16位快速乘法器的设计

|  |  |
| --- | --- |
| 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 2](#_Toc150846842)

[4 仿真结果 4](#_Toc150846843)

[5 综合结果 4](#_Toc150846844)

# 设计规格与要求

实现快速乘法器组合逻辑，要实现的功能如下：

输入为两个16位有符号数，输出32位相乘结果。要求采用Booth编码和Wallace树型结构。

计算例子：

0110000010000000 \* 1000000000000001 = 11001111110000000110000010000000

(24704) \* (-32767) = (-809475968)

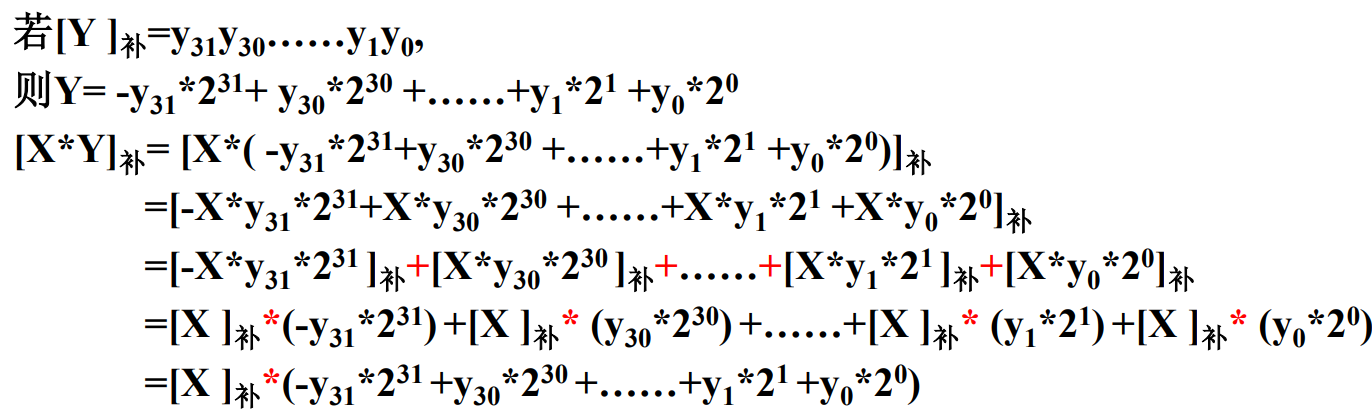
设计要求：

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）。

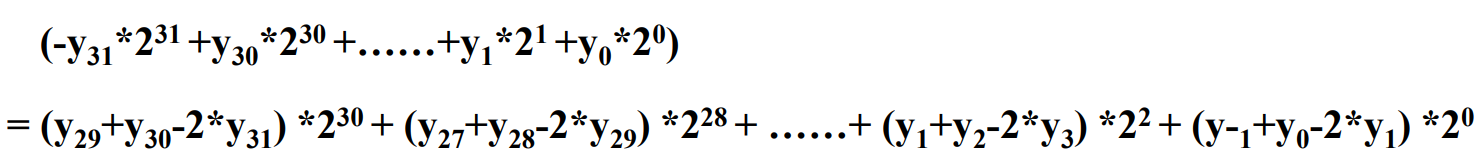
# 算法原理与算法设计

本设计采用booth编码+wallace树实现乘法器，其中核心算法便是booth编码以及华莱士树。

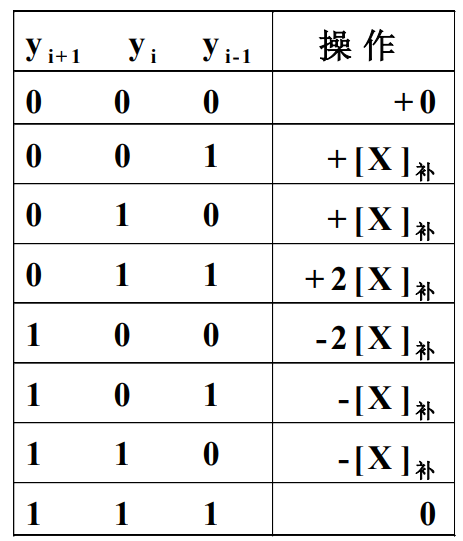
首先介绍阵列乘法器原理以及booth编码。乘法运算的输入数据是被乘数X和乘数Y，则乘法运算可以展开为：



