CA_Assignment2

Assign	
Property	
tag	homework
姓名	周鹏宇
学号	2019K8009929039

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：101001_2、001101_2、01011010_2、0000111010000101_2。

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

(3) 十六进制转换为十进制：$8\mathrm{AE}_{16}$、$\mathrm{C18D}_{16}$、$\mathrm{B379}_{16}$、100_{16}。

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

(1) 二进制转换为十进制：

$$101001_2 = 1 + 2^3 + 2^5 = 41_{10}$$

$$001101_2 = 1 + 2^2 + 2^3 = 13_{10}$$

$$01011010_2 = 2 + 2^3 + 2^4 + 2^6 = 90_{10}$$

$$0000111010000101_2 = 1 + 2^2 + 2^7 + 2^9 + 2^{10} + 2^{11} = 3717_{10}$$

(2) 十进制转换为二进制：

$$42_{10} = 2 + 2^3 + 2^5 = 101010_2$$

$$79_{10} = 1 + 2 + 2^2 + 2^3 + 2^6 = 1001111_2$$

$$811_{10} = 1 + 2 + 2^3 + 2^5 + 2^8 + 2^9 = 1100101011_2$$

$$374_{10} = 2 + 2^2 + 2^4 + 2^5 + 2^6 + 2^8 = 101110110_2$$

(3) 十六进制转换为十进制：

$$8\mathrm{AE16} = 8 \times 16^2 + 10 \times 16 + 14 = 2222_{10}$$

$$\mathrm{C18D16} = 12 \times 16^3 + 1 \times 16^2 + 8 \times 16 + 13 = 49549_{10}$$

$$\mathrm{B37916} = 11 \times 16^3 + 3 \times 16^2 + 7 \times 16 + 9 = 45945_{10}$$

$$10016 = 1 \times 16^2 = 256_{10}$$

(4) 十进制转换为十六进制：

$$81783_{10} = 1 \times 16^4 + 3 \times 16^3 + 15 \times 16^2 + 7 \times 16 + 16 = 13\mathrm{F77}_{16}$$

$$1922_{10} = 7 \times 16^2 + 8 \times 16 + 2 = 782_{16}$$

$$345208_{10} = 5 \times 16^4 + 4 \times 16^3 + 4 \times 16^2 + 7 \times 16 + 8 = 54478_{16}$$

$5756_{10} = 1 \times 16^3 + 6 \times 16^2 + 7 \times 16 + 12 = 167\mathrm{C}_{16}$

8.2

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：$0 \sim 2^{32} - 1$

原码：$-2^{31} + 1 \sim 2^{31} - 1$

补码：$-2^{31} \sim 2^{31} - 1$

8.3

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}。

Aa 十进制数	8 位原码	8位补码
45_{10}	00101101_2	00101101_2
-59_{10}	10111011_2	11000101_2
-128_{10}	溢出	10000000_2
119_{10}	01110111_2	01110111_2
127_{10}	01111111_2	01111111_2
128_{10}	溢出	溢出
0_{10}	00000000_2或 11111111_2	00000000_2
-1_{10}	10000001_2	11111111_2

8.4

请将下列数据分别视为原码和补码，从 8 位扩展到 16 位：00101100_2、11010100_2、10000001_2、00010111_2。

Aa 数据	按原码扩展到 16 位	按补码扩展到 16 位	按无符号原码扩展到16位
00101100_2	0000000000101100_2	0000000000101100_2	0000000000101100_2
11010100_2	1000000001010100_2	1111111111010100_2	0000000001010100_2
10000001_2	1000000000000001_2	1111111110000001_2	0000000001010100_2
00010111_2	0000000000010111_2	0000000000010111_2	0000000000010111_2

