**《数字逻辑》实验报告**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **姓名** | |  | | **年级** | |  |
| **学号** | |  | | **专业、班级** | |  |
| **实验名称** | **实验十四 存储器实验** | | | | | |
| **实验时间** | **2019.12** | | **实验地点** | | **DS1410** | |
| **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  通过实验，掌握有单端口、多端口存储器和FIFO队列的设计方法。 | | | | | | |
| 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。 | | | | | | |
| 1. 实验设计   一、单端口RAM  **单口RAM（同步读，同步写）**  只有一个端口，一个地址，要么读，要么写。  同步读指的是在时钟信号上升沿进行读。  **单口RAM（异步读，同步写）**  异步读，这里指给出地址的时候就进行读，即时钟不同步  二、双端口RAM  **双口RAM（同步读，同步写）**   1. 概述    1. 双口RAM，这里实现的是真双口RAM，即每个端口有独立的地址，可以完全独立地进行读写。    2. 例如，同时在端口AB进行写入，或者同时在端口AB进行读取。应当避免两个读端口或者两个写端口同时操作同一个地址，RAM中并没有此种冲突解决电路，设计者应该避免这种冲突。    3. 在FPGA上实现的时候，由于无法在内置时钟的条件下并行输入多位的数据，因此使用按键，来加载信号，然后按照时钟的时序逻辑来处理同步和异步。    4. RTL电路图      1. 设计    1. 顶层模块集成了数码管显示和双端口RAM的顶层模块。数码管显示用于方便测试地址（避免出现同时操作同一地址的情况）。    2. 双口RAM的顶层模块集成了时钟分频模块，按键消抖模块以及双口RAM模块。由于下载到FPGA开发板上，需要输入数据，而数据有一定的宽度，因此需要同时加载n位宽度的数据。考虑使用按键来完成，而按键需要进行消抖以保障正常功能。    3. 双口RAM的核心模块实现了两个端口独立的读与写，但是并未对同时操作同一地址的情况作出限制，使用时候应避免这种情况。写操作在时钟上边沿到达时进行，根据写使能触发，而读操作也在时钟上边沿到达时，根据读使能触发。   双口RAM（异步读，同步写）   1. 概述    1. 实现时，与同步读的双口RAM相似。读写分离，读操作的触发条件不再是时钟信号，而是地址信号。在地址给出的时，减小读操作。   三、FIFO   1. 读写指针的设置   分别为读和写设置三个指针：  当前读位置的指针、下一个状态即将读的位置的指针和指向下一个读的地址的指针；  当前写位置的指针、下一个状态即将写的位置的指针和指向下一个写的地址的指针。   1. 读命令   在读命令下，如果队列不为空，将当前读指针的下一个指针赋值给读指针的下一个状态，同时将队列的满标志置0。然后判断读指针的下一个指针是否和写指针的值一样。一样的话，说明队列为空；否则不为空。   1. 写命令   在数据的上升沿的时候，有写使能信号，就将数据写入  在写命令下，如果队列不为满，将当前写指针的下一个指针赋值给读指针的下一个状态，同时将队列的空标志置0。然后判断写指针的下一个指针是否和读指针的值一样。一样的话，说明队列为满；否则不为满。   1. 状态跳转   复位信号有效(rst\_n = 0)，读/写指针都指向0地址。此时队列状态为空。  复位不有效，且在时钟的上升沿，读/写指针的值，队列空，满状态的值由下一状态决定。否则保持原状态。  当队列不满且输入写使能信号的时候，数据写入信号为1。  1OYL8%TE[[75LGE5F[5UFDS | | | | | | |
| 四、实验过程或算法  RAM项目总的文件:：    1. 单端RAM  ①同步时序（仿真用）  module single\_ram\_sy(      clk, address, we, data  );  parameter DATA\_WIDTH = 8;  parameter ADDRESS\_WIDTH = 8;  parameter RAM\_DEPTH = 1 << ADDRESS\_WIDTH;  input clk;  input [ADDRESS\_WIDTH-1:0] address;  input we;  inout [DATA\_WIDTH-1:0] data;  reg [DATA\_WIDTH-1:0] data\_out;  reg [DATA\_WIDTH-1:0] mem [0:RAM\_DEPTH-1];  assign data = (!we) ? data\_out : 8'dz;  always @(posedge clk) begin      if(we == 1) begin          mem[address] <= data;      end  end  always @(posedge clk) begin      if(we == 0) begin          data\_out <= mem[address];      end  end  endmodule // single\_ram\_sy  ②异步时序（仿真用）  module single\_ram\_asyn(      clk, address, we, data  );  parameter DATA\_WIDTH = 8;  parameter ADDRESS\_WIDTH = 8;  parameter RAM\_DEPTH = 1 << ADDRESS\_WIDTH;  input clk;  input [ADDRESS\_WIDTH-1:0] address;  input we;  inout [DATA\_WIDTH-1:0] data;  reg [DATA\_WIDTH-1:0] data\_out;  reg [DATA\_WIDTH-1:0] mem [0:RAM\_DEPTH-1];  assign data = (!we) ? data\_out : 8'dz;  // write  always @(posedge clk) begin      if(we == 1) begin          mem[address] <= data;      end  end  // read  always @(we or address ) begin      if(we == 0) begin          data\_out <= mem[address];      end  end  endmodule // single\_ram\_sy    2. 