Research Project

16 位无符号整数除法器设计报告

GROUP 03

胡佳妮

陶冠宏

王子思

Date: 2012-1-14

一、 背景介绍

在 Verilog HDL 语言中虽然有除法的运算指令，但是除法运算中的除数必须是 2 的整数次幂，因此无法实现除数为任意整数的除法，很大程度上限制了它的使用领域。并且多数综合工具对于除运算指令不能综合出令人满意的结果，有些甚至不能给予综合。本次课程设计，我们小组的任务就是在以前 LCDF 实验的基础上，在 xilinx Spartan-3 实验平台上设计一个 16 位无符号证整数除法器，被除数和除数可以是任意的 16 位无符号整数，并且得到商和余数。并当除数为零的时候，提示出错信息。

二、 设计说明

A. 设计开发环境

实验平台：xilinx Spartan-3

开发环境：Xilinx ISE

硬件描述语言：Verilog HDL

B. 输入输出交互选择

输入：Spartan-3 实验平台上的按钮、拨动开关输入

输出：调用四位 7 段数码管显示模块输出

C. 核心模块设计——除法器

对于无符号的整数除法器，我们想到最简单的方法是多次相减，直至被除数小于除数。减的次数就是所得商，剩下的不够减的被除数就是余数。

这种方式虽然简单，但是效率不高。所以我们采用一种更为高效的方法——移位相减。移位相减的思想是从被除数 A 的最高位开始，每次将一位数移入寄存器，然后与除数 B 比较，如果够减，商的对应位是 1；如果不够减，商的对应位就是 0。

但是实现这种方式，我们需要每次把被除数的一位移入寄存器，操作比较麻烦，因此我们采用移位相减的思想，但用另一种较为巧妙而方便的方法实现。

除法器的主要实现步骤如下：

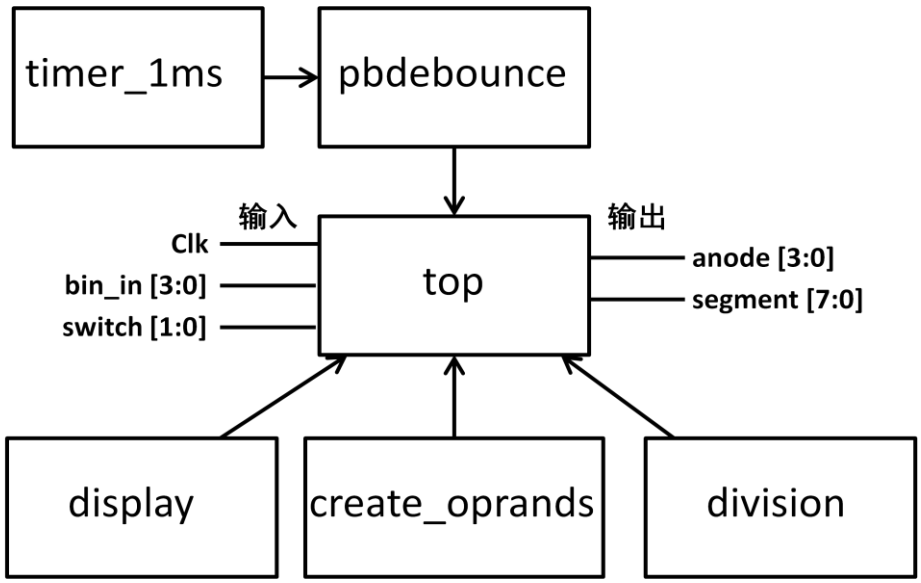
1) 对于 16 位无符号被除数 a，先将 a 转换成高 16 位全是 0、低 16 位是 a 的 32 位数 temp_a。
2) 对于 16 位无符号除数 b，将 b 转换成高 16 位是 b、低 16 位全是 0 的 32 位数 temp_b。
3) 在每个时钟周期开始时 temp_a 向左移动一位，最后一位补零，然后判断 temp_a 是否大于等于 temp_b，如是则 temp_a 减去 temp_b 并且 temp_a 的值加 1，得到的新值仍赋给 temp_a；如不是则直接进入下一步。
4) 循环上一步骤 16 次。

上面的移位、比较、减法（减法视情况而定）要进行 16 次，经过 16 个周期后，运算结束，所得到的 temp_a 的高 16 位为余数，低 16 位为商。将移位、比较和相减放在同一个循环中，去除了不必要的延时，增加了设计的可靠性。

