RFNoC & Vivado HLS Challenge
Team Rabbit Ears: ATSC Receiver

Andrew Valenzuela Lanez ANDREW.LANEZ@NAVY.MIL
United States Navy

Sachin Bharadwaj Sundramurthy, Alireza Khodamoradi {SABHARAD, ALIREZAK}@ENG.UCSD.EDU
Department of Computer Science and Engineering, University of California, San Diego, La Jolla, CA 92093 USA

Abstract

Creation of custom RFNoC (RF Network-on-chip) blocks that process a received ATSC (Advanced Television Systems Committee) signal is presented. The development workflow of RFNoC blocks may be perceived as complex and intimidating to some. Management of that workflow starting from Vivado HLS (High-Level Synthesis) 2015.4 and proceeding through the RFNoC framework is clarified with procedural steps. Coding and high-level synthesis optimization techniques that were used are discussed.

1. Introduction

The original digital television ATSC library was a contributing factor in the legal founding of GNU Radio. As interesting as it would be to delve into that historic moment, this paper instead details the effort put forth to evolve the venerable ATSC library as GNU Radio evolves with RFNoC. Real time playback of a live ATSC signal processed through the `gr-dtv` ATSC receiver is possible on high-performance computers but not on most commodity computers (Corgan, 2014). This makes ATSC receiver blocks ideal candidates for porting into RFNoC. Computation intensive tasks can be offloaded to FPGA (field-programmable gate array) logic while applying high-level synthesis optimization techniques to improve receiver throughput. This can bring GNU Radio ever closer to achieving real time ATSC playback on a typical commodity computer.

2. Contribution

RFNoC blocks that have been developed under the `atsc_rx` module and verified to run on FPGA hardware are as follows:

- **RFNoC: ATSC RX Filter**
- **RFNoC: ATSC Receiver FPLL**
- **RFNoC: DC Blocker**
- **RFNoC: AGC**
- **RFNoC: ATSC Viterbi Decoder**
- **RFNoC: ATSC Deinterleaver**
- **RFNoC: ATSC Reed-Solomon Decoder**
- **RFNoC: ATSC Depad**
- **RFNoC: ATSC RX Filter-FPLL**
- **RFNoC: DC Blocker-AGC**

All blocks above were built with these integrated features:

- Vivado HLS Source & Testbench
- HDL Testbench
- FPGA Integration
- UHD Integration
- GNU Radio Integration

There are versions of some blocks with partial integration of the Settings Register Bus. There are also versions of blocks optimized for higher throughput. FPGA implementation of these in-progress blocks could not be completed. More specifics on these extra versions will be touched upon throughout this paper.

Source code repository:

`http://github.com/Xilinx/RFNoC-HLS-ATSC-RX`

Video submission for RFNoC and Vivado HLS Challenge:

`https://www.youtube.com/watch?v=iFYgbdf7smg`

Proceedings of the 7th GNU Radio Conference, Copyright 2017 by the author(s).

3. Design

The aforementioned blocks are functional counterparts of existing blocks in the `gr-dtv` library. The rationale for porting those blocks is explained in the following.

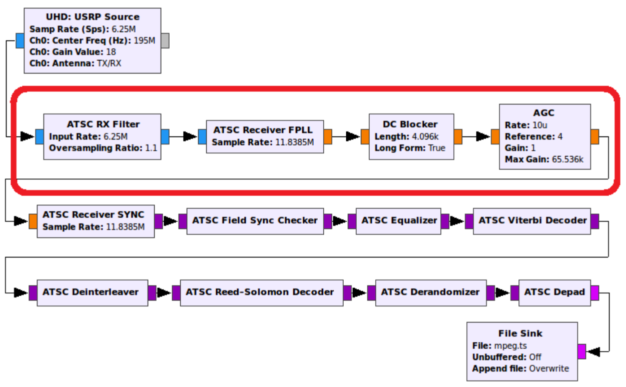


Figure 1. Example flowgraph of ATSC receiver blocks from `gr-dtv`. The four frontend blocks were among those chosen to port into RFNoC.

The Ettus Research USRP X310 packaged with a Xilinx Kintex-7 XC7K410T FPGA was used for this design. RFNoC allows for up to ten user-specified CEs (computation engines or RFNoC blocks) to be programmed onto the XC7K410T. The decision on which blocks to port onto the FPGA hinged on two factors:

Frontend proximity. Porting frontend blocks from software into hardware increases deterministic processing before the datastream falls under the whim of an operating system scheduler. RX Filter, FPLL, DC Blocker, and AGC were selected as shown in Figure 1. RX Filter minimizes unwanted ISI (intersymbol interference) then oversamples and interpolates the signal, FPLL (frequency and phase locked loop) is used for carrier acquisition, DC Blocker removes unwanted DC components, and AGC (automatic gain control) adjusts amplitudes to a reference value within a desired range.

Bottlenecks. Blocks that have higher consumption of runtime resources or cause buffers to fill are ideal candidates for porting. Figure 3 shows the top runtime consumer is from Viterbi Decoder and secondary consumer is from Reed-Solomon Decoder so those blocks were targeted. The top buffer consumer in Figure 2 is from RX Filter.

