# Design and Implementation of a Neural Network Aided Self-Interference Cancellation Scheme for Full-Duplex Radios

Yann Kurzo, Andreas Burg, Alexios Balatsoukas-Stimming
Telecommunications Circuits Laboratory
École polytechnique fédérale de Lausanne, CH-1015 Lausanne, Switzerland

***Abstract*—In-band full-duplex systems are able to transmit and receive information simultaneously on the same frequency band. Due to the strong self-interference caused by the transmitter to its own receiver, the use of non-linear digital self-interference cancellation is essential. In this work, we present a hardware architecture for a neural network based non-linear self-interference canceller and we compare it with our own hardware implementation of a conventional polynomial based canceller. We show that, for the same cancellation performance, the neural network canceller has a significantly higher throughput and requires fewer hardware resources.**

## I. Introduction

In-band full-duplex (FD) communication has for long been considered to be impractical due to the strong self-interference (SI) caused by the transmitter to its own receiver. However, more recent work on the topic (e.g., [1], [2], [3]) has demonstrated that it is in fact possible to achieve sufficient SI cancellation to make FD systems viable. A portion of the SI is usually first removed in the analog RF domain. As analog cancellation alone is often not sufficient, the residual SI needs to be cancelled in the digital domain. In principle, the residual SI should be easy to cancel since it is produced by a known transmitted signal. In practice, however, the different stages of the transceiver introduce non-linearities to the signal, such as digital-to-analog converter (DAC) and analog-to-digital converter (ADC) non-linearities, IQ imbalance, and power amplifier (PA) non-linearities. Intricate memory polynomial models have to be used in order for the digital SI cancellation to be able to handle the aforementioned non-linearities (e.g., [4], [5], [6], [7], [8]). An alternative solution, which uses a neural network (NN) to reconstruct the non-linearities in order to generate the SI cancellation signal, was recently proposed in [9] and it was shown that it can achieve similar SI cancellation performance with the state-of-the-art polynomial model of [8], but with much lower computational complexity.

Existing NN hardware accelerators, such as [10], [11], mainly target applications where both the size of the NN and the number of inputs is very large, and where producing a a few tens of outputs per second is sufficient. Communications applications, on the other hand, use relatively small NNs with few inputs, but need to provide millions of outputs per second. As such, communications applications require vastly different NN hardware accelerator architectures.

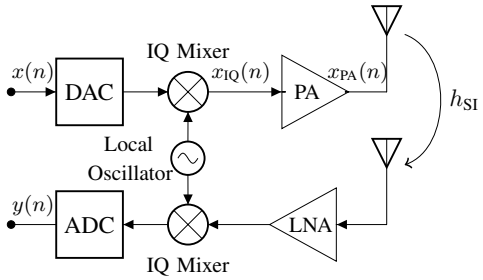


Fig. 1. Simplified wireless transceiver block diagram.

*Contribution:* In this work, we present a hardware implementation of the SI cancellation method proposed in [9] in order to quantify and translate the computational complexity gains over the state-of-the-art polynomial based model of [8] into real-world hardware resource utilization gains. We provide FPGA and ASIC implementation results that clearly demonstrate the significant gains that can be achieved by our proposed NN-based canceller in terms of both the resource utilization and the achieved throughput. To the best of our knowledge, this is the first hardware implementation of a NN-augmented communications system in literature related to the recent resurgence of machine learning for communications.

## II. Digital Self-Interference Cancellation

A basic block diagram of a full-duplex wireless transceiver is shown in Fig. 1. If we assume, for simplicity, that there is no signal-of-interest from a remote node and no thermal noise, then the received signal $y(n)$ is the SI signal. The goal of digital SI cancellation is to reproduce an accurate copy of $y(n)$, denoted by $\hat{y}(n)$, based on the transmitted baseband signal $x(n)$. This signal is then subtracted from $y(n)$, so that the residual SI signal is $y_c(n) = y(n) - \hat{y}(n)$. If $\hat{y}(n)$ is reconstructed perfectly, then the SI can be cancelled entirely and $y_c(n) = 0$. In practice, however, due to the presence of thermal noise and transceiver non-linearities, perfect SI cancellation is difficult to achieve.

