浙江大学

本科实验报告

快速加法器的设计

课程名称：	计算机组成与设计
姓　　名：	姚桂涛
学　　院：	信息与电子工程学院
专　　业：	信息工程
学　　号：	**3190105597**
指导老师：	屈民军、唐奕

2021 年 11 月 25 日

浙江大学实验报告

专业：信息工程
姓名：姚桂涛
学号：3190105597
日期：2021 年 11 月 25 日
地点：教 11-400

课程名称：计算机组成与设计　指导老师：屈民军、唐奕　成绩：
实验名称：快速加法器的设计　实验类型：设计验证　同组学生姓名：

一、 实验目的

(1) 掌握快速加法器的设计方法。

(2) 熟悉流水线技术。

(3) 掌握时序仿真的工作流程。

二、 实验任务

(1) 采用“进位选择加法”技术设计 32 位加法器，并对设计进行功能仿真和时序仿真。

(2) 采用四级流水线技术设计 32 位加法器，并对设计进行功能仿真和时序仿真。

三、 实验原理

1. 四位先行进位加法器的设计

个加数分别为 $A_3A_2A_1A_0$ 和 $B_3B_2B_1B_0$,C_{-1} 为最低位进位。设两个辅助变量分别为 $G_3G_2G_1G_0$ 和 $P_3P_2P_1P_0$，$Gi = AiBi$，$Pi = Ai + Bi$。

一个四位加法器的进位计算就变转化为

$$\begin{cases} C_0 = G_0 + P_0C_{-1} \\ C_1 = G_1 + P_1C_0 = G_1 + P_1G_0 + P_1P_0C_{-1} \\ C_2 = G_2 + P_2C_1 = G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_{-1} \\ C_3 = G_3 + P_3C_2 = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0 + P_3P_2P_1P_0C_{-1} \end{cases}$$

由上式可以看出，每一个进位的计算都直接依赖于整个加法器的最初输入，而不需要等待相邻低位的进位传递。理论上，每一个进位的计算都只需要三个门延迟时间，即产生 $G[i]$、$P[i]$ 的与门和或门，输入为 $G[i]$、$P[i]$、C_{-1} 的与门，以及最终的或门。同样道理，理论上最终结果 sum 的得到只需要四个门延迟时间。

实际上，当加数位数较大时，输入需要驱动的门数较多，其 VLSI 实现的输出时延增加很多，考虑到互连线的延时情况将会更加糟糕。因此，通常在芯片实现时先设计位数较少的超前进位加法器结构，而后以此为基本结构来构造位数较大的加法器。

2. 进位选择加法器结构

实际上，由超前进位加法器级联构成的多位加法器只是提高了进位传递的速度，其计算过程与行波进位加法器同样需要等待进位传递的完成。

借鉴并行计算的思想，人们提出了进位选择加法器结构，或者称为有条件的加法器结构（conditional sum adder），其算法的实质是增加硬件面积换取速度性能的提高。二进制加法的特点是进位或者为逻辑 1，或者为逻辑 0，二者必居其一。将进位链较长的加法器分为 M 块分别进行加法计算，对除去最低位计算块外的 M-1 块加法结构复制两份，其进位输入分别预定为逻辑 1 和逻辑 0。于是，M 块加法器可以同时并行进行各自的加法运算，然后根据各自相邻低位加法运算结果产生的进位输出，选择正确的加法结果输出。图 5.6 所示为 12 位进位选择加法器的逻辑结构图。12 位加法器划分为 3 块，最低一块（4 位）由 4 位超前进位加法器直接构成，后两块分别假设前一块的进位为 0 或 1 将两种结果都计算出来，再根据前级进位选择正确的和与进位。如果每一块加法结构内部都采用速度较快的超前进位加法器结构，那么进位选择加法器的计算时延为

$$t_{\mathrm{CSA}} = t_{\mathrm{carry}} + (M-2)t_{\mathrm{MUX}} + t_{\mathrm{sum}}$$

