Transport-Triggered Soft Cores

Pekka Jääskeläinen*, Aleksi Tervo*, Guillermo Payá Vayá†, Timo Viitanen*,
Nicolai Behmann†, Jarmo Takala* and Holger Blume†
*Laboratory of Pervasive Computing
Tampere University of Technology, Finland
Contact email: pekka.jaaskelainen@tut.fi
†Institute of Microelectronic Systems
Leibniz Universität Hannover, Germany

***Abstract*—Soft cores are used as flexible software programmable components in FPGA designs. Transport-Triggered Architecture (TTA) is interesting for this use due to its scalability, modularity, simplified register files (RF) and fine-grained compiler control, but has the drawback of wider instructions and additional multiplexing due to extensive RF port sharing. In this paper we evaluate the trade-offs of TTA in soft core use in comparison to its closest multi-issue relative, the traditional "operation triggered" VLIW architecture, as well as the Xilinx MicroBlaze, a popular single-issue soft core.**

For the compared alternatives running CHStone benchmarks, the dual-issue TTA with a monolithic RF provides the best performance/area trade-off. Its program size increase varies from 21% to 49% in comparison to the VLIW programming model. However, synthesis results on a Xilinx Zynq Z7020 device show that the dual-issue TTA requires 67% of the resources while providing up to 88% improvement in execution time compared to VLIW. Partitioning the RF is found beneficial for both VLIW and TTA programming models, resulting in a very similar FPGA resource usage, but with TTA model improving execution time up to 77%. As a single-issue soft core, we measured execution time improvements up to 173% when comparing with the TTA approach against the performance-optimized MicroBlaze with similar datapath resources.

I. Introduction

Soft cores in FPGA-based system designs are typically used to run serial control oriented parts of the application, often on top of full blown general purpose operating systems. This trend can be seen in the FPGA-optimized soft cores offered by the major FPGA vendors: The Nios II Gen 2 soft cores from Altera (Intel) [1] and the MicroBlaze from Xilinx [2] are general purpose scalar RISC cores with capabilities tuned to running complex operating systems such as Linux.

With the introduction of FPGA *system-on-a-chip (SoC)* devices that include hard processor cores that are better-equipped to run general purpose operating systems, the number of use scenarios for the general purpose soft cores from the FPGA vendors is reduced. However, the overall benefits of soft core based designs are not limited to general purpose or tightly coupled accelerator control use cases. If the soft core template is configurable enough and provides a different degree of fine grained parallelism to improve performance, the designs can be useful also for performance critical "number crunching" parts of implementations. In that case, the benefits of a software based control over fixed function, state machine based designs are seen in easier resource sharing across various algorithms and algorithm phases in the implemented application. Ideally, designers can reap the efficiency benefits of custom logic while retaining the flexibility and speed of development of a software system. Up to 70% gate count reductions have been reached with *application-specific instruction-set processor (ASIP)* based soft cores in large FPGA-based designs [3].

Exploiting parallelism is crucial in performance critical soft core designs where the achievable per-operation delays are typically much higher than with ASIC based implementations. A form of fine-grained parallelism, *instruction-level parallelism (ILP)* is a means to improve performance of the application within a single thread by issuing multiple program operations concurrently. Because dynamic multi-issue out-of-order instruction scheduler hardware incurs rather high resource costs [4], it is popular to choose a variation of the *Very Long Instruction Word (VLIW)* paradigm [5] for multi-issue soft core use.

Instead of extracting parallel instructions during execution, VLIW relies on the compiler to issue multiple instructions concurrently. However, VLIW suffers from the *register file (RF)* complexity bottleneck; for each parallel *function unit (FU)*, an RF in the datapath must typically support two parallel reads and a parallel write. VLIW also requires hardware logic to control value forwarding to avoid the RF latency in dependent operation chains. Complex RFs and the forwarding logic consume precious resources and can reduce the achievable clock frequency.

Transport-Triggered Architecture (TTA) is an "exposed datapath" processor design style which has been proposed as a scalability optimization for VLIW processors. TTA exposes additional control by letting the programmer define the data movements in the interconnection network. This helps to reduce RF complexity and avoid forward resolution hardware [6], [7], but leads to less dense instructions, which has been identified as the main drawback of TTA. In addition, the extensive port sharing that is common in TTA designs adds complexity to the IC in form of additional multiplexers. Port sharing can also add penalty to the software side due to the need to serialize simultaneous accesses to the same RF when not enough parallel ports are not available, which is a problem also with partitioned RF VLIW designs.

Previously, TTAs have been studied in the context of ASIC

implementation with soft core use receiving little attention. FPGA presents different challenges and tradeoffs than ASIC technologies due to its predefined set of coarser grain resources and routing paths. This paper presents an evaluation of the benefits and drawbacks of the TTA programming model when used for both multi-issue and single-issue soft core designs. We report our findings in terms of FPGA resource usage, instruction memory impact, and the overall execution performance.

The rest of this paper is organized as follows. Section II gives an overview of the related work. Section III discusses the TTA approach with focus on details relevant to FPGA implementation. Section IV describes our experimental setup, and Section V presents the results. Finally, Section VI concludes the paper and outlines our plans for future work.

II. Related Work

Many VLIW-based soft cores have been proposed in the past [8]–[10]. In [11], the authors acknowledge the RF to be a major bottleneck for VLIWs in FPGAs and propose improvements to FPGA fabrics to alleviate it. Various other approaches attempt to tackle the multiported RF bottleneck at the implementation or architecture level. These techniques rely on replication of data to multiple block RAMs and can require more than ten times the RAM bits in the FPGA implementation per each useful storage bit [12], [13]. Other FPGA-specific hardware techniques to optimize different aspects of soft-core VLIW processors, such as using SIMD FUs, are also presented in [14].