*1) Polynomial Non-Linear Cancellation:* A state-of-the-art polynomial SI cancellation model, which can effectively suppress IQ imbalance and PA non-linearities, was described

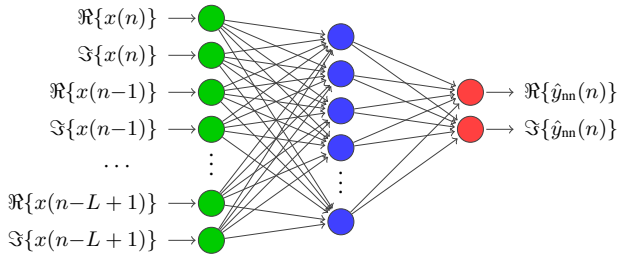


Fig. 2. Example of a neural network for the reconstruction of the non-linear component of the SI signal [9].

in [8]. Specifically, it was shown that an accurate SI cancellation signal $\hat{y}(n)$ can be obtained as:

$$\hat{y}(n) = \sum_{\substack{p=1,\\ p \text{ odd}}}^{P} \sum_{q=0}^{p} \sum_{l=0}^{L-1} \hat{h}_{p,q}(l) \underbrace{x(n-l)^q x^*(n-l)^{p-q}}_{\text{basis functions}}, \quad (1)$$

where $x(n)$ is the transmitted digital baseband signal, $L$ corresponds to the overall memory of the system, $P$ is the non-linearity order, and $\hat{h}_{p,q}$ are estimated parameters that can be obtained using, e.g., least-squares estimation.

*2) Neural Network Non-Linear Cancellation:* The NN-based method of [9] uses two steps, as illustrated in Fig. 3. First, standard linear cancellation is used in order to reconstruct the linear component of the SI, denoted by $\hat{y}_{\text{lin}}(n)$:

$$\hat{y}_{\text{lin}}(n) = \sum_{l=0}^{L-1} \hat{h}(l) x(n-l), \quad (2)$$

where $\hat{h}$ are estimated parameters that can be obtained using, e.g., least-squares estimation. A two-layer real-valued neural network, shown in Fig. 2, generates the non-linear part of the SI cancellation signal, denoted by $\hat{y}_{\text{nn}}(n)$. Finally, the two components are added in order to create the SI cancellation signal $\hat{y}(n) = \hat{y}_{\text{lin}}(n) + \hat{y}_{\text{nn}}(n)$. The denormalization step in Fig. 3 is necessary because the NN learns to reproduce a normalized (i.e., zero-mean and unit-variance) version of $\hat{y}_{\text{nn}}$, as this generally improves the convergence of NN training.

*3) Computational Complexity:* Assuming that each complex multiplication can be implemented using three real multiplications and five real additions and that each complex addition can be implemented using two real additions, the total number of real multiplications and additions that are required by the polynomial canceller is [9][1]:

$$N_{\text{ADD,poly}} = \frac{7}{4} L\,(P+1)\,(P+3) - 2, \quad (3)$$

$$N_{\text{MUL,poly}} = \frac{3}{4} L\,(P+1)\,(P+3)\,. \quad (4)$$

[1]We note that the expression for $N_{\text{ADD,poly}}$ in our previous work of [9] erroneously ignored the five real additions that are required to implement each complex multiplication. As such, the actual complexity of the polynomial canceller is even higher than that reported in [9].

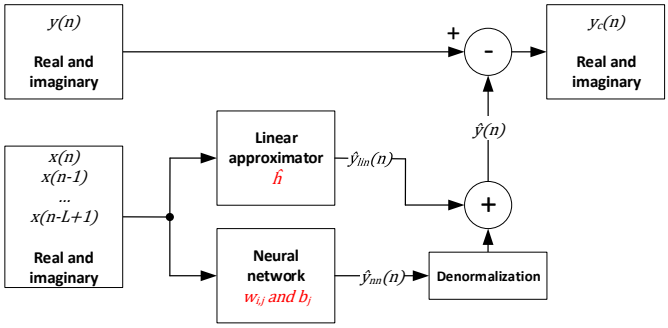


Fig. 3. Neural network based SI cancellation.

The number of real multiplications and additions that are required by the NN canceller is [9]:

$$N_{\text{ADD,NN}} = (2L+3)N_h + (7L-2), \quad (5)$$

$$N_{\text{MUL,NN}} = (2L+2)N_h + 3L, \quad (6)$$

where the second term in both expressions comes from the linear canceller. The complexity expressions for the two methods can not be compared directly because they contain different sets of parameters. In order to perform a fair comparison we select values for $L$, $P$, and $N_h$ so that the two methods have the same SI cancellation performance in Section IV.

