《计算机组成原理》实验报告

年级、专业、班级	2022 级信息安全 01 班、2022 级信息安全 02 班	**姓名**	杨小艺、姚凡
实验题目	实验一简单流水线与运算器实验		
实验时间	2024 年 4 月 20 日	**实验地点**	DS1410
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: 谭玉娟

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 5 月 1 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

设计实现双输入 32 位 ALU 模块，根据控制信号 op 实现无符号数 +、-、与、或、非和置位操作。根据实验要求，当 op 信号为 000 时，进行加操作；当 op 信号为 001 时，实现减操作；当 op 信号为

010 时，进行与操作；当 op 信号为 011 时，进行或操作；当 op 信号为 100 时，进行非操作；当 op 信号为 101 时，进行置位操作。由于实验开发板开关数量有限，实际过程将 A 输入设置为 8 位二进制数，经过位数扩展进行 32 位运算，并内置 B 为内置的 32 位十六进制数 32'h1.

2.1.2 接口定义

表 2: 接口定义模版

信号名	方向	位宽	功能描述
A	input	32-bits	操作数 1
B	input	32-bits	操作数 2
op	input	3-bits	操作码
result	output	32-bits	运算结果

2.1.3 逻辑控制

在对 A 进行相应的无符号扩展后，将其与内置操作数 B 进行 +、-、按位与、按位或、取非运算对于 SLT 操作，用 if/else 语句，当 A>B 时置零，反之置一。具体逻辑控制见下 ALU 硬件图：

