RFNoC & Vivado HLS Challenge Mid-Term Update

Team: Rabbit Ears

Team Members:
Andrew Lanez, SPAWAR Systems Center Pacific switchlanez@gmail.com
Alireza Khodamoradi, University of California, San Diego akhodamo@ucsd.edu
Sachin Bharadwaj Sundramurthy, University of California, San Diego sabharad@eng.ucsd.edu

Project Name: ATSC Receiver

Current Status

What have you done so far?
We have practiced a complete RFNoC design flow with simple implementations. We have identified the bottlenecks in the GNU Radio implementation of the ATSC Receiver. We have made progress with our original plan and with a revised plan of porting ATSC Receiver blocks into RFNoC.

How far along are you in implementing the design that you laid out in your proposal?
We have nearly completed the first of the four blocks laid out in our proposal. The ATSC Rx Filter (the frontmost block closest to the USRP) is coded and passing cosim in Vivado HLS. Its verilog code and testbench are in progress of being ported to RFNoC. We have also moved on to other blocks mentioned as “possibilities” in our original proposal. Details in “Changes from the Proposal” section.

Have you spec’d out the blocks you need, exactly what they will do, and how you’ll implement them?
See “Changes from the Proposal” section and implementation plan below.

What testbenches have you completed?
We have developed testbenches for ATSC Rx Filter, ATSC Viterbi Decoder, ATSC Deinterleaver and ATSC Reed Solomon Decoder in Vivado HLS. As far as SystemVerilog testbenches, ATSC Rx Filter has a testbench written in Verilog ready to be ported into RFNoC and SystemVerilog features to be added where needed.

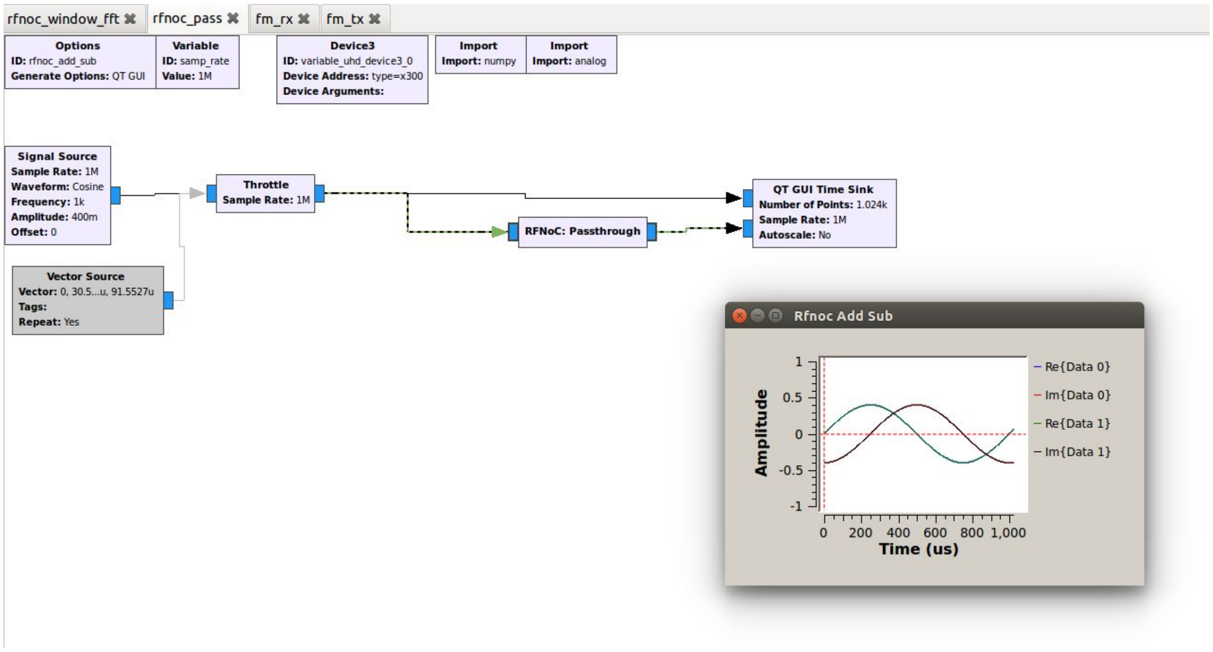
Now that you have spent some time hacking on this, give us some more details regarding your implementation plan.
For each of the modules in ATSC receiver, our implementation process is as follows:

1. Capture live ATSC signal input and output data for each ATSC receiver block being ported from GNU Radio to be used for testbenches.
2. Write the code and testbench for each module using C++ in Vivado HLS.
3. Write NoC block for the generated Verilog file and test NoC shell using SystemVerilog.
4. Generate FPGA image with blocks programmed and integrate into GNU Radio.
5. Upon completing a block, plug it into ATSC Receiver flowgraph and verify raw ATSC signal data coming out of the USRP can still be decoded into video.

