

四位加法器

计06 赵晨阳 2020012363

一. 实验目的

1. 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

二. 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. 查看二者在 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

半加器 (halfAdder)

功能为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。

```

1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.std_logic_arith.all;
4  use ieee.std_logic_unsigned.all;
5
6  entity halfAdder is
7      port(
8          a,b:in std_logic;
9          f,cout:out std_logic
10     );
11 end halfAdder;
12
13 architecture plus of halfAdder is
14 begin
15     process(a,b)
16     begin
17         f<=a xor b;
18         cout<=a and b;
19     end process;
20 end plus;

```

全加器 (fullAdder)

三个输入 `a,b,cin`，两个输出 `f,cout`，用元件例化的方法，利用两个半加器和一个或门即可实现。

$$\begin{aligned}
 f &= a \oplus b \oplus cin \\
 cout &= a \bullet b + cin \bullet (a \oplus b)
 \end{aligned}
 \tag{1}$$

```

1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.std_logic_arith.all;
4  use ieee.std_logic_unsigned.all;
5
6  entity fullAdder is
7      port(
8          a,b,cin:in std_logic;
9          f,cout:out std_logic;
10         p,g:buffer std_logic
11     );
12 end fullAdder;

```

```

13
14 architecture plus of fullAdder is
15     component halfAdder
16     port(
17         a,b:in std_logic;
18         f,cout:out std_logic
19     );
20 end component;
21 signal r:std_logic;
22 begin
23     half1:halfAdder port map(a,b,p,g);
24     half2:halfAdder port map(p,cin,f,r);
25     cout<=r or g;
26 end plus;

```

四位加法器

- 用元件例化，利用全加器 `fullAdder` 实现。
- 总共创建了两个工程文件，`bite` 为逐次进位加法器，`adder` 为超前进位加法器。
- 其中超前进位的四位加法器实现逻辑如下

$$\begin{aligned}
 C_0 &= G_0 + P_0 C_{-1}, \\
 C_1 &= G_1 + P_1 G_0 + P_1 P_0 C_{-1}, \\
 C_2 &= G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_{-1}, \\
 C_3 &= G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 C_{-1}
 \end{aligned} \tag{2}$$

```

1  ````
2  adder.vhd, 超前进位加法器
3  ````
4  library ieee;
5  use ieee.std_logic_1164.all;
6  use ieee.std_logic_arith.all;
7  use ieee.std_logic_unsigned.all;
8
9  entity adder is
10     port(
11         a,b:in std_logic_vector(3 downto 0);--输入为 a,b 两个四位二
进制数
12         c0:in std_logic;--也即 C_{-1}
13         f:out std_logic_vector(3 downto 0);--输出为一个四位二进制数

```

```

14     c4:out std_logic --表示向更高位进位
15 );
16 end adder;
17
18 architecture advance of adder is
19     component fullAdder
20     port(
21         a,b,cin:in std_logic;
22         f,cout:out std_logic;
23         p,g:buffer std_logic
24     );
25     end component;
26     signal p,g:std_logic_vector(3 downto 0);
27     signal c:std_logic_vector(2 downto 0);
28 begin --参考数电讲义，使用四个全加器实现超前进位四位加法器
29     f0:fullAdder port
30     map(a(0),b(0),c0,f=>f(0),p=>p(0),g=>g(0));
31     f1:fullAdder port
32     map(a(1),b(1),c(0),f=>f(1),p=>p(1),g=>g(1));
33     f2:fullAdder port
34     map(a(2),b(2),c(1),f=>f(2),p=>p(2),g=>g(2));
35     f3:fullAdder port
36     map(a(3),b(3),c(2),f=>f(3),p=>p(3),g=>g(3));
37     process(p,g,c0)
38     begin --超前进位加法器的逻辑表达式
39         c(0)<=g(0) or (p(0) and c0);
40         c(1)<=g(1) or (p(1) and g(0)) or (p(1) and p(0) and c0);
41         c(2)<=g(2) or (p(2) and g(1)) or (p(2) and p(1) and g(0))
42         or (p(2) and p(1) and p(0) and c0);
43         c4<=g(3) or (p(3) and g(2)) or (p(3) and p(2) and g(1))
44         or (p(3) and p(2) and p(1) and g(0)) or (p(3) and p(2) and
45         p(1) and p(0) and c0);
46     end process;
47 end advance;

