
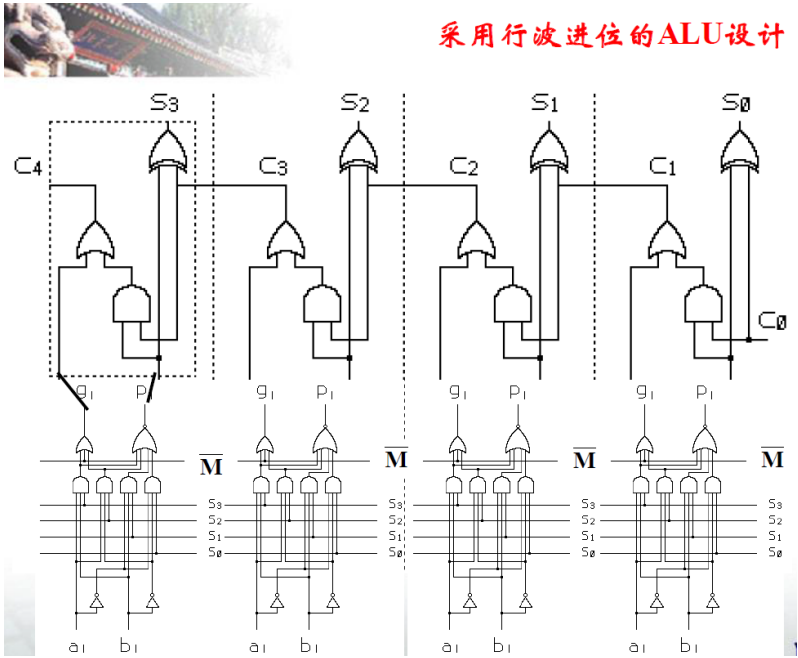


ALU设计作业

作业内容

- 
- 参考以下4bit ALU电路图和真值表，对32位ALU组合逻辑电路进行设计，并进行仿真验证。
将程序代码和有关文档打包提交。



$$S = S_3 \cdot a \cdot b + S_2 \cdot a \cdot \bar{b} + S_1 \cdot \bar{a} \cdot b + S_0 \cdot \bar{a} \cdot \bar{b}$$



选择控制信号						逻辑 A=a[i]; B=b[i];	功能
S3	S2	S1	S0	Ci	M		
0	0	0	0	1	0	0	置全0
0	0	0	1	1	0	!A & !B	nor
0	0	1	0	1	0	!A & B	notand
0	0	1	1	1	0	!A	not A
0	1	0	0	1	0	A & !B	andnot
0	1	0	1	1	0	!B	not B
0	1	1	0	1	0	A&!B !A&B	xor
0	1	1	1	1	0	!A !B	nand
1	0	0	0	1	0	A & B	and
1	0	0	1	1	0	A&B !A & !B	xnor
1	0	1	0	1	0	B	传送B
1	0	1	1	1	0	A&B !A&B !A&!B	notor
1	1	0	0	1	0	A	传送A
1	1	0	1	1	0	A&B A&!B !A&!B	or not
1	1	1	0	1	0	A&B A&!B !A&B	or
1	1	1	1	1	0	1	置全1
1	0	0	1	0	1	A ^ B ^ C	加法
0	1	1	0	1	1	(A ~^ B) ^ C	减法

文件夹内容说明

- alu
 - Testbench
 - tb_nbit_alu_top.sv
 - tb_alu_top.v
 - Vsrc
 - alu_core.v
 - alu_core_unit.v
 - Makefile
 - srclist
 - tb_alu_core.fsdb

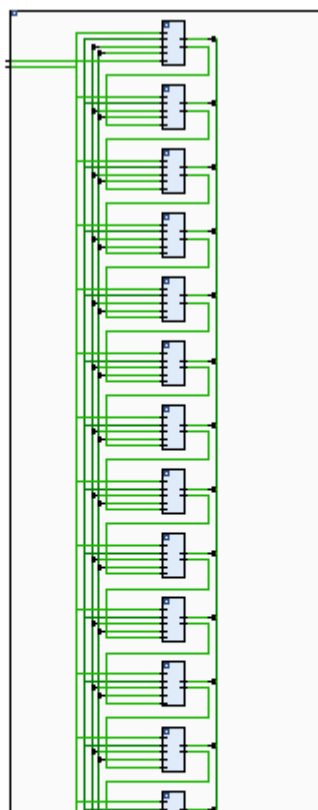
`Makefile` 包含 `make versim` 命令，该命令启动 `vcs` 对 `srclist` 的所有 `*.v` 文件进行综合和 Tb 仿真，用 `verdi` 打开 Tb 生成的 `*.fsdb` 波形。

