Reconfigurable Computing: FPGAs

ECE5755 Term Paper

Vrushank Agrawal (vca4), Siddharth Kothari (sk2793), Haran Rajkumar (hr322)

November 2023

1 Introduction

An FPGA, or Field-Programmable Gate Array, is a type of semiconductor device that is widely used in electronic circuits. Unlike traditional integrated circuits like CPUs or GPUs which have a fixed function after manufacturing, FPGAs are known for their ability to be reprogrammed to perform a variety of tasks even after they are manufactured. This flexibility makes them uniquely powerful and versatile in the world of electronics and digital logic.

The invention of FPGAS is credited to Dr. Ross Freeman, who co-founded Xilinx, a leading semiconductor company. Freeman conceived the idea of the FPGA and filed the patent for it. The first FPGA was developed and brought to market by Xilinx in 1985. Prior to this, digital circuits were typically fixed in their function once fabricated. The advent of FPGAs opened up new possibilities in customizable and adaptable digital design, leading to their wide use across various industries today.

Field Programmable Gate Arrays (FPGAs) are integrated circuits that can be programmed and reprogrammed to perform specific functions Invented by Xilinx (AMD) in 1985 Programmed via Hardware Descriptive Languages (Verilog, Lucid).

1.1 FPGAs vs ASICS

An Application-Specific Integrated Circuit (ASIC) is a type of microchip tailored for a specific task (GPUs, TPUs). These chips are designed exclusively to perform a particular function or set of functions, making them highly efficient and fast at their intended tasks, but this also means they are not reprogrammable for other uses.

FPGAs stand out for their flexibility and reconfigurability. They can be reprogrammed for various tasks, allowing for updates and functionality changes without needing new hardware. This attribute makes them ideal for rapid prototyping and testing, which in turn reduces development time and cost. Additionally, their programmability leads to a shorter time-to-market compared to ASICs, and they are more cost-effective for small production runs. FPGAs also excel in applications requiring parallel processing capabilities, as they can perform simultaneous operations efficiently.

However, FPGAs do have some disadvantages. They generally offer lower performance than ASICs due to their programmable elements. Their use also involves increased complexity, requiring specialized knowledge in hardware description languages. Moreover, for large-scale production, FPGAs tend to have a higher unit cost, making them less economical for mass production compared to ASICs.

On the other hand, ASICs are known for their higher performance. They are highly optimized for specific tasks, offering superior performance in those areas. They are also cost-effective at scale, presenting a lower per-unit cost in large quantities. Another advantage of ASICs is their smaller size, which is particularly advantageous in space-constrained applications.

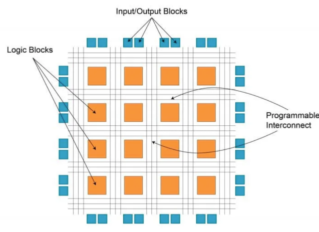
However, ASICs come with their own set of drawbacks. They involve a high initial development cost and time, with the design and fabrication process being both expensive and time-consuming. Furthermore, ASICs are inflexible; once fabricated, they cannot be altered, requiring a complete redesign for any updates or changes. This inflexibility, coupled with the higher initial investment, results in a higher risk factor in their development and deployment.

In summary, FPGAs offer flexibility and rapid deployment but with lower performance and higher unit costs at scale. ASICs provide high efficiency and performance with lower power consumption but require a higher initial investment and are inflexible post-production. The right choice depends on the specific requirements and constraints of the project. In general, FPGAs are best suited for tasks that are unique and rare enough that the cost of developing ASICs for that task does not.

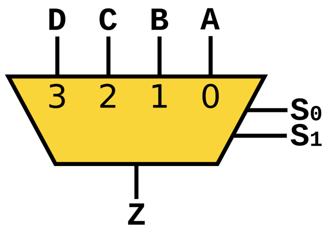
1.2 Architecture

The core architecture of an FPGA consists of three key components: Configurable Logic Blocks (CLBs), programmable interconnects, and Input/Output Blocks (IOBs) (Fig. 1a). CLBs contain the basic logic elements, typically including Look-Up Tables (LUTs) for implementing logic functions and flip-flops for memory or timing functions. These CLBs can be programmed to perform a wide range of logical operations. The programmable interconnects are a network of pathways that connect the CLBs, allowing for the creation of complex logic circuits by routing signals between different logic blocks. This interconnectivity is also programmable, giving the designer the flexibility to connect CLBs in virtually limitless ways to achieve the desired functionality. Finally, the IOBs provide the interface between the FPGA's internal logic and the external environment, allowing the FPGA to send and receive signals to and from other devices.

