中山大学数据科学与计算机学院本科生实验报告

（2019 学年秋季学期）

课程名称：**计算机组成原理实验**　　任课教师：　　助教：

年级&班级	19 计科超算	专业(方向)	
学号	18324034	姓名	林天皓
电话		Email	linth5@mail2.sysu.edu.cn
开始日期	2020.9.18	完成日期	2020.9.25

一、实验题目

实验 3 基于 vivado 设计 8 位的加减法器

二、实验目的

1. 进一步学习 Verilog HDL 的设计方法；
2. 通过设计加减法器，加深理解补码表示加减电路一体化设计原理。

三、实验内容

1. 实验原理

第一部分. 实现8位带进位，溢出、判零功能的加减法运算器

首先需要实现二进制中负数的补码表示，即为原码取反加一。然后处理加法计算中的进位和减法运算中的借位，为两个正数相加产生的上进位，小正数减去大正数产生的借位。接下来处理溢出位，分别为两个正数相加产生上溢出和负数减负数产生的下溢出最后还需要判断零和输出运算结果。

设计完成后，通过模拟功能测试几组实验样例来保证所设计的模块没有错误，分析波形图描述运算过程，最后综合，绑定引脚，在实验板上下载运行测试。

第二部分：实现8位超前进位加法器

在该项实验中，与上一项中不同的是不再使用verilog自带的+运算符进行整个二进制数的

计算，而是自己逐位地处理进位信号和运算结果并输出结果。与第一部分一样，设计完成后要进行充分的模拟测试，分析运算过程，按照流程最后在实验板上下载运行测试。

3. 实验步骤

第一部分. 8位带进位，溢出、判零功能的加减法运算器

对于操作数a,b，首先处理减法过程中补码的表示方法。根据补码规则，补码为原码求反加一，在具体实现中，采取与1111111进行异或运算的方式来取反，接下来+1，完成补码的运算工作。Verilog代码如下

```
    assign subb = (b ^ {WIDTH{sub}}) + sub; // 对于减法是取反 对于减法是
加 1，sub=1（减法）sub=0（加法）// 减法：subb = b 取反+1 ；  加法：subb = b
```

若sub为1（即该操作为减法）的情况下，subb为b的反码，否则subb等于b。

在这其中{WIDTH{sub}}这个语法代表由WIDTH个sub拼接组成的数字。

接下来处理进位借位判断，在加法情况下，只需判断运算后的第9位为1，则发生了进位减法情况下，若一个小整数减去大整数，只需判断运算后的第9位为0，则结果小于零，需要借位。

Verilog代码如下

```
    assign {cf2,sum} = a + subb;
    assign cf = cf2 ^ sub;  //进位 cf = cf2 ^ sub
```

其中，cf2为a+subb的第九位，sum为a+subb的第一到八位。

判零功能较为简单，Verilog代码如下

```
    assign zf = (sum == 0) ? 1 : 0 ; //sum=0,  则 0 标 志 zf=1
```

再行溢出判断，当两个操作数符号相同，且运算结果的符号与这两个操作数符号相反时，发生溢出，Verilog代码如下

```
    assign ovf = (a[WIDTH-1] ^ sum[WIDTH-1]) & (subb[WIDTH-1] ^ sum[WIDTH-
1]
```

最后的结果储存在sum中

整个8位二进制加减法器Verilog代码如下

```
module addsub
    #(parameter WIDTH=8)    //指定数据宽度参数，缺省值是 8 (
```

```
   (
    input [(WIDTH-1):0] a,  // 缺省位数由参数 WIDTH 决定， a 为第一位运算数
    input [(WIDTH-1):0] b,  // b 为第二位运算数
    input sub,  // sub=1 为减法，sub=0 为加法
    output [(WIDTH-1):0] sum, // sum 为 a 与 b 运算的结果
    output cf,  // cf 为进位标志
    output ovf, // ovf 为溢出标志
    output sf,  // sf 为符号标志
    output zf   // zf 为 判断为 0 标志
    );
    wire [(WIDTH-1):0] subb,subb1;
    wire cf2; // 进 位
    assign subb = (b ^ {WIDTH{sub}}) + sub; // 对于减法是取反 对于减法是加 1，sub=1
（减法）sub=0（加法）// 减法：subb = b 取反+1 ； 加法：subb = b
    assign {cf2,sum} = a + subb;
    assign sf = sum[WIDTH-1];       //sum 的最高位为符号位
    assign zf = (sum == 0) ? 1 : 0 ; //sum=0， 则 0 标 志 zf=1
    assign cf = cf2 ^ sub;  //进位 cf = cf2 ^ sub
    assign ovf = (a[WIDTH-1] ^ sum[WIDTH-1]) & (subb[WIDTH-1] ^ sum[WIDTH-1] );//
判断溢出
endmodule
```

下面进行8位加减法器的仿真测试流程

仿真Verilog代码如下

```
module addsub_test();
    reg [7:0] a,b;
    wire [7:0] sum;
    reg sub;
    wire cf,ovf,sf,zf;
    reg clk;
    addsub u0(
        .a(a),
        .b(b),
        .sub(sub),
        .sum(sum),
        .cf(cf),
        .ovf(ovf),
        .sf(sf),
        .zf(zf)
        );
    initial begin
        $dumpfile("testaddsub.vcd");
        $dumpvars;
```

```
        #200 sub = 1;
        #200 a = 8'h7f; b = 8'h2; sub = 0;
        $monitor ("%0t a=%8b b=%8b sub=%1b sum=%8b cf=%1b ovf=%1b sf=%1b zf=%1b",
$time, a, b, sub,sum,cf,ovf,sf,zf);
        #200 a = 8'hff; b = 8'h2; sub = 0;
        $monitor ("%0t a=%8b b=%8b sub=%1b sum=%8b cf=%1b ovf=%1b sf=%1b zf=%1b",
$time, a, b, sub,sum,cf,ovf,sf,zf);
        #200 a = 8'h16; b = 8'h17; sub = 1;
        $monitor ("%0t a=%8b b=%8b sub=%1b sum=%8b cf=%1b ovf=%1b sf=%1b zf=%1b",
$time, a, b, sub,sum,cf,ovf,sf,zf);
        #200 a = 8'hfe; b = 8'hff; sub = 1;
        $monitor ("%0t a=%8b b=%8b sub=%1b sum=%8b cf=%1b ovf=%1b sf=%1b zf=%1b",
$time, a, b, sub,sum,cf,ovf,sf,zf);
    end

endmodule
```

这里与老师所给的仿真文件中的测试值是相同的，只是更多的进行了$dumpfile操作和$monitor分别代表生成VCD文件以用来提供给其他的波形展示软件（如GTKwave）进行展示。这是由于vivado进行一次仿真的速度较慢，需要等待数十秒，所以我自己配置了iVerilog编译环境和GTKwave波形显示器进行仿真测试波形，vvp执行编译后的程序，再打开vcd波形文件，基本上是秒出结果，在后文实验结果中介绍具体的实验结果。

接下来按照引脚分配创建约束文件下载程序到实验板上同样用这几组数据测试。

第一部分实现8位带进位，溢出、判零功能的加减法运算器结束

第二部分：8位超前进位加法器的设计

首先确定设计目标，本次设计的目标是8位超前进位加法器可以在任意以为发生变化的时候，立即得到本次的加法结果，以及本次的加法运算有没有产生向前的进位，为了快速得到结果，我们采用了组合逻辑电路的实现方式。

首先令g表示不考虑后面的进位，这里是否有进位，下面为verilog代码

```
assign
        g[0] = a[0] & b[0],
        g[1] = a[1] & b[1],
        g[2] = a[2] & b[2],
        g[3] = a[3] & b[3],
```

```
        g[4] = a[4] & b[4],
        g[5] = a[5] & b[5],
        g[6] = a[6] & b[6],
        g[7] = a[7] & b[7];
```

