

**青 岛 科 技 大 学**

**本科综合课程设计报告**

**题 目16位原码除法器设计与FPGA实现**

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

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

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

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

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

1

10

2018

**\_\_\_\_\_\_年 \_\_\_月 \_\_\_日**

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 | 2018.9.27~2018.10.12 |
| 课程设计目的 | 通过本次课程设计，灵活运用所学知识，掌握不恢复余数加减交替法原码除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. 除法器简介………………………………………………………………………1

2. 恢复余数除法简介………………………………………………………………1

3. 加减交替除法简介………………………………………………………………3

4. 设计方案…………………………………………………………………………5

4.1 设计原理………………………………………………………………………5

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

4.2.1 16位寄存器模块…………………………………………………………6

4.2.2 16位加法器模块…………………………………………………………8

4.2.3 16位相反数求补器模块…………………………………………………10

4.2.4 移位器模块………………………………………………………………11

4.2.5 选择器模块………………………………………………………………13

4.2.6 恢复余数模块……………………………………………………………15

4.2.7 测试模块…………………………………………………………………17

4.3 功能仿真……………………………………………………………………18

4.4 逻辑综合……………………………………………………………………20

5. 总结………………………………………………………………………………22

参考文献……………………………………………………………………………23

附录…………………………………………………………………………………23

**摘 要**

除法作为四则运算中重要的组成之一，在算数运算中占据重要地位。而除法运算相比于其他运算又具有更加严苛的难度，因此对除法器的深入研究有助于深入理解其他运算的过程和原理。本文对传统的恢复余数除法器电路进行改进，采用一种不恢复余数的加减交替设计方法，设计实现16位定点原码除法器，不仅可以克服恢复余数效率低的缺点，也可以解决运算过程太过复杂等问题，从而在实际电路中使原码除法器的运算速度和面积达到更优。

**关键词：**除法器；定点原码；不恢复余数法；加减交替；运算速度

**1. 除法器简介**

除法是四则运算之一。已知两个因数的积与其中一个非零因数，求另一个因数的运算，叫做除法。

两个数相除又叫做两个数的比。若ab=c（b≠0）,用积数c和因数b来求另一个因数a的运算就是除法，写作c÷b，读作c除以b（或b除c）。其中，c叫做被除数，b叫做除数，运算的结果a叫做商。

如果除式的商数必须是整数，而除数和被除数并非因数关系的话，会出现相差的数值，其相差（以下的d）为余数。通常不定义除以零这种形式。

长除法俗称「长除」，适用于正式除法、小数除法、多项式除法（即因式分解）等较重视计算过程和商数的除法，过程中兼用了乘法和减法。根据乘法表，两个整数可以用长除法（直式除法）笔算。 如果被除数有分数部分（或者说是小数点），计算时将小数点带下来就可以；如果除数有小数点，将除数与被除数的小数点同时移位，直到除数没有小数点。算盘也可以做除法运算。

短除法俗称「短除」，适用于快速除法、多个整数同步除法（故此常用于求出最大公因数和最小公倍数）、二进位数字转换等较重视倍数测试和质因数（连乘式）的除法，过程大多只需用到九九乘法表及 9 以上少许整数的相乘因数。

带余除法就是带有余数的除法，被除数=除数×商+余数。带余除法主要包括整数的带余除法和多项式的带余除法。其中，整数的带余除法定理为：对于任意的a,b(设a≥b且b≠0)，存在唯一的商q和余数r 使得a=bq+r。多项式的带余除法则与之类似。

定点小数除法：|X|<|Y|，且Y不为0；

定点整数除法：|X|≥|Y|，且Y不为0。

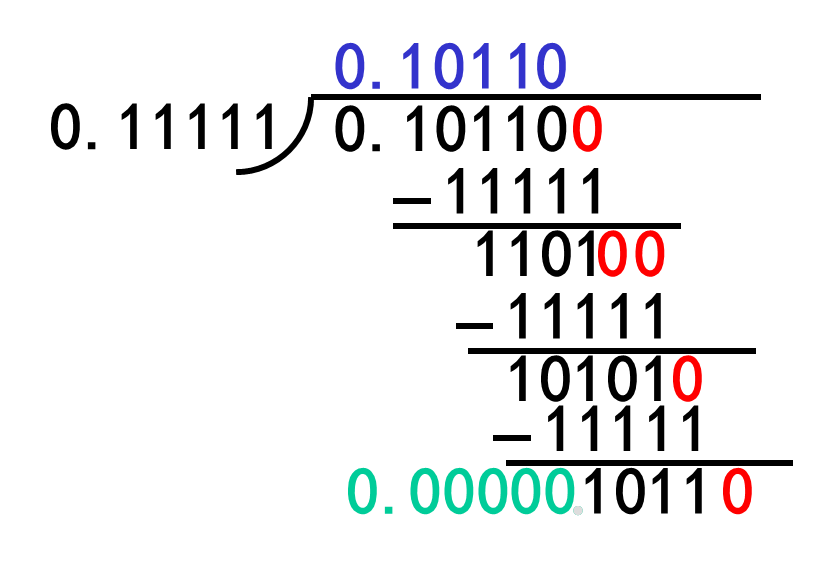
**2. 恢复余数除法简介**

定点原码一位除法器的原理是根据人工进行二进制除法的规则：判断被除数与除数的大小，若被除数小，则上商0，并在余数最低位补0，再用余数和右移一位的除数比，若够除，则上商1，否则上商0。然后继续重复上述步骤，直到除尽（即余数为0）或已得到的商的位数满足精度要求为止。另外，上商0还是1是计算者用观察比较的办法确定的，而计算机只能用做减法判断结果的符号为负还是为正来确定。当差为负时，上商为0，同时还应把除数再加到差上去，恢复余数为原来的正值之后再将其左移一位。若减得的差为0或为正值时，就没有恢复余数的操作，上商为1，余数左移一位。

设： X=X7X6X5X4X3X2X1X0，Y=Y7Y6Y5Y4Y3Y2Y1Y0，其中高四位X7X6X5X4和Y7Y6Y5Y4为符号位，低四位X3X2X1X0和Y3Y2Y1Y0为数据位。则：XY=K\*|X||Y|，其中，|X|和|Y|为X和Y的绝对值，K为X和Y的符号位的异或值。

|X||Y|利用恢复余数法求的，商根据余数的符号是正或负来判断。当为负时，上商为0，同时还应该把除数再加到差上去，恢复余数为原来的正值之后再左移一位。若差为0或为正值时，就没有恢复余数的操作，上商为1，余数左移一位。

设被除数X=0.1011，除数Y=-0.1101，除法人工计算过程：



商：0.10110，余数：0.10110×2-5。

人工进行二进制除法的规则：判断被除数与除数的大小，若被除数小，则上商0，并在余数最低位补0，再用余数和右移一位的除数比，若够除，则上商1，否则上商0。然后继续重复上述步骤，直到除尽(即余数为零)或已得到的商的位数满足精度要求为止。

右移除数，可以通过左移被除数(余数)来替代，左移出界的被除数(余数)的高位都是无用的零，对运算不会产生任何影响。另外，上商0还是1是计算者用观察比较的办法确定的，而在计算机中，只能用做减法判结果的符号为负还是为正来确定。

当差为负时上商为0，同时还应把除数再加到差上，恢复余数为原来的正值之后再将其左移一位。若差为正或为0时，没有恢复余数的操作，上商为1，余数左移一位。这种方法称为恢复余数法。

缺点：当某一次-Y的差值为负时，要多做一次+恢复余数的操作，降低了执行速度，又使控制线路变得复杂，因此在计算机中很少采用。

例：X=0.1011,Y=0.1101，用恢复余数法求X/Y。

解：[-Y]补=11.0011，取双符号位，-Y用+[-Y]补取代。



商：X/Y=0.1101，余数=0.0111×2-4。

计算机中普遍采用的是不恢复余数的除法方案，称之为加减交替法

**3. 加减交替除法简介**

加减交替法是对恢复余数除法的一种修正。当某一次求得的差值(余数Ri)为负时，不是恢复它，而是继续求下一位商，但用加上除数(+Y)的办法来取代(-Y)操作，其他操作依然不变。

