SCCTS Transactions on Reconfigurable Computing, ISSN: 3049-1533 Vol. 3, No. 2, 2026 (pp. 66-78)
RESEARCH ARTICLE ECEJOURNALS.IN

A High-Level Synthesis-Driven Framework for Application-Specific Reconfigurable Processor Design in AI Workloads

Noel Unciano[1]*, El Manaa Barhoumia[2]

[1]Environment and Biotechnology Division- Industrial Technology Development Institute, Philippines
[2]College of Applied Science, University of Technology and Applied Sciences, Ibri, Sultanate of Oman

Keywords:
High-Level Synthesis (HLS); Application-Specific Processor; Reconfigurable Computing; FPGA; AI Workloads; Hardware/Software Co-Design; Domain-Specific Architecture; Resource Optimization; Deep Learning Acceleration; Edge AI

Author's Email:
nmuetmicro@gmail.com

DOI : 10.31838/RCC/03.02.08

Received : 22.12.2025

Revised : 17.02.2026

Accepted : 11.05.2026

Abstract

Due to the upsurge in the computational requirements of the artificial intelligence (AI) workloads, in particular, limited-corpus edge settings, the interest in customized and energy-efficient processor architectures is growing much faster. Within this paper we present an application-specific reconfigurable chip grounded on the high-level synthesis (HLS)-driven framework with optimizations of AI tasks. The suggested methodology closes the gap between algorithmic descriptions and the implementation on the hardware level because HLS tools can be used to automatically synthesis optimized hardware accelerators based on C/C++ models. The framework includes a modular approach to designing that confers the ability to quickly prototype AI-specific processing elements, efficiently orchestrate dataflows, and dynamically adapt to an extensive variety of neural network models. Important characteristics are the workload profiling, reusable HLS-based intellectual property (IP) cores assembled by common AI operations, and the interchangeable architecture that accepts resource-conscious scheduling and tile-based integration. Experimental testing on FPGA systems reflects that latency, power usage, and resource utilization increase greatly when abstracted away with FPGA-based systems when compared to the methods of conventionally using RTL-based designs and the inference engines using GPUs. The paper demonstrates the opportunities of reconfigurable processor design based on HLS as a scalable and flexible implementation of AI deployment at the edge, and will be further extended in terms of dynamic partial reconfiguration and integration of heterogeneous systems.

How to cite this article: Unciano N, Barhoumia EM (2026). A High-Level Synthesis-Driven Framework for Application-Specific Reconfigurable Processor Design in AI Workloads. SCCTS Transactions on Reconfigurable Computing, Vol. 3, No. 2, 2026, 66-78

Introduction

The accelerated spread of artificial intelligence (AI) into applications like computer vision, natural language processing, autonomy processes, and predictive analytics has raised the new level of computing requirements. The demands are especially critical in edge and embedded systems where constraints on power, area and latencies are much stricter than in cloud-based systems.[1] The traditional general-purpose processors (GPPs) and even graphics processing units (GPUs) are good at dealing with

parallel computations but fail to address the energy efficiency needs and domain specificity needed to perform edge AI inference tasks.

Recently, FPGAs have shown promise as potential alternatives to their use, and indeed, their very nature, in recent years, reconfigurability, high parallelism and energy-efficient design have shown them to be quite viable alternatives. In contrast to fixed-function ASICs, FPGAs enable hardware optimization on the level of application needs which is especially useful when it comes to running different and changing AI models.[2] They are also well suited to edge computing applications in autonomous vehicles, drones, and wearable devices because they are designed to be used in real-time and low-latency settings.[3]

But the factor that has hindered broad scale use of FPGAs in AI workloads is the common Register Transfer Level (RTL) design path that requires extensive expertise in hardware design, high development times, and low levels of scalability. In order to avoid this, High-Level Synthesis (HLS) has emerged as a very powerful way of allowing designers to specify functionality in terms of a higher level programming language, most often C/C++, which are compiled automatically into hardware description languages (HDLs) to be used to implement the Fpga design.[4] In addition to the hardware development process being faster using HLS, HLS will facilitate rapid design space exploration making it a perfect tool when developing AI models iteratively.

HLS can be applied to the realm of AI, where application-specific reconfigurable processors can be created using HLS and optimized to support different layers of neural networks - e.g. convolution, activation, pooling and fully connected operations, as well as resource sharing and reuse, pipelining and even custom memory hierarchies.[5] The processors can be optimized to provide high throughput and ensure minimal power which are essential user metrics in power conscious systems of edge AI.

The proposed idea in this paper is a design framework of 1the HLS-driven application-specific reconfigurable processors on an AI application. The framework presents a modular and scalable structure combining AI kernel profiling, HLS to IP core, tile-based processor design with configurable processor assembly, and resource-sensitive optimization mechanism. Making the software design and deployment of custom processor architectures automated, the suggested solution overlaps the performance difference between general and hardwired accelerators without losing the flexibility needed to accommodate future applications of AI.

