2021 春季学期

计算机组成原理

32 位 ALU

设

计

报

告

Hollow Man

ALU 模块设计报告

一、项目简述

设计语言：Verilog

硬件描述语言

仿真环境：Vivado

最大延迟：6.426ns

二、实现功能

算术逻辑运算单元(ALU)是数字计算机中执行加、减等算术运算，执行与、或、非等逻辑运算，以及执行比较、移位等操作的功能部件，本设计报告为 MIPS 单周期处理器的 ALU 模块的功能、接口、时序及其实现。该 ALU 在 Vivado 软件环境下进行功能仿真。

实现功能：

and	00000	100100	and $1,$2,$3	$1=$2 & $3	rd <- rs & rt ；其中 rs＝$2，rt=$3，rd=$1
or	00000	100101	or $1,$2,$3	$1=$2 | $3	rd <- rs | rt ；其中 rs＝$2，rt=$3，rd=$1
xor	00000	100110	xor $1,$2,$3	$1=$2 ^ $3	rd <- rs xor rt ；其中 rs＝$2，rt=$3, rd=$1(异或）
nor	00000	100111	nor $1,$2,$3	$1=~($2 | $3)	rd <- not(rs | rt) ；其中 rs＝$2，rt=$3, rd=$1(或非）

slt	00000	101010	slt $1,$2,$3	if($2<$3) $1=1 else $1=0	if (rs < rt) rd=1 else rd=0 ；其中 rs＝$2，rt=$3, rd=$1
sltu	00000	101011	sltu $1,$2,$3	if($2<$3) $1=1 else $1=0	if (rs < rt) rd=1 else rd=0 ；其中 rs＝$2，rt=$3, rd=$1 (无符号数)
sll	00000	000000	sll $1,$2,10	$1=$2<<10	rd <- rt << shamt ； shamt 存放移位的位数， 也就是指令中的立即数，其中 rt=$2, rd=$1
srl	00000	000010	srl $1,$2,10	$1=$2>>10	rd <- rt >> shamt ； (logical) ，其中 rt=$2, rd=$1
sra	00000	000011	sra $1,$2,10	$1=$2>>10	rd <- rt >> shamt ；(arithmetic) 注意符号位保留 其中 rt=$2, rd=$1
sllv	00000	000100	sllv $1,$2,$3	$1=$2<<$3	rd <- rt << rs ；其中 rs＝$3，rt=$2, rd=$1
srlv	00000	000110	srlv $1,$2,$3	$1=$2>>$3	rd <- rt >> rs ； (logical)其中 rs＝$3，rt=$2, rd=$1
srav	00000	000111	srav $1,$2,$3	$1=$2>>$3	rd <- rt >> rs ； (arithmetic) 注意符号位保留 其中 rs＝$3，rt=$2, rd=$1

三、设计方案

注：文档仅对实现原理和部分结构进行说明，具体实现请查看项目代码。

本设计共有 1 个主模块，5 个子模块，1 个测试模块

主模块：

算数逻辑运算单元 ALU（加、减等算术运算，执行与、或、非等逻辑运算，以及执行比较、移位）

子模块（供 ALU 调用）：

8 位加法器（按位实现）

32 位加法器（调用 8 位超前进位加法器实现）

32 位减法器（被减数取补码，调用加法器实现）

1 位移位器（按位实现）

2 位移位器（按位实现）

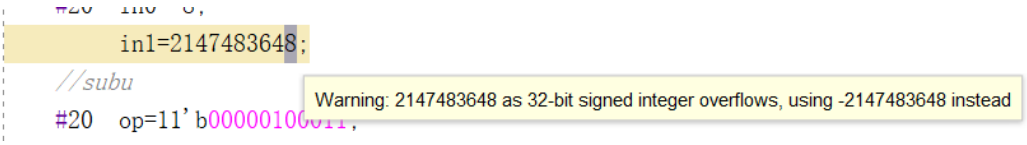
测试模块：

仿真测试，供 ALU 测试用

基本原理

（一）关于输入输出的说明

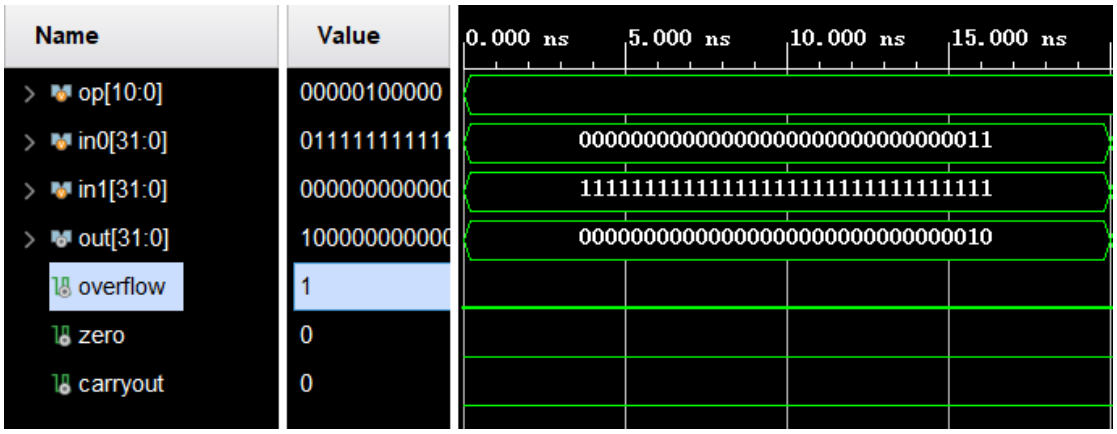
vivado 中，有符号数是采用补码的方式进行输入输出的（如下图，输入为 2^31 时已经溢出，为 2^31-1 时则会正常），因此不需要单独进行原码和补码之间的转换，为了便于观察，最终算术运算和比较结果将采用有符号十进制（Signed Decimal）和无符号十进制（Signed Decimal）展示，移位和逻辑运算采用二进制（Binary）展示。



