实验三：加法器及快速进位电路的设计

实验时间： 2017年11月13日 第十周 星期一

实验者：16307130194 陈中钰 16级 计算机科学技术学院

座位号：30

指导老师：唐志强

1 实验目的

- 了解并掌握半加器、全加器的设计和功能
- 了解并设计行波进位加法器、超前进位加法器
- 对比上述两者的延时，总体了解并对比两者的优缺点
- 加深对分层设计思想的了解，更加熟悉模块化的设计，理解模块化的优点

2 实验原理

2.1 1位全加器

- 输入：a，b，cin；输出：s，cout
- 真值表

输入			输出	
a	b	cin	s	cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

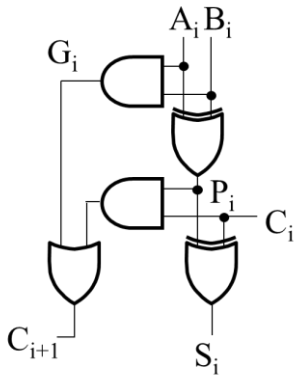
- K-Map（以 s 为例）

cin \ ab	00	01	11	10
0		1		1
1	1		1	

可得 s=a’b’cin+a’bcin’+ab’cin’+abcin=a^b^cin

同理可得 cout=ab+acin+bcin

- 逻辑电路图（右图）。其中，Ai，Bi，Si 对应 a，b，s 而 Ci，Ci+1对应 cin，cout

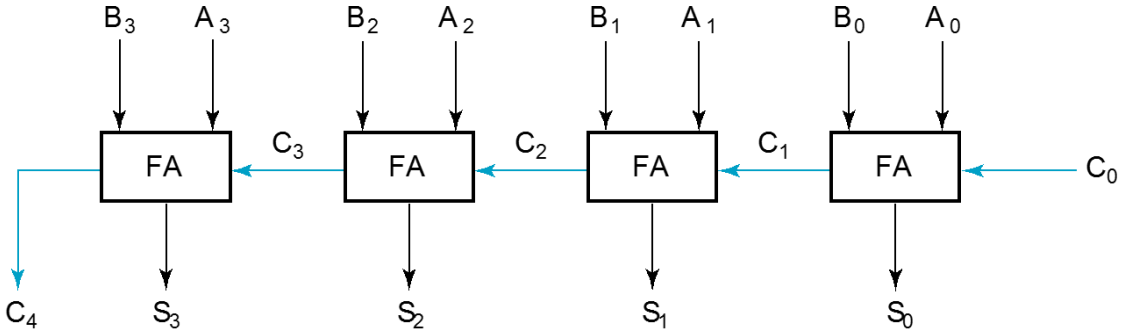


2.2 以上述1位全加器为模块，设计4位行波进位加法器

- 输入 A[3:0]，B[3:0]，C[0]；输出 S[3:0]，C[4]
- 模拟实际的加法运算过程，从低位开始，用1位全加器求和，并进位到高位，再用1位全加器计算高位的和，直至计算完毕。那么可以使用4个全加器模块，串行设计，低位的进位输出为高位的进位输入，每一个全加器都只需要计算两个加数在这一位的数字 A[i]、B[i]以及进位输入 C[i]的三者求和结果 S[i]，还有该位的进位输出 C[i+1]，

那么，先从低位开始算，并一位接一位地算下去，直至计算完毕

- 逻辑电路图



2.3 4位前进位加法器

- 输入 A[3:0]，B[3:0]，C[0]；输出 S[3:0]，C[4]
- 由于进位需要一位一位地从低位往高位传送，使得每一位的运算都是按从低位到高位的顺序进行的，耗时长，那么需要找到一种方法使得每一位都能同时计算，而要实现每一位的求和同时进行，则需要把每一位求和所需要的进位输入事先、并行计算出来，再同时计算出每一位的求和结果 s[i]
- 并行计算进位原理

先观察进位计算的规律

A	B	cin	cout
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

那么，可以发现，AB=00时一定没有进位，AB=11时一定有进位，AB=01或10时，若同时有 cin 为1，则有进位，那么可以有 cout=AB+(A^B)cin，那么令

