EDA Technology

杭州電子科技大学
HANGZHOU DIANZI UNIVERSITY

EDA

复习与习题课

考试题型

- 程序选择填空15%（每个选项只能选一次）
- 程序改错题15% （指出错误，改正，描述程序功能）
- 简答题10%
- Verilog程序转RTL图10%
- RTL图转Verilog程序10%
- 编程题20% （组合、时序电路编程，每题10分）
- 综合设计题20%

1.简答题

EDA是指？

Electronic Design Automation

电子设计自动化

FPGA是指？

Field Programmable Gate Array

现场可编程门阵列

1.简答题

- EDA设计流程包括了哪些步骤？
- 答：

设计输入（图形输入、文本输入）

↓

功能仿真（或RTL仿真）

↓

综合

↓

适配

↓

时序仿真（或门级仿真+静态时序分析）

↓

编程下载

↓

硬件测试

1.简答题

- 简单PLD包括？基本的编程原理是基于？
- 答：PROM、PLA、PAL、GAL；
- **乘积项**逻辑可编程结构
- 或 **与或阵列**可编程结构

- 复杂PLD包括？基本的编程原理是基于？
- 答：CPLD和FPGA；
- CPLD基于**乘积项（与或阵列）**逻辑可编程结构，FPGA基于SRAM查找表（LUT）逻辑可编程结构

1.简答题

- Intel提供了一种嵌入式逻辑分析仪是？它基于什么技术构建？
- 答：SignalTap，基于JTAG技术构建。

- Intel提供了一种虚拟IO的IP是？它基于什么技术构建？
- 答：In-System Sources and Probes，基于JTAG技术构建。

1.简答题

- 在Quartus中那些工具或者IP是使用了JTAG技术
- 答：
  - SignalTap
  - In-System sources and Probes
  - In-System Memory content Editor
  - System Console（不要求）
  - ……

1.简答题

- IEEE的那个标准是JTAG的？JTAG主要有哪些信号？
- 答：
  - IEEE 1149
    - TMS
    - TDO
    - TDI
    - TCK
    - TRST（可以不回答）
  - Verilog的IEEE标准是？
    - 答：IEEE Std 1364-1995
    - IEEE std 1364-2005

1.简答题

- SignalTap ，In-System Memory Content Editor，In-System Sources and Probes Editor 有什么异同？
- 答：
- 1)嵌入式逻辑分析仪SignalTap II要占据大量的存储单元作为数据缓存，在工作时只能单向地收集和显示硬件系统的信息，而不能与系统进行双向对话式测试，而且通常限制观察已设定端口引脚的信号；
- 2) 存储器内容在系统编辑器In-System Memory Content Editor能与系统进行双向对话式测试，但对象只限于存储器；
- 3) 在系统信号与源编辑器In-System Sources and Probes Editor能有效克服以上两种工具的不足，特别是对系统进行硬件测试的所有信号都不必通过I/O端口引到引脚处，及所有测试信号都在内部引入测试系统，或通过测试系统给出激励信号；
- 4) 都由FPGA的JTAG口通信。

1.简答题

- 列举速度优化方法？
- 答：流水线设计、寄存器配平、乒乓操作法、关键路径法等

- 列举资源优化方法？
- 答：资源共享、逻辑优化或串行化等

1.简答题

- 列举状态机的状态编码方式？
- 答：顺序编码、一位热码编码、格雷码、约翰逊码等。

- 以4个状态的状态机为例分别给出顺序二进制编码和一位热码对应的具体编码？
- 答：
- 二进制编码：S0=00，S1=01，S2=10，S3=11；
- 一位热码：S0=0001，S1=0010，S2=0100，S3=1000；

1.简答题

- SoC与SoPC分别是？
- 答：
- SoC：片上系统
- SoPC：片上可编程系统

2.程序填空

选择以下Verilog语句片段填入空格，以实现完整的程序功能，注意每个选项只能选一次。

A. [7:0] B. posedge C. cnt2 D. load E. wire F. reg G. 1 H. input
I. else J. assign K. pm L. ! M. negedge N. clk O. load P. if

```
// 以下是一个50%占空比的8位数控分频器设计，请填空完成程序：
module fdiv8 (clk,rstn,d,_______);
          input _______,rstn;
          _______ [7:0] d;
          output pm;
          _______ cnt2;
          reg  ________  q1;
          _______ load;
always @ (_______ clk or _______ rstn)
          _______ (!rstn)
                    q1<=0;
          else if (_______)
                    q1<=d;
          _______   q1<=q1+_____;
always @(posedge _______)
          cnt2=_______cnt2;              //二分频
assign load=(q1 == 8'b00000000);
_______ pm=_______;
endmodule
```

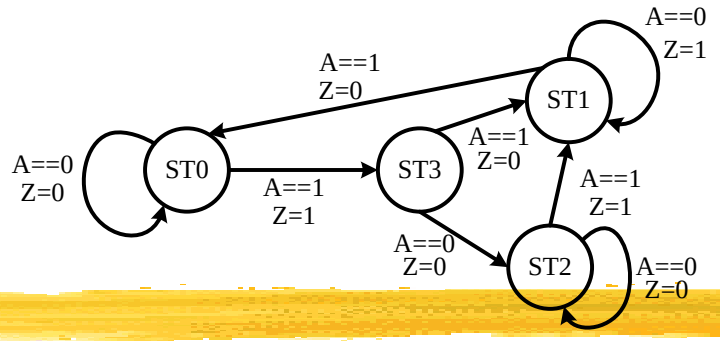
2.程序填空

选择以下Verilog语句片段填入空格，以实现完整的程序功能，注意每个选项只能选一次。

A. [7:0] B. posedge C. cnt2 D. load E. wire F. reg G. 1 H. input
I. else J. assign K. pm L. ! M. negedge N. clk O. load P. if

```
// 以下是一个50%占空比的8位数控分频器设计，请填空完成程序：
module fdiv8 (clk,rstn,d,__K__);
        input __N__,rstn;
        __H__ [7:0] d;
        output pm;
        __F__ cnt2;
        reg __A__ q1;
        __E__ load;
always @ (__B__ clk or __M__ rstn)
        __P__ (!rstn)
                q1<=0;
        else if (__D__)
                q1<=d;
        __I__ q1<=q1+__G__;
always @(posedge __O__)
        cnt2=__L__cnt2;                 //二分频
assign load=(q1 == 8'b00000000);
__J__ pm=__C__;
endmodule
```

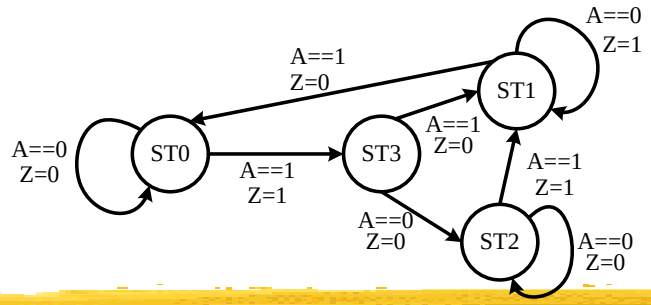
2.程序填空



A. parameter ***B***. c_state ***C***. n_state = ST3 ***D***. ST3=2 ***E***. n_state = ST0 ***F***. n_sate = ST1
G. n_state ***H***. c_state <= n_state ***I***. Z=0 ***J***. Z=1 ***K***.A ***L***. endcase ***M***. end
N. output ***O***. reg ***P***. **module** ***Q***. [1:0] ***R***. ST0 ***S***. ST2 ***T***. **;**

```
____ Mealy_FSM (A, clock, Z)____
 input A, clock;
 ____Z;
 ____Z;
______ ST0=0, ST1 =1,_____;
 reg_____ c_state,_____;
 always @(posedge clock)
   _____;
always@(c_state or ____)
   begin
    case(_____)
     ST0:
       if(A) begin
        Z =1;
        _____;
       end
       else  begin
_____;
        n_state =__  __;
       end
     ST1:
       if(A) begin
        Z =0;
        _____;
       end
     else begin
        _____;
        n_state = ST1;
       end
     ____:
       if(A) begin
        Z =1;
        n_state = ST1;
       end
     else begin
        Z=0;
        n_state = ST2;
       end
     ST3:
       begin
        Z=0;
        if(A)
         _____;
        else
         n_state = ST2;
       end
      _____
     _____
endmoule
```

2.程序填空



A. parameter ***B***. c_state ***C***. n_state = ST3 ***D***. ST3=2 ***E***. n_state = ST0 ***F***. n_sate = ST1
G. n_state ***H***. c_state <= n_state ***I***. Z=0 ***J***. Z=1 ***K***.A ***L***. endcase ***M***. end
N. output ***O***. reg ***P***. **module** ***Q***. [1:0] ***R***. ST0 ***S***. ST2 ***T***. **;**

```
 P   Mealy_FSM (A, clock, Z)__T
  input A, clock;
  __N_Z;
  ___O__Z;
 ___A___ ST0=0, ST1 =1,___D___;
  reg__Q___ c_state,__G_________;
  always @(posedge clock)
    ______H_______;
 always@(c_state or __K__)
    begin
     case(___B___)
       ST0:
         if(A) begin
           Z =1;
           __C__;
         end
         else  begin
___I____;
           n_state =__R__;
         end
       ST1:
         if(A) begin
           Z =0;
           ____E_____;
         end
       else begin
           ____J_____;
           n_state = ST1;
         end
       _S_:
         if(A) begin
           Z =1;
           n_state = ST1;
         end
       else begin
           Z=0;
           n_state = ST2;
         end
       ST3:
         begin
           Z=0;
           if(A)
             ____F_____;
           else
             n_state = ST2;
         end
        ____L____
       ___M____
      endmoule
```

3.程序改错题

```
1   module err1(CLK,LD,RST,DI,DO);
2       input CLK, LD, RST, [3:0] DI;
3       output DO;
4       reg [3:0] REG;
5       always@(posedge CLK or negedge RST)
6           if (!RST) REG<=0;
7               else if (LD) REG<=DI;
8                   else REG [2:0]<=REG [3:1];
9       assign DO<=REG[0];
10  endmodule
```

