Multimodal Neural Network Acceleration on a Hybrid CPU-FPGA Architecture: A Case Study

MEHDI TRABELSI AJILI, (Member, IEEE), AND YUKO HARA-AZUMI, (Member, IEEE)
Department of Information and Communications, School of Engineering, Tokyo Institute of Technology, Tokyo 152-8550, Japan
Corresponding author: Mehdi Trabelsi Ajili (mehdi@cad.ict.e.titech.ac.jp)
This work was supported in part by the JSPS KAKENHI under Grant JP20H04154.

ABSTRACT Internet of Things and deep learning (DL) are merging into one domain and enabling outstanding technologies for various classification tasks. Such technologies are based on complex networks that mainly target powerful platforms with rich computing resources, such as servers. Therefore, for resource-constrained embedded systems, new challenges of size, performance (i.e., latency, throughput, and accuracy), and power consumption-aware networks should be addressed, particularly when edge devices handle multimodal data (i.e., different types of real-time sensing data). In this case study, we focus on DeepSense, a time-series multimodal DL framework combining convolutional neural networks (NN) and recurrent NN to process accelerometer and gyroscope data for human activity recognition. We present a field-programmable gate array (FPGA)-based acceleration for DeepSense incorporated into a hardware/software co-design approach to achieve better latency and energy efficiency using the Xilinx Vitis AI framework. The architecture of DeepSense has drawbacks that cannot be easily alleviated by Vitis AI; therefore, we introduced a new methodology of adjusting the framework and its components (i.e., the deep learning processing unit (DPU)) to achieve a custom design suitable for such time-series multimodal NN. We implemented the accelerator on two FPGA boards and performed a quantitative evaluation by varying the DPU parameter settings to support our design approach. We demonstrated the effectiveness of our implementation against the original software implementation on mobile devices by achieving up to 2.5× and 5.2× improvement in latency and energy consumption, respectively. Through this case study, we provide crucial insights into the FPGA-based accelerator design of multimodal NN and essential aspects to consider for further improvements and adaptation in other application domains.

INDEX TERMS Deep learning, sensors, multimodal, activity recognition, FPGA.

I. INTRODUCTION
In the age of the Internet of Things (IoT) and multi-sensor edge devices, embedded systems need to process big data collected in real time from different sensors whose data are defined in different types (e.g., audio, video, text, images, and human body vital signals). Such data processing also concerns neural networks (NNs), which face software and hardware challenges for embedded systems [1]. NNs that can handle multiple types of data (i.e., modalities) are called “multimodal NNs” and can yield better detection, classification, or prediction tasks than NNs dealing with a single modality–unimodal NNs [2]. Multimodal NNs were first introduced in [3], where voice and video were combined to improve speech recognition. Afterwards, more applications, such as combining text and images for image captioning [4] and text, audio, and video for sentiment analysis [5], became the focus of multimodal NN research. Multimodal NNs have their own data representation, structures, and design approaches [6]. Among various multimodal NN applications, one big trend is medicine and personal health monitoring, including human activity recognition (HAR), where vital signals with time-series data, such as heart rate, temperature, and motion signals are handled. Although the first HAR applications started with a single accelerometer sensor [7], multimodal approaches have become the norm; they employ both shallow machine learning (ML) classifiers (e.g., random forest (RF) and decision tree (DT)) and deep

The associate editor coordinating the review of this manuscript and approving it for publication was Kathiravan Srinivasan.
learning (DL) models [8] on multiple sensor data from gyroscopes, accelerometers, or magnetometers.

In this study, we focus on DeepSense [9], a multimodal DL framework applied for HAR applications that process multiple time-series data. In various applications, DeepSense outperforms other DL (i.e., deep NN (DNN) and convolutional NN (CNN)) and shallow ML models (i.e., DT, RF, support vector machine (SVM), and restricted Boltzmann machines (RBMs)) significantly. Additionally, each of these techniques has their own drawbacks; RF and DT can be computationally complex, SVM is not well suited for noisy data, RBMs are difficult and slow to train, and DNNs and CNNs are not efficient for capturing temporal relationships. Moreover, these approaches need to fuse features of multiple sensors into a single input vector [10], whereas DeepSense presents a combination of CNNs and recurrent NNs (RNNs) where sensor-specific features are extracted individually and inter-modality relationships and temporal relationships can be learned. When more sensors are added, the model’s accuracy should improve, but there may be drawbacks of the increasing network complexity, leading to an increase in computation and system power consumption. Although DeepSense’s original software implementation is aimed at low-power mobile and IoT devices, it is still unable to match with the energy efficiency of shallow models in meeting these devices’ power requirements. Therefore, we aim to further improve the energy efficiency of DeepSense by providing a hardware accelerator while maintaining its high accuracy.

To address the above issue and because field-programmable gate arrays (FPGAs) are promising competitors to graphics processing units (GPUs) in a way that can well balance speed and energy efficiency [11], we present a hybrid CPU-FPGA co-designed acceleration model for DeepSense. To the best of our knowledge, state-of-the-art works (detailed in Section II) targeted full hardware architectures on FPGA. Although such architectures provide low-power implementations, they lack the flexibility and reusability that a hardware/software (HW/SW) co-design approach provides. Since existing types of system-on-a-chip (SoC) provide a single low-power chip combining CPU and FPGA logic with fast and power-efficient interconnects, the objective of this case study is to evaluate our CPU-FPGA HW/SW approach on such SoC, discuss its limitations, and highlight aspects to be considered for further improvements. In this study, the entire system was analyzed and designed to optimize the workload, parallelization, partitioning, and multiple data transfers between multimodal NN components so that a low-latency and low-energy implementation could be achieved. By targeting these aspects, the system can also be scalable, which means that it can handle adding more sensors without significant impact on latency or energy. Moreover, combining SW with HW acceleration allows flexibility in a way that updating NN parameters (e.g., weights) or data preprocessing becomes easy and efficient when the NN is retrained or new sensors are added. We used the Xilinx Vitis AI framework and its deep learning processing unit (DPU) for our implementation as their design approaches are becoming the mainstream of NN implementations because of their rich features and acceleration efficiency, mainly in image-oriented unimodal NNs on FPGAs. Then, we implemented and performed a quantitative evaluation on different evaluation boards by varying different DPU parameters. Our implementation achieved up to 2.5× and 5.2× improvements in latency and energy, respectively, compared with the original implementation [9].

The advantage of our proposed system is that it provides a hybrid architecture which outperforms a CPU implementation in terms of latency and energy efficiency, optimizes the partitioning of a NN composed of multiple components such as DeepSense, and preserves the multimodal NN scalability and flexibility. By using such a system, we can target a wider range of complex multimodal time-series applications in the future such as healthcare applications (e.g., stress detection combining body temperature, motion, heart rate, and oxygen saturation or emotion detection combining electroencephalogram (EEG), electromyography (EMG), and photoplethysmogram (PPG)). Also, employing other types of sensors such as Wi-Fi signals, GPS, Bluetooth, and microphones can extend the application range to navigation optimization for robotics or unmanned vehicles.

