RadioML Meets FINN: Enabling Future RF Applications With FPGA Streaming Architectures

Felix Jentzsch, *Paderborn University, 33098, Paderborn, Germany*
Yaman Umuroglu, Alessandro Pappalardo, and Michaela Blott, *AMD, D24 T683, Dublin, Ireland*
Marco Platzner, *Paderborn University, 33098, Paderborn, Germany*

Deep neural networks (DNNs) are penetrating into a broad spectrum of applications and replacing manual algorithmic implementations, including the radio frequency communications domain with classical signal processing algorithms. However, the high throughput (gigasamples per second) and low latency requirements of this application domain pose a significant hurdle for adopting computationally demanding DNNs. In this article, we explore highly specialized DNN inference accelerator approaches on field-programmable gate arrays (FPGAs) for RadioML modulation classification. Using an automated end-to-end flow for the generation of the FPGA solution, we can easily explore a spectrum of solutions that optimize for different design targets, including accuracy, power efficiency, resources, throughput, and latency. By leveraging reduced precision arithmetic and customized streaming dataflow, we demonstrate a solution that meets the application requirements and outperforms alternative FPGA efforts by 3.5× in terms of throughput. Against modern embedded graphics processing units (GPUs), we measure >10× higher throughput and >100× lower latency under comparable accuracy and power envelopes.

Deep learning is rapidly expanding into new horizons, including the communications space. Traditionally, this is a mature field of engineering, where solutions are carefully crafted by experts. However, the ever-increasing complexity of communication networks has prompted the exploration of deep neural networks (DNNs) for various use cases, ranging from traffic monitoring tasks to the physical interface design of radio frequency (RF) systems.[1] The latter is one example of the "RadioML" domain, where conventional radio signal processing is replaced by DNN-based processing. While this approach has shown great potential,[2] it also comes with great challenges, as radio signals are handled exclusively at the edge, often on highly constrained mobile devices that lack the compute or energy budget to run modern DNNs with sufficient performance. Traditional compute accelerators, such as graphics processing units (GPUs), are well-optimized for the huge DNNs in vision-based applications, but models used for RadioML pose a different challenge as they are typically much smaller and operate on short frames of time-series data. In turn, the live processing of an RF signal demands extreme throughput and ultra-low latency, which lies well beyond what is currently possible with GPUs.

To tackle these unique challenges of RadioML, joint specialization of the hardware accelerator and the DNN itself is key. Field-programmable gate arrays (FPGAs) offer the flexibility and parallelism to satisfy these requirements, but harnessing this potential is difficult, especially for nonexperts or under tight development time constraints. Here, we make the case for automatically-generated, custom-tailored accelerators for quantized DNNs, enabled by Xilinx' open-source FINN compiler framework. These accelerators follow the *streaming dataflow* architectural paradigm, involving layer-parallel processing of

0272-1732 © 2022 IEEE
Digital Object Identifier 10.1109/MM.2022.3202091
Date of current version 28 October 2022.

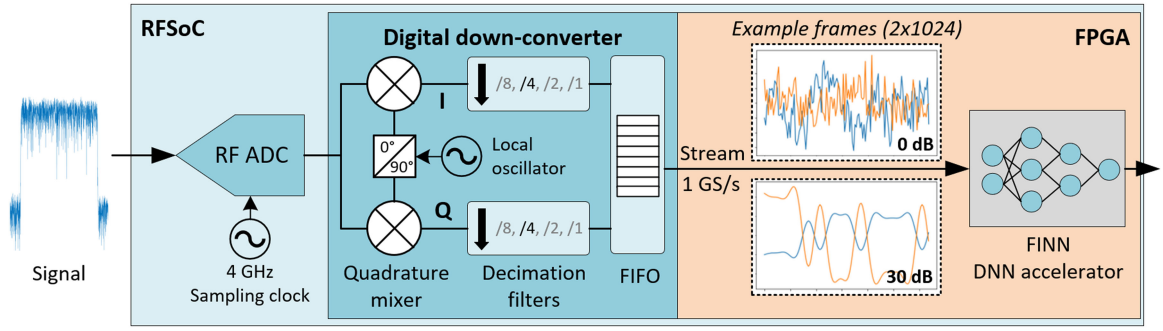

FIGURE 1. RFSoC integration of (simplified) RF front-end and DNN accelerator. Input frame examples are taken from the modulation classification dataset and come in the form of baseband in-phase/quadrature (I/Q) components (orange/blue).

the input stream, which departs from conventional compute arrays and enables deep pipelining and memory access minimization.

