**计算机组成与设计实验报告**

***实验五. 存储器实现和乘法器的优化***

学 院： 网络空间安全学院（研究院）

成 员： 孟庆丹、牛中原

2022年11月8日

***实验五. 存储器实现和乘法器的优化***

成员分工：

孟庆丹：设计实验，编写实验代码，完成仿真执行，写实验报告

牛中原：验证实验并完成实验报告

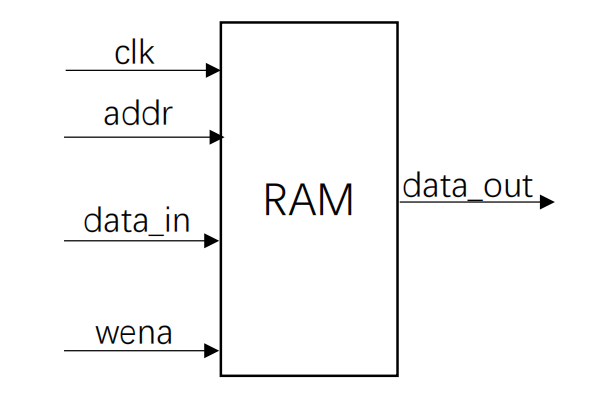
[报告摘要]

1. 在这次实验中，我们首先实现了能写入写出指定地址的8bit数据的RAM，并利用modelsim仿真进行了验证。
2. 接下来的实验中，我们将上节课实现的组合逻辑乘法器进行了时序实现，对于普通的时序乘法器，完成一次有符号的四位乘法需要四个周期，在第五个周期时钟上升沿输出结果。
3. 之后我们对时序乘法器进行改进得到LUT乘法器，使得乘法器能够在一个时钟周期内完成运算。

[报告正文]

**1. 实现一个 32\*8 bit 的 RAM。**

用时序逻辑实现一个 RAM，本题可以通过 modulesim仿真以及安路 FPGA 开发板实现。



RAM模块代码：

`timescale 1ns / 1ps

module RAM1(

 input clk,

 input wena,

 input [4:0] addr,

 input [7:0] data\_in,

 output reg [7:0] data\_out

);

reg [7:0] mymem [31:0];

always@(posedge clk)

begin

if(wena)

    mymem[addr]<=data\_in;

else

    data\_out<= mymem[addr];

end

endmodule

仿真代码：

`timescale 1ns / 1ps

module ram\_test(

);

reg clk1 = 0;

reg wena;

reg [4:0]addr;

reg [7:0]datain;

wire [7:0]dataout;

RAM1 ram(clk1,wena,addr,datain,dataout);

always #5 clk1 = ~clk1;

initial begin

    wena = 1;

    addr = 5'b00001;

    datain = 8'b00100100;

    #30

    wena = 0;

    addr = 5'b00001;

    #30

    wena = 1;

    addr = 5'b00011;

    datain = 8'b00101100;

    #30

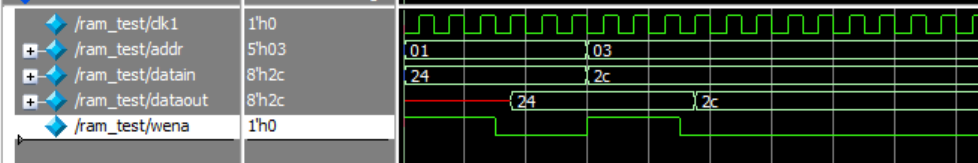
    wena = 0;

    addr = 5'b00011;

end

endmodule

仿真结果：



根据仿真结果，在wena变化的下一个时钟信号上升沿，RAM进行数据的写入或写出，在wena信号为0时，RAM进行数据的写入，当RAM信号为1时，RAM进行数据的写出。仿真结果与预期结果完全相同。