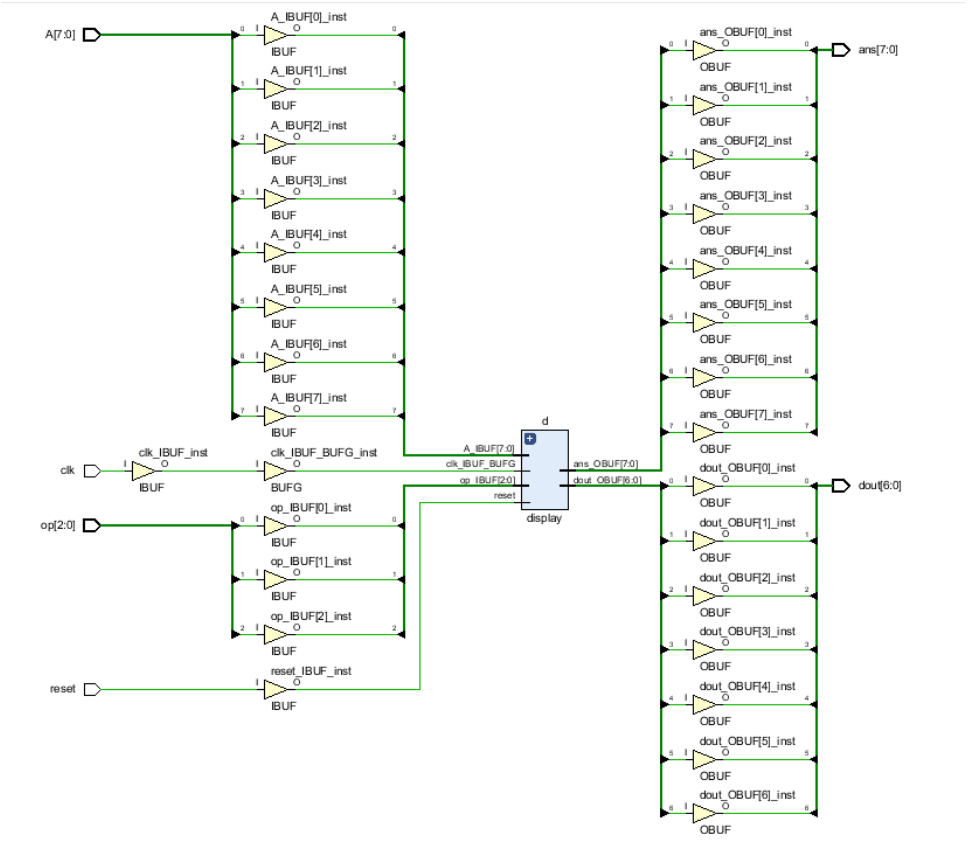

图 1: ALU 硬件图

2.2 有阻塞 4 级 8bit 全加器

2.2.1 功能描述

流水线的作用在于将大化成小、并行计算。若有 100 组 a 和 b 相加，常规串行加法器需要 100*4T 个时钟周期，而流水线加法器只需要 (100+3)T 个时钟周期。由此可见，流水线可以提高运算效率和系统吞吐率。由实验要求可知，该流水线加法器要求具有暂停和刷新功能，确保流水线可以随时暂停或刷新，且每一级的运算会受到前一级暂停的影响。实验要求设计有阻塞的 4 级 32bits 全加器，每一级进行 8bits 加法运算，分为 4 级分别进行计算：第 1 级计算 a[7:0]+b[7:0] 的结果，第 2 级计算 a[8:15]+b[8:15] 的结果，第 3 级计算 a[16:23]+b[16:23] 的结果，第 4 级计算 a[24:31]+b[24:31] 的结果。

2.2.2 接口定义

如下表表 3 所示：

表 3: 32bits 流水线全加器接口定义

信号名	方向	位宽	功能描述
a	Input	32-bits	操作数 a
b	Input	32-bits	操作数 b
clk	Input	1-bit	时钟信号
clear	Input	4-bits	刷新信号
stop	Input	4-bits	暂停信号
cin	Input	1-bit	初始输入进位信号
out_allow	Input	1-bit	是否允许输出
validIn	Input	1-bit	输入是否有效
validOut	Output	1-bit	输出是否有效
dout	Output	1-bit	结果的进位
sum	Output	32-bits	计算结果

2.2.3 逻辑控制

in1,in2,in3,in4 记录每一级流水线的进位；寄存器 pipeValid 记录每一级流水线是否还有效；寄存器 pipeReadyGo 记录是否可以用于下一级的刷新，如果没有暂停就可以继续；寄存器 pipe_allow_in 表示当前 pipe 中的值无效或准备好输出且下一级允许输入，则该级流水线允许输入；寄存器 pipe_to_next_valid 表示如果当前输入有效且该级流水线允许传输，则当前流水线能进入下一级。

在 always 语句块中，若 clear = 1，表示刷新，该级流水线中的值不再有效，只影响该级的结果，

而不影响其他周期的运算结果;若输入端允许输入,即 pipe_allow_in[n]=1,则 pipe 的值有效;如果流水线允许输入且值有效,则进行该级流水线的计算,将该级两操作数对应的 8 位和上一级的进位相加,得到新的进位和该级流水线计算结果 sum,每一级分别用 sum1,sum2,sum3,sum4 进行记录,最后的流水线输出结果为 sum4,进位为 in4;若某一级被暂停,那么前面的几级会先执行完当前操作然后暂停,后面的流水线会暂停。

设计流水线应采用非阻塞赋值,若用阻塞赋值则会在同一周期计算出结果,不符合流水线思想。

3 实验过程记录

3.1 ALU 实验过程记录及问题描述

实验过程 1:编写顶层模块 top

顶层模块需要调动 ALU 模块和 display 模块,并写明相应输入输出。

```
module top(
  input wire clk,reset,
  input [7:0]A,
  input [2:0]op,
  output [6:0]dout,
  output  [7:0]ans
    );
    wire [31:0] B=32'h1;
    wire [31:0] result;
    ALU my_alu(.A(A),.B(B),.op(op),.result(result));

    //wire [6:0]seg;
    display d(.clk(clk),.reset(reset),.s(result),.seg(dout),.ans(ans));
endmodule
```

图 2: 顶层模块代码展示

实验过程 2:编写 ALU 模块

ALU 模块是整个实验 1 的核心代码,它决定了不同输入的操作码 op 进行什么样的操作并输出操作数经过运算之后的结果。

```
module ALU(
  input wire [7:0] A,
  input wire [31:0] B,
  input [2:0] op,
  output reg [31:0] result
    );
   reg[31:0] sign_extend ;
   always@(*) begin
```

图 3: ALU 模块展示

实验过程 3:编写展示 display 模块和七段数码管模块 s7

display 模块和七段数码管模块 s7 用于上板显示操作数经过 ALU 模块运算后得到的结果。

从而验证实验结果的正确性。

```
module display(
    input wire clk,reset,
    input wire [31:0]s,
    output wire [6:0]seg,
    output reg [7:0]ans
    );
    reg [20:0]count;
    reg [3:0]digit;
    always@(posedge clk,posedge reset)
    if(reset)
        count = 0;
    else
        count = count + 1;

    always @(posedge clk)
    case(count[20:18])
```