上面的程序有两处错误
(1) Error (10170): Verilog HDL syntax error, expecting an identifier

错误（1）行号：<u>第 行</u> 应修改为：
错误（2）行号：<u>第 行</u> 应修改为：
该程序的功能为：

3.程序改错题

```
1   module err1(CLK,LD,RST,DI,DO);
2       input CLK, LD, RST, [3:0] DI;
3       output DO;
4       reg [3:0] REG;
5       always@(posedge CLK or negedge RST)
6           if (!RST) REG<=0;
7               else if (LD) REG<=DI;
8                   else REG [2:0]<=REG [3:1];
9       assign DO<=REG[0];
10  endmodule
```

上面的程序有两处错误
(1) Error (10170): Verilog HDL syntax error, expecting an identifier

错误（1）行号：第2行 应修改为：input CLK, LD, RST; input [3:0] DI;
错误（2）行号：第9行 应修改为：assign DO=REG[0];
该程序的功能为：具备同步预置异步清零功能的4位右移移位寄存器

3.程序改错题

```
1   module err2(clk,clr,dir,cnt);
2       input clk, clr, dir;
3       output [n:0] cnt;
4       wire [n:0] q;
5       parameter n<=3;
6       always @(posedge clk, posedge clr)
7           if(clr) q <= 0;
8           else begin
9               if(dir) q <= q + 1;
10              else q <= q - 1; end
11      assign cnt=q;
12  endmodule
```

上面的程序有两处错误

(1) Error (10137): Verilog HDL Procedural Assignment error : object "q" on left-hand side of assignment must have a variable data type

错误（1）行号：<u>第 行</u> 应修改为：

错误（2）行号：<u>第 行</u> 应修改为：

该程序的功能为：

3.程序改错题

```
1   module err2(clk,clr,dir,cnt);
2       input clk, clr, dir;
3       output [n:0] cnt;
4       wire [n:0] q;
5       parameter n<=3;
6       always @(posedge clk, posedge clr)
7           if(clr) q <= 0;
8           else begin
9               if(dir) q <= q + 1;
10              else q <= q - 1; end
11      assign cnt=q;
12  endmodule
```

上面的程序有两处错误

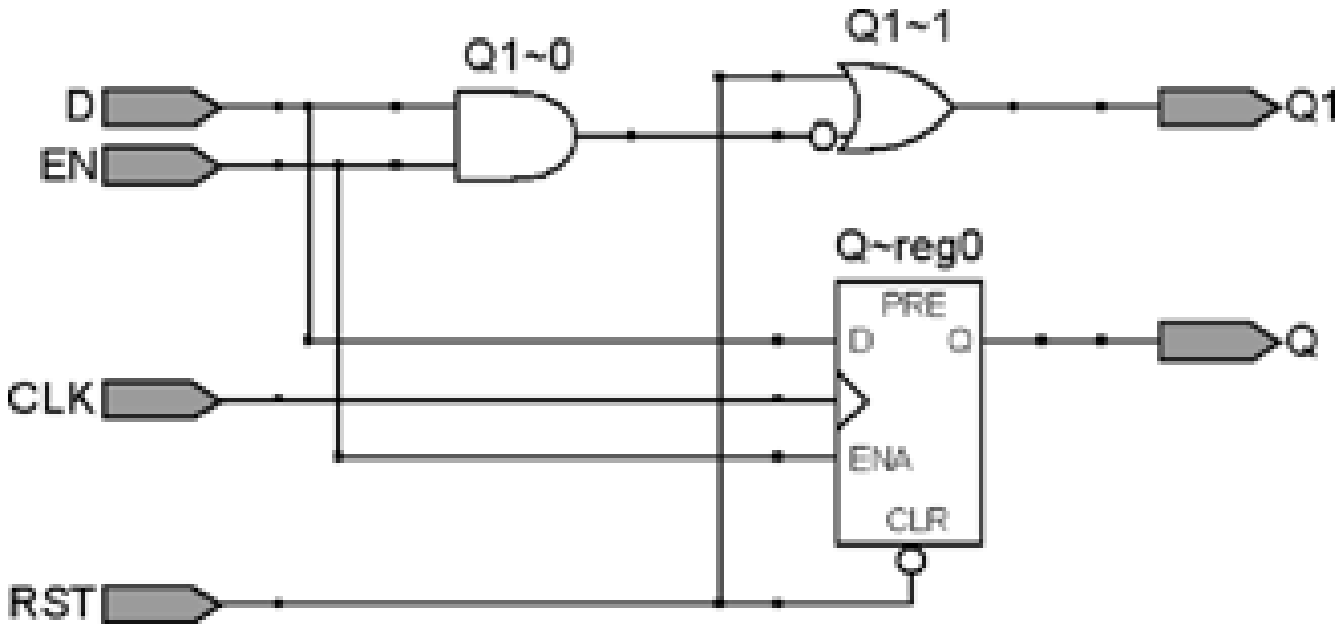
(1) Error (10137): Verilog HDL Procedural Assignment error : object "q" on left-hand side of assignment must have a variable data type

错误（1）行号：第4行　　应修改为：reg [n:0] q

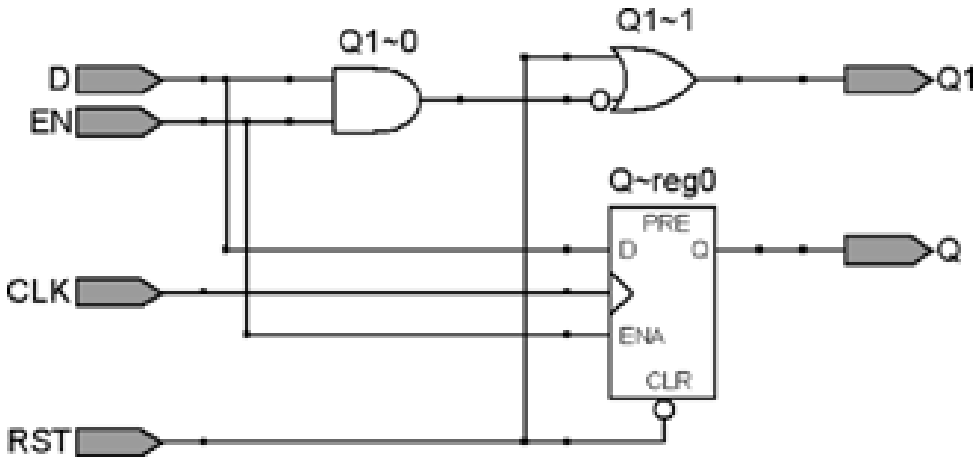
错误（2）行号：第5行　　应修改为：parameter n=3;

