|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 学部/院 | 智能与计算学部 | 年级 | 2020级 | 班级 | 网络空间安全1班 |
| 姓名 | 石子跃 | 学号 | 3020244294 | 实验日期 | 2022/5/10 |

实验项目名称 算术逻辑单元（ALU）的设计与实现

**一. 实验目的**

1. 掌握全加器和行波进位加法器的结构；

2. 熟悉加减法运算及溢出的判断方法；

3. 掌握算术逻辑单元（ALU）的结构；

4. 熟练使用 SystemVerilog HDL 的行为建模和结构化建模方法对 ALU进行描述实现；

5. 为“单周期 MIPS 处理器的设计与实现”奠定基础。

**二. 实验内容**

基于 SystemVerilog HDL 设计并实现一个 4 位 ALU 单元。整个工程的顶层 模块如图 2-4 所示，输入/输出端口如表 2-2 所示。注意，顶层模块由两个子模 块组成，其中，一个是 ALU 单元，另一个是 7 段数码管动态显示扫描单元。同 学们只需要实现 ALU 单元即可，动态显示扫描单元在工程中直接提供。

图示, 示意图

描述已自动生成表格

描述已自动生成

表格

描述已自动生成

$：所有运算均需要设置 zero 位。

&：算术右移高位补符号位的前提是对有符号数进行右移，需要特别注意这一点。

完成上述 ALU 单元的设计，必需满足如下几点要求：

1. ALU 单元的输入 A 和 B 均是补码形式。

2. 实现加法和减法时，不能使用“+”和“-”两种运算符，且只能通过一个行波进位加法器和其它必要的逻辑电路实现。

3. 可以使用“\*”运算符实现乘法，但该运算符在只适用无符号数的乘法，有符号数的乘法需要同学们考虑如何处理。

4. 实现算术右移时，可以使用运算符“>>>”。

**三．实验原理与步骤（注：步骤不用写工具的操作步骤，而是设计步骤）**

1. 写出全加器fulladder.sv代码。

module fulladder(

input A,

input B,

input Cin,

output logic Cout,

output logic S

);

assign Cout= (A&Cin)|(A&B)|(B&Cin);

assign S=((~A)&(~B)&(Cin))|(A&(~B)&(~Cin))|((~A)&(B)&(~Cin))|(A&B&Cin);

endmodule

2. 给出行波进位加法器rsa.sv的代码。

module rca(

input [3:0]A,

input [3:0]B,

input Cin,

output logic [3:0]S,

output logic Cout

);

logic Cin2;

logic Cin3;

logic Cin4;

fulladder add1(A[0],B[0],Cin,Cin2,S[0]);

fulladder add2(A[1],B[1],Cin2,Cin3,S[1]);

fulladder add3(A[2],B[2],Cin3,Cin4,S[2]);

fulladder add4(A[3],B[3],Cin4,Cout,S[3]);

endmodule

3. 写出ALU模块alu.sv代码。

module alu(

input [3 : 0] A,

input [3 : 0] B,

input [3 : 0] aluop,

output logic [7 : 0] alures,

output logic ZF,

output logic OF

);

logic Coutadd,Coutsub;

logic [3:0] Sadd;

logic [3:0] Ssub;

rca addResult(A,B,0,Sadd,Coutadd);

rca subResult(A,~B,1,Ssub,Coutsub);

always\_comb begin

OF = 0;

alures[7:4] = 4'b0000;

case(aluop)

4'b0000: alures[3:0] = A&B;

4'b0001: alures[3:0] = A|B;

4'b0010: alures[3:0] = A^B;

4'b0011: alures[3:0] = ~(A&B);

4'b0100: alures[3:0] = ~A;

4'b0101: alures[3:0] = A<<B[2:0];

4'b0110: alures[3:0] = A>>B[2:0];

4'b0111: alures[3:0] = $signed(A)>>>B[2:0];

4'b1000: alures = A\*B;

4'b1001: alures= {{4{A[3]}},A}\*{{4{B[3]}},B};

4'b1010: begin

alures[3:0] = Sadd;

if(A[3]==B[3] && Sadd[3]!=A[3]) OF = 1;

else OF = 0;

end

4'b1011: alures[3:0] = Sadd;

4'b1100: begin

alures[3:0] = Ssub;

if(A[3]!=B[3] && Ssub[3]!=A[3]) OF = 1;

else OF = 0;

end

4'b1101: alures[3:0] = Ssub;

4'b1110: alures[3:0]=$signed(A)<$signed(B)?1:0;

