IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received June 27, 2019, accepted July 17, 2019, date of publication July 26, 2019, date of current version August 12, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2931392

Very Low Power Neural Network FPGA Accelerators for Tag-Less Remote Person Identification Using Capacitive Sensors

MARWEN ROUKHAMI[1], MIHAI TEODOR LAZARESCU[2], (Senior Member, IEEE), FRANCESCO GREGORETTI[2], (Member, IEEE), YOUNES LAHBIB[3], AND ABDELKADER MAMI[1]

[1]UR-LAPER, Faculty of Sciences of Tunis, University of Tunis El-Manar, Tunis 2092, Tunisia
[2]Department of Electronics and Telecommunications, Politecnico di Torino, 10129 Torin, Italy
[3]LR-EμE, ENICarthage, University of Carthage, Tunis 2035, Tunisia

Corresponding author: Marwen Roukhami (marwen.roukhami@fst.utm.tn)

ABSTRACT Human detection, identification, and monitoring are essential for many applications aiming to make smarter the indoor environments, where most people spend much of their time (like home, office, transportation, or public spaces). The capacitive sensors can meet stringent privacy, power, cost, and unobtrusiveness requirements, they do not rely on wearables or specific human interactions, but they may need significant on-board data processing to increase their performance. We comparatively analyze in terms of overall processing time and energy several data processing implementations of multilayer perceptron neural networks (NNs) on board capacitive sensors. The NN architecture, optimized using augmented experimental data, consists of six 17-bit inputs, two hidden layers with eight neurons each, and one four-bit output. For the software (SW) NN implementation, we use two STMicroelectronics STM32 low-power ARM microcontrollers (MCUs): one MCU optimized for power and one for performance. For hardware (HW) implementations, we use four ultralow-power field-programmable gate arrays (FPGAs), with different sizes, dedicated computation blocks, and data communication interfaces (one FPGA from the Lattice iCE40 family and three FPGAs from the Microsemi IGLOO family). Our shortest SW implementation latency is 54.4 μs and the lowest energy per inference is 990 nJ, while the shortest HW implementation latency is 1.99 μs and the lowest energy is 39 nJ (including the data transfer between MCU and FPGA). The FPGAs active power ranges between 6.24 and 34.7 mW, while their static power is between 79 and 277 μW. They compare very favorably with the static power consumption of Xilinx and Altera low-power device families, which is around 40 mW. The experimental results show that NN inferences offloaded to external FPGAs have lower latency and energy than SW ones (even when using HW multipliers), and the FPGAs with dedicated computational blocks (multiply-accumulate) perform best.

INDEX TERMS Indoor person identification, capacitive sensing, neural networks, hardware design, ultra-low power FPGAs, hardware acceleration, embedded design optimization.

I. INTRODUCTION

People spend much time indoors (at home, in the office, in shops, transportation, and other spaces), yet those environments are hardly aware of their occupants, and even less of their identity, activities, intentions, or emotions. Most of the time, the occupants still need to actively engage the environment for services or customization, e.g., by tuning knobs, pressing switches, wearing recognizable devices, or voice commands.

Various indoor security and monitoring activities, such as building and home automation, security, elderly care, patient monitoring, assisted living, need real-time privacy-observant tag- and interaction-less low-power low-cost means for indoor human sensing, localization, identification, and inference of activity, intentions or emotions [1]. Technology advances continuously reduce the footprint and increase the availability, quality of service, and affordability of

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

embedding adaptive intelligence in environments, objects, and appliances [2], but effective indoor human sensing is still lagging.

Various techniques have been proposed for indoor human localization and identification. Video-, accelerometer-, and image-based face and gait recognition are among the most explored. Ubiquitous Wi-Fi network fields have also been used, but for adequate quality they may incur higher deployment and equipment cost [3]. Other investigated sensing techniques include pyroelectric infrared (PIR) [4], [5], ultrasonic [6], ultra-wideband (UWB) [7], footstep-induced structural vibrations [8], or electric potential [9].

Capacitive sensing of conductive and dielectric properties of human body was also used for indoor person sensing [10], [11], identification [1], [12], and localization [13]–[16]. Machine learning algorithms, and especially neural networks (NNs) trained on the abstract signatures of targets can be used to improve sensor performance [17], [18].

It can be difficult to efficiently implement NNs on low-resource low-power and low-cost embedded systems. In this article, we train and optimize a NN for automatic indoor human identification using augmented experimental measurements from our previous work [1], then we comparatively analyze several NN implementation and acceleration options on low-resource very low-power embedded systems, with and without hardware (HW) acceleration.

The rest of the article is organized as follows. Section II discusses related work. Section III briefly presents the measurement technique that we used to acquire the experimental data in our previous work [1]. Section IV briefly presents the NN architecture and optimization. Section V discusses several NN implementation and acceleration options, including communication. Section VI concludes the article and discusses future work.

II. RELATED WORK

We comparatively analyze several options for NN inference acceleration on low-power MCUs and ultralow-power field-programmable gate arrays (FPGAs) for automatic indoor human identification using capacitive sensors. Although we briefly describe the operation of the capacitive sensor used to collect the experimental data, and NN training and architectural optimization, we focus on the performance of NN implementation, mainly inference latency and energy.

NN inference requirements for computation, memory, and energy may often restrict their applicability to resource-constrained low-power real-time embedded systems [19]. Speed and energy are singled out by Guo *et al.* [20] as basic metrics for efficient NN implementations. While graphical processing units (GPUs) provide high performance computation at high energy cost, FPGAs can effectively accelerate NNs for significantly less energy.

FPGA NN implementations often finely tune data paths and operators, along parallel execution. Braga *et al.* [21] comparatively analyze performance, resource utilization, power consumption, and accuracy of NN HW and software (SW) implementations on Xilinx FPGAs using MicroBlaze soft processor. Blaiech *et al.* [22] propose a method to improve NN HW implementations timing and resource requirements, while Bahoura and Park [23] propose pipelined HW implementations of non-linear NN to reduce critical paths. Zhai *et al.* [24] present an accurate multi-layer perceptron NN for gas classification implemented on Xilinx Zynq FPGAs with fixed-point arithmetic. Latino *et al.* [25] propose a general NN architecture optimized for intelligent position sensor-based systems implemented on Xilinx FPGAs. Hariprasath and Prabakar [26] propose a pipelined NN architecture for multi-modal biometric pattern recognition implemented on Xilinx Virtex-4 FPGA. In [27], Bettoni *et al.* accelerate a CNN for image recognition up to 16 times than SoC GPU using a Xilinx Zynq FPGA. Indoor face recognition for identification and localization often use FPGA NN implementations [28]. Dawwd and Mahmood propose in [29] an efficient Xilinx Spartan-3E FPGA CNN implementation for video-based face recognition.

Gabriel *et al.* [30] survey the use of Xilinx and Altera FPGA families to optimize power consumption and processing time in low-power sensor systems. Although most NN HW implementations target resourceful Xilinx and Altera FPGA families [21], the static and dynamic power consumption of the most low-energy devices is often too high for long-lasting battery-powered embedded sensors [34].

In this article, we explore several NN optimizations on ultralow-power FPGAs that are specifically designed for low duty cycle and tight energy constrained applications. We comparatively analyze the performance of several SW and HW embedded implementations of a NN for person classification based on capacitive sensor measurements. Specifically, our main contributions are

- NN architectural exploration and optimization using experimental data from capacitive sensors for person identification;
- SW NN implementation on two low-power MCUs, one tuned for low power and one for performance;
- HW NN implementation on four ultralow-power FPGAs with different dedicated blocks and size;
- Characterization of MCU–FPGA communication in terms of time and energy;
- Comparative analysis of overall inference latency and energy of all NN implementation options.

III. CAPACITIVE SENSOR FOR CONTACTLESS PERSON IDENTIFICATION

Capacitive sensors are widely used mostly for ranges shorter than sensor diagonal dimension. In our previous research [14], we have explored the performance of capacitive sensors in load mode for deviceless indoor human localization at much longer ranges, 10–20 times sensor diagonal dimension.

Figure 1 shows the operating principle of capacitive sensors in load mode. Sensor plate capacitance is made of capacitances with the surrounding environment (e.g., Cpg),

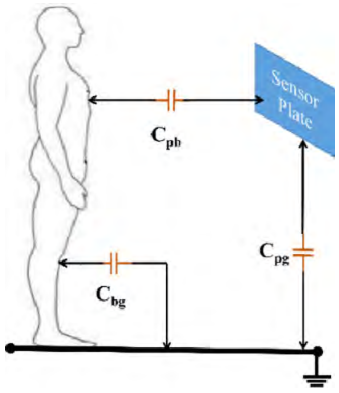

FIGURE 1. Operation principle of load-mode capacitive sensors. The sensor plate forms capacitances with the surrounding environment (Cpg) and the body of the person (Cpb). The latter changes with the distance between the person and sensor, determining changes in the overall capacitance of the sensor that can be measured.

including the capacitance Cpb, with the body of the person. The latter changes with the distance between person and sensor and determines measurable plate capacitance changes.

Plate capacitance can be measured through its reactance at a known excitation frequency. Furthermore, if we measure sensor capacitance at multiple frequencies at the same sensor–person distance, then capacitance variation with the measurement frequency is mostly due to differences in person body composition [32]. Hence, if we associate specific capacitance-frequency patterns to specific persons, we can then use the patterns to identify the persons [1].

A. PERSON IDENTIFICATION USING CAPACITIVE SENSORS

Figure 2 shows the circuit we used in our previous work [1] to measure sensor plate capacitance at different frequencies.

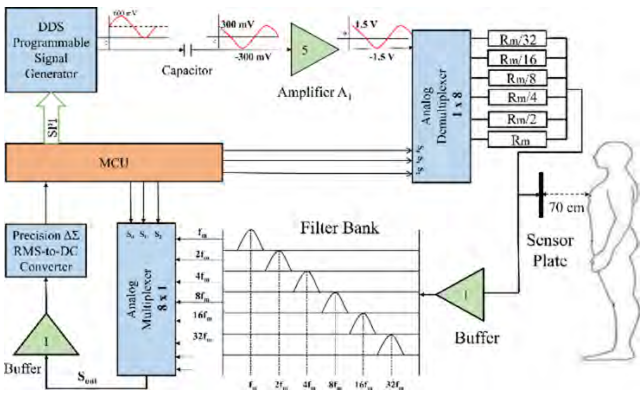

FIGURE 2. Block schematic of the person identification capacitive sensor based on the measurement of the reactance of the sensor plate capacitance with the body of a person at multiple excitation frequencies. The variation of the capacitance with the frequency is specific to body composition and can identify the person.

