四位加法器 实验报告

2018011365 张鹤潇

一、实验目的

- 掌握组合逻辑电路的基本分析和设计方法。
- 理解半加器和全加器的工作原理并掌握利用全加器构成不同字长加法器的各种方法。
- 学习元件例化的方式进行硬件电路设计。
- 学会利用软件仿真实现对数字电路的逻辑功能进行验证和分析。

二、实验任务

- 用硬件描述语言设计半加器，用半加器构建全加器；
- 用全加器构建逐次进位全加器和超前进位全加器，并进行软件仿真。

三、原理及代码实现

1. 半加器和全加器

半加器的实现代码如下。半加和$P_n = A_n \bigoplus B_n$, 进位$G_n = A_n B_n$.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
-- 1位半加器
entity half_adder_1 is
    port(
        a, b: in std_logic; -- 加数
        p, g: out std_logic -- p: 半加和, g: 进位
    );
end half_adder_1;

architecture plus of half_adder_1 is
begin
    process(a, b)
    begin
        p <= a xor b;
        g <= a and b;
    end process;
end plus;
```

在半加器的基础上实现全加器。用component语句进行元件例化，本位和$S_n = P_n \bigoplus C_{n-1}$, 进位$C_n = P_n C_{n-1} + G_n$, 其中$S_n, G_n$是半加器组件的输出。

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
-- 一位全加器
entity full_adder_1 is
    port(
        a, b, cin: in std_logic; -- 加数, 低位进位
        s, cout: out std_logic -- 本位和, 进位
    );
end full_adder_1;

architecture plus of full_adder_1 is
    component half_adder_1
        port(
          a, b: in std_logic;
          p, g: out std_logic
        );
    end component;
    signal p, g: std_logic;
begin
    adder: half_adder_1 port map(a, b, p, g);
    process(cin, p, g)
    begin
        s <= p xor cin;
        cout <= g or (cin and p);
    end process;
end plus;
```

2. 四位全加器

四位全加器代码框架如下, 分别为逐次进位加法器和超前进位加法器实现相应的architecture, 用configuration语句来控制具体采用哪种方案。

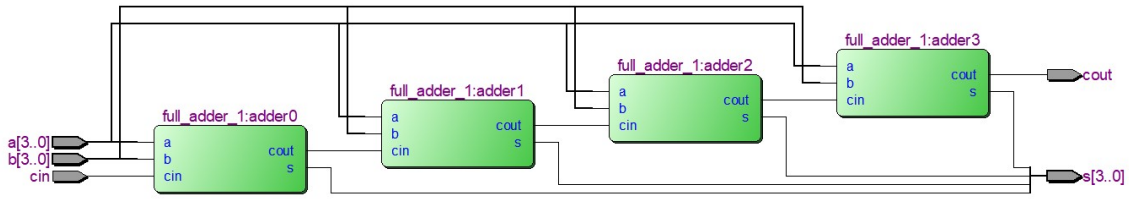
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
-- 四位全加器
entity full_adder_4 is
    port(
        a, b: in std_logic_vector(3 downto 0); -- 加数
        cin: in std_logic; -- 低位进位
        s: out std_logic_vector(3 downto 0); -- 本位和
        cout: out std_logic -- 进位
    );
end full_adder_4;

-- configuration控制构造方法
-- successive: 逐次进位 lookahead: 超前进位
configuration plus of full_adder_4 is
    for successive
    end for;
end plus;
```

逐次进位加法器

将上文实现的一位全加器逐个串联，构成逐次进位的四位加法器。

