Power Optimizations for Transport Triggered SIMD Processors

Joonas Multanen, Timo Viitanen, Henry Linjamäki, Heikki Kultala, Pekka Jääskeläinen, Jarmo Takala
Tampere University of Technology, Finland
Email: joonas.multanen@tut.fi, timo.2.viitanen@tut.fi, henry.linjamaki@tut.fi, heikki.kultala@tut.fi, pekka.jaaskelainen@tut.fi, jarmo.takala@tut.fi

Lauri Koskinen, Jesse Simonsson
University of Turku, Finland
Email: lauri.koskinen@utu.fi, jesse.simonsson@utu.fi

Heikki Berg, Kalle Raiskila, Tommi Zetterman
Nokia Technologies LTD, Finland
Email: heikki.berg@nokia.com, kalle.raiskila@nokia.com, tommi.zetterman@nokia.com

***Abstract*—Power consumption in modern processor design is a key aspect. Optimizing the processor for power leads to direct savings in battery energy consumption in case of mobile devices. At the same time, many mobile applications demand high computational performance. In case of large scale computing, low power compute devices help in thermal design and in reducing the electricity bill. This paper presents a case study of a customized low power vector processor design that was synthesized on a 28 nm process technology. The processor has a programmer exposed datapath based on the transport triggered architecture programming model. The paper's focus is on the RTL and microarchitecture level power optimizations applied to the design. Using semiautomated interconnection network and register file optimization algorithm, up to 27% of power savings were achieved. Using this as a baseline and applying register file datapath gating, register file banking and enabling clock gating of individual pipeline stages in pipelined function units, up to 26% of power and energy savings could be achieved with only a 3% area overhead. On top of this, for the measured radio applications, the exposed datapath architecture helped to achieve approximately 18% power improvement in comparison to a VLIW-like architecture by utilizing optimizations unique to transport triggered architectures.**

I. Introduction

The choice for the implementation technology to be used for an accelerator has to take in account a number of variables such as the cost of manufacturing per device on a given manufacturing method. General purpose *Central Processing Units (CPUs)* are flexible in terms of programmability and can be bought off the shelf, therefore, having a low non-recurring costs. In this sense, the extreme opposite of generic CPUs are fixed-function *Application-Specific Integrated Circuits (ASICs)*, that execute a specific function and that function only. A fixed-function ASIC can be reconfigurable or programmable to some extent, but not necessarily.

In terms of power consumption, specialized ASICs perform better than their programmable counterparts. They contain no extra logic which would consume power when executing the specific task. For the same reason, their area is usually small compared to CPUs. ASICs also perform better performance-wise since their architecture is tuned to accelerate a specific function.

Especially for portable devices where space and possibilities for cooling are limited, thermal constraints are important. Electrical energy transforms into thermal energy in an integrated circuit and, therefore, ASICs can be the only option when thermal constraints are strict. For large scale computing such as data centers and super computer clusters, energy savings translate to reduced cooling costs, essentially in the form of a smaller energy bill.

Nowadays common other alternatives for CPUs and fixed function ASICs include general-purpose *Graphical Processing Units (GPUs)* and *Field Programmable Gate Arrays (FPGAs)*. GPUs expose high degree of parallel hardware to the programmer, but are not as easy-to-use as CPUs. FPGAs are highly configurable (programmable), but they are not as power-efficient or have as high performance as ASICs. However, non-recurring engineering costs for FPGAs are a fraction compared to that of ASICs and there is no need to manufacture large batches of the targeted product to bring the unit costs to a tolerable level.

In addition, with FPGAs, testing, verification and bug fixing can be done simply by reconfiguring the device (on the *field*), whereas faulty ASICs have to go through a respin, which is both very expensive and lengthy in time.

This paper presents a case study of three power optimizations implemented for a *software defined radio (SDR) Transport Triggered Architecture (TTA)* [1] processor on the *Register Transfer Level (RTL)* and microarchitecture level. There are not many publications regarding power optimizations for TTA processors. The power consumption of the optimized TTA is compared to a VLIW-like architecture with no TTA-unique optimizations to observe the effects of TTA-unique optimizations.

