Lab1 在PYNQ中测试一个简单的加法器IP

目标

本实验教程的目标是展示如何使用Vivado来创建一个AXI-Lite Slave接口模板文件，然后修改模板文件以封装自己的简单加法器的RTL设计，最后通过PYNQ来调用该IP进行加法功能的测试。本教程将介绍：

- 如何使用Vivado的“Create and Package New IP”功能
- 如何理解AXI-Lite Slave接口模板文件、对其进行修改以封装自己的RTL设计
- 如何将AXI-Lite Slave的IP集成到硬件系统中
- 如何在PYNQ中对AXI-Lite Slave接口的IP进行读写

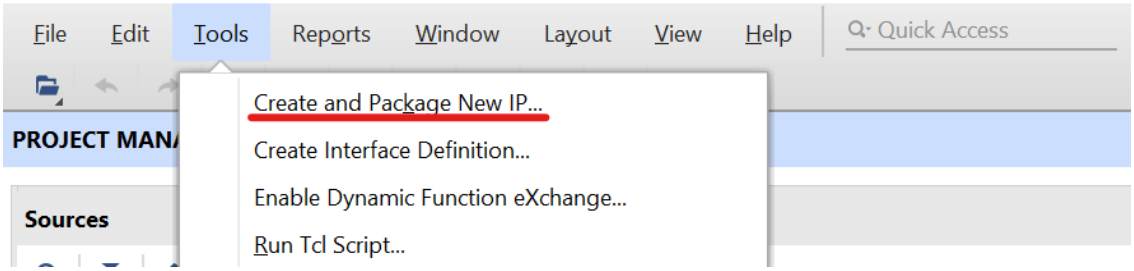
环境要求

- PYNQ-Z2远程实验室服务或物理板卡