另外，还要在模块中加一个 erro 变量，检测除数是否为零。一旦输入除数为零，则变量 erro 赋为 1，在数码管上显示“ErrO”字样。

三、 模块结构

A. 各模块关系图



图表 1 各模块关系结构图

B. Top——主程序模块

输入：

Clk：时钟脉冲

bin_in [3:0]：四个按键输入信息

switch [1:0]：两个拨动开关输入信息

输出：

anode [3:0]: 四位数码管的选择

segment [7:0]：七段数码管显示信息输出

主要功能：主程序，调用各模块命令，并控制最外层的输入输出。

图表 2 top 模块输入输出端口示意图

C. display——数码管的扫描显示和显示译码模块

输入：

Clk：时钟脉冲

ditgit [15:0]：需要显示的 16 位无符号整数

erro：出错提示信息，提示当前是否需要显示“erro”字样

输出：

anode [3:0]: 四位数码管的选择

segment [7:0]：七段数码管显示信息输出

主要功能：利用视觉残留原理，用时分复用的方法对四位数码管进行显示控制，使四位数码管同时显示要求显示的数字。

图表 3 display 模块输入输出端口示意图

D. pbdebounce——按键防抖动模块

输入：

Clk：时钟脉冲

button：按键输入信息

输出：

pbreg：判断是否是将按键信息送入电路

主要功能：防止因电路中按键按下或放开时由于机械接触性，接触点产生机械震动而导致按键对应的电信号产生瞬间脉冲串。也就是去抖动。

图表 4 pddebounce 模块输入输出端口示意图

E. timer_1ms——1ms 计时模块

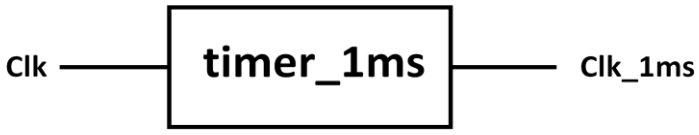
输入：