加减交替法的规则如下：

（1）当余数为正时，商上1，求下一位商的办法，是余数左移一位，再减去除数；

（2）当余数为负时，商上0，求下一位商的办法，是余数左移一位，再加上除数。

此方法不用恢复余数，所以又叫不恢复余数法。但若最后一次上商为0，而又需得到正确余数，则在这最后一次仍需恢复余数。

例：X=0.1011, Y=0.1101，用加减交替法求X/Y。

解:[-Y]补=11.0011，取双符号位，-Y用+[-Y]补取代。



商：X/Y=0.1101，余数：0.0111×2-4。

**附：补码加减交替除法**

从［X］补与［Y］补直接求［X/Y］补。

补码除法规则比原码除法规则复杂。当除数和被除数用补码表示时，判别是否够减，要比较它们的绝对值的大小

若两数同符号，要用减法，若异号，则要用加法。

对于判断是否够减，及确定本次上商1还是0的规则，还与结果的符号有关。

当商为正时，商的每一位上的值与原码表示一致；当商为负时，商的各位应是补码形式的值，一般先按各位的反码值上商，除完后，再用在最低位上加1的办法求出正确的补码值。

在被除数的绝对值小于除数的绝对值(即商不溢出)的情况下，补码一位除法的运算规则如下：

(1) 如果被除数与除数同号，用被除数减去除数；

如果被除数与除数异号，用被除数加上除数。

如果所得余数与除数同号上商1，

如果所得余数与除数异号上商0，

该商即为结果的符号位。

(2) 求商的数值部分。

如果上次上商1，将余数左移一位后减去除数；

如果上次上商0，将余数左移一位后加上除数。

如果所得余数与除数同号上商1；

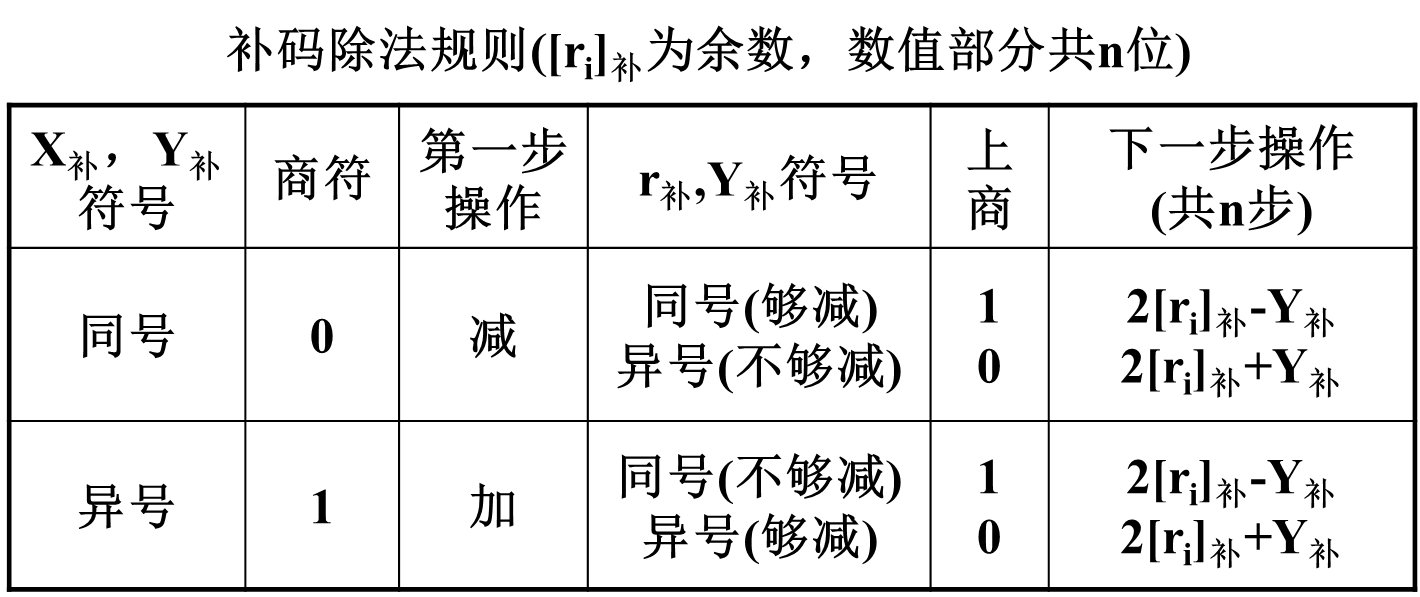
如果所得余数与除数异号上商0。

如此重复执行n-1次(设数值部分n位)。

(3) 商的最后一位一般采用恒置1的办法，并省略了最低位+1的操作，此时最大误差为±2-n。如果对商的精度要求较高，则可按规则(2)再进行一次操作并执行加[Y]补或加[-Y]补恢复余数，以求得商的第n位。

当除不尽时：若商为正，商的最低位不加1；若商为负，商的最低位加1，使商从反码值变成补码值。

当能除尽时：若除数为正，商的最低位不加1；若除数为负，商的最低位加1。



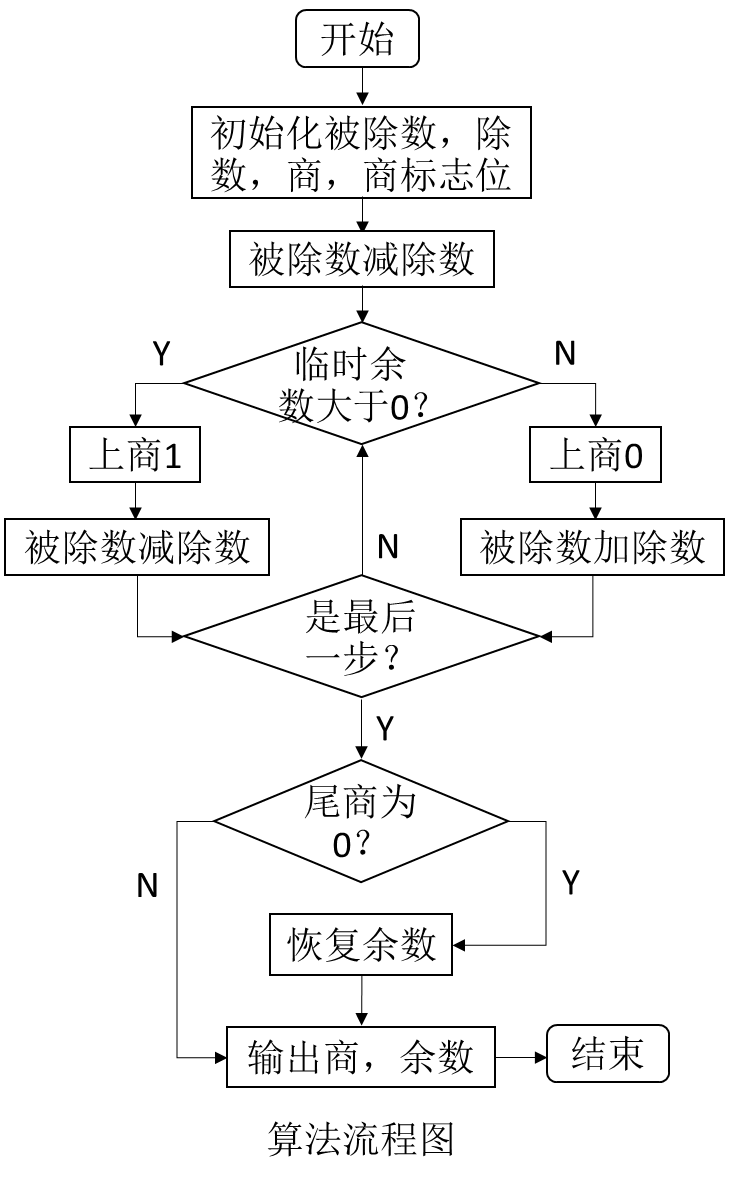
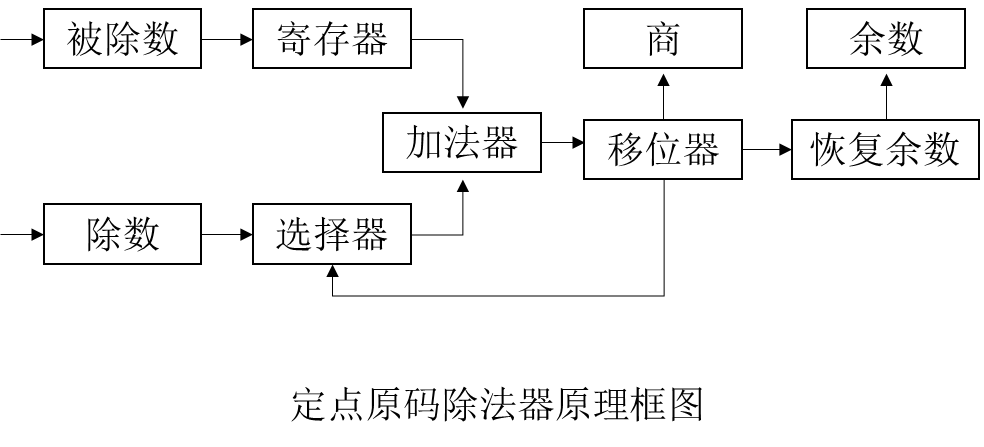
说明： (1) 表中i=0～n-1。

