# UHD4.0 RFNoC 调试记录

* [1 环境依赖](#X332b57aacb8f36956d1b8deb0549c9ccd92a2b1)
  + [1.1 Python3](#X79eed248599aacf724092bbfc88e8ed9b4a3228)
  + [1.2 Vivado](#X929896d4cf1a02d8620209c217d040675ba00aa)
  + [1.3 安装指定软件](#X6baba6806c33d6a4218539d1b049366258384c3)
* [2 创建OOT(out-of-tree)模组](#X62d977057811693297f50094290fe007041f821)
* [3 FPGA框架(无IP)](#X150fabe34dabfb465415d7b3a8539bd2cc40cf9)
  + [3.1 rfnoc\_block\_gain.v](#Xa27041134f4f2493a09fa06c7259047392bb8d5)
  + [3.2 noc\_shell\_gain.v](#Xdd57cc5a14e5dcf685333de1dd1a52c63c34b4e)
  + [3.3 rfnoc\_block\_gain\_tb.sv](#X9a13ef0803dd0a81979e4bcaa34a03516ce388a)
  + [3.4 Makefile.srcs](#X9f980c80d296ac77545e7fe828834c90eb0ac1d)
  + [3.5 gain\_x310\_rfnoc\_image\_core.yml](#X4391795903370dcb9d3ec8827e4fa008a87c80e)
  + [3.6 编译](#X4cc5bf56fa5beff8eb8d8a7ec2479a83a5909d5)
  + [3.7 仿真](#X5e017dcb67d358b4414288b92fdee41a3a5d23b)
  + [3.8 bit流](#Xfce58f61f7270376a8156c4dc95dfd1c99c7269)
* [4 FPGA框架(有IP)](#Xe4eb32959355c5477bc73d4b6d7a1f3df2ea730)
  + [4.1 仿真](#X392efae853f1a4e51498b1d2e872c89a93083fb)
  + [4.2 bit流 (方案1)](#X1700f284394f4c1a487377dcaeac04504039d3b)
  + [4.3 bit流 (方案2)](#X16e67cd98f6a187deb451381bbf9f8f0f0aa97d)
* [5 UHD框架 (C++)](#Xb15305ebca7a3b833082f92cfd60599ca617fd8)
* [6 GNU Radio框架](#Xe1b78a891a67e8f12239d82197be454fa18e6b3)
* [7 编译与烧录](#X60f8d6dbabac2f19953afc22336dff5031cb2b9)
* [8 测试](#X25fc04106883a1e1ccf6f0c12e7dc962c195793)

## 

## 1 环境依赖

### 1.1 Python3

sudo apt install git cmake g++ libboost-all-dev libgmp-dev swig \  
python3-numpy python3-mako python3-sphinx python3-lxml \  
doxygen libfftw3-dev libsdl1.2-dev libgsl-dev libqwt-qt5-dev \  
libqt5opengl5-dev python3-pyqt5 liblog4cpp5-dev libzmq3-dev \  
python3-yaml python3-click python3-click-plugins python3-zmq \  
python3-scipy python3-gi python3-gi-cairo gobject-introspection \  
gir1.2-gtk-3.0 build-essential libusb-1.0-0-dev python3-docutils \  
python3-setuptools python3-ruamel.yaml python-is-python3  
  
export PYTHONPATH=/usr/local/lib/python3/dist-packages

### 1.2 Vivado

* 建议安装至系统默认的/opt/Xilinx/ 路径下，之后步骤可省去路径指定。
* 安装缺少的库
* sudo apt install libtinfo5 libncurses5

### 1.3 安装指定软件

创建安装软件的环境文件夹<uhd\_dir>，并在其下编译源码安装，且编译安装完成后**不要**删除源码。不建议使用apt-get直接安装软件。

* UHD 4.0
* 本方案只测试过UHD4.0与UHD4.6，在UHD4+的版本中方法类似，只存在 \*\_impl.cc、\*.v等自动生成文件细小差别，可自行选择所需的UHD版本。
* cd /<uhd\_dir>  
   git clone --branch UHD-4.0 https://github.com/ettusresearch/uhd.git uhd  
   cd uhd/host/  
   mkdir build && cd build  
   cmake ../  
   make -j8  
   sudo make install
* gnuradio 3.8
* gnuradio与UHD需版本匹配，否则大概率无法工作。
* cd /<uhd\_dir>  
   git clone --branch maint-3.8 --recursive https://github.com/gnuradio/gnuradio.git gnuradio  
   cd gnuradio/  
   mkdir build && cd build  
   cmake ../  
   make -j8  
   sudo make install
* gr-ettus
* cd /<uhd\_dir>  
   git clone --branch maint-3.8-uhd4.0 https://github.com/ettusresearch/gr-ettus.git gr-ettus  
   cd gr-ettus/  
   mkdir build && cd build  
   cmake --DENABLE\_QT=True ../  
   make -j8  
   sudo make install

