RFNoC 4 Workshop

Part 2

Jonathon Pendulum – Ettus Research
Neel Pandeya – Ettus Research
GRCon 2020
Schedule

- Part 1
  - RFNoC 4 Framework Overview
  - Hands on Demos
- Part 2
  - FPGA Architecture
  - Software Implementation
  - GNU Radio Integration
  - Hands on RFNoC Block Development
Getting RFNoC

- **OS:** Ubuntu 20.04

- **Dependencies:**
  - `sudo apt install git cmake g++ libboost-all-dev libgmp-dev-dev swig \ python3-numpy python3-mako python3-sphinx python3-lxml \ doxygen libfftw3-dev libstdc++1.2-dev libgsl-dev libqwt-qt5-dev \ libqt5-opengl5-dev python3-pyqt5 liblog4c++5-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`

- **Vivado 2019.1**
  - Install missing libs: `apt install libtinfo5 libncurses5`
  - Install Design version (even if only doing simulations)
  - Simulations can run without a license
  - Building bitstreams requires full license (except E310)
Getting RFNoC

- **Software:**
  - **UHD 4.0:**
    - `git clone --branch UHD-4.0 \ https://github.com/ettusresearch/uhd.git uhd`
    - `mkdir uhd/host/build; cd uhd/host/build; cmake ..`
    - `make -j4; sudo make install`
  - **GNU Radio 3.8:**
    - `git clone --branch maint-3.8 --recursive \ https://github.com/gnuradio/gnuradio.git gnuradio`
    - `mkdir gnuradio/build; cd gnuradio/build; cmake ..`
    - `make -j4; sudo make install`
  - **gr-ettus:**
    - `git clone --branch maint-3.8-uhd4.0 \ https://github.com/ettusresearch/gr-ettus.git gr-ettus`
    - `mkdir gr-ettus/build; cd gr-ettus/build; cmake --DENABLE_QT=True ..`
    - `make -j4; sudo make install`
rfnocmodtool

- Installed with gr-ettus
- Generate OOT RFNoC modules
  - Similar concept to gr_modtool
  - Creates skeleton code for a pass through RFNoC Block
- FPGA:
  - Verilog Block code
  - SystemVerilog Test Bench
  - Image Core YAML
- UHD:
  - C++ Block Controller
  - Block Description YAML
- GNU Radio:
  - C++ Block Code
  - GRC YAML
  - Example Flowgraph
Create a OOT module

- Steps to create our block:
  - rfnocmodtool newmod tutorial
  - cd rfnoc-tutorial
  - rfnocmodtool add gain
    - Enter name of block/code (without module name prefix): gain
    - Enter valid argument list, including default arguments: (leave blank)
    - Add Python QA Code? [y/N] N
    - Add C++ QA Code? [y/N] N
    - Block NoC ID (Hexadecimal): 1234
      - Leaving blank results in random NoC ID
    - Skip Block Controllers Generation? [UHD block ctrl files] [y/N] N
    - Skip Block interface files Generation? [GRC block ctrl files] [y/N] N
RFNoC modtool output

```
rfnoc@rfnoc-vm:~$ src rfnocmodtool newmod
Name of the new module: tutorial
Creating out-of-tree module in ./rfnoc-tutorial... Done.
Use 'rfnocmodtool add' to add a new block to this currently empty module.
rfnoc@rfnoc-vm:~$ cd rfnoc-tutorial/
rfnoc@rfnoc-vm:~$ src/rfnoc-tutorial$ rfnocmodtool add
RFNoC module name identified: tutorial
Enter name of block/code (without module name prefix): gain
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): 1234
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'...
```
Directory Structure

- rfnoc/blocks
  - gain.yml – Block Description YAML
- rfnoc/fpga/rfnoc_block_gain
  - rfnoc_block_gain.v – RFNoC Block HDL
  - rfnoc_block_gain_tb.sv – RFNoC Block Test Bench
  - noc_shell_gain.v – Custom NoC Shell
- rfnoc/icores
  - gain_x310_rfnoc_image_core.yml – Image Core YAML
- lib
  - gain_block_ctrl_impl.cpp – UHD Block Controller C++ code
  - gain_impl.cc – GNU Radio Block C++ code
- include/tutorial
  - gain_block_ctrl.hpp – UHD Block Controller C++ header
  - gain.h – GNU Radio Block C++ header
- examples
  - gain.grc – Example flowgraph using RFNoC Block
RFNoC Framework