(2) 商一般采用末位恒置“1”的方法，操作简便。

**4. 设计方案**

**4.1** 设计原理

算法为不恢复余数的加减交替法，包含数据寄存器模块、加法器模块、相反数求补器模块、移位器模块、选择器模块、余数恢复器模块。



除法开始前，除数的原码放在输入寄存器中。除法开始后，首先用被除数减去除数，若运算结果大于0，商上1。若结果小于0，商上 0。然后被除数左移一位。若商的最后一位为0，需要恢复余数，否则直接输出结果，结束运算。

**4.2** 建立Verilog模型

**4.2.1** 16位寄存器模块

/\*输入寄存器（16+1位输入，16+2位输出）

\*/

module ireg(//output reg [1:0]sign,

output reg [17:0]iregout,

input [16:0]iregin,

input clk);

always @(posedge clk)

begin

iregout[16:0]<=iregin;

iregout[17]<=iregin[16];

//sign[0]<=iregin[16];

//sign[1]<=iregin[16];

end

endmodule

/\*商寄存器（16+1位输入，16+1位输出）

（输出商只需要1位符号位，即一共17位）

\*/

module oreg(output reg [16:0]oregout,

input [16:0]oregin,

input clk);

always @(posedge clk)

begin

oregout<=oregin;

end

endmodule

/\*余数寄存器

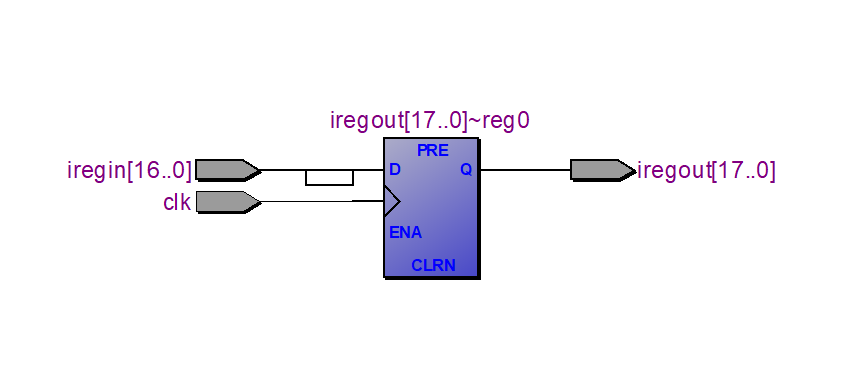
\*/

module remreg();

//用加法器替代

endmodule

主要用于输入的符号位扩展即寄存功能，寄存器模块电路结构：



寄存器模块测试：

`timescale 1ns/1ps

module IC;

wire [17:0]iregout;

wire [16:0]oregout;

reg [16:0]iregin;

reg [16:0]oregin;

reg clk;

initial

begin

clk=1'b0;

forever #10 clk=~clk;

end

initial

begin

iregin=17'b0\_1010\_1010\_1010\_1010;

oregin=17'b0\_1010\_1010\_1010\_1010;

#20 iregin=17'b0\_1010\_1000\_1010\_1000;

oregin=17'b0\_1010\_1000\_1010\_1000;

#20 iregin=17'b1\_0000\_0001\_0000\_0001;

oregin=17'b1\_0000\_0001\_0000\_0001;

#20 iregin=17'b1\_0000\_0101\_0000\_0101;

oregin=17'b1\_0000\_0101\_0000\_0101;

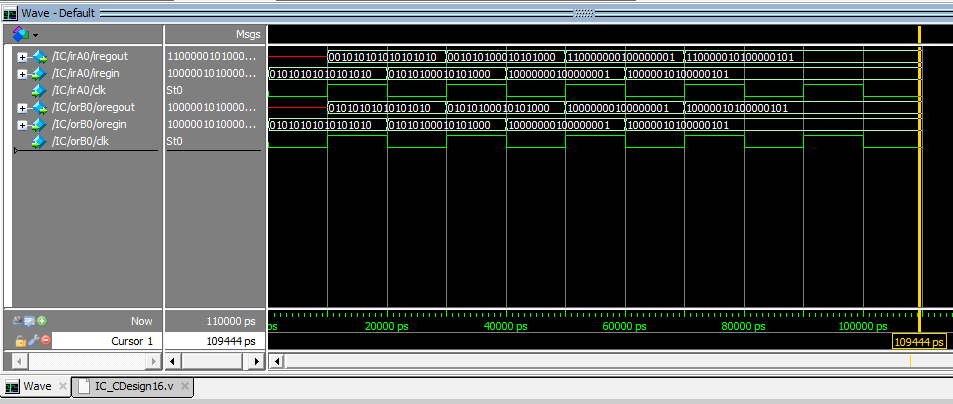
#50 $stop;

end

ireg irA0(iregout,iregin,clk);

oreg orB0(oregout,oregin,clk);

endmodule



**4.2.2** 16位加法器模块

module adder(output reg [17:0]addout,

output reg resone,

input [17:0]adda,

input [17:0]addb,

input clk);

always @(posedge clk)

begin

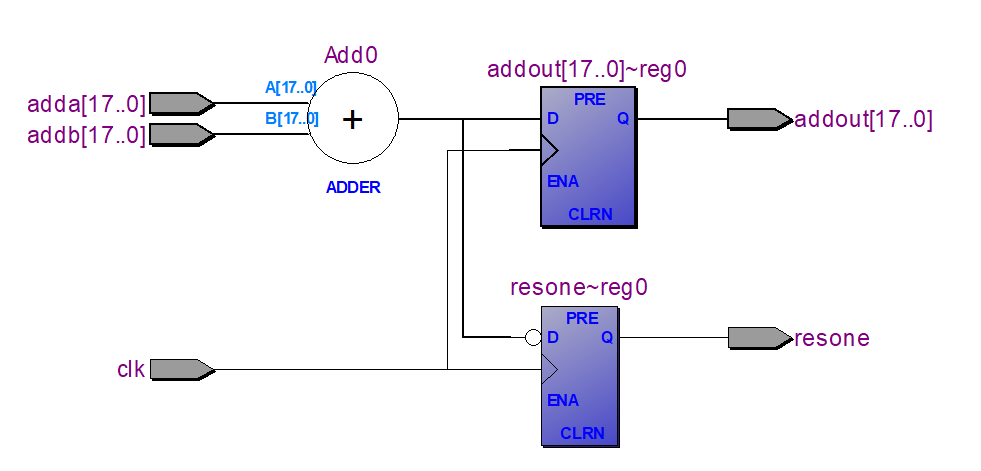
addout=adda+addb;

resone=~addout[17];

end

endmodule

加法器模块电路结构：



加法器模块测试：

`timescale 1ns/1ps

module IC;

wire [17:0]addout;

wire resone;

reg [17:0]adda;

reg [17:0]addb;

reg clk;

initial

begin

clk=1'b0;

forever #10 clk=~clk;

end

initial

begin

adda=18'b11\_0101\_0000\_0101\_0000;

addb=18'b00\_0000\_0101\_0000\_0101;