该代码可以更简单，为

```
        g = a & b;
```

同理，令p表示a或者b在该位置上至少有一个是1

```
        p = a | b;
```

c_tmp[7]表示加法整体是否有进位，其他c_tmp[i]表示第i+1位置是否有进位。

具体实现如下

```
assign
c_tmp[0] = g[0] | ( p[0] & cin ),
c_tmp[1] = g[1] | ( p[1] & g[0])
                | ( p[1] & p[0] & cin),
c_tmp[2] = g[2] | ( p[2] & g[1])
                | ( p[2] & p[1] & g[0])
                | ( p[2] & p[1] & p[0] & cin),
c_tmp[3] = g[3] | ( p[3] & g[2])
                | ( p[3] & p[2] & g[1])
                | ( p[3] & p[2] & p[1] & g[0])
                | ( p[3] & p[2] & p[1] & p[0] & cin),
c_tmp[4] = g[4] | ( p[4] & g[3])
                | ( p[4] & p[3] & g[2])
                | ( p[4] & p[3] & p[2] & g[1])
                | ( p[4] & p[3] & p[2] & p[1] & g[0])
                | ( p[4] & p[3] & p[2] & p[1] & p[0] &cin),
c_tmp[5] = g[5] | ( p[5] & g[4])
                | ( p[5] & p[4] & g[3])
                | ( p[5] & p[4] & p[3] & g[2])
                | ( p[5] & p[4] & p[3] & p[2] & g[1])
                | ( p[5] & p[4] & p[3] & p[2] & p[1] & g[0])
                | ( p[5] & p[4] & p[3] & p[2] & p[1] & p[0] & cin),
c_tmp[6] = g[6] | ( p[6] & g[5])
                | ( p[6] & p[5] & g[4])
                | ( p[6] & p[5] & p[4] & g[3])
                | ( p[6] & p[5] & p[4] & p[3] & g[2])
                | ( p[6] & p[5] & p[4] & p[3] & p[2] & g[1])
                | ( p[6] & p[5] & p[4] & p[3] & p[2] & p[1] & g[0])
                | ( p[6] & p[5] & p[4] & p[3] & p[2] & p[1] & p[0] & cin),
c_tmp[7] = g[7] | ( p[7] & g[6])
                | ( p[7] & p[6] & g[5])
                | ( p[7] & p[6] & p[5] & g[4] )
```

```
                | ( p[7] & p[6] & p[5] & p[4] & g[3])
                | ( p[7] & p[6] & p[5] & p[4] & p[3] & g[2])
                | ( p[7] & p[6] & p[5] & p[4] & p[3] & p[2] & g[1])
                | ( p[7] & p[6] & p[5] & p[4] & p[3] & p[2] & p[1] & g[0])
                | ( p[7] & p[6]& p[5] & p[4] & p[3] & p[2] & p[1] & p[0] & cin);
```

最后输出加法结果s

```
assign s[7:0] = a[7:0] ^ b[7:0] ^{c_tmp[6:0],cin};
```

上述语句通过异或表示加法，最后表示所有的进位与前置进位的拼接。

```
assign co = c_tmp[7];
```

co表示最终的进位输出。

完整实现的超前进位8位加法器完整verilog模块代码如下：

```
module add_8 ( input [7:0]a, input [7:0]b, input cin, output [7:0] s, output co );

wire [7:0]c_tmp;
wire [7:0]g;
wire [7:0]p;
assign co = c_tmp[7];
assign
        //G 表示不考虑后面的进位，这里是否有进位
        g = a & b;
assign
        //P 表示 a 或 b 在这一位上至少有一个
        p = a | b;
assign
c_tmp[0] = g[0] | ( p[0] & cin ),
c_tmp[1] = g[1] | ( p[1] & g[0])
                | ( p[1] & p[0] & cin),
c_tmp[2] = g[2] | ( p[2] & g[1])
                | ( p[2] & p[1] & g[0])
                | ( p[2] & p[1] & p[0] & cin),
c_tmp[3] = g[3] | ( p[3] & g[2])
                | ( p[3] & p[2] & g[1])
                | ( p[3] & p[2] & p[1] & g[0])
                | ( p[3] & p[2] & p[1] & p[0] & cin),
c_tmp[4] = g[4] | ( p[4] & g[3])
                | ( p[4] & p[3] & g[2])
                | ( p[4] & p[3] & p[2] & g[1])
                | ( p[4] & p[3] & p[2] & p[1] & g[0])
                | ( p[4] & p[3] & p[2] & p[1] & p[0] &cin),
c_tmp[5] = g[5] | ( p[5] & g[4])
                | ( p[5] & p[4] & g[3])
                | ( p[5] & p[4] & p[3] & g[2])
```

```
                | ( p[5] & p[4] & p[3] & p[2] & g[1])
                | ( p[5] & p[4] & p[3] & p[2] & p[1] & g[0])
                | ( p[5] & p[4] & p[3] & p[2] & p[1] & p[0] & cin),
c_tmp[6] = g[6] | ( p[6] & g[5])
                | ( p[6] & p[5] & g[4])
                | ( p[6] & p[5] & p[4] & g[3])
                | ( p[6] & p[5] & p[4] & p[3] & g[2])
                | ( p[6] & p[5] & p[4] & p[3] & p[2] & g[1])
                | ( p[6] & p[5] & p[4] & p[3] & p[2] & p[1] & g[0])
                | ( p[6] & p[5] & p[4] & p[3] & p[2] & p[1] & p[0] & cin),
c_tmp[7] = g[7] | ( p[7] & g[6])
                | ( p[7] & p[6] & g[5])
                | ( p[7] & p[6] & p[5] & g[4] )
                | ( p[7] & p[6] & p[5] & p[4] & g[3])
                | ( p[7] & p[6] & p[5] & p[4] & p[3] & g[2])
                | ( p[7] & p[6] & p[5] & p[4] & p[3] & p[2] & g[1])
                | ( p[7] & p[6] & p[5] & p[4] & p[3] & p[2] & p[1] & g[0])
                | ( p[7] & p[6]& p[5] & p[4] & p[3] & p[2] & p[1] & p[0] & cin);
assign s[7:0] = a[7:0] ^ b[7:0] ^{c_tmp[6:0],cin};
endmodule
```

接下来对该8位加法器进行仿真测试

仿真测试Verilog代码如下

```
`timescale 1ns/10ps
module testaddsum();
    reg [7:0] a,b;
    wire [7:0] sum;
    reg cin;
    wire co;
    add_8 u0(
        .a(a),
        .b(b),
        .cin(cin),
        .s(sum),
        .co(co)
        );
    initial begin
        $dumpfile("testadd_8.vcd");
        $dumpvars;
        a=8'b0000_1111;     b=8'b1111_0000;     cin=0;
        $monitor ("%0t a=%8b b=%8b cin=%1b sum=%8b co=%1b", $time, a, b, cin,sum,c
o);
        #5;
        a=8'b0101_0101;     b=8'b1010_1010;     cin=0;
```

```
        $monitor ("%0t a=%8b b=%8b cin=%1b sum=%8b co=%1b", $time, a, b, cin,sum,c
o);
        #5;
        a=8'b0000_1111;     b=8'b1111_0000;     cin=1;
        $monitor ("%0t a=%8b b=%8b cin=%1b sum=%8b co=%1b", $time, a, b, cin,sum,c
o);
        #5;
        a=8'b0001_1111;     b=8'b1111_0000;     cin=0;
        $monitor ("%0t a=%8b b=%8b cin=%1b sum=%8b co=%1b", $time, a, b, cin,sum,c
o);
        #5;

    end
```

