# 四位加法器

计 93　王哲凡　2019011200

## 一、实验目的

1. 掌握组合逻辑电路的基本分析方法与设计方法。
2. 理解半加器、全加器、加法器的分析与设计方法。
3. 学会元件例化。
4. 学会利用软件仿真实现对数字电路的验证和分析。

## 二、实验内容与要求

1. 设计半加器，并用半加器构建全加器。
2. 利用全加器构建逐次进位和超前进位的四位加法器，测试并用仿真验证。
3. 利用 HDL 自带的加法实现四位加法器，测试并用仿真验证。
4. 查看三者在 CPLD 中生成的电路，并比较其异同。

## 三、实验代码及简要原理分析

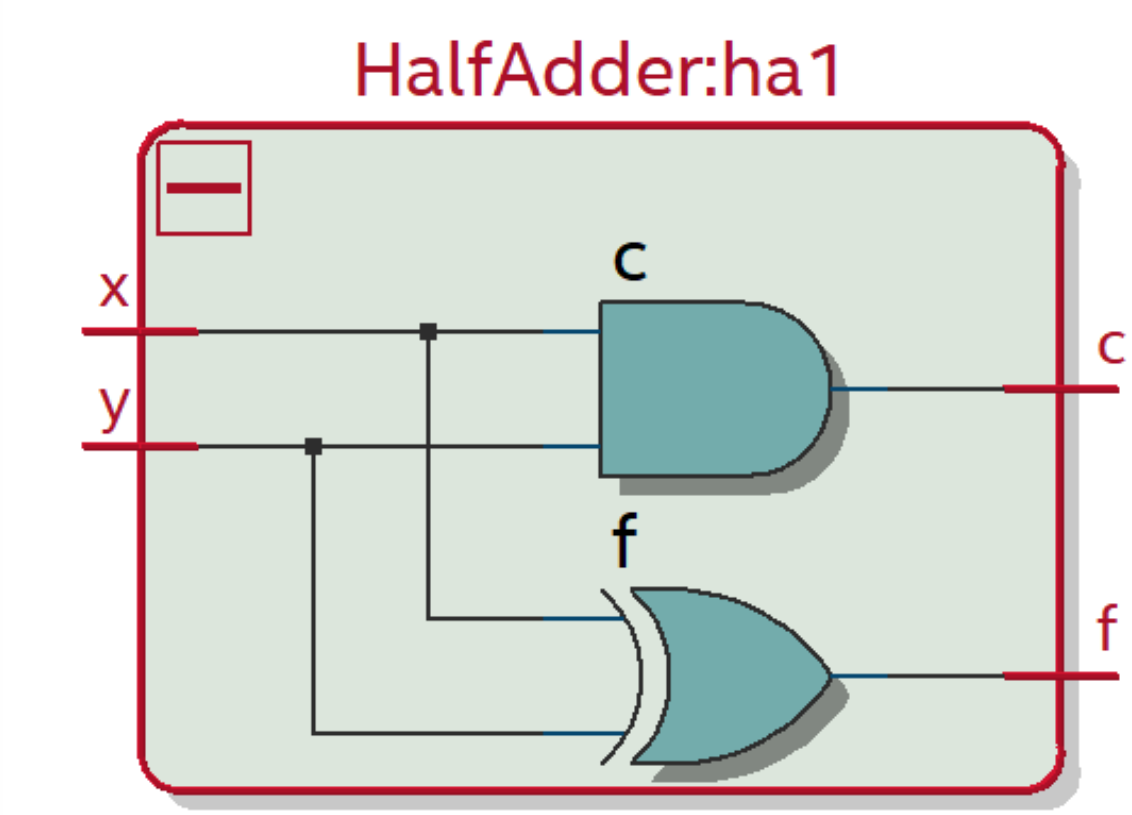
### 1. 半加器及全加器代码及生成电路

半加器中 $f = x \oplus y, c = x \cdot y$。

半加器代码:

```
module HalfAdder(
    input x,
    input y,
    output f,
    output c
);

    assign f = x ^ y;
    assign c = x & y;

endmodule

