名词解释：

1. ***EDA***：是电子设计自动化的缩写，EDA 技术就是以计算机为工具，设计者在 EDA 软件平台上，用硬件描述语言 VHDL 完成设计文件，然后由计算机自动地完成逻辑编译、化简、分割、综合、优化、布局、布线和仿真，直至对于特定目标芯片的适配编译、逻辑映射和编程下载等工作。
2. ***HDL***：硬件描述语言，是一种以文本形式描述数字电路和数字系统的语言，是指对硬件电路进行行为描述、寄存器传输描述或者结构化描述的一种新兴语言。
3. ***FPGA***（：现场可编程逻辑门阵列，它采用了逻辑单元阵列 LCA 这样一个概念，内部包括可配置逻辑模块 CLB、输入输出模块 IOB 和内部连线三个部分。FPGA 利用小型查找表（16×1RAM）来实现组合逻辑。
4. ***CPLD***：复杂的可编程逻辑器件，主要是由可编程逻辑宏单元围绕中心的可编程互连矩阵单元组成。是一种用户根据各自需要而自行构造逻辑功能的数字集成电路。其基本设计方法是借助集成开发软件平台，用原理图、硬件描述语言等方法，生成相应的目标文件，通过下载电缆（“在系统”编程）将代码传送到目标芯片中，实现设计的数字系统。
5. ***IP***: IP 是知识产权核或知识产权模块，用于 ASIC 或 FPGA/CPLD 中的预先设计好的电路功能模块。
6. ***Testbench***: 在设计数字电路系统时，通常将测试模块和功能模块分开设计，其中测试模块也称测试台（Testbench）。Testbench 是通过对设计部分施加激励，然后检查其输出正确与否来完成其验证功能的。
7. ***reg***:是寄存器数据类型的关键字，其表示一个抽象的数据存储单元。reg 只能在 initial 和 always 中赋值。而 reg 在过程赋值语句中使用。reg 型数据常用来表示 always 模块内的指定信号，代表触发器。通常在设计中要由 always 模块通过使用行为描述语句来表达逻辑关系。在 always 块内被赋值的每一个信号都必须定义为 reg 型。
8. ***wire***:是最常用的 Net 型变量。wire 表示直通，即只要输入有变化，输出马上无条件地反映。wire 使用在连续赋值语句中,即以 assign 关键字指定的组合逻辑信号。Verilog 程序模块中输入、输出信号类型默认为 wire 型。wire 型的变量综合出来一般是一根导线。
9. ***FSM***（:有限状态机。是由寄存器组和组合逻辑构成的硬件时序电路。是用来记录电路当前状态的一种电路结构。存储器记录电路当前状态，而组合逻辑用来根据当前状态和当前输入运算出电路的下一个状态。其分为两种：Meaiy 机和 Moore 机。
10. ***层次化设计***：是 Verilog HDL 设计描述的一种风格，而模块实例化是其具体的实现方式。其中一种是自顶向下设计，就是从整个系统设计的顶层开始，往下一层将系统划分为若干个子模块，然后再将每一个子模块又向下一层划分为若干的子模块。通过这样将整个系统逐次向下分解，一个顶层设计最后可以细分为若干较小的基本功能块，直到不能继续分解为止。
11. ***模块***：是 Verilog HDL 设计中的一个基本组成单元。一个模块通常就是一个电路单元器件 。一个模块的代码主要由下面几个部分构成：模块名定义、端口描述和内部功能逻辑描述。模块名必须是唯一的。
12. ***行为描述***：使用结构化过程语句对时序行为进行描述。其中结构化过程语句包括 两种语句：initial 语句和 always 语句 （行为描述以过程块为基础组成单位，一个模块的行为描述由一个或者多个并行运行的过程块组成。）
13. ***仿真***：利用仿真工具，在 PC 上对 Verilog HDL 代码所描述的电路功能进行验证。仿真是在 PC 上进行的，通过软件完成。仿真工具提供很多功能强大的调试功能，可以帮助设计者方便且迅速地查找设计中的错误。
14. ***综合***：将 Verilog HDL 描述的代码转换成实际的电路结构，转换后的电路可以用于生产并实现真正的芯片硬件电路。
15. ***阻塞赋值***：用“=”作为赋值符。阻塞语句按顺序执行，在下一条语句执行之前，上一条赋值语句必须执行完毕。组合电路中用的是阻塞赋值。

