第七次实验——乘法器

一、实验目的

1.实现$8bit \times 8bit$的乘法器

2.模拟硬件层面的二进制乘法

3.支持输入数据与输出数据的切换和显示

二、实验原理

从手算的角度来说，二进制乘法与十进制乘法毫无区别，只是进位不同罢了。以十进制乘法为例，计算$m \times \overline{a_{n-1}a_{n-2}...a_1a_0}$，手算的竖式实际就是将乘数$\overline{a_{n-1}a_{n-2}...a_1a_0}$展开为：

$$\sum_{i=0}^{n-1} a_i \times 10^i = a_{n-1} \times 10^{n-1} + a_{n-2} \times 10^{n-2} + ... + a_1 \times 10^1 + a_0 \times 10^0$$

然后运用分配律，将被乘数乘入因数中，便是结果：

$$Result = \sum_{i=0}^{n-1} a_i m \times 10^i$$

而在竖式表达式中为了节省时间，由于尾位已经对齐，可以将所有由10的幂产生的0都省略不写。这便是将乘法化为了n次简单的乘法和加法。

二进制乘法的竖式运算就是将上述式中的10都换为2：

$$Result = \sum_{i=0}^{n-1} a_i m \times 2^i$$

但由于二进制数中只有0和1两种数，因此$a_i m$要么等于0，要么就等于m自己。而后面的$\times 2^i$，也如$\times 10^i$一般，只会在$a_i m$之后产生i个0。对于计算机而言，就是将数左移i位。

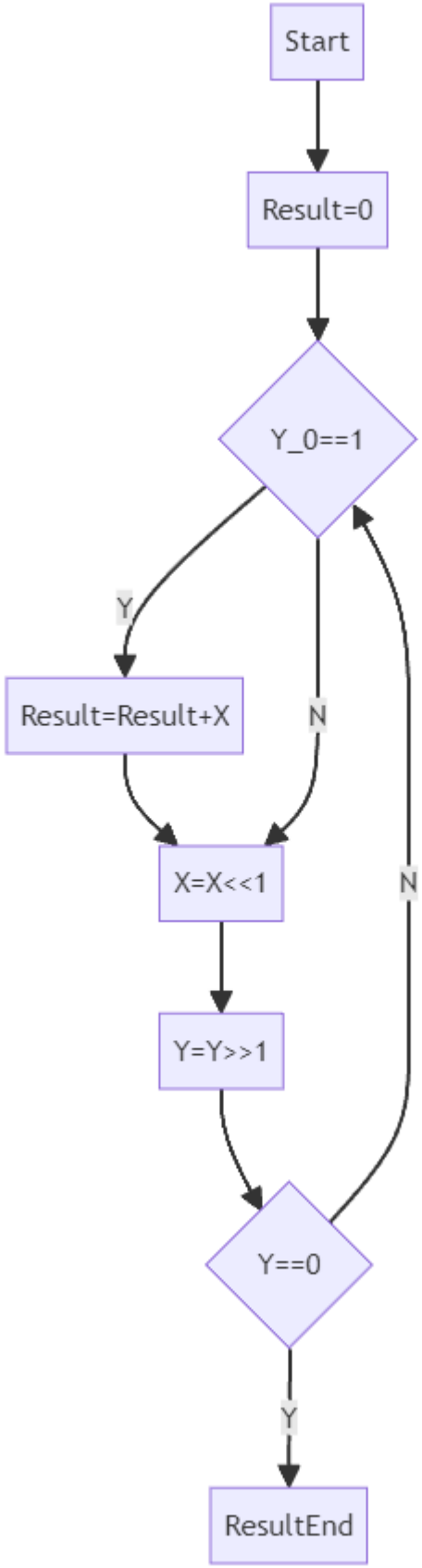
因此，二进制乘法可以改进为：首先假设结果$Result$为0。对乘数$\overline{a_{n-1}a_{n-2}...a_1a_0}$从最低位遍历到最高位，如果$a_i = 0$，那么显然$a_i m = 0$，结果$Result$加上$0$相当于不变；而如果$a_i = 1$，那么