Measurement frequency is applied to sensor plate (of capacitance C) through a resistor R. An MCU controls both the measurement frequency (using a sine wave generator) and R (selected with an analog multiplexer from predefined values) which tunes the idle RC low-pass filter cutoff frequency (no person in range) to the input frequency. C changes when a person is in range and detunes the RC filter. This changes its output amplitude, which we then use as proxy for the C value. To measure the amplitude, we first denoise the RC filter output (using a tunable band-pass filter), and then convert it to DC using an RMS-to-DC converter.

In the following we use our experimental measurements from [1], which were made for four different persons at six frequencies spaced by one octave: 5 kHz, 10 kHz, 20 kHz, 40 kHz, 80 kHz, and 160 kHz.

B. EXPERIMENTAL DATA

In our previous work [1], we used the sensor shown in Figure 2 to indirectly measure the capacitance at each of the six excitation frequencies for ten times for each of the four persons. Figure 3 shows the RC filter attenuation of the carrier frequencies.

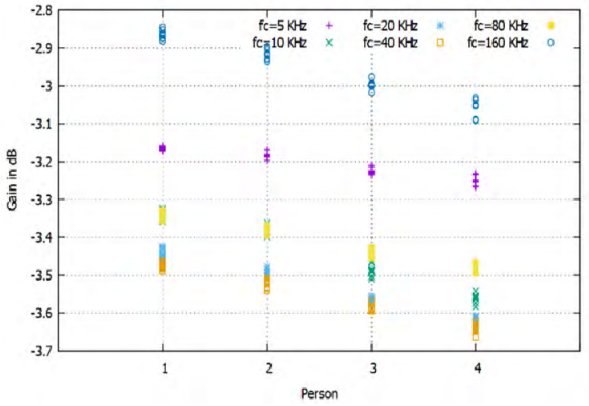

FIGURE 3. Experimental results of ten measurements of the attenuation of the low-pass RC filter (plate capacitance C and series resistor R) for each of the four persons at the six chosen frequencies.

The RC filter attenuations deviate from the expected −3 dB because of the presence and the specific body composition of the person in front of the sensor. We can see that each person has a specific ratio between the RC filter attenuations at each of the six measurement frequencies. We will use these attenuation patterns to identify the person.

IV. ARTIFICIAL NEURAL NETWORK OPTIMIZATION

Our previous comparative analysis of machine learning (ML) classification algorithms for indoor person localization using capacitive sensor data exposed significant performance variance [18]. Also, most of the best performing ML algorithms may often require excessive resources from low-power low-cost embedded target microprocessors.

In this work, we focus on the comparative analysis of several implementation and acceleration options for artificial NN [33] used to infer person identity from capacitive sensor measurements. For this purpose, we consider a system made of a low-power low-cost MCU that controls sensor operation, handles data acquisition, and communication.

We briefly introduce first how we trained and optimized a typical multilayer perceptron NN, for which we will then explore several implementation and acceleration options.

A. DATA SET GENERATION AND PREPARATION

We design and train the NN using augmented experimental data, because we had only ten experimental sets per person, which are insufficient for effective NN training.

We augment the data using Matlab R2017a following the steps shown in Figure 4. First, we add Gaussian noise and then we normalize the data values to improve NN training convergence. We split the resulting data into three sub-sets: one for NN training, one for NN training validation and optimization, and one for testing the NN performance.

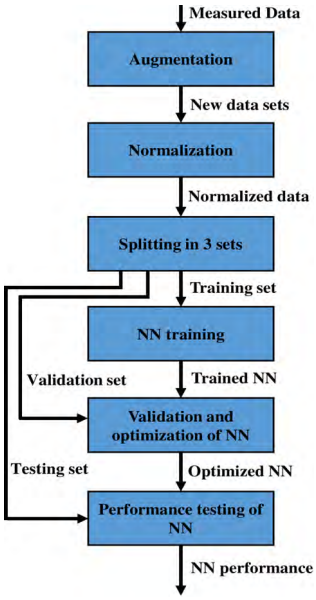

FIGURE 4. Experimental data preparation to be used for neural network training, validation and optimization, and performance testing.

1) DATA SET AUGMENTATION

Neural networks require many training data samples to learn the significant features and to avoid overfitting the specific data sets used for training. Data augmentation is commonly used to generate new data sets that preserve the key characteristics of the original sets [34]. We use 24 Gaussian noise generators, each tuned to the average and standard deviation of the experimental data for a measuring point (defined by a specific person ID and measurement frequency, see Figure 3). With the parameters shown in Table 1, we generate 10,000 labelled six-tuples for each of the four persons (40,000 tuples in total).

2) DATA SET NORMALIZATION

Input data normalization can significantly improve the convergence rate of NN training [35], [36]. We set the minimum and maximum values of each six-tuple to zero and one, respectively, then we scale the tuple intermediate values accordingly, as shown in Figure 5. The normalization procedure is lightweight, suitable for real-time execution.

TABLE 1. Average values (AVG) and standard deviation (STD) of the ten measurement capacitance values for each person at each frequency.

		Measurement frequency (kHz)					
		5	10	20	40	80	160
Person 1	AVG	-3.17	-3.34	-3.45	-3.48	-3.34	-2.86
	STD	0.004	0.011	0.013	0.007	0.007	0.010
Person 2	AVG	-3.19	-3.38	-3.50	-3.52	-3.38	-2.92
	STD	0.007	0.009	0.010	0.009	0.007	0.011
Person 3	AVG	-3.23	-3.49	-3.57	-3.59	-3.44	-3.00
	STD	0.007	0.015	0.007	0.007	0.007	0.010
Person 4	AVG	-3.25	-3.56	-3.63	-3.64	-3.48	-3.06
	STD	0.012	0.011	0.011	0.011	0.008	0.019

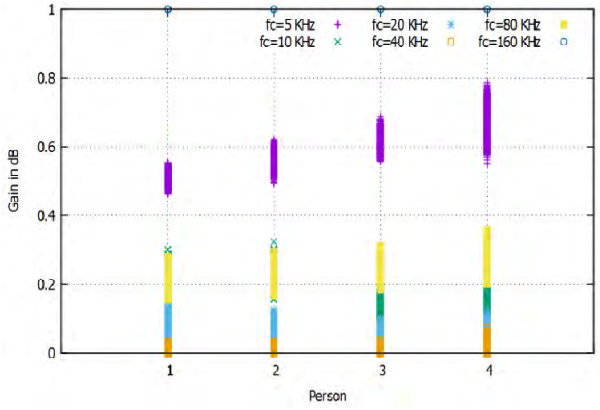

FIGURE 5. Normalized data set augmented to 40,000 tuples using Gaussian random number generators set to same average and standard deviation values as the experimental data.

3) DATA SET SPLIT

Typical data splits allocate 60–90% of tuples for the training set, and the rest evenly between validation and testing sets. We decided to split 70% for NN training, 15% for validation, and 15% for testing, which is adequate for tuning most NNs.

B. NEURAL NETWORK TRAINING AND OPTIMIZATION

We use a simple multilayer perceptron NN architecture, which is well suited for classification tasks. We set the NN inputs and outputs as shown in Figure 6 to fit the application: the six inputs are connected to the six sensor outputs, while the output encodes the inferred person ID.

According to data set split ratios, we use 28,000 six-tuples for NN training, 6,000 six-tuples for validation, and 6,000 six-tuples for NN testing. Although advanced NN optimization is outside the scope of this work, we present some basic

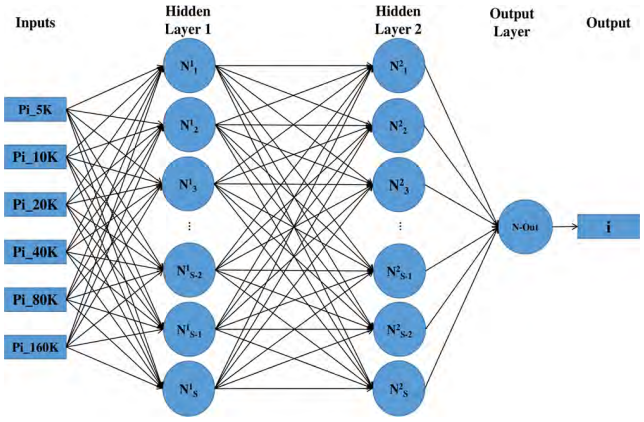

FIGURE 6. Feedforward neural network with six inputs, one output, and two hidden layers, each of variable number of neurons.

NN hyperparameter optimization, specifically the number of hidden layers, neurons per layer, and neuron activation function. We restrict the search to NNs with the same number of neurons on each hidden layer. We optimize and test each architecture ten times and report the averages.

First, we establish a NN performance baseline for each NN architecture using the non-linear logistic sigmoid activation function (LogSig) for all hidden neurons. As shown in Table 2, inference accuracy slightly improves when we use more hidden layers or more neurons on each hidden layer.

TABLE 2. Average identification error for different neural network architectures using the logistic sigmoid (LogSig) activation function.

		Number of hidden Layers			
		1	2	3	4
Number of neurons in each hidden layer	4	5.23 %	5 %	4.8 %	4.88 %
	8	4.93 %	4.9 %	4.9 %	4.91 %
	16	4.98 %	4.89 %	4.93 %	4.89 %
	32	4.94 %	4.85 %	4.86 %	4.85 %

Because the LogSig activation function requires significant resources which are hard to find on resource-constrained devices [37], [38], we replace it with the rectified linear unit (ReLU). We note that the NN accuracy degrades only slightly (see Table 3), which is acceptable for our purpose (to compare different NN implementations). Hence, from now on we will use only the ReLU function, while the output neuron always uses a linear identity activation.

