

**青 岛 科 技 大 学**

**本科综合课程设计报告**

**题 目 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx**

曲英杰、高树静、王增

**指导教师\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**辅导教师\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

曲英杰、高树静、王增

**学生姓名\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

xxx

**学生学号\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

xxx

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_院（部）\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_专业\_\_\_\_\_\_\_\_\_\_\_\_班**

集成电路设计与集成系统

xxx

信息科学技术学院

xxxx

**\_\_\_\_\_\_年 \_\_\_月 \_\_\_日**

xx

xx

集成电路设计与集成系统

集成电路设计与集成系统

|  |  |  |  |
| --- | --- | --- | --- |
| 课程设计地点 | 青岛科技大学 | 课程设计时间 | 2016.12.26~2017.1.5 |
| 课程设计目的 | 通过本次课程设计，灵活运用所学知识，掌握超前进位加法器的原理和实现方法，熟悉FPGA的设计和开发过程，在实际操作中找到不足和存在的问题，不断提升自己，进一步学习FPGA的相关操作，使自己在学习和实践中积累更多经验。 | | |
| 指导教师评语 | 指导教师签字： | | |
| 验收（答辩）意见 | 验收（答辩）人员签字： | | |
| 系（教研室）  意 见 | 教研室主任签字： | | |
| 成 绩  （百分制） |  | | |

**目 录**

摘要…………………………………………………………………………………1

1. 加法器简介………………………………………………………………………1

2. 基本加法器………………………………………………………………………1

2.1 半加器…………………………………………………………………………1

2.2 全加器…………………………………………………………………………2

2.2.1 1位全加器…………………………………………………………………2

2.2.2 4位全加器…………………………………………………………………3

3. 超前进位加法器…………………………………………………………………3

3.1 超前进位加法器的原理………………………………………………………3

3.2 4位超前进位加法器…………………………………………………………4

3.3 16位超前进位加法器…………………………………………………………5

4. 设计方案…………………………………………………………………………6

4.1 设计原理………………………………………………………………………6

4.2 建立Verilog模型………………………………………………………………6

4.2.1 4位超前进位加法器………………………………………………………6

4.2.2 16位超前进位加法器……………………………………………………7

4.2.3测试模块……………………………………………………………………7

4.3 功能仿真………………………………………………………………………7

4.4 逻辑综合………………………………………………………………………9

4.5 对比分析……………………………………………………………………10

5. 总结………………………………………………………………………………11

参考文献……………………………………………………………………………11

附录…………………………………………………………………………………12

**摘 要**

运算器是计算机的重要组成部分，其核心部件是加法器，而加减乘除四种运算都是通过加法器实现，因此要想加快运算器的速度实质上就是加快加法器的速度。本文对传统的加法器电路进行改进，采用一种组内并行组间串行的超前进位设计方法，设计实现16位超前进位加法器，不仅可以克服串行进位加法器速度低的缺点，也可以解决单纯的超前进位加法器带负载能力不足等问题，从而在实际电路中使加法器的运算速度和面积达到最优。

**关键词：**加法器；超前进位；组内并行组间串行；运算速度；面积

**1.** 加法器简介

计算机由具有各种逻辑功能的逻辑部件组成，这些逻辑部件按功能结构又可分为组合逻辑电路和时序逻辑电路。组合逻辑电路是由门电路组合而成的逻辑电路，加法器就属于组合逻辑电路[1]。

在计算机中，数据的运算都是以二进制数为单位，而两个二进制数之间的算术运算无论是加、减、乘、除，最后都是化作若干步相加运算来进行，所以加法运算是最重要也是最基本的运算。加法器的速度直接决定了整个电路的速度，对加法器的电路进行改进就显得尤为重要[2]。因为加法运算存在进位问题，使得某一位计算结果的得出和所有低于他的位相关，为了减少进位传输所消耗的时间，提高计算速度，人们设计了多种类型的加法器。

