AI Compilers Meet FPGAs: A HW/SW Co-Design Approach for Vision Transformers

AMD Open Hardware Competition 2025 - Adaptive Computing Track

Team Number:
AOHW25_216

Supervisor:
M.Sc. Johannes Geier

Team Members:
Agustin Nahuel Coppari Hollmann
Ipek Akdeniz
Michael Lobis
Osman Yaşar

Date:
August 31, 2025

AMD Open Hardware Competition
Project Report Team:
AOHW25_216

1. Abstract

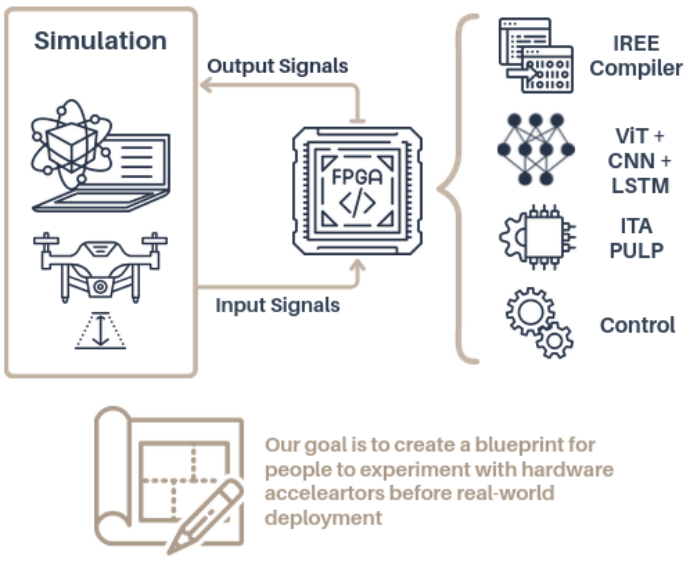
Real-time machine learning inference is critical for autonomous drone control, particularly in high-speed applications like drone racing. This project addresses the high latency of traditional CPU-based systems by developing a custom hardware-accelerated inference pipeline on an FPGA platform. We present an end-to-end HW-SW co-design framework for FPGA deployment of hardware accelerators for autonomous systems using ML compilers, tested in a drone simulation environment. Our work consists of two key contributions. First, we selected and adapted a Vision Transformer (ViT) model, quantizing it for 8-bit integer execution specifically for the drone navigation task. Second, we deployed this model on the AMD Kria KR260 Robotics Starter Kit by porting and integrating the open-source ITA (Integer Transformer Accelerator)[2, 3] from the PULP Platform. Our primary hardware contribution lies in the significant effort of adapting this specialized ASIC-centric architecture for the FPGA fabric, which involved designing a memory subsystem and building a custom wrapper with a state-driven execution engine. Moreover, we have also worked with IREE, an end-to-end compiler framework that lowers Machine Learning models down to a unified Intermediate Representation. Using IREE, we have lowered our Vision Transformer model down to low level hardware accelerator calls and CPU instructions.

---

2. Introduction

The operational agility and safety of autonomous drones are directly limited by their decision-making speed. In complex, high-speed flight scenarios, the inference latency of machine learning models running on general-purpose CPUs becomes a critical bottleneck, preventing the drone from reacting in time. This project tackles this challenge through targeted hardware acceleration on an FPGA.

Our solution is a complete hardware/software co-design that addresses a common gap in the open-source hardware community. While many powerful accelerators are published individually, they often exist in isolation, making it difficult for others to utilize in a real-world system. Our work bridges this gap by taking a state-of-the-art open-source component and redesigning a complete, practical framework that can be adapted to FPGA platforms.



On the hardware side, instead of designing an accelerator from scratch, we undertook the task of porting the Integer Transformer Accelerator (ITA) [2], an open-source project from the PULP Platform, from its original ASIC target to the AMD Kria KR260's FPGA fabric. This adaptation was non-trivial. The original ASIC design assumes a monolithic, tightly-coupled memory (TCDM) with 32 ports and utilizes clock-gating techniques unsuitable for FPGAs. Our effort, therefore, involved architectural modifications, including creating a new memory controller to map the TCDM interface to the Kria's banked URAMs and building the entire AXI-based control and data-streaming infrastructure.

Complementing the hardware adaptation was a parallel effort on the machine learning model itself. A generic hardware accelerator like the ITA imposes specific constraints on model architecture (e.g., layer dimensions, data path widths). Therefore, a standard, off-the-shelf ViT model could not be used directly. Our work included selecting a suitable base ViT architecture, modifying its structure to be compatible with the ITA's hardware constraints, and subsequently performing 8-bit quantization and training for the drone navigation task.

By developing both the adapted model and the hardware port, we created a complete and tightly integrated solution that demonstrates a practical and replicable framework for integrating powerful open-source hardware into real-time autonomous control systems, showcasing how to transform standalone designs into fully-realized, useful applications.

Complementing the hardware adaptation was a parallel effort on the machine learning model itself. A generic hardware accelerator like the ITA imposes specific constraints on model architecture (e.g., layer dimensions, data path widths). Therefore, a standard, off-the-shelf ViT model could not be used directly. Our work included selecting a suitable base ViT architecture, modifying its structure to be compatible with the ITA's hardware constraints, and subsequently performing 8-bit quantization and training for the drone navigation task. By developing both the adapted model and the hardware port, we created a complete and tightly integrated solution, demonstrating a practical and replicable framework for real-time autonomous control.

To bridge the gap between the adapted model and our custom hardware, we developed a complete software toolchain centered on the IREE (Intermediate Representation Execution Environment) compiler. This involved creating a custom hardware dispatch layer that allows the high-level IREE runtime to target our FPGA accelerator. This compiler-based approach automates the process of offloading computations, managing memory, and controlling the hardware, transforming the standalone components into a cohesive and programmable acceleration framework.

By developing the adapted model, the hardware port, and the compiler toolchain, we created a complete and tightly integrated solution. This work demonstrates a practical and replicable framework for integrating powerful open-source hardware into real-time autonomous control systems, showcasing how to transform standalone designs into fully-realized, useful applications.

3. Background and Related Work

Our project's hardware acceleration strategy is grounded in the architectural principles of modern neural networks and the practical necessities of edge computing. The core concepts include the Vision Transformer (ViT) architecture, its computationally-intensive attention layers, and the critical role of 8-bit quantization for deployment on FPGAs.

3.1. Vision Transformers (ViT)

Vision Transformers have revolutionized the field of computer vision by applying the highly successful Transformer architecture, originally designed for natural language processing, to image-based tasks. Instead of using the sliding convolutional filters of a CNN, a ViT begins by deconstructing an image into a sequence of fixed-size patches. These patches are flattened into vectors, linearly projected, and augmented with positional embeddings to retain spatial information. This sequence of "image words" is then processed by the Transformer encoder.

