

Received February 7, 2020, accepted February 25, 2020, date of publication February 28, 2020, date of current version March 11, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2977087

# Computing in the Blink of an Eye: Current Possibilities for Edge Computing and Hardware-Agnostic Programming

MOHAMMAD HOSSEIN GHASEMI<sup>1</sup>, OSCAR LUCÍA<sup>1,3</sup>, (Senior Member, IEEE),  
AND SERGIO LUCIA<sup>1,2</sup>, (Member, IEEE)

<sup>1</sup>Chair of Internet of Things for Smart Buildings, Technische Universität Berlin, 10587 Berlin, Germany

<sup>2</sup>Einstein Center Digital Future, 10117 Berlin, Germany

<sup>3</sup>Electronic Engineering and Communications Department, Aragon Institute of Engineering Research (I3A), Universidad de Zaragoza, 50018 Zaragoza, Spain

Corresponding author: Sergio Lucia (sergio.lucia@tu-berlin.de)

This work was supported in part by the Spanish Ministerio de Ciencia e Innovación (MICINN) and Agencia Española de Investigación (AEI) under Project RTC-2017-5965-6, in part by the EU through the FEDER Program, in part by the Diputación General de Aragón (DGA)-FSE, in part by DGA under Project LMP106\_18, in part by the German Research Foundation, and in part by the Open Access Publication Fund of TU Berlin.

**ABSTRACT** With the rapid advancements of the internet of things, systems including sensing, communication, and computation become ubiquitous. The systems that are built with these technologies are increasingly complex and therefore require more automation and intelligent decision-making, while often including contact with humans. It is thus critical that such interactions run smoothly in real time, and that the automation strategies do not introduce important delays, usually not larger than 100 milliseconds, as the blink of a human eye. Pushing the deployment of the algorithms on embedded devices closer to where data is collected to avoid delays is one of the main motivations of edge computing. Further advantages of edge computing include improved reliability and data privacy management. This work showcases the possibilities of different embedded platforms that are often used as edge computing nodes: embedded microcontrollers, embedded microprocessors, FPGAs and embedded GPUs. The embedded solutions are compared with respect to their cost, complexity, energy consumption and computing speed establishing valuable guidelines for designers of complex systems that need to make use of edge computing. Furthermore, this paper shows the possibilities of hardware-agnostic programming using OpenCL, illustrating the price to pay in efficiency when software can be easily deployed on different hardware platforms.

**INDEX TERMS** Internet of Things, edge computing, FPGA, system on chip, neural network.

## I. INTRODUCTION

The developments in the internet of things (IoT) are enabling the design and deployment of complex systems with ubiquitous sensing capabilities. However, the full potential of the internet of things will be unveiled only if this data can be properly processed and smart decision-making strategies can be computed, leading to important benefits in the form of energy savings, improved security or improved performance of complex interconnected systems that have a critical impact on the life of millions of persons. Typical examples of such complex systems include energy networks, smart buildings, smart factories or intelligent transportation systems. The large amount of data collected with IoT devices must be

properly processed to obtain an added value from such sensors [1]. The common system architecture for today's IoT systems is the cloud computing paradigm. In cloud-based systems, data is gathered by the edge devices and then transferred to the cloud servers to be processed. These systems require high-bandwidth internet connection to all edge nodes. Also, the cloud servers must be powerful enough to process all the data in the specific time. Different algorithms are then run based on the collected data. Recently, machine learning techniques, and especially deep learning [2], have become very popular due to the advanced predicting capabilities of deep learning models as well as the low effort required for feature engineering, which is substituted by an enlarged size of a neural network where the most important features of a selected application are automatically learnt.

The associate editor coordinating the review of this manuscript and approving it for publication was Min Jia .



FIGURE 1. Edge computing.

Transferring all these data from edge to cloud is a big challenge for future internet infrastructure and, additionally, many critical IoT systems must have a very short response time (FIGURE 1). For example, in applications with human-machine cooperation, the time delay between user input and the system response must be below the recognizable time by the human so the user does not notice this delay, and a cooperation can take place in a natural manner [4]. In other application such as autonomous cars, response time for decision-making is vital and it cannot rely on a cloud infrastructure [5].

In this context, edge computing is called to play an ever increasing key role in the development of IoT-powered systems [6]. While traditional edge processing has been limited to the filtering or aggregation of sensor data [7], the increasing improvement of computing hardware, algorithms and tailored implementations enables the deployment of complex decision-making strategies even on resource-constrained hardware platforms.

The design of IoT systems is a very interdisciplinary field. It is therefore difficult for a hardware designer to fully understand the details of complex machine learning algorithms. At the same time, it is a difficult challenge for a domain specialist to choose the optimal hardware on which different types of algorithms should be deployed. Alleviating such challenges is the main motivation of this work.

The main contribution of this paper is the evaluation of the most commonly used embedded computing platforms for the evaluation of a deep neural network that can serve as guideline for system designers or domain specialists. The case study of the evaluation (not the training) of a neural network includes many modern decision-making algorithms, ranging from predictive maintenance [8], to smart energy management systems [9] or face recognition [10]. The trade-offs for each platform are presented with a focus on computing performance, power consumption, complexity of implementation and hardware cost.

The hardware heterogeneity of IoT systems is an additional challenge for the design, programming and maintenance of such systems. Changing an algorithm from a general-purpose CPU to a low-cost microcontroller or a FPGA is a very time consuming and error-prone task. Hardware-agnostic programming languages, such as OpenCL [11], try to alleviate this problem by providing the possibility of deploying (almost) the same code on different hardware platforms.

The universality of such implementation comes at the cost of a performance decrease because customization to the details of each hardware platform is limited. This paper illustrates how large the performance decrease of a universal implementation is for the use-case of the evaluation of a deep neural network on different hardware platforms.

Several studies have been published in recent years that benchmark the performance of different edge devices. For example, [12] presents a detailed performance comparison of different Graphical Processing Units (GPUs). Most of the works that consider several hardware platforms for edge computing and machine learning focus on a comparison of CPU and GPU [13], [14]. Only few works consider the deployment on other edge devices, such as Raspberry Pis [15] or FPGAs [16]. In contrast, this paper presents the possibilities of very diverse embedded platforms, ranging from micro-controllers to GPUs, to provide broad guidelines for many different possible applications. The hardware heterogeneity is often subject of study [17], [18], but very few works have considered the use of hardware-agnostic languages [13], [19] that can lead to simple deployment on different platforms. An additional contribution of this work is to provide a quantifiable overview of the tradeoffs in performance that a general hardware-agnostic programming offers when compared to a tailored implementation, as for example, a detailed digital design in the case of FPGAs.

This paper is organized as follows. Section II presents the main advantages of edge computing and why it is important for future IoT systems. Section III describes the benchmark used to evaluate the different hardware platforms. The studied platforms and the main results of the paper are described in Section IV. Section V discusses the main conclusions that can be drawn from the results as well as the possibilities of a hardware-agnostic implementation using OpenCL. Finally, Section VI presents the conclusions and future planned work.