该程序的功能为：含异步清零的4位加/减计数器

4.根据RTL图写Verilog程序

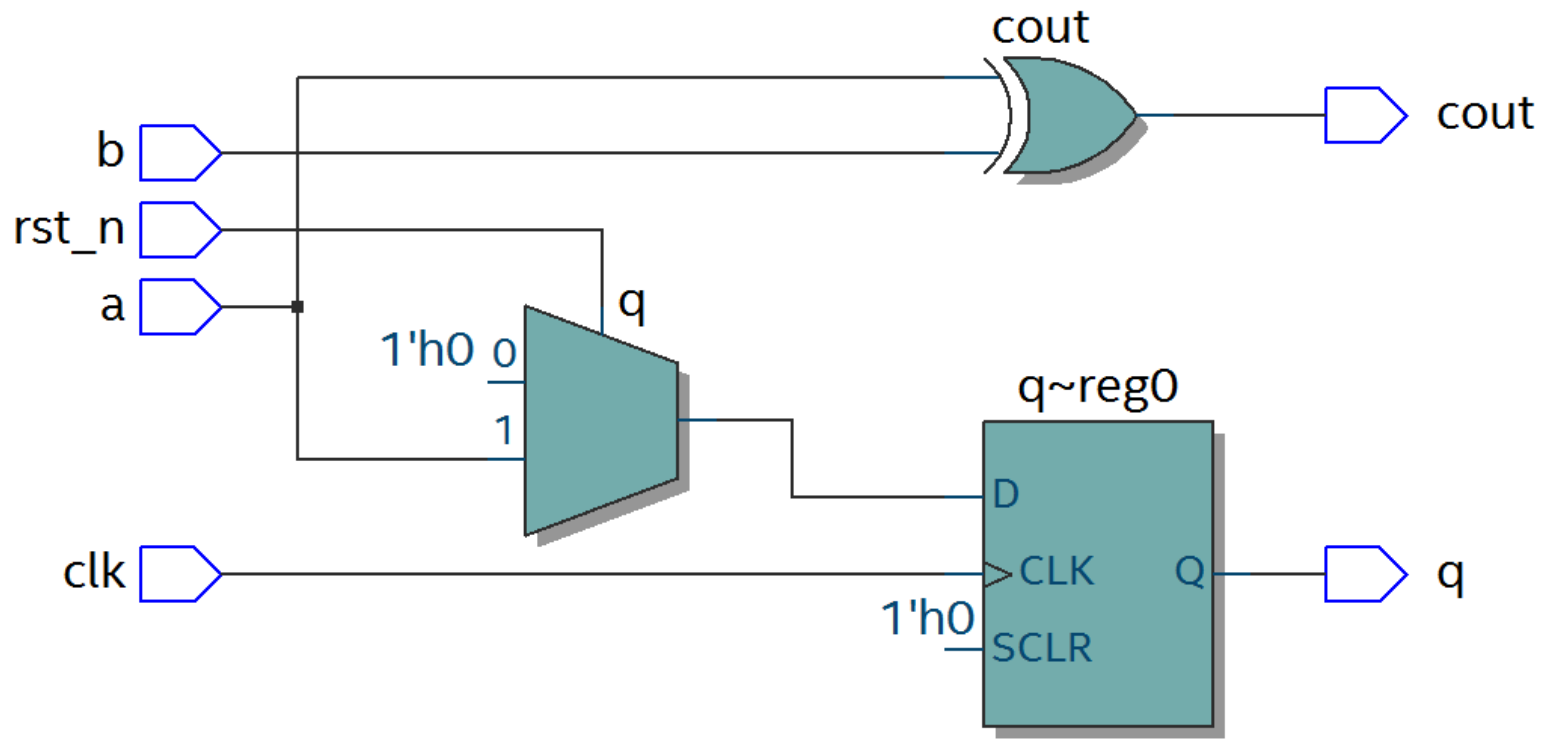


4.根据RTL图写Verilog程序

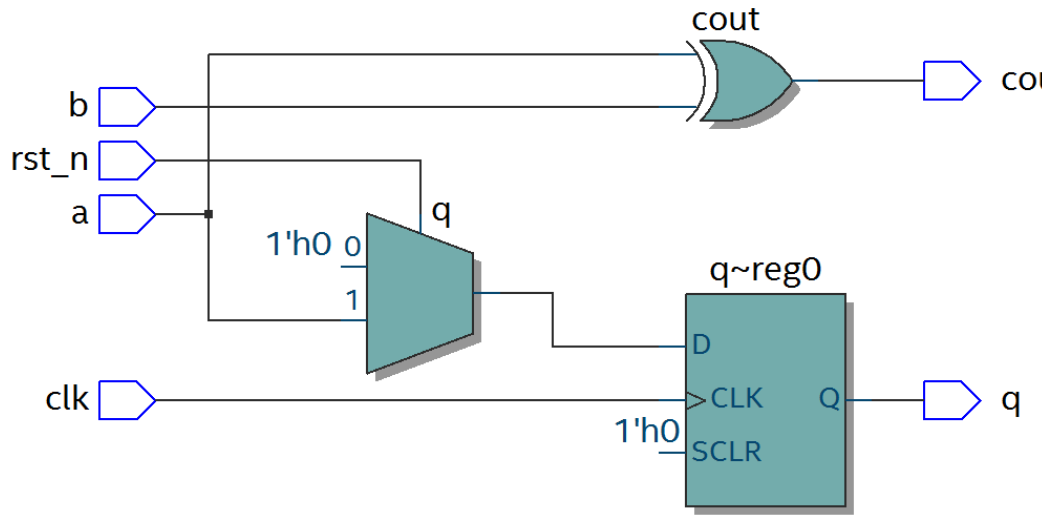



```
module test(CLK,RST,EN,D,Q,Q1);
    input CLK,RST,EN,D;
    output Q,Q1;

    wire Q10;
    reg Q;

    assign Q10=D&EN;
    assign Q1 =(~Q10)|RST;

    always@(posedge CLK or negedge RST)
        begin
            if(!RST)
                Q<=0;
            else if(EN)
                Q<=D;
        end

endmodule
```

4.根据RTL图写Verilog程序

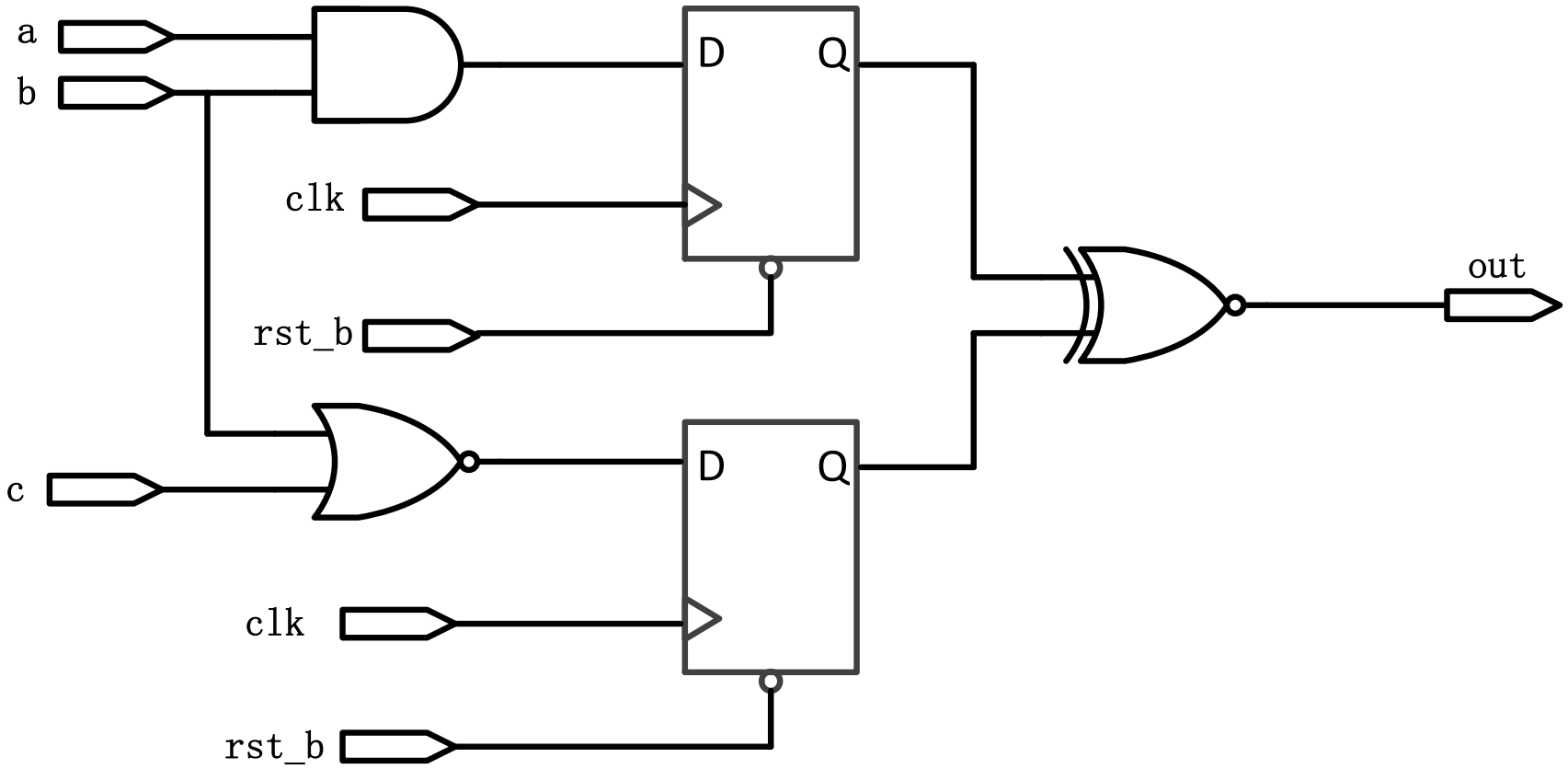


4.根据RTL图写Verilog程序

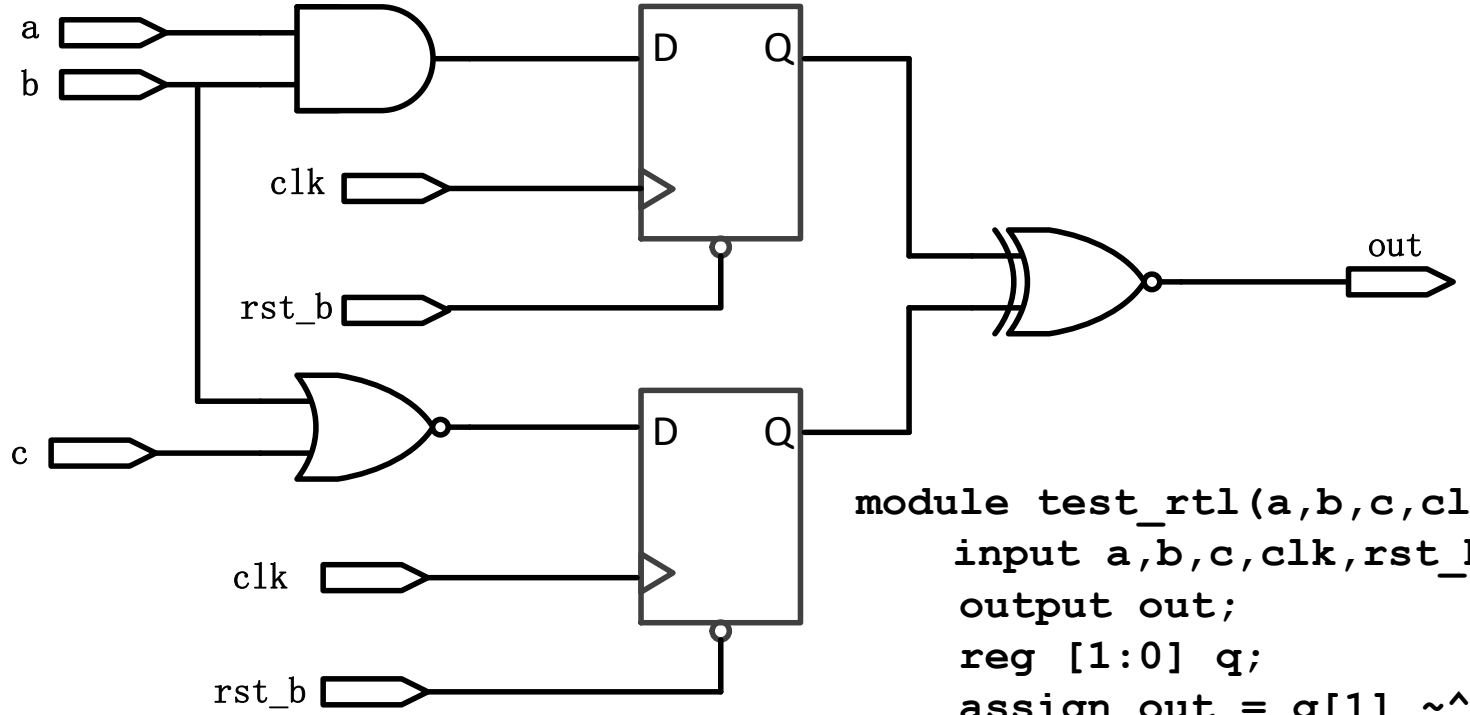



```
module test2(clk,a,b,rst_n,q,cout);
   input clk,a,b,rst_n;
   output q,cout;

   reg q;

   always@(posedge clk)
      begin
         if(!rst_n)
            q<=0;
         else
            q<=a;
      end

   assign cout=a^b;

endmodule
```

