Leveraging HW approximation for exploiting performance-energy trade-offs within the edge-cloud computing continuum

Argyris Kokkinis, Aggelos Ferikoglou, Dimitrios Danopoulos, Dimosthenis Masouros, and Kostas Siozios

Aristotle University of Thessaloniki, Thessaloniki, Greece
{akokkino, aferikog, ddanopou, dmasoura, ksiop}@physics.auth.gr

Abstract. Today, the need for real-time analytics and faster decision making mechanisms has led to the adoption of hardware accelerators, such as GPUs and FPGAs, within the edge-cloud computing continuum. Moreover, the need for energy-, yet performance-efficient solutions both in the edge and cloud has led to the rise of approximate computing as a promising paradigm, where "acceptable errors" are introduced to error-tolerant applications, thus, providing significant power-saving gains. In this work, we leverage approximate computing for exploiting performance-energy trade-offs of FPGA accelerated kernels with faster design time though an extended source-to-source HLS compiler based on Xilinx Vitis framework. We introduce a novel programming interface that operates at a high level of abstraction, thus, enabling automatic optimizations to the existing HLS design flow supporting both embedded and cloud devices through a common API. We evaluate our approach over three different application from DSP and machine learning domains and show that a decrease of 27% and 28% in power consumption, 61% and 69% in DSP utilization and 7% in clock period is achieved for Alveo U200 and ZCU104 FPGA platforms, on average.

Keywords: Edge & Cloud computing · FPGA · HW approximation · High Level Synthesis · source-to-source compiler

1 Introduction

Nowadays, the explosive growth and increasing power of IoT devices along with the rise of 5G networks have resulted in unprecedented volumes of data. Emerging use cases around autonomous vehicles, smart-cities, and smart factories require data processing and decision making closer to the point of data generation due to mission-critical, low-latency and near-real time requirements of such deployments. To this end, multi-layered computing architectures are emerging, where computing resources and applications are distributed from the edge of the network, closer to where data are gathered, to the cloud, realizing the edge-cloud computing continuum [15].

Even though edge-cloud architectures expand the computing capacity of the traditional cloud paradigm by introducing an additional huge pool of computing resources, the inefficiency of traditional CPUs to provide fast, near real-time executions has also led to the introduction of hardware accelerators, such as GPUs and FPGAs, to the aforementioned hierarchy. Typical examples of accelerators include power-efficient devices at the edge (e.g., NVIDIA Jetson and Xilinx MPSoC), to high-performance, massively-parallel devices at the cloud (e.g., NVIDIA Ampere and Xilinx Alveo). While hardware accelerators provide increased performance gains, these benefits do not come for free, as such devices typically require more power to operate. From the edge point of view, energy efficiency has always been a first class system-design concern, to provide low-power embedded systems design. On the cloud side, the considerable share of electricity expenses over the total cost of ownership (TCO) [2], as well as the shift towards energy-efficient (green) computing at the data-center level, also indicate the need for efficient hardware acceleration.

Towards building more energy proportional computing systems, approximation techniques have been identified as a promising solution to reduce energy consumption and increase performance while retaining the output quality of applications. *Approximate computing* is based on the observation that many applications feature intristic error-resilience properties [3] (i.e. in DSP or machine learning domain). On top of that, automatic compilation flows have been leveraged to apply hardware transformations in FPGA designs in order to achieve faster and more optimal results.

In this paper, we employ approximate computing for exploring performance-energy trade-offs of FPGA accelerated kernels. Specifically, the novel contributions of this paper are as follows:

- We develop an extension for Vitis framework based on word-length optimizations that takes into account resource and accuracy constraints.
- We build an end-to-end compilation flow by creating a novel source-to-source HLS compiler that can achieve significant improvements with resource-power-accuracy trade-offs.
- We provide support for both embedded and cloud FPGAs in the framework by utilizing a standardized software interface that enables seamless interoperability between devices. We evaluate our proposed framework on three popular algorithms from DSP and ML domains for both a cloud (Xilinx Alveo U200) and an embedded (MPSoC ZCU104) FPGA device.

2 Background & Related Work

2.1 HLS and RTL approaches for FPGA design

FPGA devices have been proven to perform really well when programmed with optimal configuration [4]. However, hardware can be designed at varying levels of abstraction with the commonly used levels of abstraction being the gate level, register-transfer level (RTL) and algorithmic level. These methodologies differ

with one another; RTL or many times described as hardware description language (HDL) (i.e. VHDL, Verilog) is used for circuit-level programming while behavioral approaches such as High Level Synthesis (HLS) are used at the algorithmic level [7, 9].