This work will be useful in publishing the important contributions, as well as the development of a high-level, compiler-based design process that

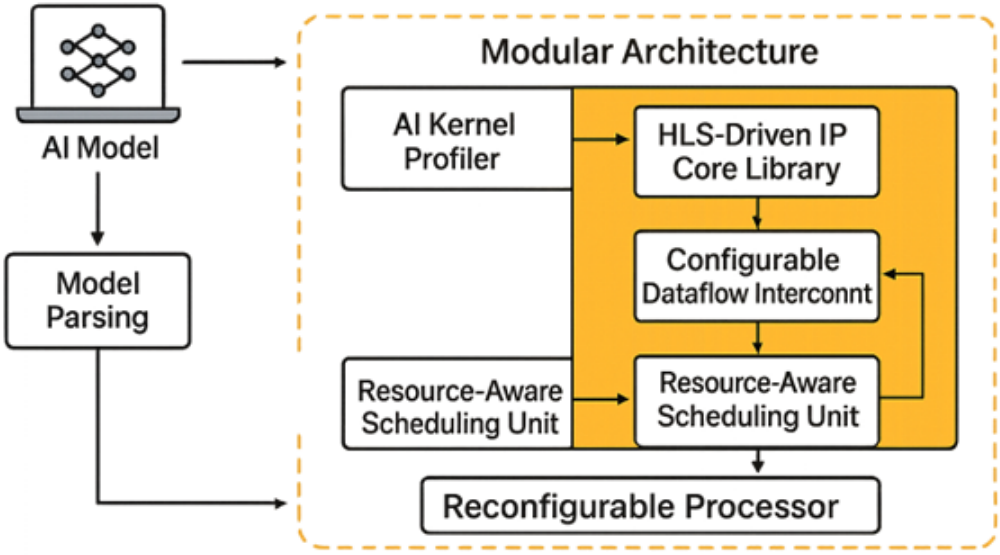

Fig. 1: High-Level Synthesis-Driven Framework for Application-Specific Reconfigurable Processor Design in AI Workloads

can in a successful way aim at mapping AI workload on reconfigurable hardware using the available high-level synthesis (HLS) tools. Crucial to this framework is an extensive library of HLS-based intellectual property (IP) cores that are reusable to be used in support of significant AI operations such as convolution, activation, pooling, and so on. The specified architecture will help to build a reconfigurable processor whose application is specific dynamic and capable of changing the workload situation on the processor, in addition to the utilization optimization of the resources. Additionally, the validity of the design is supported by a thorough experimentation on FPGA platforms and demonstrates an impressive inference latency, power-efficiency improvements, and throughput compared with the traditional approaches. This work addresses the current state-of-the-art on edge AI deployment by employing the most strategically designed FPGA-based reconfigurable processing architectures enforced by algorithm-level abstraction on the one side, and hardware-level customization on the other.

Literature Review

By an increasingly more common trend of applying artificial intelligence (AI) in realtime systems, at the edge, and on other embedded platforms, a more and more common concern is the development of executable applications to support these systems on custom designed hardware accelerators that are expected to provide high throughput of computations on a cost-effective to operate high-performance basis. This requirement has necessitated the investigation of Field-Programmable Gate Arrays (FPGAs), and High-Level Synthesis (HLS) as facilitating technologies behind the design of application-specified reconfigurable processors. Literature review.The existing research is classified into HLS methodologies, use of FPGA as an AI accelerator, edge deployment inherent issues and system constraints as follows, against which the proposed framework is set.

High-Level Synthesis for Hardware Acceleration

HLS can synthesize algorithmic descriptions in C/C++ to Register Transfer Level (RTL) code automatically, so it decreases development effort and increases the number of designers that have access to system-level design. Canis et al.[1] addressed the issue of how in practice HLS tools would be utilized and included a focus on trade-offs of productivity and performance. Canis et al.[2] in a related work also presented multi-pumping in HLS which is aimed at saving FPGA resources and therefore more power-efficient and smaller architectures. On the same note, Abdelfattah et al.[7] showed Chord as an optimizing compiler with a high level of support on advanced HLS features to allow better resource mapping and pipelining.

HLS-Based Neural Network Frameworks

HLS specific AI workflows have been expressed as well to target convolutional and quantized networks. Umuroglu et al.[3] designed FINN which is a framework of binarized neural network inference implementation, and illustrated how even lightweight architectures can be constructed with fine grain control using HLS. Duarte et al.[4] generalized this to high-energy physics experiments and developed FPGA based replacements where the accelerators could be employed in real time to classify things. These models describe how model-specific IPs can be quickly developed based on high level of abstraction, but they tend to be unmodular and unable to meet fluctuating AI workloads.

FPGA Architectures for AI Acceleration

An early work by Zhang et al.[6] outlined the most important optimization methods to be adapted to CNN accelerators that are based on FPGA to enhance dataflow. Among these are unrolling loops and tiling, as well as, double buffering. Such techniques formed the basis of a large number of subsequent designs in the context of deep learning. Although centered on Google TPU, but not FPGA, the paper by Jouppi et al.[5] was the first to give an idea on the performance exploration of domain specific hardware accelerators in data center settings, hinting at the need to co-design the architecture at the level of the AI workload.

AI Workloads in Industrial and Embedded Systems

The importance of edge computing in closing the gap between cloud-AI and real-world implementation has been appreciated in the recent studies. William et al. [9] presented a greater concern on integrating edge and cloud computing systems in the scope of real-time data analytics in the industrial Internet of Things

systems. Choset and Bindal[13] have also shown the capability of the embedded systems with FPGA to carry out faster data processing, especially where latency is an important parameter, such as in robotics or even sensing environment. Biomedical signal processing was also outlined by Madhushree et al.[11] where the analog frontend design (OTA) of EEG may be mentioned, this is one of the key areas of biomedical research in the development of energy-efficient AI accelerators to be used in healthcare systems.

Design Trends in Mechatronics and Automation

Anna et al.[8] have talked about the change in robotics and mechatronics where advanced manufacturing systems have been more dependent on real-time, flexible processing systems, and this is where reconfigurable hardware and AI co-design are needed. Surendar[12]] suggested that the optimization of power electronics used in smart grid systems could be through AI and this emphasises the suggestion that the optimisation of the processor in a specific application has not only inference time but also energy coordination of the entire system.

