XpulpNN: Accelerating Quantized Neural Networks on RISC-V Processors Through ISA Extensions

Angelo Garofalo†, Giuseppe Tagliavini †, Francesco Conti†*, Davide Rossi† and Luca Benini†*
DEI, University of Bologna, Italy† *IIS lab, ETH Zurich, Switzerland**
{angelo.garofalo, giuseppe.tagliavini, davide.rossi}@unibo.it {fconti, lbenini}@iis.ee.ethz.ch

***Abstract*—Strongly quantized fixed-point arithmetic is considered the key direction to enable the inference of CNNs on low-power, resource-constrained edge devices. However, the deployment of highly quantized Neural Networks at the extreme edge of IoT, on fully programmable MCUs, is currently limited by the lack of support, at the Instruction Set Architecture (ISA) level, for sub-byte fixed-point data types, making it necessary to add numerous instructions for packing and unpacking data when running low-bitwidth (i.e. 2- and 4-bit) QNN kernels, creating a bottleneck for performance and energy efficiency of QNN inference. In this work we present a set of extensions to the RISC-V ISA, aimed at boosting the energy efficiency of low-bitwidth QNNs on low-power microcontroller-class cores. The microarchitecture supporting the new extensions is built on top of a RISC-V core featuring instruction set extensions targeting energy-efficient digital signal processing. To evaluate the extensions, we integrated the core into a full microcontroller system, synthesized and placed&routed in 22nm FDX technology. QNN convolution kernels, implemented on the new core, run 5.3× and 8.9× faster when considering 4- and 2-bit data operands respectively, compared to the baseline processor only supporting 8-bit SIMD instructions. With a peak of 279 GMAC/s/W, the proposed solution achieves 9× better energy efficiency compared to the baseline and two orders of magnitudes better energy efficiency compared to state-of-the-art microcontrollers.**

I. Introduction

An increasing number of Internet-of-Things (IoT) applications in several fields such as agriculture, health monitoring, surveillance, structural monitoring require to acquire data from low-power sensors, process the data, and transmit it wirelessly after extensive processing, recognition, or classification. Machine Learning (ML) algorithms, including state-of-the-art Deep Learning (DL), provide effective solutions for data processing on the end-nodes of the IoT thanks to their capability to "squeeze" raw sensor data in a much more semantically dense format (e.g., classes or extracted high-level features/symbols). Recently, there has been significant interest in deploying DL functionality on top of embedded microcontrollers (MCUs), which are the standard compute platform chosen to build extreme-edge nodes thanks to their flexible software programmability, low-cost and low-power.

This effort has to run against the severe limitations that these platforms present in terms of computing capabilities and memory footprint, which may prevent meeting latency and accuracy requirements of the target DL-enhanced applications. A recently introduced algorithmic technique to reduce both the computational cost and the memory footprint of Deep Neural Networks (DNNs) is *quantization*, i.e., the representation of network weights and activations with 8-bit or smaller data types, incurring a reduced or even negligible accuracy penalty [6], [10], [11]. The authors of [11] show that coupling per-layer quantization techniques with integer thresholding based quantization, a 4-bit MobileNetV1 achieves an accuracy loss of only 4% on Top1 accuracy with respect to the fully fixed-point precision, reducing the memory footprint by 7×. Thanks to these properties, Quantized Neural Networks (QNNs) are a natural target for execution on constrained MCU-based platforms. Efforts in this direction include the CMSIS-NN library [7] proposed by ARM for 16-bit and 8-bit QNNs on Cortex-M microcontrollers; as well as PULP-NN, an open-source library targeting RISC-V processors, and supporting heavily quantized QNNs working on 8-bit, 4-bit, 2-bit, or 1-bit data [3].

TABLE I
Overview of QNN Embedded Computing Platforms and Main Metrics

	Throughput [Gop/s]	Energy Efficiency [Gop/s/W]	Power Budget [mW]	Flexibility
ASICs [2], [9]	1K - 50K	10K - 100K	1 - 1K	Low
FPGAs [8]	10 - 200	1 - 10	1 - 1K	Medium
MCUs [3]	0.1 - 2	1 - 50	1 - 1K	High
This Work	1 - 5	80 - 550	1 - 100	High

An inherent limitation of current-generation MCUs is that their ISAs lack support for low-bitwidth Single Instruction Multiple Data (SIMD) arithmetic instructions; typically, only 16-bit (e.g., ARMv7E-M) or 8-bit (e.g., RV32IMCXpulpv2 [4]) are supported. This means that in these platforms quantization is effective as a memory compression technique [11] but not as a way to save time and energy in computation; rather, it leads to non-negligible overhead [3], [12] as low-precision data have to be unpacked and extended to data types natively supported by the underlying hardware. In this work, we attack this limitation by proposing an instruction set extension to the RISC-V ISA targeting specifically the requirements of QNN inference, with support for low-bitwidth operations (8-bit, 4-bit, 2-bit). The approach we propose enables the execution of low-memory footprint and high-energy efficient (up to 550 Gop/s/W) QNN at the edge of the IoT, in a full software-programmable environment.

