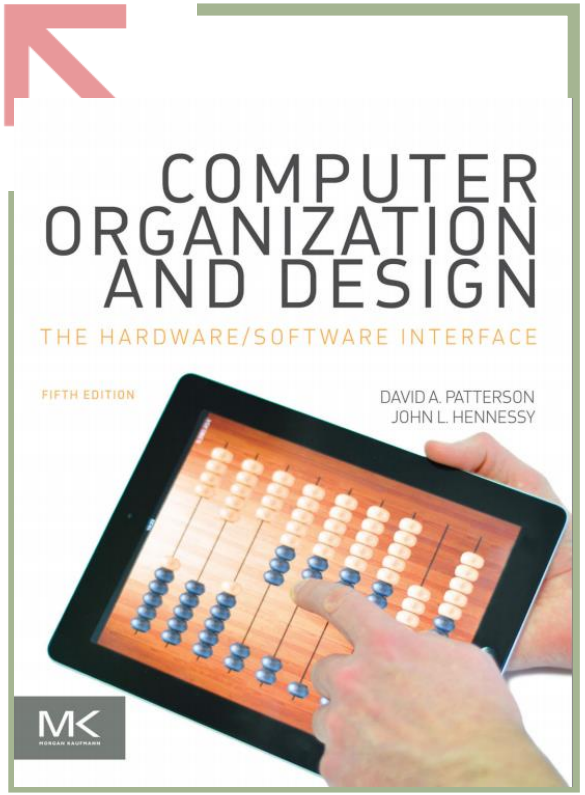
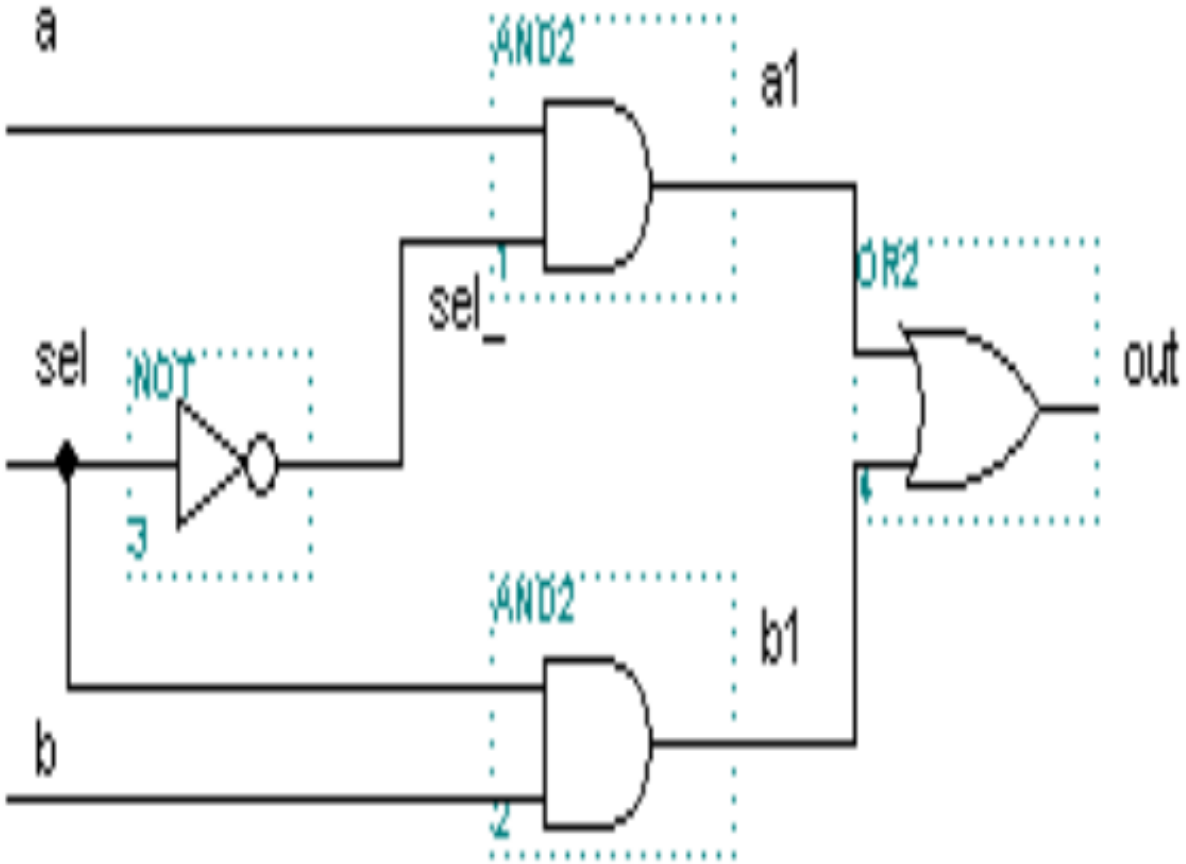
IP核的使用与ALU的设计



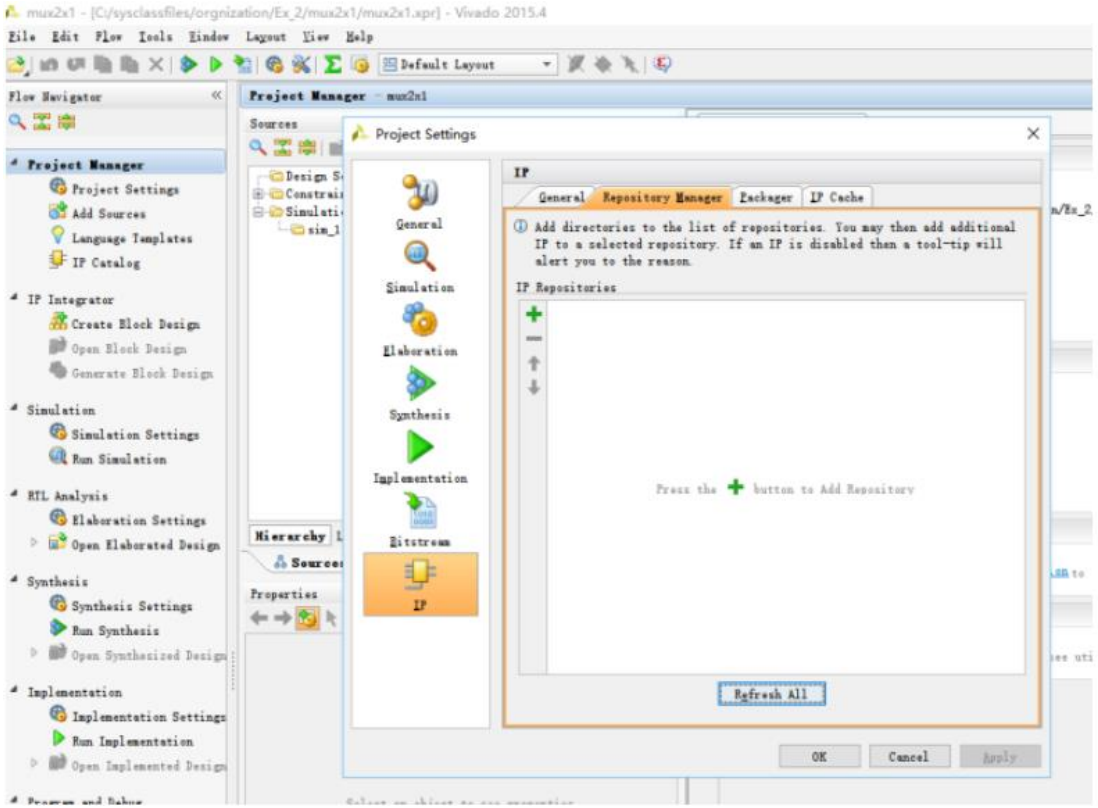
课堂练习1(2选1多路选择器)



1. 基于门电路IP核，采用block design或structure design的方式实现端口位宽可配置的2选1多路选择器
2. 创建testbench完成功能验证
3. 进行管脚约束文件配置、综合、实现、生成比特流文件，下载到板上
4. 打包封装城IP核

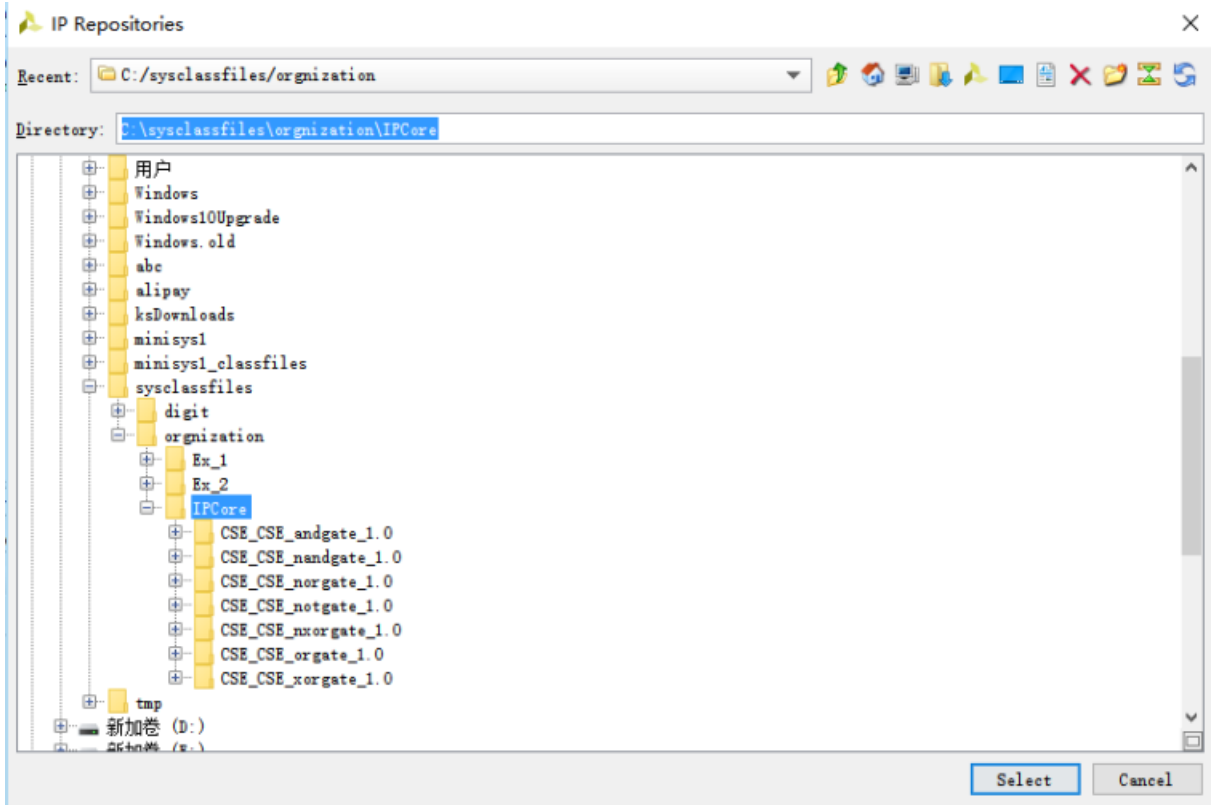
导入IP核

在 C:/sysclassfiles/organization/Ex_2 文件夹中创建一个新的项目 mux2x1。在界面左侧的Project Manager 中点击 Project Settings，打开 Project Settings 对话框，并转到 IP 项上。



