FIFO和verilog memory型的例子

memory型

memory型指的是当你需要设计多个相同的位宽的信号时，采用的语法，例：

```
reg [4:0] addr [3:0]; // 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5bits（0~4）
```

如何使用

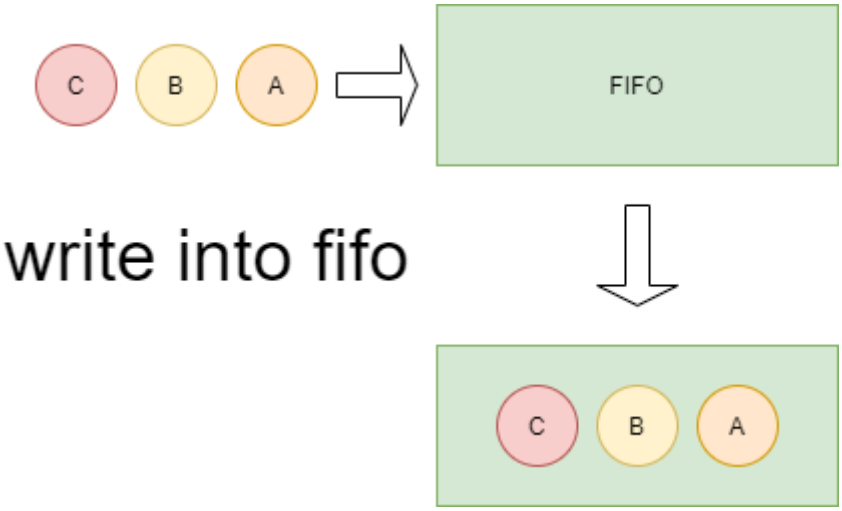
```
addr[0][3:1] = data[2:0] // 将data的0~2位的值赋值给addr的第0组信号的1~3位
temp = addr[0][3] //将addr第0组信号的第三位赋值给temp,temp是1bit
```

注意区分

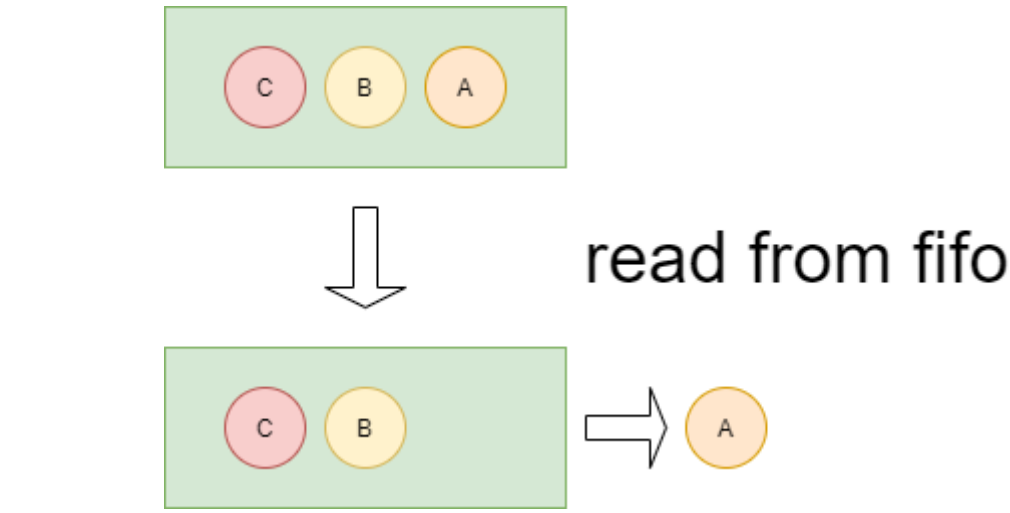
```
reg [3:0] rega; //一个4bits大小的reg型变量
reg rega [3:0]; //4个1bit大小的reg型变量组合生成的memory型变量
```

同步FIFO简单介绍

FIFO（First-In-First-Out）是一种先进先出的数据交互方式，FIFO根据时钟域不同分为**同步FIFO**和异步FIFO。下面列举同步FIFO的例子。

