**组成原理实验 基于**vivado**设计**8位的加减法器

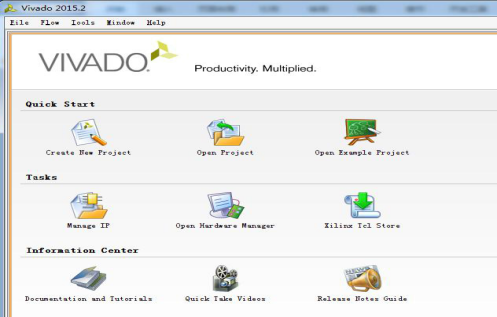
**实验内容：设计**8位的加减法器。

**建目录路径（Lab /organization/Ex\_1）**

**实验平台：**Xilinx大学计划与Digilent联合推出的“**Basys 3**” FPGA开发板

**一、打开 Vivado**

启动后，将进入以下画面：



**二、创建一个新的项目**

新项目名称为 addsub。项目创建的位置是 E:/Xlinx/VivadoProject，创建完后，整个项目文件都在E:/lab/ organization/addsub 中。

1、点击上图 Create New Project，将打开窗口如下：

2、点击 Next，进入下图：

3、在窗口中输入 addsub，并选定项目路径 E:/lab/ organization/addsub，如下图：

4、点击 Next，进入下图：

5、按上图的原选 RTL Project，点击 Next，将进入下图，因为无需建立内容，所以，以下相应窗口：不增加源文件，所以 Add source 窗口直接点击 Next（但 Target Language 选择 Verilog）；不增加 IP 核，所以 Add Existing IP 窗口直接点击 Next；不增加约束文件，所以 Add Constraints 窗口直接点击 Next；之后，将弹出窗口，如下：

6、对以上窗口进行设置，如下图：

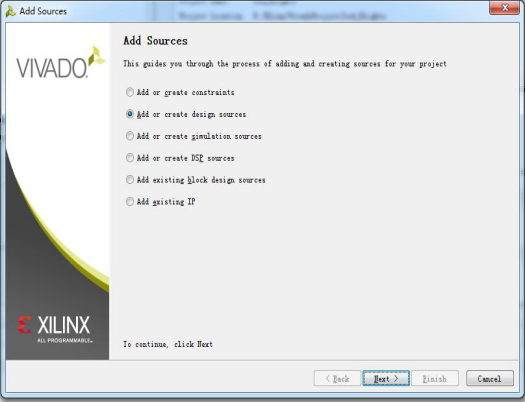
7、选择器件为 xc7a35tcpg236-1**（**Basys3 **板），**点击 Next， 将进入以下新项目概览：

8、点击 Finish 之后，进入创建项目后的界面：

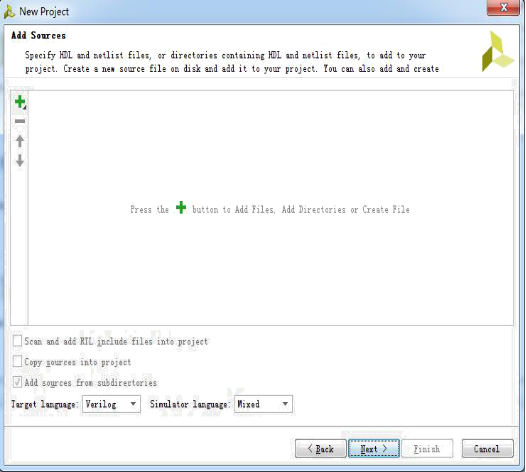
**三、添加源程序代码**

1**、添加一个可变加减运算器代码**

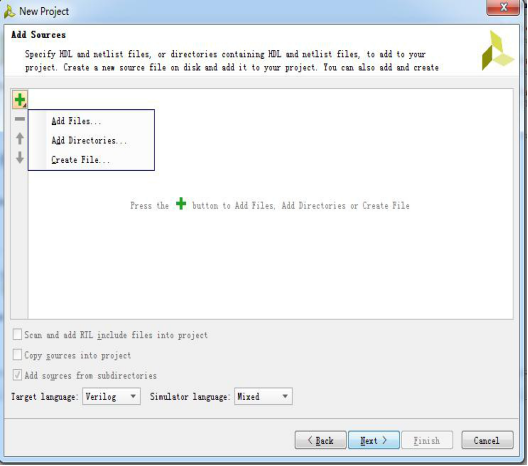
（1）在以上窗口中点击红色圈着的部分 Add Sources，或右键点击 Design Sources，在弹出的菜单中选择 Add Sources...，之后，将打开如下窗口：



