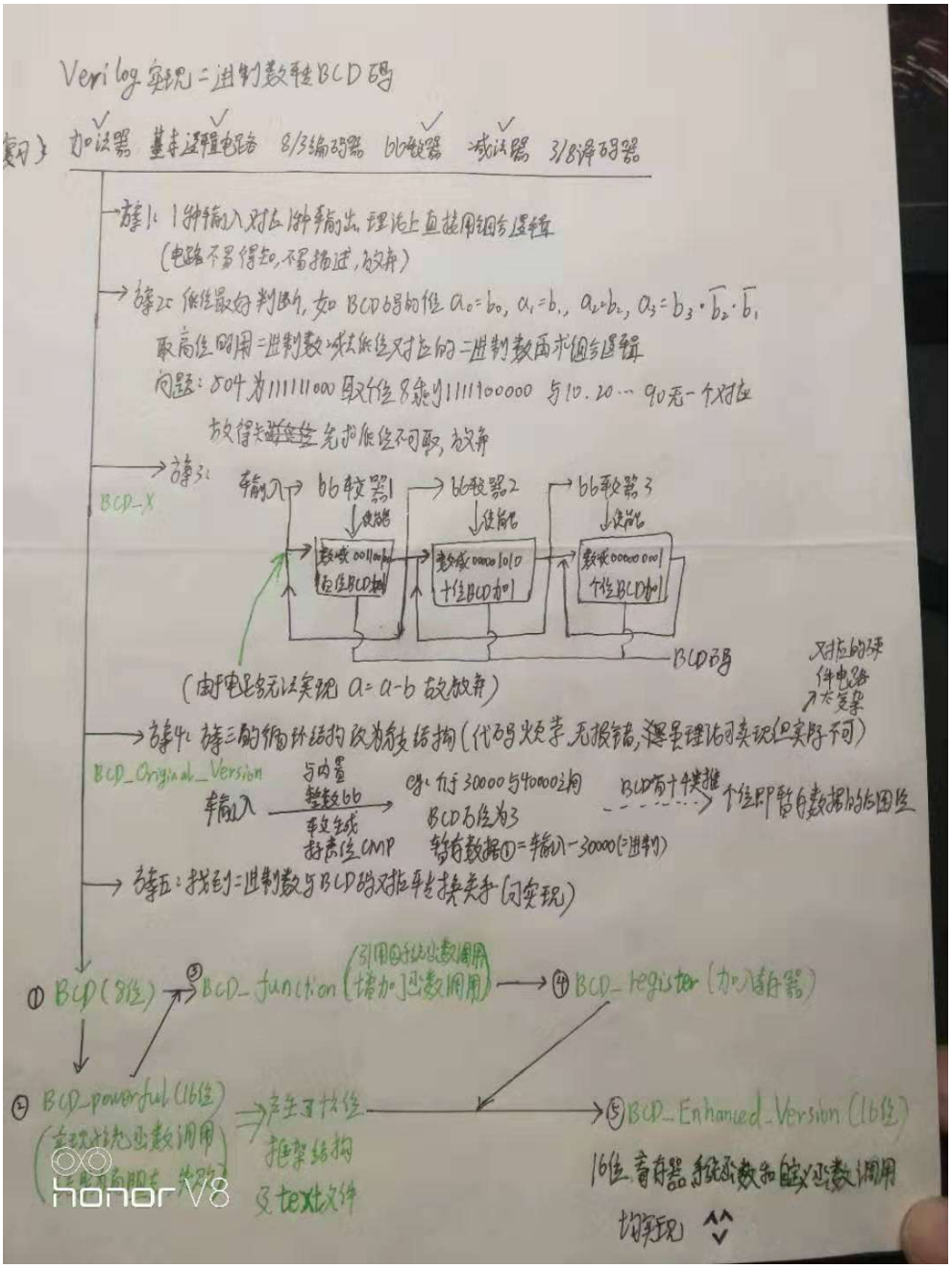
用 Verilog 语言

将 16 位二进制数转换为 20 位 BCD 码

一、坎坷的设计过程



二、关于数据类型、系统功能调用、寄存器以及函数调用

1. 数据类型

外部输入输出端口的分别用 input 和 output 定义。（两种定义方式括号里直接定义，括号里申明括号外定义）。

内部电路系统输入用 reg，输出用 wire。

函数中的输入输出无需再函数外定义。

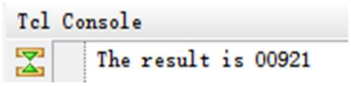
仿真源文件中只包含外电路输入输出端口，输入用 reg 输出用 wire（注：无 input 和 output）

2. 系统功能调用

调用系统功能$display 在 Tcl Console 中输出结果，$display 语句必须在仿真源文件的路径下，其语句结构如下。

```
$display("The result is %h",result);
```

其中引号部分为输出，%h 表示按照 16 进制方式输出一个数 result。结果如下。



3.寄存器的应用

```
module BCD_Remainder(
input [15:0] binary,
input CLK,
input CE,
input PRE,
output reg [19:0] result
);

reg [15:0] delybinary;
reg CMP;

always@(posedge CLK or posedge PRE)
begin
if(PRE) delybinary <= 16'b1111111111111111;
else if(CE) delybinary <= binary;
end

always@(*)
begin
 CMP = (delybinary==binary) ? 1'b1:1'b0;
 if (CMP) result = back (delybinary) ;
 else result = 0;
end
endmodule
```

定义了寄存器时钟输入端 CLK，寄存器使能端 CE，寄存器置位端 PRE，寄存器输入端 binary，输出端 delybinary。

always@(posedge CLK or posedge PRE)其中的敏感变量是 CLK 或者 PRE 的上升沿，当 PRE 上升沿有效立即置位，否则在 CLK 为上升沿并且使能端 CE 为 1 有效时输入输出相等。

下面即是寄存器输入输出相等时调用函数 back 得出结果 result。寄存器输入输出不等则清零。

3. 函数调用

格式如下

```
function [a:b] back;   //返回值，也是函数名，[a:b]为返回值大小
input [] a;//形参可以有多个，即函数输入
Input [] b;
reg/wire ..... ;   //定义函数中用到的非端口变量
begin
    (函数体)：
end
endfunction   //结束
```

（注）函数可以使代码变简介，但并不会简化电路。

如果需要多个返回值可以先用并置运算符{}，将多个返回值合并为一个长的返回值。

三、设计源文件（四种算法与其仿真结果）

仿真用的源文件除例化语句外均相同，最后附。

1、 取余法

代码和思路都是最简单的.

