ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Integrated Systems Laboratory

Semester Thesis at the
Department of Information Technology and
Electrical Engineering

# TNN-on-MCU: Efficient Ternary Inference on PULP

Students: Student A
Student B

Advisors: Georg Rutishauser
Moritz Scherer
Robert Balas

Professor: Prof. Dr. Luca Benini

Handout Date: N/A
Due Date:

# 1 Project Goals

Efficient inference of neural networks (NNs) on embedded, microcontroller-based systems is a research topic which has attracted intense interest in recent years. The tight limits imposed on memory (commonly in the 100s of KiB), storage (sometimes none, up to ~16MiB off-chip) and compute (commonly 1-8 low-power RISC cores running at 10s to 100s of MHz) resources these platforms impose on the NN models targeting them has given rise to a number of model design and training techniques to reduce model size, complexity and their memory footprint. [1]–[3]

The most ubiquitous of these techniques is aggressive **quantization**: Instead of using full-precision floating point formats to store model parameters and perform computations, parameters and intermediate results are quantized to low-bitwidth integer formats. This inherently reduces model size and the memory footprint required for inference. Furthermore, specialized instruction set architecture (ISA) extensions can be leveraged to enhance compute throughput by performing multiple operations simultaneously in a SIMD fashion. At IIS, such extensions have been developed for PULP, the research-oriented RISC-V microcontroller family developed in collaboration with the University of Bologna. For example, the most recent incarnation of the RI5CY core at the heart of PULP systems provides SIMD-style[1] single-cycle MAC instructions for 4-bit and 2-bit operands. [4]

The most extreme form of quantization yields Binary Neural Networks (BNNs), where intermediate results and model parameters are represented as single bits representing the values {-1, 1}. [5] A further advantage of BNNs is that the multiply-accumulate operations which represent the main workload of NN inference reduce to bitwise XNOR and popcount instructions. This means that they can be efficiently executed on microcontrollers implementing standard instruction set architectures, such as ARMv6/7/8-M or RISC-V. [2]

While BNNs can perform many tasks, they are commonly outperformed by **Ternary Neural Networks (TNNs)**, where weights and activations take values in the set {-1, 0, 1} (termed *trits*). While it is still possible to map such networks to standard ISAs, it is not as efficient or straightforward as for BNNs: 32 binary MACs can be executed in 4 instructions (XOR, popcount, logical shift, subtraction). In contrast, 32 ternary MACs require 13 standard instructions. However, by exploiting the existing extensions to the RISC-V ISA, this can be reduced to 16 ternary MACs in a single cycle, which should in theory already enable higher throughput for TNNs than what can be achieved for BNNs (as there are no ISA extensions to support BNN execution).

[1] https://en.wikipedia.org/wiki/SIMD
[2] https://people.eecs.berkeley.edu/~krste/papers/EECS-2016-1.pdf

**The goal of this project is to enable efficient execution of TNNs on PULP-based systems.** This will be achieved by a two-tracked process, which is why this project is proposed as a joint semester thesis for two Master's students.

## 1.1 Hardware Track:

One student will focus on hardware support. They will start with an unmodified PULP system and modify it to enable more efficient inference of TNNs. To do so, we will take advantage of the fact that a each trit represents only ~1.6 bits of information, but is represented as a 2-bit value in hardware – the fourth value that can be represented by a pair of bits is never taken on. A vector of $n$ trits can take on $3^n$ different configurations. It is thus possible to uniquely identify a sequence of $t$ trits by a bit vector of length $b$, so long as $3^t \leq 2^b$. **By choosing $t = 5$ and $b = 8$, we can store 20 trits in a 32-bit word.**

To take advantage of this compressibility, we want to **extend the ISA** by an instruction (**"compressed MAC"**) which performs a MAC operation on 20 compressed trits in one cycle by uncompressing the compressed values on-the-fly and performing the MAC operation. A very hardware-efficient encoding scheme has been constructed by Muller et al. [6], and we will adopt this scheme. This modification will enable 25% higher peak throughput at a low hardware overhead. To further optimize inference speed, a single-cycle thresholding operation will be implemented. This thresholding operation will have a second purpose, which is to allow for on-the-fly compressed encoding of the produced values, so the final outputs can be used as inputs to another compressed MAC. This will require modifications to the RI5CY core, which will need to be functionally verified and checked for performance regressions (timing degradation), as well as resource cost.