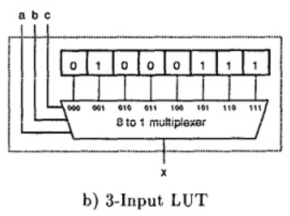
One of the main components powering the FPGA is the Look-Up Table. It is a piece of circuitry that allows it to mimic any truth table that a user can assign it to follow. To understand how a lookup-table works, first we must understand the multiplexer. A multiplexer (Figure 1b) has two sets of inputs, the main inputs and select inputs. Based on the select inputs, a subset of main inputs is allowed to pass through the multiplexer. It effectively acts as a switch that allows certain inputs to pass through. The truth table for the multiplexer in Fig. 1b is showcased in 1.



(a) Multiplexer board.



(b) Multiplexer board.



(c) Lookup table (LUT)

Figure 1: FPGA

To make a 3-input LUT, we take an 8 to 1 multiplexer (Fig. 1c) and direct the 3 inputs to the LUT to the select inputs of the multiplexer. We fix the inputs for the rest of

S1	S0	Z
0	0	A
0	1	B
1	0	C
1	1	D

Table 1: Truth table for multiplexer

a	b	c	x
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Table 2: Truth table for LUT

the multiplexer to imitate the truth table we would like to produce. Table 2 is the truth table that is replicated in the lut depicted in Figure 1c. These fixed inputs are defined by the FPGA developer and loaded onto the LUT's SRAM when the FPGA is booted up.

2 Application 1 - Data Centers

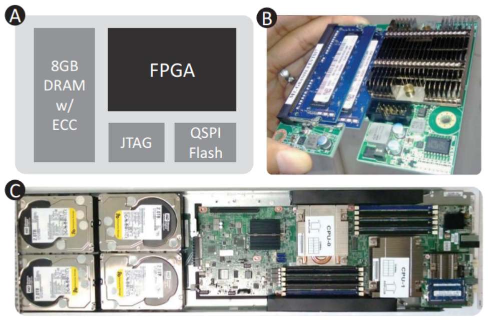
Data centres are critical to the modern digital infrastructure as they support a wide range of services, from web hosting and cloud computing to large-scale data processing and storage. The demands placed on data centres are immense and growing. They constantly need high computational capabilities to process and store enormous amounts of data and flexibility to manage workloads based on user demand and type of service. Moreover, data centres consume a significant amount of energy which has a substantial environmental impact while incurring high operational costs, and building data centres involves substantial investments which is a challenge for their economic viability.

A data server, at its core, is built around a central processing unit (CPU) that handles general computing tasks, supplemented by other key hardware components such as memory (RAM) for quick data access and storage devices like SSDs for data retention. Modern servers include several hardware accelerators to perform certain computing tasks more efficiently than the CPU. These hardware accelerators like Graphics Processing Units (GPUs) and Digital Signal Processors (DSPs) are a critical step in enhancing the server's performance for specific computationally intensive tasks that can be offloaded from the CPU. This integration significantly boosts the server's efficiency in handling those particular tasks and managing the workload of the data centre.

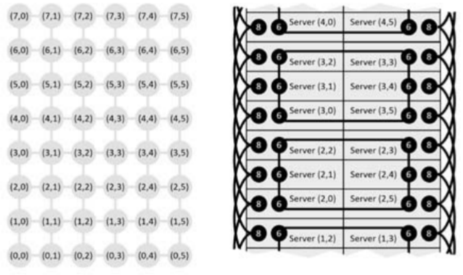
However, adding hardware accelerators has a notable architectural limitation: its static nature. Once a server is configured with specific hardware accelerators, its ability to adapt to changing needs is constrained. Unlike software solutions that can be updated or reconfigured with relative ease, hardware accelerators are fixed in their capabilities. As a result, if the computational demands or the nature of the tasks change significantly, the server might not be able to adapt efficiently without a hardware upgrade or replacement. This inherent rigidity in hardware-accelerated servers necessitates careful planning and forecasting of computational needs to ensure longevity, relevance, and economic viability in rapidly evolving technological landscapes. Essentially, data centres face a critical challenge in efficiently allocating resources where needed and adjusting to different types of computational tasks while being highly manageable or in other words homogeneous in their variations and varieties [3].