例如：用输入 54219 除以一万，商就是它的万位 5，余数为 4219。

再用余数 4219 除以一千，商就是他的千位 4，余数为 219。

如此循环往复即可得出结果。

这里的商最大是 9 所以表示为四位二进制数。

最后用并置运算符将 5 个 4 位二进制数连接即为 54219 的 BCD 码。

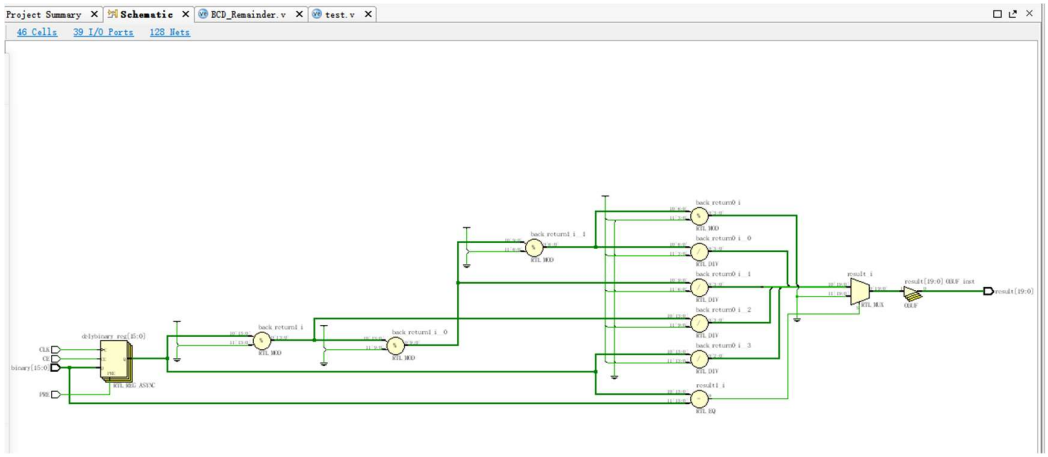
代码如下：

```
module BCD_Remainder(
input [15:0] binary,
input CLK,
input CE,
input PRE,
output reg [19:0] result
);

reg [15:0] delybinary;
reg CMP;
```

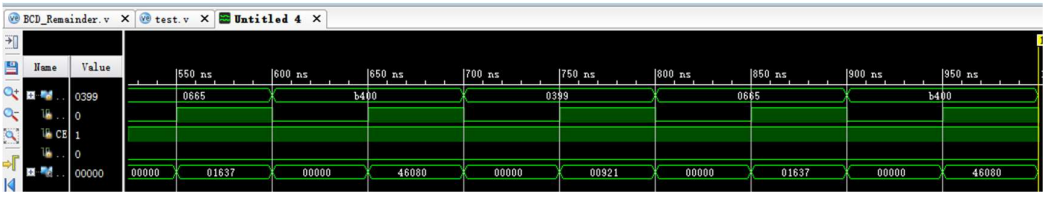
```
function [19:0] back;    //back  函数输出
input [15:0] in;             //函数输入

      reg [15:0] cw;
      reg [3:0] sw;
      reg [15:0] yw;
      reg [15:0] cq;
      reg [3:0] sq;
      reg [15:0] yq;
      reg [15:0] cb;
      reg [3:0] sb;
      reg [15:0] yb;
      reg [15:0] cs;
      reg [3:0] ss;
      reg [15:0] ys;


begin
         cw=16'b0010011100010000;
         cq=16'b0000001111101000;
         cb=16'b0000000001100100;
         cs=16'b0000000000001010;

         sw=in/cw;
         yw=in%cw;
         sq=yw/cq;
         yq=yw%cq;
         sb=yq/cb;
         yb=yq%cb;
         ss=yb/cs;
         ys=yb%cs;
         back={sw,sq,sb,ss,ys[3],ys[2],ys[1],ys[0]};
end
endfunction




always@(posedge CLK or posedge PRE)
begin
if(PRE) delybinary <= 16'b1111111111111111;
else if(CE) delybinary <= binary;
end
```

```
always@(*)
begin
  CMP = (delybinary==binary) ? 1'b1:1'b0;
  if (CMP) result = back (delybinary) ;
  else result = 0;
end
Endmodule
```

