《数字逻辑》实验报告

姓名	李燕琴 杨思怡	年级	2019 级
学号	20195633 20195217	专业、班级	计算机科学与技术（卓越） 计卓 02 计卓 01

实验名称	存储器实验		
实验时间		实验地点	
实验成绩		实验性质	□验证性 □设计性 □综合性

教师评价：

□算法/实验过程正确； □源程序/实验内容提交 □程序结构/实验步骤合理；

□实验结果正确； □语法、语义正确； □报告规范；

评语：

评价教师签名（电子签名）：

一、实验目的

（1）通过实验，掌握随机存储器的原理，并学会设计实现单端口，双端口随机存储器，可以对随机存储器中指定地址的寄存器单元进行读写操作。并运用 Verilog 语言编写程序，体会程序的逻辑性，掌握器件设计开发的注意事项。在实践中，学习掌握简单、周全的编程方法。

（2）培养数字系统设计的基本能力，理解 FIFO 的定义与功能，掌握 FIFO 的 Verilog 编写方法。

二、实验项目内容

1） 利用 BASYS3 片内存储器单元实现单端口 RAM 设计（带异步读和同步读两种模式），在时钟（clk）上升沿，采集地址（addr）、输入数据（data_in）、执行相关控制信息。当写使能（we）有效，则执行写操作，否则执行读取操作。同步与异步设计仅针对读操作：对于异步 RAM 而言，读操作为异步，即地址信号有效时，控制器直接读取 RAM 阵列；对于同步 RAM 而言，地址信号在时钟上升沿被采集。并保存在寄存器中，然后使用该地址信号读取 RAM 阵列。

2） 实现双端口（同步与异步）RAM 设计，相对于单端口 RAM 而言，双端口

RAM 存在两个存取端口，并且可独立进行读写操作，具有自己的地址（addr_a、addr_b）、数据输入（din_a、din_b）/输出端口（dout_a、dout_b）。

3） 实现 FIFO 设计，FIFO 由存储单元队列或阵列构成，和 RAM 不同的是 FIFO 没有地址，第一个被写入队列的数据也是第一个从队列中读出的数据。FIFO 可以在输入输出速率不匹配时，作为临时存储单元；可用于不同时钟域中间的同步；输入数据路径和输出数据路径之间数据宽度不匹配时，可用于数据宽度调整电路。