FPGAs offer a solution to the above problems as they have the potential to flexibly reconfigure the hardware architecture to support different workloads, but they come with challenges. FPGAs do not have the capacity to comfortably fit the accelerated functionality to support different workloads in the limited re-configurable area they provide, hence, several FPGAs (arranged in a mesh) are required to handle these specialized workloads. At the same time, adding too many FPGAs together is too costly and power-consuming which does not offset the gain from their utilization which varies with time and, hence, can be wasteful due to under-utilization.

Microsoft Catapult was one of the first projects to successfully utilize FPGAs in data servers to improve data servers' latency and throughput. As mentioned earlier, adding a mesh of FPGAs on individual servers is a monumental challenge due to their large power requirement which is not possible within the existing network and requires a complete overhaul of the architecture. In Catapult, Microsoft added a single high-end FPGA, the Altera Stratix V D5 which has more than 172,000 CLBs, in a daughter card on each server. This daughter card was connected to the host CPU through a PCIe instead of a tighter integration because of the limitations of the server design. 8GB of local DRAM was further added to accommodate certain services that needed to be run on the FPGAs. The manufactured FPGA board also contained a USB JTAG for debugging and testing and a 32MB QSPI flash drive to hold the FPGA configurations [5]. Figure 2a represents the FPGA board architecture used in Microsoft Catapult where image A is a block diagram of the FPGA board, image B is a real-life image of the sem board, and image C is a server image with the FPGA connected with the CPUs.



(a) FPGA board.



(b) Torus Network

Figure 2: FPGA architecture in Microsoft Catapult [5]

Once the FPGAs were coupled with the servers, these servers were inter-connected in groups of 48 servers called a pod. Each pod had a dedicated power distribution unit where the pod was organized in a 24U arrangement of 48 half-width 1U servers. The network contained two custom cable assemblies of eight and six shells each that were plugged into two SAS ports on each server giving the network a 6x8 Torus configuration as depicted in figure 2b. The route between any two interconnected SAS ports supported

10 Gb/s signalling rates or 20Gb/s peak bidirectional bandwidth at a sub-microsecond latency which is extremely efficient. Overall, this configuration avoids the necessity to reconfigure the power distribution of the data centre, it avoids a single point of failure which can potentially disable several servers, and it also makes the servers homogeneous and therefore highly manageable. Moreover, given the sufficiently low latency and high bandwidth within the inter-connected FPGA network, the workloads requiring more than one FPGA can be conveniently mapped across the inter-FPGA network while respecting the power, thermal, space, and operating cost limits [5].

The Catapult system was the first system to successfully leverage the potential of FPGAs in data centres by overcoming hurdles such as power requirements, heating obstacles, and acceleration function mappings. The system also solves a major chicken-and-egg problem where a 1:1 ratio is desired between data servers and applications but the question remains that which comes first, by providing reconfigurable architecture that can be used to accelerate different workloads as needed.

The Catapult system described here was the proof-of-concept which was implemented by Microsoft on a bed of 1,632 servers to deliver a 95% increase in the throughput of the ranking algorithm used by Bing search. In future iterations, the Catapult system was further enhanced by a network design called the *Configurable Cloud* which allowed the FPGA network to leverage the potential of FPGAs beyond their pod to access any other FPGA through the data centre. The network design significantly improved the system flexibility because the FPGAs could now be used locally as a compute or a network accelerator and globally as a large-scale pool of reconfigurable hardware resources [2].

Since Microsoft's Catapult system, many cloud service providers like AWS, Alibaba Cloud, and Huawei have started offering systems designers cloud-based workload acceleration as an alternative to investing in on-premises FPGA infrastructure. Essentially, cloud service providers are successfully harnessing the potential of FPGAs like low latency, high throughput, and energy efficiency to complement cloud applications. The main obstacle in the wide-scale adoption of FPGAs though, is the lack of a robust and easy-to-use software stack to deploy, manage, and scale them. Still, continuous work in the field focused on architecture enhancement, programming models, and security are seen as positive drivers for a broader range of applications for FPGA-based cloud deployment in the coming years [1].

3 Application 2 - Deep Learning

In recent years, significant advancements have transformed the landscape of neural network (NN) research, surpassing conventional algorithms across diverse domains. Various network models, including Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), have emerged, catering to image, video, and speech processing domains. Well-trained CNN models have notably elevated ImageNet's top 5 image classification accuracy from 73.8% to 84.7% [8], significantly enhancing object detection through superior feature extraction. RNNs have also set new benchmarks, minimizing word error rates in speech recognition. The remarkable adaptability of neural networks to tackle diverse pattern recognition challenges positions them as promising contenders for numerous artificial intelligence applications.

