VNUHCM Journal of Science and Technology Development 2026, 29(2):4136-4149

Open Access Full Text Article

Research Article

High Speed, Low Power for QRS Complex Detection Using a Convolutional Neural Network

Tran Thi Diem*

ABSTRACT

Cardiovascular disease (CVD) remains a major cause of mortality, particularly among young adults, underscoring the need for real-time cardiac monitoring. This study introduces an intellectual property (IP) /application-specific integrated circuit (ASIC) design based on a deep neural network (DNN) for accurate detection of QRS complexes, which are critical for arrhythmia diagnosis. By integrating deep learning, the system improves feature extraction and classification accuracy, ensuring robust detection under diverse physiological conditions. The architecture achieves high computational efficiency through optimized parallel multipliers in the convolutional layer, thereby reducing latency and hardware resource usage. Advanced physical-design techniques, including placement optimization and clock-tree synthesis, further improve speed and power efficiency. Experimental validation using the MIT-BIH Arrhythmia Database demonstrates strong performance, with a sensitivity of 99.797% and a positive predictive value of 99.936%. Implemented on a system-on-chip field-programmable gate array (SoC FPGA), the system operates at 324 MHz with a power consumption of 0.12 W. The final ASIC design, fabricated in 45 nm CMOS technology, operates at 430 MHz while consuming only 81 μW. These results represent a significant advancement in the development of efficient, low-power wearable cardiac monitoring systems for real-time ECG analysis and arrhythmia detection.

Key words: QRS complex detection, convolutional neural network, FPGA, SoPC, latency, 45nm CMOS process

University of Infomation Technology, VNU-HCM, Vietnam

Correspondence

Tran Thi Diem, University of Infomation Technology, VNU-HCM, Vietnam

Email: diemtt@uit.edu.vn

History

- Received: 08-03-2025
- Revised: 17-11-2025
- Accepted: 01-03-2026
- Published Online: 25-06-2026

DOI : https://doi.org/10.32508/vnuhcmj-std.v29i2.4435

Check for updates

INTRODUCTION

Cardiovascular diseases (CVDs) are the leading cause of mortality worldwide, underscoring the urgent need for advanced diagnostic technologies. Epidemiological studies indicate a concerning rise in the incidence of, and mortality from, CVDs among younger adults[1]. Electrocardiogram (ECG) signals provide a non-invasive means of monitoring the heart's electrical activity and play a crucial role in the diagnosis of cardiac conditions. These signals typically range from 0.05 Hz to 150 Hz but are vulnerable to various sources of noise, including baseline wander (typically <0.5 Hz) and power-line interference (typically 50–60 Hz)[2]. One of the key components of an ECG signal is the QRS complex, illustrated in Fig. 1, which comprises the Q, R, and S waves, as well as the P-R, S-T, and Q-T intervals. Among these features, the R peak is the most prominent and serves as a critical diagnostic marker of ventricular activity. Advanced analytical techniques, including clustering, classification, feature extraction, and signal synthesis, enable more precise interpretation of ECG data, thereby improving diagnostic accuracy and clinical decision-making.

Advances in QRS detection have involved both traditional algorithms and deep-learning-based methods, leading to substantial improvements in accuracy and robustness. Traditional approaches such as the stationary wavelet transform (SWT) have demonstrated high efficacy, achieving a sensitivity of 99.83 % and a positive predictive value (PPV) of 99.94 % on the MIT-BIH Arrhythmia Database[3]. In addition, real-time detection algorithms based on adaptive thresholding and mathematical morphology have shown strong performance, reaching 99.99 % sensitivity and 99.98 % PPV on the QT Database, even under noisy conditions[4].

More recently, deep learning models have further advanced QRS detection by improving feature extraction and classification accuracy. For example, a convolutional neural network (CNN) achieved an F1 score of 99.47 % on the MIT-BIH Arrhythmia Database, while a hybrid CNN–RNN model improved on this result, attaining an F1 score of 99.95 % on the same dataset[5,6]. Xiang et al.[7] introduced a two-level CNN model that achieved an F1 score of 99.89 %, while Belkadi et al.[8] developed a deep autoencoder model with sensitivity of 99.92 % and PPV of 99.85%. He et al.[9] proposed a hybrid U-Net–BiLSTM architecture that achieved a sensitivity of 99.93 % and an F1 score of 99.88 %, demon-

Cite this article : Diem T T. **High Speed, Low Power for QRS Complex Detection Using a Convolutional Neural Network** . *VNUHCMJ. Sci. Technol. Dev.* 2026; 29(2):4136-4149.

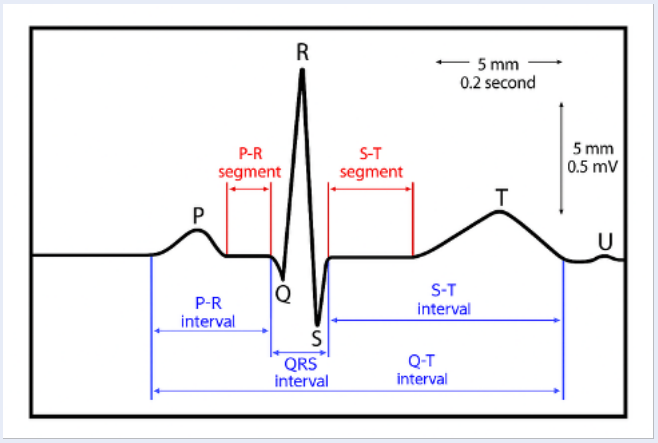

Figure 1: Annotated ECG waveform showing key cardiac intervals and segments. [Source: Author's]

strating robustness even in noisy environments. Collectively, these findings indicate that deep-learning-based methods can outperform traditional techniques for QRS detection, particularly under diverse and challenging signal conditions.

Field-programmable gate arrays (FPGAs) are widely used for hardware implementations because of their inherent parallelism, low power consumption, and high flexibility. Over the past two decades, numerous FPGA-based methods for QRS detection have been developed to improve performance and efficiency.

Meddah et al.[10] proposed a method that combines centered derivatives with amplitude and time thresholding, requiring a minimum delay of 0.2 s between detected R-peaks. Chowdhury et al.[11] introduced a fuzzy neural network approach that used 98 % of the available logic elements on an Altera FPGA device, highlighting its high computational demand. By contrast, Page et al.[12] designed a single-hidden-layer neural network that achieved a detection accuracy of 99.5 % while maintaining lower hardware resource utilization.

Panigrahy et al.[13] employed the Shannon energy envelope method, demonstrating high sensitivity and robustness. Meanwhile, the lifting-based discrete wavelet transform (DWT) method reported in[14] offered improved resource efficiency, albeit with a trade-off in detection accuracy. Collectively, these studies illustrate the trade-offs between detection accuracy, hardware resource utilization, and computational efficiency in FPGA-based QRS detection, highlighting the importance of architectures optimized for both performance and resource constraints.