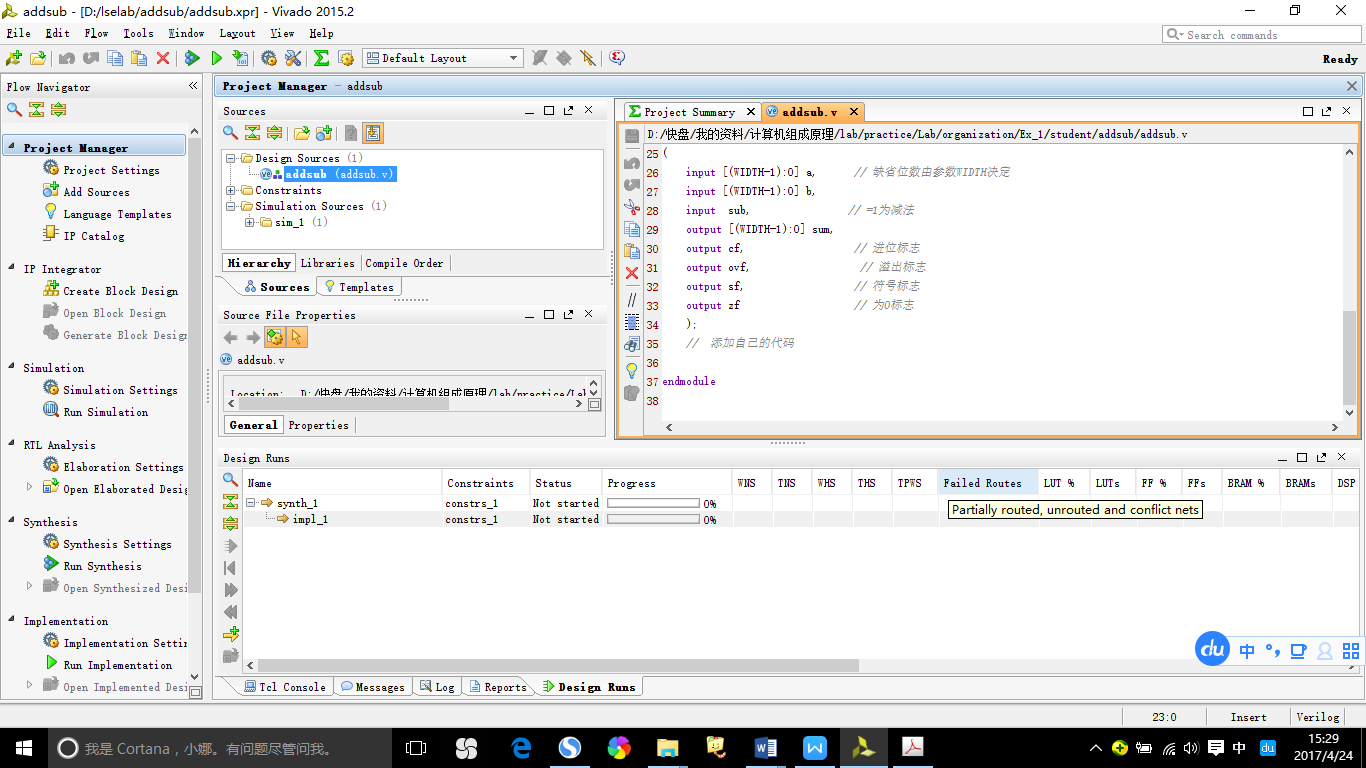
（2）按原来选择 Add or create design sources，点击 Next，将打开下窗口：



（3）在以上窗口中点击“+”之后，看看下图：



（4）在弹出的窗口中，点击 Add File...，将弹出小窗口，选择文件路径\organization\Ex\_1\student\addsub：双击addsub.v文件，如下图：



