**《数字逻辑》实验报告**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **姓名** | | 黄婧婧，蔡欣彤 | | **年级** | | 2020级 |
| **学号** | | 20204237，20204187 | | **专业、班级** | | 计算机科学与技术02班，信息安全01班 |
| **实验名称** | 14.存储器 | | | | | |
| **实验时间** | **2021.12.1** | | **实验地点** | | **D1410** | |
| **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  掌握随机存储器原理，学会 FPGA 内部存储器控制器的设计方法。  掌握单端口与双端口 RAM（随机存储器）设计与实现。  掌握 FIFO（先入先出存储队列）设计与实现。 | | | | | | |
| 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）以及控制信号。  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。 | | | | | | |
| 三、实验设计  RAM（random access memory）又称“随机存储器”，存储单元的内容按需要随意取出或者存入，速度很快，但断电时将丢失数据，所以一般被作为临时数据的存储媒介。Basys3 开发板上拥有 1,800 Kbits 快速 RAM 块，可以根据需 求定制 ROM、RAM 或者 FIFO。  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，单端口 RAM 框图如下：    双端口（同步与异步）RAM，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）以及控制信号。双端口RAM 常用于视频/图像处理设计中。双端口 RAM 框图如下：    FIFO 是一个先入先出的存储队列，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。FIFO 的框图和信号功能如下： | | | | | | |
| 四、实验过程或算法  1、单端口 RAM 设计（同步读）  端口说明：  clk 控制时钟输入，data\_in 写使能时输入存储器里的数据，addr 存储数据的地址，data\_out 读使能时输出存储器里addr处的数据，we 控制读写使能，当we=1时，读使能，执行从存储器里读取数据到data\_out，反之，写使能，执行在存储器中addr处写入data\_in;  代码设计：  module RAM1  #(parameter DATA\_WIDTH=8,  ADDR\_WIDTH=4,  RAM\_DEPTH=1<<ADDR\_WIDTH)  (  input clk,  input [DATA\_WIDTH-1:0] data\_in,  input [ADDR\_WIDTH-1:0] addr,  input we,  output reg [DATA\_WIDTH-1:0] data\_out  );  reg [DATA\_WIDTH-1:0] mem[0:RAM\_DEPTH-1];    always@(posedge clk)  if(we) begin  mem[addr] <=data\_in;  end  else begin  data\_out<=mem[addr];  end  endmodule   1. 单端口 RAM 设计（异步读）   端口设计：  clk 控制时钟输入，data\_in 写使能时输入存储器里的数据，addr 存储数据的地址，data\_out 读使能时输出存储器里addr处的数据，we 控制读写使能，当we=1时，读使能，执行从存储器里读取数据到data\_out，反之，写使能，执行在存储器中addr处写入data\_in;  代码设计：  module RAM2  #(parameter DATA\_WIDTH=8,  ADDR\_WIDTH=4,  RAM\_DEPTH=1<<ADDR\_WIDTH)  (  input clk,  input [DATA\_WIDTH-1:0] data\_in,  input [ADDR\_WIDTH-1:0] addr,  input we,  output reg [DATA\_WIDTH-1:0] data\_out  );  reg [DATA\_WIDTH-1:0] mem[0:RAM\_DEPTH-1];    always@(posedge clk)  if(we) begin  mem[addr] =data\_in;  end    always@(we or addr)//在we的前提下，addr变化即触发  if(!we) begin  data\_out=mem[addr];  end    endmodule   1. 双端口RAM设计（同步读）   端口设计：  clk 控制时钟输入，wｅａ，ｗｅｂ控制读写使能，当weａ=1时，读使能，执行从存储器里读取数据到dｏｕt＿ａ，反之，写使能，执行在存储器中addr＿ａ处写入din＿ａ，同理，ｗｅｂ类似。