电路结构如下：



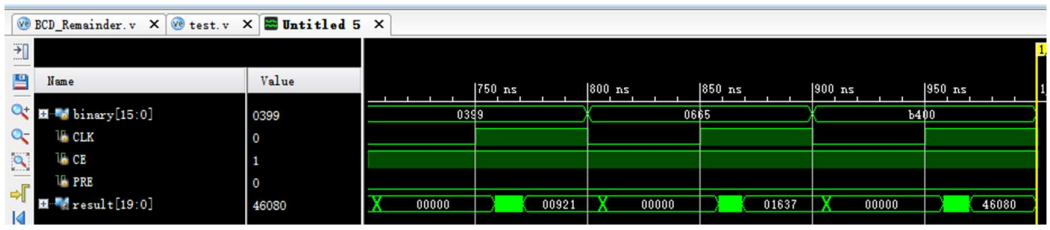
行为级仿真结果如下：



（注）寄存器输入与输出不同时清零。

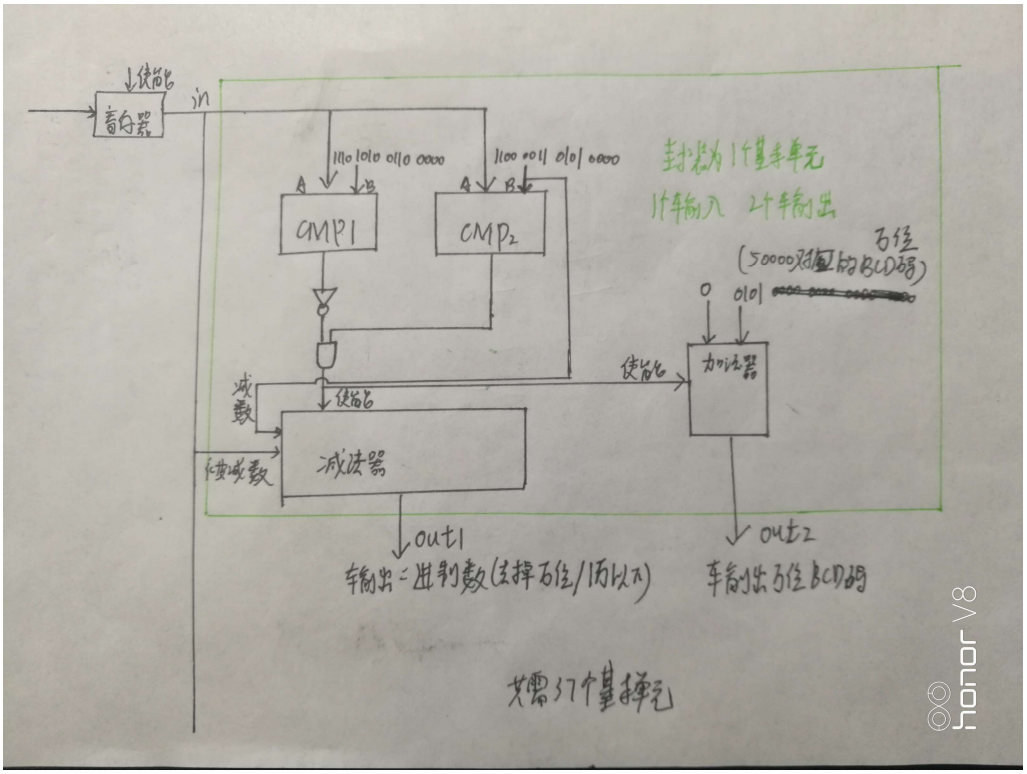
以下行为级仿真均相同不再赘述。

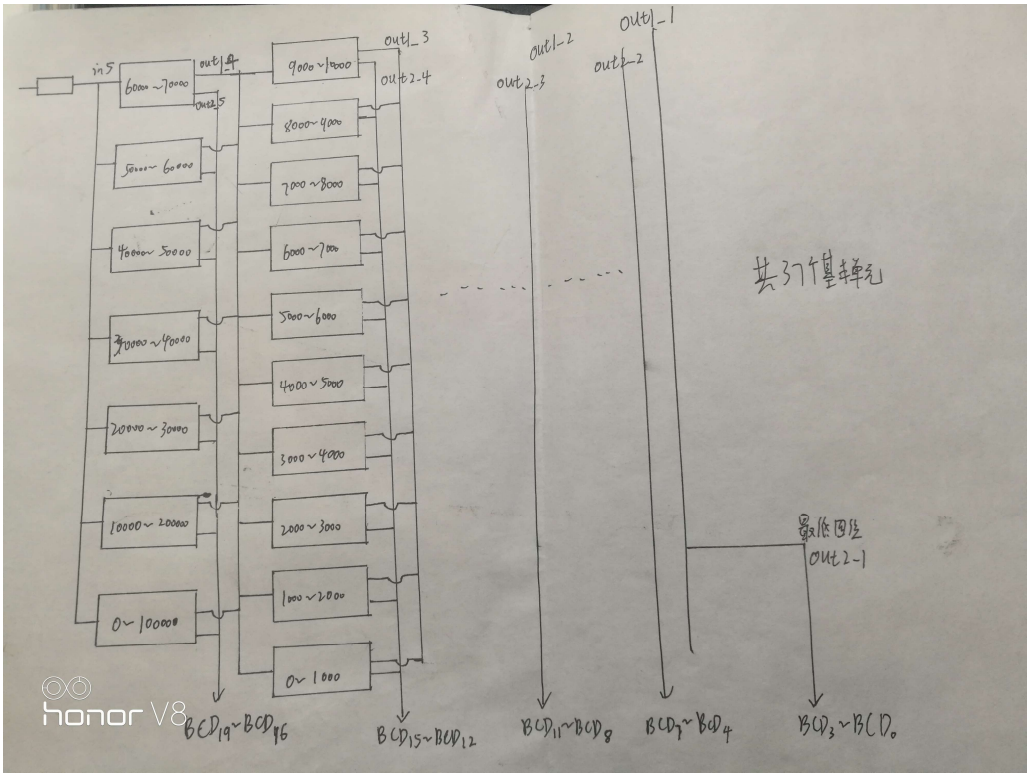
时序仿真结果如下。



2、 比较法

只用比较和加减法不用除法如何实现？设计思路如下。





通过分段去比较的方法一位一位提取。
举例 25500 提取万位时，第一列只有对应的 20000 到 30000 的基本单元导通，其他万位的基本单元均截止，输出线虽然并联但不会产生数据冲突（如 a=a-1 是不可实现的）。输出的 BCD 码即为在加法器中预留的 BCD 码，下一级输入即为本级输入减去这里的 20000，其方法本质上还是取余。第二列只有 5000 到 6000 的结构单元导通。以此类推。