4.根据RTL图写Verilog程序

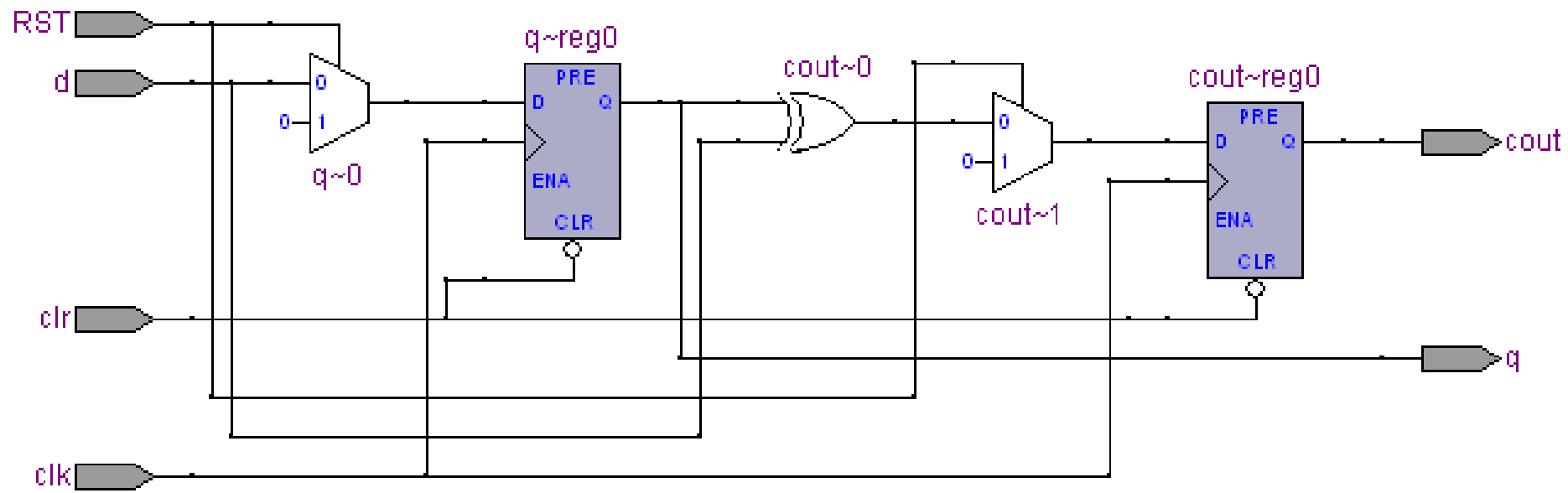


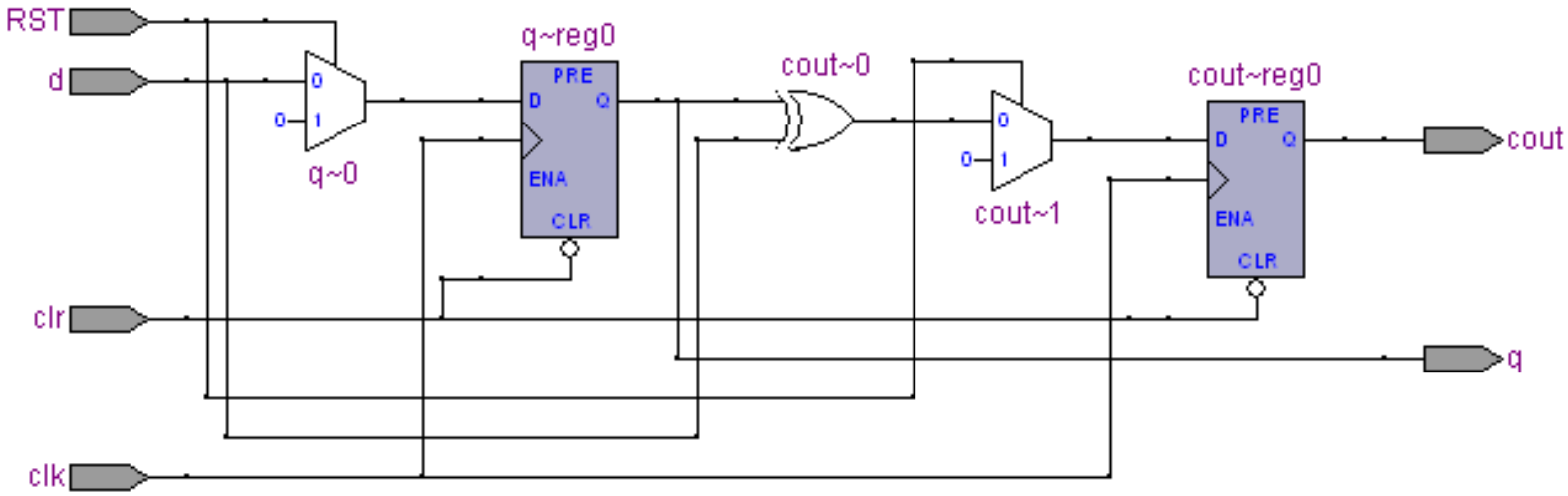
4.根据RTL图写Verilog程序




```
module test_rtl(a,b,c,clk,rst_b,out);
    input a,b,c,clk,rst_b;
    output out;
    reg [1:0] q;
    assign out = q[1] ~^ q[0];
    always@(posedge clk, negedge rst_b)
        if(!rst_b)
            q <= 0;
        else
            q <= {a&b,~(b|c)};
endmodule
```

4.根据RTL图写Verilog程序

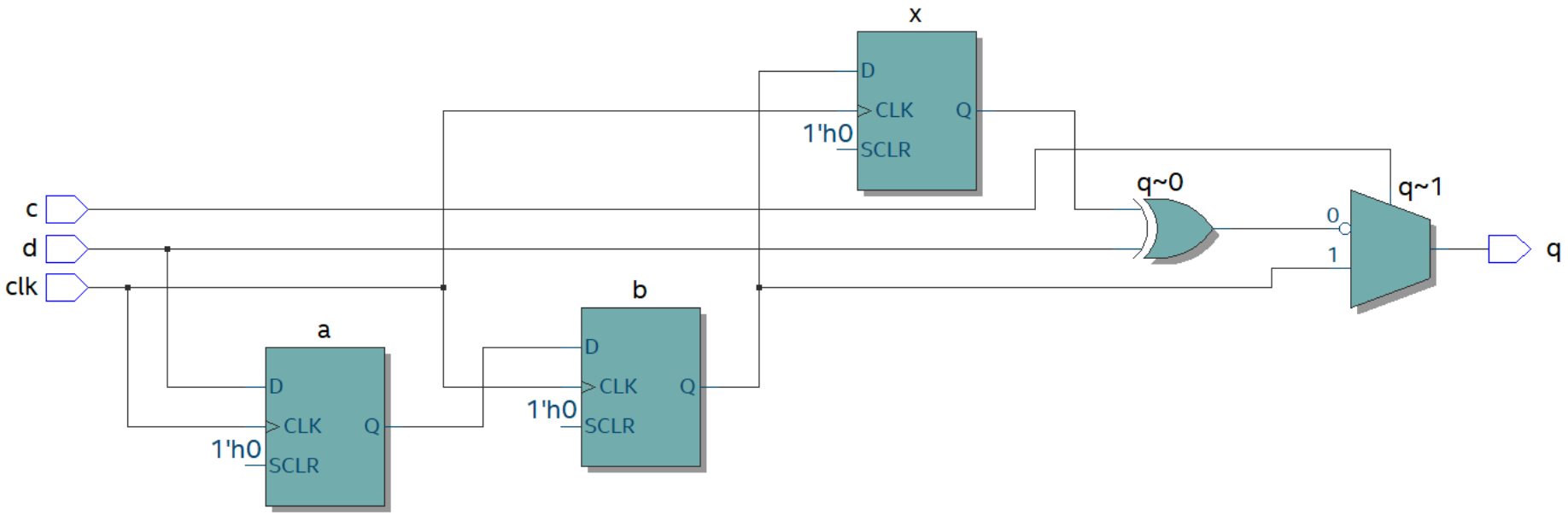





```
module abc(q,cout,d,RST,clk,clr); // 1分，包括module name
input d,clk,clr,RST;              // 1分，包括所有端口信号
output q;
output cout;
reg q,cout;              // 1分，无意义的reg不给分
always@(posedge clk or negedge clr) // 1分 加了RST无得分
begin
if (!clr) begin q<=0; cout<=0; end  // 1分
else if (RST)                       // 1分
begin q<=0;    cout <=0; end  // 1分
else
begin
q<=d;                          // 1分
cout <= d^q;                   // 1分
end
end
endmodule                      // 1分
```

5.根据Verilog程序，画出RTL图

```
module test(clk,d,q,c);
    input clk,c,d;
    output q;
    reg a;
    reg b,x;
    reg q;
always @(posedge clk) begin
    a <= d;
    b <= a;
    x <= b;
end
always@(c,b) begin
    if (c)      q<=b;
    else q<=x~^d;
end
endmodule
```




```
module test(clk,d,q,c);
    input clk,c,d;
    output q;
    reg a;
    reg b,x;
    reg q;
always @(posedge clk)
begin
    a <= d;
    b <= a;
    x <= b;
end
always@(c,b)
begin
    if(c) q<=b;
    else q<=x~^d;
end
endmodule
```