（5）在上图中，双击蓝色的部分 addsub，将出现右面窗口（代码输入窗口），看下图：

添加代码放入代码编辑窗口：这段程序描述了一个可变宽度加减运算器。依据

Subtract B from A = add 2’s complement of B to AIn 2’s complement: –B = ~B + 1

进行设计，参照





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

input [(WIDTH-1):0] b,

input sub, // =1为减法

output [(WIDTH-1):0] sum,

output cf, // 进位标志

output ovf, // 溢出标志

output sf, // 符号标志

output 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）

assign {cf2,sum} = a + subb;

//添加代码标志位

Endmodule 按 ctrl-s 存盘。

(6)添加仿真文件

如上图右击鼠标添加D: \organization\Ex\_1\student\addsub下的addsub\_sim.v文件

功能仿真：仿真文件代码如下

`timescale 1ns / 1ps

module addsub8\_sim( );

// input

reg [7:0] a = 8'h16;

reg [7:0] b = 8'h12;

reg sub = 0;

//output

wire [7:0] sum;

wire cf;

wire ovf;

wire sf;

wire zf;

// initial

addsub8 U (a,b,sub,sum,cf,ovf,sf,zf);

initial begin

#200 sub = 1;

#200 begin a = 8'h7f; b = 8'h2; sub = 0; end

#200 begin a = 8'hff; b = 8'h2; sub = 0; end

#200 begin a = 8'h16; b = 8'h17; sub = 1; end

#200 begin a = 8'hfe; b = 8'hff; sub = 1; end

end

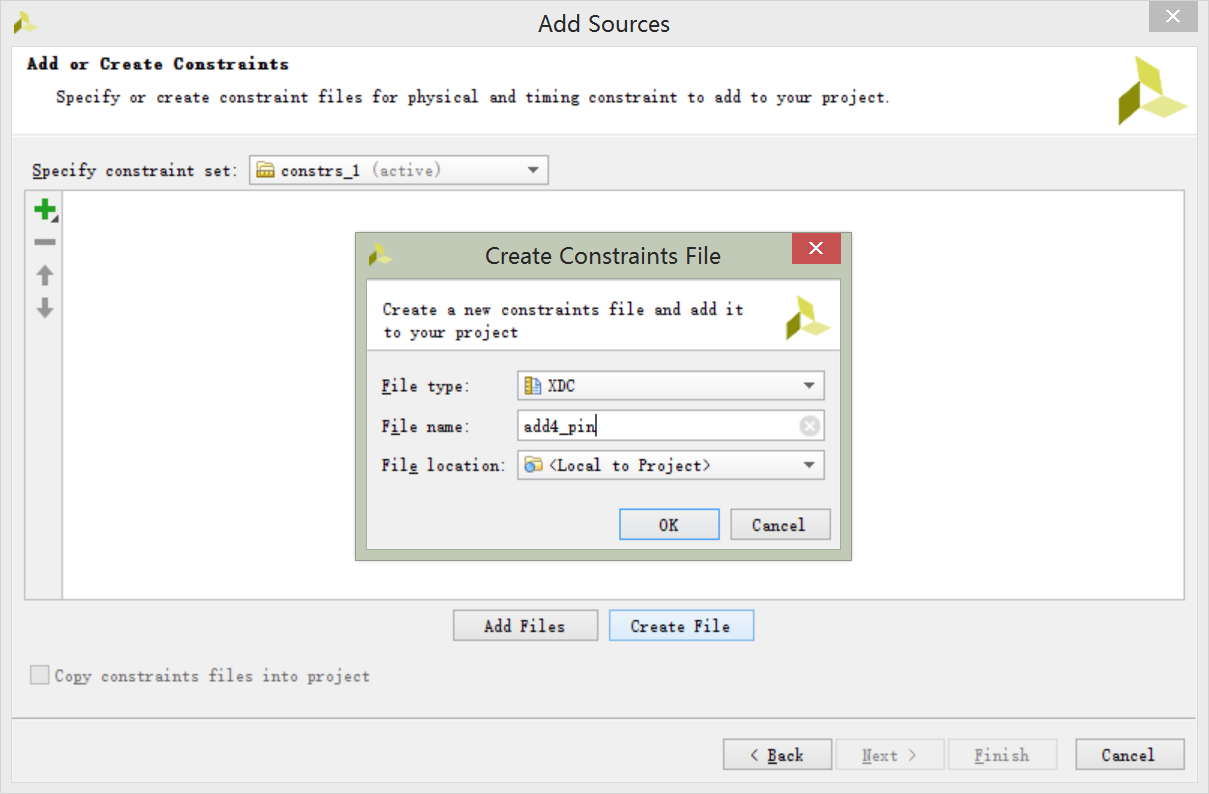
endmodule

**（7）创建约束文件**

把源文件数据宽度设定为4位，连接引脚：用SW模拟输入运算数据（比如a, b）,运算sum结果通过led输出。

在实现之前我们需要指定FPGA使用过的SW和LED的引脚。

**创建约束文件，并命名为add4\_pin**

****

**编辑约束文件：**

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 AB6 [get\_ports {a[7]}]

set\_property PACKAGE\_PIN AB7 [get\_ports {a[6]}]