- Vivado

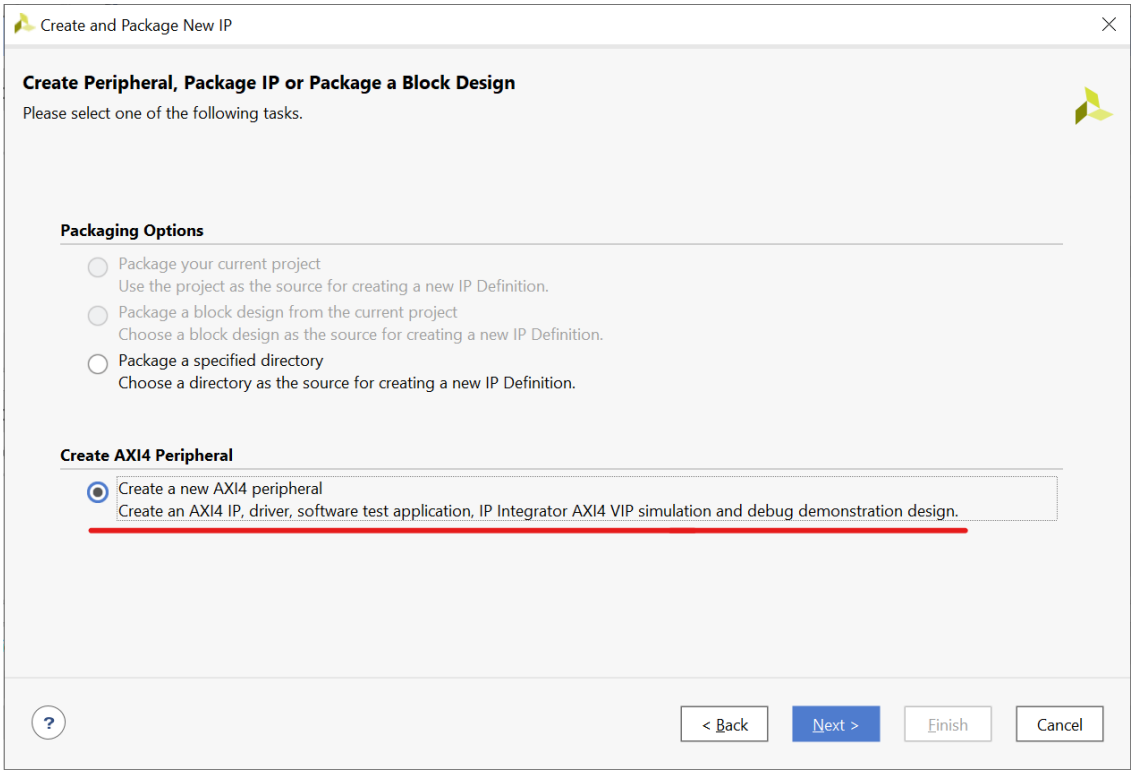
实验步骤

1. 创建一个新的AXI IP

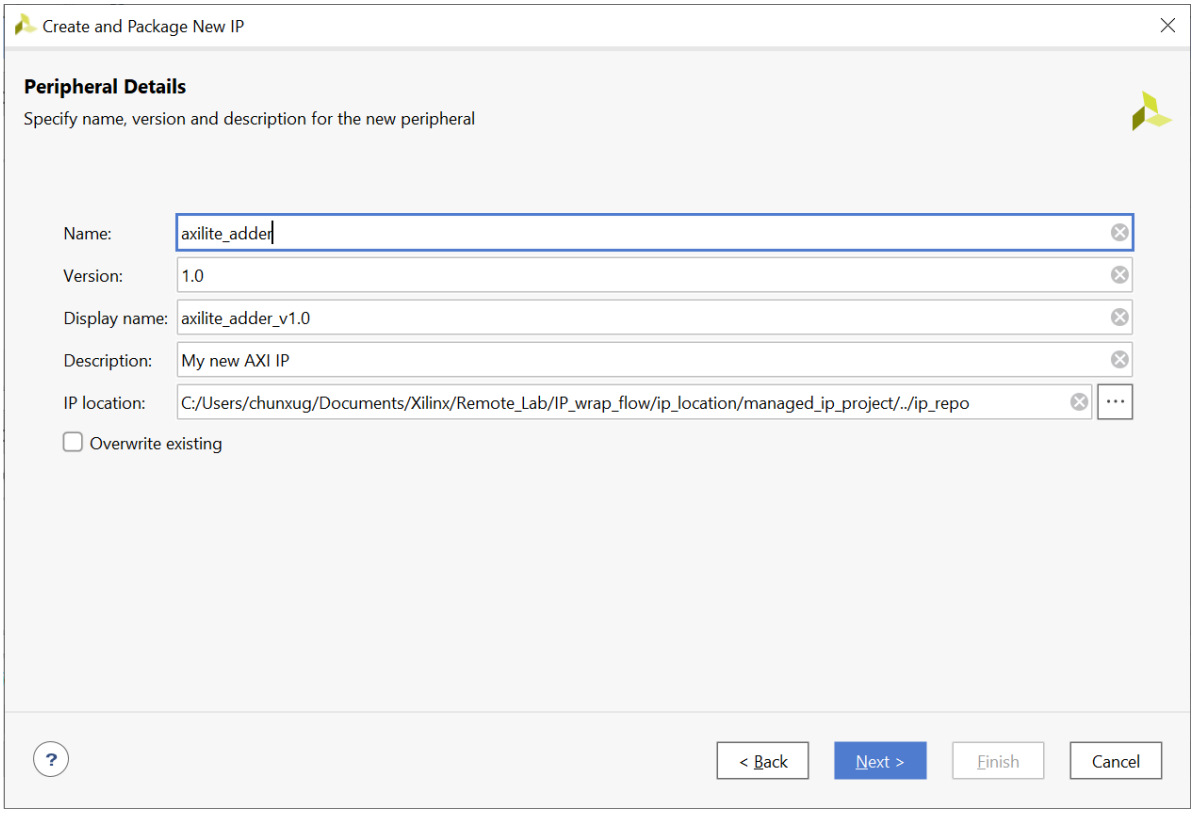
1. 打开**Vivado**软件，打开任意一个项目，然后点击左上角工具栏中的**tools**，选择**Create and Package New IP**以开始创建新的IP核。



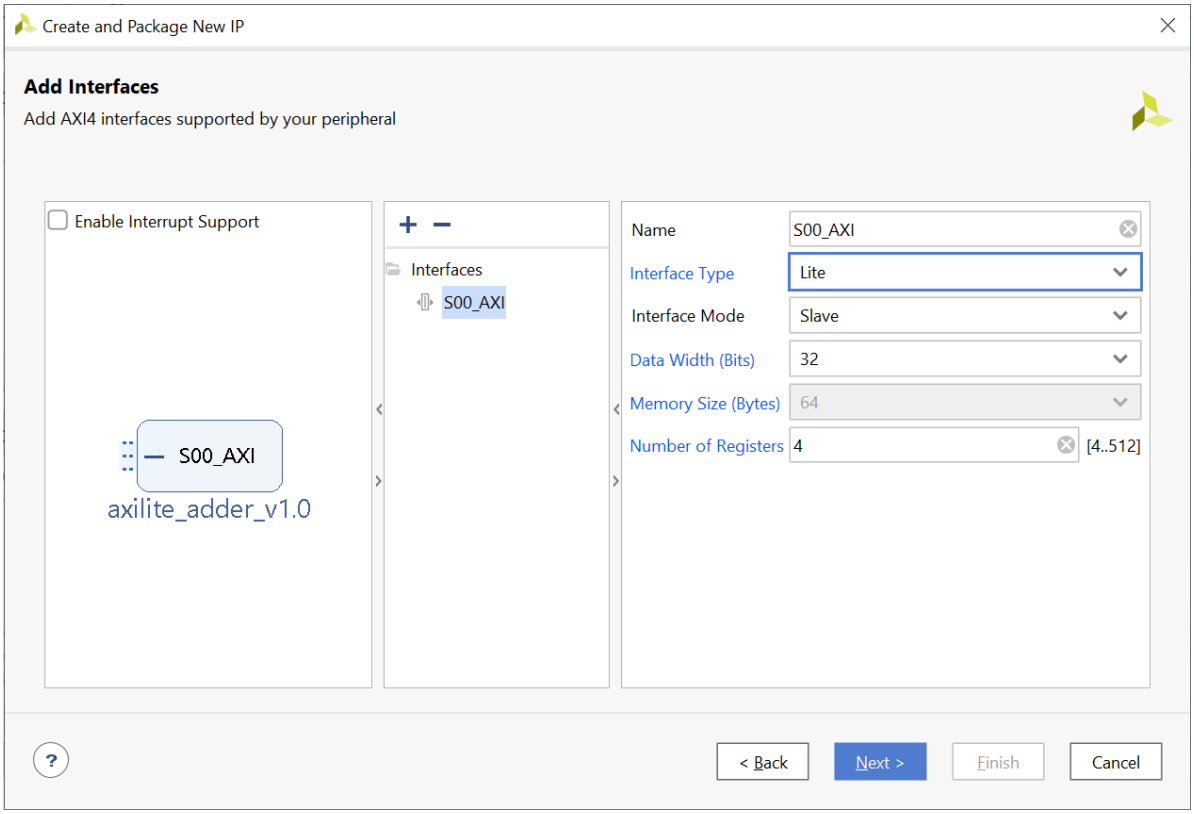
2. 在新弹出的**Create and Package New IP**窗口中选择**Next**以继续
3. 选择**Create a new AXI4 peripheral**的选项，并单击**Next**