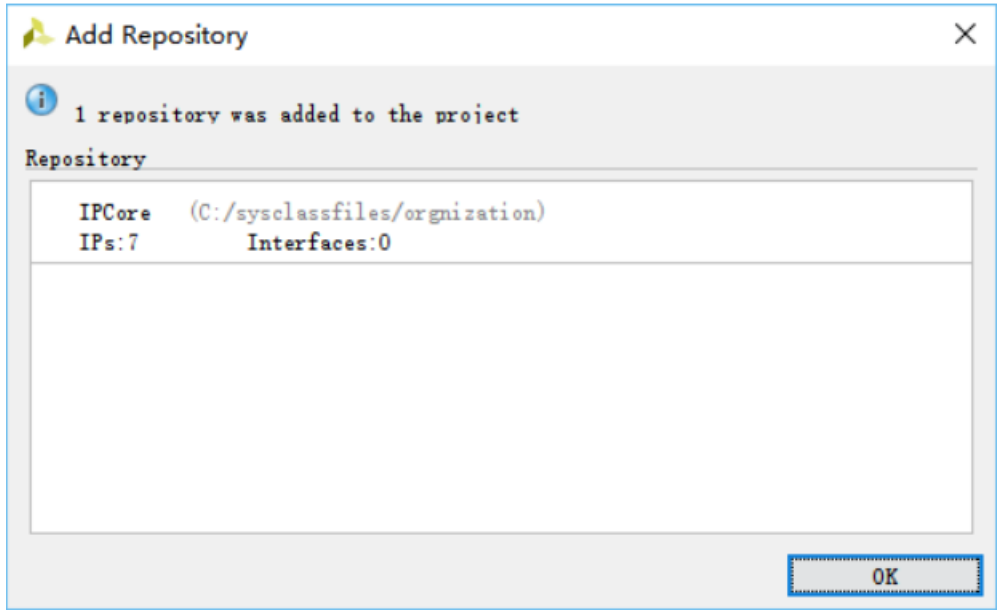
导入IP核

点击+号，找到下面的目录：C:\sysclassfiles\orgnization\IPCore。如下图所示，选择 IPCore的文件夹，点击 Select。



导入IP核

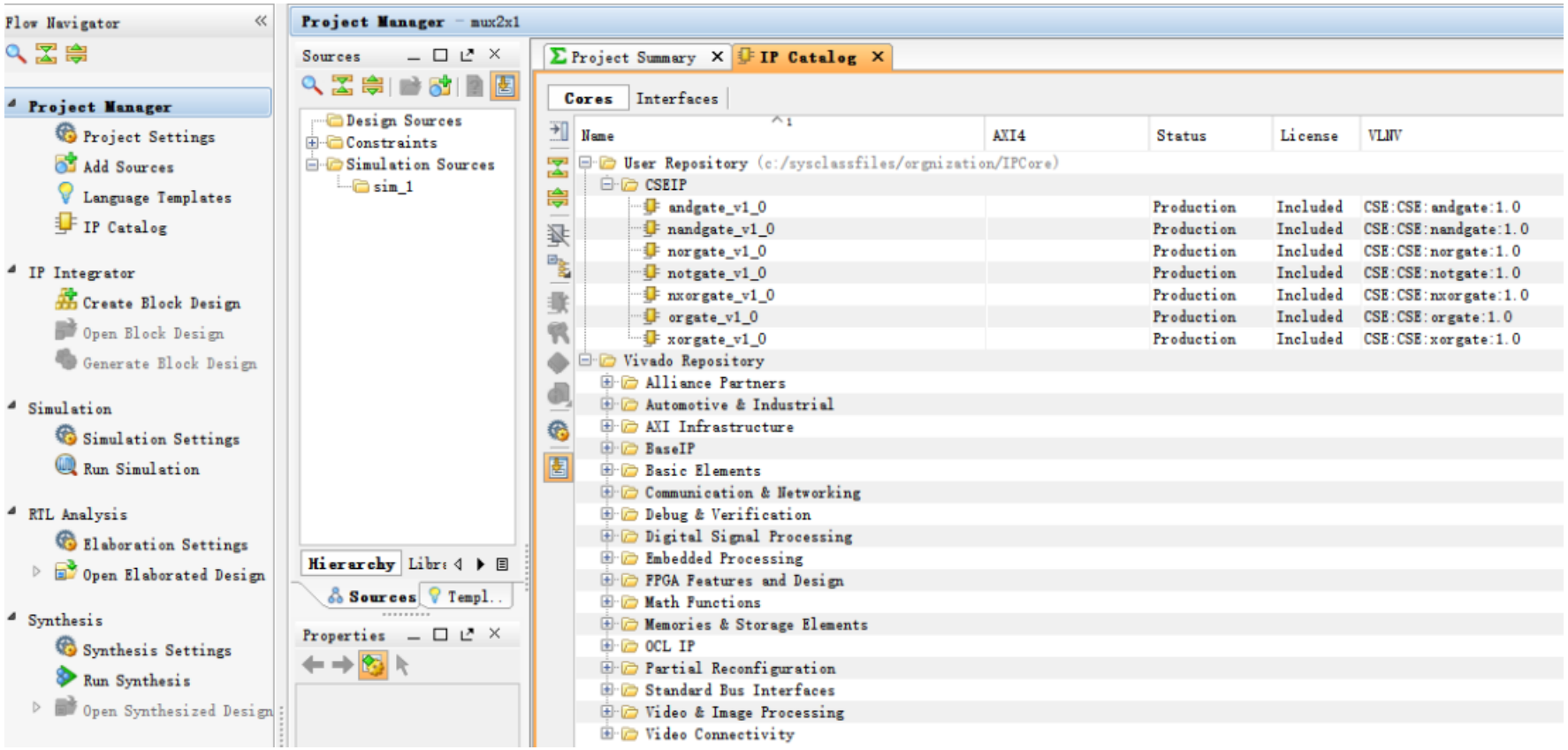
系统弹出如下图所示的 Add Repository 对话框，检查一下， 确实是 7 个 IP 核，点击OK。



现在回到了 Project Settings 对话框。点击 Apply， 然后点击 OK。

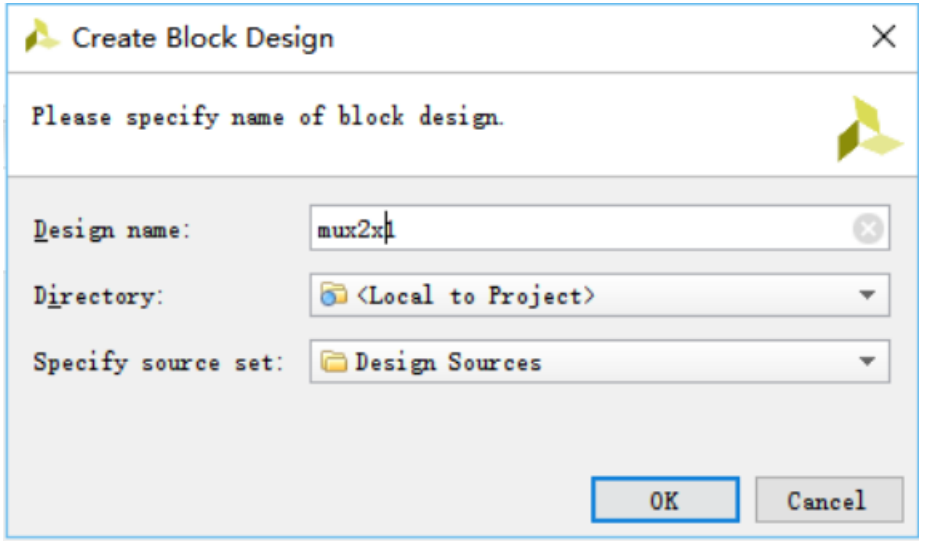
导入IP核

现在点击 Project Manager 下的 IP Catalog， 我们就会看到如下图所示的界面中， IP Catalog 中已经有了我们的 7 个 IP 核。



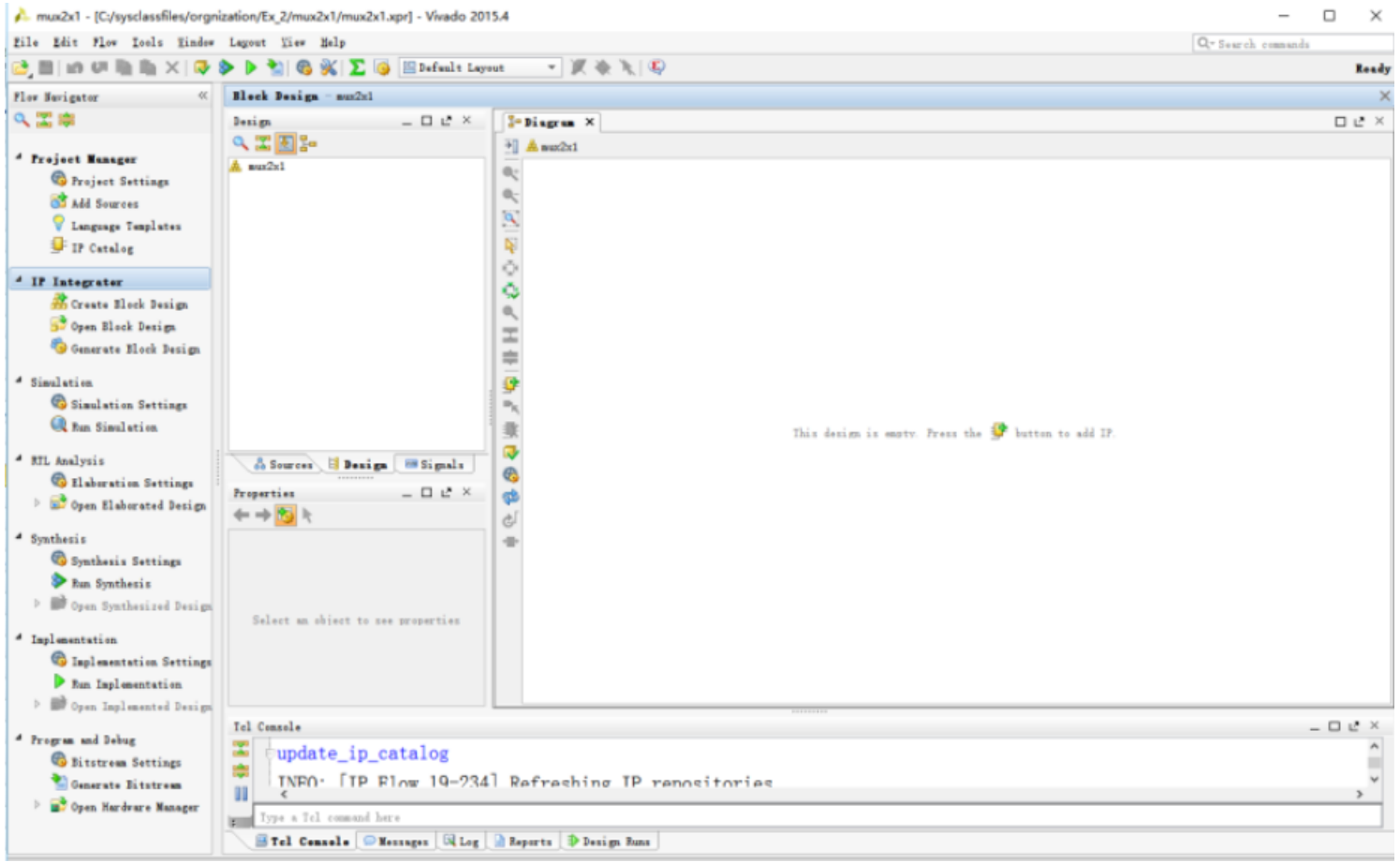
创建bd设计文件

点击 Project Manager 下的 Create Block Design，打开下图所示的对话框，按照下图设置后点击 OK.



