四位加法器

计01　容逸朗　2020010869

1　实验内容

1. 设计半加器，然后用半加器构建全加器；
2. 用全加器构建逐次进位加法器并进行仿真测试；
3. 用全加器构建超前进位加法器并进行仿真测试；
4. 使用 **VHDL** 自带的加法运算实现一个四位加法器。

2　实验原理

2.1　半加器

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

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

architecture bhv of halfadder is
begin
  process(a, b)
  begin
    s <= a xor b;
    c <= a and b;
  end process;
end bhv;
```

工作原理：对于任意加数 A, B ，我们有和 $S = A \oplus B$ 以及进位 $C = AB$。利用这个思路，我们可以记输入为 a, b ，而和与进位分别记为 s, c，那么不难得到上面的代码。

2.2 一位全加器

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

entity fulladder is
  port (
    a, b, ci: in std_logic;
    s, co: out std_logic;
    p, g: buffer std_logic
  );
end fulladder;

architecture bhv of fulladder is
  component halfadder is
    port (
      a, b: in std_logic;
      s, c: out std_logic
    );
  end component;
  signal c2: std_logic;
begin
  u1: halfadder port map(a, b, p, g); -- 处理加数
  u2: halfadder port map(p, ci, s, c2); -- 计算和
  co <= g or c2; -- 计算进位
end bhv;
```

工作原理：对于一位全加器，我们需要输入加数 a,b 以及上一位的进位 ci ，然后通过操作得到和 s 与进位 co 。计算时可以将 a,b 用半加器相加，得到临时和 p 与临时进位 g ，然后将 p,ci 用半加器相加得到一位加法的和 s 和另一个临时进位 $c2$ ，最后若有一位临时进位不为零，则进位成立，输出结果。

为了方便实现超前加法器，上面的代码也保存了 a,b 的和与进位为两个变量 p,g 。

2.3 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 fulladder4a is
  port (
    a, b: in std_logic_vector(3 downto 0);
```

```
    s: out std_logic_vector(3 downto 0);
    ci: in std_logic;
    co: out std_logic
  );
end fulladder4a;

architecture bhv of fulladder4a is
  component fulladder is
    port (
      a, b, ci: in std_logic;
      s, co: out std_logic;
      p, g: buffer std_logic
    );
  end component;
  signal c: std_logic_vector(2 downto 0);
begin
  f1: fulladder port map(a(0), b(0), ci, s(0), c(0)); -- 计算第 0 位
  f2: fulladder port map(a(1), b(1), c(0), s(1), c(1)); -- 计算第 1 位
  f3: fulladder port map(a(2), b(2), c(1), s(2), c(2)); -- 计算第 2 位
  f4: fulladder port map(a(3), b(3), c(2), s(3), co); -- 计算第 3 位
end bhv;
```

工作原理：对于逐次进位加法器，只需要将加数 a,b 与上一位的进位 ci 用一位全加器相加，便可得到对应的和 s 与进位 co。

2.4 超前进位器

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

entity adv is
  port (
    p, g: in std_logic_vector(3 downto 0);
    ci: in std_logic;
    co: out std_logic_vector(3 downto 0)
  );
end adv;

architecture bhv of adv is
begin
  -- 第 0 位进位
  co(0) <= g(0) or (p(0) and ci);
  -- 第 1 位进位
```

```
  co(1) <= g(1) or (p(1) and g(0)) or (p(1) and p(0) and ci);
  -- 第 2 位进位
  co(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 ci);
  -- 第 3 位进位
  co(3) <= 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 ci);
end;
```

工作原理：这里直接利用书 P.104 列出的公式，来计算出每一位的对应进位是什么。代码需要输入 p 和 g，即对应位置的加数利用半加器得到的结果，然后解得各数位的进位。

