LAB 5

学号:2020K8009915008

姓名:林孟颖 箱子号:79

一.实验任务概览

1. 理解Loongarch指令级的基本指令实现；
2. 分别完成5条、20条指令单周期CPU设计与验证；
3. 熟悉

二.实验设计

（一）总体设计思路

（二）重要模块1设计

1.工作原理

2.接口定义

3. 功能描述

三.实验过程

（一）实验流水

- 2022.09.07 14:40-15:10 完成算数逻辑运算指令的数据通路
- 2022.09.07 19:00-24:00 完成华莱士树乘法器的设计
- 2022.09.09 8:00-9:20　完成乘法器流水切割与验证
- 2022.09.09 19:00-21:00 完成除法器模块设计
- 2022.09.09 21:10-24:00 将乘除法器加入CPU
- 2022.09.10 8:20-

（二）实践任务10：算数逻辑运算指令和乘除法运算指令添加

1. 算数逻辑运算指令

错误1：数据通路未补充完整

（1）错误现象

```
        [ 592000 ns] Test is running, debug_wb_pc = 0x1c03eb9c
        [ 602000 ns] Test is running, debug_wb_pc = 0x1c03fa88
----[ 603615 ns] Number 8'd20 Functional Test Point PASS!!!
--------------------------------------------------------------
[ 603967 ns] Error!!!
    reference: PC = 0x1c035e64, wb_rf_wnum = 0x0d, wb_rf_wdata = 0xe0c13e64
    mycpu    : PC = 0x1c035e64, wb_rf_wnum = 0x0d, wb_rf_wdata = 0x38044fc4
--------------------------------------------------------------
$finish called at time : 604007 ns : File "D:/Desktop/cdp_ede_local/mycpu_env_10/soc_verify/soc_bram/testbench/mycpu_tb.
```

（2）分析定位过程

查看pc对应反汇编代码：

```
1c035e60:   54000c00    bl  12(0xc) # 1c035e6c <n21_pcaddu12i_test+0x1c>
1c035e64:   1d897bcd    pcaddu12i   $r13,-242722(0xc4bde)
1c035e68:   50000c00    b   12(0xc) # 1c035e74 <n21_pcaddu12i_test+0x24>
```

得知是 `pcaddu12i` 指令处理不当，查看其来源操作数定义：

```
    assign src1_is_pc    = inst_jirl | inst_bl | inst_pcaddul2i;

    assign src2_is_imm   = inst_slli_w |
                           inst_srli_w |
                           inst_srai_w |
                           inst_addi_w |
                           inst_ld_w   |
                           inst_st_w   |
                           inst_lu12i_w|
                           inst_jirl   |
                           inst_bl     ;
```

（3）错误原因

忘记在该指令时设置 `src2_is_imm` ，同理还有其他相关补充指令：

```
assign src2_is_imm   = inst_slli_w |
                       inst_srli_w |
                       inst_srai_w |
                       inst_addi_w |
                       inst_ld_w   |
                       inst_st_w   |
                       inst_lu12i_w|
                       inst_jirl   |
                       inst_bl     |
                       inst_pcaddul2i|
                       inst_andi   |
                       inst_ori    |
                       inst_xori   |
                       inst_slti   |
                       inst_sltui;
```

（4）修正效果

已经通过前三十个测试点，来到div指令的测试用例：

```
        [ 802000 ns] Test is running, debug_wb_pc = 0x1c02e7f8
----[ 805065 ns] Number 8'd29 Functional Test Point PASS!!!
--------------------------------------------------------------
[ 805387 ns] Error!!!
    reference: PC = 0x1c0496d0, wb_rf_wnum = 0x0f, wb_rf_wdata = 0x00000002
    mycpu    : PC = 0x1c0496d0, wb_rf_wnum = 0x0f, wb_rf_wdata = 0x00000000
--------------------------------------------------------------
$finish called at time : 805427 ns : File "D:/Desktop/cdp_ede_local/mycpu_env_10/soc_verify/soc_bram/testbench/mycpu_tb.
run: Time (s): cpu = 00:00:26 ; elapsed = 00:00:36 . Memory (MB): peak = 893.785 ; gain = 16.855
```

```
04
05   1c0496b8 <n30_div_w_test>:
06   n30_div_w_test():
07   1c0496b8:   028006f7    addi.w  $r23,$r23,1(0x1)
08   1c0496bc:   00150019    move    $r25,$r0
09   1c0496c0:   14ad7dac    lu12i.w $r12,355309(0x56bed)
10   1c0496c4:   03be918c    ori $r12,$r12,0xfa4
11   1c0496c8:   1441062d    lu12i.w $r13,133169(0x20831)
12   1c0496cc:   039001ad    ori $r13,$r13,0x400
13   1c0496d0:   0020358f    div.w   $r15,$r12,$r13
14   1c0496d4:   03800810    ori $r16,$r0,0x2
15   1c0496d8:   5c0d3a0f    bne $r16,$r15,3384(0xd38) # 1c04a410 <ins
16   1c0496dc:   15fb4bcc    lu12i.w $r12,-9634(0xfda5e)
```

（5）归纳总结（可选）

此部分译码与实现较为简单，只需注意在实现时分步骤完成，可考虑逐条指令实现或按照阶段实现（拆分为ID、EXE等阶段，分别处理新添加的指令）。

2. 乘法模块设计与验证

(1) 无流水切割的booth二位乘华莱士树乘法器实现

此时暂未引入二级流水，故testbench中无需引入寄存器保存操作数的值，修改testbench如下：

```
`timescale 1ns / 1ps

module mul_tb;

    // Inputs
    reg resetn;
    reg mul_clk;
    reg mul_signed;
    reg [31:0] x;
    reg [31:0] y;

    // Outputs
    wire signed [63:0] result;

    // Instantiate the Unit Under Test (UUT)
    Wallace_Mul uut (
        .mul_clk(mul_clk),
        .resetn(resetn),
        .mul_signed(mul_signed),
        .A(x),
        .B(y),
        .result(result)
    );

    initial begin
        // Initialize Inputs
        resetn = 0;
        mul_signed = 0;
        mul_clk = 0;
        x = 0;
        y = 0;
        #100;
        resetn = 1;
    end
    always #5 mul_clk = ~mul_clk;

//产生随机乘数和有符号控制信号
always @(posedge mul_clk)
begin
```

```
    x          <= $random;
    y          <= $random; //$random为系统任务，产生一个随机的32位有符号数
    mul_signed <= {$random}%2; //加了拼接符，{$random}产生一个非负数，除2取余得到0或
1
end

//参考结果
wire signed [63:0] result_ref;
assign result_ref = ({{32{x[31] & mul_signed}},x} * {{32{y[31] &
mul_signed}},y}) & {64{resetn}};
assign ok         = (result_ref == result);

//打印运算结果
initial begin
    $monitor("x = %d, y = %d, signed = %d, result =
%d,OK=%b",x,y,mul_signed,result,ok);
end

//判断结果是否正确
always @(posedge mul_clk)
begin
    if (!ok)
    begin
        $display("Error: x = %d, y = %d,result = %d, result_ref = %d,
OK=%b",x,y,result,result_ref,ok);
        $finish;
    end
end

endmodule
```

由于在计算机体系结构理论课的作业中已经编写了32位华莱士树乘法器，故此处只需进行验证：

Tcl Console × Messages Log

```
x = 2401039902, y =  112395533, signed = 0, result =    269866159539557766,OK=1
x =  201563416, y = 1756240849, signed = 0, result =    353993904843180184,OK=1
x = 2696946753, y = 2646565947, signed = 0, result =   7137647437362020091,OK=1
x =  703588691, y = 2870567510, signed = 1, result =  -1002191580892420126,OK=1
x = 4050057954, y = 2188632580, signed = 1, result =    515861049327316872,OK=1
x = 3964712152, y =  155077906, signed = 0, result =    614839258424913712,OK=1
x = 2625704505, y = 4068980453, signed = 1, result =    377231428275458813,OK=1
x =  361303851, y = 1083202945, signed = 0, result =    391365395443041195,OK=1
x =  330650919, y = 1356200353, signed = 1, result =    448428893067574407,OK=1
x = 2183281156, y =  741155672, signed = 0, result =   1618151212340116832,OK=1
x =  175018772, y = 2300163090, signed = 0, result =    402571719411525480,OK=1
x = 3635208881, y =  717411669, signed = 1, result =   -473318385641944635,OK=1
INFO: [USF-XSim-96] XSim completed. Design snapshot 'mul_tb_behav' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 1000ns
launch_simulation: Time (s): cpu = 00:00:06 ; elapsed = 00:00:09 . Memory (MB): peak = 918.609 ; gain = 0.000
```

