【实验目的】

掌握 Verilog HDL 常用语法

能够熟练阅读并理解 Verilog 代码

能够设计较复杂的数字功能电路

能够将 Verilog 代码与实际硬件相对应

【实验环境】

PC 一台

macOS操作系统

Java 运行环境(jre)

Logisim 仿真工具

vlab.ustc.edu.cn

【实验过程】

Step1:Verilog 关键字：

```
module test(
    input wire   a,b,clk,
    output reg   o);

wire   s;
   //code
endmodule
```

module/endmodule:这两个关键字用于表示模块的开始和结束，必须成对出现

input:表明端口类型为输入信号，该关键字一般用在模块的端口定义部分。

output:表明端口类型为输出信号，该关键字一般用在模块的端口定义部分。

wire:表明数据类型为线型，该关键字用在端口或内部信号的定义部分。

reg:表明数据类型为寄存器类型，这是与 wire 相对应的一种数据类型，该关键字用在端口或内部信号的定义部分。

```
module test(
    input wire   a,b,
    output reg   o);

wire   s;
assign s = a & b;
always @( * )
    o = s;// *表示任意时序控制
endmodule
```

assign:连续赋值语句关键字，此类语句将逻辑表达式的值赋给线网类型信号。

always:过程赋值语句关键字，次类语句将逻辑表达式的值赋给寄存器类型信号。

```
//if(条件) 过程语句
//else 过程语句
module test(
    input wire a,b,clk,
    output reg o);

always @( posedge clk )
begin
    if(a)
        o <= a;
    else
```

```
        o <= b;
end
endmodule
```

begin/end:顺序过程语句，必须成对出现，在 begin/end 中间可以有多条语句，这些语句是顺序执行的，该关键字允许嵌套，即 begin/end 内部可以包含其他的 begin/end 语句块。

posedge:该关键字后面跟信号名，表示该信号的上升沿这一事件。一般用在 always 语句的时序控制部分。

negedge:该关键字后面跟信号名，表示该信号的下降沿这一事件。一般用在 always 语句的时序控制部分。

if:条件语句，其后跟条件判别语句，以及过程语句，有时候会与 else 语句配合使用。

else: if 语句的分支，if、else 用于实现带有优先级的条件分支，一般出现在 always 语句的过程语句部分，而不能直接在模块内部单独出现。

```
module test(
    input wire   a,b,clk,
    output reg   o);
always @( posedge clk )
    case ({a,b})
        2'b00:o <= 1'b0;
        2'b01:o <= 1'b0;
        2'b10:o <= 1'b0;
        2'b11:o <= 1'b1;
    endcase
endmodule
```

case/endcase:具有相同优先级的多路条件分支，两个关键字必须成对出现。一般出现在 always 的过程语句部分，而不能在模块内部单

独出现。

Step2:注意：调用模块输出端要求为 wire 型！

Step3: 在 Verilog 中，有四种基本的值:0,1,x,z ；Verilog 中的常量有三种:整数、实数、字符串 ；Verilog 中有两种常用的数据类型，即 wire(线网类型)和 reg (寄存器类型)，关于两种数据类型的使用只需要遵循以下规则即可: 凡是通过 assign 语句赋值的信号(一定是组合逻辑赋值信号)，都 应定义成 wire 类型，凡是在 always 语句中赋值的信号(可能是组合 逻辑赋值信号、也可能是时序逻辑赋值信号)，都应定义成 reg 类型

Step4: Verilog 语法中的操作符，按照其功能可以分为:**算数操作符、关系操作符、逻辑操作符、归约操作符、条件操作符、移位操作符、拼接操作符。**

Step5: Verilog 中的表达式由操作数和操作符组成，可以在出现数值的任何地方使用。

Step6:模块调用：

一个模块能够在另一个模块中被引用，这样就建立了描述的层次， 使得设计大规模复杂电路的效率大大提高。模块实例语句的

形式为：模块名 实例化名(端口关联)；

端口信号可以通过位置或名称进行关联，但两种关联方式不能混 用。通过位置关联的格式为:端口表达式，通过名称关联的格式为:. 端口名称(端口表达式)