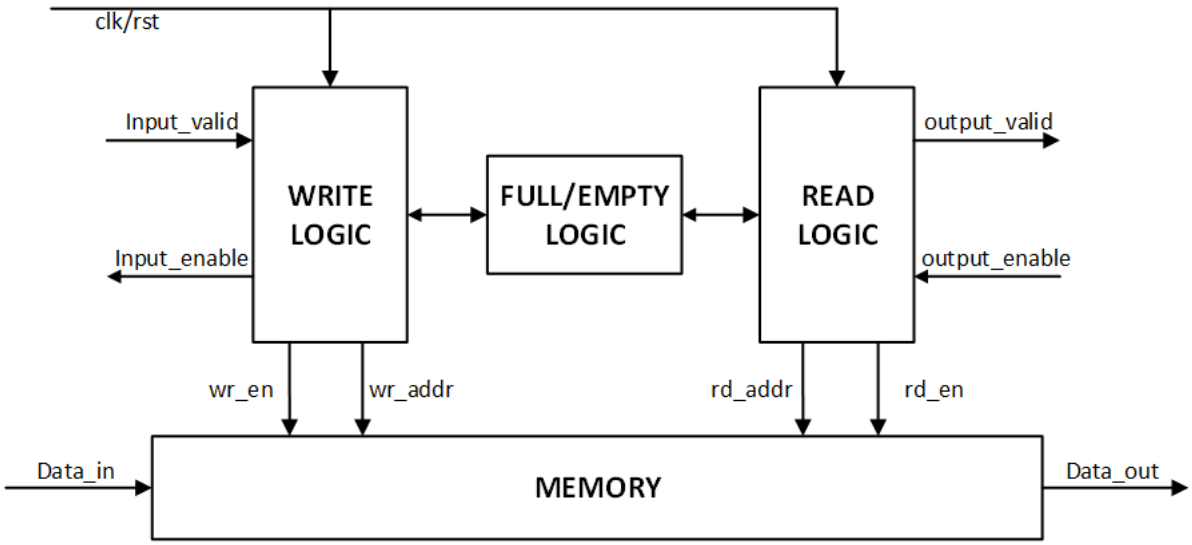


上图是一个写入FIFO的例子，图中数据ABC顺序写入FIFO，可以看到FIFO中数据按照顺序存储。



然后是从FIFO中读出数据时，根据存储的顺序（存储顺序反映了输入的顺序）依次输出数据ABC，实现了First in first out的功能。

一个通用的FIFO结构



FIFO通常包括读逻辑、写逻辑和判空/满逻辑，三个逻辑各自处理读和写的情况，并生成对应的控制信号input_enable和output_valid，以及写地址或者读地址，然后memory通过读地址或者写地址完成相应的读或者写任务。

一个8读8写的FIFO例子

8读指的是一次读出8bits数据，8写指的是一次写入8bits数据。

FIFO模块

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 2021/12/01 11:44:32
// Design Name:
// Module Name: sim_fifo
// Project Name:
// Target Devices:
```

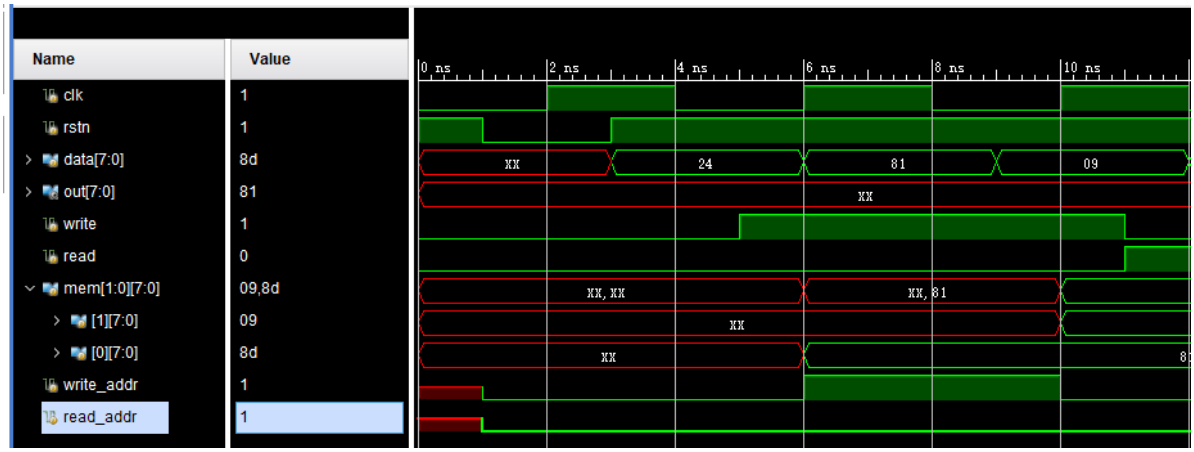
```
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module sim_fifo(
    input clk,
    input rstn,
    input write,
    input read,
    input [7:0] data,
    output reg [7:0] out
    );