```

```

1  ````
2  bite.vhd, 逐次进位加法器
3  ````
4  library ieee;
5  use ieee.std_logic_1164.all;
6  use ieee.std_logic_arith.all;
7  use ieee.std_logic_unsigned.all;

```

```

8
9 entity bite is
10     port(
11         a,b:in std_logic_vector(3 downto 0);--输入为a,b两个四位二进制
数
12         c0:in std_logic;--低位进位输入
13         f:out std_logic_vector(3 downto 0);--输出为一个四位二进制数
14         c4:out std_logic --向更高位的进位输出
15     );
16 end bite;
17
18 architecture advance of bite is
19     component fullAdder
20     port(
21         a,b,cin:in std_logic;
22         f,cout:out std_logic
23     );
24 end component;
25 signal c:std_logic_vector(2 downto 0);
26 begin--元件例化，用四个全加器实现四位加法器，没有复杂的逻辑
27 f0:fullAdder port map(a(0),b(0),c0,f(0),c(0));
28 f1:fullAdder port map(a(1),b(1),c(0),f(1),c(1));
29 f2:fullAdder port map(a(2),b(2),c(1),f(2),c(2));
30 f3:fullAdder port map(a(3),b(3),c(2),f(3),c4);
31 end advance;

```

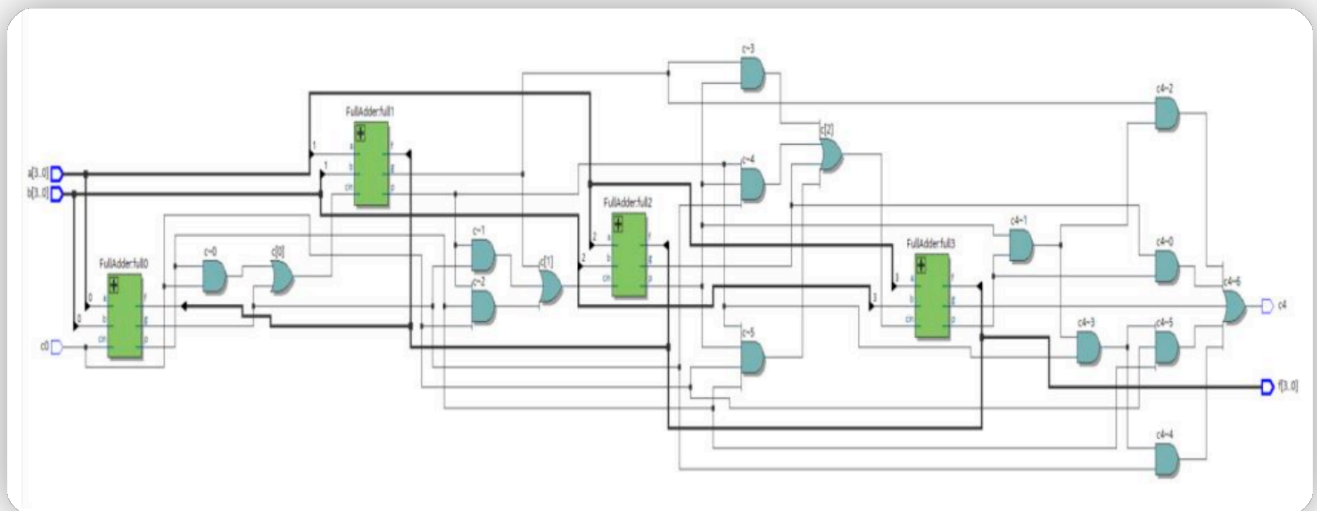
引脚分配

- 完全参考课本，如下图所示

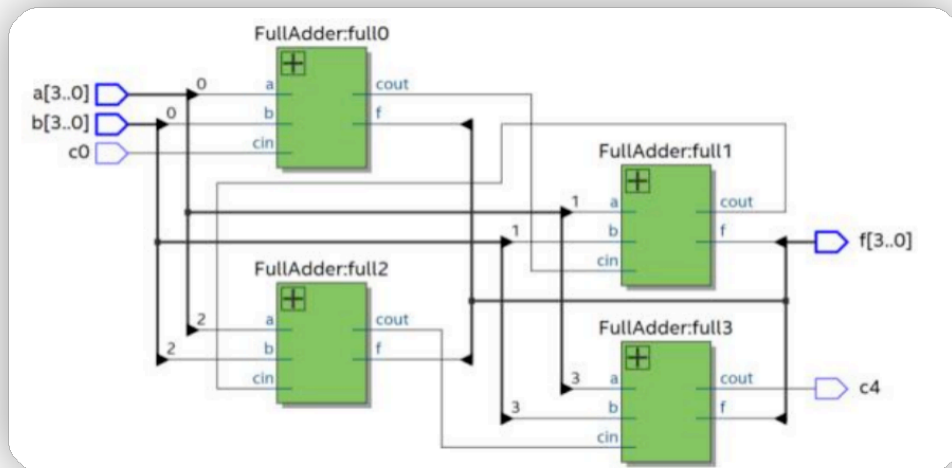
Node Name	Direction	Location
in a[3]	Input	PIN_4
in a[2]	Input	PIN_3
in a[1]	Input	PIN_2
in a[0]	Input	PIN_1
in b[3]	Input	PIN_8
in b[2]	Input	PIN_7
in b[1]	Input	PIN_6
in b[0]	Input	PIN_5
in c0	Input	PIN_12
out c4	Output	PIN_18
out f[3]	Output	PIN_17
out f[2]	Output	PIN_16
out f[1]	Output	PIN_15
out f[0]	Output	PIN_14