RTL Schematic ←→ Verilog HDL

三态门

双向端口

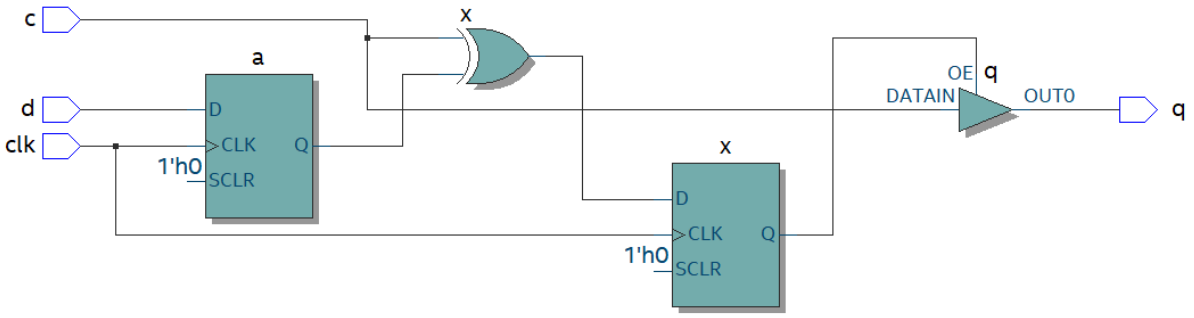
MCU的IO结构

5.根据Verilog程序，画出RTL图

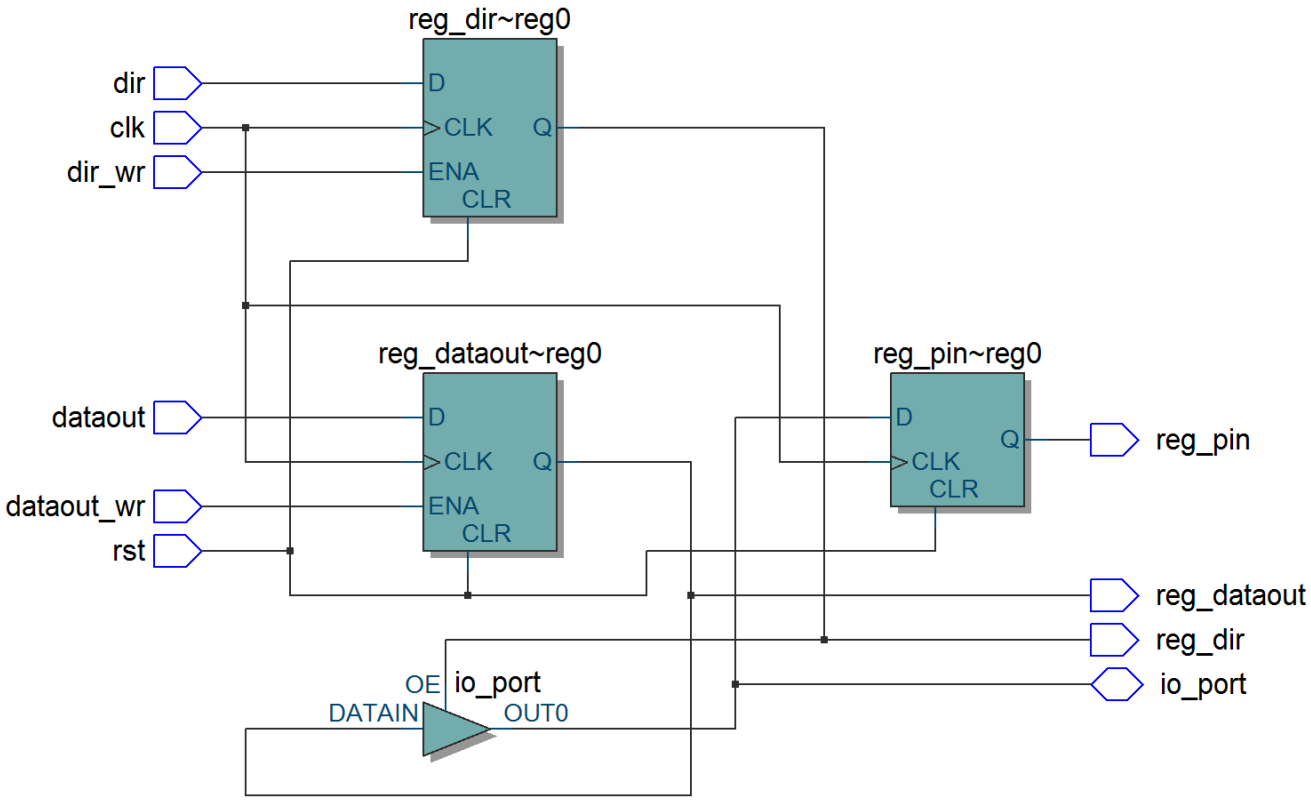
```
module test2(clk,d,q,c);
    input clk,c,d;
    output q;
    reg a;
    reg b,x;
always @(posedge clk) begin
    a <= d;
    x <= a^c;
end
assign q = x ? b : 1'bz;

endmodule
```

5.根据Verilog程序，画出RTL图

```
module test2(clk,d,q,c);
    input clk,c,d;
    output q;
    reg a;
    reg b,x;
always @(posedge clk) begin
    a <= d;
    x <= a^c;
end
assign q = x ? c : 1'bz;

endmodule
```



```
module  pio_x
(           input rst,
            input clk,
            input dataout,
            input dir,
            input dataout_wr,
            input dir_wr,
            output reg reg_dataout,
            output reg reg_dir,
            output reg reg_pin,
            inout io_port);
always@(posedge clk or posedge rst)
begin
            if(rst)     begin
                        reg_dataout <= 0;
                        reg_dir <= 0;
                        reg_pin <= 0;       end
            else  begin
                        if(dataout_wr) reg_dataout <= dataout;
                        if(dir_wr) reg_dir <= dir;
                        reg_pin <= io_port;         end
end
assign io_port = (reg_dir) ? reg_dataout : 1'bz;
endmodule
```



6.编程题

- 6.1 乘法器
- 6.2 多路选择器/复用器
- 6.3 数据分配器/解复用器
- 6.4 译码器
- 6.5 编码器
- 6.6 计数器
- 6.7 移位寄存器：串入串出、并入串出、串入并出、双向移位
- 6.8 分频器
- 6.9 状态机
- 6.10 计时器

6.1 乘法器

设计一个32位乘加器，R=R+A*B，其中A、B、R均为32位，一个clk周期内完成

```
module muladd(clk,rstn,A,B,R);//模块定义
      input clk,rstn;
      input [31:0] A,B;
      output [31:0] R;                    //端口定义
      reg [31:0] R;                       //数据类型定义
      always @ (posedge clk or negedge rstn)
//过程及敏感信号
            if (!rstn) R<=0;
              else R<=R+A*B;//乘法，累加，单时钟完成
endmodule
```

6.2 多路选择器/复用器

设计一个8选1多路选择器

```
module mux81s
(
    input [2:0] s,
    input [7:0] a,
    output y
);
assign y = a[s];
endmodule
```

6.3 数据分配器/解复用器

设计一个**1-8**数据分配器

```
module demux8a
(
    input [2:0] s,
    input a,
     // Input
    output [7:0] y
);

assign y = a << s;
endmodule
```

6.4 N=16的4-16译码器

设计一个**N=16、带极性控制的4-16译码器**

```
module decoder_x
#(
    parameter NA =4,
    parameter N = 16
)
(   input [NA-1:0] a,
    input p,                    // Polarity
    output [N-1:0] y
);
wire [N-1:0] temp_y;
assign temp_y = 1'b1 << a;
assign y = (p) ? temp_y : ~temp_y;
endmodule
```

6.5 8-3线编码器

```
module coder83(i,y);
    input [7:0] i;
    output [2:0] y;
    reg [2:0]y;
    always@(i) begin
        case (i)
        8'b0000_0001:y=3'b000;
        8'b0000_0010:y=3'b001;
        8'b0000_0100:y=3'b010;
        8'b0000_1000:y=3'b011;
        8'b0001_0000:y=3'b100;
        8'b0010_0000:y=3'b101;
        8'b0100_0000:y=3'b110;
        8'b1000_0000:y=3'b111;
        default: y=3'b000;
        endcase
    end
endmodule
```

6.6 BCD码

8421BCD码就是十进制数的4位二进制码。BCD码加法是十进制加法，不存在A（4’b1010）~F(4’b1111)这几种状态，超过9随即进一位，相当于二进制加6。

一位BCD码: 0~9
Hexadecimal --> BCD:
If Hexadecimal <10, BCD=Hexadecimal;
If Hexadecimal >=10, BCD=Hexadecimal+6

BCD码加法器设计

【例 3-14】

```
module BCD_ADDER (A,B,D) ;
  input [7:0] A,B;   output [8:0] D;
  wire [4:0] DT0, DT1 ; reg [8:0] D; reg S;
 always@ (DT0)
     begin  if (DT0[4:0] >= 5'b01010 )
    //如果低位 BCD 码的和大于等于 10,则使和加上 6，且有进位，使进位标志 S 等于 1。
              begin D[3:0] = (DT0[3:0]+4'b0110); S=1'b1; end
               else  begin D[3:0] = DT0[3:0] ; S=1'b0; end
     end //否则，将低位值赋予低位 BCD 码 D[3:0]输出，无进位，使进位标志 S 等于 0。
 always@ (DT1)   begin
     if (DT1[4:0]>=5'b01010)
    begin D[7:4] = (DT1[3:0]+4'b0110); D[8]=1'b1; end
       else begin D[7:4] = DT1[3:0] ; D[8]=1'b0;  end      end
   assign DT0 = A[3:0] + B[3:0] ;    //设没有来自低位的进位。
   assign DT1 = A[7:4] + B[7:4] + S; //S 是来自低位 BCD 码相加的进位。
endmodule
```