8.5

请将下列浮点数在不同进制间进行转换。

(1) 十进制数转换为单精度数：0、116.25、-4.375。

(2) 十进制数转换为双精度数：-0、116.25、-2049.5。

(3) 单精度数转换为十进制数：0xff800000、0x7fe00000。

(4) 双精度数转换为十进制数：0x8008000000000000、0x7065020000000000。

(1) 十进制数转换为单精度数：

- 0：其单精度数表示为 0x00000000（零的阶码全 0、尾数全 0，正零的符号为 0）
- 116.25：$116.25_{10} = 1110100.01_2 = 1.11010001 \times 2^6$，其符号为 0，阶码为 $6 + 127 = 133_{10} = 10000101_2$，尾数为 11010001000000000000000，故其单精度数表示为 0x42E88000。
- -4.375：$-4.375_{10} = -100.011_2 = -1.00011 \times 2^2$，其符号为 1，阶码为 $2 + 127 = 129_{10} = 10000001_2$，尾数为 00011000000000000000000，故其单精度数表示为 0xC08C0000。

(2) 十进制数转换为双精度数：

- -0：其双精度数表示为 0x8000000000000000（零的阶码全 0、尾数全 0，负零的符号为 1）
- 116.25：$116.25_{10} = 1110100.01_2 = 1.11010001 \times 2^6$，
  - 其符号为 0，
  - 阶码为 $6 + 1023 = 1029_{10} = 10000000101_2$，
  - 尾数为 1101000100000000000000000000000000000000000000000000，故其双精度数表示为 0x405D100000000000。
- -2049.5：$-2049.5_{10} = -100000000001.1_2 = -1.000000000011 \times 2^{11}$，
  - 其符号为 1，
  - 阶码为 $6 + 1023 = 1029_{10} = 10000000101_2$，
  - 尾数为 0000000000110000000000000000000000000000000000000000，故其双精度数表示为 0xC050030000000000。

(3) 单精度数转换为十进制数：

- 0xff800000：二进制表示为 11111111100000000000000000000000
  - 其符号为 1，
  - 阶码为 $11111111_2 = 255_{10}$，
  - 尾数为 00000000000000000000000，（阶码全 1、尾数全 0、符号为 1），故其表示的是“负无穷”。
- 0x7fe00000：二进制表示为 01111111111100000000000000000000
  - 其符号为 0，
  - 阶码为 $11111111_2 = 255_{10}$，

- 尾数为 11000000000000000000，（阶码全 1，尾数非 0）故其表示的是“非数”。

(4) 双精度数转换为十进制数：

- $0\text{x}8008000000000000$：二进制表示为 $1000000000001000000000000000000000000000000000000000000000000000$
  - 其符号为 1，
  - 阶码为 $00000000000_2 = 0$，
  - 尾数为 $1000000000000000000000000000000000000000000000000000$，故其表示的是 $-0.1_2 \times 2^{-216} = -0.5 \times 2^{-216}$，是一个非规格化非 0 负数。
- $0\text{x}7065020000000000$：二进制表示为 $0111000001100101000000100000000000000000000000000000000000000000$
  - 其符号为 0，
  - 阶码为 $11100000110_2 = 255_{10}$，
  - 尾数为 $0101000000100000000000000000000000000000000000000000$，故其表示的是 $1.01010000001_2 \times 2^{775}$。

8.6

请写出下图（图略）所示晶体管级电路图的真值表，并给出对应的逻辑表达式。

Aa Title	A	B	C	Y
Untitled	0	0	0	1
Untitled	0	0	1	1
Untitled	0	1	0	1
Untitled	0	1	1	0
Untitled	1	0	0	1
Untitled	1	0	1	0
Untitled	1	1	0	1
Untitled	1	1	1	0

