Low-Cost FPGA Implementation of Deep Learning-Based Heart Sound Segmentation for Real-Time CVDs Screening

Daniel Enériz, *Graduate Student Member, IEEE*, Antonio J. Rodriguez-Almeida, Himar Fabelo, Samuel Ortega, Francisco J. Balea-Fernandez, Gustavo M. Callico, *Senior Member, IEEE*, Nicolás Medrano, *Senior Member, IEEE*, and Belén Calvo, *Senior Member, IEEE*

***Abstract*— The development of real-time, reliable, low-cost automatic phonocardiogram (PCG) analysis systems is critical for the early detection of cardiovascular diseases (CVDs), especially in countries with limited access to primary health care programs. Once the raw PCG acquired by the stethoscope has been preprocessed, the first key task is its segmentation into the fundamental heart sounds. For this purpose, an optimized hardware implementation of the segmentation algorithm is essential to attain a computer-aided diagnostic system based on PCGs. This article presents the optimization of a U-Net-based segmentation algorithm for its implementation in a low-end field-programmable gate array (FPGA) using low-resolution fixed-point data types. The optimization strategies seek to reduce the system latency while maintaining a constrained consumption of FPGA resources, aiming for a real-time response from the stethoscope data acquisition to the CVD detection. Experimental results prove a 64% decrease in latency compared to a baseline version, a 3.9% reduction of block random access memory (BRAM), which is the limiting resource of the design, and a 70% reduction in energy consumption. To the best of our knowledge, this is the first work to exhaustively study different optimization strategies for implementing a large 1-D U-Net-based model, achieving real-time fully characterized performance.**

***Index Terms*— Cardiovascular disease (CVD) detection, computer-aid diagnostic, convolutional neural networks (CNNs), deep learning, edge AI, embedded systems, field-programmable gate array (FPGA), heart sound segmentation.**

Manuscript received 15 December 2023; revised 26 March 2024; accepted 29 March 2024. Date of publication 22 April 2024; date of current version 6 May 2024. This work was supported by FEDER, UE Projects under Grant PID2019-106570RB-I00/AEI/10.13039/501100011033, Grant PID2022-138785OB-I00/AEI/10.13039/501100011033, and Grant PID2020-116417RB-C42/AEI/10.13039/501100011033. The work of Daniel Enériz was supported by Gobierno de Aragón under Grant BOA20201210014. The work of Antonio J. Rodriguez-Almeida was supported in part by the Pre-Doctoral Fellowship through Agencia Canaria de Investigación, Innovación y Sociedad de la Información (ACIISI), Consejería de Economía, Conocimiento y Empleo, Gobierno de Canarias; and in part by the European Social Fund (FSE) under Grant POC 2014-2020, Eje 3 Tema Prioritario 74 (85%). The work of Himar Fabelo was supported by the European Union NextGenerationEU/PRTR under Grant FJC2020-043474-I and Grant MCIN/AEI/10.13039/501100011033. The Associate Editor coordinating the review process was Dr. Mohamad Forouzanfar. *(Daniel Enériz and Antonio J. Rodriguez-Almeida are co-first authors.) (Corresponding author: Nicolás Medrano.)*

Daniel Enériz, Nicolás Medrano, and Belén Calvo are with the Aragon Institute of Engineering Research, University of Zaragoza, 50018 Zaragoza, Spain (e-mail: eneriz@unizar.es; nmedrano@unizar.es; becalvo@unizar.es).

Antonio J. Rodriguez-Almeida and Gustavo M. Callico are with the Research Institute for Applied Microelectronics, Universidad de Las Palmas de Gran Canaria, 35001 Las Palmas de Gran Canaria, Spain (e-mail: aralmeida@iuma.ulpgc.es; gustavo@iuma.ulpgc.es).

Himar Fabelo is with the Research Institute for Applied Microelectronics, Universidad de Las Palmas de Gran Canaria, 35001 Las Palmas de Gran Canaria, Spain, and also with Fundación Canaria Instituto de Investigación Sanitaria de Canarias, 35012 Las Palmas de Gran Canaria, Spain (e-mail: hfabelo@iuma.ulpgc.es).

Samuel Ortega is with the Research Institute for Applied Microelectronics, Universidad de Las Palmas de Gran Canaria, 35001 Las Palmas de Gran Canaria, Spain, and also with Norwegian Institute of Food, Fisheries and Aquaculture Research, 9019 Tromsø, Norway (e-mail: sortega@iuma.ulpgc.es).

Francisco J. Balea-Fernandez is with the Research Institute for Applied Microelectronics and the Department of Psychology, Sociology and Social Work, University of Las Palmas de Gran Canaria, 35001 Las Palmas de Gran Canaria, Spain (e-mail: fbalea@cop.es).

Digital Object Identifier 10.1109/TIM.2024.3392271

I. Introduction

IN 2019, 17.9 million people died due to cardiovascular diseases (CVDs), the leading cause of death, with 32% of deaths worldwide [1]. More than three-quarters of these CVD deaths occurred in low and middle-income countries, where people with risk factors often do not have access to primary health programs for early detection and treatment. Moreover, cardiac auscultation performed by a medical doctor using a stethoscope, which is the fundamental method for CVD screening, is challenging to learn, resulting in only 20% of cardiac events being detected by internal medicine and family practice residents [2]. These two factors have motivated the development of automatic phonocardiogram (PCG) analysis in recent years [3], [4], as a computer-aided decision system based on auscultation would lead to improved accuracy and shorter diagnostic times, thus facilitating the referral of patients to cardiology doctors. Therefore, a system such as a processing unit that automatically analyses the PCG in real-time attached to the traditional stethoscope can be a feasible solution to provide a more efficient CVD screening process.

PCGs are recordings of the heart sounds made during its mechanical and physiological activity, resulting from the opening and closure of the cardiac valves. As drawn in Fig. 1, two main sounds, $S1$ and $S2$, are produced when the atrioventricular and the semilunar valves close, respectively. These sounds define the duration of the cardiac cycle, which is divided into two periods: *systole* and *diastole*. Apart from the fundamental sounds ($S1$ and $S2$), additional

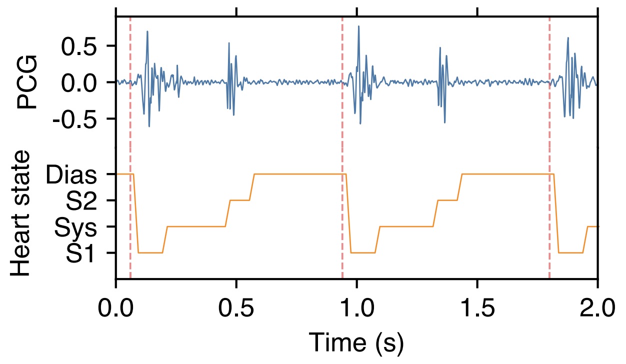

Fig. 1. Illustrative example of a PCG signal, where the fundamental sounds $S1$ and $S2$ and the systole (Sys) and diastole (Dias) intervals are labeled. The limits of each cardiac cycle are also marked with vertical lines.

sounds can appear in the PCGs. These sounds are often related to cardiac murmurs, possibly associated with CVDs. Recognizing and describing these murmurs in a first screening is crucial to deciding whether a patient must be referred to a cardiologist [5]. Please note that Fig. 1 shows an illustrative example. In medical practice, PCG signals are mixed with different noise sources such as the patient's breathing, skin contact with the stethoscope, background conversations, etc.

In recent years, several studies have addressed different tasks in the heart-sound analysis field by employing a wide variety of algorithms. One of the most basic tasks is the segmentation of the PCG, that is, the recognition of its fundamental components: $S1$, *systole* interval, $S2$, and *diastole* interval, as shown in Fig. 1. In [6], a segmentation algorithm based on a duration-dependent hidden Markov model (DHMM) was presented, first introducing an explicit model of heart-sound time duration. Based on this work, another hidden semi-Markov model (HSMM) was introduced by Springer et al. [7], which uses logistic regression in the model probabilities and additional input features, achieving significant improvements. An additional logistic regression segmentation algorithm based on the HSMM was presented in [8], which uses adaptive sojourn time parameters.

Finally, inspired by the success of U-Net [9] in image segmentation, the work presented in [10] introduces the use of convolutional neural networks (CNN) for heart-sound segmentation. The analysis of performance included in [10], showed that the U-Net-based model outperformed the existing reference algorithms to date (see [6], [7], and [8]), establishing the current state-of-the-art in this field. A summary of this analysis comparing the algorithms in [6], [7], and [10] is available in Table I, where the U-Net-based model outperforms in three different metrics. The adaptive sojourn temporal modeling described in [8] is also evaluated in [10], with slight improvements in sensitivity. For these reasons, the U-Net-based model described in [10] was the one selected to be implemented in this work.

A critical aspect that must be considered when developing an automatic PCG analyzer is the hardware running the algorithm and its time response. Because computer-aided diagnostic systems must be real-time responsive, an optimized algorithm implementation is required, especially in

TABLE I
PERFORMANCE COMPARISON WITH SEGMENTATION ALGORITHMS PROPOSED IN THE LITERATURE. DATA EXTRACTED FROM [10, TABLE I]

N	A_R (%)	S (%)	P_+ (%)	
64	82.5±2.8	83.3±3.5	87.3±3.1	Schmidt *et al.* [6]
	86.0±2.5	87.9±3.2	90.8±2.7	Springer *et al.* [7]
	91.5±1.6	**91.2±2.3**	**94.1±2.1**	Renna *et al.* [10]
128	84.5±2.8	87.1±3.7	89.7±3.8	Schmidt *et al.* [6]
	87.2±2.1	89.8±3.2	92.1±2.8	Springer *et al.* [7]
	92.6±1.6	**92.7±2.0**	**95.6±2.0**	Renna *et al.* [10]
256	85.4±3.4	89.4±4.6	91.1±3.8	Schmidt *et al.* [6]
	88.1±2.4	91.6±3.2	93.2±2.7	Springer *et al.* [7]
	93.0±1.7	**94.3±1.9**	**95.4±2.0**	Renna *et al.* [10]
512	87.4±2.6	92.7±3.1	93.3±2.8	Schmidt *et al.* [6]
	89.8±1.2	94.3±1.8	94.8±1.8	Springer *et al.* [7]
	93.7±1.0	**95.2±1.2**	**95.8±1.4**	Renna *et al.* [10]

The description of N is available in Section III. Symbols A_R, S, and P_+ stand for recording accuracy, sensitivity and positive predicted value, respectively, and are described in Section V B 1.
The results of Renna *et al.* [10] shown here use the sequential max temporal modelling, which is the temporal modelling used in this work.

computationally intensive solutions, such as machine and deep learning models. Moreover, owing to the substantial impact of CVDs in low- and middle-income countries, it is desirable to have a low-cost and internet-independent system suitable for use in areas where the main resources are not regularly available. Additionally, as clinical data are sensitive, their privacy must be ensured, making their processing undesirable in third-party datacenters such as Big Tech cloud services. For these reasons, an edge-computing solution is the best option, because this choice entails a single device acquiring and processing the data.

Four leading hardware platforms are widely used to implement algorithms: central processing units (CPUs), graphic processing units (GPUs), field-programmable gate arrays (FPGAs), and application-specific integrated circuits (ASICs). Heterogeneous systems are also emerging by combining previous platforms.

CPUs are the most general-purpose approach, but with very limited parallelization capability, whereas ASICs are specific solutions that can provide full parallelization. GPUs and FPGAs lie in between, both are general-purpose and highly parallelizable, but their nature differs in terms of flexibility and adaptability. GPUs are particularly well-suited for parallel processing tasks, making them ideal for graphics rendering and batch training of deep learning models, but often with high energy consumption. Their development methodology is straightforward thanks to the extended support of libraries such as CUDA and OpenCL [11], [12].