下面以全加器为例：

```
module add(
    input a,b,
    output sum,cout);

                        //模块主体
endmodule


module full_add(
    input a,b,cin,
    output sum,cout);

wire s,carry1,carry2;
add add_inst1(a,b,s,carry1);//通过位置关联
add add_inst2(.a(s),.b(cin),.sum(sum),.cout(cout));//通过名称关联
endmodule
```

Step7:代码实例

```
//8bit 位宽 4 选 1 选择器，纯组合逻辑
module mux_4to1(    //8bit 位宽的 4 选 1 选择器
    input [7:0] a,b,c,d,
    input [1:0] sel,
    output reg [7:0] o);//always 语句内赋值的信号都应定义成 reg 类型

always @(*)  //always 语句内实现组合逻辑
begin
    case (sel)
        2'b00: o = a;//组合逻辑使用"="进行赋值
        2'b01: o = b;
        2'b10: o = c;
        2'b11: o = d;
        default: o = 8'h0;//用 case 语句实现组合逻辑时一定要有 default
    endcase
```

```
end
endmodule
```

```
//1~10 循环计数的计数器
module cnt_1to10(
    input clk,rst_n,
    output reg [3:0] cnt);

always @(posedge clk or negedge rst_n) //时序控制条件为时钟上升沿和复位的下降沿
begin
    if(!rst_n)//复位信号优先级最高，应是第一个判断的条件
        cnt <= 4'h1;
    else if(cnt>=10)
        cnt <= 4'h1;
    else
        cnt <= cnt + 4'h1;
end
endmodule
```

【实验练习】

题目 1：

错误：if/else 不能在模块内部单独出现，而应出现在 always 语句的过程语句部分；b 在 always 语句中被赋值，应定义为 reg 类型

改正后如下：

```
module test(
    input a,
    output reg b);

always @(*)
begin
    if(a)
        b = 1'b0;
    else
        b = 1'b1;
end
endmodule
```

题目 2:

补充如下：

（当 begin 与 end 之间仅有一条语句时，可省略）

```
module test(
    input [4:0] a,
    output reg [4:0]b);

always @(*)
    b = a;
endmodule
```

题目 3:

结果写在注释中：（其中最后一问计算 k=a-b 时，a 向前借位（可理解为溢出））

```
module test(
    input  [7:0] a,b,
    output [7:0] c,d,e,f,g,h,i,j,k );

assign c = a & b;   //按位与
assign d = a | b;   //按位或
assign e = a ^ b;   //按位异或
assign f = ~a;      //按位取反
assign g = {a[3:0],b[3:0]};   //位拼接
assign h = a >> 3;    //右移
assign i = &b;    //缩位与
assign j = (a > b) ? a : b;
assign k = a - b;
endmodule

//a = 8'b0011_0011,
//b = 8'b1111_0000
//c = 8'b0011_0000
//d = 8'b1111_0011
//e = 8'b1100_0011
//f = 8'b1100_1100
```

```
//g = 8'b0011_0000
//h = 8'b0000_0110
//i = 8'b0000_0000
//j = 8'b1111_0000
//k = 8‘b0100_0011
```

题目 4:

错误：

输出信号 c 定义为 reg 型，应在 always 语句中赋值；调用模块输出端要求为 wire 型；端口信号的两种关联方式不能混用;要为调用的模块起与模块名不同的实例化名。

```
module sub_test(
    input a,b,
    output reg c);

always @(*)
    c = (a<b)? a : b;
endmodule


module test(
    input a,b,c,
    output o);

wire temp;
sub_test sub_test_1(.a(a),.b(b),.c(temp));
sub_test sub_test_2(.a(temp),.b(c),.c(o));
endmodule
```

题目 5:

错误：output 信号应写在 module test()的括号内；实例化名不能与模块名重名。

改正后代码如下：

```
module sub_test(
    input  a,b,
    output o);

assign o = a + b;
endmodule


module test(
    input a,b,
    output c);

always @(*)
begin
    sub_test sub_test_1(a,b,c);
end
endmodule
```

【总结与思考】

1. （1）实验带领我们详细回顾了 Verilog HDL 中的关键字，基本结构，数据及类型，操作符，表达式等知识。

（2）实验讲义中介绍了“模块调用”的相关知识，在简单的运用后了解了基本的规则。

2. 较为简单

3. 适中

4. 实验题目中可以加入“case/endcase”语句的相关练习