离散数学(1): 数理逻辑

Discrete Mathematics (1): Mathematical Logic

数字逻辑的逻辑基础

赵永望

zhaoyw@buaa.edu.cn

北京航空航天大学 计算机学院

二进制数

- 在计算机系统中，数字表示为二进制数。二进制数每位数字是0或1，每位逢2进1。
- 通常有8位、16位、32位或64位二进制数。

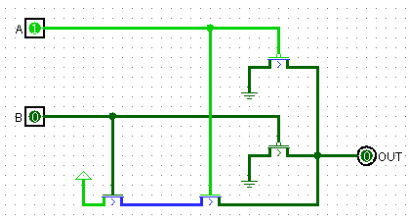
- $a_{31\sim0}$=11001100101011111100101110101010

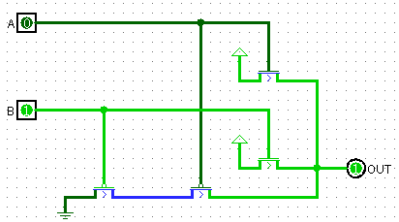
- 设$a_{31}.....a_1a_0$是32为二进制数，则二进制数与十进制数等值关系如下，
  - $(a_{31}.....a_1a_0)_2=a_{31}\times2^{31}+.....+a_1\times2+a_0$

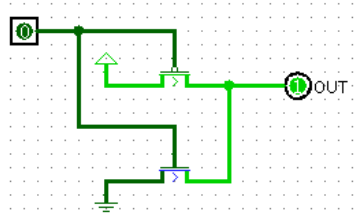
数字逻辑

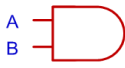
- 数字逻辑是关于数字（二进制数）的逻辑运算、关系判断及操作的逻辑表示方法，以及物理实现。
- 数字逻辑主要有组合逻辑和时序逻辑。
- 数字逻辑部件
  - 组合逻辑设计，包括编码器、译码器、比较器、数据选择器、数据分配器、奇偶校验器、算术逻辑单元、乘法器、数据扩展器等。
  - 时序逻辑设计，包括计数器、寄存器、移位器等

