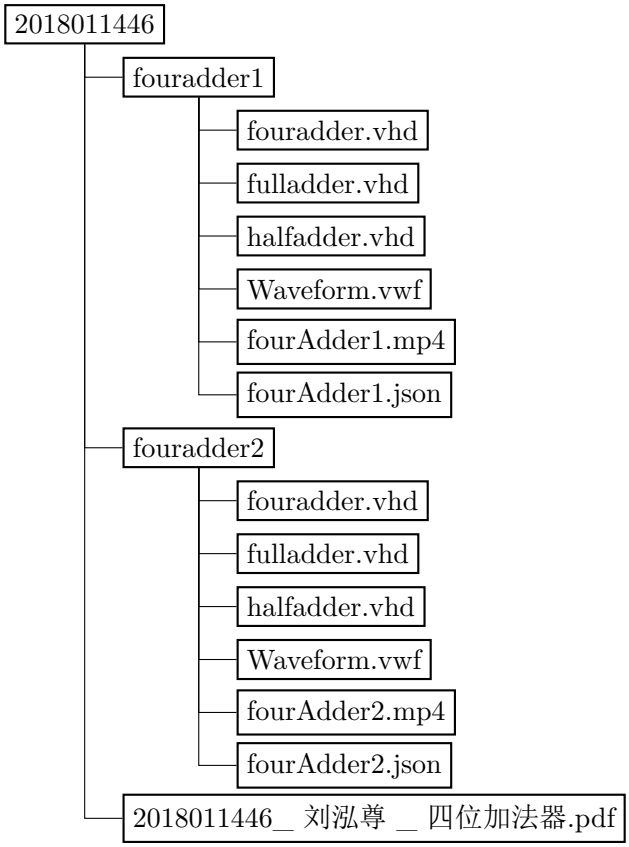
四位加法器：实验报告

刘泓尊　2018011446　计 84

2020 年 4 月 1 日

目录

1 File Structure



2 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.

3 实验任务

(1) 设计实现逐次进位加法器，使用” 半加器 → 全加器 → 逐次进位加法器” 方式，进行“功能仿真”并在实验平台上测试。

(2) 设计实现超前进位加法器，使用” 半加器 → 全加器 → 超前进位加法器” 方式，进行“功能仿真”并在实验平台上测试。

4 逐次进位加法器

本部分代码位于./fouradder1 下

4.1 1 位半加器

半加器的实现十分简单，只需要处理 $S = A \oplus B, C = AB$ 即可，具体实现如下:

halfadder.vhd

```vhdl
--一位半加器--
entity halfadder is
    port(
        a, b: in std_logic;
        so, co: out std_logic
    );
end halfadder;

architecture bhv_half of halfadder is
begin
    co <= a and b;
    so <= a xor b;
end bhv_half;
```

4.2 1 位全加器

采用元件例化的方式，使用 2 个半加器实现全加器。

fulladder.vhd

```vhdl
--一位全加器--
entity fulladder is
    port(
        ain, bin, ci: in std_logic;
        s, co: out std_logic
    );
end fulladder;

architecture bhv_full of fulladder is
    component halfadder
        port(
            a, b: in std_logic;
            so, co: out std_logic
        );
    end component;
    signal s1, c1, c2: std_logic;
begin
    u1: halfadder port map(a=>ain, b=>bin, so=>s1, co=>c1);--元件例化
    u2: halfadder port map(a=>s1, b=>ci, so=>s, co=>c2);
    co <= c1 or c2;
end bhv_full;
```