The key advantage of this approach for an autonomous drone is its ability to learn global relationships across the entire image from the very first layer. A CNN builds up its understanding hierarchically from local features, whereas a ViT can immediately assess the relationship between a patch on the far left of the scene and one on the far right. This holistic understanding allows the drone to make more informed decisions based on the full context of its environment during complex navigation tasks.

The model generates three vectors: a Query **(Q)**, a Key **(K)**, and a Value **(V)**. The Query is essentially the patch's request for context. It compares itself to the Key of every other patch to generate attention scores, which is a direct measure of inter-patch relevance. These scores then dictate how to blend all the Value vectors into a new representation.

3.3. The Importance of 8-Bit Quantization

While powerful, the computational and memory demands of ViT models make them difficult to deploy on resource-constrained and power-sensitive edge devices like a drone's flight controller. This is where 8-bit quantization becomes a critical enabling feature. Quantization is the process of converting a model's parameters (weights and biases) and intermediate calculations (activations) from high-precision 32-bit floating-point numbers to low-precision 8-bit integers. This is essential for high-performance FPGA deployment for three main reasons:

1. **Reduced Memory Footprint & Bandwidth:** INT8 data types require four times less memory than FP32. This drastically reduces the required on-chip memory, avoids the memory wall bottleneck by requiring less data to be moved, and allows for larger models to fit on-chip.
2. **Improved Computational Efficiency:** Integer arithmetic is far simpler and faster than floating-point math. On an FPGA, this means that integer operations can be implemented using fewer logic resources and can be efficiently mapped to specialized hardware blocks, enabling higher parallelism and throughput.
3. **Lower Power Consumption:** The combination of reduced data movement from memory and the use of simpler computational logic directly translates into a significant reduction

in dynamic power consumption, which is a crucial requirement for any battery-operated mobile platforms.

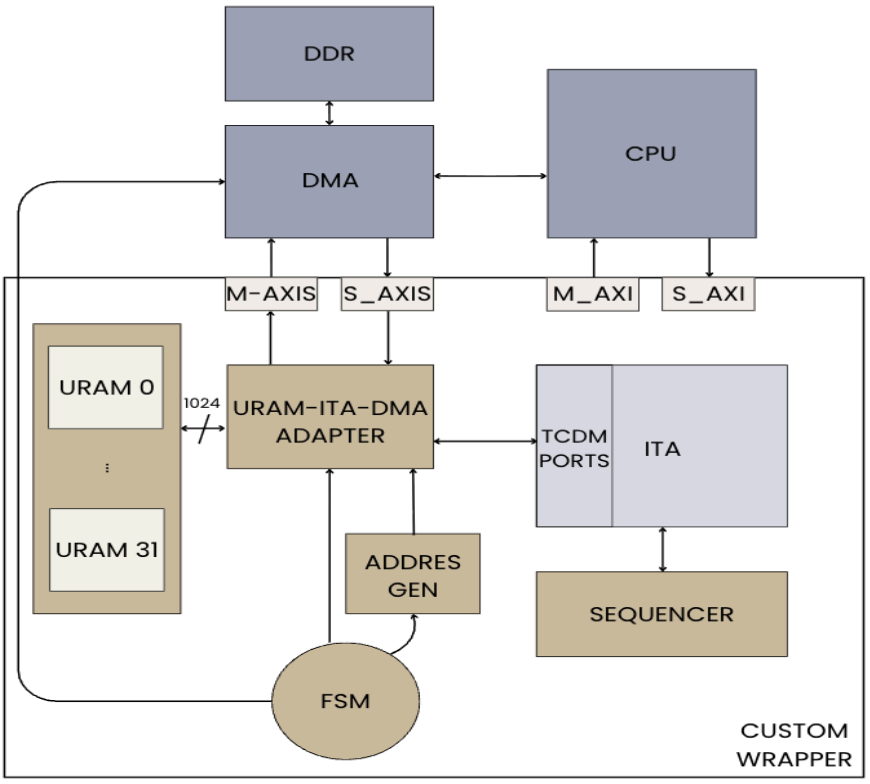
Our starting point, ITA is an open-source accelerator from the PULP Platform that was designed for this purpose. As presented in their work, ITA is a novel architecture that targets efficient inference on embedded systems by leveraging 8-bit quantization. Its core innovation is a streaming softmax implementation that operates exclusively on integer values, avoiding costly floating-point units and minimizing data movement. The original design was targeted for a 22nm ASIC process, where it achieved an energy efficiency of 16.9 TOPS/W. Our work builds directly upon this powerful open-source foundation by undertaking the significant engineering effort of porting and adapting this ASIC-centric design for deployment on an FPGA platform.

The project was developed and deployed on the AMD Kria KR260 Robotics Starter Kit. The System-on-Module (SOM) architecture provides a combination of a powerful processing system (PS) and programmable logic (PL). Its robotics-focused design and I/O capabilities allowed for straightforward integration with our ROS-based software environment.

---

4. System Design and Methodology

Our system integrates a software-based drone simulation with a hardware-accelerated ML inference pipeline. The centerpiece of the hardware acceleration is our custom wrapper around the ITA accelerator.



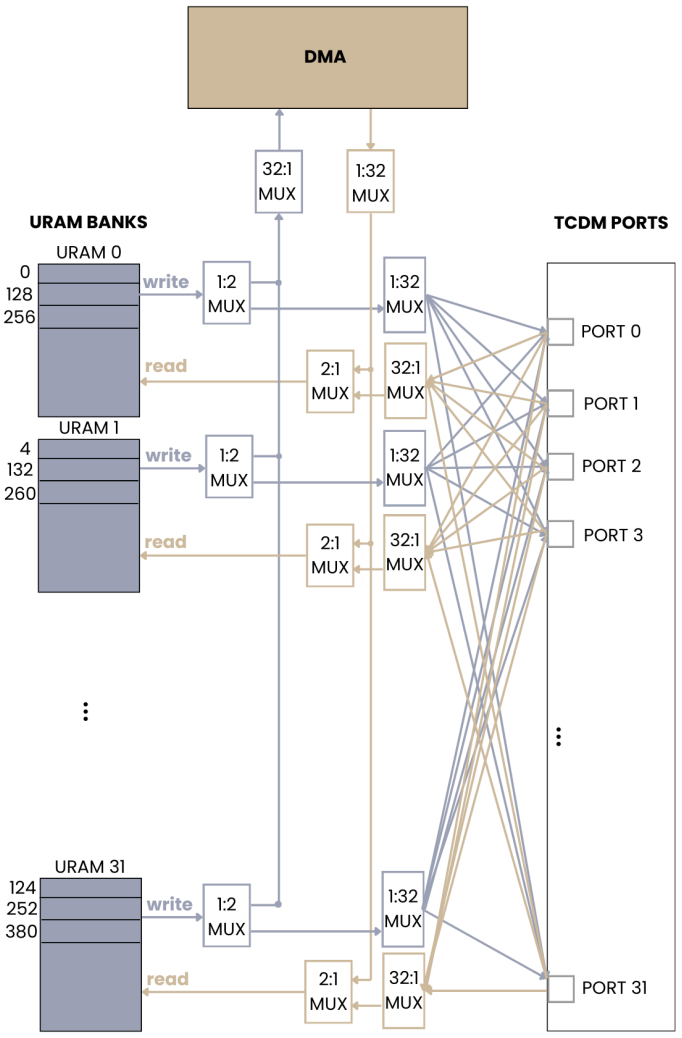
4.1. ITA Core:

This is the primary compute engine of our framework. As described in the source project, the ITA is specifically designed to accelerate the attention layers of the transformer using 8-bit integer arithmetic. Its most notable feature is an innovative softmax implementation which operates entirely on integer values. This streaming approach calculates the necessary numerator and denominator for the softmax function and processes them using a lightweight serial divider, thereby avoiding the need for resource-intensive floating-point units or large lookup tables.