```

半加器的生成电路如下：



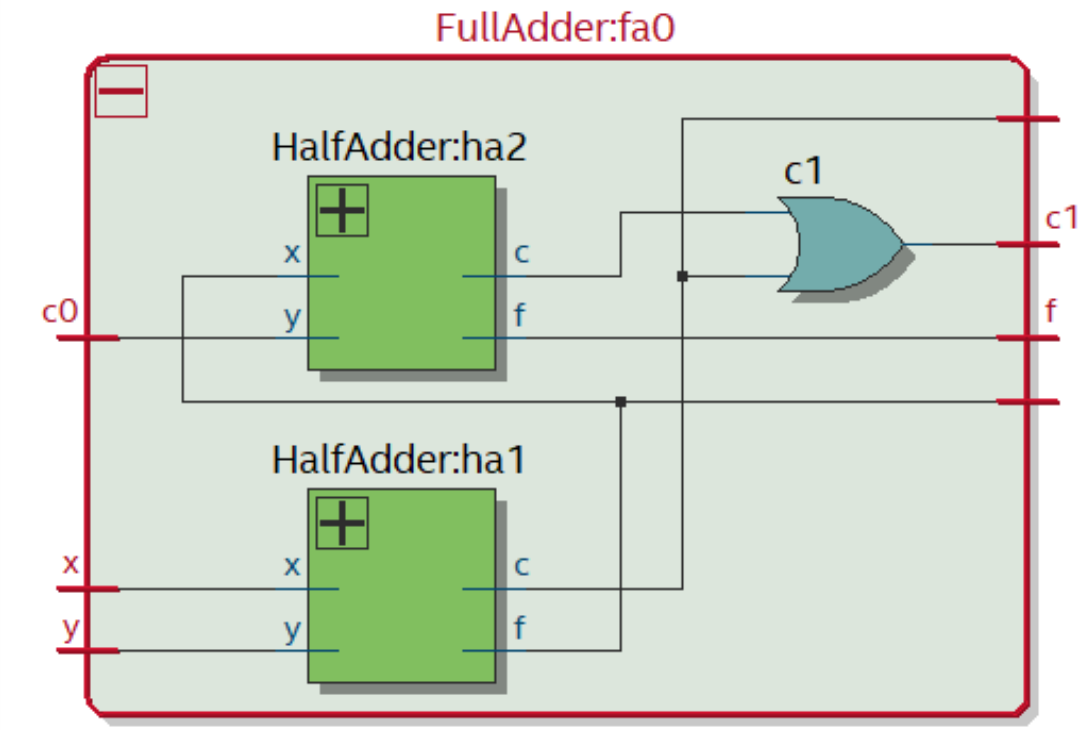

全加器中令 $p=x\oplus y, g=x\cdot y$，则 $f=x\oplus y\oplus c_0=p\oplus c_0$，$c_1=x\cdot y+x\cdot c_0+y\cdot c_0=g+(x\oplus y)\cdot c_0=g+p\cdot c_0$。

全加器代码：

```
module FullAdder(
    input x,
    input y,
    input c0,

```

```
    output f,
    output c1,
    output p,
    output g
);

    wire r;
    HalfAdder ha1(.x(x), .y(y), .f(p), .c(g));
    HalfAdder ha2(.x(p), .y(c0), .f(f), .c(r));
    assign c1 = r | g;

endmodule

