RFNoC-Interference Manual

David Haberleitner

December 16, 2020

Contents

1 Introduction

This guide shall provide a basic understanding for the RFNoC signal processing chains used to acquire and generate broadband interference on the X300-series SDRs. Furthermore the developed host software and its protocols and file formats are described in detail. If you only need instructions for the software daemon you can skip to Section 3.

Ettus provides access to the internal parts of the FPGA image through the RFNoC standard. This is a set of interfaces and protocols developed by Ettus. The RFNoC system is based on blocks that have to implement these interfaces and protocols. The whole structure is very dynamic as any block can be connected to every other block on the FPGA. The concept of moving digital signal processing to the FPGA using the RFNoC framework is used mainly for two reasons:

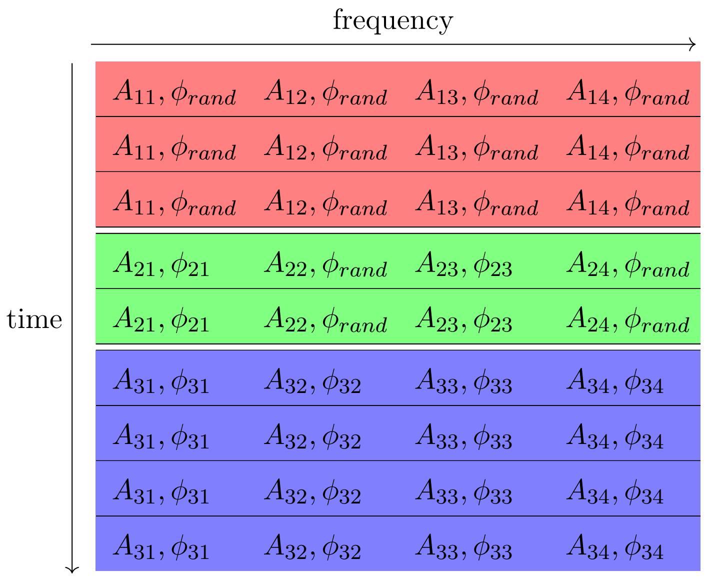
- Conserve Bandwidth between host and SDR (use slower 1GbE instead of 10GbE)
- Reduce system load on the host

The interfaces between the host and SDR are realized as so-called streamers.

1.1 Concepts

1.2 Interference Representation

In this application interference is represented in the frequency domain. The usable bandwidth is divided into N bins which are described by a magnitude and phase value. This coefficient set is called “line” in the following text and can be replayed an arbitrary amount of times to decrease the amount of data transfered between host and SDR. The phase can take on a fixed value for all repeats or random values that change on every replay. This is illustrated in the three blocks in Figure 1.



A_{11}, ϕ_{rand}	A_{12}, ϕ_{rand}	A_{13}, ϕ_{rand}	A_{14}, ϕ_{rand}
A_{11}, ϕ_{rand}	A_{12}, ϕ_{rand}	A_{13}, ϕ_{rand}	A_{14}, ϕ_{rand}
A_{11}, ϕ_{rand}	A_{12}, ϕ_{rand}	A_{13}, ϕ_{rand}	A_{14}, ϕ_{rand}
A_{21}, ϕ_{21}	A_{22}, ϕ_{rand}	A_{23}, ϕ_{23}	A_{24}, ϕ_{rand}
A_{21}, ϕ_{21}	A_{22}, ϕ_{rand}	A_{23}, ϕ_{23}	A_{24}, ϕ_{rand}
A_{31}, ϕ_{31}	A_{32}, ϕ_{32}	A_{33}, ϕ_{33}	A_{34}, ϕ_{34}
A_{31}, ϕ_{31}	A_{32}, ϕ_{32}	A_{33}, ϕ_{33}	A_{34}, ϕ_{34}
A_{31}, ϕ_{31}	A_{32}, ϕ_{32}	A_{33}, ϕ_{33}	A_{34}, ϕ_{34}
A_{31}, ϕ_{31}	A_{32}, ϕ_{32}	A_{33}, ϕ_{33}	A_{34}, ϕ_{34}

Figure 1: Exemplary illustration of interference generation with N=4 frequency bins.

