BinaryEye: A 20 kfps Streaming Camera System on FPGA with Real-Time On-Device Image Recognition Using Binary Neural Networks

Petar Jokic
System-on-Chip
CSEM SA
Zurich, Switzerland
petar.jokic@csem.ch

Stephane Emery
System-on-Chip
CSEM SA
Zurich, Switzerland
stephane.emery@csem.ch

Luca Benini
Integrated Systems Laboratory
ETH Zurich
Zurich, Switzerland
lbenini@iis.ee.ethz.ch

***Abstract*—Streaming high-speed cameras pose a major challenge to distributed cyber-physical and IoT systems, because large data volumes need to be transferred under stringent real-time constraints. Edge processing can mitigate the data deluge by extracting relevant information from image data on-device with low latency. This work presents an FPGA-based 20 kfps streaming camera system, which can classify regions of interest (ROI) within a frame with a binarized neural network (BNN) in real-time streaming mode, achieving massive data reduction. BNNs have the potential to enable energy-efficient image classifications for on-device processing. We demonstrate our system in a case study with a simple real-time BNN classifier achieving 19.28 us latency at 0.52 W power consumption and resulting in a 980x data reduction. We compare external image processing with this result, showing 3x energy savings, and discuss the used HDL/HLS design flow for BNN implementation.**

Keywords— Edge Processing, Binarized Neural Networks, Real-Time, FPGA, Image Recognition

I. Introduction

Communication-intensive applications like video surveillance and computer vision-driven automation are largely limited by the bandwidth of their communication interface. The emergence of internet of things (IoT) concepts, which aim at connecting large numbers of devices and allowing them to communicate with each other, further exacerbate the problem by increasing the number of data-streams. The reason for still transferring unprocessed data from peripheral devices to computers and servers mainly lies in the higher computational capacities of these cloud computing facilities. Some tasks though, especially in computer vision applications, require low latency and/or the data volumes cannot be sustained on standard communication links.

Edge processing [1] is an alternative for moving the data processing part of a system closer to the source or the edge of a network. This allows large data volumes to be processed, and relevant information to be extracted, before being transmitted over communication links. Additionally, this approach decreases latency, which is important for real-time tasks [2]. Looking at the use case of a streaming camera, edge processing means that sampled images get analyzed directly onboard the device. Beside image or video compression algorithms, also image classifiers based on neural networks (NN) can be used for this task [3]: The hundreds (or even thousands) of pixel values in each image can then be reduced to a handful of bytes, containing the image class.

As a motivating example, consider a 1 megapixel camera with 10-bit resolution streaming at 2 kfps. Such a camera continuously generates 26.2 Gbit/s of data. Neither a current Wi-Fi interface, achieving throughputs of up to 1.7 Gbit/s [4], nor a state-of-the art wired USB 3.1 communication link, with a claimed theoretical bandwidth of 10 Gbit/s, is capable of supporting such raw data-rates. With on-board 1000-class image recognition, the data stream could be reduced to 20 kbit/s (10 bit per image classification), allowing many cameras to be used simultaneously.

NNs, or more precisely convolutional neural networks (CNN) [5], are the current state-of-the-art systems for such image recognition tasks. But because of their computational complexity, NNs are most frequently processed on powerful servers in the cloud, requiring the images to be uploaded through a communication network. As previously shown, this is not feasible for high-speed cameras and therefore requires local image classification. On size and power-limited devices, mobile GPUs and specialized image processing chips can run image analysis at low frame-rates but do not sustain high-speed image classification [6]. The recently introduced BNNs though, significantly reduce the computational complexity of networks, reaching high frame-rates when implemented on FPGAs [6].

Increasingly larger and more complex neural network architectures are being proposed for accurate image classifiers, which sometimes even surpass human performance in some image recognition tasks [7]. Deep Neural Networks (DNNs) shifted the trend from shallow networks with very few layers to deeper ones with tens or up to hundreds of layers [8, 9, 10] which need to be stored on external DRAM due to their large size. Internal on-chip memory would be much faster and more power efficient but usually does not support the size of larger DNNs.

Thus, classifying images with a neural network is a computationally intensive task with a substantial memory footprint, challenging the used hardware architectures. High-performance CPUs, GPUs and FPGAs with fast DRAM memory are often used architectures for computing large size networks but consume vast amounts of power [11]. One of the main perfor-

978-1-5386-4155-2/18/$31.00

mance limitations in these systems is the extensive external DRAM memory access [12], often referred to as "memory bottle-neck". In many real-time applications, memory access dominates the power consumption [13], which is a big challenge for energy-constrained mobile systems.