On the other hand, FPGAs offer a unique advantage with their reconfigurable hardware, allowing for custom hardware acceleration tailored to specific algorithms and unlocking great optimization capabilities. While GPUs excel at tasks with high data parallelism, FPGAs offer a more flexible and adaptable solution while supporting high parallelization capabilities, making them suitable for a wider range of applications, especially those requiring low-latency and power efficiency, such as a real-time CVD screening device requires.

The drawback of using FPGAs is the development methodology, as they require the hardware description to be implemented. Fortunately, there are high-level synthesis (HLS) tools that enable FPGA programming from an algorithmic description, thus shortening the implementation time close to its GPU counterpart while maintaining a high-level of control over the synthesized design. Moreover, the possibility of using fixed-point data types of arbitrary lengths in HLS allows further optimization of the algorithms by lowering the resolution of the data types below 16 bits. The implementation of custom hardware for optimized inference of machine and deep learning models in FPGAs has become popular in recent years [13], [14], [15], [16], [17], thanks to the advances in HLS tools, opening up the possibility of using low-cost FPGAs as target platforms to implement artificial intelligence models. Finally, the heterogeneous platforms, such as the Xilinx[1] Zynq[2] 7000 series [18], allow the distribution of the computational workload between the CPU and FPGA during prototyping periods. This unique characteristic enhances flexibility by allowing developers to fine-tune the allocation of processing tasks based on their nature and complexity. For example, the Xilinx Zynq 7000 series seamlessly integrates a powerful ARM Cortex-A9 processor with an FPGA fabric, providing a versatile environment for algorithm development.

Additionally, one of the advantages of using deep learning models with hierarchical architectures is their ability to introduce parameters to control their size. This is especially advantageous when the model must be implemented on low-end hardware because this opens another way to adapt the model to its target optimally [19].

Only a few studies have presented hardware implementations of heart-sound segmentation algorithms based on deep learning. Kwiatkowski et al. [20] implemented a small CNN in an ARM Cortex M7 processor with an inference time of 11 ms, using an 8-bit representation. Vakamullu et al. [21] used a Raspberry Pi 3B (quad-core ARM Cortex A53) to implement a 1-D CNN. They used different combinations of the decimation factor of the data and kernel size of the CNN to fit the model on the targeted device. No execution times have been reported, even though their design has been physically validated. These works prove the feasibility of the implementation of deep learning heart-sound segmentation algorithms on microprocessors like ARM Cortex-M and -A series with real-time performance. Even so, the heart-sound segmentation algorithm is just a first step in a real-time CVD screening device, that will require the concurrent operation of multiple algorithms, probably most of them deep learning models. For these reasons, we believe an FPGA is a more suitable device for this purpose since the customization, optimization, and parallelization capabilities these devices have will unlatch the concurrent operation of the following stages, as a murmur detector. Finally, the selection of a heterogeneous platform, such as the Xilinx Zynq 7000 series will allow the rapid swapping of the computational workload between the CPU and the FPGA during the development of the system.

[1]Registered trademark.
[2]Trademarked.

This work proposes an optimization of the implementation of a U-Net-based segmentation algorithm targeting the Xilinx Zynq 7020 FPGA, as a first stage toward the development of a real-time CVDs screening device. This work involves the following contributions:

1) Reproduction of the U-Net-based segmentation algorithm with *sequential max temporal modeling*, evaluated over the 2016 Physionet/CinC Challenge dataset [22], [23].
2) Evaluation of the model over the CirCor DigiScope PCG dataset [23], [24] proving its suitability for a more extensive dataset with environmental noise.
3) Identification of novel architecture parameters that enable further control of the model size, and computation of the effects these parameters have on the performance metrics, number of model operations, and memory consumption.
4) Exploration of two different implementation strategies: one with shared memory for feature maps and the other with streaming dataflows. For each strategy, the impact of the model reduction parameters on the model accuracy, FPGA resource consumption, and latency (i.e., execution time or inference time) of the model are analyzed.
5) Offline evaluation of model performance and FPGA resource consumption with different low-resolution fixed-point representations using the aforementioned public datasets.
6) Perform the preliminary step to envision a hand-held and low-power device that automatically detects heart sound abnormalities, enabling the detection of early signs of CVDs in the clinical practice in short periods of time.

To the best of our knowledge, this is the first work that assesses an in-depth study of the U-Net-based cardiac sound segmentation algorithm targeting an FPGA implementation. It includes an exhaustive analysis of the influence of the aforementioned model reduction parameters and the optimization of the model implementation to achieve the best performance in terms of 1) classification metrics; 2) latency; and 3) FPGA resource consumption, thus demonstrating that the state-of-the-art cardiac sounds segmentation algorithm can be executed in real-time on a low-end device.

The rest of the article is organized as follows. Section II introduces the underlying concepts of the operations in CNNs and the basis of the HLS tools to the reader. The U-Net-based segmentation model is presented in Section III. Section IV includes the methodology followed for the model optimization during the training and implementation steps. The datasets used for experiments, the target FPGA, the results of the training, the HLS C simulations with fixed-point representations, the synthesis, the C/RTL co-simulation and a comparison with other implementations are included in Section V. Finally, some conclusion is drawn in Section VI.

An open-source release of the code used in this work is available on GitHub.[3]

II. Background

A. Convolutional Neural Networks

CNNs are a type of neural network capable of extracting features from data using convolutional structures inspired by the biological vision perceptron. These architectures have been particularly successful in computer vision and solving tasks such as object detection [25] with state-of-the-art accuracy. CNNs have also shown promising results in the biomedical field at tasks such as image segmentation [9] and disease classification [26].

The output of each convolutional layer is called a feature map because it is composed of the features learned by the corresponding layer. The key layer parameter is the convolutional kernel, which represents the vision receptors. Because they have a limited size, they are affected by border effects, thereby reducing the output feature map. To fix this, padding can be employed to enlarge the input with zeros, and thus cancel the border effects. In addition, stride is the parameter that controls the density of the convolution operations: the larger it is, the lower the density.

In the U-Net-based segmentation model, as temporal signals are processed (i.e., PCGs), the convolutional layers are all 1-D. In addition, they all have a stride of one and 1-D kernels with a size of three. The input matrix is denoted as $\mathbf{A} \in \mathbb{R}^{N_m \times n_{\text{in}}}$ and the output as $\mathbf{B} \in \mathbb{R}^{N_m \times n_{\text{out}}}$, where N_m is the number of elements along the time axis, n_{in} is the number of input features, and n_{out} is the number of output features, the operation is defined as

$$\mathbf{B}_{i,k} = \sum_{l=\max(0,i-1)}^{\min(N_m-1,i+1)} \sum_{j=0}^{n_{\text{in}}-1} \mathbf{A}_{l,j} \mathbf{W}_{l-i+1,j,k} \tag{1}$$

where the generic notation for elements in a matrix is $\mathbf{C}_{i,j}$, that denotes the ith element in the time axis of the jth feature. In addition, the weights tensor is denoted by $\mathbf{W} \in \mathbb{R}^{N_m \times n_{\text{in}} \times n_{\text{out}}}$ and the element subscripts in the $\mathbf{W}_{i,j,k}$ correspond to the time dimension, input features, and output features, respectively. Besides, instead of zero-padding the inputs, the spatial dimensions are preserved along the feature maps in (1) by adjusting the kernel operation limits. Finally, after the operation of the convolutional layer, a nonlinear activation function is applied to the output matrix $\mathbf{B}$. In this study, three different activation functions are used: the rectified linear unit (ReLU), which operates elementwise and is defined as

$$\text{ReLU}(z) = \max(0, z). \tag{2}$$

ArgMax, which returns the index of the maximum when operating over a vector, and *SoftMax*, which is defined as

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_j e^{z_j}}. \tag{3}$$

[3]https://github.com/eneriz-daniel/PCG-Segmentation-Model-Optimization/

TABLE II
HLS Directives Summary

Directive	Description
pipeline	Allows concurrency in operations execution
unroll	Creates copies of the loop to allow parallel execution
dataflow	Enables task and operations execution overlapping

Additionally, CNN architectures also comprise layers that manipulate the feature maps: *pooling layers* that reduce their dimensionality, *up-sampling layers* that operate conversely, increasing them, and *concatenation layers* that allow stacking them.

B. High-Level Synthesis

HLS tools have become popular in hardware design, increasing abstraction from the register transfer level (RTL), whose complexity lengthens the development time in system-on-chip (SoC) designs. Basically, they enable hardware synthesis from a high-level language, which automatically generates the equivalent hardware description language (HDL); therefore, the design is easily implemented in a hardware platform, such as an FPGA or ASIC, without the need to develop an RTL design [27], [28].

Specifically, Vivado HLS is a tool suitable for synthesizing and implementing a design from an algorithmic description, converting C/C^{++} code into HDL, which can be used to program a Xilinx FPGA. This process is based on four steps:1) HLS C simulation, which runs the description code and validates its operation; 2) synthesis, which generates the equivalent HDL from the C/C^{++} description; 3) C/RTL co-simulation, which verifies that both designs work accordingly; and 4) HDL exportation. A key feature of Vivado HLS is that it allows the use of different directives to optimize the C/C^{++} code in an FPGA-friendly manner during the synthesis step. Different directives should be selected in different sections of the code, depending on the goal (area, throughput, or latency). A summary of the directives used in this work is included in Table II. In addition, the original code must sometimes be modified to guide the synthesis process and take advantage of FPGA characteristics [29].

III. Related Work

Renna et al. [10] presented the first PCG segmentation model based on CNN. Specifically, it is an adaptation of U-Net [9], a model developed for biomedical image segmentation. To work with PCGs, the model was modified to operate with 1-D signals. A detailed schematic of the architecture is presented in Fig. 2.

Prior to the model analysis, the data must be preprocessed. First, each heart sound is bandpass filtered between 25 and 400 Hz. The spike removal method described in [6] is then applied. The next step is the generation of four different envelograms, as in [7] and [10]:

1) Hilbert envelope: extracts the absolute value of the Hilbert transform.

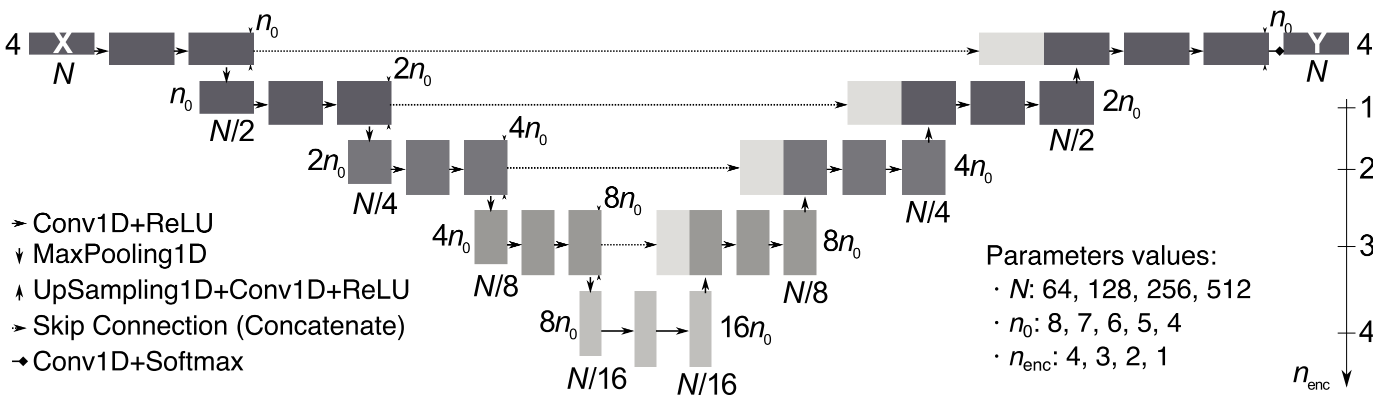

Fig. 2. Segmentation architecture scheme. The channels are represented in the ordinate axis, the time is on the abscissa axis. N is the input window length and n_0 is the base number of filters. This representation shows $n_{\text{enc}} = 4$ encoders and decoders. For visualization purposes, the concatenation of the skip connections is drawn in the time axis, while in fact is done in the channels' axis. The list of studied values for the parameters N, n_0, and n_{enc} are also included.