1.3 Implementation

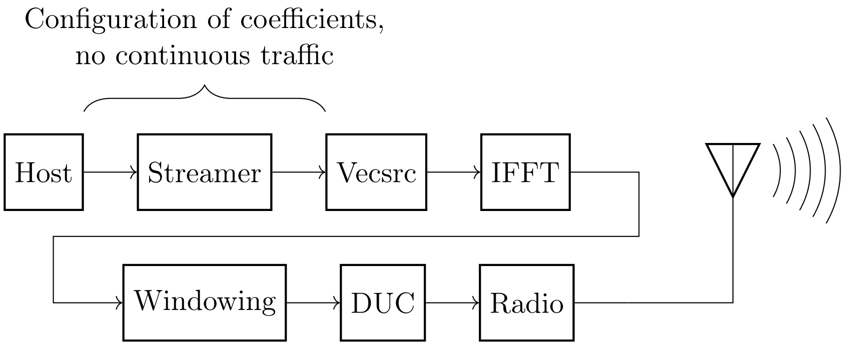


Figure 2: Flow-graph for generating interference patterns on the USRP.

The main parts of the flow graph for interference generation can be seen above. The Vecsrc block continuously repeats spectral coefficients for the FFT and expects coefficient sets as its input. The streamer in this case is not operating in a continuous manner. Instead it sends packets of coefficients to the block. Configuration over register interfaces was also considered and

implemented, but transmission speed was an issue as described later.
The Vecsrc block buffers all incoming coefficient sets in an internal FIFO. It outputs the coefficients n-times and then it moves on to the next set.
Then the repeating coefficients are fed into the inverse FFT. This block already exists in the RFNoC ecosystem.
The resulting signal is then windowed by a special block that keeps a copy of the last vector to fade the two vectors in/out.

Figure 3 shows the output stage of the Vecsrc block. It has two lookup tables for phase and magnitude values. Depending on the phase selection it either uses a random value or the selected value.

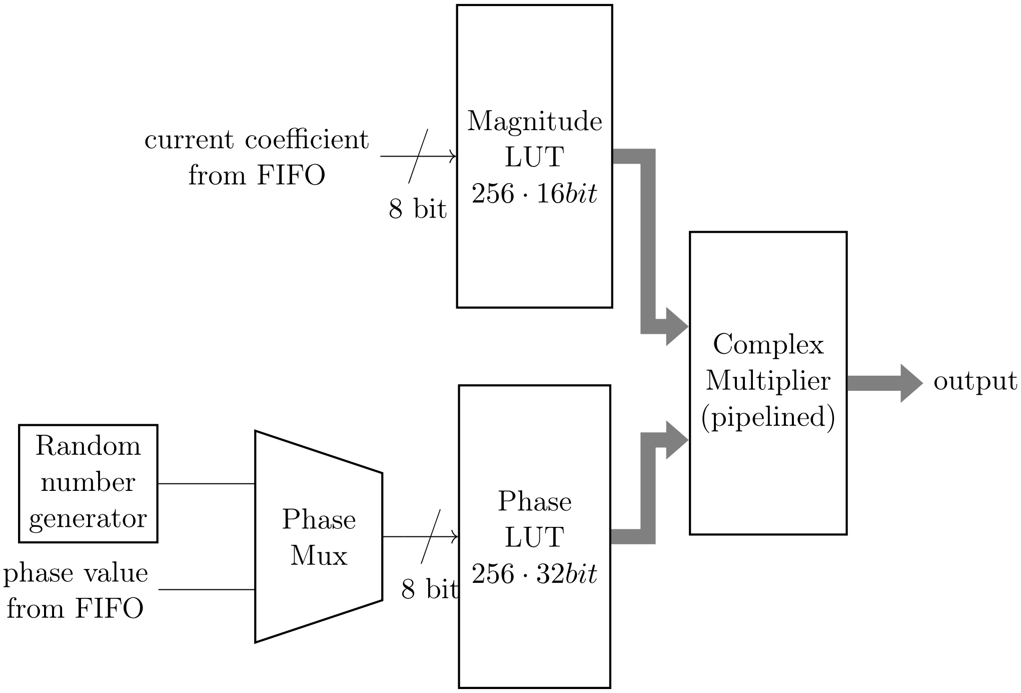

current coefficient from FIFO
8 bit
Magnitude LUT $256 \cdot 16bit$
Complex Multiplier (pipelined)
output
Random number generator
phase value from FIFO
Phase Mux
8 bit
Phase LUT $256 \cdot 32bit$

