简单组合逻辑电路的设计

计 74　傅舟涛 2017010682

1 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与分析。

2 实验内容及要求

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）（研究内容）利用 VHDL 自带的加法实现四位加法器，测试并用仿真验证；
4）（研究内容）查看三者在 CPLD 中生成的电路，并比较其异同。

3 实验原理及代码

1）半加器及全加器代码及生成电路

半加器的功能是实现一个二进制加数的加法运算。a、b 表示两个加数，f 表示半加和，cout 表示向高位的进位。其代码与 CPLD 生成电路如图 1 所示，逻辑表达式为 $f = a \oplus b, cout = a \cdot b$. 半加器、全加器及各加法器的实现较为简单或已由文字说明清楚，不再添加额外注释。

```
entity HalfAdder1 is
    port(
        a,b:in std_logic;
        f,cout:out std_logic
    );
end HalfAdder1;

architecture plus of HalfAdder1 is
begin
    process(a,b)
    begin
        f<=a xor b;
        cout<=a and b;
    end process;
end plus;
```

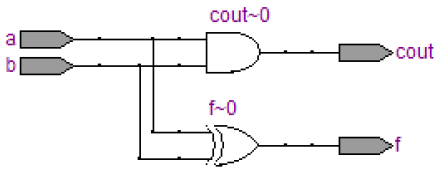


图 1 半加器代码（左）及半加器生成电路（右）

全加器的功能与真值表在上次实验中已经分析过，用元件例化的方法，利用两个半加器和一个或门实现，CPLD 生成电路如图 2 所示，代码如图 3 所示。

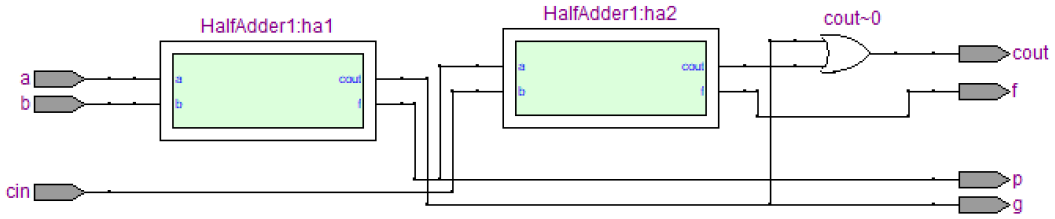


图 2 全加器生成电路

```
entity FullAdder1 is
    port(
        a,b,cin:in std_logic;
        f,cout:out std_logic;
        p,g:buffer std_logic
    );
end FullAdder1;

architecture plus of FullAdder1 is
    component HalfAdder1
        port(
            a,b:in std_logic;
            f,cout:out std_logic
        );
    end component;
    signal r:std_logic;
begin
    ha1:HalfAdder1 port map(a,b,p,g);
    ha2:HalfAdder1 port map(p,cin,f,r);
    cout<=r or g;
end plus;
```

图 3 全加器代码

2）逐次进位的四位加法器代码及生成电路

元件例化后，将对应的接口接好即可，如图 4 和图 5 所示

```
entity FA1 is
    port(
        a,b:in std_logic_vector(3 downto 0);
        cin:in std_logic;
        f:out std_logic_vector(3 downto 0);
        cout:out std_logic
    );
end FA1;

architecture plus of FA1 is
    component FullAdder1
        port(
            a,b,cin:in std_logic;
            f,cout:out std_logic
        );
    end component;
    signal c:std_logic_vector(2 downto 0);
begin
    fa0:FullAdder1 port map(a(0),b(0),cin,f(0),c(0));
    fa1:FullAdder1 port map(a(1),b(1),c(0),f(1),c(1));
    fa2:FullAdder1 port map(a(2),b(2),c(1),f(2),c(2));
    fa3:FullAdder1 port map(a(3),b(3),c(2),f(3),cout);
end plus;
```

图 4 逐次进位的四位加法器代码

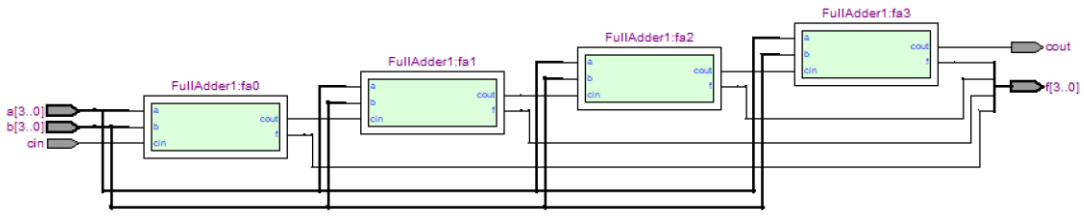


图 5 逐次进位的四位加法器生成电路

3）超前进位的四位加法器代码及生成电路

定义传递信号$P_n = A_n \oplus B_n$，进位产生信号$G_n = A_n \cdot B_n$，代入进位表达式可以得到：

$$C_0 = G_0 + P_0C_{-1}, C_1 = G_1 + P_1G_0 + P_1P_0C_{-1},$$

$$C_2 = G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_{-1},$$

$$C_3 = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0 + P_3P_2P_1P_0C_{-1}$$