By way of example, we study one of the most popular RadioML use cases: automatic modulation classification. In this task, a DNN is trained to classify the modulation scheme [e.g., frequency modulation (FM), binary phase-shift keying (BPSK), quadrature amplitude modulation (QAM)-16, etc.] of a received signal. We use the open "RadioML 2018" dataset from DeepSig,[3] which covers a wide range of modulation types and signal-to-noise ratios (SNRs) and provides a baseline 1-D convolutional neural network (CNN), showing that it can outscore traditional classification based on engineered features, such as higher order moments.

In this work, we first make the case for a well-suited target platform for modulation classification and present our end-to-end tool flow to accelerate the involved CNN. Then, we report on several FINN-generated prototypes and compare them with related FPGA implementations and GPU platforms.

CASE FOR RadioML ON RFSoC

Next-generation radio architectures need platforms that can address a wide range of requirements with the same basic hardware. This adaptability is critical to accommodate emerging and ever-changing standards. FPGAs have historically provided flexible solutions for implementing the digital front-end and interfacing requirements of recent radio generations. RF system-on-chip (RFSoC) devices, such as Xilinx' Zynq UltraScale RFSoC, improve this level of flexibility by integrating RF-sampling data converters into a single chip, next to an ARM-based processing system and programmable logic fabric. Direct RF sampling, together with optimized digital signal processing (DSP) engines, offers a much more flexible approach to traditional analog frequency translation and filtering by enabling much of the signal processing to be done in the digital domain. This also eliminates the need for external input/output interfaces, which can consume a significant amount of power. In addition, the availability of FPGA programmable logic on the same device enables direct integration of downstream applications, such as DNN processing.

Figure 1 shows how such a DNN accelerator can be integrated on an RFSoC for classifying the modulation of a received I/Q modulated signal, commonly used for software-defined radios (SDRs). The datapath begins with the analog-to-digital converter (ADC), which samples the RF signal at up to 4 gigasamples per second (GS/s). Next, the samples pass through the configurable digital down-converter block, where a quadrature mixer shifts the signal from its carrier frequency down to the equivalent baseband signal. The resulting streams of in-phase (I) and quadrature (Q) components can then be down-sampled by a factor of one to eight using decimation filters before they are stored in a first-in–first-out (FIFO) gearbox buffer, resulting in an output data rate between 4 and 0.5 GS/s for the maximum ADC sample rate. In the case of our modulation classification example, this I/Q signal representation corresponds to approximately 2.5 million frames of length 1,024. Figure 1 illustrates two exemplary frame segments of BPSK-modulated signals at 0- and 30-dB SNR. Data streams within the FPGA are implemented via the AXI-Stream protocol and feed in and out of the inference accelerator. The final output is a frame's classification result.

FINN FRAMEWORK FOR STREAMING DNN ARCHITECTURES

The open-source framework FINN[4] generates specialized DNN accelerators for FPGAs using streaming dataflow architectures, with the hardware architecture customized to the specifics of a DNN topology

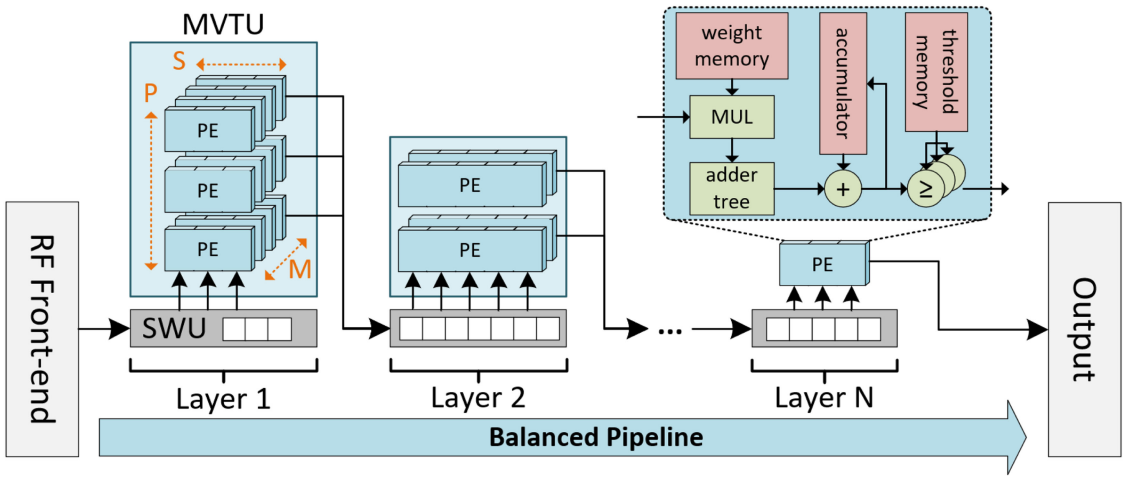

FIGURE 2. Simplified architectural overview of the FINN-generated streaming accelerator. Omits padding, pooling, and gearbox buffers between layers.