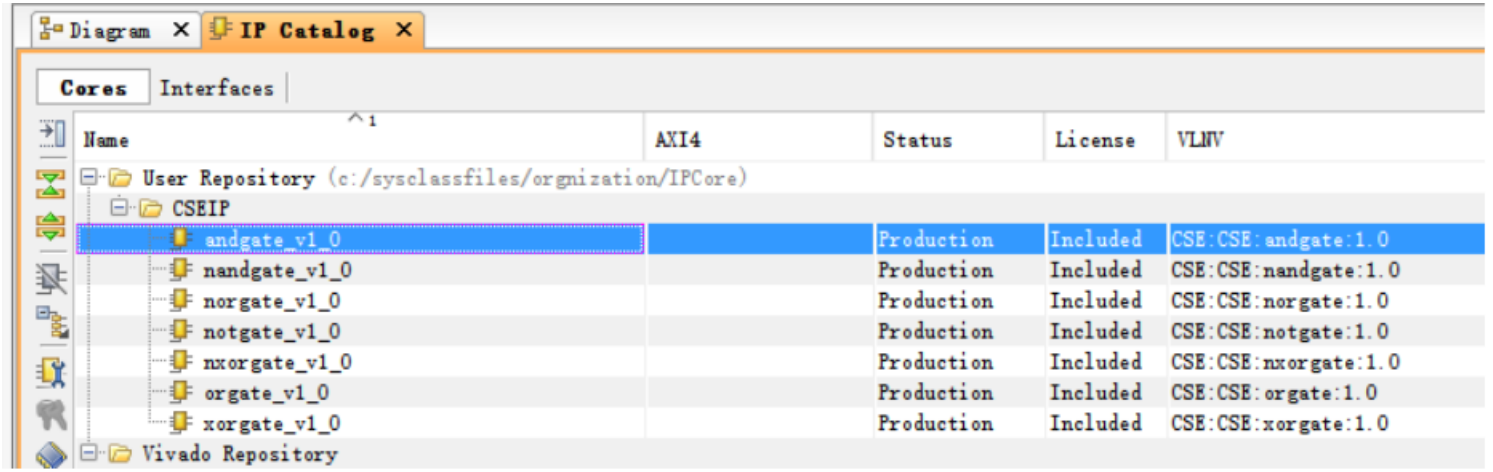
创建bd设计文件

现在得到了下图所示的界面



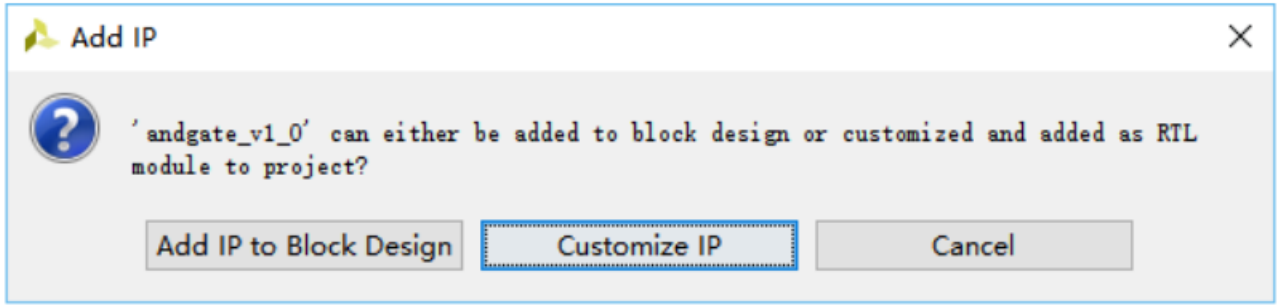
放置基本门电路

现在点击 Project Manager 下的 IP Catalog，在如下图所示的窗口中双击 andgate_v1_0



放置基本门电路

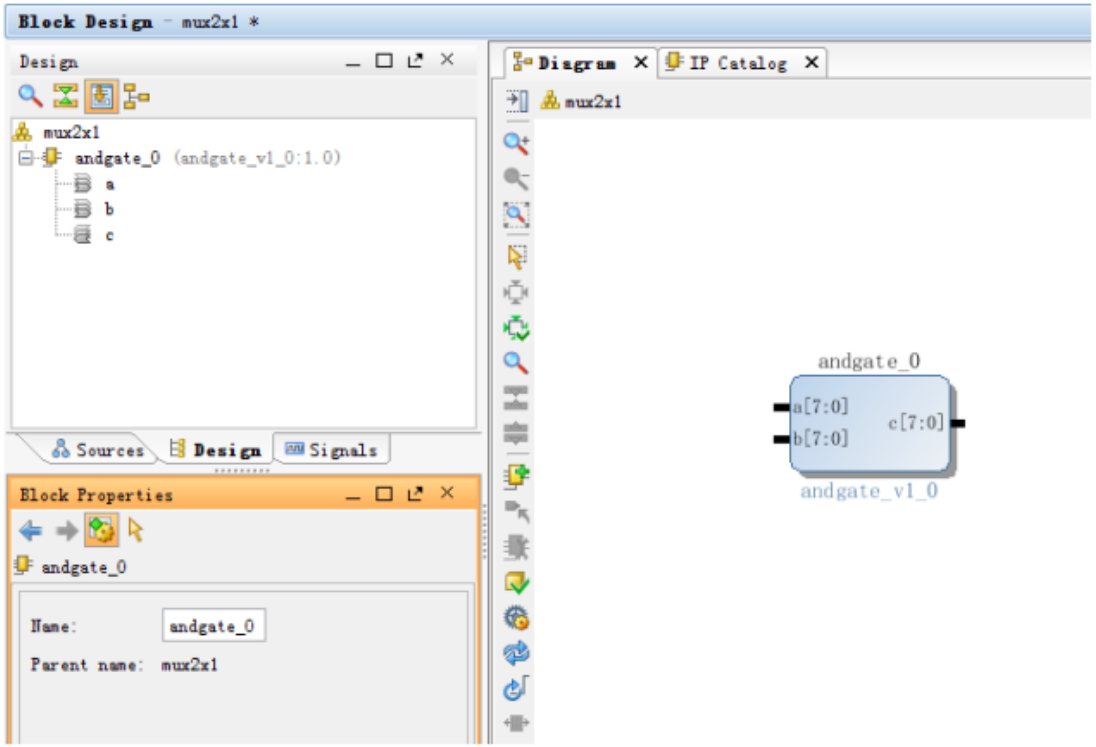
在出现的如下图所示的对话框中选择 **Add IP to Block Design**。



此时界面上出现了所选择的 IP 核心

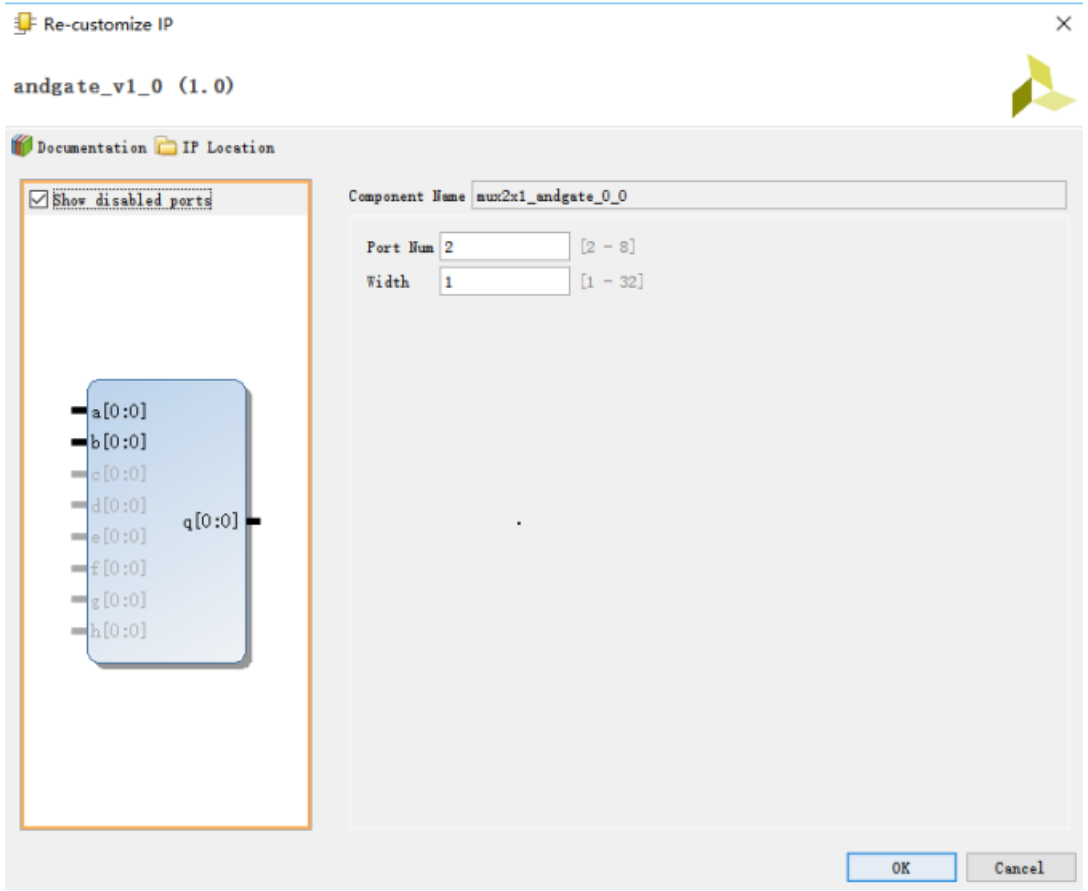
放置基本门电路

左键点击图中的 andgate_0 器件，选中它，然后右键点击它，并在弹出的菜单中选择 Customize Block... （或者双击 andgate_0 器件）