clk：内部时钟脉冲

输出：

clk_1ms：1ms 计时脉冲

主要功能：产生周期为 1ms 的计时脉冲



图表 5 timer_1ms 模块输入输出端口示意图

F. create_oprands——操作数输入模块

输入：

switch [1:0]：两个拨动开关输入信息，选择被除数（操作数 1）或除数（操作数 2）

btn [3:0]：四个按键输入信息，选择对操作数的某一位增加 1

输出：

op1 [15:0]：16 位无符号被除数（操作数 1）的输出

op2 [15:0]：16 位无符号除数（操作数 2）的输出

主要功能：产生操作数，利用拨动开关和按键对被除数和除数进行修改

图表 6 create_oprands 模块输入输出端口示意图

G. division——除法器模块

输入：

Clk：时钟

switch [1:0]：两个拨动开关输入信息，提示当前显示内容

dividend [15:0]：被除数

divisor [15:0]：除数

输出：

quotient [15:0]：商

remainder [15:0]：余数

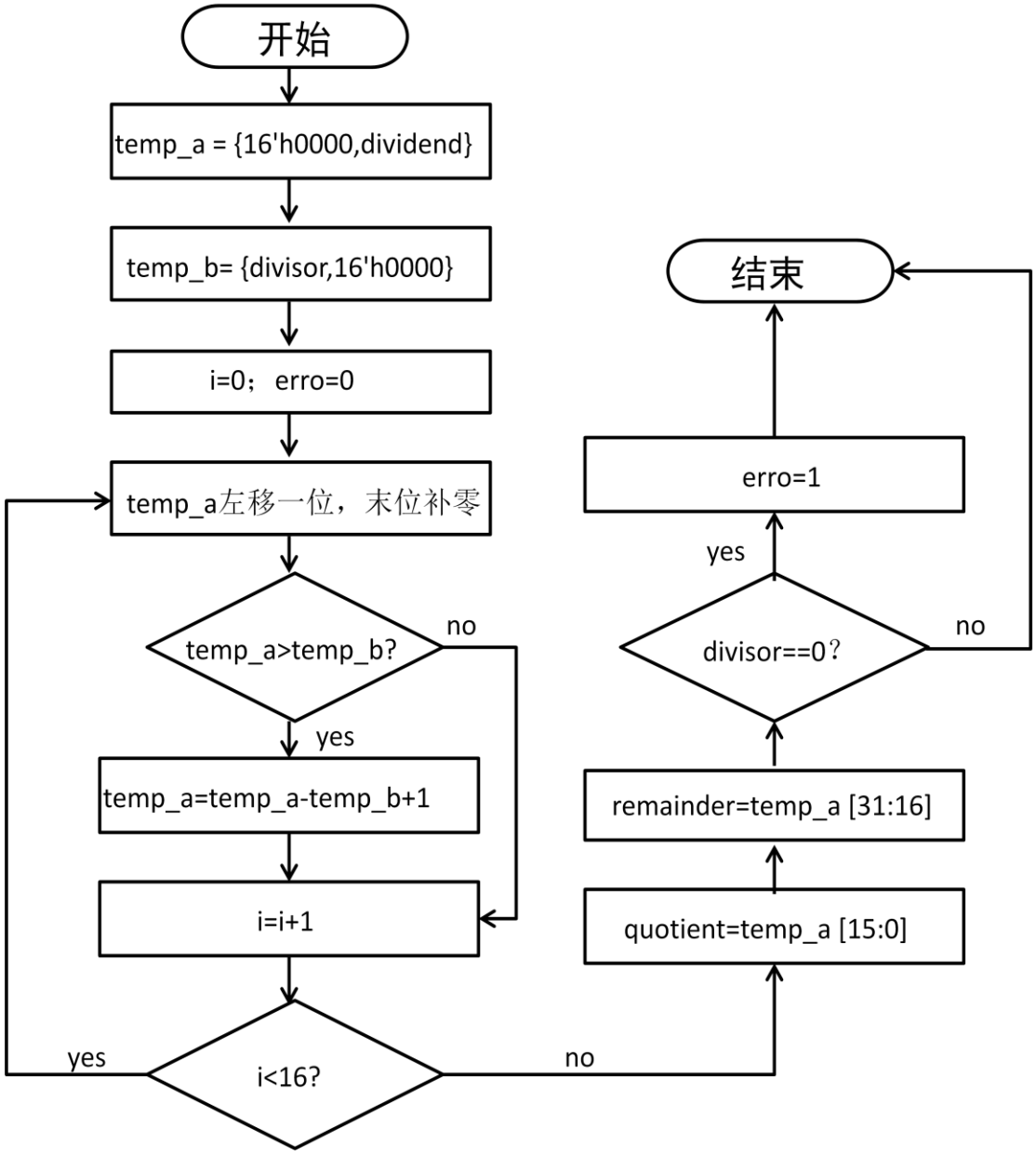
erro：错误提示信息（当除数为 0 时，erro=1；否则 erro=0）

主要功能：根据输入的被除数和除数，进行除法运算，并得到商和余数。同时根据除数是否为零，而判断是否需要提示出错信息。

图表 7 division 模块输入输出端口示意图

四、 程序流程

核心模块流程图——division：



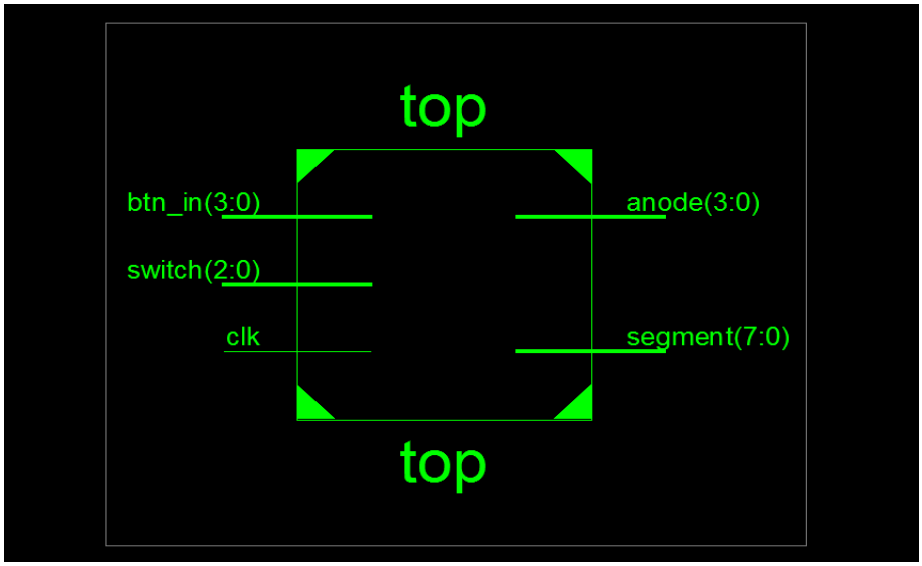
说明：由于其余模块都与 LCDF 实验课上内容大致相同，所以略去这些模块的流程图，不再赘述。