接下来按照引脚分配创建约束文件下载程序到实验板上同样用这几组数据测试。

第二部分实现8位超前进位加法器的设计结束

四、实验结果

第一部分. 8位带进位，溢出、判零功能的加减法运算器结果

仿真波形图与输出语句分别如下

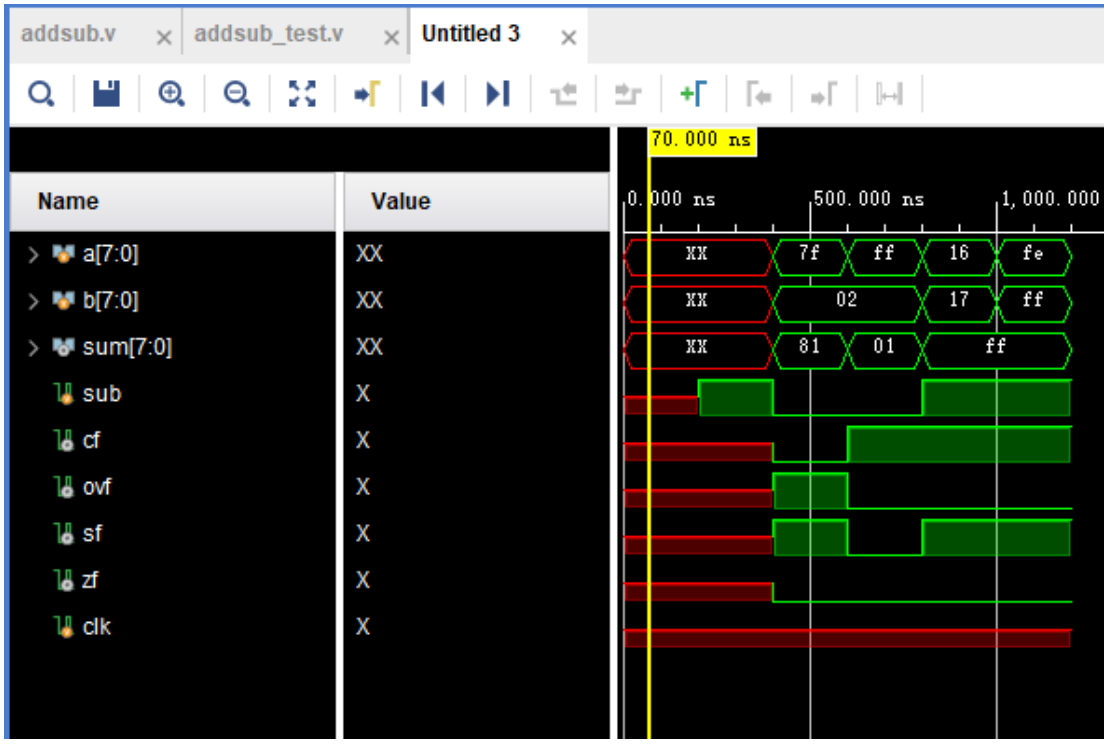

图-1 vivado展示仿真测试运行加减法运算器波形图

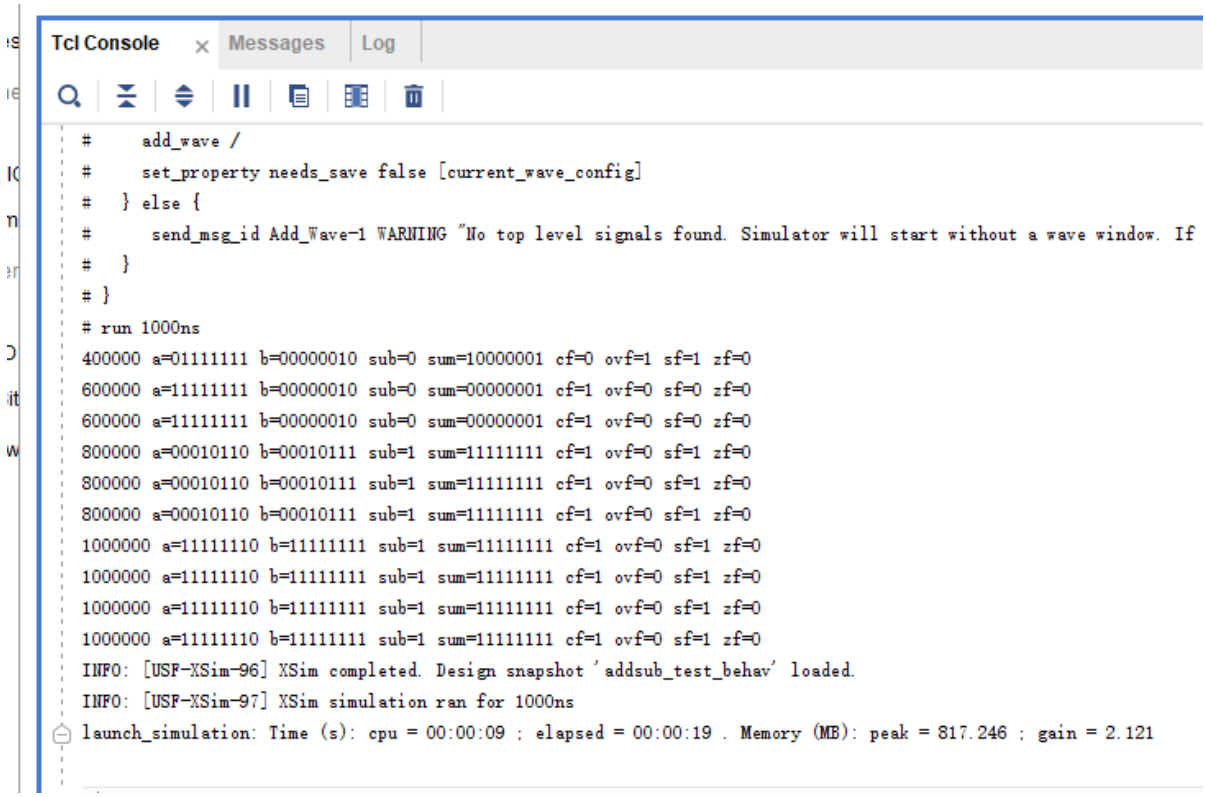

图-2 vivado展示仿真测试运行加减法运算器命令行输出

（注：不知道为何vivado中一组数据输出了多次）

自己配置了iVerilog编译环境和GTKwave波形显示器进行仿真测试波形，vvp执行编译后的程

序，再打开vcd波形文件，基本上是秒出结果，实验结果如下。

下面用于运行测试的 addsub_test.ps1的powershell脚本文件文件如下

```
$source_module="addsub"
$testbentch_module="bench\simadd"
iverilog -o "$testbentch_module.vvp" "$testbentch_module.v" "$source_module.v"
vvp -n "$testbentch_module.vvp"
gtkwave "testaddsub.vcd"
pause
```

```
Avarpow@DESKTOP-CCVBI4S  D:\programme\Computer-Organization-Experiment\ex3adder\sim
ulation  master ≡ +11 ~13 -0 !                                          [22:34]
❯ .\addsub_test.ps1
VCD info: dumpfile testaddsub.vcd opened for output.
400000 a=01111111 b=00000010 sub=0 sum=10000001 cf=0 ovf=1 sf=1 zf=0
600000 a=11111111 b=00000010 sub=0 sum=00000001 cf=1 ovf=0 sf=0 zf=0
800000 a=00010110 b=00010111 sub=1 sum=11111111 cf=1 ovf=0 sf=1 zf=0
1000000 a=11111110 b=11111111 sub=1 sum=11111111 cf=1 ovf=0 sf=1 zf=0

GTKWave Analyzer v3.3.100 (w)1999-2019 BSI

[0] start time.
[1200000] end time.
libpng warning: iCCP: cHRM chunk does not match sRGB
```

