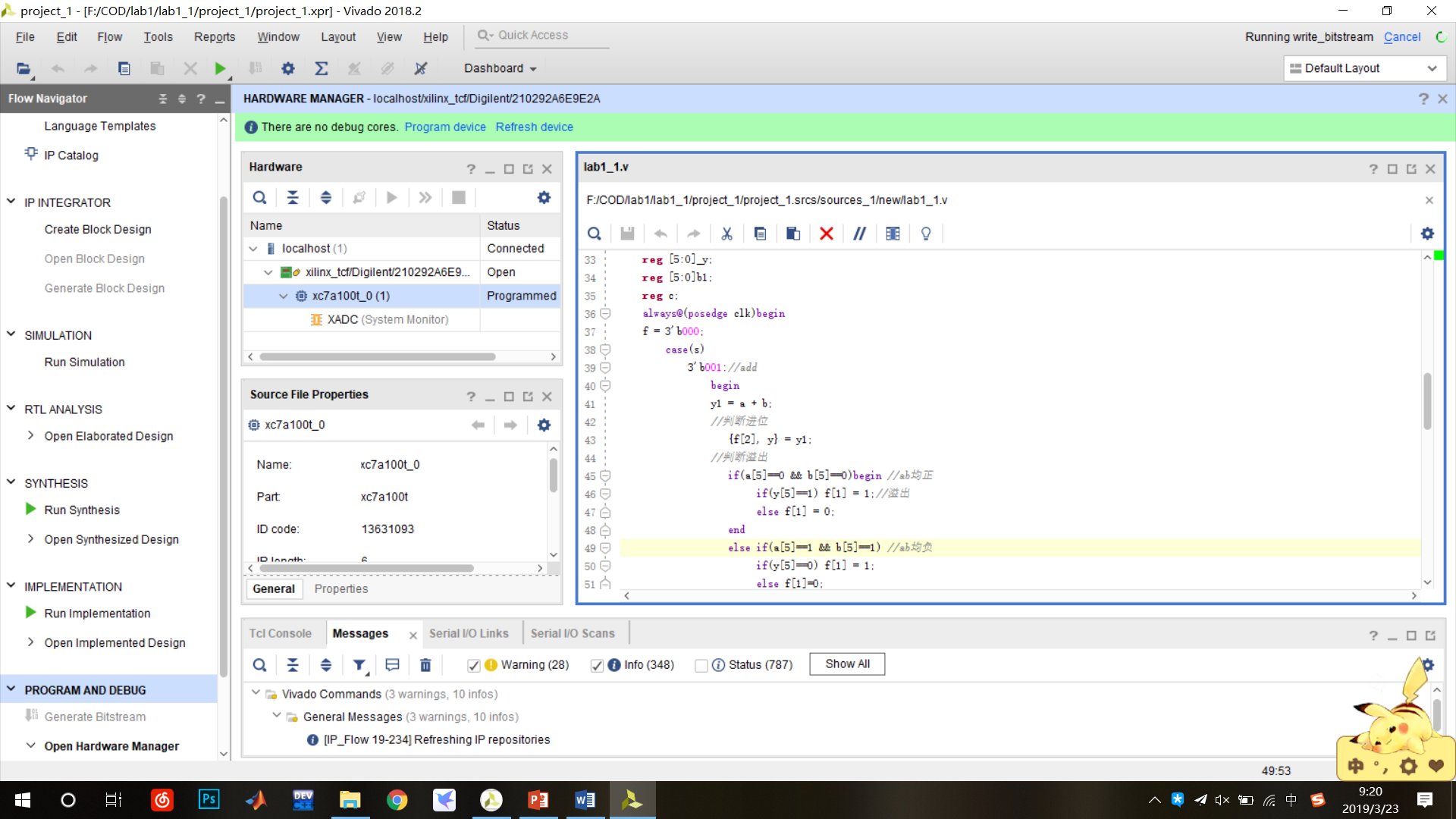
Lab1\_运算器与寄存器实验报告

PB17111623 范睿

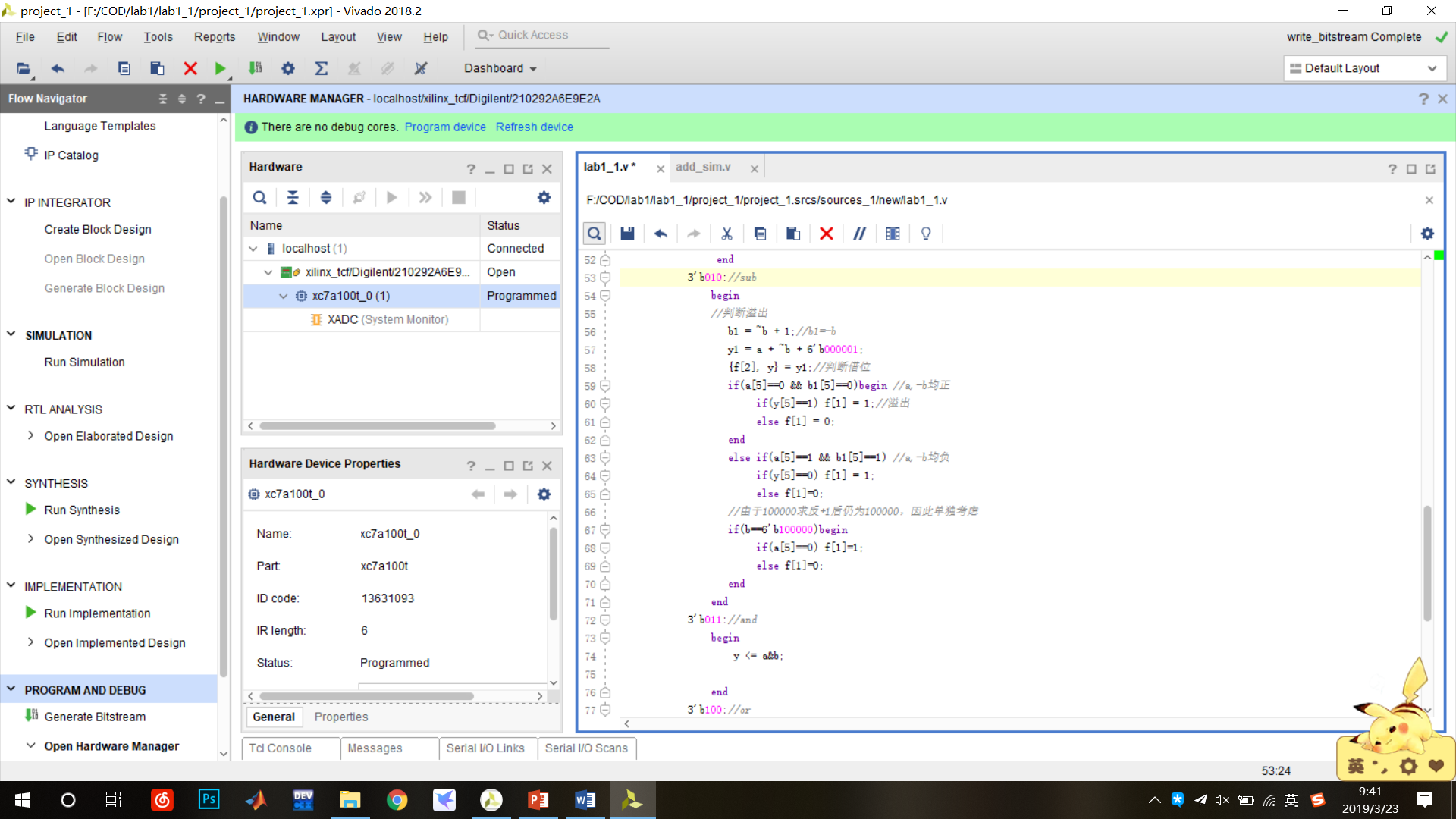
1. **逻辑设计与核心代码**
2. **运算实现**
   * 1. 加法：verilog加
     2. 减法：减数求反加一后利用verilog加
     3. 与：verilog与
     4. 或：verilog或
     5. 非：verilog非
     6. 异或：verilog异或
3. **进位/借位判断：**
   1. 进位：