五、 调试过程分析

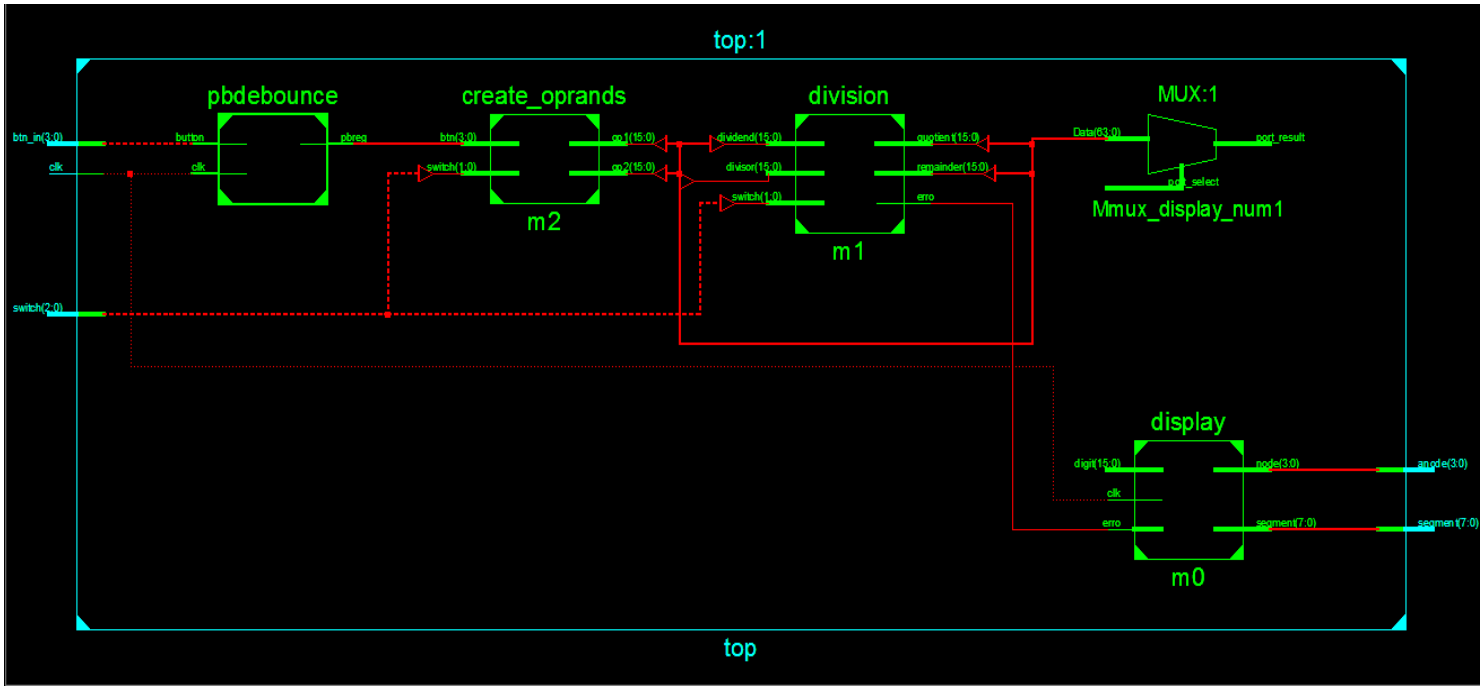
A. 综合分析

电路模块代码设计完成后，进入调试阶段。首先是语法调试，这部分与 C 语言相同，错误定位非常准确。

完成语法调试后，运行顶层代码的 RLT 综合，分析 RLT 综合电路，综合结果是否特别之处，与设计功能能否一致。分析、浏览系统综合报告，搞清系统资源使用情况，检查各模块的错误信息（包括警告信息），并一一排除。



图表 8 主程序外部 RLT 综合电路示意图



图表 9 主程序内部主要模块 RLT 综合电路示意图

B. 调试

调试顺序按时间定标模块、秒信号模块、显示模块、按键模块、除法器模块、顶层模块逐一调试。根据 ISE 软件的错误提示，对其中出现的变量定义错误、语句错误等进行修改。编译通过后，编写 UCF 文件，对引脚进行定义。完成后，将程序模块下载到 Spartan III 实验板上进行测试。

