上海交通大学网络空间安全学院

数字系统设计作业

2021 年秋季学期

说明：

这一部分中的练习题，同学们可参照例子进行练习，**不作为作业提交**。

第一部分、*Verilog Step by Step*

设计指导：

根据功能需求和设计要求，理解示例模块中每一条语句的作用，然后，对实例模块进行仿真，参照示例中的模块设计，完成每一阶段规定的练习。

1.1、简单的组合逻辑设计

设计内容：1 位数据比较器；
学习目的：基本组合逻辑电路的设计；
功能定义：比较输入数据 **a** 与 **b**，如果两个数据相同，则输出 **1**，否则输出 **0**；

设计提示：在 Verilog HDL 中，描述简单组合逻辑时，通常使用 assign 语句。

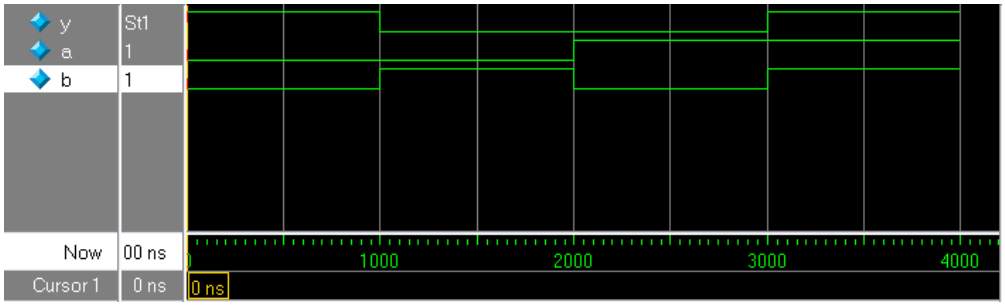
（1）1bit 比较器模块源代码：

```
1   // File: compare1b.v
2   `timescale 10 ns / 1 ns
3
4   module compare1b( y ,b ,a );
5
6       output y;    // 定义输出端口
7       input b, a; // 定义输入端口
8
9       // 实现比较器组合逻辑
10      assign  y = (a==b) ? 1 : 0;
11
12  endmodule
```

（2）仿真测试模块源代码：

```
1    `timescale 10ns / 1ns       // 定义时间单位
2
3    `include  "compare1b.v"     // 包含被仿真模块的文件
4
5    module tb_compare1b;            // 无输入输出端口列表
6
7        wire p_y;               // 连接被测模块的输出端口
8        reg  p_a, p_b;          // 连接被测模块的输入端口
9
10       // 模块调用实例，调用被仿真测试的模块
11       compare1b  m_cmp1b(.y(p_y), .b(p_b), .a(p_a) );
12
13       initial                 // 使用 initial 过程语句块产生仿真激励信号
14       begin
15           p_a = 0; p_b = 0;
16           #100 p_a = 0; p_b = 1;
17           #100 p_a = 1; p_b = 0;
18           #100 p_a = 1; p_b = 1;
19           #100 $stop;          // 系统任务，暂停仿真以便观察仿真波形
20       end
21
22       initial
23       begin    // 监控并显示被仿真模块的输入/输出
24           $monitor( "当前时间: %tns,", $time, "<----> y=%b, b=%b, a=%b", p_y, p_b, p_a);
25       end
26
27   endmodule
```

（3）仿真波形：



（4）控制台输出：

```
VSIM 5> run
# 当前时间:          0ns,<----> y=1, b=0, a=0
# 当前时间:       1000ns,<----> y=0, b=1, a=0
# 当前时间:       2000ns,<----> y=0, b=0, a=1
# 当前时间:       3000ns,<----> y=1, b=1, a=1
```