Figure 3: Structure of the Vecsrc block with random phase generation and logarithmic coefficient lookup.

The user has 3 different lookup-tables/configuration arrays to manipulate the workings of the DSP chain:

- Magnitude lookup-table: Mainly used for converting a logarithmic representation to linear complex samples

- Phase lookup-table: Phase offset that can be applied to each carrier (in a random or user defined way)
- Window function: This real function is used to fade in the new block from the FFT time signal while fading out the old block.

Each of these three memories can be configured at startup of the daemon. The configurations are provided as CSV files and passed to the program via the command line. In addition a scenario file has to be provided. This file represents the magnitude selection and the time each mask is replayed. Additionally phase can also be manitpulated.

There are three possible ways the phase of each carrier can be configured:

No additional phase values are provided for the current set Each carrier gets a random phase from the entries 1-255 in the phase table. This selection changes with every replay/FFT block.

Phase value 0 provided Does behave exactly the same way as above. (Needed to have some carriers at a random phase while others in the same set are fixed)

Phase value 1-255 provided Each value is directly used to select the phase from the internal table.

2 Hardware

As explained before, the system is based on the X300 series of SDR by *Ettus Research*. For the front-end the *UBX-160* daughterboards are used. This provides a usable bandwidth of up to 160MHz, so, for instance, the 2.4GHz ISM band is easily covered. Basically the front-end is not important to the workings of the interference generation/acquistion. It just affects the maximum bandwidth and RF performance.

3 Interference Generation Software

3.1 Installation

UHD 3.15-LTS is a prerequisite for the interference daemon. Installation is described in detail at `https://files.ettus.com/manual/page_build_guide.html`

The following commands should checkout the right branch, build and install UHD into the default directory:

```
git clone https://github.com/EttusResearch/uhd.git
cd uhd
git checkout UHD-3.15.LTS
cd host
mkdir build
cd build
cmake ..
make -j$(nproc)
sudo make install
```

After installing UHD you can use the `usrp_image_loader` tool to install the custom image: First navigate to the directory containing the interference daemon:

```
cd src/software/interference/
```

Create the build directory and change into it:

```
mkdir build && cd build
```

Now we can generate the build files using `cmake`:

```
cmake ..
```

If you installed UHD in a non-standard directory you need to specify it using

```
cmake -DCMAKE_INSTALL_PREFIX=~/uhd_dir ..
```

Finally we can build the daemon:

```
make
```

Now the binary for the interference daemon should be found in the build directory.

3.1.1 Install XML descriptions of the custom RFNoC blocks

For UHD to recognize the custom RFNoC blocks we need to install the corresponding XML files. This is done using the following commands:

```
cd rfnoc_interference/rfnoc-interference
mkdir build
cd build
cmake ..
make
sudo make install
```

3.1.2 Flashing the custom FPGA image

Flash the custom interference image to the X300 using the following command:

```
uhd_image_loader --args="type=x300,addr=192.168.14.2"
↪  --fpga-path rfnoc_interference/images/interference.img
```

Once the new FPGA image has been written to flash, the X300 has to be power-cycled, to apply the new image. After flashing the image you can test if all the blocks are available with the following command:

```
uhd_usrp_probe --args="type=x300,addr=192.168.14.2"
```

You should see the following RFNoC block (our custom ones and the ones provided by *Ettus*):

```
...
|   |   |   * DmaFIFO_0
|   |   |   * Radio_0
|   |   |   * Radio_1
|   |   |   * DUC_0
|   |   |   * vecsrc_0
|   |   |   * FFT_0
|   |   |   * fadewin_0
|   |   |   * FIFO_0
```

If the `uhd_usrp_probe` program shows unidentified blocks (`block_x`) uhd can't find the xml descriptions of the blocks, which should have been installed in Section 3.1.1.