The status for each block as per proposal is as follows:

- For development practice purposes, basic modules (Passthrough, Gain, and Negative), we have completed the entire flow.
- For ATSC Rx Filter, we have completed steps 1 and 2. We have achieved an accuracy of 96% matching the output of the GNU Radio implementation. We believe there is loss associated with every floating point math operation and certain needed functions may not be 1:1 portable such as modulo, round, VOLK, and Boost math library but are iterating to achieve higher accuracy.
- For ATSC Viterbi Decoder, ATSC Deinterleaver and ATSC Reed Solomon Decoder, we have generated test input and output data from Gnuradio and have developed test bench in Vivado HLS and are in the process of developing code for each of them. Our Deinterleaver, with integer operations, matches output with 100% accuracy in our Vivado HLS testbench.
- For rest of the modules, we have generated test input and output data and gathered relevant GNU Radio source code files to be studied.

Can you share some screenshots with us of your designs / flowgraphs?
In order to understand how RFNoC works, we initially developed simple modules (pass through, gain and negative) which received a single input and computed operations on the input. The following screenshots show the flowgraphs and outputs of those modules when a sine wave is fed as input. As for our design for the challenge, see “Changes from the Proposal” section.



Options
ID: rfnoc_add_sub
Generate Options: QT GUI

Variable
ID: samp_rate
Value: 1M

Device3
ID: variable_uhd_device3_0
Device Address: type=x300
Device Arguments:

Import
Import: numpy

Import
Import: analog

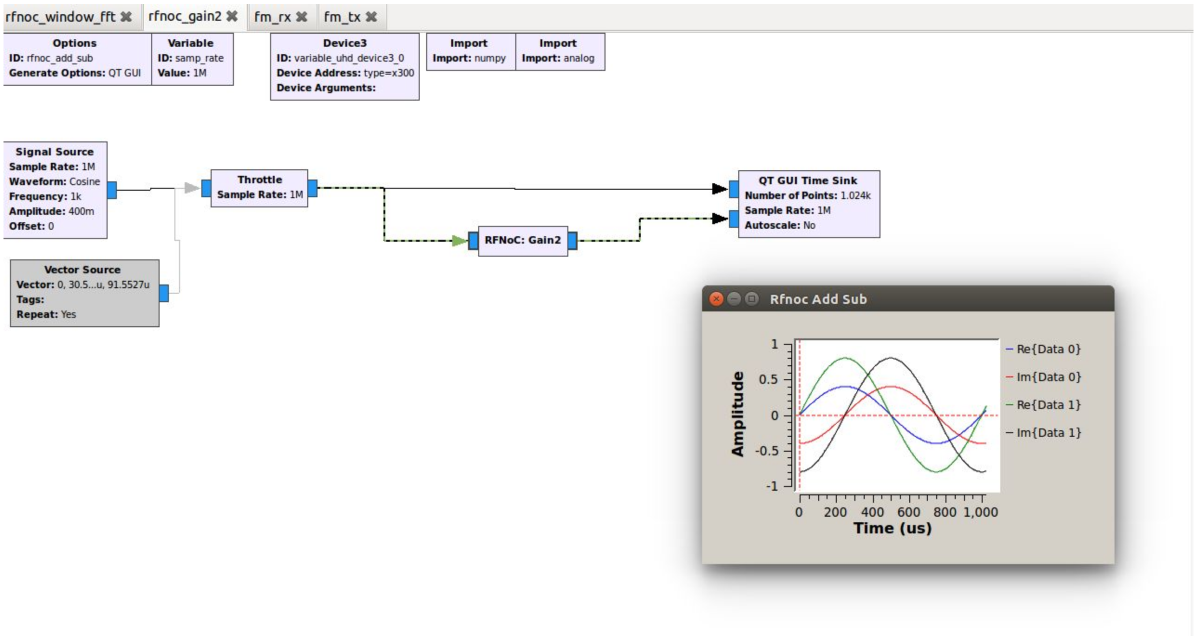
Signal Source
Sample Rate: 1M
Waveform: Cosine
Frequency: 1k
Amplitude: 400m
Offset: 0

