**北京科技大学实验报告**

学院： 自动化学院 专业： 测控技术与仪器 班级： 测控192

姓名： 张正阳 学号： 41827166 实验日期： 2021年 12月 07 日

**实验名称： 实验一：组合逻辑电路设计**

**实验目的：**

1．熟悉 Vivado 2017.2 的编译环境；

2．了解在 Vivado 2017.2 环境下运用 Verilog HDL 语言的编程开发流程，包括源程序的输入、编译、模 拟仿真及程序下载

**实验仪器：**

FPGA开发板、Vivado2017.2开发工具

**实验内容与步骤：**

1. 简单门电路的实现；

编写代码实现一个简单的非门电路：

电路逻辑表达式

具体操作步骤如下：

(1) 利用向导，建立一个新项目，工程名为 expe1，顶层文件名也指定为 not\_lab。

(2) 新建一个 Verilog HDL 文件。

(3) 编写testbench文件，进行时序仿真。

(4) 给出时序图

在 Verilog HDL 文件中输入程序：

`timescale 1ns / 1ps

1. module not\_lab(
2. input in,
3. output out
4. );
5. assign out = ~in;
6. endmodule

编写testbench程序，如下：

`timescale 1ns / 1ps

1. module not\_lab\_tb();
2. // testbench 时钟信号
3. reg clk = 0;
4. always #10 clk <= ~clk;
5. // 输出信号
6. wire out;
7. // 调用test模块
8. Not\_lab mytest(clk, out);
9. endmodule

1. 四选一数据选择器；

四选一数据选择器的实现：数据选择器又被称为多路选择器或多路开关，它是常用组合逻辑部件之一。数据选择器的功能是在选择控制端的控制下，从多个数据输入中选择一个并将其送到一个公共的输出端。四选一数据选择器输出的逻辑函数如下：

数据选择器的电路结构一般由与或门阵列组成，也有用传输门开关和门电路混合而成的。四选一数据选择器电路模块含有四个数据输入端，一个数据输出端和两个地址选择输入端构成的。

具体操作步骤如下：

(1) 利用向导，建立一个新项目，工程名为 expe2，顶层文件名也指定为 mux4\_1。

(2) 新建一个 Verilog HDL 文件。

(3) 编写testbench文件，进行时序仿真。

(4) 给出时序图

在 Verilog HDL 文件中输入四选一数据选择器程序：

`timescale 1ns / 1ps

module mux4\_1(out,in0,in1,in2,in3,sel);

input in0,in1,in2,in3;

input [1:0] sel;

output out;

reg out;

always @(sel or in0 or in1 or in2 or in3)

begin

case(sel)

2'b00: out=in0;// 若地址选择输入为00， 则out输出为in0

2'b01: out=in1;

2'b10: out=in2;

2'b11: out=in3;

default: out=1'b0;//确保不存在未赋值的分支

endcase

end

endmodule

编写testbench文件:

`timescale 1ns / 1ps

module mux4\_1\_tb();

reg[3:0] in;

reg[1:0] sel\_in;

wire f;

mux4\_1 mux4\_1(f,in[0],in[1],in[2],in[3],sel\_in);//实例化待测模块

always #150 sel\_in=sel\_in+1;

always #50 in=in+1;

initial begin

in=0; sel\_in=0;

#1200 $finish;

end

endmodule

1. 全加器的实现

全加器其实就是考虑到进位的加法器。一位全加器的电路符号如图1.1 所示，真值表如表1.1 所示。

|  |  |  |
| --- | --- | --- |
| A | 全 加 器 | **Si** |
| B |
| Cin | Cout |
|  |
|  |

图1.1 全加器电路符号

表1.1 一位全加器真值表

全加器输入

全加器输出

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
| **A** | **B** | **Cin** | **Si** | **Cout** |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 |