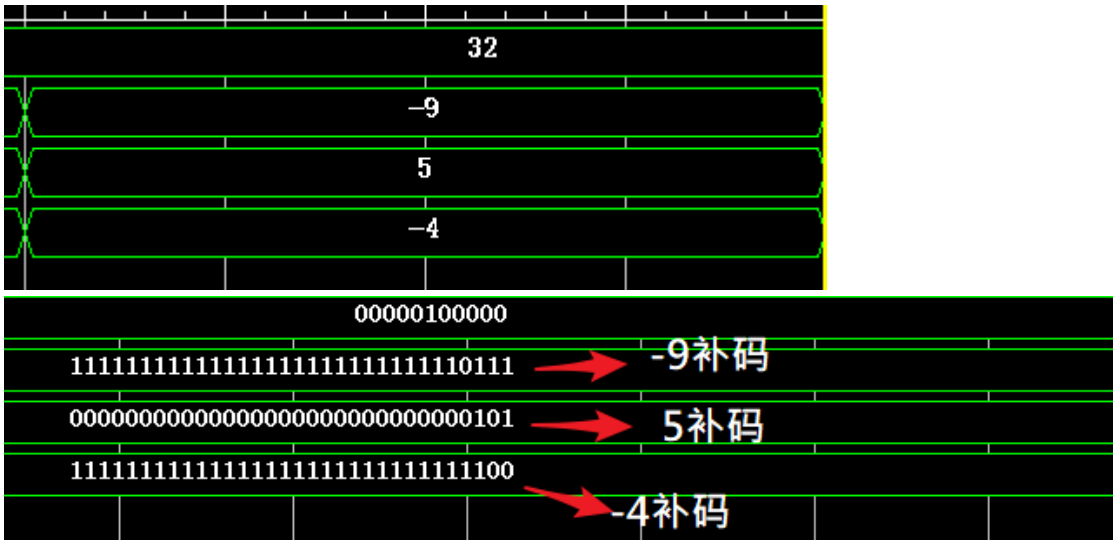
以有符号加法的两组输入为例进行解释：

```
        //add
          op=11'b00000100000;
          in0=32'b00000000000000000000000000000011; //3的补码
          in1=32'b11111111111111111111111111111111; //-1的补码
#20       in0=-9;
          in1=5;
```

第一组，采用二进制输入时，viavdo 会将其识别为有符号数的 3 和-1 ，并输出 2 的补码，使用有符号十进制查看即可得到 3+（-1）=2



第二组，采用十进制输入时，（-9）+5=（-4），使用二进制查看发现 vivado 会自动采用其补码形式进行计算，并输出-4 的补码



（二）算术运算

加法器：

加数 A	加数 B	本位 C	进位 D
0	0	0	0
0	1	1	0
1	1	0	1
1	0	1	0

由基本加法进位原理的真值表可知，本位 C 的值是加数 A 和加数 B 做异或得到的，而进位 D 的值是加数 A 与加数 B 做与得到的，故当两个数相加时，本位用异或门实现，而进位使用与门实现。

使用超前进位加法器：

$$C_{i+1} = (A_i \cdot B_i) + (A_i \cdot C_i) + (B_i \cdot C_i)$$
$$= (A_i \cdot B_i) + (A_i + B_i) \cdot C_i$$

设：

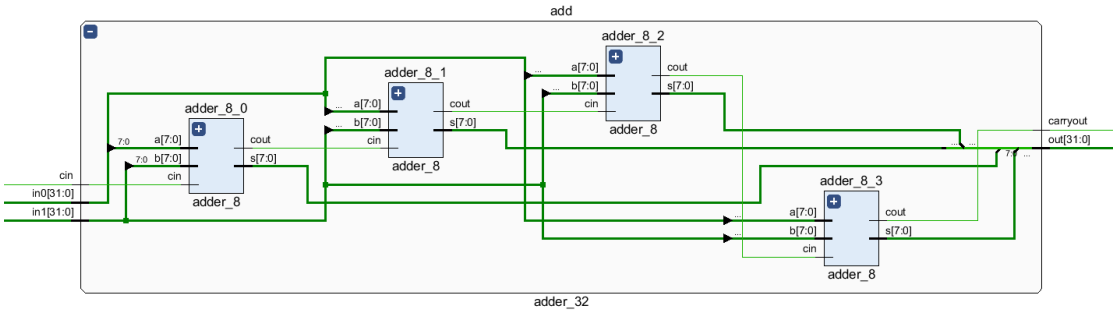
- 生成（Generate）信号：$G_i = A_i \cdot B_i$
- 传播（Propagate）信号：$P_i = A_i + B_i$

