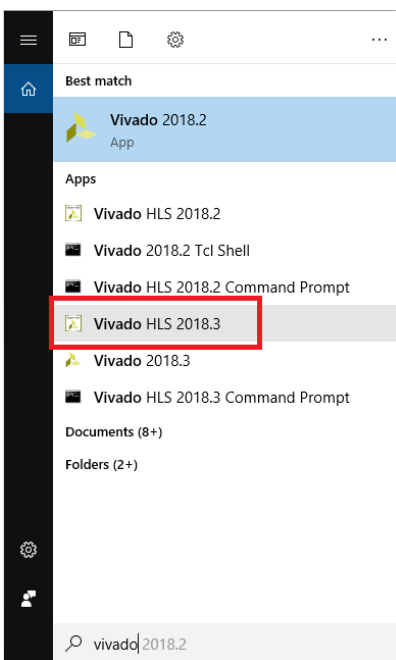


## , PYNQ-HLS 动手实验手册

## 通过 HLS 工具快速生成用于图像边缘检测的硬件 IP

## 步骤 1

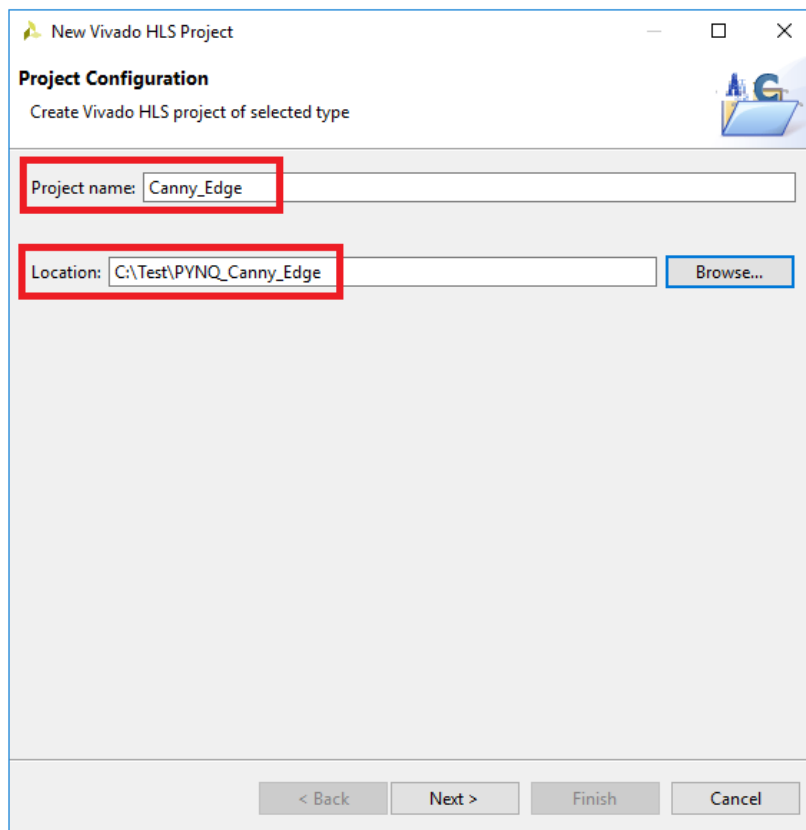
1. 打开 Vivado HLS 软件，以 2018.3 版本为例；



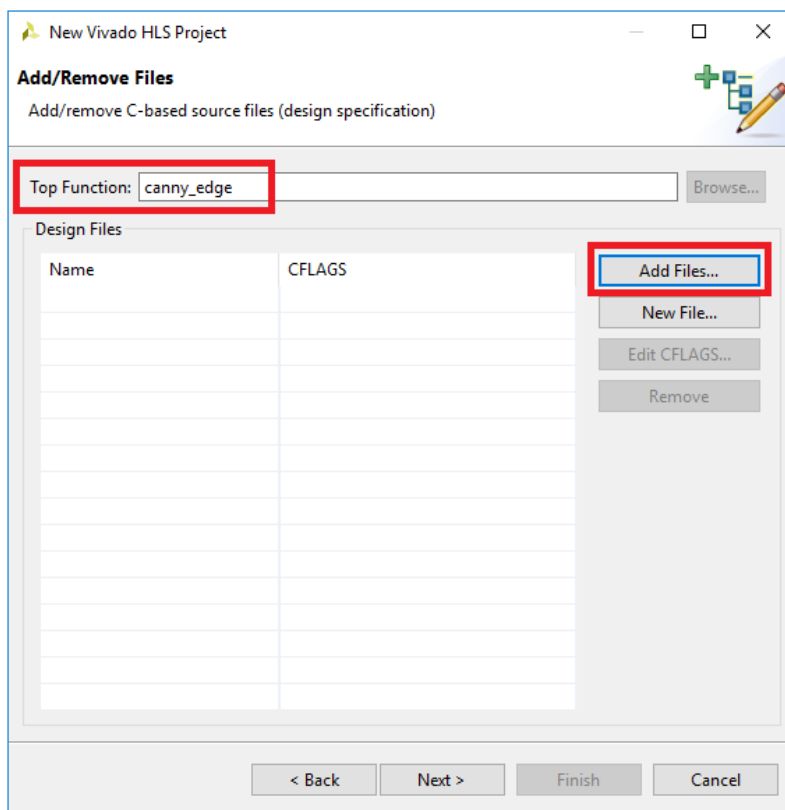
2. 点击 *Create New Project* 来创建一个新的 HLS 工程；



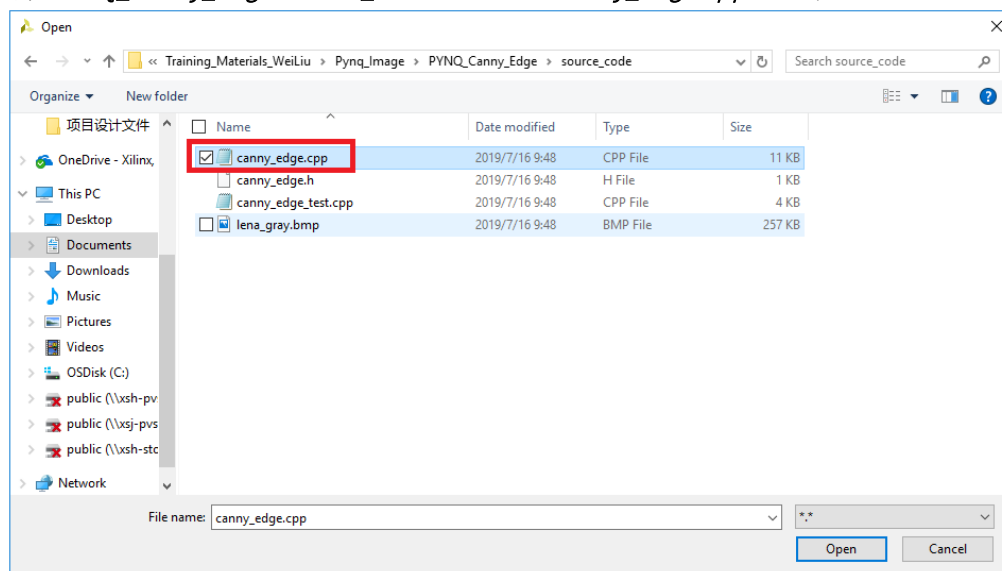
3. 为您的 HLS 工程设置存放目录和工程名称（注意：路径中不可以有中文字符或者特殊字符），点击 *Next*;

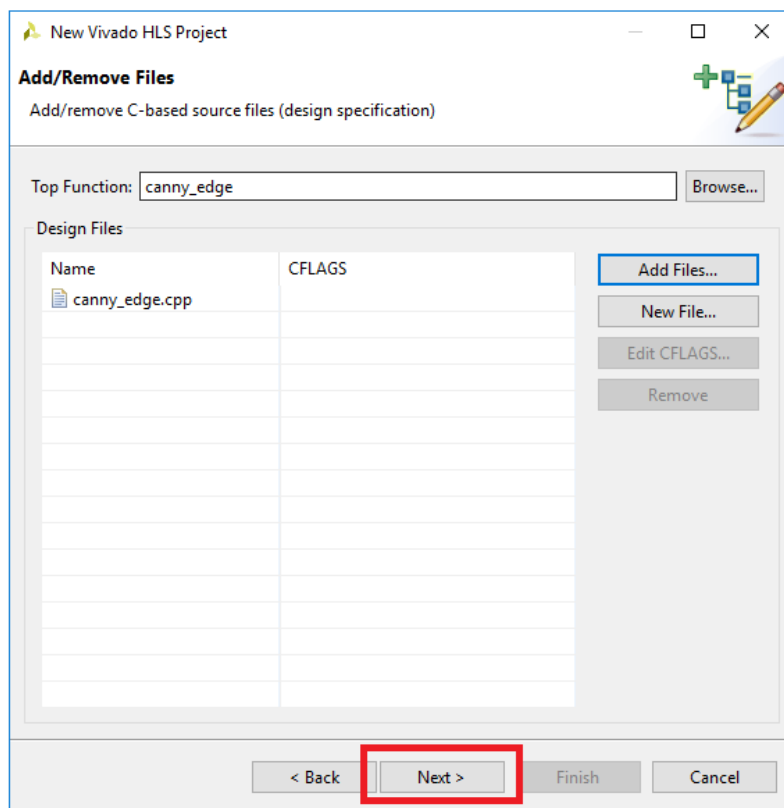


4. 设置 Top Function 名称为 *canny\_edge*，点击 *Add Files ...*;