（5）练习：

设计一个字节（8 位）比较器。

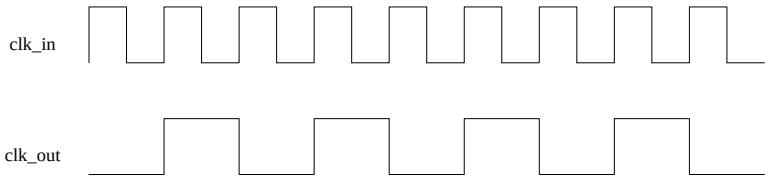
要求：比较两个字节的大小，当 **a[7:0]** 大于 **b[7:0]** 输出高电平，否则，输出低电平，编写仿真测试模块，对其功能进行完整、全面的测试。

1.2、简单时序逻辑电路的设计

设计内容：二倍分频电路的设计；

学习目的：设计基本时序逻辑电路；

功能定义：对输入的时钟信号进行分频，获得周期为输入时钟信号周期的二倍的时钟信号。要求采用同步复位，输入时钟信号 **clk_in** 和输出时钟信号 **clk_out** 见下图：



设计提示：通常使用 **always** 过程语句块和 **@(posedge clk)** 或 **@(negedge clk)** 的结构设计时序逻辑电路。

在 **always** 过程语句块中，被赋值的信号都必须定义为 **reg** 型，**reg** 型数据的缺省值为不定态 **x**。

（1）二倍分频电路模块源码：

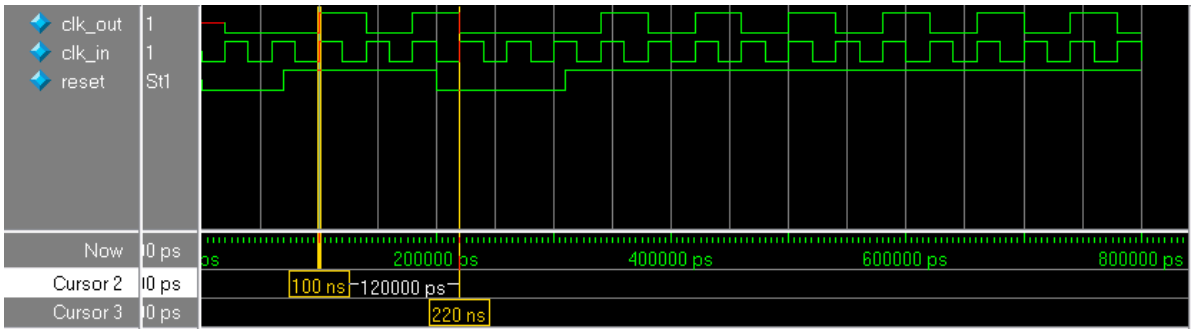
```
1
2   // File: freq_div2.v
3
4   `timescale 1ns / 100ps
5
6   module freq_div2( output reg clk_out,  // 端口列表和声明
7                     input clk_in,
8                     input reset );
9
10  always @( posedge clk_in )      // 输入时钟信号正跳沿触发
11  begin
12      if(!reset) clk_out <= 0;    // 同步复位，复用信号低电平有效
13      else clk_out <= ~clk_out;   // 使用非阻塞性赋值
14  end
15
16  endmodule
```

（2）测试模块的源代码：

```
2   // File: tb_freq_div2.v
3
4   `timescale 1ns / 100ps
5   `define CLOCK_CYCLE 20
6
7   module tb_freq_div2;
8
9       reg p_clk_in, p_rst; // 连接被测模块的输入端口
10
11      wire p_clk_out;      // 连接被测模块的输出端口
12
13      // 被测试模块调用实例
14      freq_div2 m_freq_div2( .clk_out(p_clk_out),
15                             .clk_in(p_clk_in),
16                             .reset(p_rst));
17
18      // 产生输入时钟信号
19      initial
20      begin
21          p_clk_in = 1'b0;
22          forever #`CLOCK_CYCLE p_clk_in = ~p_clk_in;
23      end
```

```
24
25          // 产生复位信号
26          initial
27          begin
28              p_rst = 1'b0;
29
30              #70 p_rst = 1'b1;
31              #130 p_rst = 1'b0;
32              #110 p_rst = 1;
33              #10000 $stop;    // 系统任务，暂停仿真
34          end
35      endmodule
```

