四位加法器

2017011620 计73 李家昊

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。

实验内容

基础内容

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。

研究内容

1. 使用VHDL自带的加法运算实现一个4位全加器。

代码分析

一位加法器元件

```
-- full_adder_1.vhd --

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 -- 1 bit full adder
    port(
        a,b,cin:in std_logic; -- a,b: 1-bit number. cin: carry input
        s,cout:out std_logic; -- s: sum result. cout: carry output
        p,g:buffer std_logic  -- p: carry propagate function. g: carry generate function.
    );
end full_adder_1;

architecture add of full_adder_1 is
begin
    process(a,b)
    begin
        p <= a xor b; -- compute carry propagate function.
        g <= a and b; -- compute carry generate function.
```

```
    end process;
    process(cin,p,g)
    begin
        s <= p xor cin; -- compute sum result
        cout <= (p and cin) or g; -- compute carry output
    end process;
end add;
```

实现了一位加法器，其中a,b是1位加数，cin为进位输入，s为全加和，cout为高位进位，p,g分别为传递进位函数和产生进位函数。其中定义两个进程，一个进程计算p,g，供超前进位加法器使用，另一个进程利用p,g计算s和cout。

在超前进位加法器的实现中，需要将p,g与signal绑定，编译器要求p,g设为buffer，若设成out则默认为不可读状态，因此这里p,g设为buffer。

四位加法器实体

```
-- full_adder_4.vhd --

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;
```

定义了四位全加器实体，a,b为四位加数，cin为进位输入，cout为进位输出，s为全加和。

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;
            p,g:buffer std_logic
        );
    end component;
    signal c:std_logic_vector(3 downto 0);

begin
    fa1_0:full_adder_1 port map(a(0),b(0),cin,s(0),c(0));
    fa1_1:full_adder_1 port map(a(1),b(1),c(0),s(1),c(1));
    fa1_2:full_adder_1 port map(a(2),b(2),c(1),s(2),c(2));
    fa1_3:full_adder_1 port map(a(3),b(3),c(2),s(3),cout);
end successive;
```

实现逐次进位加法器。用元件例化方法，实例化四个一位全加器，将四位加法器的输入输出端分别绑定到一位全加器上，将前一个全加器的进位输出端捆绑到临时变量中，再捆绑到后一个全加器的进位输入端，实现逐次进位加法器。

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;
            p,g:buffer std_logic
        );
    end component;
    signal p,g,c: std_logic_vector(3 downto 0); -- p: prop func buffer. g: gen func buffer.
c: carry buffer.

begin
    fa1_0:full_adder_1 port map(a(0),b(0),cin,s(0),p=>p(0),g=>g(0));
    fa1_1:full_adder_1 port map(a(1),b(1),c(0),s(1),p=>p(1),g=>g(1));
    fa1_2:full_adder_1 port map(a(2),b(2),c(1),s(2),p=>p(2),g=>g(2));
    fa1_3:full_adder_1 port map(a(3),b(3),c(2),s(3),p=>p(3),g=>g(3));
    process(p,g,c)
    begin -- compute look-ahead carry
        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 lookahead;
```

实现超前进位加法器。将一位加法器中的产生进位函数G(n)和传递进位函数P(n)绑定到signal中，然后快速计算出每个进位，激活每个一位全加器的第二个进程，通过p,g计算出全加和，完成加法运算。