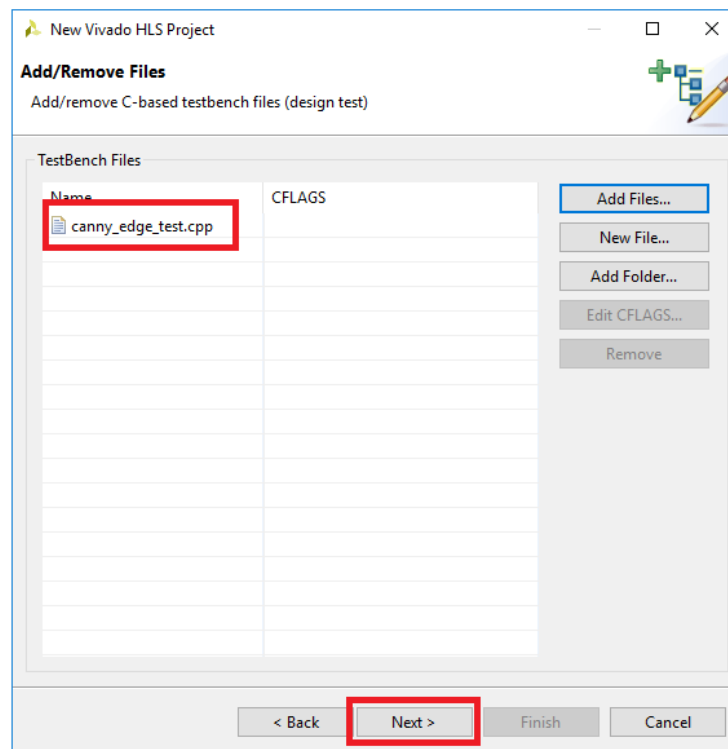
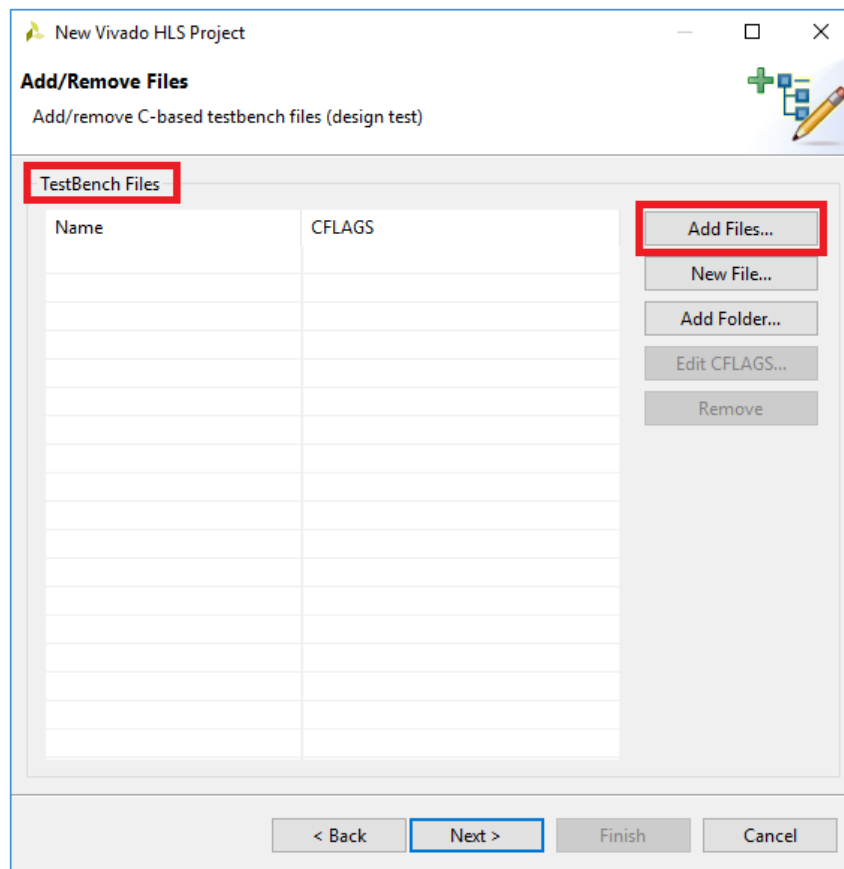



5. 添加预提供的源代码文件到工程中。  
选择 `PYNO_Canny_Edge\source_code` 目录下的 `canny_edge.cpp` 文件，然后点击 *Next*;

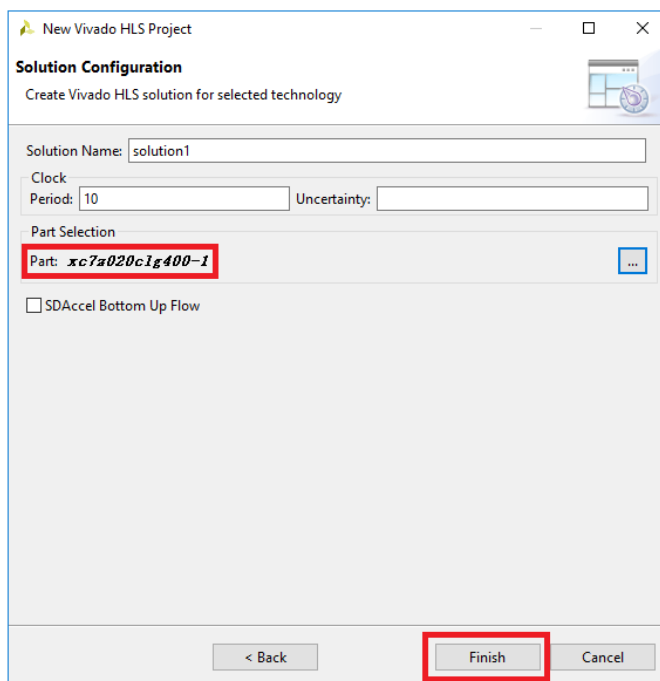
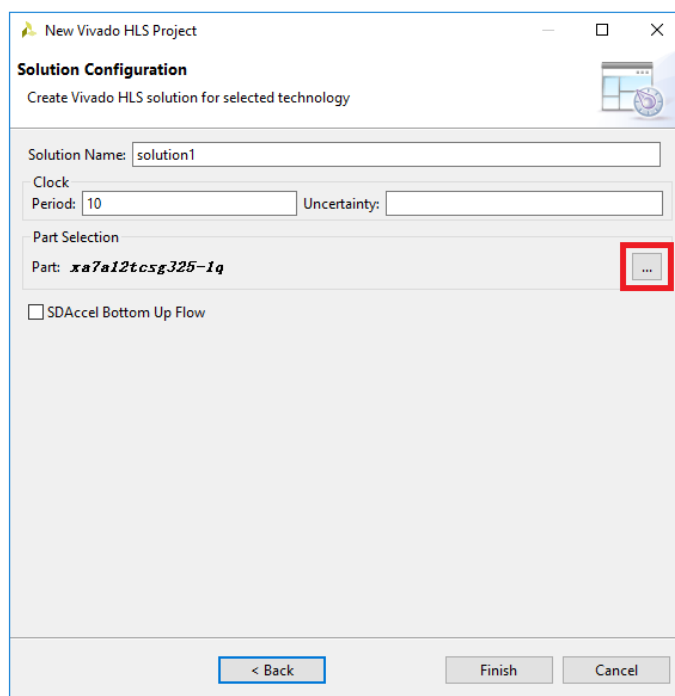




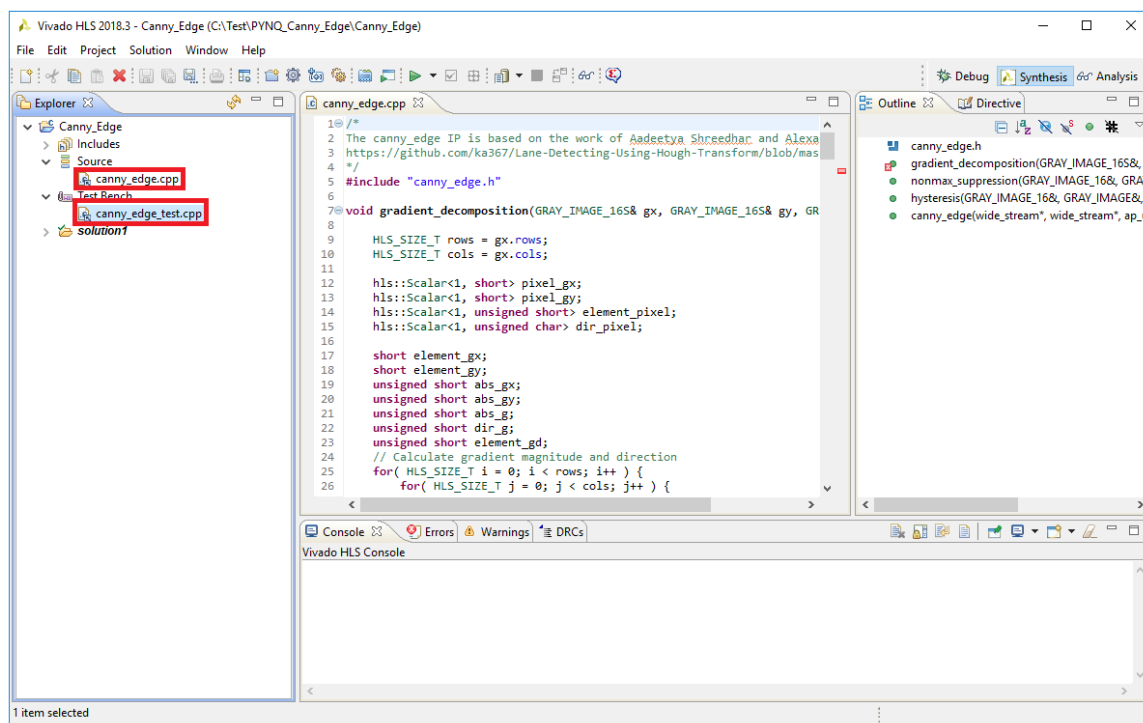
6. 添加预提供的 testbench 代码文件到工程中。  
点击 *Add Files...* 按钮，选择 *PYNQ\_Canny\_Edge\source\_code* 目录下的 *canny\_edge\_test.cpp* 文件，然后点击 *Next*;