The main contributions of this paper are the following:

- We define an ISA extension to the RISC-V ISA targeting software programmable QNN inference, namely *XpulpNN*, implementing low-bitwidth SIMD arithmetic instructions, and other domain-specific instructions to boost the performance of the quantization process [6];
- We integrate the proposed extensions on the register transfer level (RTL) description and the GCC toolchain of an open-source RISC-V processor (RI5CY [4]) featuring extensions targeting energy-efficient digital signal processing;
- We implement (i.e., full layout) a full microcontroller system (based on the open-source PULPissimo [13]) integrating the proposed processor in a commercial 22nm

978-3-9819263-4-7/DATE20/©2020 EDAA

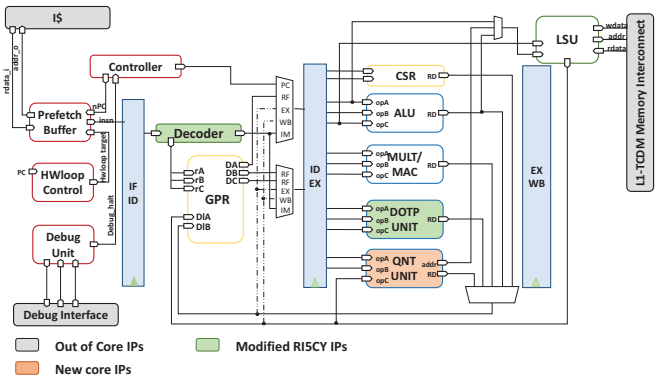

Fig. 1. Baseline and extended RI5CY pipeline.

FDX technology, to evaluate its overheads with respect to the baseline RI5CY processor. The results show that the extended processor features 11.1% area overhead, 5.9% power overhead, and negligible timing overhead compared to the original one;

- We compare the extended processor with state-of-the-art architectures and software, by running quantized convolutional layers on both commercial architecture and the baseline RI5CY core, outperforming by up to 19.3× in performance and by up to 354× in energy efficiency existing systems leveraging ARMv7E-M and XpulpV2 ISAs.

To put our results into perspective, Table I provides a summary of embedded computing platforms for QNNs targeting an operating power below 1 W with our proposed work. ASICs [1], [2], [9] can achieve up to a few Top/s/W, at the cost of inflexible execution models, and high cost for devices that have to be cheap to be economically feasible at the extreme edge of the IoT. On the other hand, edge-targeted FPGA platform such as Lattice SenseAI [8] can be reconfigured using hardware design languages or high-level synthesis. The productivity of using these solutions is not as high as that of embedded MCU programming, but they anyways guarantee higher flexibility than ASICs. However, FPGAs are much less efficient than ASICs, with typical figures [5] ranging between 1 and 10 Gop/s/W. Finally, current-generation commercial MCUs are not fast enough to target complex DNNs, and, as previously discussed, they cannot scale up their performance with quantization. Therefore, the order-of-magnitude improvements that we achieve compared to the state-of-the-art on MCUs demonstrate for the first time that software programmable edge inference of QNN models at ASICs-like efficiency is indeed possible on MCUs by mixing architectural and micro-architectural design with leading-edge near-threshold FD-SOI technology.

II. Background

1) RI5CY core: The RI5CY core, which we extend to support the proposed ISA extensions, is a 4 stage in-order single-issue pipeline, supporting the RV32IMC instruction set, plus extensions targeting energy-efficient digital signal processing (*XpulpV2*) [4]. The *XpulpV2* extensions include hardware loops, auto-incrementing load/store operations, bit manipulation instructions, fixed-point and 8-bit and 16-bit packed single-instruction-multiple-data (SIMD) operations.

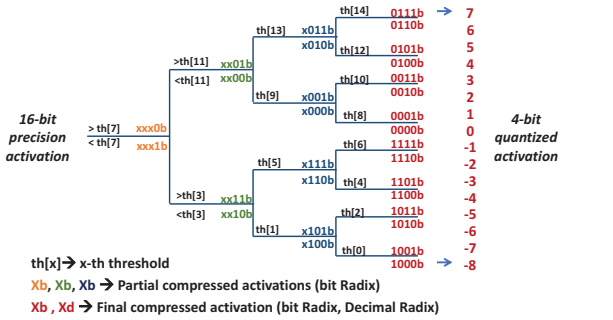

Fig. 2. Binary tree implementation of the staircase compression function for 4-bit operands and iterative construction of the result.

