Energy-Efficient Exposed Datapath Architecture With a RISC-V Instruction Set Mode

Kari Hepola, Joonas Multanen, and Pekka Jääskeläinen

***Abstract*—Transport triggered architectures (TTAs) follow the static programming model of very long instruction word (VLIW) processors but expose additional information of the processor datapath in the programming interface, which enables low-level code optimizations but results in lower code density. Multi-instruction-set architectures add flexiblity via their ability to switch instruction sets during execution. The added flexibility is interesting for VLIW-style processors because it enables reducing the large instruction stream energy footprint by using an instruction set with enhanced code density in regions with limited opportunities for exploitation of instruction level parallelism. In this article, we introduce a dual instruction-set architecture, "Dual-IS", that implements both RISC-V and TTA instruction sets with shared datapath resources by means of a lightweight microcode unit. In order to utilize the flexible architecture automatically, we introduce a compilation method that is able to independently target code for both instruction sets based on static code analysis and a microarchitectural model of the processor. Compared to a single-ISA TTA processor, we were able to lower the instruction stream energy consumption 45% on average in the best design point, which resulted in a total energy consumption reduction of 26% and a 0.4% lower run time.**

***Index Terms*—Exposed datapath, instruction level parallelism, multi-instruction-set architecture, transport triggered architecture.**

I. Introduction

In recent years, architecture designers have had to introduce more elaborate ways to increase performance of digital designs [1] as advancements in process technologies have begun to stagnate [2]. At the same time, a larger emphasis has been placed on energy consumption [3], especially in embedded use cases with a limited energy budget. The end of Moore's law has made custom hardware a viable option to meet the ever rising performance and energy efficiency targets as applications become more complex. The trade-off with custom hardware is in portability and programmability that requires more effort from the compilation framework.

Very long instruction word (VLIW) processors can exploit *instruction level parallelism* (ILP) energy-efficiently due to static compiler-oriented control, which simplifies the hardware implementation. The main drawback of VLIW processors is high instruction stream energy consumption due to low code density caused by the use of *no-operations* (NOPs) when instruction packets cannot be fully utilized in code regions with limited ILP. *Transport triggered architectures* (TTAs) follow the static VLIW-style programming model but expose the datapath to the programmer to enable low-level code optimizations, which makes scaling of multi-issue processors easier due to a modular structure and reduced register file traffic. The low-level programming interface, however, requires even wider instructions than a traditional VLIW architecture due to the additional state data on the processor datapath being exposed to the programmer.

Multi-instruction-set architectures, such as Nvidia Denver [4], are more flexible thanks to the option of switching the programming interface of the processor. Traditionally, programmable accelerators are tightly coupled to a host processor that can execute general-purpose code efficiently. Incorporating multiple instruction sets increases flexibility and therefore efficiency of executing general-purpose code on standalone accelerators, which reduces the need for using a separate host processor. Using a standalone accelerator over a multiprocessor system reduces the amount of *dark silicon* [5], leading to a smaller area utilization. In addition, expensive data copying is reduced as the program context is shared in the register file and cache hierarchy.

With VLIW architectures, multiple instruction sets enable opportunities for instruction stream energy optimizations by the use of an instruction set with a higher code density when ILP is scarce. In TTA's case, this can be taken even further by supporting an instruction set with a higher-level programming interface, such as a *reduced instruction set computer* (RISC) based instruction set. This approach helps to reduce the overhead of exposing the processor datapath as its use is more closely targeted.

The article makes the following novel contributions:

- A novel dual-mode architecture composing of an exposed datapath VLIW and RISC-V instruction sets for static exploitation of instruction level parallelism with reduced instruction stream impact.
- A microcoded control logic based RISC-V implementation for sharing datapath resources with the exposed datapath instruction set.
- A compilation method for generating code with fine-grained instruction set mode switching to utilize

Manuscript received 3 July 2023; revised 14 November 2023; accepted 19 November 2023. Date of publication 28 November 2023; date of current version 15 January 2024. Recommended for acceptance by J. Zhai. *(Corresponding author: Kari Hepola.)*

Kari Hepola and Joonas Multanen are with the Faculty of Information Technology and Communication Sciences, Tampere University, 33100 Tampere, Finland (e-mail: kari.hepola@tuni.fi).

Pekka Jääskeläinen is with the Faculty of Information Technology and Communication Sciences, Tampere University, 33100 Tampere, Finland, and, also the Intel Finland Oy, 02160 Espoo, Finland.

Digital Object Identifier 10.1109/TC.2023.3337313

the multi-issue capabilities in parallel code regions but optimizing for code density in control-oriented parts.

This article extends our preliminary work published in a conference paper [6] with the following new contributions: 1) A novel compilation method that is able to utilize both instruction sets in generated code based on static code analysis and a microarchitectural model of the processor. 2) An interface for switching instruction set modes during run-time and analysis of its effect on the processor pipeline. 3) Improved pipeline structure for the RISC-V microarchitecture, with support for pipeline flushes. 4) Comparison to a popular optimized 3rd party RISC-V processor.

By supporting multiple instruction sets, we can significantly improve the flexibility of the programmable processor, as the system has a binary compatibility with the RISC-V *instruction set architecture* (ISA). The compilation method introduced in this work utilizes both the static multi-issue capabilities of TTA and the compact instructions of RISC-V without the user having to acknowledge the compiler's use of multiple instruction sets. Thanks to the added flexibility, the processor has a lower energy consumption than a single-ISA TTA processor while benefiting from higher performance compared to a single-ISA RISC-V design. The dual-mode architecture can be viewed as an instruction stream optimization to a TTA processor, but from another point of view, as a means of adding support for static instruction level parallelism to the RISC-V instruction set, which is another novel aspect of this work.

We implement our proposed multi-instruction-set architecture in a *register-transfer level* (RTL) description, synthesize it with a modern 28 nm ASIC technology and simulate the synthesized netlist with 8 CHStone [7] benchmarks, EEMBC Coremark [8] and the Opus audio codec [9]. Opus is significantly larger than the other benchmarks, which makes it possible to evaluate the applicability of the system for large-scale embedded applications. All benchmarks were compiled with the proposed compilation flow to support fine-grained instruction set switching. The acquired instruction set flexibility reduces instruction stream energy consumption by 45% on average in the best design point, which reduces total system energy consumption by 26% and lowers run time by 0.4% compared to a single-ISA TTA processor with a similar datapath. The added hardware to support multiple instruction sets does not affect the maximum clock frequency but increases area utilization by 13% compared to a single-ISA TTA core.

II. Related Work

Due to the importance of code density and binary compatibility, previous work has researched forming more compact *compressed instruction sets* of base instruction set architectures. These compressed instruction sets constrain the instruction set so that instructions can be packed into a narrower instruction format by limiting the amount of operations, immediate values or register file addressing. ARM Thumb [10] is an example of such an instruction set. It reduced the standard 32-bit ARM instruction format to 16-bits. The mode could be switched by the programmer, and the compiler would use instruction set modality on a function-level. In addition to a function-level granularity, methods [11], [12], [13] using a more fine-grained granularity have been proposed for compressed instruction sets. The Thumb approach differed from mixed instruction set approaches such as the RISC-V C extension that could use both normal and compressed instructions without implementing multiple modes. The limitation with these approaches is that they cannot explore code generation between completely different instruction sets that have more architectural differences compared to a subset of the same instruction set.

Karaki et al. [14] proposed a dual instruction-set architecture that combined x86 and ARM instruction set modes via a hardware interpreter that translated x86 instructions to ARM *micro-operations* during run time. Their approach is interesting from instruction set modality point of view because it combines both a RISC and a *complex instruction set computer* (CISC) that have a different ISA design philosophy.

VLIW processors have been interesting for multi-instruction-set architecture research due to their wide instructions that cause a high instruction stream energy footprint in sequential code. Lin et al. [15] proposed a dual mode RISC and VLIW mode processor that relied on hierarchical instruction encoding to specify the mode. Hou et al. [16] implemented a multi-instruction-set processor with both VLIW and superscalar modes. However, their processor did not have a compiler that could target both instruction sets and therefore relied on precompiled VLIW libraries.

Multi-instruction-set architectures have seen some commercial success with Nvidia's project Denver [4]. Denver implemented both a superscalar *out-of-order* (OoO) dual-issue mode and a static 7-issue microcode mode. Code was targeted for the standard ARM instruction set and optimized dynamically during run time for the microcode mode that would bypass the out-of-order scheduler and directly forward optimized micro-operations to the in-order scheduler. While the concept of Denver is close to the processor proposed in this work, we take a different approach with static code generation and use of microcoded control logic that is able to combine an exposed datapath and a RISC architecture with shared hardware resources.

Samsung reconfigurable processor [17] implemented both VLIW and a *coarse grained array* (GGA) modes. The processor had two instruction formats, where the CGA format was wider to support a higher amount of parallelism. In addition to these two modes, Samsung reconfigurable processor was used together with different power modes [18] where the programmer could insert macros into the code in order to switch power modes in different code regions, which would power gate unused hardware resources.