In summary, the contributions of this study are as follows:

- We present an FPGA-based acceleration of a time-series multimodal DL framework, DeepSense, in a HW/SW co-design approach on a single SoC to improve latency and energy efficiency.
- To the best of our knowledge, the first attempt of employing the Xilinx Vitis AI and DPU through a new methodology to provide a customized implementation for the time-series multimodal model by studying both such a model’s properties and limitations of the Vitis AI framework is presented.
- We performed a quantitative evaluation by implementing our design on two evaluation boards and varying the DPU parameter settings. Through in-depth analyses of their impacts on the implementation’s latency, energy, and resource utilization, we provide discussions and insights about FPGA-based multimodal NN accelerator designs on embedded systems, their effectiveness against a comparable embedded CPU, system scalability, and further improvements.

The rest of this article is organized as follows. Section II presents previous works that focused on multimodal implementation on FPGA or hybrid CPU-FPGA platforms. Section III presents the architecture of DeepSense and describes Xilinx Vitis AI and DPU used for the implementation. Section IV explains our design methodology and adjustments made to both DeepSense and the Vitis AI process to achieve an efficient implementation. Section V describes our evaluation setup, results, and discussions about the implementation results and further considerations. Finally, Section VI concludes this study.
II. RELATED WORK

Although FPGA-based NN acceleration for image processing is an evolving trend, those for multimodal applications are still a new domain and are not well-studied. Reference [12] introduced a multimodal biometrics IoT security system that combines voice and video. The feature extraction and modality fusion were achieved on the CPU, whereas a connected FPGA ran an ML algorithm for classification. This system was evaluated on an Intel i7 CPU and an Intel DE5 FPGA board. Reference [13] introduced an emotion recognition platform based on physiological signals preprocessed on a RISC-V implemented on a Xilinx-7 FPGA to achieve an efficient edge IoT application.

Some literature have studied the implementation of an entire system on a single FPGA chip [14]–[18]. Reference [14] used a single Artix-7 FPGA chip for stress detection where four physiological modalities were combined by extracting their features and testing two types of ML classifiers. The best classifier achieved an accuracy of 96.7% with a power consumption of 728 mW at 200 MHz. Reference [15] targeted different modalities on the same Artix-7 FPGA by processing audio in a CNN and combining it with demographic data in a fully connected (FC) network to detect respiratory symptoms achieving accuracy of up to 87.3% with a power consumption of 240 mW at 80 MHz. SensorNet [16] presented a general multimodal data classification by a CNN implemented on the Artix-7 FPGA to test multiple applications, such as stress detection and HAR. The HAR application achieved an accuracy of 98% with a power consumption of 175 mW at 100 MHz. In addition, [17] focused on HAR, where a similar architecture to SensorNet was employed by applying a long short term memory (LSTM) RNN instead of CNN achieving a power consumption of 82 mW at 160 MHz but its accuracy declined to 87.17%. Reference [18] presented a HW/SW implementation for HAR on a Spartan-6 FPGA where software feature extraction and transformation were executed on a microblaze softcore while a two-layer NN classifier was implemented on hardware achieving a power consumption of 268 mW at 67 MHz with an accuracy of 89.2%. These works presented low-power designs due to their small-sized NNs and full hardware implementations on a single small-sized FPGA chip. However, they applied the early fusion concept where all modalities were fused into a single vector or a two-dimensional image before being processed. Therefore, their scalability and design flexibility can be challenging or inefficient when more input sensors or different applications are targeted as they need to completely redesign the multimodal NN model and reimplement its hardware accelerator.

DeepSense [9] presented a different flexible and scalable network, where sensors can be added easily, unlike the above-mentioned methods. However, the original implementation focused only on software on the CPU. Using a HW/SW approach on a CPU-FPGA SoC, the preprocessing of data and the upgrade or modification of the implemented network can be more efficient while preserving the networks’ reusability, scalability, and flexibility.

III. PRELIMINARIES

In this section, first, we briefly describe DeepSense, which is the target multimodal NN of our work and then describe the Xilinx Vitis AI and DPU, which are used for our FPGA-based accelerator implementation.

A. THE BASE MODEL: DeepSense

Multimodal NNs are promising for improving the accuracy of various complex tasks by leveraging essential information inherent in multiple modalities.

DeepSense [9] is the first open-source multimodal NN framework to solve both classification and regression problems of time-series sensor data in a unified manner on IoT and mobile devices. The goal of DeepSense is to estimate signals from noisy measurements in mobile sensing applications by collecting data for a time interval $T$. Both CNNs and RNNs are used – CNNs extract intra-interval local interactions for each modality and intra-interval global interactions among all modalities, whereas RNNs extract the intra-interval time relationships $[9]$.

As depicted in Fig. 1, DeepSense starts with three “Individual Convolutional Layers” (hereinafter, individual CNNs) at the bottom of the figure collecting data and extracting individual features of every sensor $k$ out of the total $K$ sensors. Next, three “Merge Convolutional Layers” (hereinafter, merge CNN) learn interactions between all $K$ sensors, followed by two “Recurrent Layers” (hereinafter, RNN) extracting features from time sequences. We can see two identical architectures from “Individual Convolutional Layer 1” to “Recurrent Layer 2” representing the same layers in different time steps $t$ within the $T$ interval. Finally, an FC layer “Output Layer” at the top is used for classification or regression.

In this study, we explored a basic application of heterogeneous HAR (HHAR), where $K=2$ represents data from a gyroscope and an accelerometer to detect six types of human activity (walking, standing, sitting, biking, stair up, and stair down). Various approaches for HHAR have been studied for several years, including shallow hand-crafted models and deep models. Nonetheless, DeepSense achieved better feature extraction across different users and outperformed the accuracy (by over 10%) of other methods using RF [19], SVM [19], or RBM [20] as well as networks evaluated on the same dataset in [8] based on custom DNNs and CNNs [21]. However, DeepSense still has room for improvement, particularly when implemented on embedded devices. Although the major target of DeepSense is IoT and mobile devices,
In this study, we present an FPGA-based acceleration to reduce the latency and energy consumption of DeepSense. Such an architecture will become increasingly essential to provide real-time and low-energy inferences on embedded systems for complex time-series applications (e.g., medical applications with more sensors). Our calculations based on an average embedded battery of 1,000 mAh estimated that in an ideal case where the battery holds for 24 h, we need to aim for energy consumption of less than 59 mJ.

**B. XILINX VITIS AI DEVELOPMENT STACK**

As mentioned above, customized hardware designs of FPGA-based NN accelerators have evolved remarkably in recent years. Although the first series of works employed only register-transfer level (RTL) design for individual NN nodes and layers [23], two main approaches are applied at present: (1) manually designing hardware for a target NN using hardware description languages or high-level synthesis (HLS) [24] and (2) automatically generating hardware using toolchains that incorporate ML frameworks such as Caffe and TensorFlow [25]. Along with the increasing size and complexity of NN models, approach (2) will be increasingly leveraged to become the mainstream for better design productivity. However, designers need to carefully configure the hardware parameters and optimizations that largely affect the latency and energy efficiency.

We focus on approach (2) using the Xilinx Vitis AI framework, which was developed for NN model designs targeting both server and edge FPGAs. We briefly explain the concept of Vitis AI and its computation engine – the DPU. The toolchain is available on Github\(^2\) as a complete stack for NN implementation covering both software and hardware compilation, optimization, and deployment steps. In our case study, the input is a TensorFlow model. Vitis AI quantizes the model into an 8-bit fixed-point model, then compiles it into DPU-specific instructions. The compiler determines operations supported by the DPU and delegates unsupported operations to be manually designed and deployed by the user on the CPU.