set\_property PACKAGE\_PIN V7 [get\_ports {a[5]}]

set\_property PACKAGE\_PIN AA6 [get\_ports {a[4]}]

set\_property PACKAGE\_PIN Y6 [get\_ports {a[3]}]

set\_property PACKAGE\_PIN T6 [get\_ports {a[2]}]

set\_property PACKAGE\_PIN R6 [get\_ports {a[1]}]

set\_property PACKAGE\_PIN V5 [get\_ports {a[0]}]

set\_property PACKAGE\_PIN U6 [get\_ports {b[7]}]

set\_property PACKAGE\_PIN W5 [get\_ports {b[6]}]

set\_property PACKAGE\_PIN W6 [get\_ports {b[5]}]

set\_property PACKAGE\_PIN U5 [get\_ports {b[4]}]

set\_property PACKAGE\_PIN T5 [get\_ports {b[3]}]

set\_property PACKAGE\_PIN T4 [get\_ports {b[2]}]

set\_property PACKAGE\_PIN R4 [get\_ports {b[1]}]

set\_property PACKAGE\_PIN W4 [get\_ports {b[0]}]

set\_property PACKAGE\_PIN F21 [get\_ports {sum[7]}]

set\_property PACKAGE\_PIN G22 [get\_ports {sum[6]}]

set\_property PACKAGE\_PIN G21 [get\_ports {sum[5]}]

set\_property PACKAGE\_PIN D21 [get\_ports {sum[4]}]

set\_property PACKAGE\_PIN E21 [get\_ports {sum[3]}]

set\_property PACKAGE\_PIN D22 [get\_ports {sum[2]}]

set\_property PACKAGE\_PIN E22 [get\_ports {sum[1]}]

set\_property PACKAGE\_PIN A21 [get\_ports {sum[0]}]

set\_property PACKAGE\_PIN M17 [get\_ports ovf]

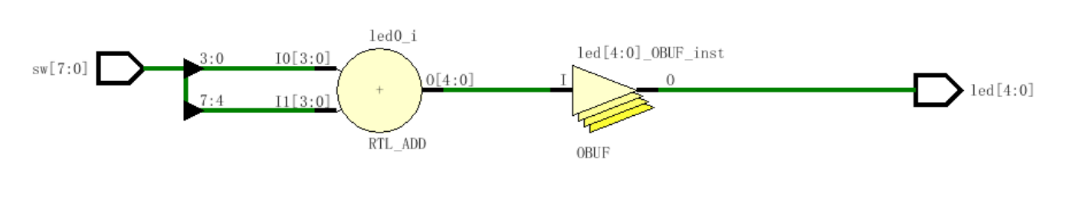
set\_property PACKAGE\_PIN M15 [get\_ports sf]

set\_property PACKAGE\_PIN K16 [get\_ports zf]

set\_property PACKAGE\_PIN Y9 [get\_ports sub]