则：$C_{i+1} = G_i + P_i \cdot C_i$

- $C_1 = G_0 + P_0 \cdot C_0$
- $C_2 = G_1 + P_1 \cdot C_1$
  $= G_1 + P_1 \cdot (G_0 + P_0 \cdot C_0)$
  $= G_1 + P_1 \cdot G_0 + P_1 \cdot P_0 \cdot C_0$
- $C_3 = G_2 + P_2 \cdot C_2$
  $= G_2 + P_2 \cdot (G_1 + P_1 \cdot G_0 + P_1 \cdot P_0 \cdot C_0)$
  $= G_2 + P_2 \cdot G_1 + P_2 \cdot P_1 \cdot G_0 + P_2 \cdot P_1 \cdot P_0 \cdot C_0$
- $C_4 = G_3 + P_3 \cdot C_3$
  $= G_3 + P_3 \cdot (G_2 + P_2 \cdot G_1 + P_2 \cdot P_1 \cdot G_0 + P_2 \cdot P_1 \cdot P_0 \cdot C_0)$
  $= G_3 + P_3 \cdot G_2 + P_3 \cdot P_2 \cdot G_1 + P_3 \cdot P_2 \cdot P_1 \cdot G_0 + P_3 \cdot P_2 \cdot P_1 \cdot P_0 \cdot C_0$