To achieve a high-performance, low-latency, and low-power hardware design, this study introduces an intellectual property (IP) / application-specific integrated circuit (ASIC) architecture based on a deep neural network (DNN) for QRS complex detection. Compared with software-based solutions, this approach offers significant advantages in optimization of execution speed and energy efficiency. The design incorporates several key innovations, including optimized parallel multipliers in the convolutional layer, an efficient softmax computation technique, and advanced physical design optimizations to improve speed and power efficiency.

When deployed on a system-on-chip (SoC) field-programmable gate array (FPGA), the proposed architecture outperforms both CPU and GPU implementations in execution time, delivering high-speed performance. In this configuration, a fully integrated SoC architecture featuring an embedded convolutional neural network (CNN) IP for QRS complex detection operates at 324 MHz with a power consumption of only 0.12 mW. Furthermore, when fabricated using a 45 nm CMOS process, the IP core achieves an impressive 430 MHz operating frequency while consuming only 81 μW of power. The main contributions are as follows:

- **Architectural Contribution:** An optimized convolutional layer architecture based on a coarse-grained configurable computing architecture (CGCA). The design uses parallel execution units (PEs) with embedded on-chip RAM per PE, which significantly accelerates convolution operations. This improves throughput and enhances resource-utilization efficiency, a critical factor in both FPGA prototyping and ASIC implementation.
- **Physical Design Contribution:** In addition to the architectural design, physical layout optimization

Table 1: The proposed 1D convolutional neural network for QRS detection designed to process input segments, each containing 101 samples. [Source: Author's]

Layers	In_size In_channel	Filters	Out_size Out_channel	Kernel	Stride	Parameters	MAC
Conv1d	(101, 1)	32	(97, 32)	5	1	192	16,160
MaxPooling	-	-	(48, 32)	3	2	-	-
Conv1d	(48, 32)	32	(24, 32)	25	1	25,632	1,228,800
Dense	768	256	-	-	-	196,864	196,608
Dense	256	128	-	-	-	32,896	32,768
Dense	128	2	-	-	-	258	256
Total parameters: 255,842, Total MAC: 1,474,592							

Note: CNN size (MAC) is calculated as: MAC ≈ In_size × In_channel × Out_channel × Filters.

techniques are proposed, including customized placement strategies, clock tree synthesis, and routing enhancements. These techniques contribute to high-frequency operation (430 MHz), while maintaining ultra-low power consumption (81 μW) in a 45-nm CMOS ASIC. This level of optimization demonstrates the design's viability for real-world wearable and edge-based health-monitoring applications.

METHODOLOGY

The 1D CNN network for QRS detection in ECG signals is summarized in Table 1. After achieving the desired performance through platform-based network training, an IP CNN accelerator was developed for the inference phase. To optimize hardware resource utilization while maintaining accuracy, key techniques implemented including parallel pipelining in the convolutional layer and the elimination of the softmax layer.

Neural Network Architecture

To improve sensitivity (Se) and positive predictivity (P+) beyond those reported in previous studies, knowledge distillation [15] was employed to train the network. This technique transfers knowledge from a complex teacher model to a simplified yet efficient student model, enabling the development of a lightweight model that maintains performance comparable to that of the teacher model while minimizing computational demands. During training, the teacher model provides both true labels and additional probabilistic outputs, known as soft targets, generated by its softmax layer. These soft targets capture nuanced inter-class relationships, allowing the student model to learn intricate patterns more effectively and thereby improve overall performance.

The training process integrates two loss functions: cross-entropy loss, which aligns the student model with the ground-truth labels, and distillation loss, which ensures alignment with the soft targets provided by the teacher model. This dual-supervision approach enables the student model to replicate the performance of the teacher model while maintaining a reduced number of parameters. The overall loss function is expressed in Eq. 1:

$$L = \alpha \cdot L_{CE} + (1 - \alpha) \cdot L_{Distillation} \quad (1)$$

where α balances the two losses.

In this study, both the teacher and student models were implemented using the same 1D CNN architecture described in Table 1. The network processes input sequences of length 101 through multiple layers for efficient feature extraction and classification. The first convolutional layer (Conv1d) consists of 32 filters with a kernel size of 5 and a stride of 1, producing an output of size (97, 32). This is followed by a MaxPooling1d layer with a kernel size of 3 and a stride of 2, which reduces the output to (48, 32). The second convolutional layer (Conv1d_2) also uses 32 filters, but with a larger kernel size of 25 and a stride of 1, yielding an output size of (24, 32).

After feature extraction, the network transitions to fully-connected (Dense) layers. The first dense layer contains 256 units, the second contains 128 units, and the final output layer (Dense_2) has 2 units for classification. The entire network consists of 255,842 parameters, and has a total multiply-accumulate (MAC) count of 1,474,592 operations, indicating its computational efficiency for high-performance 1D signal processing.

Proposal Hardware Architecture of the IP Core on FPGA SoC

Proposed Hardware Architecture for the Convolutional Layer

The convolutional layer is the most computationally intensive component in hardware-based system architectures. In this study, an optimized convolutional-layer design is implemented directly in RTL, rather than relying on auto-generated RTL from high-level synthesis (HLS) tools, as in previous work (e.g., [16]). Algorithm 1 outlines an efficient 1D convolution scheme that utilizes parallel processing elements (PEs) together with on-chip RAM to enhance overall performance. By distributing computation across multiple PEs and incorporating pipelining techniques, the proposed architecture achieves substantial latency reduction while maximizing throughput. This design strategy provides high computational efficiency and scalability, making it particularly well-suited for real-time applications with stringent data-processing requirements.

The proposed convolutional architecture uses a parallel and pipelined design to improve the efficiency of 1D convolution operations through a 2D array of four Processing Elements (PEs), as shown in Fig. 2. To optimize resource utilization and align with the network architecture, in which the number of filters is a multiple of four, the design adopts a parallel structure consisting of four PEs. The system is organized into three main modules: the feature feeding system (FFS), the weight feeding system (WFS), and the pooling buffer, all of which are interconnected through the processing element array (PEA). During initialization, input features from previous layers are loaded into the local memory of each PE according to their mapped positions, allowing simultaneous memory access across all PEs.