GNU Radio
- GRC Bindings (YAML)
- Block Code (Python / C++)

UHD
- Block Description (YAML)
- Block Controller (C++)

FPGA
- Block Test Bench (SystemVerilog)
- Block HDL (Verilog, VHDL, HLS, IP, BD)
RFNoC Framework

GNU Radio
- GRC Bindings (YAML)
- Block Code (Python / C++)

UHD
- Block Description (YAML)
- Block Controller (C++)

FPGA
- Block Test Bench (SystemVerilog)
- Block HDL (Verilog, VHDL, HLS, IP, BD)
RFNoC Block Overview

- Every block is connected to the **NoC Core**
  - Path for block to block and block to host communication
  - Implements crossbars for data and control routing
  - Support for static routing to save resources and reduce latency
  - Autogenerated
NoC Core

<table>
<thead>
<tr>
<th>Transport Interfaces</th>
<th>RFNoC Blocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 GigE</td>
<td>Radio 0</td>
</tr>
<tr>
<td>10 GigE</td>
<td>Radio 1</td>
</tr>
<tr>
<td>PCIe</td>
<td>DDC 0</td>
</tr>
<tr>
<td></td>
<td>DDC 1</td>
</tr>
<tr>
<td></td>
<td>DUC 0</td>
</tr>
<tr>
<td></td>
<td>DUC 1</td>
</tr>
<tr>
<td></td>
<td>Gain 0</td>
</tr>
</tbody>
</table>

**Autogenerated NoC Core**

- CHDR Crossbar
- Stream Endpoint 0
- Stream Endpoint 1
- Stream Endpoint 2
- Stream Endpoint 3
- Stream Endpoint 4

**RFNoC Blocks**

- Radio 0
- Radio 1
- DDC 0
- DDC 1
- DUC 0
- DUC 1
- Gain 0
NoC Core

Autogenerated NoC Core

RFNoC Blocks

Transport Interfaces

1 GigE

10 GigE

PCIe

AXIS-CHDR

AXIS-CTRL

CHDR Crossbar

Stream Endpoint 0

Stream Endpoint 1

Stream Endpoint 2

Stream Endpoint 3

Stream Endpoint 4

Ctrl Crossbar (AXIS-CTRL)

Static Routing (AXIS-CHDR)

Radio 0

Radio 1

DDC 0

DDC 1

DUC 0

DUC 1

Gain 0

Ettus Research™
NoC Core

RFNoC RX Radio
Number of Channels: 1
Block Args:
Device Select: -1
Instance Select: -1
Sample Rate (Hz): 1M
Antenna Select: RX2
Center Frequency (Hz): 10
Gain: 32
Automatic Gain Control: Default
Bandwidth (Hz): 0
DC Offset Correction: False
IQ Balance: False

RFNoC Digital Upconverter Block
Number of Channels: 1
Block Args:
Device Select: -1
Instance Select: -1
Frequency Shift (Hz): 0
Input Rate (Hz): 0

RFNoC Digital Downconverter Block
Number of Channels: 1
Block Args:
Device Select: -1
Instance Select: -1
Frequency Shift (Hz): 0
Output Rate (Hz): 1M

QT GUI Frequency Sink
FFT Size: 1024
Center Frequency (Hz): 10
Bandwidth (Hz): 1M

Autogenerated NoC Core

CHDR Crossbar
Endpoint 0
Endpoint 1
Endpoint 2
Endpoint 3
Endpoint 4

Ctrl Crossbar (AXIS-CTRL)

Static Routing (AXIS-CHDR)

RFNoC Blocks

Radio 0
Radio 1
DDC 0
DDC 1
DUC 0
DUC 1
Gain 0

Transport Interfaces
1 GigE
10 GigE
PCIe

AXIS-CHDR
AXIS-CTRL
NoC Core

Transport Interfaces

1 GigE
10 GigE
PCIe

RFNoC Blocks

Radio 0
Radio 1
DDC 0
DDC 1
DUC 0
DUC 1
Gain 0

Missing Routes

Autogenerated NoC Core

CHDR Crossbar
Stream Endpoint 0

Static Routing (AXIS-CHDR)

Ctrl Crossbar
(AXIS-CTRL)