引脚功能定义如下：

类型	引脚	取值	含义
输入	sw[1:0]	01	• 显示操作数 A[15:0] • 允许 btn[3:0]修改被除数（操作数 A）
		10	• 显示操作数 B[15:0] • 允许 btn[3:0]修改除数（操作数 B）
	btn[0]	上升沿	当 sw[1:0]=01/10 时，对 anode[0]增加 1
	btn[1]		当 sw[1:0]=01/10 时，对 anode[1]增加 1
	btn[2]		当 sw[1:0]=01/10 时，对 anode[2]增加 1
	btn[3]		当 sw[1:0]=01/10 时，对 anode[3]增加 1
输出	sw[1:0]	00	在 4 个七段数码管显示 16 位商的结果
		11	在 4 个七段数码管显示 16 位余数的结果

C. 操作过程

程序成功下载到板子上之后，可以对板上的按键和开关进行具体的操作：

1) 拨动开关，使 sw[1:0]取值 01，此时在 4 位数码管上显示操作数 A 的值，通过按下四个数位对应按钮，对操作数 A 的各位数据进行更改
2) 拨动开关，使 sw[1:0]取值 10，此时在 4 位数码管上显示操作数 B 的值，通过按下四个数位对应按钮，对操作数 B 的各位数据进行更改
3) 确定操作数 A、B，即输入被除数和除数之后，拨动开关，使 sw[1:0]取值 00，此时数码管上显示 16 位商的结果
4) 拨动开关，使 sw[1:0]取值 11 时，此时数码管上显示 16 位余数的结果

D. 测试数据

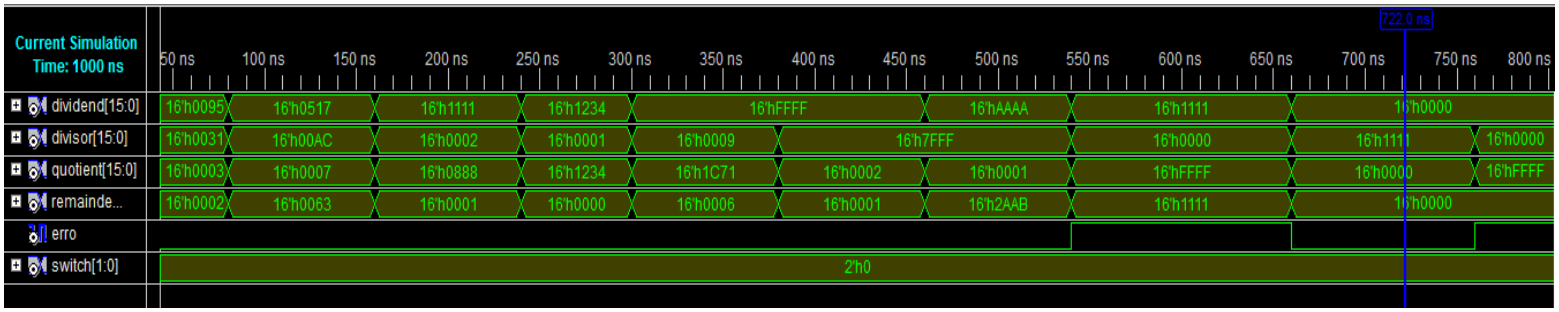
为了检验除法器的正确性，我们一共设计了 10 组数据，下面是 10 组测试数据的输入和输出及其功能：

1	输入	被除数 0095，除数 0031
	输出	商 0003，余数 0002
	功能	由于 Verilog HDL 语言中有除的运算指令，但除数必须是 2 的幂次，因此用被除数和除数都为非 2 的幂次的任意整数进行测试，验证是否实现要求的运算，结果正确
2	输入	被除数 0571，除数 00AC
	输出	商 0008，余数 0011
	功能	与上一组测试数据功能相同，再次用任意整数进行测试，以验证其正确性
3	输入	被除数 1111，除数 0002
	输出	商 0888，余数 0001
	功能	当被除数和除数都为 2 的幂次时，验证 Verilog HDL 语言中除运算指令能够实现的功能，结果正确
4	输入	被除数 1234，除数 0001
	输出	商 1234，余数 0000
	功能	当被除数为任意整数，而除数都为 2 的幂次时，验证普通数与 2 的幂次数的除法运算，结果正确
5	输入	被除数 ffff，除数 0009
	输出	商 1c71，余数 0006
	功能	被除数取最大的 16 位数，与普通数进行除法运算，验证最大被除数的边界情况，结果正确
6	输入	被除数 ffff，除数 ffff
	输出	商 0001，余数 0000
	功能	被除数和除数都为最大的 16 位数，验证最大被除数和最大除数的边界情况，结果正确
7	输入	被除数 aaaa，除数 ffff
	输出	商 0000，余数 aaaa
	功能	除数取最大的 16 位数，被除数为普通数，验证最大除数的边界情况，结果正确
8	输入	被除数 1111，除数 0000
	输出	商 ErrO，余数 ErrO