对比写代码时的顺序结构和函数调用。

顺序结构版本如下：

```
module BCD_Original_Version_Simply(
    input [15:0] binary,
    input CLK,
    input CE,
    input PRE,
    output reg [19:0] result
);

reg [15:0] delybinary;   //寄存器输出
reg CMP;


function [19:0] back;   //*********************************************back 函数输出
input [15:0] delybinary;   //delybinary 函数输入

reg [19:0] binarythousands;    //最高位是 X 位的输入
reg [19:0] binaryhudreds;
reg [19:0] binarytens;
reg [19:0] binaryones;
reg [3:0] TenThousands;   //BCD 的每一位
reg [3:0] Thousands;
reg [3:0] Hundreds;
reg [3:0] Tens;
reg [3:0] Ones;

reg CMP60000;      //比较标志位
reg CMP50000;
reg CMP40000;
reg CMP30000;
reg CMP20000;
reg CMP10000;
reg CMP9000;
```

```
reg CMP8000;
reg CMP7000;
reg CMP6000;
reg CMP5000;
reg CMP4000;
reg CMP3000;
reg CMP2000;
reg CMP1000;
reg CMP900;
reg CMP800;
reg CMP700;
reg CMP600;
reg CMP500;
reg CMP400;
reg CMP300;
reg CMP200;
reg CMP100;
reg CMP90;
reg CMP80;
reg CMP70;
reg CMP60;
reg CMP50;
reg CMP40;
reg CMP30;
reg CMP20;
reg CMP10;

reg [15:0] A60000;      //比较参考   被减数
reg [15:0] A50000;
reg [15:0] A40000;
reg [15:0] A30000;
reg [15:0] A20000;
reg [15:0] A10000;
reg [15:0] A9000;
reg [15:0] A8000;
reg [15:0] A7000;
reg [15:0] A6000;
reg [15:0] A5000;
reg [15:0] A4000;
reg [15:0] A3000;
reg [15:0] A2000;
reg [15:0] A1000;
reg [15:0] A900;
reg [15:0] A800;
```

```
reg [15:0] A700;
reg [15:0] A600;
reg [15:0] A500;
reg [15:0] A400;
reg [15:0] A300;
reg [15:0] A200;
reg [15:0] A100;
reg [15:0] A90;
reg [15:0] A80;
reg [15:0] A70;
reg [15:0] A60;
reg [15:0] A50;
reg [15:0] A40;
reg [15:0] A30;
reg [15:0] A20;
reg [15:0] A10;

begin
    back=20'd0;             //赋初值
    TenThousands=4'd0;
    Thousands=4'd0;
    Hundreds=4'd0;
    Tens=4'd0;
    Ones=4'd0;

    A60000= 16'b1110101001100000;           //参数赋值
    A50000= 16'b1100001101010000;
    A40000= 16'b1001110001000000;
    A30000= 16'b0111010100110000;
    A20000= 16'b0100111000100000;
    A10000= 16'b0010011100010000;

    A9000= 16'b0010001100101000;
    A8000= 16'b0001111101000000;
    A7000= 16'b0001101101011000;
    A6000= 16'b0001011101110000;
    A5000= 16'b0001001110001000;
    A4000= 16'b0000111110100000;
    A3000= 16'b0000101110111000;
    A2000= 16'b0000011111010000;
    A1000= 16'b0000001111101000;

    A900= 16'b0000001110000100;
    A800= 16'b0000001100100000;
```

```
    A700= 16'b0000001010111100;
    A600= 16'b0000001001011000;
    A500= 16'b0000000111110100;
    A400= 16'b0000000110010000;
    A300= 16'b0000000100101100;
    A200= 16'b0000000011001000;
    A100= 16'b0000000001100100;

    A90= 16'b0000000001011010;
    A80= 16'b0000000001010000;
    A70= 16'b0000000001000110;
    A60= 16'b0000000000111100;
    A50= 16'b0000000000110010;
    A40= 16'b0000000000101000;
    A30= 16'b0000000000011110;
    A20= 16'b0000000000010100;
    A10= 16'b0000000000001010;



begin//*****************************************************************************
**********************计算开始



//*****************************************************************************
*处理万位
            begin

            CMP60000 = (delybinary>=A60000) ? 1'b1:1'b0;
            CMP50000 = (delybinary>=A50000) ? 1'b1:1'b0;
            CMP40000 = (delybinary>=A40000) ? 1'b1:1'b0;
            CMP30000 = (delybinary>=A30000) ? 1'b1:1'b0;
            CMP20000 = (delybinary>=A20000) ? 1'b1:1'b0;
            CMP10000 = (delybinary>=A10000) ? 1'b1:1'b0;


            if(CMP60000==1)
                begin
                    TenThousands=4'd6;
                    binarythousands=delybinary-A60000;
                end
            else if((CMP60000==0)&&(CMP50000==1))
                begin
```

```
                TenThousands=4'd5;
                binarythousands=delybinary-A50000;
            end
        else if((CMP50000==0)&&(CMP40000==1))
            begin
                TenThousands=4'd4;
                binarythousands=delybinary-A40000;
            end
        else if((CMP40000==0)&&(CMP30000==1))
            begin
                TenThousands=4'd3;
                binarythousands=delybinary-A30000;
            end
        else if((CMP30000==0)&&(CMP20000==1))
            begin
                TenThousands=4'd2;
                binarythousands=delybinary-A20000;
            end
        else if((CMP20000==0)&&(CMP10000==1))
            begin
                TenThousands=4'd1;
                binarythousands=delybinary-A10000;
            end
        else if(CMP10000==0)
            begin
                TenThousands=4'd0;
                binarythousands=delybinary;
            end
        end
//*****************************************************************处理千位
        begin

        CMP9000 = (binarythousands>=A9000) ? 1'b1:1'b0;
        CMP8000 = (binarythousands>=A8000) ? 1'b1:1'b0;
        CMP7000 = (binarythousands>=A7000) ? 1'b1:1'b0;
        CMP6000 = (binarythousands>=A6000) ? 1'b1:1'b0;
        CMP5000 = (binarythousands>=A5000) ? 1'b1:1'b0;
        CMP4000 = (binarythousands>=A4000) ? 1'b1:1'b0;
        CMP3000 = (binarythousands>=A3000) ? 1'b1:1'b0;
        CMP2000 = (binarythousands>=A2000) ? 1'b1:1'b0;
        CMP1000 = (binarythousands>=A1000) ? 1'b1:1'b0;

        if(CMP9000==1)
            begin
```

```
            Thousands=4'd9;
            binaryhudreds=binarythousands-A9000;
        end
    else    if(CMP9000==0&&CMP8000==1)
        begin
            Thousands=4'd8;
            binaryhudreds=binarythousands-A8000;
        end
    else    if(CMP8000==0&&CMP7000==1)
        begin
            Thousands=4'd7;
            binaryhudreds=binarythousands-A7000;
        end
    else    if(CMP7000==0&&CMP6000==1)
        begin
            Thousands=4'd6;
            binaryhudreds=binarythousands-A6000;
        end
    else if(CMP6000==0&&CMP5000==1)
        begin
            Thousands=4'd5;
            binaryhudreds=binarythousands-A5000;
        end
    else if(CMP5000==0&&CMP4000==1)
        begin
            Thousands=4'd4;
            binaryhudreds=binarythousands-A4000;
        end
    else if(CMP4000==0&&CMP3000==1)
        begin
            Thousands=4'd3;
            binaryhudreds=binarythousands-A3000;
        end
    else if(CMP3000==0&&CMP2000==1)
        begin
            Thousands=4'd2;
            binaryhudreds=binarythousands-A2000;
        end
    else if(CMP2000==0&&CMP1000==1)
        begin
            Thousands=4'd1;
            binaryhudreds=binarythousands-A1000;
        end
```

```
            else if(CMP1000==0)
                begin
                    Thousands=4'd0;
                    binaryhudreds=binarythousands;
                end
            end
//************************************************************处理百位
            begin

            CMP900 = (binaryhudreds>=A900) ? 1'b1:1'b0;
            CMP800 = (binaryhudreds>=A800) ? 1'b1:1'b0;
            CMP700 = (binaryhudreds>=A700) ? 1'b1:1'b0;
            CMP600 = (binaryhudreds>=A600) ? 1'b1:1'b0;
            CMP500 = (binaryhudreds>=A500) ? 1'b1:1'b0;
            CMP400 = (binaryhudreds>=A400) ? 1'b1:1'b0;
            CMP300 = (binaryhudreds>=A300) ? 1'b1:1'b0;
            CMP200 = (binaryhudreds>=A200) ? 1'b1:1'b0;
            CMP100 = (binaryhudreds>=A100) ? 1'b1:1'b0;

            if(CMP900==1)
                begin
                    Hundreds=4'd9;
                    binarytens=binaryhudreds-A900;
                end
            else if((CMP900==0)&&(CMP800==1))
                begin
                    Hundreds=4'd8;
                    binarytens=binaryhudreds-A800;
                end
            else if((CMP800==0)&&(CMP700==1))
                begin
                    Hundreds=4'd7;
                    binarytens=binaryhudreds-A700;
                end
            else if((CMP700==0)&&(CMP600==1))
                begin
                    Hundreds=4'd6;
                    binarytens=binaryhudreds-A600;
                end
            else if((CMP600==0)&&(CMP500==1))
                begin
                    Hundreds=4'd5;
                    binarytens=binaryhudreds-A500;
                end
```

```
        else if((CMP500==0)&&(CMP400==1))
            begin
                Hundreds=4'd4;
                binarytens=binaryhudreds-A400;
            end
        else if((CMP400==0)&&(CMP300==1))
            begin
                Hundreds=4'd3;
                binarytens=binaryhudreds-A300;
            end
        else if((CMP300==0)&&(CMP200==1))
            begin
                Hundreds=4'd2;
                binarytens=binaryhudreds-A200;
            end
        else if((CMP200==0)&&(CMP100==1))
            begin
                Hundreds=4'd1;
                binarytens=binaryhudreds-A100;
            end
        else if(CMP100==0)
            begin
                Hundreds=4'd0;
                binarytens=binaryhudreds;
            end
        end
//************************************************************处理十位
        begin

        CMP90 = (binarytens>=A90) ? 1'b1:1'b0;
        CMP80 = (binarytens>=A80) ? 1'b1:1'b0;
        CMP70 = (binarytens>=A70) ? 1'b1:1'b0;
        CMP60 = (binarytens>=A60) ? 1'b1:1'b0;
        CMP50 = (binarytens>=A50) ? 1'b1:1'b0;
        CMP40 = (binarytens>=A40) ? 1'b1:1'b0;
        CMP30 = (binarytens>=A30) ? 1'b1:1'b0;
        CMP20 = (binarytens>=A20) ? 1'b1:1'b0;
        CMP10 = (binarytens>=A10) ? 1'b1:1'b0;


        if((CMP90==1))
            begin
                Tens=4'd9;
                binaryones=binarytens-A90;
```

```
    end
else if((CMP90==0)&&(CMP80==1))
    begin
        Tens=4'd8;
        binaryones=binarytens-A80;
    end
else if((CMP80==0)&&(CMP70==1))
    begin
        Tens=4'd7;
        binaryones=binarytens-A70;
    end
else if((CMP70==0)&&(CMP60==1))
    begin
        Tens=4'd6;
        binaryones=binarytens-A60;
     end
else if((CMP60==0)&&(CMP50==1))
    begin
        Tens=4'd5;
        binaryones=binarytens-A50;
    end
else if((CMP50==0)&&(CMP40==1))
    begin
        Tens=4'd4;
        binaryones=binarytens-A40;
    end
else if((CMP40==0)&&(CMP30==1))
    begin
        Tens=4'd3;
        binaryones=binarytens-A30;
    end
else if((CMP30==0)&&(CMP20==1))
    begin
        Tens=4'd2;
        binaryones=binarytens-A20;
     end
else if((CMP20==0)&&(CMP10==1))
    begin
        Tens=4'd1;
        binaryones=binarytens-A10;
    end
else if(CMP10==0)
    begin
        Tens=4'd0;
```

```
                    binaryones=binarytens;
                end
            end
//*************************************************************************处理个位
                Ones={binaryones[3],binaryones[2],binaryones[1],binaryones[0]};

        end
//*****************************************************************************************计算结束
    back={TenThousands,Thousands,Hundreds,Tens,Ones};
end
endfunction



always@(posedge CLK or posedge PRE)
begin
if(PRE) delybinary <= 16'b1111111111111111;
else if(CE) delybinary <= binary;
end

always@(*)
begin
  CMP = (delybinary==binary) ? 1'b1:1'b0;
  if (CMP) result = back (delybinary) ;
  else result = 0;
end
Endmodule
```