In this paper, we approach the FPGA soft core optimization challenge from the programming model aspect. Using a soft core template with a fine grained programming model, some instruction memory density is traded for simplifications in the general purpose RFs and additional software optimizations. VLIW-SCORE [15] resembles the idea to some extent. It is a VLIW variant with separate operand RFs attached to each input of each of the FUs. The organization is similar to the clustered RF approach which can be supported with both VLIW and TTA designs, but leads to additional code partitioning complexity in the compiler. The paper proposes the architecture for a specific programming language to accelerate SPICE simulations and no intentions for using it as a target for general purpose programming languages such as C were discussed.

The TTA processor paradigm was introduced in the 1970s with a control processor use case [16], [17]. The foundational ideas underlying the VLIW improvement were introduced later by *Corporaal et al.* [18] who conducted extensive research on the benefits of data transport programming in scalable statically scheduled processor architecture design. However, TTA has not been systematically optimized and evaluated for soft core use before. The closest related work to this paper is our previous publication [19] where we describe the open source TCE toolset [20] for customized TTA-based soft core design. However, the publication did not evaluate the important aspects for the soft core use case extensively, but merely demonstrated the validity of the design flow in rapid custom soft core generation accompanied with initial results.

Since GPGPU programming became more mainstream, there has also been interest to study "GPU-like" soft cores to support programming models targeting GPGPUs on FPGAs [21]. We consider the scope of these works at a higher design level than the one of this paper; the TTA soft cores under investigation can be used to design components that can act either as *processing elements (PE)* or *compute units* in OpenCL terminology [22]. The PEs can be arranged to a "GPU-like" *single-instruction multiple threads* data parallel execution organization. This is something we plan to look into in the near future.

III. Transport-Triggered Architectures

TTA exposes the datapath *interconnection* network (IC) to the programmer. In this so called *data transport programming* model, programs are defined as sequences of instructions each declaring a list of parallel *moves*, which cause the processor to perform data transfers between ports of datapath components such as FUs and RFs. Actual program functionality such as arithmetic or memory operations are *triggered* as a side effect when transporting operand data to designated trigger ports of the FUs. Fig. 1 presents the programmer's view of an example TTA processor with five transport buses which allow a maximum of five simultaneous data transports per instruction.

Because the TTA programming model is very fine grained, its instruction format requires only a little hardware logic to decode. The format resembles the ones in horizontal microcode programmed architectures [23]. The cost of encoding the instruction at a low abstraction level is reduced code density, which can be improved with instruction compression [24] and sensible instruction memory hierarchy design.

A. Interconnection Network

In the TTA variant we consider in this paper, the IC consists of *buses* and *sockets*. Sockets are used to connect ports of FUs and RFs to the move-controlled buses. The socket concept and a lower level implementation view is illustrated in Fig. 2.

In TTAs, buses with multiple writers and readers are common as they can sometimes reduce wiring. However, the programming model does not *require* bus sharing in any way. When shared buses are not optimal for the underlying implementation technology or use case, less connected and even point-to-point connections can be used similarly to other architectures. Therefore, when judging the complexity of TTA IC, a key point to realize is that one can design a VLIW or a coarse-grain reconfigurable array with their usual connectivity and resources, and then expose the connections to the programmer to convert it to a TTA. In fact, an interesting optimization starting point for a TTA design is a standard VLIW with an IC that has a point-to-point bus per parallel FU to RF port connection and optional RF forwarding paths between FU ports to avoid the RF write back latency. The length and complexity of the wiring required by the IC then

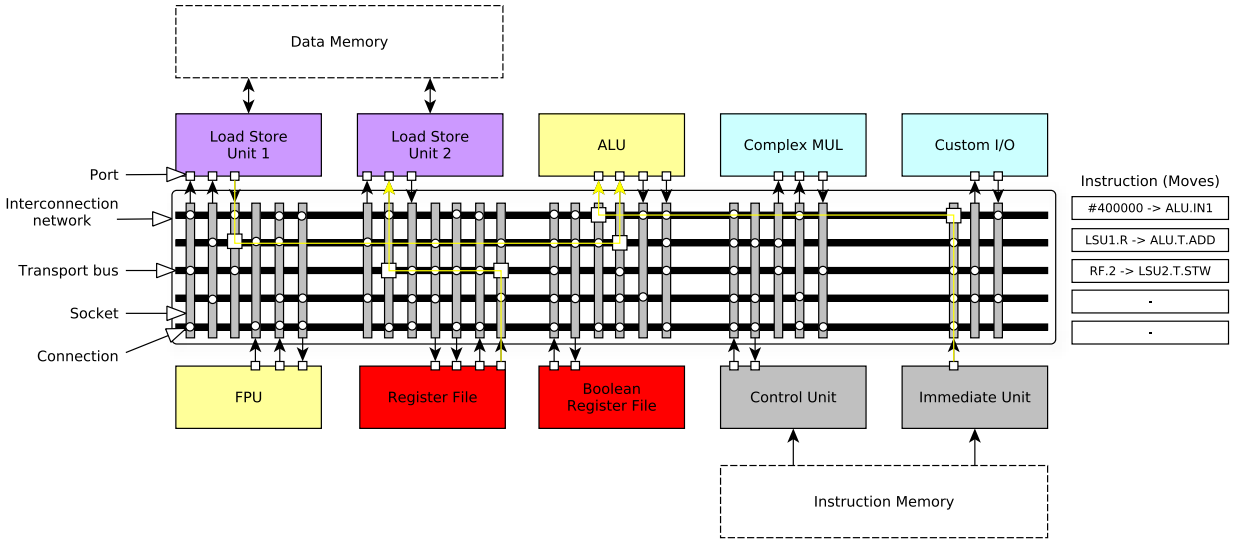


Fig. 1: Example of a TTA processor. In TTA, data transports between components are explicitly programmed. The example instruction defines an instruction with three moves controlling data transports in three buses out of the five. The instruction performs an integer summation of a value loaded from a data memory with a constant while simultaneously storing a previously computed value to memory.