## 2 创建OOT(out-of-tree)模组

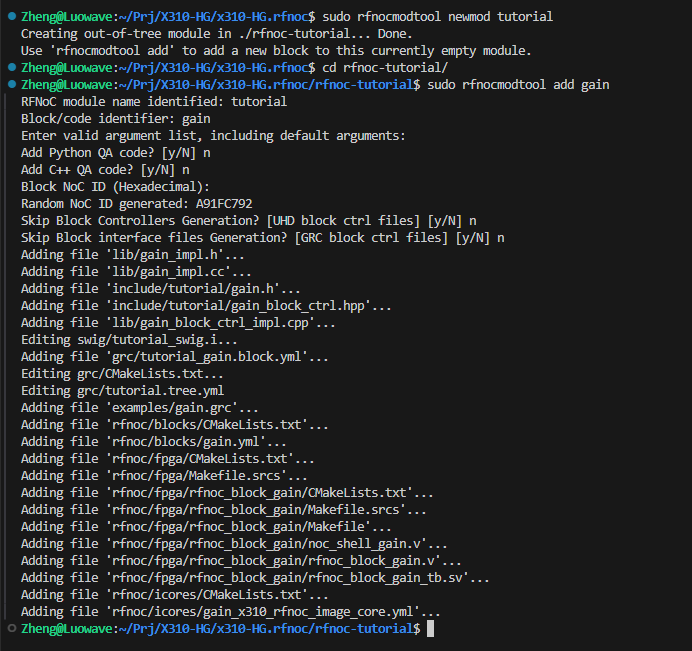
创建工作所需工作空间<work\_dir>，并键入以下指令：

cd /<work\_dir>  
 rfnocmodtool newmod <OOT\_name>  
 cd rfnoc-<OOT\_name>   
 rfnocmodtool add <block\_name>

其中<OOT\_name>为整个模组名字，类似于类名；<block\_name>为块名，类似于方法名。 在以下的示例中<OOT\_name>使用tutorial，<block\_name>使用**gain**（无IP核心）与**mult**（有IP核）演示。

键入指令后出现如下选项：

RFNoC module name identified: tutorial  
 Block/code identifier: gain  
 Enter valid argument list, including default arguments:   
 Add Python QA code? [y/N] n  
 Add C++ QA code? [y/N] n  
 Block NoC ID (Hexadecimal):   
 # 可自定义模块id，直接回车则是随机id号  
 Random NoC ID generated: A91FC792  
 Skip Block Controllers Generation? [UHD block ctrl files] [y/N] n  
 Skip Block interface files Generation? [GRC block ctrl files] [y/N] n