y1的位数比a与b多1，若y1最高位为1，说明产生进位，将f[2]赋值为1；若y1

最高位为0，说明不产生进位，则将f[2]赋值为0。

* 1. 借位：

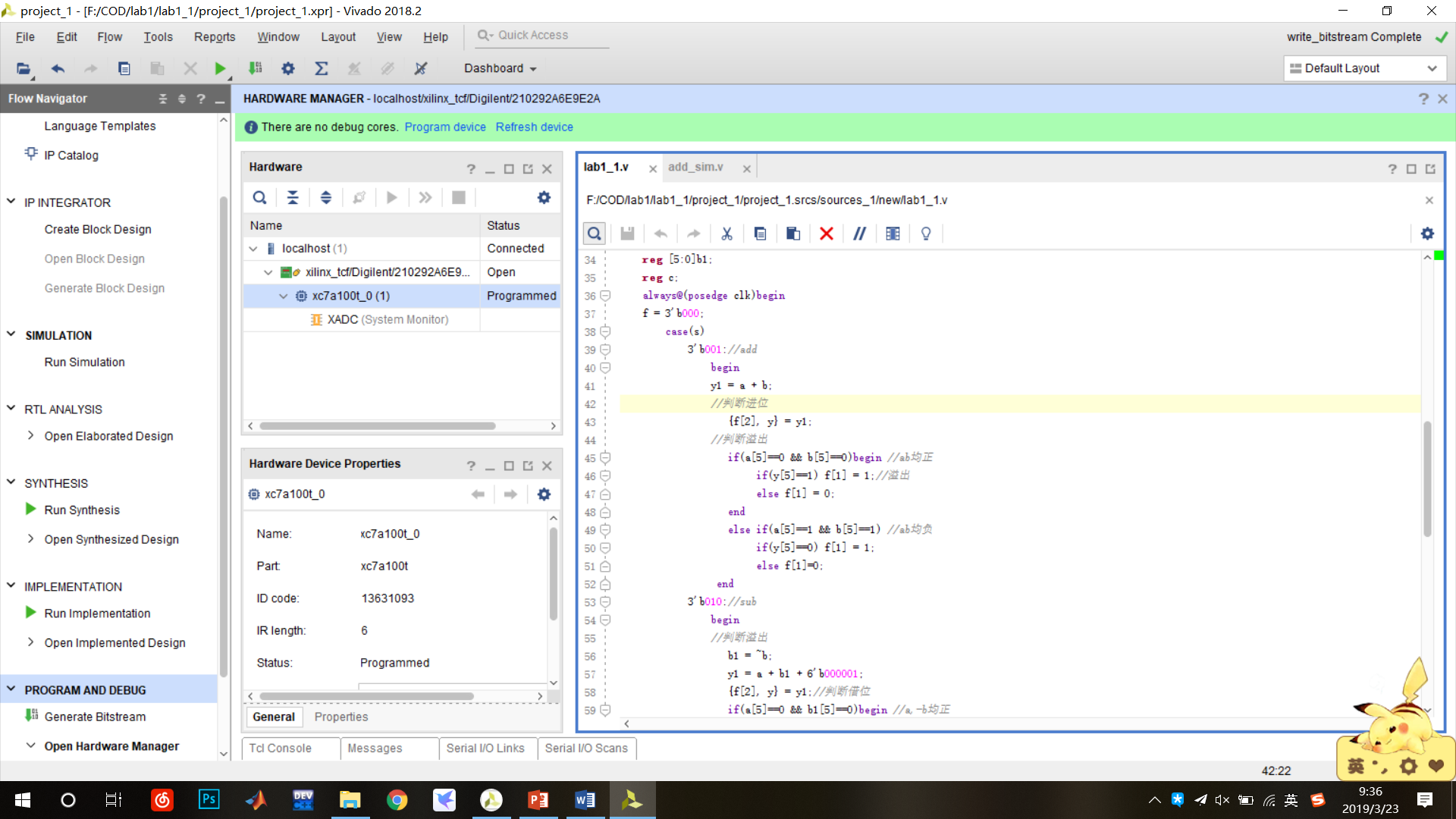


将减数求反。

y1的位数比a与b多1。将y1赋值为a与（-b）的和。

若y1最高位为1，说明产生借位，将f[2]赋值为1；若y1最高位为0，说明产生借位，将f[2]赋值为0。

1. **溢出判断**
   1. 加法判断溢出：

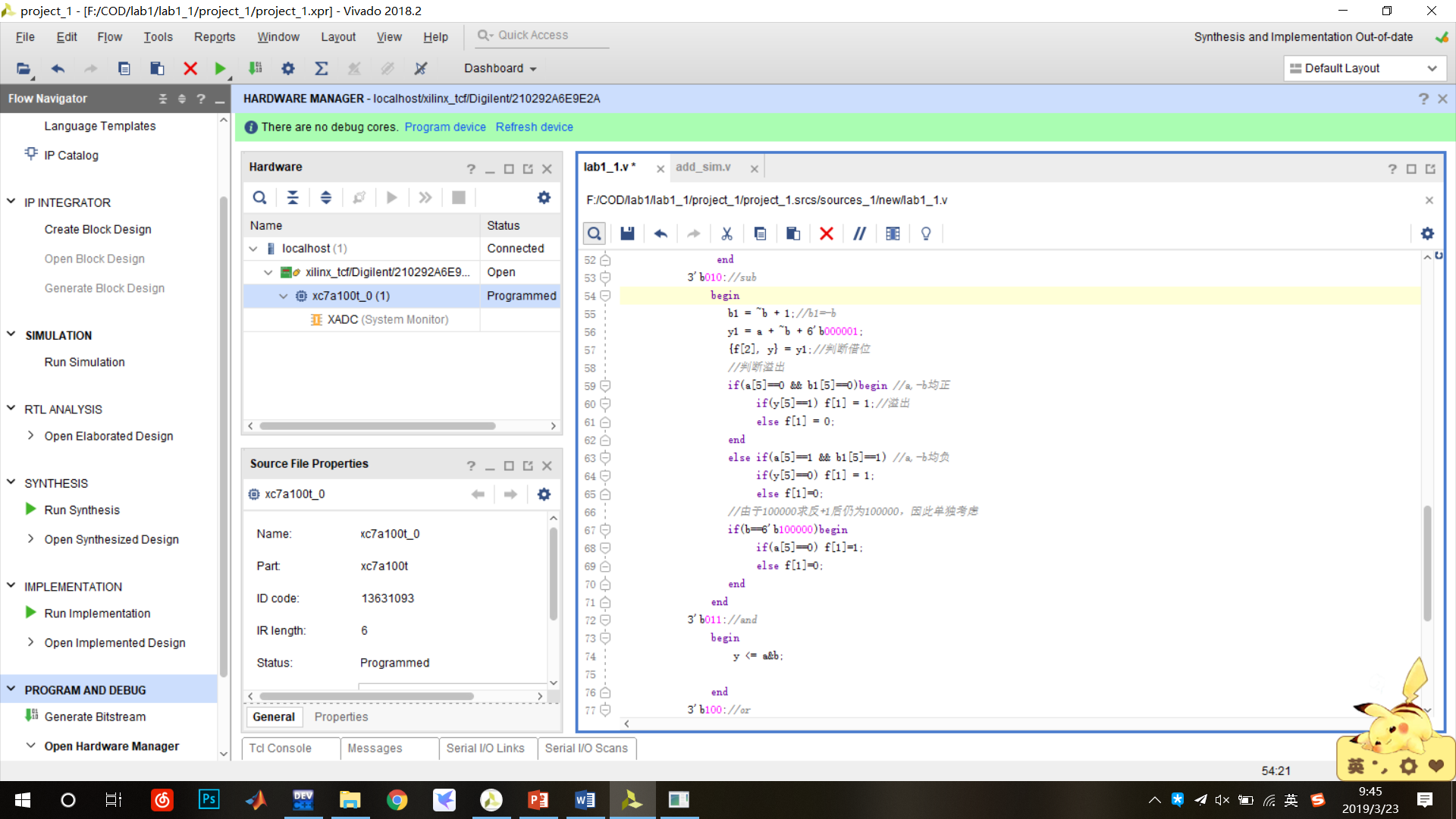


加法时，只有a与b均为正或a与b均为负时才有机会产生溢出。

若ab均正，则当结果y最高位为1时，即y为负时产生溢出。