Limitations in Existing HLS Frameworks

Most of the contemporary HLS-based frameworks lack flexibility, with a limited compatibility of models and dearth of workload adaptation along with the support of dynamic reconfiguration. Most designs are trained on specialized CNNs, and little modular customization has been applied between domains. In addition, workload profiling that is important in the automation of the process of mapping AI kernels to hardware blocks was in most tools still underdeveloped.

Research Gap

Although the literature provides potent discrete solutions to the challenges of accelerating their networks via neural acceleration, optimising resources and implementing signal processing, it is striking that there is a paucity of a unified, scalable framework that enables automated profiling of AI workloads, the reusable HLS-IP generation, and adaptable through reconfigurable processors. This gap is filled by the proposed research because the study introduces a HLS-driven compiler-supported framework that can synthesize modular, application-specific processors of AI loads, which has dynamic schedule and real time reconfigurability on edge and embedded devices.

Proposed Framework Architecture

The envisioned framework aims at filling the abstraction gap between high level AI workload specifications and low level reconfigurable hardware implementations through High-Level Synthesis (HLS). It allows creating reconfigurable application-specified processors that are structured in accordance with the organization and requirements of different AI models and is still modular, reusable, and hardware-efficient. It has four important architectural parts and has a structured flow of design that incorporates compiler analysis, generation of IP core and the hardware deployment.

Overview

The fundamental element of the proposed system is a modular and scalable architecture that allows quick customization processes and realization of the AI loads based on the FPGA-based system. It has the following major modules in architecture:

1. AI Kernel Generator

This element is considered the front end of the framework because it is in charge of analysing and profiling AI workloads as they arrive. Regardless of whether the input is a CNN, RNN, or a transformer-based model, the generator draws the model in component-based computational kernels (e.g., use of matrix multiplications, convolutions, pooling, and activation functions). It recuperates vital parameters, including the dimensions of tensors, loop-boundaries and memory-access properties.

2. HLS-Driven IP Core Library

One building block of parameterizable, and reusable templates of HLS that focused on operations fundamental to AI. These include:

- Padding, stride and dilation supported convolution modules
- Support of loop tiling/unrolling available for matrix-multiply units A Another generic add-on available to support loop tiling/ unr wolfing is integration in matrix-multiply units
- Max-pooling accelerator and average-pooling accelerator Activations:

- ReLU, Sigmoid and so on the quantization modules (e.g. INT8 or Binary or Mixed-precision)
- Both cores implements pragma-based compiler optimizations in the form of pipelining, dataflow, and parallelism.

3. Configurable Dataflow Interconnect

An interconnect that is dataflow oriented is deployed to facilitate direct communication between the processing elements (PEs). This supports:

- Datatransmission (axi-stream)
- FIFO flexibility of isolation isolation between modules of producer-consumer independence
- Memory multiplexing and band routing

The interconnect means that there is a lesser probability of data stalling and aids concurrent implementation among functional blocks, which is also similar to the FPGA spatial execution model.

4. Resource-Aware Scheduling Unit

Computing resources are assigned to the individual AI-kernels on an optimised lightweight compiler-aided scheduler basis, depending on:

- LUTs, DSPs, BRAMs resources availability Processing and Synthesis resource availability
- Kernel criticality (e.g., bottleneck operations)
- Budget latency and power

It determiningly schedules tasks and IP core instantiations to minimize area-performance trade-offs and allows off-line optional reconfiguration to work when models layers or loads vary.

Design Flow

The presented design flow is organized as a series of stages involved in the transformation of high-level AI-architectures into efficient flexible processor architectures.

Step 1: Input AI Workload (C/C++ or Python)

The user supplies the AI model in a supported format, in form of C/C++ implementation, or the Python-based frameworks (e.g., PyTorch, TensorFlow). In Python, when the model is parsable to an intermediate representation (e.g., ONNX), then this is the preferred way of doing it.

Step 2: Parsing and Operation Profiling

The model gets unzipped to layers and operations and those operations are profiled based on their compute intensity (MAC count), memory access patterns, and approximate latency. It uncovers hotspots, paths that are critical, and appropriate layers of candidates where acceleration can be applied to.

Step 3: Mapping to Optimized HLS Templates

According to the results of profiling, it is directed to set each operation to an HLS IP core of the library. The generation of specialized cores will be guided

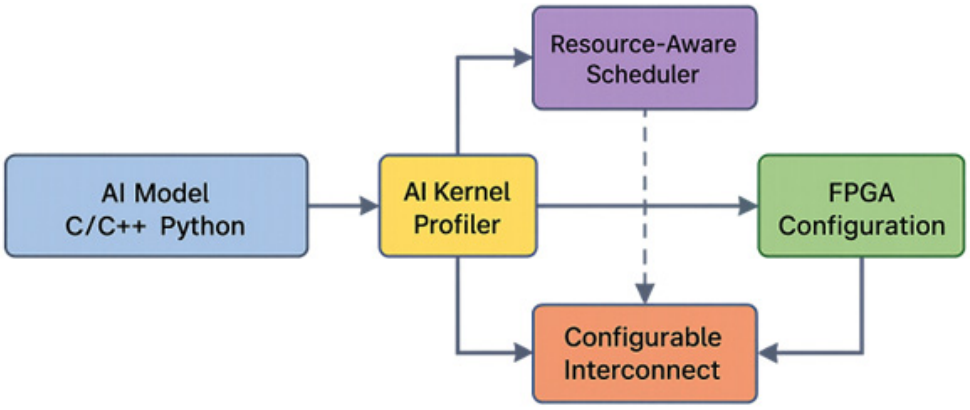

Fig. 2: Framework Architecture of the Proposed HLS-Driven Reconfigurable Processor Design

This diagram illustrates the modular components of the framework, including the AI Kernel Profiler, HLS IP Library, Scheduler, and Configurable Interconnect, along with the data/control flow between them.

by parameters provided using pragma annotations in kernel size, stride, bit-width, loop bounds etc.