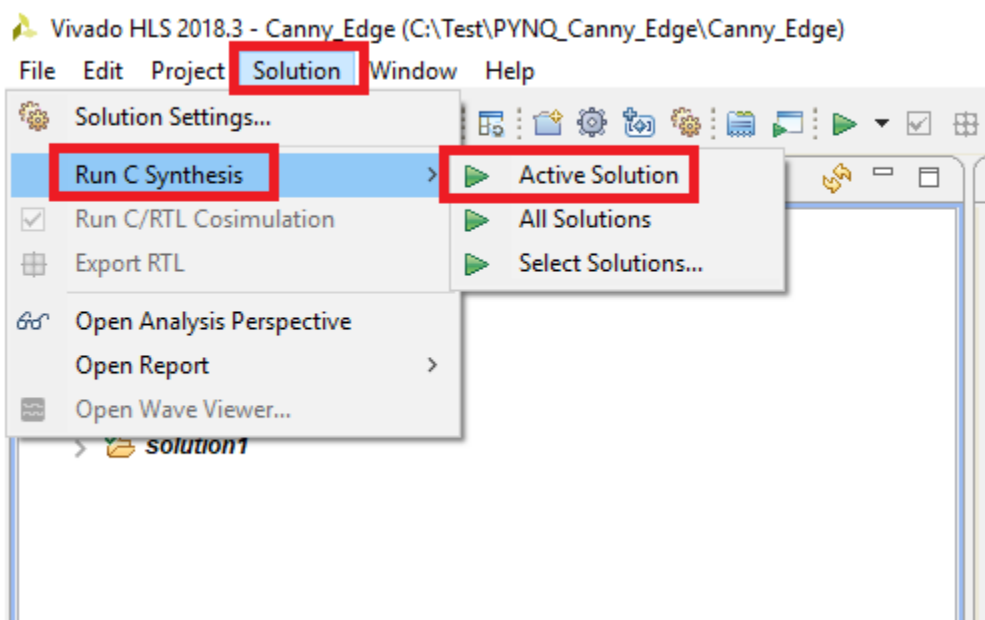
7. 修改 part 为实验使用的板卡器件类型。  
点击  按钮，选择 `PYNQ_Canny_Edge\source_code` 目录下的 `canny_edge_test.cpp` 文件，然后点击 *Finish*;



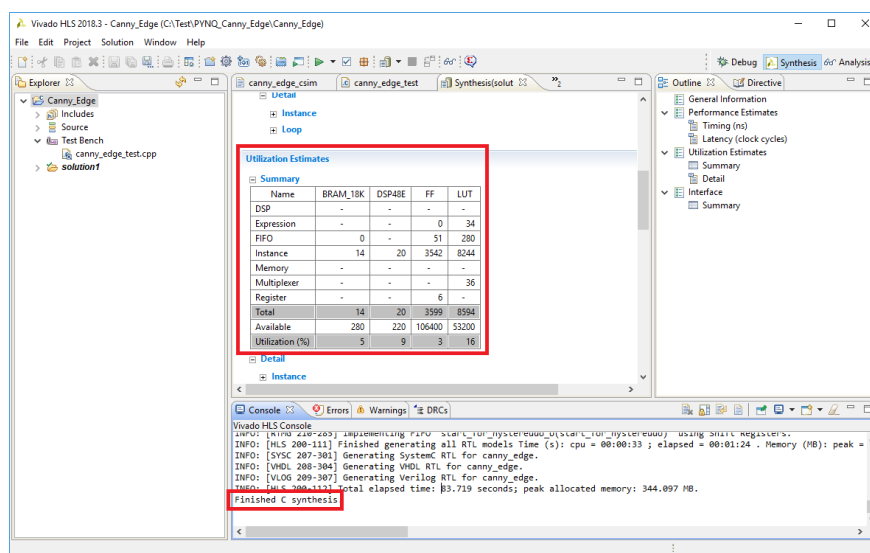
8. 浏览源代码及 testbench 源文件;



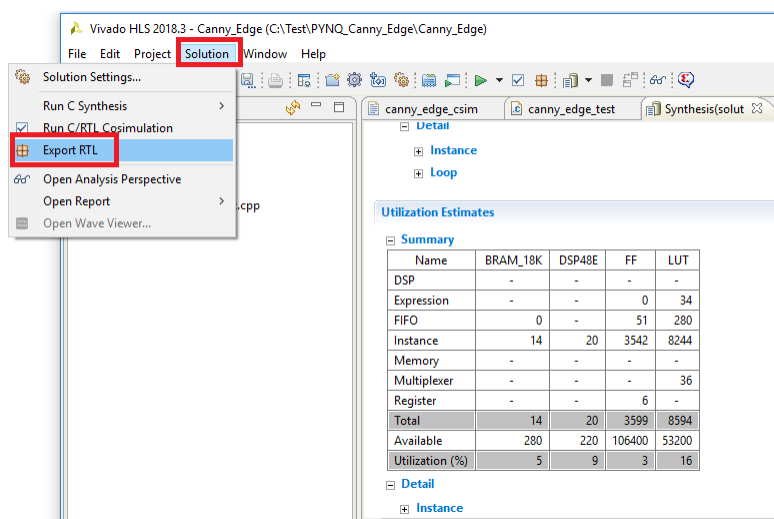
9. 点击 *Solution – Run C Synthesis – Active Solution* 对源代码进行综合；



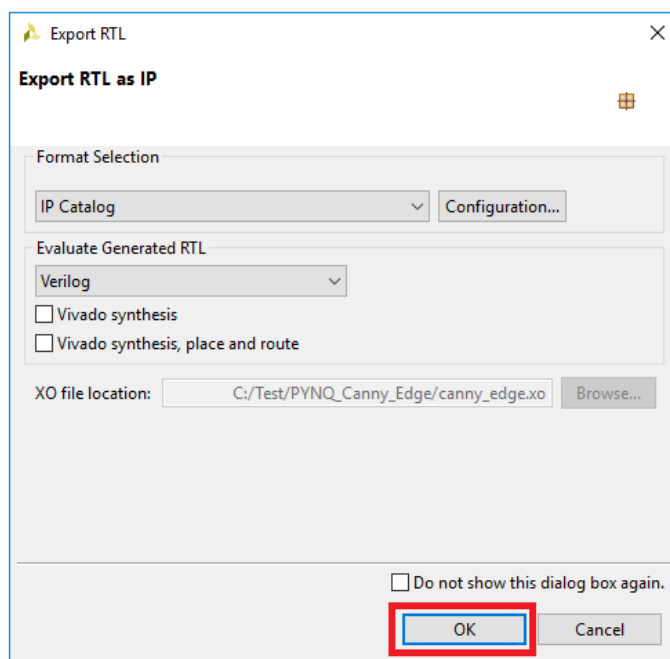
10. 等待综合完成，可查看综合后的资源利用情况；



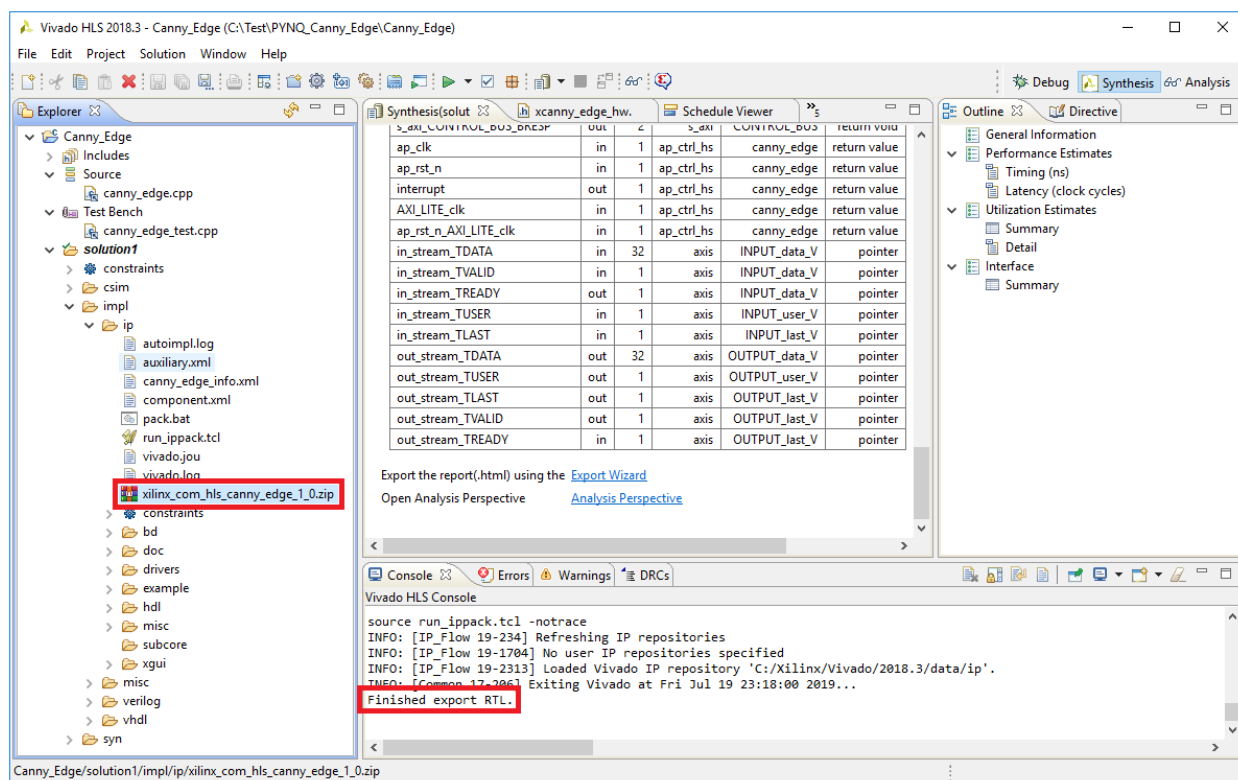
11. 点击 *Solution – Export RTL* 生成硬件 IP;







