Real-time Classification of Bird Vocalizations with CNN on a System-on-Chip Applying Hardware–Software-Co-Design Techniques and High-Level Synthesis

Knut Pröpper, Anastasiia Purtova, Lutz Leutelt
Faculty of Engineering and Computer Science
Hamburg University of Applied Sciences
Hamburg, Germany
{knut.proepper | anastasiia.purtova | lutz.leutelt}@haw-hamburg.de

***Abstract*—This research aimed to evaluate the use of high-level synthesis (HLS) within the hardware-software codesign paradigm with application to efficient implementation of convolutional neural networks (CNN) for bird vocalization classification.**

The classification model was trained using supervised learning with selected and preprocessed bird vocalizations from the Xeno-Canto database of wildlife. The preprocessing and feature extraction of the audio data involves noise reduction, silence removal and calculation of psycho-acoustically weighted MEL spectrograms which were used for CNN model training. With the applied Sequential CNN classifiers an accuracy of approx. 94% was achieved.

For implementing on an AMD Zynq UltraScale+ MPSoC, the Python solution of the classifier was analyzed module by module with a focus on the design principles of hardware software co-design and HLS. Depending on the requirements of the module, preprocessing and feature extraction modules are implemented either as software, hardware description language (HDL) or HLS-based hardware acceleration, while CNN inference is performed on the processing system using the PYNQ framework. The different variants are compared in terms of latency, resource usage and development effort.

The evaluation showed that HLS is highly effective for abstract, dataflow-oriented signal processing tasks, such as spectrogram generation. It can significantly reduce development time without compromising performance. However, traditional HDLs remain superior for low-level interfaces such as I2S audio input due to their precise control over timing and protocol details. The CNN implementation using PYNQ has shown that high-level frameworks can effectively integrate with custom hardware modules to create a cost-efficient, real-time audio classification system on an SoC.

***Index Terms*—Mel-spectrogram, convolutional neural networks, system-on-chip, hardware-software-co-design, high-level-synthesis**

I. Introduction

Recognizing bird species from audio recordings is a challenging task due to the variability of vocalization patterns, background noise, and recording conditions. Recent advances in digital signal processing and machine learning have enabled the development of automated systems that classify bird vocalizations based on acoustic characteristics (e.g. [1], [2]). Convolutional Neural Networks (CNNs) trained on preprocessed labeled data have demonstrated strong performance when applied to mel-spectrogram representations of bird songs [2].

As part of this work, several data preparation steps were undertaken. All recordings were obtained from the Xeno-Canto database [4] and filtered for the Hamburg region to focus on the most frequently represented species. After filtering redundant data, the dataset contained 347 species. For the analysis, however, only the 10 most common species were selected. Additional preprocessing steps, including normalization, noise reduction, and data augmentation, resulted in a standardized dataset of 1,800 records for further analysis. A sequential CNN architecture, consisting of convolutional and fully connected layers, was then trained on mel-spectrograms and achieved a classification accuracy of approximately 94%. This trained model provides the foundation for the system described in this paper.

Although such software-based solutions achieve high accuracy on desktop hardware, their deployment on embedded platforms requires efficient mapping of algorithms to hardware to meet constraints on latency, power consumption, and resource usage. The increasing availability of computational resources and flexible System-on-Chip (SoC) platforms opens new possibilities for embedded applications in signal processing and machine learning.

In this paper, we extend the trained CNN classifier by porting it to an AMD Zynq UltraScale+ MPSoC [13]. We investigate the role of software, high-level synthesis (HLS) and hardware description languages (HDL) within a hardware-software codesign approach. By systematically partitioning the classification pipeline into modules implemented in software, HDL, or HLS, we analyze trade-offs in terms of latency, resource utilization, and development effort. Our contributions are threefold:

- we present a trained CNN-based bird vocalization classifier with high accuracy, and

- we demonstrate its real-time implementation on an SoC, and
- we provide a comparative evaluation of software, HLS, and HDL approaches, yielding practical guidelines for embedded AI applications.