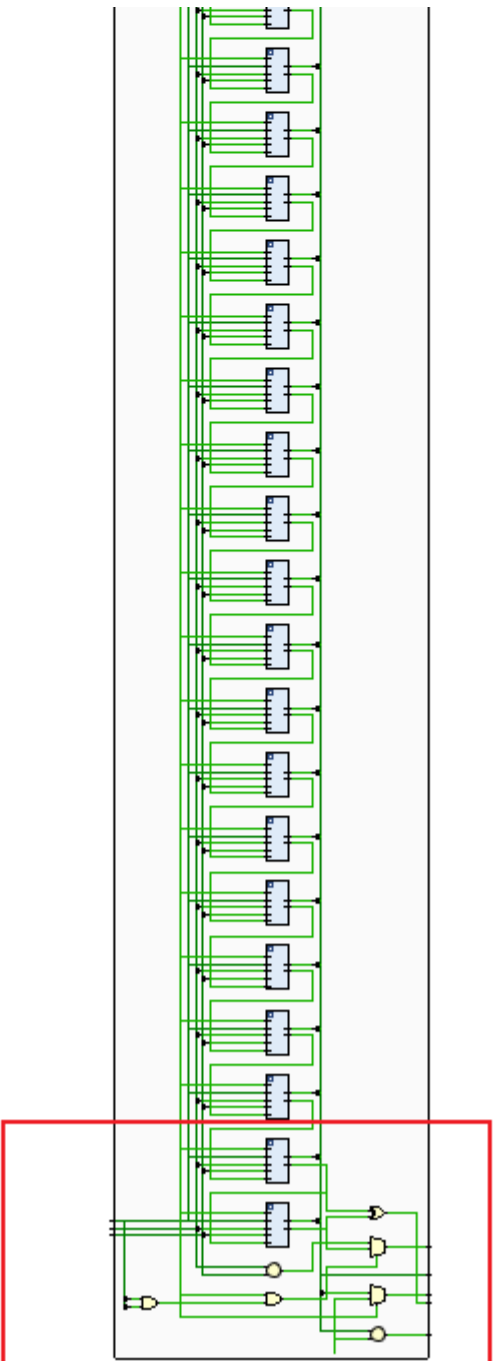
具有两个 Testbench，分别为 N bit ALU 的全覆盖率验证 Tb（N 需要数值比较小，例如 4），和 32 位 ALU 的非全覆盖率验证 Tb。如果用 `make versim`，则切换不同的 Tb 时需要修改 `srclist` 中的源文件列表。