双端RAM  ①顶层模块（上开发板用，包含时钟分频，数码管显示等模块）  module smg(clk,data,sm\_wei,sm\_duan,rst);  input clk,rst;  input [15:0] data;  output [3:0] sm\_wei;  output [7:0] sm\_duan;  //----------------------------------------------------------  //分频：更新频率为100Hz  integer clk\_cnt;    //分频周期计数  reg clk\_100Hz;  always @(posedge clk)  if(rst == 1) begin  //重置数码管分频显示      clk\_cnt <= 1'b0;      clk\_100Hz <= 1'b0;  end  else if(clk\_cnt == 32'd25000)  begin      clk\_cnt <= 1'b0;  //半个周期结束，重新计数      clk\_100Hz <= ~clk\_100Hz;    //时钟翻转  end  else clk\_cnt <= clk\_cnt + 1'b1;  //分频周期计数+1，又过去一个时钟周期  //让数码管于每个分频了的时钟周期在4个位交替闪烁  //利用人眼视觉暂留，让人看到四个数码管显示的不同内容  reg [3:0] wei\_ctrl=4'b1110;  always @(posedge clk\_100Hz)      wei\_ctrl <= {wei\_ctrl[2:0],wei\_ctrl[3]};  //段控制：每一位的数码管的8个led灯的输入  reg [3:0]duan\_ctrl;  always @(wei\_ctrl)      case(wei\_ctrl)          4'b1110:duan\_ctrl=data[2:0];          4'b1101:duan\_ctrl=data[5:3];          4'b1011:duan\_ctrl=data[8:6];          4'b0111:duan\_ctrl=data[11:9];          default:duan\_ctrl=4'hf;        endcase  //----------------------------------------------------------  //解码模块：8个led灯的输入的解码，让led灯对应的位闪烁（低电平亮灯）  reg [7:0]duan;  always @(duan\_ctrl)      case(duan\_ctrl)          4'h0:duan=8'b1100\_0000;//0          4'h1:duan=8'b1111\_1001;//1          4'h2:duan=8'b1010\_0100;//2          4'h3:duan=8'b1011\_0000;//3          4'h4:duan=8'b1001\_1001;//4          4'h5:duan=8'b1001\_0010;//5          4'h6:duan=8'b1000\_0010;//6          4'h7:duan=8'b1111\_1000;//7          4'h8:duan=8'b1000\_0000;//8          4'h9:duan=8'b1001\_0000;//9          4'ha:duan=8'b1000\_1000;//a          4'hb:duan=8'b1000\_0011;//b          4'hc:duan=8'b1100\_0110;//c          4'hd:duan=8'b1010\_0001;//d          4'he:duan=8'b1000\_0110;//e          4'hf:duan=8'b1000\_1110;//f          default : duan = 8'b1100\_0000;//0      endcase  //----------------------------------------------------------  assign sm\_wei = wei\_ctrl;  assign sm\_duan = duan;  endmodule  module top\_smg(      clk, rst, cs,      din\_a\_0, addr\_a\_0, dout\_a, we\_a, oe\_a,      din\_b\_0, addr\_b\_0, dout\_b, we\_b, oe\_b,      rst\_clk,      load,      sm\_wei, sm\_duan  );  parameter DATA\_WIDTH = 3;  parameter ADDR\_WIDTH = 3;  parameter RAM\_DEPTH = DATA\_WIDTH;  input clk, rst, cs;  input [ADDR\_WIDTH-1:0] addr\_a\_0, addr\_b\_0;  input [DATA\_WIDTH-1:0]  din\_a\_0, din\_b\_0;  output [DATA\_WIDTH-1:0]  dout\_a, dout\_b;  input we\_a, oe\_a, we\_b, oe\_b;  input rst\_clk;  input load;  output [3:0] sm\_wei;    //位控制信号  output [7:0] sm\_duan;   //段控制信号  //----------------------------------------------------------  wire [11:0]data;    //显示在每一位的数据：每3位data对应每一位数码管  wire [3:0]sm\_wei;  wire [7:0]sm\_duan;  //----------------------------------------------------------  //拼接输入信号为12位的data，然后将其输入smg实例化的U1中，操控七段数码管发光  assign data[11:9] = addr\_a\_0;  assign data[8:6] = dout\_a;  assign data[5:3] = addr\_b\_0; //不需要的计算位，将其置为0  assign data[2:0] = dout\_b;  top\_double\_ram\_sy top\_ram(      clk, rst, cs,      din\_a\_0, addr\_a\_0, dout\_a, we\_a, oe\_a,      din\_b\_0, addr\_b\_0, dout\_b, we\_b, oe\_b,      rst\_clk,      load  );  smg U1 (.clk(clk),.data(data),.sm\_wei(sm\_wei),.sm\_duan(sm\_duan),.rst(rst));  endmodule  ②同步时序模块（开发板）  module fpga\_double\_ram\_sy(      clk, rst, cs,      din\_a, addr\_a, dout\_a, we\_a, oe\_a,      din\_b, addr\_b, dout\_b, we\_b, oe\_b,      load,      memout  );   output [2:0] memout;    parameter DATA\_WIDTH = 3;  parameter ADDR\_WIDTH = 3;  parameter RAM\_DEPTH = DATA\_WIDTH \* 10;  input clk, rst, cs;  input [DATA\_WIDTH-1:0] din\_a;  input [ADDR\_WIDTH-1:0] addr\_a;  input we\_a, oe\_a;  output reg [DATA\_WIDTH-1:0] dout\_a;  // output wire [DATA\_WIDTH-1:0] memout;  input [DATA\_WIDTH-1:0] din\_b;  input [ADDR\_WIDTH-1:0] addr\_b;  input we\_b, oe\_b;  output reg [DATA\_WIDTH-1:0] dout\_b;  input load;  integer i;  // write  reg [DATA\_WIDTH-1:0] mem[RAM\_DEPTH-1:0];  wire [RAM\_DEPTH-1:0] memout\_b;  assign memout\_b = mem[addr\_b];  assign memout = mem[addr\_a];  // write  always@(posedge load) begin      if(!cs) begin          if(we\_a) begin              mem[addr\_a] <= din\_a;          end          if(we\_b) begin              mem[addr\_b] <= din\_b;          end      end      else begin          for(i = 0; i < RAM\_DEPTH; i = i + 1)              mem[i] <= 0;      end      end        // read      always@(posedge load) begin          if(oe\_a) begin              dout\_a <= mem[addr\_a];          end          if(oe\_b) begin              dout\_b <= mem[addr\_b];          end      end  Endmodule  ③按键消抖模块（上开发板用）  module debounce(      input clk,      input cs,      input rst,      input rst\_clk,      input load,      output cs\_o,      output rst\_o,      output rst\_clk\_o,      output load\_o      );      reg cs\_rrr, cs\_rr, cs\_r;      reg rst\_rrr, rst\_rr, rst\_r;      reg rst\_clk\_rrr, rst\_clk\_rr, rst\_clk\_r;      reg load\_rrr, load\_rr, load\_r;      always @(posedge clk)      begin          cs\_rrr = cs\_rr;          cs\_rr = cs\_r;          cs\_r = cs;            rst\_rrr = rst\_rr;          rst\_rr = rst\_r;          rst\_r = rst;            rst\_clk\_rrr = rst\_clk\_rr;          rst\_clk\_rr = rst\_clk\_r;          rst\_clk\_r = rst\_clk;            load\_rrr = load\_rr;          load\_rr = load\_r;          load\_r = load;      end      assign cs\_o = cs\_rrr && cs\_rr && cs\_r;      assign rst\_o = rst\_rrr && rst\_rr && rst\_r;      assign rst\_clk\_o = rst\_clk\_rrr && rst\_clk\_rr & rst\_clk\_r;      assign load\_o = load\_rrr && load\_rr && load\_r;  endmodule  ④时钟分频模块（仿真用）  module div\_clock(  input clk,  input rst\_clk,  output div\_clk\_w      );  reg div\_clk;  assign div\_clk\_w = div\_clk;  reg [21:0] cnt;  always@(posedge clk or negedge rst\_clk) begin      if(rst\_clk == 1) begin          cnt = 0;          div\_clk = 0;      end      else if(cnt == 22'd50000) begin          div\_clk = ~div\_clk;          cnt = 0;      end      else begin          cnt = cnt + 1;      end  end  endmodule  ⑤同步时序模块（上开发板用）  module top\_double\_ram\_sy(  clk, rst, cs,  din\_a\_0, addr\_a\_0, dout\_a, we\_a, oe\_a,  din\_b\_0, addr\_b\_0, dout\_b, we\_b, oe\_b,  rst\_clk,  load      );    parameter DATA\_WIDTH = 3;  parameter ADDR\_WIDTH = 3;  parameter RAM\_DEPTH = DATA\_WIDTH;  input clk, rst, cs;  input [ADDR\_WIDTH-1:0] addr\_a\_0, addr\_b\_0;  input [DATA\_WIDTH-1:0]  din\_a\_0, din\_b\_0;  output [DATA\_WIDTH-1:0]  dout\_a, dout\_b;  input we\_a, oe\_a, we\_b, oe\_b;  input rst\_clk;  input load;  wire rst\_o, cs\_o, rst\_clk\_o, load\_o; // 经过消抖后的  wire clk\_div;  div\_clock my\_div\_clk(clk, rst\_clk, clk\_div);  debounce mydebounce(clk, cs, rst, rst\_clk, load, cs\_o, rst\_o, rst\_clk\_o, load\_o);  fpga\_double\_ram\_sy my\_double\_ram\_sy(      