depends mostly on the size of the processor design; how many FUs need to be connected to other RFs and how far apart they end up being laid out on the implementation medium.

B. Function Units and Forwarding

TTA FUs can contain a mix of operations with complexity varying from simple scalar arithmetics to vector processing on customized data types. The FUs can be pipelined, multi-cycle, single-cycle and even implement complex resource sharing between multiple operations. For FU pipelining, the evaluated variant of TTA uses *semi-virtual time latching* [18] where the pipeline is controlled with valid bits as depicted in Fig. 3b. The pipeline starts an operation whenever there is a move to the trigger port, i.e., when the *o_load* signal is activated. The result can be read from the result register after the internal latency of the FU passes, but the read can be also postponed up until the time the old result is replaced by a result from a new operation.

TTAs replace hardware based forwarding logic with *software bypassing*, which allows the program to move data between FUs without accessing or even referencing a general purpose register file [6], [7]. It is also possible to utilize the (optional) storage in FU input ports by omitting an operand move in case the same data was already moved to the port. In addition to these TTA-specific optimizations, the general ability to control the timing of the operand and result data transports alleviates the parallel register file access pressure.

C. TTA as an Optimized VLIW

The use of the TTA paradigm to provide a more scalable alternative for traditional VLIW architectures has been studied extensively [18]. A key result is that TTAs can support more instruction-level parallelism with simpler register files than "traditional" VLIWs thanks to the additional programming freedoms. In VLIWs, the total number of RF ports must be

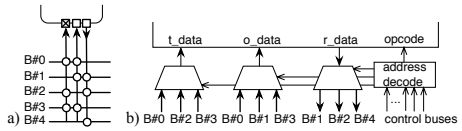


Fig. 2: Socket interface for function units: a) symbolical and b) structural presentation.

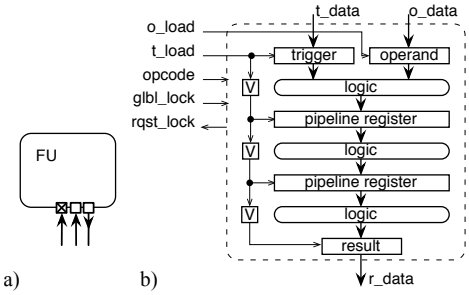
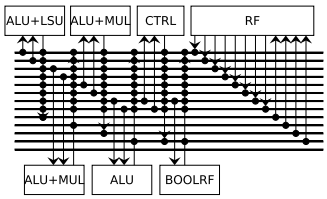


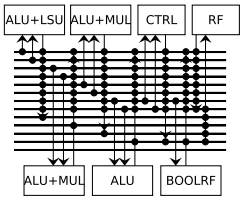
Fig. 3: Function units in the considered TTA: a) symbolical representation and b) implementation illustration diagram.

scaled according to the best-case issue rate of the processor: Each parallel FU must be able to read their operands and write their results to an RF at the same cycle. For example, in [6] it was found that by utilizing the TTA programming model, two parallel operations can be sustained by using only a two-ported RF, and an average of 3.6 parallel operations supported by a six-ported RF, whereas a traditional VLIW requires at least four read/two write and eight read/four write ports, respectively.

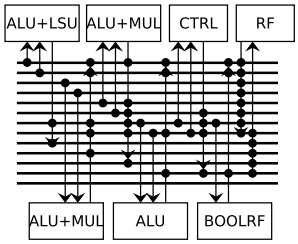
In order to illustrate how the TTA approach can simplify VLIW designs, the series of pictures in Fig. 4 demonstrate an example of a 4-issue VLIW that is optimized step-by-step to an application (domain) customized TTA datapath. In Fig. 4a the internals of an example VLIW datapath are presented as a TTA. The interconnection network contains the



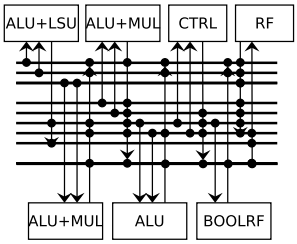
(a) Original 4-issue monolithic-RF VLIW datapath.



(b) Register file port count reduced.



(c) Bypass network connectivity pruned.



(d) Some of the buses merged.

Fig. 4: From VLIW to an optimized TTA. ALU = Arithmetic-Logic Unit, CTRL = Control Unit, BOOLRF = BOOLean Register File (predicates), LSU = Load-Store Unit, MUL = MULtiplication unit, and RF = Register File.

same connections and the RF has the same number of ports as in an "operation triggered" VLIW with a full RF bypass (forwarding) network.

Fig. 4b shows a TTA with the general purpose RF simplified without sacrificing issue rates. By utilizing the additional programming freedoms and the storage in the FU input or output, there is less need to perform concurrent accesses to the RF.