#20 adda=18'b00\_1010\_0000\_1010\_0000;

addb=18'b11\_0000\_1010\_0000\_1010;

#20 adda=18'b00\_0011\_0011\_0011\_0011;

addb=18'b00\_1100\_1100\_1100\_1100;

#20 adda=18'b11\_1001\_0000\_1001\_0000;

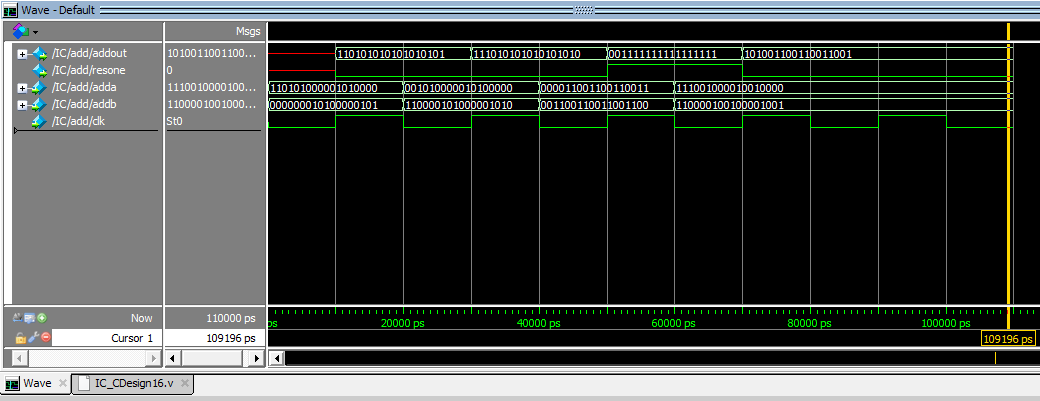
addb=18'b11\_0000\_1001\_0000\_1001;

#50 $stop;

end

adder add(addout,resone,adda,addb,clk);

endmodule



**4.2.3** 16位相反数求补器模块

module complement(output reg [16:0]comout,

input [16:0]comin,

input clk);

reg [15:0]comtmp;

always @(posedge clk)

begin

comtmp[0]=~comin[0];

comtmp[1]=~comin[1];

comtmp[2]=~comin[2];

comtmp[3]=~comin[3];

comtmp[4]=~comin[4];

comtmp[5]=~comin[5];

comtmp[6]=~comin[6];

comtmp[7]=~comin[7];

comtmp[8]=~comin[8];

comtmp[9]=~comin[9];

comtmp[10]=~comin[10];

comtmp[11]=~comin[11];

comtmp[12]=~comin[12];

comtmp[13]=~comin[13];

comtmp[14]=~comin[14];

comtmp[15]=~comin[15];

comtmp=comtmp+16'b0000\_0000\_0000\_0001;

comout=comtmp;

comout[16]=~comin[16];

end

endmodule

相反数求补器模块测试：

`timescale 1ns/1ps

module IC;

wire [16:0]comout;

reg [16:0]comin;

reg clk;

initial

begin

clk=1'b0;

forever #10 clk=~clk;

end

initial

begin

comin=17'b1\_0000\_0101\_0000\_0101;

#20 comin=17'b1\_1010\_0000\_1010\_0000;

#20 comin=17'b0\_0000\_0101\_0000\_0101;

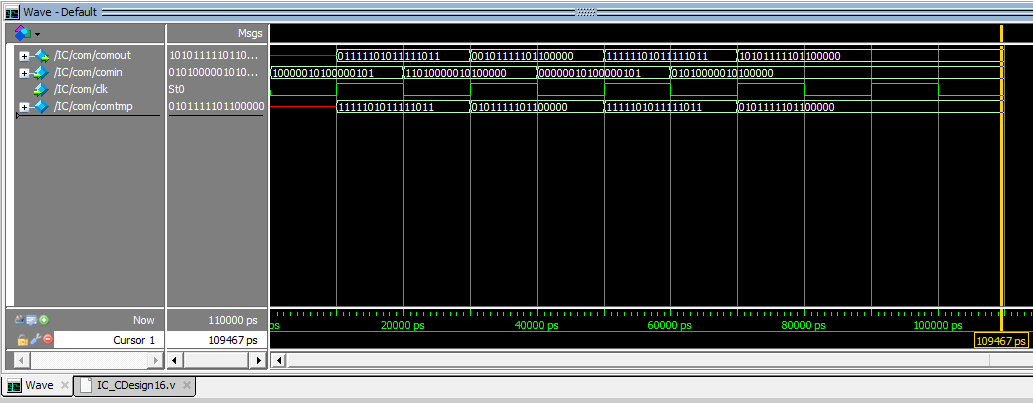
#20 comin=17'b0\_1010\_0000\_1010\_0000;

#50 $stop;

end

complement com(comout,comin,clk);

endmodule



**4.2.4** 移位器模块

module shift(output reg [17:0]remout,

output reg [16:0]resout,

input [17:0]remin,

input [16:0]resin,

input resone,

input clk);

reg [34:0]shitmp; //35=17+18=(16+1)+(16+2)

always @(posedge clk)

begin

shitmp[0]=resone;

shitmp[16:1]=resin[16:1];

shitmp[34:17]=remin[17:0];

shitmp=shitmp<<1;

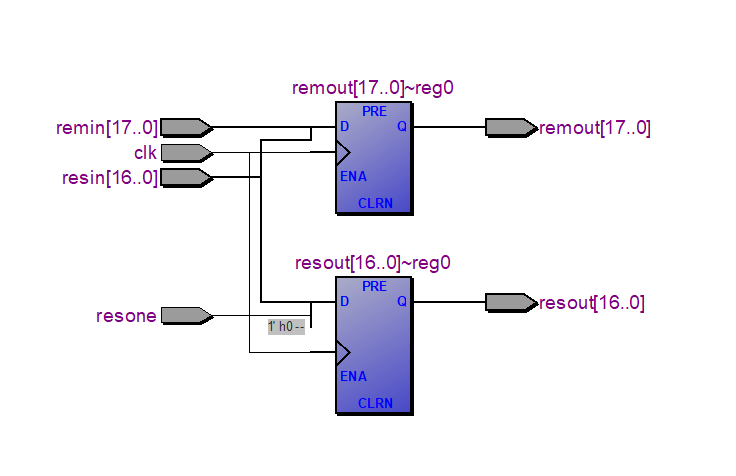
resout=shitmp[16:0];

remout[17:0]=shitmp[34:17];

end

endmodule

移位器模块电路结构：



移位器模块测试：

`timescale 1ns/1ps

module IC;

wire [17:0]remout;

wire [16:0]resout;

reg [17:0]remin;

reg [16:0]resin;

reg resone;

reg clk;

initial

begin

clk=1'b0;

forever #10 clk=~clk;

end

initial

begin

resone=1'b1;

remin=18'b11\_0000\_0101\_0000\_0101;

resin=17'b1\_0000\_0101\_0000\_0101;

#20 remin=18'b11\_1010\_0000\_1010\_0000;

resin=17'b1\_1010\_0000\_1010\_0000;

#20 remin=18'b00\_1100\_1100\_1100\_1100;

resin=17'b0\_1100\_1100\_1100\_1100;