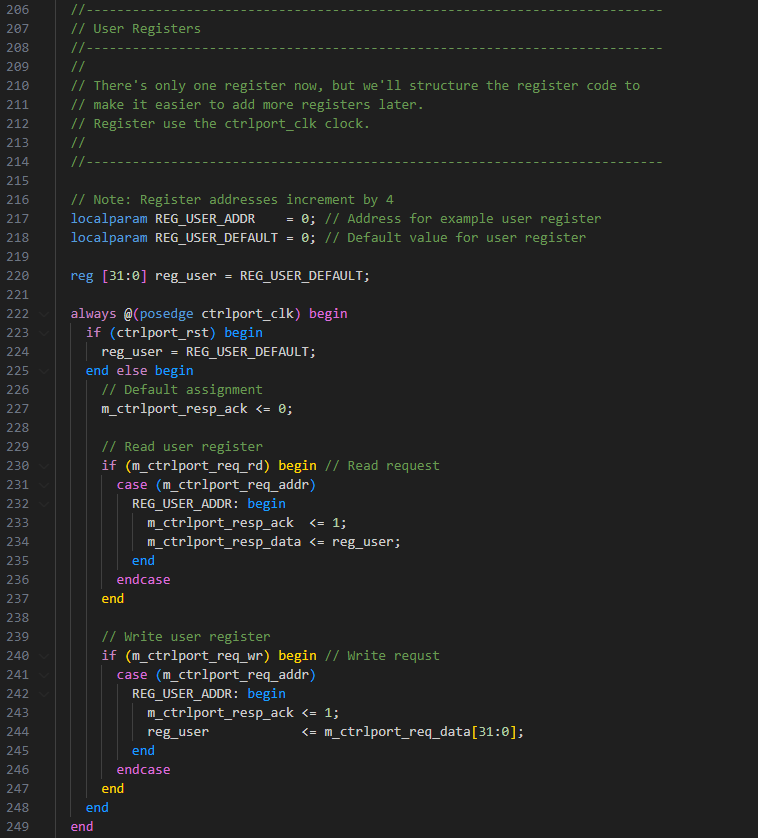
具体工作情况如下： 

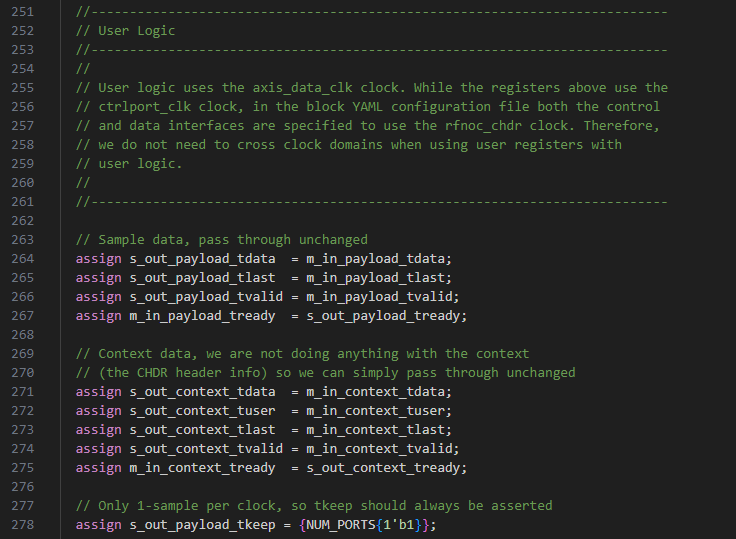
其文件树仅供参考，见[File Tree](file_tree2.1.txt)