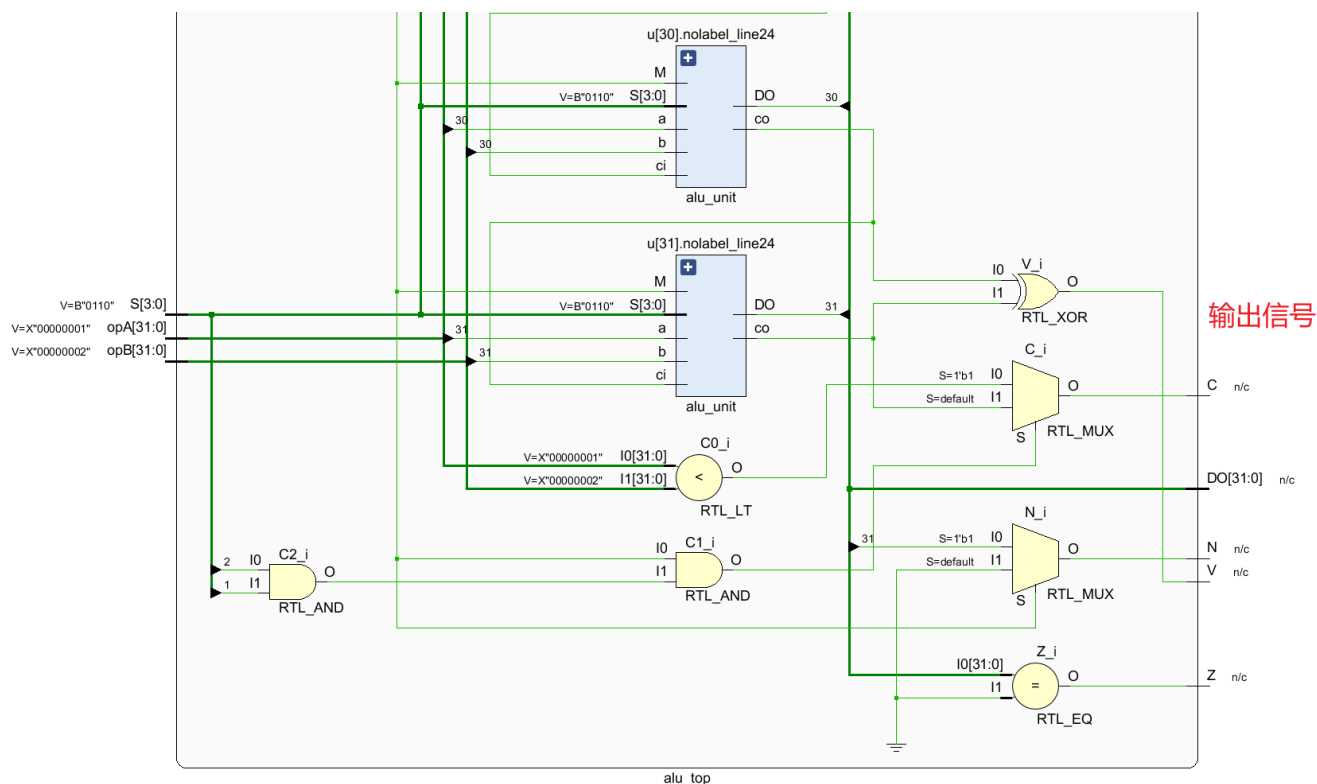
32位ALU电路设计

电路架构图

本次作业提交的ALU模型如下：







输出信号

设计注意点

进位信号C产生电路的设计

因为加减法的进位输出信号C的含义不同，减法的C表达的应该是借位的含义，因此在C输出产生之前增加 MUX，以通过判断是否是减法，而选择不同的两条数据通路，产生正确的输出进位信号。相关代码实现：

```
1 assign C = (M==1)&&(S[2]&S[1])? (opA <opB): ci [n] ; //the C of add and subtrac
```

溢出信号V产生电路的设计

本小节在课程所介绍的加法判断溢出的三种方法基础上，对有符号数加、减法的三种溢出判断方法进行理论分析，并给出了示例代码。

设两个有符号操作数分别为signed A[31: 0]、B[31: 0]，结果为signed S[31: 0]，符号位分别是A[31]、B[31]、S[31]。

方法1：

利用输入信号和输出信号的符号位是否产生变化来判断。

- 加法

加法正溢：{A[31], B[31], S[31]} = 3' b001

加法负溢：{A[31], B[31], S[31]} = 3' b110

```
1 V = (A[31]~^B[31])&&(A[31]!==S[31])
```

• 减法

减法正溢：{A[31], B[31], S[31]} = 3' b011

减法负溢：{A[31], B[31], S[31]} = 3' b100

```
1 V = (A[31]^B[31])&&(A[31]!==S[31])
```

方法2：

第 30 位（从0开始计数）向第 31 位的进位C[31]，和第 31 位向第 32 位的进位C[32]，C[31]与C[32]不同。

加法和减法同时适用该方法。

```
1 V = ci[31] ^ ci[32]
```

方法3：

将 ALU 扩展为 33 位计算，最高两位符号位不同。

加法和减法同时适用该方法。

对算术运算更深入的理解

经过课下与同学关于电路做加减法的输入数据范围以及输出信号有效性的讨论以及自己思考，先将自己的这方面理解总结如下。

电路支持输入为无符号数的加法，输入为有符号数的加法和减法。由于操作数opB在做减法时首先需要转变为补码表示，要求opB的最高位必须是符号位，因此不存在无符号数减法，或者说无符号数减法本质上和有符号数减法应归为一类。