RTL design methodology: With HDL like Verilog and VHDL developers can create low-level representations of an AI model or function, from which ultimately actual wiring can be derived. Unlike in software compiler design, RTL takes as input the register transfer level representation and involves constructs such as cells, functions, and multi-bit registers. In order to accelerate an AI algorithm on hardware several Processing Elements (PEs) that perform multiply–accumulate operations are hand-written in Verilog or VHDL code that are designed and optimized based on the constraints of the target platform (i.e. the number of multipliers, etc). Mapping an AI model's computational operations to matrix-matrix or matrix-vector multiplication modules has been widely applied in prior studies [7, 8, 13]. Also, usually tiling and ping-pong double buffers techniques are then employed to improve the throughput along with other optimizations on the RTL level.

HLS design methodology: Employing High Level Synthesis for FPGA design enables a fast development process and high flexibility. Although using HLS provides a software-like tool flow (i.e. Xilinx Vitis or Intel Quartus Prime), the developer must still learn hardware-centric concepts, such as pipelining and routing, that they may have not been exposed to in writing C-code for traditional processors. Also, studies have shown that HLS can simulate faster the effects of data type precision and other hardware approximations which are often utilized on AI applications [14, 16]. HLS automatically generates an RTL testbench which is driven by vectors generated by the original C++ code. Last, previous work has shown that HLS designs can exploit significant parallelism compared with RTL approaches and achieve similar latency improvements [5, 11].

2.2 Approximation Techniques

Approximate computing techniques are used widely in error-resilient applications, trading-off algorithmic performance with power consumption and resource utilization. The rising computational complexity of many machine learning (ML) and math tasks makes imperative the use of techniques that mitigate the power and resource expenses while keeping the application's performance in the desired levels. Reducing the operands bit widths and exploiting inexact hardware are two popular techniques that are used in approximate computing in a wide range of different applications.

Precision Scaling: A technique that changes the bit-width of input or intermediate operands to reduce storage and computing requirements. Jong Hwan Ko et al. [10] leverage the benefits of precision scaling in neural networks for audio processing achieving up to 30x processing time speedup while the performance impact degradation is less than 3.14% in the case of classification tasks. Xilinx [1] provides tools and libraries that support a wide range of fixed point

precision data types. It has been noted, that FPGA design implementations using fixed point arithmetics are more efficient than their equivalent floating point due to their limited power and resource consumption. According to [6] power savings of up to 50% have been noticed for designs that have been migrated from floating to fixed point.

Approximate Multipliers/Adders: Hardware oriented approximate techniques have been proposed. Approximate adders and multipliers modify the typical addition and multiplication process for error resilient applications. The majority of the approximate integer and floating point multiplication approaches leverage logarithmic properties to minimize the computational overhead for a given error tolerance. Saadat et al. [12] proposed a custom floating-point multiplier that achieves x57 and x28 power and area improvement respectively when used as the multiplication block in AlexNet, with no significant degradation in the accuracy.

3 Proposed Framework Extension

In this section we describe the programming interface for our HLS compiler. The motivation behind this tool is to study how our HW optimizations can benefit from automation. We extend the Vitis HLS compiler with a word-length optimization tool packaged as an extension that can automatically optimize HLS kernels by applying source-to-source transformations and can generalize to different kinds of applications. Also, three use cases were selected for demonstration which are also described in this section.

3.1 Common FPGA Interface

We built our applications on top of Xilinx Runtime library (XRT) which is an open-source software stack that facilitates management and usage of FPGA devices. The interface supports either like C/C++ or Python on host code which enabled us to generalize more the framework using a unified API. XRT was implemented as a combination of userspace and kernel driver components that provided standardized software interface in our source-to-source HLS compiler enabling seamless device interoperability.

3.2 Building a source-to-source HLS compiler

Automatic source-to-source transformation techniques have been applied in software compilation and optimization but are also an ideal solution for HLS transformations. They can greatly benefit the FPGA accelerator design in a high-level synthesis design flow because there are many temporal/spatial resource and optimization directives that HLS tools provide. The goal of our source-to-source compiler was to enable the fully automated FPGA design flows which is especially important for deploying FPGAs seamlessly both in edge and cloud domains. Figure 1 shows the whole development process and the HLS compiler

utilizing our extension for word-optimization along with the XRT interface to support edge and cloud devices such as MPSoC or Alveo FPGAs.

