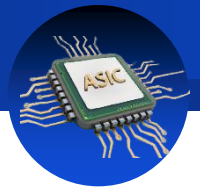
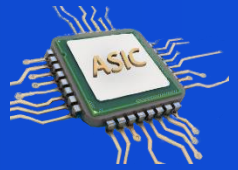
杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

数字系统与处理器



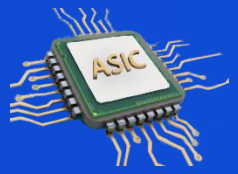
复习与习题课

课程考核方式



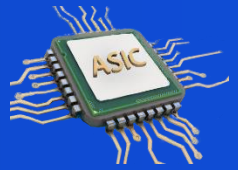
- 总评100分
- 平时40%
  - 线下线上作业（8分）
  - 课堂测试、课堂讨论（10分）
  - SPOC/MOOC线上成绩（20分）
  - 思政成绩（2分）
- 期末考试 60%

考试题型



- 单项选择题（15分，5小题）
  - 数字逻辑电路、EDA概念、设计优化
- 数字逻辑简答题（15分，3小题）
  - 卡诺图、逻辑表达式、最小项/最大项标准形式
  - 组合电路或时序电路设计（基本门、74138、74151、74153、74161、74595）
  - 真值表、逻辑图、时序波形图
- EDA简答题（10分，2小题）
  - EDA基本概念、设计优化、IP应用概念、FPGA结构原理、状态机编码、状态图
- 看Verilog画RTL 或 看RTL写Verilog（10分）
- 处理器单元电路设计（10分）
  - ALU、PC、寄存器组、移位器、比较器、三态总线、IO端口

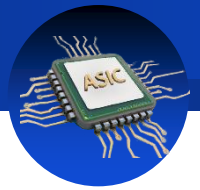
考试题型（续）



- Verilog组合电路设计题（10分）
  - 多路选择器、译码器、数据分布器、优先编码器等
  - 加法器、乘法器等算术电路
  - 复合上述电路的复杂组合电路
- Verilog时序电路设计题（10分）
  - 触发器、锁存器、移位寄存器、计数器、分频器、序列检测器等
  - 复合上述电路的复杂时序电路（含状态机）
- 综合设计题（20分）
  - 参看数字系统与处理器实验后5个实验例子（含扩展）

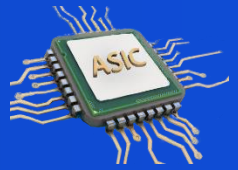
杭州电子科技大学
HANGZHOU DIANZI UNIVERSITY

数字系统与处理器



复习——数电部分

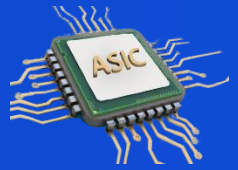
第1章 数字系统与处理器导论-续1



1. 基本逻辑运算、复合逻辑运算
2. 逻辑函数
3. 基本公式、常用公式
4. 三大规则
5. 逻辑函数的表述方法
6. 逻辑函数的最小项、最大项表达形式及互相转换
7. 逻辑函数的传统化简方法
8. 具有无关项的逻辑函数及其化简
9. Q-M化简方法

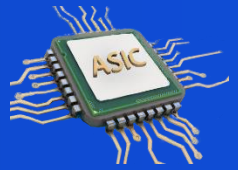
红字的为重点内容

第2章基本逻辑门与简单组合逻辑电路



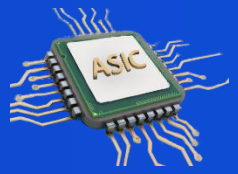
1、CMOS逻辑门
2、组合电路
3、组合电路分析
4、组合电路设计
5、编码器与译码器：74138
6、数据选择器与数据分配器：74151/74153
7、带输出高阻的特殊逻辑门
- 传输门
- 三态门
- OD门

第3章 组合逻辑电路的设计与硬件实现



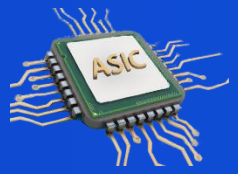
- PROM、PLA、PAL的区别
  - PLA、PAL需要看一下教材上的描述

第4章 数制表示与编码



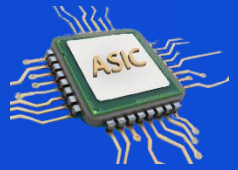
- 数制
  - 二进制/十进制/八进制/十六进制及其互相转换
- 码制
  - 原码/反码/补码的表示及运算
  - BCD码
  - 奇偶校验码/格雷码/条形码/二维码/ASCII编码
- 处理器ISA与机器码
- ~~浮点数表示与操作~~
  - ~~单精度与双精度~~
  - ~~定点数与浮点数的转换~~

第5章 运算逻辑电路设计



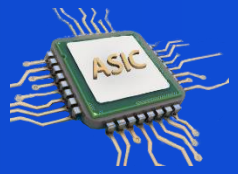
- 加法器设计
  - 半加器/全加器/多位数加法器
- 数值比较器
- 补码处理电路
  - 反码/补码电路
- 数字的显示
  - 数码管/动态扫描电路
- 广义译码器
- ALU设计
- ~~整数乘法器 Booth算法~~

第6章 基本时序电路元件与计数器



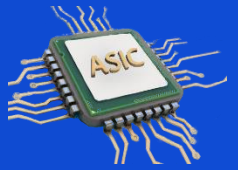
- 锁存器
  - D锁存器
- 单边沿触发器
  - D触发器
  - JK触发器
  - T/T'触发器
  - 触发器间的相互转换
- 时序波形图

第6章 基本时序电路元件与计数器



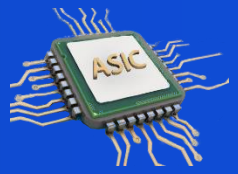
- 时序电路分析与设计
- 计数器:
  - 异步/同步计数器设计(二进制/非二进制）
  - 74161
- 分频器
  - 2分频器
  - 可预置分频器
- 寄存器组与移位寄存器
  - 寄存器组
  - 移位寄存器

第7章时序逻辑电路的设计与硬件实现



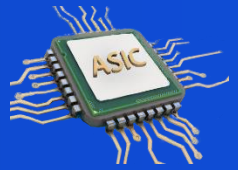
- 计数器通用设计模型
  - 计数器的一般结构模型
  - 普通二进制计数器设计讨论
  - BCD码计数器设计讨论
  - 模可控计数器设计讨论
  - 可逆计数器设计讨论

第8章 存储器



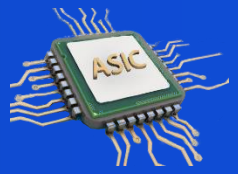
- 半导体存储器的分类
- 存储器指标
- 存储器的扩展
  - 存储器的位扩展（输出位宽扩展）
  - 存储器的字扩展（输入地址线扩展）
  - 存储器的字位扩展

第12章 D/A与A/D转换器



- D/A转换器
  - D/A转换的工作原理
  - 权电阻/倒T型电阻网络
  - 主要技术指标
- A/D转换器
  - A/D转换的工作过程
  - 主要技术指标
- DAC0832典型应用电路
- ADC0809典型应用电路

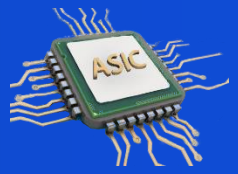
第10章 简单16位RISC处理器模型机



- RISC模型机的基本结构
  - 主要构成
  - ALU
  - 寄存器组
  - 程序计数器PC
  - 取指单元
  - 译码单元
  - 执行控制单元
  - 指令存储器
  - 数据存储器