**FIGURE 1. Overview of DeepSense [9].**

**TABLE 1. Comparison of DeepSense against other methods [9].**

| Platform          | Latency (ms) | Energy (mJ) |
|-------------------|--------------|-------------|
|                   | DeepSense    | Other Methods | DeepSense    | Other Methods |
| Nexus 5           | 39           | < 8         | 220         | < 130         |
| Intel Edison      | 100          | < 20        | 700         | < 350         |

The Nexus 5 battery has a capacity of 2,300 mAh and a voltage of 3.8 V. It holds on for an average of 15 h [22], and DeepSense HHAR runs an inference every \(T = 5\) s, meaning it consumes 7.5% of the total phone energy consumption.

\(^1\)The Nexus 5 battery has a capacity of 2,300 mAh and a voltage of 3.8 V. It holds on for an average of 15 h [22], and DeepSense HHAR runs an inference every \(T = 5\) s, meaning it consumes 7.5% of the total phone energy consumption.

\(^2\)https://github.com/Xilinx/Vitis-AI
M. Trabelsi Ajili, Y. Hara-Azumi: Multimodal Neural Network Acceleration on Hybrid CPU-FPGA Architecture

FIGURE 2. DPU top-level block diagram [26].

TABLE 2. Adjustable parameters of DPU.

| Parameter          | Settings               |
|--------------------|------------------------|
| Number of DPs      | B512, B800, B1024, B1152, B1600, B2304, B3136, B4096 |
| Architecture       | Low, High              |
| RAM Usage          | On, Off                |
| DSP Usage          | Depthwise Convolution, Elementwise Multiplication, AveragePool, Softmax, LeakyReLU |
| Low Power Mode     |                        |

the APU is an ARM CPU connected to the FPGA logic through a high-speed advanced extensible interface (AXI) bus. An off-chip random access memory (RAM) is shared between the CPU and FPGA.

The DPU described in this section is the Zynq DPU v3.3 which targets edge FPGAs and supports a range of operations for CNN and FC layers only. RNNs are not supported by this DPU. The DPU can be implemented as an intellectual property core, where some parameters can be configured as in Table 2. Architecture denotes the peak operations/clock that the DPU can achieve; e.g., the smallest B512 architecture has a peak operations/clock of 512, and the largest B4096 architecture reaches a peak operations/clock of 4,096. RAM Usage represents the on-chip block RAM (BRAM) used to store NN parameters (i.e., weights and biases) and intermediate features. DSP Usage represents a resource trade-off for PE operations (i.e., using more digital signal processors (DSPs) or lookup tables (LUTs)). The Low Power Mode disables the PE clock to save power when the DPU is idle. The extra operations list optional functions supported by the DPU but can be disabled to save resources.

During the evaluation, we search for a trade-off among performance, resource utilization, and power consumption by adjusting these parameters to determine the most energy-efficient implementation for our application.

C. CHALLENGES OF VITIS AI

Vitis AI and DPU have been developed to support most standard CNN-based models (e.g., Resnet, AlexNet, and GoogleNet). In the case of a custom network, such as DeepSense, as some layers do not meet some specifications of the DPU, we confront the following three issues in accelerator design.

First, we carefully consider how to partition DeepSense and efficiently implement it on a Zynq architecture. Because the DPU we can use does not support RNNs, we need to implement the DeepSense gated recurrent units (GRUs) on the software side. In addition, we need to investigate data transfers between the CPU and FPGA logic to properly partition NN components that would result in data transfer overhead.

Second, multimodal NNs may have the size and structure of CNNs that are unusually seen in CNNs used for image processing NNs. Although the latter usually employs a kernel size of up to $5 \times 5$, DeepSense uses a kernel size of $1 \times 18$, exceeding the size limit supported by the DPU (i.e., $16 \times 16$). Moreover, the DPU is an image-oriented unimodal accelerator that was not originally designed to support multiple inputs. Hence, to implement a multimodal NN accelerator, we need to adjust the number and size of the implemented DPs to achieve the best parallelization of individual CNNs.

Finally, the critical challenge when using DPU is to find the appropriate parameter settings of DPU to implement the adequate energy-efficient architecture. This is particularly crucial for NN models comprising of multiple CNN blocks of different sizes.

IV. METHODOLOGY FOR ENERGY-EFFICIENT MULTIMODAL NN ACCELERATION

In this study, we present a hybrid CPU-FPGA HW/SW co-designed acceleration for a time-series multimodal NN of an HHAR model built on DeepSense. We adopt the Xilinx Vitis AI framework and its DPU accelerator in our design approach by studying the workload of the model, applying appropriate adjustments to the DPU, and proposing proper model partitioning.

The goal of this work is to achieve a more effective low-latency and low-energy DeepSense architecture than the original software implementation [9] to make it suitable for low-power IoT and mobile devices. We performed a quantitative evaluation of the proposed implementation on different evaluation boards by varying the parameter settings of the DPU and analyzing their impact on the implementation’s latency, energy consumption, and resource utilization. Rather than presenting a direct comparison with other related works, we aim to provide discussions and insights about such CPU-FPGA multimodal NN accelerators, as well as their effectiveness, scalability, and challenges for further improvements.

3 For the used Vitis AI 1.3, only server-class FPGAs (i.e., Xilinx Alveo) support the RNN DPU and only for LSTM.
**FIGURE 3.** Design approach.
Vitis AI provides a set of tools that we integrate into our design approach based on three phases (Fig. 3). Notably, our work does not rely on a straightforward process that the Vitis AI framework presents (i.e., using a model from the model zoo, optimizing, quantizing, and compiling it and then deploying it on the target device); rather, we target a custom design and thus make design decisions and manual adjustments to apply these tools properly.

We now elaborate on our design strategies. In Phase 1, we focus on software preprocessing where we adjust and train our network to provide a model supported by the Vitis AI tools. Then, in Phase 2, we create our DPU-based FPGA platform according to the analysis in Phase 1 and exploration of the set of parameters that the DPU provides. Finally, in Phase 3, we integrate the trained model from Phase 1 and the DPU architecture from Phase 2 in a process using the Vitis AI tools and manual adjustments to generate a design that we evaluate to determine the optimal implementation.

A. PHASE 1: SOFTWARE PREPROCESSING
In Phase 1, the software part of the NN is preprocessed. We present the network architecture of the HHAR application implemented with DeepSense after upgrading and adjusting the original model as well as a workload analysis to break down the network components. We use this information for further design decisions in Phases 2 and 3.

1) NN ADJUSTMENT AND TRAINING
DeepSense, which has been open-sourced on Github, was originally implemented using TensorFlow 1.1. In this study, we updated the code to TensorFlow 2.3.0. For HHAR, a triaxial accelerometer and triaxial gyroscope were used. For training and inference, the same dataset which was preprocessed by the original authors, was employed. Interested readers are referred to [9] for further details on data collection and preprocessing. As shown in Fig. 4, the network is formed by two CNN branches (i.e., ACC and GYRO CNNs for accelerometer and gyroscope, respectively). Their outputs are then concatenated into the MERGE CNN and further into an RNN with two GRUs. The final output is produced by an FC layer (Fig. 4).

