《数字逻辑电路》实验报告

第 3 次实验：加法器和 ALU 设计

姓名：凌嘉伟

学号：151220061

15 级 计算机科学与技术系 5 班

邮箱：151220061@smail.nju.edu.cn

时间：2016.9.26

一、 实验目的

知识：

加法器：加法是数字系统中最常执行的运算，加法器是 ALU 的核心部件。减法可以看作是被减数与负的减数的补码相加，因此，可以用加法器同时实现加法和减法两种运算。乘法也可以利用移位相加的算法来实现，因此加法器可以说是计算机中最“繁忙”的部件了。

运算器：运算器是 CPU 设计中的重要部件，运算器包含算术运算和逻辑运算，可以进行乘除、加减和逻辑运算，逻辑运算有逻辑运算电路，乘除有专门的乘除法器，加减运算也有相应的加减运算器，另外还有专门针对于浮点数运算的浮点数运算器等。

方法：

在 Verilog 语言中，可以使用算术赋值语句和向量来执行这种算术运算。

定义：input [n-1:0] X,Y; output [n-1:0] S; 则算术赋值语句 S = X+Y； 就可以实现 n 位加法器。算术溢出信号可以用下面的表达式得到： Overflow = (xn-1 == yn-1) && (sn-1 != xn-1)

二、实验原理（背景知识）

加减运算器只完成加减运算，对于减法运算，如 a-b，可以先求出（–b）的值，然后再和 a 相加，即 a – b = a +（–b）。因此对于加减法运算器，其核心是一个加法器，当这个运算器需要做减法时，用加法器来实现减法。对于加减运算器处理的二进制数值，此数值可以被看成是有符号数（signed），也可以被看成是无符号数（unsigned），无论是有符号数还是无符号数，对于运算器来说都是二进制的码，处理的方式都是一样的，不同的是对于运算结果的处理不一样。

进位位，在二进制加法运算中是指在运算过程中，参与运算的两个数的最高位运算时向更高位产生的进位。在做减法运算时，是指参与运算的两个数的最高位运算时向更高位产生的借位。

二进制数的补码表示可以实现加减运算的统一，因此在机器中，有符号数一般都是采用补码的形式表示，对于 n 位字长的机器，补码可表示的有符号数的数值范围是– 2n-1 ~ 2n–1， 如果进行加减运算的运算结果超出了这个范围时，运算结果溢出。判断一个运算是否溢出常用的有几种方法：

1 在运算中，如果最高数值位向符号位产生的进位和和符号位向前产生进位一样（同时都有进位或者同时都没有进位），则运算结果不溢出（overflow = 0），如果这两个进位不一样，则运算结果溢出（overflow = 1）。

2 采用变形补码（即双符号位的表示方式，00 表示正数，11 表示负数）进行运算，运算是两个符号位都参加运算，如果运算结果的符号位是 00（正数）或者 11（负数），则运算 结果正确（不溢出，overflow = 0），如果运算结果符号位为 01 或者 10，则表示运算超出了 n 位二进制补码表示的范围，结果溢出了（overflow = 1）。

3 如果两个加数的符号位相同，且与和的符号位相同，则结果正确（overflow = 0），如果两个加数的符号位相同，而与结果的符号位不同，则结果溢出（overflow = 1）。如果两个加数的符号位不同（一正一负相加），那么结果肯定不溢出。

对于无符号数，如果加法运算的最高位有进位，表示此运算超出了机器字长的表示范围，就是溢出了。对于减法运算，如果运算的最高位有借位，表示被减数小于减数了，运算结果应该为负数，而无符号数不能表示负数，那么此运算也溢出了。因此，对于无符号数的运算，从进位/借位就可以看成运算结果是否溢出，不需要用溢出位来判断了。

对于加法运算： 无论是采用无符号加法器、有符号加法器、有符号加/减法器，还是无符号加/减法器，做加法时，加法器只是将输入的两个数字看成二进制的码来进行加法运算，运算的结果都是一样的，运算结果由用户来判断，如果是无符号数运算，运算结果也是无符号数；如果是有符号数运算，则运算结果是补码形式。对于进位位，是最高位的数字向更高位的进位。对于溢出位，当运算是有符号数时，溢出位是看符号位和数值的最高位是否同时有进位，如果同时有进位，或者同时没有进位，则没有溢出，Overflow = 0 ，如果不是同时有进位，则有溢出，Overflow = 1。当运算是无符号数时，溢出位是和进位位一样的。