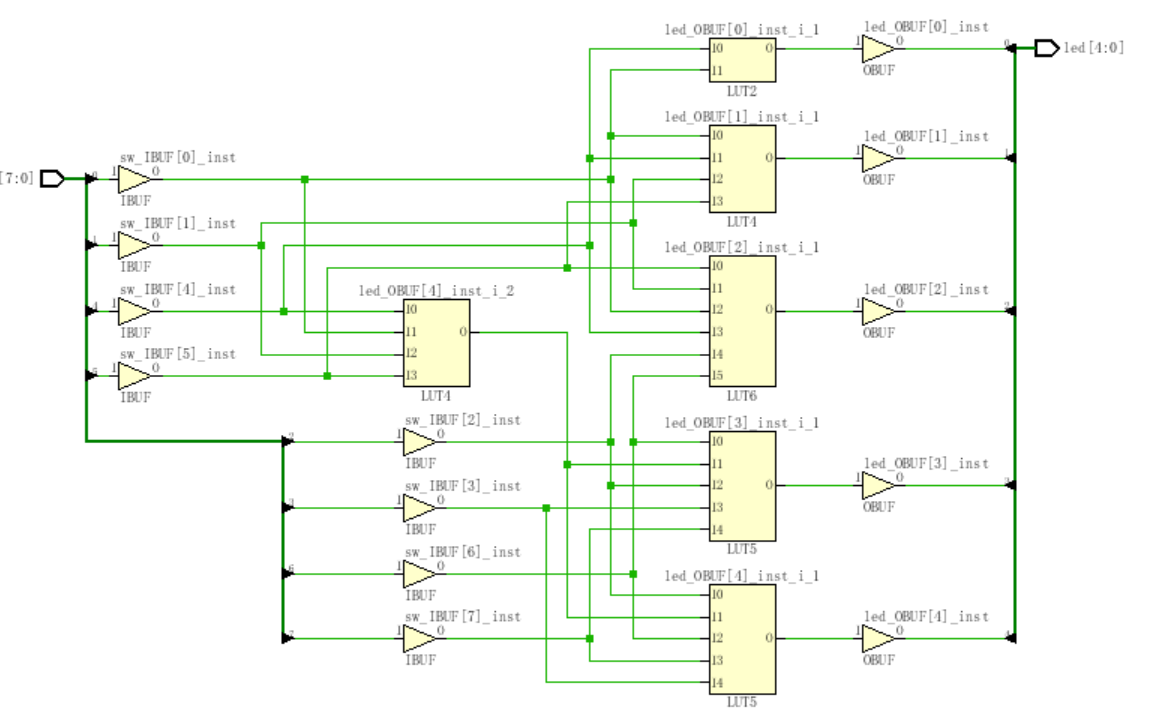
set\_property PACKAGE\_PIN M16 [get\_ports cf]