To address the limitations discussed in Section III-C, modifications were applied to the original architecture to support the DPU: the kernel size of the first Conv2D layers of ACC, GYRO, and MERGE CNNs were modified from $1 \times 18$ to $1 \times 16$. Then, subsequent layers were adjusted to save the output shapes of the original work. This model was retrained to reach an accuracy of 0.95 which is within the range of the original DeepSense work, i.e., $0.942 \pm 0.032$ [9]. After training, a TensorFlow graph was saved and a set of calibration data was extracted from the evaluation data to be used in Phase 3.

2) WORKLOAD ANALYSIS
To break down the components of the entire model and partition their implementation for the CPU and FPGA, we analyzed the layers, their number of operations, memory requirements, and the size of data transferred between components. According to this workload analysis, we determined which parts could be accelerated using the DPU engine and how to efficiently balance the remaining components on the CPU.

Tables 3 and 4 describe the parameter breakdown for inference with Batch=1 and latency breakdown on the CPU, respectively. We considered a quantized implementation whose memory size was calculated based on an 8-bit integer instead of a 32-bit floating-point format. Although the RNN had a higher number of operations and parameters than the CNNs combined (Table 3), an experimental latency
TABLE 3. Properties of the network components.

| Network Component | ACC CNN | GYRO CNN | MERGE CNN | RNN | FC |
|-------------------|---------|----------|-----------|-----|----|
| Input shape       | $1 \times 20 \times 60 \times 1$ | $1 \times 20 \times 60 \times 1$ | $1 \times 20 \times 8 \times 64$ | $1 \times 20 \times 512$ | $1 \times 120$ |
| Output shape      | $1 \times 20 \times 4 \times 64$ | $1 \times 20 \times 4 \times 64$ | $1 \times 20 \times 512$ | $1 \times 120$ | $1 \times 6$ |
| Operational (KLOPs) | 42,372 | 42,372 | 147,844 | 400,323 | 0.720 |
| Param size (KB)   | 42.944 | 42.944 | 148.416 | 315.360 | 0.726 |
| Input size (KB)   | 1.200 | 1.200 | 10.240 | 10.240 | 0.120 |
| Output size (KB)  | 5.120 | 5.120 | 10.240 | 0.120 | 0.006 |

TABLE 4. Latency breakdown on the CPU using TensorFlow Lite with 8-bit quantization (Averaged over 1,000 Runs).

| Network Part | Average Latency (ms) |
|--------------|----------------------|
| CNNs         | 12.900               |
| RNN          | 10.987               |
| FC           | 0.242                |
| Total        | 24.134               |

breakdown showed that due to the difference in their operation types, the CNNs had higher latency than the RNN (Table 4). Based on this observation and the current DPU’s limitation of not supporting RNNs, we focused on accelerating the CNNs on the FPGA and implementing the RNN on the CPU using TensorFlow Lite. The FC layer, whose number of operations and memory requirements are not large, could be implemented either on the DPU or CPU. We leave the decision of its proper implementation until a comprehensive investigation is presented in Section IV-C2.

B. PHASE 2: HARDWARE PLATFORM DESIGN

Vitis AI provides the tools and runtime application programming interfaces (APIs) to quantize, compile, and run a model on a specifically designed platform. In Phase 2, we focus on the creation of the hardware platform to provide its specifications to Vitis AI. First, we explain the process and tools used to create our custom platform, then the parameter exploration that provides the proper configuration of the DPU, and finally the generated design, which was implemented on two evaluation boards.

1) PLATFORM CREATION PROCESS

Based on the two board models used for evaluation, we create two base designs using Vivado, where we preconfigure the processing system (PS) and programmable logic (PL). Next, we configure the operating system kernel and software components to provide the libraries and drivers required to communicate with the DPU and run the inference program. PetaLinux Tools are used to compile the kernel and build the system image. Using Vitis, we package the design created in Vivado and the software components built by PetaLinux into an implementation platform on which we will connect the DPU instances. The configuration of the DPU instances is decided after parameter exploration, as explained in Section IV-B2. Finally, we build our system with the configured DPU instances and generate a bootable image that will be loaded on the evaluation board. A description file containing the detailed architecture of the configured DPUs is generated as well and will be used by the Vitis AI Compiler in Phase 3.

2) DPU PARAMETER EXPLORATION

As introduced in Section III-B, the DPU has multiple parameters that can be configured depending on the application requirements. If we consider all parameters in Table 2 (where $N$ = DPU instances, $A$ = DPU Architecture, $R$ = RAM Usage, $D$ = DSP Usage, $P$ = Low Power Mode, and $E$ = Extra operations), the algorithmic complexity of the design space would be expressed by:

$$O(NARDPE)$$

We can reduce the complexity by eliminating fixed parameters from the search space. Thus, we first determine the number of DPUs to be instantiated. The current version of DPU does not support multiple inputs; therefore, we need two DPU instances to run the two parallel branches shown in Fig. 4. The next essential aspect of hardware accelerator implementation is determining the right parameter settings of the DPU in terms of both operations/clock and storage. From the workload analysis performed in Section IV-A2, we estimated the number of operations in all CNNs to find that the MERGE CNN had the highest workload of 147,844 operations. Therefore, we needed to select the DPU such that the MERGE CNN requirements for both the number of operations and storage could be satisfied. Among the available DPU options presented in Section III-B, the smallest DPU architecture, B512, has a peak operations/clock of 512. Based on the lowest PE frequency of the boards for our evaluation (i.e., 400 MHz), B512 could execute up to 204,800 operations/ms, which is fast enough for the 147,844 operations of MERGE CNN. Concerning the on-chip memory requirement, when setting the Low RAM Usage option for B512, 16 BRAMs (equivalent to 72 KB) are reduced, and the remaining BRAMs will provide 330.75 KB. This still suffices for MERGE CNN, where 168.896 KB for the input/output features and parameters are required. We can experimentally assess this assumption in the evaluation section. Regarding the DSPs, the High DSP Usage option uses 32 DSPs for PE accumulation operations, which is equivalent to 1,418 LUTs in Low DSP Usage. We also investigated DSP’s impact during the evaluation. Finally, based on Fig. 4, we determined the operations required for DeepSense and deactivated the support of unused functions from the DPU to save resources and power consumption.

As a result, the complexity can be reduced to:

$$O(RDP)$$
where every one of these three remaining parameters takes only two values, which leaves us with a smaller design space.

3) IMPLEMENTED DESIGN
Fig. 5 shows a simplified representation of the block design synthesized and implemented on the boards. Zynq Ultra-SCALE+ represents the PS, where the CPU and the AXI peripherals communicate with the memory controller to access the external RAM. Every DPU has a memory space in the external RAM where it reads and writes its data to be accessed by the CPU; this is the default mechanism to share data between the DPU and CPU. The PS provides one high-performance (HP) master AXI interface, which accesses the DPU registers where states and memory addresses can be stored. Each DPU has one master AXI bus for instructions and two master AXI buses for data transfer, which all connect to slave HP ports of the PS. A clock wizard receives a signal pl_clk from the PS and generates a common clock signal that connects to the dpu_aclk port of each DPU to control data and scheduling. Each DPU has a separate clock port dpu_2x_clk, which doubles the clock cycles of dpu_aclk and controls the PEs. When the Low Power Mode is selected during the configuration, dpu_2x_clk_ce (clock enable) signals from both DPUs are added and connected to the clock wizard to deactivate the PE clocks when there are no computations active and hence spare energy.