RFNoC Digital Upconverter Block
Number of Channels: 1
Block Args:
Device Select: 1
Instance Select: 1
Sample Rate (Hz): 1M
Antenna Select: RX2
Center Frequency (Hz): 10
Gain: 32
Automatic Gain Control: Default
Bandwidth (Hz): 0
DC Offset Correction: False
IQ Balance: False

RFNoC Digital Downconverter Block
Number of Channels: 1
Block Args:
Device Select: 1
Instance Select: 1
Frequency Shift (Hz): 0
Input Rate (Hz): 0
Output Rate (Hz): 1M

RFNoC RX Radio
Number of Channels: 1
Block Args:
Device Select: 1
Instance Select: 1
Sample Rate (Hz): 1M
Antenna Select: RX2
Center Frequency (Hz): 10
Gain: 32
Automatic Gain Control: Default
Bandwidth (Hz): 0
DC Offset Correction: False
IQ Balance: False

17
RFNoC Block Overview

- Two types of AXI-Stream bus connections to NoC Core
  - **AXIS-CHDR**: Data packets → Sample data
    - Could be multiple buses wide for multiple ports (see DUC, DDC)
  - **AXIS-CTRL**: Control packets → Read / write user registers
- Transaction details available in specification
RFNoC Block Overview

- Autogenerated boilerplate RFNoC interfacing code
  - Created via rfnocmodtool or rfnoc_create_verilog.py
- Packetizes / De-packetizes Data and Control packets
  - Clock domain crossing
- Outputs several buses
  - Payload AXI-Stream bus -- Sample data
  - Context AXI-Stream bus -- CHDR Header and Meta data
  - Ctrl Port -- Read / write user registers
NoC Shell Internals

- **Input clocks for** `AXIS-CHDR` and `AXIS-CTRL` **bus clocks**
  - Output `axis_data_clk` derived from input `rfnoc_chdr_clk`
  - Output `ctrlport_clk` derived from input `rfnoc_ctrl_clk`
  - Usually all the same clock

- **User defined clocks (not shown)**
  - Must be defined in both the Block Description and Image Core YAML files
  - Example: `ce_clk` in DDC, DUC RFNoC blocks

---

**Payload**

**Context**

**Payload**

**Context**
NoC Shell Internals

- **Backend Interface**
  - Creates user resets: \texttt{axis\_data\_rst}, \texttt{ctrlport\_rst}
  - Provides block information and status
    - NoC ID, Buffer sizes, Number of Ports, Flush status

Diagram:

-_AXIS-CHDR to AXIS Payload & Context
- AXIS Payload & Context to_AXIS-CHDR
- Payload
- Context
- Payload
- Context

- AXIS-CHDR
- AXIS-CTRL
- AXIS
- CTRL PORT
- Clock
- Reset
NoC Shell Internals

- **AXIS-CHDR bus interfacing logic**
  - Handles packetizing / depacketizing AXIS-CHDR packets
  - User interacts with two AXI Stream buses

Backed Interface

AXIS-CHDR to
AXIS Payload & Context

- Depacketizer
- Clock Crossing FIFO
- Packet Flush

AXIS Payload & Context
to AXIS-CHDR

- Packet Flush & Gate
- Packetizer
- Clock Crossing FIFO

Ctrl Port Endpoint

Payload

Context

Payload

Context

AXIS-CHDR

AXIS-CTRL

AXIS

CTRL PORT

Clock

Reset
### CHDR Packet Breakdown

<table>
<thead>
<tr>
<th>Virtual Chan</th>
<th>EOV</th>
<th>EOB</th>
<th>Pkt Type</th>
<th>Num Metadata</th>
<th>Sequence #</th>
<th>Length</th>
<th>DST EP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(Optional) Timestamp</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>(Optional) Metadata 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Payload Data 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Blocks without a rate change generally pass through header
- Blocks with a rate change may need to adjust timestamp and EOB
- Destination Endpoint (DST EP) is updated automatically
- Virtual Channels work in progress
NoC Shell Internals

- AXIS-CHDR bus interfacing logic
  - Payload: Sample data
  - Context: CHDR header & user defined metadata

