**1、实验名称：**实验二 组合逻辑实验

**2、实验目的：**

(1) 学习用verilog设计较复杂的组合逻辑电路

(2) 进一步熟悉vivado工具

**3、实验内容：**

1. **实验2.1——多路选择器**
   1. 新建工程，用3种描述方法分别实现1位2选1选择器，并撰写测试激励，对比3个选择器的输出是否一致；
   2. 用1位2选1搭建1位4选1选择器；并仿真测试；
   3. 用两种描述方法实现4位二选一选择器，并撰写测试激励，对比输出是否一致。
2. **实验2.2——7段数码管**

4个拨码开关控制1个数码管，选择板子上的4个拨码开关作为数据输入，选择板子上的1个数码管作为输出，将输入的4位二进制转换为16进制数输出显示到数码管。

1. **实验2.3——加法器实现及性能比较**

实现32位逐位进位加法器、32位选择进位加法器，并进行性能比较。

* 1. 学习课件中逐位进位加法器、选择进位加法器的原理；
  2. 新建工程，完成两种加法器的Verilog描述，其中**一位全加器**已经提供（**module add1**在add32\_tb.v文件中已定义）直接使用即可，两种加法器请都使用add1作为基本模块开始搭建；模块名称及端口定义如下：

module csadd32 (a,b,cin,s,cout); //选择进位加法器

module rcadd32 (a,b,cin,s,cout); //逐位进位加法器

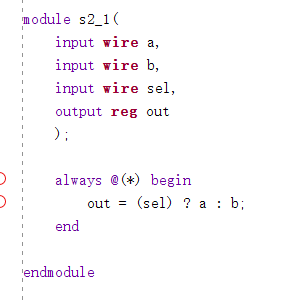
* 1. 编写测试激励：可更改add32\_tb.v文件中激励产生代码，按照自己的思路产生测试数据，通过仿真验证加法器功能，并观察两种加法器的性能情况。

