Reservoir Computing Accelerator on the bladeRF 2.0 Software-Defined Radio

Osaze Shears
Department of Electrical and Computer Engineering
Virginia Polytechic Institute and State University
Blacksburg, VA, USA
oshears@vt.edu

***Abstract*—The rise of 5G and beyond systems has fuelled research and innovation in intelligent wireless communications. Machine learning in particular has been one approach that contemporary research has been aiming to leverage for enabling cognitive radio capabilities. This paper reviews the delayed feedback reservoir, a low complexity recurrent neural network, and attempts to implement this model in the hardware of a modern software radio. This paper details the development process of the network and the approach taken to synthesize it in the software radio's existing FPGA image. The accuracy and power consumption of the hardware model is further analyzed to assess its viability to be implemented in an energy-constrained wireless device.**

I. Introduction

The advent of increasingly capable machine learning algorithms has inspired a wave of research regarding how to apply these techniques to a variety of domains. Image and video processing have experienced tremendous growth thanks to powerful convolutional neural network (CNN) algorithms that can accurately track specific objects observed in images. Speech recognition has benefited from long-short term memory (LSTM) networks that learn context from historical input data to effectively and translate languages. However, wireless communication has been a less popular domain for applying machine learning techniques. [1] recognizes this concern and provides a survey of the potential applications of machine learning techniques for wireless communication, including dynamic frequency allocation, cooperative spectrum sensing, and channel estimation.

[2] provides detailed discussion of the spectrum sharing use case, and how machine learning can help with this problem. In summary, developments in 5G and IoT devices has resulted in a rapid increase in the number of devices communicating over cellular networks. The number of connections that can be made over the cellular frequency bands is limited by the amount of spectrum available to users. Spectrum sharing is an approach that aims to multiplex spectrum allocations by allowing secondary users to transmit data over a pre-allocated channel when it is not being actively utilized by its primary users. The secondary users are able to analyze the channel activity using machine learning spectrum sensing algorithms that learn the access patterns of other users over time to avoid interfering with their transmissions, while also providing efficient data transmission. [2] overviews a joint recurrent neural network (RNN) and reinforcement learning approach which has been demonstrated by previous works to be well suited for practical spectrum sensing.

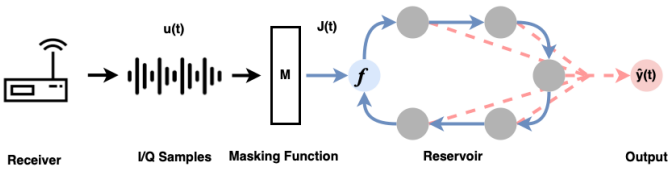

Fig. 1: A diagram of the flow of data in a spectrum sensing DFR. Here the I/Q data received from an RF transceiver is masked and processed through the DFR which predicts if the spectrum is currently being occupied.

[3] provides an alternative approach to performing machine learning-based spectrum sensing. The authors demonstrate a "reservoir computing" approach where an RNN consisting of randomized internal connections forms a reservoir state that can be used to predict the occupancy of a wireless channel. This approach was noted to require low training complexity, few hardware resources, and minimal energy consumption, enabling it to be realized in power-constrained wireless edge devices.

The RNN architecture used in [3] is known as a delayed-feedback reservoir (DFR). The DFR was initially conceptualized in [4] as a response to more complex reservoir computing architectures (e.g., echo state network). The algorithm works by masking sequential input data and integrating it with the present state of the reservoir. The resulting sum of the input and reservoir is processed through a non-linear transformation function (i.e., an activation function), and sent to a chain of N virtual nodes to be delayed by some predetermined period of time. Once all of the masked input data is processed through the reservoir, the current state of the reservoir is weighed and summed to make an output prediction. [3] shows that this approach can yields reasonable accuracy when tested in a spectrum sensing application with few antennas and low signal-to-noise ratios. Figure 1 illustrates how the DFR can be interfaced with an RF transceiver to perform spectrum sensing.

