**实验六 16位乘法器设计与实现**

**一、实验目的**

1、了解16位有符号、无符号乘法器的实现原理。

2、使用Verilog实现16位无符号乘法器和有符号乘法器。

3、使用Verilog设计两个4位无符号数的乘法电路并下载、验证其功能。

**二、实验设备**

1、装有vivado的计算机 1台

2、EGO1开发板 1块

**三、实验任务**

1、设计无符号乘法器，将两个8位无符号数相乘,得到一个16位无符号数。

2、设计有符号乘法器，将两个8位有符号数相乘,得到一个16位有符号数。

3、设计按键输入转换电路，设计七段数码管显示和控制电路： 通过8个按键输入两个4位的二进制乘数A和B，将两个乘数A、B和乘积，送给8个七段数码管进行显示。

**四、实验原理**

1、乘法的基本概念：

被乘数x为1000，乘数y为1001，下面的乘法过程是手工运算的一个步骤,而计算机在做乘法时就是模拟手工运算的执行过程。如图6.1所示

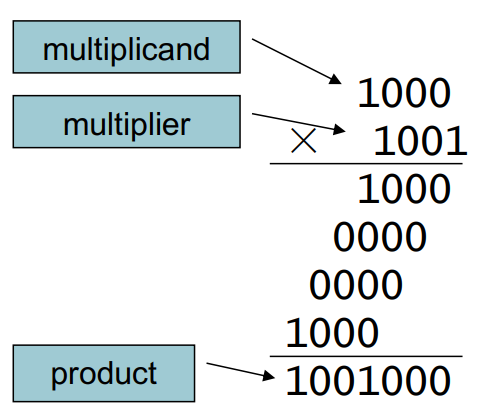
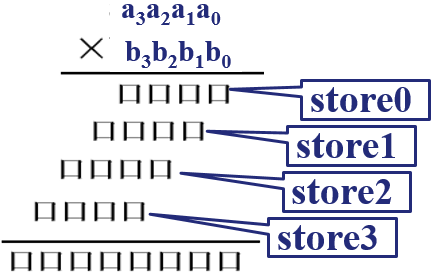
 

图6.1 手工乘法的执行过程

2、根据上面的手工计算过程，画出与之对应的硬件电路，如图6.2所示。

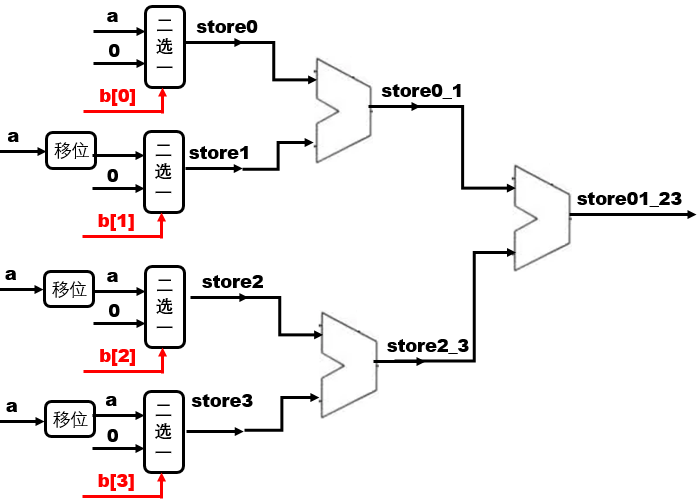


图6.2 乘法器的一种硬件结构

3、无符号乘法器的功能为:将两个16位无符号数相乘,得到一个32位无符号数﹐如图6.3所示。

接口定义:

module MULTU(

input clk, //乘法器时钟信号

input reset, //复位信号,低电平有效

input [ 15:0] a, //输入数a(被乘数)

input [ 31:0] b, //输入数b(乘数)

output [ 31:0] z //乘积输出z

);

4、有符号乘法器的功能为:将两个16位有符号数相乘,得到一个32位有符号数,如图6.4所示。

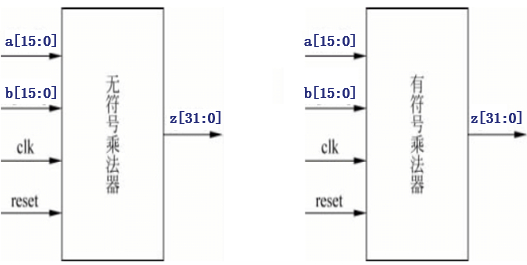


图6.3无符号乘法器 图6.4有符号乘法器

接口定义:

module MULT(

input clk, //乘法器时钟信号

input reset, //复位信号,低电平有效

input [15:0] a, //输入数a(被乘数)

input [ 15:0] b, //输入数b(乘数)

output [ 31:0] z //乘积输出z

);