利用超前进位运算器（输入为 P、G、C-1，输出为 C），可以由四个全加器和一个超前进位运算器得到超前进位的四位加法器，如图 6-8 所示。

```
entity FA2 is
    port(
        a,b:in std_logic_vector(3 downto 0);
        cin:in std_logic;
        f:out std_logic_vector(3 downto 0);
        cout:out std_logic
    );
end FA2;

architecture plus of FA2 is
    component FullAdder1
        port(
            a,b,cin:in std_logic;
            f,cout:out std_logic;
            p,g:buffer std_logic
        );
    end component;
    component Advanced
        port(
            p,g:in std_logic_vector(3 downto 0);
            cin:in std_logic;
            c:out std_logic_vector(3 downto 0)
        );
    end component;
    signal p,g:std_logic_vector(3 downto 0);
    signal c:std_logic_vector(2 downto 0);
begin
    fa0:FullAdder1 port map(a(0),b(0),cin,f=>f(0),p=>p(0),g=>g(0));
    fa1:FullAdder1 port map(a(1),b(1),c(0),f=>f(1),p=>p(1),g=>g(1));
    fa2:FullAdder1 port map(a(2),b(2),c(1),f=>f(2),p=>p(2),g=>g(2));
    fa3:FullAdder1 port map(a(3),b(3),c(2),f=>f(3),p=>p(3),g=>g(3));
    ad:Advanced port map(p,g,cin,c(0)=>c(0),c(1)=>c(1),c(2)=>c(2),c(3)=>cout);
end plus;

entity Advanced is
    port(
        p,g:in std_logic_vector(3 downto 0);
        cin:in std_logic;
        c:out std_logic_vector(3 downto 0)
    );
end Advanced;

architecture ad of Advanced is
begin
    process(p,g,cin)
    begin
        c(0)<=g(0) or (p(0) and cin);
        c(1)<=g(1) or (p(1) and g(0)) or (p(1) and p(0) and cin);
        c(2)<=g(2) or (p(2) and g(1)) or (p(2) and p(1) and g(0))
         or (p(2) and p(1) and p(0) and cin);
        c(3)<=g(3) or (p(3) and g(2)) or (p(3) and p(2) and g(1))
         or (p(3) and p(2) and p(1) and g(0)) or (p(3) and p(2) and p(1) and p(0) and cin);
    end process;
end ad;
```