**实验步骤：**

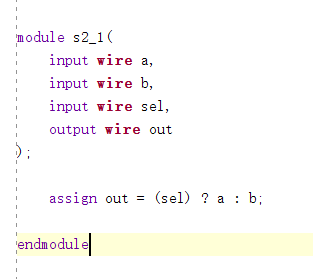
4、**实验2.1（多路选择器）的实现及仿真验证**

（1）3种描述方法分别实现1位2选1选择器Verilog代码

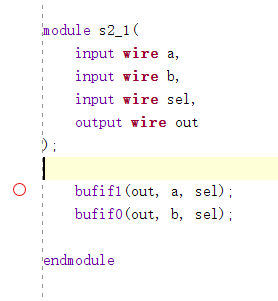
1.行为描述方式



2.数据流描述方式

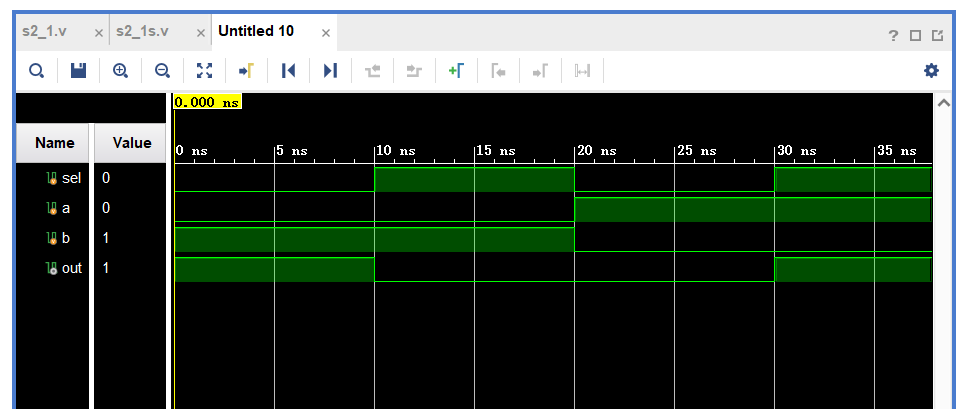


3.结构化描述方式



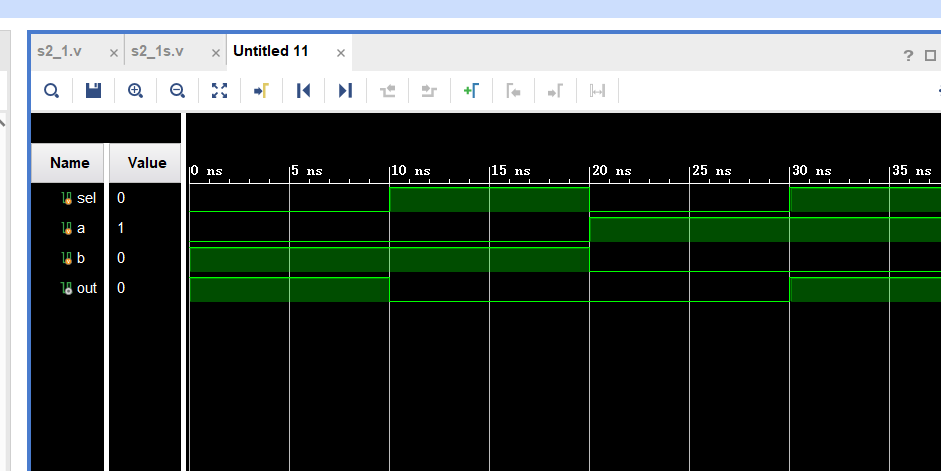
（2）（1）的仿真波形及对比说明

1.行为描述方式



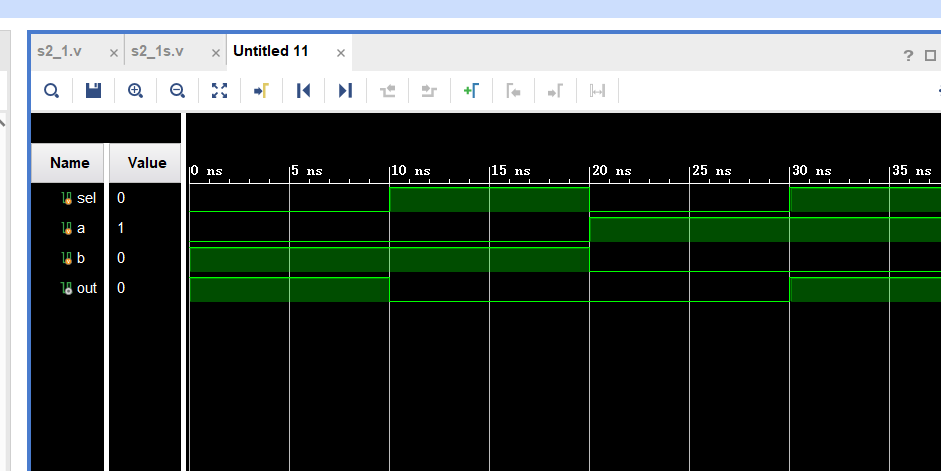