无论是用无符号加法器还是有符号加法器，做减法时，先将减数连同符号位在内按位取反，尾数加“1”，再和被减数相加。对于减的结果是有符号数还是无符号数由用户来判断，如果是无符号数运算，运算结果也是无符号数；如果是有符号数运算，则运算结果是补码形式。对于进位位，这时加法器做的加法是：a + b 按位取反的值 + 1，进位位也是这三个变量相加结果的进位，进位采用的是负逻辑，如果运算的最高位向更高位有进位 Carry = 1，表示没有借位，如果 Carry = 0，则表示有借位。对于溢出位，有符号减法运算同有符号加法运算相同判断，只是这时的加法有三个变量：a + b 按位取反的值 + 1 ，Overflow = 0 没有溢出，Overflow = 1 有溢出。对于无符号减法运算，溢出位是进位位的取反，如果 Carry = 1，则 Overflow = 0，表示没有溢出；如果 Carry = 0，则 Overflow = 1，表示有溢出。

三、实验器材/环境

软件环境：Win7 下 Vivado 2015.2

四、实验设计思路（验收实验）

实验设计原文件：

```
module ALU #(parameter WIDTH = 8) (
    input Cin,
    input [WIDTH - 1:0] A,
    input [WIDTH - 1:0] B,
    output Carry,
    output [WIDTH - 1:0] Result,
    output Zero,
    output Overflow
    );
    wire [WIDTH - 1:0] temp;
    assign temp = {WIDTH{Cin}} ^ B;
    Adder #(WIDTH) adder
    (.Cin(Cin),.A(A),.B(temp),.Carry(Carry),.Result(Result),.Zero(Zero),.Overflow(Overflow));

endmodule

module Adder #(parameter WIDTH = 8) (
    input Cin,
    input [WIDTH - 1:0] A,
```

```
    input [WIDTH - 1:0] B,
    output Carry,
    output [WIDTH - 1:0] Result,
    output Zero,
    output Overflow
    );

    assign {Carry, Result} = A + B + Cin;
    assign Zero = Result == 0 ? 1 : 0;
    assign Overflow = (A[WIDTH - 1] == B[WIDTH - 1]) && (Result[WIDTH - 1] != A[WIDTH - 1]);

endmodule
```

设计测试文件：

```
`define WIDTH 8

module ALU_tb(

    );

    integer N;
    reg Cin;
    reg [`WIDTH - 1:0] A;
    reg [`WIDTH - 1:0] B;

    wire Carry;
    wire [`WIDTH - 1:0] Result;
    wire Zero;
    wire Overflow;

    ALU    #(`WIDTH)    alu
    (.Cin(Cin), .A(A), .B(B), .Carry(Carry), .Result(Result), .Zero(Zero), .Overflow(Overflow));

    initial begin
        Cin = 0;
        A = 0;
        B = 0;
    end

    initial begin
        for (N = 0; N < 131071; N = N + 1)
            #10 {Cin, A, B} = {Cin, A, B} + 1;
```

```
                #1
                if (Cin == 0) begin
                    if (Result == A + B) begin
                        $display("PASS");
                    end
                    else begin
                        $display("FAIL [Result = %b, Cin = %b, A = %b, B = %b]s",
Result, Cin, A, B);
                        $finish;
                    end
                end
                else begin
                    if (Result == A - B) begin
                        $display("PASS");
                    end
                    else begin
                        $display("FAIL [Result = %b, Cin = %b, A = %b, B = %b]", Result,
Cin, A, B);
                        $finish;
                    end
                end
            #10 $finish;
        end

endmodule
```