4.3 4 位逐次进位加法器

基于上面实现的 1 位全加器，可以用串行的策略实现 4 位逐次进位加法器。

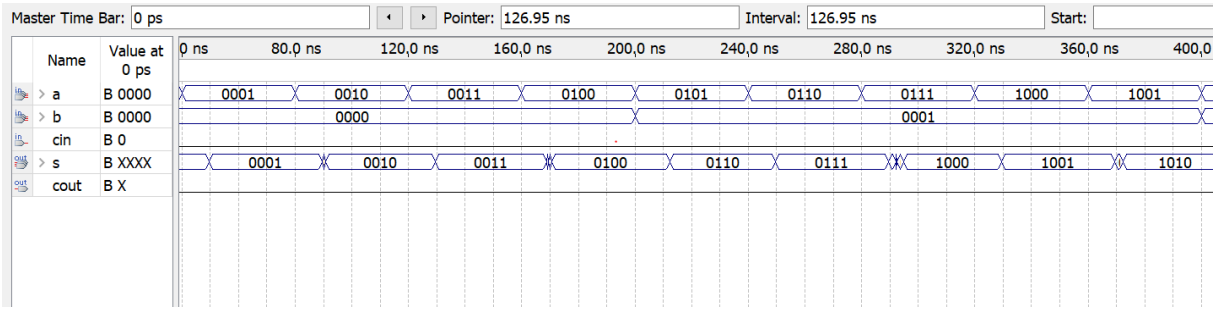
fouradder.vhd

```
--逐次进位加法器，时延9ns左右--
entity fouradder 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 fouradder;

architecture bhv of fouradder is
    component fulladder
        port(
            ain, bin, ci: in std_logic;
            s, co: out std_logic
        );
    end component;
    signal c: std_logic_vector(3 downto 0);
begin
    u1: fulladder port map(ain=>a(0), bin=>b(0), ci=>cin , s=>s(0), co=>c(0)
        );--元件例化
    u2: fulladder port map(ain=>a(1), bin=>b(1), ci=>c(0), s=>s(1), co=>c(1)
        );
    u3: fulladder port map(ain=>a(2), bin=>b(2), ci=>c(1), s=>s(2), co=>c(2)
        );
    u4: fulladder port map(ain=>a(3), bin=>b(3), ci=>c(2), s=>s(3), co=>cout
        );
end bhv;
```

4.4 仿真结果

使用 Quartus 的 ModelSim 进行仿真 (附"**Waveform.vwf**" 文件)，结果如下 从图中可以看到，

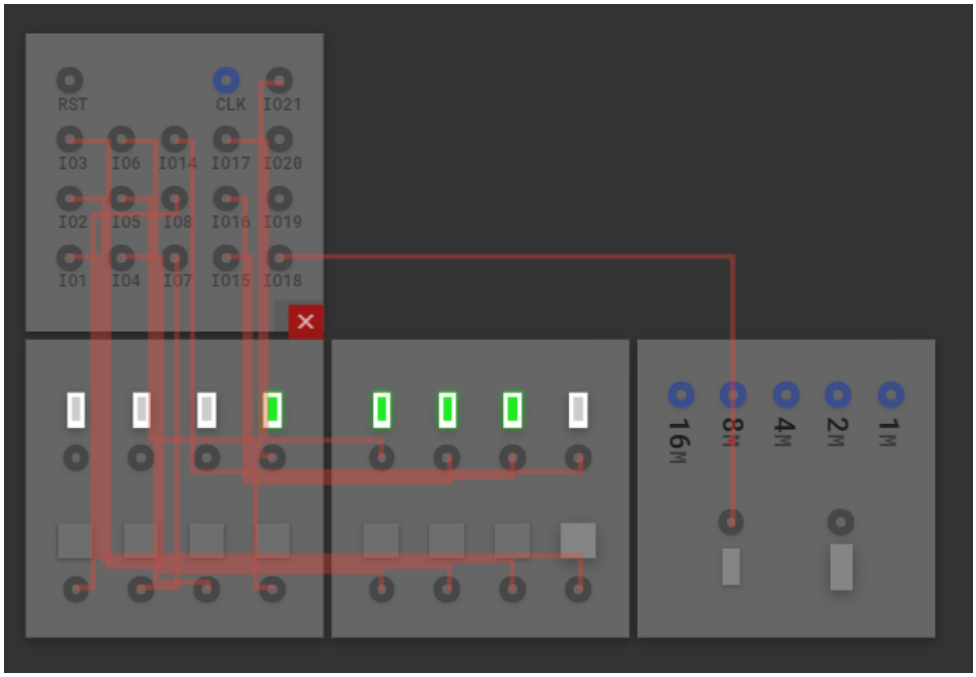


逐次进位加法器的平均时延约在 $10ns$ 左右，但是在有些情况下会达到 $12ns$，电路正确实现了加法功能。

4.5 JieLab 运行结果 (附录屏)

我将代码放在了 JieLab 实验平台上进行了验证，下面是运行时的截图，在 fourAdder1 目录下有"**fourAdder1.mp4**"，您可以更直观地检查我的实现效果. 同时附 JieLabs 导出的 fourAdder1.json 文件，便于您在平台上进行验证。

下面我给出测试过程中的截图，该图计算了 $1111+1110+1=11110$. 示数和开关的数位均为“左高右低”，与手写形式一致，以便于观察。左下两个板子的 4 个开关分别代表一个二进制数，最右侧一个板子的开关 clk 代表 C_{-1}.