三、实验设计

0、概念

RAM，随机访问存储器，二维存储阵列，其内部存储单元的内容在任何时候都可控制读入或写出，通常用作操作系统或其他正在运行的程序的临时存储介质（可称作系统内存）。

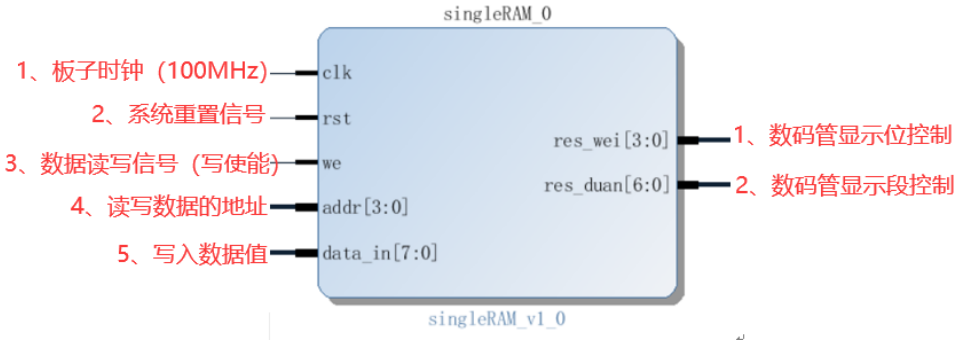
1、同步异步原理

同步中，只需捕获一个电平信号的变化或者触发信号的上升沿或下降沿，捕获后，执行程序，同时刷新相关的数据。

异步中，捕获二个及以上的电平信号或触发信号，即多个信号同时控制相应的程序的执行。

2、单端口 RAM 设计

（1）数据设计：



Private（内置数据）：

clk_1Hz，时钟分频；

[DW-1:0]ram[0:RD-1]，内置存储器；

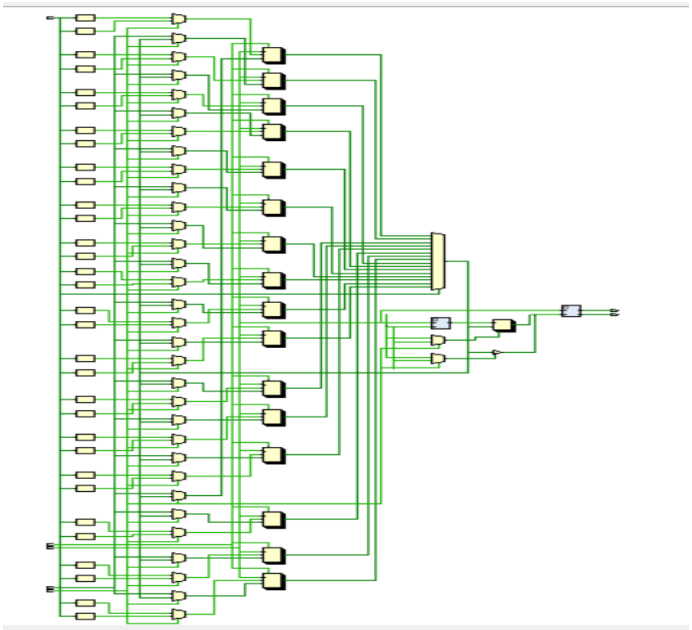
[DW-1:0]dout_syn，同步读得的数据；

[DW-1:0]dout_asyn 异步读得的数据。

（2）基本原理：

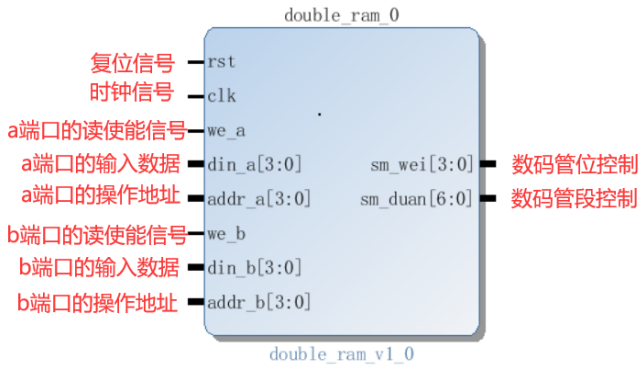
在时钟上升沿，采集地址、输入数据（data_in）、执行（we）相关控制信息。当写使能（we）有效，则执行写操作，否则执行读取操作。其中设置 clk 同步写，clk_1Hz 同步读，we 异步读。读的数据保存在寄存器中，进行显示。

（3）RTL 电路图



3、双端口 RAM 设计

（1）数据设计



Private(内置数据):

wire myclk:自定义时钟，用于适应上板与仿真的不同需求

busy_a,busy_b:a、b 端口的 busy 信号

reg [DW-1:0] mem[DP-1:0]：内置存储器

reg [DW-1:0] reg_d_a：存储 a 端口的输出内容

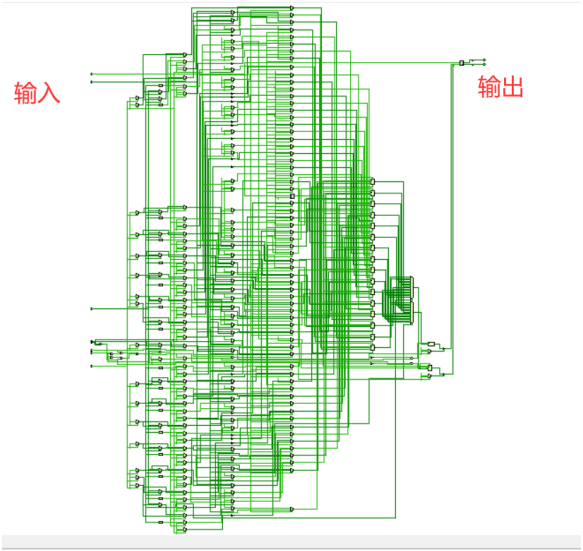
reg [DW-1:0] reg_d_b：存储 b 端口的输出内容

wire [15:0] data:数码管需显示的数据

（2）基本原理

相对于单端口 RAM 而言，双端口 RAM 存在两个存取端口，并且可独立进行读写操作，具有自己的地址（addr_a、addr_b）、数据输入（din_a、din_b）/输出端口（dout_a、dout_b）。设置两组独立端口 a,b，分别有地址 addr_a, addr_b,输入数据 din_a,din_b,输出数据 dout_a, dout_b。其中 a 端口同步读，b 端口异步读，a,b 端口均同步写数据。采用两个仲裁信号 busy_a,busy_b 解决两端口对同地址的读写冲突。当 busy 信号为高电平时表示此端口对当前操作的地址读写均有效，低电平时读写均无效。