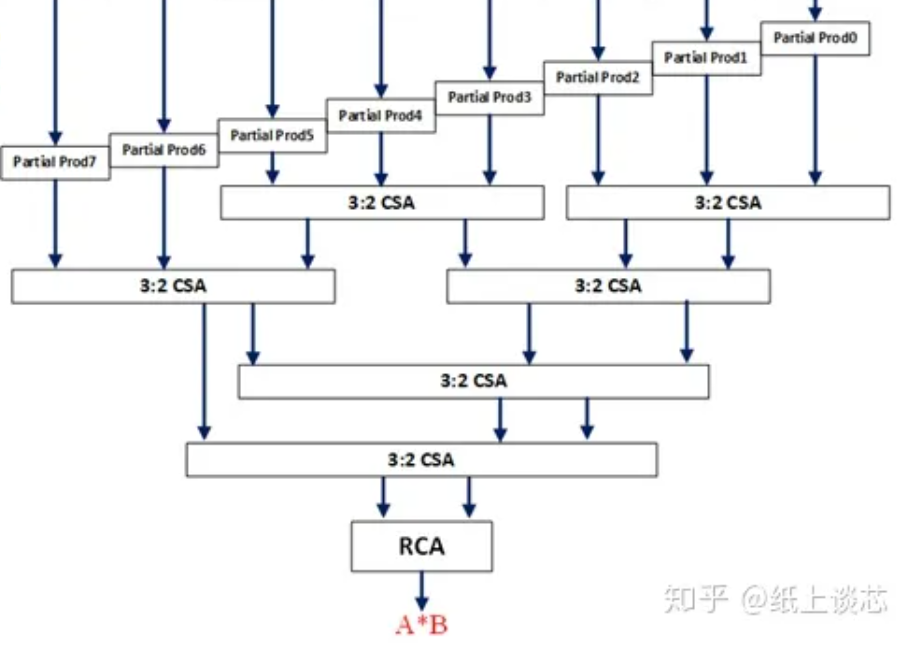
这相当于Y的每一位与X相乘，产生部分积，对于16位乘法就会产生16个部分积，后续需要将这16个部分积相加，效率较低，而booth编码则能够减小部分积的个数，提升计算效率。将上面的X\*Y表达式中（）内的乘数Y各项进行变换：



由此就可以将部分积的个数减少一半，这样的乘数变换称为radix4 booth编码。进行booth编码时，每三位进行一次编码，编码值对应为对被乘数X的操作，radix4 booth编码的值包括0，+1，-1，+2，-2，具体编码规则见下表。

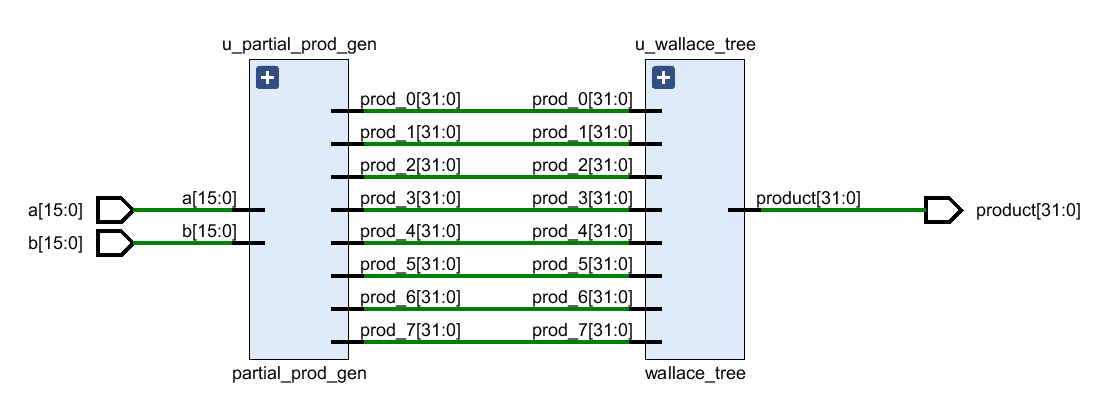


将乘数X进行booth编码后，接下来根据每一项编码值对被乘数进行操作，得到8个部分积的，下面将这8个部分积相加就可以得到最终结果。将部分积相加采用的硬件结构是华莱士树。华莱士树的基本思想是将全加器视为一个3-2压缩器，即将三个数（A，B，Cin）的加法转换为2个数的加法（S，Cout），最终逐层压缩，直到只有两个加数，最后在进行一次加法就可以完成运算。采用华莱士树能够降低部分积求和的延迟，本设计中对8个部分积进行加法的华莱士树结构如下图，其中CSA为进位保留加法器，即将每一bit加法的进位进行保留并输出，不传递给下一位，CSA实现了将三个数加法转换为两个数加法。