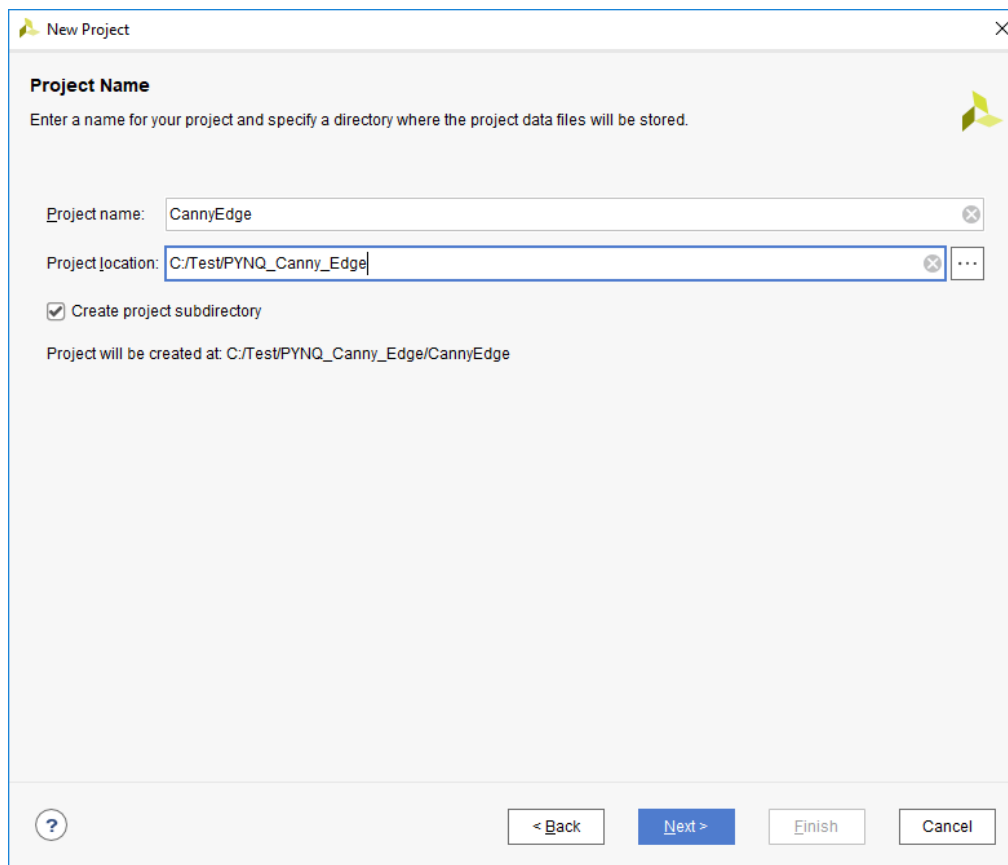
12. 该 zip 文件即为生成的硬件 IP。



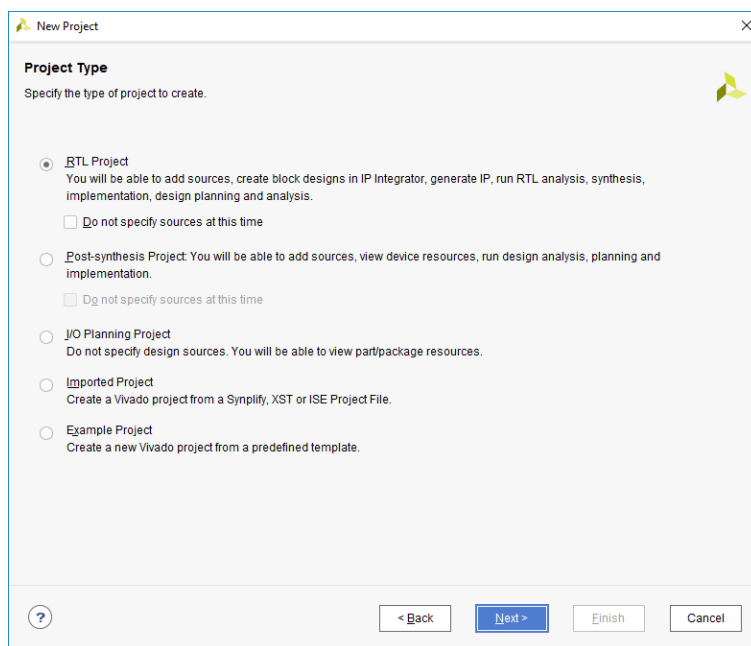
## 创建 Vivado 工程

## 步骤 2

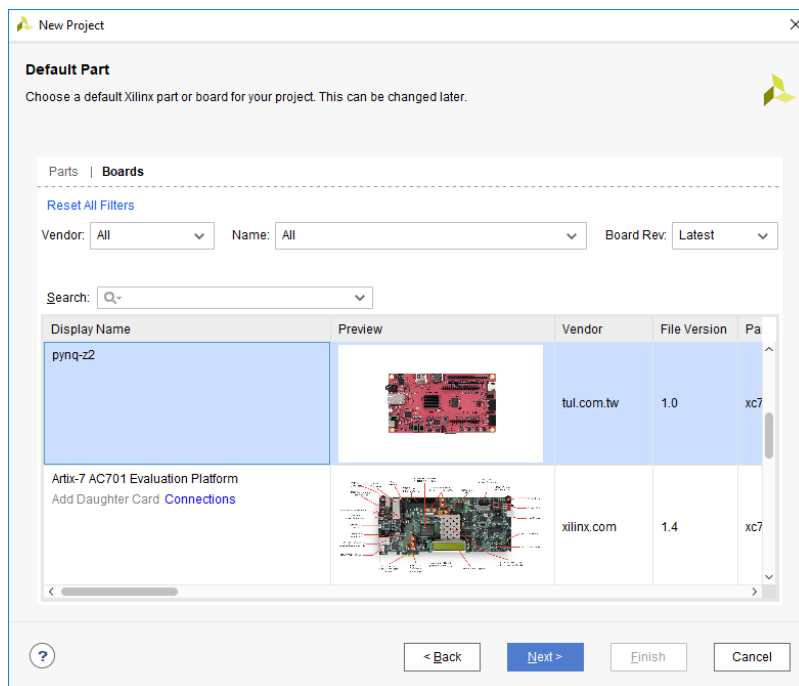
1. 启动 Vivado 工具: Start > Xilinx Design Tools > Vivado 2018.3;
2. 新建 Vivado 工程, 工程名称命名为 *CannyEdge*;



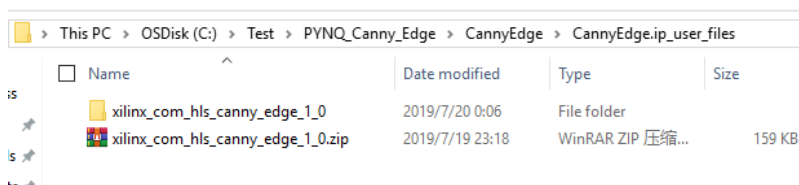
3. 一直点击 Next 按钮直到选择 Default Part 界面;