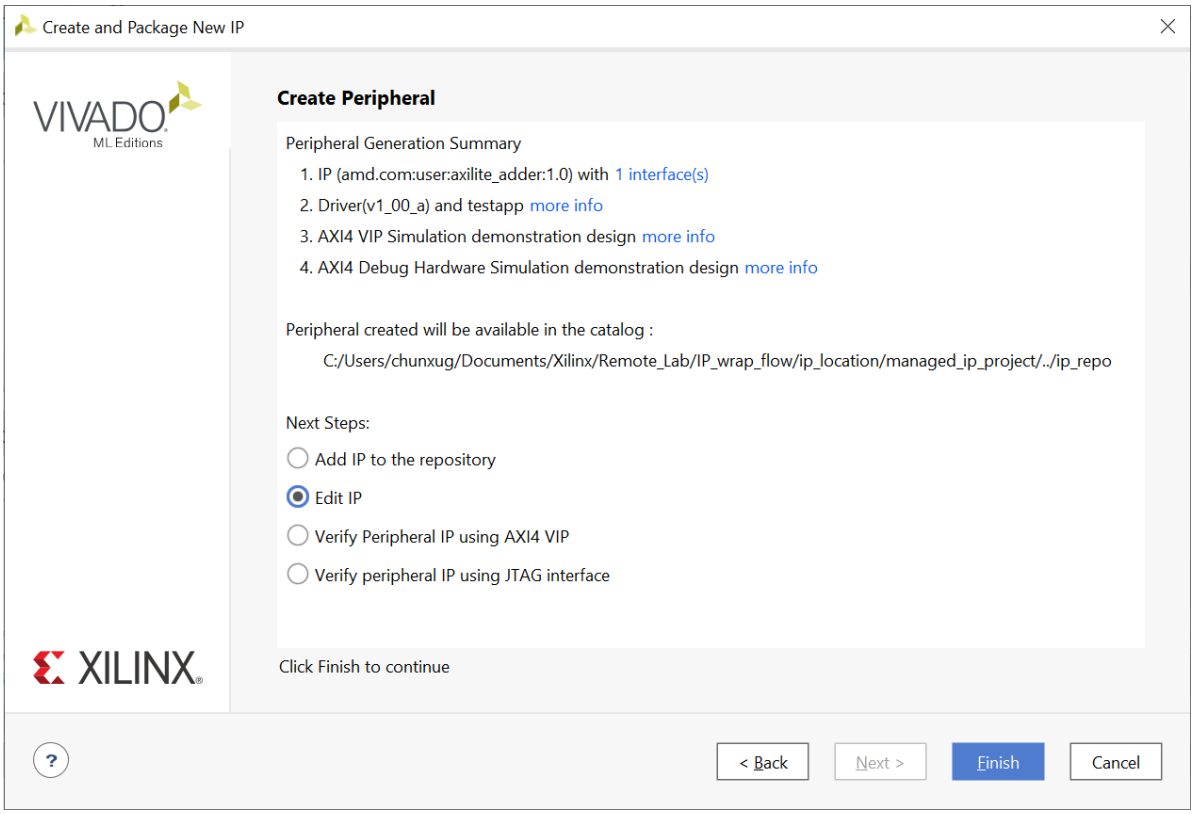
4. 在**Name**栏输入IP名称**axilite_adder**，在**IP location**栏选择自己的工作目录，然后点击**Next**



5. 进入到**Add Interfaces**界面，进行AXI接口的选择与添加。按照如下界面，添加一个**AXI Lite**的**Slave**接口，**Data Width**设置为**32**，**Number of Registers**设置为**4**，然后点击**Next**继续。



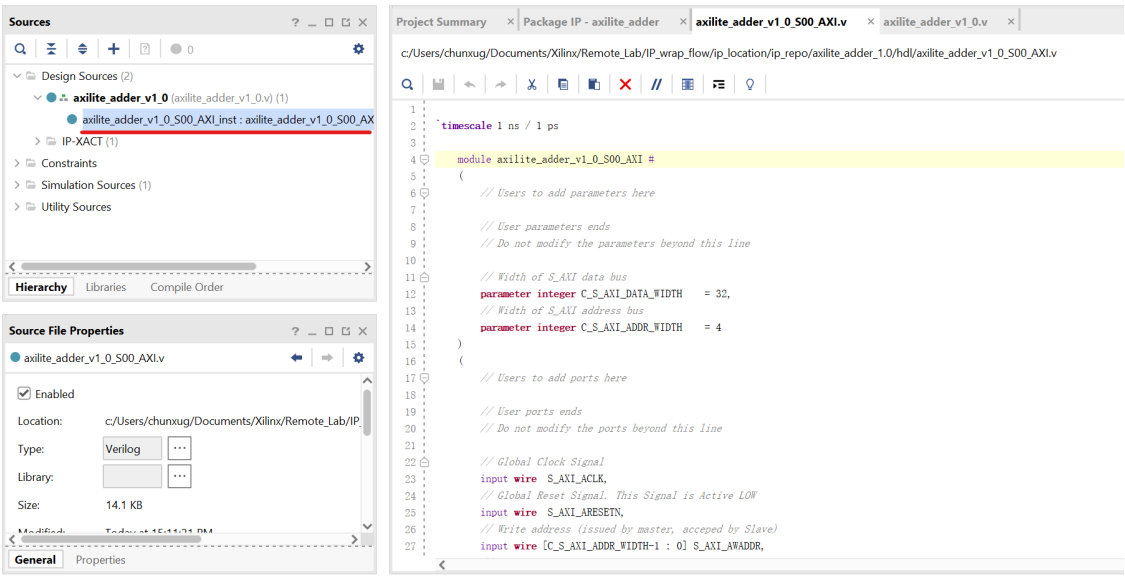
6. 选择**Edit IP**，点击**Finish**，这将打开一个新的Vivado开发界面。