5 超前进位加法器

本部分代码位于./fouradder2 下

5.1 1 位半加器

1 位半加器的实现和上面” 逐次进位加法器” 的 1 位半加器相同，在此不再赘述。

5.2 1 位全加器

1 位全加器相对于“逐次进位加法器”的 1 位全加器，主要是增加了“进位产生函数 G_n”和“进位传递函数 P_n”. 使用 $buffer$ 实现。全加器的实现同样使用了半加器的“例化”。

fulladder.vhd

```
1  --一位全加器--
2  entity fulladder is
3      port(
4          a, b, ci: in std_logic;
5          s: out std_logic;
6          p, g: buffer std_logic
```

```
    );
end fulladder;

architecture bhv_full of fulladder is
    component halfadder
        port(
            ain, bin: in std_logic;
            so, co: out std_logic
        );
    end component;
begin
    u1: halfadder port map(ain=>a, bin=>b, so=>p, co=>g);--半加器例化
    s <= ci xor p;
end bhv_full;
```

5.3 4 位超前进位加法器

4 位超前进位加法器的关键是“超前进位单元”，超前进位的输出表达式如下:

$$
\begin{aligned}
C_0 &= G_0 + P_0 C_{-1} \\
C_1 &= G_1 + P_1 G_0 + P_1 P_0 C_{-1} \\
C_2 &= G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_{-1} \\
C_3 &= G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 C_{-1}
\end{aligned}
$$

基于上述逻辑，可以使用元件例化的方式得到 P_i, G_i, 之后处理超前进位部分。代码如下:

fouradder.vhd

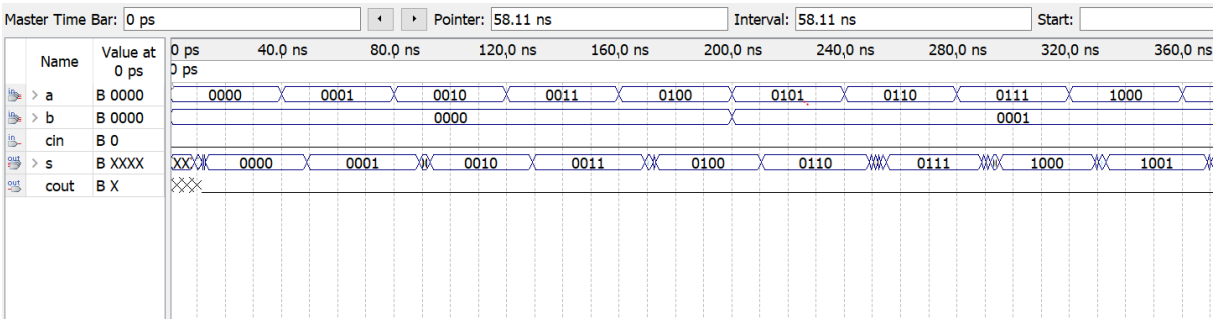
```
--Look Ahead Carry four-bit Adder, Time delay: 9ns--
entity fouradder is
    port(
        a, b: in std_logic_vector(3 downto 0);
        cin: in std_logic;
        s: out std_logic_vector(3 downto 0);
        cout: out std_logic
    );
end fouradder;

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

