Real-Time Implementation of MIMO Systems using OpenWiFi

Nathan Schimpf
Dept. of Electrical and Computer Engineering
University of Louisville
Louisville, United States
ntschi01@louisville.edu

***Abstract*— This paper summarizes contributions toward a real-time jamming-resistance wireless receiver. Previous research is considered, which prototyped a jamming-resistant receiver (JRRx) using Ettus Research software-defined radios (SDRs). The OpenWiFi project provides a hardware-software implementation of a wireless access point (AP) using Xilinx Zynq devices. Methods to develop using OpenWiFi and preliminary changes to the project are summarized. Future work is also prescribed, with the intent of showing the projects' viability for real-time system development.**

Keywords—Jamming Mitigation, Wireless Systems, OpenWiFi, Verilog

I. Introduction

In recent years, techniques to mitigate jamming and severe interference have become more viable; with the use MIMO technology, prototypes for mitigating these signals have been developed such that there is no prerequisite knowledge beyond outside of what is already used in commercial WiFi devices.

With the release of OpenWiFi at the end of 2019, tools are available for the implementation of a completely open-source WiFi device. PHY-layer processing is implemented for Orthogonal Frequency Division Multiplexing (OFDM) systems using programmable logic fabric. The project supports a number of Xilinx System-on-Chip (SoC) devices, and provides guidelines to port the project to unsupported devices.

This paper considers the efforts toward implementing a real-time jamming-mitigation system based on the OpenWiFi project. Development was performed using the Analog Devices RF System-on-Module (SoM), a development module that combines a programmable RF front-end (AD9361) with a Xilinx Zynq device (Z7035). Though complete implementation was not achieved, the workflow and initial modifications are discussed.

The structure of this paper is as follows: section 2 describes the technique and requirements of the BJM algorithm; section 3 overviews the architecture of the OpenWiFi project and processes to modify the project HDL; and section 4 specifies the contributions finished within the time-frame of this independent study, as well as the future work to continue JRRx development.

II. Overview of JRRx Design

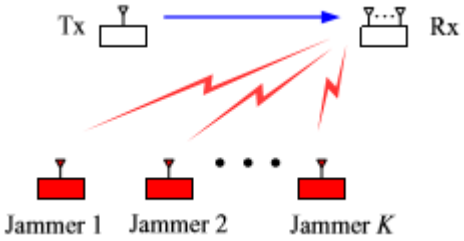

Figure 1: Anticipated Jamming Scenario

To alleviate jamming signals we consider a transmission system in Figure 1; note that the figure depicts a key requirement for the algorithm to work, **the number of receiving antennas on the AP must be greater than the total number of jamming antennas.** Given this scenario, the JRRx system can be effectively implemented. The JRRx design is illustrated in Figure 2, and broken into a synchronization module, Fast-Fourier Transform (FFT), and BJM algorithm.

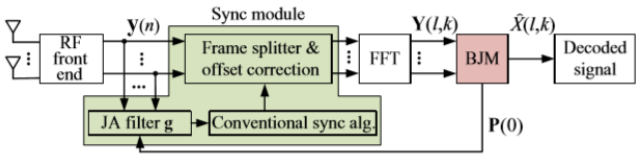

Figure 2: Jamming-Resistant Receiver Design

A. Synchronization Module

The synchronization module is responsible for identifying and correcting frames received as time-domain samples. It is separated into the development and application of a jamming alleviation filter designated **g;** the use of standard carrier and sampling-frequency offset estimation techniques; and the splitting of data segments into frames and correcting offsets based on these estimations.

To minimize computation efforts, **g** is defined for two cases: where a frame has already been found and a BJM filter applied to that frame, and where no frame has been found for a predetermined length of time. When a frame has already been found, **g** re-uses the last spatial filter for the centered subcarrier P(0). This re-use is acceptable because of the properties of OFDM systems: each subcarrier's channel can be estimated as flat-fading, and adjacent subcarriers have highly correlated channel responses. As a result, selecting P(0) can provide a coarse alleviation of the jamming signals.

