《计算机组成原理》实验报告

年级、专业、班级		**姓名**	
实验题目	实验一简单流水线与运算器实验		
实验时间		**实验地点**	
实验成绩	优秀/良好/中等	**实验性质**	□验证性 ☑设计性 □综合性
教师评价： ☑算法/实验过程正确；☑源程序/实验内容提交；☑程序结构/实验步骤合理； ☑实验结果正确；☑语法、语义正确；☑报告规范； 其他: 评价教师:			
实验目的 (1)理解流水线 (Pipeline) 设计原理； (2)了解算术逻辑单元 ALU 的原理； (3)熟悉并运用 Verilog 语言设计 ALU； (4)熟悉并运用 Verilog 语言设计流水线全加器；			

报告完成时间: 2024 年 7 月 21 日

1 实验内容

1.1 ALU 设计实验

实验要求实现以下算术运算功能，其对应的控制码及功能如下：

$F_{2:0}$	功能	$F_{2:0}$	功能
000	A + B(Unsigned)	100	$\overline{A}$
001	A - B	***101***	***SLT***
010	A AND B	110	未使用
011	A OR B	111	未使用

表 1: 算数运算控制码及功能

实验要求：

1. 根据 ALU 原理图，使用 Verilog 语言定义 ALU 模块，其中输入输出端口参考实验原理，运算指令码长度为 [2:0]。
2. 仿真时 B 端口输入为 32h'01，A 端口输入参照 4.1 中表格
3. 实现 SLT 功能。
4. 验证表 1中所有功能。
5. 给出 RTL 源程序（.v 文件）

1.2 流水线实验

本次实验为仿真实验，设计完成后仅需进行**行为仿真**。

实验要求：

1. 实现 4 级流水线 8bit 全加器，需带有流水线暂停和刷新；
2. 模拟流水线暂停，仿真时控制 10 周期后暂停流水线 2 周期（第 2 级），流水线恢复流动；
3. 模拟流水线刷新，仿真时控制 15 周期时流水线刷新（第 3 级）。

2 实验设计

2.1 ALU

2.1.1 功能描述

本次实验主要设计了两个功能模块：

1. ALU_FOR_TB: 该模块实现的是用于仿真的加、减、与、或、非和比较大小功能。

2. ALU: 该模块实现的是用于开发板的加、减、与、或、非和比较大小功能。

2.1.2 接口定义

表 2: 接口定义模版 (ALU_FOR_TB)

信号名	方向	位宽	功能描述
clk	Input	1-bit	System clock signal.
rst	Input	1-bit	Reset signal.
op	Input	3-bit	Operation code.
num1	Input	8-bit	Lowest 8-bit of the second operand.
results	Output	32-bit	Calculation Result.

表 3: 接口定义模版 (ALU)

信号名	方向	位宽	功能描述
clk	Input	1-bit	System clock signal.
rst	Input	1-bit	Reset signal.
op	Input	3-bit	Operation code.
num1	Input	8-bit	Lowest 8-bit of the second operand.
enable	Output	8-bit	Nixie tube enable signal.
segs	Output	7-bit	Nixie tube trigger signal..

2.1.3 逻辑控制

根据 op 信号选取指定功能计算 results 的结果。上述功能可以通过系统内置的操作符直接实现。在得到计算结果后, 通过分频信号 div 和使能信号 enable 使指定的数码管按特定频率闪烁, 通过视觉残留现象使各个数码管“同时” 呈现不同的数值。由 segs 信号控制指定数码管呈现的数值。

2.2 有阻塞 4 级 32bit 全加器

2.2.1 功能描述

本模块为 4 级 32-bit 流水线加法器, 每一层可以单独进行暂停或刷新。

2.2.2 接口定义

表 4: 接口定义模版 (Pipeline_adder)

信号名	方向	位宽	功能描述
clk	Input	1-bit	System clock signal.
rst	Input	1-bit	Reset signal.
validin	Input	1-bit	If there is valid data_in.
a	Input	32-bit	The first operand.
b	Input	32-bit	The second operand.
carry_in	Input	1-bit	The carry_in signal used for the adder.
out_allow	Input	1-bit	If the output is allowed.
suspend	Input	4-bit	If the specific level of the Pipeline_adder is suspended.
refresh	Input	4-bit	If the specific level of the Pipeline_adder is refreshed.
validout	Output	1-bit	If there is valid data_out.
sum_out	Output	32-bit	The calculation result of the adder.
carry_out	Output	1-bit	The carry_out signal used for the adder.

