Exploring and Evaluating the Suitability of FPGAs for Edge Computing

Robert Canady*, Xingyu Zhou*, Travis Brummett*, and Aniruddha Gokhale
Vanderbilt University, Nashville, TN, USA
Email:{robert.e.canady,xingyu.zhou,travis.s.brummett,a.gokhale}@vanderbilt.edu

Abstract

With the rapid growth in edge computing/internet of things (IoT), much research is centered around determining the kinds of applications and resources that are suitable for edge environments. However, not much data is available on whether popular programmable logic devices like field programmable gate arrays (FPGAs) are suitable in these contexts. In this paper, we conduct a feasibility study hosting a set of FPGA-based applications at the edge, which reveals the potential benefits of using FPGAs in edge/IoT scenarios. To conduct such a study and enable wider use by the scientific community, we developed EACBench (Edge Acceleration Computing Benchmark), which is an edge-based hardware acceleration testbed and benchmark. The testbed comprises a small cluster of FPGA hardware. Further, to enable large-scale edge-based experimentation, we use the parameters obtained from our small scale testbed to configure an open source simulation framework for edge/IoT computing called iFogSim. Experimental results from these studies indicate that FPGAs can achieve both low latency and low power, and are suitable in a wide range of edge applications.

1 Introduction

With the rapid growth in edge computing/internet of things (IoT), one issue that is often encountered is what computational device would be best suited for a given task under different operating scenarios [6]. For traditional, general-purpose computing applications with no specific restrictions, we can often use commonly found hardware options of CPU and GPU. However, in case of edge computing/IoT, there often exists restrictions like power consumption, battery consumption, timing requirements of an application and other constraints on the device, e.g., memory, storage, cpu speed.

In edge computing settings, CPU types can range from tiny micro-controllers to standard servers at the fog layer, which is a small-scale data center. The former has limited computation capability while the latter often has high cost in power or size. To gain maximum computation capability given a fixed power constraint, some devices are designed to serve the needs of specific applications. For instance, advances in machine learning have led to the popularity of neural acceleration chips like Intel neural compute stick and Google TPU. The devicex that provide best performance efficiency typically are the ASICs (application-specific integrated circuits). However, ASICs have a high threshold for design and manufacturing, and usually have a limited range of applications for which they are applicable.

To compensate for these disadvantages, field programmable gate arrays (FGPAs) are often regarded as a more versatile form of ASICs that can achieve both high efficiency and generalization. A FPGA-based application is realized by designing a circuit for each function of the application. However, one significant barrier to the more widespread use of FPGAs lies in a general lack of research in understanding the application domains that will benefit significantly using programming logic compared to general-purpose CPUs or other specialized devices. Specifically, the benefits of using FPGAs for edge applications are still not well-understood.

Addressing these challenges is hard due to the substantial flexibility of programming logic devices, which makes it infeasible to provide a general model to depict their performance accurately for a range of applications in a variety of operating conditions. A recent work exploring the use of FPGAs at the edge for matrix multiplication tests shows that programmable logic can also overcome limitations of GPUs with respect to throughput sensitivity to workload size, architectural adaptiveness to algorithm characteristics and energy efficiency [1] . Although this related work is a step in the desired direction, it focuses on only one application. To overcome this limitation, in this paper we choose several typical applications of FPGAs and use them to provide a baseline understanding of the suitability of FPGAs at the edge. Our work thus provides a benchmarking suite to understand the benefits of hardware acceleration at the edge.

In the remainder of the paper Section 2 describes the design

*These authors contributed equally

of a benchmark and testbed called EACBench; Section 3 provides results of our experiments on the testbed; Section 4 scales the experiments using a simulation setup; and Section 5 concludes the paper. Section 6 offers discussion topics.

2 Design of EACBench

We now describe the design and methodology of the Edge Acceleration Computing Benchmark (EACBench).