We also note that NN accuracy varies only slightly across all explored NN architectures, so there is little to gain optimizing it. Hence, we select a NN architecture with two hidden layers of eight neurons each (see Table 4). It also seems adequate to load the target devices for our analysis. Note that the first hidden layer requires only six multiplications because it has only six inputs, each connected to one of the six sensor outputs, as shown in Figure 6.

TABLE 3. Average identification error for different neural network architectures using the rectified linear unit (ReLU) activation function.

		Number of hidden Layers			
		1	2	3	4
Number of neurons in each hidden layer	4	5.42 %	5.37 %	5.64 %	5.58 %
	8	5.17 %	5.06 %	4.93 %	4.94 %
	16	5 %	4.93 %	4.96 %	5 %
	32	4.9 %	4.86 %	4.96 %	4.9 %

TABLE 4. Parameters of the neural network selected for experiments.

Layers	Neurons per layer	Multiplications per neuron	Activation function
Hidden 1	8	6	ReLU
Hidden 2	8	8	ReLU
Output	1	8	Identity

V. NEURAL NETWORK IMPLEMENTATION

NNs can be implemented in SW (running on either general (CPU) or specialized (GPU) processors), or in HW (either reconfigurable (FPGA) or application-specific (ASIC)). FPGAs can bring significant speed and energy improvements for both high-end implementations [39]–[42] and for energy- and processing-constrained embedded devices [27], [43], [44], while preserving programmability. Since most NN computations are embarrassingly parallel, they can considerably benefit from FPGAs flexibility (e.g., resource allocation, scheduling, data flow, data width).

We comparatively analyze several HW and SW implementations of the selected NN architecture. HW implementations exploit parallelization opportunities provided by different ultralow-power FPGAs, while the SW implementation uses resources of different low-power microprocessors. In our analysis, we are especially interested in trade-offs between the latency and energy consumption, including the MCU–FPGA communication overhead.

A. SOFTWARE IMPLEMENTATION ON DIFFERENT ARM CORTEX M3 MICROCONTROLLERS

We implement the NN in SW on two ARM Cortex M3 MCUs in STmicroelectronics STM32 family, one optimized for low power, STM32L152RE [45], and another optimized for performance, STM32F103RB [46]. First, we want to verify that the NN processing on the embedded devices matches the Matlab reference model. Second, we want to compare the NN performance on MCU families optimized for different tasks. Third, we want to establish a performance baseline for the HW (FPGA) implementations.

Both MCUs use an ARM Cortex M3 core, include HW multipliers, and their development boards allow to measure current consumption during program execution. We use the Keil uVision® toolset[1] to compile, debug, trace the execution, and measure the performance.

NN SW implementation calculates the forward propagation (inference) for the NN shown in Figure 6, with six inputs, two hidden layers of eight neuron each and ReLU activation functions, and one output neuron with linear identity activation function.

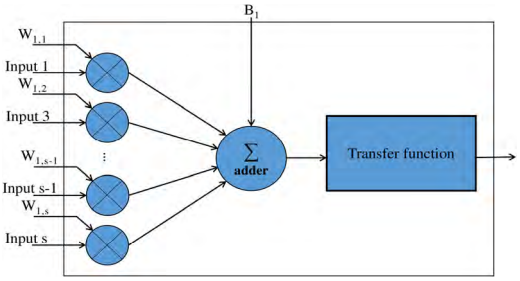

FIGURE 7. Data flow within a neuron receives the input signals Input $_n$, mutiplies each by a neuron- and input-specific weight W $_{l,n}$, the multiplication results are then added and passed through an activation (transfer) function, which calculates the output of the neuron.

TABLE 5. Comparison of neural network software implementation performance on two microcontrollers in different operation conditions.

MCU device / MCU parameter	STM32L152RE / STM32F103RB		
Clock frequency (MHz)	8 / 8	16 / 16	32 / 64
FLASH latency (wait states)	0 / 0	0 / 0	1 / 2
Clock cycles for NN inference	1474 / 1474	1474 / 1474	1740 / 1831
NN inference time (µs)	184 / 184	92.1 / 92.1	54.4 / 28.6
Current (mA)	1.85 / 4.00	3.60 / 6.80	8.60 / 23.9
Voltage (V)	3.0 / 3.3	3.0 / 3.3	3.3 / 3.3
Power (mW)	5.55 / 13.2	10.8 / 22.4	28.4 / 78.9
Energy (µJ)	1.02 / 2.43	0.99 / 2.06	1.54 / 2.26

Figure 7 shows the data flow of one neuron. SW implementation processes in sequence each input of each neuron in the NN, starting with the first hidden layer, then the second, and finally the output neuron, as follows

Table 5 shows the NN SW implementation performance for the two MCU types, in different operation conditions.

[1]Keil uVision®toolset: http://www.keil.com/mdk5/uvision/

Algorithm 1

//First hidden layer
For $i =$ *All neurons on first hidden layer* **do**
  For $j =$ *All inputs* **do**
    $Product \leftarrow input\_data\ [j] * weights\_0\ [i][j]$
    $Sum \leftarrow Sum + Product$
  End for
  $Hidden\_L0\ [i] \leftarrow Sum + b0\ [i]$
  If $Hidden\_L0\ [i] < 0$ **then**
    $Hidden\_L0\ [i] \leftarrow 0$
  End if
  $Sum \leftarrow 0$
End for
//Second hidden layer
For $i =$ *All neurons on second hidden layer* **do**
  For $j =$ *All outputs of first hidden layer* **do**
    $Product \leftarrow Hidden\_L0\ [j] * weights\_1\ [i][j]$
    $Sum \leftarrow Sum + Product$
  End for
  $Hidden\_L1\ [i] \leftarrow Sum + b1[i]$
  If $Hidden\_L1\ [i] < 0$ **then**
  $Hidden\_L1\ [i] \leftarrow 0$
  End if
  $Sum \leftarrow 0$
End for
//Output layer
For $i =$ *All outputs of second hidden layer* **do**
  $Product \leftarrow Hidden\_L1\ [i] * weights\_2$[i]
  $Sum \leftarrow Sum + Product$
End for
$Person\_ID \leftarrow sum + b2$

The low-power MCU (STM32L152RE) achieves the minimum energy per NN inference (0.99 μJ) at 16 MHz clock frequency. Inference energy at 8 MHz clock is just 3% higher (1.02 μJ), most likely because the longer inference time increases the static power contribution. At 32 MHz clock, the NN inference energy is 56% higher (1.54 μJ) because both the active power and the clock cycles for NN inference increase: Active power is 2.6 times higher (from 10.8 mW to 28.4 mW) and NN inference clock cycles increase 18% (from 1474 to 1740 cycles) due to the higher FLASH latency (1 wait state).

The performance-optimized MCU (STM32F103RB) consumes more current and operates at higher supply voltage (3.3 V instead of 3.0–3.3 V for STM32L152RE) and consumes more energy per NN inference in all operation conditions.

B. HARDWARE IMPLEMENTATION ON FPGAS

Typical multilayer perceptron (feedforward, Figure 6) NNs, have one or more interconnected layers of neurons. The input layer receives the input values which are incrementally processed by the neurons on the hidden layers as they propagate

TABLE 6. Comparison of computation resources (lookup tables (LUT) and digital signal processors (DSP)) and static power (Pstatic) consumption for some low-power FPGA families from several producers.

Producer / Family	Device	LUTs	DSPs	Pstatic (mW)
Lattice ICE Ultraplus	ICE40UP5K	5280	8	0.28
Xilinx Spartan 3	XC3S200	4320	12	41
	XC3S200A	4032	16	43
Xilinx Artix 7	XC7A100T	162240	240	41
Altera Cyclone II	EP2C8	8256	18	40
Microsemi IGLOO	AGLN250	6144	N/A	0.08
	M1AGL600	13824	N/A	0.13
	M1AGL1000	24576	N/A	0.21

towards the output(s). Figure 7 shows the processing done by each neuron. It multiplies each input by an associated weight, then it passes the sum of the multiplication results to the activation (transfer) function, which calculates the neuron output.

Table 6 shows the main FPGA resources and static power consumption for several low- and ultralow-power FPGA families. We note that FPGAs from Lattice and Microsemi consume much less static power compared to those form Xilinx or Altera. Also, the Microsemi FPGAs consume less static power than the Lattice, but the dedicated DSP (MAC) blocks of the latter may implement calculations more efficiently (we will analyze their effects later).

NNs usually have many neurons that require significant computation for inference [47], and need design tradeoffs between processing parallelization and pipelining to efficiently use the limited resources of ultralow-power FPGAs. Our NN has six inputs, two hidden layers of eight neurons each, and one output neuron (see Table 4). First hidden layer needs six multiplications, the second and the output layers need eight. The activation functions (ReLU and identity) need none.

In the following, we will consider for NN inference acceleration only the four ultralow-power FPGAs from Table 6, which we will designate as: "LAT" the Lattice iCE40 UP5K device, "MS-S" the Microsemi IGLOO AGLN250 device, "MS-M" the Microsemi IGLOO M1AGL600 device, and "MS-L" the Microsemi IGLOO M1AGL1000 device.

We program the FPGAs using manually written functional VHDL code compiled with the programming tools from the FPGA producers.

1) IMPLEMENTATION ON LATTICE ICE40 FPGA

We use the largest device in the iCE40 UltraPlus ultralow-power FPGA family from Lattice Semiconductor, the UP5K. It is the only one with eight 16-bit × 16-bit multiply and 32-bit accumulator (MAC) blocks. It also has 5280 look-up

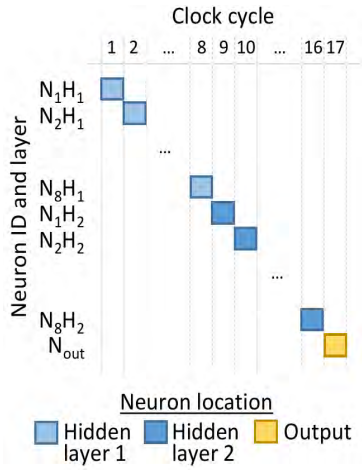

FIGURE 8. Scheduling of neural network processing on Lattice iCE40 UP5K and Microsemi IGLOO M1AGL600 FPGAs. Neurons are processed sequentially, starting with the neurons on the first hidden layer (N_1H_1 – N_8H_1), then the neurons on the second hidden layer (N_1H_2 – N_8H_2), and the output neuron (N_{out}).

tables (LUTs), 1024 kbit static random-access memory (SRAM), integrated serial peripheral interface (SPI) and inter-integrated circuit (I2C) core blocks for communication [48].

