B. 数字部分 （共 50 分）

计分栏

一 (10 分)	二 (15 分)	三 (15 分)	四 (10 分)	合计

注意：答案直接写在试卷的相应位置上。

一. 填空题（共 10 分，每空 1 分）

1. 变量是在程序运行过程中其值可以改变的量。在 Verilog 中变量分为两种，一种是线网类型，一般表示硬件电路的物理连线，另一种是 寄存器类型，或者 reg ，对应的是具有状态保持作用的电路元件。

2.
```
always begin #5 clk=0;
#10 clk=~clk; end
```
产生的波形的占空比为 1/3 或者 33% ,答 1:2 的也算对 。

3. reg 类型的数组通常用于描述存储器，语句 reg[31:0] memory[1:1024]; 所定义的存储器的位宽为 32 位 。

4. 在对变量进行赋值时，可以采用不确定值 X，高阻态 Z 进行赋值操作。定义 a 为 8 位的线网类型变量，语句 a = 8'b1; 和 a = 8'b1X; 执行完后 a 的值分别为 8'b00000001 和 8'b0000001X 。

5. IP 核在 EDA 技术和开发中具有十分重要的地位，以 HDL 方式提供的 IP 核被称为 软核 。

6. 利用位拼接运算符可以实现多个信号的某些位的重新组合，{1,0}经过位拼接之后的值为 64'H0000000100000000 。

7. 一个大型的组合电路总延时为 100ns，采用流水线将它分成两个较小的组合电路，不考虑寄存器建立时间延迟，理论上该流水线电路最高工作频率可以为 20 MHz。

8. 有表达式：
```
wire [7:0] x;
wire y;          /*请补充变量 y 的定义*/
```

```
assign x=8'hEB;
assign y= &x;
```

请问 y 的值为：_____y=1'b0_____。

二. 电路分析与设计（共 15 分，每小题 5 分）

（异步时序电路设计的方法与同步时序电路设计的方法基本一致，可以采用多个 always 块语句实现时序逻辑，区别在于异步时序电路中的多个 always 块的时钟信号可以为不同的时钟触发信号。）观察图 1 所示的异步时序电路，在 B_part 部分进行信号生成，在 C_part 部分完成逻辑控制，B_part 部分信号生成波形如 A_part 部分所示。

请解答如下问题：（注意：**本题共包含 3 道小题！并注意模块的完整性。**）

(1) 针对图 1 中 C_part 部分，利用 Verilog 语言，采用行为描述风格设计该电路，电路接口参看 C_part 部分电路，注意：不允许使用“xor”、“D_FF”等元件，模块名为 amod。

```
module amod(d, a, clk, clr, q);// 1分
   input a, d, clk, clr;
   output q;
   reg q, clk2;

   //产生次级 D 触发器时钟控制信号
   always@(posedge clk or negedge clr)
     if ( !clr )  clk2 <= 0 ;
     else clk2 <= ~( a | q ); // 2分

     //采用 D 触发器进行数据输出
  always@(posedge clk2 or negedge clr)
     if ( !clr )  q <= 0 ;
     else q <= d; // 2分
endmodule
```

按照 B 部分，则答案为：

```
module amod(clr, d_in, a, d);
input clr, d_in;
output a,d;
reg q1,q2;
always@(posedge d_in or negedge clr)
if(!clr)  q1<=0;
else q1<=~q1;
always@(negedge d_in or negedge clr)
if(!clr)  q2<=0;
else q2<=~q2;
assign a=q1 & q2;
assign d=q1 | q2;
endmodule
```