其中，t_{sum}、t_{carry} 分别为加法器的和与加法器的进位时延，t_{MUX} 为数据选择球器的时延。

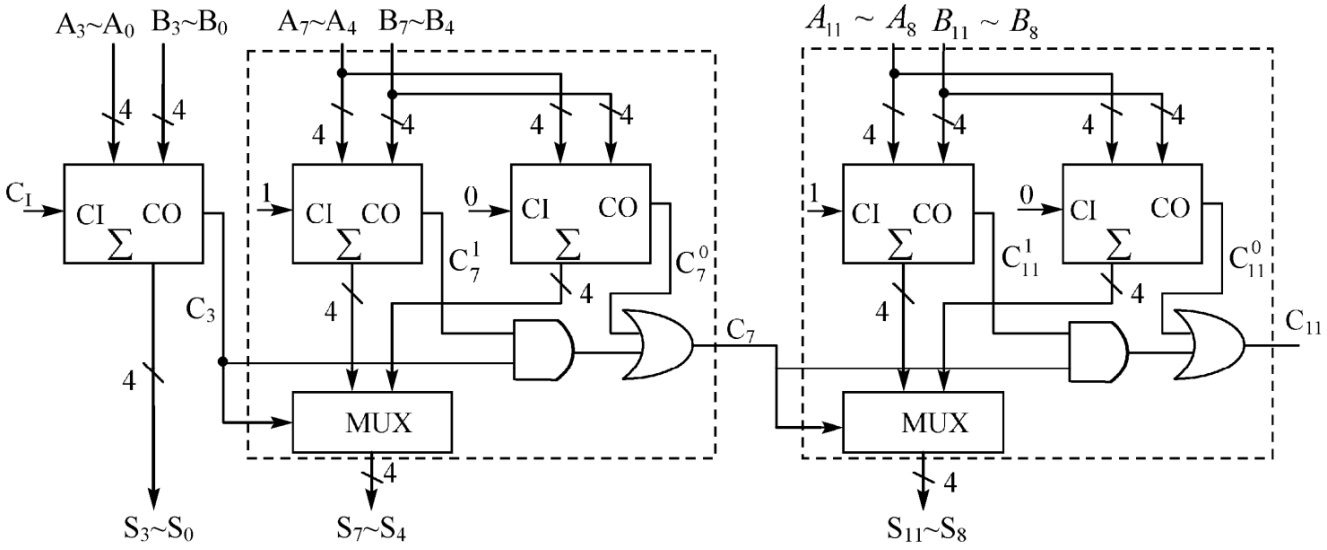

图 1: 12 进位选择器加法器原理图

四、 实验设备

装有 Vivado、ModelSim SE 软件的计算机。

五、 实验内容

1. 实验设计

32 位进位选择加法器的结构参考了讲义中 12 位进位选择加法器的设计，将加法器分成了 8 块，其中除去最低为的由 4 位超前进位加法器构成的计算块外，其余 7 块中间块设计如下图