## II. COMPUTING ON THE EDGE

With the advances of ubiquitous sensing and computing, the problem of processing an increasing amount of data becomes a significant challenge. The traditional and simplest solution to handle this issue was to collect all this data from field devices and send it to the cloud to be handled in a client-server paradigm [3], [20]. This trend was strongly supported by the development of the elastic computing approaches and their implementation in different cloud services. The development of low-cost, low power microcontrollers, cost effective FPGAs as well as tools to facilitate the hardware design (such as High Level Synthesis) increased significantly the possibilities of edge computing and their integration in Industry 4.0 [21] and telecommunications 5G technology [22].

The idea of edge computing can refer to different forms of computing, including cloudlets or mobile edge computing [3]. This work considers the case of edge computing where intelligent algorithms, for example, based on deep neural networks, can be run (not trained) on embedded devices.

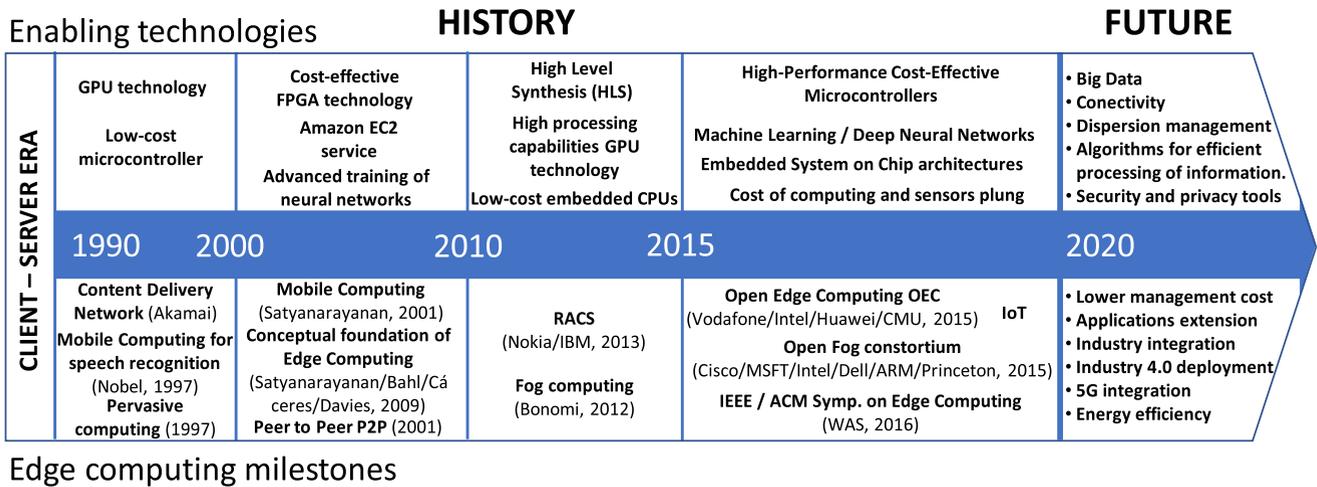


FIGURE 2. Historical introduction to edge computing: enabling technologies and applications [3].

There are several reasons to do so, such as lower latencies or lower energy consumption, which are illustrated in the remainder of this section.

**A. BANDWIDTH SHORTAGE**

By 2025, it is expected that there will be more than 40 billion connected IoT devices generating 79.4 zettabytes of data, following an annual growth of 28.7% [23]. Most data will be created by surveillance applications, but new applications in the industrial and medical areas will significantly increase the available data. Despite advances in communications, this will greatly exceed communication capabilities, highlighting the importance of edge computing.

Recent works have shown that it is possible to achieve intelligent decision-making even on resource constrained embedded devices. For example, optimization-based techniques used for energy management systems were deployed on microcontrollers [24], as well as power electronic controllers on FPGAs [25]. This avoids the necessity to continuously send all data and decisions to the cloud, limiting it to less frequent updates and upgrades.

**B. SECURITY AND PRIVACY**

Today, millions of embedded devices are used in safety and security critical applications such as industrial control systems, modern vehicles, and critical infrastructure [26]. IoT devices are always producing, consuming and exchanging critical data and this makes the security of the IoT system very important [7]. Edge computing can help the security and privacy aspects of IoT applications from several viewpoints [27], [28]. First, if less data is sent over the network, less data is vulnerable to unsecured communication. Additionally, more processing power at the edge results in more encryption and decryption power which can bring more security. For applications that contain sensitive personal data as in the field of smart homes, increasing the amount of edge computing can reduce the amount of data that users need to share with cloud services, leading to larger control of data ownership and data sovereignty issues by the user [29].

**C. RESPONSE TIME**

The latency of a cloud based system is affected by several different factors, including network effective bandwidth and computation power available at the cloud server [30]. These results in an unpredictability in the response time of the application. But in many applications, the reliability of the system actually depends on a specific and limited response time.

**III. PROPOSED BENCHMARK: DEEP LEARNING**

Comparing performance of different hardware platforms and different software implementations is a challenging task. In order to obtain fair and comparable results, the evaluation of a deep neural network is chosen as a benchmark. Neural networks already play a significant role in intelligent systems and, because of recent developments in deep learning, it is expected that their role will be even more relevant in the future.

In this work, we used a fully connected deep neural network with three neurons in the input layer and two neurons in the output layer. This structure corresponds to the neural network necessary to represent a predictive controller of a half-bridge power inverter for induction heating as presented in [31]. Control of power electronics, e.g., in the field of power systems or smart grid is an example where cloud computing is not possible because intelligent decisions based on sensor information need to be taken in the range of microseconds.

A feedforward deep neural network is defined as a composition of several functions that compute the neural network function of  $N : \mathbb{R}^{n_x} \rightarrow \mathbb{R}^{n_y}$ , and can be written as:

$$N(x; \theta, M, L) = f_{L+1} \circ g_L \circ f_L \circ \dots \circ g_1 \circ f_1(x), \quad (1)$$

where the input of the network is  $x \in \mathbb{R}^{n_x}$  and the output of the network is  $y \in \mathbb{R}^{n_y}$ .  $M$  is the number of neurons in each hidden layers and  $L$  is the number of hidden layers. Each hidden layer applies an affine function:

$$f_l(\xi_{l-1}) = W_l \xi_{l-1} + b_l, \quad (2)$$

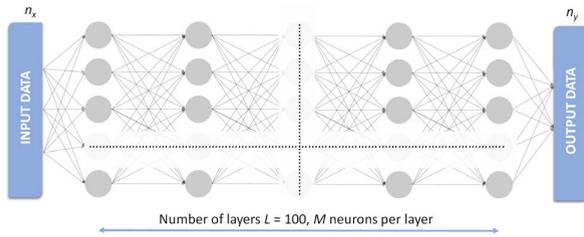


FIGURE 3. Proposed deep neural network benchmark.

where  $\xi_{l-1} \in \mathbb{R}^M$  is the output of the previous layer with  $\xi_0 = x$ . In a standard neural network, a nonlinearity in the form of an activation function is used after each affine function. As activation function, a rectifier linear unit function has been used, which is defined as:

$$g_l(f_l) = \max(0, f_l). \tag{3}$$

The parameter  $\theta = \{\theta_1, \dots, \theta_{L+1}\}$  contains all the weights  $W_l$  and biases  $b_l$  of the affine functions of each layer  $l$ .