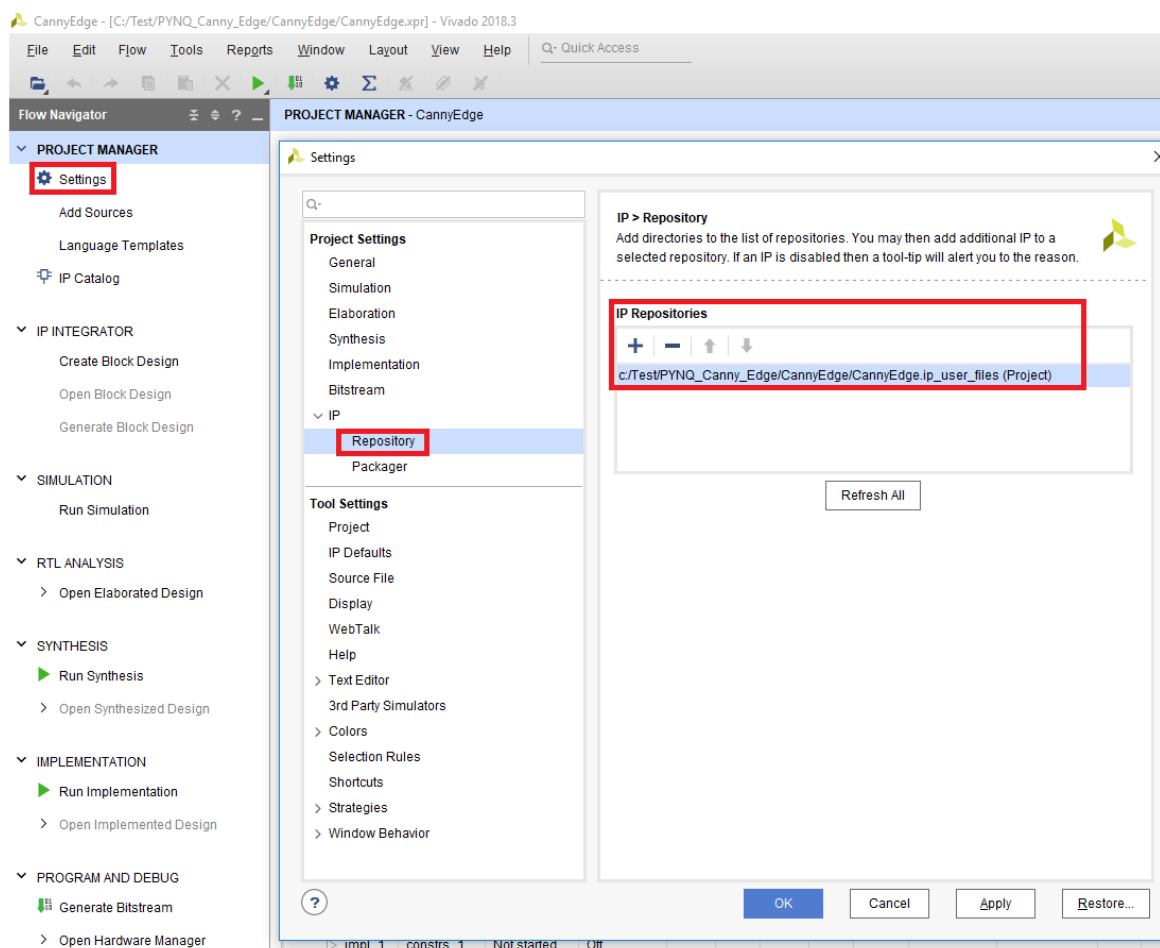
4. 选择 PYNQ-Z2 板, 完成工程创建;



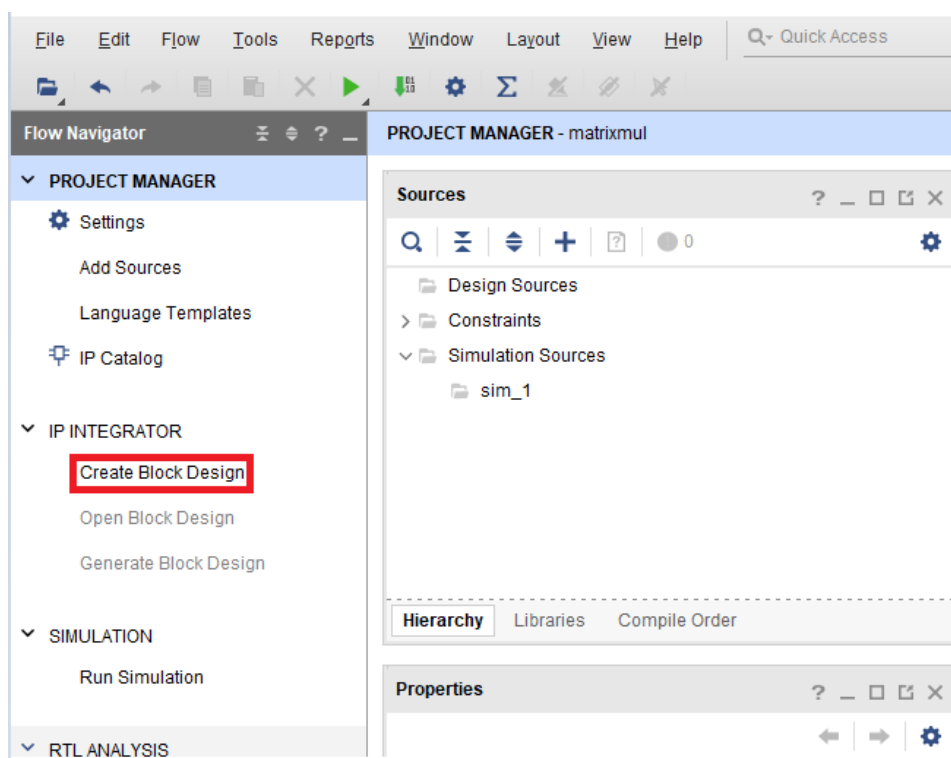
5. 将刚刚生成的 zip 包拷贝到 Vivado 工程的.ip.user\_files 后解压缩;



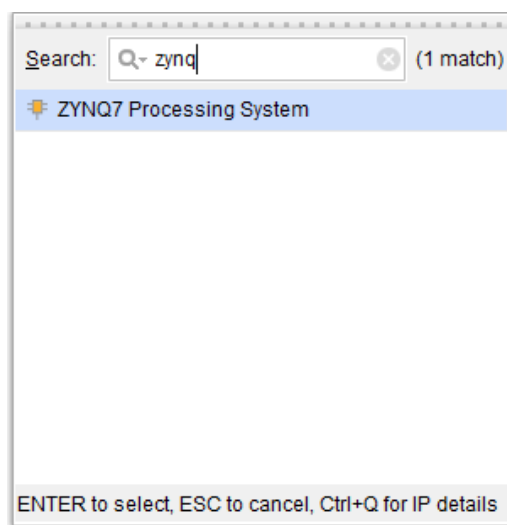
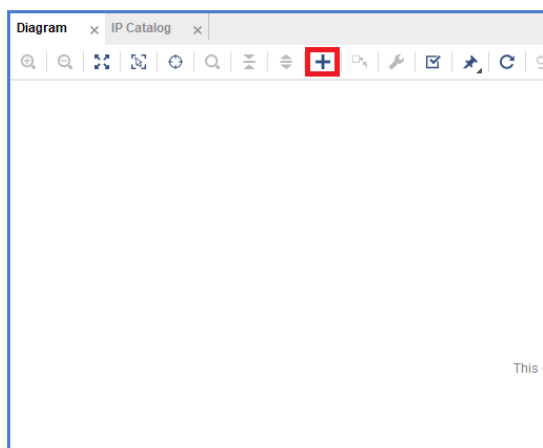
6. 将.ip.user\_files 文件夹添加到 ip 库;



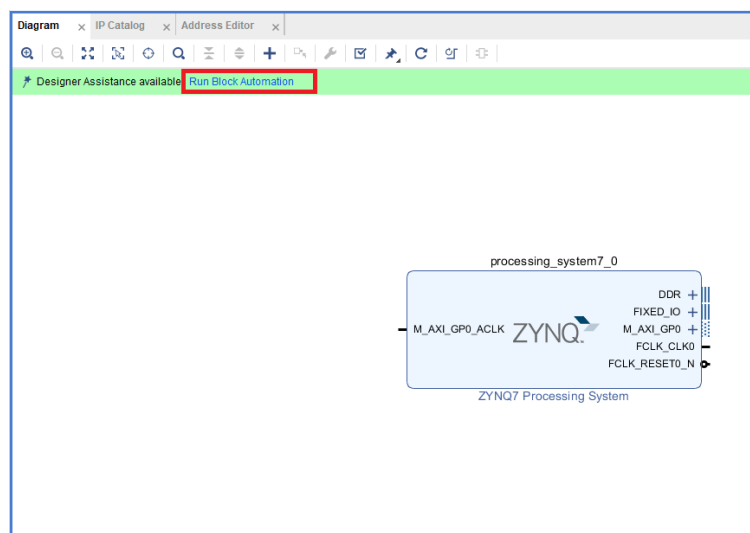
## 7. 创建 Block Design；



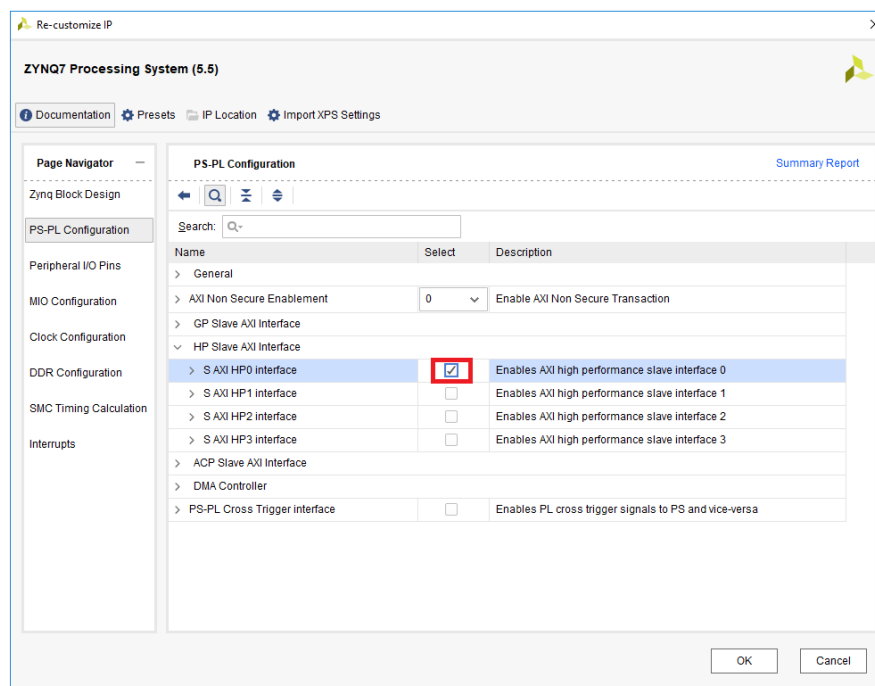
8. 添加 ZYNQ7 Processing System IP 到 Block Design;