and particular datatypes used. Each layer is instantiated with its designated compute units in hardware. On-chip data streams interconnect the compute units to form the desired network topology. The small and compact size of reduced-precision quantized DNNs (QNNs) allows us to scale performance of the accelerator via reduced resource requirements of lower precision operators and store all parameters on the chip, thus avoiding external memory bottlenecks. To achieve high accuracy for low precision QNNs, we leverage the open-source PyTorch library Brevitas.[5]

Hardware Architecture

Figure 2 visualizes the FINN-generated architecture for a CNN, which comprises a balanced pipeline of computing blocks connected through on-chip streams. FINN maps each fully connected DNN layer to a dedicated compute block, the matrix–vector threshold unit (MVTU). The MVTU performs matrix multiplication between input activations and weights, followed by the so-called "multithreshold" operation, which applies the nonlinear activation function and quantization to the output in a single, efficient step. Convolution layers are based on the same MVTU structure by lowering them to matrix–matrix multiplications where the MVTU is fed by a sliding window unit (SWU), a special stream buffer that enables windowed access to the input feature map while minimizing memory requirements.

The MVTU is parameterized in terms of input, output, and weight precision, as well as the type of resource [e.g., look-up tables (LUTs) or block memory (BRAM)] used for its internal weight and threshold memories. Furthermore, the MVTU can be parallelized in various dimensions limited only by the DNN topology and the available programmable logic resources. The dimensions comprise P parallel processing elements (PEs), which determine the number of output channels processed in parallel, and S single-instruction–multiple-data (SIMD) input channel lanes for each PE. To further scale small DNNs, such as the ones for RadioML, we extend FINN to support an additional degree of parallelism by processing M output positions simultaneously.

FINN Compiler Tool Flow

Figure 3 shows the FINN tool flow, which has a modular structure that allows the user to interactively generate a specialized architecture for a specific DNN. The framework provides a front-end, the FINN compiler with its transformation and analysis passes, and high-level synthesis (HLS)-based back-end to explore the design space in terms of resource and performance constraints. While users can build a custom step-by-step flow using the provided infrastructure, FINN also offers an automatic build flow that optimizes common DNN topologies based on a performance target and device constraints. Internally, FINN is built around an end-to-end intermediate representation (IR) based on the open neural network exchange (ONNX) format for DNN graphs. The IR also serves as the input format for quantized models, which are exported from a training front-end, such as Brevitas.

Starting from the input model, the FINN compiler performs three phases of graph transformation passes, which analyze and change the IR to gradually map it to a synthesizable accelerator architecture. In the preparation

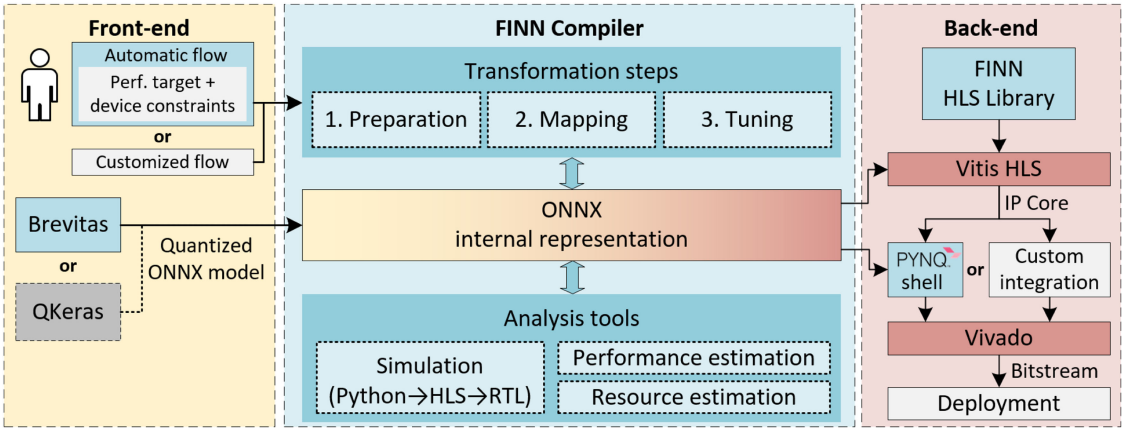

FIGURE 3. Overview of the FINN compiler flow.

phase, DNN graph nodes are rearranged or fused to make them compatible with how the back-end building blocks operate, for example, in terms of data layout. This includes convolution lowering and a "streamlining" process to merge quantization and batch-normalization factors into multithreshold operations. The mapping phase associates layers with the configurable operators implemented by the FINN HLS library, such that each node corresponds to a Vitis HLS C++ function call, which can later be synthesized to an IP block. In the tuning phase, the resource and parallelism configurations of the MVTUs are determined. A "folding" process assigns compute resources via a selection of P, S, and M to each layer to obtain the desired throughput within a balanced pipeline. Bottlenecks due to bursty behavior are avoided by automatically inserting stream buffers. FINN employs various analysis tools to guide the mapping phase. These include model-based performance and resource estimates, as well as simulation and reporting on all abstraction levels.