The main task for neural network inference computation is the affine function calculation. To calculate the output value of each layer, a matrix-vector multiplication and a vector addition must be performed. This calculation can be represented in digital form as:

$$\xi'_i = \sum_{j=1}^M W_{ij} \xi_j + b_i, \quad \forall i = 1, \dots, M, \tag{4}$$

in which  $\xi'_i$  and  $\xi$  are respectively the outputs of the current and the previous layer [32].

To enable a fair comparison between different hardware platforms, a neural network with  $L = 100$  hidden layers is chosen, except for the embedded microcontroller where 10 layers are selected because of the significant lower memory capacity. To show performance of the hardware platforms for varying problem sizes, the number of neurons per layer  $M$  is varied. FIGURE 3 presents a schematic representation of the benchmark used throughout this work.

By changing the number of layers and the number of neurons in every layer, and measuring the time and energy consumption that each platform needs to perform the required

computations defined in (1), the capabilities for edge computing of each platform can be compared.

#### IV. PLATFORM EVALUATION

A set of the most commonly used embedded computing platforms has been chosen to be compared as possible IoT edge computing alternatives (FIGURE 4): a low-power microcontroller, and embedded microcontroller, a Field Programmable Gate Array System on Chip (FPGA SoC) as well as an embedded Graphical Processing Unit (GPU). Additionally, two different general-purpose reference systems (not embeddable) have been included for comparison purposes: a common laptop CPU and a laptop GPU. This section describes the most relevant specifications of the chosen hardware platforms and discusses the obtained results. Each embedded platform has been compared with a focus on different performance indicators: computation capabilities, energy and power consumption, implementation complexity and cost.

##### A. GENERAL PURPOSE REFERENCE SYSTEMS

###### 1) INTEL I5 PROCESSOR

In order to put the performance of the embedded devices into perspective, all embedded platforms are compared to a general-purpose CPU that runs under the Windows Operating System. The technical specifications of the general-purpose CPU are listed in Table 1.

For general purpose CPU, the neural network inference was implemented exploiting Intel’s MKL library. Using this library has resulted in 45 times improvement in performance in comparison with a simple single core application running on the same device. The main motivation of using an optimized implementation for Intel’s CPUs is to have a fair comparison between the embedded platforms and a general-purpose CPU, as optimized implementations have been attempted for each hardware platform.

###### 2) NON-EMBEDDED GPU

A standard integrated Graphical Processing Unit is also considered as reference system. Its specifications are summarized in Table 2. Finally, the comparative results of both general purpose reference systems are included in Table 7 to enable a direct comparison.

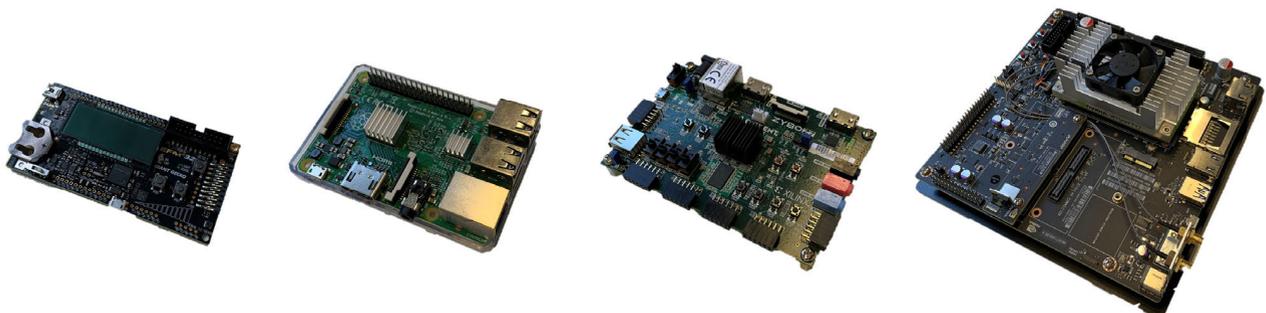


FIGURE 4. Evaluated platforms. From left to right: microcontroller, embedded processor, FPGA SoC, and embedded GPU.

**TABLE 1. Specifications of the general-purpose CPU device.**

Specification	Value
Number of cores	2
Number of threads	4
Base frequency	2.5 GHz
Max turbo frequency	3.1 GHz
Memory beside CPU	6 GB
Memory type	DDR3
Number of cores	2

**TABLE 2. Specifications of the Geforce GPU device.**

Specification	Value
Device name	Geforce GT 630M
CUDA Cores	96 Nvidia Fermi cores
Core speed	Up to 800 MHz
Memory	2GB DDR3

**TABLE 3. Specifications of the microcontroller device.**

Specification	Value
Part number	EFM32GG990F1024
Memory (RAM)	128 KB
Memory (Flash)	1 MB
Clock Frequency	48 MHz

## B. MICROCONTROLLER

Because of their simplicity and efficiency, microcontrollers have always been used as processing unit in embedded applications like edge computing. For example, in [33], a microcontroller design has been introduced for IoT edge computing and its performance and power consumption has been measured. Due to advances in microelectronics, we now have access to relatively high-performance and low-power microcontrollers. In this work, the EFM32 Giant Gecko board manufactured by Silicon Lab has been used.

The micro-controller unit of this board is based on ARM Cortex-M3 architecture. This is a mid-range low power microcontroller, so it is a suitable example for IoT edge devices. The specifications of this microcontroller are presented in Table 3.

The performance results for the neural network inference application on this microcontroller are shown in FIGURE 7 (a). Standard microcontrollers have only a few hundreds of kilobytes of RAM. Because of this, it was not possible to evaluate large neural networks, and this is the only hardware platform where a smaller network with only 10 layers has been used. Small networks can still be used to perform advanced decision-making strategies as shown in [9] for the smart building energy management system. In addition, the best performance achieved with this microcontroller and the pick power consumption are shown in Table 7.