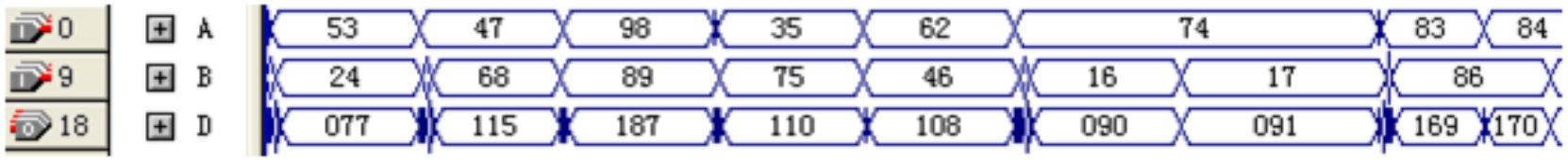


图 3-12 例 3-14 的仿真波形

6.6 BCD码加法器设计

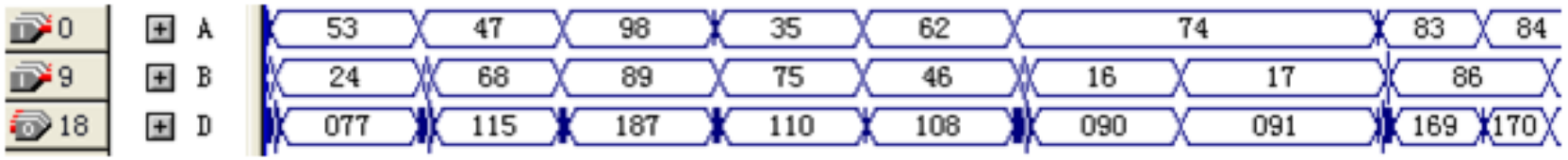


图 3-12 例 3-14 的仿真波形

A= h53 = 0101 0011
B= h24 = 0010 0100
A+B = 0111 0111 =77

A= h47 = 0100 0111
B= h68 = 0110 1000
A+B = 1011 1111 + 0110 0110
= 1 0001 0101 = 115

6.6 设计4位BCD计数器

带异步复位、同步装载、同步使能信号

```
module bcd_4d_cnt(    //4位十进制计数器
    input clk,
    input reset_n,
    input en,
    input load,
    input [15:0] d,
    output reg [15:0] bcd
);

always @ (posedge clk or negedge reset_n)
    if(!reset_n)
        bcd <= 0;
    else if(load)
        bcd <= d;
    else if(en)
        if(bcd[3:0] < 9) bcd[3:0] <= bcd[3:0] + 1'b1;
        else if(bcd[7:4] < 9) begin bcd[7:4]<=bcd[7:4] + 1'b1;bcd[3:0]<=0; end
        else if(bcd[11:8] < 9) begin bcd[11:8]<=bcd[11:8] + 1'b1;bcd[7:0]<=0; end
        else if(bcd[15:12] < 9) begin bcd[15:12]<=bcd[15:12] + 1'b1;bcd[11:0]<=0; end
        else bcd <= 0;
endmodule
```

TestBench

Clock

Reset

时序生成

$finish、$stop

打印输出

TestBench

```
module bcd_4d_cntx_tb;
reg clk;
reg reset_n;
reg en;
reg load;
reg [15:0] d;
wire [15:0] bcd;

bcd_4d_cntx bcd_4d_cntx_inst
(
        .clk(clk) ,
        .reset_n(reset_n) ,
        .en(en) ,
        .load(load) ,
        .d(d) ,
        .bcd(bcd)
);
inital
begin
        clk = 1'b0;
        forever #5 clk = ~clk;
end
```

```
initial
begin
        reset_n = 1'b0;
        en = 1'b0;
        load = 1'b0;
        d = 16'h0000;
        #105 reset_n = 1'b1;

        #10 d = 16'h1234;
        load = 1'b1;
        #10 load = 1'b0;
        #10 en = 1'b1;
        #10000000 en = 1'b0;

        #10 d = 16'h9876;
        load = 1'b1;
        #10 load = 1'b0;
        #10 en = 1'b1;
        #10000000 $stop;
end

endmodule
```

6.6 计数器

设计一个带异步复位、同步使能、同步清零、同步置位、同步装载的N=16位二进制计数器

```
module cntn
#(
    parameter N = 16
)
(
    input clk,
    input rst_n,
    input [N-1:0] d,
    input en,
    input load,
    input sclr,
    input sset,
    output reg [N-1:0] q
);

always @(posedge clk, negedge rst_n)
    if(!rst_n)
        q <= 0;
    else if(en) begin
        if(sclr) q <= 0;
        else if(sset) q <= {N{1'b1}};
        else if(load) q <= d;
        else
            q <= q + 1'b1;
    end

endmodule
```

6.6 Johnson计数器

设计一个具备异步清零的4位Johnson计数器。其计数行为是：
若当前计数器最高位为0，则执行最低位补1的左移操作；
若当前计数器最高位为1，则执行最低位补0的左移操作。
如 000->001-->011-->111-->110-->100-->000-->001 ...

```
module CNT_JS(clk, rstn, en, cnt_js);
    input clk,rstn,en;
    output reg [3:0] cnt_js;

    always@(posedge clk or negedge rstn) begin
        if (!rstn)
            cnt_js <= 4'b0;
        else begin
            if (en) begin
                if (cnt_js[3])
                    cnt_js <= {cnt_js[2:0],1'b0};
                else
                    cnt_js <= {cnt_js[2:0],1'b1};
                end
            else
                cnt_js <= cnt_js;
            end
        end
endmodule
```

6.6 BCD计数器

设计一个5位BCD计数器，带异步复位、同步装载、同步使能信号

```
module bcdcnt(clk,rstn,load,en,data,q,c5);                  //1分，包括endmodule
        input clk,rstn,load,en;
        input [19:0] data;
        output [19:0] q;                                    //1分，正确定义20位的输入输出端口
        output c5;
        reg [3:0] q4,q3,q2,q1,q0;
        wire c0,c1,c2,c3,c4;
        always @(posedge clk, negedge rstn)
                if(!rstn) q0<=0;                            //1分，异步复位
                else if (load) q0<=data[3:0];               //1分，同步装载
                else if (en)                                //1分，同步使能
                        if(q0<9) q0<=q0+1;                  //1分，判断条件累加
                        else q0<=0;                         //1分，判断条件置零
        assign c0=(q0==9);
        always @(posedge clk, negedge rstn)
                if(!rstn) q1<=0;
                else if (load) q1<=data[7:4];
                else if (c0)
                        if(q1<9) q1<=q1+1;
                        else q1<=0;
        assign c1=({q1,q0}==2'h99);
```

6.6 BCD计数器

```
always @(posedge clk, negedge rstn)
        if(!rstn) q2<=0;
        else if (load) q2<=data[11:8];
        else if (c1)
                if(q2<9) q2<=q2+1;
                else q2<=0;
assign c2=({q2,q1,q0}==3'h999);
always @(posedge clk, negedge rstn)
        if(!rstn) q3<=0;
        else if (load) q3<=data[15:12];
        else if (c2)
                if(q3<9) q3<=q3+1;
                else q3<=0;
assign c4=({q3,q2,q1,q0}==4'h9999);
always @(posedge clk, negedge rstn)
        if(!rstn) q4<=0;
        else if (load) q4<=data[19:16];
        else if (c3)
                if(q4<9) q4<=q4+1;
                else q4<=0;                                  //2分，完成功能
assign c5=({q4,q3,q2,q1,q0}==5'h99999);                      //1分，最终输出正确
endmodule
```

6.7 4位串入串出移位寄存器

```
module siso41(clk,din,dout);
    input clk,din;
    output reg dout;
    reg [3:0] q;
    always@(posedge clk) begin
        q[0] <= din;
        q[3:1] <= q[2:0];
        dout <= q[3];
        end
endmodule
```

6.7 4位并入串出移位寄存器

```
module piso41(clk,clr,din,dout);
    input clk,clr;
    input [3:0] din;
    output reg dout;
    reg [1:0] cnt;
    reg [3:0] q;
    always@(posedge clk) begin
        cnt <= cnt+1;
        if (clr)
            q<= 4'b0000;
        else begin
            if (cnt>0)
                q [3:1] <= q [2:0];
            else if (cnt == 2'b00)
                q<= din;
            end
        dout <= q[3];
        end
endmodule
```

6.7 16位串入并出移位寄存器

设计一个**16**位的串入并出右移移位寄存器

```
module shifter_s1p16(    //串行右移转并行输出
    input clk,
    input reset_n,
    input serial_in,
    output reg [15:0] parallel_out
);

    always@(posedge clk or negedge reset_n)
        if(!reset_n)
            parallel_out <= 0;
        else
            parallel_out <= {serial_in, parallel_out[15:1]};
endmodule
```

6.7 4位双向移位寄存器

设计一个4位的双向移位寄存器，可根据以下表格实现特定的功能控制。

输入控制			输出工作状态
rd_n	S	clk	q
0	任意	任意	异步清零
1	00	↑	保持
1	01	↑	右移
1	10	↑	左移
1	11	↑	并行置数

6.7 4位双向移位寄存器

设计一个4位的双向移位寄存器，可根据以下表格实现特定的功能控制。

输入控制			输出工作状态
rd_n	S	clk	q
0	任意	任意	异步清零
1	00	↑	保持
1	01	↑	右移
1	10	↑	左移
1	11	↑	并行置数

```
module shifter_194 (
    input rd_n ,
    input clk,
    input [0:3] d ,
    input dir ,
    input dil ,
    input [1:0] s ,
    output reg [0:3] q
);

always @ (posedge clk or negedge rd_n)
    if (rd_n == 1'b0)
        q <= 0;
    else
        case (s)
            2'b00 : q[0:3] <= q[0:3];          // 保持
            2'b01 : q[0:3] <= {dir, q[0:2]}; // 右移
            2'b10 : q[0:3] <= {q[1:3], dil}; // 左移
            2'b11 : q[0:3] <= d;               // 置数
          default : q[0:3] <= q[0:3];          // 保持
endcase
endmodule
```

