数字电路与数字系统实验

EX03:加法器与 ALU

191220029 傅小龙
周一 5-6 节班
1830970417@qq.com
2020 年 9 月 14 日

目录

一、实验内容……3
1.1 实验要求……3
1.2 实验工具……3
二、实验过程……4
2.1 简单加减法运算器
2.1.1 模型概述……4
2.1.2 数字抽象……4
2.1.3 建立模型……4
2.1.4 分析/综合…… 5
2.1.5 仿真测试……6
2.1.6 分配引脚……8
2.1.7 全编译……9
2.2 带逻辑运算的简单 ALU
2.2.1 模型概述……9
2.2.2 数字抽象……9
2.2.3 建立模型……10
2.2.4 分析/综合…… 12
2.2.5 仿真测试……13
2.2.6 分配引脚……17
2.2.7 全编译……18
三、实验总结……18
四、附……18
4.1 实验中遇到的问题及处理办法……18
4.2 关于思考题……19

一、实验内容

1.1 实验要求

复习全加器的原理，学习简单 ALU 的设计方式.

I)简单加减法运算器的设计

根据硬件资源，完成一个进行补码加减运算的 4 位加减运算器，此加减运算器的核心部件是一个 4 位加法器，能够根据控制端完成加、减运算，并能判断结果是否为 0，是否溢出，是否有进位等。这里，输入的操作数 A 和 B 都已经是补码。

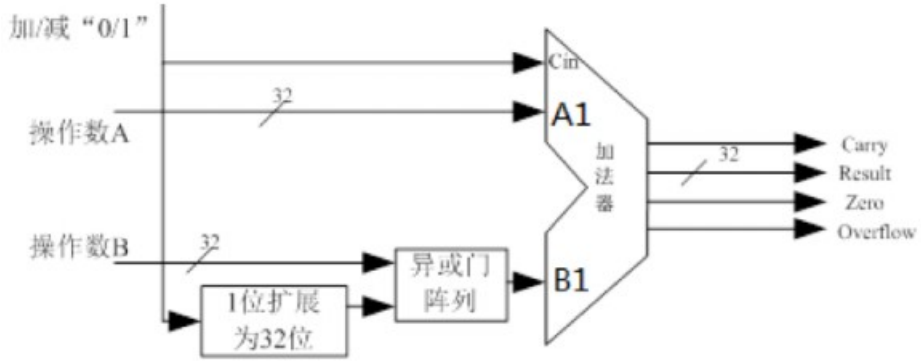


图 1-1：简单加减 ALU

II)带有逻辑运算的简单 ALU

设计一个能实现如下功能的对 4 位有符号数操作的 ALU

功能选择	功能	操作
000	加法	A+B
001	减法	A-B
010	取反	Not A
011	与	A and B
100	或	A or B
101	异或	A xor B
110	比较大小	If A>B then out = 1; else out = 0;
111	判断相等	If A==B then out = 1; else out = 0;

表 1-1：ALU 功能列表

考虑各种运算的进位位 C 和溢出位 overflflow 位的输出。（一般情况下，涉及加减运算的，可以按照加减运算器来考虑进位位和溢出位；涉及逻辑运算的，可以直接设置进位位和溢出位为“0”。比较大小时需要考虑符号位。）

1.2 实验工具

软件环境：

设计、编译、仿真：Quartus Prime Version 17.1.0 Build 590 10/25/2017 SJ Lite Edition

DE10_Standard_SystemBuilder

硬件环境： DE-10 Standard 开发平台

FPGA 芯片： Cyclone V 5CSXFC6D6F31C6

二、实验过程

2.1 简单加减法运算器

2.1.1 模型概述

使用 Verilog HDL 实现一个 4 位二进制补码的简单加减法运算器. 能够输出运算结果和进位、溢出的标志位.

2.1.2 数字抽象

I) 输入:

控制信号 ctrl： 控制运算器进行加法或减法运算.

数据输入[3:0]A, B： 4 位二进制补码操作数的输入.

II) 输出:

数据输出[3:0]C： 输出运算结果.

标志位 out_c： 进位标志.

标志位 overflow:溢出标志.

标志位 isZero： 零标志.

下表\图给出了以上输入输出信号在 DE10 平台对应的信号：

	信号名称	DE2-70 平台信号
输入	ctrl	SW[9]
	[3:0]A	SW[3:0]
	[3:0]B	SW[7:4]
输出	[2:0]y	LEDR[3:0]
	out_c	LEDR[7]
	overflow	LEDR[8]
	isZero	LEDR[9]

表 2-1-1:简单加减法运算器的输入输出信号与 DE10 平台信号对应关

2.1.3 建立模型

下表给出了 8-3 优先编码器的输出与输入的关系：

输入		输出				
控制信号 ctrl	数据输入 [3:0]A	数据输入 [3:0]B	输出信号 [3:0]C	标志位 out_c	标志位 overflow	标志位 isZero

0	A	B	{out_c, C} = A + B	(A[3] == B[3]) && (C[3] != A[3])	~(|C)
1	A	B	{out_c, C} = A + B 补	(A[3] != B[3]) && (C[3] != A[3])	

表 2-1-2:简单加减法运算器行为表

实现思路：使用 if 语句，根据控制信号 ctrl 的值选择要对输入的数进行加法还是减法操作. 加法、减法的具体实现思路见 2.2.3 中的实现思路的①、②部分.

简单加减法运算器的 Verilog HDL 实现如下：

```
module s_ALU4(ctrl, A, B, C, out_c, overflow, isZero);
    input ctrl;
    input [3:0]A;
    input [3:0]B;
    output reg [3:0]C;
    output reg out_c;
    output reg overflow;
    output reg isZero;
    reg [3:0]B_com;

    always @(*) begin
        isZero = 0;
        if(ctrl == 0) //加法
        begin
            {out_c, C} = A + B;
            overflow = (A[3] == B[3]) && (C[3] != A[3]);
            isZero = ~(|C);
        end
        else begin // 减法
            B_com = ~B + 1;
            {out_c, C} = A + B_com;
            overflow = (A[3] != B[3]) && (C[3] != A[3]);
            isZero = ~(|C);
        end
    end
endmodule
```