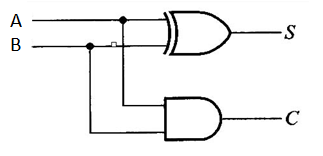
其中串行进位加法器是最简单的加法器，对于 n 位的串行进位加法器，其延时与 n 成正比，即时间复杂度是。对于相同时间复杂度的曼切斯特进位链加法器进位逻辑比较简单，并且具有较低的功耗。进位跳跃加法器是将操作数分组，每组内加法以串行进位方式进行，组之间的进位信号有条件的从低位向高位进行传播，进位跳跃加法器的时间复杂度是。进位选择加法器实际是两个并行的加法器，一个执行进位为 0 时的情况，另一个执行进位为 1 时的情况，然后根据进位情况直接选择正确的结果输出，它的时间复杂度是。超前进位加法器是当前研究和应用最为广泛的一种加法器，它是利用大量的硬件电路来实现进位逻辑，以减少进位延迟，因此它的运算速度很快，它的时间复杂度是[3]。



**2.** 基本加法器

**2.1** 半加器

半加器是指对两个输入位相加，输出一个结果位和进位，没有进位输入的加法器电路，即是实现两个一位二进制数的加法运算电路。其电路图和真值表分别如图1和表1所示。



**图1** 半加器电路图

**表1** 半加器真值表

|  |  |  |  |
| --- | --- | --- | --- |
| 输入 | | 输出 | |
|  |  |  |  |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

由表1可以直接得到半加器的逻辑表达式为：

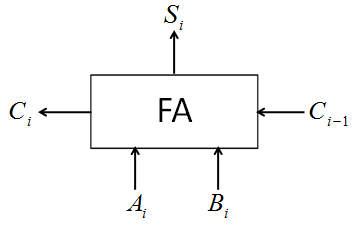
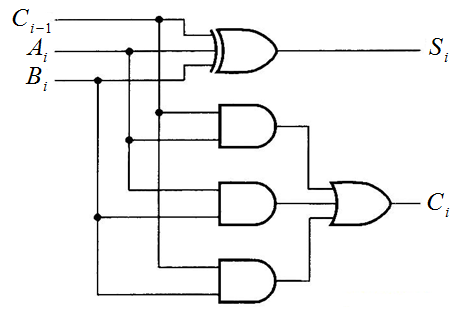


其中S为输入位A和B相加的和，可以看出半加器实质上就是逻辑电路中的异或门。半加器其实可以看做最简单的加法器，从功能上来看自身很难实现复杂的运算，一般与其它加法器组合在一起，共同实现完整的功能。

**2.2** 全加器

**2.2.1** 1位全加器

在实际作二进制加法运算时，一般两个加数都不会是一位，而是多位的。因而需要考虑从低位而来的进位，这种电路称为全加器。全加器的电路图和逻辑符号以及真值表分别如图2和表2所示。



**图2** 全加器电路图和逻辑符号

**表2** 全加器真值表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 输入 | | | 输出 | |
|  |  |  |  |  |
| 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 |

通过表2可知全加器的逻辑表达式为：

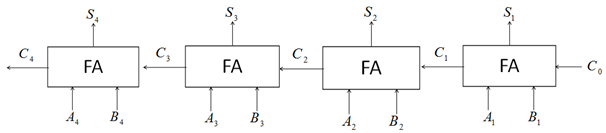


其中和表示两个加数，表示和，表示来自低位的进位，表示向高位的进位。



**2.2.2** 4位全加器

4位全加器可以采用4个1位全加器级连组成串行进位加法器，如图3所示。



**图3** 4位串行进位加法器

串行进位加法器具有器件少、成本低的优点，但因为高位运算必须等到低位进位到来之后才能进行，且延时随着操作数位数的增加而线性增大，所以它的延迟非常可观，运算速度太慢，这种电路对速度要求比较高的器件显然达不到要求。为此，需要对此电路进行改进，下面介绍的超前进位加法器就很好的解决了速度要求。

