Politecnico di Torino

Master's Degree in Electronics Engineering

Master's Degree Thesis

Implementation of Neural Network on FPGA Using HLS

Supervisor: Professor Mihai Lazarescu

Candidate: Bilal Awad (S320760)

Academic Year 2024–2025

Extended Abstract

Introduction

There has been increased need for low-power real-time AI at the edge and consequently increased need to deploy deep learning models like TCNs on hardware accelerators. TCNs can be used in sequential tasks like prediction and localization in signals but optimizing them on resource-limited platforms like FPGAs has been challenging because of issues with complexity and precision and architectural mismatch.

This work addresses these challenges by applying a trained TCN on an FPGA in order to provide high accuracy, low latency, and energy-efficiency for edge inference. Starting with a PyTorch model, this work performs a full deployment pipeline with fixed-point quantization, manual conversion to synthesizable C++, and hardware optimization with High-Level Synthesis (HLS).

The design goal primarily entails synthesizing an efficient and power-aware version of a residual TCN by trading off performance and efficiency. The power and latency are optimized in the design with careful handling of accuracy-throughput tradeoffs by means of synthesis-level transformations.

Lastly, the thesis offers a reproducible and feasible method of deploying quantized TCNs on embedded hardware and thereby presents key architecture strategies for successful edge AI deployments.

Literature Review

Hardware-accelerated neural networks increasingly play a pivotal part in energy-efficient real-time inference in edge devices. An FPGA sits in between ASIC performance and the flexibility of a GPU and fits into IoT and embedded systems applications. Vivado HLS is a High-Level Synthesis (HLS) tool for implementing tuned neural networks in C++ with the assistance of techniques such as pipelining and quantization.

While there has been extensive FPGA implementation of CNNs and RNNs, there is less work on full TCN implementations, especially those that include residual connections and dilated convolutions. While TCNs are computationally efficient at capturing long-range dependencies and thus can be used in constrained settings, the majority of the existing work is through software-level quantization or general-purpose hardware.

This thesis provides a complete FPGA implementation of a flexible-padded residual TCN with skip connections. The PyTorch baseline model is manually re-implemented in fixed-point C++ with direct control of the precision and usage of the hardware resources.

The manual paddings, LayerNorm/Dropout as an option and HLS directives have been employed in optimising performance.

The outcome is an energy-efficient reproducible TCN design on FPGAs. The design achieves a trade-off between accuracy and power efficiency and facilitates real-time inference and hence offers a viable solution for low-power sequential deep learning systems.

Methodology

This work presents a top-to-bottom design process to convert an abstract TCN network model into hardware-friendly form for real-time FPGA implementation. The design aim was to attain decent predictive accuracy with minimal computational latency, resource utilization, and energy expense.

We started with a TCN model based on TensorFlow. For easier processing of quantization steps, the model was re-implemented in PyTorch for better control over the quantization-aware training process. The network was quantized to a low-precision integer format to achieve a smaller memory footprint and negligible arithmetic complexity, a requirement for inference in an embedded system.

The PyTorch model was converted into synthesizable fixed-point C++ arithmetic through a process of quantization and hand-coded translation. This gave direct hardware synthesis control over numerical behavior and hardware resource usage. Special attention was paid to preserving functional correctness of model form, dataflow, and fixed-point operation while optimizing for FPGA hardware constraints.

To further improve performance and latency, the C++ code was implemented with High-Level Synthesis (HLS) with techniques such as pipelining loops, array partitioning, and AXI streaming interfaces. Pragmas were employed to control synthesis and uncover parallelism amongst time steps and layers.

Finally, the optimized IP core was integrated into a full FPGA system design with the help of Vivado IP Integrator. Adding to this was the introduction of clocking, reset logic, AXI memory interfaces, and control by the ZYNQ processing system. This resulted in a full embedded deployment system capable of real-time deep learning inference with minimal latency and power usage.

Key Findings

The Temporal Convolutional Network (TCN) was implemented in three variants: the TensorFlow baseline model, a PyTorch re-implementation, and a fixed-point implementation in C++ with FPGA targeting. While the TensorFlow model was the most accurate at 0.065 m^2 Mean Squared Error (MSE), the PyTorch implementation was in high fidelity at 0.1199 m^2, establishing architectural consistency in frameworks. The fixed-point FPGA

implementation resulted in an MSE of 0.2365 m^2 and was evidence that despite numeric degradation from quantization, the model was quite accurate for real-time use.