```
-- 逐次进位加法器
architecture successive of full_adder_4 is
component full_adder_1
port(
    a, b, cin: in std_logic;
    s, cout: out std_logic
);
end component;
signal c: std_logic_vector(2 downto 0); -- 记录内部进位
begin
    adder0 : full_adder_1 port map(a(0), b(0), cin, s(0), c(0));
    adder1 : full_adder_1 port map(a(1), b(1), c(0), s(1), c(1));
    adder2 : full_adder_1 port map(a(2), b(2), c(1), s(2), c(2));
    adder3 : full_adder_1 port map(a(3), b(3), c(2), s(3), cout);
end;
```



超前进位加法器

定义传递信号$P_n = A_n \oplus B_n$，进位产生信号$G_n = A_n B_n$，根据公式

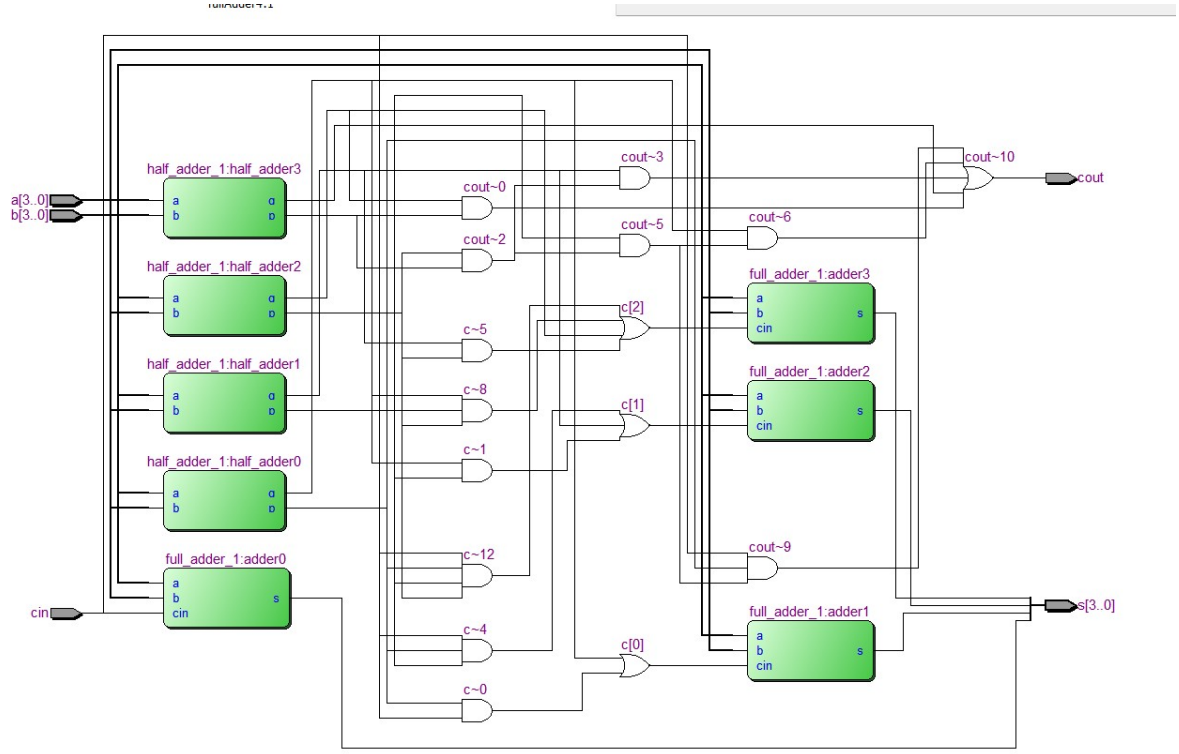
$$
\begin{aligned}
C_0 &= G_0 + P_0 C_1 \\
C_1 &= G_1 + P_1 G_0 + P_1 P_0 C_1 \\
C_2 &= G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_1 \\
C_3 &= G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 C_1
\end{aligned}
$$

设计超前进位加法器，超前进位运算部分由四个1位半加器完成。

```
-- 超前进位加法器
architecture lookahead of full_adder_4 is
    component full_adder_1
        port(
            a, b, cin: in std_logic;
            s, cout: out std_logic
        );
    end component;
    component half_adder_1
        port(
            a, b: in std_logic;
            p, g: out std_logic
        );
    end component;
    signal p, g: std_logic_vector(3 downto 0);
```