2. 修改接口模板文件

2.1 查看模板文件

1. 在新弹出的**edit_axilite_adder_v1_0**界面中，双击打开**axilite_adder_v1_0_S00_AXI_inst**文件。在**Design Source**中的外层文件**axilite_adder_v1_0.v**是顶层封装，会根据上一步中添加的AXI接口类型实例化对应AXI Interface; **axilite_adder_v1_0_S00_AXI_inst.v**则包含具体的AXI协议握手内容，也是我们需要修改与关注的部分。



2. 虽然该教程剩余部分具体介绍了需要对哪些代码进行修改，但如果读者是第一次接触该类文件，我们仍然建议读者在此花一些时间阅读下**axilite_adder_v1_0_S00_AXI_inst.v**文件的具体内容。

2.2 写入寄存器

1. **axilite_adder_v1_0_S00_AXI_inst.v**的第219到269行包含了由PS向PL AXI Lite 写寄存器的过程，其中 `slv_regN[(byte_index*8) +: 8] <= S_AXI_WDATA[(byte_index*8) +: 8];` 语句实现了从**S_AXI_WDATA**上读取数据写入到寄存器**slv_regN**的过程。

   由于我们在上一步选择了生成4个寄存器，因此这里生成了**slv_reg0-slv_reg3**四个寄存器，在PS侧进行写入后结果将保存在这4个寄存器中。

```
always @( posedge S_AXI_ACLK )
    begin
      if ( S_AXI_ARESETN == 1'b0 )
        begin
          slv_reg0 <= 0;
          slv_reg1 <= 0;
          slv_reg2 <= 0;
          slv_reg3 <= 0;
        end
      else begin
        if (slv_reg_wren)
          begin
            case ( axi_awaddr[ADDR_LSB+OPT_MEM_ADDR_BITS:ADDR_LSB] )
              2'h0:
                for ( byte_index = 0; byte_index <=
(C_S_AXI_DATA_WIDTH/8)-1; byte_index = byte_index+1 )
                  if ( S_AXI_WSTRB[byte_index] == 1 ) begin
                    // Respective byte enables are asserted as per write
strobes
                    // Slave register 0
                    slv_reg0[(byte_index*8) +: 8] <=
S_AXI_WDATA[(byte_index*8) +: 8];
                  end
              2'h1:
                for ( byte_index = 0; byte_index <=
(C_S_AXI_DATA_WIDTH/8)-1; byte_index = byte_index+1 )
                  if ( S_AXI_WSTRB[byte_index] == 1 ) begin
```

```
                    // Respective byte enables are asserted as per write
strobes
                    // Slave register 1
                    slv_reg1[(byte_index*8) +: 8] <=
S_AXI_WDATA[(byte_index*8) +: 8];
                  end

            ......

              default : begin
                          slv_reg0 <= slv_reg0;
                          slv_reg1 <= slv_reg1;
                          slv_reg2 <= slv_reg2;
                          slv_reg3 <= slv_reg3;
                        end
            endcase
          end
      end
    end
```

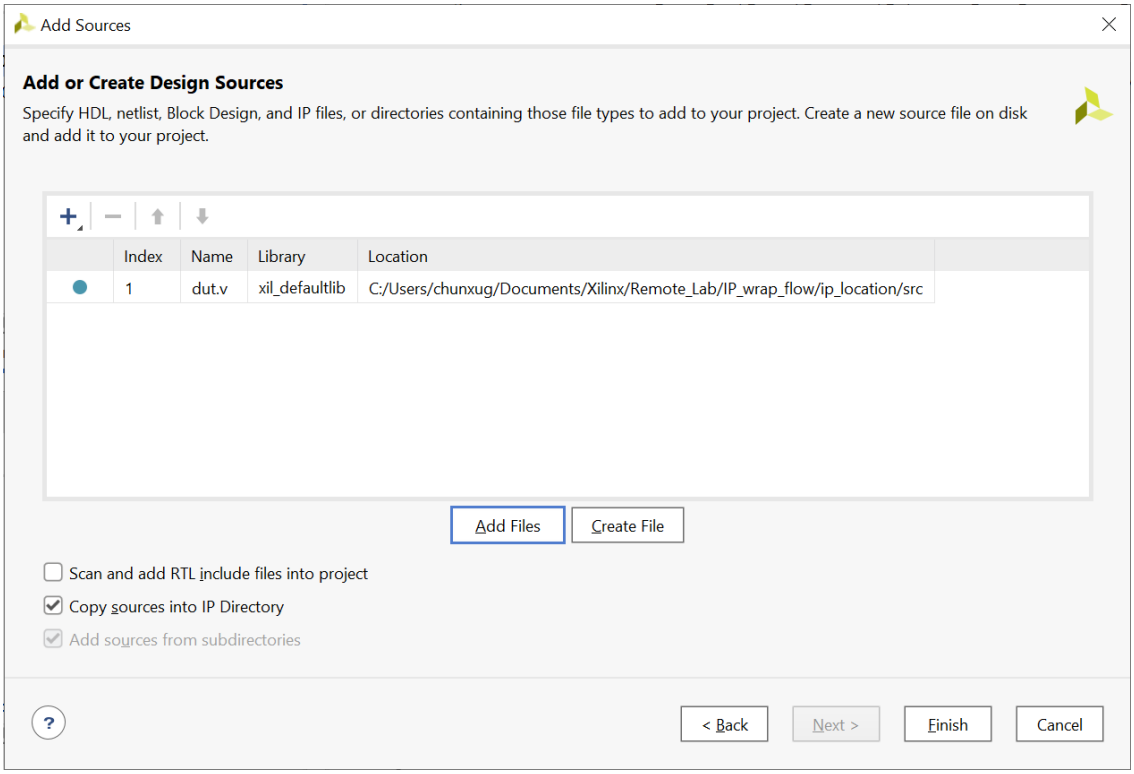
2.3 读取寄存器

1. 第365到379行包含了由PS向PL AXI Lite读寄存器的过程，`reg_data_out <= slv_regN`语句实现了从寄存器读取数据到**S_AXI_RDATA**的功能，这一部分是我们经常需要修改的。

```
    assign slv_reg_rden = axi_arready & S_AXI_ARVALID & ~axi_rvalid;
    always @(*)
    begin
          // Address decoding for reading registers
          case ( axi_araddr[ADDR_LSB+OPT_MEM_ADDR_BITS:ADDR_LSB] )
            2'h0   : reg_data_out <= slv_reg0;
            2'h1   : reg_data_out <= slv_reg1;
            2'h2   : reg_data_out <= slv_reg2;
            2'h3   : reg_data_out <= slv_reg3;
            default : reg_data_out <= 0;
          endcase
    end
```