When no frame has been previously detected, g must be calculated based on a singular-value decomposition (SVD). Taking the SVD of the received total power for each sample n provides the matrices U, Σ, and V, representing the column space, diagonal weights, and row space of the received power, respectively.

$$[U\,\Sigma\,V] = SVD(\sum_{n=1}^{N_S} y(n)y(n)^T) \quad (1)$$

Using the column space U, g is selected as the i^{th} column U_i that produces the largest cross-correlation with the received signal.

$$g = U_i, i = \text{indexOf}\left(\left(\max\left[\sum U_i{}^*(m)^T Y(m+n)\right]\right)\right) \quad (2)$$

Based on previous testing, at least one column of U will be able to completely cancel jamming signals given a frequency-flat, noise-negligible channel.

Standard synchronization techniques include course and fine- carrier frequency offset estimation (CFO) using the training fields built into the WiFi frame preamble, as well as sampling frequency offset (SFO) estimation using the pilot subcarriers in a WiFi signal.

In short, CFO correction in WiFi systems relies on predetermined signals that are added to the start of each frame. These signals are a pseudorandom, repeated pattern, and therefore can be used to estimate delay with via autocorrelation. The estimated phase offset â can be described as

$$\hat{a} = \frac{1}{N}\sum_{m=0}^{M} y_m{}^* y_{m+N} \quad (3)$$

with parameters defined in Table 1.

Table 1: Carrier Frequency Offset Parameters

Parameter	**Coarse CFO Value**	**Fine CFO Value**
N: Training Sequence Length	16	64
M: Last-Half of Training Field Length	5*N-1=79	N-1 =63
K: Total Training Field Length	10*N-1 =159	2*N-1 =127

Based on the estimated â, carrier offset can be corrected with Equation 4.

$$\hat{y}_{CFO} = ye^{-jk\hat{a}} \quad (4)$$

The SFO estimation is much more complex, as it accommodates both drifting over time and any residual offset not handled by CFO correction. The phase offset φ at the k^{th} subcarrier of symbol l is defined as

$$\phi_{l,k} = 2\pi l\left(\epsilon_r T_u f_c - \frac{N+N_g}{N}\epsilon k\right) \quad (5)$$

with parameters defined in Table 2.

For brevity, the three-step correction technique is described: first, the SFO is approximately corrected using an equation similar to Equation 4, with $\hat{a}$ instead being the second term of Equation 5. Residual offset is estimated and corrected based on a cross-correlation of known pilot values and their estimated channels; residual offset and the estimated offset for each symbol are updated based on a 4-symbol averaged autocorrelation of the pilot carriers.

Table 2:Sampling Frequency Offset Parameters

Parameter	**Definition**
$\hat{\epsilon}_0$: initial estimated frequency offset	$\hat{a}_{LTF}$
T_u: OFDM symbol duration	4 μs
f_c: carrier frequency	Dependent on WiFi channel selected
N: samples per OFDM symbol	64
N_g: guard samples per OFDM symbol	16
ϵ_r: residual frequency offset	$\epsilon - \hat{\epsilon}_0$

B. Blind-Jamming Mitigation Algorithm

Working with synchronized frames in the frequency domain, a spatial filter P can be developed such that the original signal can be accurately estimated as $\hat{X}$ via

$$\hat{X} = P^T Y \quad (6)$$

Deriving the Mean-Squared Error for this estimation yields a definition for this spatial filter

$$P = [E(YY^T)]^+ E(YX^T) \quad (7)$$

Where + represents the pseudoinverse of the bracketed matrix. To more accurately develop a spatial filter for each subcarrier, the filter will be an average of the intended subcarrier and the two subcarriers on each side. Note that P is the set of reference signals for coarse CFO estimation.

$$P(k) = \left[\sum_{(l,k')\epsilon \mathrm{P}} Y(l,k')Y(l,k')^T\right]^+ \left[\sum_{(l,k')\epsilon \mathrm{P}} Y(l,k')Y(l,k')^T\right] \quad (8)$$