$Result$就要加上m左移i位的结果。这样，遍历到乘数的最高位即可结束。当然，之后会提到也可以通过某些等效处理，提前结束遍历以提高效率。

而在硬件中又应当如何处理呢？首先对于被乘数m，为了统一，遍历过程中应当每一次都左移一位，而结果$Result$是否加上当前生成的数则由a_i决定；其次，乘数每一次都右移一位，这样每次只需取出乘数的最低位即可。并且，这样做的话，如果乘数从某一位开始其更高位都是0，那么之后的遍历$Result$都无需变化，显然可以提前结束遍历，也就是判断当前右移后的乘数是否等于0即可。

这样，$8bit \times 8bit$的$X \times Y = Result$的原理流程图就是：

graph TB Start-->R[Result=0]-->Judge1{Y_0==1}--Y-->Plus[Result=Result+X]-->SLL[X=X<<1]-->SLR[Y=Y>>1]-->Judge2{Y==0}--Y-->ResultEnd Judge1--N-->SLL Judge2--N-->Judge1


Start
Result=0
Y_0==1
Y
Result=Result+X
N
X=X<<1
N
Y=Y>>1
Y==0
Y
ResultEnd

三、实验过程

1.乘法器 `Multiplier`

这个模块是整个实验的核心。代码如下：

```
module Multiply(
    input CLK_in,
    input [7 : 0] Src1,
    input [7 : 0] Src2,
    output reg [15 : 0] Result
    );

    reg [1 : 0] stage = 0;
    reg [2 : 0] count = 0;
    reg [7 : 0] temp;
    reg [15 : 0] result;
    reg [15 : 0] extend;

    always@(posedge CLK_in) begin
        case (stage)
            2'b00: begin
                count <= 0;
                result <= 0;
                temp <= Src2;
                extend <= { 8'h00, Src1 };
                stage <= 2'b01;
            end
            2'b01: begin
                if (count == 7) stage <= 2'b10;
                else begin
                    if (temp[0] == 1) result <= result + extend;
                    else result <= result;
                    temp <= temp >> 1;
                    extend <= extend << 1;
                    count <= count + 1;
                end
                if (temp == 0) stage <= 2'b10;
            end
            2'b10: begin
                Result <= result;
                stage <= 2'b00;
            end
        endcase
    end

endmodule
```

首先，$Result$必然是$8+8=16$位的。由于$Src1$左移不能丢失高位，因此必须使用一个16位的寄存器来存储，这里被命名为$extend$。而$temp$则是$Src2$的允许修改的副本，每次都被右移。

$stage$用于标示乘法器现在所处的状态。由于不能直接修改源操作数，因此第一个状态是正在做初始化工作与生成允许修改的副本。之后进入第二个状态。这是乘法器的工作阶段，是实验原理中的流程图的实现。判断结束之后进入第三个状态，便是输出结果。

2.显示切换 Switch

这个模块用于切换显示。由于需要显示两个输入的数、或者显示计算结果，每按下一次按键，就切换为另一种显示模式。这里是利用二进制数在加一的过程中末位必在0和1之间切换实现的。

```
module Switch(
    input Button_in,
    output reg Switch_out
    );

    reg [7 : 0] count = 0;

    always@(posedge Button_in) begin
        count = count + 1;
    end

    always@(count) begin
        if (count[0] == 0) Switch_out = 0;
        else Switch_out = 1;
    end

endmodule
```

