

**本科实验报告**

|  |  |
| --- | --- |
| 课程名称： | 计算机组成 |
| 姓 名： | 胡亮泽 |
| 学 院： | 计算机科学与技术学院 |
| 系： | 计算机科学与技术系 |
| 专 业： | 计算机科学与技术 |
| 学 号： | 3120102116 |
| 指导教师： | 姜晓红 |

2014年 3 月 21 日

**浙江大学实验报告**

课程名称： Computer Organization 实验类型： 综合

实验项目名称： Lab4： multi/divider

学生姓名： 胡亮泽 专业： 计算机科学与技术

学号： 3120102116

同组学生姓名： 王艺 指导老师： 姜晓红

实验地点： 东4-509 实验日期： 2014 年 3月 21 日

1. 实验目的和要求

利用开发板和verilog编程实现乘法器，掌握乘法器原理。

1. 实验内容和原理

1.**乘法器**

实验中top主模块代码如下：

|  |
| --- |
| module top(  input wire clk, //时钟  input wire cal, //计算命令按钮  input wire reset, //重置按钮  input wire choose,//数字变化位选择开关  input wire add\_button,//数字增加按钮  input wire disp\_switch, //切换计算数字和设置数字开关  output wire[7:0] segment,  output wire[3:0] anode,  output wire led //计算完毕判断小灯  );  wire[16:0] disp\_num; //数码管显示数字  wire[16:0] cal\_num; //计算数字  wire[16:0] add\_num; //设置数字，用以改变计算的数值  wire cal\_out;  wire add\_button\_out;  assign disp\_num = disp\_switch?cal\_num:add\_num;  pbdebounce a0(clk,cal,cal\_out);  pbdebounce a1(clk,add\_button,add\_button\_out);  multiply m1(choose,add\_button\_out,reset,cal\_out,cal\_num,led,add\_num);  display m0(clk,disp\_num[15:0],anode,segment);  endmodule |

代码中各变量的含义参照注释即可。

代码中assign disp\_num = disp\_switch?cal\_num:add\_num;

用以切换显示的数码管，在开关变化下可以切换到两个数字，显示调整数字或者计算中的步骤数字。

multiply m1(choose,add\_button\_out,reset,cal\_out,cal\_num,led,add\_num);

则为乘法计算的关键模块。

该模块的代码如下：

|  |
| --- |
| module multiply(input wire choose,  input wire add\_button,  input wire reset,  input wire cal,  output reg[17:0] disp\_num,  output wire led,  output reg[17:0] num  );  reg[2:0] count; //计算步骤完成数  reg flag; //用以标记设置数和计算数的赋值是否已经完成  initial begin  count = 0;  disp\_num = 17'h01x3;  num = 17'h00x0;  end  assign led = (count==4);  always @(posedge add\_button) begin  if (choose)  num[11:8] = num[11:8] + 1;  else num[3:0] = num[3:0] + 1;  end  always @(posedge cal,posedge reset) begin  if (reset) begin  count = 0;  disp\_num = num;  flag=0;  end  else if (~led) begin  if(~flag) begin  disp\_num=num;  flag = 1;  end  else if(disp\_num[8]) begin  disp\_num[16:12] = disp\_num[16:12] + disp\_num[3:0];  disp\_num[16:8] = disp\_num[16:8]>>1;  count = count+1;  end  else begin  disp\_num[16:8] = disp\_num[16:8]>>1;  count = count+1;  end  end  end  endmodule |

新变量的用途参照注释。

利用assign led = (count==4);语句

在4位乘法器完成4个计算步骤后，是表示计算结束的小灯亮起，同时表明计算结束，同时，除重置外其他的操作失效。

而

always @(posedge add\_button) begin

if (choose)

num[11:8] = num[11:8] + 1;

else num[3:0] = num[3:0] + 1;

end

模块，则用于调整操作数，对choose选择的位数进行增加。

而计算模块中，

if (reset) begin

count = 0;

disp\_num = num;

flag=0;

end

reset重置计算数字为初始操作数，并还原count和flag的值，以达成循环的目的。

由于第一次按下计算按钮后有一次将设置数赋值给计算数的过程，所以正常情况下，需要进行5次按键完成操作。

而在led为低电平，即计算未完成的情况下，根据加法和移位的方法完成乘法，在高8位中的最低位为1时进行加法后移位，否则仅仅移位。

else if(disp\_num[8]) begin

disp\_num[16:12] = disp\_num[16:12] + disp\_num[3:0];

disp\_num[16:8] = disp\_num[16:8]>>1;

count = count+1;

end

else begin

disp\_num[16:8] = disp\_num[16:8]>>1;

count = count+1;

end

end

详情参照代码，这里不多做赘述。

|  |
| --- |
| module display(clk,digit,anode,segment  );  input wire clk;  input wire[15:0] digit;  output reg[3:0]anode;  output reg[7:0]segment;  reg [3:0] code=4'b0;  reg [15:0] count=16'b0;  always @(posedge clk)  begin  case(count[15:14])  2'b00:begin  anode<=4'b1110;  code<=digit[3:0];  end  2'b01:begin  anode<=4'b1111;  code<=digit[7:4];  end  2'b10:begin  anode<=4'b1011;  code<=digit[11:8];  end  2'b11:begin  anode<=4'b0111;  code<=digit[15:12];  end  endcase  case(code)  4'b0000: segment <= 8'b11000000;  4'b0001: segment <= 8'b11111001;  4'b0010: segment <= 8'b10100100;  4'b0011: segment <= 8'b10110000;  4'b0100: segment <= 8'b10011001;  4'b0101: segment <= 8'b10010010;  4'b0110: segment <= 8'b10000010;  4'b0111: segment <= 8'b11111000;  4'b1000: segment <= 8'b10000000;  4'b1001: segment <= 8'b10010000;  4'b1010: segment <= 8'b10001000;  4'b1011: segment <= 8'b10000011;  4'b1100: segment <= 8'b11000110;  4'b1101: segment <= 8'b10100001;  4'b1110: segment <= 8'b10000110;  4'b1111: segment <= 8'b10001110;  default: segment <= 8'b00000000;  endcase  count<=count+1;  end  endmodule |

