数字系统设计课堂练习（2）2020_11_24 参考答案

一、基本语法

（1）当下面的 Verilog 代码片段执行完后，确定在不同仿真时刻 a, b 的值。

```
reg [9:0] a, b;
initial
begin
    #5 a = 10'h2ea;
    #5 b = { a[9], 3'b111, {2{a[9:7]}} };
end
```

（a）在仿真时刻 0： a = 10’bx b = 10’bx

（b）在仿真时刻 5： a = 10’b10_1110_1010 b = 10’bx

（c）在仿真时刻 10： a = 10’b10_1110_1010 b = 10’b11_1110_1101

（2）在下面的一个程序中，当 casex 执行完后，a 和 b 的值是多少？

a = 4’b0011 b = 4’b1010

```
module test();
    reg [3:0] a = 4'b0;
    reg [3:0] b = 4'b0;
    reg [3:0] state;
    initial  begin
        state = 4'b00xz;
        #1;
        casex(state)
            4'b10??, 4'b110?, 4'b1???: begin
                                            a = 4'b1100;  b = 4'b1010;
                                        end
            4'b01xx:   begin   a = 4’b0100;  b = 4'b0101;  end
            4'b00??:   begin   a = 4'b0011;  b = 4’b1010;  end
        endcase
    end
endmodule
```

（3）当下面的 Verilog 代码片段执行完后，确定在不同仿真时刻 a, b 的值。

```
reg [2:0] a;    reg [4:0] b;
initial fork
    #5 a = 1'b1;
    #10 a = {a, 1'b0};
    #15 a = {a, 1'b1};
join
always @(*)
    case (a << 2)
        3'b100  : b = 3'd2;
        4'b1000 : b = 4'd6;
        5'b1_0100 : b = 5'd16;
        default:    b = 5'dx;
    endcase
```

（a）在仿真时刻 0： a = 3’bxxx b = 5’bx xxxx

（b）在仿真时刻 5： a = 3’b1 b = 5’b10

（c）在仿真时刻 10： a = 3’b10 b = 5’b110

（d）在仿真时刻 15： a = 3’b101 b = 5’b1 0000

（4）根据下面的部分 Verilog 语句，确定变量 a 和 b 的值：

```
reg [7:0] p = 8'b1000_0111;
reg [3:0] q = 4'b1000;
reg [3:0] a, b;
event e;
initial begin
       a = 0; b = 0;
       #5 if (|p) a = q;
       #5 if (a >= 0 ) -> e;
end
always @(e) begin b = (q >> 1); a = a | b; end
```

（a）在时刻 5 时， a = 4’b1000 b = 4’b0

（b）在时刻 10 时，a = 4’b1100 b = 4’b0100

（5）一个 Verilog 模块的部分语句如下：

```
always @(posedge clk) begin
        q[3] <= q[2];
        q[2] <= q[1];
        q[1] <= q[0];
        q[0] <= q[3];
    end
```

其中，变量 q 的当前值分别为：q = 4’b1010；请问：当下一个时钟上跳沿到来后，变量的取值是多少？ q = 4’b0101

（6）分析下面的模块，根据已给出的不同时刻的各个变量的取值，确定变量 f 在相应时刻的值。

```
module mfunc(f, en, data);
    inout [3:0] f
    input [1:0] en;
    input  [5:0] data;
    assign f = (|en) ? data : 4'bz;
endmodule
```

①在时刻 t = 5 时，en = 2’b0，data = 6’b00_1010；

此时：f = 4’bz

②在时刻 t = 10 时，en = 2’b01，data = 6’b00_1011；

此时：f = 4’b1011

③在时刻 t = 15 时，en = 2’b11，data = 6’b11_1101；

此时：f = 4’b1101

（7）根据下面的语句，判断其中的变量类型，并说明其理由。

```
assign a = b;
always @(*)
    count = c + 1;
```

a） a(wire) b） b(wire/reg) c） count(reg) d） c(wire/reg)

