简单可重构计算阵列电路

组别：数字组 D15015

目录

1. 系统方案 3

1.1 设计思路与题目分析 3

1.2 总体设计框图与模块信息 4

2. 电路与程序设计 5

2.1 ALU 计算模块 5

2.2 PE 模块 5

2.3 数据存储器模块 6

2.4 配置存储器模块 7

2.5 顶层模块 7

2.6 总览 8

3. 模块测试结果 9

3.1 ALU 计算模块 9

3.2 PE 模块 9

3.3 数据存储器模块 10

3.4 配置存储器模块 10

3.5 顶层模块 11

4. 结论 12

5. 完整代码附件 13

5.1 ALU 计算模块 13

5.2 PE 模块 13

5.3 数据存储器模块 14

5.4 配置存储器模块 15

5.5 顶层模块 15

1. 系统方案

本次比赛的实验选取数字组题目三：简单可重构计算阵列电路。本组通过 modelsim 软件设计电路并进行仿真，通过设计模块电路和测试电路，对所设计的电路进行仿真，观察输出波形，并判断电路设计合理情况。

设计一个简单可重构阵列，实现对不同计算要求的电路重构功能。主要模块包括计算模块、操作模块、配置存储器、数据存储器等。

1.1 设计思路与题目分析

根据题目要求，分别实现配置存储器存取模块，4 个数据存储器存取模块，4 个处理单元 PE 以及 PE 内部的 ALU 计算模块。

- 配置存储器存取模块

  根据测试文件中提供的地址，从 cm_memory.txt 中读取到对应的可重构阵列配置信息，将配置信息按配置位进行拆分，将其传递给处理单元 PE 和数据存储区存取模块。

