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 32 位 ALU，实现（无符号）加、减、与、或、非、小于则置位的操作，分别对应使能信号 op 为 000、001、010、011、100、101。由于 Nexy4 开发板拨码开关数量限制，实际过程将 num1 输入为 8

位二进制数，通过位数扩展进行 32 位运算。并且根据要求，num2 为内置的 32 位二进制数 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
num1	Input	32-bits	操作数 1
num2	Input	32-bits	操作数 2
op	Input	3-bits	操作码
result	Output	32-bits	计算结果

2.1.3 逻辑控制

对于扩展后的 num1 和 num2 直接运用 +、-、&、|、~ 运算符进行加、减、按位与、按位或、取非的运算，用 if...else... 判断实现 SLT 运算，通过比较 num1 和 num2 的大小关系选择置 0 或 1。硬件图如下：

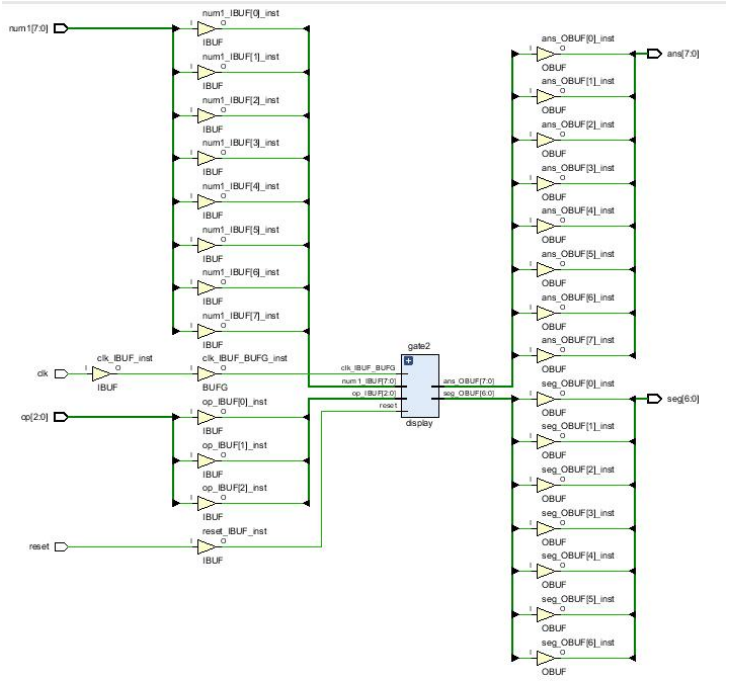

图 1: ALU 硬件图

2.2 有阻塞 4 级 8bit 全加器

2.2.1 功能描述

设计有阻塞的 4 级 32bits 全加器，每一级进行 8bits 加法运算。流水线的作用在于将大问题分解成小问题，再对小问题并行计算。如果有 10000 组 a 和 b 相加，常规串行加法器需要 10000*4T 个时钟周期，而流水线只需要 (10000+3)T。因此，流水线可以提高运算效率和系统吞吐率。同时，根据实验要求，该加法器具有暂停和刷新功能，以确保流水线可以随时暂停或刷新。

将 32 位数据分为 4 级分别计算，第 1 级计算 a[7:0]+b[7:0]，第 2 级计算 a[8:15]+b[8:15]，第 3 级计算 a[16:23]+b[16:23]，第 4 级计算 a[24:31]+b[24:31]，其中每级之间加入流水线寄存器，存储进位、还没加的数据以及已经加得的和。在每个 clk 上升沿读入数据，经过 4 个 clk 后输出加法运算结果和溢出位。

2.2.2 接口定义

表 3: 接口定义模版

信号名	方向	位宽	功能描述
clk	Input	1-bits	时钟信号
refresh	Input	4-bits	刷新键
halt	Input	4-bits	暂停键
cin_a	Output	32-bits	操作数 a
cin_b	Output	32-bits	操作数 b
c_in	Output	1-bits	低位的进位
c_out	Output	1-bits	和的进位
sum_out	Output	32-bits	和