We implement a generic neuron using the eight MAC blocks to process in parallel the multiplication of the eight inputs and weights, and LUTs to implement the eight-input adder and the activation function (see Figure 7). We use this generic neuron to process sequentially the whole NN (see Figure 8), from first neuron on the first hidden layer, (N_1H_1 – N_8H_1), to last (eighth) neuron on last (second) hidden layer, N_1H_2 – N_8H_2). At the end, we process the output neuron, changing only the activation function from ReLU to linear identity.

This implementation uses all eight MAC blocks and 2047 of the 5280 LUTs (38.77%) available on the FPGA device.

We prefer this implementation to vectorization because it gives us more control on resource allocation and scheduling.

2) IMPLEMENTATION ON MICROSEMI IGLOO FPGAS

We select three devices in the IGLOO ultralow-power FPGA family from Microsemi, AGLN250, M1AGL600, M1AGL1000 [49]. AGLN250 has 6144 programmable logic VersaTiles (which are like Xilinx configurable logic blocks, Lattice lookup tables, or Altera logic elements), M1AGL600 has 13824 VersaTiles, and M1AGL1000 has 24576 VersaTiles. No IGLOO device has DSP blocks (Table 6).

On AGLN250 we can implement two multipliers and two eight-input adders (one for each activation function, ReLU and linear identity). Each multiplier requires 1071 VersaTiles, and control and other elements use 2747 VersaTiles (see Table 7).

With less computing resources available, we can support less parallel processing on AGLN250 than on Lattice

TABLE 7. Resources used to implement the neural network main computing elements on FPGA devices in the Microsemi IGLOO family.

Resources / IGLOO type	VersaTiles used			
	Multipliers	Control	Other	Total
AGLN250 (MS-S)	2142	126	2621	4889 (79.57%)
M1AGL600 (MS-M)	8568	64	1399	10031 (72.56%)
M1AGL1000 (MS-L)	17136	63	643	17842 (72.60%)

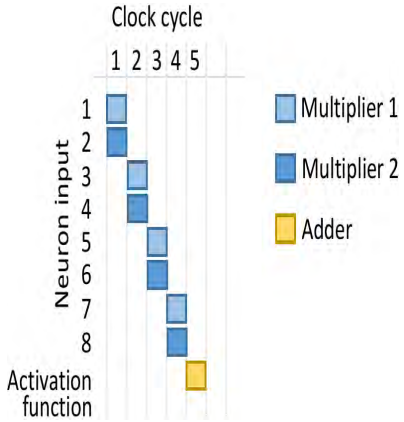

FIGURE 9. Scheduling of the main computation resources (two multipliers and one eight-input adder) in the processing of a neuron on the Microsemi IGLOO AGLN250 FPGA.

iCE40 UP5K. As shown in Figure 9, we can schedule up to two parallel multiplications. Hence, we need four clock cycles to process all eight neuron inputs, and a clock cycle to add the results using an eight-input adder and to calculate the neuron activation (with ReLU or linear identity functions, depending on neuron position in the NN).

On M1AGL600 we can implement eight multipliers, each using 1071 VersaTiles (see Table 7). Control and other elements require 1463 VersaTiles. This is less than the 2747 VersaTiles on AGLN250 mainly because higher parallelism simplifies most elements (e.g., state space and multiplexer sizes). Because we can implement on M1AGL600 the same computing resources as on Lattice iCE40 UP5K, we can use on both the same parallelism and processing schedule, as shown in Figure 8.

On M1AGL1000 we can implement 16 multipliers of 1071 VersaTiles each (see Table 7). Control and other elements take 706 VersaTiles, less than on M1AGL600 thanks to the higher parallelism which simplifies most control elements. With 16 multipliers and two adders, we can compute in parallel two neurons. As shown in Figure 10, we process the NN starting from first two neurons on first hidden layer, N_1H_1 and N_2H_1, and up to last two neurons on last hidden layer, N_7H_2 and N_8H_2, followed by the output neuron (neurons on first hidden layer run six parallel multiplications).

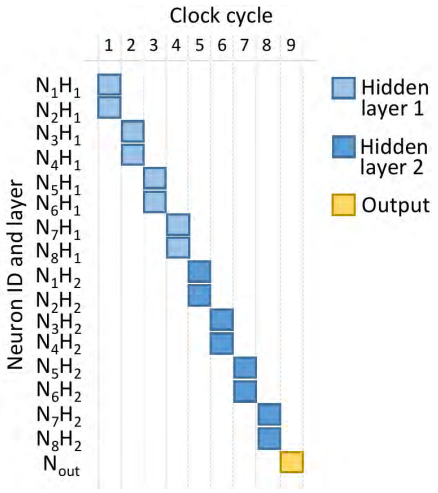

FIGURE 10. Scheduling of neural network processing on Microsemi IGLOO M1AGL1000 FPGA. Neurons are processed sequentially in pairs, starting with the first two neurons in the first hidden layer (N_1H_1, N_2H_1 – N_7H_1, N_8H_1), then the neurons in the second hidden layer (N_1H_2 , N_2 H_2 – N_7H_2, N_8H_2), and finally the output neuron (N_{out}).

3) HARDWARE IMPLEMENTATION RESULTS

Because of the limited available resources, the Microsemi IGLOO AGLN250 FPGA cannot process completely in parallel one neuron (see Figure 9). Hence, we must insert registers to hold all multiplication results stable for the eight-input adder (see Figure 7). The registers also split the longest (critical) combinatorial data path, which increases both the maximum clock frequency and the clock cycles needed for one NN inference.

All other FPGAs can perform all neuron multiplications in parallel and do not need registers after multipliers (see Figure 7). However, we implement on all FPGAs both architectures to compare closer the performance of the devices: with registers after neuron multipliers, or ''short critical path'' (SCP), and without registers after multipliers, or ''long critical path'' (LCP).

Table 8 shows the resource requested for the NN implementations on FPGAs for both architectures: SCP and LCP. On Lattice iCE40 UP5K, we use all eight MAC blocks to accelerate the neuron calculations, and about a third of the available logic blocks for other functions. Microsemi IGLOO FPGA family has no MAC blocks, hence we implement all NN functions using generic logic blocks (VersaTiles) and we tune the processing parallelism to fit the available resources (FPGA resource occupation is 70–80% on all three devices).

As shown in Table 9, the FPGA power consumption has a static component, which is consumed in any device mode, and a dynamic component, which is consumed by the FPGA elements activated for processing. Active power is the sum of the static and dynamic powers. Microsemi IGLOO devices have also a sleep mode (that preserves memory and output pin states with the main clock stopped) and a Flash*Freeze power mode (from which the FPGA can return active fast, in less than μs).

TABLE 8. Resource occupation for the neural network hardware implementations for both "short critical path" (SCP) and "long critical path" (LCP) architectures. Implementation on Lattice iCE40 UP5K (LAT) FPGA uses mostly hardware MAC blocks for processing and logic blocks mostly for control, while on Microsemi IGLOO FPGAs (MS-S for AGLN250, MS-M for M1AGL600, and MS-L for M1AGL1000) we use only logic blocks to implement all NN functions.

FPGA / arch. \ Resource		Neuron	Mult.	Total Logic blocks		MAC
				COMB	SEQ	
LAT	LCP	1	8	2047 of 5280 (38.77%)		8 / 8 (100%)
				1724	323	
	SCP	1	8	2093 of 5280 (39.64%)		
				1614	479	
MS-S	SCP	1	2	4889 of 6144 (79.57%)		N/A
				4266	623	
MS-M	LCP	1	8	10031 of 13824 (72.56%)		N/A
				9457	574	
	SCP	1	8	10212 of 13824 (73.87%)		
				9476	736	
MS-L	LCP	2	16	17842 of 24576 (72.60%)		N/A
				17129	713	
	SCP	2	16	18674 of 24576 (75.98%)		
				17651	1023	

TABLE 9. Components of power consumption for all FPGA devices (LAT for Lattice iCE40 UP5K, MS-S for AGLN250, MS-M for M1AGL600, and MS-L for M1AGL1000) for the neural network implementations for both "short critical path" (SCP) and "long critical path" (LCP) architectures.

FPGA / arch. \ Power		Power (mW)				
		Sleep	Freeze	Static	Dynam.	Active
LAT	LCP	N/A	N/A	0.277	5.97	6.24
	SCP				12.6	12.8
MS-S	SCP	0.008	0.059	0.079	7.50	7.58
MS-M	LCP	0.008	0.112	0.131	14.9	15.0
	SCP				17.4	17.5
MS-L	LCP	0.008	0.195	0.213	25.5	25.7
	SCP				34.5	34.7

The smallest FPGA, AGLN250, consumes the least static power. It also has the lowest dynamic and active power consumptions for the SCP architecture, which is the only one that it supports. For the LCP architecture instead, the Lattice iCE40 UP5K device has the lowest dynamic and active powers. This can be due to both the use of dedicated MAC blocks for computations, and because the iCE40 family uses a 40-nm CMOS process while the IGLOO family uses a 130-nm CMOS process. The lower static power consumption of IGLOO family FPGAs may be attributed mostly to the less advanced CMOS technology node they use (130 nm), but its analysis is beyond the scope of this work.

TABLE 10. Performance of NN inference in terms of execution time and energy consumption on the low-power FPGAs (LAT for Lattice iCE40 UP5K, MS-S for AGLN250, MS-M for M1AGL600, and MS-L for M1AGL1000). Long critical path (LCP) implementation (fully parallel processing of one neuron) is not possible on AGLN250, hence we implement both the "long" (LCP) and the "short" critical path (SCP) architectures on all FPGAs to help performance comparison. Implementations on LAT and MS-M have the same architectures and parallelism.

