Dual-IS: Instruction Set Modality for Efficient Instruction Level Parallelism

Kari Hepola, Joonas Multanen, and Pekka Jääskeläinen

Tampere University, Tampere, Finland
{kari.hepola, joonas.multanen, pekka.jaaskelainen}@tuni.fi

Abstract. Exploiting *instruction level parallelism* (ILP) is a widely used method for increasing performance of processors. While traditional *very long instruction word* (VLIW) processors can exploit ILP energy-efficiently thanks to static instruction scheduling, they suffer from bad code density with serial parts that cannot utilize the multi-issue capabilities. *Transport triggered architecture* (TTA) is a variation of the VLIW paradigm with an *exposed datapath* that improves scaling of VLIW processors with the cost of even wider instructions by exposing the datapath interconnection network to the programmer.
To this end, we propose Dual-IS, an architecture that implements a TTA multi-issue and RISC-V compatible single-issue instruction-set modes by means of a microcoded control path. By utilizing the instruction set modality, the TTA mode can be used in codes that benefit from ILP, while a single-issue RISC-V ISA mode reduces instruction stream energy and code size for sequential programs. Thanks to the TTA programming model, the static multi-issue mode can be implemented without additional register file ports.
The processor was synthesized on a 28 nm ASIC technology. For this design point, when instruction-set mode was selected based on energy-delay product at the program granularity, with CHStone benchmarks, Dual-IS had on average 14% lower energy-delay product compared to a single-mode TTA processor with a similar datapath, while only adding a 3% overhead in the core area. Dual-IS achieved on average 15% and in the best case 33% smaller run times than a single-issue RISC-V implementation by running programs in the TTA mode only when it was beneficial in terms of performance.

Keywords: instruction level parallelism · microcode · multiple instruction set architecture · transport triggered architecture

1 Introduction

In order to achieve higher performance, processors exploit *instruction level parallelism* (ILP) either statically in *very long instruction word* (VLIW) style or dynamically in *superscalar* processors. The benefit of the VLIW style is simplified hardware implementation acquired by static scheduling which yields better results in terms of energy efficiency and area in digital signal processing applications compared to dynamically scheduled processors. However, VLIWs suffer

from high instruction stream energy and code size in programs that cannot utilize ILP. This is caused by the use of *no-operation* instructions (NOPs) when the VLIW instruction packet cannot be completely filled with useful operations.

To make designs more flexible, designers have incorporated multiple heterogeneous cores that can be used depending on the desired use case into their *system-on-chips* (SoCs). While this results in increased flexibility, it is not an optimal solution from an area utilization point of view as it effectively adds "dark silicon" [18] to the design.

Exposed datapath processors [12] provide an instruction-set that exposes low-level aspects of the datapath to the programmer. *Transport triggered architectures* (TTAs) are highly modular architectures that follow the exposed datapath paradigm, which makes them an interesting base for VLIW processor designs. In TTAs, the programming interface is based on *moves* that transport operands and execute operations as a side-effect. Their key drawback is that like traditional VLIW processors, TTAs suffer from high code size and instruction stream energy in sequential programs. In addition, the low-level programming interface exposes more information in the instruction format which correspondingly requires more bits in the instruction word. However, the programming interface of TTAs allows efficient exploitation of ILP thanks to optimizations enabled by exposing the interconnection network to the programmer.

With the aim to provide the best of two worlds, we introduce Dual-IS, a processor that implements both a RISC-V single-issue and an exposed datapath VLIW instruction set. Dual-IS enables optimizing execution of programs in terms of performance, code size and energy efficiency by supporting two different instruction sets. We maximize the reuse of the core's datapath resources by implementing the RISC-V mode with the use of microcode that maps and sequences transport triggered move instructions for the exposed datapath. Thanks to the exposed datapath architecture, we can support a multi-issue instruction set with a *register file* (RF) complexity of a single-issue implementation.

This paper makes the following contributions:

- A novel dual-mode (RISC-V single-issue and an exposed datapath VLIW) instruction set for exploiting instruction level parallelism statically when available in the program, but not suffering from the VLIW's poor code density when there's a lack of it.
- Datapath resource sharing between the RISC-V and exposed datapath ISAs via a low-level microcode-based control implementation.
- A proof-of-concept implementation and ASIC synthesis of the dual-mode instruction set architecture as well as its performance and energy efficiency evaluation.

2 Related Work

There have been various instruction set processor approaches in the past. Lin et al. [17] proposed a unified processor architecture that had both a scalar MIPS-like *reduced instruction set computer* (RISC) instruction set and a 4-way VLIW

mode. The mode was set by a dedicated mode encoding that was specified for each instruction. To minimize code size, the architecture formed 512-bit bundles that could contain both VLIW and RISC instructions. In their approach, the VLIW mode did not implement control instructions and therefore control instructions were always executed in RISC mode.