4'b1111: alures[3:0] = A<B?1:0;

endcase

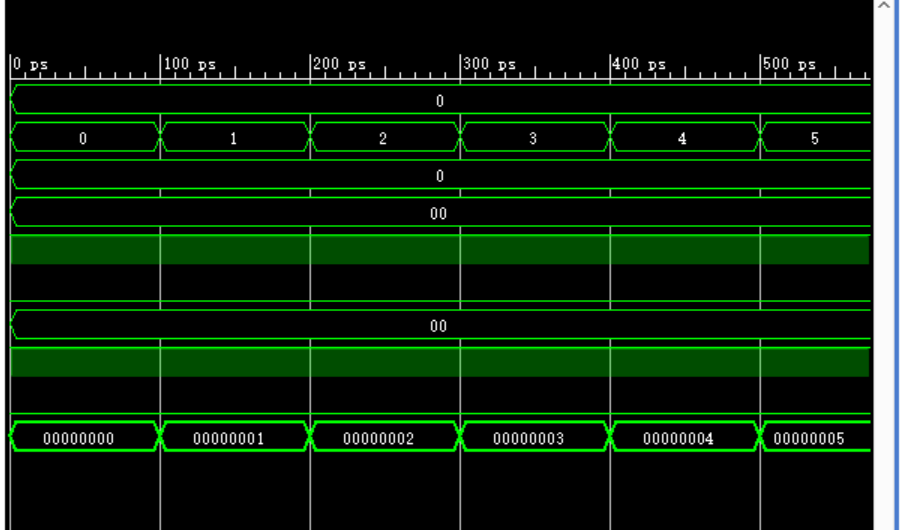
end

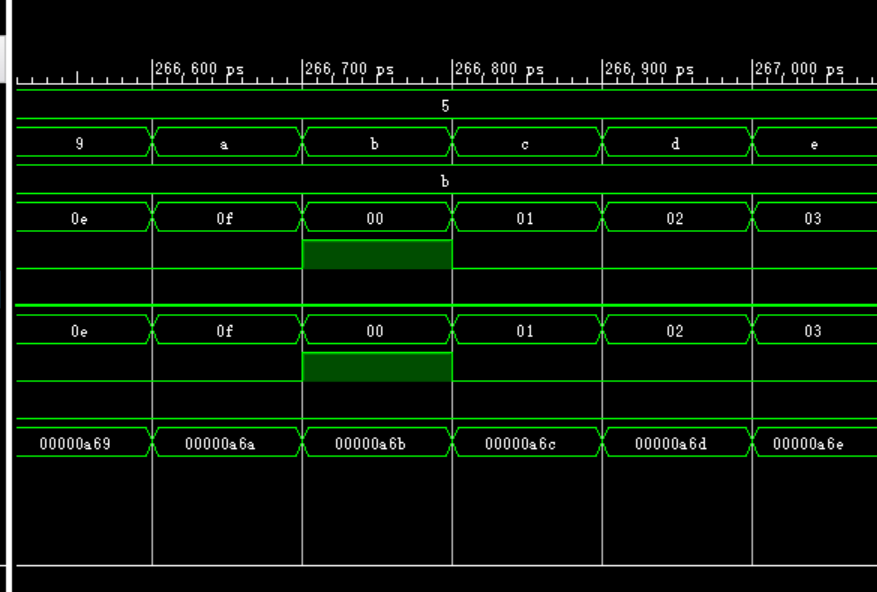
assign ZF = (alures==0) ? 1 : 0;