2.2.3 逻辑控制

对于每一个上升沿进行的加法操作, 其控制信号和操作数在前一个下降沿给定。当 suspend 信号和 refresh 信号均为低电平时, 流水线正常流动; 当 suspend 信号为高电平时, 指定阶段和之前阶段的流水线停止流动, 后续阶段的流水线正常流动; 当 refresh 信号为高电平时, 指定阶段的寄存器被清空, 其对应的加法操作无有效输出。

3 实验过程记录

3.1 ALU

3.1.1 问题 1:rst 信号为低电平时仍然有数码管亮起

问题描述: 分频信号 div 和复位信号 rst 共同控制状态信号 cnt8(3 位寄存器, 对应八种状态, 每种状态使能唯一的一个特定七段数码管), 状态控制信号 cnt8 控制使能信号 enable。当复位信号 rst 处于低电平时, 状态信号 cnt8 一定会处于某种特定状态, 使能某一七段数码管。

解决方案: 分频信号 div 和复位信号 rst 共同控制状态信号 cnt8 和使能信号 enable。当复位信号 rst 处于低电平时, 将使能信号 enable 置为高电平, 使所有数码管熄灭。

3.2 有阻塞 4 级 32bit 全加器

3.2.1 问题 1: 仿真测试中第一次加法在第五个上行沿才输出计算结果

问题描述: 4 级流水线加法器应该在第四个上行沿输出第一次加法计算结果。但在 initial 语块中, 数据输入与 clk 信号上行沿同步, 导致数据输入实际上延迟了一个周期。

解决方案: 在 clk 信号每个上行沿前的最后一个下行沿进行数据输入, 保证 clk 信号上行沿到来时数据已经被读入到加法器输入端。

3.2.2 问题 2: 第二级流水线暂停后只有一次加法输出

问题描述: 某一级流水线暂停实际上是暂停这一阶段**前**所有阶段的流水线。第二级流水线暂停后, 第三和第四级流水线应该正常流动并输出两次加法运算结果。

解决方案: 修改 initial 语块中第二级流水线暂停的 suspend 信号为 4'b0001

3.2.3 问题 3: 流水线输出结果混乱

问题描述: 4 级流水线加法器最多可以同时处理四次加法运算的不同阶段, 当某一次加法的某一级流水线运算结束后, 该阶段的加法输入必须根据下一次加法运算要求在下一个 clk 信号上行沿到来前更改, 否则将因为重复输入导致流水线混乱。

解决方案: 每一级的加法输入根据运行到这一阶段的特定次加法运算及时修改。

4 实验结果及分析

4.1 ALU 验证实验结果

以下表格和图片展示的分别是 ALU 模块的实验预期结果,RTL 电路图, 仿真图和开发板实际运行结果图。可以看出, 仿真图和开发板实际运行结果图保持一致, 均符合实验预期, 证明实验取得了良好的结果。

操作	Num1	Result
A + B(Unsigned)	8'b00000010	32'h00000003
A - B	8'b11111111	32'hFFFFFF02
A AND B	8'b11111110	32'h00000000
A OR B	8'b10101010	32'h000000AB
$\overline{A}$	8'b11110000	32'hFFFFFFFE
SLT	8'b10000001	32'h00000001