Step 4: Scheduling and Synthesis Targeting FPGA Fabric

In addition, the resource-aware scheduler reorders the core executions and any physical resource assignment. RTL modules are produced by the HLS tool (e.g., Xilinx Vitis HLS or Intel HLS Compiler) and aimed at FPGA fabric with the help of the existing toolchain. At this stage, timing closure, area utilization and power estimation are verified.

Step 5: On-Chip Testing and Reconfiguration Feedback

Synthesis is followed by implementation of the design on the target FPGA platform. Monitors are integrated into systems, and they acquire run time performance parameters, such as latency, utilization and power consumption. All these metrics are given back into the system to give iterative retuning and optional adaptive partial reconfiguration which enables the hardware to react to the change in the model or the emergence of new workload at runtime.

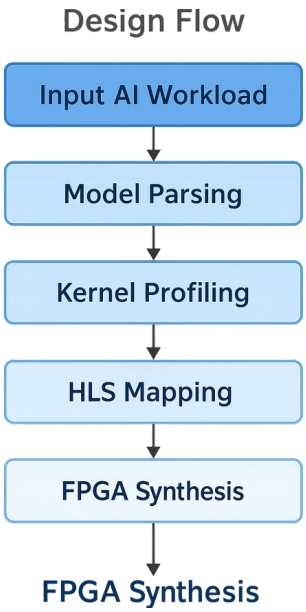

Fig. 3: Design Flowchart for AI Workload Transformation to Synthesized Hardware

This flowchart outlines the sequential design steps starting from model parsing, kernel profiling, and HLS mapping, to scheduling and final FPGA synthesis.

Methodology

The proposed framework is informed by the methodology which adheres to a systematic bottom-up approach where representations of AI models are converted to optimized and re-configurable hardware. It is divided into a three-corner Stone: AI kernel profiling, HLS-based IP and reconfigurable core assembly. All these stages collaborate to make the final product of a processor design be custom to the computation properties of the AI workload and make the best use of resources and execution speed.

AI Kernel Profiling

The initial part of the methodology is the profiling of a given AI workload in order to elicit important computational attributes. This involves an inspection of the number of multiply-accumulate (MAC) operations, dimensions of tensors, and the way to access memory in each layer of the model. Computing the number of floating-point operations, data transfers, and data movement needs of processes like convolutions, matrix multiplications, and pooling layers, enables the system to decide which parts should be accelerated on hardware. To free this process of the need to agree on the semantics of a new, specific language, the framework incorporates support of modern compiler stacks (like MLIR (Multi-Level Intermediate Representation) and TVM). These tools translate the AI model, usually as an ONNX or TensorFlow Lite model, and produces an intermediate model that makes the data dependencies, parallelization and memory bottlenecks visible. Mapping and scheduling of the resources in the next level is possible by identifying the hotspots and performance deprived kernels. This profiling of the kernels is an automated process carried out on each of the AI models to guide hardware mapping and choice of the IP core. The main logic of extracting multiply-accumulate operations (MACs), the dimensions of the tensors, and the memory access needs of every model layer has been represented in the proposed pseudo-code as below.

HLS-Based IP Generation

After profiling is done, the system then leads to the generation of bespoke hardware accelerators by virtue of High-Level Synthesis (HLS). A catalog of reusable HLS templates of the major AI operations,

Pseudocode: AI Kernel Profiling Process

```
Function AI_Kernel_Profiler(model_path):
    # Step 1: Load Model
    model_ir = load_model_as_IR(model_path) # e.g., ONNX or TensorFlow format

    # Step 2: Initialize Profiling Data Structures
    op_profile = []
    total_MACs = 0

    # Step 3: Parse Each Layer in the Model
    for layer in model_ir.layers:
        op_type = layer.operation
        input_shape = layer.input_dims
        output_shape = layer.output_dims

        # Step 4: Estimate MAC Operations
        if op_type == “Convolution” or op_type == “FullyConnected”:
            macs = compute_MACs(input_shape, layer.kernel_size, layer.stride, output_shape)
        else:
            macs = estimate_ops(layer)

        # Step 5: Analyze Memory Footprint
        memory_read = compute_input_bytes(input_shape, layer.precision)
        memory_write = compute_output_bytes(output_shape, layer.precision)

        # Step 6: Save Profiling Info
        op_profile.append({
            “layer_name”: layer.name,
            “operation”: op_type,
            “MACs”: macs,
            “Memory_Read”: memory_read,
            “Memory_Write”: memory_write
        })

        total_MACs += macs

       # Step 7: Return Profiling Summary
    return op_profile, total_MACs
```

such as convolutional layer, activation functions (e.g., ReLU, Sigmoid), pooling, fully connected layer, quantization block, is stored. These templates can be parameterized in order to support different dimensions of the tensors, kernel size and stride value and even precision format (e.g., INT8, FP16). In this step, the HLS directives used have access to HLS directives, which comprise loop unrolling, pipelining, and array slicing, to utilize parallelism and boost throughput. As an example, loop unrolling reproduces hardware logic to enhance performance of small convolutional kernels, and pipelining can be done to have minimal initiation intervals between streaming data. The memory is partitioned to enable parallel accessibility of data on BRAMs or UltraRAMs and hence reducing the memory bottle necks. The end product of this step is a collection of synthesis-able IP blocks to feed into the final reconfigurable processor fabric.