2) QNN Execution Model: The execution model we rely on for QNN inference on MCUs is the one proposed by ARM [7] and also adopted in [3] for RISC-V processors. Efficient execution of the convolution on MCUs requires two phases: the *im2col* step arranges the 3D input features of the current convolution into a 1D vector, while the dot product step is implemented as a Matrix Multiplication (MatMul) [7]. The MatMul kernel fetches from memory the weights from two consecutive filters and the input activations from two different *im2col* buffers. The reuse of the loaded buffers enables the computation of two activation outputs related to two consecutive channels in the same inner loop of the MatMul. While for 8-bit operands scaling and clamp operations are used for compression, to compress back the higher-precision result (16-bit) from the MatMul into 4- or 2-bits, an effective procedure consists of using a thresholding-based compression, described as a staircase function [6], [11]. This operation compares an input with a set of offline thresholds, which absorb bias and batch normalization, and the result determines the output quantized value. The optimal algorithm to implement the staircase function makes use of a balanced binary tree where a 16-bit comparison takes place at every node. The complexity of the algorithm is $O(n)$, where n is the number of bits in the result. Figure 2 depicts the 4-bit case, and we also show how it is possible to compute the output bits incrementally based on the result of the comparisons. Each convolution layer requires $2^Q - 1$ threshold values per channel to produce a Q-bit output. Starting from the binary-tree algorithm, we have designed a dedicated ISA instruction to speed up the compression phase.

III. RISC-V ISA Extensions

A. XpulpNN ISA Extension

The RISC-V ISA that we use as a baseline already supports an extension (*XpulpV2*) providing 8-bit and 16-bit SIMD instructions in three addressing variations: the first variation uses two registers (*pv.instr.*$\{b,h\}$), the second uses an immediate value (*pv.instr.sci.*$\{b,h\}$), and the third replicates the scalar value in a register as the second operand for the vectorial operation (*pv.instr.sc.*$\{b,h\}$).

The proposed *XpulpNN* instructions extend the RV32IMCXpulpV2 ISA with SIMD operations for 4-bit and 2-bit operands, namely *nibble* (indicated with *n*) and *crumb* (indicated as *c*) respectively, to accelerate the low-bitwidth QNN kernels. Because of the limited room available in the encoding space of RV32IMCXpulpV2 ISA, we can not address all the addressing variations mentioned above for sub-byte vector types. Consequently, we have selected only two formats, leaving aside the one which uses an immediate

TABLE II
OVERVIEW OF *XpulpNN* INSTRUCTIONS FOR *nibble* (4-BIT) VECOTR OPERANDS. THE DESCRIPTION CAN BE EXTENDED TO *crumb* (2-BIT). i IN THE TABLE REFERS TO THE INDEX IN THE VECTOR OPERAND, I.E. $i \in [0;7]$ IN THE *nibble* CASE.

ALU SIMD Op.	**Description for *nibble***
pv.add[.sc].{n, c}	rD[i] = rs1[i] + rs2[i]
pv.sub[.sc].{n, c}	rD[i] = rs1[i] - rs2[i]
pv.avg(u)[.sc].{n, c}	rD[i] = (rs1[i] + rs2[i])>>1
Vector Comparison Op.	
pv.max(u)[.sc].{n, c}	rD[i] = rs1[i] > rs2[i] ? rs1[i] : rs2[i]
pv.min(u)[.sc].{n, c}	rD[i] = rs1[i] < rs2[i] ? rs1[i] : rs2[i]
Vector Shift Op.	
pv.srl[.sc].{n, c}	rD[i] = rs1[i] >> rs2[i] Shift is logical
pv.sra[.sc].{n, c}	rD[i] = rs1[i] >> rs2[i] Shift is arithmetic
pv.sll[.sc].{n, c}	rD[i] = rs1[i] << rs2[i]
Vector abs Op.	
pv.abs.{n, c}	rD[i] = rs1[i] < 0 ? -rs1[i] : rs1[i]
Dot Product Op.	
pv.dotup[.sc].{n, c}	rD = rs1[0]*rs2[0] + ... + rs1[7]*rs2[7]
pv.dotusp[.sc].{n, c}	rD = rs1[0]*rs2[0] + ... + rs1[7]*rs2[7]
pv.dotsp[.sc].{n, c}	rD = rs1[0]*rs2[0] + ... + rs1[7]*rs2[7]
pv.sdotup[.sc].{n, c}	rD = rs1[0]*rs2[0] + ... + rs1[7]*rs2[7] + rD
pv.sdotusp[.sc].{n, c}	rD = rs1[0]*rs2[0] + ... + rs1[7]*rs2[7] + rD
pv.sdotsp[.sc].{n, c}	rD = rs1[0]*rs2[0] + ... + rs1[7]*rs2[7] + rD
Quantization Op.	
pv.qnt.{n, c}	see Section III-B2

value (i.e., *pv.instr.sci.*{*n,c*}). We argue that such a choice is not a concern for QNN applications, as immediate values can be preventively stored in a register, without the addition of significant overhead to the execution.

The core of the *XpulpNN* extensions consists of dot product operations, performed in a SIMD-like manner on packed vectors of 4- or 2-bit elements. We support different variants which are commonly used in QNN inference, providing different interpretations of the input registers; the dotp instruction can be executed interpreting one of both operands as signed or unsigned vectors. The same idea also applies to the third input register (scalar) of the sum of dot product instructions. Moreover, we support SIMD maximum, minimum, and average instructions, which are useful to speed up the average/maximum pooling QNN layers, as well as the ReLu activation function.