图 4: display 模块展示

```
module s7(
input [3:0] num,
output reg [6:0] dout
    );
    always @(*) begin
    case(num)
    4'h0:dout=7'b000_0001;
    4'h1:dout=7'b100_1111;
    4'h2:dout=7'b001_0010;
    4'h3:dout=7'b000_0110;
    4'h4:dout=7'b100_1100;
    4'h5:dout=7'b010_0100;
    4'h6:dout=7'b010_0000;
    4'h7:dout=7'b000_1111;
    4'h8:dout=7'b000_0000;
    4'h9:dout=7'b000_0100;
```

图 5: s7 模块展示

问题描述 1: 由于有一输入为内置，误以为内置设置在 ALU 模块中导致最终 ALU 模块实现只需要一输入，与实验要求不符。

解决方案: 实验过程中发现不对劲，反复阅读实验要求发现虽然上板只有一个输入，但 ALU 仍然需要两个输入，且内置是设置在 top 模块中。

问题描述 2: 上板时发现七段数码管数字没有反应。

解决方案: 经过多次测试和对代码的检验，最后发现问题并不出在代码部分，而是在分配管脚时将操作码 op 的管教分配错误，导致操作码 op 无法被改变，导致结果始终为零，七段数码管不变。

问题描述 3: 在 top 文件综合时出现无法综合的情况。

解决方案:最终发现是由于调用 display 模块时未进行实例化,导致综合无法实现,最终修改如图 6:

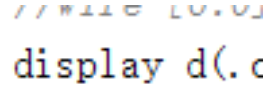

图 6: display 模块实例化添加

3.2 32bits 流水线的实验过程及问题解决

实验过程 1:相应的基本定义

32bits 的流水线加法器首先需要充分的各方面的定义,如输入的值;刷新、暂停信号;输入输出的值是否有效的判断;进位;结果等。

实验过程 2:各级加法器实现

32bits 的流水线加法器由 4 个 8bits 的流水线加法器构成,每级流水线的设计逻辑基本一致。

实验过程 3:实验问题解决

问题描述 1:在仿真的过程中,为了实现流水线的特殊流水的功能,需要每一个周期都进行一次输入,当数据输入后的四个周期输出结果,但是仿真得到的结果中,第一次输入的两个数据的结果无法显示。

解决方案:在长时间观察与思考后,我发现自己是在每一次的上升沿进行运算,而在仿真的时候在上升沿的时侯输入的第一个数据,导致第一组数据的计算结果应该在第四次上升沿的时候显示,但是四个周期才会显示结果,而第四个完整的周期是在第五个上升沿的位置,因此第一组数据结果无法显示。要解决这个问题,只需要在下降沿的时候输入数据即可。

问题描述 2:流水线计算结果没有延迟与实际不符合

解决方案:因为在实际流水线中,计算是并行的,各个部件独立运算互不影响,运算结束后将结果放入寄存器,到下一周期被刷新重新计算。因此,需要将 always 语句中的赋值方法修改为非阻塞式赋值。

问题描述 3:在仿真时,发现仿真文件中的某些数据并不能很好地体现 32bits 流水线加法器的特点。

解决方案::在选取仿真的数据的时候,应该选择典型数据,比如连续四个周期均有数据输入才能展现流水线加法的特点。

问题描述 4:其中一级流水线暂停是否会影响其余流水线。

解决方案:流水线中信号只能从前一级向后一级传递,无法从后一级向前一级传递。因此采取舍弃策略,让前几级照常运算,后几级运算的值将前面的值覆盖,若有暂停,则前几级流水线照

常工作。

4 实验结果及分析

4.1 ALU 验证实验结果

操作	Num1	Result
A + B(Unsigned)	8’b00000010	32’h00000003
A - B	8’b11111111	32’h000000fe
A AND B	8’b11111110	32’h00000000
A OR B	8’b10101010	32’h000000ab
$\overline{A}$	8’b11110000	32’hfffffff0f
SLT	8’b10000001	32’h00000000