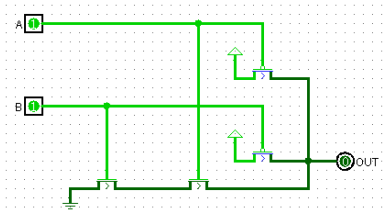
物理基础

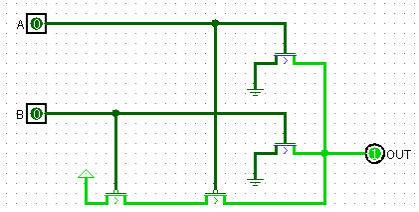


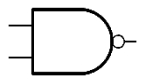


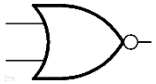


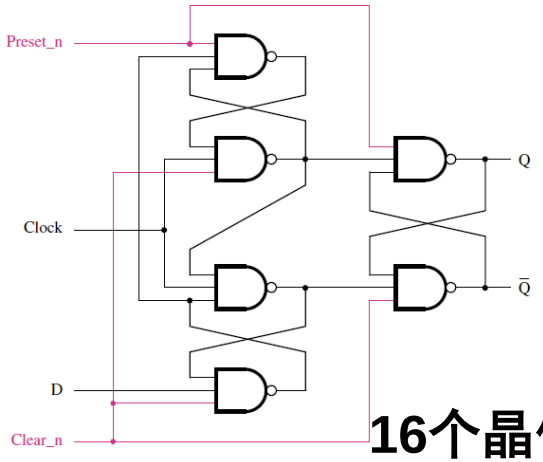






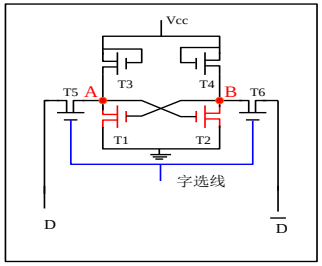




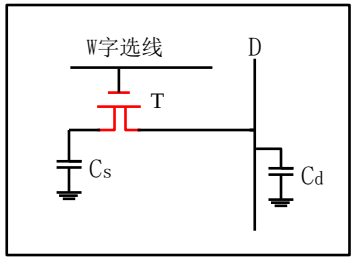


- 组合逻辑：与门、或门、非门
  与非门、或非门
- 存贮单元：SRAM、DRAM
- 时序逻辑：D触发器

6个晶体管



16个晶体管



功能的逻辑表达式一般方法

	x_0	…	x_k	…	x_{m-1}	y
0	0	0	0	0	0	…
…	…	…	…	…	…	…
k	1	0	0	1	0	1
…	…	…	…	…	…	…
2^m-1	1	…	1	1	1	…

- 若$x_{i,j}=1$，则$x'_{i,j}=x_k$，若$x_{i,j}=0$，则$x'_{i,j}=\neg x_{i,j}$
- 若$y_k=1$，k=0,…,n，则$y_k=x'_{m-1,k}\wedge\ldots\wedge x'_{0,k}$
- $y=y_0\vee\ldots\vee y_n$
- 功能可以用真值表表达
- 所有逻辑运算都可以表达为¬, ∧ ,∨,的运算。

命题逻辑是数字逻辑基础

- **概念与逻辑表达式**
  - 数字逻辑部件功能用真值表表达
  - 功能：输入与输出之间的关系。
  - 结构：实现功能的逻辑表达式。
- **逻辑结构的一般构建方法：**
  - 给定功能真值表
  - 命题逻辑方法求出（∧、∨、¬）逻辑范式
  - 构建逻辑部件（非门、与门、或门、寄存器）
  - **Verilog等软件实现**
    - 门级、数据流、行为级

Verilog设计方法问题

- **Verilog HDL是一种硬件描述语言**
  - 以文本形式来描述数字系统硬件的结构和行为的语言
  - 表示逻辑表达式，逻辑电路图
  - 表示数字逻辑系统所完成的逻辑功能
- **Verilog的基本语法与C语言相近的硬件描述语言**
  - 具备C语言的设计人员能够很快掌握Verilog硬件描述语言

描述方式

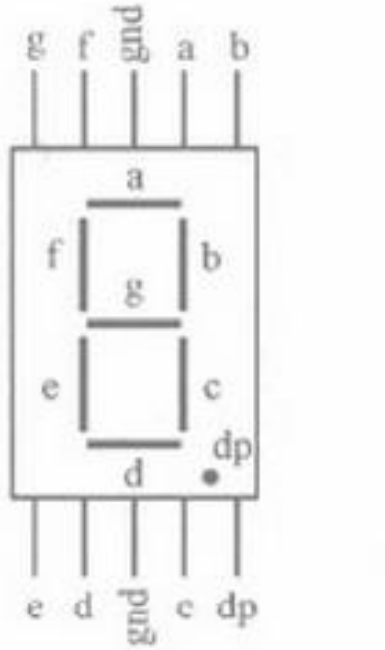
- VerilogHDL数字设计与综合（Samir Palnitkar)
- 行为级（算法）
  - 逻辑功能与算法的描述
- 数据流级
  - 通过说明数据的流程对模块描述
- 门级
  - 用逻辑门及其相互关系设计模块
- 开关级
  - 用开关、存储节点及其相互关系设计模块