Heterogeneous multicore systems can employ multiple different ISAs per core to utilize different architectural features, which enables migrating the application execution to the most suitable core based on computational needs. Composite-ISA [19] is an example of such an approach that supports configuring architectural features, such as register file depth and predication. This allows using the same superset ISA for all cores to lower process migration overhead. While heterogenous multicores can use multiple ISAs to increase flexibility, they

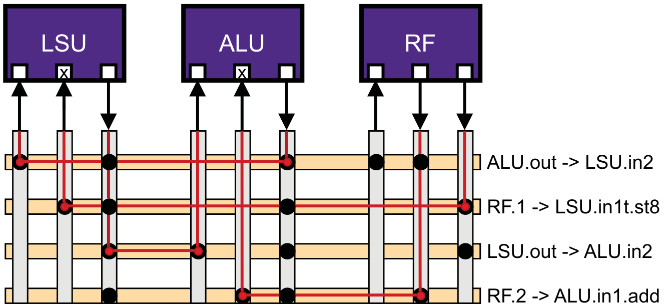


Fig. 1. An example of the transport triggered programming model where operand moves are visible to the programmer on the exposed datapath.

cannot utilize fine-grained mode switching within the application or share datapath resources with different instruction set as done in our work.

III. Transport Triggered Architectures

Transport triggered architectures are statically scheduled architectures that follow the VLIW-style instruction level parallelism paradigm, where the programmer is responsible for explicitly stating which operations are executed in parallel. Compared to traditional "operation-triggered" [20] VLIW processors, TTAs have a lower-level programming interface due to their datapath and connectivity being exposed to the programmer. The programming interface is based on *moves* that describe operand transportations on the datapath interconnection (IC) network and operations that are triggered as a side effect. In the TTA programming model, function unit ports act as registers that are able to store operand values over clock cycles, which increases the amount of state data. Fig. 1 describes how move instructions transport operands between the register file and function units, which executes operations as a side effect. Socket encodings in the TTA instruction format control the interconnect multiplexers, which allows the programmer to specify sources and destinations for operand transportations. After an operation is triggered, its value is stored into the output port of the function unit after a design-time specified amount of clock cycles, which allows pipelining operations.

With VLIW architectures, no-operations are used when the wide instruction packets cannot be fully filled due to dependencies between the operations. Exposing the datapath to the programmer creates additional state data, which leads to even wider instructions than with traditional operation-triggered VLIW architectures [21]. Wider instructions lower code density, which is the main drawback of transport triggered architectures. The main benefits of TTAs are related to register file accesses that are reduced with the following low-level code optimizations acquired by the additional instruction scheduling freedom:

- **Operand sharing** [22] happens when an operand has been previously transported to the function unit input register, which enables the following operations to share it.
- **Software bypassing** [23] allows the programmer to explicitly bypass operands without routing them through the register file.

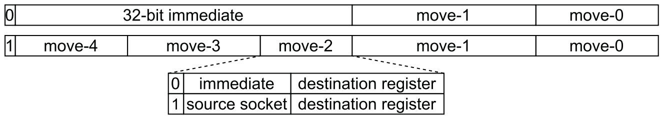


Fig. 2. A description of a TTA instruction format. Each move slot is split into destination and source fields that describe immediate, socket and register file encodings.

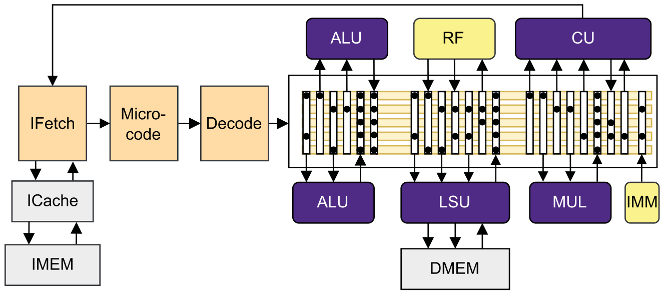


Fig. 3. Structure of the Dual-IS processor with a microcode unit between instruction fetch and decode.

- **Dead result elimination** eliminates a writeback to the register file when a variable is no longer alive.

Hoogerbrugge and Corporaal [24] showed that transport triggered architectures need 0.5 read and 0.35 write ports per operation on average, while a traditional RISC operation, such as a register add, requires two register input operands and one output operand. Reducing both the amount of register file ports and interconnect connectivity is important with wide processor architectures, as otherwise they quickly hit the ILP complexity wall [25] due to the amount of connectivity on the processor datapath. TTAs' modular structure allows heavy pruning of the interconnection network due to programmer-directed operand transportations, which enables easy scaling from high-performance [26] to small energy-efficient TTA designs [27].

Fig. 2 describes an example of a TTA instruction template for an architecture with five busses. The example template consists of two formats, one of which specifies a 32-bit immediate value. In the format, each bus is assigned a move slot field that is split into destination and source subfields. Depending on the datapath connectivity, each subfield can encode short immediates, sockets and register file indexes. If a bus has the capability to trigger an operation, the move slot field will also contain operation encodings that are passed to the function unit when the triggering operand is transported.

IV. Dual-IS Processor

The proposed Dual-IS processor implements a multi-instruction-set architecture designed around the popular open-standard RISC-V and TTA-based instruction sets. The processor structure is described in Fig. 3, where the exposed datapath, with its connectivity and function units, is connected with the rest of the processor pipeline and a microcode unit implementing RISC-V control logic.

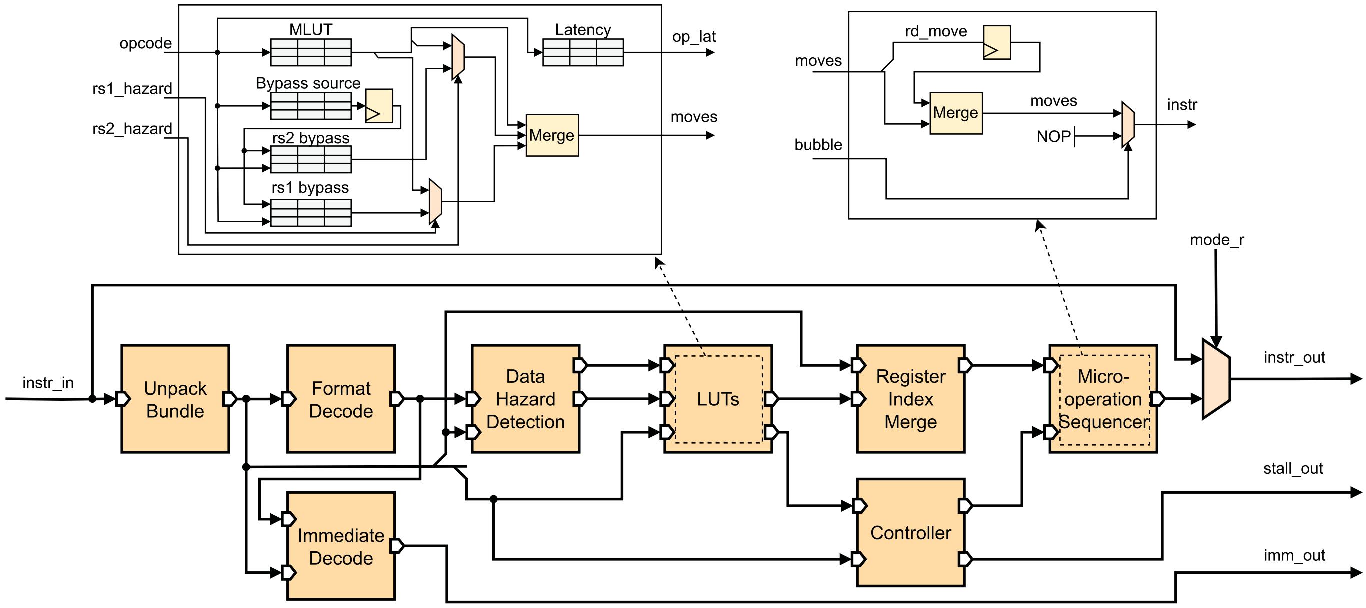


Fig. 4. Overview of the microcode unit structure. The microcode unit translates and sequences RISC-V instructions to TTA micro-operations that are decoded in the native TTA decoder. Instruction set mode is switched with a custom instruction that controls the output multiplexer inside the microcode unit, which enables to either translate the fetched instruction via the microcode or directly pass it to the native TTA decoder.

The microcode unit described in Fig. 4 is important from the instruction set modality point of view because it contains a large majority of the hardware responsible for connecting the two instruction sets. The microcode unit translates RISC-V instructions to TTA moves for the exposed datapath and sequences them. In addition, the microcode unit implements dynamic RISC features that are not found in TTAs, such as control and data hazard detection.

A. Move Look-Up Table

Translating RISC-V instructions to TTA instructions in hardware during run time requires a *move look-up table* (MLUT) that maps RISC-V instructions to moves for the exposed datapath. The structure of the move look-up table must be carefully considered to minimize the hardware overhead of the microcode. Unlike with traditional microcode, we do not require the microcode to be programmable, and therefore we implement all look-up tables with combinatory logic.

Implementing the move look-up table in a way that takes the whole RISC-V instruction as input is unfeasible due to the wide immediate and register file encodings that would require a high amount of entries in the look-up table. Instead, we slice only the *opcode* and *funct* fields from the RISC-V instruction formats, which reduces the input size between 7 and 17 bits out of which only 38 encodings are used in the base instruction set. Based on the opcode and funct fields, we can construct the correct socket and operation encodings that are stored in the look-up table.