表 4: ALU 结果表

4.2 ALU 仿真图

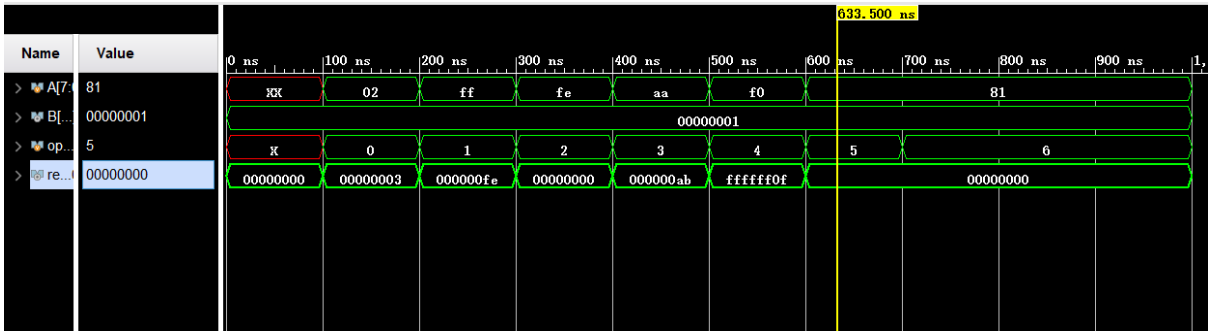

图 7: ALU 仿真图

4.3 ALU 上板实操

接下来将演示在 bit 流文件下载到实验板后，拨动按钮，输入不同操作数和操作码，七段数码管显示的结果。操作分别包括加操作，减操作，与操作，或操作，非操作以及 SLT 操作。