#20 remin=18'b00\_1111\_0000\_1111\_0000;

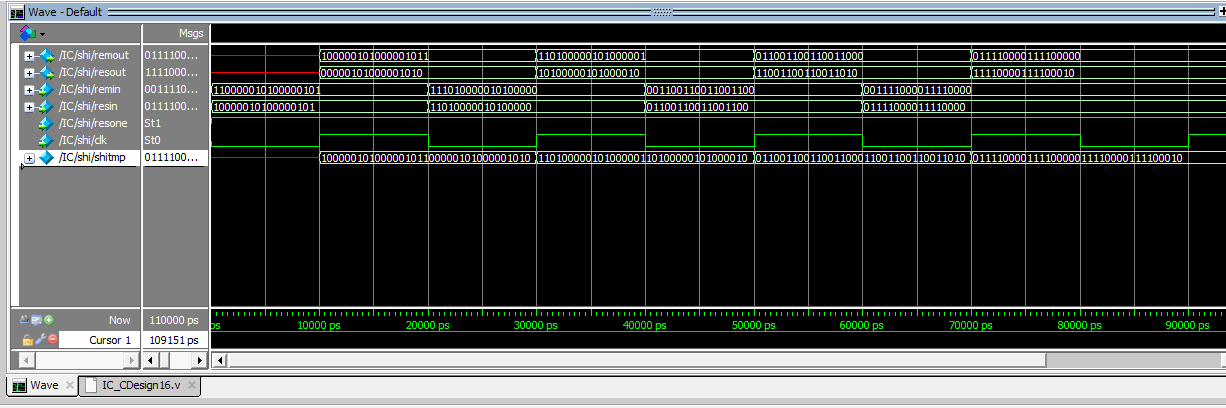
resin=17'b0\_1111\_0000\_1111\_0000;

#50 $stop;

end

shift shi(remout,resout,remin,resin,resone,clk);

endmodule



**4.2.5** 选择器模块

module chose(output reg [17:0]choout,

input [16:0]choin,

input resreg,

input clk);

reg [15:0]chotmp;

always @(posedge clk)

begin

if(!resreg)

begin

choout[16:0]=choin;

choout[17]=choin[16];

end

else if(resreg)

begin

chotmp[0]=~choin[0];

chotmp[1]=~choin[1];

chotmp[2]=~choin[2];

chotmp[3]=~choin[3];

chotmp[4]=~choin[4];

chotmp[5]=~choin[5];

chotmp[6]=~choin[6];

chotmp[7]=~choin[7];

chotmp[8]=~choin[8];

chotmp[9]=~choin[9];

chotmp[10]=~choin[10];

chotmp[11]=~choin[11];

chotmp[12]=~choin[12];

chotmp[13]=~choin[13];

chotmp[14]=~choin[14];

chotmp[15]=~choin[15];

chotmp=chotmp+16'b0000\_0000\_0000\_0001;

choout[15:0]=chotmp;

choout[16]=~choin[16];

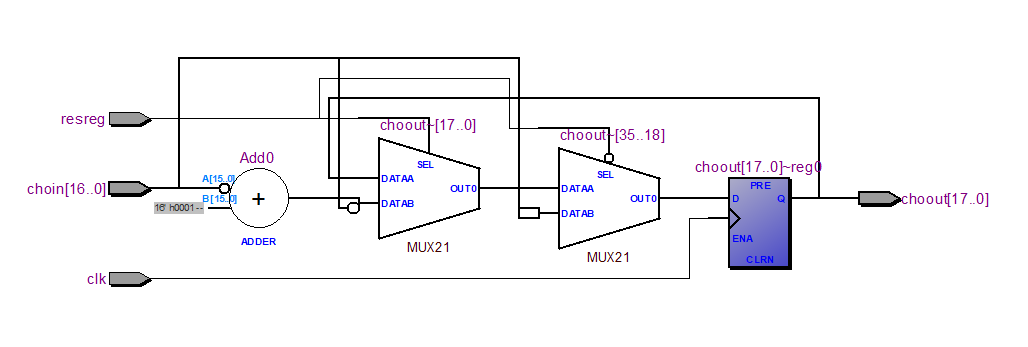
choout[17]=~choin[16];

end

end

endmodule

选择器模块电路结构：



选择器模块测试：

`timescale 1ns/1ps

module IC;

wire [17:0]choout;

reg [17:0]choin;

reg resreg;

reg clk;

initial

begin

clk=1'b0;

forever #10 clk=~clk;

end

initial

begin

resreg=1'b0;

choin=18'b11\_0000\_0101\_0000\_0101;

#20 resreg=1'b0;

choin=18'b11\_1010\_0000\_1010\_0000;

#20 resreg=1'b1;

choin=18'b00\_0000\_0101\_0000\_0101;

#20 resreg=1'b1;

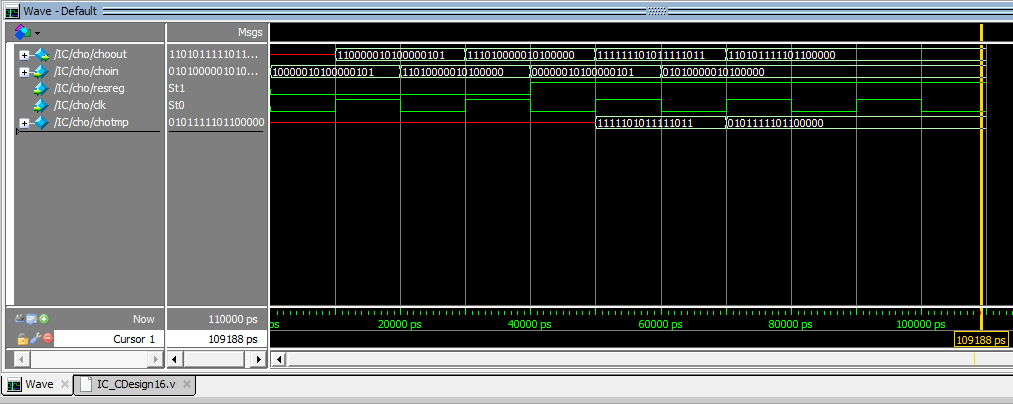
choin=18'b00\_1010\_0000\_1010\_0000;

#50 $stop;

end

chose cho(choout,choin,resreg,clk);

endmodule



**4.2.6** 恢复余数模块

module remrec(output reg [17:0]rrout,

input [17:0]rrina,

input [17:0]rrinb,

input rrreg,

input clk);

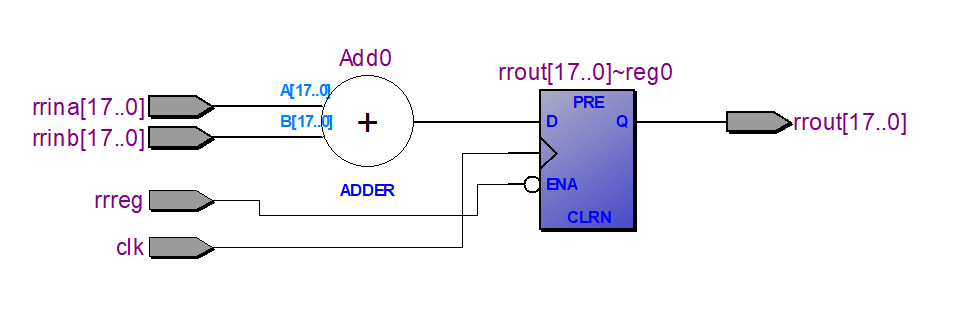
always @(posedge clk)

if(!rrreg)

rrout=rrina+rrinb;

endmodule

恢复余数模块电路结构：



恢复余数模块测试：

`timescale 1ns/1ps

module IC;

wire [17:0]rrout;

reg [17:0]rrina;

reg [17:0]rrinb;

reg rrreg;

reg clk;

initial

begin

clk=1'b0;

forever #10 clk=~clk;

end

initial

begin

rrreg=1'b0;

rrina=18'b11\_0000\_0101\_0000\_0101;

rrinb=18'b11\_1010\_0000\_1010\_0000;

