**《数字逻辑》实验报告**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **姓名** | |  | | **年级** | |  |
| **学号** | |  | | **专业、班级** | |  |
| **实验名称** | **门电路设计——实验环境与开发板的熟悉** | | | | | |
| **实验时间** | **2019/10/25** | | **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   1. 熟悉vivado开发环境以及basys3开发板环境。 2. 熟悉verilog HDL语言的基本语法。 3. 掌握vivado IP核的封装与调用。 4. 加深对多路选择器的原理理解。 5. 掌握实现同一电路的不同方式，并学会分析并理解不同方法之间的异同。 | | | | | | |
| 1. 实验项目内容 2. 设计三位4选1多路选择器电路，分别使用Block Design和verilog HDL语言编写代码两种方式实现。 3. 通过仿真、看RTL电路图、下载到板子上验证其正确性。 4. 比较Block Design和verilog HDL语言编写代码两种方式的异同（逻辑资源、RTL电路图、仿真波形等方面）。 | | | | | | |
| 1. 实验设计 2. **实验原理**    1. 本实验是设计一个3位数据的4选1数据选择器，即需要从输入的4个3位二进制数中选择输出其中的一个数据。    2. 4选1的数据选择器，地址变量选用2位二进制数：，，，来控制输出4个数据的其中一个数据。    3. 具体变量含义如下（由变量决定哪一个数据输出）   输入数据  地址变量  输出数据   1. **真值表**      |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | | 输入 | | | | | | 输出 | |  |  |  |  |  |  |  | |  |  |  |  |  |  |  | |  |  |  |  |  |  |  | |  |  |  |  |  |  |  | |  |  |  |  |  |  |  | |  |  |  |  |  |  |  |  1. **原理图** | | | | | | |
| 1. 实验过程或算法(关键步骤、核心代码注解等） 2. **Block Design** 3. 使用三个二选一IP核构成四选一IP核。 4. 将前两个二选一模块的控制端口相连，通过该控制端选出两个输入信号，再通过第三个二选一模块选择两个输入信号中的一个，达到四选一的目的。      * 1. Block Design代码   `timescale 1 ps / 1 ps  (\* CORE\_GENERATION\_INFO = "mux3\_4\_1,IP\_Integrator,  {x\_ipProduct=Vivado  2018.3,x\_ipVendor=xilinx.com,x\_ipLibrary=BlockDiagram,x\_ipName=mux3\_4\_1,x\_ipVersion=1.00.a,x\_ipLanguage=VERILOG,  num Blks=3,numReposBlks=3,numNonXlnxBlks=0,numHierBlks=0,maxHierDepth=0,synth\_mode=Global}" \*) (\* HW\_HANDOFF = "mux3\_4\_1.hwdef" \*)     module mux3\_4\_1  (cntrl, cntrl\_2, f, in0, in0\_1, in1, in1\_1);    input cntrl;  input cntrl\_2;  output [2:0]f;  input [2:0]in0;  input [2:0]in0\_1;  input [2:0]in1;  input [2:0]in1\_1;  wire cntrl\_1;  wire cntrl\_2\_1;  wire [2:0]\^in0\_1 ;  wire [2:0]in0\_1\_1;  wire [2:0]\^in1\_1 ;  wire [2:0]in1\_1\_1;  wire [2:0]mux2\_1\_0\_f;  wire [2:0]mux2\_1\_1\_f;  wire [2:0]mux2\_1\_2\_f;  assign \^in0\_1 [2:0] = in0[2:0];  assign \^in1\_1 [2:0] = in1[2:0];  assign cntrl\_1 = cntrl;  assign cntrl\_2\_1 = cntrl\_2;  assign f[2:0] = mux2\_1\_2\_f;  assign in0\_1\_1 = in0\_1[2:0];  assign in1\_1\_1 = in1\_1[2:0];  mux3\_4\_1\_mux2\_1\_0\_0 mux2\_1\_0  (.cntrl(cntrl\_1),  .f(mux2\_1\_0\_f),  .in0(in0\_1\_1),  .in1(in1\_1\_1));  mux3\_4\_1\_mux2\_1\_1\_0 mux2\_1\_1  (.cntrl(cntrl\_1),  .f(mux2\_1\_1\_f),  .in0(\^in0\_1 ),  .in1(\^in1\_1 ));  mux3\_4\_1\_mux2\_1\_2\_1 mux2\_1\_2  (.cntrl(cntrl\_2\_1),  .f(mux2\_1\_2\_f),  .in0(mux2\_1\_1\_f),  .in1(mux2\_1\_0\_f));  endmodule   * 1. 