din＿ａ为ｗｅａ写使能时输入存储器里的数据，din＿ｂ为ｗｅｂ写使能时输入存储器里的数据，adｄｒ＿ａ存储ａ数据的地址，adｄｒ＿ｂ存储ｂ数据的地址，dout＿ａ为！ｗｅａ读使能时输出存储器里addr＿ａ处的数据，dout＿ｂ为！ｗｅｂ读使能时输出存储器里addr＿ｂ处的数据。  当ａｄｄｒ＿ａ＝＝ａｄｄｒ＿ｂ时，即两个地址相同，当两者同时经行读使能，则无影响；当两者相异或同为写使能时，ａ的优先级将会高于ｂ。  代码设计：  module RAM3//同步读  #(parameter DATA\_WIDTH=4,  ADDR\_WIDTH=3,  RAM\_DEPTH=1<<ADDR\_WIDTH)  (  input clk,  input [DATA\_WIDTH-1:0] din\_a,  input [DATA\_WIDTH-1:0] din\_b,  input [ADDR\_WIDTH-1:0] addr\_a,  input [ADDR\_WIDTH-1:0] addr\_b,  input wea,  input web,  output reg [DATA\_WIDTH-1:0] dout\_a,  output reg [DATA\_WIDTH-1:0] dout\_b  );  reg [DATA\_WIDTH-1:0] mem [0:RAM\_DEPTH-1];    always@(posedge clk)  //同步写  if(addr\_a==addr\_b) begin  if(!wea && !web) begin//读  dout\_a=mem[addr\_a];  dout\_b=mem[addr\_b];  end  else if(wea) begin//a写  mem[addr\_a]=din\_a;  end  else begin//a读  dout\_a=mem[addr\_a];  end  end  else begin  if(wea)//a写  mem[addr\_a]=din\_a;  else//a读  dout\_a=mem[addr\_a];  if(web)//b写  mem[addr\_b]=din\_b;  else//b读  dout\_b=mem[addr\_b];  end  endmodule   1. 双端口RAM设计（异步读）   端口设计：  clk 控制时钟输入，wｅａ，ｗｅｂ控制读写使能，当weａ=1时，读使能，执行从存储器里读取数据到dｏｕt＿ａ，反之，写使能，执行在存储器中addr＿ａ处写入din＿ａ，同理，ｗｅｂ类似。din＿ａ为ｗｅａ写使能时输入存储器里的数据，din＿ｂ为ｗｅｂ写使能时输入存储器里的数据，adｄｒ＿ａ存储ａ数据的地址，adｄｒ＿ｂ存储ｂ数据的地址，dout＿ａ为！ｗｅａ读使能时输出存储器里addr＿ａ处的数据，dout＿ｂ为！ｗｅｂ读使能时输出存储器里addr＿ｂ处的数据。  　　当ａｄｄｒ＿ａ＝＝ａｄｄｒ＿ｂ时，即两个地址相同，当两者同时经行读使能，则无影响；当两者相异或同为写使能时，ａ的优先级将会高于ｂ。  代码设计：  module RAM4  #(parameter DATA\_WIDTH=4,  ADDR\_WIDTH=3,  RAM\_DEPTH=1<<ADDR\_WIDTH)  (  input clk,  input [DATA\_WIDTH-1:0] din\_a,  input [DATA\_WIDTH-1:0] din\_b,  input [ADDR\_WIDTH-1:0] addr\_a,  input [ADDR\_WIDTH-1:0] addr\_b,  input wea,  input web,  output reg [DATA\_WIDTH-1:0] dout\_a,  output reg [DATA\_WIDTH-1:0] dout\_b  );  reg [DATA\_WIDTH-1:0] mem [0:RAM\_DEPTH-1];    always@(posedge clk)//同步写；  if(addr\_a==addr\_b) begin  if(wea) begin//a写  mem[addr\_a]=din\_a;  end  end  else begin  if(wea)//a写  mem[addr\_a]=din\_a;  if(web)//b写  mem[addr\_b]=din\_b;  end    always@(wea or web or addr\_a or addr\_b)  if(addr\_a==addr\_b) begin  if(!wea && !web) begin//均为读  dout\_a=mem[addr\_a];  dout\_b=mem[addr\_b];  end  else if(!wea) begin//a读  dout\_a=mem[addr\_a];  end  else if(!web) //b读  dout\_b=dout\_b;  end  else begin  if(!wea)  dout\_a=mem[addr\_a];  else  dout\_a=dout\_a;  if(!web)  dout\_b=mem[addr\_b];  else  dout\_b=dout\_b;  end  endmodule   1. FIFO设计   代码顶层设计：  　　代码分为顶层及FIFO和按键消抖三个部分，FIFO实现主要功能，按键消抖实现对时钟脉冲的一个模拟。  