$$k'\epsilon[k-2, k+2] \quad (9)$$

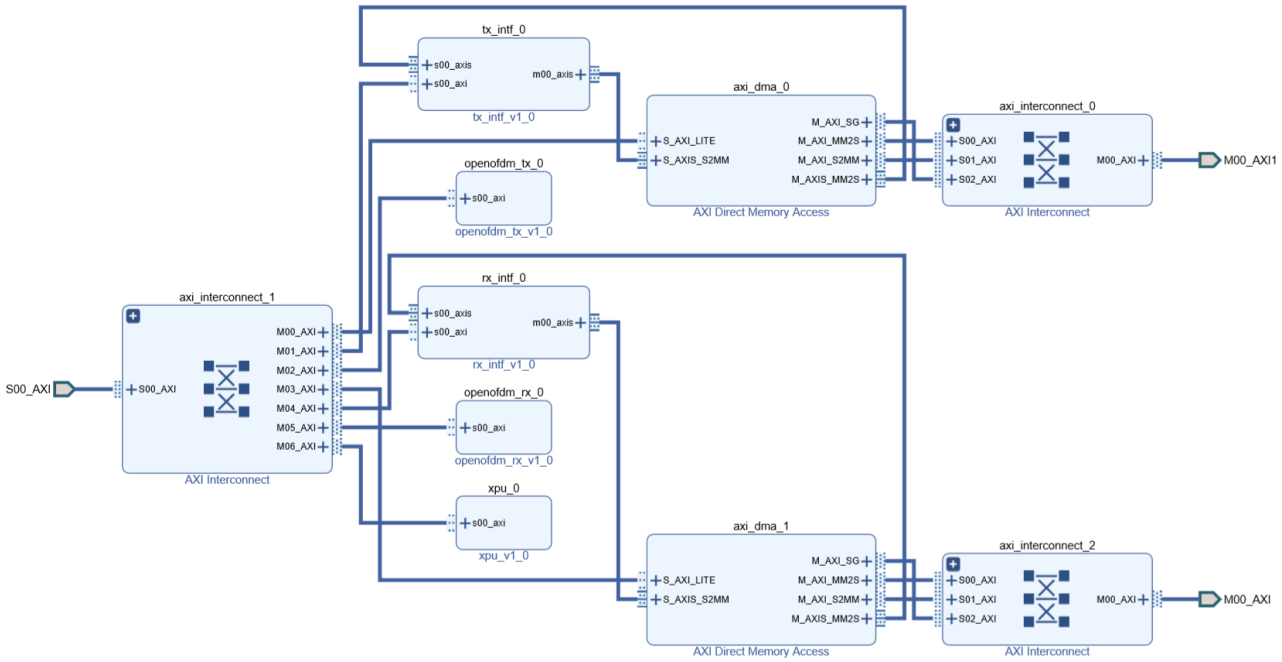

Figure 3: Condensed OpenWiFi IP Architecture

III. OpenWiFi Architecture

The OpenWiFi project incorporates a hierarchy of tools to build a modifiable, open-source WiFi access point for SoC SDRs.

At the highest level, OpenWiFi extends the reference Linux and HDL images for the SoM. Using Linux packages to interpret and generate MAC-layer packets, OpenWiFi performs several core operations; each of these operations have an associated top-level IP used in the HDL portion of the project, as well as kernel modules that are installed in the Linux system to allow parts of the IP to be configured.

- *rx_intf*, *tx_intf*: maps ADC data between the AD9361 baseband I/Q samples; converts clock domains from ADC (40 MSps) to the baseband (20 MSps). Combines processed data into MAC-layer packets, handling the DMA interface to Linux.
- *OpenOFDM_rx*, *openOFDM_tx*: handles most PHY layer processing, including packet detection; frequency offset correction; channel estimation; symbol alignment; and FEC coding. Translates between I/Q samples and MAC-layer packets.
- *Xpu*: performs time-sensitive MAC-layer controls, that need a responses faster than possible via DMA to the ARM processors; the name is a misnomer, as it is a state machine more than an integrated processor. Functions include redundant packet filtering; MAC-layer packet assembly; channel contention controls; and basic service set timing synchronization.