This paper aims to expand research on neural network enabled spectrum sensing by demonstrating a DFR implemented in a software-defined radio (SDR) to perform real-time spectrum sensing. The DFR algorithm was developed in software and evaluated against spectrum occupancy data collected by RWTH Aachen University [5]. The DFR was then implemented on the bladeRF 2.0 SDR within its embedded field-programmable gate array (FPGA) [6].

This project has important practical implications for custom FPGA logic design for emerging SDRs. Few resources exist that instruct new users how to develop and integrate their own logic into FPGA-enabled SDRs. This project demonstrates this process for the bladeRF, and can be generalized to other similar SDRs.

II. Delayed Feedback Reservoir

The delayed feedback reservoir (DFR) is a recurrent neural network (RNN) that features its nodes organized in a ring topology. As with all reservoir computing networks, training in this network occurs only at the output layer. This avoids problems that other RNNs are susceptible to, such as the vanishing and exploding gradient issues [7].

In the first stage of the DFR, the masking stage, each input sample $u(t)$ is multiplied by an $N \times 1$ matrix of scalar values M. Once the input is masked, now $J(t)$, each of the N sub-samples is sent to the DFR's non-linear node which is the first node of the reservoir, as shown in Figure 1. The DFR's non-linear node transforms the sum of the input $J(t)$ and the time-delayed reservoir output $x(t-\tau)$ based on an activation function $f(x)$. The output of the activation function is then sent to the chain of N virtual nodes that delays the feedback by τ. Equation 1 below models the behavior of the DFR's reservoir stage. Here, the input gain γ and the feedback scale η are used to adjust the influence of the new input and feedback on the reservoir state.

$$x(t) = f[\gamma \cdot J(t) + \eta \cdot x(t-\tau)], \tag{1}$$

During the readout stage, an output prediction $\hat{y}(t)$ is generated by evaluating the weighted sum of the values in each of the virtual reservoir nodes. Equation 2 models the behavior of the DFR's readout stage. In this equation, w_i is the weight of virtual node i, τ is the feedback delay, and N is the number of virtual nodes.

$$\hat{y}(t) = \sum_{i=1}^{N} w_i \cdot x(t - \frac{\tau}{N}(N-i)), \tag{2}$$

In order to train the DFR to accurately model a given time series, a standard regression can be employed. Equation 3 shows the ridge regression approach taken to generate a matrix of weights according to the previous reservoir states for each input.

$$\mathbf{w} = \frac{\mathbf{y} \cdot \mathbf{X}}{\mathbf{X} \cdot \mathbf{X^T} + \lambda \mathbf{I}}, \tag{3}$$

In this equation, $\mathbf{y}$ is an M length vector of expected outputs, $\mathbf{X}$ is a matrix of N reservoir node values for each of the M inputs, $\mathbf{I}$ is the identity matrix, and λ is the regularization coefficient.

III. Hardware DFR Implementation

To demonstrate the capabilities of the delayed feedback reservoir (DFR) for cognitive radio applications such as spectrum sensing, this project aimed to train a DFR network and synthesize it in hardware. The first step was constructing a DFR model in Python[1] and training it to accurately predict spectrum occupancy according to a fabricated data set. Spectrum occupancy measurements were obtained from RWTH Aachen University's spectrum occupancy data set. The portion of this data set that was referenced contains 6102 orthogonal frequency-division multiplexing (OFDM) frames across 40 subcarriers. Whether data was transmitted over a given frame is indicated by a 1 or a 0 in the data set. Based on the spectrum occupancy behavior for one of these 40 subcarriers, 6102 random quadrature phase-shift keying (QPSK) symbols were generated. Each symbol corresponding to a point in the sequence of OFDM frames where the spectrum was not busy was cleared from the frame. Once the array of symbols were generated, random additive white Gaussian noise (AWGN) was added to each symbol. The in-phase and quadrature (I/Q) components of each symbol were then translated into signed 16 bit binary numbers to represent the output of the SDR's analog-to-digital converters (ADC).