非红字部分仅需要了解，红字部分能自行设计

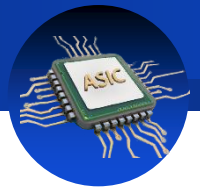
第10章 简单16位RISC处理器模型机



- 模型机的指令集ISA
  - 逻辑与运算指令
  - 分支指令
  - 数据取存指令
  - 无条件跳转指令
- 译码单元设计
- 指令存储器设计
- 数据存储器设计
- 程序计数器PC的设计

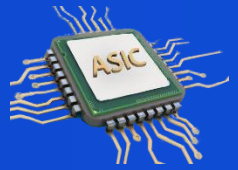
杭州電子科技大学
HANGZHOU DIANZI UNIVERSITY

数字系统与处理器



复习——EDA与CPU

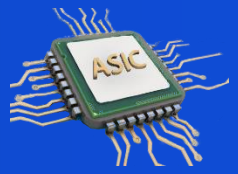
简答



- EDA中文全称是什么？

- 答：电子设计自动化

简答



- 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

↓

综合

↓

适配

↓

时序仿真（门级功能仿真与静态时序分析）

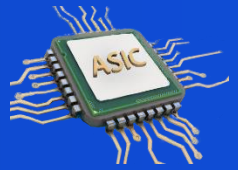
↓

下载（编程或配置）

↓

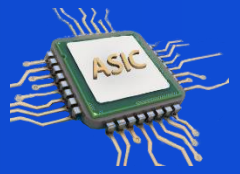
硬件测试

简答



- IP中文全称是？IP可分为哪几类?分别是什么意思？
- 答：知识产权核；
- 软IP、固IP、硬IP；
- 软IP：用Verilog/VHDL等硬件描述语言描述的功能块
- 固IP：完成了综合的功能块（门级网表）
- 硬IP：提供设计的最终掩模（版图）

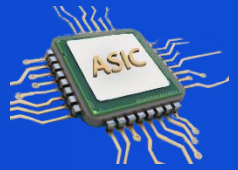
简答



- 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（与或阵列）可编程结构

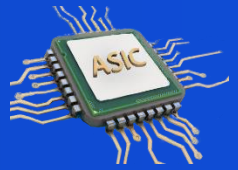
- 复杂PLD包括？基本的编程原理是基于？
- 答：CPLD和FPGA；
- CPLD基于**乘积项**逻辑（与或阵列）可编程结构，FPGA基于SRAM查找表逻辑可编程结构

简答



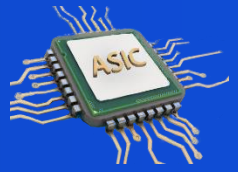
- Intel提供了一种嵌入式逻辑分析仪是？它基于什么技术构建？
- 答：SignalTap II，基于JTAG技术构建。

简答



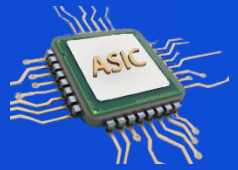
- 在Quartus中那些工具或者IP是使用了JTAG技术
- 答：
  - SignalTap
  - In-System sources and Probes
  - In-System Memory content Editor
  - ……

简答



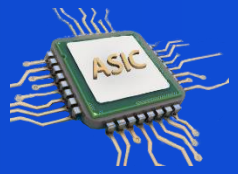
- IEEE的那个标准是JTAG的？JTAG主要有哪些信号？
- 答：
  - IEEE 1149
  - JTAG信号
    - TMS
    - TDO
    - TDI
    - TCK
    - TRST（可以不回答）

简答



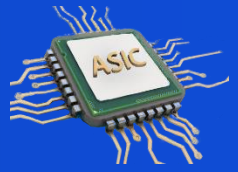
- Verilog HDL的IEEE标准号是什么？主要有那几个版本？
- 答：
  - IEEE Std 1364
    - IEEE Std 1364-1995
    - IEEE Std 1364-2001

简答



- SignalTap II, 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

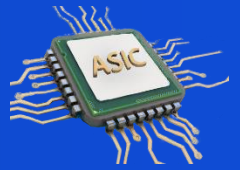
简答



- 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

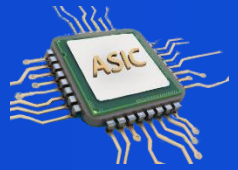
简答



- 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（顺序编码）：
- S0=00，S1=01，S2=10，S3=11；
- 一位热码：S0=0001，S1=0010，S2=0100，S3=1000；