Finally, FINN generates code from the IR, synthesizes, and stitches the layers together. The resulting standalone IP core can be integrated into any design or deployed quickly with the generated shell project and driver for Xilinx Alveo and PYNQ platforms.

Brevitas

Brevitas[5] is a PyTorch extension for neural network quantization, with a focus on quantization-aware training (QAT). It provides building blocks to model a reduced precision inference data path at training time. Due to its flexibility, DNN models can be adopted to target different styles of fixed-point computing. By accounting for the additional error introduced by quantization at training time, QAT provides superior results in terms of accuracy compared to post-training quantization approaches and can gracefully scale the precision of both parameters and activations down to binary values.

For a given target datatype, Brevitas exposes multiple hyperparameters that a user can tune to adjust the quantization algorithm to the particular training problem at hand. For example, the scale factor of a given datatype, which for traditional fixed-point datatypes is a power-of-two number, can be set to a user-defined constant, a user-initialized value learned with backpropagation, or a value initialized according to some statistics and then learned with backpropagation.

Once the network has been trained, Brevitas can export it to a downstream toolchain by encoding it in an intermediate format. For use in the FINN framework, Brevitas extends ONNX by introducing *ad hoc* quantization nodes to specify custom fixed-point datatypes.

MODULATION CLASSIFICATION CASE STUDY

We choose the automatic modulation classification use case to showcase the potential of the FINN approach for RadioML.

Models and Training

We train our models on the RadioML 2018 dataset, which contains signals in 24 different modulation schemes at an SNR range from -20 dB to $+30$ dB. To limit the design space and provide a better comparison with related work, we stay very close to the "VGG10" topology proposed alongside the dataset by DeepSig.[3] This DNN consists of seven one-dimensional convolution layers with a kernel size of three, each followed by batch-normalization, ReLU activation, and max-pooling to reduce the output feature map size by half. This CNN block is followed by two

TABLE 1. Results of FINN prototypes against existing FPGA implementations. Utilization for XCZU28DR device.

Implementation	FINN A	FINN B	FINN C	Best from Tridgell et al.[6]	Best from den Boer et al.[7]
Topology (F_c, F_d)	VGG10 (64, 128)	VGG10-S (32, 128)	VGG10-S (32, 128)	VGG10-L (128, 512)	Other 1D-Conv
# Parameters	161,000	72,000	72,000	636,000	14,000
Quantization (weight/activation)	4 bits (5-bit first layer)	4 bits	4 bits	weights: ternary activations: mixed	6 bits
Frequency	250 MHz	250 MHz	250 MHz	250 MHz	250 MHz
LUTs (util.)	267,000 (63%)	65,000 (15%)	229,000 (54%)	211,000 (50%)	106,000 (25%)
Flip-flops (util.)	120,000 (14%)	42,000 (5%)	131,000 (15%)	324,000 (38%)	61,000 (7%)
BRAM blocks (util.)	56 (5%)	25 (2%)	26 (2%)	512 (47%)	0 (0%)
DSP slices (util.)	0 (0%)	0 (0%)	0 (0%)	1407 (33%)	137 (3%)
Accuracy @ 30 dB	94.1%	91.0%	91.0%	80.2%	71.8%
Throughput [samples/s]	246 million	246 million	1750 million	500 million	250 million
Latency [μs]	11.7	11.3	2.6	8.0	4.6

dense layers and a final dense classification layer. Besides the weight and activation quantization, we adjust only the number of filters in the convolution layers (F_c) as we found this to be the second-most effective method for trading off accuracy and compute cost.

For all models, we quantize inputs to a fixed 8-bit range determined by statistical analysis of the dataset to yield a low quantization error at high SNR ($\geq$ 6 dB) for most modulations. Two variants of single-sideband amplitude modulation deviate from the Gaussian-like distribution of other modulations and perform somewhat worse with quantization, which we deem an acceptable tradeoff. We observe that the relative SNR of the data that the network is trained and tested on has a large impact on recognition performance. While low-SNR environments are important for practical applications, we focus on training and testing on high-SNR data for brevity and to facilitate comparisons to related work.