2.1.4 分析/综合

分析/综合实验成功，如下图所示：

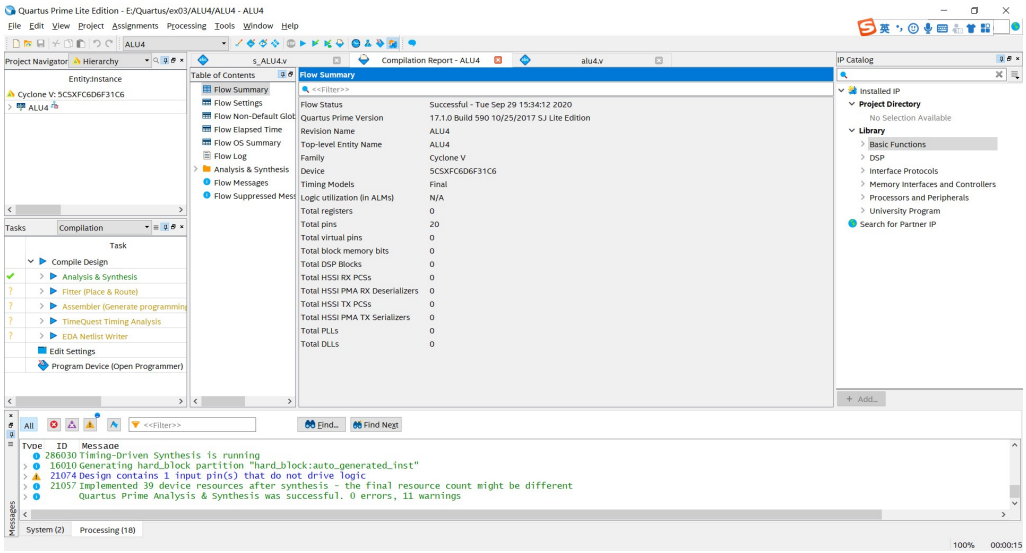
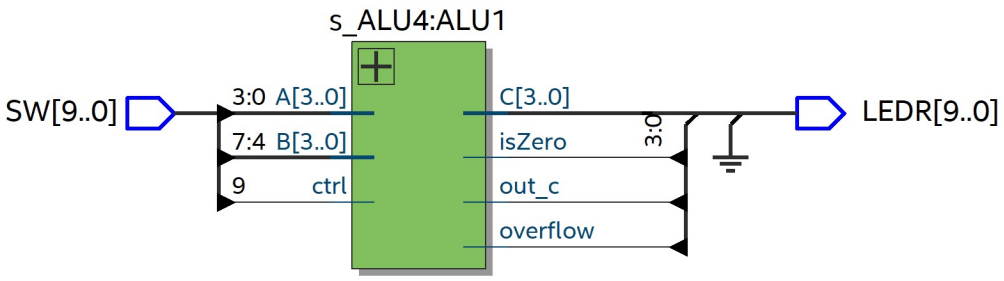


图 2-1-1：分析/综合成功



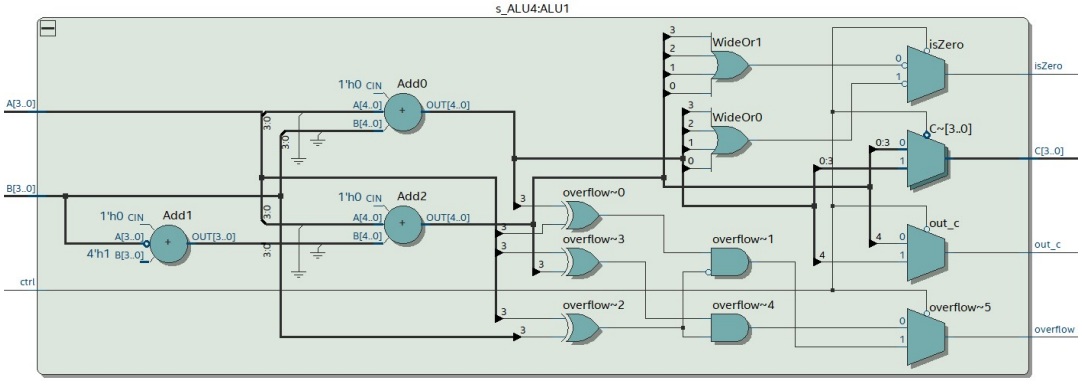


图 2-1-2:RTL 视图

2.1.5 仿真测试

由于本模型操作数较多，且位数较多，不便于通过观察波形图来进行枚举测试. 在这里使用 task 功能进行自动测试.task 的代码如下：

```
1    task ALU_check;
2        input ctrl;
3        input [3:0]a, b, c, std_c;
```

```
        input CF, std_CF, OF, std_OF, ZF, std_ZF;
        begin
            $display("ctrl = %b, a = %h, b = %h, c = %h, CF = %b, ZF
    = %b, OF = %b", ctrl, a, b, c, CF, ZF, OF);
            if(c != std_c)
                $display("c is wrong. std_c = %h", std_c);
            if(CF != std_CF)
                $display("CF is wrong. std_CF = %b", std_CF);
            if(OF != std_OF)
                $display("OF is wrong. std_OF = %b", std_OF);
            if(ZF != std_ZF)
                $display("ZF is wrong. std_ZF = %b", std_ZF);

        end
    endtask
```

带有"std_"前缀的输入值是期望的 ALU 输出，若与其对应的实际输出相符则会输出该测试样例和得到的运算结果，不符则会另外输出错误的项和正确的值.

对本模型的 ALU 的加法和减法功能给出如下测试样例：

```
    //加法测试：
    $display("Plus test:");
    SW[9] = 0;
                //1+1
                SW[3:0] = 1;
                SW[7:4] = 1;
                #20 ALU_check(SW[9], SW[3:0], SW[7:4], LEDR[3:0], 2,
    LEDR[7], 0, LEDR[8], 0, LEDR[9], 0);
                //4+5 OF = 1
                SW[3:0] = 4;
                SW[7:4] = 5;
                #20 ALU_check(SW[9], SW[3:0], SW[7:4], LEDR[3:0], 9,
    LEDR[7], 0, LEDR[8], 1, LEDR[9], 0);
                //1+(-1) ZF = 1; CF = 1
                SW[3:0] = 1;
                SW[7:4] = 4'b1111;
                #20 ALU_check(SW[9], SW[3:0], SW[7:4], LEDR[3:0], 0,
    LEDR[7], 1, LEDR[8], 0, LEDR[9], 1);

    //减法测试：
    $display("Minus test:");
    SW[9] = 1;
                //4-5
                SW[3:0] = 4;
                SW[7:4] = 5;
                #20 ALU_check(SW[9], SW[3:0], SW[7:4], LEDR[3:0],
```

```
    4'b1111, LEDR[7], 0, LEDR[8], 0, LEDR[9], 0);
            //7-6 CF = 1;
            SW[3:0] = 7;
            SW[7:4] = 6;
            #20 ALU_check(SW[9], SW[3:0], SW[7:4], LEDR[3:0], 1,
    LEDR[7], 1, LEDR[8], 0, LEDR[9], 0);
            //6-(-4) OF = 1;
            SW[3:0] = 6;
            SW[7:4] = 4'b1100;
            #20 ALU_check(SW[9], SW[3:0], SW[7:4], LEDR[3:0], 10,
    LEDR[7], 0, LEDR[8], 1, LEDR[9], 0);
    // --> end
```