3.2 Help

```
./interference --help
UHD interference daemon allowed options:
  --help                        display this help message
  --rate arg (=200000000)       sample rate the DSP chain operates at
  --phase_file arg (=phase.csv) csv file for phase lookup
  --mag_file arg (=mag.csv)     csv file for magnitude lookup
  --win_file arg (=win.csv)     csv file containing the window coefficients
  --coeff_file arg (=coeff.csv) csv file containing the interference scenario


This application reads scenario files and generates interference directly on the x300 series SDRs.
```

3.3 Example

This section describes the workflow to use interference generation on the X300 board to paint an image in the waterfall spectrum.

Now we can go on and prepare the configuration files for the window, magnitude and phase table. These configurations are provided a CSV file which can be generated using the following commands from the `src/software/scripts` directory:

```
python generate_phasetable.py ../res/phase.csv
python generate_magtable.py -m -60 -f ../res/mag.csv
python generate_window.py -n 64 -f ../res/mag.csv
```

This generates a phase table with unit vectors all around the unit circle, magnitude values from 0dBFs to -60dBFs. The window is set to only occupy 64 of the N=1024 values for a faster transition (see Figure XXXX).

Finally we can generate our scenario from a picture using:

```
python waterfall_image.py -i ../res/emce.jpg -d 3 -r 200e6 -o ../res/emce.csv
```

Now the interference daemon can be started:

```
./interference --coeff_file  ../../scripts/emce.csv --win_file
↪   ../../res/win.csv  --phase_file ../../res/phase.csv
↪  --mag_file  ../../res/mag.csv
```

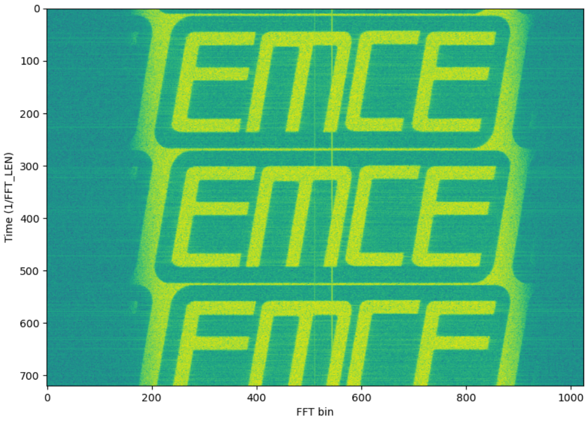


Figure 4: Waterfall plot of the spectrum at 1GHz with a bandwidth of 1.5MHz.

4 Scenario File Formats

The interference daemon supports two different file formats as described in the next sections.

4.1 CSV

The advantages of the CSV file format are that it can be generated from many frameworks that don't support binary file operations and it can also be manipulated in a text editor.
The big disadvantage is that parsing takes many resources. This effect can be observed when operating at very short repetition times as IO/parsing can't keep up and the UHD is showing underflows (U characted gets printed to `stdout`). In this case it is highly recommended to switch to the binary file format.

```
phase,0,1,10,0,...,0
mag,100,150,150,0,...,0,500
```

The CSV file above describes a single spectral configuration that gets repeated 500 times. The phase entry configures all bins except bin 2 and 3 to random phase values. As explained before, if the phase line is omitted, all bins are configured to random phase. Bin 2 is fixed to the value found in the phase LUT at address 1 and the phase of bin 3 is determined by the entry at address 10. The mag entry assigns power to bins 1, 2, and 3. The last value determines the number of repeats.

4.2 Binary file

The interference daemon also supports a binary file format for efficient storage and parsing. As large chunks of the file can be directly copied to memory, performance is a lot better than when using CSV files.

5 Interference Acquisition