当sel=0时选择b，即输出out=1

2.数据流描述方式



当sel=0时选择b，即输出out=1

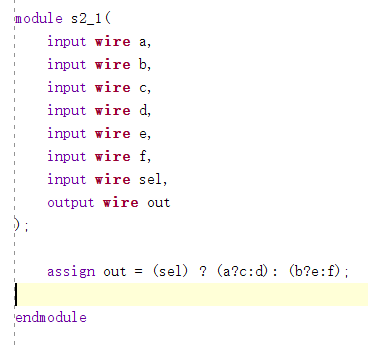
3.结构化描述方式



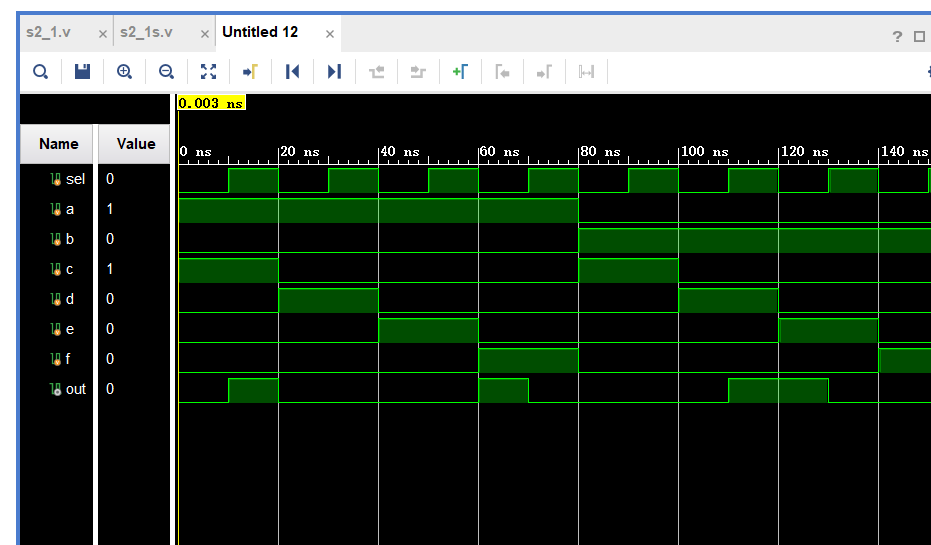
当sel=0时选择b，即输出out=1

分析上述三种图形可得，在同种输入测试下，0-35ns得出的波形图完全一致。

（3）用1位2选1搭建1位4选1选择器的Verilog代码



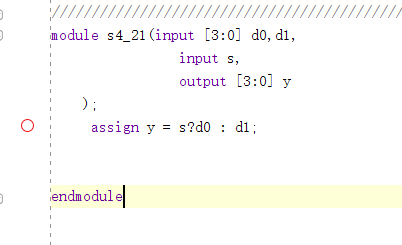
（4）（3）的仿真波形及说明



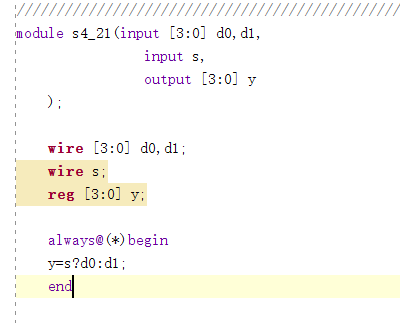