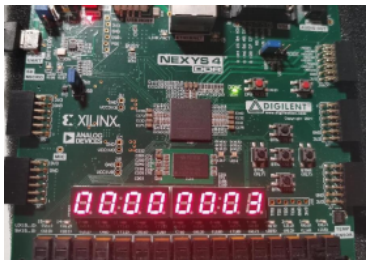

图 8: 加操作

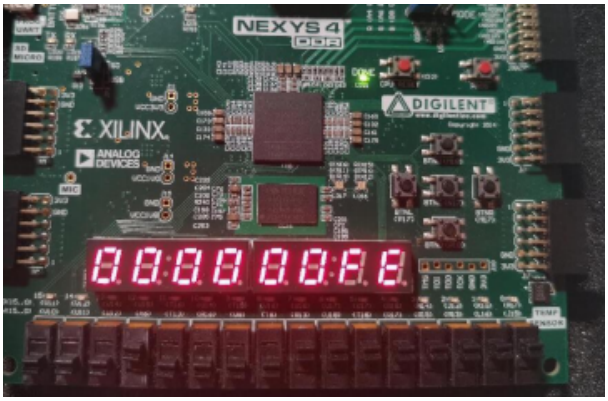

图 9: 减操作

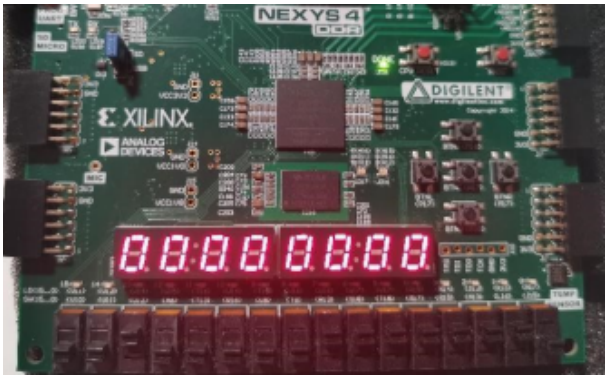

图 10: 与操作

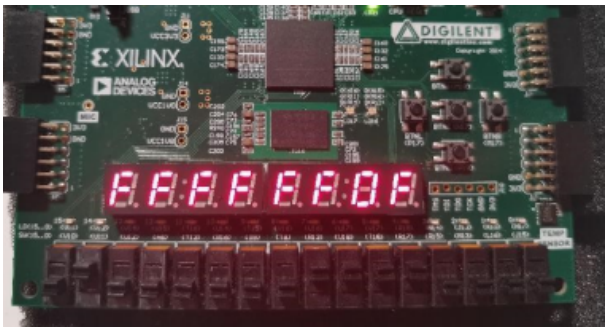

图 11: 非操作

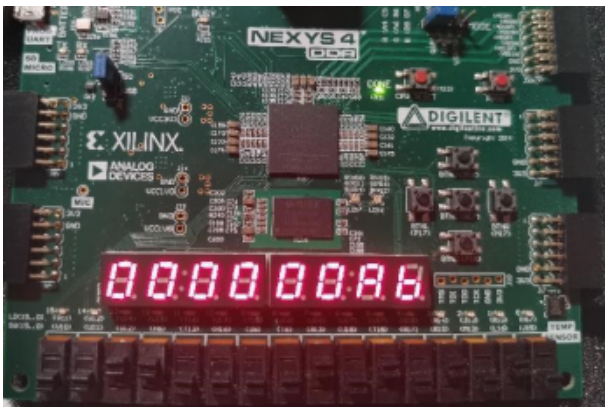

图 12: 或操作

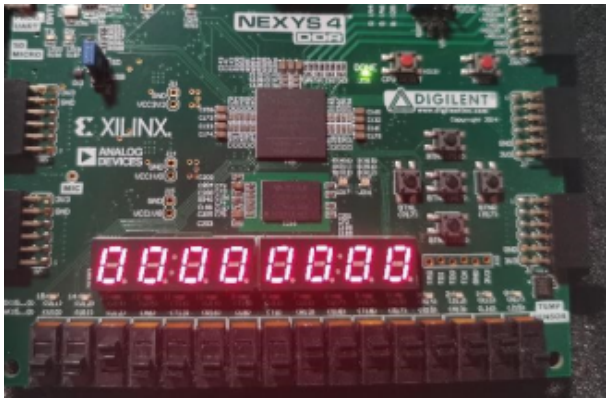

图 13: SLT 操作

4.4 流水线阻塞(暂停)仿真图

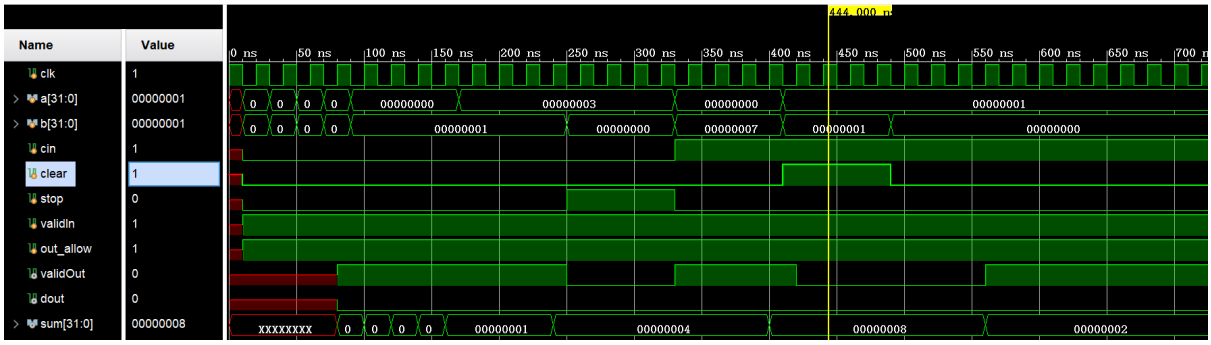

图 14: 流水线阻塞(暂停)仿真图

4.5 流水线刷新(清空)仿真图

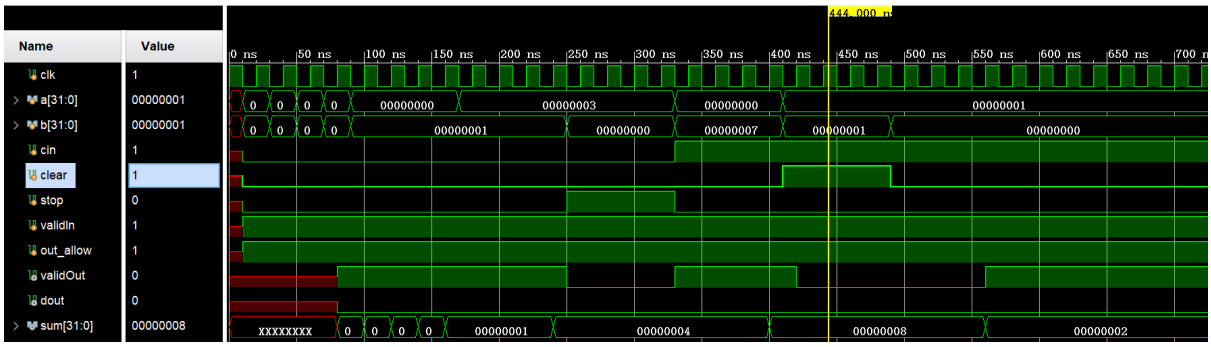

图 15: 流水线刷新(清空)仿真图

A ALU 代码

```
‘timescale 1ns / 1ps
module ALU(
input wire [7:0] A,
input [31:0] B,
```

```
input [2:0] op,
output reg [31:0] result
    );

   reg[31:0] sign_extend ;
   wire B=32'h01;

   always@(op) begin
   sign_extend={24'h0000000,A[7:0]};
   case(op)
   3'b000:result=sign_extend+B;
   3'b001:result=sign_extend-B;
   3'b010:result=sign_extend&B;
   3'b011:result=sign_extend|B;
   3'b100:result=~sign_extend;
   3'b101:
   begin
   if(sign_extend>B) result=32'b0;
   else result=32'b1;
   end
   default: result=32'b0;
   endcase
   end

endmodule
```