D. Register File Partitioning

Partitioning a single, many-ported RF to multiple smaller RFs with fewer ports helps to alleviate the RF complexity problem for both VLIW and TTA datapaths. The drawback of splitting RFs is that when too many operations require operands from the same register file, it forces serialization of operations that could be otherwise parallelized. This puts more pressure to the compiler code generation to assign variables efficiently to the RFs. However, it should be noted that while VLIW also supports splitting the RFs to reduce port counts per RF, port counts in the split RFs can be made even smaller with the TTA approach [7]. Simplifying and splitting RFs to use fewer ports greatly simplifies their implementation, but requires extra multiplexing in the IC. It is important to consider this tradeoff when implementing a VLIW or TTA in FPGA.

In the TTA case, as the IC is explicitly visible to the programmer, it is typical that the connectivity is pruned according to a single application or an application domain. Fig. 4c presents the design after underutilized bypass connections are pruned, and Fig. 4d after merging buses with connections that are rarely utilized concurrently. IC optimization also heavily affects the TTA instruction word width as the size is proportional to the number of buses (move slots in the instruction) and the binary logarithm of connections in each bus (source and destination fields in move slots). Naturally any application-specific tailoring always means reduced flexibility, which can result in performance reduction for other workloads. TTAs can reduce this negative impact by the additional degrees of programming freedom as can be seen later in the presented results.

IV. Experimental Setup

The goal for our evaluation was to quantify the benefits and drawbacks of TTAs in comparison to traditional "operation triggered" in-order architectures starting from the smallest useful design up to multi-issue machines. For the single-issue comparison, we designed a small 3-bus TTA, *m-tta-1*, comparable to a 32b scalar RISC processor. The processor includes an integer datapath with a *load-store unit (LSU)*, an *arithmetic-logic unit (ALU)* and 32 general purpose registers. This configuration was compared against two MicroBlaze cores with 3- and 5-stage pipelines, later referred to as *mblaze-3* and *mblaze-5*. In order to keep their capabilities comparable to the single-issue TTA, the MicroBlaze configurations were set to their minimum, with the exception of adding a 32-bit multiplier that is optional. For compilation of the benchmark applications, the Xilinx IDE with GCC 6.2.0 was used with the optimization flag -O3.

For the multi-issue comparison, we started with a two-issue machine with the same function units as the *m-tta-1*, but with resources to issue both a memory operation and arithmetic operation in parallel. This base datapath was extended to a three-issue machine by adding a second ALU. In order to focus on the effects of the programming model instead of the choice of operations, we included only the minimal set of FU operations required by the C compiler used in the experiments and an integer multiplication operation to avoid

TABLE I: Integer operations and their latencies (in parenthesis) included in the FUs of the evaluated base datapath. The memory operations are all to absolute addresses.

ALU	
add (1)	+
and (1)	&
eq (1)	=
gt (1)	signed >
gtu (1)	unsigned >
ior (1)	|
mul (3)	*
shl (2)	<<
shr (2)	arithmetic >>
shru (2)	logical >>
sub (1)	−
sxhw (1)	sign extend 16b
sxqw (1)	sign extend 8b
xor (1)	∧

LSU	
ldw (3)	load 32b
ldh (3)	load 16b and sign extend
ldq (3)	load 8b and sign extend
ldqu (3)	load 8b and zero extend
ldhu (3)	load 16b and zero extend
stw (0)	store 32b
sth (0)	store 16b
stq (0)	store 8b

excessive software emulation code for integer multiplications confusing the results. The 32b operations and their latencies included in the two fully pipelined FUs are listed in Table I. There is also a control unit with absolute *jump* and return address saving *call* operations.

For the multi-issue measurements, we produced various design points of interest with the same FUs. The general purpose register counts for the architectures were chosen to avoid underutilization of the distributed RAM blocks available in the target platform which had a minimum size of 32 registers.

Monolithic VLIW (m-vliw-2/3) represents monolithic RF VLIW processors. It has all the 32b registers in a single register file. The dual-issue machine (m-vliw-2) has a 64x32b RF with two write ports and four read ports, and the three-issue one (m-vliw-3) has a 96x32b RF with three write ports and six read ports. Each FU output and input has a dedicated connection to a port in the RF.

Monolithic TTA (m-tta-2/3) are TTA alternatives of *m-vliw-2/3*. The difference is that the port complexity of the monolithic RF is reduced by relying on TTA specific optimizations. *m-tta-2* has a single RF with one read and one write port and *m-tta-3* a single RF with two read and one write port, but requires more multiplexing in the IC due to the shared ports.

Partitioned VLIW (p-vliw-2/3) are versions of *m-vliw* with the monolithic RF split to two RFs for *p-vliw-2* and to three RFs for *p-vliw-3*, each with half or a third of the ports. Each individual register file has 32 registers, so the total register count and the FU connectivity of these RFs was identical.

Partitioned TTA (p-tta-2/3) are TTA versions of *p-vliw-2/3* with similarly partitioned RFs, but with only one read and one write port in each RF.

Bus merged TTA (bm-tta-2/3) is like *p-tta-2/3*, but it has its buses merged similarly to what was shown in Fig. 4d. These variations showcase the instruction width and performance impact from bus merging [25]. The IC pruning was not done per benchmark application, but aiming for the flexibility to execute all the benchmark programs efficiently.