As convolution begins, the PEs perform multiply-accumulate (MAC) operations in parallel, with each PE computing its assigned subset of the output channel space. The topology adopts a hierarchical loop structure, that distributes kernel and weight computations across the four PEs while exploiting spatial parallelism (m, m^*) and temporal pipelining to hide memory access latency. Intermediate results are accumulated within local variables (s^m) then adjusted with bias terms and written to the output buffer. The WFS supplies weights in a manner aligned with PE access patterns, thereby minimizing contention and maximizing throughput. Upon completion of computations, results are transferred through the PE output buffer to the pooling buffer for post-processing. This combination of intra-array parallelism and loop-level pipelining ensures high throughput and low latency, making the architecture suitable for real-time applications with demanding performance requirements.

Overview of the Previously Proposed Softmax Hardware Architecture

The softmax layer is commonly used as the final step in the classification stage of convolutional neural networks (CNNs). However, its conventional implementation requires substantial hardware resources because of the computational complexity of the exponential and division operations. To address this challenge, the present study incorporates an optimized softmax method previously proposed in [16]. Instead of performing full softmax computation, the proposed approach reduces hardware complexity while preserving classification performance. The baseline full-table computation architecture and the optimized architecture are shown in Fig. 3 and Fig. 4, respectively.

Workflow for Deploying the Inference AI Model on an SoC

Fig. 5 illustrates the workflow for deploying an inference AI model onto a system-on-chip (SoC) through multiple processing stages. The process begins with preprocessing, in which the AI model, stored as an .h5 file, is prepared together with input test data to ensure compatibility with the subsequent stages. Next, during the conversion phase, Python scripts are translated into C code, including a C-based testbench and model-weight conversions for High-Level Synthesis (HLS). The resulting code is then processed using Vitis HLS, where optimizations such as *#pragma hls pipeline, #pragma hls unroll, and #pragma hls array_partition* are applied to improve data throughput, execution speed, and parallel access efficiency while minimizing latency and resource usage.

Following this conversion stage, the convolutional and softmax layers are reimplemented, as described in Sections 2.2.1 and 2.2.2, to further optimize the architecture. In the SoC implementation phase, the optimized IP is integrated onto SoC platforms and rigorously tested across Xilinx and Altera FPGA architectures. This ensures robust performance, broad compatibility, and high computational efficiency. Through this streamlined workflow, the AI model is tailored to FPGA-based systems to maximize both performance and resource utilization.

Fig. 6 illustrates a system-on-chip (SoC) design that integrates a processing system (PS) and a programmable logic (PL) subsystem. The PS operates

Algorithm 1 Parallelized and pipeline 1-D Convolution

Input: $RAM_{x'\in\{0,1,2,3\}}^{m\in\{0,1,2,\ldots,M-1\}}[D']$, $weight[N \times K \times J]$, $bias[N]$
Output: $RAM_{x\in\{0,1,2,3\}}^{m\in\{0,1,2,\ldots,M-1\}}[D]$
Where: N, K, J, D, D', and Y' are the number of output channels, output channel size, kernel size, size of input memory, size of output memory and output size of the previous convolution layer, respectively;
for $k = 0$ to $K-1$ **do**
  for $y' = 0$ to $Y'-1$ **do**
    $m \leftarrow (k \times Y' + y')\%M$ ▷ m is the m^{th} PE
    $d \leftarrow (k \times Y' + y')/M$ ▷ d is the RAM address
    $RAM_{x'}^{m}[d] \leftarrow pixel[m]\ ; x' \in \{0,1,2,3\}$
for $n = 0$ to $N-1$ **do**
  for $y = 0$ to $Y-1$ with step size M **do**
    Each m^{th} PE operates in parallel:
    $s^m \leftarrow 0$ ▷ s^m Temporary variable for calculations
    for $k = 0$ to $K-1$ **do**
      for $j = 0$ to $J-1$ **do**
        $m^* \leftarrow (k \times Y' + y \times stride + j)\%M$
        $d^* \leftarrow (k \times Y' + y \times stride + j)/M$
        $w\_idx \leftarrow n \times K \times J + k \times K + j$
        $s^m \leftarrow s^m + weight[w\_idx] \times$
        $RAM_{x'}^{m^*}[d^*]$
      $pixel\_out[n \times Y + y] \leftarrow s + bias[n]$
      $m \leftarrow (n \times Y + y)\%M$
      $d \leftarrow (n \times Y + y)/M$
      $RAM_{x}^{m}[d] \leftarrow pixel\_out[m]\ ; x \in \{0,1,2,3\}$

Algorithm1: Proposal algorithm for the convolutional layer [Source: Author's]

within a GNU/Linux environment, utilizing an ARM Cortex-A53 processor to handle preprocessing and postprocessing tasks. To optimize data-transfer efficiency, a direct memory access (DMA) controller facilitates communication between DDR4 memory, which stores all network inputs and parameters, and the BRAM buffer within the PL subsystem through a 128-bit AXI bus. The design employs the AXI_full protocol, which supports advanced features such as burst-based data transfer, pipelining, and out-of-order transactions. This protocol comprises five independent channels—read address, read data, write address, write data, and write response—allowing simultaneous read and write operations with minimal latency and ensuring high throughput. These capabilities make AXI_full particularly suitable for the high-speed requirements of the convolutional neural network (CNN) implemented in the PL.

On the programmable logic (PL) side, the architecture is optimized for IP CNN-based beat detection. ECG waveforms, which serve as the input signals, are first stored in ECG signal RAM before being streamed to the CNN layers for processing. The CNN comprises convolutional, pooling, and dense layers, all managed by a dedicated controller module that oversees internal data flow within the PL and handles communication with the PS through the control bus. The CNN weights and biases are stored in dedicated RAM within the PL, ensuring low-latency access during computations. The dense layers generate the final prediction output, which is transmitted back to the PS through the AXI interface. The PS then performs postprocessing and distributes the results, completing the data-flow cycle. By leveraging the high-speed, burst-based transfer capabilities of AXI_full and the modular, scalable design of the AMBA architecture, the system enables efficient real-time signal processing for applications such as ECG beat detection.