II. Related Work

For classification of bird songs methods of both, conventional sound processing and more recently CNN approaches are applied. In [1] the focus is on conventional sound processing for classifying bird calls within a single song. Features are extracted from spectrograms using singular value decomposition (SVD) to represent key properties of bird syllables. The approach introduces an ambiguity spectrum, a frequency- and time-shift-invariant transformation of multitaper spectrograms. Similarity between syllables is measured using a cosine similarity method, and hierarchical clustering determines the number of clusters automatically. The method emphasizes syllable segmentation, distinguishing between single and double syllables, which affects classification and may require manual inspection to correct clustering errors.

Another approach [2] presents bird call classification using two CNN-based image recognition models, ResNet and Inception, as part of the BirdCLEF 2019 competition. The approach is similar to our methodology, as it clusters bird sounds based on their spectrograms and maps them to species. Bird sounds are represented by 1-second MEL-scale log-amplitude spectrograms, reflecting human auditory sensitivity. Spectrograms with low signal-to-noise ratios were discarded. To address class imbalance, image augmentation methods (e.g., Gaussian blur, random cropping, brightness adjustment) were applied, though the authors later noted some methods may have corrupted training data. Normalization was applied based on the mean and variance of the training set. Both CNN models were trained and compared, with Inception performing better, likely due to more input parameters. Challenges included noisy recordings and overlapping bird calls.

The problem of poor labeling in the Xeno-Canto database is addressed in [3], which focuses on pre-processing to reduce label noise. The authors designed a "labeling function" to classify audio as correctly labeled or noise. Audio files were processed using short-term Fourier transform. Two clustering methods were applied: unsupervised DBSCAN and supervised CNN BirdNET-Lite, with outputs compared to identify bird sounds versus noise. This method suggests an opportunity to improve dataset quality by filtering mislabeled data and selecting the clearest signals, and could be further adapted for multi-label classification where multiple species sing simultaneously.

In the past, various studies have been conducted on the efficient implementation of the necessary signal processing steps on embedded hardware and FPGAs.

A hardware implementation of Mel-Frequency Cepstral Coefficients (MFCC)-based feature extraction for speaker recognition on an FPGA platform (Xilinx Virtex-II) is presented in [5]. The work focuses on optimizing computational complexity and memory usage by using look-up tables and fixed-point arithmetic. By exploiting parallelism and pipelining, the FPGA-based solution achieves efficient real-time processing compared to conventional software approaches. The results show that MFCC extraction can be effectively implemented in hardware, offering a cost-effective and scalable alternative to DSP-based methods.

A Hardware-based acceleration for feature extraction in automatic speech recognition (ASR) using the MFCC algorithm on a Xilinx Zynq-7000 SoC is applied in [6]. The approach parallelizes computationally intensive steps such as Fast fourier Transform (FFT), mel-filtering, and Discrete Cosine Transform (DCT) to exploit FPGA hardware capabilities. Compared to sequential CPU and ARM implementations, the FPGA design achieved up to 500$\times$ and $\times$ speedup respectively, demonstrating the significant advantage of parallelized MFCC processing in real-time ASR systems.

Another hardware chip design for implementing Mel-Frequency Cepstral Coefficients (MFCC) feature extraction in speech recognition systems is presented by [7]. The focus is on reducing computational complexity and memory requirements by employing a hybrid look-up table scheme for efficient calculation of elementary functions, while fixed-point arithmetic minimizes hardware cost without sacrificing accuracy. The design, implemented on a Xilinx XC4062XL FPGA, demonstrates an area-efficient architecture for real-time MFCC extraction.

A hardware/software co-design approach for real-time image and video processing on a Xilinx Zynq SoC using high-level synthesis (HLS) is used in [8]. Morphological operations such as dilation, erosion, and convolution were implemented as hardware accelerators, integrated with software on the ARM cores. The design achieved significant reductions in execution time compared to software-only implementations while using limited FPGA resources, demonstrating the feasibility of HLS-based co-design for embedded real-time image processing tasks

HW/SW co-design of the post-quantum cryptographic scheme BIKE on embedded platforms with Xilinx Zynq-7000 SoCs is introduced in [9]. High-level synthesis (HLS) was employed to generate hardware accelerators for computationally intensive primitives (e.g., polynomial multiplications, SHAKE, SHA-3). The approach optimizes the area-performance trade-off for small FPGAs and achieves speedups ranging from 1.37$\times$ to 2.78$\times$ compared to software-only execution, depending on the target device. The paper highlights the potential of HLS-based co-design for PQC implementations under tight resource constraints