## III. Hardware Architecture

In this section, we describe a hardware architecture that implements the NN-based SI canceller of [9]. We first give a global overview of the architecture, which is followed by a more detailed explanation of each component. As shown in Fig. 4, we map each layer of the NN to a macro-pipeline stage that requires several clock cycles to compute its outputs. Each macro-pipeline stage can start its computations as soon as valid outputs from the previous pipeline stage become available.

### A. Macro-Pipeline Architecture

Let $N_I$ and $N_n$ denote the number of inputs per neuron (which is equal to the number of neurons of the previous layer) and the number of neurons for a given NN layer, respectively. The goal of a macro-pipeline stage is to process each neuron of its corresponding layer by computing the following outputs:

$$o_j = f\left(b_j + \sum_{i=0}^{N_I-1} w_{i,j} x_i\right), \quad j \in \{0, \ldots, N_n - 1\}, \quad (7)$$

where $x_i$ are the inputs, $w_{i,j}$ are the *weights*, $b_j$ are the *biases*, and $f(x)$ is a non-linear activation function ([9] uses a ReLU activation function). The architecture of each macro-pipeline stage is shown in more detail in Fig. 5. More specifically, each macro-pipeline stage contains an input interface, an array of $N_{\text{PE}}$ processing elements (PEs), a weights-and-biases memory, a control unit, and an output interface. We note that all weights, biases, and partial sums have a common bit-width of $Q$ bits and saturation is used in case of an overflow.

The $N_{\text{PE}}$ PEs, whose internal structure is shown in Fig. 6, can be used to compute (7) over multiple clock cycles using

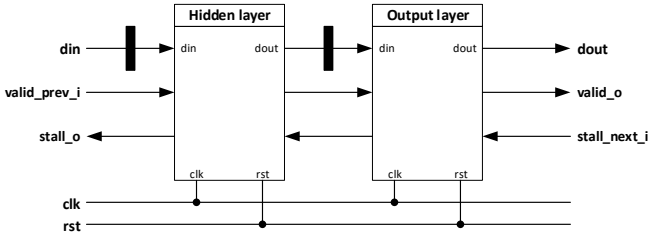


Fig. 4. Macro-pipeline architecture of the two-layer neural network.

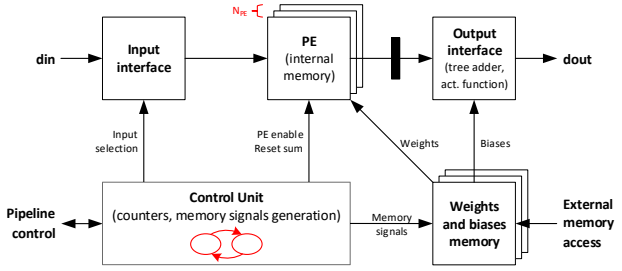


Fig. 5. Block diagram of the macro-pipeline stage architecture.

one of two possible schedules. In the neuron-by-neuron (NBN) schedule, neurons are processed sequentially and each of the $N_{\text{PE}}$ PEs computes a part of the sum in (7) for a given neuron $j$. In the input-by-input (IBI) schedule, on the other hand, the layer inputs $x_i$ are processed sequentially and the $N_{\text{PE}}$ PEs update the sum in (7) with the term $w_{i,j}x_i$ for $N_{\text{PE}}$ neurons in parallel. When an NBN macro-pipeline stage is followed by an IBI macro-pipeline stage, the IBI stage can already start performing computations once the output of the first neuron of the NBN stage has been computed, thus masking a significant part of the latency and reducing the number of interconnects between the two stages. Since the exact architecture of each macro-pipeline stage depends on the processing schedule, we describe the details of the corresponding architectures separately in the next two sections.

### B. Neuron-by-Neuron Macro-Pipeline Architecture

*1) Input Interface:* The input interface consists of $N_{\text{PE}}$ multiplexers, which route each input to the correct PE.

*2) Processing Elements:* In the NBN schedule, each PE is only associated with a single neuron, meaning that only a single partial sum needs to be stored. Thus, the PEs are simple multiply-and-accumulate (MAC) units and the memory shown in Fig. 6 is in fact a single $Q$-bit register.

