**《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** | **2023.12.05** | | **实验地点** | | **DS3402** | |
| **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  **单端口RAM设计原理：**  单端口RAM设计（带异步读和同步读两种模式），在时钟（clk）上沿采集地址（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。  **单端口同步RAM项目设计：**  顶层模块：    divider：为七段数码管提供分频后的clk；  Syn\_SinglePortRAM：单端口同步RAM；  transformer：利用加三移位法转二进制为BCD码，输入至七段数码管进行显示功能的实现；  display7seg：七段数码管显示模块；  Schematic如下：    **单端口异步RAM项目设计：**  顶层模块：    Asy\_SinglePortRAM：单端口异步RAM；  其余模块功能同上；  Schematic：    **双端口RAM设计原理：**  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。  **双端口同步RAM：**  顶层模块：    Syn\_DoublePortRAM：双端口同步RAM；  其余模块功能同上  Schematic：    **双端口异步RAM：**  顶层模块：    Asy\_DoublePortRAM：双端口同步RAM；  其余模块功能同上  schematic：    **FIFO设计原理：**  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。  **顶层模块：**    debkey：消抖模块；  FIFO：实现FIFO；  Schematic： | | | | | | |
| 四、实验过程或算法  **单端口同步RAM**  顶层模块  top\_Syn\_SinglePortRAM.v  module top\_Syn\_SinglePortRAM#(parameter DATA\_WIDTH = 4,parameter ADDR\_DEPTH = 4)(  input clk,rst,  input [ADDR\_DEPTH-1:0]addr,  input [DATA\_WIDTH-1:0]data\_in,  input we,  output wire[3:0]an,  output wire[6:0]display  );  wire[DATA\_WIDTH-1:0]data\_out;  wire clk\_div;  reg [25:0]target = 50000;  wire [15:0]BCD;  //divider  divider d(.clk(clk),.rst(rst),.target(target),.clk\_div(clk\_div));  //Syn\_SinglePortRAM  Syn\_SinglePortRAM S(.clk(clk),.rst(rst),.addr(addr),.data\_in(data\_in),.we(we),.data\_out(data\_out));  //transformer  transformer t(.data(data\_out),.BCD(BCD));  //display7seg  display7seg dis(.clk(clk\_div),.data3(BCD[15:12]),.data2(BCD[11:8]),.data1(BCD[7:4]),.data0(BCD[3:0]),.an(an),.display(display));  endmodule  分频模块  divider.v  module divider(  input clk,rst,  input [15:0] target,  output reg clk\_div  );  reg [15:0] counter;  always @(posedge clk) begin  if(rst)  begin  counter <= 0;  clk\_div <= 0;  end  else if(counter==target)  begin  counter <= 0;  clk\_div <= ~clk\_div;  end  else  counter <= counter+1;  end  endmodule  单端口同步设计模块  Syn\_SinglePortRAM.v  `timescale 1ns / 1ps  module Syn\_SinglePortRAM#(parameter DATA\_WIDTH = 4,parameter ADDR\_DEPTH = 4)(  input clk,rst,  input [ADDR\_DEPTH-1:0]addr,  input [DATA\_WIDTH-1:0]data\_in,  input we,  output reg[DATA\_WIDTH-1:0]data\_out  );  reg [DATA\_WIDTH-1:0] RAM[(1<<ADDR\_DEPTH)-1:0];  always @(posedge clk or posedge rst)  begin  if(rst)  begin:init\_RAM  integer i;//必须声明在有名字的块中，或写在外面  for(i=0;i<(1<<ADDR\_DEPTH);i=i+1)  begin  RAM[i] <= 0;  end  end  else if(we)  // write  begin  RAM[addr] <= data\_in;  end  else if(!we)  // read  begin  data\_out <= RAM[addr];  end  else  begin  data\_out <= 0;  end  end  endmodule  转换模块  transformer.v  module transformer(  input [3:0] data,  output [15:0]BCD//四位，方便输入至数码管  );  //移位加3，转换成BCD  reg [19:0] transfor\_data;  always @(\*)  begin  transfor\_data = 16'b0;  transfor\_data[3:0] = data;  repeat(4)  begin  if(transfor\_data[19:16]>4)  transfor\_data[19:16] = transfor\_data[19:16]+2'b11;  if(transfor\_data[15:12]>4)  transfor\_data[15:12] = transfor\_data[15:12]+2'b11;  if(transfor\_data[11:8]>4)  transfor\_data[11:8] = transfor\_data[11:8]+2'b11;  if(transfor\_data[7:4]>4)  transfor\_data[7:4] = transfor\_data[7:4]+2'b11;  transfor\_data[19:1] = transfor\_data[18:0];  end  end  assign BCD = transfor\_data[19:4];  endmodule  七段数码管显示模块  display7seg.v  module display7seg(  input clk,  input [3:0]data3,data2,data1,data0,  output reg[3:0]an,  output reg[6:0]display  );  reg [1:0] count;  always @(posedge clk) begin  if(count == 'b11)  count <= 0;  else  count <= count +'b1;  end  always @(posedge