An embedded system for automatic classification of six Omani date fruit varieties [10] using a hardware/software co-design on a Xilinx Zynq-7020 SoC Color aextracts shape-size features from preprocessed and segmented images and classified with an ANN. Profiling identified feature extraction and preprocessing as bottlenecks, which were accelerated in FPGA logic using Vivado HLS and SDSoC. The resulting design achieved 97.26% classification accuracy and real-time performance of 10.9 fps, representing a 14× speedup over the

baseline software implementation

Another implementation of a Convolutional Neural Network (CNN) for object recognition on an embedded FPGA platform (ZedBoard with a Xilinx Zynq SoC) uses the PYNQ framework [11]. The model is trained offline on CPU and then deployed for inference. Key performance: 100 ms latency per image and about 10 images/second throughput on the CIFAR-10 dataset, with an accuracy of 79.90%. The implementation uses only a single ARM core on the FPGA for inference; resource and energy constraints are important considerations.

III. Methodology

A. Dataset

The dataset used in this work was collected from the community-driven Xeno-Canto [4] repository, which provides access to bird vocalizations from across the world. An initial query for the Germany region yielded more than 37,000 recordings. After filtering out incomplete and irrelevant records (e.g., grasshoppers, unknown entries, and soundscape recordings), the dataset was reduced to approximately 33,500 audio files representing 347 bird species. Since the dataset was highly unbalanced, we restricted the classification task to the ten most common species, such as the Great Tit and Common Chaffinch, which together provided a sufficiently large number of samples for training and evaluation.

B. Data Preparation

The raw audio recordings were preprocessed in several steps to improve quality and consistency. First, noise reduction was applied using spectral subtraction methods implemented in librosa. Second, silence removal and normalization were performed to eliminate non-informative sections. Both manual thresholds (e.g., –30 dB for Chaffinch, –60 dB for Tits) and adaptive thresholds relative to the peak amplitude were evaluated. The recordings were then segmented into fixed-length clips of four seconds, ensuring uniformity across the dataset.

C. Data Augmentation

To further balance the dataset and increase robustness against variability, each class was augmented by splitting longer recordings into multiple four-second segments. This process expanded the dataset to approximately 2,000–2,700 segments, depending on the preprocessing strategy, and produced more homogeneous training examples.

D. Feature Extraction

For each segment, Mel-spectrograms were computed as input features. The spectrograms were generated with 128 Mel filters, a Fast Fourier Transform (FFT) size of 2048, a hop length of 512 samples, and a sampling rate of 22.05 kHz. The resulting time-frequency representations were log-scaled to approximate human auditory perception. Each spectrogram was resized to 224×224 pixels to match the CNN input requirements.

E. CNN Model Training

A sequential CNN was designed for the classification task. The architecture consisted of convolutional layers with ReLU activation, max-pooling layers, and fully connected dense layers, ending with a softmax output over the ten bird classes. The model was trained using the Adam optimizer and categorical cross-entropy loss, with a batch size of 32 and 10 epochs. Across multiple experiments, the best-performing configuration achieved a validation accuracy of approximately 94%, particularly when spectral subtraction and normalization were combined in preprocessing. This trained CNN forms the foundation of the embedded implementation.

F. System Architecture on SoC

The trained model was deployed on an AMD Zynq UltraScale+ MPSoC (Kria KV260) platform [12]. To exploit the heterogeneous architecture, a hardware–software co-design strategy was applied, assigning each module of the audio classification pipeline to the most suitable execution domain. The functional roles of the modules are as follows:

- **Audio Input Unit (AIU):** Implemented in HDL and HLS, the AIU receives audio data via the I^2S interface and converts it into an AXI stream with precise timing guarantees.
- **Signal Framing Unit (SFU):** Implemented both in software (on the RPU) and in HLS for comparison. The SFU applies FIR filtering, performs downsampling, segments the signal into frames, and applies a Hann window function.
- **Mel Processing Unit (MPU):** Designed in HLS with pipelining and dataflow optimizations. The MPU computes the mel-spectrogram as described in [7], including FFT and mel filterbank calculations.
- **Audio Classification Unit (ACU):** Executed on the processing system (PS) using the PYNQ framework. The ACU runs the trained CNN to perform the final audio classification task.