图-3 通过命令行脚本调用iverilog编译器并用vvp运行

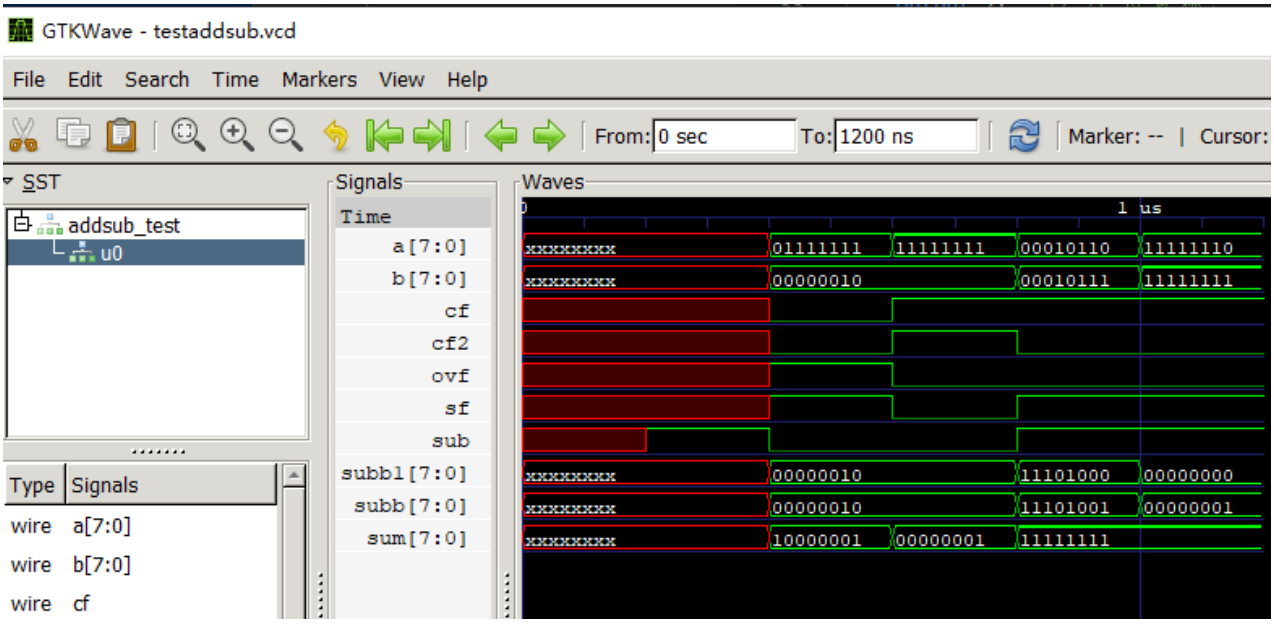

图-4 GTKwave展示二进制仿真测试运行加减法运算器波形图

由于该二进制形式的波形不方便分析，选择带符号数字做为输出形式方便观察

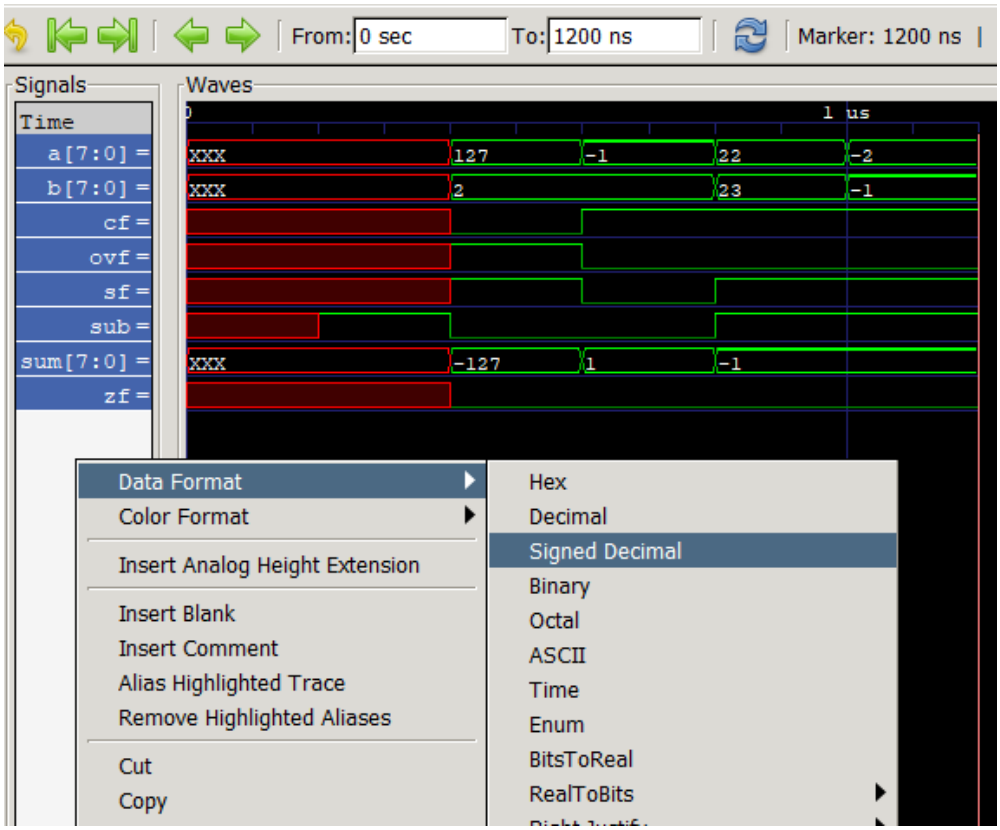

图-5 GTKwave展示有符号数十进制仿真测试运行加减法运算器波形图

分析：

第一组数据：a=127 b=2 sub=0 a+b=-127 进位cf=0 溢出ovf=1，符号sf=1，zf=0,正确

第二组数据：a=-1 b=2 sub=0 a+b=1 进位cf=0 溢出ovf=0，符号sf=1，zf=0,正确

第三组数据：a=22 b=23 sub=1 a+b=-1 进位cf=1 溢出ovf=0，符号sf=1，zf=0,正确

第四组数据：a=-2 b=-1 sub=0 a+b=-1 进位cf=1 溢出ovf=0，符号sf=1，zf=0,正确

通过上述测试结果，该模块的加法，减法输出，溢出，进位判断运行正常，完成了8位带进位，溢出、判零功能的加减法运算器的实现。

实验板测试

（注：左侧四个灯分别为 cf,ovf,sf,zf　右侧8个灯为加法结果）

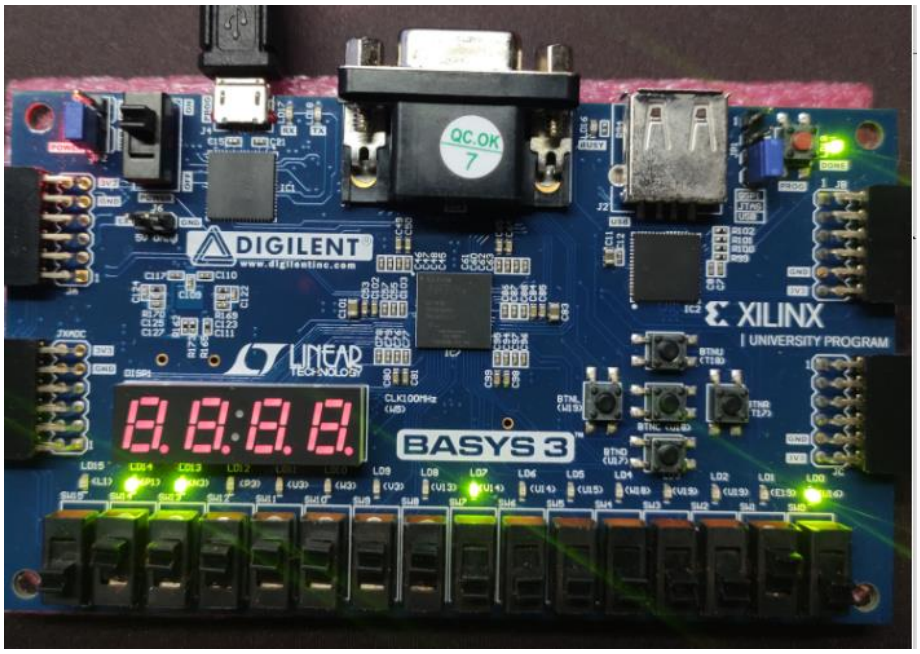

图-6 加减法器第一组测试

第一组数据：a=127 b=2 sub=0 a+b=-127 进位cf=0 溢出ovf=1，符号sf=1，zf=0,正确

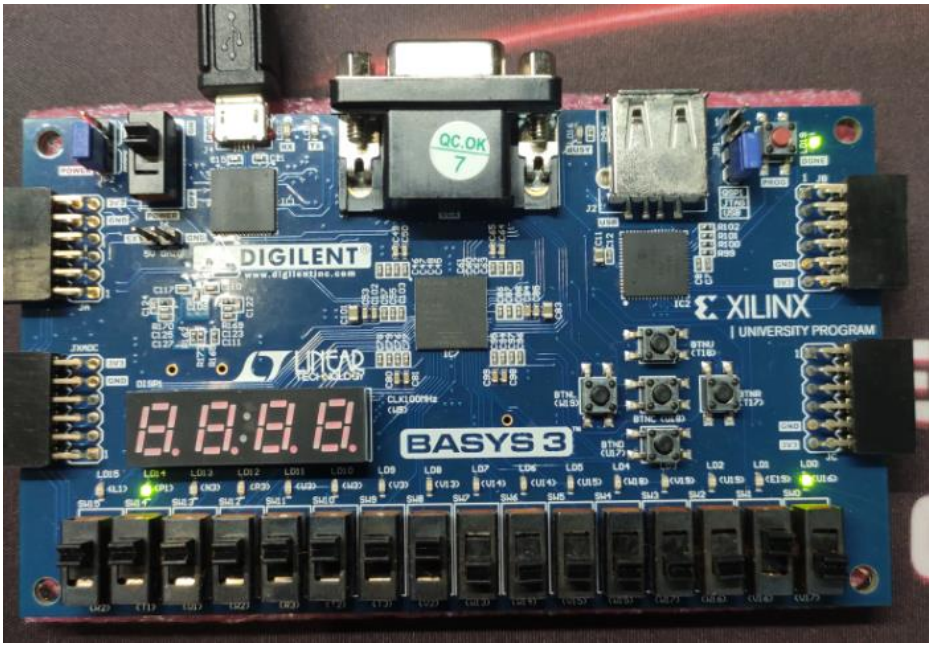

图-7 加减法器第二组测试

第二组数据：a=-1 b=2 sub=0 a+b=1 进位cf=0 溢出ovf=0，符号sf=1，zf=0,正确

（注：左侧四个灯分别为 cf,ovf,sf,zf　右侧8个灯为加法结果）

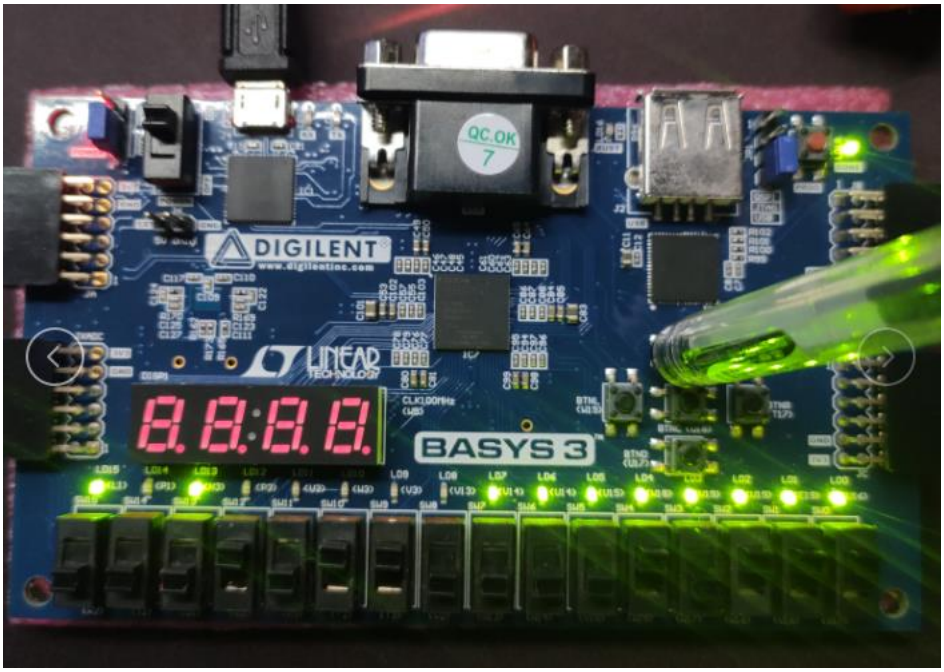

图-8 加减法器第三组测试

第三组数据：a=22 b=23 sub（被按下）=1 a+b=-1 进位cf=1 溢出ovf=0，符号sf=1，zf=0,正确

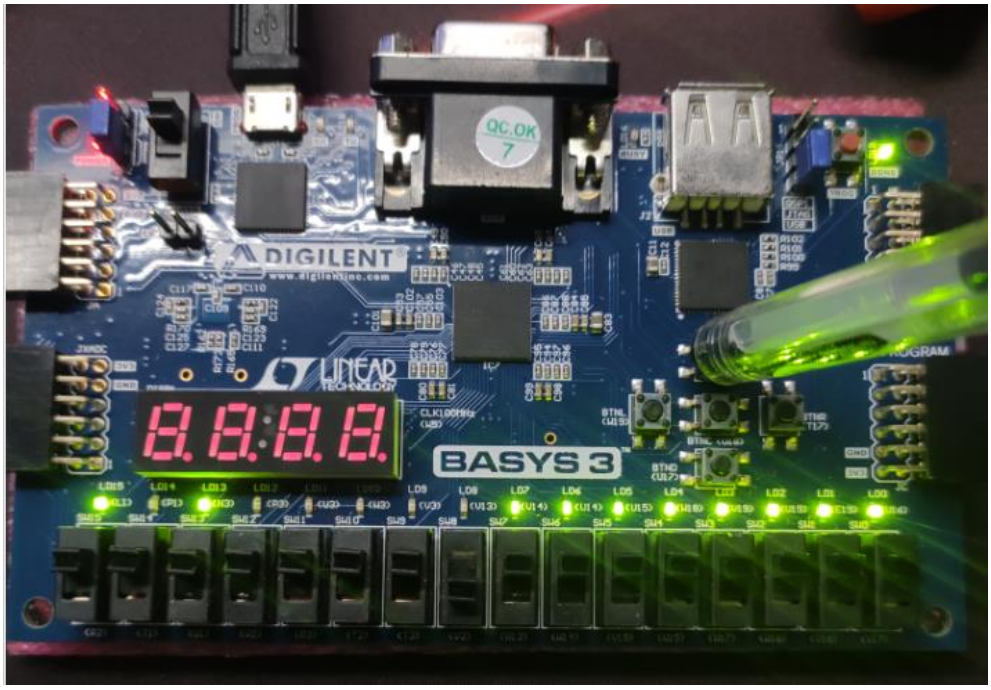