While we leverage the powerful core, porting it from its original ASIC target to our FPGA platform required architectural modifications. The source design was heavily optimized for ASIC implementation, utilizing fine-grained clock gating to reduce power consumption and latch-based memory elements for timing optimization. These techniques are not directly synthesizable or efficient on an FPGA. The first part of our work, therefore, was to refactor this logic, replacing all latches with synthesizable flip-flops and removing the clock-gating infrastructure to create a robust, FPGA-friendly version of the accelerator core.

4.2. ITA-URAM-DMA Adapter:

ITA expects a high rate of 1024 bits streams from a 32-ported Tightly-Coupled Data Memory (TCDM). This memory with each 32 bits datastream from all 32 ports from the original ASIC design is used to store all necessary data for the computation, including the model's weights and biases, the input query, key, and value matrices, and all intermediate results such as the attention score matrix (QK) and the context vector (AV) .



To bridge this architectural gap, we created the **ITA-URAM-DMA Adapter**.

When ITA requests a wide 1024-bit word, it presents 32 different addresses to the TCDM interface simultaneously. Our controller's interleaving logic ensures that these consecutive logical addresses are mapped to different physical URAM banks. For example, the word at address A maps to URAM 0, A+4 maps to URAM 1, and so on. This architecture allows a single wide access from the ITA to be serviced in

parallel by all 32 URAMs, effectively emulating the high-bandwidth TCDM required by the accelerator.

The ITA-URAM-DMA Adapter is the central arbiter for the memory subsystem. It manages access to the URAM banks from two masters: the ITA core during high-performance computation, and the DMA during data transfer phases. The adapter grants the DMA exclusive access to load weights and input data before inference begins, and again to read final results after computation is complete.

4.2.1. Key Components

- **TCDM Ports (32):** The parallel interface through which the ITA accelerator requests read and write operations when dma_mode_i is low. Each port (tcdm[0] to tcdm[31]) is an hci_core_intf bundle.
- **DMA Port:** A single, high-throughput streaming interface for loading data into the URAMs from an external source (e.g., DDR memory) and reading results back out when dma_mode_i is high.
- **URAM Banks (32):** The physical memory storage, implemented using xpm_memory_tdpram primitives configured as URAMs. Each bank has a read port (Port A) and a write port (Port B).
- **Arbitration MUX:** A top-level multiplexer, controlled by the dma_mode_i signal, that selects whether the URAM banks are controlled by the ITA's TCDM interface or the DMA interface in any given cycle.
- **ITA Crossbar:** A network of multiplexers that routes requests from the 32 TCDM ports to the appropriate URAM banks when the system is in ITA mode.
- **DMA Demultiplexer:** Logic that routes a single DMA request to the correct target URAM bank based on the address when the system is in DMA mode.

TCDM Signal of ITAs:

This mode is active when dma_mode_i is 0.

tcdm[i].req	Input	1	Request valid from TCDM Port i.
tcdm[i].wen	Input	1	Write enable from Port i. 0=Write, f=Read.
tcdm[i].add	Input	32	Byte address for the memory operation.
tcdm[i].data	Input	32	Data to be written for a write operation.
tcdm[i].be	Input	4	Byte enable for a write operation. Always has the value 0 or 0,0,0,0,0,0,0,0,0,0,0,0, f,f,0,0,0,0,0,0,0,0,0,0,0,0,f,f
tcdm[i].gnt	Output	1	Grant signal to TCDM Port i. Pulsed high for one cycle when a request is accepted.
tcdm[i].r_data	Output	32	Data returned for a read operation.
tcdm[i].r_valid	Output	1	Asserted for one cycle when r_data is valid.

4.2.2. Memory Mapping and Address Decoding

The controller implements a memory interleaving scheme to distribute sequential addresses across the 32 URAM banks. This maximizes parallelism for the ITA's access patterns.

Given a 32-bit byte address tcdm.add:

- **Bits [1:0]:** Byte offset within a 32-bit word. These bits are ignored for bank and word selection.
- **Bits [6:2]: Bank Index**. These 5 bits (BANK_SEL_W = 5) directly select one of the 32 URAM banks. The bank index is (address / 4) % 32.
- **Bits [31:7]: Local Word Address**. These upper bits form the address within the selected URAM bank. The local address is address / 128.

This results in the following mapping of 32-bit words:

- Word at Address 0x0000 -> Bank 0, Local Address 0
- Word at Address 0x0004 -> Bank 1, Local Address 0
- ...
- Word at Address 0x007C -> Bank 31, Local Address 0
- Word at Address 0x0080 -> Bank 0, Local Address 1

Example Encoding Scheme:
For a 32-bit address A**:**

A[31:7]	A[6:2]	A[1:0]
<---------->	<---------->	<------>
Local Word Address (Sent to URAM Addr)	Bank Index (0-31) (Selects URAM Index)	Byte Select (Ignored, since byte aligned)

4.2.3.. Operation and Arbitration
The controller utilizes the dual-port nature of the URAMs to separate read and write operations.

- **Handshake:**
  The tcdm.gnt signal is asserted combinationally in the same cycle as tcdm.req. This simple handshake indicates the request has been received by the controller.

- **Conflict Free Bank Arbitration:**
  The crossbar logic evaluates all incoming requests simultaneously. If multiple TCDM ports target the same URAM bank in the same clock cycle, a fixed-priority scheme resolves the conflict: tcdm[0] has the highest priority, and tcdm[31] has the lowest. Only the request from the highest-priority port is passed to the URAM bank. Lower-priority requests targeting the same bank in the same cycle are ignored for that cycle. Since ITA always issues consecutive address requests, these always map to 32 different URAM banks, therefore there are no conflicts.

- **Write Operations:**
  A TCDM port j initiates a write by asserting req with wen=0. If it wins arbitration for its target bank i, its add, data, and be signals are routed through the write crossbar to the write port (Port B) of URAM bank i.

- **Read Operations:**
  A TCDM port j initiates a read by asserting req with wen=1. If it wins arbitration for its target bank i, its add signal is routed to the read port (Port A) of URAM bank i.