无符号数加法的进位信号有效，溢出信号无效。有符号数加法的进位信号无效，溢出信号有效。有符号数减法的进位信号和溢出信号同时有效。

它们各自输入数据范围以及输出信号有效性如下表所示：

算术操作类型	输入操作数数值范围（设ALU位宽为N位）	进位输出信号有效性	溢出输出信号有效性
无符号数加法	$0 \sim 2^N - 1$	有效	无效

有符号数加法	$-2^N \sim 2^N - 1$	无效	有效
有符号数减法	$-2^N \sim 2^N - 1$	有效	有效

32位ALU仿真验证

Testbench测试思路

- 因为ALU的位宽较大，达到了32位，普通计算机无法完成所有测试用例的完整覆盖率的测试。
- 因此本次作业采取了如下方法进行了测试：
1. 编写了每种情况正确输出的仿真计算公式，实现自动化测试，可以自动检查电路输出信号的错误。
 2. 对于加法，因为没有输入信号指定输入是有符号还是无符号，因此测试默认为有符号数。
 3. 32位ALU的测试：每种逻辑运算情况用1个测试用例进行测试，对算术运算情况每种用2个测试用例进行测试。
 4. N位ALU的完整覆盖率自动化测试：编写了完整覆盖率测试Tb，ALU的位宽N需要为一个较小的数字，例如4位。已实现TODO：

~~TODO: 后续可以进一步对电路进行结构化、更多测试用例的测试，让测试达到更高覆盖率。~~

自动化测试代码

对于每种情况，列写出正确(output_golden) 输出结果的公式，通过output和output_golden的对比，实现自动化检错功能。

```
1 always @(*) begin
2     case ({S,Cin,M})
3         6'b0000_1_0: begin//all bits are 0
4             DO_golden = 'b0;
5             {C_golden,V_golden,N_golden,Z_golden} = {1'b1,1'b0,1'b0,1'b1};
6         end
7         6'b0001_1_0: begin//nor
8             DO_golden = ~opA & ~opB;
9             {C_golden,V_golden,N_golden,Z_golden} = {1'b1,1'b0,1'b0,!(|DO_golden
10        end
11 .....
12         6'b1110_1_0: begin//or
13             DO_golden = opA | opB;
14             {C_golden,V_golden,N_golden,Z_golden} = {1'b1,1'b0,1'b0,!(|DO_golden
15        end
16         6'b1111_1_0: begin//all bits are 1
17             DO_golden = 32'hffffffff;
18             {C_golden,V_golden,N_golden,Z_golden} = {1'b1,1'b0,1'b0,1'b0};
19        end
```

```

20      6'b1001_0_1: begin//add
21          {C_golden,D0_golden} = opA + opB;
22          {V_golden,N_golden,Z_golden} = {(opA[31]^opB[31])&&(opA[31]!==D0_go
23      end
24      6'b0110_1_1: begin//sub
25          D0_golden = opA - opB;
26          C_golden = opA < opB;
27          {V_golden,N_golden,Z_golden} = {(opA[31]^opB[31])&&(opA[31]!==D0_go
28      end
29      default: $display($time,"*** There is one illegal input! -- INPUT: {S,Ci
30  endcase
31 end
32 initial begin
33     #2
34     for (integer i=0; i<19; i=i+1) begin
35         #10
36         if ({C_golden,V_golden,N_golden,Z_golden}!={C,V,N,Z}) begin
37             error_count = error_count + 1;
38             if (error_count <= 10) begin
39                 //$display("***ERROR at time = %0d ***", $time);
40                 $display("ERROR at time = %0d -- INPUT:S = %b, Cin = %b, M = %b;
41             end
42             if (error_count == 10) begin
43                 $display("\n\nError count reached 10, subsequent error messages
44             end
45         end
46     end
47     if (error_count == 0)
48         $display("*** Testbench Successfully Completed! There is no error!***");
49     else begin
50         $display("\n*****");
51         $display("*** Testbench completed with %0d errors ***",error_count);
52         $display("*****\n\n");
53     end
54     $finish;
55 end