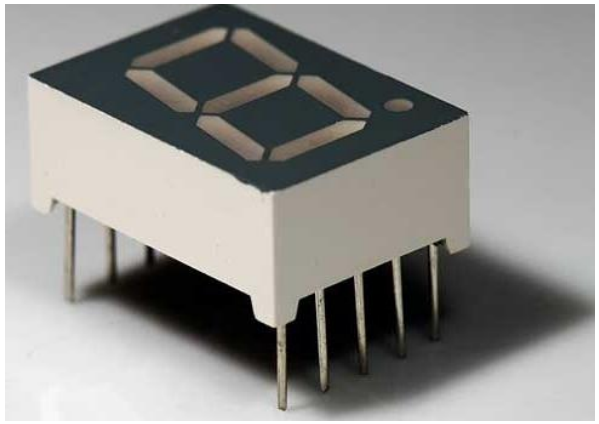
按位逻辑运算

- 设$a_{31\text{-}1\sim0}$, $b_{31\sim0}$是32位二进制数，二进制数逻辑运算是按位做非、与、或以及异或运算，运算记为~、&、|、^，即
- $\sim a_{31\sim0}=(\sim a_k)_{31\sim0}$
- $a_{31\sim0}\ \&\ b_{31\sim0}=(a_k\&b_k)_{31\sim0}$
- $a_{31\sim0}\ |\ b_{31\sim0}=(a_k|b_k)_{31\sim0}$
- $a_{31\sim0}\ \wedge\ b_{31\sim0}=(a_k\wedge b_k)_{31\sim0}$

编码器（七段数码管）

a	b	c	d	e	f	g	码
1	1	1	1	1	1	0	0
0	1	1	0	0	0	0	1
1	1	0	1	1	0	1	3
0	1	1	0	0	1	1	4
1	0	1	1	0	1	1	5
1	0	1	1	1	1	1	6
1	1	1	0	0	0	0	7
1	1	1	1	1	1	1	8
1	1	1	1	0	1	1	9
1	1	1	0	1	1	1	A
0	0	1	1	1	1	1	B
1	0	0	1	1	1	0	C
0	1	1	1	1	0	1	D
1	0	0	1	1	1	1	E
1	0	0	0	1	1	1	F





3线—8线译码器功能表（因变量关系）

- 译码器的输入是一个二进制数X，用$X_{n-1},\cdots,X_1,X_0$表示，输出是二进制数2^n-1，用$Y_{2^{**}n-1},\cdots,Y_1,Y_0$表示。

输入			输出							
X_2	X_1	X_0	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

- 一般性方法求逻辑表达式
  - $Y_0 = \neg X_2 \wedge \neg X_1 \wedge \neg X_0$
  - $Y_1 = \neg X_2 \wedge \neg X_1 \wedge X_0$
  - $Y_2 = \neg X_2 \wedge X_1 \wedge \neg X_0$
  - $Y_3 = \neg X_2 \wedge X_1 \wedge X_0$
  - $Y_4 = X_2 \wedge \neg X_1 \wedge \neg X_0$
  - $Y_5 = X_2 \wedge \neg X_1 \wedge X_0$
  - $Y_6 = X_2 \wedge X_1 \wedge \neg X_0$
  - $Y_7 = X_2 \wedge X_1 \wedge X_0$
- 32位译码器
- Verilog程序

多路选择器（5选1）

- 多路选择器是一种多路数据输入并且一路数据输出的逻辑部件

输入								输出
C_2	C_1	C_0	a_k	b_k	c_k	d_k	e_k	y_k
0	0	0	D_{0k}	×	×	×	×	D_{0k}
0	0	1	×	D_{1k}	×	×	×	D_{1k}
0	1	0	×	×	D_{2k}	×	×	D_{2k}
0	1	1	×	×	×	D_{3k}	×	D_{3k}
1	0	0	0	×	×	×	D_{4k}	D_{4k}

- 一般性方法求逻辑表达式

$z_0 = \neg C_2 \wedge \neg C_1 \wedge \neg C_0$
$z_1 = \neg C_2 \wedge \neg C_1 \wedge C_0$
$z_2 = \neg C_1 \wedge C_1 \wedge \neg C_0$
$z_3 = \neg C_1 \wedge C_1 \wedge C_0$
$z_4 = C_1 \wedge \neg C_1 \wedge \neg C_0$

- 多路选择器
- Verilog程序

$$Y_k = z_4 \wedge e_k \vee z_3 \wedge d_k \vee z_2 \wedge c_k \vee z_1 \wedge b_k \vee z_0 \wedge a_k$$

二进制加法运算

- $S_0= A_0+B_0$，被加数位A_0，加数位B_0以及进位位C_{-1}，输出有和数位S_0和进位位C_0。A_0，B_0和C_{-1}按二进制数相加，产生的二进制数的位2^0为S_0，位2^1为C_0。