2. 在实验四的四位有符号乘法器基础上，完成时序乘法器的设计和优化。

2.1时序乘法器的设计

时序乘法器会在四个时钟周期内完成运算，在第一个时钟周期进行初始化操作，在之后三个时钟周期每个周期完成一次移位相加，在第五个时钟上升沿时进行输出

时序乘法器代码：

`timescale 1ns / 1ps

module mul4b(

    input clk,

    input[3:0] A,

    input[3:0] B,

    output reg[7:0]r

    );

    reg[3:0]a2;

    reg[3:0]b2;

    reg[7:0] S; //积寄存器

    reg[7:0] T; //被乘数寄存器

    reg state=0;

    reg sign;

    reg [3:0]B\_reg;//乘数寄存器

    reg [2:0] cnt;

    always @(posedge clk) begin

        case(state)

            1'b0:begin //第一个周期 初始化

                sign = A[3] ^ B[3];//sign为1负

                a2 = A;

                b2 = B;

                if(B[3]==1)

                    b2 = ~B + 1;

                if(A[3]==1)

                    a2 = ~A + 1;

                B\_reg = b2;

                S = 8'b0000\_0000;

                T = {4'b0000,a2};

                cnt = 3'b000;

                state = 1'b1;

            end

            1'b1:begin

                if(cnt < 3'b011) begin //第2、3、4周期，运算

                    if(B\_reg[0] == 1'b1) begin

                        S <= S + T;

                    end

                    else begin

                        S <= S;

                    end

                    B\_reg <= B\_reg>>1; // B\_reg[0]置于判断位

                    T <= T<<1;

                    cnt <= cnt + 1'b1;

                end

                else begin//第5个周期，判断正负

                    if(sign)

                    begin

                        S = ~S + 1;

                    end

                    state <= 1'b0;

                    r = S;

                end

            end

            default:;

        endcase

    end

endmodule

**仿真代码：**

`timescale 1ns / 1ps

module mul4b\_test();

reg [3:0]a1;

reg [3:0]b1;

reg clk1 = 0;

wire [7:0]z1;

mul4b mul4b1(clk1, a1, b1, z1);

always #5 clk1 = ~clk1;

initial begin

    a1 = 4'b0010;

    b1 = 4'b0010;

    #100

    a1 = 4'b0011;

    b1 = 4'b0010;

    #100

    a1 = 4'b0011;

    b1 = 4'b0110;

    #100

    a1 = 4'b0101;

    b1 = 4'b0010;

    #100

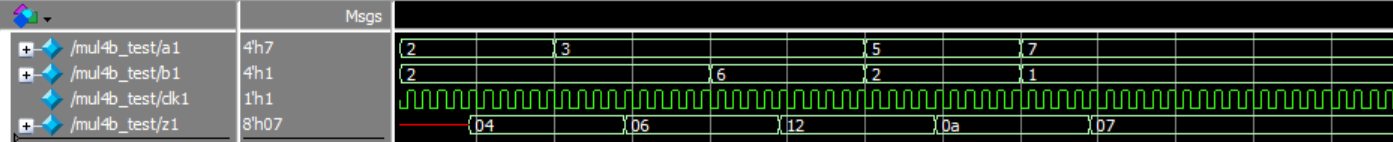
    a1 = 4'b0111;

    b1 = 4'b0001;

end

endmodule

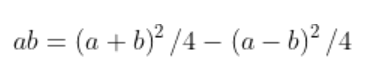
**仿真结果：**

****

我们的时序乘法器只能在运行四个周期后输出运算结果，我们认为运算速度不够快，选择利用Quarter Square乘法对其进行改进。

2.2改进的LUT时序乘法器

根据下面公式，



我们发现a与b的乘积能够通过 以及来进行表示，那么我们就可以尝试存储对应着x以及的字典进行快速的查找表，总共需要16组键值对就可以完成查找表的制作。

LUT乘法器代码：

`timescale 1ns / 1ps

// 查找表乘法器实现

module mul4c(

    input clk,

    input[3:0] A,

    input[3:0] B,

    output reg[7:0]r

    );

    reg state = 0;

    reg [4:0]add;

    reg [4:0]sub;

    reg [7:0]a1;

    reg [7:0]b1;

always @(posedge clk) begin

        if(state == 0)

        begin

            add = A + B;

            sub = A + (~B + 1);

            if (add[4] == 1)

                add = ~add + 1;

            if (sub[4] == 1)

                sub = ~sub + 1;

            state = state + 1;

        end

        else if(state == 1)

        begin

            case(add)