## 3 FPGA框架(无IP)

### 3.1 rfnoc\_block\_gain.v

其在路径/rfnoc-tutorial/rfnoc/fpga/rfnoc\_block\_gain/下，需修改其寄存器相关代码，并添加用户逻辑RTL代码。

寄存器代码如下，需修改其寄存器地址与初始值： 

用户逻辑代码如下，添加相关逻辑，其中相关时序需严格处理： 

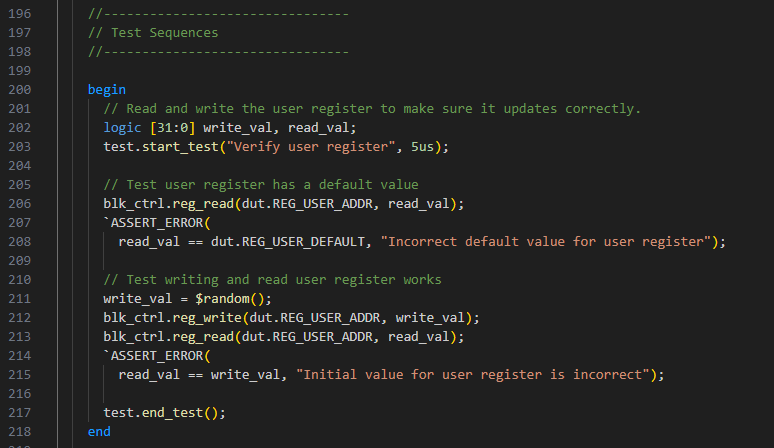
### 3.2 noc\_shell\_gain.v

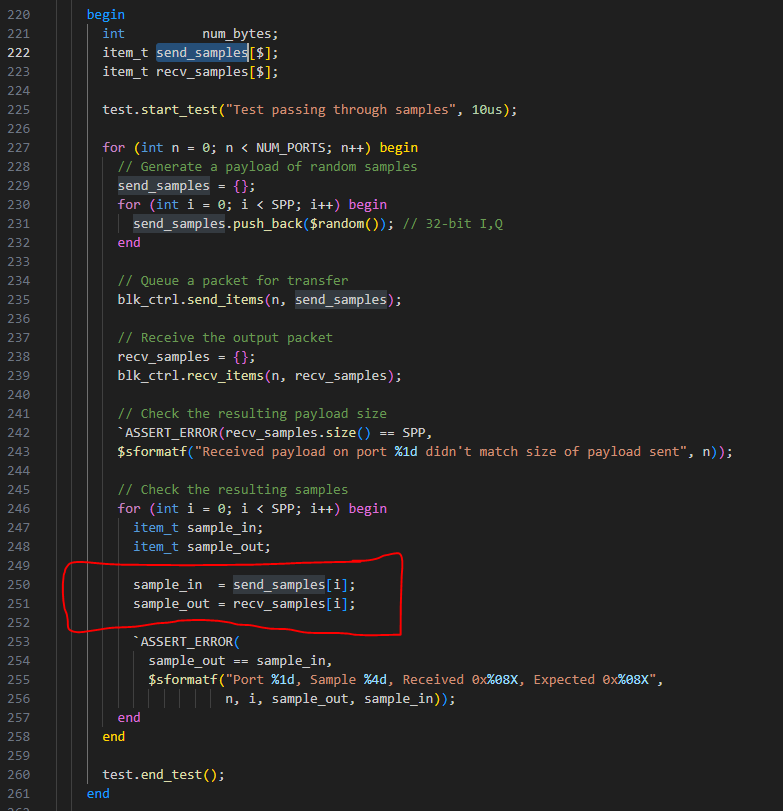
其在路径/rfnoc-tutorial/rfnoc/fpga/rfnoc\_block\_gain/下，若需要的模块与RFNoC框架间存在输入输出数据交互，则无需更改；若仅存在输出，不存在输入交互，则需修改，并严格遵守握手时序，参考代码见官方的**rfnoc\_block\_siggen**相关代码，相关文件位于：

<uhd\_dir>/UHD-4.0/uhd/fpga/usrp3/lib/rfnoc/blocks/rfnoc\_block\_siggen/

### 3.3 rfnoc\_block\_gain\_tb.sv

其在路径/rfnoc-tutorial/rfnoc/fpga/rfnoc\_block\_gain/下，需修改其寄存器读写测试以及输入输出测试代码。

寄存器读写测试代码如下，替换成RTL逻辑代码中设置的寄存器名及其地址： 

输入输出测试代码如下，修改sample\_in与sample\_out以达到输入输出验证的效果，send\_samples[i]与recv\_samples[i]分别为dut模块的用户代码块输入输出载荷。 

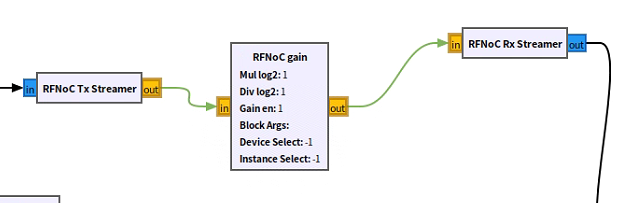
### 3.4 Makefile.srcs

其在路径/rfnoc-tutorial/rfnoc/fpga/rfnoc\_block\_gain/下，添加编译所需的.v .sv .vh文件，例如：

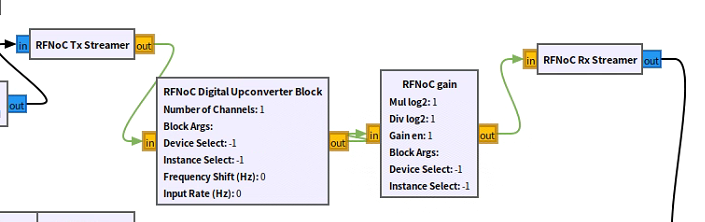
RFNOC\_OOT\_SRCS += $(addprefix $(dir $(abspath $(lastword $(MAKEFILE\_LIST)))), rfnoc\_gain\_core.v rfnoc\_block\_gain\_regs.vh)

### 3.5 gain\_x310\_rfnoc\_image\_core.yml

其在路径/rfnoc-tutorial/rfnoc/icores/下，后续工具会根据此描述文件生成RFNoC框架，默认生成的框架中用户模块被直接连接在一个Endpoint上，没有与ddc、duc等模块级联。也就是说在gnuradio的流图中无需与其他模块绑定使用，但也可以在流图中与yml描述存在的模块级联。以下部分流图仅供参考：

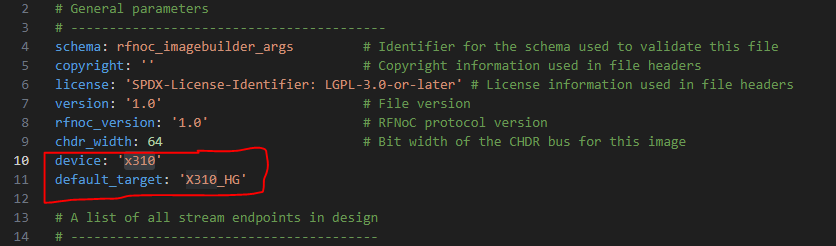


gnuradio connect 1



**若要添加多个用户逻辑代码块还需自行设计框架结构**。

在此文件中亦指定了生成的**目标设备型号**与FPGA最终实现的**硬件指标**等，具体args请在官网的具体设备下查找。



### 3.6 编译

键入以下指令：

cd /<work\_dir>/rfnoc-tutorial/  
 mkdir build && cd build  
 cmake -DUHD\_FPGA\_DIR=/<uhd\_dir>/UHD-4.0/uhd/fpga/ ../