**3.** 超前进位加法器

**3.1** 超前进位加法器的原理

对于两个多位数相加的情况，每一位都是带进位相加，这时，只要依次将低位的进位输出接到高位的进位输入，就可以构成多位加法器。对于多位加法器而言，每一位的相加结果都必须等到低一位的进位产生以后才能建立起来，那么这种加法器速度缓慢的缺点也是显而易见的。就4位加法器算法而言，在最坏情况下，做一次加法运算需要经过4个全加器的传输时间才能得到最终运算结果。对串行进位加法器研究可得，运算的延迟主要是由于进位的延迟引起的，为了提高运算速度，必须尽量减小由于进位信号逐级传递所耗费的时间。所以最好能在相加运算一开始就得到高位的进位输入信号[4]。

根据上述原理，如果通过逻辑电路事先得出加到每一位全加器上的进位输入信号，而不是从最低位开始逐位传递进位信号，就可以有效地提高运算速度，节省运算时间，把实现这种加法的器件叫做超前进位加法器。超前进位加法器能够有效减少进位的延迟，它由进位门产生进位，各进位彼此独立，不依赖于进位传播，因此延迟非常小，速度非常高。

由公式(2)可知，“”取决于本位参加运算的两个数，而与低位进位无关，因此称为进位产生函数(本次进位产生)，用表示，其含义是：若本位的两个输入均为1，必然要向高位产生进位。“”则不但与本位的两个数有关，还依赖于低位送来的进位，因此称为进位传递函数(低位进位传递)，用表示，其含义是：当两个输入中有一个为1，低位传来的进位将向更高位传送，所以进位表达式又可以写成：

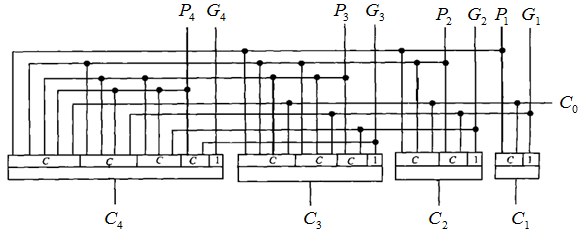


**3.2** 4位超前进位加法器

在超前进位加法器中，当进位输入进入超前进位模块，各级的进位输出将同时产生，每一位的进位输出与“和”值输出都与前面的位无关，即加法时间与位数无关，因此超前进位的特点是各级进位信号同时产生，有效的消除了逐级进位效应。根据公式(3)可以得到4位超前进位加法器的4个进位：



图4是4位超前进位加法器的逻辑图，结合公式(4)可以发现，所有各位的进位均不依赖于低位的进位，各位的进位可以同时产生。



**图4** 4位超前进位加法器逻辑图

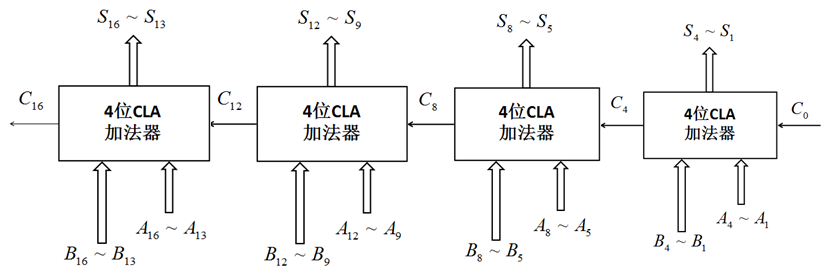
这种进位方式是快速的，若不考虑、的形成时间，从的最长延迟时间仅为，为两极门的延迟。但是随着加法器位数的增加，的逻辑表达式会变得越来越长，输入变量会越来越多，电路结果会变得越来越复杂，所以完全采用并行进位是不现实的[5]。另外，加法时间与位数无关只是一种理想情况，实际条件下延时是随位数呈线性增加的，加法器的面积也随着位数增大而迅速增加。因此，超前进位加法器结构很难实现位数较高的加法。研究表明超前进位加法器的位数为4时最佳，这时超前进位加法器可以在速度与面积之间取得最优的折衷。