TTA-Based Co-design Environment (TCE [19]) was used to design the architecture variations. TCE is a design toolset for customizing TTA processors. It can also generate RTL for FPGA or ASIC synthesis, and has both a C compiler [26] and an instruction-cycle accurate architecture simulator. In addition to supporting TTA code generation (which can be considered a superset of VLIW compilation), the compiler also supports turning off the additional TTA programming freedoms, which allowed us to produce cycle counts for the VLIW cases with minimal differences in the compilation chain. The CHStone [27] benchmark set was used as the test workload, The SoftFloat cases were not included due to lack of double precision float support in TCE.

TCE produces an instruction encoding automatically for the TTA alternatives. However, as TCE does not currently support generating a VLIW encoding and a supporting decoder hardware, we designed the instruction encoding manually for the VLIW machines with the following rationale: Each VLIW issue slot takes the form of an opcode followed by two source fields and a destination field. Both issue opcodes require 4 bits. A destination field is simply a register address, but a source field requires an extra bit to select an immediate value. The 2-issue architectures require 6 bits per register address, while the 3-issue architectures require 7 bits. The resulting instruction format is 44 bits wide for 2-issue VLIW machines, and 73 bits for 3-issue ones.

For FPGA synthesis we utilized a Xilinx Zynq Z7020 (speed grade -1) FPGA device and Vivado 2017.2 tools. The RTL for the architectures generated by TCE was synthesized first with hierarchical synthesis at 100 MHz to get resource utilization breakdowns and separately with a flattened hierarchy to find the maximum clock frequencies. No area constraints were used for the synthesis or place-and-route in this experiment.

Register files were implemented using distributed RAM utilizing a design proposed in [28]. The RFs with fewer ports could use the memory primitives directly, but in the case of *m-vliw* with its multiple write ports, additional bookkeeping logic and multiplexers were needed.

V. Results

A. Program Size

Table II summarizes the instruction width and program size statistics for the tested alternatives. The wider instructions of TTA do not directly translate to program image size because often the TTA schedules are shorter due to the additional compiler optimizations. For example, when comparing the monolithic RF cases, the instruction width of TTA is almost double of the VLIW's, but in these benchmarks the program image size increase is only from 21% to 49%. The relative size of MicroBlaze and TTA program image sizes vary more extremely depending on the benchmark, with TTA image sizes between 32% and 248% of the corresponding MicroBlaze program. In *blowfish*, for example, the considerably smaller

TABLE II: Instruction widths and total program image sizes, reported relative to *MicroBlaze* (the 3 and 5 stage versions are binary compatible) in the 1-issue case, and to *m-vliw-2/3* in the multiple-issue cases.

	instr. width	adpcm	aes	bfish	gsm	jpeg	mips	motion	sha
1-issue									
mblaze	32b	122kb	202kb	177kb	99kb	299kb	55kb	97kb	190kb
m-tta-1	43b (1.34x)	1.32x	1.10x	0.54x	1.42x	2.48x	0.89x	0.83x	0.32x
2-issue									
m-vliw-2	48b	170kb	262kb	114kb	159kb	876kb	55kb	94kb	69kb
p-vliw-2	48b (1.00x)	0.98x	1.01x	0.99x	1.01x	1.00x	1.01x	1.10x	1.03x
m-tta-2	81b (1.69x)	1.47x	1.29x	1.23x	1.49x	1.31x	1.43x	1.28x	1.21x
p-tta-2	83b (1.73x)	1.44x	1.37x	1.38x	1.48x	1.38x	1.52x	1.34x	1.28x
bm-tta-2	66b (1.38x)	1.14x	1.05x	1.10x	1.24x	1.11x	1.23x	1.04x	1.03x
3-issue									
m-vliw-3	72b	228kb	373kb	160kb	227kb	1233kb	78kb	139kb	101kb
p-vliw-3	72b (1.00x)	1.03x	1.03x	1.05x	1.03x	1.04x	1.04x	1.05x	1.01x
m-tta-3	145b (2.01x)	1.63x	1.39x	1.32x	1.58x	1.45x	1.67x	1.21x	1.08x
p-tta-3	134b (1.86x)	1.50x	1.29x	1.22x	1.48x	1.36x	1.54x	1.10x	1.01x
bm-tta-3	99b (1.38x)	1.01x	0.86x	0.85x	1.09x	0.97x	1.17x	0.76x	0.74x

TTA program size is due to using LLVM in the TCE compiler, which performs more aggressive whole program optimizations than the GCC-based compiler of MicroBlaze.

An important phenomenon to observe with TTAs is that the instructions get smaller when the IC is properly optimized [25]. The *bm-tta* machines highlight this effect. The benefit of IC optimization for program size is clearly visible in the shown results: In all of the cases *bm-tta* produces more compact programs than *p-tta* due to the reduced instruction width, while reaching similar cycle counts thanks to the TTA programming freedoms. In the 2-issue comparison, *bm-tta-2* still has instruction memory overhead compared to the VLIW cases, but it is at least halved in all cases compared to *p-tta-2*, with a maximum increase of 24% in comparison to *m-vliw-2*. In the 3-issue case, *bm-tta-3* program image sizes are in some cases smaller than with VLIW (due to the shorter schedules), with image size varying from 74% to 117% of *m-vliw-3's*.

B. Synthesis Results

The maximum clock frequencies and FPGA resource usage are shown in Table III. The interesting subcomponents in the VLIW comparison are RFs and IC. The other parts of the designs are similar with the same, or only marginally differing resource utilization. All of the alternatives used 3 DSP blocks for the multiplier.