            5'b00000:a1 = 8'b00000000;

            5'b00001:a1 = 8'b00000000;

            5'b00010:a1 = 8'b00000001;

            5'b00011:a1 = 8'b00000010;

            5'b00100:a1 = 8'b00000100;

            5'b00101:a1 = 8'b00000110;

            5'b00110:a1 = 8'b00001001;

            5'b00111:a1 = 8'b00001100;

            5'b01000:a1 = 8'b00010000;

            5'b01001:a1 = 8'b00010100;

            5'b01010:a1 = 8'b00011001;

            5'b01011:a1 = 8'b00011110;

            5'b01100:a1 = 8'b00100100;

            5'b01101:a1 = 8'b00101010;

            5'b01110:a1 = 8'b00110001;

            5'b01111:a1 = 8'b00111000;

            endcase

            case(sub)

            5'b00000:b1 = 8'b00000000;

            5'b00001:b1 = 8'b00000000;

            5'b00010:b1 = 8'b00000001;

            5'b00011:b1 = 8'b00000010;

            5'b00100:b1 = 8'b00000100;

            5'b00101:b1 = 8'b00000110;

            5'b00110:b1 = 8'b00001001;

            5'b00111:b1 = 8'b00001100;

            5'b01000:b1 = 8'b00010000;

            5'b01001:b1 = 8'b00010100;

            5'b01010:b1 = 8'b00011001;

            5'b01011:b1 = 8'b00011110;

            5'b01100:b1 = 8'b00100100;

            5'b01101:b1 = 8'b00101010;

            5'b01110:b1 = 8'b00110001;

            5'b01111:b1 = 8'b00111000;

            endcase

            r = a1 + (~b1 + 1);

            state = state + 1;

        end

    end

endmodule

仿真代码：

`timescale 1ns / 1ps

module mul4c\_test();

reg [3:0]a1;

reg [3:0]b1;

reg clk1 = 0;

wire [7:0]z1;

mul4c mul4c1(clk1, a1, b1, z1);

always #5 clk1 = ~clk1;

initial begin

    a1 = 4'b0010;

    b1 = 4'b0010;

    #100

    a1 = 4'b0011;

    b1 = 4'b0010;

    #100

    a1 = 4'b0011;

    b1 = 4'b0110;

    #100

    a1 = 4'b0101;

    b1 = 4'b0010;

    #100

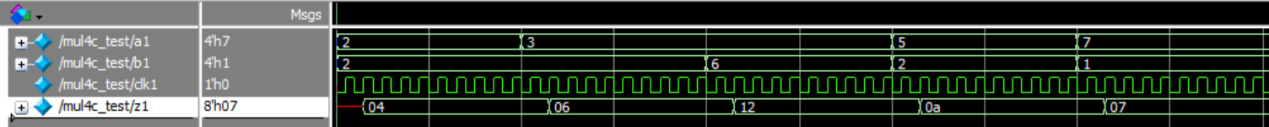
    a1 = 4'b0111;

    b1 = 4'b0001;

end

endmodule

仿真结果：



改进后的时序乘法器只需要一个周期就可以完成运算，大幅提高了运算效率，查找表法乘法器本质上是以空间换时间，我们通过对**Quarter Square乘法公式的使用，减少了查找表所占的空间，使其只需要16组数据，在提高运算效率的同时，尽可能减少了对空间的利用。**

参考资料：

https://www.cnblogs.com/vinsonnotes/articles/15818269.html

https://blog.csdn.net/weixin\_44502896/article/details/106534817

https://www.runoob.com/w3cnote/verilog-pipeline-design.html