图 6 超前进位的四位加法器代码

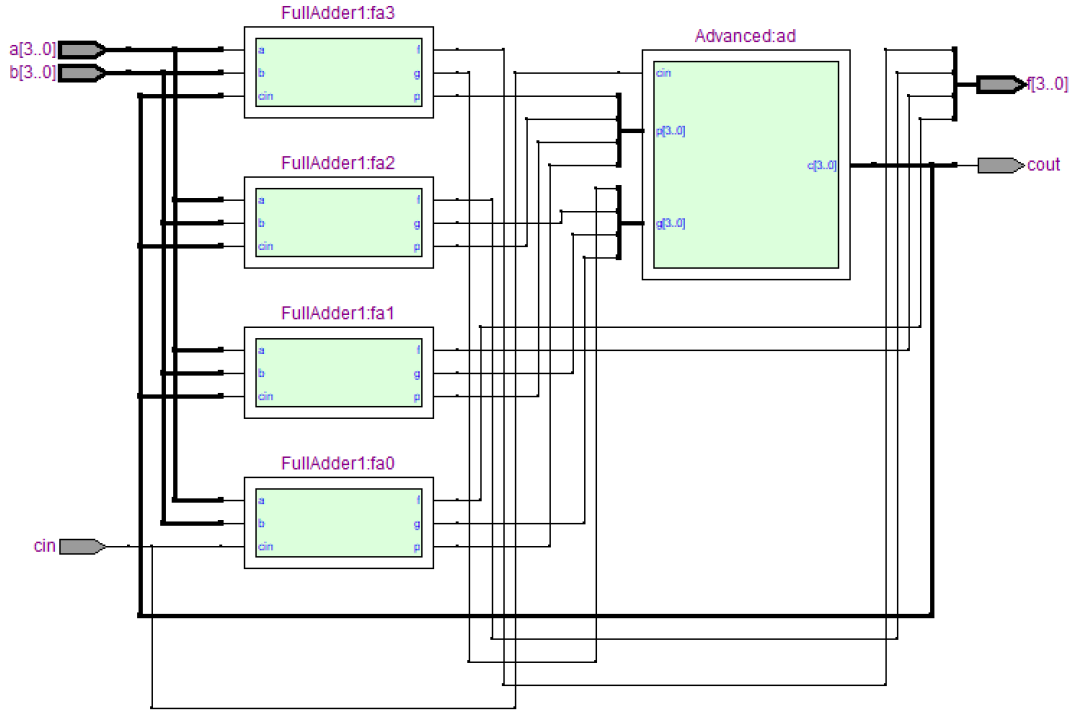


图 7 超前进位的四位加法器生成电路

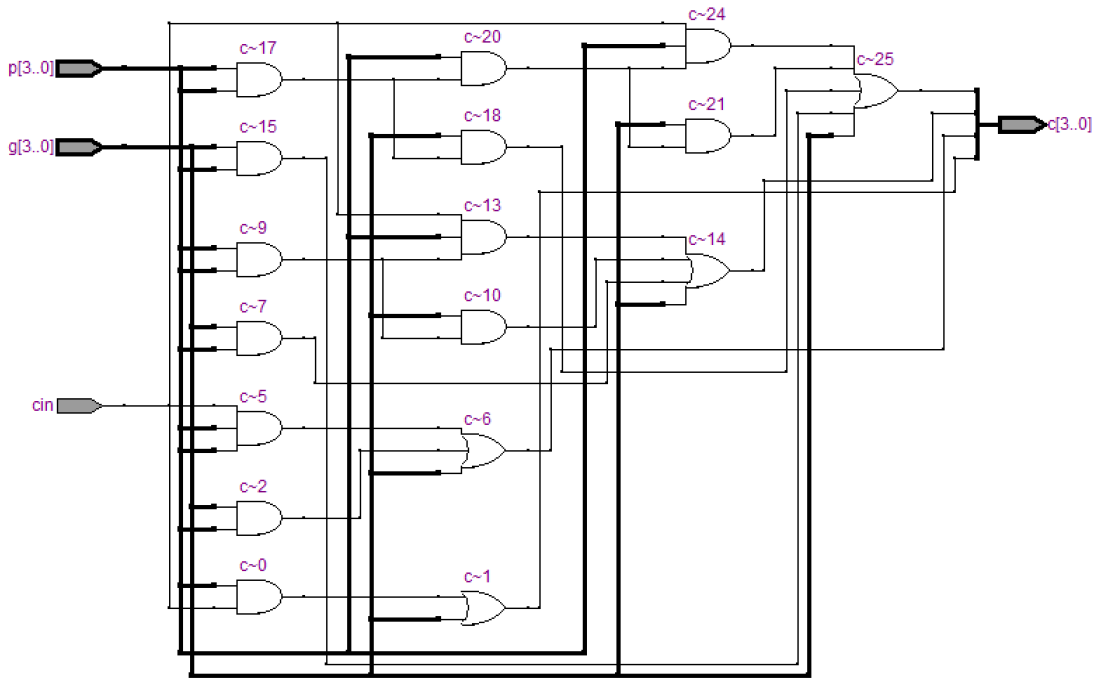


图 8 超前进位运算器生成电路

4）系统自带的四位加法器代码及生成电路

如图 9 所示，让 buf 得到加法的 5 位和，并令 cout 和 f 分别取 buf 的对应位即可。

```
entity FA3 is
    port(
        a,b:in std_logic_vector(3 downto 0);
        cin:in std_logic;
        f:out std_logic_vector(3 downto 0);
        cout:out std_logic
    );
end FA3;

architecture plus of FA3 is
    signal buf:std_logic_vector(4 downto 0);
begin
    process(a,b,cin)
    begin
        buf<="00000"+a+b+cin;
    end process;
    process(buf)
    begin
        cout<=buf(4);
        f<=buf(3 downto 0);
    end process;
end plus;
```

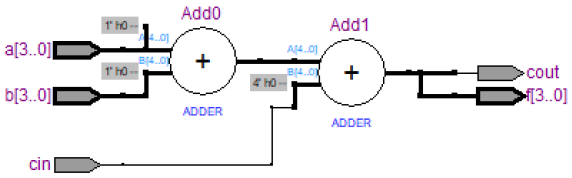


图 9 自带的四位加法器代码（左）及生成电路（右）

5）整合的代码与生成电路

为方便助教检测，将 rst 作为模式切换按键，即：初始时系统为逐次进位加法器；按下一次 rst 后，切换为超前进位加法器；再按一次，切换为系统自带加法器；再按一次，切换为逐次进位加法器。代码及生成电路如图 10 和图 11 所示。

```
entity Adder is
    port(
        a,b:in std_logic_vector(3 downto 0);
        rst,cin:in std_logic;
        f:out std_logic_vector(3 downto 0);
        mode:out std_logic_vector(1 downto 0);
        cout:out std_logic
    );
end Adder;
--加法器定义，其中mode负责输出指示（不负责内部控制）
architecture plus of Adder is
    component FA1
        port(
            a,b:in std_logic_vector(3 downto 0);
            cin:in std_logic;
            f:out std_logic_vector(3 downto 0);
            cout:out std_logic
        );
    end component;
    component FA2
        port(
            a,b:in std_logic_vector(3 downto 0);
            cin:in std_logic;
            f:out std_logic_vector(3 downto 0);
            cout:out std_logic
        );
    end component;
    component FA3
        port(
            a,b:in std_logic_vector(3 downto 0);
            cin:in std_logic;
            f:out std_logic_vector(3 downto 0);
            cout:out std_logic
        );
    end component;
    --元件例化声明
    signal f1,f2,f3:std_logic_vector(3 downto 0);
    signal cout1,cout2,cout3:std_logic;
    --f1-f3、cout1-cout3负责储存三个加法器的结果
```

```
    signal imode:std_logic_vector(1 downto 0) :="00";
    --imode负责控制使用哪个加法器
begin
    fAdder1:FA1 port map(a,b,cin,f1,cout1);
    fAdder2:FA2 port map(a,b,cin,f2,cout2);
    fAdder3:FA3 port map(a,b,cin,f3,cout3);
    --元件例化
    process(rst)
    begin
        if (rst'event and rst='1') then
            if (imode = "11") then
                imode<="00";
            else
                imode<=imode+1;
            end if;
        end if;
    end process;
    process(imode)
    begin
        mode<=imode;
    end process;
    --imode的转换
    process(imode,f1,f2,f3,cout1,cout2,cout3)
    begin
        if (imode(1)='0') then
            if (imode(0)='0') then
                f<=f1;
                cout<=cout1;
            else
                f<=f2;
                cout<=cout2;
            end if;
        else
            f<=f3;
            cout<=cout3;
        end if;
    end process;
    --imode的控制
end plus;
```