Proposal Optimization Techniques in Physical Design

Design optimization techniques are essential for improving the performance and efficiency of physical-design processes, particularly for intellectual property

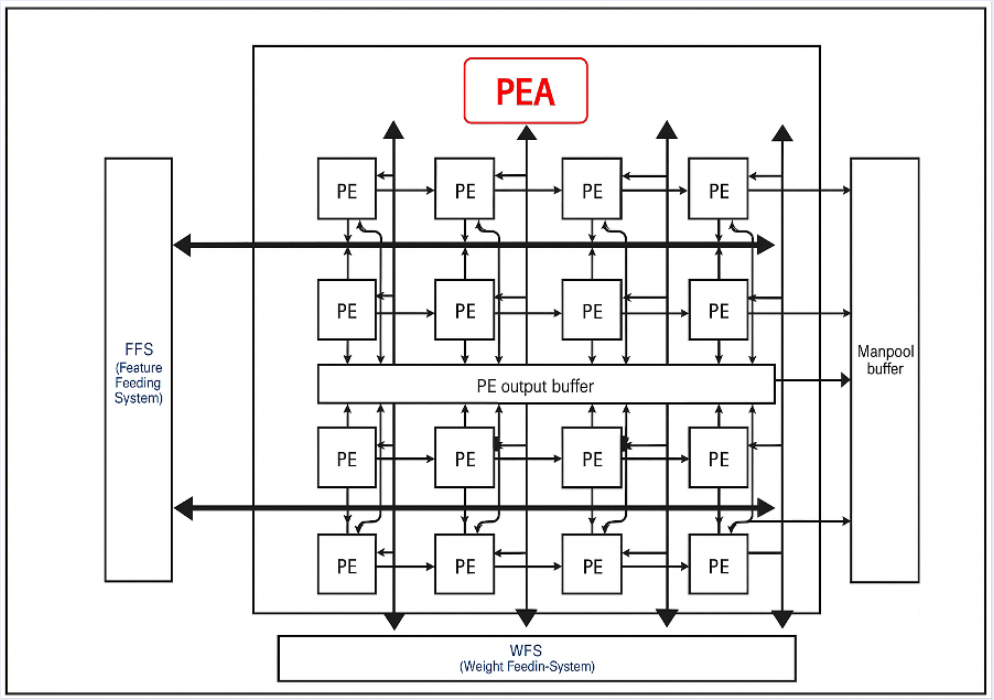

Figure 2: Proposed convolutional layer architecture with parallelism and pipelined execution. [Source: Author's]

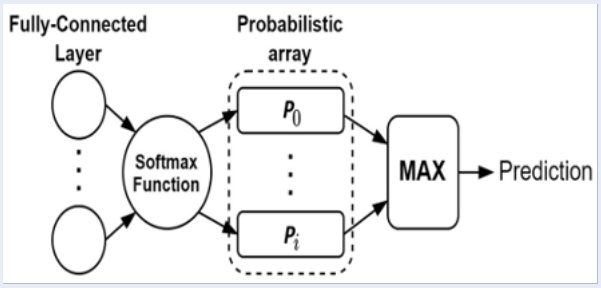

Figure 3: Softmax conventional architecture. [Source: Author's]

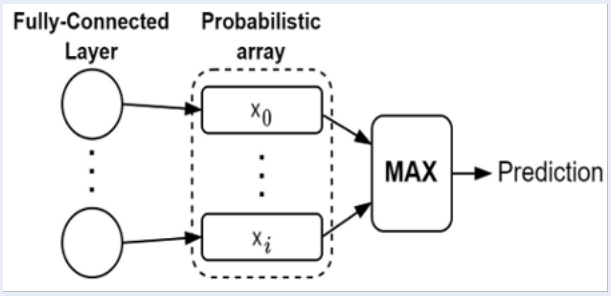

Figure 4: The softmax architecture proposed here[16]

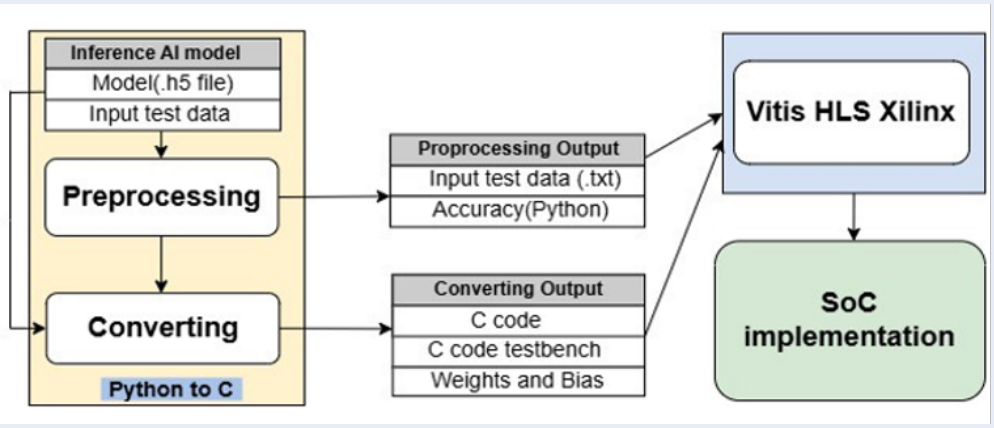

Figure 5: Workflow for deploying an inference AI model onto a SoC. [Source: Author's]

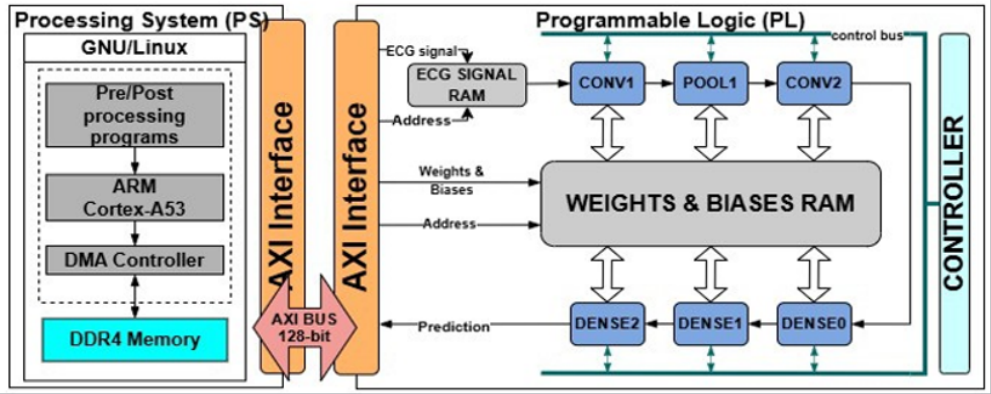

Figure 6: Details on SoC hardware design-integrated IP beat detection core. [Source: Author's]

(IP) cores. Table 2 outlines the key strategies and their respective advantages.

At the synthesis stage, replacing the conventional compile method with compile_ultra significantly improves operational speed, helping the synthesized design meet the stringent timing and performance requirements, particularly in high-frequency applications. During placement, reducing global maximum density (place_global_max_density 0.7) and increasing congestion effort (place_global_congestion_effort high) help mitigate local hotspots and routing congestion. These measures reduce design-rule-check (DRC) violations and minimize shorts, thereby improving overall layout quality and manufacturability.

In clock tree synthesis (CTS), adopting early clock flow (earlyClockFlow true) and useful-skew techniques (usefulSkewPreCTS true) improves clock-network stability and precision. In additional, employing aggressive clock-skew optimization modes (usefulSkewCCOpt standard/extreme) minimizes clock-related timing violations, which is critical for high-frequency operation in complex designs. At the routing stage, enabling useful skew after routing (usefulSkewPostRoute true) helps balance clock propagation and reduce timing-path violations, facilitating timing closure in highly constrained designs.