2.1 Methodology behind EACBench

Traditionally, FPGA-based design requires using RTL (register transfer language) like VHDL or Verilog, which however has not found widespread adoption due to difficulty in using these techniques. High-level synthesis (HLS) is a technology in circuit industry which can translate behavioral-level design descriptions into RTL descriptions. This helps overcome some of the earlier challenges making them easier for application developers to test FPGAs for edge applications. We have used HLS concepts in the design of EACBench.

2.2 Test Functions in EACBench

We chose seven practical functions to be part of this edge computing specialized benchmark. Some of these functions (i.e., applications) were taken from existing open source examples. EACBench can be used separately for edge application testing or used as a complementary aid with existing testing standards. The following seven applications are used in EACBench.

1. **Matrix Multiplication:** Matrix multiplication is the most basic data processing numerical computation in practical cases. Matrix computations represent linear transformations and are also used to represent graph relationships. It is worth noticing that one factor that matters is the data type definition, which is used for different optimizations.

2. **Norm Distance:** In many cases when we make comparisons with different input values (vectors), we need a distance metric to measure the difference between data nodes. One typical application is Clustering where the distance metrics are computed repeated until an optimal status is reached.

3. **FIR Filtering:** Finite Impulse Response (FIR) Filters are commonplace in areas of digital signal processing or even in the realm of time series. For edge computing applications that involve sensor data, this is an almost an unavoidable technique for stream processing.

4. **RGB to HSV:** Due to heterogeneity in the edge environment, different data formats might cause much trouble in compatibility. It is reasonable for FPGAs to serve as encoders and decoders in this kind of data communication and transformations. Thus, when edge environments are involved with multiple kinds of video and camera devices, there might be requirements for different color encoders for different process procedures. RGB is the most familiar where three color channels represent three basic colors of red, green and blue. Another commonly used one is HSV. HSV color space describes colors in terms of hue, saturation, and value. It shows colors (hue) with their shade (saturation or amount of gray) and brightness.

5. **Sorting:** Sorting is a well-researched problem that is meaningful for many areas. In essence, sorting is an efficient way to find the value we need in a series of data. Here we take insertion sort as the benchmark because it is easy to implement and has good performance when the input scale is limited.

6. **PID Control:** An important application area of FPGAs on the edge is in control. The proportional-integral-derivative (PID) feedback control is used in a variety of fields requiring continuously modulated control.

7. **Quantized Inference:** Due to the limited hardware resources, a recent trend for edge machine learning is to use pruned, quantized inference engine on the edge. One good example is binary neural network [5, 7]. In binary neural networks, all weights and activations have a boolean range of $\{0, 1\}$. For current deep neural network models, it is quite common for them to have millions of parameters. Quantization [2] can greatly reduce the storage and inference cost while still maintaining the accuracy at a state-of-art level.

2.3 Source-level Overview

Based on the test functions we have discussed above, we can get a sense of the high-level overview of structural characteristics of these functions. Table 1 shows the static characteristics of these test programs.

3 Experimental Settings and Results

As mentioned earlier, high-level synthesis (HLS) is a technology to translate behavioral-level design descriptions into register-transfer level (RTL) descriptions. There are a few branches for HLS using C, C++ or SystemC in different tools. Here, since we discuss the performance of programmable logic devices, we choose VivadoHLS system from Xilinx which can support all these three options. In this part, we will discuss our testbed, the programming logic device settings and corresponding results based on our test functions.