High-Level Synthesis under Accuracy Constraint HLS vendors have enriched the HLS-C language with integer and fixed-point types of arbitrary size. The constraint function of the optimization problem corresponded to the fixed-point numerical accuracy. The case study in this work on Vitis tool is a transformation that applies to floating-point multiplications and additions on a loop's critical path. The number of bits for each part becomes fixed, I_w is the number of bits for the integer part integrating the sign bit, D_w is the number of bits for the decimal part, while $F_w = I_w + D_w$ is the total number of bits. The quantization mode Q_m which dictates the behavior when greater precision is generated than can be defined by smallest fractional bit in the variable used to store the result was set to round to plus infinity. Last, in the hardware context, we also configured the number scaling from float to fixed point types to tailor the accuracy requirements of the application.

High-Level Synthesis under Resource Constraint In the resource optimization scheme, the word-lengths are first optimized and then the architecture is synthesized. The first step gave a fixed-point specification that respects the accuracy constraint. For this constraint, a dedicated resource is used for each operation which can be defined by the HLS compiler. Also, when there are enough resources and high performance is required larger word lengths can be used where the heuristic limits the search space and obtain reasonable optimization time.

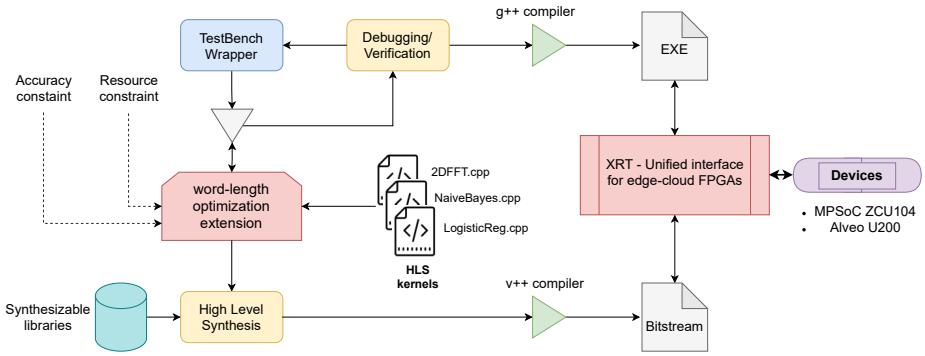


Fig. 1: HLS compiler extension

3.3 Kernels and Algorithms tested

In this study, we intentionally focused on evaluating the performance of popular algorithms as they are used as backend kernels to many frameworks in machine learning and math tasks. For this reason, we used for evaluation three kernels: Logistic Regression, 2D FFT and Gaussian Naive Bayes. All kernels were initially

optimized according to the FPGA principles for high performance using HLS optimizations such as loop unrolling, pipelining and memory partition.

Case Study 1: Two-dimensional FFT A fast Fourier transform (FFT) is an algorithm that computes the discrete Fourier transform (DFT) of a sequence. For our case study we selected a slightly different version of the classical algorithm namely 2D-FFT (two-dimensional Fast Fourier Transform) which can be used to analyze the frequency spectrum of 2D signal (matrix) data based on the primitives of 1D-FFT. First it computes the one-dimensional FFT along one dimension (row or column) then it computes the FFT of the output of the first step along the other dimension (column or row).

Case Study 2: Logistic Regression Logistic regression is one of the most popular methods for building predictive models for many complex pattern-matching and classification problems. It can label a sample with integer from 0 to $L-1$ in which L is class number. For a L class label, it needs $L-1$ vector to calculate $L-1$ margins. Its prediction function's output is linear to samples. The equation can be seen as below:

$$margin_i = \beta_{i,0} + \beta_{i,1}x_1 + \beta_{i,2}x_2 + ... + \beta_{i,n}x_n$$

Then label is decided according to L - 1 margins based on formula below:

$$Label = \begin{cases} 0, & \text{if maxMargin } \leqslant 0 \\ k, & \text{if } margin_k = \text{maxMargin} > 0 \end{cases}$$