5、无符号乘法器功能为:将两个16位无符号数相乘,得到一个32位无符号数。有符号乘法器功能为:将两个16位有符号数相乘,得到一个32位有符号数。将低16位存放在专用寄存器lo中,高16位存放在寄存器hi中。执行乘法指令过程中不产生异常。

本实验不建议使用行为级描述方式实现乘法电路，太简单。

6、下面提供几种实验实现思路,仅供参考。大家做实验时，可以选择一种，或者都实现。

**思路(1)**：两个二进制数a和 b相乘,可以认为是a和 b的每一位相乘移位后的结果相加。关于a与b的每一位相乘产生的中间结果,如果b那位是0,那么中间结果就是0;如果是1 ,那么中间结果就是在a前后补上相应位数的零，通过字符拼接的方式表示。然后将这些中间乘积相加就是最后的结果。采用这种方式时，加法器的占用数量比较多。

**思路(2）**：二进制的乘法可以用加法和移位操作完成，可以循环迭代的方法实现。每次循环时,判断b的值是否为1,然后决定是否将中间值加上a。每次循环,a左移一位,b右移一位。循环结束,最后的中间值就是最后的乘积。采用这种方式时，加法器的数量可以显著减少。

**思路(3）**：可以从Wallace Tree乘法（并行乘法电路）算法的角度出发实现。有兴趣实现的读者可以自行查阅Wallace Tree算法实现的特点。采用这种方式时，加法器的耗费数量显著增多，优点是多个加法器的并行工作，有效减少了乘法计算所需要的总时间。

7、下面按照上面思路（1）来举例说明8位无符号数乘8位无符号数的一种实现方式。

对应的verilog参考如下：

module MULTU(

input clk, //乘法器时钟信号

input reset,

input [ 7:0] a, //输入a(被乘数)

input [ 7:0] b, //输入b(乘数)

output [ 15:0] z //乘积输出z

);

reg [ 15:0 ] temp; //申请寄存器

reg [ 15:0 ] stored0;

reg [ 15:0 ] stored1;

reg [ 15:0 ] stored2;

reg [ 15:0 ] stored3;

reg [15 :0] stored4;

reg [15 :0] stored5 ;

reg [ 15:0 ] stored6;

reg [ 15:0 ] stored7;

reg [ 15:0 ] add0\_1;

reg [ 15:0 ] add2\_3;

reg [ 15:0 ] add4 \_5;

reg [ 15:0 ] add6\_7;

reg [ 15:0 ] add0t1\_2t3;

reg [ 15:0 ] add4t5\_6t7;

reg [ 15:0 ] add0t3\_4t7;

always @( posedge clk or negedge reset)

begin

if(reset) begin //reset 置零

temp <= 0;

stored0 <= 0 ;

stored1 <= 0 ;

stored2<= 0 ;

stored3<= 0 ;

stored4 <= 0 ;

stored5<= 0 ;

stored6<= 0 ;

stored7 <= 0 ;

addO\_1 <= 0;

add2\_3<= 0;

add4\_5 <= 0;

add6\_7 <= 0 ;

add0t1\_2t3<= 0;

add4t5\_6t7<= 0;

end

else begin //通过字符拼接方式表示出中间相乘值,并相加，参见图6.1和图6.2

stored0= b[0] ? { 8' b0,a} : 16'b0 ;

stored1 = b[1] ? { 7’b0,a,1'b0} :16' b0 ;

stored2= b[2] ? {6' b0,a,2'b0} :16' b0 ;

stored3= b[3] ? { 5'b0,a,3 'b0} :16' b0 ;

stored4 = b[4] ? {4' b0,a,4 'b0} :16' b0 ;

stored5= b[5] ? { 3' b0,a,5 'b0} :16' b0;

stored6= b[6] ? { 2'b0,a,6'b0} :16' b0 ;

stored7 = b[7] ? { 1' b0,a,7'b0} :16' b0 ;

add0\_1 = stored1 + stored0 ;

add2\_3= stored2 + stored3 ;

add4\_5= stored4 + stored5 ;

add6\_7= stored6 + stored7;

add0t1\_2t3= add0\_1 + add2\_3;

add4t5\_6t7 = add4\_5 + add6\_7;

temp =add0t1\_2t3 + add4t5\_6t7;

end

end

assign z = temp;

endmodule

8、写完乘法电路模块,可以采用以下数据进行测试。

a = 0, b = 0; a = 0,b = 8'b11111111;

a = 8' b10110011,b = 0 ; a = 8'b11111111, b = 8'b11111111;

a = 8'b10000000, b = 8'b10101010; a = 8'b10101010，b = 8'b1000000o;

a = 8'b101101 ; b= 8'b1101000; a = 8' b1000111, b = 8'b1110

(注意:在写32位乘法器的时候，需要用32位的数据进行测试。)