（3）RTL 电路图

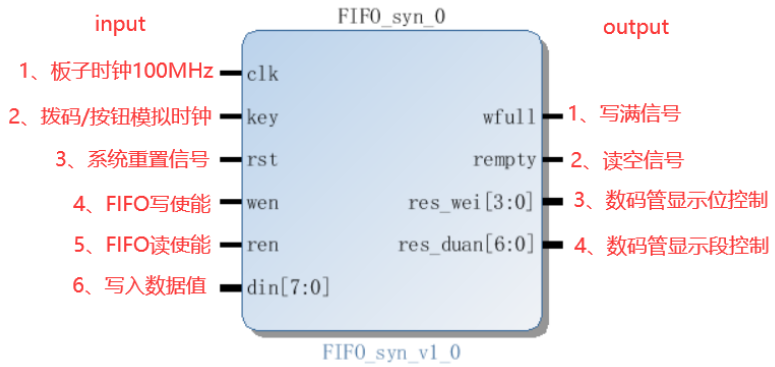


4、FIFO 设计

（1）基本原理：

FIFO 由存储单元队列或阵列构成，没有地址输入，第一个被写入队列的数据也是第一个从队列中读出的数据，其中写入读出可通过内置地址（waddr,raddr）实现，本实验基于所学，设计的是同步 FIFO，即读写均有统一时钟进行控制。每当时钟上升沿的时候，进行刷新，包括 wen = 1 时,写入数据，ren=1 时读出数据。外加写满信号 wfull 和读空信号 rempty 控制是否进行读写（二者可通过内部计数进行判断），可避免 FIFO 的读写冲突，使系统更稳定。另设置异步重置信号，使系统更完整。

（2）数据设计



Private(内置数据)：

waddr, raddr ： 内置读写指针

[DW:0]cnt： 内置 FIFO 数据记录，共有 17 种状态，0（空）~16（满）

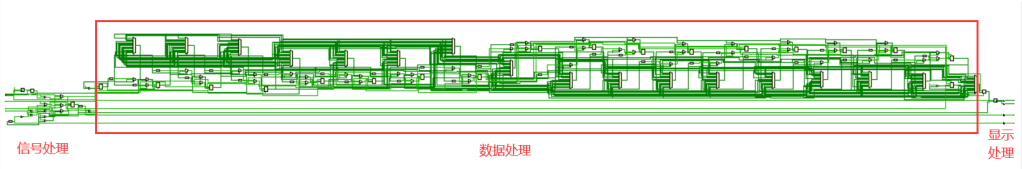
[DW-1:0]ram[0:RD-1] : 内置存储器

Clk_key: key 防抖后的结果

[DW-1:0]dout：读出数据寄存

[15:0]data={cnt,dout}：数码管数据 结果

（3）RTL 电路图



四、实验过程或算法

1、单端口 ram

第一步：时钟分频，自定义时钟分频模块 fdivider.v，输入 basys3 自带时钟 clk(100MHz),以及 clk 分频记录间隔次数 f，输出自定义时钟 myclk。

```
module fdivider(clk,f,myclk);
    input clk;
    input [31:0]f;
    output myclk;

    reg[31:0] clk_cnt=1'b0;
    reg inlineclk = 0;
    always@(posedge clk) begin
        if(clk_cnt == f[31:0]) begin
            clk_cnt<=1'b0;
            inlineclk <=~inlineclk;
        end
        else clk_cnt<=clk_cnt+1'b1;
    end
    assign myclk = inlineclk;
endmodule
```

第二步：数据写入（同步写入，异步置零），直接使用板子自带的 clk，在 wen 为 1 时同步写入数据。

```
always@(posedge clk,posedge rst) begin  // 同步写
    if(rst)
        for(i=0;i<RD;i=i+1)
            ram[i]<=8'h00;
    else begin
        if(we)  ram[addr] <= data_in;
        else ram[addr] <= ram[addr];
    end
end
```