After the correct socket and operation encodings are read from the move look-up table, missing register file fields are directly inserted from the RISC-V instruction word to the translated TTA instruction. To support TTA instruction sets with narrower immediate value support, the microcode unit passes the immediate values directly to the interconnect by bypassing the TTA decoder. This approach allows saving encoding space in the TTA instruction set and adds flexibility to the TTA instruction set design, as both instruction sets can support different ranges for immediate values.

B. Micro-Operation Sequencing

Due to the low-level programming interface of TTAs, merely translating the RISC-V instructions for the TTA decoder is not sufficient. The translated instruction must be broken to multiple micro-operations that are sequenced separately. To achieve minimal hardware overhead and minimize the amount of cycles per executed instruction, input operands should be moved to the function unit input ports in parallel, which triggers the operation execution in the function unit.

For single-cycle operations, the result move must be passed to the TTA decoder one cycle later to make sure the function unit has executed the operation before the result operand is moved to the register file on the datapath. As described in Fig. 4, this is implemented by first slicing the result operand move from the translated intermediate instruction and registering it in the micro-operation sequencer. On the following cycle, the result move is passed to the TTA decoder from the result move register, which creates a one cycle delay in the decoding of the move.

For multi-cycle operations, the control logic follows simple in-order behaviour and bubbles the pipeline until the function unit has executed the operation. This is implemented by generating a look-up table that stores operation latencies of multi-cycle operations, which enables the control hardware to bubble the pipeline based on the executed operation without it being exposed to the programmer. Listing 1 describes how a 2-cycle multiply operation causes one bubble in the pipeline as the operation is sequenced in the microcode unit.

Listing 1. Sequencing of a 2-cycle multiply operation

```
mul x3, x1, x2 ⟹
   0: RF.1 → MUL.in2, RF.2 → MUL.in1.mul
   1: ... (NOP)
   2: MUL.out → RF.3
```

C. Pipeline Hazards

Pipeline hazards are a major concern when implementing the microcode due to the very static nature of the TTA programming model. In addition to the actual microcode itself, the microcode unit must be embedded with hardware that implements data and control hazard detection.

Control hazards can happen during the execution of control instructions when a branch decision is not yet known in the pipeline. In the TTA programming model, these instructions would have programmer-visible delay slots that are set to match the pipeline depth. This enables inserting useful instructions into the pipeline slots that could be otherwise bubbled. The RISC-V ISA was designed to minimize the amount of microarchitectural details [28] and therefore has no delay slots, which leads to dynamic hardware that solves control hazards in the processor pipeline.

To overcome this mismatch between the instruction sets, the processor is embedded with control logic that stalls or flushes the pipeline when needed during the execution of control instructions. To implement pipeline flushes, the control unit of the core is extended with additional control signals that can discard the function unit and register file triggers on the datapath. These control signals are set when a taken RISC-V branch instruction enters the control unit, which discards any instructions that were fetched before the branch decision was known.

The TTA programming interface used in this work does not utilize pipeline flushes and instead exposes the delay slots to the programmer. This approach is very useful for loop regions because during loop iterations the branch is taken, and the delay slots are utilized statically. Pipeline flushes are, however, highly beneficial in control-oriented code regions, which is the use case for the RISC-V instruction set in this work. In addition to conditional branches, the programming interface differences between the instruction sets yield benefits in the execution of direct jump instructions. In the RISC-V programming interface, direct jumps can be executed immediately after decoding the instruction as there are no dependencies to the later stages of the pipeline, such as register file reads. With the TTA, direct jumps still require the explicit move on the datapath, which induces a higher latency for the operation.

Solving data hazards in pipelined RISC implementations requires either stalling the pipeline or bypassing the result to the execute stage. Since TTAs control the bypass network in software, they lack dynamic data hazard detection hardware. Implementing dynamic bypasses in the microcode unit requires data hazard detection hardware and additional look-up tables storing moves that utilize the bypass connectivity between function units. In addition, a look-up table is required to translate function unit IDs for operations, which is crucial when constructing the bypass move between the source and target function unit during a data hazard.

The standard RISC-V instruction formats support a maximum of two register input operands. As a data hazard can occur in either or both of the register operands, the microcode unit requires two look-up tables that store bypass moves from function unit output ports to function unit input ports. During each move translation, the look-up table storing function unit IDs is read, and stored in a register. If a data hazard is detected for the following instruction, the function unit ID is passed to the two-dimensional bypass look-up table together with the opcode to construct the bypass move. After the bypass move is constructed, the operand move from the register file is discarded on the data hazard bus and the bypass move is inserted instead.

D. Instruction Fetch Unit

Instruction fetch (IF) unit is a crucial component in multi-instruction-set architecture design due to the differing instruction widths between the instruction sets. To minimize any hardware overheads, we choose to bundle multiple RISC-V instructions into a packet that corresponds to the 64-bit instruction width of the TTA instruction set mode. This way, two instructions are fetched every other cycle by the instruction fetch unit when running in RISC-V mode. In order to keep full compatibility with the RISC-V ISA, the RISC-V instruction set mode must support branches to non-bundle aligned addresses. While this does not cause any significant overhead in the hardware implementation, it results in excess instructions being fetched from the instruction memory hierarchy.

E. Mode Switch Support

To increase programmability of the processor, we implement mode switching with the use of custom instructions. For the RISC-V instruction set, we utilize one of the free opcode fields that are intended for custom instructions in order to maintain full compatibility with the RISC-V specification. In an alternative approach, the instruction set mode could be explicitly specified for each instruction packet, but this would result in a lower code density due to the additional encoding space. Furthermore, the acquired benefit from specifying the mode for each instruction packet would be minimal, since the instructions should be scheduled on a basic block level in any case in order to exploit an adequate amount of instruction level parallelism.

Since, the mode switch simply toggles the instruction set, we can leave other bits besides the opcode field to a don't care state to simplify decoding of the instruction. For the TTA, we add a custom operation into the control unit in order to generate a unique encoding for the mode switch instruction. As a means to preserve the correctness of the instruction fetch addresses during a mode switch, we add hardware into the instruction fetch stage that recognizes the TTA mode switch instruction. Thus, we can minimize the overhead of switching modes, as the instruction fetch unit can keep fetching valid instructions into the pipeline.

Due to the lack of hazards, switching the instruction set mode does not induce any overhead in terms of clock cycles in the pipeline, meaning the pipeline will continue to process instructions on the new instruction set the following cycle the mode switch instruction is detected. When switching from TTA to RISC-V mode, the mode switch instruction toggles the mode state register that controls the microcode output multiplexer. The following cycle after the mode switch instruction, a RISC-V instruction packet enters the microcode unit where it is translated and sequenced accordingly. When switching from RISC-V to TTA mode, alignment of the code sections must be considered due to different instruction word sizes. The programmer must guarantee that TTA instructions are aligned in memory, which results in the use of RISC-V no-operations in regions where RISC-V and TTA segments are placed side by side.

V. Multi-Instruction-Set Compilation Flow

Programming complex architectures manually can produce highly efficient code but requires high effort, which reduces the general-purpose usability of the architecture. For multi-instruction-set architectures, writing machine code becomes even less desirable as the programmer would have to know beforehand which instruction set is more efficient for the code region, which leads to a high programming effort. To this end, we propose a novel compiler that is able to automatically target code between RISC-V and TTA instruction sets based on static code analysis and microarchitectural models of the different instruction set modes. Thanks to automatic code generation, the architecture can be efficiently programmed with the C programming language without the user having to modify the application source code to control toggling between the instruction sets.

A. Mode Switch Granularity

Considering the granularity in which the instruction set modes are switched is important for utilizing the benefits of a flexible architecture. One approach is to generate code for the processor with the native compilers of each instruction set. This whole-program granularity simplifies code generation and works well for programs with limited complexity, but leaves the architecture underutilized for complex programs that benefit from both control-oriented and parallel execution.

Function-level granularity is the next logical level, which is also the approach chosen for the Thumb [10] mode. This already enables targeting different code regions with different instruction sets based on their characteristics, such as the amount of available instruction level parallelism. A function-level granularity, however, can be too coarse-grained for large functions that contain both control-oriented and parallel code regions that should be targeted with different instruction sets.

For fine-grained granularity, basic blocks could be targeted. In our approach, thanks to the custom instructions that explicitly toggle the mode, it is possible to support a basic block granularity for the mode switches. However, due to the overhead of switching modes, it is not ideal to target individual basic blocks that are executed only once. Loop regions form hot spots in the execution runs, which makes them important for static code analysis. The added benefit of targeting loop regions is that the generated code has a high static density when performance-wise less crucial code regions are executed with the more compact instruction set, which leads to smaller instruction images. Due to the lack of locality, it is encouraged to prefer a high code density over instruction level parallelism in non-recurring code regions to minimize the number of misses to the instruction cache.