Despite their efficiency, neural network models grapple with considerable computational and storage complexities. Current research mainly emphasizes scaling up these models, exemplified by the largest CNN model demanding 39 billion floating-point operations (FLOP) and over 500 MB in model parameters for 224x224 image classification. However, as image resolutions increase, computational demands soar beyond 100 billion operations, underscoring the need for judicious computing platforms. While CPUs manage 10-100 GFLOP per second, their power efficiency hovers around 1 GOP/J, making it challenging to meet both high-performance cloud application needs and the power constraints of mobile apps. In contrast, GPUs excel with peak performances up to 10 TOP/s, serving as ideal choices for high-performance neural networking. Additionally, the gradual emergence of FPGAs as energy-efficient platforms for neural network processing is gaining traction. FPGAs harness high parallelism and streamline logic according to specific neural network calculations through tailored hardware designs, expanding their viability in this domain. Certain studies indicate that simplifying the neural network model to suit hardware doesn't impact its accuracy. Consequently, FPGAs can achieve greater energy efficiency compared to CPUs and GPUs [7].

Machine learning models can exploit massive levels of parallel computation through GPUs at a much more cost-efficient architecture than what an FPGA system would achieve with similar specifications. However, for fast serialized processes and non-computational tasks such as sorting, FPGAS are able to leverage hardware-level efficiency by morphing hardware to suit the task. This creates an interesting application for FPGAs to run real-time inference from pre-trained neural network models. In such a setup, the model can be efficiently trained on a GPU-based system where parallelism results in training speed-up, and deployed on an FPGA where a deep network (as compared to wide) can use streamlined hardware for making quicker inferences from the CNN in real-time.

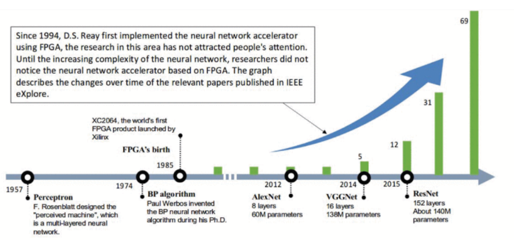


Figure 3: Development history of the neural network accelerator based on FPGA.[8]

Tools for pre-trained models on FPGAs: Typically, machine learning applications start with an initial design using a high-level Python-centric framework like Tensorflow. However, to make it work on hardware like FPGAs, there's a need to manually convert it to either C or RTL using specialized vendor tools. This conversion process demands time and specific skills, which restricts FPGAs' usage in this field. To address this challenge, researchers developed **Leflow** [4], an open-source tool-flow that translates numerical computation models from Tensorflow into hardware that can be synthesized. Leflow leverages Google's XLA compiler, which directly generates LLVM code from a Tensorflow specification, streamlining the process significantly.

In the flow depicted in Figure 3, users engage by initiating a design in Python utilizing the Tensorflow package. Tensorflow, a widely utilized framework, enables the swift specification of machine learning algorithms by employing computational graphs. Within the Tensorflow environment, users can assess the effectiveness of machine learning architecture and training algorithms.

The flow starts when a user desires FPGA hardware acceleration for a specific computational graph. To facilitate hardware generation, the process incorporates the Accelerated Linear Algebra compiler,

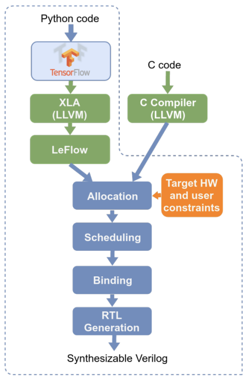
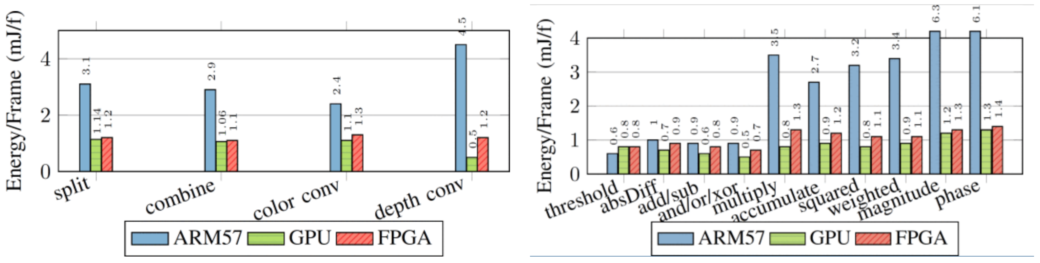