Deinterleaver was selected to close the link between the Viterbi (or trellis) Decoder and Reed-Solomon Decoder. Those blocks undo forward error correction encoded by the

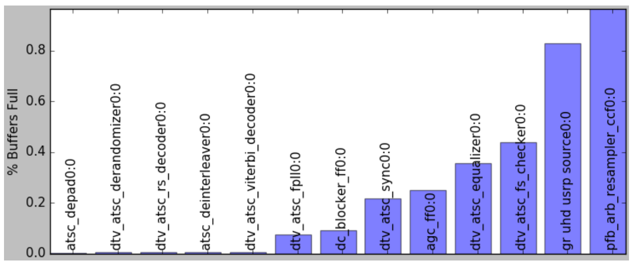


Figure 2. Average buffer usage of `gr-dtv` ATSC receiver blocks captured by ControlPort Performance Monitor.

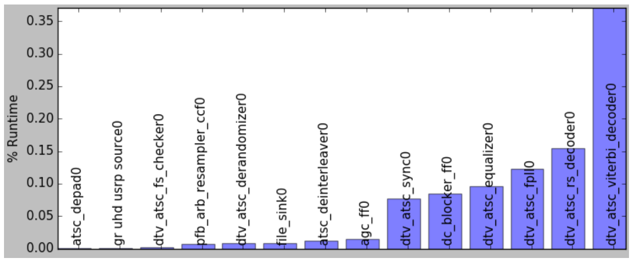


Figure 3. Average runtime usage of `gr-dtv` ATSC receiver blocks captured by ControlPort Performance Monitor.

transmitter. Depad was targeted for its simplicity and to be tried as the first block to complete the RFNoC workflow. It strips extraneous bytes leaving an MPEG video file as the final output.

Two key parameters in this receiver that drive sample rate requirements for all blocks are the 6.25 MHz sample rate coming out of the DDC (digital down coverter) at the receiver frontend and the oversampling ratio of the first block, RX Filter. It was decided that the 6.25 MHz input rate should not be modified to pass in a 6 MHz bandwidth (Figure 4) ATSC channel. Reasonable oversampling ratio values were found to range between 1.1 to 2 for the ATSC receiver software implementation to accumulate enough data to output video for playback. For an initial pass at implementing on hardware, it was decided to define target sampling rates based on the oversampling ratio of 1.1 to make implementation less constrained. Then, time permitting, iterate from there by increasing target rates and fine tune sample rates to match between blocks.

4. Implementation

For each block, workflow started in Vivado HLS 2015.4 then proceeded into RFNoC. When ready, blocks were built into an FPGA image by running the `make X310_RFNOC_HLS_HG` command which called upon Vivado to synthesize the C++ code into a Verilog package

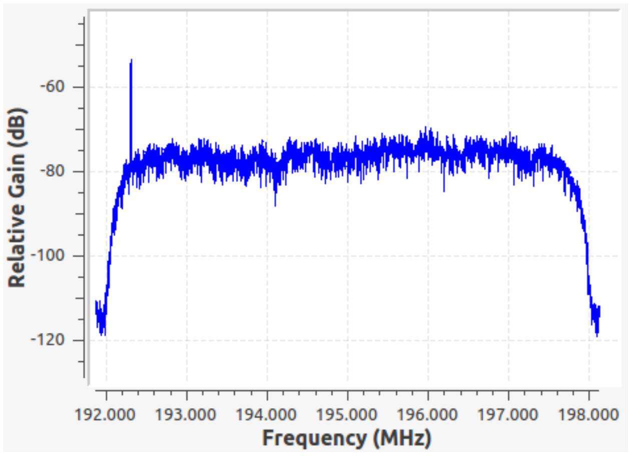


Figure 4. Spectrum of live ATSC broadcast signal received by a 1byone (Figure 14) antenna and USRP X310. Captured at the frontend between the DDC and RX Filter. 8VSB (Vestigal Side Band) modulation with 6 MHz bandwidth and pilot tone approximately 309 Khz above lower edge (Advanced Television Systems Committee, 2011).

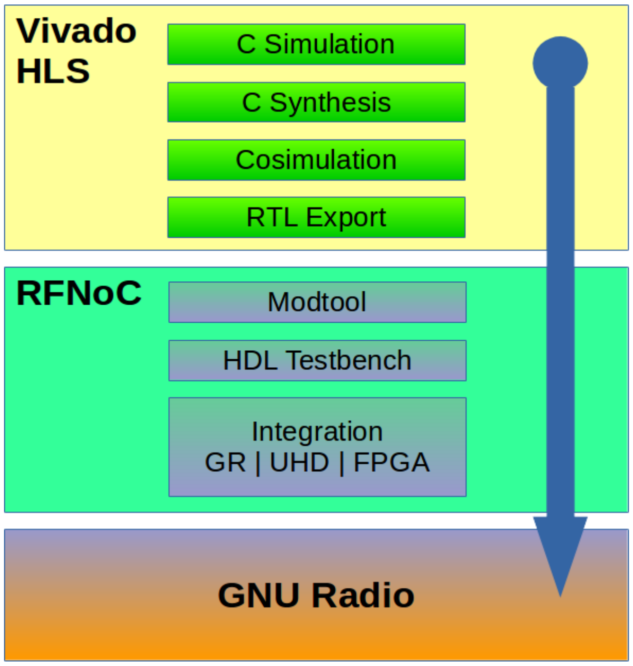


Figure 5. High-level progression of workflow.

and build a bitstream file. The bitstream file would then be programmed to the FPGA using `uhd_image_loader`. Finally, a development iteration ended with testing in GNU Radio (Figure 5).