- **Read Data Latency:**
  Read data has a fixed latency of one clock cycle from the request, specified by the URAM instantiation.

An example flow would be:

Cycle 0: A port tcdm[j] asserts req. If it wins arbitration, its address is sent to the target URAM bank i. The controller internally registers that port j is expecting data from bank i.

Cycle 1: The URAM outputs the requested data. The read return logic uses the registered bank index (i) to select the correct data from the 32 bank outputs and routes it back to the original requester on tcdm[j].r_data. The tcdm[j].r_valid signal is asserted in this cycle.

4.2.4. Streaming DMA Interface (DMA Mode)

This mode is active when dma_mode_i is 1. It provides a simple, single-port AXI-Stream-like interface for bulk data transfers.

DMA Signals:

Signal	Direction	Description
dma_we_i	Input	Selects transfer direction. 1=Write to URAM, 0=Read from URAM.
dma_addr_valid_i	Input	Indicates dma_addr_i is valid.
dma_addr_i	Input	The byte address for the current transfer word.
dma_addr_ready_o	Output	Indicates the controller is ready for a new address.
dma_wdata_valid_i	Input	Indicates dma_wdata_i is valid for a write operation.
dma_wdata_i	Input	The 32-bit data word to be written to URAM.
dma_wdata_ready_o	Output	Indicates the controller is ready for a new data word.
dma_rdata_valid_o	Output	Indicates dma_rdata_o is valid for a read operation.
dma_rdata_o	Output	The 32-bit data word read from URAM.
dma_rdata_ready_i	Input	Indicates the external system is ready for read data.

4.2.5. Operation and Handshake

The DMA interface uses a coupled handshake mechanism.

- **Write Operation (dma_we_i = 1):**
  A write to a URAM bank occurs only when both dma_addr_valid_i and dma_wdata_valid_i are asserted in the same cycle. The dma_addr_i is decoded to select

a target bank, and the dma_wdata_i is written to it. The controller signals its readiness via dma_addr_ready_o and dma_wdata_ready_o.

- **Read Operation (dma_we_i = 0):**
  A read is initiated when dma_addr_valid_i is asserted and the controller is ready (dma_addr_ready_o is high). The dma_rdata_valid_o signal will be asserted and held high until the transaction is accepted by the destination (dma_rdata_ready_i is high). This implements a standard AXI-Stream handshake.

An example flow would be:

Cycle 0: The read address is accepted and sent to the target URAM bank.

Cycle 1: The dma_rdata_valid_o signal is asserted for one cycle, and the corresponding dma_rdata_o is driven with the data from the URAM.

4.3. Control and Dataflow Logic: Our design employs a holistic control scheme to manage the complex dataflow and sequence required to run the ViT inference on FPGA.

4.3.1. Finite State Machine (FSM): The FSM acts as the top-level controller to run ITA with the wrapper on FPGA. It receives commands from the CPU via an AXI-Lite interface and sequences the entire inference process. Its responsibilities include initiating DMA transfers, triggering the Sequencer for each major computation step, and initiating the DMA transfer of final results back to memory. Its states directly correspond to the major phases of operation:

- **Data Loading States (S_WAIT_..._DATA, S_SETUP_...):** Upon receiving a start signal, the FSM first enters a series of states to manage data ingress. It asserts dma_mode_i on the memory controller, giving the DMA exclusive access to the URAMs. It then triggers the DMA Address Generator with the correct base address and transfer length (e.g., BASE_PTR_WEIGHT0[Q] and ATTN_WB_LOAD_WORDS) and waits for the dma_ag_done signal to confirm that all weights or inputs have been loaded from the AXI-Stream interface.
- **Computation States (S_START_Q, S_WAIT_Q_DONE, etc.):** Once data is in URAMs, the FSM de-asserts dma_mode_i, giving the ITA core full control of the URAMs. It then steps through the Transformer layers by asserting the start signal for each dedicated **Sequencer** module (e.g., q_start, then k_start). It polls the corresponding done signal from the sequencer (q_done, k_done) to know when each major computation step is complete before proceeding to the next.
- **Results Writeback States (S_WAIT_..._READ_READY, S_DONE_...):** After the final computation step (e.g., OW or F2) is finished, the FSM re-asserts dma_mode_i to reclaim control of the memory. It waits for the downstream system to be ready (m_axis_tready) before triggering the DMA Address Generator to stream the final results from the URAMs out through the master AXI-Stream port.

4.3.2. Sequencer: The sequencer functions as a smaller engine, translating high-level commands from the FSM into the low-level register writes required by the ITA core. For a

given step, such as calculating the Query matrix (Q), the Sequencer's internal FSM performs the following cycle-accurate tasks:

1. **Iterates through Tiles:** It maintains internal counters (outer_tile_cnt, tile_inner_cnt) to loop through every data tile required for the operation.
2. **Calculates Pointers:** In its S_CALC_AND_PROG state, it uses the current tile counters and the pre-calculated BASE_PTR arrays to compute the precise, absolute memory addresses for the current tile's input, weight, bias, and output buffers, which is then translated inside the ITA-URAM-DMA Adapter.
3. **Programs the ITA Core:** It then sequences through a series of peripheral bus writes, using its reg_prog_cnt to program each of the necessary registers (pointers, control values, etc.) into the ITA core.
4. **Triggers and Waits:** Finally, it issues a write to address 0x0 to trigger the tile computation and transitions to its S_WAIT_HWPE state. Here, it monitors the hwpe_busy_i signal, waiting for it to go low, which indicates that the tile is complete and it can proceed to program the next one. Once all tiles are processed, it asserts its done_o signal back to the main FSM.

4.3.3. DMA Address Generator: This dedicated unit operates under the control of the main FSM to generate address streams for DMA transfers. It provides the URAM with the correct sequence of addresses when writing incoming data from the AXI streaming port or reading outgoing results to be streamed off-chip. Upon receiving a start_i pulse from the main FSM, it loads the provided base_address_in and transfer_len_i. It then generates a stream of incrementing 32-bit addresses, one per clock cycle, asserting m_axis_tvalid for each. It respects backpressure from the memory controller via m_axis_tready and asserts a single-cycle done_o pulse once the specified number of addresses has been generated. This provides the memory controller with the necessary address stream to correctly place incoming DMA data or retrieve outgoing results.