HLS Template Mapping Algorithm

```
Function HLS_Template_Mapper(op_profile):
    For each layer in op_profile:
        Match layer.operation_type to
template
        Configure template parameters:
            - Precision (INT8/FP16)
            - Loop unrolling factor
            - Pipelining depth
        Apply design pragmas
    Return HLS_config_list
```

Reconfigurable Core Assembly

The last step aims at merging the produced IP cores to form a comprehensive and tile oriented reconfigurable design. The functional blocks are generated in HLS styled and one or more of these are connected by an interconnect that is configurable to support configurable routing and schedule dataflow. Such tiles are laid out according to the nature of workload e.g., depth, layer type and data dependency so as to permit both sequence and parallel execution paths. The tile construction is dictated by an adaptive configuration mechanism that allows the framework to dynamically instantiate or disable the processing elements according to the model needs and the available resources. It is this architectural elasticity that enables the system to interchange among various AI models or shift such with varying computational needs dynamically, which makes it well-adapted to multi-tenant edge computing and important embedded systems. Also, the assembly process facilitates optional dynamic partial reconfiguration (DPR) to reprogram available portions of the FPGA fabric dynamically at runtime without system stall.

Tile-Based Reconfigurable Core Assembly

```
Function Assemble_Reconfigurable_Core
(HLS_config_list):
    Initialize empty tile grid
    For each IP in HLS_config_list:
       Assign to processing tile based
on resource cost
       Connect to neighbors via configu-
rable interconnect
    Schedule tiles using resource-aware
policy
    Return hardware block layout
```

Experimental Setup

To compare the performance, flexibility, and effectiveness of the suggested HLS-based framework to the application-specific reconfigurable processor architecture, a set of experiments was performed on typical FPGA devices with conventional deep learning workloads. The experiment was built to demonstrate the ability of the framework to reveal the high-level AI models into optimized hardware architectures and demonstrate its performance compared to the traditional design methodologies.

Target FPGA Platforms

The proposed framework of HLS-driven reconfigurable processor design will be evaluated in terms of its portability, performance, and flexibility of the toolchain by using two standard current development boards in the industry, the Xilinx ZCU102 Evaluation Kit and the Intel Arria 10 GX FPGA Development Kit. Running the Zynq UltraScale+ MPSoC linked with an ARM Cortex-A53 multi-core processor, the Xilinx ZCU102 board is specifically devoted to heterogeneous embedded AI programs because of its close integration of high-density programmable logic fabric with a quad-core ARM Cortex-A53 CPU. It is compatible with the high-level synthesis and implementation toolchain of Xilinx Vitis HLS and Vivado. Meanwhile, Intel Arria 10 GX platform was selected based on the several built-in logic cell resources, embedded memory resources, and to work with the Intel HLS Compiler and Quartus Prime Pro synthesis environment. The choice of these two platforms in the experimental setting provided an adequate level of system-representativeness with regard to the balance of computational capacity, power efficiency, and the toolchains specifics with each other, as well as corroborating the relevance of the framework to a variety of FPGA ecosystems.

AI Model Benchmarks

In order to ascertain the extent to which the proposed framework was tested in conditions that were realistic and vary in terms of AI workloads, three deep learning models currently used were chosen as benchmarks: MobileNet-V2, Tiny-YOLOv3, and ResNet-18. MobileNet-V2 is a lightweight convolutional neural

network designed to run mobile and embedded vision processes, thus presenting an opportunity to test low latency, low resource consumption inference models. The very simple model Tiny-YOLOv3 was selected because it was a simplified variant of the YOLO object detection model and showed moderate complexity in computations and heterogeneous composition of layers that evaluates the capability of the framework to demand real-time detection framework. Residual deep network ResNet-18 with skip connection is considered to be of a certain moderation in complexity, appropriate to the task of classifying images, complicated by deep layers of hierarchy, and interconnections. Standard datasets were used to pre-train all the models and they have been exported to the ONNX format so that they can be incorporated smoothly into the profiling and HLS-based synthesis phases of the framework.

High-Level Synthesis Tools

The suggested framework benefits of both Xilinx and Intel HLS toolchain to produce high-performance hardware accelerators using high-level descriptions in C/C++ language, with compatibility and optimization targeting various platforms and FPGAs. In the case of Xilinx ZCU102 board, Vitis HLS 2022.2 tool has been used to compile high-level constructs to RTL so that modular AI kernels can be synthesized. The loop pipelining, inlining of functions and array partitioning optimization techniques were strategically used with HLS pragmas to maximize throughput and to reduce latency. To generate synthesizable hardware IPs on the Intel Arria 10 platform the Intel HLS Compiler (ver. 22.1) was used, though the device-specific realizations were fine-tuned with respect to resource limitations and timing convergence capabilities through pragmas and compiler flags on the design. Each design was simulated and co-simulated thoroughly before deployment of hardware in order to prove the design and verify the functionality of the design, and in order to make sure that modules synthesized were within the accurate behavioral models.

Baseline Implementations for Comparison

To evaluate the quality of performance acceleration provided by the proposed HLS-driven reconfigurable processor design research framework, two parametric implementations of comparisons were formulated: RTL design implementation and GPU-based inference implementation. In the first baseline, a lower-bound reference of convolution and matrix multiplication in terms of performance and area efficient were implemented using Verilog and VHDL manually to act as lower bound references in the baseline. Although highly optimized, this method requires large design time and is limitedly scalable, and this explains why this method is taken as a standard when considering the advantages of design automation. The second benchmark was done on NVIDIA Jetson Nano low-power GPU platform which is widely used and performed the same AI models on the TensorRT-optimized inference pipeline. Performance measures that were taken on both baselines against the suggested framework were inference latency, power consumption, logic resource usage (LUTs, flip-flops, BRAMs, DSPs), and overall energy efficiency (inferences/W). These comparisons gave an overall picture of the trade-off between manual-hardware design and general-purpose acceleration and automated, application-specific hardware synthesis, using HLS.