With the modified core in place, the student will assemble and test a complete System-on-Chip (SoC) with the core at its heart. This will potentially involve collaboration with other groups who may design further system components (e.g., specialized accelerators) to integrate those components and ensure functionality.

Starting before the system is completely assembled, the student will (again, potentially in collaboration with other students/groups) put it through a synthesis and backend flow as discussed in the VLSI II lecture, with the goal of taping out a fully functional, TNN-optimized SoC.

## 1.2 Software track:

The other student will focus on the TNN software implementation. Starting from an existing BNN application targeting PULP systems [7], they will first **collect results on its accuracy and inference speed**. The student will then **train a TNN** to replace the

original BNN using our in-house quantization toolkit (QuantLab, [3]). The TNN's statistical accuracy is expected to be higher than that of the original BNN.

The student will continue by **establishing a TNN inference infrastructure** for a baseline PULP system (with XpulpNN extensions, but no TNN-specific ISA modifications). Where possible, they will reuse the PULP-NN library [2]. This will allow us to compare the inference speed of isolated kernels as well as the mapped TNN trained in the previous step to baseline BNN kernels and the original BNN.

Having established a performance baseline for both TNN and BNN, the student will then **extend the inference code to take advantage of the ISA extensions** developed in the hardware track of the project, allowing for a comprehensive performance comparison between BNN, TNN on the baseline architecture and TNN on the optimized architecture.

# 2 Tasks

As there are cross-dependencies between the software and hardware track, close collaboration and effective communication are of vital importance to this project. A tentative outline for each track is outlined below:

## 2.1 Hardware Track

### Phase 1: Familiarization with Baseline Platform (2-3 weeks)

The PULP ecosystem is extensive, and some time will be required to familiarize with the established toolchains, software development kit and the implementation and testing flows. Do not hesitate to ask your supervisors for information!

1. Setting up baseline system, simulation flow and code compilation and deployment in the simulation environment.
2. Familiarization with the synthesis flow of the baseline system.
3. Familiarization with the workings of the existing sub-byte SIMD instructions.

### Phase 2: ISA Extension implementation (5-7 weeks)

1. Planning and compiling the complete set of ISA extensions to be implemted. The goal is to support ternary MAC operations on compressed ternary data at 20 MACs/instruction and enable fused thresholding and compression.
2. Extending the core with the planned extensions and verifying their functionality.

[3] https://iis-git.ee.ethz.ch/spmatteo/quantlab

### Phase 3: Synthesis and Backend (3-5 weeks)

***Note that the synthesis part of this phase must overlap with the last phase!*** It is very important to use synthesis results of unfinished designs as a feedback mechanism – this lets you catch potential timing degradation, suboptimal design choices at an early point.

1. Synthesis: The full system is synthesized with appropriate constraints using Synopsys Design Compiler.
2. Backend: A physical layout and padframe are designed using Cadence Innovus.

## 2.2 Software Track

### Phase 1: Baseline (2-3 weeks)

There is substantial overlap with the first phase of the hardware track, so it is advisable to coordinate especially closely here.

1. Familiarize with the PULP environment and toolchain.
2. Familiarize with the the existing BNN implementation and training flow.
3. Run the BNN in a simulated system or on a real platform (GAP8), collecting performance measurements in terms of:
    a. Runtime
    b. Inference accuracy
    c. Memory footprint

### Phase 2: TNN Training (2-4 weeks)

1. Port the training code for the BNN to the newest version of QuantLab
2. Train a TNN using the same dataset, using the same network structure as the baseline BNN, recording the accuracy.
3. Model Space Exploration: make adjustments to the TNN structure and training procedure and attempt to improve the accuracy.
4. Comparison of results: Compare statistical accuracy, parameter count and computational load of the final TNN architecture to the original BNN.