To mitigate this problem, different approaches have been proposed. On one hand, application-specific hardware is being developed, providing more efficient processing elements [13]. These techniques work well for computation-limited network structures like convolutions but are largely limited by the available memory bandwidth for fully-connected layers [12]. On the other hand, quantized networks are tackling the problem by reducing the size of the network through parameter quantization [14]. Thus, parameters can be represented with lower precision numbers, reducing both the required storage space as well as the bit-widths of computational elements. The recently introduced BNNs exploit this advantage by limiting weights and activations to 1-bit values [14, 15]. These changes substantially lower the hardware requirements compared to standard 32-bit architectures: Memory needs are reduced by 32x and the dominating multiply-accumulate (MAC) operations can be simplified to logic XNOR operations with appended bit-counting [15]. Simpler computations increase the throughput as each operation consumes less time, while lower memory needs allow parameters to be stored on-chip, reducing external memory accesses. Thus, the low memory footprint directly reduces the problematic effects of memory bottle-necks. Summarized, BNNs provide higher computational efficiencies and thus enable low-power machine learning applications.

While CPUs and GPUs were designed to efficiently process 32-bit arithmetic operations, their instruction sets do not natively support low-precision arithmetic [6]. FPGAs and application-specific integrated circuits (ASIC) can handle such operations much better as they provide arbitrary bit-widths for arithmetic and logic operations (such as XNOR). Moreover, the reduced size of learned weights in BNNs allow meaningful networks to be fully stored on internal block RAM (BRAM), avoiding expensive external DRAM transfers [6], as it would be required in CPUs. BNNs implemented on FPGAs are therefore fast and power-efficient solutions for on-board processing. ASICs have similar advantages as FPGAs and can even be designed more efficiently but are in most cases not affordable due to their time-intensive and very costly manufacturing, while being less flexible than FPGAs. One of the main drawbacks though, is the time-intensive development of FPGA firmware in hardware description language (HDL). Therefore FPGA manufacturers introduced high-level synthesis (HLS) tools, which allow faster coding in higher level languages with automatic compilation into HDL code [16]. This step substantially reduces the development time and adds efficient throughput optimization capabilities to the design process.

In this paper we present a complete implementation of a real-time image recognition system coupled with a streaming 20 kfps high-speed camera integrated on an FPGA. To the best of our knowledge this is the first such system, capable of processing all data on-board at full frame rate for performing BNN-based image recognition. Our work demonstrates that even for multi-kfps cameras, a complete configurable engine for image acquisition, ROI extraction and BNN inference can be implemented on a single FPGA while achieving low power consumption and satisfying the high frame-rate requirements. The real-time performance enables edge processing, enormously reducing the communication data traffic and ensuring privacy as images do not leave the device. We present a full case study of the deployment of our engine on a real camera. Experiments demonstrate 980x data reduction, leading to 3x energy savings compared to streaming all data and thus showing the applicability of BNNs for edge processing in high-speed camera applications.

The rest of this paper is organized as follows. Section II discusses related work and in Section III we present the implemented system as well as our case study. Experimental results are presented in Section IV. Section V concludes the paper.

II. Related work

Edge processing was well described by Shi et al. [1], illustrating the vast amounts of data being produced by IoT nodes, making efficient on-board processing inevitable. Their overview of edge processing applications suggests performing video analytics close to the data source, as we propose in this work. Ananthanarayanan et al. [2] elaborate on large-scale video analytics in huge networks and conclude that real-time video processing might even be "the killer application" for edge processing. Soro et al. [3] provided an extensive survey of the challenges in such visual sensor networks in 2009, already naming resource constraints as the biggest problem for large camera networks. One of the discussed solutions is again on-board video processing. This proposal is supported by Mohan et al. [17], estimating the number of cameras in the world to reach 13 billion by 2030 and thus pushing the video traffic portion in relation to the overall internet traffic to more than 83% in 2020.

The problem of large data bandwidths gets worse for high frame rates and further increases if multiple high-speed cameras are to be used in a network, as for example Noda et al. describe in their vehicle-tracking application [18]. By extracting position features from sampled images and only transmitting this information, they manage to transmit 1 kfps video streams from multiple cameras through a simple Ethernet link. Stevanovic et al. showed that on-board image processing is additionally useful for low-latency applications, like image-based triggering for high-speed X-ray imaging [19]. Certain extracted image features were used to self-trigger video acquisition, making it possible to record activities which cannot be detected in any other way or would require too much time to be processed externally.