During training, we approximate the compute cost for each model by calculating the number of "bit-operations" (BOPS) as a sum of all multiply-and-accumulate operations weighted by their respective operand bit-widths since this metric exhibits an approximately linear relationship to resource consumption of the resulting FINN accelerators under the same folding configuration. We find that quantizing weights and activations for the original VGG10 topology with $F_c = 64$ below a bit-width of 4 does not result in a compelling utilization-accuracy tradeoff. Instead, decreasing the number of convolution filters to $F_c = 32$ yields better accuracy and less cost than the original VGG10 quantized to 2-bit weights and activations, even if 4-bit precision is kept for the more quantization-sensitive input CNN layer. We refer to this smaller model variant as "VGG10-S" and select it alongside the original model for accelerator generation.

Prototype Results

Based on the two models VGG10 and VGG10-S, we build three distinct accelerator prototypes, each in a different corner of the vast design space. We target the XCZU28DR RFSoC device found on the ZCU111 and PYNQ RFSoC 2×2 development boards. Table 1 shows key metrics of the prototypes. Prototype "FINN A" is based on VGG10 and represents the most accurate—but resource-intensive—implementation with a peak accuracy of 94.1%, a 2.6 p.p. drop from the floating point (FP) baseline we trained to 96.7% accuracy. As for performance, FINN is configured to apply full parallelism across the input and output channel dimension, limiting throughput to one sample per FPGA clock cycle, with an actual throughput that is slightly lower (246 MS/s at 250 MHz) due to padding and pipeline inefficiencies. Note that we report throughput in samples per second instead of frames per second to decouple it from the frame size, which is 1,024 as in training.

"FINN B" is the smallest prototype and applies the same configuration to the VGG10-S model, resulting in the same performance, but lower accuracy at 91.0%. For "FINN C," we scale up the parallelism by extending

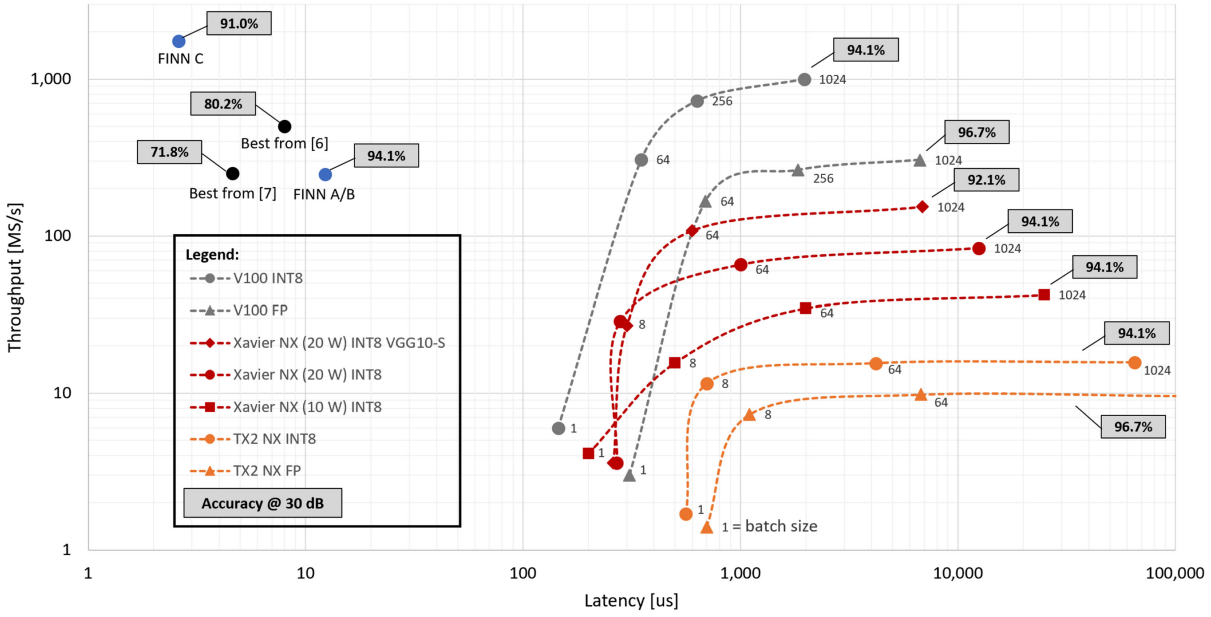

FIGURE 4. Throughput versus latency of VGG10 inference on various platforms. GPU results are batch-size-dependent and report end-to-end host latency. FPGA results assume a direct input data feed and report compute latency.