2) Homomorphic envelogram: computed by exponentiating the low-pass filtered natural logarithm of the Hilbert envelope.
3) Power spectral density (PSD) envelope: calculated from the signal spectrogram between 40 and 60 Hz with 50% overlapping windows of 0.05 s width.
4) *Wavelet envelope* computes the Shannon energy of a decomposition level after applying a Daubechies wavelet.[4]

Finally, the envelograms are downsampled to 50 Hz to reduce the computational impact and normalized to have a zero mean and unit variance. A visual example of the preprocessing step is presented in Fig. 3.

In this way, after preprocessing, a signal with four features is obtained: $\boldsymbol{x}(t) \in \mathbb{R}^4$ for $t = 0, \ldots, T-1$, where t indicates the time instant, and T is the total time of the PCG. Denoting $s(t)$ as the sequence containing the state labels for each time instant ($s(t) \in \{1, 2, 3, 4\}$, where state 1 corresponds to $S1$, state 2 corresponds to the *systole* interval, state 3 to $S2$ and state 4 to the *diastole* interval) and given $\boldsymbol{x}(t)$, the segmentation model provides an estimation of its corresponding state sequence $s(t)$. Patches of fixed length N are extracted from $\boldsymbol{x}(t)$ with a specific stride $\tau = N/8$ to be used as the input for the model, which are expressed as $\mathbf{X}(n) \in \mathbb{R}^{N\times 4}$ and obtained as follows:

$$\mathbf{X}(n) = \begin{bmatrix} \boldsymbol{x}(n\cdot\tau) \\ \vdots \\ \boldsymbol{x}(n\cdot\tau + N - 1) \end{bmatrix} \tag{4}$$

for $n = 0, \ldots, \lfloor (T-1-N/\tau) \rfloor$, where $\lfloor a \rfloor$ indicates the greatest integer lower than or equal to a.

The first stage of the network consists of four encoding blocks, where the signal is compacted in the time dimension while the number of channels is increased. This keeps only the most relevant information for PCG segmentation and reduces the impact of noise. Each encoding block is composed of two consecutive 1-D-convolutional layers with ReLU activation

[4]In [8] the Daubechies 10 wavelet at decomposition level three was used, but in our case, we used Daubechies 1 wavelet at decomposition level 4 as done in [30].

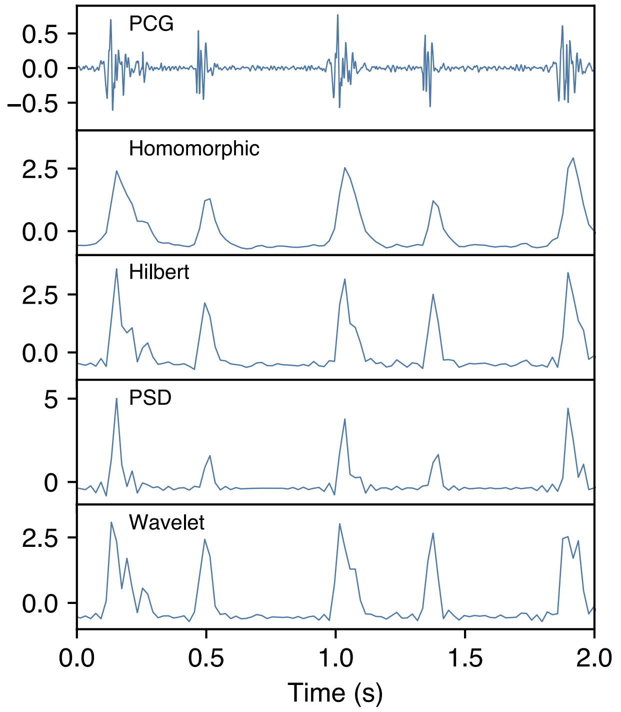

Fig. 3. PCG preprocessing example, where four different normalized envelopes-envelograms are extracted from the normalized PCG.

and a *max-pooling* layer that halves the time dimension. The number of filters of the convolutional layers in the first encoder is eight, which is doubled in each encoder to increase the number of channels. After the encoder part, two consecutive 1-D-convolutional layers with ReLU activation and 128 filters are placed in the architecture section with the highest temporal compression. It is then followed by the decoding stage, where information is expanded back in the time dimension, omitting irrelevant information from the input signals.

In more detail, each decoder has two inputs, the previous feature map, and a skip connection, allowing direct information transfer from the encoded layers to the decoded ones. First, the time dimension is doubled by an up-sampling layer followed by a 1-D-convolutional with ReLU activation, which halves the number of channels. Then, its output is concatenated along the channel axis with the skip connection originating at the analog encoder block, doubling the channels again. Subsequently, two consecutive 1-D-convolutional layers with ReLU activation are placed to decode the information and reduce the number of channels to half of the decoder input. The number of filters in each decoder layer is fixed to obtain an output with the same shape as their encoder counterparts.

As mentioned earlier, the kernel size of all convolutional layers in both the encoder and decoder blocks is fixed at 3. Additionally, a stride equal to 1 with padding "same" is used to preserve the shape of the feature maps between the layers.

Finally, there is an extra 1-D-convolutional layer with four filters and *SoftMax* activation, which provides the probability of being in each fundamental heart state per time instant of the input patch $n, \mathbf{Y}(n) \in \mathbb{R}^{N\times 4}$. Because the patch size N and stride τ for a given time instant t will influence $\mathbf{Y}(n)$, overlapping patches are used to minimize the influence of border data samples. Therefore, the information obtained from the patches is combined by averaging the state probabilities associated with different $\mathbf{Y}(n)$ values. This allows the computation of $\mathbf{y}(t) \in \mathbb{R}^4$ for $t = 0, \ldots, T-1$, which are the probabilities of each fundamental heart state at each time instant of the input recording.

The goal of the model is to estimate the sequence of heart states $s(t)$. In [10], different temporal modeling solutions were evaluated, forcing the output sequence to contain only admissible transitions between cardiac cycle states. In this work, *sequential max temporal modeling* is selected owing to its simple implementation and low computational complexity, providing performance comparable to the other strategies studied. First, a coarse estimation of $s(t)$ is obtained as follows:

$$\tilde{s}(t) = \text{argmax }\, \boldsymbol{y}(t). \tag{5}$$

Then, the output sequence $\hat{s}(t)$ is forced to contain only admissible transitions by setting $\hat{s}(0) = \tilde{s}(0)$ and using the rule

$$\hat{s}(t) = \begin{cases} \tilde{s}(t), & \text{if } \tilde{s}(t) = \tilde{s}(t-1) \bmod 4 + 1 \\ \tilde{s}(t-1), & \text{otherwise} \end{cases} \tag{6}$$

for $t > 0$.

IV. Methodology

There are two main ways to optimize the model implementation to achieve real-time performance. One is the reduction of the model architecture, which can be enabled with model parameterization in the case of hierarchical architectural models. The other is during the implementation itself, where some paradigms can be followed to optimize the model. Fig. 4 shows a summary of these two optimization routes, described in Sections IV-A and IV-B. The fixed-point representation analysis is also included in the diagram as part of the optimization, which will be discussed in Section V.

A. Model Reduction Strategy

As mentioned in Section I, one of the advantages of deep learning models is their reduction capacity. In this case, the original model is already parameterized by N, the input

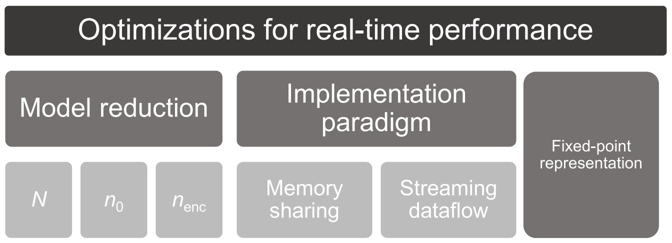

Fig. 4. Summary of the optimization strategies addressed in this study.

window length, which takes values of 64, 128, 256, and 512. This enables slight control of the model size in terms of the number of operations and feature map memory, which has $n_{fm} = 328 \cdot N$ elements. In contrast, the number of parameters remains the same, $n_w = 179\,904$, because all kernels are dependent only on the kernel size and the number of input and output filters present in each layer.

Two more parameters are identified to further control the model size: the number of filters used in the first encoder, n_0, and the number of encoders/decoders, n_{enc}, as illustrated in Fig. 2.

The first one controls the number of filters in all layers because it is duplicated at each encoding step and halved at each decoding step until the original number of filters is recovered. This parameter was initially set to 8, but it is reduced to 4 in steps of 1. Note that reducing the filter size below 4 is useless, since there are four input features, and the output size is 4. The second, the number of encoding/decoding stages, is a coarse control of the model. It was originally set to 4 and, in this work, varies from 4 to 1 in steps of 1. With these reductions, the number of weights n_w, and the total number of elements in the feature maps n_{fm}, are respectively, given by

$$n_{\text{w}} = 3 \cdot n_0 \left[8 + n_0 \left(1 + 11 \sum_{i=0}^{n_{\text{enc}}-1} 4^i \right) \right] \tag{7}$$

$$n_{\text{fm}} = N \cdot \left[8 + n_0 (2 + n_{\text{enc}}(19/2)) \right]. \tag{8}$$

Hence, 80 different models are considered ($4 \cdot N \times 5 \cdot n_0 \times 4 \cdot n_{\text{enc}}$), ranging from the minimal model with $n_w = 672$ elements and $n_{fm} = 3456$ elements to the largest model with $n_w = 179\,904$ elements and $n_{fm} = 167\,963$ elements.

B. Implementation Optimization Strategies

One of the main reasons for implementing a computationally intensive model, such as the U-Net-based model, in an FPGA is the capability this technology offers to parallelize tasks while enabling the possibility of working with arbitrary-length fixed-point data types, which can save resources in the final hardware implementation. Because this algorithm is aimed at helping physicians in real-time, in this case, the latency is considered the main key performance indicator (KPI), together with the logic resources usage, which are mainly block random access memory (BRAM), digital signal processing (DSPs) slices, flip-flops (FFs), and look-up tables (LUTs).

Algorithm 1 Memory-Based Implementation of the *Conv1D* Layers

```
Inputs: Matrix $\mathbf{A} \in \mathbb{R}^{N_m \times n_{in}}$ (either in a dedicated or shared memory space); Matrix $\mathbf{W} \in \mathbb{R}^{N_m \times n_{in} \times n_{out}}$ (in a dedicated memory space).
Outputs: Matrix $\mathbf{B} \in \mathbb{R}^{N_m \times n_{out}}$ (either in a dedicated or shared memory space).
Initialize scalars: $acc$, $l_{min}$, $l_{max}$
for all $k = 0$ to $n_{out}-1$ do
  for all $i = 0$ to $N_m-1$ do #(unroll, pipeline)
    $l_{min} = \mathbf{max}(0, i-1)$
    $l_{max} = \mathbf{min}(N_m-1, i+1)$
    $acc = 0$
    for all $l = l_{min}$ to $l_{max}$ do
      for all $j = 0$ to $n_{in}$ do
        $acc += \mathbf{A}_{l,j} \cdot \mathbf{W}_{l-i+1,j,k}$
      end for
    end for
    $\mathbf{B}_{i,k} = \mathbf{ReLU}(acc)$
  end for
end for
```

The *unroll* and *pipeline* directives are used only in the *Conv1D* layers of selected encoders and decoders of the optimized memory-sharing implementation.