```

全加器的生成电路如下：



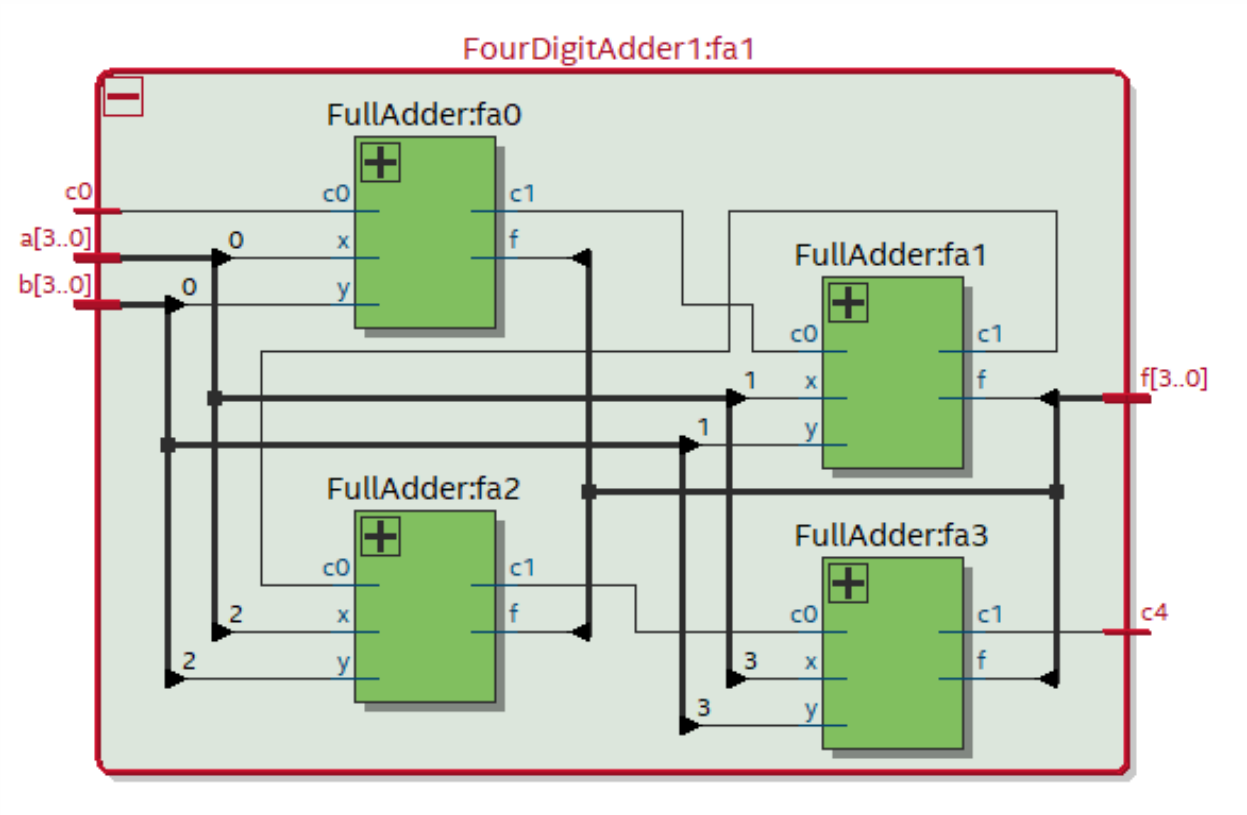

## 2. 逐次进位的四位加法器代码及生成电路

代码如下：

```
module FourDigitAdder1(
    input [3:0] a,
    input [3:0] b,
    input c0,
    output [3:0] f,
    output c4
```

```
);

    wire [2:0] c;

    FullAdder fa0(.x(a[0]), .y(b[0]), .c0(c0), .f(f[0]), .c1(c[0]));
    FullAdder fa1(.x(a[1]), .y(b[1]), .c0(c[0]), .f(f[1]), .c1(c[1]));
    FullAdder fa2(.x(a[2]), .y(b[2]), .c0(c[1]), .f(f[2]), .c1(c[2]));
    FullAdder fa3(.x(a[3]), .y(b[3]), .c0(c[2]), .f(f[3]), .c1(c4));

endmodule

```

生成电路如下：




## 3. 超前进位的四位加法器代码及生成电路

利用全加器中的 $p, g$ 可以得到：

$$
\begin{aligned}
c_1 &= g_0 + p_0 c_0 \\
c_2 &= g_1 + p_1 g_0 + p_1 p_0 c_0 \\
c_3 &= g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 c_0 \\
c_4 &= g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0 + p_3 p_2 p_1 p_0 c_0
\end{aligned}
$$

加法器代码如下：

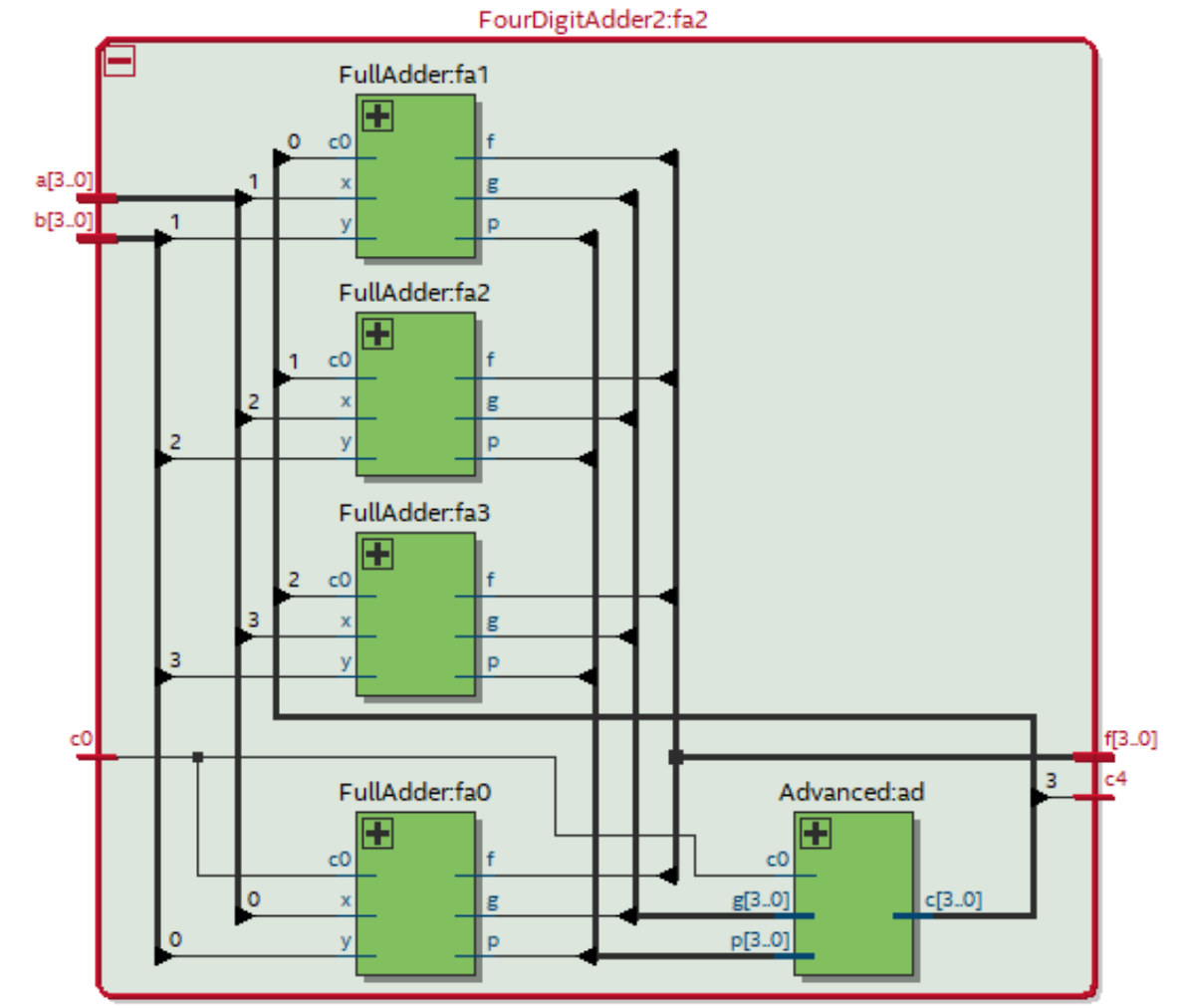
```
module FourDigitAdder2(
    input [3:0] a,
```