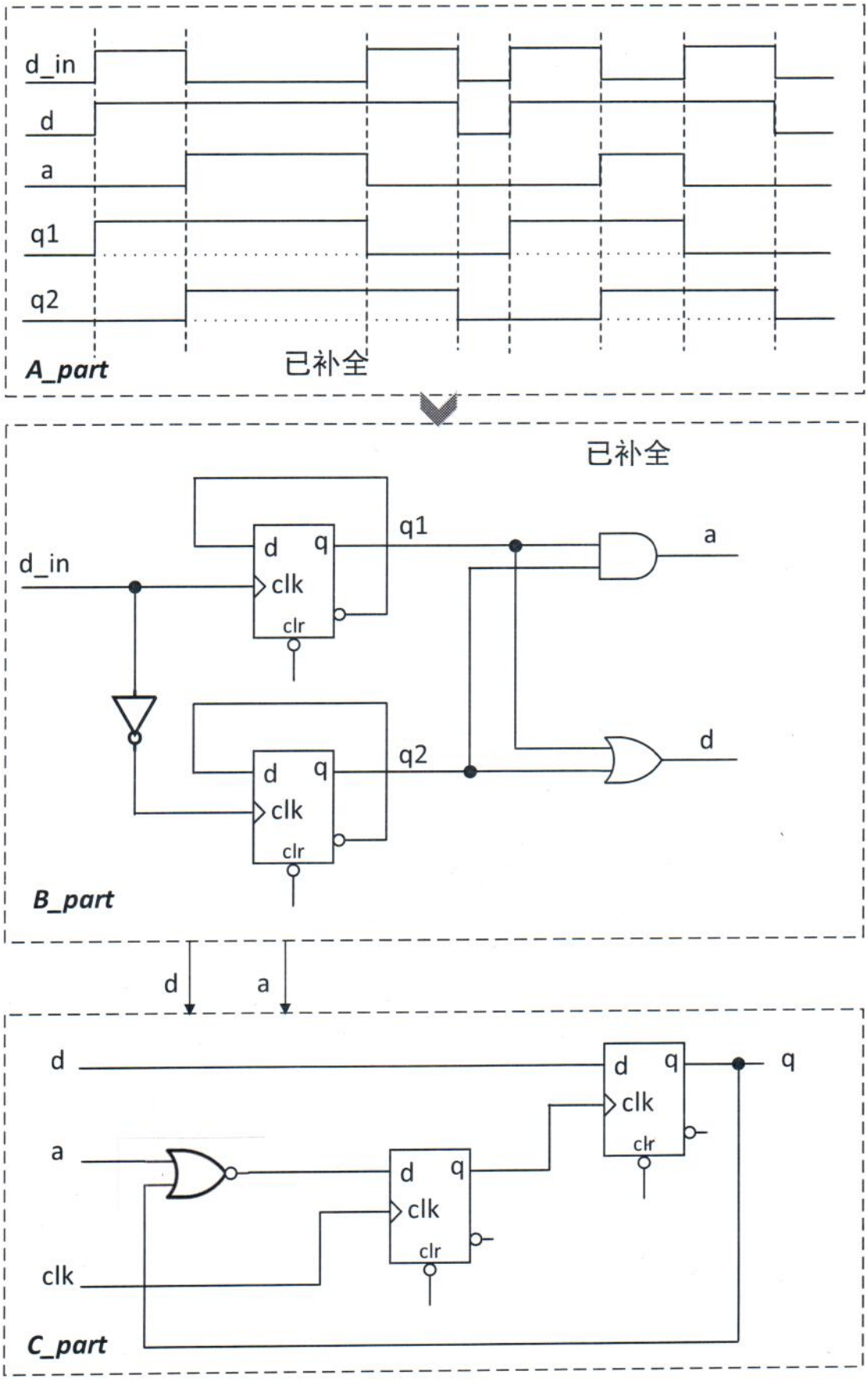
d_in
d
a
q1
q2
A_part
已补全
已补全
d_in
d
q
clk
clr
q1
a
q2
d
B_part
d
a
d
a
clk
d
q
clk
clr
q
C_part

图 1

(2) 在图 1 中的 B_part 部分完成信号匹配和生成，（依据于输入的 d_in 信号，产生对应的输出信号 d 和 a，并将生成的 d 和 a 最终接入到 C_part 部分。）输入信号 d_in 与其匹配生成的输出信号 d 和 a 的时序关系如 A_part 所示。（为了实现该模块，可以在 D 触发器（上升沿触发）的基础上完成设计。）结合 A_part 部分的部分电路，首先给出中间信号 q1 和 q2 的波形；然后在 B_part 部分补全该电路原理图。

答案参见图 1 中所示。两个波形 2 分，两个门 3 分。

(3) 在第(2)问的基础上，根据补全的 B_part 部分的电路原理图，利用 Verilog 语言，采用门级建模的方法设计该电路，模块名为 signal。电路接口参看 B_part 部分电路。（设 Verilog 语言支持多输入与门（and）、或门（or）、与非门（nand）、或非门（nor），以及反相器（not）等基本门电路，其模块接口输出参数在前，输入参数在后。）对于 D 触发器，其 Verilog 模块如下，可以直接实例化 D_FF 以实现 D 触发器的调用。

```
// module D_FF ( Delay Flip-Flop )
module D_FF ( clk, clr, D, Q, Q_n ) ;
  input clk, clr, D ;
  output Q, Q_n ;
  reg Q ;
  assign Q_n = ~Q ;
  always @ ( posedge clk or negedge clr )
    if ( !clr )  Q <= 0 ;
    else Q <= D ;
endmodule
```

```
module signal(d_in, clr, a, d);
   input d_in, clr;
   output a, d;
   wire d_in_n = ~ d_in;                          // 1分
   wire q1, q1_n, q2, q2_n;
   D_FF dff1(d_in, clr, q1_n, q1, q1_n);          // 1分
   D_FF dff2(d_in_n, clr, q2_n, q2, q2_n);        // 1分
   and and1(a, q1, q2);                           // 1分
   or or1(d, q1, q2);                             // 1分
endmodule
```