Hou et al. [11] built on this concept with their FuMicro processor. In their approach, the processor had a superscalar and VLIW ARM mode. The two modes had a different issue width: The superscalar mode had an issue width of two to minimize dependency checking while the VLIW mode was a 7-issue architecture. The instructions were fetched in 256-bit packets that could contain 16- or 32-bit instructions.

Contrary to the previous two processors, Karaki et al. [13] proposed a processor that could run both ARM and x86 instructions on shared hardware. The processor utilized a hardware interpreter that translated and sequenced ARM micro-operations from x86 instructions which enabled sharing the processor datapath between instruction-set modes. This concept is quite interesting as it combines two completely different ISA design philosophies: reduced and *complex instruction set computer* (CISC).

Crepaldi et al. [4] presented a multi-one instruction set computer that followed the *one instruction set computer* (OISC) principle. The processor had two modes that could be selected during startup. The first mode was a standard OISC mode that was based around the *subleq* instruction. The second mode was called CISC mode and it could change between 14 different run modes depending on a control register value that was explicitly stated in each instruction. In their work, however, the different modes were not used to exploit ILP.

Another interesting architecture utilizing multiple modes is Nvidia's Denver project [2]. Denver implemented a traditional hardware-decoder mode that decoded two ARM instructions per cycle and executed them in parallel if possible. The second mode, microcode mode, was used together with dynamic code optimization that formed optimized packets of micro-operations in software. In this mode, hardware-decoder was bypassed and the optimized micro-operation packets were passed directly to the in-order hardware scheduler to execute up to 7 instructions per cycle. This architectural concept is quite close to the one introduced in this paper. However, we use a an exposed datapath as the statically scheduled multi-issue instruction set together with a microcode unit that implements the RISC-V frontend for the processor core.

VLIWs have also been used in commercial designs together with code translation. For example, the Crusoe processor [14] that implemented x86 compatibility with the use of software translation to run native VLIW code. The concept is quite different from our work, as we add hardware decoding support for both instruction sets instead of using software to translate code during runtime.

Some previous work regarding microcoded control design with the RISC-V ISA has been done. Albartus et al. [1] introduced a RISC-V implementation using microcoded control logic. In their work, the datapath of the processor only allowed one transaction per cycle to occur which meant that execution of RISC-V

instructions required multiple cycles due to operand transportations. Klemmer and Große [15] used an OISC microarchitecture to implement the RISC-V ISA through microcoded control logic. In their work, they implemented the RISC-V instructions as a chain of subleq micro-operations with the use of microcode. Because of this microarchitecture, multiple cycles were needed to implement single RISC-V instructions.

Previous work on microcoded RISC-V design is close to the concept of control logic design used in this work except that the internal microarchitecture is a transport triggered architecture. In our microarchitecture, we directly map the RISC-V instructions to equivalent operations in the function units without implementing them as a chain of operations and use the microcoded control logic for operand transportations on the exposed datapath. In addition, the operand transports are sequenced in parallel to minimize cycle counts when executing instructions. Contrary to previous work, we also make use of the microcoded control logic design by using it to support multiple instruction set modes.

3 Transport Triggered Architectures

Transport triggered architectures are highly modular exposed datapath processors. Like traditional VLIW processors, TTAs are statically scheduled and allow programmers to directly exploit ILP. However, programming TTAs differs vastly from the traditional "operation triggered" architectures [9]. The TTA programming model is based on data moves on the datapath *interconnection* (IC) network which executes operations as a side-effect as demonstrated in Fig. 1. That is, the programmer is exposed to the datapath connections of the processor. The additional degrees of freedom in the programming model, however, leads to wider instruction words [12].

Compared to traditional operation triggered multi-issue machines that require a notorious amount of register file IO to support concurrent operations, transport triggered architectures can exploit ILP with reduced register file connectivity and port amount. This is thanks to the *function unit* (FU) ports being registered, and can therefore be used to store values between clock cycles. Hoogerbrugge and Corporaal [8] showed that transport triggered architecture required an average amount of 0.5 read and 0.35 write ports per operation, compared to two read and one write ports of traditional operation triggered architectures. This difference becomes quite substantial when designing VLIW architectures as increased register file complexity affects the area, timing and energy efficiency of the processor core. Multiple TTA-specific optimizations contribute to the reduced register file traffic:

- **Operand sharing** is used when two sequential operations share the same operand. This potentially saves one register file read as the input operand has already been transported to the function unit input register.
- **Software bypassing** [5] allows programmer to directly move data from function unit output port to function unit input port without routing the data through the register file.