C. PHASE 3: HW/SW CO-DESIGN
In this phase, we focus on the Vitis AI tools and the process for combining the trained network from Phase 1 and the generated hardware in Phase 2 to complete the design of our inference application.

1) QUANTIZATION AND COMPILATION
The first step in this phase is to use the Vitis AI Quantizer to generate an 8-bit quantized model from the TensorFlow model we trained. The quantizer requires the saved floating-point TensorFlow graph and a small set of calibration data to provide a new quantized TensorFlow graph that can be used by the compiler. The quantizer supports a range of layers and operations. However, if some layers are not supported, an error arises. We added a new step where we adjust the graph after such an error to extract a subgraph that the quantizer can support and manually work on the implementation of the unsupported layers. The new subgraph is processed again by the quantizer until we reach a valid quantized model, which is forwarded to the compiler. In our case, the RNN was not supported by the quantizer and needed to be extracted in a new subgraph, which we converted to a TensorFlow Lite model we will integrate into the application later.

The Vitis AI Compiler uses the quantized model and DPU architecture description exported in Phase 2 to generate a Xilinx model file containing DPU instructions and a list of operations that cannot be executed on the DPU. In our application, DPU instructions for ACC, GYRO, and MERGE CNNs are generated, whereas the concatenation operation connecting the three CNNs is left to be designed manually.

2) SOFTWARE DESIGN, SYSTEM PARTITIONING, AND SCHEDULING
In this step, we design the inference program and optimize the partitioning of the network between the CPU and FPGA. Previous steps provided DPU instructions that need to be mapped to DPU instances, TensorFlow Lite models that can be deployed on the CPU, and other operations that need to be defined manually. For flexible and fast prototyping, we used Python libraries and the Vitis AI runtime API to design our inference application.

The crucial aspect to be considered for the hybrid CPU-FPGA architecture is the bandwidth and latency of data transfers. Although most operations can be accelerated more than $10 \times$ by the DPU, the delay in writing and reading the data to and from the memory between the CPU and FPGA can be a bottleneck if the CNNs deployed on the DPU are small [27]. This situation concerns the final FC layer, which is a unique shallow layer with few input-to-output connections. Thus, executing such a shallow layer on the CPU would be more beneficial than delegating it to the DPU. Specifically, we compare in Table 5 the average execution time of the FC layer on the DPU excluding data transfers against that on the DPU including data transfers, and that on the CPU (using TensorFlow Lite). As shown in the table, although the DPU has a shorter latency than the CPU, the overhead caused by the data transfer is $5 \times$ more than the computation time. Hence, in DeepSense, it is more efficient to implement the final FC layer on the CPU.

We finally design our software, whose execution is presented in Fig. 6, and integrate it in the bootable image generated in Phase 2 to be deployed on the evaluation boards. Fig. 6a shows a simplified representation of our implementation on the Xilinx Zynq UltraScale+ MPSoC chip, whereas the timeline in Fig. 6b shows blocks in the PS and PL as their respective computation units work along with the data transfers between them. On the CPU, first, the INIT phase loads the DPU instructions for the CNNs and the TensorFlow Lite models for the RNN (i.e., GRU) and FC components. Next, the evaluation data are loaded in the LOAD DATA phase. The inference starts when the CPU feeds the inputs into the two DPUs running in parallel to execute the ACC and GYRO CNNs. The output data from the DPUs are written back to the CPU and concatenated using the manually implemented CONCAT function. Then, the concatenated data is fed into the MERGE CNN mapped to DPU 1, which writes back its

| Computation Unit | Average Latency of the FC Layer (ms) |
|------------------|-------------------------------------|
| DPU             | 0.167                               |
| DPU + transfer  | 1.004                               |
| CPU             | 0.242                               |
results to the CPU. The GRU and FC TensorFlow Lite models run sequentially on the CPU and provide the final output of the inference.

3) EVALUATION METHOD FOR OPTIMAL DESIGN DECISION
In this case study, we present a quantitative evaluation of the DPU implementation to determine the most efficient design in terms of latency and energy consumption. The complete flow we presented explains the design and on-board inference execution of one specific DPU configuration. During the parameter exploration of the DPU in Section IV-B2, we determined some fixed parameters suitable for our model, e.g., the number of DPUs and their architecture, and we assumed the required BRAM usage. We go through the process of DPU configuration, design generation, and network compilation again (steps within the light gray box in Fig. 3) to generate more board images with new DPU configurations by varying the RAM Usage, DSP Usage, and Low Power Mode. Within the Evaluation box in Fig. 3, we vary only one parameter for each new DPU configuration and observe its impact on the design for
service-level agreement. The detailed evaluation performed in Section V assesses the influence of these parameters and we can select the configuration and board that provide the most efficient design.

V. EVALUATION

This section first describes our evaluation setup and then provides results and insights obtained from the results.

A. SETUP

Aiming to achieve low-latency and low-energy implementation of a multimodal NN architecture, our evaluation targeted the Avnet Ultra96-V2 evaluation board and Xilinx ZCU102 as small-scale and medium-scale devices, respectively. Both boards integrate a Zynq UltraScale+ MPSoC family chip that contains the same CPU on the PS (Table 6) but different PLs (Table 7). As explained in Section IV-C3, we generated and implemented different DPU configurations on the two boards to evaluate the most energy-efficient design by varying some DPU parameters. For Ultra96-V2 and ZCU102, in total four and two configurations described in Table 8 were considered, respectively, by varying the on-chip RAM Usage, the DSP Usage, and the Low Power Mode of the DPU. We evaluated all explored configurations based on the design flow in Fig. 3 by selecting two instances of the B512 architecture, as concluded in Section IV-B2, and following a logical parameter selection approach; we first investigated the impact of RAM and DSP on latency, energy consumption, and resource utilization, then set their best settings to further investigate the energy efficiency gained by applying the Low Power Mode on Ultra96-V2. Finally, we compared the best configuration tested on Ultra96-V2 with the same parameter settings on ZCU102 while alternating the Low Power Mode.

We evaluated our implementations listed in Table 8 against the original DeepSense implemented on the Nexus 5 smartphone (hereinafter “Baseline”) because it showed better performance than the Intel Edison IoT board in [9]. Because TensorFlow Lite is becoming the main inference tool for most embedded and IoT devices, we also evaluated a TensorFlow Lite model run on a low-power embedded platform, the Raspberry Pi 3 Model B, which has the same CPU architecture as Zynq Ultrascale+ MPSoC. For this device, two setups were considered: “RPi3-f” using a 32-bit floating-point TensorFlow Lite model and “RPi3-q” using an 8-bit quantized TensorFlow Lite model. Table 6 lists the specifications of the CPUs on the Nexus 5 and Raspberry Pi 3 Model B as well as those on Ultra96-V2 and ZCU102.

