浙江大学

本科实验报告

32bit 进位选择加法器

课程名称：	计算机组成与设计
姓名：	
学院：	信息与电子工程学院
专业：	电子科学与技术
学号：	
指导老师：	唐奕

January 22, 2024

浙江大学实验报告

专业：电子科学与技术
姓名：
学号：
日期：January 22, 2024
地点：

课程名称：计算机组成与设计　指导老师：唐奕　成绩：
实验名称：32bit 进位选择加法器　实验类型：verilog 硬件设计　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

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。

三、 进位选择加法器设计原理

1. 四位先行进位加法器的设计

假设两个加数为：$A = A_3A_2A_1A_0$,$B = B_3B_2B_1B_0$。C_{-1} 为最低位进位，设置两个辅助变量为：$G = G_3G_2G_1G_0$,$P = P_3P_2P_1P_0$, 则：$G_i = A_i\&B_i$,$P_i = A_i + B_i$。最终，一位全加器的逻辑可以表示为：

$$S_i = P_i\overline{G_i}\oplus C_{i-1}$$

$$C_i = G_i + P_iC_{i-1}$$

利用上述关系，一个四位加法器的进位计算就转变为：

$$C_0 = G_0 + P_0C_{-1} \tag{1}$$

$$C_1 = G_1 + P_1C_0 = G_1 + P_0G_0 + P_1P_0C_{-1} \tag{2}$$

$$C_2 = G_2 + P_2C_1 = G_2 + P_2G_1 + P_2P_1G_0 + P_2P_1P_0C_{-1} \tag{3}$$

$$C_3 = G_3 + P_3C_2 = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1G_0 + P_3P_2P_1P_0C_{-1} \tag{4}$$

由此，每一个进位的计算都直接依赖于加法器最开始的输入，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的延迟时间，因此最终结果的 sum 的得到只需要四个门的延迟时间。

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 1所示为 12 位进位加法器的逻辑结构图。12 位加法器划分为 3 块，最低一块由 4 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：

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

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} 为数据选择器的时延。

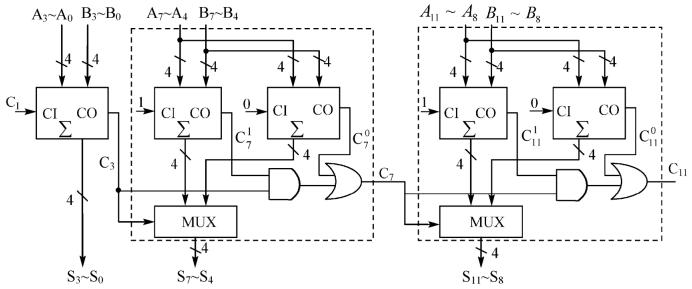

Figure 1: 12 位进位加法器原理图

四、 代码分块解读

1. 4bit 加法器设计

```
module Add_4bit(
  input[3:0] a,
  input[3:0] b,
  input c_in,

  output[3:0] sum,
  output c_out
);
  wire[3:0] g,p,c;

  assign p=a^b;
  assign g=a&b;
  assign c[0]=g[0]|(p[0]&c_in);
```

```
14      assign c[1]=g[1]|(p[1]&(g[0]|(p[0]&c_in)));
15      assign c[2]=g[2]|(p[2]&(g[1]|(p[1]&(g[0]|(p[0]&c_in)))));
16      assign c[3]=g[3]|(p[3]&(g[2]|(p[2]&(g[1]|(p[1]&(g[0]|(p[0]&c_in)))))));
17      assign c_out=c[3];
18
19      assign sum[0]=p[0]&(!g[0])^c_in;
20      assign sum[1]=p[1]&(!g[1])^c[0];
21      assign sum[2]=p[2]&(!g[2])^c[1];
22      assign sum[3]=p[3]&(!g[3])^c[2];
23
24    endmodule
```

代码设计根据式 (1) (4) 提出的 4bit 加法器原理，通过一定的化简得到。通过输入 2 个 4bit 的加数，最终输出得到 4bit 的相加结果和 1bit 的进位结果。