Table 3: OpenWiFi Hardware Development Scripts

Script	Location	Description
Prepare_adi_lib.sh	Openwifi/Openwifi-hw	Initializes Analog Devices IP library.
Prepare_adi_board_ip.sh	Openwifi/Openwifi-hw	Initializes Analog Devices reference HDL design.
Openwifi.tcl	Openwifi/Openwifi-hw /boards/$BOARD_NAME	Builds OpenWiFi Vivado project; called from Vivado, not a Linux command-line.
Sdk_update.sh	Openwifi/openwifi-hw/boards	Exports SDK project to workspace
Boot_bin_gen.sh	Openwifi/user_space	Generates boot image
Prepare_kernel.sh	Openwifi/user_space	Generate Linux image for OpenWiFi
Make_all.sh	Openwifi/driver	Compiles IP device drivers
Wgd.sh	Openwifi/user_space	Installs IP drivers

Table 4: System-on-Chip Boot Files

File	File Use	Description
system.hdf	Intermediary	Hardware Description File;
System_top.bit	Intermediary	Bitstream generated by Vivado, dictates routing to FPGA
Fsbl.elf	Intermediary	First-Stage Bootloader; combines *system.hdf* and *system_top.bit* to dictate the hardware design when booting an SoC device.
u-boot.elf	Intermediary	Universal bootloader; executable that opens the Linux operating system when the SoC is powered up.
Zynq.bif	Intermediary	Boot image source file; direct concatenation of the generated bitstream and *.elf* files
BOOT.BIN	Output – saved to boot drive (/BOOT)	Final boot image; contains all hardware configurations and bootloaders necessary for the FPGA to be configured on system powerup.
Devicetree.dtb	Output – saved to boot drive (/BOOT)	Device tree blob; provides a compiled binary describing hardware resources (processors, functional units, etc.) to the Linux operating system.
Uimage	Output – saved to boot drive (/BOOT)	Universal image; provides a minimal operating system for Xilinx SoCs.
**.ko*	Output – installed to Linux kernel	Kernel modules; drivers that enable the linux operating system to interact with the programmable logic for each IP.

Each IP within OpenWiFi has some level of configuration controlled by the Zynq processors, including the selection of antennas, supported packet types, and digital gain applied to samples. All parameters are set within the IP via AXI bus.

To limit the FPGA resources used while maintaining real-time performance, OpenWiFi balances a number of tradeoffs. Most relevant to this project are the restrictions in the CFO correction, symbol alignment, and higher-level PHY controls. At current, OpenWiFi only supports single-input, single-output (SISO) configurations. While the SoM supports multiple antennas for transmit and receive, simultaneous use of more than one antenna will be have to be implemented before starting on the jamming-mitigation techniques.

A. *Modifying the HDL*

Because OpenWiFi is intended to be modified, it includes a number of scripts to make the process accessible to users. Scripts are provided to correctly initialize the git repositories for the Analog Devices IP library, OpenWiFi IP Library, and to generate the Vivado project for a hardware platform (i.e. the SoM). A summary of each notable building script is provided in Table 3.

The HDL developed for OpenWiFi is hierarchically built using packaged Vivado IP. Because of this, any changes that need to be made to one of the core functions in OpenWiFi have to be made by editing and re-packaging the function's IP.

Once changes have been made to the Vivado project, generating the bitstream is necessary. However, the SoM does not directly use the bitstream, instead initializing boot image (BOOT.BIN). To build all necessary hardware changes, several intermediary files need to be generated in a way accessible to OpenWiFi's building process. This is accomplished by exporting the hardware and bitstream locally; launching an SDK instance; and copying SDK files to an accessible directory (via *sdk_update.sh*). Following this, the boot image can be built (via *boot_bin_gen.sh*) and copied to the SoM's storage. A summary of the intermediary and output files is provided in Table 4.

IV. Contributions toward Realtime-BJM

In the span of this semester, contributions covered a proof-of-implementation by enabling the development of MIMO techniques and providing a simple averaging filter of the two streams. Starting the project meant successfully building the boot image, drivers, and other output files from source; verifying the SoM could boot; and verifying that devices could connect to the SoM's WiFi network.

Once the system was built as-is, version control was added via a local git repository. Because of the project complexity, it was important to have a method for reverting and testing changes. Commits would follow as an IP was successfully created, modified, or integrated.