- 处理单元 PE

  根据配置信息，选取需要对应的数据，将其发送给其内部模块 ALU 计算模块。

- ALU 计算模块

  根据操作码对输入的数据进行加法、减法、与、或、异或运算。

- 数据存储器存取模块

  根据配置信息，对数据存储器进行读或写操作

1.2 总体设计框图与模块信息

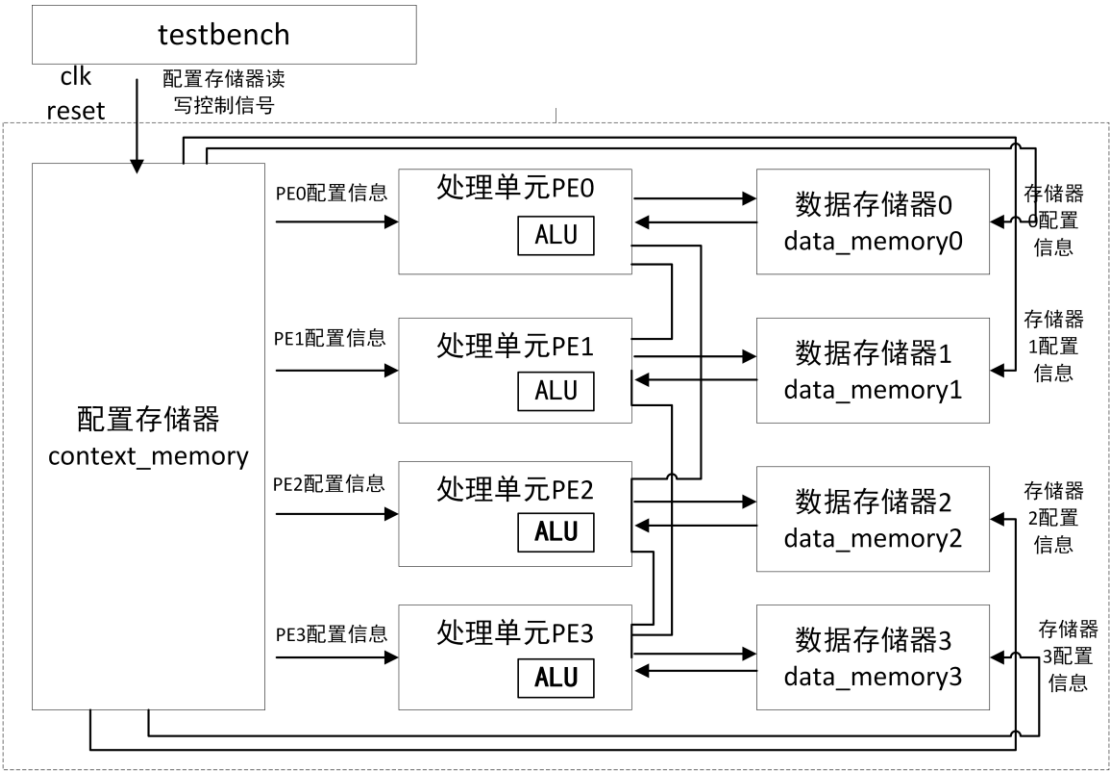

图 1 整体电路结构

表 1 模块信息

模块名	端口数	功能介绍
top	4	顶层模块
pe	7	pe 处理单元
data_memory	6	dm 数据存储器
context_memory	6	cm 数据存储器
alu	4	alu 计算单元

2. 电路与程序设计

2.1 ALU 计算模块

图 2 ALU 计算模块

表 2 ALU 端口列表

端口名	端口长度	端口性质	功能介绍
operation	**3**	**Input**	**操作码**
op_lhs	**16**	**Input**	**要进行运算的数据 1**
op_rhs	**16**	**Input**	**要进行运算的数据 2**
result	**16**	**Output**	**运算产生的结果**

ALU 计算模块，能够对输入的数据进行加法、减法、与、或、异或运算。

2.2 PE 模块

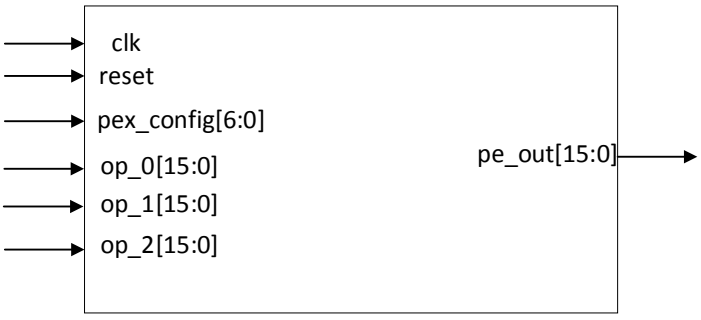

图 3 PE 模块

表 3 PE 端口列表

端口名	端口长度	端口性质	功能介绍
clk	1	Input	时钟输入
reset	1	Input	复位信号，高电平有效
pex_config	7	Input	从配置存储器中读出的配置信息
op_0	16	Input	外部输入的操作数 1
op_1	16	Input	外部输入的操作数 2
op_2	16	Input	外部输入的操作数 3
pe_out	16	Output	ALU 计算结果

根据配置流中操作数选择信号选择计算单元的输入，根据配置流中操作类型码执行相应的计算操作，并把计算结果存储在本地 16 位结果寄存器中，该寄存器的值通过端口输出，以作为其他 PE 模块和数据存储器的输入。

2.3 数据存储器模块

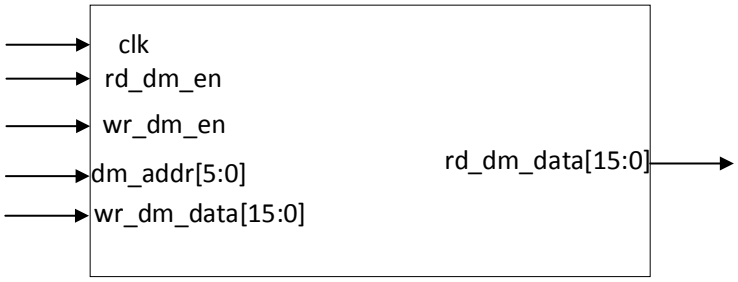

图 4 数据存储器结构

表 4 dm 端口列表

端口名	端口长度	端口性质	功能介绍
clk	1	Input	时钟输入
rd_dm_en	1	Input	读取数据使能信号，高电平有效
wr_dm_en	1	Input	写入数据使能信号，高电平有效
dm_addr	6	Input	读写数据地址
wr_dm_data	16	Input	要写入数据存储器中的数据
rd_dm_data	16	Output	从数据存储器中读出的数据