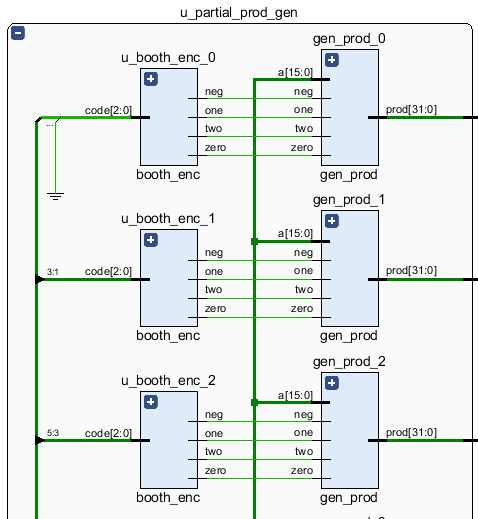


# 实现架构

经过上面算法部分的分析，本设计可以划分为booth编码并求部分积，以及通过华莱士树结构将部分积相加两个部分，如下图。



产生部分积的partial\_prod\_gen模块又可以分为booth编码以及生成部分积两个部分，如下图。



**模块端口说明**

**top module：mul\_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 |
| product | O | 32 | 输出乘积a \* b，二进制补码 |

**module: partial\_prod\_gen**

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **方向** | **位宽** | **描述** |
| a | I | 16 | 输入被乘数 |
| b | I | 16 | 输入乘数 |
| prod\_0(0-7) | O | 32 | 输出部分积 |

**module: booth\_enc**

|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **方向** | **位宽** | **描述** |
| code | I | 3 | 输入3位被乘数项 |
| neg | O | 1 | booth编码是否含负号 |
| zero | O | 1 | booth编码是否为0 |
| one | O | 1 | booth编码是否绝对值为1 |
| two | O | 1 | booth编码是否绝对值为2 |

**module: prod\_gen**

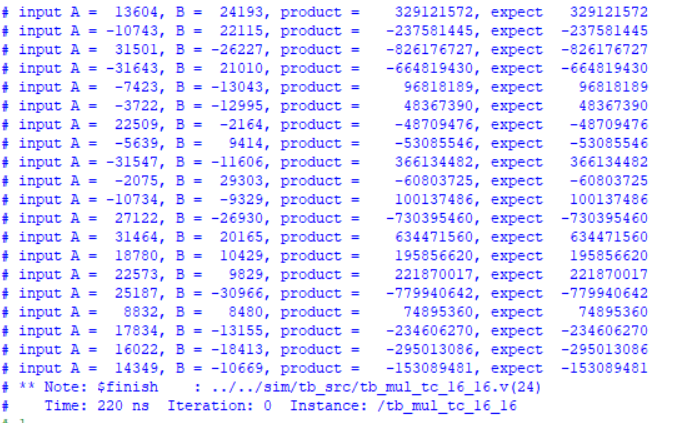
|  |  |  |  |
| --- | --- | --- | --- |
| **名称** | **方向** | **位宽** | **描述** |
| a | I | 16 | 被乘数 |
| neg | I | 1 | booth编码是否含负号 |
| zero | I | 1 | booth编码是否为0 |
| one | I | 1 | booth编码是否绝对值为1 |
| two | I | 1 | booth编码是否绝对值为2 |
| prod | O | 32 | 输出部分积 |