clk) begin  case(count)  2'b00: an <= 4'b1110;  2'b01: an <= 4'b1101;  2'b10: an <= 4'b1011;  2'b11: an <= 4'b0111;  endcase  end  always @(posedge clk) begin  case(count)  2'b00:  case (data0)  4'b0000:display = 7'b0000001;  4'b0001:display = 7'b1001111;  4'b0010:display = 7'b0010010;  4'b0011:display = 7'b0000110;  4'b0100:display = 7'b1001100;  4'b0101:display = 7'b0100100;  4'b0110:display = 7'b0100000;  4'b0111:display = 7'b0001111;  4'b1000:display = 7'b0000000;  4'b1001:display = 7'b0000100;  endcase  2'b01:  case (data1)  4'b0000:display = 7'b0000001;  4'b0001:display = 7'b1001111;  4'b0010:display = 7'b0010010;  4'b0011:display = 7'b0000110;  4'b0100:display = 7'b1001100;  4'b0101:display = 7'b0100100;  4'b0110:display = 7'b0100000;  4'b0111:display = 7'b0001111;  4'b1000:display = 7'b0000000;  4'b1001:display = 7'b0000100;  endcase  2'b10:  case (data2)  4'b0000:display = 7'b0000001;  4'b0001:display = 7'b1001111;  4'b0010:display = 7'b0010010;  4'b0011:display = 7'b0000110;  4'b0100:display = 7'b1001100;  4'b0101:display = 7'b0100100;  4'b0110:display = 7'b0100000;  4'b0111:display = 7'b0001111;  4'b1000:display = 7'b0000000;  4'b1001:display = 7'b0000100;  endcase  2'b11:  case (data3)  4'b0000:display = 7'b0000001;  4'b0001:display = 7'b1001111;  4'b0010:display = 7'b0010010;  4'b0011:display = 7'b0000110;  4'b0100:display = 7'b1001100;  4'b0101:display = 7'b0100100;  4'b0110:display = 7'b0100000;  4'b0111:display = 7'b0001111;  4'b1000:display = 7'b0000000;  4'b1001:display = 7'b0000100;  endcase  endcase  end  约束文件  set\_property IOSTANDARD LVCMOS33 [get\_ports {addr[0]}]  set\_property IOSTANDARD LVCMOS33 [get\_ports {addr[1]}]  set\_property IOSTANDARD LVCMOS33 [get\_ports {addr[2]}]  set\_property IOSTANDARD LVCMOS33 [get\_ports {addr[3]}]  set\_property IOSTANDARD LVCMOS33 [get\_ports {an[0]}]  set\_property IOSTANDARD LVCMOS33 [get\_ports {an[2]}]  set\_property IOSTANDARD LVCMOS33 [get\_ports {an[1]}]  set\_property IOSTANDARD LVCMOS33 [get\_ports {an[3]}]  set\_property IOSTANDARD LVCMOS33 [get\_ports clk]  set\_property IOSTANDARD LVCMOS33 [get\_ports {data\_in[0]}]  set\_property IOSTANDARD LVCMOS33 [get\_ports {data\_in[1]}]  set\_property IOSTANDARD LVCMOS33 [get\_ports {data\_in[2]}]  set\_property IOSTANDARD LVCMOS33 [get\_ports {data\_in[3]}]  set\_property IOSTANDARD LVCMOS33 [get\_ports {display[0]}]  set\_property IOSTANDARD LVCMOS33 [get\_ports {display[1]}]  set\_property IOSTANDARD LVCMOS33 [get\_ports {display[2]}]  set\_property IOSTANDARD LVCMOS33 [get\_ports {display[3]}]  set\_property IOSTANDARD LVCMOS33 [get\_ports {display[4]}]  set\_property IOSTANDARD LVCMOS33 [get\_ports {display[5]}]  set\_property IOSTANDARD LVCMOS33 [get\_ports {display[6]}]  set\_property IOSTANDARD LVCMOS33 [get\_ports rst]  set\_property IOSTANDARD LVCMOS33 [get\_ports we]  set\_property PACKAGE\_PIN W5 [get\_ports clk]  set\_property PACKAGE\_PIN R2 [get\_ports we]  set\_property PACKAGE\_PIN U18 [get\_ports rst]  set\_property PACKAGE\_PIN W7 [get\_ports {display[6]}]  set\_property PACKAGE\_PIN W6 [get\_ports {display[5]}]  set\_property PACKAGE\_PIN U8 [get\_ports {display[4]}]  set\_property PACKAGE\_PIN V8 [get\_ports {display[3]}]  set\_property PACKAGE\_PIN U5 [get\_ports {display[2]}]  set\_property PACKAGE\_PIN V5 [get\_ports {display[1]}]  set\_property PACKAGE\_PIN U7 [get\_ports {display[0]}]  set\_property PACKAGE\_PIN T1 [get\_ports {data\_in[3]}]  set\_property PACKAGE\_PIN U1 [get\_ports {data\_in[2]}]  set\_property PACKAGE\_PIN W2 [get\_ports {data\_in[1]}]  set\_property PACKAGE\_PIN R3 [get\_ports {data\_in[0]}]  