```
19
20  begin
21      u0: fulladder port map(a=>a(0), b=>b(0), ci=>cin, p=>p(0), g=>g(0), s=>s
            (0));
22      u1: for i in 1 to 3 generate--使用生成语句，元件例化
23          ux: fulladder port map(a=>a(i), b=>b(i), ci=>c(i-1), p=>p(i), g=>g(i
                ), s=>s(i));
24      end generate;
25      --超前进位部分--
26      c(0) <= g(0) or (p(0) and cin);
27      c(1) <= g(1) or (p(1) and g(0)) or (p(1) and p(0) and cin);
28      c(2) <= g(2) or (p(2) and g(1)) or (p(2) and p(1) and g(0))
29                   or (p(2) and p(1) and p(0) and cin);
30      c(3) <= g(3) or (p(3) and g(2)) or (p(3) and p(2) and g(1))
31                   or (p(3) and p(2) and p(1) and g(0))
32                   or (p(3) and p(2) and p(1) and p(0) and cin);
33      cout <= c(3);
34  end bhv;
```

5.4 仿真结果

使用 Quartus 的 ModelSim 进行仿真 (附"**Waveform.vwf**" 文件)，结果如下

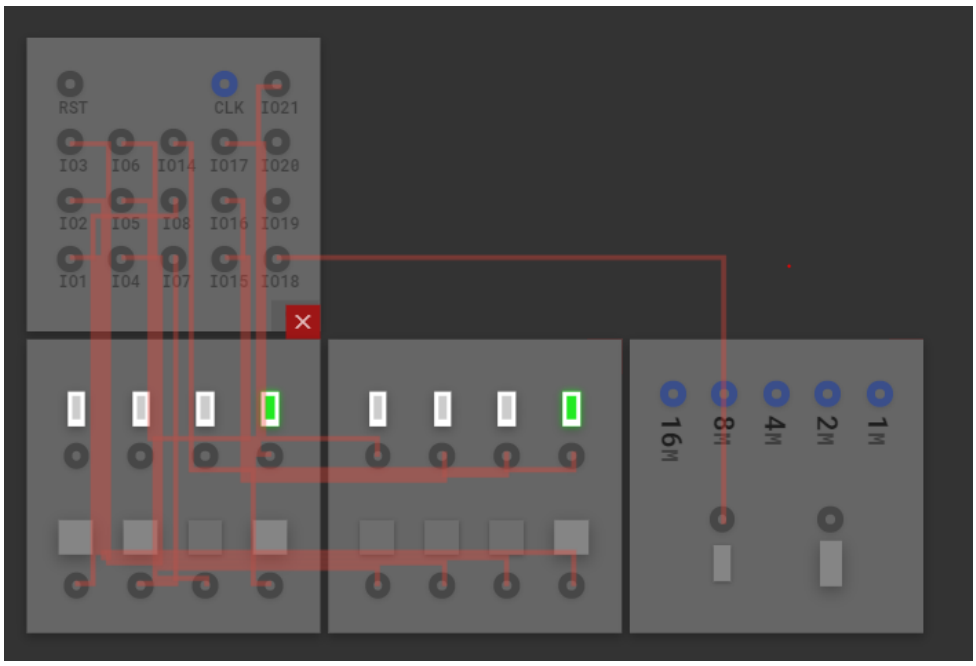


从图中可以看到，超前进位加法器的平均时延约在 $9ns$ 左右，速度只比逐次进位快一点，但是在更新的时候会有短暂的“毛刺”，电路正确实现了加法功能。初步猜测超前进位没有明显加速的原因是超前进位单元有大量的逻辑运算，会拖慢速度。

5.5 JieLab 运行结果 (附录屏)

我将代码放在了 JieLab 实验平台上进行了验证，下面是运行时的截图，在 fourAdder2 目录下有"**fourAdder2.mp4**",您可以更直观地检查我的实现效果. 同时附 JieLabs 导出的 fourAdder2.json 文件，便于您在平台上进行验证。

下面我给出测试过程中的截图，该图计算了 $0010+1110+1=10001$. 示数和开关的数位均为“左高右低”，与手写形式一致，以便于观察。左下两个板子的 4 个开关分别代表一个二进制数，最右侧一个板子的开关 clk 代表 C_{-1}.



6 实验总结

在两个任务完成后，我又测试了 VHDL 自带加法器的延时，得到的时延依然为 $9ns$ 左右，但是毛刺比较少，延时更加均衡。但我没有找到相关资料讲解 VHDL 自带的加法器，希望有时间深入研究一下。

这是我第二次进行 CPLD 的实验，有了上一次的练习，这次的实验十分顺利，VHDL 代码也是一次成形，没有遇到 bug。我学到了“元件例化”的方法，极大地提高了硬件设计的简洁性，体会到了模块化设计在电路设计中的重要性。此次实验让我对全加器、超前进位机制等有了实践性的任务，切身体会到了硬件的延时，再次领会了”理论 $\neq$ 实际”。感谢老师助教在微信群里的耐心指导!