FPGA / arch.		Clock (MHz)	Active power (mW)	Neural network inference		
				Clock cycles	Time (µs)	Energy (nJ)
LAT	LCP	21.58	6.24	20	0.93	5.79
	SCP	45.82	12.8	37	0.81	10.4
MS-S	SCP	24.26	7.58	87	3.59	27.2
MS-M	LCP	16.83	15.0	20	1.19	17.8
	SCP	26.97	17.5	37	1.37	24.0
MS-L	LCP	16.26	25.7	12	0.74	19.0
	SCP	26.15	34.7	21	0.80	27.9

Table 10 shows the performance of one NN inference. Due mostly to its dedicated MAC blocks and more advanced fabrication node, the Lattice iCE40 UP5K achieves the highest clock frequency and the lowest energy consumption for both NN implementation architectures, the SCP (45.82 MHz) and the LCP (21.58 MHz). It also consumes the lowest active power for the LCP architecture (6.24 mW), while the smallest IGLOO FPGA, AGLN250, consumes the lowest active power for the SCP architecture (7.58 mW). Thanks to the higher implementation parallelism (see Figure 10), the largest IGLOO FPGA, M1AGL1000, performs one NN inference in the shortest time for both the SCP (0.80 μs) and the LCP (0.74 μs) architectures, but the Lattice iCE40 UP5K is very close for the SCP architecture (0.81 μs).

We also note that the LCP architecture, which processes all neuron calculations in one clock cycle, is the most performant. As expected, its maximum clock frequency is lower than for SCP, at 21.58 MHz on Lattice iCE40 UP5K. But the minimum NN inference energy is also much lower, 5.79 nJ on the same device, while the inference time of 0.93 μs is just 16% longer than the best of the SCP architecture, of 0.80 μs.

To directly compare devices from different producers, we implemented the same NN architecture and scheduling on the Lattice iCE40 UP5K and the Microsemi IGLOO M1AGL600 FPGAs. While on both devices one NN inference needs 20 clock cycles for the LCP and 37 cycles for the SCP architectures, the Lattice UP5K FPGA comes on top for all other metrics for both SCP and LCP architectures (see Table 10): higher maximum clock frequency (28% higher

for LCP and 70% higher for SCP), lower active power (58% lower for LCP and 27% lower for SCP), shorter NN inference time (22% shorter for LCP and 41% shorter for SCP), and lower NN inference energy (67% lower for LCP and 57% lower for SCP). The performance difference is expected considering that the Lattice iCE40 family uses a more advanced 40-nm CMOS technology node (compared to the 130-nm node for Microsemi IGLOO family) and has also dedicated MAC blocks.

Comparing only the three IGLOO FPGAs (AGLN250, M1AGL600, and M1AGL1000 which use the same fabrication technology), the most energy-efficient is the intermediate one, M1AGL600, for both the SCP and LCP architectures. Compared to it, using the same SCP architecture the smaller AGLN250 needs 2.6 times more time per NN inference, consumes 57% less active power, but 13% more energy. The inference time on the larger M1AGL1000 device is 38% shorter for LCP and 42% shorter for SCP but consumes 71% more active power for LCP and 98% more for SCP, hence 7% more energy for LCP architecture and 16% more for SCP (see Table 10). The differences are mostly due to the different sizes of the non-computational logic (such as multiplexers or control blocks) and to the use of more clock cycles than computation parallelism ratios would imply. For instance, M1AGL600 has four times the multipliers of AGLN250 (see Table 8), yet the latter uses 4.4 times more clock cycles (87 versus 20 cycles, see Table 10), while M1AGL1000 has twice the multipliers of M1AGL600, yet the former needs more than half of the clock cycles of the latter (12 versus 20 cycles for LCP architecture, and 21 versus 37 cycles for SCP).

C. FPGA–MICROCONTROLLER COMMUNICATION

We must also consider the MCU–FPGA communication energy when offloading the NN inference processing to FPGAs. The NN expects six sensor data encoded on 17-bit integers and encodes the inference result on a 4-bit integer. Overall data transfer energy depends on bus type, data transfer rate, and bus interface operation on MCU and FPGA.

TABLE 11. Communication ports available on the MCU and the FPGA devices that we consider (LAT for Lattice iCE40 UP5K, MS-S for AGLN250, MS-M for M1AGL600, and MS-L for M1AGL1000).

Device \ Bus type	SPI	I²C	UART	Parallel
MCU STM32L152RE	2	2	3	GPIO
FPGA LAT	1	1	N/A	GPIO
FPGA MS-S	N/A	N/A	1	GPIO
FPGA MS-M	N/A	N/A	1	GPIO
FPGA MS-L	N/A	N/A	1	GPIO

Table 11 shows the communication interfaces available on each device that we consider (we include only the MCU with the best performance). The MCU has HW support for SPI, I^2C, and universal asynchronous receiver-transmitter (UART) ports. The FPGAs have either dedicated SPI and I^2C ports (Lattice iCE40 family) or UART IP cores (Microsemi IGLOO family). We can also implement ad-hoc parallel buses using general purpose input/output (GPIO) pins.

SPI, I^2C, and UART interfaces transfer data serially (the UART implements the recommended standard 232 (RS-232) protocol). Serial buses require few pins but transmit data bits one at a time, often interleaved with protocol-specific overhead. Generally, the communication protocol that supports the higher clock frequency is preferred because it reduces the overall communication time.

Parallel buses use more I/O pins. They transmit several data bits in parallel and may reduce considerably the data transfer time. The MCU and the FPGAs that we consider have enough spare GPIO pins to implement a parallel bus.

We will comparatively analyze the overall energy consumption for one NN inference of the MCU–FPGA system when using the RS-232 (the only communication core supported by Microsemi IGLOO FPGA family), the SPI (the fastest core supported by Lattice iCE40 FPGA family), the parallel bus, or no FPGA acceleration (SW NN processing).

1) RS-232 SERIAL COMMUNICATION WITH IGLOO FPGA

Microsemi IGLOO family provides only UART cores (on 521 VersaTiles) that can implement the RS-232 protocol for serial communication. The STM32L152RE UART peripherals have direct memory access (DMA, see Figure 11), meaning that the CPU can stay in low-power mode during data transfer to save energy.

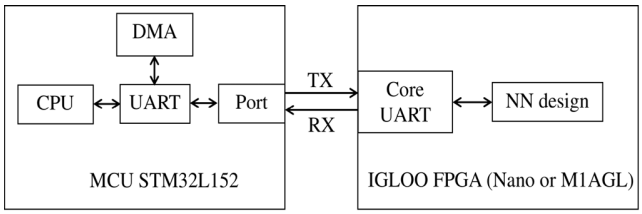

FIGURE 11. Block schematic of a UART RS-232 interface between the microcontroller and the FPGA. Direct Memory Access (DMA) can transfer data to and from the UART port without involving the CPU, which allows to save energy.

Figure 12 shows the power consumption components during a NN inference with data transferred over a bus. During T_1, sensor data are transferred from MCU memory to FPGA memory over a bus. During T_3 , the result of the NN inference is transferred from FPGA memory to MCU memory, over the same bus. During T_2 , only the FPGA is active to compute the NN inference.

The CPU can sleep in low-power mode during almost all T_1–T_2–T_3 periods, having the DMA subsystem handling the UART–MCU memory transfers and the UART handling the RS-232 protocol. The peripherals (UARTs and GPIOs of the MCU and the FPGA, and DMA on the MCU) consume power during both T_1 and T_3, to transfer data between MCU and FPGA. The FPGA always consumes static power, while its dynamic power depends on the workload (the

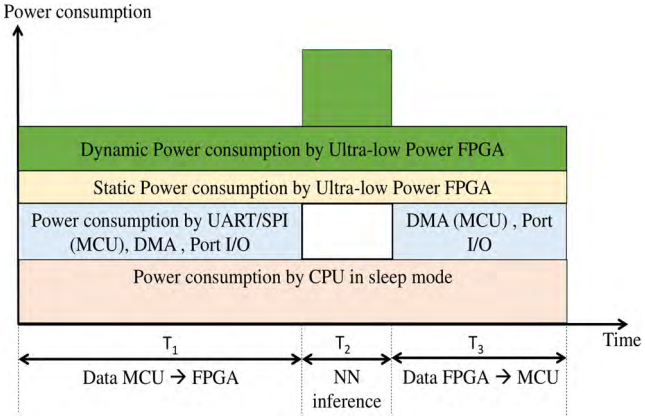

FIGURE 12. Components of power consumption for one neural network inference using data transfer between the microcontroller and an FPGA.

UART core during T_1 and T_3 , and the NN inference during T_2).

The duration of T_1 and T_3 is set to accommodate the data transfer over the RS-232 line as follows:

$$T = (N_{frames} * b_{frame})/B \tag{1}$$

where:

- T is the RS-232 transmission time (T_1, T_3);
- N_{frames} is the number of transmitted data frames;
- b_{frame} is the number of bits in one data frame;
- B is the data frame Baud rate over the serial line, which we set to the maximum supported by the MCU and the FPGA UARTs,

$$B = 921.6\,Kbit/s \tag{2}$$

T_1 needs to accommodate the serial transfer of the six 17-bit sensor values, each packed in three ten-bit RS-232 frames (made of eight bits of data, one start bit, and one stop bit) for a total transfer of 180 bits. T_3 needs to accommodate the serial transfer of the four data bits of the inference result, hence only one RS-232 ten-bit frame. With these considerations, from (1) and (2) we obtain:

$$\begin{aligned} T_1 &= \frac{6\,value \cdot 3\,frame/value \cdot 10\,bit/frame}{921600\,bit/s} \\ &= 195.3\,\mu s \end{aligned} \tag{3}$$

$$\begin{aligned} T_3 &= \frac{1\,value \cdot 1\,frame/value \cdot 10\,bit/frame}{921600\,bit/s} \\ &= 10.9\,\mu s \end{aligned} \tag{4}$$

T_2 is the duration of the NN inference on the target IGLOO FPGA, which depends on the FPGA type (see Table 10).