    reg [7:0] mem [1:0]; // 开头提到的memory型变量例子，mem是一个深度为2宽度为8bits的memory

    reg write_addr, read_addr; // 一个读地址，一个写地址，表示读或者写的位置信息

    always @(posedge clk or negedge rstn) begin
        if (rstn == 0)begin
            write_addr <= 1'b0;
            read_addr <= 1'b0;
        end
        else begin
            if (write == 1'b1) begin
                write_addr <= ~write_addr; // 更新为下一个写地址的位置
                mem[write_addr][7:0] <= data[7:0]; // 在memory中写入数据
                /*注意这里非阻塞赋值的特点，mem所取的write_addr的值是这次更新之前的值*/
                /*例如上升沿到来，write_addr为0，则memory在第0个地址写入，write_addr更新为1*/
            end
            else if (read == 1'b1) begin  // 读操作同上
                read_addr <= ~read_addr;
                out[7:0] <= mem[read_addr][7:0];
            end
            else begin

            end
        end
    end


endmodule
```

Testbench

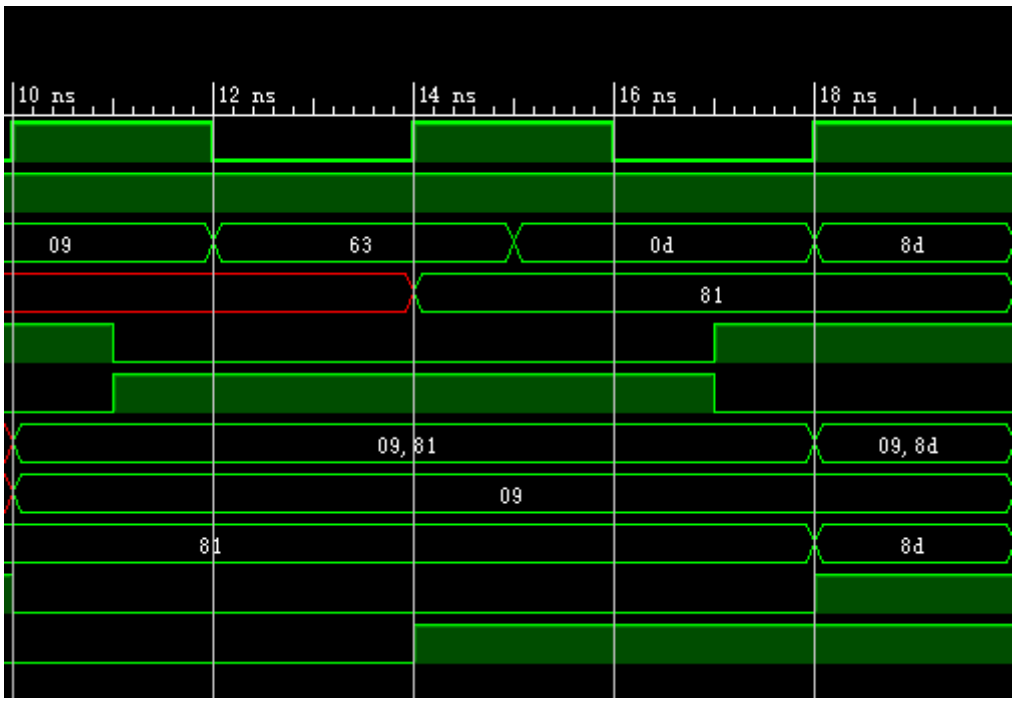
```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 2021/12/01 11:45:04
// Design Name:
// Module Name: test_sim_fifo
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module test_sim_fifo(

    );

    reg clk, rstn;
    reg [7:0] data;
    wire [7:0] out;
    reg write, read;

    sim_fifo inst_sim_fifo_0(.clk(clk),
        .rstn(rstn),
        .write(write),
        .read(read),
        .data(data),
        .out(out));

    always #2 begin
        clk = ~clk;
    end

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

    always begin
        #3;
```

```
54          data = $random()%9'b1_0000_0000;
55      end
56
57      always begin
58          #5;
59          write = 1'b1;
60          #6;
61          write = 1'b0;
62          read = 1'b1;
63          #6;
64          write = 1'b1;
65          read = 1'b0;
66          #3;
67          $finish;
68      end
69
70  endmodule
```

参考波形



如图memory中首先写入第0个位置，然后写入第1个位置



接上图，写入0x81和0x09之后，开始读数据，先进先出于是读出0x81

大家可以自行跑出波形一一分析

注：

- 关于memory的用法可以查阅verilog教程中关于memory类型的部分，简单掌握用法即可。
- FIFO也可查看相关资料。