16. ***非阻塞赋值***：用“<=”作为赋值符。非阻塞赋值语句不会阻塞同一个块语句中的其他语句的执行。时序电路中用的是非阻塞赋值。
17. ***事件控制***：通过事件的发生来触发语句的执行。一个事件通常是指一个变量，线网信号或表达式的值发生变化。所以语句的执行在制定变量的值发生变化的时刻开始，从而提供了更灵活和复杂的时序控制方法。其可以分为边沿敏感事件控制和电平敏感事件控制。

18. ***任务***：任务可以在 always 或者 initial 模块中的任何过程语句中调用。任务中可以包含带时序控制的语句。可计算多个结果值，输入和输出可为各种类型（包括 inout 型），任务可调用其他任务和函数，不向表达式返回值。

19. ***函数***：函数中不能包含时序控制语句，通过返回一个值，来响应输入信号，输入和输出至少有一个输入变量，但不能有任何 output 或 inout 型变量，可作为表达式中的一个操作数来调用，在过程赋值和连续赋值语句中均可调用，函数可调用其他函数，但不可调用其他任务，向调用它的表达式返回一个值。

20. ***异步电路***：主要是组合逻辑电路，用于生产地址译码器，但它同时也用在时序电路中，此时它没有统一的时钟，状态变化的时刻是不稳定的，通常输入信号只在电路处于稳定状态时发生变化。
21. ***同步电路***：由同步时序电路和组合逻辑电路构成的电路，其所有操作都是在严格的时钟控制下完成的。这些时序电路共享同一个时钟 CLK,而所有的状态变化都是在时钟的上升沿（或下降沿）完成的。
22. ***亚稳态：*** 数字电路对于电平小于电压阀值 VL 称之为 0，大于电压阀值 VH 称之为 1，对于从 0 到 1 之间的上跳变或者从 1 到 0 之间的下跳变期间叫做系统的亚稳态。

简答题：

1. ***模块结构有哪些组成部分？***

答：模块名定义、端口描述和内部功能逻辑描述。用关键字 modoule 定义模块名字，且模块名字必须是唯一的，接着分别用 input 和 output 关键字指定端口方向，模块逻辑功能描述包括：变量声明、数据流描述语句、门级实例化描述语句、行为描述语句及任务与函数，最后用关键字 endmodoule 结束模块的描述。

2. ***Moore 机和 Mealy 机有什么区别？***

答：Moore 机的输出只决定于状态机的当前状态，与当前输入没有关系；而 Mealy 机的输出不仅决定于当前的状态，还决定于当前时刻的输入。

3. ***Reg 型变量和 wire 型变量有什么区别？***

答：wire 表示直通，即只要输入有变化，输出马上无条件地反映；reg 表示一定要有触发，输出才会反映输入。wire 只能被 assign 连续赋值，reg 只能在 initial 和 always 中赋值。wire 使用在连续赋值语句中，而 reg 使用在过程赋值语句中。reg 型保持最后一次的赋值，而 wire 型则需要持续的驱动 .

4. ***任务和函数的区别？***

答：（1）.函数中不能包含时序控制语句，如@()、#10 等。对函数的调用，必须在同一仿真时刻返回。而任务可以包含时序控制语句，任务的返回时间和调用时间可以不同；（2）.在函数中不能调用任务，而在任务中可以调用其他任务和函数。但在函数中可以调用其他函数或函数自身（递归调用）；（3）.函数必须包含至少一个端口，且在函数中只能定义 input 端口。任务可以包含 0 个或任何多个端口，且可以定义 input、output 和 inout 端口。（4）.函数必须返回一个值，而任务不能返回值，只能通过 output 端口来传递执行结

5. ***Verilog HDL 有哪些并行语句？*** 答：进程语句，块语句，并行信号赋值语句，并行过程调用语句，元件列化语句。

6. ***Verilog HDL 有哪些描述形式？*** 答：数据流描述方式；行为描述方式；层次化描述方式。

写程序：

阻塞赋值方式定义的 2 选 1 多路选择器 (if_else)

```
module MUX21_2(out,a,b,sel);
  input a,b,sel;
  output out;
  reg out;
  always@(a or b or sel)
     begin
       if(sel==0)    out=a;              //阻塞赋值
       else              out=b;
       end
endmodule
```

用 case 语句描述的 4 选 1 数据选择器

```
module mux4_1(out,in0,in1,in2,in3,sel);
output out;
input in0,in1,in2,in3;
input[1:0] sel;
reg out;
  always @(in0 or in1 or in2 or in3 or sel)
     case(sel)
        2'b00:      out=in0;
        2'b01:      out=in1;
        2'b10:      out=in2;
        2'b11:      out=in3;
        default: out=2'bx;
     endcase
endmodule
```