端口设计：  clk控制时钟脉冲的输入，rst控制对输入数据的重置，din为输入的数据，wr\_en控制写使能，rd\_en控制读使能，key为按键的输入，dout为读出的数据，empty为显示队列是否为空的信号，full为显示队列是否为满的信号。  特殊情况说明：当写使能和读使能同时出现时，可实现边读边写。  代码设计：  顶层代码：  module FIFO\_top(  input clk,  input rst,  input [7:0] din,  input wr\_en,  input rd\_en,  input key,  output [7:0] dout,  output empty,  output full  );    wire key\_out;  Debkey　u0(.clk(clk),.reset(~rst),.key(key),.debkey(key\_out));  FIFO u1(din,wr\_en,rd\_en,key\_out,rst,dout,empty,full);  endmodule  FIFO代码：  module FIFO  #(parameter DATA\_WIDTH=8,  ADDR\_WIDTH=3,  DEPTH=1<<ADDR\_WIDTH)  (  input [DATA\_WIDTH-1:0] din,  input wr\_en,  input rd\_en,  input clk,  input rst,  output reg [DATA\_WIDTH-1:0] dout,  output empty,  output full  );  reg [DATA\_WIDTH-1:0] mem [0:DEPTH-1];  reg [ADDR\_WIDTH:0] cnt=0;  reg [ADDR\_WIDTH-1:0] front=0,rear=0;    always@(posedge clk, posedge rst)  begin  if(rst)  begin  dout=0;  front=0;  rear=0;  cnt=0;  end  else begin  if(rd\_en) begin //读操作  if(cnt!=0) begin //不为空的时候  dout=mem[front];  cnt=cnt-1;  front=(front+1) % DEPTH;//调整指针位置；  end  else  dout<=0;  end  if(wr\_en) begin//写操作  if(cnt!=DEPTH) begin //没满的时候;  mem[rear]<=din;  cnt=cnt+1;  rear=(rear+1) % DEPTH; //调整指针位置；  end  else  mem[rear]<=mem[rear];  end  end  end  assign empty=(cnt==0)? 1:0;  assign full=(cnt==DEPTH)? 1:0;  endmodule  按键消抖模块：  module debkey(clk,reset,key,debkey);  input clk;  input reset;  input key;  output debkey;  //---------------------------------------------------------------  //100Hz 分频  parameter T100Hz = 249999;  integer cnt\_100Hz;  reg clk\_100Hz;  always @(posedge clk or negedge reset)  if(!reset)  cnt\_100Hz <= 32'b0;  else begin  cnt\_100Hz <= cnt\_100Hz + 1'b1;  if(cnt\_100Hz == T100Hz) begin  cnt\_100Hz <= 32'b0;  clk\_100Hz <= ~clk\_100Hz;  end  end  //---------------------------------------------------------------  //去抖模块  reg key\_rrr,key\_rr,key\_r;  always @(posedge clk\_100Hz or negedge reset)  if(!reset) begin  key\_rrr <= 1'b1;  key\_rr <= 1'b1;  key\_r <= 1'b1;  end  else begin  key\_rrr <= key\_rr;  key\_rr <= key\_r;  key\_r <= key;  end  //---------------------------------------------------------------  assign debkey = key\_rrr & key\_rr & key\_r;  endmodule | | | | | | |
