UHD4.0 RFNoC 调试记录

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 uh
d
  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/g
  ```

```
nuradio.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/g
r-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：

```
Zheng@Luowave:~/Prj/X310-HG/x310-HG.rfnoc$ sudo rfnocmodtool newmod tutorial
Creating out-of-tree module in ./rfnoc-tutorial... Done.
Use 'rfnocmodtool add' to add a new block to this currently empty module.
Zheng@Luowave:~/Prj/X310-HG/x310-HG.rfnoc$ cd rfnoc-tutorial/
Zheng@Luowave:~/Prj/X310-HG/x310-HG.rfnoc/rfnoc-tutorial$ sudo rfnocmodtool add gain
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):
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
Adding file 'lib/gain_impl.h'...
Adding file 'lib/gain_impl.cc'...
Adding file 'include/tutorial/gain.h'...
Adding file 'include/tutorial/gain_block_ctrl.hpp'...
Adding file 'lib/gain_block_ctrl_impl.cpp'...
Editing swig/tutorial_swig.i...
Adding file 'grc/tutorial_gain.block.yml'...
Editing grc/CMakeLists.txt...
Editing grc/tutorial.tree.yml
Adding file 'examples/gain.grc'...
Adding file 'rfnoc/blocks/CMakeLists.txt'...
Adding file 'rfnoc/blocks/gain.yml'...
Adding file 'rfnoc/fpga/CMakeLists.txt'...
Adding file 'rfnoc/fpga/Makefile.srcs'...
Adding file 'rfnoc/fpga/rfnoc_block_gain/CMakeLists.txt'...
Adding file 'rfnoc/fpga/rfnoc_block_gain/Makefile.srcs'...
Adding file 'rfnoc/fpga/rfnoc_block_gain/Makefile'...
Adding file 'rfnoc/fpga/rfnoc_block_gain/noc_shell_gain.v'...
Adding file 'rfnoc/fpga/rfnoc_block_gain/rfnoc_block_gain.v'...
Adding file 'rfnoc/fpga/rfnoc_block_gain/rfnoc_block_gain_tb.sv'...
Adding file 'rfnoc/icores/CMakeLists.txt'...
Adding file 'rfnoc/icores/gain_x310_rfnoc_image_core.yml'...
Zheng@Luowave:~/Prj/X310-HG/x310-HG.rfnoc/rfnoc-tutorial$
```

其文件树仅供参考，见 File Tree

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：

```
  //-----------------------------------------------------------------------------
  // User Registers
  //-----------------------------------------------------------------------------
  //
  // There's only one register now, but we'll structure the register code to
  // make it easier to add more registers later.
  // Register use the ctrlport_clk clock.
  //
  //-----------------------------------------------------------------------------

  // Note: Register addresses increment by 4
  localparam REG_USER_ADDR    = 0; // Address for example user register
  localparam REG_USER_DEFAULT = 0; // Default value for user register

  reg [31:0] reg_user = REG_USER_DEFAULT;

  always @(posedge ctrlport_clk) begin
    if (ctrlport_rst) begin
      reg_user = REG_USER_DEFAULT;
    end else begin
      // Default assignment
      m_ctrlport_resp_ack <= 0;

      // Read user register
      if (m_ctrlport_req_rd) begin // Read request
        case (m_ctrlport_req_addr)
          REG_USER_ADDR: begin
            m_ctrlport_resp_ack  <= 1;
            m_ctrlport_resp_data <= reg_user;
          end
        endcase
      end

      // Write user register
      if (m_ctrlport_req_wr) begin // Write requst
        case (m_ctrlport_req_addr)
          REG_USER_ADDR: begin
            m_ctrlport_resp_ack <= 1;
            reg_user            <= m_ctrlport_req_data[31:0];
          end
        endcase
      end
    end
  end
```