2.4 加入Adder设计

1. 点击左上角**Sources**栏中的加号键，选择**Add or create design sources**并选择**Next**，在弹出界面中添加需要测试的文件**dut.v**并选择**Finish**。



2. 添加的**dut.v**就是我们要测试的简单IP，其实现了一个最简单的组合逻辑加法器，输入和输出都是32位无符号整型：

```
module dut(
    input wire [31:0] input1,
    input wire [31:0] input2,
    output wire [31:0] sum
);

assign sum = input1 + input2;

endmodule
```

3. 在**axilite_adder_v1_0_S00_AXI_inst.v**第116行左右添加sum信号用于连接dut的输出结果：

```
109     reg [C_S_AXI_DATA_WIDTH-1:0]    slv_reg2;
110     reg [C_S_AXI_DATA_WIDTH-1:0]    slv_reg3;
111     wire     slv_reg_rden;
112     wire     slv_reg_wren;
113     reg [C_S_AXI_DATA_WIDTH-1:0]     reg_data_out;
114     integer  byte_index;
115     reg  aw_en;
116     wire [C_S_AXI_DATA_WIDTH-1:0]   sum;
117
118     // I/O Connections assignments
```

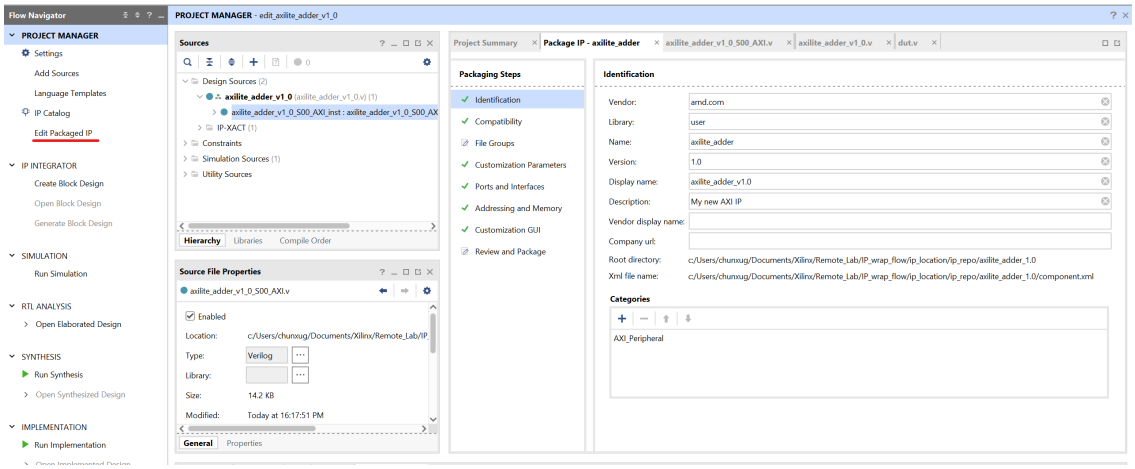
4. 在**axilite_adder_v1_0_S00_AXI_inst.v**第377行左右修改赋值语句为`2'h3    : reg_data_out <= sum`;用于将dut模块的输出结果赋值给**S_AXI_RDATA**：

```
372             // Address decoding for reading registers
373             case ( axi_araddr[ADDR_LSB+OPT_MEM_ADDR_BITS:ADDR_LSB] )
374               2'h0   : reg_data_out <= slv_reg0;
375               2'h1   : reg_data_out <= slv_reg1;
376               2'h2   : reg_data_out <= slv_reg2;
377               2'h3   : reg_data_out <= sum;
378               default : reg_data_out <= 0;
379             endcase
380       end
```

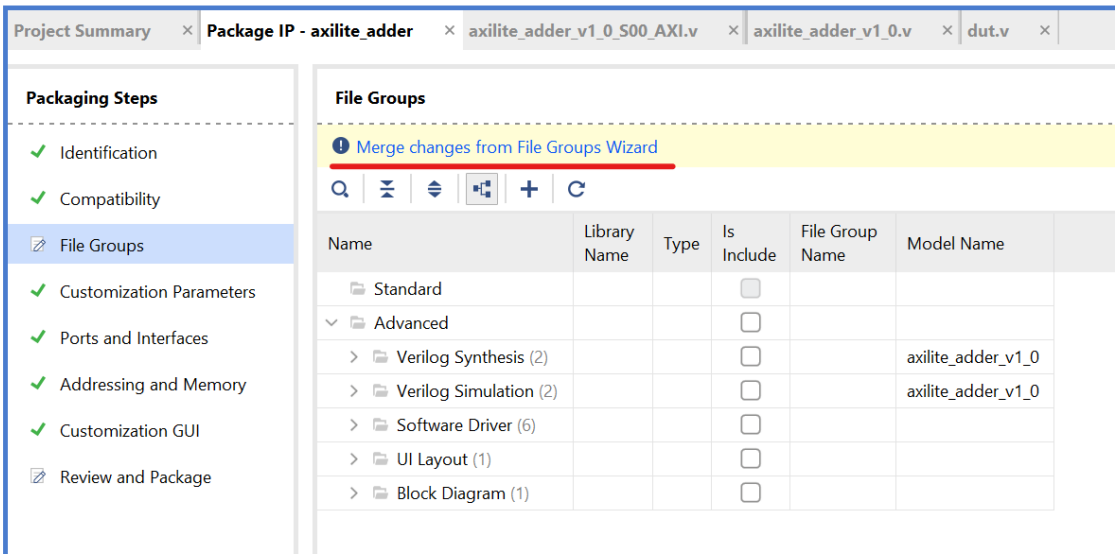
5. 在**axilite_adder_v1_0_S00_AXI_inst.v**第402行左右添加dut模块的实例化语句，输入为slv_reg0和slv_reg1，输出为sum：

```
398         end
399     end
400
401     // Add user logic here
402     dut dut_0(
403         .input1(slv_reg0),
404         .input2(slv_reg1),
405         .sum(sum)
406     );
407
408     // User logic ends
409
410     endmodule
```