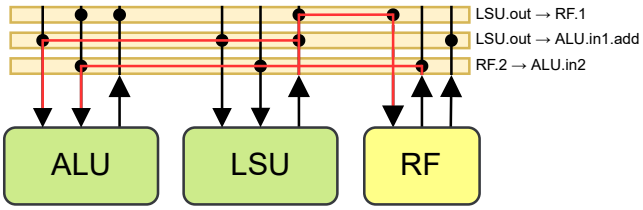

Fig. 1. Example of TTA programming model. Colored lines describe moves on the exposed datapath between function units and a register file which triggers an add operation as a side-effect.

– **Dead result elimination** reduces redundant register writes when an output operand is not needed to be written to the register file. This happens when a software bypass is used to transport the output operand to a function unit and the value is no longer needed.

In addition to reduced register file IO requirements, TTAs can be implemented with a reduced datapath connectivity when compared to operation triggered VLIWs [3]. Contrary to dynamic data hazard detection implemented in operation triggered architectures, TTAs solve data hazards in compile time which reduces hardware complexity together with static software bypasses. In addition, software bypassing eases customization of the interconnect network as the programmer is aware of the connections of the processor datapath.

The low-level programming interface of TTAs allows efficient use of latency hiding with delay slot filling where delay slots of pipelined operations can be used statically by the programmer to execute other operations.

Due to additional state data in function unit ports, external interrupts and exceptions are expensive to implement in TTA processors. However, this is not an issue when a TTA is used as a programmable accelerator as interrupts are rarely justified in this type of use case. The problem of context saving during interrupts can be solved with the concepts introduced in this work where the RISC-V mode can be used for traditional register file base context saving.

4 Dual-IS Processor

Dual-IS processor combines both a RISC-V single-issue mode for sequential code and a TTA-based VLIW mode for exploitation of ILP. An overview of the processor structure is presented in Fig. 2. From the instruction set modality point of view, the most important component is the microcode unit that translates and sequences RISC-V instructions to TTA instructions during run time. In our approach, using microcoded control logic for a RISC architecture enables the processor to share resources with the TTA mode due to the exposed datapath programming model.

The microcode unit is a crucial component that connects the two instruction set interfaces of Dual-IS processor. In addition to instruction translation and

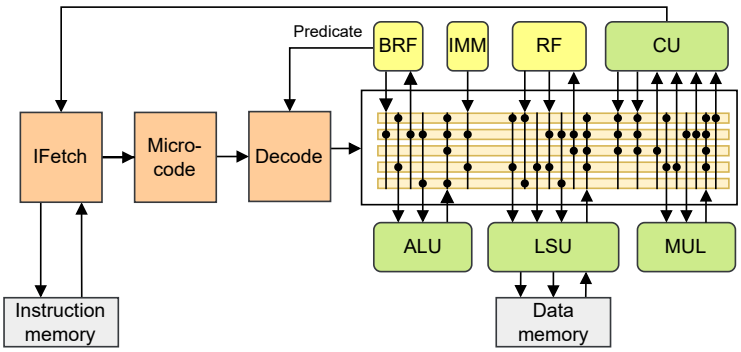

Fig. 2. Overview of the Dual-IS structure with a microcode unit between the instruction fetch and decode components.

micro-operation sequencing, the microcode unit implements all the operation triggered features that are not found in transport triggered architectures, for example, dynamic data hazard detection. In our work, we automatically generate the microcode RTL based on a high-level architecture definition which enables to heavily customize the instruction set of the TTA mode.

The internal structure of the microcode unit is presented in Fig. 3. As seen in the figure, the microcode unit is split into multiple subcomponents that implement parts of the decoding and control logic of the processor. The internal structure of the microcode unit is explored in greater detail in later sections.

By using TTA for the VLIW mode, we could expand the processor datapath without adding register file IO due to the decreased register file usage as discussed in Section 3. This enables to extend a traditional single-issue processor with a VLIW mode with relatively inexpensive hardware changes. A traditional operation triggered VLIW would need the minimum of 2 write and 4 read ports to support the same amount of peak issue width as Dual-IS.

4.1 Instruction Translation

Inside the microcode unit is a translation table that translates RISC-V instructions to TTA instructions. The translation table holds entries for each RISC-V operation and format combination that correspond to parallel moves on the exposed datapath. As the translation table does not need to be reprogrammed, it is constructed purely of combinatorial logic to minimize the hardware overhead.

To reduce the amount of entries, register file indexes and immediate values are not stored in the translation table. Instead, when a RISC-V instruction arrives to the microcode unit, bits representing register file indexes and immediate values are sliced from the instruction which leaves only the function unit and operation codes. This information is sufficient for the translation table to translate the correct function unit and socket encodings that control function unit port multiplexers. After the instruction is read from the translation table, register file indexes are inserted to the translated instruction.