FINN to allow simultaneous processing of up to 8 samples/cycle. This unlocks the unprecedented performance of 1.75 GS/s at a latency of just 2.6 μs. This increases resource efficiency (throughput over utilization) by a factor of 2. We attribute this nonlinear scaling mainly to synthesis optimization: due to the pooling structure, a balanced pipeline that takes in multiple samples simultaneously has to apply full channel unfolding to more layers, which allows for the elimination of PE multiplexing logic and zero-weight multiplications. This results in a device utilization of 54%, with significant resources still available for additional features or performance scaling. In general, adjusting parallelism and bit-width alone can scale the implementation to performance, resource, or accuracy targets well beyond what we show here.

Table 1 also includes top-performing accelerators from related work that target the same device family. The implementation described in Tridgell et al.[6] uses a larger variant of the model with more convolution and dense filters (F_d) but applies harsh quantization with ternary weights and mixed activation formats. The DNN is mapped to hardware using a custom HDL-generation framework. The reported peak accuracy is significantly lower than our results and the design manages only 2 samples/cycle, despite the relatively high resource consumption.

In contrast, the prototype shown in den Boer et al.[7] implements a custom HLS-based mapping tool and focuses on smaller CNN topologies, but uses 6-bit operands. Even for their largest model, accuracy is low (71.8%) and the reported throughput is 1 sample/cycle. Further related work is discussed in the sidebar.

GPU Comparison

To compare performance and energy efficiency with current GPUs, we use NVIDIA's TensorRT tool to run automatically optimized inference benchmarks on the Tesla V100 data-center GPU and two modules of the Jetson embedded GPU family: the previous-generation TX2 and the current-generation Xavier NX, which we run in its lowest (10 W) and highest (20 W) power modes. The Xavier NX also features dedicated INT8 compute support for more efficient DNN inference. We utilize this by applying 8-bit post-training quantization to the FP models using TensorRT, incurring an accuracy drop from 96.7% to 94.1% for VGG10 and from 95% to 92.1% for VGG10-S, although this may be alleviated via QAT.

Figure 4 shows the resulting throughput over latency, both in logarithmic scale, against the discussed FPGA accelerators. For the GPUs, we report end-to-end latency, which includes data transfer and synchronization overhead that typically ranges from 1% to 10% in this case. While the streaming accelerators take in samples as they are supplied from the digital radio front-end, and do not even need to buffer a single frame before computation begins, GPUs require

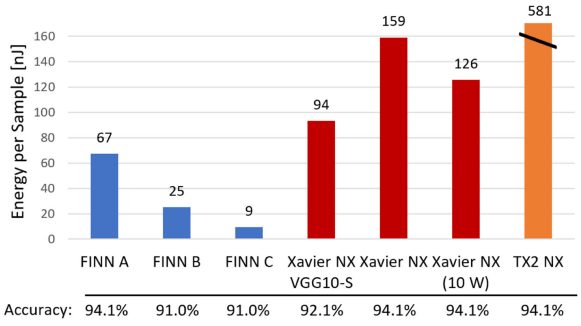

FIGURE 5. Power efficiency of FINN accelerators versus NVIDIA Jetson GPU platforms (batch-64, INT8).

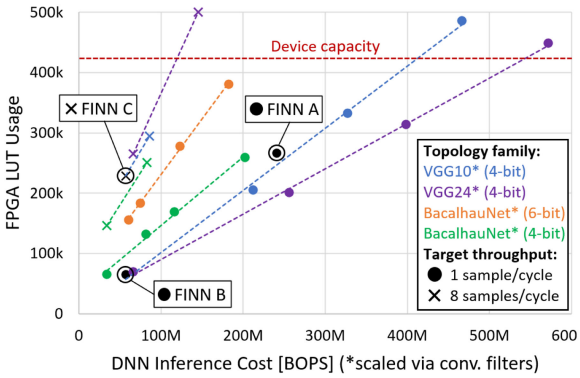

FIGURE 6. Scaling to larger DNNs and different topologies. Each data point represents a FINN-generated accelerator for a DNN of the specified topology family and inference cost, which is scaled along the *x*-axis by increasing the number of convolution filters. Our three accuracy-optimized prototypes (A, B, and C implementations from Table 1) are marked as singular experiments for reference.

the aggregation of frames into batches to utilize their computing power. This is evident in our measurements, as peak throughput (153 MS/s for Xavier NX) and minimum latency (260 μs for Xavier NX) are not achievable at the same time. Regardless, even this latency is 100× higher than that of our prototype FINN C. When comparing our solution to devices of a similar power envelope, i.e., the embedded GPUs, it becomes clear how only the FPGA streaming accelerator is capable of keeping up with the RF data rate of multiple gigasamples per second, all while delivering exceptional microsecond latency. We expect the direct RFSoC integration to only amplify this advantage for real-world systems, where additional overhead will be needed to feed signals to GPUs.