2.5 超前进位加法器

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

entity lca is
  port (
    a, b: in std_logic_vector(3 downto 0);
    s: out std_logic_vector(3 downto 0);
    ci: in std_logic;
    co: out std_logic
  );
end lca;

architecture bhv of lca is
  component fulladder is
    port (
      a, b, ci: in std_logic;
      s, co: out std_logic;
      p, g: buffer std_logic
    );
  end component;
  component adv is
  port (
      p, g: in std_logic_vector(3 downto 0);
      ci: in std_logic;
      co: out std_logic_vector(3 downto 0)
    );
  end component;
  signal p, g, c: std_logic_vector(3 downto 0);
  signal tmp: std_logic;
```

```
begin
  f1: fulladder port map(a(0), b(0), ci, s(0), tmp, p(0), g(0));
  f2: fulladder port map(a(1), b(1), c(0), s(1), tmp, p(1), g(1));
  f3: fulladder port map(a(2), b(2), c(1), s(2), tmp, p(2), g(2));
  f4: fulladder port map(a(3), b(3), c(2), s(3), tmp, p(3), g(3));
  ad: adv port map(p, g, ci, c);
  co <= c(3);
end bhv;
```

工作原理：与逐位加法器类似，首先利用一位全加器解得对应位置的临时和与进位 p, g ，然后将结果放入超前加法器中计算，然后取得上一位的进位后再计算对应位置的和。

2.6 系统加法器

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

entity add is
  port (
    a, b: in std_logic_vector(3 downto 0);
    s: out std_logic_vector(3 downto 0);
    ci: in std_logic;
    co: out std_logic
  );
end add;

architecture bhv of add is
  signal tmp: std_logic_vector(4 downto 0);
begin
  process(a, b, ci)
  begin
    tmp <= "00000" + a + b + ci;
  end process;
  process(tmp)
  begin
    co <= tmp(4);
    s <= tmp(3 downto 0);
  end process;
end bhv;
```