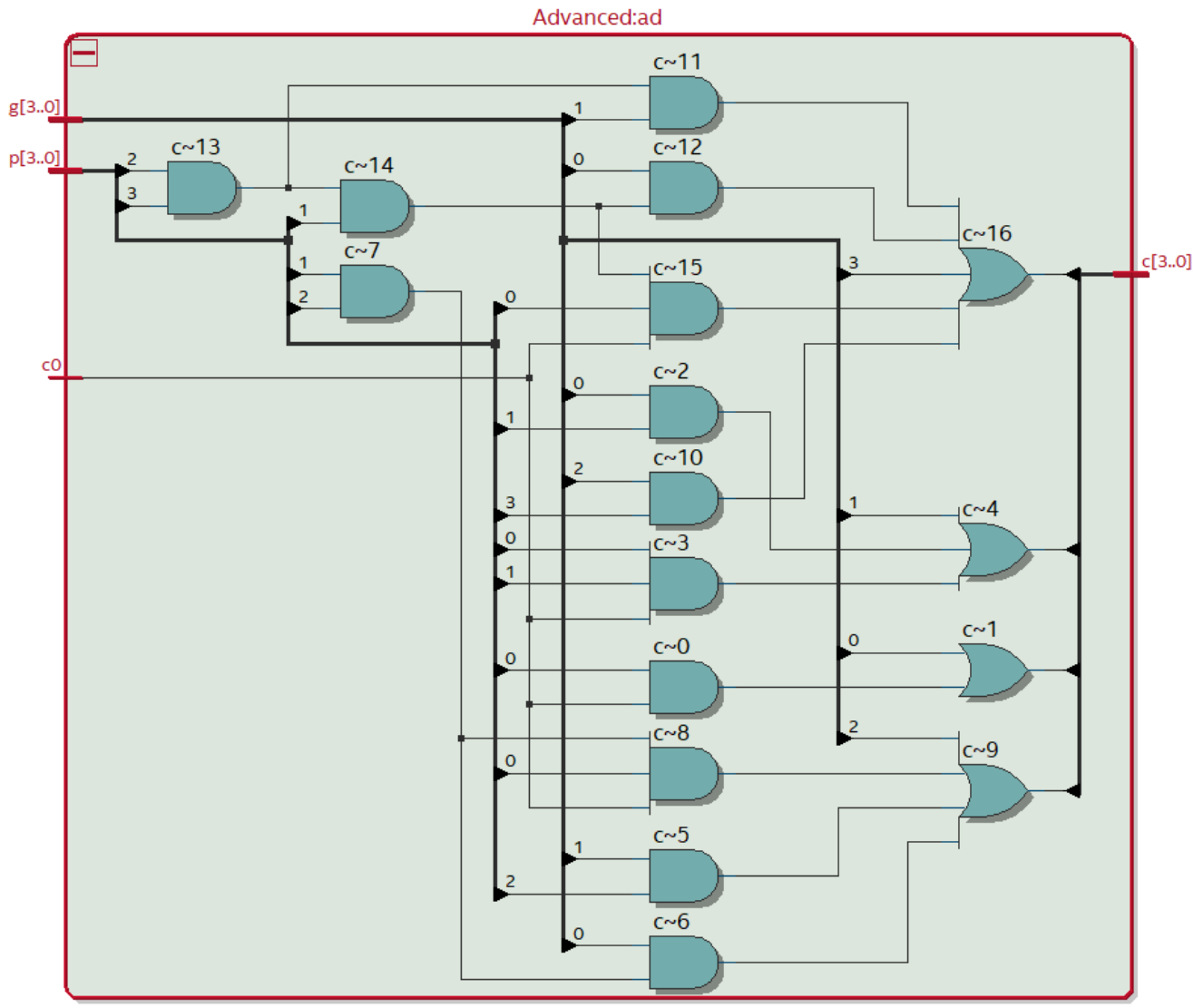
```
    input [3:0] b,
    input c0,
    output [3:0] f,
    output c4
);

    wire [2:0] c;
    wire [3:0] p;
    wire [3:0] g;

    FullAdder fa0(.x(a[0]), .y(b[0]), .c0(c0), .f(f[0]), .p(p[0]), .g(g[0]));
    FullAdder fa1(.x(a[1]), .y(b[1]), .c0(c[0]), .f(f[1]), .p(p[1]), .g(g[1]));
    FullAdder fa2(.x(a[2]), .y(b[2]), .c0(c[1]), .f(f[2]), .p(p[2]), .g(g[2]));
    FullAdder fa3(.x(a[3]), .y(b[3]), .c0(c[2]), .f(f[3]), .p(p[3]), .g(g[3]));
    Advanced ad(.p(p), .g(g), .c0(c0), .c({c4, c}));

endmodule