This paper is organized as follows. Section II describes the idea of TTA that sets the structure for the presented processor design. Section III describes power optimizations that were either implemented to the design architecture or

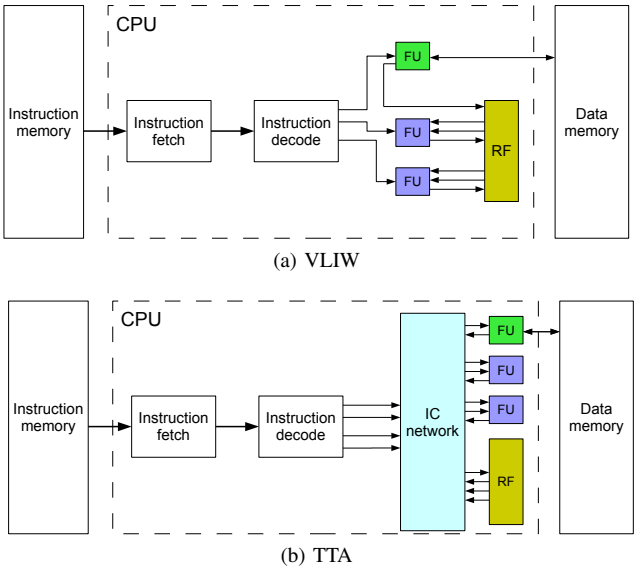

Fig. 1. The structure of VLIW and TTA processor architectures.

otherwise interesting for future research. Section IV describes the optimizations implemented to the design. Post synthesis power and area measurements are presented in Section V, and Section VI concludes the paper.

II. Transport Triggered Architecture

Very Long Instruction Word (VLIW) architecture processors expose *Instruction Level Parallelism (ILP)* to the programmer. This means that independent operations can be executed simultaneously if there are enough resources in the architecture and if the programmer/compiler instructs so.

The basic VLIW architecture is presented in Fig. 1a. A VLIW has a component to fetch instructions from instruction memory. Fetched instructions are passed to an instruction decoder, which converts instructions to signals controlling other parts of the processor. The actual program operations are done in parallel in multiple *Function Units (FUs)*. The instruction word size is proportional to the number of FUs in parallel, hence the name VLIW. VLIWs are *statically scheduled* processors, which means that the program compiler decides the order of instructions during compile-time, instead of at runtime by the processor hardware. This translates to a simpler control logic in the processor, since there is no need for the hardware to detect dependencies in the instructions and reorder them for high performance operation. An opposite approach is *dynamic scheduling* which is a common in contemporary general purpose CPUs.

A shortcoming of VLIW architecture is increased register file complexity, when the architecture is scaled [1] to support more parallelism. This is due to the fact that the complexity of the register files storing the operands and temporary results needs to account for the peak performance of the function units. The worst case scenario in this point of view would be all FUs reading or writing to/from the RF simultaneously and the RFs need to have enough read and write ports (supported parallel accesses) to satisfy this demand. Scaling the architecture also increases the complexity of *bypass* logic in VLIWs. Bypassing means that the result of an operation can be directly transported from an FU output to an FU input, without circulating it through the RF, saving a clock cycle in the instruction latency. In VLIW architectures, adding symmetric bypassing capabilities means connecting all (or a subset of) FU outputs to other FU inputs. The effect of this is that the bypass network grows quadratically when adding FUs to improve the instruction-level parallelism supported by the processor.

In 1976, Lipovski [2] presented a control processor using a so called "MOVE architecture", where operations were triggered by moving operands to inputs of FUs. Thus, the program operations were *transport triggered*, rather than *operation triggered*. Operation triggering is the traditional paradigm in processor design and means that in addition to input operands and the destination (usually in form of general purpose registers), the programmer gives the opcode defining the operation to be executed, and the hardware performs the data transfers required to execute the instruction. In TTAs, the programming model is mirrored by making the data transports the responsibility of the program. The idea of TTAs was later studied extensively by Corporaal et al. [3], who proposed the *MOVE* architecture as an improvement to traditional operation-triggered VLIWs to alleviate their scalability bottlenecks.

Like VLIWs, TTAs use instruction fetch and decode units to control the operation of the CPU and are also statically scheduled. However, unlike in VLIWs, TTAs have an *exposed datapath*, allowing the programmer to control the *InterConnection (IC)* network in addition to scheduling FU operations. The difference of TTA to VLIW is presented in Fig. 1b, where the instruction decode unit controls the IC network, to which FUs and RF(s) are connected. In a sense, the only instructions in TTAs are *Move* and *No Operation (NOP)*. The actual computation is still done in FUs, but in TTAs this happens as a "side effect" of moving data to FU *trigger ports*. FUs still need an opcode port to know which operation to execute.