成功后有如下提示

-- Configuring done  
 -- Generating done  
 -- Build files have been written to:

此时有许多可选功能实现，键入以下指令查看：

make help

The following are some of the valid targets for this Makefile:  
 ... all (the default if no target is provided)  
 ... clean  
 ... depend  
 ... install/strip  
 ... install  
 ... uninstall  
 ... testbenches  
 ... rebuild\_cache  
 ... install/local  
 ... test  
 ... list\_install\_components  
 ... edit\_cache  
 ... gnuradio-tutorial  
 ... pygen\_apps\_9a6dd  
 ... doxygen\_target  
 ... tutorial\_swig\_swig\_doc  
 ... \_tutorial\_swig\_doc\_tag  
 ... tutorial\_swig  
 ... tutorial\_swig\_swig\_compilation  
 ... pygen\_swig\_5cf62  
 ... pygen\_python\_d20bf  
 ... rfnoc\_block\_gain\_tb  
 ... gain\_x310\_rfnoc\_image\_core

### 3.7 仿真

此步骤可选择不执行验证，但即使对自己的核心逻辑代码做过验证，也无法保证其可在RFNoC框架下契合时序。**强烈建议在生成bit流前执行此仿真步骤**。

在build文件夹下键入以下指令便可查看RTL逻辑代码在RFNoC框架下的仿真是否通过：

make rfnoc\_block\_gain\_tb

如出现以下log信息且在此log下无error，则表示仿真通过。请注意有些时序错误显示pass但紧跟着下面打印ERROR，例如CHDR不匹配等信息。

========================================================  
 TESTBENCH STARTED: rfnoc\_block\_gain\_tb  
 ========================================================  
 [TEST CASE 1] (t = 0 ns) BEGIN: Flush block then reset it...  
 [TEST CASE 1] (t = 6400 ns) DONE... Passed  
 [TEST CASE 2] (t = 6400 ns) BEGIN: Verify Block Info...  
 [TEST CASE 2] (t = 6400 ns) DONE... Passed  
 [TEST CASE 3] (t = 6400 ns) BEGIN: Verify user register...  
 [TEST CASE 3] (t = 7825 ns) DONE... Passed  
 [TEST CASE 4] (t = 7825 ns) BEGIN: Test passing through samples...  
 [TEST CASE 4] (t = 8395 ns) DONE... Passed  
 ========================================================  
 TESTBENCH FINISHED: rfnoc\_block\_gain\_tb  
 - Time elapsed: 8395 ns  
 - Tests Run: 4  
 - Tests Passed: 4  
 - Tests Failed: 0  
 Result: PASSED   
 ========================================================

### 3.8 bit流

在build文件夹下键入以下指令便可编译制作bit流：

make gain\_x310\_rfnoc\_image\_core

此步骤耗时很久，根据目标设备在10min到2h不等（8核电脑环境下），最后生成的bit文件在路径：

# 具体路径以使用的设备为准，此例使用设备为X310  
 <uhd\_dir>/UHD-4.0/uhd/fpga/usrp3/top/x300/build/

## 4 FPGA框架(有IP)

### 4.1 仿真

关于RTL代码及相关处理，前面可按**3.1**~**3.6**处理，存在ip的情况与无ip在**3.7**开始有些区别。

打开/<work\_dir>/rfnoc-tutorial/rfnoc/fpga/rfnoc\_block\_gain/Makefile.srcs，并添加仿真相关的ip核心xci文件。

RFNOC\_OOT\_SRCS += $(addprefix $(dir $(abspath $(lastword $(MAKEFILE\_LIST)))), /ip/my\_gain.xci )

然后再仿真测试

make rfnoc\_block\_gain\_tb

### 4.2 bit流 (方案1)

注释上面仿真步骤中在Makefile.srcs中添加的xci文件，不然后续生成bit流时会冲突。

# RFNOC\_OOT\_SRCS += $(addprefix $(dir $(abspath $(lastword $(MAKEFILE\_LIST)))), /ip/my\_gain.xci )

将所需的ip添加至源码库中编译：

# 不同设备的编译库不同  
 cd /<uhd\_dir>/UHD-4.0/uhd/fpga/usrp3/lib/ip/  
 mkdir my\_gain && cd my\_gain  
 cp <ip\_dir>/my\_gain.xci my\_gain.xci  
 gedit Makefile.inc