源程序如下：（四位全加器）

`timescale 1ns / 1ps

module full\_adder(cout, si,ai,bi,cin);

output si;

output cout;

input ai,bi;

input cin;

assign{cout,si}=ai+bi+cin;

endmodule

testbench程序如下：

`timescale 1ns / 1ps

module full\_adder\_tb( );

reg [2:0] in;

wire sum;

wire cout;

full\_adder FA(.cout(cout),.si(sum),.ai(in[2]),.bi(in[1]),.cin(in[0]));//实例化待测模块

always #10 in=in+1;

initial begin

in=0;

#160 $finish;

end

endmodule

1. 编码器的实现

编码器通常分为两大类：普通编码器和优先编码器。其中普通编码其就是对某一个给定时刻只能对 一个输入信号进行编码的编码器，它的输入端口不允许同一时刻出现两个以上的有效输入信号；优先编 码器就是对某一个给定时刻只对优先级最高的输入信号进行编码的编码器，它的输入端口允许多个输入 信号同时有效。

现以八至三线编码器为例，介绍普通编码器的 Verilog HDL 语言程序设计。通常，八至三线编码器 的逻辑电路符号如图 1.2 所示，真值表如表1.2 所示。不难看出该编码器的工作原理为：编码器将对八个输入信号进行编码操作，然后以三位二进制码的形式输出，这里输入信号为低电平有效

D0

|  |  |  |
| --- | --- | --- |
|  | 8-3 线  编码器 | Q0 |
| D1 |
| D2 |
| Q1 |
| D3 |
| D4 |
| Q2 |
| D5 |
| D6 |
|  |
| D7 |
|  |

图 1.2 八至三线编码器的电路符号

**表 1.2 八至三线编码器的真值表**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 | Q2 | Q1 | Q0 |
| 0  1  1  1  1  1  1  1 | 1  0  1  1  1  1  1  1 | 1  1  0  1  1  1  1  1 | 1  1  1  0  1  1  1  1 | 1  1  1  1  0  1  1  1 | 1  1  1  1  1  0  1  1 | 1  1  1  1  1  1  0  1 | 1  1  1  1  1  1  1  0 | 1  1  1  1  0  0  0  0 | 1  1  0  0  1  1  0  0 | 1  0  1  0  1  0  1  0 |

具体操作过程如下：

（1） 利用项目向导，建立一个新项目，顶层文件名指定为 encoder8\_3。

（2） 新建一个 Verilog HDL 文件，并输入源程序：

module encoder8\_3(q,d);

input[7:0] d;

output[2:0] q;

reg[2:0] q;

always@(d) begin

case(d)

8'b01111111: q<=3'b111;

8'b10111111: q<=3'b110;

8'b11011111: q<=3'b101;

8'b11101111: q<=3'b100;

8'b11110111: q<=3'b011;

8'b11111011: q<=3'b010;

8'b11111101: q<=3'b001;

8'b11111110: q<=3'b000;

default: q<=3'bzzz;

endcase end

endmodule

a． 对源程序进行语法检查并编译。

b． 对项目进行时序逻辑仿真。

c． 有兴趣的同学还可以尝试分配管脚，生成\*.bit 文件。（管脚分配可参照实如下表）

d． 下载。

**实验数据：**

1. 简单门电路的实现

电脑萤幕画面

中度可信度描述已自动生成

电子仪器

中度可信度描述已自动生成

1. 四选一数据选择器；

图形用户界面

描述已自动生成

电子零件

中度可信度描述已自动生成

1. 全加器的实现

电脑萤幕画面

描述已自动生成

电子零件

低可信度描述已自动生成

1. 编码器的实现

电脑萤幕画面

描述已自动生成

电子零件

低可信度描述已自动生成

**实验名称：时序逻辑电路设计**

**实验目的：**

1．理解触发器和计数器时序器件的概念；

2．掌握触发器和计数器时序器件的Verilog HDL语言程序设计的方法

3. 比较组合逻辑器件和时序逻辑器件的Verilog HDL语言程序设计不同。

**实验仪器：**

FPGA开发板、Vivado2017.2开发工具

**实验内容与步骤：**

1. 触发器（D型）；

D 触发器的实现 在各种复杂的数字电路中，不但需要对输入信号进行算术运算和逻辑运算，还经常需要将这些信号和运算结果保存起来。因此，需要使用具有记忆功能的基本逻辑单元，能够存储一位信号的基本单元电路就 被称为触发器。根据电路结构形式和 控制方式的不同，可以将触发器分为 D 触发器、JK 触发器、T 触发器等等。这里只介绍常用的Ｄ型触发器，其他类型触发器请有兴趣的同学自己实现。

