

32 位超前进位加法器

超前进位加法器原理

对于一般的行波进位加法器中的每一个组件（全加器），我们有：

$$\begin{cases} S_i = A \oplus B \oplus C_i \\ C_{i+1} = A_i \cdot B_i + A_i \cdot C_i + B_i \cdot C_i \end{cases}$$

令 $G_i = A_i \cdot B_i$, $P_i = A_i + B_i$, 则 $C_{i+1} = G_i + P_i \cdot C_i$ 。

可见，进位中的 G_i 和 P_i 不依赖前面的数据，仅依赖本位数据，于是，我们可以将行波进位加法器改为：

$$\left\{ \begin{array}{l} C_1 = G_0 + P_0 \cdot C_{in} \\ C_2 = G_1 + P_1 \cdot C_1 \\ \quad = G_1 + P_1 \cdot G_0 + P_0 \cdot P_1 \cdot C_{in} \\ C_3 = G_2 + P_2 \cdot C_2 \\ \quad = G_2 + P_2 \cdot G_1 + P_1 \cdot P_2 \cdot G_0 + P_0 \cdot P_1 \cdot P_2 \cdot C_{in} \\ C_4 = G_3 + P_3 \cdot C_3 \\ \quad = G_3 + P_3 \cdot G_2 + P_2 \cdot P_3 \cdot G_1 + P_1 \cdot P_2 \cdot P_3 \cdot G_0 + P_0 \cdot P_1 \cdot P_2 \cdot P_3 \cdot C_{in} \\ C_{out} = C_4 \end{array} \right.$$

可见，这个四位行波进位加法器其实不需要一个一个计算，每一个进位实际上都可以直接通过输入数据进行计算。

又由于 $S_i = A \oplus B \oplus C_i$, 我们可以构建一个四位超前进位加法器，见 ADD_4.v。

扩展化-32 位超前进位加法器

由上面的分析，可见其表达式是有规律的，我们可以尝试直接构建 32 位超前进位加法器，见 ADD_32.v。

其中，python 脚本 generate.py 是用于简化重复书写的代码生成器。

模块化

可以发现，计算本位结果的计算式和计算各位 G_i 和 P_i 式子固定，就是 $S_i = A \oplus B \oplus C_i$ 和 $G_i = A_i \cdot B_i$, $P_i = A_i + B_i$, 这部分可以按位进行模块化；但是计算进位的计算式会随着位的改变而改变（增加），这部分就只能按照加法器的规模进行模块化了。

于是，一个超前进位加法器可以由 n 个全加器（带 G_i 和 P_i 输出），和一个进位发生器组成。

例如，一个 4 位超前进位加法器：

全加器定义：

```

module add_1bit(
    input wire cin,
    input wire a,
    input wire b,
    output wire S,
    output wire G,
    output wire P
);
    xor u_xor(S,a,b,cin);
    and u_and(G,a,b);
    or u_or(P,a,b);
endmodule

```

进位生成器 IO 定义：

```

module CLA_4bit(
    input wire [3:0] G,
    input wire [3:0] P,
    input wire cin,
    output wire [2:0] CI,
    output wire cout
);

```

这样就可以组一个 4 位超前进位加法器了，顶级模块 IO 定义如下：

```

module ADD_4bit(
    input wire [3:0] a,
    input wire [3:0] b,
    input wire cin,
    output wire [3:0] res,
    output wire cout
);

```

完整代码可见 ADD_t.v。

其中，值得注意的是作为全加器的输入的进位线组 C 最初是没有初始化的，而要依赖其输出的 cla 组件进行 CI 的计算，这看似循环依赖，但是仔细观察， cla 组件需要的 G 和 P 数据不依赖进位数据，而只有结果是依赖进位的，所以并没有循环依赖。

层次化扩展

超前进位加法器可以模块化构建，但是绕不开的一个问题在于进位生成器没办法模块化，这意味着构建一个 32 位超前进位加法器是非常复杂的。实际上，更常用的做法是使用层次化构建：即把一个小的加法器当作

一个整体，通过一个比较简单的进位生成器就可以完成一个大型的加法器。这有点像行波进位加法器，但原理不同。

比如，使用上面构建的 ADD_4bit 和 CLA_4bit 进行如同一个模块化四位加法器一样的连接，就可以得到一个 16 位超前进位加法器。