```

其中超前进位扩展器代码如下：

```
module Advanced(
    input [3:0] p,
    input [3:0] g,
    input c0,
    output [3:0] c
);
    assign c[0] = g[0] | (p[0] & c0);
    assign c[1] = g[1] | (p[1] & g[0]) | (p[1] & p[0] & c0);
    assign c[2] = g[2] | (p[2] & g[1]) | (p[2] & p[1] & g[0]) | (p[2] & p[1] & p[0] & c0);
    assign c[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] & c0);

endmodule
```

生成电路如下：


FourDigitAdder2:fa2
FullAdder:fa1
FullAdder:fa2
FullAdder:fa3
FullAdder:fa0
Advanced:ad
a[3..0]
b[3..0]
c0
f[3..0]
c4
c0
x
y
f
g
p
g[3..0]
p[3..0]
c[3..0]

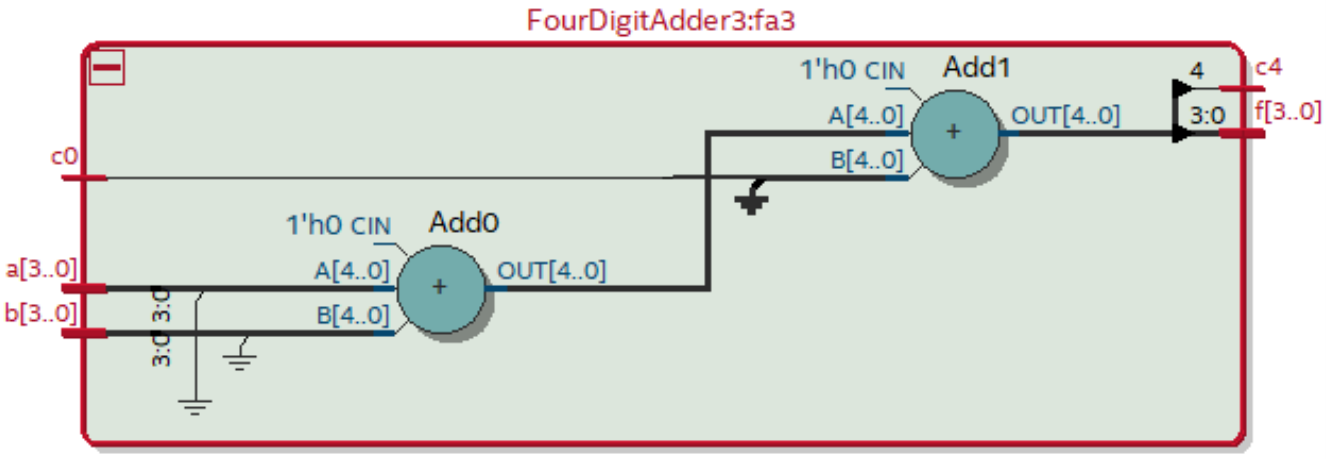



## 4. 系统自带的四位加法器代码及生成电路

代码如下：

```
module FourDigitAdder3(
    input [3:0] a,
    input [3:0] b,
    input c0,
    output [3:0] f,
    output c4
);
    assign {c4, f} = a + b + c0;

endmodule