	功能	除数为 0 时，结果需要错误提示，验证普通数与除数 0 的除法的边界情况，结果有错误提示，实现要求
9	输入	被除数 0000，除数 1111
	输出	商 0000，余数 0000
	功能	被除数为 0，除数为普通数时，运算可以进行，结果为 0，验证被除数 0 与普通数的除法的边界情况，结果正确
10	输入	被除数 0000，除数 0000
	输出	商 ErrO，余数 ErrO
	功能	被除数和除数都为 0 时，结果需有错误提示，验证 0 与 0 的除法的边界情况，结果正确

通过上述 10 组数据的测试，考虑了所有可能的边界条件及特殊情况，可以保证除法器的正确性，实现了要求的功能。

六、 核心模块模拟仿真结果



模拟波形分析：

序列 1~7 都是普通的 16 进制的除法。期中 1~5 满足一般意义上的除法：商和余数都存在且>0；而序列 6 是除数被除数相同，序列 7 是被除数小于除数。序列 8,9,10 分别验证了被除数和除数中存在 0 的情况。

例如：

0095H/0031H=0003H ••••••• 0002H

利用循环移位相减法得以实现。

第一行每次向左移并在第一行>第二行时，在第一行第 0 位标记为 1，且将 a[]向左移一位；若不够减则该第 0 位标记为 0；由此得移动 N 次，第一行高 N 位为余数，低 N 位为商。N 为该进制中的位数。

