FPGA 学习系列

同步 FIFO 实验手册

Synchronization FIFO Experiment

文档版本 01

发布日期 2023-3-18

MYFPGA.CN—技术的执著

偏向电子类的知识分享平台

关于本文档

作者信息

作者	祝浩雨	**时间**	2023-03-18
评审		**时间**	
签发		**时间**	

内容简介

本文档介绍了 FIFO 的全流程，完整、全面的进行了 FIFO 的分析，设计，详尽的展现了实验的每一步。

修订记录

修改记录累积了每次文档更新的说明。最新版本的文档包含以前所有文档版本的更新内容。

文档版本 01 (2023-03-18)

第一次正式发布。

目录

实验内容

设计一个实验，实现以下功能：

1、 设计一个 FIFO，具有读写控制逻辑、空满指示信号。

FIFO 相关名词解释

FIFO

FIFO 是一种先进先出的数据缓存器，在逻辑设计里面用的非常多，FIFO 设计可以说是逻辑设计人员必须掌握的常识性设计。FIFO 一般用在隔离两边读写带宽不一致，或者位宽不一样的地方。

使用 FIFO 一般有两个方法，第一个方法是直接调用官方的 FIFO IP，另外一个方法是自己设计 FIFO 控制逻辑。我们本节介绍的是自己设计 FIFO 逻辑。

FIFO 包括同步 FIFO 和异步 FIFO 两种，同步 FIFO 有一个时钟信号，读和写逻辑全部使用这一个时钟信号，异步 FIFO 有两个时钟信号，读和写逻辑用的各种读写时钟，本实验讲的全部是同步 FIFO。

FIFO 与普通存储器 RAM 的区别是没有外部读写地址线，使用起来非常简单，但缺点就是只能顺序写入数据，顺序的读出数据，其数据地址由内部读写指针自动加 1 完成，不能像普通存储器那样可以由地址线决定读取或写入某个指定的地址。FIFO 本质上是由 RAM 加读写控制逻辑构成的一种先进先出的数据缓冲器。

FIFO 的常见参数

参数	描述
FIFO 的宽度	FIFO 一次读写操作的数据位
FIFO 的深度	指的是 FIFO 可以存储多少个 N 位的数据（如果宽度为 N）
满标志	FIFO 已满或将要满时由 FIFO 的状态电路送出的一个信号，以阻止 FIFO 的写操作继续向 FIFO 中写数据而造成溢出（overflow）
空标志	FIFO 已空或将要空时由 FIFO 的状态电路送出的一个信号，以阻止 FIFO 的读操作继续从 FIFO 中读出数据而造成无效数据的读出（underflow）
读时钟	读操作所遵循的时钟，在每个时钟沿来临时读数据
写时钟	写操作所遵循的时钟，在每个时钟沿来临时写数据

