四位加法器实验报告

计 71 张程远 2017011429

2019.4.25

一、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的加法器的各种方法。
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逻辑电路的逻辑功能进行验证和分析。

二、实验内容

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进行测试。
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进行测试。

三、代码及原理

1. 一位加法器元件定义

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

entity half_adder is
    port(a, b: in std_logic;
    s, c: out std_logic);
end half_adder ;

architecture bhv1 of half_adder is
begin
    c <= a and b ;
    s <= a xor b ;
end bhv1 ;

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

```
use ieee.std_logic_unsigned.all;

entity adder1 is
    port(
        a,b,cin:in std_logic;
        s,cout:out std_logic;
        p,g:buffer std_logic
    );
end adder1;
architecture plus of adder1 is
    component half_adder
        port(a, b: in std_logic ;
        s, c: out std_logic) ;
    end component ;
begin
    u1: half_adder port map(a,b,p,g);
    process(cin,p,g)
    begin
        s <=cin xor p;
        cout <=g or (p and cin);
    end process;
end plus;
```

工作原理：首先实现一位半加器 half_adder，将其用元件例化的方式组成一位全加器，其中 a，b 是输入，p 和 g 是中间变量，最后的结果是 s 和 cout。直接采用课本上的公式进行运算即得。

2. 逐次进位加法器

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

entity adder4 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 adder4;

architecture successive of adder4 is
    component adder1
        port(
            a,b,cin:in std_logic;
            s,cout:out std_logic;
            p,g:buffer std_logic
        );
    end component;
    signal p,g,c:std_logic_vector(3 downto 0);
begin
    fa0:adder1 port map(a(0),b(0),cin,s(0),c(0),p(0),g(0));
    fa1:adder1 port map(a(1),b(1),c(0),s(1),c(1),p(1),g(1));
    fa2:adder1 port map(a(2),b(2),c(1),s(2),c(2),p(2),g(2));
    fa3:adder1 port map(a(3),b(3),c(2),s(3),cout,p(3),g(3));
end successive;
```

工作原理：对一位加法器使用元件例化的方式，采用级联的方式构造四位加法器。

3. 超前进位加法器

```
architecture ahead of adder4 is
    component adder1
        port(
            a,b,cin:in std_logic;
            s,cout:out std_logic;
            p,g:buffer std_logic
        );
    end component;
    signal p,g,c:std_logic_vector(3 downto 0);
```