若ab均负，则当结果y最高位为0时，即y为正时产生溢出。

* 1. 减法判断溢出：



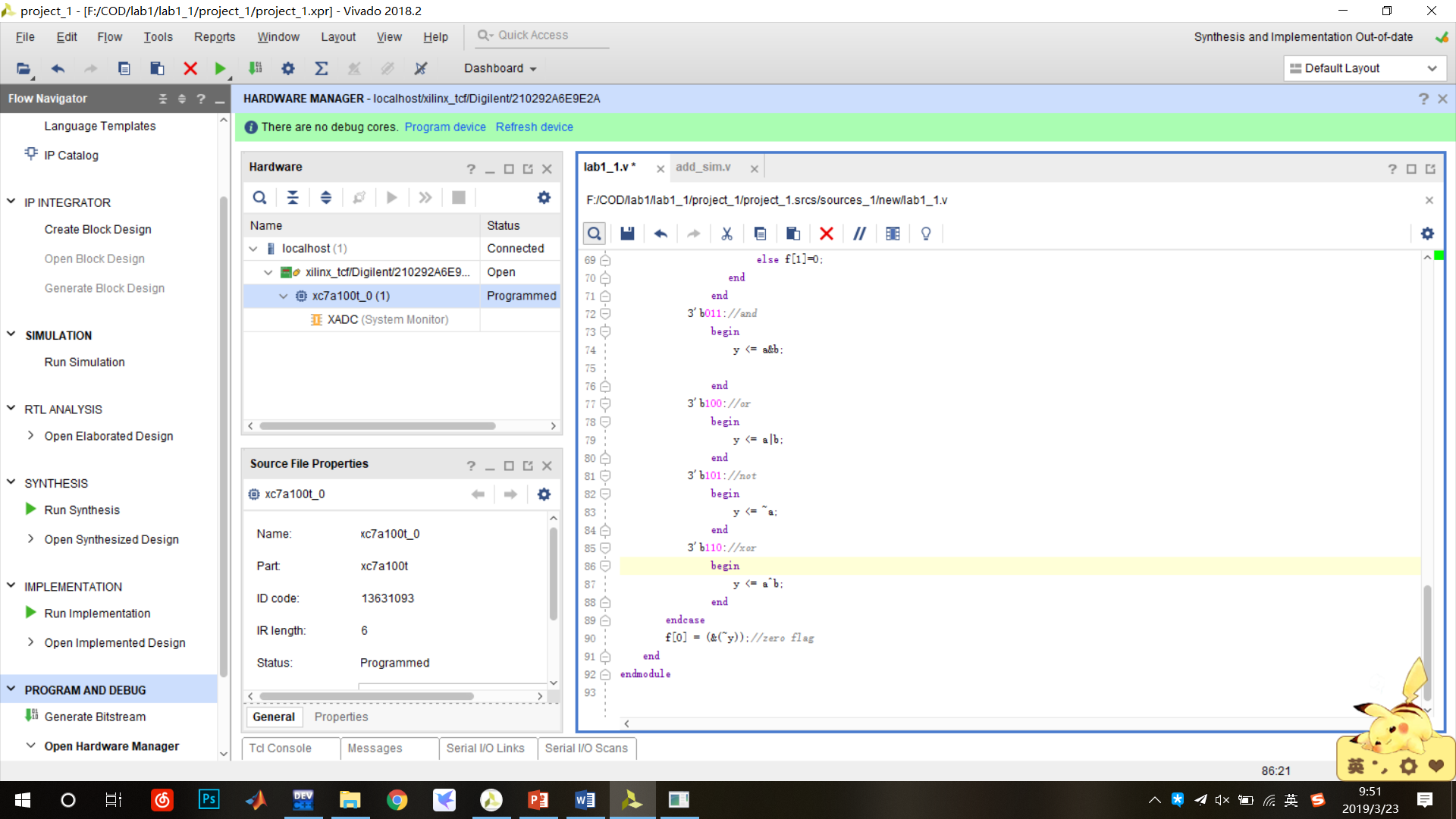
减法时，只有在a与-b均为正数或负数时才有机会产生溢出。

若a，-b均正，则当结果y最高位为1时，即y为负时产生溢出。

若a，-b均负，则当结果y最高位为0时，即y为正时产生溢出。

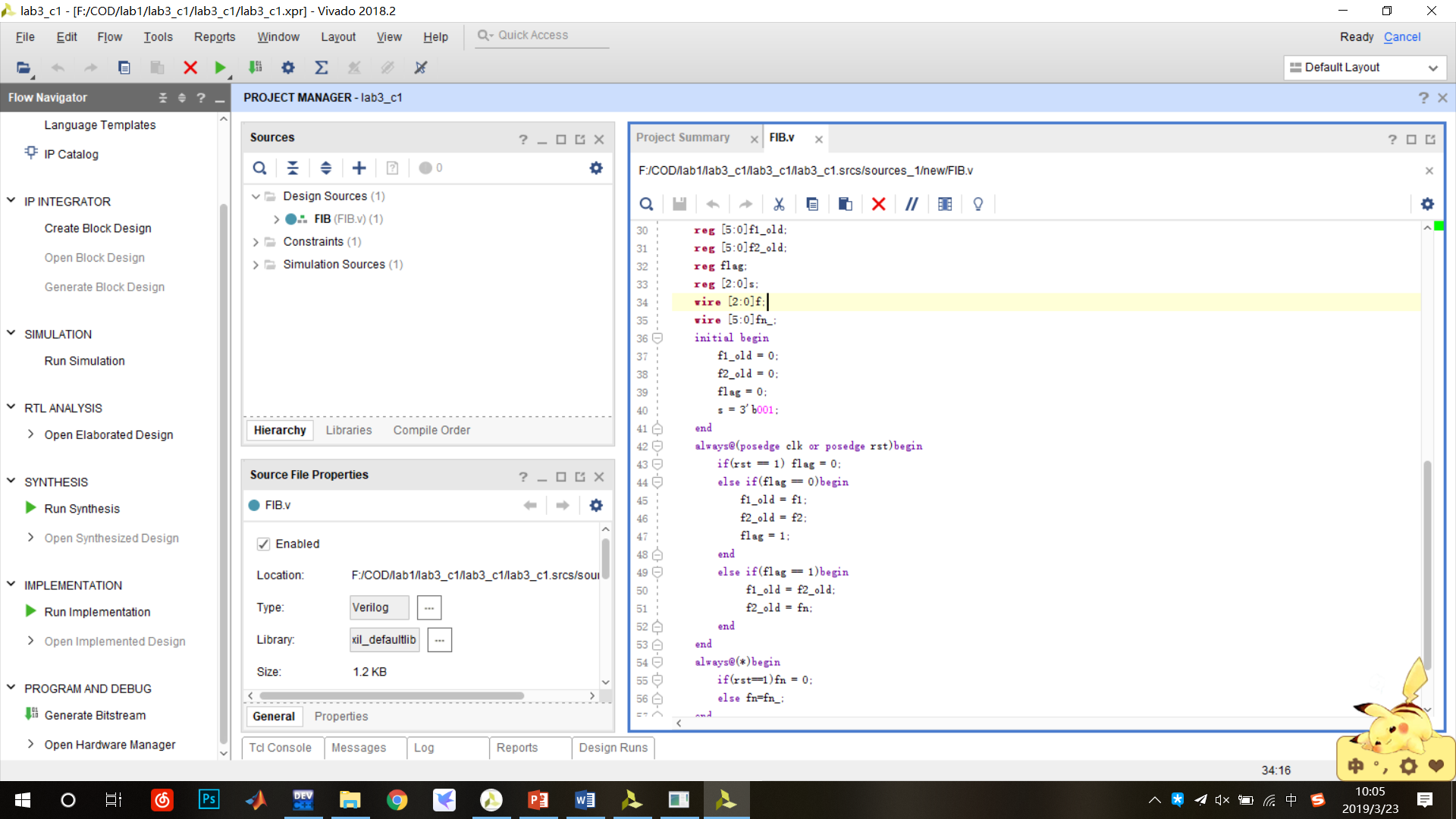
另外需要将单独讨论减数为100000，即b=-32时的情况。因为将100000求反加一后仍为100000，达不到取反的效果。所以若减数为100000，当被减数a为正数时一定会溢出，f[1]置为1。