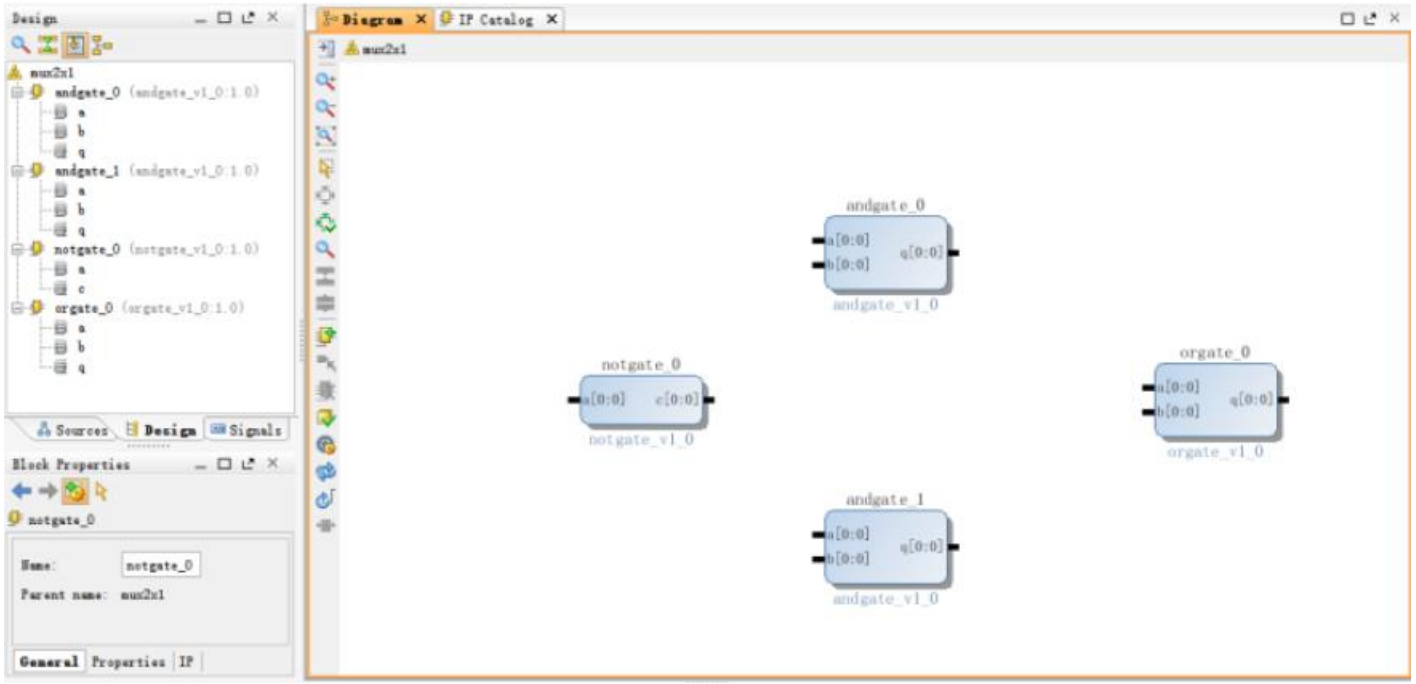
放置基本门电路

按照图中设置 Port_Num 为 2， Width 为 1，然后点击 OK。



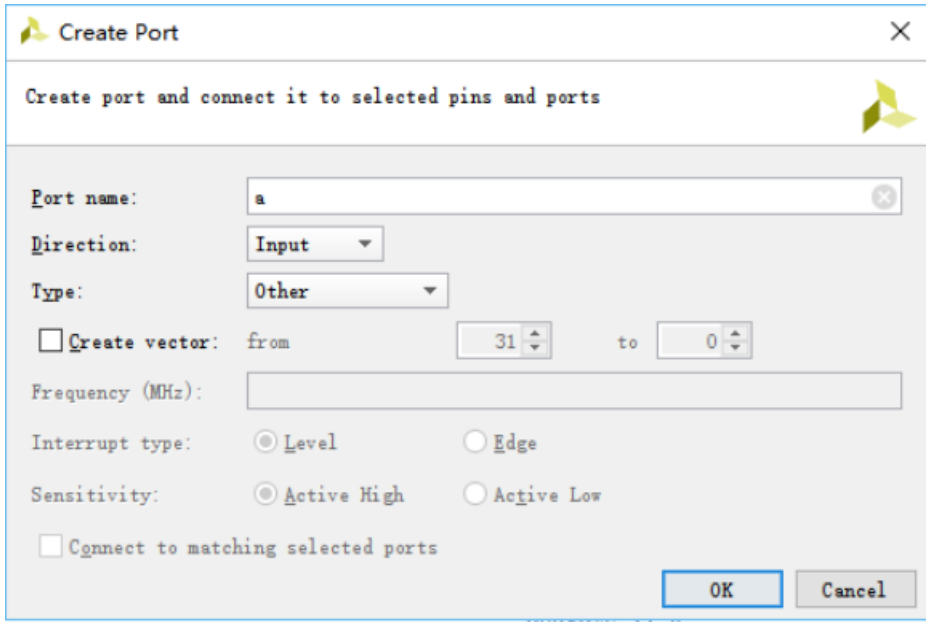
放置基本门电路

这样我们就放置好了一个与门，按照这个方法再放置另一个与门，一个或门和一个非门，并设置好他们的数据宽度都是 1，除了非门，其他几个门的数据端口数都为 2。放置好后，将他们的位置移动到如图所示。



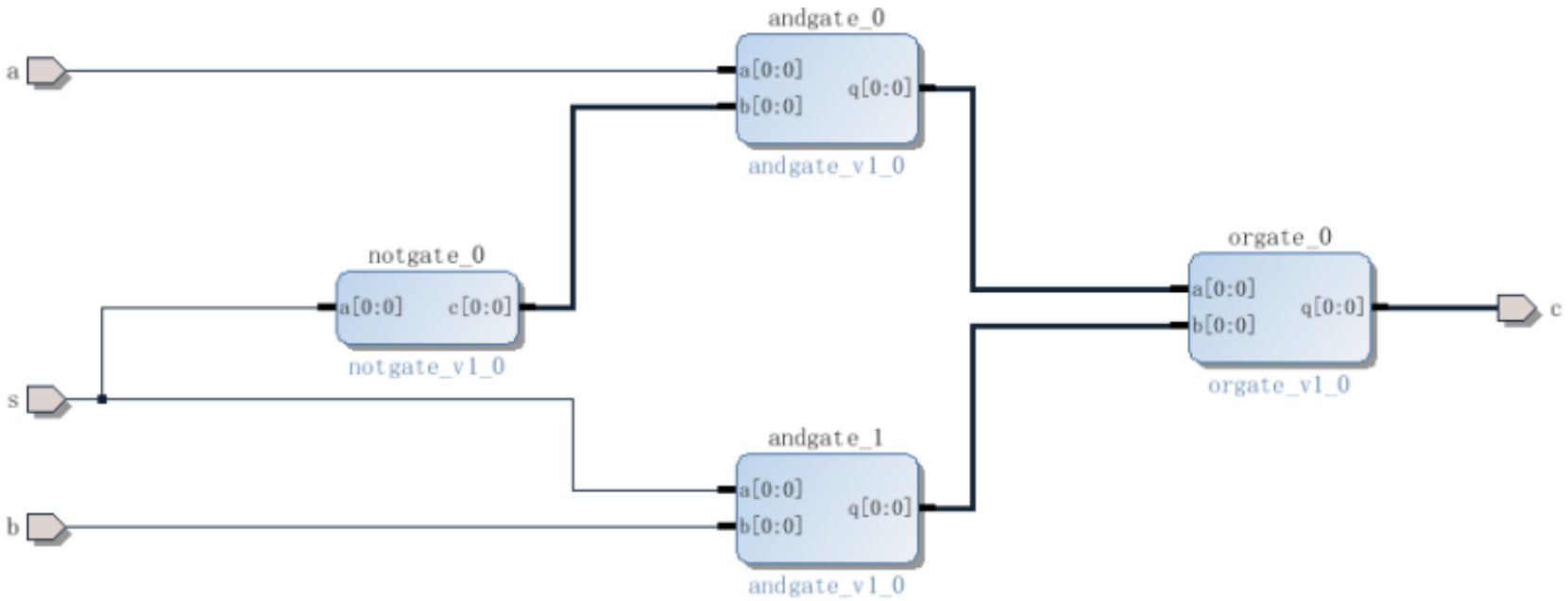
放置输入输出端口

在Diagram空白处右键点击，在弹出的菜单中选择 Create Port...。 打开如所示的对话框，按照图示设置，就添加立刻输入端 a。



放置输入输出端口

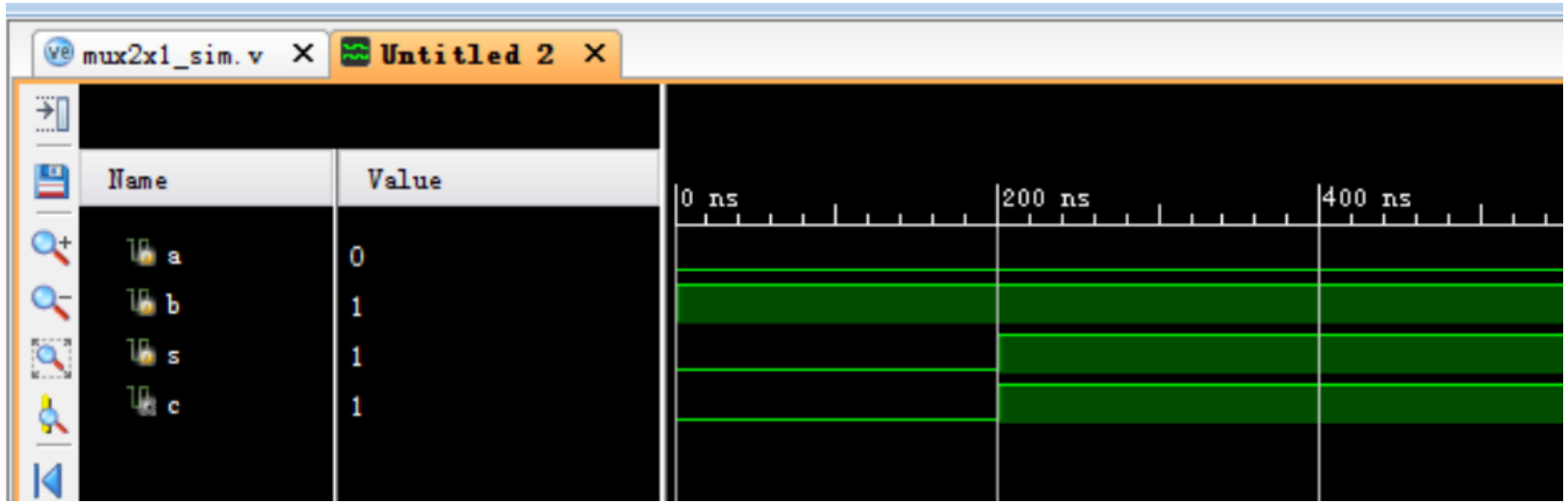
按照上述方法在增加输入端 b，输入端 s 和输出端 c。并按照图示连接好电路



仿真验证

```
`timescale 1ns / 1ps
module mux2x1_sim( );
    // input
    reg a=0;
    reg b=1;
    reg s=0;
    //output
    wire c;
    mux2x1 u(.a(a),.b(b),.s(s),.c(c));
    initial begin
        # 200 s=1;
    end
endmodule
```

仿真验证



生成输出文件

在 Project Manager 的 Source 界面中，右键点击 mux2x1，在弹出的菜单中选择 Generate Output Products...，之后再弹出的 Generate Output Products 对话框中选择 Global，点击 Generate。

生成 HDL 文件

再次在 Project Manager的 Source界面中，右键点击 mux2x1，在弹出的菜单中选择 Create HDL Wrapper...，之后再弹出的 Create HDL Wrapper 对话框中选择 Let Vivado Manage wrapper and auto-update，点击 OK。

进行管脚约束文件配置、综合、实现、生成比特流文件，下载到板上。

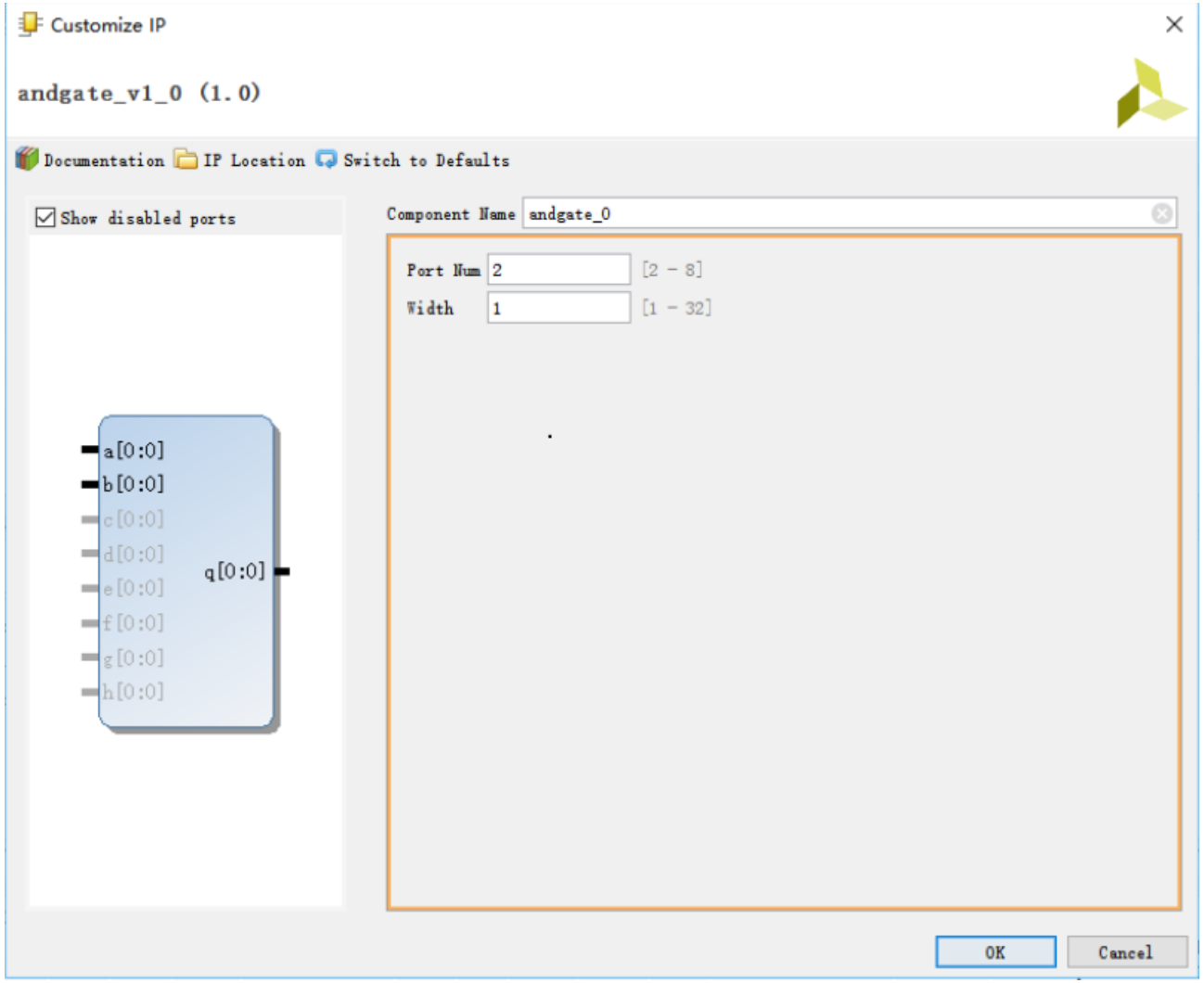
信号	器件	管脚
a	SW0	W4
b	SW1	R4
s	SW23	Y9
c[0]	G LD0	A21