Backend Interface

Depacketizer
Clock Crossing FIFOs
Packet Flush

AXIS-CHDR to AXIS Payload & Context

Packet Flush & Gate
Packetizer
Clock Crossing FIFOs

AXIS-CHDR to AXIS Payload & Context

Ctrl Port Endpoint
Payload AXI-Stream Bus

- One packet per AXIS-CHDR packet
  - Packets delineated by tlast
  - Maximum size based on MTU
- Typically 32-bits wide, SC16 samples
  - SC16 format: [31:16] Real, [15:0] Imag
- Important notes:
  - User must assert tlast
  - Packet size cannot be larger than the MTU
  - Most blocks hardcode all tkeep bits to 1
Context AXI-Stream Bus

- One packet per AXIS-CHDR packet
  - Packets delimited by tlast
- Tuser value describes each word
  - 0x0: CHDR Header
  - 0x1: CHDR Header + Timestamp
  - 0x2: Timestamp only
  - 0x3: Metadata
- Metadata is user defined
NoC Shell Internals

- **Ctrl Port Endpoint**
  - Converts AXIS-CTRL packets to/from Ctrl Port bus transactions
  - Replaces Settings Bus, bridge available (see ctrlport_to_settings_bus.v)
Ctrl Port Timing Diagram

- Read / write interface for user registers
- 20-bit Address, 32-bit Data
- Bus throttles until ack is asserted
- Supports timed commands (not shown)
Many implementation options for user code
- **HDL**: Verilog, SystemVerilog, VHDL
- **Vivado IP**: FIR, FFT, DDS, CORDIC, Turbo Decoder, etc
- **Vivado Block Diagrams (BD)**: Microblaze
- **Vivado High-Level Synthesis (HLS)**: C and C++

User read/write registers to support configuration, control, and status readback
Gain Block Overview

- Skeleton file passes through samples untouched
- User register logic instantiated but unused
Gain Block Overview

NoC Core

NoC Shell

Payload Data

Context Data (Pass through)

User Register

AXIS-CHDR

AXIS-CTRL

AXIS

CTRL PORT
Gain Block Overview

- Multiply incoming sample data by User Register’s value
Write Custom HDL

- Open `rfnoc-tutorial/rfnoc/fpga/rfccoc_block_gain/rfnoc_block_gain.v`.
- Implement gain in “User Logic” the at bottom on file.
- Requires only a few lines of code:
  - Split incoming samples into I and Q.
  - Multiple I and Q separately with lower 16-bits from the User Register.
  - Create an output sample by concatenating the lower 16-bits of each multiplier result.
  - Do not worry about modifying AXI stream control signals (i.e. tvalid, tready, tlast).
Gain Block Test Bench

- Note: Must have Vivado 2019.1 installed
- `cd rfnoc-tutorial/build`
- `make rfnoc_block_gain_tb`
- Test bench failed... need to update it!
- Edit `rfnoc-tutorial/rfnoc/fpga/rfnoc_gain_block/rfnoc_block_gain_tb.sv`
- Requires only a few lines of codes
  - Read User Register
  - Apply same gain operation in verification code
wire [15:0] i, q, gain;
wire [31:0] i_mult_gain, q_mult_gain;

assign gain = reg_user[15:0];

assign i = m_in_payload_tdata[31:16];
assign q = m_in_payload_tdata[15:0];
assign i_mult_gain = i*gain;
assign q_mult_gain = q*gain;

assign s_out_payload_tdata = {i_mult_gain[15:0], q_mult_gain[15:0]};
assign s_out_payload_tlast = m_in_payload_tlast;
assign s_out_payload_tvalid = m_in_payload_tvalid;
assign m_in_payload_tready = s_out_payload_tready;
Gain Block Test Bench (Answer)

```vhdl
logic [15:0] gain;
logic [31:0] user_reg;
blk_ctrl.reg_read(dut.REG_USER_ADDR, user_reg);
gain = user_reg[15:0];

// Check the resulting samples
for (int i = 0; i < SPP; i++) begin
    item_t sample_in;
    item_t sample_out;
    logic [15:0] i_samp, q_samp;
    logic [31:0] i_mult, q_mult;
    i_samp = send_samples[i][31:16];
    q_samp = send_samples[i][15:0];
    i_mult = i_samp*gain;
    q_mult = q_samp*gain;
    sample_in = {i_mult[15:0], q_mult[15:0]};
    sample_out = recv_samples[i];
...
```
Check Test Bench Again