1. **零标志判断**



若y为000000，零标志为1。

1. **斐波那契数列实现**
   1. 初始化



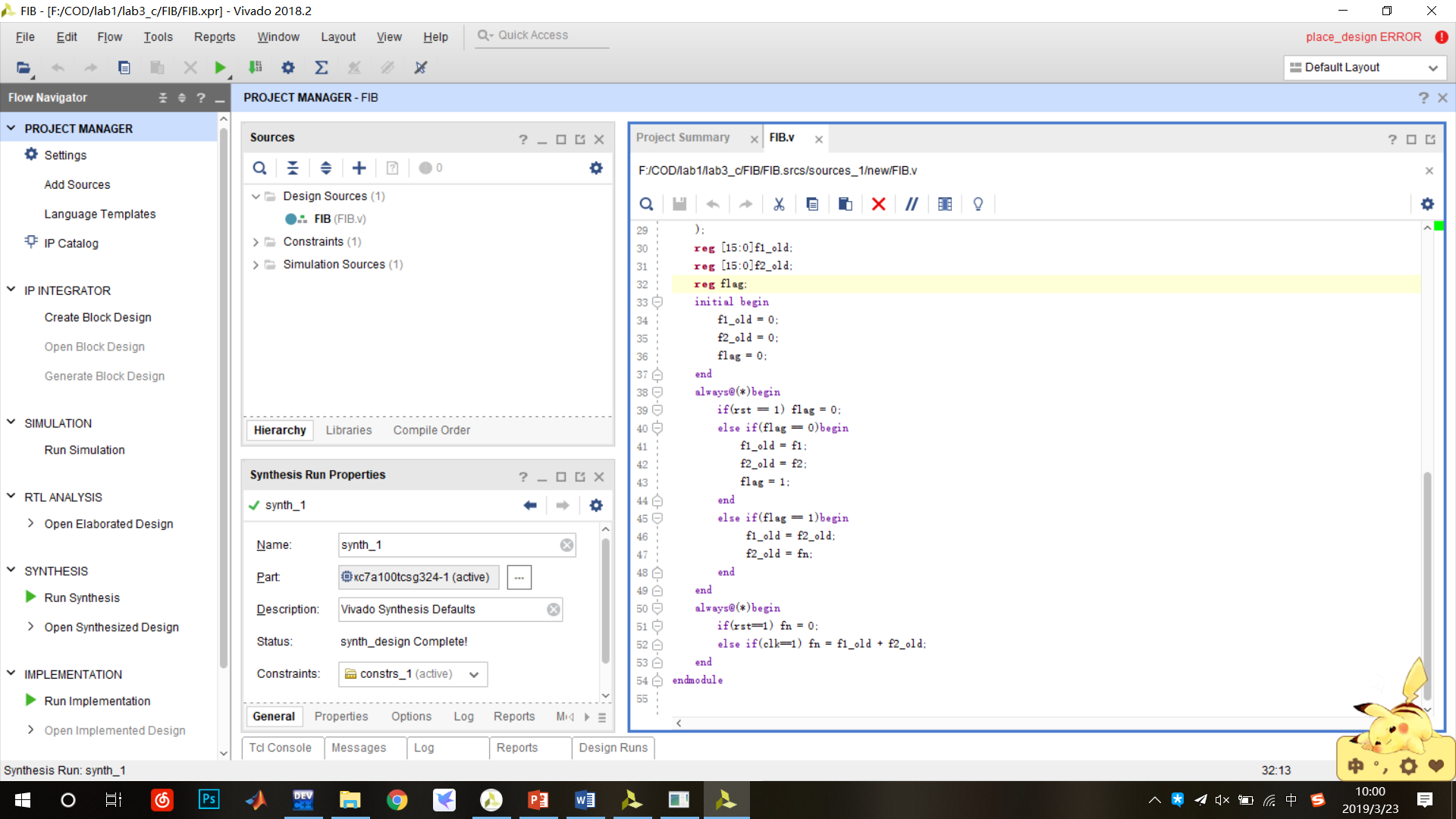
将f1\_old，f2\_old, flag均初始化为零。

f1\_old,f2\_old为斐波那契数列中任意连续三个值的前两个值。

flag用来判断当前需要将f1\_old和f2\_old循环更新还是更新为输入值。

s为输入给ALU的参数，赋值为1表示让ALU做加法操作。

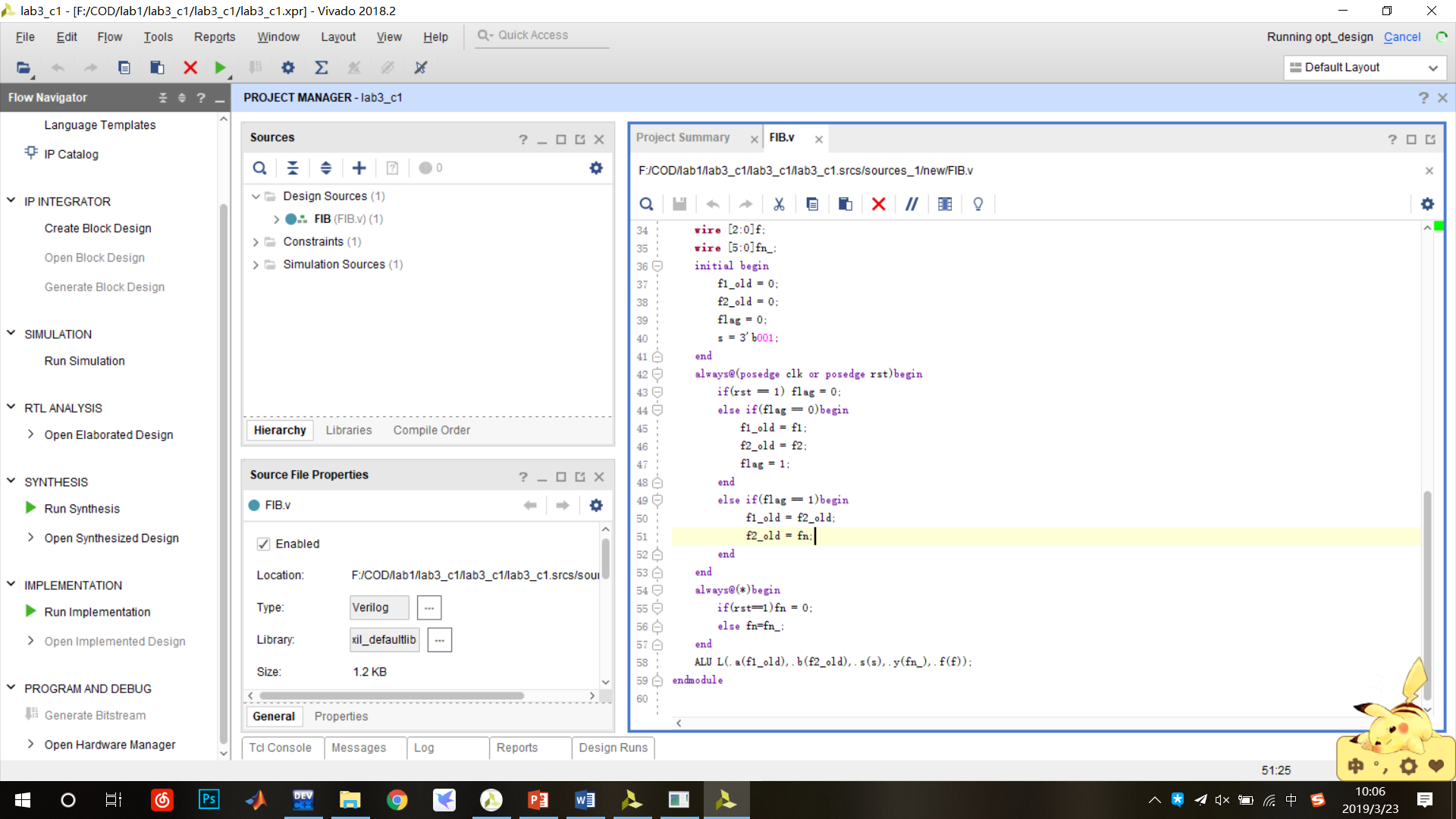