用 Verilog 语言的结构化描述设计 2 选 1 多路选择器

在 C:/sysclassfiles/organization/Ex_2 文件夹中创建 mux2x1verilog 项目。按照上面的步骤将 7 个基本门电路的 IP 核调入到项目中。

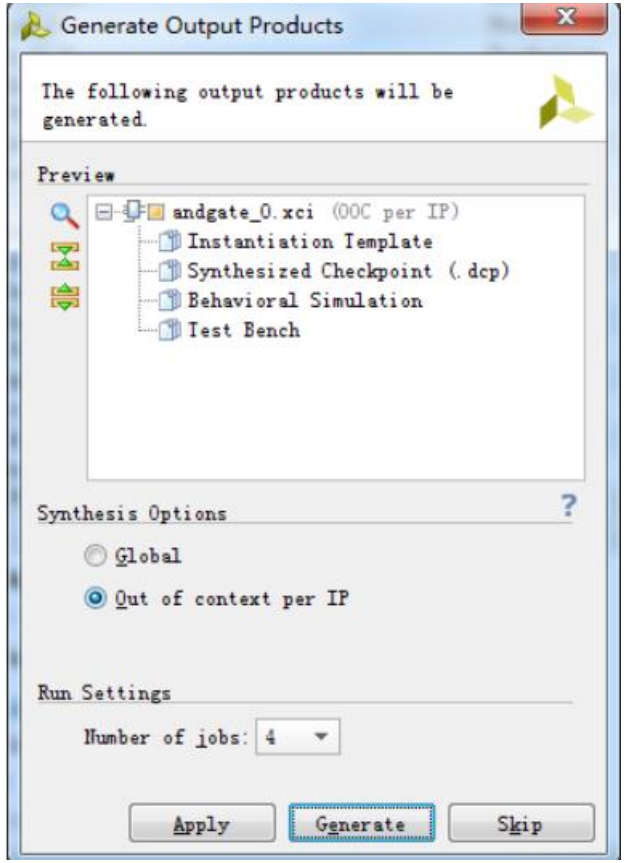
点击 Project Manager 下的 IP Catalog，在窗口中双击 andgate_v1_0，就会打开图示的窗口。按照图中设置数据宽度为 1，然后点击OK。

用 Verilog 语言的结构化描述设计 2 选 1 多路选择器



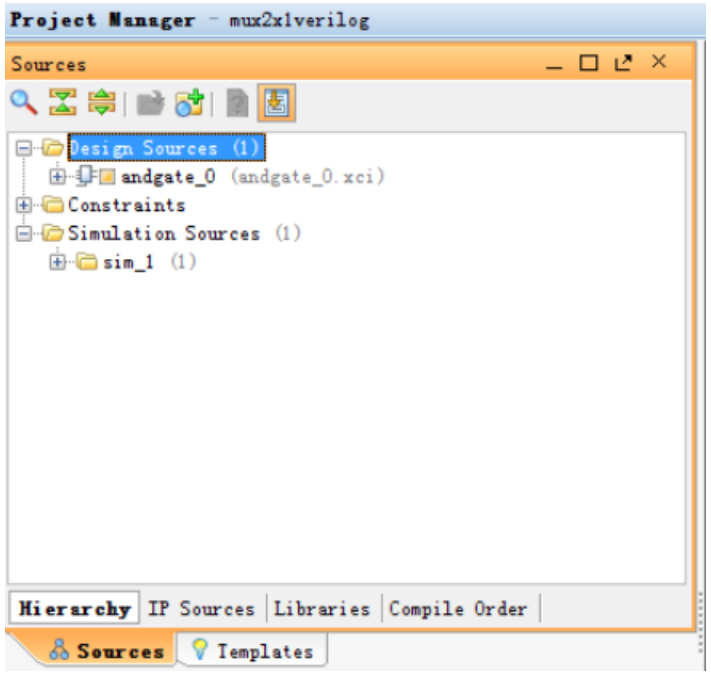
用 Verilog 语言的结构化描述设计 2 选 1 多路选择器

弹出如图示的 Generate Output Products 窗口。点击 Generate。



用 Verilog 语言的结构化描述设计 2 选 1 多路选择器

在随后弹出的对话框中点击 OK。此时我们在 Project Manager 的 Sources 中可以看到刚加进去的 andgate_0，如图示。



按照此方法再设置或门和非门。

用 Verilog 语言的结构化描述设计 2 选 1 多路选择器

创建如下所示的 mux2x1verilog.v 文件，包含 mux2x1verilog 模块。

```
module mux2x1verilog(
    input a,
    input b,
    input s,
    output c
    );
    wire a1,b1,sel;
    notgate_0 u0(.a(s),.c(sel));
    ……
endmodule
```

```
module mux2x1verilog_sim( );
    // input
    reg a=0;
    reg b=1;
    reg s=0;
    //output
    wire c;
    mux2x1verilog u(.a(a),.b(b),.s(s),.c(c));
    initial begin
        # 200 s=1;
    end
endmodule
```

加减法器的设计

使用 Verilog HDL 语言实现一个可适应从 4 位到 32 位数据运算的加减法器，其中被操作数为 a，操作数为 b，加减法控制信号为 sub，当 sub 为 1 的时候做减法，为 0 的时候做加法。另外输出为运算结果 sum， 还有进位/借位标志 cf，有符号数溢出标志 ovf，符号标志 sf以及结果为 0 标志 zf。将该加减法器封装成 IP 核 addsub。

加减法器的设计

下面是 addsub 模块的端口定义，其他部分请大家自行完善。

```
module addsub
#(parameter WIDTH=8) //指定数据宽度参数， 缺省值是 8
(
    input [(WIDTH-1):0] a, // 被操作数， 位宽由参数 WIDTH 决定
    input [(WIDTH-1):0] b, // 操作数， 位宽由参数 WIDTH 决定
    input sub, // =1 为减法
    output [(WIDTH-1):0] sum, // 结果
    output cf, // 进位标志
    output ovf, // 溢出标志
    output sf, // 符号标志
    output zf // 为 0 标志
    );
    ……
```

加减法器的设计

关于加减法器的 CF 和 OF 的问题

1. 关于无符号数的进位 CF：

如果是加法，则 CF 就是二进制运算的进位位

由于减法是将减数取反加 1（求补）后（假设减数 b 求补后为 subb）与被减数 a 相加，因此，当够减的时候反而有进位，不够减的时候反而无进位（其实是无符号数溢出），因此，CF 需要在进位位基础上取反才能表示借位（实际上是让 CF 为 1 表示被减数 a 小于减数 b，CF=0 表示被减数 a 大于减数 b）

因此 CF 在做加法的时候不将加法进位取反，做减法的时候要将加法进位取反。

加减法器的设计

关于加减法器的 CF 和 OF 的问题

2.关于有符号数的溢出问题（OF）

在组成原理中对于有符号数溢出有一个规则就是两数相加最高位和次高位都进位或者都不进位的时候没有溢出，否则就有溢出。但这必须去看最高位和次高位是不是有进位。 下面我们换一个简单的方法来判断。

1） 对于加法

有符号数加法溢出的规则其实很简单，就是两个正数相加得到负数，或者两个负数相加得到正数的时候，有符号数加法就溢出了。

2）对于减法

有符号数减法溢出的规则也很简单：一个正数减去一个负数得到一个负数或者一个负数减去一个正数得到一个正数的时候，就产生了溢出

加减法器的设计

关于加减法器的 CF 和 OF 的问题

2.关于有符号数的溢出问题（OF）

考虑一下：我们是要用加法器做减法，因此我们会将减数 b 取补后的 subb 作为加数和被加数 a 相加，因此，最终，判断溢出应该是判断 a 和 subb 相加的溢出规则。如果我们在做加法的时候让 subb=b，在做减法的时候用 subb=b 的补数，那么最终无论加法或减法，都化作了 sum=a+subb，因此只要判断 a 和 subb 相加的溢出规则。按照加法溢出规则， 请给出 of 的逻辑表达式。

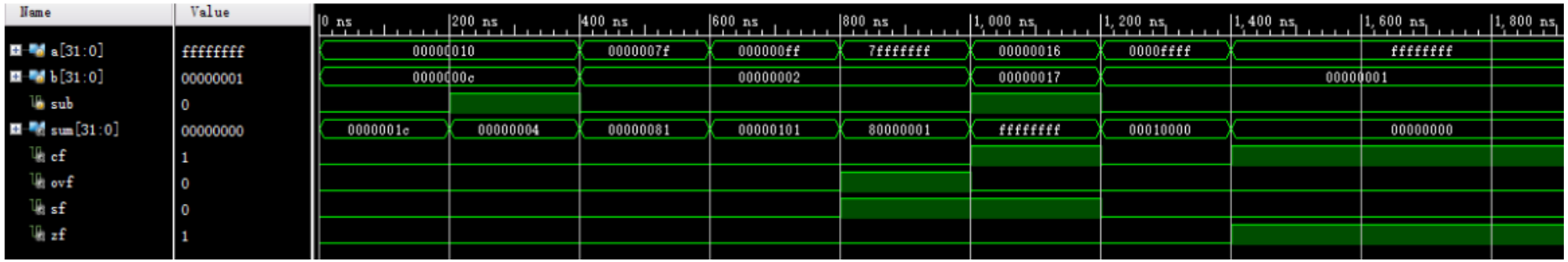
加减法器的设计

仿真设计

```
`timescale 1ns / 1ps
module addsub_sim( );
    // input
    reg [31:0] a = 32'd16;
    reg [31:0] b = 32'd12;
    reg sub = 0;
    //output
    wire [31:0] sum;
    wire cf;
    wire ovf;
    wire sf;
    wire zf;
    // initial
    addsub #(32) U (a,b,sub,sum,cf,ovf,sf,zf);
    initial begin
    #200 sub = 1;
    #200 begin a = 32'h7f; b = 32'h2; sub = 0; end
    #200 begin a = 32'hff; b = 32'h2; sub = 0; end
    #200 begin a = 32'h7fffffff; b = 32'h2; sub = 0; end
    #200 begin a = 32'h16; b = 32'h17; sub = 1; end
    #200 begin a = 32'hffff; b = 32'h1; sub = 0; end
    #200 begin a = 32'hffffffff; b = 32'h1; sub = 0; end
    end
endmodule
```