Figure 4: Leflow pipeline compared to standard HLS flow [4]

which produces an LLVM-compatible intermediate representation of the computational graph. This LLVM IR serves as input for a high-level synthesis tool responsible for allocation, scheduling, and binding, ultimately generating a hardware description in Verilog. LegUp is employed within this flow, although other High-Level Synthesis tools could also be utilized. Subsequently, the hardware description undergoes compilation by back-end FPGA compilers, like Quartus Prime, to facilitate the placement and routing of hardware onto an FPGA.

Despite the XLA compiler outputting LLVM IR and LegUp generating hardware from LLVM IR, several transformations are necessary within the IR to ensure a seamless interface between these two tools.

FPGA Energy efficiency comparison with CPUs and GPUs: Having established that one of the primary use cases for deploying a CNN on an FPGA over a GPU would be power efficiency, Xilinx Research Labs conducted a comprehensive benchmark on the run-time performance and energy efficiency of a diverse set of vision kernels [6]. The analysis encompasses the reasoning behind the inherent effectiveness or limitations of a particular underlying hardware architecture, drawing from the traits exhibited across various categories of vision kernels. Their study focuses on three frequently utilized HW accelerators for embedded vision applications: the ARM57 CPU, Jetson TX2 GPU, and ZCU102 FPGA. Each platform utilizes its respective vendor-optimized vision libraries, namely OpenCV, VisionWorks, and xfOpenCV.



(a) Input Processing Kernels

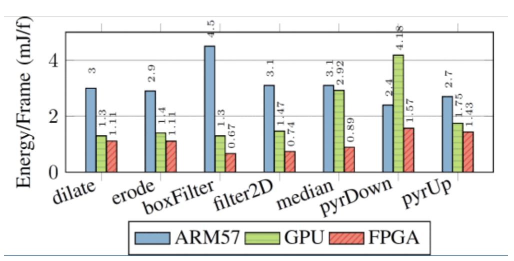
(b) Arithmetic Operations Kernels

Figure 5: Vision Kernel energy performance on CPU, GPU and FPGA[6]

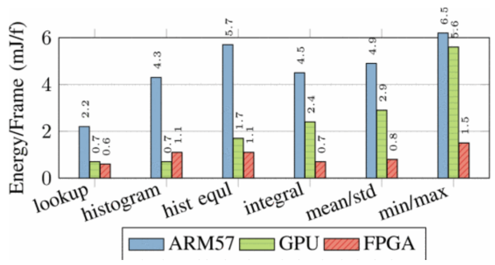
Figure (4)(a) displays the energy consumption per frame for input processing kernels. The GPU and FPGA outperformed the CPU due to their strong data parallelism, low complexity, and absence of data dependency. Compared to the CPU, the GPU and FPGA reduced energy consumption per frame by an average of 1.79× and 1.41× respectively. Notably, the GPU's bit-depth conversion achieved a 2.4× reduction compared to the FPGA, credited to its efficient use of streaming and CUDA textures in VisionWorks kernel implementation.

Figure (4)(b) depicts the performance of arithmetic and logic operations. It becomes evident that the CPU adeptly handles straightforward operations like thresholding, absolute differences, addition/subtraction, and bitwise operations. However, its efficiency diminishes notably when dealing with multiplication-based kernels like multiply, accumulate squared, weighted, magnitude, and phase. Contrarily, the GPU emerges as the most energy-efficient option compared to the CPU and FPGA. The GPU's implementations achieved an average energy reduction per frame of 4.6× and 7.2× in contrast to the CPU and FPGA, respectively. This outcome aligns with expectations, given these algorithms can be broken down into numerous parallel pieces executing identical operations (SIMT).