**3.3** 16位超前进位加法器

对于16位加法器的设计，如果仍然采用上述的迭代公式(3)，虽然最终各位的进位不会依赖之前的进位，但是越往后的进位迭代关系越长，这会导致电路的结构极为复杂，占用面积很大，而且电路的带负载能力有限，将难以驱动，所以完全采用并行进位是不可取也是不现实的。

如果将16位的超前进位加法器划分为4组，即4位为一组，在每个组内实现并行进位，而组与组之间采用串行进位，则可以既保证有较小的延迟，又可以减小电路的复杂程度，占用较小的面积，使16位超前进位加法器可以在速度与面积之间达到最优[6]。图5是组内并行组间串行的16位超前进位加法器逻辑图。



**图5** 组内并行组间串行的16位超前进位加法器逻辑图

以第一组为例，组内的进位逻辑函数、、、的表达式与前述相同，都由公式(4)产生。的信号是同时产生的，从出现到产生的延迟时间是。纵观整个16位超前进位加法器，若不考虑、的形成时间，从的最长延迟时间仅为，其中为分组的组数，那么从的延迟时间是。



**4.** 设计方案

**4.1** 设计原理

首先要考虑两个传递函数；进位产生函数和进位传送函数。



当时会产生进位，而当时会使进位输入传到输出。根据全加器原理，第位的进位及和的输出可表示为：



由此可得各位的进位输出表达式：



这样函数和产生一级时间延迟，经过计算也将产生一级时间延迟，则、输入一旦产生即可算出结果。



**4.2** 建立模型



**4.2.1** 4位超前进位加法器模块

首先我们进行4位超前进位加法器的设计，由设计方案中所述的算法，定义一个名为fulladder4的模块，该模块中的关键代码如下：

assign g0=a[0]&b[0],

g1=a[1]&b[1],

g2=a[2]&b[2],

g3=a[3]&b[3];

该段代码描述了进位产生函数，和是参与运算的两个数，与低位进位无关，具体公式为，若本位的两个输入均为1，必然要向高位产生进位。



assign p0=a[0]^b[0],

p1=a[1]^b[1],

p2=a[2]^b[2],

p3=a[3]^b[3];

该段代码描述了进位传送函数，具体为，此时不但与本位的两个数有关，还依赖于低位送来的进位，当两个输入中有一个为1，低位传来的进位将向更高位传送。



assign c1=g0|(p0&c\_in),

c2=g1|(p1&g0)|(p1&p0&c\_in),

c3=g2|(p2&g1)|(p2&p1&g0)|(p3&p2&p1&p0&c\_in),

c4=g3|(p3&g2)|(p3&p2&g1)|(p3&p2&p1&g0)|(p3&p2&p1&p0&c\_in);

该段代码描述了4位超前进位加法器的4个进位，由设计原理中的进位输出表达式获得，这些信号同时产生，与之前的进位无关。



assign sum[0]=p0^c\_in,

sum[1]=p1^c1,

sum[2]=p2^c2,

sum[3]=p3^c3;

该段代码描述的是4位的和，由得到。



**4.2.2** 16位超前进位加法器模块

接下来定义名为fulladder16的16位超前进位加法器，主要通过调用之前的4位超前进位加法器得到，关键代码如下：

fulladder4 i1(sum[3:0],c3,a[3:0],b[3:0],c\_in);

fulladder4 i2(sum[7:4],c7,a[7:4],b[7:4],c3);

fulladder4 i3(sum[11:8],c11,a[11:8],b[11:8],c7);

fulladder4 i4(sum[15:12],c15,a[15:12],b[15:12],c11);