答：a）连续赋值语句只能对 wire 类型变量进行赋值；

b）表达式的变量即可以是 wire 类型也可以是 reg 类型；

c）过程赋值语句只能对 reg 类型变量进行赋值；

d）表达式的变量即可以是 wire 类型也可以是 reg 类型；

（8）下面两个模块:（i）为 4 选 1 多路复用器的测试台模块，（ii）为 4 选 1 多路复用器。这些模块包含错误，找出模块中的错误，将其更改，并说明理由。

（i）

```
module mux4_tb;
    reg  [7:0] py;
    wire [7:0] pa, pb, pc, pd;
    reg select;

    mux4 u0(
            .y(py),
            .a(pa), .b(pb), .c(pc), .d(pd),
            .sel(select)
            );
    ……
endmodule
```

（ii）

```
module mux4(input [1:0] sel,
             input [7:0] a, b, c,
d,
             output [7:0] y );
    always @(a, b, c, d)
        case (sel)
            0:  y <= a;
            1:  y <= b;
            2:  y <= c;
            3:  y <= d;
            default: y <= 8'bx;
        endcase
endmodule
```

答:（i）模块 mux4_tb 中包含量 2 个错误:

a）py 连接模块的 mux4 的输出端，必须是线网类型 wire;

b)1 bit 宽 select 连接模块的 mux4 的 2 bits 宽输入端 sel，必须声明 select 为 2 位宽。

（ii）模块 mux4 中包含量 3 个错误:

c）由于在过程语句中进行赋值，输出端 y 应当是 reg 类型;

d）在 always 过程语句中，sel 不在敏感变量（事件控制）列表中，事件控制列表应当包含所有信号变量。

e）4 选 1 多路复用器是组合逻辑电路，必须使用阻塞赋值语句。

（i）

```
module mux4_tb;
    wire  [7:0] py;
    wire [7:0] pa, pb, pc, pd;
    reg [1:0] select;

    mux4 u0(
            .y(py),
            .a(pa), .b(pb), .c(pc), .d(pd),
            .sel(select)
            );
    ……
endmodule
```

（ii）

```
module mux4(input [1:0] sel,
            input [7:0] a, b, c,
d,
            output reg [7:0]
y );
    always @(a, b, c, d, sel)
// 或者改为：always @*   always
@(*)
        case (sel)
            0:  y = a;
            1:  y = b;
            2:  y = c;
            3:  y = d;
            default: y = 8'bx;
        endcase
endmodule
```

（9）下面是一个 Verilog 模块：

```
module top;
① reg [3:0] a = 4'b0;
   reg [3:0] b = 4'b0;
   reg [4:0] c = 5'b0;
   reg [5:0] d = 6'b0;
   reg [5:0] e = 6'b0;

  initial begin
       a = 4'b10x0;
       b = 4'b0001;
       c = 5'bZ;
    #5 c = !a == !b;
    #5 c = c & (~b);
    #5 c = c + b;
    #5 c = {c[2:0], d[3:2]};
    #5 c = c + a;
  end
```

```
②initial fork
       begin
           #0 d = 6'bz10;
           @(c) d = 6'b1;
           @(c) d = 6'b10;
           @(c) d = 6'b11;
           @(c) d = 6'b101;
           @(c) d = 6'b111;
       end
       begin
           #0 wait(d)  e = 6'bx;
           #5 wait(d)  e = 6'b11_0000;
           #5 wait(d)  e = 6'b01_1000;
           #5 wait(d)  e = 6'b00_1100;
           #5 wait(d)  e = 6'b00_0011;
           #5 wait(d)  e = 6'b10_0001;
       end
  join
```

```
endmodule
```