输入			输出	
A_0	B_0	C_{-1}	S_0	C_0
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$$S_0=\sim A_0\&\sim B_0\&C_{-1}|\sim A_0\&B_0\&\sim C_{-1}|A_0\&\sim B_0\&\sim C_{-1}|A_0\&B_0\&C_{-1}$$
$$C_0 =\sim A_0\&B_0\&C_{-1}|A_0\&\sim B_0\&C_{-1}|A_0\&B_0\&\sim C_{-1}|A_0\&B_0\&C_{-1}$$

二进制加法运算

- $S_0=\sim A_0\&\sim B_0\&C_{-1}|\sim A_0\&B_0\&\sim C_{-1}|A_0\&\sim B_0\&\sim C_{-1}|A_0\&B_0\&C_{-1}$
- $C_0=\sim A_0\&B_0\&C_{-1}|A_0\&\sim B_0\&C_{-1}|A_0\&B_0\&\sim C_{-1}|A_0\&B_0\&C_{-1}$

  $(C_{-1}=0)$
- $S_1=\sim A_1\&\sim B_1\&C_0|\sim A_1\&B_1\&\sim C_0|A_1\&\sim B_1\&\sim C_0|A_1\&B_1\&C_0$
- $C_1=\sim A_1\&B_1\&C_0|A_1\&\sim B_1\&C_0|A_1\&B_1\&\sim C_0|A_1\&B_1\&C_0$
- $S_2=\sim A_2\&\sim B_2\&C_1|\sim A_2\&B_2\&\sim C_1|A_2\&\sim B_2\&\sim C_1|A_2\&B_2\&C_1$
- $C_2=\sim A_2\&B_2\&C_1|A_2\&\sim B_2\&C_1|A_2\&B_2\&\sim C_1|A_2\&B_2\&C_1$
- $S_3=\sim A_3\&\sim B_3\&C_2|\sim A_3\&B_3\&\sim C_2|A_3\&\sim B_3\&\sim C_2|A_3\&B_3\&C_2$
- $C_3=\sim A_3\&B_3\&C_2|A_3\&\sim B_3\&C_2|A_3\&B_3\&\sim C_2|A_3\&B_3\&C_2$

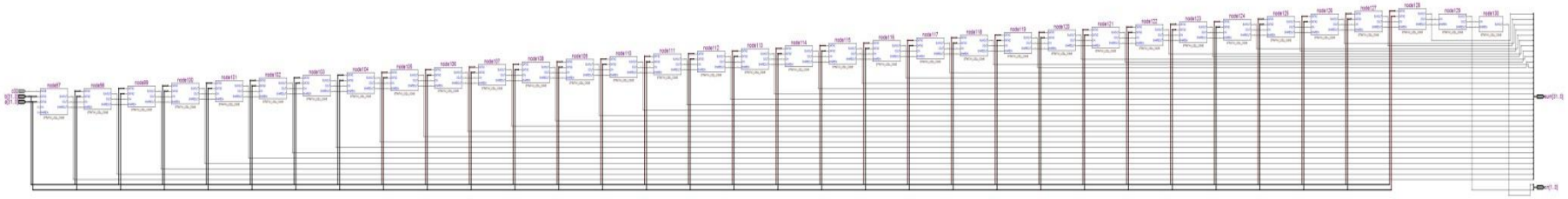
- 8位加法
- 32位加法
- ALU

addr行为级实现

- 行为级描述简单
- 类似C程序设计语言
- 逻辑部件内部结构是黑盒，无法了解其逻辑结构
- 难于构建程序结构与物理逻辑结构的对应关系

```
module adder_behavioural_32_module(a,b,c00,sum,cn);
input [31:0]a,b;
input c00;
output [31:0]sum;
output [1:0]cn;

    assign {cn,sum}=a+b+c00;
endmodule
```



addr门级实现

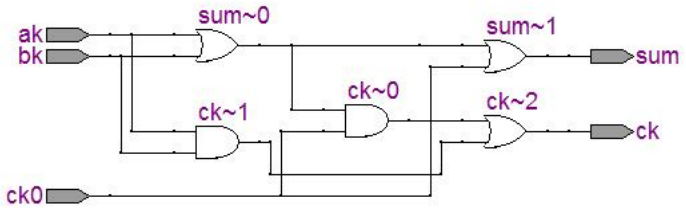
- 给出1位二进制加法真值表
- 给出1位二进制加法门级设计逻辑
- 给出32位加法器设计
- 程序结构与物理逻辑结构完全对应，易于理解与验证