Figure 5 compares measured power efficiency in terms of energy per processed sample of FINN accelerators and embedded GPUs. Our fastest and most efficient prototype consumes 16.5 W and is 17× more power efficient than the INT8 VGG10 on Xavier NX with 10.5 W, albeit with 3.1 p.p. lower accuracy. For a fairer comparison, we also run VGG10-S on the Xavier NX. Even if we assume that quantization-aware training could improve the INT8 accuracy to FP level, FINN A would lie within one percentage point of accuracy while delivering 1.4× the efficiency and 2.3× the throughput.

Scalability

To demonstrate FINN's scalability beyond the accuracy-optimized prototypes discussed before, we synthesize accelerators for larger VGG10 instances and two additional topology families: VGG24, a deeper variant of VGG10 with 3× the convolution layers, and "BacalhauNet,"[8] the winning DNN of the "Lightning-Fast Modulation Classification" problem statement of the 2021 ITU AI in 5G Challenge, which features depthwise-separable convolutions with wide kernel dimensions and residual connections. We measure DNN size as inference compute cost per classification (in BOPS) and scale it for each topology family via the number of convolution filters. As a third variable, the generated accelerators target two levels of throughput. Figure 6 plots the resulting FPGA utilization, showcasing the proportional relationship between accelerator LUT count and DNN size.

AVOIDING OBSTRUCTIVE MANUAL DESIGN EFFORTS, THE OPEN-SOURCE FINN FRAMEWORK AUTOMATICALLY GENERATES DNN ACCELERATORS AND REACHES UNPRECEDENTED RESULTS IN ALL RELEVANT METRICS, INCLUDING ENERGY EFFICIENCY.

In summary, VGG24 scales best to larger models, especially when compared to the original 6-bit BacalhauNet. A more sensible 4-bit variant performs close to VGG10 at a nominal throughput of 1 sample/cycle and scales just as well to the extremely parallel 8 samples/cycle configuration, reaching the real-world performance of 1.8-GS/s and 1.5-μs latency at 250 MHz. In some scenarios, the DNN scaling method does not quite allow for full device utilization due to underlying tool limitations. A fair power comparison against INT8 TensorRT execution on the Xavier NX platform is

RELATED WORK

SDR systems with hardware acceleration are an emerging field. Deepwave Digital released the first commercial "AI Radio Transceiver" (AIR-T)[9] in early 2020, a device that integrates a transceiver, an FPGA, and an NVIDIA Jetson TX2 embedded GPU with a GNU-Radio software stack. The FPGA performs necessary DSP tasks and acts as a bridge between the serial transceiver interface and the GPU, which is responsible for DNN inference. The system operates in the 100–200-MS/s range and relies on the shared-memory architecture of modern embedded GPUs to minimize latency, as no additional memory copy between the host (CPU) and GPU is necessary. In contrast, we combine all functions into a single SoC and remove even the last remaining external memory transfer between the transceiver and the accelerator. We also address an often mentioned argument against FPGA-based solutions, the high development effort, with our automated FINN tool flow.

Regarding the use case modulation classification, we highlight two state-of-the-art FPGA accelerator approaches[6,7] in Table 1. Other FPGA implementations for modulation classification exist and include the use of support vector machines[10] for classification, DNN processing of engineered statistical features,[11] and even spiking neural networks.[12] However, most of these approaches are not comparable to our work, e.g., due to undisclosed data sets, or lack of a flexible toolchain and meaningful GPU comparison. On the training side, efforts have been made to create more resource-efficient DNN topologies through the use of sparsity, residual connections, depthwise convolutions, or recurrent neural networks. We refer to Jdid et al.[13] for a survey.

difficult without in-depth accuracy testing, but results from our scaling experiments continue to suggest a strong efficiency advantage for FINN, which ranges from 3.4× for the largest VGG24 over 4.2× for the largest 6-bit BacalhauNet to around 30× for the fast 4-bit variants. In terms of raw performance, even the smallest BacalhauNet does not surpass 65-MS/s throughputs and 1-ms latency (batch-64) on the Xavier NX.