As shown in Figure 12, power consumption during data transfer periods T_1 and T_3 is given by:

$$\begin{aligned} P_{data_{transfer}} = {} & P_{CPU_{sleep}} + P_{UART_{MCU}} + P_{DMA} + P_{GPIO_{MCU}} \\ & + P_{FPGA_{static}} + P_{FPGA_{dynamic}} + P_{GPIO_{FPGA}} \end{aligned} \tag{5}$$

where $P_{CPU_{sleep}}$ is the CPU power in sleep mode, $P_{UART_{MCU}}$ is the power consumed by the UART peripheral of the CPU, P_{DMA} is the DMA subsystem power, $P_{GPIO_{MCU}}$ is the power consumed by the GPIO pins of the UART port of the CPU, $P_{FPGA_{static}}$ is the FPGA static power, $P_{FPGA_{dynamic}}$ is the FPGA dynamic power, and $P_{GPIO_{FPGA}}$ is the power consumed by the GPIO pins of the UART core of the FPGA.

We show in Table 12 the MCU-FPGA system energy consumption. A relatively long data transfer time noticeably increases the overall system energy consumption.

TABLE 12. Power and energy consumed by the system made of the microcontroller and the FPGA (MS-S for AGLN250, MS-M for M1AGL600, and MS-L for M1AGL1000), which exchange data over an RS-232 serial line with UART interfaces.

N on		MS-S	MS-M	MS-L
MCU power for 16 MHz clock (mW)	$P_{CPU_{sleep}}$	2.01		
	$P_{UART_{MCU}}$	0.331		
	P_{DMA}	0.696		
	$P_{GPIO_{MCU}}$	0.254		
FPGA power (mW)	$P_{FPGA_{static}}$	0.079	0.131	0.213
	$P_{FPGA_{dynamic}}$ (NN inference)	7.50	14.9	25.5
	$P_{FPGA_{dynamic}}$ (core UART)	0.58		
	$P_{GPIO_{FPGA}}$	0.014		
Latency (µs)		210	207	207
Energy (µJ)		0.845	0.846	0.864

2) SPI SERIAL COMMUNICATION WITH ICE40 FPGA

As shown in Table 11, only the Lattice iCE40 FPGA family provides an SPI core (on 40 LUTs). On MCU, we will use the DMA subsystem to transfer data between the MCU memory and the SPI port to reduce the CPU energy consumption (as we do for RS-232).

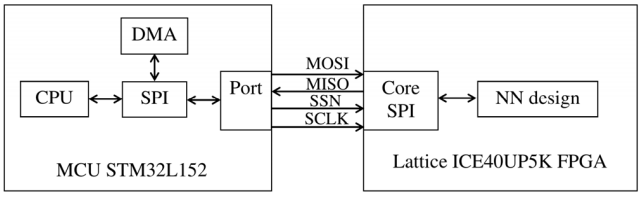

FIGURE 13. Block schematic of an SPI interface between the microcontroller and the FPGA. Direct Memory Access (DMA) transfers data to and from the SPI port without involving the CPU to save energy.

Figure 13 shows the block schematic of MCU-FPGA SPI communication.

The DMA subsystem handles data transfer between MCU memory and SPI port to keep the CPU mostly in low-power

sleep mode (see Figure 12). Peripherals (SPIs and GPIOs of MCU and FPGA, and DMA on MCU) consume power during both $T_1$and T_3, when data is transferred between the MCU and FPGA. The FPGA always consumes static power and dynamic power depending on its activity (the SPI core duringT_1 andT_3 , and the NN inference during T_2).

Like for the RS-232 protocol, T_1 and T_3 are set to accommodate data transfer over the SPI bus as follows:

$$T = (N_{frames} * b_{frame})/B \tag{6}$$

where:

- T is the SPI transmission time (T_1, T_3);
- N_{frames} is the number of transmitted data frames;
- b_{frame} is the number of bits in one data frame;
- B is the data frame Baud rate over the serial line, which we set to the maximum supported by both the MCU and the FPGA SPI ports.

The FPGA SPI core can operate up to 25 Mbit/s. STM32L152 MCU SPI is limited to 8 Mbit/s for 16 MHz CPU clock, and 16 Mbit/s for 32 MHz CPU clock. Since MCU consumption varies with the clock frequency, we will analyze the overall energy consumption for both SPI speeds.

SPI data frames can hold up to 16 data bits. Each transfer session has a 24-bit header (8-bit SPI command and 16-bit data address). Hence, T_1 is set to the duration of the transfer of the six 17-bit sensor values, each value packed in two 16-bit SPI frames, and the SPI session header. T_3 is set to the duration of the serial transfer of the four data bits of the inference result, packed in one 16-bit SPI frame, plus the SPI session header. With these considerations, from (6) we obtain for 8 Mbit/s SPI speed

$$\begin{aligned} T_1 &= \frac{6\,value \cdot 2\,frame/value \cdot 16\,bit/frame + 24\,bit}{8\,Mbit/s} \\ &= 27\,\mu s \end{aligned} \tag{7}$$

$$\begin{aligned} T_3 &= \frac{1\,value \cdot 1\,frame/value \cdot 16\,bit/frame + 24\,bit}{8\,Mbit/s} \\ &= 5\,\mu s \end{aligned} \tag{8}$$

and half of these durations when doubling the SPI speed to 16 Mbit/s

$$T_1 = 27\,\mu s/2 = 13.5\,\mu s \tag{9}$$

$$T_3 = 5\,\mu s/2 = 2.5\,\mu s \tag{10}$$

T_2 is the duration of the NN inference on the iCE40 FPGA (see Table 10).

From Figure 12, we obtain the whole system power consumption during data transfer periods T_1 and T_3 as

$$\begin{aligned} P_{data_{transfer}} = {} & P_{CPU_{sleep}} + P_{SPI_{MCU}} + P_{DMA} + P_{GPIO_{MCU}} \\ & + P_{FPGA_{static}} + P_{FPGA_{dynamic}} + P_{GPIO_{FPGA}} \end{aligned} \tag{11}$$

where $P_{CPU_{sleep}}$ is the CPU power in sleep mode, $P_{SPI_{MCU}}$ is the power consumed by the SPI peripheral of the CPU, P_{DMA} is the DMA subsystem power, $P_{GPIO_{MCU}}$ is the power consumed by the GPIO pins of the SPI port of the CPU, $P_{FPGA_{static}}$ is the FPGA static power, $P_{FPGA_{dynamic}}$ is the FPGA dynamic power, and $P_{GPIO_{FPGA}}$ is the power consumed by the GPIO pins of the SPI core of the FPGA.

TABLE 13. Power and energy consumed by the system made of the microcontroller and the FPGA (LAT for Lattice iCE40 UP5K), which exchange data over an SPI serial bus.

		NN on LAT	
		8 Mbit/s SPI	16 Mbit/s SPI
MCU power values (mW)	$P_{CPU_{sleep}}$	2.01	5.775
	$P_{SPI_{MCU}}$	0.187	0.507
	P_{DMA}	0.696	1.774
	$P_{GPIO_{MCU}}$	0.254	0.665
FPGA power values (mW)	$P_{FPGA_{static}}$	0.277	
	$P_{FPGA_{dynamic}}$ (NN inference)	5.97	
	$P_{FPGA_{dynamic}}$ (core SPI)	0.13	0.26
	$P_{GPIO_{FPGA}}$	0.147	0.294
Latency (µs)		32.9	16.9
Energy (µJ)		0.124	0.158

Table 13 shows the energy consumption of the MCU–FPGA system using an SPI serial bus for data exchange, for two SPI data rates. Data transfer takes significantly less time than when using an RS-232 serial line, hence the overall system energy consumption is reduced.

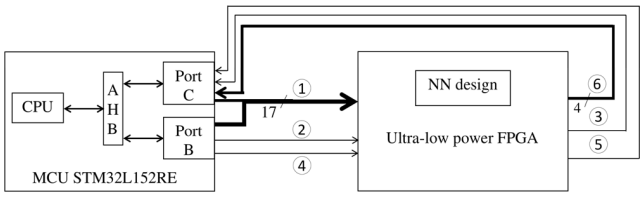

FIGURE 14. Block schematic of a parallel interface between the microcontroller (MCU) and the FPGA implemented using general purpose (GPIO) pins. The interface uses two 16-bit MCU ports to transfer the input (sensor) data, of which one port is reused to receive the inference result, and some control lines.

3) PARALLEL COMMUNICATION WITH FPGA

Parallel communication can be implemented on all FPGAs that we consider in our experiments (see Table 11) using GPIO pins and little resource or protocol overhead. Figure 14 shows the block schematic of a parallel communication between the MCU and the FPGA. It uses two 16-bit MCU ports to transfer in parallel the 17 bits of each input sample (their destination addresses within the FPGA are implicit), and a few lines for protocol signaling. One of the MCU ports is reused to receive the 4-bit NN inference result.

Unlike serial buses (see Figure 12), the parallel bus can transfer one input sample in one CPU clock cycle. Hence, we use the CPU to transfer the data between memory and ports at full speed. The CPU and the GPIOs of the ports consume power during T_1 and T_3 (see Figure 12). The FPGA consumes static power, while its dynamic power depends on the activity (the parallel port handler core during T_1 and T_3 (which is negligible) and the NN calculating the inference during T_2).

T_1, T_2 , and T_3 have the same meaning as for serial buses. T_2 depends on the target FPGA type, as shown in Table 10. We measured that T_1 takes 36 CPU clock cycles and T_3 four cycles. Hence, for 16 MHz CPU clock frequency, we have:

$$T_1 = 36\,cycles/16\,MHz = 2.25\,\mu s \quad (12)$$

$$T_3 = 4\,cycles/16\,MHz = 0.25\,\mu s \quad (13)$$

and half of these for 32 MHz CPU clock frequency:

$$T_1 = 36\,cycles/32\,MHz = 1.125\,\mu s \quad (14)$$

$$T_3 = 4\,cycles/32\,MHz = 0.125\,\mu s \quad (15)$$

From Figure 12, the power consumption during the data transfer period T_1 and T_3 is given by:

$$P_{data_{transfer}} = P_{CPU_{active}} + P_{GPIO_{MCU}} + P_{FPGA_{static}} + P_{FPGA_{dynamic}} + P_{GPIO_{FPGA}} \quad (16)$$

where $P_{CPU_{active}}$ is the CPU power in active mode, $P_{GPIO_{MCU}}$ is the power consumed by the GPIO pins of the parallel port of the CPU, $P_{FPGA_{static}}$ is the FPGA static power, $P_{FPGA_{dynamic}}$ is the FPGA dynamic power, and $P_{GPIO_{FPGA}}$ is the power consumed by the FPGA GPIO pins used for the parallel bus.