2.4 配置存储器模块

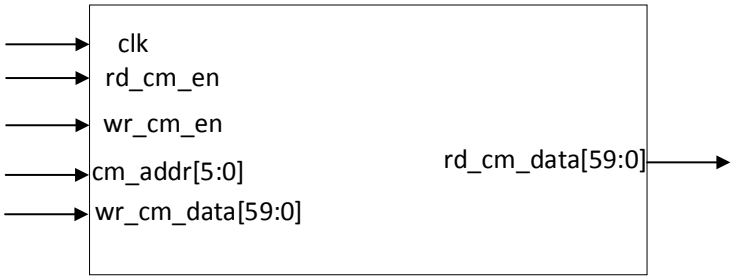

图 5 配置存储器

表 5 cm 端口列表

端口名	端口长度	端口性质	功能介绍
clk	1	Input	时钟输入
rd_cm_en	1	Input	读取配置数据使能信号，高电平有效
wr_cm_en	1	Input	写入配置数据使能信号，高电平有效
cm_addr	6	Input	读写数据地址
wr_cm_data	60	Input	要写入配置存储器中的数据
rd_cm_data	60	Output	从配置存储器中读出的数据

2.5 顶层模块

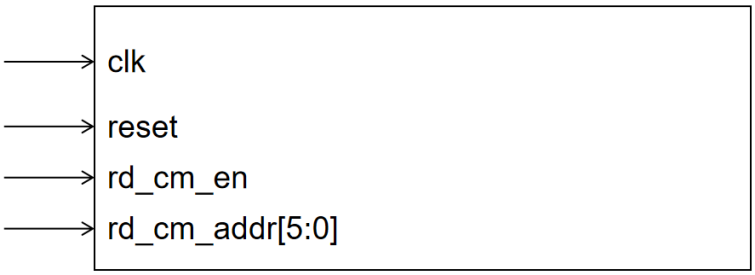

图 6 顶层模块

表 6 顶层端口列表

端口名	端口长度	端口性质	功能介绍
clk	1	Input	时钟输入
reset	1	Input	复位信号，高电平有效
rd_cm_en	1	Input	读取配置存储器使能信号
rd_cm_addr	6	Input	读取配置存储器的地址信息