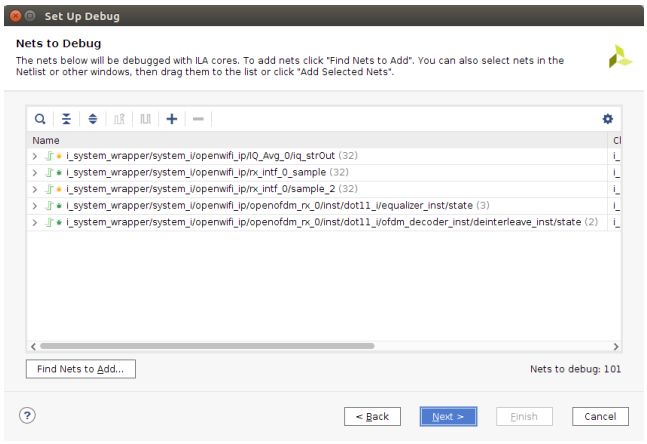

Figure 4: Selecting Synthesized Nets in Vivado Setup Debug

Created and modified IP were tested at various points. Local testbenches were written to simulate the IP before packaging; once integrated with the main project, debug nets and hardware manager sessions would be used to verify the IP operated as-expected in the context of the system; finally, externally verifying by connecting to the SoM's network ensured no system issues came from interacting the HDL with other project sections.

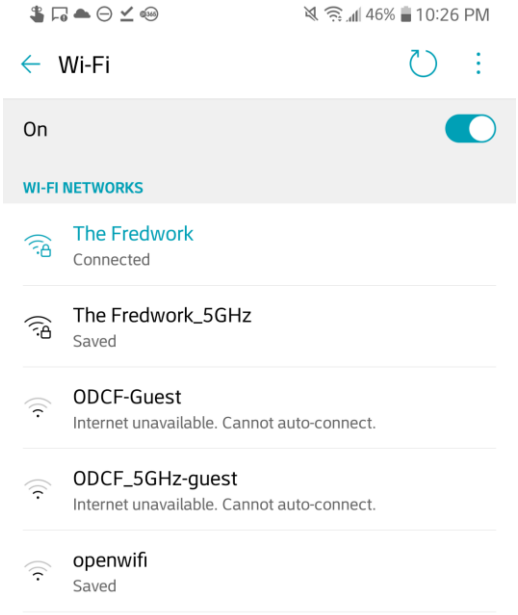

Figure 5: Accessing the OpenWiFi Network

A. *Simultaneous I/Q Stream Access*

Before developing filters, OpenWiFi's SISO limitation needed to be overcome. This single-input limitation is imposed by *rx_intf*, which selects one of the two antenna I/Q streams to provide for baseband processing. Because there is the potential for crossing clock domains, the changes to *rx_intf* needed to be more carefully considered.

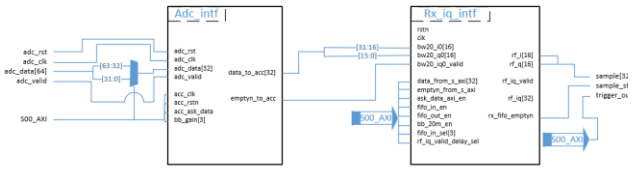

Figure 6: rx_intf PHY Architecture

While the *rx_intf* handles both MAC and PHY-layer data, the necessary changes only consider the PHY layer. Data from the RF front-end is received in a 64-bit word containing padded I/Q samples from each antenna, according to Figure 8. Internal variables are set according to addressed data from the AXI bus, including the antenna selection flag. Based on this flag, one of the two sets of samples is passed through the *adc_intf*, which applies a digital gain and decimates data to a lower clock domain, from 40 MSps to 20 MSps. This data is accessed by *rx_iq_intf*, which handles any drift between the clock domains and passes baseband samples to the remaining IP.

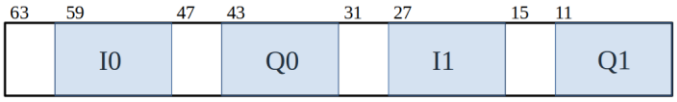

Figure 8: adc_data Format

At the heart of both *adc_intf* and *rx_iq_intf* are FIFO blocks, which provide the stability in data transfer between the two clock domains. Because of this, providing both I/Q streams effectively requires the two modules to be duplicated, along with the removal of multiplexing prior to the *adc_intf*. Final changes are denoted in Figure 7.