当被除数=0，除数非 0 时：商=0，余数=0；

当除数=0，被除数非 0 时：用代码显示 error。

当被除数，除数都等于 0 时，商=FFFFH,余数为 0。

Case8、10 中除数=0，所以 error 输出 0 或 1，通过 error 变量决定数码管的显示。

模拟波形结果符合除法器功能。

七、 程序代码

```
module top(input wire clk, input wire [3:0]btn_in, input wire
[2:0]switch, output wire [3:0]anode, output wire [7:0]segment);
//top 主程序
//变量定义
wire [15:0] dividend,divisor;
reg [15:0] display_num;
wire [3:0] btn_out;
wire [15:0] quotient,remainder;
wire [15:0]  disp_counter;
wire erro;

display m0(clk,display_num,erro,anode, segment);      //显示 module
division m1 (switch[1:0],dividend,divisor,quotient,remainder,erro);
create_oprands m2(switch[1:0], btn_out[3:0], dividend, divisor); //产
生操作数
//按键去抖动程序
pbdebounce p0(clk,btn_in[0],btn_out[0]);
pbdebounce p1(clk,btn_in[1],btn_out[1]);
pbdebounce p2(clk,btn_in[2],btn_out[2]);
pbdebounce p3(clk,btn_in[3],btn_out[3]);
always @* begin
    case (switch[1:0])
        2'b01: display_num = dividend;  //显示操作数 1，即被除数
        2'b10: display_num = divisor; //显示操作数 2，即除数
        2'b00: display_num = quotient; //显示商
         2'b11: display_num = remainder; //显示余数
    endcase
  end

endmodule
/*----------------------------------------------------------------*/
module display(                  //四位七段数码管的扫描显示和显示译码模块
```

```
input wire clk,
input wire [15:0] digit,        //显示的数据
input wire erro,
output reg [ 3:0] node,         //4 个数码管的位选
output reg [ 7:0] segment);     //七段+小数点
reg [4:0] code = 5'b0;
              //code 的值分别代表 16 进制的 0-9,A,b,C,d,E,F，还有字母 E、r、O
reg [15:0] count = 15'b0;       //count 是扫描显示的计数器
always @(posedge clk) begin
    case (count[15:14])
       2'b00 : begin           //显示第 1 位数码管
                  node <= 4'b1110;
                  if(erro==0) code <= {0,digit[3:0]};
                            //如果除数不为零，则显示对应的数值
                  else code<=5'b10011;//如果除数为零，则显示 ErrO 字样
                end
       2'b01 : begin           //显示第 2 位数码管
                  node <= 4'b1101;
                  if(erro==0) code <= {0,digit[7:4]};
                  else code<=5'b10010;
                end
       2'b10 : begin           //显示第 3 位数码管
                  node <= 4'b1011;
                  if(erro==0) code <= {0,digit[11:8]};
                  else code<=5'b10010;
                end
       2'b11 : begin          //显示第 4 位数码管
                  node <= 4'b0111;
                  if(erro==0) code <= {0,digit[15:12]};
              else code<=5'b10001;
                 end
    endcase
    case (code)
//根据 code 值，在数码管上显示出相应的值，segment 的值分别对应亮暗
           5'b00000: segment <= 8'b11000000; //0
           5'b00001: segment <= 8'b11111001; //1
           5'b00010: segment <= 8'b10100100; //2
           5'b00011: segment <= 8'b10110000; //3
           5'b00100: segment <= 8'b10011001; //4
           5'b00101: segment <= 8'b10010010; //5
           5'b00110: segment <= 8'b10000010; //6
           5'b00111: segment <= 8'b11111000; //7
           5'b01000: segment <= 8'b10000000; //8
           5'b01001: segment <= 8'b10010000; //9
```

```
            5'b01010: segment <= 8'b10001000; //A
            5'b01011: segment <= 8'b10000011; //b
            5'b01100: segment <= 8'b11000110; //C
            5'b01101: segment <= 8'b10100001; //d
            5'b01110: segment <= 8'b10000110; //E
            5'b01111: segment <= 8'b10001110; //F
            5'b10001: segment <= 8'b10000110; //E
            5'b10010: segment <= 8'b11001110; //r
            5'b10011: segment <= 8'b11000000; //O
            default:  segment <= 8'b00000000; //八段全亮
    endcase
    count <= count + 1;
end
endmodule
/*-------------------------------------------------------------------------*/
module pbdebounce //按键去抖动模块
(input wire clk,
input wire button,
output reg pbreg);
reg [7:0] pbshift;
wire clk_1ms;
timer_1ms m0(clk, clk_1ms);
always@(posedge clk_1ms) begin
     pbshift=pbshift<<1; //左移 1 位
     pbshift[0]=button;
     if (pbshift==0) pbreg=0;
     if (pbshift==8'hFF) pbreg=1;
     // pbshift 八位全为 1，则认为按键已经稳定且被按下一次，在 bin_out 上输出当前
的按键输入值
end
endmodule
/*-------------------------------------------------------------------------*/
module timer_1ms (input wire clk, output reg clk_1ms); //1ms 计时模块
reg [15:0] cnt;
initial begin
    cnt [15:0] <=0;
    clk_1ms <= 0;
end
always@(posedge clk)
    if(cnt>=25000) begin   //每经过 25000 个 clk 上升沿，将 clk_1ms 反转一次
         cnt<=0;
         clk_1ms <= ~clk_1ms;
    end
    else begin
```

```
        cnt<=cnt+1;
    end
endmodule
/*-------------------------------------------------------------------*/
module create_oprands(switch,btn,op1,op2); //产  僮魇鲛 p1、op2 模块
input [1:0] switch;
input [3:0] btn;
output [15:0] op1,op2;
reg [15:0] op1,op2;
initial begin
op1=16'b0001_0001_0001_0001; //op1 的初始值为 1111h
op2=16'b0000_0000_0000_0010; //op2 的初始值为 2h
end
always @(posedge btn[0])begin //通过按键操作改变第 0 位操作数的值
    if(switch == 4'b01) op1[ 3: 0]<= op1[ 3: 0] + 4'd1;
    else if(switch == 4'b10) op2[ 3: 0]<= op2[ 3: 0] + 4'd1;
end
always @(posedge btn[1])begin //通过按键操作改变第 1 位操作数的值
    if(switch == 4'b01) op1[ 7: 4]<= op1[ 7: 4] + 4'd1;
    else if(switch == 4'b10) op2[ 7: 4]<= op2[ 7: 4] + 4'd1;
end
always @(posedge btn[2])begin //通过按键操作改变第 2 位操作数的值
    if(switch == 4'b01) op1[ 11: 8]<= op1[ 11: 8] + 4'd1;
    else if(switch == 4'b10) op2[ 11: 8]<= op2[ 11: 8] + 4'd1;
end
always @(posedge btn[3])begin //通过按键操作改变第 3 位操作数的值
   if(switch == 4'b01) op1[ 15: 12]<= op1[ 15: 12] + 4'd1;
   else if(switch == 4'b10) op2[ 15: 12]<= op2[ 15: 12] + 4'd1;
end
endmodule
/*-------------------------------------------------------------------*/
module division(switch,dividend,divisor,quotient,remainder,erro);
input wire [1:0] switch;
input[15:0] dividend; //被除数
input[15:0] divisor; //除数
output reg [15:0] quotient; //商
output reg[15:0] remainder; //余数
output reg erro;  //出错标志
reg[15:0] tempa;
reg[15:0] tempb;
reg[31:0] temp_a;
reg[31:0] temp_b;

always @(dividend or divisor)begin
```

```
    tempa <= dividend;
    tempb <= divisor;
    end

integer i;

always @(tempa or tempb)begin
    temp_a = {16'h0000,tempa}; //temp_a 的高 16 位赋 0，低 16 位赋被除数
    temp_b = {tempb,16'h0000}; //temp_b 的高 16 位赋除数，低 16 位赋零
    for(i = 0;i < 16;i = i + 1) //16 次循环
    begin
        temp_a = {temp_a[30:0],1'b0}; //temp_a 左移一位，末位补零
        if(temp_a[31:16] >= tempb) //temp_a 的高 32 位于 tempb 比较
            temp_a = temp_a - temp_b + 1'b1; //如果够减，则商的对应位赋 1
        else
            temp_a = temp_a;//如果不够减，则商的对应位赋零
    end

if((divisor==16'b0)&&(switch[1:0]==2'b11||switch[1:0]==2'b00))
erro<=1;//如果除数为零，而且此刻正在显示商或余数，那么需要提示出错信息
    else erro<=0;
    quotient <= temp_a[15:0]; //temp_a 的高 16 位是商
    remainder <= temp_a[31:16]; //temp_a 的低 16 位是余数

end
endmodule
```