表 5: ALU 结果表

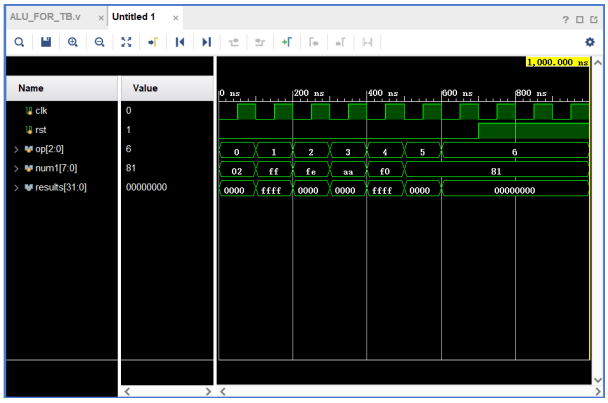

图 1: ALU 模块: 仿真图

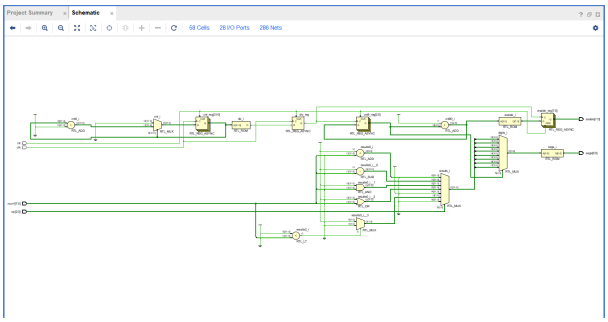

图 2: ALU 模块:RTL 电路图

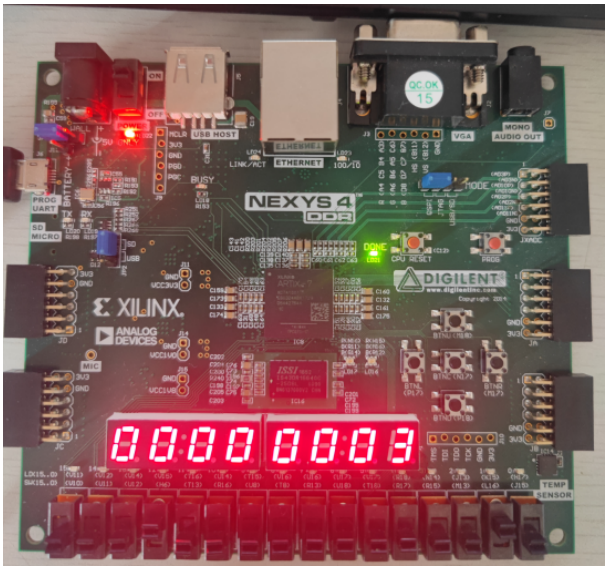

图 3: ALU 模块: 加法操作

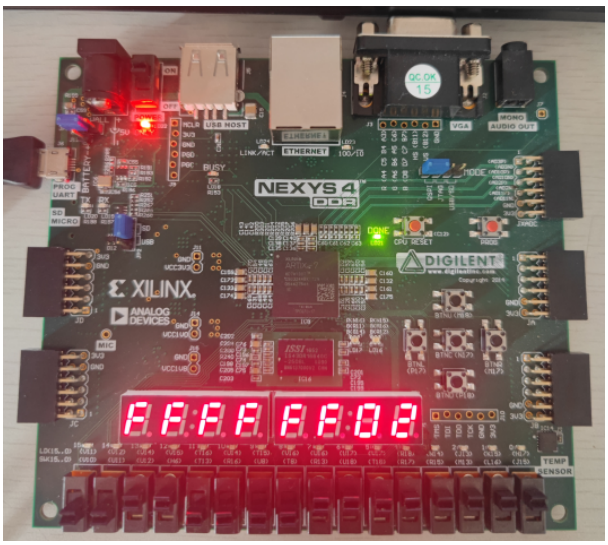

图 4: ALU 模块: 减法操作