加减法器的设计

仿真设计



课堂练习2（加减法器的设计）

使用 Verilog HDL 语言以及上述实现的加减法器 IP 核，实现一个 8 位的加减法器addsub8，并下载到板子上进行验证。 其中输入 a[7]~a[0]分别接 SW15~SW8, b[7]~b[0]分别接SW7~SW0, sub 接 SW23, sum[7]~sum[0]分别接 YLD7~YLD0。 ovf 接 GLD7， cf 接 GLD6，sf 接 GLD5， zf 接 YLD4

乘法器的设计

1. 无符号数乘法器的设计

基于原码一位乘法的数据宽度可变的无符号数乘法器 IP 核的设计

```
   1010
  ×1101
  ------
   1010
  0000
 1010
1010
--------
10000010
```

从手算乘法不难看到两个二进制无符号数相乘的算法可以描述成：

（1） 令 X=被乘数，位宽为 N 位，Y=乘数，位宽为 1，且 $Y=y_3\times2^3+y_2\times2^2+y_1\times2^1+y_0\times2^0$;

（2） 设置 CNT=位宽 N，设置积 P=0，P 的位宽是 2N，将 X 的位宽高位 0 扩展到 2N 位

（3） 判断当前乘数最低位 y_0 是否为 1，如果为 1，则令 P=P+X，否则转步骤（4）;

（4） X=X 左移一位，Y=Y 逻辑右移 1 位;

（5） CNT=CNT-1，判断 CNT 是否为 0，如果不为 0，则转到（3），否则结束。

乘法器的设计

1. 无符号数乘法器的设计

```
module mulu_hand
#(parameter WIDTH = 8)
(
    input [WIDTH-1:0] a,
    input [WIDTH-1:0] b,
    output reg [WIDTH*2-1:0] c
    );
    integer cnt;
    reg [WIDTH-1:0] y,t;
    reg [WIDTH*2-1:0] x,p;
    always @(*)
    begin
        t = {WIDTH{1'b0}};
        x = {t,a}; // 扩展被乘数到 2N 位
        p = {WIDTH*2{1'b0}}; // 积初始化为 0
        y = b;
        for(cnt = 0; cnt<WIDTH; cnt=cnt+1) // 循环迭代
        begin
            if(y[0] == 1)
                p = p + x; // 部分积+被乘数
            x = x << 1; // 被乘数左移
            y = y >> 1; // 乘数右移
        end
        c = p;
    end
endmodule
```

请大家在 Vivado 中实现该模块并仿真看其结果是否正确。

乘法器的设计

上面的方法功能上是正确的，但由于乘积的位数是被乘数和乘数位数的 2 倍因此需要一个位宽是 2N 的加法器，另外，这个方法需要用到左移和右移两种移位寄存器。

为了提高效率，计算机中对上述乘法进行了改进。我们来考察一下两个 4 位无符号数 X×Y 的计算过程的推导。假设 $Y=y_3\times2^3+y_2\times2^2+y_1\times2^1+y_0\times2^0$

$$X\times Y=X\times y_3\times2^3+X\times y_2\times2^2+X\times y_1\times2^1+X\times y_0\times2^0$$

$$=2^4\times\underline{(X\times y_3\times2^{-1}+X\times y_2\times2^{-2}+X\times y_1\times2^{-3}+X\times y_0\times2^{-4})}$$

注意上述推导式中把一个 4 位整数分成了两个部分，其中下划线部分已经成了一个纯小数。而 $2^4\times$ 的作用就是把这个纯小数再还原成整型数。因为 $2^4\times$ 只改变了小数点的位置，因此在考察乘法算法的操作过程中不去考虑这个部分，因为算法操作过程本身也不考虑小数点的位置。

我们将 $X\times y_3\times2^{-1}+X\times y_2\times2^{-2}+X\times y_1\times2^{-3}+X\times y_0\times2^{-4}$ 这部分再进行推导。

乘法器的设计

$$X\times y_3\times 2^{-1}+X\times y_2\times 2^{-2}+X\times y_1\times 2^{-3}+X\times y_0\times 2^{-4}$$
$$=2^{-1}\times(X\times y_0\times 2^{-3}+X\times y_1\times 2^{-2}+X\times y_2\times 2^{-1}+X\times y_3)$$
$$=2^{-1}\times(2^{-1}\times(X\times y_0\times 2^{-2}+X\times y_1\times 2^{-1}+X\times y_2)+X\times y_3)$$
$$=2^{-1}\times(2^{-1}\times(2^{-1}\times(X\times y_0\times 2^{-1}+X\times y_1)+X\times y_2)+X\times y_3)$$
$$=2^{-1}\times(2^{-1}\times(2^{-1}\times(2^{-1}\times(0+X\times y_0)+X\times y_1)+X\times y_2)+X\times y_3)$$

假设 $P_0=0$；则我们可以得出以下推导过程：

$$P_1=2^{-1}\times(P_0+X\times y_0)$$
$$P_2=2^{-1}\times(P_1+X\times y_1)$$
$$P_3=2^{-1}\times(P_2+X\times y_2)$$
$$P_4=2^{-1}\times(P_3+X\times y_3)$$

不失一般性，我们不难得出部分积 P_i 的推导公式为

$$P_{i+1}=2^{-1}\times(P_i+X\times y_i)\quad(i=0,1,2,\ldots,n-1)$$

乘法器的设计

在二进制中，$\times 2^{-1}$ 就是右移一位。因此，可以得出下面的无符号数原码一位乘法的操作步骤。

（1） 令 X=被乘数，位宽为 N 位，Y=乘数，位宽为 1，且 $Y=y_3\times 2^3+y_2\times 2^2+y_1\times 2^1+y_0\times 2^0$；

（2） 设置 CNT=位宽 N，设置部分积 P=0，P 的位宽是 N，设置加法进位位为 CY=0；

（3） 判断当前乘数最低位 y_0 是否为 1，如果为 1，则令{CY,P}=P+X，否则转步骤（4）；

（4） {CY,P,Y}联合右移一位，CNT=CNT-1；，

（5） 判断 CNT 是否为 0，如果不为 0，则则转到（3）。否则结束，P 是积的高 N 位，Y 中是积的低 N 位。

乘法器的设计

CNT	C	P	Y	说明
4	0	0000	1101	$P_0=0$
		+1010		$Y_0=1$，+X
	0	1010		{CY,P,Y}联合右移一位
	0	0101	0110	得 P_1
3	0	0101	0110	$Y_1=1$，{CY,P,Y}联合右移一位
	0	0010	1011	得 P_2
2		+1010		$Y_2=1$，+X
	0	1100		{CY,P,Y}联合右移一位
	0	0110	0101	得 P_3
1		+1010		$Y_3=1$，+X
	1	0000		{CY,P,Y}联合右移一位
	0	1000	0010	得 P_4
0				结束，结果是 10000010

乘法器的设计

使用 Verilog HDL 语言行为级描述方法，根据原码 1 位乘法的原理， 实现乘数和被乘数数据宽度在 4 位~32 位之间可变的无符号数乘法器 mulu。并请通过 mulu_sim.v 文件，仿真验证你的无符号数乘法器。最后，请将你的无符号数乘法器封装成 IP 核。

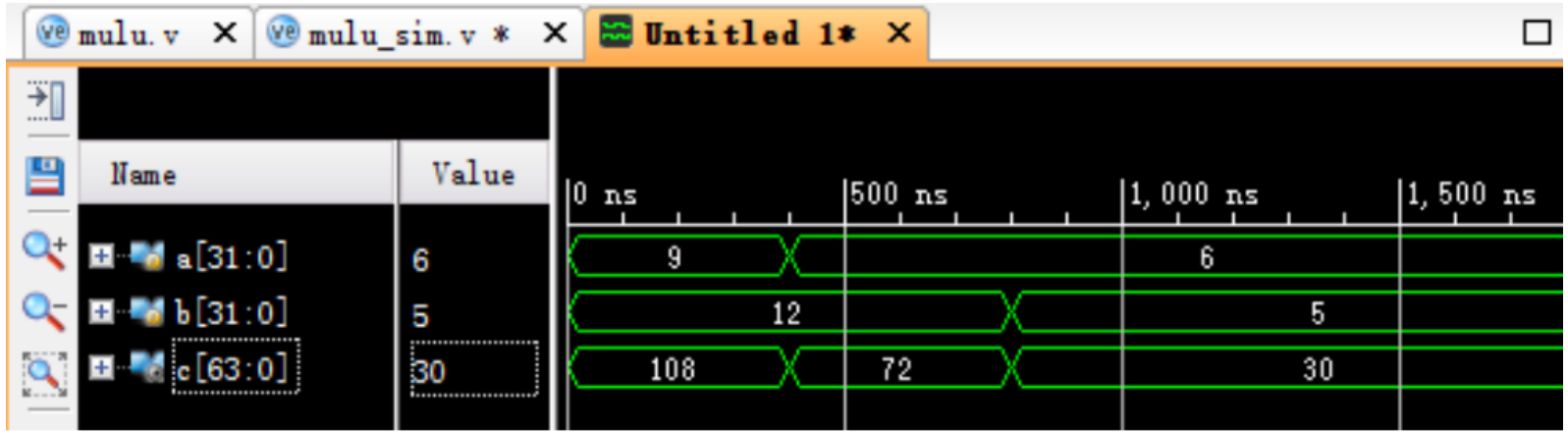
mulu 模块的端口定义如下：

```
module mulu
#(parameter WIDTH = 8)
(
    input [WIDTH-1:0] a, // 被乘数
    input [WIDTH-1:0] b, // 乘数
    output reg [WIDTH*2-1:0] c // 乘积
    );
    ……
endmodule
```

请大家根据上面的原码一位乘法算法， 自行完善设计的其他部分。

乘法器的设计

```
`timescale 1ns / 1ps
module mulu_sim( );
        // input
        reg [31:0] a = 32'd9;
        reg [31:0] b = 32'd12;
        // output
        wire [63:0] c;
        mulu #(32) u(a,b,c);
        initial begin
                #400 a = 32'd6;
                #400 b = 32'd5;
        end
endmodule
```



课堂练习3（无符号数乘法器的设计）

利用上一实验中封装的数据宽度可变的无符号数乘法器 IP 核，采用元件例化的方法实现一个 8 位无符号乘法器 mulux8。并下载到 Minisys 板子上进行验证。 被乘数 a[7:0]接到SW15~SW8， 乘数 b[7:0]接到 SW7~SW0。 乘积 c[15:0]接到 GLD7~GLD0,YLD7~YLD0

乘法器的设计

2. 有符号数乘法器的设计

数据宽度可变的有符号数乘法器 IP 核的设计

使用 Verilog HDL 语言行为级描述方法，根据原码乘法的原理，实现乘数和被乘数数据宽度在 4 位~32 位之间可变的有符号数乘法器 mul。 并请通过编写 mul_sim 文件，仿真验证你的有符号数乘法器。最后，请将你的有符号乘法器封装成 IP 核。

mul 和 mulu 最大的区别是操作数和结果都是有符号数， 由于机器内负数都是用补码表示的，因此，我们要分别判断被乘数和乘数，如果是负数， 要对它们求补， 得到原码， 再将数值位进行原码一位乘法运算。运算出来结果后，还要根据被乘数和乘数的符号位特点判断结果是否为负数，如果是负数， 需要对结果求一次补。 在做原码一位乘的时候，依然要注意进位位要参与到右移运算中。