As the QNN MatMul kernel generates higher-bitwidth results (16-bit for sub-byte kernels), a quantization step is needed to bring back the value into the target bitwidth. Adopting the QNN execution model presented in Section II-2, this step requires a staircase function which compares the intermediate output with pre-trained thresholds to obtain the quantized activation. Its implementation through a balanced binary tree is valuable from an algorithmic point of view, but it is still inefficient when executed on a tiny microcontroller. Many cycles would be wasted due to the high number of branch instructions that occur at each node of the tree, where the comparison takes place, and their negative impact on the overall performance of convolution or linear layers becomes not negligible. To highly speed up this phase, we design a multicycle instruction, namely *pv.qnt.*{*n,c*}, which handles the quantization process in hardware. This instruction takes as input two 32-bit registers, the first storing two 16-bit activations and the second the entry address of the binary tree (the address for the second activation derives from the first one, as explained in Section III-B2); as output, it computes two quantized activations in parallel, which are subsequently stored into a single 32-bit register. The instruction latency is only 9 cycles in the 4-bit case, favorably comparing to

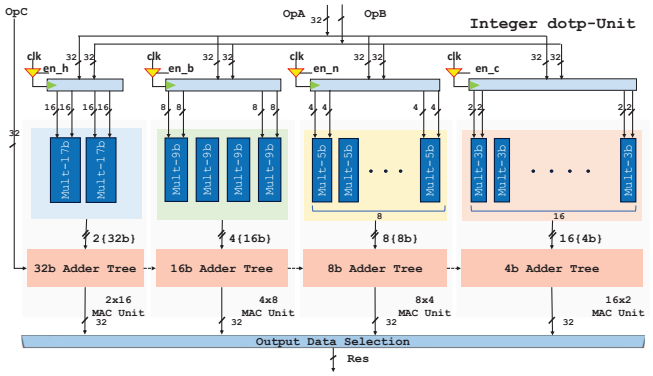

Fig. 3. Simplified block diagram of the Dot-Product Unit, including the two groups of multipliers added to support *nibble* and *crumb* operations, as well as the clock gating design to reduce the operand switching activity.

the 18 clock cycles needed on average to compress only one activation if implementing the binary tree in software. A detailed description of the instruction, as well as the hardware design to enable it, is provided in Section III-B2. To comply with RV32IMCXpulpV2 ISA, we also extend the support for *nibble* and *crumb* to ALU SIMD instructions and shifting operations.

B. Microarchitecture

In this subsection, we detail the microarchitectural design to support the ISA extensions presented above.

1) Dot-Product Unit: The baseline dot-product (*dotp*) unit available in the RI5CY core consists of two sets of multipliers, supporting 16- and 8-bit vector operands [4]. We extend the architecture, reported in Figure 3, to support also 2- and 4-bit vector operands. The proposed multipliers compute the dotp between two vectors, each containing either eight 4-bit or sixteen 2-bit elements, and accumulate the result in a 32-bit register through an adder tree, in one clock cycle. The MAC equivalent is the sum-of-dot-product (*sdotp*) operation, implemented with an additional 32-bit accumulation input at the adder tree. Both these operations are supported interpreting the operands as signed or unsigned, or the first signed and the second unsigned. To perform signed and unsigned multiplications, the 4- or 2-bit inputs are sign or zero extended; therefore, each element is a 5- or 2-bit operand, as shown in Figure 3.

The design of the extended dot-product unit aims at minimizing the impact of its longest path, making it shorter or equal to the critical path of the overall system; in our case, this is the path from the processor core to the memories and vice versa. The 4- and 2-bit vector dotp operations are near to be timing critical since the amount of logic required to sum up the partial products is higher than the 8-bit and 16-bit cases. For this reason, sharing the multiplication resources among the different bitwidth "regions" would be detrimental from a timing point of view, as the additional circuitry to select, split and distribute the operands and to enable the selected bitwidth SIMD operation would increase the impact on the overall operation speed. The main drawback of not sharing resources is in terms of area since we replicate a set of different bitwidth multipliers four times. However, some area could be saved by sharing the adder trees of the four multipliers. Also this solution, though, makes the unit slower, which is why we design the extended dotp-unit by adding two additional sets of multipliers, each equipped with a dedicated adder tree,

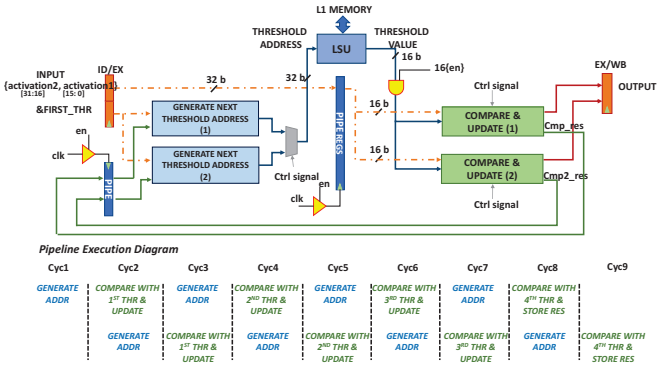

Fig. 4. Simplified block diagram of the Quantization Unit and execution timing diagram of the *pv.qnt.c*.