第三步：同步读和异步读，分别使用两个数码管显示数据，便于二者比较，其中同步使用分频时钟 1Hz。

```
//--------------------上板-------------------------------
integer f = 32'd5000_0000; // halfT = 0.5s, T = 1s
fdivider fdivider01(.clk(clk),.f(f),.myclk(clk_1Hz));
// ----------------同步读-----------------------------
reg [DW-1:0]dout_syn;
always@(posedge clk_1Hz) begin
    if(~we) dout_syn <= ram[addr];
    else dout_syn <= 8'hzz;
end
// ------------------------异步读-------------------
reg [DW-1:0]dout_asyn;
always@(*) begin
    if(~we) dout_asyn <= ram[addr];
    else dout_asyn <= 8'hzz;
end
```

第四步：读出的数据显示，直接调用自己写好的 display.v 数码管显示模块即可，将同步、异步读到的数据进行显示，会发现同步的会有相应的延迟显示。

2、双端口 RAM 的设计

第一步：时钟设计包括上板与仿真两种，以 flag 做时钟选择信号，flag 高电平选择上板时钟

```
if(flag) begin  // 上板
    integer f=32'd5000_0000; // halfT=0.5s
    fdivider fdivider01(.clk(clk),.f(f),.myclk(myclk));
end else begin // 仿真
    assign myclk=clk;
end
```

第二步：对两个端口 busy 信号的判断，亦即对读写冲突的处理。分为操作地址相同或不同两种情况，默认设置 a 端口的优先级高于 b 端口。

```
always@(*)begin
    if(addr_a!=addr_b) begin       //地址不同时，无冲突
        busy_a=1;
        busy_b=1;
    end
    else begin         //地址相同，同时读写时，设置a端口优先于b端口
        if(we_a) begin
            busy_a=1;
            busy_b=0;
        end
        else if(we_b)begin
            busy_a=0;
            busy_b=1;
         end
       else begin
            busy_a=1;      //只要不是在写就为高电平
            busy_b=1;
            end
    end
 end
```

第三步：a 端口为同步读

```
//同步读端口
always@(posedge myclk or posedge rst)begin // clk上板用1Hz，要fdivider
   if(rst)begin
       reg_d_a<=0;
    end
   else begin
       if(!we_a && busy_a)begin
          reg_d_a<=mem[addr_a];
          end
        else
         begin
          reg_d_a<=reg_d_a;
        end
   end
end
```

第四步：b 端口为异步读

```
//异步读端口
 always@(*)begin
    if(rst)begin
        reg_d_b<=0;
     end
    if(!we_b && busy_b)begin
       reg_d_b<=mem[addr_b];
    end
    else begin
        reg_d_b<=reg_d_b;
    end
end
```

第五步：a,b 端口均为同步写

```
    //write declaration
    always@(posedge myclk or posedge rst) begin
       if(rst)
          for(i=0;i<DP;i=i+1)begin
                mem[i]=4'h0;
             end
       else begin
          if(we_a && busy_a)begin
             mem[addr_a]<=din_a;
           end
           if (we_b && busy_b) begin        //用else if 的话，两个都有效，但是不在同一个地址上
             mem[addr_b]<=din_b;
           end
        end
//        else begin
//           mem[addr_a]<=mem[addr_a];           //导致其无法修改的原因
//           mem[addr_b]<=mem[addr_b];
//         end
     end
```

第六步：数据输出，调用自定义数码管显示模块，可观察到 a、b 同步读与异步读的明显区别。

3、FIFO 的设计

第一步：自定义拨码（或按钮）时钟设计，包括上板和仿真两种。其中上板需要对 key 进行防抖，设置防抖时间 100ms(按钮 10ms 左右即可)，通过比较时钟上升沿记录的数据 last 和当前数据 key，相同则直接赋值给

clk_key,否则 clk_key=0。

```
    //------------------上板-----------------------
    wire myclk;
    reg clk_key;
    integer f = 32'd500_0000; // halfT=50ms
    fdivider fdivider01(.clk(clk),.f(f),.myclk(myclk));
    reg last;
    always@(posedge myclk) last <= key;
    always@(*) clk_key <= (key==last)?key:0;
     // ------------------仿真-----------------------
//     wire clk_key;
//     assign clk_key = clk;
```