Traditional image compression techniques are based on an efficient representation of the contained information or the exploitation of limitations in the human visual perception. In many applications they are still an often used data-reduction technique [4]. Embedded image compression for high-speed image sensing was presented by Mosqueron et al. [20] but only reached compression ratios of 1:30, which is much smaller than what can be achieved with image classification. Nevertheless, traditional compression could be useful for pre-processing the data in order to keep the classifier as simple as possible.

For more complex real-time image recognition tasks, neural network-based inference engines have been presented. Cavigelli et al. [21] show real-time scene labelling based on CNNs, achieving state-of-the-art performance on an embedded GPU. In their more recent work [22] they additionally exploit frame-to-frame redundancies in static camera settings, which further improve their power efficiency. Also BNNs have been shown to perform real-time image recognition, as Mazare et al. [23] present in their object-orientation analysis system, based on a small BNN, implemented on an FPGA. Recent fast CNN-based object trackers, such as YOLOv2 [24], manage to reach a few tens of frames per second on embedded GPUs. All these approaches reach frame rates that are orders of magnitude too low for high-speed cameras, producing tens of kilo-frames per second.

To the best our knowledge, there are no published implementations of BNN-based image processing at the edge for high-speed cameras and we therefore compare our idea at the level of image classification using quantized neural networks. Implementing reduced precision neural networks for image recognition tasks though, has been proposed in many recent papers. This field of research mainly started to expand after Courbariaux et al. [14] successfully presented BNNs achieving near state-of-the-art accuracy in image recognition tasks. Since then, different approaches for improving quantized NNs have been published.

Nurvitadhi et al. evaluate the applicability of BNNs on different hardware architectures, namely FPGA, CPU, GPU and ASIC [6]. On FPGAs, BNNs can be implemented in on-chip RAM while other architectures require external DRAM access, consuming 172x more energy than the floating point multiplication of the fetched number [6]. For networks with many memory accesses, DRAM will dominate the power consumption and thus create a big disadvantage. Their results confirmed the superior performance on FPGA and ASIC implementations. Nevertheless, quantized NNs have also been optimized on mobile CPU architectures [25], improving the inference time compared to a full-precision implementation by more than 3x.

Many FPGA implementations have been presented [11, 15, 16, 26, 27, 28], showing highly energy-efficient performance reaching continuous computational efficiencies of 400 GOP/s/W and more, as shown by Umuroglu et al. [15]. Baskin et al. [28] present a BNN implementation which scales better for larger networks, allowing more complex systems to be implemented on FPGAs.

Other work focuses more on multi-bit quantization, like ternary NNs [29], which achieve slightly higher accuracies than BNNs at the cost of higher computational and memory costs. The trade-off between accuracy and network quantization has inspired Mishra et al. [30] to propose wider networks for compensating accuracy loss due to quantization. By operating in the region between the two extremes, namely BNNs and full-precision networks, they can balance accuracy versus computational cost.

III. Implementation

In this section, we discuss the proposed FPGA-based BinaryEye streaming camera with on-board image recognition. Instead of streaming all information through the communication interface, BinaryEye reduces the time and energy intensive data communications by introducing (on-device) edge processing. Classifying an image reduces the data volume from hundreds of bytes, representing each pixel value, to a single classification result. Our camera was originally designed as a streaming system, buffering sampled images in DRAM and transferring them through a USB interface. This version serves as reference for experimentally benchmarking our proposed on-board processing implementation. Due to the low computational requirements and the small memory footprint of BNNs, we were able to implement edge processing within the existing FPGA and without using power-intensive DRAM.

Fig. 1 shows the main building blocks of the system architecture. The data-flow starts at the image sensor, where pixels are streamed out as soon as an image gets triggered by the camera control block (through a USB command). A line buffer sorts the pixels and forwards complete lines. In the original streaming application, all data is directly transferred to the USB interface (indicated with a dotted line in Fig. 1). Because the USB interface might not be able to extract the image data at the speed they are being produced, a DRAM image buffer is necessary. BinaryEye instead extracts the relevant pixels from the line buffer and binarizes the reconstructed image as soon as all lines are available. Binarization of the image is necessary for binary-input classifiers and needs to be processed on-line. Thus, we implemented a thresholding function for quantization, which suffices for simple applications: If a pixel is darker than a certain threshold, it is labelled black, otherwise it is set to white, mapping it to the background. The threshold value continuously needs to be adapted to varying illumination levels, which was achieved by using the average pixel value as threshold. Fig. 2 shows examples of sampled images and their respective binarized representations. As soon as an image finishes the binarization step, the BNN classifier is triggered. Upon completing a classification, the result is buffered and transferred via USB. All camera-specific functions, like trigger-scheduling or image sensor configuration, are implemented inside the camera control block. The proposed design offers simple extension possibilities to extract multiple ROIs and process them at the same time by instantiating multiple parallel classifier-paths (if the FPGA resources allow it).