2.6 总览

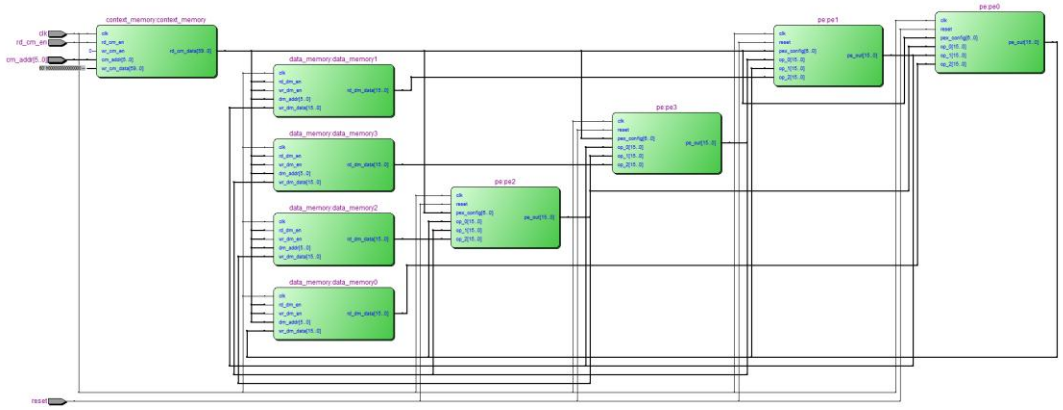

图 7 整体电路结构

3. 模块测试结果

3.1 ALU 计算模块

如图 8 所示，alu 计算模块可以根据输入的操作码正确计算出结果

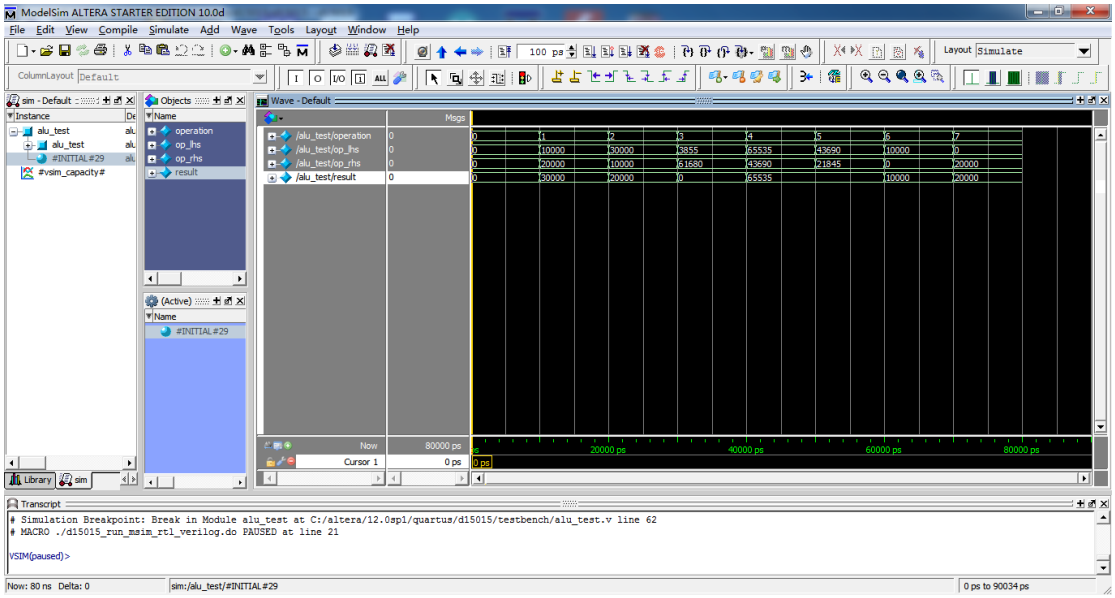

图 8 ALU 计算模块波形图

3.2 PE 模块

如图 9 所示，PE 模块可以根据输入配置信息正确的选取数据源并调用 ALU 计算模块得到正确的对应结果。

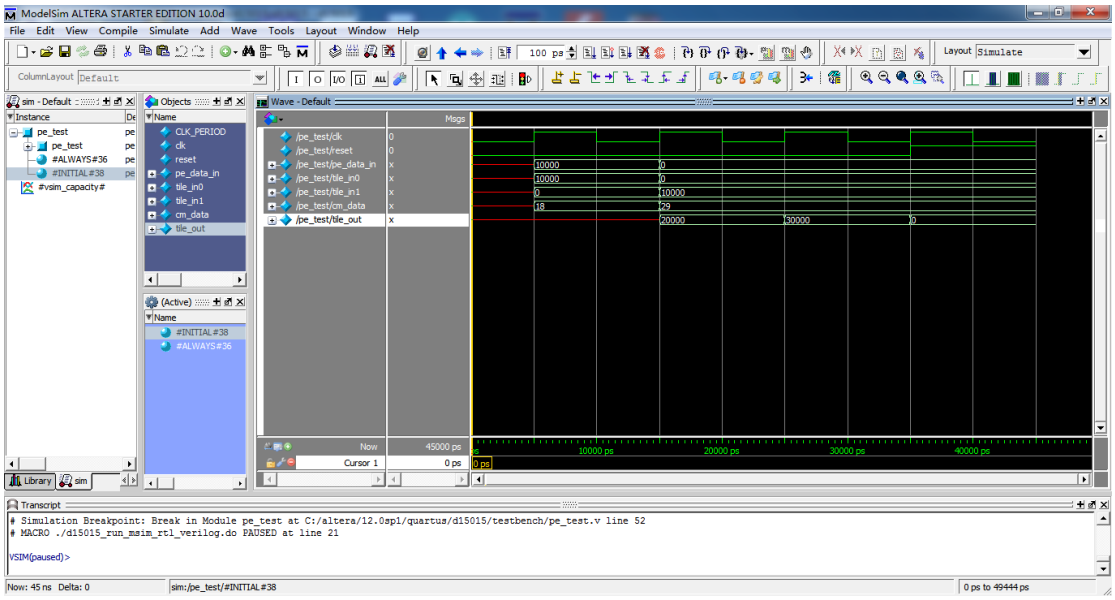

图 9 PE 模块波形图

3.3 数据存储器模块

如图 10 所示，数据存储器模块可以根据读写使能信息正确的对输入地址下的数据进行读写操作

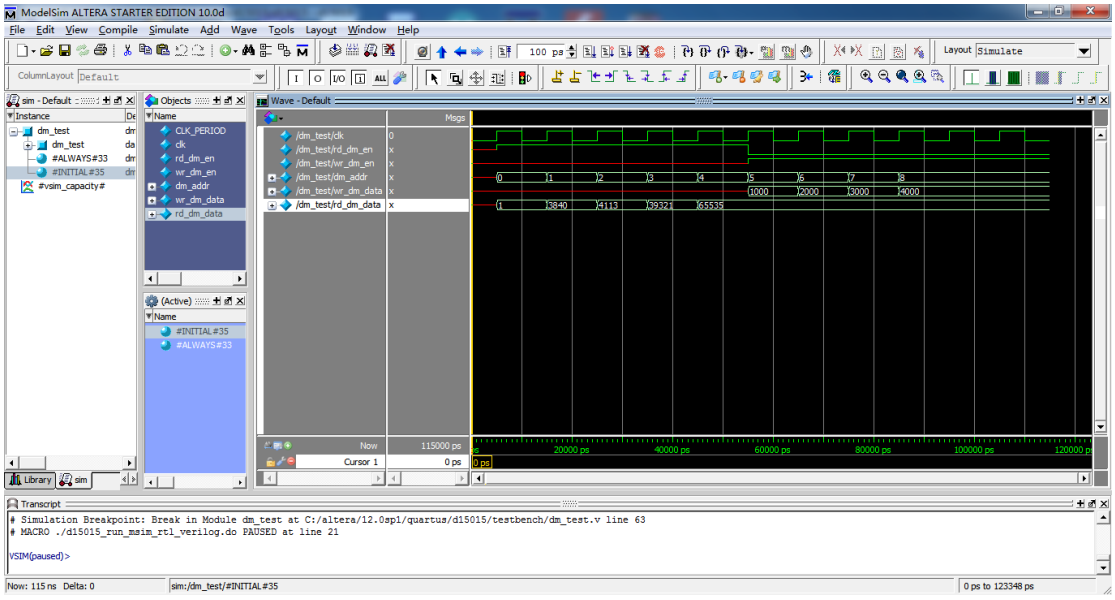

图 10 数据存储器模块波形图

3.4 配置存储器模块

如图 11 所示，配置存储器模块可以根据读写使能信息正确的对输入地址下

的数据进行读写操作

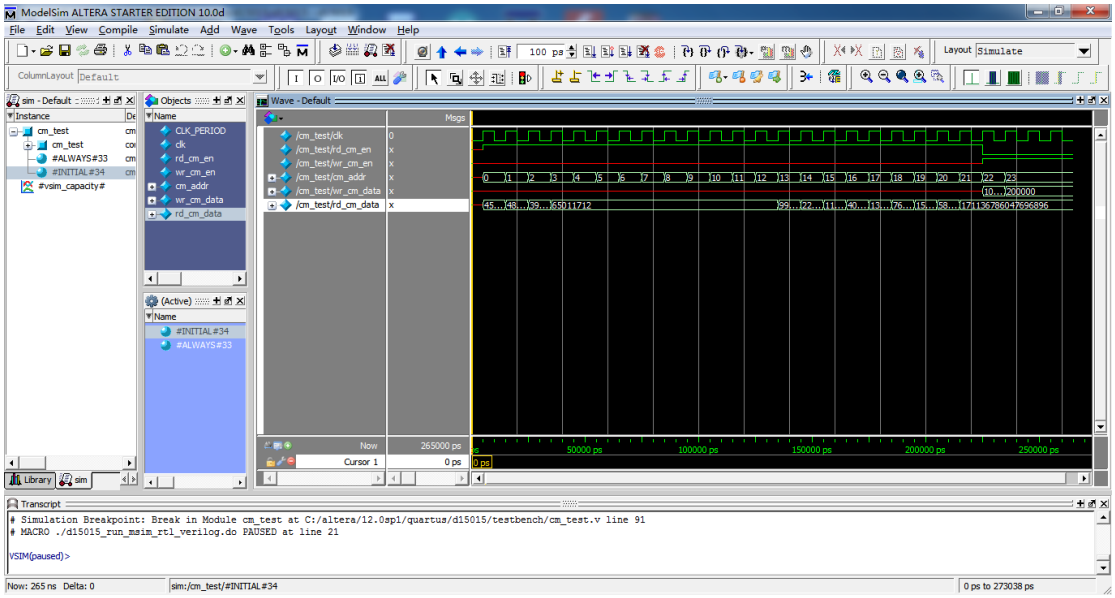

图 11 配置存储器模块波形图

3.5 顶层模块

如图 12 所示，顶层模块可以正确地调用各个子模块进行数据处理，传输等功能

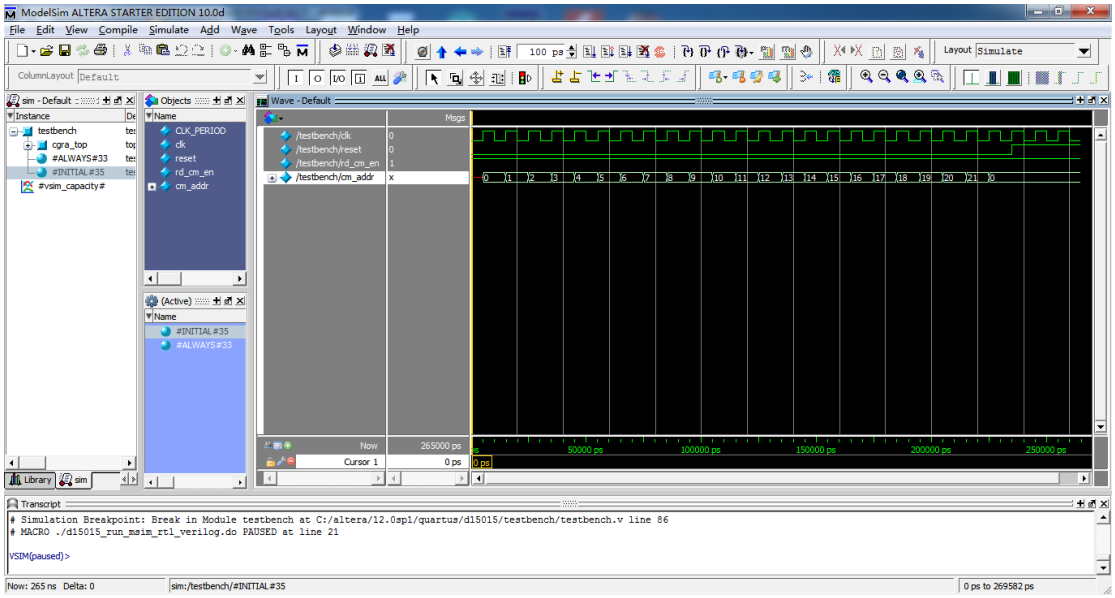

图 12 顶层模块波形图

4. 结论

本次实验，本组很好地完成了赛题要求的全部内容，通过 modelsim 仿真的方式，正确实现了题目所需模块的各个功能，做到设计电路时灵活运用。

5. 完整代码附件

5.1 ALU 计算模块

```
module alu(
    input [2:0] operation,