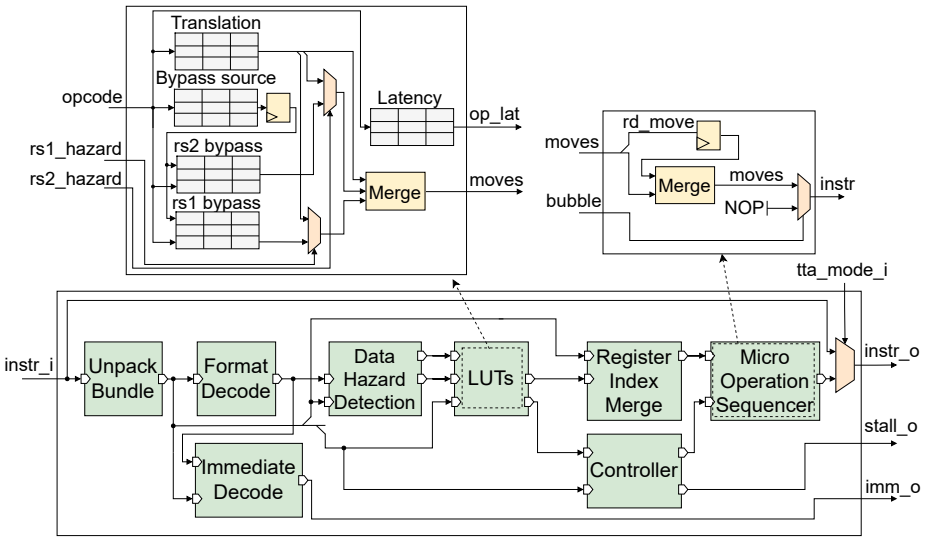

Fig. 3. Dual-IS microcode unit combines instruction translation and dynamic micro-operation sequencing. Data hazards are solved during run time with bypass lookup tables and dynamic data hazard detection. The microcode unit supports instruction set modality with the use of a control signal that is used to bypass the microcode.

Immediate values require more attention, as supporting the full immediate ranges of RISC-V instruction formats takes a considerable amount of encoding space from TTA instructions. To optimize this, the immediate values are sliced from RISC-V instructions and assigned directly to the interconnect immediate signals by bypassing the decoder. This allows having different immediate ranges between different instruction-set modes to preserve encoding space on TTA instruction formats.

4.2 Micro-operation Sequencing

Vast differences between the programming interfaces of operation and transport triggered architectures, as mentioned in Section 3, require the translated instruction to be split into multiple micro-operations. The reason behind this are the explicit moves that transport result operands on the exposed datapath after the operation has been executed. An example of such a phenomenon is a simple RISC register *add* instruction that requires a minimum of two separate instructions on a TTA:

```
Cycle 0: RF.1 → ALU.in2, RF.2 → ALU.in1.add
Cycle 1: ALU.out → RF.3
```

On the first cycle, the input operands are transported to the ALU and the add operation is triggered. On the next cycle, the result operand is transported to the register file. To implement this feature dynamically on the Dual-IS RISC-V

mode, the microcode unit splits the translated TTA instruction to two micro-operations where the first micro-operation moves the input operands and the second the output operands after the operation has been executed.

In order to support multi-cycle operations, the microcode unit is embedded with an additional *lookup table* (LUT) that stores entries for operation latencies. Fig. 3 describes the internal logic of the micro-operation sequencing. By default, the result move is delayed by one clock cycle by registering it in the micro-operation sequencer. For each instruction, operation latency is read from a lookup table. If the operation is a multi-cycle operation, the moves transporting input operands are sequenced after which the pipeline is bubbled until the operation has been executed and the second micro-operation sequenced.

4.3 Control and Data Hazards

Control instructions are more complex to sequence compared to other instruction types. The complexity stems from architectural differences: RISC-V ISA does not use programmer visible delay slots to hide latencies of control instructions. In order to preserve delay slots, and to follow the TTA programming model for the TTA mode, the microcode unit is embedded with a controller that solves control hazards during the execution of control instructions by stalling the processor pipeline. The execution of control instructions could be improved with the use of a branch predictor but we chose not to include one to minimize the amount of RISC-V specific hardware.

The programming interface of RISC-V enables manipulation of the program counter register directly without routing the data through the later stages of the processor pipeline. This is especially beneficial when the control instruction does not have a dependency on the register file. In Dual-IS, direct RISC-V jump instructions are implemented by bypassing the interconnect which reduces operation latency by one clock cycle when executing them in RISC-V mode.

In addition to control hazards, solving data hazards requires additional hardware in the microcode unit due to the static software bypasses used in TTAs. To implement dynamic bypasses in the Dual-IS processor, the microcode unit is embedded with additional lookup tables. A LUT is required for each of the register operands. These LUTs contain the bypass move from the source function unit output port to the target function unit input port for the hazard operand.