仿真代码   **module mux3\_2\_1\_bd\_sim;**      reg [2:0] i0,i0\_1,i1,i1\_1;  reg c1,c2;  wire[2:0] fout;     mux3\_4\_1 mux1(.cntrl(c1),  .cntrl\_2(c2),  .f(fout),  .in0(i0),  .in0\_1(i0\_1),  .in1(i1),  .in1\_1(i1\_1)  );  initial  begin  i0=3'b000; i0\_1=3'b001; i1=3'b010; in1\_1=3'b011;  c1=0;c2=0;  #20  c1 = 0; c2 = 1;  #20  c1 = 1; c2 = 0;  #20  c1 = 1; c2 = 1;  #20  i0=3'b100; i0\_1=3'b101; i1=3'b110; in1\_1=3'b111;  c1=0;c2=0;  #20  c1 = 0; c2 = 1;  #20  c1 = 1; c2 = 0;  #20  c1 = 1; c2 = 1;  end  endmodule   * 1. 仿真结果   **（二）Verilog HDL语言编写代码**  ①四选一代码  module **mux3\_4\_1**(z, in0, in1, in2, in3, cntrl);  *//设定输入输出端口和控制端*      output[2:0] z;      input[2:0] in0, in1, in2, in3;      input[1:0] cntrl;      reg z;  *//输入输出端口或是控制端的信号发生了变化，执行always中的语句*      always @(in0 or in1 or in2 or in3 or cntrl)  *//按控制端的值选择case中语句进行执行*         case(cntrl)         2'b00: z=in0;         2'b01: z=in1;         2'b10: z=in2;         2'b11: z=in3;         default: z=3'bx;         endcase  endmodule  写完代码后，进行综合。  ②仿真代码：  module **mux3\_4\_1\_sim**;      reg[2:0] in0,in1,in2,in3;      reg[1:0] cntrl;  wire[2:0] z;      mux3\_4\_1 mux1(.z(z), .in0(in0), .in1(in1), .in2(in2), .in3(in3), .cntrl(cntrl));      initial          begin              in0=3'b000; in1=3'b001; in2=3'b010; in3=3'b011;              cntrl=2'b00;              #20  cntrl=2'b01;              #20  cntrl=2'b10;              #20  cntrl=2'b11;              #20              in0=3'b100; in1=3'b101; in2=3'b110; in3=3'b111;              cntrl=2'b00;              #20  cntrl=2'b01;              #20  cntrl=2'b10;              #20  cntrl=2'b11;          end  endmodule  ③仿真结果 | | | | | | |
| 五、实验过程中遇到的问题及解决情况   1. 没有理清block design的实现方式。   解决方式：   1. 通过搜索block design维基百科等资源，理解block design的理念。 2. 在xilinx的官方网站中搜索block design的相关资料。 3. 在国内技术论坛（CSDN、博客园、电子发烧友等）进一步加深其理解。 | | | | | | |
| 六、实验结果及分析和（或）源程序调试过程  （一） 实验结果展示（源程序调试过程）  编写Testbench，进行行为仿真，得到各信号的数数值与波形，从而测试模块的功能。  Testbench代码:  `timescale 1ns / 1ps  module **mux3\_4\_1\_sim**;      reg[2:0] in0,in1,in2,in3;      reg[1:0] cntrl;  wire[2:0] z;      mux3\_4\_1 mux1(.z(z), .in0(in0), .in1(in1), .in2(in2), .in3(in3), .cntrl(cntrl));      initial          begin              in0=3'b000; in1=3'b001; in2=3'b010; in3=3'b011;              cntrl=2'b00;              #20  cntrl=2'b01;              #20  cntrl=2'b10;              #20  cntrl=2'b11;              #20              in0=3'b100; in1=3'b101; in2=3'b110; in3=3'b111;              cntrl=2'b00;              #20  cntrl=2'b01;              #20  cntrl=2'b10;              #20  cntrl=2'b11;          end  endmodule  行为仿真运行结果：    首先不将信号按位展开，仅查看其按照BCD编码后对应的十进制数值，观察信号选择的情况。      