Because the specific microcontroller used in this work does not have a dedicated floating-point unit, it must emulate all the floating-point operations in the software level

**TABLE 4. Specifications of the embedded processor device.**

Specification	Value
Board Model	Raspberry Pi 3 B+
SoC Part number	Broadcom BCM2837B0
Processor	4 core Cortex-A53 (ARMv8) 64-bit
Clock frequency	1.4 GHz
Memory (RAM)	1GB LPDDR2 SDRAM

using integer operations. This has a significant impact on the performance of the microcontroller inferring a neural network. Therefore, in microcontroller design flow, changing the accuracy of the computation from floating-point to fixed-point, may have a large impact on the final performance and must be considered by the developer. Also, in order to keep the accuracy in an acceptable level, a fixed-point analysis is required before the implementation.

## C. EMBEDDED PROCESSOR

A possible solution to alleviate the problem of limited performance and memory typical of low-cost microcontrollers is to use embedded processors. Embedded processors can run at significantly higher clock frequencies (up to 2 GHz) and can have memory capacities up to several GBs. In this work, the neural network inference problem is evaluated on a Raspberry Pi3 B+ embedded processor. The specifications of the Raspberry Pi3 B+ board used in this work are shown in Table 4.

Embedded processors usually need operating systems to handle some system level tasks like memory management and process handling. This also adds to the complexity of application development with embedded processors. The performance results of the implementation of the neural network inference with an embedded processor are shown in FIGURE 7 (b). Peak performance and power consumption of the embedded processor in neural network inference application are also shown in Table 7.

## D. FPGA

FPGA-SoCs were introduced in order to combine the flexibility of software design and the performance gain of FPGAs. In this work, a Xilinx Zynq device has been chosen as the target FPGA-SoC. The evaluation board used in this work is Zybo Z7 made by Digilent. The specifications of the Zybo Z7 board are in Table 5.

To have a more complete conclusion about FPGA-SoC implementation, we have evaluated all major design methodologies for FPGA-SoCs. The structure of a typical system in an FPGA-SoC is shown in FIGURE 5. The main application is performed in the software side (shown in the picture as the Hard Processing System) and the computation-intensive parts are implemented in the FPGA side (shown in the picture as Custom Hardware). The key difference between the various implementation methodologies for FPGA-SoC lies on the different approaches to design and implement the accelerator part in the FPGA side and how to integrate it with software side.

TABLE 5. Specifications of the Xilinx ZYNQ device.

Specification	Value
Device name	Z-7020
SW side	
Processor	Dual-Core ARM Cortex-A9 MP Core
L1 cache	32KB Instruction, 32KB Data per processor
L2 cache	512 KB
External Memory	1 GB DDR3
FPGA side	
Logic cells	85K
Total Block RAM	4.9 Mb
DSP slices	220

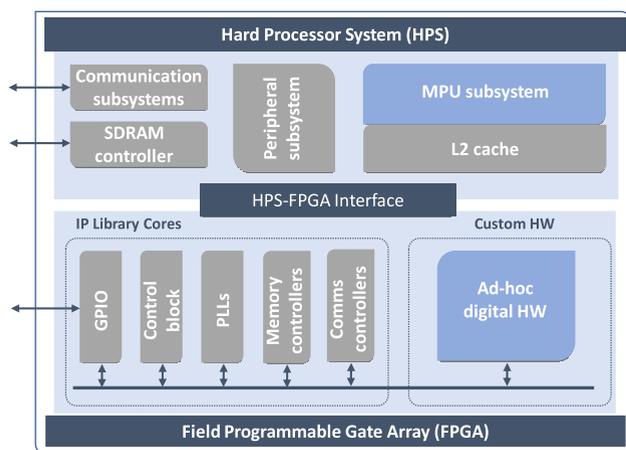


FIGURE 5. System structure of a typical FPGA-SoC system.

To accelerate an application with the FPGA-SoC, first the bottleneck of the application must be determined. The bottleneck of the application is the part that needs a significant amount of computation and where the majority of the execution time is spent. To find this, first the application must be profiled.

One way of profiling an application is to run it and then record the execution time for each function. Then by comparing this execution times, we can decide on which part to be accelerated in FPGA part. In neural network inference, the vector-matrix multiplication operation to calculate the results of every layer of the neural network is the bottle neck of the application. For this reason, in all FPGS-SoC implementations, we consider the hardware accelerator to be a matrix-vector multiplier co-processor.

In our work, we have followed three main approaches to design the system in the FPGA-SoC platform. These approaches are explained as follows.

1) REGISTER TRANSFER LEVEL (RTL)

There are a lot of research efforts to accelerate a neural network inference task with FPGAs. For example in [34], PIE has been introduced as a novel pipelined implementation to reach parallelism between layers in addition to in layer parallelism. Also in [35], a layer based structure has been introduced to perform the inference task in CNNs. Both these works have reported an improvement in performance and

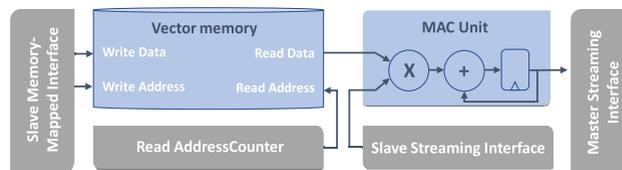


FIGURE 6. System structure of a typical FPGA-SoC system.

power consumption in FPGA implementation in comparison to regular implementations.

In this work, we have designed an accelerating core for neural network inference task. The main idea in this accelerating core is to benefit from the pipelining capability of FPGAs to parallelize the computation for each output neuron of a layer. Because we have assumed that all the data is coming from the DDR memory through the AXI bridge between the PL and PS side, the bottleneck of the performance becomes this data movement in the system.

In RTL design, first the accelerator core is manually designed and implemented with a hardware descriptor language (HDL). After testing and verifying the core, it is integrated with the software side with a platform design tool for which we have here used Vivado System Integrator. To reach a high-performance implementation for the matrix-vector multiplier accelerator core, we have proposed a hardware structure that is shown in FIGURE 6.