TTA addresses the scaling bottleneck of the VLIW architecture with its programming model. Unlike in VLIW, the number of RF ports does not directly depend on the number of FUs in TTAs. Also, in TTAs, controlling bypassing can be done at software level.

TTAs allow some unique optimizations due to the exposed datapath and FU structure. An FU can have any number of input and output ports needed. Data to non-triggering ports can be stored into them before an operation is triggered if the FU is not used before that operation, allowing an extra degree of scheduling freedom compared to the VLIW programming model. This allows input and output operands to be moved directly between FUs by the programmer (*software bypassing*) without accessing a general purpose register file for the temporary variable. *Dead result move elimination* can be done in case all uses of a result can be software bypassed; the RF needs not to be accessed at all, thus no register need to be allocated for the value. That is, the general purpose RF pressure is relieved due to the ability to move data directly to FU ports using the program. This allows smaller RF size and port number, since less data needs to be stored in them, resulting in power and area savings.

Operand sharing provides another way to eliminate un-

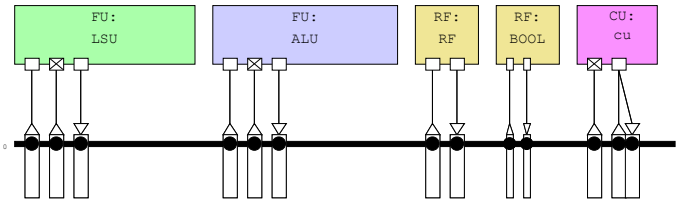

Fig. 2. Example of a TTA processor in TCE view with two FUs: A *Load-Store Unit (LSU)* and an ALU, two RFs and a CU.

necessary move operations. When two consecutive operations in an FU use the same operand value, it can be moved to the FU input register for the first operation and kept there for the second operation, saving a move operation. If the same operand is used repetitively, for example in all iterations of a loop, a move is saved for each iteration.

Due to their modular, flexible and scalable nature and due to their unique optimizations, TTAs are interesting for low-power programmable devices and as a template for rapid programmable accelerator customization. Furthermore, the modularity helps automatic datapath gating and makes automatic RTL generation of TTA processors rather straightforward.

At Tampere University of Technology a design and programming tool suite for TTAs has been developed. *TTA-based Co-design Environment (TCE)* [4] allows the user to generate *Hardware Description Language (HDL)* code for customized parallel processors and to produce parallel program images from high-level programming languages using a retargetable compiler. TCE features a graphical user interface to define data buses, components and the connections between them. TCE was used to design the customized architecture studied in this paper.

Fig. 2 presents an example of a TTA in TCE view. This architecture has one data transport bus, two FUs, two RFs and a *Control Unit (CU)*. The CU contains the *Instruction Fetch* and *Instruction Decoder* components and possibly an *Instruction Decompressor*. The components are connected to the data bus by *sockets*.

TCE presents rather many customization points to the control of the processor designer. Relevant to the optimizations described later in this paper are its FU customization capabilities. For example, in TCE, a TTA FU can contain any number of operations. New operations can be implemented or combined from existing ones to create *custom operations*. This can be useful when a program uses a series of operations repeatedly. Non-unit latency FUs can be fully pipelined or multi-cycled, the difference being that a multi-cycled operation may not accept new input operands, if an operation is being executed while fully pipelined operations can be launched at every cycle. A hybrid of these two is also possible (operations with complex pipeline resource usage patterns). In TCE, FUs can interact with other system components, such as memories or I/O devices. An example of this is the LSU in Fig. 2. It can be used for read and write operations to/from a data memory.

III. Power Optimization Techniques

The studied design is a TTA *Single Instruction, Multiple Data (SIMD)* customized processor with the primary use case in software defined radio. The processor has a 32-bit datapath for scalar operations and a 512-bit datapath for vector operations. It supports execution of 32 parallel 16-bit floating point operations. The design has on purpose a minimal number of custom operations to retain high degree of reuse of the processor for a wider range of future applications. A summary of the design is presented in Table I. Thanks to TTA based design, straightforward generation of RTL code for verification and prototyping was quite effortless. This section first describes existing power optimization techniques and then presents how three of them were implemented in the design architecture.