以上样例分别对 ALU 能否得到正确的运算结果，进位位、零指示位和溢出位的正确计算作为考察点. 通过 ModelSim 得到的仿真结果如下：

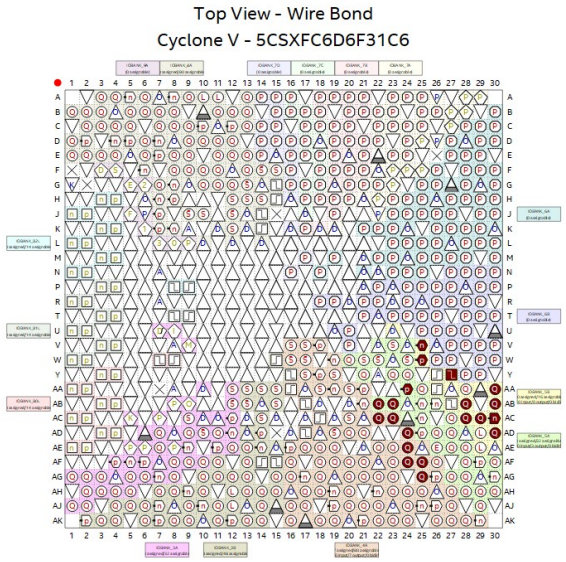
```
# Check starts here
# ************************************
# Plus test:
# ctrl = 0, a = 1, b = 1, c = 2, CF = 0, ZF = 0, OF = 0
# ctrl = 0, a = 4, b = 5, c = 9, CF = 0, ZF = 0, OF = 1
# ctrl = 0, a = 1, b = f, c = 0, CF = 1, ZF = 1, OF = 0
# Minus test:
# ctrl = 1, a = 4, b = 5, c = f, CF = 0, ZF = 0, OF = 0
# ctrl = 1, a = 7, b = 6, c = 1, CF = 1, ZF = 0, OF = 0
# ctrl = 1, a = 6, b = c, c = a, CF = 0, ZF = 0, OF = 1
# Check is over.
# ************************************
```

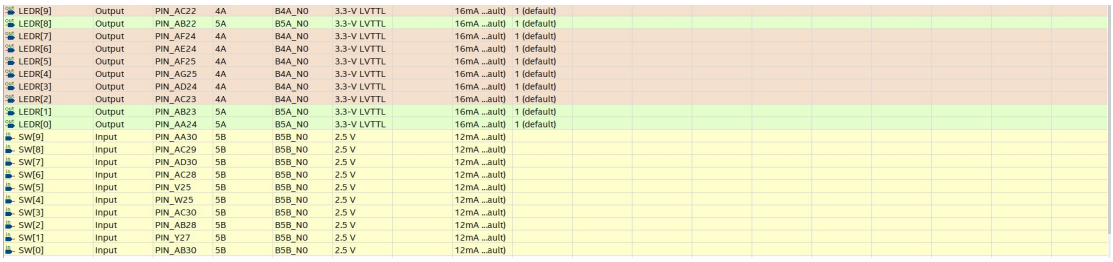
图 2-1-3：仿真结果

所有的测试样例均通过.

2.1.6 分配引脚

引脚分配使用 DE10_Standard_SystemBuilder 生成。





LEDR[9]	Output	PIN_AC22	4A	B4A_N0	3.3-V LVTTL	16mA ...ault)	1 (default)
LEDR[8]	Output	PIN_AB22	5A	B5A_N0	3.3-V LVTTL	16mA ...ault)	1 (default)
LEDR[7]	Output	PIN_AF24	4A	B4A_N0	3.3-V LVTTL	16mA ...ault)	1 (default)
LEDR[6]	Output	PIN_AE24	4A	B4A_N0	3.3-V LVTTL	16mA ...ault)	1 (default)
LEDR[5]	Output	PIN_AF25	4A	B4A_N0	3.3-V LVTTL	16mA ...ault)	1 (default)
LEDR[4]	Output	PIN_AG25	4A	B4A_N0	3.3-V LVTTL	16mA ...ault)	1 (default)
LEDR[3]	Output	PIN_AD24	4A	B4A_N0	3.3-V LVTTL	16mA ...ault)	1 (default)
LEDR[2]	Output	PIN_AC23	4A	B4A_N0	3.3-V LVTTL	16mA ...ault)	1 (default)
LEDR[1]	Output	PIN_AB23	5A	B5A_N0	3.3-V LVTTL	16mA ...ault)	1 (default)
LEDR[0]	Output	PIN_AA24	5A	B5A_N0	3.3-V LVTTL	16mA ...ault)	1 (default)
SW[9]	Input	PIN_AA30	5B	B5B_N0	2.5 V	12mA ...ault)	
SW[8]	Input	PIN_AC29	5B	B5B_N0	2.5 V	12mA ...ault)	
SW[7]	Input	PIN_AD30	5B	B5B_N0	2.5 V	12mA ...ault)	
SW[6]	Input	PIN_AC28	5B	B5B_N0	2.5 V	12mA ...ault)	
SW[5]	Input	PIN_V25	5B	B5B_N0	2.5 V	12mA ...ault)	
SW[4]	Input	PIN_W25	5B	B5B_N0	2.5 V	12mA ...ault)	
SW[3]	Input	PIN_AC30	5B	B5B_N0	2.5 V	12mA ...ault)	
SW[2]	Input	PIN_AB28	5B	B5B_N0	2.5 V	12mA ...ault)	
SW[1]	Input	PIN_Y27	5B	B5B_N0	2.5 V	12mA ...ault)	
SW[0]	Input	PIN_AB30	5B	B5B_N0	2.5 V	12mA ...ault)	

图 2-1-3 引脚分配图

2.1.7 全编译

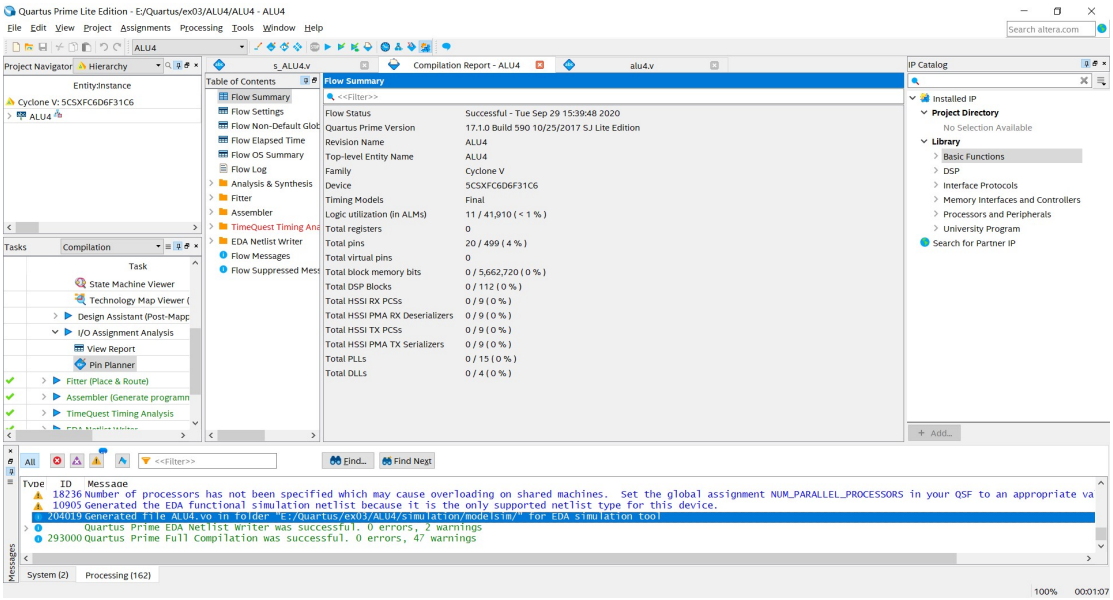


图 2-1-4 全编译成功

2.2 带逻辑运算的简单 ALU

2.2.1 模型概述

使用 Verilog HDL 实现一个带逻辑运算的简单 ALU. 能够根据不同的功能选择输入信号，实现对 2 个 4 位补码二进制数输入的不同运算. 具体的运算操作与功能选择信号的关系在表 1-1 中已给出.