该段代码连续调用了4次4位超前进位加法器，要明确每次调用时的和，向高位的进位，两个加数，来自低位的进位。

**4.2.3** 测试模块

编写测试文件，由于不可能将所有能列举的输入一一举例，所以在此输入一部分具有代表性的数据，例如可以让每四位数产生级间进位，观察结果是否正确，测试文件关键代码如下所示：

initial

begin

a=5;b=4;c\_in=0;

#20 a=2;b=6;c\_in=1;

#20 a=4'b1101;b=4'b0011;c\_in=0;

#20 a=8'b1101\_0011;b=8'b0110\_0001;c\_in=1;

#20 a=12'b0101\_1100\_0101;b=12'b1100\_0000\_0101;c\_in=0;

#20 a=16'b1011\_0001\_1100\_1010;b=16'b1100\_0000\_0011\_0000;c\_in=1;

end

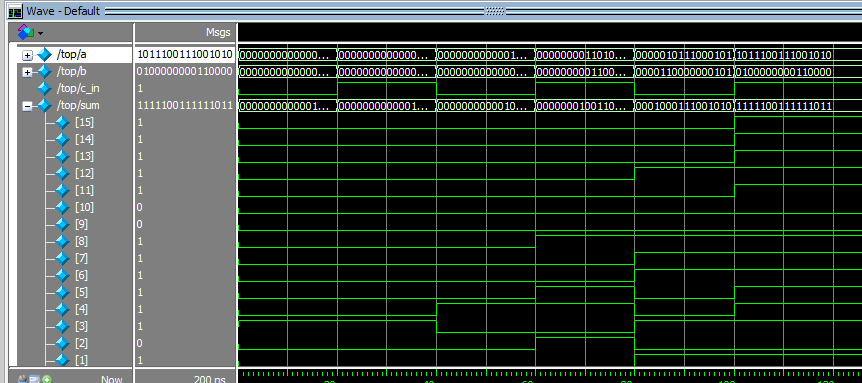
在这里进行了6组数据的测试，毎等待20个时间单位对一组数据进行测试。其中第1、3、5组数据的初始进位为0；第2、4、6组数据的初始进位为1。

**4.3** 功能仿真

功能仿真是验证所设计的模型的功能是否正确和符合要求，不涉及设计实现后实际电路的时延等问题。利用仿真工具对组内并行组间串行的16位超前进位加法器的模型进行功能仿真。

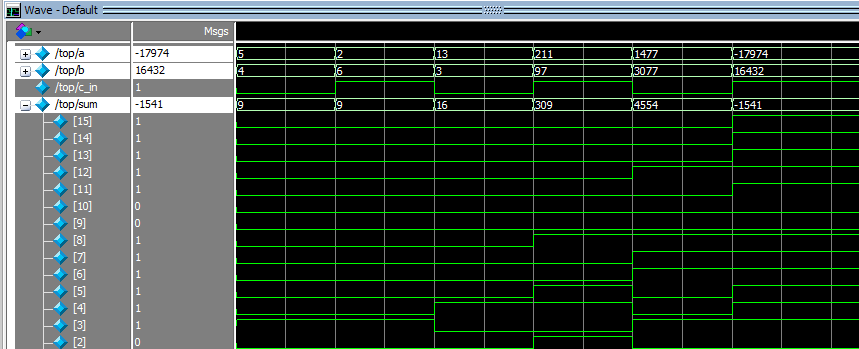


具体仿真波形图如图6所示，图中形象的显示了两个加数，初始进位位，和的数值波形。此外可以将和展开，便可观察到毎4位超前进位为一组的进位，以及进位产生函数、进位传送函数的数值。