The parameters of the DFR implemented in Python are shown in Table I. At each frame the DFR was presented with the generated I/Q components. The energy of the frame was calculated using Equation 4:

$$E = \sqrt{I^2 + Q^2} \tag{4}$$

This energy measurement was then fed into the input layer of the DFR where it was masked and sent to the single non-linear node. Once the outputs of all training samples were calculated, ridge regression was used to adjust the DFR's output weights which allowed the system to predict the spectrum occupancy. The optimal configurations for the mask range, number of reservoir nodes, input gain, and feedback scale were determined after simulating several variations of the DFR and evaluating its performance for the spectrum sensing task.

TABLE I: Delayed Feedback Reservoir Parameters

Parameter	Value
Mask Range	[-0.5,0.5]
Reservoir Nodes	50
Input Gain γ	0.5
Feedback Scale η	0.4
Floating Point Exponent Precision	8 bits
Floating Point Mantissa Precision	17 bits
Input Data Resolution	16 bits

[1]Python, HDL, and C++ code for this project is available at https://github.com/oshears/bladerf_dfr_accelerator.

After verifying the DFR in Python, a C++ model the network was developed using the Intel HLS Compiler libraries [8]. The parameters of the DFR implemented in C++ are shown in Table I. The floating point precision was chosen to minimize the number of hardware resources required by the DFR hardware implementation. After implementing the DFR in C++, its accuracy was compared to the Python model to verify its functionality. Once the software model was verified, the HLS tool was used to create the Verilog hardware description language (HDL) code corresponding to the software model. The logic utilization of the DFR IP core for the Cyclone V 5CEBA4F23C8 FPGA, shown in Table II, was obtained after synthesizing the core in Quartus Prime.

TABLE II: Delayed Feedback Reservoir Logic Utilization

Logic Element	Utilization	Utilization Percentage
Adaptive Logic Modules (ALMs)	9,149	50%
Block Memory Bits	118,212	58%
Digital Signal Processors (DSPs)	48	73%

The maximum clock rate and latency measurements were obtained from the Intel HLS Compiler report generated after running the HLS tool. The power estimates were obtained using Intel Quartus' Power Analyzer Tool. These metrics are found in Table III below.

TABLE III: Delayed Feedback Reservoir Hardware Characteristics

Attribute	Value
Maximum Clock Rate	194.40 MHz
Latency	374 Cycles
Dynamic Thermal Power	2.205 W
Static Thermal Power	0.203 W
Total Power	2.412 W

The toggle rate used for the input I/O signals was set to 12.5%. The power estimates generated from this tool indicates that the power consumption of the DFR circuit falls within the same power consumption range of a cellphone that is actively transmitting and receiving data (on the order of 1 watts to 6 watts). The power consumption of this circuit can be further reduced if it is optimized to use fixed point arithmetic, as opposed to floating point arithmetic, and if the architecture is optimized to leverage parallelism in the algorithm.

The last step in the development of this DFR IP core was verifying the hardware's functionality in a simulation and in real-time. The simulation DFR model was verified using the test bench generated by the Intel HLS tool. Due to limited time, this project was not able to test the performance of the DFR in real-time; However, we discuss what has been accomplished to support this real-time simulation and our next steps for finishing this effort in the following sections.

IV. BLADERF SDR

A. Board Design

The bladeRF 2.0 micro, shown in Figure 2, is a mid-tier software defined radio developed by Nuand. The device is capable of transmitting and receiving RF data over two transmit and two receive antennas. The SDR attributes are summarized in Table IV. These attributes were acquired from Nuand's bladeRF webpage [6].

TABLE IV: bladeRF 2.0 micro

Attribute	Capability
Frequency Range	47 MHz - 6 GHz
Sampling Rate	61.44 MHz
RX/TX Antennas	2/2
ADC/DAC Resolution	12 Bits

At the heart of this device is an Intel Altera Cyclone V FPGA. The FPGA acts as a bridge between AD9361 RFIC which receives and transmits RF data, and the host computer which performs digital signal processing on the in-phase and quadrature (I/Q) samples. The FPGA enables further configurability of this system by allowing a designer to implement custom logic in the FPGA architecture. Thus, a designer can create custom hardware units to perform filtering, modulation and cognitive radio tasks directly in the hardware. This is advantageous to designers because hardware implementations of these algorithms are likely to run faster than software implementations, in addition to consuming less power.