图 10 整体代码

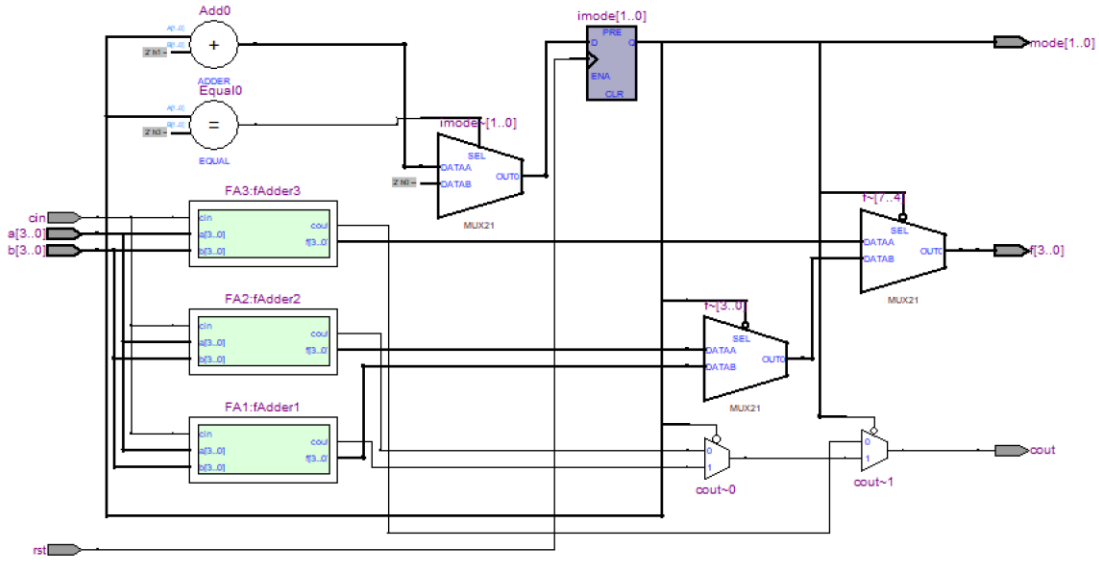


图 11 整体生成电路

4 仿真与延迟研究

为了达到仿真的目的（测量延迟和输出），将 FA1、FA2、FA3 分开，并仿真得到了正确的结果。对于不同的输入变化，延迟时间不同，取最大延迟（0000+0000+0->1111+1111+1）的输入变化，分别的仿真结果如图 12-图 14 所示。