#20 rrreg=1'b0;

rrina=18'b11\_1010\_0000\_1010\_0000;

rrinb=18'b00\_0000\_0101\_0000\_0101;

#20 rrreg=1'b1;

rrina=18'b00\_0000\_0101\_0000\_0101;

rrinb=18'b11\_1010\_0000\_1010\_0000;

#20 rrreg=1'b1;

rrina=18'b00\_1010\_0000\_1010\_0000;

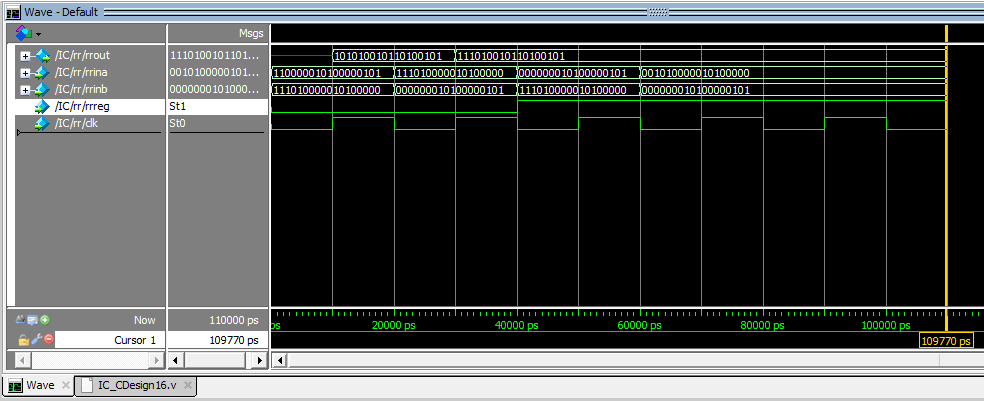
rrinb=18'b00\_0000\_0101\_0000\_0101;

#50 $stop;

end

remrec rr(rrout,rrina,rrinb,rrreg,clk);

endmodule



**4.2.7** 测试模块

`timescale 1 ns/ 1 ps

module IC;

reg clk;

wire [16:0]dividend;

wire [16:0]divisor;

wire [16:0]remainder;

wire [16:0]result;

IC\_CDesign i1(

.clk(clk),

.dividend(dividend),

.divisor(divisor),

.remainder(remainder),

.result(result));

always

#10 clk=~clk;

initial

begin

clk=0;

//dividend=17'b0\_1010\_0000\_1010\_0000;

//divisor=17'b0\_0001\_1010\_0001\_1010;

#1000 $stop;

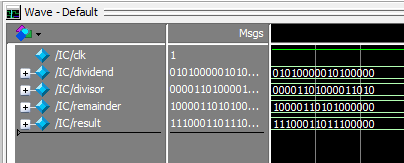
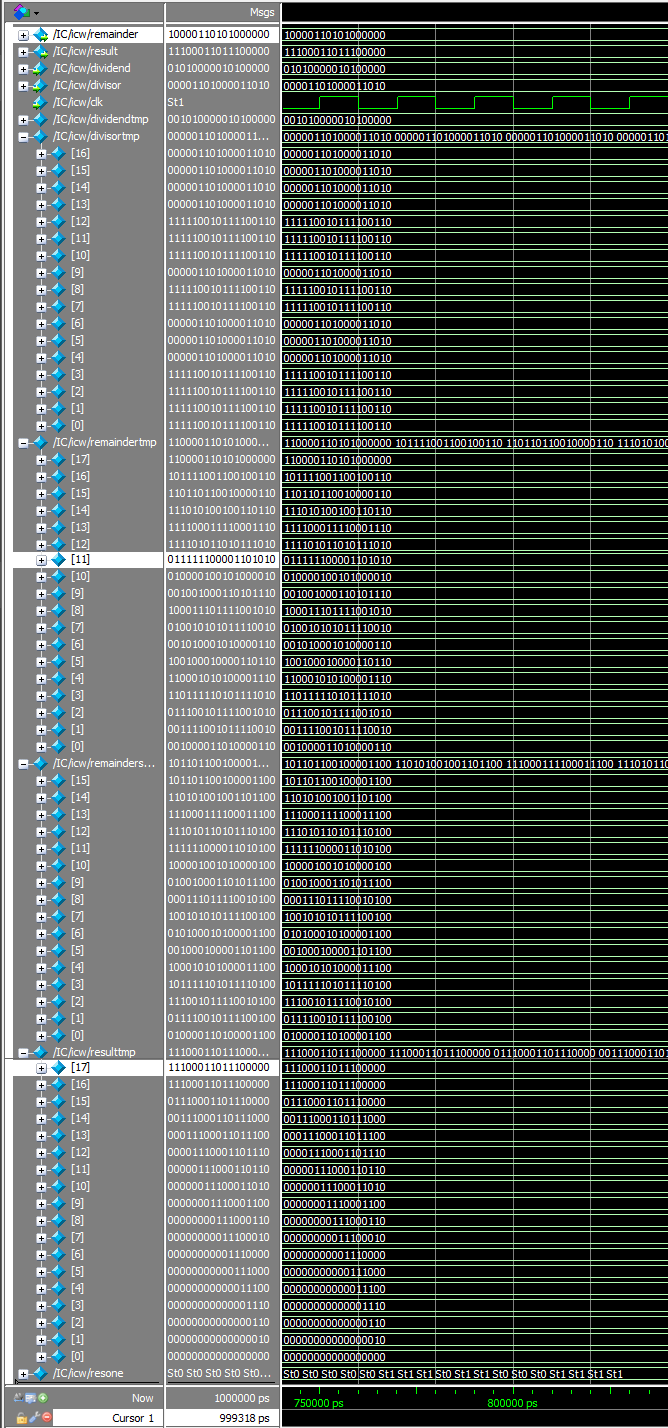
end

endmodule

**4.3** 功能仿真

功能仿真是验证所设计的Verilog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Modelsim仿真工具对不恢复余数加减交替的16位定点原码除法器的Verilog模型进行功能仿真。

具体仿真波形图如下图所示，图中形象的显示了被除数与除数、初始标志位、商和余数的数值波形。此外可以将过程各项展开，便可观察到整个除法器的运算流程及各模块的过程数值。