第二步：控制读写指针移动

```
// raddr移动
always@(posedge clk_key,posedge rst) begin
    if(rst) raddr <= 1'd0;
    else if(ren && ~rempty) raddr <= raddr + 1'b1;
    else raddr <= raddr;
end

// waddr移动
always@(posedge clk_key,posedge rst) begin
    if(rst) waddr <= 1'd0;
    else if(wen && ~wfull) waddr <= waddr + 1'b1;
    else waddr <= waddr;
end
```

第三步：同步读和同步写

```
reg [DW-1:0]dout;  // 读
always@(posedge clk_key) begin
    if(ren && ~rempty) dout <= ram[raddr];
    else dout <= {DW{1'bz}};
end
always@(posedge clk_key,posedge rst) begin // 写
    if(rst) begin
        for(i=0;i<RD;i=i+1)
            ram[i] <= {DW{1'bz}}; // 按位宽取结果，比直接的8'hzz合理
    end
    else if(wen && ~wfull) ram[waddr]<=din;
    else ram[waddr] <= ram[waddr];
end
```

第四步：内置数据记录以及空满检测，并显示最后的结果于数码管上。

```
// 内置计数
always@(posedge clk_key,posedge rst) begin
    if(rst) cnt <= 1'd0;
    else if(wen && ~wfull && ren && ~rempty) cnt <= cnt; // 同时读写，避免cnt计算冲突
    else if(wen && ~wfull) cnt <= cnt + 1'd1;
    else if(ren && ~rempty) cnt <= cnt - 1'd1;
    else cnt <= cnt;
end
// 空满检测
assign wfull = (cnt==RD)?1'b1:1'b0;
assign rempty = (cnt==0)?1'b1:1'b0;
```

五、实验过程中遇到的问题及解决情况

问题一：时钟分频模块设计时，最开始的时钟取反使用输出的 myclk 直接取反，但是会导致 myclk 因为没有初始化，数据取反无效。 解决方案，使用内置时钟 inlineclk，记录时钟取反信息，最后通过 assign 模块直接赋值给 myclk 即可。

问题二：单端口 RAM 设计中，最开始想的实现同步异步模式是通过一个按钮来控制“同步读”、“异步读”两种模式的切换，分两个 always 判断获取的信息，但是会出现两个 always 块处理一个 data_out，网表实现会报[Synth 8-3352]错误，即 always 块作用同一个数发生。解决方案不唯一，自己是通过直接将同步、异步读到的数据同时显示在两个不同的数码管上，达到同步、异步两种读的效果。

问题三：关于 FIFO 内置计数 cnt 的位宽设计——
最开始是 DW-1，但是在仿真中发现读不到第 16 个数据，后来改为 DW，为 0 时什么也没有写，1~16 分别表示写入了多少个数据，共 17 种状态，故位宽不能用 DW-1。

问题四：按键防抖问题，最开始没有对按键进行防抖，发现 wfull 的 led 灯会提前亮，加入按键防抖后，显示正常。

问题五：双端口 RAM 对 RAM 的值的修改问题。在仿真过程中发现，对 RAM 已有值的地址无法进行修改其值。经检查发现，在写模块中对寄存器的值进行了不必要的判断赋值，导致生成了锁存器。删除该部分后仿真结果符合预期。