Vector Source
Vector: 0, 30.5...u, 91.5527u
Tags:
Repeat: Yes

Throttle
Sample Rate: 1M

RFNoC: Passthrough

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



Options
ID: rfnoc_add_sub
Generate Options: QT GUI

Variable
ID: samp_rate
Value: 1M

Device3
ID: variable_uhd_device3_0
Device Address: type=x300
Device Arguments:

Import
Import: numpy

Import
Import: analog

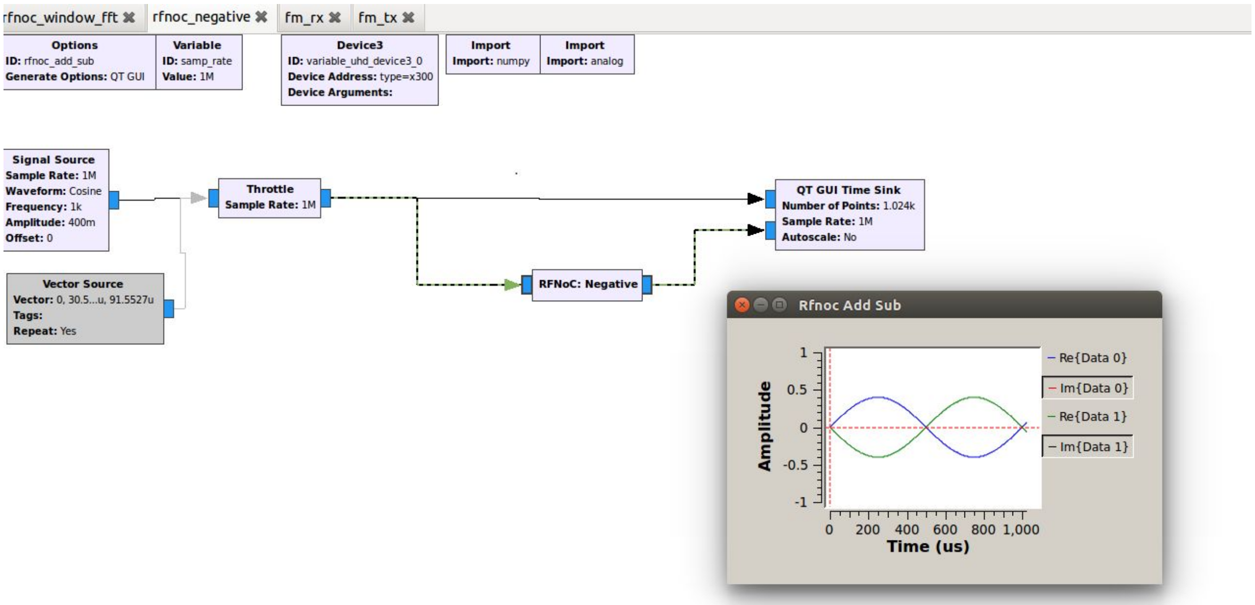
Signal Source
Sample Rate: 1M
Waveform: Cosine
Frequency: 1k
Amplitude: 400m
Offset: 0

Vector Source
Vector: 0, 30.5...u, 91.5527u
Tags:
Repeat: Yes

Throttle
Sample Rate: 1M

RFNoC: Gain2

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



Options
ID: rfnoc_add_sub
Generate Options: QT GUI

Variable
ID: samp_rate
Value: 1M

Device3
ID: variable_uhd_device3_0
Device Address: type=x300
Device Arguments:

Import
Import: numpy

Import
Import: analog

Signal Source
Sample Rate: 1M
Waveform: Cosine
Frequency: 1k
Amplitude: 400m
Offset: 0