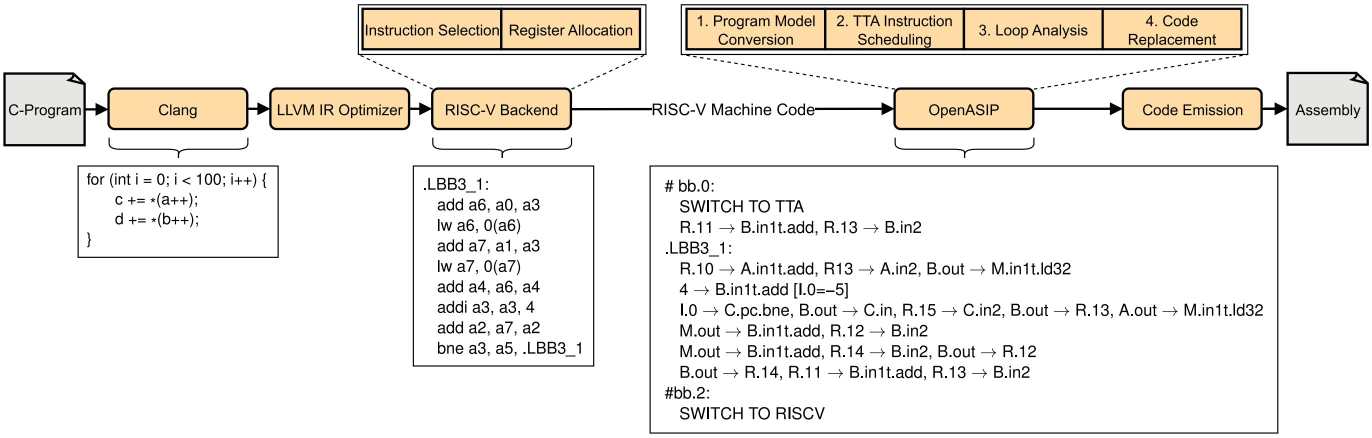
In our static code analysis, we focus on loop regions. The mode is switched to TTA mode by using a mode switch instruction in the *entry block* of the loop. A second mode switch is inserted into the *exit block* of the loop to return to RISC-V mode after the loop has been executed. This keeps the overhead of switching modes low, as multiple iterations are executed in a single mode switch cycle. With the mode switches inserted into the loop entry and exit blocks, the basic blocks contained in the loop body can be converted into TTA code with a guarantee that the mode is switched back to the RISC-V mode upon exiting the loop. Complex loops can have multiple entry and exit blocks, which increases the modifications to the loop structure.

B. Compiler

The compilation flow is built on top of the OpenASIP [29] RISC-V compiler that relies on the LLVM project. As described in Fig. 5, the compiler utilizes the standard LLVM C language frontend (Clang), middle-end optimizer and RISC-V backend, leaving all dual instruction-set specific targeting to the OpenASIP LLVM backend. Due to this structure, instruction selection and register allocation are already done in the RISC-V backend before OpenASIP is loaded into the compilation flow as a dynamic library. The hook into the OpenASIP LLVM backend is implemented as a pre-emit pass that iteratively passes machine code functions to the OpenASIP LLVM backend.

In order to retarget code to the TTA instruction set, the RISC-V backend generated machine code must be converted to the OpenASIP program object model for instruction scheduling. In this step, RISC-V machine code instructions are transformed into sequential code that describes the required operand moves and operations in the TTA programming model format. During this step, instructions can be freely constructed with moves without following the restrictions set by the RISC-V instruction formats.

In the second step, the sequential program object model is passed to the OpenASIP TTA instruction scheduler. During this stage, TTA-specific optimizations are applied, such as software bypassing, dead result elimination and operand sharing. In addition, the scheduler will exploit instruction level parallelism by scheduling the moves in parallel when possible, resulting in multiple operations being executed concurrently. The scheduler emits the function with the same control flow structure as it was



```
for (int i = 0; i < 100; i++) {
    c += *(a++);
    d += *(b++);
}
```

```
.LBB3_1:
   add a6, a0, a3
   lw a6, 0(a6)
   add a7, a1, a3
   lw a7, 0(a7)
   add a4, a6, a4
   addi a3, a3, 4
   add a2, a7, a2
   bne a3, a5, .LBB3_1
```

```
# bb.0:
   SWITCH TO TTA
   R.11 → B.in1t.add, R.13 → B.in2
.LBB3_1:
   R.10 → A.in1t.add, R13 → A.in2, B.out → M.in1t.ld32
   4 → B.in1t.add [I.0=–5]
   I.0 → C.pc.bne, B.out → C.in, R.15 → C.in2, B.out → R.13, A.out → M.in1t.ld32
   M.out → B.in1t.add, R.12 → B.in2
   M.out → B.in1t.add, R.14 → B.in2, B.out → R.12
   B.out → R.14, R.11 → B.in1t.add, R.13 → B.in2
#bb.2:
   SWITCH TO RISCV
```

Fig. 5. Compiler flow from a C program down to compiled code.

passed to help the further code analysis between the original RISC-V code.

In the third step, scheduling results are analyzed for loop regions in order to automatically choose which loops should be converted to the TTA instruction set. During the loop analysis, TTA scheduling results are compared against the original RISC-V machine code. Due to data dependent control flow, accurate cycle counts cannot be calculated for either instruction set. The analysis takes into account all basic blocks contained in the loop body and estimates cycle counts with a static profile. More accurate estimates could be acquired by passing the compiler dynamic profiling data. However, we argue that this would decrease the general purpose applicability of the compilation flow and make the code generation quality data dependent. It is also clear that with heavily control-oriented loops, exploiting instruction level parallelism becomes difficult, which favors the use of the RISC-V instruction set and makes the accuracy of the performance model less important.

In the final stage, loop regions to be executed on the TTA are chosen automatically by the compiler based on the static analysis. The compiler will discard a loop if i) cycle count estimations are higher on TTA than RISC-V mode based on the static analysis or ii) the scheduled loop code does not fit in the instruction cache. It is important to inspect the performance estimations, because loops with a large amount of data and control dependencies can execute slower when converted to TTA due to RISC-V's more efficient execution of control instructions. The size of the scheduled loop code is also an important factor, as converting a loop that does not fit into the instruction cache will cause an overhead in terms of performance and energy consumption due to the increased cache misses. The code example in Fig. 5 is converted to TTA code because it is compact enough for the instruction cache and the scheduled code is faster than the original RISC-V machine code.

With nested loops, the loop analyzer can choose to convert the entire nested loop or an arbitrary number of the inner loops contained inside the top loop. The chosen loop structures are converted to the TTA instruction set as described in Section V-A. In order to generate executables from the generated code, the TTA code regions are emitted amidst the RISC-V assembly as a raw instruction binary. This way, the standard RISC-V toolchain can be used to form the final executable for the processor.

Sufficient code generation quality can be achieved using the RISC-V backend for both instruction sets, as the TTA instruction set of Dual-IS closely follows the standard RISC-V instruction set architecture. Using a larger register file for the TTA mode would require reimplementing register allocation on the OpenASIP side to avoid unnecessary spilling, but it would enable using the OpenASIP register renamer that can resolve *anti-dependencies* during instruction scheduling to add opportunities for exploitation of ILP.

VI. Evaluation

After integrating the RTL generation and compilation flow into OpenASIP, we used the toolset to generate the RTL and program binaries for both the Dual-IS processor and the baseline designs. For evaluation, we synthesized the designs with a 28 nm ASIC technology and ran the synthesized netlists in gate-level simulations to generate cycle counts and switching activity information for power estimation with Synopsys Design Compiler. We evaluated the processors with the CHStone [7] benchmark suite, EEMBC Coremark [8] and the Opus audio codec [9] in order to focus on embedded applications using integer computation, which is the targeted use case for the evaluated processors.

A. Evaluated Designs

With OpenASIP, we generated three designs: *Dual-IS*, *OA-RISCV* and *TTA*. As an external baseline, we used CV32E40P (formerly known as RI5CY [30]) version 1.3.1 [31]. Table I describes architectural and microarchitectural properties of the processors. Dual-IS implements both a TTA and a RISC-V (RV32IM) instruction set. It has a peak 3 operations per cycle TTA mode and a single-issue in-order RISC-V mode. OA-RISCV implements a single-issue RV32IM datapath via microcoded control logic. It has the same microarchitectural functionalities as Dual-IS in RISC-V mode. TTA has the

TABLE I
FEATURES OF THE EVALUATED PROCESSORS

	CV32E40P	Dual-IS	OA-RISCV	TTA
Issue width	1	1/3	1	3
RF ports	3r2w	2r1w	2r1w	2r1w
RF size	32	32	32	32
Branch lat.	3	4	4	4
ICache assoc.	DM	DM	DM	DM
ICache size	0.5-2 kB	0.5-2 kB	0.5-2 kB	0.5-2 kB

same datapath as Dual-IS but utilizes absolute branch addressing and only implements the TTA instruction set without any RISC-V-specific instructions. All of the three generated designs have the same pipeline configuration without any additional latency added to support the RISC-V or mode switch hardware.

As an external baseline, we used CV32E40P (formerly known as RI5CY [30]) version 1.3.1 [31] that is optimized, similar to the proposed Dual-IS architecture, for embedded use cases, such as *internet of things* (IoT) endpoint devices. The pipelines of the generated cores modeled the structure of the CV32E40P core as closely as possible, with some exceptions. The cores generated with OpenASIP had a two-cycle multiplier, while the CV32E40P used a single cycle multiplier. The difference was due to the exposed datapath structure that would cause an unnecessarily long combinatorial path to the design if the multiplier unit (supporting mulh operations) was directly connected to the interconnect. This would cause a combinatorial path that would go through the register file read, interconnect multiplexers, multiplier execute and ending in the multiplier result register. In the CV32E40P, the multiplier path went through the multiplier execute and register file writeback. Similar pipeline difference was implemented with control flow instructions. With OpenASIP generated designs, signals in the instruction memory interface were registered, as connecting them to the interconnection network would cause a long combinatorial path. This increased the delay of taken branch instructions by one cycle compared to the CV32E40P core.