引脚配置文件：

```
set_property -dict { PACKAGE_PIN J15      IOSTANDARD LVCMOS33 } [get_ports { A[0] }];
#IO_L24N_T3_RS0_15 Sch=sw[0]
set_property  -dict  {  PACKAGE_PIN  L16      IOSTANDARD  LVCMOS33  }  [get_ports
{ A[1] }]; #IO_L3N_T0_DQS_EMCCLK_14 Sch=sw[1]
set_property  -dict  {  PACKAGE_PIN  M13      IOSTANDARD  LVCMOS33  }  [get_ports
{ A[2] }]; #IO_L6N_T0_D08_VREF_14 Sch=sw[2]
set_property  -dict  {  PACKAGE_PIN  R15      IOSTANDARD  LVCMOS33  }  [get_ports
{ A[3] }]; #IO_L13N_T2_MRCC_14 Sch=sw[3]
set_property  -dict  {  PACKAGE_PIN  R17      IOSTANDARD  LVCMOS33  }  [get_ports
{ A[4] }]; #IO_L12N_T1_MRCC_14 Sch=sw[4]
set_property  -dict  {  PACKAGE_PIN  T18      IOSTANDARD  LVCMOS33  }  [get_ports
{ A[5] }]; #IO_L7N_T1_D10_14 Sch=sw[5]
set_property  -dict  {  PACKAGE_PIN  U18      IOSTANDARD  LVCMOS33  }  [get_ports
{ A[6] }]; #IO_L17N_T2_A13_D29_14 Sch=sw[6]
set_property  -dict  {  PACKAGE_PIN  R13      IOSTANDARD  LVCMOS33  }  [get_ports
{ A[7] }]; #IO_L5N_T0_D07_14 Sch=sw[7]
set_property  -dict  {  PACKAGE_PIN  T8       IOSTANDARD  LVCMOS18  }  [get_ports
```

```
{ B[0] }]; #IO_L24N_T3_34 Sch=sw[8]
set_property -dict { PACKAGE_PIN U8 IOSTANDARD LVCMOS18 } [get_ports
{ B[1] }]; #IO_25_34 Sch=sw[9]
set_property -dict { PACKAGE_PIN R16 IOSTANDARD LVCMOS33 } [get_ports
{ B[2] }]; #IO_L15P_T2_DQS_RDWR_B_14 Sch=sw[10]
set_property -dict { PACKAGE_PIN T13 IOSTANDARD LVCMOS33 } [get_ports
{ B[3] }]; #IO_L23P_T3_A03_D19_14 Sch=sw[11]
set_property -dict { PACKAGE_PIN H6 IOSTANDARD LVCMOS33 } [get_ports
{ B[4] }]; #IO_L24P_T3_35 Sch=sw[12]
set_property -dict { PACKAGE_PIN U12 IOSTANDARD LVCMOS33 } [get_ports
{ B[5] }]; #IO_L20P_T3_A08_D24_14 Sch=sw[13]
set_property -dict { PACKAGE_PIN U11 IOSTANDARD LVCMOS33 } [get_ports
{ B[6] }]; #IO_L19N_T3_A09_D25_VREF_14 Sch=sw[14]
set_property -dict { PACKAGE_PIN V10 IOSTANDARD LVCMOS33 } [get_ports
{ B[7] }]; #IO_L21P_T3_DQS_14 Sch=sw[15]
set_property -dict { PACKAGE_PIN P18 IOSTANDARD LVCMOS33 } [get_ports { Cin }];
#IO_L9N_T1_DQS_D13_14 Sch=btnd

set_property -dict { PACKAGE_PIN H17 IOSTANDARD LVCMOS33 } [get_ports
{ Carry }]; #IO_L18P_T2_A24_15 Sch=led[0]
set_property -dict { PACKAGE_PIN K15 IOSTANDARD LVCMOS33 } [get_ports
{ Result[0] }]; #IO_L24P_T3_RS1_15 Sch=led[1]
set_property -dict { PACKAGE_PIN J13 IOSTANDARD LVCMOS33 } [get_ports
{ Result[1] }]; #IO_L17N_T2_A25_15 Sch=led[2]
set_property -dict { PACKAGE_PIN N14 IOSTANDARD LVCMOS33 } [get_ports
{ Result[2] }]; #IO_L8P_T1_D11_14 Sch=led[3]
set_property -dict { PACKAGE_PIN R18 IOSTANDARD LVCMOS33 } [get_ports
{ Result[3] }]; #IO_L7P_T1_D09_14 Sch=led[4]
set_property -dict { PACKAGE_PIN V17 IOSTANDARD LVCMOS33 } [get_ports
{ Result[4] }]; #IO_L18N_T2_A11_D27_14 Sch=led[5]
set_property -dict { PACKAGE_PIN U17 IOSTANDARD LVCMOS33 } [get_ports
{ Result[5] }]; #IO_L17P_T2_A14_D30_14 Sch=led[6]
set_property -dict { PACKAGE_PIN U16 IOSTANDARD LVCMOS33 } [get_ports
{ Result[6] }]; #IO_L18P_T2_A12_D28_14 Sch=led[7]
set_property -dict { PACKAGE_PIN V16 IOSTANDARD LVCMOS33 } [get_ports
{ Result[7] }]; #IO_L16N_T2_A15_D31_14 Sch=led[8]
set_property -dict { PACKAGE_PIN T15 IOSTANDARD LVCMOS33 } [get_ports
{ Zero }]; #IO_L14N_T2_SRCC_14 Sch=led[9]
set_property -dict { PACKAGE_PIN U14 IOSTANDARD LVCMOS33 } [get_ports
{ Overflow }]; #IO_L22P_T3_A05_D21_14 Sch=led[10]
```