Workflow in the RFNoC framework is already well documented at (Ettus Knowledge Base, 2017). The following discussion focuses on design implementation using Vivado HLS 2015.4 proceeded by challenges faced during implementation.

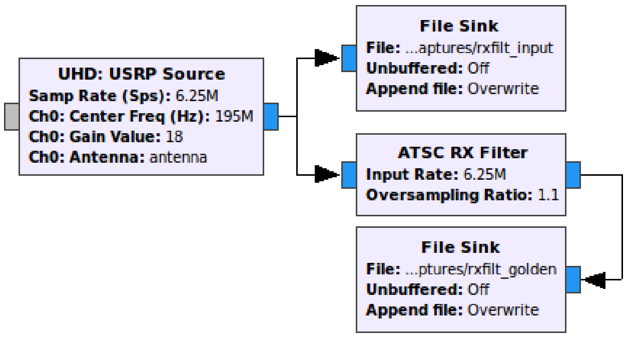
4.1. Workflow in Vivado HLS

Developing HLS Source

Signal processing source files were written in C++ to be synthesizable. Some of the Vivado HLS optimization directives that were used include `#pragma HLS PIPELINE`, `#pragma HLS UNROLL`, `#pragma HLS RESOURCE`, and `#pragma HLS ARRAY_PARTITION`. The tradeoff between optimizing to increase throughput versus optimizing to reduce utilization were kept in consideration. Detailed guidance on optimization techniques can be found at (Xilinx, 2015). Synthesizability was checked using `csynth_design` (C Synthesis in Vivado HLS). Timing could be checked using /tt export_design (RTL Export in Vivado HLS) with the Evaluate Verilog option enabled (though this last step can be time consuming and better left as a final step before FPGA integration in RFNoC).

Developing HLS Testbench

A testbench was written in C++ to send input data to the DUT and compare the output against a golden output using `csim` (C Simulation in Vivado HLS) in the C++ domain. The golden output was captured as a binary file using File Sink on the output of the counterpart or reference block in GNU Radio. Input was captured in a similar fashion with the original input source being a live ATSC signal fed from UHD: USRP Source as shown in Figure 6. In the testbench, multiple function calls to the DUT per test run is encouraged to check the boundaries between returned output data sets and accumulate initiation interval statistics.


UHD: USRP Source
Samp Rate (Sps): 6.25M
Ch0: Center Freq (Hz): 195M
Ch0: Gain Value: 18
Ch0: Antenna: antenna

File Sink
File: ...aptures/rxfilt_input
Unbuffered: Off
Append file: Overwrite

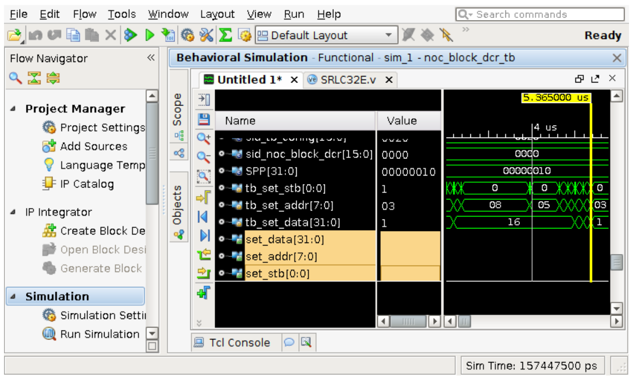
ATSC RX Filter
Input Rate: 6.25M
Oversampling Ratio: 1.1

File Sink
File: ...ptures/rxfilt_golden
Unbuffered: Off
Append file: Overwrite

Figure 6. Input and golden output binary files were captured for use in HLS testbench.

If `csim` showed passing results and `csynth_design` showed the block to be synthesizable, then `cosim` (C/RTL Cosimulation in Vivado HLS) was run to translate the C++ code into RTL (Verilog, VHDL, and/or SystemC) and apply the testbench input stimuli and output checking in the RTL domain. To synthesize the input port of the RX Filter block, for example, into the AXI Stream interface used in RFNoC, the `#pragma HLS INTERFACE axis depth=64 port=in` was used on the top level function of the block. The `depth` parameter has no bearing on synthesis. Instead, it is a control parameter for the `cosim` testbench to know how to size its input FIFO so it matches the RX Filter input array size which does have bearing on synthesis. As a final step before moving on to FPGA integration, Vivado HLS `export_design` with the Evaluate Verilog option enabled was used to check if the design met timing requirements.

Extra sanity checks were sometimes made after modifying the HLS testbench to dump DUT output values into a binary file. The binary output file was then used in a File Source block at the appropriate location in the GNU Radio ATSC receiver example. For example, if the binary file was generated from the RX Filter DUT and its HLS testbench, the UHD: USRP Source and RX Filter blocks in GNU Radio were replaced by a File Source block pointing to that binary file. This way, GNU Radio could perform more checks against the DUT output and report meaningful information such as sync errors. This also tested whether the quality of the binary data was sufficient for decoding into video. Effectively, a block completed this early on in HLS could momentarily bypass RFNoC for basic testing in GNU Radio.