2.2.3 逻辑控制

将 32bits 加法分割为 4 级，在每级之间插入流水线寄存器暂存中间数据：每一级使用 3 个寄存器分别保存存储进位、还没加的数据以及已经加得的和，传给下一级进行计算。

每一级用到的流水线寄存器有 c_out_12、c_out_23、c_out_34，用于存储进位；sum1、sum2、sum3, 用于存储当前一级计算得结果；tempa1、tempb1、tempa2、tempb2、tempa3、tempb3，用于存储该级还没用到的数据位。

此外，此流水线加法器还具有暂停和刷新功能。暂停信号 halt 为 4 位，每位控制一级。若某一级被暂停，则前面的几级会先执行完当前操作然后暂停，后面几级会执行完毕然后暂停。通过将暂停一级的输出全部赋为 Z，再对后面接收到 Z 信号的几级流水线判断并赋 Z 信号，实现暂停。刷新信号 refresh 为 4 位，每位控制一级。若某一级被刷新，则该级的数据全部清零，只影响这一级接受数据周期的一组结果，其他周期的运算结果不受影响。硬件电路图如下：

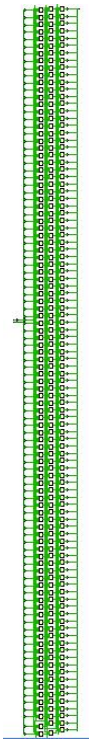

图 2: ALU 硬件图

图中存在相当多的并行逻辑器件，可见流水线在提高运算效率和吞吐率的同时也需要大量寄存器和算术逻辑单元的支持。

3 实验过程记录

3.1 问题 1:ALU 模块操作数计算错误

问题描述：在写 ALU 主模块 alu 时，忘记对输入的 8 位 num1 进行位数扩展，导致其无法与 32 位二进制数 num2 进行运算。

解决方案：给 num1 进行位数扩展：assign sign_extend_num1={24'h0,num1};

3.2 问题 2:ALU 模块之间连接不上

问题描述：写完 ALU 的主模块 alu 并导入给定 display 模块后，将其一起写入 top 设计文件然后综合，结果发现端口接线数匹配不上。

解决方案：给定 display 文件的输出端口 ans 数量定义错误，8 位输出应该是 [7:0] 而不是 [8:0]

3.3 问题 3:ALU 模块中 SLT 运算判断

问题描述：一开始想通过减法运算 num1-num2 判断 num1、num2 的大小，但由于实验要求此过程中全部是无符号数运算，因此 num1-num2 得到的结果也是无符号数，无法与 0 比较，想用减法进行判断相当麻烦。

解决方案：考虑使用 if...else 语句进行判断 num1 和 num2 大小，方便快捷。

3.4 问题 4:ALU 实验中无法综合

问题描述：由于 top 文件是调用了 alu 和 display 两个子模块，不同模块之间的数据传递需要用 wire 连通，一开始误将数据存进 reg 进行数据传递，后来发现无法综合。

解决方案：将 reg 改为 wire，于是可综合。这也符合硬件电路的实际，用导线将两个模块相连，而不是寄存器。

3.5 问题 5: 流水线实验中数据无法保存

问题描述：由于是流水线计算，一次加法需要 4 个周期，而每一周期都会输入新的数据覆盖寄存器中之前的数据，于是前一周期还没计算完成的数据需要特殊保存。

解决方案：在每一级流水线之间设置流水线寄存器，用来存储当前及其前面的流水线计算得到的结果、还没用到的几位数据和当前流水线的进位，于是下一级流水线可以使用流水线寄存器里的数据而不会缺失。