输入			输出	
A'_0	B'_0	C'_{-1}	S'_0	C'_0
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

```
module adder_ripple_1_module(ak,bk,ck0,sum,ck);
input ak,bk;
input ck0;
output sum;
output ck;

    assign ck=(ak|bk)&ck0|ak&bk;
    assign sum=ak|bk|ck0;
endmodule
```

```
module adder_ripple_32_module(Ain,Bin,c00,sum,cn);
input [31:0]Ain,Bin;
input c00;
output [31:0]sum;
output [1:0]cn;
wire [31:0]carryOut;
    adder_ripple_1_module adder0(Ain[0],Bin[0],c00,sum[0],carryOut[0]);
    adder_ripple_1_module adder1(Ain[1],Bin[1],carryOut[0],sum[1],carryOut[1]);
    adder_ripple_1_module adder2(Ain[2],Bin[2],carryOut[1],sum[2],carryOut[2]);
    adder_ripple_1_module adder3(Ain[3],Bin[3],carryOut[2],sum[3],carryOut[3]);
    adder_ripple_1_module adder4(Ain[4],Bin[4],carryOut[3],sum[4],carryOut[4]);
    adder_ripple_1_module adder5(Ain[5],Bin[5],carryOut[4],sum[5],carryOut[5]);
    adder_ripple_1_module adder6(Ain[6],Bin[6],carryOut[5],sum[6],carryOut[6]);
    adder_ripple_1_module adder7(Ain[7],Bin[7],carryOut[6],sum[7],carryOut[7]);
    adder_ripple_1_module adder8(Ain[8],Bin[8],carryOut[7],sum[8],carryOut[8]);
    adder_ripple_1_module adder9(Ain[9],Bin[9],carryOut[8],sum[9],carryOut[9]);
    adder_ripple_1_module adder10(Ain[10],Bin[10],carryOut[9],sum[10],carryOut[10]);
    adder_ripple_1_module adder11(Ain[11],Bin[11],carryOut[10],sum[11],carryOut[11]);
    adder_ripple_1_module adder12(Ain[12],Bin[12],carryOut[11],sum[12],carryOut[12]);
    adder_ripple_1_module adder13(Ain[13],Bin[13],carryOut[12],sum[13],carryOut[13]);
    adder_ripple_1_module adder14(Ain[14],Bin[14],carryOut[13],sum[14],carryOut[14]);
    adder_ripple_1_module adder15(Ain[15],Bin[15],carryOut[14],sum[15],carryOut[15]);
    adder_ripple_1_module adder16(Ain[16],Bin[16],carryOut[15],sum[16],carryOut[16]);
    adder_ripple_1_module adder17(Ain[17],Bin[17],carryOut[16],sum[17],carryOut[17]);
    adder_ripple_1_module adder18(Ain[18],Bin[18],carryOut[17],sum[18],carryOut[18]);
    adder_ripple_1_module adder19(Ain[19],Bin[19],carryOut[18],sum[19],carryOut[19]);
    adder_ripple_1_module adder20(Ain[20],Bin[20],carryOut[19],sum[20],carryOut[20]);
    adder_ripple_1_module adder21(Ain[21],Bin[21],carryOut[20],sum[21],carryOut[21]);
    adder_ripple_1_module adder22(Ain[22],Bin[22],carryOut[21],sum[22],carryOut[22]);
    adder_ripple_1_module adder23(Ain[23],Bin[23],carryOut[22],sum[23],carryOut[23]);
    adder_ripple_1_module adder24(Ain[24],Bin[24],carryOut[23],sum[24],carryOut[24]);
    adder_ripple_1_module adder25(Ain[25],Bin[25],carryOut[24],sum[25],carryOut[25]);
    adder_ripple_1_module adder26(Ain[26],Bin[26],carryOut[25],sum[26],carryOut[26]);
    adder_ripple_1_module adder27(Ain[27],Bin[27],carryOut[26],sum[27],carryOut[27]);
    adder_ripple_1_module adder28(Ain[28],Bin[28],carryOut[27],sum[28],carryOut[28]);
    adder_ripple_1_module adder29(Ain[29],Bin[29],carryOut[28],sum[29],carryOut[29]);
    adder_ripple_1_module adder30(Ain[30],Bin[30],carryOut[29],sum[30],carryOut[30]);
    adder_ripple_1_module adder31(Ain[31],Bin[31],carryOut[30],sum[31],carryOut[31]);
    assign cn[1]=carryOut[31];
    assign cn[0]=carryOut[30];
endmodule
```