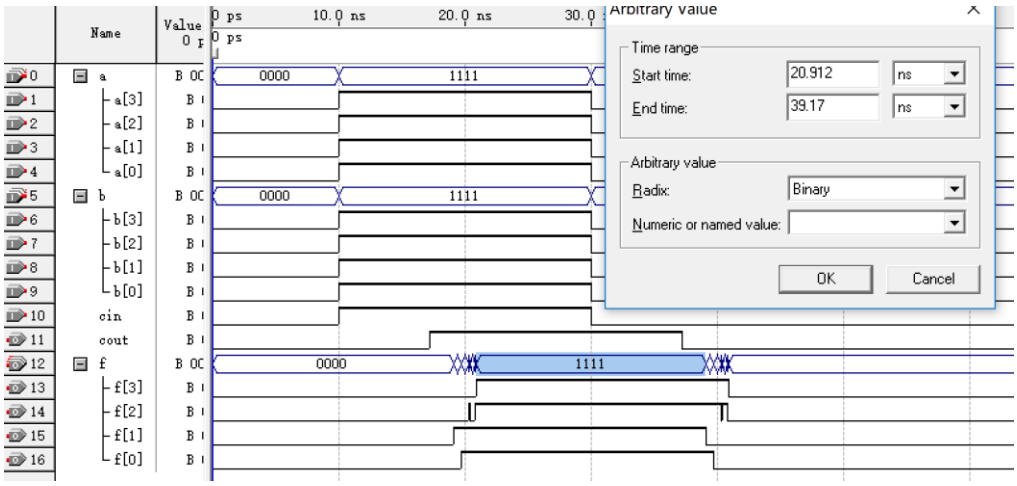


图 12 逐次进位加法器仿真延迟

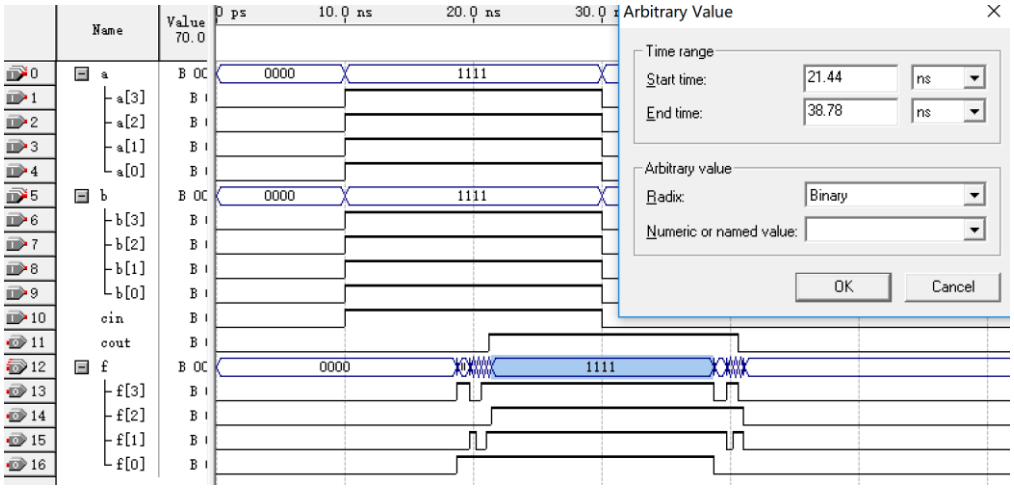


图 13 超前进位加法器仿真延迟

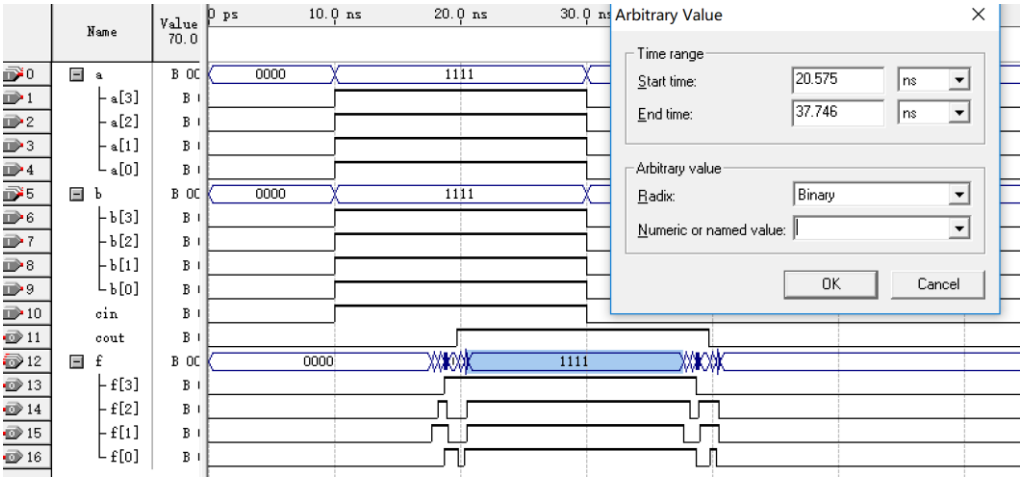


图 14 系统自带加法器仿真延迟

仿真结果为：逐次进位加法器延迟 10.912ns；超前进位加法器延迟 11.44ns；系统自带加法器延迟 10.575ns.延迟：系统自带加法器<逐次进位加法器<超前进位加法器。在本实验中，超前进位加法器未能体现其优势，这是由于位数较少，超前进位的计算较多，且 CPLD 生成的超前进位运算器（图 8）门延迟较大导致的。系统自带加法器在延迟上有一定优势，其内部完成了一些优化。

5 电路测试结果

三种加法器都得到了正确的结果，并与仿真的结果一致。不过，由于延迟时间过短，肉眼观测不到延迟的存在。

6 生成电路研究

前面提到，图 8 的设计并未如我们想象中的两级门延迟（与+或），而是进行了“优化”，“优化”后，门延迟有了大幅提高（2 级->4 级），我们希望改变这种优化，于是将中间状态存起来，代码及生成电路如图 15-16 所示，仿真结果如图 17 所示。

可以看到，经过此次改进后，加法器延迟降低到了 9.055ns，低于系统自带的加法器与逐次进位加法器，得到了比较好的效果。然而，生成电路图表明，仍然有 3 级门延迟，这是因为 quartus 进行优化后，部分中间状态利用了其他中间状态的结果生成，这意味着还有进一步改进的空间。

```
entity Advanced is
    port(
        p,g:in std_logic_vector(3 downto 0);
        cin:in std_logic;
        c:out std_logic_vector(3 downto 0)
    );
end Advanced;

architecture ad of Advanced is
    signal n:std_logic_vector(9 downto 0);
begin
    process(p,g,cin)
    begin
        n(0)<=p(0) and cin;
        n(1)<=p(1) and g(0);
        n(2)<=p(1) and p(0) and cin;
        n(3)<=p(2) and g(1);
        n(4)<=p(2) and p(1) and g(0);
        n(5)<=p(2) and p(1) and p(0) and cin;
        n(6)<=p(3) and g(2);
        n(7)<=p(3) and p(2) and g(1);
        n(8)<=p(3) and p(2) and p(1) and g(0);
        n(9)<=p(3) and p(2) and p(1) and p(0) and cin;
    end process;
    process(n,g)
    begin
        c(0)<=g(0) and n(0);
        c(1)<=g(1) and n(1) and n(2);
        c(2)<=g(2) and n(3) and n(4) and n(5);
        c(3)<=g(3) and n(6) and n(7) and n(8) and n(9);
    end process;
end ad;
```