工作原理：将初值设为 `00000` ，这样进行加法运算后便可以得到一位进位（对应向量的最高位）和四位的结果。（对应向量的其他位置）

3 电路功能测试

3.1 实际操作

3.1.1 实验用具

本次使验使用了数字逻辑实验平台中的**带译码数码管**，两个**四输入开关**，一个**可编程模块**和一个**触发器**。

3.1.2 实验步骤

根据书上的端口设计接线。然后分别测试：

1. 0 加 0

   这时结果为 0，符合预想。

2. 0 加 0，进位 (clk) 为 1

   这时结果为 1，符合预想。

3. 4 加 5

   这时结果为 9，符合预想。

4. 4 加 5，进位 (clk) 为 1

   这时数码管灯灭，符合预想。

5. 8 加 7

   这时数码管灯灭，符合预想。

6. 8 加 7 ，进位 (clk) 为 1

   这时结果为 0，进位对应的二极管亮，符合预想。

7. 8 加 8

   这时结果为 0，进位对应的二极管亮，符合预想。

8. 15 加 10

   这时结果为 9，进位对应的二极管亮，符合预想。

9. 15 加 10，进位 (clk) 为 1

   这时数码管灯灭，进位对应的二极管亮，符合预想。

至此，代码和接线均无误。

3.2 仿真实验

3.2.1 实验步骤

实验开始前算式为 $0000+0000+0=0000$ ，为了测得最大延迟值，我们会将算式更改为 $1111+1111+1=11111$ ，以此测得最大延迟值。