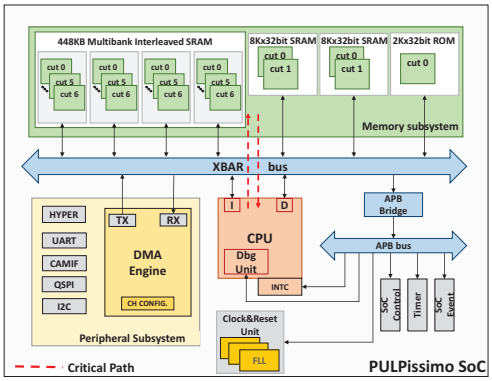

Fig. 5. Overview of the PULPissimo SoC architecture.

as shown in Figure 3. To mitigate the effects of hardware duplication on the power consumption of the core, we add a set of registers on the inputs of each bitwidth region, and we perform clock gating to avoid the switching of operands not involved in current SIMD operation.

Despite the area overhead of 19.9% with respect to the starting dotp-unit, this design choice avoids increasing the critical path, avoiding to look at strategies that would consider pipeline registers in between the multiplication and the accumulation phases. Including a pipeline would result in additional stalls when computing back-to-back dotp operations, resulting in a considerable overhead for QNN workload, where most of the computation consists of sdotp-based operations.

2) Quantization Unit: To provide the hardware support for the *pv.qnt* instruction, we extend the ex-stage of the RI5CY core with a dedicated hardware block, the quantization unit, pictured in Figure 4. The unit quantizes higher-bitwidth inputs (16-bit) into a *crumb* or *nibble* data type, through a thresholding-based compression, described so far in Section II-2. The hardware block consists of two regions, the first compares the threshold with the input (green blocks in the Figure) and the second, accordingly to the comparison result, updates the address for the next memory access (blue blocks). The quantized result is computed iteratively following the algorithm introduced in Figure 2, by inserting a result bit in an intermediate register and shifting it until the last comparison.

The critical path of the system is from the processor core to memories and vice versa; this complicates the design of the quantization unit since it has to fetch data from memory at specific addresses. In the initial design, this unit receives the value of two registers from the ID/EX pipeline, the first containing the 16-bit operand to be quantized and the second the entry point of the quantization thresholds. The two regions are cascaded without any pipeline stage in the middle; consequently, the two phases of the operation, including the threshold comparison with input data and the address update required to fetch the next threshold, occur in a combinatorial manner. After an initialization cycle to fetch the first threshold, a comparison takes place at each next step, and the address of the next threshold is generated accordingly to the result; the execution takes in total 5 cycles (3 cycles) on average (if no memory stalls occur) to obtain the 4-bit (2-bit) quantized result. Our analysis though, shows that this design increases the critical path of the system by 90%.

To meet the timing requirements of the system, the two blocks of the quantization unit must be pipelined, interleaving the comparison and the address update phases. The drawback of a pipeline execution is that it increases the output latency by almost a factor of two. However, by exploiting the 16-bit bitwidth results of the matrix multiplication, we can pack two 16-bit partial results into a single 32-bit register, passed as input to the quantization unit, to perform the quantization of two activations in an interleaved manner. To this purpose, we double the hardware resources of the quantization unit, and we split the datapath to host two half-word activations to be quantized, as visible from Figure 4. As the thresholds are stored in memory at consecutive addresses, we can compute the address of the entry point of the quantization thresholds for the second activation by adding a hard-wired fixed offset to the first one, with no need of an additional source operand to the *pv.qnt* instruction. Moreover, assuming the threshold vectors aligned in memory without loss of generality, our exploration shows that only 6-bits are needed to update the next memory address from the previous one. This remark reduces the logic in the address update block, furthermore saving timing and power resources. For the same reasons, we perform operand isolation on the comparator inputs, as depicted in Figure 4, to avoid unnecessary switching activities for this block, which is directly connected to the input of the Load Store Unit.

The pipeline execution is orchestrated by a 3-bit Finite State Machine, sending proper control signals to each sub-unit to interleave the comparison and the address update phases and to handle the eventual memory stalls, rarely happening since the quantization unit, when enabled, takes the whole control of the execution, stalling the core pipeline until the quantized result is ready. The only cause of memory stalls concerns misaligned accesses to the memory. The proposed design allows us to keep the critical path of the system unaltered and to compute two 4-bit (2-bit) quantized activations in 9 clock cycles (5 clock cycles).