$Y = \neg((A \vee B) \wedge C)$

8.7

请写出下图（图略）所示逻辑门电路图的真值表。

Aa Title	A	B	C	Y
Untitled	0	0	0	1

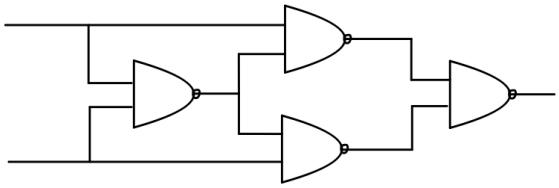
Title	A	B	C	Y
Untitled	0	0	1	1
Untitled	0	1	0	1
Untitled	0	1	1	0
Untitled	1	0	0	1
Untitled	1	0	1	0
Untitled	1	1	0	1
Untitled	1	1	1	0

$Y = \neg((A \vee B) \wedge C)$

8.8

请用尽可能少的而输入 NAND 门搭建出一个具有二输入 XOR 功能的电路

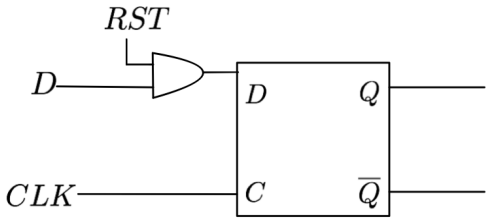
$A \oplus B = (A \wedge \neg B) \vee (\neg A \wedge B) = (A \wedge (\neg A \vee \neg B)) \vee (B \wedge (\neg A \vee \neg B))$



$Y = A \oplus B$

8.9

请用 D 触发器和常见组合逻辑门搭建出一个具有同步复位为 0 功能的触发器的电路。



8.10

证明 $[X+Y]_{补} = [X]_{补} + [Y]_{补}$。

根据补码的定义，有：

$$[X]_{补} + [Y]_{补} = 2^{n+1} + (X+Y) \pmod{2^n}$$

又n位补码额外加2^n无影响，故

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

8.11

证明 $[X-Y]_{补}=[X]_{补}+[-Y]_{补}$

根据补码的定义，有：

$[X]_{补}= \quad (2^n+X) \pmod{2^n}$

$[-Y]_{补}=(2^n-Y) \pmod{2^n}$

则自然有

$[X]_{补}+[-Y]_{补}=2^{2n+1}+(X-Y) \pmod{2^n}=[X-Y]_{补}$

8.12

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；

- 从最低位的输入 A_0、B_0、C_{in} 到最高位的进位输出 C_{out} 的延迟为 $2\times T\times 32=64T$；
- 从最低位的输入 A_0、B_0、C_{in} 到最高位的进位输入 $C_{31,\text{in}}$ 的延迟为 $2\times T\times(32-1)=62T$；
- 从最高位的输入 A_{31}、B_{31}、$C_{31,\text{in}}$ 到最高位的加和输出 S_{31} 的延迟为 $3T$；
- 故从最低位的输入 A_0、B_0、C_{in} 到最高位的加和输出 S_{31} 的延迟为 $62T+3T=65T$。

(2) 4 位一块且块内并行、块间串行的加法器；

- 从最低进位 c_0到最高进位输入c_{31}的延迟为$7\times 4T+2T=30T$
- 从最高进位输入 c_{31}到最高位和输出 s_{31} 的延迟为 $3T$
- 生成 p_i,g_i 产生 $2T$ 的延迟
- 总延迟为$30T+2T+3T=65T$

(3) 4 位一块且块内并行、块间并行的加法器。

- 生成 p_i,g_i 产生 $2T$ 的延迟
- 全加器产生$3T$延迟
- 进位生成过程中生成两层 P,G 需 $2\times 2T$，生成各层 c_1,c_2,c_3 需 $3\times 2T$
- 共计$15T$