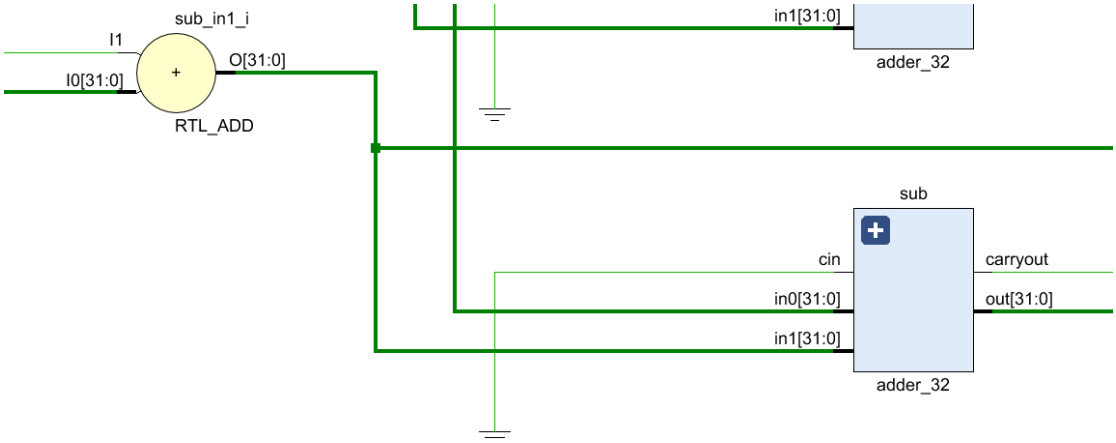
$C_{i+1} = G_i + P_i \cdot C_i$

由上式可以看出直接实现 32 位超前进位加法器会非常复杂，所以首先实现 8 位超前进位加法器，并通过 4 个 8 位超前进位加法器组成 32 位超前进位加法器。



减法器：

减法器原理与加法器原理相同，只需要将对应被减数取补码即可。



addu、subu：

addu：最高有效位向高位有进位，产生进位 carryout

subu：当被减数小于减数时，最高有效位向高位有借位，产生借位，将加法进位标志 carryout 取反即可。

add、sub：

[x]补+[y]补=[x+y]补

[x-y]补=[x]补-[y]补=[x]补+[-y]补

[-y 补]=~[y]补+1

add 溢出：两个正数相加产生负数 0111(7)+0110(6)=1101

两个负数相加产生正数 1000(-8)+1001(-7)=0001

一个正数与一个负数相加不产生溢出

sub 溢出：正数 - 正数（不产生溢出）

正数 - 负数，结果为负数（上溢）

负数 - 负数（不产生溢出）

负数 - 正数，结果为正数（下溢）

（三）逻辑运算

~：按位取反

&(and)：按位与操作

|(or)：按位或操作

^(xor)：按位异或操作

~(|)(nor)：按位或非操作

指令对标志位影响：指令执行后，CF 和 OF 置 0，ZF 根据结果是否为 0 设置

（四）比较运算

有符号数比较 slt

指令对标志位影响：in0 小于 in1 置 CF 为 1

in0 为正数，in1 为负数，out 为 0

in0 为负数，in1 为正数，out 为 1

in0 和 in1 为负数，alu 进行补码运算，1111（-1）、1110（-2），直接进行数值比较的结果与其代表的有符号数比较结果相同

in0 和 in1 为正数，直接进行比较

通过分析，后两种情况可以合并

无符号数比较 sltu

可直接进行比较

指令对标志位影响：in0 小于 in1 置 OF 为 1

（五）移位运算

逻辑左移 sll，sllv

将数据向左移动，最低位用 0 补充

逻辑右移 srl，srlv

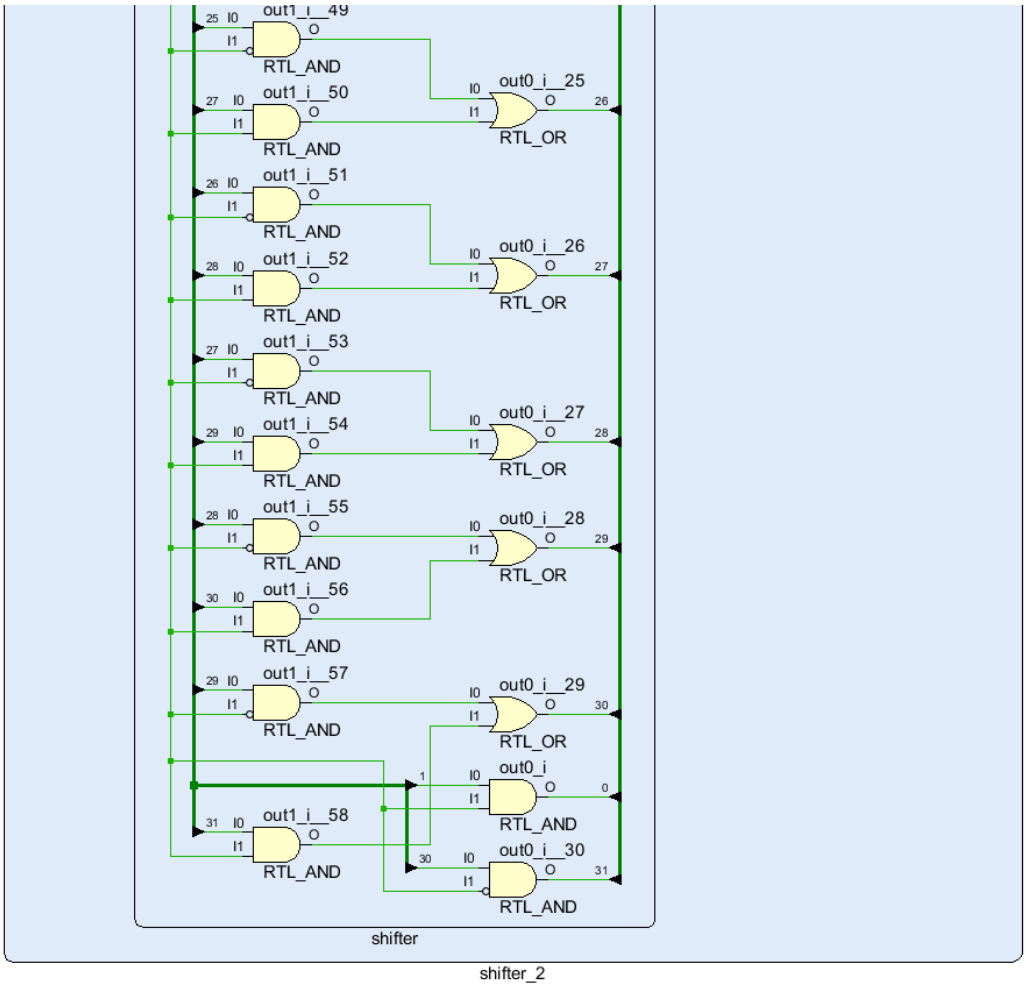
将数据向右移动，最高位用 0 补充

算术右移 sra，srav