clk, rst\_o, cs\_o,      din\_a\_0, addr\_a\_0, dout\_a, we\_a, oe\_a,      din\_b\_0, addr\_b\_0, dout\_b, we\_b, oe\_b,      load\_o  );  endmodulemodule top\_double\_ram\_sy(  clk, rst, cs,  din\_a\_0, addr\_a\_0, dout\_a, we\_a, oe\_a,  din\_b\_0, addr\_b\_0, dout\_b, we\_b, oe\_b,  rst\_clk,  load      );    parameter DATA\_WIDTH = 3;  parameter ADDR\_WIDTH = 3;  parameter RAM\_DEPTH = DATA\_WIDTH;  input clk, rst, cs;  input [ADDR\_WIDTH-1:0] addr\_a\_0, addr\_b\_0;  input [DATA\_WIDTH-1:0]  din\_a\_0, din\_b\_0;  output [DATA\_WIDTH-1:0]  dout\_a, dout\_b;  input we\_a, oe\_a, we\_b, oe\_b;  input rst\_clk;  input load;  wire rst\_o, cs\_o, rst\_clk\_o, load\_o; // 经过消抖后的  wire clk\_div;  div\_clock my\_div\_clk(clk, rst\_clk, clk\_div);  debounce mydebounce(clk, cs, rst, rst\_clk, load, cs\_o, rst\_o, rst\_clk\_o, load\_o);  fpga\_double\_ram\_sy my\_double\_ram\_sy(      clk, rst\_o, cs\_o,      din\_a\_0, addr\_a\_0, dout\_a, we\_a, oe\_a,      din\_b\_0, addr\_b\_0, dout\_b, we\_b, oe\_b,      load\_o  );  endmodule  ⑥异步时序模块（仿真用）  module double\_ram\_asyn(      clk, rst, cs,      din\_a, addr\_a, dout\_a, we\_a, oe\_a,      din\_b, addr\_b, dout\_b, we\_b, oe\_b  );    parameter DATA\_WIDTH = 8;  parameter ADDR\_WIDTH = 16;  parameter RAM\_DEPTH = DATA\_WIDTH \* 10;  input clk, rst, cs;  input [DATA\_WIDTH-1:0] din\_a;  input [ADDR\_WIDTH-1:0] addr\_a;  input we\_a, oe\_a;  output reg [DATA\_WIDTH-1:0] dout\_a;  // output wire [DATA\_WIDTH-1:0] memout;  input [DATA\_WIDTH-1:0] din\_b;  input [ADDR\_WIDTH-1:0] addr\_b;  input we\_b, oe\_b;  output reg [DATA\_WIDTH-1:0] dout\_b;  integer i;  reg [DATA\_WIDTH-1:0] mem[RAM\_DEPTH-1:0];  // write  always@(posedge clk or negedge rst) begin      if(!rst) begin          if(we\_a) begin              mem[addr\_a] <= din\_a;          end          if(we\_b) begin              mem[addr\_b] <= din\_b;          end      end      else begin          for(i = 0; i < RAM\_DEPTH; i = i + 1)              mem[i] <= 0;      end      end        // read      always@(addr\_a or addr\_b or oe\_a or oe\_b) begin          if(oe\_a) begin              dout\_a <= mem[addr\_a];          end          if(oe\_b) begin              dout\_b <= mem[addr\_b];          end      end  endmodule  ⑦同步时序模块（仿真用）  module double\_ram\_sy(      clk, rst, cs,      din\_a, addr\_a, dout\_a, we\_a, oe\_a,      din\_b, addr\_b, dout\_b, we\_b, oe\_b,      load  );    parameter DATA\_WIDTH = 8;  parameter ADDR\_WIDTH = 16;  parameter RAM\_DEPTH = ADDR\_WIDTH \* 10;  input clk, rst, cs;  input [DATA\_WIDTH-1:0] din\_a;  input [ADDR\_WIDTH-1:0] addr\_a;  input we\_a, oe\_a;  output reg [DATA\_WIDTH-1:0] dout\_a;  // output wire [DATA\_WIDTH-1:0] memout;  input [DATA\_WIDTH-1:0] din\_b;  input [ADDR\_WIDTH-1:0] addr\_b;  input we\_b, oe\_b;  output reg [DATA\_WIDTH-1:0] dout\_b;  input load;  integer i;  // write  reg [DATA\_WIDTH-1:0] mem[RAM\_DEPTH-1:0];  // write  always@(posedge clk or negedge rst) begin      if(!rst) begin          if(we\_a && !cs) begin              mem[addr\_a] <= din\_a;          end          if(we\_b && !cs) begin              mem[addr\_b] <= din\_b;          end      end      else begin          for(i = 0; i < RAM\_DEPTH; i = i + 1)              mem[i] <= 0;      end      end        // read      always@(posedge clk) begin          if(oe\_a) begin              dout\_a <= mem[addr\_a];          end          if(oe\_b) begin              dout\_b <= mem[addr\_b];          end      end  endmodule  ⑧异步时序模块（上开发板用）  module fpga\_double\_ram\_asyn(      clk, rst, cs,      din\_a, addr\_a, dout\_a, we\_a, oe\_a,      din\_b, addr\_b, dout\_b, we\_b, oe\_b  );    parameter DATA\_WIDTH = 8;  parameter ADDR\_WIDTH = 16;  parameter RAM\_DEPTH = DATA\_WIDTH \* 10;  input clk, rst, cs;  input [DATA\_WIDTH-1:0] din\_a;  input [ADDR\_WIDTH-1:0] addr\_a;  input we\_a, oe\_a;  output reg [DATA\_WIDTH-1:0] dout\_a;  // output wire [DATA\_WIDTH-1:0] memout;  input [DATA\_WIDTH-1:0] din\_b;  input [ADDR\_WIDTH-1:0] addr\_b;  input we\_b, oe\_b;  output reg [DATA\_WIDTH-1:0] dout\_b;  integer i;  reg [DATA\_WIDTH-1:0] mem[RAM\_DEPTH-1:0];  // write  always@(posedge clk or negedge rst) begin      if(!rst) begin    3. FIFO  ①顶层模块  module fifo\_cus  #(  parameter N = 8, //数据宽度  parameter M = 4 //fifo的地址宽度  )  //对队列的参数设置。便于以后代码的移植。  //如果以后要实现数据宽度为16，深度为2^8的FIFO。只需改N =16 M=8即可  (      input clko, //输入时钟      input clk, //系统时钟，用于消抖      input reset, //消抖复位信号      input rst\_n, //输入复位信号      input wr, //输入写使能      input[N-1:0] w\_data, //输入输入      input rd, //输入读使能      output empty, //输出fifo空标志      output full, //输出fifo满标志      output[N-1:0] r\_data //输出读取的数据  );      reg [N-1:0] array\_reg [5\*M - 1:0];  //寄存器组，用来充当FIFO队列      reg [M-1:0] w\_ptr\_reg, w\_ptr\_next,w\_ptr\_succ;    //定义写指针，指示当前写的位置，下一个状态写的位置，写位置的下一个位置      reg [M-1:0] r\_ptr\_reg, r\_ptr\_next,r\_ptr\_succ;    //定义读指针，指示当前读的位置，下一个状态读的位置，读位置的下一个位置      //定义FIFO满和空的信号      reg full\_reg, full\_next;      reg empty\_reg, empty\_next;      wire wr\_en; //数据写入信号      wire clkod; //消抖后的按键信号      //数据的写入，在数据的上升沿的时候，有写使能信号，就将数据写入      always@( posedge clkod ) begin          if( wr\_en )              array\_reg[w\_ptr\_reg] <= w\_data;          else              array\_reg[w\_ptr\_reg] <= array\_reg[w\_ptr\_reg];      end      /\*状态跳转      在复位信号有效(rst\_n = 0)，读/写指针都指向0地址。此时队列状态为空。      在复位不有效，且在时钟的上升沿，读/写指针的值，队列空，满状态的值由下一状态决定。否则保持 \*/      always@( posedge clkod ) begin          if( !rst\_n )    begin              w\_ptr\_reg <= 0;              r\_ptr\_reg <= 0;              full\_reg <= 1'b0;              empty\_reg <= 1'b1;  end          else    begin              w\_ptr\_reg <= w\_ptr\_next;              r\_ptr\_reg <= r\_ptr\_next;              full\_reg <= full\_next;              empty\_reg <= empty\_next;    end      end      //下一个状态的判定      always@( \* )    begin          w\_ptr\_next = w\_ptr\_reg;          r\_ptr\_next = r\_ptr\_reg;          full\_next = full\_reg;          empty\_next = empty;          w\_ptr\_succ = w\_ptr\_reg + 1'b1;          r\_ptr\_succ = r\_ptr\_reg + 1'b1;          case( {wr,rd} )          /\*读命令：在读命令下，如果队列不为空，讲当前读指针的下一个指针赋值给读指针的下一个状态，同时将队列的满标志置0。          然后判断读指针的下一个指针是否和写指针的值一样。一样的话，说明，队列为空。否则不为空。 \*/          2'b01:          begin              if( ~empty\_reg )     begin                  r\_ptr\_next = r\_ptr\_succ;                  full\_next = 0;              if( r\_ptr\_succ == w\_ptr\_reg )                  empty\_next = 1'b1;              else                  empty\_next = 1'b0;  end          end          /\*写命令：在写命令下，如果队列不为满，将当前写指针的下一个指针赋值给读指针的下一个状态，同时将队列的空标志置0。          