8.13

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？

行波进位加法器所需门的数量远小于先行进位加法器，但延时远大于先行进位加法器。

在逻辑资源有限、时序要求较低的情况下可采用行波进位加法器而非先行进位加法器。

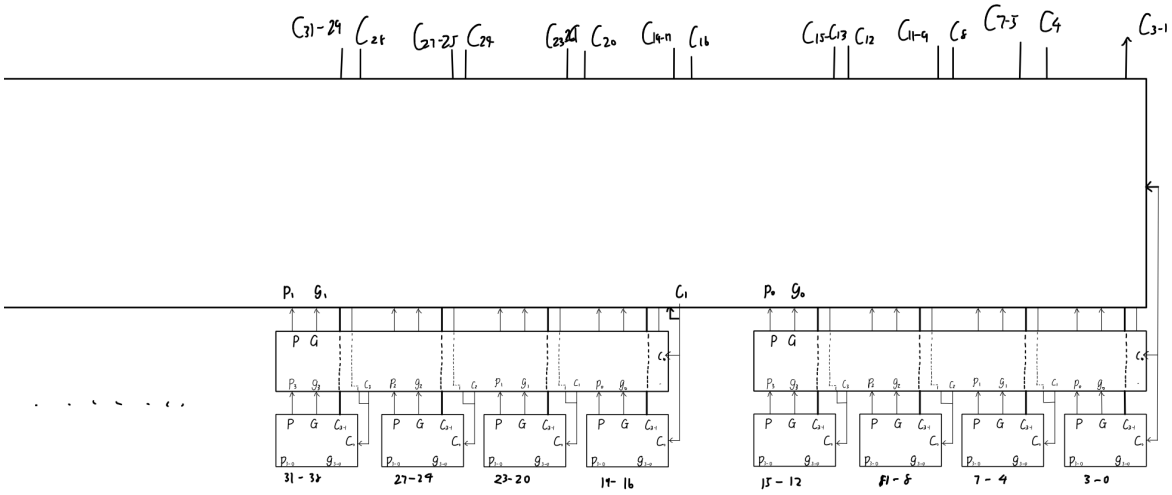
8.14

请利用图 8.21 所示的 4 位先行进位逻辑组件出块内并行且块间并行的 64 位先行进位加法器的进位逻辑，并证明其正确性。

进位逻辑图的构建见下页图。首先将 64 位数分为 16 组 4 位数生成 p_i 与 g_i

- 第一层的 4 位先行进位逻辑生成各组的块间的 G、 P 以及第 $4i+1 \sim 4i+3 (i = 0,1,2,\cdots,15)$ 位上的进位输入信号
- 第二层即可看作四组 4 位数的p_i , g_i 作为输入，由此生成 P , G 用于生成最后 的进位输出以及第 $4i (i = 1,2,3,\cdots,15)$ 位上进位输出
- 第三层总结四组(看作四位数)的进位因子， 生成 c_{64} 即 c_{out}。

由于结构的高度重复性，故只给出部分的结构图



8.15

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

考虑 5 位补码相乘得到 10 位补码，$X = -6$，$Y = -4$，则 $X \times Y = 24$。

那么 $[X]_{补} = 1010_2$，$[Y]补 = 1100_2$，$[X \times Y]_{补} = 00011000_2$。

但是 $[X]_{补} \times [Y]_{补} = 01111000_2 \pmod{2^4}$，故 $[X \times Y]_{补} \neq [X]_{补} \times [Y]_{补}$

8.16

请证明 $[X \times 2^n]_{补} = [X]_{补} \times 2^n$

考虑 m 位补码的情况

根据补码定义 $[X \times 2^n]_{补} = 2^k + X \times 2^n \pmod{2^k}$ $[X]_{补} \times 2^n = (2^k + X) \times 2^n = 2^n \times 2^k + X \times 2^n = 2^k + X \times 2^n \pmod{2^k}$ 故 $[X \times 2^n]_{补} = [X]_{补} \times 2^n$

8.17

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；

共需 2 级先行进位加法器延迟，每级先行进位加法器延迟为 $11T$，故总延迟为 $22T$

(2) 使用华莱士树及先行进位加法器。

- 华莱士树将 4 个 16 位数相加转换为 2 个 16 位数相加，其延迟为 $6T$；
- 两个数相加需要 1 级先行进位加法器，其延迟为 $11T$
- 故总延迟为 $17T$