9、至于有符号数乘法器的实现只需要进行简单的变动,即可实现。请大家自己思考。本实验不允许使用行为级(即乘号方式)方式实现乘法运算。

**五、实验步骤**

**（一）设计无符号乘法器:将两个16位无符号数相乘,得到一个32位无符号数﹐参考图6.2和图6.3类似的结构图。（实验室完成仿真验收和下载验收）**

1、**新建Vivado 工程**

**参照实验五的步骤创建新工程。**

由2个16位无符号数相乘,得到一个32位无符号数，接口定义类似如下:

module MULTU(

input clk, //乘法器时钟信号

input reset,

input [ 15:0] a, //输入a(被乘数)

input [15:0] b, //输入b(乘数)

output [ 31:0] z //乘积输出z

);

填写下面乘法器模块的verilog代码。

**乘法器模块**：

module MULTU(…… )

请在此处补充对应的verilog代码，要求在实验前完成。进实验室时会检查。验收时会围绕核心代码进行提问。

endmodule

**2、在Vivado中进行仿真测试主模块。**

建立了乘法模块后，如果需要对模块进行测试的话，就需要给电路添加相应的激励输入，观察电路输出所具有的特点。例如：对两个乘数，分别赋值为21H和36H，然后观察乘积输出结果。观察输出波形是否正确。

**激励输入模块：**

module MULTUtest(…… )

请在此处补充对应的verilog代码，要求在实验前完成。进实验室时会检查。验收时会围绕核心代码进行提问。

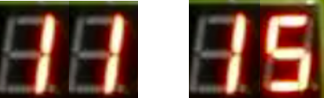
endmodule

进行仿真，验证仿真后的波形图是否正确。

在实验报告上，粘贴你的仿真截图，并分析结果的正确性。

**3、设计两个4位无符号数的乘法电路并下载、通过8个开关输入四位二进制乘数A和B，将乘数和乘积都以十进制形式显示在七段数码管上，通过开关，改变两个乘数的值，验证乘法器是否功能正确。**

**首先，在Vivado中设计输入电路。按键sw3~sw0对应于一个四位的二进制数A[3:0]，按键sw7~sw4对应于另外一个四位的二进制数B[3:0]，设计二进制数到8421BCD码的转换电路。将二进制数A[3:0]转换得到的8421BCD码对应的数值显示在两个数码管上。类似的，将二进制数B[3:0]转换得到的8421BCD码对应的的数值显示在另外两个数码管上。效果如下图所示：**



**然后， 设计一个可以实现两个4位二进制数相乘的乘法电路（参考图6.2），实现将A[3:0]与B[3:0]进行相乘。乘积结果显示在另外4个数码管上。效果如图所示：**

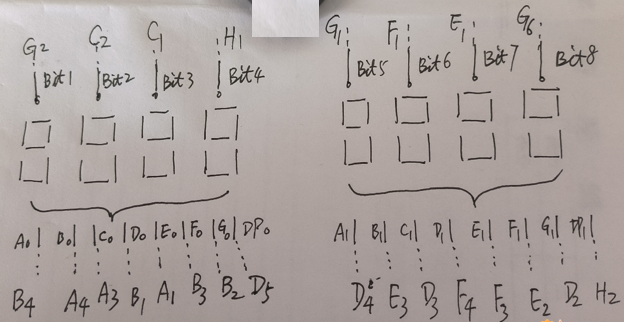


**4、在Vivado中进行引脚绑定，并且将乘积的结果通过七段数码管进行动态显示。**

**（1）要想在多个七段数码管上进行动态刷新显示乘积结果，需要设计定时10ms的时钟，每隔10ms就在一个七段数码管上显示乘积结果中的一位。由于视觉暂停效应，人眼是看不清多个七段数码管的值是分时显示的。人眼感觉到的是多个七段数码管上的值是同时显示的。**