多次调用函数可简化代码

函数版本如下：

```
module BCD_Compare_Version(
    input [15:0] binary,
    input CLK,
    input CE,
    input PRE,
```

```
        output reg [19:0] result
);
reg [19:0] jiezhi4;
reg [19:0] jiezhi3;
reg [19:0] jiezhi2;
reg [19:0] jiezhi1;

reg [15:0] delybinary;   //寄存器输出
reg CMP;

reg [19:0] binarythousands;     //最高位是 X 位的输入
reg [19:0] binaryhudreds;
reg [19:0] binarytens;
reg [3:0] TenThousands;   //BCD 的每一位
reg [3:0] Thousands;
reg [3:0] Hundreds;
reg [3:0] Tens;
reg [3:0] Ones;

reg [15:0] A60000;       //比较参考   被减数
reg [15:0] A50000;
reg [15:0] A40000;
reg [15:0] A30000;
reg [15:0] A20000;
reg [15:0] A10000;
reg [15:0] A9000;
reg [15:0] A8000;
reg [15:0] A7000;
reg [15:0] A6000;
reg [15:0] A5000;
reg [15:0] A4000;
reg [15:0] A3000;
reg [15:0] A2000;
reg [15:0] A1000;
reg [15:0] A900;
reg [15:0] A800;
reg [15:0] A700;
reg [15:0] A600;
reg [15:0] A500;
reg [15:0] A400;
reg [15:0] A300;
reg [15:0] A200;
reg [15:0] A100;
reg [15:0] A90;
```