乘法器的设计

```
module mul
#(parameter WIDTH = 8)
(
    input [WIDTH-1:0] a, // 被乘数
    input [WIDTH-1:0] b, // 乘数
    output reg [WIDTH*2-1:0] c // 乘积
    );
    integer cnt; // 循环变量
    reg [WIDTH-1:0] x,y,p; // 存放被乘数、乘数和部分积
    reg sign,cy; // 进位位
```

```
always @(*)
begin
    cy = 0;
    x = a;
    y = b;
    p = {WIDTH{1'b0}};
    if(a[WIDTH-1] == 1) // 被乘数是负数的处理
    begin
        x = ~x;
        x = x + 1'b1;
    end
        ……// 乘数是负数的处理、原码一位乘、结果符号的处理
end
endmodule
```

乘法器的设计

```
`timescale 1ns / 1ps
module mul_sim( );
        // input
        reg [31:0] a = 32'd9;
        reg [31:0] b = 32'd12;
        // output
        wire [63:0] c;
        mul #(32) u(a,b,c);
        initial begin
        #400 a = 32'hfffffffe; // -2
        #400 b = 32'hfffffffe; // -2
        end
endmodule
```

课堂练习4（有符号数乘法器的设计）

利用上一实验中封装的数据宽度可变的有符号数乘法器 IP 核，采用元件例化的方法实现一个 8 位有符号数乘法器 mulx8。并下载的 Minisys 板子上进行验证。被乘数 a[7:0] 接到SW15~SW8，乘数 b[7:0]接到 SW7~SW0。 乘积 c[15:0] 接到 YLD7~YLD0,GLD7~GLD0。

除法器的设计

1. 无符号数除法器的设计

使用 Verilog HDL 语言行为级描述方法，根据不恢复余数除法的原理，实现被除数数据宽度在 8，16， 32 位之间可变的无符号数除法器 divu。并请通过编写 divu_sim 文件，仿真验证你的无符号数除法器。最后，请将你的无符号数除法器封装成 IP 核。其中，被除数为a，除数为 b，商为 q，余数为 r（b 和 r 的宽度是被除数的一半，商的数据宽度同被除数）。作为辅助信号，有时钟信号 clk，启动除法操作的信号 start，复位信号 resetn，另外， 除法器应该有一个输出信号 busy，当除法运算开始的时候，该信号为高电平，除法运算结束后，busy 转为低电平。

除法器的设计

1. 无符号数除法器的设计

```
module divu
#(parameter WIDTH = 8)
        (
        input [WIDTH-1:0] a, // 被除数
        input [WIDTH/2-1:0] b, // 除数
        input clk,
        input start,
        input resetn,
        output [WIDTH-1:0] q, // 商
        output [WIDTH/2-1:0] r, // 余数
        output reg busy // 正在做除法
        );
        ......
endmodule
```

除法器的设计

1. 无符号数除法器的设计

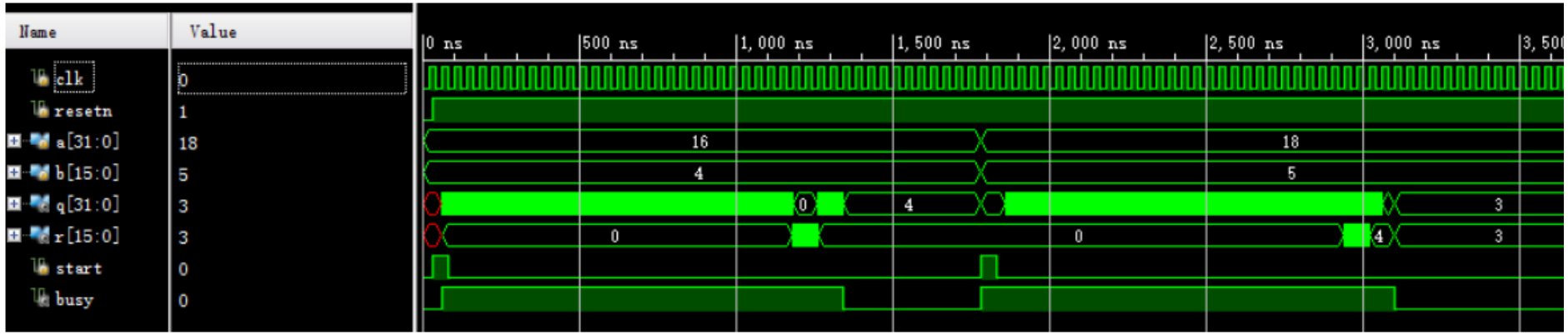
```
`timescale 1ns / 1ps
module divu_sim( );
    // input
    reg [31:0] a = 32'd16;
    reg [15:0] b = 16'd4;
    reg clk = 0;
    reg start = 0;
    reg resetn = 0;
    // output
    wire [31:0] q;
    wire [15:0] r;
    wire busy;
    divu #(32) u(a,b,clk,start,resetn,q,r,busy);
```

仿真文件

```
    initial begin
        #30 begin resetn = 1;start = 1;end
        #50 start = 0;
        #1400 begin start = 1; a = 32'd18;b = 16'd5;end
        #50 start = 0;
    end
    always #20 clk = ~clk;
endmodule
```

除法器的设计

1. 无符号数除法器的设计



数据都已经以无符号十进制数来表示， 除法的结果在 busy 信号无效的时候出来。

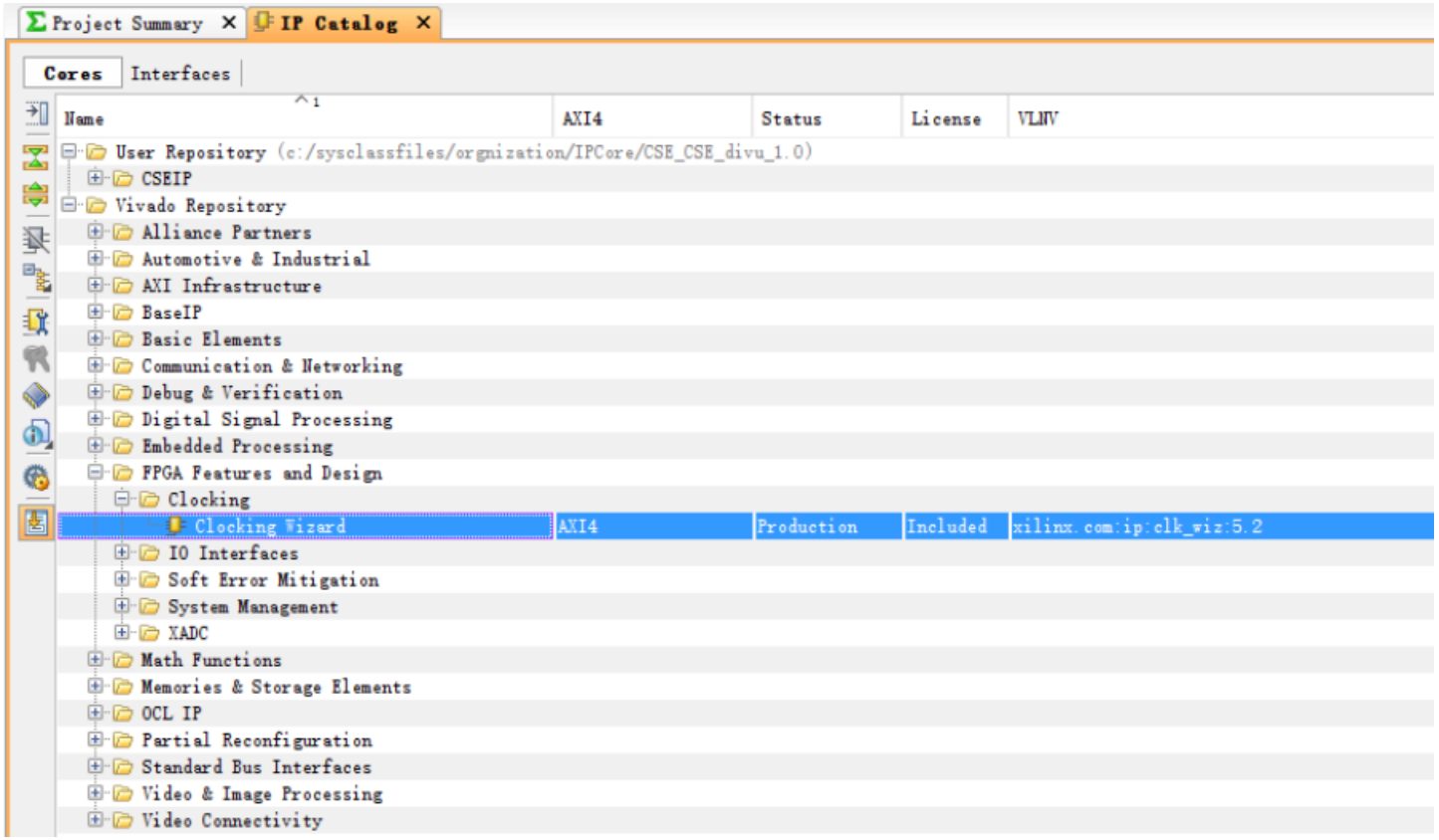
课堂练习5（无符号数除法器的设计）

16 位无符号数除法器的设计

利用上一实验中封装的数据宽度可变的无符号数除法器 IP 核，采用元件例化的方法实现一个 16 位无符号数除法器 divux16。并下载的 Minisys 板子上进行验证。被除数 a[15:0]接到 SW23~SW8， 除数 b[7:0]接到 SW7~SW0。 商 q[15:0]接到 GLD7-GLD0,YLD7~YLD0，余数 r[7:0] 接到 RLD7~RLD0。 clk 接到 Y18， resetn 接到 P20， start 接到 S4 按键开关。

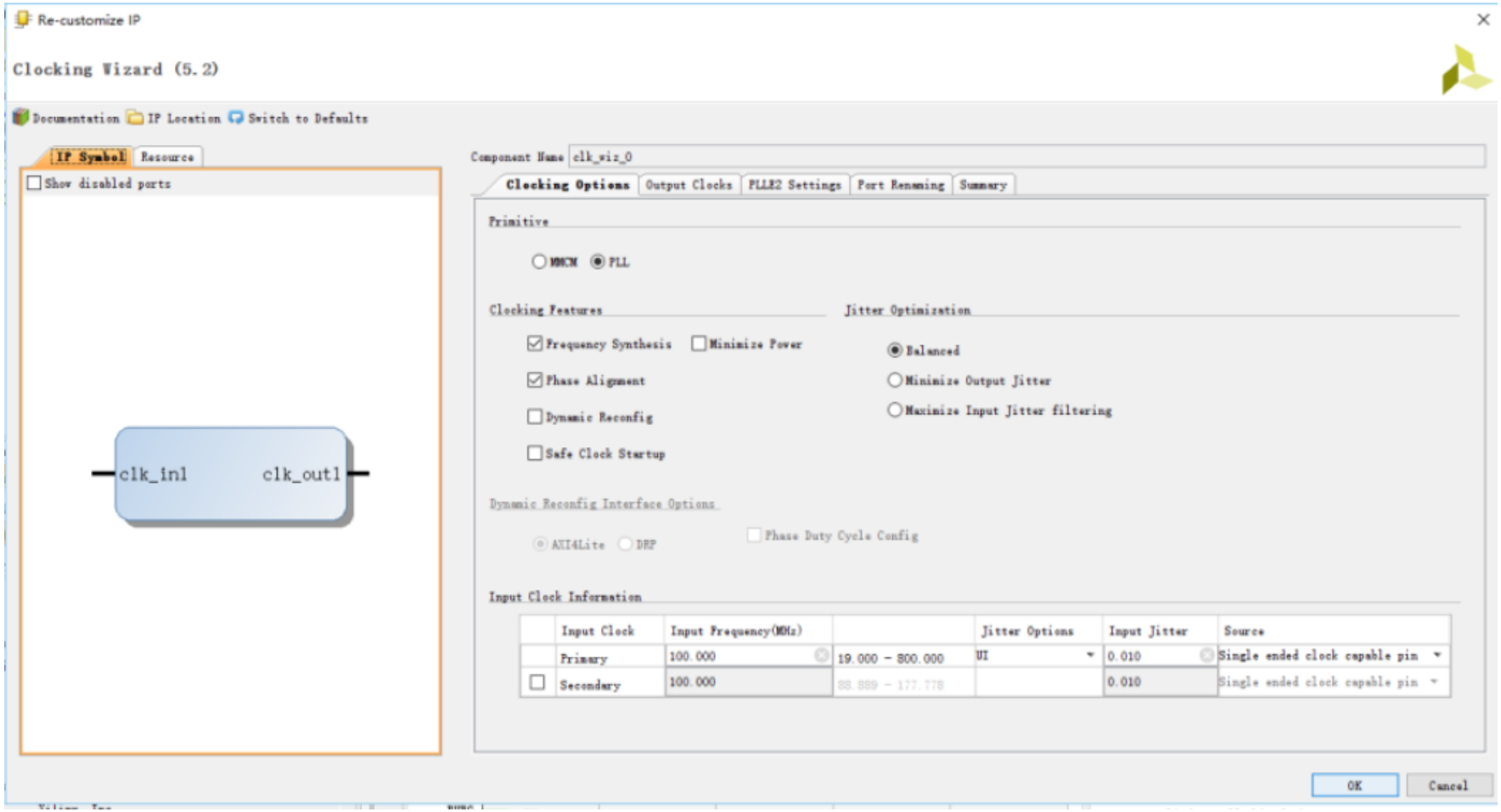
由于 Minisys 板上的时钟是 100MHz,,我们需要降频到 50MHz，为了获得一个稳定的降频时钟，我们采用 Vivado 自带的 PLL 时钟来做降频。点击 IP Catalog，并在打开的 IP 核库中双击 Xilinx Repository->FPGA Features and Design->Clocking->Clocking Wizard。