```

仿真结果

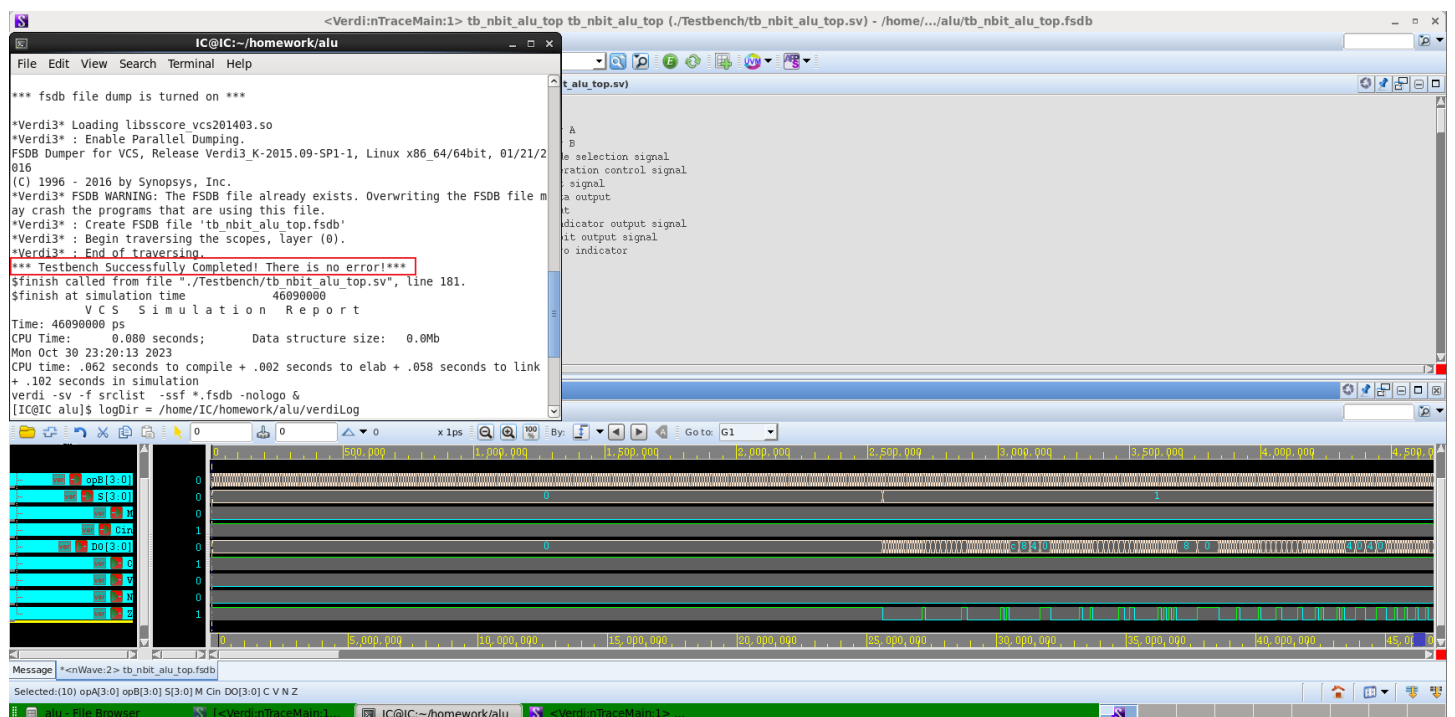
自动检错结果

32位ALU的测试

```
IC@IC:~/homework/alu
File Edit View Search Terminal Help
*Verdi3* : Create FSDb file 'tb_alu_top.fsd'
*Verdi3* : Begin traversing the scopes, layer (0).
*Verdi3* : End of traversing.
0 -- INPUT: {S,Cin,M}=xxxxxx;opA=xxxxxxx,opB=xxxxxxx,
OUTPUT: D0=xxxxxxx,C=x,V=x,N=x,Z=x
10 -- INPUT: {S,Cin,M}=000010;opA=0000ffff,opB=ff00ff00,
OUTPUT: D0=00000000,C=1,V=0,N=0,Z=1
20 -- INPUT: {S,Cin,M}=000110;opA=0000ffff,opB=ff00ff00,
OUTPUT: D0=00ff0000,C=1,V=0,N=0,Z=0
30 -- INPUT: {S,Cin,M}=001010;opA=0000ffff,opB=ff00ff00,
OUTPUT: D0=ff000000,C=1,V=0,N=0,Z=0
40 -- INPUT: {S,Cin,M}=001110;opA=0000ffff,opB=ff00ff00,
OUTPUT: D0=ffff0000,C=1,V=0,N=0,Z=0
50 -- INPUT: {S,Cin,M}=010010;opA=0000ffff,opB=ff00ff00,
OUTPUT: D0=000000ff,C=1,V=0,N=0,Z=0
60 -- INPUT: {S,Cin,M}=010110;opA=0000ffff,opB=ff00ff00,
OUTPUT: D0=00ff00ff,C=1,V=0,N=0,Z=0
70 -- INPUT: {S,Cin,M}=011010;opA=0000ffff,opB=ff00ff00,
OUTPUT: D0=ff0000ff,C=1,V=0,N=0,Z=0
80 -- INPUT: {S,Cin,M}=011110;opA=0000ffff,opB=ff00ff00,
OUTPUT: D0=ffff00ff,C=1,V=0,N=0,Z=0
90 -- INPUT: {S,Cin,M}=100010;opA=0000ffff,opB=ff00ff00,
OUTPUT: D0=0000ff00,C=1,V=0,N=0,Z=0
100 -- INPUT: {S,Cin,M}=100110;opA=0000ffff,opB=ff00ff00,
OUTPUT: D0=00ffff00,C=1,V=0,N=0,Z=0
110 -- INPUT: {S,Cin,M}=101010;opA=0000ffff,opB=ff00ff00,
OUTPUT: D0=ff00ff00,C=1,V=0,N=0,Z=0
120 -- INPUT: {S,Cin,M}=101110;opA=0000ffff,opB=ff00ff00,
OUTPUT: D0=ffffff00,C=1,V=0,N=0,Z=0
130 -- INPUT: {S,Cin,M}=110010;opA=0000ffff,opB=ff00ff00,
OUTPUT: D0=0000ffff,C=1,V=0,N=0,Z=0
140 -- INPUT: {S,Cin,M}=110110;opA=0000ffff,opB=ff00ff00,
OUTPUT: D0=00ffffff,C=1,V=0,N=0,Z=0
150 -- INPUT: {S,Cin,M}=111010;opA=0000ffff,opB=ff00ff00,
OUTPUT: D0=ff00ffff,C=1,V=0,N=0,Z=0
160 -- INPUT: {S,Cin,M}=111110;opA=0000ffff,opB=ff00ff00,
OUTPUT: D0=ffffffff,C=1,V=0,N=0,Z=0
170 -- INPUT: {S,Cin,M}=100101;opA=ffffff,opB=abcd4321,
OUTPUT: D0=abcd4320,C=1,V=0,N=1,Z=0
180 -- INPUT: {S,Cin,M}=011011;opA=ffffff,opB=0000ffff,
OUTPUT: D0=ffff0000,C=1,V=0,N=1,Z=0
190 -- INPUT: {S,Cin,M}=011011;opA=00000001,opB=00000002,
OUTPUT: D0=ffffff,C=1,V=0,N=1,Z=0
*** Testbench Successfully Completed! There is no error!***
```

