Beaivi: A 22-nm 1-GHz+ Exposed Datapath RISC-V DSP for Low-Power Applications

Kari Hepola, Joonas Multanen, Väinö-Waltteri Granat, Jakub Žádník, Roope Keskinen,
Karri Palovuori, Pekka Jääskeläinen
Tampere University, Finland

***Abstract*—Low-power digital signal processing is required for edge devices operating in energy-constrained environments. Static multi-issue machines excel in such use cases but lack the required flexibility for maintaining high code density while exploiting instruction-level parallelism. This paper introduces a novel RISC-V-based DSP architecture, "Beaivi", that extends the processor with an exposed datapath multi-issue mode for exploiting instruction-level parallelism efficiently in performance-critical code regions while preserving high code density in noncritical phases with a RISC-V mode. The dynamic code density is further improved by leveraging a dictionary compression method that programs the dictionaries on a loop basis via compiler-driven static analysis. We demonstrate the real-world applicability of the architecture by taping out the processor using a commercial 22-nm technology. The design meets timing at 1.0 GHz and draws 50 mW under a neural network inference workload.**

I. Introduction

On-going trends in computing are shifting towards higher utilization of custom hardware to accelerate target workloads, while maintaining a low energy budget for systems such as edge devices. Owing to this shift in design philosophy, *very long instruction word* (VLIW) style architectures have gained renewed interest in application domains such as computer vision and machine learning acceleration [1]–[4], in addition to traditional DSP tasks. Contrary to RISC-style machines, VLIWs suffer from poor code density, particularly in control-oriented code that does not fully utilize the multi-issue capabilities. This work proposes combining RISC-V and exposed datapath instruction sets in a way that achieves the benefits of efficient static multi-issue execution with a RISC-level code density. The flexible architecture preserves compiler-supported programmability with static code analysis automatically choosing the best mode based on the program phase.

The proposed Beaivi DSP is tailored for digital signal processing tasks via a custom instruction set extension that partially integrates with the RISC-V programming interface. As a result, the design maintains a simple hardware implementation while enabling ternary instructions through the exposed datapath mode that supports flexible addressing modes, eliminating the need for costly datapath extensions. To demonstrate the feasibility of the complex architecture in real-world applications, we fabricate the processor as a part of a multiprocessor SoC using a 22-nm ASIC technology.

The proposed DSP has the following novel aspects:

- A RISC-V-based architecture with three compiler-programmable instruction set modes: 1) A pure RISC-V, 2) exposed datapath VLIW, and 3) dynamically programmable dictionary-compressed VLIW mode, for area- and energy-efficient exploitation of *instruction-level parallelism* (ILP) with high static and dynamic code density.
- A method for implementing three-register-input instructions to a RISC-V processor without increasing the number of register file ports or data hazard logic, thus preserving an area- and power-efficient datapath.
- Integration of a custom instruction set extension into the architecture modes to enhance performance and efficiency for widely adopted digital signal processing workloads.
- Silicon-proven: The DSP has been fabricated using a commercial 22-nm ASIC process, with performance measured from the fabricated chip.

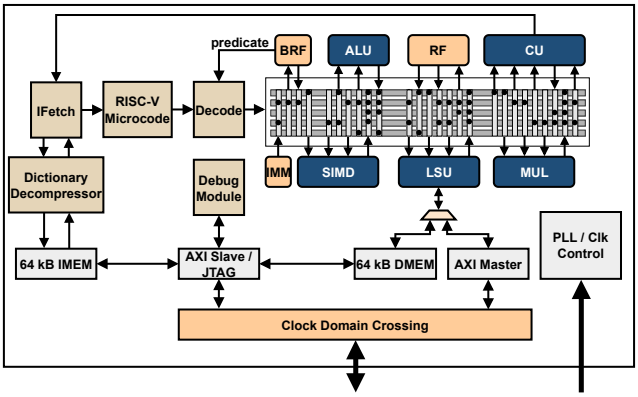


Fig. 1: Beaivi DSP including the pipeline, datapath, local memories, and external interfaces.

II. Architecture

Figure 1 illustrates the DSP subsystem with the exposed datapath pipeline coupled with RISC-V microcode and a dictionary decompressor in the core's control path. The processor is designed around a 32-bit dual-issue pipeline interfacing with a 64-bit instruction word.