- cd ~/src/rfnoc-tutorial/build
- make rfnoc_block_gain_tb
- Success!

```
TESTBENCH_STARTED: rfnoc_block_gain_tb

[TEST CASE 1] (t = 0 ns) BEGIN: Flush block then reset it...
[TEST CASE 1] (t = 6450 ns) DONE... Passed
[TEST CASE 2] (t = 6450 ns) BEGIN: Verify Block Info...
[TEST CASE 2] (t = 6450 ns) DONE... Passed
[TEST CASE 3] (t = 6450 ns) BEGIN: Verify user register...
[TEST CASE 3] (t = 7850 ns) DONE... Passed
[TEST CASE 4] (t = 8425 ns) BEGIN: Test passing through samples...
[TEST CASE 4] (t = 9025 ns) DONE... Passed

TESTBENCH FINISHED: rfnoc_block_gain_tb
- Time elapsed: 9025 ns
- Tests Run: 4
- Tests Passed: 4
- Tests Failed: 0
Result: PASSED
```
Generate Bitstream

- `rfnoc_image_builder`
  - Command line tool for building bitstreams
  - Requires YAML file specifying build parameters:
    - Device & Target (e.g. x310, X310_HG -- 1GigE + 10GigE ports)
    - RFNoC blocks (optionally specify block parameters)
    - Endpoints
    - Connections between RFNoC blocks, Endpoints, and other I/O
    - Clock domains

- `rfnocmodtool` automatically created example YAML
  - `rfnoc/icores/gain_x310_rfnoc_image_core.yml`
  - Device & Target: x310, X310_HG
  - RFNoC Blocks: 2xRadio + 2xDDC + 2xDUC + Gain Block

- **Build bitstream:** `make gain_x310_rfnoc_image_core`
  - Requires full Vivado license for most devices
  - Output bitstream located in `uhd/fpga/usrp3/top/x300/build`
Gain RFNoC Block Flow Graph

Autogenerated NoC Core

Transport Interfaces
- 1 GigE
- 10 GigE
- PCIe

RFNoC Blocks
- Radio 0
- Radio 1
- DDC 0
- DDC 1
- DUC 0
- DUC 1
- Gain 0

Signal Source
- Sample Rate: 1M
- Waveform: Cosine
- Frequency: 100k
- Amplitude: 1
- Offset: 0
- Initial Phase (Radians): 0

Throttle
- Sample Rate: 1M

RFNoC Tx Streamer

RFNoC gain
- User Register: 1
- Device Select: -1
- Instance Select: -1

RFNoC Rx Streamer

QT GUI Time Sink
- Number of Points: 1.024k
- Sample Rate: 1M
- Autoscale: No

Static Routing
- (AXIS-CHDR)

CTRL Crossbar
- (AXIS-CTRL)

Stream Endpoints
- Endpoint 0
- Endpoint 1
- Endpoint 2
- Endpoint 3
- Endpoint 4
Gain RFNoC Block Flow Graph

**Autogenerated NoC Core**

- **CHDR Crossbar**
  - Stream Endpoint 0
  - Stream Endpoint 1
  - Stream Endpoint 2
  - Stream Endpoint 3
  - Stream Endpoint 4

- **Ctrl Crossbar (AXIS-CTRL)**

- **Static Routing (AXIS-CHDR)**

**RFNoC Blocks**

- Radio 0
- Radio 1
- DDC 0
- DDC 1
- DUC 0
- DUC 1

**Transport Interfaces**

- 1 GigE
- 10 GigE
- PCIe

**Signal Source**
- Sample Rate: 1M
- Waveform: Cosine
- Frequency: 100k
- Amplitude: 1
- Offset: 0
- Initial Phase (Radians): 0

**Throttle**
- Sample Rate: 1M

**RFNoC Tx Streamer**

**RFNoC Rx Streamer**
- User Register: 1
- Block Args: Device Select: -1
- Instance Select: -1
- QT GUI Time Sink
  - Number of Points: 1.024k
  - Sample Rate: 1M
  - Autoscale: No

**Gain 0**
Gain RFNoC Block Flow Graph

Transport Interfaces

1 GigE
10 GigE
PCIe

Autogenerated NoC Core

CHDR Crossbar

Stream Endpoint 0
Stream Endpoint 1
Stream Endpoint 2
Stream Endpoint 3
Stream Endpoint 4

Static Routing (AXIS-CHDR)

Ctrl Crossbar (AXIS-CTRL)