To set a reference, a baseline implementation without any optimization strategy is developed. The implementation of the *Conv1D* layer under this paradigm is shown in Algorithm 1, where the input and output matrices **A** and **B**, respectively, have unique memory spaces. As shown in the algorithm, the convolution operation is based on nested loops. Thus, one of the more potential ways to accelerate this model is to perform loop unrolling and pipelining [31], which are the basic directives used in any loop optimization process. The first one, loop unrolling, is based on the physical implementation of more than one loop epoch, enabling a certain parallelization level. In HLS, loop unrolling was implemented using the *unroll* directive in the loops that were a bottleneck for the model latency. The second one, pipelining, enables concurrent execution using the same hardware. For this, the operations schedule is tailored to maximize hardware usage and minimize latency. The HLS directive employed to pipeline the desired section of the code was *pipeline*. Usually, the usage of these directives rapidly scales resource consumption, which makes them a poor-quality optimization control.

Fortunately, in addition to the basic optimization directives, other strategies can be followed in HLS to further improve the implementation optimization. These strategies are commonly related to the way the description code is written and/or the kind of resources it uses for its synthesis. In this study, two different strategies are tested to optimize the U-Net-based heart-sound segmentation algorithm: a memory-sharing strategy, where there is a common memory space where the feature maps are stored, and a streaming dataflow strategy, where the feature maps are treated as data streams that flow through the network. The former is similar to the optimized code that would be implemented on a CPU, whereas the latter treats the feature maps as a first-in-first-out (FIFO) queue, which requires a specific way to compute the model layer operations. Both implementation strategies are explained in detail in the following subsections.

1) Memory-Sharing Optimization Strategy: The baseline model implementation, wherein each feature map is stored in an independent array, extensively uses other FPGA logical resources. To reduce this usage, only two unique arrays are used to store all the different feature maps generated by the model. These arrays have the largest size in both dimensions (i.e., $N \times 16\ n_0$, as shown in Fig. 2), so the largest and the smallest feature maps can use the same arrays. Thus, optimization directives can be added to reduce latency by taking advantage of the saved resources.

The reason for using two different arrays is to avoid conflicts between readings and writings in the same array, which would lead to a malfunction of the algorithm because previous time instants of the input feature maps are used to compute a given time instant of the output feature map. In addition, because the model employs skipped connections, the feature maps that must be concatenated in the decoding layers must be stored in separate arrays.

The *Conv1D* implementation under this paradigm is also shown in Algorithm 1, although in this case, the input and output matrices, **A** and **B**, respectively, are saved in one of the two feature map memory spaces.

2) Streaming Dataflow Optimization Strategy: There are two main reasons for testing the streaming dataflow optimization strategy. First, with this paradigm, each feature map is treated as a FIFO, significantly reducing memory usage and access. This is expected to translate into a significant latency reduction and logic resource decrease, which would allow further optimizations by applying the previously mentioned basic optimization directives in more sections because more free logic is available in the device. Conversely, the code is less optimizable, because the data stream can only be accessed once per clock cycle. However, pipeline and loop unrolling directives can further optimize the remaining operations performed, such as multiple accumulations (MACC) and buffer accesses, can be further optimized.

Second, under this dataflow paradigm, the execution of different layers of the model can overlap, that is, before finishing the calculations of one layer, the following layer can start executing when enough input elements have been generated. This is a crucial advantage of this strategy, as it was not possible in the memory-based implementation, which limits the execution of a given layer after the completion of the previous layer.

As illustrated in Algorithm 2, significant differences exist between this implementation and the memory-based ones (i.e., the baseline and memory-sharing implementations). The HLS Stream Library allows the use of streams, which are the C constructs that enable the employment of FIFO with configurable depths in this way. Because the convolution kernel of the model is 3, the input feature map time instants are used up to three times. To enable this under the dataflow

Algorithm 2 Streaming Dataflow Implementation of the *Conv1D* Layers

```
**Inputs**: Stream $a$ (in a FIFO buffer), with the values of $\mathbf{A} \in \mathbb{R}^{N_m \times n_{in}}$ with priority of the feature dimension; Matrix $\mathbf{W} \in \mathbb{R}^{N_m \times n_{in} \times n_{out}}$ (in a dedicated memory space).
**Outputs**: Stream $b$ (in a FIFO buffer), with the values of $\mathbf{B} \in \mathbb{R}^{N_m \times n_{out}}$ with priority of the feature dimension.
**Initialize** scalars: $in_{val}, buff_{val}$
**Initialize** vector with zeros: $\mathbf{acc} \in \mathbb{R}^{n_{out}}$
**Initialize** matrix: $\mathbf{BUFF} \in \mathbb{R}^{2 \times n_{in}}$
#(*dataflow*)
**for all** $i = -1$ to $N_m$ **do**
  **for all** $j = 0$ to $n_{in}-1$ **do**
    **if** $i > -1$ **and** $i < N_m$ **then**
      $in_{val} \leftarrow a$
    **else**
      $in_{val} = 0$
    **end if**
    **for all** $l = 0$ to $2$ **do**
      **if** $l = 2$ **then**
        $buff_{val} = in_{val}$
      **else**
        $buff_{val} = \mathbf{BUFF}_{l,j}$
      **end if**
      **for all** $k = 0$ to $n_{out}-1$ **do** #(*pipeline*)
        $acc_k \mathrel{+}= buff_{val} \cdot \mathbf{W}_{l,j,k}$
        **if** $j = n_{in}-1$ **and** $l = 2$ **then**
          **if** $i \geq 1$ **then**
            $b \leftarrow \mathbf{ReLU}(acc_k)$
          **end if**
          $acc_k = 0$
        **end if**
      **end for**
      **if** $l > 0$ **then**
        $\mathbf{BUFF}_{l-1,j} = buff_{val}$
      **end if**
    **end for**
  **end for**
**end for**
```

The *pipeline* directive is used only in the *Conv1D* layers of selected encoders and decoders of the optimized streaming dataflow implementation.

paradigm, small memory buffers are used to store and reuse the samples.

V. Experiments

In this section, the procedure for performing model training and the implementation results is described. First, a subsection describing the datasets used in this study and the target FPGA is presented. Subsequently, the training process and performance metrics are described, and the model performance results are reported. Then the results of the HLS C simulation of the models with fixed-point data types, their synthesis, and C/RTL co-simulation results are discussed. Next, another subsection analyzes the effect of the fixed-point data type on the model performance, latency, and FPGA resource consumption. Finally, a comparison with other deep-learning-based heart sound segmentation model implementations is included.

A. Materials

1) Datasets: Two different datasets are used in this study. The first is the publicly available data[5] from the 2016 Physionet/CinC Challenge dataset [22], [23]. It is composed of 792 PCGs from 135 patients with and without pathologies recorded in clinical and nonclinical environments. To identify the ground-truth segmentation labels, the dataset also provides the estimated positions of the *R-peak* and *end-T-wave* points in an Electrocardiogram (ECG) recorded simultaneously with the PCG [7]. The *R-peak* and *end-T-wave* positions corresponded to the $S1$ and $S2$ states, respectively.

The second dataset is the public data[6] from the CirCor DigiScope PCG dataset [23], [24], released for the 2022 George B. Moody Physionet challenge. It is composed of 3163 PCGs from 942 patients with and without pathologies recorded during two mass screening campaigns conducted in the state of Paraíba, Brazil, between July and August 2014 and June and July 2015. These recordings have noise sources typical of an ambulatory environment, making this dataset a representative sample of real-world environments in which a PCG diagnostic aid device would be used. In this case, segmentation annotations were obtained from a semi-supervised scheme. First, the algorithms proposed in [7] and [8] and the U-Net-based model presented in [10] were used to obtain baseline labels, and then, a cardiac pathologist inspected their automatic annotations and re-annotated the misdetections. Unfortunately, labels were retained only in the segments indicated by the expert as a high-quality representative; therefore, there may or may not be a segmentation annotation at a given time in the recording.

2) Target FPGA: The target FPGA to map the U-Net-based model is the Xilinx XC7Z020, which is the programmable logic (PL) of the Xilinx Zynq 7020 low-end SoC. It includes 85 K PL cells, 53.2 K LUTs, 106.4 K FFs, 4.9 MB of BRAM, and 220 DSP slices of 18 × 25 MACC blocks. The SoC also includes a processing system (PS) consisting of a dual-core ARM Cortex-A9 with a maximum clock frequency of 667 MHz and 512 MB RAM [18].

B. Model Evaluation Methodology

To properly compare the performance of the models trained for this work with previously published results [10], the same data partition is performed for both datasets: ten-fold cross-validation with patient-exclusive splits. To reduce the HLS C simulation of the generated models (which is especially time-consuming), another partition with patient-exclusive splits for training (60%), validation (20%), and testing (20%) is also performed. The resulting model parameters of this second training are used to test the implementation of the HLS tool.

For both data partition schemes with both datasets, the categorical cross-entropy is used as the loss function for

[5]https://physionet.org/content/hss/1.0/
[6]https://physionet.org/content/circor-heart-sound/1.0.3/

the Adam optimizer, as done in [10], and the same training hyperparameters are used: learning rate of 10^{-4}, batch size of 1, and 15 epochs. The model weights at the minimum validation loss are saved.

All the training experiments are run with the Keras Python package [32] with a Tensorflow 2.8.0 backend over computing nodes with 24-core AMD EPYC 7443P CPUs, NVIDIA GeForce RTX 3090 GPUs, and 64 GB of RAM.

1) Performance Metrics: The performance metrics used for both data partition schemes include those used in [10]: the recording accuracy (A_R), defined as the fraction of instants in the entire recording output sequence $\hat{s}(t)$ that are correctly assigned to the corresponding label in the ground truth sequence $s(t)$, the positive predicted value (P_+), and the sensitivity (S), which are computed as

$$P_+ = \frac{T_p}{T_p + F_p} \tag{9}$$

$$S = \frac{T_p}{T_{\text{tot}}} \tag{10}$$

where a true positive (T_p) is counted when the center of an $S1$ (or $S2$) sound in the estimated sequence $\hat{s}(t)$ is closer than 60 ms from the corresponding sound in the ground-truth sequence $s(t)$. All others are considered false positives (F_p), and T_{tot} is defined as the total number of $S1$ and $S2$ sounds in the ground-truth sequence $s(t)$.

Another way to compute accuracy is also considered in this study. As can be seen from the previous description, the state sequence of the entire recording is used to compute the recording accuracy A_R. This requires a reconstruction step to obtain the recording output probabilities $\boldsymbol{y}(t)$ and temporal modeling to obtain the estimated sequence $\hat{s}(t)$. This is undesirable in the HLS C simulation process, which is time consuming. Therefore, we define the global accuracy (A_G) as the fraction of instants in each output probability patch $n, \mathbf{Y}(n) \in \mathbb{R}^{N\times 4}$ that have been correctly estimated compared with the ground-truth one-hot encoding state patch $n, \mathbf{S}(n) \in \mathbb{R}^{N\times 4}$ defined as

$$\mathbf{S}(n) = \begin{bmatrix} \boldsymbol{s}(n\cdot\tau) \\ \vdots \\ \boldsymbol{s}(n\cdot\tau + N - 1) \end{bmatrix} \tag{11}$$

where $\boldsymbol{s}(t)$ is the one-hot encoding version of $s(t)$.

2) Results: A comparison of the ten-fold cross-validation results of the models with $n_0 = 8$ and $n_{\text{enc}} = 4$ over the 2016 dataset with their equivalents from [10] is presented in Table III. As shown, a similar performance is achieved in both works, although our work reports slightly better results at lower N values. This may be related to the differences in random sampling of the data partition.