根据这段程序，确定运行后的结果：

t = 0, c = 5'bz , d = 6'bz10 , e = 6'bx

t = 5, c = 5'b1 , d = 6'b1 , e = 6'b11 0000

t = 10, c = 5'b0 , d = 6'b10 , e = 6'b01 1000

t = 15, c = 5'b1 , d = 6'b11 , e = 6'b00 1100

t = 20, c = 5'b100 , d = 6'b101 , e = 6'b00 0011

t = 25, c = 5'bx , d = 6'b111 , e = 6'b10 0001

二、简单模块

（1）只使用门原语语句，不使用其它任何语句，重新改写下面的 Verilog 模块。

```
module infer( output reg q,  input a, c, e);
    always @(a or c or e)
        q = ((c == 1'b1) && (e == 1'b0)) ? a : 1'b1;
endmodule
```

答：使用门原语

设：ctrl0 = (c == 1'b1) && (e == 1'b0)

ctrl = ~ctrl0

1’b1	a	ctrl	ctrl0	q
1	0	1	0	1
1	1	1	0	1
1	0	0	1	0
1	1	0	1	1

q = ctrl | a

```
module infer_gate( output q,  input a, c, e);
   wire ebar;
   wire ctrl;
   not u0(ebar, e);
   nand u1( ctrl, ebar, c);
   or u2(q, ctrl, a);
endmodule
```

（2）下面两个 Verilog 模块，分别含有两处错误，找出其中错误将其改正，并说明改正理由。分析这两个模块的功能是否相同。

i）

```
module RCA #(parameter WIDTH=8)
           ( output cout, output [WIDTH-1:0] sum, input [WIDTH-1:0] a, b );
    reg [WIDTH-2:0] c;
    genvar k;
    generate for (k = 0; k < WIDTH; k = k+1) begin: LOOP
        always @*
            case (k)
                0:          {c[k], sum[k]} = a[k] + b[k];
                (WIDTH-1):  {cout, sum[k]} = a[k] + b[k] + c[k-1];
                default:    {c[k], sum[k]} = a[k] + b[k] + c[k-1];
            endcase
    end
    endgenerate
endmodule
```

ii）

```
module FA #(parameter WIDTH = 8)
            ( output cout,
              output reg [WIDTH-1:0] sum,
              input [WIDTH-1:0] a, b
             );
    reg [WIDTH:0] carry;
    genvar k;
    generate
    for ( k = 0; k < WIDTH; k = k+1 ) begin : ripple
        assign {carry[k+1], sum[k]} = a[k] + b[k] + carry[k];
    end
    endgenerate
    assign carry[0] = 1'b0;
    assign cout = carry[WIDTH];
endmodule
```