1) Pipeline Stage Latching: Corporaal [1] distinguishes between *real time latching*, where no control logic is implemented for the FU pipeline stages and *virtual-time latching*, where FU pipeline stages propagate forward every clock cycle, except when a *global lock* signal is active. A global lock is distributed to all FUs and used to stall the processor in case of, e.g., cache misses. Virtual-time latching is further divided into *true virtual-time latching (TVTL)*, where operations in FUs are triggered if data is written to any of the input ports and *semi virtual-time latching (SVTL)* where, only writing data to the trigger port starts an operation.

2) Register File Datapath Gating: In TTAs, register file power, though reduced, is still a major factor in power consumption. Donkoh et al. [5] analyze that a large fraction of RF power is consumed in write data distribution, and reduce power by means of *write data gating*. In *local bitline data gating*, the RF is segmented, and the write data signals to inactive segments are gated to zero. Conversely, in *global bitline data gating*, the entire RF's write data signals are gated. This is beneficial when the signals come from a data bus with multiple destinations. Both approaches are similar to *operand isolation*.

3) Instruction Encoding and Address Bus Optimization: An interesting possibility to reduce TTA power consumption would be to schedule instructions in a way that acknowledges the *hamming distance* of two consecutive instructions on the instruction bus. In a bus, hamming distance is the number of wires changing their state when compared to the previous bus value. Minimizing the number of wires changing has an impact on power and energy consumption. Su et al. used *Gray code addressing* and *Cold scheduling* to reduce switching activity of the instruction and address bus [6]. Benini et al. presented the *Beach solution*, which is a suitable method for reducing address bus switching activity on special-purpose processors rather than general-purpose [7]. Instruction bus encoding could decrease the power consumption of the instruction fetch component, instruction bus and instruction memory.

4) Loop Buffer: Another energy optimization we are investigating for our test architecture is to incorporate a loop buffer. The effects of using a loop buffer for TTA processors was studied in [8], where up to 62% total energy savings were achieved. The effect is application-specific since the power and energy reduction depends on the size, number, and type of loops present in the executed program.

5) Memory Banking: Memory banking is a common technique to reduce memory power consumption and access time. The memory is divided into banks, reducing the capacitive load seen when accessing the memory. Now only the bank being

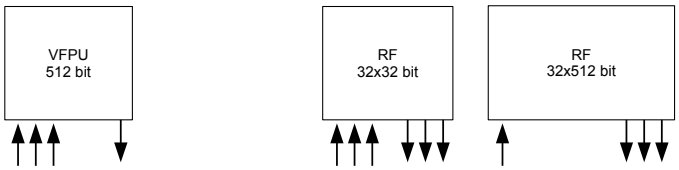

Fig. 3. Datapath components of the design that the power optimizations were targeted to. A three-stage pipelined vector floating point multiply-accumulator and two register files; a scalar and a vector one.

operated on is active for each access. Similar strategy can also be applied to register files, which has been studied previously. Balasubramonian et al. [9] partitioned a monolithic RF into two levels and divided it into banks, where each bank had their own read/write ports. A banking method by Tremblay and Joy has been patented in [10], where a multi-ported register file was divided into smaller register files. Each register file had the same amount of write ports as the original RF and the original read ports were allocated between the smaller register files. In [11], the RF was partitioned to different regions based on activity, and also used drowsyable register cells for further power reduction.

TABLE I. SUMMARY OF THE IMPLEMENTED TTA DESIGN.

target clock frequency	1GHz
instruction width	128b
transport buses	3x32b, 4x512b
registers	2x1b, 32x32b, 32x512b
ASIC technology	28nm FDSOI

IV. IMPLEMENTED OPTIMIZATIONS

Power measurements after synthesis were performed with Synopsys Design Compiler with two SDR benchmark programs optimized for the processor. Here the programs are referred to as *SDR1* and *SDR2*. [1]

The design was first synthesized to find the components consuming the majority of power. In this case, the RF and FU components consumed the majority of the total power. The vector floating point unit consumed more than 29% of total power in both test cases, indicating rather good function unit utilization for an unoptimized programmable design. These components are presented in Fig. 3. The initial power distributions for each test program are presented in Fig. 4. These measurements served as a motivation and a starting point for power optimizations.

Register files and function units combined in the design architecture took more than 65% of the total TTA core power consumption before optimizations. Since RFs and FUs were the most power-hungry components, optimizations were targeted to these two component types. Optimizations were implemented at the microarchitecture and register transfer levels.

The effect of Design Compiler's automatic power optimizations was also observed. Optimizations enabled were leakage power & dynamic power optimization and clock gating. The tool also automatically performed logic and gate level

[1] Benchmark names withheld due to confidentiality reasons.