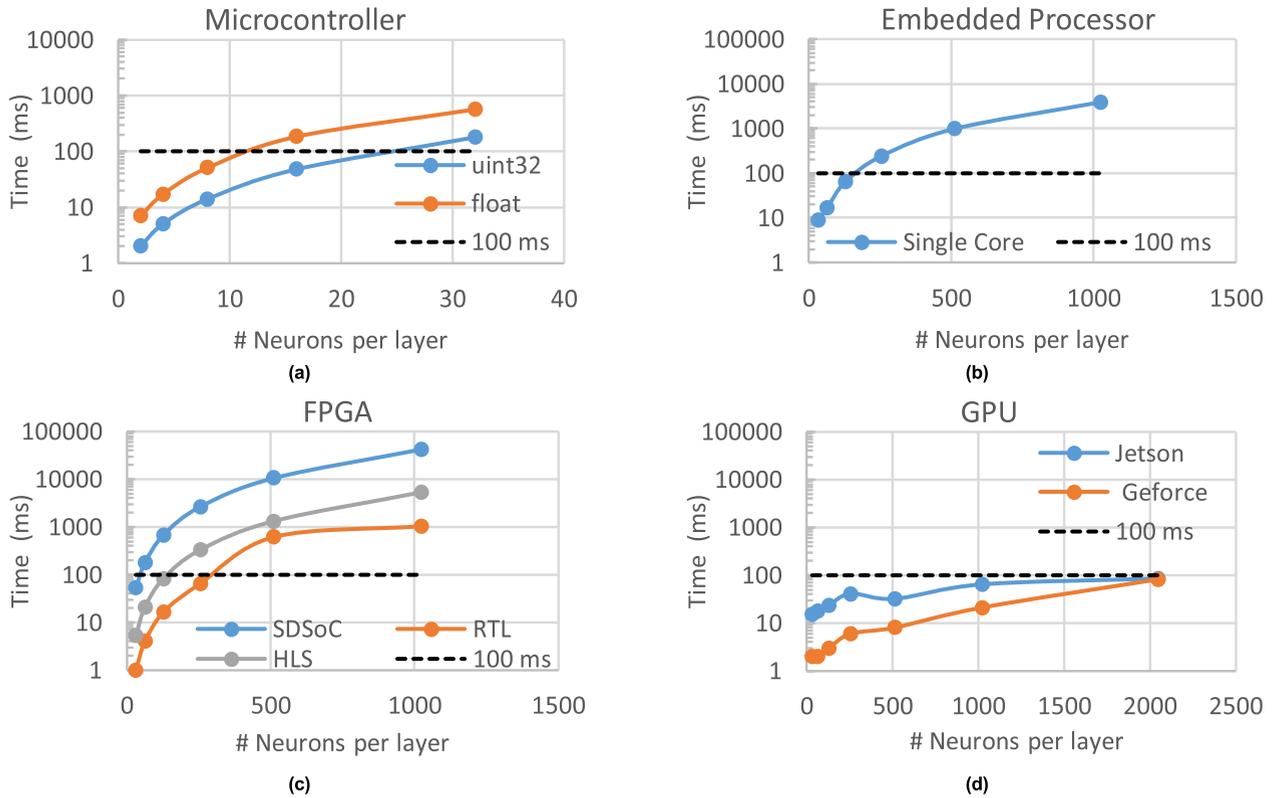
2) HIGH LEVEL SYNTHESIS (HLS)

In this approach, the accelerator core is designed and described with a high level programming language like C/C++ [25], [36]. Then, this highly described system is synthesized to a lower level hardware system. In this work, this is done through Xilinx HLS design flow. This gives the opportunity to design the accelerator core with C/C++ language and skip the complexity of RTL design and its verification.

Several works have considered this type of implementation for a neural network. For example, in [37] a pipelined architecture for a CNN has been implemented using Xilinx HLS compiler. The goal in that design was to use the loop unrolling and pipelining techniques to get the most possible hardware utilization and performance.

The integration of the accelerator core with the software side must still be done manually within the Vivado System Integrator. However, because of the limited customization capabilities, it is expected that the HLS implemented core is not as efficient as of the RTL design, both in performance and in area usage.

The most important directives that are given to the HLS compiler to customize as much as possible the hardware design are the following. The directive `#pragma HLS PIPELINE II = 1` is used to pipeline the main matrix vector multiplication with initiation interval equal to 1. `#pragma HLS INLINE` is used to make the hierarchal structure smaller and also make the opportunity of resource sharing between functions in the code.



**FIGURE 7.** Performance results for the studied platforms: (a) microcontroller, (b) embedded processor, (c) FPGA RTL, HLS and SDSoC implementations, and (d) Jetson TX2 and GForce GPUs.

**TABLE 6.** Specifications of the Nvidia embedded GPU device.

Specification	Value
Device name	Nvidia Jetson TX2
CPU	HMP Dual Denver 2/2MB L2 + Quad ARM® A57/2MB L2
GPU	NVIDIA Pascal™, 256 NVIDIA CUDA® cores
Memory	8 GB 128-bit LPDDR4 58.3 GB/s

### 3) SOFTWARE DEFINED SYSTEM (XILINX SDSOC)

In this implementation method, the system is completely defined and implemented in a software environment. The part of the design that needs to be accelerated in the FPGA side is marked as hardware-accelerated and all the accelerator cores' design and integration with the software side is performed automatically. Again, it is expected that at the same time that this will bring simplicity in the design process, it will also reduce the efficiency of the final design. The same HLS compiler and the same HLS directives are used in this case for the hardware accelerator.

The performance and power consumption of the three implementations previously described, with a decreasing level of implementation complexity, are presented in FIGURE 7 (c). As it has been previously explained, RTL achieves the highest performance due to the higher optimization grade whereas HLS and SDSoC designs simplifies the design process at the cost of significant performance loss.

### E. EMBEDDED GPU

Another way to accelerate an edge application is to use an embedded GPU besides the main processor. In this work, a Nvidia Tegra Jetson TX2 SoC with embedded GPU has been used. The specifications of the GPU-SoC are listed in Table 6.

As discussed in the previous subsection, in a heterogeneous implementation, the part of the application that requires large computation is accelerated in a co-processor. Within the Tegra Jetson TX2 SoC considered in this work, the co-processor is a Nvidia Pascal GPU with 256 computing cores. The main difference with respect to the FPGA-SoC is in the type of the co-processor. In contrast to the reprogrammable hardware present in FPGAs, the GPU-SoC offers a co-processor that contains many small processing units able to run the same operation on different parts of the data.

The results for the performance achieved with the Embedded GPU-SoC are shown in FIGURE 7 (d). According to the performance results and the power consumption measurement, the embedded GPU achieves the maximum performance and the best ratio of computation performance to power consumption from all embedded platforms. These values have been listed in the Table 7.

The Jetson TX2 SoC that has been used in this work is an GPU designed especially for embedded applications. It means that it has been optimized for logic size, cost and energy consumption. This optimization has effects on the performance of the device. But even when comparing it

**TABLE 7. Summary of performance, power consumption and design complexity for the evaluated platforms.**

Type of Platform	Platform	Max Performance (MFLOP)	Power consumption (W)	Performance per Power ratio (MFLOP/W)	Design complexity
General Purpose	Intel CPU	9670	35	260.0	Low
	Geforce GPU	10200	33	309.0	Medium
Embedded Devices	Microcontroller (float)	0.0042	0.017	0.247	Low
	Embedded Processor	53.46	5	10.62	Low
	FPGA-SoC (RTL)	200	2.1	95.00	High
	FPGA-SoC (HLS)	40	2.1	19.04	Medium
	FPGA_SoC (SDSoC)	5	2.1	3.80	Low
	Jetson TX2	9560	7.5	1274.0	Medium
OpenCL	Intel CPU	5382	35	153.0	Low
	Geforce GPU	368	33	11.0	Low
	FPGA-SoC	11.4	2.2	5.7	Low

with a non-embedded, more powerful GPU, the performance of the device is comparable (see FIGURE 7 (d)) with a significant decrease in power consumption as can be seen in Table 7.

