2(
        .clk(clk),
        .rstn(rstn),
        .data_in(data_in),
        .input_valid(input_valid),
        .output_enable(output_enable),
        .data_out(data_out),
        .input_enable(input_enable),
        .output_valid(output_valid)
    );


    always #2 begin
        clk = ~clk;
    end

    initial begin
        clk = 1'b0;
        rstn = 1'b1;
        #1 rstn = 1'b0;
        #2 rstn = 1'b1;
    end

    always begin
        #3;
        data_in = $random() % 9'b1_0000_0000;
    end

    always begin
        #6;
        input_valid = $random() % 2;
        output_enable = $random() % 2;
    end
endmodule
```

7.3 异步 FIFO 模块: 写入和读出同时进行

File: fifo_3.v

```
`timescale 1ps/1ps

module fifo_3 (
    input clk_read,
    input clk_write,
    input rstn,
    input [15:0] data_in,
    input input_valid,
    input output_enable,
    output reg [7:0] data_out,
    output reg input_enable,
    output reg output_valid
);
    reg [15:0] buffer [31:0];
    // Depth=32; use buffer[i] to refer to depth i

    reg [4:0] line_write = 5'b0;
    reg [4:0] line_read = 5'b0;

    reg readplace = 1'b0;

    always @(*) begin
        // If there is enough to read, then read it
        if ((line_write > line_read) || (line_write == line_read && !readplace)) begin
            output_valid = 1;
        end
        else begin
            output_valid = 0;
        end

        // If buffer is not full, then writing is allowed
        if (line_write == 5'b11111) begin
            input_enable = 0;
        end
        else begin
            input_enable = 1;
        end
    end

    always @(posedge clk_write or negedge rstn) begin
        if (rstn == 0) begin // 复位
            line_write <= 5'b0;
        end
        else if (input_enable && input_valid) begin
            buffer[line_write][15:0] <= data_in[15:0];
            line_write <= line_write + 1;
        end
    end

    always @(posedge clk_read or negedge rstn) begin
```

```
        if (rstn == 0) begin // 复位
            line_read <= 5'b0;
            readplace <= 1'b0;
        end
        else if (output_enable && output_valid) begin
            if (readplace) begin
                if (line_write == line_read && input_valid) begin
                    data_out[7:0] <= data_in[15:8];
                end
                else
                begin
                    data_out[7:0] <= buffer[line_read][15:8];
                end
                readplace <= 1'b0;
                line_read <= line_read + 1;
            end
            else begin
                if (line_write == line_read && input_valid) begin
                    data_out[7:0] <= data_in[7:0];
                end
                else
                begin
                    data_out[7:0] <= buffer[line_read][7:0];
                end
                readplace <= 1'b1;
            end
        end
    end
endmodule
```

File: test_fifo_3.v

```
`timescale 1ps/1ps

module test_fifo();
    reg clk_read, clk_write, rstn;
    reg [15:0] data_in;
    wire [7:0] data_out;
    reg input_valid, output_enable;
    wire input_enable, output_valid;

    fifo_3 inst_fifo_3(
        .clk_read(clk_read),
        .clk_write(clk_write),
        .rstn(rstn),
        .data_in(data_in),
        .input_valid(input_valid),
        .output_enable(output_enable),
        .data_out(data_out),
        .input_enable(input_enable),
        .output_valid(output_valid)
    );
```

```
    always #3 begin
        clk_write = ~clk_write;
    end

    always #2 begin
        clk_read = ~clk_read;
    end

    initial begin
        clk_read = 1'b0;
        clk_write = 1'b0;
        rstn = 1'b1;
        #1 rstn = 1'b0;
        #2 rstn = 1'b1;
    end

    always begin
        #3;
        data_in = $random() % 17'b1_0000_0000_0000_0000;
    end

    always begin
        #6;
        input_valid = $random() % 2;
        output_enable = $random() % 2;
    end
endmodule
```