Collectively, these optimization techniques support the development of high-performance, scalable, and robust IP cores capable of operating at elevated frequencies. By addressing potential bottlenecks during synthesis, placement, CTS, and routing, these strategies help to reduce design iterations, accelerate convergence, and improve silicon implementation quality, making them valuable for modern physical design workflows.

Table 2: Comparison of default and proposed settings for physical design optimization. [Source: Author's]

Phase	Default Setting	Proposed Setting
Synthesis	compile	compile_ultra
Placement	Set PlaceMode - place_global_max_density 0.92	Set PlaceMode -place_global_max_density 0.7
	Set PlaceMode - place_global_congestion_effort low	Set PlaceMode - place_global_congestion_effort high
	Set OptMode -usefulSkewPreCTS none Set DesignMode -earlyClockFlow none	Set OptMode -usefulSkewPreCTS true set DesignMode -earlyClockFlow true
Clock Tree Synthesis	Set OptMode -usefulSkewCCOpt none	Set OptMode -usefulSkewCCOpt standard/extreme
Route	Set OptMode -usefulSkewPostRoute none	Set OptMode -usefulSkewPostRoute true

RESULTS AND DISCUSSION

The MIT-BIH Arrhythmia Database[17], which comprises 48 annotated 30-minute recordings from 47 individuals, was used to evaluate the proposed hardware design. Following[16], four records (102, 104, 107, and 217) containing paced beats were excluded, leaving 44 records. These were evenly divided into two groups of 22 for training and evaluation, ensuring a balanced distribution of beats across the two sets. Each group included a mix of routine and complex arrhythmias, thereby preserving dataset representativeness for robust model assessment.

To prepare the ECG signals for analysis, they were resampled from 360 Hz to 250 Hz to reduce computational complexity while preserving sufficient resolution for accurate QRS detection. The signals then underwent preprocessing, including baseline-wander removal and signal normalization, as described in[16]. Each QRS complex was represented by 101 data points, covering a 100 ms window before and a 300 ms window after the R-wave position annotated in the dataset. This window length ensures the inclusion of both the QRS complex and the surrounding features needed for accurate classification.

For labeling, points within a ± 40 ms range of a positive beat annotation were classified as positive, whereas those outside this range were labeled as negative, creating a clear distinction between true and false detections.

Experimental Setup

After preprocessing, the model was trained for 100 epochs with a batch size of 128 and a dropout rate of 0.5 to prevent overfitting. The Adam optimizer (momentum 0.9, learning rate 0.001) was used with binary cross-entropy as the loss function, following[16]. Both the teacher and student models were trained using the same configuration in TensorFlow on an NVIDIA GeForce RTX 3090 GPU for high-speed processing. The hardware design was implemented in C/C++ using Xilinx High-Level Synthesis (HLS), enabling efficient design verification.

Real-time validation was conducted on a Xilinx ZCU102 FPGA, allowing faster QRS detection evaluation compared to simulation alone. Vivado 2020.2 was used for AXI4FULL interface design, BRAM generation, and IP integration. The system used the Zynq UltraScale+ MPSoC architecture, in which the processing system (PS) handled system-level operations, while the logic components and BRAMs were implemented in the programmable logic (PL) domain. In addition, the IP CNN accelerator was designed using Cadence Innovus with the 45 nm Nangate Open Cell Library.

Model performance was evaluated using sensitivity of the metric (Se), positive predictive value (+P), and latency, defined as follows:

$$Sensitivity\ (Se) = \frac{TP}{TP+FN} \tag{2}$$

$$Positive\ Predictive\ Value\ (+P) = \frac{TP}{TP+FP} \tag{3}$$

where a detected beat is classified as a true positive TP if it occurs within a ± 75 ms window of the annotated beat. If no detection occurs within this range, it is classified as a false negative (FN). Any detection that falls outside of the ± 75 ms window is treated as a false positive (FP).

$$Latency = \frac{Number\ of\ clock\ cycles}{Clock\ frequency} \tag{4}$$

where the number of clock cycles is the total number of cycles required to complete the operation and clock frequency is the operating frequency of the FPGA, measured in Hertz (Hz).

Results of the FPGA logic and SoC Design

As shown in Table 3, the proposed method achieves strong performance among software-based approaches, with Se = 0.99836 and +P = 0.99928, exceeding the results reported for several other deep learning models. For example, Silva et al. [18] reported Se = 0.9981 and +P = 0.9993, while the two-level CNN developed by Xiang et al.[7] achieved Se = 0.9977 and +P = 0.9991. The stacked autoencoder DNN[8] and traditional methods, such as the centered derivative[10] showed lower performance, with Se values below 0.9976 and a noticeable reduction in +P. Classical signal-processing techniques, including the lifting-based discrete wavelet transform, performed less well, with both Se and +P values below 0.996. These results indicate the high sensitivity and accuracy of the proposed 1D CNN-based approach for feature extraction.

As presented in Table 4, the hardware implementation of the proposed method achieve Se = 0.99797 and +P = 0.99936, which are slightly lower than the corresponding software results because of inherent arithmetic limitations. However, it provides a better balance between sensitivity and false positives compared to the Shannon energy envelope technique[13] (Se = 0.99947, +P = 0.99894). Other hardware-based approaches, such as the centered derivative and intermediate value theorem methods[10] (Se = 0.9983, +P = 0.9956), as well as the entropy measure of fuzziness[11], showed lower accuracy, with Se values ranging from 0.9931 to 0.9947 and +P values below 0.9956.

Table 5 presents the hardware resource utilization of the proposed system implemented on the ZCU102 FPGA, covering both the ECG IP core and the complete SoC system. It reports the percentages of key FPGA resources used, including LUTs, LUTRAMs, flip-flops (FFs), block RAMs (BRAMs), DSP slices, and BUFGs, relative to the available resources. For the ECG IP core, resource utilization remains exceptionally low, with only 0.84 % of LUTs, 0.02 % of LUTRAMs, 0.27 % of FFs, 0.38 % of BRAMs, and 0.56 % of DSP slices in use, while BUFG usage is 0.00%. This minimal footprint highlights the efficiency of the ECG IP core in its use of FPGA resources. When scaled to the full SoC system, resource utilization increases moderately, with 3.59 % of LUTs, 1.07 % of LUTRAMs, 2.11 % of FFs, and 13.32 % of BRAMs utilized. DSP-slice usage remains unchanged at 0.56 %, while BUFG utilization increases slightly to 0.25 %. Despite this increase, the overall resource consumption remains well within the FPGA's capacity, ensuring efficient hardware deployment. The system achieves a maximum operating frequency of 324 MHz, making it well-suited for high-speed, real-time applications. Power consumption is measured at 0.12 W for the ECG IP core and 4.104 W for the complete SoC system, demonstrating the design's effectiveness in achieving low power consumption and strong performance, particularly in real-time ECG signal processing.