五、实验的测试序列或验证方法

实验测试序列：例如-1-(-128)，-1 的补码为 11111111，此时应将对应 A 的 8 个开关全部打

开，-128 需要将对应 B 的最高位的开关打开，按下 Cin 对应的按钮 P18，可以发现 Carry 进位位变亮，而溢出位不亮，表示结果的 LED 灯全亮，表示 127 不溢出。
2-3，2 的二进制表示为 00000010,3 的二进制表示为 00000011，在对应 A 和 B 的开关打开设置相应的值后，按下 Cin 对应的按钮，可以看到结果。

六、实验过程（主要指验收实验）

先使用 vivado，编写设计原文件：

```
module ALU #(parameter WIDTH = 8) (
    input Cin,
    input [WIDTH - 1:0] A,
    input [WIDTH - 1:0] B,
    output Carry,
    output [WIDTH - 1:0] Result,
    output Zero,
    output Overflow
    );

    wire [WIDTH - 1:0] temp;

    assign temp = {WIDTH{Cin}} ^ B;
    Adder #(WIDTH) adder (.Cin(Cin), .A(A), .B(temp), .Carry(Carry), .Result(Result), .Zero(Zero), .Overflow(Overflow));

endmodule

module Adder #(parameter WIDTH = 8) (
    input Cin,
    input [WIDTH - 1:0] A,
    input [WIDTH - 1:0] B,
    output Carry,
    output [WIDTH - 1:0] Result,
    output Zero,
    output Overflow
    );

    assign {Carry, Result} = A + B + Cin;
    assign Zero = Result == 0 ? 1 : 0;
    assign Overflow = (A[WIDTH - 1] == B[WIDTH - 1]) && (Result[WIDTH - 1] != A[WIDTH - 1]);

endmodule
```