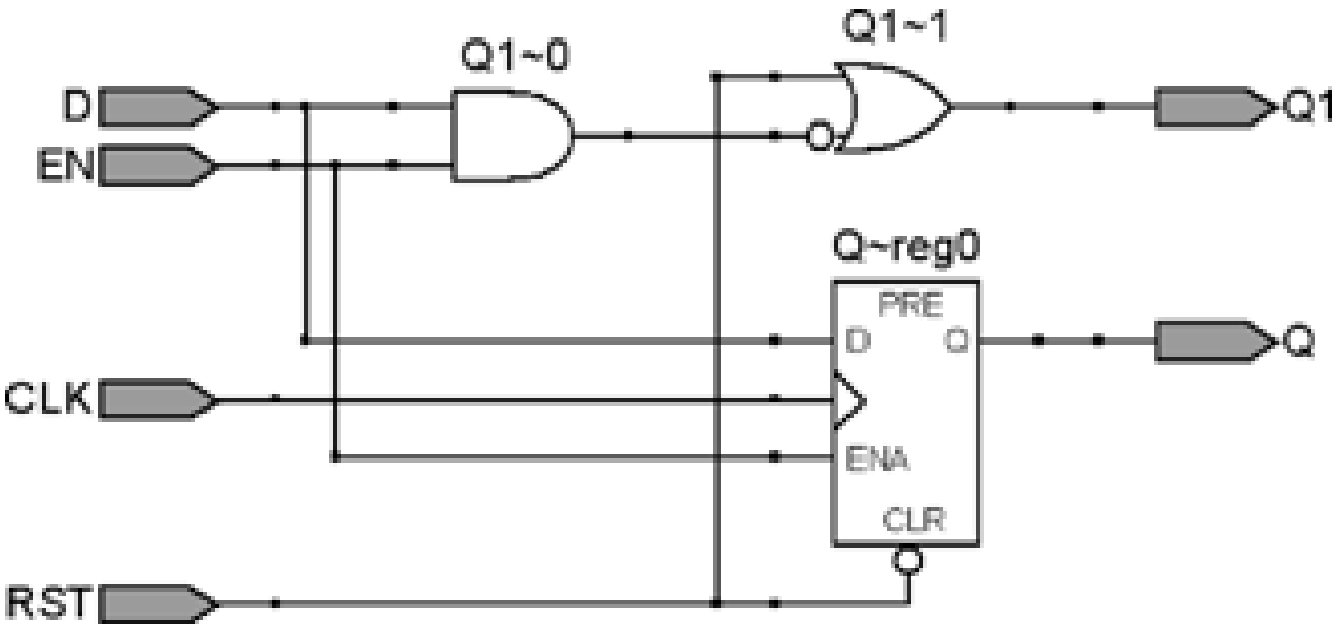
简答（RISC-V相关）



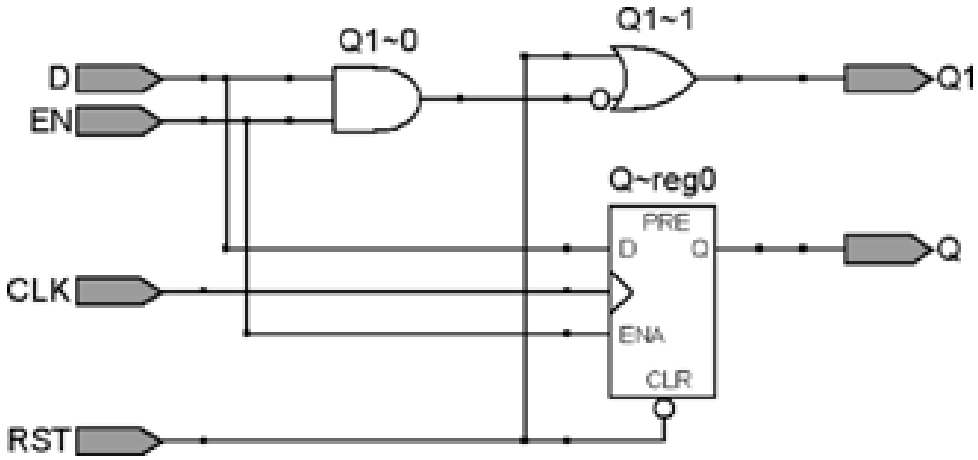
- RV32I指令集有那些指令类型？
- 答：R、I、U、B、S、J

- RISC-V处理器的寄存器x0有什么特殊性
- 答：
- 等于0

根据RTL图写Verilog程序

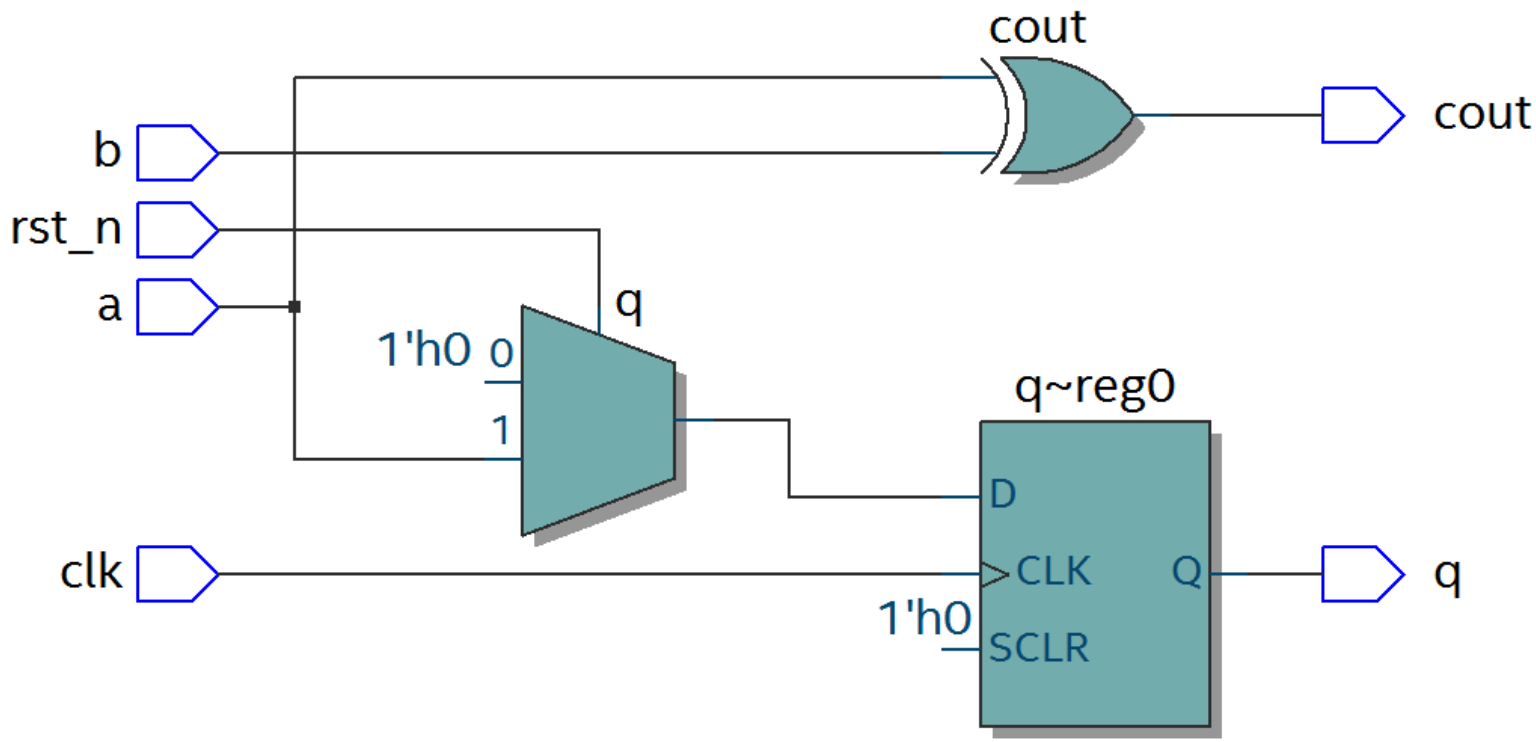


根据RTL图写Verilog程序

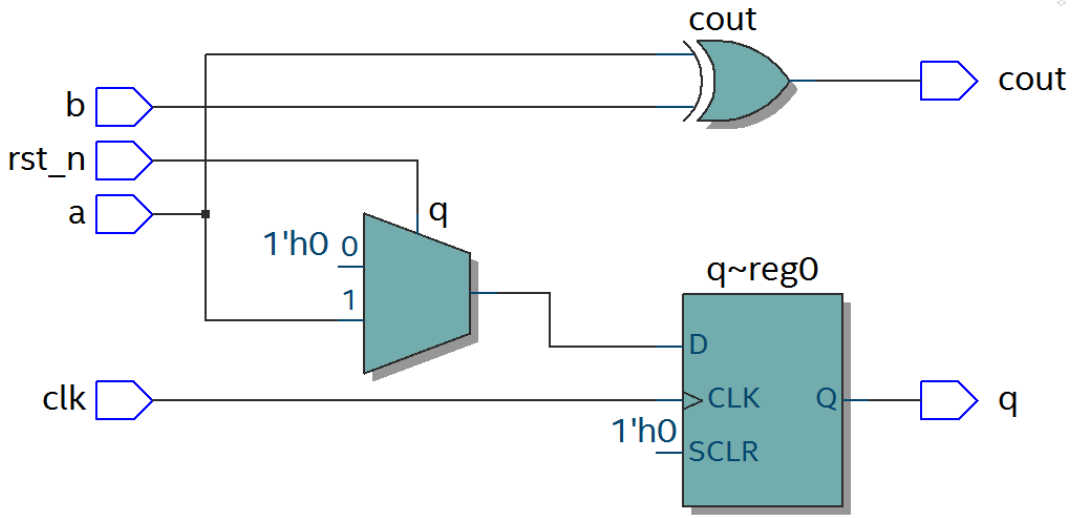



```
module test(CLK,RST,EN,D,Q,Q1);
    input CLK,RST,EN,D;
    output Q,Q1;

    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
```