Vector Source
Vector: 0, 30.5...u, 91.5527u
Tags:
Repeat: Yes

Throttle
Sample Rate: 1M

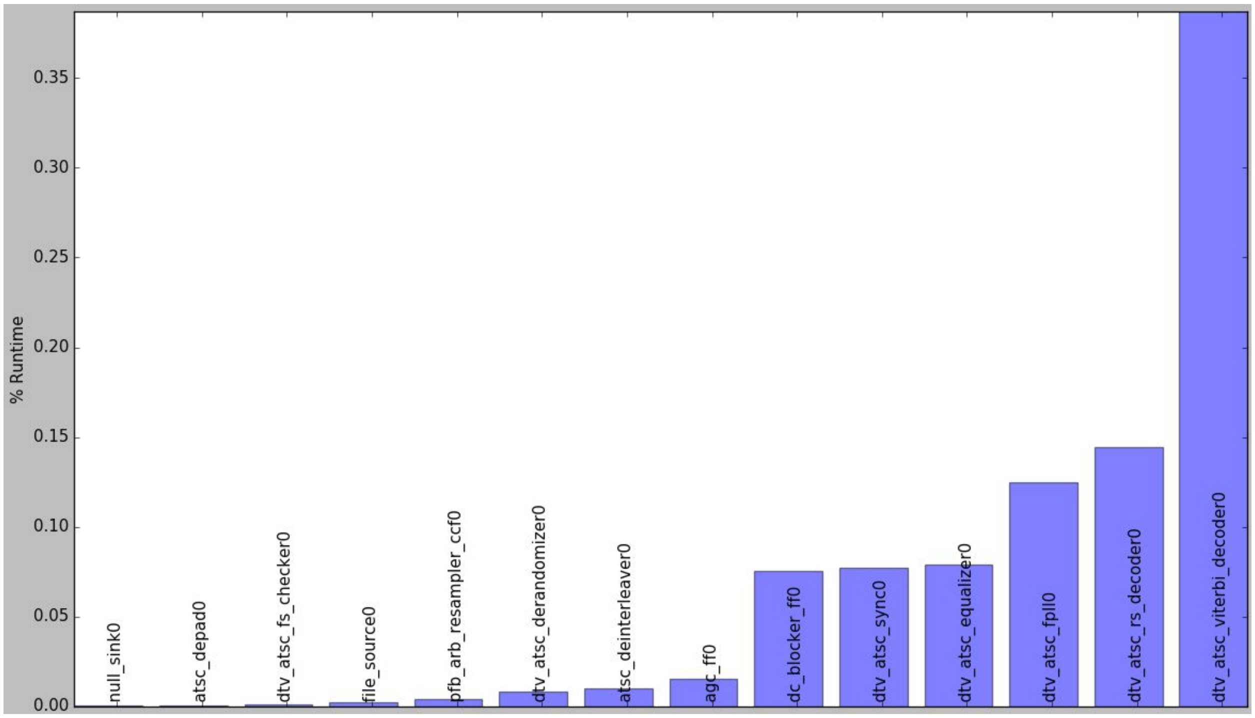
RFNoC: Negative

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

Changes from the Proposal

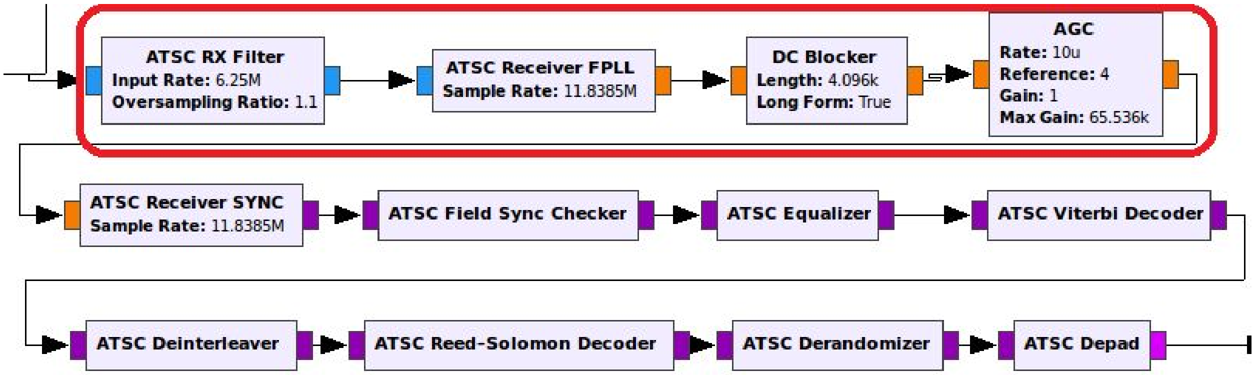
Have you had to deviate from anything in your proposal? If so, why?
How will this impact your final delivered application?