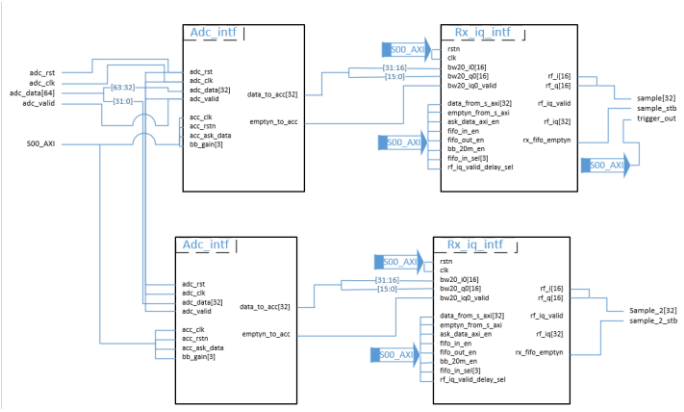

Figure 7: rx_intf PHY Final Design

As this was the first modification to the OpenWiFi project, some challenges were faced with the development process. Learning the workflow to properly edit and re-package an IP; re-synthesize the IP with the appropriate context; and setup debug tools for the system to validate the design came with a number of mistakes.

The initial design plan would re-use the AXI data for selecting the antenna, adding an option at the driver level to perform different types of MIMO-based filtering. After testing in a hardware manager session, I/Q samples from both antennas returned null. This approach was deemed unnecessary and complex. Potential causes of failure came from limitations on the size of the AXI data used; as the antenna selection flag was read from part of a specific register received, it is assumed the data passed by AXI was limited to a single bit in that register by driver-level controls.

Figure 9: Hardware Verification of Reading both I/Q Streams

The final design ignored antenna selection, instead buffering both antenna streams as I/Q samples. Once successfully built in Vivado with debug nets, a boot image was generated and the board I/Q samples were validated. This was accomplished by opening a hardware manager session in the Vivado project, after the SoM had booted from its own storage. Results from the corrected design are shown.

B. Averaging Filter

Once a successful implementation to receive both I/Q streams was achieved, a simple filtering IP was developed. Because of constraints on time, an averaging filter was selected to illustrate the process of designing, testing, and integrating a new IP for the design.

From discussions in [1], it is seen that combining antenna streams outperforms the selection of any individual antenna. Assuming this to hold for averaging as well, the filter will always improve performance; however, an external enable is included to provide user control in future designs.

The design was initialized as a new project in Vivado, with simple code. The each antenna stream was separated into its I/Q components; these components would then be added and averaged before being combined as a single data stream. If the filter is not enabled, the first antenna stream is passed through unaltered.

Simulating the IP with a testbench required ensuring both modes of operation worked, especially that no overflows occurred internally. Simulated antenna streams were offset to provide a clear difference between disabled and enabled operation.

Once successfully simulated, the averaging project was packaged and imported to OpenWiFi IP repositories. Little trouble was encountered with integrating the IP or verifying system operation. Results from the hardware manager are included to indicate successful operation of the IP.