2. 4bit 选择进位加法器设计

```
1     module Add_middle(
2       input [3:0] a,
3       input [3:0] b,
4       input c_in,
5
6       output reg [3:0] sum,
7       output c_out
8     );
9
10      wire [3:0] sum1, sum2;
11      wire c_out1, c_out2;
12      reg c1=0;
13      reg c2=1;
14      Add_4bit A41(.a(a), .b(b), .c_in(c1), .sum(sum1), .c_out(c_out1));
15      Add_4bit A42(.a(a), .b(b), .c_in(c2), .sum(sum2), .c_out(c_out2));
16      assign c_out=c_in & c_out2 | c_out1;
17
18      always @(*)begin
19      if(c_in == 0) sum=sum1;
20      else sum=sum2;
21      end
22
23    endmodule
```

4bit 选择进位加法器的设计原理可以从图 1的单个选择位中得出，调用 4bit 加法器的 module，根据输入的加数同时计算得到不同进位的 sum 和进位值，再根据输入的进位，选择相应的结果和进位值。

3. 32bit 选择进位加法器设计

```
module Add_32bit(
  input [31:0] a,
  input [31:0] b,
  input ci,

  output [31:0]sum,
  output c_out
);
  wire c1, c2, c3, c4, c5, c6, c7;
  Add_4bit A40(.a(a[3:0]), .b(b[3:0]), .c_in(ci), .sum(sum[3:0]),
      .c_out(c1));
  Add_middle AM1(.a(a[7:4]), .b(b[7:4]), .c_in(c1), .sum(sum[7:4]),
      .c_out(c2));
  Add_middle AM2(.a(a[11:8]), .b(b[11:8]), .c_in(c2), .sum(sum[11:8]),
      .c_out(c3));
  Add_middle AM3(.a(a[15:12]), .b(b[15:12]), .c_in(c3), .sum(sum[15:12]),
      .c_out(c4));
  Add_middle AM4(.a(a[19:16]), .b(b[19:16]), .c_in(c4), .sum(sum[19:16]),
      .c_out(c5));
  Add_middle AM5(.a(a[23:20]), .b(b[23:20]), .c_in(c5), .sum(sum[23:20]),
      .c_out(c6));
  Add_middle AM6(.a(a[27:24]), .b(b[27:24]), .c_in(c6), .sum(sum[27:24]),
      .c_out(c7));
  Add_middle AM7(.a(a[31:28]), .b(b[31:28]), .c_in(c7), .sum(sum[31:28]),
      .c_out(c_out));
endmodule
```

通过调用一个 4bit 加法器和 7 个 4bit 选择进位加法器的 module 搭建出一个 32bit 的选择进位加法器，输入加数及进位，计算得到 32bit 的结果和一位的进位值。

五、 仿真结果分析

1. 4bit 加法器仿真结果

4bit 加法器的仿真结果如图 2和 3所示，从仿真结果中可以看出，无论是行为仿真或者是时序仿真都能得到正确的计算结果，同时这两种仿真在结果上没有明显区别，实验结果较为理想，符合预期要求，为之后的 32bit 选择进位加法器设计奠定良好基础。

2. 32bit 选择进位加法器仿真结果

32bit 选择进位加法器的仿真结果如图 4和 5所示。从仿真结果可以看出，代码逻辑没有问题，均能得到正确的计算结果，但是，从时序仿真中，我们可以发现，在某些时间段或时间节点，计算结果并不是理想的值，而是不定态或其他的值，这是由于电路中，几路信号在开始时有时间延迟，导致这段时间的输出结果并

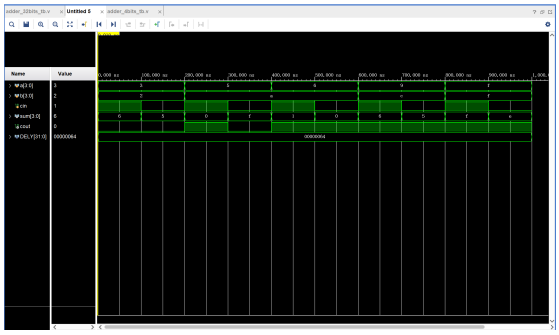

Figure 2: 4bit 行为仿真

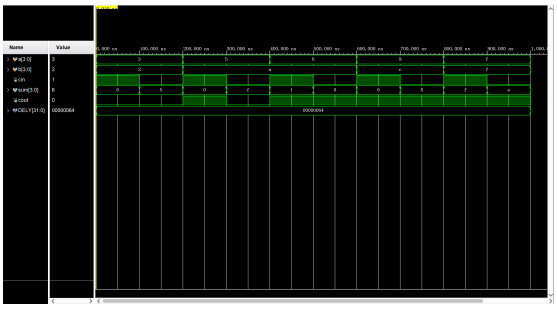

Figure 3: 4bit 时序仿真

不是输入信号的计算结果，其值时不确定的，而在输入进行切换的时候，也同样由于时间延迟，部分加法器变为切换了选择，而部分选择器还未进行切换，导致输出混乱，在短时间内出现错误结果。