```

生成电路如下：




## 5. 整合的代码与生成电路

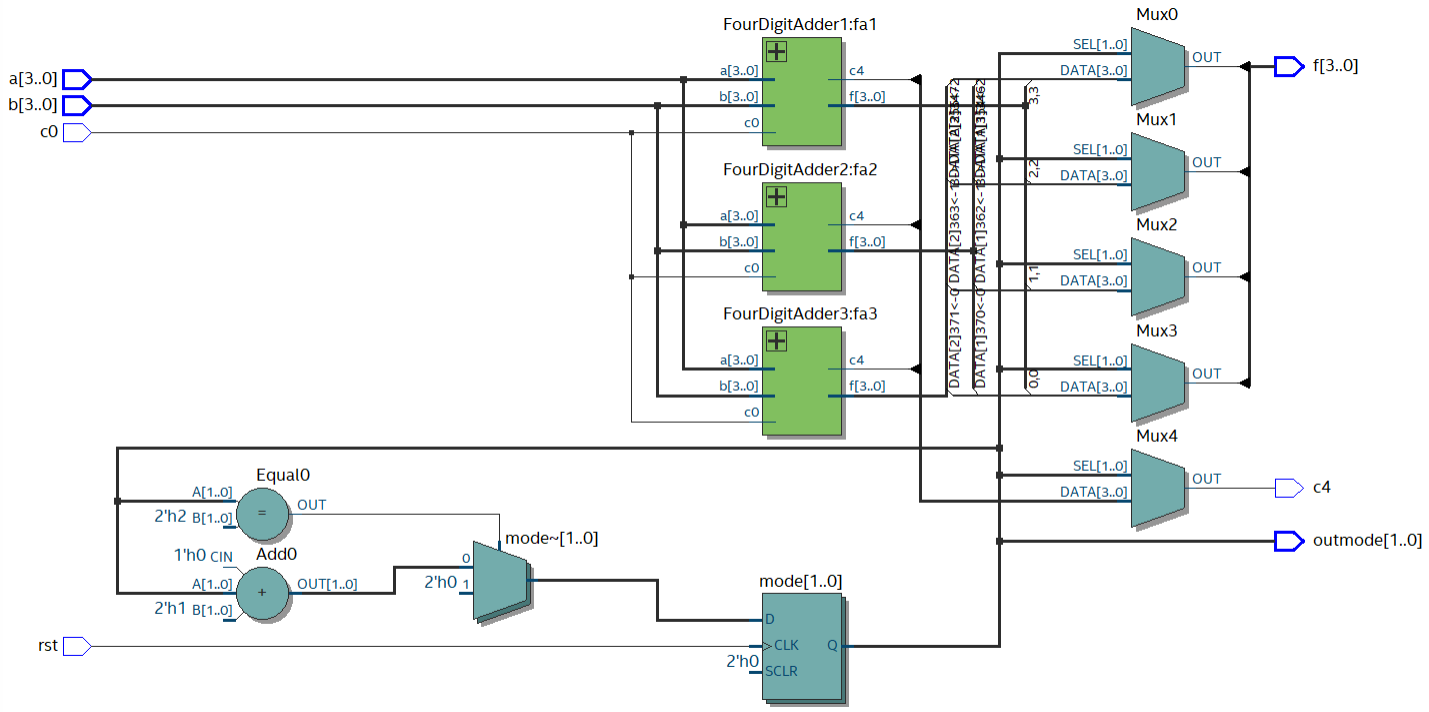
为方便检查，rst 作为切换模式键，利用 mode 和 outmode 来表示当前选择的模式，00 表示逐次进位，01 表示超前进位，10 表示系统自带，三次一循环（11 默认为逐次进位）。

具体代码如下：

```
module FourDigitAdder(
    input [3:0] a,
    input [3:0] b,
    input c0,
    input rst,
    output reg [3:0] f,
    output reg c4,
    output reg [1:0] outmode
);

    wire [3:0] f0;
    wire [3:0] f1;
    wire [3:0] f2;
    wire c40;
    wire c41;
    wire c42;
    reg [1:0] mode;

    initial mode = 2'b00;

    FourDigitAdder1 fa1(.a(a), .b(b), .c0(c0), .f(f0), .c4(c40));
    FourDigitAdder2 fa2(.a(a), .b(b), .c0(c0), .f(f1), .c4(c41));
    FourDigitAdder3 fa3(.a(a), .b(b), .c0(c0), .f(f2), .c4(c42));

    always @(mode) begin
```

```
        outmode <= mode;
    end

    always @(mode, f0, f1, f2, c40, c41, c42) begin
        case (mode)
            2'b00: begin
                f <= f0;
                c4 <= c40;
            end
            2'b01: begin
                f <= f1;
                c4 <= c41;
            end
            2'b10: begin
                f <= f2;
                c4 <= c42;
            end
            default: begin
                f <= f0;
                c4 <= c40;
            end
        endcase
    end

    always @(posedge rst) begin
        if (mode == 2'b10)
            mode <= 2'b00;
        else
            mode <= mode + 1;
    end

endmodule