当sel=0时选择b?e:f，此时b=0，即选择f，输出out=f=0

（5）用两种描述方法实现4位二选一选择器的Verilog代码

1.数据流描述方式

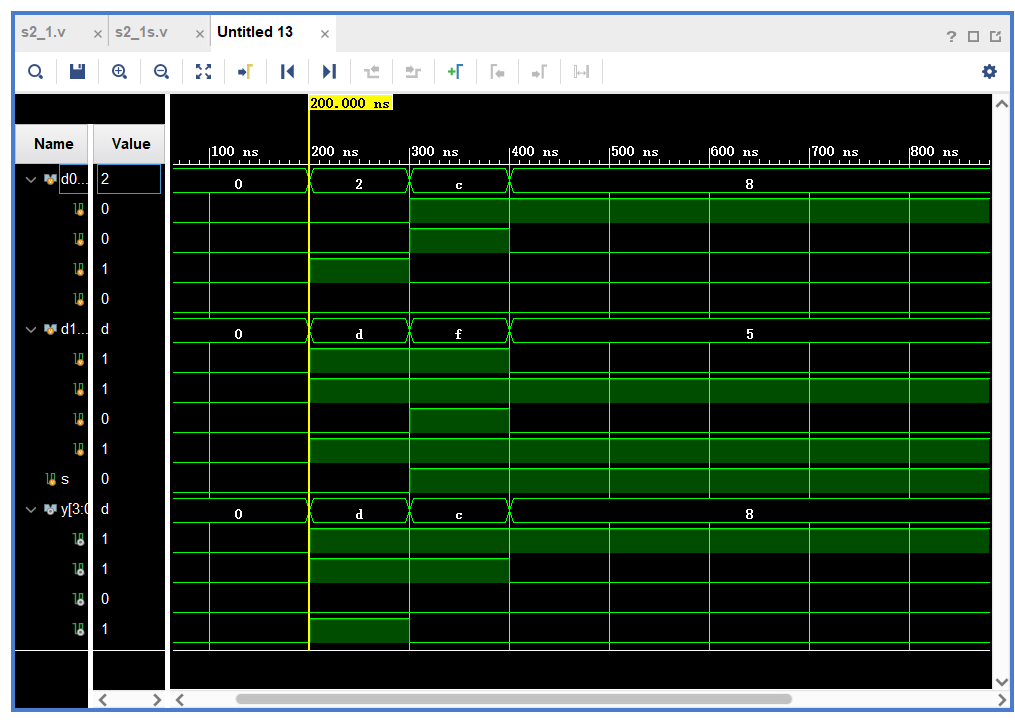


2.行为描述方式



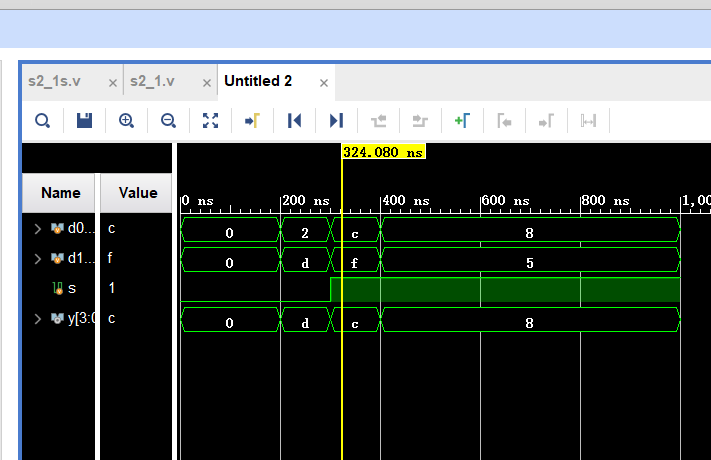
（6）（5）的仿真波形及对比说明

1.数据流描述方式



当s=0，输出选择d1，即当d1=d时输出out=d，当s=1，输出选择d0，即当d0=c时输出out=c

2.行为描述方式



当s=0，输出选择d1，即当d1=d时输出out=d，当s=1，输出选择d0，即当d0=c时输出out=c

两个波形图在同等激励下完全相同，说明二者虽然采用不同的代码风格，但达到了相同的效果。