Table 1: Source-level Characteristic of Test Functions

	Description	DataSize	Source	#Line	#BlockFunc	#ToolOptim	#Condition	#Loop
mmult	matrix multiplication	float[32*32]	sdx example	37	2	6	0	7
norm	norm distance	ap_int[16],float	hls example [8]	70	3	4	4	1
FIR	N-tap FIR filtering	in[16]	pp4fpga [3]	26	1	3	1	2
rgb2hsv	RGB to HSV line conversion	stuct[19]	hls example [8]	140	11	6	15	6
sort	insertion sort	int[16]	pp4fpga [3]	35	1	4	4	3
PID	PID controlller	float, ap_fixed,bool	hls example [8]	80	5	3	6	0
BNN	binarynet digit recognition	ap_int[32*32],int	t-huka [4]	530	13	49	24	34

3.1 Target Device Architecture: ZYNQ

There are many types of programmable logic devices. Consider the common application case where streams of data flow into a processing unit, which requires both general-purpose computations including scheduling and application-specific computations. FPGAs are often used for the latter purpose. However, communication costs between the two parts can be high without optimizations. Thus, an integrated chip that can combine these two aspects of functions together is desirable. One such hardware solution we found that satisfies these requirements is the Zynq architecture from Xilinx.

Zynq-7000 devices are equipped with ARM Cortex-A9 processors integrated with 28nm programmable logic that together can maintain a good balance between excellent performance-per-watt and maximum design flexibility. Zynq devices enable high throughput applications. There are many chips in the Zynq-7000 family. We chose the Zedboard (www.zedboard.org) with a relatively high-end chip.

3.2 Custom Optimizations

As Zynq devices contain both PS (ARM processor) and PL (programming logic), optimizations are required not only in the PL customization but also in data communications. Parallelization or pipelining improves the latency and/or throughput of the design. Loop unrolling, loop pipelining, and dataflow pipelining are some of the commonly used optimization techniques.

As a Zynq device contains both an ARM processor and programming logic together, the communications can be of two kinds: (a) the one between the chip and peripheral devices, and (b) internally between the PS (ARM processor) and PL (programming logic). It is often the case that data communication becomes the bottleneck of the design. One key difference in programming logic development is that FPGAs allow full customization of data types and their lengths. A good customization often leads to significant performance and area improvements while maintaining the FPGA accelerator at the desired accuracy level.

3.3 Synthesis Results

To make the development procedures easier and normalized, we choose the next generation Xilinx SDSoc (Software Defined System-on-chip) tool to illustrate the synthesis results for our benchmark functions. The SDSoC development environment provides a high-level C/C++ application development experience with a GUI based on the Eclipse IDE. The most significant feature of SDSoc is that it involves a full-system optimizing compiler. It greatly automates system-level profiling, software acceleration in programmable logic and system connectivity. SDSoc automatically generates communication networks with a universal data access pattern definition. This also makes benchmark comparisons under a more normalized setting.

We use Xilinx SDSoc 2018.3 to implement and test our application functions. As noted before, the target FPGA board is Zedboard in our experiments. The frequency is set to be 100MHz (=10ns/period in each latency unit). For power estimation, we set the ambient temperature at 25 °C. Even though the test functions here are light-weight, we still assume that the PS (Cortex-A9 processor) part is running at a 50% rate.

Table 2 shows the synthesis results for our test functions. We can see that under different application scenarios, the on-chip-power remains at a stable, low level. The absolute length for one period unit is $10ns$ $(1/100MHz)$.

4 Scaling the Experiments via Simulation

The edge/IoT testbed of ours comprising the Zedboards is limited in size and hence incurs significant limitations in conducting edge/IoT experiments at scale. To that end, we use the parameters obtained from using the Zedboard to configure a simulator called iFogSim [**?**]. iFogSim is a modeling tool that allows investigation and comparison of resource management techniques on the basis of QoS criteria such as latency under different workloads (tuple size and transmit rate). As an example, the developers of iFogSim present a case study of an intelligent surveillance through distributed camera networks. It is this example that serves as the basis to build the experiments presented here.