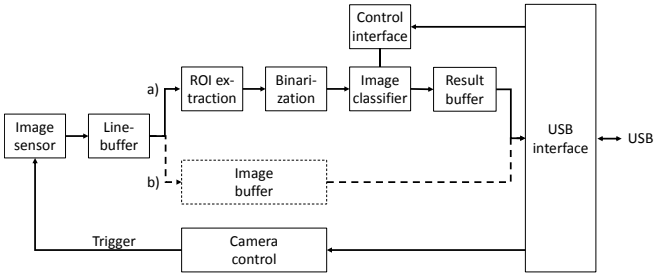

Fig. 1. System overview showing the data-flow. Path a) presents the implemented version with an on-board classifier. Path b) directly transfers images to USB.

To present the feasibility in a case study, we implemented and tested a handwritten digit classifier (based on the MNIST dataset), connected to the output-stream of the high-speed image sensor. Such a setup could for example be used in a mail sorting system, where handwritten digits on postcards or letters need to be analyzed at a very high speed. Recognized numbers can be combined to post codes which in turn tell the system where a certain card needs to be directed. Due to the required high throughput for handling thousands of letters per hour, the letters need to move through the system at the highest possible speed, making high-speed image processing inevitable. Especially the processing latency plays a crucial role as the output (e.g., the post code) is required for controlling the letter-distribution mechanism, which needs to keep up with the pace of the conveyor belt. Transmitting the same image data to a centralized classification engine would increase the latency and thus limit the maximum throughput. The implemented system therefore recognizes handwritten digits presented to the camera onboard the device and outputs the classification results. In our demonstration we used an ROI in the top right corner of the receptive field, measuring 32 x 32 pixels, which is the smallest camera-ROI possible for extracting 28 x 28 pixel images (as required for benchmarking with the MNIST dataset). Each raw 28 x 28 pixel image contains 784 pixels with a resolution of 10 bit each, summing up to 980 bytes of data. Classifying an image results in a single byte, containing the class information, which corresponds to a data compression ratio of 980:1. The 10 different classes in a handwritten digit classifier could even be encoded with 4 bits, doubling the compression ratio. One might claim that also the precision of the pixels could be reduced, but this would require the lighting conditions to be well matched with the exposure time of the camera. Without such a control circuit, the bit-width can only marginally be reduced.

A. High-Speed Camera System

The presented camera system integrates a custom-made 1 mega-pixel high-speed image sensor. It features a global shutter, allowing smear-free image acquisition of up to 20,000 grayscale frames per second (at a 32 x 32 pixel resolution). In order to transfer image data at such a high speed, the sensor is directly connected to FPGA peripherals making parallel data readouts of the 10 bit pixel values possible. An integrated ROI functionality enables the camera to process and output a limited region of the whole 1280 x 1024 pixel image. This reduces the data volume to be transferred, making faster frame rates possible and decreasing the peripheral power consumption. Fig. 3 shows the whole system, integrated in a device consisting of the image sensor, power supplies, a USB 2.0 communication interface and an FPGA, containing the firmware. A Xilinx Kintex-7 XC7K325[1] FPGA with 326,080 logic cells and 16 Mbit block RAM is implemented in this system.

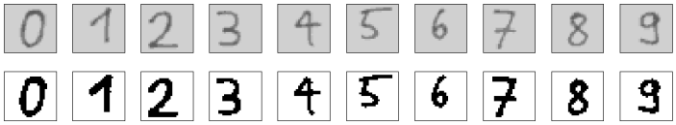

Fig. 2. Examples of 28x28 pixel images from the camera (top row) and after binarization (bottom row).

[1] https://www.xilinx.com/products/silicon-devices/fpga/kintex-7.html

Fig. 3. Image of the presented camera system with mounted objective.

B. BNN Image Classifier

As discussed in Section II, different BNN architectures have been proposed to implement image classifiers in FPGAs. Our work does not focus on the BNN classifier itself but its integration in an edge processing system. Thus, we chose to employ an existing hand-written digit classifier for demonstrating our edge processing system. Umuroglu et al. presented a framework for efficiently mapping BNNs on FPGAs (FINN) by converting the required computations into FPGA-friendly operations [15, 26]. They made an implementation of this framework publicly available[2], providing HLS code for reproducing a number of example classifiers on a Xilinx PYNQ board. Even though they targeted an ARM-based Zynq platform, we found their published results very promising for adapting it to a pure-FPGA implementation (the processor only interfaces the classifier). Thus, we based our case study implementation on their MNIST classifier "LFC max", which is a fully-binarized three-layer fully-connected (FC) network with 1024 neurons per layer. Its input is a binary 28 x 28 pixel image and it outputs a 10-bit result, each bit representing one of the ten possible digits. All network parameters are stored in on-board BRAM, avoiding any external memory access.