The Cypress FX3 USB 3.0 peripheral controller manages transactions between the SDR and the host computer. The host sends packetized data requests to the SDR to configure the frequency, voltage controlled oscillator (VCO), and gain settings. These data packets are translated from the USB protocol to UART and then interpreted by the FPGA which performs the hardware updates. The controller also reads the stream of I/Q sample data over the general programmable interface (GPIF) bus, and sends these to the host computer via the USB interface.

The last major component is the Analog Devices AD9361 RF transceiver. The transceiver is interfaced with four SMA antenna interfaces and supports multiple-input, multiple-output antenna configurations. The device is configured by the FPGA using the SPI interface which connects directly to its internal

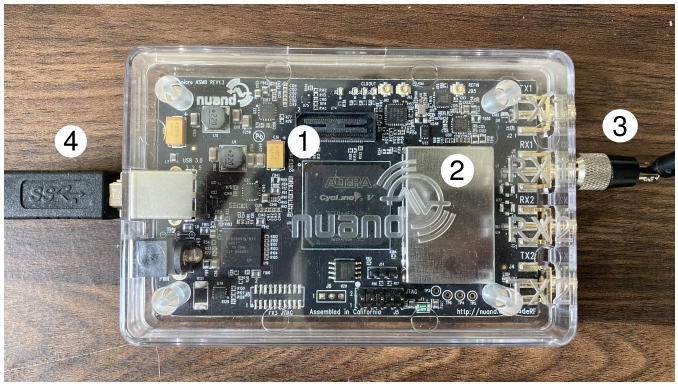

Fig. 2: A picture of the bladeRF 2.0 micro device. The device features (1) an embedded FPGA acting as the interface between the host computer and the AD9361 RFIC (2). The AD9361 is found under the RF shield and directly interfaces with the four SMA antenna interfaces (3). Data is transmitted between the bladeRF and the host via a USB 3.0 port (4)

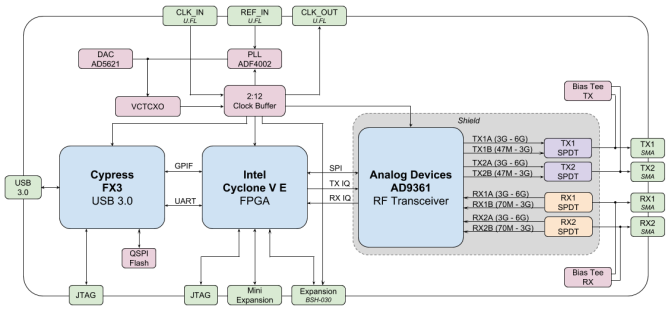

Fig. 3: A block diagram illustrating the connections between the components on the bladeRF 2.0 micro board. This figure was obtained from Nuand's webpage for the bladeRF [6].

registers. The TX IQ and RX IQ interfaces carry the IQ samples between the transceiver and the FPGA. Figure 3 shows how the integrated circuit components of the bladeRF system are connected on the board.

The challenge in developing custom hardware to run on the programmable logic of this device is the difficulty in learning and adapting custom hardware intellectual property (IP) to the bladeRF's FPGA architecture. Nuand has provided documentation to explain the software and hardware features of their device. However, designers must still spend time dissecting the extensive list of VHDL and Verilog source code that is used to program the FPGA in order to adequately integrate their own IP. The following sections will breakdown the bladeRF's FPGA architecture to provide readers with an understanding of the functional units in this SDR system.