The simulations presented here utilize the same general topologies but are modified to include FPGA devices. The topologies used in these experiments contain various assort-

Table 2: Function Synthesis Results for Zedboard

	Latency/periods	Power/w	DSP	BRAM	LUT	FF	InType/byte	OutType/byte	Transfer Time(cycles)
mmult	130044	1.534	10	48	19447	22249	8192	4096	19142
norm	3960	1.340	2	0	1907	1145	24	4	156
FIR	4980	1.363	39	1	1528	3186	68	4	234
rgb2hsv	12748	1.345	3	0	1784	1899	57	57	370
sort	8659	1.344	0	0	2983	1405	64	64	208
PID	5572	1.342	5	0	2102	1168	33	8	133
BNN	1189518	1.575	0	61	44572	15044	128	1	416

ment of layouts with devices defined with specific parameters. Specifically, we combine the specs of *rgb2hsv* image color conversion on Zedboard with the edge computing simulator iFogSim to further demonstrate the potential benefit of programmable logic devices on the edge at scale since we could not conduct experiments at scale using the limited number of boards available to us.

4.1 Setup of Devices

The devices in the iFogSim simulation are defined based on the parameters found from actual devices. The parameters that iFogSim allows one to define are node name, MIPS, RAM, uplink and downlink bandwidth, the level in the topology, cost rate per MIPS used, busy power, and idle power. Within the simulation topologies, the cloud datacenter is shown at level 0. The cloud parameters are 2660 MIPS, 4096 RAM, 16 bandwidth, 0 cost rate per MIPS used, 16*103 busy power and 16 * 83.25 idle power. The gateways are the only constant between the topologies. A gateway is shown at level 1. The parameters for the gateways used are 2800 MIPS, 4000 RAM, 10000 uplink and downlink bandwidth, 0 cost rate per MIPS used, 107.339 busy power and 83.4333 idle power.

For the rgb2hsv application scenario, the cameras appear at the edge in a set of three at level 2. There are 3 types of cameras where each set contains one of each type. The common parameters are: 1024 RAM, 0 cost rate per MIPS used, 87.53 busy power, and 82.44 idle power. The type-specific parameters are: Type 1 has 1256 MIPS and 40 bandwidth; Type 2 1536 MIPS and 70 bandwidth; and Type 3 has 847 MIPS and 55 bandwidth.

FPGAs are the focus of these experiments. They are used in two separate ways depending on which topology is being presented. In the first case, the cloud at level 0 comprises FPGAs. Alternatively, to gauge the best use of FPGAs, they are used to process the camera-produced data at level 2 in another topology. Regardless of their placement, FPGAs have the same settings of 2660 MIPS, 512 RAM,1776 bandwidth, 0.01 cost rate per MIPS used, 300 busy power, and 300 idle power, which were actual parameters we collected.

4.2 Experimental Topologies

To run the simulations, three separate base topologies were created. Each one uses a cloud-only or edge-based placement strategy. Each topology followed the same general form: a datacenter was linked to three separate edge devices through a gateway. This can be seen in the Figure 1. In an attempt to gauge the effects across larger networks each topology was also given an extended form which can be seen in Figure 2. The topologies are one of three types.

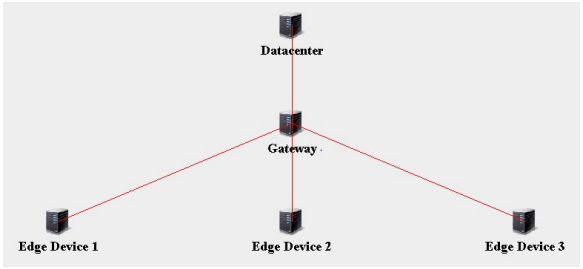

Figure 1: Topology Layout

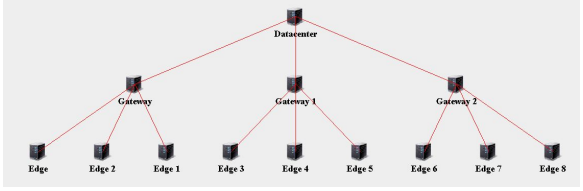

Figure 2: Large Topology Layout

1. *Topology 1*- The first topology is made up of a traditional datacenter processing camera data, and the camera devices on the edge. Each gateway in this topology has a camera of each type attached.
2. *Topology 2*- The second topology uses FPGA resources in the cloud datacenter. It maintains the same edge layout of Topology 1 with cameras.
3. *Topology 3*- The third topology maintains the cloud device as the datacenter but uses FPGA at the edge for processing camera output.