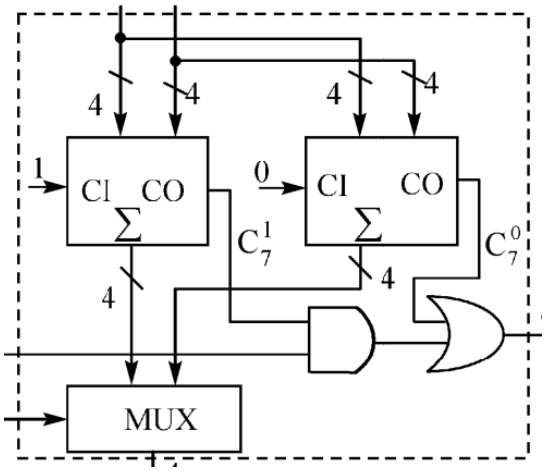

图 2: 中间块设计原理图

2. 实验代码

4 位加法器代码

```
module adder_4bits (
    a,
    b,
    ci,
    s,
    co
);
    input [3:0] a;
    input [3:0] b;
    input ci;     //最低进位
    output [3:0] s; //求和结果
    output co;    //最高进位

    wire [3:0] g,p;

    assign g = a & b;
    assign p = a | b;

    wire [3:0] tC;
    assign tC[0] = g[0] | p[0] & ci;
    assign tC[1] = g[1] | p[1] & g[0] | p[1] & p[0] & ci;
    assign tC[2] = g[2] | p[2] & g[1] | p[2] & p[1] & g[0] | p[2] & p[1] & p[0] & ci;
    assign tC[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] & ci;
    assign co = tC[3];
    wire [3:0] c;
    assign c = {tC[2:0],ci};
    assign s = p & (~g) ^ c;

```

```
29
30 endmodule
```

32 位进位选择加法器代码

```
1  module adder_32bits (
2      a,
3      b,
4      ci,
5      s,
6      co
7  );
8      input [31:0] a;
9      input [31:0] b;
10     input ci;
11     output [31:0] s;
12     output co;
13     wire [6:0] c;
14
15     // 第一块4位超前进位
16     adder_4bits adder(
17         .a(a[3:0]),
18         .b(b[3:0]),
19         .ci(ci),
20         .s(s[3:0]),
21         .co(c[0])
22     );
23     // 其余7块
24     csa csa1(
25         .a(a[7:4]),
26         .b(b[7:4]),
27         .ci(c[0]),
28         .s(s[7:4]),
29         .co(c[1])
30     );
31
32     csa csa2(
33         .a(a[11:8]),
34         .b(b[11:8]),
35         .ci(c[1]),
36         .s(s[11:8]),
37         .co(c[2])
38     );
39
40     csa csa3(
41         .a(a[15:12]),
42         .b(b[15:12]),
43         .ci(c[2]),
44         .s(s[15:12]),
45         .co(c[3])
46     );
```

```
47
48      csa csa4(
49          .a(a[19:16]),
50          .b(b[19:16]),
51          .ci(c[3]),
52          .s(s[19:16]),
53          .co(c[4])
54      );
55
56      csa csa5(
57          .a(a[23:20]),
58          .b(b[23:20]),
59          .ci(c[4]),
60          .s(s[23:20]),
61          .co(c[5])
62      );
63
64      csa csa6(
65          .a(a[27:24]),
66          .b(b[27:24]),
67          .ci(c[5]),
68          .s(s[27:24]),
69          .co(c[6])
70      );
71
72      csa csa7(
73          .a(a[31:28]),
74          .b(b[31:28]),
75          .ci(c[6]),
76          .s(s[31:28]),
77          .co(co)
78      );
79
80  endmodule
```

中间块设计代码

```
1   module csa (
2       a,
3       b,
4       ci,
5       s,
6       co
7   );
8       input [3:0] a;
9       input [3:0] b;
10      input ci;
11      output reg [3:0] s;
12      output co;
13      wire [1:0] c;
14      wire [3:0] s0;
```

```
15      wire [3:0] s1;
16
17      adder_4bits adder_1(
18          .a(a[3:0]),
19          .b(b[3:0]),
20          .ci(1),
21          .s(s1),
22          .co(c[1])
23      );
24
25      adder_4bits adder_0(
26          .a(a[3:0]),
27          .b(b[3:0]),
28          .ci(0),
29          .s(s0),
30          .co(c[0])
31      );
32      // 进位
33      assign co = c[0] | (ci & c[1]);
34      // MUX
35      always @(*) begin
36          if (ci) begin
37              s = s1;
38          end
39          else begin
40              s = s0;
41          end
42      end
43
44  endmodule
```