set\_property PACKAGE\_PIN W4 [get\_ports {an[3]}]  set\_property PACKAGE\_PIN V4 [get\_ports {an[2]}]  set\_property PACKAGE\_PIN U4 [get\_ports {an[1]}]  set\_property PACKAGE\_PIN U2 [get\_ports {an[0]}]  set\_property PACKAGE\_PIN T2 [get\_ports {addr[3]}]  set\_property PACKAGE\_PIN T3 [get\_ports {addr[2]}]  set\_property PACKAGE\_PIN V2 [get\_ports {addr[1]}]  set\_property PACKAGE\_PIN W13 [get\_ports {addr[0]}]  **单端口异步RAM**  顶层模块  top\_Asy\_SinglePortRAM.v  module top\_Asy\_SinglePortRAM#(parameter DATA\_WIDTH = 4,parameter ADDR\_DEPTH = 4)(  input clk,rst,  input [ADDR\_DEPTH-1:0]addr,  input [DATA\_WIDTH-1:0]data\_in,  input we,  output wire[3:0]an,  output wire[6:0]display  );  wire[DATA\_WIDTH-1:0]data\_out;  wire clk\_div;  reg [25:0]target = 50000;  wire [15:0]BCD;  //divider  divider d(.clk(clk),.rst(rst),.target(target),.clk\_div(clk\_div));  //Asy\_SinglePortRAM  Asy\_SinglePortRAM S(.clk(clk),.rst(rst),.addr(addr),.data\_in(data\_in),.we(we),.data\_out(data\_out));  //transformer  transformer t(.data(data\_out),.BCD(BCD));  //display7seg  display7seg dis(.clk(clk\_div),.data3(BCD[15:12]),.data2(BCD[11:8]),.data1(BCD[7:4]),.data0(BCD[3:0]),.an(an),.display(display));  endmodule  其余文件与单端口同步RAM相同  **双端口同步RAM**  顶层模块  top\_Syn\_DoublePortRAM.v  module top\_Syn\_DoublePortRAM#(parameter DATA\_WIDTH = 4,parameter ADDR\_DEPTH = 3)(  input clk,rst,  input [ADDR\_DEPTH-1:0]addr\_a,addr\_b,  input [DATA\_WIDTH-1:0]din\_a,din\_b,  input we\_a,we\_b,  output wire[3:0]an,  output wire[6:0]display,  output wire error//错误检测灯  );  wire[DATA\_WIDTH-1:0]dout\_a,dout\_b;  wire clk\_div;  reg [25:0]target = 50000;  wire [15:0]BCD\_a;  wire [15:0]BCD\_b;  //divider  divider d(.clk(clk),.rst(rst),.target(target),.clk\_div(clk\_div));  //Syn\_DoublePortRAM  Syn\_DoublePortRAM S(.clk(clk),.rst(rst),.addr\_a(addr\_a),.addr\_b(addr\_b),.din\_a(din\_a),.din\_b(din\_b),.we\_a(we\_a),.we\_b(we\_b),.dout\_a(dout\_a),.dout\_b(dout\_b),.error(error));  //transformer  transformer t\_a(.data(dout\_a),.BCD(BCD\_a));  transformer t\_b(.data(dout\_b),.BCD(BCD\_b));  //display7seg  display7seg dis(.clk(clk\_div),.data3(BCD\_a[7:4]),.data2(BCD\_a[3:0]),.data1(BCD\_b[7:4]),.data0(BCD\_b[3:0]),.an(an),.display(display));  endmodule  分频模块  divider.v  module divider(  input clk,rst,  input [15:0] target,  output reg clk\_div  );  reg [15:0] counter;  always @(posedge clk) begin  if(rst)  begin  counter <= 0;  clk\_div <= 0;  end  else if(counter==target)  begin  counter <= 0;  clk\_div <= ~clk\_div;  end  else  counter <= counter+1;  end  endmodule  双端口同步RAM  Syn\_DoublePortRAM.v  module Syn\_DoublePortRAM#(parameter DATA\_WIDTH = 4,parameter ADDR\_DEPTH = 3)(  input clk,rst,  input [ADDR\_DEPTH-1:0]addr\_a,addr\_b,  input [DATA\_WIDTH-1:0]din\_a,din\_b,  input we\_a,we\_b,  output reg[DATA\_WIDTH-1:0]dout\_a,dout\_b,  output reg error//错误检测灯  );  reg [DATA\_WIDTH-1:0] RAM[(1<<ADDR\_DEPTH)-1:0];  //error检测  always @(posedge clk)  begin  if(rst)  begin  error <= 0;  end  //地址相同时只能read  else if(we\_a&&we\_b&&(addr\_a==addr\_b)) //地址相同但都要写入时  error <= 1;//error指示灯亮  else  error <= 0;  end  //write  integer i;  always @(posedge clk)  begin  if(rst)//全部清零  begin  for(i=0;i<(1<<ADDR\_DEPTH);i = i+1)  begin  RAM[i] <= 0;  end  end  else if(we\_a&&!we\_b&&(addr\_a!=addr\_b))  RAM[addr\_a] = din\_a;  else if(!we\_a&&we\_b&&(addr\_a!=addr\_b))  RAM[addr\_b] = din\_b;  else if(we\_a&&we\_b&&(addr\_a!