Tb成功通过所有的20个测试用例。

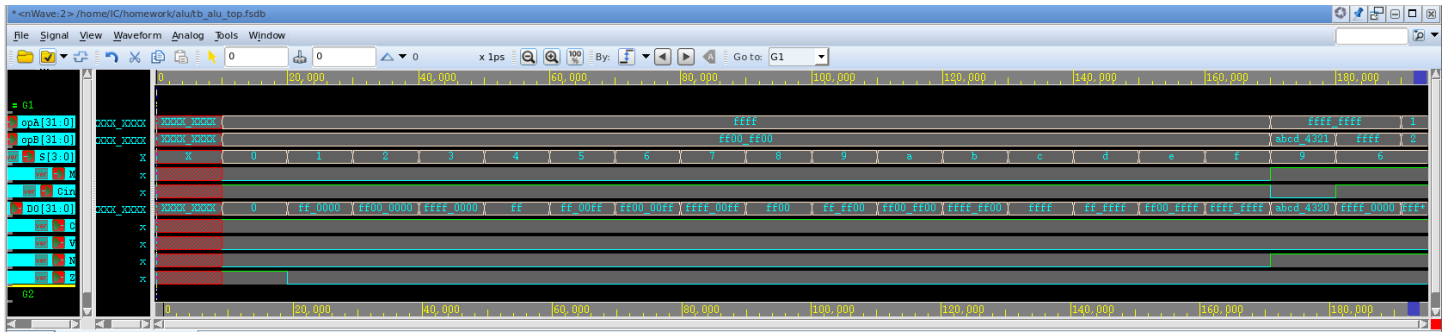
N位ALU的完整覆盖率自动化测试



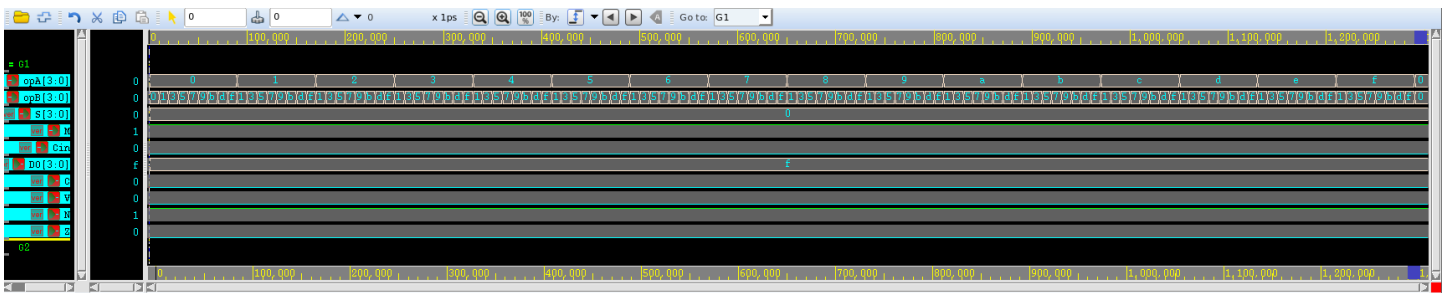
在N=4时，电路通过所有测试用例的测试。

波形截图

32位ALU的测试



N位ALU的完整覆盖率自动化测试



其它

本次作业过程中遇到的写代码的注意点

在tb文件中，输入opA和opB设置为了signed。

```
≡ alu_top.v    ≡ tb_nbit_alu_top.sv X    ≡ tb_alu_top.v    ≡ srclist    ≡ alu_unit.v

Testbench > ≡ tb_nbit_alu_top.sv
1  `timescale 1ns/1ps
2  module tb_nbit_alu_top;
3  parameter    ALU_WIDTH = 4                ;
4  reg signed   [ALU_WIDTH-1:0] opA          ; //operator A
5  reg signed   [ALU_WIDTH-1:0] opB          ; //operator B
```

在做加法的检错时，tb的进位C golden数据产生不正确，遇到如下大量报错：

```
IC@IC:~/homework/alu
File Edit View Search Terminal Help

ERROR at time = 41042 -- INPUT:S = 1001, Cin = 0, M = 1;opA=0,opB=8, OUTPUT:
DO=8,C=0,V=0,N=1,Z=0, Correct OUTPUT: DO=8,C=1 V=0,N=1,Z=0
ERROR at time = 41052 -- INPUT:S = 1001, Cin = 0, M = 1;opA=0,opB=9, OUTPUT:
DO=9,C=0,V=0,N=1,Z=0, Correct OUTPUT: DO=9,C=1 V=0,N=1,Z=0
