第八章作业

钟赟　　2016K8009915009

1、请将下列无符号数据在不同的进制表达间进行转换。

(1)　二进制转换为十进制：101011_2、001101_2、01011010_2、0000111010000101_2。

二进制	101011_2	001101_2	01011010_2	0000111010000101_2
十进制	43	13	90	3717

(2)　十进制转换为二进制：42_{10}、79_{10}、811_{10}、374_{10}。

十进制	42_{10}	79_{10}	811_{10}	374_{10}
二进制	101010	1001111	1100101011	101110110

(3)　十六进制转换为十进制：$8AE_{16}$、$C18D_{16}$、$B379_{16}$、100_{16}。

十六进制	$8AE_{16}$	$C18D_{16}$	$B379_{16}$	100_{16}
十进制	2222	49549	45945	256

(4)　十进制转换为十六进制：81783_{10}、1922_{10}、345208_{10}、5756_{10}。

十进制	81783_{10}	1922_{10}	345208_{10}	5756_{10}
十六进制	13f77	782	54478	167c

2、请给出 32 位二进制数分别视作无符号数、原码、补码时所表示的数的范围。

无符号数：$[0，2^{32}-1]$

原码：　$[-(2^{31}-1)，2^{31}-1]$

补码：　$[-2^{32}，2^{31}-1]$

3、请将下列十进制数表示为 8 位原码和 8 位补码，或者表明该数据会溢出：45_{10}，-59_{10}，-128_{10}，119_{10}，127_{10}，128_{10}，0_{10}，-1_{10}。

十进制数	**8 位原码**	**8 位补码**
45	00101101	00101101
-59	10111011	11000101
-128	溢出	10000000
119	01110111	01110111
127	01111111	01111111

128	溢出	溢出
0	00000000 或 1000000	00000000
-1	10000001	11111111

4、**请将下列数据分别视作原码和补码，从 8 位扩展为 16 位：00101100_2、11010100_2、10000001_2、00010111_2。**

原始数据	视为原码扩展	视为补码扩展
00101100	00000000 00101100	00000000 00101100
11010100	10000000 01010100	11111111 11010100
10000001	10000000 00000001	11111111 10000001
00010111	00000000 00010111	00000000 00010111

5、请将下列浮点数在不同进制间进行转换。

（1）十进制转换为单精度数：0、116.25、-4.375。

十进制	0	116.25	-4.375
单精度	0x0	0x42e88000	0xc08c0000

（2）十进制转换为双精度数：-0、116.25、-2049.5。

十进制	-0	116.25	-2049.5
双精度	0x8000000000000000	0x405d100000000000	0xc0a0030000000000

（3）单精度数转换为十进制数：0xff800000、0x7fe00000。

单精度	0xff800000	0x7fe00000
十进制	-∞	NaN

（4）双精度数转换为十进制数：0x80080000000000000、0x7065020000000000。

双精度	0x8008000000000000	0x7065020000000000
十进制	-2^{-1023}	$5378*2^{743}$

6、请写出题 6 图所示晶体管级电路图的真值表，并给出对应的逻辑表达式。

晶体管级电路图对应的真值表为：

A	B	C	Y
0	0	0	**1**

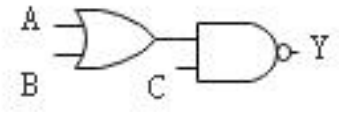
1	0	0	**1**
0	1	0	**1**
0	0	1	**1**
1	1	0	**1**
1	0	1	**0**
0	1	1	**0**
1	1	1	**0**

根据真值表可以推导逻辑表达式为：

Y = ~((A & C) | (B & C) | (A & B & C))