    input [15:0] op_lhs, op_rhs,
    output reg [15:0] result
);
    always @ (operation or op_lhs or op_rhs) begin
        case (operation)
            3'b000: result <= 16'b0;
            3'b001: result <= op_lhs + op_rhs;
            3'b010: result <= op_lhs - op_rhs;
            3'b011: result <= op_lhs & op_rhs;
            3'b100: result <= op_lhs | op_rhs;
            3'b101: result <= op_lhs ^ op_rhs;
            3'b110: result <= op_lhs;
            3'b111: result <= op_rhs;
            default: result <= 16'b0;
        endcase
    end
endmodule
```

5.2 PE 模块

```
module pe (
    input clk, reset,
    input [6:0] pex_config,
    input [15:0] op_0, op_1, op_2,
    output reg [15:0] pe_out
);
    reg [15:0] op_lhs, op_rhs;
    wire [15:0] alu_result;
    alu alu (
        .operation(pex_config[6:4]),
        .op_lhs(op_lhs),
        .op_rhs(op_rhs),
        .result(alu_result)
    );
    always @ (posedge clk or posedge reset) begin
        if (reset) pe_out <= 0;      //???
```

```
            else pe_out <= alu_result;
        end
        always @ (pex_config or op_0 or op_1 or op_2 or pe_out) begin
            case (pex_config[1:0])
                2'b00: op_lhs <= op_0;
                2'b01: op_lhs <= op_1;
                2'b10: op_lhs <= op_2;
                2'b11: op_lhs <= pe_out;
            endcase
            case (pex_config[3:2])
                2'b00: op_rhs <= op_0;
                2'b01: op_rhs <= op_1;
                2'b10: op_rhs <= op_2;
                2'b11: op_rhs <= pe_out;
            endcase
        end
endmodule
```