六、 仿真分析

1. 4 位加法器仿真分析

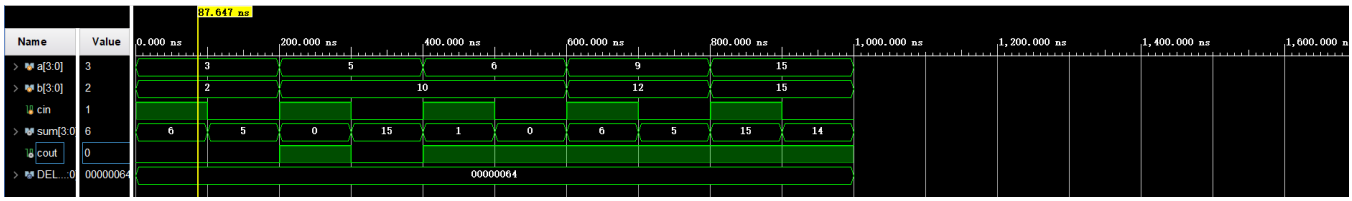

图 3: 4 位加法器仿真分析整体图

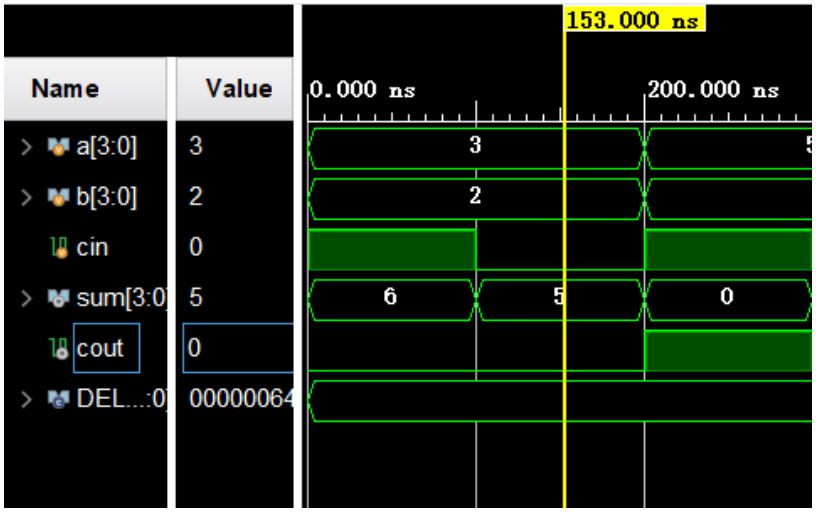

图 4: 4 位加法器仿真分析局部图

如图所示：当加数 a = 3, b = 2, 进位 ci = 0 时，和 sum = 5, 进位输出 cout = 0。符合设计预期。同时，其余测试数据经检验也符合设计预期，此不在一一赘述。

