16位超前进位加法器的设计

|  |  |
| --- | --- |
| 姓名： | 刘金硕 |
| 学号： | 202428015926051 |

# 目录

[目录 1](#_Toc150846839)

[1 设计规格与要求 1](#_Toc150846840)

[2 算法原理与算法设计 1](#_Toc150846841)

[3 实现架构 1](#_Toc150846842)

[4 仿真结果 2](#_Toc150846843)

[5 综合结果 3](#_Toc150846844)

# 设计规格与要求

实现快速加法器组合逻辑，要实现的功能如下：

输入为两个16位有符号数，输出17位相加结果。要求采用超前进位（Carry-look-ahead）结构。

计算例子：

**0**110000010000000 + **1**000000000000001 = **1**1110000010000001

(24704) + (-32767) = (-8063)

设计要求：

Verilog实现代码可综合，逻辑延迟越小越好，给出综合以及仿真结果（参考ASIC综合结果：SMIC 55nm工艺下工作时钟频率大于500 MHz）。

# 算法原理与算法设计

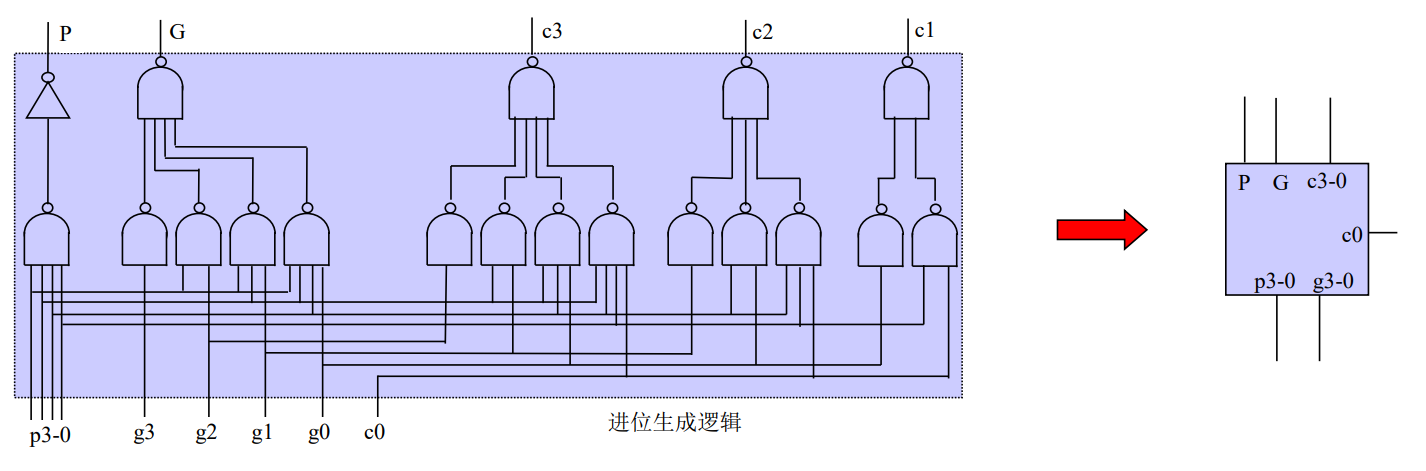
本设计拟采用块间并行，块内超前进位实现。超前进位加法器（Carry Look-Ahead Adder，CLA）可以更快地产生和传播进位，从而减少了加法的延迟。超前进位加法器的基本原理是通过递推关系并行计算进位，而不依赖中间位的进位值。计算通过下式完成：

其中G和P分别为生成因子（Generate）和传递因子P（Propagate）。生成因子G决定了每一位或是每一个块是否产生进位，。传递因子决定了每一位或一个块的输入进位是否可以传递到输出,。通过使用这种生成和传递的方式，超前进位加法器可以在不等待前一位的进位计算完成的情况下，同时计算多个位的进位。这使得进位的传播更快，整个加法器的性能更高。

本设计首先构建4位的超前进位加法器lca\_4，输入四位p和g以及输入进位ci，输出进位co以及本块的传递因子与生成因子P，G。16位加法可以由五个lca\_4构成，其中四个lca得到相应位的co，再将四个块的P和G输入进一个lca，计算第4、8、12、16位的co，然后根据0-15位的co计算sum[15:0]，以及溢出情况overflow，最后根据co[16]以及overflow值计算sum[16]，至此完成计算。