This modular partitioning enabled a systematic evaluation of software, HLS, and HDL realizations in terms of latency, resource utilization, and development effort. The overall interaction of the modules is illustrated in the flow diagram in Fig. 1, which provides an overview of the end-to-end processing chain. The results of the evaluation are discussed in Section V.

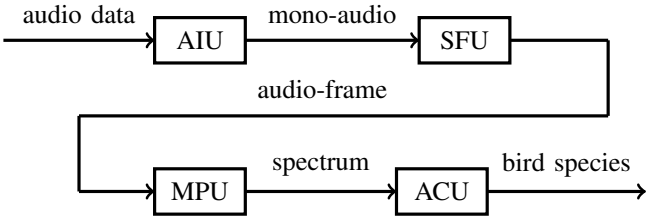

Fig. 1. Block diagram of the audio processing pipeline on the MPSoC

IV. Implementation

A. Audio Input Unit

The AIU is responsible for capturing audio samples from the I2S audio interface and providing them as a 16-bit AXI stream. Two implementations were created:

- VHDL implementation: This variant was adapted from an existing design. Using a mature VHDL base ensured reliable operation of the I2S protocol and AXI-Stream interface with minimal additional effort. VHDL remains well suited for such protocol-level modules because it allows precise cycle-level control.
- HLS implementation: In parallel, the AIU was re-implemented in C++ and synthesized with Vitis HLS. This design decision was motivated by the need to evaluate whether HLS could be a viable alternative for simple interface modules.

The comparison thus reflects both the reuse of a proven HDL design and the potential of HLS to replicate its functionality.

B. Signal Framing Unit

The Signal Framing Unit segments the incoming audio into frames of 2048 samples with 50% overlap and applies a Hanning window after downsampling the input from 44.1 kHz to 22.05 kHz. Two equivalent implementations were realized:

- Software implementation: The algorithm was first implemented on the ARM RPU core. This required only minimal development effort since the framing procedure consists of simple arithmetic operations.
- HLS implementation: The same algorithm was synthesized with Vitis HLS. Only minor modifications were necessary (e.g., AXI-Stream interface integration).

By developing both versions, the SFU provides a direct comparison of software versus HLS in terms of latency, resource usage, and development effort. The design choice illustrates how lightweight preprocessing stages can either be kept in software for flexibility or offloaded to hardware for increased throughput.

C. Mel Processing Unit

The Mel Processing Unit computes the spectrogram using FFT, Mel filterbank, and logarithmic scaling. Two HLS variants were developed to investigate the impact of algorithmic and synthesis-level optimizations:

- Unoptimized HLS: The initial version directly translated the algorithm into C++ and synthesized it without pragmas. This design required very little effort and served as a functional baseline to assess throughput and resource usage under default HLS conditions.
- Optimized HLS: To meet real-time constraints and guided by approaches from [5] and [7], the design was restructured at the algorithmic level. Specifically, the FFT was replaced by a real-valued FFT, reducing redundant operations, and the Mel filterbank was optimized to eliminate unnecessary multiplications. In addition, synthesis directives such as pipelining, loop unrolling, and dataflow were applied to further increase throughput.

This stepwise design choice reflects a deliberate trade-off: the unoptimized variant minimized development effort but failed to achieve the required performance, whereas the optimized variant required significantly more design work—both in algorithmic reformulation and in synthesis-level tuning—but successfully reduced latency to real-time levels.

D. Audio Classification Unit

The Audio Classification Unit executes the trained CNN. In this work, inference was implemented in software using the PYNQ framework on the PS. The model was deployed in Python, maintaining compatibility with the training environment. No hardware accelerator was implemented for the CNN; instead, the focus was on validating real-time operation of the end-to-end pipeline and demonstrating the seamless integration of software-based inference with hardware accelerators for preprocessing.

E. Integrated System

The complete system integrates all modules via AXI-Stream interfaces. Four variants were built to systematically evaluate different design choices:

- AIU: implemented in both VHDL and HLS.
- SFU: only the HLS version was integrated, since the PYNQ framework required a hardware-based streaming interface.
- MPU: implemented in HLS, in both unoptimized and optimized form.
- ACU: implemented in software on the Processing System (PS) using PYNQ [14].