Another important difference was in the load operation pipelines. All implementations had a load operation latency of two clock cycles. When running on the TTA instruction set, these operations had a programmer-visible delay slot that the compiler could fill with operations that have no dependencies to the load result. CV32E40P pipelines these operations in hardware and used a second register file write port to support the load store unit pipeline. With the OpenASIP RISC-V implementations, multi-cycle operations were always stalled to fit the operation latency, which happened both with multiply and load operations.

The register file implementation differed due to the CV32E40P load store unit pipeline and the custom instruction set extensions, which increased the register file port amount to three read ports and two write ports. OpenASIP-generated cores used a traditional 1w2r configuration, and the TTA instruction sets exploited instruction level parallelism via the TTA programming model without adding register file ports. It is notable that a traditional "operation triggered" 3-issue VLIW design with support for similar parallel execution as Dual-IS (ALU+LSU+JUMP / 2xALU+JUMP) would require a minimum of 2 write and 4 read ports in the register file. Other differences were extra functionalities not needed to run the benchmarks, such as interrupt support, debuggers and custom instructions that were not implemented in the OpenASIP-generated cores.

TABLE II
AREA BREAKDOWN OF THE CORES

	CV32E40P	Dual-IS	OA-RISCV	TTA
Area (μm^2)	20600	16200	12600	14400
RFs	31%	27%	33%	31%
IFetch	7%	13%	10%	8%
FUs	37%	44%	47%	48%
Decoder	12%	4%	2%	4%
CS registers	11%	-	-	-
IC	-	9%	5%	9%
Microcode	-	3%	2%	-

We added an L1 instruction cache [32] to the designs to lower instruction stream energy. The cache implemented a *direct-mapped* (DM) structure to mimic the filter cache [33] approach, with a line size of 64 bytes that is a reasonable choice for the evaluated cache sizes of 512 and 2048 bytes [34]. The instruction cache was connected to a 1 MB SRAM (enough to fit all benchmarks) that consisted of 64-byte wide banks, which enabled to fill a cache line in one fetch cycle, making the miss penalty two cycles. On the data side, the load store units were directly connected to a 256 kB SRAM that was large enough for all benchmarks.

B. *Synthesis Results*

With the OpenASIP generated processors, Dual-IS, OA-RISCV and TTA, the critical path formed in the two-cycle multiply unit limiting the maximum clock frequency to 1.61 GHz. With the CV32E40P, using an instruction cache size of 512 bytes caused a critical path in the datapath, which limited the maximum clock frequency to 1.54 GHz. Using a 2 kB instruction cache caused the critical path to move to the instruction fetch stage due to combinatory paths from the core propagating into the instruction cache, which limited the maximum clock frequency to 1.35 GHz.

The areas of the cores synthesized with the 512 byte cache configuration are listed in Table II. Compared to the single-ISA TTA core, Dual-IS used approximately 13% more area. This was affected by the more complex instruction fetch and the additional hardware in the microcode unit. In addition, the extra hardware removed timing slack from non-critical paths, which lowered the usage of smaller low power cells, increasing area even further. The OA-RISCV core had the smallest area usage due to the simple single-issue datapath, utilizing 22% less area than Dual-IS. CV32E40P utilized the most area due to the more complex register file and additional hardware that was not implemented in the other cores, such as control and status (CS) registers, which caused the CV32E40P core to utilize 27% more area than Dual-IS.

TABLE III
ANALYSIS OF DUAL-IS PROGRAM EXECUTION

	TTA Execution*	Mode Toggles	Total Cycles*
adpcm	51%	800	96903
aes	18%	226	33832
blowfish	12%	524	777166
gsm	45%	10	18277
mips	2%	4	26823
motion	96%	6	11318
sha	98%	1544	664650
jpeg	11%† / 17%‡	1074† / 1356‡	2721285† / 2632802‡
coremark	27%	642	570497
opus	24%† / 45%‡	494† / 664‡	962086† / 859560‡

*Stall cycles caused by instruction cache misses are discarded.
† 512 instruction cache.
‡ 2 kB instruction cache.

C. Performance

Table III describes the amount of clock cycles Dual-IS executed in TTA mode with different benchmarks, as well as the amount of times the instruction set was toggled during execution. Stalls cycles caused by misses to the instruction cache were discarded to describe the use of the instruction sets from a computational viewpoint, otherwise instruction memory hierarchy changes would affect mode utilization even if the program remained the same. It is clear that the more the benchmark relies on concentrated (parallel) loop kernels, the more time the processor will execute in TTA mode due to the compilation flow targeting loop structures that also form hot spots in the program execution. Except for jpeg and opus, the compiler did not convert more loops with the larger 2 kB cache compared to the 512 byte instruction cache that is capable of storing a 64 instruction loop body in TTA mode. On average, the TTA instruction set was utilized 40% of the time. Out of the ten benchmarks, sha and motion were primarily run in TTA mode, while mips favored the RISC-V instruction set due to its heavily control-oriented kernel structure. The ratio between amount of mode toggles and total amount of execution cycles showed that the mode switching overhead was insignificant compared to the total run time as described in Table III.

Run times relative to the CV32E40P baseline with different instruction cache sizes are listed in Fig. 6. With the 512 byte instruction cache, the OA-RISCV core had, on average, 13% higher run time than the CV32E40P due to the bottleneck caused by the lack of a load operation pipeline and one cycle longer multiply and branch latency. The same execution bottlenecks existed for Dual-IS when running code in RISC-V mode due to the similarities in the microarchitecture. With the larger cache size, OA-RISCV had, on average, the same performance due to the more pipelined structure, which prevented the critical path from moving to the instruction fetch stage. Dual-IS benefited from the static multi-issue capabilities, which reduced run times 5% and 18%, on average, compared to CV32E40P.

Compared to the single-ISA TTA instruction set, Dual-IS was 0.4% faster both with the 512 byte and 2 kB instruction cache. The speed-up was mainly achieved by fewer misses to the instruction cache due to a higher code density, as well as the more efficient execution of control code thanks to the RISC-V mode. With blowfish, the single-ISA TTA processor was significantly faster due to one of the program hot spots consisting of a large loop that would require a 4-kB instruction cache to fit the loop body. The Dual-IS compiler discarded such loops and executed them in RISC-V mode to eliminate the bad energy-delay tradeoff caused by cache misses. Motion was a clear outlier in the benchmarks due to a 2 kB array copy assignment which took a majority of the run time. The TTA instruction set aggressively utilized the delay slots to fill the copy operation, which lowered the loop body execution by half per iteration compared to the OA-RISCV core. Mips did not favor the use of the TTA instruction set due to the control-oriented kernel structure, which led to similar results as with the single-ISA RISC-V instruction sets but lowered the run time by 25% on average compared to the single-ISA TTA processor. Similarly, in coremark, the acquired benefit from the TTA mode was limited due to the lack of available instruction level parallelism. The opus application included encoding and decoding an audio sample, which experienced a balanced use between the RISC-V and TTA instruction sets.

D. Code Density

Another interesting phenomenon was the code density benefit acquired from the multi-instruction-set approach. Table IV describes the sizes of the instruction images with different architectures. With the 2 kB instruction cache, Dual-IS needed, on average, 21% larger instruction images than the single-ISA RISC-V instruction set. However, the Dual-IS had, on average, 43% smaller instruction images than the single-ISA TTA instruction set. The significantly lower code size is especially important in resource constrained embedded systems that have small memories, which can limit the set of applications the system can run. The multi-instruction-set architecture approach enables restricting the use of the static multi-issue capabilities to the degree where the generated instruction image can fit the instruction memory, which adds more flexibility to the system.

E. Energy Efficiency

For energy evaluation, we ran the synthesized netlists in gate-level simulation to produce *switching activity interchange format* (SAIF) files that were passed to Design Compiler to generate power estimates for the core and the instruction cache. For the instruction and data memories, we used Cacti [35] to generate access energy estimations that were used to estimate energy consumption based on the access traces gathered in simulations. Energy consumption relative to the CV32E40P baseline with different instruction cache sizes is described in Fig. 7.