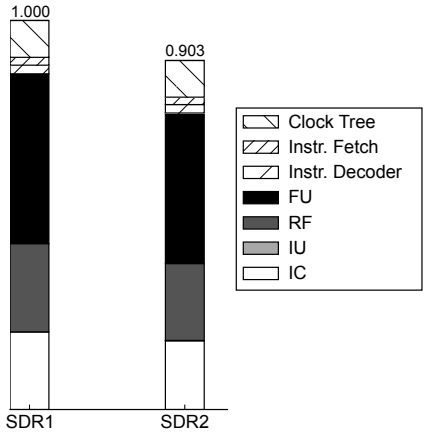

Fig. 4. Power distribution between components with two test programs before the implemented optimizations.

optimizations by logic substitution and low power placement based on switching activity. Optimizations such as ungrouping and register retiming were disabled to better see the effect on individual components rather than the whole core. Ungrouping removes logic block boundaries and can remove redundant logic leading to smaller design area. Register retiming tries to move registers through logic in the design and can improve the timing, if the critical path of the design can be shortened. The architecture was also synthesized with these optimizations, but the additional decrease in power consumption was negligible.

The most effective automated optimization was clock gating the design, which decreased the total power consumption more than 68% in both test cases. Naturally, its benefit depends on the program running on the core as it only reduces the power consumption of underutilized components. Leakage power optimization decreased the power consumption for around 3% in both test cases. Enabling the previous optimizations decreases the overall power consumption more than 71% in both test cases.

A. Interconnect and Register File Exploration

The initial design for the architecture had a quite wide instruction word and high interconnect power. We addressed this by performing automated design space exploration as described in [12]. The FUs of the initial design were first rearranged in a single-RF VLIW-like configuration, where each FU has dedicated read and write ports in a large central register file. The greedy optimization algorithm then merged interconnection buses and RF ports which were seldom simultaneously active. We ran the process until the starting instruction word size (which in the TTA's case heavily depends on the IC network connectivity) of 223 bits was reduced below 128 bits. Finally, we pruned bypass connections with a round-robin interconnect optimizer of which operation idea is described in [13]. The initial TTA, before these semiautomated optimizations, had 6x32 scalar data buses and 8x512 vector data buses. The optimization reduced the number of buses to 3x32 scalar and 4x512 vector. The resulting machine runs the SDR2 and SDR1 benchmarks 20% and 13% slower in terms of cycle counts, respectively, but provides significant area savings and power consumption reduction on instruction memory and interconnection.

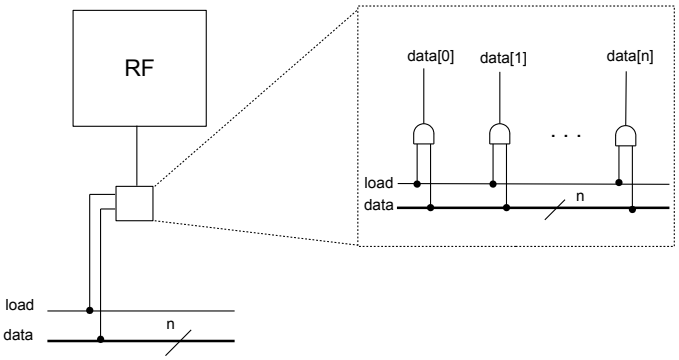

Fig. 5. Register file datapath gating using AND gates.

B. Register File Write Data Gating

Two factors make write data gating particularly interesting in our design. First, the interconnect data buses to which the RFs are connected are often connected also to FUs, and thus used instead to transport FU input data, generating a large amount of spurious RF input switching for data gating to eliminate. Second, we did not have the resources for a custom RF design as in [5] and opted instead for a standard cell based RF. The standard cell flip-flops are likely to have a higher load capacitance than custom RF cells, and due to their larger size and less regular placement, the write signals require more wiring and buffers. Consequently, write data distribution power is larger in a standard cell design, and data gating techniques more effective.

Data transfers between components and data memory in a TTA processor are passed through the interconnection network. The IC consists of data buses and input and output sockets. RFs and FUs only load in new input values if their load signal is active. In [14], RF input data was gated with its enable signal using an AND gate to reduce the load (switching) capacitance of the IC. This prevented the data bus going from the interconnect network to an RF from changing its value, if the value was not to be loaded into the RF. This decreased power consumption by decreasing the capacitive load seen by the interconnection network.