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：

```
251   //---------------------------------------------------------------------------
252   // User Logic
253   //---------------------------------------------------------------------------
254   //
255   // User logic uses the axis_data_clk clock. While the registers above use the
256   // ctrlport_clk clock, in the block YAML configuration file both the control
257   // and data interfaces are specified to use the rfnoc_chdr clock. Therefore,
258   // we do not need to cross clock domains when using user registers with
259   // user logic.
260   //
261   //---------------------------------------------------------------------------
262
263   // Sample data, pass through unchanged
264   assign s_out_payload_tdata  = m_in_payload_tdata;
265   assign s_out_payload_tlast  = m_in_payload_tlast;
266   assign s_out_payload_tvalid = m_in_payload_tvalid;
267   assign m_in_payload_tready  = s_out_payload_tready;
268
269   // Context data, we are not doing anything with the context
270   // (the CHDR header info) so we can simply pass through unchanged
271   assign s_out_context_tdata  = m_in_context_tdata;
272   assign s_out_context_tuser  = m_in_context_tuser;
273   assign s_out_context_tlast  = m_in_context_tlast;
274   assign s_out_context_tvalid = m_in_context_tvalid;
275   assign m_in_context_tready  = s_out_context_tready;
276
277   // Only 1-sample per clock, so tkeep should always be asserted
278   assign s_out_payload_tkeep = {NUM_PORTS{1'b1}};
```

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：

```
196     //---------------------------------
197     // Test Sequences
198     //---------------------------------
199
200     begin
201       // Read and write the user register to make sure it updates correctly.
202       logic [31:0] write_val, read_val;
203       test.start_test("Verify user register", 5us);
204
205       // Test user register has a default value
206       blk_ctrl.reg_read(dut.REG_USER_ADDR, read_val);
207       `ASSERT_ERROR(
208         read_val == dut.REG_USER_DEFAULT, "Incorrect default value for user register");
209
210       // Test writing and read user register works
211       write_val = $random();
212       blk_ctrl.reg_write(dut.REG_USER_ADDR, write_val);
213       blk_ctrl.reg_read(dut.REG_USER_ADDR, read_val);
214       `ASSERT_ERROR(
215         read_val == write_val, "Initial value for user register is incorrect");
216
217       test.end_test();
218     end
```

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。

```
220   begin
221     int          num_bytes;
222     item_t send_samples[$];
223     item_t recv_samples[$];
224
225     test.start_test("Test passing through samples", 10us);
226
227     for (int n = 0; n < NUM_PORTS; n++) begin
228       // Generate a payload of random samples
229       send_samples = {};
230       for (int i = 0; i < SPP; i++) begin
231         send_samples.push_back($random()); // 32-bit I,Q
232       end
233
234       // Queue a packet for transfer
235       blk_ctrl.send_items(n, send_samples);
236
237       // Receive the output packet
238       recv_samples = {};
239       blk_ctrl.recv_items(n, recv_samples);
240
241       // Check the resulting payload size
242       `ASSERT_ERROR(recv_samples.size() == SPP,
243       $sformatf("Received payload on port %1d didn't match size of payload sent", n));
244
245       // Check the resulting samples
246       for (int i = 0; i < SPP; i++) begin
247         item_t sample_in;
248         item_t sample_out;
249
250         sample_in  = send_samples[i];
251         sample_out = recv_samples[i];
252
253         `ASSERT_ERROR(
254           sample_out == sample_in,
255           $sformatf("Port %1d, Sample %4d, Received 0x%08X, Expected 0x%08X",
256                     n, i, sample_out, sample_in));
257       end
258     end
259
260     test.end_test();
261   end
```

3.4 Makefile.srcs

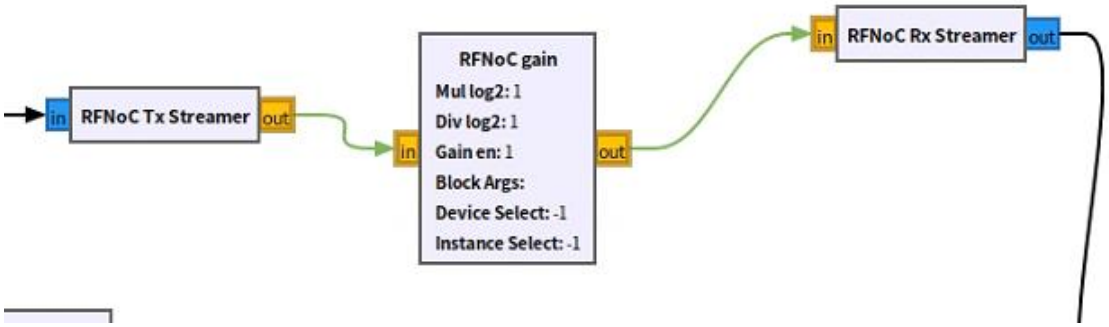
其在路径/rfnoc-tutorial/rfnoc/fpga/rfnoc_block_gain/下，添加编译所需的.v .sv .vh 文件，例如：

```
RFNOC_OOT_SRCS += $(addprefix $(dir $(abspath $(lastword $(MAKEFILE_LIST)))), rfnoc_gain_core.v rfnoc_block_gain_regs.vh)
```