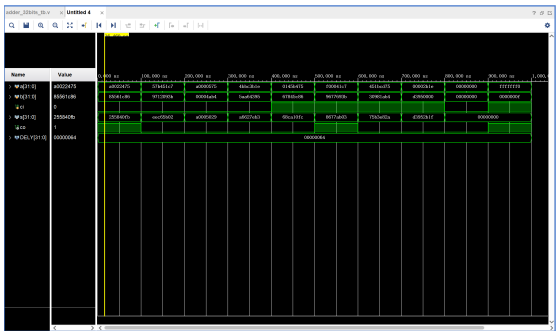

Figure 4: 32bit 行为仿真

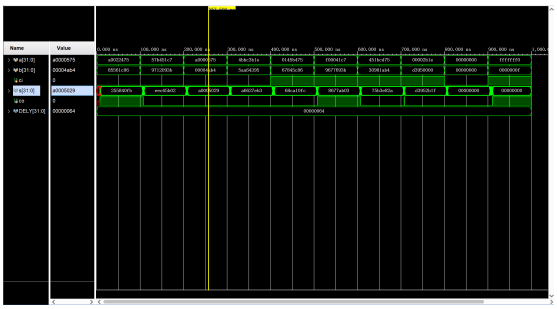

Figure 5: 32bit 时序仿真

六、　实验中遇到的问题和解决方案

在根据实验原理设计 4bit 加法器的过程中，由于对设计原理的理解不到位，导致设计出现错误，而且在仿真的过程中忽视了单独对 4bit 部分的仿真，直到在最终 32bit 加法器的仿真中出现错误，却花了很长时间在检查 32bit 的部分，浪费了很多时间。因此，在进行设计的时候，对于每一个模块都应该有相应的仿真操作来验证其正确性，保证在后面设计的正确性。

七、　思考题

1. 为什么要进行时序仿真

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。时序仿真使用的仿真器和功能仿真使用的仿真器是相同的，所需的流程和激励也是相同的；惟一的差别是为时序仿真加载到仿真器的设计包括基于实际布局布线设计的最坏情况的布局布线延时，并且在仿真结果波形图中，时序仿真后的信号加载了时延，而功能仿真没有。

时序仿真相对于功能仿真或者说行为仿真而言更接近实际电路，它根据实际的电路时序得到输出结果，从本次的设计中就可以看出，时序逻辑更能暴露出电路设计会出现的细节问题，在设计过程中除了逻辑正确之外，构成电路后也会存在其他的问题。

2. 采用流水线技术有什么优缺点

所谓流水线设计，实际上就是把规模较大、层次较多的组合逻辑电路分为几级，在每一级插入寄存器组并暂存中间数据。K 级流水线就是从组合逻辑的输入到输出恰好有 K 个寄存器组，上一级的输出是下一级的输入而又无反馈的电路。

因此，流水线可以提高电路的吞吐率，即数据输入机器的速率或处理数据的速率。每种情况下，组合逻辑都限制了机器的性能。作为一种能够提高电路性能的可选方法，可以将流水线型寄存器插入到组合逻辑的关键位置上，将逻辑分割成具有更短路径的群组。流水线技术减少了组合逻辑中的级数，缩短了存储元件之间的数据通路，并且因为能用更高的时钟频率而提高了电路的吞吐能力。

同时，流水线提高了电路的频率。它通过减少每个模块的时间长度，达到提升频率的效果。

但是，流水线技术产生了输入-输出延迟：由于插入了流水线寄存器，会导致第一个输出必须在 K 个 clk 之后才能得到。而之后每个 clk 都能得到一个输出。流水线技术是用空间（硬件）的复杂度来换取时间（性能）的复杂度。

所以，设计中应采用最小数目的流水线寄存器来获取最短的循环周期。在关键路径上的时间裕度无法满足时，才要考虑使用流水线技术。