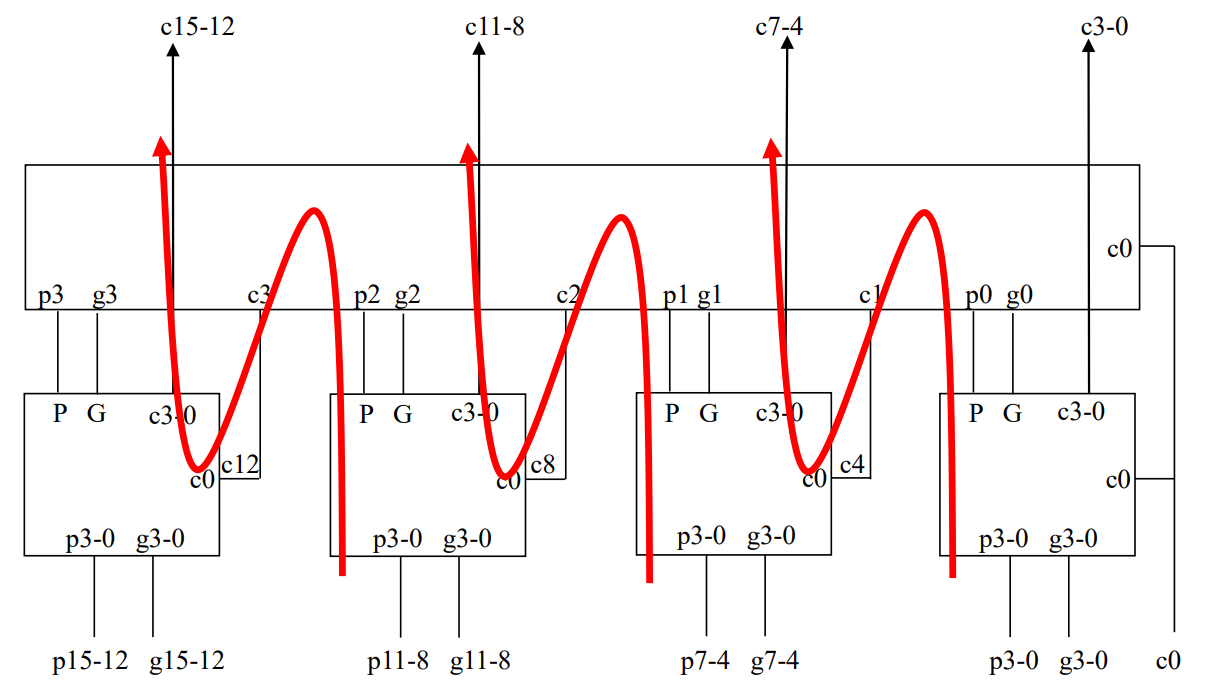
# 实现架构

**4位超前进位加法器架构**



**顶层16位加法器架构**

将上面的四位超前进位加法器连接成下面的树形结构，可以计算全部co值，接下来计算sum值逻辑较为简单不在展示。



**模块端口说明**

**top module：add\_tc\_16\_16**

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **方向** | **位宽** | **描述** |
| a | I | 16 | 输入数据，二进制补码 |
| b | I | 16 | 输入数据，二进制补码 |
| sum | O | 17 | 输出和a + b，二进制补码 |

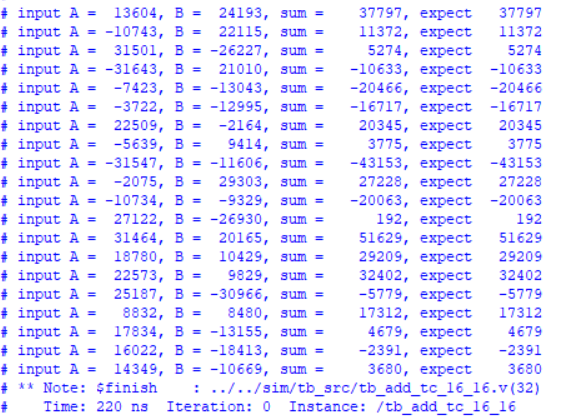
设计顶层模块，输入两个16位二进制补码a和b，输出sum=a+b，为17位补码，其中若发生溢出则第17位为溢出值，若未发生溢出则为第16位即符号位的扩展。

**module: lca\_4**

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **方向** | **位宽** | **描述** |
| g | I | 4 | 输入生成因子 |
| p | I | 4 | 输出传递因子 |
| ci | I | 1 | 输入进位 |
| G | O | 1 | 输出本块的生成因子 |
| P | O | 1 | 输出本块的传递因子 |
| co | O | 3 | 输出进位c0-c2 |