（3）仿真波形：



（4）练习：

a) 思考与分析：

参考 Modelsim 波形说明，并参见示例仿真波形图：

i）在输入时钟信号 clk_in 的第一个上升沿到来之前，输出时钟信号的波形呈红色的含义是什么？

ii）尽管复位信号已经变化，为何输出时钟 clk_out 在仿真时刻 100ns 才出现正跳沿？

iii）尽管复位信号已为低电平，为何输出时钟 clk_out 在仿真时刻 220ns 才开始复位？

b) 设计：设计输入时钟 clk_in 的二倍分频电路模块，输出信号为 clk_out，要求输出 clk_out 与示例模块的输出波形正好反相。编写仿真测试模块，对其功能进行测试，并给出仿真波形。

1.3 时序逻辑电路设计

设计内容：分频电路设计。将 10M 的时钟分频为 500K 的时钟；

学习目的：使用条件语句 **if…else** 描述较为复杂的时序关系；

功能定义：对输入的时钟信号进行分频，将 10M 的时钟分频为 500K 的时钟；要求采用同步复位，输入时钟信号 **clk_in** 和输出时钟信号 **clk_out** 见下图：

设计提示：使用计数器。

（1）分频电路模块源代码：

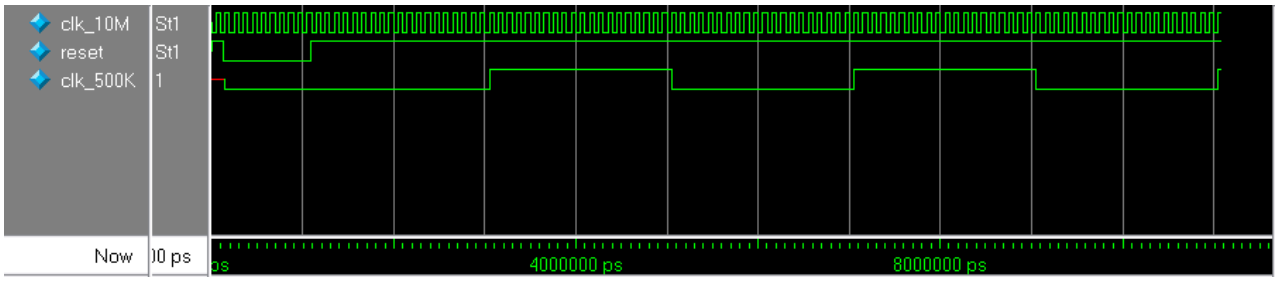
```
2   // File: freq_div20.v
3
4   module freq_div20( output reg clk_500K,
5                      input clk_10M,
6                      input reset);
7
8   // 定义计数器变量: 19 = 5'b1_0011
9   // 使用位宽为 5 的 reg 型变量保存计数值
10  reg [4:0] cnt;
11
12  always @( posedge clk_10M )
13  begin
14      if( !reset )         // 同步复位，低电平有效
15      begin
16          clk_500K <= 1'b0;
17          cnt <= 5'b0;
18      end
19      else
20      begin
21          // 根据计数器的值进行判断，以确定信号 clk_500K 信号是否反转
22          if( cnt == 5'd19)
23          begin
24              cnt <= 5'b0;
25              clk_500K <= ~clk_500K;
26          end
27          else
28            cnt <= cnt + 1'b1;
29      end
30  end // EOF always
31
32  endmodule
```

（2）测试模块的源代码：

```
6   `define CLOCK_CYCLE 50
7
8   `include "freq_div20.v"
9
10  // 仿真测试模块，没有输入/输出端口列表
11  module tb_freq_div20;
12
13      // 使用 reg 型变量连接被测试模块的输入端口
14      reg p_clk_10M;
15      reg p_rst;
16
17      // 使用 wire 型变量连接被测试模块的输出端口
18      wire p_clk_500K;
19
20      // 调用被仿真测试模块实例
21      freq_div20 m_fd20( .clk_500K(p_clk_500k),
22                         .clk_10M(p_clk_10M),
23                         .reset(p_rst));
```