B. FPGA Architecture

The bladeRF's bridge logic between the AD9361 RFIC and the host computer is implemented on the Intel Altera Cyclone V FPGA (5CEBA4F23C8). In this architecture, UART packets are received from the external FX3 USB controller and processed by a Nios II soft processor system. If the packets contain configuration data, the soft processor translates these packets into configuration commands that get sent to the external power monitor, local oscillator, or RFIC. The processor system is able to translate this data using several internal IP cores that create I2C and SPI transactions. Otherwise if the packets contain data requests, the processor instructs its internal AD9361 interface controller to read data from the RFIC and send it to the receive first-in first-out queue (RX FIFO). The RX FIFO acts as a buffer that synchronizes the rate at which the host is requesting RX samples to the rate that the RFIC is making them available. During its transition through the RX FIFO, the signed 12 bit in-phase and quadrature samples from the RFIC ADCs are sign-extended to 16 bits and concatenated to make 32 bit samples. These samples are sent to the host computer using the FX3 GPIF bridge module. Table V shows the logic utilization of the unmodified FPGA architecture.

Figure 5 illustrates the FPGA architecture observed in the top level bladeRF HDL file, along with the proposed

TABLE V: bladeRF 2.0 micro A4 FPGA Logic Utilization

Logic Element	Utilization	Utilization Percentage
Adaptive Logic Modules (ALMs)	6,280	34%
Package Pins	173	77%
Block Memory Bits	1,824,256	58%
Digital Signal Processors (DSPs)	8	12%
Phase Locked Loops (PLLs)	3	75%

modifications to embed the DFR core and test memory. The bladeRF Python package is used to read the received samples from the bladeRF device. In addition to reading and writing data to the device, the package can also be used to program the FPGA with a custom bit file (.rbf).

V. Results

A. NARMA10

To validate the practicality of the DFR as a predictive model, this project first chose to measure its performance on the nonlinear autoregressive moving average time series (NARMA10), a standard recurrent neural network benchmark. For this task, each input sample $u(k)$ is generated in the range 0 to 0.5, and each output $y(k)$ is generated using Equation 5 below.

$$y(k+1) = 0.3y(k-1) + 0.05y(k)[\sum_{i=0}^{9} y(k-i)] + 1.5u(k-9)u(k) + 0.1, \quad (5)$$

The highest performing algorithm capable of modeling this time series achieved a normalized root mean squared error (NRMSE) of 0.1 [11]. Our model sacrifices a slight reduction in accuracy by achieving a a NRMSE of 0.2, in exchange for using a significantly simpler, hardware-friendly network architecture. The predictive performance of the DFR in modeling this time series is visualized in Figure 4.

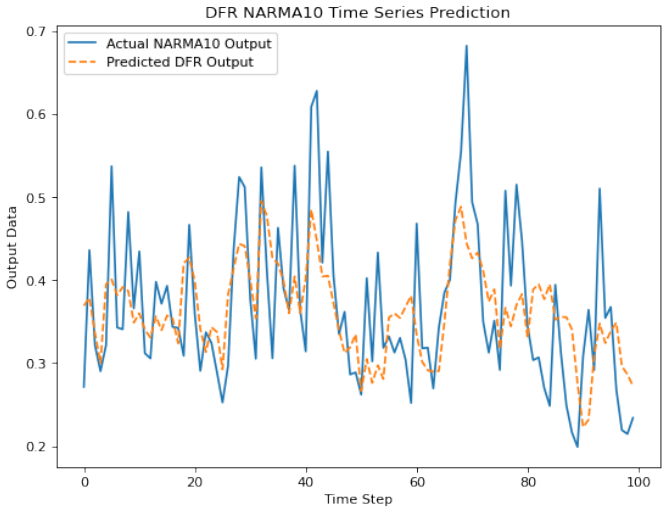

Fig. 4: A graph showing the DFR's predictive performance on the NARMA10 Time Series.