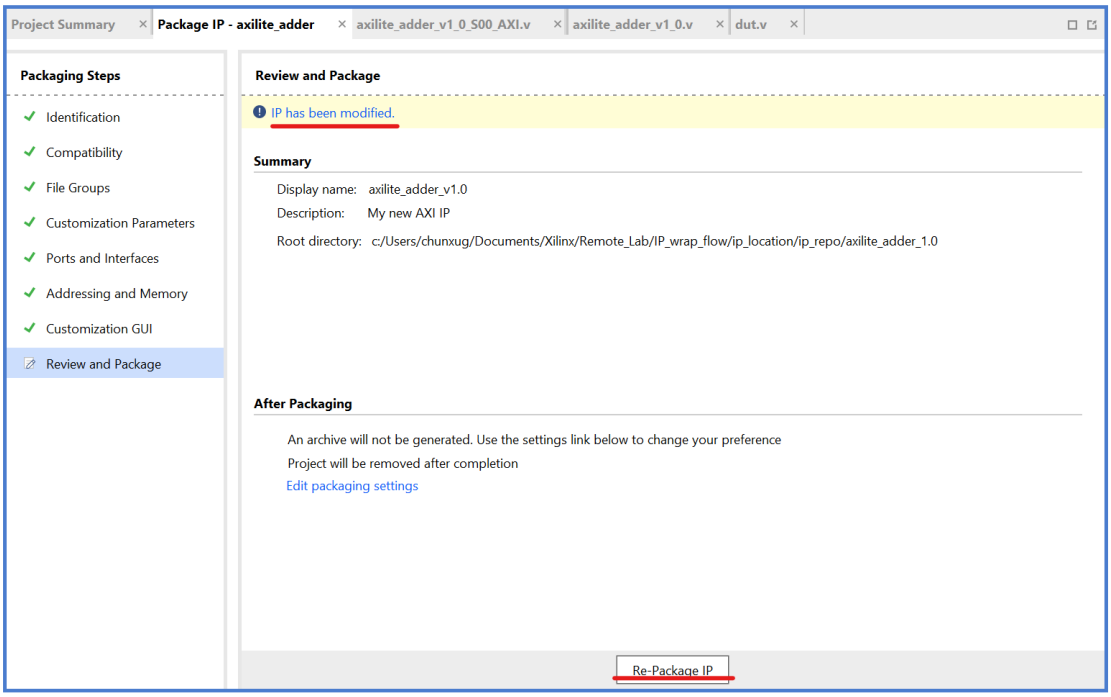
6. 点击最左侧列导航栏的**Edit Packaged IP**选项，弹出**Package IP**窗口：



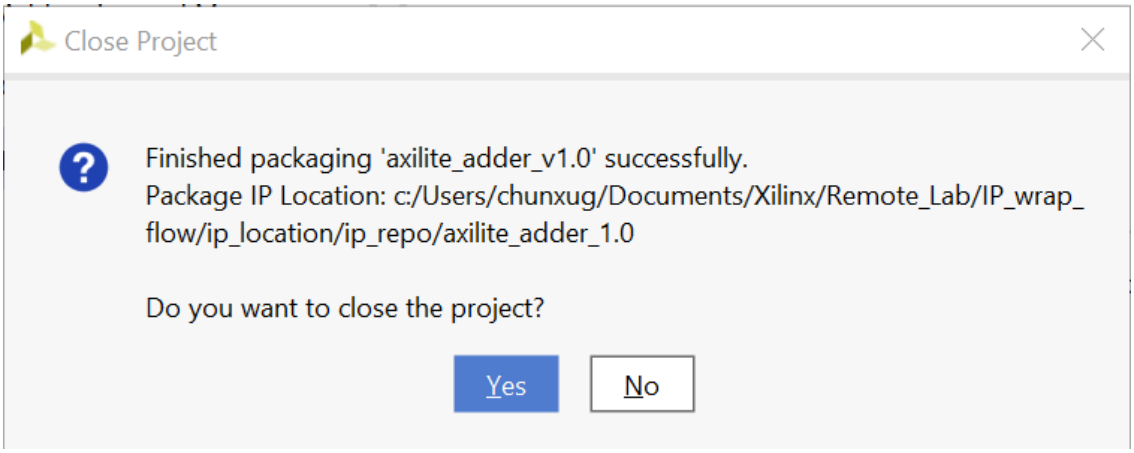
7. 在**Package IP**窗口中点击**File Groups**，再点击蓝色的**Merge changes from File Groups Wizard**选项



8. 在**Package IP**窗口中点击**Review and Package**，再点击蓝色的**IP has been modified**选项，之后点击最下方的**Re-Package IP**选项。