3.6 问题 6: 流水线实验中暂停、刷新恢复前流水线的控制

问题描述：实验要求现对各级分别暂停和刷新，但如何控制暂停/刷新后的流水线。如：流水线暂停后仍有数据传入，此时如何控制流水线使其暂停不工作。

解决方案：原本考虑加入 valid 记录两级之间是否可以传送数据，后来发现要考虑的变量太多，是否可接收、是否可发送、是否暂停、是否刷新等等。不妨采用 if...else 判断，若某一级被暂停，则前面的几级会先执行完当前操作然后暂停，后面几级会执行完毕然后暂停。此处将暂停一级的流水线输出赋为 Z，则后面几级流水线接受到的数据都是 Z，通过 if 语句判定，若流水线寄存器传入的数据为 Z，则输出也赋为 Z，由此将后面的流水线全部暂停。若某一级被刷新，则该级的数据全部清零，只影响一次结果。

3.7 问题 7: 流水线实验中扩展运算超限

问题描述：为了简便处理进位与和，采用拼接运算符进行运算：c_out_34,sum3<=’bz。但最后一级流水线的 sum 为 32 位，拼接后 32 位超出限制。

解决方案：分开赋值：c_out<=’bz;sum_out<=’bz;

3.8 问题 8: 流水线计算结果没有延迟，不符合实际

问题描述：流水线加法器采用流水线思想，将 32 位加法分为 4 级分别计算，计算结果有 4 级延迟，而一开始仿真结果显示没有延迟。

解决方案：将 always 语句内部的赋值语句改为非阻塞赋值，与实际相符。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，因此应该采用非阻塞赋值语句。

3.9 问题 9: 流水线某一级暂停时如何处理前面几级流水线

问题描述：当流水线加法器某一级暂停时，可以通过对输出值赋 Z 的操作对后面几级流水线进行暂停，但前几级并不好暂停，设计时也没考虑到，因此发生错误。

解决方案：一开始考虑设置 valid 为前几级提供信号判断，但由于流水线的原因，信号无法从后一级流水线向前一级传递。最后采用舍弃策略，前几级照常运算，后运算的值将前面的值覆盖，若暂停情况下仍有输入，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’hFFFFFF0F
SLT	8’b10000001	32’h00000000

表 4: ALU 结果表

根据实验要求，Num2 内置为 1，用 ALU 分别计算 A + B、A - B、A AND B、A OR B、$\overline{A}$、SLT，所得结果与预期一致。仿真结果如图 3，上板子结果如图 4~10。其中图 2 为初始状态，即全部为 0，此时 A=0，B=1，op=000（即 Add 操作），因此结果为 4，图 5~10 为加、减、与、或、非、小于则置位的运算，与预期一致。