图-9 加减法器第四组测试

第四组数据：a=-2 b=-1 sub=0 a+b=-1 进位cf=1 溢出ovf=0，符号sf=1，zf=0,正确

综上，该部分完成了实现8位带进位，溢出、判零功能的加减法运算器。

第二部分：超前进位8位加法器的设计

仿真波形图与文本输出如下

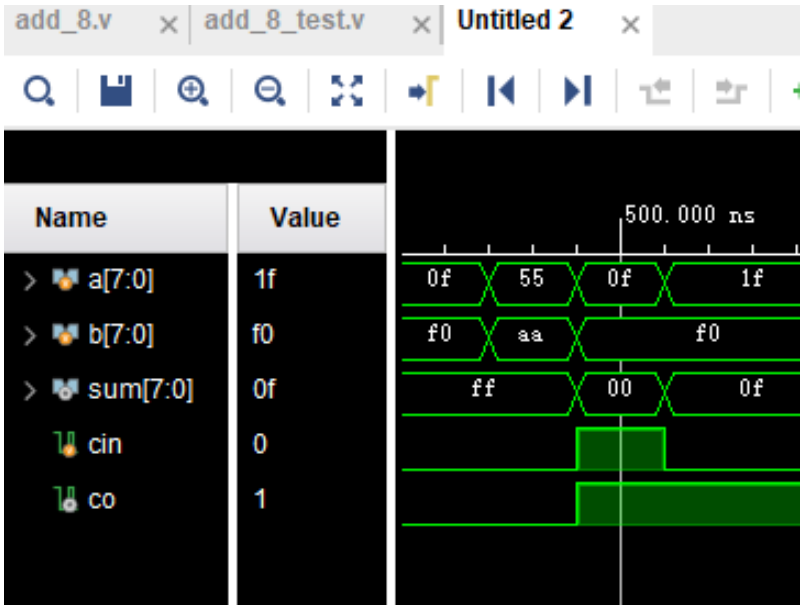

图-10 vivado展示超前进位8位加法器波形图

```
# run 1000ns
0 a=00001111 b=11110000 cin=0 sum=11111111 co=0
200000 a=01010101 b=10101010 cin=0 sum=11111111 co=0
200000 a=01010101 b=10101010 cin=0 sum=11111111 co=0
400000 a=00001111 b=11110000 cin=1 sum=00000000 co=1
400000 a=00001111 b=11110000 cin=1 sum=00000000 co=1
400000 a=00001111 b=11110000 cin=1 sum=00000000 co=1
600000 a=00011111 b=11110000 cin=0 sum=00001111 co=1
600000 a=00011111 b=11110000 cin=0 sum=00001111 co=1
600000 a=00011111 b=11110000 cin=0 sum=00001111 co=1
600000 a=00011111 b=11110000 cin=0 sum=00001111 co=1
INFO: [USF-XSim-96] XSim completed. Design snapshot 'add_8_test_behav' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 1000ns
launch_simulation: Time (s): cpu = 00:00:05 ; elapsed = 00:00:07 . Memory (MB): peak = 829.887 ; gain = 0.000
```