（2）含二级流水的booth二位乘华莱士树乘法器实现

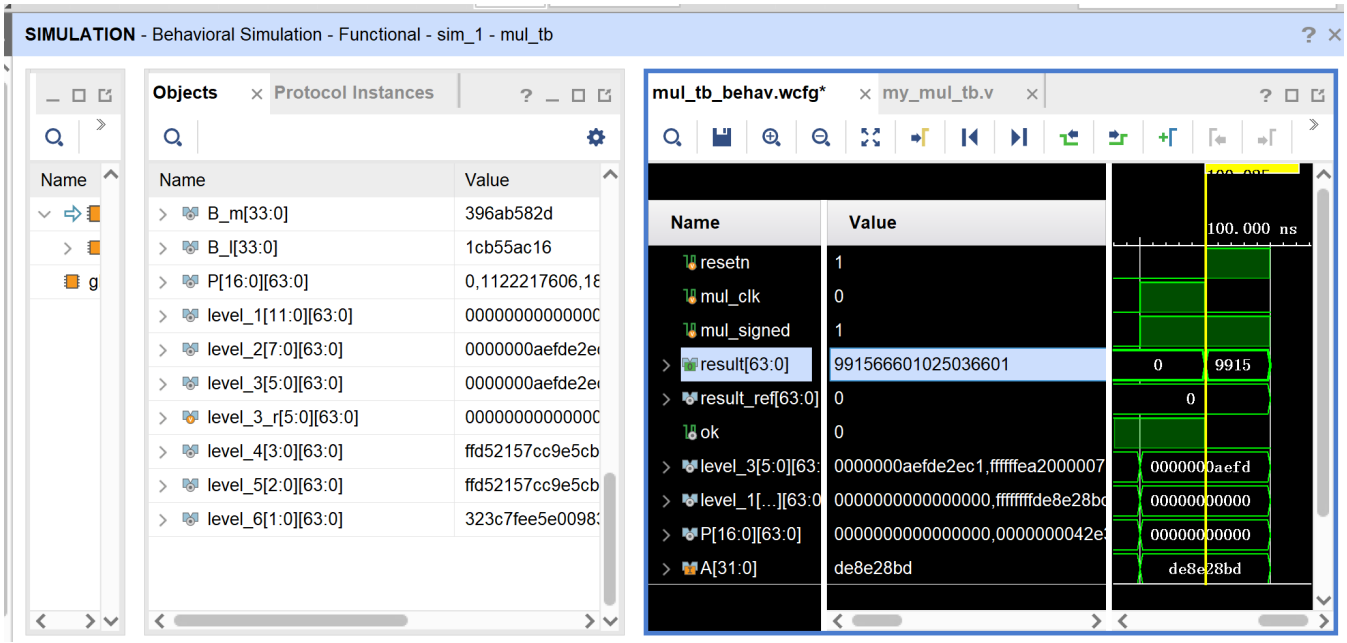
1. 思路分析

现阶段构建的华莱士树一共有六层，每层华莱士树中的加法器是并行的。故考虑在3~4层加法器中做流水线分割，使得两级流水线的操作时长差异较小。

2. Debug过程

- **BUG：切分流水线时忘记使用寄存器**

起初在level 3和4之间做了切分后发现result在resetn一失效后就变化：



- **分析定位过程 & 错误原因**：

查看实现逻辑：

```
    assign level_3[5] = level_2[7];
//----------------------------------------流水级切分-----------------------------
    reg  [63:0] level_3_r [5:0];
    always @(posedge mul_clk) begin
        if(~resetn)
            {level_3_r[0],level_3_r[1],level_3_r[2],
             level_3_r[3],level_3_r[4],level_3_r[5]} <= {6{64'b0}};
        else
            {level_3_r[0],level_3_r[1],level_3_r[2],
             level_3_r[3],level_3_r[4],level_3_r[5]} <= {level_3[0],level_3[1],le
                                                         level_3[3],level_3[4],lev
    end
//----------------------------------------Level 4-------------------------------
    wire [63:0] level_4 [3:0];
    Adder adder4_1 (
        .in1(level_3[0]),
        .in2(level_3[1]),
        .in3(level_3[2]),
        .C(level_4[0]),
        .S(level_4[1])
    );
    Adder adder4_2 (
        .in1(level_3[3]),
        .in2(level_3[4]),
        .in3(level_3[5]),
        .C(level_4[2]),
```

发现切分后高层加法器忘记使用切分过程使用的寄存器。

- 修改效果

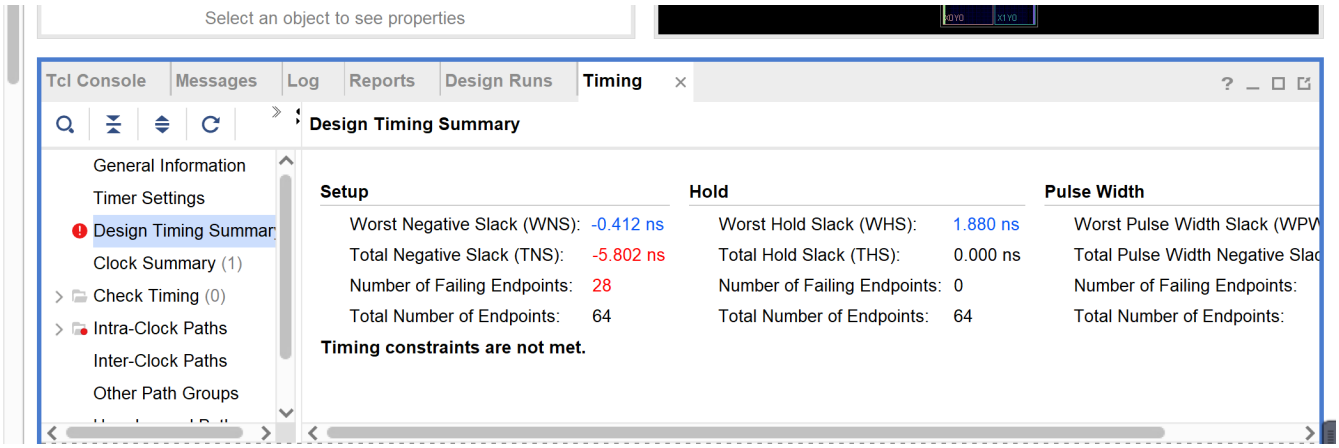
在第四层中使用保存了第三层数据的寄存器后可正常运行：

Tcl Console × Messages Log

```
x =  1153746313, y =  3540625062, signed = 0, result =   4084983110997896406, OK=1
x =  3527370916, y =  4238803193, signed = 0, result =  -3494912972073416828, OK=1
x =   183514901, y =   695240530, signed = 0, result =    127586997034137530, OK=1
x =   292576034, y =    86856970, signed = 0, result =     25412267807856980, OK=1
x =  1809618903, y =   670028111, signed = 1, result =   1212495535206982233, OK=1
x =   455333174, y =  1632970690, signed = 0, result =    743545727326670060, OK=1
x =  1483593648, y =   952476017, signed = 0, result =   1413087368693540016, OK=1
x =  1303016347, y =  1241690516, signed = 0, result =   1617943040262865052, OK=1
x = -1959001578, y = -1932102631, signed = 1, result =    378499210298695178, OK=1
x =  2915309659, y =   840304484, signed = 0, result =   2449747778706210956, OK=1
x =  3214763135, y =  3530088100, signed = 0, result =  -7098346986527358116, OK=1
x =  2792135244, y =  1634154434, signed = 0, result =    4562780189310271896, OK=1
x =  3133644917, y =  1397495206, signed = 0, result =    4379253748813767902, OK=1
x =  2865452629, y =  1825141209, signed = 0, result =    5229855675625288461, OK=1
x =  4111455466, y =  3738280637, signed = 0, result =  -3076969715273939774, OK=1
```

（3）流水vs非流水：综合效果比对

简单加了时钟约束以及输入输出的延迟约束后，无流水分割的乘法器仿真结果如下：



可见总违约时间约为5.802ns，综合的电路时序难以满足时序要求，再使用如下命令查看最差的50条setup timing path:

```
report_design_analysis -max_paths 50 -setup
```