---

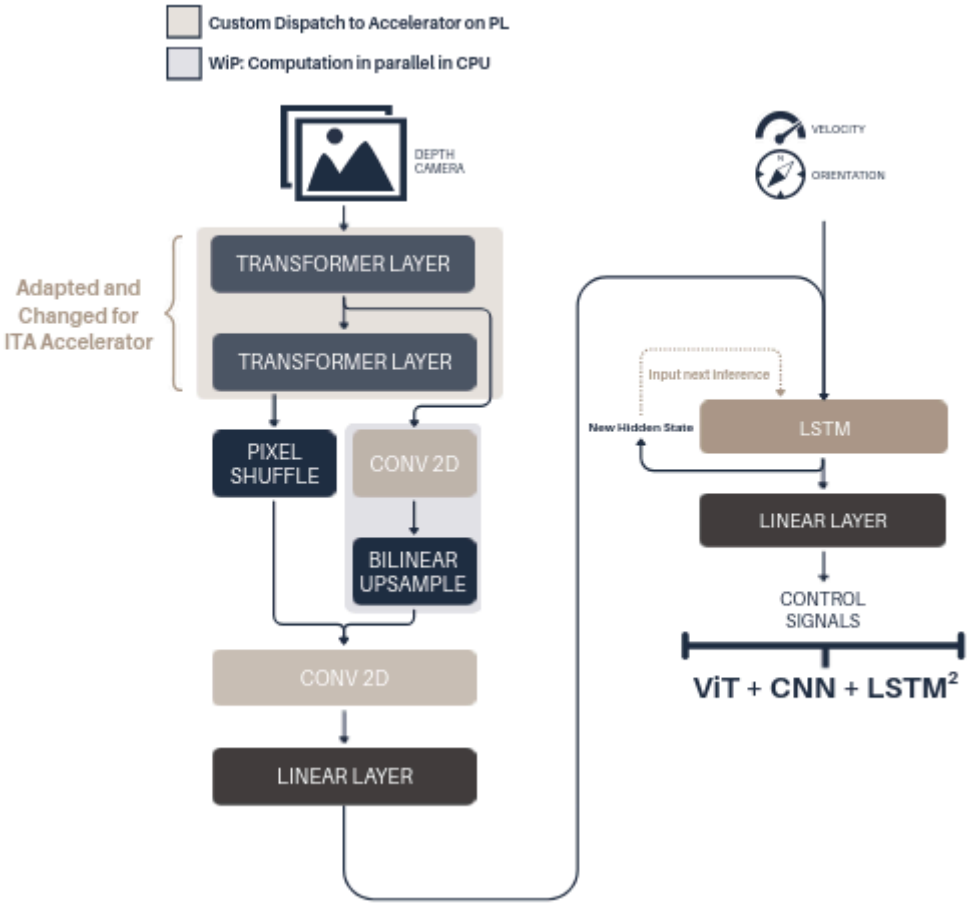
5. Software Framework and IREE Compiler

A central contribution of this project is the integration of a compiler toolchain for compiling a Vision Transformer model and deploying it on our custom FPGA hardware. This framework bridges the gap between high-level ML model development on a host PC and low-level execution on the embedded Kria SOM. It integrates a Hardware-in-the-Loop (HIL) simulation environment and leverages the IREE (Intermediate Representation Execution Environment) compiler to manage this complex flow.

5.1 ML Model with ViT

Our implementation adapts the Vision Transformer architecture specifically for drone navigation tasks, following the approach demonstrated in the VitFly paper. The model processes 60×90 grayscale images from a simulated depth camera, converting them into patches that are processed through transformer layers optimized for the ITA accelerator's constraints.

The architectural parameters were carefully selected to align with the ITA's hardware capabilities while maintaining model expressiveness. The sequence length of 128 tokens optimizes memory bandwidth utilization and computational unit efficiency within the accelerator. The embedding dimension of 64 provides sufficient representational capacity while fitting within the URAM memory constraints of the FPGA fabric. The projection dimension of 192 enables efficient attention computation on the specialized hardware units, while the feed-forward dimension of 256 maximizes utilization of the processing elements. The implementation uses a single attention head to reduce hardware complexity while maintaining adequate performance for the navigation task.



The model architecture incorporates several custom components designed specifically for hardware acceleration. The overlap patch merging module uses fixed output sizes to ensure predictable memory access patterns, which is crucial for efficient FPGA implementation. The attention layers are modified to use hardware-friendly matrix dimensions that align with the ITA's datapath width and memory organization. Similarly, the feed-forward networks are optimized for the accelerator's computational pipeline, ensuring maximum throughput and minimal idle cycles during inference.

5.2 Quantization and Custom Modules

The deployment of ViT models on resource-constrained FPGA hardware necessitates aggressive quantization strategies. Our approach implements symmetric 8-bit quantization throughout the network, with particular attention to the ITA's specific requirements for integer-only computation.

Our quantization-aware training pipeline uses PyTorch's quantization framework with custom configurations tailored to the ITA's signed symmetric 8-bit format. The quantization configuration specifies a range from -128 to 127 for both activations and weights, using per-tensor symmetric quantization without range reduction. This approach ensures compatibility with the hardware's integer arithmetic units while maintaining sufficient numerical precision for accurate inference.

The most critical custom component is the integer-approximated softmax implementation, which operates entirely in the integer domain to match the ITA's streaming softmax unit. This module implements the same bit-shifting and approximation algorithms used in the hardware, maintaining compatibility with both QAT training through float simulation and post-conversion inference through integer execution. During training, the module provides surrogate gradients to maintain proper gradient flow while simulating the exact quantization behavior that will occur in hardware.

The attention and feed-forward blocks are implemented as quantization-aware modules with explicit control over quantization boundaries. These modules use QuantStub and DeQuantStub placements for precise management of quantization points and FloatFunctional operations for matrix multiplications that respect quantization constraints. A particular challenge addressed is the transition between signed 8-bit inputs and unsigned 8-bit softmax outputs, which required careful handling to maintain numerical accuracy while adhering to the hardware's datapath requirements.

The validation framework ensures bit-exact correspondence between PyTorch simulation and hardware execution through extraction of quantized weights, biases, and requantization parameters. This process generates comprehensive test vectors for hardware verification and exports the model to ONNX format with custom operation replacements to ensure compatibility with the IREE compilation pipeline.

5.3 Compilation and Deployment Workflow

Our compilation and deployment workflow handles the complexities of cross-compilation, where tools are built on a powerful x86-64 host machine to generate code that runs on the target aarch64 ARM processor of the Kria SOM.