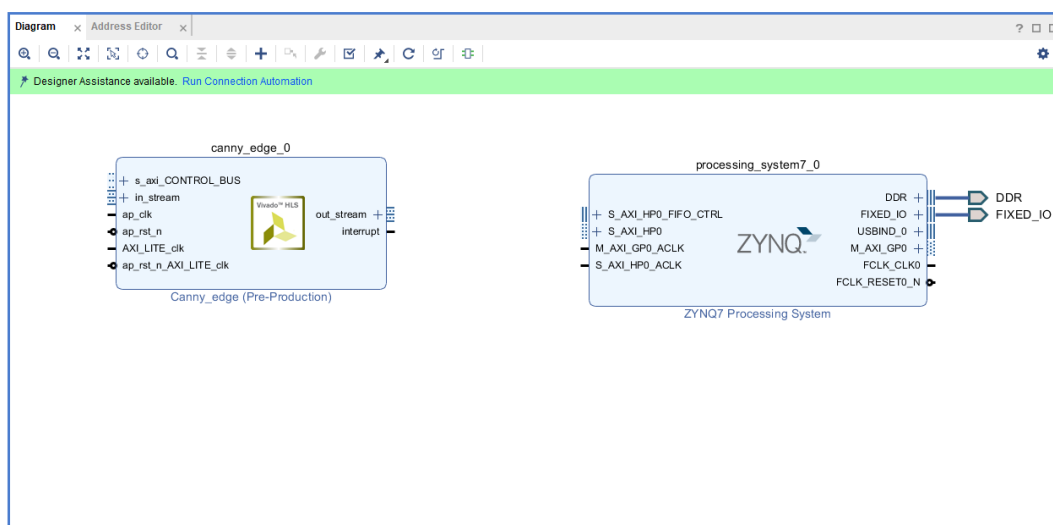
9. 点击 *Run Block Automation*, 保持默认配置, 点击 OK;



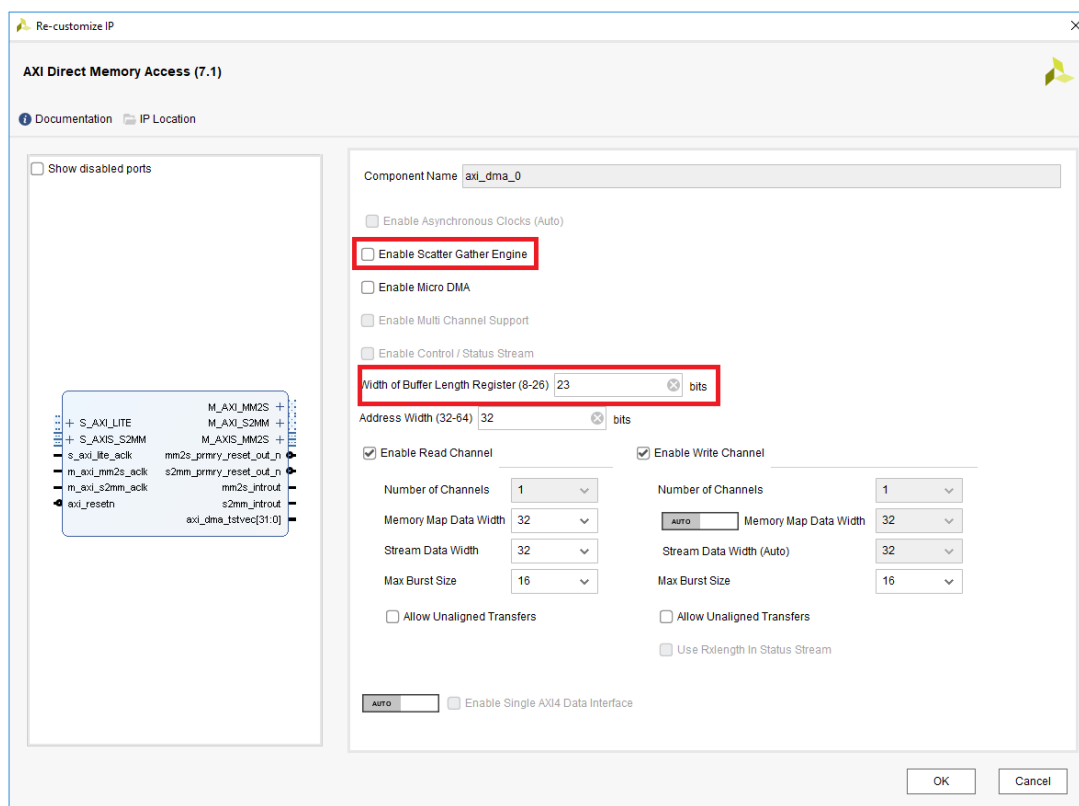
10. 为 Zynq Processing System 增加 HP0 口;



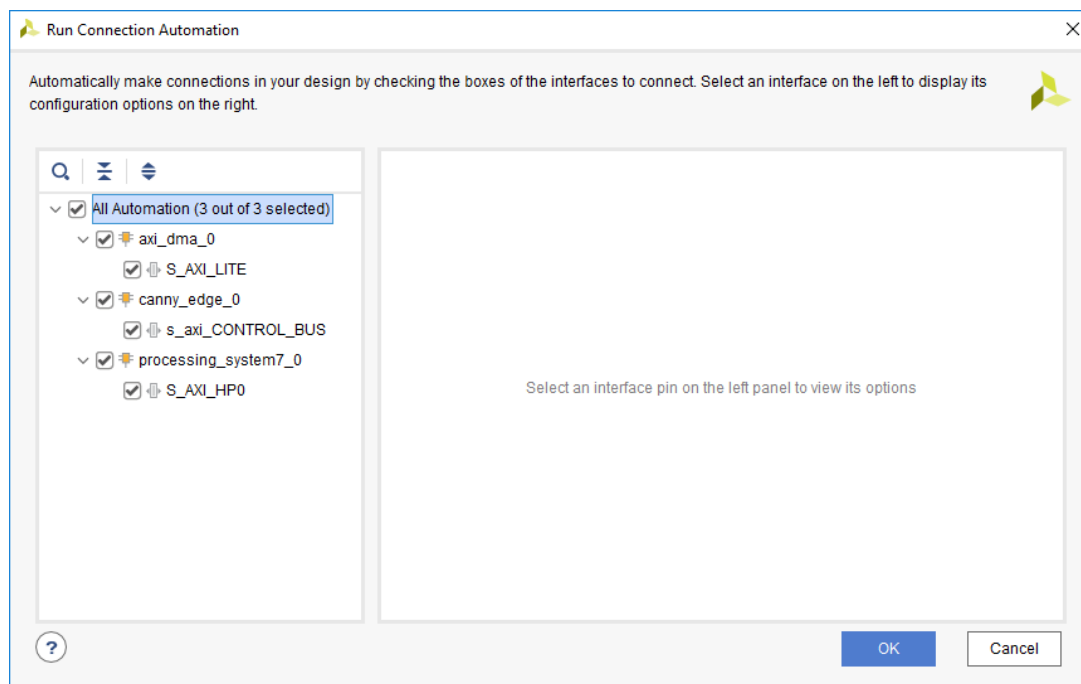
11. 例化刚刚 HLS 生成的 Canny\_Edge IP 到 Block Design;

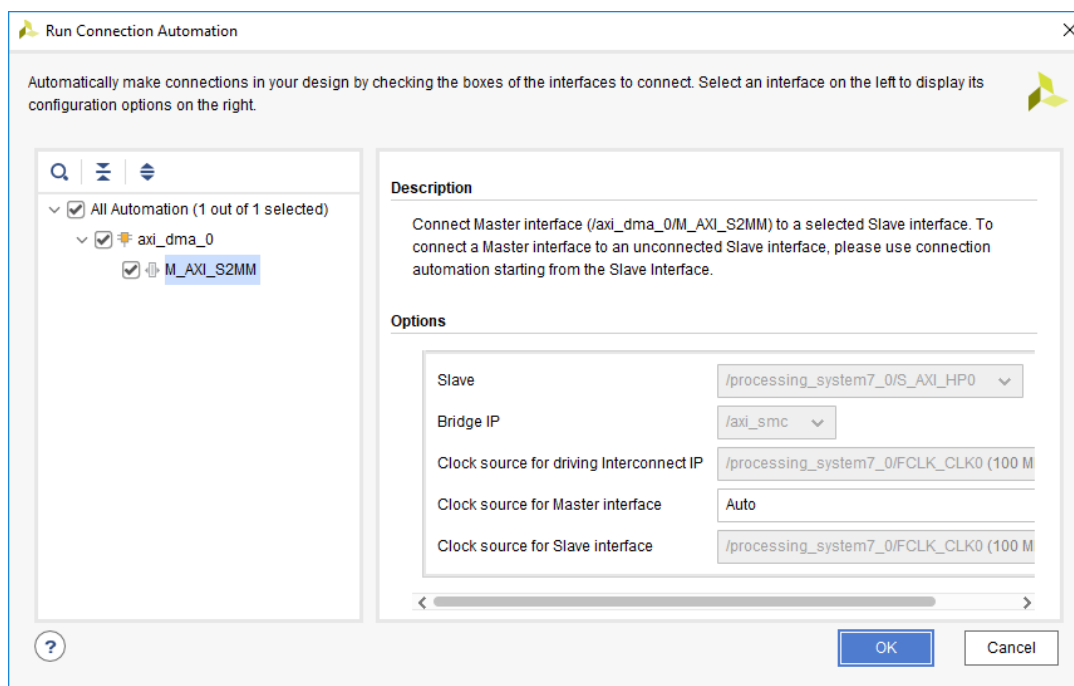


12. 例化 1 个 AXI DMA IP，并按照下图进行配置;