FIFO 读写指针的工作原理

写指针总是指向下一个将要被写入的单元，读指针总是指向当前要被读出的数据，复位时，写指针和读指针都指向第 1 个单元(编号为 0)。

FIFO 的“空”/“满”检测：

FIFO 设计的关键：产生可靠的 FIFO 读写指针和生成 FIFO“空”/“满”状态标志。

当读写指针相等时，表明 FIFO 为空，这种情况发生在复位操作时，或者当读指针读出

FIFO 中最后一个字后，追赶上了写指针时，如下图所示：

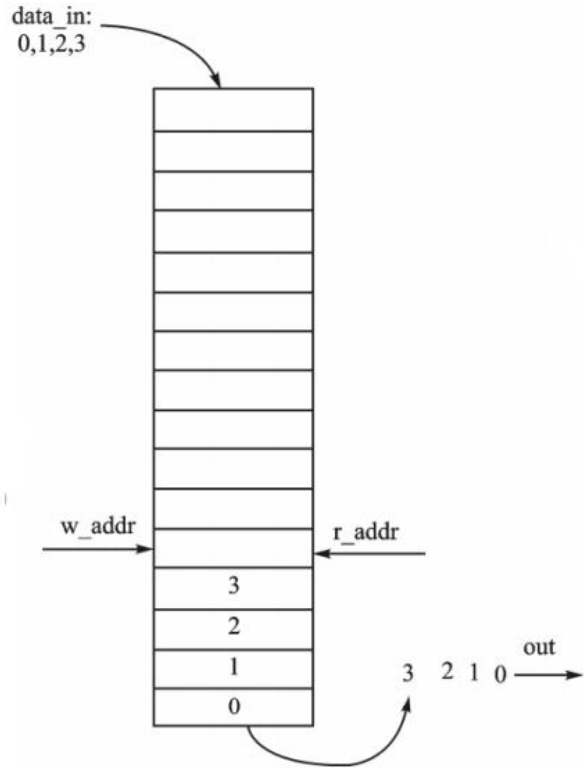


图 1 FIFO 读写地址示意图

当读写指针再次相等时，表明 FIFO 为满，这种情况发生在，当写指针转了一圈，折回来又追上了读指针，如下图：

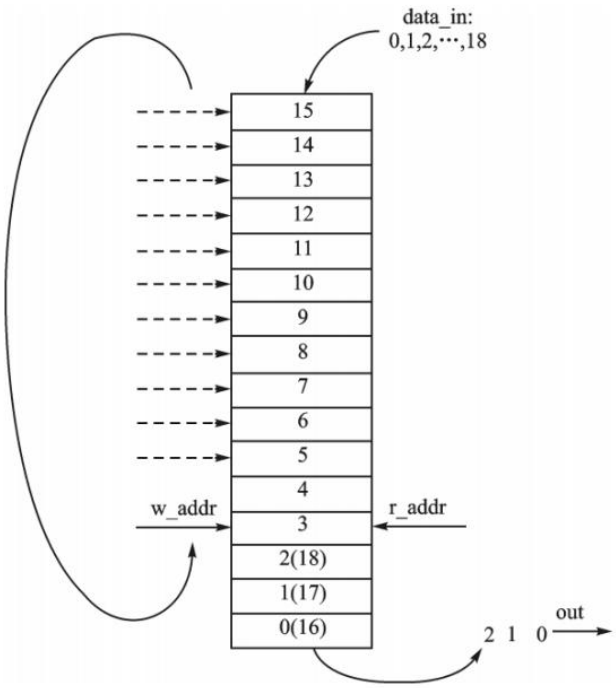


图 2 FIFO 读写地址转圈示意图

实验原理

区分是满状态还是空状态的方法

设计一个计数器，该计数器(count)用于指示当前 FIFO 中数据的个数。计数器变化的过程如下：

复位的时候，count 初始化为 0；

如果 FIFO 读使能和写使能同时有效的时候，count 不加也不减；表示同时对 FIFO 进行读写操作的时候，FIFO 中的数据个数不变；

如果写使能有效且 full=0，则 count+1;表示写操作且 FIFO 未满的时候，FIFO 中的数据个数增加了 1；

如果读使能有效且 empty=0，则 count-1；表示读操作且 FIFO 未空的时候，FIFO 中的数据个数减少了 1；

count=0 的时候，表示 FIFO 空，需要设置 empty=1;如果 count=16 的时候，表示 FIFO 现在已经满，需要设置 full=1。

下面设计一个 16*8（16 是深度，8 是位宽）的 FIFO

1、功能定义：

用 16*8（16 是深度，8 是位宽）RAM 实现一个同步先进先出（FIFO）队列设计。由写使能端控制数据流写入 FIFO，并由读使能控制 FIFO 中数据的读出。写入和读出的操作由时钟的上升沿触发。当 FIFO 的数据满和空的时候分别设置相应的高电平加以指示。

2、FIFO 信号定义：

信号名称	I/O	功能描述	备注
Rst	In	全局复位（低有效）	
Clk	In	全局时钟	频率 10Mhz;
Wr_en	In	低有效写使能	
Rd_en	In	低有效读使能	
Data_in [7：0]	In	数据输入端	
Data_out [7：0]	Out	数据输出端	
Empty	Out	空指示信号	为高时表示 FIFO 空
Full	Out	满指示信号	为高时表示 FIFO 满

3、顶层模块划分及功能实现

该同步 FIFO 可划分为如下四个模块：

1	存储器模块（RAM）	用于存储数据
2	读地址模块（rd_addr）	用于读地址的产生
3	写地址模块（wr_addr）	用于写地址的产生
4	标志模块（flag_gen）	用于产生 FIFO 当前空满状态