5、**实验2.2（7段数码管）的实现及验证**

（1）系统输入输出信号定义

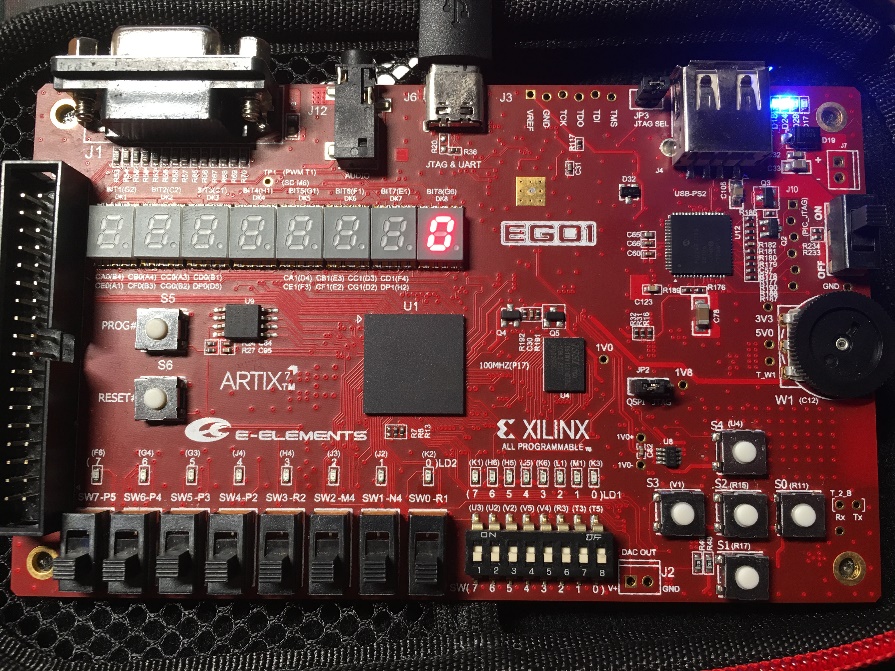
输入信号key1,key2,key3,key4

分别分配到开关SW0（引脚R1）,SW1（引脚M4）,SW2（引脚N4）,SW3（引脚R2）

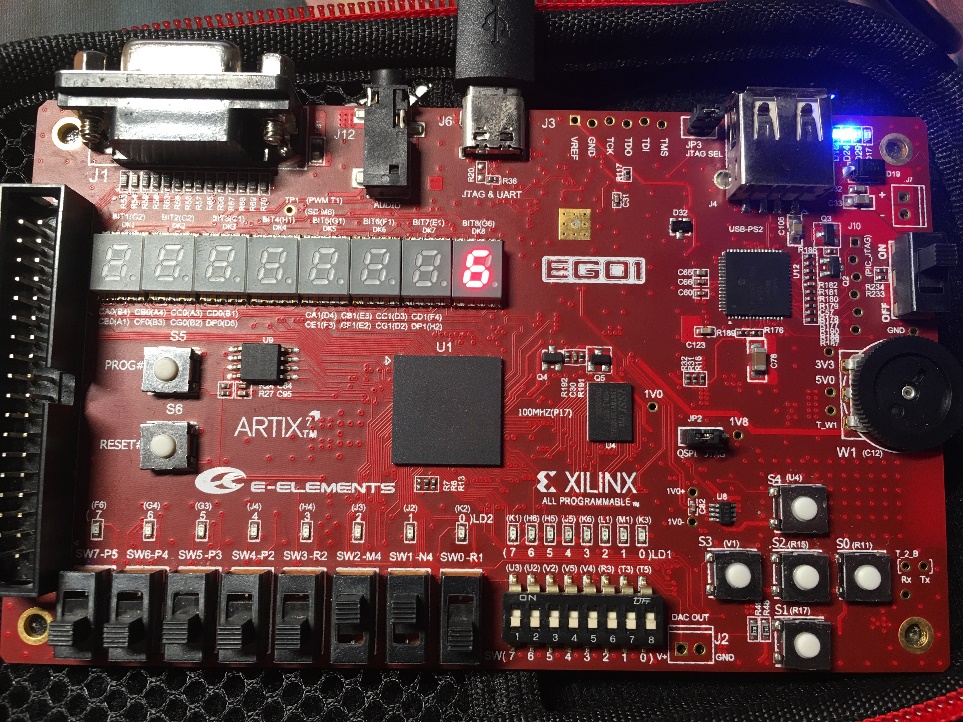
输出信号seg[7:0]

分别分配到数码管七根发光二极管LED1\_CA（引脚D4）, LED1\_CB（引脚E3）, LED1\_CC（引脚D3）, LED1\_CD（引脚F4）, LED1\_CE（引脚F3）, LED1\_CF（引脚E2）, LED1\_CG（引脚D2）,