| 1. 实验过程中遇到的问题及解决情况   1.刚开始没有考虑地址相同时读写冲突问题，后面设计时以a的优先级更高。  2.存储器的实现一开始并不清楚，后面查了资料才知道用二维的寄存器实现。  3.仿真时考虑如何体现出同步读与异步读的区别。 | | | | | | |
| 1. 实验结果及分析和（或）源程序调试过程   1、2仿真代码相同  仿真代码：  module RAM2\_sim(  );  reg [3:0] addr;  reg we;  reg [7:0] data\_in;  reg clk;  wire [7:0] data\_out;    RAM2 u(clk,data\_in,addr,we,data\_out);    initial begin  clk=0;  addr=4'b0000;  data\_in=8'b0000\_0001;  we=0;    #100 begin  we=1'b1;  end  begin  addr=4'd2;data\_in=8'd5;  end  #100 begin  addr=4'd4;data\_in=8'd8;  end  #100 begin  addr=4'd6;data\_in=8'd13;  end    #100 begin  addr=4'd8;data\_in=8'd22;  end    #100 begin  we=1'b0;  addr=4'd2;  end    #100 begin  addr=4'd4;  end    #100 begin  addr=4'd6;  end    #100 begin  addr=4'd8;  end  end    always #10 clk=~clk;  endmodule   1. 单端口 RAM 设计（同步读）（黄线部分可以看出异步读与同步读的区别）      1. 单端口 RAM 设计（异步读）     3、4仿真文件相同  仿真代码：  module RAM4\_sim(  );  reg [2:0] addr\_a,addr\_b;  reg [3:0] din\_a,din\_b;  reg we\_a,we\_b,clk;  wire [3:0] dout\_a,dout\_b;    RAM4 u(clk,din\_a,din\_b,addr\_a,addr\_b,we\_a,we\_b,dout\_a,dout\_b);    initial begin    clk=0;  addr\_a=3'b000;  addr\_b=3'b000;  din\_a=4'b0000;  din\_b=4'b0000;  we\_a=0;  we\_b=0;    #20 begin  we\_a=1'b1;  we\_b=1'b1;  end  begin  addr\_a=3'd0;addr\_b=3'd1;  din\_a=4'd1;din\_b=4'd10;  end    #20 begin  addr\_a=3'd2;addr\_b=3'd3;  din\_a=4'd3;din\_b=4'd4;  end    #20 begin  addr\_a=3'd4;addr\_b=3'd5;  din\_a=4'd5;din\_b=4'd6;  end    #20 begin  addr\_a=3'd6;addr\_b=3'd7;  din\_a=4'd7;din\_b=4'd8;  end    #20 begin  we\_b=1'b0;  addr\_a=3'd4;addr\_b=3'd0;  din\_a=8'd5;  end    #20 begin  addr\_a=3'd5;addr\_b=3'd1;  din\_a=8'd6;  end    #20 begin  we\_a=0;  addr\_a=3'd6;addr\_b=3'd2;  din\_a=8'd7;  end    #20 begin  addr\_a=3'd6;addr\_b=3'd3;  din\_a=8'd7;  end    end    always#10 clk=~clk;  endmodule   1. 单端口 RAM 设计（同步读）      1. 双端口RAM设计（异步读）      1. FIFO设计（可实现边读边写的功能）   仿真代码：  module FIFO\_sim(  );  reg [7:0] a=8'b01010101;  reg clk=0;  reg wr=1;  reg rd=0;  reg reset=0;  wire [7:0] out;  wire empty,full;  always #10 clk=~clk;    FIFO u(a,wr,rd,clk,reset,out,empty,full);    initial begin  #10 a=8'b00000001;  #20 a=8'b00000011;  #20 a=8'b00000111;  #20 a=8'b00001111;  #20 a=8'b00011111;  #20 a=8'b00111111;  #100 begin wr=0;rd=1;end  end    endmodule    七、小组分工情况说明  黄婧婧负责构思设计，实现功能。  蔡欣彤负责仿真模拟，负责代码优化，撰写实验报告。 | | | | | | |
|  | | | | | | |
|  | | | | | | |