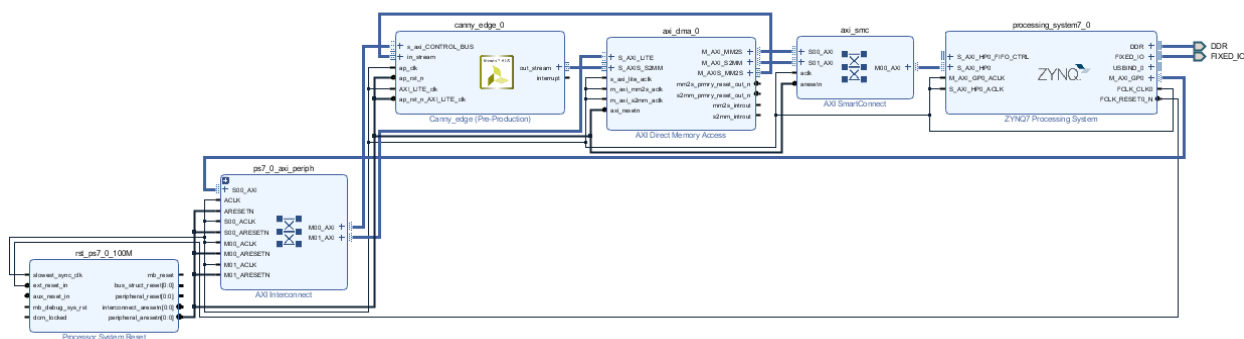


13. 点击 2 次 *Run Connection Automation*;



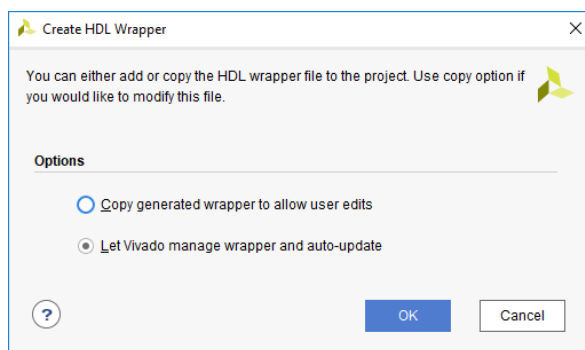
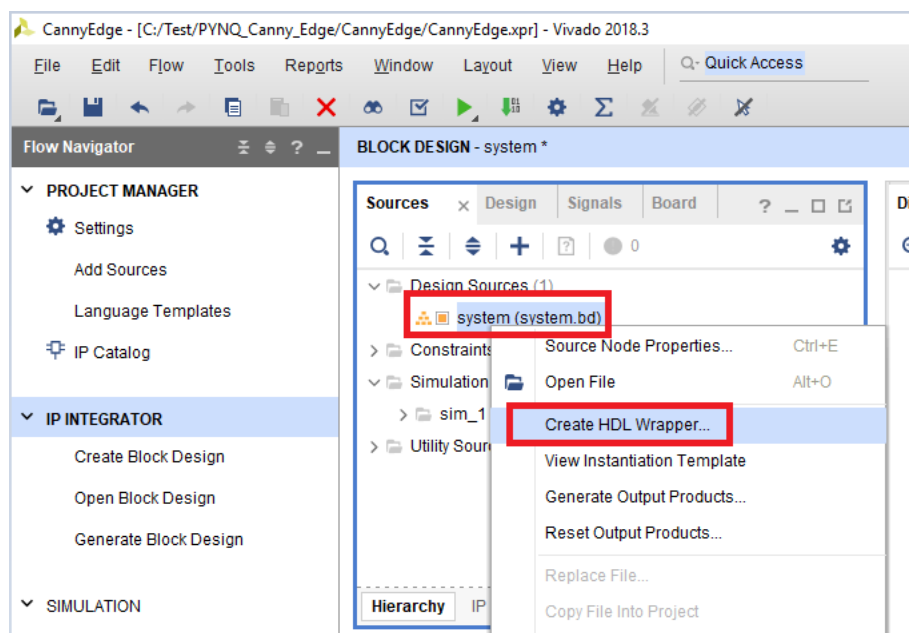


14. 手动连接 Canny\_Edge 与 AXI-DMA 之间的 AXI Stream 接口，最终的 Block Design 如下图所示；

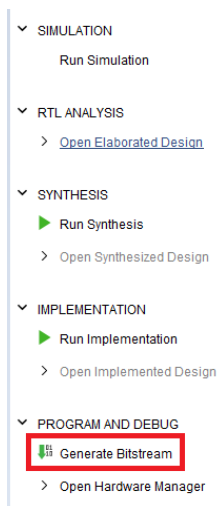


15. 创建 HDL Wrapper 文件；

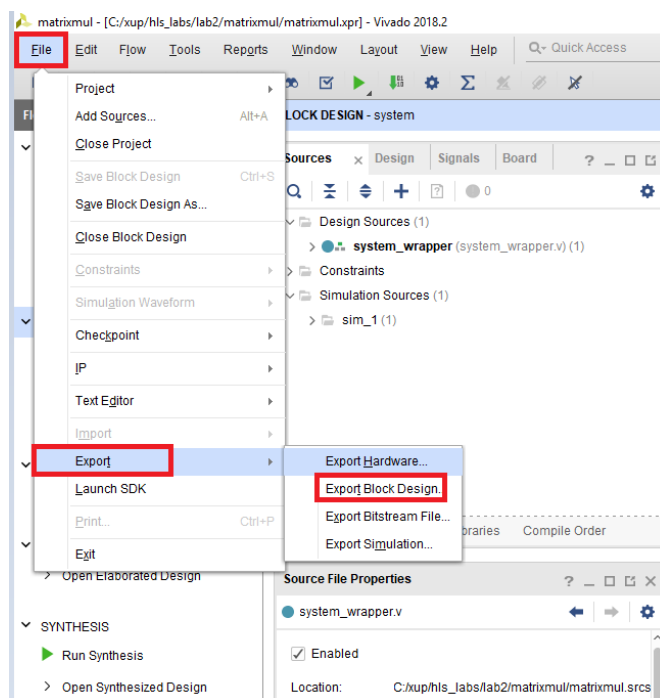




16. 生成 Bitstream 文件（根据电脑配置不同，可能会花）；



17. Export Block Design 的 tcl 文件；



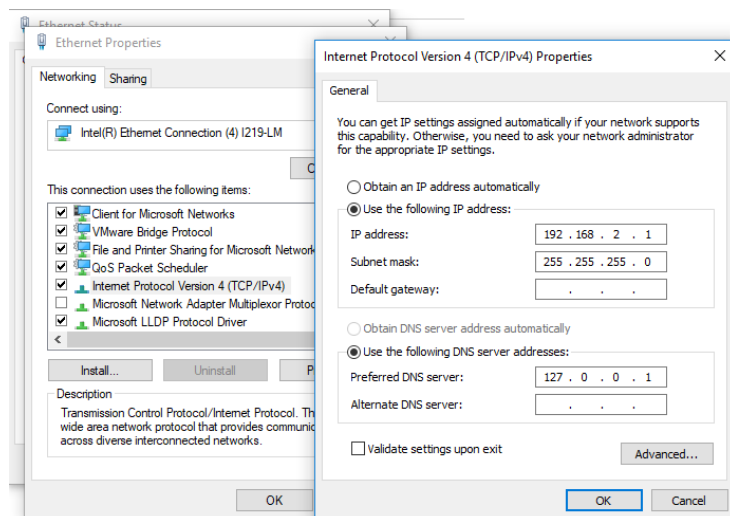
18. 将 tcl 文件和 bitstream 文件和 Vivado 过程\bd\system\hw\_handoff 目录内的.hwh 文件拷贝到 {PYNQ\_Canny\_Edge}\notebooks\CannyEdge\bitstream 目录，并将 tcl 文件和 bitstream 文件和 hwh 文件重命名为 CannyEdge.tcl，CannyEdge.bit 和 CannyEdge.hwh；

<< Training_Materials_WeiLiu > Pynq_Image > PYNQ_Canny_Edge > notebooks > CannyEdge > bitstream				
<input type="checkbox"/>	Name	Date modified	Type	Size
<input type="checkbox"/>	CannyEdge.bit	2019/7/20 0:49	BIT File	3,951 KB
<input type="checkbox"/>	CannyEdge.tcl	2019/7/20 0:49	TCL File	46 KB

