上海交通大學

SHANGHAI JIAO TONG UNIVERSITY

实验报告

课程名称：数字逻辑设计

项目名称：Lab3

学院(系)：电子信息与电气工程学院

专　　业：微电子科学与工程

学生姓名：王赟恺　　学号：522031910274

2024 年 05 月 04 日

目录

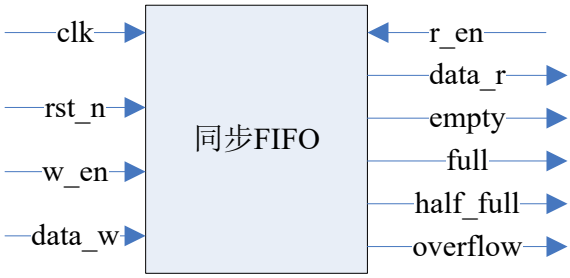
1 FIFO_REFERENCE

1.1 实验电路功能

同步 FIFO,深度为 16，每个存储单元位宽为 8 位，产生空、满、半满、溢出标志。每次读写 FIFO 数据位宽都为 8 位。

其中，clk：输入时钟；rst_n：输入复位信号，低电平有效；w_en：写使能；r_en：读使能；data_w：写入 FIFO 的数据；data_r：从 FIFO 中读出的数据；empty：读空信号，指示 FIFO 为空；full：写满信号，指示 FIFO 为满；half_full：半满信号；overflow：溢出信号，当 FIFO 已经满的时候，继续有写入的数据时跳为 1。

1.2 实验电路模块端口图



同步 FIFO 模块端口图

1.3 设计思路

同步 FIFO 的设计核心在于对于写指针和读指针的维护，以及如何根据两个指针之间的情况判断空、满、半满、溢出的情况。

采用参数化设计，设定 parameter fifo_depth = 16, parameter fifo_bitwidth = 8。以实现模块更好的通用性。适用于 fifo_depth 取 2 的整数幂，fifo_bitwidth 取任意值的情况。

维护写指针和读指针:

当 r_en 有效且 FIFO 不处于空状态，读操作有效，读指针前进一位，指向下一次有效读取的字节位置。

当 w_en 有效且 FIFO 不处于满状态，写操作有效，写指针前进一位，指向下一次有效写的字节位置。

根据写指针和读指针的位置判断空、满、半空、溢出情况:

为了区分空和全满的情况，将读指针和写指针都增加额外的一位，来表示读指针和写指针追逐的情况。

当 FIFO 空时，读指针和写指针的所有位都一样。

当 FIFO 满时，读指针和写指针相差 16，这是它们增加的额外的一位不相同，剩余 4 位相同

当 FIFO 半满时，读指针和写指针相差 8，增加的额外一位可能相同，(FIFO 的地址未跨越 32)，也可能不同(FIFO 的地址跨越 32)，第 4 位不相同，剩余 3 位相同。

1.4 代码分析

参数设计部分:

```
#( parameter fifo_depth = 16,
       parameter fifo_bitwidth = 8)
localparam bits_fifo_depth =
$clog2(fifo_depth);   //calculate the code length of
fifo_depth
```

判定标志位部分：

```
assign  empty = (rd_ptr[bits_fifo_depth : 0] ==
wr_ptr[bits_fifo_depth : 0]);
assign  full = ((rd_ptr[bits_fifo_depth] ==
~wr_ptr[bits_fifo_depth]) & (rd_ptr[bits_fifo_depth
- 1 : 0] == wr_ptr[bits_fifo_depth - 1 : 0]));
assign  half_full = ((rd_ptr[bits_fifo_depth - 1] ==
~wr_ptr[bits_fifo_depth - 1]) &
(rd_ptr[bits_fifo_depth - 2 : 0] ==
wr_ptr[bits_fifo_depth - 2 : 0]));

//overflow
always @(posedge clk or negedge rst_n)
if(!rst_n)
   overflow <= 1'b0;
else
       overflow <= full & w_en;
```