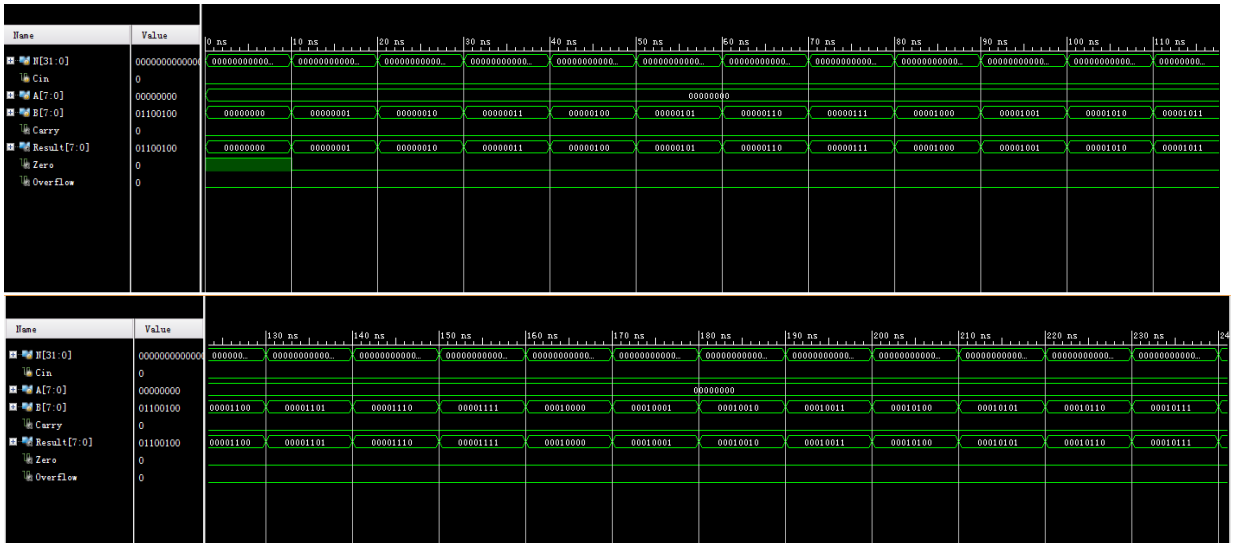
再编写测试文件：

ALU.v | ALU_tb.v | add.xdc

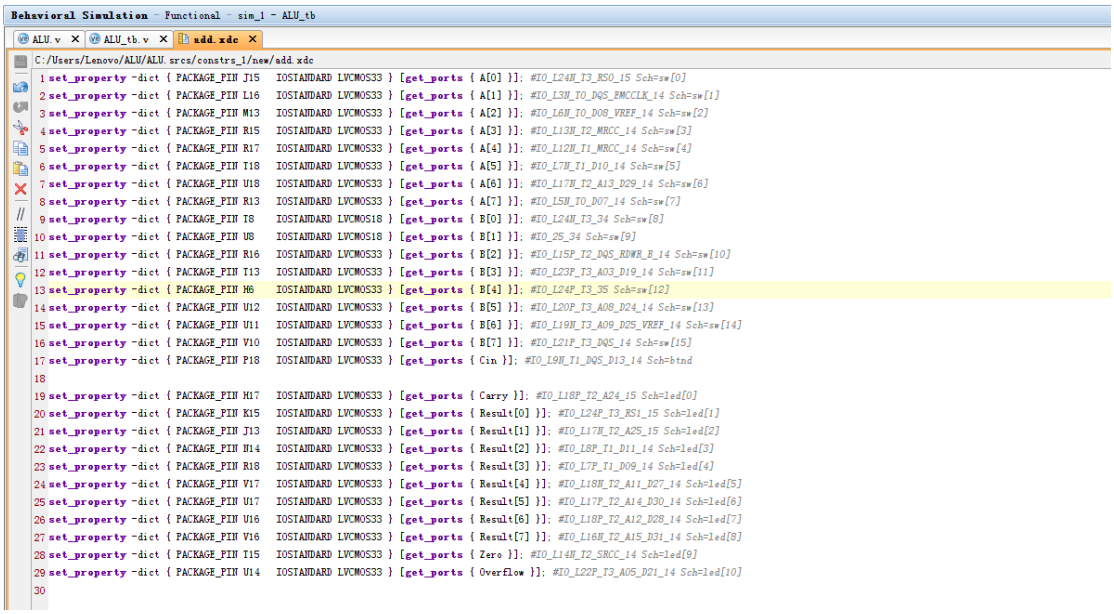
C:/Users/Lenovo/Downloads/ALU_tb.v

```
28      integer N;
29      reg Cin;
30      reg [`WIDTH - 1:0] A;
31      reg [`WIDTH - 1:0] B;
32
33      wire Carry;
34      wire [`WIDTH - 1:0] Result;
35      wire Zero;
36      wire Overflow;
37
38      ALU #(`WIDTH) alu (.Cin(Cin), .A(A), .B(B), .Carry(Carry), .Result(Result), .Zero(Zero), .Overflow(Overflow));
39
40      initial begin
41          //$monitor ("Time = %d, Cin = %b, A = %b, B = %b, Carry = %b, Result = %b, Zero = %b, Overflow = %b", $time, Cin, A, B, Carry, Result, Zero, Overflow);
42          Cin = 0;
43          A = 0;
44          B = 0;
45      end
46
47      initial begin
48          for (N = 0; N < 131071; N = N + 1)
49              #10 {Cin, A, B} = {Cin, A, B} + 1;
50              #1
51              if (Cin == 0) begin
52                  if (Result == A + B) begin
53                      $display("PASS");
54                  end
55                  else begin
56                      $display("FAIL [Result = %b, Cin = %b, A = %b, B = %b]\n", Result, Cin, A, B);
57                      $finish;
58                  end
59              end
60              else begin
61                  if (Result == A - B) begin
62                      $display("PASS");
63                  end
64                  else begin
65                      $display("FAIL [Result = %b, Cin = %b, A = %b, B = %b]", Result, Cin, A, B);
66                      $finish;
67                  end
68              end
69          #10 $finish;
70      end
```