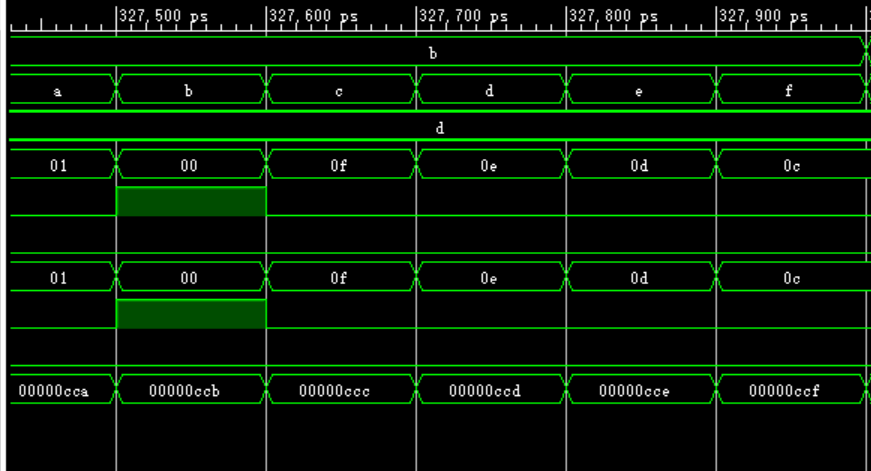
endmodule

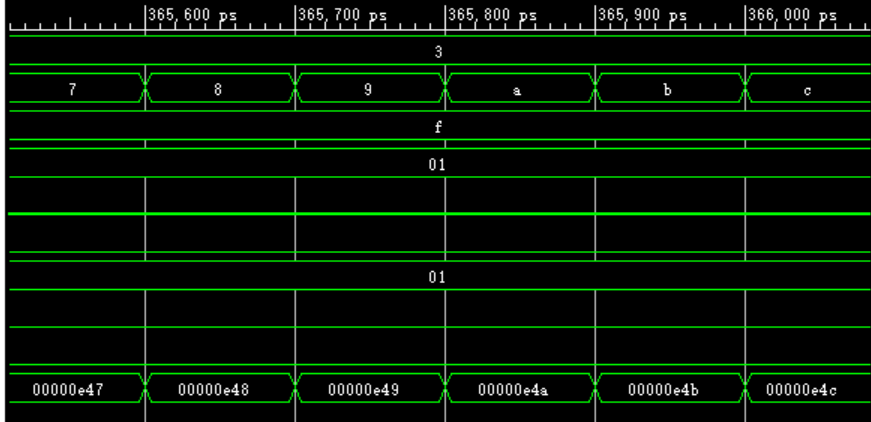
**四．仿真与实验结果（注：仿真需要给出波形图截图，截图要清晰，如果波形过长，可以分段截取；实验结果为远程FPGA硬件云平台的截图）**