图 15 超前进位运算器改 1 代码

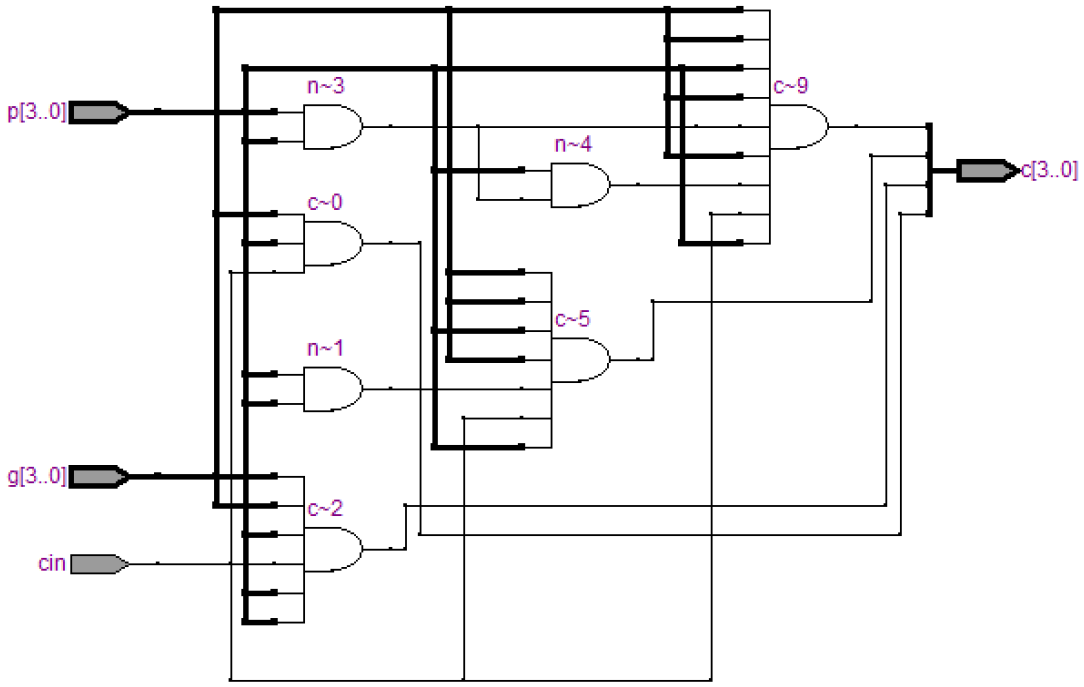


图 16 超前进位运算器改 1 生成电路

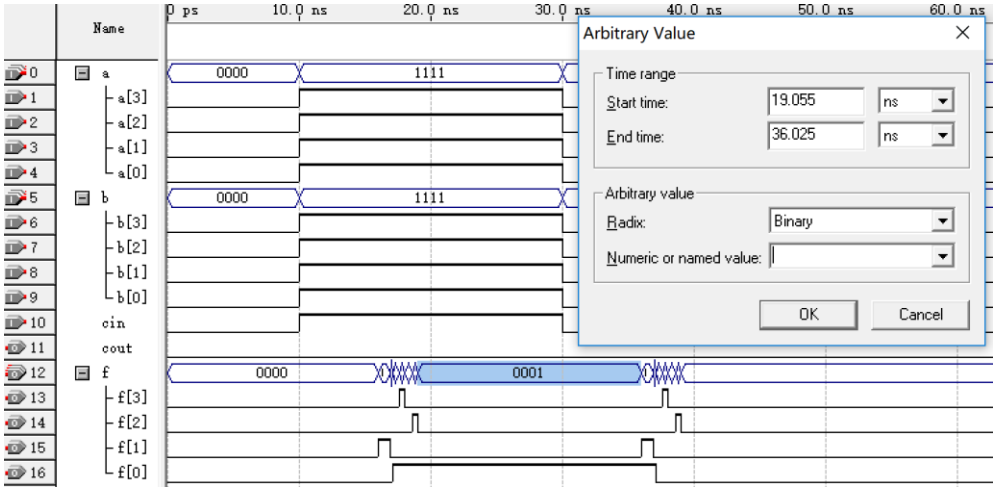
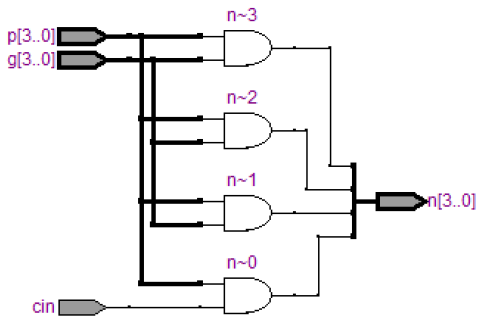
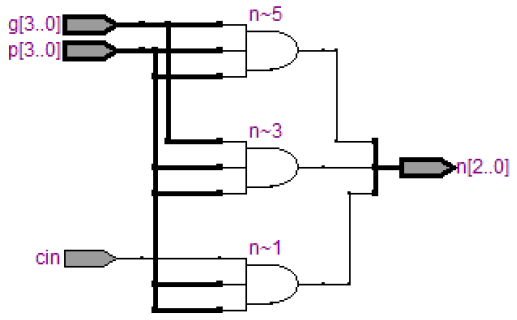