三. RTL分析，生成原理图，见下图



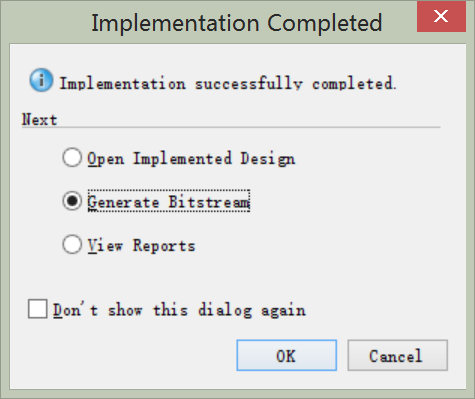
四. 综合

 综合后的原理图，见下图：



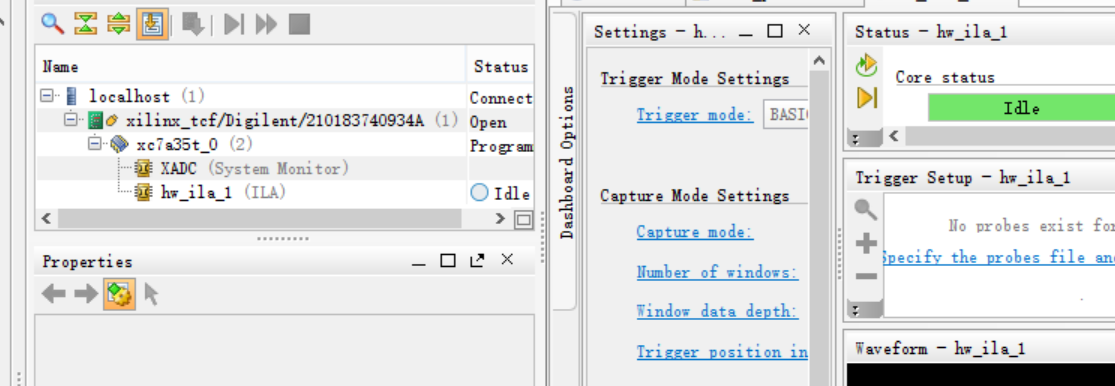
五. 生成比特流

 点击生成比特流，见下图：



六. 下载程序

 （1）与硬件相连

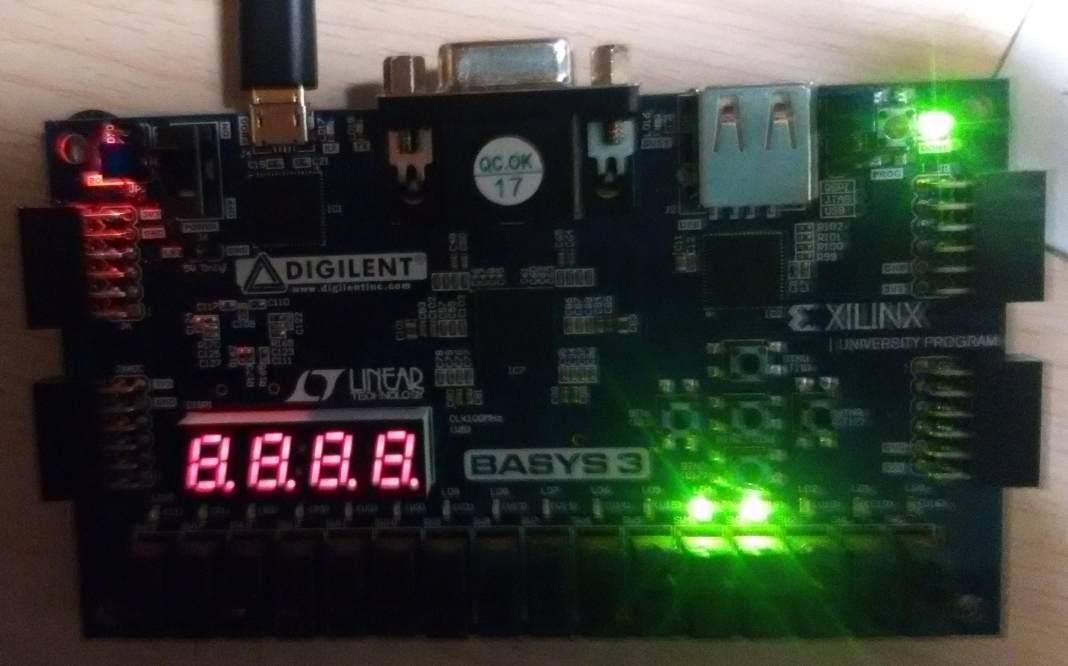


 （2）下载程序

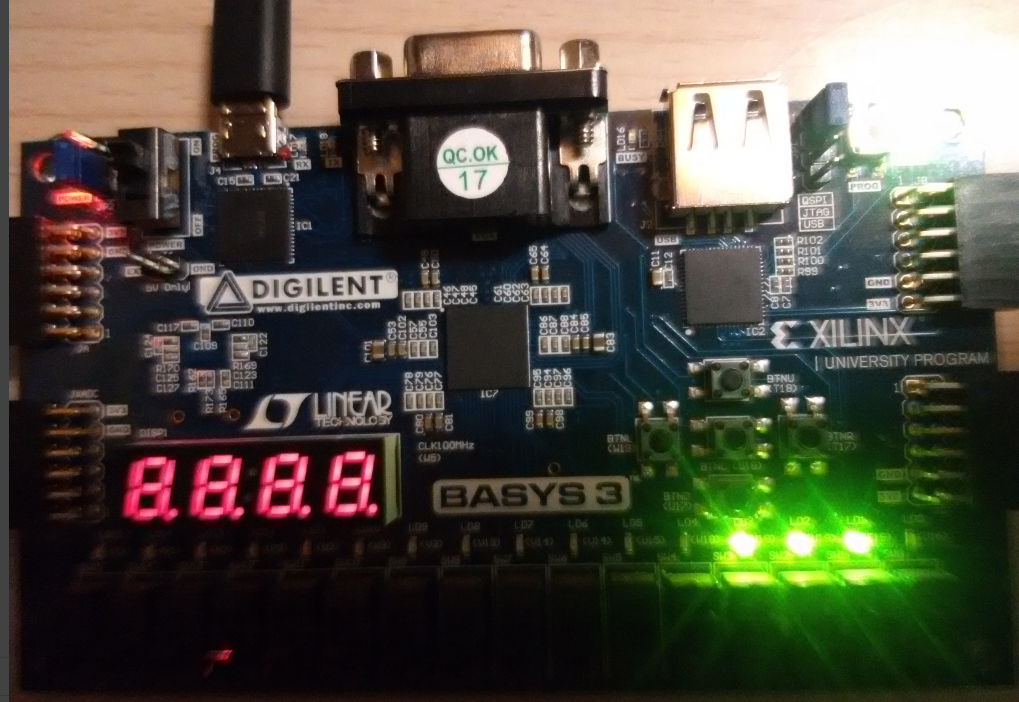
blob.png七. 实物演示

 两个四位数分别为sw[7:4]和sw[3:0]

 （1）当sw[7:4]=4'b1011, sw[3:0]=4'b1101时，计算结果为5'b11000



 (2)当sw[7:4]=4'b0011, sw[3:0]=4'b1011时，计算结果为5'b01110



**附加实验：**

1. **在完成上述实验基础上实现4位超前进位加法器设计并仿真实现**

**超前进位加法器参考**

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

//////////////////////////////////////////////////////////////////////////////////

module chaoadder(A,B,C0,C4,F);

input[3:0] A,B;

input C0;

output[3:0] F;

output C4;

wire[3:0] A,B;

reg [3:0] F,G,P;

reg C4;

reg C1,C2,C3;

always@(\*)

begin

G[0]=A[0]&B[0];

G[1]=A[1]&B[1];

G[2]=A[2]&B[2];

G[3]=A[3]&B[3];

P[0]=A[0]|B[0];

P[1]=A[1]|B[1];

P[2]=A[2]|B[2];

P[3]=A[3]|B[3];

C1=G[0]|(P[0]&C0);

C2=G[1]|(P[1]&C1);

C3=G[2]|(P[2]&C2);

C4=G[3]|(P[3]&C3);

F[0]=A[0]^B[0]^C0;

F[1]=A[1]^B[1]^C1;

F[2]=A[2]^B[2]^C2;

F[3]=A[3]^B[3]^C3;

end

endmodule

仿真实现：

`timescale 1ns / 1ps

module simadder(

);

// Inputs

reg [3:0] A;

reg [3:0] B;

reg C0;

// Outputs

wire C4;

wire [3:0] F;