B 32bit 流水线全加器代码

```
`timescale 1ns / 1ps
module stallable_pipeline_adder(
    input clk,
    input [31:0] a,
    input [31:0] b,
    input cin,  //初始进位信号
    input clear,  //刷新
    input stop, //暂停
    input validIn,  //输入值是否有效
    input out_allow,    //是否允许输出

    output validOut,    //输出是否有效
    output dout,    //结果进位
    output [31:0] sum    //结果
    );

    reg in1,in2,in3,in4;    //每级流水线的进位
    reg [3:0] pipeValid;    //记录流水线是否有效
```

```
wire [3:0] pipeReadyGo;
wire [3:0]pipe_allow_in;
wire [2:0] pipe_to_next_valid;

//一级流水线
reg [7:0] sum1; //低8位的和
assign pipeReadyGo[0] = !stop;
assign pipe_allow_in[0] = (!pipeValid[0]) || (pipeReadyGo[0] && pipe_allow_in
    [1]);
assign pipe_to_next_valid[0] = pipeValid[0] && pipeReadyGo[0];

always @(posedge clk) begin
    if(clear)begin
        pipeValid[0] <= 1'b0;
        //如果刷新，则pipe中的值不再有效
    end
    else if(pipe_allow_in[0])begin
        pipeValid[0] <= validIn;
        //如果输入端允许输入，则第一个pipe的值有效，否则无效
    end
    if(validIn && pipe_allow_in[0])begin
    //如果允许输入且值有效
        {in1,sum1} <= {1'b0,a[7:0]} + {1'b0,b[7:0]} + cin;
    end
end

//二级流水线
reg [15:0] sum2;     //第15~8位的和，同时拼接低8位的和
assign pipeReadyGo[1] = !stop;
assign pipe_allow_in[1] = (!pipeValid[1]) || (pipeReadyGo[1] && pipe_allow_in
    [2]);
assign pipe_to_next_valid[1] = pipeValid[1] && pipeReadyGo[1];

always @(posedge clk) begin
    if(clear)begin
        pipeValid[1] <= 1'b0;
    end
    else if(pipe_allow_in[1])begin
        pipeValid[1] <= pipe_to_next_valid[0];
    end
    if(pipe_to_next_valid[0] && pipe_allow_in[1])begin
        {in2,sum2} <= {{1'b0,a[15:8]} + {1'b0,b[15:8]} + in1 , sum1};
    end
end

//三级流水线
reg [23:0] sum3;
//第23~16位的和，同时拼接低16位的和
assign pipeReadyGo[2] = !stop;
```

```
    assign pipe_allow_in[2] = (!pipeValid[2]) || (pipeReadyGo[2] && pipe_allow_in
        [3]);
    assign pipe_to_next_valid[2] = pipeValid[2] && pipeReadyGo[2];

    always @(posedge clk) begin
        if(clear)begin
            pipeValid[2] <= 1'b0;
        end
        else if(pipe_allow_in[2])begin
            pipeValid[2] <= pipe_to_next_valid[1];
        end
        if(pipe_to_next_valid[1] && pipe_allow_in[2])begin
            {in3,sum3} <= {{1'b0,a[23:16]} + {1'b0,b[23:16]} + in2 , sum2};
        end
    end

    //四级流水线
    reg [31:0] sum4; //第32~24位的和, 同时拼接低24位的和
    assign pipeReadyGo[3] = !stop;
    assign pipe_allow_in[3] = (!pipeValid[3]) || (pipeReadyGo[3] && out_allow);

    always @(posedge clk) begin
        if(clear)begin
            pipeValid[3] <= 1'b0;
        end
        else if(pipe_allow_in[3])begin
            pipeValid[3] <= pipe_to_next_valid[2];
        end
        if(pipe_to_next_valid[2] && pipe_allow_in[3])begin
            {in4,sum4} <= {1'b0,a[31:24]} + {1'b0,b[31:24]} + sum3;
        end
    end

    assign validOut = pipeValid[3] && pipeReadyGo[3];
    assign sum = sum4;
    assign dout = in4;
endmodule
```