The provided FINN example is based on a Xilinx Vivado HLS project, which compiles the MNIST classifier, written in C++, into HDL code for the FPGA logic and software code for running the processor. We extracted the compiled HDL code, implementing the BNN, and designed the needed interfaces for embedding the classifier into the existing FPGA camera firmware. As depicted in Fig. 1, all network parameters can be loaded via USB through a control interface. This allows network parameter updates to be easily transferred without having to reprogram the device or even changing the HDL code.

For the final integration, all HDL components, including the compiled classifier, can be synthesized in Xilinx Vivado and loaded onto the FPGA. This design concept makes the classifier easily replaceable without having to write HDL code and thus takes the quickly evolving nature of (neural) network architectures into account. All that needs to be done for

[2] https://github.com/Xilinx/BNN-PYNQ

updating the network is compiling the new classifier, replacing the old one in the HDL project and synthesizing the new implementation, as depicted in Fig. 4. Which networks are possible to be implemented on the FPGA mainly depends on the available memory size and the number of logic cells on the FPGA. Our implementation only utilized around one quarter of the available BRAM resources as shown in Tab. I. According to the resource utilizations presented in the FINN paper, we could also fit their convolutional CIFAR-10 and SVHN classifiers. In fact, much larger BNNs could be implemented on the used FPGA, implementing more demanding tasks for other edge processing applications.

IV. Experiments and Results

To evaluate the proposed BinaryEye system on the example of the implemented MNIST case study, we compare its performance with other quantized MNIST implementations. Moreover, we present the power-advantages and the latency benefits of on-board image recognition for edge processing. As this work does not focus on the neural network itself, but mainly on its implementation for building a streaming edge processing node, the classification accuracy plays a secondary role. But we would like to emphasize that even though full-precision networks are often performing slightly better in the benchmarks than their binarized versions, BNNs have been shown to achieve comparable results at much lower computational costs. This is not limited to simple datasets like MNIST, but also applies to more complex networks as for example CIFAR-10 object recognition [14, 15].

TABLE I. Resource utilization of the FPGA

Implementation	Original camera	BinaryEye
Utilization: LUT	24% (48k)	43% (88k)
Utilization: FF	20% (81k)	28% (115k)
Utilization: BRAM	3% (14)	28% (124)

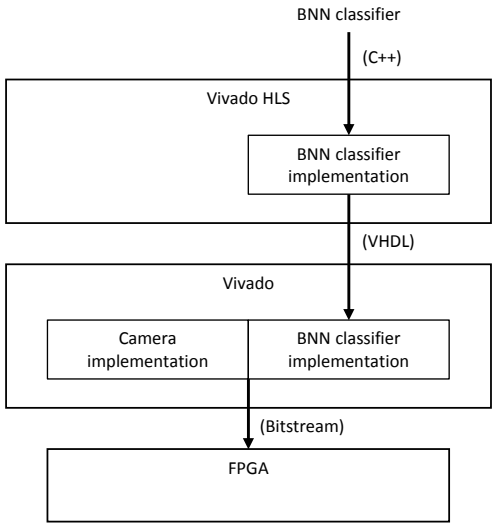

Fig. 4. Design-flow for implementing a high-level-coded BNN (C++) on the camera FPGA using HLS. Updating the BNN only requires the design to be re-compiled in Vivado HLS and re-synthesized in Vivado.