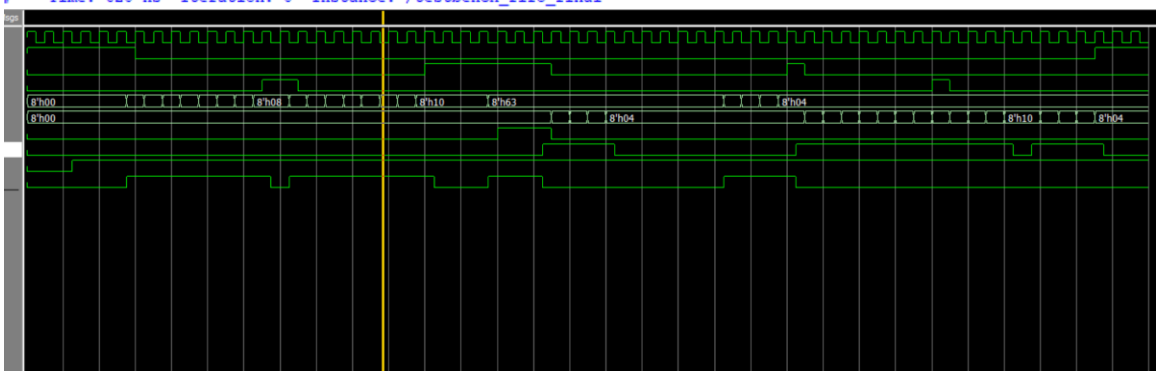
维护读写指针更新部分:

```
//wr_ptr
always @(posedge clk or negedge rst_n)
if(!rst_n)
   wr_ptr <= 'b0;
```

```
else if(w_en & ~full)
   wr_ptr <= wr_ptr + 1'b1;
else
   wr_ptr <= wr_ptr;

//read_ptr
always @(posedge clk or negedge rst_n)
if(!rst_n)
   rd_ptr <= 'b0;
else if(r_en & ~empty)
   rd_ptr <= rd_ptr + 1'b1;
else
       rd_ptr <= rd_ptr;
```

1.5 波形图

```
#
# after reading 4 data, right, empty = 0 full = 0 half_full = 0 overflow = 0
#
# storing           1 again  w_en=1 r_en=0
#
# storing           2 again  w_en=1 r_en=0
#
# storing           3 again  w_en=1 r_en=0
#
# storing           4 again  w_en=1 r_en=0
#
# reading data            5     5
#
# reading data            6     6
#
# reading data            7     7
#
# reading data            8     8
#
# reading data            9     9
#
# reading data           10    10
#
# reading data           11    11
#
# reading data           12    12
#
# reading data           13    13
#
# reading data           14    14
#
# reading data           15    15
#
# reading data           16    16
#
# reading data            1     1
#
# reading data            2     2
#
# reading data            3     3
#
# reading data            4     4
#
# ********************
# done, without error
# ********************
#
# ** Note: $stop    : E:/ToDo/Verilog2024/Lab3/code/fifo_tb2024.v(177)
#    Time: 620 ns  Iteration: 0  Instance: /testbench_fifo_final
```

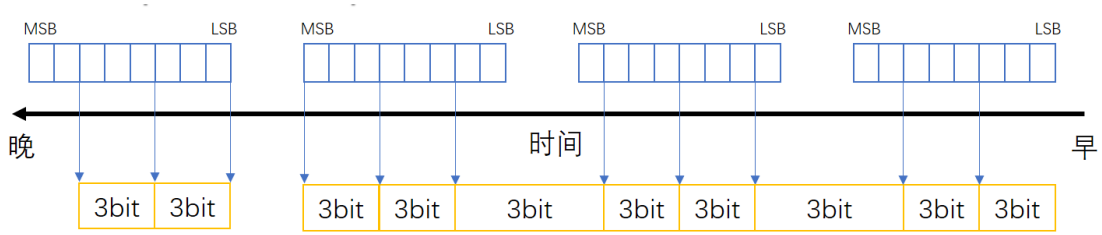


提供的 Testbench 通过，观察输出波形符合 FIFO 功能

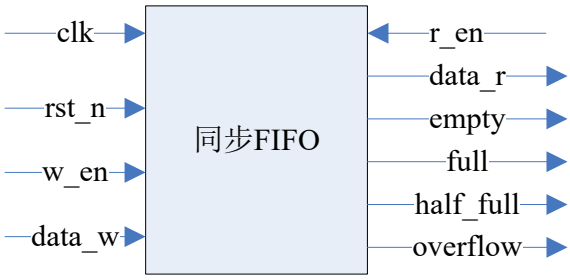
2 FIFO8to3

2.1 实验电路功能

基于 FIFO 进行位宽的转换，要求在数据写入 FIFO 时使用 8bit 位宽，读出 FIFO 时使用 3bit 位宽（从低位开始读出，如下图所示）。



2.2 实验电路模块端口图



2.3 设计思路

在 FIFO 的基础上更改，采用了状态机的写法。标志位信号的判断不改变，写指针和写操作不改变，将读指针和读操作的改变放入状态机中。

根据第一次完全在同一字节中读出的 3bit 的位置(共三种情况)，设为 STATE0, STATE3, STATE6。STATE0 经过有效读依次进入 STATE1，STATE2。STATE3 经过有效读依次进入 STATE4，STATE5。STATE6 经过有效读依次进入 STATE7，STATE0。构成状态机的转移方程。