Fig. 7 compares latency (s) and throughput (GOP/s) across three processing platforms: CPU, GPU, and FPGA. In terms of latency, the FPGA achieves the best performance, with the lowest latency at 0.0016 s, which is 38.5 % lower than that of the CPU (0.0026 s) and 50 % lower than that of the GPU (0.0032 seconds). In terms of throughput, the FPGA also outperforms the other platforms, delivering the highest value of 0.92 GOP/s, which is 62.3 % higher than that of the CPU (0.567 GOP/s) and 100 % higher than that of the GPU (0.46 GOP/s). The lower performance of the GPU is attributed primarily to a data-transfer bottleneck, as it processes individual data segments sequentially rather than in parallel batches, which are more typical during training. These results indicate that the FPGA provides the most favorable balance of latency and throughput among the three platforms, making it well-suited to high-performance applications in which these metrics are critical.

Table 6 provides a comprehensive comparison of the proposed method with previous works[10,13,14,16,20] on the ZCU102 platform in terms of hardware resource utilization and computational efficiency. The proposed design substantially reduces LUT and FF usage compared with the most resource-intensive methods[10,13], using less than half the LUTs and only 2 % of the registers reported in[13]. Compared with the CNN-based design[16], it achieves a 5 % reduction in both LUT and FF usage while maintaining the same DSP and BRAM utilization, indicating improved logic efficiency.

However, it uses slightly more LUTs and FFs than certain wavelet-based designs[14,20] after normalization of DSP and BRAM usage. BRAM and DSP utilization are identical to[16] but higher than those in[14], while power consumption is slightly higher than that reported in[16] and several times higher than that of the ultra-efficient design in[10]. Nevertheless, this increase is associated with the higher operating frequency, which is up to eight times that of[14] and within 2 %

Table 3: Comparison of software-based QRS detection [Source: Author's]

Methodology	Se	+P
1-D CNN (Proposed)	0.99836	0.99928
1-D CNN [18]	0.9981	0.9993
Two-level CNN [7]	0.9977	0.9991
Stacked autoencoder DNN [3]	0.9976	0.9924
Centred derivative and intermediate value theorem [10]	0.9958	0.9968
Lifting-Based Discrete Wavelet Transform [14]	0.9947	0.9920
CNN [18]	0.9935	0.9929

Table 4: Hardware-based QRS detection comparison [Source: Author's]

Methodology	Se	+P
1-D CNN (Proposed)	0.99797	0.99936
Shannon energy envelope [13]	0.9947	0.9984
Centred derivative and intermediate value theorem [10]	0.9983	0.9956
Entropy measure of fuzziness [11]	0.9974	0.9974
Kalman filtering-based adaptive threshold [19]	0.9939	0.9931
Lifting-Based Discrete Wavelet Transform [14]	0.9947	0.9920

Table 5: Comparison of hardware-resource between the study's IP ECG Core and previous research [Source: Author's]

Resource	ECG IP Utilization (%)	SoC Utilization (%)	Available
LUT	2305 (0.84%)	9872 (3.59%)	274080
LUTRAM	23 (0.02%)	1545 (1.07%)	144000
FF	1437 (0.27%)	11561 (2.10%)	548160
BRAM	3.5 (0.38%)	121.5 (13.32%)	912
DSP	14 (0.56%)	14 (0.56%)	2520
BUFG	0 (0.00%)	1 (0.25%)	404
Maximum Frequency	324 MHz	324 MHz	
Power Consumption	0.12W	4.104W	

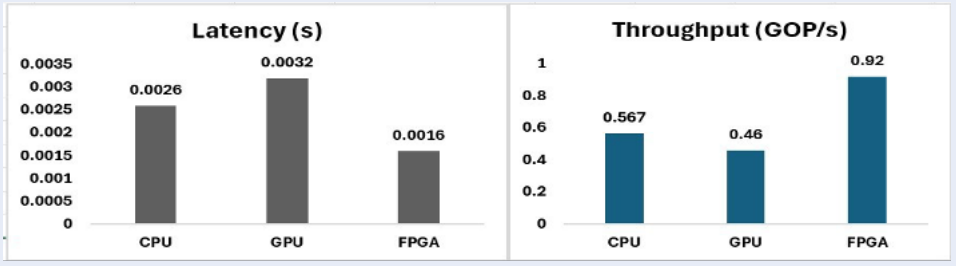

Figure 7: Comparison of latency and throughput per segment (101 samples) between CPU (Intel Core i9), GPU (NVIDIA GeForce RTX 3090) and FPGA (ZCU102) [Source: Author's]

Table 6: Hardware resource comparison between our IP ECG Core with previous research [Source: Author's]

Metric	Centred Derivative and Intermediate Value [10]	Lifting-Based Discrete Wavelet Transform [14]	Biorthogonal Spline Wavelet Transform [20]	Shannon Energy Envelope [13]	1D-CNN (ZCU102) [16]	Our Proposed (ZCU102)
Device	Ncxy-4	Artix-7	Virtex-6	Virtex-5	ZCU102	ZCU102
LUTs	7157	2197	2102	5728	2196	2305
Registers	2542	585	1320	88456	1355	1437
BRAMs	N/A	2	N/A	N/A	3.5	3.5
DSPs	4	N/A	N/A	N/A	14	14
Frequency	N/A	44 Mhz	N/A	N/A	320 Mhz	324 Mhz
Power (W)	0.042	N/A	N/A	N/A	0.11	0.12

of that in [16]. This frequency advantage is important because the design targets periodic batch-mode ECG analysis (e.g., hourly or daily) rather than continuous real-time sampling at 1–500 Hz. The ability to process large volumes of accumulated ECG data rapidly with low latency makes this high-throughput, high-frequency architecture well-suited to modern energy-aware deployments.

Physical Design Results

To determine the optimal IP floor plan size, three designs were evaluated: Trial 1 (3780 μm $\times$ 3780 μm), Trial 2 (3000 μm $\times$ 3000 μm), and Trial 3 (2800 μm $\times$ 2800 μm). As shown in Fig. 8, Trial 1 resulted in unused space at the top and bottom after placement, indicating an oversized and inefficient layout. By contrast, Trial 2 (3000 μm $\times$ 3000 μm) provided a more balanced design, with acceptable congestion levels, density distribution, and layout efficiency. Although Trial 1 exhibited fewer congestion hotspots, it wasted space, making Trial 2 the better compromise between floor plan size and performance. Trial 3 (2800 μm $\times$ 2800 μm) further reduced the floor plan size but caused severe congestion and uneven density, leading to timing violations and placement irregularities that prevented design completion during routing. Trial 2 (3000 μm $\times$ 3000 μm) was therefore selected as the optimal floor plan because it achieved efficient resource utilization without compromising performance.