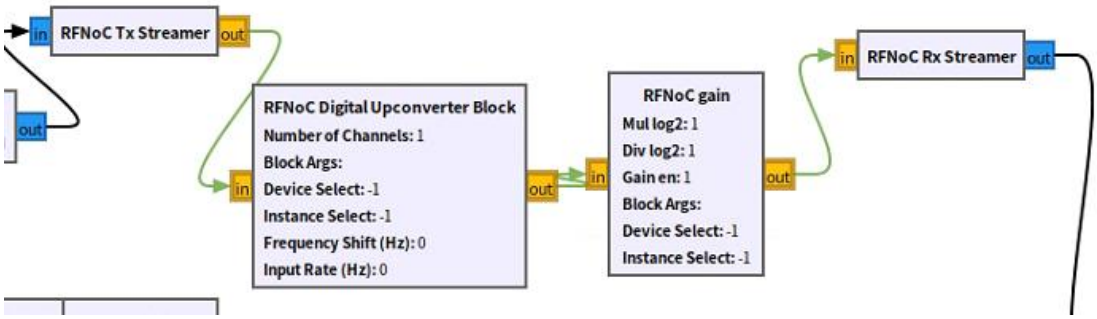
3.5 gain_x310_rfnoc_image_core.yml

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。

```
# General parameters
# ----------------------------------------
schema: rfnoc_imagebuilder_args         # Identifier for the schema used to validate this file
copyright: ''                           # Copyright information used in file headers
license: 'SPDX-License-Identifier: LGPL-3.0-or-later' # License information used in file headers
version: '1.0'                          # File version
rfnoc_version: '1.0'                    # RFNoC protocol version
chdr_width: 64                          # Bit width of the CHDR bus for this image
device: 'x310'
default_target: 'X310_HG'

# A list of all stream endpoints in design
# ----------------------------------------
```

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。

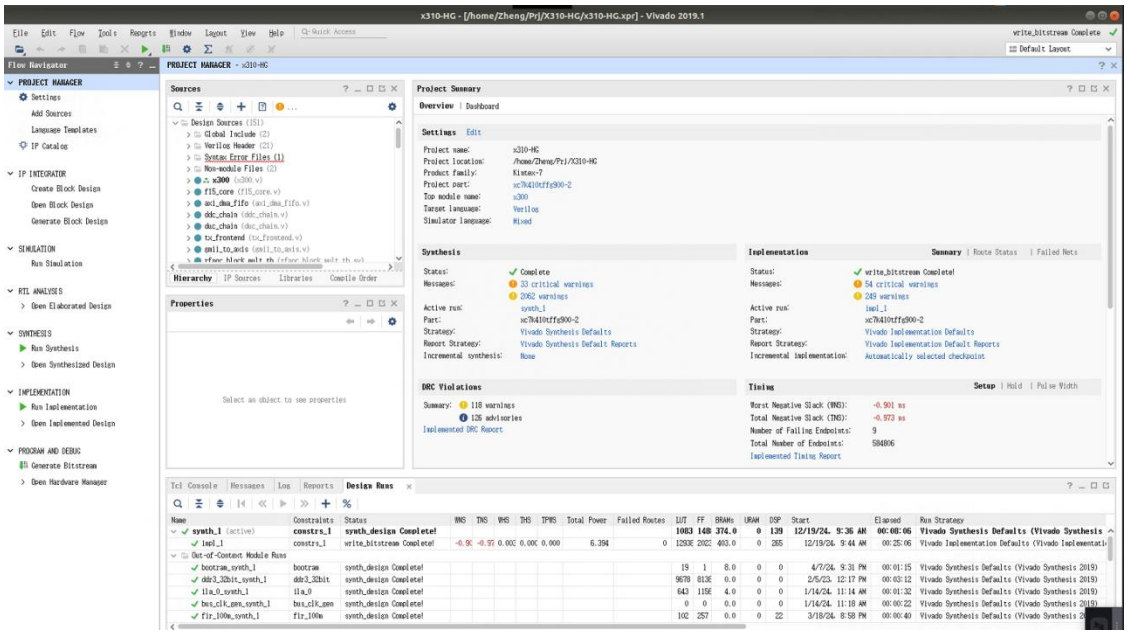
```
cd /<work_dir>/rfnoc-tutorial/
gedit CMakeLists.txt
```

```
// 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
)
```