For the implementations compared above, our evaluation considered the following metrics:

- Latency [ms]: We used a software timing function to measure inference time. Because latency is nondeterministic depending on the system status and memory accesses, 1,000 inferences were performed on the evaluation data from the same dataset mentioned in Section IV-A1 to calculate the average inference time.
- Energy per inference [mJ]: We calculated the energy consumption according to the latency and average power consumption during an inference. Both Ultra96-V2 and ZCU102 have a power management bus (PMBus), which is connected to the main voltage rails of the chip. We measure the power in real-time using a software tool that accesses the PMBus currents and voltages through the operating system running on the PS.
- Resource utilization [%]: We compared the resource utilization (i.e., LUT, flip-flop (FF), BRAM, and DSP) of the different boards using different configurations. This metric was evaluated only for our hybrid CPU-FPGA implementations.
- Accuracy [%]: We compared the output of every inference with the expected label to obtain the percentage of correct inferences over the total number of evaluated data.

B. RESULTS

1) LATENCY AND ENERGY CONSUMPTION

In Sections IV-B2 and IV-C2, we qualitatively determined the required DPU settings for an efficient implementation. During this evaluation, we examined the impact of setting changes on latency and energy consumption. Figs. 7 and 8 describe the latency and energy consumption per inference of all the compared implementations, respectively.

First, we consider the CPU-based software implementation results (i.e., Baseline, RPi3-f, and RPi3-q). As shown in Fig. 7, the latency of RPi3-f doubled that of Baseline. This can be explained by the Raspberry Pi 3 CPU having half the clock frequency of the Nexus 5 CPU. In terms of energy consumption...
efficiency. Fig. 8 shows that RPi3-f consumes less energy than Baseline despite the higher latency due to its low-power architecture. RPi3-q shows further improvements compared with Baseline. Although the latency is almost the same as that of Baseline, the energy is improved by 3.2×. Using a floating-point model in Baseline degrades the performance compared to quantized models, as can be seen by comparing software quantization in RPi3-q against RPi3-f; the former reduced the latency and energy by 2.17× and 2.28×, respectively, than the latter. From these results, we can conclude that software quantization significantly reduces the latency and energy (by half in this case study).

Next, the impacts of BRAM Usage, DSP Usage, and the Low Power Mode are disclosed by comparing U96-1 and U96-2, U96-2 and U96-3, and U92-3 and U96-4, respectively. Although Fig. 7 shows that the latency is unaffected, Fig. 8 shows the energy consumption changes by these parameters other than DSP Usage (i.e., comparison of U96-2 and U96-3). A barely visible reduction of 1.95 mJ between U96-1 and U96-2 can confirm our expectation about saving BRAMs, whereas a noticeable reduction of 6.05 mJ between U96-3 and U96-4 proves the efficiency of the Low Power Mode. Among these four configurations on the same Ultra96-V2 board, we confirmed that by adjusting the right parameters of the DPU, U96-4 achieved a significant energy reduction of 8 mJ than U96-1. In comparison, because the ZCU102 board has a higher frequency than the Ultra96-V2 board, from Fig. 7, ZCU-1 and ZCU-2 achieved the lowest latency. By activating the Low Power Mode, the energy consumption of ZCU-2 is reduced to reach the same energy consumption of U96-1 but not enough to compete with U96-4.

To summarize our findings from the above observations, the latency is not significantly affected by modifying the BRAM, DSP or Low Power Mode parameters. The latency improvement observed between the U96 and ZCU configurations can only be explained by the faster clock frequency supported by the ZCU102 board. Regarding the energy reduction, the resource reduction by choosing a smaller BRAM in U96-2 and U96-3 than U96-1 showed a small energy reduction, whereas a more significant reduction in U96-4 was achieved due to the clock enable signals explained in Section IV-B3 which deactivate the PEs when the DPU has no active calculations. Similarly the energy reduction observed between ZCU-1 and ZCU-2 is explained by the addition of the clock enable signals.

Overall, ZCU-2 showed the best latency, namely 2.9× faster than the Baseline and RPi3-q and 1.2× faster than U96-4. However, its energy consumption was 1.1× higher than that of U96-4. U96-4 showed a latency reduction of 2.5× against Baseline and RPi3-q and an energy reduction of 5.2× and 1.6× against the Baseline and RPi3-q, respectively. From these results, we can consider that U96-4 yields the best trade-off between latency and energy consumption to reach our initial goal of achieving the DeepSense HHR task’s energy of less than 59 mJ. We even reached 42 mJ for the optimized U96-4 architecture.

2) RESOURCE UTILIZATION
Next, we compare the resource utilization to analyze the capacity of the tested boards and the scalability of the FPGA implementation of DeepSense in future cases where more sensors are deployed. For each hybrid CPU-FPGA implementation, the utilization ratio of each resource (i.e., LUT, FF, BRAM, and DSP) against the available resources in Table 7 is shown in Fig. 9.

Among the four U96 implementations, the BRAM and DSP utilization largely varied according to the parameters applied to the DPU. By contrast, the LUT and FF utilization was almost stable at approximately 80% and 60%, respectively. Because we deployed two B512 DPUs in our implementations, Fig. 9 implies that one DPU uses 40% of LUTs
and 35%–42% of BRAMs, meaning that these values represent the bottleneck to implementing any additional DPU because B512 is the smallest architecture.

In comparison, the ZCU102 board has more capacity to implement additional DPUs with even larger architectures. The two implemented B512 DPUs used less than 20% of the available resources, implying that we can theoretically implement up to 10 B512 DPUs\footnote{Practically, the used Vitis AI 1.3 supports only up to four DPUs} or select larger DPU architectures if we need a more complex CNN implementation. Insights about implementation scalability are further discussed in Section V-C.

Regarding the relationship between resource utilization and energy consumption in Section V-B1, the former had a small impact on the latter. Comparing U96-1 and U96-2, a reduction of 15% in BRAMs led to a 1.95 mJ energy reduction. A 17% DSP increase in U96-3 resulted in a 5% LUT reduction with no significant impact on energy consumption. Activating the Low Power Mode does not show any difference in resource utilization in U96-4, other than a negligible increase in LUTs. In U96-4, new clock enable signals are added to control the DPU activity, contributing to a non-negligible energy reduction. The same observation is valid for ZCU-1 and ZCU-2.

### 3) ACCURACY

In Table 9, we compare the accuracy of the original TensorFlow trained model, TensorFlow Lite quantized model, and DPU quantized model employed for Baseline, RPi3-q, and U96/ZCU implementations, respectively. Notably, all FPGA configurations used the same 8-bit quantization scheme. The TensorFlow Lite model also uses default 8-bit quantization.

From the table, the accuracy depends on the quantization and not on the used DPU configuration. It is known that applying quantization to NN parameters always impacts the data representation and thus accuracy. Vitis AI provides an 8-bit quantization for TensorFlow models. Using 8-bit fixed-point representations for activations and weights can further reduce the computational complexity and thus the hardware implementation complexity on the DPU, calculation speed, and memory requirements, resulting in lower resource utilization and lower power consumption than using 32-bit floating-point operation units. The achieved accuracy in our implementation had some accuracy loss but is still acceptable, considering other methods in [1], [28]. Therefore, we can consider that an 8-bit data representation is suitable for time-series data using DeepSense.