图 17 超前进位运算器改 1 仿真结果

强行用元件例化的方式，可以将超前进位运算器变成二级门延迟，结果如图 18-20 所示，仿真延迟没有变化。





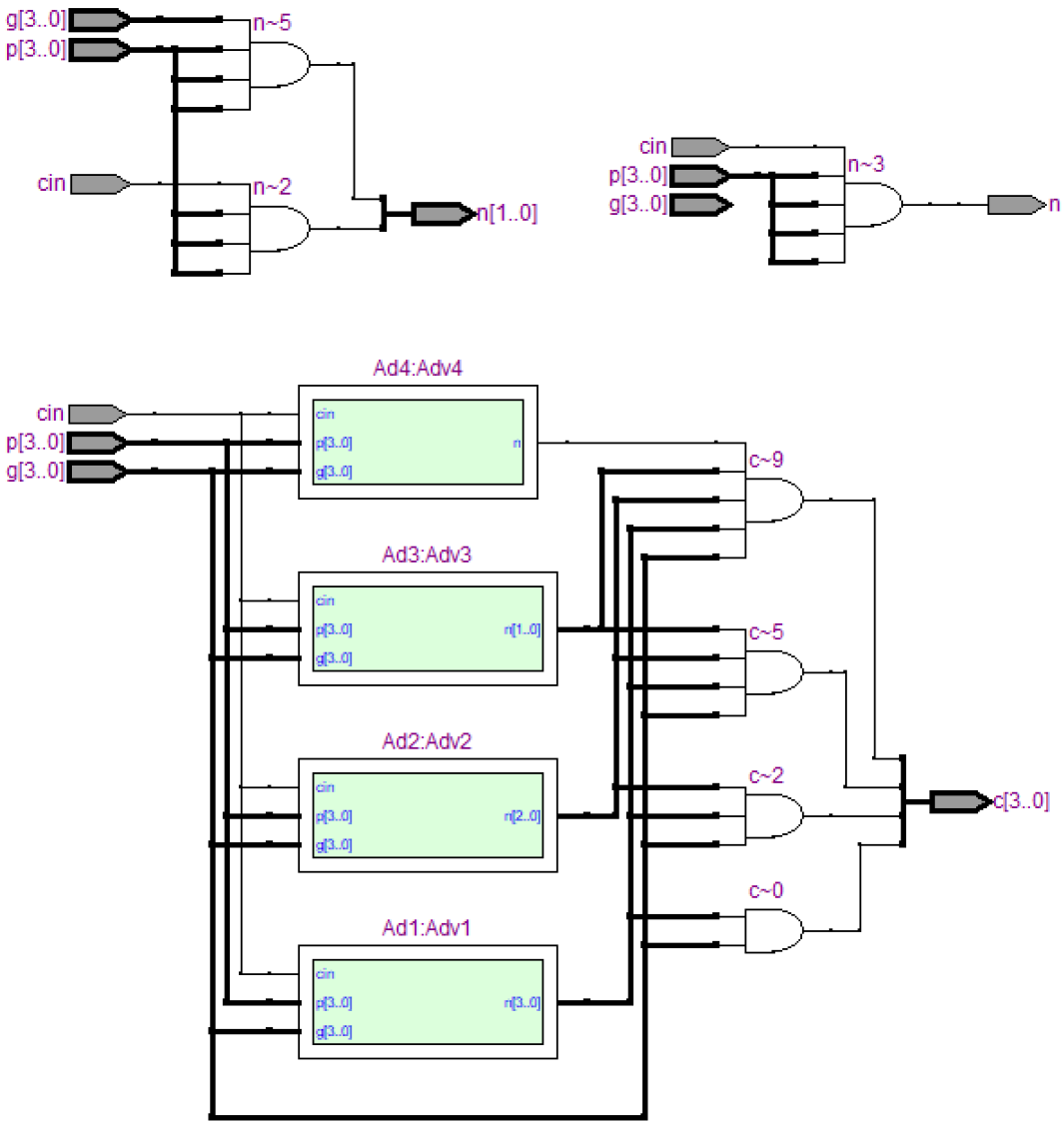


图 18 超前进位运算器改 2 生成电路

（左上 Ad1，右上 Ad2，左下 Ad3，右下 Ad4，下方 Advanced）

```
entity Advanced is
    port(
        p,g:in std_logic_vector(3 downto 0);
        cin:in std_logic;
        c:out std_logic_vector(3 downto 0)
    );
end Advanced;

architecture ad of Advanced is
    component Ad1
        port(
            p,g:in std_logic_vector(3 downto 0);
            cin:in std_logic;
            n:out std_logic_vector(3 downto 0)
        );
    end component;
    component Ad2
        port(
            p,g:in std_logic_vector(3 downto 0);
            cin:in std_logic;
            n:out std_logic_vector(2 downto 0)
        );
    end component;
    component Ad3
        port(
            p,g:in std_logic_vector(3 downto 0);
            cin:in std_logic;
            n:out std_logic_vector(1 downto 0)
        );
    end component;
    component Ad4
        port(
            p,g:in std_logic_vector(3 downto 0);
            cin:in std_logic;
            n:out std_logic
        );
    end component;
    signal n:std_logic_vector(9 downto 0);
begin
    Adv1:Ad1 port map(p,g,cin,n(3 downto 0));
    Adv2:Ad2 port map(p,g,cin,n(6 downto 4));
    Adv3:Ad3 port map(p,g,cin,n(8 downto 7));
    Adv4:Ad4 port map(p,g,cin,n(9));
    process(n,g)
    begin
        c(0)<=g(0) and n(0);
        c(1)<=g(1) and n(1) and n(4);
        c(2)<=g(2) and n(2) and n(5) and n(7);
        c(3)<=g(3) and n(3) and n(6) and n(8) and n(9);
    end process;