然后判断写指针的下一个指针是否和读指针的值一样。一样的话，说明，队列为满。否则不为满。          \*/          2'b10:          begin              if( ~full\_reg ) begin                  w\_ptr\_next = w\_ptr\_succ;                  empty\_next= 0;              if( w\_ptr\_succ == r\_ptr\_reg )                  full\_next = 1'b1;              else    full\_next = 1'b0;              end          end          /\*读写命令：在读写命令下， 直接改变对应指针的下一个状态值。\*/          2'b11:          begin              if( ~full\_reg && ~empty\_reg )   begin                  w\_ptr\_next = w\_ptr\_succ;                  r\_ptr\_next = r\_ptr\_succ;    end              //在满的状态，不允许写              else if( full\_reg ) begin                  r\_ptr\_next = r\_ptr\_succ;                  full\_next = 0;  end              //在空的状态，不允许写              else if( empty\_reg )    begin                  w\_ptr\_next = w\_ptr\_succ;                  empty\_next = 0; end          end          endcase      end       assign r\_data = array\_reg[r\_ptr\_reg];   // 数据的读取。数据读取一直在进行，不过读取的是之前的值。       assign wr\_en = wr & ~full\_reg;  //当队列不满且输入写使能信号的时候，数据写入信号为1      // 满/空输出信号的赋值      assign full = full\_reg;      assign empty = empty\_reg;      debounce mydebouncer(clk,clko,reset,clkod);  endmodule  ②按键消抖模块  module debounce(      input clk,      input in,      input reset,      output out      );      reg key\_rrr,key\_rr,key\_r;            always @(posedge clk or negedge reset)              if(!reset)                  begin                      key\_rrr <=0;                      key\_rr <=0;                      key\_r <=0;                  end              else                  begin                      key\_rrr <= key\_rr;                      key\_rr <= key\_r;                      key\_r <= in;                  end          assign out = key\_rrr & key\_rr & key\_r;  endmodule  module debounce(      input clk,      input in,      input reset,      output out      );      reg key\_rrr,key\_rr,key\_r;            always @(posedge clk or negedge reset)              if(!reset)                  begin                      key\_rrr <=0;                      key\_rr <=0;                      key\_r <=0;                  end              else                  begin                      key\_rrr <= key\_rr;                      key\_rr <= key\_r;                      key\_r <= in;                  end          assign out = key\_rrr & key\_rr & key\_r;  endmodule | | | | | | |
| 1. 实验过程中遇到的问题及解决情况 2. 发现同步时序和异步时序的写法无法在开发板上体现。   解决：  检查后发现，时钟即使分频了依旧很快，在开发板上无法体现是必然现象。   1. 在双端口异步读逻辑的实现时，由于写操作使用了系统时钟clk，而读操作不适用时钟，在实现的时候出现了错误。   解决：  通过查阅资料,也就是说连接到一个同步电路的时钟输入的信号不是同步的，此时需要设置set\_property CLOCK\_DEDICATED\_ROUTE FALS。  set\_property CLOCK\_DEDICATED\_ROUTE FALSE [get\_nets load\_IBUF]  set\_property CLOCK\_DEDICATED\_ROUTE FALSE [get\_nets rst\_IBUF]  set\_property CLOCK\_DEDICATED\_ROUTE FALSE [get\_nets rst\_clk\_IBUF]  set\_property CLOCK\_DEDICATED\_ROUTE FALSE [get\_nets oe\_a\_IBUF]  set\_property CLOCK\_DEDICATED\_ROUTE FALSE [get\_nets oe\_b\_IBUF]  set\_property CLOCK\_DEDICATED\_ROUTE FALSE [get\_nets addr\_a\_IBUF]  set\_property CLOCK\_DEDICATED\_ROUTE FALSE [get\_nets addr\_b\_IBUF]  https://img-blog.csdnimg.cn/20190421161907581.png?x-oss-process=image/watermark,type_ZmFuZ3poZW5naGVpdGk,shadow_10,text_aHR0cHM6Ly9ibG9nLmNzZG4ubmV0L3dlaXhpbl80MzE2MjgxOA==,size_16,color_FFFFFF,t_70 | | | | | | |