As was expected, the monolithic VLIW register file reduces the clock frequency for the multi-issue cases. This is emphasized in *m-vliw-3* where the maximum frequency drops to 146 MHz. When using the partitioned RF, the differences between VLIW and TTA architectures are not major, but the TTA variants still reach somewhat higher clock frequency. This is despite the fact that the comparison is not completely fair to the TTA as the VLIW would include the additional forward resolution logic and write back stages in the instruction pipeline, which might end up increasing the critical path length.

We consider the achieved clock frequencies rather good given the relatively low operation pipelining and the FPGA device speed grade. It might be possible to increase the clock frequencies by adding more pipelining, but this would move the latencies to the software side where additional cycles are often hard to hide with the limited ILP available in typical programs.

The differences in clock frequencies are clearer in the 1-issue comparison against the MicroBlaze cores, with the TTA reaching up to 28% higher clock frequency. However, in comparison to MicroBlaze, the TTA pays for its higher clock frequency with 15% higher logic utilization than the performace-focused *mblaze-5*.

The 2-issue utilization figures are very similar outside the RFs. The more complex RF of *m-vliw-2* needs significantly more resources, requiring 6 to 14 times more logic than the other machines, but the difference falls to 25-50% when the whole core is taken into account. The 3-issue numbers show a similar pattern, with *m-vliw-3* requiring 9 to 27 times more resources for the RF, and 41% to 65% for the core.

C. Execution Performance

For soft cores aimed for acceleration, the achieved execution performance per consumed resources is eventually the most interesting metric. To this end, Table IV shows the instruction cycle counts, and Fig. 5 shows the expected runtime with the achieved maximum clock frequencies.

In terms of execution performance, the TTA alternatives clearly outrun their VLIW counterparts as well as both MicroBlaze configurations. In comparison to the MicroBlaze cores, *m-tta-1* shows speedups between 15% and 173% relative to the speed optimized *mblaze-5*.

In the monolithic RF case, *m-tta* sees speedups between 24% and 88% compared to *m-vliw* in the 2-issue case and between 20% and 202% in the 3-issue case. While the split RF cases reach similar clock frequencies for both issue widths, the TTA specific compiler optimizations improve performance up to 77% for the 2-issue comparison and up to 174% for the 3-issue comparison.

D. Discussion

The relevance of the instruction memory size increase in comparison to the achieved resource savings and the performance improvements naturally depends on the case at

TABLE III: FPGA resource usage and maximum clock frequency, reported relative to *mblaze-3*, *m-vliw-2* and *m-vliw-3* for the respective issue widths. RF port counts are listed for TTA and VLIW architectures to highlight the relationship between RF complexity and logic utilization. LUT = look-up table, RF = register files, LUT as RAM = LUT used by distributed RAM primitives, IC = interconnection network (numbers not available for MicroBlaze), and FF = flip-flop.

	RF read ports	RF write ports	fmax (MHz)	LUT utilization core	RF / LUT as RAM	IC	FF
1-issue							
mblaze-3	-	-	169	715	128 / 128	-	303
mblaze-5	-	-	174 (1.03x)	829 (1.16x)	64 (0.50x) / 64	-	582 (1.92x)
m-tta-1	1	1	216 (1.28x)	956 (1.34x)	24 (0.19x) / 24	265	507 (1.67x)
2-issue							
m-vliw-2	4	2	176	1806	638 / 352	439	694
p-vliw-2	2	1	203 (1.15x)	1441 (0.80x)	96 (0.15x) / 96	587 (1.34x)	632 (0.91x)
m-tta-2	1	1	212 (1.20x)	1208 (0.67x)	44 (0.07x) / 44	437 (1.00x)	599 (0.86x)
p-tta-2	1	1	213 (1.21x)	1342 (0.74x)	48 (0.08x) / 48	542 (1.23x)	619 (0.89x)
bm-tta-2	1	1	212 (1.20x)	1212 (0.67x)	48 (0.08x) / 48	438 (1.00x)	590 (0.85x)
3-issue							
m-vliw-3	6	3	146	3825	1970 / 1056	680	977
p-vliw-3	2	1	194 (1.33x)	2710 (0.71x)	144 (0.07x) / 144	1290 (1.90x)	923 (0.94x)
m-tta-3	2	1	167 (1.14x)	2399 (0.63x)	210 (0.11x) / 176	932 (1.37x)	895 (0.92x)
p-tta-3	1	1	197 (1.35x)	2651 (0.69x)	72 (0.04x) / 72	1290 (1.90x)	908 (0.93x)
bm-tta-3	1	1	189 (1.29x)	2320 (0.61x)	72 (0.04x) / 72	1023 (1.50x)	850 (0.87x)

TABLE IV: Cycle counts. Numbers for other architectures given in relation to *mblaze-3* or *m-vliw* for easier comparison.