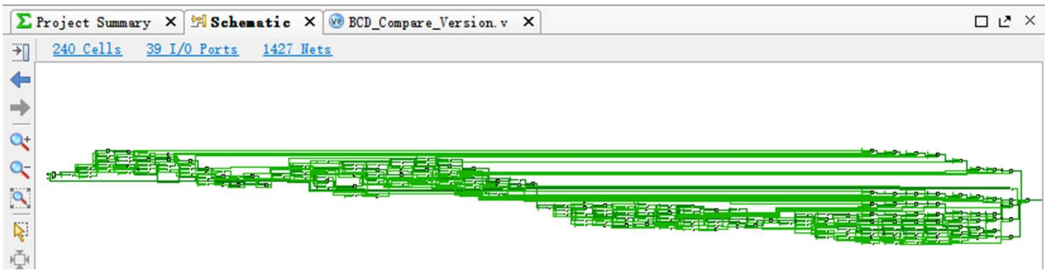
```
reg [15:0] A80;
reg [15:0] A70;
reg [15:0] A60;
reg [15:0] A50;
reg [15:0] A40;
reg [15:0] A30;
reg [15:0] A20;
reg [15:0] A10;

function [19:0] back;    //**********************************************back  函数输出
input [15:0] in;    //函数输入
input reg [15:0] A9;
input reg [15:0] A8;
input reg [15:0] A7;
input reg [15:0] A6;
input reg [15:0] A5;
input reg [15:0] A4;
input reg [15:0] A3;
input reg [15:0] A2;
input reg [15:0] A1;

reg [9:1] CMP;
reg [3:0] result;
reg [15:0] residue;

begin

        CMP[9] = (in>=A9) ? 1'b1:1'b0;
        CMP[8] = (in>=A8) ? 1'b1:1'b0;
        CMP[7] = (in>=A7) ? 1'b1:1'b0;
        CMP[6] = (in>=A6) ? 1'b1:1'b0;
        CMP[5] = (in>=A5) ? 1'b1:1'b0;
        CMP[4] = (in>=A4) ? 1'b1:1'b0;
        CMP[3] = (in>=A3) ? 1'b1:1'b0;
        CMP[2] = (in>=A2) ? 1'b1:1'b0;
        CMP[1] = (in>=A1) ? 1'b1:1'b0;

        if(CMP[9]==1)
            begin
                result=4'd9;
                residue=in-A9;
            end
        else    if(CMP[9]==0&&CMP[8]==1)
            begin
```

```
            result=4'd8;
            residue=in-A8;
        end
    else    if(CMP[8]==0&&CMP[7]==1)
        begin
        result=4'd7;
        residue=in-A7;
        end
    else    if(CMP[7]==0&&CMP[6]==1)
        begin
        result=4'd6;
        residue=in-A6;
        end
    else if(CMP[6]==0&&CMP[5]==1)
        begin
            result=4'd5;
            residue=in-A5;
        end
    else if(CMP[5]==0&&CMP[4]==1)
        begin
            result=4'd4;
            residue=in-A4;
        end
    else if(CMP[4]==0&&CMP[3]==1)
        begin
            result=4'd3;
            residue=in-A3;
        end
    else if(CMP[3]==0&&CMP[2]==1)
        begin
            result=4'd2;
            residue=in-A2;
        end
    else if(CMP[2]==0&&CMP[1]==1)
        begin
            result=4'd1;
            residue=in-A1;
        end
    else if(CMP[1]==0)
        begin
            result=4'd0;
            residue=in;
        end
    back={result,residue};
```

```
end
endfunction


always@(posedge CLK or posedge PRE)
begin
if(PRE) delybinary <= 16'b1111111111111111;
else if(CE) delybinary <= binary;
end

always@(*)
begin
CMP = (delybinary==binary) ? 1'b1:1'b0;
if (CMP)
    begin
    A60000= 16'b1110101001100000;                    //参数赋值
    A50000= 16'b1100001101010000;
    A40000= 16'b1001110001000000;
    A30000= 16'b0111010100110000;
    A20000= 16'b0100111000100000;
    A10000= 16'b0010011100010000;

    A9000= 16'b0010001100101000;
    A8000= 16'b0001111101000000;
    A7000= 16'b0001101101011000;
    A6000= 16'b0001011101110000;
    A5000= 16'b0001001110001000;
    A4000= 16'b0000111110100000;
    A3000= 16'b0000101110111000;
    A2000= 16'b0000011111010000;
    A1000= 16'b0000001111101000;

    A900= 16'b0000001110000100;
    A800= 16'b0000001100100000;
    A700= 16'b0000001010111100;
    A600= 16'b0000001001011000;
    A500= 16'b0000000111110100;
    A400= 16'b0000000110010000;
    A300= 16'b0000000100101100;
    A200= 16'b0000000011001000;
    A100= 16'b0000000001100100;

    A90= 16'b0000000001011010;
    A80= 16'b0000000001010000;
```

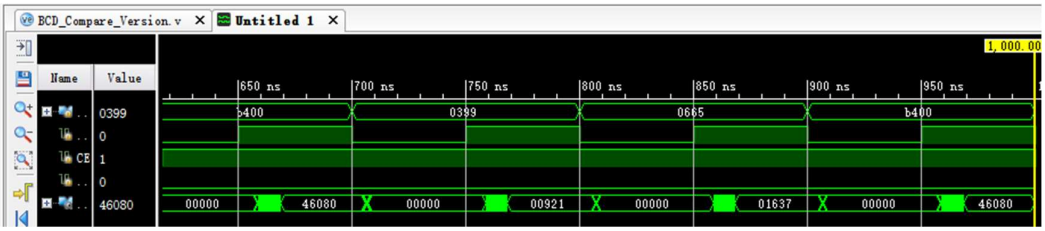
```
    A70= 16'b0000000001000110;
    A60= 16'b0000000000111100;
    A50= 16'b0000000000110010;
    A40= 16'b0000000000101000;
    A30= 16'b0000000000011110;
    A20= 16'b0000000000010100;
    A10= 16'b0000000000001010;


jiezhi4=back(delybinary,A60000,A60000,A60000,A60000,A50000,A40000,A30000,A20000,A1000
0);
    TenThousands={jiezhi4[19],jiezhi4[18],jiezhi4[17],jiezhi4[16]};
    binarythousands=jiezhi4;


jiezhi3=back(binarythousands,A9000,A8000,A7000,A6000,A5000,A4000,A3000,A2000,A1000);
    Thousands={jiezhi3[19],jiezhi3[18],jiezhi3[17],jiezhi3[16]};
    binaryhudreds=jiezhi3;

    jiezhi2=back(binaryhudreds,A900,A800,A700,A600,A500,A400,A300,A200,A100);
    Hundreds={jiezhi2[19],jiezhi2[18],jiezhi2[17],jiezhi2[16]};
    binarytens=jiezhi2;

    jiezhi1=back(binarytens,A90,A80,A70,A60,A50,A40,A30,A20,A10);
    Tens={jiezhi1[19],jiezhi1[18],jiezhi1[17],jiezhi1[16]};
    Ones={jiezhi1[3],jiezhi1[2],jiezhi1[1],jiezhi1[0]};

    result={TenThousands,Thousands,Hundreds,Tens,Ones};
    end
else result = 0;
end
Endmodule
```