6.8 分频数可控的分频器

设计一个分频数可控的分频器，可以通过输入信号控制分频的系数，将外部输入时钟信号分别进行2、4、6、8分频

```
module evenfdiv(clkout,clkin,rst,N);
    output clkout;
    input clkin;
    input rst;
    input [3:0]N;
    reg[2:0]cnt;
    reg clkout;
    always @(posedge clkin or negedge rst) begin
        if(!rst) begin
            cnt<=0;
            clkout<=0;
            end
        else begin
            if(cnt==N/2 - 1) begin
                clkout<=~clkout;
                cnt<=0;
                end
            else
                cnt<=cnt+1;
            end
        end
endmodule
```

6.9 状态机

已知状态机的状态转移如下表，请写出Moore类型状态机

状态	输入	条件满足	条件不满足	输出
St0 (rst==1)	A=1	St1	St0	Y=3'h1
St1	B=0	St2	St4	Y=3'h0
St2	A=0	St3	St1	Y=3'h5
St3	A=1	St4	St5	Y=3'h3
St4	B=1	St5	St2	Y=3'h6
St5		St0	St1	Y=3'h7

Moore类型状态机
状态图（要求会画）

已知状态机的状态转移如下表，请写出Moore类型状态机

状态	输入	条件满足	条件不满足	输出
St0 (rst==1)	A=1	St1	St0	Y=3'h1
St1	B=0	St2	St4	Y=3'h0
St2	A=0	St3	St1	Y=3'h5
St3	A=1	St4	St5	Y=3'h3
St4	B=1	St5	St2	Y=3'h6
St5		St0	St1	Y=3'h7

```
module mooretest (clk,rst,A,B,Y);
input A, B, clk, rst; output reg [11:0] Y;
parameter st0=0,st1=1,st2=2,st3=3,st4=4,st5=5;
reg [2:0] cst, nst;
always@(posedge clk or negedge rst) begin
if(!rst) cst<=st0;
else cst<=nst;
end
always@(cst or A or B) begin
case (cst)
st0: begin Y<=3'h1; if(A==1) nst<=st1; else nst<=st0; end
st1: begin Y<=3'h0; if(B==0) nst<=st2; else nst<=st4; end
st2: begin Y<=3'h5; if(A==0) nst<=st3; else nst<=st1; end
st3: begin Y<=3'h3; if(A==1) nst<=st4; else nst<=st5; end
st4: begin Y<=3'h6; if(B==1) nst<=st5; else nst<=st2; end
st5: begin Y<=3'h7; nst<=st0; end
default: nst<=st1;
endcase
end
endmodule
```

6.10 计时器、倒计时

- 60秒倒计时
- 0~99s计时
- 百分秒表（mm:ss:88）
- 时钟（hh:mm:ss:tt + 秒闪）

计时器

```
module xclock
#(
        parameter N = 50_000_000/100
)
(
        input clk,      // 50MHz
        input reset,
        input [3:0] d,
        input [3:0] addr,
        input en,
        input load,
        output [31:0] q,
        output p_secflash,
        output p_day
);
localparam NW = $clog2(N);
reg [NW-1:0] fcnt;
reg [7:0] ten_msec,sec,minu,hour;
wire p_ten_msec = (fcnt==N-1);
assign p_secflash = ((ten_msec >= 8'h49);
wire p_sec = (ten_msec==8'h99)&p_ten_msec;
wire p_minu = (q[15:0]==16'h5999)&p_ten_msec;
wire p_hour = (q[23:0]==24'h595999)&p_ten_msec;
assign p_day = (q==32'h23595900)&p_ten_msec;

assign q = {hour,minu,sec,ten_msec} ;

always @(posedge clk,posedge reset)
        if(reset) fcnt <= 0;
        else if(fcnt < N-1)
                fcnt <= fcnt + 1'b1;
        else
                fcnt <= 0;
```

```
always @(posedge clk,posedge reset)
        if(reset) ten_msec <= 0;
        else if(load)
        begin
                if(addr==0) ten_msec[3:0] <= d;
                if(addr==1) ten_msec[7:4] <= d;
        end
        else if(en&p_ten_msec)
                if(ten_msec>=8'h99) ten_msec <= 0;
                else if(ten_msec[3:0]>=4'h9) begin
ten_msec[7:4]<=ten_msec[7:4]+1'b1;ten_msec[3:0]<=0; end
                else ten_msec[3:0] <= ten_msec[3:0] + 1'b1;

always @(posedge clk,posedge reset)
        if(reset) sec <= 0;
        else if(load)
        begin
                if(addr==2) sec[3:0] <= d;
                if(addr==3) sec[7:4] <= d;
        end
        else if(en&p_sec)
                if(sec>=8'h59) sec <= 0;
                else if(sec[3:0]>=4'h9) begin
sec[7:4]<=sec[7:4]+1'b1;sec[3:0]<=0; end
                else sec[3:0] <= sec[3:0] + 1'b1;
```

```
always @(posedge clk,posedge reset)
                if(reset) minu <= 0;
                else if(load)
                begin
                                if(addr==4) minu[3:0] <= d;
                                if(addr==5) minu[7:4] <= d;
                end
                else if(en&p_minu)
                                if(minu>=8'h59) minu <= 0;
                                else if(minu[3:0]>=4'h9) begin
minu[7:4]<=minu[7:4]+1'b1;minu[3:0]<=0; end
                                else minu[3:0] <= minu[3:0] + 1'b1;


always @(posedge clk,posedge reset)
                if(reset) hour <= 0;
                else if(load)
                begin
                                if(addr==6) hour[3:0] <= d;
                                if(addr==7) hour[7:4] <= d;
                end
                else if(en&p_hour)
                                if(hour>=8'h23) hour <= 0;
                                else if(hour[3:0]>=4'h9) begin
hour[7:4]<=hour[7:4]+1'b1;hour[3:0]<=0; end
                                else hour[3:0] <= hour[3:0] + 1'b1;



endmodule
```

计时器_tb

```
`timescale 1ns/100ps
module xclock_tb();
reg clk;          // 50MHz
reg reset;
reg [3:0] d;
reg [3:0] addr;
reg en;
reg load;
wire [31:0] q;
wire p_secflash;
wire p_day;

initial
begin
                clk = 1'b0;
                forever #10 clk = ~clk;
end

initial
begin
                reset = 1'b1;
                d = 0;
                load = 1'b0;
                en = 1'b0;
                addr = 0;
                #30 reset = 1'b0;
                en = 1'b1;
                #120000 en = 1'b0;

                addr=4'd0;
                d = 4'h9;
                #10 load = 1'b1;
                #30 load = 1'b0;

                addr=4'd1;
                d = 4'h9;
                #10 load = 1'b1;
                #30 load = 1'b0;

                addr=4'd2;
                d = 4'h9;
                #10 load = 1'b1;
                #30 load = 1'b0;

                addr=4'd3;
                d = 4'h5;
                #10 load = 1'b1;
                #30 load = 1’b0;

                addr=4'd4;
                d = 4'h9;
                #10 load = 1'b1;
                #30 load = 1'b0;

                addr=4'd5;
                d = 4'h5;
                #10 load = 1'b1;
                #30 load = 1'b0;

                addr=4'd6;
                d = 4'h3;
                #10 load = 1'b1;
                #30 load = 1'b0;

                addr=4'd7;
                d = 4'h2;
                #10 load = 1'b1;
                #30 load = 1'b0;

                #20 en = 1'b1;

                #120000 $stop;
end
xclock #(.N(50)) dut_xclock
(
                .clk(clk) ,
                .reset(reset) ,
                .d(d) ,
                .addr(addr) ,
                .en(en) ,
                .load(load) ,
                .q(q) ,
                .p_secflash(p_secflash) ,
                .p_day(p_day)
);
endmodule
```

7.课内实验相关

7.1 模可控计数器（必考）

7.2 正弦波发生器（例化语句IP使用、DDS）

7.3 VGA图像显示（显示控制）

7.4 序列检测器（状态机、状态转换图）

7.5 乐曲演奏电路

7.1 设计一个模为60的BCD码加法计数器

```
module count60bcd
(
    input [7:0]data,
    input load, cin,clk, reset,
    output reg [7:0] qout,
    output cout
);
always@(posedge clk) begin              // c1k上升岩计数
    if (reset) qout<=0;
    else if (load)qout<= data;                          //同步复位else_if (cin)begin[/同步予数
                if (qout[3:0]==9) begin                 //低位是台为9,是则回0,并判断高位是含为5
                                qout[3:0]<=0;
                                if (qout[7:4]==5)
                                                qout[7:4]<=0;
                                else
                                                qout [7:4]<= qout [7:4]+1; //高位不为5,则加1
                                end
                else
                                qout[3:0]<=qout[3:0]+1; //低位不为9,则加1end
    end
assign cout = ((qout==8'h59) & cin);  //产生进位输出信号
endmodule
```