Upon measuring runtime statistics using CtrlPort Performance Monitor, we observed the combined runtime of our originally proposed four blocks ATSC Rx Filter/PFB (1%), ATSC Receiver FPLL (13%), DC Blocker (7%) and AGC (2%) is 22%. Whereas, the top two consumers, ATSC Viterbi Decoder (40%) and ATSC Reed-Solomon decoder (15%), dominate at 55% combined. Two of us are currently pursuing those along with the ATSC DeInterleaver (2%) connected in between to optimize 57% of the runtime.



With ATSC Rx Filter nearly complete, the third one of us will continue the spirit of offloading frontend to hardware by taking on the most significant frontend block: FPLL. That makes for 14% of runtime being optimized at the frontend then, as a stretch, DC Blocker would up it to 21%.

Original Plan (22% runtime consumers)


ATSC RX Filter
Input Rate: 6.25M
Oversampling Ratio: 1.1

ATSC Receiver FPLL
Sample Rate: 11.8385M

DC Blocker
Length: 4.096k
Long Form: True

AGC
Rate: 10u
Reference: 4
Gain: 1
Max Gain: 65.536k

ATSC Receiver SYNC
Sample Rate: 11.8385M

ATSC Field Sync Checker

ATSC Equalizer

ATSC Viterbi Decoder

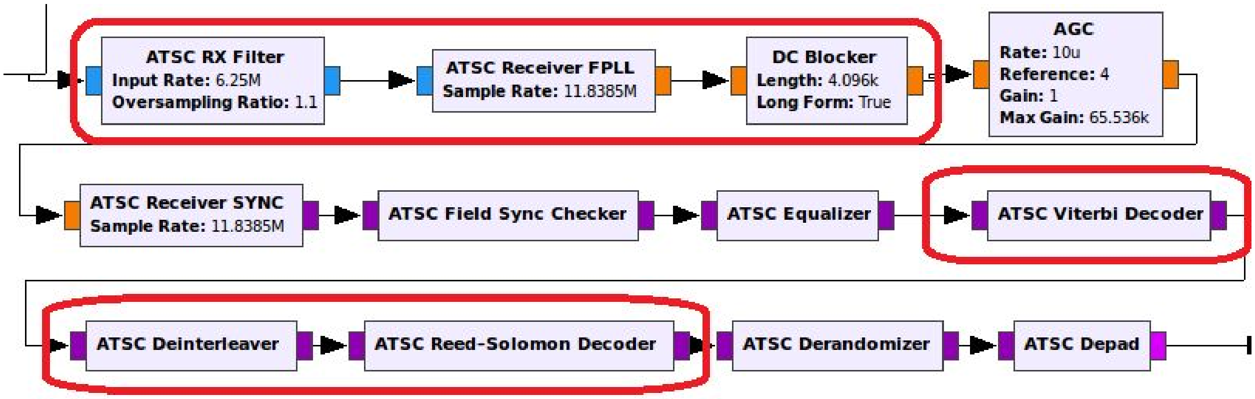
ATSC Deinterleaver

ATSC Reed-Solomon Decoder

ATSC Derandomizer

ATSC Depad

Revised Plan (78% runtime consumers)