CONCLUSION

On the basis of modulation classification, we have shown how FPGA streaming architectures suit DNN-based RF signal processing perfectly, especially in RFSoC systems with integrated radio front-ends. Avoiding obstructive manual design efforts, the open-source FINN framework automatically generates DNN accelerators and reaches unprecedented results in all relevant metrics, including energy efficiency. Compared to current embedded GPUs, we achieve orders of magnitude better latency (microseconds versus milliseconds) and throughput (gigasamples per second versus hundreds of megasamples per second), while keeping the quantization-induced accuracy penalty under control. Future work includes the implementation of a live RFSoC demonstrator and exploration of promising DNN topologies and techniques, such as ResNets and sparsity.

REFERENCES

1. T. J. O'Shea, K. Karra, and T. C. Clancy, "Learning to communicate: Channel auto-encoders, domain specific regularizers, and attention," in *Proc. IEEE Int. Symp. Signal Process. Inf. Technol.*, 2016, pp. 223–228.
2. T. Erpek, T. J. O'Shea, Y. E. Sagduyu, Y. Shi, and T. C. Clancy, "Deep learning for wireless communications," 2020, *arXiv:2005.06068*.
3. T. J. O'Shea, T. Roy, and T. C. Clancy, "Over-the-air deep learning based radio signal classification," *IEEE J. Sel. Topics Signal Process.*, vol. 12, no. 1, pp. 168–179, Feb. 2018.
4. M. Blott et al., "FINN-R: An end-to-end deep-learning framework for fast exploration of quantized neural networks," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 11, no. 3, Dec. 2018, Art. no. 16.
5. A. Pappalardo, "Brevitas (software)." Accessed: Aug. 22, 2022. [Online]. Available: https://doi.org/10.5281/zenodo.5779154
6. S. Tridgell, D. Boland, P. H. Leong, R. Kastner, A. Khodamoradi, and Siddhartha, "Real-time automatic modulation classification using RFSoC," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. Workshops*, 2020, pp. 82–89.
7. H. den Boer, R. Muller, S. Wong, and V. Voogt, "FPGA-based deep learning accelerator for RF applications," in *Proc. IEEE Mil. Commun. Conf.*, 2021, pp. 751–756.

8. J. Rosa et al., "BacalhauNet: A tiny CNN for lightning-fast modulation classification," *ITU J. Future Evolving Technol.*, vol. 3, no. 2, pp. 252–260, 2022.
9. J. Ferguson, P. Witkowski, W. Kirschner, and D. Bryant, "Deepwave digital creates an AI enabled GPU receiver for a critical 5G sensor," white paper, 2020. [Online]. Available: https://developer.nvidia.com/blog/wp-content/uploads/2020/01/NVIDIA_Blog_v2.pdf
10. C. Cardoso, A. R. Castro, and A. Klautau, "An efficient FPGA IP core for automatic modulation classification," *IEEE Embedded Syst. Lett.*, vol. 5, no. 3, pp. 42–45, Sep. 2013.
11. A. F. de Castro, R. S. R. Milléo, L. H. A. Lolis, and A. A. Mariano, "Artificial neural network based automatic modulation classification system applied to FPGA," in *Proc. ACM Symp. Integr. Circuits Syst. Des.*, 2021, pp. 1–6.
12. A. Khodamoradi, K. Denolf, and R. Kastner, "S2N2: A FPGA accelerator for streaming spiking neural networks," in *Proc. ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays*, 2021, pp. 194–205.
13. B. Jdid, K. Hassan, I. Dayoub, W. H. Lim, and M. Mokayef, "Machine learning based automatic modulation recognition for wireless communications: A comprehensive survey," *IEEE Access*, vol. 9, pp. 57851–57873, 2021.

FELIX JENTZSCH is a Ph.D. student with Paderborn University. His research focuses on automated hardware/software co-design for reconfigurable computing systems. Jentzsch received a master's degree in computer engineering from Paderborn University. He is a Member of IEEE. Contact him at felix.jentzsch@upb.de.

YAMAN UMUROGLU is a senior MTS with AMD, Dublin, Ireland. His research interests include full-stack view of DNNs with a focus on high efficiency and spans hardware-network co-design, techniques for efficient arithmetic, sparsity and quantization. Contact him at yamanu@amd.com.

ALESSANDRO PAPPALARDO is a staff researcher with AMD, Dublin, Ireland. His research interests include neural network co-design and acceleration on reconfigurable hardware. Contact him at alessand@amd.com.

MICHAELA BLOTT works as a Senior Fellow with AMD, Dublin, Ireland. Her research interests include compute architectures, reconfigurable computing, and machine learning. She is a Member of IEEE. Contact her at mblott@amd.com.

MARCO PLATZNER is a professor for computer engineering with Paderborn University, Paderborn, Germany. His research interests include reconfigurable computing, hardware-software co-design, and parallel architectures. He is a Senior Member of IEEE. Contact him at platzner@upb.de.