Name	Value
sid_noc_block_dcr[15:0]	0000
SPP[31:0]	00000010
tb_set_stb[0:0]	1
tb_set_addr[7:0]	03
tb_set_data[31:0]	1
set_data[31:0]	
set_addr[7:0]	
set_stb[0:0]	

Figure 7. The HDL testbench uses Vivado Simulator or XSIM. The GUI as shown can be enabled using `make noc_block_[BLOCK NAME]_tb GUI=1`. The logic analyzer feature was used here to observe DC Blocker settings bus signals.

Iterating Between HLS and RFNoC

After a block passed `cosim` and met timing in `export_design`, it was ready to test against the RFNoC HDL testbench. The C++ source files that were fed into `export_design` got converted into Verilog and Xilinx XCI IP source files. Those files were used as the DUT in the RFNoC HDL testbench. The binary input and golden output files used in HLS were converted to ASCII representations using MATLAB (Python would have worked as well) which were then used as input and golden output array variables in the SystemVerilog HDL testbench. If the DUT had bugs revealed by the HDL testbench (Figure 7) or by running its FPGA implementation in GNU Radio, it was debugged in HLS or HDL testbench, re-packaged with `export_design`, then retested. This process was repeated until the RFNoC block implementation functioned as desired in GNU Radio.

4.2. Challenges

The scheduler is a noteworthy feature of GNU Radio dataflow that is not accessible to RFNoC blocks. The software implementation of RX Filter uses the `set_history()` function to recall samples from the previous set of inputs. If not for this feature, the polyphase FIR filterbank–theory from (harris, 2004)–in RX Filter would output starting transients whenever a new set of samples is filtered. `set_history()` prepends the incoming samples with trailing samples from the previous input. Filterbank phase arms are adjusted such that the starting transient overlaps in phase with the previous ending transient. The overlapping transients are then not sent to output (overlap-and-discard method). The RFNoC: ATSC RX Filter block cannot access `set_history()` so previous input samples (specifically the last 18) must be stored internally in FPGA logic then fed into the filterbank before the next set of input samples arrive. Considerations like this must be made when porting existing GNU Radio blocks into RFNoC.

An peculiarity was found while testing the RFNoC: ATSC Receiver FPLL block in GNU Radio. All samples were being output with a gain of $3.276700480546441 \times 10^4$ applied over the expected output values. This almost looked like a 15-bit shift left operation and did not manifest when running the HLS testbench nor the HDL testbench. To resolve this, the FPLL HLS source file was modified to simply divide all outputs by that gain. Of course, the HLS and HDL testbenches had to be updated to multiply that gain back onto the outputs before checking them. The source of this mysterious gain was never found.

The GNU Radio implementation of DC Blocker has a default setting of processing 4,096 samples in and out and using a ”long form” of nested loops in its moving averager. Using these parameters in the initial HLS implementation, the DUT would pass `csim` but `cosim` would timeout after running for more than 24 hours. This may be due to the enormous estimated maximum initiation interval of 251,969,541 clock cycles reported by `csynth_design`. The target initiation interval II in clock cycles can be calculated as

$$\mathtt{II} = \lfloor \frac{\mathtt{f_{CE\_CLK}}}{\mathtt{f_s}} \times \mathtt{n} \rfloor \quad (1)$$

where $\mathtt{f_{CE\_CLK}}$ is the computation engine clock rate, $\mathtt{f_s}$ is the sample rate, and $\mathtt{n}$ is the number of samples processed. Given the CE clock is $\mathtt{f_{CE\_CLK}} = 214\ \mathtt{MHz}$, the required output sample rate determined from studying the receiver in Figure 1 is $\mathtt{f_s} = 11.8385 \times 10^6\ \mathtt{MS/s}$ (megasamples per second), and the number of output samples per function call to DC Blocker is $\mathtt{n} = 4,096$, applying these values to equation 1 the target initiation interval becomes $\mathtt{II} = 2,314\ \mathtt{clock\ cycles}$. To minimize clocks, it was decided to reduce the delay line length and not use ”long form” so that less nested loops were used with two delay lines instead of four in the moving averager. The DC Blocker software implementation could still synchronize with and decode a live ATSC signal with minumum delay line length 128 (which changed the II target to 2,315) and ”long form” disabled. After applying these changes to the HLS source code, initiation interval reduced to 207,365 clock cycles. Optimization directives `#pragma HLS ARRAY_PARTITION` with `cyclic factor=16` applied to both delay lines and `#pragma HLS UNROLL factor=16` applied to nested loops brought initiation interval down to 31,671 clock cycles. The optimizations were still not enough to meet the target initiation interval. Setting the unroll and cyclic factor parameters to 64 reduced initiation interval to 6,543 with no errors or warnings in HLS. However, image build resulted in a critical warning on timing and the RFNoC implementation functioned erratically in hardware. This was a recurring issue and further discussion on this is in Section 5.