9. 提示完成**Package**操作，可以选择**No**先不关闭该窗口。



3. 在Vivado中进行IP集成

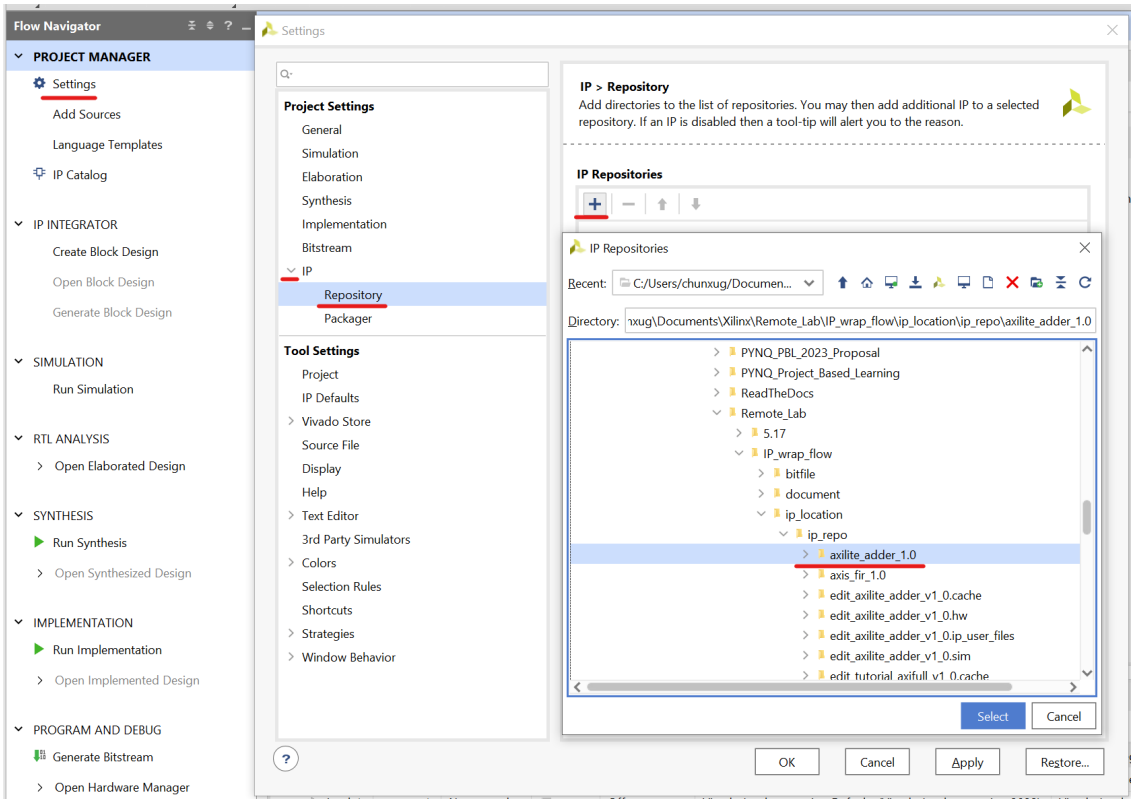
> 该部分内容与该教程第二部分“在Vivado中进行IP集成”完全一致，在此我们略去重复图片，读者如有疑问或对该部分GUI不熟悉，可参考原教程对应部分、有详细的逐步图片展示。

3.1 创建一个新Vivado项目

1. 打开**Vivado**软件，点击**Create Project**，创建一个新的项目，点击**Next**
2. 在**Project name**输入项目名**axilite_adder_system**，点击右侧的 **...** 按键选择一个合适的目录位置，点击**Next**
3. 进入**Project Type**界面，勾选上**Do not specify sources at this time**，再点击**Next**
4. 进入**Default Part**界面，在**Search**栏中搜索**xc7z020clg484-1**，将其选中，再点击**Next**
5. 点击**Finish**完成项目创建

3.2 导入IP

1. 我们需要首先将刚封装完的AXI-Lite IP导入到**Vivado**中，点击左侧窗口**Flow Navigator**中的**Settings** 选项，弹出**Settings**窗口
2. 将左侧的**Project Settings**中展开**IP**栏目，选中**Repository**项，点击右侧面板中的 **+** 按键，在弹出窗口中选择我们刚才封装的IP **axilite_adder**，再点击**Select**