A. Transport Triggered Architecture

Transport triggered architecture (TTA) [5] is an advanced variation of a VLIW-style processor. It replaces the traditional operation triggered VLIW with an "exposed datapath" programming model whose goal is to mitigate ILP scaling issues [6] via low-level code optimizations, such as *software bypassing* [7] and *operand sharing* [8]. These techniques reduce the register file pressure, which leads to a more resource-efficient datapath together with reduced control logic thanks to

978-3-9826741-1-7/DATE26/© 2026 EDAA

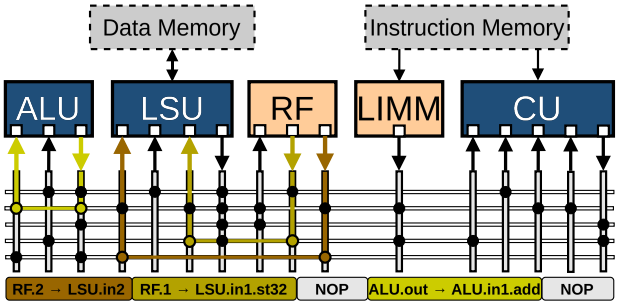


Fig. 2: The TTA programming model is based on move instructions that explicitly transport operands on the datapath.

the ultra-static programming interface. Figure 2 illustrates an example of the move-based programming model that executes operations as a side effect of the operand moves. Exposed datapath processors excel in regions with a high amount of instruction-level parallelism. However, in serial code sections, where the multi-issue capabilities are underutilized, they suffer from low code density and require NOPs to handle control and data dependencies, similar to traditional VLIW processors. Exposed datapath processors utilize more encoding space due to the programmer-visible datapath, which can further decrease code density compared to traditional architectures [9].

B. Instruction Set Modality

This work limits exposing the datapath only to performance-critical code regions that benefit from it, while utilizing a RISC-style programming interface for other program phases. With this method, it is possible to achieve high static code density with similar performance to a native exposed datapath architecture. Utilizing the exposed datapath programming model in performance-critical code poses the challenge of dynamic code density, since these regions form hot spots in the program execution, contributing a major portion of the amount of fetched instruction bits. From this point of view, it becomes necessary to utilize compression for the exposed datapath instructions in order to lower the energy footprint of the instruction stream. We utilize run-time programmable dictionaries [10] that are programmed at loop granularity using static compiler-based analysis. The method allows mixing compressed and uncompressed instructions by packing the former into a bundle, whose size matches the size of the uncompressed instruction template. This principle introduces a third instruction mode, where the instruction bits do not define move instructions but indexes pointing to a dictionary for each move slot.

C. Instruction Set Extensions

The processor includes a custom instruction set extension to accelerate DSP workloads, such as fixed-point computation. The extension also adds instructions with saturation mechanics for preventing overflows. The benefit of such *application-specific instruction-set processor* (ASIP) [11] -based approach is high flexibility, which makes it possible to execute different workloads using the same custom hardware. Table I describes the domain-specific custom instructions, with the RISC-V mode only supporting operations with two inputs, since these easily map to the R-format. The exposed datapath mode has a higher flexibility in terms of addressing modes, which makes it possible to implement the ternary instructions without increasing the register file complexity and still utilize them efficiently in the exposed datapath mode via software bypassing and operand sharing. In addition, we add packed SIMD variants for common ALU operations and a 32b multiply-accumulate operation.

III. RISC-V Microcode

The low-level programming interface of the transport triggered architecture makes it possible to use it as a superset for different architecture styles. We leverage this property by adopting a multi-instruction-set architecture method [12] that introduces an additional decode step into the pipeline, which enables to lower RISC-style instructions into TTA moves in hardware during run time. The microcode can be implemented as a lightweight abstraction on top of the existing exposed datapath pipeline, since it essentially adds the RISC-specific decode and control logic, while sharing the other parts of the control, decode and datapath hardware.

Figure 3 illustrates the microcode implementation. At its core, the microcode unit utilizes design-time generated hard-coded *lookup tables* (LUTs) that translate fetched RISC-V instructions into TTA move encodings that control the datapath multiplexers and opcode signals. As exposed datapath processors use software bypassing, the hardware must dynamically translate and sequence the matching bypass moves for each operand from their own look up tables, while keeping track of the destination function unit of the previous instruction.

In order to keep the overhead of the RISC-V-specific hardware low, we adopt a simple in-order pipeline. Following this approach, we stall the pipeline during the execution of multi-cycle operations by reading an operation latency value from a lookup table and setting a stall counter based on the read value. While this lowers the performance of the pipeline, it can be justified by the use case of the RISC-V mode, which does not target performance-critical code sections and instead is used as a means for increasing static code density of non-critical phases. The RISC-V mode supports flushing instructions during branches for efficient execution of control code by masking the function unit execution and register file write signals, which can be done with low-cost hardware changes. The support for toggling between instruction sets is implemented by multiplexing the microcode-generated instruction with the original fetched instruction. This multiplexer is controlled by custom instructions that are implemented both for the RISC-V and exposed datapath instruction sets.