As there can be multiple function units, a third LUT that maps operations to function units is required. To support bypass moves between multiple different function units, the two LUTs containing bypass moves must be two-dimensional where the operation encoding and the source function unit form a set of keys.

Fig. 3 presents the dynamic bypass feature in more detail. The LUT component contains the translation, bypass source and operand bypass LUTs that are constructed of purely combinatorial logic. When a data hazard is detected, the register file move on the data hazard bus is discarded by multiplexing it with the move read from the bypass lookup table. This way, the correct operand value is moved directly to the function unit input port by bypassing the register file.

4.4 Mode Switching

Supporting multiple instruction sets induces problems for the *instruction fetch* (IF) design. The main challenge is how to support the different instruction word sizes of different instruction-set with minimal complexity. In our approach, we chose to bundle the RISC-V instructions to a wider instruction packet that is fetched at a time. The width of the packet corresponds to one TTA instruction width which allows simple instruction fetch design. Effectively this restricts the TTA instruction word to be a multiple of the single-issue instruction word size while minimizing the hardware overhead.

This design choice, however, does not come without issues because branches between non-bundle-aligned addresses must be supported. This also has an overhead in terms of fetched instruction bits because when jumping between non-bundle-aligned addresses, redundant instructions are fetched. This problem could be minimized by aligning all jump targets with the cost of larger code size.

In Dual-IS, the instruction-set mode is controlled by a signal that is visible in the hardware interface of the processor. The mode switching itself requires very little additional logic because the RISC-V mode shares the same decoder with the TTA mode. In the microcode unit, the translated and sequenced micro-operations are multiplexed with the fetched instruction which enables to bypass the microcoding and execute TTA instructions as demonstrated in Fig. 3. Because we chose to decode RISC-V immediate values directly in the microcode unit to save encoding space in TTA instruction formats, the decoder requires an additional multiplexer that discards the immediate value output by the microcode unit and assigns the one produced by the decoder instead. Overall, the additional hardware changes required by mode switching are relatively small.

5 Evaluation

For evaluation, we generated multiple designs that are discussed in more depth in Section 5.1. The generated designs were synthesized with Synopsys Design Compiler and a 28 nm ASIC technology to acquire post-synthesis properties of the cores. In addition, the cores were benchmarked with the widely used CHStone benchmark suite [6] in ModelSim RTL simulation to acquire cycle counts and switching activity information for power estimation.

5.1 Evaluated Designs

We used the open-source OpenASIP toolset [7] to generate and design our evaluated cores. OpenASIP supports customization, generation, simulation and programming of TTA-based *application-specific instruction-set processors* (ASIPs). It ships with an automatically retargeting LLVM-based compiler that supports heavy customization of the processor architecture.

To evaluate the overhead of multiple instruction-set modes, an in-order 4-stage single-issue RISC-V core with hardware multiplier support was designed

with the toolset by using a TTA with a reduced interconnect network connectivity and hardware resources as the internal microarchitecture together with a generated microcode unit. Similarly, a TTA core was generated without the RISC-V mode and the RISC-V specific branch, call and auipc instructions that cannot be utilized by the OpenASIP's compiler. The architectures were targeted for small embedded applications without floating point computation.

Both Dual-IS and single-mode TTA processors have five interconnect buses and concurrently accessible function units which enables the execution of a memory, an arithmetic and a control flow operation in parallel. The TTA instruction set has an instruction width of 64 bits that is double the width of RISC-V instruction word which enables to bundle instructions without using padding bits. For the single-mode TTA and Dual-IS, instruction memories consisted of 64-bit banks to match the instruction fetch width. The instruction memory of the single-issue processor consisted of 32-bit elements.

To optimize the critical path, load and multiply operations were implemented as 2-cycle operations. For the single-issue instruction sets, execution of these operations caused one stall cycle. The TTA instruction sets, thanks to their low-level programming model, were able to utilize static latency hiding to minimize the overhead of multi-cycle operations. In addition to multi-cycle load and multiply operations, control operations had programmer-visible delay slots that could be exploited by the TTA instruction set.

5.2 Synthesis Results

Out of the three designs, the single-issue core achieved the highest clock frequency of 2.08 GHz. The maximum clock frequency for the single-mode TTA design was 2.04 GHz and for Dual-IS 2.00 GHz. The negligible overhead of Dual-IS comes from the more complex control unit that adds additional multiplexers to the interconnect which has a small effect on timing. The additional hardware of Dual-IS, however, was not directly on the critical path of the design.

Table 1 lists the area utilization and breakdown of the designs. Dual-IS utilized 3% more area than the single-mode TTA core and compared to the RISC-V core, the area overhead was 17%. In Dual-IS, the microcode unit itself utilized 3% of the area and in the single-mode RISC-V design only 2%. The larger area overhead of the microcode unit in Dual-IS was due to the additional multiplexers that are required to switch instruction-set modes. Overall, the area overhead of the microcode unit was negligible.