```
    signal c: std_logic_vector(2 downto 0);
begin
    half_adder0 : half_adder_1 port map(a(0), b(0), p(0), g(0));
    half_adder1 : half_adder_1 port map(a(1), b(1), p(1), g(1));
    half_adder2 : half_adder_1 port map(a(2), b(2), p(2), g(2));
    half_adder3 : half_adder_1 port map(a(3), b(3), p(3), g(3));
    adder0 : full_adder_1 port map(a(0), b(0), cin, s(0));
    adder1 : full_adder_1 port map(a(1), b(1), c(0), s(1));
    adder2 : full_adder_1 port map(a(2), b(2), c(1), s(2));
    adder3 : full_adder_1 port map(a(3), b(3), c(2), s(3));
    process(cin, p, g)
    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);
        cout <= 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;
```

生成电路如下：

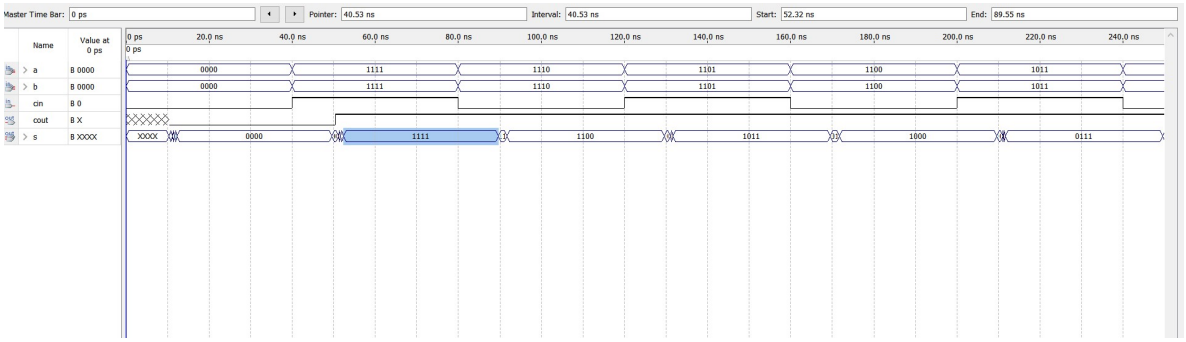


四、仿真实验

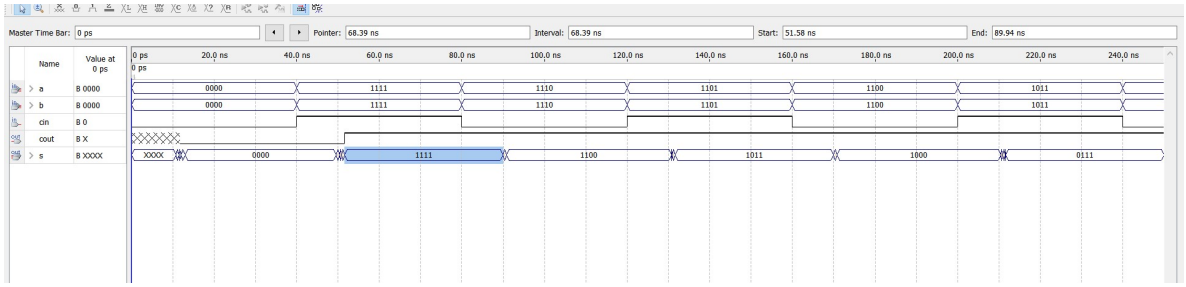
本机仿真

更改四位全加器的configuration条件，在本机上进行time simulation，测试程序正确性和延迟时间。

逐次进位加法器仿真结果如下：



超前进位加法器仿真结果如下:

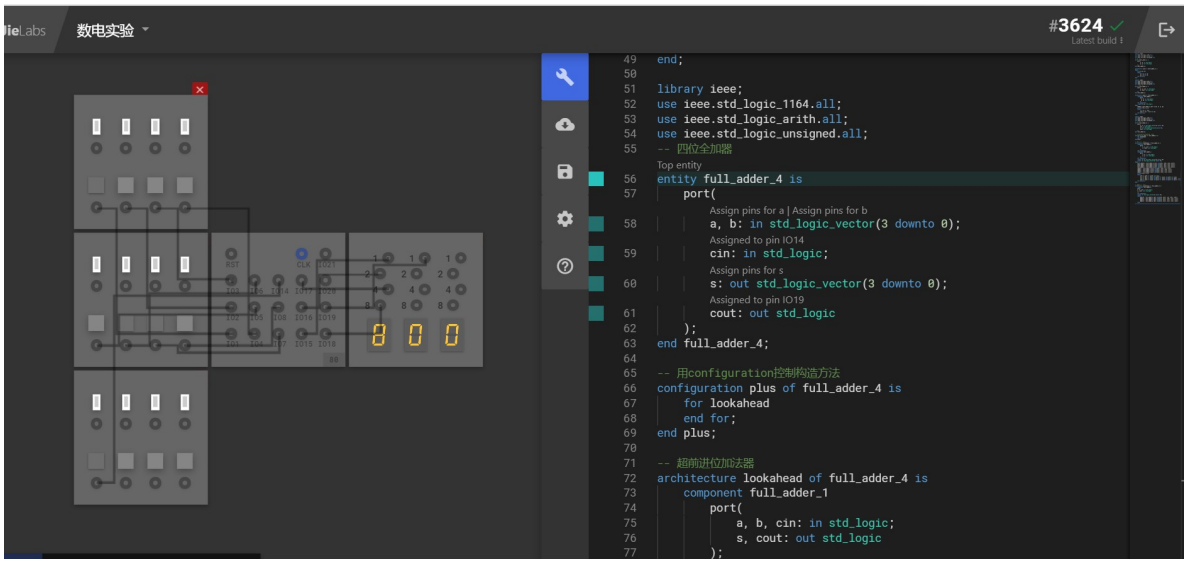


由此可以验证程序的正确性。

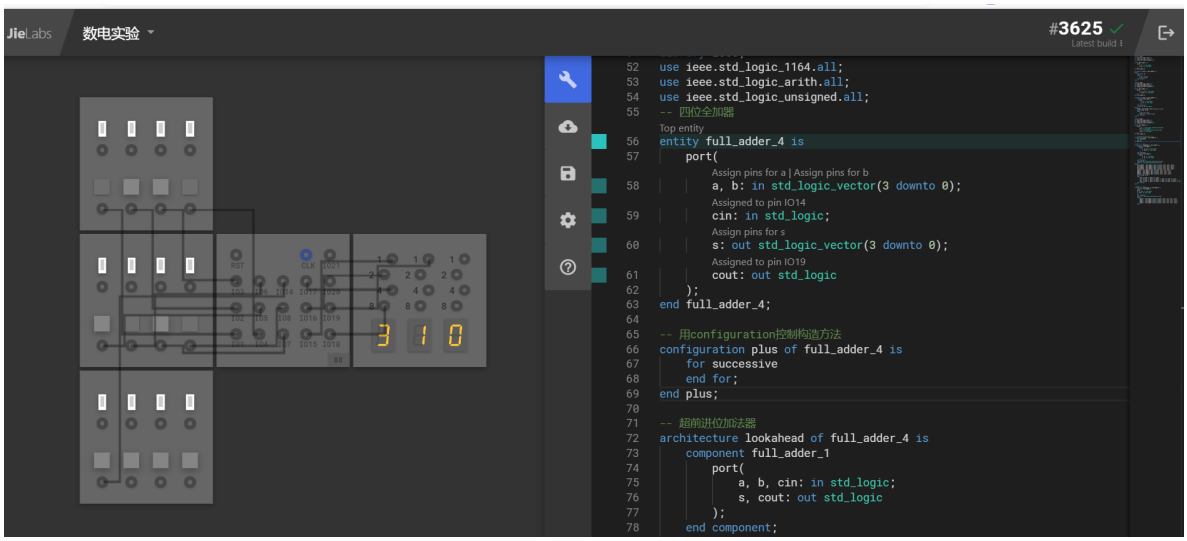
当输入从0000+0000(+0000)变化为1111+1111(+0001)时，逐次进位加法器延迟12.32 ns, 超前进位加法器延迟11.58 ns.

JieLab实验

在jielab上实验如图，左边三组开关分别控制a,b,cin的输入，右边两个数码管显示四位全加器的输出结果。



如上图所示，使用超前进位全加器，0001+0110(+0001)=1000(+00000).



如上图所示，使用逐次进位全加器，1001+1010(+0000)=0011(+10000).

五、总结反思

经过本次实验，我对元件例化设计有了更深的了解。当同一功能有多种实现时，用configuration语句来加以控制十分方便。看到自己写的代码生成复杂的CPLD电路，确实很有成就感。