图-11 vivado展示超前进位8位加法器命令行输出

```
Avarpow@DESKTOP-CCVBI4S  D:\programme\Computer-Organization-Experiment\ex3adder\sim
ulation  master ≡ +14 ~13 -0 !                                          [00:38]
> .\addsub_test.ps1
VCD info: dumpfile testaddsub.vcd opened for output.
400000 a=01111111 b=00000010 sub=0 sum=10000001 cf=0 ovf=1 sf=1 zf=0
600000 a=11111111 b=00000010 sub=0 sum=00000001 cf=1 ovf=0 sf=0 zf=0
800000 a=00010110 b=00010111 sub=1 sum=11111111 cf=1 ovf=0 sf=1 zf=0
1000000 a=11111110 b=11111111 sub=1 sum=11111111 cf=1 ovf=0 sf=1 zf=0

GTKWave Analyzer v3.3.100 (w)1999-2019 BSI

[0] start time.
[1200000] end time.
```

图-12 iverilog展示超前进位8位加法器命令行输出

（注：不知道为何vivado中一组数据输出了多次）

分析：

第一组测试数据为00001111+11110000，输入进位为0 输出为11111111，输出进位0，正确，

第二组测试数据为01010101+10101010，输入进位为0 输出为11111111，输出进位0，正确，

第三组测试数据为00001111+11110000，输入进位为1 输出为00000000，输出进位1，正确，

第四组测试数据为00011111+11110000，输入进位为0 输出为00001111，输出进位1，正确，

通过上述仿真测试结果，该模块的加法，进位运行正常，完成了8位超前进位加法器的实现。

实验板上测试

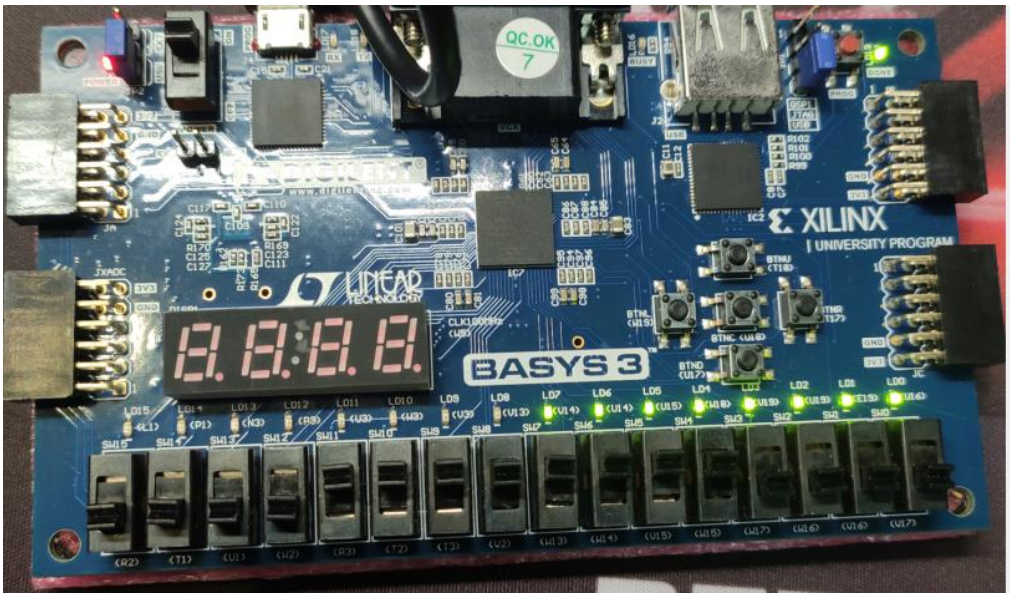

图-13 超前进位8位加法器第一组测试

第一组测试数据为00001111+11110000，输入进位为0 输出为11111111，输出进位0，正确，

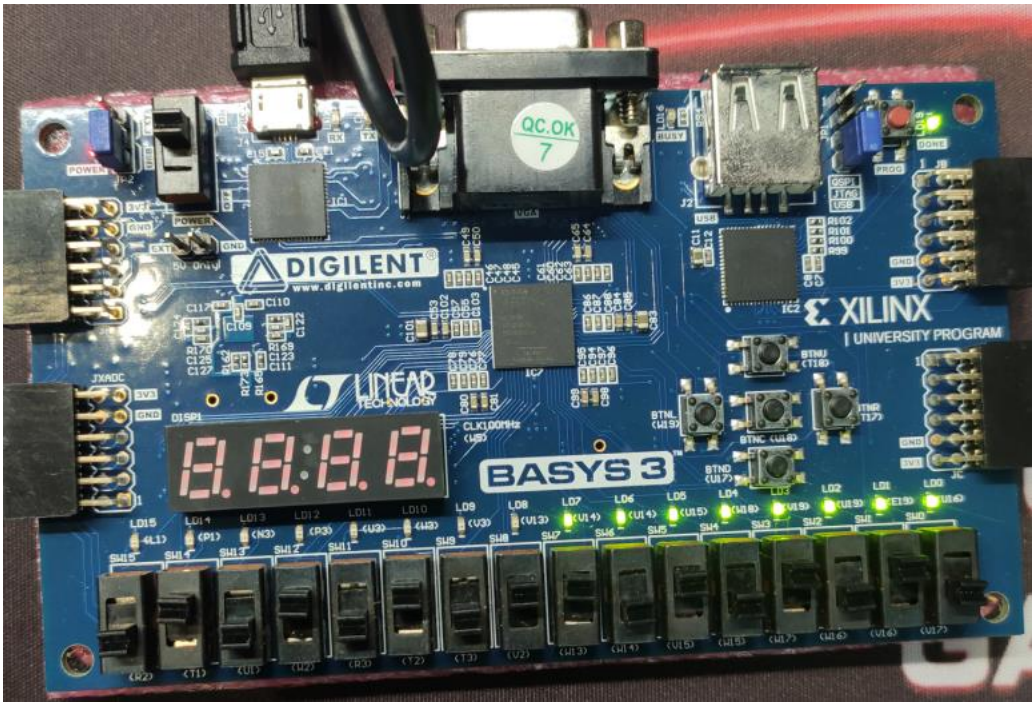

图-14 超前进位8位加法器第二组测试

第二组测试数据为01010101+10101010，输入进位为0 输出为11111111，输出进位0，正确，

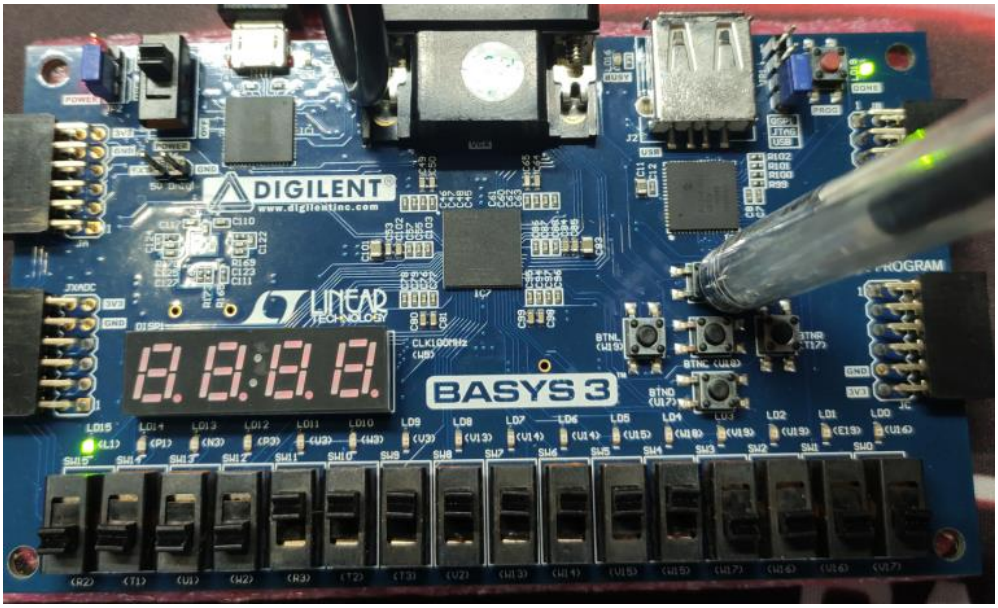

图-15 超前进位8位加法器第三组测试

第三组测试数据为00001111+11110000，输入进位为1 输出为00000000，输出进位1，正确，

（用笔点击进位输入按钮，左侧为进位输出灯）

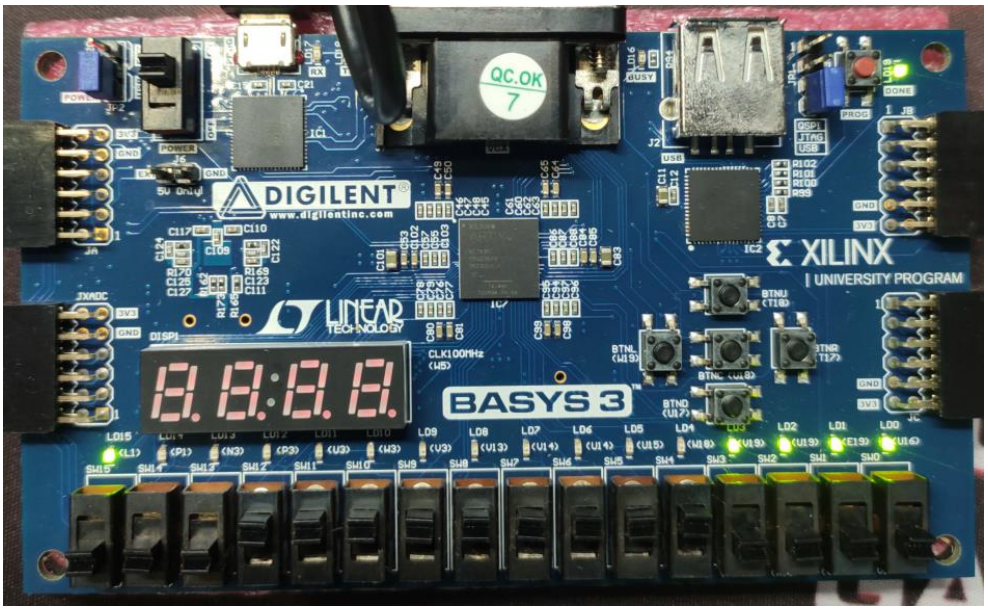

图-16 超前进位8位加法器第四组测试

第四组测试数据为00011111+11110000，输入进位为0 输出为00001111，左侧输出进位灯亮起，输出进位1，正确，

综上，该部分完成了8位超前进位加法器的实现。

五、实验感想

本次实验完成了两个任务，一个是实现 8 位带进位，溢出、判零功能的加减法运算器，以及实现 8 位超前进位加法器。进一步了解了 verilog 的使用，同时学习了更多的调试代码的方法，比如使用$dumpvars 与$dumpfiles 来导出 vcd 通用波形文件，使用$monitor 来输出变量内容，同时使用字符串格式化如%8b 输出 8 位二进制文件。再就是在设计过程中要学会编写测试用例，通过多组完善的测试用例来检验自己的设计是不是符合预期，来避免错误的发生。除此之外，我本周还学习使用了除了 vivado 之外的 Verilog 编译器，例如

Icarus Verilog（http://iverilog.icarus.com/）

,比 vivado 轻量化很多，对于简单的仿真可以秒出结果，帮助我加速了开发的过程。同时测试使用了 GTKwave 软件来查看.Vcd 波形图文件，该软件可以通过多种格式，如二进制，十

进制，十六进制等等展示波形图，vivado 中如何调整显示的格式目前还不太会使用，希望能在后续的实验练习中学会使用。

附录（流程图，注释过的代码）：

本次实验为两个单模块项目，各个模块单独设计，无相关联的流程。

代码：

8 位带进位，溢出、判零功能的加减法运算器模块

addsub.v

```
`timescale 1ns / 1ps



module addsub
    #(parameter WIDTH=8)    //指定数据宽度参数，缺省值是 8 (
   (
    input [(WIDTH-1):0] a,  // 缺省位数由参数 WIDTH 决定， a 为第一位运算数
    input [(WIDTH-1):0] b,  // b 为第二位运算数
    input sub,  // sub=1 为减法，sub=0 为加法
    output [(WIDTH-1):0] sum, // sum 为 a 与 b 运算的结果
    output cf,  // cf 为进位标志
    output ovf, // ovf 为溢出标志
    output sf,  // sf 为符号标志
    output zf   // zf 为 0 标志
    );
    wire [(WIDTH-1):0] subb,subb1;
    wire cf2; // 进 位
    assign subb1 = b ^ {WIDTH{sub}}; // 对于减法是取反
    assign subb  =  subb1 + sub;    // 对于减法是加 1，sub=1（减法）sub=0（加法）
// 减法：subb = b 取反+1 ；  加法：subb = b
    assign {cf2,sum} = a + subb;
    assign sf = sum[WIDTH-1];       //sum 的最高位为符号位
    assign zf = (sum == 0) ? 1 : 0 ; //sum=0， 则 0 标 志 zf=1
    assign cf = cf2 ^ sub;  //进位 cf = cf2 ^ sub
    assign ovf = (a[WIDTH-1] ^ sum[WIDTH-1]) & (subb[WIDTH-1] ^ sum[WIDTH-1]

); //判断溢出
endmodule
```

8 位带进位，溢出、判零功能的加减法运算器仿真测试代码

simadd.v

```
`timescale 1ns/10ps
module addsub_test();
    reg [7:0] a,b;
    wire [7:0] sum;
    reg sub;
    wire cf,ovf,sf,zf;
    reg clk;
    addsub u0(
        .a(a),
        .b(b),
        .sub(sub),
        .sum(sum),
        .cf(cf),
        .ovf(ovf),
        .sf(sf),
        .zf(zf)
        );
    initial begin
        $dumpfile("testaddsub.vcd");
        $dumpvars;
        #200 sub = 1;
        #200 a = 8'h7f; b = 8'h2; sub = 0;
        $monitor ("%0t a=%8b b=%8b sub=%1b sum=%8b cf=%1b ovf=%1b sf=%1b zf=%1b",
$time, a, b, sub,sum,cf,ovf,sf,zf);
        #200 a = 8'hff; b = 8'h2; sub = 0;
        $monitor ("%0t a=%8b b=%8b sub=%1b sum=%8b cf=%1b ovf=%1b sf=%1b zf=%1b",