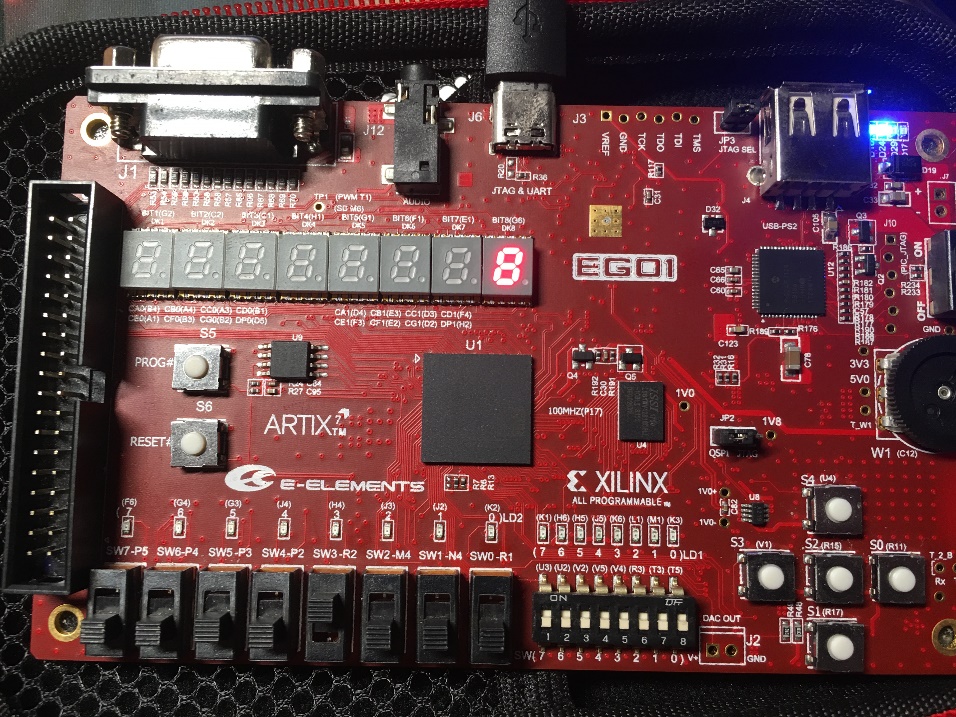
（3）板级测试验证



测试图1:未拨下任何开关，即输入0000，输出显示0



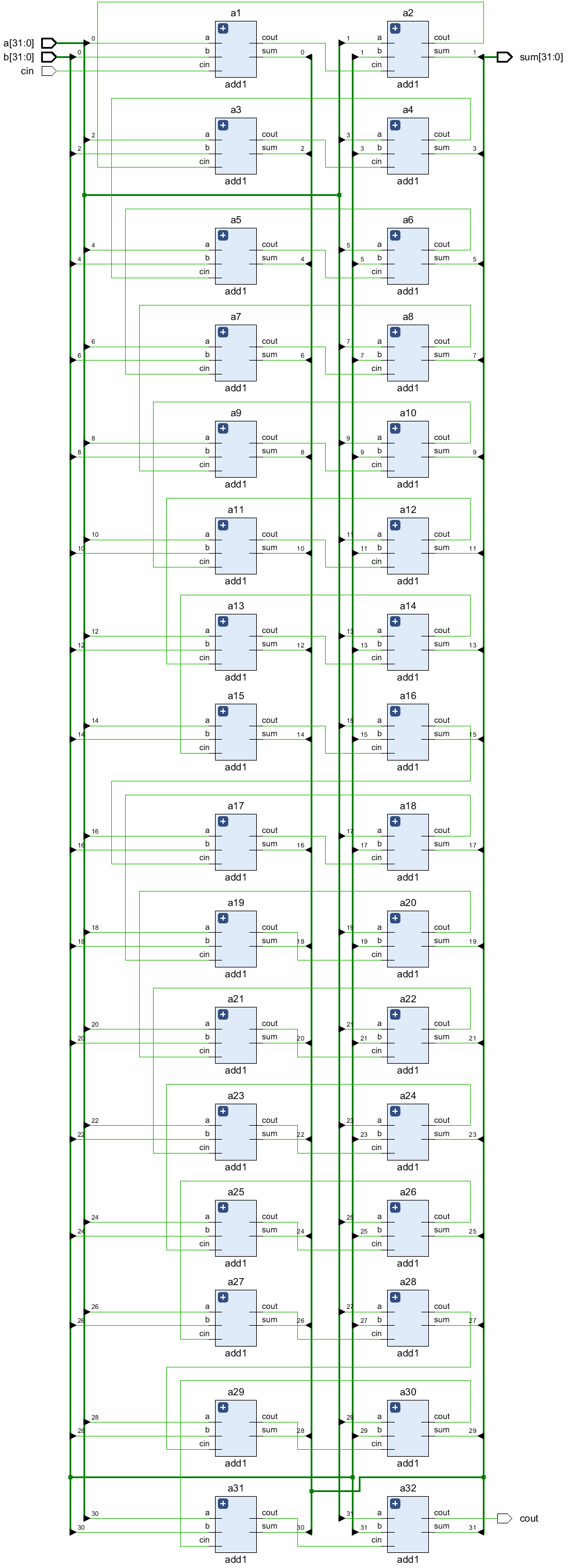
测试图2:拨下开关SW2及SW1，即输入0110，输出显示6



测试图3:拨下开关SW4，即输入1000，输出显示8

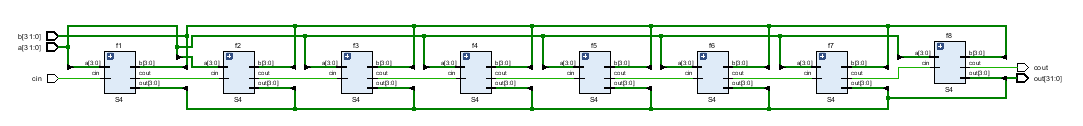
6、**实验2.3（加法器及性能比较）的实现及仿真验证**

（1）32位逐位进位加法器模块RTL分析结构图及说明



逐位进位加法器由32个一位全加器构成，每一个一位全加器接分别接入两个一位输入和一个低位进位，当低位进位传出进位信息后进行本位计算，再将本位进位传入高位，以此类推进行计算。