根据RTL图写Verilog程序

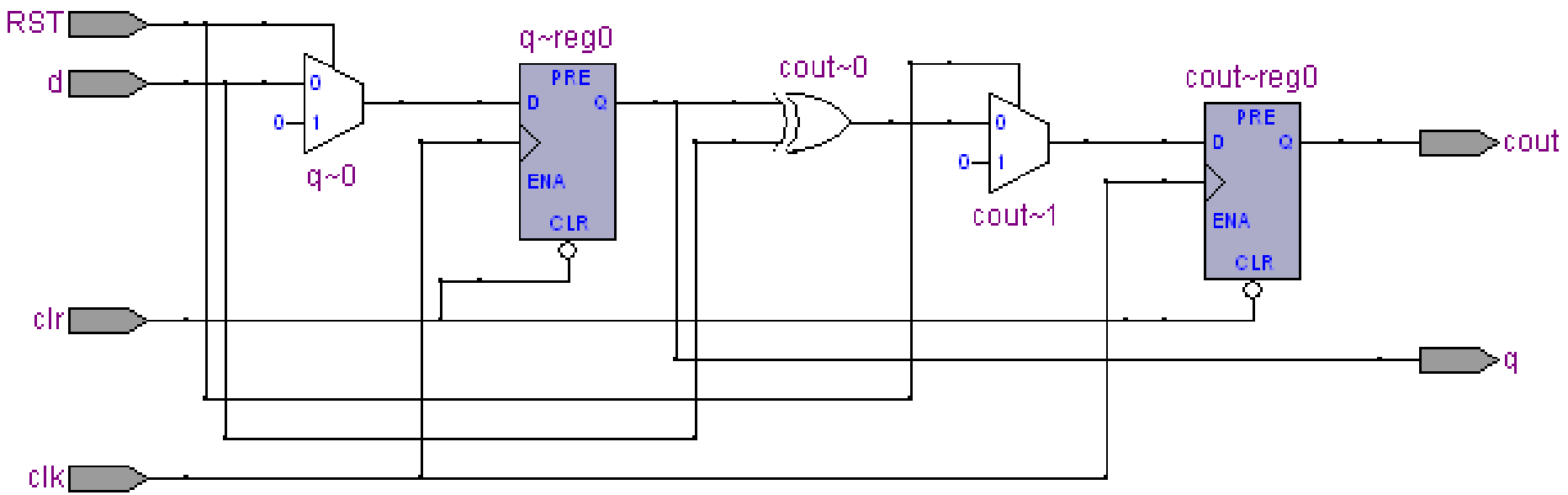


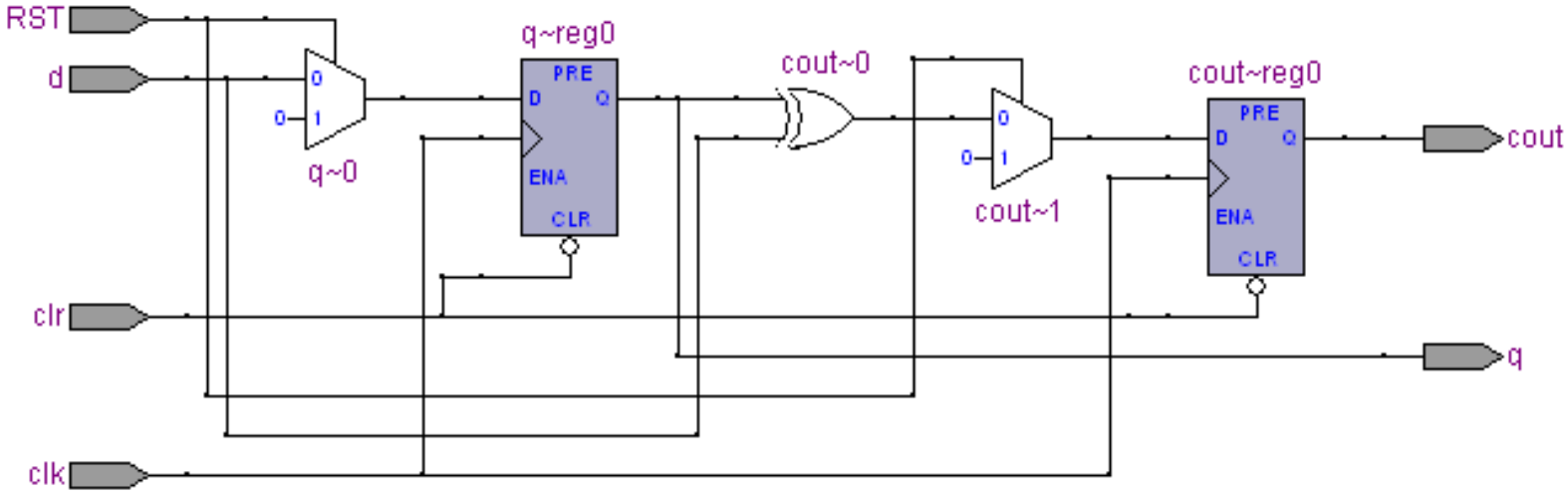
根据RTL图写Verilog程序




```
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
```

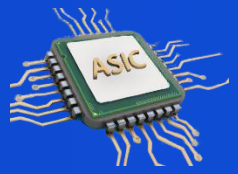
看RTL图写Verilog程序



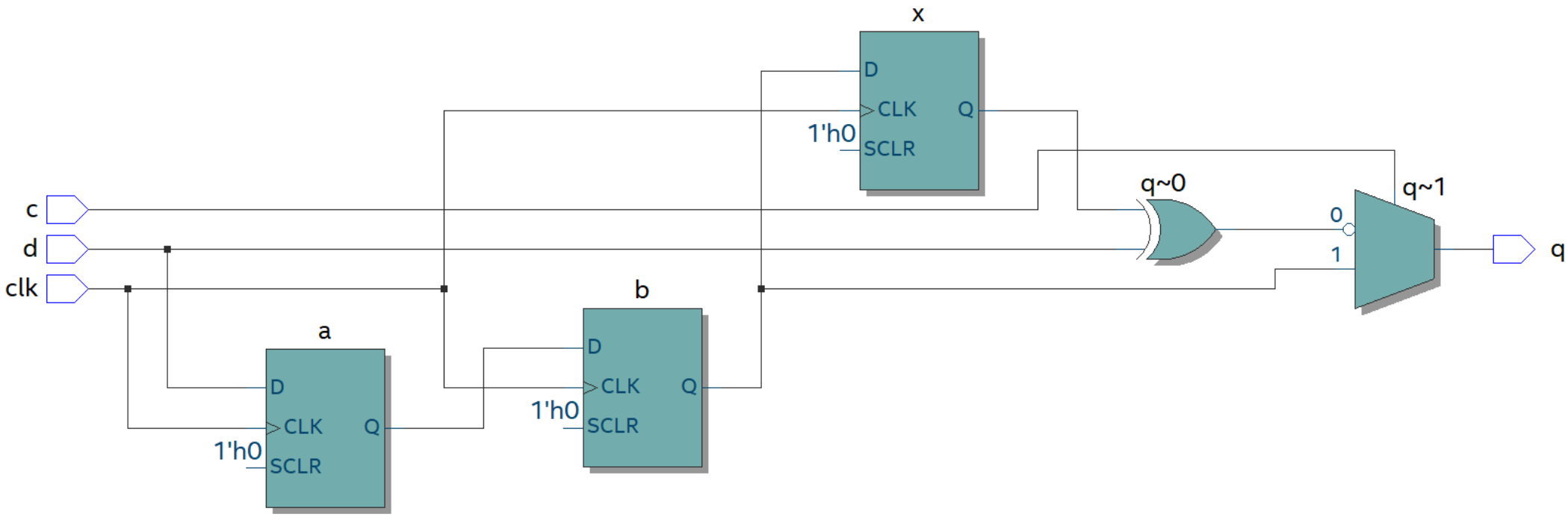



```
module abc(q,cout,d,RST,clk,clr); // 1分，包括module name
input d,clk,clr,RST;              // 1分，包括所有端口信号
output q;
output cout;
reg q,cout;             // 1分，无意义的reg不给分
always@(posedge clk or negedge clr) // 1分 加了RST无得分
begin
if (!clr) begin q<=0; cout<=0; end  // 1分
else if (RST)                 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

阅读下列Verilog程序，画出RTL图

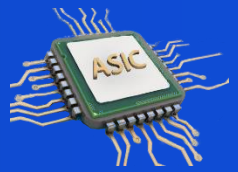