7.2 正弦波发生器设计

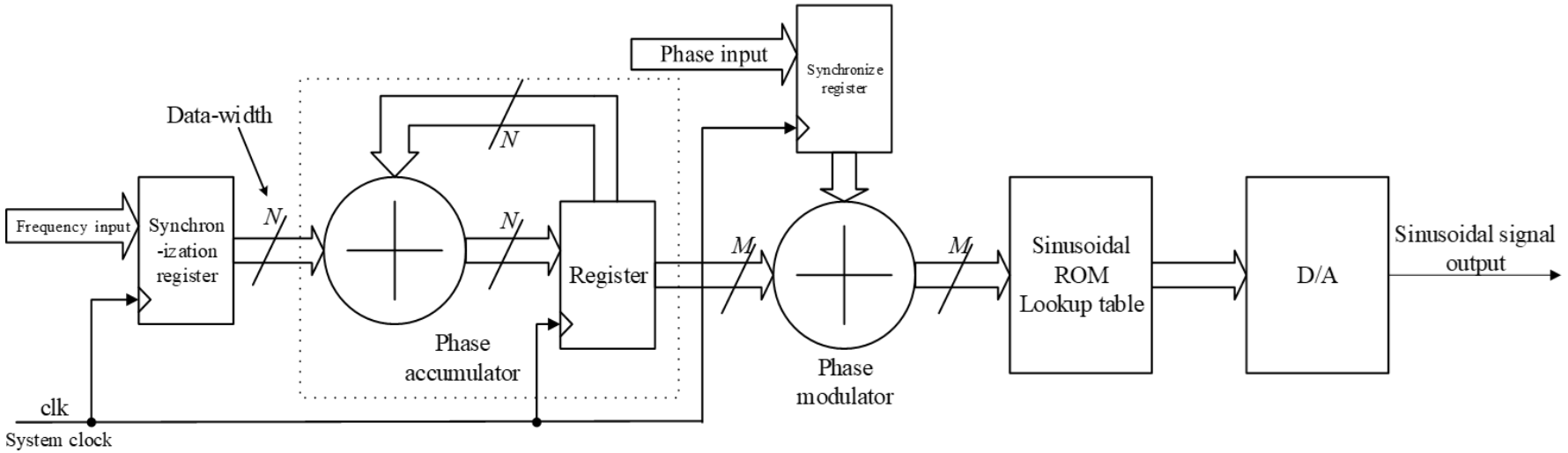


Figure: Basic DDS structure

7.2 正弦波发生器设计

```
module dds_top(
                    input clk,
                    input           rst_n,
                    input           [2:0] key
);

wire [23:0] FreqWord;
wire [23:0] PhaseWord;
wire [9:0] AmpWord;

assign FreqWord = {8'b0, key[0], 2'd1, 13'd0};
assign PhaseWord = {4'b0, key[1], 19'd0};
assign AmpWord = {1'b1, key[2], 8'd0};

dds_mult dds_mult_inst
(
                    .clk(clk) ,       // input  clk_sig
                    .rst_n(rst_n) ,   // input  rst_n_sig
                    .FreqWord(FreqWord) ,       // input [N-1:0] FreqWord_sig
                    .PhaseWord(PhaseWord) ,     // input [N-1:0] PhaseWord_sig
                    .AmpWord(AmpWord) ,         // input [MA-1:0] AmpWord_sig
                    .q_sin() ,        // output [W-1:0] q_sin_sig
                    .q_mult()         // output [W-1:0] q_mult_sig
);
endmodule
```

7.3 VGA简单图像显示控制模块设计

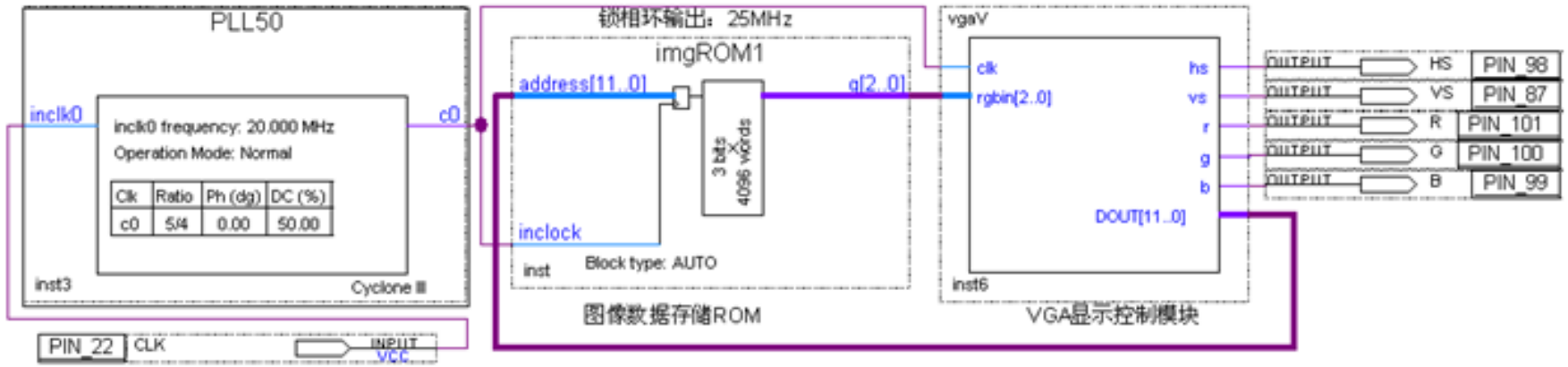


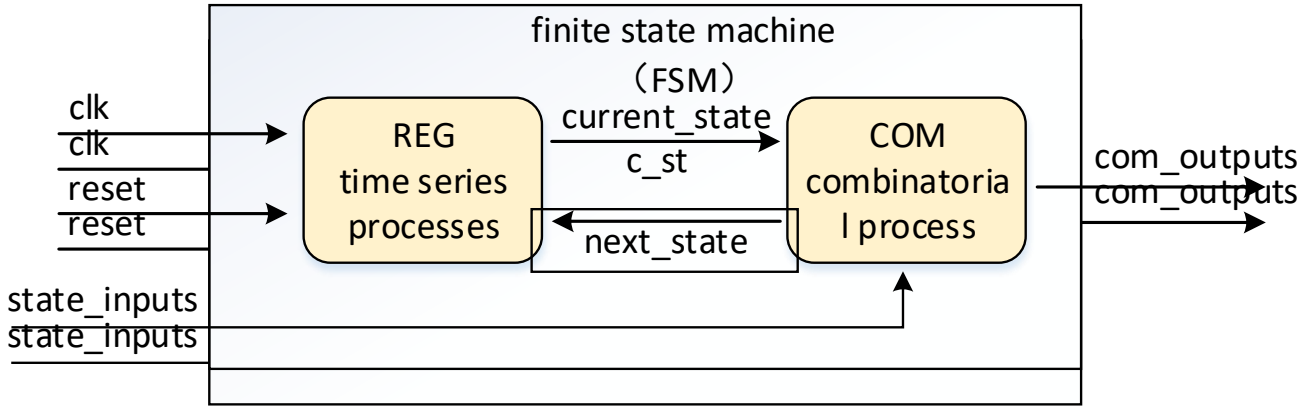
图 8-23 VGA 图像显示控制模块原理图

8-4 VGA简单图像显示控制模块设计

【例 8-33】

```
module vgaV (clk, hs, vs, r, g, b, rgbin, DOUT);
   input clk ;     //工作时钟 25MHz
   output hs,vs;   //场同步，行同步信号
   output r,g,b ;  // 红，绿，蓝信号，
   input[2:0] rgbin; //像素数据
   output[11:0] DOUT; //图像数据 ROM 的地址信号
   reg[9:0] hcnt, vcnt;      reg r,g,b;      reg hs,vs;
   assign DOUT = {vcnt[5:0], hcnt[5:0]} ;
   always @(posedge clk)  begin  //水平扫描计数器
      if (hcnt<800)   hcnt<=hcnt+1 ;
      else            hcnt<={10{1'b0}} ;
     end
   always @(posedge clk)   begin  //垂直扫描计数器
      if (hcnt==640+8)    begin
         if (vcnt<525)  vcnt<=vcnt+1 ;
         else    vcnt<={10{1'b0}} ;  end   end
   always @(posedge clk)   begin //场同步信号发生
      if ((hcnt>=640+8+8) & (hcnt<640+8+8+96))
        hs<=1'b0 ; else  hs<=1'b1 ;    end
   always @(vcnt)   begin   //行同步信号发生
      if ((vcnt>=480+8+2) & (vcnt<480+8+2+2))
        vs<=1'b0 ;  else  vs<=1'b1 ;  end
   always @(posedge clk)   begin
      if (hcnt<640 & vcnt<480)   //扫描终止
      begin  r<=rgbin[2] ;  g<=rgbin[1] ;  b<=rgbin[0];   end
      else begin  r<=1'b0; g<=1'b0;  b<=1'b0;  end
   end
endmodule
```

7.4 序列检测器



State Machine with Multiprocess Structure

```
module seq_detect(
                input           clk,
                   input        rst_n,
                input           data_in,
                output          wire  sout
);
parameter s0=0, s1=1, s2=2, s3=3, s4=4, s5=5, s6=6, s7=7, s8=8; //状态机的不同状态
reg  [3:0]                                  current_state;
reg  [3:0]                                  next_state;
always @(posedge clk or negedge rst_n) begin
                if (!rst_n)
                                current_state <= s0;
                else
                                current_state <= next_state;
end
always @(*) begin
                case(current_state)
                                s0 : if(data_in == 1'b1) next_state <= s1; else next_state <= s0;
                                s1 : if(data_in == 1'b1) next_state <= s2; else next_state <= s0;
                                s2 : if(data_in == 1'b1) next_state <= s3; else next_state <= s0;
                                s3 : if(data_in == 1'b0) next_state <= s4; else next_state <= s3;
                                s4 : if(data_in == 1'b1) next_state <= s5; else next_state <= s0;
                                s5 : if(data_in == 1'b0) next_state <= s6; else next_state <= s2;
                                s6 : if(data_in == 1'b0) next_state <= s7; else next_state <= s1;
                                s7 : if(data_in == 1'b0) next_state <= s8; else next_state <= s1;
                                s8 : if(data_in == 1'b0) next_state <= s0; else next_state <= s1;
                                default : next_state <= s0;
                endcase
end
assign sout = (current_state == s8);
endmodule
```

教材复习

EDA概述（掌握概念）

Verilog语法入门（熟练掌握编程）

- 程序结构、数据类型、行为语句、运算符与机构描述
- 组合电路设计/时序电路设计

EDA工具（实验）

- 仿真、FPGA硬件实现、IP应用

Verilog设计深入（三态门，设计优化）

状态机（Moore，会写）

TestBench（会写）