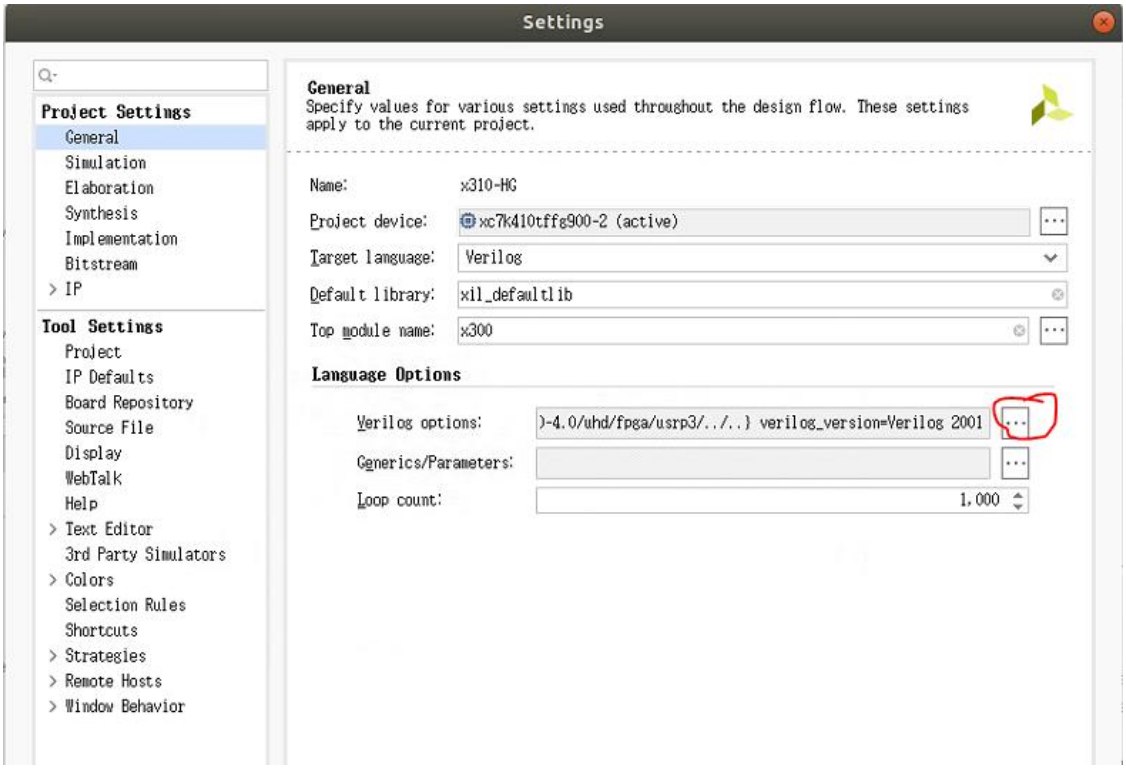
```
cd build
make gain_x310_rfnoc_image_core
```