*3) Control Unit:* The main tasks of the control unit are to distribute the computations to the PEs and to stall the computations when no valid inputs are available or when the following macro-pipeline stage is not ready to accept new outputs. The computations are dispatched to the PEs as follows. When $N_{\text{PE}} \leq N_I$, all $N_{\text{PE}}$ PEs are used to process a single neuron at a time and $N_n \left\lceil \frac{N_I}{N_{\text{PE}}} \right\rceil$ clock cycles are required to process all neurons. When $N_{\text{PE}} > N_I$, we constrain $N_{\text{PE}}$ so that $N_{\text{PE}} = kN_I$, $k \in \mathbb{N}$, meaning that $k$ neurons are processed in parallel and $\left\lceil \frac{N_n N_I}{N_{\text{PE}}} \right\rceil$ clock cycles are required to process all neurons.

*4) Weights and Biases Memories:* The weights and biases memories are used to store $w_{i,j}$ and $b_j$ and they can be written externally to re-configure the NN. The weights are organized in a memory that is $N_{\text{PE}}Q$ bits wide so that all PEs can be provided with data in parallel. A single word of the weights memory contains $N_{\text{PE}}$ weight values corresponding to $k$ different neurons. The biases memory, on the other hand, has a bit-width of $kQ$ bits.

*5) Output Interface:* The output interface adds the partial sums from the $N_{\text{PE}}$ PEs using an adder tree, it adds the biases, and it applies the non-linear activation function for each of the $k$ neurons that are being processed in parallel. A register is added between the PEs and the output interface in order to reduce the critical path of the architecture. Moreover, the output interface forwards the outputs of the $k$ neurons that are processed in parallel to the next macro-pipeline stage.

*6) Latency:* If $N_{\text{PE}}$ is chosen carefully so that $\frac{N_I}{N_{\text{PE}}}$ and $\frac{N_n N_I}{N_{\text{PE}}}$ are always integers, then it takes

$$\mathcal{L} = \frac{N_n N_I}{N_{\text{PE}}} + 1 \tag{8}$$

clock cycles to produce all outputs of a NN layer. However, one full set of outputs for a NN layer is actually produced every $\frac{N_n N_I}{N_{\text{PE}}}$ cycles, so that the throughput of the NBN macro-pipeline stage is

$$\mathcal{T} = \frac{N_{\text{PE}}}{N_n N_I} \tag{9}$$

Moreover, the first $k$ outputs of an NBN macro-pipeline stage become available after

$$\mathcal{L}_f = \frac{N_I}{N_{\text{PE}}} + 1 \tag{10}$$

clock cycles and after that $k$ new outputs are produced in every clock cycle. This means that a potential IBI macro-pipeline stage that follows can already start its computations after $\mathcal{L}_f$ clock cycles and that only $k \leq N_n$ outputs need to be forwarded to the next stage in each clock cycle.

### C. Input-by-Input Macro-Pipeline Architecture

*1) Input & Output Interface:* The input and output interfaces are similar to that of the NBN macro-pipeline stage, the main difference being that the IBI output interface forwards the outputs of all $N_n$ neurons that are processed in parallel to the next macro-pipeline stage.

*2) Processing Elements:* In the IBI schedule, each PE can be associated with multiple neurons, meaning that several partial sums potentially need to be stored. Thus, the PEs are MAC units and the memory shown in Fig. 6 has a dimension of $\left\lceil \frac{N_n}{N_{\text{PE}}} \right\rceil \times Q$ bits.

*3) Control Unit:* In the IBI schedule, when $N_{\text{PE}} \leq N_n$, all $N_{\text{PE}}$ PEs are used to update the $N_n$ neurons sequentially with the new input value $x_i$ and $N_I \left\lceil \frac{N_n}{N_{\text{PE}}} \right\rceil$ clock cycles are required to process all neurons. When $N_{\text{PE}} > N_n$, we constrain $N_{\text{PE}}$ so that $N_{\text{PE}} = kN_n$, $k \in \mathbb{N}$, meaning that $k$ inputs are processed in parallel and $\left\lceil \frac{N_n N_I}{N_{\text{PE}}} \right\rceil$ clock cycles are required to process all neurons.