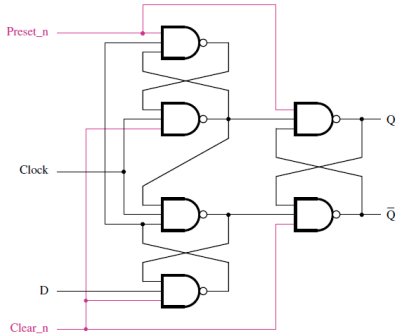
程序设计描述方式的选择

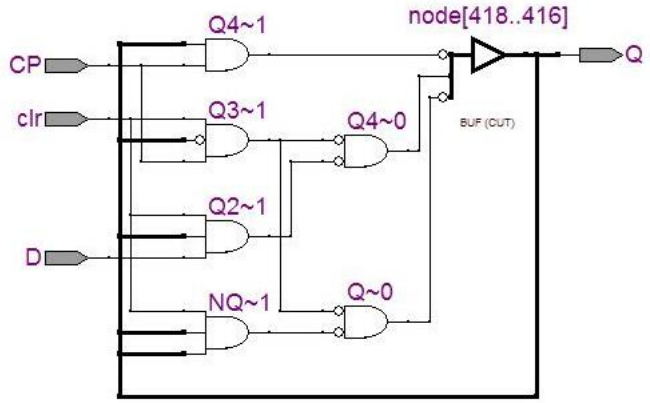
- 虽然行为级描述简单，但是，内部逻辑结构是黑盒，难于构建程序结构与物理逻辑结构的显示映射。
- 虽然门级描述略繁琐，但是，其结构简单，并且易于构建程序结构与物理逻辑结构的显示映射。
- 门级描述能够比较好地培养硬件程序设计的基本思维方法与能力。

寄存器设计

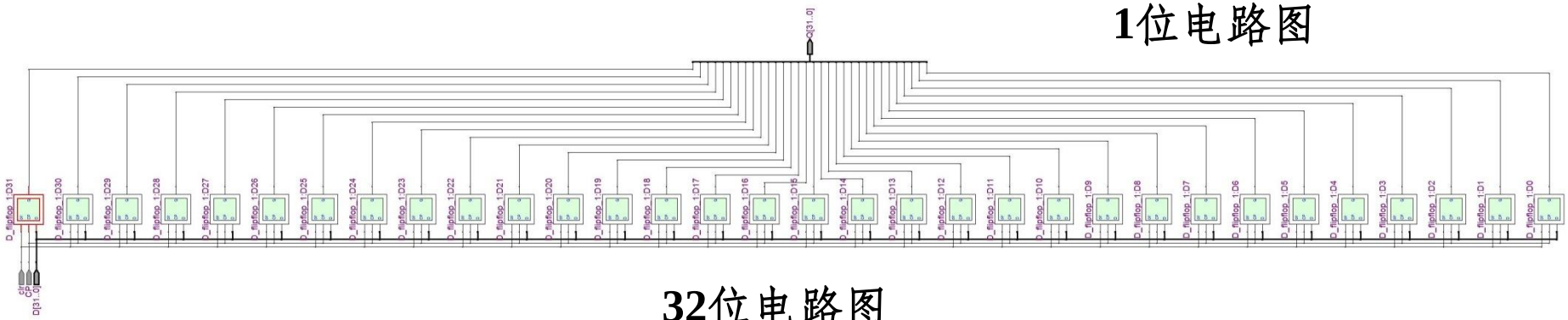
```
module D_flipflop_1(clr,CP,D,Q);
input clr,CP;
input D;
output Q;
wire NQ,Q4,Q3,Q2,Q1;
wire S;
      assign S=1'b1;
      assign NQ=~(Q&Q4&clr);
      assign Q=~(NQ&Q3&S);
      assign Q2=~(Q4&D&clr);
      assign Q4=~(Q2&Q3&CP);
      assign Q3=~(Q1&CP&clr);
      assign Q1=~(Q2&Q3&S);
endmodule
```