* 1. 计算操作数



若rst有效（为0），将flag置为0，这时两操作数将被置为输入值，flag置为1。

若rst无效（为1），将f1\_old置为f2\_old, f2\_old置为fn。（fn为上一次的计算结果）

* 1. 计算结果



若rst有效（为0），将结果fn置为0。

若rst无效（为1），将结果fn置为ALU的计算结果fn\_。

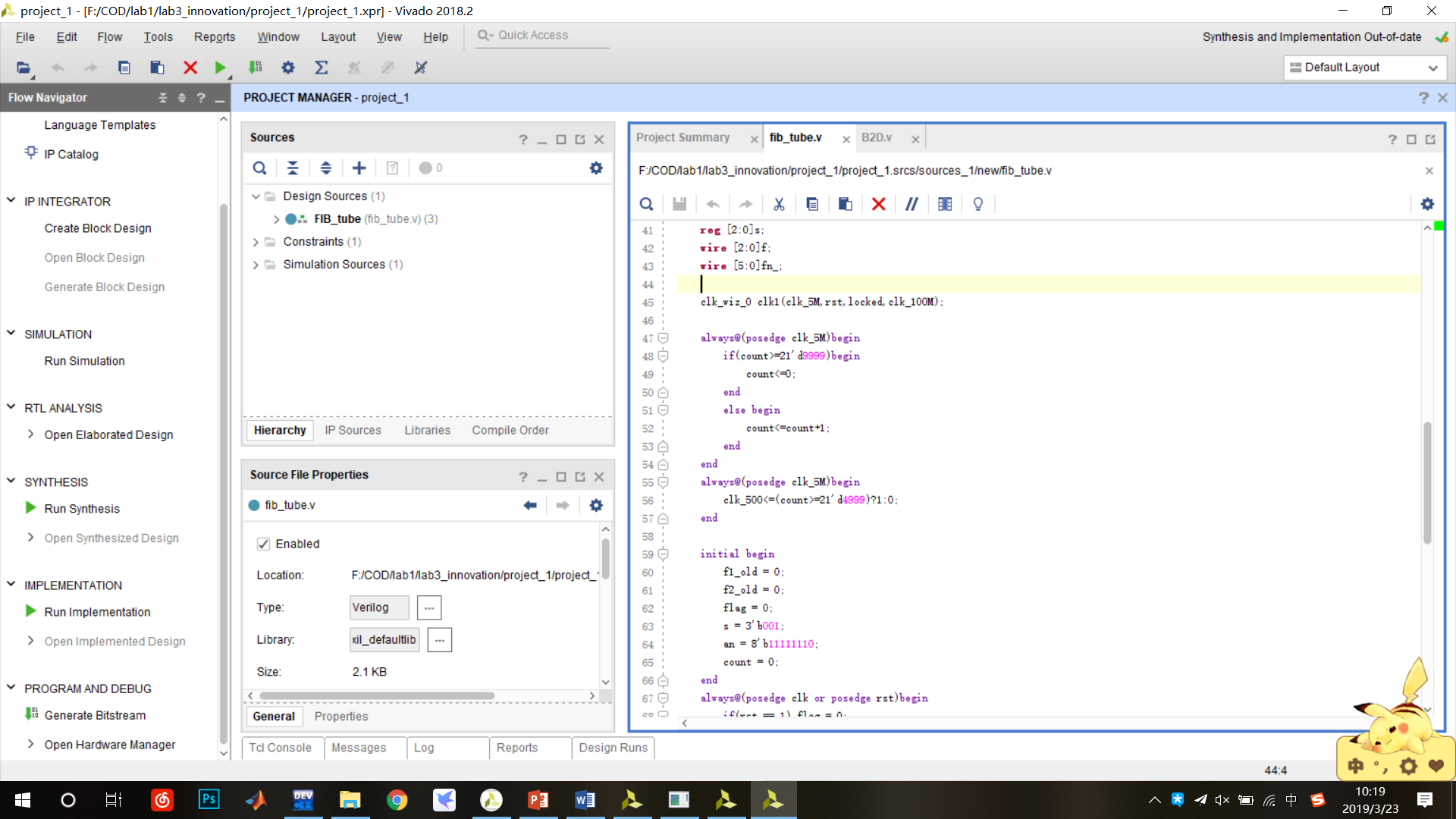
对ALU的模块调用为：将f1\_old和f2\_old作为两操作数的输入，s为控制输入，fn\_为结果输出，f为进位溢出和零标志判断。