点击项目导航栏 Simulation 下的“Run Simulation”，选择 Run Behavioral Simulation，即可看到仿真画面：



再编写引脚配置文件：

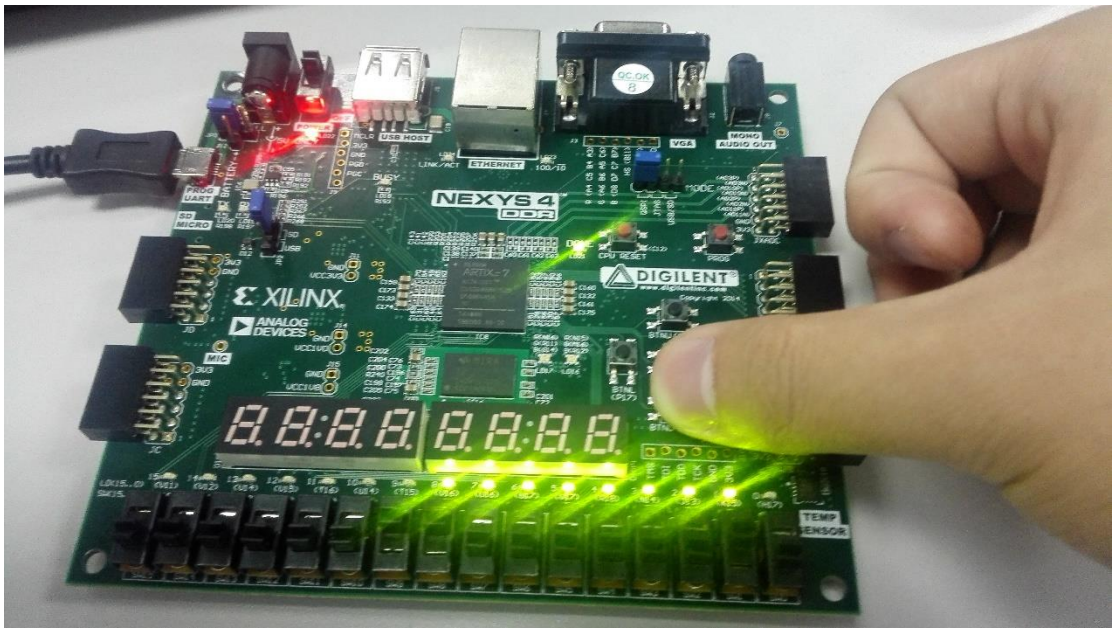


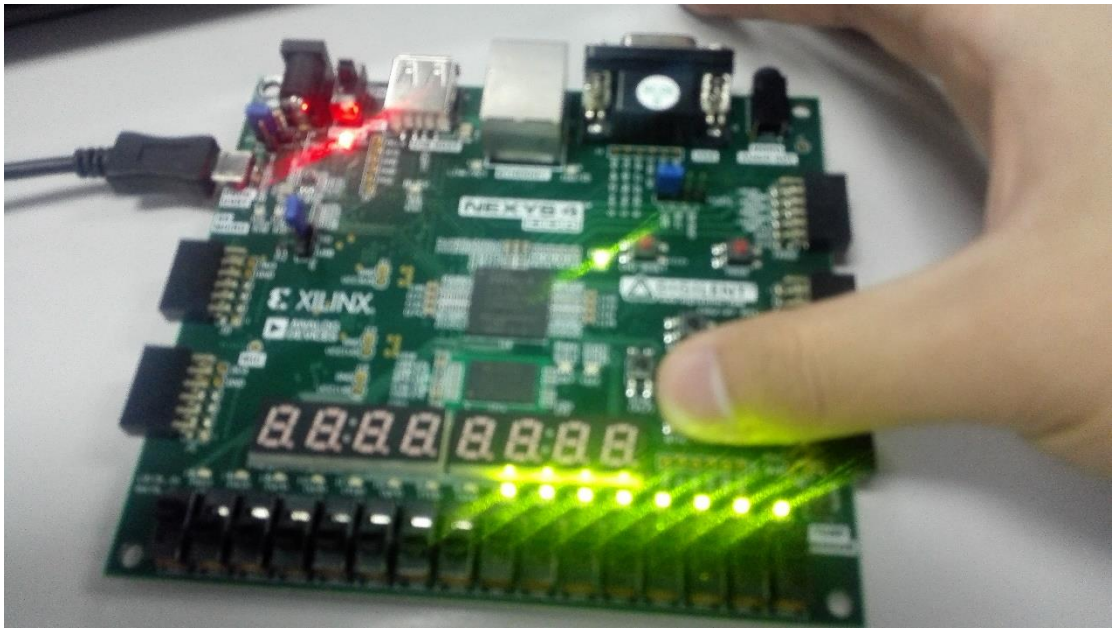
将开发平台连接到电脑上，在 Flow Navigator 中点击 Program and Debug 下的 Generate Bitstream 选项，工程会自动完成综合、实现、Bit 文件生成过程，完成之后，可点击 Open Implemented Design 来查看工程实现结果。

成功连接后，右击 FPGA 芯片，选择 Program Device。 此时，Hardware 一栏中出现硬件平台上可编程的器件 XC7A100T_0。在对应的 FPGA 器件上右击，选择 Assign Programming File，指定所需的 bit 文件。