IV. Programmable Dictionary Compression

We build our compression scheme on a state-of-the-art run-time programmable dictionary compression method [10]. The benefit of this method lies in its flexibility, which makes it possible to update the dictionaries on a loop-granularity during run time, leading to high compression ratios compared to more coarse-grained methods while limiting the execution time overhead. Moreover, the compression method uses parallel dictionaries that break the instruction into several parts. This

TABLE I: Included domain-specific instructions for accelerating DSP-type workloads and fixed point computation.

Operation	Description	Types	RISC-V Support
ADD-/SUBRHI	Adds/substracts subword values, then rounds up and propagates the MSBs to the result.	(un)signed 8x4/16x2	X
MULRHI	Integer multiply between inputs with rounded MSBs propagating to the result.	signed 8x4/16x2	X
REFLECT	Reflects the input bits.	(un)signed 32	
SATACCDOT	Saturating dot product with accumulation.	(un)signed 8x4/16x2	
SATSUBU	Integer subtraction with negative difference saturated to zero.	unsigned 8x4/16x2	X
SHRRU	Logical shift right with rounding and saturation.	unsigned 8x4/16x2	X
SHUFFLE2	Shuffles two vectors based on the third input.	8x4/16x2	
VCAST	Casts a vector to another width based on the selection operand	unsigned 8x4/16x2	X

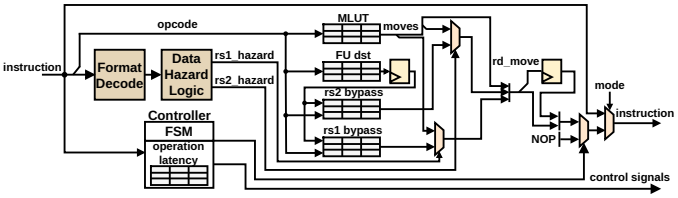


Fig. 3: The microcode unit adds support for exposed datapath and RISC-V instruction set modes via a programmable bypass.

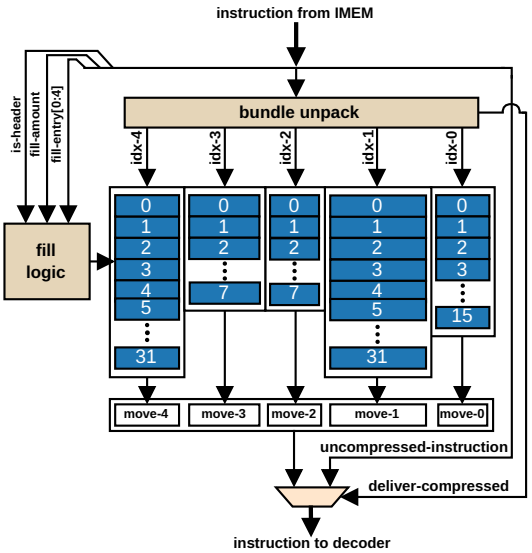


Fig. 4: The decompressor reads move entries from dictionaries based on the indexes that are included in the compressed instruction bundle. The instruction template includes addressing bits for explicitly expressing dictionary programming, compressed and uncompressed instructions.

fits well with the move-based programming model, where each move slot can be mapped to its own dictionary. To support compressed instructions with a minimal hardware overhead, the compressed instructions are formed into a bundle, whose size matches the width of uncompressed instructions.

A. Should Both Instruction Sets Support Compression?

Similarly to the exposed datapath mode, the run-time programmable dictionary compression leverages the locality property of loops. In the scope of the use case of the processor, the hot loops of the program should be converted to the exposed datapath mode to allow maximum performance and efficiency for critical code regions with a trade-off in code density. From this point of view, the overhead of supporting dictionary compression for both instruction sets is not justified,

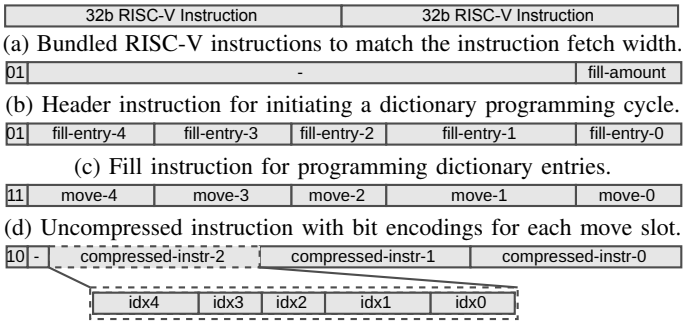


Fig. 5: The DSP's instruction formats.