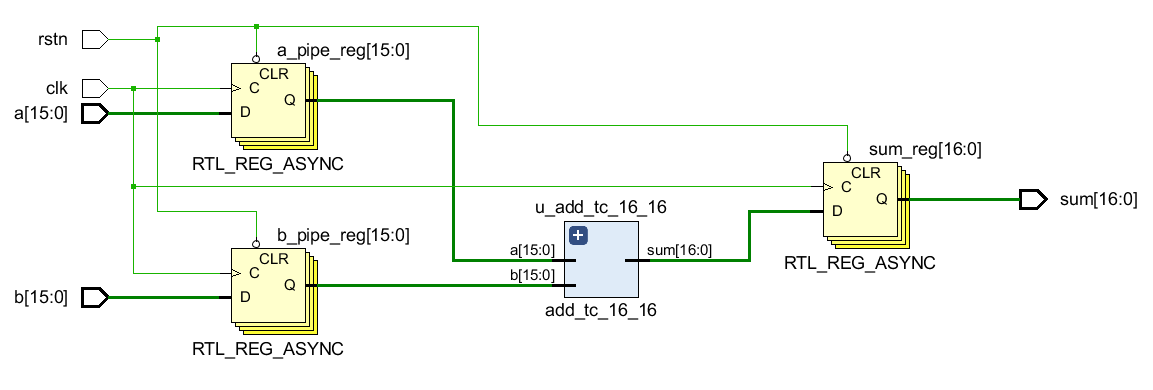
# 仿真结果

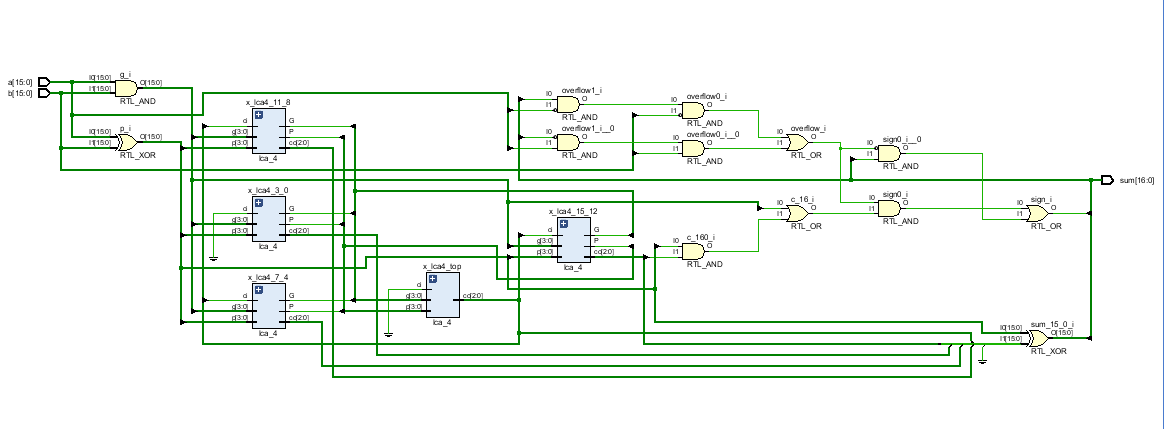
testbench中随机输入20个加数A,B，用monitor监测输入与输出值，在tcl控制台打印得到的加法结果如下，可以看到对于20次随机输入，输出均与正确结果一致（expect由+运算符得到），可以认为模块结果正确。



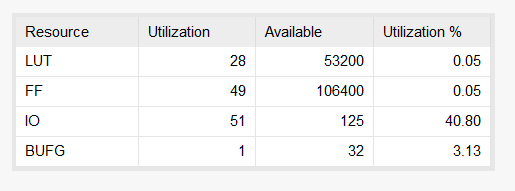
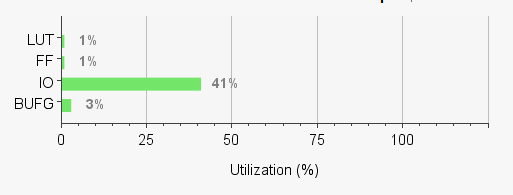
# 综合结果

**RTL SCHEMATIC**



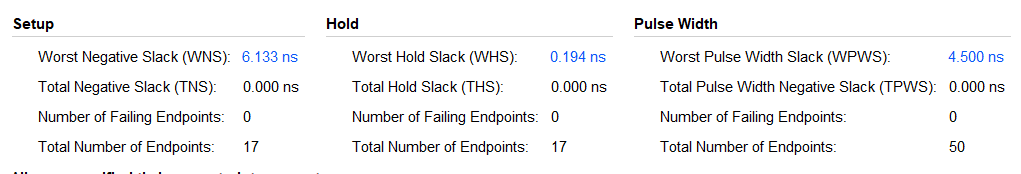


**FPGA资源使用**



**时序分析**

时钟约束为100MHZ，时序分析结果如下：



可以计算出最大工作频率约为