In Figure (5)(a), FPGA outperforms GPU and CPU in filtering operations. The



(a) Filters Operations Kernels



(b) Image Analysis Kernels

Figure 6: Vision Kernel energy performance on CPU, GPU and FPGA[6]

FPGA's implementation achieved an average energy reduction of 1.8× compared to the GPU and 7.4× compared to the CPU per frame. The memory access patterns and mathematical complexity of linear filters align well with the parallel processing capabilities of the GPU and FPGA. However, median filters, unlike linear ones, present challenges for efficient GPU implementation due to their unconventional method of sorting input elements. Morphological operations like dilate and erode involve intricate hit and miss functions, making them more challenging to implement efficiently compared to filtering functions, hence explaining the lower frame rate and higher energy consumption in VisionWorks's implementations of small (3×3) filter kernels shown in Figure (5)(a).

In Figure (5)(b), the performance of image analysis kernels is depicted. For kernels like lookup table, histogram, and histogram equalization, the FPGA's energy consumption per frame sees an average reduction of 1.2× compared to the GPU. However, in kernels involving intricate branching conditions and complex memory access patterns—such as integral image, mean/std, and min/max locations—the FPGA's implementation achieved a more substantial average reduction ratio of 3.5× compared to the GPU.

Disadvantages of using FPGAs in for deploying CNNs: Having emphasised the reconfigurability and energy efficiency of FPGAs, they also come with their own unique drawbacks that one must consider before applying FPGAs to a deep learning pipeline. Reconfiguration costs: The adaptability of FPGA platforms is a mixed bag. While it offers numerous advantages in speeding up computations, the time cost of reconfiguration cannot be overlooked. Reconfiguration generally occurs in two ways: static and dynamic. Static reconfiguration, known as compile-time reconfiguration, involves configuring the hardware before a task begins and locking it until the task is completed. On the other hand, dynamic reconfiguration, done during runtime, allows the hardware to adjust as needed. However, this method is prone to delays, leading to longer runtimes.

Complexity in Programming: Despite the longstanding proposal and considerable maturity of reconfigurable computing architecture, it hasn't gained widespread popularity. The primary reason lies in the programming aspect. While traditional CPU programming relies on high-level abstract languages within a mature system, reconfigurable computing demands hardware programming expertise. Generally utilizing hardware programming languages like Verilog or VHDL, this requirement significantly increases the learning curve and time investment for programmers.

All in all we've seen how FPGAs put forth a strong use case for deploying pre-trained neural networks for making inferences in real-time. While GPUs are able to achiveve massive levels of parallelism that speed up training models, FPGAs are able to provide a more energy efficient system to deploy them given the necessary adaptations to write HDL code for the same.

References

[1] Christophe Bobda et al. "The Future of FPGA Acceleration in Datacenters and the Cloud". In: *ACM Transactions on Reconfigurable Technology and Systems* 15 (Feb. 2022), pp. 1–42. DOI: 10.1145/3506713.

[2] Adrian M. Caulfield et al. "A cloud-scale acceleration architecture". In: *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 2016, pp. 1–13. DOI: 10.1109/MICRO.2016.7783710.

[3] Intel. *Acceleration in the Data Center - Intel® FPGA*. https://www.intel.com/content/www/us/en/data-center/products/programmable/overview.html. Online; accessed: 2023-11-19.

[4] Daniel H. Noronha, Bahar Salehpour, and Steven J. E. Wilton. *LeFlow: Enabling Flexible FPGA High-Level Synthesis of Tensorflow Deep Neural Networks*. 2018. arXiv: 1807.05317 [cs.LG].

[5] Andrew Putnam et al. "A Reconfigurable Fabric for Accelerating Large-Scale Datacenter Services". In: *IEEE Micro* 35.3 (2015), pp. 10–22. DOI: 10.1109/MM.2015.42.

[6] Murad Qasaimeh et al. "Comparing Energy Efficiency of CPU, GPU and FPGA Implementations for Vision Kernels". In: *2019 IEEE International Conference on Embedded Software and Systems (ICESS)*. 2019, pp. 1–8. DOI: 10.1109/ICESS.2019.8782524.

[7] Chao Wang et al. "Service-Oriented Architecture on FPGA-Based MPSoC". In: *IEEE Transactions on Parallel and Distributed Systems* 28.10 (2017), pp. 2993–3006. DOI: 10.1109/TPDS.2017.2701828.

[8] Teng Wang et al. "An Overview of FPGA Based Deep Learning Accelerators: Challenges and Opportunities". In: *2019 IEEE 21st International Conference on High Performance Computing and Communications; IEEE 17th International Conference on Smart City; IEEE 5th International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*. 2019, pp. 1674–1681. DOI: 10.1109/HPCC/SmartCity/DSS.2019.00229.