Similar datapath gating of the RFs was implemented in TCE's processor generator and is now automatically included in the generated RTL for the designed processors. The structure is presented in Fig. 5. A gating block is generated for each RF write port.

C. Pipelined Function Unit Clock Gating Enhancement

In TTAs designed with TCE, the function units have well-defined pipeline behavior. Operations are started by writing data to a trigger port, and the results are available n cycles afterwards, where n is the latency of the FU. As discussed in IV, clock gating is an efficient way to reduce power consumption due to eliminating unnecessary switching activity. However, depending on the pipeline stage control implementation, automatic clock gating by synthesis tools may not always be possible.

A straightforward implementation of SVTL used in TCE updates the input registers based on a *load* signal which indicates writes to the trigger port. The data then falls through the subsequent pipeline stages which are only disabled during a global stall. However, this results in inefficient clock gating, as shown in Fig. 6, top. All pipeline registers except for the inputs are constantly clocked even while the FU is idle.

In order to better match the clock gating with TTA operation semantics, RTL implementations of pipelined FUs should be modified to propagate the load signal to further pipeline stages, resulting in the hardware shown in Fig. 6, bottom. This requires some extra hardware: flip-flops to delay the load signal and AND gates with an the global lock signal inverted to evaluate the clock enable condition. Separate pipeline stages require their own clock gating elements.

The presented design utilizes a pipelined three-stage vector floating point unit, which before improvement consumed 29.2% of total core power with SDR1 and 29.1% with SDR2 test programs. Restructuring the FPU RTL for load signal propagation as discussed above decreases its power consumption by almost one-third. As usual with an optimization that concentrates on reducing the idle time power consumption, if the FU is highly utilized, the effect of this modification is smaller compared to when the use is sporadic.

D. Register File Banking

The design in this paper had a 32x512-bit register file for vector operations. Implementing RFs with standard cell flip-flops instead of dedicated register file standard cells can save power in a situation, where the optimal RF size and number of ports are not available as an RF standard cell. In our design the vector RF had one write and three read ports, that were common to all banks. Different bank sizes for the RF were evaluated and the optimal power consumption was reached with 16 banks (2 registers per bank). Banking was implemented by multiplexing input data lines going to registers based on the *most significant bits* of the register write address. The implemented banking reduced the RF power consumption over 35% for both benchmark programs.

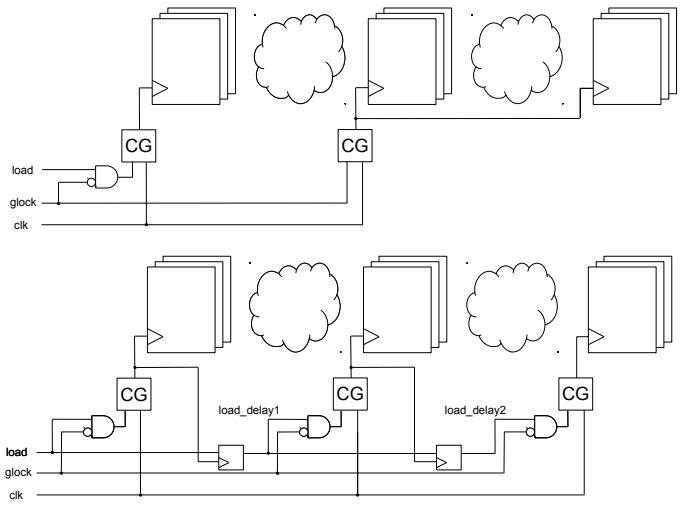

Fig. 6. Clock gating in a three-stage pipelined FU. Top: A straightforward pipeline. The load signal is only used to clock gate the input registers. Later stages are only gated by global lock. Bottom: Extra hardware propagates delayed load signals to later stages and uses them for clock gating.

V. Effects on Power and Area

For the synthesis and power analysis, switching activity information was produced with Modelsim which was used in Synopsys Design Compiler's topographical synthesis. The design was synthesized using a 28nm FDSOI process technology and 1 GHz target clock frequency. The described optimizations were synthesized and measured first individually, and finally all together to measure their total effect on power consumption.