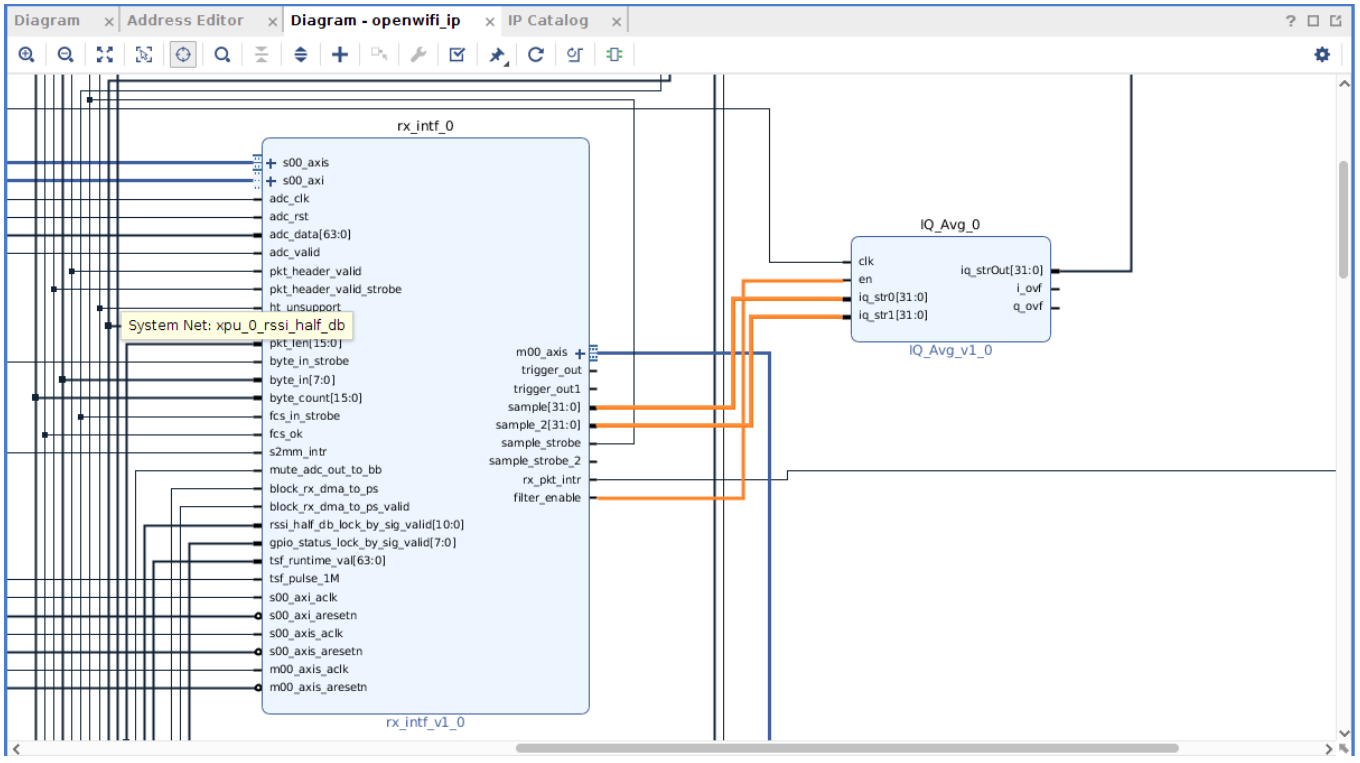

Figure 10: Integrated Averaging Filter with Modified rx_intf

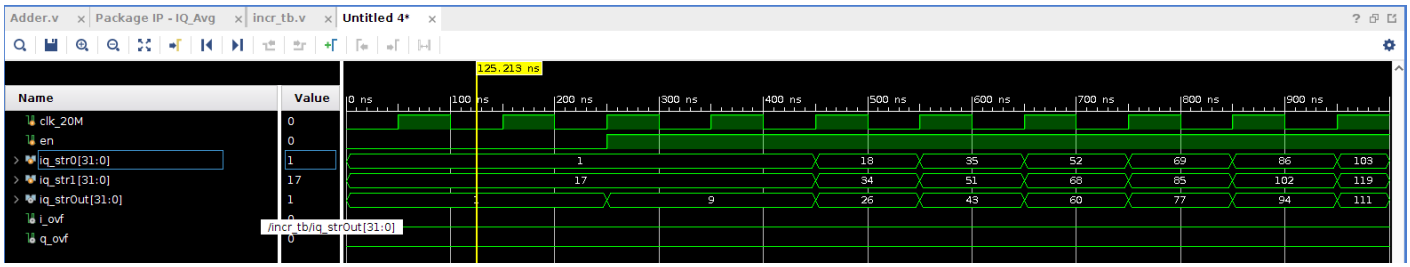

Figure 11: Averaging Filter Simulation Results

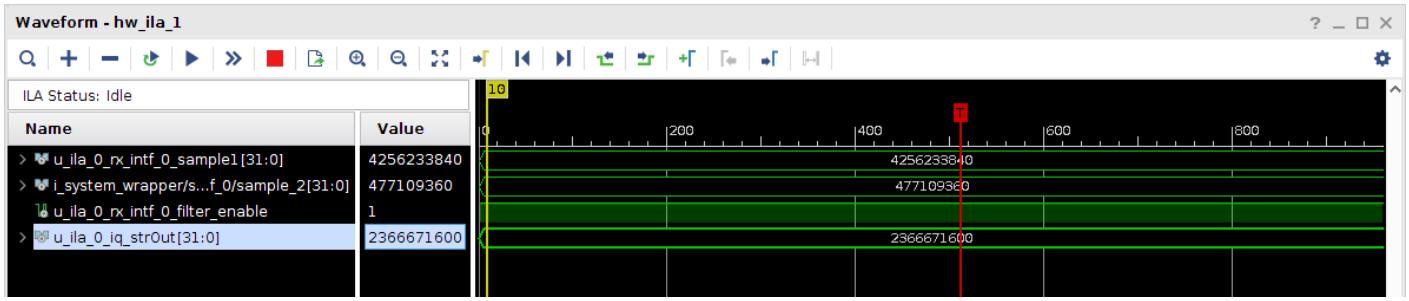

Figure 12: Averaging Filter Readback in Hardware Manager Session

C. *Future Work*

While implementation of the BJM algorithms was not accomplished in the timeframe, consideration was given to the requirements and design process anticipated.

Given that *OpenOFDM_rx* already handles the standardized offset correction, only the jamming-allievation filter **g** and blind-jamming spatial filter **P** will need to be implemented. One of the aforementioned tradeoffs in OpenWiFi is that CFO correction does not utlilize the LTF; it is possible, though unlikely, that this will need to incorporated with *OpenOFDM_rx* as well. Other sections of *OpenOFDM_rx* may have to be duplicated for the second antenna stream, such as those for frame detection and performing FFTs.

Because both new functions, **g** and **P**, rely on matrix algebra techniques – SVD and pseudoinverse, respectively – it is recommended that each algorithm is built as a separate IP using Vivado High-Level Synthesis. HLS provides functions for both techniques, allowing the algorithms to be built quickly. However, there may be challenges in optimizing the resource usage and delay associated with designing in HLS.

As each algorithm is designed and integrated, careful use of testbenches should be considered, particularly incorporating known inputs and expected outputs based on previous Matlab prototypes.

V. Conclusions