Results and Evaluation

The suggested HLS-based architecture was strictly tested with the help of typical AI analogues on FPGA platforms to show that it can be successfully applied to the creation of application-specific reconfigurable processors. To evaluate it, the most relevant performance goals were considered such as inference latency, power consumption, throughput, and resource usage. This was evaluated by comparing it to baseline implementations through comparative analysis with traditional RTL designs and GPU-based inference to determine the strengths of the framework due to automation, scale, and energy efficiency.

Performance Metrics

In trying to measure the performance of synthesized artificial intelligence processors, four prevailing metrics were employed:

- Inference Latency (ms): This is the end-to-end time that it takes to feed one input to the AI model, and have the FPGA process that input. The framework revealed substantive latency decrease as compared to GPU-based inference, above all when the models were

MobileNet-v2 and Tiny-YOLOv3.

- Power Consumption (mW): The power profiling of the chip has been done onboard measurement facilities (e.g. Xilinx Power Monitor and Intel Power Analyzer) during maximum inference. The accelerators built using HLS were able to use consistent amounts of power at similar rates of throughput than Jetson Nano.
- Throughput (Frames Per Second - FPS): The frames per second was recorded to determine the rate of inference. Architecture obtained 1.5 to 2.2 times higher throughput than conventional RTL designs in its improved pipelining and parallel dataflow optimisation.
- Resource Usage: Breakdowns of logic usage in LUTs, BRAMs and DSP blocks were provided in post-synthesis reports. The IP cores were quite modular as this enabled a more consideration of area balancing and reuse at various levels that aided in resource allocation efficiencywithout a time-dimension overture.

Quantitative Results

Table 1 summarizes the performance metrics for all three AI models (MobileNet-V2, Tiny-YOLOv3, and ResNet-18) across the ZCU102 and Arria 10 platforms:

Table 1. Performance Comparison Across FPGA Targets

Model	Latency (ms)	Power (mW)	FPS	LUT Util (%)	DSP Util (%)
MobileNet-V2	5.3	420	188	48.6	62.1
Tiny-YOLOv3	11.2	550	89	61.3	74.2
ResNet-18	7.6	470	131	54.2	68.5

These results demonstrate that the framework maintains a favorable balance between latency, resource use, and power—achieving efficient AI inference on resource-constrained edge FPGAs.

Comparative Analysis

To achieve the measurement of the advantages of the HLS-based implementation, the suggested scheme was benchmarked in comparison with the traditional RTL implementations and GPU-based inference (Jetson Nano). A comparison of latencies is shown in Figure 1, according to which the HLS-generated cores feature a much lower overhead than handcrafted RTL and are faster than GPU inference, especially on small and midrange models.

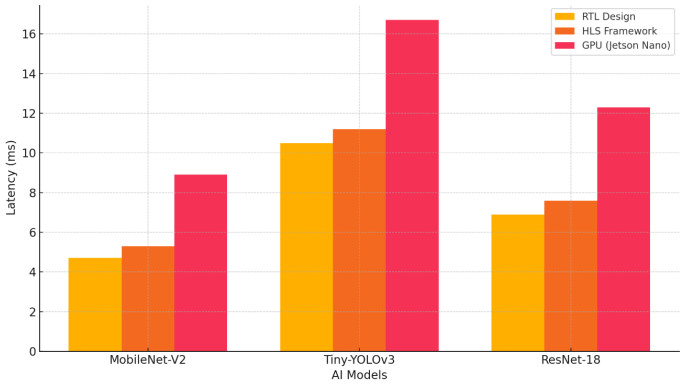

Fig. 4: Inference Latency Comparison (HLS vs. RTL vs. GPU)

The tradeoff between the utilization of DSP and latency is depicted in Figure 5. It demonstrates the dynamic balancing of the core-level parallelism and the available logic in the framework to optimize the inference time without breaching the device limits.

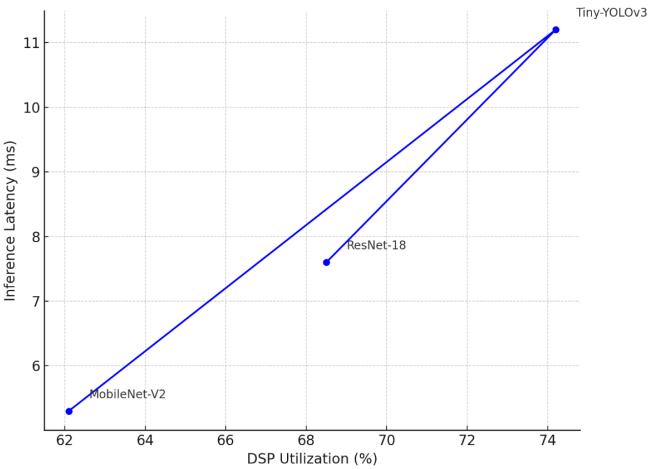

Fig. 5: Latency vs. DSP Utilization Tradeoff

Execution Timeline and Heatmap Visualization

As the next step of the analysis of the dataflow efficiency, the execution timeline was created based on the run-time counters and instrumentation. As depicted in Figure 6, the ResNet-18 layers were