#### F. PLATFORM INDEPENDENT IMPLEMENTATION

One big challenge for IoT developers to move their designs toward edge computing is linked to the platform-dependent implementation. Designing a system based on an embedded GPU is completely different from implementing the same system based on an FPGA-SoC. The details of each hardware platform are very diverse, and they must be considered to have an optimized system.

To deal with this diversity in design and implementation methods, some solutions have been introduced. In this work, the use of OpenCL as a unified design method for all platforms has been evaluated. An OpenCL neural network inference application has been tested on the three different platforms that currently support OpenCL and were used in this work (Intel CPU, Geforce GPU and FPGA SoC).

OpenCL is a heterogeneous programming language that is designed to be platform independent. However, it is still not possible to have exactly the same OpenCL code running on all platforms.

An OpenCL code consists of two different parts, the host program that runs on the main processor and the device program (kernel code) that is going to be accelerated on the co-processor which can be a GPU, an FPGA or simply another CPU. Because of the different architecture in co-processor devices, it is not possible to get the same performance gain from them with the same kernel code. For example, in FPGAs, pipelining is very important but for GPUs data parallelism is more vital.

The performance results for each platform running the OpenCL application are listed in Table 7. As expected, the performance of the OpenCL program is significantly lower than the individually optimized application for each

platform. However, the much lower complexity of the implementation may make it considerable for edge computation and IoT development.

## V. DISCUSSION

This section discusses the main conclusions and observations that can be obtained from the results presented previously. While the exact performance metrics might depend on the application at hand, it is expected that these results can serve as an indicator for the estimated performance of the different hardware platforms and the different types of implementation which can be used as guidelines by non-expert software/hardware designers (see summary on FIGURE 8).

### A. PERFORMANCE

For heterogeneous computing platforms, it is very important to consider the concept of concept of the computation to communication (CTC) ratio when analyzing the obtained results (see VI). As explained in [37], with higher CTC ratios, more performance gain is achievable until the boundaries of the system's memory capabilities are reached.

The considered embedded IoT test application has a low CTC ratio. In each layer of our test neural network application, we are performing a matrix-vector multiplication and a vector-vector addition. The CTC rate will be different significantly larger for applications that use large input dimensions, as e.g., in the field of image recognition. This means that the I/O bandwidth can rapidly become the main bottleneck of the implementation, reducing the advantages of hardware accelerators.

Among the embedded devices evaluated in this work, microcontrollers have weakest computing performance, in the order of tens of kFLOPS. Most of low-cost low-energy microcontrollers do not have a dedicated floating-point unit and they have to emulate those operations in software level. Changing the accuracy of the computations from floating-point to fixed-point can improve their performance significantly. In this work, a 32-bit fixed-point operation

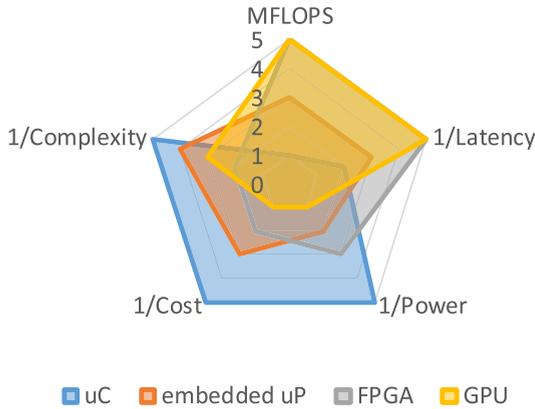


FIGURE 8. Comparative chart.

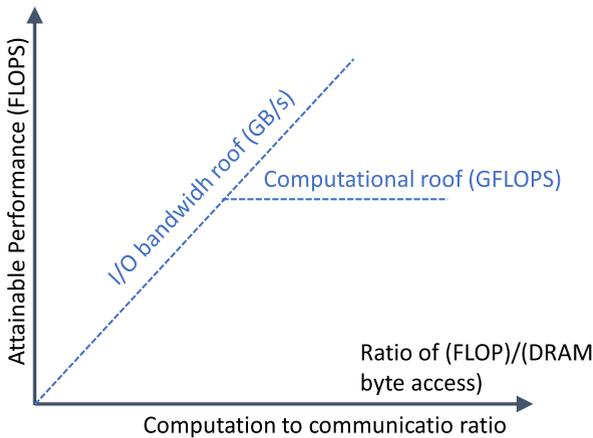


FIGURE 9. Computational roof detailed in [37].

instead of floating point resulted in three times lower computing times for the same network.

Embedded processors use an operating system to handle tasks like scheduling and memory management. Still, because of more powerful processors and also more memory available to them, these processors deliver significantly more performance than microcontrollers.

For further increases in performance on embedded devices, heterogeneous implementations can be considered. FPGA-SoCs are suitable platforms for edge computing and bring a wide variety of implementation options. The application type is very important when choosing a proper implementation method for FPGA-SoCs, especially in cases where the input/output bandwidth between accelerator code and software, or the optimization of the digital resources, is the main bottleneck. Automatic tools such as HLS significantly reduce the design time but can also lead to important performance decreases when compared to a very detailed RTL implementation. For instance, whereas RTL implementation can easily achieve initialization intervals equal to one for the FPU MAC unit, i.e. one new data each clock cycle, HLS or SDSoc implementations require more cycles. Although in other applications this issue can be addressed by interleaving

the input array to the core, this is not useful in the considered edge computing application where task parallelism rather than data parallelism is required. The parallelization capabilities of embedded GPUs can be easily leveraged to obtain very strong performances, especially in the case of large problems.

**B. POWER CONSUMPTION**

For embedded devices that are going to be used in the edge of the IoT systems, power consumption is a critical performance indicator. In some of the applications, the edge device must operate with battery power. As expected, low-power microcontrollers have the least power consumption among all the platforms. However, to have a fair comparison between platforms, the performance per power ration is evaluated for all platforms and shown in Table 7.

As it can be seen, the power consumed by the embedded devices is much less than the power consumption of the general-purpose CPUs and GPUs. Among embedded devices, embedded GPUs are promising alternatives for complex calculations on the edge, as they have significantly better performance/power ratio as it can be seen for the Jetson TX2.

To measure the power for the Jetson TX2 and the EFM32 microcontroller, the available internal measurement capabilities that the boards offer were used. For the FPGA board, the overall power was measured using an external microcontroller and a shunt resistor. The power consumption that is reported in Table 7 for the general-purpose platforms is the one provided by the specifications of the hardware given by the vendor.

**C. COMPLEXITY OF DESIGN**

More complexity in design means more development time that results in longer time to market and higher development costs. While the flexibility in software design is usually high, it is not the case for hardware design. In order to combine the positive features of each side, techniques like heterogeneous programming and hardware-software co-design have been introduced.