硬件连接完成。

下面分别是 2-3 的结果和(-1)-(-128)的结果：





七、实验结果、结论等

结果：完成了一个可以实现加减功能的 ALU，并在硬件上得到验证。

八、实验中遇到的问题及解决方案

问题：无

九、实验的启示/意见和建议

1、 本次实验难度较大，实验安排较为合理。
2、 实现更多功能的 ALU

十、 实验时间

1 预习时间约 1 小时。
2 实验课后在本次实验上又花的时间约 1 小时。

十一、实验思考题

A． 如果给 ALU 添加逻辑运算等其他功能该如何添加？

代码如下：

```
module ALU(S,A,B,CIN,F,G_L,P_L);
input [2:0] S;
input [7:0] A,B;
input CIN;
output [7:0] F;
output G_L,P_L;
```

```
reg [7:0] F;
reg G_L,P_L,GG,GP;
reg [7:0] G,P;
integer I;
always @ (S or A or B or CIN or G or P or GG or GP) begin
    for (I = 0;I <= 7;I = I + 1) begin
        G[i] = A[i] & B[i];
        P[i] = A[i] | B[i];
    end
    GG = G[0];GP = P[0];
    for (I = 1;I <= 7;I = I + 1) begin
        GG = G[i] | (GG & P[i]);
        GP = P[i] & GP;
    end
    G_L = ~GG;P_L = ~GP;
    case (S)
        3'd0:F = 8'b0;
        3'd1:F = B – A – 1 + CIN;
        3'd2:F = A – B – 1 + CIN;
        3'd3:F = A + B + CIN;
        3'd4:F = A ^ B;
        3'd5:F = A | B;
        3'd6:F = A & B;
        3'd7:F = 8'b11111111;
        default:F = 8'b0;
    endcase
end
end module
```

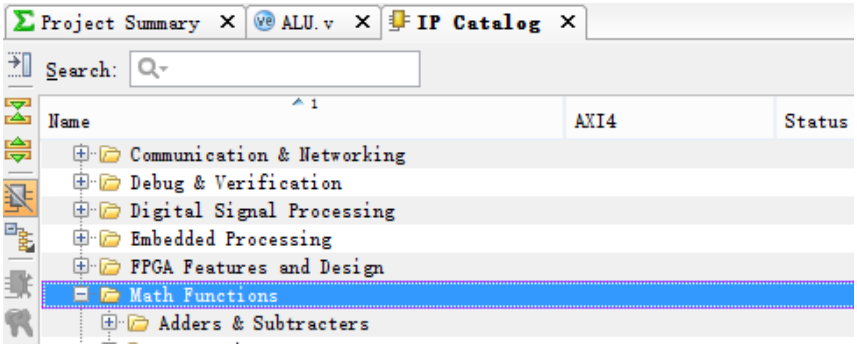
代码注解：本段代码第一个 for 循环用于为每个加法器段（i 的范围是 0-7）构造内部进位信号 G[i]及其传递信号 P[i]，第二个 for 循环把这些信号组合成 8 位的组进位信号 G_L 及其传递信号 P_L。在第 i 次迭代中，变量 GG 表明 ALU 是否会产生一个加法器的第 i 段的进位，与此类似，在第 i 次迭代中，GP 表明 ALU 是否会传递加法器的第 i 段进位。GP 最后的值是所有 i 所对应 P[i]信号的与，而且输出信号 P_L 是这个值的补。case 语句用于从 8 个输出函数 F 中选择一个。