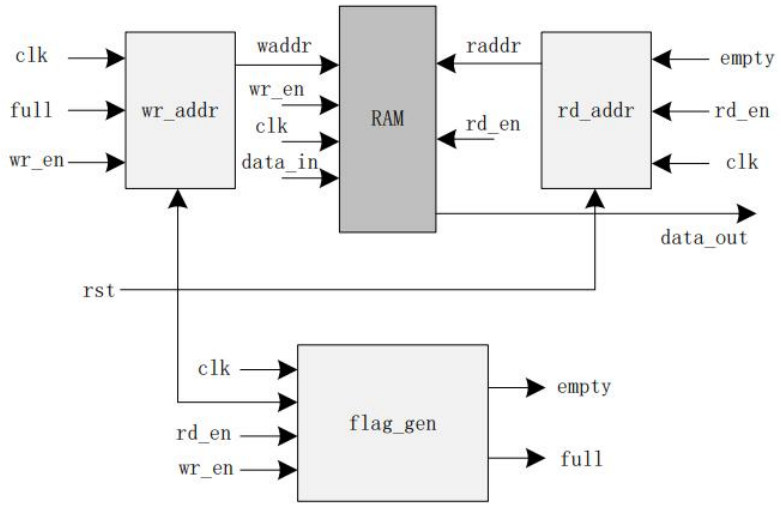


图 3 同步 FIFO 的模块划分

① RAM 模块

本设计中的 FIFO 采用采用 16*8 双口（一个写口，一个读口）RAM，以循环读写的方式实现；

读地址：根据 RD_addr_gen 模块产生的读地址，在读使能(rd_en)为高电平的时候，将 RAM 中 rd_addr[3:0]地址中的对应单元的数据在时钟上升沿到来的时候，读出到 data_out[7:0]中。

写地址：根据 WR_addr_gen 产生的写地址，在写使能(wr_en)为高电平的时候，将输入数据(data_in[7:0])在时钟上升沿到来的时候，写入 wr_addr[3:0]地址对应的单元。

② WR_addr_ge

该模块用于产生 FIFO 写数据时所用的地址。由于 16 深度的 RAM 单元可以用 4 位地址线寻址。本模块用 4 位计数器(wr_addr[3:0])实现写地址的产生。

在复位信号为 0（复位有效）时，写地址值为 0。

如果 FIFO 未满(full=0，FIFO 满了如果还继续写入，会导致新数据覆盖掉原来的旧数据)且有写使能（wr_en）有效，则 wr_addr[3:0]加 1；否则写地址不变。

③ RD_addr_gen

该模块用于产生 FIFO 读数据时所用的地址。由于 16 个 RAM 单元可以用 4 位地址线寻址。本模块用 4 位计数器(rd_addr[3:0])实现读地址的产生。

在复位信号为 0（复位有效）时，读地址值为 0。

如果 FIFO 未空（empty=0，FIFO 空了如果还继续读，会导致 FIFO 读出的数据不是期望的数据或者不确定的数据）且有读使能（rd_en）有效，则 rd_addr[3:0]加 1；否则读地址不变。

④ flag_gen 模块

flag_gen 模块产生 FIFO 空满标志。本模块设计没有使用读写地址判定 FIFO 是否空满（也可以使用读写地址做判断），而是设计一个计数器，该计数器(count)用于指示当前 FIFO 中数据的个数。由于 FIFO 中最多只有 16 个数据，因此采用 5 位计数器来指示 FIFO 中数据个数。具体计算如下：