The optimization techniques discussed in Section 2.3 significantly improve design performance. Table 7 compares the default and optimized settings, highlighting the key modifications that lead to these gains. Reducing global density from 92 % in the default configuration to a more efficient value improves resource allocation. Enabling "useful skew" and "early clock flow", which were previously disabled, enhances timing control and reduces clock delay. Increasing the timing and congestion effort levels from medium to high improves the accuracy of timing analysis and helps alleviate layout congestion.

Power analysis shows a negligible increase in static leakage power, while dynamic internal power increases because of higher activity levels and clock rates. However, dynamic switching power decreases slightly, indicating improved switching efficiency, likely because of clock tree optimizations. The most significant improvement is the increase in design frequency, demonstrating the effectiveness of these optimizations in achieving a more efficient and reliable design.

Table 8 compares the proposed design with previous studies and shows significant improvements across several key performance metrics. Using an advanced 45 nm process, the design achieves higher integration density and greater efficiency, offering up to four times the scalability of older 180 nm [21], 65 nm [22], and 350 nm [23] technologies. Operating at 0.95 V, it reduces power consumption by nearly 50 % compared with earlier implementations running at 1.8 V [21,24], in line with modern low-power requirements. With a chip area of 9.00 mm^2, the design balances compactness and performance: it is more than 100 times larger than the 65 nm design in [22] but remains approximately 30 % smaller than the 350 nm design in [23]. Achieving an operating frequency of 430 MHz and 450 MHz, the design outperforms previous implementations by up to 1400 times, particularly when compared to 1 MHz [21] and 0.3 MHz [24], demonstrating substantially higher processing speed.

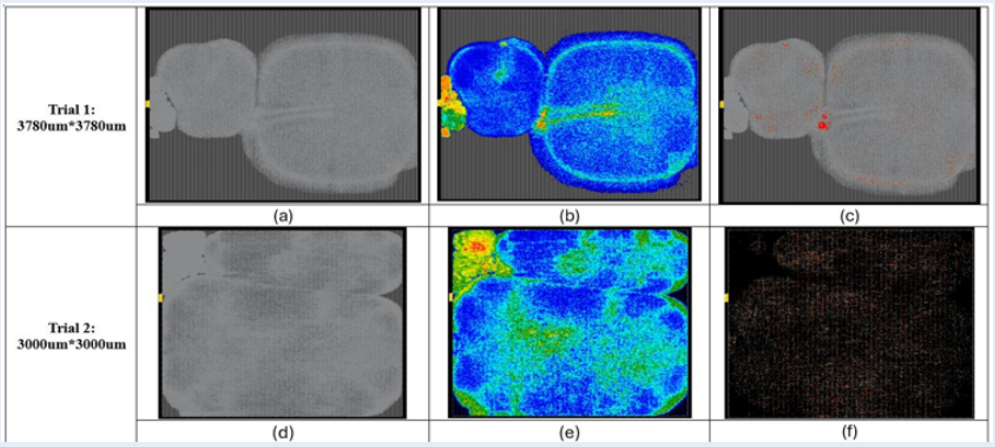

Figure 8 : Congestion, highlight density, and floor plan detail for two trials (Trial 1: 3780um × 3780um, Trial 2: 3000um×3000um). (a-c) show the congestion, highlight density, and detailed floorplan of trial 1, respectively, while panels (d-f) show the corresponding results for trial 2. [Source: Author's]

Table 7: Comparison of results between default and optimized settings [Source: Author's]

Version	Default Setting	Optimization Setting
Global density	Default (92%)	YES
Useful skew / Early clock flow	Default (NO)	YES / ENABLE
Effort timing	Default (Medium)	High
Effort congestion		High
Power Summary		
Static - Leakage	79.49 nW	80.45 nW
Dynamic - Internal	318.15 mW	340.18 mW
Dynamic - Switching	368.95 mW	358.45 mW
Design Frequency	200 MHz	420 MHz

Note: Both versions have the same floorplan size.

The high operating frequency reported here is consistent with the intended application. The objective is to integrate the QRS detection IP into wearable devices, enabling rapid and efficient real-time QRS complex detection over user-defined time intervals such as seconds, minutes, hours, or days. By contrast, the design of low-frequency IPs for real-time ECG signal acquisition represents a separate aspect of system-level optimization.

In addition, power consumption is reduced to 81 μW, which is substantially lower than that reported by competing designs [23,25,26], reinforcing the energy-efficient nature of the architecture. With an accuracy of 99.8 %, the design also exceeds previous benchmarks by more than three percentage points, outperforming the 96.55% reported in [23], ensuring reliability and robustness. Overall, the proposed design provides strong performance in frequency, power efficiency, and accuracy while maintaining a compact footprint and leveraging advanced fabrication technology, making it well-suited to high-performance wearable applications.

CONCLUSION

This study presents a hardware accelerator for QRS complex detection using a deep neural network (DNN), addressing key requirements in wearable cardiac monitoring. The design emphasizes energy efficiency and high-speed performance by optimizing parallel multipliers in the convolutional layer, which reduce latency and hardware resource usage. In addition, physical-design optimizations improve both operating frequency and power efficiency. Validation on a SoC FPGA using the MIT-BIH database demonstrates strong performance, achieving a sensitivity of 0.99797, a positive predictive value of 0.99936, and a

Table 8: Comparison of ASIC benchmarks with this work [Source: Author's]

Benchmarks	ASSCC 2009 [21]	ASSCC 2014 [22]	ICECS 2020 [23]	ITBE 2012 [24]	AS 2023 [25]	Ms 2020 [26]	This Work	This Work
Tech (nm)	180	65	65	350	55	90	45 (NanGate)	12
Fabrication	CHRT	CMOS	GF	CMOS	UMC	CMOS	—	TSMC
Voltage (V)	1.8	1.0	1.0	1.8	1.2	NA	0.95	0.8
Area (mm^2)	1.1	0.07	12.3	1.11	0.33	1.9	9.00	1.21
Frequency (MHz)	1M	1K	—	0.3	100K	NA	430M	450M
Hardware Type	ASIC	ASIC	ASIC	SoC	ASIC	ASIC	ASIC	ASIC
Memory (kB)	—	1.6	—	128	—	NA	—	—
Power (μW)	176.00	0.55	184.71	0.83	12.88	9,780.00	81	94.87
Accuracy (%)	99.63	98.65	96.55	99.31	96.69	NA	99.8	99.8

power consumption of 0.12 W at 324 MHz, while outperforming CPU- and GPU-based solutions in both latency and throughput. When implemented in 45nm CMOS technology, the design achieves even further gains, operating at 430 MHz while consuming only 81 μW. Overall, this work provides a robust, high-performance solution for on-body cardiac monitoring, meeting the stringent requirements of wearable devices and paving the way for future innovations in low-power arrhythmia detection systems.