Table 14 shows the energy consumption of the MCU–FPGA system that uses a parallel bus for data exchange, for two CPU clock frequencies. Data transfer time is very short compared to serial buses, which reduces significantly the overall system energy consumption.

TABLE 14. Power and energy consumed by the system made of a microcontroller and an FPGA, which exchange data over a parallel bus.

			NN on LAT	NN on MS-S	NN on MS-M	NN on MS-L
MCU power values (mW)	$P_{CPU_{active}}$	16 MHz	10.8			
		32 MHz	28.38			
	$P_{GPIO_{MCU}}$	16 MHz	0.2544			
		32 MHz	0.6653			
FPGA power values (mW)	$P_{FPGA_{static}}$		0.277	0.079	0.131	0.213
	$P_{FPGA_{dynamic}}$ (NN inference)		5.967	7.498	14.856	25.459
	$P_{GPIO_{FPGA}}$	16 MHz	0.694	1.84		
		32 MHz	1.388	3.67		
Latency (µs)		16 MHz	3.427	6.09	3.69	3.24
		32 MHz	2.177	4.84	2.44	1.99
Energy (µJ)		16 MHz	0.039	0.067	0.0533	0.0534
		32 MHz	0.0504	0.09	0.067	0.065

D. DISCUSSION

When offloading the NN inference from the MCU to an external FPGA, the communication between the MCU and the FPGA using the RS-232 serial line has the highest latency, 207 μs–210 μs, and the highest energy consumption, 845 nJ–864 nJ, because the data transfer time is dominant (see Table 12). Data communication over the SPI reduces latency by roughly an order of magnitude, to 16.9 μs–32.9 μs, and the overall system energy by 5–7 times, to 124 nJ–158 nJ (see Table 13). With a parallel data communication, the system is the most efficient, with latency lowered by roughly another order of magnitude, between 1.99 μs and 6.09 μs, and the energy is reduced between 39 nJ and 90 nJ (see Table 14).

The inference latency of the NN implemented in SW is between 54.4 μs and 92.1 μs, and the energy between 990 nJ and 1530 nJ for the STM32L152RE MCU with clock frequencies of 16 MHz and 32 MHz respectively (see Table 5). SW NN latency is only better than the HW one with data communication over an RS-232 serial line. SW NN inference energy is however higher than any HW implementation, including the data transfer energy.

Moreover, specialized blocks (like the Lattice iCE40 UP5K (LAT) MACs) may improve the energy performance by more than three times, even when a higher parallelism (exploiting the available resources in Microsemi IGLOO M1AGL1000 (MS-L)) lowers the NN inference time (see Table 10). It should be also noted that the Lattice FPGA uses a more advanced CMOS technology node (40 nm) than the Microsemi FPGAs (130 nm).

The experimental results show that processing the NN inference on external ultralow-power FPGAs is more energy-efficient than the SW processing of power-optimized MCUs, even with HW multipliers. Speed-optimized MCUs are even less efficient. Higher MCU–FPGA data transfer speeds may significantly reduce the overall transfer energy and latency. Parallel buses perform better for this reason and the energy reduction due to shorter transfer time offsets the increase of energy consumption due to the extra GPIOs that they use.

VI. CONCLUSION

We comparatively analyzed the performance of several SW and HW implementations of a typical multilayer perceptron NN for person identification using capacitive sensors. We optimized the NN architecture using augmented experimental data from our previous experiment, and we implemented the NN in SW on two low-power MCUs

(one optimized for low power and one optimized for performance), and two architectural options in HW (with different critical path lengths), on four ultralow-power FPGAs with different sizes and specialized blocks (MACs).

The experimental results show that the NN HW implementation is more efficient both as latency and energy, except when using a slow MCU–FPGA communication (like the RS-232), which may excessively increase the overall latency. Using fast communication, FPGA acceleration of NN inference can reduce its latency and energy consumption by more than an order of magnitude.

As future work we plan to integrate an ultralow-power FPGA with a low-power MCU on a capacitive sensor to analyze the dynamic behavior of the system, including the operation under very low duty cycles.

We also plan to use high-level synthesis (HLS) techniques to implement the NN on FPGA to explore more architectures and computational models (e.g., vectorization) and compare HLS implementation quality and manual optimization [50].

REFERENCES

[1] J. Iqbal, M. T. Lazarescu, O. B. Tariq, A. Arif, and L. Lavagno, "Capacitive sensor for tagless remote human identification using body frequency absorption signatures," *IEEE Trans. Instrum. Meas.*, vol. 67, no. 4, pp. 789–797, Apr. 2018.

[2] J. Xiao, Z. Zhou, Y. Yi, and L. M. Ni, "A survey on wireless indoor localization from the device perspective," *ACM Comput. Surv.*, vol. 49, no. 2, p. 25, Oct. 2016.

[3] C. Wang, S. Chen, Y. Yang, F. Hu, F. Liu, and J. Wu, "Literature review on wireless sensing-Wi-Fi signal-based recognition of human activities," *Tsinghua Sci. Technol.*, vol. 23, no. 2, pp. 203–222, Apr. 2018.

[4] S. Sharma and A. Ghose, "Analysis of analog signal from PIR-sensors for human analytics in an IOTized environment," in *Proc. 3rd IEEE Int. Conf. Internet Things, Smart Innov. Usages (IoT-SIU)*, Feb. 2018, pp. 1–6.

[5] J. S. Fang, Q. Hao, D. J. Brady, B. D. Guenther, and K. Y. Hsu, "Real-time human identification using a pyroelectric infrared detector array and hidden Markov models," *Opt. Express*, vol. 14, no. 15, pp. 6643–6658, Jul. 2006.

[6] N. Khalil, D. Benhaddou, O. Gnawali, and J. Subhlok, "Nonintrusive occupant identification by sensing body shape and movement," in *Proc. ACM Int. Conf. Syst. Energy-Efficient Built Environ. (ACM BuildSys)*, 2016, pp. 1–10.

[7] G. Mokhtari, Q. Zhang, C. Hargrave, and J. C. Ralston, "Non-wearable UWB sensor for human identification in smart home," *IEEE Sensors J.*, vol. 17, no. 11, pp. 3332–3340, Jun. 2017.

[8] S. Pan, T. Yu, M. Mirshekari, J. Fagert, A. Bonde, O. J. Mengshoel, H. Y. Noh, and P. Zhang, "FootprintID: Indoor pedestrian identification through ambient structural vibration sensing," in *Proc. ACM Interact. Mobile Wearable Ubiquitous Technol.*, vol. 1, no. 3, pp. 89:1–89:31, Sep. 2017. doi: 10.1145/3130954.

[9] T. G. Puppendahl, X. Dellangnol, C. Hatzfeld, B. Fu, M. Kupnik, A. Kuijper, M. Hastall, J. Scott, and M. Gruteser, "Platypus: Indoor localization and identification through sensing of electric potential changes in human bodies," in *Proc. 14th Int. Conf. MobiSys. ACM*, 2016, pp. 17–30.

[10] A. W. Braun, R. Wichert, A. Kuijper, and D. W. Fellner, "Capacitive proximity sensing in smart environments," *J. Ambient Intell. Smart Environ.*, vol. 7, no. 4, pp. 483–510, 2015.

[11] T. Grosse-Puppendahl, C. Holz, G. Cohn, R. Wimmer, O. Bechtold, S. Hodges, M. S. Reynolds, and J. R. Smith, "Finding common ground: A survey of capacitive sensing in human-computer interaction," in *Proc. SIGCHI Conf. Hum. Factors Comput. Syst. (CHI)*, 2017, pp. 3293–3315.

[12] J. Iqbal, A. Arif, O. B. Tariq, M. T. Lazarescu, and L. Lavagno, "A contactless sensor for human body identification using RF absorption signatures," in *Proc. IEEE Sensors Appl. Symp. (SAS)*, Mar. 2017, pp. 1–6.

[13] A. Arshad, S. Khan, A. H. M. Z. Alam, R. Tasnim, T. S. Gunawan, R. Ahmad, and C. Nataraj, "An activity monitoring system for senior citizens living independently using capacitive sensing technique," in *Proc. Instrum. Meas. Technol. Conf. Proc. (I2MTC)*, May 2016, pp. 1–6.

[14] A. R. Akhmareh, M. T. Lazarescu, O. B. Tariq, and L. Lavagno, "A tagless indoor localization system based on capacitive sensing technology," *Sensors*, vol. 16, no. 9, p. 1448, 2016.

[15] J. Iqbal, M. T. Lazarescu, O. B. Tariq, and L. Lavagno, "Long range, high sensitivity, low noise capacitive sensor for tagless indoor human localization," in *Proc. 7th IEEE Int. Workshop Adv. Sensors Int. (IWASI)*, Vieste, Italy, Jun. 2017, pp. 189–194.

[16] J. Iqbal, M. T. Lazarescu, A. Arif, and L. Lavagno, "High sensitivity, low noise front-end for long range capacitive sensors for tagless indoor human localization," in *Proc. 3rd IEEE Int. Forum Res. Technol. Soc. Ind. (RTSI)*, Modena, Italy, Sep. 2017, pp. 189–194.

[17] O. I. Abiodun, A. Jantan, A. E. Omolara, K. V. Dada, N. Mohamed, and H. Arshad, "State-of-the-art in artificial neural network applications: A survey," *Heliyon*, vol. 4, no. 11, p. e00938, Nov. 2018. doi: 10.1016/j.heliyon.2018.e00938.

[18] O. B. Tariq, M. T. Lazarescu, J. Iqbal, and L. Lavagno, "Performance of machine learning classifiers for indoor person localization with capacitive sensors," *IEEE Access*, vol. 5, pp. 12913–12926, Jul. 2017.

[19] B. Lavi, M. F. Serj, and I. Ullah, "Survey on deep learning techniques for person re-identification task," in *Proc. CVPR*, Jun. 2018. [Online]. Available: https://arxiv.org/abs/1807.05284

[20] K. Guo, S. Zeng, J. Yu, Y. Yang, and H. Yang, "A survey of FPGA based neural network accelerator," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 9, no. 4, p. 11, Dec. 2017.