The main benefit of Dual-IS in terms of energy compared to the single-ISA TTA processor was a lower instruction stream energy consumption that was achieved by a reduced amount of misses to the instruction cache. Compared to the single-ISA TTA processor, Dual-IS had a 40% smaller instruction stream (instruction cache + instruction SRAM) energy consumption

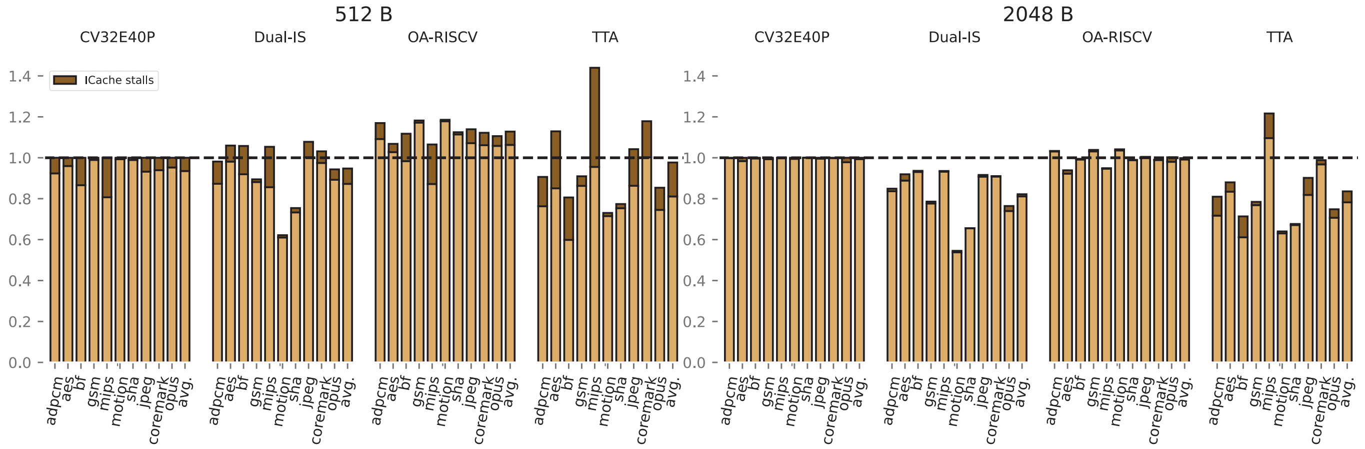


Fig. 6. Run times relative to the CV32E40P baseline with different instruction cache sizes.

TABLE IV
CODE SIZES

Benchmark	RISC-V	Dual-IS	TTA
adpcm	11 kB	14 kB	21 kB
aes	20 kB	21 kB	38 kB
blowfish	11 kB	11 kB	19 kB
gsm	5 kB	6 kB	12 kB
mips	2 kB	2 kB	7 kB
motion	5 kB	8 kB	9 kB
sha	3 kB	4 kB	7 kB
jpeg	14 kB	16†/18‡ kB	31 kB
coremark	12 kB	15 kB	24 kB
opus	294 kB	311†/315‡ kB	775 kB

† 512 instruction cache.
‡ 2 kB instruction cache.

with a 512 byte cache and 45% with the 2 kB cache. Compared to the CV32E40P processor, the instruction stream energy was 17% and 12% higher on average due to more bits being fetched from the instruction cache and memory due to lower code density. The longer pipeline structure caused the RISC-V mode to fetch more instructions compared to the CV32E40P core, which increased power consumption of the instruction stream further.

The single-ISA TTA core was the most energy efficient out of the evaluated designs in terms of energy consumed by the core alone. It boils down to the lack of dynamic hardware because compared to Dual-IS the main power consumption difference was in the instruction fetch unit and microcode, as seen in the power breakdown in Table V. Compared to the single-ISA RISC-V cores, the TTA had a marginally more complex datapath with more connectivity between the function units and an additional reduced ALU for exploitation of ILP, which increased power consumption. On average, Dual-IS core consumed 22% more energy compared to the single-ISA TTA core. Compared to the OA-RISCV, Dual-IS consumed only 5% more energy and 17% less than the CV32E40P. The higher energy consumption of the CV32E40P core compared to OA-RISCV was driven by the additional hardware, mainly the control and status registers that were not implemented in the other cores.

In terms of total energy consumption of the system, Dual-IS consumed 6% more energy with the 512 byte cache and 3% more with the 2 kB cache compared to the OA-RISCV processor. Compared to CV32E40P, the energy consumption was 4% higher with the 512 byte cache and 4% lower with the 2 kB cache. The difference to the single-ISA TTA processor was more significant due to the high instruction stream overhead of TTA, which made Dual-IS 29% and 26% more energy efficient than the single-ISA TTA processor.

Energy delay product (EDP) [36] is a metric that describes the trade-off between energy consumption and performance. EDP results relative to the CV32E40P baseline are listed in Fig. 8. Dual-IS had the lowest average EDP out of all processors with all configurations. Compared to the single-ISA TTA processor, the difference was the most significant due to similar level of performance but reduced instruction stream energy, which led Dual-IS to have 29% lower EDP with a 512 byte cache and 26% lower with the 2 kB cache. Compared to the OA-RISCV core, Dual-IS had 10% and 14% lower EDP. With the CV32E40P core, the difference was 1% and 20%.

F. Comparison to Previous Work

Table VI lists properties of published RISC-V implementations that are targeted for energy-efficient embedded applications. As the processors were implemented with different process technologies and estimated under different operating voltages, we used a scaling model [37] to scale them to 32 nm 0.95 V that is the closest point in the model compared to our 28 nm 0.95 V evaluation. Based on the power estimations, Dual-IS is approximately 3 times more power-efficient than RISC-V^2 at a similar operating frequency. Compared to the energy-optimized zero-riscy, Dual-IS has 5 times higher operating frequency but 35 times higher power consumption.

HAMSA-DI [38] is an extended version of the CV32E40P core with improved microarchitecture that supports in-order dual-issue execution. Adding a second issue slot to the processor increased the area by 65% compared to the baseline CV32E40P. A 58% performance improvement and 22% energy consumption reduction are reported over the CV32E40P core

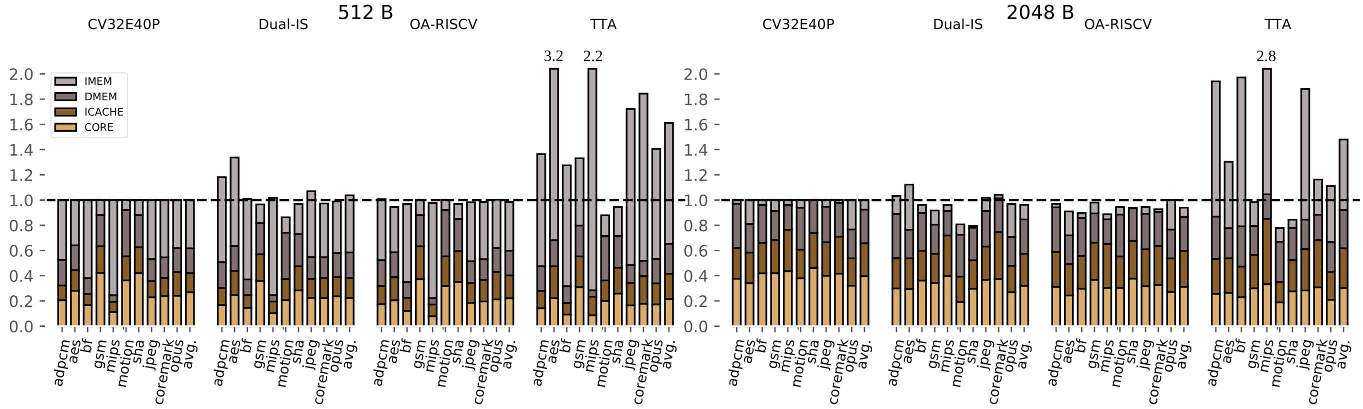


Fig. 7. Energy consumption relative to the CV32E40P baseline with different instruction cache sizes.

for the embench [39] nettle-aes benchmark that has a high amount of parallelism. Our measurements showed a 60% performance increase and a 38% reduction in the core's energy consumption over the CV32E40P when running nettle-aes on Dual-IS, while utilizing 21% less area. We evaluated the average power consumption of the HAMSA-DI core based on the nettle-aes, coremark and dot product benchmarks reported in the publication and scaled them based on our CV32E40P evaluation. The estimated average power consumption was approximately 10 mW, which is 40% higher than with Dual-IS. In HAMSA-DI, additional hardware was added to dynamically schedule two instructions per cycle and the register file was extended to feature five read and three write ports. In Dual-IS, the hardware complexity is lower thanks to the static exposed datapath multi-issue mode, which does not require dynamic hardware for instruction scheduling or extensions to the register file structure.

G. Discussion

For the evaluation, we used the CHStone benchmark suite that contains complex enough kernels that are beneficial to divide into different program regions for the instruction sets via static code analysis. Simple kernels such as matrix multiplication are not ideal for multi-instruction-set architectures because the limited program complexity would cause most of the program to be run on one instruction set while still carrying the overhead of the multi-instruction-set hardware, which would favor the use of a traditional processor or a fixed function accelerator.

Instruction stream hierarchy has a major impact on the system-level performance and energy efficiency, which plays a key role in the comparisons due to differing code densities between the instruction sets. The filter cache approach used for evaluation is simple but enables high energy efficiency for loop kernels that can fit the cache. Using a loop buffer [40] would increase efficiency for small loop kernels, which is ideal for the TTA instruction set as its use is targeted for loop regions and would complement the concepts introduced in this work.

In the scope of this work, the proposed multi-instruction-set architecture could be seen as an instruction compression [41] scheme that switches to a higher-level programming interface to improve code density. Traditional instruction compression methods such as dictionary compression or compressed instruction subsets, for example, the RISC-V C extension, are orthogonal approaches and could be applied on top of this work to further improve static compression ratios and instruction stream energy consumption.