	mblaze-3	mblaze-5	m-tta-1
adpcm	283954	0.90x	**0.53x**
aes	84892	0.92x	**0.42x**
blowfish	2081752	0.89x	**0.66x**
gsm	33731	0.87x	**0.66x**
jpeg	4483651	**0.91x**	0.98x
mips	72650	0.97x	**0.73x**
motion	12670	**0.97x**	1.05x
sha	1843148	0.87x	**0.56x**

	m-vliw-2	p-vliw-2	m-tta-2	p-tta-2	bm-tta-2
adpcm	142402	1.01x	0.84x	**0.81x**	0.82x
aes	39491	0.99x	0.77x	**0.68x**	0.87x
blowfish	1594847	0.95x	**0.73x**	0.77x	0.84x
gsm	27279	1.00x	0.74x	**0.69x**	0.78x
jpeg	4731551	1.01x	0.88x	**0.86x**	0.93x
mips	53612	1.00x	**0.97x**	1.00x	1.02x
motion	17362	1.05x	0.64x	**0.62x**	0.65x
sha	1172304	1.01x	0.71x	**0.67x**	0.77x

	m-vliw-3	p-vliw-3	m-tta-3	p-tta-3	bm-tta-3
adpcm	133718	1.03x	0.76x	0.75x	**0.67x**
aes	37899	1.01x	0.59x	**0.57x**	0.65x
blowfish	1552318	1.01x	**0.53x**	0.53x	0.59x
gsm	26760	1.01x	0.57x	**0.56x**	0.62x
jpeg	4638550	1.03x	**0.77x**	0.77x	0.80x
mips	51661	1.02x	0.96x	**0.95x**	0.98x
motion	17154	1.00x	0.38x	**0.37x**	0.41x
sha	1121799	1.00x	**0.45x**	0.45x	0.50x

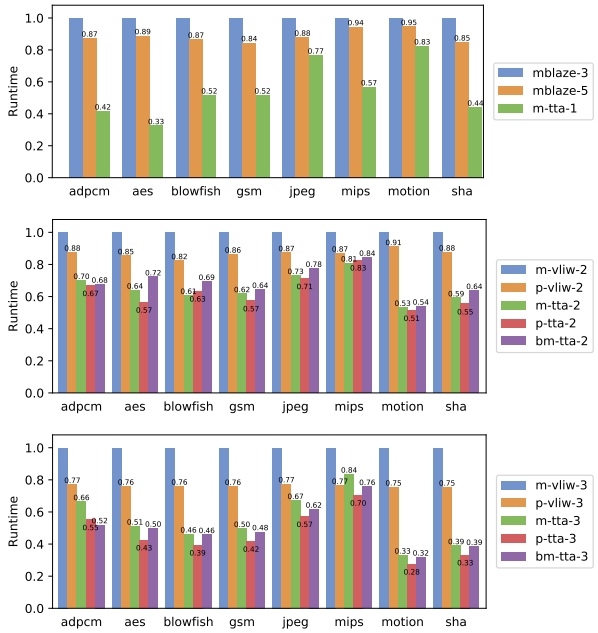


Fig. 5: Execution times according to the achieved max clock frequencies, normalized to *mblaze-3* for the 1-issue comparison, and *m-vliw-2/3* for the multi-issue comparisons.

hand. Instruction memory hierarchy can be implemented with multiple levels with the lowest level in an external memory to store large programs. The storage in lower levels of the memory hierarchy can be shared between several cores in case of multicore designs. It is thus not typical to include a large dedicated on-chip program memory per core, but to either include a small on-chip program memory to store only the core loops, or use a multilevel hierarchy with a small instruction cache and a large external memory. The resource consumption impact of a larger RF, on the other hand, is paid for each core also in case of multicore designs.

Figure 6 shows performance and logic utilization of the tested architectures to visualize overall logic efficiency. The design points closest to origin provide the best resources/performance tradeoff. The 1-issue and 2-issue TTAs have the best efficiency for the used benchmark suite.

VI. Conclusions and Future Work

This paper presented the first pragmatic study of the TTA approach in comparison to the VLIW programming model and a vendor specific scalar architecture in FPGA soft core use. We measured the effects of the TTA programming model to the program size, FPGA resource usage, and execution performance. We confirmed that the main drawback of TTA

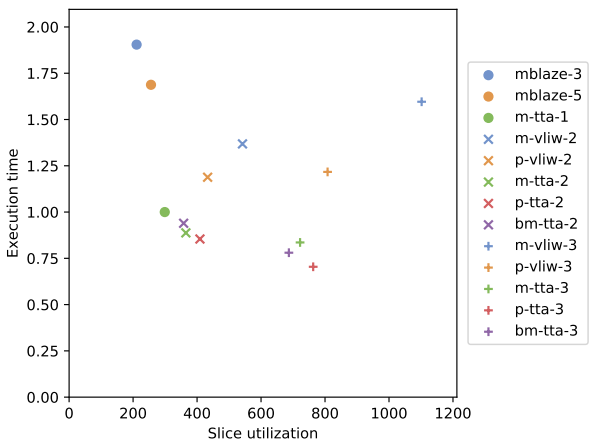


Fig. 6: Slice utilization and performance of the tested architectures. Overall execution time is calculated as the geometric mean of all benchmarks, normalized to *m-tta-1*.

is the larger program size with up to 67% increase with a monolithic RF multi-issue design. However, the program size can be reduced with a careful TTA IC design, and it can be made less significant with a multi-level instruction memory hierarchy. The 2-issue comparison demonstrated achievable FPGA resource savings with TTA up to 33%, and up to 39% for the 3-issue case, in comparison to datapaths with VLIW RFs. This saving does not even include the additional hardware required by the more complex VLIW control unit. The speedups received from the TTA programming freedoms were considerable, up to 88% faster than the VLIW.

In the future, we will investigate the limits of TTA in implementing more powerful soft cores. We will study the effects from adding more powerful FUs, more and larger RFs and using SIMD FUs. We are also interested in various multi-core/manycore configurations and FPGA-optimized instruction compression methods.

Acknowledgment

This work was funded by Academy of Finland (decision 297548), Business Finland (FiDiPro project StreamPro, 1846/31/2014) and the HSA Foundation. We also would like to thank Mr. Ryan Hinton for his valuable feedback from the point of view of an FPGA engineering expert.