i）答：输出端口 cout 和 sum 过程语句中进行赋值，应当声明为 reg 类型。

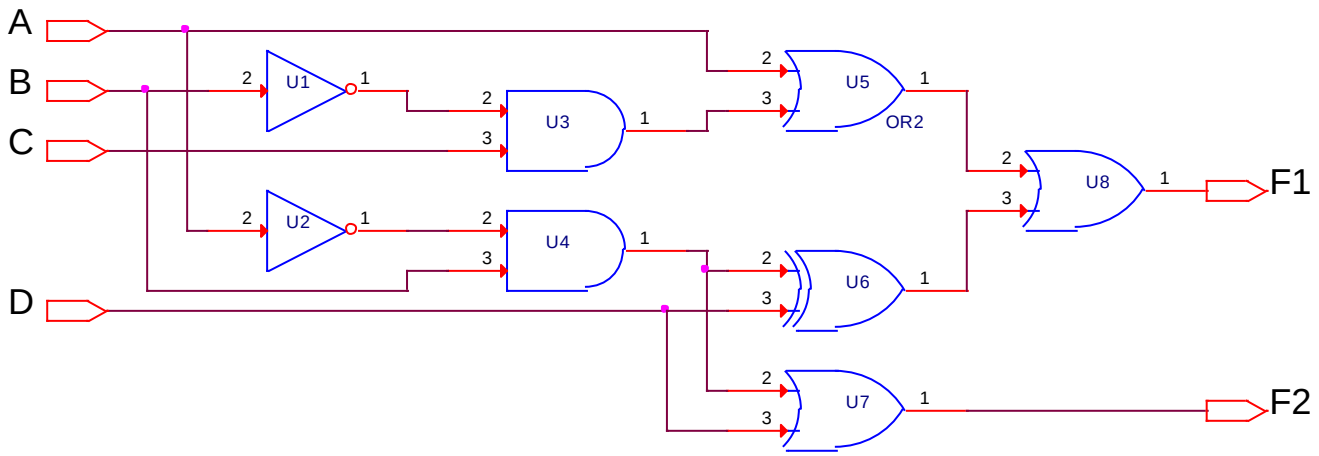
```
module RCA #(parameter WIDTH=8)
            ( output reg cout, output reg [WIDTH-1:0] sum, input [WIDTH-1:0] a,
b );
    reg [WIDTH-2:0] c;
    genvar k;
    generate for (k = 0; k < WIDTH; k = k+1) begin: LOOP
        always @*
            case (k)
                0:          {c[k], sum[k]} = a[k] + b[k];
                (WIDTH-1):  {cout, sum[k]} = a[k] + b[k] + c[k-1];
                default:    {c[k], sum[k]} = a[k] + b[k] + c[k-1];
            endcase
    end
    endgenerate
endmodule
```

ii）答：输出端口 sum 和中间变量 carry 在连续赋值语句中进行赋值，应当声明为 wire 类型。

```
module FA #(parameter WIDTH = 8)
            ( output cout,
              output [WIDTH-1:0] sum,     // output reg [WIDTH-1:0] sum,
              input [WIDTH-1:0] a, b
             );
    wire [WIDTH:0] carry;   // reg [WIDTH:0] carry;
    genvar k;
    generate
    for ( k = 0; k < WIDTH; k = k+1 ) begin : ripple
        assign {carry[k+1], sum[k]} = a[k] + b[k] + carry[k];
    end
    endgenerate
    assign carry[0] = 1'b0;
    assign cout = carry[WIDTH];
endmodule
```

答：这两个模块的功能完全相同，均为参数化方式设计的全加器。

(3)根据电路原理图：



1）只使用**连续赋值语句**设计 Verilog 模块，模块开始语句为：

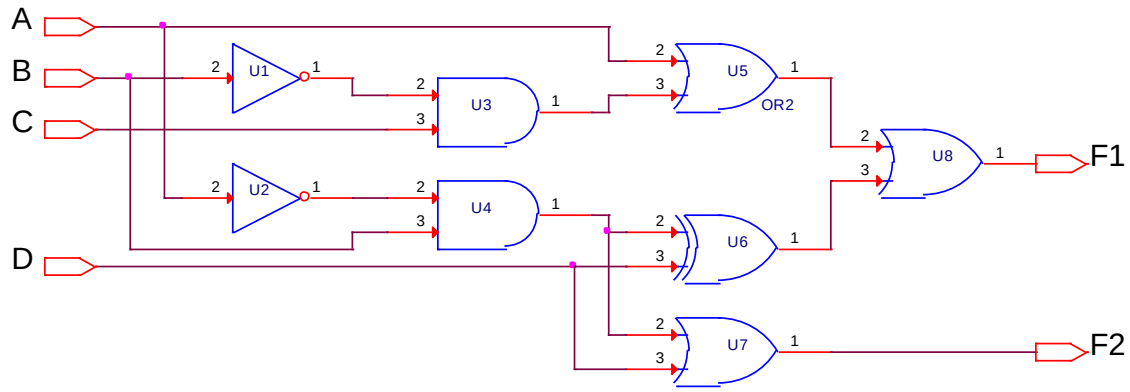
module comb1(output F1, F2, input A, B, C, D);