4.2 流水线阻塞（暂停）仿真图

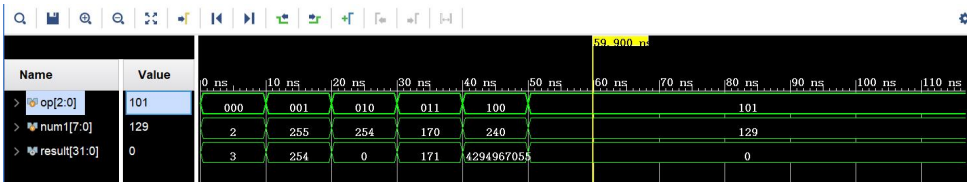

图 3: ALU 仿真结果

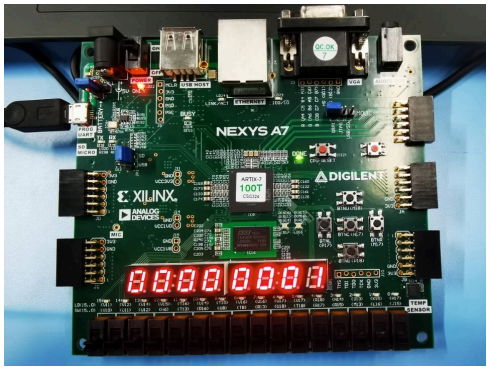

图 4: 初始状态

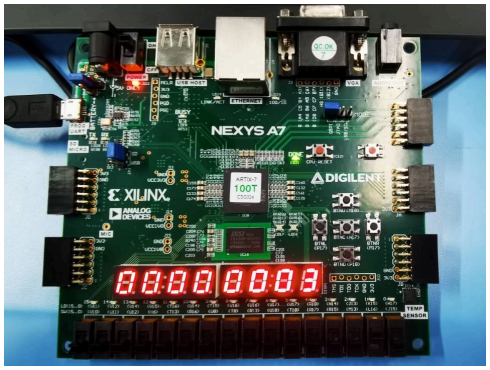

图 5: add 运算

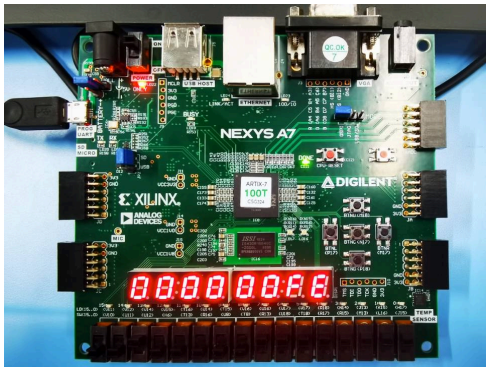

图 6: sub 运算

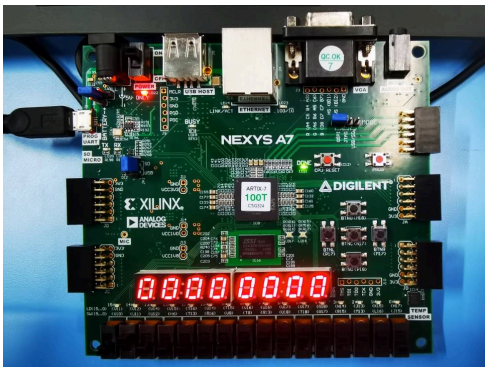

图 **7:** and 运算

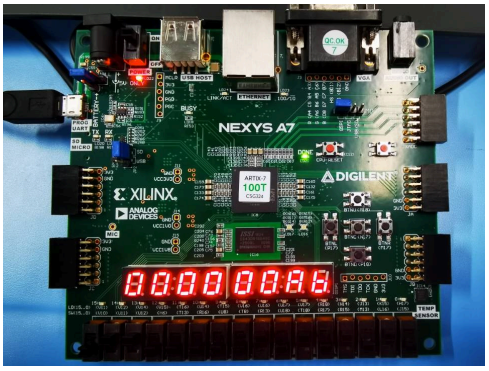

图 **8:** or 运算

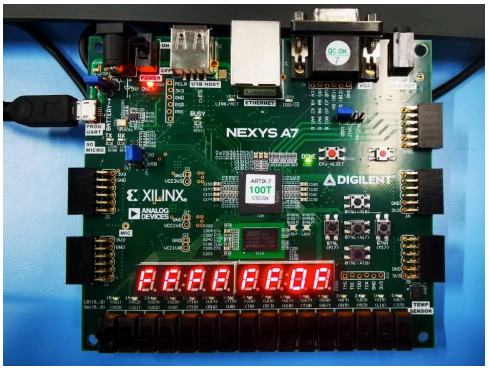

图 **9:** not 运算

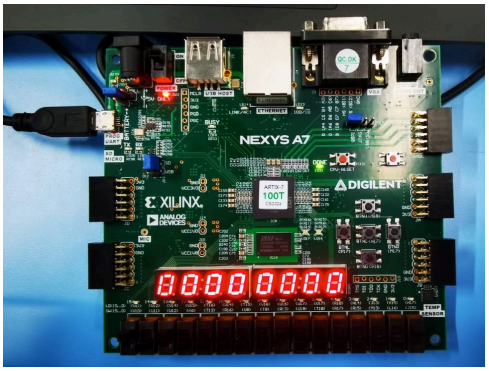

图 **10:** SLT 运算

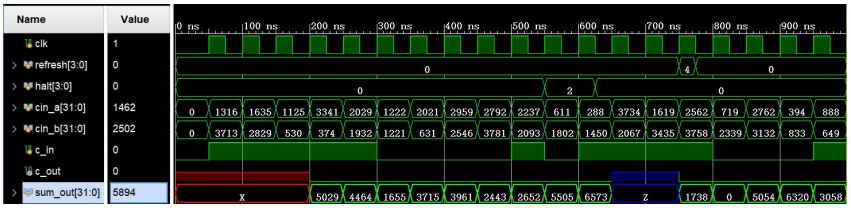

图 **11:** 流水线加法器仿真

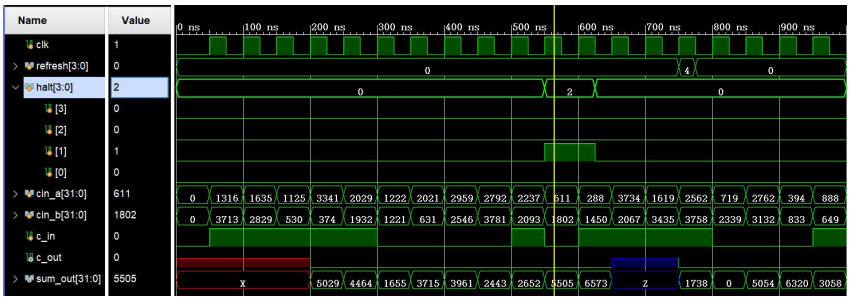

图 **12:** 流水线阻塞仿真图

4.3 流水线刷新(清空)仿真图

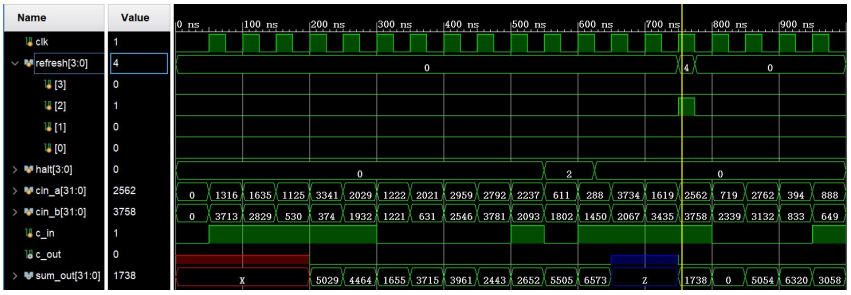

图 **13:** 流水线刷新仿真

A ALU 代码

```
‘timescale 1ns / 1ps

module alu(
  input wire [2:0] op,
  input wire [7:0] num1,
  output reg [31:0] result
);
  wire [31:0] num2;
  wire [31:0] sign_extend_num1;
  assign num2=32'h0000_0001;
  assign sign_extend_num1={24'h0,num1};

  always@(*)begin
    case(op)
      3'b000:result=sign_extend_num1+num2;
```

```
        3'b001:result=sign_extend_num1-num2;
        3'b010:result=sign_extend_num1&num2;
        3'b011:result=sign_extend_num1|num2;
        3'b100:result=~sign_extend_num1;
        3'b101:begin
          if(sign_extend_num1<num2)    result=32'h0000_0001;
          else                         result=32'h0000_0000;
        end
        default:result=32'hxxxx_xxxx;
      endcase
    end

endmodule
```