Tab. III summarizes the performance of state-of-the-art MNIST BNN classifiers [11, 15]. For this benchmark, the accuracy measured in this work is less than 1.5% below the highest reported classification rate of 99.79% (using a full-precision network) [31]. Because our classifier network is an exact reproduction of the FINN implementation by Umuroglu et al. [15], the accuracy of our system is identical to their benchmark performance. But unlike their purely computational implementation of the BNN, our system receives real-world data from an image sensor. Therefore BinaryEye is limited by the camera speed of 20 kfps, while their fastest classifier-implementation reaches 1,561 kfps at 2.44 us latency. Apart from the lower input frame rate due to the camera, we also clocked the classifier at a lower speed, namely 100 MHz, which is sufficient and more power-efficient for 20 kfps classifications. They clocked the classifier at 200 MHz and thus consumed more than 16x more power. Additionally, we did not use any of the speed optimizations available for this classifier, such as pipelining or adaptions of the network folding parameters to available FPGA resources, which further increased our inference time to 18.49 us. While such optimizations would improve the inference speed due to a higher degree of parallelization, they would also increase the power consumption, which we try to avoid. Our inference time would allow 2 ROIs to be classified within the period of a frame, while 78 ROIs per frame would be possible by optimizing the implementation to the published 1,561 kfps version. Based on the current resource utilization, we expect this version to also be feasible on our FPGA, but at the cost of the mentioned power disadvantages. The unutilized FPGA resources could also be used to instantiate multiple classifiers in parallel, each one for different ROI selections. This would theoretically multiply the image throughput by the number of parallel classification-lines. We expect 3 parallel classification lines to fit inside our FPGA, making a total of 6 ROI classifications possible at 20 kfps.

With a 4x larger network than ours, Liang et al. [11] show a comparable accuracy on the MNIST benchmark but consume considerably more power. For this small network, their computational performance is still lower than what Umuroglu et al. [15] report, but achieves almost 4x higher values on the CIFAR-10 benchmark, making it promising for larger networks. Alemdar et al. [32] achieve similar performance by using a ternary FC network, reaching only slightly lower accuracies but more than 20x higher frame rates. By further reducing the classification accuracy they even achieve a lower power consumption than we present in this work. Compared to our reference classifier from the FINN paper [15], their maximum frame rates are almost 6x lower, making FINN more promising for possible frame rate improvements in the camera.

In terms of power consumption, our 20 kfps implementation consumes slightly less than Umuroglu et al. [15] measured on their 12.2 kfps classifier. Even though our frame rate is higher, the powerful ARM processors on their Zynq FPGA causes some extra consumption. We avoid this by implementing the complete system on FPGA without any processor. In order to quantify the energy savings of edge processing, we compare the power consumption of BinaryEye with the consumption of a pure streaming implementation, which directly transmits all images without any pre-processing (the original

camera implementation). Therefore we synthesized the FPGA firmware once with the embedded classifier and once without it (see dotted path in Fig. 1). By evaluating both implementations on the same camera hardware for a task of recording a limited number of 10,000 images at 20 kfps, the energy savings could be determined. The power consumption was measured during the image acquisition and classification phase as well as during the data-transfer period. Tab. II summarizes the results and shows that edge processing reduces the energy consumption for the given task by a factor of more than 3x. This can be explained by the 980x reduction of the data volume and the associated data transfer time. Although the additional classifier functionalities on BinaryEye are slightly increasing the power consumption compared to the original system, the savings in transmission time still reduce the total energy consumption. The reduction ratio does not depend on the number of recorded images as the energy consumption of both implementations linearly depend on the number of images, making it a constant.

Apart from this improvement, it needs to be noted that the system without classifier could not support continuous image streaming at this frame-rate. The image transfer time (1.24 s) is longer than the acquisition time (0.5 s), causing the image buffer to overflow quickly. A faster data-interface (e.g., USB 3.1) would reduce the transfer time but still consume more power and possibly limit the maximum frame-rate. Additionally, the pure streaming implementation outsources the processing power to an external device, which could be included in the calculation to be further increasing the power consumption.

While the reduced power consumption of the edge processing system might not be a big advantage in stationary implementations, its low latency is of paramount importance for real time applications. The short latency of BinaryEye allows the system to react to an input image within 19.28 us. If the image gets processed externally, the latency becomes the sum of the transfer time to the external processing unit, the actual processing time and the communication time back to the device. From the time measurements in the original camera implementation, it can be estimated that transferring an image to a computer amounts to 124 us. The same time will be required to send it back to the camera. If, for example, the detection of a certain object has to trigger a video acquisition sequence, at least 248 us latency need to be taken into account. This makes BinaryEye at least 12x faster (in terms of latency) than the camera with external image processing, even without taking the external processing time into account. In applications like mail sorting, the reduced latency would allow letters to be processed quicker as the sorting mechanism would get its directing command faster, increasing the machine throughput.

V. Conclusion

Binarized neural networks for image recognition offer significant computational benefits and memory savings compared to full-precision implementations of the same network. This work presented a high-speed streaming camera with real-time on-board classification for edge processing, demonstrating these advantages on a real system. On-board processing of the image, as opposed to external (cloud) computing, enabled the camera to reduce the transmission data and thus achieved high frame rates in streaming mode without congesting its communication interface. The pure FPGA implementation made it possible to efficiently exploit the simplified computations as well as the low memory footprint of BNNs, allowing the whole network to be stored in power-efficient on-chip BRAM.