UCF 定义文件

```
net "clk" loc = "T9";
net "segment[0]" loc = "E14";
net "segment[1]" loc = "G13";
net "segment[2]" loc = "N15";
net "segment[3]" loc = "P15";
net "segment[4]" loc = "R16";
net "segment[5]" loc = "F13";
net "segment[6]" loc = "N16";
net "segment[7]" loc = "P16";
net "anode[0]" loc = "D14";
net "anode[1]" loc = "G14";
net "anode[2]" loc = "F14";
net "anode[3]" loc = "E13";
net "btn_in[3]" loc = "L14";
net "btn_in[2]" loc = "L13";
net "btn_in[1]" loc = "M14";
```

```
net "btn_in[0]" loc = "M13";
net "btn_in[0]" clock_dedicated_route = false;
net "btn_in[1]" clock_dedicated_route = false;
net "btn_in[2]" clock_dedicated_route = false;
net "btn_in[3]" clock_dedicated_route = false;
net "switch[1]" loc= "K13";
net "switch[0]" loc ="K14";
```

八、 组内成员分工说明及贡献比例

成员分工

资料查阅：陶冠宏、王子思
程序代码：胡佳妮、陶冠宏、王子思
报告设计说明、模块结构、程序流程部分：胡佳妮
程序调试及分析：陶冠宏
模块仿真模拟及分析：王子思
执行过程视频拍摄及后期制作：陶冠宏、胡佳妮

贡献比例

组长：3100102879 胡佳妮 34%
组员：3100102697 陶冠宏 33%
3100103158 王子思 33%

九、 参考资料

[1] 周殿凤、王俊华，“基于 FPGA 的 32 位除法器设计”，信息化研究第 36 卷第 3 期
[2] 姚茂群，叶汉能，张立彬，“基于 FPGA 的除法器设计”，杭州师范大学学报（自然科学版）第 9 卷第 6 期
[3] 刘杰，“高速整数除法器的实现及仿真”，福建电脑 2007 年第 10 期
[4] 张延伟, 杨金岩, 葛爱学等，“Verilog HDL 程序设计实例详解”，（北京）人民邮电出版社, 2008
[5] (美)J.Bhasker 著 ； 徐振林等译，“Verilog HDL 硬件描述语言”，（北京）机械工业出版社, 2000