end ad;
```

```
entity Ad1 is
    port(
        p,g:in std_logic_vector(3 downto 0);
        cin:in std_logic;
        n:out std_logic_vector(3 downto 0)
    );
end Ad1;

architecture a1 of ad1 is
begin
    n(0)<=p(0) and cin;
    n(1)<=p(1) and g(0);
    n(2)<=p(2) and g(1);
    n(3)<=p(3) and g(2);
end a1;

entity Ad3 is
    port(
        p,g:in std_logic_vector(3 downto 0);
        cin:in std_logic;
        n:out std_logic_vector(1 downto 0)
    );
end Ad3;

architecture a3 of Ad3 is
begin
    n(0)<=p(2) and p(1) and p(0) and cin;
    n(1)<=p(3) and p(2) and p(1) and g(0);
end a3;
```

```
entity Ad2 is
    port(
        p,g:in std_logic_vector(3 downto 0);
        cin:in std_logic;
        n:out std_logic_vector(2 downto 0)
    );
end Ad2;

architecture a2 of Ad2 is
begin
    n(0)<=p(1) and p(0) and cin;
    n(1)<=p(2) and p(1) and g(0);
    n(2)<=p(3) and p(2) and g(1);
end a2;


entity Ad4 is
    port(
        p,g:in std_logic_vector(3 downto 0);
        cin:in std_logic;
        n:out std_logic
    );
end Ad4;

architecture a4 of Ad4 is
begin
    n<=p(3) and p(2) and p(1) and p(0) and cin;
end a4;
```

图 19 超前进位运算器改 2 代码

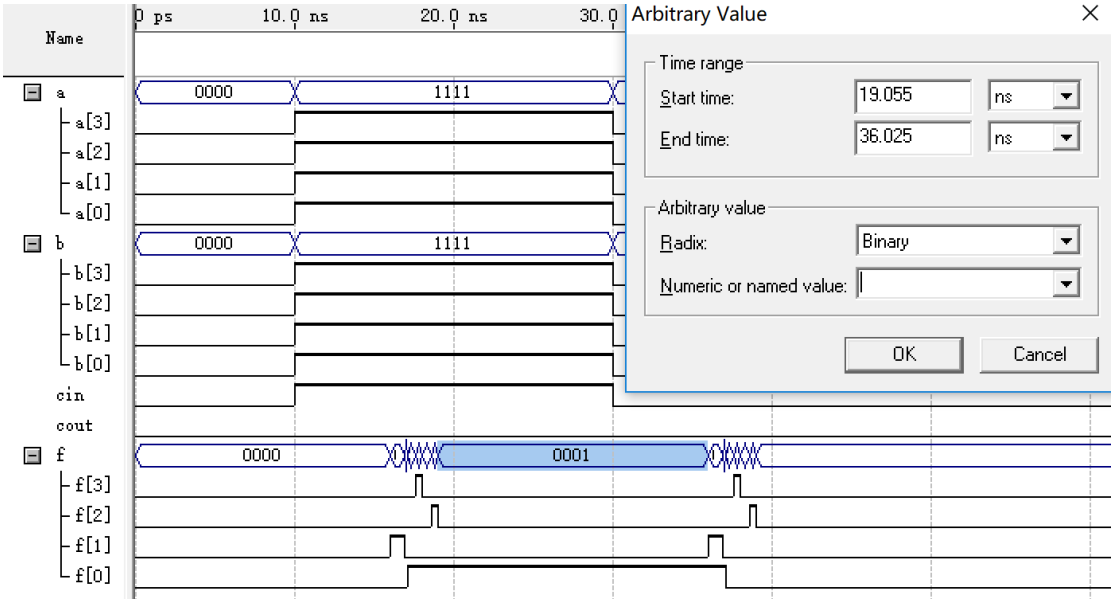


图 20 超前进位运算器改 2 仿真结果

6 quartus 特性研究：遇到的 bug 总结

1，map、all、sll 等是系统保留字，声明信号和接口时不能使用这些名字；

2，语言不区分大小写，不能在一个 entity 里同时定义 C 和 c 两个接口；

3，并发代入语句可以在结构体的进程外使用，不需要 process 关键字，但除此之外必须用 process 关键字，如进程需要 case、if 等语句时；

4，进程中信号赋值为非阻塞赋值，所有信号/接口在 process 结束时统一赋值；

5，一个 vhd 文件中，可以没有 entity 或有一个 entity，可以没有、有一个或多个 architecture；但是如果有 entity，必须与 vhd 文件名相同，因此每个 vhd 文件最多有一个 entity。