4.3 Simulation Results

We now present the results of our experiments.

4.3.1 Power Consumption

In [**?**], the authors demonstrate that placement affects power consumption. In our simulation results shown in Figure 3, this was observed in the cloud device which increased its power consumption for traditional data center. However, this was not seen with the FPGA at the cloud or at the edge level. Power consumption of each independent devices remained the same across the various topology sizes. The variation in power consumption was not significant enough in these simulations and might be more evident in larger models.

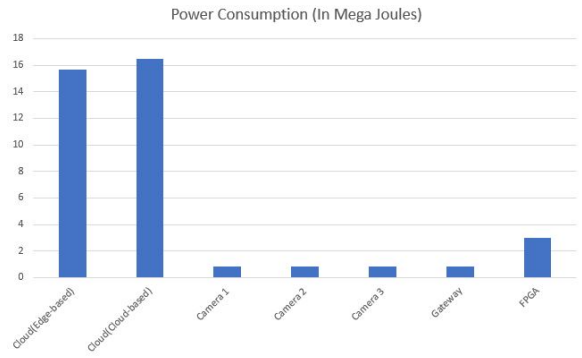

Figure 3: Power Consumption

4.3.2 Time Consumption

The total latency of each simulation decreases as FPGAs were introduced into the topology. The default topology 1 with no FPGAs had the longest latency on the cloud and the second longest on the edge. In topology 2, a FPGA is introduced at the cloud level and latency decreased for both placement strategies maintaining the pattern of the cloud implementation taking longer than the edge. This was also seen with Topology 3 which had the shortest latency. The latencies for the smaller implementations can be seen in Figure 4. Interestingly the only exception to this pattern in the larger implementations is Topology 1. In the larger implementation of this topology the cloud-based scheduling approach executes faster than the edge-based one and the cloud-based implementation of Topology 2. However, the introduction of FPGAs on the edge still decreases the latency. These results can be seen in Figure 5.

From our simulations, FPGAs seem to be working on different levels smoothly. Obviously, their introduction has no effect on the network usage and they consume same power level with camera devices used in these topologies, but they are a more powerful device. FPGAs generally improved latency with one exception to the observed patterns in the cloud-based approach for the larger topology without an FPGA. This might be attributed to the cloud device defined in these experiments and the workload on the edges. However, regardless of the topology size, if it had FPGAs on the edge it would have executed faster. FPGAs seem like a viable option for edge-based systems.