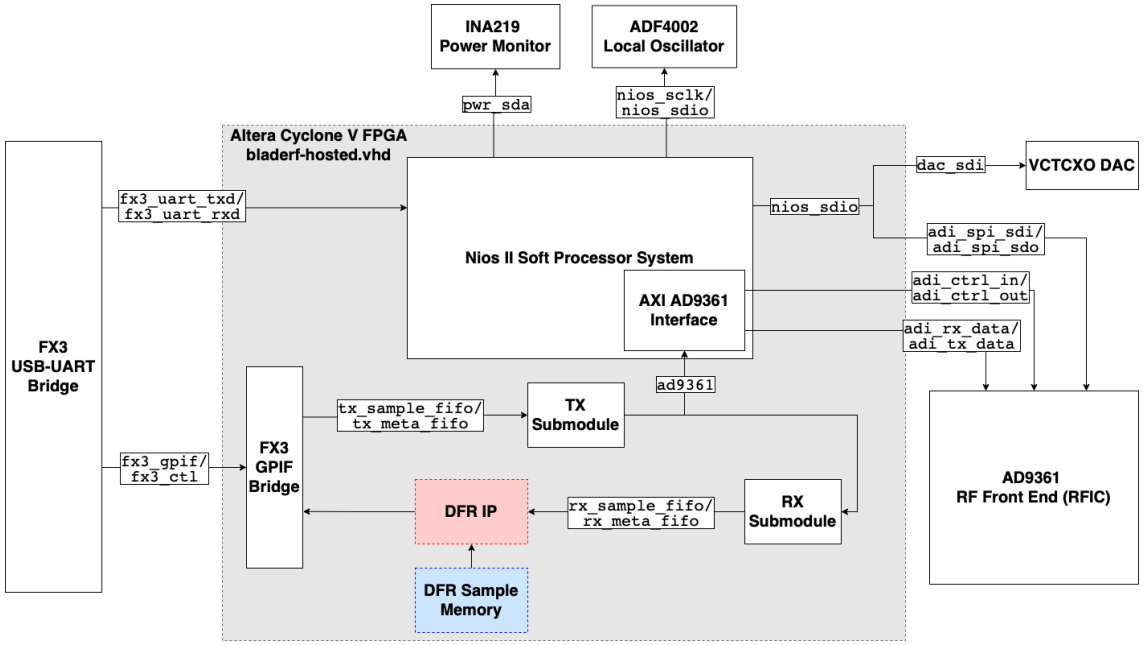

Fig. 5: A diagram of the bladeRF's FPGA architecture. The red box indicates the developed DFR HLS IP core that will perform spectrum sensing. The blue box indicates a test read only memory (ROM) that was included to verify the real-time functionality of the DFR core. This diagram is a modified version of that which appears on the bladeRF's GitHub page in [9].

B. Spectrum Sensing

The performance of the DFR in modeling the spectrum occupancy data was evaluated across three different signal-to-noise (SNR) ratios. For each data set, the DFR was modeled and trained according to the equations in Section II and the parameters in Table I. Since spectrum sensing is a binary classification task requiring the model to determine if the spectrum is free or occupied, this project used receiver operating characteristic (ROC) curves to evaluate the model's accuracy. ROC curves show the relationship between the false positive rate and true positive rate for varying thresholds that are used in the model's decision making process. The false positive and true positive rates respectively are calculated according to the Equations 6 and 7 below.

$$FPR = \frac{FP}{TN + FP} \tag{6}$$

$$TPR = \frac{TP}{TP + FN} \tag{7}$$

In these equations, for a given threshold, FP represents the number of false positives, FN represents the number of false negatives, TP represents the number of true positives, and TN represents the number of true negatives. When the area under the ROC curve (AUC) of a model is 1, it indicates that the model is able to perfectly predict the data set. Table VI shows the AUC values for the DFR models trained to perform spectrum sensing at the three tested SNR levels.