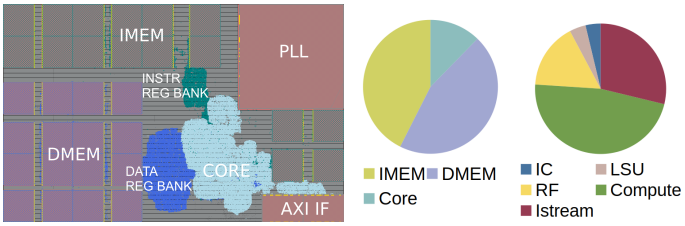
since the RISC-V mode is primarily targeted for regions with low locality where the compression would not be utilized.

B. Hardware Support

Run-time decompression requires hardware extensions to the processor's control path. Figure 4 describes the decompressor block that contains the dictionary elements. Since we support mixing compressed and uncompressed instructions, the instruction templates, illustrated in Figure 5, have dedicated control bits to express the instruction type. The programming cycle begins by indicating with a header instruction the amount of dictionary entries to fill. The following instructions are then added to the dictionaries by the fill logic that extracts the move slots from the template. After this cycle, when a compressed instruction bundle enters the instruction stream, the control logic halts the PC register and executes all compressed instructions from the bundle before resuming fetching new instructions. Uncompressed instructions, on the other hand, are directly passed down the control path.

C. Entry Selection

Since the dictionaries have a finite size, it is important to select dictionary entries efficiently to maximize dynamic code density. The entry selection algorithm, extended from [10], uses a cost-score metric to find optimal dictionary entries. For each converted exposed datapath loop, we create a set of entry candidates by forming compression windows from the basic blocks. The compression windows describe the required dictionary entries for each possible compression bundle. Subsequently, the algorithm iteratively inspects how many compressed bundles, scaled by the basic block trip count, would be created in the entire loop region if the candidate is added to the dictionary. We further penalize this score by the number of added dictionary entries normalized by the move slot dictionary size. By using



(a) (b) (c)

Fig. 6: The layout (a), toplevel's area breakdown (b), core's area breakdown (c).

the trip count to scale the compressed bundles, the algorithm naturally emphasizes the innermost basic blocks of nested loops, while also evaluating the outer loops. The algorithm continues to evaluate the entry candidates until the dictionary is full or no candidate is able to compress additional instructions.

V. Multi-instruction-set Compiler

Code generation is essential for utilizing the flexibility of the multimode architecture efficiently. The compiler used in this work is built on top of LLVM, due to its flexibility and extensibility in supporting customized architectures. Since we target only performance-critical loop regions for the exposed datapath, it is suitable to use the RISC-V backend for instruction selection and register allocation and integrate into this flow to convert the code regions of interest into exposed datapath code. Naturally, this requires a program model conversion step, where the RISC-V machine instructions are converted into unscheduled sequential TTA moves that are used to build data dependency and control flow graphs for instruction scheduling. The scheduling of the move instructions consequently results in multiple operations being executed in parallel, constrained by data dependencies and the datapath resources. The compiler runs a static analysis on the scheduling results and compares it against the RISC-V microarchitectural model to select loop regions for code conversion. This way, only loops that are beneficial to be run on the exposed datapath are converted. Since the processor has a static access latency both to the instruction and data memory, the loop conversion analysis can be directly done based on the scheduling results.

After the static analysis, the compiler operates the dictionary compression algorithm on the converted loop regions. First, the compiler finds the optimal dictionary entries based on the heuristic described in Section IV-C. Using the available dictionary entries, bundles of compressed instructions that point to dictionary indexes are formed. Finally, the compiler replaces the original RISC-V loop regions with partially compressed exposed datapath code, and inserts accompanying mode switch instructions into the loop prologue and epilogue in addition to the necessary dictionary header and fill instructions.

VI. Subsystem and Physical Design

The DSP was taped out as a part of a multiprocessor test chip using a commercial 22-nm technology and commercial EDA tooling. The processor subsystem, illustrated in Figure 1, has 64-kB instruction and data memory scratchpads that are connected to JTAG and AXI-slave interfaces, allowing program offloading. Additionally, the load-store unit integrates with an AXI-master interface for accessing in- and off-chip memory, as well as for supporting direct communication with other subsystems. The DSP has its own *phase locked loop* (PLL) IP, which makes it possible to operate the processor on a high frequency without other system-level components creating timing bottlenecks, and, on the other hand, it enables flexible frequency scaling for target workloads.

The fabricated chip was successfully tested at a 1.0 GHz clock frequency using a 0.9 V operating voltage, which leaves room for further clock frequency increase through voltage scaling at the expense of energy consumption. The design area was 0.75 mm^2 with the layout of the processor illustrated in Figure 6a. For testing and debugging purposes, we placed 128-byte instruction and 512-byte data register banks alongside the SRAM modules to allow running small kernels without accessing the local or external memories. The core utilizes only 10% of the total area with the detailed breakdown shown in Figure 6c. The lightweight microcode unit only accounts for 2% of the core's area. Thanks to the exposed datapath programming interface, we maintain a small register file (2 reads and 1 write port) and a pruned *interconnection network* (IC), which results in a power- and area-efficient design.