D触发器
原理图





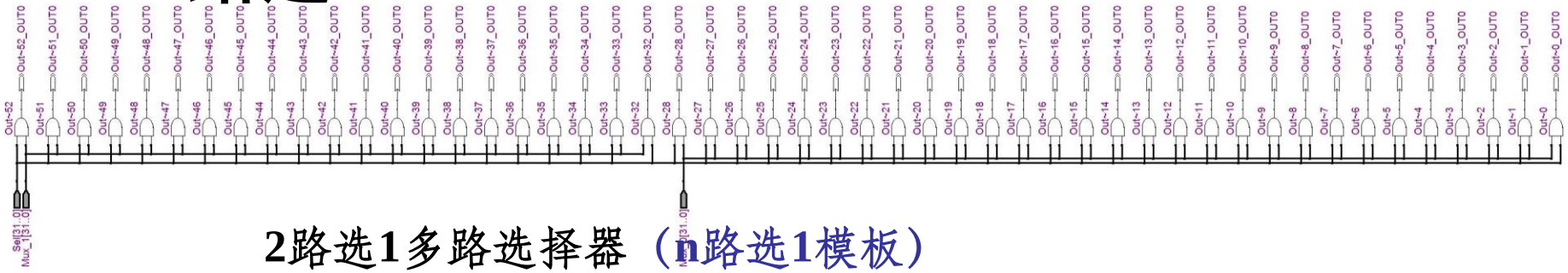
1位电路图



32位电路图

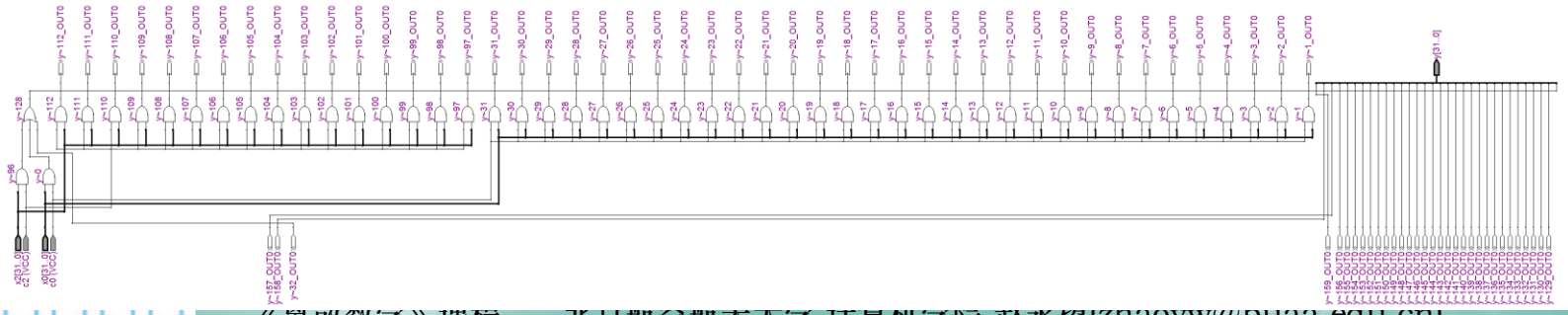
多路选择器设计

- 32路选1



2路选1多路选择器（n路选1模板）

```
module muxu3_module(x0,c0,x1,c1,x2,c2,y);
input [31:0]x0,x1,x2;
input c0,c1,c2;
output [31:0]y;
      assign y=(x0&{32{c0}})|(x1&{32{c1}})|(x2&{32{c2}});
endmodule
```