$time, a, b, sub,sum,cf,ovf,sf,zf);
        #200 a = 8'h16; b = 8'h17; sub = 1;
        $monitor ("%0t a=%8b b=%8b sub=%1b sum=%8b cf=%1b ovf=%1b sf=%1b zf=%1b",
$time, a, b, sub,sum,cf,ovf,sf,zf);
        #200 a = 8'hfe; b = 8'hff; sub = 1;
        $monitor ("%0t a=%8b b=%8b sub=%1b sum=%8b cf=%1b ovf=%1b sf=%1b zf=%1b",
$time, a, b, sub,sum,cf,ovf,sf,zf);
        #200;
    end

endmodule
```

8 位带进位，溢出、判零功能的加减法运算器使用 iverilog 运行脚本

addsub_test.ps1

```
$source_module="addsub"
```

```
$testbentch_module="bench\simadd"
    $testbentch_module.vvp
iverilog -o "$testbentch_module.vvp" "$testbentch_module.v" "$source_module.v"
vvp -n "$testbentch_module.vvp"
gtkwave "testaddsub.vcd"

pause
```

引脚约束文件

Addsub_pin.xdc

```
set_property IOSTANDARD LVCMOS33 [get_ports {a[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {a[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {a[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {a[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {a[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {a[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {a[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {a[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {b[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {b[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {b[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {b[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {b[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {b[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {b[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {b[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sum[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sum[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sum[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sum[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sum[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sum[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sum[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sum[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports cf]
set_property IOSTANDARD LVCMOS33 [get_ports ovf]
set_property IOSTANDARD LVCMOS33 [get_ports sf]
set_property IOSTANDARD LVCMOS33 [get_ports sub]
set_property IOSTANDARD LVCMOS33 [get_ports zf]
set_property PACKAGE_PIN R2 [get_ports {a[7]}]
set_property PACKAGE_PIN T1 [get_ports {a[6]}]
set_property PACKAGE_PIN U1 [get_ports {a[5]}]
set_property PACKAGE_PIN W2 [get_ports {a[4]}]
set_property PACKAGE_PIN R3 [get_ports {a[3]}]
```

```
set_property PACKAGE_PIN T2 [get_ports {a[2]}]
set_property PACKAGE_PIN T3 [get_ports {a[1]}]
set_property PACKAGE_PIN V2 [get_ports {a[0]}]
set_property PACKAGE_PIN W13 [get_ports {b[7]}]
set_property PACKAGE_PIN W14 [get_ports {b[6]}]
set_property PACKAGE_PIN V15 [get_ports {b[5]}]
set_property PACKAGE_PIN W15 [get_ports {b[4]}]
set_property PACKAGE_PIN W17 [get_ports {b[3]}]
set_property PACKAGE_PIN W16 [get_ports {b[2]}]
set_property PACKAGE_PIN V16 [get_ports {b[1]}]
set_property PACKAGE_PIN V17 [get_ports {b[0]}]
set_property PACKAGE_PIN V14 [get_ports {sum[7]}]
set_property PACKAGE_PIN U14 [get_ports {sum[6]}]
set_property PACKAGE_PIN U15 [get_ports {sum[5]}]
set_property PACKAGE_PIN W18 [get_ports {sum[4]}]
set_property PACKAGE_PIN V19 [get_ports {sum[3]}]
set_property PACKAGE_PIN U19 [get_ports {sum[2]}]
set_property PACKAGE_PIN E19 [get_ports {sum[1]}]
set_property PACKAGE_PIN U16 [get_ports {sum[0]}]
set_property PACKAGE_PIN T18 [get_ports sub]
set_property PACKAGE_PIN L1 [get_ports cf]
set_property PACKAGE_PIN P1 [get_ports ovf]
set_property PACKAGE_PIN N3 [get_ports sf]
set_property PACKAGE_PIN P3 [get_ports zf]
```