复位的时候，count=0；

如果 wr_en 和 rd_en 同时有效的时候，count 不加也不减；表示同时对 FIFO 进行读写操作的时候，FIFO 中的数据个数不变。

如果 wr_en 有效且 full=0，则 count+1;表示写操作且 FIFO 未满时候，FIFO 中的数据个数增加了 1；

如果 rd_en 有效且 empty=0，则 count-1；表示读操作且 FIFO 未满时候，FIFO 中的数据个数减少了 1；

如果 count=0 的时候，表示 FIFO 空，需要设置 empty=1;如果 count=16 的时候，表示 FIFO 现在已经满，需要设置 full=1。

实验设计

FIFO 控制模块

```
module sync_fifo(
    input               clk     ,
    input               rst     ,
    input               wr_en   ,
    input               rd_en   ,
    input       [7:0]   data_in ,
    output reg  [7:0]   data_out,
    output reg          empty   ,
    output reg          full
);

reg     [3:0]   wr_addr ;
reg     [3:0]   rd_addr ;
reg     [4:0]   count   ;

parameter max_count = 5'b10000 ;
parameter max1_count = 5'b01111 ;

// 定义一个二维的 RAM，实际设计中根据 RAM 深度和宽度可能会采用 Vedor 提供的 RAM
reg     [7:0]   fifo [0 : max_count] ;

//读操作
always @ (posedge clk or negedge rst) begin
    if (rst == 1'b0)
        data_out <= 0;
    else if (rd_en==1 && empty==0)
        data_out <= fifo[rd_addr];
End

//写操作
always @ (posedge clk) begin
    if (wr_en==1 && full==0)
        fifo[wr_addr] <= data_in;
End

//更新读地址
always @ (posedge clk or negedge rst) begin
    if (rst == 1'b0)
```

```
        rd_addr <= 4'b0000;
    else if (empty==0 && rd_en==1)
        rd_addr <= rd_addr + 1;
End

//更新写地址
always @ (posedge clk or negedge rst) begin
    if (rst == 1'b0)
        wr_addr <= 4'b0000;
    else if (full==0 && wr_en==1)
        wr_addr <= wr_addr + 1;
    else
        wr_addr <= 4'b0000;
End

// ============================================================
/*
定义一个 count 计数，用于表示 FIFO 中当前剩余的数据个数，
当只有写时，count 计数加 1，
当只有读时候，count 计数减 1，
当有读有写时候，计数不变。这个标记用于产生空满状态，
当 count 计数为 0 时候，则 FIFO 当前状态为空，
当 count 计数计数为 max_count 时，则 FIFO 当前状态为满。
*/
//更新标志位
always @ (posedge clk or negedge rst) begin
    if (rst == 1'b0)
        count<=0;
    else begin
        case({wr_en,rd_en})
            2'b00:
                count <= count;
            2'b01:
                if(count !== 5'b00000)
                    count <= count - 1;
            2'b10:
                if(count !== max1_count)
                    count <= count + 1;
            2'b11:
                count <= count;
        endcase
    end
End
```

```
always @(count) begin
    if(count == 5'b00000)
        empty = 1;
    else
        empty = 0;
End

always @(count) begin
    if (count == max_count)
        full = 1;
    else
        full = 0;
end
// ============================================================

endmodule
```

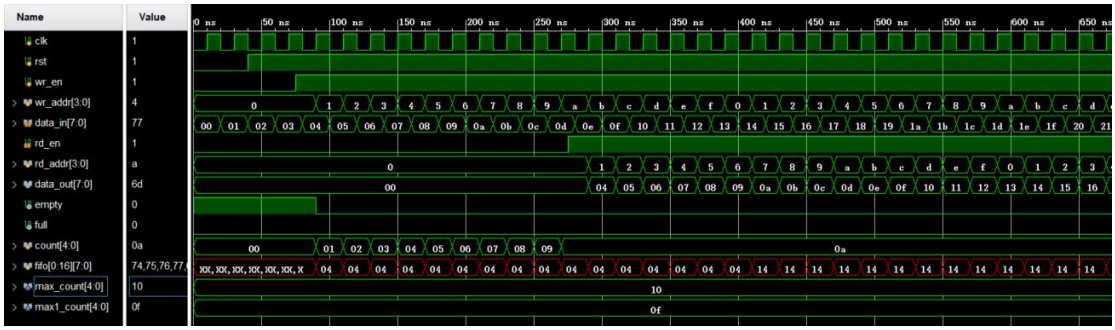
仿真测试

TestBench 文件

```
`timescale 1ns/1ps
module TB();

reg             clk         ;
reg    [7:0]    data_in     ;
reg             rd_en       ;
reg             rst         ;
reg             wr_en       ;
wire   [7:0]    data_out    ;
wire            empty       ;
wire            full        ;

initial begin
    clk = 0;
    rst = 0;
    rd_en = 0;
    wr_en = 0;
    data_in = 0;

    #40
    rst = 1 ;
    #35
    wr_en = 1;
    #200
    rd_en = 1;
End

always #20 data_in <= data_in + 1;

always #10 clk = ~clk;

sync_fifo u_sync_fifo (
    .clk        (clk),
    .data_in    (data_in),
    .data_out   (data_out),
    .empty      (empty),
    .full       (full),
    .rd_en      (rd_en),
```

```
    .rst        (rst),
    .wr_en      (wr_en)
);


endmodule
```

仿真结果：



经过测试可以看出，当复位撤销（复位信号低有效）之后，在写使能拉高有效之后，写地址即开始不停的加一累加，然后写数据也开始变化，FIFO 的 count 也开始累加计数，empty 空标记也开始变为非空。

当读使能拉高有效之后，读地址即开始不停的加一，然后读数据在下一拍也开始变化（FIFO 读有效后第二拍数据读出，FIFO 读延迟为 1 个时钟周期），可以看出第一次写 FIFO 的数据是 04，然后第一次读出 FIFO 的数据也是 04。

当仅有写的时候，FIFO 的 count 加 1，当有读有写的时候，FIFO 的 count 保持不变。

引用文献

[1]阿东.正点原子 FPGA 开发板教程