**（2）对于EGO1开发板上的8个七段数码管，左边的四个数码管的段码输入A、B、C、D、E、F、G和小数点DP是共用的，高电平对应的段会亮，可以通过位码控制信号选中某个数码管显示，位码高电平有效，位码为高电平时，对应的某个数码管选中会亮。右边的四个数码管的段码输入A、B、C、D、E、F、G和小数点DP也是共用的，但和左边的四个数码管是不相连的。右边四个数码管的控制方式和左边类似。**

**（3）EGO1开发板的8个七段数码管的结构如下图。**



**5、在vivado中生成bit流文件和下载，在EGO1开发板上进行乘数输入,观察七段数码管上的乘积输出结果。验证结果是否正确。在实验报告上只能放置一张下载成功的拍照。**

**参考代码：**

**//控制8个数码管动态显示的模块。seg0[3:0]和seg1[3:0]分别是两个8421BCD码。**

**//信号select为8个数码管的位选控制。**

**module ShowTwoSeg7( input clk, input [3:0] seg0, input [3:0] seg1,**

**output reg [7:0] select, output reg [7:0] seg );**

**reg[2:0] pos;**

**reg [3:0] bcd;**

**always @(posedge clk) begin**

**pos <= pos + 1'b1;**

**end**

**always @(\*) begin**

**case(pos)**

**3'b000: begin**

**select = 8'b00000001;**

**bcd = seg0;**

**end**

**3'b001: begin**

**select = 2'b00000010;**

**bcd = seg0;**

**3'b010: begin**

**select = 8'b00000100;**

**bcd = seg0;**

**end**

**3'b011: begin**

**select = 2'b00001000;**

**bcd = seg0;**

**3'b100: begin**

**select = 8'b00010000;**

**bcd = seg1;**

**end**

**3'b101: begin**

**select = 2'b00100000;**

**bcd = seg1;**

**3'b110: begin**

**select = 8'b01000000;**

**bcd = seg1;**

**end**

**3'b111: begin**

**select = 2'b10000000;**

**bcd = seg1;**

**end**

**endcase**

**end**

**//8421BCD码bcd与一位数码管的8段A、B、C、D、E、F、G、DP之间的对应关系**

**always @(\*) begin**

**case(bcd)**

**4'h0: seg = 8'hfc;**

**4'h1: seg = 8'h60;**

**4'h2: seg = 8'hda;**

**4'h3: seg = 8'hf2;**

**4'h4: seg = 8'h66;**

**4'h5: seg = 8'hb6;**

**4'h6: seg = 8'hbe;**

**4'h7: seg = 8'he0;**

**4'h8: seg = 8'hfe;**

**4'h9: seg = 8'hf6;**

**// 4'ha: seg = 8'hee;**

**// 4'hb: seg = 8'h3e;**

**// 4'hc: seg = 8'h9c;**

**// 4'hd: seg = 8'h7a;**

**// 4'he: seg = 8'h9e;**

**// 4'hf: seg = 8'h8e;**

**default: seg = 8'h00; //八段全熄灭**

**endcase**

**end**

**initial begin**

**pos=0;**

**end**

**endmodule**

**//数码管动态扫描所需要的时钟模块**

**module clk\_div(**

**input clk0,**

**output reg clk**

**);**

**parameter N = 32'd51200, WIDTH = 32 - 1;**

**reg [WIDTH : 0] number = 0;**

**initial clk0=0;**

**always @(posedge clk0) begin**

**if (number == N - 1) begin**

**number <=0;**

**clk <= ~clk;**

**end**

**else begin**

**number <= number + 1;**

**end**

**end**

**endmodule**

**//主模块没有给出代码需要大家自己设计，主要是完成引脚定义和子模块的例化。**

**module top（clk,** **data1, data2, mul ,select, result）；**

**input clk; // 输入时钟，需要分频后进行动态扫描控制。**

**input[3:0] data1; // 输入乘数A**

**input[3:0] data2; //输入乘数B**

**output reg[7:0] select; // 八个数码管的位选控制端**

**output reg[7:0] mul; // 乘数A和B的十、个位分时送给mul,由mul送给数码管**

**output reg[7:0] result; //乘积的千、百、十、个位分时送给result。进行动态显示**

**补充说明：**

**1、验收到步骤2，只有仿真图的验收，验收成绩是80分。验收到步骤5，下载成功，有下载后的演示验收，验收成绩是：100分。**

**2、验收时，要简单介绍一下相关的波形代表的信号，相关的波形对应的乘数、被乘数、乘积具体的值等。**

**六、实验思考：**

1、按照实验原理里的第6条思路（2）进行设计，所对应的电路结构是怎样的。

2、按照实验原理里的第6条思路（3）进行设计，所对应的电路结构是怎样的。