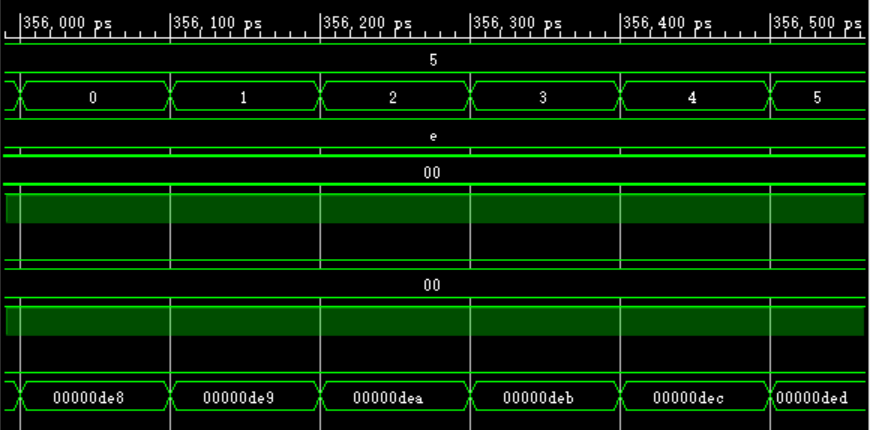
注：远程FPGA硬件云平台截图只需要一个测试激励即可

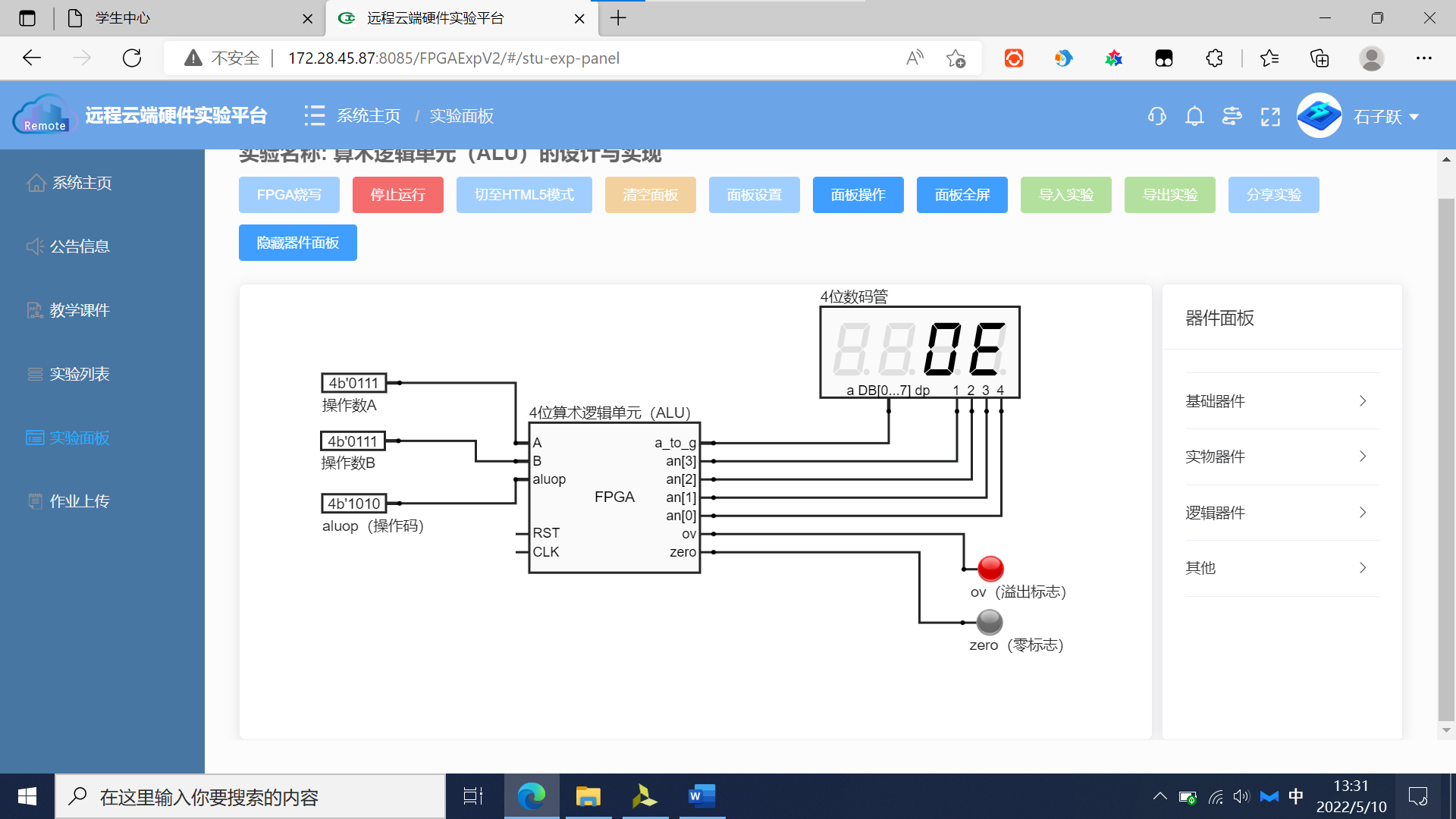












**五．实验中遇到的问题和解决办法**

算术右移：

算术右移是有符号数的右移，使用$signed(A)>>>B[2:0];只要系统知道了这是个有符号数，那么直接用‘>>>’就可以了。

有符号数加法：

由于A和B都是以补码的形式存在，所以进行加法时就可以直接相加就可以，不管他的正负，最后再根据结果判断是否有溢出和正负情况，如果两个有符号数相加且符号相同，如果结果的符号位改变，那么证明会有溢出的出现

有符号数乘法：

将其按照符号位扩展为八位后，可用\*计算。

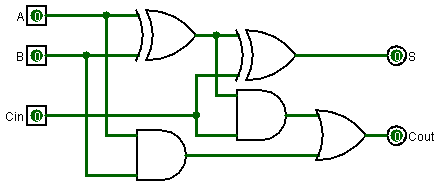
有符号数减法：

与加法类似，只不过需要多一步将‘-B’转变为‘+（-B）’，再补码中只需要将B取非加一就可以得到他的相反数，但是要注意正数最大到7，而负数最小为-8，补码为1000，溢出的判断条件改为当两操作数的符号位不同时，若结果的符号位与被减数的符号位不同则发生溢出。

其他的所有情况都没有什么难度。

**六．简答题**

1. 画出实现加/减法运算的逻辑电路原理图，并说明为什么加/减法可以只使用一个加法器进行实现？



A为被加数

B为加数

Cin为相邻低位来的进位数

S为相加后输出本位

Cout为向相邻高位进位数

S = A ^ B ^ Cin

Cout = (A ^ B) & Cin + A & B

如图所示，加法。至于减法，只需要将A-B转化为A+（-B）即可，再根据补码的规则-B=~B+1，就可以将减法转化为加法。这时将传入的B变为反B，Cin置为1，即可完成A-B的操作。

2. 给出有符号数加/减法溢出的判断规则？

如果两个有符号数相加且符号相同，如果结果的符号位改变，那么证明会有溢出的出现。

减法当两操作数的符号位不同时，若结果的符号位与被减数的符号位不同则发生溢出。

对于有符号数减法(A-B)：若A正B负，结果S是负数(S[3]=1)，溢出；若A负B正，结果S是正数(S[3]=0)，溢出。特殊的如果B是1000，是没有对应的正数的，这时候如果A[3]=0,也就是说A是正数，则一定会溢出。其他情况就不会溢出。

对于有符号数加法：若A和B都是负数，结果S是正数，溢出；若A和B都是正数，结果S是负数，溢出，其他情况均不会造成溢出。

3. 请说明在自动化仿真测试中，如何合理地设计测试向量，选择测试向量的原则是什么？

1）首先测试向量需要覆盖到操作类型（aluop）的每一种情况。

2）测试向量之间差别要尽量大一些，简单的问题没要一直重复。

3）测试向量要尽可能覆盖到所有特殊值、特殊情况、临界情况，保证电路在各种情况下都能得到正确结果。

**教师签字：**

**年 月 日**