==========

8 位超前进位加法器模块

add_8.v

```
module add_8 ( input [7:0]a, input [7:0]b, input cin, output [7:0] s, output co );

wire [7:0]c_tmp;
wire [7:0]g;
wire [7:0]p;
assign co = c_tmp[7];
assign
        //G 表示不考虑后面的进位，这里是否有进位
        g = a & b;
assign
        //P 表示 a 或 b 在这一位上至少有一个
        p = a | b;
assign
c_tmp[0] = g[0] | ( p[0] & cin ),
c_tmp[1] = g[1] | ( p[1] & g[0])
                | ( p[1] & p[0] & cin),
c_tmp[2] = g[2] | ( p[2] & g[1])
```

```
                | ( p[2] & p[1] & g[0])
                | ( p[2] & p[1] & p[0] & cin),
c_tmp[3] = g[3] | ( p[3] & g[2])
                | ( p[3] & p[2] & g[1])
                | ( p[3] & p[2] & p[1] & g[0])
                | ( p[3] & p[2] & p[1] & p[0] & cin),
c_tmp[4] = g[4] | ( p[4] & g[3])
                | ( p[4] & p[3] & g[2])
                | ( p[4] & p[3] & p[2] & g[1])
                | ( p[4] & p[3] & p[2] & p[1] & g[0])
                | ( p[4] & p[3] & p[2] & p[1] & p[0] &cin),
c_tmp[5] = g[5] | ( p[5] & g[4])
                | ( p[5] & p[4] & g[3])
                | ( p[5] & p[4] & p[3] & g[2])
                | ( p[5] & p[4] & p[3] & p[2] & g[1])
                | ( p[5] & p[4] & p[3] & p[2] & p[1] & g[0])
                | ( p[5] & p[4] & p[3] & p[2] & p[1] & p[0] & cin),
c_tmp[6] = g[6] | ( p[6] & g[5])
                | ( p[6] & p[5] & g[4])
                | ( p[6] & p[5] & p[4] & g[3])
                | ( p[6] & p[5] & p[4] & p[3] & g[2])
                | ( p[6] & p[5] & p[4] & p[3] & p[2] & g[1])
                | ( p[6] & p[5] & p[4] & p[3] & p[2] & p[1] & g[0])
                | ( p[6] & p[5] & p[4] & p[3] & p[2] & p[1] & p[0] & cin),
c_tmp[7] = g[7] | ( p[7] & g[6])
                | ( p[7] & p[6] & g[5])
                | ( p[7] & p[6] & p[5] & g[4] )
                | ( p[7] & p[6] & p[5] & p[4] & g[3])
                | ( p[7] & p[6] & p[5] & p[4] & p[3] & g[2])
                | ( p[7] & p[6] & p[5] & p[4] & p[3] & p[2] & g[1])
                | ( p[7] & p[6] & p[5] & p[4] & p[3] & p[2] & p[1] & g[0])
                | ( p[7] & p[6]& p[5] & p[4] & p[3] & p[2] & p[1] & p[0] & cin);
assign s[7:0] = a[7:0] ^ b[7:0] ^{c_tmp[6:0],cin};
endmodule
```

8 位超前进位加法器仿真测试代码

simadd_8.v

```
`timescale 1ns/10ps
module testaddsum();
    reg [7:0] a,b;
    wire [7:0] sum;
    reg cin;
    wire co;
    add_8 u0(
        .a(a),
```

```
        .b(b),
        .cin(cin),
        .s(sum),
        .co(co)
        );
    initial begin
        $dumpfile("testadd_8.vcd");
        $dumpvars;
        a=8'b0000_1111;     b=8'b1111_0000;     cin=0;
        $monitor ("%0t a=%8b b=%8b cin=%1b sum=%8b co=%1b", $time, a, b, cin,sum,c
o);
        #5;
        a=8'b0101_0101;     b=8'b1010_1010;     cin=0;
        $monitor ("%0t a=%8b b=%8b cin=%1b sum=%8b co=%1b", $time, a, b, cin,sum,c
o);
        #5;
        a=8'b0000_1111;     b=8'b1111_0000;     cin=1;
        $monitor ("%0t a=%8b b=%8b cin=%1b sum=%8b co=%1b", $time, a, b, cin,sum,c
o);
        #5;
        a=8'b0001_1111;     b=8'b1111_0000;     cin=0;
        $monitor ("%0t a=%8b b=%8b cin=%1b sum=%8b co=%1b", $time, a, b, cin,sum,c
o);
        #5;

    end

endmodule
```

8 位超前进位加法器使用 iverilog 运行脚本

add_8_test.ps1

```
$source_module="add_8"
$testbentch_module="bench\simadd_8"
    $testbentch_module.vvp
iverilog -o "$testbentch_module.vvp" "$testbentch_module.v" "$source_module.v"
vvp -n "$testbentch_module.vvp"
gtkwave "testadd_8.vcd"
pause
```

引脚约束文件

add8_pin.xdc

```
set_property IOSTANDARD LVCMOS33 [get_ports {a[7]}]
```

```
set_property IOSTANDARD LVCMOS33 [get_ports {a[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {a[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {a[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {a[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {a[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {a[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {a[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {b[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {b[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {b[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {b[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {b[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {b[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {b[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {b[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {s[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {s[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {s[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {s[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {s[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {s[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {s[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {s[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports cin]
set_property IOSTANDARD LVCMOS33 [get_ports co]
set_property PACKAGE_PIN R2 [get_ports {a[7]}]
set_property PACKAGE_PIN T1 [get_ports {a[6]}]
set_property PACKAGE_PIN U1 [get_ports {a[5]}]
set_property PACKAGE_PIN W2 [get_ports {a[4]}]
set_property PACKAGE_PIN R3 [get_ports {a[3]}]
set_property PACKAGE_PIN T2 [get_ports {a[2]}]
set_property PACKAGE_PIN T3 [get_ports {a[1]}]
set_property PACKAGE_PIN V2 [get_ports {a[0]}]
set_property PACKAGE_PIN W13 [get_ports {b[7]}]
set_property PACKAGE_PIN W14 [get_ports {b[6]}]
set_property PACKAGE_PIN V15 [get_ports {b[5]}]
set_property PACKAGE_PIN W15 [get_ports {b[4]}]
set_property PACKAGE_PIN W17 [get_ports {b[3]}]
set_property PACKAGE_PIN W16 [get_ports {b[2]}]
set_property PACKAGE_PIN V16 [get_ports {b[1]}]
set_property PACKAGE_PIN V17 [get_ports {b[0]}]
set_property PACKAGE_PIN V14 [get_ports {s[7]}]
set_property PACKAGE_PIN U14 [get_ports {s[6]}]
set_property PACKAGE_PIN U15 [get_ports {s[5]}]
set_property PACKAGE_PIN W18 [get_ports {s[4]}]
set_property PACKAGE_PIN V19 [get_ports {s[3]}]
```

```
set_property PACKAGE_PIN U19 [get_ports {s[2]}]
set_property PACKAGE_PIN E19 [get_ports {s[1]}]
set_property PACKAGE_PIN U16 [get_ports {s[0]}]
set_property PACKAGE_PIN T18 [get_ports cin]
set_property PACKAGE_PIN L1 [get_ports co]
```