Case Study 3: Gaussian Naive Bayes In machine learning, naive Bayes classifiers are a family of simple "probabilistic classifiers" based on applying Bayes' theorem with strong (naive) independence assumptions between the features. We chose to implement the extension of Naive Bayes which is the Gaussian Naive Bayes which supports real-valued attributes, most commonly by assuming a Gaussian distribution. The likelihood of the features is as below:

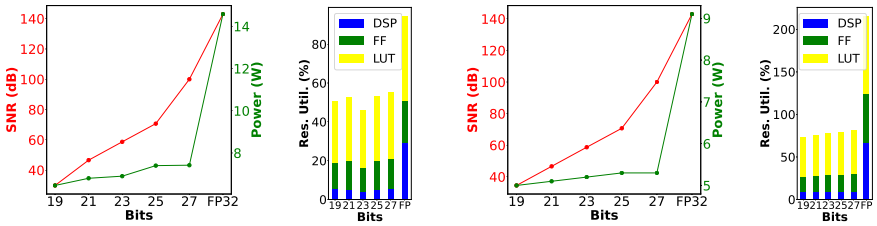
$$P(x_i \mid y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right)$$

4 Evaluation

In each of the case studies we evaluate our word-length optimization extension using the proposed compilation flow. The trade offs of the approximation techniques are evident in terms of accuracy, power consumption and resource utilization. The provided results underline the importance of the proposed HLS source-to-source compiler for designing approximate algorithms under a set of constraints in an automated manner.

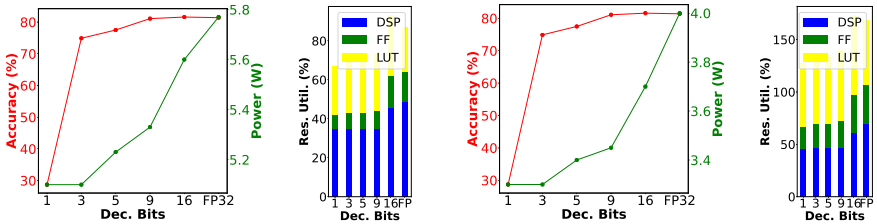
Device Setup

For the device setup we evaluated our algorithms on a Zynq UltraScale+ MPSoC ZCU104 edge FPGA and an Alveo U200 cloud FPGA. The MPSoC



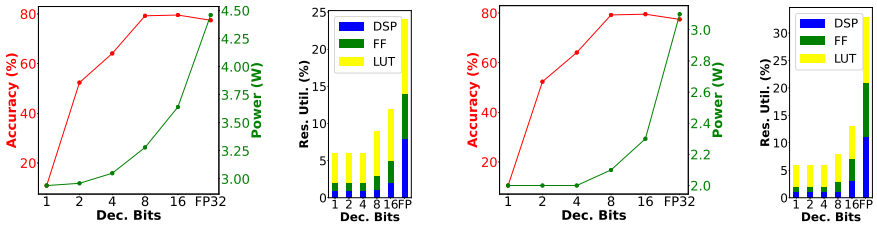
(a) Alveo U200 (b) MPSoC ZCU104

Fig. 2: Signal to noise ratio, power and utilization percentage per resource vs decimal bits for 2D FFT algorithm in U200 (a) and ZCU104 (b) platforms



(a) Alveo U200 (b) MPSoC ZCU104

Fig. 3: Accuracy, power and utilization percentage per resource vs decimal bits for logistic regression algorithm in U200 (a) and ZCU104 (b) platforms



(a) Alveo U200 (b) MPSoC ZCU104

Fig. 4: Accuracy, power and utilization percentage per resource vs decimal bits for gaussian naive bayes algorithm in U200 (a) and ZCU104 (b) platforms

is a System on a Chip (SoC) which has a 16nm XCZU9EG FPGA with 504K logic cells and 48mb of total on-chip memory. The SoC also includes a quad-core ARM Cortex-A53 CPU with 4GB 64-bit DDR4 memory. Regarding the Alveo U200, it is a PCIe attachable FPGA card built on 16nm architecture with 64GB

DDR4 off-chip memory of 77 GB/s bandwidth and 35MB on-chip memory of 31 TB/s total bandwidth.

4.1 Resource Utilization & Algorithmic Performance

Figures 2, 3 and 4 show the performance, power consumption (on-chip) and resource utilization for different decimal bit approximations for the aforementioned case studies. The implemented kernels were simulated on Alveo U200 and MPSoC ZCU104 platforms with 300 MHz target clock frequency. Power analysis was performed on RTL level for each kernel using Vivado Design Suite.