**图6** 仿真波形图

为了更直观的观察每组测试数据的结果，将两个加数以及产生的和用十进制数的形式表达出来，具体如图7所示。



**图7** 十进制仿真波形图

通过图7可知各组数据的测试结果如下所示：

第1组数据，输入数据：a=5,b=4,c\_in=0，输出数据：sum=9；

第2组数据，输入数据：a=2,b=6,c\_in=1，输出数据：sum=9；

第3组数据，输入数据：a=13,b=3,c\_in=0，输出数据：sum=16；

第4组数据，输入数据：a=211,b=97,c\_in=1，输出数据：sum=309；

第5组数据，输入数据：a=1477,b=3077,c\_in=0，输出数据：sum=4554。

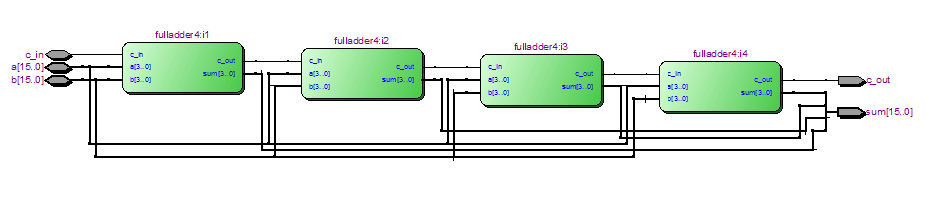
第6组数据，输入数据：a=-17974,b=16432,c\_in=1，输出数据：sum=-1541。

**4.4** 逻辑综合

逻辑综合是将硬件描述语言描述的设计输入转换成与或非门、触发器、存储器等基本元件的连接关系，根据约束条件对生成的门级逻辑连接进行优化，生成为下一步实现所利用的网表文件。由于组内并行，组间串行的16位超前进位加法器模块在进行综合是问题较多，首先，不可能实现data\_in的正常输入，因为时钟周期的速度较快，外部根本无法跟上其存储输入(实际该模块的输入是与其他模块搭配使用的，而两个模块必须在同一时钟下运作) [7]。

另外，开发板上无法找到32个输入端口(主要是设备较旧)，再次很难得到一周期的使能信号，还有一个是输出也很难找到32个连续的led灯，输出较困难，其中较容易解决的是一周期的使能信号：可以通过按键输入该信号，但需要进行按键消抖操作，可为其专门设置一个按键消抖模块，经按键消抖之后在将数据送到该运算器，还有一个可解决的是输入问题，可通过复位置数方式导入。

在经过以上优化之后，16位超前进位加法器模块的RTL Viewer图如图8所示：



**图8** 16位超前进位加法器模块RTL Viewer图

用进行逻辑综合，首先将程序中的input改为inout：



在fulladder16模块中：

inout [15:0] a,b;

inout c\_in;

之后将测试数据写入其中，因为每次只能写一组测试数据，所以分6次写完6组数据：

assign a=0000\_0000\_0000\_0101;

assign b=0000\_0000\_0000\_0100;

assign c\_in=0;

assign a=0000\_0000\_0000\_0010;

assign b=0000\_0000\_0000\_0110;

assign c\_in=1;

assign a=0000\_0000\_0000\_1101;

assign b=0000\_0000\_0000\_0011;

assign c\_in=0;

assign a=0000\_0000\_1101\_0011;

assign b=0000\_0000\_0110\_0001;

assign c\_in=1;

assign a=0000\_0101\_1100\_0101;

assign b=0000\_1100\_0000\_0101;