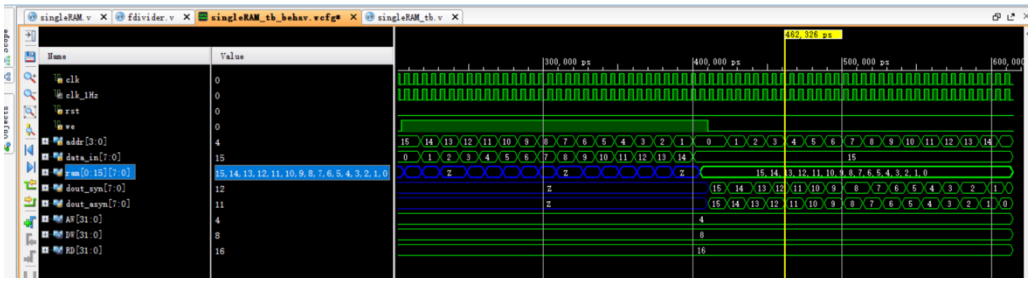
六、实验结果及分析和（或）源程序调试过程

1、单端口 ram

①仿真结果

为了方便仿真，重新设定了自定义时钟。

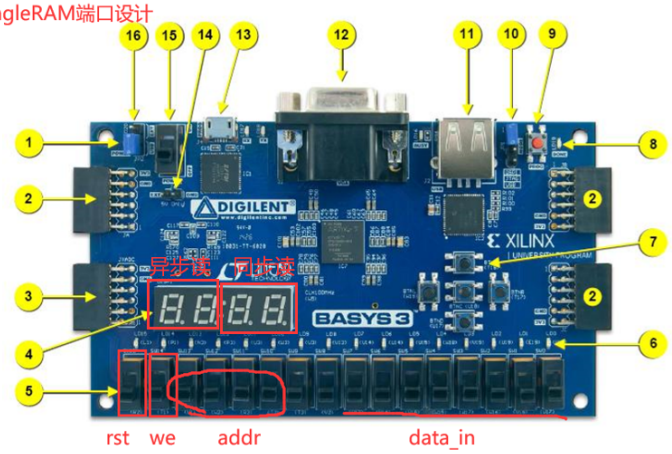
```
//------------------仿真-------------------------------
assign clk_1Hz = clk;
```