=addr\_b))  begin  RAM[addr\_a] = din\_a;  RAM[addr\_b] = din\_b;  end  end  //read  //syn\_a  always @(posedge clk)  begin  if(rst)  begin  dout\_a <= 0;  end  else if(!we\_a)  begin  dout\_a <= RAM[addr\_a];  end  else  dout\_a <= 0;  end  // syn\_b  always @(posedge clk)  begin  if(rst)  begin  dout\_b <= 0;  end  else if(!we\_b)  begin  dout\_b <= RAM[addr\_b];  end  else  dout\_b <= 0;  end  endmodule  转换模块  transformer.v  module transformer(  input [3:0] data,  output [15:0]BCD  );  reg [19:0] transfor\_data;  always @(\*)  begin  transfor\_data = 16'b0;  transfor\_data[3:0] = data;  repeat(4)  begin  if(transfor\_data[19:16]>4)  transfor\_data[19:16] = transfor\_data[19:16]+2'b11;  if(transfor\_data[15:12]>4)  transfor\_data[15:12] = transfor\_data[15:12]+2'b11;  if(transfor\_data[11:8]>4)  transfor\_data[11:8] = transfor\_data[11:8]+2'b11;  if(transfor\_data[7:4]>4)  transfor\_data[7:4] = transfor\_data[7:4]+2'b11;  transfor\_data[19:1] = transfor\_data[18:0];  end  end  assign BCD = transfor\_data[19:4];  endmodule  七段数码管显示模块  display7seg.v  module display7seg(  input clk,  input [3:0]data3,data2,data1,data0,  output reg[3:0]an,  output reg[6:0]display  );  reg [1:0] count;  always @(posedge clk) begin  if(count == 'b11)  count <= 0;  else  count <= count +'b1;  end  always @(posedge clk) begin  case(count)  2'b00: an <= 4'b1110;  2'b01: an <= 4'b1101;  2'b10: an <= 4'b1011;  2'b11: an <= 4'b0111;  endcase  end  always @(posedge clk) begin  case(count)  2'b00:  case (data0)  4'b0000:display = 7'b0000001;  4'b0001:display = 7'b1001111;  4'b0010:display = 7'b0010010;  4'b0011:display = 7'b0000110;  4'b0100:display = 7'b1001100;  4'b0101:display = 7'b0100100;  4'b0110:display = 7'b0100000;  4'b0111:display = 7'b0001111;  4'b1000:display = 7'b0000000;  4'b1001:display = 7'b0000100;  endcase  2'b01:  case (data1)  4'b0000:display = 7'b0000001;  4'b0001:display = 7'b1001111;  4'b0010:display = 7'b0010010;  4'b0011:display = 7'b0000110;  4'b0100:display = 7'b1001100;  4'b0101:display = 7'b0100100;  4'b0110:display = 7'b0100000;  4'b0111:display = 7'b0001111;  4'b1000:display = 7'b0000000;  4'b1001:display = 7'b0000100;  endcase  2'b10:  case (data2)  4'b0000:display = 7'b0000001;  4'b0001:display = 7'b1001111;  4'b0010:display = 7'b0010010;  4'b0011:display = 7'b0000110;  4'b0100:display = 7'b1001100;  4'b0101:display = 7'b0100100;  4'b0110:display = 7'b0100000;  4'b0111:display = 7'b0001111;  4'b1000:display = 7'b0000000;  4'b1001:display = 7'b0000100;  endcase  2'b11:  case (data3)  4'b0000:display = 7'b0000001;  4'b0001:display = 7'b1001111;  4'b0010:display = 7'b0010010;  4'b0011:display = 7'b0000110;  4'b0100:display = 7'b1001100;  4'b0101:display = 7'b0100100;  4'b0110:display = 7'b0100000;  4'b0111:display = 7'b0001111;  4'b1000:display = 7'b0000000;  4'b1001:display = 7'b0000100;  endcase  endcase  end  endmodule  **双端口异步RAM**  top\_Asy\_DoublePortRAM.v  module top\_Asy\_DoublePortRAM#(parameter DATA\_WIDTH = 3,parameter ADDR\_DEPTH = 3)(  input clk,rst,  input [ADDR\_DEPTH-1:0]addr\_a,addr\_b,  input [DATA\_WIDTH-1:0]din\_a,din\_b,  input we\_a,we\_b,  output wire[3:0]an,  output wire[6:0]display,  output wire error//错误指示灯  );  wire[DATA\_WIDTH-1:0]dout\_a,dout\_b;  wire clk\_div;  reg [25:0]target = 50000;  wire [15:0]BCD\_a;  wire [15:0]BCD\_b;  //divider  divider d(.clk(clk),  .rst(rst),  .target(target),  .clk\_div(clk\_div));  //Syn\_DoublePortRAM  Asy\_DoublePortRAM S(.clk(clk),  .rst(rst),  .addr\_a(addr\_a),  .addr\_b(addr\_b),  .din\_a(din\_a),  .din\_b(din\_b),  .we\_a(we\_a),  .we\_b(we\_b),  .dout\_a(dout\_a),  .dout\_b(dout\_b),  .error(error));  //transformer  transformer t\_a(.data(dout\_a),.BCD(BCD\_a));  transformer t\_b(.data(dout\_b),.BCD(BCD\_b));  //display7seg  display7seg