三. 电路分析与测试（共 15 分，第 1 小题 8 分，第 2 小题 7 分）

针对一个对 8 位并行输入的数据字中各个比特位为“1”的个数进行计数输出的模块，在全局时钟节拍下实现“1”个数的计数，如图 2 所示。同时保证计数个数从 0 依次增长到最大值并保持（如图 2 中第一个数据字 8'h07，其各个比特位为“1”的个数为 3，在第 5 个采样时刻，计数个数保持 3 不变）。当输入的 8 位数据字的采样周期中“1”的计数没能增长到最大值就更新到新的数据字（如图 2 中第二个数据字 8'h4F，其各个比特位为“1”的个数为 5，需要 6 个采样时刻才能完全计满，但在第 5 个采样时刻数据字变化成 8'h08），则计数器对更新后的数据字中“1”的个数重新进行计数。

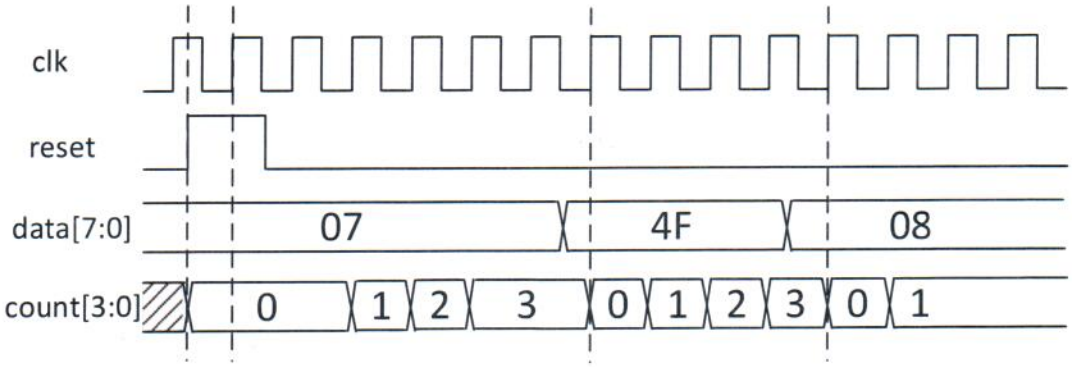

图 2

该模块顶层结构框图如图 3 所示。

图 3

请解答如下问题：（注意：本题共包含 2 道小题！并注意逻辑的正确性。）

(1) 根据计数功能，利用 Verilog 语言，采用可综合风格设计计数器 bitcounter 模块，接口参照图 3。

```
module bitcounter (clk, reset, count,  data);
   input clk, reset;
   input [7:0] data;
   output [3:0] count;
   reg [7:0] databuffer;                          // 1分
   reg [2:0] max_1;
   reg [3:0] count;

   always@( posedge clk or posedge reset)
      if ( reset )  begin                         // 2分
         databuffer <= 0;
         max_1 <= 0;
         count <=0;     end
      else begin
         if (databuffer != data) begin            // 3分
            databuffer <= data;
            max_1 <= data[0] + data[1] + data[2] + data[3]
               + data[4] + data[5] + data[6] +data[7];
            count <= 0;       end
         else  begin
            if (count <= max_1 -1)                // 2分
               count <= count + 1;
         end
      end
endmodule
```

(2) （为了验证模块设计的正确性，需要编写测试程序（test bench）。）测试程序在全局时钟控制下生成如图4所示的激励信号。测试流中共有255个测试数据，在 reset 信号之后，从 8'h01 依次增长到 8'hFF，（即测试数据内容与数据序号相同；图4中第一个数据 8'h00 为 reset 信号后初始化值）测试数据时间长度为时钟周期的随机整倍数（随机数的系统函数为$random），随机范围为 1~8 倍，请编写相应的测试模块，并注意测试时钟周期。