References

[1] Altera Corporation (Now Part of Intel), "Nios II gen 2 processor reference guide," available in the web, October 2016.

[2] Xilinx, Inc., "Using the MicroBlaze processor to accelerate cost-sensitive embedded system development," available in the web, June 2016.

[3] Synopsys, "Synopsys and Fuji Xerox (a success story)," available in https://synopsys.com, 2015.

[4] H. Wong, V. Betz, and J. Rose, "High-performance instruction scheduling circuits for superscalar out-of-order soft processors," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 11, no. 1, pp. 1:1–1:22, Jan. 2018.

[5] J. A. Fisher, "Very long instruction word architectures and the ELI-512," in *Int. Symp. on Computer architecture (ISCA)*, 1983.

[6] J. Hoogerbrugge and H. Corporaal, "Register file port requirements of Transport Triggered Architectures," in *Int. Symp. on Microarchitecture*, Nov.-Dec. 1994.

[7] J. Janssen and H. Corporaal, "Partitioned register file for TTAs," in *Workshop on Microprogramming (MICRO-28)*, 1996.

[8] I. Tili, K. Ovtcharov, and J. G. Steffan, "Reducing the performance gap between soft scalar CPUs and custom hardware with TILT," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 10, no. 3, Jun. 2017.

[9] F. Anjam, M. Nadeem, and S. Wong, "A VLIW softcore processor with dynamically adjustable issue-slots," in *Int. Conf. on Field-Programmable Technology*, Dec 2010.

[10] Y. Lei, Y. Dou, J. Zhou, and S. Wang, "VPFPAP: A special-purpose VLIW processor for variable-precision floating-point arithmetic," in *Int. Conf. on Field Programmable Logic and Applications*, Sept 2011.

[11] M. Purnaprajna and P. Ienne, "Making wide-issue VLIW processors viable on FPGAs," *ACM Transactions on Architecture and Code Optimization*, vol. 8, no. 4, 2012.

[12] F. Anjam, S. Wong, and F. Nadeem, "A multiported register file with register renaming for configurable softcore VLIW processors," in *Int. Conf. on Field-Programmable Technology*, Dec 2010.

[13] B. C. C. Lai and J. L. Lin, "Efficient designs of multiported memory on FPGA," *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 1, Jan 2017.

[14] S. Nolting, G. Paya-Vaya, and H. Blume, "Optimizing VLIW-SIMD processor architectures for FPGA implementation," in *ICT.OPEN*, 2011.

[15] N. Kapre and A. DeHon, "VLIW-SCORE: Beyond C for sequential control of SPICE FPGA acceleration," in *Int. Conf. on Field-Programmable Technology*, Dec 2011.

[16] G. Lipovski, "The architecture of a simple, effective control processor," in *2nd Euromicro Symposium on Microprocessing and Microprogramming*, October 1976.

[17] D. Tabak and G. Lipovski, "MOVE architecture in digital controllers," *IEEE Journal of Solid-State Circuits*, vol. 15, no. 1, February 1980.

[18] H. Corporaal, *Microprocessor Architectures: From VLIW to TTA*. Chichester, UK: John Wiley & Sons, 1997.

[19] O. Esko, P. Jääskeläinen, P. Huerta, C. de La Lama, J.Takala, and J. Martinez, "Customized exposed datapath soft-core design flow with compiler support," *Int. Conf. on Field Programmable Logic and Applications (FPL)*, 2010.

[20] P. Jääskeläinen, V. Guzma, A. Cilio, and J. Takala, "Codesign toolset for application-specific instruction-set processors," in *SPIE Multimedia Mobile Devices*, 2007.

[21] M. A. Kadi, B. Janssen, J. Yudi, and M. Huebner, "General-purpose computing with soft GPUs on FPGAs," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 11, no. 1, pp. 5:1–5:22, Jan. 2018.

[22] *OpenCL Specification v1.0r48*, Khronos Group, Oct. 2009.

[23] P. Jääskeläinen, H. Kultala, T. Viitanen, and J. Takala, "Code density and energy efficiency of exposed datapath architectures," *J. Signal Process. Syst.*, vol. 80, no. 1, pp. 49–64, 2014.

[24] J. Heikkinen, J. Takala, and H. Corporaal, "Dictionary-based program compression on customizable processor architectures," *Microprocessors and Microsystems*, vol. 33, no. 2, 2009.

[25] T. Viitanen, H. Kultala, P. Jääskeläinen, and J. Takala, "Heuristics for greedy transport triggered architecture interconnect exploration," in *Proc. Int. Conf. on Compilers, Architecture and Synthesis for Embedded Systems (CASES)*. ACM, 2014.

[26] H. Kultala, P. Jääskeläinen, J. IJzerman, T. Viitanen, M. Mäkitalo, and J. Takala, "Exposed datapath optimizations for loop scheduling," in *Int. Conf. on Embedded Computer Systems: Architectures, MOdeling, and Simulation (SAMOS)*, July 2017.

[27] Y. Hara, H. Tomiyama, S. Honda, and H. Takada, "Proposal and quantitative analysis of the CHStone benchmark program suite for practical C-based high-level synthesis," *J. Inf. Process.*, vol. 17, 2009.

[28] C. E. LaForest and J. G. Steffan, "Efficient multi-ported memories for FPGAs," in *Proc. Int. Symp. on Field Programmable Gate Arrays (FPGA)*, 2010.