Paths	Requirement	Path Delay	Logic Delay	Net Delay	Clock Skew	Slack	Clock Relationshi
Path #1	10.000	10.387	5.224(51%)	5.163(49%)	0.000	-0.412	Safely Timed
Path #2	10.000	10.381	5.199(51%)	5.182(49%)	0.000	-0.406	Safely Timed
Path #3	10.000	10.367	5.303(52%)	5.064(48%)	0.000	-0.392	Safely Timed
Path #4	10.000	10.329	5.166(51%)	5.163(49%)	0.000	-0.354	Safely Timed
Path #5	10.000	10.321	5.161(51%)	5.160(49%)	0.000	-0.346	Safely Timed
Path #6	10.000	10.309	5.245(51%)	5.064(49%)	0.000	-0.334	Safely Timed
Path #7	10.000	10.293	5.227(51%)	5.066(49%)	0.000	-0.318	Safely Timed

Slack	Clock Relationship	Logic Levels	Routes	Logical Path
-0.412	Safely Timed	19	0	IBUF LUT1 CARRY4 CARRY4 CARRY4 CARRY4 CARRY4 CARRY4 LUT6 LUT3 LUT3 LUT6 LUT5 LUT
-0.406	Safely Timed	18	0	IBUF LUT1 CARRY4 CARRY4 CARRY4 CARRY4 CARRY4 CARRY4 LUT6 LUT3 LUT3 LUT6 LUT5 LUT
-0.392	Safely Timed	19	0	IBUF LUT1 CARRY4 CARRY4 CARRY4 CARRY4 CARRY4 CARRY4 LUT6 LUT3 LUT3 LUT6 LUT5 LUT
-0.354	Safely Timed	18	0	IBUF LUT1 CARRY4 CARRY4 CARRY4 CARRY4 CARRY4 CARRY4 LUT6 LUT3 LUT3 LUT6 LUT5 LUT
-0.346	Safely Timed	19	0	IBUF LUT1 CARRY4 CARRY4 LUT6 LUT4 LUT5 LUT5 LUT3 LUT3 CARRY4 CARRY4 CARRY4 CARRY
-0.334	Safely Timed	18	0	IBUF LUT1 CARRY4 CARRY4 CARRY4 CARRY4 CARRY4 CARRY4 LUT6 LUT3 LUT3 LUT6 LUT5 LUT

Path_delay约为10s，且logic delay和net delay基本各占一半，可以看到许多条path的逻辑级数高达18、19。

> 逻辑级数过多（Logic Levels）：一般可以修改代码，增加寄存器以降低逻辑级数。

修改为二级流水后再次查看时序报告：

Design Timing Summary

Setup		Hold		Pulse Wi
Worst Negative Slack (WNS):	5.451 ns	Worst Hold Slack (WHS):	1.880 ns	Wors
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Numb
Total Number of Endpoints:	64	Total Number of Endpoints:	64	Total

All user specified timing constraints are met.

综合的时序良好，未出现违约的情况。再查看Path Delay：

Paths	Requirement	Path Delay	Logic Delay	Net Delay	Clock Skew	Slack	Clock Relationship	Logic Levels	Routes
Path #1	10.000	4.524	3.357(75%)	1.167(25%)	0.000	5.451	Safely Timed	3	0
Path #2	10.000	4.524	3.357(75%)	1.167(25%)	0.000	5.451	Safely Timed	3	0
Path #3	10.000	4.524	3.357(75%)	1.167(25%)	0.000	5.451	Safely Timed	3	0
Path #4	10.000	4.524	3.357(75%)	1.167(25%)	0.000	5.451	Safely Timed	3	0
Path #5	10.000	4.524	3.357(75%)	1.167(25%)	0.000	5.451	Safely Timed	3	0

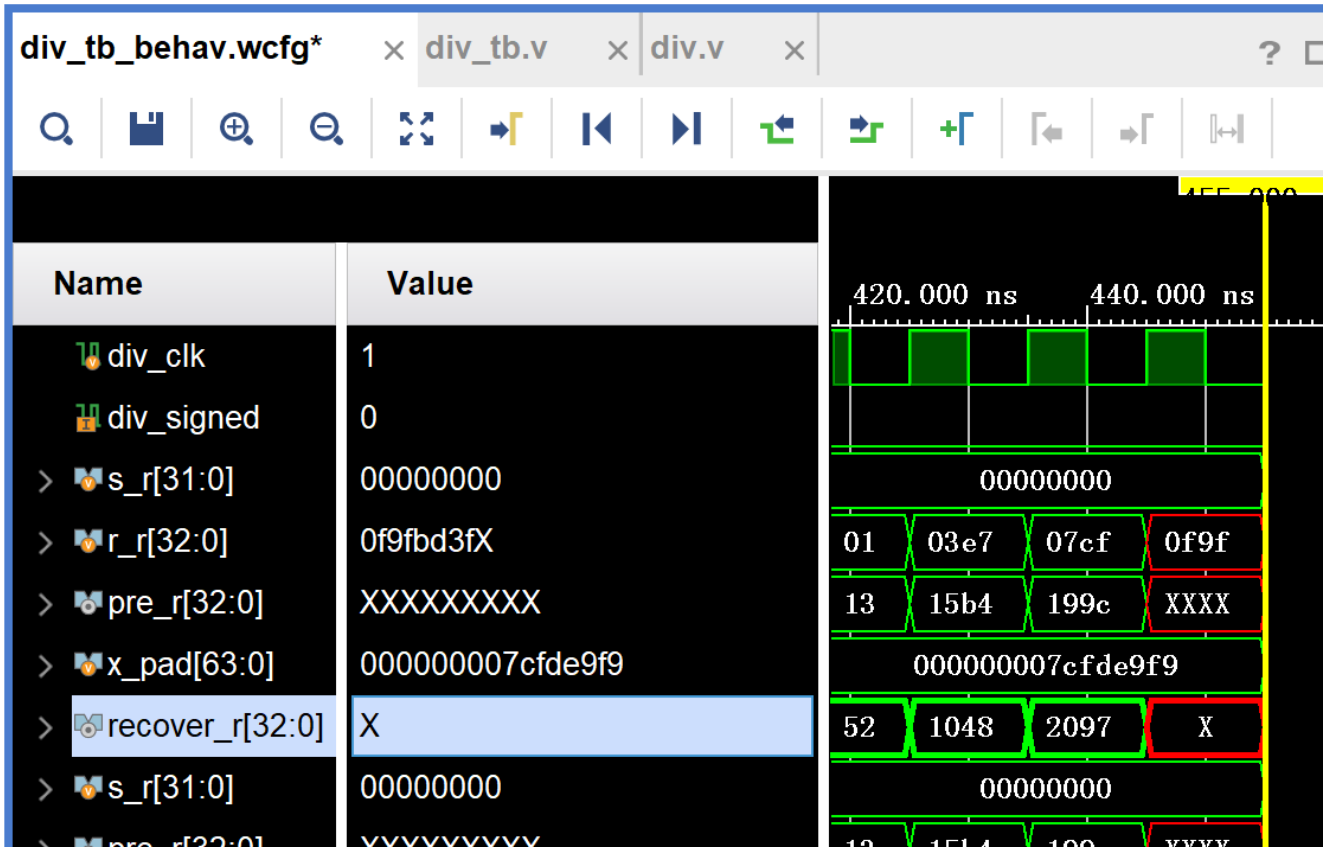
可见总的Path Delay减少为原来的一半以下，且逻辑级数缩减为3。

3.除法模块设计与验证

此部分使用实验所提供的testbench测试验证设计的除法器模块。

(1) 错误：除法器余数末位未定义

（1）错误现象



（2）分析定位过程

查看实现代码：

```
always @(posedge div_clk) begin
    if(~resetn)
        r_r <= 33'b0;
    if(div & ~complete) begin
        if(~|counter)   //余数初始化
            r_r <= {32'b0, abs_x[31]};
        else
            r_r <= {recover_r,x_pad[31 - counter]};
    end
end
```