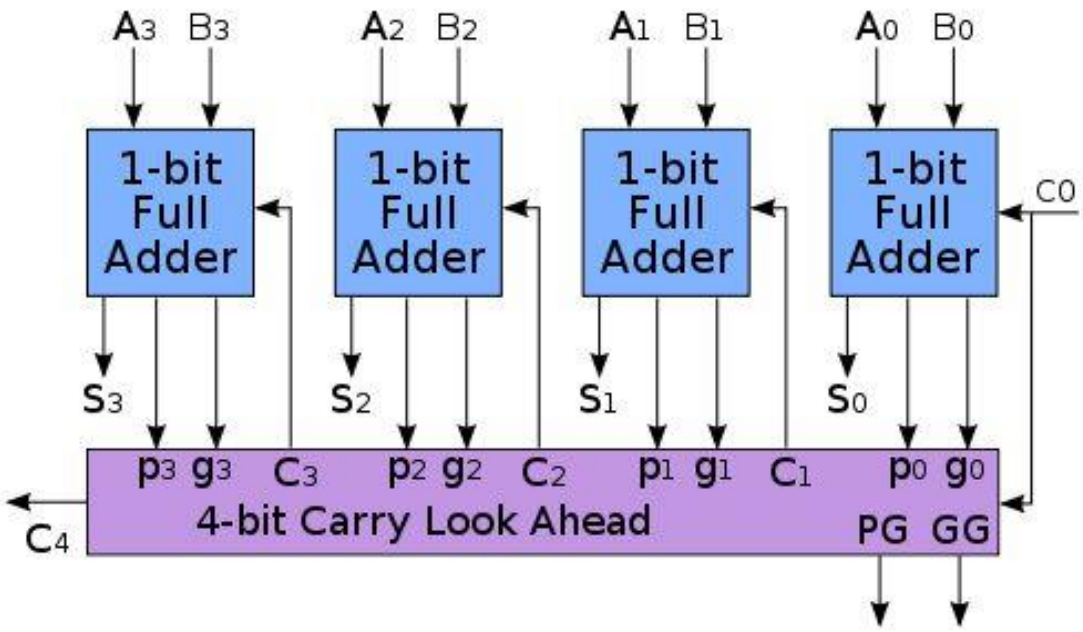
g[i]=A[i]B[i]	0<=i<=3
p[i]=A[i]^B[i]	
c[i+1]=g[i]+p[i]c[i]	
s[i]=A[i]^B[i]^c[i]=p[i]^c[i]	

那么进一步有

c[1]=g[0]+p[0]c[0]
c[2]=g[1]+p[1]c[1]=g[1]+p[i](g[0]+p[0]c[0])
c[3]=g[2]+p[2]c[2]= g[2]+p[2]g[1]+p[i](g[0]+p[0]c[0]))
c[4]=g[3]+p[3]c[3]= g[3]+p[3](g[2]+p[2]g[1]+p[i](g[0]+p[0]c[0])))

首先先计算出 g[3:0]和 p[3:0]的值，再据此计算出 c[4:1]（其中 c[0]为 cin，c[4]为 cout），那么每一位的求和运算都能直接取进位输入，能并行完成计算。

- 逻辑电路图



3 实验内容

3.1 1位全加器

- Verilog 代码

```
module full_adder_1(input a, input b, input cin, output s, output
cout);
    assign s=~a&~b&cin | ~a&b&~cin | a&~b&~cin | a&b&cin;
    assign cout=a&b | a&cin | b&cin;
endmodule
```

3.2 以上述1位全加器为模块，设计4位行波进位加法器

- Verilog 代码

```
module ripple_carry_adder_4(input [3:0] a, input [3:0] b, input cin,
output [3:0] s, output cout);
    wire [4:0] c;
    assign c[0]=cin;
    full_adder_1 bit0(a[0+:1],b[0+:1],c[0+:1],s[0+:1],c[1+:1]);
    full_adder_1 bit1(a[1+:1],b[1+:1],c[1+:1],s[1+:1],c[2+:1]);
    full_adder_1 bit2(a[2+:1],b[2+:1],c[2+:1],s[2+:1],c[3+:1]);
    full_adder_1 bit3(a[3+:1],b[3+:1],c[3+:1],s[3+:1],c[4+:1]);
    assign cout=c[4];
endmodule
```

3.3 4位超前进位加法器（其中 s[3:0]的计算按要求使用模块化来实现）

```
module carry_look_ahead_adder_4(input [3:0] a, input [3:0] b, input
cin, output [3:0] s, output cout);
    wire [3:0] g; wire [3:0] p; wire [4:0] c;
    assign g[3:0]=a[3:0]&b[3:0];
    assign p[3:0]=a[3:0]^b[3:0];
```

```
    assign c[0]=cin;
    assign c[1]=g[0]|p[0]&c[0];
    assign c[2]=g[1]|p[1]&g[0]|p[1]&p[0]&c[0];
    assign c[3]=g[2]|p[2]&g[1]|p[2]&p[1]&g[0]|p[2]&p[1]&p[0]&c[0];
    assign c[4]=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[0];
    assign cout=c[4];
    xor_2 x(p[3:0], c[3:0], s[3:0]);
endmodule

module xor_2(input [3:0] p, input [3:0] c, output [3:0] s);
    assign ans[3:0]=p[3:0]^c[3:0];
endmodule
```

4 实验结论

- 实现了1位全加器
- 以1位全加器为底层模块，设计了4位行波进位加法器、4位超前进位加法器
- 行波进位加法器需要从低位到高位一步步串行计算，而超前进位加法器则是并行计算，后者更快

5 实验感想

- 虽然超前进位加法器计算更快，但是额外使用了更多的逻辑门和线，成本会更高，而且当超前进位位数更多时，逻辑门个数呈指数增长，成本也随之快速增加，因此超前进位加法器的位数不能过多，一般是4位宽的。而虽然行波进位加法器计算较慢，但是成本较低。如果要实现更多位的加法，则需要超前进位加法器和行波进位加法器搭配使用（如下图中的16位加法器，每4位使用1个4位宽的超前进位加法器进行计算，而这4个超前进位加法器之间，用一个4位宽的行波进位加法器进行串行连接），既能提高速度，成本也不会增加太多。其中 CLAA 为超前进位加法器，A[i]、B[i]、S[i]均为宽度为4的向量，C[i]宽度为1