### C. DISCUSSIONS

In this study, we evaluated a multimodal NN model accelerated on an FPGA against CPU-based software implementation. As discussed above, our hybrid CPU-FPGA architecture successfully improved both latency and energy consumption, making it suitable for low-power embedded systems. Our implementation provides the flexibility of software and the efficiency of hardware but still has some aspects to be considered, especially when targeting more complex multimodal NNs.

**Chip size selection:** To implement basic multimodal applications, such as HHAR applications, on a small embedded device, a chip with the size of the Zynq Ultrascale+ MPSoC ZU3EG (implemented in Ultra96-V2) or even smaller is an adequate platform to achieve the required computations with low energy consumption. Larger chips, such as the Zynq Ultrascale+ MPSoC ZU9EG (implemented in ZCU102), provide faster computations and more design freedom but incur more power consumption. Such powerful platforms are thus suitable for large-scale multimodal NNs, such as image and video processing, rather than time-series signals.

**Appropriate DPU adjustment:** In addition to selecting an adequate platform, the adjustment of the DPU configuration is an essential factor to fulfill the application requirements under the chip’s constraints. From our evaluation, we can conclude that the Low Power Mode of the DPU is a necessary option for embedded devices, as it can achieve energy reduction without influencing resource utilization and latency. Other DPU settings (i.e., selecting the DPU architecture, on-chip memory size, and number of DPUs) highly depend on the target application workload. From the energy-efficiency perspective, the smallest DPU architecture with the smallest memory setting should be selected among the combinations that can fulfill the application requirements. In addition, the number of DPUs should be carefully decided as it plays a key role in parallelization and resource utilization, particularly

| TABLE 9. Comparison of the accuracy. |
|--------------------------------------|
| Baseline    | RPi3-q   | U96/ZCU  |
| Accuracy    | 93.31%   | 94.69%   | 94.19%   |

VOLUME 10, 2022
when targeting larger-scale multimodal NNs using numerous sensors.

**Scalability:** Unlike other multimodal NNs that require complete customization of the model for a particular task, DeepSense is a flexible multimodal NN that only needs to add CNN branches and extend the merge CNN when introducing more sensors [9]. From its hardware acceleration perspective, the scalability to handle the increasing size and complexity of multimodal NNs depends on the number of parallel DPsUs that can be added. As discussed in Section V-B2, using a chip with the size of Ultra96-V2 is challenging when an application requires more than two sensors because it is not possible to implement more than two DPU instances. One solution is to rely on software-based scheduling to map the individual CNN branch of every sensor on an available DPU and maximize the parallelization in such a way that all CNN branches can execute as fast as possible. However, by increasing the number of sensors, the total execution time of all CNN branches is expected to become a bottleneck. An alternative solution is to use a larger chip (e.g., the ZCU102 chip) that can deploy additional DPsUs to benefit from parallelism. Although enlarging the design will increase power consumption, the gain in latency reduction achieved by more parallelization may compensate for energy efficiency and accelerate the entire NN model.

**HW/SW partitioning:** NN acceleration on a hybrid CPU-FPGA architecture also depends on the HW/SW partitioning of NN components and the data transfers between them through shared memory. As investigated in Section IV-C2, the basic design strategy is to keep simple computations on the CPU while offloading heavy computations on the DPU. In addition to such workload analyses, to determine the appropriate partitioning of the entire NN model, memory access bottleneck should be carefully analyzed and considered for complex NN models whose NN components (i.e., CNN, RNN, and FC) run across the CPU and DPU so that data transfers can be suppressed. We concluded in our design that the FC layer runs faster on the CPU due to the data transfer overhead, although the calculation is 1.4× faster on DPU. However, the FC layer latency can be improved by changing the DPU data transfer mechanism or using a different RTL/HLS implementation relying only on on-chip memory. In a future direction, efficient memory hierarchies should be studied to bypass the external shared memory and boost latency and energy efficiency.

**RNN acceleration:** Finally, our current approach focused mainly on accelerating the CNN parts by the DPU because Table 4 disclosed that CNNs had a latency exceeding the RNN in software implementation. After applying the CNN acceleration, the RNN became the new bottleneck to be targeted for acceleration. In existing works, RNNs are also candidates for acceleration on FPGA [29], and their accelerators present better energy efficiency than CPU/GPU [30]. Unlike these works, our challenge is to achieve an efficient implementation alongside the DPU and extend the HW/SW interface to integrate the RNN in the Vitis AI environment. This will be the main subject of our future work.

**VI. CONCLUSION**

Accelerating a multimodal NN framework using time-series sensor data for low-power embedded systems is challenging. Multimodal NN models combining CNN, RNN, and FC (e.g., DeepSense) have recently been studied to target IoT and mobile applications but still have room for improvement in terms of both computing and energy efficiency. In this article, we conducted a case study of a multimodal NN acceleration targeting DeepSense and aimed to achieve a hybrid CPU-FPGA co-designed architecture for an HHAR application based on accelerometer and gyroscope data. We introduced a three-phase methodology using the Xilinx Vitis AI framework and its DPU. First, we focused on NN software preprocessing and its workload analysis, then the hardware design exploration by considering different configurations of the DPU accelerator, and finally, the HW/SW co-design by adjusting the Vitis AI tools to support such a multimodal NN. We mainly targeted CNN acceleration, partitioned other components on the CPU, and performed a quantitative evaluation on two different boards. In our evaluation, we demonstrated that our hybrid CPU-FPGA co-design approach can improve the inference latency and energy consumption by 2.5× and 5.2×, respectively, compared with the original software implementation on a mobile phone’s CPU. We also provided useful insights for future research directions for FPGA-based multimodal NN accelerator designs. The subject of our future work is to achieve hardware acceleration of the RNN for further latency and energy improvement. We also aim to customize a more complex application using more than two sensors to demonstrate the scalability of our implementation.

**REFERENCES**

[1] B. L. Deng, G. Li, S. Han, L. Shi, and Y. Xie, “Model compression and hardware acceleration for neural networks: A comprehensive survey,” *IEEE Int. Conf. Big Data*, vol. 8, no. 4, pp. 485–532, Apr. 2020.

[2] R. Ramachandram and G. W. Taylor, “Deep multimodal learning: A survey on recent advances and trends,” *IEEE Signal Process. Mag.*, vol. 34, no. 6, pp. 96–108, Nov. 2017.

[3] J. Ngiam, A. Khosla, M. Kim, J. Nam, H. Lee, and A. Y. Ng, “Multimodal deep learning,” in *Proc. 28th Int. Conf. Mach. Learn.*, 2011, pp. 689–696.

[4] N. Srivastava and R. S. Salakhutdinov, “Multimodal learning with deep Boltzmann machines,” in *Proc. 25th Int. Conf. Neural Inf. Process. Syst.*, vol. 2, 2012, pp. 2222–2230.

[5] A. Zadeh, R. Zellers, E. Pincus, and L.-P. Morency, “Multimodal sentiment intensity analysis in videos: Facial gestures and verbal messages,” *IEEE Intell. Syst.*, vol. 31, no. 6, pp. 79–88, Nov. 2016.

[6] T. Baltrusaitis, C. Ahuja, and L.-P. Morency, “Multimodal machine learning: A survey and taxonomy,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 41, no. 2, pp. 423–443, Feb. 2019.