（3）错误原因

注意到最后一次循环内counter值为32，`x_pad[31 - counter]`会导致数组下标越界，应在最后一次循环时将当前余数赋值为recover_r。

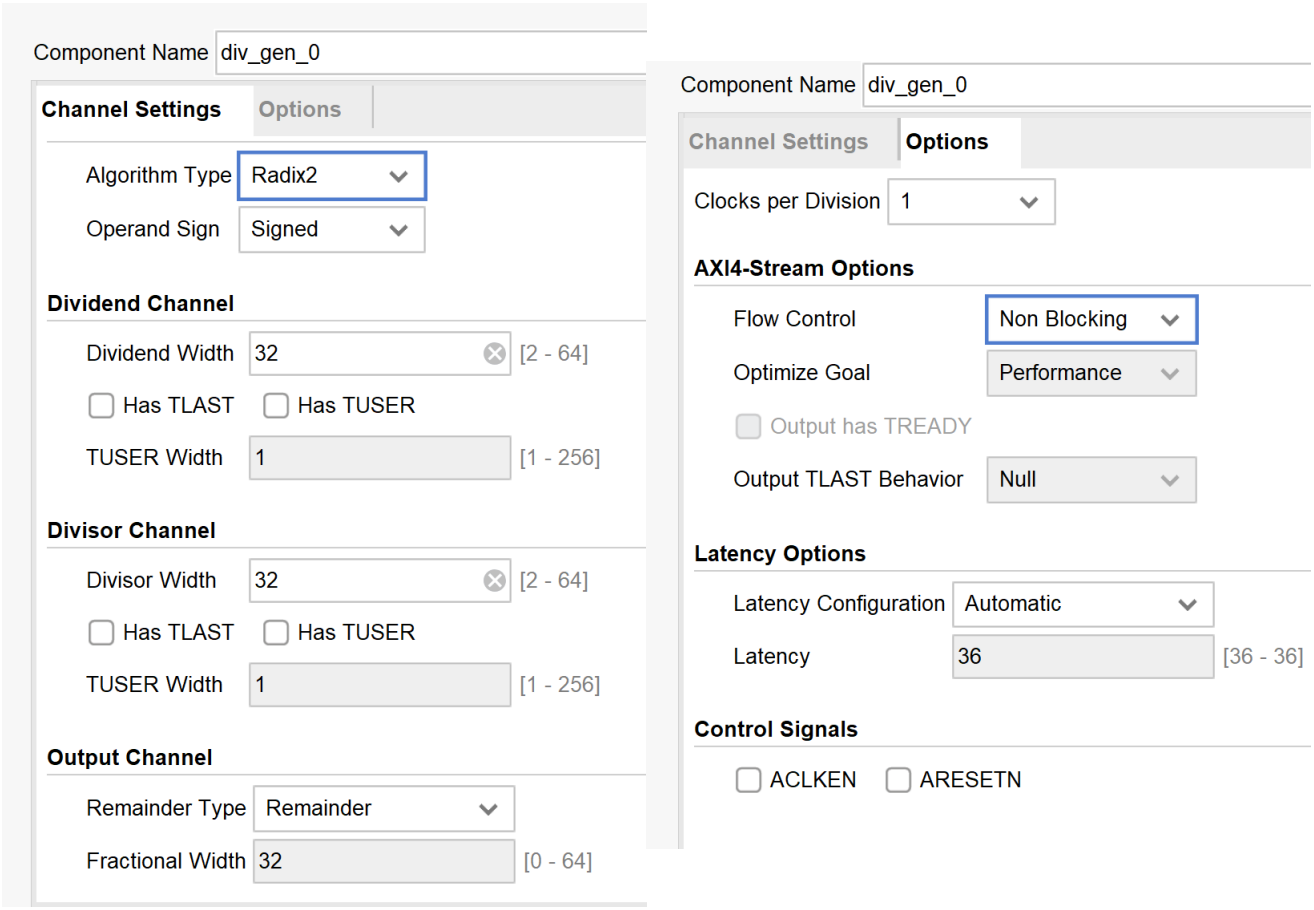
```
    always @(posedge div_clk) begin
        if(~resetn)
            r_r <= 33'b0;
        if(div & ~complete) begin
            if(~|counter)   //余数初始化
                r_r <= {32'b0, abs_x[31]};
            else
                r_r <=  (counter == 32) ? recover_r : {recover_r, x_pad[31 -
counter]};
        end
    end
```

（4）修正效果

```
[time@          5333775000]: x=-1071997312, y= -864333672, signed=1, s=          1, r= -207663640, s_OK=1, r_OK=1
[time@          5334145000]: x= 3672468149, y= 1486512561, signed=0, s=          2, r=  699443027, s_OK=1, r_OK=1
[time@          5334505000]: x=-2104734715, y= -175289621, signed=1, s=         12, r=   -1259263, s_OK=1, r_OK=1
[time@          5334865000]: x= 1805558231, y= -615294794, signed=1, s=         -2, r=  574968643, s_OK=1, r_OK=1
[time@          5335235000]: x= 2768963146, y= 3186633851, signed=0, s=          0, r= 2768963146, s_OK=1, r_OK=1
[time@          5335605000]: x= -504043325, y= 1208929168, signed=1, s=          0, r= -504043325, s_OK=1, r_OK=1
[time@          5335965000]: x=  359587626, y= 2822369872, signed=0, s=          0, r=  359587626, s_OK=1, r_OK=1
[time@          5336345000]: x= 2090341881, y=-1833228251, signed=1, s=         -1, r=  257113630, s_OK=1, r_OK=1
[time@          5336695000]: x= -565746500, y=   32087811, signed=1, s=        -17, r=  -20253713, s_OK=1, r_OK=1
[time@          5337075000]: x= 1288050073, y=-1665728455, signed=1, s=          0, r= 1288050073, s_OK=1, r_OK=1
[time@          5337455000]: x= 2265332238, y= 2862366293, signed=0, s=          0, r= 2265332238, s_OK=1, r_OK=1
```

（2）和Vivado IP核性能对比

按照讲义所给设定使用Radix2算法、选择输入输出位宽为32位、选择余数类型为整数型、采取Non Blocking输出，所提供的IP核Latency为36，大于自设计的除法器的34拍：



(3) 归纳总结

4. 将乘除法模块加入系统

(1) 思路分析

若将乘除法模块内置到原本alu中，需要给其额外增加引线、调整alu_op的位宽。

- 增加引线
  - 需要引入时钟信号为乘除法运算打拍
  - 需要引入resetn信号初始化乘法器和除法器
  - 需要引入乘除法结束的握手信号告知CPU运算是否结束
- alu_op位宽+6：
  新增有/无符号的乘法/除法/取模运算，共六种新操作。

(2) Debug过程

错误1：valid与allowin信号处理不当

（1）错误现象

写回寄存器有误，且PC提前更新

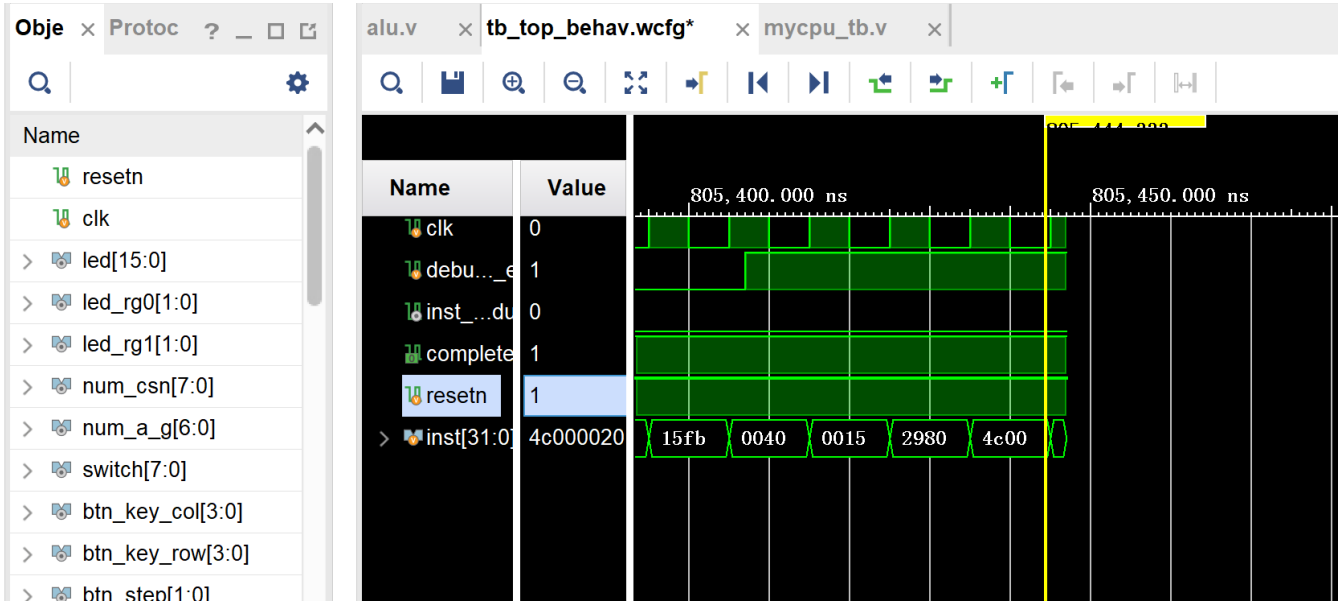
```
----[ 805065 ns] Number 8'd29 Functional Test Point PASS!!!
----------------------------------------------------------------
[ 805407 ns] Error!!!
    reference: PC = 0x1c0496d0, wb_rf_wnum = 0x0f, wb_rf_wdata = 0x00000002
    mycpu    : PC = 0x1c0496d4, wb_rf_wnum = 0x10, wb_rf_wdata = 0x00000002
----------------------------------------------------------------
----------------------------------------------------------------
```