# 仿真结果

首先先检测题目要求中给出的计算例子(24704) \* (-32767) = (-809475968)，波形如下图，可以看到乘积结果正确。

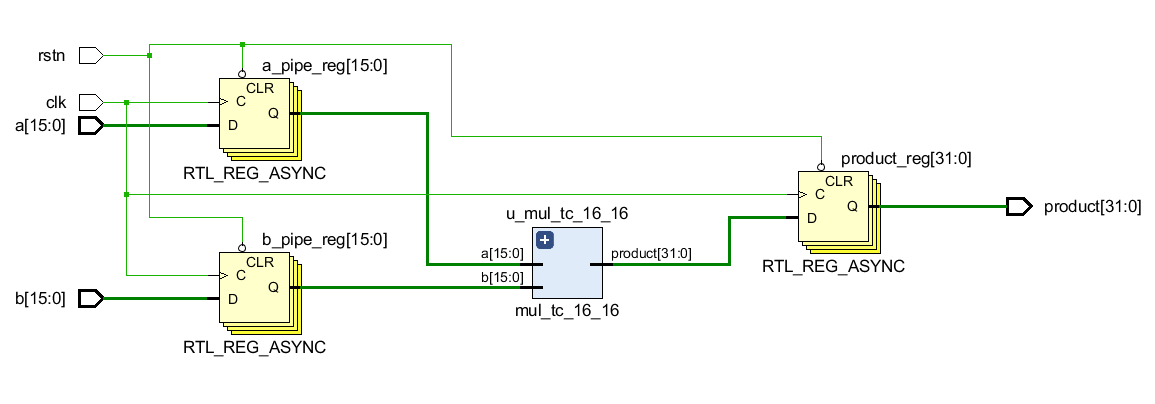


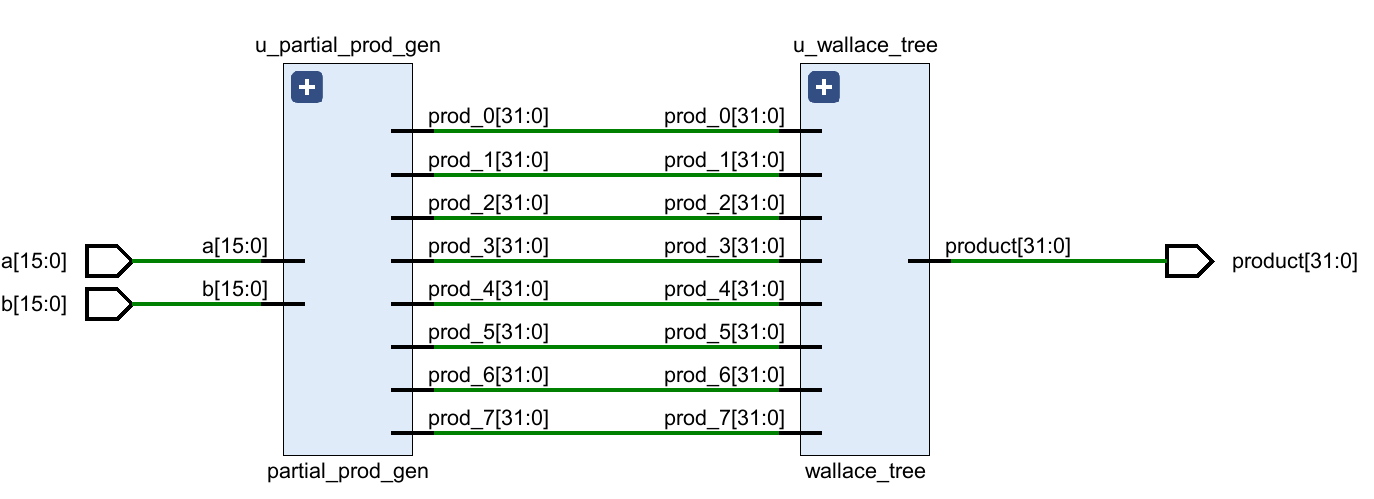
接下来在testbench中随机输入20个输入A,B，用monitor监测输入与输出值，在tcl控制台打印得到的加法结果如下，可以看到对于20次随机输入，输出均与正确结果一致（expect由\*运算符得到），可以认为模块结果正确。