The Viterbi block underwent a similarly radical improvement in optimization. An earlier implementation of the Viterbi algorithm relied on many loops and nested loops. It had a 2,659,376 clock cycle initiation interval though the target was 163,124 clock cycles. Vivado HLS was found to not be unrolling loops that greatly would benefit from being pipelined. Calls to the same function within the loops may have been a reason the loops could not be unrolled. These functions were copied many times and numbered to match loop iterations and loops were manually unrolled. This enabled more pipelining and reduced initiation interval to 138,920 which met the target.

RFNoC: FIFO blocks were used (Figure 13) to compensate for mismatching data rates from hardware to software. As more RFNoC blocks were developed, more RFNoC: FIFO instances were needed, pushing total block count closer to the 10 CE limitation on the USRP X310. A solution was to combine blocks at the HLS level. RX Filter arbitrarily outputs either 60 or 61 pairs of float (IQ) samples at a time. FPLL requires one pair of float samples in to output a single float at a time. Hence, no dynamic FIFO was needed in between when combining the two into one RFNoC: ATSC RX Filter-FPLL block. DC Blocker, however, requires a rigid 128 samples in and out. AGC only requires one sample in and out so it was combined with DC Blocker at the HLS level to implement RFNoC: DC Blocker-AGC. Combining blocks at the HLS or User IP level to increase available CE slots came at the cost of a slight decrease in overall sample rate but overhead incurred from packetization (Figure 8) was eliminated.

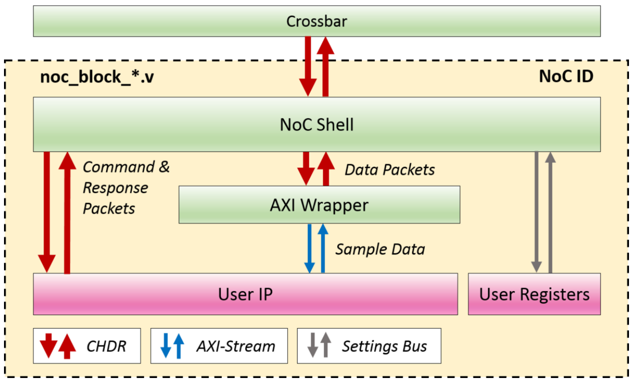


Figure 8. CHDR (compressed header, an Ettus-specific protocol) packets pass between AXI Wrapper, NoC Shell, and Crossbar and between User IP and NoC Shell. Information and image retrieved from (Ettus Knowledge Base, 2017).

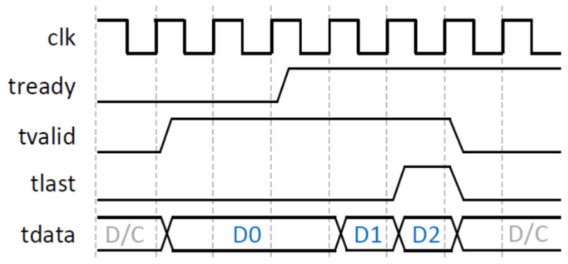


Figure 9. AXI Stream signals. Image retrieved from (Ettus Knowledge Base, 2017).

Implementation of the settings register bus would be re-

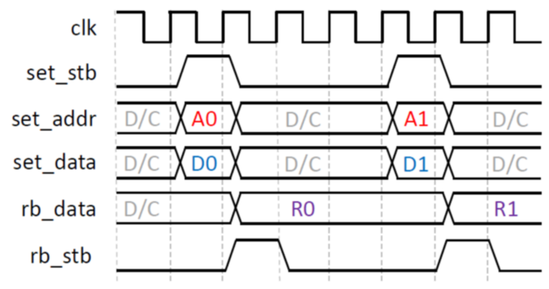


Figure 10. Settings Bus signals. Image retrieved from (Ettus Knowledge Base, 2017).

quired to make parameters such as the oversampling ratio in RX Filter or delay line length in DC Blocker programmable by the user. Provisions for this bus were implemented in HLS source, HLS testbench, NoC block, HDL testbench, and GNU Radio and UHD integration. Values were not observed to be propagating to the settings registers in Vivado Simulator (Figure 7). It was questionable whether the settings bus ports were getting synthesized correctly from HLS source files. Guidance for synthesizing AXI Stream interface ports is well documented in (Xilinx, 2017). The `#pragma HLS INTERFACE axis` directive is used to synthesize ports that match the AXI stream signals used by RFNoC in Figure 9. Similar documentation for synthesizing the settings bus signals shown in Figure 10 could not be found. The `#pragma HLS INTERFACE ap_stable` and `#pragma HLS INTERFACE ap_none` directives described in (Xilinx, 2015) were experimented with to attempt synthesis of the `set_addr`, `set_data`, `set_stb` settings bus ports from HLS source files. They are data port interface directives, however, have no associated I/O protocol. Regardless, the settings registers were implemented on FPGA and tested in GNU Radio and did not function as desired. This is why settings bus is marked ”with partial integration” in Section 2.

5. Results

This section presents the resulting BRAM (Block RAM), DSP, FF (flip-flop), and LUT (lookup table) utilization and sample rates of the RFNoC blocks. Live performance of the RFNoC blocks running in GNU Radio is then discussed.