（2）分析定位过程

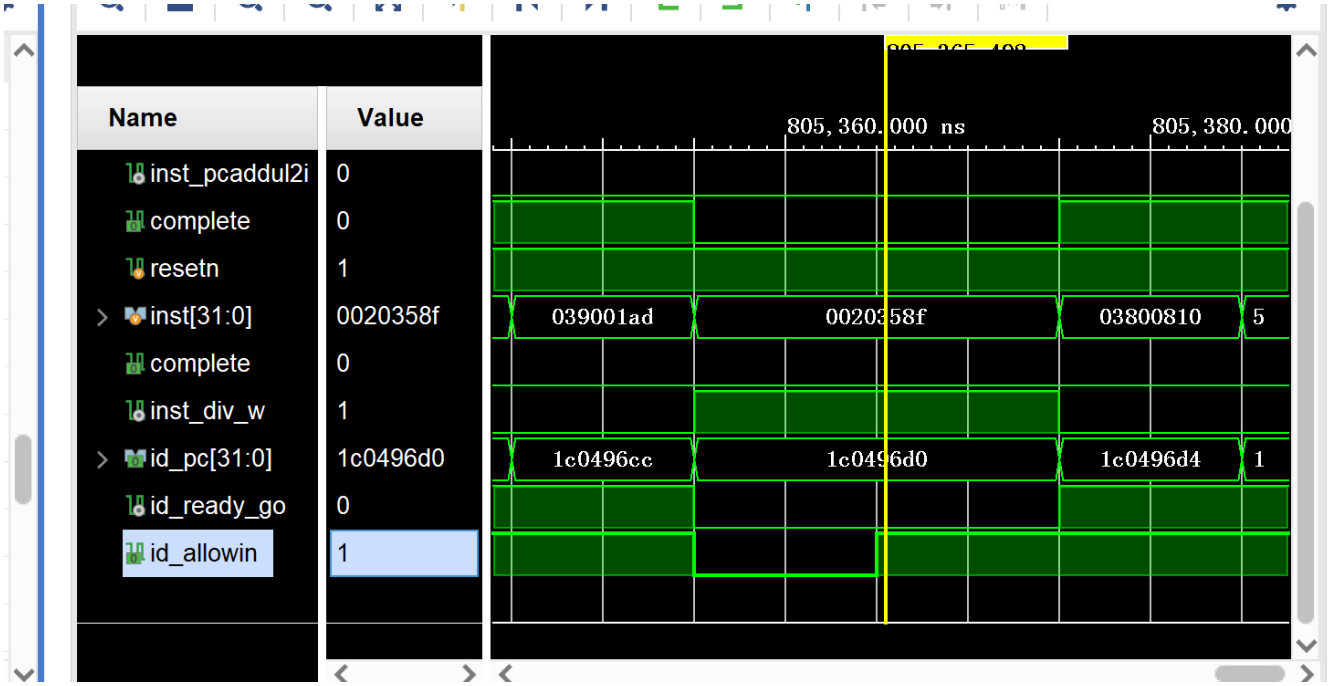
查看当前指令：

```
1c01bbf4:   0397b1ad    ori $r13,$r13,0x5ec
1c01bbf8:   0021358f    div.wu  $r15,$r12,$r13
1c01bbfc:   00150010    move    $r16,$r0
```

得知是div.w指令。再查看波形：



发现complete信号阻塞流水线拍数不足，导致PC和指令都提前更新，再查看控制流水的信号：



```
    assign id_ready_go      = alu_complete;
    assign id_allowin       = ~id_valid | id_ready_go & exe_allowin;
    assign id_to_exe_valid  = id_valid & id_ready_go;
    always @(posedge clk) begin
        if(~resetn)
            id_valid <= 1'b0;
        else
            // 有效条件：if向id输入有效、非转移指令、id允许接收数据
            id_valid <= if_to_id_valid & ~br_taken & id_allowin;
    end
```

（3）错误原因

上述设计中 `id_allowin` 和 `id_valid` 相关，前者会在后者失效时拉高，而后者在前者失效的下一个周期也会失效，但实际上当前流水级不允许流入时（`id_allowin=0`）并不意味着当前阶段锁存的数据无效（`id_valid=0`），故修改如下，使得 `id_allowin` 变为 `id_valid` 更新的条件，而非直接赋值：

```
    always @(posedge clk) begin
        if(~resetn | br_taken)
            id_valid <= 1'b0;
        else if(id_allowin)
            id_valid <= if_to_id_valid;
    end
```

（4）修正效果

通关至下一个bug：

```
        [ 952000 ns] Test is running, debug_wb_pc = 0x1c05b70c
----[ 953735 ns] Number 8'd33 Functional Test Point PASS!!!
----------------------------------------------------------------
[ 954057 ns] Error!!!
    reference: PC = 0x1c00f2b0, wb_rf_wnum = 0x0f, wb_rf_wdata = 0x40f0c088
    mycpu    : PC = 0x1c00f2b0, wb_rf_wnum = 0x0f, wb_rf_wdata = 0x00000000
----------------------------------------------------------------
$finish called at time : 954097 ns : File "D:/Desktop/cdp_ede_local/mycpu_env_10/soc_verify/soc_bram/testbe
```

（5）归纳总结（可选）

错误2：错误简介命名

（1）错误现象

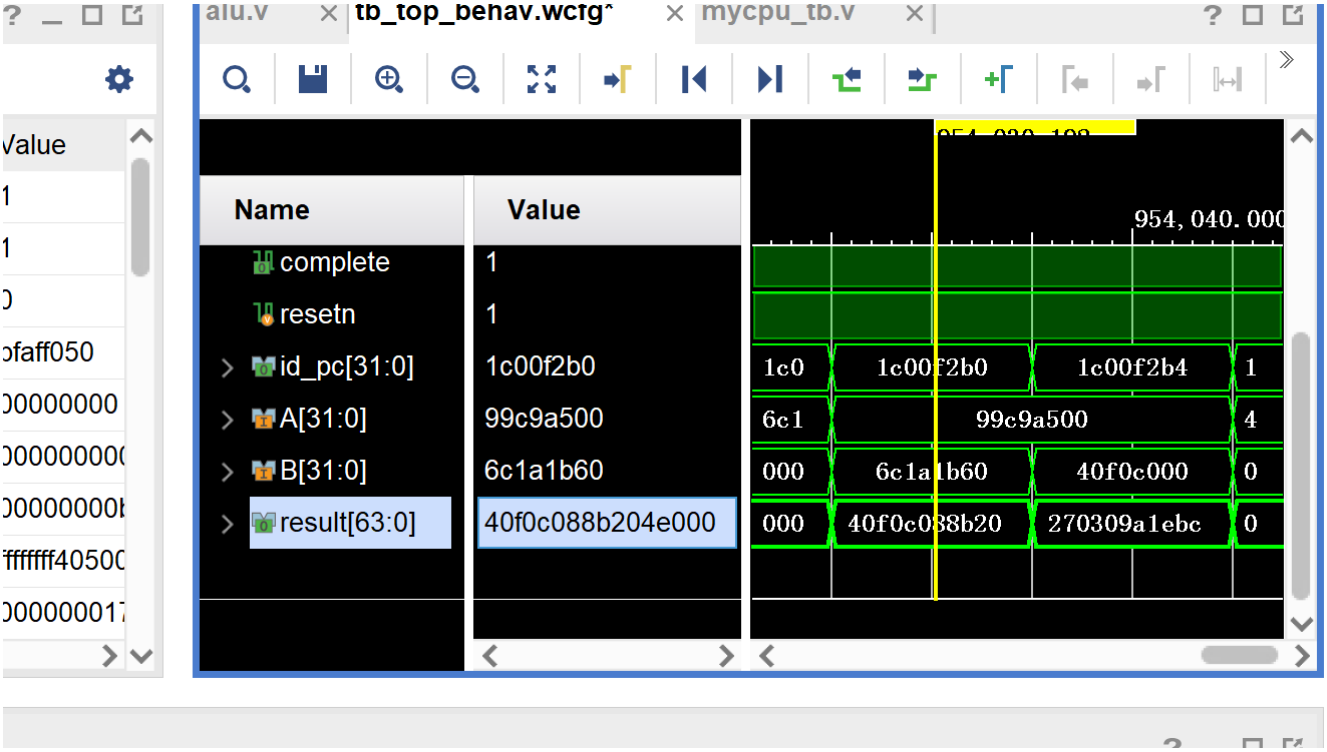
如上，写回数据不对。

（2）分析定位过程

查看对应指令：

```
1c00f2ac:   03ad81ad    ori $r13,$r13,0xb60
1c00f2b0:   001d358f    mulh.wu $r15,$r12,$r13
1c00f2b4:   1481e190    lu12i.w $r16,265996(0x40f0c)
```