仿真波形图

通过上图可知各组数据的测试结果如下所示：

被除数：0\_1010\_0000\_1010\_0000，

除数：0\_0001\_1010\_0001\_1010，

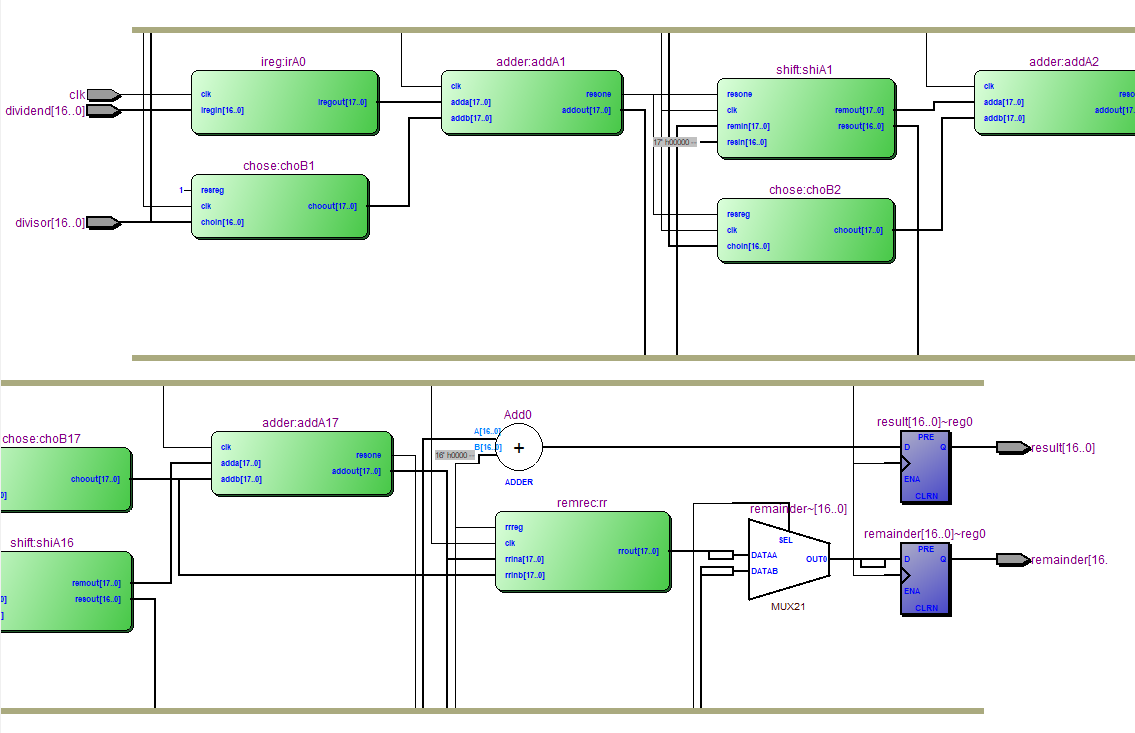
商：1\_1100\_0110\_1110\_0000，

余数：1\_0000\_1101\_0100\_0000。

**4.4** 逻辑综合

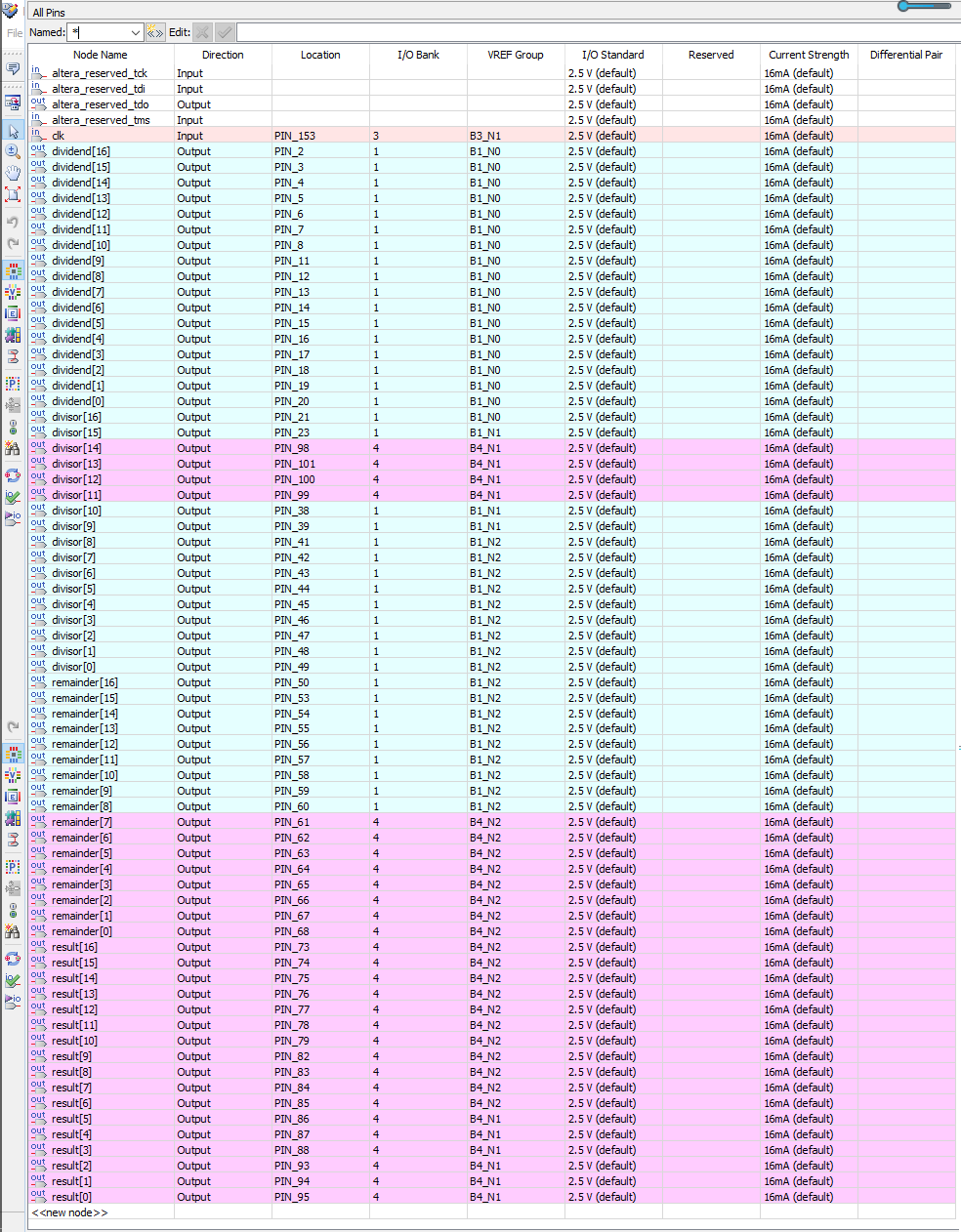
逻辑综合是将硬件描述语言描述的设计输入转换成与或非门、触发器、存储器等基本元件的连接关系，根据约束条件对生成的门级逻辑连接进行优化，生成为下一步实现所利用的网表文件。

经过优化后，16位定点原码除法器模块的RTL Viewer图如下图所示：



16位定点原码除法器模块RTL Viewer图

用QuartusII进行逻辑综合，进行分配引脚，具体如下图所示。之后便可以在FPGA开发板上进行逻辑综合。



引脚分配图

综合结果：

Flow Status Successful - Wed Oct 10 14:47:01 2018

Quartus II 64-Bit Version 13.0.0 Build 156 04/24/2013 SJ Full Version

Revision Name IC\_CDesign

Top-level Entity Name IC\_CDesign

Family Cyclone

Device EP1C12Q240C8

Timing Models Final

Total logic elements 1,369 / 12,060 ( 11 % )

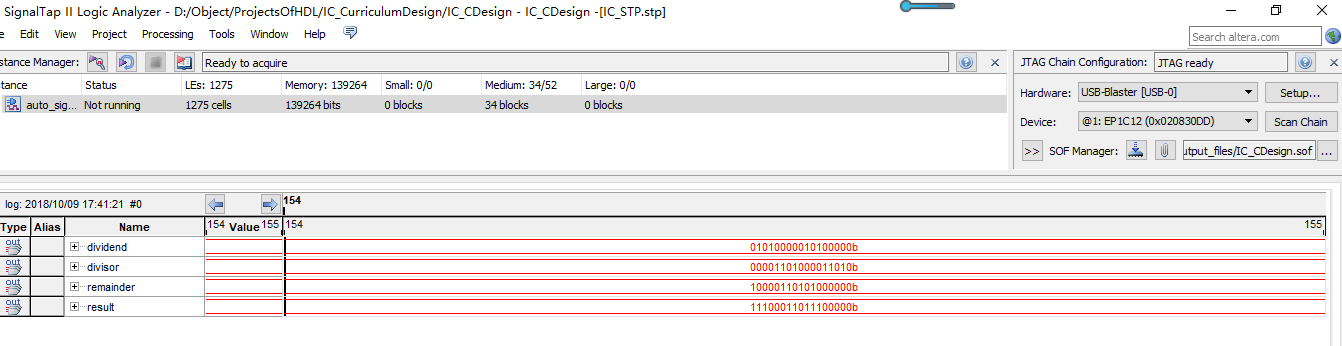
Total pins 69 / 173 ( 40 % )

Total virtual pins 0

Total memory bits 139,264 / 239,616 ( 58 % )