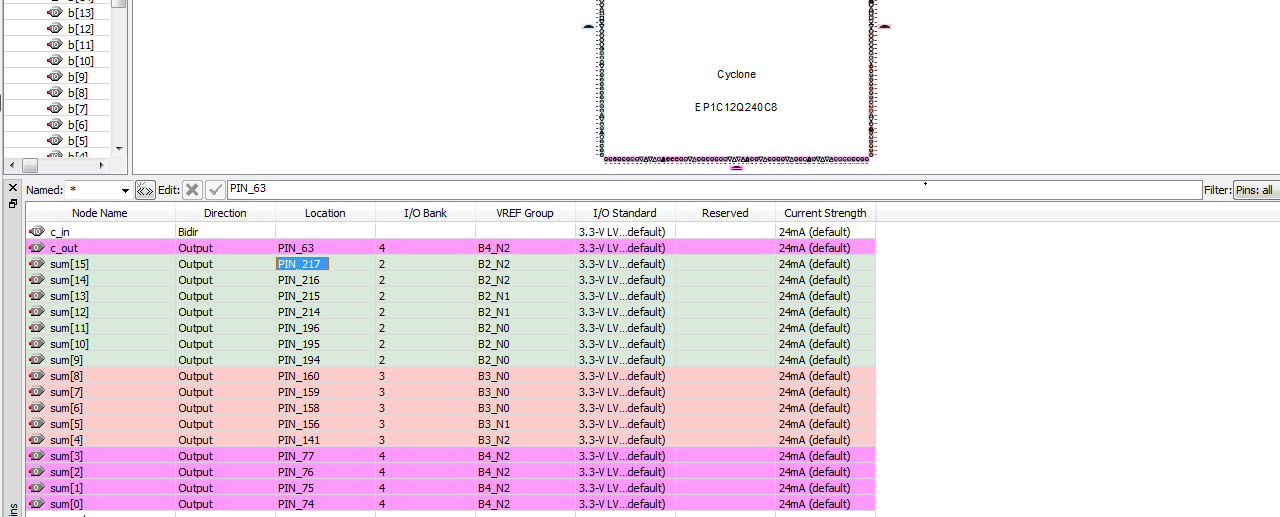
assign c\_in=0;

assign a=1011\_0001\_1100\_1010;

assign b=1100\_0000\_0011\_0000;

assign c\_in=1;

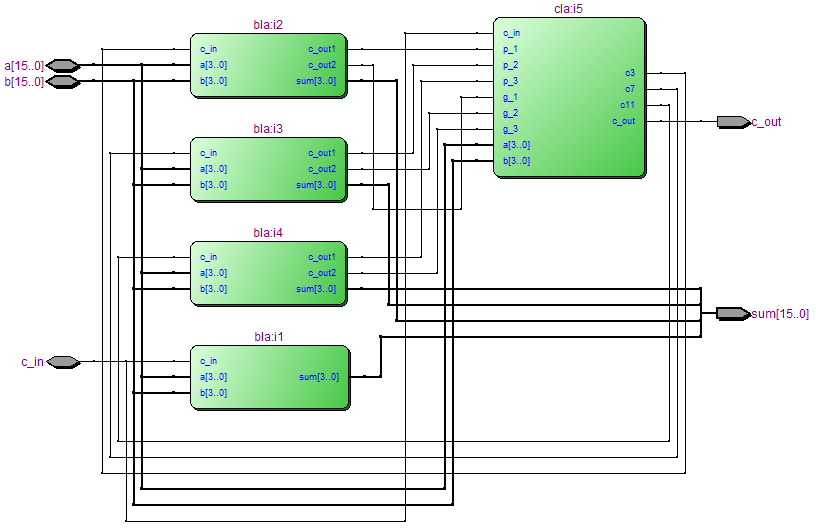
接下来进行分配引脚，具体如图9所示。之后便可以在FPGA开发板上进行逻辑综合。



**图9** 引脚分配图

**4.5** 对比分析

现在将组内并行，组间串行的16位超前进位加法器与组内组间都为并行的16位超前进位加法器进行对比。组内组间都为并行的16位超前进位加法器的RTL Viewer图如下所示。



**图10** 组内组间都为并行的16位超前进位加法器的RTL Viewer图

通过图9和图10的对比以及延迟的对比，可以发现组内组间都为并行的方法因为不牵扯到串联，拥有更快的速度，所以延迟要比本文的方法小一些。但是因为全部采用并行的电路结构更为复杂，所以本文的方法结构更为简单，占用面积更小，可以达到面积和速度的最优，在现实中拥有更广泛的市场。