确定为mul指令，查看对应pc下乘法器中数据：



发现计算结果正确，但未能成功传递给CPU，查看CPU传递给alu的操作码：

```
    assign alu_op[12] = inst_mul_w ;
    assign alu_op[13] = inst_mulh_w;
    assign alu_op[14] = inst_mulh_w;
```

ALU内部操作码译码：

```
assign op_mul  = alu_op[12];
assign op_mulh = alu_op[13];
assign op_mulhu= alu_op[14];
```

（3）错误原因

CPU传递给ALU操作码有误，应改为：

```
    assign alu_op[14] = inst_mulh_wu;
```

（4）修正效果

PASS!

```
       [1012000 ns] Test is running, debug_wb_pc = 0x1c050428
----[1016825 ns] Number 8'd35 Functional Test Point PASS!!!
       [1022000 ns] Test is running, debug_wb_pc = 0x1c03a69c
       [1032000 ns] Test is running, debug_wb_pc = 0x1c03a9b4
       [1042000 ns] Test is running, debug_wb_pc = 0x1c03acb4
       [1052000 ns] Test is running, debug_wb_pc = 0x1c03afd0
       [1062000 ns] Test is running, debug_wb_pc = 0x1c03b2c8
       [1072000 ns] Test is running, debug_wb_pc = 0x1c03b5d4
       [1082000 ns] Test is running, debug_wb_pc = 0x1c03b8e8
----[1089945 ns] Number 8'd36 Functional Test Point PASS!!!
================================================================
Test end!
----PASS!!!
$finish called at time : 1090385 ns : File "D:/Desktop/cdp_ede_local/mycpu_env_10/soc_verify/soc_bram/testbench/mycpu_tb.v" Line 270
run: Time (s): cpu = 00:00:51 ; elapsed = 00:00:59 . Memory (MB): peak = 1078.051 ; gain = 0.000
```

（5）归纳总结（可选）

（三）实践任务11：转移指令&访存指令的添加

（1）转移指令添加

此部分设计较为简单，blt和bge类指令可以直接利用alu的计算结果做出判断，补充完数据通路后一遍过：

```
----------------------------------------------------------------
[1239327 ns] Error!!!
    reference: PC = 0x1c02ef4c, wb_rf_wnum = 0x10, wb_rf_wdata = 0x0000000b
    mycpu    : PC = 0x1c02ef4c, wb_rf_wnum = 0x10, wb_rf_wdata = 0x00000000
----------------------------------------------------------------
$finish called at time : 1239367 ns : File "D:/Desktop/cdp_ede_local/mycpu_env_11/soc_verify/soc_bra
```

可以看到出错的地方已经到ld.w指令：

```
1c02ef44:   299e8084    st.w    $r4,$r4,1952(0x7a0)
1c02ef48:   299e80a5    st.w    $r5,$r5,1952(0x7a0)
1c02ef4c:   281e8590    ld.b    $r16,$r12,1953(0x7a1)
1c02ef50:   289e8085    ld.w    $r5,$r4,1952(0x7a0)
1c02ef54:   289e80a4    ld.w    $r4,$r5,1952(0x7a0)
```

继续添加访存指令的数据通路。

（2）访存指令的添加

添加访存指令需注意其读寄存器时要将源寄存器改为rd寄存器，以及补充alu加操作以计算访存地址，且需留心写内存、写寄存器时的位宽。同时注意到讲义建议将数据通路放在访存阶段，即需要将st和ld指令的信息从ID阶段传至EXE阶段（同时因为上次实验设计时已经在EXE模块内完成了数据的预取，故无需再传递至MEM模块）。

- ld类指令：
- st类指令：可考虑将目前传输的1位的`mem_we`扩充为4位，作为写入时的掩码。

（3）错误原因

（4）修正效果

（5）归纳总结（可选）

错误1：位宽定义有误

（1）错误现象

写回寄存器数据出错：

```
============================================================
Test begin!
----[  12095 ns] Number 8'd01 Functional Test Point PASS!!!
------------------------------------------------------------
[  12197 ns] Error!!!
    reference: PC = 0x1c00f2b4, wb_rf_wnum = 0x0e, wb_rf_wdata = 0x0000aaaa
    mycpu    : PC = 0x1c00f2b4, wb_rf_wnum = 0x0e, wb_rf_wdata = 0xbfaff090
------------------------------------------------------------
$finish called at time : 12237 ns : File "D:/Desktop/adp_edo_local/mycpu_env_11/sco_verify/sco
```

（2）分析定位过程

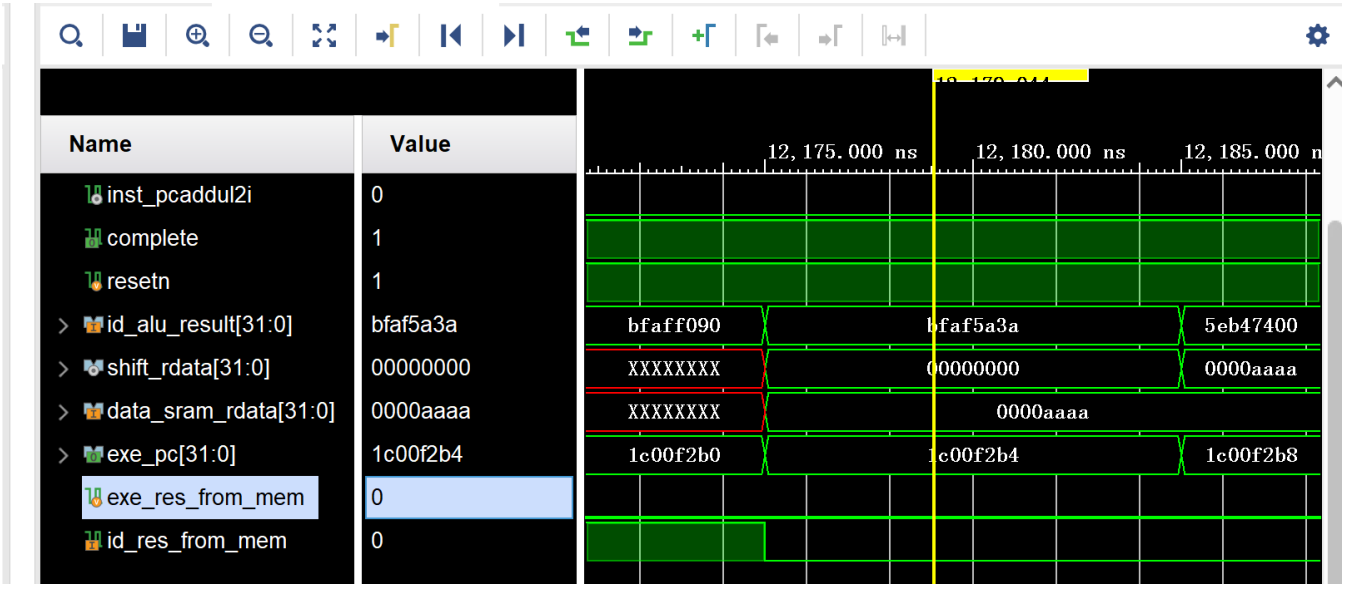
查看指令类型：

```
1c00f2ac:   1400016d    lu12i.w $r13,11(0xb)
1c00f2b0:   02aaa9ad    addi.w  $r13,$r13,-1366(0xaaa)
1c00f2b4:   2880018e    ld.w    $r14,$r12,0
1c00f2b8:   0015b5ce    xor $r14,$r14,$r13
```