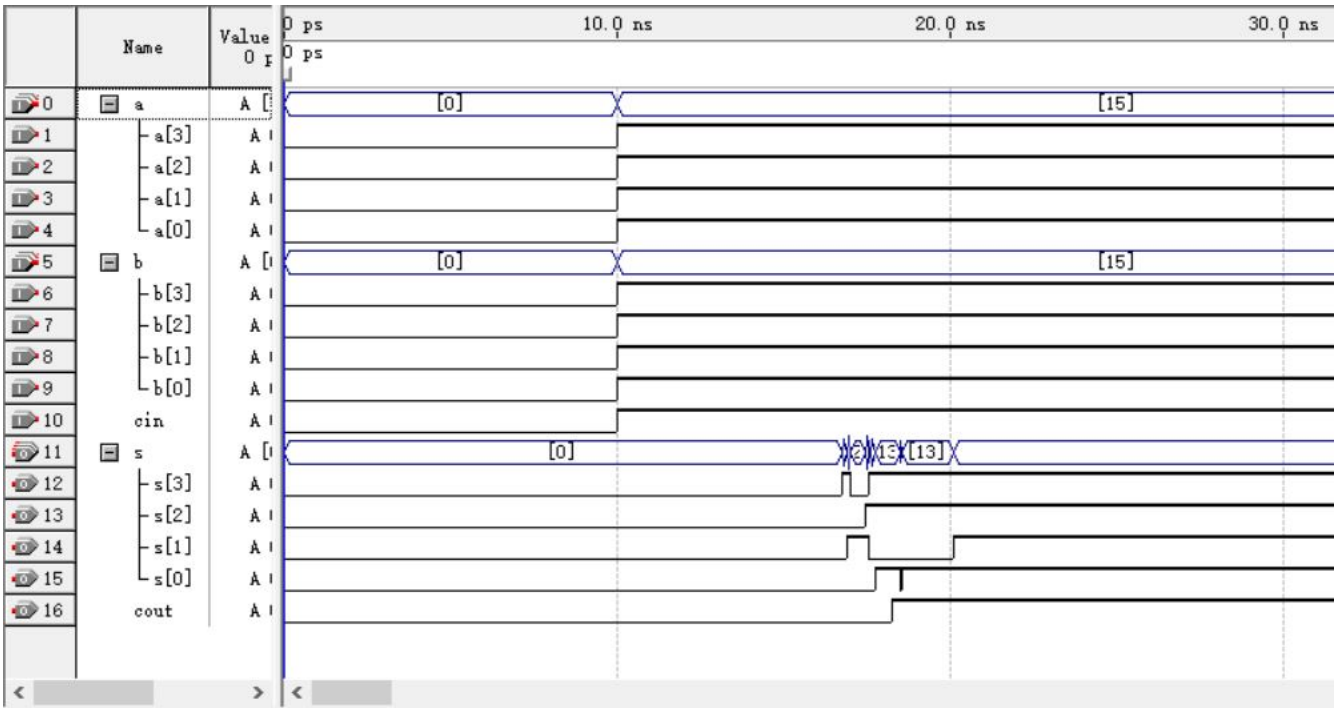
系统自带加法器

```
architecture system of full_adder_4 is
    signal result_full: std_logic_vector(4 downto 0);
begin
    process (a,b,cin)
    begin
        result_full <= "00000" + a + b + cin; -- using operator '+' defined in system
        s <= result_full(3 downto 0); -- get sum result
        cout <= result_full(4); -- get the carry output
    end process;
end system;
```

实现系统自带加法器。定义一个buffer存放5位的加法结果，通过引入"00000"，将a,b,cin转换为5位vector进行求和。最终取结果的后四位作为全加和输出，第一位作为进位输出。