Our system achieved real-time 20 kfps image recognition with 19.28 us latency. The implemented case study of a handwritten digit classifier demonstrated near-state-of-the-art accuracy at 0.52 W power consumption. Classifying images on-board the camera reduced the data volume to be transmitted by 980x, enabling power saving of 3x compared to full-data streaming. Freeing capacity on the communication channel additionally enabled multiple other cameras to be attached to the same network, making applications with interacting high-speed cameras possible, even without complex specialized interfaces. These results indicated that BNNs could help edge processing to be implemented in other FPGA-based systems. The fact that more complex classifiers using BNNs have been shown to achieve comparable accuracies as their full-precision counterparts (e.g., CIFAR-10 classifier [14, 15]), makes BNN-based classifiers a promising solution for many other applications that go beyond the presented case study.

The configurability with high-level programming tools like Vivado HLS simplifies the deployment of FPGAs and their adaption to more complex applications. Combined with the benefits arising from the efficient handling of BNNs, FPGAs can offer simple implementations of edge processing systems.

TABLE II. Comparison between pure image-streaming and the presented on-board image classification

Measurement (10k images)	**Original**	**BinaryEye**
Image acquisition: average power [W]	13.26	13.78
Image acquisition: time [s]	0.5	0.5
Data transfer: power [W]	11.96	12.22
Data transfer: time [s]	1.24	0.00024
Data volume to be transferred [kB]	9,800	10
Total energy (acquisition + transfer) [Ws]	21.46	6.89
Relative energy consumption	100 %	32 %

TABLE III. Comparison to prior work

Implementation	**Network**	**Platform**	**Net size [Mbit/MOP]**	**MNIST Benchmark**			**Power [W]**		**Perf. [GOPS/s]**
				Accuracy	***Inference [us]/fps***	***Latency [us]***	***Chip***	***Wall***	
This work	FC binary	XC7K325T	2.9/ 5.8	98.40%	18.49 / 10k	19.28	0.52	12.7	58
This work	FC binary	XC7K325T	2.9/ 5.8	98.40%	18.49 / 20k	19.28	0.52	13.8	116
Umuroglu et al. [15]	FC binary	ZC706	2.9/ 5.8	98.40%	- / 1561k	2.44	8.8	22.6	9085.67
Umuroglu et al. [15]	FC binary	ZC706	2.9/ 5.8	98.40%	- / 12.2k	282	0.8	7.9	70.76
Alemdar et al. [32]	FC ternary	XC7K160T	-	97.76%	- / 255’102	5.37	0.32	-	-
Alemdar et al. [32]	FC ternary	XC7K160T	-	98.33%	- / 255’102	15.6	2.76	-	-
Liang et al. [11]	FC binary	5SGSD8	10.1/20.2	98.24%	3.39 / -	-	-	26.2	5905.40

VI. REFERENCES

[1] W. Shi, J. Cao, Q. Zhang, Y. Li and L. Xu, "Edge computing: vision and challenges," *IEEE Internet of Things Journal,* vol. 3, pp. 637-646, Oct 2016.

[2] G. Ananthanarayanan, P. Bahl, P. Bodík, K. Chintalapudi, M. Philipose, L. Ravindranath and S. Sinha, "Real-time video analytics: The killer app for edge computing," *Computer,* vol. 50, pp. 58-67, 2017.

[3] S. Soro and W. R. Heinzelman, "A survey of visual sensor networks.," *Adv. in MM,* vol. 2009, pp. 640386:1-640386:21, 2009.

[4] G. Kokkonis, K. E. Psannis, M. Roumeliotis and D. Schonfeld, "Real-time wireless multisensory smart surveillance with 3D-HEVC streams for internet-of-things (IoT)," *The Journal of Supercomputing,* vol. 73, pp. 1044-1062, Mar 2017.

[5] Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE,* vol. 86, pp. 2278-2324, Nov 1998.

[6] E. Nurvitadhi, D. Sheffield, J. Sim, A. Mishra, G. Venkatesh and D. Marr, "Accelerating binarized neural networks: comparison of FPGA, CPU, GPU, and ASIC," in *2016 International Conference on Field-Programmable Technology (FPT)*, 2016.

[7] K. He, X. Zhang, S. Ren and J. Sun, "Delving deep into rectifiers: surpassing human-level performance on ImageNet classification," in *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, Washington, 2015.

[8] A. Krizhevsky, I. Sutskever and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, USA, 2012.