TABLE V
Power Breakdown of the Cores

	CV32E40P	Dual-IS	OA-RISCV	TTA
RFs	10%	13%	14%	15%
IFetch	18%	22%	24%	18%
FUs	19%	28%	27%	35%
Microcode	-	6%	8%	-
Decoder	23%	16%	11%	18%
CS registers	22%	-	-	-
IC	-	12%	13%	12%

The main code generation bottleneck is the limitation to loop region analysis. With more complex global analysis or dynamic profiling, frequently executed code regions could be retargeted on a basic-block level outside of loop regions to achieve higher code generation quality. We also observed that due to the complex TTA instruction scheduling heuristics, at times better scheduling results were achieved when using the RISC-V backend for instruction selection compared to the OpenASIP-generated compiler backend. The main difference is the lack of multiple addressing modes for memory operations, which restricts the RISC-V backend to use only the base + offset addressing scheme.

VII. Future Work

The architecture and the implementation of the proof-of-concept processor were targeted for embedded applications, such as low-power IoT, where energy efficiency is an important optimization factor. This was also reflected in the evaluation that used applications focusing on integer computation. In the future, we plan to research utilizing the Dual-IS concept on a high-performance use case, which requires a higher degree of parallelism and floating point computation.

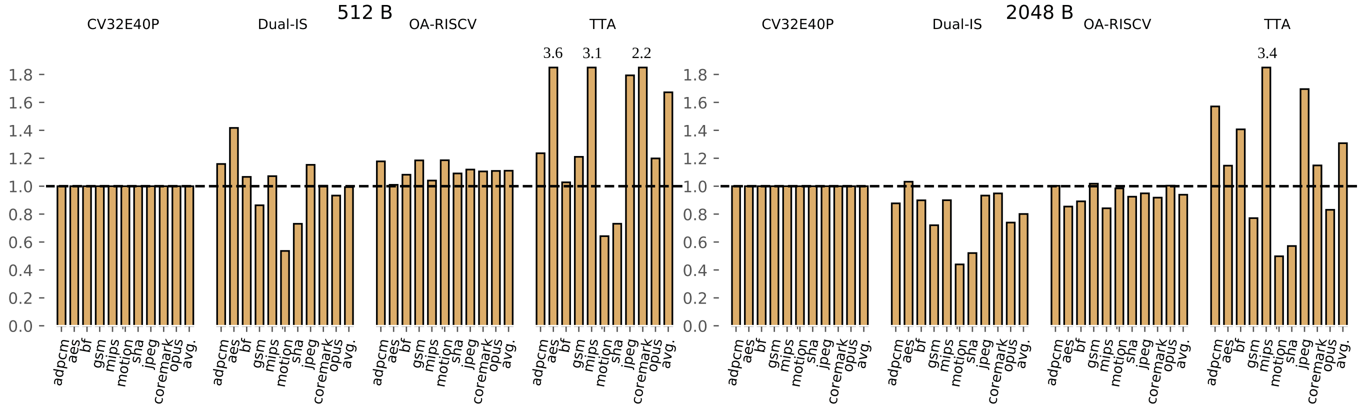


Fig. 8. Energy delay product relative to the CV32E40P baseline with different instruction cache sizes.

TABLE VI
COMPARISON OF RISC-V IMPLEMENTATIONS FOR ENERGY-EFFICIENCY CRITICAL EMBEDDED USE CASES

Processor	**Issues**	**Pipeline**	**Process [nm]**	**Area [mm^2]**	**Frequency [GHz]**	**Power [mW]**
Dual-IS	single/triple	in-order	28	0.016**	1.6**	7**
OA-RISCV	single	in-order	28	0.013**	1.6**	6**
zero-riscy [42]	single	in-order	65	0.027* ∣ 0.008†	0.2* ∣ 0.3†	0.4* ∣ 0.2†
CV32E40P [30] [31]	single	in-order	28	0.021**	1.4**	8**
HAMSA-DI [38]	dual	in-order	16	0.034‡	1.4‡	10‡
RISC-V^2 [43]	dual	out-of-order	45	0.24* ∣ 0.11†	1.0* ∣ 1.7†	23† ∣ 20*

*Based on cited publications.
**Based on our own measurements.
† Scaled to 32 nm 0.95 V.
‡ 28 nm estimation based on [38] and our measurements.

The RISC-V microarchitecture would have to be extended for high performance computing, which calls for superscalar and out-of-order execution to achieve high performance in code regions that are not optimal for static multi-issue execution. By following the multi-instruction-set architecture approach, sufficient code generation quality could be achieved without using complex global code optimizations, such as trace scheduling [44], as code with limited parallelism can be targeted for the dynamic multi-issue mode and code regions with a high amount of static ILP can be converted to the VLIW instruction set. Even in HPC use cases, the proposed dual instruction-set approach could yield benefits by disabling the complex control hardware that consumes a significant amount of energy in out-of-order superscalar processors [45] or enable a higher degree of parallelism than the superscalar mode thanks to the better scaling of the exposed datapath architecture.

VIII. CONCLUSION

In this article, we introduced a dual instruction-set architecture, “Dual-IS”, that supports both a TTA and the RISC-V instruction set via a lightweight microcode hardware unit. To better utilize the flexible architecture, we implemented a compiler that automatically targets code for both instruction sets from a high-level programming language. The compiler performs static code analysis to choose loop regions that are beneficial to execute on the TTA instruction set based on the instruction cache size and scheduling results.

We implemented the proposed architecture in RTL and compared it against multiple baseline designs with different instruction cache configurations post-synthesis. The additional hardware of Dual-IS did not affect the maximum clock frequency but caused 13% higher area utilization compared to a single-ISA TTA core with similar datapath resources. Dual-IS brought significant instruction stream energy savings due to the fine-grained use of both the compact RISC-V and multi-issue TTA instruction sets, which lowered the instruction stream energy consumption 45% on average in the best design point. The reduced instruction stream energy footprint translated into a total system energy saving of 26% compared to a single-ISA TTA processor. In terms of energy-delay product, Dual-IS had 26% lower energy delay product on average compared to a single-ISA TTA processor and 20% lower than the CV32E40P.

The proposed approach enables the benefits of an exposed datapath in exploitation of static instruction level parallelism without expensive dynamic hardware of out-of-order superscalar processors while minimizing the instruction stream energy impact in serial regions with the compact RISC-V instruction set. The results show that the Dual-IS architecture can significantly improve energy efficiency without inducing significant hardware overhead or reducing performance compared to a single-ISA processor.

ACKNOWLEDGMENT

The authors are grateful for the funding from Academy of Finland (decisions #331344 and #353199), which made this research possible.

REFERENCES

[1] J. L. Hennessy and D. A. Patterson, "A new golden age for computer architecture," *Commun. ACM*, vol. 62, no. 2, pp. 48–60, 2019.

[2] T. N. Theis and H.-S. P. Wong, "The end of moore's law: A new beginning for information technology," *Comput. Sci. Eng.*, vol. 19, no. 2, pp. 41–50, 2017.

[3] M. Horowitz, "1.1 computing's energy problem (and what we can do about it)," in *Proc. IEEE Int. Solid-State Circuits Conf. Dig. Tech. Papers (ISSCC)*, Piscataway, NJ, USA: IEEE, 2014, pp. 10–14.

[4] D. Boggs, G. Brown, N. Tuck, and K. Venkatraman, "Denver: Nvidia's first 64-bit ARM processor," *IEEE Micro*, vol. 35, no. 2, pp. 46–55, Mar./Apr. 2015.

[5] M. B. Taylor, "Is dark silicon useful? Harnessing the four horsemen of the coming dark silicon apocalypse," in *Proc. 49th Annu. Des. Automat. Conf.*, 2012, pp. 1131–1136.

[6] K. Hepola, J. Multanen, and P. Jääskeläinen, "Dual-IS: Instruction set modality for efficient instruction level parallelism," in *Proc. 35th Int. Conf. Archit. Comput. Syst. (ARCS)*, Heilbronn, Germany. Cham, Germany: Springer, Sep. 13–15, 2022, pp. 17–32.

[7] Y. Hara, H. Tomiyama, S. Honda, H. Takada, and K. Ishii, "Chstone: A benchmark program suite for practical c-based high-level synthesis," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, Piscataway, NJ, USA: IEEE, 2008, pp. 1192–1195.

[8] The Embedded Microprocessor Benchmark Consortium. "Coremark benchmark." EEMBC. Accessed: Oct. 17, 2023. [Online]. Available: https://www.eembc.org/coremark/

[9] "Opus audio codec." Opus. Accessed: Oct. 25, 2023. [Online]. Available: https://opus-codec.org/

[10] S. Segars, K. Clarke, and L. Goudge, "Embedded control problems, Thumb, and the ARM7TDMI," *IEEE Micro*, vol. 15, no. 5, pp. 22–30, Oct. 1995.

[11] A. Shrivasta and N. Dutt, "Energy efficient code generation exploiting reduced bit-width instruction set architectures (rISA)," in *Proc. Asia South Pacific Des. Automat. Conf. (ASP-DAC) (IEEE Cat. No. 04EX753)*, Piscataway, NJ, USA: IEEE, 2004, pp. 475–477.

[12] A. Halambi, A. Shrivastava, P. Biswas, N. Dutt, and A. Nicolau, "An efficient compiler technique for code size reduction using reduced bit-width ISAs," in *Proc. Des., Automat. Test Europe Conf. Exhib.*, Piscataway, NJ, USA: IEEE, 2002, pp. 402–408.

[13] S. Lee, J. Lee, C. Y. Park, and S. L. Min, "Selective code transformation for dual instruction set processors," *ACM Trans. Embedded Comput. Syst. (TECS)*, vol. 6, no. 2, pp. 10–es, 2007.