2. 32 位进位选择器加法器仿真分析

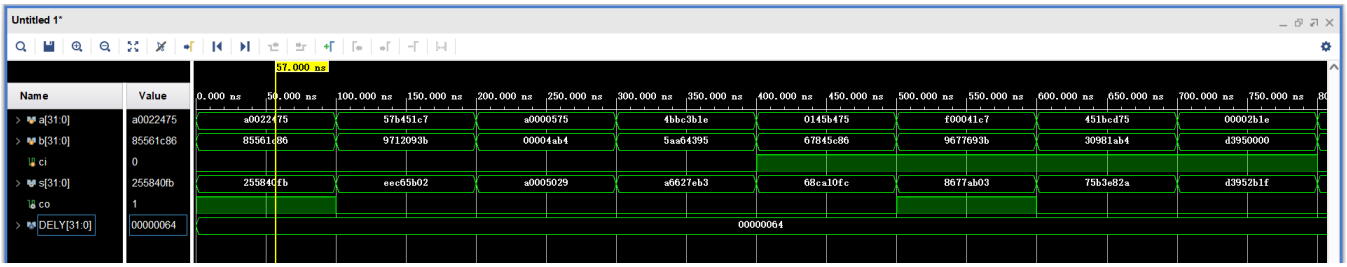

图 5: 32 位加法器仿真分析整体图

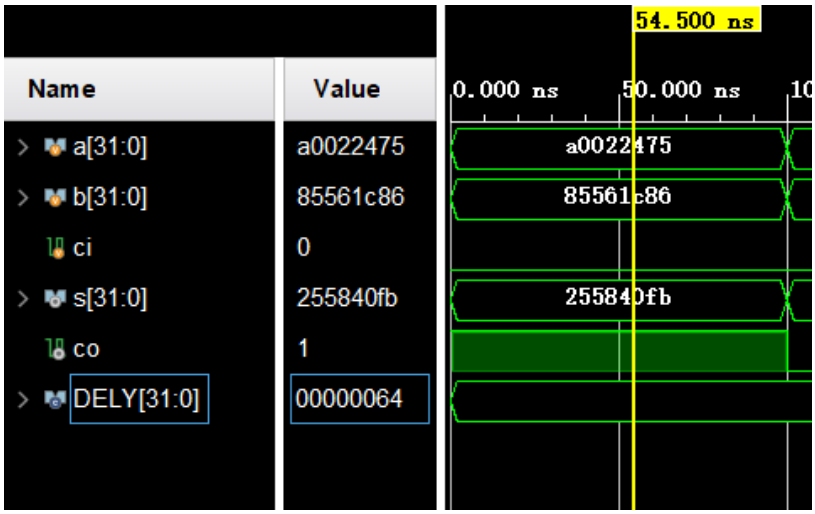

图 6: 32 位加法器仿真分析局部图 1

如图所示：当加数 a = 0xa0022475, b = 0x85561c86, 进位 ci = 0 时，和 sum = 0x255740fb, 进位输出 cout = 1。符合设计预期。

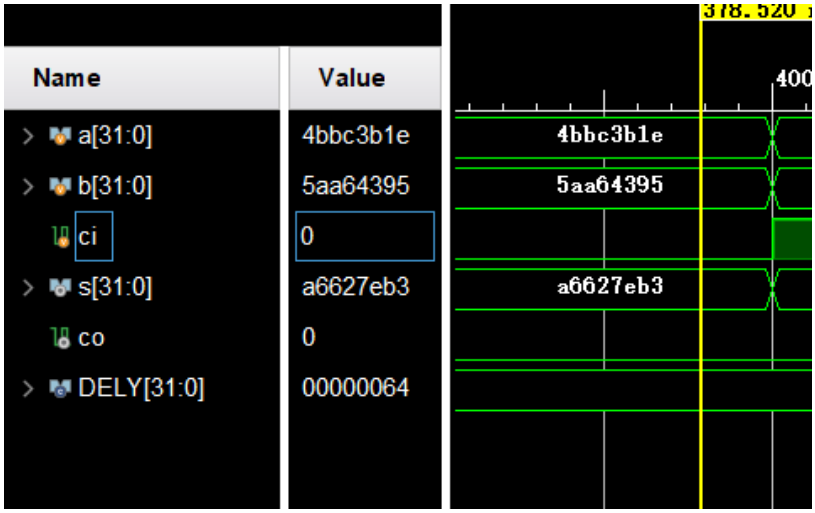

图 7: 32 位加法器仿真分析局部图 2

如图所示：当加数 a = 0x4bbc3b1e, b = 0x5aa64395, 进位 ci = 0 时，和 sum = 0xa6627eb3, 进位输出 cout = 0。符合设计预期。同时，其余测试数据经检验也符合设计预期，此不在一一赘述。

七、 思考题

(1) 为什么要进行时序仿真？

仿真过程是正确实现设计的关键环节，用来验证设计者的设计思想是否正确，及在设计实现过程中各种分布参数引入后，其设计的功能是否依然正确无误。而时序仿真使用布局布线后器件给出的模块和连线的延时信息，在最坏的情况下对电路的行为作出实际地估价。可以比较真实地反映出器件的延时情况

(2) 采用流水线技术有什么优缺点？

优点：流水线技术是将操作执行工作量分成若干个时间上均衡的操作段，从流水线的起点连续地输入，流水线的各操作段以重叠方式执行。这使得操作执行速度只与流水线输入的速度有关，而与处理所需的时间无关。这样，在理想的流水操作状态下，其运行效率很高。缺点：流水线会增大资源的使用。