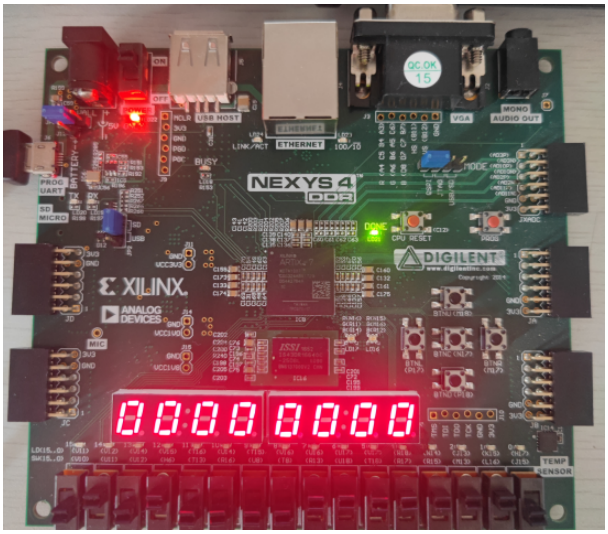

图 5: ALU 模块: 与操作

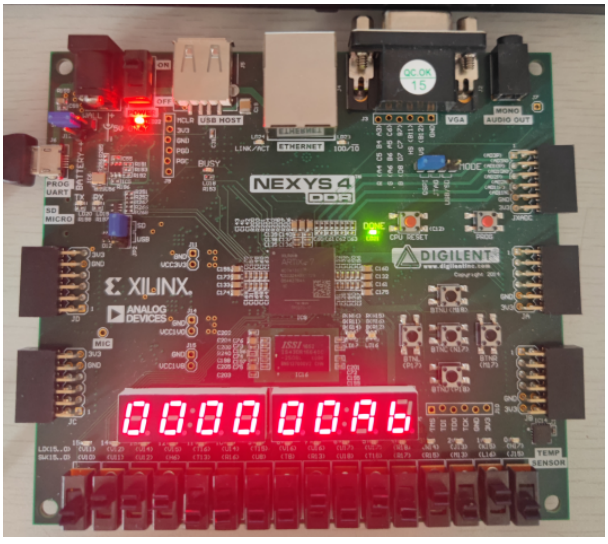

图 6: ALU 模块: 或操作

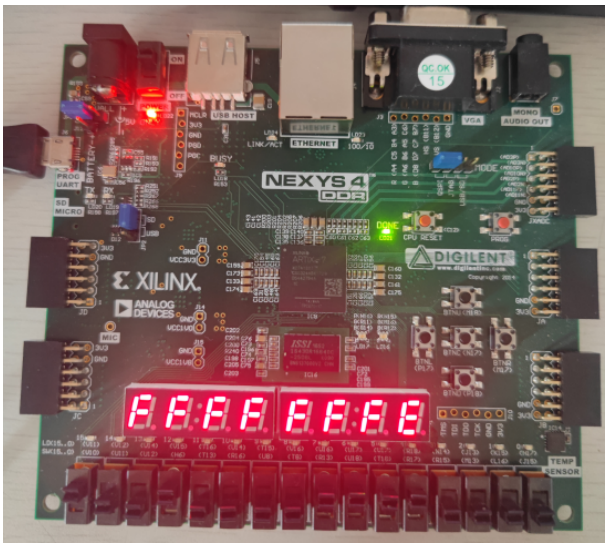

图 7: ALU 模块: 非操作

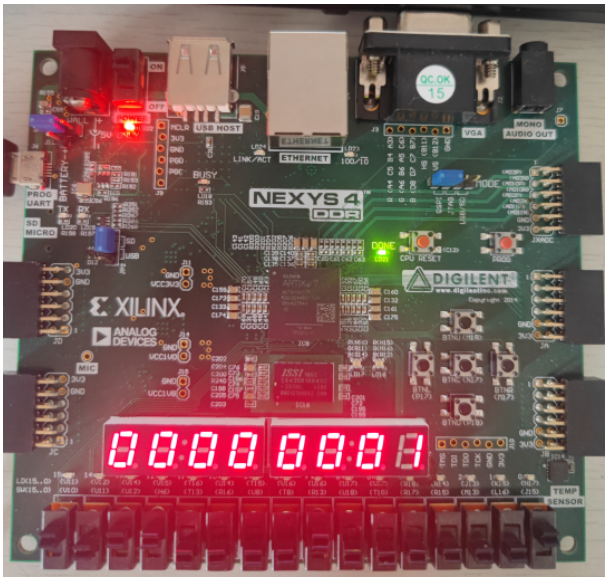

图 8: ALU 模块:SLT 操作