VII. Evaluation

The RTL of the proposed design and its high-level language compiler was generated by utilizing OpenASIP [13], extending it with dictionary compression and multi-instruction-set hardware and programming features. Full system measurements are done by using four digital signal processing applications, described in Table II, selected to represent low-power edge device use cases appropriate for a SoC of this scale: 1) a 2D *discrete cosine transform* (DCT) (128x128 8-bit grayscale image with 16-bit quantization) based on [14], 2) an *adaptive scalable texture compression* (ASTC) [15] encoder (12x12 block size) for 84x84 image with 4 channels, 3) JPEG [16] image coder from the CHStone benchmark suite [17], and 4) a *keyword spotting* (KWS) model inference described in Section VII-A. ASTC is a relevant application for near-sensor edge devices that offload compressed image or video data to a remote device for inference [18]. DCT and JPEG represent traditional DSP applications, while KWS demonstrates the DSP's capability for small-scale real-time inference. In this section, we refer to the RISC-V/TTA architecture target as *Dual-IS*.

A. Keyword Spotting Classification

To evaluate the DSP's performance in AI workloads, we use the *keyword spotting* (KWS) task from the MLPerf Tiny benchmark [19] using a custom TVM-based [20] software stack. At 53 kB, DSCNN is the only model in the suite that fits entirely into the DSP's data memory. We measure the power consumption directly from the test board while running the inference. Since the DSP is part of a multiprocessor SoC, and we cannot power gate all other subsystems, we first measure the idle power consumption of the SoC and subtract that from the measured power when DSP is operating under full load.

TABLE II: Application characteristics.

	astc	dct	jpeg	kws
SIMD	8x4	16x2	-	8x4
Operations	reflect, satsubu shrru, sataccdot	shuffle2, mulrhi, addrhi, subrhi	-	sataccdot

TABLE III: MLPerf Tiny DSCNN performance [21].

	Energy (μJ)	Latency (ms)
Cortex-M33	656	29
Cortex-M4	1820	43
Cortex-M7	1376	11
This Work	250	5

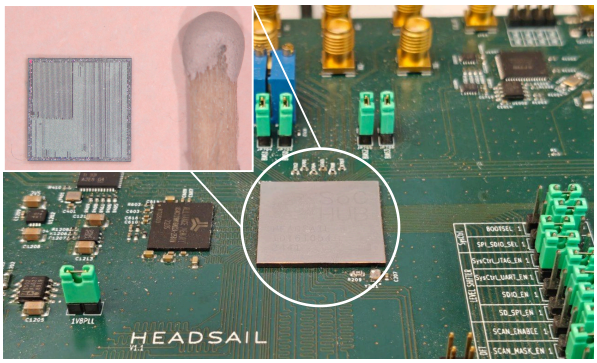


Fig. 7: The chip integrated with the test board to run full system evaluation on real-time audio processing.

Running at 1 GHz, we achieve an average latency of 5 ms per inference and power consumption of 50 mW (250 μJ per inference), while maintaining over 90% accuracy. Table III lists latency and energy numbers for KWS inference with different processors, as reported in the MLPerf tiny benchmark results [21]. The proposed DSP achieves 50–90% lower latency with 60–90% lower energy consumption compared to ARM-M family of processors without using fixed-function accelerators and while maintaining compiler-supported programmability.

To demonstrate the DSP's real-time capabilities, we implement an end-to-end audio classification flow based on the CNN described in [22]. The test setup, illustrated in Figure 7, uses a UART-connected microphone and the on-board boot processor to write audio clips containing 8 000 samples to shared SRAM for the DSP to classify. With the model's 16-kHz sampling rate, this imposes a 500 ms real-time inference constraint. Double buffering enables inference to be run concurrently with audio sampling, while classification results are signaled by blinking an LED via the IC, indicating the recognized label.

B. Code Density

Table IV lists the instruction image sizes and amount of fetched instruction bytes with different configurations. Generally, compiling the programs purely to the RISC-V ISA achieves the best static code density. In KWS and ASTC, the RISC-V configuration relies on full-width multiply instructions to execute dot product accumulations, since it lacks support for the dot product instructions, which causes a significant increase in code size. Using the Dual-IS scheme shows a clear benefit in static code density with an average reduction of 16% in code size compared to the pure TTA instruction binary.

TABLE IV: Code size* (kB) and amount of fetched instruction bytes† (MB) with different configurations.