In some instances, the RFNoC image build process resulted in a critical warning for timing not being met. This was despite Vivado HLS 2015.4 giving no indication of timing not being met from `csynth_design`, `cosim`, nor

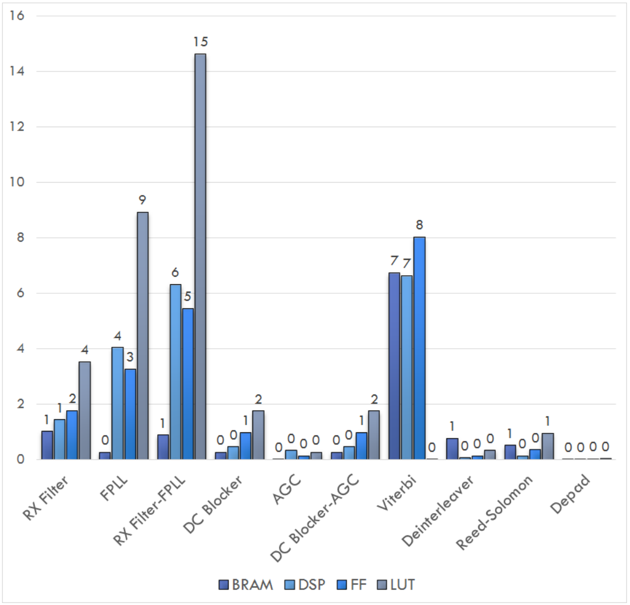


Figure 11. RFNoC block percent utilization based on values from Table 1.

`export_design` with the Evaluate Verilog option enabled. Reed-Solomon v2 was one such block that exhibited this behavior. It was significantly optimized to exceed its target throughput. No issues were reported from Vivado HLS 2015.4 but the image builder reported critical warnings on timing and the block functioned erratically on hardware (junk data). This was despite HDL testbench showing passing results. For blocks behaving like this, sometimes one optimization had to be removed at a time between image builds to isolate the offending optimization responsible for the critical warning or erratic behavior. This put an invisible limit on allowable optimizations and made target sample rates in a working hardware implementation unreachable. However, before iterating back optimizations, it was always worth testing an image on hardware; on a few fortunate ocassions the image functioned as desired despite critical warnings.

Synchronization and decoding into playable video (Figure 12) and audio was achieved but, because target sample rates were unmet (Table 2), video playback was not approaching real time as originally desired. Placing RFNoC: FIFO blocks in front of RFNoC blocks as shown in Figure 13 allowed for processing of a live ATSC signal into playable video and audio but with five to ten times more latency than the pure software version of the receiver and with chunks of samples being dropped. A couple seconds of audio and video would render from a decoded output data size ranging anywhere between 20 to 50 megabytes.

Table 1. RFNoC block utilization area reported by Vivado HLS C Synthesis in terms of units. These blocks were verified to function properly in hardware.

RFNoC Block	BRAM	DSP	FF	LUT
RX Filter	16	22	8960	8929
FPLL	4	62	16542	22641
RX Filter-FPLL	14	97	27639	37158
DC Blocker	4	7	4909	4427
AGC	0	5	661	641
DC Blocker-AGC	4	7	4909	4427
Viterbi	107	102	40754	50374
Deinterleaver	12	1	675	823
Reed-Solomon	8	2	1798	2390
Depad	0	0	130	77
Total Available	1590	1540	508400	254200

For the sake of finding new avenues to improve sample rates, very brief experimentation was done with Vivado HLS 2017.1. Optimizations to RX Filter that failed to meet timing in version 2015.4 were passing in version 2017.1. A single afternoon of further optimizing RX Filter in version 2017.1 netted a ten-fold improvement in sample rate over the 2015.4 optimized version. That huge leap in improvement was accomplished in an extremely short time span relative to the month spent optimizing RX Filter in version 2015.4 to net only a 40% improvement. Optimizing DC Blocker in Vivado HLS 2017.1 netted similar improvements. AGC was below its target sample rate in 2015.4 but exceeded its target by a factor of three in 2017.1. This all made Vivado HLS 2017.1 seem very promising. However, attempting to build an image with Vivado 2017.1 linked to RFNoC failed because RFNoC did not support the Xilinx IP generated by Vivado 2017.1 HLS. Regardless, this finding shows great promise if future revisions of RFNoC were to support newer and seemingly improved versions of Vivado HLS. It seems to be common opinion that coding in pure Verilog is the best way to meet timing and throughput requirements. That gap appears to be closing rapidly as improved versions of Vivado HLS are released.

6. Lessons Learned

A shotgun approach to this project was used by selecting many blocks to port into RFNoC with hopes of hitting the bullseye. With Viterbi being the largest resource consumer and bottleneck and its RFNoC implementation working and

Table 2. RFNoC block sample rates were calculated by applying initiation intervals reported by Vivado HLS Cosimulation and the 214 MHz CE clock value to equation 1 without the floor operator and solving for f_s. Target output rates were calculated by multiplying the 6.25 MHz receiver input rate with a factor determined by the internal functionality of each respective block. RX Filter samples per output is an average of its arbitrarily resampled output and an irrational number. Blocks denoted with ”v2” were optimized for higher throughput. Vivado HLS `export_design` with the Evaluate Verilog option enabled showed RX Filter v2, DC Blocker v2, and AGC v2 did not meet timing (Reed-Solomon v2 did) and all functioned erratically on hardware.