3-8 译码器

```
module decoder_38(out,in);
output[7:0] out;
input[2:0] in;
reg[7:0] out;
always @(in)
  begin
     case(in)
        3'd0: out=8'b11111110;
        3'd1: out=8'b11111101;
        3'd2: out=8'b11111011;
        3'd3: out=8'b11110111;
        3'd4: out=8'b11101111;
        3'd5: out=8'b11011111;
        3'd6: out=8'b10111111;
        3'd7: out=8'b01111111;
    endcase
  end
endmodule
```

2-4 译码器

```
module binary_decoder_2_4
       (
              input          i_en,
              input [1:0]    i_dec,
              output reg [3:0]    o_dec    );
       always @ (i_en or i_dec)
              if (i_en)
                     case (i_dec)
                            2'b00: o_dec = 4'b0001;
                            2'b01: o_dec = 4'b0010;
                            2'b10: o_dec = 4'b0100;
                            2'b11: o_dec = 4'b1000;
                            default:
                                   o_dec = 4'bxxxx;
                     endcase
              else
                     o_dec = 4'b0000;
 Endmodule
```

4-2 编码 （优先用 casex 1xxx，01xx，001x

```
module binary_encoder_4_2
       (
              input    [3:0] i_dec,
              output [1:0] o_dec
       );
```

```
assign o_dec = (i_dec == 4'b0001) ? 2'b00 :
(i_dec == 4'b0010) ? 2'b01 :
(i_dec == 4'b0100) ? 2'b10 :
```

```
   (i_dec == 4'b1000) ? 2'b11 : 2'bxx;                    endmodule
```

定义一个一位的全加器

```
module fadder_1( i_A, i_B, i_Cin,o_S,o_Cout);
input    i_A, i_B;              //输入端口 i_A,i_B
input    i_Cin;                  //输入端口 i_Cin
output o_S, o_Cout;         //输出端口 o_S,o_Cout
//  计算结果值：o_S = i_A⊕i_B⊕i_Cin
  assign    o_S = i_A ^ i_B ^ i_Cin;
 //  计算进位值：o_Cout=（i_A⊕i_B）i_Cin+(i_A)(i_B)
assign    o_Cout = (i_A ^ i_B) & i_Cin | i_A & i_B;
endmodule
```

同步置数、同步清零的计数器

```
module count(out,data,load,reset,clk);
   output[7:0] out;
   input[7:0] data;
   input load,clk,reset;
   reg[7:0] out;
   always @(posedge clk)                          //clk 上升沿触发
      begin
        if (!reset)       out = 8'h00;       //同步清 0，低电平有效
        else if (load)      out = data;         //同步预置
        else              out = out + 1;       //计数
      end
endmodule
```

模 10 计数器

```
   module    cnt10_v(clr, clk,ena,q,cout);
   input                clr, clk,ena;
   output[3:0]         q;
   output               cout;
   reg[3:0]            q;
   reg                   cout;
   always    @(posedge clk or posedge clr)
      begin
        if (clr)
        begin    q=4'b0000; cout=0; end
        else if (ena)
            if (q==4'b1001)
              begin q=4'b0000; cout=0; end
            else
            begin q=q+1 ;cout=q[3] && q[0]; end
        end
   endmodule
```

8 位移位寄存器

```
module shifter(din,clk,clr,dout);
   input din,clk,clr;
   output[7:0] dout;
   reg[7:0] dout;
   always @(posedge clk)
      begin
        if (clr)    dout<= 8'b00000000;            //同步清 0，高电平有效
        else
          begin
            dout <= dout << 1;     //输出信号左移一位（>>1 时候   dout[7]<=din）
            dout[0] <= din;              //输入信号补充到输出信号的最低位
          end
       end
      endmodule  【 else if (~ldn)begin q=d; end else】
                       【begin q[6:0]=q[7: 1];q[7]=dsr; 】
```

用 always 过程语句描述的简单算术逻辑单元

```
`define add 3'd0
`define minus 3'd1
`define band 3'd2
`define bor 3'd3
`define bnot 3'd4
module alu(out,opcode,a,b);
   output[7:0] out;
   reg[7:0] out;
   input[2:0] opcode;              //操作码
   input[7:0] a,b;                 //操作数
   always@(opcode or a or b)       //电平敏感的 always 块
    begin
      case(opcode)
        `add: out = a+b;             //加操作
        `minus: out = a-b;           //减操作
        `band: out = a&b;            //按位与，若&a，表缩减与
        `bor: out = a|b;               //求或
        `bnot: out=~a;               //求按位反
        default: out=8'hx;           //未收到指令时，输出任意态
      endcase
    end
endmodule
```

逻辑与&& 逻辑的等于 1，0 表真假