Fig. 5 shows the distribution of the models resulting from each reduction parameters combination in terms of total recording accuracy and the number of MACC operations. It is remarkable how the $N = 64$ models rapidly scale in accuracy while maintaining a constrained number of operations. Also, it can be noticed that slightly better performance is reached for higher N values. This is an effect

TABLE III

PERFORMANCE COMPARATION OF THE MODELS WITH $n_0 = 8$ AND $n_{\text{enc}} = 4$ TRAINED WITH TEN-FOLD CROSS-VALIDATION OVER THE 2016 DATASET WITH RENNA ET AL. [10] RESULTS. BEST RESULTS FOR BOTH MODELS ON EACH METRIC ARE HIGHLIGHTED

N	A_R (%)	S (%)	P_+ (%)	
64	91.5±1.6	91.2±2.3	94.1±2.1	Renna *et al.* [10]
	92.5±1.4	94.0±2.2	94.4±1.8	This work
128	92.6±1.6	92.7±2.0	95.6±2.0	Renna *et al.* [10]
	93.5±1.5	95.2±2.0	95.2±1.9	This work
256	93.0±1.7	94.3±1.9	95.4±2.0	Renna *et al.* [10]
	93.8±1.5	**95.7±1.9**	95.6±2.3	This work
512	**93.7±1.0**	**95.2±1.2**	**95.8±1.4**	Renna *et al.* [10]
	93.6±1.3	**95.7±1.6**	**95.8±1.3**	This work

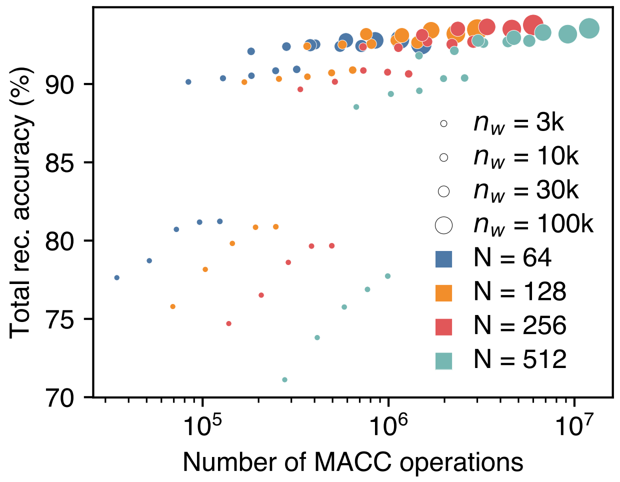

Fig. 5. Total recording accuracy of each model parameters combination for the ten-fold cross-validation trainings over the 2016 dataset in function of the number of MACC operations. The diameter of each point represents the number of weights in each model, n_w.

of the reduction of the dataset due to the necessity of samples with longer segmentation annotations, and thus not an intrinsic improvement due to the model architecture. For these reasons, only the results of $N = 64$ models are considered in the remainder of the article, although a complete report is available in the GitHub code repository.

The results of the ten-fold cross-validation of the models for the 2016 and 2022 datasets are presented in Table IV. In terms of the models' reduction strategy, it is noticeable the effect of the coarse parameter n_{enc}. Its reduction from $n_{\text{enc}} = 4$ to $n_{\text{enc}} = 3$ barely decreases the model performance, and when it is further reduced to $n_{\text{enc}} = 2$, the effect remains contained. At $n_{\text{enc}} = 1$, the model is truly limited, showing significant downgrades, especially for lower n_0 values. Meanwhile, the effect of n_0 on model performance is smoother than that of n_{enc}. Generally, negligible downgrades are observed when n_0 is reduced, although it becomes relevant at $n_{\text{enc}} = 1$, as previously mentioned.

The results of the training with the second data partition scheme, where training, validation, and testing splits are used, show the same parameter effects as the cross-validation ones, and thus, they are not fully reported. Only the global accuracy A_G is included, which is contained in the HLS C simulation

TABLE IV

Performance Metrics Averages and Standard Deviations Computed From the Ten-Fold Cross Validation Evaluation of the $N = 64$ Models for Both Datasets. Best Results for Both Datasets in Each Metric Are Highlighted

		A_R (%)				A_G (%)				S (%)				P_+ (%)			
	n_0 \ n_{enc}	4	3	2	1	4	3	2	1	4	3	2	1	4	3	2	1
2016	8	92.5±1.4	92.4±1.4	90.9±1.7	81.2±2.5	91.9±1.5	91.4±1.6	90.0±1.9	83.5±2.1	94.0±2.2	93.7±2.1	91.4±2.4	78.5±3.4	**94.4±1.8**	94.3±1.7	93.5±2.0	88.4±2.4
	7	**92.8±1.3**	92.4±1.4	90.8±1.9	81.2±2.6	**92.0±1.4**	91.5±1.5	90.0±1.8	83.2±2.0	94.2±2.1	93.7±2.2	91.2±2.7	78.7±3.7	94.1±1.8	94.2±1.8	93.5±2.2	88.5±2.8
	6	**92.8±1.3**	92.5±1.5	90.5±1.8	80.7±2.2	91.9±1.5	91.5±1.6	89.9±1.8	82.7±2.0	**94.5±1.9**	94.0±2.4	90.9±2.5	78.0±3.1	**94.4±1.8**	94.1±1.9	93.8±1.6	88.5±2.3
	5	**92.8±1.2**	92.4±1.3	90.4±1.8	78.7±2.4	91.7±1.5	91.3±1.6	89.4±1.9	81.8±2.0	**94.5±1.8**	93.8±2.2	90.9±2.6	75.5±3.8	**94.4±1.6**	94.2±2.0	93.3±2.0	87.7±2.6
	4	92.5±1.4	92.1±1.5	90.1±1.8	77.6±3.6	91.6±1.5	91.1±1.6	88.9±1.8	80.6±2.3	94.1±2.2	93.3±2.4	90.5±2.7	73.9±5.0	**94.4±1.8**	94.1±2.0	92.8±1.8	86.5±2.9
2022	8	90.2±0.8	90.1±0.9	89.1±0.9	84.6±0.9	**90.1±0.8**	90.0±0.9	88.9±0.9	84.7±0.9	96.0±1.0	95.9±1.0	94.2±1.1	87.8±1.0	96.0±0.8	**96.1±0.9**	95.4±1.0	93.2±0.9
	7	90.2±0.9	90.0±0.9	89.0±0.9	84.1±0.9	**90.1±0.9**	89.9±0.9	88.8±0.9	84.3±0.9	96.1±1.1	95.7±1.1	94.0±1.0	87.1±1.0	**96.1±0.9**	95.9±1.0	95.4±0.9	92.8±0.8
	6	**90.3±0.9**	90.0±0.9	88.8±0.9	84.0±0.9	**90.1±0.8**	89.8±0.8	88.7±0.8	84.0±0.9	**96.2±1.0**	95.8±1.0	93.9±1.0	87.1±1.0	**96.1±0.8**	96.0±0.9	95.3±0.9	92.8±0.9
	5	90.2±0.9	89.9±0.9	88.6±1.0	82.7±1.2	90.0±0.8	89.7±0.9	88.4±0.9	83.1±1.0	96.1±1.0	95.6±1.0	93.6±1.2	85.5±1.5	96.0±0.9	95.8±0.9	95.2±0.8	92.3±1.0
	4	90.0±0.9	89.8±0.9	88.1±1.0	81.9±1.0	89.7±0.9	89.5±0.8	87.9±0.9	82.2±0.9	95.9±1.0	95.5±1.0	93.0±1.2	84.5±1.2	95.9±1.0	95.8±1.0	95.0±0.9	91.8±1.3

TABLE V

Global Accuracy of the $N = 64$ Models From the Definitive Trainings Using Floating-Point and Q8.8 Fixed-Point Representations and Their Differences for the Both Datasets. Best Results for Both Datatypes Are Highlighted for Both Models

		GPU Inference - Floating-point accuracy (%)				FPGA inference - Q8.8 fixed-point accuracy (%)				Difference (%)			
	n_0 \ n_{enc}	4	3	2	1	4	3	2	1	4	3	2	1
2016	8	90.55	89.50	88.10	82.16	90.51	89.47	88.09	82.09	0.04	0.03	0.00	0.07
	7	**91.01**	89.25	87.92	81.66	**90.95**	89.26	87.95	81.63	0.06	-0.02	-0.03	0.03
	6	90.07	89.32	88.30	81.46	90.08	89.37	88.27	81.46	-0.02	-0.05	0.03	-0.01
	5	89.78	89.17	87.62	79.71	89.74	89.15	87.65	79.77	0.04	0.02	-0.03	-0.06
	4	90.28	89.84	86.45	78.62	90.30	89.78	86.47	78.71	-0.02	0.05	-0.03	-0.09
2022	8	**91.16**	90.73	89.64	85.76	**91.14**	90.72	89.66	85.73	0.02	0.01	-0.03	0.03
	7	91.09	90.62	89.88	84.92	91.11	90.62	89.89	84.85	-0.02	0.01	-0.01	0.07
	6	91.10	90.59	89.39	84.85	91.10	90.58	89.42	84.82	0.01	0.01	-0.03	0.02
	5	90.58	90.35	89.26	84.33	90.39	90.41	89.26	84.29	0.19	-0.05	0.00	0.04
	4	90.64	90.32	88.92	83.09	90.64	90.26	88.90	83.07	0.00	0.06	0.02	0.02

results available in Table V and labeled as floating-point accuracy. Note that these results are slightly different from the cross-validation results owing to the different ratios between the training and testing splits. In this case, it is 60/20, while for the cross-validation is 90/10.

C. HLS C Simulation

Once the model is validated in the training stage, it is manually ported to C^{++} to enable its implementation through Vivado HLS for the inference stage. This tool allows the use of arbitrary-length fixed-point data types through the Arbitrary Precision Data Type Library. A model represented by lower-resolution data types is expected to show a downgrade in performance compared to the model described in the higher-level training framework that uses 32-bit floating-point representation. This is owing to the quantization effect that appears when arithmetic operations are performed with lower-resolution fixed-point data types. To characterize this downgrade, the HLS C simulation feature can be used to virtualize the model implementation using the selected data type. In this stage, a Q8.8 data type is used, that is, eight integer bits and eight fractional bits, for a total of 16 bits.

In addition, the activation function of the last convolutional layer, *SoftMax*, is substituted. This function is beneficial during the training stage because it is a smoother version of *ArgMax*, enabling faster training processes. However, because it requires exponential operations, it is computationally expensive; therefore, for implementation purposes, it is better to use *ArgMax* which can be easily implemented with comparators and small memory elements.

To measure the performance of the models on both datasets, HLS C simulations are conducted for each model parameter combination on each dataset. Note that the model performance is independent of the model implementation because they are all equivalent to the Keras model. Thus, only the HLS C simulation results for the baseline implementation are reported herein. The results are presented in Table V. It is remarkable that the difference between the floating-point and Q8.8 fixed-point performance is independent of the model parameters N, n_0, and n_{enc}. The average downgrades in the 2016 dataset are $0.04 \pm 0.13\%$ and $0.01 \pm 0.04\%$ for the 2022 dataset.

D. Synthesis

This subsection presents the synthesis results obtained after applying HLS directives to reduce the model latency and memory consumption, as well as some code modifications to fully exploit the parallelization capabilities of the FPGA. To obtain realistic resource consumption results, the source file includes basic interface directives that set the input and output interfaces as AXI4-Lite slaves [33], except for the input and output streams of the dataflow version, which are set as AXI4 Stream [34].