The model compilation process transforms the quantized PyTorch model through several stages. First, the QAT model is converted to ONNX format with custom operations replaced by IREE-compatible equivalents. Next, the ONNX model is imported to IREE's MLIR representation using the iree-import-onnx tool. Custom transform sequences are then applied to identify and group operations suitable for hardware acceleration. Finally,

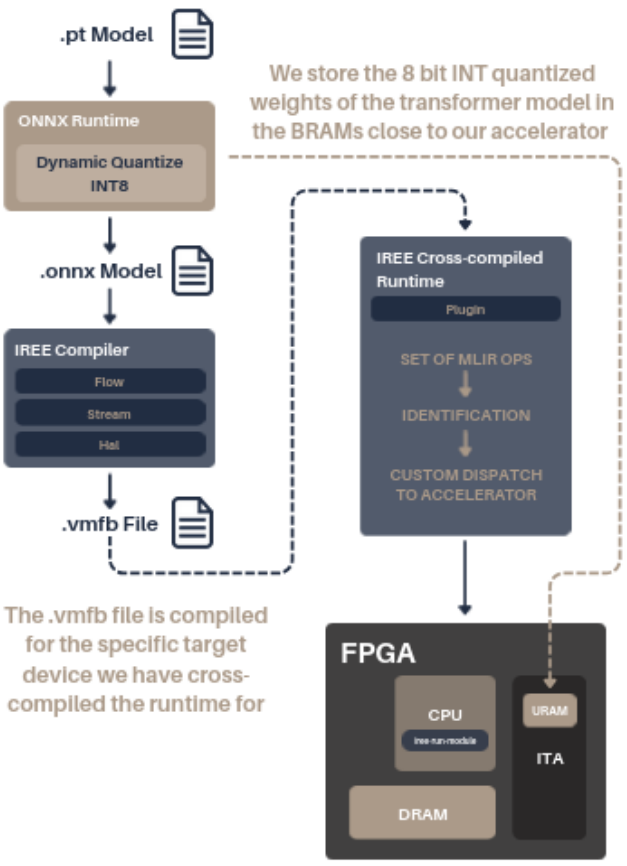
the model is compiled to the target vmfb format with embedded custom dispatch calls for hardware execution.

The final step involves cross-compiling the C++ runtime application that executes on the Kria. This application links against the IREE runtime libraries and our custom dispatch plugin, creating the final executable for deployment on the target hardware. The runtime application includes the IREE runtime for model execution, a custom Hardware Abstraction Layer driver for ITA communication, Linux kernel interfaces for DMA operations, and a UDP communication stack for simulation integration.

5.4 IREE Custom Dispatch for Hardware Acceleration

The centerpiece of our software-to-hardware interface is a custom IREE dispatch plugin that extends the IREE runtime to offload computations to our novel hardware backend. Using MLIR's Transform Dialect, our custom transforms identify transformer attention patterns in the compiled graph and replace them with calls to our hardware accelerator.

The transform-based operation detection system uses pattern matching to identify suitable operations during compilation. When attention layer patterns are detected, they are grouped together and replaced with custom hardware dispatch calls. This approach enables transparent hardware acceleration without requiring changes to the high-level model code.



The dispatch mechanism translates high-level inference calls into low-level hardware operations including AXI DMA transfers for input and output data, AXI-Lite register programming for control parameters, hardware execution management and status monitoring, and result retrieval with appropriate format conversion. The memory management subsystem handles the complex data flow between different memory domains, including host DDR for model weights and buffers, FPGA URAM for high-speed computation, and CPU cache for temporary storage.

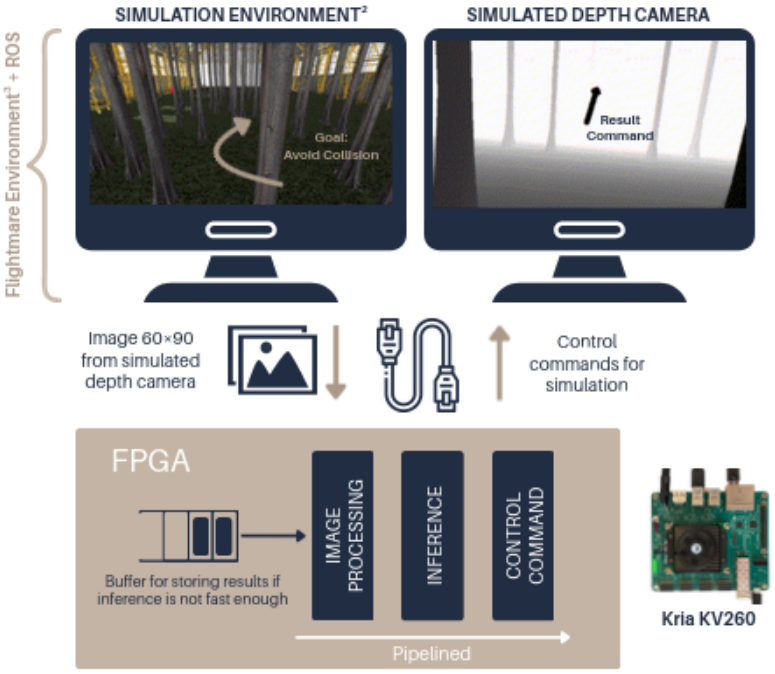
This integrated approach transforms the standalone hardware accelerator into a seamless extension of the standard ML inference pipeline. The dispatch layer coordinates efficient data staging and movement, manages memory alignment and padding requirements, maintains coherency between CPU and FPGA views of data, and provides comprehensive error handling and recovery mechanisms. The result is transparent hardware acceleration

that enables significant performance improvements while maintaining compatibility with standard ML frameworks and development workflows.

---

6. Implementation

The project was implemented using **System Verilog and Verilog**. The logic for the FSM, Sequencer, and Address Generator was implemented as a synthesizable hardware controller that autonomously manages the entire inference pipeline. The dimensions of the Transformer model layers (e.g., sequence length, embedding size) are parameterized, allowing the controller to calculate all necessary memory pointers and loop bounds at compile time, creating a highly specialized and efficient control path.



The ITA is controlled by the Kria's ARM CPU running a Ubuntu environment and ROS. The interaction follows these steps:

1. **Pattern Detection:** At compile time, our AI compiler detects the Attention Layer operations, and groups them together into a single dispatch group.
2. **Dispatch to Accelerator:** At run time, the dispatch group dispatched to the accelerator to be executed.
3. **Initiation:** The CPU initiates an inference task via an AXI-Lite interface, writing control information (e.g., base addresses of data in DDR memory) to an AXI DMA IP, which handles the transfer.
4. **Data Transfer:** Upon receiving a start command, the accelerator's FSM configures the system's central DMA to stream the required input data (e.g., image patches, weights) from external DDR memory. This data flows into the accelerator through its slave

AXI-Stream (S_AXIS) port and is written into the on-chip URAMs by the memory controller.

5. **Computation:** The ITA processes the data, using the URAM for intermediate storage. The FSM and Sequencer work together to feed the ITA core with the correct data tiles in the correct sequence for each stage of the MHA calculation.
6. **Results Return:** Once the computation is complete, the FSM configures the memory controller to read the final result matrix from the URAM and stream it out of the master AXI-Stream (M_AXIS) port. This data is received by the central DMA and written back into a pre-allocated buffer in DDR memory.
7. **Completion:** The accelerator raises an interrupt to signal to the CPU that the inference is complete and the results are ready in DDR memory.
8. **Communication with Simulation:** On the higher level, our FPGA communicates with the host PC that runs the drone simulation environment through a UDP channel with an ethernet cable.

---

7. Framework Implementation

The primary objective of this project was the successful implementation of the framework, creating an end-to-end path from a high-level machine learning model to a custom hardware accelerator on the AMD Kria KR260 platform.