	TTA	RISC-V	Dual-IS	Compressed Dual-IS
astc	4.6* | 1.0†	3.2 | 1.0	4.2 | 0.9	3.9 | 0.6
dct	1.8 | 3.3	0.9 | 1.8	1.7 | 3.1	1.7 | 2.4
jpeg	51.1 | 16.7	29.0 | 8.5	42.4 | 11.0	37.7 | 8.9
kws	33.2 | 45.2	21.6 | 61.6	21.7 | 42.4	20.1 | 26.7

Dictionary compression grows this difference to 22%. The additional flexibility of controlling the code size is especially beneficial for small edge devices, such as the fabricated SoC that has a limited memory budget. In this type of scenarios, the exposing of the datapath can be limited to a point where the program binary can fit into the memory.

In these applications, apart from JPEG, Dual-IS utilizes the RISC-V mode scarcely, since the applications consist of clear hot regions that are converted to exposed datapath code. This is reflected in the dynamic code density over the pure TTA execution. In JPEG, which is more heterogeneous in its workload characteristics between different program phases, the fetches from the instruction memory are reduced 34%, while with other applications the difference is approximately 6%, which averages to a 13% reduction in the amount of fetched instruction bits. Incorporating dictionary compression, however, has a stronger effect on the dynamic code density since it is targeted for program hot spots. Compared to the pure TTA execution, compressed Dual-IS reduces the amount of fetched instruction bits by 40% on average and 16% compared to the pure RISC-V ISA.

C. Performance

Figure 8a illustrates execution times when the code is targeted for different architecture variants. The RISC-V utilization, highlighted in the figure, is highly dependent on the application's computational characteristics. For instance, the DCT application consists of one deeply nested loop that is converted entirely into exposed datapath code. JPEG, on the other hand, favours the RISC-V mode due to control-oriented phases such as Huffman coding dominating the workload. As the dictionaries are programmed in loop prologues, the performance overhead of the compression remains minimal, 0.8% on average and 2% in the worst case.

Interestingly, mixing the RISC-V and exposed datapath instruction sets during code generation achieves on average 1% higher performance than the pure TTA instruction set. This is partly due to techniques such as pipeline flushing and shorter unconditional jump latencies resulting in more efficient execution of control-oriented code, which is not possible with the TTA execution model due to its visible delay slots. Additionally, due to small differences in the compiler backends, the LLVM-generated unscheduled machine code does not perfectly match between the TTA and Dual-IS, which can lead to further differences during the complex heuristics-based instruction scheduling. On average, the execution time is 42% lower with Dual-IS than with the pure RISC-V instruction set thanks to the static multi-issue execution and ternary instruction support.

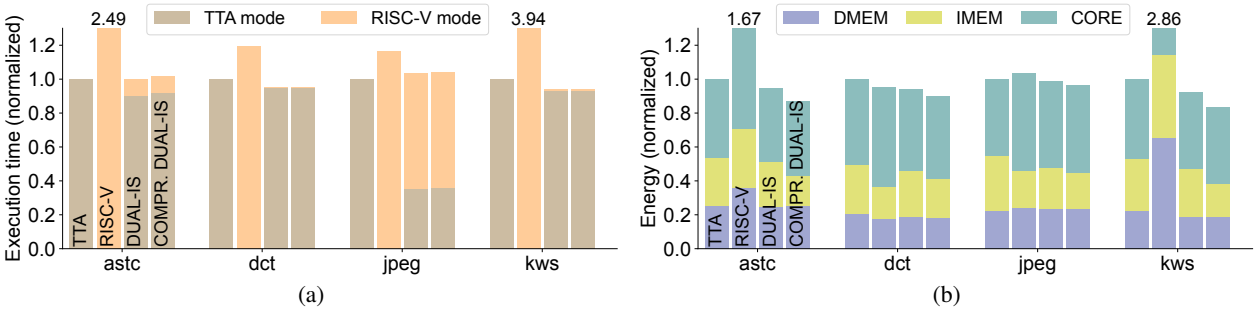


Fig. 8: Normalized execution time (a) and energy consumption (b) for different configurations and applications.

TABLE V: Comparison of low-power DSPs.

	Issue	Tech.	Voltage	Freq.	Power
CV32E40P*[23]	single	65 nm	NA	250 MHz	17 mW
HAMSA-DI [24]	dual	16 nm	0.8 V	1.0 GHz	55 mW
Ickes *et. al* [25]	quad	28 nm	1.0 V	587 MHz	113 mW
This Work	dual	22 nm	0.9 V	1.0 GHz	50 mW

* Values are extracted from [24].