8.18

请系统描述采用两位 Booth 编码和华莱士树的补码乘法是如何处理 $[-X]_{补}$ 和 $[-2X]_{补}$ 的部分 积的。

两位 Booth 编码下部分积的生成形式为 $Y = y_n-1 \cdots y_0 + c_0$，其中对于 $[-X]_{补}$ 则将各位取反， 将 c_0 置 1;对于 [−2X]_补 则同样将 c_0 置 1，将各位取反后作为高一位的值，高位丢弃，低位置 0。

8.19

- 华莱士树

```
`timescale 10ns / 1ns

module wallace_16(
    input [16:0] num,
    input [13:0] cin,
    output [13:0] cout,
    output c,
    output s
);

    /* level 1 wires */
    wire sL1_1;
    wire sL1_2;
    wire sL1_3;
    wire sL1_4;
    wire sL1_5;

    /* level 2 wires */
    wire sL2_1;
    wire sL2_2;
    wire sL2_3;
    wire sL2_4;

    /* level 3 wires */
    wire sL3_1;
    wire sL3_2;

    /* level 4 wires */
    wire sL4_1;
    wire sL4_2;
```

```
    /* level 5 wires */
    wire sL5_1;

    /* level 1 adders */
    adder adderL1_1(
        .a(num[16]),
        .b(num[15]),
        .c(num[14]),
        .cout(cout[0]),
        .s(sL1_1)
    );

    adder adderL1_2(
        .a(num[13]),
        .b(num[12]),
        .c(num[11]),
        .cout(cout[1]),
        .s(sL1_2)
    );

    adder adderL1_3(
        .a(num[10]),
        .b(num[9]),
        .c(num[8]),
        .cout(cout[2]),
        .s(sL1_3)
    );

    adder adderL1_4(
        .a(num[7]),
        .b(num[6]),
        .c(num[5]),
        .cout(cout[3]),
        .s(sL1_4)
    );

    adder adderL1_5(
        .a(num[4]),
        .b(num[3]),
        .c(num[2]),
        .cout(cout[4]),
        .s(sL1_5)
    );

    /* level 2 adders */
    adder adderL2_1(
        .a(sL1_1),
        .b(sL1_2),
        .c(sL1_3),
        .cout(cout[5]),
        .s(sL2_1)
    );

    adder adderL2_2(
        .a(sL1_4),
        .b(sL1_5),
        .c(num[1]),
        .cout(cout[6]),
        .s(sL2_2)
    );

    adder adderL2_3(
        .a(num[0]),
        .b(cin[0]),
```

```
        .c(cin[1]),
        .cout(cout[7]),
        .s(sL2_3)
    );

    adder adderL2_4(
        .a(cin[2]),
        .b(cin[3]),
        .c(cin[4]),
        .cout(cout[8]),
        .s(sL2_4)
    );

    /* level 3 adders */
    adder adderL3_1(
        .a(sL2_1),
        .b(sL2_2),
        .c(sL2_3),
        .cout(cout[9]),
        .s(sL3_1)
    );

    adder adderL3_2(
        .a(sL2_4),
        .b(cin[5]),
        .c(cin[6]),
        .cout(cout[10]),
        .s(sL3_2)
    );

    /* level 4 adders */
    adder adderL4_1(
        .a(sL3_1),
        .b(sL3_2),
        .c(cin[7]),
        .cout(cout[11]),
        .s(sL4_1)
    );

    adder adderL4_2(
        .a(cin[8]),
        .b(cin[9]),
        .c(cin[10]),
        .cout(cout[12]),
        .s(sL4_2)
    );

    /* level 5 adders */
    adder adderL5_1(
        .a(sL4_1),
        .b(sL4_2),
        .c(cin[11]),
        .cout(cout[13]),
        .s(sL5_1)
    );

    /* level 6 adders*/
    adder adderL6_1(
        .a(sL5_1),
        .b(cin[12]),
        .c(cin[13]),
        .cout(c),
        .s(s)
    );
```

```
endmodule
```