This paper presents the techniques and sample workflow to validate PHY layer designs in a real-time wireless system. Blind-Jamming Mitigation techniques are described as a goal for the implementation. The OpenWiFi project is presented as a viable method for implementing these PHY techniques in a close-to-real-world scenario. OpenWiFi's architecture is introduced in such that it can be implemented and modified at the hardware level, and sample projects are built to demonstrate the development process. Future work is prescribed, such that BJM techniques could be implemented with continued research.

Acknowledgment

The author would like to thank Dr. Huacheng Zeng and Hossein Pirayesh for their support and guidance throughout his studies of wireless systems.

VI. References

[1] Xilinx, "Bootgen User Guide," 28 September 2018. [Online]. Available: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2018_2/ug1283-bootgen-user-guide.pdf.

[2] Xilinx, "Vivado Design Suite User Guide: Programming and Debugging," 30 October 2019. [Online]. Available: https://www.xilinx.com/support/documentation/sw_manuals/xilinx2012_2/ug908-vivado-programming-debugging.pdf.

[3] M. Gast, 802.11 Wireless Networks: The Definitive Guide, Sebastopol: O'Reilly Media, 2013.

[4] J. Shi, "OpenOFDM: Synthesizable, Modular Verilog Implementation of 802.11 OFDM Decoder," 2017. [Online]. Available: https://openofdm.readthedocs.io/en/latest/.

[5] J. Xianjun, L. Wei and M. Michael, "open-source IEEE802.11/Wi-Fi baseband chip/FPGA design," GitHub, 2019. [Online]. Available: https://github.com/open-sdr/openwifi.

[6] H. Zeng, "Demo abstract: An anti-jamming wireless communication system," in *IEEE INFOCOM: Conference on Computer Communications Workshops*, Honolulu, 2018.

[7] H. Zeng, C. Cao, H. Li and Q. Yan, "Enabling jamming-resistant communications in wireless MIMO networks," in *IEEE Conference on Communications and Network Security*, Las Vegas, 2017.

[8] R. E. Ziemer and W. H. Tranter, "Angle Modulation and Multiplexing," in *Principles of Communications: Systems, Modulation, and Noise*, 7th ed., Wiley, 2015, pp. 156-196.