在数字电路中，D 触发器是最为简单也是最为常用的一种基本时序逻辑电路，它是构成数字电路系统的 基础。大体可分为如下几类：基本的 D 触发器；同步复位的 D 触发器；异步复位的 D 触发器；同步置位/ 复位的 D 触发器；异步置位/复位的 D 触发器。

下面先分别介绍各个 D 触发器的具体工作原理，然后再介绍具体操作步骤。

（1）基本的 D 触发器

在数字电路中，一个基本的上升沿 D 触发器的逻辑电路符号如图 2.1 所示，其功能表如表 2.1 所示。 根据下面的电路符号和功能表不难看出，一个基本的 D 触发器的工作原理为：当时中信号的上升沿到来时，输入端口 D 的数据将传递给输出端口 Q 和输出端口 Q。在此，输出端口 Q 和输出端口 Q 除了反相之 外，其他特性都是相同的。

表 2.1 D 触发器的功能表

|  |  |  |
| --- | --- | --- |
|  | D Q  CP Q | q |
| clk | qb |
|  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| D | CP | Q | Q |
| X | 0 | 保持 | 保持 |
| X | 1 | 保持 | 保持 |
| 0 | 上升沿 | 0 | 1 |
| 1 | 上升沿 | 1 | 0 |

d

图 2.1 电路符号

a．利用向导，建立一个新项目，工程名为 expe3，顶层文件名指定为 D；

b．新建一个 Verilog HDL 文件，并输入源程序：

module D(

input d,

input clk,

output reg q,

output reg qd

);

always@(posedge clk) begin

q <= d; qd <= ~d; end

endmodle

c． 对源程序进行语法检查和编译；

d．编写testbench程序，进行时序仿真；

（2） 同步复位的 D 触发器

在数字电路中，一种常见的带有同步复位控制端口的上升沿 D 触发器的逻辑电路符号如图 2.3 所示，它 的功能表如表 2.2 所示。不难看出，只有在时钟信号的上升沿到来并且复位控制端口的信号有效时，D 触发 器才进行复位操作，即将输出端口 Q ~~的~~值置为逻辑 0，而把输出端口 Q 的值置为逻辑 1。

表2.2 D 触发器的功能表

d

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| R | D | CP | Q | Q |
| 0 | X | 上升沿 | 0 | 1 |
| 1 | X | 0 | 保持 | 保持 |
| 1 | X | 1 | 保持 | 保持 |
| 1 | 0 | 上升沿 | 0 | 1 |
| 1 | 1 | 上升沿 | 1 | 0 |

|  |  |  |
| --- | --- | --- |
|  | D Q  CP  Q  R | q |
| clk | qb |
| reset |
|  |
|  |

图2.3 D 触发器电路符号

源程序如下：

module sync\_rddf(clk,reset,d,q,qb);

input clk,reset,d;

output q,qb;

reg q,qb;

always @(posedge clk) begin

if(!reset) begin

q<=0;

qb<=1;

end

else begin

q<=d;

qb<=~d;

end

end endmodule

（3） 异步复位的 D 触发器

常见的带有异步复位控制端口的上升沿 D 触发器的逻辑电路符号如图 2.4 所示，它的功能表如表2.3 所示。不难看出，只要复位控制端口的信号有效，D 触发器就会立即进行复位操作。可见，这时的复位操作 是与时钟信号无关的。