To properly assess the effect of optimization strategies on different combinations of model parameters, the synthesis

TABLE VI

SYNTHESIS RESULTS OF THE DIFFERENT IMPLEMENTATIONS WITHOUT OPTIMIZATIONS OF THE $N = 64$ MODELS USING Q8.8 FIXED-POINT DATA TYPES. RESOURCES CONSUMPTIONS OVER THE AVAILABLE IN THE XC7Z2020 ARE MARKED IN RED

		BRAM (%)				DSP (%)				FF (%)				LUT (%)				Best Latency (ms)				Worst Latency (ms)			
	n_0 \ n_{enc}	4	3	2	1	4	3	2	1	4	3	2	1	4	3	2	1	4	3	2	1	4	3	2	1
Preliminary	8	103	37	18	11	10	8	6	4	8	6	4	3	33	26	19	12	50.78	24.37	11.07	4.33	75.98	36.41	16.50	6.43
	7	103	37	18	11	10	8	6	4	9	7	5	3	38	30	22	13	46.17	21.91	9.70	3.51	69.08	32.73	14.46	5.22
	6	63	28	18	11	10	8	6	4	9	7	5	3	38	30	22	13	34.05	16.21	7.22	2.65	50.93	24.20	10.75	3.93
	5	63	28	18	11	10	8	6	4	9	7	5	3	38	30	22	13	23.65	11.24	4.98	1.85	35.35	16.77	7.40	2.74
	4	43	25	18	11	10	8	6	4	8	6	4	3	33	26	19	12	13.01	6.36	2.99	1.26	19.42	9.46	4.43	1.85
Memory Sharing	8	97	31	14	8	10	8	6	4	7	6	4	3	32	26	19	12	50.78	24.37	11.07	4.33	75.98	36.41	16.50	6.43
	7	97	31	14	8	10	8	6	4	9	7	5	3	40	31	22	14	46.17	21.91	9.70	3.51	69.08	32.73	14.46	5.22
	6	57	23	13	8	10	8	6	4	9	7	5	3	39	31	22	14	34.05	16.21	7.22	2.65	50.93	24.20	10.75	3.93
	5	57	23	13	8	10	8	6	4	9	7	5	3	39	30	22	13	23.65	11.24	4.98	1.85	35.35	16.77	7.40	2.74
	4	35	18	12	8	10	8	6	4	7	6	4	3	32	25	19	12	13.01	6.36	2.99	1.26	19.42	9.46	4.43	1.85
Streamed	8	98	31	12	6	10	8	6	4	8	6	5	3	39	30	22	13	17.76	7.43	3.37	1.61	20.71	8.65	3.92	1.87
	7	94	28	11	6	10	8	6	4	8	6	5	3	40	31	22	14	13.60	5.69	2.58	1.24	15.86	6.63	3.01	1.44
	6	54	19	10	6	10	8	6	4	8	6	5	3	40	31	22	14	10.00	4.19	1.90	0.92	11.66	4.88	2.21	1.06
	5	54	19	10	6	10	8	6	4	8	6	5	3	40	31	22	13	6.95	2.91	1.33	0.65	8.10	3.39	1.54	0.75
	4	35	16	10	6	10	8	6	4	8	6	4	3	38	30	22	13	4.46	1.87	0.85	0.42	5.19	2.18	0.99	0.49

results of the baseline, memory-sharing, and streaming dataflow implementations without any optimizations are presented in Table VI. Remarkably, the limiting resource in all implementations is the BRAM, which is almost or above 100% for the models with $n_0 \in \{8, 7\}$ and $n_{\text{enc}} = 4$. Also, it can be noticed that this resource has a stepped scaling. This is probably due to the instantiation of memory blocks, which must have a power-of-two depth. In terms of DSP, it is shown that this resource is only dependent on the n_{enc} parameter, and the same consumption appears across different implementations. This is because of the lack of optimization directives, which means that only a single slice is used for each *Conv1D* layer. Finally, the dependence on n_{enc} is also the main effect in FF and LUT consumption, although it is noticeable that they consume less for $n_0 = \{4, 8\}$. This may also be related to memory organization in power-of-two blocks, where memory accesses are inherently optimized. Additionally, in memory-based implementations, LUTs also have a slim dependence on N.

However, latency is affected by all model parameters. For memory-based implementations, it scales linearly with N, and in the streamed implementation is slightly lower. In the case of the n_0 and n_{enc} parameters, the latency decreased rapidly. Overall, these dependencies enable an extensive range of latency values, as indicated in the complete results. In the case of memory-based implementations, the highest measured latency is 406.39 ms, while the lowest is 1.26 ms. For the streamed implementation, this range is more constrained, from 100.62 ms down to 0.42 ms. This is due to the remarkable latency reduction this implementation strategy presents compared with the memory-based implementations, which achieves an average latency decrease factor of 3.71 ± 0.61, with a minimum of 2.69 and a maximum of 4.81.

To better characterize the maximum potential of each implementation strategy, the $N = 64$, $n_0 = 8$, and $n_{\text{enc}} = 4$ models are implemented with the maximum optimization available, using both strategies. The procedure followed in each paradigm is presented in the following subsections,

TABLE VII

SYNTHESIS, C/RTL CO-SIMULATION AND POWER CONSUMPTION RESULTS OF THE $N = 64$, $n_0 = 8$, AND $n_{\text{enc}} = 4$ MODEL USING Q8.8 FIXED-POINT DATA TYPES FOR DIFFERENT IMPLEMENTATIONS. THE LOWEST CONSUMPTIONS AND LATENCIES ARE HIGHLIGHTED

		Implementation		
		Non-optimized baseline	*Optimized memory-sharing*	*Optimized streaming*
BRAM (%)		103	**97**	99
DSP (%)		**10**	56	**10**
FF (%)		**8**	17	10
LUT (%)		**33**	93	44
Synthesis Latency (ms)	Best	50.78	65.91	**5.97**
	Worst	75.98	70.09	**5.97**
Cosimulation Latency (ms)	Best	82.12	82.05	**29.27**
	Worst	82.16	82.10	**29.27**
Power consumption (mW)	Dynamic	673	964	**558**
	Static	169	180	**163**
	Total	842	1144	**722**
Energy per inference (mJ)		69	94	**21**

and the FPGA resource consumption, latency, and power consumption results are presented in Table VII. The power consumption results are estimated by the Vivado tool.

1) Memory-Sharing Implementation Optimization: Using HLS to reduce model latency in the memory-sharing paradigm, optimizations based on the aforementioned directives are performed from the central part of the network (the one that requires more clock cycles) to the borders until the model implementation reaches the maximum logic resources available in the FPGA. As labeled in Algorithm 1, loop unrolling (*unroll*) and pipelining (*pipeline*) directives are included in the second outer loop of the *Conv1D* layers (i loop) of the last encoder, central part, and two first decoders. This shows the best results after loop-by-loop and layer-by-layer exhaustive analyses. Other optimization paths have also been explored: loop unrolling and pipelining in the *max-pooling* and *up-sampling* layers, *ArgMax* and ReLU activation functions, and feature map array partitions. None of these modifications

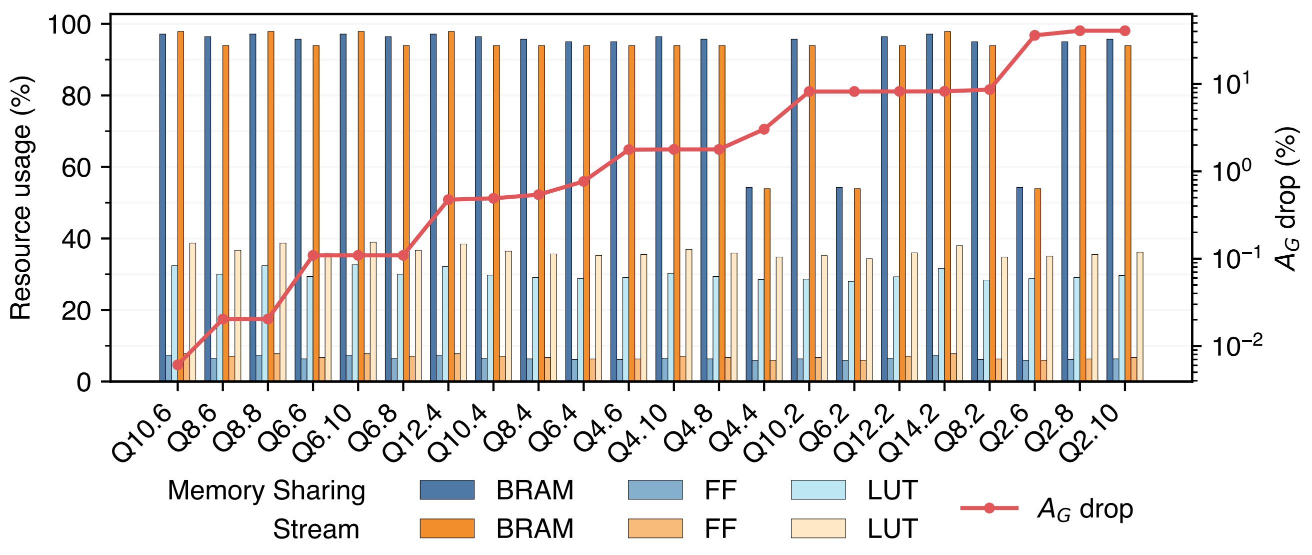

Fig. 6. BRAM, FF, and LUT consumption in each implementation paradigm and global accuracy (A_G) drop (respect to the floating-point performance) over the 2022 dataset for each tested fixed-point data type.

led to significant improvements in latency. These optimizations reduced the model latency by approximately 8% during the synthesis.

Including these directives has led to a significant increase in FPGA resource consumption, resulting in a 36% increase in total power consumption.

2) Streaming Dataflow Implementation Optimization: As expected, under the streaming dataflow paradigm using the HLS Stream Library under the *dataflow* directive, the latency decreases significantly while resource consumption remains similar to the baseline implementation values. As in the memory-sharing implementation, the limiting layers are the central part and its nearest neighbors. In this case, this effect is so relevant that the latency of the second central *Conv1D* layer is 99.99% of the total latency, followed by the two next *Conv1D* layers, from the first decoder, both with a latency of 83.61% of the total.

Therefore, the optimization of *Conv1D* layer is based on the usage of the *pipeline* directive, which inclusion in the innermost loop (k loop in Algorithm 2) significantly decreases the latency of the layer by allowing the overlapped operation of different sections of the code. With this directive, the latency of the second *Conv1D* layer of the central part decreases by a factor of three, ceasing to be the layer with the largest latency. Given this, the model can be further optimized by using the *pipeline* directive in the innermost loops of the *Conv1D* layers with more latency than the already pipelined *Conv1D* of the central part. These layers are the other layers in the central part, the layers of the last encoder, and those in the first two decoders. With these directives, the model latency is newly limited by the second *Conv1D* layer of the central part, making the model three times faster with a reduction of 3.9% of BRAM, which is the critical resource of this design.

Thanks to this reduction in BRAM consumption and the constrained increase in other resources, the total power consumption of this implementation is 14% lower than the baseline.

E. C/RTL Co-Simulation

To verify that the RTL design generated by the HLS tool works according to the C description, C/RTL co-simulation analysis are assessed. In addition, this provides more realistic latency values than those estimated during the synthesis stage. Because this process is time-consuming, only the baseline and optimized memory-sharing and streaming dataflow implementations of the $N = 64$, $n_0 = 8$, and $n_{\text{enc}} = 4$ models are launched. The results are listed in Table VII. As shown, there is an increase in the co-simulation latency compared with the synthesis results. This is mainly because, during synthesis, model interfaces are not considered for latency computation. Furthermore, Vivado HLS does not correctly compute the latency of dataflow systems during synthesis, which explains the larger increase in the co-simulation latency of the stream implementation. Nevertheless, an overall decrease in latency of 64% is achieved when the baseline and the optimized streaming dataflow implementations are compared.