IV. Results and Discussion

To perform power, performance and area evaluations, we implement the original RI5CY core and the extended version in a full open-source microcontroller system, namely PULPissimo, featuring a μDMA with a full set of peripherals and 512 kB of SRAM memory. Figure 5 provides an overview of the PULPissimo architecture. We synthesize the two platforms with Synopsys Design Compiler-2016.03, and we perform complete place & route flow using Cadence Innovus-15.20.100 in a 22nm FDX technology using the worst-case corner (SS, 0.59 V, -40 °C / 125 °C). To compare the average power con-

TABLE III
AREA AND POWER CONSUMPTION.

Area [μm^2] (Overhead vs. baseline [%])			
	RI5CY [4]	**Ext. RI5CY No Pow. Manag.**	**Ext. RI5CY Pow. Manag.**
Total	19729.9	21424.9 (8.59%)	21912.8 (11.1%)
dotp-Unit	5708.9	6755.8 (18.3%)	6844.4 (19.9%)
ID Stage	6363.1	6530.2 (1%)	6677.8 (5%)
EX Stage	9500.9	11129.1 (17.1%)	11251.6 (18.4%)
LSU	518.0	610.8 (17.9%)	591.2 (14.1%)
Core Power Consumption on 8-bit MatMul at 0.75V, 250MHz [mW]			
Leak. Power	0.023	0.032	0.031
Dyn. Power	1.13	1.38	1.19
Tot. Power	1.15	1.41	1.22
Overhead [%]	–	22.5%	5.9%
PM Savings [%]	–	–	13.5%
PULPissimo SoC Total Power Consumption at 0.75V, 250MHz [mW]			
8-bit MatMul	5.93	6.28 (5.8%)	6.04 (1.8%)
4-bit MatMul	–	8.14	5.71
2-bit MatMul	–	8.99	5.87
GP application	5.65	8.20 (45.2%)	5.85 (3.5%)

sumption, the performance and the energy efficiency, we implement a set of convolution layers operating on a 16×16×32 input tensor with a filter size of 64×3×3×32, characterized by different bitwidth (8-, 4-, 2-bits), extending the kernels described in [3] with *XpulpNN*. As sub-byte data types are not available in compilation toolchains, we have extended the open-source GCC compiler targeting PULPissimo with the *machine description* of the new instructions and a set of related built-in functions, which provide an interface based on integer values; this approach enables efficient use of the 32-bit hardware registers while ensuring a higher optimization level of the binary code compared to inline assembly.

A. Implementation Setup Results

The total area of the extended core is 0.022 mm^2, with an overhead of 11.1% with respect to the RI5CY core, due to the extended dotp unit and the addition of the quantization unit in the ex-stage of the core, while the PULPissimo SoC area is 0.998 mm^2 with the new core.

To provide an accurate power consumption estimation of the two PULPissimo SoC, implementing the RI5CY and the extended core respectively, and to characterize the whole system-level power consumption of both original 8-bit integer sdotp operation and the new sub-byte (4- and 2-bit kernels) instructions, we conduce post-place-&-route power simulation in the typical corner (TT, 0.65 V, 25 °C). To this end, the Value Change Dump (VCD) traces of the systems executing the various instructions are generated with Mentor Modelsim 10.6c and passed to Synopsys Prime Time-2016.03 to extract the power numbers. The core power estimation is performed at 250 MHz, running an 8-bit MatMul kernel on the RI5CY and on the extended core. To assess the benefits of clock gating and operand isolation design, the latter core is synthesized in two versions, with and without the power management. As visible in Table III, the extended core without power management features a not negligible power overhead of 22.5%. Performing operand isolation on critical operands highly reduces their switching activity, achieving a total power consumption only 5.9% higher than the RI5CY baseline. The improvements given by the power management are clearly visible also at the system level, where a general-purpose application, consisting of a mix of load/store, control and arithmetic operations, runs in the same power envelope on the PULPissimo Soc implementing the RI5CY core and the new one, not jeopardizing the efficiency of the core on general-purpose benchmarks.

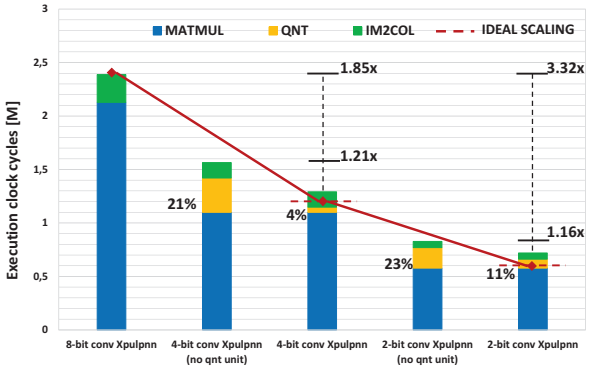

Fig. 6. Linear scaling of sub-byte kernel performance with respect to 8-bit one and impact of the *pv.qnt.*{*n,c*} on the 4- and 2-bit kernel execution cycles.