除法器的设计


Project Summary
IP Catalog
Cores
Interfaces
Name
AXI4
Status
License
VLNV
User Repository (c:/sysclassfiles/orgnization/IPCore/CSE_CSE_divu_1.0)
CSEIP
Vivado Repository
Alliance Partners
Automotive & Industrial
AXI Infrastructure
BaseIP
Basic Elements
Communication & Networking
Debug & Verification
Digital Signal Processing
Embedded Processing
FPGA Features and Design
Clocking
Clocking Wizard
AXI4
Production
Included
xilinx.com:ip:clk_wiz:5.2
IO Interfaces
Soft Error Mitigation
System Management
XADC
Math Functions
Memories & Storage Elements
OCL IP
Partial Reconfiguration
Standard Bus Interfaces
Video & Image Processing
Video Connectivity

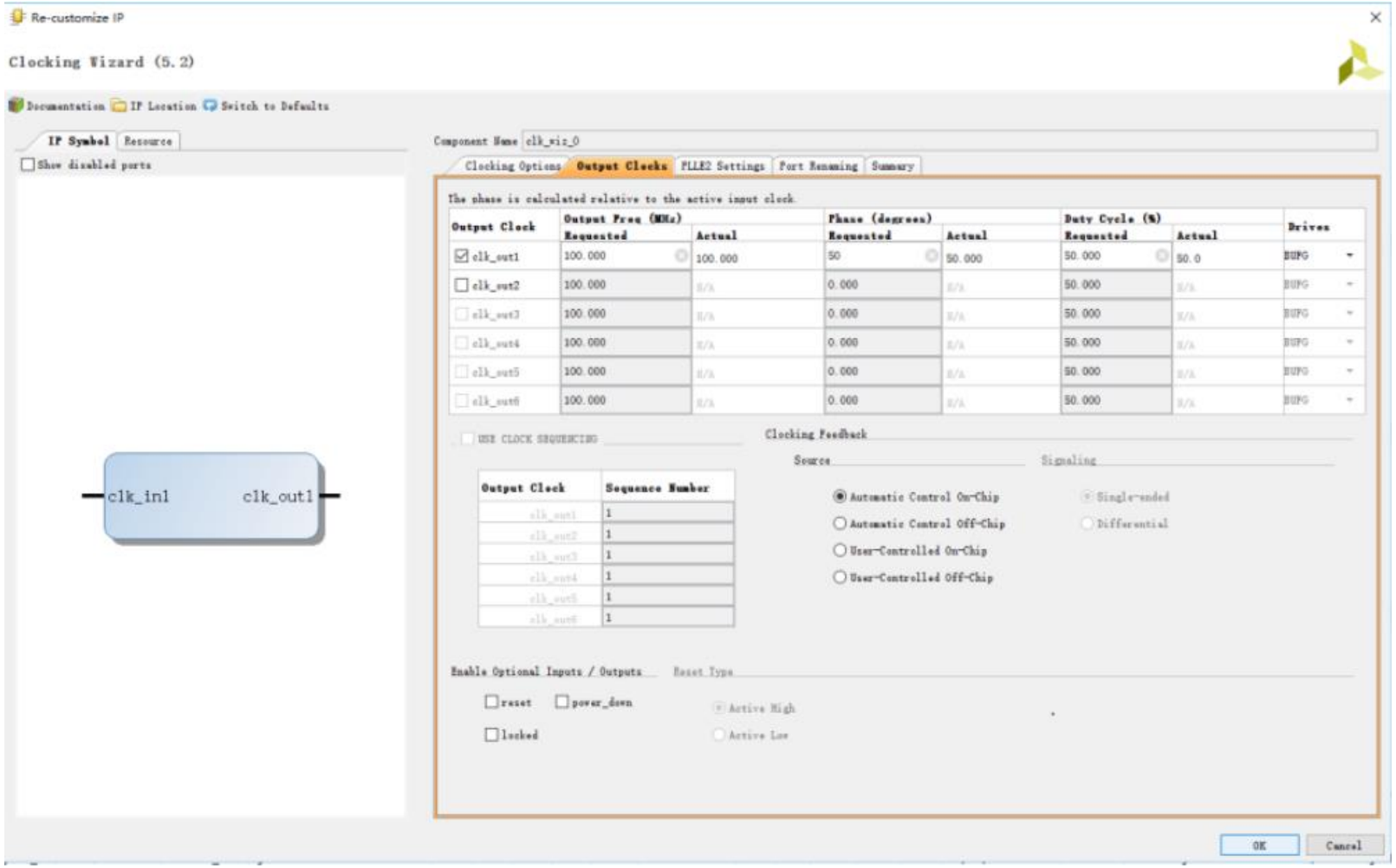
除法器的设计

在时钟 IP 核设置中，分别如下图所示设置。



除法器的设计

在时钟 IP 核设置中，分别如下图所示设置。



除法器的设计

将上一题做好的 divu IP 核调进来，在 divux16.v 中实例化这两个 IP 核，注意判断除数为 0 的情况，另外还要注意启动触发器需要 start 信号有效而 busy 信号无效的时候才行。

divux16 模块的端口定义如下：

```
module divux16(
    input [15:0] a,
    input [7:0] b,
    input clk,
    input start,
    input reset,
    output [15:0] q,
    output [7:0] r
    );
    ......
```

除法器的设计

2.有符号数除法器的设计

使用 Verilog HDL 语言行为级描述方法，根据不恢复余数除法的原理，实现被除数数据宽度在 8，16， 32 位之间可变的有符号数除法器 div。并请通过编写 div_sim 文件，仿真验证你的有符号数除法器。最后，请将你的有符号数除法器封装成 IP 核。其中，被除数为 a，除数为 b（宽度是被除数的一半），商为 q，余数为 r（b 和 r 的宽度是被除数的一半，商的数据宽度同被除数）。作为辅助信号，有时钟信号 clk，启动除法操作的信号 start，复位信号resetn，另外，除法器应该有一个输出信号 busy，当除法运算开始的时候，该信号为高电平，除法运算结束后，busy 转为低电平。 商和余数的符号规则是，被除数和除数符号相同，商的符号位正，否则商的符号位负。余数的符号同被除数。

除法器的设计

2.有符号数除法器的设计

```
module div
#(parameter WIDTH = 8)
        (
        input [WIDTH-1:0] a,
        input [WIDTH/2-1:0] b,
        input clk,
        input start,
        input resetn,
        output [WIDTH-1:0] q, // 商
        output [WIDTH/2-1:0] r, // 余数
        output reg busy // 正在做除法
        );
        ……
        endmodule
```

除法器的设计

仿真文件

```
module div_sim( );
    // input
    reg [31:0] a = 32'd16;
    reg [15:0] b = 16'd4;
    reg clk = 0;
    reg start = 0;
    reg resetn = 0;
    // output
    wire [31:0] q;
    wire [15:0] r;
    wire busy;
    div #(32) u(a,b,clk,start,resetn,q,r,busy);
    initial begin
        #30 begin resetn = 1;start = 1;end
        #50 start = 0;
        #1400 begin start = 1; a = 32'd18;b = 16'd5;end
        #50 start = 0;
        #1400 begin start = 1; a = 32'hfffffffee;b = 16'd5;end
        #50 start = 0;
        #1400 begin start = 1; a = 32'd18;b = 16'hfffb;end
        #50 start = 0;
    end
    always #20 clk = ~clk;
endmodule
```

课堂练习6（无符号数除法器的设计）

16 位有符号数除法器的设计

利用上一实验中封装的数据宽度可变的有符号数除法器 IP 核，采用元件例化的方法实现一个 16 位有符号数除法器 divx16。并下载的 Minisys 实验板上进行验证。 被除数 a[15:0]接到 SW23~SW8，除数 b[7:0]接到 SW7~SW0。商 q[15:0]接到 YLD7-YLD0,GLD7~GLD0，余数 r[7:0] 接到 RLD7~RLD0。 clk 接到 Y18， resetn 接到 P20， start 接到 S4 按键开关。

运算器(ALU)的设计

利用前面设计好的可变数据位数的加减法器 IP 核以及可配置输入端数目和数据位宽的与、或、非、异或门、多路选择器 1P 核以及 8 位桶形移位器 IP 核作为基本元件，利用Vivado 的 Block Design 或者利用 Verilog 的结构化描述方法与数据流描述方法完成一个 8 位的运算器 alu8_verilog

运算器(ALU)的设计

其功能如表所示。其中 op[3:0]是操作码，输入的 8 位操作数分别是 a[7:0]和 b[7:0]，运算结果是 res[7:0]，另外输出还有进位标志 cf，有符号数溢出标志 of，结果为 0 标志 zf 以及有符号数符号标志 sf。

op[3]	op[2]	op[1]	op[0]	功能
0	0	X	0	res=a+b，输出有 cf,of,zf,sf
0	0	X	1	res=a-b，输出有 cf,of,zf,sf
0	1	0	0	res=a and b，输出有 cf=0,of=0,zf,sf
0	1	0	1	res=a or b，输出有 cf=0,of=0,zf,sf
0	1	1	0	res=~a，输出有 cf=0,of=0,zf,sf
0	1	1	1	res=a xor b，输出有 cf=0,of=0,zf,sf
1	X	0	0	res=a 逻辑右移 b[2:0]位，cf=0,of=0,zf,sf
1	X	0	1	res=a 算数右移 b[2:0]位，cf=0,of=0,zf,sf
1	X	1	X	res=a 左移 b[2:0]位，cf=0,of=0,zf,sf

运算器(ALU)的设计

将你的设计综合、实现、并下载到 Minisys 实验板上， 信号与管脚的对应关系是：a[7:0]-sw15-sw8， b[7:0]-sw7-sw0，op[3:0]-sw23~sw20， res[7:0]-YLD7~YLD0， cf-RLD4，of-RLD5， zf-RLD6， sf-RLD7。