D. Energy Consumption

Flexible use of the architecture results in instruction stream energy savings, illustrated in Figure 8b using gate-level power simulations to provide detailed breakdowns. On average, we reduce the instruction memory energy consumption 12% through the Dual-IS method, and, in the best case, in JPEG, 25% compared to the pure TTA target. With dictionary compression, the energy consumption of the instruction memory is 33% lower on average. The benefit of the exposed datapath is demonstrated in the core's energy consumption. The low-level code optimizations reduce register accesses and interconnection toggling, which results in lower energy consumption. This is clearly observed in JPEG that uses the same instructions for both the TTA and RISC-V target, and in DCT that apart from the shuffle instructions shares the same arithmetic utilization, as listed in Table II. In these applications, the RISC-V target consumes, on average, 22% more core energy than the TTA target, mainly due to the increased interconnection and RF utilization. The microcode unit itself has only a small effect on the power consumption. With Dual-IS, the microcode accounts only for 2.2% of the core's and 1.1% of the total power consumption. The instruction stream optimizations lead to a total energy consumption reduction of 11% compared to the pure TTA architecture and 47% compared to the RISC-V target.

VIII. Related Work

Table V lists published low-power DSPs. Apart from Ickes *et. al* [25], these DSPs are based on the RISC-V ISA. HAMSA-DI [24], a dual-issue variation of the CV32E40P [23], is the most interesting reference point, since it also supports dual-issue execution, but does this by means of a superscalar pipeline. Similarly to the CV32E40P, HAMSA-DI supports dot product instructions that are similar to the ones used in this work, but without saturation mechanics. The authors report a power consumption of 55 mW in a dot product benchmark corresponding to a similar computational load as our measured neural network inference, which runs at 50 mW. We achieve this level of power efficiency with a higher operating voltage, larger process node and two times larger data memory, demonstrating the potential of the proposed work in low-power use cases.

Multi-instruction-set architectures have been scarcely researched, the most notable work being Denver [26] that implemented a processor with dynamically and statically scheduled modes. Lin *et al.* [27] and Hou *et al.* [28] proposed processors with RISC and VLIW modes. Hepola *et al.* [12] proposed combining the low-level programming interface of exposed datapath architectures with RISC-V through microcode, which serves as a basis for this work. Instruction dictionary compression [29]–[32] has been researched at different granularities. Multanen *et al.* [10] proposed updating the dictionaries at a loop-basis to minimize the performance overhead, which we also apply in our work. The novelty of the proposed architecture lies in its combination of run-time programmable compression with a flexible multi-instruction-set scheme. This approach leads to a highly optimized instruction stream while preserving static multi-issue execution support in combination with an application-specific instruction set within a fabricated chip.

IX. Conclusions

This paper presents a novel silicon-proven "Beaivi" DSP, featuring a unique architectural approach that supports three distinct instruction set modes with integrated DSP extensions. The design, implemented using a 22-nm process and targeting low-power applications, utilizes 0.75mm^2 of area with an integrated PLL and tightly coupled instruction and data memories. The fabricated chip was tested at a 1.0 GHz clock frequency and a 0.9 V operating voltage, while drawing 50 mW under a neural network inference workload, measured directly from the test board. Compared to a pure exposed datapath architecture, the proposed architecture achieves 22% smaller code size and reduces the amount of fetched instruction bits 40%, leading to an 11% reduction in energy consumption without increasing execution time. The results highlight the benefit of the proposed architecture in low-power DSP use cases and its flexibility as a compiler-supported code generation target.

Acknowledgements

The authors are grateful for the funding from Academy of Finland (decision #353199) and the TRISTAN project. TRISTAN has received funding from the Key Digital Technologies Joint Undertaking (KDT JU) under grant agreement nr. 101095947.

REFERENCES

[1] R. Kaplan, "Intel Gaudi 3 AI accelerator: Architected for gen AI training and inference," in *Hot Chips 36 Symposium*. IEEE, 2024.

[2] M. H. Ionica and D. Gregg, "The Movidius Myriad architecture's potential for scientific computing," *IEEE Micro*, 2015.

[3] L. Codrescu, "Architecture of the Hexagon™ 680 DSP for mobile imaging and computer vision," in *Hot Chips 27 Symposium*. IEEE, 2015.

[4] A. Rico, S. Pareek, J. Cabezas *et al.*, "AMD XDNA™ NPU in Ryzen™ AI processors," *IEEE Micro*, 2024.

[5] H. Corporaal, *Microprocessor Architectures: from VLIW to TTA*. John Wiley & Sons, Inc., 1997.

[6] ——, "TTAs: Missing the ILP complexity wall," *Journal of Systems Architecture*, 1999.

[7] V. Guzma, T. Pitkänen, and J. Takala, "Use of compiler optimization of software bypassing as a method to improve energy efficiency of exposed data path architectures," *EURASIP Journal on Embedded Systems*, 2013.

[8] J. Hoogerbrugge and H. Corporaal, "Register file port requirements of transport triggered architectures," in *Proceedings of the 27th annual international symposium on Microarchitecture*, 1994.