RFNoC Block	Samples per Output	Max Initiation Interval	Target Output Rate (MS/s)	Actual Output Rate (MS/s)	Target Met	Works on FPGA
RX Filter	60.61	34427	11.8385	0.3768		✓
RX Filter v2	60.61	3157	11.8385	4.1087		
FPLL	1	424	11.8385	0.5047		✓
RX Filter-FPLL	60.61	69563	11.8385	0.1865		✓
DC Blocker	128	15964	11.8385	1.7159		✓
DC Blocker v2	128.00	1938	11.8385	14.1342	✓	
AGC	1	26	11.8385	8.2308		✓
AGC v2	1.00	6	11.8385	35.6667	✓	
DC Blocker-AGC	128	19421	11.8385	1.4104		✓
Viterbi	2256	138920	2.9596	3.4753	✓	✓
Deinterleaver	188	1895	2.9596	21.2306	✓	✓
Reed-Solomon	188	96592	2.9596	0.4165		✓
Reed-Solomon v2	188	9955	2.9596	4.0414	✓	
Depad	188	71	2.9596	566.6479	✓	✓

meeting the target throughput, it can be considered that a bullseye was hit. Multiple other blocks struck right around the mark. Simpler blocks such as Deinterleaver and Depad far surpassed target sample rates and only came about to be developed during times when little to no progress could be made on more challenging blocks such as RX Filter, DC Blocker, and Viterbi. This approach may have been inefficient at times with efforts being spread thin across many blocks. It is uncertain whether a more focused, sniper approach on fewer blocks would have yielded better results.

Image builds took two to three hours on higher performance workstations in the UCSD SeaLab. But with the majority of build attempts being done from home on an HP laptop, it took about seven hours per build. It wasn’t until the final weeks of the project that a novice Linux user discovered the `indicator-cpufreq` package for Ubuntu. Build times were then reduced to about five hours. It also helped to convert a Windows Surface Pro 3 tablet to dual boot with Ubuntu 16.04.1 so two images could be built simultaneously. Further, multiple instances of Vivado HLS 2015.4 could be running on both machines while both were also building images. Resourceful tricks to streamline pro-

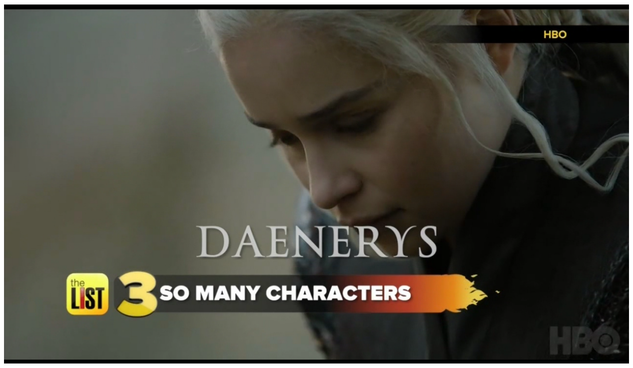
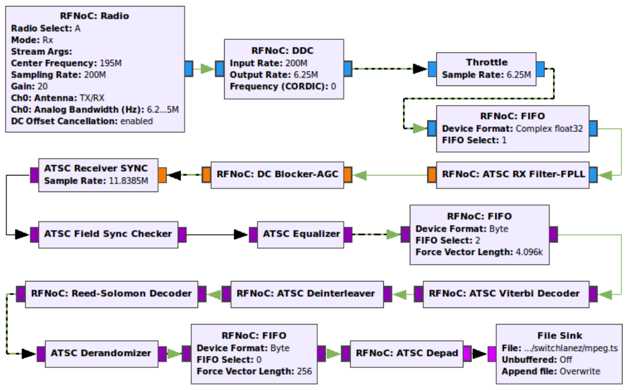


Figure 12. Screenshot of video from a live 195 MHz ATSC signal (ABC San Diego KGTV Channel 10) that was processed through implemented RFNoC blocks.



RFNoC: Radio
Radio Select: A
Mode: Rx
Stream Args:
Center Frequency: 195M
Sampling Rate: 200M
Gain: 20
Ch0: Antenna: TX/RX
Ch0: Analog Bandwidth (Hz): 6.2...5M
DC Offset Cancellation: enabled

RFNoC: DDC
Input Rate: 200M
Output Rate: 6.25M
Frequency (CORDIC): 0

Throttle
Sample Rate: 6.25M

RFNoC: FIFO
Device Format: Complex float32
FIFO Select: 1

RFNoC: ATSC RX Filter-FPLL

RFNoC: DC Blocker-AGC

ATSC Receiver SYNC
Sample Rate: 11.8385M

ATSC Field Sync Checker

ATSC Equalizer

RFNoC: FIFO
Device Format: Byte
FIFO Select: 2
Force Vector Length: 4.096k

RFNoC: ATSC Viterbi Decoder

RFNoC: ATSC Deinterleaver

RFNoC: Reed-Solomon Decoder

ATSC Derandomizer

RFNoC: FIFO
Device Format: Byte
FIFO Select: 0
Force Vector Length: 256

RFNoC: ATSC Depad

File Sink
File: .../switchlanez/mpeg.ts
Unbuffered: Off
Append file: Overwrite

Figure 13. Flowgraph with RFNoC blocks implemented and RFNoC: FIFO blocks and a Throttle block were placed to mitigate dropping of samples. To reduce losses, it is advisable to capture the DDC output first to a file, then feed that to the rest of the receiver.