= ~A & ~B | ~C

7、请写出下图所示逻辑门电路图的真值表。



真值表为：

A	B	C	Y
0	0	0	**1**
1	0	0	**1**
0	1	0	**1**
0	0	1	**1**
1	1	0	**1**
1	0	1	**0**
0	1	1	**0**
1	1	1	**0**

8、请用尽可能少的二输入 NAND 门搭建出一个具有二输入 XOR 功能的电路。

设输入为 A 和 B，Y 为 A XOR B 的输出，则有

Y = ~A & B | A & ~B

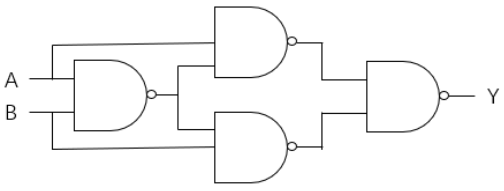
= ~(~(~A & B) & ~(A & ~B))

= ~(~(~A & B | ~B & B) & ~(A & ~B | ~A & A))

= ~(~((~A | ~B) & B) & ~(A & (~A | ~B)))

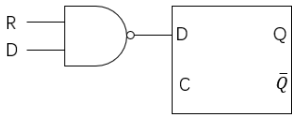
= ~(~(~(A & B) & B) & ~(A & ~(A & B)))

电路图如下：



9、请用 D 触发器和常见组合逻辑门电路搭建出一个具有同步复位功能为 0 功能的触发器的电路。

当 R 为 0 时，执行复位功能，电路图如下：



10、证明$[X+Y]_{补}=[X]_{补}+[Y]_{补}$。

证明：设 X、Y 分别有 1 位符号位，n-1 位数据位，则 X、Y 的数值分以下四种情况：

(1) X > 0, Y > 0

则 X + Y > 0，$[X]_{补} = 2^n + X \pmod{2^n} = X$，　$[Y]_{补} = 2^n + Y \pmod{2^n} = Y$,