比较法电路结构如下：



由此可知函数调用只是简化代码不能简化电路结构。

其时序仿真如下：



3、 类二进制除法器的多次调用

由比较法可知其代码十分繁琐，那能不能用简单的代码来实现呢？而且还不能用除法和取余。

基本思想：二进制数左移一位则乘二，右移一位则除二。
提取千位时，一千左移便是 1，2，4，8 千。
09219 除以一千结果为 9 即是一千加八千 1001。
比较 09219 和 1000 左移四位（8000），09219>8000 则 CMP3=1，09219-8000=1219。
1219<(4000) 则 CMP2=0，1219-0=129。类推最后 CMP1=0，CMP0=1。余数 219
所以 CMP[4,3,2,1]=1001 即为千位的 BCD 码，最后剩下的 219 便是余数。这也是二进制除法器的基本思路。
219 去下一级，处理百位即可。

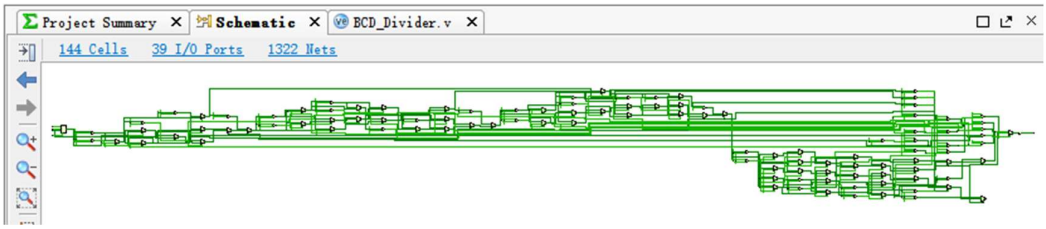
代码如下：

```
module BCD_Divider(
    input [15:0] binary,
    input CLK,
    input CE,
    input PRE,
    output reg [19:0] result
);
    reg [19:0] jiezhi4;
    reg [19:0] jiezhi3;
    reg [19:0] jiezhi2;
    reg [19:0] jiezhi1;
    reg [15:0] YUSHU4;
    reg [15:0] YUSHU3;
    reg [15:0] YUSHU2;
    reg [15:0] YUSHU1;
    reg [15:0] delybinary;
    reg CMP;

    reg [3:0] TenThousands;
    reg [3:0] Thousands;
```

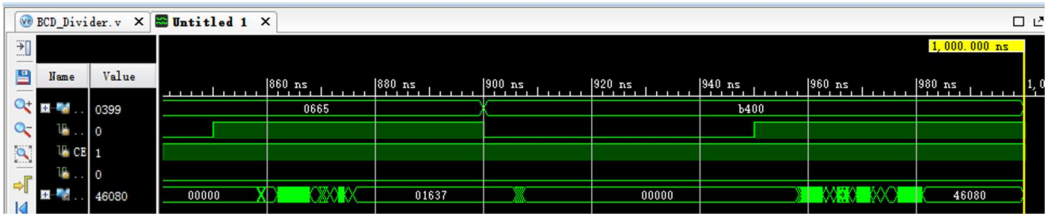
```
    reg [3:0] Hundreds;
    reg [3:0] Tens;
    reg [3:0] Ones;
    reg [15:0] cw;
    reg [15:0] cq;
    reg [15:0] cb;
    reg [15:0] cs;

function [19:0] back;    //back  函数输出
input [15:0] beichushu;
input [15:0] chushu;

    reg [19:0] beichushuK;
    reg [19:0] chushuK;
    reg [15:0] yushu;
    reg [3:0] shang;
    reg [3:0] fuzhu;
    reg CMP1;
integer i;
begin
        shang=4'b0000;
        fuzhu=4'b0000;
        beichushuK={fuzhu,beichushu};    //被除数扩展
        chushuK={chushu,fuzhu};//除数扩展      低位补零相当于左移四位

        for(i=4;i>0;i=i-1)
        begin
            chushuK=chushuK >> 1;
            CMP1 = (beichushuK >= chushuK) ? 1'b1:1'b0;
            if(CMP1==1)
               begin
                  yushu=beichushuK-chushuK;
                  beichushuK=yushu;    //下一轮被除数是现在的余数
               end
            else if(CMP1==0)
               begin
                  yushu=beichushuK;
//                    beichushuK=yushuK;  下一轮被除数是现在的余数  没变
               end
                  shang=shang << 1;//得出商
                  shang[0]=CMP1;
     end
       back={shang,yushu};
end
```

```
endfunction


always@(posedge CLK or posedge PRE)
   begin
      if(PRE) delybinary <= 16'b1111111111111111;
      else if(CE) delybinary <= binary;
   end

always@(*)
begin
   cw=16'b0010011100010000;
   cq=16'b0000001111101000;
   cb=16'b0000000001100100;
   cs=16'b0000000000001010;
   CMP = (delybinary==binary) ? 1'b1:1'b0;
    if (CMP)
        begin
        jiezhi4=back(delybinary,cw);
        TenThousands={jiezhi4[19],jiezhi4[18],jiezhi4[17],jiezhi4[16]};
        YUSHU4=back(delybinary,cw);

        jiezhi3=back(YUSHU4,cq);
        Thousands={jiezhi3[19],jiezhi3[18],jiezhi3[17],jiezhi3[16]};
        YUSHU3=back(YUSHU4,cq);

        jiezhi2=back(YUSHU3,cb);
        Hundreds={jiezhi2[19],jiezhi2[18],jiezhi2[17],jiezhi2[16]};
        YUSHU2=back(YUSHU3,cb);

        jiezhi1=back(YUSHU2,cs);
        Tens={jiezhi1[19],jiezhi1[18],jiezhi1[17],jiezhi1[16]};
        YUSHU1=back(YUSHU2,cs);

        Ones={YUSHU1[3],YUSHU1[2],YUSHU1[1],YUSHU1[0]};

        result = {TenThousands,Thousands,Hundreds,Tens,Ones};
        end
     else result = 0;
   end
Endmodule
```