dis(.clk(clk\_div),.data3(BCD\_a[7:4]),.data2(BCD\_a[3:0]),.data1(BCD\_b[7:4]),.data0(BCD\_b[3:0]),.an(an),.display(display));  endmodule  其余文件与双端口同步RAM相同  **FIFO**  顶层模块  top\_FIFO.v  module top\_FIFO#(parameter DATA\_WIDTH = 4,parameter ADDR\_DEPTH = 4)(  input button,clk,rst,wr\_en,rd\_en,  input [DATA\_WIDTH-1:0]data\_in,  output empty,full,  output wire[3:0]an,  output wire[6:0]display  );  wire button\_deb;  wire[DATA\_WIDTH-1:0]data\_out;  wire clk\_div;  reg [25:0]target = 50000;  wire [15:0]BCD;  //divider  divider d(.clk(clk),.rst(rst),.target(target),.clk\_div(clk\_div));  //debkey  debkey deb(.clk(clk),.rst(rst),.key\_in(button),.key\_out(button\_deb));  //FIFO  FIFO F(.clk(button\_deb),.rst(rst),.wr\_en(wr\_en),.rd\_en(rd\_en),.data\_in(data\_in),.empty(empty),.full(full),.data\_out(data\_out));  //transformer  transformer t(.data(data\_out),.BCD(BCD));  //display7seg  display7seg dis(.clk(clk\_div),.data3(BCD[15:12]),.data2(BCD[11:8]),.data1(BCD[7:4]),.data0(BCD[3:0]),.an(an),.display(display));  endmodule  分频模块  divider.v  module divider(  input clk,rst,  input [15:0] target,  output reg clk\_div  );  reg [15:0] counter;  always @(posedge clk) begin  if(rst)  begin  counter <= 0;  clk\_div <= 0;  end  else if(counter==target)  begin  counter <= 0;  clk\_div <= ~clk\_div;  end  else  counter <= counter+1;  end  endmodule  消抖模块  devkey.v  module debkey(  input clk,  input rst,  input key\_in,  output key\_out  );  parameter T100Hz = 249999;  integer cnt\_100Hz;  reg clk\_100Hz;  always @(posedge clk) begin  if(rst)  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  end  reg[2:0]key\_rrr,key\_rr,key\_r;  always @(posedge clk\_100Hz) begin  if(rst)  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\_in;  end  end  assign key\_out = key\_rrr&key\_rr&key\_r;  endmodule  FIFO实现模块  FIFO.v  module FIFO#(parameter DATA\_WIDTH = 4,parameter ADDR\_DEPTH = 4)(  input clk,rst,wr\_en,rd\_en,  input [DATA\_WIDTH-1:0] data\_in,  output reg empty,full,  output reg[DATA\_WIDTH-1:0] data\_out  );  reg [DATA\_WIDTH-1:0] FIFO[(1<<ADDR\_DEPTH) - 1:0];  reg [ADDR\_DEPTH-1:0]head;  reg [ADDR\_DEPTH-1:0]rear;  reg [ADDR\_DEPTH:0]NUM;    //empty  always @(\*) begin  if(NUM==0)  empty<=1;  else  empty<=0;  end  //full  always @(\*) begin  if(NUM==(1<<ADDR\_DEPTH))  full<=1;  else  full<=0;  end  //NUM  always @(posedge clk or posedge rst) begin  if(rst)  NUM<=0;  else if(!wr\_en&&!rd\_en)//no write no read  NUM<=NUM;  else if(wr\_en&&!rd\_en&&(NUM<(1<<ADDR\_DEPTH)))//wirte no read  NUM<=NUM+'b1;  else if(!wr\_en&&rd\_en&&(NUM>0))//read no write  NUM<=NUM-'b1;  else if(wr\_en&&rd\_en)  NUM<=NUM;  end  //write  integer i;  always @(posedge clk or posedge rst) begin  if(rst)  begin  rear<='b0;  for(i=0;i<(1<<ADDR\_DEPTH);i=i+1)  FIFO[i]<=0;  end  else if(wr\_en&&(NUM<(1<<ADDR\_DEPTH)))//not full  begin  FIFO[rear]<=data\_in;  rear<=(rear+1)%(1<<ADDR\_DEPTH);  end  else if(wr\_en&&(NUM>(1<<ADDR\_DEPTH)-1))//full  rear<=rear;  end  //read  always @(posedge clk or posedge rst) begin  if(rst)  begin  head<='b0;  data\_out<=0;  end    else if(rd\_en&&(NUM!=0))  begin  data\_out<=FIFO[head];  head<=(head+1)%(1<<ADDR\_DEPTH);  end  else if(NUM==0)  data\_out<=0;  end  endmodule  转换模块  transformer.v  module transformer(  input [3:0] data,  output [15:0]BCD  );  reg [19:0] transfor\_data;  always @(\*)  begin  transfor\_data = 16'b0;  transfor\_data[3:0] = data;  repeat(4)  begin  if(transfor\_data[19:16]>4)  transfor\_data[19:16] = transfor\_data[19:16]+2'b11;  if(transfor\_data[15:12]>4)  transfor\_data[15:12] = transfor\_data[15:12]+2'b11;  if(transfor\_data[11:8]>4)  transfor\_data[11:8] = transfor\_data[11:8]+2'b11;  