$[X]_{补}+[Y]_{补} = X + Y$

$[X+Y]_{补} = 2^n + X + Y \pmod{2^n} = X + Y = [X]_{补}+[Y]_{补}$

(2) X > 0, Y < 0

$[X]_{补} = 2^n + X \pmod{2^n} = X$，　$[Y]_{补} = 2^n + Y$,

如果 X + Y > 0，$[X]_{补}+[Y]_{补} = X + Y + 2^n \pmod{2^n} = X + Y$

$[X+Y]_{补} = 2^n + (X + Y) = X + Y = [X]_{补}+[Y]_{补}$

如果 X + Y < 0，$[X]_{补}+[Y]_{补} = X + Y + 2^n$

$[X+Y]_{补} = 2^n + (X + Y) = [X]_{补}+[Y]_{补}$

(3) X < 0, Y > 0

与第(2)种情况同理。

(4) X < 0, Y < 0

则 X + Y < 0，$[X]_{补} = 2^n + X$，　$[Y]_{补} = 2^n + Y$,

$[X+Y]_{补} = 2^n + X + Y$

由于 X 和 Y 的绝对值都小于等于 2^{n-1}，故 X + Y 的绝对值小于等于 2^n，所以

$[X]_{补}+[Y]_{补} = X + Y + 2^n + 2^n \pmod{2^n} = X + Y + 2^n = [X+Y]_{补}$

综上，可证$[X+Y]_{补}=[X]_{补}+[Y]_{补}$。

11、证明$[X-Y]_{补}=[X]_{补}+[-Y]_{补}$。

证明：运用第 10 题中结论可得：

$[X-Y]_{补} = [X+(-Y)]_{补} = [X]_{补}+[-Y]_{补}$

12、假设每个“非门”“与非门”“或非门”的扇入不超过 4 个且每个门的延迟为 T，请给出下列不同实现的 32 位加法器的延迟。

1) 行波进位加法器。

31*2T(加法器间进位延迟) + 3T(第一位全加器延迟) = 65T。

2) 4 位一块且块内并行、块间串行的加法器。

2T(产生 p，g) + 2T(块内产生进位) + 2T*7(块间产生进位) = 18T

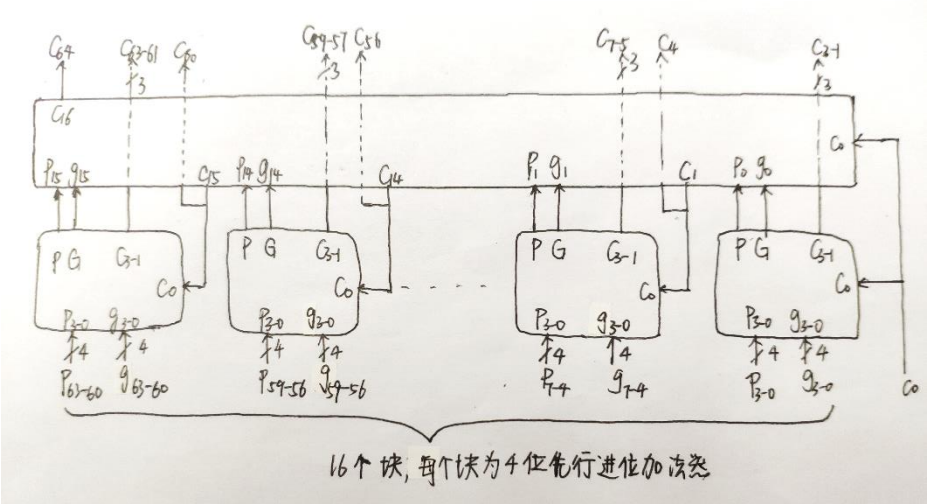
3) 4 位一块且块内并行、块间并行的加法器。

2T(产生 p,g) + 2T(产生下层 P,G) + 2T(产生进位输入) + 2T(产生进位输出) = 8T

13、作为设计者，在什么情况下会使用行波进位加法器而非先行进位加法器?

行波进位加法器电路结构简单，当设计的硬件要求尽量简单，不太考虑运行速度时会选择行波进位加法器。

14、请利用图 8.21 所示的 4 位先行进位逻辑组建出块内并行的 64 位先行进位加法器的进位逻辑，并证明其正确性。



15、举例说明$[X \times Y]_{补} \neq [X]_{补} \times [Y]_{补}$。

令 X = Y = -3 = -0011，$[X]_{补} = [Y]_{补} = 1101$，

$[X \times Y]_{补} = [0011 \times 0011]_{补} = 00001001$

$[X]_{补} \times [Y]_{补} = 1101 \times 1101 = 10101001$，故$[X \times Y]_{补} \neq [X]_{补} \times [Y]_{补}$。

16、证明$[X \times 2^n]_{补} = [X]_{补} \times 2^n$。

证明：设x的二进制表示为$\overline{a_0 a_1 a_2 a_3 \cdots a_n}$，其中$a_0$为符号位。

1. 当x为正数时，即$a_0 = 0$，x的补码与原码相等，即：

$$[x]_{补} = x = \overline{0 a_1 a_2 a_3 \cdots a_n}$$

$\times 2^n$表示左移n位，对x是正数没有影响，所以：

$$[x \times 2^n]_{补} = [x]_{补} \times 2^n$$

2. 当x为负数时，即$a_0 = 1$，则x的二进制表示为$\overline{1 a_1 a_2 a_3 \cdots a_n}$。
x的补码求法为：符号位不变，数值位取反再加1，所以：

$$[x]_{补} = \overline{1 b_1 b_2 b_3 \cdots (b_n + 1)}$$

其中，$b_i = 1 - a_i$

进一步有：

$$\begin{aligned} {}_{补} \times 2^n &= \overline{1 b_1 b_2 b_3 \cdots (b_n + 1)} \times 2^n \\ &= \overline{1 b_1 b_2 b_3 \cdots (b_n + 1) 00 \cdots 0} \end{aligned}$$

$$\begin{aligned} {}_{补} &= \overline{[1 a_1 a_2 a_3 \cdots a_n 00 \cdots 0]_{补}} \\ &= \overline{1 b_1 b_2 b_3 \cdots b_n 11 \cdots (1+1)} \\ &= \overline{1 b_1 b_2 b_3 \cdots (b_n + 1) 00 \cdots 0} \end{aligned}$$