Display模块中，由于第2个数码管没有起到任何作用，因此在分频时，直接关闭该数码管

2'b01:begin

anode<=4'b1111;

code<=digit[7:4];

end

实验中ucf文件如下：

|  |
| --- |
| NET "clk" LOC="T9";  NET "cal" LOC="M13";  NET "cal" CLOCK\_DEDICATED\_ROUTE = FALSE;  NET "reset" LOC="M14";  NET "choose" LOC="F12";  NET "add\_button" LOC="L14";  NET "add\_button" CLOCK\_DEDICATED\_ROUTE = FALSE;  NET "led" LOC="K12";  NET "disp\_switch" LOC="K13";  NET "anode[0]" LOC="D14";  NET "anode[1]" LOC="G14";  NET "anode[2]" LOC="F14";  NET "anode[3]" LOC="E13";  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"; |

**2．除法器**

做完乘法器后，对除法器的概念和原理理解是有很大的帮助的，因此，除法器的实现过程更加简单了。

本次实验中，除法器的top主模块代码和乘法器完全相同，除了运算模块由multiplier改为divider，和choose选择按钮改为两位，以便对最高位进行修改。这里不作赘述。

另外，去抖动、display等模块也完全相同，下面重点分析divider运算模块。

该模块代码如下：

|  |
| --- |
| module divider(input wire[1:0] choose,  input wire add\_button,  input wire reset,  input wire cal,  output reg[17:0] disp\_num,  output wire led,  output reg[17:0] num  );  reg[2:0] count;  reg flag;  initial begin  count = 0;  disp\_num = 17'h00103;  num = 17'h0830a;  end  assign led = (count==5);  always @(posedge add\_button) begin  if (choose[1]&choose[0])  num[15:12] = num[15:12] + 1;  else if(choose[1]&~choose[0])  num[11:8] = num[11:8] + 1;  else num[3:0] = num[3:0] + 1;  end  always @(posedge cal,posedge reset) begin  if (reset) begin  count = 0;  disp\_num = num;  flag=0;  end  else if (~led) begin  if(~flag) begin  disp\_num=num;  flag = 1;  end  else if (count==4) begin  disp\_num[16:12]=disp\_num[16:12]-disp\_num[3:0];  disp\_num[11:8] = (disp\_num[11:8]<<1)+1;  count = count+1;  end  else if(disp\_num[16:12]<disp\_num[3:0]) begin  disp\_num[16:8] = disp\_num[16:8]<<1;  count = count+1;  end  else if(disp\_num[16:12]>=disp\_num[3:0]) begin  disp\_num[16:12]=disp\_num[16:12]-disp\_num[3:0];  disp\_num[16:8] = (disp\_num[16:8]<<1)+1;  count = count+1;  end  end  end  endmodule |

较乘法器而言，改变的地方主要有：

1. assign led = (count==5); 到运算结束为止需要5个运算步骤，加上第一次的赋值，需要按下6次运算按钮。
2. 第二个always触发模块中的运算。

这里重点分析第二点：

首先在以下代码中

else if (count==4) begin

disp\_num[16:12]=disp\_num[16:12]-disp\_num[3:0];

disp\_num[11:8] = (disp\_num[11:8]<<1)+1;

count = count+1;

end

由于实验要求，高8位在完成计算后需要分别显示余数和商，但由于每次计算结束后都会左移，为下一次的运算做准备，因此，在最后一次运算之前，需要改变移位的位数，即仅仅将高八位中的第四位左移。

在运算的代码中：

else if(disp\_num[16:12]<disp\_num[3:0]) begin

disp\_num[16:8] = disp\_num[16:8]<<1;

count = count+1;

end

else if(disp\_num[16:12]>=disp\_num[3:0]) begin

disp\_num[16:12]=disp\_num[16:12]-disp\_num[3:0];

disp\_num[16:8] = (disp\_num[16:8]<<1)+1;

count = count+1;

end

end

若高位较除数小，则仅仅左移，否则会在减去除数的基础上左移。

1. 实验过程和数据记录

乘法器实验中采用的检验数据如下：

1. 15 \* 8 = 120

|  |  |  |  |
| --- | --- | --- | --- |
| **0** | **A** | **X** | **D** |
| 0 | 5 | X | D |
| 6 | A | X | D |
| 3 | 5 | X | D |
| 8 | 2 | X | D |

运行结果完全对应。

除法器：

0x83/0x0a=ox0d……1

结果：

|  |  |  |  |
| --- | --- | --- | --- |
| 8 | 3 | X | A |
| 0 | 6 | X | A |
| c | D | X | A |
| 5 | B | X | A |
| b | 3 | X | A |
| 1 | d | X | A |

实验结果完全符合

1. 实验结果

本次实验中的问题主要有3点：

1. 一定要去抖动。对于这种对于按键次数要求严格的实验，没有去抖将会造成结果不受控制。
2. 一定要注意进位。在忘记进位的情况下会造成实验数据出错。
3. 注意时钟干扰。可以再ucf文件中，通过对干扰的变量增加语句：

NET”变量名” CLOCK\_DEDICATED\_ROUTE=FALSE

来解决。

在解决了以上问题后，本次实验圆满完成。

1. 讨论与心得

本次实验主要加深了我对于乘法器和除法器在计算机中执行原理的理解，并通过实验获得了新思路！