RFNoC Blocks

Radio 0
Radio 1
DDC 0
DDC 1
DUC 0
DUC 1

Gain 0

RFNoC RX Radio
Number of Channels: 1
Block Args:
Device Select: 1
Instance Select: 1
Sample Rate (Hz): 1M
Antenna Select: RX2
Center Frequency (Hz): 1G
Gain: 32
Automatic Gain Control: Default
Bandwidth (Hz): 0
DC Offset Correction: False
IQ Balance: False

RFNoC Digital Downconverter Block
Number of Channels: 1
Block Args:
Device Select: 1
Instance Select: 1
Frequency Shift (Hz): 0
Output Rate (Hz): 1M

RFNoC gain User Register: 1
Block Args:
Device Select: 1
Instance Select: 1

RFNoC Rx Streamer

QT GUI Frequency Sink
FFT Size: 1.024k
Center Frequency (Hz): 10
Bandwidth (Hz): 1M
RFNoC Framework

GNU Radio
- GRC Bindings (YAML)
- Block Code (Python / C++)

UHD
- Block Description (YAML)
- Block Controller (C++)

FPGA
- Block Test Bench (SystemVerilog)
- Block HDL (Verilog, VHDL, HLS, IP, BD)
RFNoC Framework

GNU Radio
- GRC Bindings (YAML)
- Block Code (Python / C++)

UHD
- Block Description (YAML)
- Block Controller (C++)

FPGA
- Block Test Bench (SystemVerilog)
- Block HDL (Verilog, VHDL, HLS, IP, BD)
Block Description File

- **YAML**

- Tells UHD about block configuration, capabilities:
  - NoC ID
  - Location of Makefile.srcs for HDL source
  - Input clocks
  - Block parameters
  - Block I/O

- **Skeleton file generated by rfnocmodtool**
  - rfnoc-tutorial/rfnoc/blocks/gain.yml
  - Everything already setup for our gain example!
Gain Block Description

NoC Core

NoC Shell

Payload Data

User Register

schema: rfnoc_modtool_args
module_name: gain
version: 1.0
rfnoc_version: 1.0
chdr_width: 64

noc_id: 0xF31E7600
makefile_srcs: ...

AXIS-CHDR

AXIS-CTRL

AXIS

CTRL PORT
Gain Block Description

NoC Core

NoC Shell

Payload Data

User Register

Context Data

(Pass through)

CLOCKS:
- name: rfnoc_chdr
  freq: "[]"
- name: rfnoc_ctrl
  freq: "[]"
- name: ce
  freq: "[]"

AXIS-CHDR

AXIS-CTRL

AXIS

CTRL PORT
Gain Block Description

NoC Core

NoC Shell

Payload Data

User Register

context_data

control:
  sw_iface: nocscript
  fpga_iface: ctrlport
  interface_direction: slave
  fifo_depth: 32
  clk_domain: ce
  ctrlport:
    byte_mode: False
    timed: False
    has_status: False
Gain Block Description

---

**NoC Core**

**AXIS-CHDR**

**NoC Shell**

**Payload Data**

**Context Data** (Pass through)

**AXIS-CTRL**

**CTRL PORT**

---

**data:**

- fpga_iface: axis_data
- clk_domain: ce

**inputs:**

- in:
  - item_width: 32
  - nipc: 1
  - info_fifo_depth: 32
  - payload_fifo_depth: 32
  - format: int32
  - mdata_sig: ~

**outputs:**

- out:
  - item_width: 32
  - nipc: 1
  - info_fifo_depth: 32
  - payload_fifo_depth: 32
  - format: int32
  - mdata_sig: ~
C++ Code
RFNoC block’s software control interface
Create properties to control block
- Access user regs
- Enforce block I/O requirements (e.g. SC16)
- Propagate updates to other blocks
Exposes custom methods
- Example: set_coefficients() for FIR filter RFNoC block
Skeleton file generated by rfnocmodtool
- rfnoc-tutorial/lib/gain_block_ctrl_impl.cpp
- Everything already setup for our gain example!
Python Bindings!