**5.** 总结

本文对16位组内超前进位组间串行进位的超前进位加法器进行了设计，掌握了16位超前进位加法器的原理和设计方法，并通过和对该加法器进行功能仿真和逻辑综合，对整个设计和验证过程有了更深刻的理解。通过本次课程设计也明确了16位组内超前进位组间串行进位的超前进位加法器具有速度快，电路结构简单，占用面积小，带负载能力强的特点，而与16位组内组间都为超前进位的加法器进行对比后，得出本文的加法器在实际电路中运算速度和面积可以达到最优的结论。



参考文献

[1]胡乃平 曲英杰 周艳平编著. 计算机组成与结构 ---北京：清华大学出版社.2011.10

[2]曲英杰 方卓红编著.超大规模集成电路设计 ---北京：人民邮电出版社.2015.2

[3]王礼平，王观凤“超前进位加法器基本单元电路及其组合方案的优化计”中南民族大学学报(自然科学版)第 23卷第2期2004年6月

[4]孙岩.高性能算术逻辑部件研究与全定制设计.国防科技大学研究生论文集.2005.11.

[5]夏宇闻. 译.Verilog HDL 数字设计与综合[M].2 版.北京:电子工业出版社,2004年

[6]黄舒怀，蔡敏. 超前进位加法器的一种优化设计半导体技术第29卷，第8期. 2004.8

[7] 雷普红.高性能算术逻辑部件的设计与验证:硕士学位论文.长沙:国防科技大学，2006.3

附 录

**16位组内超前进位组间串行进位的超前进位加法器模块：**

module fulladder16(sum,c\_out,a,b,c\_in);

output [15:0] sum;

output c\_out;

input [15:0] a,b;

input c\_in;

wire c3,c7,c11,c15;

fulladder4 i1(sum[3:0],c3,a[3:0],b[3:0],c\_in);

fulladder4 i2(sum[7:4],c7,a[7:4],b[7:4],c3);

fulladder4 i3(sum[11:8],c11,a[11:8],b[11:8],c7);

fulladder4 i4(sum[15:12],c15,a[15:12],b[15:12],c11);

assign c\_out=c15;

endmodule

module fulladder4(sum,c\_out,a,b,c\_in);

output [3:0] sum;

output c\_out;

input [3:0] a,b;

input c\_in;

wire p0,g0,p1,g1,p2,g2,p3,g3;

wire c1,c2,c3,c4;

assign p0=a[0]^b[0],

p1=a[1]^b[1],

p2=a[2]^b[2],

p3=a[3]^b[3];

assign g0=a[0]&b[0],

g1=a[1]&b[1],

g2=a[2]&b[2],

g3=a[3]&b[3];

assign c1=g0|(p0&c\_in),

c2=g1|(p1&g0)|(p1&p0&c\_in),

c3=g2|(p2&g1)|(p2&p1&g0)|(p3&p2&p1&p0&c\_in),

c4=g3|(p3&g2)|(p3&p2&g1)|(p3&p2&p1&g0)|(p3&p2&p1&p0&c\_in);

assign sum[0]=p0^c\_in,

sum[1]=p1^c1,

sum[2]=p2^c2,

sum[3]=p3^c3;

assign c\_out=c4;

endmodule

**测试模块：**

module top;

reg[15:0] a,b;

reg c\_in;

wire[15:0] sum; wire c\_out;

fulladder16 adder(.sum(sum),.c\_out(c\_out),.a(a),.b(b),.c\_in(c\_in));

initial

begin

a=5;b=4;c\_in=0;

#20 a=2;b=6;c\_in=1;

#20 a=4'b1101;b=4'b0011;c\_in=0;

#20 a=8'b1101\_0011;b=8'b0110\_0001;c\_in=1;

#20 a=12'b0101\_1100\_0101;b=12'b1100\_0000\_0101;c\_in=0;

#20 a=16'b1011\_0001\_1100\_1010;b=16'b1100\_0000\_0011\_0000;c\_in=1;

end

endmodule