This setup resulted in four end-to-end system configurations, summarized in Table IV-E

TABLE I
Breakdown of variants

Variants	AIU	SFU	MPU	ACU
1	VHDL	HLS	HLS (Std.)	PYNQ
2	HLS	HLS	HLS (Std.)	PYNQ
3	VHDL	HLS	HLS (Opt.)	PYNQ
4	HLS	HLS	HLS (Opt.)	PYNQ

The final design was implemented and verified in Vivado. Fig. 2 shows the generated block design, including the streaming connections between AIU, SFU, and MPU in the programmable logic, and the ACU running on the processing system under PYNQ.

V. Results

A. Audio Input Unit

The AIU was implemented in VHDL and HLS. Both versions correctly received the I2S input and produced a 16-bit AXI stream without synchronization errors. Latency was one cycle in both cases. Resource utilization is shown in Table

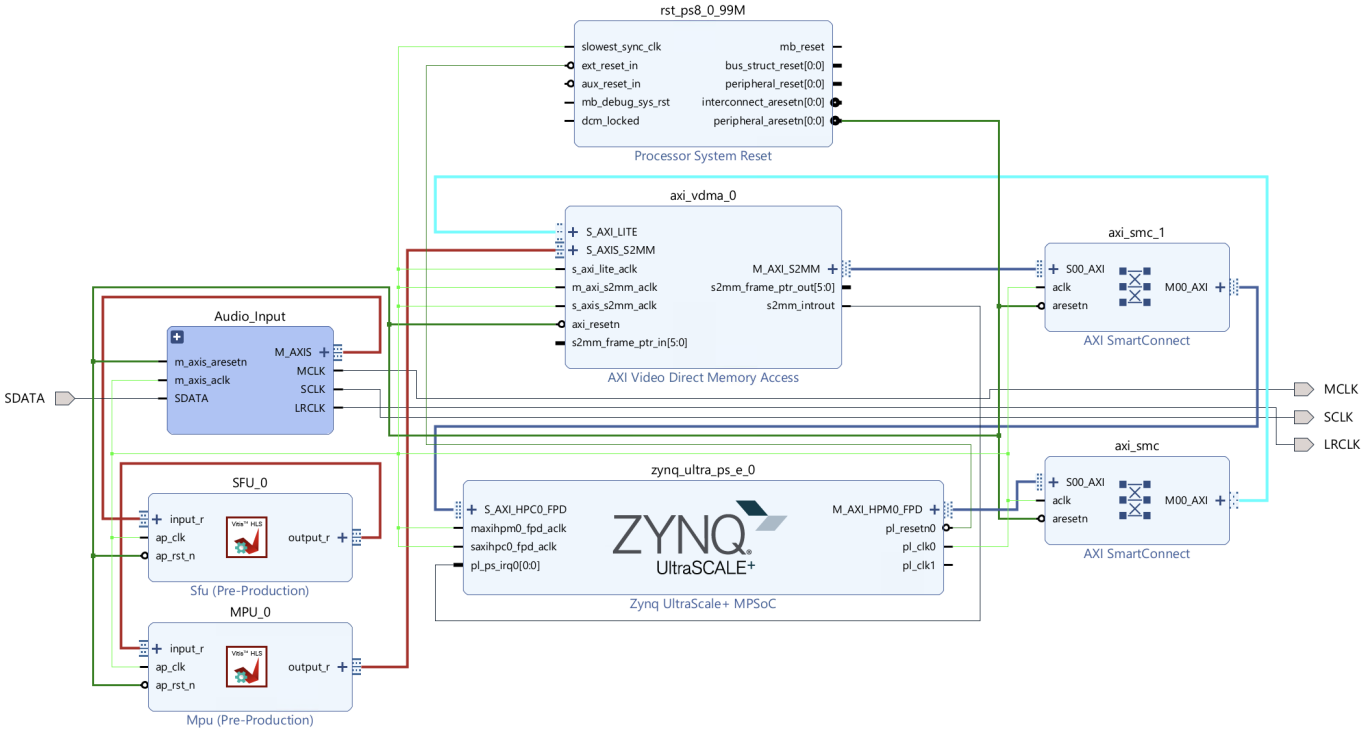

Fig. 2. Block diagram of the implemented hardware design

V-A. Development effort was higher for HLS due to manual handling of timing and state machines like with a HDL, while VHDL allowed an better testbench.