- Python bindings available for most blocks!
- Example for gain block:

```python
import uhd
import gain
import numpy as np

graph = uhd.rfnoc.RfnocGraph("type=x300")
tx_streamer = graph.create_tx_streamer(1, uhd.usrp.StreamArgs("sc16", "sc16"))
rx_streamer = graph.create_rx_streamer(1, uhd.usrp.StreamArgs("sc16", "sc16"))
gb = graph.get_block("0/gain#0")
gb.poke32(0, 1)
graph.connect(tx_streamer, 0, gb.get_unique_id(), 0)
graph.connect(gb.get_unique_id(), 0, rx_streamer, 0)
graph.commit()
num_samps = 4 * tx_streamer.get_max_num_samps()
send_samps = np.array([[0x40004000] * num_samps], dtype="int32")
tx_md = uhd.types.TXMetadata()
tx_md.start_of_burst = True
tx_md.end_of_burst = True
recv_samps = np.zeros((1, num_samps), dtype="int32")
rx_md = uhd.types.RXMetadata()
num_sent = tx_streamer.send(send_samps, uhd.types.TXMetadata())
num_recv = rx_streamer.recv(recv_samps, rx_md, 0.1)
graph.release()
```
RFNoC apps can use only UHD’s C++ API
See: uhd/host/examples/rfnoc-example
RFNoC Framework

GNU Radio
- GRC Bindings (YAML)
- Block Code (Python / C++)

UHD
- Block Description (YAML)
- Block Controller (C++)

FPGA
- Block Test Bench (SystemVerilog)
- Block HDL (Verilog, VHDL, HLS, IP, BD)
RFNoC Framework

GNU Radio
- GRC Bindings (YAML)
- Block Code (Python / C++)

UHD
- Block Description (YAML)
- Block Controller (C++)

FPGA
- Block Test Bench (SystemVerilog)
- Block HDL (Verilog, VHDL, HLS, IP, BD)
GNU Radio Block Code

- C++ or Python
- How does GNU Radio interface to RFNoC?
  - via C++ infrastructure code in gr-ettus
  - Users extend a base class for their RFNoC blocks
  - Some blocks can use base class “as is”
    - Built-in property set methods
- Skeleton file generated by rfnocmodtool
  - rfnoc-tutorial/lib/gain_impl.cc
  - Gain block can use build-in property set methods to set the user register so no edits required!
GNU Radio Companion Bindings

- YAML
- Describes GNU Radio blocks to GRC
- No recompilation!
- Requirement of GNU Radio Companion
- Not strictly necessary for GNU Radio
- More info: wiki.gnuradio.org/index.php/YAML_GRC
- Skeleton file generated by rfnocmodtool
  - rfnoc-tutorial/grc/tutorial_gain.block.yml
  - Already has user register defined with callback so no edits required!
RFNoC Framework

GNU Radio
- GRC Bindings (YAML)
- Block Code (Python / C++)

UHD
- Block Description (YAML)
- Block Controller (C++)

FPGA
- Block Test Bench (SystemVerilog)
- Block HDL (Verilog, VHDL, HLS, IP, BD)
Installation

- Install:
  - cd ~/src/rfnoc-tutorial/build
  - make install

- Program bitstream:
  - Turn on X310
  - uhd_image_loader
    - --args="type=x300" or --args="addr=<USRP IP>"
    - --fpga-path=~/src/uhd/fpga/usrp3/top/x300/build/usrp_x310_fpga_HG.bit
  - Alternative: viv_jtag_program
    - source setup_env.sh in uhd/fpga/usrp3/top/x300
Testing in HW

- Open and run gain flowgraph
  - gnuradio-companion
  - Open ~/src/rfnoc-tutorial/examples/gain.grc
  - Run->Execute or F6
Testing in HW
Final Takeaway

RFNoC is for FPGAs as GNU Radio is for GPPs

<table>
<thead>
<tr>
<th>Feature</th>
<th>RFNoC</th>
<th>GNU Radio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Infrastructure for SDR applications</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Handles data movement between blocks</td>
<td>✔ (AXIS-Based)</td>
<td>✔ (Circular buffers)</td>
</tr>
<tr>
<td>Takes care of boring and recurring tasks</td>
<td>✔ (Flow control, addressing, routing)</td>
<td>✔ (R/W pointer updating, tag handling)</td>
</tr>
<tr>
<td>Provides library of blocks</td>
<td>✔ (Growing)</td>
<td>✔ (Huge)</td>
</tr>
<tr>
<td>Has a graphical front end</td>
<td>✔ (gr-ettus)</td>
<td>✔ (GRC)</td>
</tr>
<tr>
<td>Open Source</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Writes your blocks for you</td>
<td>✗</td>
<td>✗</td>
</tr>
</tbody>
</table>