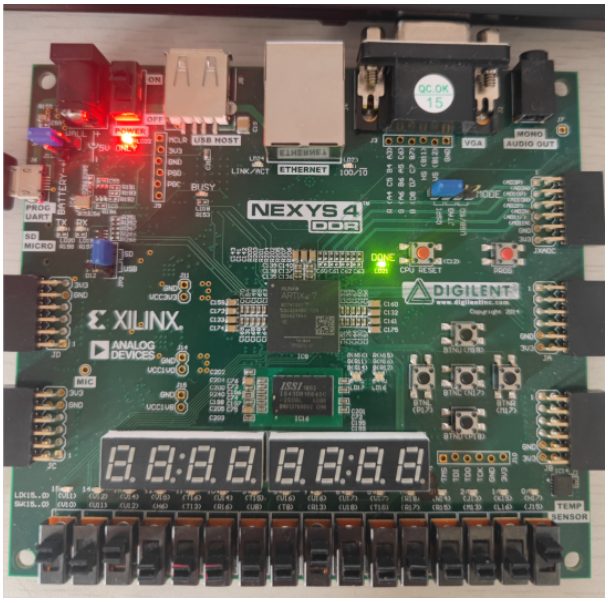

图 9: ALU 模块:RST 操作

4.2 流水线阻塞(暂停)仿真图

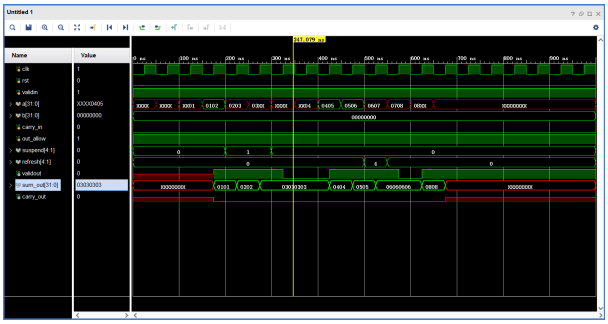

图 10: 流水线阻塞(暂停)仿真图(第二级暂停/第三次加法暂停两个周期)

4.3 流水线刷新(清空)仿真图

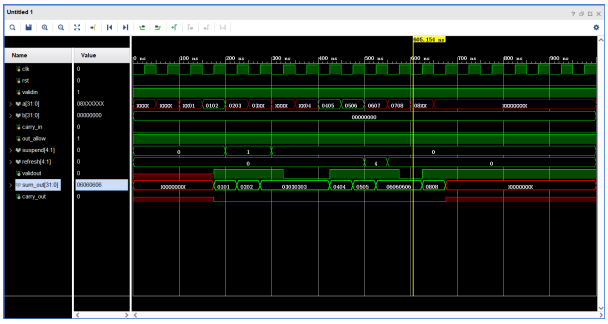

图 11: 流水线刷新(清空)仿真图(第三级刷新/第七次加法无有效输出)