所以有：

$$[x \times 2^n]_{补} = [x]_{补} \times 2^n$$

3. 当x为0时，显然成立。

17、假设每个“非门”“与非门”“或非门”的扇入不超过 4 个且每个门的延迟为 T，请给出下列不同实现将 4 个 16 位数相加的延迟。

（1） 使用多个先行进位加法器

4 个 16 位数相加，需要两层先行进位加法器，进位逻辑与 14 题相似。

生成 p，g 需要 2T，生成 P，G 需要 2T，上层生成块间金额为 i 需要 2T，下层的三亏块产生全部进位需要 2T，最终全加器做加法需要 3T，计算得两层加法器延迟共位 22T。

（2） 使用华莱士树及先行进位加法器

使用华莱士树将 4 个 16 位数相加转化为 16 个 4 位数相加，4 个数相加的华莱士树结构图见书中图 8.37 中的一个单元。此时需要 16 个该单元并行相加。

每一层华莱士树的延迟为 3T，每一单元为 2 层，故产生最后两个加数需要 6T，最后的全加器需要 11T，共需 17T。

18、请系统描述采用两位 Booth 编码和华莱士树的补码乘法器是如何处理[-X]和[-2X]的部分积。

首先，两位 Booth 乘法通过选择信号生成逻辑来根据乘数的低三位来生成若干部分积（假设有 m 个 n 位部分积），再通过华莱士树对部分积进行转置，变为 n 个 m 位部分积，再通过若干层华莱士树（每一位为若干个全加器），将操作数简化为 2 个，再用一个全加器将两个操作数按位相加。