2.2.2 数字抽象

I) 输入:

功能选择信号[2:0]ctrl：控制 ALU 执行相应的功能.

操作数输入[3:0]A, B： 操作数以 4 位二进制补码的形式输入.

II) 输出:

输出端[3:0]Y：以 4 位二进制补码的形式输出运算结果.

标志位 CF: 进位标志.

标志位 OF: 溢出标志.

下表\图给出了以上输入输出信号在 DE10 平台对应的信号：

	信号名称	DE2-70 平台信号
输入	[2:0]ctrl	KEY[3:0]
	[3:0]A	SW[3:0]
	[3:0]B	SW[7:4]
输出	[3:0]Y	LEDR[3:0]
	CF	LEDR[9]
	OF	LEDR[8]

表 2-2-1:带逻辑运算的简单 ALU 的输入输出信号与 DE10 平台信号对应关系

2.2.3 建立模型

下表给出了 8-3 优先编码器的输出与输入的关系：

输入			输出		
功能选择信号[2:0]ctrl	操作数[3:0]A	操作数[3:0]B	输出信号[2:0]Y	输出指示信号CF	输出指示信号OF
3’b000	A	B	{CF, Y| = A + B		OF = (A[3] == B[3]) && (Y[3] != A[3])
3’b001	A	B	{CF, Y} = A + B 补		OF = (A[3] != B[3]) && (Y[3] != A[3])
3’b010	A	X	~A	0	0
3’b011	A	B	A & B	0	0
3’b100	A	B	A | B	0	0
3’b101	A	B	A ^ B	0	0
3’b110	A	B	A > B(带符号比较)	0	0
3’b111	A	B	A == B	0	0

表 2-2-2:带逻辑运算的简单 ALU 行为表

实现思路：使用 case 语句实现各功能，根据功能选择信号调用对应的块.

①加法：进位信号可以用下面的表达式得到：

```
{CF, Y} = A + B;
```

该表达式执行后 CF 得到进位信号，Y 为运算结果.对于溢出信号，可以通过检测运算结果和操作数的符号的一致性得到：若两个操作数符号相同但运算结果的符号与之相异，则发生溢出.

②减法：对减数 B 取其负数的补码与 A 相加即可得到对应的差.进位信号可由下面的表达式得到：

```
B_com = ~B + 1;
{CF, Y} = A + B_com;
```

溢出信号与加法对溢出信号的计算相类似：若上面 2 式中的 A 和 B_com 符号相同但与运算结果 Y 符号不同则发生溢出.

③按位取反：Y = ~A;

④逻辑与：Y = A & B;

⑤逻辑或：Y = A | B;

⑥异或：Y = A ^ B;

⑦比较大小：由于需要考虑符号位，故将输入的 2 个二进制补码操作数转换为整型数比较.比较结果可由如下语句实现：

```
integer a;
integer b;
a = -A[3]*8+A[2]*4+A[1]*2+A[0];
b = -B[3]*8+B[2]*4+B[1]*2+B[0];
Y = a > b;
```

带逻辑运算的简单 ALU 的 Verilog HDL 实现如下：

```
module myALU4_2(ctrl, A, B, Y, CF, OF);
    input [2:0]ctrl;
    input [3:0]A;
    input [3:0]B;
    output reg [3:0]Y;
    output reg CF;
    output reg OF;
    integer a;
    integer b;
    reg [3:0]B_com;

    always @(*) begin
           CF = 0; OF = 0; Y = 0;
       case(ctrl)
           0:begin   //加法
               {CF, Y} = A + B;
               OF = (A[3] == B[3]) && (Y[3] != A[3]);
           end
           1:begin   //减法
               //Y = 4'b1110;
               B_com = ~B + 1;
               {CF, Y} = A + B_com;
               OF = (A[3] != B[3]) && (Y[3] != A[3]);
           end
           2:begin   //取反
               Y = ~A;
               //Y = 4'b1100;
           end
           3:begin   //与
               Y = A & B;
           end
```

```
32                  4:begin    //或
33                      Y = A | B;
34                  end
35                  5:begin    //异或
36                      Y = A ^ B;
37                  end
38                  6:begin    //比较大小
39                      a = -A[3]*8+A[2]*4+A[1]*2+A[0];
40                      b = -B[3]*8+B[2]*4+B[1]*2+B[0];
41                      Y = a > b;
42                  end
43                  7:begin    //判断相等
44                      Y = (A == B);
45                  end
46              endcase
47          end
48      endmodule
```