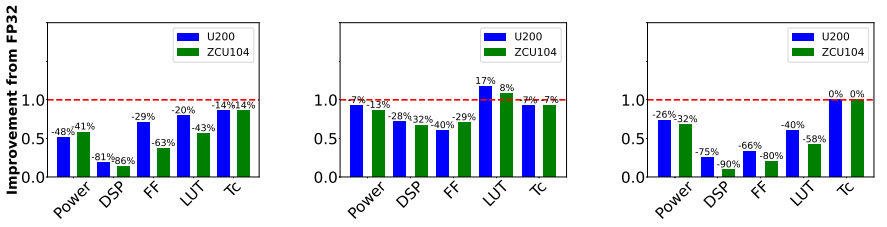
Figures 2a and 2b show the relation between accuracy, power and resource utilization for different levels of approximation for the 2D FFT kernel. An increase in decimal bits results to a rise in algorithm's accuracy and power consumption for both platforms. From 19 to 27 decimal bits there is a linear increase in Signal-to-Noise Ratio (SNR) reaching 80 dBs, while the power consumption for U200 and ZCU104 does not exceed 8 W and 5.5 W respectively. In the case of U200 for up to 27 bits the resource utilization does not significantly differentiate, with the utilization of digital signal processing (DSP) units, flip flops (FF) and lookup tables (LUT) not exceeding 5.4%, 15.4% and 34.4%. Similar results are observed in the ZCU104 platform with the utilization percentages reaching values up to 9%, 21% and 51%. The utilization per resource type for the same number of decimal bits is higher in the case of ZCU104 due to fewer available resources of the MPSoC FPGAs compared to U200.

Figures 3a and 3b show the effect of precision scaling on logistic regression algorithm. For up to 5 decimal bits there is a disproportionate rise between algorithmic performance and resource utilization with the percentage of utilized DSPs, LUTs and FFs not exceeding 21%, 11% and 17% in the case of U200 platform. At the same time, the algorithm's accuracy increases from 28.7% to 77.5% and reaches a plateau. Specifically, any further increase at the decimal bits causes fluctuations up to 5.3% in accuracy while the rise in power consumption may reach 0.6 W on both platforms.

Finally, figures 4a and 4b present the results for gaussian naive bayes algorithm. Regarding the relationship between power consumption, accuracy and decimal bits, the same trend as in figures 3a and 3b can be observed. In particular, for 8 decimal bits the achieved accuracy is almost equal to the floating point (79.25%) while the power consumption is 1 W lower for both U200 and ZCU104. Additionally, resource utilization figures depict that for up to 8 decimal bits the utilization of DSPs, FFs and LUTs does not exceed 1%, 2% and 6% for U200 and 1%, 2% and 5% for ZCU104. For the floating point data type, an increase in DSPs (x8 and x11), FFs (x3 and x5) and LUTs (x1.7 and x2.4) is observed compared to 8 decimal bits for both platforms.

4.2 Approximate & Default Algorithm Comparative Analysis

In figure 5, the floating point implementation of each case study is compared with the corresponding approximate implementation that satisfies the predefined



(a) 2D FFT (b) Logistic Regression (c) Gaussian Naive Bayes

Fig. 5: Savings in power consumption, resource utilization and maximization of the achievable target clock frequency due to the employment of approximate techniques for 2D FFT, logistic regression and gaussian naive bayes algorithms.

performance criteria (SNR above 95 dBs and accuracy above 95% of the floating point) in terms of power, resource utilization and achieved clock frequency. Figures 5a, 5b and 5c exhibit the advantages of the proposed HLS approximation mechanism for the studied ML and DSP algorithms derived from our source-to-source HLS compiler automatically. Our compiler extension leads to a decrease of 27% and 28% in power consumption and a decrease of 61% and 69% in DSP utilization for U200 and ZCU104 platforms, on average. Meanwhile, the significant reduction in used resources and hence in critical paths, leads to lower Tc (clock period) and as a result higher clock frequency (7% lower clock period for both platforms).