## 连接 PYNQ-Z2 板卡测试

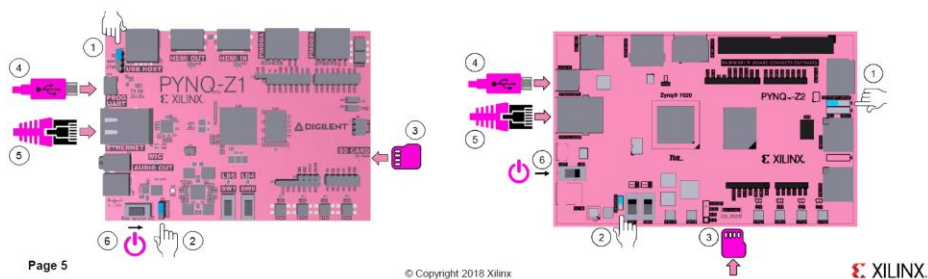
## 步骤 3

1. 将笔记本或者 PC 机的 IP 地址设置为 192.168.2.X

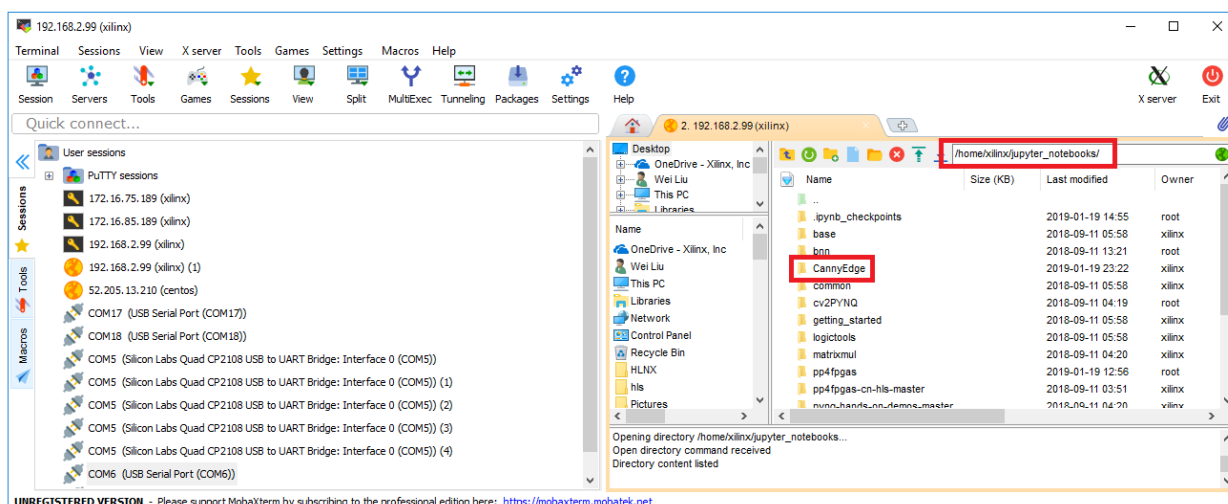
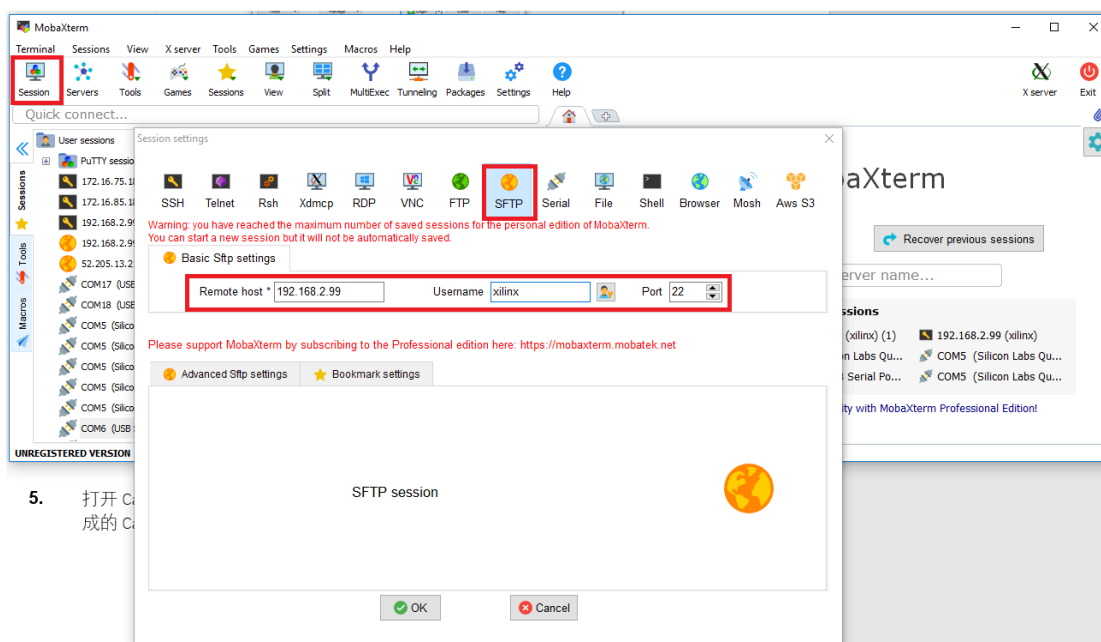


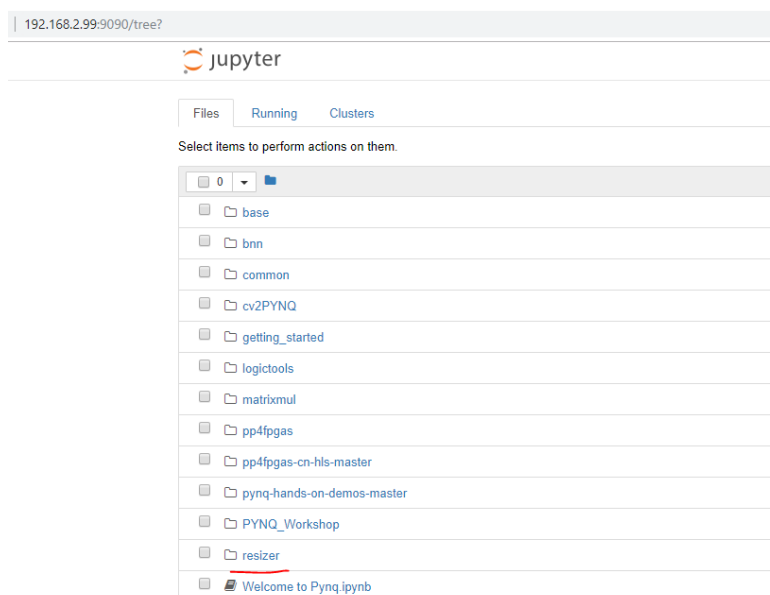
2. 按如下方式设置好 PYNQ-Z2，并将 PYNQ-Z2 通过网线连接到 PC 后，然后上电；

1. Configure board to boot from SD Card
2. Set Jumper to power from USB
3. Insert SD Card
4. Connect USB cable
5. Connect Ethernet cable to PC or to a Switch/Router
6. Power On



3. 待 PYNQ-Z2 启动完成后，通过 MobaXterm 等 sftp 工具将{PYNQ-Resizer}\notebooks 下的整个 CannyEdge 文件夹传到板卡的 jupyter\_notebooks 目录下；  
注:用户名与密码均为: xilinx





5. 打开 CannyEdge 目录，运行 CannyEdge\_PL.ipynb 和 CannyEdge\_PS.ipynb，查看 HLS 生成的 CannyEdge 硬件 IP 的运行效果。

192.168.2.99:9090/notebooks/CannyEdge/CannyEdge.ipynb

jupyter CannyEdge Last Checkpoint: 01/20/2019 (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

**Canny Edge in Programmable Logic (PL) - Application Notebook**

*This reference design illustrates how to run a Canny Edge IP on the Programmable Logic (PL) using Jupyter Notebooks and Python*

**Image Canny detection in Programmable Logic**

**Import libraries**

```
In [1]: from PIL import Image
import numpy as np
from IPython.display import display
from pynq import Xink
from pynq import Overlay
```

## Create an Image object using PIL in SW ¶

We will load image from the SD card and create a PIL Image object.

```
In [11]: image_path = "images/paris.jpg"
original_image = Image.open(image_path)
display(original_image)
```



```
In [17]: pic_width, pic_height = original_image.size
print("Image size: {}x{} pixels.".format(pic_width, pic_height))

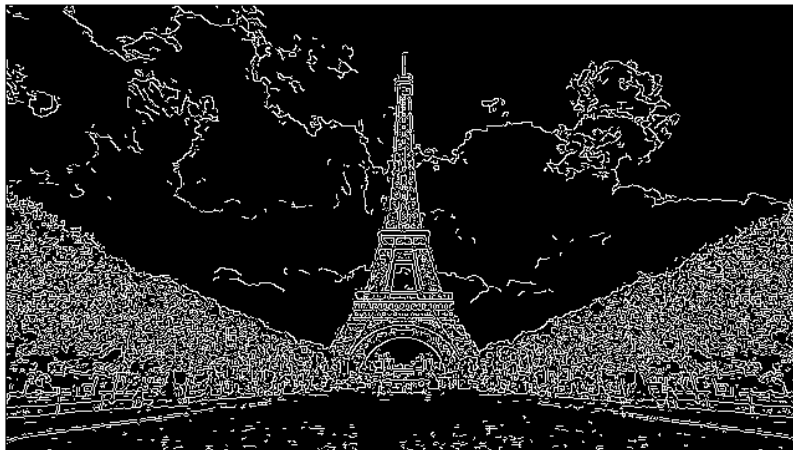
Image size: 640x360 pixels.
```

---

```
In [34]: dma.sendchannel.wait()
```

```
In [35]: dma.recvchannel.wait()
```

```
In [36]: result = Image.fromarray(out_buffer)
display(result)
print("Canny edged in Hardware(PL): {}x{} pixels.".format(pic_width, pic_height))
```



Canny edged in Hardware(PL): 640x360 pixels.

Finally we need to reset all the contiguous memory buffers.

```
In [37]: in_buffer.close()
out_buffer.close()
```

```
In [38]: xlnk.xlnk_reset()
```