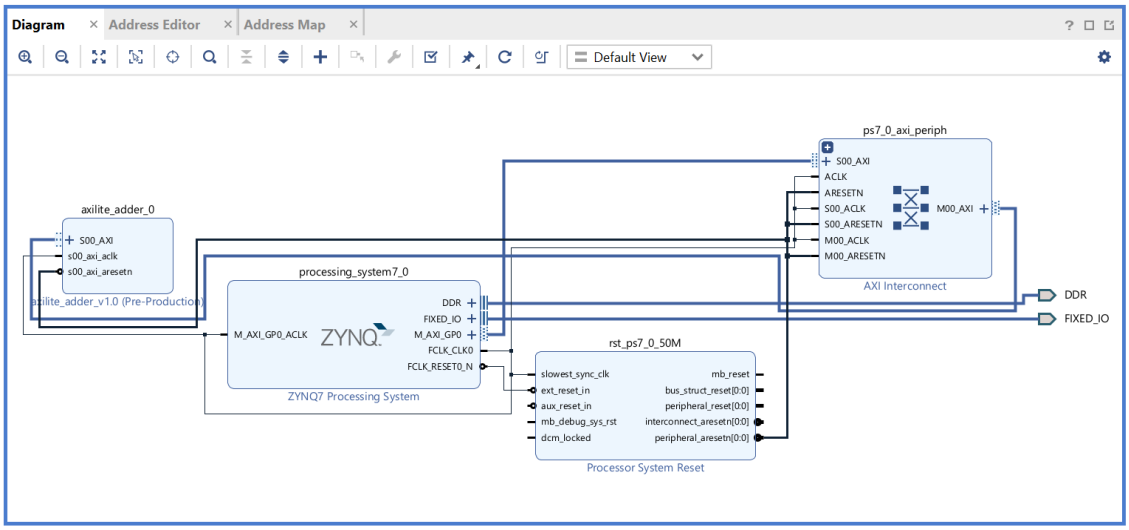


3. 可以看到对应的IP已经被成功添加到了工程中，在两个窗口中依次单击**OK**来关闭这些窗口

3.3 创建Block Design

1. 下面我们创建一个**Block Design**，利用**Vivado**的IP集成功能来构建完整系统。在左侧的**Flow Navigator**中点击**IP INTEGRATOR > Create Block Design**，在弹出的**Create Block Design** 窗口中保持各选项不变，设计名称使用默认的**design_1**，点击**OK**创建**Block Design**
2. 在出现的**Diagram**窗口中点击上方的 **+** 按钮，会弹出一个搜索框，在输入栏中键入**zynq**，双击备选项中出现的**ZYNQ7 Processing System**，即可将该IP添加到设计中
3. 在窗口上方会出现蓝色下划线提示**Run Block Automation**, 单击该区域弹出对应窗口，我们保持默认设置不变，直接点击**OK**

4. 点击Diagam窗口上方的 + 按钮，搜索**axilite_adder**，可以看到我们刚才导入的IP已经可以使用了，双击**axilite_adder**以将其添加到设计中
5. 下面我们对设计进行自动连线。点击窗口上方的蓝色下划线提示**Run Connection Automation**，弹出对应窗口，将左侧**All Automation** 选项勾选上，再点击**OK**
6. 系统将根据对应接口自动进行连线，我们可以得到如下图的设计



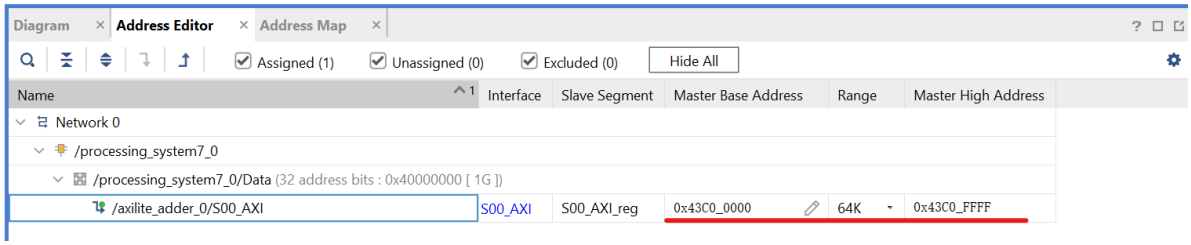
7. 在**Diagram**上侧的工具栏中点击勾形图标**Validate Design**，对设计进行验证
8. 在左侧的**Source > Design Sources > design_1**选项上右键，选择**Generate Output Products**
9. 在弹出窗口中保持各配置不变，点击**Generate**，这一过程将耗费约1分钟的时间
10. 在左侧的**Source > Design Sources > design_1**选项上右键，选择**Create HDL Wrapper**，在弹出窗口中保持选项不变并点击**OK**，完成后可以看到在**design_1.bd**上层嵌套了一层**design_1_wrapper.v**文件

3.4 综合与生成比特流

1. 在左侧的**Flow Navigator**中选择**Run Synthesis**，在弹出窗口中保持选择不变并选择**OK**
2. 综合完成后，会弹出**Synthesis Completed**窗口，在**Next**栏中保持默认的**Run Implementation**选项，并点击**OK**，如果出现新弹窗，同样保持默认选项并点击**OK**即可
3. **Implementation**结束后，会弹出**Implementation Completed**窗口，在**Next**栏中选择**Generate Bitstream**选项，并点击**OK**，如果出现新弹窗，同样保持默认选项并点击**OK**即可
4. 比特流生成后，会弹出**Bitstream Genreation Completed**窗口，我们直接点击**Cancel**即可
5. 至此，我们已经完成了硬件部分的设计与导出

3.5 地址分配

1. 在连接完上述设计后，我们可以点开**Diagram**窗口旁边的**Address Editor**窗口来查看Vivado为我们的IP分配的地址，可以看到Offset为0x43C0_0000，如果我们后续在PYNQ中使用**MMIO**进行寄存器读写的话就会需要知道这些值。



4. PYNQ框架中进行测试

4.1 提取bit与hwh文件

1. 在文件管理器中访问 **\axilite_adder_system\axilite_adder_system.runs\impl_1** 目录，该目录下的**design_1_wrapper.bit**文件即为生成的比特流文件，将其复制到自己的文件夹中保存，并重命名为**adder.bit**

2. 在文件管理器中访问 **\axilite_adder_system\axilite_adder_system.gen\sources_1\bd\design_1\hw_handoff** 目录，其中的**design_1.hwh**即为我们需要的**hardware handoff**文件，将其复制到自己的文件夹中保存，并重命名为**adder.hwh**

4.2 访问Jupyter

1. 请先完成PYNQ远程实验室的账号注册与**Jupyter**访问
2. 登录**Jupyter**界面，点击界面右上方的**upload**按钮，将以下文件上传到开发板上
    - **/jupyter** 目录下的**adder.ipynb**
    - 上一步中得到的**adder.bit**与**adder.hwh**文件
        - 如果你在前面操作中导出失败了，你也可以先使用 **/overlay** 目录下的**adder.bit**与**adder.hwh**文件上传，以完成余下实验

4.3 部署与运行Overlay

1. 在**Jupyter**中进入到**adder.ipynb**页面，**Kernel**自动加载完成显示为**Python3**字样
2. 点击窗口上侧的**Run**按钮，**Jupyter Notebook**会执行当前**Cell**，同时自动切换到下一个**Cell**
3. 完成按照顺序依次点击**Run**至结束即可，各代码块的含义在**Jupyter Notebook**中已经标注，请阅读**Jupyter Notebook**中的信息继续完成实验。