ERROR at time = 41062 -- INPUT:S = 1001, Cin = 0, M = 1;opA=0,opB=a, OUTPUT:
DO=a,C=0,V=0,N=1,Z=0, Correct OUTPUT: DO=a,C=1 V=0,N=1,Z=0
ERROR at time = 41072 -- INPUT:S = 1001, Cin = 0, M = 1;opA=0,opB=b, OUTPUT:
DO=b,C=0,V=0,N=1,Z=0, Correct OUTPUT: DO=b,C=1 V=0,N=1,Z=0
ERROR at time = 41082 -- INPUT:S = 1001, Cin = 0, M = 1;opA=0,opB=c, OUTPUT:
DO=c,C=0,V=0,N=1,Z=0, Correct OUTPUT: DO=c,C=1 V=0,N=1,Z=0
ERROR at time = 41092 -- INPUT:S = 1001, Cin = 0, M = 1;opA=0,opB=d, OUTPUT:
DO=d,C=0,V=0,N=1,Z=0, Correct OUTPUT: DO=d,C=1 V=0,N=1,Z=0
ERROR at time = 41102 -- INPUT:S = 1001, Cin = 0, M = 1;opA=0,opB=e, OUTPUT:
DO=e,C=0,V=0,N=1,Z=0, Correct OUTPUT: DO=e,C=1 V=0,N=1,Z=0
ERROR at time = 41112 -- INPUT:S = 1001, Cin = 0, M = 1;opA=0,opB=f, OUTPUT:
DO=f,C=0,V=0,N=1,Z=0, Correct OUTPUT: DO=f,C=1 V=0,N=1,Z=0

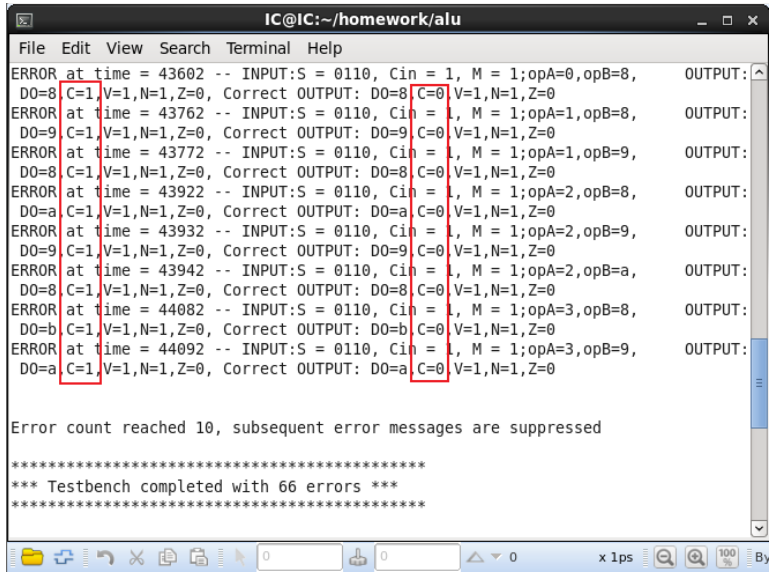
Error count reached 10, subsequent error messages are suppressed

*****
*** Testbench completed with 194 errors ***
*****
```