[9] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke and A. Rabinovich, "Going deeper with convolutions," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[10] K. He, X. Zhang, S. Ren and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[11] S. Liang, S. Yin, L. Liu, W. Luk and S. Wei, "FP-BNN: binarized neural network on FPGA," *Neurocomputing,* 2017.

[12] J. Qiu, J. Wang, S. Yao, K. Guo, B. Li, E. Zhou, J. Yu, T. Tang, N. Xu, S. Song, Y. Wang and H. Yang, "Going deeper with embedded FPGA platform for convolutional neural network," in *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, New York, NY, USA, 2016.

[13] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz and W. J. Dally, "EIE: efficient inference engine on compressed deep neural network," in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, 2016.

[14] M. Courbariaux, Y. Bengio and J.-P. David, "BinaryConnect: training deep neural networks with binary weights during propagations," in *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, Cambridge, 2015.

[15] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre and K. 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*, New York, NY, USA, 2017.

[16] R. Zhao, W. Song, W. Zhang, T. Xing, J.-H. Lin, M. Srivastava, R. Gupta and Z. Zhang, "Accelerating binarized convolutional neural networks with software-programmable FPGAs," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, New York, NY, USA, 2017.

[17] A. Mohan, K. Gauen, Y. H. Lu, W. W. Li and X. Chen, "Internet of video things in 2030: a world with many cameras," in *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2017.

[18] A. Noda, M. Hirano, Y. Yamakawa and M. Ishikawa, "A networked high-speed vision system for vehicle tracking," in *2014 IEEE Sensors Applications Symposium (SAS)*, 2014.

[19] U. Stevanovic, M. Caselle, S. Chilingaryan, A. Herth, A. Kopmann, M. Vogelgesang, M. Balzer and M. Weber, "High-speed camera with embedded FPGA processing," in *Proceedings of the 2012 Conference on Design and Architectures for Signal and Image Processing*, 2012.

[20] R. Mosqueron, J. Dubois and M. Paindavoine, "Embedded image processing/compression for high-speed CMOS sensor," in *2006 14th European Signal Processing Conference*, 2006.

[21] L. Cavigelli, M. Magno and L. Benini, "Accelerating real-time embedded scene labeling with convolutional networks," in *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2015.

[22] L. Cavigelli, P. Degen and L. Benini, "CBinfer: change-based inference for convolutional neural networks on video data," *CoRR,* vol. abs/1704.04313, 2017.

[23] A. Mazare, L. M. Ionescu, A. I. Lita and G. Serban, "Real time system for image acquisition and pattern recognition using Boolean neural network," in *2015 38th International Spring Seminar on Electronics Technology (ISSE)*, 2015.

[24] J. Redmon and A. Farhadi, "YOLO9000: better, faster, stronger," *CoRR,* vol. abs/1612.08242, 2016.

[25] Y. Umuroglu and M. Jahre, "Streamlined deployment for quantized neural networks," *CoRR,* vol. abs/1709.04060, 2017.

[26] N. J. Fraser, Y. Umuroglu, G. Gambardella, M. Blott, P. Leong, M. Jahre and K. Vissers, "Scaling binarized neural networks on reconfigurable logic," in *Proceedings of the 8th Workshop and 6th Workshop on Parallel Programming and Run-Time Management Techniques for Many-core Architectures and Design Tools and Architectures for Multicore Embedded Computing Platforms*, New York, NY, USA, 2017.

[27] D. J. M. Moss, E. Nurvitadhi, J. Sim, A. Mishra, D. Marr, S. Subhaschandra and P. H. W. Leong, "High performance binary neural networks on the Xeon+ FPGA platform," in *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*, 2017.

[28] C. Baskin, N. Liss, A. Mendelson and E. Zheltonozhskii, "Streaming architecture for large-scale quantized neural networks on an FPGA-based dataflow platform," *CoRR,* vol. abs/1708.00052, 2017.

[29] A. Prost-Boucle, A. Bourge, F. Pétrot, H. Alemdar, N. Caldwell and V. Leroy, "Scalable high-performance architecture for convolutional ternary neural networks on FPGA," in *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*, 2017.

[30] A. K. Mishra, E. Nurvitadhi, J. J. Cook and D. Marr, "WRPN: wide reduced-precision networks," *CoRR,* vol. abs/1709.01134, 2017.

[31] L. Wan, M. Zeiler, S. Zhang, Y. LeCun and R. Fergus, "Regularization of neural networks using Dropconnect," in *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, Atlanta, 2013.

[32] H. Alemdar, V. Leroy, A. Prost-Boucle and F. Pétrot, "Ternary neural networks for resource-efficient AI applications," in *2017 International Joint Conference on Neural Networks (IJCNN)*, 2017.