再将各信号按位展开，观察信号的数值与对应的位值的情况。  通过仿真得到的信号可以看出：  当选择信号cntrl值为0时，输出信号z的值为4，与输入信号in0的值一致；  当选择信号cntrl值为1时，输出信号z的值为5，与输入信号in1的值一致；  当选择信号cntrl值为2时，输出信号z的值为6，与输入信号in2的值一致；  当选择信号cntrl值为3时，输出信号z的值为7，与输入信号in3的值一致；  程序烧写成功后，将开发板通电。开发板下方左边12个拨码开关定义为输入信号in0、in1、in2、in3，最右边2个拨码开关为选择信号cntrl。    首先将最右边的两个拨码均拨下，即选择信号为00，此时推上最右边的拨码开关，即将输入信号in0置为100，可以发现输出信号也为100。    将左边三个拨码开关均拨上，即将输入信号in0置为111，可以发现输出信号也为111。    此时将除了左边三个拨码开关全拨上，发现没有输出信号，即只要cntrl信号为00，输出信号与in0始终一致。  同理测试当cntrl为01、10和11时，发现输出始终与in1,in2,in3相同。  通过如上现象可以说明本次设计的模块设计正确，功能正常。  （二） block design方式和verilog HDL方式的异同比较  说明：本次比较了  方式（a）verilog HDL语句的三目运算符  方式（b）verilog HDL语句的case  方式（c）block design使用2选1 IP核  方式（d）block design 使用4选1 IP核   1. RTL电路图情况  |  |  | | --- | --- | | （a）：？ |  | | （b）case |  | | （c）Block Design 2选1 IP core |  | | （d）Block Design 4选1 IP core |  |  1. 可以看出使用verilog HDL语言编写代码，不论是使用case 语句还是三目运算符，其RTL电路图没有明显差别。、 2. 而使用block design，通过封装的ip核的方式可以发现，被封装的ip核内部电路不可见，只有ip核的输入输出端口以及各个ip核的逻辑连接关系式可见的。 3. 逻辑资源使用情况  |  |  | | --- | --- | | （a）：？ | 计算机生成了可选文字: 0  Name  threeMux4t01  Hierarchy  Slice LUTs  （ 20800 ）  3  LUT as Logic  （ 20800 ）  3  Bonded 10B  （ 106 ）  Slice  （ 8150 ）  1 | | 计算机生成了可选文字: Resource  LUT  0  16 ％  Utilization  50  Available  3  Utilization （ ％ ）  Utilization ％  100  20800  106  0 ． 01  16 ． 04 | | （b）case |  | |  | | （c）Block Design 2选1 IP core |  | |  | | （d）Block Design 4选1 IP core |  | |  |  1. 可以发现，使用3个2选1 IP core通过block design进行设计实现的时候，还是用了个6个LUT as Logic，而其他的方式均使用了3个LUT as logic。 2. IO资源使用情况均一致。 3. 因此使用3个2选1 IP core通过block design设计使用的逻辑资源较多。 4. 功率分析   在统一环境下   |  |  | | --- | --- | | （a）：？ |  | | （b）case |  | | （c）Block Design 2选1 IP core |  | | （d）Block Design 4选1 IP core |  |   为了便于比较，作图如下。     1. Total power：verilog case的方式显著高于其他三者，功率较大。使用2选1的方式稍微高于另外两者。 2. Junction（结温）：verilog case的方式显著高于其他三者，此种方式产生了较高温度。使用2选1的方式稍高于另外两者。 3. Therma margin：verilog case的方式显著高于其他，说明此种方式对环境要求较高。        1. 通过此表可以看出，通过block design使用2选1实现，logic power显著大于其他三者，其在逻辑资源的使用上功率较大。 2. 而case的方式虽然总体功率较大，但是在logic上并无差异，这里体现在signals和IO上。 3. 检查 4. 我们对这样的结果表示怀疑，因此我们对几种方式的配置细节做了梳理，发现导致verilog case方式 IO power较高的原因并非是设计方式造成的，而是因为其IO standard与其他情况不一致（使用这种方式在FPGA开发板进行测试的，因此修改了对应的标准， 而其他方式使用的默认标准）。将其他方式的IO standard从LVCMO18改为LACMO33后，IO Power没有明显差异。   六、小组分工情况说明  本实验较为基础，因此每个人都独立的完成了整个实验的代码编写等，在此基础上做交流分析。   1. 田润泽，负责实验目的、内容，实验结果（分析不同实现方式的资源、功耗等异同），协同小组其他成员完善、补充相应的部分，整理、美化小组实验报告。 2. 尹宇慧，负责实验设计部分（包括原理图、真值表、电路图和仿真代码编写等）。 3. 姚语涵，负责实验过程和算法部分（包括代码编写等）。 4. 蔡嘉轩，负责实验结果部分（包括实验中各个环节的结果说明、开发板的操作演示等）。 | | | | | | |