// Instantiate the Unit Under Test (UUT)

chaoadder uut

(.A(A),

.B(B),

.C0(C0),

.C4(C4),

.F(F)

);

initial begin

// Initialize Inputs

C0 = 0;

// Wait 100 ns for global reset to finish

#100;

// Add stimulus here

begin A = 'B1100;B='B1011;C0 = 0;end

#100;

begin A = 'B1011;B='B0010;C0 = 1;end

#100;

begin A = 'B1011;B='B1101;C0 = 0;end

#100;

begin A = 'B1010;B='B0010;C0 = 1;end

#100;

A = 'B0111;B='B1000;C0= 0;

#100;

begin A = 'B0011;B='B0100;C0 = 1;end

#100;

begin A = 'B1001;B='B0001;C0 = 0;end

#100;

end

endmodule

**2. 尝试编写8位超前进位加法器设计实验并仿真实现**

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[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];

assign

p[0] = a[0] | b[0],

p[1] = a[1] | b[1],

  p[2] = a[2] | b[2],

p[3] = a[3] | b[3],

  p[4] = a[4] | b[4],

p[5] = a[5] | b[5],

p[6] = a[6] | b[6],

p[7] = a[7] | b[7];

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] ） | ( 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

**注：**

reg 寄存器型  
wire 线型  
wire 赋值用 assign 使用符号 =   
例如：wire a；   
assign a = 1；  
reg 赋值 必须在 时序逻辑块中，使用符号 =   
例如： reg b；  
always（posedge clk）  
b = 1；  
可以想象 ：  
wire 一根线，没有存储能力，即时改变值；  
reg 一个小房子，能存储上一次值，需要在clk的驱动下改变值。

输入不能设置为寄存器

|  |  |
| --- | --- |
|  |  |