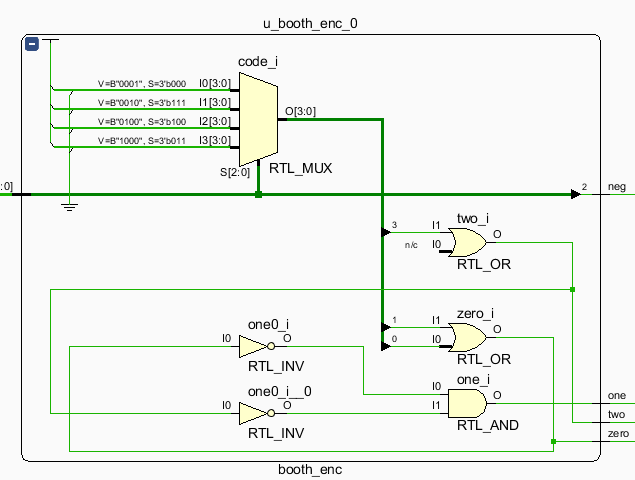
# 综合结果

**RTL SCHEMATIC**

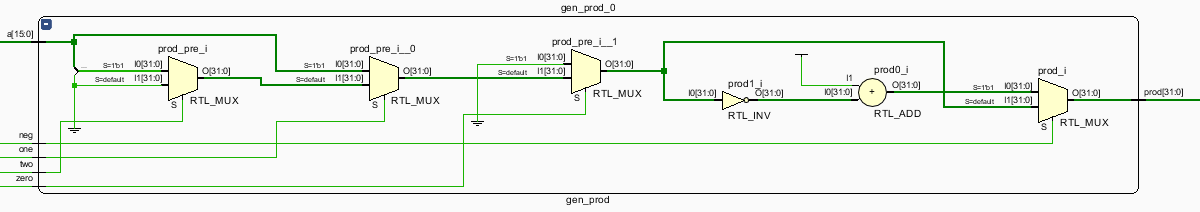




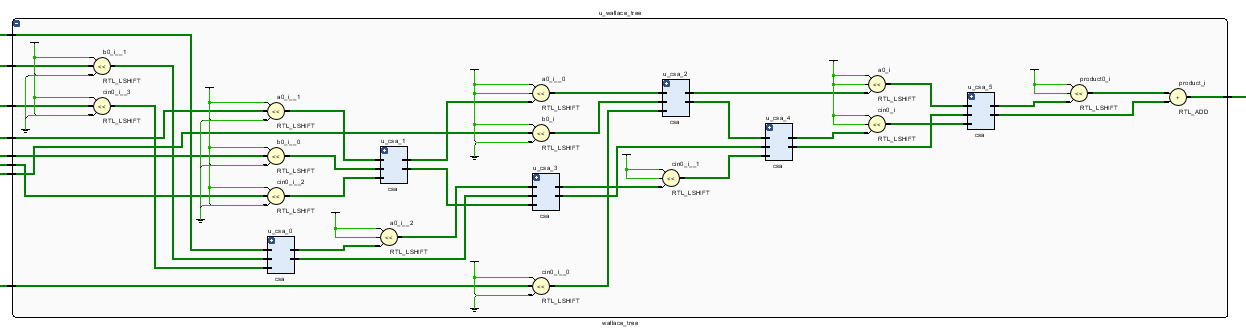
**booth\_enc模块**



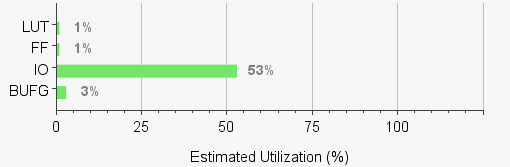
**gen\_prod模块**

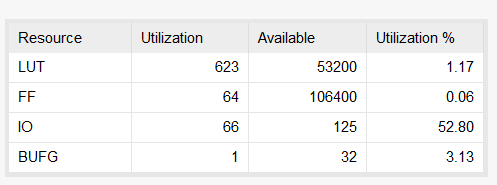


**wallace\_tree模块**



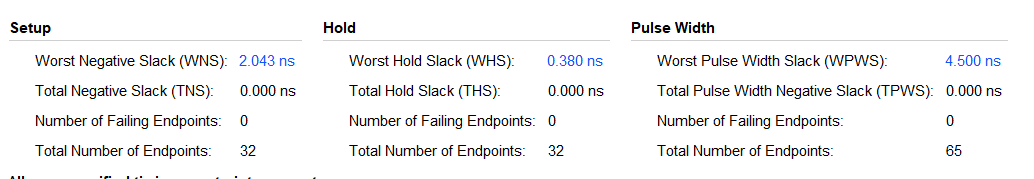
**FPGA资源使用**





**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