Considering that the batch size (i.e., the number of samples the model evaluates per inference) was set to 1, the co-simulation latency directly results in the model latency. In the optimized streaming implementation, the model takes 2 926 713 cycles to process a sample at a clock frequency of 100 MHz, which means an inference time of 29 ms. The model input is a window of 64 samples sampled at 50 Hz, i.e., a signal of 1.28 s duration. Hence, a 29 ms processing time can be considered as real time with a significant margin for this task.

F. Low-Resolution Fixed-Point Datatype Effects

Finally, different low-resolution fixed-point data types between 16 and 8 bits have been considered for both HLS C simulation and synthesis. To simplify this analysis, the model parameters are set as $N = 64$, $n_0 = 8$, and $n_{\text{enc}} = 4$. The results are shown in Fig. 6, where the data types are sorted according to the global accuracy drop. Generally, in terms

TABLE VIII
COMPARISON BETWEEN THIS WORK AND OTHER HARDWARE IMPLEMENTATION OF DEEP LEARNING-BASED MODEL FOR HEART SOUND SEGMENTATION

	Kwiatkowski *et al.* [20]	Vakamullu *et al.* [21]	This work
Hardware (freq.)	ARM Cortex-M7 CPU (480 MHz)	Rasp. Pi 3B ARM Cortex-A53 CPU (1.5 GHz)	Xilinx Zynq 7020 FPGA (100 MHz)
#Classes	3	2	4
Datatype	8-bit integer	32-bit float	Q8.8 (16-bit)
Dataset	Physionet/CinC Challenge [22], [23]	HS Challenge 2011 [35], Littman Library [36], Michigan HS [37] and 8 volunteers	Physionet/CinC Challenge [22], [23] and CirCor DigiScope Phonocardiogram [23], [24]
#PCG-#Subjects	792-135	-	(792, 3163)-(135, 942)
Model	Tiny CNN	2-layers CNN	U-Net-based (CNN)
#Conv. Layers	3	2	23
Acc. (%)	89.84	87.78	90.51
Inference time (ms)	12	-	29
Power consumption (mW)	-	-	722

of resource consumption, slight reductions appear when the number of bits of the data type is reduced. Only at 8-bit representations, the BRAM drops significantly. This is due to the Vivado HLS packing method for the elements of the input AXI Lite interfaces, which forces them to use the nearest greater power-of-two bits. Therefore, the 14-, 12-, and 10-bit data types use the same BRAM as the 16-bit data type, which is the most significant usage. In addition, the global accuracy drop (compared with the floating-point performance) is affected by the combination of two effects: quantization, which is more significant when the number of fractional bits is low, and overflow, which appears when the number of integer bits is low. If an accuracy drop of less than 0.2% is considered acceptable, at least six bits are required for the decimal part to reduce the accuracy drop due to quantization. In the case of overflow, the integer part must have at least six bits; otherwise, the accuracy drop starts to increase owing to this effect.

G. Comparison With Other Implementations

Table VIII shows a comparison between the optimized streaming implementation of the U-Net-based heart sound segmentation algorithm and other existing implementations of heart sound segmentation algorithms using deep learning. It is worth noticing that, to the best of our knowledge, all other existing implementations found in the literature are based on CNNs. Additionally, as far as we know, this work is the only one that has been implemented on an FPGA, while the other two reported works have been deployed on a CPU. This platform diversity limits the fairness of the comparison. Nonetheless, it shows the state-of-the-art in this field, setting the basis for further research to improve accuracy, reduce inference times, or lower power consumption. First, the clock frequency of this implementation is 4.8 times lower than the one employed in [20] and 15 times lower than in [21]. This, together with the fact that FPGAs are less power demanding than CPUs would expectedly imply a significant decrease in the power consumption of this implementation compared to the other two. However, because only this work has reported a power consumption estimation, a comparative study was not possible. All works have used a CNN to perform the segmentation, differing in their architecture. Kwiatkowski et al. [20] used three convolutional layers with intercalated *max-pooling* and *batch-normalization* layers, and Vakamullu et al. [21] implemented just two convolutional layers followed by *max-pooling* layers. Our model has 23 convolutional layers and uses the U-Net architecture, with encoding and decoding stages. Given this, it is clear that our model is significantly more complex than the other two. For this reason, the classification results reported in this work achieved a more accurate segmentation considering the four heart sound components of a PCG, whereas [20] distinguished between *S1*, *S2*, and the rest of the signal, and [21] limited the model to only systole and diastole detection. Note that the three models have been trained with different datasets, so performance may depend on this factor. Finally, [20] reported a lower model inference time than this work. However, this could be related to the fact that they used a significantly smaller model with only three convolutional layers, halving the number of bits used in their implementation (8-bit representation against 16-bit), and they used 4.8 times the frequency employed in this work. Thus, equivalent or even lower inference times could be achieved by porting this design to an FPGA with a higher clock frequency. The work presented in [21] did not report any inference times for their implementation.

VI. Discussion and Conclusion

To the best of our knowledge, this work presents for the first time an exhaustive optimization study of the U-Net-based heart sound segmentation algorithm, which is the current state-of-the-art in this field, being tested in both the 2016 Physionet/CinC Challenge dataset and the CirCor DigiScope PCG dataset. To enable its implementation in an FPGA, an HLS tool was used to achieve significant improvements in terms of latency and logical resource usage, which allow its implementation on a low-end FPGA with real-time performance. As far as we know, there are not previously reported works that contain an implementation on this platform for heart-sound segmentation. The main result of this work is the reduction in inference time achieved by the optimized streaming implementation, compared to the baseline version. The co-simulation results showed that it was reduced from 82.12 to 29.27 ms, which is a 64% reduction of the original inference time. Additionally, the use of BRAM, the limiting FPGA resource, was also reduced by 3.9% in the optimized streaming implementation, which reported 99% BRAM usage, compared to 103% in the baseline. These two results have led to a significant 70% reduction in energy per inference, which was 69 mJ in the baseline and 21 mJ in the optimized streaming implementation. To achieve this, different optimizations have been evaluated.

First, the fact that hierarchical deep learning models can be easily reduced was considered. This had a direct impact on the number of MACC operations and, thus, on FPGA resource consumption. Two additional reduction parameters were identified in this study, proving that FPGA resource consumption can decrease significantly while maintaining segmentation model performance.

Second, two different implementation optimization strategies were tested: a *memory-sharing paradigm* and a *streaming dataflow paradigm*. Both strategies showed improvements in FPGA resource consumption and execution latency compared with the baseline implementation. Between them, this study demonstrates that the streaming dataflow implementation strategy obtains significantly better results than memory-based implementations because it treats the feature maps as a flow using FIFO queues, enabling the overlapping execution of consecutive layers. This drastically reduces the latency compared with memory-based approaches but requires a redesign of the description code.

In addition, both implementation paradigms were optimized with high-level directives, which were included in the bottlenecks of the designs. This reduction in latency came at the cost of increased FPGA resource consumption, with the BRAM close to 100% for both optimized implementations. If a larger model with better accuracy is released in the future, its implementation would require either a reduction of the FPGA consumption at the expense of latency (fewer optimization directives) or the use of larger FPGAs, which would increase the cost of the system and its power consumption.

Through the development of the model optimization, some limitations of the Vivado HLS tool were identified. For example, HLS might not consider the AXI interfaces declared at a high level to compute the latency and the latency derived from the streaming dataflow strategy at synthesis is not reproduced in the C/RTL co-simulation, but it is still significantly better than the memory-shared alternative. Hence, even though this tool boosts the hardware design and has been useful in significantly accelerating this model, manual fine-tuning of the generated HDL might be necessary to optimize this design completely.

As mentioned in Section I, few studies have implemented deep learning models to segment PCGs, and they have used small-sized architectures. Thus, to the best of our knowledge, this work is the first to exhaustively study different optimization strategies for implementing a large 1-D U-Net-based model with an estimated inference time of 29 ms using a 16-bit fixed-point representation. Considering that the length of the input window for these models was $N = 64$ and the sampling frequency was 50 Hz, a real-time response required less than $N/50 \approx 1.28$ s. Hence, it is feasible to implement this model in a computer-aided decision system to automatically identify the heart states in a PCG and potentially help physicians identify abnormalities in the patient's heart recordings with more complex analysis algorithms. The comparison with the state-of-the-art hardware implementations of similar algorithms evidenced the impact that the hardware optimization of this model had in the final results, outperforming them in accuracy and achieving real-time performance with significantly lower clock frequency. This is related to lower power consumption, thus being a more suitable solution for a low-cost and low-power computer-aid system.

Finally, note that this is also reproducible for any U-Net-based architecture, including the different model reduction parameters and the two tested implementation optimization strategies, which have been proven to accelerate the model in a low-end FPGA.

In future works, to obtain a functional heart-sound segmentation device attached to the stethoscope for real-time processing of the PCG, the preprocessing stage should also be optimized and implemented on the same hardware platform. To achieve this, an analog-to-digital converter should be introduced in the design without compromising the temporal restrictions. Since the estimated inference time of the segmentation part is 29 ms, there is a feasible temporal margin of more than 1.2 s to read and preprocess the data obtained by the sensor. Then, it is intended to perform an online evaluation of the physical platform, thus validating the prototype of a hand-held device capable of automatically detecting cardiac abnormalities from a PCG at an early stage.

Acknowledgment

The authors thank I3A (Aragon Institute of Engineering Research) for the use of High-Performance Computing (HPC) cluster HERMES.

References

[1] World Health Org. (WHO). *Cardiovascular Diseases (CVDs)*. Accessed: Nov. 13, 2023. [Online]. Available: https://www.who.int/news-room/fact-sheets/detail/cardiovascular-diseases-(cvds)

[2] S. Mangione, "Cardiac auscultatory skills of internal medicine and family practice trainees: A comparison of diagnostic proficiency," *JAMA*, vol. 278, no. 9, p. 717, Sep. 1997, doi: 10.1001/jama.1997.03550090041030.

[3] S. Li, F. Li, S. Tang, and F. Luo, "Heart sounds classification based on feature fusion using lightweight neural networks," *IEEE Trans. Instrum. Meas.*, vol. 70, pp. 1–9, 2021.

[4] A. Bhardwaj, S. Singh, and D. Joshi, "Explainable deep convolutional neural network for valvular heart diseases classification using PCG signals," *IEEE Trans. Instrum. Meas.*, vol. 72, pp. 1–15, 2023.

[5] H. Vermarien, "Phonocardiography," in *Encyclopedia of Medical Devices and Instrumentation*, 1st ed. Hoboken, NJ, USA: Wiley, 2006.

[6] S. E. Schmidt, C. Holst-Hansen, C. Graff, E. Toft, and J. J. Struijk, "Segmentation of heart sound recordings by a duration-dependent hidden Markov model," *Physiological Meas.*, vol. 31, no. 4, pp. 513–529, Apr. 2010.

[7] D. B. Springer, L. Tarassenko, and G. D. Clifford, "Logistic regression-HSMM-based heart sound segmentation," *IEEE Trans. Biomed. Eng.*, vol. 63, no. 4, pp. 822–832, Apr. 2016, doi: 10.1109/TBME.2015.2475278.

[8] J. Oliveira, F. Renna, T. Mantadelis, and M. Coimbra, "Adaptive sojourn time HSMM for heart sound segmentation," *IEEE J. Biomed. Health Informat.*, vol. 23, no. 2, pp. 642–649, Mar. 2019.

[9] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional networks for biomedical image segmentation," in *Lecture Notes in Computer Science*. Cham, Switzerland: Springer, 2015, pp. 234–241.

[10] F. Renna, J. Oliveira, and M. T. Coimbra, "Deep convolutional neural networks for heart sound segmentation," *IEEE J. Biomed. Health Informat.*, vol. 23, no. 6, pp. 2435–2445, Nov. 2019, doi: 10.1109/JBHI.2019.2894222.