3.2.2 仿真结果

随意选择几个加数作仿真，可以得到如下结果：

1. 逐次进位加法器

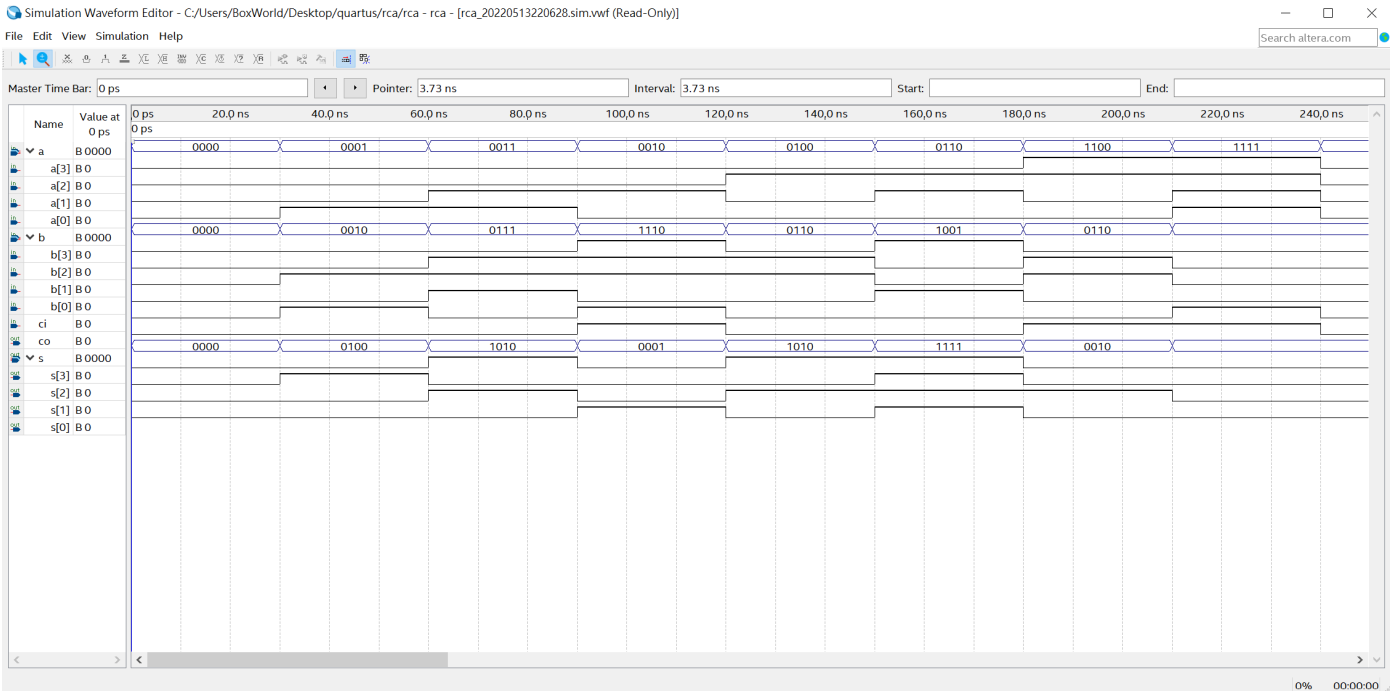

图1　逐次进位加法器仿真结果

2. 超前进位加法器

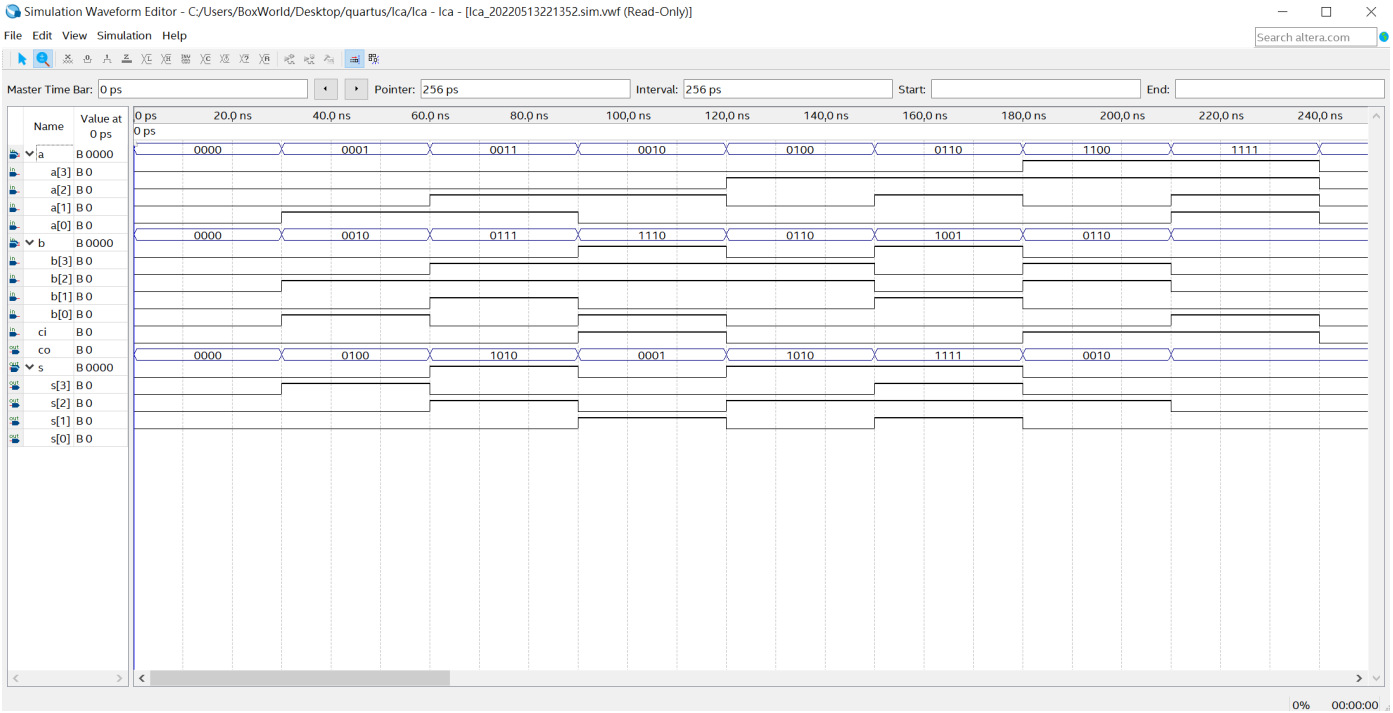

图2　超前进位加法器仿真结果

可以看见，两种加法器的结果均正确。

4 延迟比较

1. 对于逐次进位加法器，其延迟为 $53.06 - 40 = 13.06ns$ ：

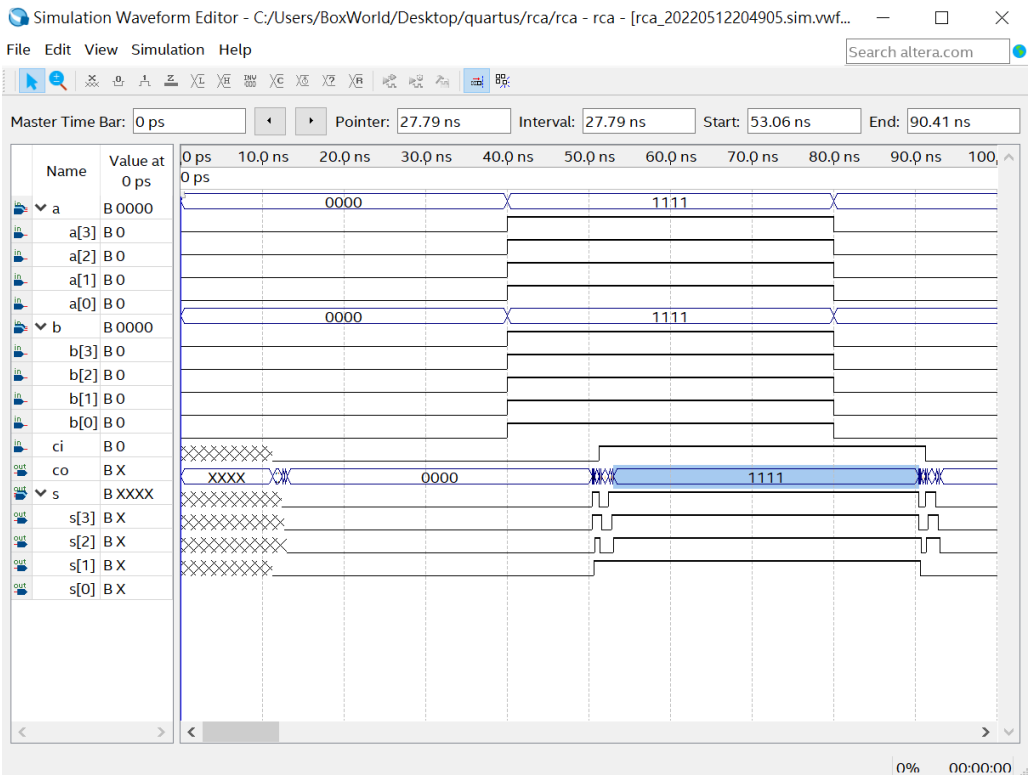

图3　逐次进位加法器仿真结果