5.3 Performance

In this work, we utilized instruction set modality only at program scope which allowed to use the same program binaries for Dual-IS and single-mode architectures. While in-program mode switching would offer more flexibility and gain from the Dual-IS architecture, choosing the instruction set mode at program scope works well for applications that are clearly either control oriented or parallel. Thereby, Dual-IS instruction set runs have the same clock cycle counts as

Table 1. Area breakdown of the cores.

	Dual-IS	**TTA**	**RISC-V**
Area (μm^2)	12900	12500	11000
RFs	37%	40%	42%
FUs	35%	40%	38%
IF	8%	3%	8%
Decoder	5%	5%	3%
IC	10%	12%	8%
Microcode	3%	-	2%

the matching single-mode processors as the instruction set modality does not have an overhead in terms of cycle counts. The clock cycles of TTA and RISC-V instruction sets are presented in Table 2.

Running programs with the TTA instruction set reduced clock cycle counts by 17% on average and 21% if mips was excluded. The most notable improvement was seen in the blowfish benchmark which had a 35% reduction in clock cycles when run with the TTA instruction set. The control-oriented mips benchmark did not benefit from ILP, which is why cycle counts were 15% higher than in RISC-V mode. The use of branches and lower jump latency contributed to the lower clock cycle count as the TTA instruction set used a slower predicate + direct jump to implement branches. Run times followed the same trend as the clock cycles. However, as the single-mode RISC-V and TTA implementations are able to achieve higher clock frequencies, Dual-IS runs suffer a run time overhead of 4 and 2% when compared to their matching single-mode TTA and RISC-V implementations. With Dual-IS, when running mips in RISC-V mode and other benchmarks in TTA mode, the run time was only 0.4% higher on average compared to the single-mode TTA architecture. Compared to the single-mode RISC-V architecture, the run time was on average 15% lower.

5.4 Energy Efficiency

Energy consumption of the processors was evaluated by generating *switching activity information files* (SAIFs) in RTL simulation and using them to generate power reports for the synthesized designs. As a 128 kB instruction memory was

Table 2. Cycle counts and instruction code sizes.

Benchmark	**TTA cycles**	**RISC-V cycles**	**TTA code size**	**RISC-V code size**
adpcm	138480	168448	13 kB	8 kB
aes	23753	25894	12 kB	5 kB
blowfish	542882	837995	7 kB	6 kB
gsm	13058	17518	11 kB	6 kB
mips	37670	32839	5 kB	2 kB
motion	2140	2641	6 kB	3 kB
sha	428790	602243	12 kB	7 kB
jpeg	2262176	2617326	85 kB	44 kB

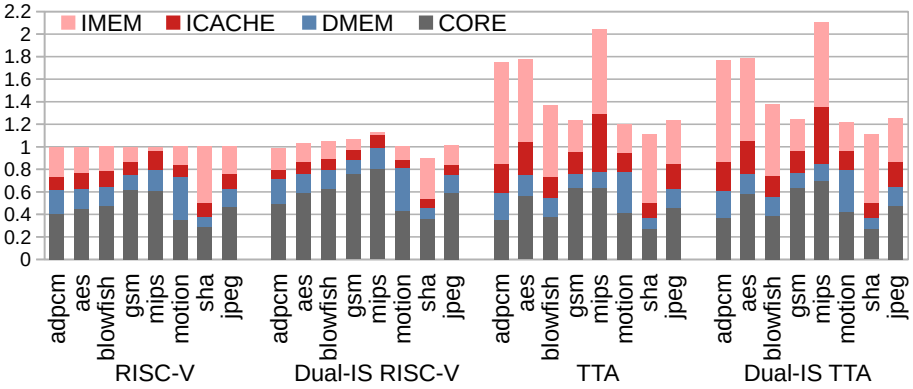

Fig. 4. Energy consumption relative to single-mode RISC-V baseline.

required to successfully compile all the benchmarks, we evaluated the system with that instruction memory size. For the instruction cache, we chose a direct-mapped 2 kB cache with a line size of 16 bytes. The data memory size was 64 kB because that was sufficient for compiling all the benchmarks.

We evaluated the instruction and data stream energy by generating memory access traces in RTL simulation and used Cacti [16] for generating estimates of the access energies. The memory access traces generated in RTL simulation were processed with a cache simulator to find the amount of accesses to the instruction cache and memory. The evaluated energies at maximum clock frequencies are described in Fig. 4 where energy foot print of the core, data memory, instruction cache and instruction memory are separated.