（2）32位选择进位加法器模块RTL分析结构图及说明



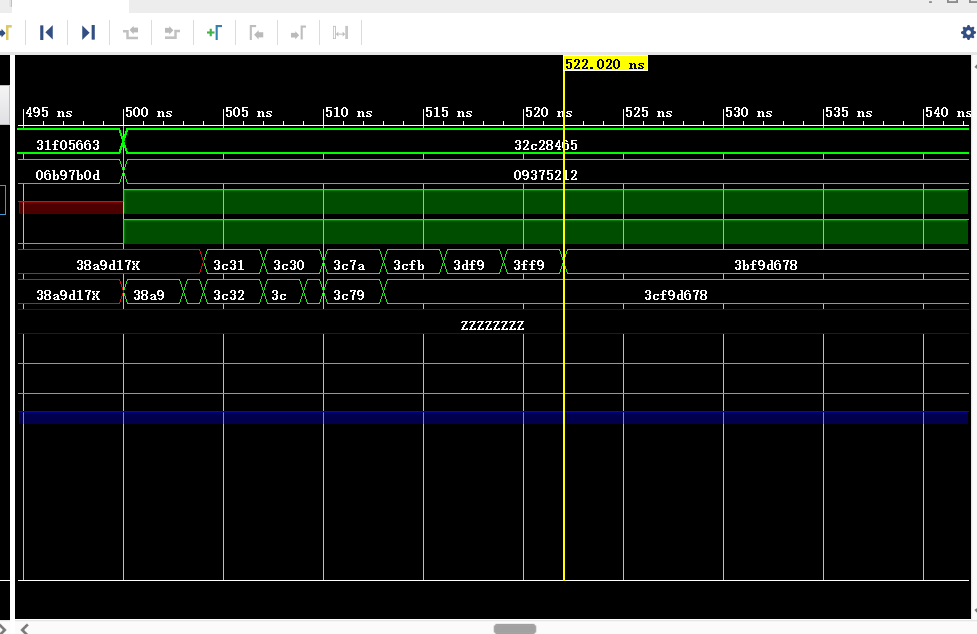
选择进位加法器由8个四位全加器级联构成，每个四位全加器由两个逐位进位加法器构成，一个进位输入0，另一个进位输入1。计算过程中首先并行计算好两种计算结果，最后根据低位传入的进位结果选择计算方案，相比逐位法减少了计算时间。

（3）测试激励的设计

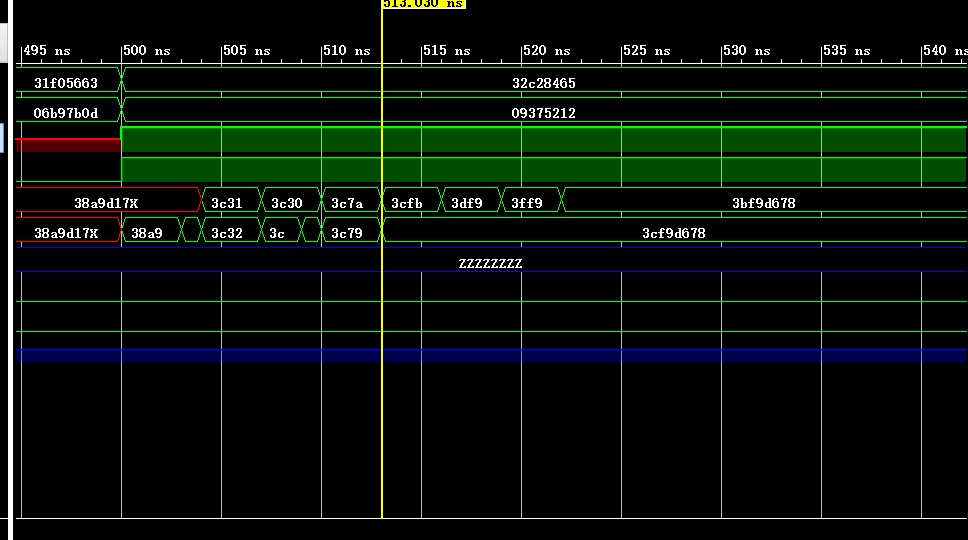
测试方案：测试数据由机器随机生成两个32位数字，两个数字相加得到正确结果，再利用此结果与加法器得到的结果相比较，如果相同则加法器计算正确。

性能比较：两个计算器同时运行进行计算，比较其得到正确结果的时间，耗费时间越短则该加法器效率越高，以此来判断加法器性能。

（4）仿真波形及说明



逐位进位加法器耗费22.020ns



选择进位加法器耗费13.030ns

图中可见随机生成两个32位数字，上方代表逐位加法器，下方代表选择加法器。

不难发现，逐位加法器耗费22.02ns计算正确，选择加法器耗费13.03ns计算正确，选择加法器的性能要由于逐位加法器。

**11、附录：你所实现的各种32位加法器Verilog代码 + 测试激励代码**

32位逐位进位加法器Verilog代码

module rcadd32(

input [31:0] a, input [31:0] b, input cin,

output [31:0] sum, output cout

);

wire [31:0] station;

add1 a1 (a[0], b[0], cin, sum[0], station[0]);

add1 a2 (a[1], b[1], station[0], sum[1], station[1]);

add1 a3 (a[2], b[2], station[1], sum[2], station[2]);

add1 a4 (a[3], b[3], station[2], sum[3], station[3]);

add1 a5 (a[4], b[4], station[3], sum[4], station[4]);

add1 a6 (a[5], b[5], station[4], sum[5], station[5]);

add1 a7 (a[6], b[6], station[5], sum[6], station[6]);