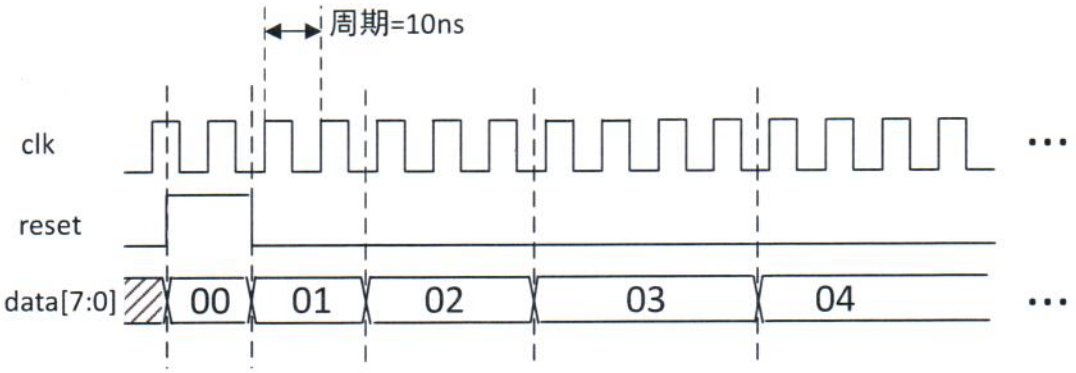

图 4

```
`timescale 1ns/1ns
module tb_bitcounter ;
   reg clk, reset ;
   reg [7:0] data;
   wire [3:0] count;

   reg [8:0] i;
   reg [12:0] timedelay;
   bitcounter uut (clk, reset, count, data);    // 1分

   initial begin
      clk = 0;                                  // 1分
      reset = 0;
      data = 8'hxx;
      #6 reset = 1 ; data = 0 ;                 // 1分
      #15 reset = 0;
      data = 1;
      for (i =2; i<256; i=i+1) begin            // 1分
         timedelay = 10*({$random}%8+1);        // 1分
        # timedelay data = i;  end               // 1分
      #100 $stop ;   end

   always #5 clk = ~clk;                        // 1分
endmodule
```

四. 有限状态机设计（共 10 分）

请采用“二段式”的风格，实现如图 5 所示的有限状态机，注意模块的完整性和正确性。

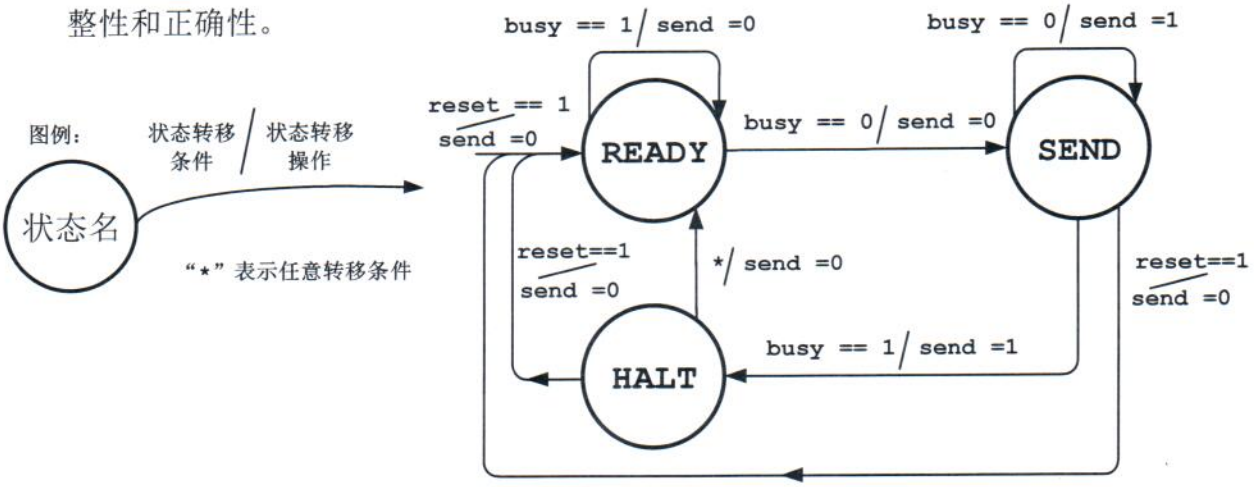

图 5

```
module tx_FSM (clock, reset, busy, send);
   parameter [1:0] READY = 2'b00, SEND = 2'b11, HALT = 2'b10 ;
   input clock ;
   input reset, busy ;
   output send ;

   reg [1:0] state, next_state ;
   always @( posedge clock or posedge reset )  // 请填写代码
      if ( reset ) begin                          //1分
         state <= READY ;       end
      else state <= next_state ;                  // 1分

   always @( busy or state )       // 请填写代码
    case( state )
       READY :                                    //2分
          if( busy == 1 ) next_state <= READY ;
          else next_state <= SEND ;
       SEND :                                     //2分
          if( busy == 0 ) next_state <= SEND ;
          else next_state <= HALT ;
       HALT : next_state <= READY ;               //1分
       default : next_state <= READY ;            //1分
    endcase

   assign send = ( state == SEND ) ? 1 : 0 ;  //2分


endmodule
```