From the instruction stream energy point of view, bundling the RISC-V instructions to packets of two was beneficial as wider banks have lower access energy per bit which lowered the instruction stream energy of the Dual-IS RISC-V runs by 29% on average. Dual-IS had the same amount of cache misses as the single-mode RISC-V processor because the bundles never exceed cache lines. Furthermore, even though Dual-IS fetched more bytes from the cache due to excess instructions that are fetched in the bundle during jumps, it had the same amount of fetched bytes from the instruction memory because during a cache miss the whole line is replaced in the cache.

The reduced energy achieved by bundling instructions compensates for the energy overhead of the more complex datapath and control logic of Dual-IS. In Dual-IS RISC-V runs, the core itself, however, consumed 27% more energy on average compared to the single-mode RISC-V core. The energy overhead of the core is expected due to the underlying multi-issue microarchitecture that executes the single-issue RISC-V instructions. In addition, Dual-IS function units are more complex compared to the single-mode RISC-V implementation because extra registers are required to store values in the function unit input ports between cycles to implement the transport triggering behaviour for the TTA mode.

The TTA instruction set suffered a significant overhead in terms of energy consumption in benchmark applications that expose little ILP. This is expected

due to the wider instruction word that contained NOPs when move slots could not be utilized for operand transports. Furthermore, the programmer-visible delay slots increased the number of redundant instructions when they could not be utilized for operations in sequential code. On average, compared to the single-mode RISC-V processor, the single-mode TTA processor consumed 48% more energy in total. Dual-IS TTA mode consumed 1% more energy in total compared to the single-mode TTA processor due to the extra hardware that is required for the RISC-V instruction set and switching of instruction sets but the overhead was also compensated by the slightly lower clock frequency of Dual-IS.

With Dual-IS, it is beneficial to run benchmarks in RISC-V mode when frequent instruction cache misses and lack of ILP cause the instruction stream energy to be significant. *Energy-delay product* (EDP) [10] is a widely used metric that combines energy efficiency and execution speed of circuits. When instruction set was chosen based on the lowest EDP value, Dual-IS had 14% lower EDP compared to the single-mode TTA design. This way, the run time was 6% higher than with the TTA processor but energy consumption was lowered by 18%.

Power breakdown of different components is listed in Table 3. In the single-mode RISC-V core, the microcode unit consumed on average 6% power. In Dual-IS, the equivalent power consumption was 9%. The difference is due to the additional multiplexers that are needed to switch between RISC-V and TTA modes. When running Dual-IS on TTA mode, the microcode unit consumed 4% of the total power due to multiplexers responsible for mode switching.

Table 3. Power breakdown of the designs.

Component	RISC-V	Dual-IS RISC-V	TTA	Dual-IS TTA
Register Files	15%	16%	13%	13%
Function Units	31%	34%	47%	42%
Instruction Fetch	7%	7%	5%	7%
Decoder	11%	14%	14%	15%
Interconnect	16%	15%	18%	16%
Microcode	6%	9%	-	4%

5.5 Discussion

With the evaluated benchmark set we were able to reduce cycle counts on average by 17% with the TTA instruction set. Higher average performance improvement could be acquired by concentrating on benchmarks with more exploitable ILP. However, it should be noted that this benchmark set was chosen to evaluate the benefit of flexible architectures and their efficient use in the minimization of the instruction stream energy in statically scheduled multi-issue machines when ILP is scarce. Additional improvements to minimize the TTA instruction stream energy could be done by using a loop oriented L0 cache such as a loop buffer that would fit the program hot spots. That would steer the design towards using the TTA mode mostly for loop kernels and the RISC-V mode for control oriented parts and to retain binary compatibility with unmodified RISC-V binaries.

6 Conclusions

In this paper, we introduced Dual-IS, a processor that provides both a single-issue RISC-V and exposed datapath VLIW instruction-set modes with shared resources. To support the two instruction sets in the same microarchitecture, we implemented the RISC-V mode with the use of microcode-based control that extends the exposed datapath core with decoding and control logic. By using an exposed datapath instruction set for the TTA mode, we could implement the core with the same level of register file complexity as a single-issue implementation. Instruction set modality of Dual-IS enables to select instruction-set modes to optimize performance, energy efficiency or code size which adds flexibility.

Dual-IS achieved on average 15% and in the best case 33% smaller run times than a single-issue RISC-V processor when instruction-set modes were selected at the program scope to optimize performance. Compared to a single-mode TTA processor, when selecting the instruction set modes of Dual-IS based on energy-delay product, 14% lower EDP was achieved. This way benchmarks not benefiting from instruction level parallelism were run in RISC-V mode which lowered the energy consumption by 18% and increased run time only by 6% on average compared to a single-mode TTA processor.