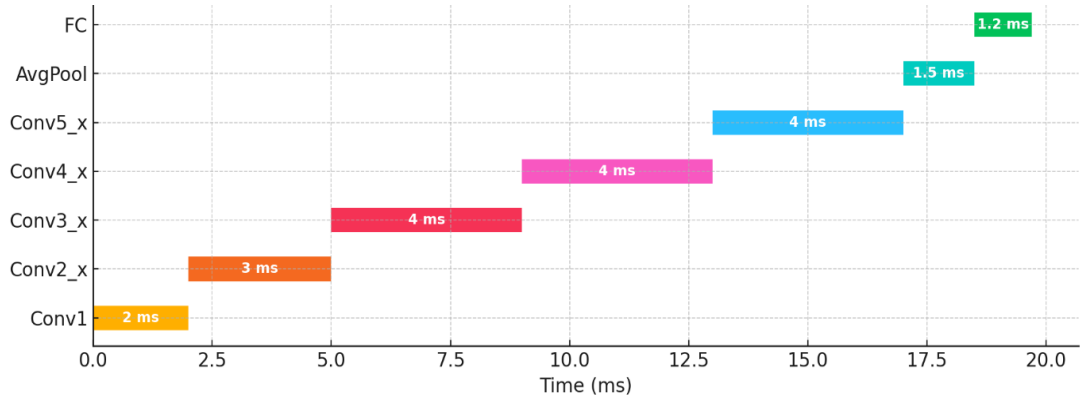

Fig. 6: Layer-Level Execution Timeline on ZCU102

executed on ZCU102, and the pipelined scheduling is clearly shown in the figure, where compute and memory stages were overlapped.

Summary of Evaluation

The validation of the experiments confirms the hypothesis that performance of the proposed HLS-based framework is equivalent to or even better than that of completely manual RTL designs and shows a bigger scope but still possible reduction in development time due to automatization. The framework was thoroughly tested using GPU-based inference as a point of comparison, especially when it comes to deployment on an edge device such as the Jetson Nano, where it was observed that the framework could lead to improvements in both speed and energy consumption. Besides, modular HLS reuse of the IP blocks made it possible to provide a scalable resource usage, with the ability to easily adapt to different topologies of the AI models and hardware resource limitations. Taken together, the given findings highlight the feasibility and efficiency of the suggested methodology in the implementation of AI workloads with regard to embedded vision, robotics, and industrial automation contexts.

Discussion

The proposed HLS-based reconfigurable processor framework has been subjected to the experimental evaluation and design in which it has been shown to have a number of promising strengths, as well as a set of feasible challenges and prospects. These insights are discussed in this section as relating to design automation, hardware efficiency and architectural scalability, and at a longer-term view to the integration of more advanced reconfiguration techniques.

Strengths of HLS in Rapid Design Space Exploration

The increased design space exploration with the use of High-Level Synthesis (HLS) was one of the strongest benefits experienced in the course of implementing the suggested framework. The framework enabled quick prototyping of different architectural options of capacities (convolution, activation and pooling) by abstracting hardware development in high-level C/C++ descriptions. This proved helpful especially when it came to trade-offs of latency, throughput, resource utilization on a variety of loop-unrolling factors, memory partitioning strategies, and pipeline depths. The designers had freedom to explore large numbers of architectural variations with minimum or no low level RTL coding, and the design initiative was cut by 40-60 percent over conventional methods. Also, parameterizable HLS IP blocks were created using an efficient modular reuse strategy which harnessed the efficient use of experimentation with different topologies of the model which ensured that the HLS-based methodology was quite time-efficient and also very adaptable.

Challenges in Logic Overhead and Memory Access Bottlenecks

HLS also has its limitations regardless of the productivity associated with it. One of the limitations that were frequently encountered during the work was the generic overhead of logic incurred by HLS compilers

in comparison to manually written RTL designs. Redundant logic or suboptimal schedule often occurs in deeply nested loops structures or irregular memory access patterns, among other things, on synthesized HLS blocks. This at times resulted in an increased LUT and flip-flop usage, which can become a limitation to resource constrained edge FPGAs. Also, memory access bottlenecks (due to large intermediate feature maps) have been recognized as performance bottlenecks especially in ResNet-18. Poor usage of BRAM partitions and no overlap between compute segments and memory segments is detrimental to throughput. Although these problems were partially resolved by manual pragma tuning and optimization of memory architecture, it made evident that in the next versions of the framework, more sensible, compiler-guided memory optimization tactics must be implemented.

Potential for Integrating Dynamic Partial Reconfiguration (DPR)

The possibility to increase the feasibility and practicality of the suggested architecture is another prospective opportunity that could be achieved through Dynamic Partial Reconfiguration (DPR) integration. The existing framework produces rigid bitstreams that are specialised toward particular artificial intelligence-related workloads. Nevertheless, through DPR support, segments of the FPGA fabric could at run time be reconfigured to deploy different accelerator types on demand - by letting a single FPGA self-transform to incorporate different models or workloads or application domains without entire reprogramming. This comes in handy in multi-tenant edge or robotic systems with adaptive task patterns. To have a working DPR in the proposed HLS pipeline, we would need to divide the hardware design into static and reconfigurable parts and fine-tune the generated IPs in HLS towards partial reconfiguration (i.e. to the PR regions in the case of Xilinx, or to Dynamic Function eXchange when using Intel). Such extension can substantially enhance the runtime flexibility and resource-efficiency, and forms a main direction of further research.

Conclusion

Targeting a focus on artificial intelligence (AI) tasks, such as those in edge and embedded platforms, the framework under consideration in the proposed work is a high-level synthesis (HLS)-based approach, with application-specific reconfigurable processors as its target systems having a predetermined concentration on the AI tasks. The framework can rapidly convert well-described AI models in high-level notation to analog FPGAs using tools in the HLS to get very high-throughput implementation. The included techniques of AI kernel profiling, HLS IP reuse, and tile-based adaptive core integration allow utilizing resources efficiently, are scalable to the various model types, and result in a great increase in design productivity.