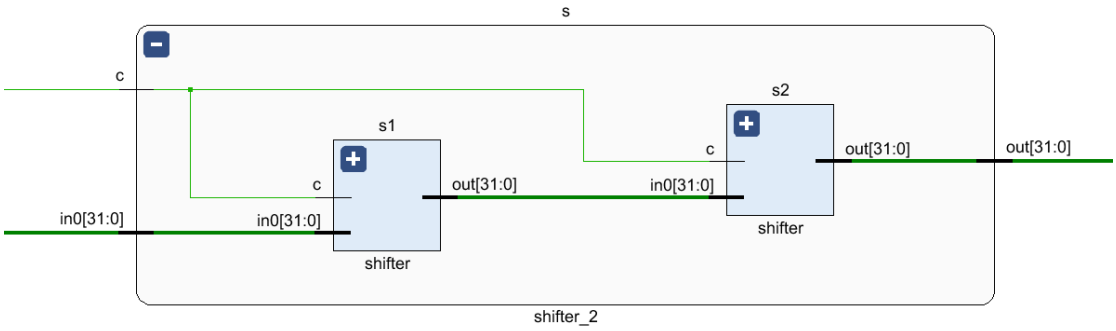
将各位依次右移指定位数，然后在左侧用原符号位补齐

指令对标志的影响：将最后移出的移位写入 CF

移位器（1 位，部分）：

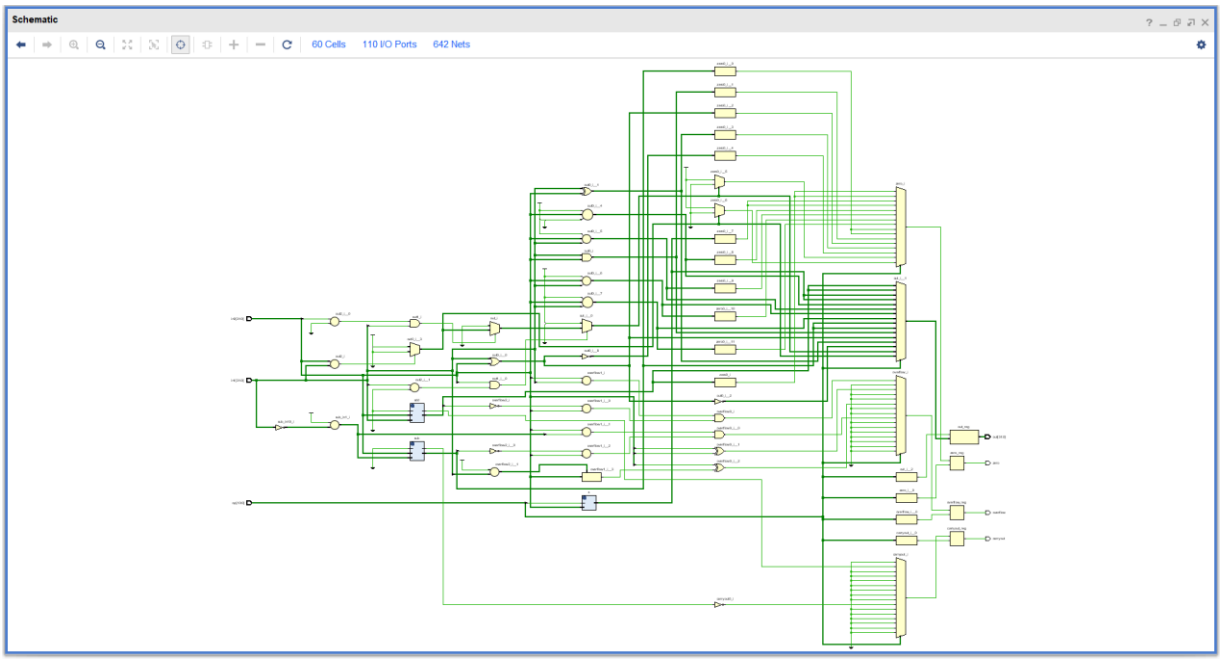


移位器（2 位）：



运行结果以及延迟

电路设计：

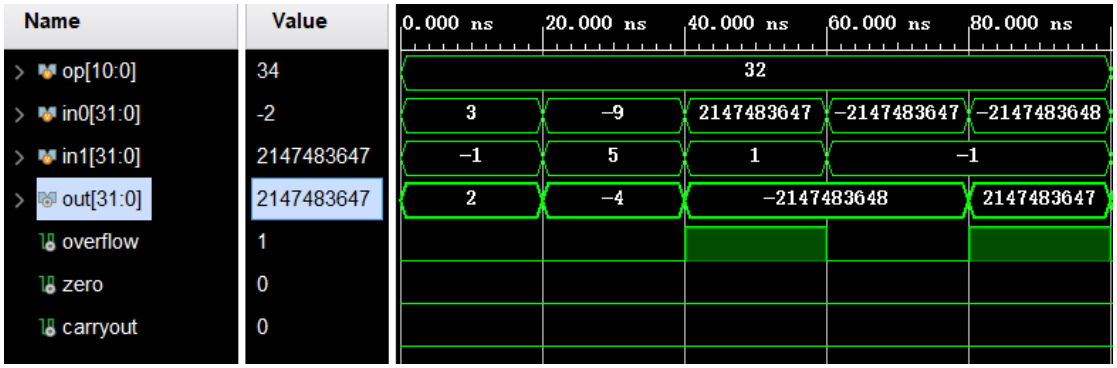


线路延迟：

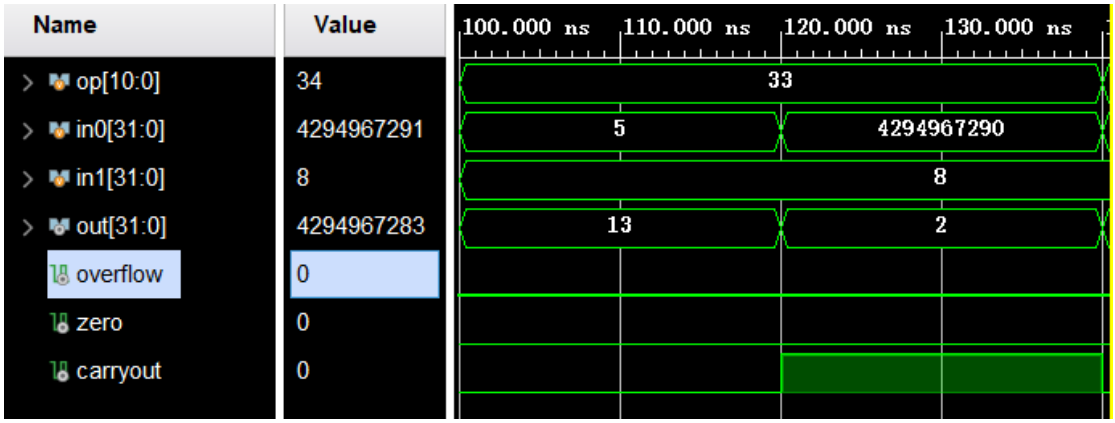
Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock
Path 1	∞	13	14	114	in1[1]	zero_reg/D	6.426	1.919	4.508	∞	input port clock
Path 2	∞	13	14	114	in1[1]	out_reg[29]/D	5.933	1.922	4.012	∞	input port clock
Path 3	∞	13	14	114	in1[1]	out_reg[25]/D	5.930	1.919	4.012	∞	input port clock
Path 4	∞	13	14	114	in1[1]	out_reg[27]/D	5.930	1.919	4.012	∞	input port clock
Path 5	∞	13	14	114	in1[1]	out_reg[26]/D	5.922	1.922	4.001	∞	input port clock
Path 6	∞	13	14	114	in1[1]	out_reg[31]/D	5.906	2.108	3.799	∞	input port clock
Path 7	∞	13	14	114	in1[1]	out_reg[28]/D	5.754	2.108	3.647	∞	input port clock
Path 8	∞	13	14	114	in1[1]	out_reg[30]/D	5.754	2.108	3.647	∞	input port clock
Path 9	∞	12	13	114	in1[1]	overflow_reg/D	5.665	1.866	3.800	∞	input port clock
Path 10	∞	12	13	114	in1[1]	out_reg[23]/D	5.587	2.055	3.533	∞	input port clock