电路结构如下：



时序仿真结果如下：



4、 移位相加法

基本思路：找二进制数和 BCD 码的对应关系

1.将二进制码左移一位（或者乘 2）

2.找到左移后的码所对应的个，十，百位。

3.判断在个位和百位的码是否大于 5，如果是则该段码加 3。

4.继续重复以上三步直到移位 8 次后停止。

如

下面是一个例子，将 1111_1111 转换为 BCD码，如果8bit数据最终移位得到18bit 数据，那么个位，十位，百位分别对应12~9，16~13，18~17位。

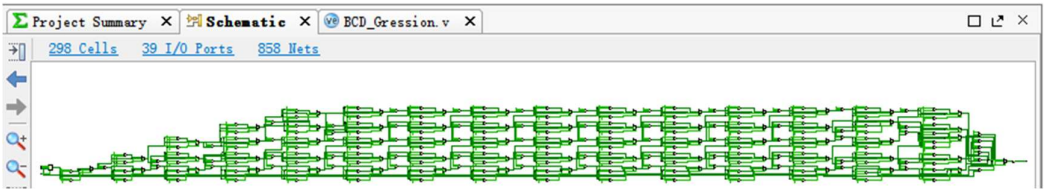
Operation	Hundreds	Tens	Units	Binary	
HEX				F	F
Start				1 1 1 1	1 1 1 1
Shift 1			1	1 1 1 1	1 1 1
Shift 2			1 1	1 1 1 1	1 1
Shift 3			1 1 1	1 1 1 1	1
Add 3			1 0 1 0	1 1 1 1	1
Shift 4		1	0 1 0 1	1 1 1 1	
Add 3		1	1 0 0 0	1 1 1 1	
Shift 5		1 1	0 0 0 1	1 1 1	
Shift 6		1 1 0	0 0 1 1	1 1	
Add 3		1 0 0 1	0 0 1 1	1 1	
Shift 7	1	0 0 1 0	0 1 1 1	1	
Add 3	1	0 0 1 0	1 0 1 0	1	
Shift 8	1 0	0 1 0 1	0 1 0 1		
BCD	2	5	5		

扩展为 16 位即可：

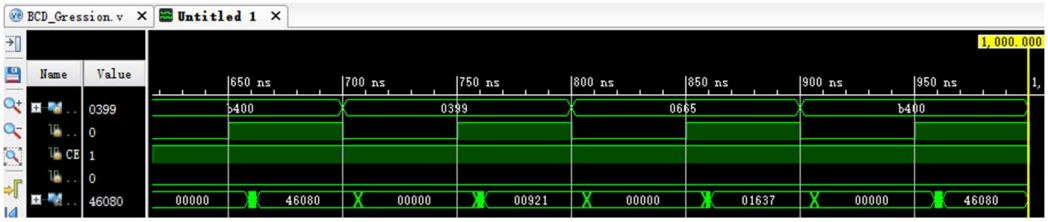
```
module BCD_Gression(
    input [15:0] binary,
    input CLK,
    input CE,
    input PRE,
    output reg [19:0] result
);

  reg [15:0] delybinary;
  reg CMP;

function [19:0] back;   //back 函数输出
  input [15:0] delybinary;
  reg [3:0] TenThousands;
  reg [3:0] Thousands;
  reg [3:0] Hundreds;
  reg [3:0] Tens;
  reg [3:0] Ones;
  integer i;
  begin
    back=20'd0;
    TenThousands=4'd0;
    Thousands=4'd0;
    Hundreds=4'd0;
    Tens=4'd0;
    Ones=4'd0;
    for(i=15;i>=0;i=i-1)
      begin
        if(TenThousands>=5)
          TenThousands = TenThousands+3;
        if(Thousands>=5)
          Thousands = Thousands+3;
        if(Hundreds>=5)
          Hundreds = Hundreds+3;
        if(Tens>=5)
          Tens=Tens+3;
        if(Ones>=5)
          Ones=Ones+3;
        TenThousands = TenThousands << 1;
        TenThousands[0] = Thousands[3];
        Thousands = Thousands << 1;
        Thousands[0] = Hundreds[3];
        Hundreds = Hundreds << 1;
```

```
            Hundreds[0] = Tens[3];
            Tens=Tens << 1;
            Tens[0] = Ones[3];
            Ones = Ones << 1;
            Ones[0] = delybinary[i];
          end
      back={TenThousands,Thousands,Hundreds,Tens,Ones};
   end
endfunction


always@(posedge CLK or posedge PRE)
   begin
      if(PRE) delybinary <= 16'b1111111111111111;
      else if(CE) delybinary <= binary;
   end

always@(*)
   begin
      CMP = (delybinary==binary) ? 1'b1:1'b0;
      if (CMP) result = back (delybinary) ;
      else result = 0;
   end
Endmodule
```

电路结构：



时序仿真：



二、仿真源文件

```
module text;
```

```
reg [15:0] binary;
reg CLK;
reg CE;
reg PRE;
wire [19:0] result;

BCD_Gression uut(.binary(binary),.result(result),.CLK(CLK),.CE(CE),.PRE(PRE));
begin
   always
    begin
       assign binary=16'b1011010000000000;
       assign CE=1;
       assign PRE=0;
       assign CLK=0;
       #50;
       assign binary=16'b1011010000000000;
       assign CE=1;
       assign PRE=0;
       assign CLK=1;
       #50;
       assign binary=16'b0000001110011001;
       assign CE=1;
       assign PRE=0;
       assign CLK=0;
       #50;
       assign binary=16'b0000001110011001;
       assign CE=1;
       assign PRE=0;
       assign CLK=1;
       #50;
       assign binary=16'b0000011001100101;
       assign CE=1;
       assign PRE=0;
       assign CLK=0;
       #50;
       assign binary=16'b0000011001100101;
       assign CE=1;
       assign PRE=0;
       assign CLK=1;
       #50;
       $display("The result is %h",result);
    end
end
endmodule
```