针对“一通读”、“一通写”进行仿真，结果符合预期。

②端口设计

singleRAM端口设计

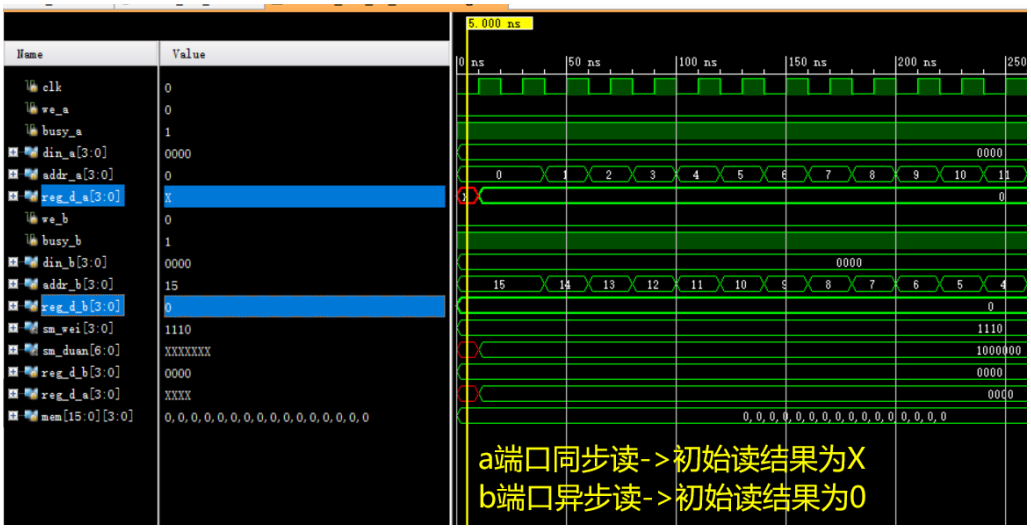


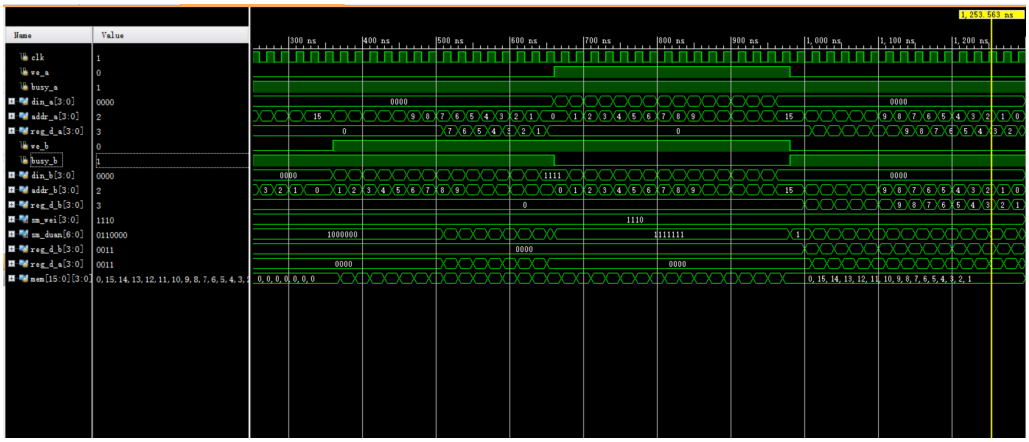
③上板结果

见附件视频。

2、双端口 Ram

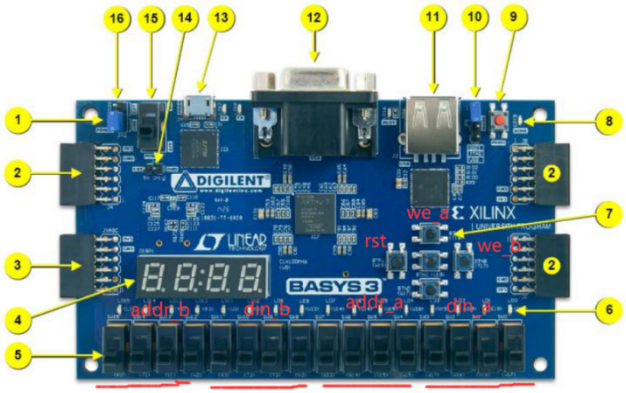
①仿真结果：





测试方案为：同步读；a 写，b 读；a、b 同时写，由仿真结果可见，结果符合预期。

②端口设计：

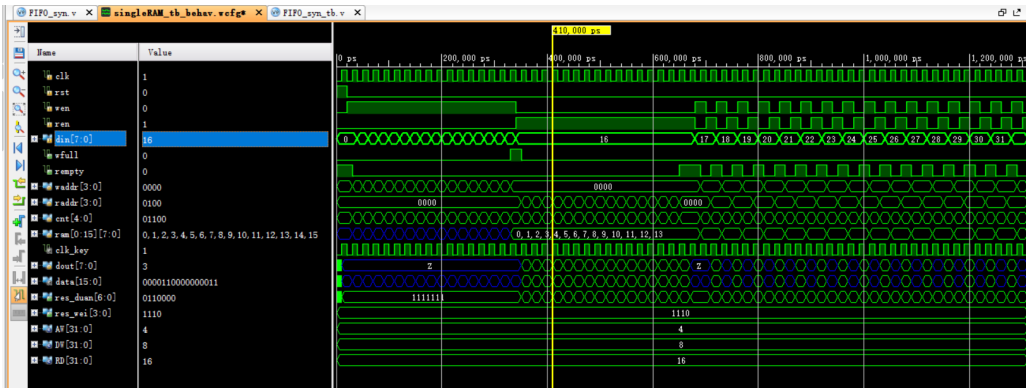


③上板结果：

见附件视频。

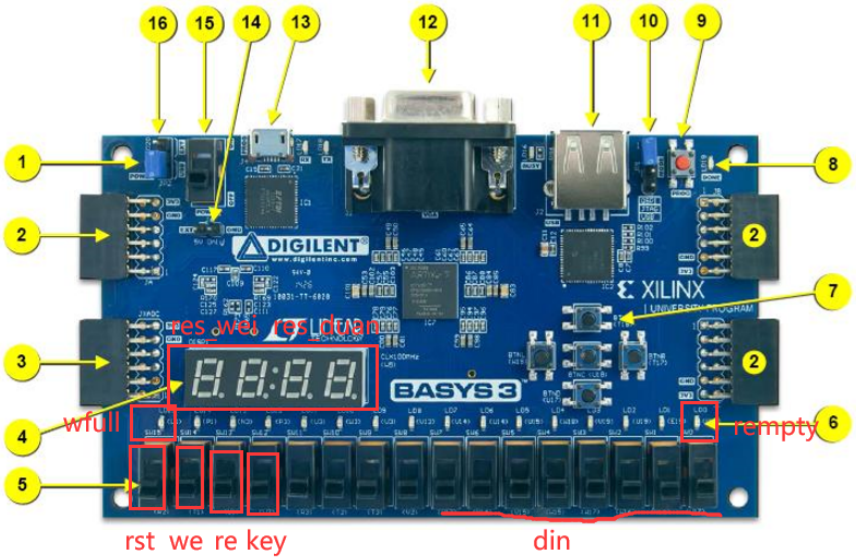
3、FIFO 设计

①仿真结果：



通过模拟“一通写”，“一通读”，“一写一读”，结果符合预期。

②端口设计：



③上板结果：

见附件视频。

七、小组分工情况说明

李燕琴：单端口 RAM 和 FIFO 的设计与实现

杨思怡：双口 RAM 的设计与实现