运行结果：

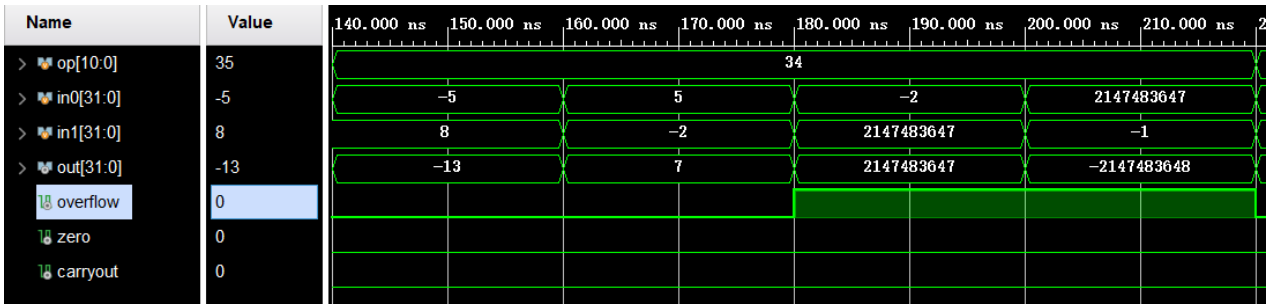
add



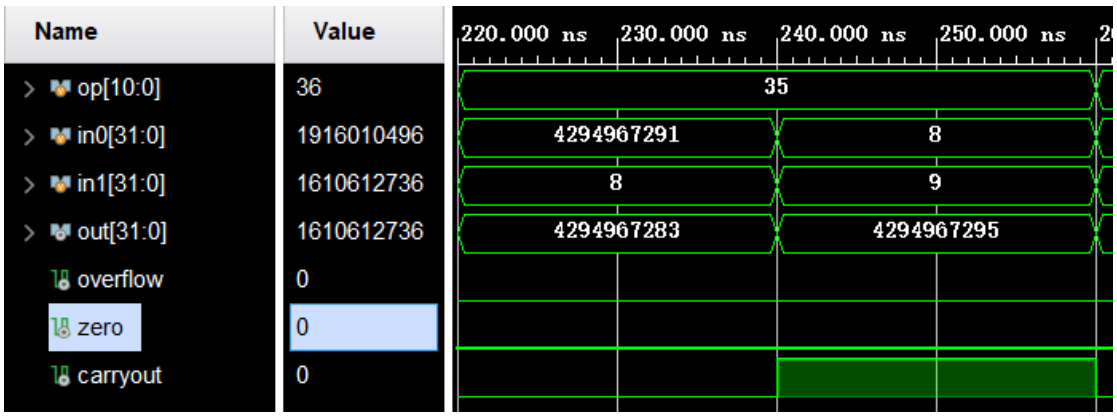
addu



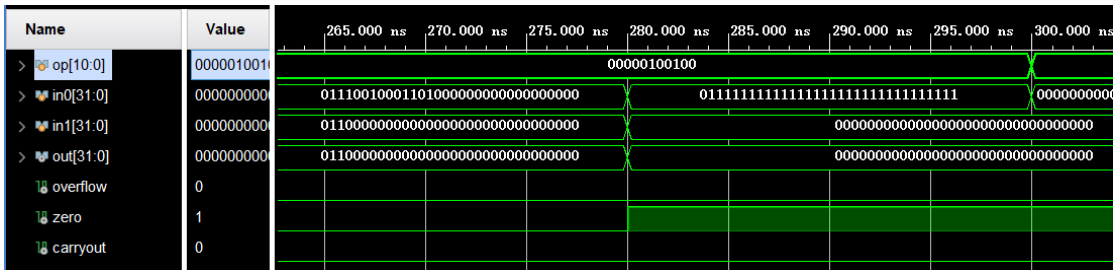
sub


Name
Value
140.000 ns
150.000 ns
160.000 ns
170.000 ns
180.000 ns
190.000 ns
200.000 ns
210.000 ns
op[10:0]
35
34
in0[31:0]
-5
-5
5
-2
2147483647
in1[31:0]
8
8
-2
2147483647
-1
out[31:0]
-13
-13
7
2147483647
-2147483648
overflow
0
zero
0
carryout
0