TABLE II
RESOURCE UTILIZATION OF AIU IMPLEMENTATIONS

Variant	DSP	LUT	FF	BRAM
VHDL	0	12	77	0
HLS	0	36	75	0

B. Signal Framing Unit

The SFU was realized as a software implementation on the RPU and as an HLS design. Both variants correctly performed downsampling, segmentation, and windowing. Measured latencies and resources are given in Table V-B. The software version required minimal effort, while HLS needed additional setup for AXI-Stream interfaces but could be synthesized directly from the same algorithmic code.

TABLE III
LATENCY AND RESOURCE UTILIZATION OF SFU IMPLEMENTATIONS

Variant	Latency [μs]	DSP	LUT	FF	BRAM
Software	2.1	-	-	-	-
HLS	0.3	3	799	222	2

C. Mel Processing Unit

The MPU was implemented in two HLS variants. Both generated correct spectrograms compared to reference data. Latency and resource utilization are summarized in Table V-C.

TABLE IV
LATENCY AND RESOURCE UTILIZATION OF MPU IMPLEMENTATIONS

Variant	Latency [μs]	DSP	LUT	FF	BRAM
HLS Standard	169	20	4737	5364	14
HLS Optimized	86.4	43	5905	6386	14

D. Audio Classification Unit

The ACU executed CNN inference in software on the Processing System (PS) using PYNQ [14]. Accuracy was 94%, identical to the training baseline. After initialization, latency stabilized at 400 ms per classification. The implementation effort was minimal, as the trained Python model could be deployed directly in the PYNQ framework.

E. Integrated System

All modules were integrated into a full pipeline. End-to-end operation was validated with ILA. Total latency for a 4-second frame was 410 ms, dominated by the ACU. Without the ACU, latency was 172 µs (standard MPU) and 90 µs (optimized MPU). Resource usage of the complete design is listed in Table V-E.

TABLE V
RESOURCE UTILIZATION OF INTEGRATED SYSTEM VARIANTS

Variants	DSP	LUT	FF	BRAM
1	23	7983	9300	17
2	23	8017	9298	17
3	46	9104	10261	17
4	46	9142	10251	17

VI. Discussion

The evaluation of the four system variants highlights distinct strengths and limitations of VHDL, HLS, and software approaches within a hybrid SoC design.

AIU (interface-oriented logic): Both VHDL and HLS implementations were functionally identical. However, the VHDL design required slightly more manual effort but offered precise control over protocol timing. The HLS version enabled faster prototyping but did not provide measurable performance benefits for this low-level task. This confirms that VHDL remains advantageous for protocol-bound modules where cycle accuracy is critical.

SFU (lightweight preprocessing): Both software and HLS implementations of the SFU fulfilled real-time requirements. The processor implementation was highly flexible and easy to develop, while the HLS version reduced latency even further with moderate resource usage. In the integrated system, only the HLS version could be used due to PYNQ's streaming-based architecture. This shows that software remains viable for simple signal processing, but practical integration requirements can favor hardware implementations.

MPU (compute-intensive processing): The contrast between unoptimized and optimized HLS designs demonstrates the impact of algorithmic reformulation and synthesis directives. Applying a real-valued FFT and optimized Mel filterbank reduced latency by nearly 50%, at the cost of increased resources. This underlines that HLS can provide significant performance gains, but only when combined with algorithm-level optimizations.

ACU (CNN inference): Running the CNN in software with PYNQ provided functional correctness and seamless deployment, but introduced the dominant share of system latency (400 ms per frame). While sufficient for validation, this highlights the need for hardware acceleration or model compression in future work.

Integrated system: The hybrid architecture validated the overall concept: preprocessing and feature extraction in programmable logic (VHDL/HLS) combined with flexible inference in software. Resource utilization remained moderate in all variants, leaving sufficient headroom for future extensions.

VII. Conclusion

In this paper, we presented a hardware–software co-design approach for real-time classification of bird vocalizations on an AMD Zynq UltraScale+ MPSoC. Building on a trained CNN model that achieved 94% accuracy on mel-spectrogram inputs, we mapped the classification pipeline onto modular implementations using software, high-level synthesis (HLS), and hardware description language (HDL).