表 2.3 D 触发器的功能表

d

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| R | D | CP | Q | Q |
| 0 | X | X | 0 | 1 |
| 1 | X | 0 | 保持 | 保持 |
| 1 | X | 1 | 保持 | 保持 |
| 1 | 0 | 上升沿 | 0 | 1 |
| 1 | 1 | 上升沿 | 1 | 0 |

|  |  |  |
| --- | --- | --- |
|  | D Q  CP  Q  R | q |
| clk | qb |
| reset |
|  |
|  |

图 2.4 D 触发器电路符号

**源程序如下：**

module async\_rddf(clk,reset,d,q,qb);

input clk,reset,d;

output q,qb;

reg q,qb;

always @(posedge clk or negedge reset) begin

if(!reset) begin

q<=0;

qb<=1;

end

else begin

q<=d;

qb<=~d;

end

end endmodule

（4） 同步置位/复位的 D 触发器

同时带有置位控制和复位控制端口的 D 触发器也是经常使用的，同样它也具有同步异步两种方式。这 里我们给出同步置位/复位的 D 触发器的源程序及仿真结果，请读者根据已经介绍的内容自己实现异步置位/复位的 D 触发器。

带有同步置位/复位端口的上升沿 D 触发器的逻辑电路符号如图 2.5 所示，它的功能表如表2.4 所示。 不难看出，只有在时钟信号的上升沿到来并且同步置位/复位端口的信号有效时，D 触发器才可以进行置位 或者复位操作。

表 2.4 D 触发器的功能表

S

D Q

CP Q R

set

d

q

clk

qb

reset

图 2.5 电路符号

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| S | R | D | CP | Q | Q |
| 0 | 1 | X | 上升沿 | 1 | 0 |
| 1 | 0 | X | 上升沿 | 0 | 1 |
| 1 | 1 | X | 0 | 保持 | 保持 |
| 1 | 1 | 0 | 上升沿 | 0 | 1 |
| 1 | 1 | 1 | 上升沿 | 1 | 0 |

**源程序如下：**

module sync\_rsddf(clk,reset,set,d,q,qb);

input clk,reset,set;

input d;

output q,qb;

reg q,qb;

always @(posedge clk) begin

if(!set && reset) begin

q<=1;

qb<=0;

end

else if(set && !reset) begin

q<=0;

qb<=1;

end

else begin

q<=d;

qb<=~d;

end

end endmodule

1. 计数器（递增、递减）；

（1） 加法计数器

加法计数器的动作是，每次时钟脉冲信号 clk 为上升沿时，计数器会将计数值加 1。以图2.6 为例，它 是 2bits 的计数器，所以计数值（由 Q1Q0 组成），依次是 0，1，2，3，0，1„，周而复始。

在图2.6 的波形图里，透露了这样几个信息：

i. 一个两 bit 计数器，它所能计数的范围是 0~3（22-1）。同理，n bits 的计数器所能计数的范围是 0~2n-1。

ii. 分别由 Q0、Q1 得到的波形频率是时钟脉冲信号 clk 的 1/2、1/4，亦即是将时钟脉冲信号的 clk 频率除 2、除 4，因此图2.6 又常被称为除 4 计数器。

iii. 由上讨论推广可知，n bits 计数器可获得的信号之多是频率除 2n 的结果。

Q1

CP 加法计数器

Q2

C

Q1 Q2

合计 0 1 2 3 0

图2.6 加法计数器的相关波形

**源程序如下：**

module addcounter(clk,Q) ; input clk ;

output[3 :0] Q ; reg[3 :0] Q ;

initial begin Q = 4'h0; end

always @(posedge clk)begin Q<=Q+1 ;

end endmodule

**实验数据：**

1(1)

Testbench 文件如下：

`timescale 1ns / 1ps

module test();

reg clk = 0;

reg d = 0;

always #10 clk = ~clk;

always #20 d = ~d;

wire q;

wire qb;