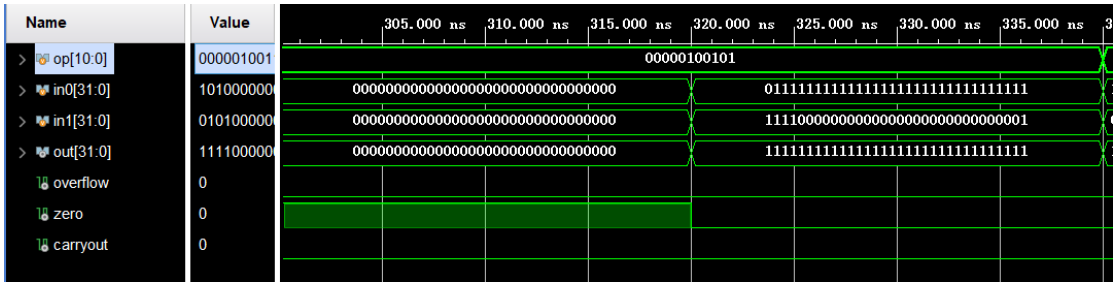
subu


Name
Value
220.000 ns
230.000 ns
240.000 ns
250.000 ns
op[10:0]
36
35
in0[31:0]
1916010496
4294967291
8
in1[31:0]
1610612736
8
9
out[31:0]
1610612736
4294967283
4294967295
overflow
0
zero
0
carryout
0

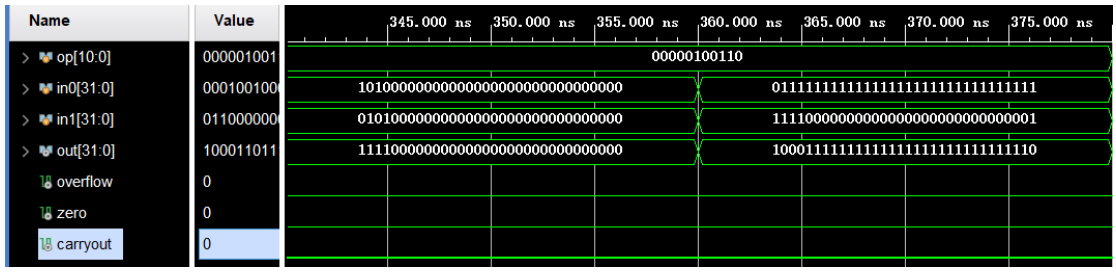
and


Name
Value
265.000 ns
270.000 ns
275.000 ns
280.000 ns
285.000 ns
290.000 ns
295.000 ns
300.000 ns
op[10:0]
00000100
in0[31:0]
00000000
01110010001101000000000000000000
01111111111111111111111111111111
00000000
in1[31:0]
00000000
01100000000000000000000000000000
00000000000000000000000000000000
out[31:0]
00000000
01100000000000000000000000000000
00000000000000000000000000000000
overflow
0
zero
1
carryout
0