2.2.4 分析/综合

分析/综合实验成功，如下图所示：

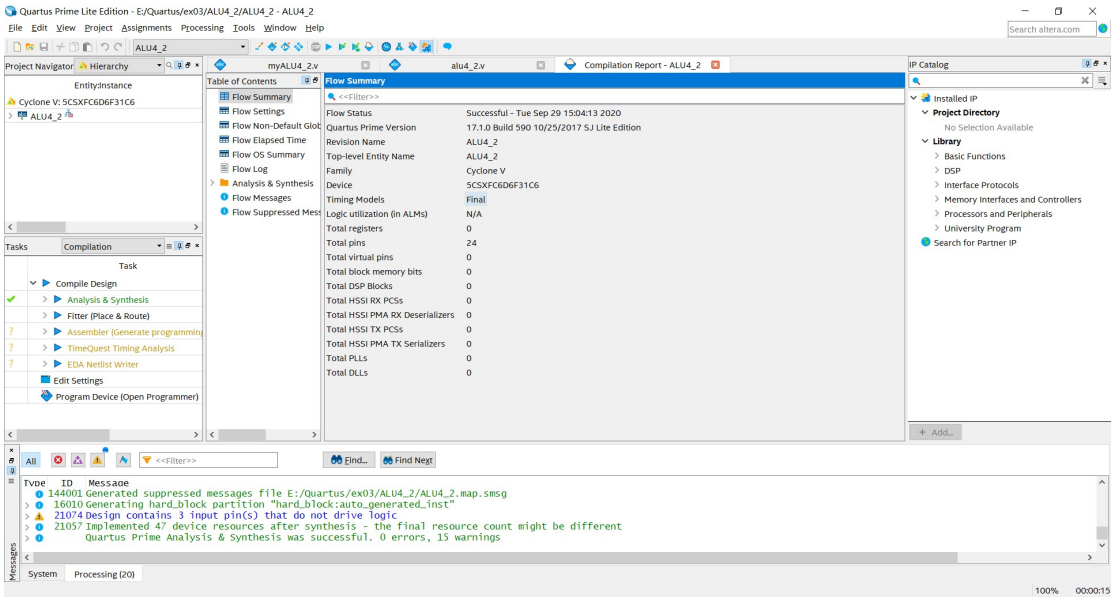
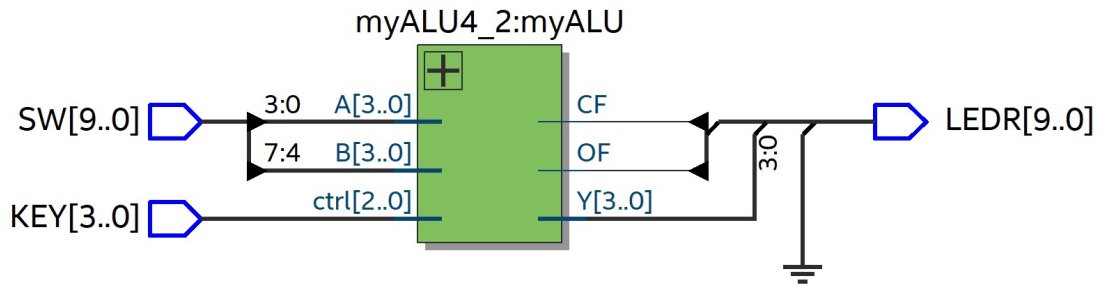


图 2-2-1：分析/综合成功



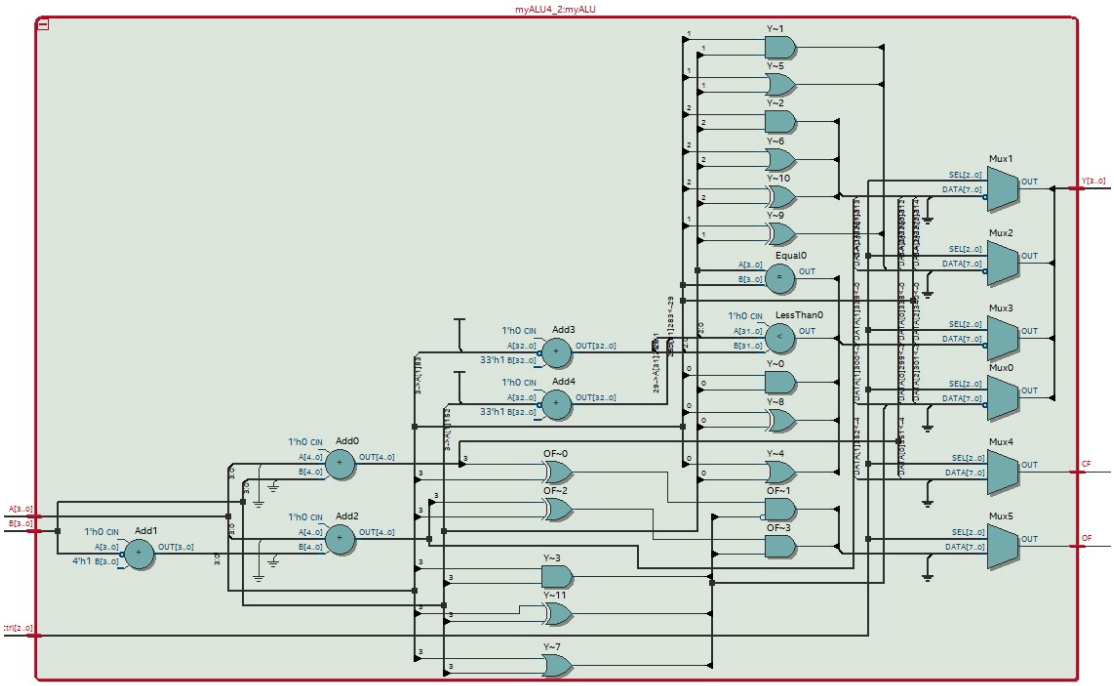


图 2-2-2:RTL 视图

2.2.5 仿真测试

类似 2.1.5 中所述，采用 task 方式对该模型进行仿真测试. task 代码如下：

```
1    task ALU_check;
2        input [2:0]ctrl;
3        input [3:0]a, b, c, std_c;                    //测试结果和预期结果
4        input CF, std_CF, OF, std_OF;
5
6        begin
7            $display("ctrl = %b, a = %h, b = %h, c = %h, CF = %b, OF
8    = %b", ctrl, a, b, c, CF, OF);
9            if(c != std_c)
10               $display("c is wrong. std_c = %h", std_c);
11           if(CF != std_CF)
```

```
            $display("CF is wrong. std_CF = %b", std_CF);
        if(OF != std_OF)
            $display("OF is wrong. std_OF = %b", std_OF);
    end
endtask
```

带有"std_"前缀的输入值是期望的 ALU 输出，若与其对应的实际输出相符则会输出该测试样例和得到的运算结果，不符则会另外输出错误的项和正确的值.