```
begin
    fa0:adder1 port map(a(0),b(0),cin,s => s(0),p => p(0),g => g(0));
    fa1:adder1 port map(a(1),b(1),c(0),s => s(1),p => p(1),g => g(1));
    fa2:adder1 port map(a(2),b(2),c(1),s => s(2),p => p(2),g => g(2));
    fa3:adder1 port map(a(3),b(3),c(2),s => s(3),p => p(3),g => g(3));
    process(p,g)
    begin
        c(0) <= g(0) or (cin and p(0));
        c(1) <= g(1) or (p(1) and g(0)) or (cin and p(0) and p(1));
        c(2) <= g(2) or (p(2) and g(1)) or (p(2) and p(1) and g(0)) or (cin and
p(2) and p(1) and p(0));
        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 (cin and p(3) and p(2) and p(1) and p(0));
    end process;
end ahead;
```

工作原理：仍然采用元件例化的方式，并使用书本上的超前进位方式，通过计算 p 和 g 得到答案。

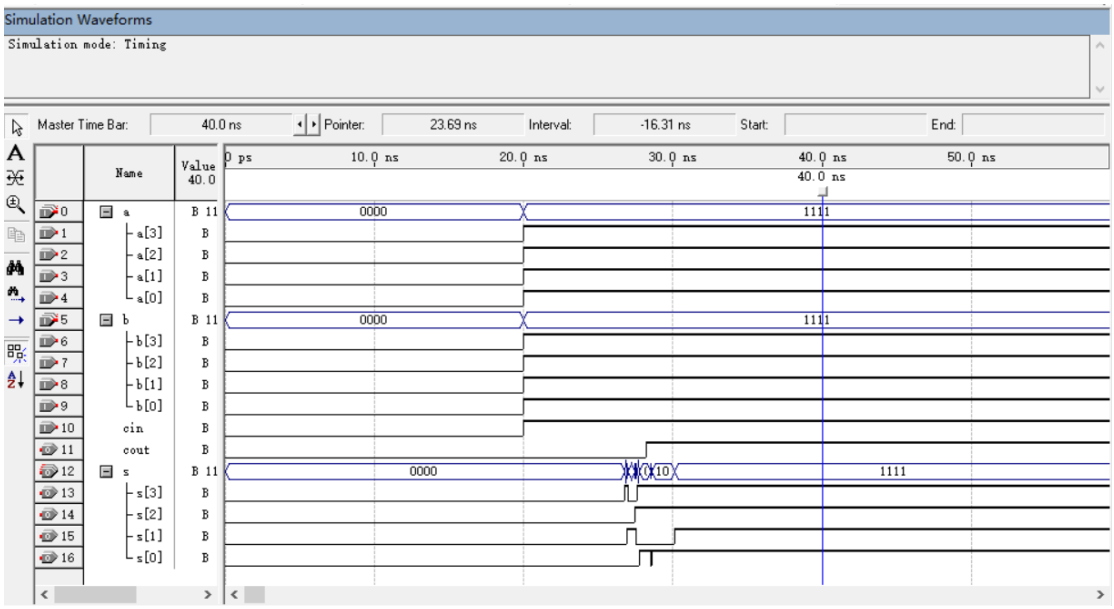
4. 系统自带加法