The semiautomated IC network and register file optimization decreased the power usage of the design by 19.0% for SDR1 and 27.5% for SDR2. However, it caused the cycle counts of the benchmarks to increase by 13.3% and 19.5%, respectively. Thus, the optimization brought a saving of 8.3% for SDR1 and 13.4% for SDR2 in total energy. In addition, it was estimated that the reduced instruction size reduces the instruction memory read operation power consumption from 20.0 mW to 10.0 mW if we compare a cache scaled for an 256b instruction to one scaled to 128b. Before the optimization, the instruction width was 223b, which was padded to the next power of two, hence 256b. The area estimation for the instruction memory was reduced from 0.16 mm^2 to 0.07 mm^2. Estimates for instruction memory power usage were produced with CACTI [15]. The instruction memory was modeled with 1024 words at a temperature of 300K with no banking, using 32 nm technology node and ITRS-LSTP cells. Taking into account the increase in cycle times, we can estimate energy savings of 43.4% for SDR1 and 40.3% for SDR2 on instruction memory reads.

RF datapath gating decreased the total power usage of the core by 8.2% when running SDR1 and by 7.8% when running SDR2. In both cases, the gating logic caused an area overhead of 2.0% to the total core area. The power consumption of the RF blocks decreased by 10.5% and the IC by 18.5%. when running SDR1. For SDR2, the RF blocks power decreased by 5.5% and the IC by 15.7%. The implemented gating logic consumed 5.4% of total core power with SDR1 and 5.5% with SDR2.

RF banking decreased the total power consumption by 11.5% when running SDR1 and 8.4% when running SDR2, with 1.8% area overhead in both cases.

The floating point unit clock gating enhancement decreased the total power usage of the core by 10.4% when running SDR1 and 11.5% when running SDR2. The total area of the core decreased slightly by 0.7%.

Total power decrease with the optimizations combined was 24.8% when running SDR1 and 26.1% when running SDR2. In both cases, the optimizations caused an area overhead of 3.3%. The decrease of overall energy consumption of the design is directly proportional to the decrease in power consumption with the two programs used here, since the cycle counts and the clock frequency remained constant. The effects of optimizations are summarized in Fig. 7.

It is interesting to evaluate the exposed datapath design in comparison to a traditional operation triggered VLIW architecture to estimate the benefits of TTA-specific optimizations. This was conducted by first evaluating a TTA processor with similar datapath resources as those in the studied design, but with most of the TTA-unique optimizations turned off in the compiler, thus simulating limits where a VLIW programming model might get regarding the instruction cycles with similar function units and register files.

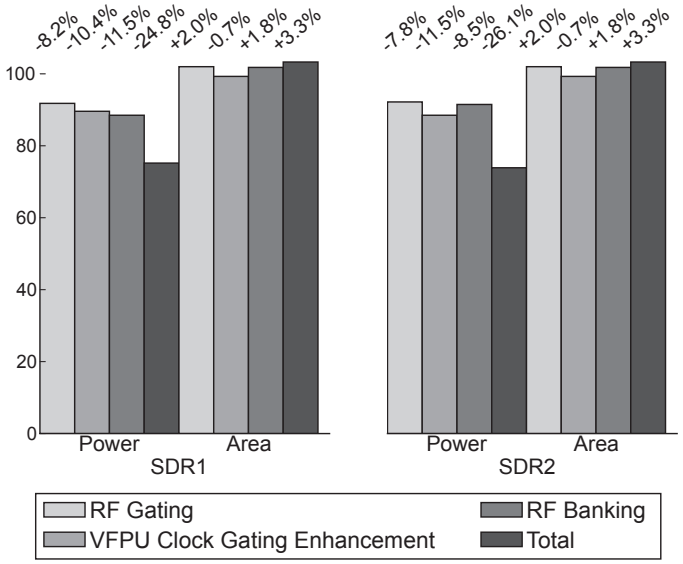

Fig. 7. Effect of power optimizations on test architecture. Numbers compared to the IC and RF optimized design as baseline. Synthesized with Design Compiler's power optimizations enabled.

Then, to estimate the resource savings, we iteratively added data buses and register files until the cycle counts achieved by the design were approximately the same as the ones of the studied TTA design. Power measurements after synthesis using register file datapath gating and the floating point unit clock gating enhancement were performed. These measurements showed that the TTA design with the TTA-unique optimizations enabled used 18% less power on average. This translates to 18% total energy reduction, since the execution time for both designs was matched. In the VLIW-like model, the instruction scheduler of the compiler backend still had the freedom of scheduling the timing of data transports, a feature not available in VLIW programming models.