The hardware overhead from instruction set modality and RISC-V ISA was negligible (3%), as was the impact to the clock frequency (2%) when compared to a single-mode TTA processor with similar datapath resources. This demonstrates that multi-issue TTA processors can be extended with RISC-V binary support at a low cost while providing the ability to minimize VLIW overhead in sequential programs with the serial execution mode.

In this paper, we used different instruction-set modes at a program granularity to optimize either performance or energy efficiency. In the future, we plan to investigate compiler support for switching of instruction-set modes and study at which granularity it is most beneficial to toggle modes. In addition to decreased instruction stream energy for the TTA, in-program mode switching would allow to share the local data in the caches and register files, which would be beneficial in terms of performance and latency compared to offloading tasks to a separate co-processor. In-program mode switching would be especially well suited for programs that have both control-oriented and parallel code regions.

Acknowledgments

The work for this publication was funded by European Union's Horizon 2020 research and innovation programme under Grant Agreement No 871738 (CPSoSaware) and Academy of Finland (decision #331344).

References

1. Albartus, N., Nasenberg, C., Stolz, F., Fyrbiak, M., Paar, C., Tessier, R.: On the design and misuse of microcoded (embedded) processors—a cautionary note. In: 30th USENIX Security Symposium (USENIX Security 21) (2021)

2. Boggs, D., Brown, G., Tuck, N., Venkatraman, K.: Denver: Nvidia's first 64-bit arm processor. IEEE Micro (2015)
3. Corporaal, H.: TTAs: Missing the ILP complexity wall. Journal of Systems Architecture (1999)
4. Crepaldi, M., Merello, A., Di Salvo, M.: A multi-one instruction set computer for microcontroller applications. IEEE Access (2021)
5. Guzma, V., Jääskeläinen, P., Kellomäki, P., Takala, J.: Impact of software bypassing on instruction level parallelism and register file traffic. In: Embedded Computer Systems: Architectures, Modeling, and Simulation. Springer Berlin Heidelberg (2008)
6. Hara, Y., Tomiyama, H., Honda, S., Takada, H., Ishii, K.: CHStone: A benchmark program suite for practical C-based high-level synthesis. In: 2008 IEEE International Symposium on Circuits and Systems (ISCAS) (2008)
7. Hepola, K., Multanen, J., Jääskeläinen, P.: OpenASIP 2.0: Co-Design Toolset for RISC-V Application-Specific Instruction-Set Processors. In: IEEE 33rd International Conference on Application-specific Systems, Architectures and Processors (ASAP) (2022)
8. Hoogerbrugge, J., Corporaal, H.: Register file port requirements of transport triggered architectures. In: Proceedings of the 27th Annual International Symposium on Microarchitecture. MICRO 27, Association for Computing Machinery (1994)
9. Hoogerbrugge, J., Corporaal, H.: Transport-triggering versus operation-triggering. In: Proceedings of the 5th International Conference on Compiler Construction. CC '94, Springer-Verlag (1994)
10. Horowitz, M., Indermaur, T., Gonzalez, R.: Low-power digital design. In: Proceedings of 1994 IEEE Symposium on Low Power Electronics (1994)
11. Hou, Y., Hu, H., Xu, Y., Guo, D., Wang, X., Fu, J., Qiu, K.: FuMicro: A fused microarchitecture design integrating in-order superscalar and VLIW. VLSI Design (2016)
12. Jääskeläinen, P., Kultala, H., Viitanen, T., Takala, J.: Code density and energy efficiency of exposed datapath architectures. Journal of Signal Processing Systems (2015)
13. Karaki, H., Akkary, H., Shahidzadeh, S.: X86-ARM binary hardware interpreter. In: 2011 18th IEEE International Conference on Electronics, Circuits, and Systems (2011)
14. Klaiber, A.: The technology behind Crusoe processors. Transmeta Technical Brief (2000)
15. Klemmer, L., Große, D.: An exploration platform for microcoded RISC-V cores leveraging the one instruction set computer principle. In: 2022 IEEE Computer Society Annual Symposium on VLSI (ISVLSI) (2022)
16. Li, S., Chen, K., Ahn, J.H., Brockman, J.B., Jouppi, N.P.: CACTI-P: Architecture-level modeling for SRAM-based structures with advanced leakage reduction techniques. In: 2011 IEEE/ACM International Conference on Computer-Aided Design (ICCAD) (2011)
17. Lin, T.J., Chao, C.M., Liu, C.H., Hsiao, P.C., Chen, S.K., Lin, L.C., Liu, C.W., Jen, C.W.: A unified processor architecture for RISC & VLIW DSP. In: Proceedings of the 15th ACM Great Lakes Symposium on VLSI. Association for Computing Machinery (2005)
18. Taylor, M.B.: Is dark silicon useful? harnessing the four horsemen of the coming dark silicon apocalypse. In: DAC Design Automation Conference 2012 (2012)