B 32bit 流水线全加器代码

```
module stallable_pipeline_adder(
  input clk,
  input [3:0] refresh,
  input [3:0] halt,
  input [31:0] cin_a,
  input [31:0] cin_b,
  input c_in,
  output reg c_out,
  output reg[31:0] sum_out
);
  reg c_out_12,c_out_23,c_out_34;
  //reg [7:0] sum1,sum2,sum3;
  reg [7:0] sum1;
  reg [15:0] sum2;
  reg [23:0] sum3;
  reg [23:0] tmpa1,tmpb1;
  reg [15:0] tmpa2,tmpb2;
  reg [7:0] tmpa3,tmpb3;
//  assign tmpa1=cin_a[31:8];

//    reg go1,go2,go3;
//    reg come2,come3,come4;
//    wire valid_12,valid_23,valid_34;
//    assign valid_12=go1&come2;
//    assign valid_23=go2&come3;
//    assign valid_34=go3&come4;

  // pipeline 1
  always@(posedge clk)begin
    if(halt[0])begin
      {c_out_12,sum1}<='bz;
```

```
        tmpa1<='bz;
        tmpb1<='bz;
      end
      else if(refresh[0]) begin
        {c_out_12,sum1}<=9'b0;
        tmpa1<=24'b0;
        tmpb1<=24'b0;
      end
      else begin
        {c_out_12,sum1}<=cin_a[7:0]+cin_b[7:0];
        tmpa1<=cin_a[31:8];
        tmpb1<=cin_b[31:8];
//              go1<=1;
      end
    end
    // pipeline 2
    always@(posedge clk)begin
      if(halt[1]) begin
        {c_out_23,sum2}<='bz;
        tmpa2<='bz;
        tmpb2<='bz;
      end
      else if(refresh[1]) begin
        {c_out_23,sum2}<=17'b0;
        tmpa2<=16'b0;
        tmpb2<=16'b0;
      end
      else begin
        if(c_out_12===1'bz) begin
          sum2[7:0]<='bz;
          {c_out_23,sum2[15:8]}<='bz;
          tmpa2<='bz;
          tmpb2<='bz;
        end
        else begin
          sum2[7:0]<=sum1;
          {c_out_23,sum2[15:8]}<=tmpa1[7:0]+tmpb1[7:0]+c_out_12;
          tmpa2<=tmpa1[23:8];
          tmpb2<=tmpb1[23:8];
//              go2<=1;
//              come2<=1;
        end
      end
    end
    // pipeline 3
    always@(posedge clk)begin
      if(halt[2]) begin
        {c_out_34,sum3}<='bz;
        tmpa3<='bz;
```

```
        tmpb3<='bz;
      end
      else if(refresh[2]) begin
        {c_out_34,sum3}<=25'b0;
        tmpa3<=8'b0;
        tmpb3<=8'b0;
      end
      else begin
        if(c_out_23===1'bz) begin
          sum3[15:0]<='bz;
          {c_out_34,sum3[23:16]}<='bz;
          tmpa3<='bz;
          tmpb3<='bz;
        end
        else begin
          sum3[15:0]<=sum2;
          {c_out_34,sum3[23:16]}<=tmpa2[7:0]+tmpb2[7:0]+c_out_23;
          tmpa3<=tmpa2[15:8];
          tmpb3<=tmpb2[15:8];
          //go3<=1;
          //come3<=1;
        end
      end
    end
    // pipeline 4
    always@(posedge clk) begin
      if(halt[3]) begin
        //{c_out,sum_out}<=33'bzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz;
        c_out<='bz;
        sum_out<='bz;
      end
      else if(refresh[3]) begin
        c_out<='b0;
        sum_out<='b0;
      end
      else begin
        if(c_out_34===1'bz) begin
          sum_out[23:0]<='bz;
          {c_out,sum_out[31:24]}<='bz;
        end
        else begin
          sum_out[23:0]<=sum3;
          {c_out,sum_out[31:24]}<=tmpa3[7:0]+tmpb3[7:0]+c_out_34;
          //come4<=1;
        end
      end
    end

endmodule
```