2）只使用**过程语句**设计 Verilog 模块，模块名为： **comb2**

3）编写测试平台模块。

答：1）使用连续赋值语句设计

```
// File: comb1.v
module comb1( output F1, F2, input A, B, C, D );
    wire w11, w12, w21, w22, w31, w32;
    assign w11 = ~B,
           w12 = ~A,
          w21 = C & w11,
          w22 = B & w12,
          w31 = A | w21,
          w32 = D ^ w22,
          F1  = w31 | w32,
          F2  = D | w22;
endmodule
```



2）使用过程语句设计

```
// File: comb1.v
module comb2( output reg F1, F2, input A, B, C, D );
    reg w11, w12, w21, w22, w31, w32;
    always @(*) begin
        w11 = ~B;          w12 = ~A;
        w21 = C & w11;     w22 = B & w12;
        w31 = A | w21;     w32 = D ^ w22;
        F1  = w31 | w32;   F2  = D | w22;
    end
endmodule
```

3）测试平台模块

```
`timescale 1ns/1ns
`include "comb1.v"    // `include "comb2.v"
module tb_comb();
    parameter LOOP = 15;
   reg pa, pb, pc, pd;
   wire pF1, pF2;

    comb1 uut0( .F1(pF1), .F2(pF2), .A(pa), .B(pb), .C(pc), .D(pd) );

    initial   begin
        {pa, pb, pc, pd} = 4'b0;
        repeat(LOOP)  begin
            #1 {pa, pb, pc, pd} = {pa, pb, pc, pd} + 1'b1;
        end
    end
    initial
        $monitor("At time %4t, {A,B,C,D}=%b, F1=%b, F2=%b", $time,
                 {pa,pb,pc,pd}, pF1, pF2);
endmodule
```

Transcript

```
VSIM 15> run
# At time     0,     {A,B,C,D}=0000, F1=0, F2=0
# At time     1,     {A,B,C,D}=0001, F1=1, F2=1
# At time     2,     {A,B,C,D}=0010, F1=1, F2=0
# At time     3,     {A,B,C,D}=0011, F1=1, F2=1
# At time     4,     {A,B,C,D}=0100, F1=1, F2=1
# At time     5,     {A,B,C,D}=0101, F1=0, F2=1
# At time     6,     {A,B,C,D}=0110, F1=1, F2=1
# At time     7,     {A,B,C,D}=0111, F1=0, F2=1
# At time     8,     {A,B,C,D}=1000, F1=1, F2=0
# At time     9,     {A,B,C,D}=1001, F1=1, F2=1
# At time    10,     {A,B,C,D}=1010, F1=1, F2=0
# At time    11,     {A,B,C,D}=1011, F1=1, F2=1
# At time    12,     {A,B,C,D}=1100, F1=1, F2=0
# At time    13,     {A,B,C,D}=1101, F1=1, F2=1
# At time    14,     {A,B,C,D}=1110, F1=1, F2=0
# At time    15,     {A,B,C,D}=1111, F1=1, F2=1

VSIM 16>
```

（4）使用行为建模的方式，只使用过程语句块，重新改写下面的 Verilog 模块：

```
module comb( output f, g, input a,b,c );
    wire na, nb, nc, t3, t5, t6;

    not n1( na, a ),
        n2( nb, b ),
        n3( nc, c );
    and #2 a1( t3, na, b, c );
    and #1 a2( t5, a, nb, c ),
           a3( t6, a, b, nc );
    or b1( f, t3, t6 );
    or #3 b2( g, a, t5 );
endmodule
```

答：上面 Verilog 模块使用行为建模的方式重新编写如下：

```
module comb( output reg f, g, input a,b,c );
    reg t3, t5, t6;

    always @(*)
        #2 t3 = (~a) & b & c;

    always @(*)
        #1 t5 = a & (~b) & c;

    always @(*) t6 = a & b & (~c);

    always @(*) f = t3 | t6;

    always @(*)
        #3 g = a | t5;
endmodule
```