ATSC RX Filter
Input Rate: 6.25M
Oversampling Ratio: 1.1

ATSC Receiver FPLL
Sample Rate: 11.8385M

DC Blocker
Length: 4.096k
Long Form: True

AGC
Rate: 10u
Reference: 4
Gain: 1
Max Gain: 65.536k

ATSC Receiver SYNC
Sample Rate: 11.8385M

ATSC Field Sync Checker

ATSC Equalizer

ATSC Viterbi Decoder

ATSC Deinterleaver

ATSC Reed-Solomon Decoder

ATSC Derandomizer

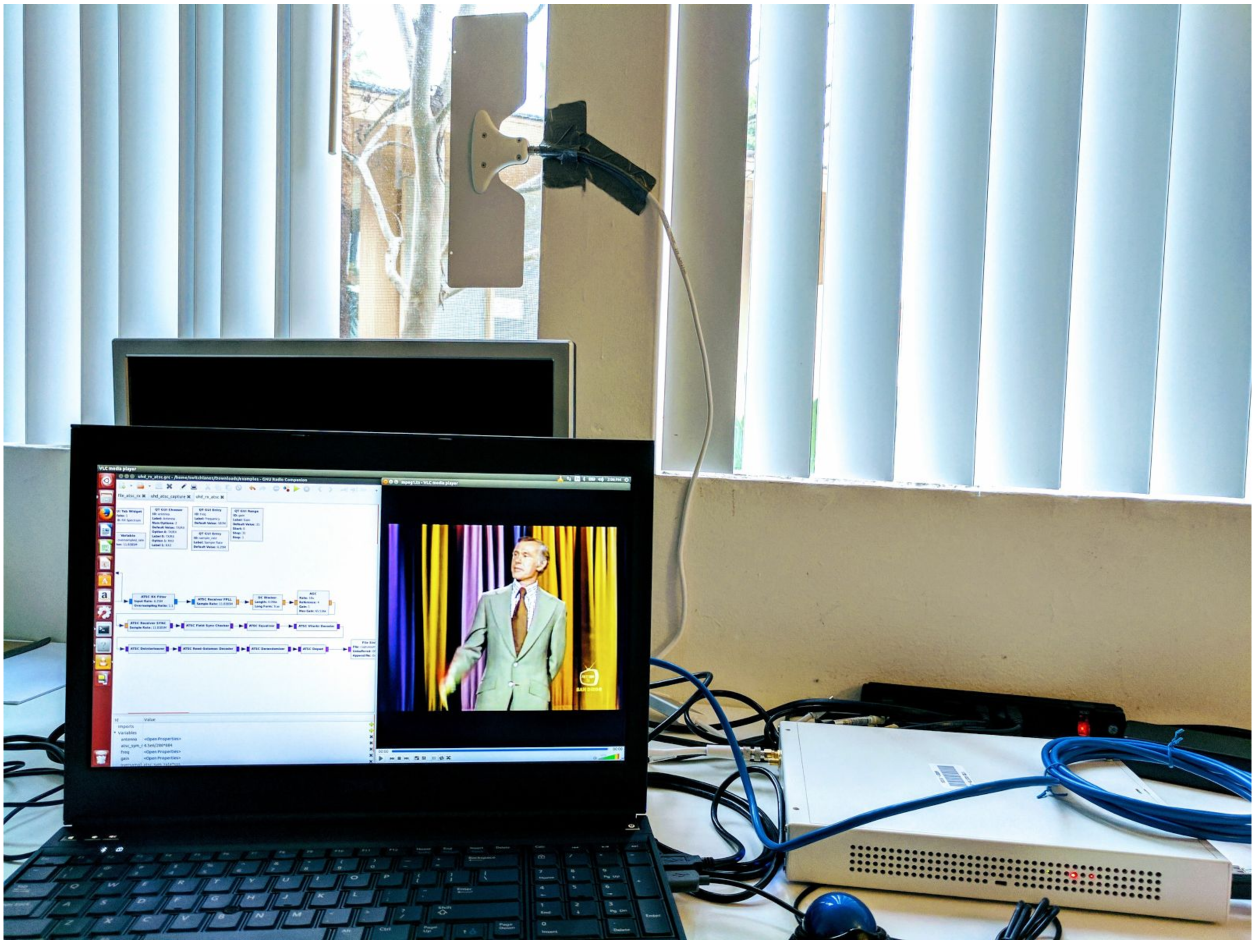
ATSC Depad

Combining frontend with decoders makes for 71% of runtime (78% if we include DC Blocker) being optimized in this revised plan. With work underway on these big consumers, we are confident we will achieve playback of an ATSC live broadcast channel in real-time.

Outstanding Issues

Are you having issues with anything?
User defined parameters in GNU Radio blocks from the flowgraph GUI may be limited in our implementation. We plan to implement our blocks with fixed parameters first, then add more features to the IPs when we can.

Is there something in particular you are really struggling with conceptually, a nasty bug that is holding you back, or a missing feature that is a barrier to your work?
We have been receiving help from Ettus Research/RFNoC team to resolve bugs.



“If it weren’t for Philo T. Farnsworth, inventor of television, we’d still be eating frozen radio dinners.”
-Johnny Carson