对本模型的 ALU 的加法和减法功能给出如下测试样例：

```
//加法：
$display("Plus test:");
KEY[2:0] = 0;
        //1+1
        SW[3:0] = 1;
        SW[7:4] = 1;
        #20 ALU_check(KEY[2:0], SW[3:0], SW[7:4], LEDR[3:0], 2,
LEDR[9], 0, LEDR[8], 0);
        //4+5 OF = 1
        SW[3:0] = 4;
        SW[7:4] = 5;
        #20 ALU_check(KEY[2:0], SW[3:0], SW[7:4], LEDR[3:0], 9,
LEDR[9], 0, LEDR[8], 1);
        //1+(-1) CF = 1
        SW[3:0] = 1;
        SW[7:4] = 4'b1111;
        #20 ALU_check(KEY[2:0], SW[3:0], SW[7:4], LEDR[3:0], 0,
LEDR[9], 1, LEDR[8], 0);

//减法：
$display("Minus test:");
KEY[2:0] = 1;
        //4-5
        SW[3:0] = 4;
        SW[7:4] = 5;
        #20  ALU_check(KEY[2:0],  SW[3:0],  SW[7:4],  LEDR[3:0],
4'b1111, LEDR[9], 0, LEDR[8], 0);
        //7-6 CF = 1;
        SW[3:0] = 7;
        SW[7:4] = 6;
        #20 ALU_check(KEY[2:0], SW[3:0], SW[7:4], LEDR[3:0], 1,
LEDR[9], 1, LEDR[8], 0);
        //6-(-4) OF = 1;
        SW[3:0] = 6;
        SW[7:4] = 4'b1100;
        #20 ALU_check(KEY[2:0], SW[3:0], SW[7:4], LEDR[3:0], 10,
```

```
37    LEDR[9], 0, LEDR[8], 1);
38    //取反:
39    $display("Neg test:");
40    KEY[2:0] = 2;
41          SW[3:0] = 4'b1010;
42          #20  ALU_check(KEY[2:0],  SW[3:0],  SW[7:4],  LEDR[3:0],
43    4'b0101, LEDR[9], 0, LEDR[8], 0);
44    //与:
45    $display("And test:");
46    KEY[2:0] = 3;
47          SW[3:0] = 4'b1010;
48          SW[7:4] = 4'b0110;
49          #20  ALU_check(KEY[2:0],  SW[3:0],  SW[7:4],  LEDR[3:0],
50    4'b0010, LEDR[9], 0, LEDR[8], 0);
51    //或:
52    $display("Or test:");
53    KEY[2:0] = 4;
54          SW[3:0] = 4'b1010;
55          SW[7:4] = 4'b1100;
56          #20  ALU_check(KEY[2:0],  SW[3:0],  SW[7:4],  LEDR[3:0],
57    4'b1110, LEDR[9], 0, LEDR[8], 0);
58    //异或:
59    $display("Xor test:");
60    KEY[2:0] = 5;
61          SW[3:0] = 4'b1010;
62          SW[7:4] = 4'b1100;
63          #20  ALU_check(KEY[2:0],  SW[3:0],  SW[7:4],  LEDR[3:0],
64    4'b0110, LEDR[9], 0, LEDR[8], 0);
65    //比较大小:
66    $display("GreaterThan test:");
67    KEY[2:0] = 6;
68          //5 > 1
69          SW[3:0] = 5;
70          SW[7:4] = 4;
71          #20 ALU_check(KEY[2:0], SW[3:0], SW[7:4], LEDR[3:0], 1,
72    LEDR[9], 0, LEDR[8], 0);
73          //5 > -2
74          SW[3:0] = 5;
75          SW[7:4] = 4'b1110;
76          #20 ALU_check(KEY[2:0], SW[3:0], SW[7:4], LEDR[3:0], 1,
77    LEDR[9], 0, LEDR[8], 0);
78          //1 < 5
79          SW[3:0] = 1;
80          SW[7:4] = 5;
```

```
81          #20 ALU_check(KEY[2:0], SW[3:0], SW[7:4], LEDR[3:0], 0,
82    LEDR[9], 0, LEDR[8], 0);
83              //-2 < 5
84              SW[3:0] = 4'b1110;
85              SW[7:4] = 5;
86              #20 ALU_check(KEY[2:0], SW[3:0], SW[7:4], LEDR[3:0],
87    0, LEDR[9], 0, LEDR[8], 0);
88    //判断相等:
89    $display("Equal test:");
90    KEY[2:0] = 7;
91              // 5!=-2
92              SW[3:0] = 5;
93              SW[7:4] = 4'b1110;
94              #20 ALU_check(KEY[2:0], SW[3:0], SW[7:4], LEDR[3:0],
95    0, LEDR[9], 0, LEDR[8], 0);
96              //1==1
97              SW[3:0] = 1;
98              SW[7:4] = 1;
99              #20 ALU_check(KEY[2:0], SW[3:0], SW[7:4], LEDR[3:0],
100   1, LEDR[9], 0, LEDR[8], 0);
101   // --> end
```

以上样例分别对 ALU 能否得到正确的运算结果，进位位、溢出位的正确计算作为考察点. 通过 ModelSim 得到的仿真结果如下：

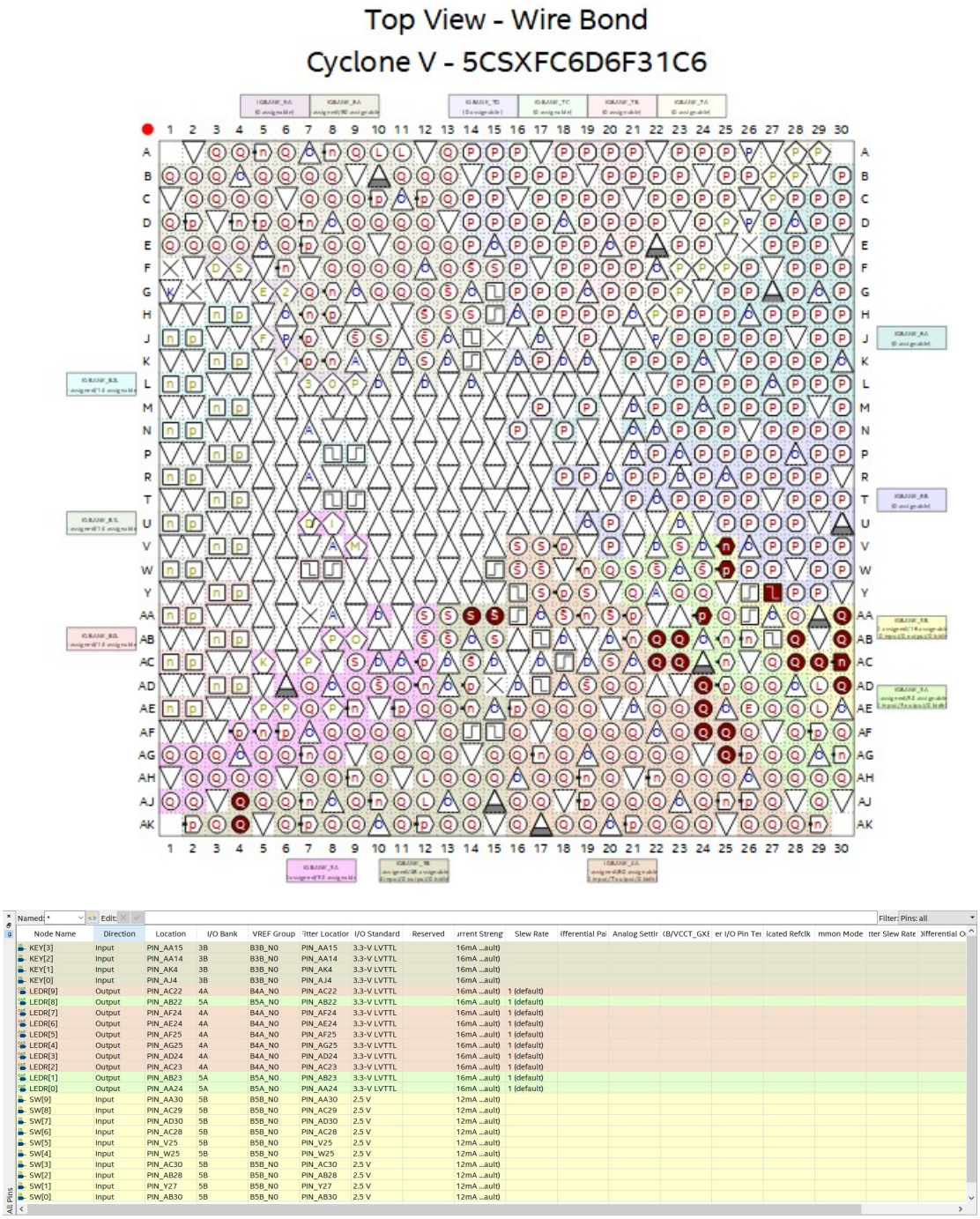
```
# Plus test:
# ctrl = 000, a = 1, b = 1, c = 2, CF = 0, OF = 0
# ctrl = 000, a = 4, b = 5, c = 9, CF = 0, OF = 1
# ctrl = 000, a = 1, b = f, c = 0, CF = 1, OF = 0
# Minus test:
# ctrl = 001, a = 4, b = 5, c = f, CF = 0, OF = 0
# ctrl = 001, a = 7, b = 6, c = 1, CF = 1, OF = 0
# ctrl = 001, a = 6, b = c, c = a, CF = 0, OF = 1
# Neg test:
# ctrl = 010, a = a, b = c, c = 5, CF = 0, OF = 0
# And test:
# ctrl = 011, a = a, b = 6, c = 2, CF = 0, OF = 0
# Or test:
# ctrl = 100, a = a, b = c, c = e, CF = 0, OF = 0
# Xor test:
# ctrl = 101, a = a, b = c, c = 6, CF = 0, OF = 0
# GreaterThan test:
# ctrl = 110, a = 5, b = 4, c = 1, CF = 0, OF = 0
# ctrl = 110, a = 5, b = e, c = 1, CF = 0, OF = 0
# ctrl = 110, a = 1, b = 5, c = 0, CF = 0, OF = 0
# ctrl = 110, a = e, b = 5, c = 0, CF = 0, OF = 0
# Equal test:
# ctrl = 111, a = 5, b = e, c = 0, CF = 0, OF = 0
# ctrl = 111, a = 1, b = 1, c = 1, CF = 0, OF = 0
```