| 六、实验结果及分析和（或）源程序调试过程  1. 仿真：  （一）单端口RAM  ①同步时序  在时钟的下降沿给出地址值（从0-255），同时给出要写入的数据（1-256）。    写入操作后，进行读测试。可以看到，在给定地址上读取的值正是之前写入的值，且落后于地址给出的时间，与时钟的上升沿一致。这正是同步读的体现。    module   sim\_single\_ram\_sy(  );  reg clk;  reg [7:0] address;  wire [7:0] data;  reg [7:0] dataIn;  // wire [7:0] data\_out;  // wire [7:0] mem;  reg we;  assign data = (we == 1) ? dataIn : 8'dz;  initial begin      clk = 0;      address = 0;      dataIn = 0;      we = 0;  end  always #5 clk = ~clk;  integer i;  initial begin      // write      #100 we = 1;      for (i = 0; i < 256; i = i + 1) begin         @(negedge clk) begin             address = i;             dataIn = dataIn + 1;         end      end      #100 we = 0;      for (i = 0; i < 256; i = i + 1) begin          @(negedge clk) begin              address = i;          end      end  end  single\_ram\_sy my\_single\_ram\_sy(.clk(clk), .address(address), .we(we), .data(data));  endmodule //  ②异步时序  写入仍然是同步逻辑，而读为异步逻辑，可以看到，数据在时钟的下降沿读出，与地址信号同步（这是由于在仿真时，地址信号在时钟下降给出）。    检查时，问及为什么后面的数据为0（如图）    这是由于地址ff写入的数据就是0    module sim\_single\_ram\_asyn(  );  reg clk;  reg [7:0] address;  wire [7:0] data;  reg [7:0] dataIn;  // wire [7:0] data\_out;  // wire [7:0] mem;  reg we;  assign data = (we == 1) ? dataIn : 8'dz;  initial begin      clk = 0;      address = 0;      dataIn = 0;      we = 0;  end  always #5 clk = ~clk;  integer i;  initial begin      // write      #100 we = 1;      for (i = 0; i < 256; i = i + 1) begin         @(negedge clk) begin             address = i;             dataIn = dataIn + 1;         end      end      #100 we = 0;      for (i = 0; i < 256; i = i + 1) begin          @(negedge clk) begin              address = i;          end      end  end  single\_ram\_sy my\_single\_ram\_sy(.clk(clk), .address(address), .we(we), .data(data));  endmodule //  （二）双端口RAM  ①时钟分频（按键消抖不在此赘述）  module sim\_div\_clock(      );    reg clk, cs, rst;  reg [2:0] din\_a;  reg [2:0] addr\_a;  reg we\_a, oe\_a;  wire [2:0] dout\_a;  reg [2:0] din\_b;  reg [2:0] addr\_b;  reg we\_b, oe\_b;  wire [2:0] dout\_b;  wire [2:0] memout;  reg clk\_rst;  reg load;      initial begin          clk  = 0;          clk\_rst = 0;          #5 clk\_rst = 1;          #5 clk\_rst = 0;      end        always #5 clk = ~clk;      // top\_div\_test test\_div(clk, clk\_rst);      top\_double\_ram\_sy test(      clk, rst, cs,      din\_a, addr\_a, dout\_a, we\_a, oe\_a,      din\_b, addr\_b, dout\_b, we\_b, oe\_b,      clk\_rst,      load      );  endmodule  ②异步时序  测试两个端口的读写，首先是A写B读，写入1，2,3,4，5,6,7，可以看到，此时AB的读结果均为X，表示没有读到。  而后A读B写，可见A成功读到了写入的数据，且与地址是信号同步，在时钟的下降沿输出，为同步读逻辑。而B写入数据后，在之后的B读操作中，得到了正确的结果。    module sim\_double\_ram\_asyn;  reg clk, cs, rst;  reg [7:0] din\_a;  reg [15:0] addr\_a;  reg we\_a, oe\_a;  wire [7:0] dout\_a;  reg [7:0] din\_b;  reg [15:0] addr\_b;  reg we\_b, oe\_b;  wire [7:0] dout\_b;  integer i = 1;  // a写b读  always #5 clk = ~clk;  initial begin      clk = 0;      cs = 1;      rst = 0;      din\_a = 0;      addr\_a = 0;      we\_a = 0;      oe\_a = 0;        din\_b = 0;      addr\_b = 0;      we\_b = 0;      oe\_b = 0;  #50 @(posedge clk) begin      cs = 0;      din\_a = 0;      din\_b = 0;      we\_a = 1'b1;      oe\_b = 1'b1;      we\_b = 1'b0;      oe\_a = 1'b0;  end  for(i = 0; i < 8; i = i + 1) begin      @(negedge clk) begin          addr\_a = i;          din\_a = din\_a + 1;          addr\_b = i+8;      end  end  // a读b写  #50 @(posedge clk) begin      din\_a = 0;      din\_b = 0;      we\_a = 1'b0;      oe\_b = 1'b0;      we\_b = 1'b1;      oe\_a = 1'b1;  end  for(i = 0; i < 8; i = i + 1) begin      @(negedge clk) begin          addr\_a = i;          din\_b = din\_b + 1;          addr\_b = i+8;      end  end  // a读b读  #50 @(posedge clk) begin      din\_a = 0;      din\_b = 0;      we\_a = 1'b0;      we\_b = 1'b0;      oe\_a = 1'b1;      oe\_b = 1'b1;  end  for(i = 0; i < 8; i = i + 1) begin      @(negedge clk) begin          addr\_a = i;          addr\_b = i + 8;      end  end  // a写b写  #50 @(posedge clk) begin      din\_a = 0;      din\_b = 0;      we\_a = 1'b1;      we\_b = 1'b1;      oe\_a = 1'b0;      oe\_b = 1'b0;  end  for(i = 0; i < 8; i = i + 1) begin      @(negedge clk) begin          addr\_a = i;          addr\_b = i + 8;          din\_a = din\_a + 1;          din\_b = din\_b + 2;      end  end  // a读b读  #50 @(posedge clk) begin      din\_a = 0;      din\_b = 0;      we\_a = 1'b0;      we\_b = 1'b0;      oe\_a = 1'b1;      oe\_b = 1'b1;  end  for(i = 0; i < 8; i = i + 1) begin      @(negedge clk) begin          addr\_a = i;          addr\_b = i + 8;      end  end  #50 @(posedge clk) begin      cs = 1;  end  end  double\_ram\_asyn my\_ram(      clk, rst, cs,      din\_a, addr\_a, dout\_a, we\_a, oe\_a,      din\_b, addr\_b, dout\_b, we\_b, oe\_b  );  endmodule // sim\_double\_ram  ③同步时序  在地址0,1,2上分别写入数据1,2，3，然后进行读操作，给出地址分别为2,1，0，可以看到读出的数据为3,2,1，为正确的。并且数据在时钟的上升沿给出。    这是时钟的信息：    （三）FIFO  module fifo\_sim;      parameter N = 8; //数据宽度      parameter M = 4; //fifo的地址宽度      reg clko; //输入时钟      reg clk; //系统时钟，用于消抖      reg reset; //消抖复位信号      reg rst\_n; //输入复位信号      reg wr; //输入写使能      reg[N-1:0] w\_data; //输入输入      reg rd; //输入读使能      wire empty; //输出fifo空标志      wire full; //输出fifo满标志      wire[N-1:0] r\_data; //输出读取的数据        initial begin          clk = 0;          clko = 0;      end        initial begin          reset = 1; rst\_n = 1; wr = 0; rd = 0;          #10 reset = 0; rst\_n = 0; wr = 1;          #100 rd = 1;      end        initial begin          #10 w\_data = 8'b11010101;          #500 w\_data = 8'b00010001;          #1000 w\_data = 8'b01011001;          #1500 w\_data = 8'b00000000;  end        always begin          #2 clk = ~clk;      end        always begin          #10 clko = ~clko;      end  fifo\_cus f(       clko, //输入时钟       clk, //系统时钟，用于消抖       reset, //消抖复位信号       rst\_n, //输入复位信号       wr, //输入写使能       w\_data, //输入输入       rd, //输入读使能       empty, //输出fifo空标志       full, //输出fifo满标志       r\_data //输出读取的数据  );  endmodule  2. 开发板：  FIFO测试：   1. 往FIFO内写入元素      1. 连续写入多次，发现最右边的LED灯亮起，说明FIFO已经达到满状态      1. 连续读出FIFO中的元素，发现最右边的LED灯熄灭，说明FIFO已经不再满，且发现读出的元素和写入的元素一一对应，全部相等，且先写入的元素先读出。      1. 连续读出，直到右数第二个LED灯亮起，说明FIFO中的元素已经全部读出，此时FIFO处于空状态。     RAM测试  （1）    （2）    （3）    通过充分测试可知本次实验设计正确，功能正确。  七、小组分工情况说明  蔡嘉轩、田润泽：负责实验结果部分（开发板实现、操作演示等）、实验设计部分（包括原理图、电路图、代码等）。  尹宇慧：负责协同小组其他成员调试程序，完善、补充相应的部分  姚语涵：负责实验结果部分（编写仿真代码等），撰写报告 | | | | | | |