ACKNOWLEDGMENT

This research is funded by Vietnam National University Ho Chi Minh City (VNU-HCM) under grant number B2026-26-06.

COMPETING INTERESTS

The author declares that there are no competing interests.

AUTHORS' CONTRIBUTIONS

Thi Diem Tran conceived the study, designed the methodology, conducted the experiments, analyzed and interpreted the data, and wrote and revised the manuscript. The author read and approved the final manuscript.

REFERENCES

1. Zhao D. Epidemiological features of cardiovascular disease in Asia. JACC: Asia. 2021;1(1):1–13.
2. Li C, Zheng C, Tai C. Detection of ECG characteristic points using wavelet transforms. IEEE Transactions on Biomedical Engineering. 1995;42(1):21–28.
3. Belkadi MA, Daamouche A. A robust QRS detection approach using stationary wavelet transform. Multimedia Tools and Applications. 2021;80(15):22843–22864.
4. Yazdani S, Vesin JM. Extraction of QRS fiducial points from the ECG using adaptive mathematical morphology. Digital Signal Processing. 2016;56:100–109.
5. Lee JS, Lee SJ, Choi M, Seo M, Kim SW. QRS detection method based on fully convolutional networks for capacitive electrocardiogram. Expert Systems with Applications. 2019;134:66–78.
6. Islam MS, Islam MN, Hashim N, Rashid M, Bari BS, Farid FA. New hybrid deep learning approach using BiGRU-BiLSTM and multilayered dilated CNN to detect arrhythmia. IEEE Access. 2022;10:58081–58096.
7. Xiang Y, Lin Z, Meng J. Automatic QRS complex detection using two-level convolutional neural network. Biomedical Engineering Online. 2018;17:1–17.
8. Belkadi MA, Daamouche A, Melgani F. A deep neural network approach to QRS detection using autoencoders. Expert Systems with Applications. 2021;184:115528.
9. He R, Liu Y, Wang K, Zhao N, Yuan Y, Li Q, et al. Automatic detection of QRS complexes using dual channels based on U-Net and bidirectional long short-term memory. IEEE Journal of Biomedical and Health Informatics. 2020;25(4):1052–1061.
10. Meddah K, Talha MK, Bahoura M, Zairi H. FPGA-based system for heart rate monitoring. IET Circuits, Devices & Systems. 2019;13(6):771–782.
11. Chowdhury SR. Field programmable gate array based fuzzy neural signal processing system for differential diagnosis of QRS complex tachycardia and tachyarrhythmia in noisy ECG signals. Journal of Medical Systems. 2012;36:765–775.
12. Page A, Kulkarni A, Mohsenin T. Utilizing deep neural nets for an embedded ECG-based biometric authentication system. In: 2015 {IEEE} Biomedical Circuits and Systems Conference ({BioCAS}). IEEE; 2015. p. 1–4.
13. Panigrahy D, Rakshit M, Sahu PK. FPGA implementation of heart rate monitoring system. Journal of Medical Systems. 2016;40(3):49.
14. Gon A, Mukherjee A. FPGA-based low-cost architecture for R-peak detection and heart-rate calculation using lifting-based discrete wavelet transform. Circuits, Systems, and Signal Processing. 2023;42(1):580–600.

15. Tran TD, Le KT, Nguyen ALT. Training low-latency deep spiking neural networks with knowledge distillation and batch normalization through time. In: 2022 5th International Conference on Computational Intelligence and Networks ({CINE}). IEEE; 2022. p. 1–6.
16. Ngo BT, Thai NHH, Pham HL, Tran TD. A high-performance FPGA based on convolutional neural network for QRS complex detection. In: 2024 15th International Conference on Computing Communication and Networking Technologies ({ICCCNT}). IEEE; 2024. p. 1–6.
17. Moody GB, Mark RG. The MIT-BIH arrhythmia database on CD-ROM and software for use with it. In: Proceedings Computers in Cardiology. IEEE; 1990. p. 185–188.
18. Silva P, Luz E, Wanner E, Menotti D, Moreira G. QRS detection in ECG signal with convolutional network. In: Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications: 23rd Iberoamerican Congress, {CIARP} 2018, Madrid, Spain, November 19-22, 2018, Proceedings 23. Springer; 2019. p. 802–809.
19. Zhang Z, Yu Q, Zhang Q, Ning N, Li J. A Kalman filtering based adaptive threshold algorithm for QRS complex detection. Biomedical Signal Processing and Control. 2020;58:101827.
20. Berwal D, Kumar A, Kumar Y. Design of high performance QRS complex detector for wearable healthcare devices using biorthogonal spline wavelet transform. ISA Transactions. 2018;81:222–230.
21. Phyu MW, Zheng Y, Zhao B, Liu X, Wang YS. A real-time ECG QRS detection ASIC based on wavelet multiscale analysis. In: 2009 {IEEE} Asian Solid-State Circuits Conference. IEEE; 2009. p. 293–296.
22. Bayasi N, Saleh H, Mohammad B, Ismail M. 65-nm ASIC implementation of QRS detector based on Pan and Tompkins algorithm. In: 2014 10th International Conference on Innovations in Information Technology ({IIT}). IEEE; 2014. p. 84–87.
23. Tefai HT, Saleh H, Tekeste T, Alqutayri M, Mohammad B. ASIC implementation of a pre-trained neural network for ECG feature extraction. In: 2020 {IEEE} International Symposium on Circuits and Systems ({ISCAS}). IEEE; 2020. p. 1–5.
24. Ieong CI, Mak PI, Vai MI, Martins RP. Sub-μw QRS detection processor using quadratic spline wavelet transform and maxima modulus pair recognition for power-efficient wireless arrhythmia monitoring. In: 2016 21st Asia and South Pacific Design Automation Conference ({ASP-DAC}). IEEE; 2016. p. 21–22.
25. Zhang C, Chang J, Guan Y, Li Q, Wang X, Zhang X. A low-power ECG processor ASIC based on an artificial neural network for arrhythmia detection. Applied Sciences. 2023;13(17):9591.
26. QING L. Electrocardiogram QRS detection hardware accelerator for ASIC implementation; 2020.