2. 对于超前进位加法器，其延迟为 $52.47 - 40 = 12.47ns$ ：

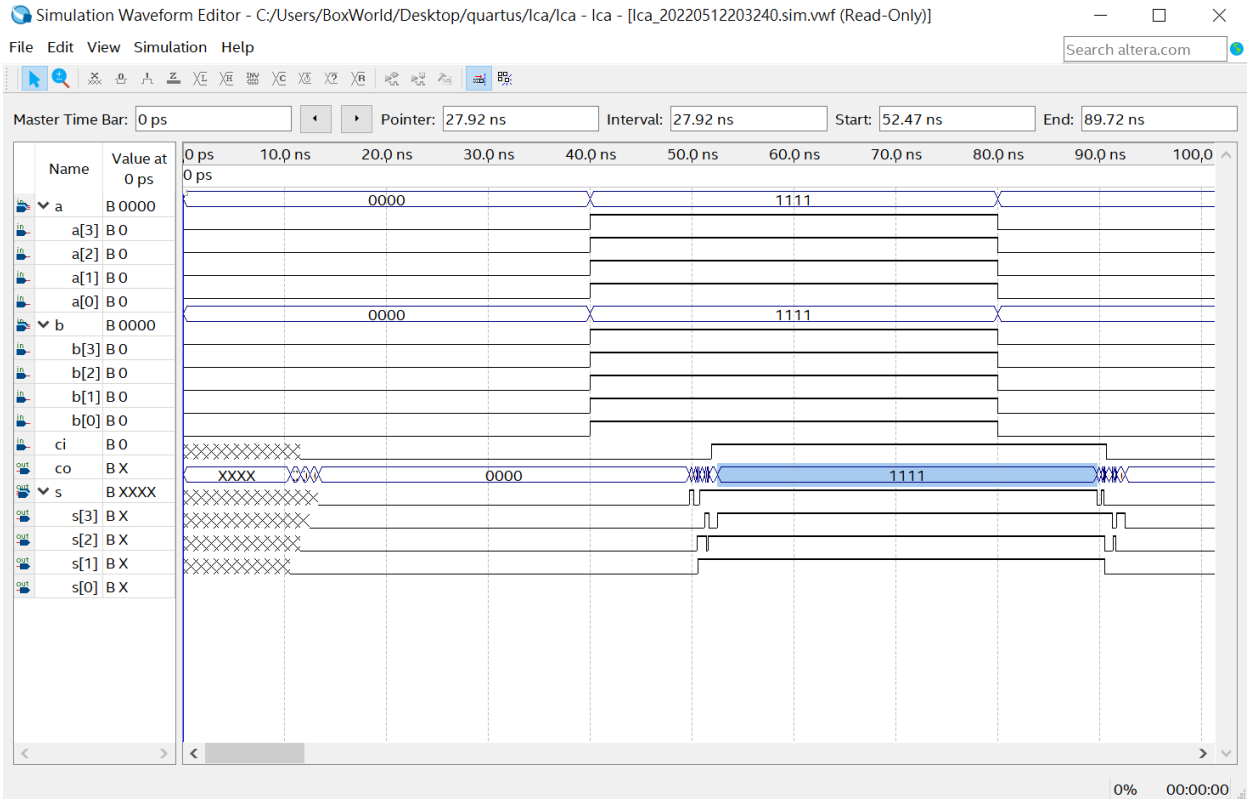

图4　超前进位加法器仿真结果

3. 对于系统自带加法器，其延迟为 $52.28 - 40 = 12.28ns$ ：

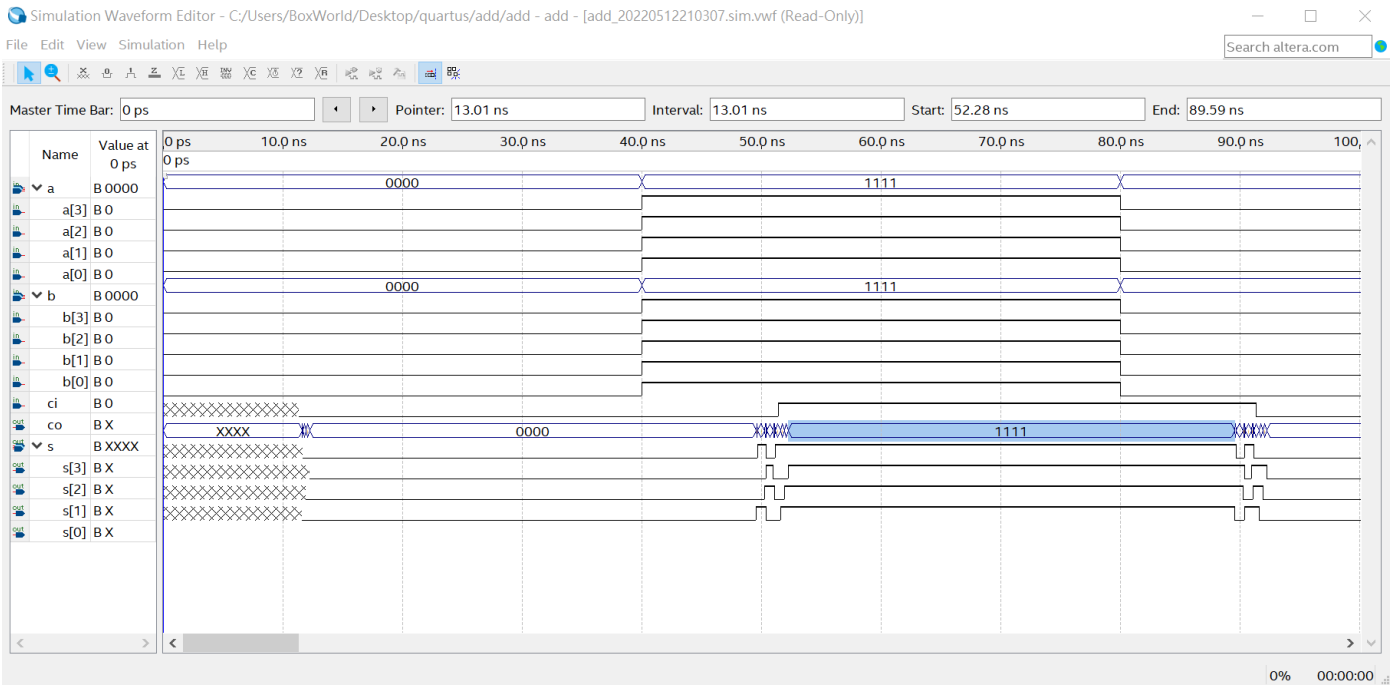

图5　系统自带加法器仿真结果

在本次仿真实验中，测得系统自带加法器的延迟小于超前进位加法器，而超前进位加法器的延迟则小于逐次进位加法器。这说明了超前进位加法器的在运算上的效能确实优于逐次进位加法器，而系统自带的加法器则可能在底层实现了优化，因此可以得到更好的效果。