add1 a8 (a[7], b[7], station[6], sum[7], station[7]);

add1 a9 (a[8], b[8], station[7], sum[8], station[8]);

add1 a10 (a[9], b[9], station[8], sum[9], station[9]);

add1 a11 (a[10], b[10], station[9], sum[10], station[10]);

add1 a12 (a[11], b[11], station[10], sum[11], station[11]);

add1 a13 (a[12], b[12], station[11], sum[12], station[12]);

add1 a14 (a[13], b[13], station[12], sum[13], station[13]);

add1 a15 (a[14], b[14], station[13], sum[14], station[14]);

add1 a16 (a[15], b[15], station[14], sum[15], station[15]);

add1 a17 (a[16], b[16], station[15], sum[16], station[16]);

add1 a18 (a[17], b[17], station[16], sum[17], station[17]);

add1 a19 (a[18], b[18], station[17], sum[18], station[18]);

add1 a20 (a[19], b[19], station[18], sum[19], station[19]);

add1 a21 (a[20], b[20], station[19], sum[20], station[20]);

add1 a22 (a[21], b[21], station[20], sum[21], station[21]);

add1 a23 (a[22], b[22], station[21], sum[22], station[22]);

add1 a24 (a[23], b[23], station[22], sum[23], station[23]);

add1 a25 (a[24], b[24], station[23], sum[24], station[24]);

add1 a26 (a[25], b[25], station[24], sum[25], station[25]);

add1 a27 (a[26], b[26], station[25], sum[26], station[26]);

add1 a28 (a[27], b[27], station[26], sum[27], station[27]);

add1 a29 (a[28], b[28], station[27], sum[28], station[28]);

add1 a30 (a[29], b[29], station[28], sum[29], station[29]);

add1 a31 (a[30], b[30], station[29], sum[30], station[30]);

add1 a32 (a[31], b[31], station[30], sum[31], station[31]);

assign cout=station[31];

endmodule

32位选择进位加法器Verilog代码

module S4(

wire [3:0] a,

wire [3:0] b,

input wire cin,

output wire [3:0] out,

output wire cout

);

wire [3:0] sum1;

wire [3:0] sum2;

wire [3:0] station1;

wire [3:0] station2;

add1 a1 (a[0], b[0], 0, sum1[0], station1[0]);

add1 a2 (a[1], b[1], station1[0], sum1[1], station1[1]);

add1 a3 (a[2], b[2], station1[1], sum1[2], station1[2]);

add1 a4 (a[3], b[3], station1[2], sum1[3], station1[3]);

add1 a5 (a[0], b[0], 1, sum2[0], station2[0]);

add1 a6 (a[1], b[1], station2[0], sum2[1], station2[1]);

add1 a7 (a[2], b[2], station2[1], sum2[2], station2[2]);

add1 a8 (a[3], b[3], station2[2], sum2[3], station2[3]);

assign out = cin ? sum2 : sum1;

assign cout = cin ? station1[3] : station2[3];

endmodule

module csadd32(

input [31:0] a, input [31:0] b, input cin,

output [31:0] out, output cout

);

wire [7:0] station;

S4 f1(a[3:0], b[3:0], cin, out[3:0], station[0]);

S4 f2(a[7:4], b[7:4], station[0], out[7:4], station[1]);

S4 f3(a[11:8], b[11:8], station[1], out[11:8], station[2]);

S4 f4(a[15:12], b[15:12], station[2], out[15:12], station[3]);

S4 f5(a[19:16], b[19:16], station[3], out[19:16], station[4]);

S4 f6(a[23:20], b[23:20], station[4], out[23:20], station[5]);

S4 f7(a[27:24], b[27:24], station[5], out[27:24], station[6]);

S4 f8(a[31:28], b[31:28], station[6], out[31:28], station[7]);

assign cout = station[7];

endmodule

测试激励代码

module add32\_tb();

reg [31:0]a;

reg [31:0]b;

reg cin;

reg clk;

wire [31:0]s0,s1,s2;

wire cout0,cout1,cout2;

initial begin

a = 4'bxxxx;

b = 4'bxxxx;

cin = 1'bx;

clk = 0;

end

always #100 clk = ~clk;

always@(posedge clk)begin

a={$random}%2\*\*30;

b={$random}%2\*\*30;

cin={$random}%2;

#150;

cin=1'bx;

end

rcadd32 B(a,b,cin,s0,cout0);//逐位加法器

csadd32 A(a,b,cin,s1,cout1);//选择加法器

endmodule