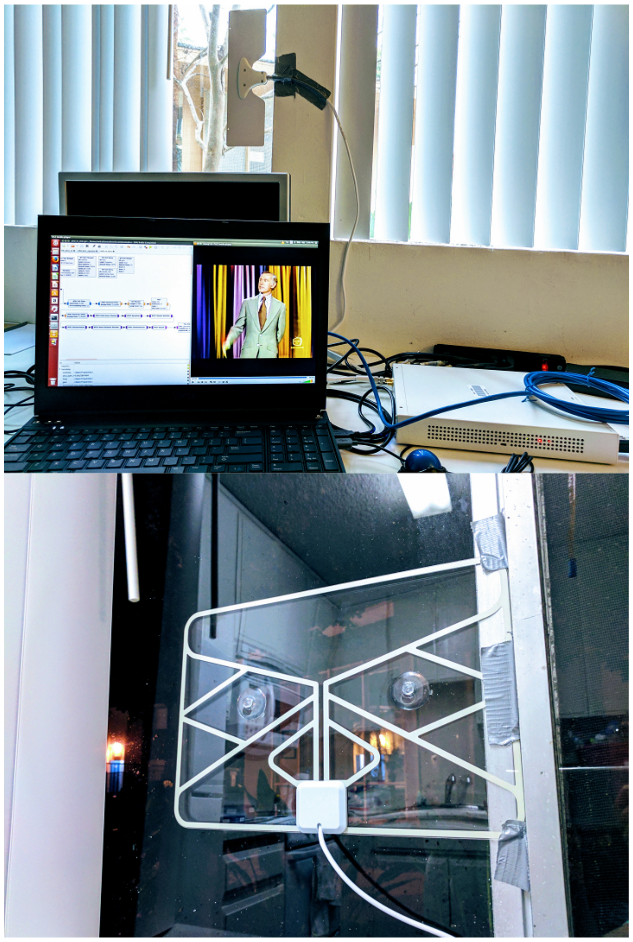


Figure 14. Top: Mohu Leaf antenna connected to a USRP X310. This antenna was eventually returned to the vendor due to poor reception. Bottom: 1byone antenna had consistent reception of the 195 MHz-centered ATSC channel at this specific position on the window and was used for primary testing.

ductivity are fruitful for projects of this nature.

Initial difficulties with getting any signal in the primary test location was resolved by changing from a 25-mile advertised range Mohu Leaf to a 50-mile advertised range 1byone antenna. The latter came bundled with an amplifier but performed better with the amplifier switched off. There was one specific position on the window (Figure 14) and one channel (ABC Network) at 195 MHz where the received signal (Figure 4) was acceptable. Testing had been done very early on in the project in a different location with a higher grade antenna resulting in higher reception quality signified by a smaller receiver output file playing back a longer duration of video. But access to that location was very limited and those tests had been done before any RFNoC blocks were developed. More testing in the better location with better antenna is a future action item.

It was realized partway through the project that real time playback was an ambitious stretch goal. Although real time playback was not achieved in this iteration of development, HLS optimizations made it possible for several blocks to meet their respective targets and for all blocks to process data into playable video. As more mature versions of Vivado HLS become supported by RFNoC, synthesis and reporting will improve to help realize real time processing and playback of an ATSC signal assisted by RFNoC.

References

Advanced Television Systems Committee, ATSC. Doc. a/53 part 2:2011. In *ATSC Digital Television Standard Part 2: RF/Transmission System Characteristics*, pp. 9, 24. Advanced Television Systems Committee, Retrieved from `http://www.atsc.org/wp-content/uploads/2015/03/a\_53-Part-2-2011.pdf`, 2011. [Online; accessed 30-June-2017].

Corgan, J. M. Johnathan Corgan on Twitter: ”Real time, over-the-air, pure software ATSC digital television reception with #usrp and #gnuradio. `http://t.co/t46BPG28gB`”. Retrieved from `http://twitter.com/jmcorgan/status/491702698052296705`, 2014. [Online; accessed 30-June-2017].

Ettus Knowledge Base, Ettus Research. In *Getting Started with RFNoc Development*, Retrieved from `http://kb.ettus.com/Getting\_Started\_with\_RFNoC\_Development`, 2017. [Online; accessed 30-June-2017].

harris, f. In *Multirate Signal Processing for Communication Systems*, chapter 7.5. Prentice Hall, Inc., Upper Saddle River, NJ, 2004.

Xilinx, Inc. Ug902 (v2015.4). In *Vivado Design Suite User Guide: High Level Synthesis*, Retrieved from `http://www.xilinx.com/support/documentation/sw\_manuals/xilinx2015\_4/ug902-vivado-high-level-synthesis.pdf`, 2015. [Online; accessed 30-June-2017].

Xilinx, Inc. Ug1037 (v4.0). In *Vivado Design Suite: AXI Reference Guide*, Retrieved from `http://www.xilinx.com/support/documentation/ip\_documentation/axi\_ref\_guide/latest/ug1037-vivado-axi-reference-guide.pdf`, 2017. [Online; accessed 30-June-2017].