- 加法器

```
`timescale 10ns / 1ns

module adder(
    input a,
    input b,
    input c,
    output cout,
    output s
);

assign s = a ^ b ^ c;
assign cout = a & b | a & c | b & c;

endmodule
```

- booth乘法器

```
`timescale 10ns / 1ns

 module BoothGen(
     input [2:0] y,
     input [63:0] x,
     output [63:0] p,
     output c
 );

    wire smx;    //S_minus x
    wire sx;     //S_x
    wire smdx;   //S_minus double x
    wire sdx;    //S_double x

    wire [63:0] not_x;

    assign smx = (y[2] & y[1] & ~y[0]) | (y[2] & ~y[1] & y[0]);
    assign sx = (~y[2] & y[1] & ~y[0]) | (~y[2] & ~y[1] & y[0]);
    assign smdx = y[2] & ~y[1] & ~y[0];
    assign sdx = ~y[2] & y[1] & y[0];

    assign not_x = ~x;

    assign c = smx | smdx;
    assign p =      ({64{smx}} & not_x)
                |   ({64{smdx}} & {not_x[62:0], 1'b1})
                |   ({64{sx}} & x)
                |   ({64{sdx}} & {x[62:0], 1'b0});

 endmodule
```

- 处理输入

```
`timescale 10ns / 1ns

module entry(
    input [32:0] x,
    input [32:0] y,

    output [63:0] p1,
    output [63:0] p2,
    output [63:0] p3,
    output [63:0] p4,
    output [63:0] p5,
    output [63:0] p6,
    output [63:0] p7,
    output [63:0] p8,
    output [63:0] p9,
    output [63:0] p10,
    output [63:0] p11,
    output [63:0] p12,
    output [63:0] p13,
    output [63:0] p14,
    output [63:0] p15,
    output [63:0] p16,
    output [63:0] p17,

    output c1,
    output c2,
    output c3,
    output c4,
    output c5,
    output c6,
    output c7,
    output c8,
    output c9,
    output c10,
    output c11,
    output c12,
    output c13,
    output c14,
    output c15,
    output c16,
    output c17
);

    wire [63:0] x_extended;

    assign x_extended[63:33] = {31{x[32]}};
    assign x_extended[32:0] = x;

    partialGen pg1(
        .y({y[1:0], 1'b0}),
        .x(x_extended),
        .p(p1),
        .c(c1)
    );

    partialGen pg2(
        .y(y[3:1]),
        .x(x_extended << 2),
        .p(p2),
        .c(c2)
    );

    partialGen pg3(
```

```
        .y(y[5:3]),
        .x(x_extended << 4),
        .p(p3),
        .c(c3)
    );

    partialGen pg4(
        .y(y[7:5]),
        .x(x_extended << 6),
        .p(p4),
        .c(c4)
    );

    partialGen pg5(
        .y(y[9:7]),
        .x(x_extended << 8),
        .p(p5),
        .c(c5)
    );

    partialGen pg6(
        .y(y[11:9]),
        .x(x_extended << 10),
        .p(p6),
        .c(c6)
    );

    partialGen pg7(
        .y(y[13:11]),
        .x(x_extended << 12),
        .p(p7),
        .c(c7)
    );

    partialGen pg8(
        .y(y[15:13]),
        .x(x_extended << 14),
        .p(p8),
        .c(c8)
    );

    partialGen pg9(
        .y(y[17:15]),
        .x(x_extended << 16),
        .p(p9),
        .c(c9)
    );

    partialGen pg10(
        .y(y[19:17]),
        .x(x_extended << 18),
        .p(p10),
        .c(c10)
    );

    partialGen pg11(
        .y(y[21:19]),
        .x(x_extended << 20),
        .p(p11),
        .c(c11)
    );

    partialGen pg12(
        .y(y[23:21]),
```

```
        .x(x_extended << 22),
        .p(p12),
        .c(c12)
    );

    partialGen pg13(
        .y(y[25:23]),
        .x(x_extended << 24),
        .p(p13),
        .c(c13)
    );

    partialGen pg14(
        .y(y[27:25]),
        .x(x_extended << 26),
        .p(p14),
        .c(c14)
    );

    partialGen pg15(
        .y(y[29:27]),
        .x(x_extended << 28),
        .p(p15),
        .c(c15)
    );

    partialGen pg16(
        .y(y[31:29]),
        .x(x_extended << 30),
        .p(p16),
        .c(c16)
    );

    partialGen pg17(
        .y({y[32], y[32:31]}),
        .x(x_extended << 32),
        .p(p17),
        .c(c17)
    );

endmodule
```