检查之后发现是，opA+opB时由于输入是有符号数，因此会进行符号扩展，导致C_golden不正确。
因此加入强制类型转换，修改为以下代码。

```
141 6'b1001_0_1: begin//add
142 {C_golden,DO_golden} = $unsigned(opA) + $unsigned(opB);
143 {V_golden,N_golden,Z_golden} = {(opA[ALU_WIDTH-1]^opB[ALU_WIDTH-1])&&(opA[ALU_WIDTH-1]!=DO_golden)
144 end
```

但是报如下错。



最后取消输入信号的signed，代码改为如下，仿真成功

```
141 6'b1001_0_1: begin//add
142 {C_golden,DO_golden} = opA + opB;
143 {V_golden,N_golden,Z_golden} = {(opA[ALU_WIDTH-1]^opB[ALU_WIDTH-1])&&(opA[ALU_WIDTH-1]!=DO_golden)
144 end
145 6'b110_1_1: begin//sub
146 DO_golden = $signed(opA) - $signed(opB);
147 C_golden = DO_golden[ALU_WIDTH-1];
148 {V_golden,N_golden,Z_golden} = {(opA[ALU_WIDTH-1]^opB[ALU_WIDTH-1])&&(opA[ALU_WIDTH-1]!=DO_golden)
149 end
150 default: $display($time,"*** There is one illegal input! -- INPUT: {S,Cin,M}=%b%b%b;opA=%h,opB=%h,\t
151 endcase
```

一个未完全解决的问题

在写Tb时，经常需要在for循环中设置时延。但是如果写成下面的情况，即end之前紧跟了一个#<时间>，那么会报无法识别到end的错。

```

initial begin
    for (integer i=0;i<18;i=i+1) begin
        {S,Cin,M} = select_signal[i*6 +: 6];
        for (test = 0; test <= LOOP_MAX_LOGICAL; test = test +1) begin
            opA = test[2*ALU_WIDTH-1:ALU_WIDTH];
            opB = test[ALU_WIDTH-1:0];
            #10;
        end
    end
end
end

```

目前我想到的解决办法是新建一个变量，如integer flag，然后将所有这类end之前的#<时间>语句改为如下语句：

```

65  initial begin
66      for (integer i=0;i<18;i=i+1) begin
67          {S,Cin,M} = select_signal[i*6 +: 6];
68          for (test = 0; test <= LOOP_MAX_LOGICAL; test = test +1) begin
69              opA = test[2*ALU_WIDTH-1:ALU_WIDTH];
70              opB = test[ALU_WIDTH-1:0];
71              #10 flag = 0;
72          end
73      end
74  end

```

通过这种方法规避掉了问题。