B． 参考“利用 ip 完成工程”，思考如何用 Vivado 提供的现成的加法器 ip 来实现一个加法器。

直接调用 ip 核：

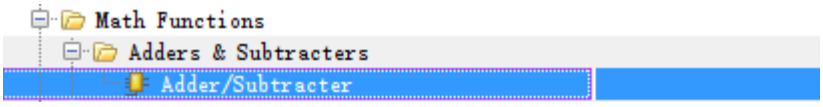
1、 点击 IP Catalog

2、 点击 Math Functions

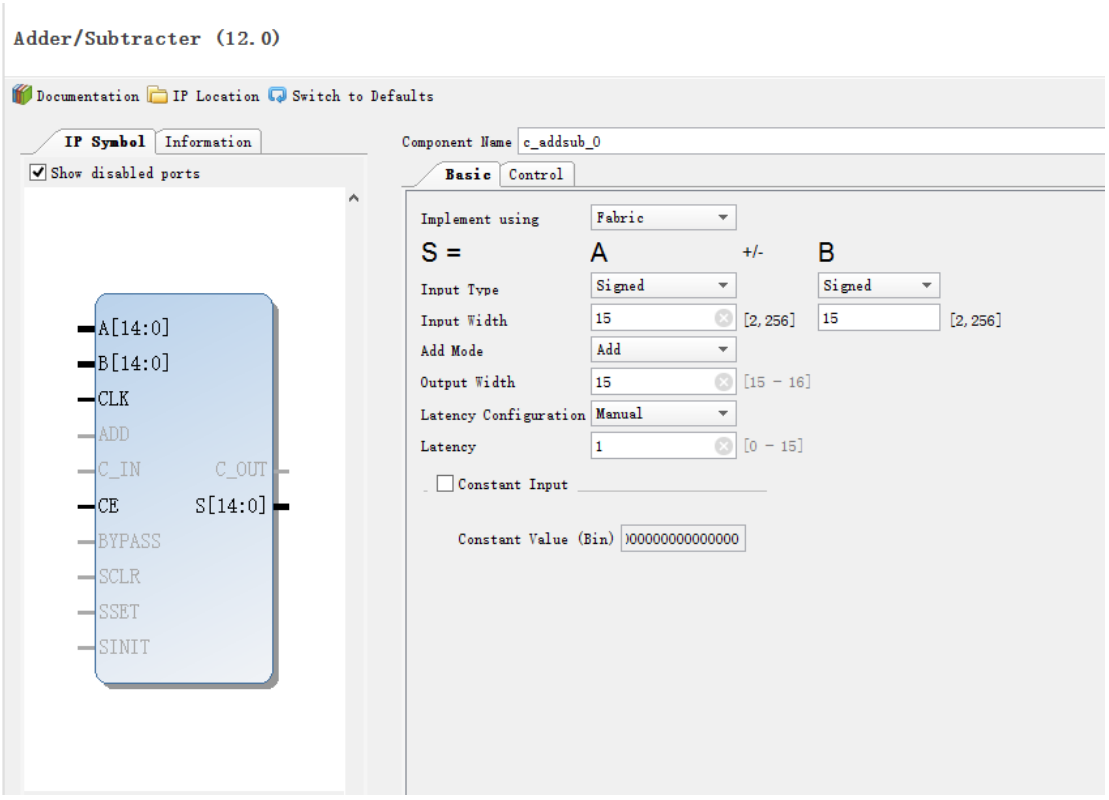


看到有个 Adders&Subtracters，即加法&减法器

3、 点击后会出现如下情形，继续点击 Adder/subtracter



4、 发现一个加法器模型



5、根据自己的需要，配置加法器，注意图中的器件名称、引脚名称及宽度

6、在 Verilog 程序中直接实例化该器件并调用

C. 请参考“简单加减运算器的设计及标志位的判断.pdf”，结合自己的电路，总结溢出位和进位位的判断。

溢出位：

1、 在运算中，如果最高数值位向符号位产生的进位和和符号位向前产生进位一样（同时都有进位或者同时都没有进位），则运算结果不溢出（overflow = 0），如果这两个进位不一样，则运算结果溢出（overflow = 1）。

2、 采用变形补码（即双符号位的表示方式，00 表示正数，11 表示负数）进行运算，运 算是两个符号位都参加运算，如果运算结果的符号位是 00（正数）或者 11（负数），则运算 结果正确（不溢出，overflow = 0），如果运算结果符号位为 01 或者 10，则表示运算超出了 n 位二进制补码表示的范围，结果溢出了（overflow = 1）。

3、 如果两个加数的符号位相同，且与和的符号位相同，则结果正确（overflow = 0），如果两个加数的符号位相同，而与结果的符号位不同，则结果溢出（overflow = 1）。如果两个加数的符号位不同（一正一负相加），那么结果肯定不溢出。

进位位：

计算机处理的数字都是二进制数，进行的运算是以 2^n 为模的运算，其中 n 是计算机的字长。所以，–b = 2^n – b 那么，a – b = a +（–b）= a +（2^n – b），对于 a +（2^n – b）这个运算，使运算结果产生进位位的条件是：a +（2^n – b）> 2^n – 1，

因为 a 和 b 都是二进制整数，因此，进行 a +(2^n – b)这个运算，产生进位位的条件是：a > = b。即，当 a > = b 时进位位 carry = 1，当 a < b 时，进位位 carry = 0。

而对于我们真正要进行的运算：a – b，产生借位的条件是：a < b。即，当 a > = b 时应该有进位位 carry = 0，当 a < b 时，应该有进位位 carry = 1。这一条件与进行 a +（2^n – b）这个运算真正产生的进位位 carry 的值正好相反。

因此，在利用加法器进行减法运算时，加法器产生的进位位和实际进行减法运算时应该产生的借位是相反的。