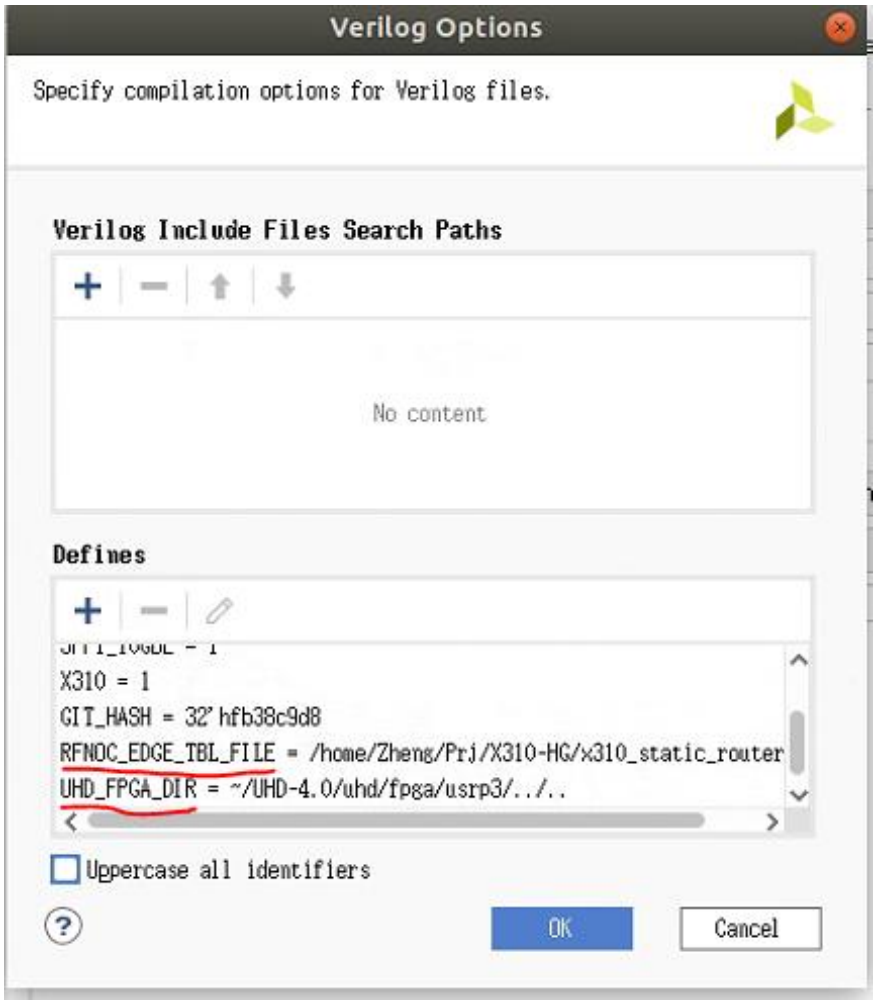
打开 Vivado 的 UI 界面，并打开源工程



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/../..


Settings
Project Settings
General
Simulation
Elaboration
Synthesis
Implementation
Bitstream
IP
Tool Settings
Project
IP Defaults
Board Repository
Source File
Display
WebTalk
Help
Text Editor
3rd Party Simulators
Colors
Selection Rules
Shortcuts
Strategies
Remote Hosts
Window Behavior
General
Specify values for various settings used throughout the design flow. These settings apply to the current project.
Name:
x310-HG
Project device:
xc7k410tffg900-2 (active)
Target language:
Verilog
Default library:
xil_defaultlib
Top module name:
x300
Language Options
Verilog options:
)-4.0/uhd/fpga/usrp3/../..} verilog_version=Verilog 2001
Generics/Parameters:
Loop count:
1.000



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

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

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。

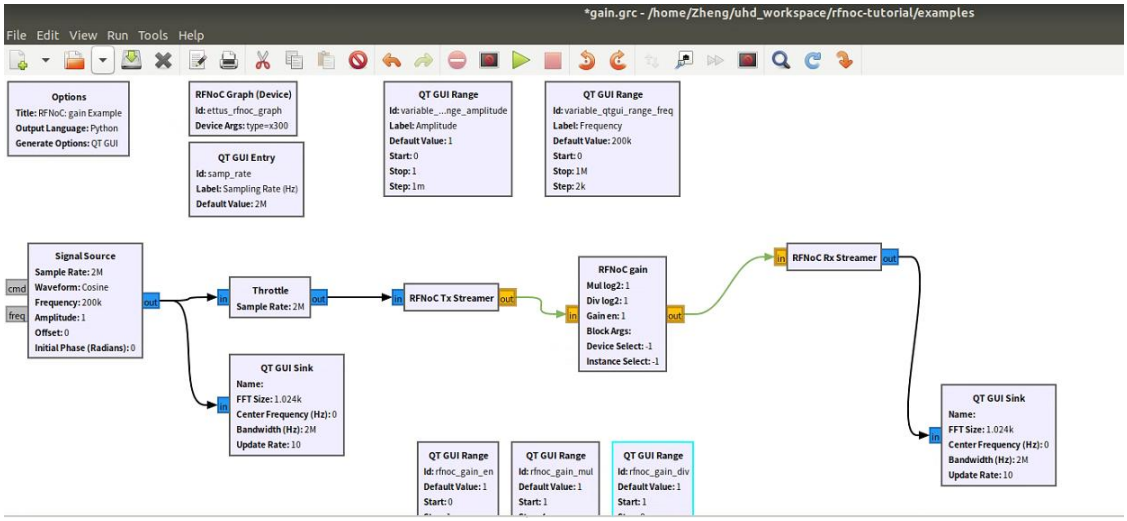
```
uhd_image_loader --args "type=x300,addr=192.168.10.2" --fpga-path /<uhd_dir>/UHD-4.0/uhd/fpga/usrp3/top/x300/build/usrp_x310_fpga_HG.bit
```

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/
```