- 顶层文件

```
`timescale 10ns / 1ns

module mul_top(
    input mul_clk,
    input resetn,
    input mul_signed,
    input [31:0] x,
    input [31:0] y,
    output [63:0] result
);

    wire [63:0] p1;
    wire [63:0] p2;
    wire [63:0] p3;
    wire [63:0] p4;
    wire [63:0] p5;
    wire [63:0] p6;
```

```
    wire [63:0] p7;
    wire [63:0] p8;
    wire [63:0] p9;
    wire [63:0] p10;
    wire [63:0] p11;
    wire [63:0] p12;
    wire [63:0] p13;
    wire [63:0] p14;
    wire [63:0] p15;
    wire [63:0] p16;
    wire [63:0] p17;

    wire c1;
    wire c2;
    wire c3;
    wire c4;
    wire c5;
    wire c6;
    wire c7;
    wire c8;
    wire c9;
    wire c10;
    wire c11;
    wire c12;
    wire c13;
    wire c14;
    wire c15;
    wire c16;
    wire c17;

    wire [32:0] x_input;
    wire [32:0] y_input;
    wire [63:0] s;
    wire [63:0] c;
    wire [63:0] final_result;
    wire cout1;
    wire cout2;

    assign x_input = (mul_signed == 1'b1) ? {x[31], x} : {1'b0, x};
    assign y_input = (mul_signed == 1'b1) ? {y[31], y} : {1'b0, y};

    reg [63:0] c_reg;
    reg [63:0] s_reg;
    reg [1:0] cin_reg;

    always @(posedge mul_clk)
    begin
        if (resetn == 1'b0)
        begin
            c_reg <= 64'b0;
            s_reg <= 64'b0;
            cin_reg <= 2'b0;
        end
        else
        begin
            c_reg <= c;
            s_reg <= s;
            cin_reg <= {cout2, cout1};
        end
    end

    entry entry_module(
        .x(x_input),
        .y(y_input),
```

```
        .p1(p1),
        .p2(p2),
        .p3(p3),
        .p4(p4),
        .p5(p5),
        .p6(p6),
        .p7(p7),
        .p8(p8),
        .p9(p9),
        .p10(p10),
        .p11(p11),
        .p12(p12),
        .p13(p13),
        .p14(p14),
        .p15(p15),
        .p16(p16),
        .p17(p17),
        .c1(c1),
        .c2(c2),
        .c3(c3),
        .c4(c4),
        .c5(c5),
        .c6(c6),
        .c7(c7),
        .c8(c8),
        .c9(c9),
        .c10(c10),
        .c11(c11),
        .c12(c12),
        .c13(c13),
        .c14(c14),
        .c15(c15),
        .c16(c16),
        .c17(c17)
    );
```

8.20

单精度和双精度浮点数能表示无理数 π 吗？为什么？

单精度、双精度浮点数本质上是有限小数，不能表示无限不循环小数 π,但可以近似表示