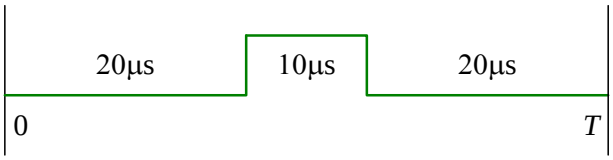
```
24
25          // 产生 10MHz 输入时钟脉冲
26          initial
27          begin
28              p_clk_10M = 1'b0;
29
30              // 时钟信号每 50ns 翻转一次
31              forever
32              begin
33                  #`CLOCK_CYCLE;
34                  p_clk_10M = ~p_clk_10M;
35              end
36          end
37
38          // 产生复位信号
39          initial
40          begin
41              p_rst = 1'b1;
42
43              #125 p_rst = 1'b0;
44              #960 p_rst = 1'b1;
45
46              // 仿真器运行 10000 个时间单位（这里为：ns）后，
47              // 调用系统任务暂停仿真
48              #10000 $stop;
49          end
50
51      endmodule
```

（3）仿真波形：



（4）练习：

利用 10M 的时钟，设计一个单周期形状如下的周期波形。



1.4 阻塞赋值与非阻塞赋值

设计内容：阻塞赋值模块、非阻塞赋值模块；

学习目的：掌握阻塞赋值与非阻塞赋值的概念和区别；

（1）阻塞赋值模块源代码：

```
2   // File: blocking.v
3
4   `timescale 1ns / 1ns
5
6   module blocking( c, b, a, clk);
7
8       output reg [3:0] c, b;
9
10      input [3:0] a;
11      input clk;
12
13      always @(posedge clk)
14      begin
15          b = a;     // Blocking Assignments
16          c = b;     // Blocking Assignments
17
18          $display("阻塞赋值: c = %d, b = %d, a = %d.", c, b, a);
19      end
20  endmodule
```

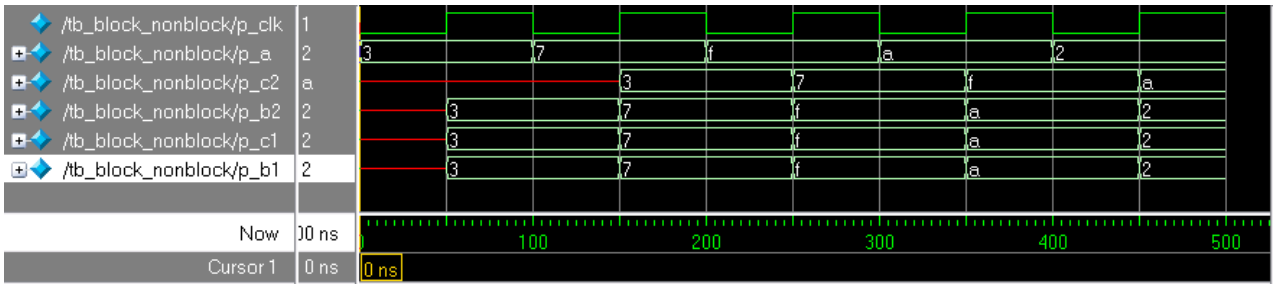
（2）非阻塞赋值模块源代码：

```
2   // File: nonblocking.v
3
4   `timescale 1ns / 1ns
5
6   module nonblocking(output reg [3:0] c,
7                      output reg [3:0] b,
8                      input [3:0] a,
9                      input clk);
10      always @(posedge clk)
11      begin
12          b <= a;
13          c <= b;
14
15          $display("非阻塞赋值: c = %d, b = %d, a = %d.", c, b, a);
16      end
17  endmodule
```