[21] A. L. S. Braga, C. H. Llanos, D. Gohringer, and J. Obie, "Performance, accuracy, power consumption and resource utilization analysis for hardware/software realized artificial neural networks," in *Proc. IEEE 5th Int. Conf. Bio-Inspired Comput., Theories Appl. (BIC-TA)*, Sep. 2010, pp. 1629–1636.

[22] A. G. Blaiech, K. B. Khalifa, M. Boubaker, and M. H. Bedoui, "Multi-width fixed-point coding based on reprogrammable hardware implementation of a multi-layer perceptron neural network for alertness classification," in *Proc. 10th Int. Conf. Intell. Syst. Design Appl. (ISDA)*, Dec. 2010, pp. 610–614.

[23] M. Bahoura and C.-W. Park, "FPGA-implementation of high-speed MLP neural network," in *Proc. 18th IEEE Int. Conf. Electron., Circuits Syst. (ICECS)*, Dec. 2011, pp. 426–429.

[24] X. Zhai, A. A. S. Ali, A. Amira, and F. Bensaali, "MLP neural network based gas classification system on Zynq SoC," *IEEE Access*, vol. 4, pp. 8138–8146, 2016.

[25] C. Latino, M. A. Moreno-Armendariz, and M. Hagan, "Realizing general MLP networks with minimal FPGA resources," in *Proc. Int. Joint Conf. Neural Netw.*, Jun. 2009, pp. 1722–1729.

[26] S. Hariprasath and T. N. Prabakar, "A pipelined approach for FPGA implementation of multi modal biometric pattern recognition using prototype based supervised neural network," in *Proc. Int. Conf. Commun. Signal Process.*, Nov. 2014, pp. 1837–1843.

[27] M. Bettoni, G. Urgese, Y. Kobayashi, E. Macii, and A. Acquaviva, "A convolutional neural network fully implemented on FPGA for embedded platforms," in *Proc. New Gener. CAS (NGCAS)*, Genoa, Italy, Sep. 2017, pp. 49–52.

[28] S. A. Abbas and G. Vicithra, "Actualization of face detection in FPGA using neural network," in *Proc. Int. Conf. Wireless Commun., Signal Process. Netw. (WiSPNET)*, Chennai, India, 2016, pp. 634–638.

[29] S. A. Dawwd and B. S. Mahmood, "Video-based face recognition via convolutional neural networks," in *New Approaches to Characterization and Recognition of Faces*. Croatia, Rijeka: InTech, 2011, pp. 131–152.

[30] G. Gabriel, C. Jara, J. Pomares, A. Alabdo, L. Poggi, and F. Torres, "A survey on FPGA-based sensor systems: Towards intelligent and reconfigurable low-power sensors for computer vision, control and signal processing," *Sensors*, vol. 14, no. 4, pp. 6247–6278, 2014. [Online]. Available: https://www.mdpi.com/1424-8220/14/4/6247

[31] A. de la Piedra, A. Braeken, and A. Touhafi, "Sensor systems based on FPGAs and their applications: A survey," *Sensors*, vol. 12, no. 9, pp. 12235–12264, 2012.

[32] B. J. Klauenberg and D. Miklavčič, *Radio Frequency Radiation Dosimetry and its Relationship to the Biological Effects of Electromagnetic Fields*, vol. 82. Springer, 2012.

[33] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–444, May 2015.

[34] J. Lemley, S. Bazrafkan, and P. Corcoran, "Smart augmentation learning an optimal data augmentation strategy," *IEEE Access*, vol. 4, pp. 5858–5869, Apr. 2017.

[35] J. Sola and J. Sevilla, "Importance of input data normalization for the application of neural networks to complex industrial problems," *IEEE Trans. Nucl. Sci.*, vol. 44, no. 3, pp. 1464–1468, Jun. 1997.
[36] T. Jayalakshmi and A. Santhakumaran, "Statistical normalization and back propagation for classification," *Int. J. Comput. Theory Eng.*, vol. 3, no. 1, pp. 89–93, 2011.
[37] V. Tiwari and N. Khare, "Hardware implementation of neural network with sigmoidal activation functions using CORDIC," *Microprocess. Microsyst.*, vol. 39, no. 6, pp. 373–381, 2015.
[38] I. Sahin and I. Koyuncu, "Design and implementation of neural networks neurons with radBas, logsig, and tansig activation functions on FPGA," *Elektronika ir elektrotechnika*, vol. 120, no. 4, pp. 51–54, 2012.
[39] L. Lavagno, M. T. Lazarescu, L. Papaefstathiou, A. Brokalakis, J. Walters, B. Kienhuis, and F. Schäfer, "HEAP: A highly efficient adaptive multi-processor framework," *Microprocess. Microsyst.*, vol. 37, no. 8, pp. 1050–1062, 2013.
[40] A. Arif, F. A. Barrigon, F. Gregoretti, J. Iqbal, L. Lavagno, M. T. Lazarescu, L. Ma, M. Palomino, and J. L. L. Segura, "Performance and energy-efficient implementation of a smart city application on FPGAs," *J. Real-Time Image Process.*, pp. 1–15, Jun. 2018.
[41] P. Sayyah, M. T. Lazarescu, S. Bocchio, E. Ebeid, G. Palermo, D. Quaglia, A. Rosti, and L. Lavagno, "Virtual platform-based design space exploration of power-efficient distributed embedded applications," *ACM Trans. Embedded Comput. Syst.*, vol. 14, no. 3, p. 49, 2015.
[42] 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 *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays ACM*, 2016, pp. 26–35.
[43] L. Jiao, C. Luo, W. Cao, X. Zhou, and L. Wang, "Accelerating low bit-width convolutional neural networks with embedded FPGA," in *Proc. IEEE 27th Int. Conf. Field Program. Logic Appl. (FPL)*, Sep. 2017, pp. 1–4.
[44] G. G. Lemieux, J. Edwards, J. Vandergriendt, A. Severance, R. De Iaco, A. Raouf, H. Osman, T. Watzka, and S. Singh, "TinBiNN: Tiny binarized neural network overlay in about 5,000 4-LUTs and 5mW," 2019, *arXiv:1903.06630*. [Online]. Available: https://arxiv.org/abs/1903.06630
[45] *DocID17659 Rev 12, STM32L15x6/8/B*, Rev12, STMicroelectron., Geneva, Switzerland, Apr. 2016.
[46] *DocID13587 Rev 17, STM32F103x8,*, Rev17, STMicroelectron., Geneva, Switzerland, Aug. 2015.
[47] J. Síma and P. Orponen, "General-purpose computation with neural networks: A survey of complexity theoretic results," *Neural Comput.*, vol. 15, no. 12, pp. 2727–2778, 2003.
[48] *iCE40 UltraPlus Family: Data Sheet*, Lattice Semicond., Hillsboro, OR, USA, Aug. 2017.
[49] *DS0095: IGLOO Low Power Flash FPGAs With Flash* Freeze Technology, Data Sheet*, Microsemi, Aliso Viejo, CA, USA, May 2016.
[50] A. Qamar, F. B. Muslim, F. Gregoretti, L. Lavagno, and M. T. Lazarescu, "High-level synthesis for semi-global matching: Is the juice worth the squeeze?" *IEEE Access*, vol. 5, pp. 8419–8432, 2016.

MARWEN ROUKHAMI received the master's degree in electronic, electrotechnical and automatic (EEA) from the Faculty of Sciences of Tunis, in 2013. He is currently pursuing the Ph.D. degree in electronics with the Laboratory of the Energy Efficiency and Renewable Energies (LAPER), Faculty of Sciences of Tunis, Tunis. From 2013 to 2016, he was a Teaching Assistant with the National Engineering School of Carthage, Tunisia. His research interests include embedded system design and wireless sensor networks.

MIHAI TEODOR LAZARESCU received the Ph.D. degree from the Politecnico di Torino, Italy, in 1998. He was a Senior Engineer with Cadence Design Systems and founded several startups. He currently serves as an Assistant Professor with the Politecnico di Torino. He coauthored more than 40 scientific publications and several books. His research interests include sensors for indoor localization, reusable WSN platforms, high-level hardware/software co-design, and high-level synthesis of WSN applications.

FRANCESCO GREGORETTI (M'75) graduated from the Politecnico di Torino, Italy, in 1975. From 1976 to 1977, he was an Assistant Professor with the Swiss Federal Institute of Technology, Lausanne, Switzerland, and from 1983 to 1985, a Visiting Scientist with the Department of Computer Science, Carnegie Mellon University, Pittsburgh, USA. He is currently a Professor in microelectronics with the Politecnico di Torino. His current research interests include digital electronics, VLSI circuits, massively parallel multi-microprocessor systems for VLSI CAD tools and in image processing architectures, especially co-design methodologies for complex electronic systems, to methodologies for reduction of electromagnetic emissions and power consumption of processing architectures using asynchronous methodologies.

YOUNES LAHBIB received the Engineer Diploma degree in electrical and electronic engineering from the EcoleNationaled'Ingénieurs de Monastir (ENIM), Tunisia, in 2000, and the master's degree in electronic devices and materials and the Ph.D. degree in physics-microelectronics from the Faculty of Sciences, Monastir, in 2002 and 2006, respectively. He was with ST Microelectronics, from 2001 to 2006, as a Reseach and Development Engineer preparing his Ph.D. Since 2007, he has been an Assistant Professor in microelectronics and embedded systems with the Department of Electrical Engineering and Computer Sciences, EcoleNationaled'Ingénieurs de Carthage, University of Carthage. He is also a member of the EμE Laboratory, working on embedded cryptographic systems, and DSP and FPGA accelerations methods. His research interest includes HLS, SystemCSoC modeling and assertions-based verification of hardware systems.

ABDELKADER MAMI received the Dissertation H.D.R. (Enabling to Direct Research) from the University of Lille, France, in 2003. He is currently a Professor with the Faculty of Sciences of Tunis (FST) and a Member of Scientific Advisor with the Faculty of Science of Tunis, Tunisia. He is the President of the thesis committee of electronics with the Faculty of Sciences of Tunis and the Director of the Laboratory of the Energy Efficiency and Renewable Energies (LAPER), Tunis.