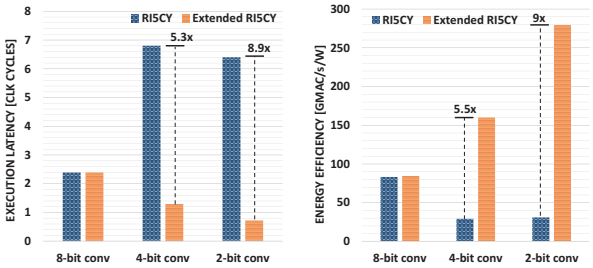

Fig. 7. Execution latency and energy efficiency of convolution kernels run on RI5CY and the extended RI5CY cores.

B. Benchmarking

To evaluate the performance and the energy efficiency gain of the proposed RISC-V ISA extensions, we benchmark the proposed core on a set of different bitwidth convolution layers. To justify the need of a dedicated instruction to speed up the quantization phase (i.e., *pv.qnt*{*n,c*}), we implement two variants of sub-byte convolution kernels, the first performing the quantization in software, the second fully exploiting the *XpulpNN* ISA extensions. The results of their execution on the extended RI5CY core, in terms of clock cycles, are reported in Figure 6. The adoption of the *pv.qnt* instruction reduces the impact of quantization on the total number of execution cycles down to 4% for 4-bit and 11% for 2-bit, overall reducing the execution cycles of 4- and 2-bit kernels by 1.21× and 1.16× with respect to the ones using a software quantization. The results also highlight that, with the *XpulpNN* extensions, the performance of sub-byte kernels scales almost linearly with respect to the 8-bit kernel execution.

As a further evaluation we compare the execution of 8-, 4-, and 2-bit convolution kernels, in terms of execution cycles, on the RI5CY core, using the *XpulpV2* ISA, and on the extended one, implementing the *XpulpNN* extensions. To run the sub-byte kernels on RI5CY, additional instructions to unpack and pack the low-bitwidth operands must be included in the code [3], as the lowest data type supported at ISA level with SIMD instructions is 8-bit. Due to fully ISA support for *nibble* and *crumb* SIMD operations, on the new core the sub-byte kernels run 5.3× (4-bit convolution) and 8.9× (2-bit convolution) faster than the ones implemented on RI5CY, where the additional packing/unpacking functions introduce a high overhead. Then, relying on the power estimations of the two PULPissimo reported in Table III, we also compare our core with the baseline in terms of energy efficiency. The results in Figure 7 show that our proposed power-aware core design

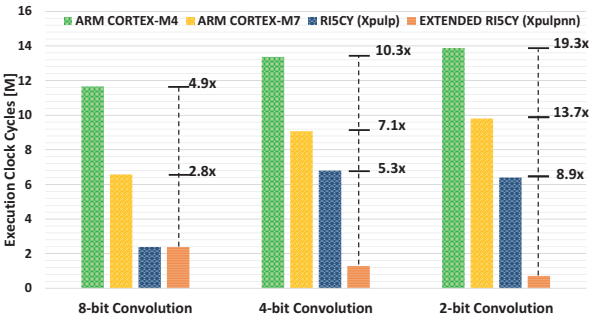

Fig. 8. Comparison, in terms of execution cycles, of the extended RI5CY core with state-of-the-art architectures when running convolution kernels.

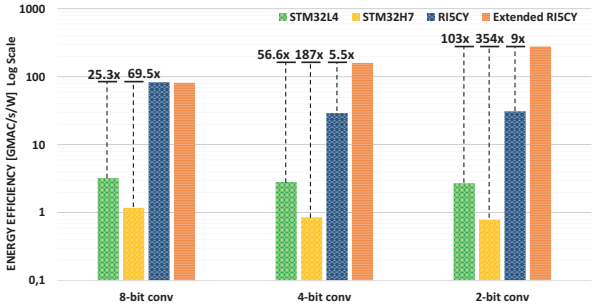

Fig. 9. Energy efficiency comparison on different bitwidth convolution kernels.

highly increases the energy efficiency of sub-byte kernels up to 9× without reducing the efficiency for 8-bit QNN kernels. These achievements demonstrate the effectiveness of extending the ISA support to low-bitwidth operands to obtain high performance and high energy efficiency on highly-quantized QNN kernels. To compare with state-of-the-art architectures and software, we run the sub-byte kernels on off-the-shelf STM32H743 [14] and STM32L476 [15] commercial microcontrollers based on ARM CORTEX-M7 and CORTEX-M4 cores respectively, using the extended CMSIS-NN [12] library. Figure 8 shows the execution latency, in clock cycles, of 8-, 4- and 2-bit kernels running on our proposed core, RI5CY, STM32L4 and STM32H7, while Figure 9 shows the energy efficiency. In terms of performance, with the proposed ISA extensions and the micro-architectural design to support them, we improve the execution of sub-byte kernels by one order of magnitude with respect to the execution on STM32L4 and STM32L7. Moreover, coupling the proposed power-aware micro-architectural design with leading-edge near-threshold FD-SOI technology, we achieve energy efficiency which is two orders of magnitudes higher with respect to the state-of-the-art microcontrollers, 103× and 354× better than STM32L4 and STM32H7 respectively, on the 2-bit kernel.