1. **数码管对斐波那契数列的实现**

a) 斐波那契数列实现

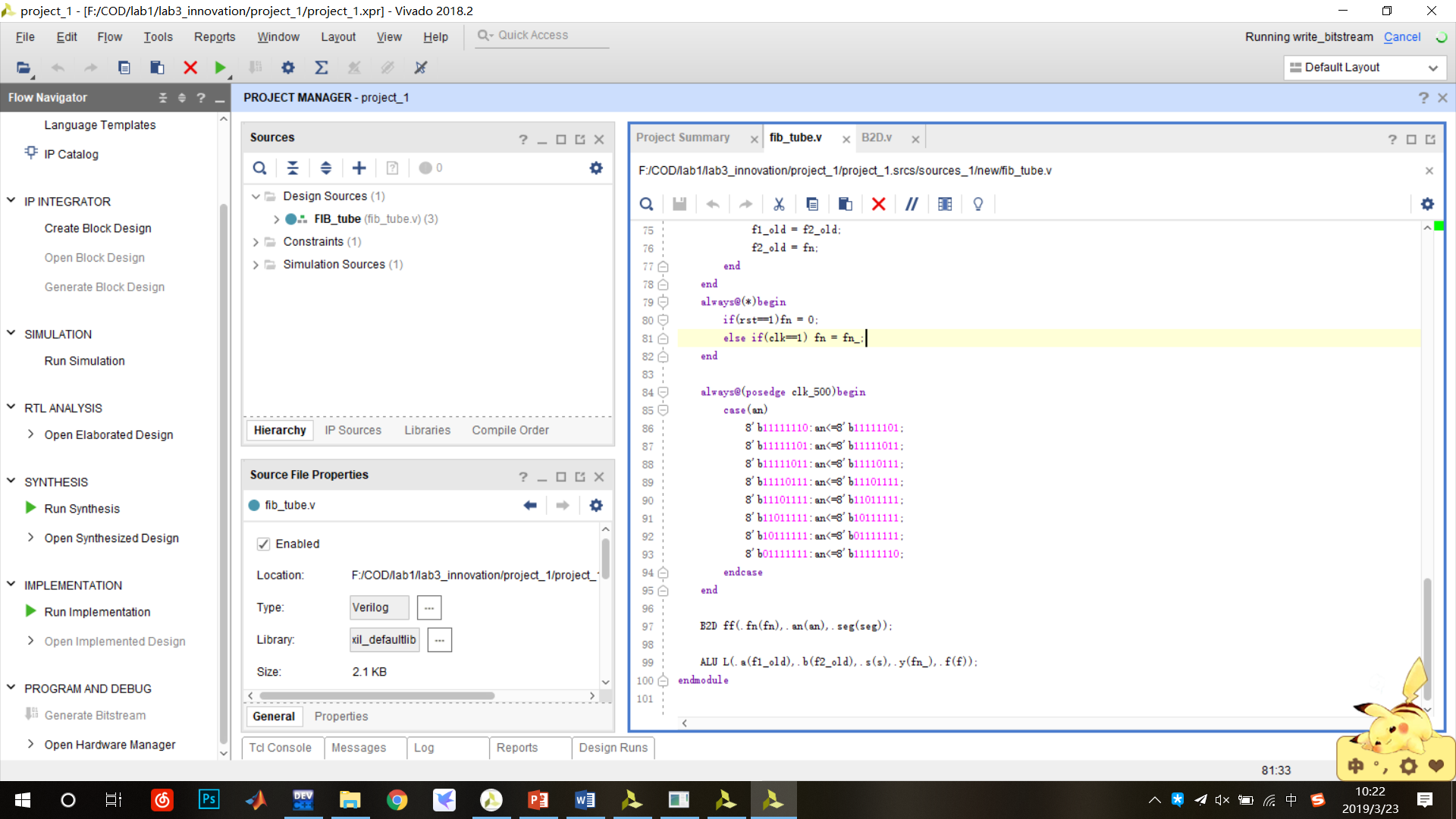
同前

b) 时钟分频



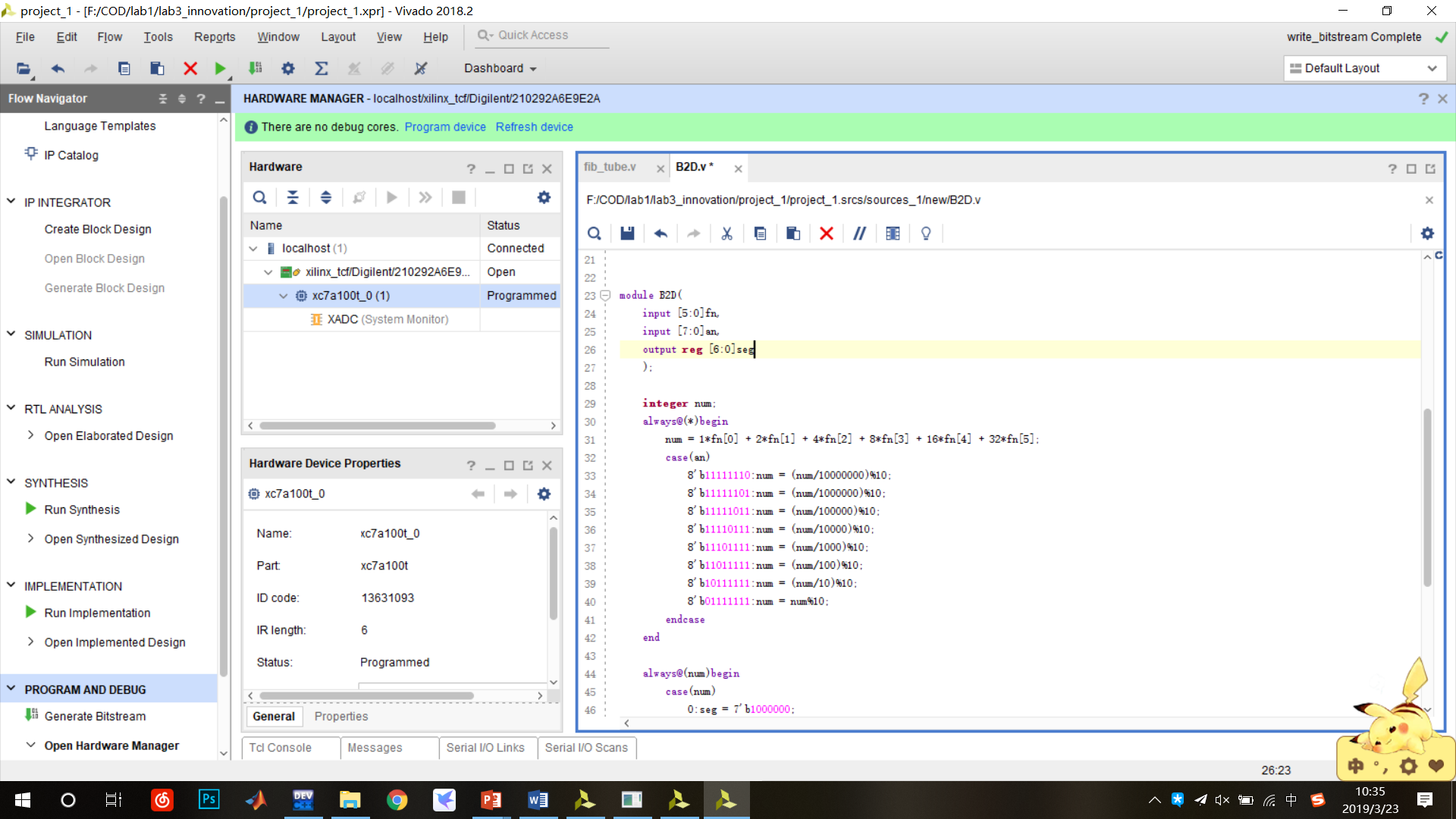
调用时钟模块产生5MHz的时钟，再利用分频模块产生500Hz的时钟进行扫描。

c) 更新位选信号an



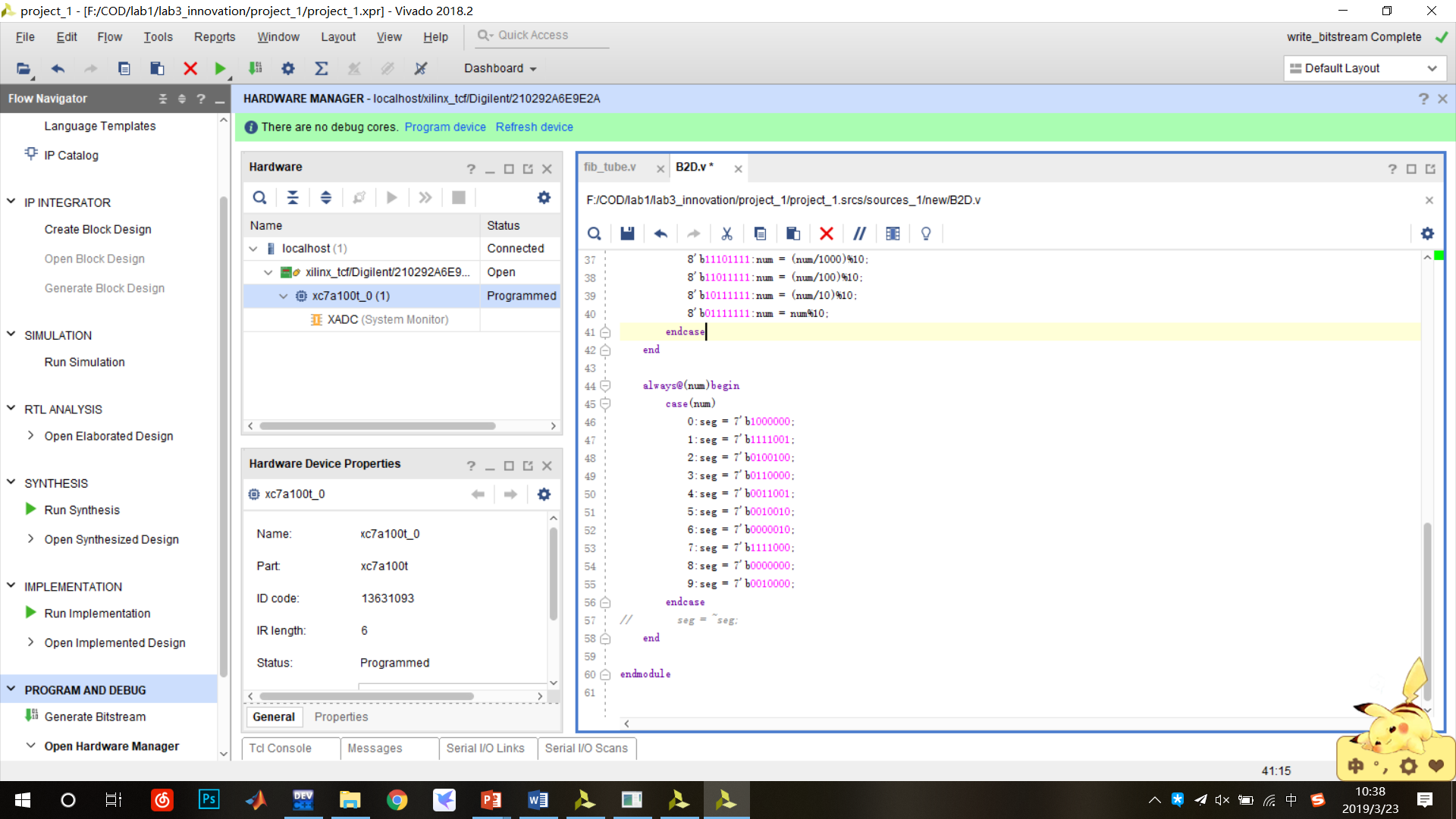
每当500Hz时钟上升沿到达时，跟新an的值。为0的位表示当前扫描到第几个位置。（an赋初值为8’b1111110）