*4) Weights and Biases Memories:* The weights and biases memories are similar to those of the NBN macro-pipeline stage. A single word of the weights memory contains $N_{\text{PE}}$ weight values corresponding to $k$ different neurons. The biases memory, on the other hand, has a bit-width of $N_n Q$ bits.

*5) Latency:* Similarly to the NBN schedule, if $N_{\text{PE}}$ is chosen carefully so that $\left\lceil \frac{N_n}{N_{\text{PE}}} \right\rceil$ and $\left\lceil \frac{N_n N_I}{N_{\text{PE}}} \right\rceil$ are always integers, then the latency and the throughput are

$$\mathcal{L} = \frac{N_n N_I}{N_{\text{PE}}} + 1, \quad \text{and} \quad \mathcal{T} = \frac{N_{\text{PE}}}{N_n N_I}, \tag{11}$$

respectively. Moreover, all $N_n$ outputs of an IBI macro-pipeline stage become available simultaneously after

$$\mathcal{L}_f = \frac{N_n N_I}{N_{\text{PE}}} + 1 \quad \text{clock cycles.} \tag{12}$$

### D. Overall Neural Network Canceller Architecture

The overall architecture for the two-layer NN of [9] consists of two macro-pipeline stages, one for the hidden layer and one for the output layer, and pipeline registers are added between the macro-pipeline stages. The hidden layer uses an NBN macro-pipeline stage, while the output layer uses an IBI macro-pipeline stage. For the hidden layer, we have $N_I = 2L$ and $N_n = N_h$, while for the output layer we have $N_I = N_h$ and $N_n = 2$. The $N_I = 2L$ inputs of the first macro-pipeline stage that implements the computations of the hidden layer are assumed to all be available in parallel. The number of PEs instantiated for the hidden layer and the output layer is $N_{\text{PE},h}$ and $N_{\text{PE},o}$, respectively. The computations for the linear canceller are done in parallel with the NN by instantiating a standard complex FIR filter. If we denote the throughput of the hidden and the output macro-pipeline stages by $\mathcal{T}_h$ and $\mathcal{T}_o$, respectively, then the throughput of the two-layer NN architecture is

$$\mathcal{T} = \min\left(\mathcal{T}_h, \mathcal{T}_o\right). \tag{13}$$

Finally, we note that we constrain the denormalization step shown in Fig. 3 to scaling with powers of two, which can be implemented efficiently with simple shifting operations, both during training and during inference.

## IV. FPGA and ASIC Implementation Results

In this section, we present implementation results for the NN-based canceller and we compare it with a polynomial canceller. Since, to the best of our knowledge, there are no published implementations of polynomial cancellers in the literature, we provide our own reference implementation. Due to space limitations, we do not describe the implementation in detail, but it is largely based on the NN architecture since the main computational task of the polynomial canceller is similar to the NN canceller, i.e., to compute a weighted sum. The main differences are that the input interface also computes the basis functions and that the PEs operate directly on complex numbers. Each complex PE of the polynomial canceller is implemented using three real multipliers.

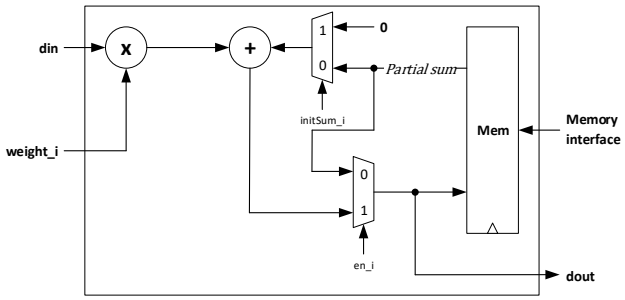


Fig. 6. Detailed view of the PE architecture that is used by both the NBN and the IBI macro-pipeline stages.

TABLE I
Comparison of NN-based and polynomial cancellers.

| | Polynomial | Neural Network |
|---|---|---|
| Cancellation (dB) | $-44.8$ | $-44.4$ |
| Real Parameters | 520 | 550 |
| Real Multiplications | 780 | 543 |
| Real Additions | 1818 | 611 |

### A. Comparison Setup

In order to provide a fair comparison between the NN-based SI canceller and the polynomial canceller, we select $L$, $N_h$, $P$, and the quantization bit-width $Q$ so that the fixed-point performance of the two cancellers is as similar as possible. For performance evaluation, we used the same dataset that was used in [9], which consists of a 10 MHz QPSK-modulated OFDM signal sampled at 20 MHz that is generated using the testbed described in [12] and [13].