The evaluation demonstrated that:

- HDL is the most effective choice for interface-oriented modules, such as the I2S audio input, where precise timing is critical.
- HLS provides an efficient and developer-friendly solution for dataflow-oriented signal processing tasks such as spectrogram generation, reducing development effort while still meeting real-time constraints.
- Software execution on the processing system remains highly valuable for CNN inference and integration, offering flexibility and rapid prototyping despite limited efficiency.

References

[1] M. Große Ruse, D. Hasselquist, B. Hansson, M. Tarka, and M. Sandsten, "Automated analysis of song structure in complex birdsongs," Animal Behaviour, vol. 112, pp. 39–51, 2016.

[2] Incze, H.-B. Jancs´o, Z. Szil´agyi, A. Farkas, and C. Sulyok, "Bird sound recognition using a convolutional neural network," in 2018 IEEE 16th International Symposium on Intelligent Systems and Informatics (SISY), pp. 000295–000300, 2018.

[3] F. Michaud, J. Sueur, M. Le Cesne, and S. Haupert, "Unsupervised classification to improve the quality of a bird song recording dataset," Ecological Informatics, vol. 74, p. 101952, 2023.

[4] Willem-Pier Vellinga ; Robert Planqué ; Sander M. Pieterse: www.xeno-canto.org: a decade on.

[5] Ehkan, Phaklen ; Zakaria, Fazrul F. ; Warip, MNM ; Sauli, Zaliman ; Elshaikh, Mohamed: Hardware Implementation of MFCC-Based Feature Extraction for Speaker Recognition. 339, S. 471–480. – ISSN 978-3-319-07673-7

[6] Choo, Chang ; Chang, Young-Uk ; Moon, Il-Young: FPGA-Based Hardware Accelerator for Feature Extraction in Automatic Speech Recognition. 13, Nr. 3, S. 145–151.

[7] Wang, Jia-Ching ; Wang, Jhing-Fa ; Weng, Yu-Sheng: Chip design of MFCC extraction for speech recognition. 32, Nr. 1, S. 111–131.

[8] M. S. AZZAZ, A. MAALI, R. KAIBOU, I. KAKOUCHE, M. SAAD and H. HAMIL, "FPGA HW/SW Codesign Approach for Real-time Image Processing Using HLS," 2020 1st International Conference on Communications, Control Systems and Signal Processing (CCSSP), El Oued, Algeria, 2020, pp. 169-174, doi: 10.1109/CCSSP49278.2020.9151686.

[9] A. C. Ammari, L. Khriji and M. Awadalla, "HW/SW Co-'esign for Dates Classification on Xilinx Zynq SoC," 2020 26th Conference of Open Innovations Association (FRUCT), Yaroslavl, Russia, 2020, pp. 10-15, doi: 10.23919/FRUCT48808.2020.9087548.

[10] G. Montanaro, A. Galimberti, E. Colizzi and D. Zoni, "Hardware-Software Co-Design of BIKE with HLS-Generated Accelerators," 2022 29th IEEE International Conference on Electronics, Circuits and Systems (ICECS), Glasgow, United Kingdom, 2022, pp. 1-4, doi: 10.1109/ICECS202256217.2022.9970992.

[11] Dhouibi, M., Salem, A.K.B. and Saoud, S.B., 2020, October. CNN for object recognition implementation on FPGA using PYNQ framework. In 2020 IEEE Eighth International Conference on Communications and Networking (ComNet) (pp. 1-6). IEEE.

[12] AMD-Xilinx, "Kria KV260 Vision AI Starter Kit: User Guide," Advanced Micro Devices, Inc., San Jose, CA, USA, 2021. [Online]. Available: https://www.amd.com/de/products/system-on-modules/kria/k26/kv260-vision-starter-kit.html

[13] AMD, Zynq UltraScale+ Device Technical Reference Manual (UG1085), Version 2.5, March 2025. [Online]. Available at: https://docs.amd.com/v/u/en-US/ug1085-zynq-ultrascale-trm, accessed: September 21, 2025.

[14] AMD, PYNQ: Python Productivity for Zynq, [online] Available at: http://www.pynq.io/, accessed: September 21, 2025.