[14] H. Karaki, H. Akkary, and S. Shahidzadeh, "X86-ARM binary hardware interpreter," in *Proc. 18th IEEE Int. Conf. Electron., Circuits, Syst.*, Piscataway, NJ, USA: IEEE, 2011, pp. 145–148.

[15] T.-J. Lin et al., "A unified processor architecture for RISC & VLIW DSP," in *Proc. 15th ACM Great Lakes Symp. VLSI*, 2005, pp. 50–55.

[16] Y. Hou et al., "FuMicro: A fused microarchitecture design integrating in-order superscalar and VLIW," vol. 2016, 2016, Art. no. 8787919, doi: 10.1155/2016/8787919.

[17] D. Suh, K. Kwon, S. Kim, S. Ryu, and J. Kim, "Design space exploration and implementation of a high performance and low area coarse grained reconfigurable processor," in *Proc. Int. Conf. Field-Programmable Technol.*, Piscataway, NJ, USA: IEEE, 2012, pp. 67–70.

[18] N. R. Miniskar, R. R. Patil, R. N. Gadde, Y.-c. R. Cho, S. Kim, and S. H. Lee, "Intra mode power saving methodology for CGRA-based reconfigurable processor architectures," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, Piscataway, NJ, USA: IEEE, 2016, pp. 714–717.

[19] A. Venkat, H. Basavaraj, and D. M. Tullsen, "Composite-ISA cores: Enabling multi-ISA heterogeneity using a single ISA," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Piscataway, NJ, USA: IEEE, 2019, pp. 42–55.

[20] J. Hoogerbrugge and H. Corporaal, "Transport-triggering vs. operation-triggering," in *Proc. 5th Int. Compiler Construction Conf. (CC)*, Edinburgh, U.K., Springer, Apr. 7–9, 1994, pp. 435–449.

[21] P. Jääskeläinen, H. Kultala, T. Viitanen, and J. Takala, "Code density and energy efficiency of exposed datapath architectures," *J. Signal Process. Syst.*, vol. 80, pp. 49–64, Jul. 2015.

[22] H. Kultala, J. Multanen, P. Jääskeläinen, T. Viitanen, and J. Takala, "Impact of operand sharing to the processor energy efficiency," in *Proc. 18th CSI Int. Symp. Comput. Archit. Digit. Syst. (CADS)*, Piscatway, NJ, USA: IEEE, 2015, pp. 1–6.

[23] V. Guzma, P. Jääskeläinen, P. Kellomäki, and J. Takala, "Impact of software bypassing on instruction level parallelism and register file traffic," in *Proc. 8th Int. Embedded Comput. Syst., Archit., Model., Simul., Workshop (SAMOS)*, Samos, Greece. Berlin, Germany: Springer, Jul. 21–24, 2008, pp. 23–32.

[24] J. Hoogerbrugge and H. Corporaal, "Register file port requirements of transport triggered architectures," in *Proc. 27th Annu. Int. Symp. Microarchitecture*, 1994, pp. 191–195.

[25] H. Corporaal, "TTAs: Missing the ILP complexity wall," *J. Syst. Archit.*, vol. 45, nos. 12–13, pp. 949–973, 1999.

[26] H. Kultala et al., "Lordcore: Energy-efficient OpenCL-programmable software-defined radio coprocessor," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 27, no. 5, pp. 1029–1042, May 2019.

[27] J. Multanen, H. Kultala, P. Jääskeläinen, T. Viitanen, A. Tervo, and J. Takala, "Lotta: Energy-efficient processor for always-on applications," in *Proc. IEEE Int. Workshop Signal Process. Syst. (SiPS)*, Piscatway, NJ, USA: IEEE, 2018, pp. 193–198.

[28] A. S. Waterman, "Design of the RISC-V instruction set architecture," Univ. California, Berkeley, Berkeley, CA, USA, Technical Report No. UCB/EECS-2016-1, 2016.

[29] K. Hepola, J. Multanen, and P. Jääskeläinen, "OpenASIP 2.0: Co-design toolset for RISC-V application-specific instruction-set processors," in *Proc. IEEE 33rd Int. Conf. Appl.-Specific Syst., Architect. Processors (ASAP)*, Piscatway, NJ, USA: IEEE, 2022, pp. 161–165.

[30] M. Gautschi et al., "Near-threshold RISC-V core with DSP extensions for scalable IoT endpoint devices," *IEEE Trans. Very Large Scale Integration (VLSI) Syst.*, vol. 25, no. 10, pp. 2700–2713, Oct. 2017.

[31] "CV32E40P." GitHub. Accessed: May 25, 2023. [Online]. Available: https://github.com/openhwgroup/cv32e40p

[32] V. Saljooghi, A. Bardizbanyan, M. Själander, and P. Larsson-Edefors, "Configurable RTL model for level-1 caches," in *Proc. NORCHIP*, Piscataway, NJ, USA: IEEE, 2012, pp. 1–4.

[33] J. Kin, M. Gupta, and W. H. Mangione-Smith, "The filter cache: An energy efficient memory structure," in *Proc. 30th Annu. Int. Symp. Microarchitecture*, Piscataway, NJ, USA: IEEE, 1997, pp. 184–193.

[34] A. J. Smith, "Line (block) size choice for CPU cache memories," *IEEE Trans. Comput.*, vol. 100, no. 9, pp. 1063–1075, Sep. 1987.

[35] S. Li, K. Chen, J. H. Ahn, J. B. Brockman, and N. P. Jouppi, "CACTI-P: Architecture-level modeling for SRAM-based structures with advanced leakage reduction techniques," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des. (ICCAD)*, Piscataway, NJ, USA: IEEE, 2011, pp. 694–701.

[36] M. Horowitz, T. Indermaur, and R. Gonzalez, "Low-power digital design," in *Proc. IEEE Symp. Low Power Electron.*, Piscataway, NJ, USA: IEEE, 1994, pp. 8–11.

[37] A. Stillmaker and B. Baas, "Scaling equations for the accurate prediction of CMOS device performance from 180 nm to 7 nm," *Integration*, vol. 58, pp. 74–81, Jun. 2017.

[38] Y. Kra, Y. Shoshan, Y. Rudin, and A. Teman, "HAMSA-DI: A low-power dual-issue RISC-V core targeting energy-efficient embedded systems," *IEEE Trans. Circuits Syst. I, Reg. Papers*, early access, 2023.

[39] "Embench™: A modern embedded benchmark suite." FOSSI Foundation. Accessed: Nov. 10, 2023. [Online]. Available: https://www.embench.org/

[40] L. H. Lee, B. Moyer, and J. Arends, "Instruction fetch energy reduction using loop caches for embedded applications with small tight loops," in *Proc. Int. Symp. Low Power Electron. Des.*, 1999, pp. 267–269.

[41] C. Lefurgy, P. Bird, I.-C. Chen, and T. Mudge, "Improving code density using compression techniques," in *Proc. 30th Annu. Int. Symp. Microarchitecture*, Piscataway, NJ, USA: IEEE, 1997, pp. 194–203.

[42] P. D. Schiavone et al., "Slow and steady wins the race? A comparison of ultra-low-power RISC-V cores for Internet-of-Things applications," in *Proc. 27th Int. Symp. Power Timing Model., Optim. Simul. (PATMOS)*, Piscataway, NJ, USA: IEEE, 2017, pp. 1–8.

[43] K. Patsidis, C. Nicopoulos, G. C. Sirakoulis, and G. Dimitrakopoulos, "RISC-V 2: A scalable RISC-V vector processor," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, Piscataway, NJ, USA: IEEE, 2020, pp. 1–5.

[44] Fisher, "Trace scheduling: A technique for global microcode compaction," *IEEE Trans. Comput.*, vol. C-30, no. 7, pp. 478–490, Jul. 1981.

[45] O. Chatzopoulos, G. Papadimitriou, W. S. Wong, and D. Gizopoulos, "Energy efficiency of out-of-order CPUs: Comparative study and microarchitectural hotspot characterization of RISC-V designs," in *Proc. IEEE Int. Symp. Workload Characterization (IISWC)*, 2023, pp. 216–220.

Kari Hepola received the M.Sc. degree in electrical engineering from Tampere University, Tampere, Finland, in 2022, where he is currently working toward the D.Sc. degree. His current research interests include multiinstruction-set architectures and application-specific instruction-set processors, with the goal of increasing energy efficiency and flexibility of software programmable processors.

Joonas Multanen received the M.Sc. degree in electrical engineering from Tampere University of Technology, in 2015, and the Ph.D. degree from Tampere University (TAU), Finland, in 2021. He is currently a Postdoctoral Researcher with the Faculty of Information Technology and Communication Sciences, TAU. His research interests include energy efficient computer architectures.

Pekka Jääskeläinen is an Associate Professor, and leads the Customized Parallel Computing Research group of Tampere University and works for Intel as a Principal Engineer. He has been contributing to heterogeneous platform customization and programming topics since early 2000s. In addition to his academic publication activities, he maintains or actively contributes to multiple heterogeneous computing related open source projects such as OpenASIP, portable computing language, and chipStar. He is interested in methods and tools to reduce the engineering effort in design and programming of diverse heterogeneous platforms, and generally in hardware and compiler techniques to reduce the energy consumption of programmable processors.