```
module test(clk,d,q,c);
    input clk,c,d;
    output q;
    reg a;
    reg b,x;
    reg q;
always @(posedge clk)
begin
    a <= d;
    b <= a;
    x <= b;
end
always@(c,b,x,d)
begin
    if (c)       q<=b;
    else q<=x~^d;
end
endmodule
```

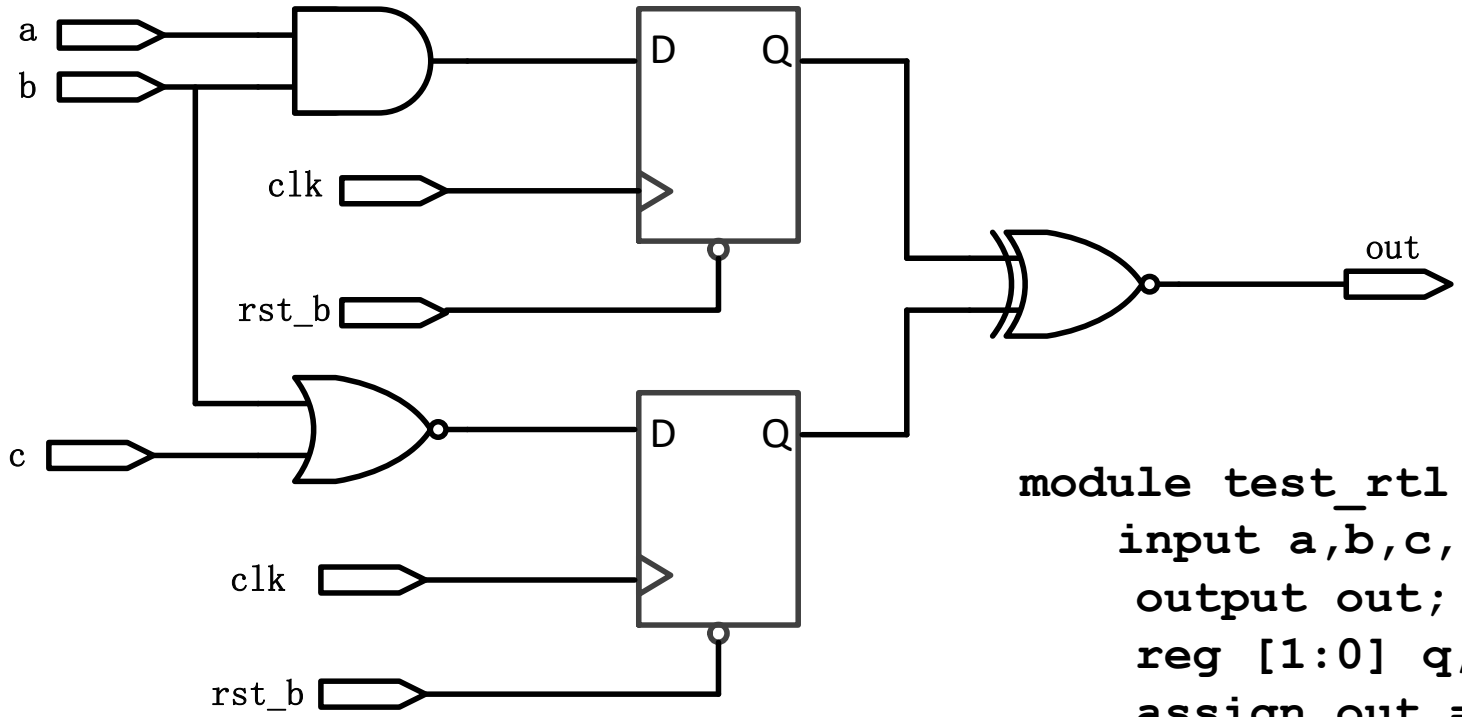



```
module test(clk,d,q,c);
     input clk,c,d;
     output q;
     reg a;
     reg b,x;
     reg q;
always @(posedge clk)
begin
     a <= d;
     b <= a;
     x <= b;
end
always@(c,b,x,d)
begin
     if (c) q<=b;
     else q<=x~^d;
end
endmodule
```

阅读下列Verilog程序，画出RTL图

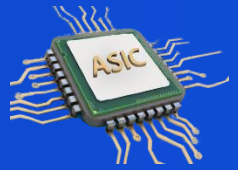



```
module test_rtl(a,b,c,clk,rst_b,out);
   input a,b,c,clk,rst_b;
    output out;
    reg [1:0] q;
    assign out = q[1] ~^ q[0];
    always@(posedge clk, negedge rst_b)
       if(!rst_b)
          q <= 0;
      else
          q <= {a&b,~(b|c)};
endmodule
```




```
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
```

练习



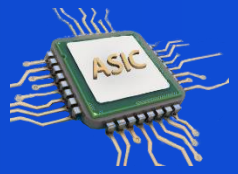
设计一个带异步复位、同步使能、同步清零、同步置位、同步装载的N=16位二进制计数器

课堂练习

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
```

4位Johnson计数器



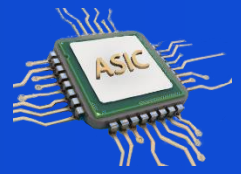
设计一个具备异步清零的4位Johnson计数器。其计数行为是：

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 ...

4位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；

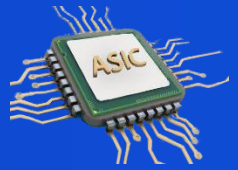
若当前计数器最高位为1，则执行最低位补0的左移操作。

如 000->001-->011-->111-->110-->100-->000-->001

...

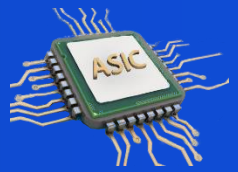
```
module CNT_JS(clk, rstn, en, cnt_js);
    input clk,rstn,en;
    output reg [3:0] cnt_js;

    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
```

32位乘加器



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

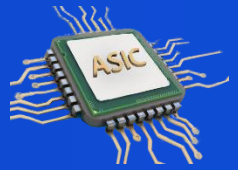
32位乘加器



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

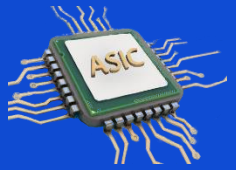
```
module muladd(clk,rstn,A,B,R);                          //模块定义
       input clk,rstn;
       input [31:0] A,B;
       output [31:0] R;                                  //端口定义
       reg [31:0] R;                                //数据类型定义
       always @ (posedge clk or negedge rstn)     //过程及敏感信号
              if (!rstn) R<=0;
                     else R<=R+A*B;          //乘法，累加，一个时钟完成
endmodule
```

组合电路设计与时许电路设计



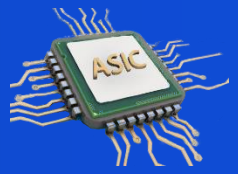
- 多路选择器/复用器
- 数据分配器/解复用器
- 译码器
- 编码器
- 算术逻辑单元
- BCD码计数器
- 移位寄存器：串入串出、并入串出、串入并出、双向移位
- 分频器

多路选择器/复用器



设计一个8选1多路选择器

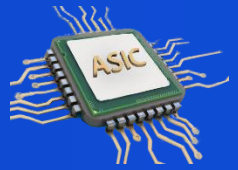
多路选择器/复用器



设计一个8选1多路选择器

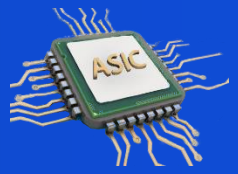
```
module mux81s
(
    input [2:0] s,
    input [7:0] a,
    output y
);
assign y = a[s];
endmodule
```