# Makefile.inc中添加以下代码  
 include $(TOOLS\_DIR)/make/viv\_ip\_builder.mak  
 LIB\_IP\_MY\_GAIN\_SRCS = $(IP\_BUILD\_DIR)/my\_gain/my\_gain.xci  
 LIB\_IP\_MY\_GAIN\_OUTS = $(addprefix $(IP\_BUILD\_DIR)/my\_gain/, \  
 my\_gain.xci.out \  
 synth/my\_gain.vhd \  
 )  
 $(LIB\_IP\_MY\_GAIN\_SRCS) $(LIB\_IP\_MY\_GAIN\_OUTS) : $(LIB\_IP\_DIR)/my\_gain/my\_gain.xci  
 $(call BUILD\_VIVADO\_IP,my\_gain,$(ARCH),$(PART\_ID),$(LIB\_IP\_DIR),$(IP\_BUILD\_DIR),0)  
 # 保存后离开

gedit ../Makefile.inc

# 在其中的.PHONY: lib\_ip前添加  
 include $(LIB\_IP\_DIR)/my\_gain/Makefile.inc  
 LIB\_IP\_XCI\_SRCS += $(LIB\_IP\_MY\_GAIN\_SRCS)  
 LIB\_IP\_SYNTH\_OUTPUTS += $(LIB\_IP\_MY\_GAIN\_OUTS)  
 # 保存后离开

cd /<work\_dir>/rfnoc-tutorial/build/  
 make gain\_x310\_rfnoc\_image\_core

目标文件路径同**3.8**中一样。

### 4.3 bit流 (方案2)

此方案需要源码编译生成的Vivado图形化工程(.xpr)，并在Vivado的UI界面中编译。

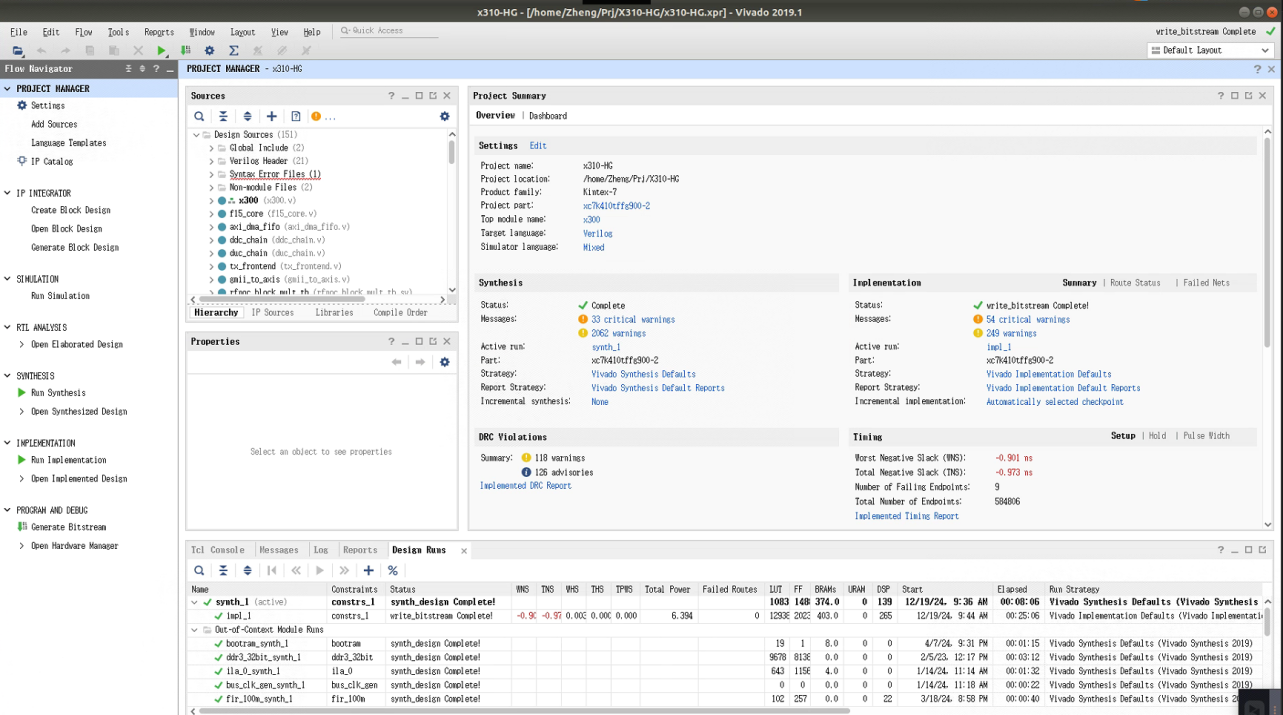
cd /<work\_dir>/rfnoc-tutorial/  
 gedit CMakeLists.txt