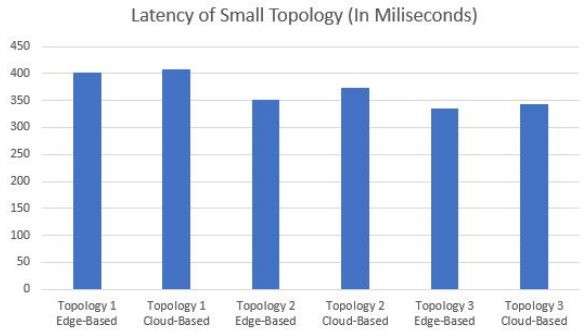

Figure 4: latency of Small Topologies

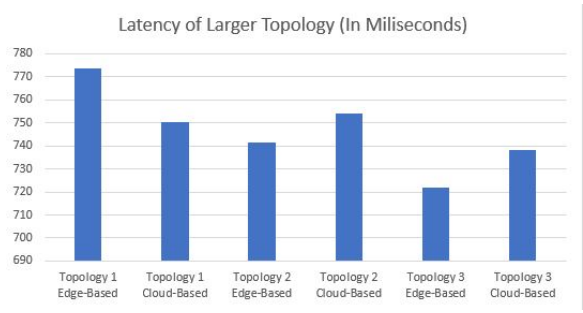

Figure 5: latency of Large Topologies

5 Conclusions and Future Work

In this paper, we studied typical applications of programmable logic devices on edge computing architectures and demonstrate the beneficial roles of FPGAs in edge/IoT scenarios. We design EACBench, which is an edge-computing oriented hardware acceleration testsuite that can be used to help future researchers and engineers decide whether it is a good idea to incorporate FPGA devices. Based on hardware specs, we further incorporate state-of-art edge simulation tools to display roles of FPGAs. Our future investigations will comprise the following. First, we will include network connection applications to fulfill the communication test area. Second, we plan to improve our test functions to make them more efficient. We will also extend these to more programmable logic architectures.

The apparatus for this work is available from github at `https://github.com/doc-vu/EACBench.git`.

6 Discussion Topics

Here we provide details on the discussion topics along the points listed in the call for papers.

6.1 Desired Feedback

Although we have used actual parameters to configure the simulations and obtain the results, we would like to hear from the community if this approach is acceptable, and any other suggestions for scaling up the experiments. We would also like to receive feedback on how we can improve the EACBench to include additional functions and scale its capabilities.

6.2 Controversial Points

In this paper we have focused on the use of FPGAs at the edge. That by itself may be controversial for a variety of reasons. First, programming an FPGA for an application requires additional tooling, which may not be always available. Second, the edge is a highly dynamic environment and there may be a need to dynamically schedule and migrate applications. How can this be made feasible given that the circuitry needs to be programmed and how can this be achieved on-the-fly may become points of debate.

6.3 Expected Interest in the Topic

The use of FPGAs as resource types in the cloud realm is only recently starting to receive attention. Given the many benefits of FPGAs, we believe that a focus on FPGAs for the edge/IoT may spur interest within the community.

6.4 Open Issues

The points we listed above where we desire feedback, and solutions to address the controversial points and limitations listed below form open issues.

6.5 Limitations of the Work

As noted above, we have used simulations to scale EACBench. This was based on parameters obtained using the Zedboard. We have not experimented with other FPGAs and do not know if their performance is similar. Moreover, our experiments used single functions. We need additional experimentation with more complex applications like stream processing applications.

Acknowledgments

This work was supported in part by AFOSR DDDAS FA9550-18-1-0126 and AFRL/LMCO StreamlinedML program. Any opinions, findings, and conclusions or recommendations expressed are those of the author(s) and do not necessarily reflect the views of these funding agencies.

References

[1] Saman Biookaghazadeh, Ming Zhao, and Fengbo Ren. Are fpgas suitable for edge computing? In *Usenix Workshop on Hot Topics in Edge Computing (HotEdge 18)*, 2018.

[2] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[3] Ryan Kastner, Janarbek Matai, and Stephen Neuendorffer. Parallel programming for fpgas. *arXiv preprint arXiv:1805.03648*, 2018.

[4] H. Nakahara. Binary CNN optimized for Zybo (Zynq-7010)). https://github.com/t-kuha/sdsoc/tree/master/binarynet, 2018. [Online; accessed 19-July-2019].

[5] Eriko Nurvitadhi, David Sheffield, Jaewoong Sim, Asit Mishra, Ganesh Venkatesh, and Debbie Marr. Accelerating binarized neural networks: Comparison of fpga, cpu, gpu, and asic. In *2016 International Conference on Field-Programmable Technology (FPT)*, pages 77–84. IEEE, 2016.

[6] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, 3(5):637–646, 2016.

[7] Yaman Umuroglu, Nicholas J Fraser, Giulio Gambardella, Michaela Blott, Philip Leong, Magnus Jahre, and Kees Vissers. Finn: A framework for fast, scalable binarized neural network inference. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 65–74. ACM, 2017.

[8] Xilinx Incorporation. Open Source HLx Examples. https://github.com/Xilinx/HLx_Examples/, 2016. [Online; accessed 19-July-2019].