数据分配器/解复用器



设计一个**1-8**数据分配器

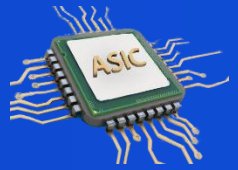
数据分配器/解复用器



设计一个1-8数据分配器

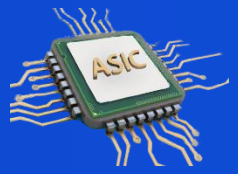
```
module demux8a
(
    input [2:0] s,
    input a,          // Input
    output [7:0] y
);

assign y = a << s;

endmodule
```

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

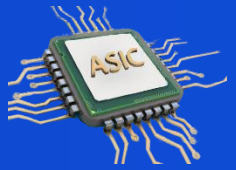
N=16的4-16译码器



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
```

8-3线编码器

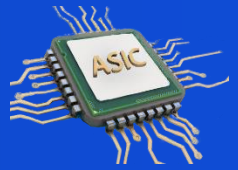


8-3线编码器

```
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
```

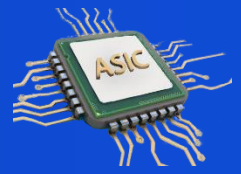
```
module enc38
(    input [7:0] i,
    output reg[2:0] y );

always @*
    if(i[0]) y=3'd0;
    else if(i[1]) y=3'd1;
    else if(i[2]) y=3'd2;
    else if(i[3]) y=3'd3;
    else if(i[4]) y=3'd4;
    else if(i[5]) y=3'd5;
    else if(i[6]) y=3'd6;
    else if(i[7]) y=3'd7;
    else y=3'b0;
endmodule
```

算术逻辑单元设计



设计一个8位的ALU，支持加、减、与、或、非运算。

算术逻辑单元ALU设计




```
`define add 3'd0
`define minus 3'd1
`define band 3'd2
`define bor 3'd3
`define bnot 3'd4
module ALU8 (a,b,opcode,out);
    input [7:0] a,b;
    input [2:0] opcode;
    output reg [7:0] out;
    always@(opcode,a,b) begin
        case (opcode)
        `add: out = a+b;
        `minus: out = a-b;
        `band: out = a&b;
        `bor: out = a|b;
        `bnot: out = ~a;
        default: out = 8'hx;//未收到指令输出任意态
        endcase
    end
endmodule
```

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
```

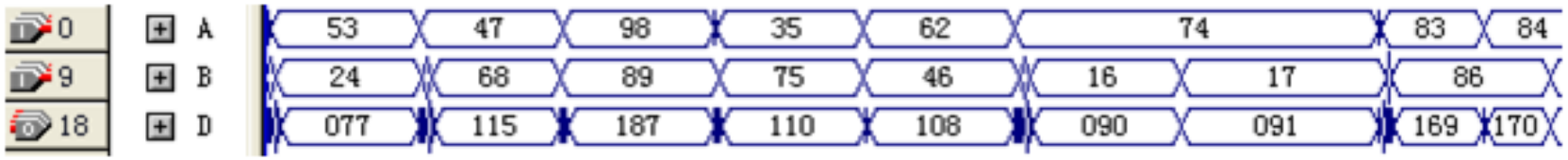


图 3-12 例 3-14 的仿真波形

BCD码加法器设计

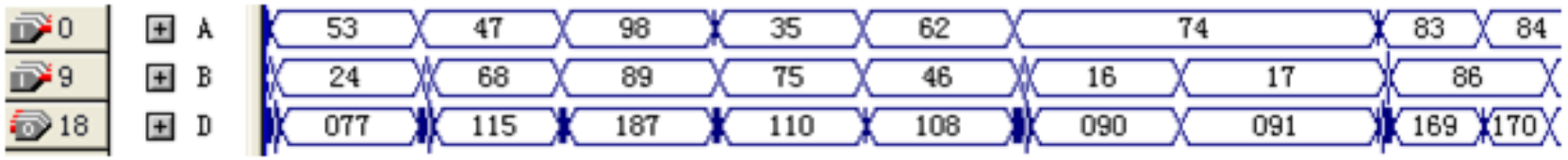


图 3-12 例 3-14 的仿真波形

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

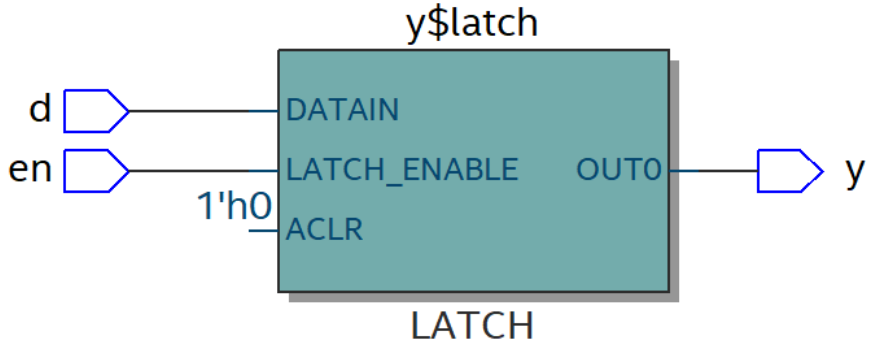
reg 与 wire

- reg
  - 既描述 连线 也可以描述 寄存器

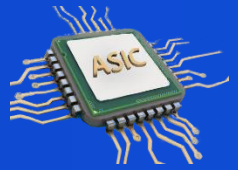
- reg y;
- always @*
  y = a; // 连线

- wire
  - 既描述 连线 也可以描述 寄存器

- wire y;
- assign y=en?d:y;
- //锁存器

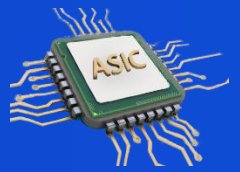


设计4位BCD十进制计数器



带异步复位、同步装载、同步使能信号

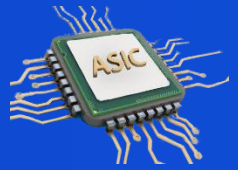
设计4位BCD十进制计数器



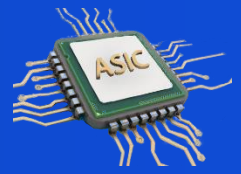
带异步复位、同步装载、同步使能信号

```
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
```

设计一个模为60的BCD码加法计数器

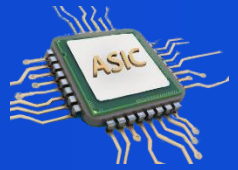


设计一个模为60的BCD码加法计数器

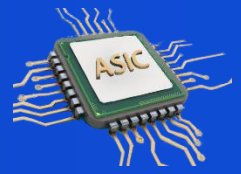



```
module count60 (qout, cout, data, load, cin, reset, clk);
    output [7:0] qout;
    output cout;
    input [7:0] data;
    input load, cin, clk, reset;
    reg [7:0] qout;
    always@(posedge clk) begin     //clk上升沿计数
        if (reset) qout<= 0;
        else if (load) qout <= data;   //同步复位
        else if (cin) begin            //同步置数
            if (qout[3:0]==9) begin //低位是否为9，是则回0，并判断高位是否为5
                qout[3:0] <=0;
                if (qout[7:4]==5)
                    qout [7:4] <=0;
                else
                qout [7:0]<= qout [7:0]+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1
            end
        end
    assign cout = ((qout==8'h59) & cin)?1:0; //产生进位输出信号
endmodule
```