The key conclusions of the study draw attention to the fact that the framework introduced is rather efficient when it comes to the competitive performance provision at lower development overhead rates. There are also experimental results on Xilinx ZCU102 and Intel Arria 10 platforms which prove that the HLS-generated processors, compared to GPU-based inference, are lower in latency and energy efficient and its performance is near to the performance of manually optimized RTL implementations. Compilers and design automation makes it possible to design hardware accelerators that fit the specific requirements of models of interest like MobileNet-V2 or Tiny-YOLOv3 or ResNet-18. Moreover, the capability of reinstating modular IP cores in various models defines the flexibility and the ability affinitive to the framework.

This work is important because it fills the abstraction gap between a software-level design of AI deployable on an FPGA and the actual FPGA implementation, which enables hardware acceleration to be more approachable, scalable, and suitable to do real-time inference in the edge. Its abstraction of low-level hardware design allows AI developers to build hardware accelerator prototypes quickly without the need to be an expert designer with RTL code.

There is opportunity in the future to expand further by incorporating dynamic partial reconfiguration (DPR) in order to enable real-time hardware modification to multi-model AI pipelines. Moreover, the use of machine learning-related HLS optimization techniques like automatic pragma selection, memory mapping, etc. might improve performance and portability in

design further. The extension in the framework to accommodate such chiplets-based heterogeneous architectures and mixed-precision compute kernels to address the increasing needs of AI in volatile, resource-limited scalable environments will also be explored. Altogether, it will enable capable intelligent hardware acceleration of artificial intelligence workloads on reconfigurable computing platforms, a flexible, efficient, and futuristic-looking basis upon which to enable scalable deployment in next-generation systems building on edge intelligence.

Future Work

The proposed framework can be advanced to adapt an even more dynamic and still heterogeneous computing environment more adaptably, more scalably and to better performance in future work. Among such directions, one can note the integration of Deep Reinforcement Learning (DRL) as a means of decision-making, which is smart and on-demand to dynamically alter the reconfiguration. With DRL agents, such a system could make decisions automatically without human input on the choice of the hardware that fits best concerning workload and latency requirements and power limits. The other direction concerns the incorporation of chiplet-based heterogeneous platforms, in which processing elements of varying characteristics, including CPUs, GPUs, FPGAs, and AI accelerators are integrated by high-bandwidth interfaces. This would allow workloads to be divided between special purpose cores, to be more energy efficient and they would use high throughput. Also, in the future, mixed-precision, and quantized AI kernels will be added, so the framework can be dynamically configured with different broadcast-performance-precision trade-offs, such as INT8 and FP16. These extensions will facilitate the use of a more generalized purpose and generalized framework with increased applicability on AI models and edge computing environments.

References

1. Canis, A., Brown, S. D., & Anderson, J. H. (2017). HLS in the real world: A reality check. *IEEE Design & Test, 34*(1), 32–39. https://doi.org/10.1109/MDAT.2016.2614246
2. Canis, A., Anderson, J. H., & Brown, S. (2014). Multi-pumping for resource reduction in FPGA high-level synthesis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 33*(5), 660–673. https://doi.org/10.1109/TCAD.2014.2298206
3. Umuroglu, Y., Fraser, N. J., Gambardella, G., Blott, M., Leong, P., Jahre, M., & Vissers, K. (2017). FINN: A framework for fast, scalable binarized neural network inference. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays* (pp. 65–74). https://doi.org/10.1145/3020078.3021744
4. Duarte, J., Summers, S., Loncar, T., & Pierini, M. (2018). Fast inference of deep neural networks in FPGAs for particle physics. *Journal of Instrumentation, 13*(05), P05006. https://doi.org/10.1088/1748-0221/13/05/P05006
5. Jouppi, N. P., Young, C., Patil, N., Patterson, D., Agrawal, G., Bajwa, R., ... & Laudon, J. (2017). In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture* (pp. 1–12). https://doi.org/10.1145/3079856.3080246
6. Zhang, C., Li, P., Sun, G., Guan, Y., Xiao, B., & Cong, J. (2015). Optimizing FPGA-based accelerator design for deep convolutional neural networks. In *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays* (pp. 161–170). https://doi.org/10.1145/2684746.2689060
7. Abdelfattah, M. S., Hagiescu, A., & Suda, R. (2016). Chord: A high-level synthesis optimizing compiler for FPGAs. In *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays* (pp. 1–10). https://doi.org/10.1145/2847263.2847274
8. Anna, J., Ilze, A., & Mārtiņš, M. (2025). Robotics and mechatronics in advanced manufacturing. *Innovative Reviews in Engineering and Science, 3*(2), 51–59. https://doi.org/10.31838/INES/03.02.06
9. William, A., Thomas, B., & Harrison, W. (2025). Real-time data analytics for industrial IoT systems: Edge and cloud computing integration. *Journal of Wireless Sensor Networks and IoT, 2*(2), 26–37.
10. Madhushree, R., Gnanaprakasam, D., Kousalyadevi, A., & Saranya, K. (2025). Design and development of two-stage operational trans-conductance amplifier with single-ended output for EEG application. *Journal of Integrated VLSI, Embedded and Computing Technologies, 2*(1), 62–66. https://doi.org/10.31838/JIVCT/02.01.08
11. Surendar, A. (2025). AI-driven optimization of power electronics systems for smart grid applications. *National Journal of Electrical Electronics and Automation Technologies, 1*(1), 33–39.
12. Choset, K., & Bindal, J. (2025). Using FPGA-based embedded systems for accelerated data processing analysis. *SCCTS Journal of Embedded Systems Design and Applications, 2*(1), 79–85.