2.4 代码分析

状态机组合逻辑转移方程(以 STATE0，STATE1 为例)

```
always @(*)
begin
   case(state)
   STATE0:
      if(r_en & !empty)
      begin
         state_nxt = STATE1;
         data_r_nxt = fifo_mem[rd_ptr[3:0]][2:0];
         rd_ptr_nxt = rd_ptr;
```

```
        end
        else
        begin
            state_nxt = STATE0;
            data_r_nxt = data_r;
            rd_ptr_nxt = rd_ptr;
        end
    STATE1:
        if(r_en & !empty)
        begin
            state_nxt = STATE2;
            data_r_nxt = fifo_mem[rd_ptr[3:0]][5:3];
            rd_ptr_nxt = rd_ptr + 1'b1;
        end
        else
        begin
            state_nxt = STATE1;
            data_r_nxt = data_r;
            rd_ptr_nxt = rd_ptr;
        end
    default:state_nxt = 3'b000;
    endcase
     end
```

状态机时序部分

```
always @(posedge clk or negedge rst_n)
begin
    if(!rst_n) begin
        rd_ptr <= 5'b00000;
    end
    else
        rd_ptr <= rd_ptr_nxt;
```

```
    end
always @(posedge clk or negedge rst_n)
begin
   if(!rst_n) begin
      data_r <= 3'b000;
   end
   else
      data_r <= data_r_nxt;
    end
```

2.5 测试代码分析

测试代码的主体复用了助教给出的 SystemVerilog 版本。但是由于 Verilog 不支持 C 中的队列数据结构，在自己编写的 TestBench 中用 Verilog 的 Task 特性编写了模拟队列入队和出队的 Write 和 Read.由于 TestBench 除了 DUT 模块部分，代码不用考虑可综合性，故采用 C 语言的写法实现 Task，但是期间会受到 Verilog 语法的约束。比如变量不能像 C 中一样声明为 int 等类型，在 Verilog 只能使用 reg 类型，而且要注意位宽的控制，防止隐藏的溢出。

验证的思路是在时钟信号的下降沿产生标准模型和待测试模型的输入，待下一个时钟信号上升沿待测试模型产生输出(有延时),再与标准模型产生的输出比较(无延时)。

Read 和 Write 函数直接维护 space_avail 变量来产生标准的参照信号(output_golden_empty/full/half_full)等，无需手动模拟标准的参照信号再输入 TestBench。

```
task write;
    input w_en;
    input [7:0] in_data;
    output out_golden_full;
    output out_golden_empty;
    output out_golden_half_full;
    output out_golden_overflow;

    if(w_en) begin
        if(space_avail < 7'd8) begin
            out_golden_overflow = 1'b1;
            out_golden_full = 1'b1;
            out_golden_empty = 1'b0;
```

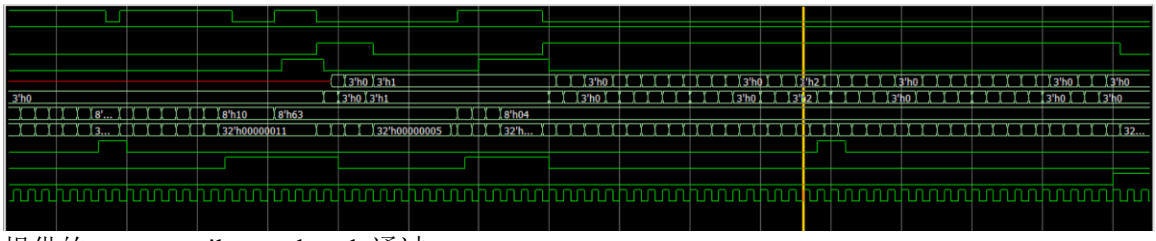
```
            out_golden_half_full = 1'b0;
            $display("CAN'T WRITE WHEN FIFO IS FULL,
OVERFLOW\n");
        end
        else begin
            data_stored[tail_fifo_ptr[6:0] +: 8] =
in_data;
            tail_fifo_ptr = tail_fifo_ptr + 8;
            space_avail = space_avail - 8;


            if(space_avail == 7'd64)
                out_golden_half_full = 1'b1;
            else
                out_golden_half_full = 1'b0;

            if(space_avail < 7'd8)
                out_golden_full = 1'b1;
            else
                out_golden_full = 1'b0;
            out_golden_empty = 1'b0;
            out_golden_overflow = 1'b0;
        end
    end

    else
        out_golden_overflow = 1'b0;

    endtask
```