D aa(d,clk ,q,qb);

endmodule

电脑萤幕画面

描述已自动生成

电子仪器

低可信度描述已自动生成

1(2)

module test();

reg clk = 0;

reg d = 0;

reg reset = 0;

always #50 reset <= ~reset;

always #10 clk <= ~clk;

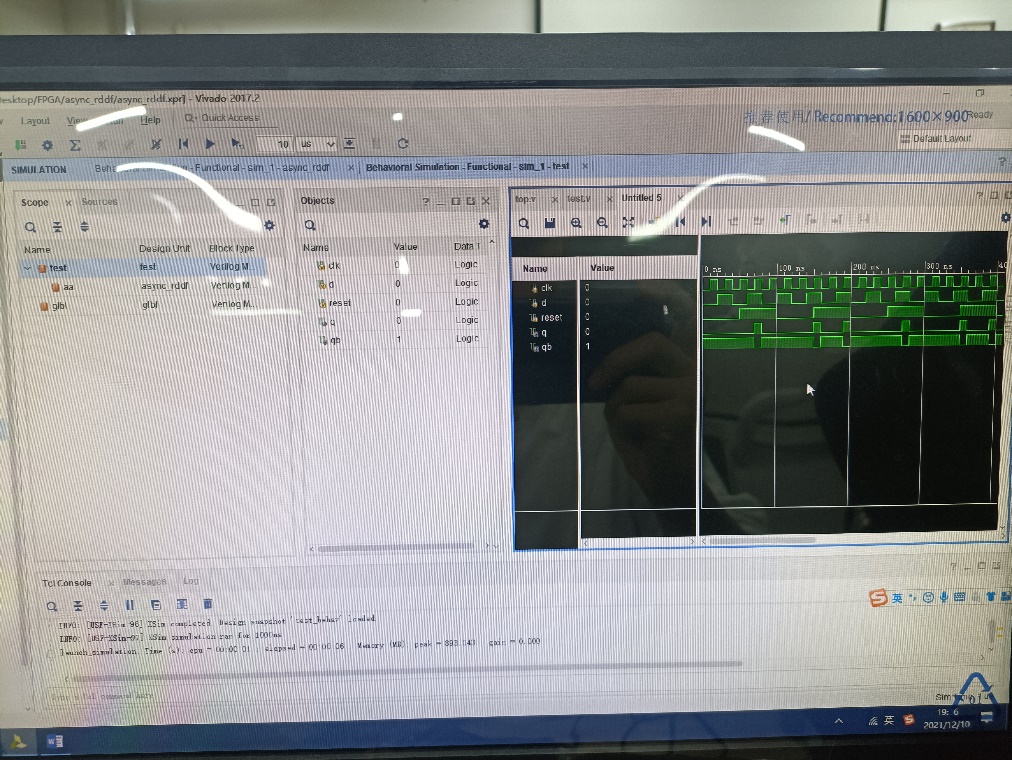
always #20 d <= ~d;

wire q;

wire qb;

sync\_rddf aa(clk,reset,d,q,qb);

endmodule



电子仪器

中度可信度描述已自动生成

1(3)

module test();

reg clk = 0;

reg d = 0;

reg reset = 0;

always #50 reset <= ~reset;

always #10 clk <= ~clk;

always #20 d <= ~d;

wire q;

wire qb;

async\_rddf aa(clk,d,reset,q,qb);

endmodule

电脑萤幕画面

低可信度描述已自动生成

电子仪器

中度可信度描述已自动生成

1(4)

module t();

reg clk = 0;

reg reset = 0;

reg set = 0;

reg d = 0;

always #5 clk <= ~clk;

always #30 reset <= ~reset;

always #20 set <= ~set;

always #10 d <= ~d;

wire q;

wire qb;

sync\_rsddf aa(clk,reset,set,d,q,qb);

endmodule

电脑萤幕画面

低可信度描述已自动生成

电子零件

低可信度描述已自动生成

2(1)

图片包含 图形用户界面

描述已自动生成

电子零件

低可信度描述已自动生成