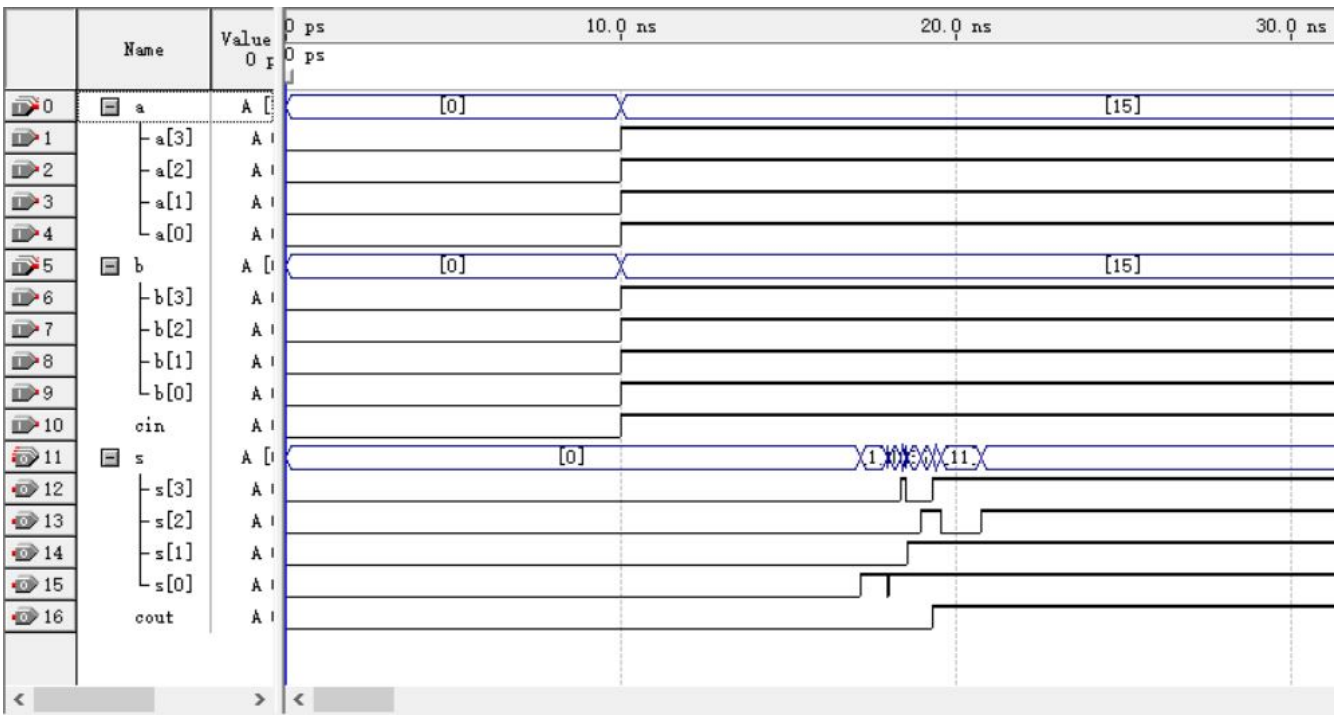
仿真结果

逐次进位加法器



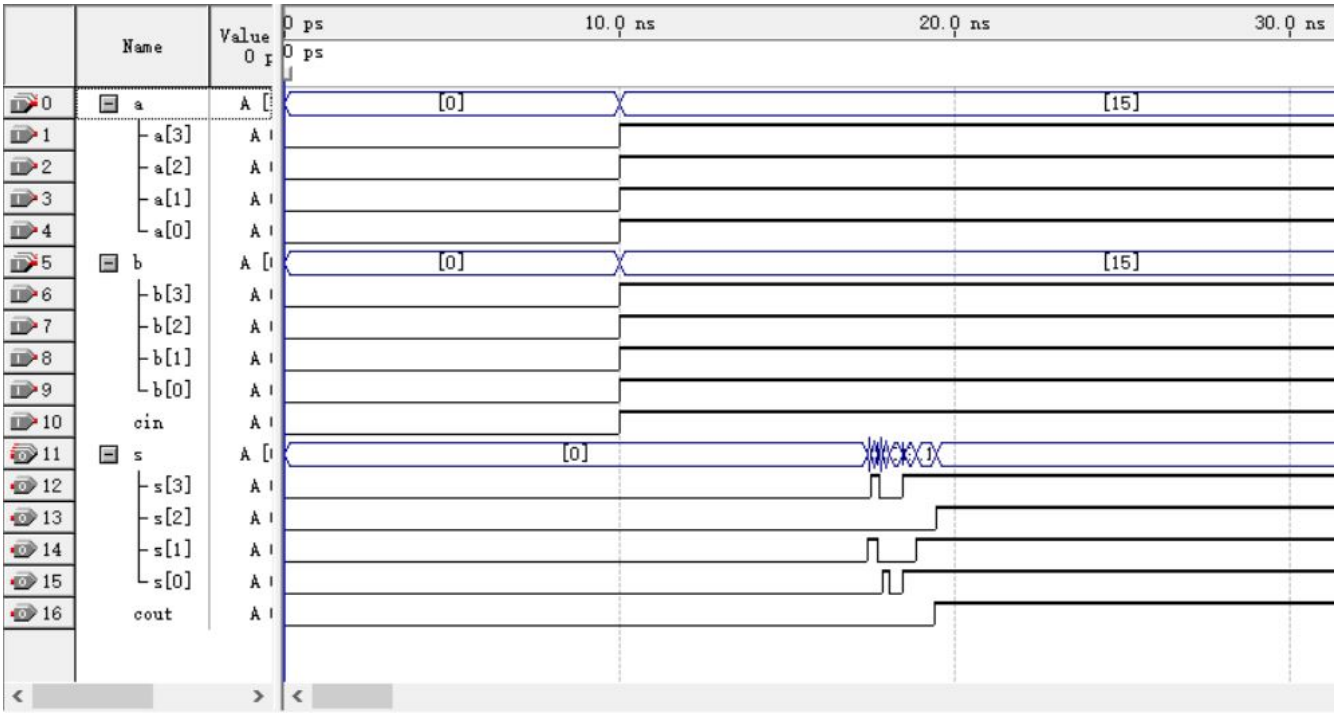
令输入信号从全0变为全1，观察输出信号，输出延时为10.1ns。

超前进位加法器



令输入信号从全0变为全1，观察输出信号，输出延时为10.75ns。与逐次进位加法器差异不大。

系统自带加法器



令输入信号从全0变为全1，观察输出信号，输出延时为9.45ns。可以看出，系统自带加法器比自己实现的逐次进位加法器和超前进位加法器速度更快，可见CPLD对加法运算进行了底层优化。

功能测试结果

三个加法器均能产生正确结果，与仿真结果相同。单次加法所需时间较短，仅靠人的感官难以感受到延时的差异。

实验小结

1. 掌握了元件例化的方法，体会到硬件设计的模块化思想。
2. 加深了对半加器和全加器的理解，学会了用n个一位全加器构成一个n位加法器的技能。
3. 掌握了逐次进位加法器和超前进位加法器的原理，并用代码加以实现。
4. 掌握了Quartus的仿真功能，仿真使电路的调试更加方便。