4位串入串出移位寄存器

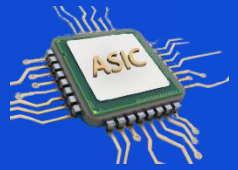


4位串入串出移位寄存器

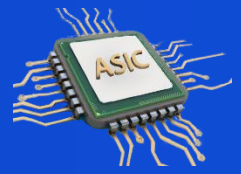



```
module siso41(clk,din,dout);
    input clk,din;
    output reg dout;
    reg [3:0] q;
    always@(posedge clk) begin
        q[0] <= din;
        q[3:1] <= q[2:0];
        dout <= q[3];
        end
endmodule
```

4位并入串出移位寄存器

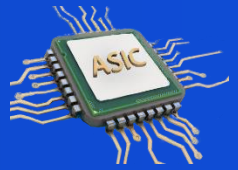


4位并入串出移位寄存器



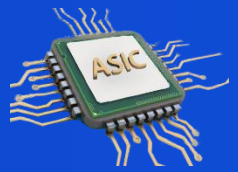

```
module piso41(clk,clr,din,dout);
    input clk,clr;
    input [3:0] din;
    output reg dout;
    reg [1:0] cnt;
    reg [3:0] q;
    always@(posedge clk) begin
        cnt <= cnt+1;
        if (clr)
            q<= 4'b0000;
        else begin
            if (cnt>0)
                q [3:1] <= q [2:0];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

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

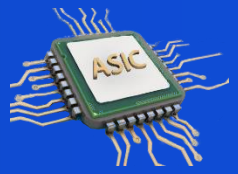
16位串入并出移位寄存器



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
```

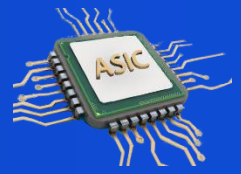
4位双向移位寄存器



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

4位双向移位寄存器

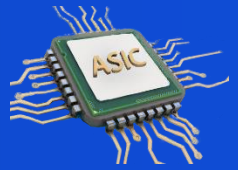


设计一个4位的双向移位寄存器，可根据以下表格实现特定的功能控制。

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

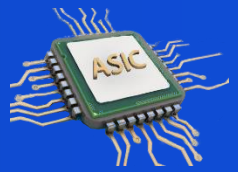
```
module shifter_194 (
    input rd_n ,
    input clk,
    input [0:3] d ,
    input dir ,
    input dil ,
    input [1:0] s ,
    output reg [0:3] q
);

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
```

分频数可控的分频器



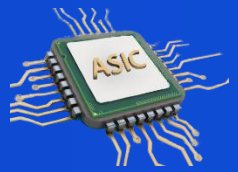
设计一个分频数可控的分频器，可以通过输入信号控制分频的系数N，将外部输入时钟信号分别进行2、4、6、8分频

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
```

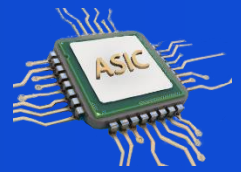


已知状态机的状态转移如下表，请写出Moore类型状态机

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

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
```

VGA简单图像显示控制模块设计

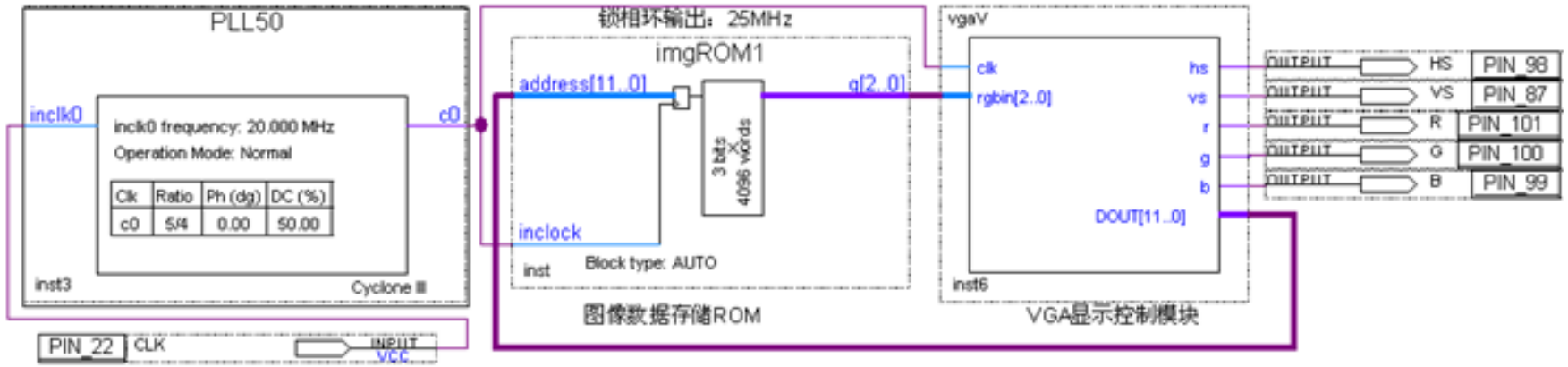
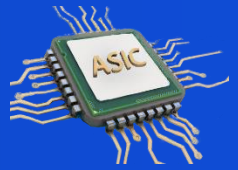


图 8-23 VGA 图像显示控制模块原理图

【例 8-33】

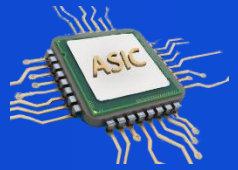
```
module vgaV (clk, hs, vs, r, g, b, rgbin, DOUT);
   input clk ;     //工作时钟 25MHz
   output hs,vs;   //场同步，行同步信号
   output r,g,b ;  // 红，绿，蓝信号，
   input[2:0] rgbin; //像素数据
   output[11:0] DOUT; //图像数据 ROM 的地址信号
   reg[9:0] hcnt, vcnt;      reg r,g,b;      reg hs,vs;
   assign DOUT = {vcnt[5:0], hcnt[5:0]} ;
   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
```

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

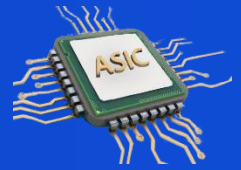
BCD计数器



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

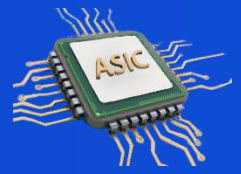
```
module bcdcnt(clk,rstn,load,en,data,q,c5);                    //1分，包括endmodule
        input clk,rstn,load,en;
        input [19:0] data;
        output [19:0] q;                                      //1分，正确定义20位的输入输出端口
        output c5;
        reg [3:0] q4,q3,q2,q1,q0;
        wire c0,c1,c2,c3,c4; assign q={q4,q3,q2,q1,q0};
        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&en)
                        if(q1<9) q1<=q1+1;
                        else q1<=0;
        assign c1=({q1,q0}==2'h99);
```

BCD计数器




```
always @(posedge clk, negedge rstn)
        if(!rstn) q2<=0;
        else if (load) q2<=data[11:8];
        else if (c1&en)
                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&en)
                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&en)
                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
```

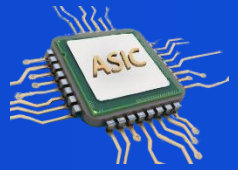
BCD计数器的TestBench




```
module bcdcnt_tb;
reg clk,rstn,load,en;
reg [19:0] data;
wire [19:0] q;  //1分，正确定义20位的输入输出端口
wire c5;
bcdcnt bcdcnt_inst
(           .clk(clk) ,// input  clk_sig
            .rstn(rstn) ,          // input  rstn_sig
            .load(load) ,          // input  load_sig
            .en(en) , // input  en_sig
            .data(data) ,          // input [19:0] data_sig
            .q(q) ,     // output [19:0] q_sig
            .c5(c5)    // output  c5_sig
);
initial
begin
            clk = 1'b0;
            forever #5 clk = ~clk;
end
```