5 Conclusion

In this work we proposed a new programming interface based on the new Xilinx Vitis framework. Our optimizations were implemented as an extension to our source-to-source HLS compiler that operates at a high level of abstraction enabling automatic accuracy and resource optimizations to the existing HLS design flow. Also, we provided a common API to support both embedded and cloud FPGAs using the standardized XRT software interface. The HLS transformations were tested on three applications producing a diverse set of kernel versions which gave a broad range of resource-accuracy-power tradeoffs. From a research point of view, we are focusing on adding further functionality to our framework with additional approximation features. The spectrum of possible design space tradeoffs is vast but this work shed some light to the area with successful results aiming to make FPGAs contribute fundamentally with minimum programming effort into the software-hardware ecosystem.

Acknowledgment

This work has been supported by the E.C. funded program SERRANO under H2020 Grant Agreement No: 101017168

References

1. Xilinx, https://www.xilinx.com/
2. Barroso, L.A., Hölzle, U.: The case for energy-proportional computing. Computer **40**(12), 33–37 (2007)
3. Chippa, V.K., Chakradhar, S.T., Roy, K., Raghunathan, A.: Analysis and characterization of inherent application resilience for approximate computing. In: 2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC). pp. 1–9 (2013). https://doi.org/10.1145/2463209.2488873
4. Danopoulos, D., Kachris, C., Soudris, D.: A Quantitative Comparison for Image Recognition on Accelerated Heterogeneous Cloud Infrastructures, pp. 171–189 (09 2019). https://doi.org/10.1201/9780429399602-8
5. Danopoulos, D., Kachris, C., Soudris, D.: Utilizing cloud fpgas towards the open neural network standard. Sustainable Computing: Informatics and Systems **30**, 100520 (2021). https://doi.org/https://doi.org/10.1016/j.suscom.2021.100520, https://www.sciencedirect.com/science/article/pii/S2210537921000135
6. Finnerty, A., Ratigner, H.: Reduce power and cost by converting from floating point to fixed point. WP491 (v1. 0) (2017)
7. Guan, Y., Liang, H., Xu, N., Wang, W., Shi, S., Chen, X., Sun, G., Zhang, W., Cong, J.: Fp-dnn: An automated framework for mapping deep neural networks onto fpgas with rtl-hls hybrid templates. In: 2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM). pp. 152–159 (2017). https://doi.org/10.1109/FCCM.2017.25
8. Guo, K., Sui, L., Qiu, J., Yao, S., Han, S., Wang, Y., Yang, H.: Angel-eye: A complete design flow for mapping cnn onto customized hardware. In: 2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI). pp. 24–29 (2016). https://doi.org/10.1109/ISVLSI.2016.129
9. Homsirikamol, E., George, K.G.: Toward a new hls-based methodology for fpga benchmarking of candidates in cryptographic competitions: The caesar contest case study. In: 2017 International Conference on Field Programmable Technology (ICFPT). pp. 120–127 (2017). https://doi.org/10.1109/FPT.2017.8280129
10. Ko, J.H., Fromm, J., Philipose, M., Tashev, I., Zarar, S.: Precision scaling of neural networks for efficient audio processing. arXiv preprint arXiv:1712.01340 (2017)
11. Nane, R., Sima, V., Pilato, C., Choi, J., Fort, B., Canis, A., Chen, Y., Hsiao, H., Brown, S., Ferrandi, F., Anderson, J., Bertels, K.: A survey and evaluation of fpga high-level synthesis tools. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **35**, 1–1 (12 2015). https://doi.org/10.1109/TCAD.2015.2513673
12. Saadat, H., Bokhari, H., Parameswaran, S.: Minimally biased multipliers for approximate integer and floating-point multiplication. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **37**(11), 2623–2635 (2018)

13. Sharma, H., Park, J., Suda, N., Lai, L., Chau, B., Kim, J.K., Chandra, V., Esmaeilzadeh, H.: Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural network. 2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA) pp. 764–775 (2018)
14. Shawahna, A., Sait, S.M., El-Maleh, A.: Fpga-based accelerators of deep learning networks for learning and classification: A review. IEEE Access **7**, 7823–7859 (2019). https://doi.org/10.1109/ACCESS.2018.2890150
15. Shi, W., Cao, J., Zhang, Q., Li, Y., Xu, L.: Edge computing: Vision and challenges. IEEE internet of things journal **3**(5), 637–646 (2016)
16. Wess, M., P D, S.M., Jantsch, A.: Neural network based ecg anomaly detection on fpga and trade-off analysis. pp. 1–4 (05 2017). https://doi.org/10.1109/ISCAS.2017.8050805