[7] Y. Chen and X. Xue, “A deep learning approach to human activity recognition based on single accelerometer,” in *Proc. IEEE Int. Conf. Syst., Man, Cybern.*, Oct. 2015, pp. 1488–1492.

[8] V. Radu, C. Tong, S. Bhattacharya, N. D. Lane, C. Mascolo, M. K. Marina, and F. Kawar, “Multimodal deep learning for activity and context recognition,” *Proc. ACM Interact. Mobile, Wearable Ubiquitous Technol.*, vol. 1, no. 4, pp. 1–27, 2018.
[9] S. Yao, S. Hu, Y. Zhao, A. Zhang, and T. Abdelzaher, “DeepSense: A unified deep learning framework for time-series mobile sensing data processing,” in Proc. 26th Int. Conf. World Wide Web, Apr. 2017, pp. 351–360.

[10] Y. Tian, X. Wang, L. Chen, and Z. Liu, “Wearable sensor-based human activity recognition via two-layer diversity-enhanced multiclassifier recognition method,” Sensors, vol. 19, no. 9, p. 2039, Apr. 2019.

[11] K. Guo, S. Zeng, J. Yu, Y. Wang, and H. Yang, “[DL] A survey of FPGA-based neural network inference accelerators,” ACM Trans. Reconfigurable Technol. Syst., vol. 12, no. 1, pp. 1–26, 2019.

[12] O. Olazabal, M. Gofman, Y. Bai, Y. Choi, N. Sandico, S. Mitra, and K. Guo, S. Zeng, J. Yu, Y. Wang, and H. Yang, “A survey of FPGA-based neural network inference accelerators,” ACM Trans. Reconfigurable Technol. Syst., vol. 12, no. 1, pp. 1–26, 2019.

[13] A. Jafari, A. Ganesan, C. S. K. Thalisetty, V. Sivasubramanian, T. Oates, and T. Mohsenin, “SensorNet: A scalable and low-power deep convolutional neural network for multimodal data classification,” IEEE Trans. Sensor Syst., vol. 1, no. 1, pp. 274–287, Jan. 2019.

[14] A. N. Mazumder, H.-A. Rashid, and T. Mohsenin, “An energy-efficient low power LSTM processor for human activity monitoring,” in Proc. IEEE 33rd Int. Syst.-Chip Conf. (SOCC), Sep. 2020, pp. 54–59.

[15] K. Basterretxea, J. Echanobe, and I. D. Campo, “A wearable human activity recognition system on a chip,” in Proc. Conf. Design Archit. Signal Image Process., Oct. 2014, pp. 1–8.

[16] A. Sisen, H. Blunck, S. Bhattacharya, T. S. Pretont, M. B. Kjergaard, A. M. Dey, T. Sonne, and M. M. Jensen, “Smart devices are different: Assessing and Mitigating Mobile sensing heterogeneities for activity recognition,” in Proc. 13th ACM Conf. Embedded Networked Sensor Syst., Nov. 2015, pp. 127–140.

[17] S. Bhattacharya and N. D. Lane, “From smart to deep: Robust activity recognition on smartwatches using deep learning,” in Proc. IEEE Int. Conf. Pervasive Comput. Commun. Workshops (PerCom Workshops), Mar. 2016, pp. 1–6.

[18] S. Minzner, P. Schmidt, A. Reiss, M. Hanselmann, R. Stiefelhagen, and R. Durichen, “CNN-based sensor fusion techniques for multimodal human activity recognition,” in Proc. ACM Int. Symp. Wearable Comput., Sep. 2017, pp. 158–165.

[19] C. Min, J. Song, C. Yoo, S. Kang, Y. Lee, S. Lee, P. Park, C. Lee, and S. Choi, “Sandra helps you learn: The more you walk, the more battery your phone drains,” in Proc. ACM Int. Joint Conf. Pervasive Ubiquitous Comput. (UbiComp), 2015, pp. 421–432.

[20] G.-P. K. Economou, E. P. Mariato, N. M. Economopoulos, D. Lymberopoulos, and C. E. Goutis, “FPGA implementation of artificial neural networks: An application on medical expert systems,” in Proc. 4th Int. Conf. Microelectron. Neural Netw. Fuzzy Syst., 1994, pp. 287–293.

[21] A. Shavalia, S. M. Sait, and A. El-Mahdy, “FPGA-based accelerators of deep learning networks for learning and classification: A review,” IEEE Access, vol. 7, pp. 7823–7859, 2019.

[22] S. I. Venieris, A. Kouris, and C.-S. Bouganis, “Toolflows for mapping machine learning algorithms on FPGAs,” in Proc. 4th Int. Conf. Microelectron. Neural Netw. Fuzzy Syst., 1994, pp. 287–293.

[23] J. Zhu, L. Wang, H. Liu, S. Tian, Q. Deng, and J. Li, “An efficient task assignment framework to accelerate DPU-based convolutional neural network inference on FPGAs,” IEEE Access, vol. 8, pp. 83224–83237, 2020.

[24] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, “Quantization and training of neural networks for efficient integer-arithmetic-only inference,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit., Jun. 2018, pp. 2704–2713.

[25] S. Mittal and S. Umesh, “A survey on hardware accelerators and optimization techniques for RNNs,” J. Syst. Archit., vol. 112, Jan. 2021, Art. no. 101839.

[26] E. Nurvitadhi, J. Sim, D. Sheffield, A. Mishra, S. Krishnan, and D. Marr, “Accelerating recurrent neural networks in analytics servers: Comparison of FPGA, CPU, GPU, and ASIC,” in Proc. 26th Int. Conf. Field Program. Log. Appl. (FPL), Aug. 2016, pp. 1–4.

MEHDI TRABELSI AJILI (Member, IEEE) received the B.Sc. and M.Sc. degrees in electrical engineering and information technology from Technische Universität München (TUM), Germany, in 2012 and 2014, respectively. He is currently pursuing the Ph.D. degree with the School of Engineering, Tokyo Institute of Technology, Japan. He worked in several departments, TUM, as a Research Intern, between 2011 and 2014, in the fields of real-time systems and hardware/software co-design. Between 2015 and 2017, he worked as an Embedded Engineer in automotive development with iSYS RTS, Munich, Germany. In 2017, he worked as an Electrical Engineer Specialist in embedded and server development with Continental GmbH, Schwalbach, Germany. His research interests include FPGA design, deep learning, and hardware/software co-design for embedded systems.

YUKO HARA-AZUMI (Member, IEEE) received the Ph.D. degree in information science from Nagoya University, Japan, in 2010. She was a JSPS Postdoctoral Research Fellow with Ritsumeikan University, from 2010 to 2012, where she was also a Visiting Scholar with the University of California at Irvine, Irvine, USA, and the Karlsruhe Institute of Technology, Germany. In 2012, she joined the Nara Institute of Science and Technology, as an Assistant Professor. Since 2014, she has been with the Tokyo Institute of Technology, where she is currently an Associate Professor. Her research interests include system-level design automation, especially on high-level and logic synthesis, microprocessor architectures, and hardware/software co-design for embedded/IoT systems. She currently serves as an Organizing and Program Committees of several premier conferences, including DAC, ICCAD, DATE, CASES, ASP-DAC, and FPL.