```

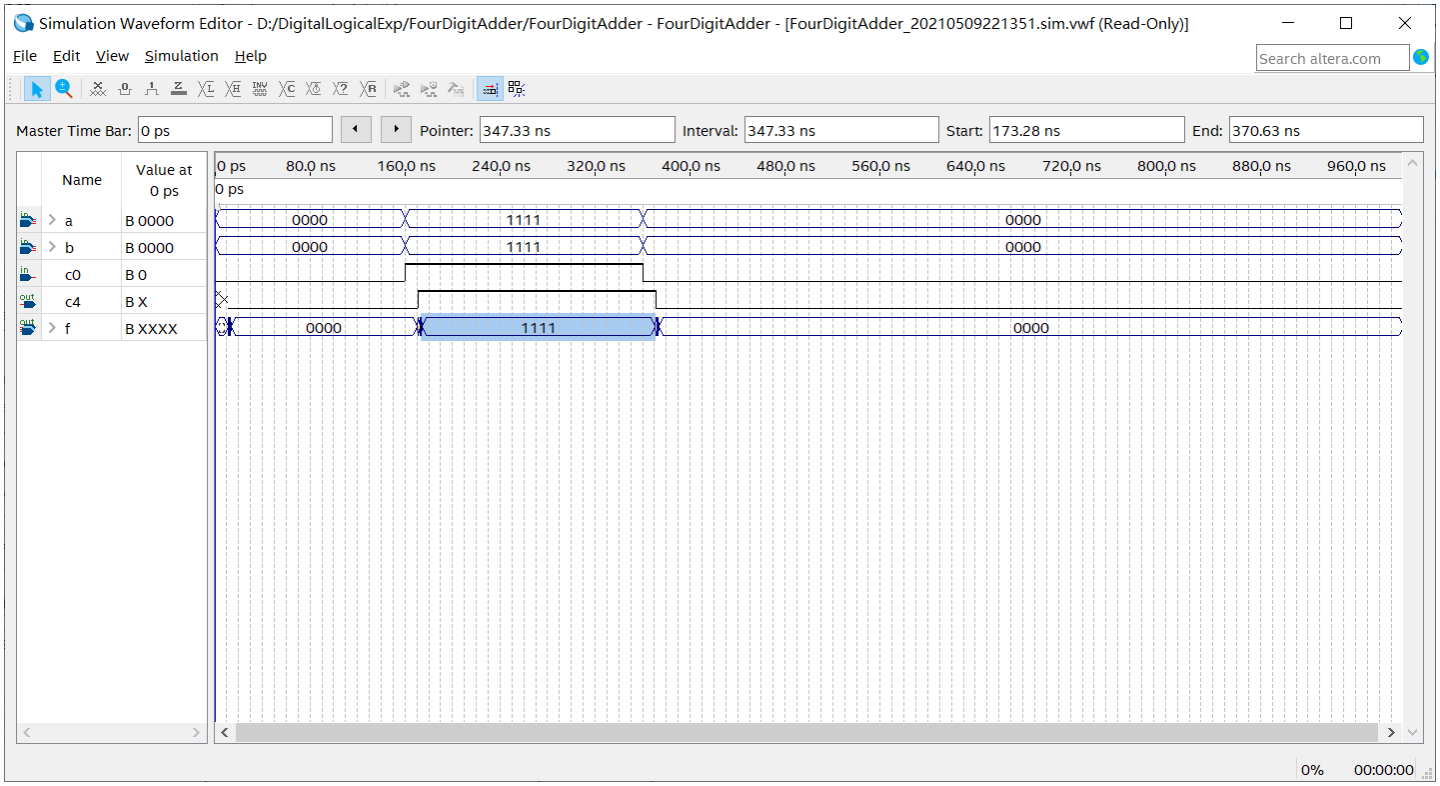
生成电路如下：



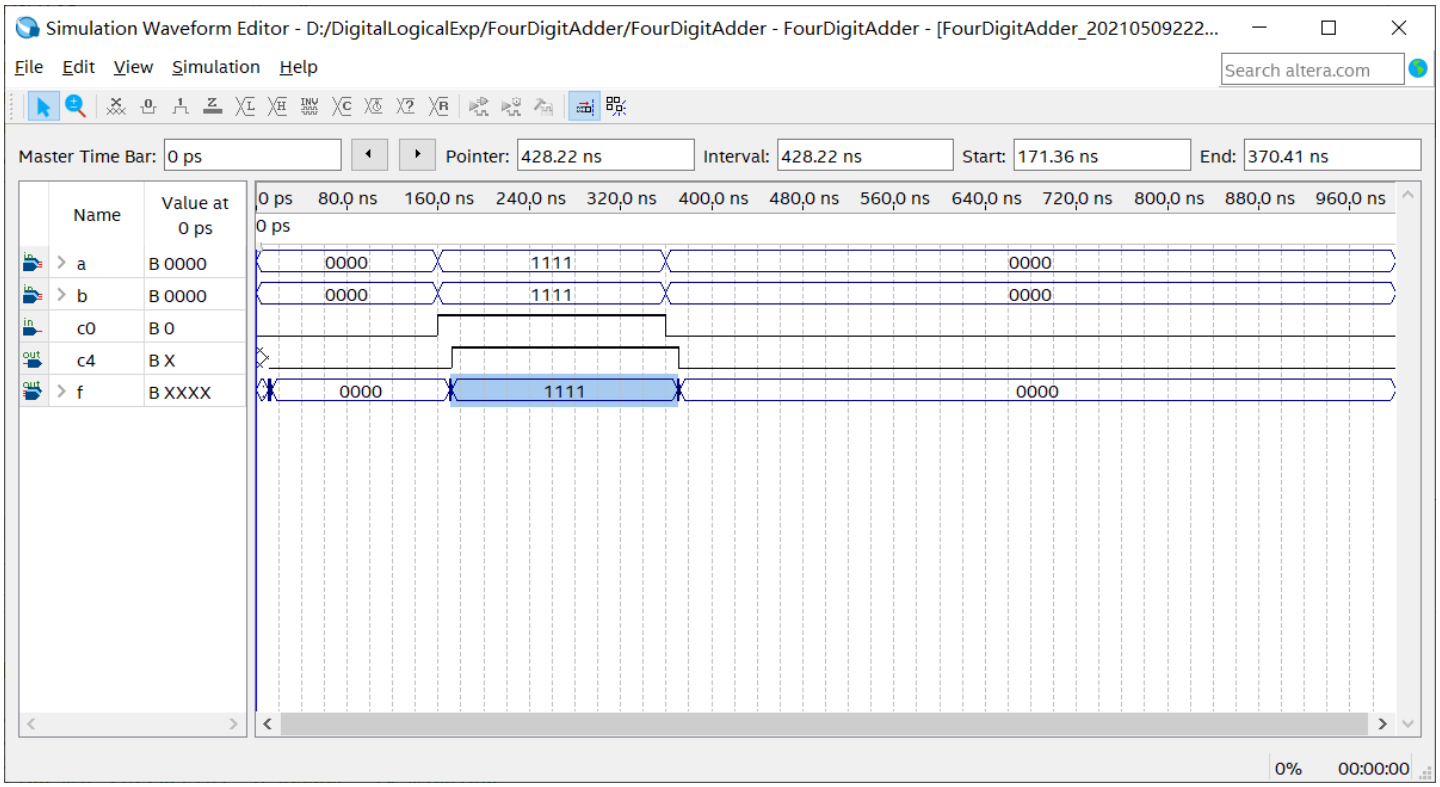

# *四、仿真与延迟探究*

三种加法器的电路测试都得到了正确结果，无法观察到延迟。

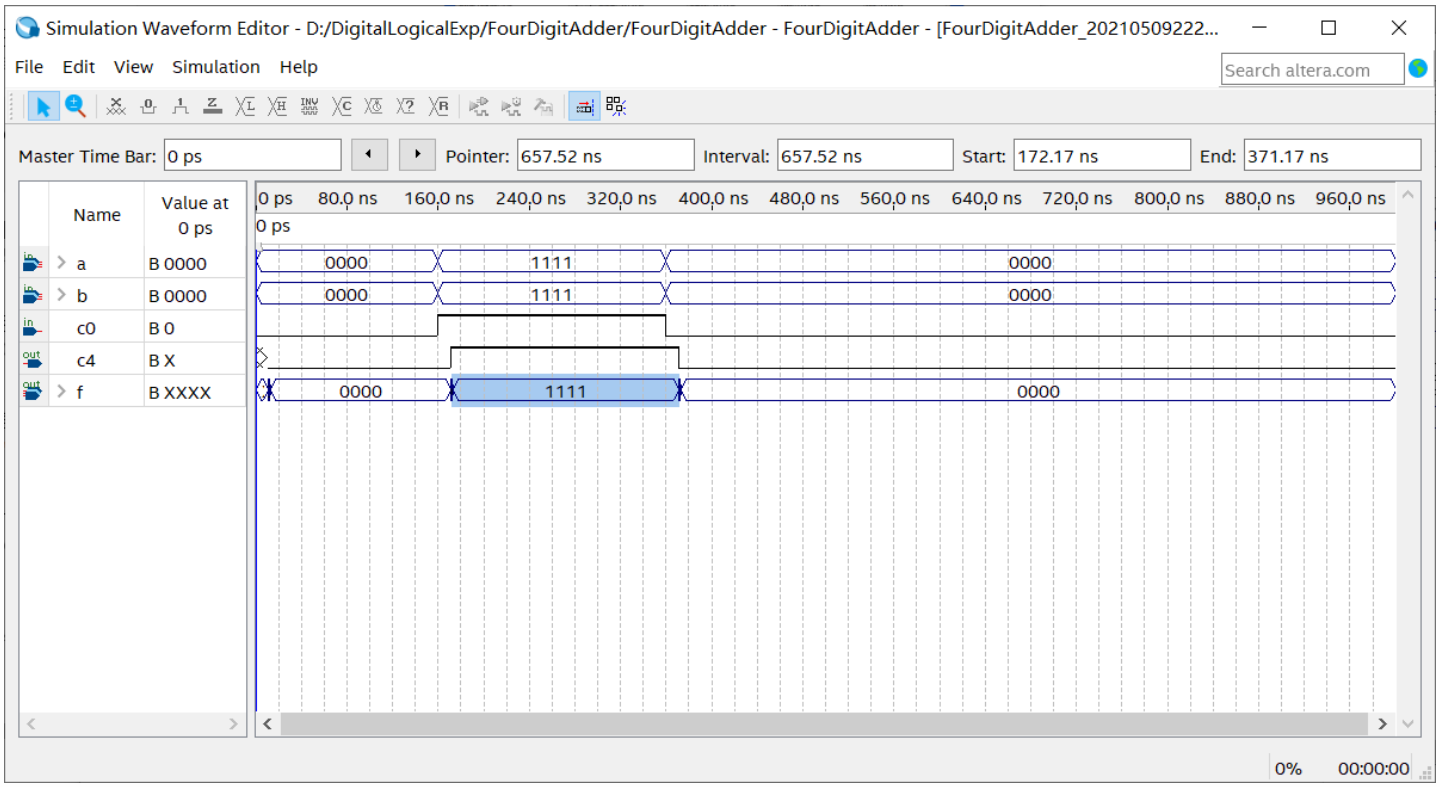
## 1. 逐次进位的四位加法器仿真




## 2. 超前进位的四位加法器仿真



### 3. 系统自带的四位加法器仿真




### 4. 总结

仿真结果中可以看到，逐次进位加法器延迟 $13.28ns$，超前进位加法器延迟 $11.36ns$，系统自带加法器延迟 $12.17ns$。

可以看到三种加法器的延迟并没有很大的区别，这是由于位数过少难以体现系统加法器与超前进位加法器的优势。

## *五、实验收获*

### 1. Verilog 收获

学习了模块实例化的方法，并且对于组合逻辑有了很多应用的空间。

### 2. Quartus 收获

可以通过项目的 `settings` 来修改顶层模块文件的选择，进而方便在三种加法器中进行切换。