A ALU 代码

```
module ALU(clk,rst,op,num1,enable,segs);
parameter MAX_CNT=32'h20000;

input wire clk;
input wire rst;
input wire [2:0] op;
input wire [7:0] num1;
output reg [7:0] enable;
output reg [6:0] segs;

wire [31:0] num2=32'h01;
wire [31:0] A,B;
reg div=1'b0;
reg [31:0] results;
reg [2:0] cnt8=3'b0;
reg [3:0] digits;
integer cnt=0;

assign A=num2;
assign B={24'h0,num1};

always@(op or A or B)
begin
    case(op)
    3'b000:results=A+B;
    3'b001:results=A-B;
    3'b010:results=A&B;
    3'b011:results=A|B;
    3'b100:results=~A;
    3'b101:results=(A<B)?32'b1:32'b0;
    default:results=32'b0;
    endcase
end

always@(posedge clk or negedge rst)
begin
    if(!rst)
    begin
        div=1'b0;
        cnt=0;
    end
    else
    begin
        if(cnt==MAX_CNT)
        begin
            div=1'b1;
            cnt=0;
        end
        else
```

```
        begin
            div=1'b0;
            cnt=cnt+1;
        end
    end
end

always@(posedge div or negedge rst)
begin
    if(!rst)
    begin
        cnt8=3'b0;
        enable=8'b1111_1111;
    end
    else
    begin
         cnt8=cnt8+1'b1;
         case(cnt8)
            3'b000:enable=8'b1111_1110;
            3'b001:enable=8'b1111_1101;
            3'b010:enable=8'b1111_1011;
            3'b011:enable=8'b1111_0111;
            3'b100:enable=8'b1110_1111;
            3'b101:enable=8'b1101_1111;
            3'b110:enable=8'b1011_1111;
            3'b111:enable=8'b0111_1111;
            default:enable=8'b1111_1111;
         endcase
    end
end

always@(cnt8 or results)
begin
    case(cnt8)
    3'b000:digits=results[3:0];
    3'b001:digits=results[7:4];
    3'b010:digits=results[11:8];
    3'b011:digits=results[15:12];
    3'b100:digits=results[19:16];
    3'b101:digits=results[23:20];
    3'b110:digits=results[27:24];
    3'b111:digits=results[31:28];
    default:digits=4'b0;
    endcase
end

always@(digits)
begin
    case(digits)
```

```
            4'h0:segs = 7'b000_0001;//0
            4'h1:segs = 7'b100_1111;//1
            4'h2:segs = 7'b001_0010;//2
            4'h3:segs = 7'b000_0110;//3
            4'h4:segs = 7'b100_1100;//4
            4'h5:segs = 7'b010_0100;//5
            4'h6:segs = 7'b010_0000;//6
            4'h7:segs = 7'b000_1111;//7
            4'h8:segs = 7'b000_0000;//8
            4'h9:segs = 7'b000_0100;//9
            4'ha:segs = 7'b000_1000;//a
            4'hb:segs = 7'b110_0000;//b
            4'hc:segs = 7'b011_0001;//c
            4'hd:segs = 7'b100_0010;//d
            4'he:segs = 7'b011_0000;//e
            4'hf:segs = 7'b011_1000;//f
        endcase
    end
endmodule
```

B ALU_FOR_TB 代码

```
module ALU_FOR_TB(clk,rst,op,num1,results);
input wire clk;
input wire rst;
input wire [2:0] op;
input wire [7:0] num1;
output wire [31:0] results;

wire [31:0] num2=32'h01;
wire [31:0] A,B;

assign A=num2;
assign B={24'h0,num1};
assign results=(op==3'b000)?A+B:
               (op==3'b001)?A-B:
               (op==3'b010)?A&B:
               (op==3'b011)?A|B:
               (op==3'b100)?~A:
               (op==3'b101)?((A<B)?32'b1:32'b0):32'b0;
endmodule
```

C ALU 仿真文件代码

```
module alu_tb();
```

```
    reg clk;
    reg rst;
    reg [2:0] op;
    reg [7:0] num1;
    wire [31:0] results;

    always #50 clk=~clk;

    initial
    begin
        clk<=1'b0;
        rst<=1'b0;
        op<=3'b000;
        num1<=8'b0000_0010;
        #100
        op<=3'b001;
        num1<=8'b1111_1111;
        #100
        op<=3'b010;
        num1<=8'b1111_1110;
        #100
        op<=3'b011;
        num1<=8'b1010_1010;
        #100
        op<=3'b100;
        num1<=8'b1111_0000;
        #100
        op<=3'b101;
        num1<=8'b1000_0001;
        #100
        op<=3'b110;
        #100
        rst<=1'b1;
    end

    ALU_FOR_TB myalu(clk,rst,op,num1,results);
endmodule
```