[11] J. Nickolls, I. Buck, M. Garland, and K. Skadron, “Scalable parallel programming with CUDA: Is CUDA the parallel programming model that application developers have been waiting for?” *Queue*, vol. 6, no. 2, pp. 40–53, Mar. 2008, doi: 10.1145/1365490.1365500.

[12] J. E. Stone, D. Gohara, and G. Shi, “OpenCL: A parallel programming standard for heterogeneous computing systems,” *Comput. Sci. Eng.*, vol. 12, no. 3, pp. 66–73, May 2010.

[13] E. Wang et al., “Deep neural network approximation for custom hardware: Where We’ve been, where We’re going,” *ACM Comput. Surveys*, vol. 52, no. 2, pp. 1–39, Mar. 2020, doi: 10.1145/3309551.

[14] C. N. Coelho et al., “Automatic heterogeneous quantization of deep neural networks for low-latency inference on the edge for particle detectors,” *Nature Mach. Intell.*, vol. 3, no. 8, pp. 675–686, Jun. 2021, doi: 10.1038/s42256-021-00356-5.

[15] Y. Umuroglu et al., “FINN: A framework for fast, scalable binarized neural network inference,” in *Proc. ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays*, Feb. 2017, pp. 65–74.

[16] S. I. Venieris and C. Bouganis, “fpgaConvNet: A framework for mapping convolutional neural networks on FPGAs,” in *Proc. 24th Annu. Int. Symp. Field-Program. Custom Comput. Mach. (FCCM)*, 2016, pp. 40–47.

[17] A. Huang, Z. Cao, C. Wang, J. Wen, F. Lu, and L. Xu, “An FPGA-based on-chip neural network for TDLAS tomography in dynamic flames,” *IEEE Trans. Instrum. Meas.*, vol. 70, pp. 1–11, 2021.

[18] Xilinx Inc. *Zynq-7000 SoC Data Sheet: Overview*. Accessed: Sep. 21, 2023. [Online]. Available: https://www.xilinx.com/content/dam/xilinx/support/documents/data_sheets/ds190-Zynq-7000-Overview.pdf

[19] A. C. Hernandez-Ruiz, D. Enériz, N. Medrano, and B. Calvo, “Motor-imagery EEGNet-based processing on a low-spec SoC hardware,” in *Proc. IEEE Sensors*, Sydney, NSW, Australia, 2021, pp. 1–4, doi: 10.1109/SENSORS47087.2021.9639747.

[20] K. K. Kwiatkowski, D. P. Pau, T. Leung, and O. Di Marco, “Phonocardiogram segmentation with tiny computing,” in *Proc. IEEE Int. Conf. Consum. Electron. (ICCE)*. NV, USA: IEEE, Jan. 2023, pp. 1–4, doi: 10.1109/ICCE56470.2023.10043562.

[21] V. Vakamullu, S. Trivedy, M. Mishra, and A. Mukherjee, “Convolutional neural network based heart sounds recognition on edge computing platform,” in *Proc. IEEE Int. Instrum. Meas. Technol. Conf. (I2MTC)*. ON, Canada: IEEE, May 2022, pp. 1–6, doi: 10.1109/I2MTC48687.2022.9806693.

[22] C. Liu et al., “An open access database for the evaluation of heart sound algorithms,” *Physiol. Meas.*, vol. 37, no. 12, pp. 2181–2213, Dec. 2016.

[23] A. L. Goldberger et al., “PhysioBank, PhysioToolkit, and PhysioNet: Components of a new research resource for complex physiologic signals,” *Circulation*, vol. 101, no. 23, pp. 1–11, Jun. 2000, doi: 10.1161/01.cir.101.23.e215.

[24] J. Oliveira et al., “The CirCor DigiScope dataset: From murmur detection to murmur classification,” *IEEE J. Biomed. Health Informat.*, vol. 26, no. 6, pp. 2524–2535, Jun. 2022.

[25] A. Dhillon and G. K. Verma, “Convolutional neural network: A review of models, methodologies and applications to object detection,” *Prog. Artif. Intell.*, vol. 9, no. 2, pp. 85–112, Jun. 2020.

[26] S. M. Anwar, M. Majid, A. Qayyum, M. Awais, M. Alnowami, and M. K. Khan, “Medical image analysis using convolutional neural networks: A review,” *J. Med. Syst.*, vol. 42, no. 11, p. 226, Nov. 2018, doi: 10.1007/s10916-018-1088-1.

[27] J. Cong, B. Liu, S. Neuendorffer, J. Noguera, K. Vissers, and Z. Zhang, “High-level synthesis for FPGAs: From prototyping to deployment,” *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 30, no. 4, pp. 473–491, Apr. 2011, doi: 10.1109/TCAD.2011.2110592.

[28] S. Lahti, P. Sjövall, J. Vanne, and T. D. Hämäläinen, “Are we there yet? A study on the state of high-level synthesis,” *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 38, no. 5, pp. 898–911, May 2019.

[29] Xilinx Inc. (2020). *Vivado Design Suite User Guide: High-Level Synthesis (UG902), V2019.2*. Accessed: Sep. 21, 2023. [Online]. Available: https://docs.xilinx.com/v/u/2019.2-English/ug902-vivado-high-level-synthesis

[30] L. Huiying, L. Sakari, and H. Iiro, “A heart sound segmentation algorithm using wavelet decomposition and reconstruction,” in *Proc. 19th Annu. Int. Conf. IEEE Eng. Med. Biol. Society. Magnificent Milestones Emerg. Opportunities Med. Eng.*, Jul. 1997, pp. 1630–1633, doi: 10.1109/IEMBS.1997.757028.

[31] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, “Optimizing FPGA-based accelerator design for deep convolutional neural networks,” in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*, Feb. 2015, pp. 161–170.

[32] F. Chollet et al. (2015). *Keras*. Accessed: May 2, 2023. [Online]. Available: https://keras.io

[33] (2013). *Chapter B1: AMBA AXI4-Lite*. Accessed: Sep. 21, 2023. [Online]. Available: https://developer.arm.com/documentation/ihi0022/e/AMBA-AXI4-Lite-Interface-Specification/AMBA-AXI4-Lite?lang=en

[34] ARM Ltd. (2021). *AMBA AXI-Stream Protocol Specification*. Accessed: Sep. 21, 2023. [Online]. Available: https://developer.arm.com/documentation/ihi0051/b/?lang=en

[35] P. Bentley, G. Nordehn, M. Coimbra, and S. Mannor. (2011). *The PASCAL Classifying Heart Sounds Challenge*. Accessed: Jan. 30, 2024. [Online]. Available: http://www.peterjbentley.com/heartchallenge/index.html

[36] 3M Littman. *Littman Library*. Accessed: Jan. 30, 2024. [Online]. Available: https://web.archive.org/web/20200223212248/http

[37] Med. School Univ. Michigan. *Heart Sound and Murmur Library*. Accessed: Jan. 30, 2024. [Online]. Available: https://www.med.umich.edu/lrc/psb_open/html/repo/primer_heartsound/primer_heartsound.html

Daniel Enériz (Graduate Student, IEEE) received the B.S. degree in physics and the M.S. degree in physics and physical technologies from the University of Zaragoza, Zaragoza, Spain, in 2019 and 2020, respectively, where he is currently pursuing the Ph.D. degree.

His current research at the Group of Power Electronics and Microelectronics, Aragon Institute for Engineering Research (GEPM-I3A) includes the design of electronic systems, intelligent instrumentation, and the edge computing of Neural Networks.

Antonio J. Rodriguez-Almeida received the B.S. degree in telecommunications engineering from the Universidad de Las Palmas de Gran Canaria, Las Palmas de Gran Canaria, Spain, in 2018, and the M.S. degree in biomedical engineering from the Universitat Politècnica de València, València, Spain, in 2020. He is currently pursuing the Ph.D. degree with the Institute for Applied Microelectronics, University of Las Palmas de Gran Canaria. His research interests include the development of deep learning models for chronic disease management and their hardware implementation.

Himar Fabelo received the master’s degree in telecommunication engineering and the Ph.D. degree in telecommunication technologies from the University of Las Palmas de Gran Canaria, Las Palmas de Gran Canaria, Spain, in 2014, and 2019, respectively.

Since then, he has conducted his research activity at the Institute for Applied Microelectronics, University of Las Palmas de Gran Canaria. In 2022, he obtained the Juan de La Cierva Formación Post-Doctoral Grant at the Fundación Canaria Instituto de Investigación Sanitaria de Canarias. His research interests include the use of machine learning techniques applied to hyperspectral images in tumor tissue analysis in real-time during surgery.

Samuel Ortega received the B.Sc. degree in telecommunication engineering and the M.Sc. and Ph.D. degrees in telecommunication technologies from the University of Las Palmas de Gran Canaria, Las Palmas de Gran Canaria, Spain, in 2015, 2016, and 2021, respectively.

In 2021, he started to work as a Post-Doctoral Researcher at the Norwegian Institute of Food Fisheries and Aquaculture Research (NOFIMA), Tromsø, Norway, where he was established as a permanent Research Scientist in 2022. In 2023, he began working part-time as a Research Scientist at UiT The Arctic University of Norway, Tromsø, while continuing his main position at NOFIMA. His research interests include the use of hyperspectral imaging and machine learning for medical and food quality applications.

Francisco J. Balea-Fernandez received the B.Sc. and Ph.D. degrees in psychology from the Pontifical University of Salamanca, Salamanca, Spain, in 2001 and 2007, respectively, the B.M. degree in medicine from the University of Las Palmas de Gran Canaria (ULPGC), Las Palmas de Gran Canaria, Spain, in 2011, the M.Sc. degree in clinical medicine from Camilo Jose Cela University, Madrid, Spain, in 2016, and the Ph.D. degree in biomedical research specializing in geriatrics from the ULPGC, Las Palmas de Gran Canaria, in 2021.

From 2001 to 2003, he did a course in the doctoral program of clinical neuropsychology at the University of Salamanca, Salamanca. He has been a Part-time Professor at ULPGC, since 2011.

Gustavo M. Callico (Senior Member, IEEE) received the M.S. degree in telecommunication engineering and the Ph.D. and European Doctorate degrees from the University of Las Palmas de Gran Canaria (ULPGC), Las Palmas de Gran Canaria, Spain, in 1995 and 2003, respectively.

In 2022, he was a Full Professor at ULPGC and developed his research activities Institute for Applied Microelectronics. His current research interests include hyperspectral systems for cancer detection, artificial intelligence algorithms, real-time super-resolution algorithms, synthesis-based design for SOCs, and circuits for multimedia processing and video coding standards.

Nicolás Medrano (Senior Member, IEEE) received the B.Sc. and Ph.D. degrees in physics from the University of Zaragoza, Zaragoza, Spain, in 1989 and 1998, respectively.

He is currently a Full Professor of electronics with the Faculty of Physics, University of Zaragoza, and a member of the Group of Power Electronics and Microelectronics of the Aragon Institute for Engineering Research (GEPM-I3A). His current research interests include hardware implementation of neural network models for signal processing, smart sensor interfaces, wireless sensor networks, and intelligent instrumentation.

Belén Calvo (Senior Member, IEEE) received the B.Sc. degree in physics and the Ph.D. degree in electronic engineering from the University of Zaragoza, Zaragoza, Spain, in 1999 and 2004, respectively.

She is currently a Full Professor of electronics with the Faculty of Physics, University of Zaragoza, and a member of the Group of Power Electronics and Microelectronics of the Aragon Institute for Engineering Research (GEPM-I3A). Her research interests include low-voltage low-power CMOS design, on-chip smart sensor interfaces, and edge neural network models.