图 2-2-3：仿真结果

所有的测试样例均通过.

2.2.6 分配引脚

引脚分配使用 DE10_Standard_SystemBuilder 生成。



Named: * Edit: Filter: Pins: all

Node Name	Direction	Location	I/O Bank	VREF Group	itter Locatior	I/O Standard	Reserved	rrent Streng	Slew Rate	ifferential Pai	Analog Settir	CB/VCCT_GXE	er I/O Pin Ter	icated Refclk	mmon Mode	tter Slew Rate	ifferential O
KEY[3]	Input	PIN_AA15	3B	B3B_N0	PIN_AA15	3.3-V LVTTL		16mA ...ault)									
KEY[2]	Input	PIN_AA14	3B	B3B_N0	PIN_AA14	3.3-V LVTTL		16mA ...ault)									
KEY[1]	Input	PIN_AK4	3B	B3B_N0	PIN_AK4	3.3-V LVTTL		16mA ...ault)									
KEY[0]	Input	PIN_AJ4	3B	B3B_N0	PIN_AJ4	3.3-V LVTTL		16mA ...ault)									
LEDR[9]	Output	PIN_AC22	4A	B4A_N0	PIN_AC22	3.3-V LVTTL		16mA ...ault)	1 (default)								
LEDR[8]	Output	PIN_AB22	5A	B5A_N0	PIN_AB22	3.3-V LVTTL		16mA ...ault)	1 (default)								
LEDR[7]	Output	PIN_AF24	4A	B4A_N0	PIN_AF24	3.3-V LVTTL		16mA ...ault)	1 (default)								
LEDR[6]	Output	PIN_AE24	4A	B4A_N0	PIN_AE24	3.3-V LVTTL		16mA ...ault)	1 (default)								
LEDR[5]	Output	PIN_AF25	4A	B4A_N0	PIN_AF25	3.3-V LVTTL		16mA ...ault)	1 (default)								
LEDR[4]	Output	PIN_AG25	4A	B4A_N0	PIN_AG25	3.3-V LVTTL		16mA ...ault)	1 (default)								
LEDR[3]	Output	PIN_AD24	4A	B4A_N0	PIN_AD24	3.3-V LVTTL		16mA ...ault)	1 (default)								
LEDR[2]	Output	PIN_AC23	4A	B4A_N0	PIN_AC23	3.3-V LVTTL		16mA ...ault)	1 (default)								
LEDR[1]	Output	PIN_AB23	5A	B5A_N0	PIN_AB23	3.3-V LVTTL		16mA ...ault)	1 (default)								
LEDR[0]	Output	PIN_AA24	5A	B5A_N0	PIN_AA24	3.3-V LVTTL		16mA ...ault)	1 (default)								
SW[9]	Input	PIN_AA30	5B	B5B_N0	PIN_AA30	2.5 V		12mA ...ault)									
SW[8]	Input	PIN_AC29	5B	B5B_N0	PIN_AC29	2.5 V		12mA ...ault)									
SW[7]	Input	PIN_AD30	5B	B5B_N0	PIN_AD30	2.5 V		12mA ...ault)									
SW[6]	Input	PIN_AC28	5B	B5B_N0	PIN_AC28	2.5 V		12mA ...ault)									
SW[5]	Input	PIN_V25	5B	B5B_N0	PIN_V25	2.5 V		12mA ...ault)									
SW[4]	Input	PIN_W25	5B	B5B_N0	PIN_W25	2.5 V		12mA ...ault)									
SW[3]	Input	PIN_AC30	5B	B5B_N0	PIN_AC30	2.5 V		12mA ...ault)									
SW[2]	Input	PIN_AB28	5B	B5B_N0	PIN_AB28	2.5 V		12mA ...ault)									
SW[1]	Input	PIN_Y27	5B	B5B_N0	PIN_Y27	2.5 V		12mA ...ault)									
SW[0]	Input	PIN_AB30	5B	B5B_N0	PIN_AB30	2.5 V		12mA ...ault)									