5.3 数据存储器模块

```
module data_memory (
        input clk, rd_dm_en, wr_dm_en,
        input [5:0] dm_addr,
        input [15:0] wr_dm_data,
        output reg [15:0] rd_dm_data
);
        reg [15:0] dm [63:0];
        initial begin
            $readmemb("C:\\altera\\12.0sp1\\quartus\\d15015\\dm_memory.txt", dm);
        end
        always @ (posedge clk) begin
            if (rd_dm_en) rd_dm_data <= dm[dm_addr];
            if (wr_dm_en) begin
                dm[dm_addr] <= wr_dm_data;
                $writememb("C:\\altera\\12.0sp1\\quartus\\d15015\\dm_memory.txt", dm);
            end
        end
endmodule
```

5.4 配置存储器模块

```
module context_memory (
    input clk, rd_cm_en, wr_cm_en,
    input [5:0] cm_addr,
    input [59:0] wr_cm_data,
    output reg [59:0] rd_cm_data
);
    reg [59:0] cm [63:0];
    initial begin
        $readmemb("C:\\altera\\12.0sp1\\quartus\\d15015\\cm_memory.txt", cm);
    end
    always @ (posedge clk) begin
        if (rd_cm_en) rd_cm_data <= cm[cm_addr];
        if (wr_cm_en) begin
            cm[cm_addr] <= wr_cm_data;
            $writememb("C:\\altera\\12.0sp1\\quartus\\d15015\\cm_memory.txt", cm);
        end
    end
endmodule
```

5.5 顶层模块

```
module top (
    input clk, reset, rd_cm_en,
    input [5:0] cm_addr
);
    wire [7:0] dm0_control, dm1_control, dm2_control, dm3_control;
    wire [6:0] pe0_config, pe1_config, pe2_config, pe3_config;
    wire [15:0] pe0_out, pe1_out, pe2_out, pe3_out;
    wire [15:0] rd_dm0_data, rd_dm1_data, rd_dm2_data, rd_dm3_data;
    pe pe0 (
        .clk(clk),
        .reset(reset),
        .pex_config(pe0_config),
        .op_0(pe2_out),
        .op_1(pe1_out),
        .op_2(rd_dm0_data),
        .pe_out(pe0_out)
    );
    pe pe1 (
        .clk(clk),
        .reset(reset),
```

```
        .pex_config(pe1_config),
        .op_0(pe3_out),
        .op_1(pe0_out),
        .op_2(rd_dm1_data),
        .pe_out(pe1_out)
);
pe pe2 (
        .clk(clk),
        .reset(reset),
        .pex_config(pe2_config),
        .op_0(pe0_out),
        .op_1(pe3_out),
        .op_2(rd_dm2_data),
        .pe_out(pe2_out)
);
pe pe3 (
        .clk(clk),
        .reset(reset),
        .pex_config(pe3_config),
        .op_0(pe1_out),
        .op_1(pe2_out),
        .op_2(rd_dm3_data),
        .pe_out(pe3_out)
);
data_memory data_memory0 (
        .clk(clk),
        .rd_dm_en(dm0_control[0]),
        .wr_dm_en(dm0_control[1]),
        .dm_addr(dm0_control[7:2]),
        .wr_dm_data(pe0_out),
        .rd_dm_data(rd_dm0_data)
);
data_memory data_memory1 (
        .clk(clk),
        .rd_dm_en(dm1_control[0]),
        .wr_dm_en(dm1_control[1]),
        .dm_addr(dm1_control[7:2]),
        .wr_dm_data(pe1_out),
        .rd_dm_data(rd_dm1_data)
);
data_memory data_memory2 (
        .clk(clk),
        .rd_dm_en(dm2_control[0]),
        .wr_dm_en(dm2_control[1]),
```

```
        .dm_addr(dm2_control[7:2]),
        .wr_dm_data(pe2_out),
        .rd_dm_data(rd_dm2_data)
    );
    data_memory data_memory3 (
        .clk(clk),
        .rd_dm_en(dm3_control[0]),
        .wr_dm_en(dm3_control[1]),
        .dm_addr(dm3_control[7:2]),
        .wr_dm_data(pe3_out),
        .rd_dm_data(rd_dm3_data)
    );
    context_memory context_memory (
        .clk(clk),
        .rd_cm_en(rd_cm_en),
        .wr_cm_en(1'b0),
        .cm_addr(cm_addr),
        .wr_cm_data(60'b0),
        .rd_cm_data({
            dm3_control,
            dm2_control,
            dm1_control,
            dm0_control,
            pe3_config,
            pe2_config,
            pe1_config,
            pe0_config
        })
    );
endmodule
```