High synthesis optimizations played a crucial role in meeting real-time deadlines. The latency of the reference non-pipelined design was 16.51 μs and was brought down to 14.58 μs with the help of partial pipelining techniques (i.e., pipelining at smaller modules). A completely pipelined design with module-level streaming brought down the latency to 9.65 μs at an initiation interval (II) of 242 cycles and provided high throughput for steady inference workloads.

The transition from partial to full pipelining was accompanied by a steep resource usage boost: LUTs increased from 45k to 156k, FFs from 16k to 178k, and DSPs from 157 to 203. Increase in resource consumption apart, the pipelined design delivered a much improved latency/throughput behavior. Vivado's Report Power power estimation indicated on-chip power dissipation of 1.871 W, which was partitioned into static (0.365 W), dynamic logic (0.892 W), clock (0.422 W), and BRAM/DSP (0.192 W) components. The energy cost per inference was 18.05 μJ, demonstrating the energy efficiency of the fixed-point, quantized implementation.

By combining high-level synthesis with fixed-point optimization and quantization carried out manually, the resulting solution strikes a good tradeoff among speed, power consumption and accuracy. The results confirm the suitability of small real-time TCN inference pipelines' deployment on resource-constrained FPGAs for edge computing applications.

Discussion

The results affirm fixed-point quantization and FPGA acceleration as an appropriate option for real-time implementation of deep models like TCNs in constrained systems. The accuracy compromises—from 0.065 m^2 MSE in TensorFlow to Pytorch 0.1199 m^2 to 0.2365 m^2 in fixed-point C++ are acceptable in the edge application in question where latency and energy are of concern. The results show an acceptable tradeoff between hardware constraints and model accuracy.

Unlike previous work, which tends to leave off at software-level quantization or doesn't account for hardware deployment, this work differs in providing an end-to-end system—from model development and quantization to hardware synthesis and evaluation. The integration of high-level synthesis (HLS) for FPGA implementation with pipelining at the loop level and fixed-point custom arithmetic sets this work apart by illustrating the potential in translating software optimizations into real hardware efficiency.

The analysis also determined the principal bottlenecks in architecture—namely the impact of residual blocks on end-to-end latency—and demonstrated how pipelining and streaming (e.g., via #pragma_HLS_DATAFLOW) can mitigate them. The area-performance

trade-off was also explicitly measured and demonstrated how pipelining depth boosts throughput but with increased LUT, FF, and DSP usage.

To put simply, this thesis contributes a reproducible and jointly optimized design method bridging the hardware deployment efficiency and edge AI model accuracy gap.

Practical Implications

This effort establishes the feasibility and efficacy of applying Temporal Convolutional Networks on edge devices via efficient fixed-point quantization, PyTorch-to-C++ compilation, and HLS pipelining. The design proposed here has low latency (9.65 μs) and energy consumption (18.05 μJ per inference), and this meets the specifications of real-time and energy-limited applications like activity monitoring and robotics.

It shows that with meticulous design, pipelining and aggressive quantization can be used without substantial loss of accuracy. It is applicable widely to small time-series models and validates the potential of FPGAs as a viable choice for low-power applications as an alternative to traditional embedded processors.

Limitations and Future Directions

While TCN was successfully implemented on FPGA with reasonable latency and energy performance values, there exist real constraints. Quantization in fixed points resulted in a reduction in accuracy with a priority assigned to precision-efficiency tradeoff. Pipelining also resulted in an increase in hardware resource utilization, which may limit implementation on small-size FPGAs.

Future work may involve mixed-precision computing, hardware-aware neural architecture search, and real-world deployment with extensive power analysis to improve the accuracy and real-world usability even further.

Conclusion

This work discusses an end-to-end process of porting a Temporal Convolutional Network from TensorFlow to FPGA and re-implementation in PyTorch with control over quantization and synthesis to fixed-point C++. Incremental design steps from non-pipelined to pipelined permitted performance optimization in real-time while being within the hardware limits.

One of the end designs achieved 9.65 μs latency and 18.05 μJ energy per inference with an imperceptible degradation in accuracy and proved that the FPGA can be used in deep learning for energy-efficient and low-latency edge applications. The pipeline opens the door to future work on the implementation of neural networks on reconfigurable hardware.