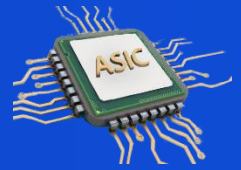
```
initial
begin
            rstn = 1'b0;
            load = 1'b0;
            en = 1'b0;
            data = 20'h12345;;
            #15 rstn = 1'b1;
            #10 data = 20'h23456;
            #10 load = 1'b1;
            #10         en = 1'b1;

            #1000000 en = 1'b0;
            #10 data = 20'h22222;
            #10 load = 1'b1;
            #10         en = 1'b1;

            #10000000 $stop;   // $finish
end      endmodule

endmodule
```

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 + 秒闪）

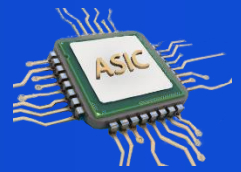
时钟




```
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;
```

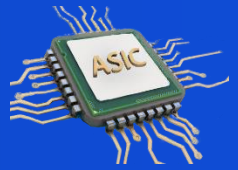
```
always @(posedge clk,posedge reset)
                  if(reset) ten_msec <= 0;
                  else if(load)
                  begin
                                    if(addr==0) ten_msec[3:0] <= d;
                                    if(addr==1) ten_msec[7:4] <= d;
                  end
                  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;
```

时钟



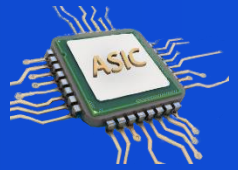

```
always @(posedge clk,posedge reset)
                if(reset) minu <= 0;
                else if(load)
                begin
                                if(addr==4) minu[3:0] <= d;
                                if(addr==5) minu[7:4] <= d;
                end
                else if(en&p_minu)
                                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
```

CPU相关



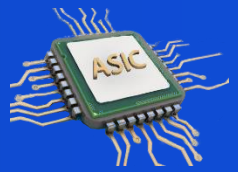
- 寄存器组
- PC
- ALU
- RV32I指令类型

ALU设计



设计一个32位ALU，支持加、减、与、或运算。
其端口信号为：input [31:0] A,B ,input [1:0] ALUOP, output [31:0] Y。

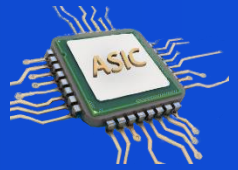
ALU设计



设计一个32位ALU，支持加、减、与、或运算。
其端口信号为：input [31:0] A,B ,input [1:0] ALUOP, output [31:0] Y。

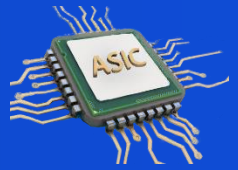
```
module alu
#(parameter ADD=0,SUB=1,AND=2,OR=3)
(    input [1:0] aluop,
     input [31:0] alua,alub,
     output reg[31:0] alur);
 always @*
     case(aluop)
          ADD : alur = alua + alub;
          SUB : alur = alua - alub;
          AND : alur = alua & alub;
          OR : alur = alua | alub;
default : alur = alua + alub;
     endcase
endmodule
```

寄存器组



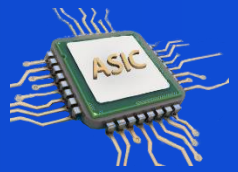
参考EDA书上CPU那章代码

PC



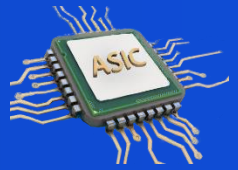
看实验9

设计4路PWM发生器



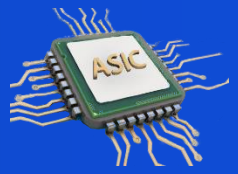
- 设计一个分频器，输入50MHz，输出1MHz（预分频器）
  - 输入：clk，reset_n
  - 输出 pclk
- 设计一个可控计数器，最大计数值为999999999，最小计数值为1，当d输入大于最大值时自动限制到最大
  - 输入clk，d（计数值控制），en，load（装d值），reset_n
  - 输出q（输出位宽为32位）
- 使用上述两个模块，构建4路PWM发生
  - 输入除了必要信号外，还有p0,p1,p2,p3（位宽为32位）控制占空比
  - 输入s0,s1,s2,s3（32位）为各路高电平起始位置
  - 输出x0,x1,x2,x3为PWM输出信号

相关概念



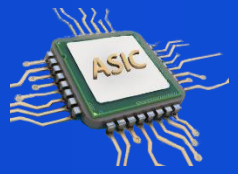
- PWM　　脉宽调制
  - Pulse Width Modulation
- 时钟域（Clock domain）
  - 电路中由同一个时钟信号控制的区域。
  - 整个4路PWM发生器模块处于同一个时钟域，即模块内所有寄存器受到同一个时钟驱动

预分频器



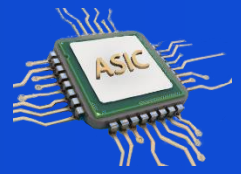

```
module fdiv
#(
    parameter N = 50,
    parameter NW = 6
)
(
    input clk,reset_n,
    output reg pclk
);
reg [NW-1:0] cnt;
always @(posedge clk,negedge
reset_n)
    if(!reset_n)
        cnt <= 0;
    else if(cnt < N-1)
        cnt <= cnt + 1'b1;
    else
        cnt <= 0;
always @(posedge clk,negedge
reset_n)
    if(!reset_n) pclk <= 1'b0;
    else if(cnt == 1) pclk <= 1'b1;
    else pclk <= 1'b0;
endmodule
```

可控计数器




```
module cnt
#( parameter NUM = 9_9999_9999)
(
    input clk, en,load, reset_n,
    input [31:0] d,
    output reg[31:0] q
);
reg [31:0] qmax;
always @(posedge clk,negedge reset_n)
    if(! reset_n) qmax <= 0; else if(load) qmax <= (d>NUM) ? NUM : d;
always @(posedge clk,negedge reset_n)
    if(! reset_n) q <= 0;
else if(en)
    if(q < qmax )
        q <= q + 1;
    else
        q <= 0;
endmodule
```

顶层（Top-Level）设计




```
module pwm4gen
(   input clk,reset_n,load,
    input [31:0] d,
    input [31:0] s0,s1,s2,s3,p0,p1,p2,p3,
    output reg x0,x1,x2,x3 );
wire pclk;
wire [31:0] q;
fdiv i_fdiv
(   .clk(clk),
    .reset_n(reset_n),
    .pclk(pclk)
);
cnt i_cnt
(   .clk(clk),
    .en(pclk),
    . reset_n(reset_n),
    .load(load),
    .d(d),
    .q(q)
);
```

```
always @(posedge clk,negedge reset_n)
    if(!reset_n)
    begin
        x0 <= 1'b0;
        x1 <= 1'b0;
        x2 <= 1'b0;
        x3 <= 1'b0;
    end
    else
    begin
        x0 <= 1'b0;
        x1 <= 1'b0;
        x2 <= 1'b0;
        x3 <= 1'b0;
        if((q>=s0)&&(q<p0)) x0 <= 1'b1;
        if((q>=s1)&&(q<p1)) x1 <= 1'b1;
        if((q>=s2)&&(q<p2)) x2 <= 1'b1;
        if((q>=s3)&&(q<p3)) x3 <= 1'b1;
    end
endmodule
```