and（or）,xor

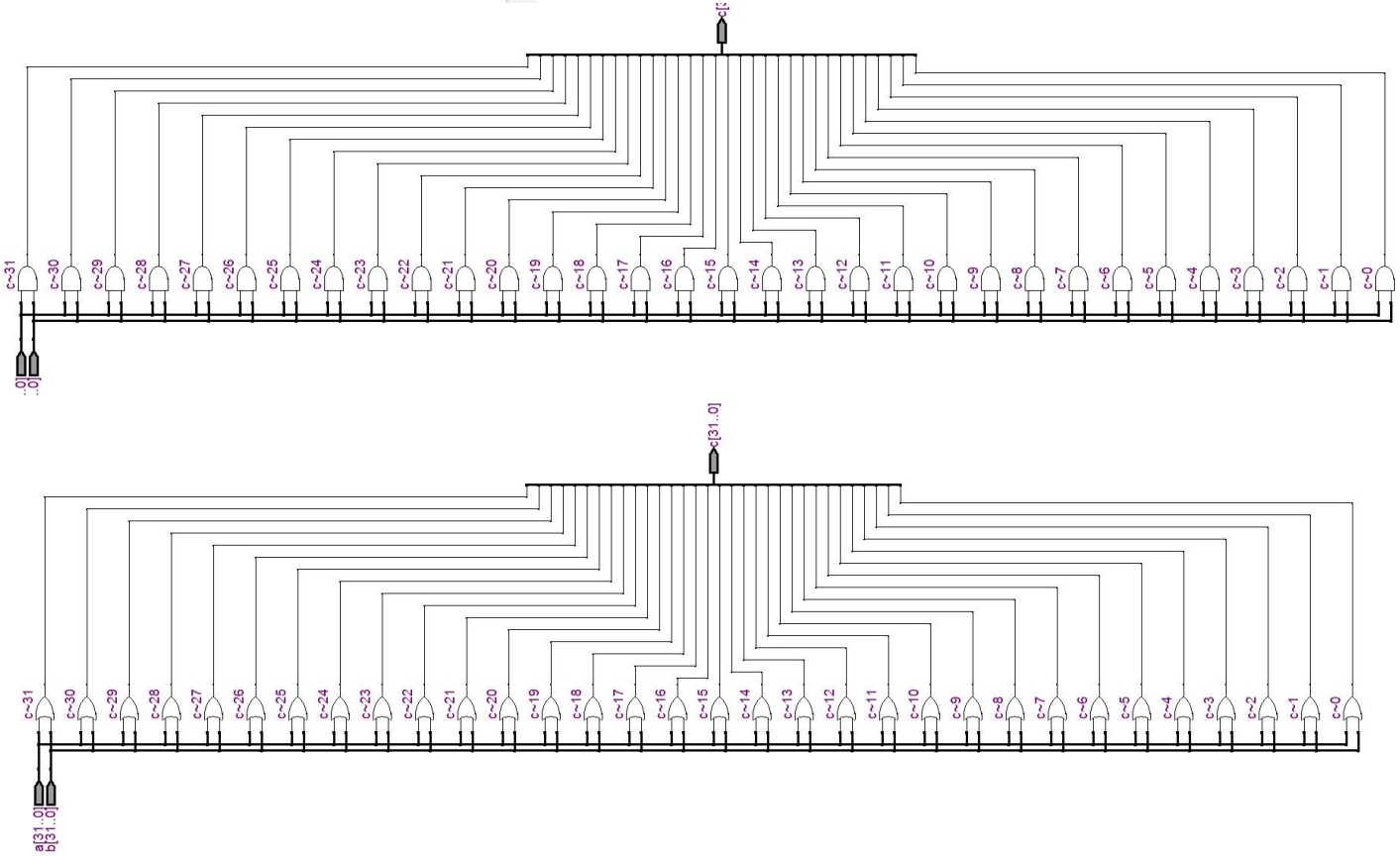
```
module and32_module(a,b,c);
 input [31:0]a,b;
 output [31:0]c;

     assign c=a&b;
endmodule
```

```
module xor32_module(a,b,c);
 input [31:0]a,b;
 output [31:0]c;

     assign c=~a&b|a&~b;
endmodule
```

- 按位二进制运算
- and
- or
- xor



本　节　完!

问题与解答?