In this work, heterogeneous programming was evaluated using with OpenCL and CUDA on FPGAs and GPUs. CUDA resulted in a better performance for Nvidia’s GPUs, but the main advantage of OpenCL is that it is hardware independent and it is usable for both GPUs and FPGAs.

Also, the HLS tools are becoming popular for hardware design, especially in FPGAs. New approaches in HLS techniques have enabled the developers to manage hardware accelerated applications without writing even a single line of HDL code. They need much lower design time, but they are still limited in design flexibility. However, many designs can be tested easily, making it possible to perform an automated sweep of the design space to obtain near optimal hardware designs as done in [29].

**TABLE 8. Computing alternatives with a running time of 100 ms.**

Platform	Power (W)	# Layers	# Neurons per layer	# Parameters
Microcontroller	0.017	10	11	1265
Embedded Processor	5	100	152	2 311 160
FPGA-SoC (RTL)	2.1	100	272	7 399 760
FPGA-SoC (HLS)	2.1	100	138	1 905 090
FPGA-SoC (SDSoC)	2.1	100	43	185 115
Embedded GPU	7.5	100	2048	419 440 640

#### D. COST

Cost is an important decision factor that will determine in many applications the technology to be deployed. Nowadays, the cheapest technology available are microprocessor, existing a wide range of families and manufacturers available in a range of prices starting in just few cents. However, as it has been explained, their edge computing capabilities are limited. In recent years, proliferation of embedded CPUs such as Raspberry Pi and others has made possible the implementation of inexpensive stand-alone solutions within the range of tens of dollars, making them an excellent option for the low-mid cost range or large deployments. Finally, when high computing capabilities are required, both GPUs and FPGAs are required. Despite the high computing capabilities and parallel processing of FPGAs, advances in microelectronics combined with the optimized GPU architectures have made the latter option to have the best performance – cost / energy ratio.

#### E. COMPUTING IN THE BLINK OF AN EYE

A common feature in most edge devices is their short response times. Many of these devices are used to interact with human users. To provide users with the required performance, the possibility of human-machine cooperation or a pleasant user experience, these devices must be able to response within a short time interval that is not recognizable by user. To illustrate the possibilities of each platform for real-time human-machine, Table 8 presents the largest neural network that can be evaluated in the approximate time of the eye blinking (100 ms), including also the necessary power. The size of the neural network with three inputs and two outputs can be measured by the number of layers and neurons in each layer, or by the total amount of parameters that define the neural network. All the information can be seen in Table 8, which shows that embedded GPUs can deal with the largest networks.

#### VI. CONCLUSION

Rapid developments of the internet of things, have made sensing, communication and computation ubiquitous. These technologies are increasingly complex and, therefore, require more automation and intelligent decision-making.

When interacting with humans, low latency becomes essential, being the focus of this work.

In this paper, some the most relevant technologies to implement edge computing systems available to designers have been reviewed and discussed, including microprocessors, embedded processors, FPGAs, and embedded GPUs. In order to compare and discuss these technologies, a relevant deep neural network case of study has been selected and implemented in the different platforms and development tools.

The obtained results for each platform have been compared in terms of processing power, energy, latency, design complexity, and cost. Nowadays, microprocessors offer the most affordable solution for low-complexity and low-cost systems, whereas embedded processors are excellent alternatives when additional processing power is required. Modern FPGAs also offer high performance, at the cost of additional complexity. This can be solved by using HLS or SDSoC design methodologies, at the cost of reducing the system performance. Finally, modern GPU architectures offer not only the best computation capabilities but also the best computation-energy ratio, being the hardware platform of choice for highly demanding computing edge systems.

#### REFERENCES

- [1] E. Baccarelli, P. G. V. Naranjo, M. Scarpiniti, M. Shojafar, and J. H. Abawajy, "Fog of everything: Energy-efficient networked computing architectures, research challenges, and a case study," *IEEE Access*, vol. 5, pp. 9882–9910, 2017.
- [2] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, p. 436, May 2015.
- [3] M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 1, pp. 30–39, Jan. 2017.
- [4] F. Lamnabhi-Lagarrigue, "Systems & control for the future of humanity, research agenda: Current and future roles, impact and grand challenges," *Annu. Rev. Control*, vol. 43, pp. 1–64, Jan. 2017.
- [5] W. Shi and S. Dustdar, "The promise of edge computing," *Computer*, vol. 49, no. 5, pp. 78–81, May 2016.
- [6] W. Yu, F. Liang, X. He, W. G. Hatcher, C. Lu, J. Lin, and X. Yang, "A survey on the edge computing for the Internet of Things," *IEEE Access*, vol. 6, pp. 6900–6919, 2018.
- [7] R. Lu, K. Heung, A. H. Lashkari, and A. A. Ghorbani, "A lightweight privacy-preserving data aggregation scheme for fog computing-enhanced IoT," *IEEE Access*, vol. 5, pp. 3302–3312, 2017.
- [8] S.-J. Wu, N. Gebrael, M. A. Lawley, and Y. Yih, "A neural network integrated decision support system for condition-based optimal predictive maintenance policy," *IEEE Trans. Syst., Man, Cybern., A, Syst. Humans*, vol. 37, no. 2, pp. 226–236, Mar. 2007.
- [9] B. Karg and S. Lucia, "Deep learning-based embedded mixed-integer model predictive control," in *Proc. Eur. Control Conf. (ECC)*, Jun. 2018, pp. 2075–2080.
- [10] R. Feraud, O. J. Bernier, J.-E. Viallet, and M. Collobert, "A fast and accurate face detector based on neural networks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 23, no. 1, pp. 42–53, Jan. 2001.
- [11] J. E. Stone, D. Gohara, and G. Shi, "OpenCL: A parallel programming standard for heterogeneous computing systems," *Comput. Sci. Eng.*, vol. 12, no. 3, pp. 66–73, May 2010.
- [12] S. Mittal, "A survey on optimized implementation of deep learning models on the NVIDIA Jetson platform," *J. Syst. Archit.*, vol. 97, pp. 428–442, Jan. 2019.
- [13] J. Gu, Y. Liu, Y. Gao, and M. Zhu, "OpenCL caffe: Accelerating and enabling a cross platform machine learning framework," in *Proc. ACM Int. Conf.*, vols. 19–21, 2016, pp. 1–5.
- [14] T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, and Z. Zhang, "MXNet: A flexible and efficient machine learning library for heterogeneous distributed systems," 2015, *arXiv:1512.01274*. [Online]. Available: <http://arxiv.org/abs/1512.01274>