The hardware accelerator subsystem was fully designed in RTL and successfully synthesized for the target FPGA. This process verified that the architecture, including the ported compute core and all custom control and memory interface logic, is compatible with the physical resources of the device. The functional correctness of the hardware was validated to ensure all components operate according to their design specifications.

Complementing the hardware, the complete software toolchain was developed and deployed. This included the crucial integration with the IREE compiler environment, establishing a seamless connection between the host processor and the FPGA fabric. The runtime software was successfully executed on the Kria's ARM processor, confirming its ability to manage the hardware and validate the end-to-end software control path.

The successful implementation of these interconnected hardware and software elements confirms our integrated HW/SW co-design approach. This wrok now stands as a complete and verified framework, providing a solid foundation for deploying accelerated machine learning in real-time robotics applications.

---

6. Implementation Results

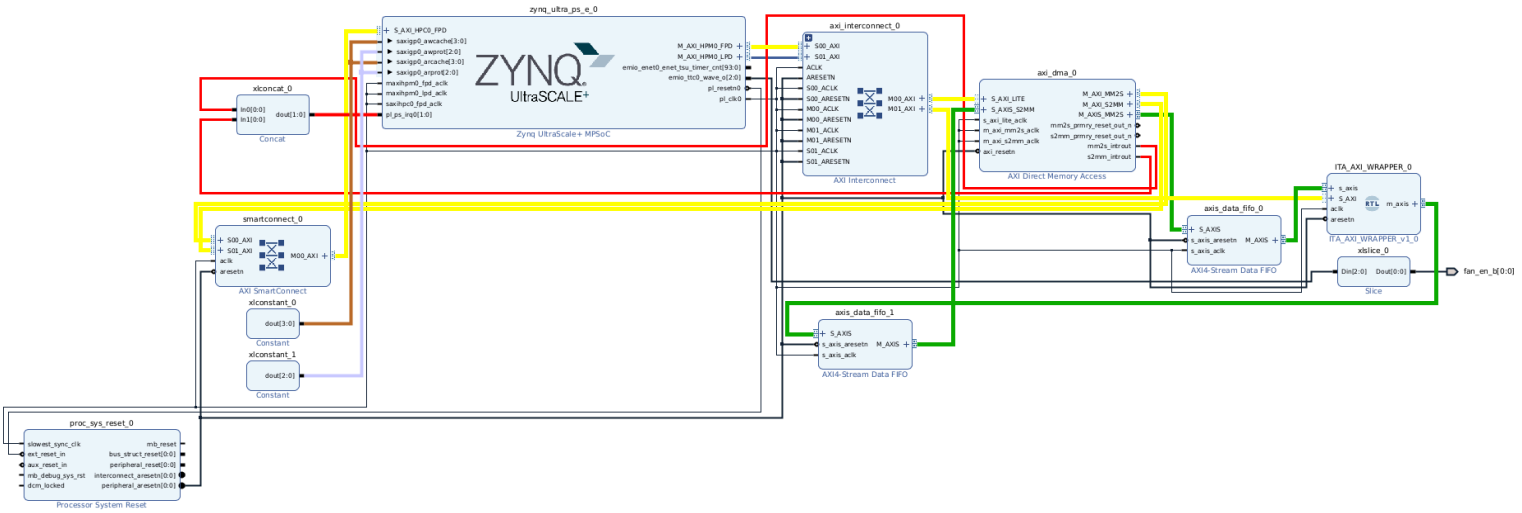
6.1 System Architecture and Signal Flow

The complete system implementation on the AMD Kria KR260 platform demonstrates successful integration of the ITA accelerator with the Zynq UltraScale+ processing system. The block diagram illustrates the three primary signal domains that enable seamless communication between the ARM processor and the FPGA accelerator.

The red signal paths represent the interrupt infrastructure that manages asynchronous communication between the hardware accelerator and the software stack. These interrupt signals notify the ARM processor when DMA transfers are complete, when the ITA has finished processing a computation tile, and when results are ready for retrieval. The interrupt system ensures efficient CPU utilization by eliminating the need for polling-based status checking during hardware execution.

The green signal paths indicate the high-throughput data streaming connections that handle the bulk movement of inference data. These AXI Stream interfaces carry input image patches, model weights, intermediate computation results, and final inference outputs between the DDR memory system and the on-chip URAM storage. The streaming architecture enables sustained data throughput that matches the computational bandwidth of the ITA core.

The yellow signal paths represent the AXI control infrastructure that manages configuration, status monitoring, and low-bandwidth command communication. These connections include AXI-Lite interfaces for register programming, DMA configuration channels, and system control signals that coordinate the overall inference pipeline execution.



6.2 Resource Utilization Analysis

The synthesis and implementation results demonstrate efficient utilization of the FPGA fabric while maintaining adequate headroom for system expansion and optimization. The resource consumption reflects the complexity of integrating a complete transformer accelerator with comprehensive memory and control infrastructure.

Resource Type	Used	Available	Utilization %
CLB LUTs	73,003	117,120	62.33%
CLB Registers	48,523	234,240	20.72%
Block RAM	82	144	56.94%
UltraRAM	32	64	50.00%
DSP48E2	553	1,248	44.31%

The logic utilization of 62.33% indicates substantial computational complexity while preserving room for additional features or optimization. The register utilization of 20.72% reflects the pipeline-heavy nature of the design, with extensive use of registers for timing closure and data staging throughout the accelerator pipeline.

6.3 Memory Subsystem Performance

The memory architecture achieves the target bandwidth requirements for sustained transformer inference through strategic utilization of the available memory hierarchy. The UltraRAM allocation represents a critical design decision that enables the high-bandwidth parallel access patterns required by the ITA core.

Memory Type	Quantity	Total Capacity	Bandwidth Utilization
URAM288	32 banks	9 MB	1,024 bits/cycle
RAMB36E2	77 blocks	2.8 MB	Variable
RAMB18E2	10 blocks	0.36 MB	Variable

The 32-bank URAM configuration provides the parallel memory access capability essential for the ITA's computational throughput. Each bank supplies 32 bits per cycle, delivering the aggregate 1,024-bit bandwidth that matches the accelerator's datapath requirements.

6.4 Computational Resource Allocation

The DSP utilization demonstrates effective mapping of the transformer's arithmetic operations to the FPGA's dedicated computational resources. The allocation balances the requirements of matrix multiplication, quantization operations, and control logic across the available DSP48E2 blocks.

Component	DSP48E2 Count	Utilization %	Primary Function
ITA Core	425	34.05%	Matrix operations, quantization
Memory Controller	89	7.13%	Address generation, data formatting
System Infrastructure	39	3.13%	Control logic, status monitoring
Total	553	44.31%	Complete system

The ITA core consumes the majority of DSP resources for its matrix multiplication units and quantization hardware. The memory controller requires significant DSP allocation for address calculation and data formatting operations that support the complex memory access patterns. The remaining DSP utilization supports various control and monitoring functions throughout the system.