（3）测试模块源代码：

```
2   // File: tb_block_nonblock.v
3
4   `timescale 1ns / 1ns    // 时间单位: 1ns, 时间精度: 1ns
5
6   `include "blocking.v"
7   `include "nonblocking.v"
8
9   module tb_block_nonblock;
10
11      // 线网型变量, 连接输出两个被测试模块的输出端口
12      wire [3:0] p_c2, p_b2;
13      wire [3:0] p_c1, p_b1;
14
15      // reg 型变量, 连接输出两个被测试模块的输入端口, 施加激励信号
16      reg [3:0] p_a;
17      reg p_clk;
18
19      // 被测试模块调用实例
20      nonblocking m_nblk(.c(p_c2), .b(p_b2), .a(p_a), .clk(p_clk));
21
22      blocking m_blk(.c(p_c1), .b(p_b1), .a(p_a), .clk(p_clk));
23
24      initial
25      begin
26          p_clk = 0;
27          forever #50 p_clk = ~p_clk;
28      end
29
30      initial
31      begin
32          $display("At time: %tns", $time);
33          p_a = 4'h3;
34          #100 $display("At time: %tns", $time);
35          p_a = 4'h7;
36          #100 $display("At time: %tns", $time);
37          p_a = 4'hf;
38          #100 $display("At time: %tns", $time);
39          p_a = 4'ha;
40          #100 $display("At time: %tns", $time);
41          p_a = 4'h2;
42
43          #100 $stop; // 调用系统任务, 暂停仿真
44      end
45
46  endmodule
```

（4）仿真波形：



（5）仿真器控制台输出：

```
VSIM 23> run
# At time:          0ns
# 阻塞赋值: c = 3, b = 3, a = 3.
# 非阻塞赋值: c = x, b = x, a = 3.
# At time:          100ns
# 阻塞赋值: c = 7, b = 7, a = 7.
# 非阻塞赋值: c = x, b = 3, a = 7.
# At time:          200ns
# 阻塞赋值: c = 15, b = 15, a = 15.
# 非阻塞赋值: c = 3, b = 7, a = 15.
# At time:          300ns
# 阻塞赋值: c = 10, b = 10, a = 10.
# 非阻塞赋值: c = 7, b = 15, a = 10.
# At time:          400ns
# 阻塞赋值: c = 2, b = 2, a = 2.
# 非阻塞赋值: c = 15, b = 10, a = 2.
```

（6）思考题：

在 blocking 模块中按如下写法，仿真的结果会有什么样的变化？给出仿真波形。

```
2    ////////////////////////////////
3    // 1. 使用一个 always 过程语句块
4
5        always @(posedge clk)
6        begin
7            c = b;
8            b = a;
9        end
10
11   ////////////////////////////////
12
13
14   ////////////////////////////////
15   // 2. 使用两个 always 过程语句块
16
17   always @(posedge clk) b = a;
18
19   always @(posedge clk) c = b;
20
21   ////////////////////////////////
```

1.5 用 always 块设计比较复杂的组合逻辑电路

设计内容：复杂组合逻辑电路设计；

学习目的：掌握用 always 实现组合逻辑电路的方法；

功能定义：简单的指令译码电路。根据输入指令对输入数据执行相应的操作，包括：加、减、与、或和求反。无论指令还是数据发生变化，都将产生新的的结果。

设计提示：使用电平敏感的 always 过程块，与时序逻辑不同，在 @ 后的括号内没有边沿敏感关键词，如 posedge 或 negedge ，就能触发 always 块的动作，并且，采用 case 结构来进行分支判断。如果采用 assign 语句，表达起来非常复杂。

特别说明： 在设计组合逻辑电路时，在 always-case 结构中适当运用 default，在 always-if 结构中适当运用 else，通常可以综合为纯组合逻辑，尽管被赋值的变量一定要定义为 reg 型。不过，如果不使用 default 或 else 对缺省项进行说明，则有可能生成额外的、不该出现的锁存器。