```
architecture system of adder4 is
    signal sys:std_logic_vector(4 downto 0);
begin
    process(a,b)
    begin
        sys<="00000"+a+b+cin;
    end process;
    process(sys)
    begin
        cout<=sys(4);
        s<=sys(3 downto 0);
    end process;
end system;
```

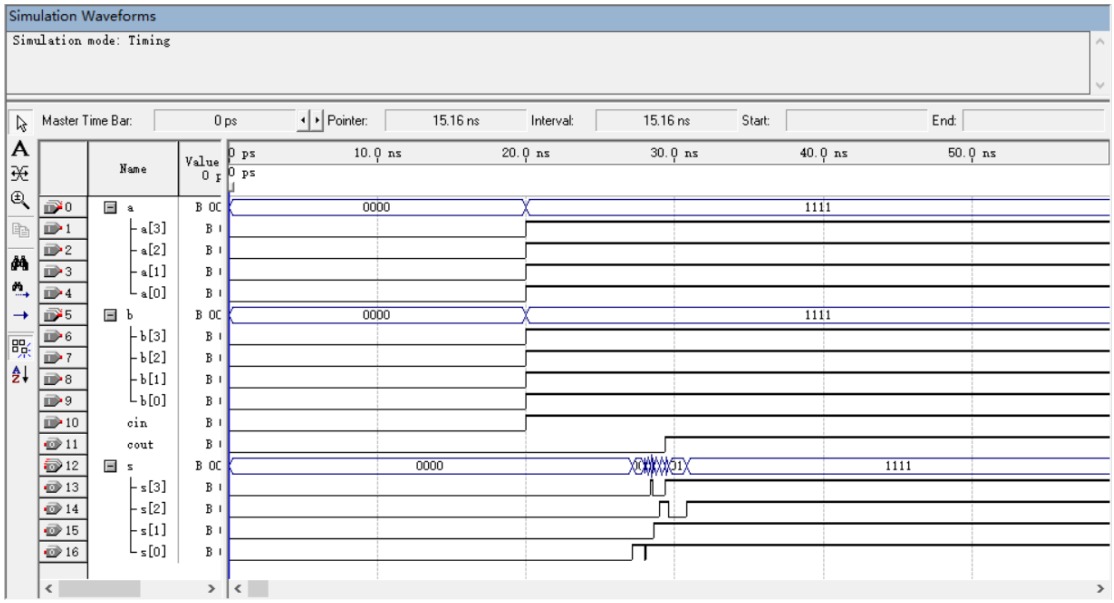
工作原理：使用系统自带的“+”直接进行运算，使用“00000”将四位加数变为五位，从而完成加法运算。这里没有使用元件例化。

四、仿真结果

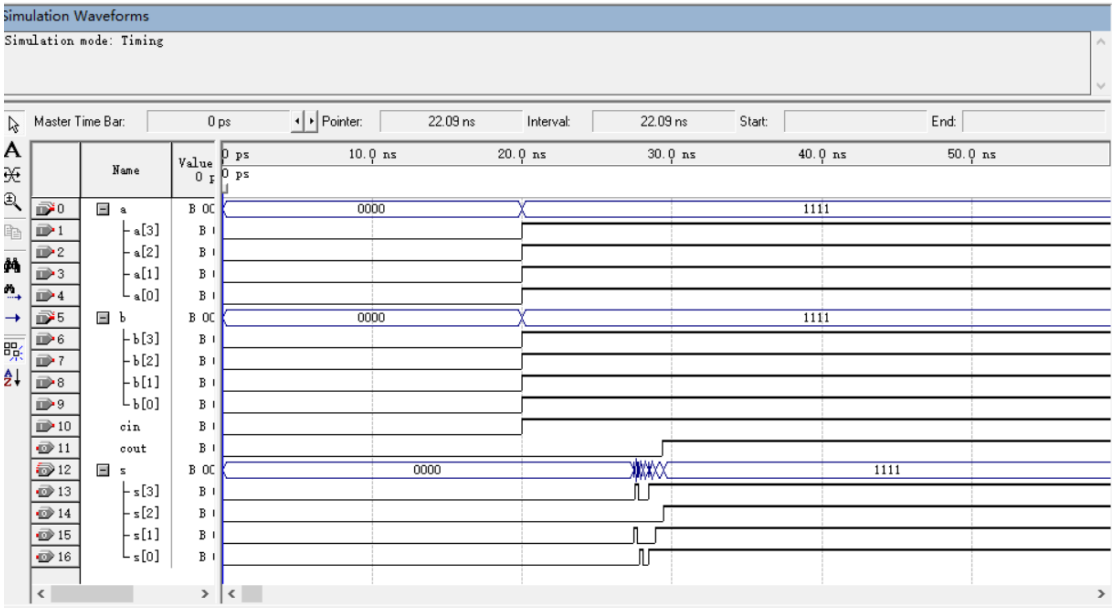
1. 逐次进位加法器仿真结果



2. 超前进位加法器仿真结果



3. 系统自带加法仿真结果



五、结果分析与小结

1. 结果分析

忽略延迟部分，三个加法器都给出了正确的结果；考虑延迟部分，理论上，超前进位加法器应当比串行加法器的延迟小，但是由于输入的加法都仅限于 4 位，位数比较少，超前进位反而要进行比较多的计算，所以相比而言并未体现出太多优势,整体延迟时间都在 5-6ns 左右；系统自带的加法运算相比于前两种延迟时间明显更少，约在 3ns 左右，很有可能是在加号的源代码实现上存在一定的优化。

2. 实验小结

本次实验是我第二次做 CPLD 实验，相比于上一次的手忙脚乱，这次我对 quartus 更加熟悉，遇到的问题和解决时间也相应更少，所以本次实验做起来比较轻车熟路。这次实验学习了 quartus 自带仿真功能的使用，有助于在我实际接线之前检查代码的正确性，有助于提升我的实验能力。期待下一次的数电实验！