Total PLLs 0 / 2 ( 0 % )

下载到开发板，用QuartusII的SignalTapII进行硬件测试，传回的数据波形与用Modelsim仿真的数据一致，硬件测试正确。



**5. 总结**

本文对16位不恢复余数加减交替法定点原码除法器进行了设计，掌握了16位定点原码除法器的原理和设计方法，并通过Verilog和QuartusII对该除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位不恢复余数加减交替法定点原码除法器具有速度快，电路结构简单，占用面积小，带负载能力强的特点，而与16位恢复余数定点原码除法器相比，本文的除法器在实际电路中运算速度和面积可以达到更优。

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]袁春风编著. 计算机组成与系统结构 ---北京：清华大学出版社.2010.4

[4]李亚民编著. 计算机组成与系统结构 ---北京：清华大学出版社.2000年

[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

附 录

**16位不恢复余数加减交替定点原码除法器模块：**

/\*顶层模块

//divisor除数

//dividend被除数

//result商

//remainder余数

//sign符号

//clk时钟信号

32->16->8->4;33->17->9->5;

\*/

module IC\_CDesign(//output [1:0]sign, //两个1位符号位

output reg [16:0]remainder, //16是最高位

output reg [16:0]result,

output [16:0]dividend,

output [16:0]divisor,

input clk);

wire [16:0]dividendori=17'b0\_1010\_0000\_1010\_0000;

wire [16:0]divisorori=17'b0\_0001\_1010\_0001\_1010;

assign dividend=dividendori;

assign divisor=divisorori; wire [17:0]dividendtmp;

wire [17:0]divisortmp[16:0];

wire [17:0]remaindertmp[17:0];

wire [17:0]remaindershitmp[15:0];

wire [16:0]resulttmp[17:0];

assign resulttmp[0]=17'b0\_0000\_0000\_0000\_0000;

wire resone[17:0];

assign resone[0]=1'b1; //首次操作为减除数

ireg irA0(dividendtmp,dividendori,clk);

chose choB1(divisortmp[0],divisorori,resone[0],clk);

adder addA1(remaindertmp[0],resone[1],dividendtmp,divisortmp[0],clk);

shift shiA1(remaindershitmp[0],resulttmp[1],remaindertmp[0],resulttmp[0],resone[1],clk);

chose choB2(divisortmp[1],divisor,resone[1],clk);

adder addA2(remaindertmp[1],resone[2],remaindershitmp[0],divisortmp[1],clk);

shift shiA2(remaindershitmp[1],resulttmp[2],remaindertmp[1],resulttmp[1],resone[2],clk);

chose choB3(divisortmp[2],divisor,resone[2],clk);

adder addA3(remaindertmp[2],resone[3],remaindershitmp[1],divisortmp[2],clk);

shift shiA3(remaindershitmp[2],resulttmp[3],remaindertmp[2],resulttmp[2],resone[3],clk);

chose choB4(divisortmp[3],divisor,resone[3],clk);

adder addA4(remaindertmp[3],resone[4],remaindershitmp[2],divisortmp[3],clk);

shift shiA4(remaindershitmp[3],resulttmp[4],remaindertmp[3],resulttmp[3],resone[4],clk);

chose choB5(divisortmp[4],divisor,resone[4],clk);

adder addA5(remaindertmp[4],resone[5],remaindershitmp[3],divisortmp[4],clk);

shift shiA5(remaindershitmp[4],resulttmp[5],remaindertmp[4],resulttmp[4],resone[5],clk);

chose choB6(divisortmp[5],divisor,resone[5],clk);

adder addA6(remaindertmp[5],resone[6],remaindershitmp[4],divisortmp[5],clk);

shift shiA6(remaindershitmp[5],resulttmp[6],remaindertmp[5],resulttmp[5],resone[6],clk);

chose choB7(divisortmp[6],divisor,resone[6],clk);

adder addA7(remaindertmp[6],resone[7],remaindershitmp[5],divisortmp[6],clk);

shift shiA7(remaindershitmp[6],resulttmp[7],remaindertmp[6],resulttmp[6],resone[7],clk);

chose choB8(divisortmp[7],divisor,resone[7],clk);

adder addA8(remaindertmp[7],resone[8],remaindershitmp[6],divisortmp[7],clk);

shift shiA8(remaindershitmp[7],resulttmp[8],remaindertmp[7],resulttmp[7],resone[8],clk);

chose choB9(divisortmp[8],divisor,resone[8],clk);

adder addA9(remaindertmp[8],resone[9],remaindershitmp[7],divisortmp[8],clk);

shift shiA9(remaindershitmp[8],resulttmp[9],remaindertmp[8],resulttmp[8],resone[9],clk);

chose choB10(divisortmp[9],divisor,resone[9],clk);

adder addA10(remaindertmp[9],resone[10],remaindershitmp[8],divisortmp[9],clk);

shift shiA10(remaindershitmp[9],resulttmp[10],remaindertmp[9],resulttmp[9],resone[10],clk);

chose choB11(divisortmp[10],divisor,resone[10],clk);

adder addA11(remaindertmp[10],resone[11],remaindershitmp[9],divisortmp[10],clk);

shift shiA11(remaindershitmp[10],resulttmp[11],remaindertmp[10],resulttmp[10],resone[11],clk);

chose choB12(divisortmp[11],divisor,resone[11],clk);

adder addA12(remaindertmp[11],resone[12],remaindershitmp[10],divisortmp[11],clk);

shift shiA12(remaindershitmp[11],resulttmp[12],remaindertmp[11],resulttmp[11],resone[12],clk);

chose choB13(divisortmp[12],divisor,resone[12],clk);

adder addA13(remaindertmp[12],resone[13],remaindershitmp[11],divisortmp[12],clk);

shift shiA13(remaindershitmp[12],resulttmp[13],remaindertmp[12],resulttmp[12],resone[13],clk);

chose choB14(divisortmp[13],divisor,resone[13],clk);

adder addA14(remaindertmp[13],resone[14],remaindershitmp[12],divisortmp[13],clk);

shift shiA14(remaindershitmp[13],resulttmp[14],remaindertmp[13],resulttmp[13],resone[14],clk);

chose choB15(divisortmp[14],divisor,resone[14],clk);

adder addA15(remaindertmp[14],resone[15],remaindershitmp[13],divisortmp[14],clk);

shift shiA15(remaindershitmp[14],resulttmp[15],remaindertmp[14],resulttmp[14],resone[15],clk);

chose choB16(divisortmp[15],divisor,resone[15],clk);

adder addA16(remaindertmp[15],resone[16],remaindershitmp[14],divisortmp[15],clk);

shift shiA16(remaindershitmp[15],resulttmp[16],remaindertmp[15],resulttmp[15],resone[16],clk);

chose choB17(divisortmp[16],divisor,resone[16],clk);

adder addA17(remaindertmp[16],resone[17],remaindershitmp[15],divisortmp[16],clk);

assign resulttmp[17]=resulttmp[16]+resone[17];

remrec rr(remaindertmp[17],remaindertmp[16],divisortmp[16],resone[17],clk);

always @(posedge clk)

begin

result=resulttmp[17];

if(resone[17]) //如果最后一位商为1，则不恢复余数，否则恢复余数

remainder=remaindertmp[16];

else if(!resone[17])

remainder=remaindertmp[17];

end

endmodule

**测试模块：**

/\*激励

\*/

`timescale 1 ns/ 1 ps

module IC;

reg clk;

wire [16:0]dividend;

wire [16:0]divisor;

wire [16:0]remainder;

wire [16:0]result;

IC\_CDesign i1(

.clk(clk),

.dividend(dividend),

.divisor(divisor),

.remainder(remainder),

.result(result));

always

#10 clk=~clk;

initial

begin

clk=0;

//dividend=17'b0\_1010\_0000\_1010\_0000;

//divisor=17'b0\_0001\_1010\_0001\_1010;

#1000 $stop;

end

endmodule