### Phase 3: TNN Inference with XpulpNN (2-4 weeks)

1. Starting from either the BNN inference codebase or the PULP-NN library, implement the required ternary kernels (if not already present in PULP-NN)
2. Record performance metrics of the implementation, evaluated on representative kernels (or an uninitialized full model of the TNN established in phase 2):

   a. Computational Throughput
   b. Memory Footprint
3. Compare the results to the metrics for the BNN gathered in Phase 1. If appropriate, optimize the implementations.

### Phase 4: TNN Inference on optimized Core (3-4 weeks)

By this time, the ISA extensions should be at least specified from the HW track.

1. Extend the kernels established in Phase 3 with kernels supporting the ISA extensions.
2. Compare the optimized kernels' performance to the ones established in Phase 3
3. Map the parametrized network from Phase 2 to PULP and verify correct functionality.

## 2.3 Both Tracks

### Final Phase: Wrap-Up, Report Writing and Presentation (2-3 weeks)

1. Clean up and document the codebase so that future users can easily navigate it. **This is a crucial step and will contribute to your final grade!**
2. Compile your results and document your work in a joint report. This report should contain (but is not limited to) the following:
   a. Background, Theory & Motivation – What was the goal, and what should the reader know before reading your report?
   b. Software Implementation and Methodology: How was the TNN trained? How was the PULP software designed? Motivate your design choices where possible/appropriate!
   c. Software results: Compare relevant metrics such as inference speed, accuracy, etc. between the different implementations and discuss these results!
   d. Hardware Implementation and Methodology: What instructions did you implement, how do they work and how did you modify the core to implement them? How does the complete system look?
   e. Hardware Results: Present and discuss relevant metrics, e.g., area overhead of your extensions, impact on max. operating frequency, total area of the design, if possible simulated power consumption
   f. Hardware documentation: As your design will be taped out, it is of **critical importance** that you document it in a way that allows people not involved in this project to use it. To this end, you will write a datasheet in which you will document the platform: all the integrated peripherals, the final ISA specification, the pinout and electrical characteristics, design-for-test (DFT) specifications etc. must all be

documented. Note that the components not designed by you already have documentation which you can include in this datasheet – this step is largely a matter of extending existing documentation.

3. Presentation: You will be given the opportunity to present your work in an IIS seminar. You will be allotted 15 minutes for your presentation, followed by a 5-minute Q&A session.

# 3 Milestones

## 3.1 Hardware Track

By the end of **Phase 1** (2-3 weeks after project start) the following should be completed:

- Running example programs and tests on simulated baseline PULP system.
- Completing a synthesis of the baseline system, collecting baseline area and timing numbers.

By the end of **Phase 2** (7-9 weeks after project start) the following should be completed:

- Full formal specification of the ISA extensions
- Synthesizable, verified RI5CY core HDL with implemented extensions
- Synthesis results (timing, area) for the modified core

By the end of **Phase 3** (11-13 weeks after project start) the following should be completed:

- Layouted chip ready for tapeout in GDSII format

## 3.2 Software Track

By the end of **Phase 1** (2-3 weeks after project start) the following should be completed:

- Running the baseline BNN on GAP8 or in a simulated platform
- Reproduction of BNN accuracy and runtime numbers

By the end of **Phase 2** (4-6 weeks after project start) the following should be completed:

- Trained TNN to solve the sound event detection task
- Comparison of key metrics with baseline BNN:
    - Statistical Accuracy
    - Parameter Count
    - Number of Operations

By the end of **Phase 3** (7-9 weeks after project start) the following should be completed:

- All required kernels implemented with PULP-NN or from scratch using XpulpNN ISA extensions
- Performance metrics of the implemented kernels on selected, representative layers:
    - Compute Throughput
    - Memory Footprint

By the end of **Phase 4** (10-12 weeks after project start) the following should be completed:

- All required kernels implemented using ISA extensions developed in the hardware track
- Full comparison to results from Phases 3 and 2 for the complete network:
    - Compute Throughput
    - Memory Footprint

## 3.3 Both Tracks

By the end of the **Final Phase** (14 weeks after project start) the following should be completed:

- Finalized Documentation and Codebase
- Completed Report
- Presentation

# 4 Project Organization

## 4.1 Weekly Report and Weekly Meeting

There will be a weekly report sent by the student to Professor Benini and the supervisors at the end of every week. The main purpose of this report is to document the project's state and progress. The report, along with all other relevant documents (source code, datasheets, papers), should be uploaded regularly to the assigned shared account. The weekly meeting should be used by the student as a way to communicate any problems that have arisen during the week; also, the weekly meeting will be used to discuss the next steps.

## 4.2 Tapeout Deadline

As your design will be taped out and made into a real chip, you will have to deliver the completed design by a set deadline. Silicon foundries operate on fixed schedules which they determine months in advance. For that reason, t**his deadline is immutable and non-negotiable – if the design is not ready by then, your chip cannot be taped out!**

**The tapeout deadline is TBD**

## 4.3 Final Report

A pdf copy of the report has to be turned in and remains the property of the Integrated Systems Laboratory.

## 4.4 Final Presentation

At the end of the project, the outcome of the thesis will be presented in a 20 minutes presentation with an additional 5 minutes allotted to a question-and-answer session. The presentation will be given in the context of a group meeting of the Integrated Systems Laboratory.

# 5 Further Resources

**Todo:**

- **Link to Gianmarco's BNN training and implementation on PULP**
- **Link to PULP documentation – what's a good repository for this? Manu's video?**
- **More stuff???**

# References


[1] J. Lin, W. M. Chen, Y. Lin, J. Cohn, C. Gan, and S. Han, "MCUNet: Tiny Deep Learning on IoT Devices," *arXiv*, no. NeurIPS, pp. 1–15, 2020.

[2] A. Garofalo, M. Rusci, F. Conti, D. Rossi, and L. Benini, "Pulp-NN: Accelerating quantized neural networks on parallel ultra-low-power RISC-V processors," *Philos. Trans. R. Soc. A Math. Phys. Eng. Sci.*, vol. 378, no. 2164, 2020, doi: 10.1098/rsta.2019.0155.

[3] M. Rusci, A. Capotondi, and L. Benini, "Memory-Driven Mixed Low Precision Quantization For Enabling Deep Network Inference On Microcontrollers," 2019, [Online]. Available: http://arxiv.org/abs/1905.13082.

[4] A. Garofalo, G. Tagliavini, F. Conti, D. Rossi, and L. Benini, "XpulpNN: Accelerating Quantized Neural Networks on RISC-V Processors Through ISA Extensions," *Proc. 2020 Des. Autom. Test Eur. Conf. Exhib. DATE 2020*, pp. 186–191, 2020, doi: 10.23919/DATE48585.2020.9116529.

[5] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-net: Imagenet classification using binary convolutional neural networks," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 9908 LNCS, pp. 525–542, 2016, doi: 10.1007/978-3-319-46493-0_32.

[6] O. Muller, A. Prost-Boucle, A. Bourge, and F. Petrot, "Efficient decompression of binary encoded balanced ternary sequences," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 27, no. 8, pp. 1962–1966, 2019, doi: 10.1109/TVLSI.2019.2906678.

[7] G. Cerutti, R. Andri, L. Cavigelli, E. Farella, M. Magno, and L. Benini, "Sound event detection with binary neural networks on tightly power-constrained IoT devices," *ACM Int. Conf. Proceeding Ser.*, 2020, doi: 10.1145/3370748.3406588.