（1）指令译码器模块源代码：

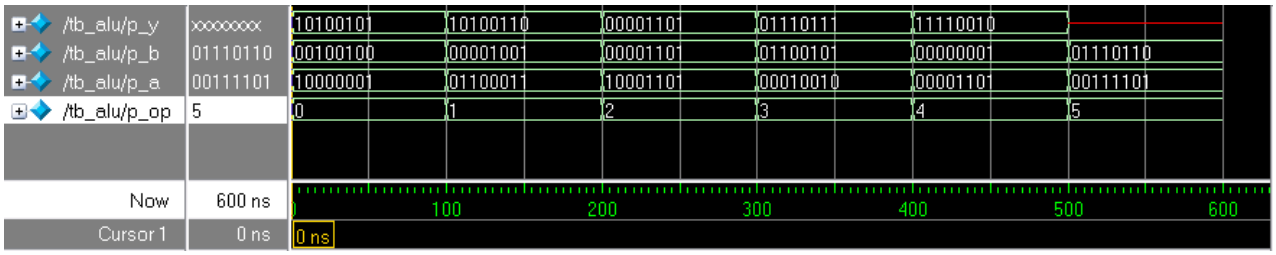
```
2   // File: alu.v
3
4   `timescale 1ns / 1ns
5
6   `define OP_PLUS  3'd0
7   `define OP_MINUS 3'd1
8   `define OP_AND   3'd2
9   `define OP_OR    3'd3
10  `define OP_NOT   3'd4
11
12  module alu( y, opcode, b, a);
13      output reg [7:0] y;
14
15      input [2:0] opcode;  // 操作码
16      input [7:0] b, a;    // 操作数
17
18  // 电平敏感的 always 过程语句块块, Verilog 1995 Std
19  //  always@(opcode or b or a)
20
21      always@(*)  // 电平敏感的 always 过程语句块, Verilog 2001 Std
22      begin
23          case( opcode )
24              `OP_PLUS:  y = b + a;   // 加
25              `OP_MINUS: y = b - a;   // 减
26              `OP_AND:   y = b & a;   // 与
27              `OP_OR:    y = b | a;   // 或
28              `OP_NOT:   y = ~a;      // 求反
29              default:   y = 8'hx;    // 未收到指令时, 输出任意态
30          endcase
31      end
32  endmodule
```

在设计组合逻辑电路时，在 always-case 结构中适当运用 default，在 always-if 结构中适当运用 else，通常可以综合为纯组合逻辑，尽管被赋值的变量一定要定义为 reg 型。不过，如果不使用 default 或 else 对缺省项进行说明，则有可能生成额外的、不该出现的锁存器。

```
2   // File: tb_alu.v
3
4   `timescale 1ns / 1ns
5
6   `include  "alu.v"
7
8   module tb_alu;
9
10      wire [7:0] p_y;
11
12      reg [7:0] p_b, p_a;
13      reg [2:0] p_op;
14
15      parameter NMOD = 256;
16
17      integer NUM;
18
19      alu m_alu(.y(p_y), .opcode(p_op), .b(p_b), .a(p_a) );
20
```

```
21        initial
22        begin
23
24            NUM = 5;
25
26            p_b = {$random} % NMOD; // Give a radom number blongs to [0,255].
27            p_a = {$random} % NMOD; // Give a radom number blongs to [0,255].
28
29            p_op = 3'h0;
30
31            repeat( NUM )
32            begin
33                #100;
34                p_b = {$random} % NMOD; //Give a radom number.
35                p_a = {$random} % NMOD; //Give a radom number.
36
37                p_op = p_op + 1'b1;
38            end
39
40            #100 $stop;
41        end
42    endmodule
```

（3）仿真波形：



（4）练习：

使用 always-case 结构设计一个八路数据选择器。

要求：每路输入数据与输出数据均为 4 位 2 进制数，当选择开关（至少 3 位）或输入数据发生变化时，输出数据也相应地变化。