To evaluate the hardware version of the spectrum sensing DFR core, the bladeRF's top level HDL file was modified to include the DFR IP core, in addition to a sample memory to test the core's real-time performance on the initial data set. Instructions for modifying the FPGA image were provided on the bladeRF's GitHub repository Wiki page [9]. Due to time contraints, this project was unable to test the hardware DFR against the synthetic spectrum sensing data; However, the bulk of the effort required to integrate the necessary IP and HDL modifications has been completed to allow future efforts to perform this. All generated HDL and modifications to the bladeRF's FPGA image can be found in the GitHub repository for this project[1].

TABLE VI: Spectrum Sensing DFR Performance

SNR	Area Under the Curve (AUC)
0dB	0.733
-5dB	0.712
-10dB	0.666

VI. Conclusion

This project demonstrated that the delayed feedback reservoir is a low area, and low power recurrent neural network that has the potential to accurately predict time series tasks. Specifically the paper highlights the DFR's ability to model the NARMA10 time series with a NRMSE of 0.2, and a synthetic spectrum sensing data set with an SNR of -10dB with 67% accuracy. The paper identifies the bladeRF SDR as a platform for synthesizing and testing the DFR against spectrum sensing data in real-time. The paper overviews the bladeRF's architecture and indicates how the DFR can be

[1]Python, HDL, and C++ code for this project is available at https://github.com/oshears/bladerf_dfr_accelerator.

effectively integrated and tested in the device's FPGA. A repository of code supporting this project was published to support future development of hardware accelerator IP cores on the bladeRF platform.

In the future, this project will continue its implementation of the hardware DFR by first testing the core against the fabricated spectrum sensing data stored in a hardware read-only memory (ROM). Once the core is proven to produce the same results as the software model, it will be interfaced with the receive sample chain of the SDR. This would allow the core to make spectrum occupancy predictions using practical I/Q data. An effort of this nature would be one of the first to demonstrate a neural network performing spectrum sensing in low-cost, open-source hardware.

REFERENCES

[1] M. E. Morocho-Cayamcela, H. Lee, and W. Lim, "Machine learning for 5G/B5G mobile and wireless communications: potential, limitations, and future directions," *IEEE Access*, vol. 7, no. 1, pp. 137184-137206, Sept. 2019.

[2] K. Cohen, "Machine Learning for Spectrum Access and Sharing," in *Machine learning for future wireless communications.*, F. L. Luo ed. Hoboken, NJ, USA: John Wiley and Sons, 2019, ch. 1, pp. 3-20.

[3] K. Hamedani, L. Liu, S. Liu, H. He, and Y. Yi, "Deep spiking delayed feedback reservoirs and its application in spectrum sensing of mimo-ofdm dynamic spectrum sharing," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 02, Apr. 2020, pp. 1292-1299.

[4] L. Appeltant et al., "Information processing using a single dynamical node as complex system," *Nature communications*, vol. 2, no. 1, pp. 1-6, Sept. 2011.

[5] RWTH Aachen University, Department of Wireless Networks. "RWTH Aachen University Static Spectrum Occupancy Measurement Campaign." RWTH Aachen University. https://download.mobnets.rwth-aachen.de/. (accessed Oct. 17, 2021).

[6] Nuand. "bladeRF 2.0 micro xA4." Nuand.com. https://www.nuand.com/product/bladerf-xa4/. (accessed Oct. 17, 2021).

[7] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," in *Int. Conf. on Mach. Learn.*, 2013, pp. 1310-1318.

[8] Intel Coorporation, "Intel® High Level Synthesis Compiler." https://www.intel.com/content/www/us/en/software/programmable/quartus-prime/hls-compiler.html (accessed Oct. 17, 2021).

[9] Nuand. "bladeRF Source" GitHub.com. https://github.com/Nuand/bladeRF. (accessed Oct. 17, 2021).

[10] D. Bogdan and L. Sin. "AD9361 HDL Reference Designs." Analog Devices. https://wiki.analog.com/resources/eval/user-guides/ad-fmcomms2-ebz/reference_hdl. (accessed Oct. 17, 2021).

[11] H. Jaeger. "Adaptive nonlinear system identification with echo state networks," *Advances in neural information processing systems* vol. 15, no. 1, pp. 609-616, Jan. 2002.