For $L = 13$, $P = 7$, and $N_h = 18$, the performance of the two cancellers is very similar, as can be seen in Table I. In Fig. 7, we show the cancellation performance for the NN-based canceller and the polynomial based canceller as a function of $Q$. We observe that, for the same cancellation performance, the NN-based canceller generally requires a lower quantization bit-width $Q$. For the hardware implementation results, we choose $Q = 17$ for the NN-based canceller and $Q = 23$ for the polynomial canceller so that the two cancellers have the same fixed-point cancellation performance.

We set $N_{\text{PE},h} = 52$ and $N_{\text{PE},o} = 4$ for the NN-based canceller so that $\mathcal{T}_h = \mathcal{T}_o = 1/9$, meaning that the macro-pipeline is perfectly balanced and one cancellation sample is output every 9 clock cycles. Furthermore, 2 complex PEs are instantiated for the NN-based canceller in order to perform the linear cancellation step in the same time. For the polynomial canceller, we use $N_{\text{PE},h} = 20$ complex PEs so that the 260 complex multiplications required to compute (1) for $L = 13$ and $P = 7$ can be carried out in 13 clock cycles, which means that one cancellation sample is output every 13 clock cycles.

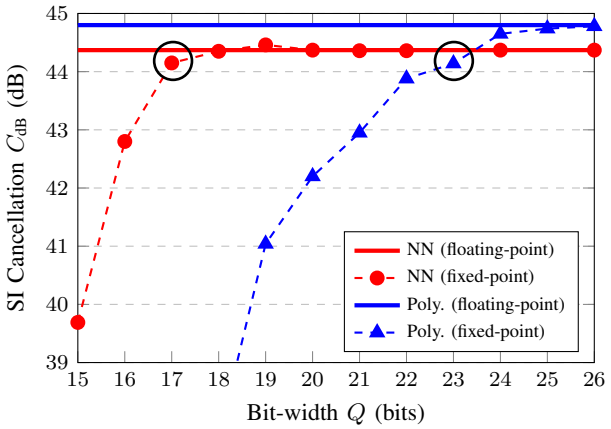


Fig. 7. SI cancellation as a function of the datapath bit-width $Q$.

TABLE II
FPGA IMPLEMENTATION RESULTS (VIRTEX-7 XC7VX485TFFG1161).

| | Polynomial | Neural Network |
|---|---|---|
| LUT (logic) | 6710/303.6k (2.21%) | 2831/303.6k (0.93%) |
| LUT (RAM) | 1638/130.8k (1.25%) | 1678/130.8k (1.28%) |
| Registers | 3922/607.2k (0.65%) | 2625/607.2k (0.43%) |
| DSP Slices | 132/2.8k (4.71%) | 62/2.8k (2.21%) |
| Frequency (MHz) | 67.2 | 92.0 |
| Throughput (MS/s) | 5.2 | 10.2 |

### B. Implementation Results

The placed-and-routed implementation results on a Xilinx Virtex-7 FPGA are given in Table II. We observe that the NN-based canceller has significantly lower resource utilization than the polynomial canceller and a 96% higher throughput. The higher throughput of the NN-based canceller comes both from a lower number of cycles per sample and from a higher operating frequency compared to the polynomial canceller. We also note that the polynomial canceller requires approximately two times more DSP slices than the NN-based canceller. This happens because the DSP slices on Xilinx Virtex-7 FPGAs do not support multiplications between two $Q = 23$-bit values and two DSP slices have to be instantiated for each multiplication in the polynomial canceller.

The fully placed-and-routed ASIC implementation results using a 28 nm FD-SOI technology are shown in Table III. We observe that the NN-based canceller has a 60% better throughput and that it occupies 11% less area than the polynomial canceller, leading to an 81% better hardware efficiency. Similarly to the FPGA results, the better throughput of the NN-based canceller comes both from a lower number of cycles per sample and from a higher operating frequency compared to the polynomial canceller.