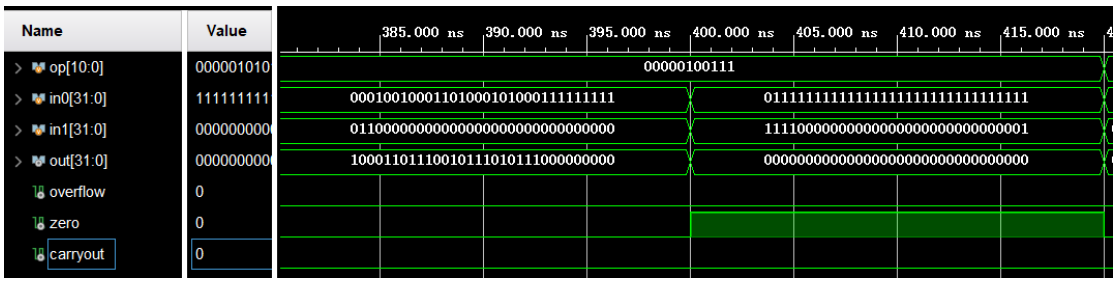
or



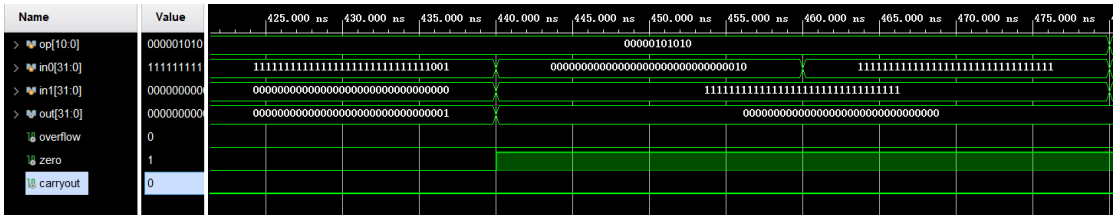
xor



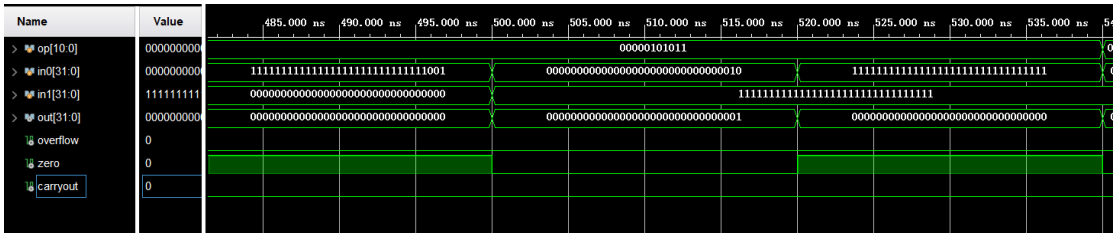
nor



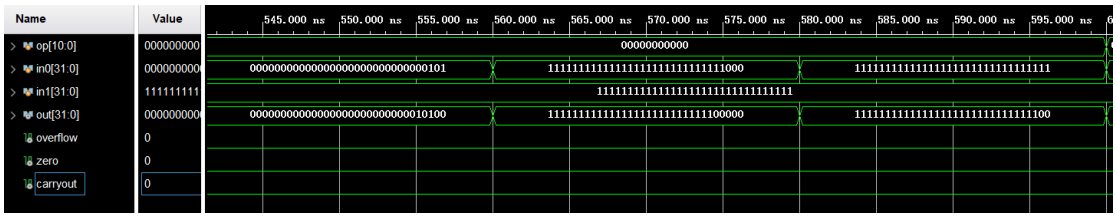
slt



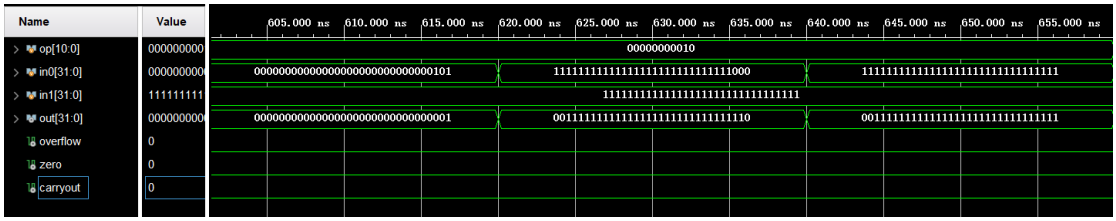
sltu



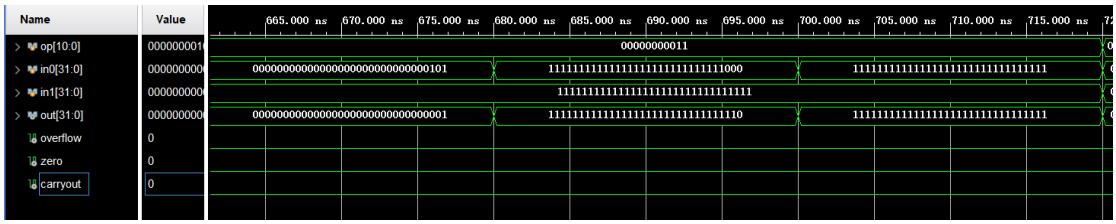
sll



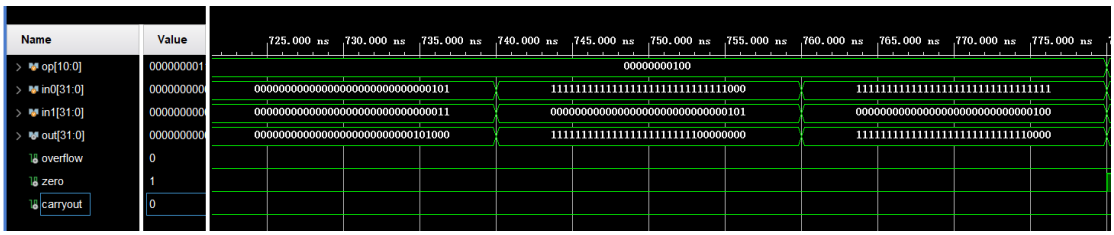
srl



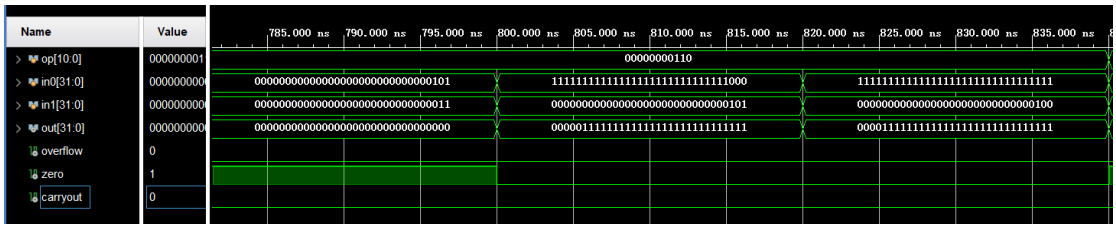
sra



sllv



srlv



srav