All Pins

图 2-2-3 引脚分配图

2.2.7 全编译

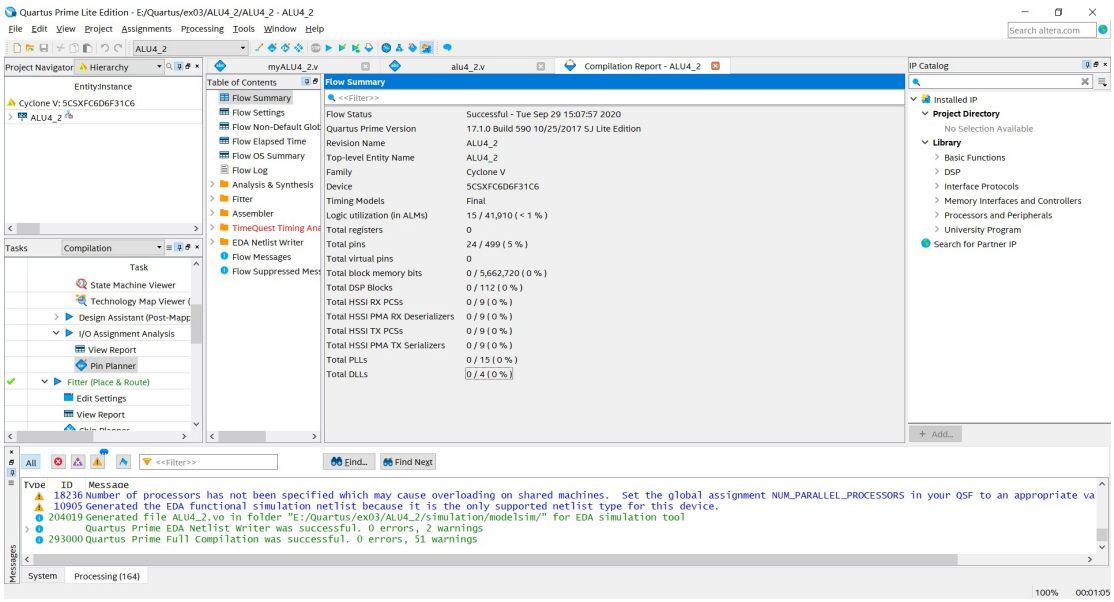


图 2-2-4 全编译成功

三、实验总结

本次实验主要实现了能够对四位补码二进制数进行加法、减法、与、或、非、异或、比较大小、比较相等操作的算数逻辑单元，通过仿真、实操验证了设计正确性。

本次实验的仿真测试部分(2.1.5 节和 2.2.5 节)中实践了 test bench 中的 task 功能. 该功能能提高对于操作数多、长的模型的测试效率，省去了阅读波形图的麻烦，同时也能避免遍历所有可能的值而带来的时间、人力的浪费，而去通过用具有典型意义和关键的样例来使仿真测试更具针对性.

四、附

4.1 实验中遇到的问题及处理办法

问题 1：

ALU 在进行减法运算时 CF 指示位不正确.

原因分析及处理办法：

设计初对 ALU 减法的实现语句如下：

```
1          {CF,  Y} = A - B
```

在仿真测试时发现无法通过对 CF 位检测的样例（样例对应位置：2.1.5 测试样例对应代码 14~18 行）.经过进一步的测试发现上面的语句中的 CF 得到的是减法运算过程中最高位的借位指示，而本实验中要求的 CF 为补码加法的最高位进位信号. 故需要将上述语句中的减

法改为加法（补码相加的方式）.如下所示：

```
reg [3:0]B_com;
B_com = ~B + 1;
{CF, Y} = A + B_com;
```

4.2 关于思考题

4.2.1 思考题 1：

在此加减运算的运算器中，如果用判断参与运算的加数和运算结果符号位是否相同的方法来判断是否溢出，那么此时判断溢出位的时候，应该是比较操作数 A、B 和运算结果的符号位，还是比较 A1、B1 和运算结果的符号位？

应该比较 A1、B1 和运算结果的符号位.是否相同的方法判断是否溢出. 在做加法运算时，A,B 和 A1,B1 为相同的一组数. 在做减法时， B1 为 B 的反码. 加法器内部执行的运算为 A1+B1+Cin, 判断 overflow 的依据和加法相同. 故应比较 A1、B1 和运算结果的符号位.

4.2.2 思考题 2：

方法一：

```
assign t_no_Cin = {n{ Cin } }^B ;
assign {Carry, Result} = A + t_no_Cin + Cin;
assign Overflow = (A[n-1] == t_no_Cin[n-1]) && (Result [n-1] != A[n-1]);
```

方法二：

```
assign t_add_Cin = {n{ Cin } }^B + Cin;//在这里请注意^运算和 + 运算的顺序
assign {Carry, Result} = A + t_add_Cin;
assign Overflow = (A[n-1] == t_add_Cin[n-1]) && (Result [n-1] != A[n-1]);
```

以上两种方法的产生的运算结果、进位位和溢出位值都是完全是一样的吗？如果不一样，为什么结果会不一样呢？在哪一步产生了差别？哪一个正确？

当 Cin = 0, 即进行加法操作时，{n{ Cin } }^B = B, 故方法一、方法各个语句可化为如下形式：

方法一：

```
assign t_no_Cin = B;
assign {Carry, Result} = A + B;
assign Overflow = (A[n-1] == B[n-1]) && (Result [n-1] != A[n-1]);
```

方法二：

```
assign t_add_Cin = B;
assign {Carry, Result} = A + B;
assign Overflow = (A[n-1] == B[n-1]) && (Result [n-1] != A[n-1]);
```

当 Cin = 1，即进行减法操作时，{n{ Cin } }^B = ~B，故在做减法时方法一、方法各个语句可化为如下形式：

方法一：

```
assign t_no_Cin = ~B;
assign {Carry, Result} = A + ~B + 1 = A - B;
assign Overflow = (A[n-1] == ~B[n-1]) && (Result [n-1] != A[n-1]);
```

方法二：

```
assign t_add_Cin = ~B + 1 = -B;
assign {Carry, Result} = A - B;
assign Overflow = (A[n-1] == (-B)[n-1]) && (Result [n-1] != A[n-1]);
```

*：上面的-B 为了表述方便作为 B 的负数的补码的表示，实际上这么写是不对的.

方法二正确。方法一的 Overflow 检测是不正确的，且方法 1 的 Carry 位检测也有问题，比如样例 0-0, A+~B+1 即是 0000+1111+0001,该式子得到的 Carry 位为 1,实际上的 Carry 位应该为 0. 方法二的 Overflow 检测其实也有情况会出问题，例如在计算(-1)-(-8)时 Overflow 会得到 1，实际结果为 0，但这也是补码表示系统中-8 没有对应的正数+8 所致.

4.2.3 思考题 3：

在判断输出结果是否为零的时候也有两种判断方式，一种是用 if 语句，将 Result 和“0”相比较，这样在硬件上会产生一个比较器。还可以使用如下语句：

```
assign zero = ~(| Result);
```

“| Result”操作称为一元约简运算，这个运算在硬件上几个逻辑门就可以实现了，请查阅 Verilog 相关语法资料，了解此运算的操作过程。选择你认为好的方式来进行结果是否为“0”的判断。

查阅相关语法资料发现： 一元约简运算符是单目运算符，其运算规则类似于位运算符中的与、或、非，但其运算过程不同。约简运算符对单个操作数进行运算，最后返回一位数，其运算过程为：首先将操作数 的第一位和第二位进行与、或、非运算；然后再将运算结果和第三位进行与、或、非运算； 依次类推直至最后一位。

题中给出的语句中的~(| Result)即可根据上面的描述转换成如下的等价表达式：

```
assign zero = ~(Result[0]|Result[1]|Result[2]||Result[3]);
                                        //对于操作数为四位的加法器
```

不难发现该表达式和用 if 语句将 Result 和 0 比较是逻辑等价的，而一元约简运算在硬件上只需要几个逻辑门。相比于 if 语句需要使用比较器而言，一元约简运算的实现的代价更小，运行效率更高。故认为采用一元约简效率更高。