再查看现阶段写回数据的实现：

```
    assign exe_mem_result[ 7: 0]   = shift_rdata[ 7: 0];
    assign exe_mem_result[15: 8]   = op_ld_b  ? {8{shift_rdata[7]}} :
                                     op_ld_bu ? 8'b0                :
                                     shift_rdata[15: 8];
    assign exe_mem_result[31:16]   = op_ld_b  ? {16{shift_rdata[7]}} :
                                     op_ld_hu ? {16{shift_rdata[15]}}:
                                     (op_ld_bu | op_ld_hu) ? 16'b0   :
                                     shift_rdata[31:16];
```

再查看`shift_data`，发现其与金标准一致，但写回数据错选择为alu的计算结果：



同时注意到exe阶段未能正确接受id阶段的res_from_mem信号。

（3）错误原因

如下：修改mem_we的位宽时出错，应为4位：

```
    reg         exe_res_from_mem;
    reg  [4 :0] exe_mem_we      ;
    reg         exe_rf_we       ;
```

（4）修正效果

（5）归纳总结（可选）

错误2：组合逻辑环

（1）错误现象

波形停止，console不打印debug信息。

（2）分析定位过程

因为只修改了EXE和ID模块，可以尝试在二者中间寻找组合逻辑环：

首先看到数据冲突处理时用到了 `exe_rf_wdata` ：

```
    // 数据冲突时处理有先后顺序，以最后一次更新为准
    assign rj_value  =  conflict_r1_exe ? exe_rf_wdata:
                        conflict_r1_mem ? mem_rf_wdata:
                        conflict_r1_wb  ? wb_rf_wdata : rf_rdata1;
    assign rkd_value =  conflict_r2_exe ? exe_rf_wdata:
                        conflict_r2_mem ? mem_rf_wdata:
                        conflict_r2_wb  ? wb_rf_wdata : rf_rdata2;
```

再看到`exe_rf_wdata`的定义涉及了`id_alu_result`，而alu计算结果又与`rj_value`相关。

```
    assign shift_rdata   = {24'b0, data_sram_rdata} >> {id_alu_result[1:0],
3'b0};
```

（3）错误原因

存在组合逻辑环。

（4）修正效果

波形可正常往后跑，console继续打印debug信息。

```
        [ 212000 ns] Test is running, debug_wb_pc = 0x1c060d80
        [ 222000 ns] Test is running, debug_wb_pc = 0x1c061d20
----[ 225585 ns] Number 8'd10 Functional Test Point PASS!!!
        [ 232000 ns] Test is running, debug_wb_pc = 0x1c06bc8c
        [ 242000 ns] Test is running, debug_wb_pc = 0x1c06cc2c
----[ 243955 ns] Number 8'd11 Functional Test Point PASS!!!
        [ 252000 ns] Test is running, debug_wb_pc = 0x1c0460f8
        [ 262000 ns] Test is running, debug_wb_pc = 0x1c047098
----[ 262325 ns] Number 8'd12 Functional Test Point PASS!!!
        [ 272000 ns] Test is running, debug_wb_pc = 0x1c04fd7c
----[ 280695 ns] Number 8'd13 Functional Test Point PASS!!!
```

（5）归纳总结（可选）

错误3：脑子不好使

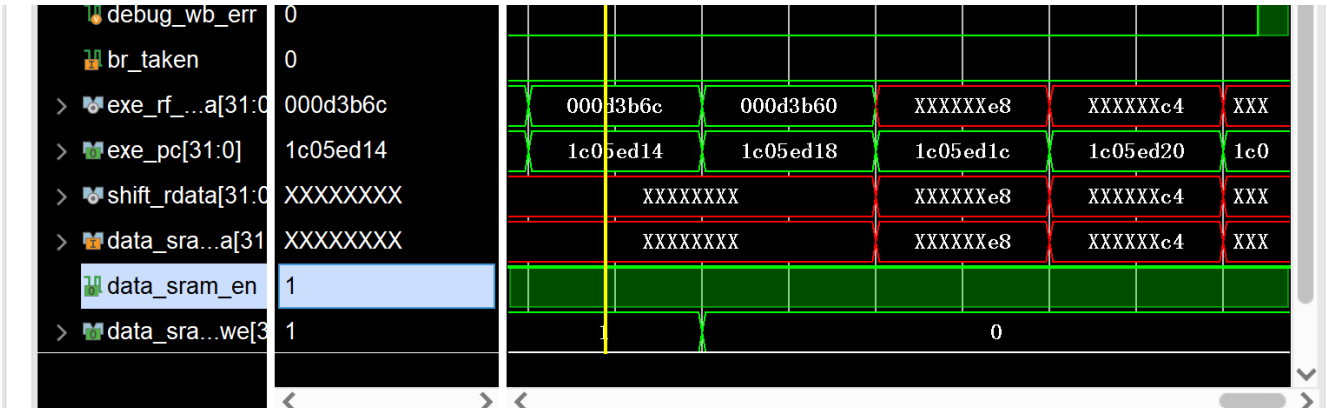
（1）错误现象

写回数据未定义

```
----[ 280695 ns] Number 8'd13 Functional Test Point PASS!!!
------------------------------------------------------------------
[ 281087 ns] Error!!!
    reference: PC = 0x1c05ed1c, wb_rf_wnum = 0x0a, wb_rf_wdata = 0xc822c7e8
    mycpu    : PC = 0x1c05ed1c, wb_rf_wnum = 0x0a, wb_rf_wdata = 0xxxxxxxe8
------------------------------------------------------------------
$finish called at time : 281127 ns : File "D:/Desktop/cdp_ede_local/mycpu_env_11/soc_verify/soc_bram/te
run: Time (s): cpu = 00:00:13 ; elapsed = 00:00:17 . Memory (MB): peak = 913.238 ; gain = 3.469
```

（2）分析定位过程

查看指令得知是ld.w指令，发现读出数据始终无效，查看前面写内存的操作，发现写有效信号始终为1或0。



查看写有效信号 `data_sram_we`，不知什么时候手抽填了一个逐位或：

```
//          op_la_bu,op_la_h, op_la_hu, op_la_wj = ia_inst_z
assign data_sram_en     = id_res_from_mem || (|id_mem_we);
assign data_sram_we     = |id_mem_we;
assign data_sram_addr   = id_alu_result;
assign data_sram_wdata[ 7: 0]   = id_rkd_value[ 7: 0];
```

（3）错误原因

写有效信号出错。

（4）修正效果

```
        [1182000 ns] Test is running, debug_wb_pc = 0x1c072f7c
        [1192000 ns] Test is running, debug_wb_pc = 0x1c073ff4
----[1201895 ns] Number 8'd39 Functional Test Point PASS!!!
        [1202000 ns] Test is running, debug_wb_pc = 0x1c00f2b4
        [1212000 ns] Test is running, debug_wb_pc = 0x1c0221bc
        [1222000 ns] Test is running, debug_wb_pc = 0x1c0232b8
        [1232000 ns] Test is running, debug_wb_pc = 0x1c02439c
----[1238935 ns] Number 8'd40 Functional Test Point PASS!!!
--------------------------------------------------------------
[1239327 ns] Error!!!
    reference: PC = 0x1c02ef4c, wb_rf_wnum = 0x10, wb_rf_wdata = 0x0000000b
    mycpu    : PC = 0x1c02ef4c, wb_rf_wnum = 0x10, wb_rf_wdata = 0xxxxxxxxx
--------------------------------------------------------------
$finish called at time : 1239367 ns : File "D:/Desktop/cdp_ede_local/mycpu_env_11/soc_verify/soc_bi
```