// 检索字符串\_rfnoc\_image\_builder\_exe，找到以下指令  
 add\_custom\_target(${\_target\_name}  
 COMMAND ${\_rfnoc\_image\_builder\_exe} -F ${UHD\_FPGA\_DIR} -y ${CMAKE\_CURRENT\_SOURCE\_DIR}/${\_rfnoc\_image\_core\_SRC} -I ${CMAKE\_SOURCE\_DIR}/rfnoc  
 )

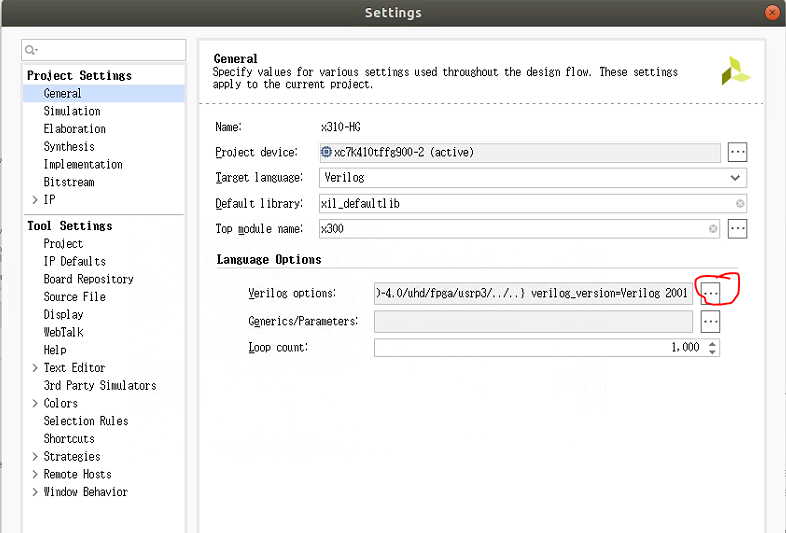
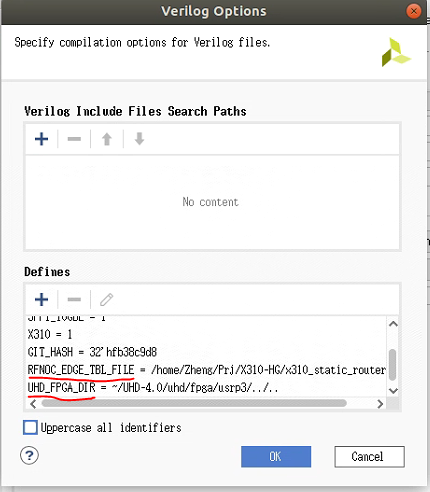
// 将上面代码段的add\_custom\_target后面添加指令，让其只生成编译需要替换的hex与image core而不生成bit流  
 add\_custom\_target(${\_target\_name}  
 COMMAND ${\_rfnoc\_image\_builder\_exe} -F ${UHD\_FPGA\_DIR} -y ${CMAKE\_CURRENT\_SOURCE\_DIR}/${\_rfnoc\_image\_core\_SRC} -I ${CMAKE\_SOURCE\_DIR}/rfnoc --generate-only  
 )

cd build  
 make gain\_x310\_rfnoc\_image\_core

打开Vivado的UI界面，并打开源工程



打开选项界面Tools -> Settings -> general -> Verilog options，并修改参数RFNOC\_EDGE\_TBL\_FILE与参数UHD\_FPGA\_DIR。

* RFNOC\_EDGE\_TBL\_FILE: 改为/<work\_dir>/rfnoc-tutorial/icores/x310\_static\_router.hex
* UHD\_FPGA\_DIR: 改为/<uhd\_dir>/UHD-4.0/uhd/fpga/usrp3/../..
* 
* 

然后将刚刚在<work\_dir>中生成的image core替换到该工程的image core。

若为标准x310工程，image core为 /x310-HG.srcs/sources\_1/imports/usrp3/top/x300/x310\_rfnoc\_image\_core.v

在UI界面中检查是否为刚生成的image core内容，确认无误后便可在UI界面中完成Synthesis -> Implementation -> Bitstream的流程。在这里可以用图形化界面添加调用IP核，更为直观方便。