3.显示模块 Display

显示模块已经写过多次，直接复用以前的代码。

```
module Display(
    input CLK_in,
    input [15 : 0] Data,
    output reg [3 : 0] segment,
    output reg [6 : 0] position
    );

    reg [1 : 0] temp;
    reg [3 : 0] x;

    always@(posedge CLK_in) begin
        case (temp)
            0: begin x = Data[15 : 12]; segment = 4'b0111; end
            1: begin x = Data[11 : 8]; segment = 4'b1011; end
            2: begin x = Data[7 : 4]; segment = 4'b1101; end
            3: begin x = Data[3 : 0]; segment = 4'b1110; end
        endcase
        temp = temp + 1;
        if (temp % 4 == 0) temp = 0;
    end

    always@(*) begin
        case (x)
            4'h0: position = 7'b0000001;
            4'h1: position = 7'b1001111;
            4'h2: position = 7'b0010010;
            4'h3: position = 7'b0000110;
            4'h4: position = 7'b1001100;
            4'h5: position = 7'b0100100;
            4'h6: position = 7'b0100000;
            4'h7: position = 7'b0001111;
            4'h8: position = 7'b0000000;
            4'h9: position = 7'b0000100;
            4'hA: position = 7'b0001000;
            4'hB: position = 7'b0010000;
            4'hC: position = 7'b0110001;
            4'hD: position = 7'b0010001;
            4'hE: position = 7'b0110000;
            4'hF: position = 7'b0111000;
        endcase
    end
```

```
endmodule
```

4.时钟分频 CLK_div

时钟分频用于提供显示模块所需时钟频率。

```
module CLK_div #(parameter N = 99999)(
    input CLK_in,
    output CLK_out
    );

    reg [31 : 0] counter = 0;
    reg out = 0;

    always@(posedge CLK_in) begin
        if (counter == N - 1) counter <= 0;
        else counter <= counter + 1;
    end

    always@(posedge CLK_in) begin
        if (counter == N - 1) out <= !out;
    end

    assign CLK_out = out;

endmodule
```

5.按键消抖 ButtonDebounce

按键消抖模块需要接入专用的时钟频率。

```
module ButtonDebounce(
    input CLK_in,
    input Button_in,
    output Button_out
    );

    reg [2 : 0] Button = 0;

    always@(posedge CLK_in) begin
        Button[0] <= Button_in;
        Button[1] <= Button[0];
        Button[2] <= Button[1];
    end

    assign Button_out = (Button[2] & Button[1] & Button[0]) | (~Button[2] & Button[1] & Button[(

endmodule
```

6.顶层文件 Extern

顶层文件负责对外界显示，故命名为 Extern 。

```
module Extern(
    input CLK,
    input Button,
    input [15 : 0] Switch,
    output [1 : 0] Light,
    output [3 : 0] Segment,
    output [6 : 0] Position
    );

    wire CD_wire;
    wire CD2_wire;
    wire BD_wire;
    wire DS_wire;
    wire [15 : 0] Result_wire;

    CLK_div CD (
        .CLK_in( CLK ),
        .CLK_out( CD_wire )
    );

    CLK_div2 CD2 (
        .CLK_in( CLK ),
        .CLK_out( CD2_wire )
    );

    ButtonDebounce BD (
        .CLK_in( CD2_wire ),
        .Button_in( Button ),
        .Button_out( BD_wire )
    );

    Switch DS (
        .Button_in( BD_wire ),
        .Switch_out( DS_wire )
    );

    Display(
        .CLK_in( CD_wire ),
        .Data( DS_wire ? Result_wire : Switch ),
        .segment( Segment ),
        .position( Position )
    );
```

```
    Multiply(
        .CLK_in( CD_wire ),
        .Src1( Switch[15 : 8] ),
        .Src2( Switch[7 : 0] ),
        .Result( Result_wire )
    );

    assign Light = DS_wire ? 2'b10 : 2'b01;

endmodule
```

四、实验结果

实验结果已在实验课上通过检查。

五、实验总结

由于经历过从零开始的MIPS CPU编写，吸取过非常多的经验教训，乘法器实验完成地非常顺利。总计使用一个小时，从零开始完成代码编写、约束填写、综合实现、烧板，一次通过，没有错误。

我认为所有实验的关键，是要搞明白实验原理。如果能有比较长时间的思考，提前预想将要经历的步骤，动手实现就越简单、越没有犯错误的余地。