四位加法器实验报告

计73 郑林楷 2017011474

实验目的

1. 掌握组合逻辑电路的基本分析和设计方法。
2. 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。
3. 学习元件例化的方式进行硬件电路设计。
4. 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。

实验任务

1. 设计实现逐次进位加法器，进行软件仿真并在实验平台上测试。
2. 设计实现超前进位加法器，进行软件仿真并在实验平台上测试。
3. 使用 VHDL 自带的加法运算实现一个 4 位全加器。
4. 查看逐次进位加法器、超前进位加法器和 VHDL 自带加法器在 CPLD 中生成的电路，并比较这三者的异同。

实验内容

实现上，将四位加法器设计成一个元件，并为每种不同的内部实现方式编写对应的 `architecture`，最后通过 `configuration` 来控制四位加法器内部实现方案的选择。四位加法器的 VHDL 如下：

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity full_adder_4 is
    port(
        a, b: in std_logic_vector(3 downto 0);
        cin: in std_logic;
        s: out std_logic_vector(3 downto 0);
        cout: out std_logic
    );
end full_adder_4;

configuration plus of full_adder_4 is
    for carry_plus
    end for;
end plus;
```

1. 逐次进位加法器

原理与设计

使用一位半加器实现一位全加器，再使用一位全加器级联实现，将前一级的进位输出传递给下一级。

实现上，将一位全加器设计为了一个元件，并在逐次进位加法器中进行例化。

实验代码

一位半加器的 VHDL 代码如下：

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity half_adder_1 is
    port(
        a, b: in std_logic;
        p, g: out std_logic
    );
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;
```

一位全加器的 VHDL 代码如下：

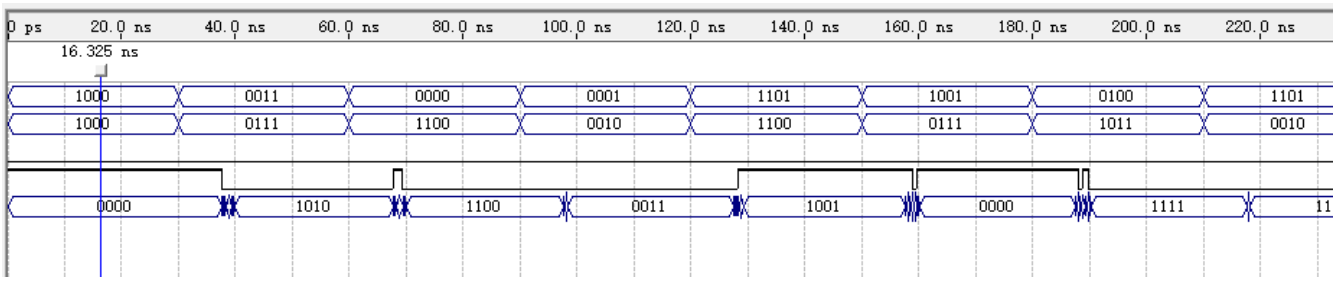
```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity full_adder_1 is
    port(
        a, b, cin: in std_logic;
        s, cout: out std_logic
    );
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;
```

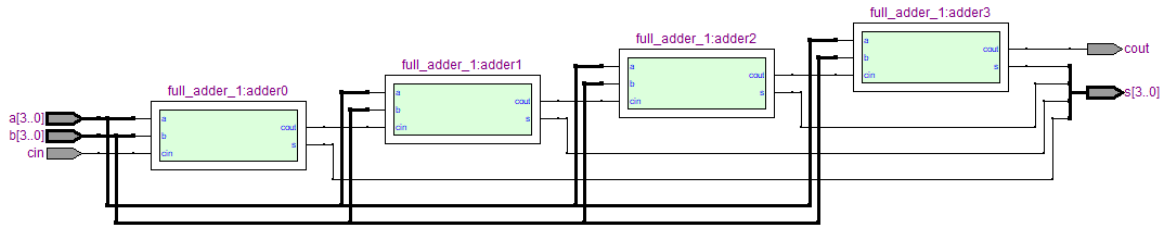
逐次进位加法器的 VHDL 代码如下：

```
architecture ripple_plus 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);
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 ripple_plus;
```

仿真和生成电路



生成的 CPLD 电路如下：



2. 超前进位加法器

原理与设计

定义进位传递信号 P 和进位产生信号 G 如下：

$$P_n = A_n \oplus B_n$$
$$G_n = A_n B_n$$

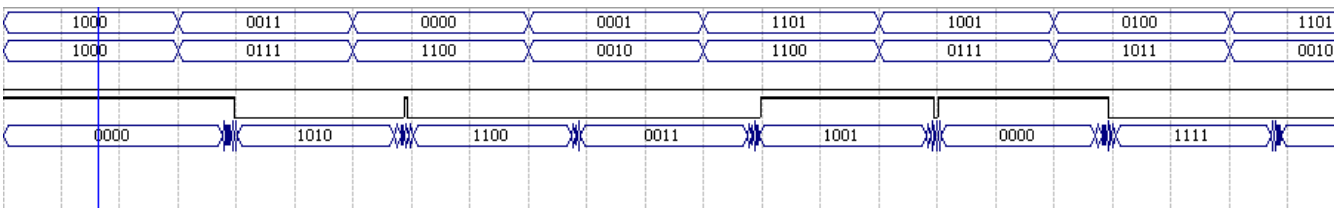
使用快速进位扩展器，输入 P、G 以及 C_{in}，输出 $C_0 \sim C_3$，并传递给四个全加器。四 个全加器可以并行计算求出答案。