CPLD 生成电路

- 超前进位的四位加法器



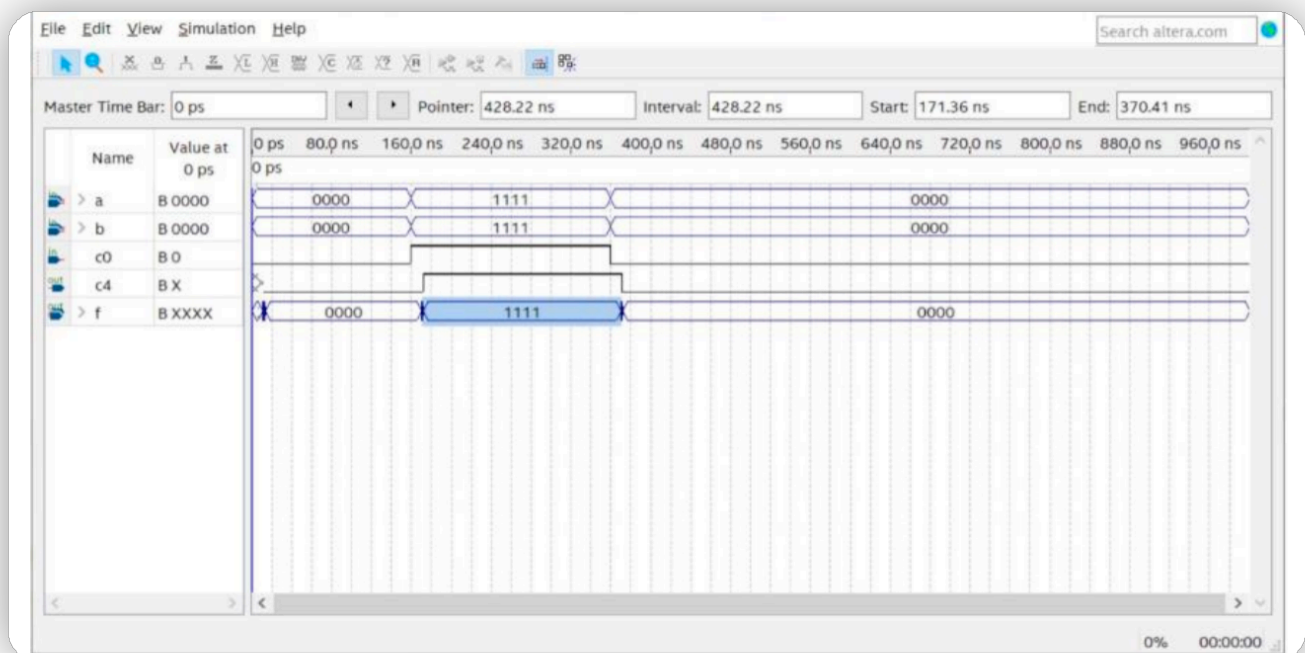
- 逐次进位的四位加法器



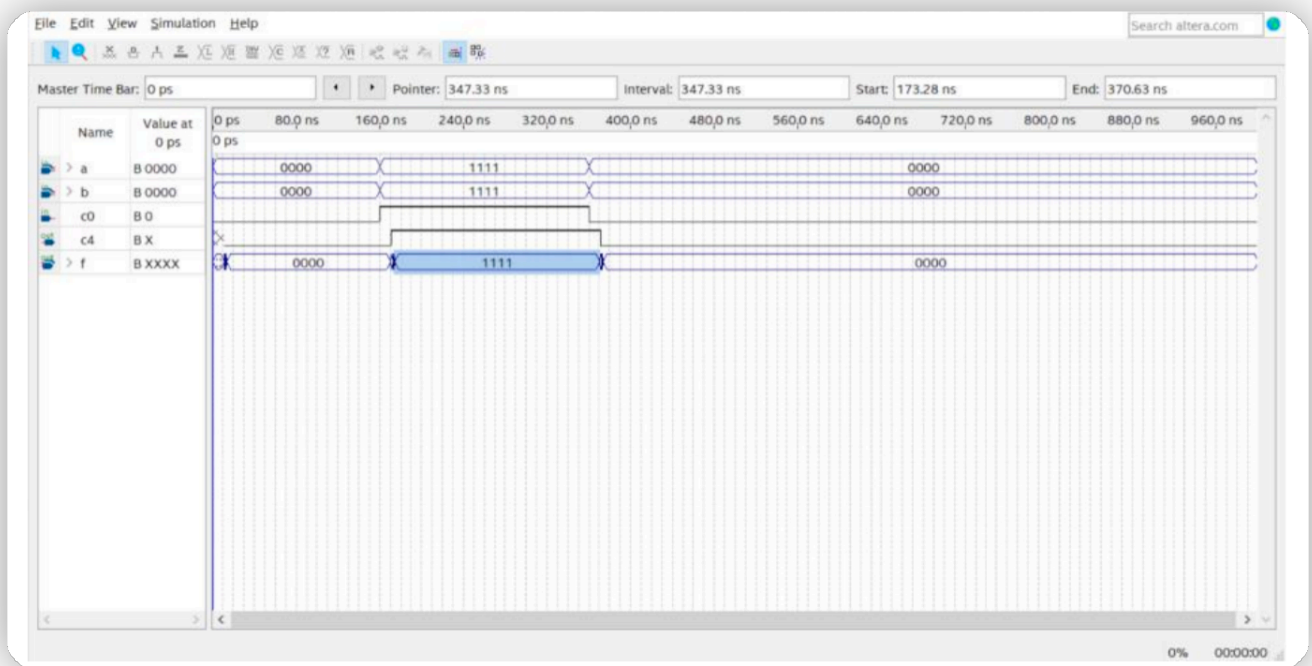
- 相同点——都是由全加器与其他简单门电路组成
- 不同点——超前进位加法器使用了更多的与非门，电路更加复杂，然而性能占有优势

仿真与延迟

- 对逐次进位加法器与超前进位加法器进行仿真测试
- 超前进位仿真，延迟为 $171.36 - 160 = 11.36ns$



- 逐次进位仿真，延迟为 $173.28 - 160 = 13.28ns$



- 本以为超前进位加法器会优势明显。可能是由于位数不够高，故而没能体现出超前进位加法器的优势。

总结

- 学会了元件实例化的映射方法
- 掌握了面向对象式的硬件编程方法，通过多个 `vhd` 文件来实现一个顶层 `entity`。