V. Conclusion

In this work we have presented *XpulpNN*, a set of RISC-V ISA extensions to accelerate low-precision QNN models on embedded microcontrollers. The proposed ISA extensions, which include SIMD operations for nibble and crumb operands, as well as a dedicated instruction to speed up the quantization process, have been integrated into the micro-architecture of an open-source RISC-V core (RI5CY). To evaluate the proposed extensions we have integrated the extended core into an open-source MCU platform (PULPissimo), and synthesized and place&routed it into a commercial 22nm FDX technology, at the operating frequency of 250 MHz and the supply voltage of 0.65 V. We have shown that from a physical implementation viewpoint, the extended core features a negligible overhead in timing, an overhead of 11.1% in area, an overhead of 5.9% in power (when executing an 8-bit convolutional kernel), compared to the original baseline, due to the additional hardware required to implement the instructions. It should be noted that, at the system level, the proposed extension features a power overhead of only 1.8% with respect to the original RI5CY, not jeopardizing the general-purpose nature of the extended core. On the other hand, providing a speedup ranging from 5.3× and 8.9× for execution of QNN convolutional layers, it boosts the energy efficiency of QNN layers inference by 5.5× to 9× with respect to the baseline, and by more than two orders of magnitude when compared to commercial microcontrollers based on ARM architectures such as STM32L4 and STM32H7, paving the way to software programmable QNN inference at the edge of the IoT at ASIC-like efficiency.

References

[1] R. Andri, L. Cavigelli, D. Rossi, and L. Benini. YodaNN: An architecture for ultralow power binary-weight CNN acceleration. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 37(1):48–60, 2018.

[2] G. Desoli, N. Chawla, T. Boesch, S.-p. Singh, et al. 14.1 A 2.9 TOPS/W deep convolutional neural network SoC in FD-SOI 28nm for intelligent embedded systems. In *2017 IEEE Int. Solid-State Circuits Conference (ISSCC)*, pages 238–239. IEEE, 2017.

[3] 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. *arXiv preprint arXiv:1908.11263*, 2019.

[4] M. Gautschi, P. D. Schiavone, A. Traber, I. Loi, et al. Near-threshold RISC-V core with DSP extensions for scalable IoT endpoint devices. *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, 25(10):2700–2713, 2017.

[5] V. Gokhale, A. Zaidy, A. X. M. Chang, and E. Culurciello. Snowflake: An efficient hardware accelerator for convolutional neural networks. In *2017 IEEE Int. Symp. on Circuits and Systems (ISCAS)*, pages 1–4. IEEE, 2017.

[6] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. *The Journal of Machine Learning Research*, 18(1):6869–6898, 2017.

[7] L. Lai, N. Suda, and V. Chandra. Cmsis-nn: Efficient neural network kernels for arm cortex-m cpus. *arXiv preprint arXiv:1801.06601*, 2018.

[8] Lattice. Lattice sensAI Delivers 10X Performance Boost for Low-Power, Smart IoT Devices at the Edge. https://www.latticesemi.com/About/Newsroom/PressReleases/2019/201911sensAI. Last accessed on Sept. 19.

[9] J. Lee, C. Kim, S. Kang, D. Shin, et al. UNPU: A 50.6TOPS/W unified deep neural network accelerator with 1b-to-16b fully-variable weight bit-precision. In *2018 IEEE Int. Solid-State Circuits Conference-(ISSCC)*, pages 218–220. IEEE, 2018.

[10] B. Moons, K. Goetschalckx, N. Van Berckelaer, and M. Verhelst. Minimum energy quantized neural networks. In *51st Asilomar Conference on Signals, Systems, and Computers*, pages 1921–1925. IEEE, 2017.

[11] M. Rusci, A. Capotondi, and L. Benini. Memory-Driven Mixed Low Precision Quantization For Enabling Deep Network Inference On Microcontrollers. *arXiv preprint arXiv:1905.13082*, 2019.

[12] M. Rusci, A. Capotondi, F. Conti, and L. Benini. Work-in-progress: Quantized nns as the definitive solution for inference on low-power arm mcus? In *2018 Int. Conference on Hardware/Software Codesign and System Synthesis (CODES+ ISSS)*, pages 1–2. IEEE, 2018.

[13] P. D. Schiavone, D. Rossi, A. Pullini, A. Di Mauro, et al. Quentin: an Ultra-Low-Power PULPissimo SoC in 22nm FDX. In *2018 IEEE SOI-3D-Subthreshold Microelectronics Technology Unified Conference (S3S)*, pages 1–3, Oct 2018.

[14] STMicroelectronics. Stm32h743xl datasheet. https://www.st.com/resource/en/datasheet/stm32h743bi.pdf. Last accessed on Sept. 19.

[15] Stmicroelectronics. Stm32l476xx datasheet. https://www.st.com/resource/en/datasheet/stm32l476je.pdf. Last accessed on Sept. 19.