2.6 隐患

1.未测试 r_en 和 w_en 同时为高电平的情形

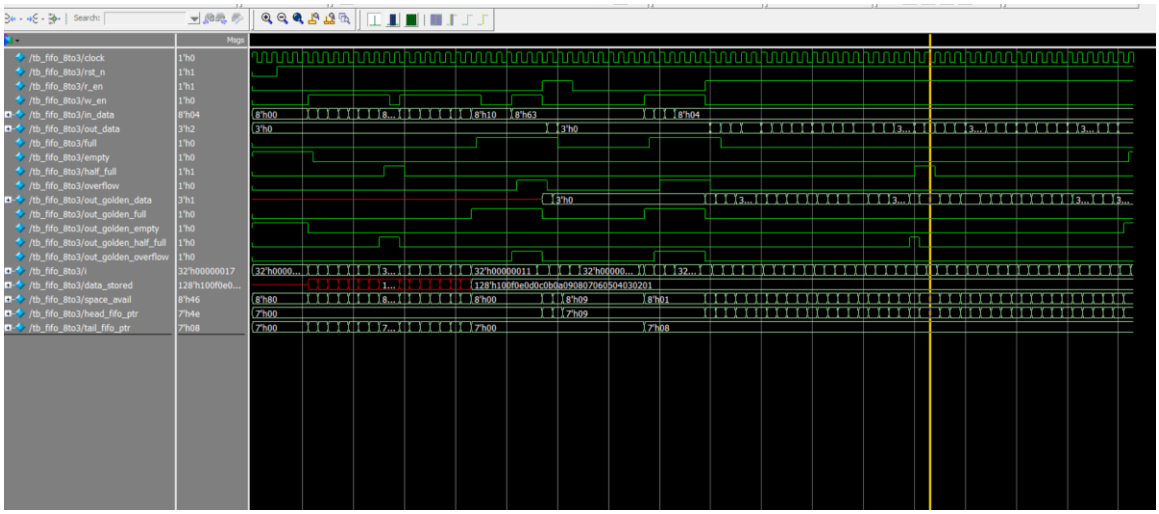
2.7 波形图与仿真结果

```
# reading data 0   0
#
# reading data 2   2
#
# reading data 1   1
#
# reading data 4   4
#
# reading data 5   5
#
# reading data 0   0
#
# reading data 0   0
#
# reading data 3   3
#
# reading data 0   0
#
# reading data 5   5
#
# reading data 1   1
#
# reading data 0   0
#
# reading data 7   7
#
# reading data 0   0
#
# reading data 6   6
#
# reading data 3   3
#
# reading data 0   0
#
# reading data 0   0
#
# reading data 2   2
#
# reading data 4   4
#
# reading data 0   0
#
# reading data 0   0
#
# ********************
# done, without error
# ********************
#
# ** Note: $stop    : E:/ToDo/Verilog2024/Lab3/code/fifo_tb2024_8to3.sv(201)
#    Time: 880 ns  Iteration: 0  Instance: /testbench_fifo_final
```



提供的 systemverilog testbench 通过
观察输出波形符合 FIFO 要求

```
#
# reading data 0  actually get 0
#
# reading data 2  actually get 2
#
# reading data 1  actually get 1
#
# reading data 4  actually get 4
#
# reading data 5  actually get 5
#
# reading data 0  actually get 0
#
# reading data 0  actually get 0
#
# reading data 3  actually get 3
#
# reading data 0  actually get 0
#
# reading data 5  actually get 5
#
# reading data 1  actually get 1
#
# reading data 0  actually get 0
#
# reading data 7  actually get 7
#
# reading data 0  actually get 0
#
# reading data 6  actually get 6
#
# reading data 3  actually get 3
#
# reading data 0  actually get 0
#
# reading data 0  actually get 0
#
# reading data 2  actually get 2
#
# reading data 4  actually get 4
#
# reading data 0  actually get 0
#
# reading data 0  actually get 0
#
# CAN'T READ WHEN FIFO IS EMPTY
#
# ** Note: $stop    : E:/ToDo/Verilog2024/Lab3/code/fifo_mytb2024_8to3.v(50)
#    Time: 865 ns  Iteration: 1  Instance: /tb_fifo_8to3
```



自己编写的 testbench 通过
观察输出波形符合 FIFO 要求