6.5 Timing Analysis and Performance Validation

The timing analysis reveals robust design margins that confirm successful implementation at the target operating frequency of 25 MHz. The critical timing paths demonstrate positive slack values, indicating that the design operates reliably within specified performance parameters.

Timing Metric	Value	Status
`Worst Negative Slack (WNS)	+1.690 ns	PASSED
Total Negative Slack (TNS)	0 ns	PASSED
Clock Period	40 ns (25 MHz)	MET
Setup Time Margin	>1.6 ns	ADEQUATE

The timing analysis shows that the most critical paths occur within the ITA core's data processing pipeline, specifically in the input buffer and weight streaming subsystems. These paths maintain comfortable timing margins of approximately 1.69 ns, providing sufficient headroom for process, voltage, and temperature variations during operation.

The clock distribution network demonstrates balanced propagation delays across the FPGA fabric, with clock skew maintained within acceptable limits. The single-clock domain approach simplifies timing closure while enabling deterministic data flow throughout the accelerator pipeline.

---

8. Marketability, Re-usability, and Future Work

Beyond academic exploration for the competition, this project has a strong focus on practical application and community contribution. The HW/SW co-design framework developed in this

project is a viable solution for a range of market-driven applications where real-time and low-power inference is crucial. Potential markets include:

- **Commercial Drone Systems:** Inspection, surveying, and security drones that require onboard intelligence to operate safely and effectively beyond visual line of sight.
- **Autonomous Mobile Robots (AMRs):** Warehouse and logistics robots that can use Vision Transformer models for advanced scene understanding and navigation in dynamic environments.
- **Edge Vision Applications:** Any smart camera or edge device that needs to perform complex visual analysis, such as traffic monitoring or industrial quality control, without relying on cloud connectivity. This framework could demonstrate a low-risk, high-performance path for companies to integrate powerful Transformer-based AI into their edge products using off-the-shelf, commercially supported hardware.
- **Open-Source Contribution:** To contribute to the community, our project is fully open-source. The complete project, including the RTL source code, Vivado block designs, and ROS integration software, is available on GitHub:
- **Reusable Framework:** We provide a well-documented framework for setup and run. The clear separation between the main FSM, the step-sequencer, and the DMA logic serves as a robust template that can be adapted for other robotics acceleration projects with custom hardware solutions.

1. Drone Repository:

https://github.com/OpenHardware-Initiative/Drone-OA-IREE-ViT-Accelerator

2. RTL repository:

https://github.com/OpenHardware-Initiative/ITA-FPGA

3. Software Repository:

https://github.com/OpenHardware-Initiative/kria-inference/tree/e24e335b289f97a4756c1ed249392c99c4ae8652

8.1. Future Work

While this project successfully demonstrates a complete end-to-end accelerated pipeline, it also serves as a strong foundation for several exciting future research and development directions.

- **Physical Drone Deployment and Real-World Validation:** The critical next step is to transition the system from simulation to a physical drone platform. This would involve integrating the Kria KR260 with the drone's flight controller and camera system to validate the performance and reliability improvements in real-world flight conditions. This would provide the ultimate proof of the accelerator's practical value and expose new challenges such as sensor noise, vibration, and environmental variations.
- **Development of a Modular Acceleration Framework:** The current implementation creates a tightly-coupled link between a specific ViT model and a specific hardware accelerator. A highly valuable future effort would be to evolve this into a more flexible framework. This framework would aim to:

    - **Decouple the Model from the Hardware:** Create a standardized software abstraction layer that allows different quantized ViT models to be deployed without requiring significant changes to the control logic.
    - **Enable Different Accelerators:** Define a common hardware interface that would allow different hardware accelerators (like the ITA, a CNN accelerator, or future designs) to be "plugged into" directly to the system. This would create a powerful platform for researchers and developers to easily benchmark and compare different acceleration strategies for robotics applications.
- **Exploration of Hybrid Model Architectures:** With a robust acceleration platform in place, future work could explore more advanced and hybrid model architectures, for instance transformers with convolutional layers for a better performance-accuracy trade-off on edge devices.

8.1 For reusability and reproduction of workflow

We have put a very detailed guide in the repository where you can find step by step what to do, as well as some extra tips from our experience working on this project.

---

9. Conclusion

This project successfully demonstrates a complete, end-to-end solution for accelerating Vision Transformer inference on an FPGA for real-time autonomous drone control.By integrating a compiler-based hardware adaptation and model optimization, we have created a practical and replicable framework for deploying high-performance transformers on edge robotics platforms.

Crucially, we bridged the gap between the software model and our custom hardware using the IREE compiler framework on FPGAs. By developing a custom dispatch backend, we enabled a seamless toolchain that automates the offloading of computations from the host CPU to the FPGA fabric. This compiler-centric approach is the key to transforming the standalone hardware and model into a cohesive, programmable system.
Our hardware contribution was the successful porting of the open-source, ASIC-centric ITA to the AMD Kria KR260 platform. This required significant architectural redesigns, including the development of a novel banked URAM memory controller and a state-driven control fabric to manage the accelerator's complex operational sequence. Concurrently, we adapted a ViT model for a drone navigation task, performing 8-bit quantization to align it with the hardware's integer-only datapath.

This work not only validates the effectiveness of the ITA architecture on FPGAs but also provides a comprehensive, open-source framework for future research and development in real-time, ML-driven autonomous systems.

---

10. References

[1] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby**,** “An image is worth 16×16 words: Transformers for image recognition at scale,” *arXiv preprint arXiv:2010.11929*, 2021.

[2] “ITA: An Energy-Efficient Attention and Softmax Accelerator for Quantized Transformers” Gamze Islamoglu, Moritz Scherer, Gianna Paulin, Tim Fischer, Victor J. B. Jung, Angelo Garofalo, Luca Benini

[3] The PULP Platform, "ITA: Integer-Only Transformer Accelerator," GitHub Repository, 2022. [Online]. Available: https://github.com/pulp-platform/ITA

[4] Open Robotics, "Robot Operating System 2 (ROS 2) Documentation," [Online]. Available: https://docs.ros.org/

[5] Microsoft, "AirSim: Open-source simulator for autonomous vehicles," [Online]. Available: https://microsoft.github.io/AirSim/

[6] A. Bhattacharya, N. Rao, D. Parikh, P. Kunapuli, Y. Wu, Y. Tao, N. Matni, and V. Kumar, “Vision transformers for end-to-end vision-based quadrotor obstacle avoidance,” Proc. IEEE Int. Conf. Robot. Autom. (ICRA), 2025.

[7] IREE Authors, "IREE (Intermediate Representation Execution Environment)," GitHub Repository, 2024. [Online]. Available: https://github.com/iree-org/iree

[8] AMD, "Kria KR260 Robotics Starter Kit," Product Page. [Online]. Available: https://www.xilinx.com/products/som/kria/kr260-robotics-starter-kit.html