## V. CONCLUSION

In the paper, we described a high-throughput hardware architecture for a NN-based self-interference cancellation scheme for full-duplex radios. Our implementation results show that the NN-based canceller has a lower computational complexity and that a 22% lower datapath quantization bit-width to achieve the same cancellation performance as a polynomial cancellation scheme. The NN-based canceller thus requires significantly fewer resources on an FPGA and achieves an 81% better hardware efficiency than the polynomial canceller when implemented for an ASIC target.

TABLE III
ASIC IMPLEMENTATION RESULTS (28 NM FD-SOI).

| | Polynomial | Neural Network |
|---|---|---|
| Area ($\text{mm}^2$) | 0.36 | 0.32 |
| Frequency (MHz) | 226 | 250 |
| Throughput (MS/s) | 17.4 | 27.8 |
| Efficiency (MS/s/$\text{mm}^2$) | 48 | 87 |

PAR results using slow corners, 0.7 V voltage, 125° C temperature.

## VI. ACKNOWLEDGMENT

The authors gratefully acknowledge the support of NVIDIA Corporation with the donation of the Titan Xp GPU used for this research. This work has been supported by the Swiss National Science Foundation under grant #182621.

## REFERENCES

[1] M. Jain, J. I. Choi, T. Kim, D. Bharadia, S. Seth, K. Srinivasan, P. Levis, S. Katti, and P. Sinha, "Practical, real-time, full duplex wireless," in *Int. Conf. on Mobile Computing and Networking. ACM*, 2011, pp. 301–312.

[2] M. Duarte, C. Dick, and A. Sabharwal, "Experiment-driven characterization of full-duplex wireless systems," in *IEEE Trans. Wireless Commun., vol. 11, no. 12*, Dec. 2012, pp. 4296–4307.

[3] D. Bharadia, E. McMilin, and S. Katti, "Full duplex radios," in *ACM SIGCOMM*, 2013, pp. 375–386.

[4] A. Sahai, G. Patel, C. Dick, and A. Sabharwal, "On the impact of phase noise on active cancelation in wireless full-duplex," *IEEE Trans. Veh. Technol.*, vol. 62, no. 9, pp. 4494–4510, Nov. 2013.

[5] V. Syrjala, M. Valkama, L. Anttila, T. Riihonen, and D. Korpi, "Analysis of oscillator phase-noise effects on self-interference cancellation in full-duplex OFDM radio transceivers," *IEEE Trans. Wireless Commun.*, vol. 13, no. 6, pp. 2977–2990, June 2014.

[6] L. Anttila, D. Korpi, E. Antonio-Rodrìguez, R. Wichman, and M. Valkama, "Modeling and efficient cancellation of nonlinear selfinterference in MIMO full-duplex transceivers," in *Globecom Workshops*, 2014, pp. 777–783.

[7] A. Balatsoukas-Stimming, A. C. M. Austin, P. Belanovic, and A. Burg., "Baseband and RF hardware impairments in full-duplex wireless systems: experimental characterisation and suppression," *EURASIP J. on Wireless Comm. and Netw.*, vol. 2015, no. 142, 2015.

[8] D. Korpi, L. Anttila, and M. Valkama, "Nonlinear self-interference cancellation in MIMO full-duplex transceivers under crosstalk," *EURASIP J. on Wireless Comm. and Netw.*, vol. 2017, no. 1, p. 24, Feb. 2017.

[9] A. Balatsoukas-Stimming, "Non-linear digital self-interference cancellation for in-band full-duplex radios using neural networks," in *IEEE Int. Workshop on Signal Proc. Advances in Wireless Commun. (SPAWC)*, Jun. 2018, pp. 1–5.

[10] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing FPGA-based accelerator design for deep convolutional neural networks," in *ACM/SIGDA Int. Symp. on Field-Programmable Gate Arrays*, Feb. 2015, pp. 161–170.

[11] Y. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE J. of Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, Jan. 2017.

[12] A. Balatsoukas-Stimming, P. Belanovic, K. Alexandris, and A. Burg, "On self-interference suppression methods for low-complexity full-duplex MIMO," in *Asilomar Conf. on Signals, Systems and Computers*, Nov. 2013, pp. 992–997.

[13] P. Belanovic, A. Balatsoukas-Stimming, and A. Burg, "A multipurpose testbed for full-duplex wireless communications," in *IEEE Int. Conf. on Electronics, Circuits, and Systems (ICECS)*, Dec. 2013, pp. 70–71.