实验代码

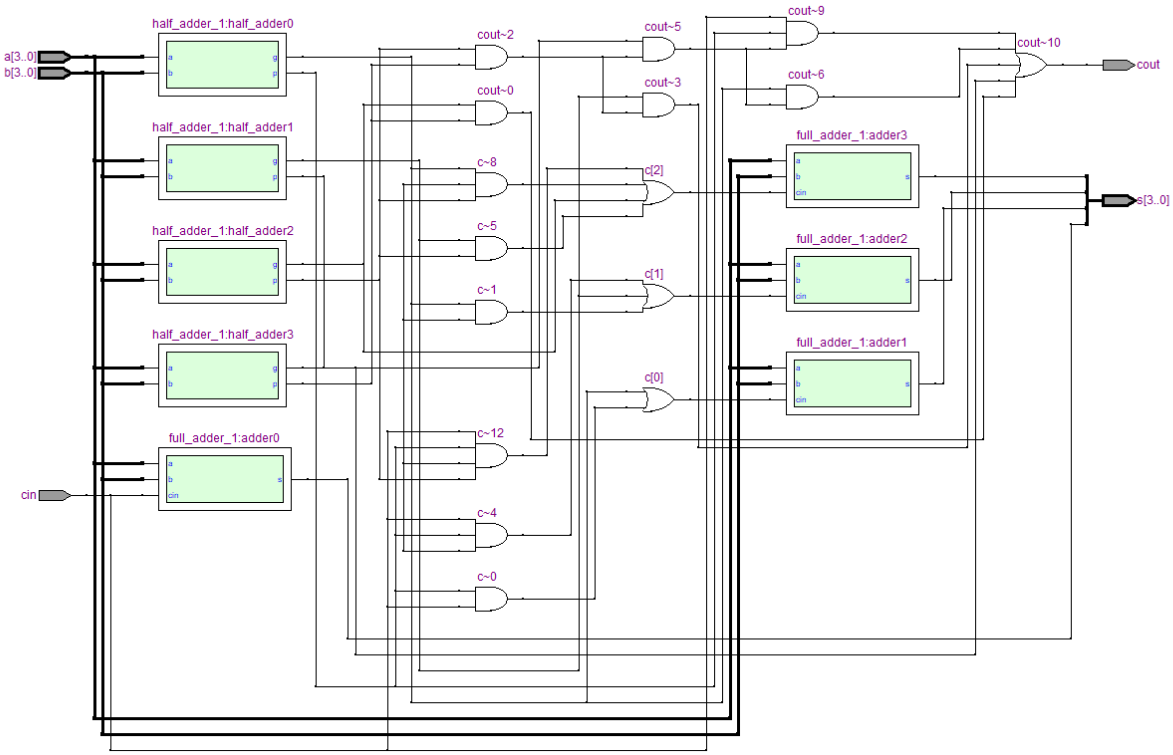
超前进位加法器的 VHDL 代码如下：

```
architecture carry_plus 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 carry_plus;
```

仿真与生成电路



生成的 CPLD 电路如下：



3. VHDL 自带加法器

原理与设计

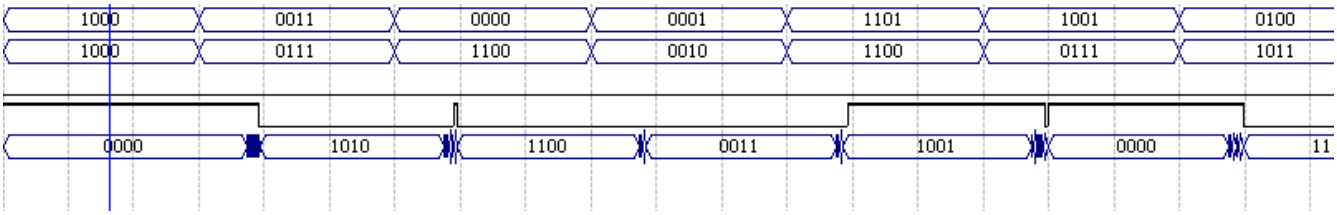
使用 VHDL 中的整数加法运算直接实现加法器。

实验代码

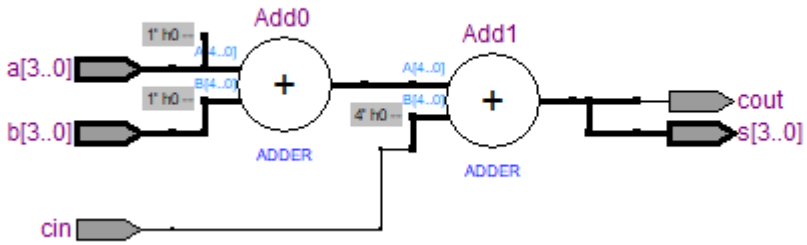
VHDL 自带加法器的 VHDL 代码如下：

```
architecture sys_plus of full_adder_4 is
    signal sys: std_logic_vector(4 downto 0);
begin
    process(cin, a, b)
    begin
        sys <= "00000" + a + b + cin;
    end process;
    process(sys)
    begin
        s <= sys(3 downto 0);
        cout <= sys(4);
    end process;
end sys_plus;
```

仿真与生成电路



生成的 CPLD 电路如下：



总结与心得

这次实验下来非常顺利，基本上没有遇到过什么大的问题。在选择哪种实现方式的时候我学习并使用了 `configuration`，同时也第一次学会了生成 CPLD 电路，感觉特别有意思。在编写超前进位加法器的过程中我发现其延迟好像和逐次进位加法器差的不是很多，一开始还以为是正常现象，但在我生成了 CPLD 电路后我才发现我超前进位加法器写错了，导致其无法并行。看看 CPLD 电路是否和预期的相符合，这也是可以作为检查的一种手段。