错误4：数据通路未补充完整

（1）错误现象

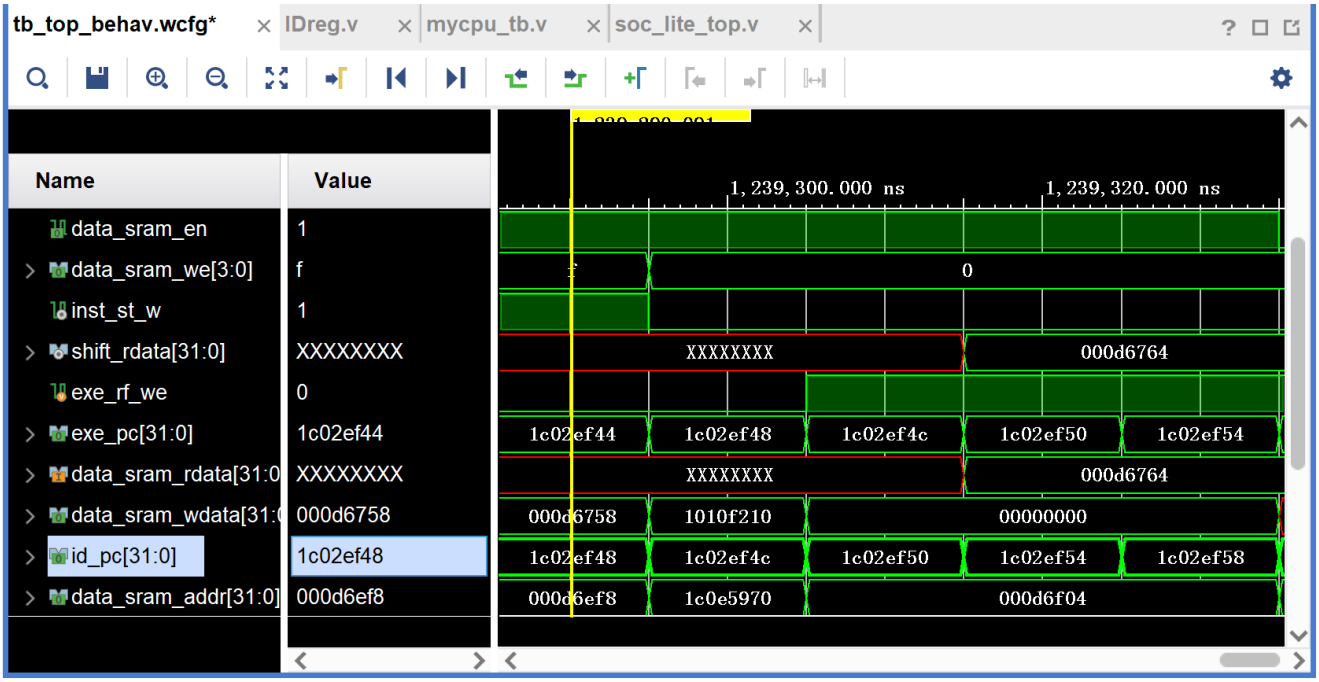
如上，写回数据未定义。

（2）分析定位过程

查看反汇编代码：

```
1c02ef44:   299e8084    st.w    $r4,$r4,1952(0x7a0)
1c02ef48:   299e80a5    st.w    $r5,$r5,1952(0x7a0)
1c02ef4c:   281e8590    ld.b    $r16,$r12,1953(0x7a1)
1c02ef50:   289e8085    ld.w    $r5,$r4,1952(0x7a0)
1c02ef54:   289e80a4    ld.w    $r4,$r5,1952(0x7a0)
```

确定是`ld.b`指令，此时读出数据无效：



猜测是地址未正确给出，考虑到其数据通路与`inst_st_w`类似，故与之比对。

（3）错误原因

```
    assign src2_is_imm   = inst_slli_w |
                           inst_srli_w |
                           inst_srai_w |
                           inst_addi_w |
                           inst_ld_w   |
                           inst_st_w   |
                           inst_lu12i_w|
                           inst_jirl   |
                           inst_bl     |
                           inst_pcaddul2i|
                           inst_andi   |
                           inst_ori    |
                           inst_xori   |
                           inst_slti   |
                           inst_sltui;
```

如上述，忘记指定其操作数2来源。

（4）修正效果

此时读出的数据有效：

```
----[1238935 ns] Number 8'd40 Functional Test Point PASS!!!
--------------------------------------------------------------
[1239327 ns] Error!!!
    reference: PC = 0x1c02ef4c, wb_rf_wnum = 0x10, wb_rf_wdata = 0x0000000b
    mycpu    : PC = 0x1c02ef4c, wb_rf_wnum = 0x10, wb_rf_wdata = 0x00c83b0b
--------------------------------------------------------------
$finish called at time : 1239367 ns : File "D:/Desktop/cdp_ede_local/mycpu_env_11/soc_verify/soc_bram/testbench/mycpu_t
```

（5）归纳总结（可选）

补充新指令的时候注意联系其相似指令补充数据通路，避免遗漏。

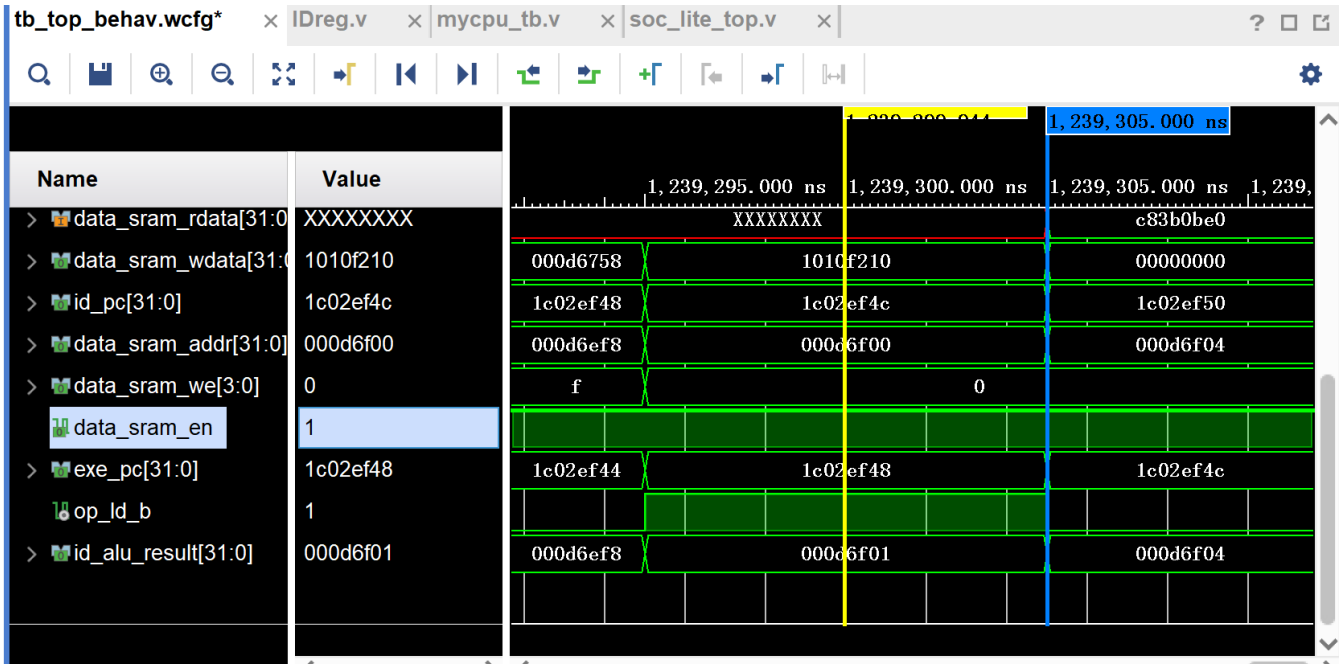
错误5：忽略内存异步读特性

（1）错误现象

写回数据错误

（2）分析定位过程

注意到此指令仍是ld.b指令，且写回数据的后8byte是正确的，猜测是其未能在确定写回数据正确判断指令类型。查看op_ld_b拉高时机：



发现其在exe_pc等于出错点pc值的前一个周期拉高，查看其实现：

```
    assign {op_st_b, op_st_h, op_st_w, op_ld_b,
            op_ld_bu,op_ld_h, op_ld_hu, op_ld_w} = id_inst_zip;
```

（3）错误原因

上述处理中不加区分地对ld和st类指令都使用了组合逻辑在流水级间传递，但由于ld指令在给出有效地址后需要等待一个周期以获取读数据，此时再使用其指令本身信息来判断写回数据，故其应该使用非阻塞赋值，滞后一个周期。

（4）修正效果

PASS!

```
----[1332195 ns] Number 8'd45 Functional Test Point PASS!!!
        [1342000 ns] Test is running, debug_wb_pc = 0x1c02df48
        [1352000 ns] Test is running, debug_wb_pc = 0x1c02eee8
----[1352075 ns] Number 8'd46 Functional Test Point PASS!!!
==============================================================
Test end!
----PASS!!!
$finish called at time : 1352515 ns : File "D:/Desktop/cdp_ede_local/mycpu_env_11/soc_verify/soc_bram/tes
run: Time (s): cpu = 00:00:46 ; elapsed = 00:01:21 . Memory (MB): peak = 1079.109 ; gain = 0.000
```

（5）归纳总结（可选）

注意数据预取的处理对逻辑选择的影响!

四.实验总结

在处理乘除法器的时候需要尤其注意数组的宽度以及符号的处理，因为设计过程中涉及许多符号扩展的部分，如定义部分积、booth两位乘时应至少取三位符号位等。