if(transfor\_data[7:4]>4)  transfor\_data[7:4] = transfor\_data[7:4]+2'b11;  transfor\_data[19:1] = transfor\_data[18:0];  end  end  assign BCD = transfor\_data[19:4];  endmodule  七段数码管显示模块  display7seg.v  module display7seg(  input clk,  input [3:0]data3,data2,data1,data0,  output reg[3:0]an,  output reg[6:0]display  );  reg [1:0] count;  always @(posedge clk) begin  if(count == 'b11)  count <= 0;  else  count <= count +'b1;  end  always @(posedge clk) begin  case(count)  2'b00: an <= 4'b1110;  2'b01: an <= 4'b1101;  2'b10: an <= 4'b1011;  2'b11: an <= 4'b0111;  endcase  end  always @(posedge clk) begin  case(count)  2'b00:  case (data0)  4'b0000:display = 7'b0000001;  4'b0001:display = 7'b1001111;  4'b0010:display = 7'b0010010;  4'b0011:display = 7'b0000110;  4'b0100:display = 7'b1001100;  4'b0101:display = 7'b0100100;  4'b0110:display = 7'b0100000;  4'b0111:display = 7'b0001111;  4'b1000:display = 7'b0000000;  4'b1001:display = 7'b0000100;  endcase  2'b01:  case (data1)  4'b0000:display = 7'b0000001;  4'b0001:display = 7'b1001111;  4'b0010:display = 7'b0010010;  4'b0011:display = 7'b0000110;  4'b0100:display = 7'b1001100;  4'b0101:display = 7'b0100100;  4'b0110:display = 7'b0100000;  4'b0111:display = 7'b0001111;  4'b1000:display = 7'b0000000;  4'b1001:display = 7'b0000100;  endcase  2'b10:  case (data2)  4'b0000:display = 7'b0000001;  4'b0001:display = 7'b1001111;  4'b0010:display = 7'b0010010;  4'b0011:display = 7'b0000110;  4'b0100:display = 7'b1001100;  4'b0101:display = 7'b0100100;  4'b0110:display = 7'b0100000;  4'b0111:display = 7'b0001111;  4'b1000:display = 7'b0000000;  4'b1001:display = 7'b0000100;  endcase  2'b11:  case (data3)  4'b0000:display = 7'b0000001;  4'b0001:display = 7'b1001111;  4'b0010:display = 7'b0010010;  4'b0011:display = 7'b0000110;  4'b0100:display = 7'b1001100;  4'b0101:display = 7'b0100100;  4'b0110:display = 7'b0100000;  4'b0111:display = 7'b0001111;  4'b1000:display = 7'b0000000;  4'b1001:display = 7'b0000100;  endcase  endcase  end  endmodule  约束文件  set\_property CLOCK\_DEDICATED\_ROUTE FALSE [get\_nets button\_IBUF]  set\_property IOSTANDARD LVCMOS33 [get\_ports {an[0]}]  set\_property IOSTANDARD LVCMOS33 [get\_ports {an[1]}]  set\_property IOSTANDARD LVCMOS33 [get\_ports {an[2]}]  set\_property IOSTANDARD LVCMOS33 [get\_ports {an[3]}]  set\_property IOSTANDARD LVCMOS33 [get\_ports button]  set\_property IOSTANDARD LVCMOS33 [get\_ports clk]  set\_property IOSTANDARD LVCMOS33 [get\_ports {data\_in[0]}]  set\_property IOSTANDARD LVCMOS33 [get\_ports {data\_in[1]}]  set\_property IOSTANDARD LVCMOS33 [get\_ports {data\_in[2]}]  set\_property IOSTANDARD LVCMOS33 [get\_ports {data\_in[3]}]  set\_property IOSTANDARD LVCMOS33 [get\_ports {display[0]}]  set\_property IOSTANDARD LVCMOS33 [get\_ports {display[1]}]  set\_property IOSTANDARD LVCMOS33 [get\_ports {display[2]}]  set\_property IOSTANDARD LVCMOS33 [get\_ports {display[3]}]  set\_property IOSTANDARD LVCMOS33 [get\_ports {display[4]}]  set\_property IOSTANDARD LVCMOS33 [get\_ports {display[5]}]  set\_property IOSTANDARD LVCMOS33 [get\_ports {display[6]}]  set\_property IOSTANDARD LVCMOS33 [get\_ports full]  set\_property IOSTANDARD LVCMOS33 [get\_ports rd\_en]  set\_property IOSTANDARD LVCMOS33 [get\_ports rst]  set\_property IOSTANDARD LVCMOS33 [get\_ports wr\_en]  set\_property IOSTANDARD LVCMOS33 [get\_ports empty]  set\_property PACKAGE\_PIN R2 [get\_ports wr\_en]  set\_property PACKAGE\_PIN U18 [get\_ports rst]  set\_property PACKAGE\_PIN T1 [get\_ports rd\_en]  set\_property PACKAGE\_PIN L1 [get\_ports full]  set\_property PACKAGE\_PIN P1 [get\_ports empty]  set\_property PACKAGE\_PIN W7 [get\_ports {display[6]}]  set\_property PACKAGE\_PIN W6 [get\_ports {display[5]}]  set\_property PACKAGE\_PIN U8 [get\_ports {display[4]}]  set\_property PACKAGE\_PIN V8 [get\_ports {display[3]}]  set\_property PACKAGE\_PIN U5 [get\_ports {display[2]}]  set\_property PACKAGE\_PIN V5 [get\_ports {display[1]}]  set\_property PACKAGE\_PIN U7 [get\_ports {display[0]}]  set\_property PACKAGE\_PIN U1 [get\_ports {data\_in[3]}]  set\_property PACKAGE\_PIN W2 [get\_ports {data\_in[2]}]  set\_property PACKAGE\_PIN R3 [get\_ports {data\_in[1]}]  set\_property PACKAGE\_PIN T2 [get\_ports {data\_in[0]}]  set\_property PACKAGE\_PIN W5 [get\_ports clk]  set\_property PACKAGE\_PIN W19 [get\_ports button]  set\_property PACKAGE\_PIN W4 [get\_ports {an[3]}]  set\_property PACKAGE\_PIN V4 [get\_ports {an[2]}]  set\_property PACKAGE\_PIN U4 [get\_ports {an[1]}]  set\_property PACKAGE\_PIN U2 [get\_ports {an[0]}] | | | | | | |
| 五、实验过程中遇到的问题及解决情况  1.设计双端口RAM的时候会出现“竞争-冒险”现象，即两个写使能和地址信号都有效时如果不进行处理，则结果具有不确定性  解决方案：写使能有效时判断地址是否相同，相同的话则是error指示灯亮，提示此时两个写操作的地址相同，保证只进行读操作。  2.设计FIFO时，下载到开发板上会出现“一直读一直写”的现象  解决方案：发现是时钟频率太快了，于是将clk绑定到开关上，手动控制clk，成功实现一次只操作一次。 | | | | | | |
| 六、实验结果及分析和（或）源程序调试过程  **单端口RAM仿真：**  对于同步读：addr的输入均在时钟下降沿，data\_out预期在上升沿输出，因此将会看到data\_out相对于addr滞后的波形图。  对于异步读：addr的输入均在时钟下降沿，data\_out预期在下降沿输出，因此将会看到data\_out相对于addr同步的波形图。  仿真代码（同步异步相同）：  `timescale 1ns / 1ps  module Syn\_SinglePortRAM\_tb(  );  //parameter  parameter DATA\_WIDTH = 4;  parameter ADDR\_DEPTH = 4;  //inputs  reg clk,rst;  reg [ADDR\_DEPTH-1:0]addr;  reg [DATA\_WIDTH-1:0]data\_in;  reg we;  //output  wire[DATA\_WIDTH-1:0]data\_out;  //init  initial begin  clk = 0;  forever #5 clk = ~clk;  end  Syn\_SinglePortRAM u(  .clk(clk),  .rst(rst),  .addr(addr[ADDR\_DEPTH-1:0]),  .data\_in(data\_in[DATA\_WIDTH-1:0]),  .we(we),  .data\_out(data\_out[DATA\_WIDTH-1:0])  );  initial begin  #10 rst = 1;//reset  #10 rst = 0;  //write test  #5 we = 1;  #10 addr=4'b0000;data\_in=$random;  #10 addr=4'b0001;data\_in=$random;  #10 addr=4'b0010;data\_in=$random;  #10 addr=4'b0011;data\_in=$random;  #10 addr=4'b0100;data\_in=$random;  #10 addr=4'b0101;data\_in=$random;  #10 addr=4'b0110;data\_in=$random;  #10 addr=4'b0111;data\_in=$random;  #10 addr=4'b1000;data\_in=$random;  #10 addr=4'b1001;data\_in=$random;  #10 addr=4'b1010;data\_in=$random;  #10 addr=4'b1011;data\_in=$random;  #10 addr=4'b1100;data\_in=$random;  #10 addr=4'b1101;data\_in=$random;  #10 addr=4'b1110;data\_in=$random;  #10 addr=4'b1111;data\_in=$random;  //syn\_read test  #10 we = 0;  #10 @(negedge clk) addr=4'b0000;  #10 @(negedge clk) addr=4'b0001;  #10 @(negedge clk) addr=4'b0010;  #10 @(negedge clk) addr=4'b0011;  #10 @(negedge clk) addr=4'b0100;  #10 @(negedge clk) addr=4'b0101;  #10 @(negedge clk) addr=4'b0110;  #10 @(negedge clk) addr=4'b0111;  #10 @(negedge clk) addr=4'b1000;  #10 @(negedge clk) addr=4'b1001;  #10 @(negedge clk) addr=4'b1010;  #10 @(negedge clk) addr=4'b1011;  #10 @(negedge clk) addr=4'b1100;  #10 @(negedge clk) addr=4'b1101;  #10 @(negedge clk) addr=4'b1110;  #10 @(negedge clk) addr=4'b1111;  end  endmodule  同步读：    从波形图中可以看到，读取时，时钟下降沿时输入addr，在上升沿时才有data\_out输出。  异步读：    读取时，时钟下降沿时输入addr，同时data\_out输出。  **双端口RAM仿真：**  `timescale 1ns / 1ps  module DoublePortRAM\_tb();  reg clk,we1,we2,rst;  reg [2:0] addr1;  reg [2:0] addr2;  reg [3:0] d\_in1;  reg [3:0] d\_in2;  wire [3:0] d\_out1;  wire [3:0] d\_out2;  wire error;  Syn\_DoublePortRAM #(4,3) utt(  .clk(clk),.rst(rst),  .addr\_a(addr1), .addr\_b(addr2),  .din\_a(d\_in1), .din\_b(d\_in2),  .we\_a(we1), .we\_b(we2),  .dout\_a(d\_out1), .dout\_b(d\_out2),  .error(error));  always #5 clk = ~clk;  integer i;  initial begin  rst = 0;  clk = 0;  we1 = 0;  we2 = 0;  addr1 = 0;  addr2 = 0;  d\_in1 = 0;  d\_in2 = 0;    @(negedge clk)  begin  we1 = 1;  we2 = 1;  end  //写入测试1 ---两个地址冲突时  for (i=0;i<8;i=i+1)  begin  @(negedge clk)  begin  addr1 = i;  d\_in1 = i;  addr2 = i;  d\_in2 = i;  end  end  //写入测试2----两个地址同时写入，但位置不同  for(i=0;i<7;i=i+1)  begin  @(negedge clk)  begin  addr1=i;  d\_in1=i;  addr2=i+1;  d\_in2=i+i;  end  end  //读测试---测试刚才的数据有没有正确的被输入  @(negedge clk)  begin  we1 = 0;  we2 = 0;  end  for (i=0;i<8;i=i+1)  begin  @(negedge clk)  begin  addr1 = i;  addr2 = i;  end  end  //写入测试3---单端口写入b  @(negedge clk)  begin  we1 = 0;  we2 = 1;  end  for (i=0;i<8;i=i+1)  begin  @(negedge clk)  begin  addr2 = i;  d\_in2=7-i;  d\_in1=1;  end  end  @(negedge clk)  begin  we1 = 0;  we2 = 0;  end  for (i=0;i<8;i=i+1)  begin  @(negedge clk)  begin  addr2 = i;  end  end  //写入测试4---单端口写入a  @(negedge clk)  begin  we1 = 1;  we2 = 0;  end  for (i=0;i<8;i=i+1)  begin  @(negedge clk)  begin  addr1 = i;  d\_in1=2\*i;  d\_in2=i;  end  end  @(negedge clk)  begin  we1 = 0;  we2 = 0;  end  for (i=0;i<8;i=i+1)  begin  @(negedge clk)  begin  addr1 = i;  end  end  #20  $stop;  end  endmodule  同步读：    开始，同时写，当地址一样时，error为1；地址不一样时，error变为0。然后，1写2读，d\_out1输出0；1读2写，d\_out2输出0，由于不允许一个读，另一个修改同一个地址的数据，此时error也为1，地址不同时变为0；最后同时读，均有输出。并且读取时，时钟下降沿时输入addr，在上升沿时才有输出。  异步读：    读取时，时钟下降沿时输入addr，同时out输出。  **FIFO仿真：**  `timescale 1ns / 1ps  module FIFO\_tb(  );  //parameter  parameter DATA\_WIDTH = 4;  parameter ADDR\_DEPTH = 4;  reg clk,rst,wr\_en,rd\_en;  reg [DATA\_WIDTH-1:0]data\_in;  wire empty,full;  wire [DATA\_WIDTH-1:0] data\_out;  //init  initial begin  clk = 0;  forever #5 clk = ~clk;  end  FIFO F(  .clk(clk),  .rst(rst),  .wr\_en(wr\_en),  .rd\_en(rd\_en),  .data\_in(data\_in[DATA\_WIDTH-1:0]),  .empty(empty),  .full(full),  .data\_out(data\_out[DATA\_WIDTH-1:0]));  initial begin  #10 rst = 1;//reset  #10 rst = 0;  //write test  #10 wr\_en = 1;rd\_en=0;data\_in=$random;  #10 data\_in=$random;  #10 data\_in=$random;  #10 data\_in=$random;  #10 data\_in=$random;  #10 data\_in=$random;  #10 data\_in=$random;  #10 data\_in=$random;  #10 data\_in=$random;  #10 data\_in=$random;  #10 data\_in=$random;  #10 data\_in=$random;  #10 data\_in=$random;  #10 data\_in=$random;  #10 data\_in=$random;  #10 data\_in=$random;  //read test  #50 wr\_en = 0;rd\_en=1;  //write test  #10 wr\_en = 1;rd\_en=0;data\_in=$random;  #10 data\_in=$random;  #10 data\_in=$random;  #10 data\_in=$random;  #10 data\_in=$random;  #10 data\_in=$random;  #10 data\_in=$random;  #10 data\_in=$random;  #10 data\_in=$random;  #10 data\_in=$random;  #10 data\_in=$random;  #10 data\_in=$random;  #10 data\_in=$random;  #10 data\_in=$random;  #10 data\_in=$random;  #10 data\_in=$random;  //read test  #100 wr\_en = 0;rd\_en=1;  end  endmodule  仿真结果    **单端口RAM上板演示效果如下（由于同步和异步在开发板上无法演示出区别，故这里不进行同步和异步的区分）：**      这里开启写入按钮，将数值11（1011）存入到11（1011）的地址之中    这里关闭写入按钮，即转为读取状态，可以看到由11的地址可以读取出数值11，说明板子运行情况正常，达到期望的读写功能    这里演示开发板能存储的最大数值    可以看到在按下重置按钮后，各地址存储的数据全部归零，重置功能正常运行，至此检验完毕。  **双端口RAM演示：**      对a端口进行读写操作  在b端口的地址对a端口写入的数值进行访问    在相同地址，a,b两端口同时写入数据时，我们所设置的开发板的报错灯会亮    此时，只要通过关闭某一个写入开关，或者更改某一个地址，均可以消除报错灯    在按下重置键后，可以看到，原本存储有7的地址上存储的数据被清零，至此所有功能验证完毕  **FIFO演示：**      可以看到，在没有输入数据的情况下队列为空，队列空灯亮    打开写入开关，依次按下button输入数据，直到队列满灯亮      关闭写入开关，打开读取开关，依次按下button读出数据，直到队列空灯亮    再次打开写入开关，输入数据使队列不空后，按下重置键，可以看到队列空灯亮，数据已清空，至此所有功能验证完毕  七、小组分工情况说明  ：参与实验原理分析，实验设计，设计文件编写。撰写实验报告  ：参与实验原理分析，实验设计，仿真文件编写与测试。撰写实验报告  ：参与实验原理分析，实验设计，综合，生成bit流，并下载至开发板验证功能是否完整，有无错误，进行修改。撰写实验报告 | | | | | | |