d) 更新段选信号seg



fn为计算好的结果的输入，num为fn转为十进制的值，为整型。

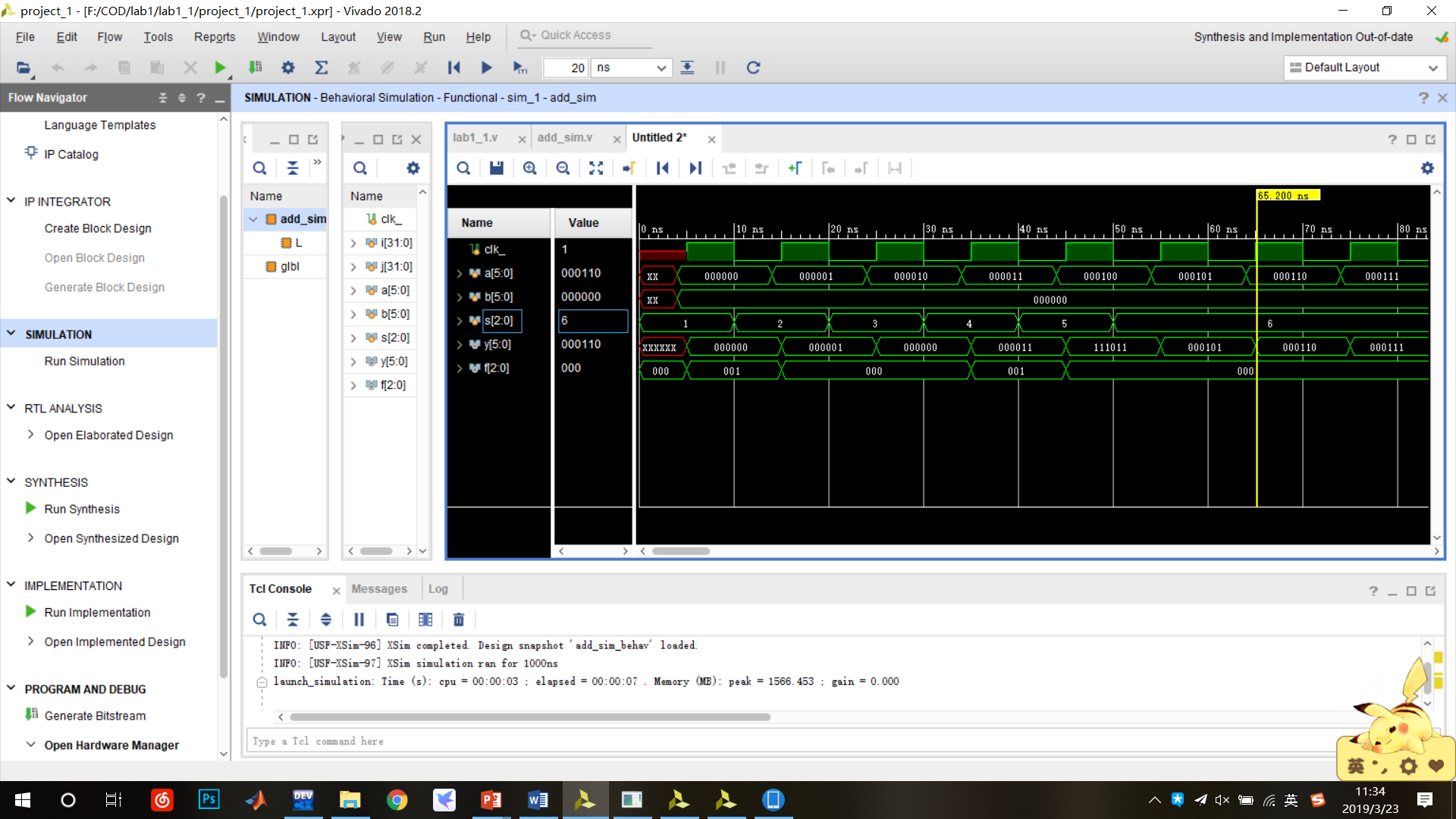
根据an位选的不同，选择将num的那个位置上的值提取出来。



根据num的不同，将段选信号seg赋值为不同值以显示正确数字。

1. **仿真与下载结果**
2. **ALU**

仿真：



S为1,2,3,4,5,6分别对应加，减，与，或，非，异或。

下载：

加法：



输入为011111和011111。Led灯左端显示溢出。

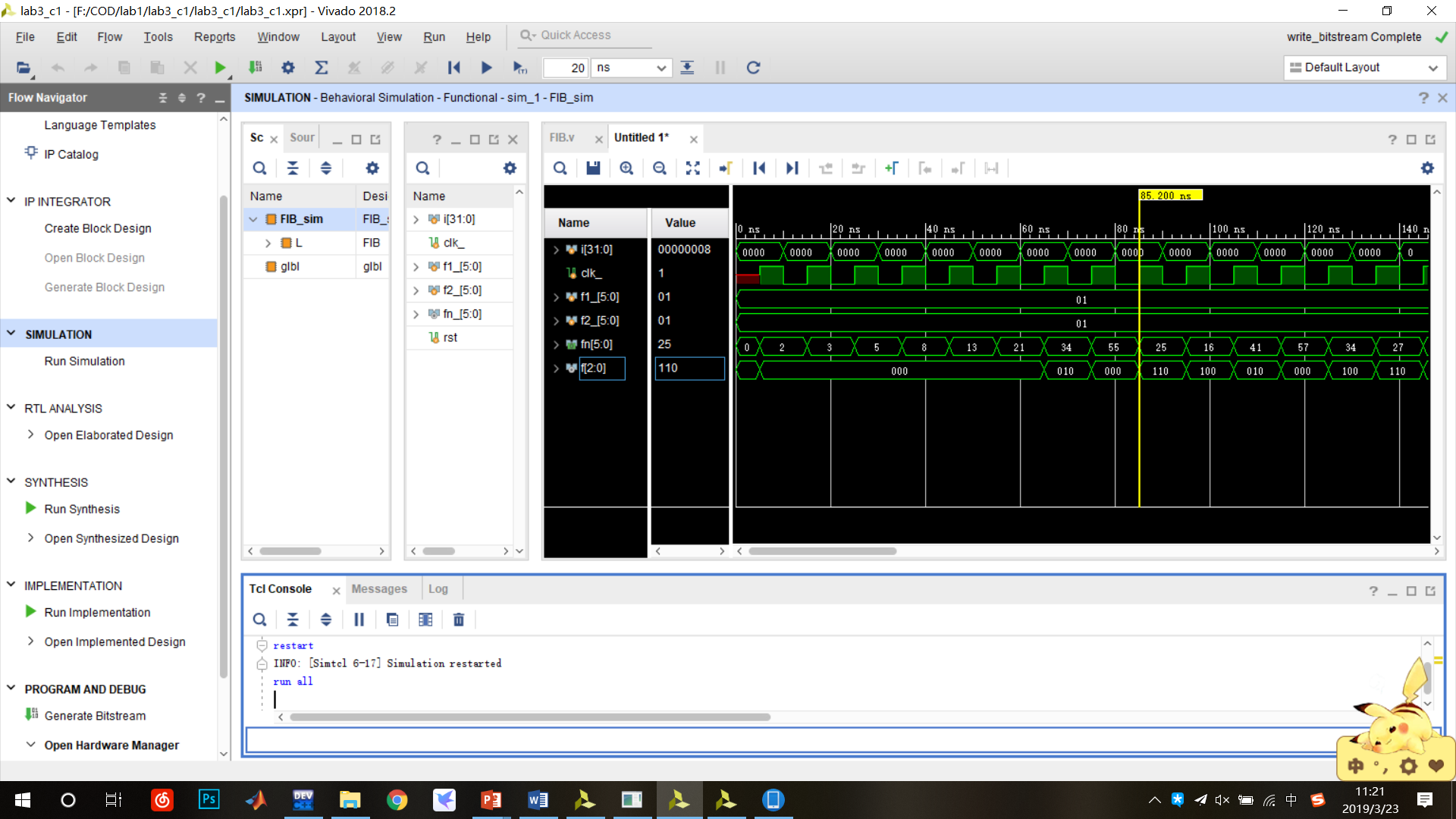
减法



输入为000000和000000。Led灯左端显示借位（左一）和零标志（左三）。

1. **斐波那契**

仿真：



黄线之后可以由最后一行的f值看出已经溢出。

下载：

输入为1和1



1. **数码管实现斐波那契**

输入为1和1



（灭掉的数码管是因为照相机可以分辨出来当前的扫描频率，500Hz，但人眼分辨不出）

1. **结果分析和实验总结**

从斐波那契的实验结果和仿真结果来看，ALU的运算结果没有问题。

但我觉得我ALU的实现有点繁琐。一个是溢出判断。不是通过最高位和次高位的进位判断的，而是通过结果和操作数的数值，在后期cpu的实现上可能会拖慢速度。另一个是灵活性不太好。我这个ALU只能运算六位及以下的操作数，如果要运算六位以上的数需要大改，不好。