D 32bit 流水线全加器代码

```
    module Pipeline_adder(clk,rst,validin,a,b,carry_in,out_allow,suspend,refresh,
        validout,sum_out,carry_out);
    input clk;
    input rst;
    input validin;
    input [31:0] a;
    input [31:0] b;
```

```
input carry_in;
input out_allow;
input [4:1] suspend;
input [4:1] refresh;
output validout;
output reg [31:0] sum_out;
output reg carry_out;

reg carry_out_t1,carry_out_t2,carry_out_t3;
reg pipe1_valid;
reg pipe2_valid;
reg pipe3_valid;
reg pipe4_valid;
reg [7:0] sum_out_t1;
reg [15:0] sum_out_t2;
reg [23:0] sum_out_t3;

wire pipe1_allowin;
wire pipe2_allowin;
wire pipe3_allowin;
wire pipe4_allowin;
wire pipe1_ready_go;
wire pipe2_ready_go;
wire pipe3_ready_go;
wire pipe4_ready_go;
wire pipe1_to_pipe2_valid;
wire pipe2_to_pipe3_valid;
wire pipe3_to_pipe4_valid;


assign pipe1_ready_go=!suspend[1];
assign pipe2_ready_go=!suspend[2];
assign pipe3_ready_go=!suspend[3];
assign pipe4_ready_go=!suspend[4];
assign pipe1_allowin=!pipe1_valid||(pipe1_ready_go&&pipe2_allowin);
assign pipe2_allowin=!pipe2_valid||(pipe2_ready_go&&pipe3_allowin);
assign pipe3_allowin=!pipe3_valid||(pipe3_ready_go&&pipe4_allowin);
assign pipe4_allowin=!pipe4_valid||(pipe4_ready_go&&out_allow);
assign pipe1_to_pipe2_valid=pipe1_valid&&pipe1_ready_go;
assign pipe2_to_pipe3_valid=pipe2_valid&&pipe2_ready_go;
assign pipe3_to_pipe4_valid=pipe3_valid&&pipe3_ready_go;
assign validout=pipe4_valid&&pipe4_ready_go;

always@(posedge clk)
begin
    if(rst||refresh[1])
    begin
        pipe1_valid<=1'b0;
    end
```

```
    else if(pipe1_allowin)
    begin
        pipe1_valid<=validin;
    end
    if(validin&&pipe1_allowin)
    begin
        {carry_out_t1,sum_out_t1}<={1'b0,a[7:0]}+{1'b0,b[7:0]}+carry_in;
    end
end

always@(posedge clk)
begin
    if(rst||refresh[2])
    begin
        pipe2_valid<=1'b0;
    end
    else if(pipe2_allowin)
    begin
        pipe2_valid<=pipe1_to_pipe2_valid;
    end
    if(pipe1_to_pipe2_valid&&pipe2_allowin)
    begin
        {carry_out_t2,sum_out_t2}<={{1'b0,a[15:8]}+{1'b0,b[15:8]}+carry_out_t1,
            sum_out_t1};
    end
end

always@(posedge clk)
begin
    if(rst||refresh[3])
    begin
        pipe3_valid<=1'b0;
    end
    else if(pipe2_allowin)
    begin
        pipe3_valid<=pipe2_to_pipe3_valid;
    end
    if(pipe2_to_pipe3_valid&&pipe3_allowin)
    begin
        {carry_out_t3,sum_out_t3}<={{1'b0,a[23:16]}+{1'b0,b[23:16]}+
            carry_out_t2,sum_out_t2};
    end
end

always@(posedge clk)
begin
    if(rst||refresh[4])
    begin
        pipe4_valid<=1'b0;
```

```
        end
        else if(pipe4_allowin)
        begin
            pipe4_valid<=pipe3_to_pipe4_valid;
        end
        if(pipe3_to_pipe4_valid&&pipe4_allowin)
        begin
            {carry_out,sum_out}<={{1'b0,a[31:24]}+{1'b0,b[31:24]}+carry_out_t3,
                sum_out_t3};
        end
    end
endmodule
```