## 5 UHD框架 (C++)

根据用户需求，主要改写寄存器映射。需要改写

/<work\_dir>/rfnoc-tutorial/lib/gain\_block\_ctrl\_impl.cpp

/<work\_dir>/rfnoc-tutorial/include/tutorial/gain\_block\_ctrl.hpp

下面给出简单改写后的文件，仅供参考：

[gain\_block\_ctrl\_impl.cpp](/example/gain_block_ctrl_impl.cpp)

[gain\_block\_ctrl.hpp](/example/gain_block_ctrl.hpp)

## 6 GNU Radio框架

根据用户需求，主要改写gnuradio图形化界面模块的对用户接口，以及UHD映射。需要改写

/<work\_dir>/rfnoc-tutorial/grc/tutorial\_gain.block.yml

下面给出简单改写后的文件，仅供参考：

[tutorial\_gain.block.yml](example\\tutorial_gain.block.yml)

## 7 编译与烧录

在处理完上面的FPGA -> UHD -> GNU Radio的关系后还需整体编译，并添加至环境中。

cd /<work\_dir>/rfnoc-tutorial/build/  
 make gnuradio-tutorial  
 make -j8  
 sudo make install

将之前生成的bit流烧录至目标设备中。

**此步骤需谨慎进行，若对FPGA源代码改动较大容易变砖，建议在Vivado的UI界面用jtag调试通过后再烧写。若变砖了也可在重新上电后，通过jtag调试原版bit流，再使用uhd\_image\_loader烧写。**

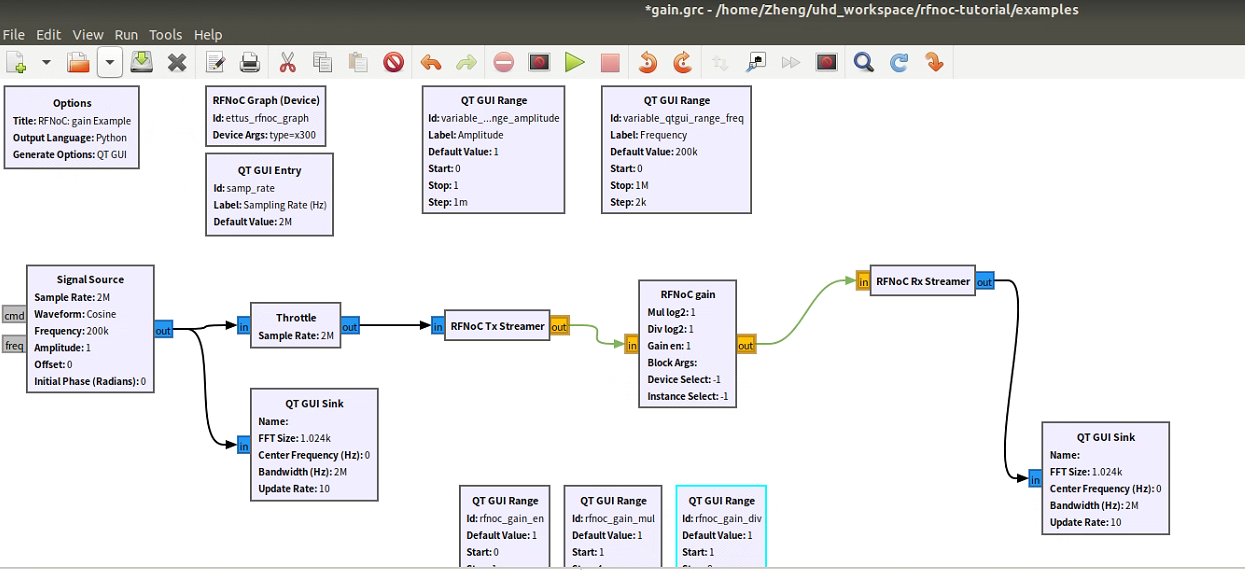
uhd\_image\_loader --args "type=x300,addr=192.168.10.2" --fpga-path /<uhd\_dir>/UHD-4.0/uhd/fpga/usrp3/top/x300/build/usrp\_x310\_fpga\_HG.bit

## 8 测试

打开GNU Radio

sudo gnuradio-companion

图形化界面中打开文件/<work\_dir>/rfnoc-tutorial/example/gain.grc



点击运行，若弹出理想波形则成功。

有时gnuradio会报错找不到\*.py的包，此时需手动搬移：

cp -r /usr/local/lib/python3/dist-packages/<缺少的包名> /usr/lib/python3/dist-packages/