[9] J. Heikkinen, J. Takala, A. Cilio, and H. Corporaal, "On efficiency of transport triggered architectures in DSP applications," *Advances in Systems Engineering, Signal Processing and Communications*, 2002.

[10] J. Multanen, B. de Bruin, H. Corporaal, and P. Jääskeläinen, "Energy-efficient instruction compression with programmable dictionaries," *Design Automation for Embedded Systems*, 2024.

[11] K. Keutzer, S. Malik, and A. R. Newton, "From ASIC to ASIP: The next design discontinuity," in *Proceedings. IEEE International Conference on Computer Design: VLSI in Computers and Processors*. IEEE, 2002.

[12] K. Hepola, J. Multanen, and P. Jääskeläinen, "Energy-efficient exposed datapath architecture with a RISC-V instruction set mode," *IEEE Transactions on Computers*, 2023.

[13] ——, "OpenASIP 2.0: Co-design toolset for RISC-V application-specific instruction-set processors," in *33rd International Conference on Application-specific Systems, Architectures and Processors*. IEEE, 2022.

[14] P. Lee and F.-Y. Huang, "Restructured recursive DCT and DST algorithms," *IEEE Transactions on Signal Processing*, 1994.

[15] J. Nystad, A. Lassen, A. Pomianowski *et al.*, "Adaptive scalable texture compression," in *Proceedings of the Fourth ACM SIGGRAPH/Eurographics Conference on High-Performance Graphics*, 2012.

[16] G. K. Wallace, "The JPEG still picture compression standard," *Commun. ACM*, 1991.

[17] Y. Hara, H. Tomiyama, S. Honda *et al.*, "Chstone: A benchmark program suite for practical C-based high-level synthesis," in *International Symposium on Circuits and Systems*. IEEE, 2008.

[18] J. Žádník, M. Mäkitalo, and P. Jääskeläinen, "Pruned lightweight encoders for computer vision," in *24th International Workshop on Multimedia Signal Processing*. IEEE, 2022.

[19] C. Banbury, V. J. Reddi, P. Torelli *et al.*, "MLPerf tiny benchmark," *arXiv preprint arXiv:2106.07597*, 2021.

[20] T. Chen, T. Moreau, Z. Jiang *et al.*, "TVM: An automated end-to-end optimizing compiler for deep learning," in *13th USENIX Symposium on Operating Systems Design and Implementation*, 2018.

[21] MLCommons, "MLPerf tiny benchmark v1.2 results," https://mlcommons.org/benchmarks/inference-tiny/, 2024, accessed: 2025-05-04.

[22] S. Abdoli, P. Cardinal, and A. L. Koerich, "End-to-end environmental sound classification using a 1D convolutional neural network," *Expert Systems with Applications*, 2019.

[23] M. Gautschi, P. D. Schiavone, A. Traber *et al.*, "Near-threshold RISC-V core with DSP extensions for scalable IoT endpoint devices," *Transactions on very large scale integration (VLSI) systems*, 2017.

[24] Y. Kra, Y. Shoshan, Y. Rudin, and A. Teman, "HAMSA-DI: A low-power dual-issue RISC-V core targeting energy-efficient embedded systems," *Transactions on Circuits and Systems I: Regular Papers*, 2023.

[25] N. Ickes, G. Gammie, M. E. Sinangil *et al.*, "A 28 nm 0.6 V low power DSP for mobile applications," *Journal of Solid-State Circuits*, 2011.

[26] D. Boggs, G. Brown, N. Tuck *et al.*, "Denver: Nvidia's first 64-bit ARM processor," *IEEE Micro*, 2015.

[27] T.-J. Lin, C.-M. Chao, C.-H. Liu *et al.*, "A unified processor architecture for RISC & VLIW DSP," in *Proceedings of the 15th ACM Great Lakes Symposium on VLSI*, 2005.

[28] Y. Hou, H. Hu, Y. Xu *et al.*, "FuMicro: A fused microarchitecture design integrating in-order superscalar and VLIW," *VLSI Design*, 2016.

[29] C. Lefurgy, P. Bird, I.-C. Chen *et al.*, "Improving code density using compression techniques," in *Proceedings of the Annual International Symposium on Microarchitecture*, 1997.

[30] L. Benini, A. Macii, E. Macii *et al.*, "Selective instruction compression for memory energy reduction in embedded systems," in *Proceedings of the International Symposium on Low Power Electronics and Design*, 1999.

[31] M. Brorsson and M. Collin, "Adaptive and flexible dictionary code compression for embedded applications," in *Proceedings of the International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, 2006.

[32] M. Thuresson, M. Själander, and P. Stenstrom, "A flexible code compression scheme using partitioned look-up tables," in *High Performance Embedded Architectures and Compilers*, 2009.