19、用 Verilog 语言设计一个 32 位输入宽度的定点补码乘法器，要求使用 Booth 算法和华莱士树。

乘法模块 mul 调用 booth 算法模块 booth_2b 和 Wallace tree 模块 wallace_16，完成乘法运算。

```
module mul(
    input         mul_signed,       //为 1 时，表示有符号乘法，否则为无符号乘法
    input  [31:0] x,                //输入 x
    input  [31:0] y,                //输入 y
    output [63:0] result
);
    //booth 2 bits
    wire [16:0] booth_c;                              //Booth Couts
    wire [63:0] booth_part_prod[16:0];                //part products before switch
    wire [16:0] part_prod [63:0];                     //part products after switch
    wire [63:0] x_mul = {{32{x[31]&mul_signed}}, x}; //进行计算的乘数
    wire [33:0] y_mul = {{ 2{y[31]&mul_signed}}, y}; //进行计算的乘数

    genvar j;
    generate
    for(j = 0; j < 17; j = j + 1) begin : booth
        if(j == 0) begin
          booth_2b booth_0(
                          .Y({y_mul[1:0], 1'b0}),
                          .X(x_mul            ),
                          .P(booth_part_prod[0]),
                          .C(booth_c[0]       )
                         );
        end else begin
          booth_2b booth(
                          .Y(y_mul[2*j+1:2*j-1]),
                          .X(x_mul<<(2*j)      ),
```

```
                    .P(booth_part_prod[j]),
                    .C(booth_c[j]       )
                );
 end   end
endgenerate

//switch part products
genvar k;
generate
for(k = 0; k < 64; k = k + 1)    begin: switch
       assign part_prod[k] = {   booth_part_prod[16][k],
                                 booth_part_prod[15][k],
                                 booth_part_prod[14][k],
                                 booth_part_prod[13][k],
                                 booth_part_prod[12][k],
                                 booth_part_prod[11][k],
                                 booth_part_prod[10][k],
                                 booth_part_prod[ 9][k],
                                 booth_part_prod[ 8][k],
                                 booth_part_prod[ 7][k],
                                 booth_part_prod[ 6][k],
                                 booth_part_prod[ 5][k],
                                 booth_part_prod[ 4][k],
                                 booth_part_prod[ 3][k],
                                 booth_part_prod[ 2][k],
                                 booth_part_prod[ 1][k],
                                 booth_part_prod[ 0][k] };
    end
endgenerate

reg  [16:0] booth_c_r;             //booth couts from pipeline 1
reg  [16:0] part_prod_r[63:0];      //part prods from pipeline 1
wire [13:0] c[63:0];               //couts for wallace tree
wire [63:0] add_src1, add_src2;     //final add operands

always @(posedge mul_clk)    begin
    if(de_to_ex_valid & ex_allowin) begin
           booth_c_r <= booth_c;
end   end

genvar m;
generate
    for(m = 0; m < 64; m = m + 1) begin : part_prod_trans
        always @(posedge mul_clk)           begin
            if(de_to_ex_valid & ex_allowin) begin
                part_prod_r[m] <= part_prod[m];
            end
        end
    end
endgenerate

assign c[0] = booth_c_r[13:0];
assign add_src2[0] = booth_c_r[14];

genvar i;
generate
for(i = 0; i < 64; i = i + 1)    begin: wallace
    if(i < 63) begin
        wallace_16 wallace(
                              .cin    (c[i]         ),
                              .data_in(part_prod_r[i]),
                              .cout   (c[i+1]       ),
                              .C      (add_src2[i+1] ),
                              .S      (add_src1[i]   )
                          );
    end else begin
        wallace_16 wallace_63(
                              .cin    (c[63]         ),
                              .data_in(part_prod_r[63]),
                              .S      (add_src1[63]   )
```

```
                                );
        end
    end
    endgenerate

    assign result = add_src1 + add_src2 + booth_c_r[15] + booth_c_r[16];
endmodule

module booth_2b(      //booth 2bits algorithm
    input [2:0] Y,
    input [63:0] X,
    output [63:0] P,
    output C
);
    assign P = Y==3'b001 ? X      :
               Y==3'b010 ? X      :
               Y==3'b011 ? X<<1   :
               Y==3'b100 ? ~(X<<1):
               Y==3'b101 ? ~X     :
               Y==3'b110 ? ~X     :
                           68'd0;
    assign C = Y==3'b100 || Y==3'b101 || Y==3'b110;
endmodule

module wallace_16(        // Wallace tree
    input  [13:0] cin,
    input  [16:0] data_in,
    output [13:0] cout,
    output        C,
    output        S
);
    wire B1_2, A1_2, C1_3, B1_3, A1_3, A2_0, C2_1, B2_1, A2_1, B3_1, A3_1, A4_0, B4_0,
A5_0;
    function [1:0] Add_1;
        input A, B, CIN;      begin
          Add_1 = A + B + CIN;
        end
    endfunction
    assign {cout[ 0], B1_2} = Add_1(data_in[ 4], data_in[ 3], data_in[ 2]),
           {cout[ 1], A1_2} = Add_1(data_in[ 7], data_in[ 6], data_in[ 5]),
           {cout[ 2], C1_3} = Add_1(data_in[10], data_in[ 9], data_in[ 8]),
           {cout[ 3], B1_3} = Add_1(data_in[13], data_in[12], data_in[11]),
           {cout[ 4], A1_3} = Add_1(data_in[16], data_in[15], data_in[14]),
           {cout[ 5], A2_0} = Add_1(    cin[ 2],     cin[ 1],      cin[ 0]),
           {cout[ 6], C2_1} = Add_1(data_in[ 0],     cin[ 4],      cin[ 3]),
           {cout[ 7], B2_1} = Add_1(       A1_2,        B1_2,  data_in[ 1]),
           {cout[ 8], A2_1} = Add_1(       A1_3,        B1_3,         C1_3),
           {cout[ 9], B3_1} = Add_1(       A2_0,      cin[ 6],      cin[ 5]),
           {cout[10], A3_1} = Add_1(       A2_1,        B2_1,         C2_1),
           {cout[11], B4_0} = Add_1(    cin[ 9],      cin[ 8],      cin[ 7]),
           {cout[12], A4_0} = Add_1(       A3_1,        B3_1,       cin[10]),
           {cout[13], A5_0} = Add_1(       A4_0,        B4_0,       cin[11]),
           {C       , S    } = Add_1(       A5_0,      cin[13],      cin[12]);
endmodule
```

20、单精度和双精度浮点数能表示无理数 π 吗？为什么？

π 是无限小数，而单精度和双精度的位数有限，因此不能准确表示 π，只能在有限的位数里表示 π 的近似数。理论上二进制可以表示所有不可计算数，但是需要在无限位的前提下。