1. **完整代码**

**1.ALU**

module ALU(

input clk,

input [5:0] a,

input [5:0]b,

input [2:0]s,

output reg [5:0]y,

output reg [2:0]f//[进位/借位，溢出，零标志]

);

initial begin

f = 3'b000;

end

reg [6:0]y1;

reg [5:0]\_y;

reg [5:0]b1;

reg c;

always@(posedge clk)begin

f = 3'b000;

case(s)

3'b001://add

begin

y1 = a + b;

//判断进位

{f[2], y} = y1;

//判断溢出

if(a[5]==0 && b[5]==0)begin //ab均正

if(y[5]==1) f[1] = 1;//溢出

else f[1] = 0;

end

else if(a[5]==1 && b[5]==1) //ab均负

if(y[5]==0) f[1] = 1;

else f[1]=0;

end

3'b010://sub

begin

b1 = ~b + 1;//b1=-b

y1 = a + ~b + 6'b000001;

{f[2], y} = y1;//判断借位

//判断溢出

if(a[5]==0 && b1[5]==0)begin //a,-b均正

if(y[5]==1) f[1] = 1;//溢出

else f[1] = 0;

end

else if(a[5]==1 && b1[5]==1) //a,-b均负

if(y[5]==0) f[1] = 1;

else f[1]=0;

//由于100000求反+1后仍为100000，因此单独考虑

if(b==6'b100000)begin

if(a[5]==0) f[1]=1;

else f[1]=0;

end

end

3'b011://and

begin y <= a&b; end

3'b100://or

begin y <= a|b; end

3'b101://not

begin y <= ~a; end

3'b110://xor

begin y <= a^b; end

endcase

f[0] = (&(~y));//zero flag

end

endmodule

**ALU仿真**

module add\_sim(

);

reg clk\_;

integer i,j;

reg [5:0]a;

reg [5:0]b;

reg [2:0]s;

wire [5:0]y;

wire [2:0]f;

ALU L(.clk(clk\_), .a(a), .b(b), .s(s), .y(y), .f(f));

initial begin

for(i=0;i<62;i=i+1) begin

#5 assign clk\_=1;

#5 assign clk\_=0;

end

end

initial begin

#4 a = 6'b000000;

for(j=0;j<62;j=j+1) begin

#10 a <= a + 6'b000001;

end

end

initial begin

#4 b = 6'b000000;

// #320 b = 6'b111010;

end

initial begin

s = 3'b001;

#10 s = 3'b010;

#10 s = 3'b011;

#10 s = 3'b100;

#10 s = 3'b101;

#10 s = 6'b110;

end

endmodule

**2. 斐波那契**

module FIB(

input [5:0]f1,

input [5:0]f2,

input clk,

input rst,

output reg [5:0]fn

);

reg [5:0]f1\_old;

reg [5:0]f2\_old;

reg flag;

reg [2:0]s;

wire [2:0]f;

wire [5:0]fn\_;

initial begin

f1\_old = 0;

f2\_old = 0;

flag = 0;

s = 3'b001;

end

always@(posedge clk or posedge rst)begin

if(rst == 1) flag = 0;

else if(flag == 0)begin

f1\_old = f1;

f2\_old = f2;

flag = 1;

end

else if(flag == 1)begin

f1\_old = f2\_old;

f2\_old = fn;

end

end

always@(\*)begin

if(rst==1)fn = 0;

else fn=fn\_;

end

ALU L(.a(f1\_old),.b(f2\_old),.s(s),.y(fn\_),.f(f));

endmodule

**斐波那契仿真**

module FIB\_sim(

);

integer i;

reg clk\_;

reg [5:0]f1\_;

reg [5:0]f2\_;

wire [5:0]fn\_;

reg rst;

FIB L(.f1(f1\_),.f2(f2\_),.clk(clk\_),.fn(fn\_),.rst(rst));

initial begin

for(i=0;i<62;i=i+1) begin

#5 assign clk\_=1;

#5 assign clk\_=0;

end

end

initial begin

f1\_=6'b000001; #500 f1\_=6'b000011;end

initial begin

f2\_=6'b000001;

#500 f2\_=6'b000011;

end

initial begin

rst = 0;

#500 rst = 1;

#5 rst = 0;

end

endmodule

**3.斐波那契数码管**

initial begin

f1\_old = 0;

f2\_old = 0;

flag = 0;

s = 3'b001;

an = 8'b11111110;

count = 0;

end

always@(posedge clk or posedge rst)begin

if(rst == 1) flag = 0;

else if(flag == 0)begin

f1\_old = f1;

f2\_old = f2;

flag = 1;

end

else if(flag == 1)begin

f1\_old = f2\_old;

f2\_old = fn;

end

end

always@(\*)begin

if(rst==1)fn = 0;

else if(clk==1) fn = fn\_;

end

always@(posedge clk\_500)begin

case(an)

8'b11111110:an<=8'b11111101;

8'b11111101:an<=8'b11111011;

8'b11111011:an<=8'b11110111;

8'b11110111:an<=8'b11101111;

8'b11101111:an<=8'b11011111;

8'b11011111:an<=8'b10111111;

8'b10111111:an<=8'b01111111;

8'b01111111:an<=8'b11111110;

endcase

end

B2D ff(.fn(fn),.an(an),.seg(seg));

ALU L(.a(f1\_old),.b(f2\_old),.s(s),.y(fn\_),.f(f));

endmodule

module FIB\_tube(

input [5:0]f1,

input [5:0]f2,

input clk\_100M,

input clk,

input rst,

output [6:0]seg,

output reg [7:0]an

);

wire clk\_5M;

wire locked;

reg [5:0]fn;

reg clk\_500;

reg [21:0]count;

reg [5:0]f1\_old;

reg [5:0]f2\_old;

reg flag;

reg [2:0]s;

wire [2:0]f;

wire [5:0]fn\_;

clk\_wiz\_0 clk1(clk\_5M,rst,locked,clk\_100M);

always@(posedge clk\_5M)begin

if(count>=21'd9999)begin

count<=0;

end

else begin

count<=count+1;

end

end

always@(posedge clk\_5M)begin

clk\_500<=(count>=21'd4999)?1:0;

end

module B2D(

input [5:0]fn,

input [7:0]an,

output reg [6:0]seg

);

integer num;

always@(\*)begin

num = 1\*fn[0] + 2\*fn[1] + 4\*fn[2] + 8\*fn[3] + 16\*fn[4] + 32\*fn[5];

case(an)

8'b11111110:num = (num/10000000)%10;

8'b11111101:num = (num/1000000)%10;

8'b11111011:num = (num/100000)%10;

8'b11110111:num = (num/10000)%10;

8'b11101111:num = (num/1000)%10;

8'b11011111:num = (num/100)%10;

8'b10111111:num = (num/10)%10;

8'b01111111:num = num%10;

endcase

end

always@(num)begin

case(num)

0:seg = 7'b1000000;

1:seg = 7'b1111001;

2:seg = 7'b0100100;

3:seg = 7'b0110000;

4:seg = 7'b0011001;

5:seg = 7'b0010010;

6:seg = 7'b0000010;

7:seg = 7'b1111000;

8:seg = 7'b0000000;

9:seg = 7'b0010000;

endcase

// seg = ~seg;

end

endmodule