E 32bit 流水线全加器仿真文件代码

```
module pipeline_adder_tb();
reg clk;
reg rst;
reg validin;
reg [31:0] a;
reg [31:0] b;
reg carry_in;
reg out_allow;
reg [4:1] suspend;
reg [4:1] refresh;
wire validout;
wire [31:0] sum_out;
wire carry_out;

always #25 clk=~clk;
//在下行沿进行数据读入，读入的数据在紧随其后的上行沿参与运算
//X表示该输入可以为任意值，对运算结果没有任何影响（无效输入）
initial
begin
    clk<=1'b0;
    rst<=1'b0;
    validin<=1'b1;
    out_allow<=1'b1;
    a<=32'bXXXXXXXX_XXXXXXXX_XXXXXXXX_00000001;//第一次加法第一级读入
    b<=32'b00000000_00000000_00000000_00000000;
    carry_in<=1'b0;
    suspend<=4'b0000;
    refresh<=4'b0000;
    #50
    a<=32'bXXXXXXXX_XXXXXXXX_00000001_00000010;//第二次加法第一级读入
    #50
    a<=32'bXXXXXXXX_00000001_00000010_00000011;//第三次加法第一级读入
```

```
        #50
        a<=32'b00000001_00000010_00000011_00000100;//第一次加法结束，第四次加法第一
            级读入
        #50
        suspend<=4'b0001;//流水线第二级暂停
        a<=32'b00000010_00000011_XXXXXXXX_XXXXXXXX;//第二次加法结束，第三次加法运算
            至第三级，第四次加法第二级运算暂停，读入无效,第五次加法第一级读入无效
        #50
        a<=32'b00000011_XXXXXXXX_XXXXXXXX_XXXXXXXX;//第三次加法结束，第三级读入无
            效，第四次加法第二级读入无效且运算暂停，第五次加法第一级读入无效
        #50
        suspend<=4'b0000;//流水线恢复流动
        a<=32'bXXXXXXXX_XXXXXXXX_00000100_00000101;//第三、四级读入无效，第四次加法
            运算至第二层，第五次加法第一级读入
        #50
        a<=32'bXXXXXXXX_00000100_00000101_00000110;//第四级读入无效，第四级加法运算
            至第三层，第五次加法运算至第二层，第六次加法第一级读入
        #50
        a<=32'b00000100_00000101_00000110_00000111;//第四次加法结束，第七次加法第一
            级读入，流水线完全恢复正常运转
        #50
        a<=32'b00000101_00000110_00000111_00001000;//第五次加法结束，第七次加法运算
            至第二级，第八次加法第一级读入
        #50
        refresh<=4'b0100;//流水线第三级清空
        a<=32'b00000110_00000111_00001000_XXXXXXXX;//第六次加法结束，第七次加法运算
            至第三级被清空,第八次加法运算至第二级
        #50
        refresh<=4'b0000;//流水线第三级不清空
        a<=32'b00000111_00001000_XXXXXXXX_XXXXXXXX;//第七次加法无有效输出结果，流水
            线继续输出第六次加法结果，第八次加法运算至第三层
        #50
        a<=32'b00001000_XXXXXXXX_XXXXXXXX_XXXXXXXX;//第八次加法结束并正常输出运算结
            果
        #50
        a<=32'bXXXXXXXX_XXXXXXXX_XXXXXXXX_XXXXXXXX;
    end

    Pipeline_adder myadder(clk,rst,validin,a,b,carry_in,out_allow,suspend,refresh,
        validout,sum_out,carry_out);
endmodule
```