VI. Conclusion

Power consumption is nowadays an important point of consideration in processor design, especially for portable devices. In this paper, three power optimizations on the RTL level were implemented for a customized high power performance SIMD TTA processor design case. In addition, the TTA interconnection network and register files were optimized by semiautomated design space exploration tools. The design was synthesized on 28 nm process technology and the achieved decrease in power consumption for the tailored SDR1 and SDR2 test programs were 24.8% and 26.1% respectively with no penalty in cycle count. In both cases, the optimizations brought an area increase of 3.3%.

The exposed datapath architecture allowed TTA-unique optimizations, operand sharing, software bypassing and dead result move elimination, which helped in reducing register file accesses and on top of the implemented optimizations helped to achieve on average approximately 18% energy saving in

comparison with a VLIW-like processor. Further power optimizations such as incorporating a loop buffer and investigating energy-efficient instruction encoding are ongoing.

ACKNOWLEDGMENT

This work was mainly funded by the Academy of Finland (funding decision 253087), Finnish Funding Agency for Technology and Innovation (project "Parallel Acceleration", funding decision 40115/13), and ARTEMIS JU under grant agreement no 621439 (ALMARVI).

REFERENCES

[1] H. Corporaal, *Microprocessor Architectures: From VLIW to TTA*. Chichester, England: John Wiley & Sons, Ltd., 1998.

[2] G. J. Lipovski, "Architecture of a simple, effective control processor," in *Proc. Symposium on Micro Architecture*, 1976, pp. 187–194.

[3] H. Corporaal and H. J. Mulder, "MOVE: A Framework for High-performance Processor Design," in *Proc. ACM/IEEE Conference on Supercomputing*, Albuquerque, NM, Nov. 18-22 1991, pp. 692–701.

[4] (2015) TCE: TTA-based Co-design Environment. [Online]. Available: http://tce.cs.tut.fi

[5] E. Donkoh, T. S. Ong, Y. N. Too, and P. Chiang, "Register file write data gating techniques and break-even analysis model," in *Proc. ACM/IEEE International Symposium on Low Power Electronics and Design*, Redondo Beach, CA, July 30-Aug. 1 2012, pp. 149–154.

[6] C.-L. Su, C.-Y. Tsui, and A. M. Despain, "Saving power in the control path of embedded processors," *IEEE Design and Test of Computers*, vol. 11, no. 4, pp. 24–30, Winter 1994.

[7] L. Benini, G. De Mecheli, E. Macii, M. Poncino, and S. Quer, "Power Optimization of Core-based Systems by Address Bus Encoding," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 6, no. 4, pp. 554–562, Dec. 1998.

[8] V. Guzma, T. Pitkänen, and J. Takala, "Effects of loop unrolling and use of instruction buffer on processor energy consumption," in *Proc. International Symposium on System on Chip (SoC)*, Tampere, Finland, Oct. 31-Nov. 2 2011, pp. 82–85.

[9] R. Balasubramonian, S. Dwarkadas, and D. Albonesi, "Reducing the complexity of the register file in dynamic superscalar processors," in *34th ACM/IEEE International Symposium on Microarchitecture*, Austin, TX, Dec. 1-5 2001, pp. 237–248.

[10] M. Tremblay and W. Joy, "Apparatus and method for optimizing die utilization and speed performance by register file splitting," Jan. 2002, US Patent 6,343,348.

[11] X. Guan and Y. Fei, "Reducing power consumption of embedded processors through register file partitioning and compiler support," in *Proc. International Conference on Application-Specific Systems, Architectures and Processors*, Leuven, Belgium, July 2-4 2008, pp. 269–274.

[12] T. Viitanen, H. Kultala, P. Jääskelainen, and J. Takala, "Heuristics for greedy transport triggered architecture interconnect exploration," in *Proc. International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, Uttar Pradesh, India, Oct. 12-17 2014, pp. 1–7.

[13] J. Hoogerbrugge and H. Corporaal, "Automatic synthesis of transport triggered processors," in *Proc. First Annual Conference in Advanced School for Computing and Imaging*, Heijen, The Netherlands, May 16-18 1995, pp. 1–11.

[14] Y. He, D. She, B. Mesman, and H. Corporaal, "MOVE-Pro: A low power and high code density TTA architecture," in *Proc. International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation*, Samos, Greece, July 18-21 2011, pp. 294–301.

[15] (2015) HP Labs : CACTI. [Online]. Available: http://www.hpl.hp.com/research/cacti/