- [15] X. Zhang, Y. Wang, and W. Shi, "PCamp: Performance comparison of machine learning packages on the edges," in *Proc. USENIX Workshop Hot Topics Edge Comput. (HotEdge)*, 2018, vol. 2, no. 1, pp. 1–6.
- [16] M. Papadonikolakis, C.-S. Bouganis, and G. Constantinides, "Performance comparison of GPU and FPGA architectures for the SVM training problem," in *Proc. Int. Conf. Field-Program. Technol.*, Dec. 2009, pp. 388–391.
- [17] T. Hao *et al.*, "Edge AIBench: Towards comprehensive end-to-end edge computing benchmarking," in *Benchmarking, Measuring, and Optimizing* (Lecture Notes in Computer Science), vol. 11459, C. Zheng and J. Zhan, Eds. Cham, Switzerland: Springer, 2019. [Online]. Available: [https://link.springer.com/chapter/10.1007/978-3-030-32813-9\\_3#citeas](https://link.springer.com/chapter/10.1007/978-3-030-32813-9_3#citeas)
- [18] C.-H. Hong and B. Varghese, "Resource management in Fog/Edge computing," *ACM Comput. Surv.*, vol. 52, no. 5, pp. 1–37, Sep. 2019.
- [19] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," in *Proc. IEEE Int. Symp. Workload Characterization (IISWC)*, Oct. 2009, pp. 44–54.
- [20] M. Gusev and S. Dustdar, "Going back to the roots—The evolution of edge computing, an IoT perspective," *IEEE Internet Comput.*, vol. 22, no. 2, pp. 5–15, Mar. 2018.
- [21] D. Georgakopoulos, P. P. Jayaraman, M. Frazia, M. Villari, and R. Ranjan, "Internet of Things and edge cloud computing roadmap for manufacturing," *IEEE Cloud Comput.*, vol. 3, no. 4, pp. 66–73, Jul. 2016.
- [22] K. Zhang, Y. Mao, S. Leng, Q. Zhao, L. Li, X. Peng, L. Pan, S. Maharjan, and Y. Zhang, "Energy-efficient offloading for mobile edge computing in 5G heterogeneous networks," *IEEE Access*, vol. 4, pp. 5896–5907, 2016.
- [23] Cisco, "Cisco visual networking index: Forecast and trends, 2017–2022," Cisco, San Jose, CA, USA, White Paper, 2019.
- [24] G. Serale, M. Fiorentini, A. Capozzoli, D. Bernardini, and A. Bemporad, "Model predictive control (MPC) for enhancing building and HVAC system energy efficiency: Problem formulation, applications and opportunities," *Energies*, vol. 11, no. 3, p. 631, Mar. 2018.
- [25] S. Lucia, D. Navarro, O. Lucia, P. Zometa, and R. Findeisen, "Optimized FPGA implementation of model predictive control for embedded systems using high-level synthesis tool," *IEEE Trans Ind. Informat.*, vol. 14, no. 1, pp. 137–145, Jan. 2018.
- [26] M. Mukherjee, R. Matam, L. Shu, L. Maglaras, M. A. Ferrag, N. Choudhury, and V. Kumar, "Security and privacy in fog computing: Challenges," *IEEE Access*, vol. 5, pp. 19293–19304, 2017.
- [27] J. A. Stankovic, "Research directions for the Internet of Things," *IEEE Internet Things J.*, vol. 1, no. 1, pp. 3–9, Feb. 2014.
- [28] A.-R. Sadeghi, C. Wachsmann, and M. Waidner, "Security and privacy challenges in industrial Internet of Things," in *Proc. 52nd ACM/EDAC/IEEE Des. Autom. Conf. (DAC)*, Jun. 2015, pp. 1–6.
- [29] K. Irion, "Government cloud computing and national data sovereignty," *Policy Internet*, vol. 4, nos. 3–4, pp. 40–71, Jan. 2013.
- [30] X. Meng, W. Wang, and Z. Zhang, "Delay-constrained hybrid computation offloading with cloud and fog computing," *IEEE Access*, vol. 5, pp. 21355–21367, 2017.
- [31] S. Lucia, D. Navarro, B. Karg, H. Sarnago, and O. Lucia, "Deep learning-based model predictive control for resonant power converters," *IEEE Trans Ind. Informat.*, to be published.
- [32] B. Karg and S. Lucia, "Efficient representation and approximation of model predictive control laws via deep learning," 2018, *arXiv:1806.10644*. [Online]. Available: <http://arxiv.org/abs/1806.10644>
- [33] A. Pullini, D. Rossi, I. Loi, A. Di Mauro, and L. Benini, "Mr. Wolf: A 1 GFLOP/s energy-proportional parallel ultra low power SoC for IOT edge processing," in *Proc. IEEE 44th Eur. Solid State Circuits Conf. (ESSCIRC)*, Sep. 2018, pp. 274–277.
- [34] Y. Zhao, Q. Yu, X. Zhou, X. Zhou, X. Li, and C. Wang, "PIE: A pipeline energy-efficient accelerator for inference process in deep neural networks," in *Proc. IEEE 22nd Int. Conf. Parallel Distrib. Syst. (ICPADS)*, Dec. 2016, pp. 1067–1074.
- [35] C. Huang, S. Ni, and G. Chen, "A layer-based structured design of CNN on FPGA," in *Proc. IEEE 12th Int. Conf. ASIC (ASICON)*, Oct. 2017, pp. 1037–1040.
- [36] O. Jimenez, O. Lucia, I. Urriza, L. A. Barragan, D. Navarro, and V. Dinavahi, "Implementation of an FPGA-based online hardware-in-the-loop emulator using high-level synthesis tools for resonant power converters applied to induction heating appliances," *IEEE Trans. Ind. Electron.*, vol. 62, no. 4, pp. 2206–2214, Apr. 2015.
- [37] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing FPGA-based accelerator design for deep convolutional neural networks," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays (FPGA)*, Monterey, CA, USA, 2015, pp. 161–170.



applications, and high-performance computing.



Scholar with the Center for Power Electronics Systems (CPES), Virginia Tech, in 2009 and 2012. He has also been with TU Berlin, Germany, since 2019. His main research interests include resonant power conversion, wide-bandgap devices, and digital control, mainly applied to contactless energy transfer, induction heating, electric vehicles, and biomedical applications. In these topics, he has published more than 75 international journal articles and 125 conference papers. He has filed more than 40 international patents.

Dr. Lucía is a member of the Aragon Institute for Engineering Research (I3A) and an Active Member of the Power Electronics (PELS) and the Industrial Electronics (IES) societies. He is currently an Associate Editor of the IEEE TRANSACTIONS ON INDUSTRIAL ELECTRONICS, the IEEE TRANSACTIONS ON POWER ELECTRONICS, and the IEEE OPEN JOURNAL OF THE INDUSTRIAL ELECTRONICS SOCIETY.



of Internet of Things for Smart Buildings, TU Berlin, and the Einstein Center Digital Future. His research efforts focus on decision-making under uncertainty, distributed control, and embedded optimization using micro-controllers, and FPGAs in the framework of the Internet of Things. His applications of interest include smart buildings and energy systems. He is an Active Member of the IEEE CSS Society. He is an Associate Editor of the *Journal of Process Control*.

• • •